ENSC-2013n°468

THÈSE DE DOCTORAT
DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Présentée par

Monsieur Benjamin MONMEGE

Pour obtenir le grade de

DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Domaine :
Informatique

Sujet de la thèse :

Spécification et Vérification de Propriétés Quantitatives : Expressions, Logiques et Automates

Thèse présentée et soutenue à Cachan le 24 octobre 2013 devant le jury composé de :

Benedikt BOLLIG	Chargé de recherche	Co-directeur de thèse
Olivier CARTON	Professeur	Examinateur
Manfred DROSTE	Professeur	Rapporteur
Paul GASTIN	Professeur	Co-directeur de thèse
Sylvain LOMBARDY	Professeur	Rapporteur
Jean-Marc TALBOT	Professeur	Examinateur
Jacques SAKAROVITCH	Professeur	Examinateur

Laboratoire Spécification et Vérification
École Normale Supérieure de Cachan, UMR 8643 du CNRS
61, avenue du Président Wilson
94235 CACHAN Cedex, France

English translation of the title: *Specification and Verification of Quantitative Properties: Expressions, Logics, and Automata.*

Abstract

Automatic verification has nowadays become a central domain of investigation in computer science. Over 25 years, a rich theory has been developed leading to numerous tools, both in academics and industry, allowing the verification of Boolean properties – those that can be either true or false. Current needs evolve to a finer analysis, a more quantitative one. Extension of verification techniques to quantitative domains has begun 15 years ago with probabilistic systems. However, many other quantitative properties are of interest, such as the lifespan of an equipment, energy consumption of an application, the reliability of a program, or the number of results matching a database query. Expressing these properties requires new specification languages, as well as algorithms checking these properties over a given structure. This thesis aims at investigating several formalisms, equipped with weights, able to specify such properties: denotational ones – like regular expressions, first-order logic with transitive closure, or temporal logics – or more operational ones, like navigating automata, possibly extended with pebbles.

A first objective of this thesis is to study expressiveness results comparing these formalisms. In particular, we give efficient translations from denotational formalisms to the operational one. These objects, and the associated results, are presented in a unified framework of graph structures. This permits to handle finite words and trees, nested words, pictures or Mazurkiewicz traces, as special cases. Therefore, possible applications are the verification of quantitative properties of traces of programs (possibly recursive, or concurrent), querying of XML documents (modeling databases for example), or natural language processing.

Second, we tackle some of the algorithmic questions that naturally arise in this context, like evaluation, satisfiability and model checking. In particular, we study some decidability and complexity results of these problems depending on the underlying semiring and the structures under consideration (words, trees...).

Finally, we consider some interesting restrictions of the previous formalisms. Some permit to extend the class of semirings on which we may specify quantitative properties. Another is dedicated to the special case of probabilistic specifications: in particular, we study syntactic fragments of our generic specification formalisms generating only probabilistic behaviors.

Résumé

La vérification automatique est aujourd'hui devenue un domaine central de recherche en informatique. Depuis plus de 25 ans, une riche théorie a été développée menant à de nombreux outils, à la fois académiques et industriels, permettant la vérification de propriétés booléennes – celles qui peuvent être soit vraies soit fausses. Les besoins actuels évoluent vers une analyse plus fine, c'est-à-dire plus quantitative. L'extension des techniques de vérification aux domaines quantitatifs a débuté depuis 15 ans avec les systèmes probabilistes. Cependant, de nombreuses autres propriétés quantitatives existent, telles que la durée de vie d'un équipement, la consommation énergétique d'une application, la fiabilité d'un programme, ou le nombre de résultats d'une requête dans une base de données. Exprimer ces propriétés requiert de nouveaux langages de spécification, ainsi que des algorithmes vérifiant ces propriétés sur une structure donnée. Cette thèse a pour objectif l'étude de plusieurs formalismes permettant de spécifier de telles propriétés, qu'ils soient dénotationnels – expressions régulières, logiques monadiques ou logiques temporelles – ou davantage opérationnels, comme des automates pondérés, éventuellement étendus avec des jetons.

Un premier objectif de ce manuscript est l'étude de résultats d'expressivité comparant ces formalismes. En particulier, on donne des traductions efficaces des formalismes dénotationnels vers celui opérationnel. Ces objets, ainsi que les résultats associés, sont présentés dans un cadre unifié de structures de graphes. Ils peuvent, entre autres, s'appliquer aux mots et arbres finis, aux mots emboîtés (*nested words*), aux images ou aux traces de Mazurkiewicz. Par conséquent, la vérification de propriétés quantitatives de traces de programmes (potentiellement récursifs, ou concurrents), les requêtes sur des documents XML (modélisant par exemple des bases de données), ou le traitement des langues naturelles sont des applications possibles.

On s'intéresse ensuite aux questions algorithmiques que soulèvent naturellement ces résultats, tels que l'évaluation, la satisfaction et le *model checking*. En particulier, on étudie la décidabilité et la complexité de certains de ces problèmes, en fonction du semi-anneau sous-jacent et des structures considérées (mots, arbres...).

Finalement, on considère des restrictions intéressantes des formalismes précédents. Certaines permettent d'étendre l'ensemble des semi-anneau sur lesquels on peut spécifier des propriétés quantitatives. Une autre est dédiée à l'étude du cas spécial de spécifications probabilistes : on étudie en particulier des fragments syntaxiques de nos formalismes génériques de spécification générant uniquement des comportements probabilistes.

Remerciements

J'ai eu la chance d'être dirigé par Paul Gastin et Benedikt Bollig durant mes trois années de doctorat. Ils ont toujours pris le temps de m'écouter et su trouver les mots justes pour me guider. Je les ai tout d'abord connus comme mes professeurs, que ce soit en première année de mon cursus à l'École Normale Supérieure de Cachan, ou lors de mon M2 au Master Parisien de Recherche en Informatique. Leurs talents pédagogiques à ces occasions – comme d'ailleurs tout au long de ces trois ans – couplés à leur grande générosité ont permis de construire une atmosphère stimulante bien que bon enfant, propice à la recherche que nous avons menée ensemble. Pour leur patience et leur savoir, leurs interrogations et leurs réponses, leurs encouragements et leur soutien inconditionnel, je voudrais les remercier chaleureusement.

Je remercie également les rapporteurs de ce manuscrit, Manfred Droste et Sylvain Lombardy, ainsi que les autres membres du jury, Olivier Carton, Jean-Marc Talbot et Jacques Sakarovitch, pour leurs commentaires, et le temps qu'ils m'ont consacré.

Durant mon stage de Master 2, j'ai également été co-encadré par Marc Zeitoun, avec qui j'ai pu continuer à travailler durant les trois dernières années. Un grand merci pour son soutien et son écoute pendant le stage, ainsi que pendant le doctorat.

Durant ces trois années, j'ai eu la chance de faire partie du Laboratoire Spécification et Vérification de l'École Normale Supérieure de Cachan. Je remercie tous ses membres qui ont fait de mon passage parmi eux un moment fantastique. Une pensée toute particulière pour mon ancien tuteur, Nicolas Markey, avec qui j'ai pu organiser le groupe de travail des axes Tempo et Mexico, Serge Haddad, que j'ai assisté pour un cours de probabilités, Sylvain Schmitz et Jean Goubault-Larrecq, pour les pauses déjeuner cruciverbistes, l'équipe administrative pour leur travail indispensable d'organisation, en particulier pour les voyages que j'ai pu réaliser... Je remercie également Michèle Sebag pour m'avoir fait découvrir le vaste monde de l'apprentissage statistique.

J'aimerais également adresser mes remerciements à Rupak Majumdar et Pierre Ganty qui m'ont encadré durant mon stage de Master 1, et Johan Montagnat qui m'a guidé pendant mon stage de Licence 3 : je leur dois mes premières expériences du monde de la recherche, et celles-ci furent des plus agréables et enrichissantes.

Une pensée toute particulière pour Peter Habermehl, Martin Leucker, Normann Decker et Daniel Thoma avec qui j'ai pu collaborer au sein du projet LeMon. Je remercie également Gilles Geeraerts et Thomas Brihaye pour m'avoir offert l'opportunité de rejoindre le projet Cassting et les équipes de l'Université Libre de Bruxelles et de l'Université de Mons pour mon post-doc.

Je ne peux résister à l'envie de remercier chaleureusement mes partenaires de bureau, Benoît Barbot, Aiswarya Cyriac, Assalé Adjé, Baptiste Gourdin, Ştefan Ciobâcă et plus généralement toutes les doctorantes et tous les doctorants du laboratoire pour l'ambiance chaleureuse qu'ils y font régner.

Un petit mot supplémentaire pour Aiswarya, ma sœur jumelle académique, avec qui j'ai passé d'incroyables moments de complicité et de chamailleries (et désolé Benoît si parfois tu as eu l'impression d'être coincé entre deux feux...). Nous avons aussi eu le plaisir de co-organiser le séminaire des doctorants, le bien nommé *Goûter des doctorants*, qui nous a permis d'apprendre et de découvrir énormément dans un moment de convivialité. Que notre amitié soit le socle de nos coopérations futures !

Je voudrais aussi remercier ici ma famille et mes amis pour leur soutien sans faille, leurs encouragements tout au long de mon cursus et leur amour. Vous êtes le support sur lequel je pourrai toujours venir me reposer. Finalement, tout mon amour et toute ma gratitude vont à Florian pour avoir été à mes côtés depuis toutes ces années, que tu as remplies non seulement de joie mais aussi des pâtisseries dignes des plus grands chefs !

Benjamin Monmege
Cachan, France
Octobre 2013

Contents

CHAPTER 1

Introduction

1.1 From Boolean to Quantitative Verification

Automatic verification is nowadays a central domain of research in computer science. Software supports systems in many critical applications: embedded and communication systems, Internet and e-commerce, health, finance, transport, energy, etc. Due to the economic and human cost of an error, it is important to develop tools to ensure a high degree of confidence in these applications. Formal methods provide an appropriate framework for achieving this goal: the development of the theory for over 25 years now provides many tools to verify properties of such systems, especially when the expected response is Boolean, namely whenever the property is either satisfied or is not.

The process of verification with formal methods usually follows a classical pattern in three steps.

1. **Modeling.** The system to be verified must be modeled in a formal way. Such a model can be finite or infinite depending on the modeling and can have a more or less rich structure: words, trees, or more complex graphs (such as Kripke structures). It could also be the case that the model generates such structures (e.g., a finite state automaton generating finite words), and that we want to check some properties of all the generated structures.
2. **Specification.** The property that the system should verify must be formally specified, i.e., written in a predetermined language. Regular expressions, temporal logics and first-order or monadic second-order logic are well-known high-level specification languages of properties over finite or infinite words. They can be generalized to deal with richer structures like trees for example. More specifically, properties of trees or graphs may be specified using some propositional dynamic logics or XPath and their extensions. Notice that we may also use some automata to specify properties.
3. **Algorithms.** The typical question consists in *model checking* the system against the specification, i.e., verify whether or not the model verifies the property (see [CGP99] for a general overview of model checking). This may be seen as an evaluation of the property over a given structure. Another typical question, usually called *satisfiability problem* (or emptiness problem), consists in checking whether the specified property admits at least one structure verifying it. The golden tool for answering these problems is to translate the high-level specification languages into a low-level one, namely *automata*, which are a powerful, yet tractable fragment of Turing machines (see [VW86] for the first application of automata techniques to verification).

Current verification needs are evolving towards a more detailed analysis, i.e., a *quantitative* one. The extension of the verification to quantitative fields began in the specific context of probabilistic systems, i.e., systems that may use randomization. In this mature field, theoretical methods are already transferred into usable tools like PRISM [KNP11]. Nevertheless, it is still an active area of research, see, e.g., [GO10, FGO12] with new (un)decidability results about probabilistic automata.

However, there are many other interesting quantitative properties outside the scope of probabilistic systems. Specifically, we may want to assess costs required to complete a task, the lifetime of a device, the energy requirements of an application, the reliability of a program, or the number of responses selected by a query in a database.

In this manuscript, the quantities only come from the specifications. In particular, we will consider the same models as in the Boolean setting, i.e., without any weights. This differs from other extensions like real-time verification (see, e.g., [AD94] introducing timed automata), where models can be seen as time words (every position of the word is associated to a time stamp being a positive real number), probabilistic verification (see, e.g., [BK08]) where models are Markov chains or Markov decision processes, or verification of systems dealing with unbounded data (see, e.g., [KF94] introducing finite-memory automata), where models are data words (every position of the word has both a label from a finite alphabet and a data from an infinite domain).

Henceforth, we consider a model, and a quantitative specification, and the model checking problem becomes a *model evaluation* problem, i.e., the computation of the quantity associated with the model by the specification. The classical model checking problem can then be seen as a special case where the computed quantity is indeed a Boolean value. However, the quantitative framework permits to address many more properties.

Counting Problems. A first common problem is to go from the verification of a property over a model to the computation of the total number of witnesses of this property in the model. As an example, a model could be an XML document representing a database, and a quantitative specification could ask the number of elements of the database selected by a given query: the quantity is in that case a natural number. Another interesting example could consist of a picture (a finite two-dimensional grid with a finite alphabet encoding the pixel colors) in which we may count the number of patterns of interest, e.g., counting the number of monochromatic squares in a picture.

Cost Optimization. Another interesting application lies in the optimization field. For example, considering a routing graph, we may ask for the length of the smallest path from a vertex to another. In terms of energy, considering a model being the execution trace of a sensor – with some actions requiring energy and some others representing an energy refill – we may ask for the minimal amount of energy needed at the beginning in order to perform the whole trace.

Probabilities. Even though our aim is to verify more general quantitative properties than just probabilities, we still can consider the probabilistic setting as a special case of our framework. For example, considering a graph as a finite arena, we may want to compute the probability that a uniform random walk goes from one point to another. Probabilities may also model reliability: e.g., it would be interesting to compute the reliability of a system, namely the probability that it never fails or runs out of energy.

Transducers. In the formal language community, quantities may be *languages*. Henceforth, a quantitative specification can map a given word to a language of words, the automaton view of it being commonly called transducers. Generalized to trees for example, this is the basis of some XML tree transducers, a fundamental tool for the database and natural language processing communities.

1.2 High-Level Specification Languages

Examples of high-level specification languages are regular expressions, or various kinds of logics, like monadic second order logic, first order logic, temporal logics or propositional dynamic logics. Whereas regular expressions are widely used for pattern matching in different areas, and lexical analysis in particular, temporal and propositional dynamic logics are broadly used in the formal verification field, and first or second order logic have a more theoretical interest. We give three examples below of extensions of classical specification languages to the quantitative framework: weighted regular expressions, probabilistic linear temporal logic and weighted monadic second order logic. In the three cases, the motto is always the same: *Maintaining a very similar syntax but evaluating a specification over a richer weight structure permits to specify some quantitative properties easily.*

1.2.1 Weighted Regular Expressions

Over finite words, Kleene first introduced in [Kle56] the theoretical bases for the study of regular languages, namely those being definable by a regular expression such as $a^\star ba^\star$ or $(a+b)^\star ca^\star + a^\star(bc)^\star$. Schützenberger extended in [Sch61] these bases to the quantitative setting considering some regular formal power series, i.e., regular functions mapping each word to a weight (that we may seen as a generalization of the previous case if the weight is a Boolean value). Keeping the same syntax as usual regular expressions – simply introducing some constant weight additionally – we may represent easily those regular formal power series. For example, if we want to map every word w over an alphabet A to the number of occurrences of a pattern p in it, we may simply consider the weighted regular expression $A^\star pA^\star$: if evaluated in the Boolean setting, it simply checks the occurrence of p as a factor of w, but evaluated over the set of natural numbers, it will output the number of such occurrences.

1.2.2 Probabilistic Linear Temporal Logic

For temporal logics, we may easily extend Linear Temporal Logic (LTL) with a probabilistic version. Usually, an LTL formula is constructed from atomic formulae from the finite alphabet A, namely a for every letter $a \in A$, on top of which we may apply temporal operators X (next), F (finally) and G (globally) being unary and U (until) being binary – if φ and ψ are formulae, $\mathsf{X}\,\varphi$, $\mathsf{F}\,\varphi$, $\mathsf{G}\,\varphi$ and $\varphi \;\mathsf{U}\; \psi$ are formulae too – and Boolean connectives – if φ and ψ are formulae $\neg\varphi$, $\varphi \vee \psi$ and $\varphi \wedge \psi$ are formulae too. The semantics is defined with respect to a word $w = a_0 a_1 \ldots a_{n-1} \in A^\star$ and a position i of this word ($0 \leq i \leq n-1$). We define $w, i \models \varphi$ by induction over the formula φ:

$w, i \models a$ if $a_i = a$

$w, i \models \mathsf{X}\,\varphi$ if $i < n-1$ and $w, i+1 \models \varphi$

$w, i \models \mathsf{F}\,\varphi$ if there exists $j \geq i$ such that $w, j \models \varphi$

$w, i \models \mathsf{G}\,\varphi$ if for all $j \geq i \quad w, j \models \varphi$

$w, i \models \varphi \;\mathsf{U}\; \psi$ if there exists $j \geq i$ such that $w, j \models \psi$ and for all $i \leq \ell < j \quad w, \ell \models \varphi$

$w, i \models \neg\varphi$ if not $w, i \models \varphi$

$w, i \models \varphi \vee \psi$ if $w, i \models \varphi$ or $w, i \models \psi$

$w, i \models \varphi \wedge \psi$ if $w, i \models \varphi$ and $w, i \models \psi$

For example, with the formula $\varphi = \mathsf{F}(a \wedge \mathsf{X}\, b)$, a word $w \in A^\star$ verifies $w, 0 \models \varphi$ if, and only if, w contains ab as a factor. We may search for a probabilistic version of this logic. Notice that, contrary to most of other works in this field (see, e.g., [BK08] for an overview of model checking), we still consider non-probabilistic models, here words. However, we enrich the specification, i.e., LTL formulae with some constant weights, here probabilities

p between 0 and 1. More precisely, we replace atomic formulae a and $\neg a$ by pa and $p(\neg a)$ with p a probability and a a letter of the alphabet. Now, each formula φ maps a pair (w, i) of word and position of this word to a probability, denoted by $[\![\varphi]\!](w, i)$, namely the probability that (w, i) verifies φ (the probabilities coming from the formula itself and not from the model). Rather than probability, one may consider this as a way to model a system receiving information from unreliable sensors. Naming a, b, c, ... the messages that may be received from the different sensors, the quantitative specification verifies a property of a given trace of execution of the system, associating to each message two different probabilities: the true-positive probability, i.e., the probability p that a message a appearing in the trace (checked by the LTL formula a) has really been sent by the sensor, and the true-negative probability, i.e., the probability p' that a message not appearing in the trace (checked by the LTL formula $\neg a$) has really not been sent by the sensor. Such a situation is modeled by the formula $pa \vee p'(\neg a)$. In particular, this permits to consider false-positive events (with probability $1 - p$), namely when a message seems to be received but has not really been sent, or false-negative events (with probability $1 - p'$), namely when a message is lost.

The semantics is defined inductively over the formulae: during the definition, we maintain as an invariant the fact that the sum of the semantics of a formula and its negation is always 1. Henceforth, we let

$$[\![\neg\varphi]\!](w, i) = 1 - [\![\varphi]\!](w, i)\,.$$

For the other Boolean connectives, we may simply define the semantics of the conjunction as a product (considering that the verification of each property is independant from the rest):

$$[\![\varphi \wedge \psi]\!](w, i) = [\![\varphi]\!](w, i) \times [\![\psi]\!](w, i)\,,$$

and adapt from the de Morgan law $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$, the semantics for the disjunction:

$$[\![\varphi\vee\psi]\!](w, i) = [\![\varphi]\!](w, i)+[\![\psi]\!](w, i)-[\![\varphi]\!](w, i)\times[\![\psi]\!](w, i) = [\![\varphi]\!](w, i)+[\![\neg\varphi]\!](w, i)\times[\![\psi]\!](w, i)\,.$$

For atomic formulae and operator X, we simply let

$$[\![pa]\!](w, i) = \begin{cases} p & \text{if } a_i = a \\ 0 & \text{otherwise,} \end{cases}$$

$$[\![\mathsf{X}\,\varphi]\!](w, i) = \begin{cases} [\![\varphi]\!](w, i+1) & \text{if } i < n-1 \\ 0 & \text{otherwise,} \end{cases}$$

In classical LTL, a formula $\mathsf{F}\,\varphi$ may be developped in the equivalent formula $\varphi \vee \mathsf{X}(\mathsf{F}\,\varphi)$: intuitively, it says that φ *finally* holds either when φ holds right now, or when $\mathsf{F}\,\varphi$ holds in the next position. In the probabilistic setting, we may translate this statement (considering the previous semantics for the disjunction and the negation) in

$$\begin{aligned}[\![\mathsf{F}\,\varphi]\!](w, i) &= [\![\varphi]\!](w, i) + [\![\mathsf{F}\,\varphi]\!](w, i+1) - [\![\varphi]\!](w, i) \times [\![\mathsf{F}\,\varphi]\!](w, i+1) \\ &= [\![\varphi]\!](w, i) + [\![\neg\varphi]\!](w, i) \times [\![\mathsf{F}\,\varphi]\!](w, i+1)\,.\end{aligned}$$

This equation can be applied iteratively to obtain the semantics of formula $\mathsf{F}\,\varphi$:

$$[\![\mathsf{F}\,\varphi]\!](w, i) = \sum_{j\geq i}\Big(\prod_{i\leq \ell<j}[\![\neg\varphi]\!](w, \ell)\Big) \times [\![\varphi]\!](w, j)\,.$$

We set the semantics of other temporal operators similarly, letting

$$[\![\mathsf{G}\,\varphi]\!](w, i) = \prod_{j\geq i}[\![\varphi]\!](w, j)$$

$$[\![\varphi \;\mathsf{U}\; \psi]\!](w, i) = \sum_{j\geq i}\Big(\prod_{i\leq \ell<j}[\![\neg\psi]\!](w, \ell) \times [\![\varphi]\!](w, \ell)\Big) \times [\![\psi]\!](w, j)\,.$$

For example, consider the word *abba*. The formula $\varphi = (\frac{2}{3}a \vee \frac{2}{3}(\neg a)) \mathsf{U} \frac{3}{4}a$, that we can abusively write $\frac{2}{3} \mathsf{U} \frac{3}{4}a$ since the left term is always interpreted as the constant probability $\frac{2}{3}$, interprets on this word as

$$[\![\varphi]\!](abba, 0) = \frac{3}{4} + 0 + 0 + \left(\frac{1}{4} \times \frac{2}{3}\right) \times \left(1 \times \frac{2}{3}\right) \times \left(1 \times \frac{2}{3}\right) \times \frac{3}{4}\,.$$

The intuition is that the left term of the *Until* operator models a *discounting* factor, modeling the fact that a sooner witness is preferable. However, in order to keep a probabilistic interpretation of the result, the first witness a, apart from counting to three fourth of the result, also *decreases* the power of the second witness with an initial factor $\frac{1}{4} = 1 - \frac{3}{4}$. Notice also that the right term of the *Until* operator does not give a true-negative probability, that we may interpret as a probability 0, which results in the two zeros in the semantics.

The *Finally* operator can then be seen as a special *Until* operation without discount: indeed, as for the classical LTL, we have the following semantical equivalence $\mathsf{F}\,\varphi = \top \mathsf{U} \varphi$ where $\top$ is an atomic formula always evaluating to 1 (that we can write $1a \vee 1(\neg a)$ for example). Then, formula $\varphi = \mathsf{F}(\frac{2}{3}a)$ interprets over the word *abb* as

$$[\![\varphi]\!](abb, 0) = \frac{2}{3} + 0 + 0$$

as well as over the word *bba*:

$$[\![\varphi]\!](bba, 0) = 0 + 0 + 1 \times 1 \times \frac{2}{3}$$

No discount factor is taken into account in the *Finally* operator.

1.2.3 Weighted Monadic Second Order Logic

Apart from the previous probabilistic example, we may notice that this semantics has been defined by the dual use of *sums* and *products* of probabilities. The same duality is used to define the semantics of weighted regular expressions (one may find a formal definition in [Sak09], or later in Chapter 3). Indeed, in many works about quantitative specification, such a duality is common: the weight structure is equipped with an algebraic structure of *semiring*. It permits to treat in a unified and elegant way many quantitative settings, as for example, real numbers with the usual addition and multiplication (used for counting or probabilities), natural numbers equipped with a minimum operation and the addition (used for optimization) or languages of words with the union and the concatenation (used in transducers for example).

Recently, Droste and Gastin introduced in [DG07] a weighted monadic second order logic over words. They also defined a fragment of this latter which captures exactly the regular formal power series. The main idea is again to keep a similar syntax to the classical monadic second order logic – simply adding constant weights as atomic formulae – but to give a quantitative semantics to every formula. This is done using semirings, by interpreting (as for probabilistic LTL) disjunctions as sums, conjunctions as products and by extension, existential and universal quantifications as addition and multiplication over positions or sets of positions of the word.

However, Droste and Gastin showed that, in order for the series denoted by a formula to remain regular, it is necessary to restrict the use of universal quantification, leading to the first logical characterization of regular formal power series.

Notice that, apart from words, regular expressions and various logics have been intensively extended to more complex models like ranked trees, nested words, pictures, or infinite models. In particular, the logic defined in [DG07] has been further studied over many models: ranked and unranked trees [DV06, DV09], infinite words [DR07, DM10a], infinite trees [Rah07], nested words [Mat10, DP12], or pictures [Fic11].

The first aim of this manuscript is to describe and study more general and expressive high-level specification formalisms of quantitative properties. Rather than only considering words, or trees, we study a general framework of graphs, having as special cases all previously mentioned finite models.

1.3 The Success Story of Automata

It is usually difficult to directly reason about, or verify, high-level specifications. It is well-known, for instance, that the best method to parse a regular expression is to use *automata* (see, e.g., [AU72] or [FHW10] with an original and recent presentation in both the classical and the weighted settings). Also, checking a model against a logical specification is usually achieved efficiently by first translating the specification into an equivalent automaton, as introduced initially in [VW86].

In the classical setting, Kleene proves in [Kle56] the equivalence between languages definable with regular expressions – namely the regular languages – and those recognized by finite-state automata. A further equivalence between monadic second order logic and finite-state automata, usually known as the Büchi-Elgot-Trakhtenbrot theorem, follows from [Büc59, Elg61, Tra61].

In the weighted setting, Schützenberger also introduced *weighted automata* in [Sch61], a quantitative extension of finite-state automata (see [DKV09, BR10, Sak09] for recent books about extension of automata theory to the weighted setting among others). The computation result of a weighted automaton on a word is not a Boolean value distinguishing accepted behaviors from those rejected. Instead, a computation over a word produces a weight in a semiring. The weight of a word is the *sum* of the weights of the different possible computations over the word, each calculated as the *product* of the weight of the visited transitions. In [Sch61], Schützenberger already presented an extension of Kleene theorem, i.e., an equivalence between regular formal power series (that are definable by weighted regular expressions for example), and formal power series recognized by weighted automata. This seminal result is still a central object of research, either to make efficient translations from weighted regular expressions to weighted automata [CF03, AM06] and implement them [FHW10], or to extend it to other structures, like infinite words for example [DM10b].

Surprisingly, the logical equivalence generalizing the result of the Boolean case only appeared recently, in terms of the logic introduced in [DG07]. As previously mentioned, this is not the full weighted monadic second order logic, but rather a restricted fragment of it. In particular, this fragment does not contain entirely the weighted first order logic, by lack of power of weighted automata. The previously mentioned articles introducing similar fragments over richer models, like trees or infinite models, also prove an equivalence between the logic and weighted automata over the subsequent models. For example, in [DPV04], authors prove the equivalence between a fragment of weighted monadic second order logic over finite ranked trees, and a weighted extension of tree automata.

A second aim of this manuscript is to introduce a more general class of weighted automata, at least able to compute all the formal power series definable by weighted first order formulae.

Rather than considering *branching* automata (like bottom-up tree automata [CLDG^{+}08] or graph automata [Tho91]), we focus on *navigating* automata. Over words for instance, these automata may be seen as a weighted generalization of two-way automata (first introduced by Rabin and Scott in [RS59]). An advantage of these navigating automata is their possible direct generalization to any class of graphs. These automata are still not general enough to recognize all formal power series defined by weighted first order formulae. However, following ideas of [EH99], it is possible to add *pebbles* to navigating automata in order to recognize more behaviors: those pebbles permit to mark temporarily the model with some tokens. Tree-walking automata, and pebble tree-walking automata, have been extensively studied recently [BSSS06, SS07, Boj08]. Such automata models have attracted increasing interest, particularly in the context of queries manipulating XML documents and XPath (XML Path Language) queries [tCS10]. In [EH07], considering ranked trees (and also general classes of graphs), authors prove, in the Boolean setting, that pebble navigating automata are expressively equivalent to the first order logic enriched with a transitive closure operator.

One of the main contributions of this manuscript is to define weighted pebble navigating automata, and to introduce a weighted transitive closure operator, to prove some equivalence results between formal power series recognized by such automata and formulae of weighted first order logic with a weighted transitive closure operator.

A weighted extension of tree-walking automata has been introduced and studied in [FM09]. However, the authors only consider a non-looping version of these automata, and do not consider the extension with pebbles.

It has to be noticed that some other results of equivalence between quantitative generalizations of automata and quantitative logics have also been obtained by other authors, with different purposes. Quantitive properties of infinite words have been considered in [Col09], for example, by the mean of *regular cost functions*. These may be defined in terms of *B-* or *S-automata*, and have equivalent logical and algebraic characterizations. More recently, [ADD$^+$13] investigates tree transducers as a way to define quantitative functions over finite words: the model, called *cost register automata*, defines some *regular functions*, also with an equivalent logical formalism based on monadic second order logic.

1.4 Algorithms

Translating high-level specification formalisms into automata is interesting because this enables efficient algorithms, e.g., with respect to model checking (or evaluation) and emptiness checking. For example, some classical automata-based techniques to verify systems have been extended in [CCH$^+$05] to the weighted setting: authors use *bound functions* in order to transfer classical methods and results. Algorithms over weighted automata (determinization, minimization, etc.) have also been extensively studied since the 60s, see [Moh09] for an overview. For example, the minimization of a weighted automaton (over finite words) over a field can be done in polynomial time (as already shown in the seminal paper of Schützenberger [Sch61]): as an interesting implication for the Boolean setting, this implies that the equivalence of *unambiguous* finite-state automata can be performed in polynomial time, and not only in polynomial space as it is the case for general non-deterministic finite-state automata. All these efficient algorithms have already permitted to apply the techniques to several areas such as natural language processing [KM09] and language recognition, or the compression of digital images (see [DKV09, Part IV]).

In this manuscript, we will consider the evaluation and emptiness problems of our newly introduced class of pebble weighted automata. In particular, we give a precise complexity for the evaluation of pebble weighted automaton, in terms of the size of the model (usually huge), and the automaton (usually more reasonable).

Notice that other authors design algorithms for weighted automata, in other contexts. For example, in [CDH08], authors study quantitative languages of infinite words: rather than using a semiring, they consider more complex product operations in order to model discounting or averaging. This has been further studied with alternating automata in [CDH09], and also [AK11] over finite words. They also consider some *mean-payoff expressions* using automata as atomic expressions in [CDE$^+$10], in order to specify more complex quantitative properties.

1.5 Outline

This manuscript aims at studying different formalisms describing quantitative properties of graph structures. We start in **Chapter 2** by introducing the graphs we study, and the weight domains we use.

We introduce *hybrid weighted expressions* in **Chapter 3**, namely a generalization of weighted regular expressions over general classes of graphs. They may also use some variables in order to generate more behaviors. As usual, we give a denotational (and inductive) semantics of these expressions. As a preliminary for subsequent proofs, we also give a more algebraic semantics of them, in terms of *marked graphs.*

Chapter 4 is devoted to the introduction of our class of *pebble weighted automata*. We start by the special case of weighted navigating automata (without pebbles) and prove, in this case as well as in the more general one, a Kleene-Schützenberger theorem stating the equivalence with weighted expressions and hybrid weighted expressions respectively. The translations from expressions to automata are done carefully in order to generate small automata.

In **Chapter 5**, we explain how to evaluate a pebble weighted automaton over a given model. After giving a very general evaluation algorithm, we consider some special cases (words, trees and nested words) and give more efficient algorithms.

Logical specifications are studied in **Chapter 6**. We first recall monadic second order logics over graphs, permitting to prove some decidability results for the emptiness of a pebble weighted automaton. Afterwards, we introduce our class of *weighted first order logic with transitive closure.* The main result of this chapter is to prove the expressive equivalence between this logic and pebble weighted automata. The translation from logics to automata is made efficiently (as for hybrid weighted expressions), in particular by a careful use of the pebbles: this translation requires to consider *searchable* classes of graphs, i.e., graphs in which a walking automaton can enumerate every vertex faithfully. Conversely, the translation from automata to logic – with a more theoretical interest of completeness and robustness of the class of automata – is also shown: it requires to consider the new concept of *zonable* classes of graphs, namely graphs that we can partition into zones expressible into the logic. Finally, hybrid navigational logics are considered at the end of this chapter: they generalize the classical propositional dynamic logic and temporal logics to capture the power of pebble weighted automata.

Chapter 7 focuses on some natural restrictions of the specification languages considered before. In particular, these restrictions permit to extend the set of weights studied, and consider for example the set of positive and negative real numbers (forbidden in the rest of this manuscript for reasons that we present in Chapter 3). The restrictions are of two sorts: one, semantical, restricts the set of runs that automata may follow to non-looping ones, whereas the second, syntactical, directly restricts the possibility of the automata to only one-way runs. We show that both restrictions indeed coincide over finite words. We also prove that the emptiness problem quickly becomes undecidable in this restricted case, in particular because we allow more general semirings.

We consider the probabilistic case as a special case of the weighted setting in **Chapter 8**. The aim is to design a fragment of hybrid weighted expressions and pebble weighted automata generating only probabilistic behaviors of graphs. After presenting the restrictions, we again prove a Kleene-Schützenberger theorem relating these two formalisms. As a special case, this permits to give a set of regular expressions (with probabilities) that define exactly the behaviors recognized by Rabin probabilistic automata [Rab63]. If we forget about the probabilities, this also gives an insight of some deterministic restrictions of the model of automata we study in this manuscript.

Finally, we present in **Chapter 9** an implementation of the algorithm of evaluation of a hybrid weighted expression in the case of words.

References

The results presented in this manuscript are extensions of works published in articles co-authored with Benedikt Bollig, Paul Gastin and Marc Zeitoun. These articles only explored the case of finite words and nested words, and this manuscript presents the extension to more general models of graphs. The following articles have been presented at international peer-reviewed conferences:

[1] Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Pebble weighted automata and transitive closure logics. In Samson Abramsky, Friedhelm Meyer auf der Heide, and Paul Spirakis, editors, *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP'10)*, Lecture Notes in Computer Science, pages 587–598. Springer, Bordeaux, France, 2010.

[2] Paul Gastin and Benjamin Monmege. Adding Pebbles to Weighted Automata. In Nelma Moreira and Rogério Reis, editors, *Proceedings of the 17th International Conference on Implementation and Application of Automata (CIAA'12)*, Lecture Notes in Computer Science, pages 28–51. Springer, Porto, Portugal, 2012.

[3] Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. A Probabilistic Kleene Theorem. In Madhavan Mukund and Supratik Chakraborty, editors, *Proceedings of the 10th International Symposium on Automated Technology for Verification and Analysis (ATVA'12)*, Lecture Notes in Computer Science, pages 400–415. Springer, Thiruvananthapuram, India, 2012.

[4] Benedikt Bollig, Paul Gastin, and Benjamin Monmege. Weighted Specifications over Nested Words. In Frank Pfenning, editor, *Proceedings of the 16th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'13)*, volume 7794 of *Lecture Notes in Computer Science*, pages 385–400. Springer, 2013.

The following articles have also been submitted to international journals:

[5] Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Pebble weighted automata and transitive closure logics. Submitted to *Transactions on Computational Logic*. February 2012.

[6] Paul Gastin and Benjamin Monmege. Adding Pebbles to Weighted Automata: Easy Specification & Efficient Evaluation. Accepted to *Theoretical Computer Science: Special Issue of the 6th International Workshop Weighted Automata, Theory and Applications*. August 2012.

During my three years of PhD, I have also collaborated with other authors on topics not directly related with the work presented in this manuscript. The following additional articles have hence been published during my PhD studies:

[7] Pierre Ganty, Rupak Majumdar, and Benjamin Monmege. Bounded Underapproximations. *Formal Methods in System Design*, 40(2):206–231, 2012.

[8] Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A Fresh Approach to Learning Register Automata. In Marie-Pierre Béal and Olivier Carton, editors, *Proceedings of the 17th International Conference on Developments in Lan-*

guage Theory (DLT'13), volume 7907 of *Lecture Notes in Computer Science*. Springer, Marne-la-Vallée, France, 2013.

CHAPTER **2**

Preliminaries

General Notations

$\lvert E\rvert$	the cardinality of a finite set E
$\mathfrak{P}(E)$	the powerset of a set E
$[a,b]$	the closed real interval $\{x \in \mathbb{R} \mid a \leq x \leq b\}$
$[a \mathinner{..} b]$	the closed integer interval $\{i \in \mathbb{Z} \mid a \leq i \leq b\}$
$A \to B$	mapping (or function) from A to B
$A \rightharpoonup B$	partial mapping (or function) from A to B
$f_{\mid A}$	restriction of a function $f\colon B \to C$ to subset A of B

We will use the same notation $(x_i)_{i \in I}$ for families and sequences of elements. The main difference lies in the index set I: it is considered as an ordered set in the case of a sequence, but families are not ordered.

The domain of a partial mapping $f\colon A \rightharpoonup B$ is the set of elements $a \in A$ for which image $f(a)$ is well-defined: it is denoted by $\mathrm{dom}(f)$ in the following. Substitutions (also called valuations in the following) are partial mappings from a (possibly infinite) set Var of variables to a set A. The substitution of domain $\{x\}$ mapping x to value $a \in A$ is denoted by $[x \mapsto a]$. Moreover, if σ is a substitution of domain D, $\sigma[x \mapsto a]$ is the substitution σ' of domain $D \cup \{x\}$ such that $\sigma'(x) = a$ and $\sigma'(y) = \sigma(y)$ for every $y \in D$ different from x.

2.1 Graph Structures

In order to introduce the definitions and results of this manuscript in a unified way for different kinds of structures like words, trees, nested words, etc. we will use graphs in the sequel. Among other works, this unified presentation of our framework has been inspired by [Tho91, EH07, Fic07]. Graphs we will consider are graphs of bounded degree with labels on vertices and edges.

We fix a finite alphabet A and another finite set D that will be used for encoding the *directions* of the graph. This set is always supposed to be *symmetrical* in the following meaning: there exists a bijective mapping $\cdot^{-1}\colon D \to D$ such that for every direction $d \in D$, $(d^{-1})^{-1} = d$. For example $\{\to, \leftarrow\}$ with $\to^{-1} = \leftarrow$ and $\leftarrow^{-1} = \to$ is a valid set of directions. We extend the mapping $\cdot^{-1}$ to sets of directions: if $D' \subseteq D$, $D'^{-1} = \{d^{-1} \mid d \in D'\}$.

Definition 2.1. An (A, D)-*graph* (or simply a graph, if A and D are clear from the context) is a tuple $G = (V, (E_d)_{d\in D}, \lambda)$ such that

- V is a non-empty and finite set of vertices;
- for all $d \in D$, $E_d \subseteq V \times V$ is a *functional irreflexive relation* describing the d-edges of the graph, i.e., for all $v \in V$, there exists at most a vertex $v' \in V$ such that $(v, v') \in E_d$, and then it verifies $v' \neq v$;
- for all $d \in D$, d^{-1}-edges are the reversed d-edges, i.e.,

$$E_{d^{-1}} = \{(v, v') \mid (v', v) \in E_d\}\,;$$

- $\lambda\colon V \to A$ is a labeling of vertices with letters of the alphabet.

In the following, we denote $E = \bigcup_{d\in D} E_d$ the full set of edges of a graph. ■

A *path* in such a graph is a finite non-empty sequence $(v_k)_{0\leq k\leq n}$ of vertices of the graph such that successive vertices are linked by an edge, i.e., for every $0 \leq k \leq n-1$, $(v_k, v_{k+1}) \in E$. Such a path is said to be a path from v_0 to v_n, and its *length* is n, i.e., the number of edges it contains. **Throughout this manuscript, we restrict ourselves to connected graphs**, i.e., such that for every pair of vertices, there always exists a path from the first one to the second one. For a subset D' of D, we say that a path is a D'-path if every edge it contains is labeled by a direction of D': with previous notations, for every $0 \leq k \leq n-1$, $(v_k, v_{k+1}) \in \bigcup_{d\in D'} E_d$. A $\{d\}$-path is also called a d-path for conciseness. Finally, a D'-*cycle* is a non-empty D-path from a vertex to itself.

We may equip every graph $G = (V, (E_d)_{d\in D}, \lambda)$ with a *distance* function $\mathsf{d}\colon V \times V \to \mathbb{N}$ with $\mathsf{d}(v, v')$ being the length of the shortest path from v to v'. In particular, it verifies the following properties:

- $\mathsf{d}(v, v') = 0$ if, and only if, $v = v'$, for every vertices $v, v' \in V$;
- $\mathsf{d}(v, v') = \mathsf{d}(v', v)$ for every vertices $v, v' \in V$;
- $\mathsf{d}(v, v'') \leq \mathsf{d}(v, v') + \mathsf{d}(v', v'')$, for every vertices $v, v', v'' \in V$.

Vertices of a graph come with a type. In a graph G, the *type* of vertex v, denoted $\mathsf{type}(v)$, is the set of all possible directions available from v, namely

$$\mathsf{type}(v) = \{d \in D \mid \exists v' \in V \quad (v, v') \in E_d\}$$

A *pointed graph* is a tuple $G = (V, (E_d)_{d\in D}, \lambda, v^{(i)}, v^{(f)})$ with $(V, (E_d)_{d\in D}, \lambda, \chi)$ a graph and $v^{(i)}$, $v^{(f)}$ some vertices of V, used as initial and final vertices. The set of pointed graphs with alphabet A and directions D is denoted by $\mathcal{G}(A, D)$.

Throughout this manuscript, if not specified, a graph $G \in \mathcal{G}(A, D)$ will be supposed to have set of vertices denoted V, sets of edges $(E_d)_{d\in D}$, etc.

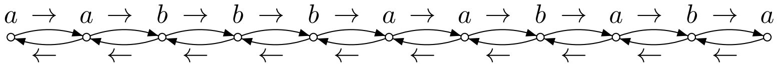


Figure 2.1: Graph representation of a word

2.1.1 Words

A *word* is a finite sequence $w = a_0 a_1 \cdots a_{n-1}$ of letters $a_i \in A$. The empty word is denoted by ε. The set of words over the alphabet A is denoted by $A^\star$, whereas the set of non-empty words is denoted by A^+. The *length* of the word w is the number n of letters it contains, and is often denoted by $|w|$. We also denote by $|w|_a$ the number of occurrences of letter a in word w. Words may be concatenated: the *concatenation* of two words w_1 and w_2, denoted $w_1 \cdot w_2$ (or shortly $w_1 w_2$), is the word obtained by appending the letters of w_2 after the letters of w_1.

We model the set of words in our graph framework by considering as set of directions $D = \{\rightarrow, \leftarrow\}$, with $\rightarrow^{-1} = \leftarrow$. Notice that the set $\mathcal{G}(A, D)$ then *contains* all the words of A^+. However, it also contains graphs that do not represent words (e.g., cycles), and hence we only consider the subset of graphs that represents finite words. More precisely, we let $\mathcal{W}\text{ord}(A)$ be the set of pointed graphs $([0 \mathinner{..} n-1], (E_\rightarrow, E_\leftarrow), \lambda, 0, n-1)$ with $n \geq 1$, such that $E_\rightarrow = \{(i, i+1) \mid i \in [0 \mathinner{..} n-2]\}$ and $E_\leftarrow = \{(i, i-1) \mid i \in [1 \mathinner{..} n-1]\}$. For these graphs, the initial vertex is the first position of the word, whereas the final vertex is the last one. The graph representation of the word *aabbbaababa* is depicted in Figure 2.1: this word has length $n = 11$.

2.1.2 Ranked Trees

A *ranked alphabet* is an alphabet A where each letter $a \in A$ comes with an arity denoted by $\text{ar}(a) \in \mathbb{N}$. A ranked tree is then a finite term labeled by letters of a ranked alphabet. More formally, a *ranked tree* is a partial mapping[1] $t \colon \mathbb{N}^\star \rightharpoonup A$ with domain $\text{dom}(t)$ satisfying the following properties

- $\text{dom}(t)$ is finite and nonempty;
- $\text{dom}(t)$ is prefix closed, i.e., if a sequence is in $\text{dom}(t)$, each of its prefixes is also in $\text{dom}(t)$;
- for all $u \in \text{dom}(t)$, if $\text{ar}(t(u)) = n \geq 0$, then $\{i \in \mathbb{N} \mid ui \in \text{dom}(t)\} = [0 \mathinner{..} n-1]$.

Notice that in the case $n = 0$, the interval $[0 \mathinner{..} n-1]$ is empty. Elements of the domain $\text{dom}(t)$ are called *nodes* of the tree. Each node $u \in \text{dom}(t)$ inherits an arity $\text{ar}(t(u))$ from its labeling. Nodes of arity 0 are called *leaves*, whereas nodes of non-null arity are called *internal nodes*. The root of t is the node ε. A node $ui \in \text{dom}(t)$ is said to be a *child* of node u, sometimes called the ith child of u, whereas u is the *parent* of node ui. We denote by $|t|$ the cardinality of the domain $\text{dom}(t)$ of t, i.e., its number of nodes.

We may represent these trees in our graph framework by considering the set of directions $D = \{\uparrow_i, \downarrow_i \mid 0 \leq i < \text{ar}(a), a \in A\}$, with $\uparrow_i^{-1} = \downarrow_i$. The set $\mathcal{G}(A, D)$ then *contains* all the ranked trees over alphabet A. Indeed this set of graphs contains also graphs which are not well-formed, e.g., because a vertex admits as type $\{\downarrow_2\}$, or because the arity of the symbol labeling a vertex is not correct. Hence, we let $\mathcal{T}\text{ree}(A)$ be the subset of $\mathcal{G}(A, D)$ containing only well-formed ranked trees. More formally, $\mathcal{T}\text{ree}(A)$ is the set of pointed graphs $(\text{dom}(t), (E_d)_{d \in D} \lambda, \varepsilon, \varepsilon)$ where t is a ranked tree as defined previously, such that $\lambda(u) = t(u)$ for every vertex $u \in \text{dom}(t)$, and for every $i \in [0 \mathinner{..} \max_{a \in A} \text{ar}(a) - 1]$, $E_{\downarrow_i} = \{(u, ui) \mid u, ui \in \text{dom}(t)\}$ and $E_{\uparrow_i} = \{(ui, u) \mid u, ui \in \text{dom}(t)\}$. Notice that in these graphs, the initial and final vertices are the same, namely the root of the tree. An example of ranked

[1] Here, $\mathbb{N}^\star$ denotes the set of words over the infinite alphabet $\mathbb{N}$.

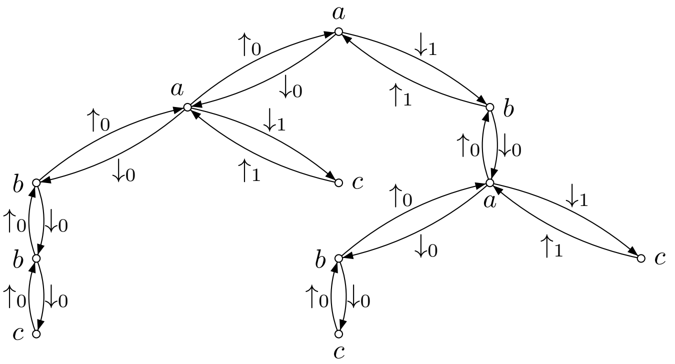


Figure 2.2: A ranked tree

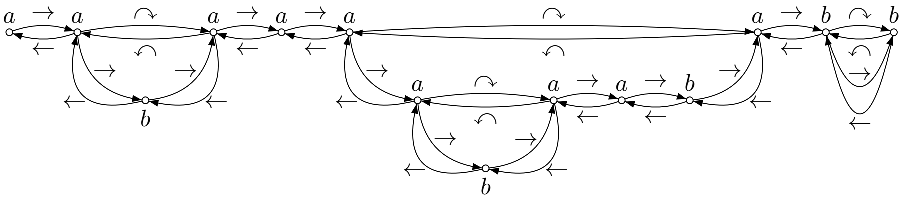


Figure 2.3: A nested word

tree over the alphabet $A = \{a, b, c\}$ with a of arity 2, b of arity 1 and c of arity 0, is depicted in Figure 2.2: in this tree, the rightmost leaf is encoded by the sequence 101.

2.1.3 Nested Words

Nested words are structures particularly useful to denote XML documents or finite executions of recursive programs. Indeed, these structures have in common to contain an underlying linear order over their positions, and additional informations. In well-formed XML documents, an opening tag (e.g., `<ul>` or `<h1>` for HTML documents) is linked to its matched closing tag (`</ul>` and `</h1>`). In the same way, for recursive programs, positions encoding a call of a program is linked to its matched return.

Formally, we will describe the set of nested words over an alphabet A as a subset of the pointed graphs $\mathcal{G}(A, D)$ with $D = \{\rightarrow, \leftarrow, \curvearrowright, \curvearrowleft\}$ with $\rightarrow^{-1} = \leftarrow$ and $\curvearrowright^{-1} = \curvearrowleft$. Whereas $\rightarrow$ and $\leftarrow$ are used, as for words, to denote the successor and predecessor edges in the linear order, directions $\curvearrowright$ and $\curvearrowleft$ encode the links in the nested words (opening/closing tags, call/matching return). We let $\mathcal{N}\text{est}(A)$ the subset of $\mathcal{G}(A, D)$ containing all pointed graphs $([0 \mathinner{..} n-1], (E_{\rightarrow}, E_{\leftarrow}, E_{\curvearrowright}, E_{\curvearrowleft}), \lambda, 0, n-1)$ with $n \geq 1$, $E_{\rightarrow} = \{(i, i+1) \mid i \in [0 \mathinner{..} n-2]\}$, and $E_{\curvearrowright} \subset \{(i, j) \mid i < j\}$ the set of nesting edges verifying for every $(i, j), (i', j') \in E_{\curvearrowright}$ that

- nesting edges do not share positions: $i = i'$ if, and only if, $j = j'$;
- nesting edges do not cross: if $i < i'$ then either $j < i'$ or $j > j'$).

An example of nested word is depicted in Figure 2.3. Its set of vertices is $[0 \mathinner{..} 13]$, depicted in increasing order, and has as $\curvearrowright$-edges

$$E_{\curvearrowright} = \{(1, 3), (5, 11), (6, 8), (12, 13)\}\,.$$

On the picture, we have structured vertically the vertices with respect to their *call-depth*. By definition, the call-depth of a vertex in a nested word is the number of nesting edges jumping over this vertex: more formally, the call-depth of a vertex v is $|\{(v_1, v_2) \in E_{\curvearrowright} \mid$

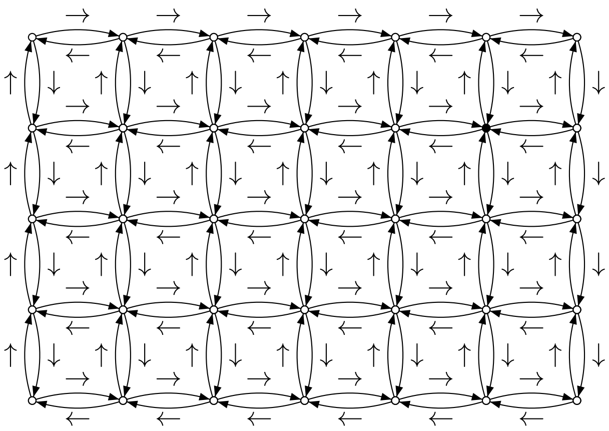


Figure 2.4: A picture

$v_1 < v < v_2\}|$. For example, vertex 4 and 5 have call-depth 0, vertex 9 has call-depth 1 and vertex 7 is the only vertex of call-depth 2.

2.1.4 Grids and Pictures

Finite grids are another interesting example of graphs. As noticed by [BH67], pictures may be seen as finite two-dimensional grids. We may model those by graphs with sets of directions $D = \{\rightarrow, \leftarrow, \downarrow, \uparrow\}$ (with $\rightarrow^{-1} = \leftarrow$ and $\downarrow^{-1} = \uparrow$), labeled by letters in an alphabet A encoding gray levels or colors of the pixels. Let $\mathcal{P}\mathrm{ict}(A)$ be the subset of $\mathcal{G}(A, D)$ containing all pointed graphs $([0 \mathinner{..} n_1 - 1] \times [0 \mathinner{..} n_2 - 1], (E_\rightarrow, E_\leftarrow, E_\downarrow, E_\uparrow), \lambda, (0,0), (n_1 - 1, n_2 - 1))$ with $n_1 \geq 1$ being the width of the picture, $n_2 \geq 1$ its height, and with edges

$$\begin{aligned}
E_\rightarrow &= \{((i,j),(i+1,j)) \mid i \in [0 \mathinner{..} n_1 - 2], j \in [0 \mathinner{..} n_2 - 1]\} \\
E_\leftarrow &= \{((i,j),(i-1,j)) \mid i \in [1 \mathinner{..} n_1 - 1], j \in [0 \mathinner{..} n_2 - 1]\} \\
E_\downarrow &= \{((i,j),(i,j+1)) \mid i \in [0 \mathinner{..} n_1 - 1], j \in [0 \mathinner{..} n_2 - 2]\} \\
E_\uparrow &= \{((i,j),(i,j-1)) \mid i \in [0 \mathinner{..} n_1 - 1], j \in [1 \mathinner{..} n_2 - 1]\}
\end{aligned}$$

An example of picture with pixels either white or black is represented in Figure 2.4. Its width is 7 and its height is 5. For example, the only black pixel is the vertex indexed $(5,1)$ of the graph.

2.1.5 Mazurkiewicz Traces

Another domain of application is the study of distributed systems, and, for example, concurrent programs with a static set of processes that may synchronize through rendez-vous. In this setting, Mazurkiewicz traces have been acknowledged to be a very useful and succinct model for sets of communications of such programs.

We now define Mazurkiewicz in terms of their graph representations, following ideas of [DR95]. We fix a finite set of n processes, that we index by integers in $[1 \mathinner{..} n]$. Vertices of the graph represent actions executed by the processes: hence, we fix a finite alphabet A representing internal actions and rendez-vous of the program. More precisely, we suppose fixed a mapping $\mathrm{proc}\colon A \rightarrow \mathfrak{P}([1 \mathinner{..} n])$ such that $\mathrm{proc}(a) \neq \emptyset$ for every $a \in A$: the set $\mathrm{proc}(a)$ is the subset of processes involved in the execution of action a (a is an internal action if, and only if, $\mathrm{proc}(a)$ is a singleton). Edges of the graph relate, for each process, the actions in which this process is involved, ordered linearly: hence, for every process p, we label edges by $\downarrow_p$ and $\uparrow_p$ to encode the linear order of actions of a process. Let

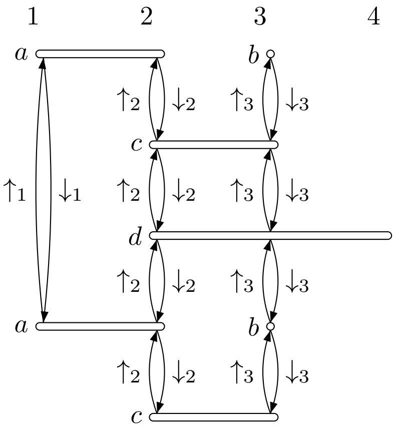


Figure 2.5: A Mazurkiewicz trace

$D = \{\downarrow_p, \uparrow_p | \; p \in [1 \mathinner{..} n]\}$ be the set of directions. Formally, we denote by $\mathcal{MT}\mathrm{race}_{\mathrm{proc}}(A)$ the subset of $\mathcal{G}(A, D)$ containing all pointed graphs $G = (V, (E_d)_{d \in D}, \lambda, v^{(i)}, v^{(f)})$ verifying the following properties:

- every vertex $v \in V$ is labeled by an action a such that $\mathrm{proc}(a)$ is compatible with the type of v, i.e., $\{p \in [1 \mathinner{..} n] \mid \downarrow_p \in \mathsf{type}(v) \vee \uparrow_p \in \mathsf{type}(v)\} \subseteq \mathrm{proc}(a)$;
- for every process $p \in [1 \mathinner{..} n]$, the set of vertices V_p labeled by actions involving process p is linearly ordered by the successor relation $E_{\downarrow_p}$: if $V_p = \{v \in V \mid p \in \mathrm{proc}(\lambda(v))\}$, there exists a unique maximal $\downarrow_p$-path, which visits exactly the vertices of V_p;
- the set of directions $D' = \{\downarrow_p \mid p \in [1 \mathinner{..} n]\}$ generates a partial order over the vertices: if there is a D'-path from vertex v to v' and from v' to v, then $v = v'$;
- $v^{(i)}$ is the first vertex in the linear order induced by the smallest active process $\min\{p \in [1 \mathinner{..} n] \mid \exists v \in V \; V_p \neq \emptyset\}$;
- $v^{(f)}$ is the last vertex in the linear order induced by the largest active process $\min\{p \in [1 \mathinner{..} n] \mid \exists v \in V \; V_p \neq \emptyset\}$.

The graph representation of a Mazurkiewicz trace is depicted in Figure 2.5. It is over alphabet $\{a, b, c\}$ with set of processes $\{1, 2, 3, 4\}$ verifying $\mathrm{proc}(a) = \{1, 2\}$, $\mathrm{proc}(b) = \{3\}$, $\mathrm{proc}(c) = \{2, 3\}$ and $\mathrm{proc}(d) = \{2, 3, 4\}$. It contains seven vertices. We have represented the processes from left to right. In this Mazurkiewicz trace, the sequence of actions involving process 2 is *acdac*, which is the linear order used in the definition. Action b is the unique internal action (for process 3).

2.1.6 Ordered Graphs

Words and nested words have an interesting point in common, namely that these structures are naturally linearly ordered, and that a particular direction in D defines this order. This is an interesting property that we will exploit later. Let us formalize this notion in a more general way. A direction set D is said to be *ordered* if there exists a partition of D into two sets $D_\rightarrow$ and $D_\leftarrow$ such that for all $d \in D$, $d \in D_\rightarrow$ if, and only if, $d^{-1} \in D_\leftarrow$. Directions in $D_\rightarrow$ (respectively, $D_\leftarrow$) are said to be *forward* directions (respectively, *backward* directions). In the case of words, direction set $\{\rightarrow, \leftarrow\}$ may be partitioned as $D_\rightarrow = \{\rightarrow\}$ and $D_\leftarrow = \{\leftarrow\}$, whereas for nested words, the partition is $D_\rightarrow = \{\rightarrow, \curvearrowright\}$ and $D_\leftarrow = \{\leftarrow, \curvearrowleft\}$.

By extension, an (A, D)-graph $G = (V, (E_d)_{d \in D}, \lambda)$ is said to be $\rightarrow$*-ordered* (or simply *ordered* if $\rightarrow$ is clear from the context) if D is ordered, $\rightarrow \in D_\rightarrow$, V comes with a total linear order $\leq$ generated by $\rightarrow$ – in the sense that a vertex v' is the direct successor of v in the linear order $\leq$ if, and only if, $(v, v') \in E_\rightarrow$ – and for all edge $(v, v') \in E$, $v < v'$ if, and only if, there exists $d \in D_\rightarrow$ such that $(v, v') \in E_d$. E.g., words and nested words are $\to$-ordered graphs with respect to the natural linear order defined over their positions, which is generated by the direction $\to$.

Finally, a pointed graph $G = (V, (E_d)_{d\in D}, \lambda, v^{(i)}, v^{(f)})$ is said to be an *ordered pointed graph* if $G = (V, (E_d)_{d\in D}, \lambda)$ is an ordered graph with $\leq$ the total linear order on V which has $v^{(i)}$ as minimal element and $v^{(f)}$ as maximal one. Notice that this enforces that ordered pointed graphs with set of vertices different from a singleton must have distinct initial and final vertices.

2.2 Weight Domains

The definitions of algebraic tools used in this manuscript are extracted from several references: [KS85, Kui97, DK09].

2.2.1 Monoids

A *monoid* is a set $\mathbb{M}$ equipped with a binary operation $\circ$ and a designated element 1 such that:

- $\circ$ is associative, i.e., $m \circ (m' \circ m'') = (m \circ m') \circ m''$ for all elements m, m', m'' of $\mathbb{M}$;
- 1 is a neutral element of $\circ$, i.e., $m \circ 1 = 1 \circ m = m$ for all elements m of $\mathbb{M}$.

In the following, we will denote $(\mathbb{M}, \circ, 1)$ such a monoid. We often write mm' instead of $m \circ m'$. The monoid $\mathbb{M}$ is said to be *commutative* if the operation $\circ$ is commutative, i.e., $m \circ m' = m' \circ m$ for all elements m and m' of $\mathbb{M}$. Alternatively, a commutative monoid may be denoted by $(\mathbb{M}, +, 0)$.

A *submonoid* of a monoid $(\mathbb{M}, \circ, 1)$ is a subset $\mathbb{M}'$ of $\mathbb{M}$ containing the neutral element 1 and stable by multiplication: for all $m_1, m_2 \in \mathbb{M}'$, $m_1 \circ m_2 \in \mathbb{M}'$. Then, denoting $\circ$ the restriction of $\circ$ to elements of $\mathbb{M}'$ (well-defined by hypothesis), the structure $(\mathbb{M}', \circ, 1)$ is automatically a monoid.

A *morphism* φ of a monoid $\mathbb{M}$ into a monoid $\mathbb{M}'$ is a mapping $\varphi\colon \mathbb{M} \to \mathbb{M}'$ compatible with the neutral elements and operations in $\mathbb{M}$ and $\mathbb{M}'$, i.e.,

- $\varphi(1) = 1$;
- $\varphi(m_1 \circ m_2) = \varphi(m_1) \circ \varphi(m_2)$ for every elements m_1, m_2 of $\mathbb{M}$.

The commutative monoid $(\mathbb{M}, +, 0)$ is said to be *complete* if every family $(m_i)_{i\in I}$ of elements of $\mathbb{M}$ over an arbitrary indexed set I is summable to some element in $\mathbb{M}$, denoted $\sum_{i\in I} m_i$, and called *sum* of the family, such that the following conditions are satisfied:

- $\sum_{i\in\emptyset} m_i = 0$, $\sum_{i\in\{j\}} m_i = m_j$ and $\sum_{i\in\{j,k\}} m_i = m_j + m_k$;
- if $I = \bigcup_{j\in J} I_j$ is a partition, $\sum_{j\in J} \left(\sum_{i\in I_j} s_i\right) = \sum_{i\in I} s_i$.

Intuitively, this means that it is possible to define infinite sums that extend the binary addition and satisfies an infinite version of the associativity axiom.

The monoid $(\mathbb{M}, +, 0)$ is said to be *continuous* if it is a complete monoid and the following conditions are satisfied:

- the relation $\leq_{\mathbb{M}}$ defined over $\mathbb{M}$ by $m_1 \leq_{\mathbb{M}} m_2$ if $m_2 = m_1 + m$ for some $m \in \mathbb{M}$, for every $m_1, m_2 \in \mathbb{M}$, is a partial order, that we call the *natural order* of the monoid;
- the sum $\sum_{i\in I} m_i$ is the *least upper bound* with respect to $\leq_{\mathbb{M}}$ of the finite sums $\sum_{i\in J} m_i$ for J finite subsets of I, for every family $(m_i)_{i\in I}$ in $\mathbb{M}$: more precisely, $\sum_{i\in J} m_i \leq_{\mathbb{M}} \sum_{i\in I} m_i$ for every finite subset J of I, and for all $m \in \mathbb{M}$, if $\sum_{i\in J} m_i \leq_{\mathbb{M}} m$ for all finite subset J of I, then $\sum_{i\in I} m_i \leq_{\mathbb{M}} m$.

Intuitively, this means that every infinite sum can be approximated by finite partial sums. In the literature, such monoids are sometimes called ω*-continuous* monoids, especially when the focus is on summation of countable families, as it will be the case in this manuscript.

A monoid $(\mathbb{M}, \circ, 1)$ is *graded* if it is equipped with a *gradation*, i.e., a morphism φ from the monoid $\mathbb{M}$ to the monoid $\mathbb{N}$ of natural numbers (equipped with addition) such that $\varphi(m) > 0$ for every element m of $\mathbb{M}$ different from 1.

2.2.2 Semirings

A *semiring* is a set $\mathbb{S}$ equipped with two binary operations $\oplus$ and $\otimes$ and two designated neutral elements $0_{\mathbb{S}}$ and $1_{\mathbb{S}}$ (respectively denoted by 0 and 1 if $\mathbb{S}$ is clear from the context) such that:

- $(\mathbb{S}, \oplus, 0_{\mathbb{S}})$ is a commutative monoid;
- $(\mathbb{S}, \otimes, 1_{\mathbb{S}})$ is a monoid;
- $\otimes$ distributes over $\oplus$, i.e., $m \otimes (m' \oplus m'') = (m \otimes m') \oplus (m \otimes m'')$ and $(m \oplus m') \otimes m'' = (m \otimes m'') \oplus (m' \otimes m'')$ for all elements m, m', m'' of $\mathbb{M}$;
- $0_{\mathbb{S}}$ is a zero, i.e., $m \otimes 0_{\mathbb{S}} = 0_{\mathbb{S}} \otimes m = 0_{\mathbb{S}}$ for all elements m of $\mathbb{M}$.

If the monoid $(\mathbb{S}, \otimes, 1_{\mathbb{S}})$ is commutative, the semiring itself is said to be *commutative*. In the following, we denote by $(\mathbb{S}, \oplus, \otimes, 0_{\mathbb{S}}, 1_{\mathbb{S}})$ the semirings.

A *subsemiring* of a semiring $(\mathbb{S}, \oplus, \otimes, 0_{\mathbb{S}}, 1_{\mathbb{S}})$ is a subset $\mathbb{S}'$ of $\mathbb{S}$ containing the neutral elements $0_{\mathbb{S}}$ and $1_{\mathbb{S}}$, and stable by addition and multiplication: for all $m_1, m_2 \in \mathbb{S}'$, $m_1 \oplus m_2 \in \mathbb{S}'$ and $m_1 \otimes m_2 \in \mathbb{S}'$. Then, denoting again $\oplus$ and $\otimes$ the restriction of these operations to elements of $\mathbb{S}'$ (well-defined by hypothesis), the structure $(\mathbb{S}', \oplus, \otimes, 0_{\mathbb{S}}, 1_{\mathbb{S}})$ is automatically a semiring.

A *morphism* φ of a semiring $\mathbb{S}$ into a monoid $\mathbb{S}'$ is a mapping $\varphi \colon \mathbb{S} \to \mathbb{S}'$ compatible with the neutral elements and operations in $\mathbb{S}$ and $\mathbb{S}'$, i.e.,

- $\varphi(0_{\mathbb{S}}) = 0_{\mathbb{S}'}$ and $\varphi(1_{\mathbb{S}}) = 1_{\mathbb{S}'}$;
- $\varphi(m_1 \oplus m_2) = \varphi(m_1) \oplus \varphi(m_2)$ and $\varphi(m_1 \otimes m_2) = \varphi(m_1) \otimes \varphi(m_2)$, for every elements m_1, m_2 of $\mathbb{S}$.

The semiring $(\mathbb{S}, \oplus, \otimes, 0_{\mathbb{S}}, 1_{\mathbb{S}})$ is said to be *complete* if the monoid $(\mathbb{S}, \oplus, 0_{\mathbb{S}})$ is complete, and if the product distributes over the infinite sums, i.e.,

$$\Big(\bigoplus_{i\in I} s_i\Big) \otimes \Big(\bigoplus_{j\in J} t_j\Big) = \bigoplus_{(i,j)\in I\times J} (s_i \otimes t_j)$$

for all families $(s_i)_{i\in I}$ and $(t_j)_{j\in J}$ of elements of $\mathbb{S}$. Informally, this means that it is possible to define infinite sums that extend the binary addition and satisfies infinite versions of associativity (see the definition of complete monoids) and distributivity axioms.

In a complete semiring, every element s of $\mathbb{S}$ admits a *star*, denoted $s^\star$, and defined by $s^\star \overset{\text{def}}{=} \bigoplus_{i\in\mathbb{N}} s^i$ (where s^i is defined recursively by $s^0 = 1_{\mathbb{S}}$ and $s^{i+1} = s^i \otimes s$). Every element s also admits a *quasi-inverse*, denoted s^+, and defined by $s^+ \overset{\text{def}}{=} \bigoplus_{i\in\mathbb{N}\setminus\{0\}} s^i$. These two objects are related by the following proposition, immediate to prove by using the axioms of the definition of complete semirings:

Proposition 2.2. *Let $\mathbb{S}$ be a complete semiring. Then for all $s \in \mathbb{S}$,*

1. $s^+ = s \otimes s^\star = s^\star \otimes s$;
2. $s^\star = 1_{\mathbb{S}} \oplus s^+$.

The semiring $(\mathbb{S}, \oplus, \otimes, 0_{\mathbb{S}}, 1_{\mathbb{S}})$ is said to be *continuous* if it is a complete semiring, and if the monoid $(\mathbb{S}, \oplus, 0_{\mathbb{S}})$ is continuous.

Example 2.3. We will use many examples of semirings in this manuscript:

- the *real semiring* $(\mathbb{R}, +, \times, 0, 1)$, having a subsemirings the *rational semiring* $(\mathbb{Q}, +, \times, 0, 1)$, the *integer semiring* $(\mathbb{Z}, +, \times, 0, 1)$ and the *natural semiring* $(\mathbb{N}, +, \times, 0, 1)$;
- the *positive real completed semiring* $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$, from which can be extracted the *natural completed semiring* $(\mathbb{N} \cup \{+\infty\}, +, \times, 0, 1)$, which are both continuous;
- the *Boolean semiring* $(\{0, 1\}, \vee, \wedge, 0, 1)$, denoted $\mathbb{B}$, which is continuous;
- the *tropical semiring* $(\mathbb{R} \cup \{+\infty, -\infty\}, \max, +, -\infty, 0)$, denoted $\mathbb{T}$, and the *arctic semiring* $(\mathbb{R} \cup \{+\infty, -\infty\}, \min, +, +\infty, 0)$, denoted $\mathbb{A}$, which are continuous;

- $([0,1], \min, \max, 1, 0)$ used in some probabilistic applications, which is continuous;
- the language semiring $(\mathfrak{P}(A^\star), \cup, \cdot, \emptyset, \{\varepsilon\})$ where the concatenation $L_1 \cdot L_2$ of two languages is defined as $\{w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2\}$, which is also continuous.

■

It is interesting to recall that not every complete semiring is continuous. For example, if we equip the semiring $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$ with the infinite summation defined by $\bigoplus_{i\in I} s_i = +\infty$ if, and only if, $\{i \mid s_i \neq 0\}$ is infinite or if $s_j = +\infty$ for some $j \in I$ (and the well-defined finite sum otherwise), the semiring is complete but not continuous. Indeed, $\bigoplus_{i\in\mathbb{N}} 1/2^i = +\infty$ whereas every finite sum is bounded above by the usual infinite sum $\sum_{i\in\mathbb{N}} 1/2^i = \lim_{n\to\infty} \sum_{i=0}^{n} 1/2^i = 2$.

Continuous semirings are interesting since they permit to get more equalities involving stars of elements. We give in the next theorem some of these, whose proofs may be found, e.g., in [Kui97, Theorem 2.4].

Theorem 2.4. *Let $\mathbb{S}$ be a continuous semiring. For every $s, s' \in \mathbb{S}$:*
(i) $(s \otimes s')^\star \otimes s = s \otimes (s' \otimes s)^\star$;[2]
(ii) $(s \oplus s')^\star = s^\star \otimes (s' \otimes s^\star)^\star = (s^\star \otimes s')^\star \otimes s^\star$.

The semiring $(\mathbb{S}, \oplus, \otimes, 0_\mathbb{S}, 1_\mathbb{S})$ is said to be *positive* if the two following conditions are verified:

- $\mathbb{S}$ is *zerosumfree*: if $s_1 \oplus s_2 = 0_\mathbb{S}$, then $s_1 = s_2 = 0_\mathbb{S}$, for every $s_1, s_2 \in \mathbb{S}$;
- $\mathbb{S}$ has no *divisors of zero*: if $s_1 \otimes s_2 = 0_\mathbb{S}$, then $s_1 = 0_\mathbb{S}$ or $s_2 = 0_\mathbb{S}$, for every $s_1, s_2 \in \mathbb{S}$.

We easily have that

Proposition 2.5. *[Gol99, Proposition 22.28] Every complete semiring is zerosumfree.*

Proof. Let $(\mathbb{S}, \oplus, \otimes, 0_\mathbb{S}, 1_\mathbb{S})$ be a complete semiring. Let s_1 and s_2 be two elements of $\mathbb{S}$ such that $s_1 \oplus s_2 = 0_\mathbb{S}$. By distributivity over infinite sums, we have

$$\bigoplus_{i\in\mathbb{N}} (s_1 \oplus s_2) = \bigoplus_{i\in\mathbb{N}} 0_\mathbb{S} = \bigoplus_{i\in\mathbb{N}} 0_\mathbb{S} \otimes 0_\mathbb{S} = 0_\mathbb{S} \otimes \left(\bigoplus_{i\in\mathbb{N}} 0_\mathbb{S} \right) = 0_\mathbb{S}$$

By partitioning the infinite sum $\bigoplus_{i\in\mathbb{N}} (s_1 \oplus s_2)$ in three parts, we obtain

$$0_\mathbb{S} = s_1 \oplus \bigoplus_{i\in\mathbb{N}\setminus\{0\}} s_1 \oplus \bigoplus_{i\in\mathbb{N}} s_2$$

Notice that $\mathbb{N} \setminus \{0\}$ is in bijection with $\mathbb{N}$. Generally, if $\varphi\colon J \to I$ is a bijection, and $(s_i)_{i\in I}$ a family of elements of $\mathbb{S}$, then, by using the partitioning again, we have

$$\bigoplus_{i\in I} s_i = \bigoplus_{j\in J} \bigoplus_{i\in\varphi(j)} s_i = \bigoplus_{j\in J} s_{\varphi(j)}$$

the last equality coming from the first property of complete monoids. Hence, infinite sums are preserved by bijective renaming of families. In our case, we obtain that

$$0_\mathbb{S} = s_1 \oplus \bigoplus_{i\in\mathbb{N}} s_1 \oplus \bigoplus_{i\in\mathbb{N}} s_2$$

By refactoring the partition, we finally get

$$0_\mathbb{S} = s_1 \oplus \bigoplus_{i\in\mathbb{N}} (s_1 \oplus s_2) = s_1 \oplus 0_\mathbb{S} = s_1$$

which proves that $s_1 = 0_\mathbb{S}$. Since $s_1 \oplus s_2 = 0_\mathbb{S}$, we also have $s_2 = 0_\mathbb{S}$. □

[2] One may equivalently state that $(s \otimes s')^\star = 1_\mathbb{S} \oplus s \otimes (s' \otimes s)^\star \otimes s'$.

However, it is not true that every complete (or even continuous) semiring is positive, as it is shown in Remark 2.6.

Finally, the semiring $(\mathbb{S}, \oplus, \otimes, 0_\mathbb{S}, 1_\mathbb{S})$ is said to be *idempotent* if it is idempotent for the addition: it is sufficient to verify $1_\mathbb{S} \oplus 1_\mathbb{S} = 1_\mathbb{S}$. The tropical and arctic semirings are typical examples of idempotent semirings.

2.2.3 Formal Power Series and Polynomials

In all this section, we fix a semiring $(\mathbb{S}, \oplus, \otimes, 0, 1)$.

Let Z be a set. A *formal power series* f over Z (or *series* for short) is a map $f\colon Z \to \mathbb{S}$. Alternatively, f can be seen as a formal sum $\bigoplus_{z\in Z} f(z)z$, so that $f(z)$ is called the *coefficient* of z in f. The set of series over Z with coefficients in $\mathbb{S}$ is denoted by $\mathbb{S}\langle\langle Z\rangle\rangle$.

The *support* of a series $f \in \mathbb{S}\langle\langle Z\rangle\rangle$ is the set $\{z \in Z \mid f(z) \neq 0\}$, and is denoted $\mathrm{supp}(f)$. A series with a finite support is called a *polynomial*. The set of polynomials over Z with coefficients in $\mathbb{S}$ is denoted by $\mathbb{S}\langle Z\rangle$.

We can lift addition and product[3] from $\mathbb{S}$ to $\mathbb{S}\langle\langle Z\rangle\rangle$ pointwisely by defining for every series f and g:

$$(f \oplus g)(z) = f(z) \oplus g(z) \quad \text{and} \quad (f \odot g)(z) = f(z) \otimes g(z) \qquad \text{for all } z \in Z\,.$$

The product of series defined thereby is usually called the *Hadamard product* of two series. Then $(\mathbb{S}\langle\langle Z\rangle\rangle, \oplus, \odot, 0, 1)$ is a semiring where 0 (respectively 1) denotes the series mapping every element $z \in Z$ to 0 (respectively 1). This semiring is commutative if, and only if, $\mathbb{S}$ is a commutative semiring.

When $(Z, \circ, 1_Z)$ is a monoid, it is possible to define another product for series: the *Cauchy product* $f \otimes g$ of two series f and g is then defined by

$$(f \otimes g)(z) = \bigoplus_{z = x\,\circ\, y} f(x) \otimes g(y) \qquad \text{for all } z \in Z\,.$$

This sum may be infinite, but is well-defined when either the monoid is graded or the semiring is complete. The Cauchy product is then associative and admits as neutral element the polynomial 1_Z, whose only non null coefficient is the one of 1_Z which is 1. Hence, $(\mathbb{S}\langle\langle Z\rangle\rangle, \oplus, \otimes, 0, 1_Z)$ is a semiring when the monoid is graded or the semiring is complete.

When the semiring $\mathbb{S}$ is complete, we can also lift infinite sums pointwise to $\mathbb{S}\langle\langle Z\rangle\rangle$ which becomes a complete semiring. Moreover, if $\mathbb{S}$ is a continuous semiring, $\mathbb{S}\langle\langle Z\rangle\rangle$ is also continuous, equipped with a natural order defined as the pointwise extension of the natural order of $\mathbb{S}$.

2.2.4 Matrices

In all this section, we fix a semiring $(\mathbb{S}, \oplus, \otimes, 0, 1)$.

Let I and J be two non-empty sets. A *matrix* indexed by $I\times J$ is a mapping $M\colon I\times J \to \mathbb{S}$: value $M(i, j)$ is often denoted as $M_{i,j}$, and called (i, j)-coefficient of M. The matrix itself is sometimes denoted by $(M_{i,j})_{i\in I, j\in J}$. The set of all matrices indexed by $I \times J$ and with values in $\mathbb{S}$ is denoted by $\mathbb{S}^{I\times J}$. If $I' \subseteq I$ and $J' \subseteq J$, we denote by $M_{|I'\times J'}$ the submatrix of M indexed by $I' \times J'$.

We may define neutral elements and binary operations so that sets of matrices become monoids and semirings. First, denoting by 0 the matrix in $\mathbb{S}^{I\times J}$ with all coefficients equal to 0, and defining $M \oplus M'$, for every matrices $M, M' \in \mathbb{S}^{I\times J}$, as the matrix with (i, j)-coefficient being $M_{i,j} \oplus M'_{i,j}$, we obtain that $(\mathbb{S}^{I\times J}, \oplus, 0)$ is a commutative monoid.

[3] The notation $\odot$ used instead of the more natural one, $\otimes$, to denote the lifting of the product is motivated by the introduction of the Cauchy product below, often considered as the *natural* product of formal power series.

Moreover if J is finite, we may define the product of two matrices $M \in \mathbb{S}^{I\times J}$ and $M' \in \mathbb{S}^{J\times K}$ as being the matrix $M'' \in \mathbb{S}^{I\times K}$ with (i,k)-coefficient given by $\bigoplus_{j\in J} M_{i,j} \otimes M'_{j,k}$. Denoting by $\mathrm{Id} \in \mathbb{S}^{I\times I}$ the square matrix with 1 on the main diagonal, and 0 at other positions, $(\mathbb{S}^{I\times I}, \oplus, \otimes, 0, \mathrm{Id})$ is a semiring. In case $\mathbb{S}$ is a continuous semiring, $\mathbb{S}^{I\times I}$ is also a continuous semiring, and we denote by $M^\star$ the infinite sum $\sum_{n\geq 0} M^n$.

Remark 2.6. If we consider the continuous semiring $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$, the semiring of square matrices with coefficients in this semiring is continuous but not commutative, and not positive, as it contains zero divisors: for example for square matrices of size 2, we have

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = 0$$

■

In [Con71], Conway proved a very useful lemma permitting to compute the star of a matrix with additions, products and stars in the semiring. It can also be seen as a rewriting of McNaughton-Yamada algorithm, or the state elimination method used to construct a regular expression equivalent to a finite-state automaton. We state it here and will use it at several occasions in this manuscript. Proofs can be found in [KS85] or [Sak09]: they are both based on an adaptation of Arden's lemma in the case of continuous semirings.

Lemma 2.7. *Let $\mathbb{S}$ be a continuous semiring and $M \in \mathbb{S}^{I\times I}$ be a square matrix. For any block decomposition $M = \left(\begin{array}{cc} A & B \\ C & D \end{array}\right)$ with A and D square matrices, we have*

$$M^\star = \left(\begin{array}{cc} (A \oplus B \otimes D^\star \otimes C)^\star & A^\star \otimes B \otimes (D \oplus C \otimes A^\star \otimes B)^\star \\ D^\star \otimes C \otimes (A \oplus B \otimes D^\star \otimes C)^\star & (D \oplus C \otimes A^\star \otimes B)^\star \end{array}\right)$$

CHAPTER 3

Hybrid Expressions

Simplicity is the glory of expression.

Walt Whitman

Regular expressions are widely used in many programming languages to concisely denote string patterns. They permit to efficiently locate particular words or sequences of characters in a text, a web page or any source of characters. Written with a flexible and handy syntax, they are interpreted by a parser which aims at executing it over the source text.

More theoretically, they are bound to the name of Stephen Cole Kleene who exhibited [Kle56] the concept of regular languages: regular expressions are then a way to denote such regular languages easily. His famous eponymous theorem states that this class of languages coincides with the ones recognized by finite state automata: we will come back to this fundamental result in Chapter 4.

Whereas regular expressions are now widely used and studied in order to denote patterns of sequences of characters, i.e., words over a finite alphabet, it is not so long ago that they have been used for extended purposes. Amongst various works, we may cite caterpillar expressions, introduced by [BKW00] – which are regular expressions navigating in unranked trees – and forest expressions, to generate more branching behaviors, exposed in [Boj07] (also exposed in [Cyr10, BS12] for nested words).

Another domain of extension consists in introducing quantities in expressions. Whereas weighted regular series have been discovered by Marcel-Paul Schützenberger in [Sch61] – both with weighted automata and the extension of Kleene theorem stating the equivalence of both formalisms – the notion of weighted expression as a way to denote such series has emerged later (see, e.g., [Sak09] for a recent presentation of this large field). It has to be noticed that recent works are still produced in order to investigate further these weighted expressions and several of their extensions. For example, new parsing algorithms of (weighted) regular expressions are still implemented, as we may find in [FHW10] recently. Weighted expressions have also been investigated in the framework of infinite words, with an average semantics, in [DM10b].

Our goal in this chapter is to define and study a new extension of weighted expressions, able to denote a broader class of quantitative properties, yet with a readable formalism inspired by the rich history of regular expressions. An original part of this work is the use of variables to enrich the power of expressions. After giving the syntax and semantics of these new expressions, we will consider some possible restrictions or other denotations, that could permit to express more easily some properties (for example, by hiding the use of variables, which could be seen as a weak point of this formalism, even if it permits to express rich quantitative properties). As a final section, we will give an algebraic view of the semantics of these expressions, that will be of the greatest interest to prove several theorems of next chapters.

3.1 Hybrid Weighted Expressions

In this section, we introduce expressions with weights and variables. Like classical regular expressions, their syntax is based on operators $+$ (for alternatives), $\cdot$ (for sequences), and $\star$ (for iteration). Then, $+$ and $\cdot$ are interpreted as sum and Cauchy product in a semiring, respectively.

We first introduce these expressions with some examples, one on words, a second on nested words and a last one over graphs.

Example 3.1. We consider a word over the alphabet $\{a, b\}$. The classical regular expression $(a+b)^\star \cdot b \cdot (a+b)^\star$ *checks* that the given word contains an occurrence of letter b. For conciseness reasons, and for seek of uniformity, we will rather use the shortcut $\rightarrow$ to denote the non-guarded *move to the right* encoded by the choice $(a+b)$, and use a weighted semantics, for example in the semiring $(\mathbb{N} \cup \{+\infty\}, +, \times, 0, 1)$. Hence, expression $\rightarrow^\star \cdot b \cdot \rightarrow^\star$ *counts* the number of occurrences of letter b in the word: 4 in the word *baabbaba* for example. Possibly, we can decouple the test and the move understood in the symbol b: indeed, we introduce the test $b?$ that stay on the same position and checks whether the current position holds letter b. Therefore, we may alternatively use the equivalent expression $\rightarrow^\star \cdot b? \cdot \rightarrow \cdot \rightarrow^\star$. ■

Example 3.2. We consider a nested word over the alphabet $\{a, b\}$. Consider the more complex task of counting the total number of occurrences of the letter b *inside a context with a call position labeled with* a: more formally we want to sum over all possible call positions labeled with a, the number of occurrences of b that appear strictly in-between this position and the matching return. For the nested word of Figure 2.3, we must count 4 (in particular, position 7 must count for both call positions 5 and 6). In our formalism, we will achieve this task again in semiring $(\mathbb{N} \cup \{+\infty\}, +, \times, 0, 1)$ using expression:

$$E = \rightarrow^\star \cdot (a? \wedge \curvearrowright ?) \cdot x! \big(\rightarrow^\star \cdot x? \cdot \curvearrowright \cdot (\neg x? \cdot \leftarrow)^+ \cdot b? \cdot \rightarrow^\star\big) \cdot \rightarrow^\star .$$

First, we search for a call position labeled with a using expression $\rightarrow^\star \cdot (a? \wedge \curvearrowright ?)$: there, we again use the test $a?$ to check the letter without moving, and moreover test whether the current position is a call position using $\curvearrowright ?$. Then, we mark, with $x!-$, the call position of the interesting context with a variable of name x: this permits us to compute independently the subexpression between parentheses on the nested word (starting from the beginning again) with the previous position marked with variable x. The latter subexpression first searches for the variable with $\rightarrow^\star \cdot x?$, follows the nesting edge from the call to the corresponding return, and then moves backward inside the context with $(\neg x? \cdot \leftarrow)^+$ to pick non-deterministically a position carrying letter b. ■

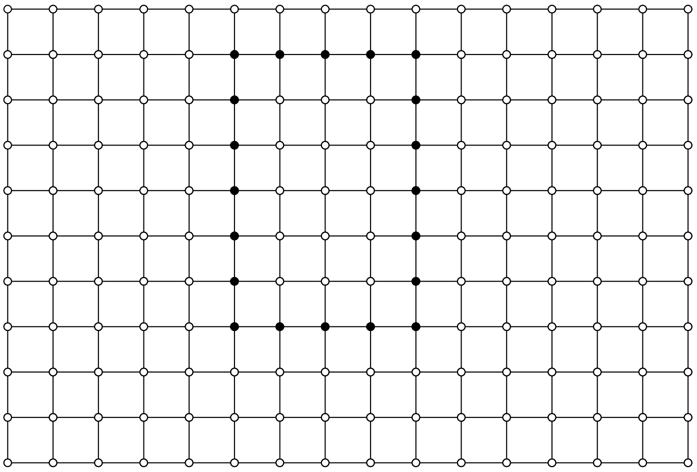


Figure 3.1: A picture with a highly contrasted rectangle

Example 3.3. Consider now a picture in grayscales, over the alphabet $[0\mathinner{..}255]$ representing the level of gray of the pixel. The same way regular expressions are a way to express word patterns, our expression could be a way to define pictorial patterns. We start with the toy example consisting of finding rectangles in a picture. The pattern will consist of monocolor pixels grouped in a rectangle shape encircled by pixels of highly contrasted colors, like in the Figure 3.1. Supposing that the top left corner of the rectangle is marked with a variable of name x, the expression will follow the rectangle clockwisely, expecting to reach back the position marked with variable x after changing directions 3 times. Moreover, the expression checks at every point that the outside of the rectangle has a grayscale highly contrasted with the (constant) grayscale of the rectangle.

$$\begin{aligned}E_{\text{rectangle}} = x? \cdot \sum_{c \in [0\mathinner{..}255]} & (c? \cdot F^c_{\uparrow} \cdot \rightarrow)^+ \cdot F^c_{\uparrow} \cdot (c? \cdot F^c_{\rightarrow} \cdot \downarrow)^+ \cdot F^c_{\rightarrow} \\ & \cdot (c? \cdot F^c_{\downarrow} \cdot \leftarrow)^+ \cdot F^c_{\downarrow} \cdot (c? \cdot F^c_{\leftarrow} \cdot \uparrow)^+ \cdot F^c_{\leftarrow} \cdot x?\end{aligned}$$

where the expression F^c_d tests that the position in direction d has a color sufficiently distinct from the grayscale c:

$$F^c_d = \sum_{\substack{c' \in [0\mathinner{..}c-50] \\ \cup [c+50\mathinner{..}255]}} d \cdot c'? \cdot d^{-1}$$

Notice that we do not care about the color of the interior of the rectangle, but we could refine our expression to also take it into account. Imagine that our final goal is to count the number of such highly contrasted rectangles in a picture, and then compare this number with the number of another pattern described by an expression E_{pattern}, to compute the maximum of these two cardinalities. A maximum operation is then necessary, which forces us to use semiring $(\mathbb{N} \cup \{-\infty, +\infty\}, \max, +, -\infty, 0)$. Then, counting the number of rectangles cannot be done by using a non-deterministic choice (as if in semiring $(\mathbb{N} \cup \{+\infty\}, +, \times, 0, 1)$). We rather use a deterministic search of the picture, using the variable x to check whether we found a new rectangle:

$$\begin{aligned}G_{\text{rectangle}} = \Big(\downarrow \cdot \big([0 + 1 \cdot (x!(\rightarrow^\star \cdot \downarrow^\star \cdot E_{\text{rectangle}} \cdot \rightarrow^\star \cdot \downarrow^\star))] \cdot \rightarrow\big)^+ \cdot \\ \neg(\rightarrow?) \cdot \leftarrow^+ \cdot \neg(\leftarrow?)\Big)^+ \cdot \neg(\downarrow?) \cdot \rightarrow^\star\end{aligned}$$

The expression inside the rectangle bracket is the sum between weight 0 and weight 1 multiplied by the check of expression $E_{\text{rectangle}}$ over the current rectangle: sum is evaluated as a maximum, and product as an addition, hence this whole expression computes 1 if there is a highly contrasted rectangle with upper left corner at the current position, and 0 otherwise. Inside the $x!-$ operator, starting from the upper left pixel of the picture (as this is the initial vertex we chose in our representation of pictures), we first search (unambiguously) for the position holding variable x, evaluate expression $E_{\text{rectangle}}$, and reach the lower right pixel of the picture (the final vertex) to continue the computation. By the same way, we may obtain a formula G_{pattern} counting the number of occurrences of the other pattern, and finally consider formula $G_{\text{rectangle}}+G_{\text{pattern}}$ to count the maximum between these two cardinalities, as $+$ is evaluated using the first operation of the semiring, i.e., the max operation in this case. ■

We turn to the formal syntax of expressions. We let $\mathrm{Var} = \{x, y, \dots\}$ be an infinite set of variables. In all this chapter, we let D be a fixed set of directions. Pebble weighted expressions are built upon simple Boolean *tests* that we define first.

Definition 3.4. We let $\mathrm{Test}(A, D, \mathrm{Var})$ (or shortly, Test) be the set of formulae defined by the following grammar:

$$\alpha ::= \top \mid \mathrm{init?} \mid \mathrm{final?} \mid a? \mid d? \mid x? \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \tag{3.1}$$

where $a \in A$, $d \in D$ and $x \in \mathrm{Var}$. ■

Thus, a test is a Boolean combination of atomic checks allowing one to verify whether a given vertex has label a, whether it enables a certain direction d, or whether it carries an occurrence of variable x, respectively. Recall that we defined in Chapter 2 the notion of type $\tau \in \mathfrak{P}(D)$ of a vertex of a graph, as a subset of D: in particular, we can check the type of a vertex by using a *maximal* Boolean combination of directions.

Given a test α of Test, a variable $x \in \mathrm{Var}$ is *free* in α if the latter contains the atom $x?$. More formally, we denote $\mathrm{Free}(\alpha)$ the (finite) set of pebble names free in α, which is defined inductively by letting

$$\begin{aligned}
\mathrm{Free}(\top) &= \mathrm{Free}(\mathrm{init?}) = \mathrm{Free}(\mathrm{final?}) = \mathrm{Free}(a?) = \mathrm{Free}(d?) = \emptyset\\
\mathrm{Free}(x?) &= \{x\}\\
\mathrm{Free}(\neg\alpha) &= \mathrm{Free}(\alpha)\\
\mathrm{Free}(\alpha_1 \wedge \alpha_2) &= \mathrm{Free}(\alpha_1 \vee \alpha_2) = \mathrm{Free}(\alpha_1) \cup \mathrm{Free}(\alpha_2)
\end{aligned}$$

with $a \in A$, $d \in D$ and $x \in \mathrm{Var}$.

We are now ready to define the semantics of tests, in the general case of graphs. Let $G = (V, (E_d)_{d\in D}, \lambda, v^{(i)}, v^{(f)}) \in \mathcal{G}(A, D)$ be a pointed graph. Let $v \in V$ be a position and $\sigma\colon \mathrm{Var} \rightharpoonup V$ be a valuation. We define inductively the semantics of a test $\alpha \in \mathrm{Test}$ with free variables $\mathrm{Free}(\alpha)$ contained in the domain $\mathrm{dom}(\sigma)$ of the valuation: we denote $G, \sigma, v \models \alpha$ if α is verified over the given model which is defined in Table 3.1.

Notice that the semantics only depends on the valuation of free variables of the test. Next, we present hybrid weighted expressions over a continuous semiring $(\mathbb{S}, \oplus, \otimes, 0, 1)$.

Definition 3.5. We let $\mathrm{HWE}(\mathbb{S}, A, D, \mathrm{Var})$ (or shortly HWE) be the set of *hybrid weighted expressions* defined by the following grammar:

$$E ::= s \mid \alpha \mid d \mid E + E \mid E \cdot E \mid E^{+} \mid x!E$$

where $s \in \mathbb{S}$, $\alpha \in \mathrm{Test}(A, D, \mathrm{Var})$, $d \in D$ and $x \in \mathrm{Var}$. ■

The set of variables appearing in an expression E of HWE (either in tests or in constructions $x!-$) is denoted by $\mathrm{Var}(E)$. Given an expression E of HWE, a variable $x \in \mathrm{Var}$ is *free* in E if it is free in some test used in E, and not bound by an expression $x!-$. More

Table 3.1: Semantics of tests in $\mathrm{Test}(A, D, \mathrm{Var})$

$$
\begin{array}{l}
G, \sigma, v \models \top \\
G, \sigma, v \models \text{init? if, and only if, } v = v^{(i)} \\
G, \sigma, v \models \text{final? if, and only if, } v = v^{(f)} \\
G, \sigma, v \models a\text{? if, and only if, } \lambda(v) = a \\
G, \sigma, v \models d\text{? if, and only if, } d \in \mathsf{type}(v) \\
G, \sigma, v \models x\text{? if, and only if, } \sigma(x) = v \\
G, \sigma, v \models \neg\alpha \text{ if, and only if, } G, \sigma, v \not\models \alpha \\
G, \sigma, v \models \alpha \wedge \alpha' \text{ if, and only if, } G, \sigma, v \models \alpha \text{ and } G, \sigma, v \models \alpha' \\
G, \sigma, v \models \alpha \vee \alpha' \text{ if, and only if, } G, \sigma, v \models \alpha \text{ or } G, \sigma, v \models \alpha'
\end{array}
$$

formally, we denote $\mathrm{Free}(E)$ the (finite) set of free variabes in E, which is defined inductively by letting

$$
\begin{aligned}
\mathrm{Free}(s) = \mathrm{Free}(d) &= \emptyset & \mathrm{Free}(\alpha) &= \mathrm{Free}(\alpha) \\
\mathrm{Free}(E^+) &= \mathrm{Free}(E) & \mathrm{Free}(E_1 + E_1) &= \mathrm{Free}(E_1) \cup \mathrm{Free}(E_2) \\
\mathrm{Free}(x!E) &= \mathrm{Free}(E) \setminus \{x\} & \mathrm{Free}(E_1 \cdot E_2) &= \mathrm{Free}(E_1) \cup \mathrm{Free}(E_2)
\end{aligned}
$$

with $s \in \mathbb{S}$, $\alpha \in \mathrm{Test}$, $d \in D$ and $x \in \mathrm{Var}$.

We now turn to the semantics of hybrid weighted expressions. A hybrid weighted expression is interpreted over a pointed graph $G = (V, (E_d)_{d \in D}, \lambda, v^{(i)}, v^{(f)}) \in \mathcal{G}(A, D)$ with a marked initial vertex v, a marked final vertex v' and a valuation $\sigma \colon \mathrm{Var} \rightharpoonup V$. The atomic expression $d \in D$ has its natural interpretation as the binary relation E_d and is evaluated 1 or 0 depending on whether or not $(v, v') \in E_d$. On the contrary, expressions s, α, $x!E$ are non-progressing and require $v = v'$. In particular, $x!E$ evaluates E in the graph G, from $v^{(i)}$ to $v^{(f)}$, with the current vertex marked with a new occurrence of variable x. Formally, the semantics of a hybrid weighted expression $E \in \mathrm{HWE}$ with $\mathrm{Free}(E) \subseteq \mathrm{dom}(\sigma)$ is the weight in $\mathbb{S}$ denoted $[\![E]\!](G, \sigma, v, v')$ defined inductively in Table 3.2. In this table, we denote E^n the n-th iteration of expression E defined inductively by $E^1 = E$ and $E^{n+1} = E^n \cdot E$ for $n > 0$; occasionally, we may denote the expression 1 as E^0. Notice that the infinite sum used to define the semantics of E^+ is well-defined as we consider a continuous semiring $\mathbb{S}$. We will see in Chapter 7 a way to remove the continuity condition over the semiring, at the price of restricting the expressions.

Notice that the semantics does only depend on the valuation of free pebble names of the pebble weighted expression. If the assignment σ is not relevant, i.e., if there are no free pebble names in the expression, we may denote $[\![E]\!](G, v, v')$ the semantics.

By default, v and v' are the initial and final vertices of the pointed graph G, i.e., $v^{(i)}$ and $v^{(f)}$, so that we use $[\![E]\!](G)$ (respectively, $[\![E]\!](G, \sigma)$) as shortcuts for $[\![E]\!](G, v^{(i)}, v^{(f)})$ (respectively, $[\![E]\!](G, \sigma, v^{(i)}, v^{(f)})$): in that case, we may see $[\![-]\!]$ as a formal power series in $\mathbb{S}\langle\!\langle \mathcal{G}(A, D) \rangle\!\rangle$.

Remark 3.6. We get the classical Kleene star as an abbreviation: $E^\star = 1 + E^+$. As usual, we may sometimes remove the symbol $\cdot$ used for the concatenation of two expressions and write $E_1 E_2$ for $E_1 \cdot E_2$. In words and nested words for example, it is also convenient to introduce macros for "check-and-move": $a \overset{\text{def}}{=} a? \cdot \rightarrow$. This allows us, for example, to use common syntax such as $(ab)^+abc$, or to write $\rightarrow^\star abba \leftarrow^+ \text{init?} \rightarrow^\star baab \rightarrow^\star$ to identify words having both $abba$ and $baab$ as factors. ■

We call *depth* of an expression E in HWE, denoted by $\mathrm{depth}(E)$, its maximal number of

Table 3.2: Semantics of HWE($\mathbb{S}, A, D, \mathrm{Var}$)

$$
\begin{aligned}
\llbracket s \rrbracket(G,\sigma,v,v') &= \begin{cases} s & \text{if } v = v' \\ 0 & \text{otherwise} \end{cases} \\
\llbracket \alpha \rrbracket(G,\sigma,v,v') &= \begin{cases} 1 & \text{if } v = v' \wedge G,\sigma,v \models \alpha \\ 0 & \text{otherwise} \end{cases} \\
\llbracket d \rrbracket(G,\sigma,v,v') &= \begin{cases} 1 & \text{if } (v,v') \in E_d \\ 0 & \text{otherwise} \end{cases} \\
\llbracket E_1 \cdot E_2 \rrbracket(G,\sigma,v,v') &= \bigoplus_{v'' \in V} \llbracket E_1 \rrbracket(G,\sigma,v,v'') \otimes \llbracket E_2 \rrbracket(G,\sigma,v'',v') \\
\llbracket E_1 + E_2 \rrbracket(G,\sigma,v,v') &= \llbracket E_1 \rrbracket(G,\sigma,v,v') \oplus \llbracket E_2 \rrbracket(G,\sigma,v,v') \\
\llbracket E^+ \rrbracket(G,\sigma,v,v') &= \bigoplus_{n>0} \llbracket E^n \rrbracket(G,\sigma,v,v') \\
\llbracket x!E \rrbracket(G,\sigma,v,v') &= \begin{cases} \llbracket E \rrbracket(G,\sigma[x \mapsto v], v^{(i)}, v^{(f)}) & \text{if } v = v' \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

nested $x!-$ operators. It is defined inductively by

$$
\begin{aligned}
\mathrm{depth}(s) &= \mathrm{depth}(d) = \mathrm{depth}(\alpha) = 0 \\
\mathrm{depth}(E^+) &= \mathrm{depth}(E) \\
\mathrm{depth}(E_1 + E_2) &= \mathrm{depth}(E_1 \cdot E_2) = \max(\mathrm{depth}(E_1), \mathrm{depth}(E_2)) \\
\mathrm{depth}(x!E) &= 1 + \mathrm{depth}(E)
\end{aligned}
$$

Example 3.7. Over the semiring $(\mathbb{N} \cup \{-\infty, +\infty\}, \max, +, -\infty, 0)$, consider the hybrid weighted expression (over nested words)

$$E = \big[1 \cdot \curvearrowright ? \cdot \rightarrow \cdot \neg(\curvearrowleft ?) + 0 \cdot \big(\neg(\curvearrowright ? \vee \curvearrowleft ?) \cdot \rightarrow \cdot \neg(\curvearrowleft ?) + \curvearrowright + \curvearrowleft ? \cdot \rightarrow \cdot \neg(\curvearrowleft ?)\big)\big]^\star \cdot x? \cdot \rightarrow^\star .$$

Notice the use of $1 \in \mathbb{N}$ which is not the unit of the semiring. Moreover, operations $+$ are resolved by the max operator, whereas concatenation implies the use of addition in $\mathbb{N} \cup \{-\infty, +\infty\}$. For every nested word $G \in \mathcal{N}\mathrm{est}(A)$ (with set of vertices $[0 \mathinner{..} n]$), and every vertex v, $\llbracket E \rrbracket(G, [x \mapsto v])$ computes the call-depth of vertex v. Indeed the first Kleene star is unambiguous, meaning that only one path starting from position 0 will lead to x in this iteration; along this path – the shortest one – we only count the number of times we enter inside the context of a call position. Hence, the maximal call-depth of a nested word can be computed with expression $E' = \rightarrow^\star \cdot (x!E) \cdot \rightarrow^\star$. ■

3.2 Syntactical Subclasses of Hybrid Weighted Expressions

We described in the previous section the full class of hybrid weighted expressions we will use in this manuscript. However, it may be the case that only a subset of the *features* we presented is sufficient to model the desired quantitative property. In this section, we present some interesting subclasses of HWE.

3.2.1 Weighted Expressions

At the first sight, the variable mechanism of hybrid weighted expressions may seem intricate and far-fetched from the spirit of classical regular (or even weighted) expressions. We will see that this complication is unavoidable in order to express powerful quantitative properties, and yet permits to express them in an elegant and concise way. However, we consider now the subclass of HWE that does not use any variables. Let $(\mathbb{S}, \oplus, \otimes, 0, 1)$ be a continuous semiring again.

Definition 3.8. We let $\mathrm{WE}(\mathbb{S}, A, D, \mathrm{Var})$ (or shortly WE) be the set of *weighted expressions* defined by the following grammar:

$$E ::= s \mid \alpha \mid d \mid E + E \mid E \cdot E \mid E^{+}$$

where $s \in \mathbb{S}$, $\alpha \in \mathrm{Test}(A, D, \mathrm{Var})$ and $d \in D$. ■

Notice that we keep in $\mathrm{Test}(A, D, \mathrm{Var})$ the possibility to check the presence of a variable over a vertex, however, we discard the ability to place a new occurrence of a variable with the $x!-$ operator.

As a subclass of HWE, weighted expressions in WE inherits their semantics from those of expressions in HWE. As an illustration that weighted expressions may be useful, even without pebbles, we give another example here.

Example 3.9. Weighted expressions may be very handy in the field of Natural Language Processing (see [KM09]), in particular for automatic translation, speech recognition or transliteration. All these tasks have in common to split the problem into independent parts, certain directly related to the specific task and others related to the knowledge of the current language. For example, in the translation task from French sentences to English sentences, one splits the problem into first knowing translation of single words and then modeling English sentences (knowledge which is independent from the translation task). The second part, namely to know whether a sequence of words is a *good* English sentence, is known as *language modeling.* Often this knowledge is learned from a large corpus of English texts, and stored into a formal model, e.g., a weighted finite state automaton representing the probability distribution $\mathbb{P}$ of well-formed English sentences. The translation task is then resolved by first generating several English sentences from the original French one (due to ambiguity of the word-by-word translation task), and then choosing among this set of sentences the ones with highest probability.

One broadly used language model is the n-gram model, where the probability of a word in a sentence depends only on the previous $n-1$ words: for example in a 1-gram model, only the individual word frequencies are relevant to generate well-formed English sentences, whereas in a 2-gram model, the probability of a word depends on the very same frequency distribution and also the previous word. Typical values of the parameter n are 2 or 3, and the largest value for which a full model of English has been produced is 7 [BPX^{+}07]. To formally describe these models, and further study them, let us define them using weighted expressions. Let D denote the dictionary of words in the language. Suppose we are given the conditional probability distributions $\mathbb{P}(w_n \mid w_1, \ldots, w_{n-1})$ in the n-gram model (with $w_i \in D$ for all i). The probability of a sentence $(w_i)_{1 \leq i \leq m} \in D^m$ can be given by the following weighted regular expression in a 1-gram model and a 3-gram model:

$$E_1 = \Big(\sum_{u \in D} u \cdot \mathbb{P}(u)\Big)^{\star} \qquad E_3 = \rightarrow \cdot \rightarrow \cdot \Big(\sum_{u,v,w \in D} \leftarrow \cdot \leftarrow \cdot\, u \cdot v \cdot w \cdot \mathbb{P}(w \mid u, v)\Big)^{\star}$$

Expression E_3 uses the opportunity to move forward and backward to easily recover the context. Notice that expression E_3 is quite readable and intuitive. One could write an equivalent expression only moving forward, but imagine how intricate it would be since positions would have to encode the context, i.e., the last two words.

Actually, expression E_3 is not small since the sum hides the very big set D^3: for a dictionary of size 1 million, this seems already unpracticable. But in practice, a much smaller expression could be sufficient. First, for many words, the frequency distribution of the word w is a sufficiently good approximation of the conditional probability $\mathbb{P}(w \mid u, v)$. Let us denote D_0 this set of words. For instance, the probability of observing the word `the` may not really depend on the previous words. Then, let D_1 be the set of words (disjoint from D_0) such that only the previous word is necessary to describe the probability. Finally, let D_2 be the rest of the dictionary, i.e., the set of words such that the two previous words are really necessary to describe the probability $\mathbb{P}(u_n \mid u_{n-2}, u_{n-1})$. Now, we may replace expression E_3 by the following expression, whose size is much smaller if D_0 and D_1 contain enough words:

$$\Big(\sum_{w \in D_0} w \cdot \mathbb{P}(w) + \sum_{w \in D_1, v \in D} \leftarrow \cdot\, v \cdot w \cdot \mathbb{P}(w \mid v) + \sum_{w \in D_2, u, v \in D} \leftarrow \cdot \leftarrow \cdot\, u \cdot v \cdot w \cdot \mathbb{P}(w \mid u, v)\Big)^{\star}$$

■

3.2.2 Chop Weighted Expressions: An Alternative to Variables

Sometimes variables are unavoidable in order to express naturally some quantitative properties.

Example 3.10 (continued from Example 3.9)**.** We add internationalization to the natural language processing task previously studied, which means that the user has the ability to write/speak alternately in two or more languages, e.g., English and French. All tasks such as automatic translation or speech recognition are now more complex since there is no a priori knowledge of the current language of the speaker. Again, splitting the problem into independent parts, we have to know the probability distributions $\mathbb{P}_L$ for every involved language L, and, assuming a current language L, we should be able to solve the language processing task with a procedure Task_L. Then, before processing the next word, we start a computation which re-reads the current prefix of the text in order to compute using $\mathbb{P}_L$ the probability that the current language L is still valid. The next word is then processed with the current or the alternate language. In order to compute the probability that the current language is still valid, we may use variables (and intuitively we need): we mark the current position with a variable and read the current prefix of the text with a subexpression modeling the current language. Then we return to the marked position in order to resume the top level computation, in either languages, depending on the probability we computed. ■

More critically, variables really have some intrinsic expressiveness power:

Proposition 3.11. *There exists a hybrid weighted expressions E over the semiring* $(\mathbb{N} \cup \{+\infty\}, +, \times, 0, 1)$ *such that no weighted expression $E' \in \mathrm{WE}$ is equivalent to E, i.e., generates the same semantics over every graph.*

Proof. This result is more easily proved by means of automata, and is indeed a corollary of Theorem 4.18 which states the same result for automata, by using translation from automata to expressions and vice versa stated in Theorems 4.9 and 4.31. Stated in the syntax of hybrid weighted expressions, the counter example that is shown to be not recognized by a weighted expression is

$$E = \Big(x!\big((2 \cdot \rightarrow)^{+}\big) \cdot \rightarrow\Big)^{+}$$

evaluated over the set of words $\mathcal{W}\mathrm{ord}(A)$. □

The expression used as a counter example in this proof indeed uses a variable in an awkward manner. It is only used in order to ensure that we recompute the inner expression *as many times as* the number of vertices in the graph. This suggests that we may still gain in expressive power by enriching weighted expressions with a restricted version of variables which would permit such an iteration. As an alternative, and following a terminology of

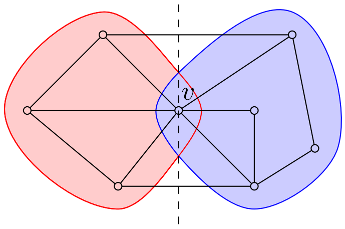


Figure 3.2: Chop decomposition of a graph associated to vertex v

[LPS08], we propose to add a *chop* operation to the syntax of weighted expressions: this basically mimics the sequentialization $\cdot$, but by *chopping* the structure at the position where the sequence is performed. This new operator has also the flavor of the subtree relativisation operator W introduced in [tCS10].

We start by defining what is a chop in a graph. Let $G = (V, E, \lambda, \chi, v^{(i)}, v^{(f)})$ be an ordered pointed graph, with $\leq$ the total linear order on V. Given $v \in V$, the chop decomposition of G associated to v is the pair of ordered pointed graphs (G_1, G_2) such that

$$G_1 = (V_1, (E_d \cap (V_1 \times V_1))_{d\in D}, \lambda_{|V_1}, v^{(i)}, v)$$
$$G_2 = (V_2, (E_d \cap (V_2 \times V_2))_{d\in D}, \lambda_{|V_2}, v, v^{(f)})$$

where $V_1 = \{v' \in V \mid v' \leq v\}$ and $V_2 = \{v' \in V \mid v \leq v'\}$. We denote again $\leq$ the total linear order on V_1 and V_2.

Definition 3.12. We let $\mathrm{chopWE}(\mathbb{S}, A, D, \mathrm{Var})$ (or shortly chopWE) be the set of *chop weighted expressions* defined by the following grammar:

$$E ::= s \mid \alpha \mid d \mid E + E \mid E \cdot E \mid E \dagger E \mid E^+$$

where $s \in \mathbb{S}$, $\alpha \in \mathrm{Test}(A, D, \mathrm{Var})$ and $d \in D$. ■

We define the semantics of such expressions in the same way as for hybrid weighted expressions (in particular, we may use free variables, even though it is not possible to place fresh variables anymore). We only complete Table 3.2 with the semantics of the new chop operator:

$$[\![E_1 \dagger E_2]\!](G, \sigma, v, v') = \begin{cases} [\![E_1]\!](G_1, \sigma, v^{(i)}, v) \otimes [\![E_2]\!](G_2, \sigma, v, v^{(f)}) & \text{if } v = v' \\ 0 & \text{otherwise} \end{cases}$$

where (G_1, G_2) denotes the chop decomposition of G associated to v.

For example, over words of $\mathcal{W}\mathrm{ord}(A)$, the hybrid weighted expression in the proof of Proposition 3.11 is equivalent to the chop weighted expression

$$\Big(\big(2 \cdot (2 \cdot \rightarrow)^\star \dagger (2 \cdot \rightarrow)^\star\big) \cdot \rightarrow\Big)^+$$

Without much surprise, variables permit to simulate chop operations over ordered pointed graphs.

Theorem 3.13. *For every $E \in \mathrm{chopWE}$, there exists $E' \in \mathrm{HWE}$ with $\mathrm{Free}(E') = \mathrm{Free}(E)$ equivalent to E, i.e., such that for all ordered pointed graph $G \in \mathcal{G}(A, D)$, $v, v' \in V$ and $\sigma \colon \mathrm{Var} \rightharpoonup V$ with domain containing $\mathrm{Free}(E)$, we have*

$$[\![E]\!](G, \sigma, v, v') = [\![E']\!](G, \sigma, v, v')\,.$$

Proof. We proceed by induction on the syntax of $E \in \text{chopWE}$. Once for all, we fix two variables x and y of Var that do not appear in $\text{Free}(E)$. We also denote $\to$ the direction in D that generates the total linear order $\leq$ over V.

In case E is of the form s, α, d, $E_1 + E_2$, $E_1 \cdot E_2$ or E_1^+, we can directly conclude by induction, as these constructs are allowed in HWE.

The interesting case is when $E = E_1 \dagger E_2$. By induction, let E_1' and E_2' be the expressions in HWE equivalent to E_1 and E_2. We encode the chop vertex (i.e., the vertex where E will be evaluated) with variable x: as E_1' must go from the initial vertex to the chop vertex and E_2' from the chop vertex to the final vertex, we will consider expression

$$x!(E_1' \cdot x? \cdot E_2')\,.$$

However, this expression is not equivalent to E in general. Indeed, the crucial point differentiating expressions in HWE and chop expressions is that E_1 and E_2 are evaluated on *subgraphs* G_1 and G_2 of the original graph G, whereas hybrid weighted expressions E_1' and E_2' are evaluated on the same graph G. In particular, if expression E_1 performs a test $d?$ or a move d with $d \in D$, it may fail in the subgraph G_1 on which it is evaluated, even though it would have succeeded in the whole graph G: this is the case if the particular edge labeled with d of the whole graph has been *chopped*. Hence, we have to slightly modify E_1' and E_2' before combining them to get expression E'. This is done by first replacing every test $d?$ over a direction $d \in D_\to$ appearing in E_1' by the concatenation $d \cdot d^{-1}$, and, similarly, every test $d?$ over a direction $d \in D_\leftarrow$ appearing in E_2' by $d \cdot d^{-1}$. Then, we replace every move d with $d \in D_\to$, appearing in E_1' by

$$d \cdot y!(\to^\star \cdot y? \cdot \to^\star \cdot x? \cdot \to^\star)$$

and, similarly, every move d with $d \in D_\leftarrow$, appearing in E_2' by

$$d \cdot y!(\to^\star \cdot x? \cdot \to^\star \cdot y? \cdot \to^\star)\,.$$

We denote E_1'' and E_2'' the expressions obtained after making these transformations. We let

$$E' = x!(E_1'' \cdot x? \cdot E_2'')\,.$$

Consider now an ordered pointed graph G, a vertex $v \in V$ and a valuation $\sigma\colon \text{Var} \rightharpoonup V$. Denoting by (G_1, G_2) the chop decomposition of G associated to v, we easily show by induction over E_1 and E_2 that

$$[\![E_1]\!](G_1, \sigma, v^{(i)}, v) = [\![E_1'']\!](G, \sigma[x \mapsto v], v^{(i)}, v)$$
$$[\![E_2]\!](G_2, \sigma, v, v^{(f)}) = [\![E_2'']\!](G, \sigma[x \mapsto v], v, v^{(f)})$$

By definition of the semantics of the chop operator from one side, and the $x!-$ and $\cdot$ operators from the other side, we then obtain:

$$[\![E]\!](G, \sigma, v, v) = [\![E']\!](G, \sigma, v, v)\,. \qquad \square$$

We let as future work the extension of this result to other classes of graphs (not necessarily ordered). An interesting question is the reciprocal of this proposition, i.e., whether the chop operation is strong enough to simulate the pebble features. Notice that this reciprocal has a flavor of separation theorem, in the sense of [HR05, Mar04].

3.3 Algebraic Semantics of Hybrid Weighted Expressions

In this last section, we propose to redefine the semantics of hybrid weighted expressions in a more algrebraic way. This will be particularly useful when proving Kleene-Schützenberger theorems in Chapter 4. To introduce this algebraic semantics, we first consider the semantics of usual weighted regular expressions over finite words[1], i.e., expressions defined by the grammar

$$E ::= sa \mid E + E \mid E \cdot E \mid E^+$$

where $s \in \mathbb{S}$ and $a \in A$. Notice that sa is now considered as a basic expression (and not the sequence of weight s and letter a) in order to define only proper expressions, and hence avoid difficulties in the definition of the semantics of the Kleene iteration. We may hence consider general – not necessarily continuous – semirings $\mathbb{S}$.

The semantics of an expression E is now a mapping from words in A^+ to $\mathbb{S}$: in particular, no need for valuations or initial and final vertices to define the semantics as we will see. Hence, we consider the semantics of an expression E as a formal power series $|E| \in \mathbb{S}\langle\langle A^+\rangle\rangle$ over the words of A^+. As $A^\star$ is a graded monoid, $(\mathbb{S}\langle\langle A^\star\rangle\rangle, \oplus, \otimes, 0, 1\varepsilon)$ is a semiring, where $\otimes$ is the Cauchy product of two series. This permits to define trivially the semantics by lifting the operations of the semiring[2]:

$$|sa| = sa \qquad |E_1 + E_2| = |E_1| \oplus |E_2| \qquad |E_1 \cdot E_2| = |E_1| \otimes |E_2|$$

It only remains to deal with Kleene iteration. Indeed, we may define the Kleene iteration of a series $f \in \mathbb{S}\langle\langle A^\star\rangle\rangle$, in case this one is a *proper* series, i.e., if $f(\varepsilon) = 0$. In fact, the *Kleene iteration* of a proper formal power series $f \in \mathbb{S}\langle\langle A^\star\rangle\rangle$ is defined by $f^+(\varepsilon) = 0$ and for every word $w \in A^+$ by

$$f^+(w) = \bigoplus_{n\geq 1} \bigoplus_{\substack{w_1,w_2,\dots,w_n \in A^+\\ w=w_1w_2\cdots w_n}} f(w_1) \otimes f(w_2) \otimes \cdots \otimes f(w_n)$$

which is well-defined as the inner sum is empty for every n greater than the length of w. As the semantics $|E|$ of any expression is proper, as a formal power series of $\mathbb{S}\langle\langle A^+\rangle\rangle$, this permits to define the semantics of the Kleene iteration of an expression E by:

$$|E^+| = |E|^+$$

Indeed, this definition of the semantics of a weighted regular expression by series operations is the way used by Schützenberger to define *regular formal power series* – the weighted equivalent of regular languages – namely the smallest class of series which contains the series sa with $s \in \mathbb{S}$ and $a \in A$, and which is closed by addition, Cauchy product and Kleene iteration of proper series. Weighted regular expressions are just a way to *denote* such regular series.

Remark 3.14. In that regular case, it is also possible, as explained in [Sak09, Chapter III.5], to remove the graded condition of the monoid, at the price of ensuring that the semiring is continuous. We do not present this generalization/restriction here in details, but we will indeed follow the same path in our case. ■

We now apply this elegant and powerful way of defining the semantics of expressions to our hybrid weighted expressions. Considering the navigation feature, and the presence of variables, this will be more complex than for the weighted regular languages. In particular, this requires to find a structure – if possible a monoid, but indeed we will not achieve this final goal – on which we will build formal power series, and the underlying operations. We fix A a finite alphabet and D a finite set of directions. Objects of the structure will be marked graphs. A *marked graph* is of the form (G, σ, v, v') where $G \in \mathcal{G}(A, D)$ is a pointed

[1] We will indeed focus on *proper* weighted regular expressions as it is enough for our purpose.

[2] We recall that sw denotes the formal power series whose only non-zero coefficient is the one of $w \in A^\star$, which is equal to $s \in \mathbb{S}$.

graph, $\sigma\colon \mathrm{Var} \rightharpoonup V$ is a valuation with a finite domain, and $v, v' \in V$ are vertices. These are exactly the structures we use to define the semantics of hybrid weighted expressions. Let $\mathcal{MG}(A, D, \mathrm{Var})$ (or shortly $\mathcal{MG}$ if the parameters are implicit) be the set of all marked graphs.

We may equip the set of marked graphs with a product operation. Given $(G_1, \sigma_1, v_1, v_1')$ and $(G_2, \sigma_2, v_2, v_2')$ two marked graphs, the *product* of these marked graphs, denoted by $(G_1, \sigma_1, v_1, v_1') \circ (G_2, \sigma_2, v_2, v_2')$, is defined whenever $G_1 = G_2$, $\sigma_1 = \sigma_2$ and $v_1' = v_2$, by

$$(G_1, \sigma_1, v_1, v_1') \circ (G_1, \sigma_1, v_1', v_2') = (G_1, \sigma_1, v_1, v_2')$$

The product operation defined that way is a partial binary operation. It admits several partial units: indeed we have

$$(G_1, \sigma_1, v_1, v_1') \circ (G_1, \sigma_1, v_1', v_2') = (G_1, \sigma_1, v_1, v_1') \text{ if, and only if, } v_2' = v_1'$$

A marked graph of the form (G, σ, v, v) is henceforth called a *partial unit*. The set of all partial units is denoted by $\mathcal{U}(A, D, \mathrm{Var})$ (or shortly $\mathcal{U}$ if the parameters are implicit).

We have now defined an algebraic structure over the set of marked graphs, not as neat as a monoid structure, but this will be sufficient. Before coming back to the algebraic definition of the semantics of hybrid weighted expressions, we explore the properties of this new structure, that we call a partial monoid.

3.3.1 Partial Monoids

A *partial monoid* is a set $\mathbb{M}$ equipped with a *partial* binary operation $\circ$ and a subset of elements called *partial units*, and denoted U such that:

- $\circ$ is partially associative, i.e., $m \circ (m' \circ m'')$ is defined if, and only if, $(m \circ m') \circ m''$ is defined, and in that case, $m \circ (m' \circ m'') = (m \circ m') \circ m''$ for all elements m, m', m'' of $\mathbb{M}$;
- every partial unit $u \in U$ is a partial neutral element of $\circ$, i.e., if $m \circ u$ (respectively, $u \circ m$) is defined then $m \circ u = m$ (respectively, $u \circ m = m$) for all element m of $\mathbb{M}$;
- every element admits a unique left partial unit, and a unique right partial unit, i.e., for all $m \in \mathbb{M}$, there is a unique $u \in U$ such that $m \circ u$ is defined, and also, there is a unique $u \in U$ such that $u \circ m$ is defined.

Ideally, a monoid is a partial monoid with the binary operation being totally defined and with the set of partial units being the singleton set consisting of the neutral element.

Example 3.15. $(\mathcal{MG}(A, D, \mathrm{Var}), \circ, \mathcal{U}(A, D, \mathrm{Var}))$ is a partial monoid. Clearly, $\circ$ is partially associative and every partial unit is a partial neutral element of it. Moreover, for every marked graph (G, σ, v, v') and every partial unit (G', σ', v'', v''), their product $(G, \sigma, v, v') \circ (G', \sigma', v'', v'')$ is defined if, and only if, $G = G'$, $\sigma = \sigma'$ and $v' = v''$, which defines uniquely the partial unit. Hence, every element admits a unique left partial unit, and a unique right partial unit. ■

Consider now a continuous semiring $(\mathbb{S}, \oplus, \otimes, 0, 1)$. The set $\mathbb{S}\langle\langle\mathbb{M}\rangle\rangle$ of formal power series over the partial monoid $\mathbb{M}$ is equipped with an addition operation (this does not require any operation over $\mathbb{M}$ indeed), and we may also define partial Cauchy products, basically with the same definition as the Cauchy product when $\mathbb{M}$ is a monoid. Formally, the *partial Cauchy product* $f \otimes g$ of two series f and g of $\mathbb{S}\langle\langle\mathbb{M}\rangle\rangle$ is defined by

$$(f \otimes g)(m) = \bigoplus_{m = m' \circ m''} f(m') \otimes g(m'') \qquad \text{for all } m \in \mathbb{M}\,.$$

The sum now ranges over all pairs (m', m'') for which the product $m' \circ m''$ is defined and such that $m = m' \circ m''$. Notice that this sum may be infinite (but is well-defined since the semiring is continuous) but is necessarily non empty because of the existence of the left and right units of m. Interestingly, this Cauchy product permits to equip $\mathbb{S}\langle\langle\mathbb{M}\rangle\rangle$ with a structure of semiring, which is indeed continuous.

Proposition 3.16. *If* $(\mathbb{M}, \circ, U)$ *is a partial monoid, and* $(\mathbb{S}, \oplus, \otimes, 0, 1)$ *a continuous semiring, then* $(\mathbb{S}\langle\langle\mathbb{M}\rangle\rangle, \oplus, \otimes, 0, \mathbb{1}_U)$ *is a continuous semiring*[3].

Proof. We detail below the different elements proving that $(\mathbb{S}\langle\langle\mathbb{M}\rangle\rangle, \oplus, \otimes, 0, \mathbb{1}_U)$ is a continuous semiring.

- For every set $\mathbb{M}$, $(\mathbb{S}\langle\langle\mathbb{M}\rangle\rangle, \oplus, 0)$ is a commutative monoid.
- The novelty is to prove that $(\mathbb{S}\langle\langle\mathbb{M}\rangle\rangle, \otimes, \mathbb{1}_U)$ is a monoid. Indeed, $\otimes$ is easily shown to be an associative operation, by following the same proof as for monoids. Moreover, $\mathbb{1}_U$ is a neutral element for $\otimes$. As a matter of fact, we have for every $m \in \mathbb{M}$:

$$\begin{aligned}(f \otimes \mathbb{1}_U)(m) &= \bigoplus_{m = m' \circ m''} f(m') \otimes \mathbb{1}_U(m'') \\ &= \sum_{\substack{m' \in \mathbb{M}, m'' \in U \\ m = m' \circ m''}} f(m') \\ &= \sum_{\substack{m'' \in U \\ m = m \circ m''}} f(m) \\ &= f(m)\end{aligned}$$

  where the last equality follows from the existence and unicity of the right partial unit of m. A symmetric proof may be used to prove that $\mathbb{1}_U \otimes f = f$.
- $\otimes$ distributes over $\oplus$, and 0 is an annihilator by lifting these properties from the semiring $\mathbb{S}$.
- Sum of family $(f_i)_{i \in I}$ is given, for every $m \in \mathbb{M}$, by

$$\big(\sum_{i \in I} f_i\big)(m) = \sum_{i \in I} f_i(m)\,.$$

  From this pointwise definition of the summation, it is not difficult to lift completeness and continuity from the semiring $\mathbb{S}$ to $\mathbb{S}\langle\langle\mathbb{M}\rangle\rangle$.

□

Remark 3.17. The previous proposition permits to conclude that the structure

$$(\mathbb{S}\langle\langle\mathcal{MG}(A, D, \mathrm{Var})\rangle\rangle, \oplus, \otimes, 0, \mathbb{1}_{\mathcal{U}(A,D,\mathrm{Var})})$$

is a continuous semiring. ■

As in every continuous semiring, the Kleene iteration $f^\star$ of every series $f \in \mathbb{S}\langle\langle\mathbb{M}\rangle\rangle$ is well-defined. We may use in the following the notation f^+ as the partial Cauchy product $f \otimes f^\star$.

3.3.2 Application to the Semantics of Hybrid Weighted Expressions

We now come back to the main purpose of defining algebraically the semantics of hybrid weighted expressions. To every hybrid weighted expression $E \in \mathrm{HWE}$ will be associated a series $|E| \in \mathbb{S}\langle\langle\mathcal{MG}\rangle\rangle$, inductively on the structure of E.

[3] The neutral element for the Cauchy product, $\mathbb{1}_U$, is the characteristic series of the partial units of $\mathbb{M}$, defined by $\mathbb{1}_U(m) = 1$ if $m \in U$, and 0 otherwise.

The semantics of $E = s$ (with $s \in \mathbb{S}$) is defined by

$$|s| = s\mathbb{1}_{\mathcal{U}}$$

the series which associates s to the partial units and 0 otherwise. Similarly, the semantics of $E = \alpha$ (with $\alpha \in \text{Test}$) is defined by

$$|\alpha| = \mathbb{1}_{\{(G,\sigma,v,v)\in\mathcal{U} \mid G,\sigma,v\models\alpha\}}$$

For every direction $d \in D$, we finally define

$$|d| = \mathbb{1}_{\{(G,\sigma,v,v')\in\mathcal{MG} \mid (v,v')\in E_d\}}$$

These three base cases are not surprising and resemble the previous definition of the semantics. But, indeed, the hardest is done as the inductive cases are trivial, in the sense that they are nothing more than a rewriting of the formulae used for weighted regular expressions:

$$|E_1 + E_2| = |E_1| \oplus |E_2| \qquad |E_1 \cdot E_2| = |E_1| \otimes |E_2| \qquad |E^+| = |E|^+$$

It only remains to deal with the semantics of $E = x!E_1$. Indeed, this can be seen as a projection operation over the series, but is inductively defined by the same equation as previously:

$$|x!E_1|(G,\sigma,v,v') = \begin{cases} |E_1|(G,\sigma[x\mapsto v],v^{(i)},v^{(f)}) & \text{if } v = v' \\ 0 & \text{otherwise} \end{cases}$$

Proposition 3.18. *For every* $E \in \text{HWE}$*, and for every marked graph* $(G,\sigma,v,v') \in \mathcal{MG}$ *with* $\text{Free}(E) \subseteq \text{dom}(\sigma)$*,* $[\![E]\!](G,\sigma,v,v') = |E|(G,\sigma,v,v')$*.*

Proof. We proceed by induction over E. The equality is trivially verified for expressions s, α, d and $x!E$.

If $E = E_1 + E_2$, we have by induction

$$\begin{aligned}[\![E_1 + E_2]\!](G,\sigma,v,v') &= [\![E_1]\!](G,\sigma,v,v') \oplus [\![E_2]\!](G,\sigma,v,v') \\ &= |E_1|(G,\sigma,v,v') \oplus |E_2|(G,\sigma,v,v') \\ &= |E_1 + E_2|(G,\sigma,v,v')\end{aligned}$$

as the addition of series is defined pointwisely.

If $E = E_1 \cdot E_2$, we have

$$\begin{aligned}[\![E_1 \cdot E_2]\!](G,\sigma,v,v') &= \bigoplus_{v''\in V} [\![E_1]\!](G,\sigma,v,v'') \otimes [\![E_2]\!](G,\sigma,v'',v') \\ &= \bigoplus_{v''\in V} |E_1|(G,\sigma,v,v'') \otimes |E_2|(G,\sigma,v'',v') \\ &= \bigoplus_{\substack{(G,\sigma,v,v')= \\ (G_1,\sigma_1,v_1,v_1')\circ \\ (G_2,\sigma_2,v_2,v_2')}} |E_1|(G_1,\sigma_1,v_1,v_1') \otimes |E_2|(G_2,\sigma_2,v_2,v_2') \\ &= (|E_1| \otimes |E_2|)(G,\sigma,v,v') \\ &= |E_1 \cdot E_2|(G,\sigma,v,v')\end{aligned}$$

Finally, if $E = E_1^+$, we have

$$[\![E_1^+]\!](G,\sigma,v,v') = \bigoplus_{n>0} [\![E_1^n]\!](G,\sigma,v,v')$$

An induction over n shows that

$$[\![E_1^n]\!](G,\sigma,v,v') = |E_1|^n(G,\sigma,v,v')$$

with f^n defined for every series f by $f^1 = f$ and $f^{n+1} = f \otimes f^n$. By definition of the Kleene iteration of a series, we conclude that

$$[\![E_1^+]\!](G,\sigma,v,v') = |E_1|^+(G,\sigma,v,v') = |E_1^+|(G,\sigma,v,v')$$

□

CHAPTER 4

Navigating Automata

> Le petit Poucet les laissait crier, sachant bien par où il reviendrait à la maison, car en marchant il avait laissé tomber le long du chemin les petits cailloux blancs qu'il avait dans ses poches.[1]
>
> Charles Perrault, *Le Petit Poucet*

In this chapter, we explore a more operational view of quantitative specification, in terms of weighted automata. Originally investigated by Marcel-Paul Schützenberger in [Sch61] as a way to compute rational series, they have been extended and studied extensively since then. In particular, much has been done to transfer Kleene's result, stating the equivalence of regular expressions and finite-state automata. This result, extended to the weighted case in Schützenberger theorem, is not only of theoretical interest, but is also practically important: in particular, transforming a denotational specification into an automaton must be done as fast as possible. Classical fast algorithms in the Boolean case are based on Glushkov constructions (see [Glu60, Glu61]), further investigated, e.g., in Berry-Sethi algorithm (see [BS86]). These algorithms have been extended into the weighted case (see [Sak09]): a unified version of several algorithms is studied in [AM06].

Before presenting our framework of weighted automata navigating over graphs, we start by recalling standard definitions of weighted automata (over words). Usually, a weighted automaton over an alphabet A and semiring $\mathbb{S}$ is a tuple (Q, λ, μ, ν) with Q a finite set of states, $\lambda \in \mathbb{S}^Q$ a row vector of initial weights, $\mu\colon A \to \mathbb{S}^{Q\times Q}$ a mapping from letters to matrices of transitions, and $\nu \in \mathbb{S}^Q$ a column vector of final weights. For example, over the

[1] "Little Thumb let them cry on, knowing very well how to get home again, for, as he came, he took care to drop all along the way the little white pebbles he had in his pockets."

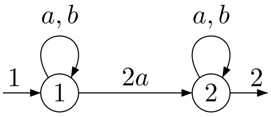


Figure 4.1: A first weighted automaton

alphabet $\{a, b\}$ and the semiring $(\mathbb{N}, +, \times, 0, 1)$, we may consider the weighted automaton defined by $Q = \{1, 2\}$, $\lambda = \begin{pmatrix}1 & 0\end{pmatrix}$, $\mu(a) = \begin{pmatrix}1 & 2\\0 & 1\end{pmatrix}$, $\mu(b) = \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$ and $\nu = \begin{pmatrix}0\\2\end{pmatrix}$. It is depicted in Figure 4.1.

Its semantics may be defined using runs, having a certain weight: this is the way we will define the semantics of our automata. However, in this simple example, the semantics may be defined in a more algebraic and direct way. Indeed, notice that the mapping μ may be uniquely extended as a morphism from the free monoid $A^\star$ (i.e., the monoid generated by elements of A) to the monoid of square matrices $\mathbb{S}^{Q\times Q}$. This is done by letting $\mu(\varepsilon)$ being the identity matrix, and for every word w, and letter $a \in A$ by letting $\mu(aw) = \mu(a) \times \mu(w)$ where × denotes here the matrix product. Then, the semantics of a weighted automaton $\mathcal{A}$ is the formal power series obtained by associating to every word $w \in A^\star$ the coefficient

$$|\mathcal{A}|(w) = \lambda \times \mu(w) \times \nu \in \mathbb{S}\,.$$

For example, the weighted automaton $\mathcal{A}$ of Figure 4.1 generates the formal power series mapping every word w to the coefficient $2(|w|_a + 1)$. Indeed, as $\mu(b)$ is the identity matrix, we have $\mu(w) = \mu(a^{|w|_a})$, and an induction shows that

$$\mu(a^n) = \begin{pmatrix}1 & 2n\\0 & 1\end{pmatrix}$$

A computation, shown in our setting in Theorem 4.18, shows that weighted automata over the natural semiring may only recognize formal power series f such that $f(w) = 2^{O(|w|)}$, bounding immediately their expressive power. Our starting point is then to explore more expressive automata: this will be done by adding navigational features, as well as pebbles.

Whereas the first section investigates the navigational issue, and states a first Kleene-Schützenberger theorem, the second one will add pebble features and prove an extended theorem relating these automata with the expressions defined in the previous chapter. This chapter is an extension of some of the results presented in [2] in the special case of words, and [4] in the special case of nested words.

4.1 Weighted Automata over Graphs

4.1.1 Syntax and Semantics over a Continuous Semiring

In this section, we define weighted automata navigating over graphs. We will simply call them weighted automata, even though they are not the same objects as weighted automata we can find in the literature, and that we have described previously. We will see however that they naturally contain weighted automata over words.

Definition 4.1. A *weighted automaton* over a semiring $\mathbb{S}$ is a tuple $\mathcal{A} = (Q, A, D, I, \Delta, F)$ where

- Q is a finite set of states;
- A is a finite alphabet;
- D is a finite set of directions;
- $I \in \mathbb{S}^Q$ is the vector of initial weights;

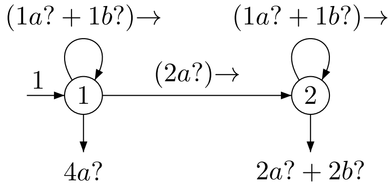


Figure 4.2: Weighted automaton $\mathcal{A}'$

- $\Delta \in \mathbb{S}\langle\text{Test}(A, D, \text{Var})\rangle\langle D\rangle^{Q\times Q}$ is the weighted transition matrix[2]: a transition goes from a state q to a state q' performing a test α and an action given by a direction d, and has weight $\Delta_{q,q'}(d)(\alpha)$;
- $F \in \mathbb{S}\langle\text{Test}(A, D, \text{Var})\rangle^{Q}$ is the final vector: each state q is mapped to a polynomial F_q of tests.

The class of all weighted automata over $\mathbb{S}$ is denoted by WA($\mathbb{S}$) or WA if the semiring is clear from the context, or not relevant. ■

Example 4.2. We first consider the case of words in this example, i.e., we suppose that $D = \{\leftarrow, \rightarrow\}$. Let $\mathcal{A}' = (\{1, 2\}, \{a, b\}, D, I, \Delta, F)$ be the weighted automaton in WA($\mathbb{N}$) (over the semiring $(\mathbb{N}, +, \times, 0, 1)$) defined by

$$I = \begin{pmatrix} 1 & 0 \end{pmatrix} \qquad \Delta = \begin{pmatrix} (1a? + 1b?)\rightarrow & (2a?)\rightarrow \\ 0 & (1a? + 1b?)\rightarrow \end{pmatrix} \qquad F = \begin{pmatrix} 4a? \\ 2a? + 2b? \end{pmatrix}$$

Automaton $\mathcal{A}'$ is depicted in Figure 4.2. Notice that $(1a? + 1b?)\rightarrow$ denotes the formal power series $f \in \mathbb{S}\langle\text{Test}\rangle\langle D\rangle$ with $f(\leftarrow) = 0$ and $f(\rightarrow)$ being the formal power series of $\mathbb{S}\langle\text{Test}\rangle$ having for only non-zero coefficients the one of tests $a?$ and $b?$ which are both equal to 1. ■

Intuitively, a weighted automaton navigates in a pointed graph, keeping in its finite memory the current state. At every time step, the automaton chooses an available direction of D and follows it, under the condition that some test of Test(A, D, Var) is verified on the current vertex of the graph. Notice that we allow the presence of some tests $x?$ of a variable x: this will become useful in the following of this chapter. Henceforth, to be evaluated, the weighted automaton $\mathcal{A}$ also needs a valuation of the free variables Free($\mathcal{A}$), namely the (finite) set of variables appearing in tests of non-zero coefficients of the polynomials in Δ and F.

A run collects a sequence of moves from one vertex to another, and the weight of a run is the *product* (in the semiring $\mathbb{S}$) of the weights of the transitions taken during the run. The non-determinism is resolved by using the *sum* of the semiring, i.e., a weighted automaton maps every possible pointed graph to the sum of the weights of the *accepted* runs over this graph, namely those that start in the initial vertex and end in the final one. Now, we formalize this definition of the semantics.

A run will be described as a sequence of configurations of $\mathcal{A}$. A *configuration* of $\mathcal{A} = (Q, A, D, I, \Delta, F)$ is an element (G, σ, q, v) composed of a pointed graph $G \in \mathcal{G}(A, D)$, a valuation $\sigma\colon \text{Var} \rightharpoonup V$, a state q of Q and a vertex v of V. Any two configurations with a fixed graph G and a fixed valuation σ give rise to a *concrete transition* $(G, \sigma, q, v) \rightsquigarrow (G, \sigma, q', v')$ whenever the pair (v, v') is an edge in E. Its *weight* is then defined by

$$\bigoplus_{\substack{d\in D \mid (v,v')\in E_d \\ \alpha\in\text{Test} \mid G,\sigma,v\models\alpha}} \Delta_{q,q'}(d)(\alpha)\,. \tag{4.1}$$

[2] We recall that Test(A, D, Var) is the set of test formulae defined in Chapter 3, sometimes called Test in the following.

Remark 4.3. Notice that this sum is finite because $\Delta_{q,q'}(d)$ is a polynomial for every $d \in D$. ■

A *run* of $\mathcal{A}$ is a sequence of configurations related successively by concrete transitions. Its *weight* is the product of transition weights, multiplied from left to right. We are interested in runs that start at some vertex v, in state q, and end in some configuration with vertex v' and state q'. Therefore, we let $[\![\mathcal{A}_{q,q'}]\!](G,\sigma,v,v')$ be the sum of the weights of all runs leading from configuration (G,σ,q,v) to configuration (G,σ,q',v'). If we consider a continuous semiring, this *a priori* infinite sum is well-defined.

Finally, the semantics of weighted automaton $\mathcal{A}$ maps every pointed graph $G \in \mathcal{G}(A,D)$ and valuation $\sigma\colon \mathrm{Var} \rightharpoonup V$ with domain containing the free variables of $\mathcal{A}$ to an element of the continuous semiring $\mathbb{S}$ by including the initial and final weights, letting

$$[\![\mathcal{A}]\!](G,\sigma) = \bigoplus_{q,q' \in Q} I_q \otimes [\![\mathcal{A}_{q,q'}]\!](G,\sigma,v^{(i)},v^{(f)}) \otimes \bigoplus_{\substack{\alpha \in \mathrm{Test} \mid \\ G,\sigma,v^{(f)} \models \alpha}} F_{q'}(\alpha)\,.$$

As for the hybrid weighted expressions, we may write $[\![\mathcal{A}]\!](G)$ and $[\![\mathcal{A}_{q,q'}]\!](G,v,v')$ in the case where $\mathcal{A}$ has no free variables.

Example 4.4 (Continued from Example 4.2)**.** We consider a pointed graph $G \in \mathcal{W}\mathrm{ord}(\{a,b\})$ representing a finite word $w \in \{a,b\}^+$. The weighted automaton $\mathcal{A}'$ has no free variables so that we can forget about the valuation part in the configurations and in the semantics. A run with non-zero weight in $\mathcal{A}'$ from a configuration $(G,1,v^{(i)})$ to $(G,q,v^{(f)})$ (with $q \in \{1,2\}$) visits first only state 1, and then either stops in state 1, or uses the transition from state 1 to state 2 and then loops in state 2. If it stops in state 1, to accept, and get a non-zero weight, the final test $a?$ must be verified, in which case weight 4 is multiplied. This simulates the behavior of the (classical) weighted automaton $\mathcal{A}$ of Figure 4.1 where the transition from state 1 to state 2 – with weight 2 – is taken *in the last position of the word*, before accepting with weight 2. Otherwise, the run in $\mathcal{A}'$ takes the transition from state 1 to state 2, simulating the same behavior in $\mathcal{A}$. It finally loops in state 2 before accepting with weight 2 in the final vertex, since $2a? + 2b?$ is a polynomial which evaluates as the constant polynomial 2. Finally, this shows that $[\![\mathcal{A}']\!](G) = |\mathcal{A}|(w)$. ■

Remark 4.5. Notice that the previous example indeed shows that we can translate every (classical) weighted automaton over finite words in our navigating formalism. We must particularly care about the vector of final weights: indeed, the last transition of a (classical) weighted automaton leads from a configuration in the last vertex of a word to the outside of this word. In contrast, it is forbidden in our setting to exit the set of vertices of a graph. Henceforth, we simulate the last transition by the vector of final weights, by using the power of test formulae in Test. For example, in automaton $\mathcal{A}$ of Figure 4.1, if a run ends with the looping transition on state 2, it is simulated in $\mathcal{A}'$ by testing $a? + b?$ (without making the $\rightarrow$ move which would informally exit the set of vertices) and multiplying by the final weight 2 of state 2. ■

Remark 4.6. In the sequel, we will continuously use terminologies borrowed from the usual Boolean setting. For example, we may talk about *initial state* (respectively, *final state*) for a state having an initial weight (respectively, final polynomial) different from 0. By extension, an *accepting run* is a run which starts in a configuration $(G,\sigma,q,v^{(i)})$ with q an initial state, and ends in a configuration $(G,\sigma,q',v^{(f)})$ with q' a final state, and having a weight different from 0. ■

Example 4.7. Consider the weighted automaton $\mathcal{A} \in \mathrm{WA}(\mathbb{R}^+ \cup \{+\infty\})$ (over the semiring $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$) depicted in Figure 4.3, where p represents a probability in $[0,1]$, and over alphabet $A = \{a\}$. It still runs over finite words, now using the capability to move in two directions. Consider such a word $w = a_0 \cdots a_{n-1}$ of length n, encoded in a graph

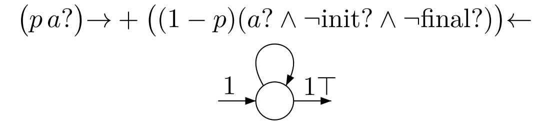

$\big(p\,a?\big)\!\rightarrow + \big((1-p)(a? \wedge \neg\text{init}? \wedge \neg\text{final}?)\big)\!\leftarrow$
1
$1\top$

Figure 4.3: A weighted automaton $\mathcal{A} \in \mathrm{WA}(\mathbb{R}^+ \cup \{+\infty\})$

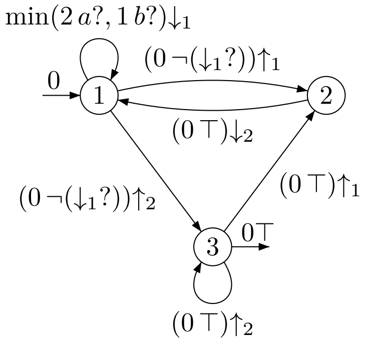

$\min(2\,a?, 1\,b?)\!\downarrow_1$
0
1
$(0\,\neg(\downarrow_1?))\!\uparrow_1$
2
$(0\,\top)\!\downarrow_2$
$(0\,\neg(\downarrow_1?))\!\uparrow_2$
$(0\,\top)\!\uparrow_1$
3
$0\top$
$(0\,\top)\!\uparrow_2$

Figure 4.4: Weighted automaton $\mathcal{A}$ mimicking a depth-first-search of a binary tree

$G \in \mathcal{W}\mathrm{ord}(\{a\})$. Once in the final vertex of the graph, the automaton is forced to accept with weight 1, since neither the → transition, nor the ← transition – because of the test ¬final? – can be taken. On the other hand, the test ¬init? in the transition is there to ensure that direction ← is not taken on the first position of the word: notice however, that the semantics of the automaton would not have been different if this test was removed. Indeed, there is a bijection between the *accepting runs* of $\mathcal{A}$, and the paths in G from the initial vertex to the final vertex, that stop the first time they reach this final vertex. The weight of an accepting run is then $p^{m_\rightarrow}(1-p)^{m_\leftarrow}$ where $m_\rightarrow$ (respectively, $m_\leftarrow$) is the number of → directions (respectively, ← directions) taken along the run: considering the length n of the word, we must have $m_\rightarrow = m_\leftarrow + n - 1$. Notice that there is an infinite number of accepting runs, which means that the semantics of $\mathcal{A}$ involves an infinite sum, making it crucial to consider a complete semiring. This example is further studied in Example 8.4, giving a probabilistic interpretation of the semantics of this automaton. ■

Example 4.8. We consider now an example over binary trees. Let $A = \{a, b, c\}$ with a and b being binary letters, and c a nullary letter. Hence, we will consider binary trees where a and b are labels of internal nodes and c the label of the leaves. Let $\mathcal{A}$ be the weighted automaton in $\mathrm{WA}(\mathbb{N} \cup \{+\infty\})$ (over the semiring $(\mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0)$) depicted in Figure 4.4. Over each binary tree $G \in \mathcal{T}\mathrm{ree}(A)$, the automaton $\mathcal{A}$ admits a unique accepting run following a depth-first-search of the tree: it visits each leaf exactly once, in state 1, and each internal node three times, the first one in state 1, then in state 2 and the last time in state 3. During the first visit of each internal node, we count 2 if the node is labeled by a and 1 if it is labeled by b. All other transitions count 0, hence, the weight of a run, defined as the sum[3] of the weights of the transitions, is precisely $2n_a + n_b$ if n_a (respectively, n_b) is the number of internal nodes labeled by a (respectively, b). By adding to this automaton three more states (in a separated connected component) related in the same way by simply exchanging the weights 1 and 2 in the self loop, we produce an automaton with exactly two accepting runs over each binary tree: the non-determinism is then resolved by taking a minimum, and the semantics then maps a binary tree to the value $\min(2n_a+n_b, n_a+2n_b)$. ■

[3] The weight of a run is the *product* (in the semiring) of the weights of the transitions of the run: here the product operation of the semiring is an *addition*...

4.1.2 A First Kleene-Schützenberger Theorem

We are now ready to state the first result of this manuscript: its aim is to relate a denotational formalism – weighted expressions – with a computational formalism – weighted automata. From one side, translating weighted expressions into weighted automata may be seen as a prerequisite to efficient parsing of weighted expressions: therefore, we will try to make this translation as efficient as possible. From the other side, this is a theoretical result involving some closure properties of the weighted automata: this shows the robustness of the class of automata we considered.

Theorem 4.9. *Let $\mathbb{S}$ be a continuous semiring.*

1. *For each weighted expression $E \in \mathrm{WE}(\mathbb{S})$, we can construct a weighted automaton $\mathcal{A}_E \in \mathrm{WA}(\mathbb{S})$ with $\mathrm{Free}(\mathcal{A}_E) = \mathrm{Free}(E)$, equivalent to E, i.e., $[\![E]\!](G,\sigma) = [\![\mathcal{A}_E]\!](G,\sigma)$ for every pointed graph G and every valuation σ with domain containing $\mathrm{Free}(E)$.*
2. *For each weighted automaton $\mathcal{A} \in \mathrm{WA}(\mathbb{S})$, we can construct a weighted expression $E_\mathcal{A} \in \mathrm{WE}(\mathbb{S})$ with $\mathrm{Free}(E_\mathcal{A}) = \mathrm{Free}(\mathcal{A})$, equivalent to $\mathcal{A}$, i.e., $[\![\mathcal{A}]\!](G,\sigma) = [\![E_\mathcal{A}]\!](G,\sigma)$ for every pointed graph G and every valuation σ with domain containing $\mathrm{Free}(\mathcal{A})$.*

The rest of this section is devoted to the proof of this theorem. We start by proving the second item, as its proof will help us for the proof of the first one.

From Weighted Automata to Weighted Expressions

This section is devoted to the construction of a weighted expression $E_\mathcal{A}$ equivalent to a given weighted automaton $\mathcal{A}$. We will show that we can use the classical constructions to transform regular expressions into finite-state automata, e.g., the state elimination method of Brzozowski and McCluskey [BM63], the procedure of McNaughton and Yamada [MY60] or the recursive algorithm [Con71]. We refer to the survey of Sakarovitch [Sak12, Section 6.2] where these methods are presented and compared for (classical) weighted automata over finite words.

In the state elimination method, states are progressively suppressed and transitions are labeled with weighted expressions. Therefore, it is convenient to consider automata allowing weighted expressions in the labels of transitions: we first introduce these *generalized* weighted automata and then show how to exploit them to translate automata into expressions.

Definition 4.10. A *generalized weighted automaton* is a tuple $\mathcal{A} = (Q, A, D, I, \Delta, F)$ where

- Q is a finite set of states;
- A is a finite alphabet;
- D is a finite set of directions;
- $I \in \mathbb{S}^Q$ is the initial vector;
- $\Delta \in \mathrm{WE}(\mathbb{S}, A, D)^{Q\times Q}$ is the transition matrix;
- $F \in \mathbb{S}\langle \mathrm{Test}(A, D, \mathrm{Var})\rangle^Q$ is the final vector.

The class of all generalized weighted automata over $\mathbb{S}$ is denoted by gWA($\mathbb{S}$) or gWA if the semiring is clear from the context, or not relevant. ■

The differences compared with weighted automata in WA reside in the introduction of weighted expressions in the transition matrix. The semantics of such a generalized weighted automaton is given by a variation of the semantics of weighted automata: now, the weight of a concrete transition $(G, \sigma, q, v) \rightsquigarrow (G, \sigma, q', v')$ is defined by $[\![\Delta_{q,q'}]\!](G, \sigma, v, v')$ (the semantics of a weighted expression). This permits to update accordingly the definition of $[\![\mathcal{A}_{q,q'}]\!](G, \sigma, v, v')$. We introduce another step before giving the final semantics as this will be useful in the following: we denote $[\![\mathcal{A}]\!](G, \sigma, v, v')$ the semantics of the generalized automaton *from vertex v to vertex v'*, which is defined by

$$[\![\mathcal{A}]\!](G,\sigma,v,v') = \bigoplus_{q,q'\in Q} I_q \otimes [\![\mathcal{A}_{q,q'}]\!](G,\sigma,v,v') \otimes \bigoplus_{\substack{\alpha\in\mathrm{Test}\mid\\ G,\sigma,v'\models\alpha}} F_{q'}(\alpha)\,.$$

Finally, we let $[\![\mathcal{A}]\!](G,\sigma) = [\![\mathcal{A}]\!](G,\sigma,v^{(i)},v^{(f)})$ be the semantics from the initial to the final vertex of the pointed graph G with valuation σ.

Notice that every weighted automaton can be seen as a generalized weighted automaton.

Proposition 4.11. *From a weighted automaton $\mathcal{A}\in\mathrm{WA}(\mathbb{S})$, we can construct an equivalent generalized weighted automaton $\tilde{\mathcal{A}}\in\mathrm{gWA}(\mathbb{S})$ with the same set of free variables.*

Proof. Let $\mathcal{A}=(Q,A,D,I,\Delta,F)$ be a weighted automaton. We construct an equivalent generalized weighted automaton $\tilde{\mathcal{A}}=(Q,A,D,I,\tilde{\Delta},F)$ by letting for every $q,q'\in Q$,

$$\tilde{\Delta}_{q,q'} = \sum_{\alpha\in\mathrm{Test},d\in D} \Delta_{q,q'}(d)(\alpha)\cdot\alpha\cdot d$$

which is a weighted expression, since this sum is finite (Δ has finite support by definition). A simple rewriting permits to show that both definitions of the semantics coincide in that case. □

We first relate the semantics of a generalized weighted automaton with the star of the semantic matrix $[\![\Delta]\!]$, whose coefficients are given by $[\![\Delta_{q,q'}]\!]$. For each $q,q'\in Q$, the entry $\Delta_{q,q'}$ is a weighted expression in $\mathrm{WE}(\mathbb{S})$ and its semantics $[\![\Delta_{q,q'}]\!]$ may hence be seen as a series over marked graphs of $\mathcal{MG}(A,D,\mathrm{Var})$ (that we denote $\mathcal{MG}$ in the following). Hence, the semantic matrix verifies $[\![\Delta]\!] = ([\![\Delta_{q,q'}]\!])_{(q,q')\in Q^2}\in\mathbb{S}\langle\!\langle\mathcal{MG}\rangle\!\rangle^{Q\times Q}$.

Recall that $\mathbb{S}\langle\!\langle\mathcal{MG}\rangle\!\rangle$ is a continuous semiring by Remark 3.17. For each finite set Q, the semiring of matrices $\mathbb{S}\langle\!\langle\mathcal{MG}\rangle\!\rangle^{Q\times Q}$ is also continuous. Hence, given a matrix H in $\mathbb{S}\langle\!\langle\mathcal{MG}\rangle\!\rangle^{Q\times Q}$, the *star* matrix

$$H^\star = \bigoplus_{n\geq 0} H^n \in \mathbb{S}\langle\!\langle\mathcal{MG}\rangle\!\rangle^{Q\times Q}$$

is well-defined. Applying this to $H=[\![\Delta]\!]$ we can generalize the classical relation between the semantics of a one-way weighted automaton and the star of a matrix of series.

Proposition 4.12. *Let $\mathcal{A}=(Q,A,D,I,\Delta,F)$ be a weighted automaton. For all states $q,q'\in Q$, the two series $[\![\mathcal{A}_{q,q'}]\!]$ and $([\![\Delta]\!]^\star)_{q,q'}$ are equal, i.e.,*

$$[\![\mathcal{A}_{q,q'}]\!](G,\sigma,v,v') = ([\![\Delta]\!]^\star)_{q,q'}(G,\sigma,v,v')$$

for all marked graph $(G,\sigma,v,v')\in\mathcal{MG}$.

Proof. We show by induction on $n\geq 0$ that for every marked graph $(G,\sigma,v,v')\in\mathcal{MG}$, $([\![\Delta]\!]^n)_{q,q'}(G,\sigma,v,v')$ computes the sum of the weights of runs of length n from configuration (G,σ,q,v) to configuration (G,σ,q',v').

This is true for $n=0$, as $\mathbb{1}_{\mathcal{U}}(G,\sigma,v,v')=1$ if, and only if, $v=v'$. Then, assuming the property for $n-1\geq 0$, we prove it for n. A run of length $n>0$ starts with a transition followed by a run of length $n-1$. Hence, the sum of the weights of runs of length n from configuration (G,σ,q,v) to configuration (G,σ,q',v') is computed by

$$\bigoplus_{q''\in Q,v''\in V} [\![\Delta]\!]_{q,q''}(G,\sigma,v,v'')\otimes([\![\Delta]\!]^{n-1})_{q'',q'}(G,\sigma,v'',v')\,.$$

Indeed, by using the definition of the matrix multiplication induced by the Cauchy product, we have

$$\begin{aligned}(\llbracket\Delta\rrbracket^n)_{q,q'}(G,\sigma,v,v') &= (\llbracket\Delta\rrbracket \otimes \llbracket\Delta\rrbracket^{n-1})_{q,q'}(G,\sigma,v,v')\\ &= \bigoplus_{q''\in Q} \left(\llbracket\Delta\rrbracket_{q,q''} \otimes (\llbracket\Delta\rrbracket^{n-1})_{q'',q'}\right)(G,\sigma,v,v')\,.\end{aligned}$$

Then, applying the definition of the Cauchy product in the partial semiring over the partial monoids of marked graphs, we obtain

$$\begin{aligned}&(\llbracket\Delta\rrbracket^n)_{q,q'}(G,\sigma,v,v')\\ &\quad= \bigoplus_{q''\in Q} \bigoplus_{\substack{(G,\sigma,v,v')=\\(G_1,\sigma_1,v_1,v_1')\circ\\(G_2,\sigma_2,v_2,v_2')}} \llbracket\Delta\rrbracket_{q,q''}(G_1,\sigma_1,v_1,v_1') \otimes (\llbracket\Delta\rrbracket^{n-1})_{q'',q'}(G_2,\sigma_2,v_2,v_2')\end{aligned}$$

which is equal to

$$\bigoplus_{q''\in Q, v''\in V} \llbracket\Delta\rrbracket_{q,q''}(G,\sigma,v,v'') \otimes (\llbracket\Delta\rrbracket^{n-1})_{q'',q'}(G,\sigma,v'',v')$$

as $(G,\sigma,v,v') = (G_1,\sigma_1,v_1,v_1') \circ (G_2,\sigma_2,v_2,v_2')$ implies $G_1 = G_2 = G$, $\sigma_1 = \sigma_2 = \sigma$, $v = v_1$, $v' = v_2'$ and $v_1' = v_2$. □

The same way we introduced the semantic matrix $\llbracket\Delta\rrbracket$ we may also introduce

- the semantic initial row vector $\llbracket I\rrbracket \in \mathbb{S}\langle\langle\mathcal{MG}\rangle\rangle^Q$ defined by

$$\llbracket I\rrbracket_q = \bigoplus_{\substack{G\in\mathcal{G}(A,D)\\ \sigma\colon \mathrm{Var}\rightharpoonup V}} I_q\ (G,\sigma,v^{(i)},v^{(i)})\,;$$

- the semantic final column vector $\llbracket F\rrbracket \in \mathbb{S}\langle\langle\mathcal{MG}\rangle\rangle^Q$ defined by

$$\llbracket F\rrbracket_q = \bigoplus_{\substack{G\in\mathcal{G}(A,D)\\ \sigma\colon \mathrm{Var}\rightharpoonup V}} \bigoplus_{\substack{\alpha\in\mathrm{Test}\mid\\ G,\sigma,v^{(f)}\models\alpha}} F_q(\alpha)\ (G,\sigma,v^{(f)},v^{(f)})\,.$$

As a corollary of Proposition 4.12, we obtain that for every pointed graph G and every valuation σ,

$$\llbracket\mathcal{A}\rrbracket(G,\sigma) = (\llbracket I\rrbracket \otimes \llbracket\Delta\rrbracket^\star \otimes \llbracket F\rrbracket)(G,\sigma,v^{(i)},v^{(f)}) \tag{4.2}$$

where $\otimes$ is interpreted as the matrix multiplication induced by the Cauchy product over $\mathbb{S}\langle\langle\mathcal{MG}\rangle\rangle$.

Now, Lemma 2.7 shows that the entries of the star of a matrix H are in the rational closure[4] of the entries of H. Using this fact, we can prove

Proposition 4.13. *Let $\mathcal{A} = (Q,A,D,I,\Delta,F)$ be a weighted automaton. We can construct a matrix $\Phi(\Delta) \in \mathrm{WE}^{Q\times Q}$ which is equivalent to the automaton, i.e., such that the following matrix equality holds:*

$$\llbracket\Phi(\Delta)\rrbracket = \llbracket\Delta\rrbracket^\star$$

Moreover, the entries of $\Phi(\Delta)$ are in the rational closure *of the entries of Δ.*

[4] The rational closure is the closure under sum (+), product (·) and star (⋆).

Proof. We construct by induction a matrix $\Phi(\Delta)$ with coefficients in WE such that $[\![\Phi(\Delta)]\!] = [\![\Delta]\!]^\star$. If $|Q| = 1$, i.e., when $\Delta \in \mathrm{WE}$, then we simply let $\Phi(\Delta) = \Delta^\star$. From the algebraic definition of the semantics of weighted expressions, we directly obtain $[\![\Phi(\Delta)]\!] = [\![\Delta]\!]^\star$.

Next, if $|Q| > 1$, we apply Lemma 2.7 in the semiring $\mathbb{S}\langle\!\langle \mathcal{MG} \rangle\!\rangle$ with the decomposition

$$\Delta = \begin{pmatrix} E & F \\ G & H \end{pmatrix} \qquad \text{that implies} \qquad [\![\Delta]\!] = \begin{pmatrix} [\![E]\!] & [\![F]\!] \\ [\![G]\!] & [\![H]\!] \end{pmatrix}$$

with H of size 1, to get

$$[\![\Delta]\!]^\star = \begin{pmatrix} ([\![E]\!] \oplus [\![F]\!] \otimes [\![H]\!]^\star \otimes [\![G]\!])^\star & [\![E]\!]^\star \otimes [\![F]\!] \otimes ([\![H]\!] \oplus [\![G]\!] \otimes [\![E]\!]^\star \otimes [\![F]\!])^\star \\ [\![H]\!]^\star \otimes [\![G]\!] \otimes ([\![E]\!] \oplus [\![F]\!] \otimes [\![H]\!]^\star \otimes [\![G]\!])^\star & ([\![H]\!] \oplus [\![G]\!] \otimes [\![E]\!]^\star \otimes [\![F]\!])^\star \end{pmatrix}$$

As H is a matrix of size 1, this is a weighted expression, so that $H^\star$ is well-defined. To mimick this semantical computation, we then let

$$\Phi(\Delta) = \begin{pmatrix} \Phi(E + F \cdot H^\star \cdot G) & \Phi(E) \cdot F \cdot (H + G \cdot \Phi(E) \cdot F)^\star \\ H^\star \cdot G \cdot \Phi(E + F \cdot H^\star \cdot G) & (H + G \cdot \Phi(E) \cdot F)^\star \end{pmatrix}$$

Notice that we apply the operator Φ twice over matrices of size $|Q| - 1$. Moreover, we extend the concatenation of weighted expressions to matrices of weighted expressions in a straightforward way. Finally, notice that $G \cdot \Phi(E) \cdot F$ is a matrix of size 1 so that $(H + G \cdot \Phi(E) \cdot F)^\star$ is well-defined. We now prove that $[\![\Phi(\Delta)]\!] = [\![\Delta]\!]^\star$.

From the algebraic definition of the semantics and Proposition 3.18 of weighted expressions, H and $(H + G \cdot \Phi(E) \cdot F)^\star$ being weighted expressions, we obtain

$$[\![H^\star]\!] = [\![H]\!]^\star \qquad \text{and} \qquad [\![(H + G \cdot \Phi(E) \cdot F)^\star]\!] = ([\![H]\!] \oplus [\![G]\!] \otimes [\![\Phi(E)]\!] \otimes [\![F]\!])^\star$$

By induction we get

$$[\![\Phi(E + F \cdot H^\star \cdot G)]\!] = [\![E + F \cdot H^\star \cdot G]\!]^\star = ([\![E]\!] \oplus [\![F]\!] \otimes [\![H]\!]^\star \otimes [\![G]\!])^\star$$

which permits to conclude by considering again the algebraic properties implied by Proposition 3.18. □

As a corollary, using (4.2) and the fact that $[\![F]\!]$ may be easily transformed into an equivalent column vector with coefficients in $\mathrm{WE}(\mathbb{S}, A, D)$, we have proved item 2 of Theorem 4.9.

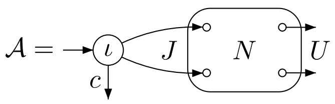
Example 4.14 (Continued from Example 4.8)**.** We apply the technique presented in this section to find a weighted expression equivalent to the weighted automaton $\mathcal{A}$ presented in Figure 4.4. Throughout this proof, we allow ourself to simplify the weighted expressions produced, e.g., by remarking that, in the semiring $(\mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0)$, $+\infty \cdot E$ is always equivalent to $+\infty$, $+\infty + E$ is equivalent to E, and $0 \cdot E$ is equivalent to E. Also, we remove tests $\top$ which do not change the semantics of weighted expressions.

First, we translate $\mathcal{A}$ into a generalized weighted automaton. We obtain the following matrix of weighted expressions

$$\widetilde{\Delta} = \begin{pmatrix} (2a? + 1b?){\downarrow_1} & \neg({\downarrow_1}?){\uparrow_1} & \neg({\downarrow_1}?){\uparrow_2} \\ {\downarrow_2} & +\infty & +\infty \\ +\infty & {\uparrow_1} & {\uparrow_2} \end{pmatrix}$$

Considering the initial and final vectors, we only need to compute the $(1, 3)$-coefficient of the matrix $\Phi(\widetilde{\Delta})$. Considering the block decomposition given in the previous proof, we have

$$E = \begin{pmatrix} (2a? + 1b?){\downarrow_1} & \neg({\downarrow_1}?){\uparrow_1} \\ {\downarrow_2} & +\infty \end{pmatrix}, \; F = \begin{pmatrix} \neg({\downarrow_1}?){\uparrow_2} \\ +\infty \end{pmatrix}, \; G = \begin{pmatrix} +\infty & {\uparrow_1} \end{pmatrix} \text{ and } H = {\uparrow_2}$$



$$\mathcal{A} = \begin{pmatrix} 1 & \boxed{0} \end{pmatrix} \begin{pmatrix} 0 & \boxed{J} \\ \boxed{0} & \boxed{N} \end{pmatrix} \begin{pmatrix} c \\ \boxed{U} \end{pmatrix}$$

Figure 4.5: Graphical and matrix representations of automata

Noticing that $(+\infty)^\star = 0$, we can compute $\Phi(E)$ and obtain

$$\begin{aligned}\Phi(E)_{1,1} &= \big((2a? + 1b?)\downarrow_1 + \neg(\downarrow_1?)\uparrow_1\downarrow_2\big)^\star \\ \Phi(E)_{2,1} &= \downarrow_2\big((2a? + 1b?)\downarrow_1 + \neg(\downarrow_1?)\uparrow_1 \downarrow_2\ \big)^\star\end{aligned}$$

We finally find $\Phi(\widetilde{\Delta})_{1,3}$ by computing the upper coefficient of $\Phi(E) \cdot F \cdot (H + G \cdot \Phi(E) \cdot F)^\star$:

$$\begin{aligned}\Phi(\widetilde{\Delta})_{1,3} = \big((2a? + 1b?)\downarrow_1 + \neg(\downarrow_1?)\uparrow_1\downarrow_2\big)^\star\neg(\downarrow_1?)\uparrow_2 \\ \Big(\uparrow_2 + \uparrow_1\downarrow_2\big((2a? + 1b?)\downarrow_1 + \neg(\downarrow_1?)\uparrow_1\downarrow_2\big)^\star\neg(\downarrow_1?)\uparrow_2\Big)^\star\end{aligned}$$

which also gives a weighted expression equivalent to the weighted automaton $\mathcal{A}$. ■

From Weighted Expressions to Weighted Automata

We now prove item 1 of Theorem 4.9, i.e., how to transform a weighted expression to an equivalent weighted automaton. Efficient translations have been well-studied both in the Boolean and in the weighted (one-way) cases. Glushkov's translation [Glu61] (or Berry-Sethi [BS86]) is acknowledged to be among the best ones. The good news is that this construction can be adapted to cope with two-way moves as we will show in this section. The construction is by structural induction on the expression. The induction hypothesis to maintain is slightly strengthened, in order to prove the result, as presented in the following proposition.

Proposition 4.15. *For each weighted expression $E \in \mathrm{WE}$ we can construct a weighted automaton $\mathcal{A}_E \in \mathrm{WA}$ such that $[\![\mathcal{A}_E]\!] = [\![E]\!]$, i.e., for all $(G, \sigma, v, v') \in \mathcal{MG}$ we have*

$$[\![\mathcal{A}_E]\!](G, \sigma, v, v') = [\![E]\!](G, \sigma, v, v')\,.$$

We define the *literal-length* $\ell\ell(E)$ of a weighted expression as the number of occurrences of directions from D. We construct an automaton $\mathcal{A}_E$ with $1 + \ell\ell(E)$ states.

For the rational operations ($+$, $\cdot$, *, and $^+$), we can still use the classical constructions even though we are working with navigating weighted automata. Nethertheless, for seek of clarity, we repeat the whole proof in the following. We adopt the presentation of standard automata found in [Sak12]. A standard automaton $\mathcal{A} = (Q, A, D, I, \Delta, F)$ has a single initial state ι with (initial) weight 1, all other states have initial weight 0. Moreover, the initial state ι has no ingoing transition. We use both the graphical representation and the matrix representation of an automaton of Figure 4.5, where the entries of the matrices J and N are polynomials in $\mathbb{S}\langle \mathrm{Test}\rangle\langle D\rangle$.

Since terminal weights allow polynomials over Test with the vector $F \in \mathbb{S}\langle \mathrm{Test}\rangle^Q$, we will be able to cope with expressions of the form $E \cdot \varphi?$ and $E \cdot s$ without adding unnecessary states. The constant term c and the entries of U are polynomials in $\mathbb{S}\langle \mathrm{Test}\rangle$. For $s \in \mathbb{S}$, $\alpha \in \mathrm{Test}$ and $d \in D$, we simply write s for $s\top$, α for 1α, and d for $1\top d$.

We start with atoms. Compared to the classical (one-way) translation, a slight difference is that we are using tests (α) and directions (d) instead of letters for the atoms. The automata for the atoms are depicted in Figure 4.6 and we can easily see that they are equivalent to the corresponding atoms.

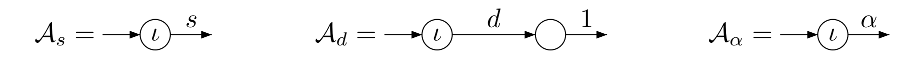


Figure 4.6: Automata for atomic expressions

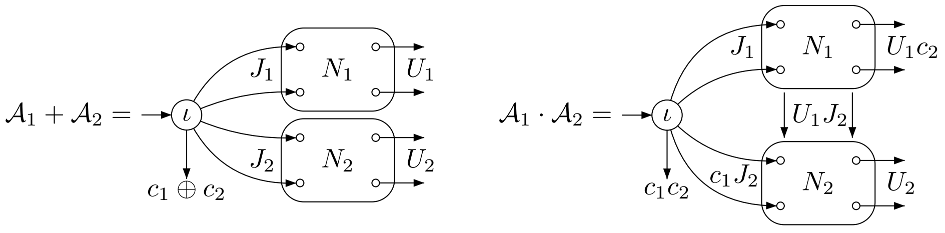
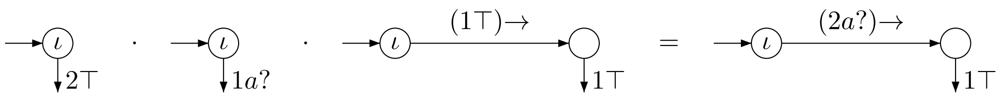


Figure 4.7: Automata for sum and concatenation

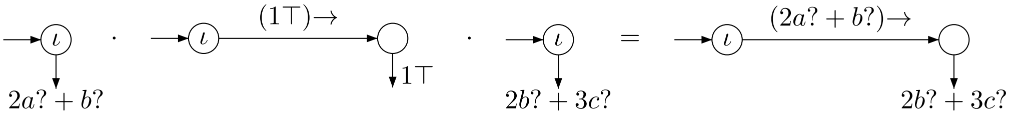
The constructions for sum and concatenation are as usual, and are depicted in Figure 4.7. In the concatenation, we are overloading[5] the product notation as follows. The product of two monomials $s_1\alpha_1$ and $s_2\alpha_2$ from $\mathbb{S}\langle\text{Test}\rangle$ should be understood as $(s_1 \otimes s_2)(\alpha_1 \wedge \alpha_2)$ to stay in $\mathbb{S}\langle\text{Test}\rangle$. Hence $c_1 \otimes c_2$ and the entries of $U_1 \otimes c_2$ are in $\mathbb{S}\langle\text{Test}\rangle$. Similarly, in $U_1 \otimes J_2$, the product of a monomial $s_1\alpha_1 \in \mathbb{S}\langle\text{Test}\rangle$ and a monomial $s_2\alpha_2 d \in \mathbb{S}\langle\text{Test}\rangle\langle D\rangle$ (with $d \in D$) is defined as $(s_1 \otimes s_2)(\alpha_1 \wedge \alpha_2)d$. Hence, the entries of the matrices $c_1 \otimes J_2$ and $U_1 \otimes J_2$ are in $\mathbb{S}\langle\text{Test}\rangle\langle D\rangle$. The matrix representation is therefore:

$$\mathcal{A}_1 \cdot \mathcal{A}_2 = \left(\begin{array}{c|c|c} 1 & 0 & 0 \end{array}\right) \left(\begin{array}{c|c|c} 0 & J_1 & c_1 \otimes J_2 \\ \hline 0 & N_1 & U_1 \otimes J_2 \\ \hline 0 & 0 & N_2 \end{array}\right) \left(\begin{array}{c} c_1 \otimes c_2 \\ \hline U_1 \otimes c_2 \\ \hline U_2 \end{array}\right)$$

Example 4.16. For instance, the automaton for $2a = 2a?{\rightarrow}$ is computed as follows:



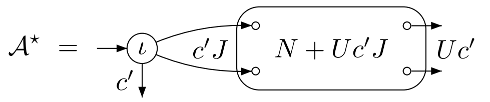
Similarly, for the expression $E = (2a? + b?){\rightarrow}(2b? + 3c?)$ we compute the concatenation of 3 automata as follows:



■

Finally, the star is also computed as usual with the construction depicted in Figure 4.8, where $c' \in \mathbb{S}\langle\text{Test}\rangle$ is defined to be equivalent to $c^\star$ as follows. Since $c \in \mathbb{S}\langle\text{Test}\rangle$ and test formulas are closed under Boolean connectives, we find an equivalent expression $c = \sum_i s_i\alpha_i$ with $(\alpha_i)_i$ pairwise incompatible test formulae ($i \neq j$ implies $\alpha_i \wedge \alpha_j$ unsatisfiable). Then

[5] Moreover, notice that we denote by M_1M_2 the product of matrices and elements in drawings, whereas it is denoted by $M_1 \otimes M_2$ in texts, for spacing reasons.



$$\mathcal{A}^\star = \begin{pmatrix} 1 & \boxed{0} \end{pmatrix} \begin{pmatrix} 0 & c' \otimes J \\ 0 & N \oplus U \otimes c' \otimes J \end{pmatrix} \begin{pmatrix} c' \\ U \otimes c' \end{pmatrix}$$

Figure 4.8: Automaton for Kleene star

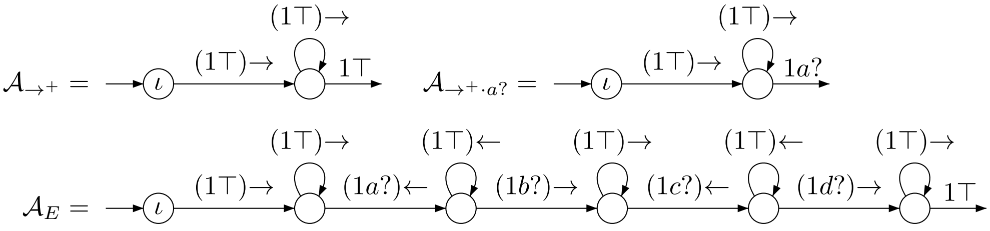
we can easily check that $[\![c]\!]^\star = [\![c']\!]$ with $c' = \sum_i s_i^\star \alpha_i$. The particular case $c = 0 = 0\top$ gives $c' = 1 = 1\top$. Notice that $s_i^\star \in \mathbb{S}$ is well-defined since the semiring is complete.

As for the concatenation, we can check that the entries of $U \otimes c'$ are in $\mathbb{S}\langle\text{Test}\rangle$ and the entries of $U \otimes c' \otimes J$ are in $\mathbb{S}\langle\text{Test}\rangle\langle D\rangle$. The strict iteration $\mathcal{A}^+$ is computed similarly by simply changing the final weight of ι to $c'' = \sum_i s_i^+ \alpha_i$ (note that $0^+ = 0$), but keeping the other occurrences of c' in $c' \otimes J$, $U \otimes c' \otimes J$ and $U \otimes c'$.

Example 4.17. (Continued from Example 4.16) For instance, for expression

$$E = \rightarrow^+ a? \leftarrow^+ b? \rightarrow^+ c? \leftarrow^+ d? \rightarrow^+ ,$$

we can compute the automaton as follows:



■

Let us briefly discuss the complexity of our translation. Clearly, the number of states of the automaton $\mathcal{A}_E$ is 1 plus the literal-length $\ell\ell(E)$ of expression E. The time complexity is cubic in the length of E, since in the worst case, each step requires to compute a bounded number of multiplications of a matrix and a vector of sizes bounded by $\ell\ell(E)$. As future works, it would be interesting to search for a quadratic algorithm: one possible direction could be to generalize the notion of star normal form introduced in [BK93] for word languages or the algorithm presented in [AM06] for classical weighted expressions and automata.

To conclude this section, we prove the correctness of the constructions. This correctness is trivial for the atoms. For sum, product and star, the proof is similar to the case of classical (one-way) weighted expressions and automata. Indeed, from (4.2), we know that $[\![\mathcal{A}]\!] = [\![I]\!] \otimes [\![\Delta]\!]^\star \otimes [\![F]\!]$: notice that this is an equality between series over marked graphs. Using the special form of standard automata and Lemma 2.7, we have

$$[\![\Delta]\!] = \begin{pmatrix} 0 & [\![J]\!] \\ 0 & [\![N]\!] \end{pmatrix} \qquad [\![\Delta]\!]^\star = \begin{pmatrix} 1 & [\![J]\!] \otimes [\![N]\!]^\star \\ 0 & [\![N]\!]^\star \end{pmatrix}$$

and we get $[\![\mathcal{A}]\!] = [\![c]\!] \oplus [\![J]\!] \otimes [\![N]\!]^\star \otimes [\![U]\!]$.

For the concatenation, the matrix N of $\mathcal{A} = \mathcal{A}_1 \cdot \mathcal{A}_2$ satisfies (applying Lemma 2.7):

$$\llbracket N \rrbracket = \left(\begin{array}{|c|c|} \hline \llbracket N_1 \rrbracket & \llbracket U_1 \otimes J_2 \rrbracket \\ \hline 0 & \llbracket N_2 \rrbracket \\ \hline \end{array} \right)$$

$$\llbracket N \rrbracket^\star = \left(\begin{array}{|c|c|} \hline \llbracket N_1 \rrbracket^\star & \llbracket N_1 \rrbracket^\star \otimes \llbracket U_1 \otimes J_2 \rrbracket \otimes \llbracket N_2 \rrbracket^\star \\ \hline 0 & \llbracket N_2 \rrbracket^\star \\ \hline \end{array} \right) .$$

Hence we obtain

$$\begin{aligned} \llbracket \mathcal{A} \rrbracket &= \llbracket c \rrbracket \oplus \llbracket J \rrbracket \otimes \llbracket N \rrbracket^\star \otimes \llbracket U \rrbracket \\ &= \llbracket c_1 \otimes c_2 \rrbracket \oplus \llbracket J_1 \rrbracket \otimes \llbracket N_1 \rrbracket^\star \otimes \llbracket U_1 \otimes c_2 \rrbracket \\ &\qquad \oplus \llbracket J_1 \rrbracket \otimes \llbracket N_1 \rrbracket^\star \otimes \llbracket U_1 \otimes J_2 \rrbracket \otimes \llbracket N_2 \rrbracket^\star \otimes \llbracket U_2 \rrbracket \\ &\qquad \oplus \llbracket c_1 \otimes J_2 \rrbracket \otimes \llbracket N_2 \rrbracket^\star \otimes \llbracket U_2 \rrbracket \\ &= (\llbracket c_1 \rrbracket \oplus \llbracket J_1 \rrbracket \otimes \llbracket N_1 \rrbracket^\star \otimes \llbracket U_1 \rrbracket) \otimes (\llbracket c_2 \rrbracket \oplus \llbracket J_2 \rrbracket \otimes \llbracket N_2 \rrbracket^\star \otimes \llbracket U_2 \rrbracket) \\ &= \llbracket \mathcal{A}_1 \rrbracket \otimes \llbracket \mathcal{A}_2 \rrbracket \end{aligned}$$

For the star construction, the correctness is obtained as follows using classical rational identities and $\llbracket c \rrbracket^\star = \llbracket c' \rrbracket$:

$$\begin{aligned} \llbracket \mathcal{A} \rrbracket^\star &= (\llbracket c \rrbracket \oplus \llbracket J \rrbracket \otimes \llbracket N \rrbracket^\star \otimes \llbracket U \rrbracket)^\star \\ &= (\llbracket c \rrbracket^\star \otimes \llbracket J \rrbracket \otimes \llbracket N \rrbracket^\star \otimes \llbracket U \rrbracket)^\star \otimes \llbracket c \rrbracket^\star \\ &= (\llbracket c' \rrbracket \otimes \llbracket J \rrbracket \otimes \llbracket N \rrbracket^\star \otimes \llbracket U \rrbracket)^\star \otimes \llbracket c' \rrbracket \\ &= \llbracket c' \rrbracket \oplus \llbracket c' \rrbracket \otimes \llbracket J \rrbracket \otimes (\llbracket N \rrbracket^\star \otimes \llbracket U \rrbracket \otimes \llbracket c' \rrbracket \otimes \llbracket J \rrbracket)^\star \\ &\qquad\qquad \otimes \llbracket N \rrbracket^\star \otimes \llbracket U \rrbracket \otimes \llbracket c' \rrbracket \\ &= \llbracket c' \rrbracket \oplus \llbracket c' \rrbracket \otimes \llbracket J \rrbracket \otimes (\llbracket N \rrbracket \oplus \llbracket U \rrbracket \otimes \llbracket c' \rrbracket \otimes \llbracket J \rrbracket)^\star \otimes \llbracket U \rrbracket \otimes \llbracket c' \rrbracket \\ &= \llbracket c' \rrbracket \oplus \llbracket c' \otimes J \rrbracket \otimes \llbracket N \oplus U \otimes c' \otimes J \rrbracket^\star \otimes \llbracket U \otimes c' \rrbracket = \llbracket \mathcal{A}^\star \rrbracket \end{aligned}$$

This closes the proof of Theorem 4.9.

4.2 Pebble Weighted Automata

We considered weighted automata navigating over graphs as a way to parse weighted expressions. Could it be possible to translate the full set of hybrid weighted expressions into weighted automata? Consider as an example the hybrid weighted expression

$$E = \Big(x!\big(2(2{\rightarrow})^\star\big){\rightarrow}\Big)^\star x!\big(2(2{\rightarrow})^\star\big)$$

over the set $D = \{\rightarrow, \leftarrow\}$ of directions and the continuous semiring $(\mathbb{N} \cup \{+\infty\}, +, \times, 0, 1)$. Over a graph $G \in \mathcal{W}\mathrm{ord}(A)$ representing a finite word of length n, it computes the weight $\llbracket E \rrbracket(G) = 2^{n^2}$. Indeed, the expression successively marks every vertex of G with the variable x, and then computes the subexpression $2(2{\rightarrow})^\star$ over G, that computes 2^n. We now show that such a function cannot be generated by a weighted automaton.

Theorem 4.18. *Let $D = \{\rightarrow, \leftarrow\}$ and A an alphabet. There exists no weighted automaton $\mathcal{A} \in \mathrm{WA}$ over the semiring $(\mathbb{N} \cup \{+\infty\}, +, \times, 0, 1)$ such that for every graph $G \in \mathcal{W}\mathrm{ord}(A)$, $\llbracket \mathcal{A} \rrbracket(G) = 2^{|V|^2}$.*

Proof. We suppose that there exists a weighted automaton $\mathcal{A} \in \mathrm{WA}$ such that for every pointed graph $G \in \mathcal{W}\mathrm{ord}(A)$, $[\![\mathcal{A}]\!](G) = 2^{n^2}$ if G has n vertices. In particular, the semantics of every graph representing a word is finite. This implies that every accepting run visits only distinct configurations. Indeed, if there exists an accepting run over G (having n vertices indexed from 0 to $n-1$), visiting two occurrences of the same configuration, it is of the form $(G, q_0, 0) \rightsquigarrow^* (G, q, i) \rightsquigarrow^+ (G, q, i) \rightsquigarrow^* (G, q_f, n-1)$ with $I_{q_0} \times s_1 \times s_2 \times s_3 \times F_{q_f} \neq 0$, where s_1 is the weight of the subrun from $(G, q_0, 0)$ to (G, q, i), s_2 the weight of the loop over (G, q, i) and s_3 the weight from (G, q, i) to $(G, q_f, n-1)$. Then, we may build a run of weight $I_{q_0} \times s_1 \times s_2^k \times s_3 \times F_{q_f}$ over the same word, for every $k \geq 0$, by iterating the loop over configuration (G, q, i). Notice that s_2 is a positive integer, hence, $s_2 \geq 1$. Finally, we obtain

$$[\![\mathcal{A}]\!](G) \geq \sum_{k=0}^{\infty} I_{q_0} \times s_1 \times s_2 \times s_3 \times F_{q_f} = I_{q_0} \times s_1 \times \left(\sum_{k=0}^{\infty} s_2\right) \times s_3 \times F_{q_f} = +\infty$$

which violates the fact that $[\![\mathcal{A}]\!](G)$ must be finite.

Therefore, we have proved that every accepting run over a graph G must visit at most once every configuration, and hence has a length bounded by $|Q| \times n$. Denoting by M the greatest weight appearing in the description of the automaton $\mathcal{A}$, we obtain that the weight of a run must be bounded by $M^{|Q| \times n + 2}$. Moreover, a run is uniquely defined by the sequence of states it visits and of directions ($\rightarrow$ or $\leftarrow$) it follows: hence the number of runs (that do not visit twice a configuration) is bounded by $(2|Q|)^{|Q| \times n}$. In total, we obtain that

$$[\![\mathcal{A}]\!](G) \leq (2M|Q|)^{|Q| \times n + 2} = o(2^{n^2})$$

Hence, for n large enough, $[\![\mathcal{A}]\!](G) < 2^{n^2}$, which proves that $\mathcal{A}$ cannot recognize the series mentioned above. □

As we way to overcome this limitation, we now decribe how to add *pebbles* to weighted automata: a pebble is a device used to mark temporarily a vertex with a variable. It mimicks in a more operational way the operator $x!-$ of the hybrid weighted expressions. After giving the extended syntax and semantics, we will consider in details what is the difference between pebble weighted automata and hybrid weighted expressions. However, we will still be able to prove some Kleene-Schützenberger theorem relating HWE and *layered* pebble weighted automata.

4.2.1 Extended Syntax and Semantics

Pebbles have names taken from the infinite set Var of variables introduced in Chapter 3.

Definition 4.19. A *pebble weighted automaton* over a semiring $\mathbb{S}$ is a tuple $\mathcal{A} = (Q, A, D, I, \Delta, F)$ where

- Q is a finite set of states;
- A is a finite alphabet;
- D is a finite set of directions;
- $I \in \mathbb{S}^Q$ is the vector of initial weights;
- $\Delta \in \mathbb{S}\langle \mathrm{Test}(A, D, \mathrm{Var})\rangle\langle \mathrm{Actions}\rangle^{Q \times Q}$ is the weighted transition matrix, where $\mathrm{Actions} = D \cup \{\mathsf{drop}_x \mid x \in \mathrm{Var}\} \cup \{\mathsf{lift}\}$ is the set of available actions;
- $F \in \mathbb{S}\langle \mathrm{Test}(A, D, \mathrm{Var})\rangle^Q$ is the final vector: each state q is mapped to a polynomial F_q of tests.

The class of all pebble weighted automata over $\mathbb{S}$ is denoted by $\mathrm{PWA}(\mathbb{S})$ or PWA if the semiring is clear from the context, or not relevant. ■

Note that we add the ability for the weighted automata to *drop* a pebble of name x with the action drop_x and to later *lift* a pebble with the action lift. Pebbles are recorded in a last-in-first-out fashion, i.e., they are handled by a stack. This is the reason why lift actions are not labeled with the name of the pebble on which they will be applied: this is not a part of the choice of the automaton. In this manuscript, we will only consider stacks of bounded size, i.e., we will limit *a priori* the number of pebbles possibly dropped during an execution of the automaton.

Moreover, variables are reusable: this means that we have an unbounded supply of pebbles each marked by a pebble name in Var. More than one pebble with name x can be placed during the run, but only the last dropped is *visible*. However, when the latter will be lifted, the previous occurrence of pebble x will become visible again. Notice that our notion of reusability is slightly different from the notion of invisibility introduced in [EHS07], where only the last dropped pebbles are visible, even if several of them have the same pebble name.

Let $K \geq 0$ be an integer bound. Configurations are enriched with the stack of pebbles currently dropped: a *K-configuration* of a pebble weighted automaton $\mathcal{A} = (Q, A, D, I, \Delta, F)$ is a tuple (G, σ, q, π, v) composed of a pointed graph $G \in \mathcal{G}(A, D)$, a valuation $\sigma\colon \mathrm{Var} \rightharpoonup V$, a state q of Q, a stack of length at most K containing pairs of variables and vertices, $\pi \in (\mathrm{Var} \times V)^k$ with $k \leq K$ (by convention, the top of the stack will be placed on the right of the word π), and a vertex v of V. A configuration is a K-configuration for some K. Reusability of variables means that a pebble name may occur at several places in π, but only its topmost occurrence is visible. Having this in mind, we associate to every stack π and valuation σ, an enriched valuation $\sigma_\pi\colon \mathrm{Var} \rightharpoonup V$ by $\sigma_\varepsilon = \sigma$, and $\sigma_{\pi(x,v)} = \sigma_\pi[x \mapsto v]$.

Any two K-configurations with such a fixed graph G and fixed valuation σ give rise to a *concrete transition* $(G, \sigma, q, \pi, v) \rightsquigarrow (G, \sigma, q', \pi', v')$. Its *weight* is then defined by

$$\begin{cases} \displaystyle\bigoplus_{\substack{d \in D \mid (v,v') \in E_d \\ \alpha \in \mathrm{Test} \mid G, \sigma_\pi, v \models \alpha}} \Delta_{q,q'}(d)(\alpha) & \text{if } \pi = \pi' \\ \displaystyle\bigoplus_{\alpha \in \mathrm{Test} \mid G, \sigma_\pi, v \models \alpha} \Delta_{q,q'}(\mathsf{drop}_x)(\alpha) & \text{if } v' = v^{(i)}, \text{ and } \pi' = \pi(x, v) \\ \displaystyle\bigoplus_{\alpha \in \mathrm{Test} \mid G, \sigma_\pi, v \models \alpha} \Delta_{q,q'}(\mathsf{lift})(\alpha) & \text{if } v = v^{(f)}, \text{ and } \pi = \pi'(y, v') \text{ for some } y \in \mathrm{Var} \end{cases}$$

Remark 4.20. In this definition, we suppose that the valuation σ_π has a domain that contains the free variables in all the tests α that appear. We will come back later on this point. ■

Notice that the first case, dealing with actions being a direction of D, is a straightforward extension of the semantics of weighted automata given in (4.1): we simply ensure that the stack of pebbles is not changed during such a concrete transition. The second case, dealing with drop transitions, set the next vertex to be the initial vertex $v^{(i)}$ of the graph, enriching the stack of pebbles with the pair (x, v) composed of the name of the dropped pebble and the vertex v where it is dropped. Notice that this transition is enabled only if π' has size bounded by K, i.e., if less than K pebbles have been dropped so far. The third case, dealing with lift transitions, assumes that the current vertex is the final vertex $v^{(f)}$ of the graph, pops the top of the stack (that therefore should be non-empty), and moves to the vertex v' where the last dropped pebble was.

Remark 4.21. It might seem awkward to enforce the condition on the initial and final vertices in the two last cases of this definition: we will indeed use these conditions to simplify a little some proofs later on. Indeed, another natural choice would have been to let the automaton stay on its current vertex when a pebble is dropped, and assume that it is on the vertex holding the last dropped pebble to let it lift this pebble. In principle, this other choice would have led to an automaton model generating the same behaviors. This is the case if the class of graphs in which automata navigate permit to *deterministically* move to the initial vertex, to the final vertex or to any vertex marked with a visible pebble: words, ranked trees, nested words and pictures have clearly this property for example. Another possibility could be to relax the property only for one of these two actions. For example, it is possible to let lift transitions happen everywhere in the graph by simply erasing the condition $v = v^{(f)}$ in the third case. Again, for the previously cited classes of graphs, this would not have changed the behaviors since it is possible to deterministically move to the final vertex and perform only lift transitions on $v^{(f)}$. In the following, we henceforth may use the possibility to lift pebbles anywhere without harm. However, notice that in every case, a lift transition must lead the automaton in the vertex where the lifted pebble was: if this is not the case, pebbles are called *strong* in the literature, whereas our pebbles are called *weak*, as detailed, e.g., in [BSSS06]. Even though strong pebbles are showed to have the same expressive power as weak pebbles in the Boolean setting (see [BSSS06]), we do not know if this is the case in our weighted setting, hence we leave the investigation of strong pebbles for future work. ■

A K-run is a sequence of K-configurations related successively by concrete transitions. Its *weight* is the product of transition weights, multiplied from left to right. A run is a K-run for some K. We are interested in runs that start at some vertex v, in state q with empty stack, and end in some configuration with vertex v', in state q' with empty stack. Therefore, we let $[\![\mathcal{A}_{q,q'}]\!]_K(G,\sigma,v,v')$ be the sum of the weights of all K-runs leading from configuration $(G,\sigma,q,\varepsilon,v)$ to configuration $(G,\sigma,q',\varepsilon,v')$. If we consider a continuous semiring, this *a priori* infinite sum is again well-defined.

Before defining the semantics of pebble weighted automata, we must first extend the notion of free variables to pebble weighted automata. It is not as simple as in the case of weighted automata, since new pebbles can be dropped making the set of variables currently allocated in the valuation σ_π dynamic. Hence, we let $\mathrm{Free}(\mathcal{A})$ be the (finite) set of variables that must be in the domain of a valuation σ such that at least one run starting in a configuration $(G,\sigma,q,\varepsilon,v)$ requires this variable to be in the domain of σ_π to define the weight of one of its concrete transitions. We also let $\mathrm{Var}(\mathcal{A})$ be the whole set of variables appearing in the transition matrix and the final vector of $\mathcal{A}$ (either in tests $x?$ or in drop transitions drop_x).

Finally, the semantics of the pebble weighted automaton $\mathcal{A}$ maps every pointed graph G and valuation σ with domain containing the free variables of $\mathcal{A}$ to an element of the continuous semiring $\mathbb{S}$ by including the initial and final weights, letting

$$[\![\mathcal{A}]\!]_K(G,\sigma) = \bigoplus_{q,q'\in Q} I_q \otimes [\![\mathcal{A}_{q,q'}]\!]_K(G,\sigma,v^{(i)},v^{(f)}) \otimes \bigoplus_{\substack{\alpha\in\mathrm{Test}\,\mid\\ G,\sigma,v^{(f)}\models\alpha}} F_{q'}(\alpha)\,.$$

Remark 4.22. As we did for weighted automata in Remark 4.6, we may introduce the same notions of *initial state*, *final state* and *accepting run* in pebble weighted automata. In particular, an accepting run starts and ends with an empty stack of pebbles. ■

Example 4.23. Notice first that every weighted automaton of WA is also a pebble weighted automaton in PWA, and that in this case, the definition of free variables coincide. Moreover, pebbles permit to design automata in a *compositional* way. For example, consider the weighted automaton of Figure 4.2 mapping each word w represented by a graph $G \in \mathcal{W}\mathrm{ord}(A)$ to $2(|w|_a+1)$. Then, it is really easy to design a pebble weighted automaton mapping the same graph to $\big(2(|w|_a+1)\big)^{|w|_b}$. The idea is to drop a pebble (of name x in the following) on every vertex of the graph labeled by b, and simply execute the weighted automaton of Figure 4.2 each time a pebble is dropped, lifting the pebble at the end of its computation. The pebble weighted automaton obtained is depicted in Figure 4.9. Notice that if we allow lift transitions to happen anywhere in the graph as suggested in Remark 4.21, then we should change the label of the transition from state 2 to state 3 to $(2\,\mathrm{final?})\mathsf{lift}$. ■

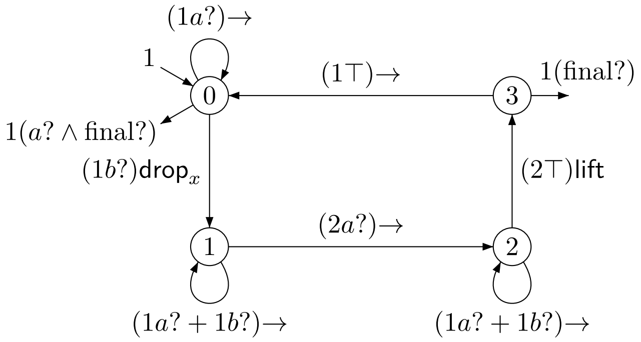


Figure 4.9: A pebble weighted automaton

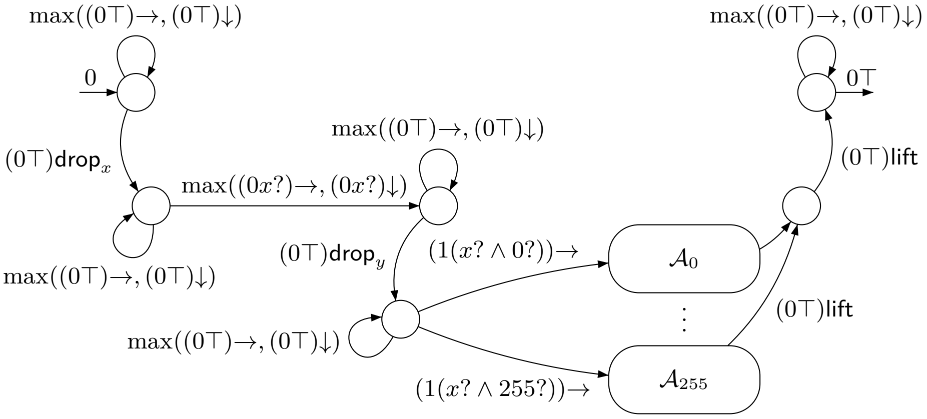


Figure 4.10: Pebble weighted automaton $\mathcal{A}$ computing the area of the biggest monochromatic rectangle of a picture

Example 4.24. As another example of pebble weighted automaton, consider the formal power series over pictures mapping each picture to the area of the biggest monochromatic rectangle it contains, in semiring $(\mathbb{N}\cup\{-\infty\}, \max, +, -\infty, 0)$. We show that we can generate it using a pebble weighted automaton $\mathcal{A}$. We consider $A = [0\mathinner{..}255]$ to be the alphabet of gray levels, and $D = \{\rightarrow, \leftarrow, \downarrow, \uparrow\}$ as set of directions. Automaton $\mathcal{A}$ uses two variables x and y in order to mark the upper-left and lower-right corners of a rectangle in the picture. At first, $\mathcal{A}$ chooses non-deterministically a position to drop x and then a position to drop y. The color c of the position hosting pebble x is put in the memory of the automaton. Then, starting from this position, the automaton scans each position of the rectangle checking that it has the gray level c, and computing 1 for each such position: to do so, the automaton can for example drop a pebble of name z on this position, in order to check that (*i*) it is below and on the right of x, (*ii*) above and on the left of y, and (*iii*) finally come back to this position to continue the computation. We denote by $\mathcal{A}_c$ the sub-automaton having two free variables x and y (and a further variable z which is not free) that checks whether the area limited by positions marked by variables x and y is a monochromatic rectangle of color c, and computing its area. Then automaton $\mathcal{A}$ is depicted in Figure 4.10: notice that this is typically a case where lift transitions are not necessarily performed on the final vertex, but as explained in Remark 4.21, this is not a problem in the case of pictures. Each accepting run – a run having a weight different from $-\infty$ in this case – computes the area of a monochromatic rectangle, by summing (again addition is the product of the semiring) 1 for every position inside this rectangle. The non-determinism is resolved by taking the maximum of the weights of all the accepting runs, which proves that $\mathcal{A}$ computes the area of

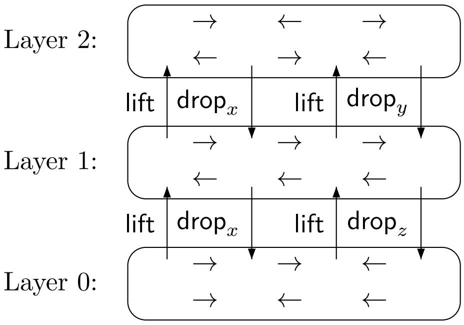


Figure 4.11: A 2-layered pebble weighted automaton

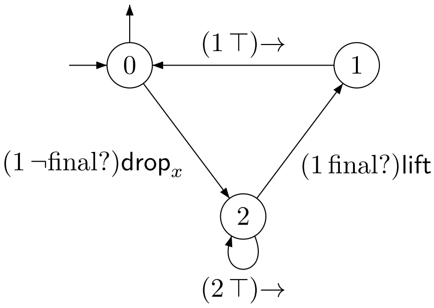

0
1
2
$(1\,\top)\!\rightarrow$
$(1\,\neg\text{final?})\mathsf{drop}_x$
$(1\,\text{final?})\mathsf{lift}$
$(2\,\top)\!\rightarrow$

Figure 4.12: 1-layered pebble weighted automaton $\mathcal{A}$

the biggest monochromatic rectangle. Notice that there are several accepting runs that check the same rectangle – since in the initial state, we allow the automaton to follow any sequence of $\rightarrow$ and $\downarrow$ actions for example – but this is not an issue here as the semiring is idempotent, in the sense that the maximum verifies $\max(s, s) = s$ for every $s \in \mathbb{N} \cup \{-\infty\}$. ■

4.2.2 Layered Pebble Weighted Automata

The bound K on the length of the stack can be seen as a semantical restriction, and it is natural to transform it into a syntactical restriction. Hence, we consider the natural subclass of pebble weighted automata that store in their state an information about the height of the current stack of pebbles.

Definition 4.25. A PWA $\mathcal{A} = (Q, A, D, I, \Delta, F)$ is *K-layered* if there is a mapping $\ell \colon Q \to [0 \mathinner{..} K]$ satisfying, for all $q, q' \in Q$,

- if $I_q \neq 0$ or $F_q \neq 0$ then $\ell(q) = K$;
- if there is $\alpha \in \text{Test}$ and $d \in D$ such that $\Delta_{q,q'}(d)(\alpha) \neq 0$ then $\ell(q) = \ell(q')$;
- if there is $\alpha \in \text{Test}$ such that $\Delta_{q,q'}(\mathsf{lift})(\alpha) \neq 0$ then $\ell(q') = \ell(q) + 1$;
- if there is $\alpha \in \text{Test}$ and $x \in \text{Var}$ such that $\Delta_{q,q'}(\mathsf{drop}_x)(\alpha) \neq 0$ then $\ell(q') = \ell(q) - 1$.

A layered PWA is a K-layered PWA for some K. ■

Figure 4.11 schematizes a 2-layered PWA, whereas Figure 4.12 depicts a 1-layered PWA with mapping ℓ given by: $\ell(0) = \ell(1) = 1$ and $\ell(2) = 0$.

We naturally obtain the following lemma relating pebble weighted automaton and layered ones.

Lemma 4.26. *For every K-layered pebble weighted automaton $\mathcal{A}$, $[\![\mathcal{A}]\!]_K(G, \sigma) = [\![\mathcal{A}]\!]_{K'}(G, \sigma)$ for every $K' > K$, every pointed graph G and every valuation σ.*

Proof. We only have to notice that every configuration appearing in an accepting run of $\mathcal{A}$ is necessarily a K-configuration. Hence, allowing more configurations do not add any accepting runs. □

Hence, for a K-layered pebble weighted automaton, we let $[\![\mathcal{A}]\!](G,\sigma)$ to be $[\![\mathcal{A}]\!]_K(G,\sigma)$. The next lemma explains why the K-layered restriction is a syntactical equivalent to the semantical restriction over the stack.

Lemma 4.27. *For every pebble weighted automaton $\mathcal{A}$ and for every $K \geq 0$, there exists a K-layered pebble weighted automaton $\mathcal{A}_K$ such that for every pointed graph G and every valuation σ: $[\![\mathcal{A}]\!]_K(G,\sigma) = [\![\mathcal{A}_K]\!](G,\sigma)$.*

Proof. We simply extend the state space of $\mathcal{A}$ to contain the layer information. If $\mathcal{A} = (Q, A, D, I, \Delta, F)$, we let $\mathcal{A}_K = (Q \times [0 \mathinner{..} K], A, D, I', \Delta', F')$ with

- $I'_{(q,K)} = I_q$ and $I'_{(q,k)} = 0$ for every state $q \in Q$, and $k < K$;
- $\Delta'_{(q,k),(q',k)}(d)(\alpha) = \Delta_{q,q'}(d)(\alpha)$ for every $k \in [0 \mathinner{..} K]$, $\alpha \in \text{Test}$, $d \in D$, and $q, q' \in Q$;
- $\Delta'_{(q,k),(q',k-1)}(\mathsf{drop}_x)(\alpha) = \Delta_{q,q'}(\mathsf{drop}_x)(\alpha)$ for every $k \in [1 \mathinner{..} K]$, $\alpha \in \text{Test}$, $x \in \text{Var}$, and $q, q' \in Q$;
- $\Delta'_{(q,k),(q',k+1)}(\mathsf{lift})(\alpha) = \Delta_{q,q'}(\mathsf{lift})(\alpha)$ for every $k \in [0 \mathinner{..} K-1]$, $\alpha \in \text{Test}$, and $q, q' \in Q$;
- all other transition weight in Δ' is set to 0;
- $F'_{(q,K)} = F_q$ and $F'_{(q,k)} = 0$ for every state $q \in Q$, and $k < K$.

By mapping a state (q,k) to layer $\ell((q,k)) = k$, we can verify that $\mathcal{A}_K$ is K-layered. Hence, $[\![\mathcal{A}_K]\!](G,\sigma) = [\![\mathcal{A}_K]\!]_K(G,\sigma)$ for every pointed graph G and every valuation σ. Moreover, by forgetting about the second component of states in $\mathcal{A}_K$, we can prove that K-runs of $\mathcal{A}$ over (G,σ) are in bijection with K-runs of $\mathcal{A}_K$ over (G,σ), and this bijection preserves the weight of these runs. This shows that $[\![\mathcal{A}]\!]_K(G,\sigma) = [\![\mathcal{A}_K]\!]_K(G,\sigma)$, which concludes the proof. □

Henceforth, to study its semantics $[\![-]\!]_K$, we can always suppose that a pebble weighted automaton is K-layered.

Another stronger restriction will be useful in the following. It consists of associating every layer of an automaton to a given variable: every drop transition appearing in this layer must then uses this special variable. We call such an automaton strongly layered.

Definition 4.28. A PWA $\mathcal{A} = (Q, A, D, I, \Delta, F)$ is *K-strongly layered* if it is K-layered (with mapping ℓ as defined previously) and if there exists a mapping $\mathcal{P}_\ell \colon [1 \mathinner{..} K] \to \text{Var}$ such that for every states $q, q' \in Q$, if $\Delta_{q,q'}(\mathsf{drop}_x)(\alpha) \neq 0$ then $x = \mathcal{P}_\ell(\ell(q))$. A strongly layered PWA is a K-strongly layered PWA for some K. ■

Every 0-layered PWA is trivially 0-strongly layered. Moreover, if the set of variables appearing in drop transitions of a K-layered automaton is reduced to a singleton $\{x\}$, then this automaton is strongly layered.

Interestingly, we may now define in another way the notion of free variables of a strongly layered automaton as we know exactly the sequence of pebbles that may be dropped. Indeed, $x \in \text{Var}$ is a free variable if it appears in a test from a transition starting in layer k such that x is not one of the pebbles dropped in upper layers, i.e., $x \notin \{\mathcal{P}_\ell(j) \mid j > k\}$. Notice that this gives an algorithm to compute the set $\text{Free}(\mathcal{A})$ of free variables of a strongly layered automaton.

Finally, notice that, at the price of extending the state space of an automaton, every series recognized by a K-layered PWA can also be recognized by a K-strongly layered PWA.

Proposition 4.29. *For every K-layered* PWA *$\mathcal{A}$, we can construct an equivalent K-strongly layered* PWA *$\mathcal{A}'$.*

Proof. Let $\mathcal{A} = (Q, A, D, I, \Delta, F)$ be a K-layered PWA, and ℓ be the mapping describing its layers as introduced in Definition 4.25. Let $\{z_1, \dots, z_K\}$ be a fresh set of K pebble names. We construct a K-strongly layered PWA $\mathcal{A}' = (\bigcup_{0\leq k\leq K} \ell^{-1}(k)\times\mathrm{Var}(\mathcal{A})^{K-k}, A, D, I', \Delta', F')$ equivalent to $\mathcal{A}$: the second component of the states records the sequence of pebble names of $\mathcal{A}$ that are supposed to be dropped, even though automaton $\mathcal{A}'$ simulates it by dropping some fresh pebble names. For initial and final weights, we simply set $I'_{(q,\varepsilon)} = I_q$ and $F'_{(q,\varepsilon)} = F_q$ if $\ell(q) = K$ and $I_{(q,\theta)} = F_{(q,\theta)} = 0$ if $\ell(q) < K$. For the transitions, we first explain how to transform tests formulae: for every test formula α and every sequence $\theta = \theta_K \cdots \theta_{k+1} \in \mathrm{Var}^{K-k}$, we let α_θ be the test formula defined by replacing in α every basic test $x?$, for $x = \theta_i$ for some i by the basic test $z_j?$, with j the smallest i such that $x = \theta_i$ (this choice of j permits to conform with the reusability of pebble names). Then, we define the new weighted transition matrix with:

$$\Delta'_{(q,\theta),(q',\theta')}(d)(\alpha_\theta) = \begin{cases} \Delta_{q,q'}(d)(\alpha) & \text{if } \theta' = \theta \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta'_{(q,\theta),(q',\theta')}(\mathsf{drop}_z)(\alpha_\theta) = \begin{cases} \Delta_{q,q'}(\mathsf{drop}_x)(\alpha) & \text{if } \theta' = \theta \cdot x \text{ and } z = z_{\ell(q)} \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta'_{(q,\theta),(q',\theta')}(\mathsf{lift})(\alpha_\theta) = \begin{cases} \Delta_{q,q'}(\mathsf{lift})(\alpha) & \text{if } \theta = \theta' \cdot y \text{ for some } y \in \mathrm{Var} \\ 0 & \text{otherwise.} \end{cases}$$

where $d \in D$, $x \in \mathrm{Var}$. Clearly, by letting $\ell(q, \theta) = \ell(q)$, automaton $\mathcal{A}'$ is K-layered. Moreover, it is easy to check that $\mathcal{A}'$ is even K-strongly layered as only pebble name z_k may be dropped in layer k. To prove that $\mathcal{A}$ and $\mathcal{A}'$ are equivalent, we construct a bijection f between configurations. This bijection transforms each configuration $(G, \sigma, (q, \theta), \pi, v)$ appearing in a run of $\mathcal{A}'$, with $\theta = \theta_K \cdots \theta_{K-k}$, into configuration $(G, \sigma, q, \pi_\theta, v)$ of $\mathcal{A}$ where π_θ is built in the following way: every pair (z_i, v_i) appearing in π is replaced by the pair (θ_i, v_i). We extend the function f to runs. By distinguishing cases where the action is a direction, a drop or a lift, we may verify that this mapping transforms a run of $\mathcal{A}'$ into a run of $\mathcal{A}$ having the same weight. Moreover, this function is a bijection as it preserves the length of the runs and is a bijection when restricted to the configurations. Hence, this proves that $[\![\mathcal{A}]\!](G, \sigma) = [\![\mathcal{A}']\!](G, \sigma)$ for every pointed graph G and valuation σ. □

4.2.3 Dynamically Marked Graphs

In order to relate hybrid weighted expressions and pebble weighted automata, we need to further study the semantics of our automaton model. Looking for a more algebraic definition of the semantics similar to what we studied for weighted automata (without pebbles), we will define dynamically marked graphs, as an extension of marked graphs in which the content of the stack of pebbles will be encoded, as it can now change along the computation of a pebble weighted automaton.

A *dynamically marked graph* is of the form $(G, \sigma, \pi, v, \pi', v')$ where $G \in \mathcal{G}(A, D)$ is a pointed graph, $\sigma\colon \mathrm{Var} \rightharpoonup V$ is a partial mapping encoding a valuation of pebbles, $v, v' \in V$ are vertices, and $\pi, \pi' \in (\mathrm{Var} \times V)^\star$ are stacks of pebbles. Let $\mathcal{DMG}(A, D, \mathrm{Var})$ (or shortly $\mathcal{DMG}$ if the parameters are implicit) be the set of all dynamically marked graphs. The partial product defined over marked graphs may be extended over dynamically marked graph by defining

$$(G_1, \sigma_1, \pi_1, v_1, \pi'_1, v'_1) \circ (G_2, \sigma_2, \pi_2, v_2, \pi'_2, v'_2) = (G_1, \sigma_1, \pi_1, v_1, \pi'_2, v'_2)$$

in the case where $G_1 = G_2$, $\sigma_1 = \sigma_2$, $\pi'_1 = \pi_2$ and $v'_1 = v_2$, this product being not defined in all other cases. By letting

$$\mathcal{DU}(A, D, \mathrm{Var}) = \{(G, \sigma, \pi, v, \pi', v') \in \mathcal{DMG}(A, D, \mathrm{Var}) \mid \pi = \pi', v = v'\}$$

be the set of *dynamically partial units*, we obtain the following algebraic structures, by simply extending the results from the non-dynamic case.

Proposition 4.30. $(\mathcal{DMG}(A, D, \mathrm{Var}), \circ, \mathcal{DU}(A, D, \mathrm{Var}))$ *is a partial monoid, and for every continuous semiring* $\mathbb{S}$*,* $(\mathbb{S}\langle\langle\mathcal{DMG}(A, D, \mathrm{Var})\rangle\rangle, \oplus, \otimes, 0, \mathbb{1}_{\mathcal{DU}(A,D,\mathrm{Var})})$ *is a continuous semiring.*

Proof. $(\mathcal{DMG}(A, D, \mathrm{Var}), \circ, \mathcal{DU}(A, D, \mathrm{Var}))$ is a partial monoid for exactly the same reasons as the ones developped in Example 3.15. Then, we conclude using Proposition 3.16. □

4.2.4 The Full Kleene-Schützenberger Theorem

We will now prove our completed Kleene-Schützenberger Theorem.

Theorem 4.31. *Let* $\mathbb{S}$ *be a continuous semiring.*

1. *For each hybrid weighted expression* $E \in \mathrm{HWE}(\mathbb{S})$*, we can construct a layered pebble weighted automaton* $\mathcal{A}_E \in \mathrm{PWA}(\mathbb{S})$ *with* $\mathrm{Free}(\mathcal{A}_E) = \mathrm{Free}(E)$ *equivalent to* E*, i.e.,* $[\![E]\!](G, \sigma) = [\![\mathcal{A}_E]\!](G, \sigma)$ *for every pointed graph* G *and every valuation* σ *with domain containing* $\mathrm{Free}(E)$*.*
2. *For each layered pebble weighted automaton* $\mathcal{A} \in \mathrm{PWA}(\mathbb{S})$*, we can construct a hybrid weighted expression* $E_{\mathcal{A}} \in \mathrm{HWE}(\mathbb{S})$ *with* $\mathrm{Free}(E_{\mathcal{A}}) = \mathrm{Free}(\mathcal{A})$*, equivalent to* $\mathcal{A}$*, i.e.,* $[\![\mathcal{A}]\!](G, \sigma) = [\![E_{\mathcal{A}}]\!](G, \sigma)$ *for every pointed graph* G *and every valuation* σ *with domain containing* $\mathrm{Free}(\mathcal{A})$*.*

From Pebble Weighted Automata to Hybrid Weighted Expressions

As for the case without pebble, we will start with the proof of the second item. For that, we will need to introduce generalized pebble weighted automata, using a new sort of expressions over their transitions: these will not be hybrid weighted expressions, but *pebble* weighted expressions, able to perform decoupled drop and lift moves, contrary to the $x!-$ operator of HWE.

Definition 4.32. We let $\mathrm{PWE}(\mathbb{S}, A, D, \mathrm{Var})$ (or shortly PWE) be the set of *pebble weighted expressions* defined by the following grammar:

$$E ::= s \mid \alpha \mid d \mid \mathsf{drop}_x \mid \mathsf{lift} \mid E + E \mid E \cdot E \mid E^+$$

where $s \in \mathbb{S}$, $\alpha \in \mathrm{Test}(A, D, \mathrm{Var})$, $d \in D$ and $x \in \mathrm{Var}$. ■

The semantics of these pebble weighted expressions is directly defined algebraically as a series over dynamically marked graphs. Indeed, we denote $\{\!|E|\!\}$ the series of the continuous semiring $\mathbb{S}\langle\langle\mathcal{DMG}(A, D, \mathrm{Var})\rangle\rangle$ inductively defined in Table 4.1.

We can now define generalized pebble weighted automata, as weighted automata with pebble weighted expressions over their transitions.

Definition 4.33. A *generalized pebble weighted automaton* is a tuple $\mathcal{A} = (Q, A, D, I, \Delta, F)$ where

- Q is a finite set of states;
- A is a finite alphabet;
- D is a finite set of directions;
- $I \in \mathbb{S}^Q$ is the initial vector;
- $\Delta \in \mathrm{PWE}(\mathbb{S}, A, D, \mathrm{Var})^{Q\times Q}$ is the transition matrix;
- $F \in \mathbb{S}\langle\mathrm{Test}(A, D, \mathrm{Var})\rangle^Q$ is the final vector.

The class of all generalized pebble weighted automata over $\mathbb{S}$ is denoted by $\mathrm{gPWA}(\mathbb{S})$ or gPWA if the semiring is clear from the context, or not relevant. ■

Table 4.1: Semantics of pebble weighted expressions

$$\begin{aligned}
\{\!\{s\}\!\} &= s\mathbb{1}_{\mathcal{DU}}\\
\{\!\{\alpha\}\!\} &= \mathbb{1}_{\{(G,\sigma,\pi,v,\pi,v)\in\mathcal{DU} \mid G,\sigma_\pi,v\models\alpha\}}\\
\{\!\{d\}\!\} &= \mathbb{1}_{\{(G,\sigma,\pi,v,\pi,v')\in\mathcal{DMG} \mid e=(v,v')\in E\wedge d\in\chi(e)\}}\\
\{\!\{\mathsf{drop}_x\}\!\} &= \mathbb{1}_{\{(G,\sigma,\pi,v,\pi(x,v),v^{(i)})\in\mathcal{DMG}\}}\\
\{\!\{\mathsf{lift}\}\!\} &= \mathbb{1}_{\{(G,\sigma,\pi'(y,v'),v^{(f)},\pi',v')\in\mathcal{DMG} \mid y\in\mathrm{Var}\}}\\
\{\!\{E_1+E_2\}\!\} &= \{\!\{E_1\}\!\}\oplus\{\!\{E_2\}\!\}\\
\{\!\{E_1\cdot E_2\}\!\} &= \{\!\{E_1\}\!\}\otimes\{\!\{E_2\}\!\}\\
\{\!\{E^+\}\!\} &= \{\!\{E\}\!\}^+
\end{aligned}$$

The semantics is given by a variation of the semantics of PWA: now, the weight of a concrete transition $(G,\sigma,q,\pi,v)\rightsquigarrow(G,\sigma,q',\pi',v')$ (with fixed graph and valuation) is defined by $\{\!\{\Delta_{q,q'}\}\!\}(G,\sigma,\pi,v,\pi',v')$.

Then, for a dynamically marked graph $(G,\sigma,\pi,v,\pi',v')\in\mathcal{DMG}$, we define

$$\{\!\{\mathcal{A}_{q,q'}\}\!\}(G,\sigma,\pi,v,\pi',v')$$

as the sum of weights of the runs from configuration (G,σ,q,π,v) to configuration (G,σ,q',π',v'). As for the case without pebbles, we can define the semantic matrix $\{\!\{\Delta\}\!\}$ whose (q,q')-coefficient is given by $\{\!\{\Delta_{q,q'}\}\!\}\in\mathbb{S}\langle\!\langle\mathcal{DMG}\rangle\!\rangle$. Copying, *mutatis mutandis*, the proofs of Propositions 4.12 and 4.13 we obtain the following proposition.

Proposition 4.34. *Let $\mathcal{A}=(Q,A,D,I,\Delta,F)$ be a generalized pebble weighted automaton. For all $q,q'\in Q$, the two series $\{\!\{\mathcal{A}_{q,q'}\}\!\}$ and $(\{\!\{\Delta\}\!\}^\star)_{q,q'}$ are equal. Moreover, we can construct a matrix $\Phi(\Delta)\in\mathrm{PWE}^{Q\times Q}$ (whose entries are in the rational closure of the entries of Δ) satisfying the following matrix equality:*

$$\{\!\{\Phi(\Delta)\}\!\}=\{\!\{\Delta\}\!\}^\star\,.$$

In particular, the semantics of a generalized pebble weighted automaton verifies: $\{\!\{\mathcal{A}\}\!\}=\{\!\{I\}\!\}\otimes\{\!\{\Delta\}\!\}^\star\otimes\{\!\{F\}\!\}=\{\!\{I\}\!\}\otimes\{\!\{\Phi(\Delta)\}\!\}\otimes\{\!\{F\}\!\}$.

Remark 4.35. The partial monoid $\mathcal{MG}$ is embedded in $\mathcal{DMG}$ by identifying the marked graph (G,σ,v,v') with the dynamically marked graph $(G,\sigma,\varepsilon,v,\varepsilon,v')$. Also, any pebble weighted automaton $\mathcal{A}=(Q,A,D,I,\Delta,F)$ can be considered as generalized pebble weighted automaton and the semantics over marked graphs coincide: $[\![\mathcal{A}_{q,q'}]\!](G,\sigma,v,v')=\{\!\{\mathcal{A}_{q,q'}\}\!\}(G,\sigma,\varepsilon,v,\varepsilon,v')$. This is in particular the case as we enforced the drop and lift actions to respectively reset the automaton to the initial vertex and to happen on the final vertex of the graph, which is coherent with the semantics of generalized pebble weighted automata. ■

By applying this remark, Proposition 4.34 constructs a pebble weighted expression equivalent over marked graphs to a pebble weighted automaton $\mathcal{A}$. But our aim is to construct a hybrid weighted expression which is equivalent to $\mathcal{A}$. This is achieved below using as a tool the detour via pebble weighted expressions.

First, we show that HWE can be seen as a fragment of PWE if we interpret $x!E$ as a macro for $\mathsf{drop}_x\cdot E\cdot\mathsf{lift}$. Indeed with this interpretation, the semantics coincide:

Lemma 4.36. *Let $E \in \mathrm{HWE}$ and let $(G,\sigma,\pi,v,\pi',v') \in \mathcal{DMG}$. Then,*

$$\{\!|E|\!\}(G,\sigma,\pi,v,\pi',v') \neq 0 \quad \textit{implies} \quad \pi' = \pi\,.$$

Moreover, if $\pi' = \pi$, we have

$$\{\!|E|\!\}(G,\sigma,\pi,v,\pi,v') = [\![E]\!](G,\sigma_\pi,v,v')\,.$$

Proof. We proceed by structural induction on HWE. For atoms of HWE, the result is clear from the definition of $\{\!| - |\!\}$. For sum, concatenation and star, the result is trivial since both semantics are compositional. The interesting case is $x!E$ which is interpreted as $\mathsf{drop}_x \cdot E \cdot \mathsf{lift}$ in PWE. By definition of $\{\!| - |\!\}$ and the Cauchy product in $\mathbb{S}\langle\!\langle \mathcal{DMG} \rangle\!\rangle$ we have

$$\{\!|\mathsf{drop}_x \cdot E \cdot \mathsf{lift}|\!\}(G,\sigma,\pi,v,\pi',v') = \bigoplus_{y \in \mathrm{Var}(E)} \{\!|E|\!\}(G,\sigma,\pi(x,v),v^{(i)},\pi'(y,v'),v^{(f)})\,.$$

Since $E \in \mathrm{HWE}$, we obtain by induction that each term of the sum above equals 0 if $\pi(x,v) \neq \pi'(y,v')$. Hence, $\{\!|\mathsf{drop}_x \cdot E \cdot \mathsf{lift}|\!\}(G,\sigma,\pi,v,\pi',v') \neq 0$ implies $\pi' = \pi$. Moreover, in that case, either $v' \neq v$ and

$$\{\!|\mathsf{drop}_x \cdot E \cdot \mathsf{lift}|\!\}(G,\sigma,\pi,v,\pi',v') = 0 = [\![x!E]\!](G,\sigma_\pi,v,v')\,,$$

or $v' = v$ and

$$\begin{aligned}\{\!|\mathsf{drop}_x \cdot E \cdot \mathsf{lift}|\!\}(G,\sigma,\pi,v,\pi',v') &= \{\!|E|\!\}(G,\sigma,\pi(x,v),v^{(i)},\pi'(x,v'),v^{(f)}) \\ &= [\![E]\!](G,\sigma_{\pi(x,v)},v^{(i)},v^{(f)}) \\ &= [\![x!E]\!](G,\sigma_\pi,v,v)\end{aligned}$$

where the second equality holds by induction and the third one follows from the definition of $\sigma_{\pi(x,v)}$. □

Hence, HWE can be seen as a fragment of PWE, and, in Proposition 4.34, if Δ is a matrix of hybrid weighted expressions, then, $\Phi(\Delta)$ is in the rational closure of the entries of Δ, hence is also a matrix of hybrid weighted expressions. However, if we start with $\mathcal{A}$ being a layered pebble weighted automaton, Δ is not a matrix of hybrid weighted expressions, hence we must refine this proof to cope with this more specific setting.

We first start by extending the definition of layered pebble weighted automata to generalized pebble weighted automata. Again, the idea is to decompose the state space into layers: the transition matrix relates states of the same layer with expressions in HWE, states of a layer with states of the lower level (respectively, upper level) with pebble weighted expressions involving drop_x (respectively, lift).

Definition 4.37. A generalized pebble weighted automaton $\mathcal{A} = (Q,A,D,I,\Delta,F)$ is K-*layered* if there is a mapping $\ell\colon Q \to [0\,..\,K]$ satisfying, for all $q,q' \in Q$,

- if $I_q \neq 0$ or $F_q \neq 0$ then $\ell(q) = K$;
- transitions may only relate neighbor layers, i.e., if the expression $\Delta_{q,q'}$ is not equal to 0 then $|\ell(q) - \ell(q')| \leq 1$;
- if $\ell(q) = \ell(q')$, then $\Delta_{q,q'} \in \mathrm{HWE}$;
- if $\ell(q') = \ell(q) - 1$, then the pebble weighted expression $\Delta_{q,q'}$ is of the form

$$\sum_{x \in \mathrm{Var}(\mathcal{A})} \delta^x_{q,q'} \cdot \mathsf{drop}_x$$

with $\delta^x_{q,q'} \in \mathbb{S}\langle \mathrm{Test} \rangle$;

- if $\ell(q') = \ell(q) + 1$, then the pebble weighted expression $\Delta_{q,q'}$ is of the form
$$\lambda_{q,q'} \cdot \mathsf{lift}$$
with $\lambda_{q,q'} \in \mathbb{S}\langle\text{Test}\rangle$.

A layered generalized pebble weighted automaton is a K-layered generalized pebble weighted automaton for some K. ■

Notice that a K-layered pebble weighted automaton can be seen as a K-layered generalized pebble weighted automaton very easily. We now extend Proposition 4.13 to any K-layered generalized pebble weighted automaton.

Proposition 4.38. *Let $\mathcal{A} = (Q, A, D, I, \Delta, F)$ be a 1-layered generalized pebble weighted automaton. We can construct a 0-layered generalized pebble weighted automaton $\mathcal{A}^{(1)} = (\ell^{-1}(1), A, D, I^{(1)}, \Delta^{(1)}, F^{(1)})$ which is equivalent to $\mathcal{A}$, i.e., such that the two series $\{\!\!\{\mathcal{A}_{q,q'}\}\!\!\}$ and $\{\!\!\{\mathcal{A}^{(1)}_{q,q'}\}\!\!\}$ are equal for all $q, q' \in \ell^{-1}(1)$.*

By ordering the states of Q by layer, first states of layer 1 and then states of layer 0, we obtain as a block decomposition
$$\Delta = \begin{pmatrix} N & D \\ L & P \end{pmatrix}$$
with N a square matrix indexed by states of layer 1, and P a square matrix indexed by states of layer 0.

Let $q_1, q_1' \in \ell^{-1}(1)$ be in layer 1 and $q_0, q_0' \in \ell^{-1}(0)$ be in layer 0. Then, D is a $\ell^{-1}(1) \times \ell^{-1}(0)$ *drop*-matrix whose (q_1, q_0)-entry can be written
$$\sum_{x \in \text{Var}(\mathcal{A})} \delta^x_{q_1,q_0} \cdot \mathsf{drop}_x$$
with $\delta^x_{q_1,q_0} \in \mathbb{S}\langle\text{Test}\rangle$. The (q_0', q_1')-entry of the $\ell^{-1}(0) \times \ell^{-1}(1)$ *lift*-matrix L can similarly be written
$$\lambda_{q_0',q_1'} \cdot \mathsf{lift}$$
with $\lambda_{q_0',q_1'} \in \mathbb{S}\langle\text{Test}\rangle$. Now, P is a $\ell^{-1}(0) \times \ell^{-1}(0)$ matrix of hybrid weighted expressions in HWE and we may apply Proposition 4.34 in order to get a matrix $\Phi(P)$ of hybrid weighted expressions which is equivalent to the iteration of P: $\{\!\!\{\Phi(P)\}\!\!\} = \{\!\!\{P\}\!\!\}^\star$ which also implies $[\![\Phi(P)]\!] = [\![P]\!]^\star$ as P and $\Phi(P)$ only contains hybrid weighted expressions (by using Lemma 4.36). From (D, P, L), we define the $\ell^{-1}(1) \times \ell^{-1}(1)$ matrix N' with coefficients in HWE by
$$N'_{q_1,q_1'} = \sum_{q_0,q_0' \in \ell^{-1}(0)} \sum_{x \in \text{Var}} \delta^x_{q_1,q_0} \cdot x!\big(\Phi(P)_{q_0,q_0'} \cdot \lambda_{q_0',q_1'}\big)\,. \tag{4.3}$$

The matrix N' is also denoted $C(D, P, L)$ below. Note that the maximal depth of the entries of N' is at most 1 plus the maximal depth of the entries of P since the construction $\Phi(P)$ does not increase the depth of expressions.

Lemma 4.39. *Series over dynamically marked graphs $\{\!\!\{N'\}\!\!\}$ and $\{\!\!\{D\}\!\!\} \otimes \{\!\!\{P\}\!\!\}^\star \otimes \{\!\!\{L\}\!\!\}$ are equal.*

Proof. Recall that when viewing hybrid weighted expressions as pebble weighted expressions, expression $x!E$ is interpreted as $\mathsf{drop}_x \cdot E \cdot \mathsf{lift}$. This implies that the hybrid weighted expression $x!\big(\Phi(P)_{q_0,q_0'} \cdot \lambda_{q_0',q_1'}\big)$ is equivalent to $\mathsf{drop}_x \cdot \Phi(P)_{q_0,q_0'} \cdot \lambda_{q_0',q_1'} \cdot \mathsf{lift}$. We deduce that
$$\{\!\!\{N'_{q_1,q_1'}\}\!\!\} = \sum_{q_0,q_0'} \{\!\!\{D_{q_1,q_0}\}\!\!\} \otimes \{\!\!\{\Phi(P)_{q_0,q_0'}\}\!\!\} \times \{\!\!\{L_{q_0',q_1'}\}\!\!\}\,.$$

Since $\{\!\!\{\Phi(P)\}\!\!\} = \{\!\!\{P\}\!\!\}^\star$ by Proposition 4.34, the result follows. □

To conclude the proof of Proposition 4.38, we simply set $\Delta^{(1)} = N + N'$. Now, for all $q, q' \in \ell^{-1}(1)$, we have $\{\!|\mathcal{A}_{q,q'}|\!\} = (\{\!|\Delta|\!\}^\star)_{q,q'}$ by Proposition 4.34. Moreover, the upper-left block of $\{\!|\Delta|\!\}^\star$ is $(\{\!|N|\!\} + \{\!|D|\!\} \otimes \{\!|P|\!\}^\star \otimes \{\!|L|\!\})^*$ which is, by Lemma 4.39, equal to $(\{\!|N|\!\} + \{\!|N'|\!\})^\star = \{\!|\Delta^{(1)}|\!\}^*$. Using again Proposition 4.34, we finally get for all $q, q' \in \ell^{-1}(1)$ that $\{\!|\mathcal{A}^{(1)}_{q,q'}|\!\} = (\{\!|\Delta^{(1)}|\!\}^\star)_{q,q'} = (\{\!|\Delta|\!\}^\star)_{q,q'} = \{\!|\mathcal{A}_{q,q'}|\!\}$.

Proposition 4.38 can then be generalized to an arbitrary number of layers.

Proposition 4.40. *Let $\mathcal{A} = (Q, A, D, I, \Delta, F)$ be a K-layered generalized pebble weighted automaton. We can construct a 0-layered generalized pebble weighted automaton $\mathcal{A}^{(K)} = (\ell^{-1}(K), A, D, I^{(K)}, \Delta^{(K)}, F^{(K)})$ which is equivalent to $\mathcal{A}$, i.e., such that the two series $\{\!|\mathcal{A}_{q,q'}|\!\}$ and $\{\!|\mathcal{A}^{(K)}_{q,q'}|\!\}$ are equal for all $q, q' \in \ell^{-1}(K)$.*

Proof. The proof is by induction on K. When $K = 0$ we simply set $\mathcal{A}^{(0)} = \mathcal{A}$. For $K > 0$, we order the states of Q by layer as previously, in a decreasing order. We obtain as a block decomposition

$$\Delta = \begin{pmatrix} N^{(K)} & D^{(K)} & 0 & \dots & 0 \\ L^{(K-1)} & N^{(K-1)} & D^{(K-1)} & \dots & 0 \\ 0 & L^{(K-2)} & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & N^{(1)} & D^{(1)} \\ 0 & 0 & \dots & L^{(0)} & N^{(0)} \end{pmatrix}$$

with $N^{(k)}$ a square matrix indexed by states of layer k. We set

$$\Delta^{(K)} = N^{(K)} + C(D^{(K)}, \Delta^{(K-1)}, L^{(K-1)})$$

where the matrix $\Delta^{(K-1)}$ is the transition matrix of the 0-layered generalized pebble weighted automaton obtained by induction, by considering only states of layers $\leq K - 1$. Correctness follows from Proposition 4.38. □

Finally, consider a K-layered pebble weighted automaton $\mathcal{A} = (Q, A, D, I, \Delta, F)$. By using Remark 4.35, we may see $\mathcal{A}$ as a K-layered *generalized* pebble weighted automaton with

$$\{\!|\mathcal{A}_{q,q'}|\!\}(G, \sigma, \varepsilon, v, \varepsilon, v') = [\![\mathcal{A}_{q,q'}]\!](G, \sigma, v, v') \,.$$

Proposition 4.40 then produces a 0-layered generalized pebble weighted automaton $\mathcal{A}^{(K)}$ verifying the series (of dynamically marked graphs) equality

$$\{\!|\mathcal{A}^{(K)}_{q,q'}|\!\} = \{\!|\mathcal{A}_{q,q'}|\!\}$$

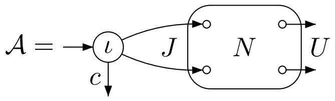
for every states q, q' of layer K. From Proposition 4.34, we deduce that the matrix $H = \Phi(\Delta^{(K)})$ satisfies the series equality

$$\{\!|H_{q,q'}|\!\} = \{\!|\mathcal{A}_{q,q'}|\!\}$$

for all q, q' of layer K. Notice that the coefficients of the transition matrix $\Delta^{(K)}$ of $\mathcal{A}^{(K)}$ (with 0 layers) are necessarily pebble weighted expressions of HWE. This is also the case of matrix $H = \Phi(\Delta^{(K)})$. Hence, by Lemma 4.36, and using notations of this lemma, we have

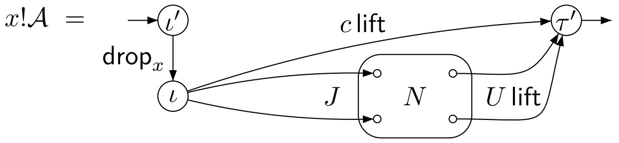
$$\{\!|H_{q,q'}|\!\}(G, \sigma, \pi, v, \pi, v') = [\![H_{q,q'}]\!](G, \sigma_\pi, v, v')$$

for all q, q' of layer K. Therefore, the pebble weighted expression $E_{\mathcal{A}}$ obtained by considering the matrix product of row I (with coefficients in $\mathbb{S}$), square matrix H and column F (with



$$\mathcal{A} = \begin{pmatrix} 1 & \boxed{0} \end{pmatrix} \begin{pmatrix} 0 & \boxed{J} \\ \boxed{0} & \boxed{N} \end{pmatrix} \begin{pmatrix} c \\ \boxed{U} \end{pmatrix}$$

Figure 4.13: A standard automaton $\mathcal{A}$



$$x!\mathcal{A} = \begin{pmatrix} 1\;0 \mid 0\,\boxed{0} \end{pmatrix} \left(\begin{array}{cc|cc} 0 & 0 & \mathsf{drop}_x & \boxed{0} \\ 0 & 0 & 0 & \boxed{0} \\ \hline 0 & c\,\mathsf{lift} & 0 & \boxed{J} \\ \boxed{0} & \boxed{U\,\mathsf{lift}} & \boxed{0} & \boxed{N} \end{array}\right) \left(\begin{array}{c} 0 \\ 1 \\ \hline 0 \\ \boxed{0} \end{array}\right)$$

Figure 4.14: Automaton for $x!-$

coefficients in $\mathbb{S}$) verifies

$$\begin{aligned} [\![E_{\mathcal{A}}]\!](G,\sigma) &= \bigoplus_{q,q'\in Q} I_q \otimes [\![H_{q,q'}]\!](G,\sigma,v^{(i)},v^{(f)}) \otimes F_{q'} \\ &= \bigoplus_{q,q'\in Q} I_q \otimes [\![\mathcal{A}_{q,q'}]\!](G,\sigma,v^{(i)},v^{(f)}) \otimes F_{q'} \\ &= [\![\mathcal{A}]\!](G,\sigma) \end{aligned}$$

This ends the proof of item 2 of Theorem 4.31. Notice that the depth of $E_{\mathcal{A}}$ is at most K if $\mathcal{A}$ is a K-layered pebble weighted automaton.

From Hybrid Weighted Expressions to Pebble Weighted Automata

We now give the proof of item 1 of Theorem 4.31. More precisely, we extend Proposition 4.15 as follows:

Proposition 4.41. *For each hybrid weighted expression $E \in \mathrm{HWE}$, we can construct a layered pebble weighted automaton $\mathcal{A}_E \in \mathrm{PWA}$ such that $[\![\mathcal{A}_E]\!] = [\![E]\!]$, i.e., for all $(G,\sigma,v,v') \in \mathcal{MG}$ we have*

$$[\![\mathcal{A}_E]\!](G,\sigma,v,v') = [\![E]\!](G,\sigma,v,v')\,.$$

The translation from weighted expressions to weighted automata has to be completed to deal with the new operator $x!E$. We give this construction now: it should drop the pebble on the current position, evaluate E from the initial to the final vertex of the pointed graph and finally lift the pebble. We start from a standard automaton $\mathcal{A}$ equivalent to E, as depicted in Figure 4.13. From it, we construct the following standard automaton $x!\mathcal{A}$, depicted in Figure 4.14. In the matrix representation, we have put first the states ι' and τ' of the topmost layer.

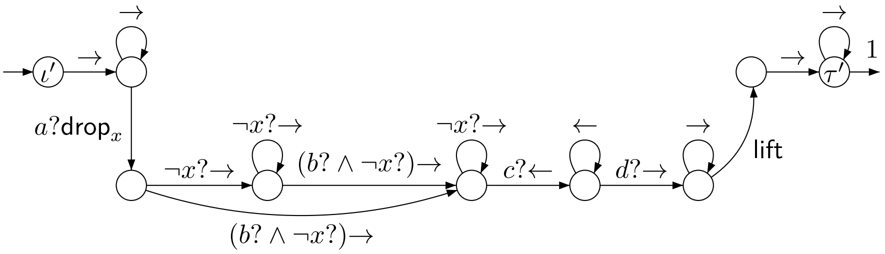


Figure 4.15: Automaton equivalent to the expression of Example 4.42

Example 4.42. (Continued from Example 4.17) For instance, consider expression E (over words) below:

$$E = \rightarrow^+ a?x!\Big((\neg x?\rightarrow)^\star b?(\neg x?\rightarrow)^+ c?\leftarrow^+ d?\rightarrow^+\Big)\rightarrow^\star .$$

The construction applied to E, using an automaton for the expression inside the $x!-$ operator obtained similarly to the one of Example 4.17, gives the pebble weighted automaton depicted in Figure 4.15. ■

It remains to prove the correctness of the new construction for the $x!-$ operation. Assume for simplicity that $\mathcal{A}$ is a 0-layered automaton, then $\mathcal{A}' = x!\mathcal{A}$ is a 1-layered automaton. We write the corresponding block decomposition of the transition matrix of $\mathcal{A}'$ as $\Delta' = \begin{pmatrix} 0 & D \\ L & \Delta \end{pmatrix}$ where Δ is the transition matrix of $\mathcal{A}$, D is the drop-matrix with only non-zero entry being $D_{\iota',\iota} = \mathsf{drop}_x$, and L is the lift matrix with non-zero entries being $c\,\mathsf{lift}$ and in the column matrix $U\,\mathsf{lift}$. By Lemma 2.7, the upper-left block of $\{\!|\Delta'|\!\}^\star$ is $\{\!|D|\!\} \otimes \{\!|\Delta|\!\}^\star \otimes \{\!|L|\!\}$.

By Remark 4.35, we have

$$\begin{aligned} [\![\mathcal{A}']\!](G,\sigma,v,v') &= (\{\!|\Delta'|\!\}^\star)_{\iota',\tau'}(G,\sigma,\varepsilon,v,\varepsilon,v') \\ &= (\{\!|D|\!\} \otimes \{\!|\Delta|\!\}^\star \otimes \{\!|L|\!\})_{\iota',\tau'}(G,\sigma,\varepsilon,v,\varepsilon,v') \\ &= (\{\!|C(D,\Delta,L)|\!\})_{\iota',\tau'}(G,\sigma,\varepsilon,v,\varepsilon,v') \,. \end{aligned}$$

Let Q be the set of non-initial states of $\mathcal{A}$. By definition of $C(D,M,L)$ given in (4.3), we have

$$C(D,M,L)_{\iota',\tau'} = x!c + \sum_{q\in Q} x!\big(\Phi(\Delta)_{\iota,q} \cdot U_q\big)$$

which is equivalent to expression $x!\Big(c + \sum_{q\in Q} \Phi(\Delta)_{\iota,q} \cdot U_q\Big)$. Hence, $[\![\mathcal{A}']\!](G,\sigma,v,v')$ is different from 0 only if $v = v'$, as for $[\![x!E]\!](G,\sigma,v,v')$. In that case, we have

$$[\![\mathcal{A}']\!](G,\sigma,v,v) = [\![c + \sum_{q\in Q} \Phi(\Delta)_{\iota,q} \cdot U_q]\!](G,\sigma[x\mapsto v],v^{(i)},v^{(f)}) \,.$$

By induction hypothesis, we have $[\![\mathcal{A}]\!](G,\sigma[x\mapsto v],v^{(i)},v^{(f)}) = [\![E]\!](G,\sigma[x\mapsto v],v^{(i)},v^{(f)})$

which implies that

$$\begin{aligned}
[\![x!E]\!](G,\sigma,v,v) &= [\![E]\!](G,\sigma[x\mapsto v],v^{(i)},v^{(f)}) \\
&= \{\!|\mathcal{A}|\!\}(G,\sigma[x\mapsto v],\varepsilon,v^{(i)},\varepsilon,v^{(f)}) \\
&= \big(\{\!|I|\!\}\otimes\{\!|\Delta|\!\}^\star\otimes\{\!|F|\!\}\big)(G,\sigma[x\mapsto v],\varepsilon,v^{(i)},\varepsilon,v^{(f)}) \\
&= \big(\{\!|I|\!\}\otimes\{\!|\Phi(\Delta)|\!\}\otimes\{\!|F|\!\}\big)(G,\sigma[x\mapsto v],\varepsilon,v^{(i)},\varepsilon,v^{(f)}) \\
&= \{\!|c+\sum_{q\in Q}\Phi(\Delta)_{\iota,q}\cdot U_q|\!\}(G,\sigma[x\mapsto v],\varepsilon,v^{(i)},\varepsilon,v^{(f)}) \\
&= [\![c+\sum_{q\in Q}\Phi(\Delta)_{\iota,q}\cdot U_q]\!](G,\sigma[x\mapsto v],v^{(i)},v^{(f)}) \\
&= [\![\mathcal{A}']\!](G,\sigma,v,v)\,.
\end{aligned}$$

This ends the proof of Theorem 4.31.

CHAPTER 5

Query Evaluation

The value of an idea lies in the using of it.

Thomas A. Edison

This chapter is devoted to the evaluation of a quantitative property over a given graph. For example, we may consider a quantitative property of XML documents (e.g., the number of nodes verifying a given property) and try to evaluate this query over a fixed database. This is the simplest problem to consider. However, it is far from trivial considering the quantitative queries we introduced in Part 1. Indeed, the semantics of several of the specification formalisms we introduced involve infinite sums, and trivial attemps to compute their semantics – even inefficient – are not possible *a priori*. We choose an automata-based approach to solve this problem, i.e., we consider that the specification is given as a pebble weighted automaton. It is another problem to efficiently translate other formalisms in automata models, and this has been presented previously. Our goal is henceforth to design algorithms evaluating pebble weighted automata over given graphs.

The first section presents a first generic approach for general classes of graphs, with a reasonable complexity. It is based on the computation of stars of matrices. Next sections present specialized algorithms for words, trees and nested words with better complexities. Whereas for words and nested words, we use the underlying linear order to guide our evaluation algorithm, the case of trees is resolved by considering a bottom-up computation. The special case of words has been originally published in [2], whereas the case of nested words is presented in [4].

5.1 Generic Evaluation of Pebble Weighted Automata

Our procedure of evaluation consists in constructing the weighted graph of configurations and, for each layer successively, computing the weights of paths for all pairs of vertices. Before explaining this procedure and its complexity in detail, we present an algorithm for computing the weights of paths in a weighted graph.

5.1.1 Weights of Paths in Weighted Graphs

In this section, we consider weighted finite graphs whose edges are equipped with weights in a continuous semiring $\mathbb{S}$. Every finite path of this graph has a weight, defined as the product of the weights of the edges it visits. For every pair of vertices of this graph, we may consider the set of paths leading from one to the other, and we associate to this pair the sum of the weights of these paths.

As an example of instantiation, consider the tropical semiring $(\mathbb{N}\cup\{+\infty\}, \min, +, +\infty, 0)$. Every pair of vertices of a weighted graph is now associated with the distance separating them. In this context, we may use the Floyd-Warshall algorithm [War62, Flo62] to compute this distance for all pairs of points. The time complexity of this algorithm is cubic in the number of vertices of the graph. Our method consists in generalizing this approach to graphs equipped with weights in a general continuous semiring.

The method is based on matrix computations. Indeed, consider the adjacency matrix M of the graph, namely the matrix indexed by $V \times V$ whose (v, v')-coefficient is the weight of the edge from vertex v to vertex v' if it exists, and 0 (the zero of the semiring) otherwise. Then, for every integer n, the (v, v')-coefficient of matrix M^n is the sum of the weights of the paths from v to v' of length exactly n. Hence, the weight we are searching for is exactly computed by the star $M^\star = \bigoplus_{n\geq 0} M^n$ of matrix M. We show below that computing the star of a square matrix has the same complexity as computing the product of matrices, i.e., can be done with a cubic number of sum and products in the semiring, to which we must add a linear number of scalar star operations.

Lemma 5.1. *Let $\mathbb{S}$ be a continuous semiring and $M \in \mathbb{S}^{n\times n}$ be a square matrix of dimension n. We can compute $M^\star$ with n scalar star operations and $\mathcal{O}(n^3)$ scalar sum and product operations.*

Proof. Let $M \in \mathbb{S}^{n\times n}$ be a matrix over a continuous semiring $\mathbb{S}$. Consider the block decomposition $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ with A a square matrix in $\mathbb{S}^{(n-1)\times(n-1)}$ and $D \in \mathbb{S}$. The Conway computation of the star of matrix given in Lemma 2.7 gives

$$M^\star = \begin{pmatrix} (A \oplus B \otimes D^\star \otimes C)^\star & A^\star \otimes B \otimes (D \oplus C \otimes A^\star \otimes B)^\star \\ D^\star \otimes C \otimes (A \oplus B \otimes D^\star \otimes C)^\star & (D \oplus C \otimes A^\star \otimes B)^\star \end{pmatrix}$$

This is not a good decomposition to obtain the complexity announced: in particular, to compute $M^\star$ it requires to precompute $A^\star$ and $(A \oplus B \otimes D^\star \otimes C)^\star$ which are two stars of $(n-1)\times(n-1)$-matrices, generating in total an exponential number of scalar star operations. In contrast, we propose an a priori more complicated formula to compute the star of matrix M, which will ensure the stated complexity:

$$M^\star = \begin{pmatrix} A' & A' \otimes B \otimes D^\star \\ D^\star \otimes C \otimes A' & D^\star \oplus D^\star \otimes C \otimes A' \otimes B \otimes D^\star \end{pmatrix}. \tag{5.1}$$

with $A' = (A \oplus B \otimes D^\star \otimes C)^\star$.

Indeed, it is only necessary to prove that the two right blocks are equal, i.e.,

$$\begin{aligned} A^\star \otimes B \otimes (D \oplus C \otimes A^\star \otimes B)^\star &= A' \otimes B \otimes D^\star \\ (D \oplus C \otimes A^\star \otimes B)^\star &= D^\star \oplus D^\star \otimes C \otimes A' \otimes B \otimes D^\star \end{aligned}$$

These identities are corollaries of the identities of Theorem 2.4. First,

$$\begin{aligned}
A' \otimes B \otimes D^\star &= (A \oplus B \otimes D^\star \otimes C)^\star \otimes B \otimes D^\star \\
&= A^\star \otimes (B \otimes D^\star \otimes C \otimes A^\star)^\star \otimes B \otimes D^\star && \text{by identity } (ii) \\
&= A^\star \otimes B \otimes D^\star \otimes (C \otimes A^\star \otimes B \otimes D^\star)^\star && \text{by identity } (i) \\
&= A^\star \otimes B \otimes (D \oplus C \otimes A^\star \otimes B)^\star && \text{by identity } (ii)
\end{aligned}$$

which proves the first identity. Then,

$$\begin{aligned}
(D \oplus C \otimes A^\star \otimes B)^\star &= D^\star \otimes (C \otimes A^\star \otimes B \otimes D^\star)^\star && \text{by identity } (ii) \\
&= D^\star \oplus D^\star \otimes C \otimes A^\star \otimes B \otimes D^\star \otimes (C \otimes A^\star \otimes B \otimes D^\star)^\star && \text{by Proposition 2.2}
\end{aligned}$$

By using again that

$$A' \otimes B \otimes D^\star = A^\star \otimes B \otimes D^\star \otimes (C \otimes A^\star \otimes B \otimes D^\star)^\star$$

we finally obtain

$$(D \oplus C \otimes A^\star \otimes B)^\star = D^\star \oplus D^\star \otimes C \otimes A' \otimes B \otimes D^\star$$

which proves the second identity, and also concludes the proof of (5.1).

Then, computing the matrix $A \oplus B \otimes D^\star \otimes C \in \mathbb{S}^{(n-1)\times(n-1)}$ requires one scalar star operation, and a quadratic number of scalar sum and product operations. By induction, we can then compute its star, the matrix A'. Then, (5.1) allows us to compute $M^\star$ with an additional quadratic number of scalar sum and product operations. The result follows by induction. □

5.1.2 Application to the Evaluation Problem

We fix in this section an alphabet A, a set of directions D and a continuous semiring $\mathbb{S}$. Let $\mathcal{A}$ be a layered pebble weighted automaton, G be a pointed graph and σ be a valuation with domain containing the free variables of $\mathcal{A}$. We now explain how to compute

$$[\![\mathcal{A}]\!](G,\sigma) = \bigoplus_{q_i,q_f \in Q} I_{q_i} \otimes [\![\mathcal{A}_{q_i,q_f}]\!](G,\sigma,v^{(i)},v^{(f)}) \otimes \bigoplus_{\substack{\alpha \in \mathrm{Test} \mid \\ G,\sigma,v^{(f)} \models \alpha}} F_{q_f}(\alpha)\,.$$

This is done by induction on the number of layers of the automaton $\mathcal{A}$.

In case of 0 layers, we may consider the weighted graph of configurations of $\mathcal{A}$ over a fixed pair (G,σ): its adjacency matrix has size $|Q| \times |V|$, and its $((q,v),(q',v'))$-coefficient is 0 in case $(v,v') \notin E$, and otherwise is given by

$$\bigoplus_{\substack{d \in D \mid (v,v') \in E_d \\ \alpha \in \mathrm{Test} \mid G,\sigma,v \models \alpha}} \Delta_{q,q'}(d)(\alpha)\,.$$

Remark 5.2. We suppose in the following that the set of all these coefficients, with v, v' being fixed, is computable with $\mathcal{O}(|\Delta|)$ scalar sum operations, which highly depends on the actual implementation of automata. Supposing that for every states q, q', direction d, there exists only a bounded number of tests α such that $\Delta_{q,q'}(d)(\alpha)$ would for example permit to bound this computation time by $\mathcal{O}(|Q|^2)$, supposing that D is fixed. ■

Then, notice that $[\![\mathcal{A}_{q_i,q_f}]\!](G,\sigma,v^{(i)},v^{(f)})$ is exactly the sum of the weights of the paths from configuration $(G,\sigma,q_i,v^{(i)})$ to configuration $(G,\sigma,q_f,v^{(f)})$ in this weighted graph. Hence, using Lemma 5.1, by computing the star of the adjacency matrix, and multiplying by the initial and final weights, we obtain the semantics $[\![\mathcal{A}]\!](G,\sigma)$. In total, we have

Proposition 5.3. *We can compute the semantics $[\![\mathcal{A}]\!](G,\sigma)$ of a 0-layered pebble weighted automaton $\mathcal{A}$ over a pointed graph G, for a valuation σ with domain containing the free variables of $\mathcal{A}$, with*

- $\mathcal{O}(|Q| \times |V|)$ *scalar star operations,*
- $\mathcal{O}(|Q|^3 \times |V|^3)$ *scalar products, and*
- $\mathcal{O}(|Q|^3 \times |V|^3 + |\Delta| \times |V|^2)$ *scalar sums.*

We now apply the same technique to a K-layered pebble weighted automaton.

Theorem 5.4. *We can compute the semantics $[\![\mathcal{A}]\!](G,\sigma)$ of a K-layered pebble weighted automaton $\mathcal{A}$ that may drop p distinct variables, over a pointed graph G, for a valuation σ with domain containing the free variables of $\mathcal{A}$, with*

- $\mathcal{O}(|Q| \times |V|^{p+1})$ *scalar star operations,*
- $\mathcal{O}((p+1) \times |Q| \times |\Delta| \times |V|^{p+2} + |Q|^3 \times |V|^{p+3})$ *scalar sums and products.*

Proof. We apply recursively the method previously presented in the case of 0 layers. More precisely, we denote by $\mathcal{A}^k$ the k-layered pebble weighted automaton obtained by removing from $\mathcal{A}$ all the states of layer $K,\dots,k+1$. We then compute $[\![\mathcal{A}^k_{q,q'}]\!](G,\sigma',v^{(i)},v^{(f)})$ for every states $q,q' \in \ell^{-1}(k)$ and every valuation σ' compatible with σ and the layer k: this means that we only consider valuations $\sigma' = \sigma[x_K \mapsto v_K]\cdots[x_{k+1} \mapsto v_{k+1}]$ with $x_K,\dots,x_{k+1}$ variables that may be dropped in layers $K,\dots,k+1$, respectively. Indeed, notice that the semantics $[\![\mathcal{A}^k_{q,q'}]\!](G,\sigma',v^{(i)},v^{(f)})$ is exactly the sum of the weights of the runs of $\mathcal{A}$ from configuration $(G,\sigma,q,\pi,v^{(i)})$ to configuration $(G,\sigma,q',\pi,v^{(f)})$, with intermediary states in layers no more than k, when $\sigma' = \sigma_\pi$.

For the base case, $\mathcal{A}^0$ is a 0-layered pebble weighted automaton. By applying Lemma 5.1, we can compute the weights $[\![\mathcal{A}^0_{q,q'}]\!](G,\sigma',v^{(i)},v^{(f)})$ for every states $q,q' \in \ell^{-1}(0)$ and every valuation $\sigma' = \sigma_\pi$ with π a stack of K dropped pebbles. As we consider reusable pebbles, the number of possible valuations σ' is at most $|V|^p$. Hence, all these weights can be computed with

- $\mathcal{O}(|\ell^{-1}(0)| \times |V| \times |V|^p)$ scalar star operations,
- $\mathcal{O}(|\ell^{-1}(0)|^3 \times |V|^3 \times |V|^p)$ scalar products, and
- $\mathcal{O}((|\ell^{-1}(0)|^3 \times |V|^3 + |\Delta_{|\ell^{-1}(0)\times\ell^{-1}(0)}| \times |V|^2) \times |V|^p)$ scalar sums.

Then, by supposing that all the weights for layers $0,1,\dots,k$ have been computed, the weights for layer $k+1$ can be computed by the following procedure. For every valuation σ' compatible with layer $k+1$, we consider the square matrix of length $|\ell^{-1}(k+1)| \times |V|$ with the weights of the transitions inside layer $k+1$ updated by adding the weight

$$\bigoplus_{x\in\mathrm{Var}(\mathcal{A})} \bigoplus_{q'\in\ell^{-1}(k)} \left[\bigoplus_{q\in\ell^{-1}(k)} \bigoplus_{\substack{\alpha\in\mathrm{Test}\,|\\ G,\sigma',v\models\alpha}} \Delta_{r,q}(\mathsf{drop}_x)(\alpha) \otimes [\![\mathcal{A}^k_{q,q'}]\!](G,\sigma'[x\mapsto v],v^{(i)},v^{(f)}) \right] \otimes \bigoplus_{\substack{\alpha'\in\mathrm{Test}\,|\\ G,\sigma'[x\mapsto v],v^{(f)}\models\alpha'}} \Delta_{q',r'}(\mathsf{lift})(\alpha')$$

in the coefficient $((r,v),(r',v))$. This update of the matrix can be done with some extra scalar sums and products in $\mathbb{S}$: indeed, the element in the brackets requires in total a number of sums and products bounded by $\mathcal{O}(|\ell^{-1}(k)| \times |\Delta_{|\ell^{-1}(k+1)\times\ell^{-1}(k)}|)$ whereas the external computations require an additional $\mathcal{O}(p \times |\ell^{-1}(k)| \times |\Delta_{|\ell^{-1}(k)\times\ell^{-1}(k+1)}|)$ sums and products. In total, for all the layers, vertices and valuations, these update operations require $\mathcal{O}((p+1) \times |Q| \times |\Delta| \times |V|^{p+2})$ scalar sums and products.

Afterwards, computing the star of the matrix permits to meet the requirement. For all valuations, the computation for layer $k+1$ therefore requires

- $\mathcal{O}(|\ell^{-1}(k+1)| \times |V| \times |V|^p)$ scalar star operations,
- $\mathcal{O}(|\ell^{-1}(k+1)|^3 \times |V|^3 \times |V|^p)$ scalar products, and
- $\mathcal{O}((|\ell^{-1}(k+1)|^3 \times |V|^3 + |\Delta_{|\ell^{-1}(k+1)\times\ell^{-1}(k+1)}| \times |V|^2) \times |V|^p)$ scalar sums.

We obtain a total number of operations of the shape

- $\mathcal{O}(\sum_{0\leq k\leq K} |\ell^{-1}(k)| \times |V|^{p+1})$ for the scalar star operations,
- $\mathcal{O}((p+1) \times |Q| \times |\Delta| \times |V|^{p+2} + \sum_{0\leq k\leq K} |\ell^{-1}(k)|^3 \times |V|^{p+3})$ for the products, and
- $\mathcal{O}((p+1)\times|Q|\times|\Delta|\times|V|^{p+2}+\sum_{1\leq k\leq K}(|\ell^{-1}(k)|^3\times|V|^{p+3}+|\Delta_{|\ell^{-1}(k)\times\ell^{-1}(k)}|\times|V|^{p+2}))$ for the sums,

which can be overapproximated by the bound in the theorem. □

It is important to notice that the complexity with respect to the pebble weighted automaton does not depend on the number K of layers but only on the total number of states. The number of variables occurs in the exponent but since we allow reusability, this number may be rather small. This is in the same vein as restricting the number of variables in first-order logic, without restricting the quantifier depth. Restricting the number of variables often results in much lower complexity. For instance, the complexity of the evaluation (model-checking) problem of first-order logic over relational structures drops from PSPACE to PTIME when the number of variables is bounded [Var95a, Var95b]. In our case, if we bound the number p of such variables, we obtain a polynomial complexity for the evaluation of a quantitative query, even for pebble weighted automata with a large number of layers.

In the two next sections, we will give specialized and more efficient algorithms in the case of words, trees and nested words.

5.2 Specialized Algorithm for Words

In this section, we study the evaluation problem of a K-layered pebble weighted automaton $\mathcal{A}$ over a given finite word w, represented as a graph: given a graph $G \in \mathcal{W}\text{ord}(A)$ and a valuation $\sigma\colon \text{Var} \rightharpoonup V$ with domain containing the free variables of $\mathcal{A}$, compute $[\![\mathcal{A}]\!](G,\sigma)$.

Before explaining how to evaluate efficiently pebble weighted automata, we recall how this can be done for classical finite-state automata. In case of a deterministic finite-state automaton, we can easily check whether a word w is accepted by an automaton by following the single computation in the automaton labeled by w, from left to right: at every position of the word, we only have to keep *one memory cell* containing the state reached after the current prefix of w. This memory cell can be updated with each letter in constant time using the transition function. On the overall, this leads to a complexity $\mathcal{O}(|w|)$ (in particular, independent of the automaton size).

For a non-deterministic finite-state automaton, this complexity is not achievable anymore. A first solution consists of determinizing the automaton and then applying the previous method: we obtain a complexity $\mathcal{O}(2^n + |w|)$ where n is the number of states of the non-deterministic finite-state automaton. This method cannot be extended to weighted automata, as they are not necessarily determinizable. Hence, we would rather use a dynamic programming method. We will pay the price of non-determinism by using more *memory cells* during the left-to-right computation of the runs of the automaton over the word. We use one memory cell s_q for every state q of the automaton. Cell s_q is a Boolean which is true after reading a word w if, and only if, there exists a run labeled by w leading from some initial state to state q. Cells are initially true for initial states and false otherwise. The update of the cells when reading a letter a is done by the multiplication of the row vector of cells (of dimension $1 \times |Q|$) with the adjacency matrix of the transitions of the automaton labeled with letter a. Hence, each update requires a number of operations $\mathcal{O}(n^2)$, leading to an overall complexity of $\mathcal{O}(n^2|w|)$. This method can naturally be extended to the weighted setting using memory cells containing values in $\mathbb{S}$. More precisely, cell s_q holds the sum of weights of runs starting from some initial state and ending in state q. The update now uses the weighted transition matrix and can be done with the same complexity.

We now explain how to evaluate layered pebble weighted automata with similar methods, in particular, permitting to avoid constructing the whole $(|Q| \times |V|)$-square matrix as in the previous section, and hence saving some scalar operations in $\mathbb{S}$.

Theorem 5.5. *We can compute the semantics $[\![\mathcal{A}]\!](w,\sigma)$ of a K-layered pebble weighted automaton $\mathcal{A}$ that may drop p distinct variables, over a nonempty finite word $w \in A^+$, for a valuation σ with domain containing the free variables of $\mathcal{A}$,*

- *with $\mathcal{O}(|Q| \times |w|^{p+1})$ scalar star operations,*
- *$\mathcal{O}((p+1) \times |Q|^3 \times |w|^{p+1})$ scalar products, and*
- *$\mathcal{O}((p+1) \times |Q|^3 \times |w|^{p+1} + |\Delta| \times |w|^{p+1})$ scalar sums.*

Proof. The navigation of automata is resolved by adding more memory cells (a quadratic number with respect to $n = |Q|$), namely those that compute weights of the back-loops and forth-loops. We deal with layers inductively. In the whole proof, we fix a word $w \in A^+$ encoded in a graph $G \in \mathcal{W}\mathrm{ord}(A)$ with[1] $V = \{1, \dots, |w|\}$, and a valuation σ with domain containing the free variables of $\mathcal{A}$.

Recall that, for all valuations $\sigma'\colon \mathrm{Var} \rightharpoonup V$, layers $k \in \{0, \dots, K\}$ and states $q, q' \in \ell^{-1}(k)$, $[\![\mathcal{A}_{q,q'}]\!](G,\sigma')$ is the sum of weights of the runs from configurations $(G,\sigma',q,1,\varepsilon)$ to $(G,\sigma',q',|w|,\varepsilon)$: observe that the stack of pebbles is empty at the beginning of these runs, hence they stay in layers $k, k-1, \dots, 0$. In the following, these values (and others that will be defined later) will be grouped into matrices indexed by subsets of states (and no longer by pairs of states and vertices). In particular, we let

$$B^{(k)}_{\sigma'} = ([\![\mathcal{A}_{q,q'}]\!](G,\sigma',1,|w|))_{q,q' \in \ell^{-1}(k)}$$

be such a matrix indexed by $\ell^{-1}(k) \times \ell^{-1}(k)$.

Let $k \in \{0, \dots, K\}$ be a layer of the automaton. If $k > 0$, we assume by induction that for all valuations σ' compatible with σ and layer $k-1$ (with the same definition as in the proof of Theorem 5.4) we have already computed the matrices $B^{(k-1)}_{\sigma'}$. For each valuation σ' compatible with σ and layer k, we will compute the matrix $B^{(k)}_{\sigma'}$ *reading the graph G* from left to right. Formally, for $1 \leq i \leq |V|$, we define the matrix $B^{\to i}_{\sigma'} = (B^{\to i}_{\sigma,q,q'})_{q,q' \in \ell^{-1}(k)}$ where $B^{\to i}_{\sigma',q,q'}$ is the sum of weights of the runs from configuration $(G,\sigma',q,\varepsilon,1)$ to $(G,\sigma',q',\varepsilon,i)$ with intermediary configurations of the form (G,σ',r,π,j) with $\pi \neq \varepsilon$ or $j \leq i$ (see left of Figure 5.1). These are the runs which move from the beginning of the word to position i, staying on the left of i, unless some pebbles are currently dropped (if a pebble is dropped, then automaton can read the whole word). In order to compute inductively $B^{\to i}_{\sigma'}$ using $B^{\to i-1}_{\sigma'}$, we also define the matrices $B^{\curvearrowleft i}_{\sigma'} = (B^{\curvearrowleft i}_{\sigma',q,q'})_{q,q' \in \ell^{-1}(k)}$ where $B^{\curvearrowleft i}_{\sigma',q,q'}$ is the sum of weights of the runs from configuration $(G,\sigma',q,\varepsilon,i)$ to $(G,\sigma',q',\varepsilon,i)$ with intermediary configurations of the form (G,σ',r,π,j) with $\pi \neq \varepsilon$ or $j \leq i$ (see right of Figure 5.1). Again, these runs stay on the left of their starting position except possibly when they drop pebbles.

By definition we have $B^{\to 1}_{\sigma'} = B^{\curvearrowleft 1}_{\sigma'}$. Moreover, for $1 < i \leq |V|$, a run from position 1 to position i staying on the left of i can be decomposed as a run from position 1 to position $i-1$ followed by a right move, followed by a back-loop over position i:

$$B^{\to i}_{\sigma'} = B^{\to i-1}_{\sigma'} \otimes M^{\to,(k)}_{\sigma',i-1} \otimes B^{\curvearrowleft i}_{\sigma'}\,.$$

Here and in the following, we will denote by $M^{d,(k)}_{\sigma',i}$ the $\ell^{-1}(k) \times \ell^{-1}(k)$-matrix with (q,q')-coefficient given by

$$\bigoplus_{\alpha \in \mathrm{Test} \mid G,\sigma',i \models \alpha} \Delta_{q,q'}(d)(\alpha)$$

[1] We locally change in this proof the encoding of the positions of a word from $\{0, \dots, |w|-1\}$ to $\{1, \dots, |w|\}$ in order to ease some notations.

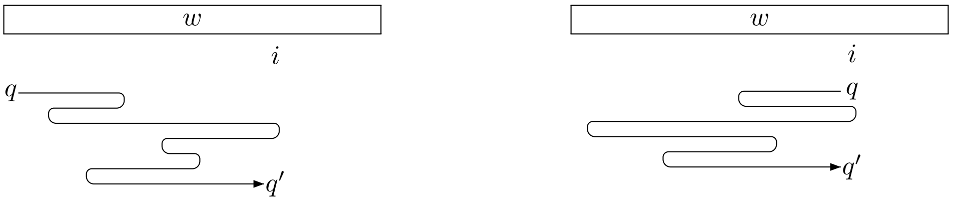


Figure 5.1: Runs of a navigating automaton

for $d \in \{\rightarrow, \leftarrow\}$: this coefficient denotes the weight of taking a transition with move d from state q to state q' on position i with current valuation σ'. We will also need similar matrices for drop and lift moves. We denote by $M^{\mathsf{drop}_x,(k)}_{\sigma',i}$ the $\ell^{-1}(k) \times \ell^{-1}(k-1)$-matrix with (q,q')-coefficient given by

$$\bigoplus_{\alpha \in \mathrm{Test} | G,\sigma',i \models \alpha} \Delta_{q,q'}(\mathsf{drop}_x)(\alpha)\,.$$

Lift transitions only occur on position $|w|$ of the word, hence we do not need to give a vertex to define similar matrix for lift: let $M^{\mathsf{lift},(k)}_{\sigma'}$ be the $\ell^{-1}(k-1) \times \ell^{-1}(k)$-matrix with (q,q')-coefficient given by

$$\bigoplus_{\alpha \in \mathrm{Test} | G,\sigma',|w| \models \alpha} \Delta_{q,q'}(\mathsf{lift})(\alpha)\,.$$

Notice that the precomputation of all these matrices (for every position and valuation) can be computed with an additional number of scalar sums bounded above by $\mathcal{O}(|\Delta| \times |w|^{p+1})$.

We now explain how to compute the matrices $B^{\curvearrowleft i}_{\sigma'}$ inductively from left to right. At the bottom layer ($k = 0$) no further pebble can be dropped so that we get

$$\begin{aligned} B^{\curvearrowleft 1}_{\sigma'} &= \mathrm{Id} \\ B^{\curvearrowleft i}_{\sigma'} &= \left(M^{\leftarrow,(k)}_{\sigma',i} \otimes B^{\curvearrowleft i-1}_{\sigma'} \otimes M^{\rightarrow,(k)}_{\sigma',i-1}\right)^{\star} && \text{if } 1 < i \leq |w|\,. \end{aligned}$$

If $k > 0$, the run may immediately drop some pebble of name $x \in \mathrm{Var}$ on position $1 \leq i \leq |w|$ resulting in the *nested* computation of

$$N^{(k)}_{\sigma',i} = \sum_{x \in \mathrm{Var}(\mathcal{A})} M^{\mathsf{drop}_x,(k)}_{\sigma',i} \otimes B^{(k-1)}_{\sigma'[x \mapsto i]} \otimes M^{\mathsf{lift},(k)}_{\sigma'[x \mapsto i]}\,.$$

Notice that matrix $B^{(k-1)}_{\sigma'[x \mapsto i]}$ has been precomputed since $\sigma'[x \mapsto i]$ is compatible with σ and layer $k-1$ (σ' being compatible with σ and layer k). If $i > 1$, the run may also start by moving left, hence we obtain

$$\begin{aligned} B^{\curvearrowleft 1}_{\sigma'} &= \left(N^{(k)}_{\sigma',1}\right)^{\star} \\ B^{\curvearrowleft i}_{\sigma'} &= \left(N^{(k)}_{\sigma',i} \oplus M^{\leftarrow,(k)}_{\sigma',i} \otimes B^{\curvearrowleft i-1}_{\sigma'} \otimes M^{\rightarrow,(k)}_{\sigma',i-1}\right)^{\star} && \text{if } 1 < i \leq |w| \end{aligned}$$

Finally, once all these matrices have been obtained, we can compute the behavior of layer k with the formula

$$B^{(k)}_{\sigma'} = B^{\rightarrow |w|}_{\sigma'}\,.$$

To conclude this proof, it remains to count the number of scalar operations in the whole computation. For this, we first count the number of matrix operations (sum, product and star) and then infer the number of scalar operations assuming standard algorithms on matrices (quadratic for sum, cubic for product and cubic for star using Lemma 5.1).

Fix a layer k. For all valuations σ' compatible with σ and layer k (there are at most $|w|^p$ such valuations), matrices $B_{\sigma'}^{\to i}$, $B_{\sigma'}^{\curvearrowleft i}$ must be computed for every $1 \leq i \leq |w|$. When $k = 0$, the total number of matrix operations (sum, product, star) is $\mathcal{O}(|w|^p \times |w|)$, which corresponds to $\mathcal{O}(|\ell^{-1}(0)| \times |w|^{p+1})$ scalar star operations and $\mathcal{O}(|\ell^{-1}(0)|^3 \times |w|^{p+1})$ scalar sum and product operations. For $k > 0$, the computation of $N_{\sigma',i}^{(k)}$ takes $\mathcal{O}(p(|\ell^{-1}(k)||\ell^{-1}(k-1)|^2 + |\ell^{-1}(k)|^2|\ell^{-1}(k-1)|))$ scalar sums and products for each σ' and i. Hence, the total number of scalar sum and product operations for computing all matrices $B_{\sigma'}^{\to i}$, $B_{\sigma'}^{\curvearrowleft i}$ of layer k is now $\mathcal{O}(p(|\ell^{-1}(k)||\ell^{-1}(k-1)|^2 + |\ell^{-1}(k)|^2|\ell^{-1}(k-1)| + |\ell^{-1}(k)|^3)|w|^{p+1})$, whereas the total number of scalar star operations is $\mathcal{O}(|\ell^{-1}(k)| \times |w|^{p+1})$.

Summing over all $k \leq K$, we get a total number of $\mathcal{O}((p+1)|Q|^3|w|^{p+1})$ scalar sum and products operations since

$$|\ell^{-1}(0)|^3 + \sum_{k=1}^{K} |\ell^{-1}(k)||\ell^{-1}(k-1)|^2 + |\ell^{-1}(k)|^2|\ell^{-1}(k-1)| + |\ell^{-1}(k)|^3 \leq |Q|^3$$

and similarly a total number of $\mathcal{O}(|Q| \times |w|^{p+1})$ scalar star operations. □

Indeed, we see below that it is possible to decrease again the complexity, if we start with a *strongly* layered pebble weighted automaton.

Theorem 5.6. *We can compute the semantics $[\![\mathcal{A}]\!](w,\sigma)$ of a K-strongly layered pebble weighted automaton $\mathcal{A}$ that may drop p distinct variables, over a word $w \in A^+$, for a valuation σ with domain containing the free variables of $\mathcal{A}$,*

- *with $\mathcal{O}(|Q| \times |w|^{\max(p,1)})$ scalar star operations,*
- *$\mathcal{O}(|Q|^3 \times |w|^{\max(p,1)})$ scalar products, and*
- *$\mathcal{O}(|Q|^3 \times |w|^{\max(p,1)} + |\Delta| \times |w|^{\max(p,1)})$ scalar sums.*

Proof. The idea is to decrease by 1 the exponent of $|w|$ in the complexity by reducing the total number of valuations σ' for which we must compute the matrices used in the proof of Theorem 5.8. Indeed, knowing the unique pebble that may be dropped in a given layer permits to forget the precise position of this pebble when computing the matrices of this layer.

Let $0 \leq k \leq K$ be a layer of $\mathcal{A}$. In addition to matrices defined in the proof of Theorem 5.8, we introduce new matrices $B_{\sigma'}^{i \curvearrowright}$ and $B_{\sigma'}^{i \to}$ in order to compute the sum of weights of runs which stay on the right of their starting position (except when they drop pebbles). For every states $q, q' \in \ell^{-1}(k)$ and $1 \leq i \leq |w|$, we denote by $B_{\sigma',q,q'}^{i \curvearrowright}$ the sum of weights of the runs from configuration $(G, \sigma', q, \varepsilon, i)$ to $(G, \sigma', q', \varepsilon, i)$ with intermediary configurations of the form (G, σ', r, π, j) with $\pi \neq \varepsilon$ or $j \geq i$. Finally, we denote by $B_{\sigma',q,q'}^{i \to}$ the sum of weights of the runs from configuration $(G, \sigma', q, \varepsilon, i)$ to $(G, \sigma', q', \varepsilon, |w|)$ with intermediary configurations of the form (G, σ', r, π, j) with $\pi \neq \varepsilon$ or $j \geq i$.

Hence, we will compute twice as many matrices, however we will gain in complexity by replacing the usual valuation σ' by its restriction σ'' to the pebbles in $\mathrm{Var}(\mathcal{A}) \backslash \{x_k\}$, where we denote by x_k the pebble name being dropped in layer k. Indeed, with $j = \sigma'(x_k)$, we can compute the matrix $B_{\sigma'}^{(k)}$ by splitting the word w into three parts: positions appearing before j, position j and positions appearing after j. Hence, for $1 < j = \sigma'(x_k) < |w|$, we can compute $B_{\sigma'}^{(k)}$ with (notice that $\sigma''[x_k \mapsto j] = \sigma'$)

$$\begin{aligned} B_{\sigma'}^{(k)} = B_{\sigma''}^{\to j-1} \otimes M_{\sigma'',j-1}^{\to,(k)} \otimes \Big(& \quad M_{\sigma',j}^{\mathsf{drop}_{x_k},(k)} \otimes B_{\sigma'}^{(k-1)} \otimes M_{\sigma'}^{\mathsf{lift},(k)} \\ & \oplus M_{\sigma',j}^{\leftarrow,(k)} \otimes B_{\sigma''}^{\curvearrowleft j-1} \otimes M_{\sigma'',j-1}^{\to,(k)} \\ & \oplus M_{\sigma',j}^{\to,(k)} \otimes B_{\sigma''}^{j+1 \curvearrowright} \otimes M_{\sigma'',j+1}^{\leftarrow,(k)} \Big)^{\star} \otimes M_{\sigma',j}^{\to,(k)} \otimes B_{\sigma''}^{j+1 \to} . \end{aligned}$$

When $j = 1$, the formula becomes

$$\begin{aligned}B^{(k)}_{\sigma'} = \Big(\quad & M^{\mathsf{drop}_{x_k},(k)}_{\sigma',1} \otimes B^{(k-1)}_{\sigma'} \otimes M^{\mathsf{lift},(k)}_{\sigma'} \\ & \oplus M^{\rightarrow,(k)}_{\sigma',1} \otimes B^{2\circlearrowright}_{\sigma''} \otimes M^{\leftarrow,(k)}_{\sigma'',2} \Big)^{\star} \otimes M^{\rightarrow,(k)}_{\sigma',1} \otimes B^{2\rightarrow}_{\sigma''}\end{aligned}$$

whereas for $j = |w|$, it is

$$\begin{aligned}B^{(k)}_{\sigma'} = B^{\rightarrow|w|-1}_{\sigma''} \otimes M^{\rightarrow,(k)}_{\sigma'',|w|-1} \otimes \Big(\quad & M^{\mathsf{drop}_{x_k},(k)}_{\sigma',|w|} \otimes B^{(k-1)}_{\sigma'} \otimes M^{\mathsf{lift},(k)}_{\sigma'} \\ & \oplus M^{\leftarrow,(k)}_{\sigma',|w|} \otimes B^{\circlearrowleft|w|-1}_{\sigma''} \otimes M^{\rightarrow,(k)}_{\sigma'',|w|-1} \Big)^{\star} .\end{aligned}$$

It remains to compute the matrices $B^{\rightarrow i}_{\sigma''}$ and $B^{\circlearrowleft i}_{\sigma''}$ for $1 \leq i \leq |w|$ (in particular we will use them in the previous formulae for $i \in [1 \mathinner{..} j-1]$), and the matrices $B^{i\circlearrowright}_{\sigma''}$ and $B^{i\rightarrow}_{\sigma''}$ for $1 \leq i \leq |w|$ (we use them for $i \in [j+1 \mathinner{..} |w|]$). First, if $k > 0$ then a pebble of name $x_k \in \mathrm{Var}$ may be dropped on position $1 \leq i \leq |w|$ (with $i \neq j$) resulting in the *nested* computation of

$$N^{(k)}_{\sigma'',i} = M^{\mathsf{drop}_{x_k},(k)}_{\sigma'',i} \otimes B^{(k-1)}_{\sigma''[x_k \mapsto i]} \otimes M^{\mathsf{lift},(k)}_{\sigma''[x_k \mapsto i]} .$$

We let $N^{(0)}_{\sigma'',i} = 0$. Then, it is easy to verify that for $1 < i \leq |w|$:

$$\begin{aligned} B^{\circlearrowleft 1}_{\sigma''} &= \left(N^{(k)}_{\sigma'',1}\right)^{\star} \qquad \text{and} \qquad B^{\rightarrow 1}_{\sigma''} = B^{\circlearrowleft 1}_{\sigma''} \\ B^{\circlearrowleft i}_{\sigma''} &= \left(N^{(k)}_{\sigma'',i} \oplus M^{\leftarrow,(k)}_{\sigma'',i} \otimes B^{\circlearrowleft i-1}_{\sigma''} \otimes M^{\rightarrow,(k)}_{\sigma'',i-1}\right)^{\star} \\ B^{\rightarrow i}_{\sigma''} &= B^{\rightarrow i-1}_{\sigma''} \otimes M^{\rightarrow,(k)}_{\sigma'',i-1} \otimes B^{\circlearrowleft i}_{\sigma''} \end{aligned}$$

and for $1 \leq i < |w|$:

$$\begin{aligned} B^{|w|\circlearrowright}_{\sigma''} &= \left(N^{(k)}_{\sigma'',|w|}\right)^{\star} \qquad \text{and} \qquad B^{|w|\rightarrow}_{\sigma''} = B^{|w|\circlearrowright}_{\sigma''} \\ B^{i\circlearrowright}_{\sigma''} &= \left(N^{(k)}_{\sigma'',i} \oplus M^{\rightarrow,(k)}_{\sigma'',i} \otimes B^{i+1\circlearrowright}_{\sigma''} \otimes M^{\leftarrow,(k)}_{\sigma'',i+1}\right)^{\star} \\ B^{i\rightarrow}_{\sigma''} &= B^{i\circlearrowright}_{\sigma''} \otimes M^{\rightarrow,(k)}_{\sigma'',i} \otimes B^{i+1\rightarrow}_{\sigma''} . \end{aligned}$$

The computation of the four types of matrices, for all valid positions i and all partial valuations σ'', requires globally $\mathcal{O}(|w|^{p-1} \times |w|)$ matrix operations (unless $p = 0$, in which case $\mathcal{O}(|w|)$ matrix operations are required). Notice also that the precomputation of the matrices $M^{d,(k)}_{\sigma'',i}$ has to be done only for partial valuations σ'' and i, and also for the complete valuations $\sigma''[x \mapsto j]$ and $i = j$, which requires only $\mathcal{O}(|\Delta| \times |w|^{\max(p,1)})$ scalar sums. Hence, this improves the overall complexity as announced. □

Notice that if $p \leq 1$ then any K-layered pebble weighted automaton is *strongly* K-layered. In this case, we get an evaluation algorithm using $\mathcal{O}(|Q| \times |w|)$ scalar star operations and $\mathcal{O}(|Q|^3 \times |w|)$ scalar products and $\mathcal{O}(|Q|^3 \times |w| + |\Delta| \times |w|)$ scalar sums.

5.3 Extension to Trees

We now study the evaluation problem of layered pebble weighted automata over ranked trees. Contrary to words, we modeled ranked trees by pointed graphs having their initial and final vertex equal to the root. Hence, a run of a 0-layered pebble weighted automaton is a sequence of configurations which follows a path from the root to the root. Hence, we only need to compute the loop matrices from the previous proofs, which permits to even simplify the proof.

Theorem 5.7. *We can compute the semantics $[\![\mathcal{A}]\!](t,\sigma)$ of a K-layered pebble weighted automaton $\mathcal{A}$ that may drop p distinct variables, over a ranked tree t, for a valuation σ with domain containing the free variables of $\mathcal{A}$, with*

- $\mathcal{O}(|Q| \times |t|^{p+1})$ *scalar star operations,*
- $\mathcal{O}((p+1) \times |Q|^3 \times |t|^{p+1})$ *scalar products, and*
- $\mathcal{O}((p+1) \times |Q|^3 \times |t|^{p+1} + |\Delta| \times |t|^{p+1})$ *scalar sums.*

Proof. Again, we reason by induction on the layers of the automaton, and, we use similar notations as in the case of words. In particular, we consider a pointed graph $G \in \mathcal{T}\mathrm{ree}(A)$, that models the tree t.

Let $k \in \{0,\dots,K\}$ be a layer of the automaton. If $k > 0$, we assume by induction that for all valuations σ' compatible with σ and layer $k-1$ (with the same definition as in the proof of Theorem 5.4) we have already computed the matrices $B^{(k-1)}_{\sigma'}$. For each valuation σ' compatible with σ and layer k, we now compute the matrix $B^{(k)}_{\sigma'}$ by a *bottom-up* procedure on the tree. Formally, we define the matrices $B^{u\circlearrowright}_{\sigma'} = (B^{u\circlearrowright}_{\sigma',q,q'})_{q,q'\in\ell^{-1}(k)}$ where $B^{u\circlearrowright}_{\sigma',q,q'}$ is the sum of weights of the runs from configuration $(G,\sigma',q,\varepsilon,u)$ to $(G,\sigma',q',\varepsilon,u)$ with intermediary configurations of the form (G,σ',r,π,v) with $\pi \neq \varepsilon$ or $v = uu'$ with $u' \in \mathbb{N}^\star$ (i.e., v in the subtree rooted in u). Again, these runs stay in the subtree of vertex u except when they drop pebbles.

For vertices u that are leaves, we have $B^{u\circlearrowright}_{\sigma'} = \mathrm{Id}$ for layer $k = 0$, and $B^{u\circlearrowright}_{\sigma'} = \left(N^{(k-1)}_{\sigma',u}\right)^\star$ otherwise. For vertices u that are internal nodes, if $k = 0$, a loop under node u is composed of a succession of move to one of the children of u, a loop under that particular child, and a move back to u. Hence, we let

$$B^{u\circlearrowright}_{\sigma'} = \left(\bigoplus\nolimits_{0\leq i\leq \mathrm{ar}(\lambda(u))-1} M^{\downarrow_i,(k)}_{\sigma',u} \otimes B^{ui\circlearrowright}_{\sigma'} \otimes M^{\uparrow_i,(k)}_{\sigma',ui}\right)^\star$$

For $k > 0$, we must also consider the possibility to drop a pebble on u so that

$$B^{u\circlearrowright}_{\sigma'} = \left(N^{(k-1)}_{\sigma',u} \oplus \bigoplus\nolimits_{0\leq i\leq \mathrm{ar}(\lambda(u))-1} M^{\downarrow_i,(k)}_{\sigma',u} \otimes B^{ui\circlearrowright}_{\sigma'} \otimes M^{\uparrow_i,(k)}_{\sigma',ui}\right)^\star$$

Once all these matrices are computed, we obtain the behavior of layer k with

$$B^{(k)}_{\sigma'} = B^{\varepsilon\circlearrowright}_{\sigma'}$$

since the root ε is both the initial vertex and the final one. The count of the number of operations is similar to the previous cases. □

We believe that ideas similar to those used for words in Theorem 5.6 would permit to decrease the complexity in the case of trees. Indeed, a vertex of a tree separates the tree into two parts: the subtree rooted by the vertex, and its context, i.e., the tree where the previous subtree has been removed and the vertex replaced by a hole. Mimicking the procedure of Theorem 5.6 would require to introduce several types of matrices for every vertex v: loops in the subtree of v (the ones introduced in the previous proof), loops in the context, loops below the root in the context of v and paths from the root to v in the context of v and vice versa. Computing those five matrices for every vertex v requires to carefully order the set vertices, as a bottom-up procedure would not be enabled anymore. Full details of this procedure are left for future work.

5.4 Extension to Nested Words

Finally, we consider the case of nested words. A strategy could be to encode nested words into binary trees, in a way that layered pebble weighted automata could be translated faithfully. This is not at all trivial, since we consider walking automata and that usual encodings do

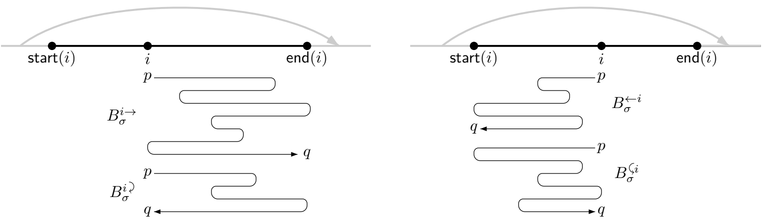

start(i)
i
end(i)
p
$B^{i\to}_{\sigma}$
q
p
$B^{i\circlearrowright}_{\sigma}$
q
start(i)
i
end(i)
p
$B^{\leftarrow i}_{\sigma}$
q
p
$B^{\circlearrowleft i}_{\sigma}$
q

Figure 5.2: Representation of the four types of matrices

not preserve the paths between vertices: indeed, such paths must be translated in automata in a non ambiguous way to preserve the semantics. Instead of this encoding, we give a direct proof for nested words. This is also a good opportunity to illustrate the possibility to extend the algorithm we applied for words and trees for more complex classes of graphs of bounded degree: the key point is to consider carefully the different *looping paths* of a graph. Nested words must then be seen as a complicated enough example permitting to illustrate this reasoning.

Theorem 5.8. *We can compute the semantics* $[\![\mathcal{A}]\!](G,\sigma)$ *of a* K*-layered pebble weighted automaton* $\mathcal{A}$ *that may drop* p *distinct variables, over a pointed graph* $G \in \mathcal{N}\mathrm{est}(A)$ *modeling a nested word, for a valuation* σ *with domain containing the free variables of* $\mathcal{A}$*, with*

- $\mathcal{O}(|Q| \times |V|^{p+1})$ *scalar star operations,*
- $\mathcal{O}((p+1) \times |Q|^3 \times |V|^{p+1})$ *scalar products, and*
- $\mathcal{O}((p+1) \times |Q|^3 \times |V|^{p+1} + |\Delta| \times V^{p+1})$ *scalar sums.*

Proof. In the whole proof, we fix a nested word $G \in \mathcal{N}\mathrm{est}(A)$, with set of vertices $V = \{1, \dots, |V|\}$, and a valuation σ with domain containing the free variables of $\mathcal{A}$. We follow the same basic idea used to evaluate pebble weighted automata over words, namely computing matrices of weights for partial runs. The navigation is resolved by computing simultaneously matrices of weights of the back-loops and forth-loops, whereas we deal with layers inductively. Finally, we deal with call-return edges by using a hierarchical order based on the call-depth to compute the different matrices. Hence, for every position $i \in V$ we consider the pair $(\mathsf{start}(i), \mathsf{end}(i))$ of start and end positions as follows

$$\begin{aligned} \mathsf{start}(i) &= \min\{j \in V \mid j \leq i \wedge \forall \ell \quad j \leq \ell \leq i \implies \mathsf{c\text{-}d}(\ell) \geq \mathsf{c\text{-}d}(i)\} \\ \mathsf{end}(i) &= \max\{j \in V \mid i \leq j \wedge \forall \ell \quad i \leq \ell \leq j \implies \mathsf{c\text{-}d}(\ell) \geq \mathsf{c\text{-}d}(i)\} \end{aligned}$$

For positions of call-depth 0, the start position is 1 whereas the end position is $|V|$. For positions of call-depth at least 1 (see Figure 5.2), the start position is the linear successor of the closest call in the past such that its matched return is after position j, whereas its end position is the linear predecessor of this return position.

Let $\mathcal{A} = (Q, A, \{\rightarrow, \leftarrow, \curvearrowleft, \curvearrowright\}, I, \Delta, F)$ be a K-layered pebble weighted automaton. For every layer $k \in \{0, \dots, K\}$, we again want to compute the matrix $B^{(k)}_{\sigma'}$.

Fix a layer $k \in \{0, \dots, K\}$ of the automaton. Suppose by induction that we have already computed matrices $B^{(k-1)}_{\sigma'}$ for every valuation σ' compatible with σ and layer $k-1$. For a valuation σ' compatible with σ and layer k, the matrix $B^{(k)}_{\sigma'}$ will be obtained by the computation of four types of matrices for every position (see Figure 5.2). For example, $B^{i\to}_{\sigma',p,q}$ (respectively, $B^{i\circlearrowright}_{\sigma',p,q}$) is the sum of weights of the runs from configuration $(G, \sigma', p, \varepsilon, i)$ to $(G, \sigma', q, \varepsilon, \mathsf{end}(i))$ (respectively, $(G, \sigma', q, \varepsilon, i)$) with intermediary configurations of the form (G, σ', r, π, j) with $\pi \neq \varepsilon$ or $i \leq j \leq \mathsf{end}(i)$.

We compute these four types of matrices for every position i by decreasing value of call-depth. Suppose this has been done for every position of call-depth greater than d. We describe how to compute matrices $B^{i\to}_{\sigma'}$ and $B^{i\circlearrowright}_{\sigma'}$ for every position i of call-depth d, by decreasing values of i.[2] Similarly, matrices $B_{\sigma'}^{\leftarrow i}$ and $B_{\sigma'}^{\circlearrowleft i}$ can be computed by increasing values of positions i having call-depth d.

We extend the definition of matrix $M_{\sigma',i}^{d,(k)}$ to direction $d \in \{\curvearrowright, \curvearrowleft\}$ in a straighforward manner.

There are four distinct cases:

If $i = \mathsf{end}(i)$: The only way to loop on the right of i, staying on the left of $\mathsf{end}(i) = i$ is to drop a pebble a certain number of times. With the same definition of the matrix $N_{\sigma',i}$, we have

$$B_{\sigma'}^{i\circlearrowright} = \left(N_{\sigma',i}\right)^{\star} = B_{\sigma'}^{i\rightarrow} \,.$$

If $i < \mathsf{end}(i)$ and i is not a call: Then, looping on the right of i either starts by dropping a pebble over i, or starts with a right move, followed by a loop on the right of $i+1$ and a left move. All of this may be iterated using a star operation:

$$B_{\sigma'}^{i\circlearrowright} = \left(N_{\sigma',i} \oplus M_{\sigma',i}^{\rightarrow,(k)} \otimes B_{\sigma'}^{i+1\circlearrowright} \otimes M_{\sigma',i+1}^{\leftarrow,(k)}\right)^{\star} .$$

Moving to the right of position i, until $\mathsf{end}(i)$, can be decomposed as a loop on the right of i, followed by a right move (from that point, we will not reach position i anymore) and a run from $i+1$ to $\mathsf{end}(i+1) = \mathsf{end}(i)$:

$$B_{\sigma'}^{i\rightarrow} = B_{\sigma'}^{i\circlearrowright} \otimes M_{\sigma',i}^{\rightarrow,(k)} \otimes B_{\sigma'}^{i+1\rightarrow} \,.$$

If $i < \mathsf{end}(i)$ and $i \curvearrowright i+1$: The situation is very similar except that there are two moves leading from i to $i+1$ and two moves leading from $i+1$ to i.

$$\begin{aligned}
B_{\sigma'}^{i\circlearrowright} &= \left(N_{\sigma',i} \oplus (M_{\sigma',i}^{\rightarrow,(k)} \oplus M_{\sigma',i}^{\curvearrowright,(k)}) \otimes B_{\sigma'}^{i+1\circlearrowright} \otimes (M_{\sigma',i+1}^{\leftarrow,(k)} \oplus M_{\sigma',i+1}^{\curvearrowleft,(k)})\right)^{\star} \\
B_{\sigma'}^{i\rightarrow} &= B_{\sigma'}^{i\circlearrowright} \otimes (M_{\sigma',i}^{\rightarrow,(k)} \oplus M_{\sigma',i}^{\curvearrowright,(k)}) \otimes B_{\sigma'}^{i+1\rightarrow} \,.
\end{aligned}$$

If $i < \mathsf{end}(i)$ and $i \curvearrowright j$ with $j \neq i+1$: Looping on the right of i consists of either (1) dropping a pebble over i, or (2) looping on the right of i without ever reaching j, or (3) going to j, looping on j staying between positions $i+1$ and $\mathsf{end}(i)$, and going back to i without reaching position j again. All of this is again possibly iterated using a star operation. Notice that positions $i+1$ and $j-1$ have a call-depth greater than d, hence their four types of matrices have been computed previously. We have $\mathsf{end}(i) = \mathsf{end}(j)$, $\mathsf{end}(i+1) = j-1$ and $\mathsf{start}(j-1) = i+1$. First we define a matrix for the runs going from i to j, another one for those looping over j, and a last one for those going from j to i:

$$\begin{aligned}
\text{Goto-Return}_{\sigma',i} &= M_{\sigma',i}^{\curvearrowright,(k)} \oplus M_{\sigma',i}^{\rightarrow,(k)} \otimes B_{\sigma'}^{i+1\rightarrow} \otimes M_{\sigma',j-1}^{\rightarrow,(k)} \\
\text{Loop-Return}_{\sigma',j} &= \left(B_{\sigma'}^{j\circlearrowright} \otimes M_{\sigma',j}^{\leftarrow,(k)} \otimes B_{\sigma'}^{\circlearrowleft j-1} \otimes M_{\sigma',j-1}^{\rightarrow,(k)}\right)^{\star} \otimes B_{\sigma'}^{j\circlearrowright} \\
\text{Goto-Call}_{\sigma',j} &= M_{\sigma',j}^{\curvearrowleft,(k)} \oplus M_{\sigma',j}^{\leftarrow,(k)} \otimes B_{\sigma'}^{\leftarrow j-1} \otimes M_{\sigma',i+1}^{\leftarrow,(k)} \,.
\end{aligned}$$

Notice that the product of these three matrices generates exactly the runs described above in (3). Runs looping on the right of i are then computed by

$$\begin{aligned}
B_{\sigma'}^{i\circlearrowright} = \Big[&N_{\sigma',i} \oplus M_{\sigma',i}^{\rightarrow,(k)} \otimes B_{\sigma'}^{i+1\circlearrowright} \otimes M_{\sigma',i+1}^{\leftarrow,(k)} \\
&\oplus \text{Goto-Return}_{\sigma',i} \otimes \text{Loop-Return}_{\sigma',j} \otimes \text{Goto-Call}_{\sigma',j}\Big]^{\star} .
\end{aligned}$$

[2] These positions can be partitioned according to their associated start and end positions, and computed independently.

Finally, to go from i to $\mathsf{end}(i)$, we split the runs considering the last time we reach position i and the last time we visit a position on the left of j:

$$B^{i\rightarrow}_{\sigma'} = B^{i\curvearrowleft}_{\sigma'} \otimes \text{Goto-Return}_{\sigma',i} \otimes \left(B^{j\curvearrowleft}_{\sigma'} \otimes M^{\leftarrow,(k)}_{\sigma',j} \otimes B^{\curvearrowright j-1}_{\sigma'} \otimes M^{\rightarrow,(k)}_{\sigma',j-1}\right)^{\star} \otimes B^{j\rightarrow}_{\sigma'} \,.$$

If $i = 1$ then $\mathsf{end}(i) = |V|$ and we have $B^{(k)}_{\sigma'} = B^{1\rightarrow}_{\sigma'}$. Hence, we have computed the behavior of layer k.

Similar counts as for words permits to show the announced complexity of this algorithm. □

Notice that if the nested word is in fact a word, our algorithm only needs to compute the two sets of matrices $B^{i\rightarrow}_{\sigma'}$ and $B^{i\curvearrowleft}_{\sigma'}$ with a backward visit of the positions of the word. This is indeed a slightly different algorithm than the one presented for words where the positions are visited in a forward manner.

CHAPTER 6

Logical Specifications

Tweedledum and Tweedledee
Agreed to have a battle!
For Tweedledum said Tweedledee
Had spoiled his nice new rattle.

Just then flew down a monstrous crow,
As black as a tar-barrel!
Which frightened both the heroes so,
They quite forgot their quarrel.

'I know what you're thinking about,' said Tweedledum; 'but it isn't so, nohow.'
'Contrariwise,' continued Tweedledee, 'if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic.'

Lewis Carroll, *Through the Looking Glass*

In their book [CE11], the authors motivate the introduction of monadic logics over graphs by claiming:

> Monadic second-order formulas are even more important for specifying sets of graphs than for specifying languages because there is no convenient notion of graph automaton.

Indeed, monadic logics may be the foundations of a theory of regular languages of graphs. As a special case, for finite or infinite words and trees, those languages coincide with the ones recognized by some automata models. This is a hint for the robustness of this class of languages. In the weighted setting, weighted extensions of monadic logics have been defined [DG09] that do not coincide with any of the weighted automata models that have been discovered so far. Indeed, it was shown by [DG09] that a very restricted fragment of their logic generates exactly the languages recognized by weighted automata (one-way and without pebbles) over finite words, namely the *syntactically restricted monadic second order logic*. This logical characterization has been generalized to various other settings like finite ranked trees [DV06], and unranked trees [DV06], nested words [DP12], or infinite words [DM10a].

Our idea was initially to follow another path to get some logical characterization of a family of weighted automata. Rather than starting from a very powerful logic, like weighted monadic second order logic, and strongly restricting it to recover the power of weighted automata, we chose to start with a weaker logic (weighted first order logic) to which we add a weighted transitive closure operator. This idea has been partly inspired by results of [EH07] claiming the equivalence, in the Boolean setting, between tree-walking automata with pebbles and first order logic with transitive closure.

In this chapter, we will hence consider different logical formalisms starting with the classical (Boolean) monadic second order logic. We show how to use the already known decidability results to deduce the decidability of the emptiness of pebble weighted automata under some hypotheses. We then develop our main weighted logic, composed of first-order constructs to which we add a weighted transitive closure. Sections 6.4 and 6.5 then present in full details the proof of equivalence between weighted logical and automatic formalisms. We finish this chapter with the quick investigation of hybrid navigational logics, which can be seen as an extension of weighted expressions with more powerful logical features. Some logics and proof ideas present in this chapter are published in restricted cases in [1, 4], but the graph setting is a new contribution of this manuscript, as well as the detailed study of hybrid navigational logics.

6.1 Monadic Second Order Logic over Graphs

As a preliminary, we recall the classical definitions of monadic second order logic over graphs.

Let us fix infinite supplies of first-order variables $\mathrm{Var} = \{x, y, z, t, x_1, x_2, \dots\}$, and of second-order variables $\mathrm{VAR} = \{X, Y, X_1, X_2, \dots\}$. The set $\mathrm{MSO}(A, D)$ (or simply MSO if the alphabet A and the set D of directions are clear from the context) of monadic second-order formulae over A and D is given by the grammar:

$$\begin{aligned}\varphi ::= {} & \top \mid (x = y) \mid \mathsf{init}(x) \mid \mathsf{final}(x) \mid P_a(x) \mid R_d(x, y) \mid R_d^\star(x, y) \mid x \in X \mid \\ & \neg\varphi \mid \varphi \vee \varphi \mid \exists x\, \varphi \mid \exists X\, \varphi\end{aligned}$$

where $a \in A$, $d \in D$, $x, y \in \mathrm{Var}$ and $X \in \mathrm{VAR}$. As usual, the set $\mathrm{FO}(A, D)$ (or simply FO) of first-order formulae over A is the fragment of $\mathrm{MSO}(A, D)$ without second-order quantifications $\exists X$.

For $\varphi \in \mathrm{MSO}(A, D)$, we define as usual $\mathrm{Free}(\varphi)$ the set of free variables of φ. If $\mathrm{Free}(\varphi) = \emptyset$, then φ is called a *sentence*. For a pointed graph $G \in \mathcal{G}(A, D)$, we now consider a valuation σ to be a function that maps first-order variables in Var to elements of V and second-order

Table 6.1: Semantics of MSO

$G, \sigma \models \top$	
$G, \sigma \models (x = y)$	if, and only if, $\sigma(x) = \sigma(y)$
$G, \sigma \models \mathsf{init}(x)$	if, and only if, $\sigma(x) = v^{(i)}$
$G, \sigma \models \mathsf{final}(x)$	if, and only if, $\sigma(x) = v^{(f)}$
$G, \sigma \models P_a(x)$	if, and only if, $\lambda(\sigma(x)) = a$
$G, \sigma \models R_d(x, y)$	if, and only if, $(\sigma(x), \sigma(y)) \in E_d$
$G, \sigma \models R_d^\star(x, y)$	if, and only if, there is a d-path from $\sigma(x)$ to $\sigma(y)$
$G, \sigma \models x \in X$	if, and only if, $\sigma(x) \in \sigma(X)$
$G, \sigma \models \neg\varphi$	if, and only if, $G, \sigma \not\models \varphi$
$G, \sigma \models \varphi_1 \vee \varphi_2$	if, and only if, $G, \sigma \models \varphi_1$ or $G, \sigma \models \varphi_2$
$G, \sigma \models \exists x\, \varphi$	if, and only if, there exists $v \in V$ such that $G, \sigma[x \mapsto v] \models \varphi$
$G, \sigma \models \exists X\, \varphi$	if, and only if, there exists $I \subseteq V$ such that $G, \sigma[X \mapsto I] \models \varphi$

variables in VAR to subsets of V. As usual, for $x \in \mathrm{Var}$ and $v \in V$, $\sigma[x \mapsto v]$ denotes the valuation that maps x to v and, otherwise, coincides with σ, and for $X \in \mathrm{VAR}$ and $I \subseteq V$, $\sigma[X \mapsto I]$ is defined similarly.

For every *pointed* graph G and every σ with domain containing the free variables of φ, we define $G, \sigma \models \varphi$ by induction over the formula φ, as shown in Table 6.1. Note in particular that the semantics of φ only depends on the restriction of σ to the free variables of φ.

In the following, we will use several shortcuts in order to keep formulae readable and of reasonable size. It is however important to carefully notice what is the logical fragment needed to express them. For example, we allow to state the existence of a d-path of length k in a graph between two given vertices, with a formula denoted by $R_d^k(x, y)$, with $k \geq 0$: this formula can be obtained by the following recursive process, implying that this formula is expressible in FO

$$R_d^0(x, y) = (x = y)\,, \qquad R_d^{k+1}(x, y) = \exists z\; R_d(x, z) \wedge R_d^k(y, z) \tag{6.1}$$

We will also allow modulo constraints on such paths following a direction d, denoted by $(x, y) \equiv_d m[\ell]$ with $0 \leq m \leq \ell - 1$: this formula is evaluated to true if, and only if, there exists a d-path from the vertex encoded by x to the one encoded by y, whose length is congruent to m modulo ℓ. This formula is MSO-definable by

$$\begin{aligned}((x, y) \equiv_d m[\ell]) = \forall X \Big(\big[(y \in X) \wedge \big(\forall z, z' (z \in X \wedge R_d^\ell(z', z)) \implies z' \in X \big) \big] \\ \implies \exists z \in X\; R_d^m(x, z) \Big).\end{aligned}$$

Notice that this modulo operator may also be defined in first-order logic extended with a *transitive closure operator*. More formally, such an operator, denoted by TC in the following, takes a formula φ with two distinguished free variables, e.g., x and y, and gives a formula $\mathrm{TC}_{x,y}\varphi$ with the same free variables as φ. Let $G \in \mathcal{G}(A, D)$ be a pointed graph and σ be a valuation of the free variables of φ. We may interpret the semantics of φ as a binary relation $R_{G,\sigma}(\varphi)$ over V

$$R_{G,\sigma}(\varphi) = \{(v, v') \mid G, \sigma[x \mapsto v, y \mapsto v'] \models \varphi\}$$

meaning that

$$G, \sigma \models \varphi \text{ if, and only if } (\sigma(x), \sigma(y)) \in R_{G,\sigma}(\varphi)$$

Consider the transitive closure relation $R_{G,\sigma}(\varphi)^+$ of $R_{G,\sigma}(\varphi)$ defined as usual as the union of the sequence $(R_{G,\sigma}(\varphi)^k)_{k\geq 1}$ defined by

$$\begin{cases} R_{G,\sigma}(\varphi)^1 = R_{G,\sigma}(\varphi) \\ R_{G,\sigma}(\varphi)^{k+1} = \{(v, v'') \mid (v, v') \in R_{G,\sigma}(\varphi) \text{ and} \\ \qquad\qquad\qquad (v', v'') \in R_{G,\sigma}(\varphi)^k \text{ for some } v' \in V\} \end{cases}$$

The semantics of $\mathrm{TC}_{x,y}\varphi$ is then defined as

$$G, \sigma \models \mathrm{TC}_{x,y}\varphi \text{ if, and only if } (\sigma(x), \sigma(y)) \in R_{G,\sigma}(\varphi)^+$$

If we want to stress the fact that x and y are free variables of the new built formula, we may write it $[\mathrm{TC}_{x,y}\varphi](x,y)$, or even $[\mathrm{TC}_{x,y}\varphi](x',y')$ if we want to change their name.

We denote by FO + posTC the logic composed of first order logic to which we add *positive* transitive closure operators, i.e., having an even number of negations above them. Indeed, if we push the negations over the atoms, it is exactly the logic defined by

$$\begin{aligned}\varphi ::= {} & \top \mid (x = y) \mid \mathsf{init}(x) \mid \mathsf{final}(x) \mid P_a(x) \mid R_d(x,y) \mid R^\star_d(x,y) \mid \\ & \neg\top \mid \neg(x = y) \mid \neg\mathsf{init}(x) \mid \neg\mathsf{final}(x) \mid \neg P_a(x) \mid \neg R_d(x,y) \mid \neg R^\star_d(x,y) \mid \\ & \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x\, \varphi \mid \forall x\, \varphi \mid \mathrm{TC}_{x,y}\varphi\end{aligned}$$

where $a \in A$, $d \in D$ and $x, y \in \mathrm{Var}$. We may also denote formula $\neg(x = y)$ by $x \neq y$ in the following.

The modulo operator can then be equivalently defined with formula

$$((x,y) \equiv_d m[\ell]) = \exists z\, R^m_d(x,z) \wedge \big([\mathrm{TC}_{x,y} R^\ell_d(x,y)](z,y) \vee z = y\big).$$

In case D is an ordered set of directions, with a distinguished forward direction $\to$, we will use formula $x \leq y$ as a shortcut for $R^\star_\to(x,y)$: this is in reference of $\to$-ordered graphs on which we may then interpret the formula, which come with a linear order $\leq$ generated by $\to$.

6.2 Emptiness of Pebble Weighted Automata is Decidable

Monadic second-order logic has been extensively studied over graphs, in particular the decidability status of the satisfiability problem over a certain class $\mathfrak{G}$ of graphs. For example, it is well-known that for graphs of bounded degree, this problem is decidable if, and only if, all the graphs of $\mathfrak{G}$ have a bounded *clique-width* (see [CE11] for the definition and the proof of this result).

Using this result, we now consider the emptiness problem of our classes of pebble weighted automata, which is, given a layered pebble weighted automaton $\mathcal{A}$, does there exist a pointed graph $G \in \mathfrak{G}$ and a valuation σ of the free variables of $\mathcal{A}$ such that $[\![\mathcal{A}]\!](G,\sigma) \neq 0$. This is the simplest question that may be asked, and this is indeed decidable if we consider layered automata over positive continuous semirings, and bounded clique-width graphs.

Theorem 6.1. *Let $\mathfrak{G}$ be a class of (A,D)-pointed graphs with bounded clique-width, and $\mathbb{S}$ be a positive continuous semiring. The following problem is decidable:*

> ***Data:*** *a layered pebble weighted automaton $\mathcal{A}$ over A, D, $\mathbb{S}$*
> ***Question:*** *does there exist $G \in \mathfrak{G}$ and σ a valuation with domain containing* $\mathrm{Free}(\mathcal{A})$ *such that $[\![\mathcal{A}]\!](G,\sigma) \neq 0$?*

Proof. We reduce this problem to the same problem in the Boolean semiring (which is a typical positive continuous semiring). We will then show the decidability status for the Boolean semiring.

Consider a layered pebble weighted automaton $\mathcal{A} = (Q, A, D, I, \Delta, F)$ over a positive continuous semiring $\mathbb{S}$. Let $G \in \mathcal{G}(A,D)$ be a pointed graph and σ a valuation with domain containing the free variables of $\mathcal{A}$. As the semiring is positive, it is zerosumfree (indeed, being continuous would have been sufficient here, by using the result of Proposition 2.5). Then, as the semantics of $\mathcal{A}$ over (G,σ) is defined as the sum of the weights of the accepting runs of $\mathcal{A}$ over (G,σ), we have that $[\![\mathcal{A}]\!](G,\sigma) \neq 0$ if, and only if, there exists an accepting run ρ of

$\mathcal{A}$ over (G,σ) with weight different from 0. As the semiring is positive, it contains no zero divisor, and hence, every transition in ρ has a weight different from 0. Hence, let us consider the layered pebble weighted automaton $\mathcal{B} = (Q, A, D, I', \Delta', F')$ over the Boolean semiring $(\{0,1\}, \vee, \wedge, 0, 1)$ obtained from $\mathcal{A}$ by replacing every weight different from 0 appearing in I, Δ or F by 1, and letting the weight 0 unchanged. The previous reasoning permits to prove that for every pointed graph G

$$[\![\mathcal{A}]\!](G,\sigma) \neq 0 \quad \text{if, and only if,} \quad [\![\mathcal{B}]\!](G,\sigma) \neq 0\,.$$

Now, we prove that the problem over the Boolean semiring is decidable. Let $\mathcal{A} = (Q, A, D, I, \Delta, F)$ be a layered pebble weighted automaton over the Boolean semiring. The decidability is shown by encoding runs of $\mathcal{B}$ into a formula of MSO. More precisely, we exhibit a formula φ with free variables $\mathrm{Free}(\mathcal{A})$ such that for every pointed graph $G \in \mathcal{G}(A,D)$ and every valuation σ with domain containing $\mathrm{Free}(\mathcal{A})$:

$$[\![\mathcal{B}]\!](G,\sigma) = 1 \quad \text{if, and only if,} \quad G, \sigma \models \varphi$$

Hence, we reduce the initial problem to the satisfiability of a formula of MSO over a class of graphs which are of bounded clique-width. This problem is known to be decidable, as recalled previously.

We do not present the construction of the formula φ in full details, but rather use the translation from *walking automata with nested pebbles* into the logic $\mathrm{FO} + \mathrm{posTC}$ proved by [EH07]: notice that their model of automata may have several heads and uses strong pebbles, hence our model can be translated in theirs. To conclude, it suffices to notice that all formulae of $\mathrm{FO} + \mathrm{posTC}$ can be translated into equivalent formulae of MSO: proof of this fact is given in [CE11]. □

6.3 Weighted logics

Our aim is to define a denotational model to describe the behavior of weighted automata. This has already been done by Droste and Gastin [DG09]: they have introduced weighted logics with syntax close to monadic second-order logic, extending the semantics by using addition and product of a semiring to evaluate disjunctions/existential quantifications, and conjunctions/universal quantifications, respectively. At the price of strong restrictions on the shape of the formulae, they succeeded to prove an expressiveness result (in our formalism, this result is stated in Theorem 6.8).

The syntax of [DG09] is purely quantitative, though Boolean connectives can be expressed indirectly. As it may be somewhat confusing to interpret purely logical formulae in a weighted manner, we slightly modify the original syntax, by clearly separating the Boolean and the quantitative parts: our weighted logic consists of a Boolean kernel, such as MSO or FO, augmented with quantitative operators (addition, multiplication, sum and product quantifications, and possibly weighted transitive closure). Thus, our language will allow us to test explicitly for Boolean properties, and to perform computations. We shall see that, under some hypotheses, the formalism of [DG09] and ours are equivalent. However, in addition to being more intuitive, the new syntax allows flexibility to study expressiveness. For instance, one can investigate how the underlying Boolean logic influences the computational power (see Lemma 6.5).

6.3.1 General definitions

Definition 6.2 (Weighted logics)**.** Given a class $\mathcal{L}$ of Boolean formulae, we denote by $\mathrm{wMSO}(\mathcal{L})$ the class of *weighted monadic second order logic* defined by

$$\Phi ::= s \mid \varphi \mid \Phi \oplus \Phi \mid \Phi \otimes \Phi \mid \textstyle\bigoplus_x \Phi \mid \bigotimes_x \Phi \mid \bigoplus_X \Phi \mid \bigotimes_X \Phi$$

$$
\begin{aligned}
[\![s]\!](G,\sigma) &= s\\
[\![\varphi]\!](G,\sigma) &= \begin{cases} 1 & \text{if } G,\sigma \models \varphi \\ 0 & \text{otherwise} \end{cases}\\
[\![\Phi_1 \oplus \Phi_2]\!](G,\sigma) &= [\![\Phi_1]\!](G,\sigma) \oplus [\![\Phi_2]\!](G,\sigma)\\
[\![\Phi_1 \otimes \Phi_2]\!](G,\sigma) &= [\![\Phi_1]\!](G,\sigma) \otimes [\![\Phi_2]\!]G,\sigma)\\
[\![\textstyle\bigoplus_x \Phi]\!](G,\sigma) &= \bigoplus_{v\in V} [\![\Phi]\!](G,\sigma[x\mapsto v])\\
[\![\textstyle\bigotimes_x \Phi]\!](G,\sigma) &= \bigotimes_{v\in V} [\![\Phi]\!](G,\sigma[x\mapsto v])\\
[\![\textstyle\bigoplus_X \Phi]\!](G,\sigma) &= \bigoplus_{I\subseteq V} [\![\Phi]\!](G,\sigma[X\mapsto I])\\
[\![\textstyle\bigotimes_X \Phi]\!](G,\sigma) &= \bigotimes_{I\subseteq V} [\![\Phi]\!](G,\sigma[X\mapsto I])
\end{aligned}
$$

Table 6.2: Semantics of formulae in wMSO($\mathcal{L}$)

where $s \in \mathbb{S}$, $\varphi \in \mathcal{L}$, $x \in \mathrm{Var}$ and $X \in \mathrm{VAR}$. Disabling the sum and product indexed by set variables, we define the class wFO($\mathcal{L}$) of *weighted first order logic* by

$$\Phi ::= s \mid \varphi \mid \Phi \oplus \Phi \mid \Phi \otimes \Phi \mid \textstyle\bigoplus_x \Phi \mid \bigotimes_x \Phi$$

where $s \in \mathbb{S}$, $\varphi \in \mathcal{L}$ and $x \in \mathrm{Var}$. ■

We denote again by Free(Φ) the set of free variables of a formula $\Phi \in$ wMSO($\mathcal{L}$). The semantics $[\![\Phi]\!]$ of Φ maps a pair (G,σ), composed of a pointed graph $G \in \mathcal{G}(A,D)$ and a valuation σ with domain containing the free variables of Φ, to a value in $\mathbb{S}$ as showed inductively in Table 6.2. Hereby, if $\mathbb{S}$ is not commutative, we assume that the products follow a specified order of V and the associated lexicographic order on the power set $\{0,1\}^V$. As for the Boolean case, notice that the semantics $[\![\Phi]\!]$ does not depend on the assignment of variables not in Free(Φ). Alternatively, we may see $[\![\cdot]\!]$ as a series in $\mathbb{S}\langle\!\langle \mathcal{G}(A,D)\rangle\!\rangle$, defining only the semantics of sentences. Henceforth, a series $f \in \mathbb{S}\langle\!\langle \mathcal{G}(A,D)\rangle\!\rangle$ is said to be wMSO($\mathcal{L}$)-definable if there exists a sentence $\Phi \in$ wMSO($\mathcal{L}$) such that $[\![\Phi]\!] = f$.

Example 6.3. In the Boolean semiring $\mathbb{B}$, recognizable and wMSO(MSO)-definable series of words (encoded as graphs in $\mathcal{W}\mathrm{ord}(A)$) coincide. In contrast, for the natural semiring $\mathbb{N}$, the definition yields $[\![\bigotimes_x \bigotimes_y 2]\!](G) = 2^{|V|^2}$ for every $G \in \mathcal{W}\mathrm{ord}(A)$ modeling a word, which is not recognized by a weighted automaton over words as proved in Theorem 4.18. ■

Having the capability to mix Boolean and weighted operators permits to design more easily complex formulae.

Example 6.4. Assume we have an XML document representing a database storing information about car models and car parts: it is depicted in Figure 6.1 in the shape of a tree, but we will consider it as a nested word in the following. Each car model is described, among other things, by its list of car parts (Figure 6.1 (a)). Each model of car part has an attached set of currently reported errors indicated by vertices labeled Err in Figure 6.1 (b). One can express that there is a car model using some car part with an error by the first order formula $\exists x,y,z,t \quad \varphi(x,y,z,t)$, where

$$\varphi = [\mathsf{CModel}(x) \wedge \mathsf{Part}(y) \wedge x \prec y] \wedge [\mathsf{PModel}(z) \wedge \mathsf{Err}(t) \wedge z \prec t] \wedge \mathsf{Match}(y,z)$$

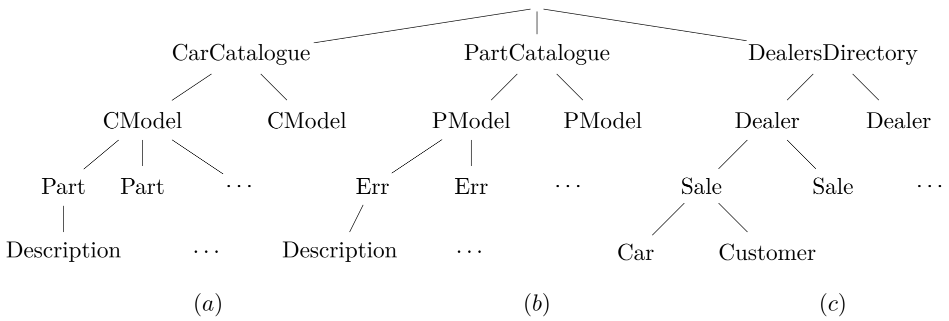


Figure 6.1: Part of an XML document for the car database

This is a purely logical statement where $x \prec y$ means that node x is an ancestor of node y in the XML document which can be written as

$$\exists z \quad R_{\curvearrowright}(x,z) \wedge R^{\star}_{\rightarrow}(x,y) \wedge R^{\star}_{\rightarrow}(y,z)$$

and $\mathsf{Match}(y,z)$ is a shortcut meaning that the car part at y matches the part model at z. In this manuscript, we abstracted data away but this is typically the place where we could use data from an infinite alphabet to model more knowledge of the database.

One may want a more precise information, for example the total number of errors for all car models. This can be achieved by replacing existential quantifications with sums: $\bigoplus_{x,y,z,t} \varphi(x,y,z,t)$, and computing in the natural semiring $(\mathbb{N},+,\times,0,1)$ instead of the Boolean semiring.

Assume now that the database also includes car dealers, see Figure 6.1 (c). Each dealer records a list of performed sales. Here is the shape of a formula computing the maximal number of errors to be fixed per dealer:

$$\bigoplus_d \mathsf{Dealer}(d) \otimes \bigotimes_u \Big[(d \prec u \wedge \mathsf{Car}(u)) \otimes \bigotimes_{x,y,z,t}(\mathsf{Match}'(u,x) \wedge \varphi(x,y,z,t))\Big]$$

Here, one needs to interpret $\oplus$ and $\otimes$ as max and sum operations which are available in the semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$. ■

We denote by AP the class of atomic propositions, i.e., that contains only formulae $x = y$, $\mathsf{init}(x)$, $\mathsf{final}(x)$, $P_a(x)$, $R_d(x,y)$, $R^{\star}_d(x,y)$, $x \in X$ and their negations. As explained at the beginning of the section, the logic wMSO($\mathbb{S}$) (respectively, wFO($\mathbb{S}$)) introduced in [DG09] over words is exactly the class of formulae wMSO(AP) (respectively, wFO(AP)). Transposed into our syntax, the transformation described in their Definition 4.3 and proved in Lemma 4.4, permits to state:

Lemma 6.5. *Let D be an ordered set of directions with a distinguished forward direction $\rightarrow$. Then, for every Boolean formula $\varphi \in \mathrm{MSO}(A,D)$ (respectively, $\varphi \in \mathrm{FO}(A,D)$), there exists a weighted formula $\Phi \in \mathrm{wMSO(AP)}$ (respectively, $\Phi \in \mathrm{wFO(AP)}$), with the same set of free variables, such that for every $\rightarrow$-ordered pointed graph G and every valuation σ, $[\![\varphi]\!](G,\sigma) = [\![\Phi]\!](G,\sigma)$.*

Proof. We will express Boolean connectives with their quantitative counterparts.

We simulate disjunction with $\oplus$, existential first-order quantification with $\bigoplus_x$ and existential second-order quantification with $\bigoplus_X$. More precisely, for each MSO-formula φ, we effectively construct two weighted formulae $\Phi^+_\varphi, \Phi^-_\varphi \in \mathrm{wMSO(AP)}$ (which, in addition, belong to wFO(AP) if φ is an FO-formula) such that for every $\rightarrow$-ordered pointed graph G and every σ with domain containing the free variables of φ, we have $[\![\Phi^+_\varphi]\!](G,\sigma), [\![\Phi^-_\varphi]\!](G,\sigma) \in \{0,1\}$, and

$$[\![\Phi^+_\varphi]\!](G,\sigma) = 1 \iff G,\sigma \models \varphi \qquad \text{and} \qquad [\![\Phi^-_\varphi]\!](G,\sigma) = 0 \iff G,\sigma \models \varphi\,.$$

To simplify notations, $x \leq y$ will denote the atomic proposition $R^{\star}_{\rightarrow}(x,y)$ with $\rightarrow$ generating the order $\leq$ over the vertices, whereas $X \leq Y$ will denote the lexicographic order over the sets of positions, that can be defined in wMSO(AP) by

$$\begin{aligned}
&X < Y = \bigoplus_y \left[y \in Y \otimes \neg(y \in X) \otimes \bigotimes_z (y \leq z \oplus (z < y \otimes z \in (X\Delta Y)^c))\right]\\
&X \leq Y = X < Y \oplus \bigotimes_z z \in (X\Delta Y)^c,
\end{aligned}$$

Here, $z \in (X\Delta Y)^c$ denotes the formula $(z \in X \otimes z \in Y) \oplus (\neg(z \in X) \otimes \neg(z \in Y))$, so that $\bigotimes_z (y \leq z \oplus (z < y \otimes z \in (X\Delta Y)^c))$ tests whether X and Y agree before position y.

We define formulae Φ^+_φ and Φ^-_φ by structural induction on φ, removing ambiguity by picking the leftmost witness of the formulae (in particular in disjunction and existential quantification):

- if $\varphi \in \mathrm{AP}$ then $\Phi^+_\varphi = \varphi$ and $\Phi^-_\varphi = \neg\varphi$ (with $\neg\neg\psi = \psi$ by convention);
- $\Phi^+_{\varphi\vee\psi} = \Phi^+_\varphi \oplus (\Phi^-_\varphi \otimes \Phi^+_\psi)$ and $\Phi^-_{\varphi\vee\psi} = \Phi^-_\varphi \otimes \Phi^-_\psi$;
- $\Phi^+_{\neg\varphi} = \Phi^-_\varphi$ and $\Phi^-_{\neg\varphi} = \Phi^+_\varphi$;
- $\Phi^+_{\exists x\varphi} = \bigoplus_x \left[\Phi^+_\varphi \otimes \bigotimes_y [x \leq y \oplus (y < x \otimes \Phi^-_\varphi)]\right]$ and $\Phi^-_{\exists x\varphi} = \bigotimes_x \Phi^-_\varphi$;
- $\Phi^+_{\exists X\varphi} = \bigoplus_X \left[\Phi^+_\varphi \otimes \bigotimes_Y [X \leq Y \oplus (Y < X \otimes \Phi^-_\varphi)]\right]$ and $\Phi^-_{\exists X\varphi} = \bigotimes_X \Phi^-_\varphi$.

□

Hence, in ordered graphs, we can always replace a Boolean FO or MSO formula by its wFO(AP) or wMSO(AP) equivalent and we obtain:

Corollary 6.6. *Let $\mathbb{S}$ be a semiring and D be an ordered set of directions. Let $f \in \mathbb{S}\langle\langle \mathcal{G}(A,D)\rangle\rangle$. Then,*

- *f is* wMSO(MSO)*-definable if, and only if, f is* wMSO(AP)*-definable.*
- *f is* wFO(FO)*-definable if, and only if, f is* wFO(AP)*-definable.*

For a Boolean formula $\varphi \in \mathcal{L}$ and two formulae $\Phi, \Psi \in \mathrm{wMSO}(\mathcal{L})$, we define the macro $\varphi?\,\Phi : \Psi$ as

$$\varphi?\,\Phi : \Psi = (\varphi \otimes \Phi) \oplus (\neg\varphi \otimes \Psi)$$

as a natural generalization of the Boolean if-then-else operation: its semantics is $[\![\Phi]\!]$ if φ holds, and $[\![\Psi]\!]$ otherwise. We simply write $\varphi?\,\Phi$ for $\varphi?\,\Phi : 1$.

6.3.2 Previous expressiveness result

For $\mathcal{L}$ a Boolean logic closed under $\vee$, $\wedge$ and $\neg$, an *$\mathcal{L}$-step formula* is a formula obtained from the grammar

$$\Phi ::= s \mid \varphi \mid \Phi \oplus \Phi \mid \Phi \otimes \Phi, \qquad \text{with } s \in \mathbb{S} \text{ and } \varphi \in \mathcal{L}. \tag{6.2}$$

The following lemma shows in particular that an $\mathcal{L}$-step formula can take only a finite number of values.

Lemma 6.7. *For every $\mathcal{L}$-step formula Φ, one can construct an equivalent formula $\Psi = \bigoplus_{I\in K}(\psi_I \otimes s_I)$ with K finite, $\psi_I \in \mathcal{L}$ and $s_I \in \mathbb{S}$. More precisely,* $\mathrm{Free}(\Phi) = \mathrm{Free}(\Psi)$ *and $[\![\Phi]\!](G,\sigma) = [\![\Psi]\!](G,\sigma)$ for all pairs (G,σ) of graph and valuation.*

Proof. Call $\varphi_1, \dots, \varphi_p$ the $\mathcal{L}$-formulae occurring in the expression of Φ given by the grammar (6.2). For $I \subseteq \{1,\dots,p\}$, let Φ_I be the formula obtained by replacing in Φ each φ_i by 1 if $i \in I$ and by 0 otherwise, so that Φ_I has no free variable and $[\![\Phi_I]\!]$ is a constant $s_I \in \mathbb{S}$. Let $\psi_I = \bigwedge_{i\in I}\varphi_i \wedge \bigwedge_{i\notin I}\neg\varphi_i$, which is an $\mathcal{L}$-formula, since $\mathcal{L}$ is closed under $\wedge$ and $\neg$. Let $\Psi = \bigoplus_I(\psi_I \otimes s_I)$. Clearly, $\mathrm{Free}(\Phi) = \mathrm{Free}(\Psi)$.

Fix a graph G and a valuation σ. Let $J = \{i \mid G,\sigma \models \varphi_i\}$, so that $G,\sigma \models \psi_J$ and $G,\sigma \not\models \psi_I$ for $I \neq J$. Therefore, $[\![\Psi]\!](G,\sigma) = s_J = [\![\Phi_J]\!](G,\sigma) = [\![\Phi]\!](G,\sigma)$, where the last equality comes from the definition of J. □

From now on, we freely use Lemma 6.7, using the special form it provides for $\mathcal{L}$-step formulae. All MSO-step formulae are clearly recognizable. By [DG09], $\bigotimes_x \Phi$ is recognizable over words for any MSO-step formula Φ. A restricted fragment sREMSO of wMSO(AP) (called the syntactically restricted formulae) was defined in [DG09]. Essentially, sREMSO consists of wMSO(MSO) formulae Φ in existential normal form (i.e., no second-order universal quantifier and all second-order existential quantifiers at the beginning of the formula) such that if Φ contains $\Psi \otimes \Psi'$ as a subformula then the values in $\mathbb{S}$ appearing in Ψ and Ψ' commute element-wise, and such that the use of first-order product $\bigotimes_x$ is restricted to MSO-step formulae. Note that if the semiring is commutative, the requirement over the product $\Psi \otimes \Psi'$ is trivially verified. A consequence of their work is the following statement.

Theorem 6.8 ([DG09]). *A formal power series over finite words is recognizable by a (classical) weighted automaton if, and only if, it is definable in* sREMSO*.*

6.3.3 Weighted transitive closure

We now define other fragments of wMSO(MSO), which also carry enough expressiveness to generate all recognizable series of words, and even series recognized by pebble weighted automata over some classes of graphs. Contrary to sREMSO that allows second-order quantitative operators, and finally restricts them, we rather extend the fragment wFO(AP), containing only first-order quantitative operators, with a weighted equivalent of the Boolean transitive closure.

Weighted transitive closure

For a formula $\Phi(x, y)$ with at least two free variables x and y, we introduce a *weighted transitive closure operator* $\mathrm{wTC}_{x,y}\Phi$ having as free variables the same variable as Φ. As for its Boolean counterpart, we may denote it $[\mathrm{wTC}_{x,y}\Phi](x, y)$ if we want to stress the existence of two special free variables, or even $[\mathrm{wTC}_{x,y}\Phi](x', y')$ if we need to change the name of the variables. Its semantics is defined by

$$[\![[\mathrm{wTC}_{x,y}\Phi](x', y')]\!](G, \sigma) = \bigoplus_{\sigma(x')=v_0, v_1, \ldots, v_m=\sigma(y')} \bigotimes_{0 \le k \le m-1} [\![\Phi]\!](G, \sigma[x \mapsto v_k, y \mapsto v_{k+1}]) \tag{6.3}$$

where the sum ranges over all $m > 0$ and all sequences $(v_k)_{0 \le k \le m}$ of vertices of the graph G with $v_0 = \sigma(x')$ and $v_m = \sigma(y')$. Notice that this sum may be infinite, and hence requires the semiring to be continuous to be well-defined.

To ease notation, we sometimes write $[\![\Phi(x, y)]\!](G, v, v')$ instead of $[\![\Phi(x, y)]\!](G, [x \mapsto v, y \mapsto v'])$.

In order to compare automata and logics, we need to define a bounded restriction on this very powerful operator.

Bounded restriction

We introduce a *bounded weighted transitive closure operator* $\mathrm{wTC}^N_{x,y}$ for each integer $N > 0$ (or $\mathrm{wTC}^{\mathrm{b}}_{x,y}$ if N is unspecified). Its semantics consists in restricting the sequences of (6.3) so that $\mathsf{d}(v_k, v_{k+1}) \le N$ for every $k \in \{0, \ldots, m-1\}$. Notice that given two vertices v and v', formula $\mathsf{d}(x, y) \le N$, defined by

$$(\mathsf{d}(x, y) \le N) = \exists x_1 \exists x_2 \cdots \exists x_{N-1}$$
$$\bigvee_{d_1, \ldots, d_N} R_{d_1}(x, x_1) \wedge \bigwedge_{1 \le n \le N-2} R_{d_{n+1}}(x_n, x_{n+1}) \wedge R_{d_N}(x_{N-1}, y)$$

when interpreted over the graph with x mapped to v and y mapped to v', verifies that $\mathsf{d}(v,v') \leq N$. Equivalently,

$$\mathrm{wTC}^N_{x,y}\Phi = \mathrm{wTC}_{x,y}(\mathsf{d}(x,y) \leq N \otimes \Phi)\,.$$

We will see later that this restriction is *a priori* unavoidable in the weighted case.

In the following, if $\mathcal{L}$ is a Boolean logic, we denote by wFO+wTC$^\mathrm{b}(\mathcal{L})$ the weighted logic defined by

$$\Phi ::= s \mid \varphi \mid \Phi \oplus \Phi \mid \Phi \otimes \Phi \mid \textstyle\bigoplus_x \Phi \mid \bigotimes_x \Phi \mid \mathrm{wTC}^N_{x,y}\Phi$$

with $s \in \mathbb{S}$, $\varphi \in \mathcal{L}$, $x, y \in \mathrm{Var}$ and $N \in \mathbb{N} \setminus \{0\}$. A formula Φ of wFO+wTC$^\mathrm{b}(\mathcal{L})$ is said to be *unambiguous* if its semantics is either 0 or 1 for every possible graph and valuation.

It has to be noticed that modulo formulae can be simulated in wFO+wTC$^\mathrm{b}$(AP). Indeed, for d a direction, we consider the formula

$$\textstyle\bigoplus_z R^m_d(x,z) \otimes \big([\mathrm{wTC}^\ell_{x_1,x_2}(x_1 \neq y \otimes R^\ell_d(x_1,x_2))](z,y) \oplus z = y\big)$$

where $R^m_d(x,z)$ is defined almost as in (6.1) but using $\bigoplus_z$ instead of $\exists z$ and $\otimes$ instead of $\wedge$. The only change is the verification $x_1 \neq y$ in the transitive closure, making sure that we stop the first time y is reached. Notice that if y is not linked to x by a d-path, then the transitive closure will evaluate to 0 which is correct. Then, for every graph $G \in \mathcal{G}(A,D)$ and valuation σ of variables x and y, its semantics over (G,σ) is either 0 or 1. Moreover, it admits the same semantics as the modulo constraint $(x,y) \equiv_d m[\ell]$. Henceforth, we use formula $(x,y) \equiv_d m[\ell]$ as a macro in the following.

6.4 From Logics to Automata

This section shows in which context series definable by formulae of wFO+wTC$^\mathrm{b}$(AP) are recognizable by layered PWA. Indeed, we do not know how to show such a result in general, but instead we need an hypothesis over the graphs on which the automata will run. This restriction consists in considering *searchable classes of graphs*, as described in [EH07]:

> A family of graphs is searchable if there exists a (fixed) [...] deterministic graph-walking automaton with [...] pebbles that, for each graph in the family, and each node of the graph, when started in that node in the initial state the automaton halts after completing a walk along the graph during which each node is visited at least once. [...] Thus the automaton serves as a guide for the family of graphs, and makes it possible to establish and traverse a total order of the nodes of the graph.

We first define formally the notion of searchable class of graphs in our context[1], before giving the translation from logics into automata.

6.4.1 Searchable Classes of Graphs

Definition 6.9. A class of pointed graphs $\mathfrak{G}$ is said to be *searchable* if there exists a weighted automaton $\mathcal{A}_\mathfrak{G} \in \mathrm{WA}$ with weights 0 and 1 (and hence independent from the semiring), without free variables, and with two distinguished states q_i and q_o, verifying that for every pointed graph $G = (V, (E_d)_{d\in D}, \lambda) \in \mathfrak{G}$, there exists a total linear order $\leq$ over V, with minimal element $v^{(i)}$ and either $v^{(f)} = v^{(i)}$ or $v^{(f)}$ is the maximal element of $\leq$, such that

[1] Whereas, in [EH07] they use *strong pebbles*, i.e., pebbles that may be lifted without moving back the head to the position of the pebble, we indeed chose to consider *weak pebbles*.

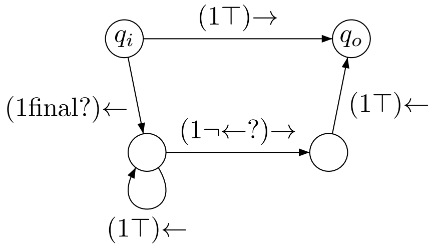


Figure 6.2: Automaton exploring the class of →-ordered pointed graphs

1. for every vertex $v \in V \setminus \{v^{(f)}\}$, there exists a unique non-zero run in $\mathcal{A}_{\mathfrak{G}}$ from configuration (G, q_i, v) reaching state q_o: this run ends in configuration (G, q_o, v') where v' is the direct successor of v in $\leq$;
2. if $v^{(f)} \neq v^{(i)}$, there exists a unique non-zero run in $\mathcal{A}_{\mathfrak{G}}$ from configuration $(G, q_i, v^{(f)})$ reaching state q_o: this run ends in configuration $(G, q_o, v^{(i)})$.

$\mathcal{A}_{\mathfrak{G}}$ is called a *guide* for the class $\mathfrak{G}$. ■

Every class of pointed graphs $\mathfrak{G}$ which contains only →-ordered pointed graphs is easily searchable, by the automaton $\mathcal{A}_{\mathfrak{G}}$ depicted in Figure 6.2. Hence, the linear order of the previous definition matches the linear order $\leq$ defined in ordered graphs. However, not only ordered graphs are searchable. For example, pictures are searchable: one possible linear order is to read each line of the picture from left to right. Hence the automaton simply does a right move if it is possible, otherwise, it goes to the next line and comes back to its beginning. Notice that trees are also searchable by using a depth-first-search that may be done using our navigational automata as stated for example in [Boj08], and used in Example 4.8.

Mazurkiewicz traces are also searchable. Indeed, we enumerate processes in increasing order, and once in the upper vertex of a given process, we may enumerate all vertices related to this process, that have not been visited by smaller processes. Moreover, jumping from a process to the next one can be done unambiguously in the following way: starting from the upper vertex of a process, we explore with a depth-first-search the processes reachable from this vertex, keeping in memory the (finite) set of processes visited so far. We stop when the next process is found, and we simply go up to reach its top vertex. The graphs being connected, this algorithm terminates. On the Mazurkiewicz trace depicted in Figure 2.5, jumping from process 3 to process 4 is achieved in the following way. Starting in the top vertex of process 3, we go down reaching a vertex in common with process 2. We reach the top vertex of process 2, and henceforth jump to process 1. Visiting process 1 stops very quickly, since its other vertex is common with process 2, already in memory. We go back to the top vertex of process 2, and go down. Reaching the vertex in common with process 4 permits to conclude the algorithm.

Not all classes of graphs are known to be searchable however. For example, the class of planar graphs is known to be not searchable [Rol80]. Mazes (which are subgraphs of 2-dimensional finite grids) are known to be not searchable [Bud78], even if automata are allowed to use one pebble [Hof81]. However, it is an open problem to know if those mazes are searchable with 2 or more layers of pebbles. We have chosen to restrict guides to be automata without pebbles for sake of simplicity, and because our practical examples of structures are all searchable without pebbles. However, we claim that our work may easily be extended with guides using pebbles, in a layered way.

6.4.2 Translation of Formulae into Automata

Theorem 6.10. *Let $\mathbb{S}$ be a semiring and $\mathfrak{G}$ a searchable class of pointed graphs. For every formula $\Phi \in \mathrm{wFO{+}wTC^b(FO)}$, there exists a layered pebble weighted automaton $\mathcal{A}_\Phi$ with $\mathrm{Free}(\mathcal{A}_\Phi) = \mathrm{Free}(\Phi)$, such that for every graph $G \in \mathfrak{G}$ and every valuation σ with domain containing the free variables of Φ, $[\![\mathcal{A}_\Phi]\!](G,\sigma) = [\![\Phi]\!](G,\sigma)$.*

Proof. In all of this proof, we consider the guide $\mathcal{A}_\mathfrak{G}$ of the searchable class $\mathfrak{G}$.

We give a proof by induction over the formulae in $\mathrm{wFO{+}wTC^b(FO)}$. We start by proving the property for the unweighted part of $\mathrm{wFO{+}wTC^b(FO)}$, i.e., for Booolean formulae of FO. The difficulty lies in the fact that we must simulate a Boolean formula – which henceforth has a weighted semantics $[\![-]\!]$ either 0 or 1 – with weighted automata that do not possess a Boolean fragment. Indeed, we must produce automata that simulate first-order formulae in an *unambiguous* way. Notice that, at least over the class of words, it could be rather easy to produce a *deterministic* finite state automaton simulating every first-order formula. However, the size of such an automaton would be non elementary in the size of the formula (because of the alternance of quantifiers), and this is *a priori* no longer possible with more general classes of graphs (even searchable). Indeed, we propose a totally different construction, that takes advantage of the pebbles, in order to design pebble weighted automata of linear size with respect to the size of the formula. More precisely, for every $\varphi \in \mathrm{FO}$, we construct a layered pebble weighted automaton $\mathcal{B}^\varphi$ having one initial state ι and two (final) states $\texttt{ok}$ and $\texttt{ko}$ such that for all pointed graph $G \in \mathfrak{G}$ and valuation $\sigma\colon \mathrm{Var} \rightharpoonup V$ with domain containing the free variables of φ, the semantics satisfies:

$$[\![\mathcal{B}^\varphi_{\iota,\texttt{ok}}]\!](G,\sigma,v^{(i)},v^{(f)}) = \begin{cases} 1 & \text{if } G,\sigma \models \varphi \\ 0 & \text{otherwise,} \end{cases}$$

$$[\![\mathcal{B}^\varphi_{\iota,\texttt{ko}}]\!](G,\sigma,v^{(i)},v^{(f)}) = \begin{cases} 0 & \text{if } G,\sigma \models \varphi \\ 1 & \text{otherwise.} \end{cases}$$

We obtain automaton $\mathcal{A}_\varphi$ by considering $\mathcal{B}^\varphi$ with ι (respectively, $\texttt{ok}$) having initial (respectively, final) weight 1. To get an automaton for the negation of a formula, we simply exchange states $\texttt{ok}$ and $\texttt{ko}$. These automata will use free variables to encode free variables of formula φ.

For formula $\mathsf{init}(x)$, the automaton starts by verifying that the initial vertex hosts variable x, and then follows the guide to reach the final vertex of the graph. Formula $\mathsf{final}(x)$ is dealt with similarly. For $P_a(x)$, the automaton simply scans the input graph using the guide, searching for a vertex that verifies the type $a? \wedge x?$. Similarly, for formula $x = y$, at vertex x, we verify that the vertex also holds variable y. For the atomic formula $R_d(x,y)$, the automaton scans the input graph, searching for a vertex holding variable x, and at that point follows the edge labeled with direction d to reach vertex holding variable y. It then follows the guide until the final vertex of the graph.

The construction for formula $R_d^\star(x,y)$ has to be made carefully. Indeed, the automaton must be able to compute both this formula and its negation. The latter states that there is no d-path from x to y: this is the case when either there exists a vertex z without d-successor but with a d-path from x to z not containing y, or there exists a d-cycle containing x but not y. These two conditions, as well as the positive check that x and y are related by a d-path, can be checked with an automaton. This automaton first follows the guide to reach the vertex holding variable x. It then follows the unique d-path until either it reaches y, or it reaches x, or it deadlocks. We finally follow the guide to reach the final vertex and accept or reject depending on the case.

The construction for disjunction $\xi = \varphi \vee \psi$ is described in Figure 6.3: in this picture, as well as the following ones, when the guide $\mathcal{A}_\mathfrak{G}$ is called, we suppose it is from its initial state q_i to its final state q_o. We start computing φ and stop if it is verified, otherwise, we

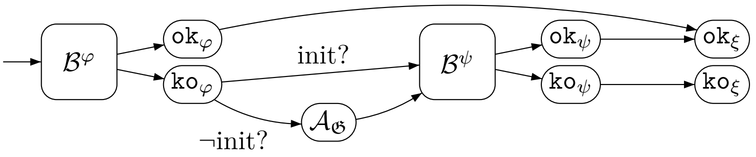


Figure 6.3: Weighted automaton for disjunction $\xi = \varphi \vee \psi$

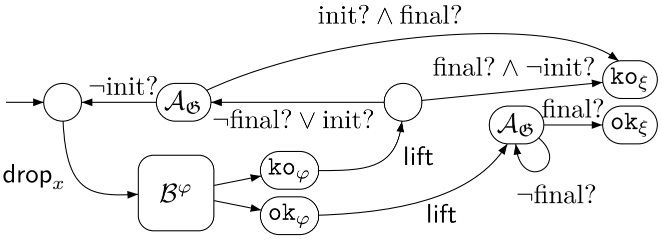


Figure 6.4: Weighted automaton for existential quantification $\xi = \exists x\ \varphi$

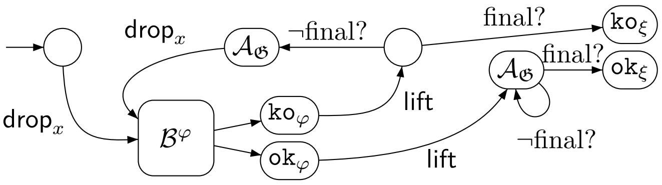


Figure 6.5: Weighted automaton for existential quantification $\xi = \exists x\ \varphi$ in case $v^{(i)} \neq v^{(f)}$

reset to the initial vertex of the graph using the guide, and check formula ψ. Notice that we use in this figure, and in the following, some transitions labeled with only tests, i.e., non moving transitions. It is not difficult to translate these transitions into our model, e.g., by enforcing the automaton to make a move d followed by its opposite d^{-1} in a deterministic way, or to remove this non-moving transitions by merging them with the next moving one.

Finally, the construction for existential quantification $\xi = \exists x\ \varphi$ is described in Figure 6.4. During a run of this automaton, a pebble of name x is successively dropped over each vertex of the graph using the guide, in order to simulate automaton $\mathcal{B}^\varphi$ over every vertex: the apparent complexity of the tests in layer 0 takes care of both cases $v^{(i)} = v^{(f)}$ (then only the transition labeled by init? ∧ final? permits to exit in state $\mathtt{ko}_\xi$) and $v^{(i)} \neq v^{(f)}$ (then only the transition labeled by final? ∧ ¬init? permits to exit in state $\mathtt{ko}_\xi$). The simplest automaton, only taking care of the case $v^{(i)} \neq v^{(f)}$ is described in Figure 6.5. If such a run ends in state $\mathtt{ok}_\xi$, it means that at some vertex v, formula φ has been positively verified (the part of the run in lower layer ended in state $\mathtt{ok}_\varphi$). More precisely, as $\mathcal{B}^\varphi$ is unambiguous, in that case, all the runs of non-zero weight will end in state $\mathtt{ok}_\xi$ after dropping the pebble on vertex v, henceforth computing the weight 1, meaning that formula ξ is verified. On the contrary, if no position verifies formula φ, all the runs will end in state $\mathtt{ko}_\xi$, meaning that ξ is not verified.

It remains to prove that the class of layered pebble weighted automata is closed under the weighted constructs of wFO+wTC$^\mathrm{b}$(FO). First, notice that the constant series $s \in \mathbb{S}$ is easily recognized by a 0-layered PWA, using the guide to go from $v^{(i)}$ to $v^{(f)}$ and computing the weight s at the beginning of the unique run. Note also that if a series is recognizable by a K-layered pebble weighted automaton, it is also recognizable by a $(K+1)$-layered pebble weighted automaton. Therefore, given two layered pebble weighted automata, one can assume that they have the same number of layers, and the same set of pebble names.

Let $\mathcal{A}_1, \mathcal{A}_2$ be two K-layered pebble weighted automata over A. Closure under $\oplus$ is as usual obtained using the disjoint union of the automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

The K-layered pebble weighted automaton that recognizes the Hadamard product which maps every graph G and valuation σ to $[\![\mathcal{A}_1]\!](G,\sigma) \otimes [\![\mathcal{A}_2]\!](G,\sigma)$ consists of three phases: first, it simulates the automaton $\mathcal{A}_1$ until it reaches the final vertex $v^{(f)}$ in a final state of $\mathcal{A}_1$; then, using the guide, it reaches back the initial vertex of the graph, unless $v^{(f)} = v^{(i)}$ which it can test by using the basic test formula init? on its current position; finally, it simulates the automaton $\mathcal{A}_2$ and exits in a final state of $\mathcal{A}_2$. Notice that the guide is only used in the topmost layer.

For first-order quantifiers, we use an extra layer. From a K-layered pebble weighted automaton $\mathcal{A}$ computing $[\![\Phi(x)]\!]$, we construct two $(K+1)$-layered pebble weighted automata computing respectively $[\![\bigoplus_x \Phi]\!]$ and $[\![\bigotimes_x \Phi]\!]$.

Automaton for $\bigoplus_x \Phi$ follows the guide, and at some point nondeterministically (but not going over $v^{(f)}$, and unambiguously as the guide visits exactly once every vertex) chooses a vertex in the graph where it drops pebble x. Then, it simulates automaton $\mathcal{A}$. After this simulation, it lifts the pebble and terminates the computation, going to the final vertex of the graph using again the guide. Notice that the guide visiting each vertex exactly once, the automaton computes exactly the semantics of $\bigoplus_x \Phi$.

For $\bigotimes_x \Phi$, the new automaton drops successively pebble x on every position of the input graph (again using the guide). Whenever it drops pebble x, it simulates $\mathcal{A}$ until it reaches the final vertex where it lifts the pebble.

Finally, assume that we have a K-layered pebble weighted automaton $\mathcal{A}$ which evaluates some wFO+wTC$^{\mathrm{b}}$(AP) formula $\Phi(x,y)$. We construct a $(K+2)$-layered pebble weighted automaton $\mathcal{A}'$ in order to evaluate $[\mathrm{wTC}^M_{x,y}\Phi](x',y')$: x' and y' are now two free variables. Following the guide, $\mathcal{A}'$ moves to the vertex $v_{x'}$ holding the free variable x'. It drops pebble x. Whenever $\mathcal{A}'$ drops pebble x on some vertex v_x, it enters some special state q_{drop_x}. Starting again from the beginning, it searches for vertex v_x with the guide, and then moves to some guessed vertex v_y at a distance less or equal to M, remembering the sequence of directions it uses to reach it. It drops pebble y. In order to compute the transitive closure in an unambiguous way, the automaton now starts to check that the path it followed from v_x to v_y is the *minimal* one amongst those that lead from v_x to v_y: the minimiality criterion we consider is the lexicographic order over the sequences of directions, supposing an order has been fixed over D. To check this, $\mathcal{A}'$ explores deterministically the (finite) set of sequences of directions of length at most M in the lexicographic order: it stops the first time it reaches vertex v_y and rejects the current run if the sequence of directions is not the one recorded before. Now, this sequence can be flushed from the memory: this is particularly interesting for complexity reasons we study after this proof. $\mathcal{A}'$ then simulates $\mathcal{A}$ resulting in the evaluation of $\Phi(v_x, v_y)$. When the simulation of $\mathcal{A}$ is completed, $\mathcal{A}'$ uses the guide to return to vertex v_y. Then, $\mathcal{A}'$ non-deterministically moves to vertex v_x where pebble x was dropped remembering the sequence of directions in the reverse order between both vertices. Again, in order to simulate the transitive closure unambiguously, it checks that the guessed sequence of directions is the minimal one in the lexicographic order. $\mathcal{A}'$ lifts pebble y and pebble x, and then returns to vertex v_y by using the memorized sequence of directions. If y' is held by the current vertex, $\mathcal{A}'$ enters a final state. Also – even if y' is at the current vertex – $\mathcal{A}'$ drops pebble x again in order to continue the evaluation of the transitive closure. □

Notice that the layered pebble weighted automaton constructed in this proof has a number of states *linear* with respect to the size of the formula Φ, defined as the number of operators used in Φ, and denoted by $|\Phi|$ in the following. In particular, the constants M appearing in the bounded transitive closuresare not considered for the complexity. We suppose also that the size of $\mathcal{A}_{\mathfrak{G}}$ is constant. In that case, every closure construction adds only a fixed finite number of states.[2] This is an interesting result even in the Boolean case as it shows that we can construct *unambiguous* layered pebble automata for every formula of first order logic (and even with some transitive closures).

Remark 6.11. Notice that it would have been possible to add some other atomic Boolean formulae φ and still get the result of Theorem 6.10, as soon as we have an unambiguous automaton $\mathcal{B}^{\varphi}$ equivalent to it and its negation. We cite two non-trivial examples here.

Consider the class of trees and a Boolean formula over variables x and y, stating that x is an ancestor of y, i.e., that there exists a $\{\downarrow_i \mid i \geq 0\}$-path from x to y. The unambiguous automaton for this formula first moves to vertex holding x, and then executes a depth-first search of the subtree rooted in x until either it finds y and accepts, or reaches back x and rejects.

As a second example, consider the class of pictures and a Boolean formula over variables x and y, stating that y is at the bottom-right of x, i.e., that there exists a $\{\downarrow, \rightarrow\}$-path from x to y. Getting an equivalent unambiguous automaton now requires the use of a pebble: indeed, starting from the vertex holding x, we search in every row below and on the right of x if y appears. This is done by marking with a pebble the vertex below x on this row and searching the row from left to right. The pebble is then moved from one row to the following one until the last row. ■

As a side effect of this translation from logic to automata, it is possible to evaluate a formula of wFO+wTC$^{\mathrm{b}}$(FO) by using the evaluation of layered pebble weighted automata, presented in Theorem 5.4, and get the following complexity result:

Corollary 6.12. *Let $\mathfrak{G}$ be a searchable class of pointed graphs. We can compute the semantics $[\![\Phi]\!](G,\sigma)$ of a formula $\Phi \in$ wFO+wTC$^{\mathrm{b}}$(FO) that uses p distinct variables, over a pointed graph $G \in \mathfrak{G}$, for a valuation σ with domain containing the free variables of $\mathcal{A}$, with $\mathcal{O}(|\Phi| \times |V|^{p+1})$ scalar star operations and $\mathcal{O}((p+1) \times |\Phi|^3 \times |V|^{p+3})$ scalar sums and products in $\mathbb{S}$.*

This result can be made more efficient when specialized for words, trees or nested words, accordingly to what is showed in Chapter 5.

6.5 From Automata to Logics

This section is devoted to a more theoretical result translating automata formalisms back into logics. Mainly, the aim is to show robustness of the automaton model we exhibited, even though the monadic second order logic is still unreachable by this mean.

Proofs of the next theorems are partly based on ideas present in [Tho82] and [EH07]. The idea is to encode runs of automata into a formula using transitive closures. Interestingly, the weighted extension we present – moreover in the case of graphs – leads to new problems in this encoding: it is not sufficient to be able to simulate at least *one* run for every accepted structure, but this simulation has to be done faithfully with respect to the weights.

We split the proof into two parts. First, we define the notion of zonable classes of graphs, and prove that all of the graph classes we are interested in are zonable. Then, abstracting the technical difficulties inherent of this definition, we start from a zonable class of graphs and prove that we can construct a formula equivalent to a given automaton, for all the graphs of this class.

[2] Notice that if M is taken into account for the complexity, then the construction for the transitive closure requires an additional number of states of the form $|D|^M$ to remember the paths.

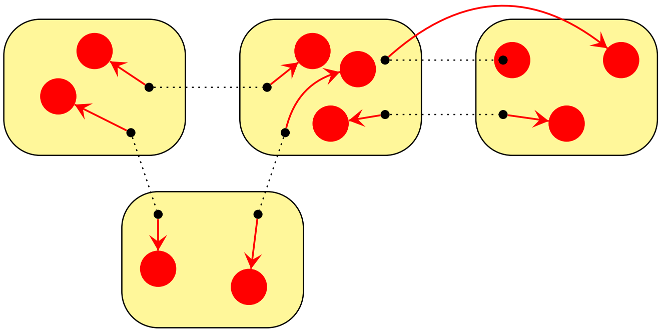


Figure 6.6: Zone partitioning of a graph: zones are related by wires depicted with dashed lines. The encoding of wire (v, v'), for every integer $n \in [0\,..\,N-1]$ is depicted by a red area linked to the vertex v', or containing vertex v'.

6.5.1 Zonable Classes of Graphs

Zonability is a combinatorial notion we introduce to cut a graph into small *zones*, however wide enough to encode each edge relating two distinct zones into one of these two zones. This zone decomposition will be used later to encode runs of a weighted automaton navigating in the graphs, abstracting the internal behaviors of a single zone, and simply considering the jump through edges relating two distinct zones. Henceforth, we also need the zonability to be uniformally computable by a fixed formula for the class.

Definition 6.13. Let A be an alphabet and D be a set of directions. A class $\mathfrak{G}$ of pointed graphs $\mathcal{G}(A, D)$ is said to be *zonable* if for every natural number[3] $N \in \mathbb{N}$, there exists a bound $M \in \mathbb{N}$ such that for every pointed graph $G = (V, (E_d)_{d\in D}, \lambda, v^{(i)}, v^{(f)}) \in \mathfrak{G}$, there exists

- an equivalence relation $\sim$ over V such that equivalence classes of $\sim$, which are called *zones*, have a diameter bounded by M, i.e., for every pair of vertices v, v' of the zone there exists a path composed of vertices of the zones from v to v' of length bounded by M. In the following, edges $(v, v') \in E$ such that $v \not\sim v'$ are called *wires*, and the set of wires of a graph are denoted by $\mathcal{W}$;
- an injective mapping $f\colon \mathcal{W} \times [0\,..\,N-1] \to V$ such that for every wire (v, v') and every integer $n \in [0\,..\,N-1]$, the vertex $f((v, v'), n)$ is in the same zone as v or v'.

Moreover, for $\mathcal{L}$ a weighted logic, the class $\mathfrak{G}$ is said to be *$\mathcal{L}$-zonable* if these objects can be effectively and uniformally definable by unambiguous formulae of $\mathcal{L}$: for every natural number $N \in \mathbb{N}$, we must have an unambiguous formula $\mathsf{same\text{-}zone}(z_1, z_2)$ of $\mathcal{L}$ with free variables z_1 and z_2, and for every $n \in [0\,..\,N-1]$, an unambiguous formula $\mathsf{f}(z_1, z_2, n) = x$ of $\mathcal{L}$ with free variables x, z_1 and z_2, verifying that for every pointed graph $G \in \mathfrak{G}$:

- $v \sim v'$ if, and only if, $[\![\mathsf{same\text{-}zone}(z_1, z_2)]\!](G, [z_1 \mapsto v, z_2 \mapsto v']) = 1$;
- $f((v_1, v_2), n) = v$ if, and only if, $[\![\mathsf{f}(z_1, z_2, n) = x]\!](G, [z_1 \mapsto v_1, z_2 \mapsto v_2, x \mapsto v]) = 1$.

■

Figure 6.6 depicts a zone partition of the vertices of a graph. In the sequel, wires will be depicted with the same dashed convention, and the red area linked to a vertex v part of a wire (v, v') depicts the set of vertices $f((v, v'), n)$ (with $n \in [0\,..\,N-1]$ used to encode this wire.

Notice that a zone, being of diameter bounded by M, must be of size bounded by $|D|^M$, since the graphs we study are of bounded degree. Notice in particular that if a graph has a number of vertices bounded by M – which is verifyable by a formula of FO for example

[3] Natural number N will encode the number of states of the automaton to translate into the logic.

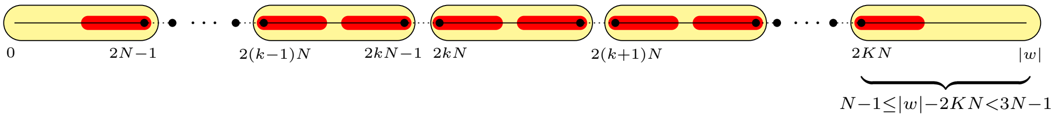


Figure 6.7: Zone partitioning of a word and description of the encoding function f

– then we can consider that there is a single zone containing all the vertices, and hence no wires. Therefore, in the examples we give below, we only consider graphs having at least $M+1$ vertices.

It has to be noticed that wires relate two distinct zones of the graph. Hence, they can be characterized by the formula

$$\mathsf{wire}(z,z') = \bigoplus_x \bigoplus_{n\in[0\,..\,N-1]} (\mathsf{f}(z,z',n)=x)$$

which is an unambiguous formula as f is an injection.

Before stating the translation theorems, we give a non-exhaustive list of zonable classes of graphs.

Words

Zones of words will be subwords of length $2N$ (see Figure 6.7), except the last zone that may contain at most positions $3N-1$ positions: hence each zone has a diameter bounded by $M = 3N-1$. They can be described using modulo computations: henceforth, we define formula $\mathsf{same\text{-}zone}(z_1,z_2)$ by

$$\bigoplus_x (\mathsf{init},x) \equiv_{\rightarrow} 0[2N] \otimes \Big[\bigoplus_{0\le k_1,k_2<2N} R^{k_1}_{\rightarrow}(x,z_1)\otimes R^{k_2}_{\rightarrow}(x,z_2) \oplus \bigoplus_{0\le k<3N} R^{k}_{\rightarrow}(x,\mathsf{final})\otimes \bigoplus_{2N\le k_1,k_2<k} R^{k_1}_{\rightarrow}(x,z_1)\otimes R^{k_2}_{\rightarrow}(x,z_2)\Big]\,.$$

Moreover, wires will simply be edges of the form $(2kN-1,2kN)$ or $(2kN,2kN-1)$ (except possibly the last ones). For every integer $n\in[0\,..\,N-1]$, we set $f((2kN-1,2kN),n) = 2kN-1-n$, and $f((2kN,2kN-1),n) = 2kN+n$. This defines an injection as wires are separated by a distance of $2N$. These functions may be defined by the following $\mathsf{f}(z_1,z_2,n) = x$ formula:

$$\begin{aligned}&\big[R_{\rightarrow}(z_1,z_2)\otimes R^{n}_{\rightarrow}(x,z_1)\otimes((\mathsf{init},z_2)\equiv_{\rightarrow} 0[2N])\big]\\ \oplus\, &\big[R_{\leftarrow}(z_1,z_2)\otimes R^{n}_{\rightarrow}(z_1,x)\otimes((\mathsf{init},z_1)\equiv_{\rightarrow} 0[2N])\big]\end{aligned}$$

Hence, $\mathcal{W}\mathrm{ord}(A)$ is a wFO+wTC$^{\mathrm{b}}$(AP)-zonable class of graphs.

Pictures

Similar ideas to cut pictures into zones have been used for other purposes in [Mat98]. Zones of pictures will be square subpictures of width $4N$ (see Figure 6.8), except the zones on the right and the bottom of the pictures that may be a little larger as in the case of words. The largest zone possible is the one on the right bottom corner which can have width and height bounded above by $6N-1$. Hence, each zone has a diameter bounded by $M = 2\times(6N-1)-1$. Similarly to the previous case, this can be tested using modulo computations. Forgetting, for the sake of simplicity, about the zones on the right and on the bottom, we obtain as

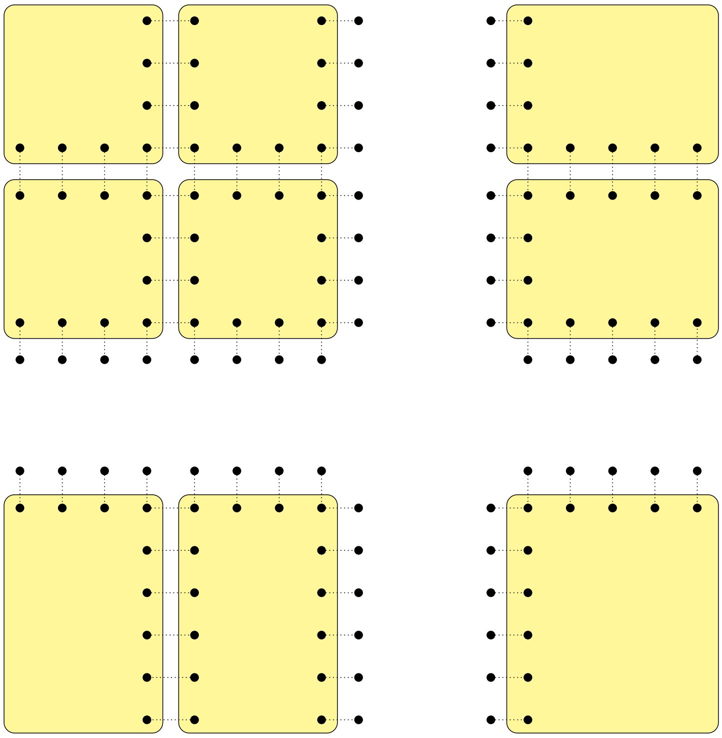


Figure 6.8: Zone partitioning of a picture

formula $\mathsf{same\text{-}zone}(z_1, z_2)$:

$$\begin{aligned}&\textstyle\bigoplus_{x,y}(\mathsf{init}, x) \equiv_{\rightarrow} 0[4N] \otimes (x, y) \equiv_{\downarrow} 0[4N]\\ &\otimes \textstyle\bigoplus_z \bigoplus_{0\leq k_1,k_2<4N} R_{\rightarrow}^{k_1}(y, z) \otimes R_{\downarrow}^{k_2}(z, z_1)\\ &\otimes \textstyle\bigoplus_z \bigoplus_{0\leq k_1,k_2<4N} R_{\rightarrow}^{k_1}(y, z) \otimes R_{\downarrow}^{k_2}(z, z_2)\end{aligned}$$

Notice that in this formula, y denotes the position at the upper left corner of the zone containing z_1 and z_2. Each zone (except the larger ones) has at most $4 \times 4N$ wires, and we will encode a wire in the zone from which it exits (as for the case of words): hence each zone must have at least $4 \times 4N \times N = (4N)^2$ positions available which is exactly the case. Deciding of a decodable order between the wires, it is easy to design a formula $\mathsf{f}(z_1, z_2, n) = x$ for every $n \in [0 \mathinner{..} N - 1]$: for example, we may consider a partitioning of each zone into 4 disjoint rectangles of height N each reserved for the wires of one border of the zone. This shows that $\mathcal{P}\mathrm{ict}(A)$ is a wFO+wTC$^{\mathrm{b}}$(AP)-zonable class of graphs.

Trees

In ranked trees, we consider zones to be subtrees of height at least $2N$ with roots at height 0 modulo $2N$: in particular, every subtree having a root at height 0 modulo $2N$ which has height less than $2N$ is not a zone and hence belongs to the zone which is above it. However, it is easy to check that each zone has a height bounded by $2 \times 2N - 1$, and hence a diameter bounded by $M = 2\times(4N-1)-1$. Notice that the height of a node modulo $2N$ is computable by a formula of wFO+wTC$^{\mathrm{b}}$(AP), as it suffices to check the length modulo $2N$ of the *unique* path leading from the root to the node, which can be done by formula $\mathsf{height}(x) \equiv k[2N]$

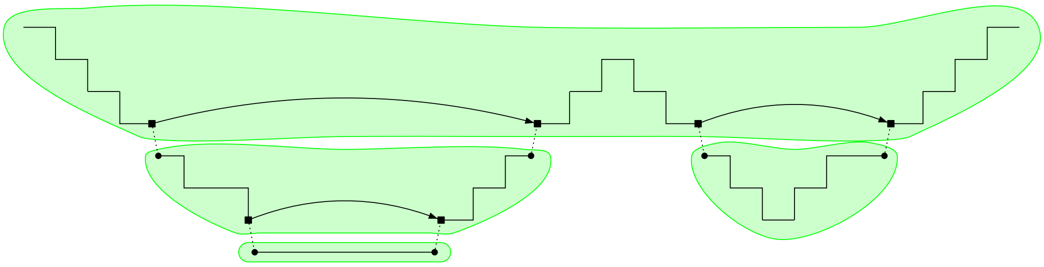


Figure 6.9: Chunk decomposition of a nested word: special calls and returns are the square vertices, whereas special wires are depicted with dashed lines. There are 4 chunks, each of call-depth at most $2 \times (2N)$.

defined by

$$\bigoplus\nolimits_z (x,z) \in R_\uparrow^k \otimes \Big([\mathrm{wTC}_{x,y}^{2N}(x,y) \in R_\uparrow^{2N}](z, \mathsf{init}) \oplus z = \mathsf{init}\Big)$$

where we have denoted $(x,y) \in R_\uparrow^\ell$ the formula

$$\bigoplus\nolimits_{z_1,\dots,z_{\ell-1}} (x, z_{\ell-1}) \in R_\uparrow \otimes (z_{\ell-1}, z_{\ell-2}) \in R_\uparrow \otimes \cdots \otimes (z_2, z_1) \in R_\uparrow \otimes (z_1, y) \in R_\uparrow$$

and $(x,y) \in R_\uparrow$ the formula $(x,y) \in R_{\uparrow_1} \oplus \cdots \oplus (x,y) \in R_{\uparrow_K}$ with K the maximal arity of a letter in the ranked alphabet A.

Wires are edges that enter or exit the root of a zone: hence, each root, except the root of the whole tree is part of exactly two wires relating the root to its parent. Notice that it is not possible, as for the case of words, to encode a wire in the zone from which it exits. Indeed, a zone may be connected to an exponential number of wires – the wires at the bottom – and the zone may not contain enough nodes to encode them all. Instead, we encode a wire in the subtree of the root. Hence, each zone must encode at most $2 \times N$ pairs, which is possible as it contains at least this number of vertices (considering its height). Again, it is easy to decide of a decodable order and to design a formula $\mathsf{f}(z_1, z_2, n) = x$: for example, we may consider the leftmost branch of this subtree of length $2N$ (which exists considering the hypothesis on its height) and use the N first to encode the entering wire, and the others to the exiting wire. This proves that $\mathcal{T}\mathrm{ree}(A)$ is a wFO+wTC$^\mathrm{b}$(AP)-zonable class of graphs.

Nested Words

Let $G \in \mathcal{N}\mathrm{est}(A)$ be a nested word of length n. Notice first that we cannot use the zone partitioning of the underlying word: indeed, since there are nesting edges, each zone may then be wired to too many zones, and it would then be impossible to encode all the wires.

Instead, we describe zones using two levels. First, we cut the nested words into *chunks*: a chunk is a connected subset of vertices having call-depth of the form $2Nk + i$ for a fixed k and $i \in [0 \mathinner{..} 2N - 1]$. For the same reasons than for trees, chunks that are have a size less than $4N$ – in particular, they necessarily have a call-depth bounded by $2N$ – are glued to the chunks *above* them. This implies that every chunk has a call-depth bounded by $2 \times 2N$. Every chunk (unless the topmost one with smallest call-depth) is surrounded by a pair of *special vertices*, related by a nesting edge. A chunk decomposition is depicted in Figure 6.9: the special vertices are represented by squares. More formally, we call v a *special call* (respectively, a *special return*) if it is a call vertex (respectively, a return vertex) of call-depth a multiple of $2N$, such that there are at least $4N$ positions in-between the vertex and its matched call (respectively return). We call *special wires* the edges relating one chunk to another: these wires necessarily touch some special vertex.

Notice that we can test if a vertex x has a call-depth equal to 0 modulo $2N$ with the following expression starting at x:

$$x?\Big(\big(\curvearrowleft?{\to}(\curvearrowright{\to}+\neg(\curvearrowright?\vee\curvearrowleft?){\to})^\star\curvearrowleft?\big)^{2N}\Big)^\star{\to}(\curvearrowright{\to}+\neg(\curvearrowright?\vee\curvearrowleft?){\to})^\star\text{final}?$$

Intuitively, this expression iterates as many times as necessary a sequence of $2N$ moves decrementing of one unit the call-depth. Each decrement is composed of a $\to$-edge at a return node, followed by the iteration of some moves not changing the call-depth: either a $\curvearrowright$ followed by a $\to$, or a $\to$ move taking place on an internal vertex (neither a call or a return). After this iteration, we simply verify that the final vertex has the same-call depth than the current vertex.

One first attempt to prove that nested words are zonable could be to express with a logical formula this expression, transforming each Kleene star iteration into a bounded transitive closure. However, this is not immediate, since the external Kleene star iteration is *unbounded* in the sense that it can make unbounded steps, because of the internal Kleene star.

Henceforth, we test if a vertex x has a call-depth equal to 0 modulo $2N$ with a more advanced method. We indeed use as a tool *summary paths*. The summary path issued from a vertex v consists in the shortest path from position v to the final vertex of the nested word. Indeed, it is the only path from v that jumps through the $\curvearrowright$-edges in call vertices, and otherwise follow the $\to$-edges. On the nested word depicted in Figure 2.3 for example, the summary path from vertex 2 (first vertex labeled by b) visits exactly sequentially the vertices 2, 3, 4, 5, 11, 12 and 13. We will say that this vertex is at a summary-distance 6 from the final vertex. More generally, the summary-distance between a vertex v and the final vertex is defined as the length of the summary path issued from v.

We can verify that a vertex x is at summary-distance 0 modulo $2N$ from the final vertex with the following expression:

$$x?\big((\curvearrowright+\neg(\curvearrowright?){\to})^{2N}\big)^\star\text{final}?\,.$$

Notice that this Kleene star iteration is bounded, so that we can design an equivalent formula $\sigma\text{-dist}(x)\equiv 0[2N]$ of wFO+wTC$^\text{b}$(AP) for this expression:

$$\big[\mathrm{wTC}^{2N}_{x_0,y_{2N}}(\textstyle\bigoplus_{x_1,\dots,x_{2N-1}}\displaystyle\bigotimes_{0\leq i<2N}\sigma\text{-step}(x_i,x_{i+1})\big](x,v^{(f)})$$

with

$$\sigma\text{-step}(x,y)=R_\curvearrowright(x,y)\oplus\neg(\textstyle\bigoplus_z R_\curvearrowright(x,z))\otimes R_\to(x,y)\,.$$

The key ingredient is that we can moreover *compute* the call-depth of a vertex v by following the summary-path issued from v. Consider first a vertex v_0 at a summary-distance 0 modulo $2N$ from the final vertex. We denote by $(v_k)_{0\leq k\leq K}$ the ordered sequence of vertices of the summary path issued from v_0, with $v_K=v^{(f)}$. Using the hypothesis over v_0, we know that K is a multiple of $2N$. We will maintain the call-depth modulo $2N$ of the vertices v_{2iN} for $i\in[0\mathinner{..}K/2N]$, checking at the end that the call-depth of vertex v_K we computed is 0. We finally explain how to compute the call-depth modulo $2N$ of the vertices v_{2iN} with a bounded transitive closure: the transitive closure jumps over vertices $v_{2iN}+c$ that encode the call-depth c modulo $2N$ of vertices v_{2iN}. Henceforth, we must simply design a formula $\Phi(x',y')$ with x' encoding $v_{2iN}+c_i$ and y' encoding $v_{2(i+1)N}+c_{i+1}$:

$$\begin{aligned}\Phi(x',y')=\textstyle\bigoplus_{x_0,\dots,x_{2N}}&\bigoplus_{k_0,\dots,k_{2N}}\bigotimes_{0\leq i<2N}\beta_{k_i,k_{i+1}}(x_i,x_{i+1})\\&\otimes\sigma\text{-dist}(x_0)\equiv 0[2N]\otimes\sigma\text{-step}^{k_0}(x',x_0)\otimes\sigma\text{-step}^{k_{2N}}(y',x_{2N})\end{aligned}$$

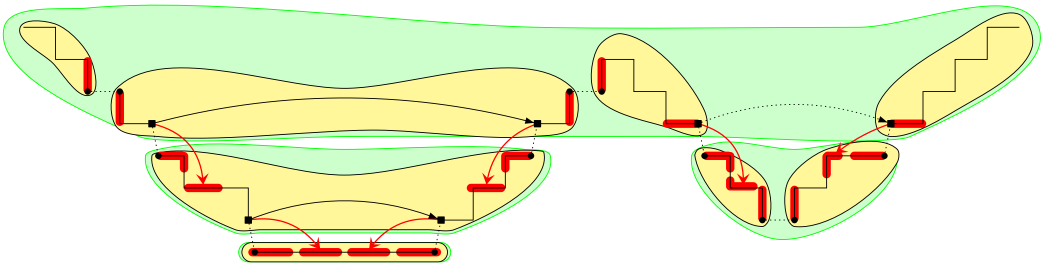


Figure 6.10: Zone partitioning of a nested word and encoding of wires: the special call and returns on the right are parts of 4 wires (2 special wires and 2 non-special ones), and henceforth linked to two zones of encoding each.

where[4]

$$\begin{aligned}
\beta_{k,k}(z,z') &= R_{\curvearrowright}(z,z') \oplus \big(\neg(\textstyle\bigoplus_t R_{\curvearrowright}(z,t)) \otimes R_{\rightarrow}(z,z') \otimes \neg(\textstyle\bigoplus_t R_{\curvearrowright}(t,z'))\big)\\
\beta_{k,k-1}(z,z') &= \neg(\textstyle\bigoplus_t R_{\curvearrowright}(z,t)) \otimes R_{\rightarrow}(z,z') \otimes \textstyle\bigoplus_t R_{\curvearrowright}(t,z')
\end{aligned}$$

and we denote by σ-step$^k(x,y)$ the iteration k times of formula σ-step, which is easily definable in wFO+wTC$^{\mathrm{b}}$(AP).

Then, formula

$$\big[\mathrm{wTC}^{4N}_{x',y'}\Phi(x',y')\big](x-c, v^{(f)})$$

when evaluated with x mapped to vertex v_0 (at summary-distance 0 modulo $2N$ from the final vertex), verifies that this vertex has call-depth equal to c modulo $2N$. Notice that variables x and y must be linked by a piece of the summary-path of length $2N$, so that variables x' and y' are at a distance at most $4N$, justifying the bound on the transitive closure. Finally, we are able to design a formula $\mathsf{c\text{-}d}(x) \equiv 0[2N]$ verifying that vertex x has call-depth 0 modulo $2N$, by first considering the first vertex v_0 on the right of x with summary-distance 0 modulo $2N$ from the initial vertex, and by applying the previous formula with c the call-depth of v_0 deduced from the hypothesis that x has call-depth 0 modulo $2N$.

This is very easy to check that a call and its underlying return are separated by at least $4N$ positions, so that we can design formulae $\mathsf{sc}(x)$ and $\mathsf{sr}(y)$ of wFO+wTC$^{\mathrm{b}}$(AP) checking that x and y are respectively a special call and a special return.

Chunks may be arbitrarily large, and hence may have an unbounded diameter. To overcome this difficulty, we cut each chunk into zones by grouping vertices of a chunk into sets of $3N$ vertices, following the linear order. As for words, the last zone of a chunk may have between $3N$ and $4N-1$ vertices. In any case, each zone has a diameter bounded by $M = 4N-1$, and we may design a formula $\mathsf{same\text{-}zone}(z_1, z_2)$ similarly to the word case.

Finally, wires may be of two types: either a wire relates two vertices of different chunks (called special wires previously), in which case they are encoded into the chunks containing the vertices of greatest call-depth, or a wire relates two vertices of the same chunks, in which case we follow the same kind of encoding as for words. Notice that every zone may have many wires, but only hosts the vertices encoding at most two of them.

An example of zone partitionning, as well as the encoding of wires, is depicted in Figure 6.10.

This finishes the outline of the proof that $\mathcal{N}\mathrm{est}(A)$ is a wFO+wTC$^{\mathrm{b}}$(AP)-zonable class of graphs.

[4] In the second formula, if $k = 0$, we assume that $k-1$ is equal to $2N-1$, as we count modulo $2N$.

Mazurkiewicz Traces

We finally explain the zonability for the class $\mathcal{MT}\mathrm{race}_{\mathrm{proc}}(A)$ of Mazurkiewicz traces, with a fixed distribution $\mathrm{proc}\colon A \to \mathfrak{P}([1\mathinner{..}n])$ of the finite alphabet of rendez-vous or internal actions, into the sets of processes it concerns. We recall that the set of directions is $D = \{\downarrow_p, \uparrow_p \mid p \in [1\mathinner{..}n]\}$, i.e., we can follow the linear orders of each process.

Notice first that there exists a formula $x < y$ whose semantics is exactly the partial order of a trace. Indeed, a vertex v is smaller in this partial order that v' if there exists some vertices $v = v_1, v_2, \ldots, v_n, v_{n+1} = v'$ and a permutation σ of the n processes $[1\mathinner{..}n]$ such that for all $i \in [1\mathinner{..}n]$, there exists a $\downarrow_{\sigma(i)}$-path from v_i to v_{i+1}.[5] Then we can easily deduce the following formula $x < y$ of FO:

$$\exists x_1, \ldots, x_{n+1}\ (x_1 = x) \wedge (x_{n+1} = y) \wedge \bigvee_{\substack{\sigma \text{ permutation}\\ \text{of } [1\mathinner{..}n]}} \bigwedge_{1 \leq i \leq n} R_{\downarrow_{\sigma(i)}}(x_i, x_{i+1})\,.$$

As for nested words, the zone partition of a Mazurkiewicz trace is achieved in a hierarchical way: we first split the trace considering only the first process, then considering the second one, etc. For each process, the splitting is done using the computation of some distances modulo a fixed integer on the linear order of this process, as for words. The idea is to consider the *history* of a vertex of the trace. In a trace $G \in \mathcal{MT}\mathrm{race}_{\mathrm{proc}}(A)$, the history of vertex v is the set of vertices that are smaller than or equal to v in the partial order of the trace.

For a given natural number N, we now define the zone splitting of a trace G, with a bound $M = 4n^2N$. We first consider the set V_1 of vertices attached to process 1. On the linear order defined by $\downarrow_1$, we consider all the vertices $(v_i)_{0 \leq i \leq K_1}$ at distance 0 modulo $2nN$ from the minimial element of the process: in particular $v_i < v_{i+1}$ for every $i \in [0\mathinner{..}K_1 - 1]$. As for words, we may not consider the deepest such vertex if it is at a distance less than nN from the last vertex of the process. Those vertices define a partition of the vertices of the trace, by considering the histories they generate: we attach to vertex v_i the set of vertices in the history of v_i that are not in the history of v_{i-1} (in case $i \neq 0$). The last set of vertices of the partition consists of all the vertices that are not in the history of v_{K_1}. The classes defined by this partition are called 1-classes below.

Notice that a 1-class is necessarily connected, and moreover, the restriction to every process is also connected: if it contains at least one vertex of process $p \in [1\mathinner{..}n]$, we call *minimum* and *maximum* of the 1-class with respect to process p the smallest and the greatest vertices of process p in the class, respectively. If a 1-class contains at most M vertices, it is of diameter bounded by M, then it is considered as a zone. Otherwise, we further split the 1-class. Indeed, if the 1-class contains at least $4n^2N + 1$ vertices, there should exist a process p such that at least $4nN + 1$ vertices of the class are attached to process p: moreover, p must be different to 1 since each 1-class contains at most $3nN$ vertices. We consider the smallest $p \in [2\mathinner{..}n]$ verifying this property. We then split considering the set V_p of vertices attached to process p. More precisely, we denote by $(v_i)_{0 \leq i \leq K_2}$ the vertices attached to process p at distance 0 modulo $2nN$ from the minimum of the 1-class with respect to process p. Then, we split the 1-class into several p-classes with the same process than before. We may continue this construction until each class is indeed a zone (notice that we split the classes by increasing order of processes which ensures the termination).

We now explain how to build the formula $\mathsf{same\text{-}zone}(z_1, z_2)$. The checking is done by searching for the hypothetical zone in which z_1 and z_2 could be, by considering the successive p-classes in which the two vertices are. Indeed, every zone is contained in a succession of p-classes, and hence, we represent this zone with the sequence of pairs of minimum

[5] The definition of the partial order permits to use any $\{\downarrow_p \mid p \in [1\mathinner{..}n]\}$-path, but we can transform any such path into the succession of some $\downarrow_{\sigma(i)}$-paths for a given permutation σ: in fact, the path comes back to a process p already visited before, we can simply replace the path in-between by a succession of $\downarrow_p$ moves, and hence simplify the path.

and maximum with respect to process p of each p-class in which it is contained. We first consider the 1-class in which z_1 and z_2 lie: this is done in a similar way as for words, i.e., by considering the successive vertices attached to process 1 at a distance 0 modulo $2nN$ to the minimal element, and stopping when we obtain a vertex that has z_1 and z_2 in its history. We then check the size of the 1-class obtained, to know whether we should continue to split the current class or not. If it is the case, the process p along which we should split can be computed easily, and then, the following splitting is done similarly by considering the distance modulo the minimal element of the class with respect to p.

Finally, the encoding of the wires is very similar to the one of words. Indeed, each zone contains at most 2 wires per process, and hence at most $2n$ wires. We encode a wire in the zone from which it exits (as for words) so that $2nN$ vertices are required. Notice, from the construction above, that each p-class contains at least $2nN$ vertices attached to process p. Hence, the formula $\mathsf{f}(z_1, z_2, n) = x$ is written by considering any computable ordering of those vertices and the wires of the zone.

This finishes the outline of the proof that $\mathcal{MT}\mathrm{race}_{\mathrm{proc}}(A)$ is a wFO+wTC$^{\mathrm{b}}$(AP)-zonable class of graphs.

6.5.2 Logical Characterization of Weighted Automata

Theorem 6.14. *Let $\mathfrak{G}$ be a* wFO+wTC$^{\mathrm{b}}$(FO)*-zonable class of graphs in $\mathcal{G}(A, D)$. Then, from every weighted automaton $\mathcal{A}$, we can construct a formula $\Phi_{\mathcal{A}}$ of* wFO+wTC$^{\mathrm{b}}$(FO)*, with* $\mathrm{Free}(\Phi_{\mathcal{A}}) = \mathrm{Free}(\mathcal{A})$*, that is equivalent to $\mathcal{A}$ over $\mathfrak{G}$, i.e., such that for every graph $G \in \mathfrak{G}$ and every valuation σ of the free variables of $\mathcal{A}$, we have $[\![\mathcal{A}]\!](G, \sigma) = [\![\Phi_{\mathcal{A}}]\!](G, \sigma)$.*

Proof. From a weighted automaton $\mathcal{A}$ with set of states Q, we construct a formula of wFO+wTC$^{\mathrm{b}}$(FO) in several steps. First, we will use an outermost transitive closure in order to make macro steps in the graph, and then compute the weight of the runs in-between macro steps with a tower of transitive closures, inspired by the Kleene theorem. The main difficulty in this second step is that Kleene iterations in our weighted expressions are not naturally bounded (in the sense that they perform jumps of possibly unbounded length): otherwise, we could mimick proof of [EH07] and construct a formula of wFO+wTC(FO) equivalent to the automaton. The current impossibility to translate this fragment of logic back into our automaton model prevents us to use this simpler proof. As a way to overcome this challenge, we cut the input graph into zones that have a bounded diameter. Then, the outermost transitive closure simply jumps from one zone to an adjacent one, and the inner ones simulate the behavior of automaton $\mathcal{A}$ over a zone of bounded diameter, using McNaughton-Yamada construction [MY60] for example: this ensures that every transitive closure operator performs only bounded jumps.

Using formula $\mathsf{same\text{-}zone}(z_1, z_2)$ and $\mathsf{f}(z_1, z_2, n) = x$ for $N = |Q|$, we may design a formula Ξ_{q_i,q_f} that computes the weight of the runs from q_i at initial vertex to q_f at final vertex. Then, the semantics of the automaton $\mathcal{A}$ is obtained by considering the formula

$$\bigoplus_{q_i,q_f} I_{q_i} \otimes \Xi_{q_i,q_f} \otimes \Big(\bigoplus_{\alpha \in \mathrm{Test}} \varphi_\alpha(\mathsf{final}) \otimes F_{q_f}(\alpha) \Big) .$$

In this formula and in the following ones, we denote by $\varphi_\alpha(z)$ the FO formula – with free variables z and the free variables of α – that permits to verify the local test $\alpha \in \mathrm{Test}$ over the vertex denoted by z. We can easily associate such a formula with every test by induction, letting

$$\varphi_\top(z) = \top, \quad \varphi_{\mathrm{init}?}(z) = \mathsf{init}(z), \quad \varphi_{\mathrm{final}?}(z) = \mathsf{final}(z)$$
$$\varphi_{a?}(z) = P_a(z), \quad \varphi_{d?}(z) = \exists y \; R_d(z, y), \quad \varphi_{x?}(z) = (x = z)$$

and by directly translating the Boolean closure from test formulae to FO formulae.

We first describe a formula $\mathsf{single\text{-}transition}_{q,q'}(z_1, z_2)$ that computes the sum of the weights of the transitions going from vertex z_1 in state q to vertex z_2 in state q':

$$\mathsf{single\text{-}transition}_{q,q'}(z_1, z_2) = \bigoplus_{d\in D,\alpha\in\mathrm{Test}} (\varphi_\alpha(z_1) \wedge R_d(z_1, z_2)) \otimes \Delta_{q,q'}(d)(\alpha)\,.$$

We will then use a transitive closure to compute the weights of the runs that remain in a given zone, knowing the first and last vertices and the states. This transitive closure will be bounded, since the zone is supposed to be of diameter bounded by M. More precisely, under the condition that vertices z_1 and z_2 are in the same zone Z, formula $\Phi_{q,q'}(z_1, z_2)$ computes the sum of the weights of the non-empty runs, that remain inside zone Z, going from vertex z_1 in state q to vertex z_2 in state q'. To build such a formula, we follow the McNaughton-Yamada algorithm, i.e., we generate formulae $\Phi^R_{q,q'}(z_1, z_2)$ with $R \subseteq Q$, computing the sum of the weights of the non-empty runs, that remain inside zone Z, going from vertex z_1 in state q to vertex z_2 in state q', that visit as intermediary states only states of R. We will then let $\Phi_{q,q'}(z_1, z_2) = \Phi^Q_{q,q'}(z_1, z_2)$. We construct these formulae by induction over R. As a base case, we have

$$\Phi^\emptyset_{q,q'}(z_1, z_2) = \mathsf{single\text{-}transition}_{q,q'}(z_1, z_2)\,.$$

Then, if $q'' \notin R$, the runs from q to q' with intermediary states in $R \cup \{q''\}$ either do not visit state q'' or visit it at least once. Hence, we may compute their weights with formula

$$\begin{aligned}\Phi^{R\cup\{q''\}}_{q,q'}(z_1, z_2) = \Phi^R_{q,q'}(z_1, z_2) \oplus \textstyle\bigoplus_{z_1',z_2'} & \mathsf{same\text{-}zone}(z_1, z_1') \otimes \Phi^R_{q,q''}(z_1, z_1')\\ & \otimes \big([\mathrm{wTC}^M_{x,y}(\Phi^R_{q'',q''}(x, y))](z_1', z_2') \oplus (z_1' = z_2')\big)\\ & \otimes \mathsf{same\text{-}zone}(z_2', z_2) \otimes \Phi^R_{q'',q'}(z_2', z_2)\,.\end{aligned}$$

Finally, formula Ξ_{q_i,q_f} initializes the run and then uses a transitive closure operator to jump from a zone to the following one (see Figure 6.11):

$$\begin{aligned}\Xi_{q_i,q_f} = \bigoplus_{q,q'\in Q} \textstyle\bigoplus_{z_1,z_2,z_1',z_2'} & \Big[\mathsf{same\text{-}zone}(\mathsf{init}, z_1) \otimes \Phi_{q_i,q}(\mathsf{init}, z_1)\\ & \otimes \textstyle\bigoplus_{x',y'} \big(\mathsf{f}(z_1, z_1', q) = x' \otimes \mathsf{f}(z_2, z_2', q') = y' \otimes [\mathrm{wTC}^{3M}_{x,y}\Psi](x', y')\big)\\ & \otimes \mathsf{same\text{-}zone}(z_2', \mathsf{final}) \otimes \bigoplus_{q''\in Q} \mathsf{single\text{-}transition}_{q',q''}(z_2, z_2') \otimes \Phi_{q'',q_f}(z_2', \mathsf{final})\Big]\end{aligned}$$

with $\Psi(x, y)$ the formula

$$\begin{aligned}\textstyle\bigoplus_{z_1,z_2,z_1',z_2'} \bigoplus_{q,q'\in Q} & \Big[(\mathsf{f}(z_1, z_2, q) = x) \otimes (\mathsf{f}(z_1', z_2', q') = y)\\ & \otimes \textstyle\bigoplus_{q''\in Q} \mathsf{single\text{-}transition}_{q,q''}(z_1, z_2) \otimes \Phi_{q'',q'}(z_2, z_1')\Big]\,.\end{aligned}$$

In formula Ψ as well as in the final part of formula Ξ_{q_i,q_f}, as single transition has to be performed in order to jump through the wire: indeed, formula $\Phi_{q,q'}$ only remains inside a zone and hence cannot perform it.

Notice that variables x and y in formula Ψ must be at a distance bounded by $3M$ since each zone has a diameter M and position encoding a wire must be in one of the two zones adjacent to this wire. Formula $\Psi(x, y)$ simply makes the first step jumping over the wire, before using formula $\Phi_{q'',q'}$ computing the weights of all the interesting parts of runs. The correctness of this construction follows from the fact that semirings are distributive: indeed, we merge the whole set of runs into subsets visiting the same sequence of wires. □

Notice that the set of free variables of formula produced by this proof is exactly the set of free variables of the test formula appearing in the description of the automaton, which is exactly the set of free variables of the automaton.

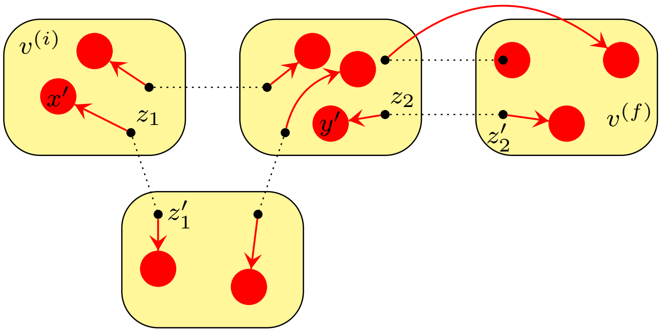


Figure 6.11: Instantiation of formula Ξ_{q_i,q_f}

6.5.3 Logical Characterization of Weighted Automata with Pebbles

We now generalize the previous result to also deal with pebbles.

Theorem 6.15. *Let $\mathfrak{G}$ be an* wFO+wTC$^{\mathrm{b}}$(FO)*-zonable class of graphs in $\mathcal{G}(A,D)$. Then, from every layered pebble weighted automaton $\mathcal{A}$, we can construct an equivalent formula $\Phi_{\mathcal{A}}$ of* wFO+wTC$^{\mathrm{b}}$(FO)*, with* $\mathrm{Free}(\Phi_{\mathcal{A}}) = \mathrm{Free}(\mathcal{A})$*, equivalent to $\mathcal{A}$ over $\mathfrak{G}$, i.e., such that for every graph $G \in \mathfrak{G}$ and valuation σ of the free variables of $\mathcal{A}$, we have $[\![\mathcal{A}]\!](G,\sigma) = [\![\Phi_{\mathcal{A}}]\!](G,\sigma)$.*

Proof. Very little has to be done with respect to the proof in the case without pebbles. Indeed, the proof goes by an induction on the number of layers of automaton $\mathcal{A}$. The case of 0 layer is done in Theorem 6.14.

Then, considering a $(K+1)$-layered automaton $\mathcal{A}$, we apply the same construction as before over the states of layer $K+1$, and simply enriching the formula $\mathsf{single\text{-}transition}_{q,q'}(z_1,z_2)$ to also deal with the pebbles. This formula must now compute the sum of the weights of the transitions from z_1 in state q to z_2 in state q' if $z_1 \neq z_2$, and if $z_1 = z_2$ must compute the sum of the weights of the runs from z_1 in state q to z_1 in state q' with intermediary states in lower layers. By induction, for every states q'_1 and q'_2 of layer K, we have a formula $\Xi^x_{q'_1,q'_2}(z_1)$ that computes the sum of the weights of the runs only visiting states of layers at most K from the initial vertex in q'_1 to the final vertex in q'_2, initially with pebble x dropped on vertex z_1. Hence, we now define $\mathsf{single\text{-}transition}_{q,q'}(z_1,z_2)$ by

$$\begin{aligned}
&(z_1 \neq z_2) \otimes \Big[\bigoplus_{d\in D,\alpha\in \mathrm{Test}} (\varphi_\alpha(z_1) \wedge R_d(z_1,z_2)) \otimes \Delta_{q,q'}(d)(\alpha)\Big] \\
\oplus\, &(z_1 = z_2) \otimes \Big[\bigoplus_{\substack{q'_1,q'_2\in Q\\ x\in\mathrm{Var}\\ \alpha_1,\alpha_2\in\mathrm{Test}}} (\varphi_{\alpha_1}(z_1) \wedge \varphi_{\alpha_2}(\mathsf{final})) \otimes \Delta_{q,q'_1}(\mathsf{drop}_x)(\alpha_1) \\
&\qquad\qquad \otimes \Xi_{q'_1,q'_2}(x \mapsto z_1) \otimes \Delta_{q'_2,q'}(\mathsf{lift})(\alpha_2)\Big]
\end{aligned}$$

where $\Xi_{q'_1,q'_2}(x \mapsto z_1)$ denotes the formula $\Xi_{q'_1,q'_2}$ where free occurrences of x in the test formulae are replaced by z_1. In particular, if a variable remains free in the automaton $\mathcal{A}$, it will not be replaced in such a way in the construction, so that it will be a free variable of the formula produced. □

6.6 Hybrid Navigational Logics

As a conclusion of this chapter dealing with logics, we now present some alternative ways of specifying quantitative properties. These are based on Propositional Dynamic Logic (PDL) introduced by Fischer and Ladner in [FL79], and that we formally define later, or XPath: this may seem very close to our weighted expressions, and this is indeed the case, as we designed the latter keeping PDL or XPath in mind. The main difference is the clear separation between state formulae and path formulae (also called programs in the literature). Moreover, in our weighted extension, we clearly separate the Boolean fragment from the weighted part, as we did for the previous logics. In weighted expressions, we inserted no Boolean fragment: henceforth this is the biggest novelty and advantage of this new fragment of logic. As we did for weighted expressions though, we then consider adding some expressive power by adding some variables that can be placed on the graph temporarily: this is refered in the literature as Hybrid Proposition Dynamic Logic (HPDL) (see, e.g., [ABM00, FdRS03]). Again, we give some efficient translation from these logics to weighted automata with pebbles. As a corollary, this may permit to apply the evaluation algorithms presented in Chapter 5, and hence extends algorithms presented in [FL79, Lan05] for model checking PDL formulae against some fixed Kripke structures.

Indeed, we may alternatively have considered linear temporal logics and their extensions by first-order quantifications, automata, or expressions as done for the Boolean case by [LS07]. We have investigated this direction in [4]: it can be seen as a special case of the weighted propositional dynamic logic we present in this manuscript. The following example simply presents some ideas specific to temporal logics.

Example 6.16. We consider in this example the probabilistic Linear Temporal Logic (LTL) presented in Chapter 1. Using pebbles in probabilistic expressions or automata is a natural and powerful way to deal with nesting in LTL formulae. Indeed, temporal logics implicitly use a free variable to denote the position where a formula has to be evaluated. We will mark this position with a pebble, say x, in expressions $E_\varphi(x)$ or automata $\mathcal{A}_\varphi(x)$ associated with LTL formulas φ.

Consider an LTL formula $\mathsf{F}\,\varphi$, for *Finally* φ. Given a word w and a position i in w, we are interested in the probability $[\![\mathsf{F}\,\varphi]\!](w,i)$ that φ holds in w at position i. As reminder, the semantics has been defined by

$$[\![\mathsf{F}\,\varphi]\!](w,i) = \sum_{j\geq i}\Big(\prod_{i\leq \ell<j}[\![\neg\varphi]\!](w,\ell)\Big)\times[\![\varphi]\!](w,j)\,.$$

For every LTL formula φ, we are aiming at an equivalent hybrid weighted expression $E_\varphi(x)$ which evaluates to $[\![\varphi]\!](w,i)$ over word w when pebble x marks position i: in particular, we require the expression to run from the initial position to the final one. Let us illustrate this inductive construction for LTL formulas. For $\mathsf{F}\,\varphi$, we set

$$E_{\mathsf{F}\,\varphi}(x) = \mathrm{init}?{\rightarrow^\star}x?\big(\big(y!E_{\neg\varphi}(y)\big){\rightarrow}\big)^\star\big(y!E_\varphi(y)\big){\rightarrow^\star}\mathrm{final}?\,.$$

The expression starts at the beginning of the word and moves to the right until it discovers the position marked with variable x. Then, for each $j\geq i$, it iterates $j-i$ times the computation of $\neg\varphi$ with the current position marked by y, moving to the right between two computations. Finally, it computes φ with $y!E_\varphi(y)$ before moving to the last position of the word. Indeed, we only need a single variable name, by using the reusability, so that we let

$$E_{\mathsf{F}\,\varphi}(x) = \mathrm{init}?{\rightarrow^\star}x?\big(\big(x!E_{\neg\varphi}(x)\big){\rightarrow}\big)^\star\big(x!E_\varphi(x)\big){\rightarrow^\star}\mathrm{final}?\,.$$

Similarly, for $\mathsf{G}\,\varphi$, we have $[\![\mathsf{G}\,\varphi]\!](w,i) = \prod_{j\geq i}[\![\varphi]\!](w,j)$, leading to the simpler expression

$$E_{\mathsf{G}\,\varphi}(x) = \mathrm{init}?{\rightarrow^\star}x?\big(\big(x!E_\varphi(x)\big){\rightarrow}\big)^\star\mathrm{final}?\,.$$

The last test (final?) is useful to enforce the preceding star operation to capture the whole suffix of the word from the position marked by x.

Finally, the expression for the *Until* modality is

$$E_{\varphi\mathsf{U}\psi}(x) = \mathrm{init}?{\rightarrow^\star}x?\big(\big(x!(E_{\neg\psi}(x){\leftarrow^\star}E_\varphi(x))\big){\rightarrow}\big)^\star\big(x!E_\psi(x)\big){\rightarrow^\star}\mathrm{final}?\,.$$

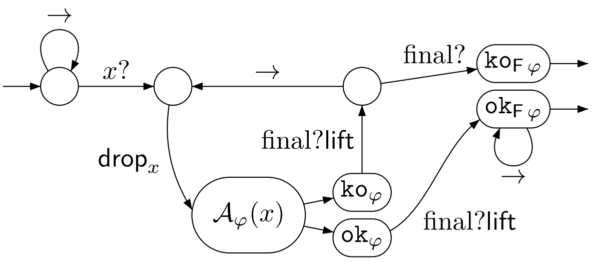


Figure 6.12: Automata for the *Finally* operator

In terms of automata, let us assume – as in the proof of Theorem 6.10 – that, for every formula φ, there is an automaton $\mathcal{A}_\varphi$ with two designated terminal states, `ok` and `ko`, such that runs ending in `ok` (and at the end of the word) compute expression E_φ and those ending in `ko` compute expression $E_{\neg\varphi}$. The figure below depicts the pebble weighted automaton for the modality *Finally*. The one for *Globally* is obtained by considering the equivalence $\mathsf{G}\,\varphi = \neg\,\mathsf{F}(\neg\varphi)$, i.e., by exchanging the states $\mathtt{ok}_\varphi$ and $\mathtt{ko}_\varphi$, as well as the two final states. ■

6.6.1 Weighted Propositional Dynamic Logic

First, we present the Boolean fragment of formulae from Propositional Dynamic Logic that we consider. It is composed of state formulae α and *local* path formulae (or programs) π defined by

$$\alpha ::= \top \mid \text{init} \mid \text{final} \mid a \mid x \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid E\pi$$
$$\pi ::= \bot \mid \text{stay} \mid d \mid \pi;\pi \mid \alpha?\,\pi : \pi$$

where $a \in A$, $d \in D$ and $x \in \text{Var}$. We let DPDL (for *Deterministic Propositional Dynamic Logic*) the set of state formulae hence defined. The determinism comes from the fact that we can see a program π as deterministic once the state formula α are evaluated (no disjunction or Kleene iteration over programs in particular). We consider this restriction for technical reasons that we stress later.

The semantics of a state formula α is a set of vertices of a graph equipped with a valuation of free variables. The semantics of a path formula π is a set of pairs of vertices of a graph equipped with a valuation of free variables. These semantics are defined in Table 6.3 for a given pointed graph $G \in \mathcal{G}(A, d)$, for vertices v and v', and a valuation σ of domain containing the free variables of the formula.

Remark 6.17. The path formula $\alpha?\,\text{stay} : \bot$ is necessarily non moving, in the sense that if $G, \sigma, v, v' \models \alpha?\,\text{stay} : \bot$ then $v = v'$, and is indeed equivalent to the state formula α. ■

The fact that we disallow disjunction and Kleene iteration of path formulae implies the following deterministic condition:

Lemma 6.18. *Programs of* DPDL *are deterministic, i.e., for every program* $\pi \in$ DPDL*, graph* G*, valuation* σ*, and vertices* v*,*

- *either there exists a unique vertex* v' *such that* $G, \sigma, v, v' \models \pi$*;*
- *or for all vertices* v'*,* $G, \sigma, v, v' \not\models \pi$*.*

Proof. The proof goes by induction over the program $\pi \in$ DPDL.

If $\pi = \bot$, the second case holds. If $\pi = \text{stay}$, the first case holds with $v' = v$. If $\pi = d$, either $d \in \mathsf{type}(v)$ (d is enabled in v), in which case the first case holds with v' the unique vertex such that $(v, v') \in E_d$, or $d \notin \mathsf{type}(v)$, in which case the second case holds.

If $\pi = \pi'; \pi''$, by applying the induction hypothesis to π', we may have two cases:

Table 6.3: Semantics of DPDL formulae

$G, \sigma, v \models \top$
$G, \sigma, v \models \text{init}$ if, and only if, $v = v^{(i)}$
$G, \sigma, v \models \text{final}$ if, and only if, $v = v^{(f)}$
$G, \sigma, v \models a$ if, and only if, $\lambda(v) = a$
$G, \sigma, v \models x$ if, and only if, $\sigma(x) = v$
$G, \sigma, v \models \neg\alpha$ if, and only if, $G, \sigma, v \not\models \alpha$
$G, \sigma, v \models \alpha \wedge \alpha'$ if, and only if, $G, \sigma, v \models \alpha$ and $G, \sigma, v \models \alpha'$
$G, \sigma, v \models \alpha \vee \alpha'$ if, and only if, $G, \sigma, v \models \alpha$ or $G, \sigma, v \models \alpha'$
$G, \sigma, v \models E\pi$ if, and only if, there exists $v' \in V$ such that $G, \sigma, v, v' \models \pi$

$G, \sigma, v, v' \not\models \bot$
$G, \sigma, v, v' \models \text{stay}$ if, and only if, $v = v'$
$G, \sigma, v, v' \models d$ if, and only if, $(v, v') \in E_d$
$G, \sigma, v, v' \models \pi; \pi'$ if, and only if, there exists $v'' \in V$ such that
$G, \sigma, v, v'' \models \pi$ and $G, \sigma, v'', v' \models \pi'$
$G, \sigma, v, v' \models \alpha?\,\pi : \pi'$ if, and only if, $G, \sigma, v \models \alpha$ and $G, \sigma, v, v' \models \pi$,
or $G, \sigma, v \not\models \alpha$ and $G, \sigma, v, v' \models \pi'$

- either there exists a unique vertex v' such that $G, \sigma, v, v' \models \pi'$: the property then follows from the induction hypothesis applied to π'' starting in vertex v';
- or for all vertex v', $G, \sigma, v, v' \not\models \pi'$, in which case the same property holds for π.

Finally, for $\pi = \alpha?\,\pi' : \pi''$, there are two cases again:

- either $G, \sigma, v \models \alpha$, in which case the result follows from the induction hypothesis applied to π';
- or $G, \sigma, v \not\models \alpha$, in which case the result follows from the induction hypothesis applied to π''.

□

On top of that fragment of logic, we add weights by mimicking the Boolean operators with a weighted semantics.

Definition 6.19. We denote by WPDL the class of *weighted propositional dynamic logic* defined by

$$\varphi ::= s \mid \varphi \oplus \varphi \mid \varphi \otimes \varphi \mid \Sigma\psi$$
$$\psi ::= d \mid \alpha?\,\psi : \psi \mid \varphi \mid \psi \cdot \psi \mid \psi + \psi \mid \psi^+$$

where $s \in \mathbb{S}$, $a \in A$, $d \in D$ and $\alpha \in \text{DPDL}$. ■

As for weighted expressions, the semantics will map a pointed graph, a valuation and either one or two vertex(ices) to a weight of a continuous semiring $\mathbb{S}$, as described in Table 6.4. As for weighted expressions, we denote in this table, ψ^n the n-th iterate of ψ defined inductively by

$$\psi^1 = \psi \qquad \text{and} \qquad \psi^{n+1} = \psi^n \cdot \psi \text{ for } n \in \mathbb{N}\,.$$

Remark 6.20. Notice that if φ and φ' are state formulae of WPDL, the weighted path formula $\alpha?\,\varphi : \varphi'$ is necessarily non moving and is indeed equivalent to the state formula $\Sigma(\alpha?\,\varphi : \varphi')$. This is the reason why we did not introduce construction $\alpha?\,\varphi : \varphi'$ directly in the weighted state formulae. ■

Every weighted expression of WE can be seen as a path formula of WPDL where the Boolean path formulae are disallowed. In particular, notice that a Boolean state formula α

Table 6.4: Semantics of WPDL

$$\begin{aligned}
[\![s]\!](G,\sigma,v) &= s\\
[\![\varphi \oplus \varphi']\!](G,\sigma,v) &= [\![\varphi]\!](G,\sigma,v) \oplus [\![\varphi']\!](G,\sigma,v)\\
[\![\varphi \otimes \varphi']\!](G,\sigma,v) &= [\![\varphi]\!](G,\sigma,v) \otimes [\![\varphi']\!](G,\sigma,v)\\
[\![\Sigma\psi]\!](G,\sigma,v) &= \bigoplus_{v'\in V} [\![\psi]\!](G,\sigma,v,v')
\end{aligned}$$

$$\begin{aligned}
[\![d]\!](G,\sigma,v,v') &= \begin{cases} 1 & \text{if } G,\sigma,v,v' \models d\\ 0 & \text{otherwise.}\end{cases}\\
[\![\alpha?\,\psi : \psi']\!](G,\sigma,v,v') &= \begin{cases} [\![\psi]\!](G,\sigma,v,v') & \text{if } G,\sigma,v \models \alpha\\ [\![\psi']\!](G,\sigma,v,v') & \text{otherwise.}\end{cases}\\
[\![\varphi]\!](G,\sigma,v,v') &= \begin{cases} [\![\varphi]\!](G,\sigma,v) & \text{if } v = v'\\ 0 & \text{otherwise.}\end{cases}\\
[\![\psi \cdot \psi']\!](G,\sigma,v,v') &= \bigoplus_{v''\in V} [\![\psi]\!](G,\sigma,v,v'') \otimes [\![\psi']\!](G,\sigma,v'',v')\\
[\![\psi + \psi']\!](G,\sigma,v,v') &= [\![\psi]\!](G,\sigma,v,v') \oplus [\![\psi']\!](G,\sigma,v,v')\\
[\![\psi^+]\!](G,\sigma,v,v') &= \bigoplus_{n\geq 1} [\![\psi^n]\!](G,\sigma,v,v')
\end{aligned}$$

can be transformed into a non moving path formula by the construction $\alpha?\,1:0$. However, this logic seems strictly more expressive, by the use of a richer Boolean fragment, and the *summation* operator $\Sigma\psi$. We now prove that we can efficiently translate this logic into the formalism of pebble weighted automata.

Theorem 6.21. *Let $\mathbb{S}$ be a continuous semiring and $\mathfrak{G}$ be a searchable class of graphs. For each path formula ψ in* WPDL*, we can construct a layered pebble weighted automaton (using a single variable) $\mathcal{A} \in$* PWA($\mathbb{S}$) *with* Free($\mathcal{A}$) = Free(ψ)*, equivalent to ψ, i.e., $[\![\psi]\!](G,\sigma,v,v') = [\![\mathcal{A}]\!](G,\sigma,v,v')$ for every pointed graph G, every valuation σ with domain containing* Free(ψ) *and every vertices v and v'.*

Proof. Once again the proof contains two separate parts: the constructions for the Boolean part, and the ones for the weighted operators.

We first explain the constructions for the Boolean part DPDL. As in Theorem 6.10, we produce, for every Boolean state formula α, a layered pebble weighted automaton $\mathcal{B}^\alpha$ with one initial state ι and two (final) states `ok` and `ko` such that for all pointed graph $G \in \mathfrak{G}$, every vertex v and every valuation σ,

$$\begin{aligned}
[\![\mathcal{B}^\alpha_{\iota,\mathtt{ok}}]\!](G,\sigma,v,v') &= \begin{cases} 1 & \text{if } G,\sigma,v \models \alpha \text{ and } v = v'\\ 0 & \text{otherwise,}\end{cases}\\
[\![\mathcal{B}^\alpha_{\iota,\mathtt{ko}}]\!](G,\sigma,v,v') &= \begin{cases} 1 & \text{if } G,\sigma,v \not\models \alpha \text{ and } v = v'\\ 0 & \text{otherwise.}\end{cases}
\end{aligned}$$

Notice that contrary to the previous proof, automaton $\mathcal{B}^\alpha$ starts in vertex v (which is not encoded as a free variable anymore), and must accept on this vertex.

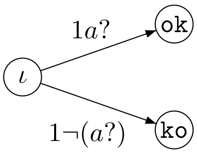


Figure 6.13: Automaton for state formula a

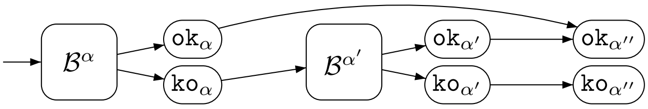


Figure 6.14: Weighted automaton for disjunction $\alpha'' = \alpha \vee \alpha'$

The constructions for all the formulae but the new construct $E\pi$ are very similar to the proof for first-order formulae[6]. E.g., for the atom $\alpha = a$, the automaton is depicted in Figure 6.13, whereas the disjunction $\alpha \vee \alpha'$ is depicted in Figure 6.14.

For a formula $\alpha = E\pi$, we follow an exploration close to the first-order existential quantification. Indeed, we start by dropping a pebble x on the current position, in order to be able to come back to it at the end of the computation. However, during the computation of the path formula π we do not use the variable x. When the pebble is dropped, we use the guide to find the pebble again. We now explain how to translate the program π into an automaton $\mathcal{B}^\pi$ with a single initial state and two final states `ok` and `ko` again. For every graph G, valuation σ, and vertices v, there are two possibilities thanks to Lemma 6.18

- either there exists a unique vertex v' such that $G, \sigma, v, v' \models \pi$ in which case, for every $v'' \in V$:

$$[\![\mathcal{B}^\pi_{\iota,\mathtt{ok}}]\!](G,\sigma,v,v'') = \begin{cases} 1 & \text{if } v'' = v' \\ 0 & \text{otherwise}, \end{cases}$$
$$[\![\mathcal{B}^\pi_{\iota,\mathtt{ko}}]\!](G,\sigma,v,v'') = 0\,;$$

- or for all vertex v', $G, \sigma, v, v' \not\models \pi$ in which case there exists a unique vertex v' such that for all $v'' \in V$:

$$[\![\mathcal{B}^\pi_{\iota,\mathtt{ok}}]\!](G,\sigma,v,v'') = 0$$
$$[\![\mathcal{B}^\pi_{\iota,\mathtt{ko}}]\!](G,\sigma,v,v'') = \begin{cases} 1 & \text{if } v'' = v' \\ 0 & \text{otherwise}. \end{cases}$$

Automaton $\mathcal{B}^\pi$ is produced similarly to the proof of Theorem 4.9 (by induction over π), and the proof works since the programs are supposed to be deterministic. Indeed, automata for atoms $\perp$, stay and d are depicted in Figure 6.15, whereas the one for the concatenation is depicted in Figure 6.16. Notice that it verifies the desired property since, either π is *not executable* (the second case above), and then the exit point is uniquely determined by it, or π is executable (the first case above) and reach a unique vertex from which we can apply the property to π': however, it can be the case that $\pi; \pi'$ is not executable but still do not accept in state $\mathtt{ko}_{\pi''}$ in the vertex in which it started. Finally, making use of automaton $\mathcal{B}^\alpha$ constructed inductively, we give an automaton for program $\alpha?\pi : \pi'$ in Figure 6.17: again the property is verified for this automaton because the test α is either true or false at the current position and then we can use the inductive property for π and π'. Finally, we

[6] Here again, we allow the use of non-moving transitions, knowing that we can remove them with standard techniques, as explained in page 93.

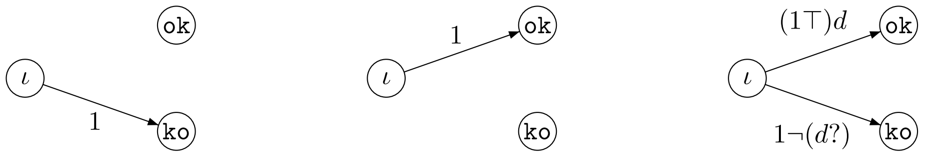


Figure 6.15: Automata for program $\bot$, stay and d

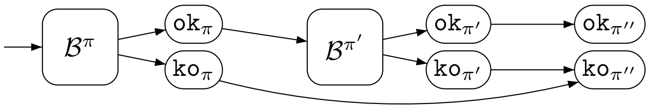


Figure 6.16: Automaton for program $\pi'' = \pi; \pi'$

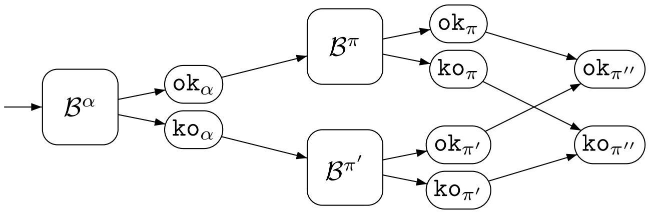


Figure 6.17: Automaton for program $\pi'' = \alpha?\,\pi : \pi'$

obtain the automaton for state formula $E\pi$ as explained before by dropping a pebble, using the guide $\mathcal{A}_{\mathfrak{G}}$ to find it, simulating automaton $\mathcal{B}^{\pi}$ and lifting the pebble to come back to the original vertex (indeed, we need to use the guide again to reach the final vertex before lifting the pebble, but we forget about this search in the picture). This construction is depicted in Figure 6.18.

It remains to translate the weighted constructions of WPDL. We construct for every state formula φ an automaton $\mathcal{A}_{\varphi}$ such that $[\![\mathcal{A}_{\varphi}]\!](G, \sigma, v, v') = 0$ if $v \neq v'$ and

$$[\![\mathcal{A}_{\varphi}]\!](G, \sigma, v, v) = [\![\varphi]\!](G, \sigma, v)\,.$$

Simultaneously, we construct for every path formula ψ an automaton $\mathcal{A}_{\psi}$ such that

$$[\![\mathcal{A}_{\psi}]\!](G, \sigma, v, v') = [\![\psi]\!](G, \sigma, v, v')\,.$$

These constructions are done with a simulatneous induction.

For base cases, namely s for state formulae and d for path formulae, this is done as for weighted expressions in Theorem 4.9. The sum $\varphi \oplus \varphi'$ and the product $\varphi \otimes \varphi'$ are translated respectively with a non-deterministic choice or a sequential composition: we strongly use at that point that automata $\mathcal{A}_{\varphi}$ and $\mathcal{A}_{\varphi'}$ come back to their original position at the end of their computation. For the $\Sigma\psi$ formula, exactly the same construct as for $E\pi$ is done

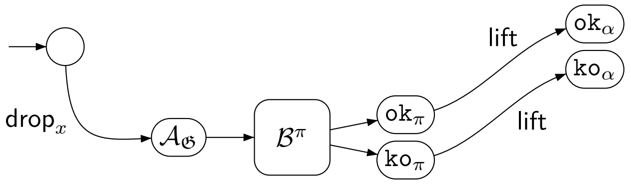


Figure 6.18: Automaton for state formula $\alpha = E\pi$

in this weighted case: supposing that automaton $\mathcal{A}_\psi$ is constructed, the automaton starts by dropping a pebble at the current vertex, uses the guide to find it, and then simulates automaton $\mathcal{A}_\psi$ (ignoring the pebble however); at the end of its computation, we simply lift the pebble (after reaching the final vertex by using the guide again) to come back to the original vertex (since $\Sigma\psi$ is a state formula). Path formulae are translated using exactly the same constructions as in the proof of Theorem 4.9: notice that we add the *if-then-else* construct that simply use automaton $\mathcal{B}^\alpha$ before launching one of the two continuations depending on the result obtained. □

The number of layers of the automaton constructed is linear in the depth of operators Σ in the formula. Moreover, the number of states of $\mathcal{A}$ is linear in the size of ψ. We strongly used the determinism of the Boolean path formulae DPDL in order to obtain this result. Indeed, notice that we could extend the fragment of Boolean path formulae as long as the result of Lemma 6.18 holds: for example, a construction $(\alpha?\pi)^\star;\neg\alpha$, iterating the program π *as many times as* α holds, would still be deterministic and enable the same proof, modulo some cycle checking. As an example, we refer to Chapter 8 where expressions are made deterministic (before adding to them probabilities) in a more general setting.

6.6.2 Hybrid Weighted Propositional Dynamic Logic

In Hybrid Propositional Dynamic Logics, variables are used to mark temporarily some positions in order to be able to come back to it later in the computation. This is similar to the variable feature we adopted in hybrid weighted expressions. Indeed, we simply extend WPDL with an operator $x!\psi$ whose semantics is the semantics of ψ from the initial vertex to the final one with a valuation updated with x mapping to the current vertex. As we restricted ourselves to local Boolean formulae, it is useless to add this operator in the Boolean setting.

Definition 6.22. We denote by HWPDL the class of *hybrid weighted propositional dynamic logic* defined by

$$\begin{aligned}\varphi &::= s \mid \varphi\oplus\varphi \mid \varphi\otimes\varphi \mid \Sigma\psi \mid x!\psi\\ \psi &::= d \mid \alpha?\,\psi:\psi \mid \varphi \mid \psi\cdot\psi \mid \psi+\psi \mid \psi^+\end{aligned}$$

where $s\in\mathbb{S}$, $a\in A$, $d\in D$, $x\in\mathrm{Var}$, and $\alpha\in\mathrm{DPDL}$. ■

We simply give the definition of the new operator $x!\psi$: for every graph G, valuation σ and vertices v, v', we let

$$[\![x!\psi]\!](G,\sigma,v,v') = \begin{cases}[\![\psi]\!](G,\sigma[x\mapsto v],v^{(i)},v^{(f)}) & \text{if } v=v'\,,\\ 0 & \text{otherwise.}\end{cases}$$

The proof of Theorem 6.21 coupled with the translation of the operator $x!-$ proved in Theorem 4.31 permits to translate efficiently every formula of this extended logic into layered pebble weighted automata.

Theorem 6.23. *Let $\mathbb{S}$ be a continuous semiring and $\mathfrak{G}$ be a searchable class of graphs. For each path formula ψ in* HWPDL*, we can construct a layered pebble weighted automaton $\mathcal{A}\in\mathrm{PWA}(\mathbb{S})$ with* $\mathrm{Free}(\mathcal{A})=\mathrm{Free}(\psi)$*, equivalent to ψ, i.e., $[\![\psi]\!](G,\sigma,v,v')=[\![\mathcal{A}]\!](G,\sigma,v,v')$ for every pointed graph G, every valuation σ with domain containing* $\mathrm{Free}(\psi)$*, and every vertices v and v'.*

CHAPTER 7

Simple and One-Way Restrictions of the Specification Languages

La semplicità è l'ultima forma della sofisticazione.[1]

Leonardo da Vinci

We consider in this chapter some restrictions of the specification languages that permit to consider non necessarily continuous semirings, and in particular $(\mathbb{R}, +, \times, 0, 1)$ and its subsemirings. In order to avoid infinite sums in the semantics, which are no longer well-defined, we have to limit the constructions of expressions and logics, and the power of automata. We mainly focus on automata. In the following, we will consider two possible directions. The first one is of a *semantical* nature, namely to consider a simple semantics of automata by taking into account only those runs that do not contain *cycles*. Another orthogonal possibility would consist in a *syntactical* restrictions over automata, we consider one-way pebble weighted automata, being able to move in ordered graphs only in the forward direction. Indeed, we will see that these two restrictions generate the same behaviors, at least for the word case.

The first section gives a formal definition of a simple semantics of automata, disallowing cyclic runs. In Section 2, we define one-way pebble weighted automata. In both cases, weights are taken from a (non necessarily complete) semiring. We show in Section 3 that this relaxation over the semiring permits to turn the emptiness problem of an automaton undecidable very quickly. Finally, we prove that over words, the semantical and syntactical restrictions generate the same class of formal power series. Last section is devoted to some

[1] "Simplicity is the ultimate sophistication".

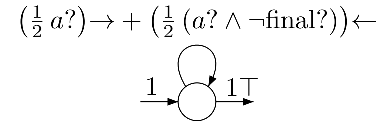

$\left(\frac{1}{2}\, a?\right)\!\rightarrow + \left(\frac{1}{2}\,(a? \wedge \neg\text{final?})\right)\!\leftarrow$
1
1⊤

Figure 7.1: A weighted automaton $\mathcal{A} \in \mathrm{WA}(\mathbb{R}^+ \cup \{+\infty\})$

perspectives. Most of the results are extracted from [1], and further extended to the setting of this manuscript.

7.1 Simple Semantics of Pebble Weighted Automata

A run (respectively a K-run) of a weighted automaton (respectively, a pebble weighted automaton) is said to be *simple* if it contains at most one occurrence of each configuration (respectively K-configuration). Equivalently, a run is not simple if it contains twice the same configuration.

Lemma 7.1. *The number of simple runs (respectively, K-runs) of a weighted automaton (respectively, a pebble weighted automaton) over a given pointed graph is finite.*

Proof. The number of configurations or K-configurations is finite, and hence so is the number of simple runs which are sequences of pairwise distinct configurations. □

The *simple semantics* of a weighted automaton or a pebble weighted automaton is then adapted from the semantics we defined before by only considering simple runs. In the following, it will be denoted by $[\![\mathcal{A}]\!]_{\mathrm{s}}$ for a weighted automaton $\mathcal{A}$ and $[\![\mathcal{A}]\!]_{\mathrm{s},K}$ for a pebble weighted automaton $\mathcal{A}$. Notice that this simple semantics may be defined even in non-continuous semirings, since the number of simple runs is finite and hence, this semantics only involves a finite summation. Before taking advantage of this relaxation, we consider in the two following propositions cases of continuous semirings. Over Boolean semirings first, this does not add any power.

Proposition 7.2. *Over the Boolean semiring, both semantics coincide. More precisely, for every pebble weighted automaton $\mathcal{A}$ over the Boolean semiring, and integer K (equal to* 0 *if $\mathcal{A}$ is a weighted automaton), we have $[\![\mathcal{A}]\!]_K = [\![\mathcal{A}]\!]_{\mathrm{s},K}$.*

Proof. In the Boolean semiring, a pointed graph G is mapped to 1 by the semantics $[\![\mathcal{A}]\!]_K$ if, and only if, there exists an accepting run of $\mathcal{A}$ over G that uses only stack of height at most K. If such an accepting run contains a loop, we may simply remove it, i.e., remove all configurations in-between the two occurrences of the same configuration as well as one occurrence of this latter. By repeating this process, we may construct a simple run, which permits to prove that a pointed graph G is mapped to 1 by the semantics $[\![\mathcal{A}]\!]_K$ if, and only if, there exists a *simple* accepting run of $\mathcal{A}$ over G that uses only stack of height at most K. □

This property is very specific to the Boolean semiring, as in other continuous semirings, simple semantics may restrict the power of a weighted automaton.

Proposition 7.3. *Over the continuous semiring* $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$*, there exists a series over finite words that is recognizable by a weighted automaton with the usual semantics, and not the simple one.*

Proof. Consider the weighted automaton $\mathcal{A}$ depicted in Figure 7.1 over alphabet $A = \{a\}$, already studied in Example 4.7.

We fix a pointed graph $G \in \mathcal{W}\mathrm{ord}(A)$ with $n > 0$ vertices. We now show that $[\![\mathcal{A}]\!](G) = \frac{1}{n}$. To prove it, let p_i be the sum of the weights of the runs that start in vertex $i \in [0 \mathinner{..} n-1]$ and end in vertex $n-1$ (in this vertex, any run is blocked as no transition of non-zero weight is still fireable). Indeed, $[\![\mathcal{A}]\!](G) = p_0$.

First, notice that $p_{n-1} = 1$ and $p_0 = \frac{1}{2}p_1$. Then, for every $i \in [1 \mathinner{..} n-2]$, we have

$$p_i = \frac{1}{2}p_{i-1} + \frac{1}{2}p_{i+1}$$

This recurrence relation admits as solutions the sequences

$$p_i = \lambda + \mu i$$

since the polynomial $X = \frac{1}{2} + \frac{1}{2}X^2$ admits 1 as only root. Then, using the seed equation $p_{n-1} = 1$, we obtain $\lambda + \mu(n-1) = 1$, and using the seed relation $p_0 = \frac{1}{2}p_1$, it comes $\lambda = \frac{1}{2}(\lambda + \mu)$. We deduce from the second that $\lambda = \mu$, and with the first conclude that

$$\lambda = \mu = \frac{1}{n}$$

From this, we obtain $p_0 = \lambda = \frac{1}{n}$.

Suppose now that there exists a weighted automaton $\mathcal{A}$ such that $[\![\mathcal{A}]\!]_{\mathrm{s}}(G) = \frac{1}{|V|}$ for every pointed graph $G \in \mathcal{W}\mathrm{ord}(A)$. Denoting by $s_1, \dots, s_k$ the set of non-negative reals appearing in the transition matrix and in the initial and final vectors of $\mathcal{A}$, we obtain that the set of $\{1/n \mid n \in \mathbb{N} \setminus \{0\}\}$ should then be entirely included into the $\mathbb{N}[s_1, \dots, s_k]$ the set of polynomials over $s_1, \dots, s_k$ with coefficients in $\mathbb{N}$. *A fortiori*, $\{1/n \mid n \in \mathbb{N} \setminus \{0\}\}$ is also included in ring generated by $s_1, \dots, s_k$ over $\mathbb{Z}$. As the ring generated by $\{1/n \mid n \in \mathbb{N} \setminus \{0\}\}$ over $\mathbb{Z}$ is $\mathbb{Q}$, this would mean that $\mathbb{Q}$ is included in a finitely generated ring over $\mathbb{Z}$, which is false (as the remark after this proof recalls). Finally, this proves that there is no weighted automaton $\mathcal{A}$ such that $[\![\mathcal{A}]\!]_{\mathrm{s}}(G) = \frac{1}{|V|}$ for every pointed graph $G \in \mathcal{W}\mathrm{ord}(A)$. □

Remark 7.4. We simply recall why $\mathbb{Q}$ cannot be included in a finitely generated ring K over $\mathbb{Z}$. If it was so, considering I a maximal ideal of K, we know that the quotient K/I would be a field (by maximality) which is finite (using, e.g., [Bou64], Corollary 1, page 64). By considering an homomorphism of the ring $\mathbb{Q}$ into the ring K, and by taking the quotient of K by I, we would obtain a homomorphism from the field $\mathbb{Q}$ to the finite field K/I: however, as a homomorphism of fields is injective, this leads to a contradiction. ■

The previous proposition shows that loops are indeed useful in weighted automata navigating in graphs (even without pebbles): this must be understood as the fact that the classical semantics is strictly more expressive than the simple one. However, we will see that this is no longer the case if we consider weighted automata (with or without pebbles) that may only visit ordered graphs in a one-way fashion.

7.2 One-way Pebble Weighted Automata

We restrict our attention to ordered pointed graphs in this section. Similarly to our study of one-way pebble weighted expression, we will first define a special class of automata that may run over these ordered graphs and generate a semantics even in non-continuous semirings. We will afterwards come back to the relation between these one-way pebble weighted automata and the pebble weighted automata equipped with the simple semantics.

Definition 7.5. A *one-way pebble weighted automaton* over a semiring $\mathbb{S}$ is a pebble weighted automaton $\mathcal{A} = (Q, A, D, I, \Delta, F)$ with D an ordered set of directions, verifying

- $\Delta_{q,q'}(d)(\alpha) = 0$ for every states $q, q' \in Q$, $\alpha \in \text{Test}$ and backward direction $d \in D_{\leftarrow}$;
- if $\Delta_{q,q'}(\mathsf{lift})(\alpha) \neq 0$ for some states $q, q' \in Q$, and test $\alpha \in \text{Test}$, then $\Delta_{q',q''}(\mathsf{drop}_x)(\alpha') = 0$ for every state $q'' \in Q$, $\alpha' \in \text{Test}$ and $x \in \text{Var}$.

The class of all one-way pebble weighted automata over $\mathbb{S}$ is denoted by 1PWA($\mathbb{S}$) or 1PWA if the semiring is clear from the context, or not relevant. ■

Notice that the second condition is easily justifiable: indeed, a lift transition followed by a drop transition would just result in resetting the current position to the initial vertex of the graph, without dropping a new pebble. Similarly, we may adapt this definition to let 1WA($\mathbb{S}$) be the class of one-way weighted automata over $\mathbb{S}$ (only the first condition is necessary in that case).

As subclasses of PWA and WA, these one-way (pebble) weighted automata inherit the notions of configurations and runs. We show that, over an ordered pointed graph, these automata have a finite number of accepting K-runs for every K. This will permit to define their semantics, even over non-continuous semirings. Indeed, we show that all runs of a one-way (pebble) weighted automaton are simple.

Lemma 7.6. *All runs of a one-way pebble weighted automaton are simple. As a special case, this is also true for one-way weighted automata.*

Proof. Consider a one-way pebble weighted automaton $\mathcal{A} = (Q, A, D, I, \Delta, F)$ and its semantics $[\![\mathcal{A}]\!]_K$ with $K \geq 0$: in the following, we only consider K-configurations. Given an ordered pointed graph G, we define a total order on the set of configurations of $\mathcal{A}$ over G, so that runs of $\mathcal{A}$ over G form an increasing sequence of configurations.

To do so, we consider the *measure* of a K-configuration (G, σ, q, π, v) (with $\pi \in (\text{Var} \times V)^k$, $k \leq K$) to be the word over the alphabet $V \uplus \{\perp, \top\}$ defined by[2]

$$\text{meas}(G, \sigma, q, \pi, v) = \begin{cases} \pi_{|V} \cdot v \cdot \perp & \text{if } \exists q' \in Q, \alpha \in \text{Test}, x \in \text{Var} \\ & \quad \text{such that } \Delta_{q,q'}(\mathsf{drop}_x)(\alpha) \neq 0 \\ \pi_{|V} \cdot v \cdot \top & \text{otherwise.} \end{cases}$$

We order the set $V \uplus \{\perp, \top\}$ by the order $\leq$ given by the ordered graph, and by letting $\perp < v < \top$ for every vertex $v \in V$. Henceforth, finite sequences over this set are ordered with the associated lexicographic (total) order, denoted by $\prec$.

We now show that for every accepting K-run $\rho = (G, \sigma, q_m, \pi_m, v_m)_{0 \leq m \leq h}$ of $\mathcal{A}$, the sequence of measures of the configurations is increasing, i.e.,

$$\text{meas}(G, \sigma, q_m, \pi_m, v_m) \prec \text{meas}(G, \sigma, q_{m+1}, \pi_{m+1}, v_{m+1})$$

for all $0 \leq m < h$. This ensures that the run is simple (and also that it is of bounded length as the set of K-configurations is finite). Let $0 \leq m < h$. The concrete transition from configuration $\gamma_m = (G, \sigma, q_m, \pi_m, v_m)$ to configuration $\gamma_{m+1} = (G, \sigma, q_{m+1}, \pi_{m+1}, v_{m+1})$ has non-zero weight (as the run ρ is accepting). In particular,

- If $\pi_m = \pi_{m+1}$, then the weight of the transition is

$$\bigoplus_{\substack{d \in D \mid (v_m, v_{m+1}) \in E_d \\ \alpha \in \text{Test} \mid G, \sigma_{\pi_m}, v_m \models \alpha}} \Delta_{q_m, q_{m+1}}(d)(\alpha)$$

  Using the first condition in the definition of one-way pebble weighted automata, we know that this weight is non-zero only if one of the directions is a forward direction $d \in D_{\rightarrow}$. By definition of ordered graphs, this ensures that $v_m < v_{m+1}$. Therefore, for some $\alpha, \beta \in \{\top, \perp\}$:

$$\text{meas}(\gamma_m) = (\pi_m)_{|V} \cdot v_m \cdot \alpha \prec (\pi_m)_{|V} \cdot v_{m+1} \cdot \beta = \text{meas}(\gamma_{m+1}) \,.$$

[2] In this proof, for $\pi \in (\text{Var} \times V)^k$, we denote by $\pi_{|V}$ the word in V^k obtained from π by erasing the name of the variables in Var.

- If $\pi_{m+1} = \pi_m(x, v_m)$ and $v_{m+1} = v^{(i)}$ (the initial vertex), then the non-zero weight of the transition is
$$\bigoplus_{\alpha \in \mathrm{Test} | G, \sigma_{\pi_m}, v_m \models \alpha} \Delta_{q_m, q_{m+1}}(\mathsf{drop}_x)(\alpha)$$
Again, this implies that for some $\alpha \in \{\bot, \top\}$ we have
$$\mathrm{meas}(\gamma_m) = (\pi_m)_{|V} \cdot v_m \cdot \bot \prec (\pi_m)_{|V} \cdot v_m \cdot v^{(i)} \cdot \alpha = \mathrm{meas}(\gamma_{m+1})\,.$$
- If $\pi_m = \pi_{m+1}(y, v_{m+1})$ for some $y \in \mathrm{Var}$, then the non-zero weight is
$$\bigoplus_{\alpha \in \mathrm{Test} | G, \sigma_{\pi_m}, v_m \models \alpha} \Delta_{q_m, q_{m+1}}(\mathsf{lift})(\alpha)$$
Hence, for some $\alpha \in \{\bot, \top\}$ we have
$$\mathrm{meas}(\gamma_m) = (\pi_{m+1})_{|V} \cdot v_{m+1} \cdot v_m \cdot \alpha \prec (\pi_{m+1})_{|V} \cdot v_{m+1} \cdot \top = \mathrm{meas}(\gamma_{m+1})\,.$$
Notice that the $\top$ in $\mathrm{meas}(\gamma_{m+1})$ comes from the fact that lift transitions may not be followed by drop transitions, by definition of 1PWA.

In all cases, we have $\mathrm{meas}(\gamma_m) \prec \mathrm{meas}(\gamma_{m+1})$, which concludes the proof. □

Alternatively, we may define the semantics of one-way (pebble) weighted automata as the simple semantics defined in the previous section. The previous lemma shows that this indeed define the same semantics.

7.3 Undecidability of Emptiness

Recall from Theorem 6.1 that the emptiness problem is decidable for layered pebble weighted automata over continuous semirings with no zero divisors. The fact that the semantics of one-way pebble weighted automata may be defined over non-continuous semirings makes the emptiness problem become undecidable.

Theorem 7.7. *The emptiness problem of threshold languages of series recognizable by one-way pebble weighted automata is undecidable. This is already the case when automata have 2 pebbles names, 2 layers and weights in the semiring* $(\mathbb{Z}, +, \times, 0, 1)$ *(in particular, a semiring without zero divisor).*

Proof. We reduce the emptiness problem of 2-counter machines. A *2-counter machine* $\mathcal{M}$ is a tuple $(\mathit{Loc}, \Delta, \mathit{init}, F)$ with Loc a finite set of locations, $\Delta = \mathit{Loc} \times \mathit{Instr} \times \mathit{Loc}$ the set of transitions where $\mathit{Instr} = \{\mathit{Inc}^p, \mathit{Dec}^p, \mathit{ZTest}^p \mid p \in \{1,2\}\}$, $\mathit{init} \in \mathit{Loc}$ the initial location and $F \subseteq \mathit{Loc}$ the set of final locations. If $w = (\ell_0, D_1, \ell_1) \cdots (\ell_{n-1}, D_n, \ell_n) \in \Delta^\star$ and $D \in \mathit{Instr}$, we set $|w|_D = |\{k \in [1 \mathinner{..} n] \mid D_k = D\}|$, and $w_j = (\ell_0, D_1, \ell_1) \cdots (\ell_{j-1}, D_j, \ell_j)$ for $1 \leq j \leq n$. We define the value of counter $p \in \{1,2\}$ after step $j \in \{1, \dots, n\}$ as $c^p_j(w) = |w_j|_{\mathit{Inc}^p} - |w_j|_{\mathit{Dec}^p}$, and we also set $c^p_0 = 0$.

An *accepting run* of $\mathcal{M}$ may be seen as a graph $G \in \mathcal{W}\mathrm{ord}(\Delta)$, representing a word $w = (\ell_0, D_1, \ell_1) \cdots (\ell_{n-1}, D_n, \ell_n) \in \Delta^+$ such that:

1. $\ell_0 = \mathit{init}$ and $\ell_n \in F$,
2. $c^p_j(w) \geq 0$ for all $j \in \{1, \dots, n\}$ and $p \in \{1,2\}$,
3. if $D_j = \mathit{ZTest}^p$, then $c^p_{j-1}(w) = 0$ for all $j \in \{1, \dots, n\}$ and $p \in \{1,2\}$.

We also denote $c^p_j(G)$ the value $c^p_j(w)$ in the following. The emptiness problem for 2-counter machines consists in deciding whether a given 2-counter machine has an accepting run. It is well known to be undecidable [Min67]. This problem reduces to emptiness of one-way weighted automata with 2 pebble names and 2 layers over $(\mathbb{Z}, +, \times, 0, 1)$, as we show now.

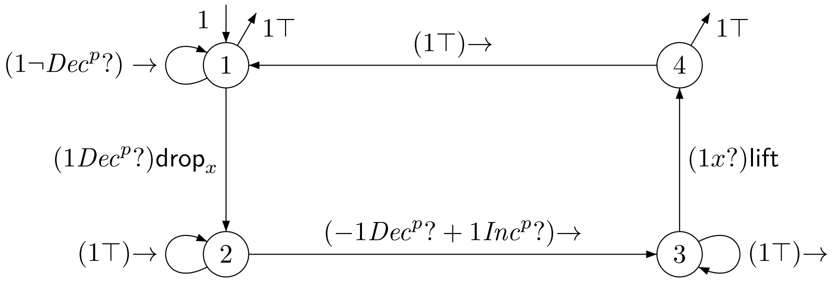


Figure 7.2: Automaton checking that the counter p is non-negative

From a 2-counter machine $\mathcal{M}$, we build such an automaton $\mathcal{A}$ assigning a nonzero weight to accepting runs of $\mathcal{M}$, and weight 0 to all other words. Hence, $\mathcal{A}$ has a nonzero semantics if, and only if, $\mathcal{M}$ has an accepting run.

A graph $G \in \mathcal{W}\mathrm{ord}(\Delta)$, representing a word $(\ell_0, D_1, \ell_1) \cdots (\ell_{n-1}, D_n, \ell_n)$, is not an accepting run of $\mathcal{M}$ if, and only if, it does not consist of consecutive transitions, or it violates either 1, 2 or 3. Let $\mathcal{D}^p = \{j \in V \mid D_j = Dec^p\}$ and $\mathcal{Z}^p = \{j \in V \mid D_j = ZTest^p\}$. Then G violates 2 if, and only if, for some $p \in \{1, 2\}$, there exists $j \in \mathcal{D}^p$ such that $c^p_{j-1}(G) = 0$, i.e.,

$$\prod_{j \in \mathcal{D}^p} c^p_{j-1}(G) = 0\,. \tag{7.1}$$

The value computed to check that the counter always stays non-negative can also be computed with the 1-layered one-way pebble weighted automaton of Figure 7.2: this automaton drops a pebble on each letter Dec^p and then computes the difference between the number of Inc^p symbols and the number of Dec^p symbols before the pebble, using a non deterministic choice to mimic the sum. We now prove that this automaton computes the weight in (7.1). We fix a graph $G \in \mathcal{W}\mathrm{ord}(\Delta)$ representing a word $w = (\ell_0, D_1, \ell_1) \cdots (\ell_{n-1}, D_n, \ell_n)$, and denote by $i_1 < i_2 < \cdots < i_m$ the vertices of G such that $D_i = Dec^p$. Notice that each accepting run of the automaton drops pebble x on each vertex i_k (for every $1 \leq k \leq m$), and, after each such drop, must choose a vertex j_k to follow the transition from state 2 to state 3: moreover, we have $1 \leq j_k < i_k$ and $D_{j_k} \in \{Inc^p, Dec^p\}$. Denoting J_k the set of such vertices j_k for every k, the set of accepting runs is therefore in bijection with the set $J = J_1 \times \cdots \times J_m$. The weight of the run defined by the tuple $(j_k)_{1 \leq k \leq m}$ is then given by

$$\prod_{k=1}^{m} f(j_k)$$

where $f(j) = 1$ if $D_j = Inc^p$, -1 if $D_j = Dec^p$, and 0 otherwise. Hence, the semantics of the automaton over G is given by

$$\begin{aligned}
\sum_{(j_k)_{1\leq k \leq m} \in J} \prod_{k=1}^{m} f(j_k) &= \sum_{j_1 \in J_1} \sum_{j_2 \in J_2} \cdots \sum_{j_m \in J_m} \prod_{k=1}^{m} f(j_k) \\
&= \prod_{k=1}^{m} \sum_{j_k \in J_k} f(j_k) \qquad \text{(by distributivity of } \times \text{ over } +) \\
&= \prod_{k=1}^{m} c^p_{i_k - 1}(G)
\end{aligned}$$

Next, if 2 holds, w violates 3 if, and only if, for some $p \in \{1, 2\}$, there exists $j \in \mathcal{Z}^p$ such that $c^p_{j-1}(G) > 0$, that is, if, and only if,

$$\prod_{j \in \mathcal{Z}^p} \prod_{1 \leq k \leq j-1} (c^p_{j-1}(G) - k) = 0 \tag{7.2}$$

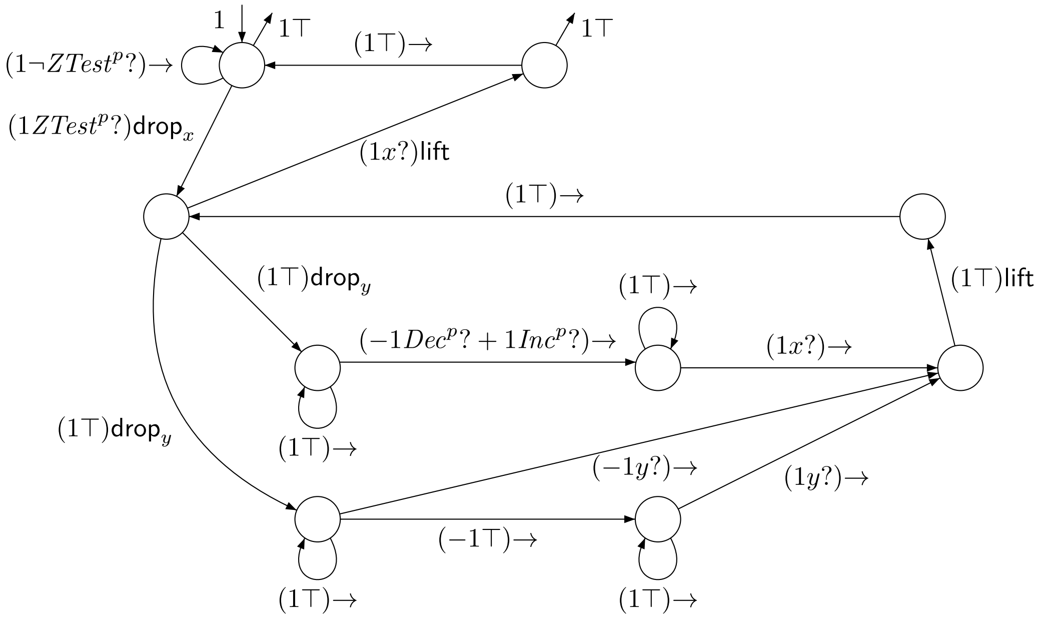


Figure 7.3: Automaton checking the zero tests $ZTest^p$

since $c^p_{j-1}(G) \in [0 \mathinner{..} j-1]$. This weight can be computed with the 2-layered one-way pebble weighted automaton of Figure 7.3. Intuitively, it drops pebble x sequentially on every position performing action $ZTest^p$ to compute the external product. When pebble x is dropped (over a vertex j), on every vertex k before the one holding x (i.e., $k \leq j-1$), it drops a pebble y choosing non-deterministically either to compute $c^p_{j-1}(G)$ (the upper part) or $-k$ (the lower part). The proof of correctness is done similarly to the previous automaton, and strongly relies on the distributivity of the multiplication over the addition again.

Furthermore, automaton $\mathcal{A}$ has weight 0 transitions to check if 1 is violated, or if two consecutive letters of Δ are not consecutive transitions in $\mathcal{M}$. Further, on each position $j \in \mathcal{D}^1 \cup \mathcal{D}^2 \cup \mathcal{Z}^1 \cup \mathcal{Z}^2$, automaton $\mathcal{A}$ drops a pebble in order to compute the products (7.1) or (7.2) as explained just before. □

It is easy to construct a formula of wFO(AP) equivalent to automaton $\mathcal{A}$, showing that emptiness is also undecidable for wFO(AP). Indeed, the weight of (7.1) may be computed by[3]

$$\bigotimes_x \Big[P_{Dec^p}(x)?\Big(\bigoplus_y \big[y < x \otimes \big(P_{Inc^p}(y) \oplus (P_{Dec^p}(y) \otimes (-1))\big)\big]\Big)\Big]$$

whereas the one of (7.2) may be computed by

$$\begin{aligned}&\bigotimes_x \Big[P_{ZTest^p}(x)?\Big(\bigotimes_z z < x? \\ &\quad \Big\{\bigoplus_y \big[(-1 \otimes y \leq z) \oplus \big(y < x \otimes \big(P_{Inc^p}(y) \oplus (-1 \otimes P_{Dec^p}(y))\big)\big)\big]\Big\}\Big)\Big]\end{aligned}$$

An anecdotic, yet intriguing, question concerns the emptiness problem of threshold language of automata with one or zero layers. We do not know the decision status of these problems, and let them as open questions.

[3] We recall that if φ is a Boolean formula and Φ a weighted formula, $\varphi?\,\Phi$ denotes the formula $(\varphi \otimes \Phi) \oplus (\neg\varphi)$. Intuitively, it computes Φ only when φ holds.

7.4 One-way versus Simple over Words

We show in this section that every pebble weighted automaton equipped with the simple semantics can be effectively translated into an equivalent (over words) one-way pebble weighted automaton, with the same number of layers. Notice that the simple assumption is necessary to state such a result, as Proposition 7.3 exhibits a weighted automaton that cannot be recognized by a simple weighted automaton.

In all the rest of this section, we let $D = \{\rightarrow, \leftarrow\}$. The only hypothesis required over the semiring is its commutativity. We suppose at the beginning that the pebble weighted automaton is K-strongly layered, but Proposition 4.29 and Lemma 4.27 show that this is not a restriction indeed.

Theorem 7.8. *Let $\mathbb{S}$ be a commutative semiring. For every K-strongly layered pebble weighted automaton $\mathcal{A}$ with the simple semantics over words, there exists an equivalent K-strongly layered one-way pebble weighted automaton $\mathcal{B}$ with the same set of free variables, i.e., for every non-empty finite word $w \in A^+$, and every valuation σ, we have $[\![\mathcal{A}]\!]_{\mathrm{S}}(w, \sigma) = [\![\mathcal{B}]\!](w, \sigma)$.*

The proof is by induction on the number of layers. We start with the base case of automata without pebbles. The next proposition is a generalization in the weighted setting of the result, originally proved in [RS59], stating that two-way feature does not add expressive power to non-deterministic finite automata over finite strings. There are mainly two proof techniques for this result. Most textbooks presenting this result, or article elaborating over it, use the proof of [She59], which starts with a two-way automaton and construct an equivalent one-way automaton by enriching the state with a relation table recording for every pair of states (q, q'), whether it is possible to reach q' from q with a loop on the current prefix. The key argument is that there is a finite number of such tables and that it can be computed simultaneously to the main run.

Observe that this method is not applicable in our weighted setting. Indeed, the relation table should now be enriched with the precomputed weights of the subruns, which cannot be recorded within a finite memory (as the weights may grow with the length of the prefix read so far). Instead, we will use the crossing-sequence method which is closer to the proof technique in [RS59], and fully explained in [HU79]. For the sake of clarity, we provide a complete proof below in our weighted setting. In [Ans90], a similar proof was given for weighted automata without pebbles, but we discovered it only after the publication of our article [1]. However, it has to be noticed that our proof is *a priori* stronger. Indeed, Anselmo considers only *syntactically simple* weighted automaton, i.e., automata that only generate simple accepting runs, whereas we consider the simplicity as a *semantical restriction.*

Proposition 7.9. *Let $\mathbb{S}$ be a commutative semiring. For every weighted automaton $\mathcal{A}$ with the simple semantics over words, there exists an equivalent one-way weighted automaton $\mathcal{B}$, i.e., for every word $w \in A^+$ and every valuation σ, we have $[\![\mathcal{A}]\!]_{\mathrm{S}}(w, \sigma) = [\![\mathcal{B}]\!](w, \sigma)$.*

Proof. Let $\mathcal{A} = (R, A, D, I, \Delta, F)$ be a weighted automaton. We construct an equivalent one-way weighted automaton $\mathcal{B} = (Q', A, D, I', \Delta', F')$ such that $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]_{\mathrm{s}}$ over the set of words A^+ (represented as graphs in $\mathcal{W}\mathrm{ord}(A)$).

Intuitively, the idea is to record in a state of $\mathcal{B}$ the *crossing-sequence* of states of $\mathcal{A}$ which is observed in some accepting run ρ while scanning the current position of the input word, see Figure 7.4. Since the simple semantics of a weighted automaton is computed as the sum over *simple* accepting runs, a crossing-sequence consists of pairwise distinct states and its length is therefore bounded by the number of states of $\mathcal{A}$. Finally, the commutativity of the semiring allows us to compute the weight of an accepting run ρ along a corresponding run of $\mathcal{B}$.

In order to ease the matching of consecutive crossing-sequences of $\mathcal{A}$, we also record in a state of $\mathcal{B}$ the input letter from A of the current position, its type (as a subset of D),

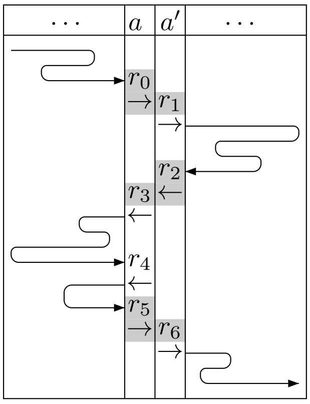


Figure 7.4: Run of a WA and some crossing-sequences

and the sequence of moves from D performed by $\mathcal{A}$. Therefore, we will define Q' as a *finite* subset of

$$T = A \times \mathfrak{P}(D) \times \mathrm{Free}(\mathcal{A}) \times (RD)^\star(R \cup R{\to})\,.$$

The first component records the letter of the current position, the second component its type, and the third the exact set of free variables over it. Those three components together are sufficient to decide whether a test formula $\alpha \in$ Test is verified or not over the current position. Indeed, if $(a, t, \mathcal{P}) \in A \times \mathfrak{P}(D) \times \mathrm{Free}(\mathcal{A})$, we define in Table 7.1 the satisfaction relation $(a, t, \mathcal{P}) \models \alpha$ by induction over α. We can easily verify by induction that for every

Table 7.1: An alternative definition of the semantics of tests in Test

$(a, t, \mathcal{P}) \models \top$
$(a, t, \mathcal{P}) \models$ init? if, and only if, $\leftarrow \notin t$
$(a, t, \mathcal{P}) \models$ final? if, and only if, $\rightarrow \notin t$
$(a, t, \mathcal{P}) \models b?$ if, and only if, $a = b$
$(a, t, \mathcal{P}) \models d?$ if, and only if, $d \in t$
$(a, t, \mathcal{P}) \models x?$ if, and only if, $x \in \mathcal{P}$
$(a, t, \mathcal{P}) \models \neg\alpha$ if, and only if, $(a, t, \mathcal{P}) \not\models \alpha$
$(a, t, \mathcal{P}) \models \alpha \wedge \alpha'$ if, and only if, $(a, t, \mathcal{P}) \models \alpha$ and $(a, t, \mathcal{P}) \models \alpha'$
$(a, t, \mathcal{P}) \models \alpha \vee \alpha'$ if, and only if, $(a, t, \mathcal{P}) \models \alpha$ or $(a, t, \mathcal{P}) \models \alpha'$

pointed graph $G \in \mathcal{W}\mathrm{ord}(A)$, valuation σ and vertex v,

$$G, \sigma, v \models \alpha \quad \text{if, and only if,} \quad (\lambda(v), \mathsf{type}(v), \sigma^{-1}(v)) \models \alpha$$

We say that a state in T is *R-simple* if the word in its $(RD)^\star(R \cup R{\to})$-component does not contain two occurrences of some state in R. For instance, Figure 7.4 exhibits two crossing-sequences which are

$$q = (a, D, \emptyset, r_0{\to}r_3{\leftarrow}r_4{\leftarrow}r_5{\to}) \quad \text{and} \quad q' = (a', D, \emptyset, r_1{\to}r_2{\leftarrow}r_6{\to})\,.$$

We describe now the transition matrix Δ' of $\mathcal{B}$. Two states $q = (a, t, \mathcal{P}, \tau)$ and $q' = (a', t', \mathcal{P}', \tau')$ in Q' can be *matched* in a transition of $\mathcal{B}$ if τ has one more right move than the number of left moves in τ': $|\tau|_\to = 1 + |\tau'|_\leftarrow$. The weight[4] $\Delta_{q,q'}(\to)(a? \wedge t? \wedge \mathcal{P}?)$ associated with this transition is the product of the weights of the transitions of $\mathcal{A}$ *contained* in the

[4] In the following, for $t \in \mathfrak{P}(D)$ (respectively, $\mathcal{P} \subseteq \mathrm{Free}(\mathcal{A})$), we use $t?$ (respectively, $\mathcal{P}?$) as a shortcut for $\bigwedge_{d \in t} d? \wedge \bigwedge_{d \notin t} \neg d?$ (respectively, $\bigwedge_{x \in \mathcal{P}} x? \wedge \bigwedge_{x \in \mathrm{Free}(\mathcal{A}) \setminus \mathcal{P}} \neg x?$).

pair (q, q'), i.e., right transitions from q to q' and left transitions from q' to q. On Figure 7.4, these transitions are on a gray background. If q and q' cannot be matched we let $\Delta_{q,q'}$ to be 0.

Formally, a triple $(r, \rightarrow, r')$ with $r, r' \in R$ is said to be *contained* in (q, q') if there exists $k \in [1 \mathinner{..} |q|_{\rightarrow}]$ such that (1) r is the state just before the kth occurrence of $\rightarrow$ in q, and (2) r' the state just after the $(k-1)$th occurrence[5] of $\leftarrow$ in q'. Similarly, a triple $(r', \leftarrow, r)$ with $r, r' \in R$ is said to be contained in (q, q') if there exists $k \in [1 \mathinner{..} |q'|_{\leftarrow}]$ such that (1) r' the state just before the kth occurrence of $\leftarrow$ in q', and (2) r is the state just after the kth occurrence of $\rightarrow$ in q. For instance in Figure 7.4, triples contained in the depicted pair of states are $(r_0, \rightarrow, r_1)$, $(r_5, \rightarrow, r_6)$, and $(r_2, \leftarrow, r_3)$.

We let $\mathsf{Tr}(q, q')$ be the set of triples contained in (q, q'). We then define the polynomial $\Delta'_{q,q'}(\rightarrow) \in \mathbb{S}\langle\text{Test}\rangle$ of $\rightarrow$ transitions of $\mathcal{B}$ induced by the matching states (q, q') as the monomial[6]:

$$\begin{aligned}\Delta'_{q,q'}(\rightarrow)(a? \wedge t? \wedge \mathcal{P}?) = & \bigotimes_{(r,\rightarrow,r') \in \mathsf{Tr}(q,q')} \ \bigoplus_{\substack{\alpha \in \text{Test} \\ (a,t,\mathcal{P}) \models \alpha}} \Delta_{r,r'}(\rightarrow)(\alpha) \\ \otimes & \bigotimes_{(r',\leftarrow,r) \in \mathsf{Tr}(q,q')} \ \bigoplus_{\substack{\alpha \in \text{Test} \\ (a',t',\mathcal{P}') \models \alpha}} \Delta_{r',r}(\leftarrow)(\alpha) \end{aligned} \tag{7.3}$$

where $q = (a, t, \mathcal{P}, \tau)$ and $q' = (a', t', \mathcal{P}', \tau')$.

Finally, the initial vector I' maps the R-simple states of the form

$$(a, \{\rightarrow\}, \mathcal{P}, r{\rightarrow}(R{\rightarrow})^\star) \qquad \text{or} \qquad (a, \emptyset, \mathcal{P}, r)$$

to I_r, and all other states to 0. The final vector F' must moreover check that the $(a, t, \mathcal{P})$-component of the final state is correct, so that it maps the R-simple states of the form

$$(a, \{\leftarrow\}, \mathcal{P}, (R{\leftarrow})^\star r) \qquad \text{or} \qquad (a, \emptyset, \mathcal{P}, r)$$

to the monomial of $\mathbb{S}\langle\text{Test}\rangle$ given by

$$F_r\ (a? \wedge t? \wedge \mathcal{P}?)$$

with $t = \{\leftarrow\}$ or $t = \emptyset$ respectively, and other states to 0.

Lemma 7.10 below proves that there is a weight preserving bijection between the simple accepting runs of $\mathcal{A}$ and the accepting runs of $\mathcal{B}$. We conclude that $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]_{\mathrm{s}}$ since for all $G \in \mathcal{W}\text{ord}(A)$ and valuation σ, $[\![\mathcal{A}]\!]_{\mathrm{s}}(G, \sigma)$ is the sum of the weights of the simple accepting runs of $\mathcal{A}$ over (G, σ), whereas $[\![\mathcal{B}]\!](G, \sigma)$ is the sum of the weights of the accepting runs of $\mathcal{B}$ over (G, σ). □

Lemma 7.10. *For every $G \in \mathcal{W}\text{ord}(A)$ and valuation σ, there is a weight preserving bijection between the simple accepting runs of $\mathcal{A}$ over (G, σ) and the accepting runs of $\mathcal{B}$ over (G, σ).*

Proof. Let $G \in \mathcal{W}\text{ord}(A)$ with n vertices be a graph denoting a word $w = a_0 \cdots a_{n-1}$, and a valuation σ. We first show how to map a simple accepting run $\rho = \rho_0\rho_1 \cdots \rho_m$ of $\mathcal{A}$ over (G, σ) (where each ρ_j is a configuration of the form (G, σ, q, v)), to an accepting run of $\mathcal{B}$ over (G, σ).

For $i \in [0 \mathinner{..} n-1]$, we construct the crossing-sequence $q_i = (a_i, \mathsf{type}(i), \sigma(i), \tau_i)$ of $\mathcal{B}$ associated with vertex i, as illustrated in Figure 7.4. Formally, we define τ_i as the "projection" of the configurations visiting vertex i, where configuration (G, σ, r, i) is "projected" to $r{\rightarrow}$

[5] In case $k = 1$, the state just after the 0th occurrence of $\leftarrow$ in q' is simply the first state of q'.

[6] All coefficients of tests distinct from $a? \wedge t? \wedge \mathcal{P}?$ are set to 0. Notice also that a product over an emptyset is equal to 1.

(respectively, $r{\leftarrow}$ or r) if it is followed by some $(G, \sigma, r', i+1)$ (respectively, $(G, \sigma, r', i-1)$ or is the last configuration). Note that the word τ_i is R-simple since ρ is a simple run, so that $q_i \in Q'$.

Observe also that $|\tau_i|_\rightarrow = 1 + |\tau_{i+1}|_\leftarrow$ for every $i \leq n-2$. Indeed, ρ must reach position $i+1$ for a first time, and then, each time the run ρ goes to the left of position $i+1$, it will in the future come back to $i+1$ from i. Therefore, q_i and q_{i+1} can be matched for all $i \leq n-2$. Moreover, the weight $\Delta'_{q_i,q_{i+1}}(\rightarrow)(a_i? \wedge \mathtt{type}(i)? \wedge \sigma(i)?)$ is not equal to zero, since for every triple $(r, d, r') \in \mathsf{Tr}(q_i, q_{i+1})$ there is at least one weight $\Delta_{r,r'}(d)(\alpha)$ of (7.3) which is different from 0 (the run ρ being accepting). We deduce that $f(\rho) = (G, \sigma, q_0, 0) \cdots (G, \sigma, q_{n-1}, n-1)$ is an accepting run of $\mathcal{B}$.

We now show that f preserves the weights, i.e., that the weight of ρ in $\mathcal{A}$ is equal to the weight of $f(\rho)$ in $\mathcal{B}$, for all simple run ρ of $\mathcal{A}$ over (G, σ). Indeed, the weight of ρ in $\mathcal{A}$ is the product of the weights of (left or right) transitions appearing in ρ. With $f(\rho) = (G, \sigma, q_0, 0) \cdots (G, \sigma, q_{n-1}, n-1)$, every right transition of ρ starting from vertex i is contained in the pair (q_i, q_{i+1}), so that its weight is computed in the transition from i to $i+1$ in $f(\rho)$, whereas every left transition of ρ starting from vertex i is contained in the pair (q_{i-1}, q_i), so that its weight is computed in the transition from $i-1$ to i in $f(\rho)$. We obtain the result by commutativity of the semiring $\mathbb{S}$. Moreover, initial and final weights are preserved by the transformation.

We finally prove that f is a bijection by constructing its inverse function. We consider an accepting run $(G, \sigma, q_0, 0) \cdots (G, \sigma, q_{n-1}, n-1)$ of $\mathcal{B}$. Write $q_i = (a_i, t_i, \mathcal{P}_i, \tau_i)$ for $i \in [0\,..\,n-1]$: necessarily, $t_i = \mathtt{type}(i)$ and $\mathcal{P}_i = \sigma(i)$. We recover a simple run ρ of $\mathcal{A}$ over (G, σ) as the output of $\mathrm{Run}(0, \tau_0, \ldots, \tau_{n-1})$ where the recursive function Run is defined by:

```
Function Run($i, \tau_0, \ldots, \tau_{n-1}$)
  match $\tau_i$ with
    $r{\rightarrow}\tau'_i$ :
      output $(G, \sigma, r, i)$;
      Run $(i+1, \tau_0, \ldots, \tau'_i, \ldots, \tau_{n-1})$;
    $r{\leftarrow}\tau'_i$ :
      output $(G, \sigma, r, i)$;
      Run $(i-1, \tau_0, \ldots, \tau'_i, \ldots, \tau_{n-1})$;
    $r$ : (* necessarily $i = n-1$ *)
      output $(G, \sigma, r, n-1)$;
```

We can show by induction that

$$f(\mathtt{Run}(0, \tau_0, \ldots, \tau_{n-1})) = (G, \sigma, q_0, 0) \cdots (G, \sigma, q_{n-1}, n-1)$$

Also, for every simple accepting run ρ of $\mathcal{A}$ with

$$f(\rho) = (G, \sigma, (a_0, t_0, \mathcal{P}_0, \tau_0), 0) \cdots (G, \sigma, (a_{n-1}, t_{n-1}, \mathcal{P}_{n-1}, \tau_{n-1}), n-1)$$

we have $\mathtt{Run}(0, \tau_0, \ldots, \tau_{n-1}) = \rho$. Therefore, f is a bijection. □

The proof of Proposition 7.9 may be easily adapted to various extensions of weighted automata. For instance, instead of restricting to *simple* runs when computing the semantics of a weighted automaton $\mathcal{A}$, we may allow k-simple runs (for some fixed k) in which no configuration is visited more than k times. Even when $k = 2$, we do not know an easy way to construct a weighted automaton $\mathcal{A}'$ whose semantics over 1-simple runs coincide with the semantics of $\mathcal{A}$ over 2-simple runs. But the proof of Proposition 7.9 allows to cope with this extension simply by allowing k-simple words when defining the set Q' of states of $\mathcal{B}$. Hence, we can construct a one-way weighted automaton $\mathcal{B}$ whose semantics coincides with the semantics of $\mathcal{A}$ over k-simple runs, showing that this extension does not add expressive power.

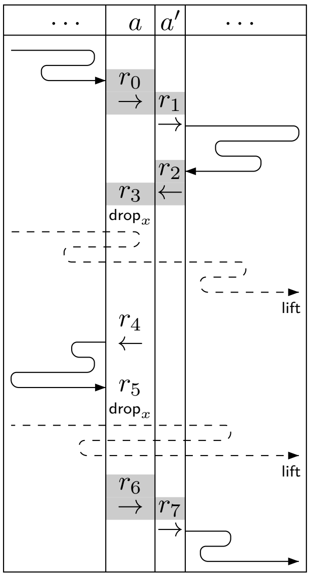


Figure 7.5: Run of a pebble weighted automaton and some crossing-sequences

We introduce another extension which will be useful in the proof of Theorem 7.8 below. We generalize the initial and final vectors of a weighted automaton $\mathcal{A} = (R, A, D, I, \Delta, F)$ by letting I (respectively, F) be a mapping from simple sequences of R^+ to $\mathbb{S}$ (respectively, $\mathbb{S}\langle\text{Test}\rangle$). The weight of a run is adapted by considering as initial weight the one computed by I over the sequence of states visiting the initial position, and the final weight computed by verifying the test formula on the last configuration of the run as usual. Notice that this is indeed a generalization as every weighted automaton $\mathcal{A} = (R, A, D, I, \Delta, F)$ can be encoded into this new formalism by letting I (respectively, F) map all simple sequences of states starting with a state q (respectively, ending with a state q) to the original initial weight (respectively, final polynomial) of this state. Again, the proof of Proposition 7.9 allows to cope with this extension easily.

We extend now the translation of Proposition 7.9 to pebble weighted automata. The general idea is an induction on the number of layers, but it has to be done with great care so that all and only the *simple* runs of the weighted automaton are taken into account.

Proof of Theorem 7.8. We start with a K-strongly layered pebble weighted automaton $\mathcal{A} = (R, A, D, I, \Delta, F)$, and a generalized acceptance condition: I (respectively, F) is mapping of simple sequences in R^+ to $\mathbb{S}$ (respectively, to $\mathbb{S}\langle\text{Test}\rangle$). The semantics is computed as previously by appending to the weight of the runs the initial weight of the sequence of states visiting the initial position *while no pebbles are dropped*, and the final weight of the sequence of states visiting the final position *while no pebbles are dropped* with the test formula verified on the last configuration of the run. This generalization will be useful in the inductive step of the construction.

If $K = 0$, Proposition 7.9 permits to conclude, as noticed previously. We now explain the construction for $K > 0$. By the strongly layered hypothesis, we know that the states in layer K of the automaton may only drop pebbles of a certain name, that we denote by x in the following. In particular, a crossing-sequence may now also include drop_x moves, see Figure 7.5. The parts of a run having at least one pebble dropped (dashed in Figure 7.5) will be deferred to the inductive step. Hence, a crossing-sequence consists only of the states visiting some position when no pebbles are dropped. If the run is simple, then this sequence of states is also simple.

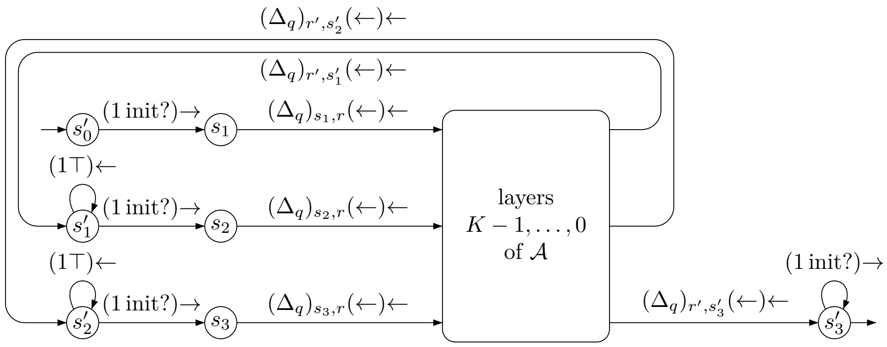

$(\Delta_q)_{r',s'_2}(\leftarrow)\leftarrow$
$(\Delta_q)_{r',s'_1}(\leftarrow)\leftarrow$
s'_0
$(1\,\text{init?})\rightarrow$
s_1
$(\Delta_q)_{s_1,r}(\leftarrow)\leftarrow$
$(1\top)\leftarrow$
s'_1
$(1\,\text{init?})\rightarrow$
s_2
$(\Delta_q)_{s_2,r}(\leftarrow)\leftarrow$
layers
$K-1,\dots,0$
of $\mathcal{A}$
$(1\top)\leftarrow$
s'_2
$(1\,\text{init?})\rightarrow$
s_3
$(\Delta_q)_{s_3,r}(\leftarrow)\leftarrow$
$(\Delta_q)_{r',s'_3}(\leftarrow)\leftarrow$
$(1\,\text{init?})\rightarrow$
s'_3

Figure 7.6: The pebble weighted automaton $\mathcal{A}_q$ filling the drop-lift gaps of the crossing-sequence q

As in the proof of Proposition 7.9, we let Q' be the (finite) set of *R-simple* states in $T = A \times \mathfrak{P}(D) \times \mathrm{Free}(\mathcal{A}) \times (R\{\leftarrow, \rightarrow, \mathsf{drop}_x\})^*(R \cup R\rightarrow)$. For instance, Figure 7.5 exhibits the crossing-sequence

$$q = (a, D, \emptyset, r_0{\rightarrow}r_3\mathsf{drop}_x r_4{\leftarrow}r_5\mathsf{drop}_x r_6{\rightarrow})$$

in which the states r_0, r_3, r_4, r_5, r_6 must be pairwise distinct.

For each state $q = (a, t, \mathcal{P}, \tau) \in Q'$, we construct a $(K-1)$-strongly layered pebble weighted automaton $\mathcal{A}_q = (R_q, A, D, I_q, \Delta_q, F_q)$ which computes the sum of the weights of the tuples of runs filling the drop-lift gaps in the crossing-sequence q, i.e., the dashed lines in Figure 7.5.

From a crossing-sequence $q = (a, t, \mathcal{P}, \tau) \in Q'$, we first extract the sequence $\theta(\tau)$ of pairs of states surrounding drop_x actions. For instance,

$$\begin{aligned} &\theta(r_0{\rightarrow}r_3\mathsf{drop}_x r_4{\leftarrow}r_5\mathsf{drop}_x r_6{\rightarrow}) = (r_3, r_4)(r_5, r_6) \\ \text{and} \quad &\theta(r_1\mathsf{drop}_x r_2\mathsf{drop}_x r_3{\rightarrow}) = (r_1, r_2)(r_2, r_3)\,. \end{aligned}$$

We assume below that $\theta(\tau) = (r_1, r'_1)(r_2, r'_2)\cdots(r_N, r'_N)$ with $N > 0$, and $\ell(r_i) = \ell(r'_i) = K$. Note that $N \leq |R|$ since $r_1, r_2, \dots, r_N$ must be pairwise distinct.

We let $R_q = \{r \in R \mid \ell(r) < K\} \uplus \{s'_0, s_1, s'_1, \dots, s_N, s'_N\}$ be the set of states of $\mathcal{A}_q$. The automaton $\mathcal{A}_q$ should fill the drop-lift gaps (r_j, r'_j) for all $j \in [1\,..\,N]$. To do so, it simply simulates $\mathcal{A}$ from state r_j to r'_j without jumping in layer K: here we use the fact that a pebble weighted automaton may test the presence of a pebble without even having dropped it. The additional states in $\{s'_0, s_1, s'_1, \dots, s_N, s'_N\}$ and transitions will make sure that all gaps (r_j, r'_j) are filled, see Figure 7.6. States s'_{j-1} are used to reset the head to the initial vertex between two simulations, and we move from s'_{j-1} to s_j at the beginning of the word[7]:

$$\begin{aligned} (\Delta_q)_{s'_{j-1}, s'_{j-1}}(\leftarrow)(\top) &= 1 && \text{for } 1 < j \leq N \\ (\Delta_q)_{s'_{j-1}, s_j}(\rightarrow)(\text{init?}) &= 1 && \text{for } 1 \leq j \leq N\,. \end{aligned}$$

A drop_x transition of $\mathcal{A}$ from state r_j, $j \in [1\,..\,N]$, to state r of layer $K-1$ is simulated in $\mathcal{A}_q$ by a $\leftarrow$ transition from s_j to r reading the second letter of the word (note that when $\mathcal{A}_q$ is in state s_j it must be on the second position of the word due to the $\rightarrow$ transition from s'_{j-1} to s_j): it outputs weight $(\Delta_q)_{s_j,r}(\leftarrow)(\top)$ defined by

$$(\Delta_q)_{s_j,r}(\leftarrow) = \bigoplus_{\substack{\alpha \in \mathrm{Test} \\ (a,t,\mathcal{P}) \models \alpha}} \Delta_{r_j,r}(\mathsf{drop}_x)(\alpha)\top\,. \tag{7.4}$$

[7] In case w is a single letter word, we may precompute the whole weight and compute it by using final weight (testing also that the current vertex is both initial and final). Henceforth, we suppose in the following that $w \notin A$.

A lift transition of $\mathcal{A}$ from a state r' of layer $K-1$ reaching r'_j, $j \in [1..N]$, is simulated in $\mathcal{A}_q$ by a $\leftarrow$ transition from r' to s'_j given by

$$(\Delta_q)_{r',s'_j}(\leftarrow) = \bigoplus_{a' \in A, \mathcal{P}' \subseteq \mathrm{Free}(\mathcal{A})} \; \bigoplus_{\substack{\alpha \in \mathrm{Test} \\ (a',\{\leftarrow\},\mathcal{P}') \models \alpha}} \Delta_{r',r'_j}(\mathsf{lift})(\alpha)(a'? \wedge \{\leftarrow\}? \wedge \mathcal{P}'?)\,.$$

Finally, when $\mathcal{A}_q$ reaches s'_N, it terminates with $\rightarrow$ moves until the final vertex of the word:

$$(\Delta_q)_{s'_N,s'_N}(\rightarrow)(\top) = 1$$

All other unspecified coefficients of the matrix Δ_q are set to 0.

We finally let I_q map *simple* sequences in $s'_0 R^\star s'_1 R^\star \cdots s'_{N-1} R^\star$ to 1 and others to 0, and we let F_q map *simple* sequences in $R^\star s'_N$ to $1\top$ and others to 0. Considering these definitions, an accepting run $\hat{\rho}$ of $\mathcal{A}_q$ can be split into a sequence of runs $\hat{\rho} = \rho'_0 \hat{\rho}_1 \rho'_1 \cdots \hat{\rho}_N \rho'_N$ where for each $1 \leq j \leq N$

- the run $\hat{\rho}_j$ fills the drop-lift gap (r_j, r'_j). It goes from s_j to s'_j and simulates a run ρ_j of $\mathcal{A}$ which starts with a drop_x transition from r_j to a state r of layer $K-1$, and ends with a lift transition from a state r' of layer $K-1$ to r'_j without lifting this pebble in-between. Moreover, runs $\hat{\rho}_j$ and ρ_j have the same weight, for all $1 \leq j \leq N$;
- the run ρ'_0 consists of a single transition from s'_0 to s_1, the run ρ'_j for $0 < j < N$ loops on state s'_j to reset the head to the initial position and then moves to state s_{j+1}, and the run ρ'_N consists of transitions looping on s'_N to reach the final position. Moreover, runs ρ'_j have weight 1, for all $0 \leq j \leq N$.

Note that, the run $\hat{\rho}$ is simple if, and only if, the sequence of runs $\rho_1, \ldots, \rho_N$ is *globally* simple, i.e., a configuration may not occur twice in some ρ_j and it may not occur both in ρ_i and ρ_j with $i \neq j$. This is important since the runs $\rho_1, \ldots, \rho_N$ fill the drop-lift gaps of the crossing-sequence $(a, t, \mathcal{P}, \tau)$ and are therefore part of a single run of $\mathcal{A}$. This explains why we have a single copy of $\mathcal{A}$ in $\mathcal{A}_q$. If instead we used N copies of $\mathcal{A}$ inside $\mathcal{A}_q$, one for each drop-lift gap (r_j, r'_j) then the simplicity of $\hat{\rho}$ would not imply the *global* simplicity[8] of $\rho_1, \ldots, \rho_N$.

Clearly, all accepting runs of $\mathcal{A}_q$ start from state s'_0 and end in state s'_N. But this basic initial and final condition is not sufficient to ensure that the drop-lift gaps are filled correctly. For instance, $\mathcal{A}_q$ could go directly from s'_0 to s'_N without ever visiting $s'_1, \ldots, s'_{N-1}$. It could also visit all the new states but not in the intended order, which would not correspond to a sequence of runs of $\mathcal{A}$ filling the gaps $(r_1, r'_1) \cdots (r_N, r'_N)$. This is why we use the generalized acceptance condition for I_q.

Towards a uniform construction below, we also define an automaton $\mathcal{A}_q$ when the crossing-sequence q has no drop_x ($N = 0$). In this case, $\mathcal{A}_q$ has a single state s'_0 with a loop moving right with weight 1 in order to simply scan the word from beginning to end.

By induction, for each $q \in Q'$ we can construct a $(K-1)$-strongly layered one-way pebble weighted automaton $\mathcal{B}_q$ which is equivalent to $\mathcal{A}_q$, more precisely, such that for each graph $G \in \mathcal{W}\mathrm{ord}(A)$ and each valuation σ, there is a weight preserving bijection between the simple accepting runs of $\mathcal{A}_q$ over (G, σ) and the accepting runs of $\mathcal{B}_q$ over (G, σ). We let $\mathcal{B}_q = (Q'_q, A, D, I'_q, \Delta'_q, F'_q)$. Note that, if the crossing-sequence q has no drop then we may choose $\mathcal{B}_q = \mathcal{A}_q$.

We define now the K-strongly layered one-way pebble weighted automaton $\mathcal{B} = (Q, A, D, I', \Delta', F')$ associated with $\mathcal{A}$. Its set of states is

$$Q = Q' \uplus \overline{Q'} \uplus \biguplus_{q \in Q'} Q'_q\,.$$

[8] Alternatively, we could define an equivalence relation $\sim$ on states of $\mathcal{A}_q$ and use a $\sim$-simple semantics of runs, as originally done in [1].

The initial mapping I' maps the states in Q' of the form

$$(a, \{\rightarrow\}, \mathcal{P}, (R\{\rightarrow, \mathsf{drop}_x\})^\star R\rightarrow) \quad \text{or} \quad (a, \emptyset, \mathcal{P}, (R\mathsf{drop}_x)^\star R)$$

to the weight given by I over their projections on R. Similarly, the final mapping F' maps the states in $\overline{q'} \in \overline{Q'}$ with q' of the form

$$(a, \{\leftarrow\}, \mathcal{P}, (R\{\leftarrow, \mathsf{drop}_x\})^\star R) \quad \text{or} \quad (a, \emptyset, \mathcal{P}, (R\mathsf{drop}_x)^\star R)$$

to the polynomial given by F over their projections on R with the test formulae enriched with $a? \wedge t? \wedge \mathcal{P}?$, where $t = \{\leftarrow\}$ or $t = \emptyset$ respectively.

The automaton $\mathcal{B}$ contains a copy of each $\mathcal{B}_q$ in order to simulate the parts of a run when the first pebble x is dropped (dashed lines in Figure 7.5).

From each state $q = (a, t, \mathcal{P}, \tau) \in Q'$, we first call (the copy of) the automaton $\mathcal{B}_q$ dropping pebble x and when the computation of $\mathcal{B}_q$ is done we lift the pebble returning to the copy $\bar{q} \in \overline{Q'}$ of $q \in Q'$: for every $s, s' \in Q'_q$,

$$\Delta'_{q,s}(\mathsf{drop}_x)(\top) = (I'_q)_s \quad \text{and} \quad \Delta'_{s',\bar{q}}(\mathsf{lift}) = (F'_q)_{s'} \,.$$

Then, we perform a → action to the next crossing-sequence $q' \in Q'$. As in the proof of Proposition 7.9, two crossing-sequences $q = (a, t, \mathcal{P}, \tau)$ and $q' = (a', t', \mathcal{P}', \tau')$ of Q' match if $|\tau|_\rightarrow = 1 + |\tau'|_\leftarrow$. We assume $\mathsf{Tr}(q, q')$ to be defined as in the proof of Proposition 7.9, so that we may define the weight of → moves of $\mathcal{B}$ induced by the matching crossing-sequences (q, q') as the product of the weights of the (left or right) transitions of $\mathcal{A}$ contained in (q, q'):

$$\begin{aligned}\Delta'_{\bar{q},q'}(\rightarrow)(a? \wedge t? \wedge \mathcal{P}?) = & \bigotimes_{(r,\rightarrow,r')\in\mathsf{Tr}(q,q')} \ \bigoplus_{\substack{\alpha\in\text{Test}\\(a,t,\mathcal{P})\models\alpha}} \Delta_{r,r'}(\rightarrow)(\alpha)\\ \otimes & \bigotimes_{(r',\leftarrow,r)\in\mathsf{Tr}(q,q')} \ \bigoplus_{\substack{\alpha\in\text{Test}\\(a',t',\mathcal{P}')\models\alpha}} \Delta_{r',r}(\leftarrow)(\alpha)\end{aligned}$$

Lemma 7.11 below proves that there is a weight preserving bijection between the simple accepting runs of $\mathcal{A}$ and the accepting runs of $\mathcal{B}$. We conclude that $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]_\mathrm{s}$ since for all $G \in \mathcal{W}\mathrm{ord}(A)$ and all valuation σ, $[\![\mathcal{A}]\!]_\mathrm{s}(G, \sigma)$ is the sum of the weights of the simple accepting runs of $\mathcal{A}$ over (G, σ), whereas $[\![\mathcal{B}]\!](G, \sigma)$ is the sum of the weights of the accepting runs of $\mathcal{B}$ over (G, σ). □

Lemma 7.11. *For every $G \in \mathcal{W}\mathrm{ord}(A)$ and valuation σ, there is a weight preserving bijection between the simple accepting runs of $\mathcal{A}$ over (G, σ) and the accepting runs of $\mathcal{B}$ over (G, σ).*

Proof. Let $G \in \mathcal{W}\mathrm{ord}(A)$ with n vertices denoting a word $w = a_0 \cdots a_{n-1}$, and a valuation σ. We will decompose the bijection into several ones, all preserving the weights in an adequate manner.

To extend the proof of Lemma 7.10, we first associate with a simple accepting run ρ of $\mathcal{A}$ over (G, σ) a tuple

$$f(\rho) = (q_0, g_0, q_1, g_1, \ldots, q_i, g_i, \ldots, q_{n-1}, g_{n-1})$$

where $q_i = (\lambda(i), \mathsf{type}(i), \sigma(i), \tau_i)$ is the crossing-sequence at vertex $i \in [0 \mathinner{..} n-1]$ and the tuple $g_i = (\rho^i_1, \ldots, \rho^i_{N_i})$ consists of the sequence of subruns of ρ with pebble x dropped on vertex i, together with the surrounding drop-lift transitions.

Formally, for $0 \leq i \leq n-1$, to define the crossing sequence q_i, we let τ_i be the "projection" of the configurations visiting vertex i with an empty stack of pebbles, where configuration $(G, \sigma, r, \varepsilon, i)$ is "projected" to $r \rightarrow$ (respectively, $r \leftarrow$, $r\mathsf{drop}_x$ or r) if it is followed by some $(G, \sigma, r', \varepsilon, i{+}1)$ (respectively, $(G, \sigma, r', \varepsilon, i{-}1)$, $(G, \sigma, r', (x, i), 0)$ or is the last configuration).

Moreover, for $0 \leq i \leq n-1$ with $N_i = |\tau_i|_{\mathsf{drop}_x}$, we extract from ρ the tuple $g_i = (\rho^i_1, \dots, \rho^i_{N_i})$ of subruns of the form

$$(G, \sigma, r_0, \varepsilon, i)(G, \sigma, r_1, \pi_1, 0) \cdots (G, \sigma, r_{k-1}, \pi_{k-1}, n-1)(G, \sigma, r_k, \varepsilon, i)$$

where $\pi_\ell = (x, i) \cdot \pi'_\ell$ for all $0 < \ell < k$, i.e., where the first pebble is continuously dropped on position i.

We first show that f is a bijection from simple accepting runs of $\mathcal{A}$ over (G, σ) to the set, denoted $\mathrm{Seq}_{G,\sigma}$, of sequences

$$(q_0, g_0, q_1, g_1, \dots, q_i, g_i, \dots, q_{n-1}, g_{n-1})$$

verifying:

- $q_i \in Q'$ for all $i \in [0 \mathinner{..} n-1]$, $I'_{q_0} \neq 0$, and $F'_{\overline{q_{n-1}}} \neq 0$;
- for all $i \in [0 \mathinner{..} n-2]$, the pair (q_i, q_{i+1}) matches, i.e., $|\tau_i|_{\rightarrow} = 1 + |\tau_{i+1}|_{\leftarrow}$;
- for all $i \in [0 \mathinner{..} n-1]$, denoting $N_i = |\tau_i|_{\mathsf{drop}_x}$, g_i is a tuple $(\rho^i_1, \dots, \rho^i_{N_i})$ of subruns of $\mathcal{A}$, globally simple, each starting with a drop_x on position i, ending with the first time this pebble is lifted, and that *matches* with $\theta(q_i)$ (i.e., the jth pair in $\theta(q_i)$ contains exactly the pair of first and last states of the run ρ^i_j).

It is not difficult to check that $f(\rho) \in \mathrm{Seq}_{G,\sigma}$ whenever ρ is a simple accepting run of $\mathcal{A}$ over (G, σ). We define the weight of such a sequence in $\mathrm{Seq}_{G,\sigma}$ as

$$I'_{q_0} \otimes \Big(\bigotimes_{i=0}^{n-2} \big[\Delta(g_i) \otimes \Delta'_{\overline{q_i}, q_{i+1}}(\rightarrow)(a_i? \wedge t_i? \wedge \mathcal{P}_i?) \big] \Big) \otimes \Delta(g_n) \otimes F'_{\overline{q_{n-1}}}$$

where $\Delta(g_i)$ is the product of the weights of runs ρ^i_j for $j \in [1 \mathinner{..} N_i]$ if $g_i = (\rho^i_1, \dots, \rho^i_{N_i})$. We can easily show that f preserves the weights.

We finally prove that f is a bijection by constructing its inverse function. For every $(q_0, g_0, q_1, g_1, \dots, q_{n-1}, g_{n-1}) \in \mathrm{Seq}_{G,\sigma}$, consider $q_i = (a_i, t_i, \mathcal{P}_i, \tau_i)$ for $i \in [0 \mathinner{..} n-1]$. We recover a simple accepting run ρ of $\mathcal{A}$ over (G, σ) as the output of $\mathrm{RunP}(0, \tau_0, g_0, \dots, \tau_{n-1}, g_{n-1})$ where the recursive function RunP is defined by:

```
Function RunP($(i, \tau_0, g_0, \dots, \tau_{n-1}, g_{n-1})$)
match $\tau_i$ with
    $r{\rightarrow}\tau'_i$ :
        output $(G, \sigma, r, \varepsilon, i)$;
        RunP $(i+1, (\dots, \tau'_i, g_i, \dots))$;
    $r{\leftarrow}\tau'_i$ :
        output $(G, \sigma, r, \varepsilon, i)$;
        RunP $(i-1, (\dots, \tau'_i, g_i, \dots))$;
    $r\mathsf{drop}_x\tau'_i$ :
        match $g_i$ with
            $(\rho', g'_i)$ :
                output $\rho'$;
                RunP $(i, (\dots, \tau'_i, g'_i, \dots))$;
    $r$ : (* necessarily $i = n-1$ *)
        output $(G, \sigma, r, \varepsilon, n-1)$;
```

We then consider each tuple g_i of runs of $\mathcal{A}$ with the first pebble x dropped. It has already been explained in the previous proof how to construct a weight preserving bijection between a tuple $g_i = (\rho^i_1, \dots, \rho^i_N)$, which matches with a sequence of pairs $\theta(q_i)$, and an accepting run $\hat{\rho}_i = \rho'_0 \hat{\rho}^i_1 \rho'_1 \cdots \hat{\rho}^i_N \rho'_N$ of $\mathcal{A}_{q_i}$ over $(G, \sigma[x \mapsto i])$ so that g_i is globally simple if, and only if, $\hat{\rho}_i$ is simple: it consists of filling some gaps (with the runs ρ'_j).

Using the induction hypothesis, we know that there is a weight preserving bijection between simple accepting runs $\hat{\rho}_i$ of $\mathcal{A}_{q_i}$ over $(G, \sigma[x \mapsto i])$ and accepting runs $\overline{\rho}_i$ of $\mathcal{B}_{q_i}$ over $(G, \sigma[x \mapsto i])$.

We finally build a weight preserving bijection between accepting runs $\overline{\rho}_i$ of $\mathcal{B}_{q_i}$ over $(G, \sigma[x \mapsto i])$ and runs $\overline{g}_i$ of $\mathcal{B}$ over (G, σ) from position 0 to position $n-1$. Finally, coming back to the sequences of $\mathrm{Seq}_{G,\sigma}$, these intermediary bijections permit to build a weight preserving bijection from sequences $(q_0, g_0, \dots, q_i, g_i, \dots, q_{n-1}, g_{n-1}) \in \mathrm{Seq}_{G,\sigma}$ to accepting runs

$$(G, \sigma, q_0, \varepsilon, 0)\overline{g}_0(G, \sigma, \overline{q}_0, \varepsilon, 0) \cdots (G, \sigma, q_{n-1}, \varepsilon, n-1)\overline{g}_{n-1}(G, \sigma, \overline{q}_{n-1}, \varepsilon, n-1)$$

of $\mathcal{B}$ over (G, σ), which concludes the proof of correctness. □

This theorem does not hold in the case of more general classes of ordered graphs, as nested words for example. Indeed, even in the Boolean semiring, consider the language of nested words where every call vertex is labeled with the same letter as its matching return. This language is clearly recognizable with an automaton (without any pebbles) walking in the nested word from left to right and checking on every call vertex – by jumping *back and forth* to its matching return – that these two vertices are labeled with the same letter. Notice that this automaton requires only simple runs. However, it is not difficult to prove that no one-way automaton can recognize this language. Indeed, consider such a one-way automaton $\mathcal{A}$ (without pebbles), having $n = |Q|$ states. We consider nested words belonging to the language and of length $2n + 1$, where vertex $i \in [0 \mathinner{..} n-1]$ is a call vertex matched with the return vertex $2n - i$, and vertex n is labeled with a fixed letter. Over an alphabet with two letters, there are 2^n such nested words, since it suffices to choose the labels of the n first vertices (those of index $i \in [0 \mathinner{..} n-1]$). For each of these nested words, automaton $\mathcal{A}$ must have an accepting run, that visits every vertex (otherwise we may easily swap the labels of one of the vertex not visited and obtain a contradiction). As it is one-way, it necessarily visits the first n vertices before the n last ones. Considering the state reached on vertex n (the one with a fixed letter), there are two distinct nested words with this state being identical. By considering the nested word composed of the first n letters of one, and the last n ones of the other, we obtain a nested word that is not in the language, but that is recognized by $\mathcal{A}$. Hence, this contradicts the existence of a one-way automaton recognizing the language.

However, even for general classes of graphs, every layered pebble weighted automaton equipped with the simple semantics is equivalent to a *weighted graph automaton*, namely a graph automaton covering graphs with tiles as described, e.g., by [Tho91], where transitions are moreover equipped with weights. This model has been explored in [Fic07]. We do not investigate the question in further details in this manuscript but the idea is again to use crossing-sequences: whereas for words, we can verify with a one-way automaton that two successive crossing-sequences can be matched, we need the power of graph automata for more general classes of graphs. For example, in nested words, at a given return vertex, we may use the information of the corresponding call vertex to check if the crossing-sequences form a real run.

7.5 Restrictions over Expressions and Logics

We informally propose in this last section some possibilities to restrict expressions and logics to remove the need of computing infinite sums. Throughout this section, we only consider ordered graphs. Hence, we let D be an ordered set of directions, and we denote by $D_{\rightarrow}$ and $D_{\leftarrow}$ the partition of D into forward and backward directions. We also fix a semiring $(\mathbb{S}, \oplus, \otimes, 0, 1)$, not necessarily continuous.

7.5.1 One-way Weighted Expressions

Restricting ourselves to continuous semirings seems somewhat annoying but unavoidable in order to ensure the well-definition of the semantics of expressions E^+ which, in the most general case, involve an infinite sum (see Table 3.2). Mimicking one-way pebble weighted automata, we may consider one-way hybrid weighted expressions, with a *proper* restriction, as considered by Schützenberger: proper expressions are the only expressions allowed under a Kleene star operation. Hence, we consider the set of *one-way hybrid weighted expressions* E defined by the following grammar:

$$E ::= s \mid \alpha \mid x!E \mid \underline{E} \mid E+E \mid E\cdot E$$
$$\underline{E} ::= d_\to \mid \underline{E}+\underline{E} \mid E\cdot\underline{E} \mid \underline{E}\cdot E \mid \underline{E}^+$$

where $s \in \mathbb{S}$, $\alpha \in \text{Test}$, $d_\to \in D_\to$, and $x \in \text{Var}$. Expressions $\underline{E}$ are called proper one-way hybrid weighted expressions.

As a subclass of HWE, one-way hybrid weighted expressions inherits their semantics from those of hybrid weighted expressions, in the case of a continuous semiring. We now show that, indeed, we can extend the semirings to non necessarily continuous ones.

Lemma 7.12. *Let E and $\underline{E}$ be respectively a one-way and proper one-way hybrid weighted expression. Then,*

$$[\![E]\!](G,\sigma,v,v') \neq 0 \textit{ implies } v \leq v' \quad \textit{and} \quad [\![\underline{E}]\!](G,\sigma,v,v') \neq 0 \textit{ implies } v < v'$$

for every ordered pointed graph $G \in \mathcal{G}(A,D)$, vertices $v,v' \in V$ and valuation $\sigma\colon \text{Var} \rightharpoonup V$, where $\leq$ denotes the order on the vertices of G.

Proof. The proof is by simultaneous induction over one-way and proper one-way hybrid weighted expressions. For convenience, we present first the induction over one-way hybrid weighted expressions.

If $E = s \in \mathbb{S}$, $E = \alpha \in \text{Test}$ or $E = x!E'$, $[\![E]\!](G,\sigma,v,v') \neq 0$ implies $v = v'$, and hence $v \leq v'$.

If $E = \underline{E}$ is proper, we directly conclude by induction.

If $E = E_1 + E_2$, then $[\![E]\!](G,\sigma,v,v') \neq 0$ implies that either $[\![E_1]\!](G,\sigma,v,v') \neq 0$ or $[\![E_2]\!](G,\sigma,v,v') \neq 0$ (as $0 \oplus 0 = 0$), which permits to conclude by induction.

If $E = E_1 \cdot E_2$, then $[\![E]\!](G,\sigma,v,v') \neq 0$ implies that there exists a vertex $v'' \in V$ such that $[\![E_1]\!](G,\sigma,v,v'') \otimes [\![E_2]\!](G,\sigma,v'',v') \neq 0$, and hence that both $[\![E_1]\!](G,\sigma,v,v'') \neq 0$ and $[\![E_2]\!](G,\sigma,v'',v') \neq 0$ (as $0 \otimes s = s \otimes 0 = 0$ for every $s \in \mathbb{S}$). By induction, this proves that $v \leq v''$ and $v'' \leq v'$, which permits to deduce that $v \leq v'$ by transitivity.

We now turn to the part of the induction proof related to proper one-way hybrid weighted expressions. If $\underline{E} = d_\to \in D_\to$, $[\![\underline{E}]\!](G,\sigma,v,v') \neq 0$ implies $(v,v') \in E_{d_\to}$: by definition of ordered graphs, this ensures that $v < v'$.

The proofs in case $\underline{E}$ is of the form $\underline{E}_1 + \underline{E}_2$, $E_1 \cdot \underline{E}_2$ or $\underline{E}_1 \cdot E_2$ are similar to the one-way case, achieved previously.

Finally, if $\underline{E} = (\underline{E}_1)^+$, then $[\![\underline{E}]\!](G,\sigma,v,v') \neq 0$ implies that there exists $n \geq 1$ such that $[\![(\underline{E}_1)^n]\!](G,\sigma,v,v') \neq 0$. An easy induction over n permits to show that

$$[\![(\underline{E}_1)^n]\!](G,\sigma,v,v') = \bigoplus_{v_1,\ldots,v_{n-1}\in V} \bigotimes_{j=0}^{n-1} [\![\underline{E}_1]\!](G,\sigma,v_j,v_{j+1})$$

where we let $v_0 = v$ and $v_n = v'$. Hence, we can deduce that there exists $v_1,\ldots,v_{n-1} \in V$ such that for all $j \in [0\,..\,n-1]$, $[\![\underline{E}_1]\!](G,\sigma,v_j,v_{j+1}) \neq 0$. By induction, this shows that

$$v = v_0 < v_1 < \cdots < v_{n-1} < v_n = v'$$

Knowing that $n \geq 1$, this shows by transitivity that $v < v'$. □

This lemma shows that the *a priori* infinite sum used to define the semantics of $\underline{E}^+$ contains indeed only a finite number of non-zero elements, and hence exists in all semirings $\mathbb{S}$: more precisely, if n is the number of vertices in an ordered pointed graph G, and $\underline{E}$ is a proper one-way pebble weighted expressions, we have

$$[\![\underline{E}^k]\!](G,\sigma,v,v') = 0$$

for every $k \geq n+1$, valuation $\sigma\colon \mathrm{Var} \rightharpoonup V$ (with $\mathrm{Free}(\underline{E}) \subseteq \mathrm{dom}(\sigma)$), and $v, v' \in V$. Finally, we have shown that we can define the semantics of one-way hybrid weighted expressions with weights in a semiring, non necessarily continuous.

We believe that it is now possible to prove another correspondence theorem between one-way pebble weighted automata and one-way hybrid weighted expressions. The idea would be to adapt the proof of Theorem 4.31 so that both constructions ensure the one-way restrictions. Starting with an automaton, we would need to prove that the proof of the general case (in particular the computation of the star of a matrix) still goes through this one-way restriction: even though the set of series over (dynamically) marked graphs is no longer a continuous semiring, we think that it would be possible to extract from it a *star semiring* (see e.g., [KS85] for a formal definition) considering only *proper* marked graphs, i.e., those in which initial and final vertices are not the same. For the other direction, the construction of an automaton from an expression is exactly the same, and only the proof of correctness has to be adapted, again dealing with star semirings.

Our choice of one-way pebble weighted expressions may seem too restrictive. In particular, for the operator $x!E$, it is not absolutely necessary that E is one-way too. We could also imagine that E is going backward only, or more generally is changing from a forward to a backward computation and vice versa only a finite number of times. However, Kleene iterations must still be applied over expressions using only one type of directions, and being proper. Hence, a Kleene operator will only be allowed over *forward-proper* or *backward-proper* expressions, namely expressions that may only move in one direction, and will not stay on the same position. An alternative restriction would then consist in defining simultaneously forward, forward-proper, backward, backward-proper and one-way hybrid weighted expressions by the following grammar:

$$\begin{aligned}
E &::= F \mid B \mid E+E \mid E \cdot E\\
F &::= s \mid \alpha \mid x!E \mid \underline{F} \mid F+F \mid F \cdot F\\
\underline{F} &::= d_\rightarrow \mid \underline{F}+\underline{F} \mid F \cdot \underline{F} \mid \underline{F} \cdot F \mid \underline{F}^+\\
B &::= s \mid \alpha \mid x!E \mid \underline{B} \mid B+B \mid B \cdot B\\
\underline{B} &::= d_\leftarrow \mid \underline{B}+\underline{B} \mid B \cdot \underline{B} \mid \underline{B} \cdot B \mid \underline{B}^+
\end{aligned}$$

where $s \in \mathbb{S}$, $\alpha \in \mathrm{Test}$, $d_\rightarrow \in D_\rightarrow$, $d_\leftarrow \in D_\leftarrow$, and $x \in \mathrm{Var}$. Expressions F (respectively, B) are called forward hybrid weighted expressions (respectively, backward hybrid weighted expressions), whereas their underlined versions are their proper versions.

The proof of Lemma 7.12 can easily be adapted to prove that

Lemma 7.13. *Let F and $\underline{F}$ be respectively a forward and proper forward hybrid weighted expression. Then,*

$$[\![F]\!](G,\sigma,v,v') \neq 0 \textit{ implies } v \leq v' \quad \textit{and} \quad [\![\underline{F}]\!](G,\sigma,v,v') \neq 0 \textit{ implies } v < v'$$

for every ordered pointed graph $G = (V,E,\lambda,\chi,v^{(i)},v^{(f)}) \in \mathcal{G}(A,D)$, vertices $v, v' \in V$ and valuation $\sigma\colon \mathrm{Var} \rightharpoonup V$, where $\leq$ denotes the order on the vertices of G.

Similarly, let B and $\underline{B}$ be respectively a backward and proper backward hybrid weighted expression. Then,

$$[\![B]\!](G,\sigma,v,v') \neq 0 \textit{ implies } v' \leq v \quad \textit{and} \quad [\![\underline{B}]\!](G,\sigma,v,v') \neq 0 \textit{ implies } v' < v\,.$$

Indeed, only the base case of formulae $x!E$ has to be adapted, and the proof for the backward version is symmetrical. Hence, this shows that we can again define the semantics of one-way hybrid weighted expressions with weights in a semiring, not necessarily continuous.

In the case of words, reversal-bounded hybrid weighted expressions can not generate more behaviors than one-way hybrid weighted expressions. Indeed, using Theorem 4.31 again, reversal-bounded hybrid weighted expressions can be translated into pebble weighted automata that do not loop: in particular, we can use their simple semantics and use Theorem 7.8 to transform them into one-way pebble weighted automata. We may then conclude by the result stated before linking one-way hybrid weighted expressions and one-way pebble weighted automata.

7.5.2 First-order logic with Restricted Transitive Closure

We now finish this chapter by presenting a logical fragment permitting to express exactly the expressive power of pebble weighted automata equipped with a simple semantics. We have studied this extension in [1] in a slightly different formalism and thus do not develop it further in this manuscript.

Indeed, the idea is to restrict the transitive closure to a simple and a one-way transitive closure, as we did for automata.

The simplicity restriction, which is a semantical restriction, permits to define a transitive closure in non-continuous semirings. For a formula $\Phi(x,y)$ with at least two free variables x and y, the *simple weighted transitive closure operator* $[\mathrm{swTC}_{x,y}\Phi](x',y')$ has as free variables the fresh variables x' and y', and all the free variables of Φ except x and y. Its semantics is defined by the same formula as in (6.3) where the sum now runs over all $m > 0$ and all sequences $(v_k)_{0\leq k\leq m}$ of vertices of the graph G with $v_0 = \sigma(x')$ and $v_m = \sigma(y')$, such that, either $m = 1$, or $m > 1$ and all positions of the sequence are pairwise distinct. This sum is finite as the graph has finitely many vertices. Notice that if $\sigma(x') = \sigma(y') = v$, then the semantics is $[\![\Phi]\!](G, \sigma[x \mapsto v, y \mapsto v])$.

Example 7.14. Let $\Psi = \mathrm{swTC}_{x,y}1$ over $\mathbb{N}$. Then for a word u of length n, we have $[\![\Psi]\!](u,1,n) = \sum_{m=0}^{n-2} m!\binom{n-2}{m}$, since the sum in the semantics of swTC ranges over all sequences of pairwise distinct positions in $[1\mathinner{..}n]$ starting in 1 and ending in n. ■

For the *one-way* restriction, we introduce the operator $\mathrm{wTC}^{\rightarrow}_{x,y}\Phi$ whose semantics consists in restricting the sum of (6.3) when $m > 1$ to *increasing* sequences $(v_k)_{0\leq k\leq m}$ of vertices of the ordered graph G. Equivalently, we can define it by

$$\mathrm{wTC}^{\rightarrow}_{x,y}\Phi = \mathrm{swTC}_{x,y}(x \leq y \otimes \Phi) = (x = y \otimes \Phi(x,y)) \oplus \mathrm{wTC}_{x,y}(x < y \otimes \Phi)$$

as the simplicity condition ensures that non-decreasing simple sequences of length $m > 1$ are indeed increasing. Intuitively, the $\mathrm{wTC}^{\rightarrow}_{x,y}$ operator generalizes the forward transitive closure operator of the Boolean case: a formula $\varphi(x,y)$ with two free variables defines a binary relation on positions of a word w, namely $\{(i,j) \mid (i < j) \wedge w,i,j \models \varphi\}$. The relation defined by $\mathrm{wTC}^{\rightarrow}_{x,y}\varphi$ is the transitive closure of this relation.

An alternative to define the semantics of this operator, is to use powers of the formula: we define recursively the m-th power of a formula Φ, with this one-way restriction

$$\begin{aligned}\Phi^1_{\leq}(x,y) &= (x \leq y) \otimes \Phi(x,y),\\ \Phi^{m+1}_{\leq}(x,y) &= \textstyle\bigoplus_z \big[(x < z < y) \otimes \Phi(x,z) \otimes \Phi^m_{\leq}(z,y)\big], \quad \text{for } m \geq 1.\end{aligned}$$

The left-to-right restriction of the transitive closure permits finally to give a third equivalent definition, *a priori* simpler to use:

$$[\![\mathrm{wTC}^{\rightarrow}_{x,y}\Phi]\!] = \sum_{m\geq 1} [\![\Phi^m_{\leq}(x,y)]\!] .$$

This infinite sum is well-defined, since on each pair (G, σ), only finitely many terms assume a nonzero value: indeed $[\![\Phi^m_{\leq}(x,y)]\!](G,\sigma) = 0$ if $m \geq |V|$, the number of vertices of G.

Finally, we can reconsider the bounded restriction considered in Chapter 6, and introduce *bounded one-way* and *bounded simple* weighted transitive closure operators, respectively denoted by $\mathrm{wTC}^{\mathrm{b}\rightarrow}_{x,y}\Phi$ and $\mathrm{swTC}^{\mathrm{b}}_{x,y}\Phi$.

Following exactly the same proof as the one of Theorem 6.10 with the one-way restriction over the transitive closure, we would be able to prove that every formula of $\mathrm{wFO}+\mathrm{wTC}^{\mathrm{b}\rightarrow}(\mathrm{FO})$ can be translated into an equivalent layered one-way pebble weighted automaton 1PWA, over a searchable class of graphs. This is an extension of Theorem 11(3) in [1].

Reciprocally, we proved in the case of words in [1] that layered pebble weighted automata equipped with the simple semantics can be translated into a formula of $\mathrm{wFO}+\mathrm{wTC}^{\mathrm{b}\rightarrow}(\mathrm{FO})$, first by making them one-way and then by mimicking the proof we gave in a more general setting for Theorem 6.15.

CHAPTER **8**

Probabilistic Specification Formalisms

> Les grands artistes ont du hasard dans leur talent et du talent dans leur hasard.[1]
>
> Victor Hugo, *Océan Prose*

This chapter is devoted to the study of probabilistic specifications. There are actually a variety of models for probabilistic systems, comprising Segala systems, generative systems, stratified systems, Markov chains, etc. (see [vGSS95, Seg06] for overviews). Those models may involve non-determinism and *generate* some behavior according to probability distributions over states. Alternatively, they may make a probabilistic decision depending on the input letter, like (reactive) probabilistic automata [Paz71]. The latter go back to Rabin [Rab63] and are an object of ongoing research considering decision problems such as emptiness, language equivalence [Sch61, Tze92, CMR06, KMO$^+$12], and the value 1 problem [GO10].

Our starting point of view, as in all this manuscript, is that specifications shall represent quantitative properties of graphs. In particular, rather than at bisimulation equivalence, we are looking at language equivalence in terms of formal power series, i.e., mappings from graphs to elements from the real-valued interval $[0,1]$. Our goal is to apply our results in this particular case of probabilistic specifications. Especially, what kind of probabilistic properties can we specify with hybrid weighted expressions/pebble weighted automata/formulae? As a first attempt, we could simply consider these specification formalisms with weights restricted to the subset $[0,1]$ of the continuous semiring $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$. However, this first naive solution is not satisfactory. Indeed, having only weights in $[0,1]$ does not ensure that the semantics of every graph will lie in the interval $[0,1]$ and henceforth be interpreted as a probability.

In the first section, we consider *pebble probabilistic automata* (that capture two-way probabilistic automata [DS89, Rav07] over words for instance) and define the class of *hybrid*

[1] "Great artists have randomness in their talent and talent in their randomness."

probabilistic expressions, as a syntactical fragment of hybrid weighted expressions. In particular, the sum and the Kleene star operator have to be handled with care. We prove that these two models generate exactly the same probabilistic behaviors: it has to be noticed that we have not been able to avoid redoing the whole proof of Theorem 4.31.

In the second section, we study the special case of classical Rabin probabilistic automata, i.e., one-way automata over finite words. We define a set of probabilistic expressions that are equivalent to probabilistic automata.

It has to be noted that there have been numerous approaches to characterizing probabilistic systems in terms of algebraic expressions and process calculi [vGSS95, BK01, DP07]. A unifying framework is due to [SBBR11], who consider probabilistic systems in a general coalgebraic setting. This allows them to derive algebraic expressions and a corresponding Kleene theorem, as well as full axiomatizations for many of those (and even for weighted automata over arbitrary semirings). Their and above-mentioned works are mainly aiming at axiomatization of probabilistic-system behaviors in terms of bisimulation equivalence, so their focus is on *system* models including non-determinism and generative probability distributions. In this chapter, we consider probabilistic automata, which are a more appropriate machine model for our purpose, e.g., for *evaluating* queries. Moreover, while the syntax of process-algebraic expressions is tailored to modeling probabilistic systems and uses action prefixing, fixed points, and process variables, we provide expressions with concatenation and a Kleene star. Thus, our expressions are closer to language-theoretic operations and more convenient to use in query languages.

Note that the fact that we also consider navigating devices distinguishes our work from all above-mentioned references. To the best of our knowledge, we present the first Kleene-Schützenberger correspondence for probabilistic two-way automata. Results of this chapter have been presented in [3] in the case of words.

Throughout this chapter, we consider the continuous semiring $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$.

8.1 Pebble Probabilistic Automata

In this section and the following, we explain how to interpret (pebble) weighted automata and (hybrid) weighted expressions in a probabilistic setting.

Considering first automata, not every pebble weighted automaton with weights in the subset $[0,1]$ of the semiring $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$ can be interpreted as a probabilistic automaton. Indeed, a first property of interest should be that every graph (e.g., word) is mapped to a weight in $[0,1]$. The definition below puts some syntactical restrictions over the pebble weighted automata to ensure it. Naturally, they have the flavor of those for probabilistic Rabin automata, as introduced in [Rab63].

We reuse here ideas introduced in the proof of Theorem 7.8, namely the fact that a test formula α can be verified using a finite abstraction of the current configuration of an automaton. Whereas only three components were needed over words (namely the letter, the type and the set of pebbles), we now add two more Booleans b_i and b_f in $\{0,1\}$ to check respectively formula init? and final?. Indeed, we define $(a, \tau, \mathcal{P}, b_i, b_f) \models \alpha$ as in Table 7.1, simply changing the cases where α is either init? and final? by

$$(a, \tau, \mathcal{P}, b_i, b_f) \models \text{init? if, and only if, } b_i = 1$$
$$(a, \tau, \mathcal{P}, b_i, b_f) \models \text{final? if, and only if, } b_f = 1$$

For other formulae, b_i and b_f are not used. Similarly to the case of words, for every graph $G \in \mathcal{G}(A, D)$, valuation σ and vertex $v \in V$, we may define $b_i(v)$ (respectively, $b_f(v)$) to be 1 if, and only if, $v = v^{(i)}$ (respectively, $v = v^{(f)}$). Then,

$$G, \sigma, v \models \alpha \text{ if, and only if, } (\lambda(v), \mathsf{type}(v), \sigma^{-1}(v), b_i(v), b_f(v)) \models \alpha\,.$$

In the following, a tuple $(a, \tau, \mathcal{P}, b_i, b_f)$ will be called a *local configuration.* The set of local configurations of an automaton $\mathcal{A}$ is denoted Γ:

$$\Gamma = A \times \mathfrak{P}(D) \times \mathfrak{P}(\mathrm{Var}(\mathcal{A})) \times \{0,1\}^2$$

and such a local configuration will be denoted by γ.

Definition 8.1. A *pebble probabilistic automaton* (PPA) is a pebble weighted automaton $\mathcal{A} = (Q, A, D, I, \Delta, F)$ over $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$ such that the following conditions hold:

1. I is a probability distribution over the states, i.e., $\sum_{q \in Q} I_q = 1$;
2. for all $q \in Q$, and local configuration $\gamma \in \Gamma$, we have[2]

$$\sum_{\alpha \in \mathrm{Test} \mid \gamma \models \alpha} \Big(F_q(\alpha) + \sum_{m \in \mathrm{Actions}, q' \in Q} \Delta_{q,q'}(m)(\alpha) \Big) \leq 1\,.$$

■

The idea is to ensure that for every possible configuration, the transition function defines a probability distribution over the possible continuations (or termination), i.e., the next action to follow, and the next state.

Remark 8.2. In the following, a state $q \in Q$ will be said *accepting* if F_q is different from 0. ■

A priori, $[\![\mathcal{A}]\!]_K$ associates to every pair (G, σ) of graph and valuation an element of $\mathbb{R}^+ \cup \{+\infty\}$. However, one can show the following:

Proposition 8.3. *For every pebble probabilistic automaton $\mathcal{A}$ and bound $K > 0$, every graph $G \in \mathcal{G}(A, D)$ and every valuation σ, we have $[\![\mathcal{A}]\!]_K(G, \sigma) \in [0,1]$.*

Proof. For every pair (G, σ), we prove that the graph of K-configurations is a finite Markov chain, so that the semantics $[\![\mathcal{A}]\!]_K(G, \sigma)$, being interpreted as a probability to reach a set of states of the Markov chain, must be in $[0,1]$.

We first recall that a K-configuration has the shape (G, σ, q, π, v) with $q \in Q$, $\pi \in (\mathrm{Var}(\mathcal{A}) \times V)^k$ with $k \leq K$, and $v \in V$. For a fixed pair (G, σ), this generates a finite set of states for the Markov chain. To this set of K-configurations, we add a single fresh state denoted $\checkmark$.

Clearly, the initial distribution reflects the distribution I, so that the K-configuration $(G, \sigma, q, \varepsilon, v^{(i)})$ has initial probability I_q and others have probability 0.

The transitions of the Markov chain are defined by the concrete transitions between K-configurations $(G, \sigma, q, \pi, v) \rightsquigarrow (G, \sigma, q', \pi', v')$, whose weight has been defined after Definition 4.19. Moreover, we add a transition from any final K-configuration $(G, \sigma, q, \varepsilon, v^{(f)})$ to the state $\checkmark$ with probability defined by the polynomial F_q where test formulae are again interpreted as before. It is not difficult to see that the weights only depend on the local configuration $(\lambda(v), \mathsf{type}(v), \sigma^{-1}(v), b_i(v), b_f(v))$, so that the restriction of Definition 8.1 actually generates a Markov chain.

Finally, the semantics $[\![\mathcal{A}]\!]_K(G, \sigma)$ is the probability to reach the state $\checkmark$, so that it belongs to $[0,1]$. □

The proof above establishes a strong connection between pebble probabilistic automata and finite Markov chains. This connection also provides an algorithm for evaluating a pebble probabilistic automaton with respect to a given graph and valuation, which reduces to computing the probability of reaching a final configuration in the *synchronized* Markov chain (see, for example, [BK08]).

$$\big(p\,(\neg\text{final?})\big)\!\rightarrow + \big((1-p)\,\top\big)\!\leftarrow$$

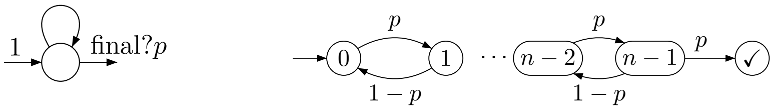


Figure 8.1: A probabilistic automaton $\mathcal{A} \in$ PPA and the Markov chain obtained by synchronizing $\mathcal{A}$ with a word of length n

Example 8.4. (Continued from Example 4.7) Consider the pebble probabilistic automaton $\mathcal{A}$ (2-way, without pebbles) depicted on the left of Figure 8.1 (where $0 < p < 1$) with $D = \{\leftarrow, \rightarrow\}$. The synchronized Markov chain of $\mathcal{A}$ with respect to a graph $G \in \mathcal{W}\mathrm{ord}(A)$, representing a word of length $|V| = n$, is depicted on the right of the same figure. This Markov chain represents a random walk over a straight line of bounded length. Using the same proof as the one used in Proposition 7.3 for the special case $p = 1/2$, we can show that $\mathcal{A}$ verifies

$$[\![\mathcal{A}]\!](G) = \frac{1}{1 + s + \ldots + s^{n+1}}$$

where $s = \frac{1-p}{p}$, for every $G \in \mathcal{W}\mathrm{ord}(A)$. ■

8.2 Hybrid Probabilistic Expressions

Now, we define hybrid probabilistic expressions that capture the expressive power of probabilistic pebble automata. These are syntactical restrictions of hybrid weighted expressions in HWE. In the following, we consider hybrid weighted expressions with weights in $[0, 1]$, modulo the following trivial identities:

$$0 + E \equiv E + 0 \equiv E \qquad E \cdot 0 \equiv 0 \cdot E \equiv 0 \qquad E \cdot 1 \equiv 1 \cdot E \equiv E \qquad 0^* \equiv 1$$

as well as $p \cdot E \equiv E \cdot p$ (for $p \in [0, 1]$) which models commutativity.

As for automata, we have to restrict the syntax of expressions since otherwise values greater than 1 could be obtained. For instance, the weighted expression over words $(a + ab)(ba + a)$ should not be a *probabilistic* expression since it evaluates to 2 on the word aba. The restriction will be both on sum and star.

Since we aim at expressions which are equivalent to pebble probabilistic automata, we examine first which type of hybrid weighted expressions are obtained from pebble probabilistic automata by applying our construction of Theorem 4.31 (interpreting a pebble probabilistic automaton as a pebble weighted automaton). For the example automaton in Figure 8.2, the weighted expression would be $\big[\frac{1}{6}a(a+b) + \frac{1}{2}a\big]^\star \cdot (\frac{1}{3}a? + b?)$. Now, the expression $\big[\frac{1}{6}a(a+b) + \frac{1}{2}a\big]^\star \cdot (a? + b?)$, obtained by changing the subexpression $\frac{1}{3}a?$ into $a?$, should be disallowed, because it corresponds to an automaton violating the condition over its transition matrix. On the other hand, $\big[\frac{1}{6}a(a+b)+\frac{1}{2}a\big]^* \cdot (\frac{1}{3}a? + \frac{1}{2}b?)$ would be acceptable: we obtain a corresponding probabilistic automaton from the previous automaton depicted in Figure 8.2 by setting $\mathbb{P}(1, b, 3) = \frac{1}{2}$.

Henceforth, we will restrict the sum and star operations to get the *probabilistic* fragment. Doing so, we lose commutativity, associativity and distributivity hence we enforce these properties explicitly. The rule for introducing concatenation is also strengthened to allow concatenation inside a context: if $E_1 + E_2$ and G are probabilistic expressions, so does $E_1 + E_2 \cdot G$.

[2] Actions $= D \cup \{\mathsf{drop}_x \mid x \in \mathrm{Var}\} \cup \{\mathsf{lift}\}$ is the set of available actions

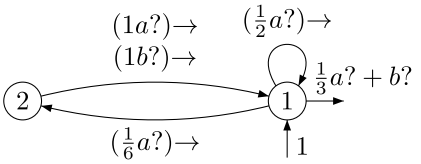


Figure 8.2: A probabilistic automaton equivalent to $\left[\frac{1}{6}a(a+b)+\frac{1}{2}a\right]^\star \cdot (\frac{1}{3}a? + b?)$

Definition 8.5. *Hybrid probabilistic expressions* (HPE) are expressions in HWE built by the following rules[3]:

- the atom rules:

$$(p)\ \frac{p \in [0,1]}{p \in \mathrm{HPE}} \qquad (\alpha)\ \frac{\alpha \in \mathrm{Test}}{\alpha \in \mathrm{HPE}} \qquad (d)\ \frac{d \in D}{d \in \mathrm{HPE}}$$

- the inductive rules:

$$(+_{\mathrm{det}})\ \frac{E_1 \in \mathrm{HPE} \quad E_2 \in \mathrm{HPE} \quad \alpha \in \mathrm{Test}}{\alpha \cdot E_1 + (\neg\alpha)\cdot E_2 \in \mathrm{HPE}}$$

$$(+_{\mathrm{prob}})\ \frac{E_1 \in \mathrm{HPE} \quad E_2 \in \mathrm{HPE} \quad p \in [0,1]}{p \cdot E_1 + (1-p)\cdot E_2 \in \mathrm{HPE}}$$

$$(\cdot)\ \frac{E_1 + E_2 \in \mathrm{HPE} \quad G \in \mathrm{HPE}}{E_1 + E_2 \cdot G \in \mathrm{HPE}} \qquad (\star)\ \frac{E_1 + E_2 \in \mathrm{HPE}}{E_1^\star \cdot E_2 \in \mathrm{HPE}} \qquad (x!)\ \frac{E \in \mathrm{HPE} \quad x \in \mathrm{Var}}{x!E \in \mathrm{HPE}}$$

- the following associativity, commutativity and distributivity rules (later denoted as ACD-rules):

$$\begin{array}{llcl}
\mathbf{A}_+ & E + (G+H) \in \mathrm{HPE} & \longleftrightarrow & (E+G)+H \in \mathrm{HPE} \\
\mathbf{C}_+ & E + G \in \mathrm{HPE} & \longleftrightarrow & G + E \in \mathrm{HPE} \\
\mathbf{A}_\cdot & E \cdot (G \cdot H) \in \mathrm{HPE} & \longleftrightarrow & (E \cdot G) \cdot H \in \mathrm{HPE} \\
\mathbf{D}_\cdot & E \cdot (G+H) \in \mathrm{HPE} & \longleftrightarrow & E \cdot G + E \cdot H \in \mathrm{HPE} \\
\mathbf{D}_\cdot & (E+G) \cdot H \in \mathrm{HPE} & \longleftrightarrow & E \cdot H + G \cdot H \in \mathrm{HPE} \\
\mathbf{D}_{x!} & x!(E+G) \in \mathrm{HPE} & \longleftrightarrow & x!E + x!G \in \mathrm{HPE}
\end{array}$$

■

There are two *guarded* sums. The first one ($+_{\mathrm{det}}$) is deterministic and guarded by a test formula. The second one ($+_{\mathrm{prob}}$) is probabilistic. Also, the star operation contains an implicit choice which is either to iterate again the expression or to exit the loop. This choice also has to be guarded which is the reason for the precondition $E_1 + E_2 \in \mathrm{PE}$ in the rule ($\star$). The guard could be deterministic as in $(ab)^\star b$, i.e., $(a? \cdot \rightarrow \cdot b? \cdot \rightarrow)^\star \cdot b? \cdot \rightarrow$ when written in plain, or probabilistic as in $(\frac{1}{3}(aa+bb))^\star \frac{2}{3}$. Finally, with the above restrictions, we lose the classical ACD identities, hence we enforce these properties explicitely with the ACD-rules which allow to rewrite a hybrid probabilistic expression in order to apply the star rule as needed.

The semantics of hybrid probabilistic expressions is inherited from the one of hybrid weighted expressions. It is not straightforward that expressions from HPE generate only probabilistic behaviors, in particular because of the rules ($\cdot$) and ($\star$). We will indeed not prove it directly but obtain this fact from Theorem 8.8 translating every hybrid probabilistic expression into an equivalent pebble probabilistic automaton.

[3] The rules have to be understood in the following sense: on the top, there are preconditions to be fullfilled, in which case, the bottom builds a new hybrid probabilistic expression. For example, the rule ($+_{\mathrm{prob}}$) constructs the expression $p \cdot E_1 + (1-p)\cdot E_2$ at the condition that E_1 and E_2 are hybrid probabilistic expressions and that p is a probability.

Example 8.6 (Continued from Example 8.4)**.** An equivalent probabilistic expression to describe the probability associated with the previous random walk is given by

$$E = (p(\neg\text{final?})\!\rightarrow + (1-p)\!\leftarrow)^\star \text{final?}\, p\,.$$

Moreover, let $G = p(\neg\text{final?})\!\rightarrow + (1-p)\!\leftarrow$ so that $E = G^\star\text{final?}\,p$. Notice that E is indeed an expression in HPE because $G + \text{final?}\,p \in \text{HPE}$, since it can be rewritten by ACD-rules, and equivalences into $p(\text{final?} + \neg\text{final?}\!\rightarrow) + (1-p)\!\leftarrow\ \in \text{HPE}$. ■

We define now the multiset $\text{Terms}(E)$ of terms of an expression $E \in \text{HPE}$. This will be crucial for the translation from expressions to automata in the next section. Intuitively, if we suppose that summation is pushed up as much as possible by means of ACD-rules, then the *multiset* of terms consists of all expressions that occur in this big outermost sum. Formally, the definition is by induction over $E \in \text{HPE}$. When E is an atom, we let $\text{Terms}(E) = \{\!\{E\}\!\}$ be the singleton multiset containing only the atom itself. Moreover,[4]

$$\begin{aligned}
\text{Terms}(E_1 + E_2) &= \text{Terms}(E_1) \uplus \text{Terms}(E_2)\\
\text{Terms}(E_1 \cdot E_2) &= \{\!\{E_1' \cdot E_2' \mid E_1' \in \text{Terms}(E_1), E_2' \in \text{Terms}(E_2)\}\!\}\\
\text{Terms}(E^\star) &= \{\!\{E^\star\}\!\}\\
\text{Terms}(x!E) &= \{\!\{x!E' \mid E' \in \text{Terms}(E)\}\!\}\,.
\end{aligned}$$

Note that, if an expression G can be obtained from an expression E through ACD-rules, then we have $\text{Terms}(G) = \text{Terms}(E)$. Also, it is immediate by induction that the converse also holds:

Lemma 8.7. *Let* $E \in \text{HPE}$ *with* $\text{Terms}(E) = \{\!\{E_i \mid i \in \mathcal{I}\}\!\}$. *Using ACD-rules, we can rewrite* E *into* $\sum_{i\in\mathcal{I}} E_i$. *In particular, we have* $[\![E]\!] = \sum_{i\in\mathcal{I}}[\![E_i]\!]$. *Henceforth, we identify* E *and* $\sum_{i\in\mathcal{I}} E_i$.

Notice finally that, by making use of the context in the rule $(\cdot)$, any sum of (some) terms of a probabilistic expression is also a probabilistic expression: if $\mathcal{I}' \subseteq \mathcal{I}$ we get $\sum_{i\in\mathcal{I}'} E_i \in \text{HPE}$ by concatenating expression 0 after expressions E_i with $i \notin \mathcal{I}'$, and applying equivalences.

8.3 The Probabilistic Kleene-Schützenberger Theorem

We prove in this section that pebble probabilistic automata are effectively equivalent to hybrid probabilistic expressions. Notice that we cannot simply use the proof of the general Schützenberger-Kleene theorem, stated in Theorem 4.31, as there are restrictions both on the automata and the expressions, which are of different natures.

We start with the construction of automata from expressions. The problematic cases are concatenation and iteration due to the precondition $E + F \in \text{HPE}$. To deal with these cases, we construct from $E \in \text{HPE}$ a pebble probabilistic automaton $\mathcal{A}$ which simultaneously recognizes all *terms* of E.

Theorem 8.8. *From any expression* $E \in \text{HPE}$*, we can effectively construct an equivalent layered pebble probabilistic automaton* $\mathcal{A} = (Q, A, D, I, \Delta, F)$. *More precisely, if* $\text{Terms}(E) = \{\!\{E_i \mid i \in \mathcal{I}\}\!\}$*, the set of accepting states of* $\mathcal{A}$ *is* $\{f_i \mid i \in \mathcal{I}\}$ *and for all* $i \in \mathcal{I}$ *the expression* E_i *is equivalent to the pebble probabilistic automaton* $\mathcal{A}[f_i] = (Q, A, D, I, \Delta, \widetilde{F})$ *obtained by considering* f_i *as only accepting state, i.e.,* $\widetilde{F}_{f_i} = F_{f_i}$ *and* $\widetilde{F}_q = 0$ *for* $q \neq f_i$.

[4] Here, and in the following, $A \uplus B$ denotes the sum (i.e., the disjoint union) of the multisets A and B: in particular, it takes into account the repetitions.

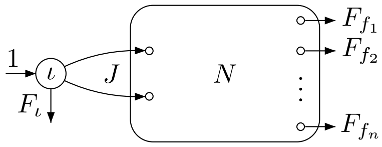


Figure 8.3: Standard pebble probabilistic automaton produced by the construction, from a hybrid probabilistic expression with terms $\{\!\{E_1, E_2, \dots, E_n\}\!\}$. Notice that the initial state ι may be related to one of the terms, as it is shown on the picture.

Proof. The construction is by structural induction on the expression $E \in$ HPE. As for Theorem 4.31, the automaton we construct is a standard automaton, i.e., it has a unique initial state ι, with no ingoing transition. We have depicted in Figure 8.3 the pebble probabilistic automaton $\mathcal{A}$ constructed from a hybrid probabilistic expression E. In particular, we separate the matrix Δ of transitions, in blocks, J regrouping the transitions from the initial state ι, and N the transitions from (and to) states distinct from ι.

In this proof, we say that $\mathcal{A}$ and E are equivalent if, for all graphs $G \in \mathcal{G}(A, D)$, valuation $\sigma\colon \mathrm{Free}(E) \to V$, and vertices $u, v \in V$,

$$[\![E]\!](G, \sigma, u, v) = \sum_{q,q' \in Q} I_q \times [\![\mathcal{A}_{q,q'}]\!](G, \sigma, u, v) \times \sum_{\alpha \in \mathrm{Test} \mid G, \sigma, v \models \alpha} F_{q'}(\alpha)\,.$$

The cases $p \in [0, 1]$, $\alpha \in$ Test, and $d \in D$ are done as in the non-probabilistic case. For each atom, the resulting automaton is in fact a probabilistic automaton, making use of the fact that F may contain some test for the expression α.

Now let $E, E' \in$ HPE be such that $\mathrm{Terms}(E) = \{\!\{E_i \mid i \in \mathcal{I}\}\!\}$ and $\mathrm{Terms}(E') = \{\!\{E'_j \mid j \in \mathcal{J}\}\!\}$. By induction hypothesis, we have constructed two suitable pebble probabilistic automata $\mathcal{A} = (Q, A, D, I, \Delta, F)$ and $\mathcal{A}' = (Q', A, D, I', \Delta', F')$ with respective accepting states $\{f_i \mid i \in \mathcal{I}\}$ and $\{f'_j \mid j \in \mathcal{J}\}$. Without loss of generality, we assume that $Q \cap Q' = \emptyset$, and we denote ι and ι' their respective unique initial states.

Consider $E'' = \alpha E + \neg\alpha E'$. We have $\mathrm{Terms}(E'') = \{\!\{\alpha E_i \mid i \in \mathcal{I}\}\!\} \uplus \{\!\{\neg\alpha E'_j \mid j \in \mathcal{J}\}\!\}$. We construct a pebble probabilistic automaton $\mathcal{A}'' = (Q \uplus Q' \uplus \{\iota''\}, A, D, I'', \Delta'', F'')$ with a fresh state ι'', unique initial state in the distribution I''. We let

$$F''_{\iota''} = \sum_{\alpha' \in \mathrm{Test}} F_\iota(\alpha')\ (\alpha \wedge \alpha') + F'_{\iota'}(\alpha')\ (\neg\alpha \wedge \alpha')$$

whereas $F''_q = F_q$ if $q \in Q$ and $F''_q = F'_q$ if $q \in Q'$. From the new initial state ι'', we duplicate each transition from ι to a state $q \in Q$ by testing moreover the formula α, and similarly for transitions to $q' \in Q'$:

$$\begin{aligned}
\Delta''_{\iota'',q}(d)(\alpha \wedge \alpha') &= \Delta_{\iota,q}(d)(\alpha') && \text{for } q \in Q, \alpha' \in \mathrm{Test}, d \in D\\
\Delta''_{\iota'',q}(d)(\neg\alpha \wedge \alpha') &= \Delta'_{\iota',q}(d)(\alpha') && \text{for } q \in Q', \alpha' \in \mathrm{Test}, d \in D\\
\Delta''_{q,q'} &= \Delta_{q,q'} && \text{for } q, q' \in Q\\
\Delta''_{q,q'} &= \Delta'_{q,q'} && \text{for } q, q' \in Q'
\end{aligned}$$

The construction for $E'' = pE + (1-p)E'$ is similar, and depicted in Figure 8.4. We simply multiply the probability of the first transition from ι'' by the probability p or $1-p$ depending on whether the next state is in Q or Q', and let $F''_{\iota''} = pF_\iota + (1-p)F'_{\iota'}$. Notice that in the picture, we did not mention the previous initial states ι and ι' as they are not reachable anymore.

For the concatenation, we assume that $E = G + H$ and $E'' = G + H \cdot E'$. We have $\mathcal{I} = \mathcal{K} \uplus \mathcal{L}$ with $\mathrm{Terms}(G) = \{\!\{E_i \mid i \in \mathcal{K}\}\!\}$ and $\mathrm{Terms}(H) = \{\!\{E_i \mid i \in \mathcal{L}\}\!\}$. Hence, we have

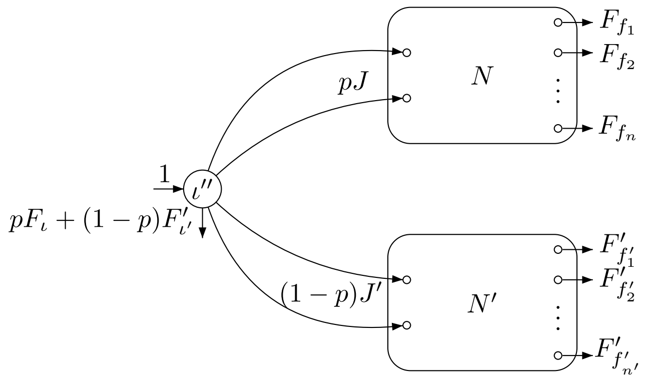


Figure 8.4: Construction for the rule $(+_{\text{prob}})$

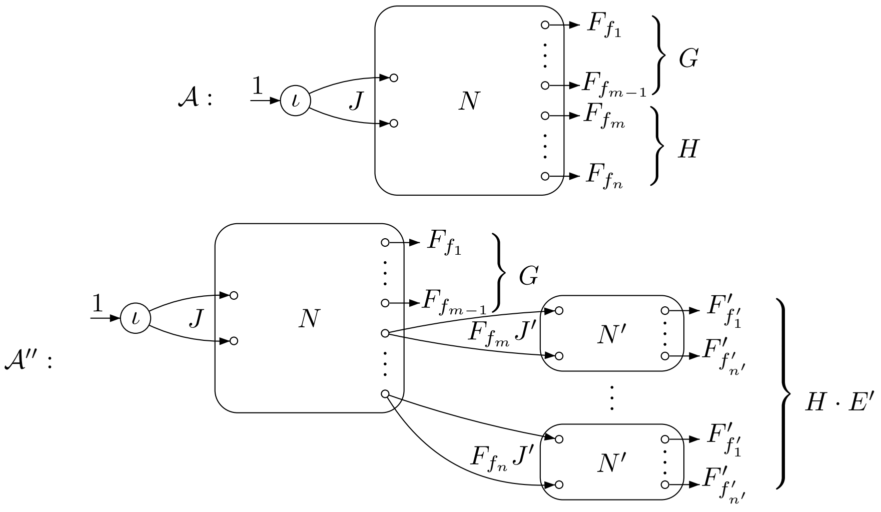


Figure 8.5: Construction for the rule $(\cdot)$: on the top, automaton $\mathcal{A}$ equivalent to expression $E = G + H$ (for simplifying, ι is supposed to be not associated to a term in this picture), and on the bottom, automaton $\mathcal{A}''$ equivalent to expression $E'' = G + H \cdot E'$

$\text{Terms}(E'') = \{\!\{E_i \mid i \in \mathcal{K}\}\!\} \uplus \{\!\{E_i \cdot E'_j \mid (i,j) \in \mathcal{L} \times \mathcal{J}\}\!\}$. Notice that the construction used in Theorem 4.9 does not work here, because of the requirement to keep track of the different terms of the expression. The idea is rather to use multiple copies of $\mathcal{A}'$, one for each term of E'', as depicted in Figure 8.5. More precisely, the automaton $\mathcal{A}''$ for E'' consists of one copy of $\mathcal{A}$ and a copy $\mathcal{A}'_i$ of $\mathcal{A}'$ for every $i \in \mathcal{L}$. First, $\mathcal{A}''$ simulates $\mathcal{A}$ until it reaches some final state f_i of $\mathcal{A}$. Then, if $i \in \mathcal{L}$, transitions starting from the initial state of $\mathcal{A}'_i$ are duplicated from f_i in the same way as previously. The accepting states of $\mathcal{A}''$ consist of $\{f_i \mid i \in \mathcal{K}\}$ and a copy of accepting states in Q' for each $i \in \mathcal{L}$.

For the Kleene star, we assume that $E = G + H$ and $E'' = G^\star \cdot H$. We have $\mathcal{I} = \mathcal{K} \uplus \mathcal{L}$ with $\text{Terms}(G) = \{\!\{E_i \mid i \in \mathcal{K}\}\!\}$ and $\text{Terms}(H) = \{\!\{E_i \mid i \in \mathcal{L}\}\!\}$. Hence, we have $\text{Terms}(E'') = \{\!\{G^\star \cdot E_i \mid i \in \mathcal{L}\}\!\}$. We construct the pebble probabilistic automaton $\mathcal{A}'' = (Q, A, D, I, \Delta'', F'')$ with accepting states $\{f_i \mid i \in \mathcal{L}\}$ and by duplicating transitions exiting from ι, to transitions from the states f_i for $i \in \mathcal{K}$.

Construction for $E'' = x!E$ is done similarly as for the non-probabilistic case: however, as for the previous construction for the rule $(\cdot)$, we need to keep track of the terms of E'', henceforth we must duplicate the final states. More precisely, the terms of E'' are of the form $x!E_i$ for all terms E_i of E. Hence, we construct the automaton $\mathcal{A}'' = (Q \uplus \{\iota''\} \uplus \{f''_i \mid$

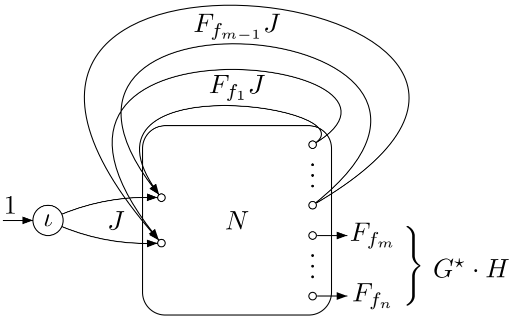


Figure 8.6: Construction for the rule ($\star$), supposing that automaton $\mathcal{A}$, equivalent to expression $E = G + H$, is given in Figure 8.5: the automaton depicted is equivalent to expression $E'' = G^\star \cdot H$.

$i \in \mathcal{I}\}, A, D, I'', \Delta'', F'')$ with a fresh state ι'', unique initial state in the distribution I'', and as many fresh states f_i'' as terms in E. The transitions in-between states of Q are kept in $\mathcal{A}''$, whereas we add drop transitions

$$\Delta''_{\iota'',\iota}(\mathsf{drop}_x) = 1\top$$

and lift transitions

$$\Delta''_{f_i,f_i''}(\mathsf{lift}) = F_{f_i}$$

for every $i \in \mathcal{I}$.

Finally, if E'' is obtained from E via ACD-rules, we have Terms(E'') = Terms(E) so we can keep the same automaton: $\mathcal{A}'' = \mathcal{A}$. □

By Proposition 8.3, the semantics of pebble probabilistic automata only assumes values in $[0, 1]$. This carries over to hybrid probabilistic expressions.

Corollary 8.9. *For every hybrid probabilistic expression E, for every pointed graph $G \in \mathcal{G}(A, D)$ and valuation σ, we have $[\![E]\!](G, \sigma) \in [0, 1]$.*

We turn now to the construction of expressions (HPE) which are equivalent to automata (PPA). We cannot directly follow the same procedure as in the non-probabilistic case (Theorem 4.31). Indeed, we must ensure throughout the proof that we produce expressions in HPE. To this aim, we strongly rely on ACD-rules. The rough idea is again to construct a generalized automaton and remove one by one each state (Brzozowski-McCluskey algorithm), starting with the deepest layers. For the purpose of the presentation, we rather follow a method similar to the McNaughton-Yamada algorithm.

Theorem 8.10. *Let $\mathcal{A} = (Q, A, D, I, \Delta, F)$ be a K-layered pebble probabilistic automaton. We can effectively construct a hybrid probabilistic expression E such that $[\![E]\!](G, \sigma) = [\![\mathcal{A}]\!](G, \sigma)$ for every pair of graph $G \in \mathcal{G}(A, D)$ and valuation σ.*

Proof. As usual, we denote by ℓ the mapping defining the layers of $\mathcal{A}$. For every local configuration $\gamma = (a, \tau, \mathcal{P}, b_i, b_f) \in \Gamma$, we use a macro $\gamma?$ denoting the test formula

$$a? \wedge \tau? \wedge \mathcal{P}? \wedge \alpha_i \wedge \alpha_f$$

with α_i being init? if $b_i = 1$ and $\neg$init? otherwise, and similarly for α_f. It simply tests whether the current configuration verifies the local configuration γ.

For each state $q \in \ell^{-1}(k)$ $(k < K)$, we let

$$\mathsf{End}_q = \sum_{\gamma \in \Gamma} \gamma? \cdot \sum_{\alpha \in \mathrm{Test} \mid \gamma \models \alpha} \ \sum_{q' \in \ell^{-1}(k+1)} \Delta_{q,q'}(\mathsf{lift})(\alpha)$$

be the hybrid probabilistic expression representing the lift transitions starting from state q: it is a hybrid probabilistic expression since

$$\sum_{\alpha\in\mathrm{Test}\mid\gamma\models\alpha}\ \sum_{q'\in\ell^{-1}(k+1)} \Delta_{q,q'}(\mathsf{lift})(\alpha)\in[0,1]$$

by using the fact that $\mathcal{A}$ is a pebble probabilistic automaton, and by using repeatedly rule $(+_{\mathrm{det}})$ over each local configuration[5]. Similarly, when $q\in\ell^{-1}(K)$, we let

$$\mathsf{End}_q=\sum_{\gamma\in\Gamma}\gamma?\cdot\sum_{\alpha\in\mathrm{Test}\mid\gamma\models\alpha}F_q(\alpha)$$

which is a hybrid probabilistic expression, for the same reason.

For each $q\in\ell^{-1}(k)$ $(k\leq K)$, we will construct a hybrid probabilistic expression

$$E_q=\mathsf{End}_q+\sum_{q'\in\ell^{-1}(k)}E_{q,q'}\cdot\mathsf{End}_{q'}$$

with $E_{q,q'}$ that computes the sum of the probabilities of *non-empty* runs starting from state q, ending in state q' and visiting only states in layers $0,\dots,k$. This construction is by induction on $k\in\{0,\dots,K\}$. Within each layer $k\leq K$, we follow usual procedures to translate automata into expressions.

For $q\in\ell^{-1}(k)$ and $X\subseteq\ell^{-1}(k)$ we let

$$\mathsf{End}_q^X=\begin{cases}\mathsf{End}_q & \text{if } q\in X\\ 1 & \text{otherwise.}\end{cases}$$

and we will construct by induction on X a hybrid probabilistic expression

$$E_q^X=\mathsf{End}_q+\sum_{q'\in\ell^{-1}(k)}E_{q,q'}^X\cdot\mathsf{End}_{q'}^X\,.$$

For every $q,q'\in\ell^{-1}(k)$, $[\![E_{q,q'}^X]\!](G,\sigma,v,v')$ is the sum of the probabilities of *non-empty* runs starting from configuration $(G,\sigma,q,v,\varepsilon)$, ending in configuration $(G,\sigma,q',v',\varepsilon)$, with intermediary states either in X or in layers $0,\dots,k-1$. Notice in particular that $E_q=E_q^{\ell^{-1}(k)}$.

We now present the details of the induction over (k,X) (with the lexicographic order): for presentation purpose, we start by first presenting the base case on X, i.e., $X=\emptyset$, for $k=0$ and then $k>0$, followed by the inductive step on X, common for all k.

- Consider first the construction for $X=\emptyset$. Recall that, for each state $q\in\ell^{-1}(k)$, and for every local configuration $\gamma\in\Gamma$, by definition of pebble probabilistic automata, we have

$$\sum_{\alpha\in\mathrm{Test}\mid\gamma\models\alpha}\Big(F_q(\alpha)+\sum_{m\in\mathrm{Actions},q'\in Q}\Delta_{q,q'}(m)(\alpha)\Big)\leq 1\,. \tag{8.1}$$

  - We start with the case $k=0$ and $K=0$. Now, since q is in layer 0, drop actions may be discarded. Moreover, layer 0 is the only layer, so that lift transitions are also discarded. Starting from (8.1) and using $(+_{\mathrm{det}})$ rule repeatedly, we obtain that

$$\sum_{\gamma\in\Gamma}\gamma?\cdot\sum_{\alpha\in\mathrm{Test}\mid\gamma\models\alpha}\Big(F_q(\alpha)+\sum_{d\in D,q'\in\ell^{-1}(0)}\Delta_{q,q'}(d)(\alpha)\Big)$$

[5] Indeed, we use again some semantical equivalence, noticing that if $\gamma\neq\gamma'$ are two different local configurations, test formula $\gamma?\wedge\neg(\gamma'?)$ is equivalent to $\gamma?$.

is a hybrid probabilistic expression. Hence, using ACD-rules and rule $(\cdot)$, expression

$$\begin{aligned} E_q^{\emptyset} = & \sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} F_q(\alpha) \\ & + \sum_{q'\in\ell^{-1}(0)} \sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} \sum_{d\in D} \Delta_{q,q'}(d)(\alpha)\cdot d \end{aligned}$$

is a hybrid probabilistic expression. We conclude by letting

$$E_{q,q'}^{\emptyset} = \sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} \sum_{d\in D} \Delta_{q,q'}(d)(\alpha)\cdot d$$

and noticing that the first term of the sum in $E_q^{\emptyset}$ is equal to End_q and that $\mathsf{End}_{q'}^{\emptyset} = 1$.

- In case $k = 0$ but $K > 0$, layer 0 is not the topmost layer. Henceforth, we will consider lift transitions, rather than the probabilities of acceptance: starting again from (8.1), we obtain that

  $$\sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} \Big(\sum_{q'\in\ell^{-1}(1)} \Delta_{q,q'}(\mathsf{lift})(\alpha) + \sum_{d\in D, q'\in\ell^{-1}(0)} \Delta_{q,q'}(d)(\alpha)\Big)$$

  is a hybrid probabilistic expression. Similarly to the previous case, expression

  $$\begin{aligned} E_q^{\emptyset} = & \sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} \sum_{q'\in\ell^{-1}(1)} \Delta_{q,q'}(\mathsf{lift})(\alpha) \\ & + \sum_{q'\in\ell^{-1}(0)} \sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} \sum_{d\in D} \Delta_{q,q'}(d)(\alpha)\cdot d \end{aligned}$$

  is a hybrid probabilistic expression. We conclude as previously letting

  $$E_{q,q'}^{\emptyset} = \sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} \sum_{d\in D} \Delta_{q,q'}(d)(\alpha)\cdot d\,.$$

- We then consider the case $0 < k = K$. Compared to the case $k = 0$, we also have to consider drop transitions which lead to the lower layer. By induction, we have already constructed expressions $E_r^{\ell^{-1}(k-1)}$ for all states $r \in \ell^{-1}(k-1)$ of the lower layer:

  $$E_r^{\ell^{-1}(k-1)} = \mathsf{End}_r + \sum_{r'\in\ell^{-1}(k-1)} E_{r,r'}^{\ell^{-1}(k-1)} \cdot \mathsf{End}_{r'}\,.$$

  Recall that

  $$\mathsf{End}_r = \sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} \sum_{q'\in\ell^{-1}(k)} \Delta_{r,q'}(\mathsf{lift})(\alpha)\,.$$

  By applying ACD-rules, we obtain that $E_r^{\ell^{-1}(k-1)}$ is equivalent to the hybrid probabilistic expression

  $$\begin{aligned} \sum_{q'\in\ell^{-1}(k)} \Bigg(& \sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} \Delta_{r,q'}(\mathsf{lift})(\alpha) \\ & + \sum_{r'\in\ell^{-1}(k-1)} E_{r,r'}^{\ell^{-1}(k-1)} \cdot \sum_{\gamma\in\Gamma} \gamma? \cdot \sum_{\alpha\in\mathrm{Test}|\gamma\models\alpha} \Delta_{r',q'}(\mathsf{lift})(\alpha)\Bigg)\,. \end{aligned}$$

We may use the rule $(x!)$, as well as ACD-rules again to obtain the hybrid probabilistic expression

$$G_r = \sum_{q' \in \ell^{-1}(k)} G_{r,q'}$$

$$G_{r,q'} = x!\left(\sum_{\gamma \in \Gamma} \gamma? \cdot \sum_{\alpha \in \mathrm{Test} | \gamma \models \alpha} \Delta_{r,q'}(\mathsf{lift})(\alpha) \right.$$
$$\left. + \sum_{r' \in \ell^{-1}(k-1)} E_{r,r'}^{\ell^{-1}(k-1)} \cdot \sum_{\gamma \in \Gamma} \gamma? \cdot \sum_{\alpha \in \mathrm{Test} | \gamma \models \alpha} \Delta_{r',q'}(\mathsf{lift})(\alpha) \right).$$

Then, if $q \in \ell^{-1}(k)$, (8.1) can be rewritten (removing the lift transitions since we are in the topmost layer) as

$$\sum_{\alpha \in \mathrm{Test} | \gamma \models \alpha} F_q(\alpha) +$$
$$\sum_{\alpha \in \mathrm{Test} | \gamma \models \alpha} \Big(\sum_{d \in D, q' \in \ell^{-1}(k)} \Delta_{q,q'}(d)(\alpha) + \sum_{\substack{x \in \mathrm{Var}(\mathcal{A}) \\ r \in \ell^{-1}(k-1)}} \Delta_{q,r}(\mathsf{drop}_x)(\alpha) \Big) \in [0,1]$$

Using rule $(+_{\mathrm{det}})$ repeatedly to concatenate the tests $\gamma?$ to the left, and rule $(\cdot)$ to concatenate the previous G_r expression on the right of the drop transition, as well as the directions d, we obtain that

$$\sum_{\gamma \in \Gamma} \gamma? \cdot \sum_{\alpha \in \mathrm{Test} | \gamma \models \alpha} F_q(\alpha)$$
$$+ \sum_{\gamma \in \Gamma} \gamma? \cdot \sum_{\alpha \in \mathrm{Test} | \gamma \models \alpha} \Big(\sum_{\substack{d \in D \\ q' \in \ell^{-1}(k)}} \Delta_{q,q'}(d)(\alpha) \cdot d + \sum_{\substack{x \in \mathrm{Var}(\mathcal{A}) \\ r \in \ell^{-1}(k-1)}} \Delta_{q,r}(\mathsf{drop}_x)(\alpha) \cdot G_r \Big)$$

is a hybrid probabilistic expression. Finally, using ACD-rules a last time, we may rewrite it as

$$\sum_{\gamma \in \Gamma} \gamma? \cdot \sum_{\alpha \in \mathrm{Test} | \gamma \models \alpha} F_q(\alpha)$$
$$+ \sum_{q' \in \ell^{-1}(k)} \sum_{\gamma \in \Gamma} \gamma? \cdot \sum_{\alpha \in \mathrm{Test} | \gamma \models \alpha} \Big(\sum_{d \in D} \Delta_{q,q'}(d)(\alpha) \cdot d + \sum_{\substack{x \in \mathrm{Var}(\mathcal{A}) \\ r \in \ell^{-1}(k-1)}} \Delta_{q,r}(\mathsf{drop}_x)(\alpha) \cdot G_{r,q'} \Big)$$

We conclude as previously by letting

$$E_{q,q'}^{\emptyset} = \sum_{\gamma \in \Gamma} \gamma? \cdot \sum_{\alpha \in \mathrm{Test} | \gamma \models \alpha} \Big(\sum_{d \in D} \Delta_{q,q'}(d)(\alpha) \cdot d$$
$$+ \sum_{\substack{x \in \mathrm{Var}(\mathcal{A}) \\ r \in \ell^{-1}(k-1)}} \Delta_{q,r}(\mathsf{drop}_x)(\alpha) \cdot G_{r,q'} \Big).$$

– The last case, $0 < k < K$, requires to combine the two last cases: indeed, contrary to the case $k = K$, lift transitions rather than probabilities of acceptance have to be taken into account.

- We turn now to the induction step on X, which is common for every layer k. Let $X \subseteq \ell^{-1}(k)$ and $r \in \ell^{-1}(k) \setminus X$. By induction, we assume that hybrid probabilistic expressions E_q^X have been constructed for all $q \in \ell^{-1}(k)$. We construct $E_q^{X\cup\{r\}}$.
  We have $E_r^X = \mathsf{End}_r + \sum_{q' \in \ell^{-1}(k)} E_{r,q'}^X \cdot \mathsf{End}_{q'}^X \in \text{HPE}$ and $\mathsf{End}_r^X = 1$ since $r \notin X$. With rule $(\star)$ we get
  $$G_r^X = \left(E_{r,r}^X\right)^\star \cdot \Big(\mathsf{End}_r + \sum_{q' \in \ell^{-1}(k)\setminus\{r\}} E_{r,q'}^X \cdot \mathsf{End}_{q'}^X\Big) \in \text{HPE}\,.$$
  Now, $E_q^X = \mathsf{End}_q + \sum_{q' \in \ell^{-1}(k)} E_{q,q'}^X \cdot \mathsf{End}_{q'}^X \in \text{HPE}$ and $\mathsf{End}_r^X = 1$. Using rule $(\cdot)$, we can plug G_r^X after $E_{q,r}^X$ and we obtain the hybrid probabilistic expression
  $$\begin{aligned} E_q^{X\cup\{r\}} &= \mathsf{End}_q + E_{q,r}^X \cdot G_r^X + \sum_{q' \in \ell^{-1}(k)\setminus\{r\}} E_{q,q'}^X \cdot \mathsf{End}_{q'}^X \\ &= \mathsf{End}_q + E_{q,r}^X \cdot \left(E_{r,r}^X\right)^\star \cdot \mathsf{End}_r + \sum_{q' \in \ell^{-1}(k)\setminus\{r\}} \left(E_{q,q'}^X + E_{q,r}^X \cdot \left(E_{r,r}^X\right)^\star \cdot E_{r,q'}^X\right) \cdot \mathsf{End}_{q'}^X \\ &= \mathsf{End}_q + \sum_{q' \in \ell^{-1}(k)} E_{q,q'}^{X\cup\{r\}} \cdot \mathsf{End}_{q'}^{X\cup\{r\}} \end{aligned}$$
  where the last equality uses $\mathsf{End}_r^{X\cup\{r\}} = \mathsf{End}_r$ and $\mathsf{End}_{q'}^{X\cup\{r\}} = \mathsf{End}_{q'}^X$ if $q' \in \ell^{-1}(k) \setminus \{r\}$.

Finally, the HPE $\sum_{q \in \ell^{-1}(K)} I_q \cdot E_q$ is equivalent to the automaton $\mathcal{A}$ which concludes the proof. □

8.4 What it Implies for Rabin Probabilistic Automata

We finish this chapter by some corollaries of the previous result in terms of classical probabilistic finite automata as introduced and studied in [Rab63, Paz71], recognizing finite words. A *probabilistic finite automaton* over alphabet A is usually a tuple $\mathcal{A} = (Q, I, \mathbb{P}, F)$ where Q is the finite set of states, $I \in [0,1]^Q$ is an initial probability distribution, $F \in [0,1]^Q$ is a column vector mapping every state to its probability of acceptance, and $\mathbb{P} : (A \to [0,1])^{Q\times Q}$ is a matrix that assigns a probability to every pair of states, and every letter of the alphabet, verifying
$$\sum_{q' \in Q} \mathbb{P}_{q,q'}(a) \leq 1$$
for all $(q, a) \in Q \times A$.

A run of $\mathcal{A}$ over $w = a_0 \cdots a_{n-1} \in A^+$ is a sequence of states $\rho = q_0 q_1 \cdots q_n$. In particular, after *reading* letter a_{n-1} the automaton ends without any letter to read, forced to accept or reject the word, i.e., giving it a probability of acceptance[6]. For such a run ρ, we set
$$\mathbb{P}(\rho) = I_{q_0} \times \left(\prod_{i=0}^{n-1} \mathbb{P}_{q_i,q_{i+1}}(a_i)\right) \times F_{q_n}\,.$$

The semantics of the probabilistic automaton $\mathcal{A}$ is the formal power series $[\![\mathcal{A}]\!] : A^+ \to [0,1]$ given by
$$[\![\mathcal{A}]\!](w) = \sum_{\rho} \mathbb{P}(\rho)$$

[6] This definition is not completely consistent with the rest of this manuscript where runs are sequences of configurations, each of these being attached to a position of the word.

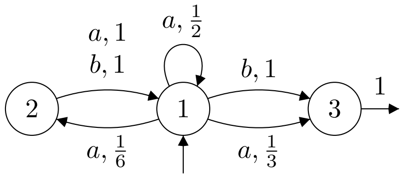


Figure 8.7: A probabilistic finite automaton

where the sum ranges over all runs ρ over w. It is well-known that probabilistic automata associate to every word a probability, i.e., $[\![\mathcal{A}]\!](w) \in [0,1]$ for every word $w \in A^+$.

Notice that this model is not formally a special case of the pebble probabilistic automata introduced previously. Indeed, a pebble probabilistic automaton can only *test* the last letter of a word by making either a left move, by dropping a pebble or by accepting. In the contrary, finite-state automata – and thus probabilistic finite automata too – may read the last letter and access a virtual position on the right of the word, forced to accept (or reject) when they reach this deadlock position. As done in Example 4.2, we may however very easily translate every probabilistic finite automaton into a pebble probabilistic automaton by merging the final acceptance probability with the last transition.

Example 8.11. We depict in Figure 8.7 as a classical probabilistic automaton the pebble probabilistic automaton depicted in Figure 8.2. Transitions with probability 0 are omitted. You may observe that state 3 is not used in the pebble probabilistic automaton since it is not neccessary anymore after the transformation explained before this example. ■

Theorems 8.8 and 8.10 indeed permit to define a fragment of hybrid probabilistic expressions equivalent to these probabilistic automata. Again, we use a as a shortcut for the expression $a?{\rightarrow}$. However, to stop the computation, we use test formula final? signifying that the expression has reached the end of the word.

Definition 8.12. We let PE be the class of *probabilistic expressions*, fragment of WE, defined by the following rules:

- the atom rules:

$$(p)\ \frac{p \in [0,1]}{p \in \mathrm{PE}} \qquad (a)\ \frac{a \in A}{a \in \mathrm{PE}} \qquad (\mathrm{final?})\ \frac{}{\mathrm{final?} \in \mathrm{PE}}$$

- the inductive rules:

$$(+_{\mathrm{act}})\ \frac{E_a \in \mathrm{PE}\ (\forall a \in A) \quad p \in [0,1]}{\mathrm{final?}p + \sum_{a \in A} a \cdot E_a \in \mathrm{PE}} \qquad (+_{\mathrm{prob}})\ \frac{E_1 \in \mathrm{PE} \quad E_2 \in \mathrm{PE} \quad p \in [0,1]}{p \cdot E_1 + (1-p) \cdot E_2 \in \mathrm{PE}}$$

$$(\cdot_{\mathrm{weak}})\ \frac{E_1 \in \mathrm{PE} \quad E_2 \in \mathrm{PE}}{E_1 \cdot E_2 \in \mathrm{PE}} \qquad (\star)\ \frac{E_1 + E_2 \in \mathrm{PE}}{E_1^\star \cdot E_2 \in \mathrm{PE}}$$

- the following associativity, commutativity and distributivity rules (later denoted as ACD-rules):

$$\begin{array}{llcl}
\mathbf{A}_+ & E + (G + H) \in \mathrm{PE} & \longleftrightarrow & (E+G)+H \in \mathrm{PE} \\
\mathbf{C}_+ & E + G \in \mathrm{PE} & \longleftrightarrow & G + E \in \mathrm{PE} \\
\mathbf{A}_. & E \cdot (G \cdot H) \in \mathrm{PE} & \longleftrightarrow & (E \cdot G) \cdot H \in \mathrm{PE} \\
\mathbf{D}_. & E \cdot (G + H) \in \mathrm{PE} & \longleftrightarrow & E \cdot G + E \cdot H \in \mathrm{PE} \\
\mathbf{D}_. & (E + G) \cdot H \in \mathrm{PE} & \longleftrightarrow & E \cdot H + G \cdot H \in \mathrm{PE}
\end{array}$$

■

There are two *guarded* sums. The first one ($+_{\text{act}}$) is deterministic and guarded by the letter to be read. The second one ($+_{\text{prob}}$) is probabilistic. Probabilistic expressions clearly are a subset of hybrid probabilistic expressions. However, contrary to hybrid probabilistic expressions, probabilistic expressions are forced to *read* the last letter (as for probabilistic finite automata). Henceforth, to define easily their semantics, we may add a last position on the right of the word, labeled with a fresh symbol $\lhd$, and define the semantics $[\![E]\!](w)$ of the probabilistic expression $E \in \mathrm{PE}$ over a word w, as the probability $[\![E]\!](w\lhd)$ where E is seen as an expression of HPE here. For instance, the probabilistic automaton in Figure 8.7 may be translated into probabilistic expression $\left[\frac{1}{6}a(a+b) + \frac{1}{2}a\right]^\star \cdot (\frac{1}{3}a+b) \cdot \mathrm{final?}$: in particular, the last position labeled with $\lhd$, is never tested by probabilistic expression. Notice that the concatenation rule ($\cdot$) has been weakened to ($\cdot_{\text{weak}}$) in this definition. In this simpler case, the stronger rule can indeed be derived from the weaker one, which we do not know for the general case:

Lemma 8.13. *Let $E \in \mathrm{PE}$ with $\mathrm{Terms}(E) = \{\!\{E_i \mid i \in \mathcal{I}\}\!\}$. Let $F_i \in \mathrm{PE}$ for each $i \in \mathcal{I}$. We have*

$$\overline{E} = \sum_{i\in\mathcal{I}} E_i \cdot F_i \in \mathrm{PE}\,.$$

Proof. We proceed by structural induction over the rules generating E. This is trivial for atoms and also for ACD-rules since they preserve the multiset of terms. Let $G, H \in \mathrm{PE}$ with $\mathrm{Terms}(G) = \{\!\{G_j \mid j \in \mathcal{J}\}\!\}$ and $\mathrm{Terms}(H) = \{\!\{H_k \mid k \in \mathcal{K}\}\!\}$.

Consider $E = p{\cdot}G{+}(1{-}p){\cdot}H$. Then, $\mathrm{Terms}(E) = \{\!\{p{\cdot}G_j \mid j \in \mathcal{J}\}\!\} \uplus \{\!\{(1{-}p){\cdot}H_k \mid k \in \mathcal{K}\}\!\}$. Hence we have $\mathcal{I} = \mathcal{J} \uplus \mathcal{K}$, $E_i = p \cdot G_i$ for $i \in \mathcal{J}$ and $E_i = (1-p) \cdot H_i$ for $i \in \mathcal{K}$. By induction, we have $\overline{G} = \sum_{i\in\mathcal{J}} G_i{\cdot}F_i \in \mathrm{PE}$ and $\overline{H} = \sum_{i\in\mathcal{K}} H_i{\cdot}F_i \in \mathrm{PE}$. Therefore, $p{\cdot}\overline{G}{+}(1{-}p){\cdot}\overline{H} \in \mathrm{PE}$. This expression can be rewritten into $\overline{E}$ with ACD-rules.

The proof is similar for $E = \mathrm{final?}p + \sum_{a\in A} a \cdot E_a$.

Consider $E = G \cdot H$ so that $\mathcal{I} = \mathcal{J} \times \mathcal{K}$ and $E_{(j,k)} = G_j \cdot H_k$ for $(j,k) \in \mathcal{I}$. By induction, for every $j \in \mathcal{J}$, we get $\overline{H}_j = \sum_{k\in\mathcal{K}} H_k \cdot F_{(j,k)} \in \mathrm{PE}$. Again by induction we obtain $\sum_{j\in\mathcal{J}} G_j \cdot \overline{H}_j \in \mathrm{PE}$. This expression can be rewritten into $\overline{E}$ with ACD-rules.

Consider now $E = G^\star \cdot H$ assuming $G + H \in \mathrm{PE}$ instead of $G, H \in \mathrm{PE}$. We have $\mathcal{I} = \mathcal{K}$ and $E_i = G^\star \cdot H_i$ for $i \in \mathcal{I}$. Applying the induction hypothesis to the expression $G + H$ with $F_j = 1$ for $j \in \mathcal{J}$, we obtain $\sum_{j\in\mathcal{J}} G_j + (\sum_{k\in\mathcal{K}} H_k \cdot F_k) \in \mathrm{PE}$. From the star rule we deduce that $(\sum_{j\in\mathcal{J}} G_j)^\star \cdot (\sum_{k\in\mathcal{K}} H_k \cdot F_k) \in \mathrm{PE}$. Using ACD-rules, this can be rewritten into $\overline{E} = \sum_{i\in\mathcal{I}} G^\star \cdot H_i \cdot F_i$. □

Using special cases of the proofs of the main general theorem, it is possible to show:

Theorem 8.14. *Probabilistic finite automata and probabilistic expressions in* PE *recognize the same series of finite words.*

With Theorem 8.14, decidability of the equivalence problem for probabilistic automata carries over to probabilistic expressions (provided the probabilities in an expression are rational numbers), whereas their threshold problem is undecidable (as this problem is undecidable for automata, as originally proved in [Paz71]). We could also state further undecidability results about isolated cutpoints using, e.g., results of [BMT77].

Corollary 8.15.

1. *The equivalence problem for probabilistic expressions is decidable: given probabilistic expressions E and F of* PE*, does $[\![E]\!] = [\![F]\!]$ hold?*
2. *The threshold problem for probabilistic expressions is undecidable: given an alphabet A, a probabilistic expression $E \in \mathrm{PE}$ and $0 < p < 1$, is there a word $w \in A^+$ such that $[\![E]\!](w) \geq p$?*

CHAPTER **9**

Implementation: QuantiS tool

Quanti, Quantos, Quantorum, Quantis, Quantis.

Plural declension of latine word for “how much”.

We describe in this last chapter an implementation of some algorithms presented in this manuscript. It is available in tool called QuantiS. Further informations and the instruction to install this tool may be found on the webpage `http://www.lsv.ens-cachan.fr/Software/quantis/`.

9.1 Objective and Implementation Details

The goal of this tool is to permit the evaluation of quantitative specifications over finite words. Henceforth, we focus on the specialized algorithms presented in Theorem 5.5. We also use the translation presented in Theorem 4.31 from hybrid weighted expressions to layered pebble weighted automata.

The tool has been written in Objective Caml[1]. The first released version of QuantiS permits the evaluation of a hybrid weighted expression over a word in the semiring $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$. The syntax to use the tool is the following:

```
QuantiS -expr <expression> -word <non-empty word> [-timer]
```

The expression may be written in plain text[2], since a parser deduces from it the formal hybrid weighted expression. For instance,

```
QuantiS -expr '(x!((2..->)^*).->)^*' -word 'aaa'
```

evaluates the expression $(x!((2 \cdot \rightarrow)^\star) \cdot \rightarrow)^\star$ over the word aaa, outputting the weight 16:

[1] `http://caml.inria.fr`

[2] Notice that weights being floating numbers, they must have a dot: e.g., the real number 2 may be written `2.`, whereas the real number 0.2 may be written either `0.2` or `.2`. Also, the Kleene operators are written `^*` and `^+`, in order to differentiate them with the sum of two expressions.

```
Evaluation of expression :
     (x!((2..->)^*).->)^*
on string :
     aaa
Result:  16.
```

We allow the use of a timer with the `-timer` option. For instance,

```
QuantiS -expr '(x!((2..->)^*).->)^*' -word 'aaa' -timer
```

outputs

```
Evaluation of expression :
     (x!((2..->)^*).->)^*
on string :
     aaa
Result:  16.
Time:  0.000266s
```

The use of a timer permits to observe the time of execution of the evaluation algorithm (the transformation from the expression to the automaton is not taken into account here).

9.2 Experiments

In the archive containing the tool, the script file tests.sh of directory /examples contains a list of examples of usage of the tool. Here are the expressions we may find in this list:

- $(2\cdot a? + b?)\cdot \rightarrow \cdot (2\cdot b? + 3\cdot c?)$;
- $a?\cdot x!(2\cdot \rightarrow^\star)$, in order to test the use of a variable;
- $(x!(2\cdot \rightarrow^\star)\cdot \rightarrow)^\star$, mapping a word w to $2^{(|w|-1)^2}$;
- $\rightarrow^+ a?x!\big((\neg x?\rightarrow)^\star b?(\neg x?\rightarrow)^+ c?\leftarrow^+ d?\rightarrow^+\big)\rightarrow^\star$, used in Example 4.42;
- $(x!((x!((2\cdot\rightarrow)^\star)\cdot\rightarrow)^\star)\cdot\rightarrow)^\star$, testing the reusability of variables, and mapping a word w to $2^{(|w|-1)^3}$. In particular, over the word a^{10}, the tool outputs the value 2.82×10^{219}: even if this value is large, the computation time is only of roughly 8 milliseconds, instead of 0.3 milliseconds over the word aa;
- $(\neg\text{final}?\cdot(0.25\cdot\rightarrow + 0.75\cdot\leftarrow))^\star\cdot\text{final}?$, inspired from Examples 4.7 and 8.4. Figure 9.1 depicts the execution time needed to evaluate the expression over a word of the form a^n as a function of n. We may clearly observe the linear dependency in the size n of the word, as observed in Theorem 5.5 (here the number p of variables that may be dropped by the automaton equivalent to E is 0).
- $(\rightarrow^\star\cdot\leftarrow^\star)^n$, concatenating n *zigzag* expressions. This *a priori* useless expression permits to observe the behavior of the tool over expressions of increasing size. Figure 9.2 depicts the execution time needed to evaluate the expression over a word of length 1000 as a function of the parameter n. We may guess a cubic dependency in n, as observed in Theorem 5.5.
- expressions E_n constructed by induction letting for $n\geq 1$:

$$\begin{cases} E_0 = \rightarrow^* \\ E_n = \rightarrow^* \cdot x_n!(\rightarrow^\star\cdot\leftarrow^\star\cdot x_n?\cdot E_{n-1})\cdot\rightarrow^\star \end{cases}$$

  Expression E_n uses n different variables, henceforth, this example permits to observe the behavior of the tool over expressions with increasing number of variables. Figure 9.3 depicts the execution time needed to evaluate E_n over a word of length 3 as a function of the parameter n: notice the use of a logarithmic scale for the vertical axis. We may see an exponential dependency in n, as observed again in Theorem 5.5.

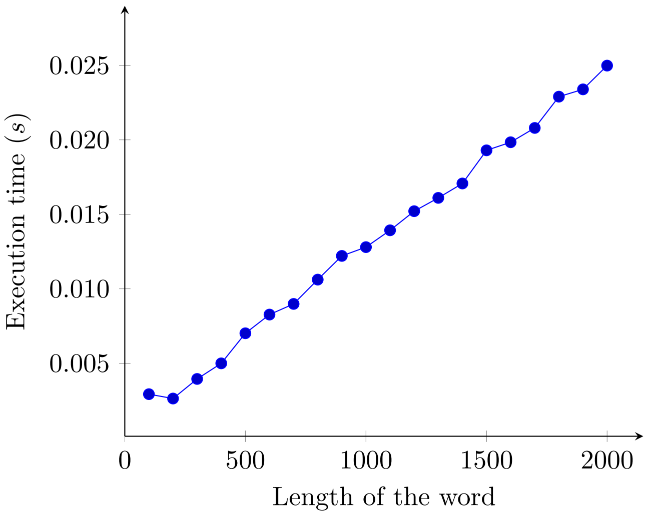


Figure 9.1: Plot of the execution time of the tool over expression $(\neg\text{final?} \cdot (0.25 \cdot \rightarrow + 0.75 \cdot \leftarrow))^{\star} \cdot \text{final?}$ as a function of the length of the word

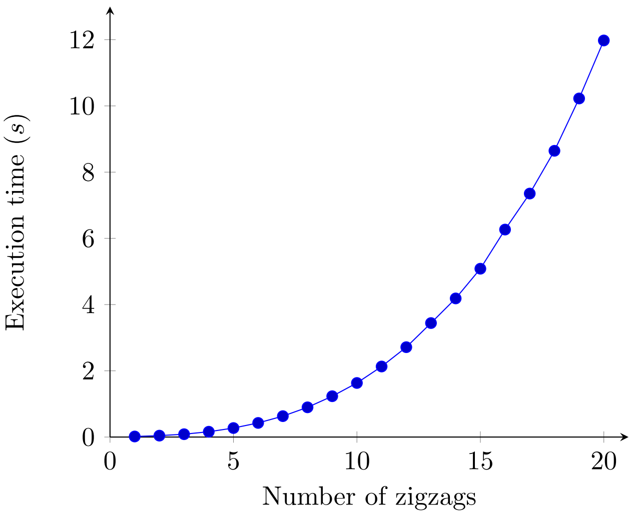


Figure 9.2: Plot of the execution time of the tool over zigzag expressions as a function of the size of the expression

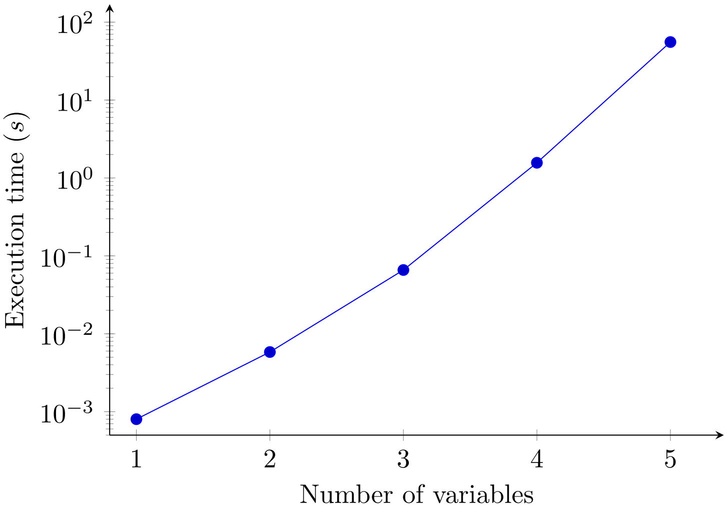


Figure 9.3: Plot of the execution time of the tool over expressions using n variables as a function of n

CHAPTER 10

Perspectives

The aim of this manuscript was to study high-level formalisms to specify quantitative properties, and to use automata techniques in order to evaluate them efficiently over general classes of graphs. The process of specification and verification in the quantitative field requires a three-dimensional diagram: the first dimension consists of selecting the set of weights – semirings in all this manuscript – in which the desired quantitative result lies, the second contains the classes of graphs representing the model to be verified – finite words, trees, nested words, pictures, traces, or abstract sets of graphs, for instance – and the last dimension permits to choose the language of specification adequate to model faithfully the property of interest – we studied hybrid weighted expressions, weighted first order logic with transitive closure and other fragments of weighted monadic second order logic, hybrid weighted propositional dynamic logic, and probabilistic special cases.

Along the manuscript, we already presented some short-term perspectives, that are the basis for future works. For instance, in Chapter 7, we briefly presented high-level formalisms (expressions and logics) able to recognize exactly the quantitative behaviors of one-way weighted automata (with and without pebbles). In this case, where the semiring must not be continuous, we may try to apply the results to various areas. However, this application process may require to further refine the high-level specification languages, in order to make them more practical. In this perspective, the chop expressions that we introduced in Definition 3.12 can be seen as a first step to reduce the usage of variables in the high-level frameworks. For instance, we believe that *one-way chop expressions*, i.e., where we restrict the use of directions to forward directions (over ordered pointed graphs), would be equivalent to one-way pebble weighted automata in the case of finite words: whereas the translation from these chop expressions to automata is made possible by Theorem 3.13, the reverse translation would certainly hold in this very specialized case. In other cases, it would be interesting to find some strong enough specification formalisms not using variables, still able to generate a large class of quantitative behaviors: it appears, at the end of this manuscript, that a compromise always seems to be done between the use of variables (sometimes not handy) and the potential power of the specifications.

In Chapter 5, we presented several specialized algorithms for evaluating a layered pebble weighted automaton over words, trees and nested words. Whereas we considered the case of strongly layered automata for words, we did not explore it in the setting of general classes of graphs, trees or nested words. We believe it is of the greatest interest: in particular, the case of strongly layered automata is very useful if the high-level specification formalism we consider is directly translatable into a strongly layered pebble weighted automaton, as it is the case, e.g., for weighted propositional dynamic logic in Theorem 6.21, or probabilistic linear temporal logic as we briefly showed in Example 6.16.

In the probabilistic setting of Chapter 8, we only considered expressions as a high-level formalism. It could be interesting to also find a syntactical fragment of logical formalisms translatable into the model of probabilistic automata we considered. For instance, such probabilistic logics equivalent to probabilistic automata (without pebbles) have been considered in [Wei12], for finite and infinite words.

We conclude this manuscript with some broader perspectives. First, the structures we consider, finite graphs with labeled vertices and edges, could be further generalized. For instance, we may consider adding *data* in the models, i.e., consider the labeling of vertices as a pair composed of a letter from a finite alphabet and a data value from an infinite data domain. This is of great interest when considering nested words or trees as a model of databases. For instance, in Example 6.4, we used a formula $\mathsf{Match}(x, y)$ supposed to match two nodes of an XML document holding similar data: in the setting of logics over data words for instance, this would be a new atom of the logic. Notice that hybrid propositional dynamic logics are close to logics used in this context, like freeze linear temporal logic [DL06]. Moreover, pebble automata have also been studied in the context of models with data [NSV04]. Pebbles are then used both as a way to mark a vertex of the graph, but also to store the data value of this vertex in order to compare it with other data values. It would be interesting to study the interleaving of weighted specifications and the data part of XML documents: can we extend the translation results and the evaluation algorithms presented in this manuscript for instance?

Another way to generalize the structures we consider might be to add weights in the models (notice that this can be seen as a special case of the previous one, since weights are data from an infinite data domain being the semiring). Indeed, our graphs currently do not handle any weights: only the specification generates some weights. In certain applications, like in the probabilistic setting, it is important that the model contains probabilities too. We believe that this generalization could be handled by the following method. First, we must consider a new operator (both in the high-level formalisms, and the automata) in order to replace a constant weight of the semiring, by the weight in the graph. Afterwards, we believe that it is easy to modify the translations from specification formalisms to automata, as well as the evaluation algorithms in order to take into account this new operator.

A second direction of further research lies in the evaluation problem itself. Indeed, with respect to usual model-checking problems, we consider an instance where we check one single fixed structure against a specification. A natural question is then the following: considering a model generating some structures, a finite-state automaton generating finite words for example, can we evaluate the specification over all the generated structures, and aggregate the results? The aggregation operator should be able to take as an input a (possibly infinite) set of weights from the semiring, and generate a weight as output.

For instance, considering the semiring of positive real numbers equipped with the usual addition and product, we may consider an aggregation operator being a supremum (or an infimum) of the set of weights. This way, we would be able to check that *all* the specified values of the generated structures are lower (or greater) than a given threshold: this becomes closer to the traditional point of view of probabilistic computation tree logic (see [BK08], for an overview) or the quantitative languages studied in [CDH08].

The frontier between decidability and undecidability is very thin when we consider this extension: for instance, as we recalled in Chapter 8, the threshold problem of probabilistic Rabin automata – whether or not all probabilities generated by such an automaton are greater than a constant probability (that can be seen as a special case of the previous problem where the finite-state automaton generates all words) – is undecidable [Paz71].

Another way to extend the model-checking problem is to rather consider bisimulation tasks, i.e., comparing two models to check whether they can simulate each other, and hence behave the same in every context. This setting has been generalized recently in [FLT10]: authors consider some *distances* between weighted automata models. We believe that this direction is promising and would be worth to be considered with the broader and more powerful models of automata introduced in this manuscript.

Last, but not least, we would like to apply the techniques developped in this manuscript to several areas, like natural language processing or for the quantitative verification of softwares. This requires a richer implementation of the algorithms presented. The tool QuantiS developped, and presented in Chapter 9, must be seen as a first prototype in this context: the latter is focused on a single semiring and only for words. We would like to extend this tool to richer semirings and graph structures, in a modularised way, similar to the Vaucanson platform [LRGS04].

List of Figures

List of Tables

References

[1] Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Pebble weighted automata and transitive closure logics. In Samson Abramsky, Friedhelm Meyer auf der Heide, and Paul Spirakis, editors, *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP'10) – Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 587–598, Bordeaux, France, July 2010. Springer.

[2] Paul Gastin and Benjamin Monmege. Adding pebbles to weighted automata. In Nelma Moreira and Rogério Reis, editors, *Proceedings of the 17th International Conference on Implementation and Application of Automata (CIAA'12)*, volume 7381 of *Lecture Notes in Computer Science*, pages 28–51, Porto, Portugal, July 2012. Springer.

[3] Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. A probabilistic Kleene theorem. In Madhavan Mukund and Supratik Chakraborty, editors, *Proceedings of the 10th International Symposium on Automated Technology for Verification and Analysis (ATVA'12)*, Lecture Notes in Computer Science, pages 400–415, Thiruvananthapuram, India, October 2012. Springer.

[4] Benedikt Bollig, Paul Gastin, and Benjamin Monmege. Weighted specifications over nested words. In Frank Pfenning, editor, *Proceedings of the 16th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'13)*, volume 7794 of *Lecture Notes in Computer Science*, pages 385–400, Roma, Italy, March 2013. Springer.

Bibliography

[ABM00] Carlos Areces, Patrick Blackburn, and Maarten Marx. The Computational Complexity of Hybrid Temporal Logics. *Logic Journal of IGPL*, 8(5):653–679, 2000.

[AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[ADD$^+$13] Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Ragothaman, and Yifei Yuan. Regular Functions and Cost Register Automata. In *28th Annual Symposium on Logic in Computer Science (LICS'13)*. IEEE Computer Society Press, 2013. Invited paper.

[AK11] Shaull Almagor and Orna Kupferman. Max and Sum Semantics for Alternating Weighted Automata. In *Automated Technology for Verification and Analysis (ATVA'11)*, volume 6996 of *Lecture Notes in Computer Science*. Springer, 2011.

[AM06] Cyril Allauzen and Mehryar Mohri. A Unified Construction of the Glushkov, Follow, and Antimirov Automata. In *Proceedings of the 31st international conference on Mathematical Foundations of Computer Science (MFCS'06)*, volume 4162 of *Lecture Notes in Computer Science*, pages 110–121. Springer-Verlag, 2006.

[Ans90] Marcella Anselmo. Two-Way Automata with Multiplicity. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 88–102. Springer, 1990.

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling.* Prentice-Hall, 1972.

[BH67] Manuel Blum and Carl Hewitt. Automata on a 2-Dimensional Tape. In *Proceedings of the 8th Annual Symposium on Switching and Automata Theory (SWAT'67)*, 1967.

[BK93] Anne Brüggeman-Klein. Regular Expressions into Finite Automata. *Theoretical Computer Science*, 120:197–213, 1993.

[BK01] Peter Buchholz and Peter Kemper. Quantifying the Dynamic Behavior of Process Algebras. In *Process Algebra and Probabilistic Methods. Performance Modelling and Verification*, volume 2165 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2001.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* MIT Press, 2008.

[BKW00] Anne Brüggeman-Klein and Derick Wood. Caterpillars: A Context Specification Technique. *Markup Languages*, 2(1):81–106, 2000.

[BM63] John A. Brzozowski and Edward J. McCluskey. Signal Flow Graph Techniques for Sequential Circuit State Diagrams. *IEEE Transactions on Electronic Computers*, 12(9):67–76, 1963.

[BMT77] Alberto Bertoni, Giancarlo Mauri, and Mauro Torelli. Some Recursive Unsolvable Problems Relating to Isolated Cutpoints in Probabilistic Automata. In *Proceedings of the Fourth Colloquium on Automata, Languages and Programming*, pages 87–94, 1977.

[Boj07] Mikołaj Bojańczyk. Forest Expressions. In *Computer Science Logic: 21st International Workshop (CSL'07)*, volume 4646 of *Lecture Notes in Computer Science*, 2007.

[Boj08] Mikołaj Bojańczyk. Tree-Walking Automata. In *Language and Automata Theory and Applications (LATA'08)*, volume 5196 of *Lecture Notes in Computer Science*. Springer, February 2008.

[Bou64] Nicolas Bourbaki. *Algèbre commutative, Chapitre V.* Hermann, 1964.

[BPX$^+$07] T. Brants, A.C. Popat, P. Xu, F.J. Och, and J. Dean. Large Language Models in Machine Translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, 2007.

[BR10] Jean Berstel and Christophe Reutenauer. *Noncommutative Rational Series With Applications.* Cambridge University Press, 2010.

[BS86] Gérard Berry and Ravi Sethi. From Regular Expressions to Deterministic Automata. *Theoretical Computer Science*, 48:117–126, 1986.

[BS12] Laura Bozzelli and César Sánchez. Visibly Rational Expressions. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'12)*, volume 18 of *LIPIcs*, pages 211–223. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[BSSS06] Mikołaj Bojańczyk, Mathias Samuelides, Thomas Schwentick, and Luc Segoufin. Expressive Power of Pebble Automata. In *Proceedings of the 33rd international conference on Automata, Languages and Programming - Volume Part I (ICALP'06)*, volume 4051 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 2006.

[Büc59] J. Richard Büchi. Weak Second-Order Arithmetic and Finite Automata. Technical report, University of Michigan, 1959.

[Bud78] Lothar Budach. Automata and Labyrinths. *Mathematische Nachrichten*, 86:195–282, 1978.

[CCH$^+$05] Arindam Charkrabarti, Krishnendu Chatterjee, Thomas A. Henzinger, Orna Kupferman, and Rupak Majumdar. Verifying Quantitative Properties Using Bound Functions. In *Proceedings of the 13 IFIP WG 10.5 international conference on Correct Hardware Design and Verification Methods (CHARME'05)*, volume 3725 of *Lecture Notes in Computer Science*, pages 50–64, 2005.

[CDE$^+$10] Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-Payoff Automaton Expressions. In *Proceedings of the 21st international conference on Concurrency theory (CONCUR'10)*, volume 6269 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2010.

[CDH08] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative Languages. In *Proceedings of the 22nd International Workshop: Computer Science Logic (CSL'08)*, volume 5213 of *Lecture Notes in Computer Science*, pages 385–400. Springer, 2008.

[CDH09] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating Weighted Automata. In *Proceedings of the 17th International Conference on Fundamentals of computation theory (FCT'09)*, volume 5699 of *Lecture Notes in Computer Science*, pages 3–13, 2009.

[CE11] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic-Second-Order Logic, a Language Theoretic Approach.* Cambridge University Press, 2011.

[CF03] Pascal Caron and Marianne Flouret. From Glushkov WFAs to Rational Expressions. In *Proceedings of the 7th International Conference on Developments in Language Theory (DLT'03)*, volume 2710 of *Lecture Notes in Computer Science*, pages 183–193, 2003.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.

[CLDG$^+$08] Hubert Comon-Lundh, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications.* 2008.

[CMR06] Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. On the Computation of Some Standard Distances Between Probabilistic Automata. In *Proceedings of the 11th International Conference on Implementation and Application of Automata (CIAA'06)*, volume 4094 of *Lecture Notes in Computer Science*, pages 137–149. Springer, 2006.

[Col09] Thomas Colcombet. The Theory of Stabilisation Monoids and Regular Cost Functions. In *Proceedings of the 36th Internatilonal Colloquium on Automata, Languages and Programming: Part II (ICALP'09)*, number 5556 in Lecture Notes in Computer Science, pages 139–150. Springer, 2009.

[Con71] John Horton Conway. *Regular Algebra and Finite Machines.* Chapman & Hall, 1971.

[Cyr10] Aiswarya Cyriac. Temporal Logics for Concurrent Recursive Programs. Master's thesis, Master Parisien de Recherche en Informatique, Paris, France, 2010.

[DG07] Manfred Droste and Paul Gastin. Weighted Automata and Weighted Logics. *Theoretical Computer Science*, 380(1-2):69–86, June 2007.

[DG09] Manfred Droste and Paul Gastin. Weighted Automata and Weighted Logics. In Werner Kuich, Heiko Vogler, and Manfred Droste, editors, *Handbook of Weighted Automata*, EATCS Monographs in Theoretical Computer Science, chapter 5, pages 175–211. Springer, 2009.

[DK09] Manfred Droste and Werner Kuich. Semirings and Formal Power Series. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, EATCS Monographs in Theoretical Computer Science, chapter 1, pages 3–27. Springer, 2009.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata.* EATCS Monographs in Theoretical Computer Science. Springer, 2009.

[DL06] Stéphane Demri and Ranko Lazić. LTL with the Freeze Quantifier and Register Automata. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 17–26, Seattle, Washington, USA, August 2006. IEEE Computer Society Press.

[DM10a] Manfred Droste and Ingmar Meinecke. Describing Average- and Longtime-Behavior by Weighted MSO Logics. In *Proceedings of the 35th International Conference on Mathematical Foundations of Computer Science (MFCS'10)*, volume 6281 of *Lecture Notes in Computer Science*, pages 537–548, 2010.

[DM10b] Manfred Droste and Ingmar Meinecke. Regular Expressions on Average and in the Long Run. In *Proceedings of the 15th International Conference on Implementation and Application of Automata (CIAA'10)*, volume 6482 of *Lecture Notes in Computer Science*, pages 211–221. Springer-Verlag, 2010.

[DP07] Yuxin Deng and Catuscia Palamidessi. Axiomatizations for Probabilistic Finite-State Behaviors. *Theoretical Computer Science*, 373:92–114, 2007.

[DP12] Manfred Droste and Bundit Pibaljommee. Weighted Nested Word Automata and Logics Over Strong Bimonoids. In *Proceedings of the 17th International Conference on Implementation and Application of Automata (CIAA'12)*, volume 7381 of *Lecture Notes in Computer Science*, pages 138–148. Springer, 2012.

[DPV04] Manfred Droste, Christian Pech, and Heiko Vogler. A Kleene Theorem for Weighted Tree Automata. *Theory of Computing Systems*, 38(1):1–38, September 2004.

[DR95] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces.* World Scientific, 1995.

[DR07] Manfred Droste and George Rahonis. Weighted Automata and Weighted Logics with Discounting. In *Proceedings of the 12th International Conference on Implementation and Application of Automata (CIAA'07)*, volume 4783 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2007.

[DS89] Cynthia Dwork and Larry J. Stockmeyer. On the Power of 2-Way Probabilistic Finite State Automata. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (SFCS'89)*, pages 480–485. IEEE Computer Society, 1989.

[DV06] Manfred Droste and Heiko Vogler. Weighted Tree Automata and Weighted Logics. *Theoretical Computer Science*, 366(3):228–247, November 2006.

[DV09] Manfred Droste and Heiko Vogler. Weighted Logics for Unranked Tree Automata. *Theory of Computing Systems*, June 2009.

[EH99] Joost Engelfriet and Hendrik Jan Hoogeboom. Tree-Walking Pebble Automata. *Jewels are forever, contributions to Theoretical*, pages 72–83, 1999.

[EH07] Joost Engelfriet and Hendrik Jan Hoogeboom. Automata with Nested Pebbles Capture First-Order Logic with Transitive Closure. *Logical Methods in Computer Science*, 3:1–27, 2007.

[EHS07] Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML Transformation by Tree-Walking Transducers with Invisible Pebbles. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'07)*, pages 63–72. ACM, 2007.

[Elg61] Calvin C. Elgot. Decision Problems of Finite Automata Design and Related Arithmetics. *Transactions of the American Mathematical Society*, 98:21–52, 1961.

[FdRS03] Massimo Franceschet, Maarten de Rijke, and Bernd-Holger Schlingoff. Hybrid Logics on Linear Structures: Expressivity and Complexity. In *Proceedings of the 10th International Symposium on Temporal Representation and Reasoning, and Fourth International Conference on Temporal Logic (TIME'03)*, pages 192–202. IEEE Computer Society, 2003.

[FGO12] Nathanaël Fijalkow, Hugo Gimbert, and Youssouf Oualhadj. Deciding the Value 1 Problem for Probabilistic Leaktight Automata. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science (LICS'12)*, pages 295–304. IEEE Computer Society Press, 2012.

[FHW10] Sebastian Fischer, Frank Huch, and Thomas Wilke. A Play on Regular Expressions: Functional Pearl. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming (ICFP'10)*. ACM, 2010.

[Fic07] Ina Fichtner. *Characterizations of Recognizable Picture Series*. PhD thesis, Universität Leipzig, 2007.

[Fic11] Ina Fichtner. Weighted Picture Automata and Weighted Logics. *Theory of Computing Systems*, 48(1):48–78, 2011.

[FL79] Michael J. Fischer and Richard E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest Path. *Communication of the ACM*, 5(6):345, 1962.

[FLT10] Uli Fahrenberg, Kim G. Larsen, and Claus Thrane. Quantitative Analysis of Weighted Transition Systems. *Journal of Logic and Algebraic Programming*, 79:689–703, 2010.

[FM09] Zoltán Fülöp and Loránd Muzamel. Weighted Tree-Walking Automata. *Acta Cybernetica*, 19:275–293, 2009.

[Glu60] Victor M. Glushkov. On a Synthesis Algorithm for Abstract Automata. *Ukrainian Mathematical Journal*, 12(2):147–156, 1960. In Russian.

[Glu61] Victor M. Glushkov. The Abstract Theory of Automata. *Russian Mathematical Surveys*, 16:1–53, 1961. In Russian.

[GO10] Hugo Gimbert and Youssouf Oualhadj. Probabilistic Automata on Finite Words: Decidable and Undecidable Problems. In *Proceedings of the 37th International Colloquium conference on Automata, Languages and Programming: Part II (ICALP'10)*, volume 6199 of *Lecture Notes in Computer Science*, pages 527–538. Springer, 2010.

[Gol99] Jonathan S. Golan. *Semirings and their Applications*. Springer, 1999.

[Hof81] Frank Hoffmann. One Pebbles Does Not Suffice to Search Plane Labyrinths. *Fundamentals of Computation Theory*, 117:433–444, 1981.

[HR05] Ian Hodkinson and Mark Reynolds. Separation - Past, Present, and Future. *Essays in Honour of Dov Gabbay on his 60th Birthday*, 27:117–142, 2005.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *An Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[KF94] Michael Kaminski and Nissim Francez. Finite-Memory Automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[Kle56] Stephen C. Kleene. Representation of Events in Nerve Nets and Finite Automata. *Automata Studies*, pages 3–42, 1956.

[KM09] Kevin Knight and Jonathan May. Applications of Weighted Automata in Natural Language Processing. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, EATCS Monographs in Theoretical Computer Science, chapter 14, pages 555–579. Springer, 2009.

[KMO$^+$12] Stefan Kiefer, Andrzej S. Murawski, Joël Ouaknine, Björn Wachter, and James Worrell. On the Complexity of the Equivalence Problem for Probabilistic Automata. In *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'12)*, volume 7213 of *Lecture Notes in Computer Science*, pages 467–481. Springer, 2012.

[KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[KS85] Werner Kuich and Arto Salomaa. *Semirings, Automata and Languages.* EATCS Monographs in Theoretical Computer Science. Springer-Verlag, 1985.

[Kui97] Werner Kuich. Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 9, pages 609–678. Springer-Verlag, 1997.

[Lan05] Martin Lange. Model Checking Propositional Dynamic Logic with All Extras. *Journal of Applied Logic*, 4(1):39–49, 2005.

[LPS08] Kamal Lodaya, Paritosh K. Pandya, and Simoni S. Shah. Marking the Chops: An Unambiguous Temporal Logic. In *Fifth IFIP International Conference On Theoretical Computer Science (TCS'08)*, pages 461–476. Springer, 2008.

[LRGS04] Sylvain Lombardy, Yann Régis-Gianas, and Jacques Sakarovitch. Introducing Vaucanson. *Theoretical Computer Science*, 328(1-2):77–96, 2004.

[LS07] Martin Leucker and César Sánchez. Regular Linear Temporal Logic. In *Proceedings of the 4th International Conference on Theoretical Aspects of Computing (ICTAC'07)*, volume 4711 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2007.

[Mar04] Maarten Marx. Conditional XPath, the First Order Complete XPath Dialect. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT (PODS'04)*, pages 13–22, New York, NY, USA, 2004. ACM.

[Mat98] Oliver Matz. One Quantifier Will Do in Existential Monadic Second-Order Logic over Pictures. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98)*, volume 1450 of *Lecture Notes in Computer Science*, pages 751–759. Springer, 1998.

[Mat10] Christian Mathissen. Weighted Logics for Nested Words and Algebraic Formal Power Series. *Logical Methods in Computer Science*, 6(1), 2010.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc., 1967.

[Moh09] Mehryar Mohri. Weighted Automata Algorithms. In Werner Kuich, Heiko Vogler, and Manfred Droste, editors, *Handbook of Weighted Automata*, EATCS Monographs in Theoretical Computer Science, chapter 6, pages 213–254. Springer, 2009.

[MY60] Robert F. McNaughton and Hisao M. Yamada. Regular Expressions and State Graphs for Automata. *IRE Transactions on Electronic Computers*, EC-9(1):39 –47, 1960.

[NSV04] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite State Machines for Strings over Infinite Alphabets. *ACM Transactions on Computational Logic*, 5:403–435, 2004.

[Paz71] Azaria Paz. *Introduction to Probabilistic Automata.* Academic Press, 1971.

[Rab63] Michael O. Rabin. Probabilistic Automata. *Information and Control*, 6(3):230–245, 1963.

[Rah07] George Rahonis. Weighted Müller Tree Automata and Weighted Logics. *Journal of Automata Languages and Combinatorics*, 12:455–483, 2007.

[Rav07] Bala Ravikumar. On some Variations of Two-Way Probabilistic Finite Automata Models. *Theoretical Computer Science*, 376(1-2):127–136, 2007.

[Rol80] H. A. Rollik. Automaten in Planaren Graphen. *Acta Informatica*, 13(3):287–298, 1980.

[RS59] Michael O. Rabin and Dana Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[Sak09] Jacques Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009.

[Sak12] Jacques Sakarovitch. Automata and Expressions. In *AutoMathA Handbook.* 2012. To appear.

[SBBR11] Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, and Jan Rutten. Quantitative Kleene Coalgebras. *Information and Computation*, 209(5):822–849, 2011.

[Sch61] Marcel-Paul Schützenberger. On the Definition of a Family of Automata. *Information and Control*, 4:245–270, 1961.

[Seg06] Roberto Segala. Probability and Nondeterminism in Operational Models of Concurrency. In *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR'06)*, volume 4137 of *Lecture Notes in Computer Science*, pages 64–78. Springer, 2006.

[She59] John C. Shepherdson. The Reduction of Two-Way Automata to One-Way Automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.

[SS07] Mathias Samuelides and Luc Segoufin. Complexity of Pebble Tree-Walking Automata. In *Proceedings of the 16th International Conference on Fundamentals of Computation Theory (FCT'07)*, volume 4639 of *Lecture Notes in Computer Science*, pages 458–469. Springer, 2007.

[tCS10] Balder ten Cate and Luc Segoufin. Transitive Closure Logic, and Nested Tree Walking Automata, and Xpath. *Journal of the ACM*, 57(3):1–41, March 2010.

[Tho82] Wolfgang Thomas. Classifying Regular Events in Symbolic Logic. *Journal of Computer and System Sciences*, 25:360–376, 1982.

[Tho91] Wolfgang Thomas. On Logics, Tilings, and Automata. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming (ICALP'91)*, volume 510 of *Lecture Notes in Computer Science*, pages 441–453. Springer, 1991.

[Tra61] Boris A. Trakhtenbrot. Finite Automata and Logic of Monadic Predicates. *Doklady Akademii Nauk SSSR*, 149:326–329, 1961.

[Tze92] Wen-Guey Tzeng. A Polynomial-Time Algorithm for the Equivalence of Probabilistic Automata. *SIAM Journal on Computing*, 21(2):216–227, 1992.

[Var95a] Moshe Y. Vardi. Alternating Automata and Program Verification. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1995.

[Var95b] Moshe Y. Vardi. On the Complexity of Bounded-Variable Queries. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'95)*, pages 266–276. ACM, 1995.

[vGSS95] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, Generative and Stratified Models of Probabilistic Processes. *Information and Computation*, 121(1):59–80, 1995.

[VW86] Moshe Y. Vardi and Pierre Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Proceedings of the 1st Annual Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society Press, 1986.

[War62] Stephen Warshall. A Theorem on Boolean Matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[Wei12] Thomas Weidner. Probabilistic Automata and Probabilistic Logic. In *Proceedings of the 37th International Conference on Mathematical Foundations of Computer Science (MFCS'12)*, volume 7464 of *Lecture Notes in Computer Science*, pages 813–824. Springer, 2012.

Nomenclature

A^+	Set of non-empty words over alphabet A, page 13
$A^\star$	Set of words over alphabet A, page 13
α	Test formula, page 26
$\mathrm{ar}(a)$	Arity of letter a, page 13
$\lvert E\rvert$	Cardinality of set E, page 11
$\mathrm{depth}(E)$	Depth of expression E, page 27
$\mathsf{d}(v, v')$	Distance between two vertices of a graph, page 12
ε	Empty word, page 13
$\mathrm{chopWE}(\mathbb{S}, A, D, \mathrm{Var})$	Set of chop weighted expressions, page 31
$\mathrm{WE}(\mathbb{S}, A, D, \mathrm{Var})$	Set of weighted expressions, page 29
$\mathrm{HWE}(\mathbb{S}, A, D, \mathrm{Var})$	Set of hybrid weighted expressions, page 26
$\mathrm{PWE}(\mathbb{S}, A, D, \mathrm{Var})$	Set of pebble weighted expressions, page 59
$\mathrm{Free}(\mathcal{A})$	Set of free variables in automaton $\mathcal{A}$, page 41
$\mathrm{Free}(E)$	Set of free variables in expression E, page 27
$\mathrm{Free}(\alpha)$	Set of pebble names free in formula α, page 26
$\mathcal{G}(A, D)$	Set of pointed graphs with alphabet A and directions D, page 12
$[a, b]$	Closed real interval, page 11
$[a \mathinner{..} b]$	Closed integer interval, page 11
$A \rightharpoonup B$	Partial mapping (or function), page 11
$A \to B$	Mapping (or function), page 11
$\mathcal{MG}(A, D, \mathrm{Var})$	Set of marked graphs, page 34
$\mathcal{MT}\mathrm{race}_{\mathrm{proc}}(A)$	Set of graph representations of Mazurkiewicz traces with alphabet A, page 16
$\mathbb{M}$	Monoid, page 17
$\mathcal{N}\mathrm{est}(A)$	Set of graph representations of the nested words over A, page 14
$\mathcal{P}\mathrm{ict}(A)$	Set of graph represetations of the pictures with pixel in A, page 15

$\mathfrak{P}(E)$	Powerset of set E, page 11
$f_{\mid A}$	Restriction of a function, page 11
$\mathbb{S}$	Semiring, page 18
$\mathbb{S}\langle Z\rangle$	Set of polynomials over Z with coefficients in $\mathbb{S}$, page 20
$\mathbb{S}\langle\langle Z\rangle\rangle$	Set of formal power series over Z with coefficients in $\mathbb{S}$, page 20
$\mathbb{S}^{I\times J}$	Matrices indexed by $I \times J$ with coefficients in $\mathbb{S}$, page 20
$\mathrm{supp}(f)$	Support of a formal power series f, page 20
$\mathrm{Test}(A, D, \mathrm{Var})$	Set of test formulae, page 26
$\mathcal{T}\mathrm{ree}(A)$	Set of graph representations of the ranked trees over A, page 13
$\mathsf{type}(v)$	Type of vertex v, page 12
$\mathcal{U}(A, D, \mathrm{Var})$	Set of units of the marked graphs, page 34
$\mathrm{Var}(E)$	Set of variables in E, page 26
$\mathrm{WA}(\mathbb{S})$	Set of weighted automata over $\mathbb{S}$, page 41
$\mathrm{PWA}(\mathbb{S})$	Set of pebble weighted automata over $\mathbb{S}$, page 52
$\mathrm{gWA}(\mathbb{S})$	Set of generalized weighted automata over $\mathbb{S}$, page 44
$\mathrm{gPWA}(\mathbb{S})$	Set of generalized pebble weighted automata over $\mathbb{S}$, page 59
$1\mathrm{WA}(\mathbb{S})$	Set of one-way weighted automata over $\mathbb{S}$, page 116
$1\mathrm{PWA}(\mathbb{S})$	Set of one-way pebble weighted automata over $\mathbb{S}$, page 116
$\lvert w\rvert$	Length of word w, page 13
w	Word, page 13
$\mathcal{W}\mathrm{ord}(A)$	Set of graph representations of the words in A^+, page 13

Index