Cunf: A Tool for Unfolding and Verifying Petri Nets with Read Arcs

César Rodríguez and Stefan Schwoon

LSV (ENS Cachan & CNRS & INRIA), France

Abstract. Cunf is a tool for building and analyzing unfoldings of Petri nets with read arcs. An unfolding represents the behaviour of a net by a partial order, effectively coping with the state-explosion problem stemming from the interleaving of concurrent actions. C-net unfoldings can be up to exponentially smaller than Petri net unfoldings, and recent work proposed algorithms for their construction and verification. Cunf is the first implementation of these techniques, it has been carefully engineered and optimized to ensure that the theoretical gains are put into practice.

1 Overview

Petri nets are a model for concurrent, distributed systems. Unfoldings are a well-established technique for verifying properties of Petri nets; their use for this purpose was initially proposed by McMillan [6]. The unfolding of a (Petri) net is another net of acyclic structure that fully represents the state-space (reachable markings) of the first, see, e.g., fig. 1 (c). Because unfoldings represent behaviour by acyclic structures rather than by interleaved actions, they are often exponentially smaller than the state-space of N, and never larger than it.

Recently, the unfolding construction was extended to Petri nets with read arcs, also called *contextual nets* (c-nets) [2]. This extension is partially motivated by the fact that c-net unfoldings can yet again be exponentially smaller than Petri net unfoldings. In this paper, we present Cunf the first tool for constructing and analyzing c-net unfoldings, freely available from [8]. The theoretical basis of the tool was presented in [2, 1, 9].

We assume the reader is familiar to Petri nets [7]. A *c-net* is a Petri net where in addition to the ordinary *arcs* (arrows) between *places* and *transitions*, one may use *read arcs*. Figure 1 (a) shows a c-net, read arcs are the undirected lines; we say that t_2 *reads* p_4. A *marking* enables t_2 if it puts tokens on places p_1 and p_4; but firing t_2 only consumes the token in p_1, the token in p_4 remains there.

Every c-net can be seen as the marking-equivalent Petri net that results from replacing read arcs for pairs of arcs, as in fig. 1 (a) and (b). Although both nets have the same markings, remark, however, that t_1, t_2 are concurrent in (a), both read p_4 at the same time, but not in (b), as they compete for the token in p_4.

The *unfolding* of a bounded c-net N is another well-defined, finite, *acyclic* c-net $\mathcal{P}_N$, where each place (resp. transition) of $\mathcal{P}_N$ is labelled by a place (resp. transition) of N and such that runs of $\mathcal{P}_N$ are labelled by runs of N.

D. Van Hung and M. Ogawa (Eds.): ATVA 2013, LNCS 8172, pp. 492–495, 2013.

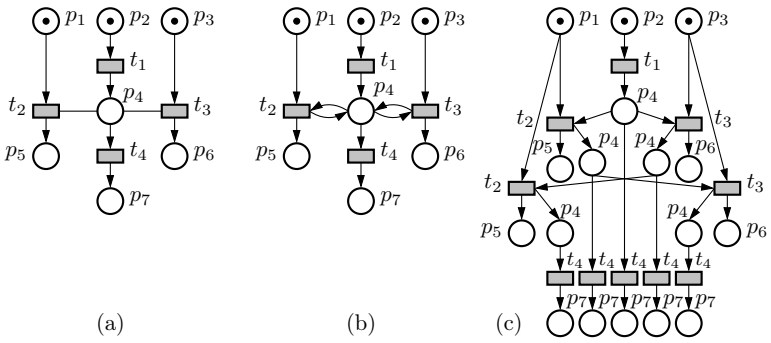

Fig. 1. (a) a c-net; (b) its encoding into a Petri net; (c) unfolding of (b)

The crucial property of $\mathcal{P}_N$ is that for every marking m of N, it contains a marking m' labelled by m.[1] Reachability on N is a PSPACE-complete problem which, however, reduces to an NP-complete problem on $\mathcal{P}_N$. The decreased complexity comes from $\mathcal{P}_N$ being acyclic. Thus, $\mathcal{P}_N$ can be seen as a symbolic representation of the reachable markings of N, particularly compact for concurrent systems.

Abstractly, this explains why c-net unfoldings can be even smaller than plain unfoldings. The unfolding of fig. 1 (a), which is isomorphic to (a), exploits that t_1, t_2 are concurrent and leaves them unmodified. The unfolding of (b), however, need to *unfold* the loops around p_4, explicitly producing all *interleavings* of the reading transitions t_1, t_2 — up to exponentially many of them for n readers.

2 Cunf and Cna's Algorithms and Their Implementation

Our toolset mainly consists of two programs: Cunf constructs unfoldings of 1-safe (places carry 0 or 1 tokens) c-nets, and Cna (Contextual Net Analyser) carries out verification on them.

Unfoldings are built iteratively. Cunf starts with an unfolding prefix containing just a copy of the initial marking of N. This prefix is extended with one *event* (transition), called *possible extension*, yielding a new prefix; this is repeated until the unfolding prefix is big enough to represent all reachable markings of N. Computing the possible extensions is NP-complete, and requires solving (a variant of) the coverability problem for sets of places of the prefix.

Cunf achieves this by a *concurrency relation* on the (enriched) places of the prefix [1]. This relation can be seen as a database that serves to both solve coverability queries and update the relation itself. Cunf spends more than 80% of the time computing the concurrency relation. Efficient computation of the unfolding, thus, almost entirely relies on the efficient computation of this relation, which Cunf implements with adjacency lists. The tool uses around a dozen optimizations for handling these lists, some of them are reported in [1, Sec. 6].

[1] The reader acquainted with the literature on unfoldings may have realized that by *unfolding*, or $\mathcal{P}_N$, in this paper we mean the unique, finite, marking-complete branching process one builds from the c-net after fixing a complete adequate order [1].

To keep the unfolding finite, certain enriched events are marked as *cut-offs*, they are the pruning points of the otherwise potentially infinite branches. Intuitively, those are events whose *history* reaches a marking already reached by (the history of) an event previously added, cf. [1].

Cunf is a mature tool that comprises around 4000 lines of C code, carefully profiled and optimized during its 3 years of existence. It is a command-line tool, but comes with scripts to translate the output of graphical c-net editors such as Coloane [5]. Cunf and Cna are integrated in the Cosyverif [3] environment, which facilitates its invocation and usage. Also, several c-net generators (Conway's game of life, Dekker's algorithm) are distributed with the tool [8].

Cna inputs unfoldings generated with Cunf and searches for reachable markings of the original c-net that enable no transition (deadlocks) or mark a set of given places (coverability). Cna generates, out of the unfolding, a propositional formula whose models coincide with the (offending) traces searched by the tool. It relies on Minisat [4] to solve the formula, and displays the trace if it is found. Notice that once the unfolding is built, it can serve to answer multiple queries. Around 10 optimizations for reducing the solving time are implemented. We highlight, e.g., the elimination of *stubborn events* [9], i.e., certain events that negatively impact the performance of Minisat's unit-propagation; or the reductions of the *asymmetric conflict graph*, see [9]. In our benchmarks, Cna has better accumulated solving time than previously existing verification tools [9], which proves that Cna's algorithms are practical.

3 Experiments and Applications for the Tool

Every c-net can be encoded into an equivalent Petri net. The c-net unfolding can be smaller, but its construction algorithm is more involved. This posed several questions: Are c-net unfoldings smaller than ordinary ones? Can they be computed faster? Is reachability checking practical on c-net unfoldings?

These questions drove our experiments [1,9]. Considerable effort was invested into assuring the efficiency and correctness of the unfolder and analyser. In [1], we applied Cunf to a benchmark of around 100 nets gathered from the unfolding literature, comparing the results to those obtained from other well established Petri-net based tools. Contextual unfolding was significantly *faster* in almost all examples, and *smaller* in roughly half of them. These unfoldings had between 10^2 and 10^5 events; among the larger ones, Cunf unfolded an average rate of 25000 events per second, running on a 2.67GHz CPU.

Table 1 shows some experimental results. For each example, Cunf is run on the c-net and its Petri net encoding, and deadlocked-markings are searched with Cna. Running times, number of events in the unfoldings, and *histories* [1] for c-net unfoldings are shown. The numbers for the plain unfoldings are in fact *ratios* over corresponding numbers in c-net unfolding. C-net unfolding is faster than plain unfolding in 4 cases, and slower in 3. In this 3 cases, however, it is between 14 and 58 times smaller. Cna running times are much smaller than Cunf's ones, so verification seems not to be the bottleneck.

Table 1. Experimental results, see the text for more information

Net		Contextual Unfoldings				Plain Unfoldings		
		Cunf			Cna	Cunf		Cna
Name	Ddlk.	Time	Hist.	Ev.	Time	Time	Ev.	Time
Bds(1)	No	0.10	4210	1830	0.01	4.19	7.05	8.00
Byz	No	2.36	8044	8044	1.68	1.35	1.83	0.23
Ftp	No	16.30	50928	50928	0.06	2.26	1.80	4.19
Rw(1,2)	No	0.924	49179	49179	0.01	1.42	1.00	1.50
Key(4)	Yes	1.32	21742	4754	0.01	0.82	14.64	64.00
Dij(5)	No	9.89	126240	10702	1.17	0.48	14.04	1.85
Dek(60)	No	4.92	216120	3720	0.03	0.86	58.10	0.43

C-net unfoldings are smaller when the c-nets contain transitions that *concurrently read* common resources, as t_1, t_2 in fig. 1 (a). This happens naturally in several applications. Last two examples of table 1 are models of the Dijkstra's and Dekker's mutual exclusion algorithms where c-net unfoldings exhibit important gains. Recall that in both algorithms, *concurrent processes* need to *read* other processes' state variables, hence the gain. Verification of mutual exclusion protocols, we believe, could be an important application of c-net unfoldings.

Hazard checking in asynchronous circuits (ACs) [6] is another promising application. A network of asynchronous boolean gates can be modelled by a c-net, where each gadget encoding a boolean gate contains many read arcs [9]. Hazards are undesirable behaviours of ACs, whose existence reduces to a coverability question on the c-net [6]. In our experiments, we observed that signal changes in the circuit could propagate in many different orders, which were distinguished by Petri-net unfoldings but not by c-net unfoldings, reducing the unfolding size [9].

References

1. Baldan, P., Bruni, A., Corradini, A., König, B., Rodríguez, C., Schwoon, S.: Efficient unfolding of contextual Petri Nets. Theo. Comp. Sci. 449, 2–22 (2012)
2. Baldan, P., Corradini, A., König, B., Schwoon, S.: McMillan's complete prefix for contextual nets. In: Jensen, K., van der Aalst, W.M.P., Billington, J. (eds.) ToPNoC I. LNCS, vol. 5100, pp. 199–220. Springer, Heidelberg (2008)
3. Cosyverif Project: Cosyverif `http://www.cosyverif.org`
4. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
5. LIP6/MoVe Team: Coloane, `http://coloane.lip6.fr/`
6. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 164–177. Springer, Heidelberg (1993)
7. Murata, T.: Petri Nets: Properties, analysis and applications. Proc. of the IEEE 77(4), 541–580 (1989)
8. Rodríguez, C.: Cunf, `http://www.lsv.ens-cachan.fr/~rodriguez/tools/cunf/`
9. Rodríguez, C., Schwoon, S.: Verification of Petri Nets with Read Arcs. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 471–485. Springer, Heidelberg (2012)