Basics of model checking

Paul Gastin
LIAFA (Paris) and LSV (Cachan)
Paul.Gastin@liafa.jussieu.fr
Paul.Gastin@lsv.ens-cachan.fr

MOVEP, Dec. 2004

Need for formal verifications methods

Critical systems
- Transport
- Energy
- Medicine
- Communication
- Finance
- Embedded systems
- ...

Complementary approaches
- Theorem prover
- Model checking
- Test

Model Checking

3 steps
- Constructing the model $M$ (transition systems)
- Formalizing the specification $\varphi$ (temporal logics)
- Checking whether $M \models \varphi$ (algorithmics)

Main difficulties
- Size of models (combinatorial explosion)
- Expressivity of models or logics
- Decidability and complexity of the model-checking problem
- Efficiency of tools

Challenges
- Extend models and algorithms to cope with more systems.
  Infinite systems, parameterized systems, probabilistic systems, concurrent systems, timed systems, hybrid systems, ...
- Scale current tools to cope with real-size systems.
  Needs for modularity, abstractions, symmetries, ...
Constructing the model

Example: Men, Wolf, Goat, Cabbage

Model = Transition system
- State = who is on which side of the river
- Transition = crossing the river

References


Outline

1. Introduction
2. Models
   - Specification
     - Linear Time Specifications
     - Branching Time Specifications

Transition system

MWGC → WC → MG → W → MGC → G → MWC → MWG → WC → MWGC

MWG → MGC → G → MWC → MWG → WC → MWGC

C → MWG → MWC → G → MWC → MWG

??

??

??

???
### Kripke structure

\[ M = (S, A, T, I, AP, \ell) \]

- \( S \): set of states (often finite)
- \( T \subseteq S \times A \times S \): set of transitions
- \( I \subseteq S \): set of initial states
- \( AP \): set of atomic propositions
- \( \ell : S \to 2^{AP} \): labelling function.

### Kripke structures with variables

\[ M = (S, A, V, T, I, AP, \ell) \]

- \( V \): set of typed variables, e.g., boolean, [0..4], ...
- Condition: formula involving variables
- Update: modification of variables
- Transition: \( p \mapsto \text{condition, label, update}, q \)

**Programs = Kripke structures with variables**

- Program counter = states
- Instructions = transitions
- Variables = variables

### Using variables

**Digicode**

- \( \text{cpt} < n \)
- \( B, C \)
- \( \text{cpt}++ \)

**Diagram**

\[ \begin{array}{c}
1 \quad A \\
B, C \\
\text{cpt}++ \\
3 \\
\text{OPEN} \\
\end{array} \]

**Program**

- How can we easily describe big systems?

### Expanding variables (\( n = 2 \))

**Digicode**

- \( \text{cpt} < n \)
- \( B, C \)
- \( \text{cpt}++ \)

**Diagram**

\[ \begin{array}{c}
1, 0 \quad A \\
B, C \\
\text{cpt}++ \\
3, 0 \\
\text{OPEN} \\
\end{array} \]

**Program**

- How can we easily describe big systems?
**Symbolic representation**

Logical representation

Synchronization by states: \((P, P)\) is forbidden

**Synchronized products**

General product

- Components: \(M_i = (S_i, A_i, T_i, I_i, AP_i, \ell_i)\)
- Product: \(M = (S, A, T, I, AP, \ell)\) with
  \[ S = \prod_i S_i, \quad A = \prod_i (A_i \cup \{\varepsilon\}), \quad I = \prod_i I_i \]
  \[ T = \{(p_1, \ldots, p_n) \overset{(a_1, \ldots, a_n)}{\rightarrow} (q_1, \ldots, q_n) | \text{ for all } i, (p_i, a_i, q_i) \in T_i \text{ or } \}
  \quad p_k = q_i \text{ and } a_i = \varepsilon\]
  \[ AP = \bigcup_i AP_i \text{ and } \ell(p_1, \ldots, p_n) = \bigcup_i \ell(p_i) \]

Synchronized products are restrictions of the general product.

- Synchronous: \(A_{\text{sync}} = \prod_i A_i\)
- Asynchronous: \(A_{\text{async}} = \bigcup_i A_i\)
- By states: \(S_{\text{sync}} \subseteq S\)
- By labels: \(A_{\text{sync}} \subseteq A\)
- By transitions: \(T_{\text{sync}} \subseteq T\)

**Modular description of concurrent systems**

Elevator

- Cabin: \(\text{Closed} \rightarrow \text{Opened}\)
- Door for level \(i\): \(\text{Closed} \rightarrow \text{Open}\)
- Call for level \(i\): \(\text{False} \rightarrow \text{True}\)

The actual system is a synchronized product of all these automata. It consists of (at most) \(3 \times 2^3 \times 2^3 = 192\) states.

**Example: Printer manager**

Synchronization by states: \((P, P)\) is forbidden

Idle

\(\text{Wait} \rightarrow \text{Print}\)

\(\text{Idle} \rightarrow \text{Wait} \rightarrow \text{Print}\)

\(\text{Wait} \rightarrow \text{Print}\)
**Example: Elevator**

Synchronization by actions

**Example: digicode**

Synchronization by transitions

**Example: Peterson's algorithm (1981)**

Synchronization by shared variables

**High-level descriptions**

- Sequential programs = transition system with variables
- Concurrent programs with shared variables
- Concurrent programs with Rendez-vous
- Concurrent programs with FIFO communication
- Petri net
- ...
Models: expressivity versus decidability

(Un)decidability
- Automata with 2 integer variables = Turing powerful
- Restriction to variables taking values in finite sets
- Asynchronous communication: unbounded fifo channels = Turing powerful
- Restriction to bounded channels

Some infinite state models are decidable
- Petri nets. Several unbounded integer variables but no zero-test.
- Pushdown automata. Model for recursive procedure calls.
- Timed automata.
- ...

Static and dynamic properties

Static properties
Example: Mutual exclusion
Most safety properties are static.
They can be reduced to reachability.

Dynamic properties
Example: Every request should be eventually granted.
\[ \forall t, (\text{Call}, t) \rightarrow \exists t', (\text{atLevel}, t') \land \text{openDoor}, (t')) \]
The elevator should not cross a level for which a call is pending without stopping.
\[ \forall \forall t', (\text{Call}, t) \land t \leq t' \land \text{atLevel}, (t')) \rightarrow \exists t \leq t' \leq t'', (\text{atLevel}, (t'') \land \text{openDoor}, (t''))) \]

First Order specifications

First order logic
- These specifications can be written in FO(<).
- FO(<) has a good expressive power.
  ... but FO(<)-formulas are not easy to write and to understand.
- FO(<) is decidable.
  ... but satisfiability and model checking are non elementary.

Temporal logics
- no variables: time is implicit.
- quantifications and variables are replaced by modalities.
- Usual specifications are easy to write and read.
- Good complexity for satisfiability and model checking problems.
Linear versus Branching

Let $M = (S, T, I, AP, t)$ be a Kripke structure.

Linear specifications

Example: The printer manager is fair. On each run, whenever some process requests the printer, it eventually gets it.

Execution sequences (runs): $\sigma = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots$ with $s_i \rightarrow s_{i+1} \in T$

Two Kripke structures having the same execution sequences satisfy the same linear specifications.

Actually, linear specifications only depend on the label of the execution sequence

$$\ell(\sigma) = \ell(s_0) \rightarrow \ell(s_1) \rightarrow \ell(s_2) \rightarrow \cdots$$

Branching specifications

Example: Each process has the possibility to print first.

Such properties depend on the execution tree.

Execution tree = unfolding of the transition system

Linear Temporal Logic (Pnueli 1977)

Syntax: $LTL(AP, X, U)$

$$\varphi ::= \bot \mid p \in AP \mid \neg \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi$$

Semantics: $t = [N, \leq, \lambda]$ with $\lambda : N \rightarrow \Sigma = 2^{AP}$ and $x \in N$

$t, x \models p$ if $p \in \lambda(x)$
$t, x \models \neg \varphi$ if $t, x \not\models \varphi$
$t, x \models \varphi \lor \psi$ if $t, x \models \varphi$ or $t, x \models \psi$
$t, x \models X \varphi$ if $\exists y, x \leq y \land t, y \models \varphi$
$t, x \models \varphi U \psi$ if $\exists z, x \leq z \land t, z \models \psi \land \forall y. (x \leq y < z) \rightarrow t, y \models \varphi$

Example

$$\varphi U \psi \quad \varphi U \psi \quad \cdots \quad \varphi U \psi$$

Outline

1 Introduction
2 Models
3 Specification
   - Linear Time Specifications
   - Branching Time Specifications

Linear Temporal Logic (Pnueli 1977)

Macros:

- **Eventually:** $F \varphi = T U \varphi$
  $$\begin{array}{c}
  F \varphi \\
  \cdot \\
  \cdot \\
  \cdot \\
  \cdot \\
  \varphi
  \end{array}$$

- **Always:** $G \varphi = \neg F \neg \varphi$
  $$\begin{array}{c}
  G \varphi \\
  \cdot \\
  \cdot \\
  \cdot \\
  \cdot \\
  \neg \varphi
  \end{array}$$

- **Weak until:** $\varphi W \psi = G \varphi \lor \varphi U \psi$
  $$\begin{array}{c}
  \varphi W \psi \\
  \cdot \\
  \cdot \\
  \cdot \\
  \cdot \\
  \varphi \psi
  \end{array}$$

- **Release:** $\varphi R \psi = \psi W (\varphi \land \psi) = \neg (\neg \psi U \neg \varphi)$
  $$\begin{array}{c}
  \varphi R \psi \\
  \cdot \\
  \cdot \\
  \cdot \\
  \cdot \\
  \psi
  \end{array}$$

- **Next until:** $\varphi X U \psi = X (\varphi U \psi)$
  $$\begin{array}{c}
  \varphi X U \psi \\
  \cdot \\
  \cdot \\
  \cdot \\
  \cdot \\
  \varphi \psi
  \end{array}$$

- **Next until:** $X \psi = \bot X U \psi$ and $\varphi U \psi = \psi \lor (\varphi \land \varphi X U \psi)$.
Linear Temporal Logic (Pnueli 1977)

Specifications:
- Safety: $G$ good
- MutEx: $\neg F(\text{crit}_1 \land \text{crit}_2)$
- Liveness: $GF$ active
- Response: $G(\text{request} \rightarrow F\text{grant})$
- Response': $G(\text{request} \rightarrow X(\neg \text{request} \land \text{grant}))$
- Release: reset $R$ alarm
- Strong fairness: $GF\text{request} \rightarrow GF\text{grant}$
- Weak fairness: $FG\text{request} \rightarrow GF\text{grant}$

Past LTL

Semantics: $t = [N, \leq, \lambda]$ with $\lambda : N \rightarrow \Sigma = 2^{AP}$ and $x \in N$

$t, x \models Y \varphi$ if $\exists y. y < x \land t, y \models \varphi$

$t, x \models \varphi S \psi$ if $\exists z. z \leq x \land t, z \models \psi \land \forall y. (z < y \leq x) \rightarrow t, y \models \varphi$

Example

```
\begin{align*}
 & t, x \models Y \varphi \quad \text{if} \quad \exists y. y < x \land t, y \models \varphi \\
 & t, x \models \varphi S \psi \quad \text{if} \quad \exists z. z \leq x \land t, z \models \psi \land \forall y. (z < y \leq x) \rightarrow t, y \models \varphi
\end{align*}
```

LTL versus PLTL

$G(\text{grant} \rightarrow Y(\neg \text{grant} S \text{request}))$

$= (\text{request} R \neg \text{grant}) \land G(\text{grant} \rightarrow (\text{request} \lor X(\text{request} R \neg \text{grant})))$

Theorem (Laroussinie & Markey & Schnoebelen 2002)
PLTL may be exponentially more succinct than LTL.
Expressivity

Theorem (Kamp 68)

\[ \text{LTL}(Y, S, X, U) = \text{FO}_\Sigma(\leq) \]

Separation Theorem (Gabbay, Pnueli, Shelah & Stavi 80)

For all \( \varphi \in \text{LTL}(Y, S, X, U) \) there exist \( \varphi^i \in \text{LTL}(Y, S) \) and \( \varphi^j \in \text{LTL}(X, U) \) such that for all \( w \in \Sigma^\omega \) and \( k \geq 0 \),

\[ w, k \models \varphi \iff w, k \models \bigvee_i \varphi^i \land \bigvee_j \varphi^j \]

Corollary: \( \text{LTL}(Y, S, X, U) = \text{LTL}(X, U) \)

For all \( \varphi \in \text{LTL}(Y, S, X, U) \) there exist \( \varphi^i \in \text{LTL}(X, U) \) such that for all \( w \in \Sigma^\omega \),

\[ w, 0 \models \varphi \iff w, 0 \models \varphi^i \]

Elegant algebraic proof of \( \text{LTL}(X, U) = \text{FO}_\Sigma(\leq) \) due to Wilke 98.

Satisfiability for LTL

Let \( AP \) be the set of atomic propositions and \( \Sigma = 2^{AP} \).

(Initial) Satisfiability problem

**Input:** A formula \( \varphi \in \text{LTL}(Y, S, X, U) \)

**Question:** Existence of \( w \in \Sigma^\omega \) such that \( w, 0 \models \varphi \).

Theorem (Sistla & Clarke 85, Lichtenstein et. al 85)

The satisfaction problem for LTL is PSPACE-complete

Model checking for LTL

Model checking problem

**Input:** A Kripke structure \( M = (S, T, I, AP, \ell) \) and a formula \( \varphi \in \text{LTL} \)

**Question:** Does \( M \models \varphi \) ?

- Universal MC: \( M \models \varphi \) if \( \ell(\sigma), 0 \models \varphi \) for all initial infinite run of \( M \).
- Existential MC: \( M \models \varphi \) if \( \ell(\sigma), 0 \models \varphi \) for some initial infinite run of \( M \).

Theorem (Sistla & Clarke 85, Lichtenstein et. al 85)

The Model checking problem for LTL is PSPACE-complete

\[ \text{MC}(X, U) \leq_P \overline{\text{SAT}}(X, U) \] (Sistla & Clarke 85)

Let \( M = (S, T, I, AP, \ell) \) be a Kripke structure and \( \varphi \in \text{LTL}(X, U) \)

Introduce new atomic propositions: \( AP_S = \{ at_s \mid s \in S \} \)

Define \( AP' = AP \uplus AP_S \)

\[ \Sigma' = 2^{AP'} \]

\( \pi : \Sigma' \rightarrow \Sigma^\omega \) by \( \pi(a) = a \cap AP \).

Let \( w \in \Sigma^\omega \). We have \( w \models \varphi \) iff \( \pi(w) \models \varphi \)

Define

\[ \psi_M = \left( \bigvee_{s \in T} at_s \right) \land \bigwedge_{s \in S} \left( \bigvee_{t \neq s} \left( at_s \land \neg at_t \land \bigwedge_{p \in \ell(t)} p \land \bigwedge_{p \in \ell(s)} \neg p \land \bigvee_{t \in T(s)} X at_t \right) \right) \]

We have \( w \models \psi_M \) iff \( \pi(w) = \ell(\sigma) \) for some initial infinite run \( \sigma \) of \( M \).

Therefore, \( M \not\models \varphi \) iff \( \ell(\sigma) \models \neg \varphi \) for some initial infinite run \( \sigma \) of \( M \)

\( w \models \psi_M \land \neg \varphi \) for some \( w \in \Sigma^\omega \)

\( \psi_M \land \neg \varphi \) is satisfiable
Decision procedure for LTL

The core
From an LTL formula \( \varphi \), construct a Büchi automaton \( \mathcal{A}_\varphi \) such that
\[
\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi) = \{ w \in \Sigma^\omega \mid w, 0 \models \varphi \}.
\]

Satisfiability (initial)
Check the Büchi automaton \( \mathcal{A}_\varphi \) for emptiness.

Model checking
Construct the product \( B = M \times \mathcal{A}_{\lnot \varphi} \) so that the successful runs of \( B \) correspond to the successful run of \( \mathcal{A} \) satisfying \( \lnot \varphi \).
Then, check \( B \) for emptiness.

Büchi automata for some LTL formulas

Definition
Recall that \( \Sigma = 2^{AP} \). For \( p, q \in AP \), we let
\[
\begin{align*}
\varSigma_p &= \{ a \in \Sigma \mid p \in a \} \quad \text{and} \quad \Sigma_{\neg p} = \Sigma \setminus \varSigma_p \\
\varSigma_{p \land q} &= \varSigma_p \cap \varSigma_q \quad \text{and} \quad \varSigma_{p \lor q} = \varSigma_p \cup \varSigma_q \\
\varSigma_{p \land \neg q} &= \varSigma_p \setminus \varSigma_q \quad \ldots
\end{align*}
\]

Examples
\[
\begin{align*}
\text{\texttt{F} } p: & \quad \begin{array}{c}
1 \quad \varSigma_p \\
\implies \text{1} \quad \varSigma_p
\end{array} \\
\text{\texttt{XX} } p: & \quad \\
\text{\texttt{G} } p: & \quad \begin{array}{c}
1 \quad \varSigma_p
\end{array}
\end{align*}
\]
Büchi automata for some LTL formulas

Examples

- \( F G p \):

- \( G F p \):

- \( G(p \rightarrow F q) \):

Büchi automata are not closed under complement.

Properties

- Büchi automata are closed under union, intersection, complement.
  - Union: trivial
  - Intersection: easy (exercise)
  - Complement: hard

Let \( \varphi = F((p \land X^n \neg p) \lor (\neg p \land X^n p)) \)

Büchi automata

Exercises

Given Büchi automata for \( \varphi \) and \( \psi \),

- Construct a Büchi automaton for \( X \varphi \) (trivial)
- Construct a Büchi automaton for \( \varphi U \psi \)

This gives an inductive construction of \( A_\varphi \) from \( \varphi \in \text{LTL}(X, U) \) . . .

... but the size of \( A_\varphi \) might be non-elementary in the size of \( \varphi \).
**Generalized Büchi automata**

Definition: acceptance on states
\[ A = (Q, \Sigma, I, T, F_1, \ldots, F_n) \text{ with } F_i \subseteq Q. \]
An infinite run \( \sigma \) is successful if it visits infinitely often each \( F_i \).

\[ \text{Synchronized product with} \]

**Negative normal form**

Syntax (\( p \in AP \))
\[ \varphi ::= \bot \mid p \mid \neg p \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid X \varphi \mid \varphi U \varphi \mid \varphi F \varphi \]

Any formula can be transformed in NNF
\[ \begin{align*}
\neg X \varphi &= X \neg \varphi \\
\neg(\varphi U \psi) &= (\neg \varphi) R (\neg \psi) \\
\neg(\varphi R \psi) &= (\neg \varphi) U (\neg \psi) \\
\neg(\varphi \lor \psi) &= (\neg \varphi) \land (\neg \psi) \\
\neg(\varphi \land \psi) &= (\neg \varphi) \lor (\neg \psi)
\end{align*} \]

Note that this does not increase the number of Temporal subformulas.

**GBA to BA**

Definition
\[ Z \subseteq \text{NNF} \text{ is reduced if} \]
\[ \begin{align*}
&\varphi \text{ formulas in } Z \text{ are of the form } p, \neg p, \text{ or } X \beta, \\
&\bot \notin Z \text{ and } \{p, \neg p\} \notin Z \text{ for all } p \in AP.
\end{align*} \]

Reduction graph

\[ \begin{align*}
\text{Vertices: subsets of NNF} \\
\text{Edges: Let } Y \subseteq \text{NNF} \text{ and let } \alpha \in Y \text{ maximal not reduced.}
\end{align*} \]

If \( \alpha = \alpha_1 \lor \alpha_2 \):
\[ Y \rightarrow Y \setminus \{\alpha\} \cup \{\alpha_1\}, \]
\[ Y \rightarrow Y \setminus \{\alpha\} \cup \{\alpha_2\}, \]

If \( \alpha = \alpha_1 \land \alpha_2 \):
\[ Y \rightarrow Y \setminus \{\alpha\} \cup \{\alpha_1, \alpha_2\}, \]

If \( \alpha = \alpha_1 R \alpha_2 \):
\[ Y \rightarrow Y \setminus \{\alpha\} \cup \{\alpha_1, \alpha_2\}, \]
\[ Y \rightarrow Y \setminus \{\alpha\} \cup \{\alpha_2, X \alpha\}, \]

If \( \alpha = \alpha_1 U \alpha_2 \):
\[ Y \rightarrow Y \setminus \{\alpha\} \cup \{\alpha_2\}, \]
\[ Y \rightarrow Y \setminus \{\alpha\} \cup \{\alpha_1, X \alpha\}. \]

Note the mark \( \alpha \) on the last edge.
**Reduction graph**

Example: \( \varphi = G(p \rightarrow Fq) \)

\[ \varphi = G(\neg p \lor Fq) \]
\[ q, X \varphi \]
\[ \neg p \lor Fq, X \varphi \]
\[ Fq, X \varphi \]
\[ \neg p, X \varphi \]
\[ XFq, X \varphi \]

State = set of obligations.
Reduce obligations to literals and next-formulas.
Note again the mark \( Fq \) on the last edge.

**Automaton \( A_\varphi \)**

Example: \( \varphi = G(p \rightarrow Fq) \)

\[ \varphi = G(\neg p \lor Fq) \]
\[ q, X \varphi \]
\[ \neg p \lor Fq, X \varphi \]
\[ Fq, X \varphi \]
\[ \neg p, X \varphi \]
\[ XFq, X \varphi \]

Transition = check literals and move forward.
Simplification

**Automaton \( A_\varphi \)**

Definition: For \( Y \subseteq \text{NNF} \), let

- \( \text{Red}(Y) = \{ Z \text{ reduced} \mid Y \xrightarrow{\alpha} Z \} \)
- \( \text{Red}_\alpha(Y) = \{ Z \text{ reduced} \mid Y \xrightarrow{\alpha} Z \text{ without using an edge marked with } \alpha \} \)

Definition: For \( Z \subseteq \text{NNF} \) reduced, define

- \( \text{next}(Z) = \{ \alpha \mid X \alpha \in Z \} \)
- \( \Sigma_Z = \bigcap_{p \in Z} \Sigma_p \cap \bigcap_{\neg p \in Z} \Sigma_{\neg p} \)

Automaton \( A_\varphi \)

- States: \( Q = 2^{\text{sub}(\varphi)}, \quad I = \{ \varphi \} \)
- Transitions: \( T = \{ Y \xrightarrow{\Sigma \alpha} \text{next}(Z) \mid Y \in Q \text{ and } Z \in \text{Red}(Y) \} \)
- Acceptance: \( T_a = \{ Y \xrightarrow{\Sigma \alpha} \text{next}(Z) \mid Y \in Q \text{ and } Z \in \text{Red}_\alpha(Y) \} \)
  for each \( \alpha = \alpha_1 \cup \alpha_2 \in \text{sub}(\varphi) \).

**Automaton \( A_\varphi \)**

Theorem: \( L(A_\varphi) = L(\varphi) \)

- \( |Q| \leq 2^{|\varphi|} \)
- number of acceptance tables = number of until sub-formulas.

Corollary

Satisfiability and Model Checking are decidable in PSPACE.

Remark

An efficient construction is based on Very Weak Alternating Automata.
(Gastin & Oddoux, CAV’01)

The domain is still very active.
**Possibility is not expressible in LTL**

**Example**

\( \varphi \): Whenever \( p \) holds, it is possible to reach a state where \( q \) holds.

\( \varphi \) cannot be expressed in LTL.

Consider the two models:

- \( M_1 \):
  
  \[
  \begin{array}{c}
  1 \\
  \rightarrow \\
  p, \ q \\
  2 \\
  \rightarrow \\
  p \\
  3 \\
  \rightarrow \\
  q \\
  4 \\
  \end{array}
  \]

- \( M_2 \):
  
  \[
  \begin{array}{c}
  1 \\
  \rightarrow \\
  p \\
  2 \\
  \rightarrow \\
  p, \ q \\
  3 \\
  \rightarrow \\
  q \\
  4 \\
  \end{array}
  \]

\( M_1 \models \varphi \) but \( M_2 \not\models \varphi \)

\( M_1 \) and \( M_2 \) satisfy the same LTL formulas.

---

**Original References**


---

**Outline**

1. Introduction
2. Models
3. Specification
   - Linear Time Specifications
   - Branching Time Specifications

---

**Quantification on runs**

**Example**

\( \varphi \): Whenever \( p \) holds, it is possible to reach a state where \( q \) holds.

\[ \varphi = \text{AG}(p \rightarrow \text{EF } q) \]

- \( E \): for some infinite run
- \( A \): for all infinite run

**Some specifications**

- \( \text{EF } \varphi \): \( \varphi \) is possible
- \( \text{AG } \varphi \): \( \varphi \) is an invariant
- \( \text{AF } \varphi \): \( \varphi \) is unavoidable
- \( \text{EG } \varphi \): \( \varphi \) holds globally along some path
**CTL* (Emerson & Halpern 86)**

**Syntax:** CTL*: Computation Tree Logic

\[ \varphi ::= \bot | p (p \in \text{AP}) | \neg \varphi | \varphi \vee \varphi | X \varphi | \varphi U \varphi | E \varphi | A \varphi \]

**Semantics:**
Let \( M = (S, T, I, \text{AP}, \ell) \) be a Kripke structure and \( \sigma \) an infinite run of \( M \).

- \( \sigma, i \models E \varphi \) if \( \sigma', 0 \models \varphi \) for some infinite run \( \sigma' \) such that \( \sigma'(0) = \sigma(i) \)
- \( \sigma, i \models A \varphi \) if \( \sigma', 0 \models \varphi \) for all infinite runs \( \sigma' \) such that \( \sigma'(0) = \sigma(i) \)

**State formulas**
A formula of the form \( p \) or \( E \varphi \) or \( A \varphi \) only depends on the current state.
State formulas are closed under boolean connectives.
If \( \varphi \) is a state formula, define \( S(\varphi) = \{ s \in S \mid s \models \varphi \} \)

---

**Model checking of CTL***

**Model checking problem**

- **Input:** A Kripke structure \( M = (S, T, I, \text{AP}, \ell) \) and a formula \( \varphi \in \text{CTL}^* \)
- **Question:** Does \( M \models \varphi \)?

**Remark**

\[ M \models \varphi \iff \ell(\sigma), 0 \models \varphi \text{ for all initial infinite run of } M. \]
\[ \iff I \subseteq S(A \varphi) \]

**Theorem**
The model checking problem for CTL* is PSPACE-complete

**Proof**
PSPACE-hardness: follows from LTL \( \subseteq \text{CTL}^* \).
PSPACE-easiness: inductively compute \( S(\psi) \) for all state formulas.

---

**Computing \( S(\psi) \)**

**State formulas**

- \( S(p) = \{ s \in S \mid p \in \ell(s) \} \)
- \( S(\neg \psi) = S \setminus S(\psi) \)
- \( S(\psi_1 \land \psi_2) = S(\psi_1) \cap S(\psi_2) \)
- \( S(\psi_1 \lor \psi_2) = S(\psi_1) \cup S(\psi_2) \)
- \( S(E \psi) = ? \) Compute \( A'_\psi \), replacing state subformulas of \( \psi \) by new atomic propositions.

To check whether \( s \in S(E \psi) \), check for emptiness the synchronized product of \( A'_\psi \) and \( M \) with initial state \( s \).

- \( A \psi = \neg E \neg \psi \)

**Model checking**

\[ M \models \varphi \iff I \subseteq S(A \varphi). \]

---

**CTL (Clarke & Emerson 81)**

**Syntax:** CTL: Computation Tree Logic

\[ \varphi ::= \bot | p (p \in \text{AP}) | \neg \varphi | \varphi \vee \varphi | EX \varphi | AX \varphi | E \varphi U \varphi | A \varphi U \varphi \]

**Remarks**
The semantics is inherited from CTL*.
All CTL-formulas are state formulas. Hence, we have a simpler semantics.

**Semantics:** only state formulas
Let \( M = (S, T, I, \text{AP}, \ell) \) be a Kripke structure and let \( s \in S \).

- \( s \models p \) if \( p \in \ell(s) \)
- \( s \models EX \varphi \) if \( \exists s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \text{ with } s_1 \models \varphi \)
- \( s \models AX \varphi \) if \( \forall s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \text{ we have } s_1 \models \varphi \)
- \( s \models E \varphi U \psi \) if \( \exists s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \text{ with } s_j = \psi \) and \( s_k \models \psi \) for all \( 0 \leq k < j \)
- \( s \models A \varphi U \psi \) if \( \forall s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \text{ with } s_j = \psi \) and \( s_k \models \psi \) for all \( 0 \leq k < j \).
CTL (Clarke & Emerson 81)

Semantics: only state formulas
Let $M = (S, T, I, AP, \ell)$ be a Kripke structure without deadlocks and let $s \in S$.

- $s \models p$ if $p \in \ell(s)$
- $s \models \Box \varphi$ if $\exists s' \rightarrow s'$ with $s' \models \varphi$
- $s \models \Diamond \varphi$ if $\forall s \rightarrow s'$ we have $s' \models \varphi$
- $s \models E \varphi U \psi$ if $\exists s = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_k \rightarrow s_{k+1}$ with $s_j \models \psi$ and $s_k \models \varphi$ for all $0 \leq k < j$
- $s \models A \varphi U \psi$ if $\forall s = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_j \geq 0$ with $s_j \models \psi$ and $s_k \models \varphi$ for all $0 \leq k < j$

Macros
- $E \varphi = E \top U \varphi$ and $A \varphi = A \top U \varphi$
- $F \varphi = \top U \varphi$
- $E \varphi = \neg A \neg \varphi$ and $A \varphi = \neg EF \neg \varphi$

Equivalent formulas
- $\Diamond \varphi = \neg \Box \neg \varphi$
- $A \varphi U \psi = \neg E (\neg \psi U \psi)$
  $= \neg E (G \neg \psi \land E \neg \psi)$
  $= \neg E (G \neg \neg \varphi \land \neg \psi)$
  $= \neg E G \neg \psi \lor \neg E \psi \land \neg \psi$
- $A G(\text{req} \rightarrow F \text{grant}) = AG(\text{req} \rightarrow F \text{grant})$
- $A G F \varphi = AG AF \varphi$
- $E F G \varphi = EF EG \varphi$
- $E G E F \varphi \neq E G F \varphi$
- $A F A G \varphi \neq A F G \varphi$
- $E G E X \varphi \neq E G X \varphi$

Model checking of CTL

Input: A Kripke structure $M = (S, T, I, AP, \ell)$ and a formula $\varphi \in \text{CTL}$

Question: Does $M \models \varphi$?

Remark
$M \models \varphi$ iff $I \subseteq S(\varphi)$

Theorem
The model checking problem for CTL is decidable in time $O(|M| \cdot |\varphi|)$

Proof
Marking algorithm.
procedure mark(ϕ)

case ϕ = p ∈ AP
    for all s ∈ S do s.ϕ := (p ∈ ℓ(s));

case ϕ = ¬ϕ1
    mark(ϕ1);
    for all s ∈ S do s.ϕ := ¬s.ϕ1;

case ϕ = ϕ1 ∨ ϕ2
    mark(ϕ1); mark(ϕ2);
    for all s ∈ S do s.ϕ := s.ϕ1 ∨ s.ϕ2;

case ϕ = EXϕ1
    mark(ϕ1);
    for all s ∈ S do s.ϕ := false;
    for all (t, s) ∈ T do if s.ϕ1 then t.ϕ := true;

case ϕ = AXϕ1
    mark(ϕ1);
    for all s ∈ S do s.ϕ := true;
    for all (t, s) ∈ T do if ¬s.ϕ1 then t.ϕ := false;

procedure mark(ϕ)

case ϕ = p ∈ AP
    for all s ∈ S do s.ϕ := (p ∈ ℓ(s));

case ϕ = ¬ϕ1
    mark(ϕ1);
    for all s ∈ S do s.ϕ := ¬s.ϕ1;

case ϕ = ϕ1 ∨ ϕ2
    mark(ϕ1); mark(ϕ2);
    for all s ∈ S do s.ϕ := s.ϕ1 ∨ s.ϕ2;

case ϕ = EXϕ1
    mark(ϕ1);
    for all s ∈ S do s.ϕ := false;
    for all (t, s) ∈ T do if s.ϕ1 then t.ϕ := true;

case ϕ = AXϕ1
    mark(ϕ1);
    for all s ∈ S do s.ϕ := true;
    for all (t, s) ∈ T do if ¬s.ϕ1 then t.ϕ := false;

fairness

Fairness

Only fair runs are of interest

- Each process is enabled infinitely often: \( \bigwedge_i GF \text{run}_i \)
- No process stays ultimately in the critical section: \( \bigwedge_i \neg GFCS_i = \bigwedge_i GF\neg CS_i \)

Fair Kripke structure

\( M = (S, T, I, AP, \ell, \mathcal{F}) \) where \( \mathcal{F} = \{ F_1, \ldots, F_n \} \) with \( F_i \subseteq S \).
An infinite run \( \sigma \) is fair if it visits infinitely often each \( F_i \)

Fair quantifications

\( E_f \varphi = E(\text{fair} \land \varphi) \) and \( A_f \varphi = A(\text{fair} \rightarrow \varphi) \)

where
\[ \text{fair} = \bigwedge_i GF F_i \]
**fair CTL**

Syntax of fair-CTL

\[ \varphi ::= \bot \mid p \ (p \in AP) \mid \neg \varphi \lor \varphi \mid E_f X \varphi \mid A_f X \varphi \mid E_f \varphi U \varphi \mid A_f \varphi U \varphi \]

Lemma: CTL\(_f\) cannot be expressed in CTL

Consider the Kripke structure \( M_k \) defined by:

\[ \begin{array}{c}
2k-2 \\
\vdots \\
2k-1 \\
2k-3 \\
p
\end{array} \]

\[ \begin{array}{c}
3 \\
\vdots \\
2 \\
1 \\
p
\end{array} \]

- \( M_k, 2k \models EGF p \) but \( M_k, 2k-2 \not\models EGF p \)
- If \( \varphi \in CTL \) and \( |\varphi| \leq m \leq k \) then \( M_k, 2k \models \varphi \) iff \( M_k, 2m \models \varphi \)

If the fairness condition is \( \ell^{-1}(p) \) then \( E_f F \top \) cannot be expressed in CTL.

**Model checking of CTL\(_f\)**

First step: Computation of Fair = \( \{ s \in S \mid M, s \models E_f F \top \} \)

Compute the SCC of \( M \) with Tarjan’s algorithm (in linear time).

Let \( S’ \) be the union of the SCCs which intersect each \( F_i \).

Then, Fair is the set of states that can reach \( S’ \).

Note that reachability can be computed in linear time.

Reductions

\[ E_f X \varphi = EX(Fair \land \varphi) \quad \text{and} \quad E_f \varphi U \psi = E \varphi U (Fair \land \psi) \]

It remains to deal with \( A_f \varphi U \psi \).

Recall that \( A \varphi U \psi = \neg EG \neg \psi \lor \neg E \neg \psi U (\neg \varphi \land \neg \psi) \)

This formula also holds for the fair quantifications.

Hence, we only need to compute the semantics of \( E_f G \varphi \).

**Missing in this talk**

- Symbolic model checking for CTL using BDDs.
- \( \mu \)-calculus
- \( \ldots \)