Model Checking

Alcino Cunha

October 18, 2017

Kripke Structures

Definition

Let A be a set of atomic propositions. A *Kripke structure* is a tuple:

$$(S, I, R, L)$$

where

- S is a finite set of states.
- $I \subseteq S$ is the set of initial states.
- $R \subseteq S \times S$ is a total transition relation:

$$\forall s \in S \cdot s.R \neq \emptyset, \text{ where } s.R = \{s' \mid (s, s') \in R\}$$

- $L : S \to 2^A$ is a function that labels each state with the set of atomic propositions true in that state.

Kripke Structures

- A path in a structure $M = (S, I, R, L)$ is an infinite sequence of states $\pi = s_0 s_1 s_2 \ldots$, such that $\forall i \geqslant 0 \cdot (s_i, s_{i+1}) \in R$.
- Given a path π its i-th state will be denoted by π_i.
- The suffix of π starting at its i-th state will be denoted by π^i.
- Abusing the notation, we will usually denote the set of paths in M by M.

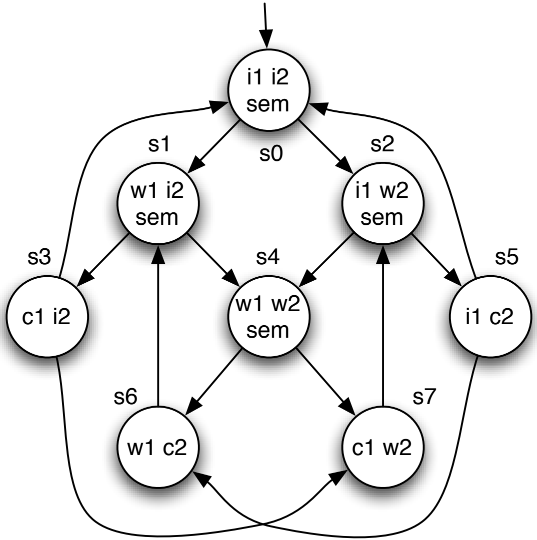
Mutual exclusion with a semaphore

```
while true:
  i1 : // iddle
  w1 : request sem
  c1 : // critical section
       release sem

||

while true:
  i2 : // iddle
  w2 : request sem
  c2 : // critical section
       release sem
```

Mutual exclusion with a semaphore



Introduction

- Properties of reactive systems usually fall under two categories:
  - **Safety** A safety property states that "bad things" do not happen.
  - **Liveness** A liveness property states that "good things" do happen (eventually).
- Most safety properties can be easily stated directly on Kripke structures. For example, mutual exclusion:

$$\{c_1, c_2\} \notin s_0.R^*$$

- But how to express safety properties like "an agent cannot be in its critical section without requesting it before"?

Introduction

- We can also state some animation properties directly on Kripke structures. For example, reversibility:

$$\forall s \in s_0.R^* \cdot s_0 \in s.R^*$$

- But how to express properties like evolution in mutual exclusion problems?
- We need a richer formalism in which to express properties that restrict the valid computations of the system.
- Temporal logic can be such formalism: although time is not mentioned explicitly, modal operators allow us to express rich causal orders within computations.
- Standard temporal logic is state oriented: the particular sequence of actions that lead to a computation is irrelevant.

Models of Time

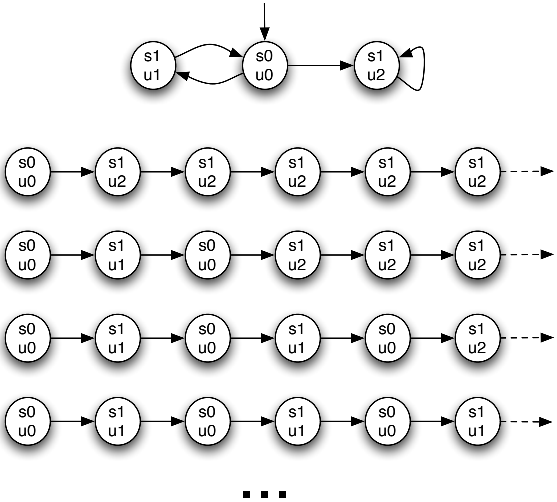
- There are two basic models of time in temporal logic:

  Linear Time The behavior of the system is the set of all infinite paths starting in initial states.

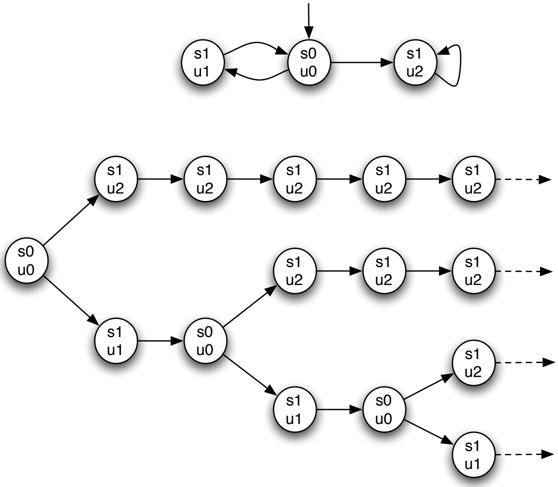
  Branching Time The behavior of the system is the set of all infinite computation trees unrolled from initial states.

- Both can be determined from a Kripke structure.

Linear Time


s1 u1
s0 u0
s1 u2
s0 u0 → s1 u2 → s1 u2 → s1 u2 → s1 u2 → s1 u2
s0 u0 → s1 u1 → s0 u0 → s1 u2 → s1 u2 → s1 u2
s0 u0 → s1 u1 → s0 u0 → s1 u1 → s0 u0 → s1 u2
s0 u0 → s1 u1 → s0 u0 → s1 u1 → s0 u0 → s1 u1
...

Branching Time



CTL

- *Computation Tree Logic* (CTL) is a branching time temporal logic.
- Besides classical operators, CTL has:

  Path quantifiers Used to describe the branching structure in the computation tree.

  Temporal operators Used to describe properties of a path through the tree.
- There are two type of formulas in CTL:

  State formulas Which are true in a specific state.

  Path formulas Which are true along a specific path.

Path Quantifiers and Temporal Operators

- Path quantifiers:

$\mathsf{A}\, f$	f holds for all computation paths.
$\mathsf{E}\, f$	f holds for some computation path.

- Temporal operators:

$\mathsf{X}\, f$	f holds in the next state.
$\mathsf{F}\, f$	Eventually (or in the future) f holds.
$\mathsf{G}\, f$	f always (or globally) holds.
$f\, \mathsf{U}\, g$	g eventually holds and until then f always holds.
$g\, \mathsf{R}\, f$	f holds up to a state where g holds, although g is not required to hold eventually.

- Temporal operators X, F, and G are sometimes denoted using $\bigcirc$, $\Diamond$, and $\Box$, respectively.

CTL Syntax

- Let A be the set of atomic propositions. State formulas are built from the following rules:
  - If $p \in A$, then p is a state formula.
  - If f and g are state formulas, then $\neg f$, $f \vee g$, $f \wedge g$, and $f \supset g$ are state formulas.
  - If f is a path formula, then $\mathrm{E}\, f$, and $\mathrm{A}\, f$ are state formulas.
- The syntax of path formulas is given by the following rule:
  - If f and g are state formulas, then $\mathrm{X}\, f$, $\mathrm{F}\, f$, $\mathrm{G}\, f$, $f\, \mathrm{U}\, g$, and $g\, \mathrm{R}\, f$ are path formulas.

CTL Semantics

- We will define the semantics of CTL with respect to a Kripke structure $M = (S, I, R, L)$.
- Given a state formula f we will denote the fact the f holds in M by $M \models f$.
- $M \models f$ if and only if for all initial state $s \in I$ we have $M, s \models f$ (see next slide).

Semantics of CTL State Formulas

- If f is a state formula, $M, s \models f$ means that f holds at state s in M. The relation $\models$ is defined inductively as follows (p is an atomic proposition, f and g are state formulas, and h is a path formula):

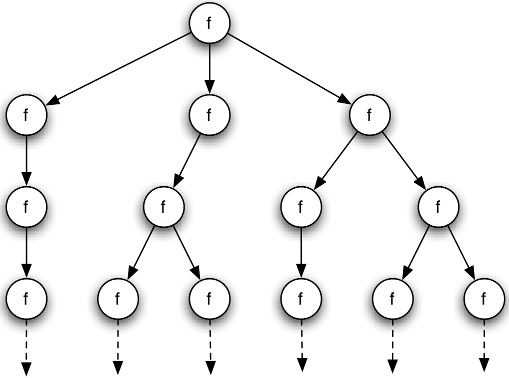
$$
\begin{array}{lcl}
M, s \models p & \Leftrightarrow & p \in L(s) \\
M, s \models \neg f & \Leftrightarrow & M, s \not\models f \\
M, s \models f \vee g & \Leftrightarrow & M, s \models f \vee M, s \models g \\
M, s \models f \wedge g & \Leftrightarrow & M, s \models f \wedge M, s \models g \\
M, s \models f \supset g & \Leftrightarrow & M, s \not\models f \vee M, s \models g \\
M, s \models \mathrm{A}\, h & \Leftrightarrow & \forall\, \pi \in M, \pi_0 = s \cdot M, \pi \models h \\
M, s \models \mathrm{E}\, h & \Leftrightarrow & \exists\, \pi \in M, \pi_0 = s \cdot M, \pi \models h
\end{array}
$$

Semantics of CTL Path Formulas

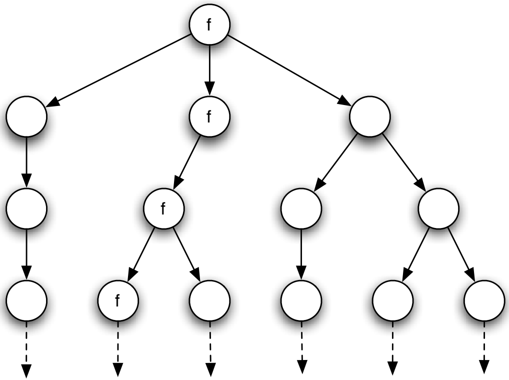
- If f is a path formula, $M, \pi \models f$ means that f holds along path π in M. The relation $\models$ is defined inductively as follows (f and g are state formulas):

$$
\begin{array}{lcl}
M, \pi \models \mathrm{X}\, f & \Leftrightarrow & M, \pi_1 \models f \\
M, \pi \models \mathrm{F}\, f & \Leftrightarrow & \exists i \geqslant 0 \cdot M, \pi_i \models f \\
M, \pi \models \mathrm{G}\, f & \Leftrightarrow & \forall i \geqslant 0 \cdot M, \pi_i \models f \\
M, \pi \models f \,\mathrm{U}\, g & \Leftrightarrow & \exists i \geqslant 0 \cdot M, \pi_i \models g \wedge \forall 0 \leqslant j < i \cdot M, \pi_j \models f \\
M, \pi \models g \,\mathrm{R}\, f & \Leftrightarrow & \forall i \geqslant 0 \cdot M, \pi_i \models f \vee \exists 0 \leqslant j < i \cdot M, \pi_j \models g
\end{array}
$$

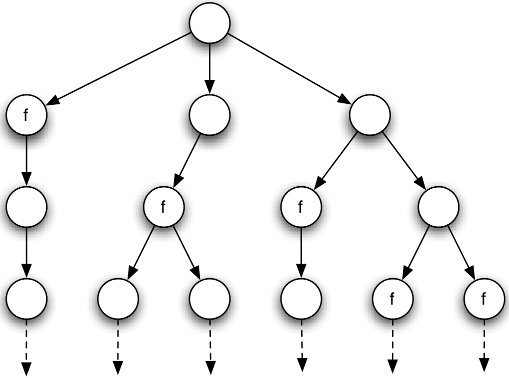
Basic CTL operators: AG f



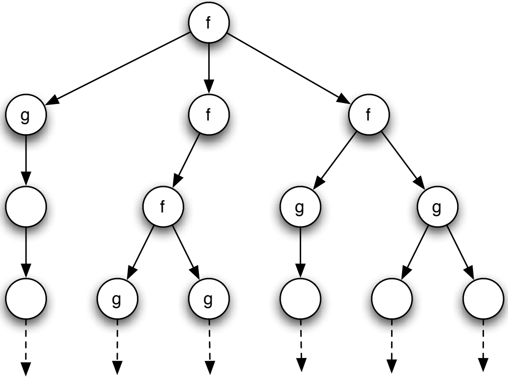
Basic CTL operators: EG f



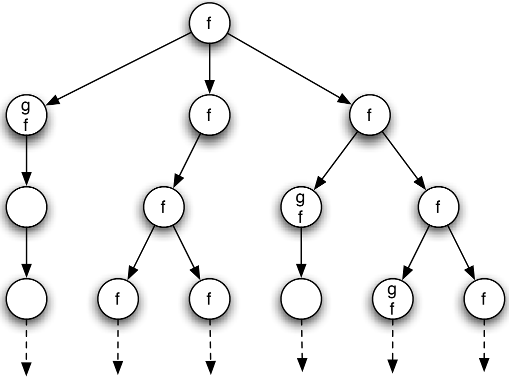
Basic CTL operators: AF f



Basic CTL operators: f AU g



Basic CTL operators: g AR f

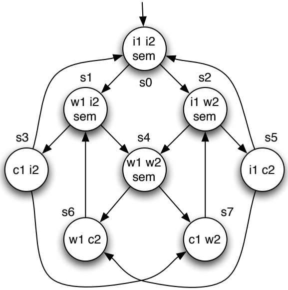


Minimal Set of CTL Operators

- All CTL formulas can be expressed using five operators: $\neg$, $\vee$, EX, EU e EG.

$$
\begin{aligned}
f \wedge g &\equiv \neg(\neg f \vee \neg g) \\
f \supset g &\equiv \neg f \vee g \\
\mathrm{AX}\, f &\equiv \neg\, \mathrm{EX} \neg f \\
\mathrm{EF}\, f &\equiv \mathit{true}\ \mathrm{EU}\ f \\
\mathrm{AG}\, f &\equiv \neg\, \mathrm{EF} \neg f \\
\mathrm{AF}\, f &\equiv \neg\, \mathrm{EG} \neg f \\
f\ \mathrm{AR}\ g &\equiv \neg(\neg f\ \mathrm{EU}\ \neg g) \\
f\ \mathrm{ER}\ g &\equiv \mathrm{EG}\, g \vee g\ \mathrm{EU}\ (f \wedge g) \\
f\ \mathrm{AU}\ g &\equiv \neg(\neg f\ \mathrm{ER}\ \neg g)
\end{aligned}
$$

Examples of CTL formulas



- Mutual exclusion: $\mathrm{AG}\,\neg(c_1 \wedge c_2)$
- Evolution: $\mathrm{AG}(w_1 \supset \mathrm{AF}\,c_1) \wedge \mathrm{AG}(w_2 \supset \mathrm{AF}\,c_2)$
- Reversibility: $\mathrm{AG}\,\mathrm{EF}(i_1 \wedge i_2 \wedge sem \wedge \neg w_1 \wedge \ldots)$
- No takeover: $\mathrm{AG}((w_1 \wedge i_2) \supset (c_1\,\mathrm{AR}\,\neg c_2)) \wedge \ldots$

LTL Syntax

- Unlike CTL, the *Linear Temporal Logic* LTL has no path quantifiers and all formulas are path formulas.
- Let A be the set of atomic propositions. The syntax of path formulas is given by the following rules:
  - If $p \in A$, then p is a path formula.
  - If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, and $f \supset g$, $\mathrm{X}\, f$, $\mathrm{F}\, f$, $\mathrm{G}\, f$, $f\ \mathrm{U}\ g$, and $g\ \mathrm{R}\ f$ are path formulas.

LTL Semantics

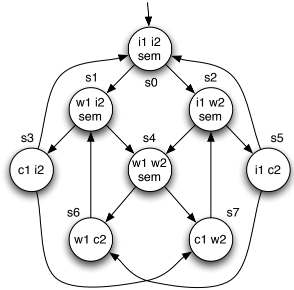
- We will define the semantics of LTL with respect to a Kripke structure $M = (S, I, R, L)$.
- Given a path formula f we will denote the fact the f holds in M by $M \models f$.
- $M \models f$ if and only if for all paths $\pi \in M$ such that $\pi_0 \in I$ we have $M, \pi \models f$ (see next slide).

Semantics of LTL Path Formulas

- If f is a path formula, $M, \pi \models f$ means that f holds along path π in M. The relation $\models$ is defined inductively as follows (p is an atomic proposition and f and g are path formulas):

$$
\begin{array}{lcl}
M, \pi \models p & \Leftrightarrow & p \in L(\pi_0) \\
M, \pi \models \neg f & \Leftrightarrow & M, \pi \not\models f \\
M, \pi \models f \vee g & \Leftrightarrow & M, \pi \models f \vee M, \pi \models g \\
M, \pi \models f \wedge g & \Leftrightarrow & M, \pi \models f \wedge M, \pi \models g \\
M, \pi \models f \supset g & \Leftrightarrow & M, \pi \not\models f \vee M, \pi \models g \\
M, \pi \models \mathrm{X}\, f & \Leftrightarrow & M, \pi^1 \models f \\
M, \pi \models \mathrm{F}\, f & \Leftrightarrow & \exists i \geqslant 0 \cdot M, \pi^i \models f \\
M, \pi \models \mathrm{G}\, f & \Leftrightarrow & \forall i \geqslant 0 \cdot M, \pi^i \models f \\
M, \pi \models f\, \mathrm{U}\, g & \Leftrightarrow & \exists i \geqslant 0 \cdot M, \pi^i \models g \wedge \forall 0 \leqslant j < i \cdot M, \pi^j \models f \\
M, \pi \models g\, \mathrm{R}\, f & \Leftrightarrow & \forall i \geqslant 0 \cdot M, \pi^i \models f \vee \exists 0 \leqslant j < i \cdot M, \pi^j \models g
\end{array}
$$

Examples of LTL formulas



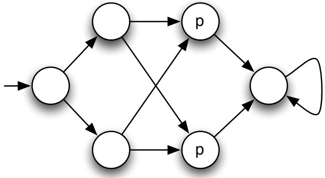
- Mutual exclusion: $\mathsf{G}\,\neg(c_1 \wedge c_2)$
- Evolution: $\mathsf{G}(w_1 \supset \mathsf{F}\,c_1) \wedge \mathsf{G}(w_2 \supset \mathsf{F}\,c_2)$
- No takeover: $\mathsf{G}((w_1 \wedge i_2) \supset (c_1\,\mathsf{R}\,\neg c_2)) \wedge \ldots$

LTL vs CTL

- Most properties can be expressed both in LTL and CTL, but the expressive power of both logics is incomparable.
- For example, reversibility cannot be expressed in LTL:

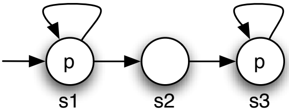
$$\mathrm{AG}\,\mathrm{EF}\,\mathit{init}$$

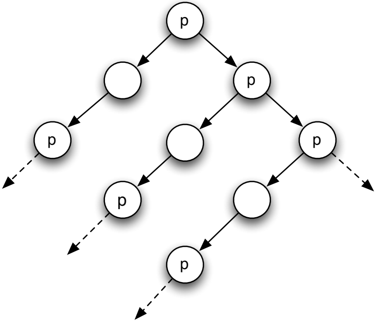
- LTL formulas are also not equivalent to the CTL formulas obtained by preceding each temporal operator by A. For example, $\mathrm{AF}\,\mathrm{AX}\,p$ and $\mathrm{F}\,\mathrm{X}\,p$ have different semantics.



LTL vs CTL

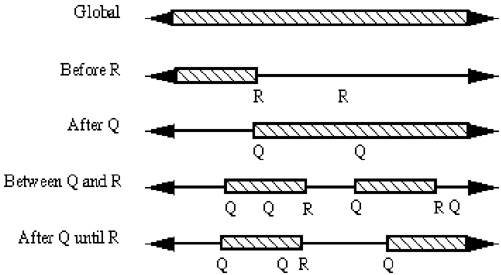
- Although a computation tree is more expressive than a set of computations, there are properties that can only be expressed in LTL.
- For example, $\mathsf{F}\,\mathsf{G}\,p$ cannot be expressed in CTL. Namely, its not equivalent to $\mathsf{AF}\,\mathsf{AG}\,p$.





Spec Patterns

`http://patterns.projects.cs.ksu.edu`



Model Checking

- We will focus on model checking techniques for CTL.
- Given a Kripke structure $M = (S, I, R, L)$ and a CTL formula f, the goal of model checking is to find the set of all states in M that satisfy f:

$$[\![f]\!]_M \equiv \{s \in S \mid M, s \models f\}$$

- Formula f holds in a model M iff it holds in its initial states:

$$M \models f \Leftrightarrow I \subseteq [\![f]\!]_M$$

- Two different approaches to model checking:
  - Explicit: Based on an explicit enumeration and traversal of the Kripke structure.
  - Symbolic: When the Kripke structure is implicitly modeled by propositional formulas.

Explicit Model Checking

- It suffices to handle six cases: atomic propositions and operators $\neg$, $\vee$, EX, EG, and EU.
- Given a Kripke structure $M = (S, I, R, L)$, an atomic proposition p, and state formulas f and g we have:

$$\begin{aligned} [\![p]\!]_M &= L^{-1}(p) = \{s \in S \mid p \in L(s)\} \\ [\![\neg f]\!]_M &= S - [\![f]\!]_M \\ [\![f \vee g]\!]_M &= [\![f]\!]_M \cup [\![g]\!]_M \end{aligned}$$

- The states that satisfy EX f are the predecessors of states that satisfy f:

$$[\![\mathrm{EX}\, f]\!]_M = R.[\![f]\!]_M = \{s \in S \mid \exists t \in [\![f]\!]_M \cdot (s, t) \in R\}$$

Explicit Model Checking EU

- To compute $[\![f \text{ EU } g]\!]$ we start from set $[\![g]\!]$ and successively add predecessors that satisfy f:

$$
\begin{array}{l}
\textbf{checkEU } ([\![f]\!], [\![g]\!]) \equiv \\
\quad T \leftarrow [\![g]\!]; \\
\quad [\![f \text{ EU } g]\!] \leftarrow [\![g]\!]; \\
\quad \textbf{while } T \neq \emptyset \\
\quad\quad \textbf{choose } s \in T; \\
\quad\quad T \leftarrow T - \{s\}; \\
\quad\quad \textbf{for } t \in R.s \\
\quad\quad\quad \textbf{if } t \notin [\![f \text{ EU } g]\!] \wedge t \in [\![f]\!] \\
\quad\quad\quad\quad [\![f \text{ EU } g]\!] \leftarrow [\![f \text{ EU } g]\!] \cup \{t\}; \\
\quad\quad\quad\quad T \leftarrow T \cup \{t\}; \\
\quad \textbf{return } [\![f \text{ EU } g]\!];
\end{array}
$$

Explicit Model Checking EG

- Given a Kripke structure $M = (S, I, R, L)$, to model check $\mathrm{EG}\, f$ it suffices to restrict M to the states that satisfy f:

$$M_f = (\llbracket f \rrbracket, I \cap \llbracket f \rrbracket, R \cap (\llbracket f \rrbracket \times \llbracket f \rrbracket), L \cap (\llbracket f \rrbracket \times \llbracket f \rrbracket))$$

Lemma

$M, s \models \mathrm{EG}\ f$ iff $s \in \llbracket f \rrbracket$ and there exists a path in M_f from s to some node t in a *nontrivial strongly connected component* of M_f.

- A SCC (*strongly connected component*) C is a maximal subgraph where every node is reachable from every other node along a directed path entirely contained in C.
- C is also *nontrivial* iff it has more than one node or it contains one node with a self-loop.

Explicit Model Checking EG

- To compute $[\![\text{EG}\, f]\!]$ we first compute all states belonging to nontrivial SCCs of M_f with function **scc** and successively add all predecessors in $[\![f]\!]$.
- $\mathbf{scc}(M_f)$ can be computed efficiently with Tarjan's algorithm.

$$
\begin{array}{l}
\mathbf{checkG}\ ([\![f]\!]) \equiv \\
\quad T \leftarrow \cup\{C \mid C \in \mathbf{scc}(M_f) \wedge \neg\mathbf{trivial}(C)\}; \\
\quad [\![\text{EG}\, f]\!] \leftarrow T; \\
\quad \mathbf{while}\ \ T \neq \emptyset \\
\qquad \mathbf{choose}\ \ s \in T; \\
\qquad T \leftarrow T - \{s\}; \\
\qquad \mathbf{for}\ \ t \in R_f.s \\
\qquad\quad \mathbf{if}\ \ t \notin [\![\text{EG}\, f]\!] \\
\qquad\qquad [\![\text{EG}\, f]\!] \leftarrow [\![\text{EG}\, f]\!] \cup \{t\}; \\
\qquad\qquad T \leftarrow T \cup \{t\}; \\
\quad \mathbf{return}\ \ [\![\text{EG}\, f]\!];
\end{array}
$$

Fairness

- Some liveness properties can only be satisfied assuming that some kind of fairness holds in the system.
- For example, evolution in mutual exclusion algorithms usually only holds if we assume the scheduler is fair to the processes, i.e., all processes have the opportunity to execute once in a while.
- A fairness constraint can usually be specified with formula f that is required to hold infinitely often in valid execution paths.
- The verification of specification g under such fairness constraint f can be directly expressed in LTL as $(\mathsf{G}\,\mathsf{F}\,f) \supset g$.
- Unfortunately it cannot be expressed in CTL.

Explicit Model Checking CTL with Fairness

- To model check the operator EG under fairness it suffices to restrict the model to fair SCCs. A SCC is fair if $C \cap [\![f]\!] \neq \emptyset$.
- Now the formula EG *true* only holds in a state s iff there is a fair path starting from s.
- Given that a path is fair iff any of its suffixes is fair, we can model check f EU g under fairness by invoking the standard model checking procedure as follows:

$$\textbf{checkEU}([\![f]\!], [\![g]\!] \cap [\![\mathrm{EG}\, \mathit{true}]\!])$$

- Similarly for the remaining operators.

Symbolic Model Checking

- Although explicit model checking is rather efficient it cannot cope with the state explosion that occurs in many reactive systems.
- Symbolic model checking tackles this problem by avoiding the explicit construction of the state space: the states and the transition relation of a Kripke structure are captured by propositional formulas, defined over the variables that encode the state of the model.
- Model checking is reduced to checking the validity and equivalence of propositional formulas.
- These can be done very efficiently by using techniques like *Ordered Binary Decision Diagrams*.

Symbolic Model Checking for CTL

- The set of states $[\![f]\!]$ where a formula f is valid is no longer represented extensionally: instead it is represented by a propositional formula that is valid precisely in those states.
- For temporal operators EG and EU it can be determined by fixpoints based on the respective expansion laws:

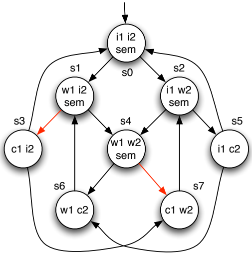
$$EGf \equiv f \wedge \mathrm{EX}\,\mathrm{EG}\,f$$
$$[\![\mathrm{EG}\,f]\!] = \nu(\Pi), \text{ where } \Pi(h) = [\![f]\!] \wedge [\![\mathrm{EX}\,h]\!]$$

$$f\ \mathrm{EU}\ g \equiv g \vee (f \wedge \mathrm{EX}(f\ \mathrm{EU}\ g))$$
$$[\![f\ \mathrm{EU}\ g]\!] = \mu(\Pi), \text{ where } \Pi(h) = [\![g]\!] \vee ([\![f]\!]_M \wedge [\![\mathrm{EX}\,h]\!])$$

- Notice that fixpoints are computed symbolically: for example, to compute a least fixpoint we start with formula *false* and perform disjunctions until two equivalent formulas are computed in successive iterations.

Encoding Transitions

- The transitions R of a model M can encoded by a formula ϕ_R that mentions normal variables to denote their value in the pre-state and primed versions to denote the value in the post-state.



$$
\begin{aligned}
in_1 \equiv\ & w_1 \wedge sem \wedge \neg w_1' \wedge c_1' \wedge \neg sem' \\
& \wedge \\
& i_1' = i_1 \wedge i_2' = i_2 \wedge w_2' = w_2 \wedge c_2' = c_2 \\
\ldots &
\end{aligned}
$$

$$\phi_R \equiv req_1 \vee in_1 \vee out_1 \vee req_2 \vee in_2 \vee out_2$$

Symbolic Model Checking EX

- To model check EX f a temporary existential quantifier is used.

$$[\![\mathrm{EX}\, f]\!] = \exists \overline{x}' \cdot [\![f]\!]' \wedge \phi_R$$

- $[\![f]\!]'$ is the formula obtained from $[\![f]\!]$ by replacing all variables by the corresponding primed version.
- Intuitively, the formula $[\![\mathrm{EX}\, f]\!]$ will be valid in a state s if there is some valuation to the primed variables that is accessible from s and for which f is valid.
- The existential quantifier is then eliminated by expansion. For example, for boolean variables we have:

$$\exists x \cdot f \equiv f|_{x \leftarrow true} \vee f|_{x \leftarrow false}$$

Symbolic Model Checking EX

$$\phi_R \equiv (a \wedge \neg b \wedge \neg a' \wedge b') \vee (\neg a \wedge b \wedge \neg a' \wedge b')$$

$$\begin{aligned}
[\![EX\ \ b]\!] &\equiv \exists a', b' \cdot \phi_R \wedge b' \\
&\equiv \exists a', b' \cdot \phi_R \\
&\equiv \exists a' \cdot \phi_R|_{b' \leftarrow true} \vee \phi_R|_{b' \leftarrow false} \\
&\equiv \exists a' \cdot (a \wedge \neg b \wedge \neg a') \vee (\neg a \wedge b \wedge \neg a') \\
&\equiv (a \wedge \neg b) \vee (\neg a \wedge b)
\end{aligned}$$

$$\begin{aligned}
[\![EX\ \ a]\!] &\equiv \exists a', b' \cdot \phi_R \wedge a' \\
&\equiv \exists a', b' \cdot (a \wedge \neg b \wedge \neg a' \wedge b' \wedge a') \vee \ldots \\
&\equiv false
\end{aligned}$$

Bibliography

- C. Baier and J.-P. Katoen. Principles of Model Checking. MIT Press, 2008.
- Robert Tarjan. Depth-First Search and Linear Graph Algorithms. SIAM Journal of Computing 1: 146–160. 1972.
- Edmund M. Clarke, E. Allen Emerson, A. Prasad Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. ACM TOPLAS 8(2): 244-263. 1986.
- Randal E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers* C-35(8):677–691. 1986.
- Kenneth L. McMillan. *Symbolic Model Checking*. Springer, 1993.