Protocol Verification and State Space Methods

Lars M. Kristensen[1] and Wojtek Penczek[2]

Department of Computer Engineering
Bergen University College, NORWAY

Institute of Computer Science, PAS, and University of Podlasie, Poland

Advanced Course on Petri Nets, Rostock, September 2010

Outline

Lecture 2

Introduction to model checking

Outline

- Standard non-symbolic model checking algorithms for CTL and LTL.
- Partial order reductions for $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}_{-\mathrm{X}}$.
- Introduction to symbolic model checking for CTL.
- BDD- and SAT-based model checking.

Lecture 2

Introduction to model checking

Outline

- Standard non-symbolic model checking algorithms for CTL and LTL.
- Partial order reductions for $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}_{-\mathrm{X}}$.
- Introduction to symbolic model checking for CTL.
- BDD- and SAT-based model checking.

Lecture 2

Introduction to model checking

Outline

- Standard non-symbolic model checking algorithms for CTL and LTL.
- Partial order reductions for $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}_{-\mathrm{X}}$.
- Introduction to symbolic model checking for CTL.
- BDD- and SAT-based model checking.

Lecture 2

Introduction to model checking

Outline

- Standard non-symbolic model checking algorithms for CTL and LTL.
- Partial order reductions for $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}_{-\mathrm{X}}$.
- Introduction to symbolic model checking for CTL.
- BDD- and SAT-based model checking.

Lecture 2

Introduction to model checking

Outline

- Standard non-symbolic model checking algorithms for CTL and LTL.
- Partial order reductions for $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}_{-\mathrm{X}}$.
- Introduction to symbolic model checking for CTL.
- BDD- and SAT-based model checking.

Lecture 2

Introduction to model checking

Outline

- Standard non-symbolic model checking algorithms for CTL and LTL.
- Partial order reductions for $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}_{-\mathrm{X}}$.
- Introduction to symbolic model checking for CTL.
- BDD- and SAT-based model checking.

Introduction to model checking

Standard non-symbolic model checking algorithms for CTL and LTL.

Model checking for Kripke models

Model checking problem

$$M \overset{?}{\models} \varphi$$

a Kripke model a modal formula

Syntax of CTL*

Syntax

S1. every member of $\mathcal{PV}$ is a state formula,

S2. if φ and ψ are state formulas, then so are $\neg\varphi$ and $\varphi \wedge \psi$,

S3. if φ is a path formula, then $A\varphi$ and $\mathrm{E}\varphi$, are state formulas,

P1. any state formula φ is also a path formula,

P2. if φ, ψ are path formulas, then so are $\varphi \wedge \psi$ and $\neg\varphi$,

P3. if φ, ψ are path formulas, then so is $\mathrm{X}\varphi$, $\mathrm{G}\varphi$, and $\varphi \mathrm{U} \psi$.

CTL* consists of the set of all state formulae.

Variety of sublogics of CTL^*

Definition

- $\mathrm{LTL} \subset \mathrm{CTL}^*$ is the fragment of CTL^* in which all modal formulas are of the form $A\varphi$, where φ does not contain the state modalities A, E.
- $\mathrm{CTL} \subset \mathrm{CTL}^*$ is the fragment of CTL^* in which A, E, and the path modalities U and G may only appear paired: $A\mathrm{X}$, EX, $A\mathrm{U}$, EU, $A\mathrm{G}$, and EG.

Semantics of CTL^*

$M = (G, \iota, \Pi, V)$ - a model and $\pi = g_0 a_0 g_1 \cdots$ - an infinite path of G.
π_i denotes the suffix $g_i a_i g_{i+1} \cdots$ of π

S1. $g \models q$ iff $q \in V(g)$, for $q \in PV$,

S2. $g \models \neg\varphi$ iff not $g \models \varphi$,
$g \models \varphi \wedge \psi$ iff $g \models \varphi$ and $g \models \psi$,

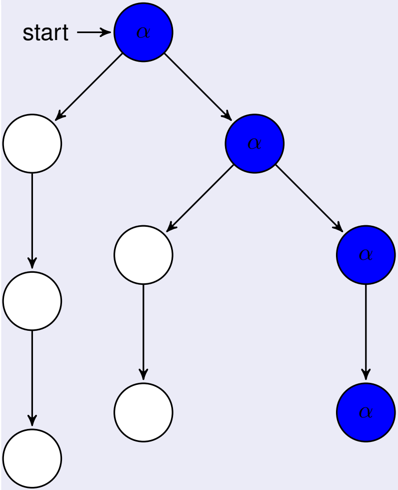
S3. $g \models A\varphi$ iff $\pi \models \varphi$ for every path π starting at g,
$g \models \mathrm{E}\varphi$ iff $\pi \models \varphi$ for some path π starting at g,

P1. $\pi \models \varphi$ iff $g_0 \models \varphi$ for any state formula φ,

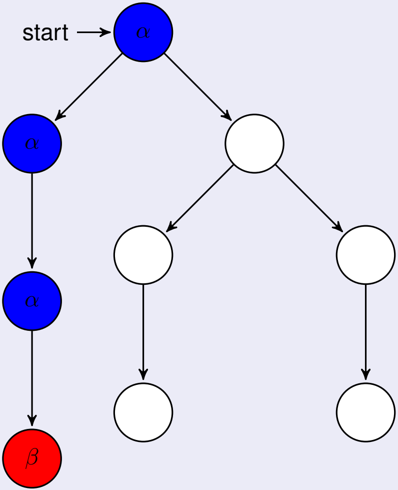
P2. $\pi \models \neg\varphi$ iff not $\pi \models \varphi$,
$\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$,

P3. $\pi \models \mathrm{X}\varphi$ iff $\pi_1 \models \varphi$,
$\pi \models \mathrm{G}\varphi$ iff $\pi_j \models \varphi$ for all $j \geq 0$,
$\pi \models \varphi \mathrm{U} \psi$ iff there is an $i \geq 0$ such that $\pi_i \models \psi$ and $\pi_j \models \varphi$ for all $0 \leq j < i$.

Semantics in Examples

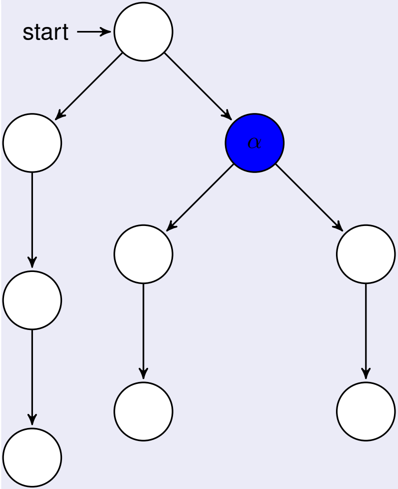


$M, start \models \mathrm{EG}\alpha$

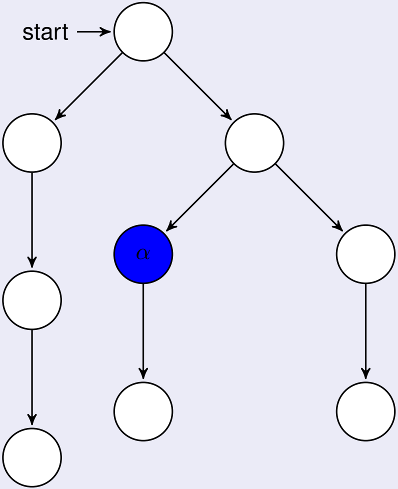


$M, start \models E\alpha\mathrm{U}\beta$

Semantics in Examples



$M, start \models EX\alpha$



$M, start \models EF\alpha$

Model checking CTL by state labeling

State labelling

If we do not bother about the size of a model, then the simplest approach to CTL model checking, called state labelling, can be used.

Algorithm

We show a deterministic algorithm, based on state labelling, for determining whether a CTL formula φ is true at a state $s \in S$ in a finite model $M = ((S, s^0, \to), V)$, of time complexity $O(|\varphi| \times (|S| + |\to|))$.

Model checking CTL by state labeling

State labelling

If we do not bother about the size of a model, then the simplest approach to CTL model checking, called state labelling, can be used.

Algorithm

We show a deterministic algorithm, based on state labelling, for determining whether a CTL formula φ is true at a state $s \in S$ in a finite model $M = ((S, s^0, \rightarrow), V)$, of time complexity $O(|\varphi| \times (|S| + |\rightarrow|))$.

Model checking CTL by state labeling

Algorithm

The algorithm is designed so that when it finishes, each state *s* of *M* is labelled with the subformulas of φ which are true at *s*.

- The algorithm operates in stages.
- The *i*-th stage handles all subformulas of φ of length *i* for $i \leq |\varphi|$.
- Thus, at the end of the last stage each state will be labelled with all subformulas of φ which are true at it.

Model checking CTL by state labeling

Algorithm

The algorithm is designed so that when it finishes, each state *s* of *M* is labelled with the subformulas of φ which are true at *s*.

- The algorithm operates in stages.
- The *i*-th stage handles all subformulas of φ of length *i* for $i \leq |\varphi|$.
- Thus, at the end of the last stage each state will be labelled with all subformulas of φ which are true at it.

Model checking CTL by state labeling

Algorithm

The algorithm is designed so that when it finishes, each state *s* of *M* is labelled with the subformulas of φ which are true at *s*.

- The algorithm operates in stages.
- The *i*-th stage handles all subformulas of φ of length *i* for $i \leq |\varphi|$.
- Thus, at the end of the last stage each state will be labelled with all subformulas of φ which are true at it.

Model checking CTL by state labeling

Algorithm

The algorithm is designed so that when it finishes, each state s of M is labelled with the subformulas of φ which are true at s.

- The algorithm operates in stages.
- The i-th stage handles all subformulas of φ of length i for $i \leq |\varphi|$.
- Thus, at the end of the last stage each state will be labelled with all subformulas of φ which are true at it.

Model checking CTL by state labeling

CTL operators

Each of the operators of CTL can be expressed in terms of the three operators EX, EG, and EU.

Five cases

So, only 5 cases have to be considered, where φ is:

$\neg\psi$,

$\psi_1 \wedge \psi_2$,

$\mathrm{EX}\psi$,

$\mathrm{E}(\psi_1 \mathrm{U} \psi_2)$, or

$\mathrm{EG}\psi$.

Algorithm

The algorithm is discussed for the last two cases only, as the others are straightforward.

Model checking CTL by state labeling

CTL operators

Each of the operators of CTL can be expressed in terms of the three operators EX, EG, and EU.

Five cases

So, only 5 cases have to be considered, where φ is:
$\neg\psi$,
$\psi_1 \wedge \psi_2$,
$\mathrm{EX}\psi$,
$\mathrm{E}(\psi_1 \mathrm{U} \psi_2)$, or
$\mathrm{EG}\psi$.

Algorithm

The algorithm is discussed for the last two cases only, as the others are straightforward.

Model checking CTL by state labeling

CTL operators

Each of the operators of CTL can be expressed in terms of the three operators EX, EG, and EU.

Five cases

So, only 5 cases have to be considered, where φ is:
$\neg\psi$,
$\psi_1 \wedge \psi_2$,
$\mathrm{EX}\psi$,
$\mathrm{E}(\psi_1 \mathrm{U} \psi_2)$, or
$\mathrm{EG}\psi$.

Algorithm

The algorithm is discussed for the last two cases only, as the others are straightforward.

Model checking CTL by state labeling

Formula $\varphi = \mathrm{E}(\psi_1 \mathrm{U} \psi_2)$

- The algorithm first finds all the states which are labelled with ψ_2 and labels them with φ.
- It goes backwards using the relation $\rightarrow^{-1}$ and finds all the states which can be reached by a path in which each state is labelled with ψ_1.
  All such states are labelled with φ.

Complexity

This step requires time $O(|S| + |\rightarrow|)$.

Model checking CTL by state labeling

Formula $\varphi = \mathrm{E}(\psi_1 \mathrm{U} \psi_2)$

- The algorithm first finds all the states which are labelled with ψ_2 and labels them with φ.
- It goes backwards using the relation $\rightarrow^{-1}$ and finds all the states which can be reached by a path in which each state is labelled with ψ_1.
  All such states are labelled with φ.

Complexity

This step requires time $O(|S| + |\rightarrow|)$.

Model checking CTL by state labeling

Formula $\varphi = \mathrm{E}(\psi_1 \mathrm{U} \psi_2)$

- The algorithm first finds all the states which are labelled with ψ_2 and labels them with φ.
- It goes backwards using the relation $\rightarrow^{-1}$ and finds all the states which can be reached by a path in which each state is labelled with ψ_1.
  All such states are labelled with φ.

Complexity

This step requires time $O(|S| + |\rightarrow|)$.

Model checking CTL by state labeling

Formula $\varphi = \mathrm{EG}\psi$

- The graph $(S', \rightarrow')$ is constructed, where $S' = \{s \in S \mid M, s \models \psi\}$ and $\rightarrow' = \rightarrow \cap (S' \times S')$.
- $(S', \rightarrow')$ is partitioned into strongly connected components and those states which belong to the components of size greater than 1 or with a self-loop are selected and labelled with φ.
- The algorithm goes backwards from these states using $\rightarrow^{-1}$ and finds all those states which can be reached by a path in which each state is labelled with ψ. It labels these states with φ.

Complexity

This step requires time $O(|S| + |\rightarrow|)$.

Model checking CTL by state labeling

Formula $\varphi = \mathrm{EG}\psi$

- The graph $(S', \to')$ is constructed, where $S' = \{s \in S \mid M, s \models \psi\}$ and $\to' = \to \cap (S' \times S')$.
- $(S', \to')$ is partitioned into strongly connected components and those states which belong to the components of size greater than 1 or with a self-loop are selected and labelled with φ.
- The algorithm goes backwards from these states using $\to^{-1}$ and finds all those states which can be reached by a path in which each state is labelled with ψ. It labels these states with φ.

Complexity

This step requires time $O(|S| + |\to|)$.

Model checking CTL by state labeling

Formula $\varphi = \mathrm{EG}\psi$

- The graph $(S', \rightarrow')$ is constructed, where $S' = \{s \in S \mid M, s \models \psi\}$ and $\rightarrow' = \rightarrow \cap (S' \times S')$.
- $(S', \rightarrow')$ is partitioned into strongly connected components and those states which belong to the components of size greater than 1 or with a self-loop are selected and labelled with φ.
- The algorithm goes backwards from these states using $\rightarrow^{-1}$ and finds all those states which can be reached by a path in which each state is labelled with ψ. It labels these states with φ.

Complexity

This step requires time $O(|S| + |\rightarrow|)$.

Model checking CTL by state labeling

Formula $\varphi = \mathrm{EG}\psi$

- The graph $(S', \to')$ is constructed, where $S' = \{s \in S \mid M, s \models \psi\}$ and $\to' = \to \cap (S' \times S')$.
- $(S', \to')$ is partitioned into strongly connected components and those states which belong to the components of size greater than 1 or with a self-loop are selected and labelled with φ.
- The algorithm goes backwards from these states using $\to^{-1}$ and finds all those states which can be reached by a path in which each state is labelled with ψ. It labels these states with φ.

Complexity

This step requires time $O(|S| + |\to|)$.

Model checking CTL by state labeling

Example

Consider the model **M** shown below and the CTL formula
$\varphi = \mathrm{E}(p_1 \mathrm{U}(\mathrm{EG}p_2))$.

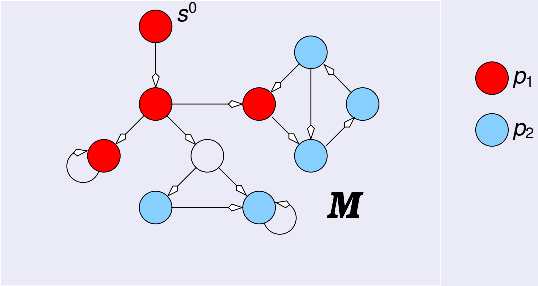
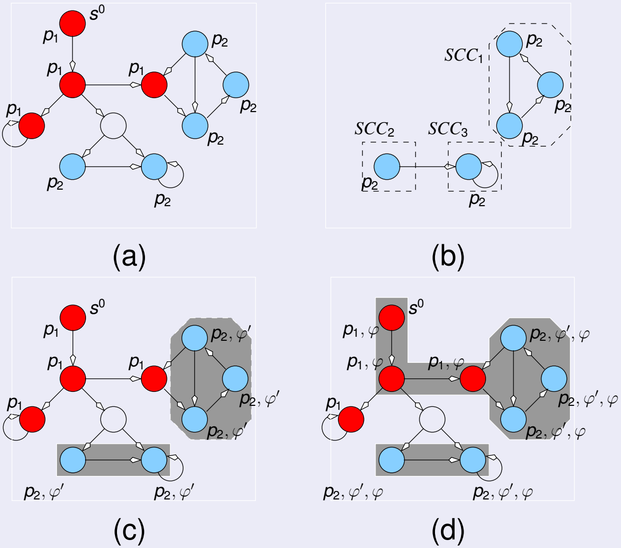


Figure: The model **M**.

Model checking CTL by state labeling

Labelling M with $\varphi = \mathrm{E}(p_1 \mathrm{U}(\mathrm{EG}p_2))$; $\varphi' = \mathrm{EG}p_2$



References and other approaches

Reference

The original state labelling algoritm for CTL was introduced by Clarke, Emerson, and Sistla in 1986.

Automata theoretic approaches

- By checking non-emptiness of the product of the automaton representing a system and an automaton accepting all the models of the negation of a formula, via ...
- A translation from CTL to alternating tree automata.
- A translation from LTL to Buchi or Streett automata.

References and other approaches

Reference

The original state labelling algoritm for CTL was introduced by Clarke, Emerson, and Sistla in 1986.

Automata theoretic approaches

- By checking non-emptiness of the product of the automaton representing a system and an automaton accepting all the models of the negation of a formula, via ...
- A translation from CTL to alternating tree automata.
- A translation from LTL to Buchi or Streett automata.

References and other approaches

Reference

The original state labelling algoritm for CTL was introduced by Clarke, Emerson, and Sistla in 1986.

Automata theoretic approaches

- By checking non-emptiness of the product of the automaton representing a system and an automaton accepting all the models of the negation of a formula, via ...
- A translation from CTL to alternating tree automata.
- A translation from LTL to Buchi or Streett automata.

Introduction to model checking for knowledge and time

Partial order reductions for $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}_{-\mathrm{X}}$.

Networks of automata

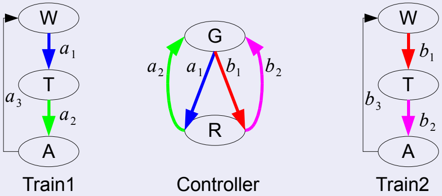


Figure: TC composed of two trains and the controler

Algorithm DFS-POR

DFS-POR

DFS-POR is used to compute paths of the reduced model.
A stack represents the path $\pi = g_0 a_0 g_1 a_1 \cdots g_n$ currently being visited.
For g_n, the following three operations are computed in a loop:

1. The set $en(g_n) \subseteq Act$ of enabled actions is identified and a subset $E(g_n) \subseteq en(g_n)$ of possible actions is heuristically selected.
2. For any action $a \in E(g_n)$ compute the successor state g' of g_n such that $g_n \xrightarrow{a} g'$, and add g' to the stack.
   Recursively proceed to explore the submodel originating at g'.
3. Remove g_n from the stack.

Algorithm DFS-POR

DFS-POR

DFS-POR is used to compute paths of the reduced model.
A stack represents the path $\pi = g_0 a_0 g_1 a_1 \cdots g_n$ currently being visited.
For g_n, the following three operations are computed in a loop:

1. The set $en(g_n) \subseteq Act$ of enabled actions is identified and a subset $E(g_n) \subseteq en(g_n)$ of possible actions is heuristically selected.
2. For any action $a \in E(g_n)$ compute the successor state g' of g_n such that $g_n \xrightarrow{a} g'$, and add g' to the stack.
   Recursively proceed to explore the submodel originating at g'.
3. Remove g_n from the stack.

Algorithm DFS-POR

DFS-POR

DFS-POR is used to compute paths of the reduced model.
A stack represents the path $\pi = g_0 a_0 g_1 a_1 \cdots g_n$ currently being visited.
For g_n, the following three operations are computed in a loop:

1. The set $en(g_n) \subseteq Act$ of enabled actions is identified and a subset $E(g_n) \subseteq en(g_n)$ of possible actions is heuristically selected.
2. For any action $a \in E(g_n)$ compute the successor state g' of g_n such that $g_n \xrightarrow{a} g'$, and add g' to the stack.
   Recursively proceed to explore the submodel originating at g'.
3. Remove g_n from the stack.

Algorithm DFS-POR

DFS-POR

DFS-POR is used to compute paths of the reduced model.
A stack represents the path $\pi = g_0 a_0 g_1 a_1 \cdots g_n$ currently being visited.
For g_n, the following three operations are computed in a loop:

1. The set $en(g_n) \subseteq Act$ of enabled actions is identified and a subset $E(g_n) \subseteq en(g_n)$ of possible actions is heuristically selected.
2. For any action $a \in E(g_n)$ compute the successor state g' of g_n such that $g_n \xrightarrow{a} g'$, and add g' to the stack.
   Recursively proceed to explore the submodel originating at g'.
3. Remove g_n from the stack.

Conditions for selection of $E(g)$

Conditions

C1 No action $a \in Act \setminus E(g)$ that is dependent on an action in $E(g)$ can be executed before an action in $E(g)$ is executed.

C2 On every cycle in the constructed state graph there is at least one node g for which $E(g) = en(g)$.

C3 Each action in $E(g)$ is invisible, i.e., does not change $V(g)$.

C4 If $E(g) \neq en(g)$, then $E(g)$ is a singleton.

Conditions for selection of $E(g)$

Conditions

C1 No action $a \in Act \setminus E(g)$ that is dependent on an action in $E(g)$ can be executed before an action in $E(g)$ is executed.

C2 On every cycle in the constructed state graph there is at least one node g for which $E(g) = en(g)$.

C3 Each action in $E(g)$ is invisible, i.e., does not change $V(g)$.

C4 If $E(g) \neq en(g)$, then $E(g)$ is a singleton.

Conditions for selection of $E(g)$

Conditions

C1 No action $a \in Act \setminus E(g)$ that is dependent on an action in $E(g)$ can be executed before an action in $E(g)$ is executed.

C2 On every cycle in the constructed state graph there is at least one node g for which $E(g) = en(g)$.

C3 Each action in $E(g)$ is invisible, i.e., does not change $V(g)$.

C4 If $E(g) \neq en(g)$, then $E(g)$ is a singleton.

Conditions for selection of $E(g)$

Conditions

C1 No action $a \in Act \setminus E(g)$ that is dependent on an action in $E(g)$ can be executed before an action in $E(g)$ is executed.

C2 On every cycle in the constructed state graph there is at least one node g for which $E(g) = en(g)$.

C3 Each action in $E(g)$ is invisible, i.e., does not change $V(g)$.

C4 If $E(g) \neq en(g)$, then $E(g)$ is a singleton.

Correctness

Theorem

Let M be a model and $M' \subseteq M$ be the reduced model generated by the DFS-POR algorithm. The following conditions hold:

a) *If the choice of $E(g)$ is given by* **C1, C2, C3**, *then $M \models \varphi$ iff $M' \models \varphi$, for any $\mathrm{LTL}_{-\mathrm{X}}$ formula φ.*

b) *If the choice of $E(g)$ is given by* **C1, C2, C3** *and* **C4**, *then $M \models \varphi$ iff $M' \models \varphi$, for any $\mathrm{CTL}^*_{-\mathrm{X}}$ formula φ.*

Correctness

Theorem

Let M be a model and $M' \subseteq M$ be the reduced model generated by the DFS-POR algorithm. The following conditions hold:

a) *If the choice of $E(g)$ is given by* **C1, C2, C3**, *then $M \models \varphi$ iff $M' \models \varphi$, for any $\mathrm{LTL}_{-\mathrm{X}}$ formula φ.*

b) *If the choice of $E(g)$ is given by* **C1, C2, C3** *and* **C4**, *then $M \models \varphi$ iff $M' \models \varphi$, for any $\mathrm{CTL}^*_{-\mathrm{X}}$ formula φ.*

Correctness

Theorem

Let M be a model and $M' \subseteq M$ be the reduced model generated by the DFS-POR algorithm. The following conditions hold:

a) *If the choice of $E(g)$ is given by* **C1, C2, C3**, *then $M \models \varphi$ iff $M' \models \varphi$, for any $\mathrm{LTL}_{-\mathrm{X}}$ formula φ.*

b) *If the choice of $E(g)$ is given by* **C1, C2, C3** *and* **C4**, *then $M \models \varphi$ iff $M' \models \varphi$, for any $\mathrm{CTL}^*_{-\mathrm{X}}$ formula φ.*

Experimental Results - Trains and controler (TC)

TC

Property: if the train 1 is in the tunnel, then no other train is in the tunnel at the same time:

$$AG(\text{in\_tunnel}_1 \to \bigwedge_{i=2}^{n} \neg\text{in\_tunnel}_i),$$

State spaces

$F(n)$ - the size of the full state space.
$R(n)$ - the size of the reduced state space.

- $F(n) = c_n \times 2^{n+1}$, for some $c_n > 1$,
- $R(n) = 3 + 4(n-1)$.

The reduced state space is *exponentially smaller* than the original one, for both LTL_{-X} and CTL^*_{-X}.

Experimental Results - Trains and controler (TC)

TC

Property: if the train 1 is in the tunnel, then no other train is in the tunnel at the same time:

$$AG(\text{in\_tunnel}_1 \rightarrow \bigwedge_{i=2}^{n} \neg \text{in\_tunnel}_i),$$

State spaces

$F(n)$ - the size of the full state space.
$R(n)$ - the size of the reduced state space.

- $F(n) = c_n \times 2^{n+1}$, for some $c_n > 1$,
- $R(n) = 3 + 4(n-1)$.

The reduced state space is *exponentially smaller* than the original one, for both $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}^*_{-\mathrm{X}}$.

Experimental Results - Trains and controler (TC)

TC

Property: if the train 1 is in the tunnel, then no other train is in the tunnel at the same time:

$$AG(\text{in\_tunnel}_1 \to \bigwedge_{i=2}^{n} \neg\text{in\_tunnel}_i),$$

State spaces

$F(n)$ - the size of the full state space.
$R(n)$ - the size of the reduced state space.

- $F(n) = c_n \times 2^{n+1}$, for some $c_n > 1$,
- $R(n) = 3 + 4(n-1)$.

The reduced state space is *exponentially smaller* than the original one, for both $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}^*_{-\mathrm{X}}$.

Introduction to model checking

Introduction to symbolic model checking for CTL.

Fixed-point verification for CTL

Introduction

Symbolic and non-symbolic model checking methods can exploit the fixed point characterization of CTL formulas.

Using fixpoints

Labelling the states with the subformulas or computation of OBDD representation of a formula uses the standard algorithms for computing the least and the greatest fixpoints as follows.

Fixed-point verification for CTL

Introduction

Symbolic and non-symbolic model checking methods can exploit the fixed point characterization of CTL formulas.

Using fixpoints

Labelling the states with the subformulas or computation of OBDD representation of a formula uses the standard algorithms for computing the least and the greatest fixpoints as follows.

Fixed points approach

Axioms for CTL

- $\mathrm{EG}\varphi \equiv \varphi \wedge \mathrm{EXEG}\varphi$,
- $\mathrm{E}(\varphi \mathrm{U} \psi) \equiv \psi \vee (\varphi \wedge \mathrm{EX}(\mathrm{E}(\varphi \mathrm{U} \psi)))$,

Fixed point characterization of CTL

Let $[\![\varphi]\!] = \{s \in S \mid s \models \varphi\}$.

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap [\![\mathrm{EXEG}\varphi]\!]$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap [\![\mathrm{EXE}(\varphi \mathrm{U} \psi)]\!])$

Pre-set

Let $pre(X) = \{s \in S \mid (\exists s' \in X)\ s \to s'\}$, for $X \subseteq S$.

Characterization

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap pre([\![\mathrm{EG}\varphi]\!])$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre([\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]))$.

Fixed points approach

Axioms for CTL

- $\mathrm{EG}\varphi \equiv \varphi \wedge \mathrm{EXEG}\varphi$,
- $\mathrm{E}(\varphi \mathrm{U} \psi) \equiv \psi \vee (\varphi \wedge \mathrm{EX}(\mathrm{E}(\varphi \mathrm{U} \psi)))$,

Fixed point characterization of CTL

Let $[\![\varphi]\!] = \{s \in S \mid s \models \varphi\}$.

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap [\![\mathrm{EXEG}\varphi]\!]$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap [\![\mathrm{EXE}(\varphi \mathrm{U} \psi)]\!])$

Pre-set

Let $pre(X) = \{s \in S \mid (\exists s' \in X)\ s \rightarrow s'\}$, for $X \subseteq S$.

Characterization

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap pre([\![\mathrm{EG}\varphi]\!])$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre([\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]))$.

Fixed points approach

Axioms for CTL

- $\mathrm{EG}\varphi \equiv \varphi \wedge \mathrm{EXEG}\varphi$,
- $\mathrm{E}(\varphi \mathrm{U} \psi) \equiv \psi \vee (\varphi \wedge \mathrm{EX}(\mathrm{E}(\varphi \mathrm{U} \psi)))$,

Fixed point characterization of CTL

Let $[\![\varphi]\!] = \{s \in S \mid s \models \varphi\}$.

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap [\![\mathrm{EXEG}\varphi]\!]$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap [\![\mathrm{EXE}(\varphi \mathrm{U} \psi)]\!])$

Pre-set

Let $pre(X) = \{s \in S \mid (\exists s' \in X)\ s \rightarrow s'\}$, for $X \subseteq S$.

Characterization

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap pre([\![\mathrm{EG}\varphi]\!])$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre([\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]))$.

Fixed points approach

Axioms for CTL

- $\mathrm{EG}\varphi \equiv \varphi \wedge \mathrm{EXEG}\varphi$,
- $\mathrm{E}(\varphi \mathrm{U} \psi) \equiv \psi \vee (\varphi \wedge \mathrm{EX}(\mathrm{E}(\varphi \mathrm{U} \psi)))$,

Fixed point characterization of CTL

Let $[\![\varphi]\!] = \{s \in S \mid s \models \varphi\}$.

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap [\![\mathrm{EXEG}\varphi]\!]$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap [\![\mathrm{EXE}(\varphi \mathrm{U} \psi)]\!])$

Pre-set

Let $pre(X) = \{s \in S \mid (\exists s' \in X)\ s \to s'\}$, for $X \subseteq S$.

Characterization

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap pre([\![\mathrm{EG}\varphi]\!])$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre([\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]))$.

Fixed points approach

Axioms for CTL

- $\mathrm{EG}\varphi \equiv \varphi \wedge \mathrm{EXEG}\varphi$,
- $\mathrm{E}(\varphi\mathrm{U}\psi) \equiv \psi \vee (\varphi \wedge \mathrm{EX}(\mathrm{E}(\varphi\mathrm{U}\psi)))$,

Fixed point characterization of CTL

Let $[\![\varphi]\!] = \{s \in S \mid s \models \varphi\}$.

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap [\![\mathrm{EXEG}\varphi]\!]$,
- $[\![\mathrm{E}(\varphi\mathrm{U}\psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap [\![\mathrm{EXE}(\varphi\mathrm{U}\psi)]\!])$

Pre-set

Let $pre(X) = \{s \in S \mid (\exists s' \in X)\ s \to s'\}$, for $X \subseteq S$.

Characterization

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap pre([\![\mathrm{EG}\varphi]\!])$,
- $[\![\mathrm{E}(\varphi\mathrm{U}\psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre([\![\mathrm{E}(\varphi\mathrm{U}\psi)]\!]))$.

Fixed points approach

Axioms for CTL

- $\mathrm{EG}\varphi \equiv \varphi \wedge \mathrm{EXEG}\varphi$,
- $\mathrm{E}(\varphi \mathrm{U} \psi) \equiv \psi \vee (\varphi \wedge \mathrm{EX}(\mathrm{E}(\varphi \mathrm{U} \psi)))$,

Fixed point characterization of CTL

Let $[\![\varphi]\!] = \{s \in S \mid s \models \varphi\}$.

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap [\![\mathrm{EXEG}\varphi]\!]$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap [\![\mathrm{EXE}(\varphi \mathrm{U} \psi)]\!])$

Pre-set

Let $pre(X) = \{s \in S \mid (\exists s' \in X)\ s \to s'\}$, for $X \subseteq S$.

Characterization

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap pre([\![\mathrm{EG}\varphi]\!])$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre([\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]))$.

Fixed points approach

Axioms for CTL

- $\mathrm{EG}\varphi \equiv \varphi \wedge \mathrm{EXEG}\varphi$,
- $\mathrm{E}(\varphi \mathrm{U} \psi) \equiv \psi \vee (\varphi \wedge \mathrm{EX}(\mathrm{E}(\varphi \mathrm{U} \psi)))$,

Fixed point characterization of CTL

Let $[\![\varphi]\!] = \{s \in S \mid s \models \varphi\}$.

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap [\![\mathrm{EXEG}\varphi]\!]$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap [\![\mathrm{EXE}(\varphi \mathrm{U} \psi)]\!])$

Pre-set

Let $pre(X) = \{s \in S \mid (\exists s' \in X)\ s \rightarrow s'\}$, for $X \subseteq S$.

Characterization

- $[\![\mathrm{EG}\varphi]\!] = [\![\varphi]\!] \cap pre([\![\mathrm{EG}\varphi]\!])$,
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!] = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre([\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]))$.

Fixed points

Functions

Define two functions on 2^S, which fixed points are equal to respectively $[\![\mathrm{EG}\varphi]\!]$ and $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$.

1. $\tau_{\mathrm{EG}\varphi}(X) = [\![\varphi]\!] \cap pre(X)$,
2. $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X) = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre(X))$

Computing fixed points

- $[\![\mathrm{EG}\varphi]\!]$ is the greatest fixpoint of $\tau_{\mathrm{EG}\varphi}(X)$, so it can be computed as $\tau^k_{\mathrm{EG}\varphi}(S)$ for some finite k.
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$ is the least fixpoint of $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X)$, so it can be computed as $\tau^l_{\mathrm{E}(\varphi \mathrm{U} \psi)}(\emptyset)$ for some finite l.

  The above characterization can be now used for defining a model checking algorithm mchk for the formulas of CTL.

Fixed points

Functions

Define two functions on 2^S, which fixed points are equal to respectively $[\![\mathrm{EG}\varphi]\!]$ and $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$.

1. $\tau_{\mathrm{EG}\varphi}(X) = [\![\varphi]\!] \cap pre(X)$,
2. $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X) = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre(X))$

Computing fixed points

- $[\![\mathrm{EG}\varphi]\!]$ is the greatest fixpoint of $\tau_{\mathrm{EG}\varphi}(X)$, so it can be computed as $\tau^k_{\mathrm{EG}\varphi}(S)$ for some finite k.
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$ is the least fixpoint of $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X)$, so it can be computed as $\tau^l_{\mathrm{E}(\varphi \mathrm{U} \psi)}(\emptyset)$ for some finite l.

The above characterization can be now used for defining a model checking algorithm mchk for the formulas of CTL.

Fixed points

Functions

Define two functions on 2^S, which fixed points are equal to respectively $[\![\mathrm{EG}\varphi]\!]$ and $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$.

1. $\tau_{\mathrm{EG}\varphi}(X) = [\![\varphi]\!] \cap pre(X)$,
2. $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X) = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre(X))$

Computing fixed points

- $[\![\mathrm{EG}\varphi]\!]$ is the greatest fixpoint of $\tau_{\mathrm{EG}\varphi}(X)$, so it can be computed as $\tau^k_{\mathrm{EG}\varphi}(S)$ for some finite k.
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$ is the least fixpoint of $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X)$, so it can be computed as $\tau^l_{\mathrm{E}(\varphi \mathrm{U} \psi)}(\emptyset)$ for some finite l.

  The above characterization can be now used for defining a model checking algorithm mchk for the formulas of CTL.

Fixed points

Functions

Define two functions on 2^S, which fixed points are equal to respectively $[\![\mathrm{EG}\varphi]\!]$ and $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$.

1. $\tau_{\mathrm{EG}\varphi}(X) = [\![\varphi]\!] \cap pre(X)$,
2. $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X) = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre(X))$

Computing fixed points

- $[\![\mathrm{EG}\varphi]\!]$ is the greatest fixpoint of $\tau_{\mathrm{EG}\varphi}(X)$, so it can be computed as $\tau^k_{\mathrm{EG}\varphi}(S)$ for some finite k.
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$ is the least fixpoint of $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X)$, so it can be computed as $\tau^l_{\mathrm{E}(\varphi \mathrm{U} \psi)}(\emptyset)$ for some finite l.

  The above characterization can be now used for defining a model checking algorithm mchk for the formulas of CTL.

Fixed points

Functions

Define two functions on 2^S, which fixed points are equal to respectively $[\![\mathrm{EG}\varphi]\!]$ and $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$.

1. $\tau_{\mathrm{EG}\varphi}(X) = [\![\varphi]\!] \cap pre(X)$,
2. $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X) = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre(X))$

Computing fixed points

- $[\![\mathrm{EG}\varphi]\!]$ is the greatest fixpoint of $\tau_{\mathrm{EG}\varphi}(X)$, so it can be computed as $\tau^k_{\mathrm{EG}\varphi}(S)$ for some finite k.
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$ is the least fixpoint of $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X)$, so it can be computed as $\tau^l_{\mathrm{E}(\varphi \mathrm{U} \psi)}(\emptyset)$ for some finite l.

The above characterization can be now used for defining a model checking algorithm mchk for the formulas of CTL.

Fixed points

Functions

Define two functions on 2^S, which fixed points are equal to respectively $[\![\mathrm{EG}\varphi]\!]$ and $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$.

1. $\tau_{\mathrm{EG}\varphi}(X) = [\![\varphi]\!] \cap pre(X)$,
2. $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X) = [\![\psi]\!] \cup ([\![\varphi]\!] \cap pre(X))$

Computing fixed points

- $[\![\mathrm{EG}\varphi]\!]$ is the greatest fixpoint of $\tau_{\mathrm{EG}\varphi}(X)$, so it can be computed as $\tau^k_{\mathrm{EG}\varphi}(S)$ for some finite k.
- $[\![\mathrm{E}(\varphi \mathrm{U} \psi)]\!]$ is the least fixpoint of $\tau_{\mathrm{E}(\varphi \mathrm{U} \psi)}(X)$, so it can be computed as $\tau^l_{\mathrm{E}(\varphi \mathrm{U} \psi)}(\emptyset)$ for some finite l.

  The above characterization can be now used for defining a model checking algorithm mchk for the formulas of CTL.

Algorithm

Model checking algorithm based of fixpoint characterization

$mchk(M, \varphi)$ {

if $\varphi \in PV$, then return $V^{-1}(\varphi)$,

if $\varphi = \neg\psi$, then return $S \setminus mchk(M, \psi)$,

if $\varphi = \varphi_1 \vee \varphi_2$, then return $mchk(M, \varphi_1) \cup mchk(M, \varphi_2)$,

if $\varphi = \mathrm{EX}\psi$, then return $mchk_{\mathrm{EX}}(M, \psi)$,

if $\varphi = \mathrm{EG}\psi$, then return $mchk_{\mathrm{EG}}(M, \psi)$,

if $\varphi = \mathrm{E}(\psi_1 \mathrm{U} \psi_2)$, then return $mchk_{\mathrm{EU}}(M, \psi_1, \psi_2)$,

}

Algorithm: Model checking procedures

$mchk_{\mathrm{EX}}(M, \psi)\{$
$X := mchk(M, \psi);$
$Y := pre(X);$
return Y };

$mchk_{\mathrm{EG}}(M, \psi)\{$
$X := mchk(M, \psi);$
$Y := S;$
$Z := \emptyset;$
while $(Z \neq Y)\{$
$Z := Y;$
$Y := X \cap pre(Y)\}$
return Y };

$mchk_{\mathrm{EU}}(M, \psi_1, \psi_2)\{$
$X := mchk(M, \psi_1);$
$Y := mchk(M, \psi_2);$
$Z := \emptyset;$
$W := S;$
while $(Z \neq W)\{$
$W := Z;$
$Z := Y \cup (X \cap pre(Z))\}$
return Z };

Algorithm: Model checking procedures

$mchk_{\mathrm{EX}}(M, \psi)\{$
$X := mchk(M, \psi);$
$Y := pre(X);$
return Y };

$mchk_{\mathrm{EG}}(M, \psi)\{$
$X := mchk(M, \psi);$
$Y := S;$
$Z := \emptyset;$
while $(Z \neq Y)\{$
$Z := Y;$
$Y := X \cap pre(Y)\}$
return Y };

$mchk_{\mathrm{EU}}(M, \psi_1, \psi_2)\{$
$X := mchk(M, \psi_1);$
$Y := mchk(M, \psi_2);$
$Z := \emptyset;$
$W := S;$
while $(Z \neq W)\{$
$W := Z;$
$Z := Y \cup (X \cap pre(Z))\}$
return Z };

Introduction to OBDDs

OBDD

OBDD (Ordered Binary Dicision Diagrams) are used for succint represenation of Boolean functions.
Consider a Boolean function:

$$f : \{0, 1\}^n \longrightarrow \{0, 1\}$$

A function can be represented by the results of all the valuations of some propositional formula over n propositional variables.

Example

For example the function $f(x_1, x_2) = x_1 * x_2$ is represented by the formula $p_1 \wedge p_2$.
Each Boolean function can be represented by an OBDD.

Introduction to OBDDs

OBDD

OBDD (Ordered Binary Dicision Diagrams) are used for succint represenation of Boolean functions.
Consider a Boolean function:

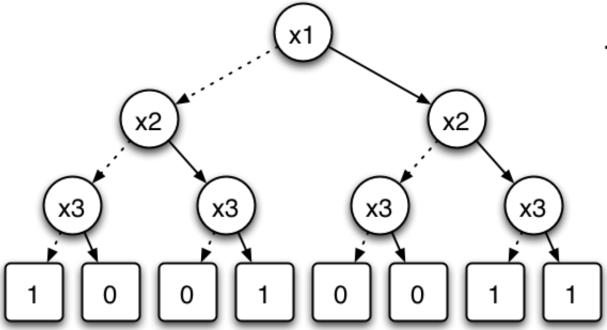
$$f : \{0, 1\}^n \longrightarrow \{0, 1\}$$

A function can be represented by the results of all the valuations of some propositional formula over n propositional variables.

Example

For example the function $f(x_1, x_2) = x_1 * x_2$ is represented by the formula $p_1 \wedge p_2$.
Each Boolean function can be represented by an OBDD.

Introduction to OBDDs



x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure: BDD representing the boolean function *f* (source: Wikipedia)

Introduction to OBDDs

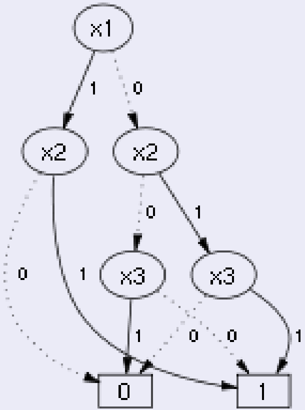


Figure: Canonical OBDD representing the boolean function f (source: Wikipedia)

Introduction to OBDDs

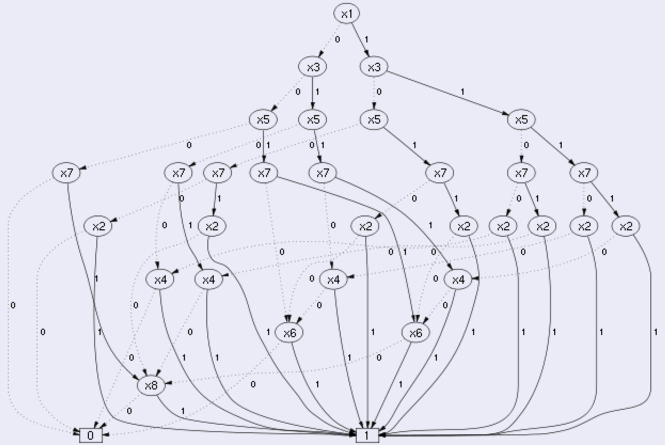


Figure: OBDD with a bad variable ordering (source: Wikipedia)

Introduction to OBDDs

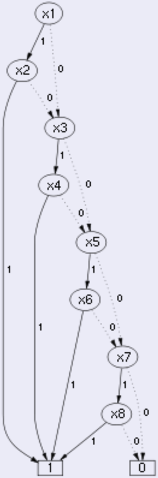


Figure: OBDD with a good variable ordering (source: Wikipedia)

Operations on OBDDs

The following operations can be implemented by polynomial-time graph manipulation algorithms:

- disjunction,
- conjunction,
- negation,
- implication,
- equivalence.

Introduction to OBDDs

Operations on OBDDs

The following operations can be implemented by polynomial-time graph manipulation algorithms:

- disjunction,
- conjunction,
- negation,
- implication,
- equivalence.

Operations on OBDDs

The following operations can be implemented by polynomial-time graph manipulation algorithms:

- disjunction,
- conjunction,
- negation,
- implication,
- equivalence.

Operations on OBDDs

The following operations can be implemented by polynomial-time graph manipulation algorithms:

- disjunction,
- conjunction,
- negation,
- implication,
- equivalence.

Introduction to OBDDs

Operations on OBDDs

The following operations can be implemented by polynomial-time graph manipulation algorithms:

- disjunction,
- conjunction,
- negation,
- implication,
- equivalence.

Operations on OBDDs

The following operations can be implemented by polynomial-time graph manipulation algorithms:

- disjunction,
- conjunction,
- negation,
- implication,
- equivalence.

OBBD-based model checking for CTLK

Fixed point algorithms on OBDD

The algorithms computing for each formula φ the set of states $[\![\varphi]\!]$ in which φ holds, can operate on the OBDD representations of the states.

Encoding

This requires to encode the states and the transition relation of a model M by propositional formulas, and then to represent these formulas by OBDDs.

Model checking

$M, s^0 \models \varphi$ is translated to checking whether $s^0 \in [\![\varphi]\!]$.

$$OBDD(\{s^0\}) \wedge OBDD([\![\varphi]\!]) \stackrel{?}{=} OBDD(\emptyset)$$

$OBDD(S)$ denotes the OBDD representing the set of states S.

OBBD-based model checking for CTLK

Fixed point algorithms on OBDD

The algorithms computing for each formula φ the set of states $[\![\varphi]\!]$ in which φ holds, can operate on the OBDD representations of the states.

Encoding

This requires to encode the states and the transition relation of a model M by propositional formulas, and then to represent these formulas by OBDDs.

Model checking

$M, s^0 \models \varphi$ is translated to checking whether $s^0 \in [\![\varphi]\!]$.

$$OBDD(\{s^0\}) \wedge OBDD([\![\varphi]\!]) \stackrel{?}{=} OBDD(\emptyset)$$

$OBDD(S)$ denotes the OBDD representing the set of states S.

OBBD-based model checking for CTLK

Fixed point algorithms on OBDD

The algorithms computing for each formula φ the set of states $[\![\varphi]\!]$ in which φ holds, can operate on the OBDD representations of the states.

Encoding

This requires to encode the states and the transition relation of a model M by propositional formulas, and then to represent these formulas by OBDDs.

Model checking

$M, s^0 \models \varphi$ is translated to checking whether $s^0 \in [\![\varphi]\!]$.

$$OBDD(\{s^0\}) \wedge OBDD([\![\varphi]\!]) \stackrel{?}{=} OBDD(\emptyset)$$

$OBDD(S)$ denotes the OBDD representing the set of states S.

SAT solvers

Complexity

- Problem: is a propositional formula satisfiable?
- Theoretical complexity: NP-complete (Cook, 1971),
- Practical and efficient SAT solvers: only in the last decade,
- Many competing algorithms: DPLL scheme is the most successful,
- A general idea: search efficiently for a satisfying assignment.

Efficiency

A SAT-solver is a heuristics only, but it can be very “clever”.
Modern SAT-solvers can decide formulas composed of hundreds of thousands of propositional variables in a reasonable time.

SAT solvers

Complexity

- Problem: is a propositional formula satisfiable?
- Theoretical complexity: NP-complete (Cook, 1971),
- Practical and efficient SAT solvers: only in the last decade,
- Many competing algorithms: DPLL scheme is the most successful,
- A general idea: search efficiently for a satisfying assignment.

Efficiency

A SAT-solver is a heuristics only, but it can be very “clever”.
Modern SAT-solvers can decide formulas composed of hundreds of thousands of propositional variables in a reasonable time.

SAT solvers

Complexity

- Problem: is a propositional formula satisfiable?
- Theoretical complexity: NP-complete (Cook, 1971),
- Practical and efficient SAT solvers: only in the last decade,
- Many competing algorithms: DPLL scheme is the most successful,
- A general idea: search efficiently for a satisfying assignment.

Efficiency

A SAT-solver is a heuristics only, but it can be very “clever”.
Modern SAT-solvers can decide formulas composed of hundreds of thousands of propositional variables in a reasonable time.

SAT solvers

Complexity

- Problem: is a propositional formula satisfiable?
- Theoretical complexity: NP-complete (Cook, 1971),
- Practical and efficient SAT solvers: only in the last decade,
- Many competing algorithms: DPLL scheme is the most successful,
- A general idea: search efficiently for a satisfying assignment.

Efficiency

A SAT-solver is a heuristics only, but it can be very "clever". Modern SAT-solvers can decide formulas composed of hundreds of thousands of propositional variables in a reasonable time.

SAT solvers

Complexity

- Problem: is a propositional formula satisfiable?
- Theoretical complexity: NP-complete (Cook, 1971),
- Practical and efficient SAT solvers: only in the last decade,
- Many competing algorithms: DPLL scheme is the most successful,
- A general idea: search efficiently for a satisfying assignment.

Efficiency

A SAT-solver is a heuristics only, but it can be very “clever”.
Modern SAT-solvers can decide formulas composed of hundreds of thousands of propositional variables in a reasonable time.

SAT solvers

Complexity

- Problem: is a propositional formula satisfiable?
- Theoretical complexity: NP-complete (Cook, 1971),
- Practical and efficient SAT solvers: only in the last decade,
- Many competing algorithms: DPLL scheme is the most successful,
- A general idea: search efficiently for a satisfying assignment.

Efficiency

A SAT-solver is a heuristics only, but it can be very “clever”.
Modern SAT-solvers can decide formulas composed of hundreds of thousands of propositional variables in a reasonable time.

SAT-solvers

Details

- Efficient data representation,
- Heuristics for deducing and learning information,
- Heuristics: frequently efficient in practice,
- CNF: conjunctive normal form, conjunction of disjunctions of literals,

$$(\neg p_1) \wedge (p_1 \vee p_4 \vee \neg p_5) \wedge (\neg p_2 \vee p_3) \wedge (p_4 \vee p_5)$$

SAT-solvers

Details

- Efficient data representation,
- Heuristics for deducing and learning information,
- Heuristics: frequently efficient in practice,
- CNF: conjunctive normal form, conjunction of disjunctions of literals,

$$(\neg p_1) \wedge (p_1 \vee p_4 \vee \neg p_5) \wedge (\neg p_2 \vee p_3) \wedge (p_4 \vee p_5)$$

SAT-solvers

Details

- Efficient data representation,
- Heuristics for deducing and learning information,
- Heuristics: frequently efficient in practice,
- CNF: conjunctive normal form, conjunction of disjunctions of literals,

$$(\neg p_1) \wedge (p_1 \vee p_4 \vee \neg p_5) \wedge (\neg p_2 \vee p_3) \wedge (p_4 \vee p_5)$$

SAT-solvers

Details

- Efficient data representation,
- Heuristics for deducing and learning information,
- Heuristics: frequently efficient in practice,
- CNF: conjunctive normal form, conjunction of disjunctions of literals,

$$(\neg p_1) \wedge (p_1 \vee p_4 \vee \neg p_5) \wedge (\neg p_2 \vee p_3) \wedge (p_4 \vee p_5)$$

Two fragments of CTL

Syntax of ACTL

The logic ACTL is the restriction of CTL such that it consists of the formulas of the form: $\mathrm{AX}\alpha$, $\mathrm{A}(\alpha \mathrm{U} \beta)$, $\mathrm{AG}\alpha$.
So, the formulas are only in the universal form (no negation applied to modalities).

Syntax of ECTL

The language of ECTL is defined as $\{\neg\varphi \mid \varphi \in \mathrm{ACTL}\}$
After 'pushing' negation down the formula, we have the formulas only in the existential form (no negation applied to modalities): $\mathrm{EX}\alpha$, $\mathrm{E}(\alpha \mathrm{U} \beta)$, $\mathrm{EG}\alpha$.

Two fragments of CTL

Syntax of ACTL

The logic ACTL is the restriction of CTL such that it consists of the formulas of the form: $\mathrm{AX}\alpha$, $\mathrm{A}(\alpha \mathrm{U} \beta)$, $\mathrm{AG}\alpha$.
So, the formulas are only in the universal form (no negation applied to modalities).

Syntax of ECTL

The language of ECTL is defined as $\{\neg\varphi \mid \varphi \in \mathrm{ACTL}\}$
After 'pushing' negation down the formula, we have the formulas only in the existential form (no negation applied to modalities): $\mathrm{EX}\alpha$, $\mathrm{E}(\alpha \mathrm{U} \beta)$, $\mathrm{EG}\alpha$.

Idea of Bounded Model Checking BMC

BMC

- BMC: to prove that an ECTL formula holds or that an ACTL formula does not hold in M
- 1.
     - If $\psi \in \mathrm{ACTL}$, take negation $\varphi := \neg\psi$
     - If $\psi \in \mathrm{ECTL}$, $\varphi := \psi$
  2. Take a fragment M' of the model M (preserving φ, i.e., $M' \models \varphi$ implies $M \models \varphi$),
  3. Translate $M' \models \varphi$ to a Boolean formula $[M'] \wedge [\varphi]_{M'}$, ($M' \models \varphi$ iff $[M'] \wedge [\varphi]_{M'}$ is satisfiable),
  4. Check the satisfiability of $[M'] \wedge [\varphi]_{M'}$ with SAT-solvers.

Conclusion:
If $[M^{'}] \wedge [\varphi]_{M'}$ is satisfiable, then $M \models \varphi$.

Idea of Bounded Model Checking BMC

BMC

- BMC: to prove that an ECTL formula holds or that an ACTL formula does not hold in M
- 1.
     - If $\psi \in \mathrm{ACTL}$, take negation $\varphi := \neg\psi$
     - If $\psi \in \mathrm{ECTL}$, $\varphi := \psi$
  2. Take a fragment M' of the model M (preserving φ, i.e., $M' \models \varphi$ implies $M \models \varphi$),
  3. Translate $M' \models \varphi$ to a Boolean formula $[M'] \wedge [\varphi]_{M'}$, ($M' \models \varphi$ iff $[M'] \wedge [\varphi]_{M'}$ is satisfiable),
  4. Check the satisfiability of $[M'] \wedge [\varphi]_{M'}$ with SAT-solvers.

Conclusion:
If $[M'] \wedge [\varphi]_{M'}$ is satisfiable, then $M \models \varphi$.

Idea of Bounded Model Checking BMC

BMC

- BMC: to prove that an ECTL formula holds or that an ACTL formula does not hold in M
- 1.
     - If $\psi \in \mathrm{ACTL}$, take negation $\varphi := \neg\psi$
     - If $\psi \in \mathrm{ECTL}$, $\varphi := \psi$
  2. Take a fragment M' of the model M (preserving φ, i.e., $M' \models \varphi$ implies $M \models \varphi$),
  3. Translate $M' \models \varphi$ to a Boolean formula $[M'] \wedge [\varphi]_{M'}$, ($M' \models \varphi$ iff $[M'] \wedge [\varphi]_{M'}$ is satisfiable),
  4. Check the satisfiability of $[M'] \wedge [\varphi]_{M'}$ with SAT-solvers.

Conclusion:

If $[M'] \wedge [\varphi]_{M'}$ is satisfiable, then $M \models \varphi$.

Idea of Bounded Model Checking BMC

BMC

- BMC: to prove that an ECTL formula holds or that an ACTL formula does not hold in M
- 1.
     - If $\psi \in \mathrm{ACTL}$, take negation $\varphi := \neg\psi$
     - If $\psi \in \mathrm{ECTL}$, $\varphi := \psi$
  2. Take a fragment M' of the model M (preserving φ, i.e., $M' \models \varphi$ implies $M \models \varphi$),
  3. Translate $M' \models \varphi$ to a Boolean formula $[M'] \wedge [\varphi]_{M'}$, ($M' \models \varphi$ iff $[M'] \wedge [\varphi]_{M'}$ is satisfiable),
  4. Check the satisfiability of $[M'] \wedge [\varphi]_{M'}$ with SAT-solvers.

Conclusion:

If $[M'] \wedge [\varphi]_{M'}$ is satisfiable, then $M \models \varphi$.

Idea of Bounded Model Checking BMC

BMC

- BMC: to prove that an ECTL formula holds or that an ACTL formula does not hold in M
- 1.
     - If $\psi \in \mathrm{ACTL}$, take negation $\varphi := \neg\psi$
     - If $\psi \in \mathrm{ECTL}$, $\varphi := \psi$
  2. Take a fragment M' of the model M (preserving φ, i.e., $M' \models \varphi$ implies $M \models \varphi$),
  3. Translate $M' \models \varphi$ to a Boolean formula $[M'] \wedge [\varphi]_{M'}$, ($M' \models \varphi$ iff $[M'] \wedge [\varphi]_{M'}$ is satisfiable),
  4. Check the satisfiability of $[M'] \wedge [\varphi]_{M'}$ with SAT-solvers.

Conclusion:
If $[M'] \wedge [\varphi]_{M'}$ is satisfiable, then $M \models \varphi$.

Idea of Bounded Model Checking BMC

BMC

- BMC: to prove that an ECTL formula holds or that an ACTL formula does not hold in M
- 1. - If $\psi \in$ ACTL, take negation $\varphi := \neg\psi$
     - If $\psi \in$ ECTL, $\varphi := \psi$
  2. Take a fragment M' of the model M (preserving φ, i.e., $M' \models \varphi$ implies $M \models \varphi$),
  3. Translate $M' \models \varphi$ to a Boolean formula $[M'] \wedge [\varphi]_{M'}$, ($M' \models \varphi$ iff $[M'] \wedge [\varphi]_{M'}$ is satisfiable),
  4. Check the satisfiability of $[M'] \wedge [\varphi]_{M'}$ with SAT-solvers.

Conclusion:

If $[M'] \wedge [\varphi]_{M'}$ is satisfiable, then $M \models \varphi$.

Idea of Bounded Model Checking BMC

BMC

- BMC: to prove that an ECTL formula holds or that an ACTL formula does not hold in M
- 1. ⋆ If $\psi \in \mathrm{ACTL}$, take negation $\varphi := \neg\psi$
     ⋆ If $\psi \in \mathrm{ECTL}$, $\varphi := \psi$
  2. Take a fragment M' of the model M (preserving φ, i.e., $M' \models \varphi$ implies $M \models \varphi$),
  3. Translate $M' \models \varphi$ to a Boolean formula $[M'] \wedge [\varphi]_{M'}$, ($M' \models \varphi$ iff $[M'] \wedge [\varphi]_{M'}$ is satisfiable),
  4. Check the satisfiability of $[M'] \wedge [\varphi]_{M'}$ with SAT-solvers.

Conclusion:
If $[M'] \wedge [\varphi]_{M'}$ is satisfiable, then $M \models \varphi$.

BMC for an ECTL formula φ

- Let φ be an ECTL formula,
- Iterate for $k := 1$ to $|M|$,
- Select the k−model M_k (of the paths of length k),
- Select the $f_k(\varphi)$-submodels of M_k (of $f_k(\varphi)$ paths),
- Translate the transition relation of the k−paths of M_k to a propositional formula $[M^{\varphi,\iota}]_k$,
- Translate φ over all the $f_k(\varphi)$-submodels to a propositional formula $[\varphi]_{M_k}$,
- Check the satisfiability of $[M,\varphi]_k := [M^{\varphi,\iota}]_k \wedge [\varphi]_{M_k}$.

BMC for an ECTL formula φ

- Let φ be an ECTL formula,
- Iterate for $k := 1$ to $|M|$,
- Select the $k-$model M_k (of the paths of length k),
- Select the $f_k(\varphi)$-submodels of M_k (of $f_k(\varphi)$ paths),
- Translate the transition relation of the $k-$paths of M_k to a propositional formula $[M^{\varphi,\iota}]_k$,
- Translate φ over all the $f_k(\varphi)$-submodels to a propositional formula $[\varphi]_{M_k}$,
- Check the satisfiability of $[M,\varphi]_k := [M^{\varphi,\iota}]_k \wedge [\varphi]_{M_k}$.

BMC for an ECTL formula φ

- Let φ be an ECTL formula,
- Iterate for $k := 1$ to $|M|$,
- Select the $k-$model M_k (of the paths of length k),
- Select the $f_k(\varphi)$-submodels of M_k (of $f_k(\varphi)$ paths),
- Translate the transition relation of the $k-$paths of M_k to a propositional formula $[M^{\varphi,\iota}]_k$,
- Translate φ over all the $f_k(\varphi)$-submodels to a propositional formula $[\varphi]_{M_k}$,
- Check the satisfiability of $[M,\varphi]_k := [M^{\varphi,\iota}]_k \wedge [\varphi]_{M_k}$.

BMC for an ECTL formula φ

- Let φ be an ECTL formula,
- Iterate for $k := 1$ to $|M|$,
- Select the $k-$model M_k (of the paths of length k),
- Select the $f_k(\varphi)$-submodels of M_k (of $f_k(\varphi)$ paths),
- Translate the transition relation of the $k-$paths of M_k to a propositional formula $[M^{\varphi,\iota}]_k$,
- Translate φ over all the $f_k(\varphi)$-submodels to a propositional formula $[\varphi]_{M_k}$,
- Check the satisfiability of $[M,\varphi]_k := [M^{\varphi,\iota}]_k \wedge [\varphi]_{M_k}$.

BMC for an ECTL formula φ

- Let φ be an ECTL formula,
- Iterate for $k := 1$ to $|M|$,
- Select the $k-$model M_k (of the paths of length k),
- Select the $f_k(\varphi)$-submodels of M_k (of $f_k(\varphi)$ paths),
- Translate the transition relation of the $k-$paths of M_k to a propositional formula $[M^{\varphi,\iota}]_k$,
- Translate φ over all the $f_k(\varphi)$-submodels to a propositional formula $[\varphi]_{M_k}$,
- Check the satisfiability of $[M,\varphi]_k := [M^{\varphi,\iota}]_k \wedge [\varphi]_{M_k}$.

BMC for an ECTL formula φ

- Let φ be an ECTL formula,
- Iterate for $k := 1$ to $|M|$,
- Select the $k-$model M_k (of the paths of length k),
- Select the $f_k(\varphi)$-submodels of M_k (of $f_k(\varphi)$ paths),
- Translate the transition relation of the $k-$paths of M_k to a propositional formula $[M^{\varphi,\iota}]_k$,
- Translate φ over all the $f_k(\varphi)$-submodels to a propositional formula $[\varphi]_{M_k}$,
- Check the satisfiability of $[M,\varphi]_k := [M^{\varphi,\iota}]_k \wedge [\varphi]_{M_k}$.

BMC for an ECTL formula φ

- Let φ be an ECTL formula,
- Iterate for $k := 1$ to $|M|$,
- Select the $k-$model M_k (of the paths of length k),
- Select the $f_k(\varphi)$-submodels of M_k (of $f_k(\varphi)$ paths),
- Translate the transition relation of the $k-$paths of M_k to a propositional formula $[M^{\varphi,\iota}]_k$,
- Translate φ over all the $f_k(\varphi)$-submodels to a propositional formula $[\varphi]_{M_k}$,
- Check the satisfiability of $[M, \varphi]_k := [M^{\varphi,\iota}]_k \wedge [\varphi]_{M_k}$.

Function f_k

Define the function $f_k : \mathrm{ECTL} \to \mathbb{N}$ as follows:

- $f_k(p) = f_k(\neg p) = 0$, where $p \in \mathcal{PV}$,
- $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$,
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$,
- $f_k(\mathrm{EX}\alpha) = f_k(\alpha) + 1$,
- $f_k(\mathrm{EG}\alpha) = (k+1) \cdot f_k(\alpha) + 1$,
- $f_k(\mathrm{E}(\alpha \mathrm{U} \beta)) = k \cdot f_k(\alpha) + f_k(\beta) + 1$.

Intuition

The function $f_k(\alpha)$ computes the number of symbolic paths (sufficient) to represent submodels of M_k in the propositional translation of α.

Function f_k

Define the function f_k :$\mathrm{ECTL} \to \mathbb{N}$ as follows:

- $f_k(p) = f_k(\neg p) = 0$, where $p \in \mathcal{PV}$,
- $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$,
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$,
- $f_k(\mathrm{EX}\alpha) = f_k(\alpha) + 1$,
- $f_k(\mathrm{EG}\alpha) = (k + 1) \cdot f_k(\alpha) + 1$,
- $f_k(\mathrm{E}(\alpha \mathrm{U} \beta)) = k \cdot f_k(\alpha) + f_k(\beta) + 1$.

Intuition

The function $f_k(\alpha)$ computes the number of symbolic paths (sufficient) to represent submodels of M_k in the propositional translation of α.

Function f_k

Define the function f_k :$\mathrm{ECTL} \to \mathbb{N}$ as follows:

- $f_k(p) = f_k(\neg p) = 0$, where $p \in \mathcal{PV}$,
- $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$,
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$,
- $f_k(\mathrm{EX}\alpha) = f_k(\alpha) + 1$,
- $f_k(\mathrm{EG}\alpha) = (k+1) \cdot f_k(\alpha) + 1$,
- $f_k(\mathrm{E}(\alpha \mathrm{U} \beta)) = k \cdot f_k(\alpha) + f_k(\beta) + 1$.

Intuition

The function $f_k(\alpha)$ computes the number of symbolic paths (sufficient) to represent submodels of M_k in the propositional translation of α.

Function f_k

Define the function f_k :$\mathrm{ECTL} \to \mathbb{N}$ as follows:

- $f_k(p) = f_k(\neg p) = 0$, where $p \in \mathcal{PV}$,
- $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$,
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$,
- $f_k(\mathrm{EX}\alpha) = f_k(\alpha) + 1$,
- $f_k(\mathrm{EG}\alpha) = (k+1) \cdot f_k(\alpha) + 1$,
- $f_k(\mathrm{E}(\alpha \mathrm{U} \beta)) = k \cdot f_k(\alpha) + f_k(\beta) + 1$.

Intuition

The function $f_k(\alpha)$ computes the number of symbolic paths (sufficient) to represent submodels of M_k in the propositional translation of α.

Function f_k

Define the function f_k :$\mathrm{ECTL} \rightarrow \mathbb{N}$ as follows:

- $f_k(p) = f_k(\neg p) = 0$, where $p \in \mathcal{PV}$,
- $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$,
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$,
- $f_k(\mathrm{EX}\alpha) = f_k(\alpha) + 1$,
- $f_k(\mathrm{EG}\alpha) = (k + 1) \cdot f_k(\alpha) + 1$,
- $f_k(\mathrm{E}(\alpha \mathrm{U} \beta)) = k \cdot f_k(\alpha) + f_k(\beta) + 1$.

Intuition

The function $f_k(\alpha)$ computes the number of symbolic paths (sufficient) to represent submodels of M_k in the propositional translation of α.

Function f_k

Define the function f_k :$\mathrm{ECTL} \to \mathbb{N}$ as follows:

- $f_k(p) = f_k(\neg p) = 0$, where $p \in \mathcal{PV}$,
- $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$,
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$,
- $f_k(\mathrm{EX}\alpha) = f_k(\alpha) + 1$,
- $f_k(\mathrm{EG}\alpha) = (k+1) \cdot f_k(\alpha) + 1$,
- $f_k(\mathrm{E}(\alpha \mathrm{U} \beta)) = k \cdot f_k(\alpha) + f_k(\beta) + 1$.

Intuition

The function $f_k(\alpha)$ computes the number of symbolic paths (sufficient) to represent submodels of M_k in the propositional translation of α.

Function f_k

Define the function f_k :$\mathrm{ECTL} \rightarrow \mathbb{N}$ as follows:

- $f_k(p) = f_k(\neg p) = 0$, where $p \in \mathcal{PV}$,
- $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$,
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$,
- $f_k(\mathrm{EX}\alpha) = f_k(\alpha) + 1$,
- $f_k(\mathrm{EG}\alpha) = (k + 1) \cdot f_k(\alpha) + 1$,
- $f_k(\mathrm{E}(\alpha \mathrm{U} \beta)) = k \cdot f_k(\alpha) + f_k(\beta) + 1$.

Intuition

The function $f_k(\alpha)$ computes the number of symbolic paths (sufficient) to represent submodels of M_k in the propositional translation of α.

Bounded semantics for ECTLpK

Let $M = (K, \mathcal{V})$ be a model and $k \in \mathbb{N}_+$.
The $k-$model for M is a structure

$$M_k = ((\mathrm{G}, P_k, \iota), \mathcal{V}),$$

where

- G - a set of the global states,
- P_k is the set of all the paths of M of length k,
- $\mathcal{V}$ - a valuation function.

Bounded semantics for ECTLpK

Let $M = (K, \mathcal{V})$ be a model and $k \in \mathbb{N}_+$.
The $k-$model for M is a structure

$$M_k = ((\mathrm{G}, P_k, \iota), \mathcal{V}),$$

where

- G - a set of the global states,
- P_k is the set of all the paths of M of length k,
- $\mathcal{V}$ - a valuation function.

Bounded semantics for ECTLpK

Let $M = (K, \mathcal{V})$ be a model and $k \in \mathbb{N}_+$.
The $k-$model for M is a structure

$$M_k = ((\mathrm{G}, P_k, \iota), \mathcal{V}),$$

where

- G - a set of the global states,
- P_k is the set of all the paths of M of length k,
- $\mathcal{V}$ - a valuation function.

Bounded semantics for **ECTLpK**

Let $M = (K, \mathcal{V})$ be a model and $k \in \mathbb{N}_+$.
The $k-$model for M is a structure

$$M_k = ((\mathrm{G}, P_k, \iota), \mathcal{V}),$$

where

- G - a set of the global states,
- P_k is the set of all the paths of M of length k,
- $\mathcal{V}$ - a valuation function.

Bounded semantics for ECTLpK

Let $M = (K, \mathcal{V})$ be a model and $k \in \mathbb{N}_+$.
The $k-$model for M is a structure

$$M_k = ((\mathrm{G}, P_k, \iota), \mathcal{V}),$$

where

- G - a set of the global states,
- P_k is the set of all the paths of M of length k,
- $\mathcal{V}$ - a valuation function.

Bounded semantics for ECTL

$$
\begin{array}{lll}
s \models \mathrm{EX}\alpha & \text{iff} & \exists \pi \in P_k \ (\pi(0) = s \text{ and } \pi(1) \models \alpha), \\
s \models \mathrm{EG}\alpha & \text{iff} & \exists \pi \in P_k (\pi(0) = s \text{ and } \\
 & & \forall_{0 \leq j \leq k} \pi(j) \models \alpha \wedge loop(\pi) \neq \emptyset), \\
s \models \mathrm{E}(\alpha \mathrm{U} \beta) & \text{iff} & (\exists \pi \in P_k) \ (\pi(0) = s \text{ and } \\
 & & \exists_{0 \leq j \leq k} (\pi(j) \models \beta \text{ and } \forall_{0 \leq i < j} \pi(i) \models \alpha)).
\end{array}
$$

Intuition

The bounded semantics for $s \models \mathrm{EG}\alpha$ says that there is a k-path π, which starts at s, all its states satisfy α and π is a loop, which means that one of the states of π is a $\rightarrow$-successor of $\pi(k)$.
$loop(\pi)$ returns the indeces of such states.

Bounded semantics for ECTL

$$
\begin{array}{lll}
s \models \mathrm{EX}\alpha & \text{iff} & \exists \pi \in P_k \; \big(\pi(0) = s \; \text{ and } \; \pi(1) \models \alpha\big), \\
s \models \mathrm{EG}\alpha & \text{iff} & \exists \pi \in P_k \big(\pi(0) = s \; \text{ and } \\
 & & \forall_{0 \le j \le k} \pi(j) \models \alpha \wedge loop(\pi) \neq \emptyset\big), \\
s \models \mathrm{E}(\alpha \mathrm{U} \beta) & \text{iff} & (\exists \pi \in P_k) \; \big(\pi(0) = s \; \text{ and } \\
 & & \exists_{0 \le j \le k} \big(\pi(j) \models \beta \; \text{ and } \; \forall_{0 \le i < j} \pi(i) \models \alpha\big)\big).
\end{array}
$$

Intuition

The bounded semantics for $s \models \mathrm{EG}\alpha$ says that there is a k-path π, which starts at s, all its states satisfy α and π is a loop, which means that one of the states of π is a $\rightarrow$-successor of $\pi(k)$.
$loop(\pi)$ returns the indeces of such states.

Main references and next lecture

References

D. Peled.
All from one, one for all: On model checking using representatives.
In *Proc. of CAV*, LNCS 697, p. 409–423, 1993.

R. Gerth, R. Kuiper, D. Peled, and W. Penczek.
A partial order approach to branching time logic model checking.
Information and Computation, 150:132–152, 1999.

W. Penczek, A. Pólrola:
Advances in Verification of Time Petri Nets and Timed Automata:
A Temporal Logic Approach.
Springer 2006.

Next lecture

Specification and model checking of Time Petri Nets and Timed Automata.

Main references and next lecture

References

D. Peled.
All from one, one for all: On model checking using representatives.
In *Proc. of CAV*, LNCS 697, p. 409–423, 1993.

R. Gerth, R. Kuiper, D. Peled, and W. Penczek.
A partial order approach to branching time logic model checking.
Information and Computation, 150:132–152, 1999.

W. Penczek, A. Pólrola:
Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach.
Springer 2006.

Next lecture

Specification and model checking of Time Petri Nets and Timed Automata.

Thank you