An Overview of Model Checking

Ma Li

Center for Logic, Language and Cognition
Philosophy Department
Peking University

November 23, 2009

Outline

The Need for Formal Methods

- informal techniques
  - simulation
  - testing
- formal techniques
  - model checking

The Need for Formal Methods

- informal techniques
  - simulation
  - testing
- formal techniques
  - model checking

The Need for Formal Methods

- informal techniques
  - simulation
  - testing
- formal techniques
  - model checking

The Need for Formal Methods

- informal techniques
  - simulation
  - testing
- formal techniques
  - model checking

The Need for Formal Methods

- informal techniques
  - simulation
  - testing
- formal techniques
  - model checking

Advantages that Model Checking Enjoys

- completely automatical
- terminate with true or a counterexample
- fast due to Partial specification
- logic used for specification has strong expressive power

Advantages that Model Checking Enjoys

- completely automatical
- terminate with true or a counterexample
- fast due to Partial specification
- logic used for specification has strong expressive power

Advantages that Model Checking Enjoys

- completely automatical
- terminate with true or a counterexample
- fast due to Partial specification
- logic used for specification has strong expressive power

Advantages that Model Checking Enjoys

- completely automatical
- terminate with true or a counterexample
- fast due to Partial specification
- logic used for specification has strong expressive power

disadvantages that Model Checking suffers from

- state space explosion

Outline

How to Do Model Checking

- the first step: convert a design into a formalism, sometimes use abstraction.
- the second step: specify the properties that the designs must satisfy.
- the third step: verify whether the model obtained in the first step satisfied the specification got in the second step.
- error trace may result from both incorrect modeling and incorrect specification.

How to Do Model Checking

- the first step: convert a design into a formalism, sometimes use abstraction.
- the second step: specify the properties that the designs must satisfy.
- the third step: verify whether the model obtained in the first step satisfied the specification got in the second step.
- error trace may result from both incorrect modeling and incorrect specification.

How to Do Model Checking

- the first step: convert a design into a formalism, sometimes use abstraction.
- the second step: specify the properties that the designs must satisfy.
- the third step: verify whether the model obtained in the first step satisfied the specification got in the second step.
- error trace may result from both incorrect modeling and incorrect specification.

How to Do Model Checking

- the first step: convert a design into a formalism, sometimes use abstraction.
- the second step: specify the properties that the designs must satisfy.
- the third step: verify whether the model obtained in the first step satisfied the specification got in the second step.
- error trace may result from both incorrect modeling and incorrect specification.

Outline

syntax

- **X**("next time") requires that a property holds in the second state of the path.
- The **F**("eventually" or "in the future") operator is used to assert that a property will hold at some state on the path.
- **G**("always" or "globally") specifies that a property holds at every state on the path.

syntax

- **X**("next time") requires that a property holds in the second state of the path.
- The **F**("eventually" or "in the future") operator is used to assert that a property will hold at some state on the path.
- **G**("always" or "globally") specifies that a property holds at every state on the path.

syntax

- **X**("next time") requires that a property holds in the second state of the path.
- The **F**("eventually" or "in the future") operator is used to assert that a property will hold at some state on the path.
- **G**("always" or "globally") specifies that a property holds at every state on the path.

syntax

- The **U** ("until") operator is a bit more complicated since it is used to combine two properties. It holds if there is a state on the path where the second property holds, and at every preceding state on the path, the first property holds.
- **R**("release") is the logical dual of the **U** operator. It requires that the second property holds along the path up to and including the first state where the first property hold. However, the first property is not required to hold eventually.

syntax

- The **U** ("until") operator is a bit more complicated since it is used to combine two properties. It holds if there is a state on the path where the second property holds, and at every preceding state on the path, the first property holds.
- **R**("release") is the logical dual of the **U** operator. It requires that the second property holds along the path up to and including the first state where the first property hold. However, the first property is not required to hold eventually.

syntax

- If $p \in AP$, then p is a state formula.
- I f and g are state formulas, then $\neg f$, $f \vee g$ and $f \wedge g$ are state formulas.
- f is a path formula, the $\mathbf{E}f$ and $\mathbf{A}f$ are state formulas.

syntax

- If $p \in AP$, then p is a state formula.
- I f and g are state formulas, then $\neg f$, $f \vee g$ and $f \wedge g$ are state formulas.
- f is a path formula, the $\mathbf{E}f$ and $\mathbf{A}f$ are state formulas.

syntax

- If $p \in AP$, then p is a state formula.
- I f and g are state formulas, then $\neg f$, $f \vee g$ and $f \wedge g$ are state formulas.
- f is a path formula, the $\mathbf{E}f$ and $\mathbf{A}f$ are state formulas.

syntax

It is easy to see that the operators $\vee$, $\neg$, **X**, **U**, and **E** are sufficient to express any other CTL^* formulas.

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$
- $f\mathbf{R}g \equiv \neg(\neg f\mathbf{U}\neg g)$
- $\mathbf{F}f \equiv \mathit{True}\ \mathbf{U}\ f$
- $\mathbf{G}f \equiv \neg\mathbf{F}\neg f$
- $\mathbf{A}(f) \equiv \neg\mathbf{E}\neg(f)$

syntax

It is easy to see that the operators $\vee$, $\neg$, **X**, **U**, and **E** are sufficient to express any other CTL^* formulas.

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$
- $f\mathbf{R}g \equiv \neg(\neg f\mathbf{U}\neg g)$
- $\mathbf{F}f \equiv True\ \mathbf{U}\ f$
- $\mathbf{G}f \equiv \neg\mathbf{F}\neg f$
- $\mathbf{A}(f) \equiv \neg\mathbf{E}\neg(f)$

syntax

It is easy to see that the operators $\vee$, $\neg$, **X**, **U**, and **E** are sufficient to express any other CTL^* formulas.

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$
- $f\mathbf{R}g \equiv \neg(\neg f\mathbf{U}\neg g)$
- $\mathbf{F}f \equiv \mathit{True}\ \mathbf{U}\ f$
- $\mathbf{G}f \equiv \neg\mathbf{F}\neg f$
- $\mathbf{A}(f) \equiv \neg\mathbf{E}\neg(f)$

syntax

It is easy to see that the operators $\vee$, $\neg$, **X**, **U**, and **E** are sufficient to express any other CTL^* formulas.

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$
- $f\mathbf{R}g \equiv \neg(\neg f\mathbf{U}\neg g)$
- $\mathbf{F}f \equiv \mathit{True}\ \mathbf{U}\ f$
- $\mathbf{G}f \equiv \neg\mathbf{F}\neg f$
- $\mathbf{A}(f) \equiv \neg\mathbf{E}\neg(f)$

syntax

It is easy to see that the operators $\vee$, $\neg$, **X**, **U**, and **E** are sufficient to express any other CTL^* formulas.

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$
- $f\mathbf{R}g \equiv \neg(\neg f\mathbf{U}\neg g)$
- $\mathbf{F}f \equiv \mathit{True}\ \mathbf{U}\ f$
- $\mathbf{G}f \equiv \neg\mathbf{F}\neg f$
- $\mathbf{A}(f) \equiv \neg\mathbf{E}\neg(f)$

syntax

Two additional rules are needed to specify the syntax of path formulas:

- If f is a state formula, then f is also a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$, and $f\mathbf{R}g$ are path formulas.

CTL^* is the set of state formulas generated by the above rules.

syntax

Two additional rules are needed to specify the syntax of path formulas:

- If f is a state formula, then f is also a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$, and $f\mathbf{R}g$ are path formulas.

CTL^* is the set of state formulas generated by the above rules.

semantics

- $M, s \models p \quad \Leftrightarrow p \in L(s)$.
- $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1$
- $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$
- $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$
- $M, s \models \mathbf{E} g_1 \quad \Leftrightarrow$ there is a path π from s such that $M, \pi \models g_1$.

semantics

- $M, s \models p \quad \Leftrightarrow p \in L(s)$.
- $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1$
- $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$
- $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$
- $M, s \models \mathbf{E} g_1 \quad \Leftrightarrow$ there is a path π from s such that $M, \pi \models g_1$.

semantics

- $M, s \models p \quad \Leftrightarrow p \in L(s)$.
- $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1$
- $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$
- $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$
- $M, s \models \mathbf{E}g_1 \quad \Leftrightarrow$ there is a path π from s such that $M, \pi \models g_1$.

semantics

- $M, s \models p \quad \Leftrightarrow p \in L(s)$.
- $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1$
- $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$
- $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$
- $M, s \models \mathbf{E} g_1 \quad \Leftrightarrow$ there is a path π from s such that $M, \pi \models g_1$.

semantics

- $M, s \models p \quad \Leftrightarrow p \in L(s)$.
- $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1$
- $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$
- $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$
- $M, s \models \mathbf{E} g_1 \quad \Leftrightarrow$ there is a path π from s such that $M, \pi \models g_1$.

semantics

- $M, s \models \mathbf{A}g_1 \quad \Leftrightarrow$ for every path π starting from s, $M, \pi \models g_1$.
- $M, \pi \models f_1 \Leftrightarrow s$ is the first state of π and $M, s \models f_1$
- $M, s \models \neg g_1 \Leftrightarrow M, s \not\models g_1$
- $M, \pi \models g_1 \vee g_2 \Leftrightarrow M, \pi \models g_1$ or $M, \pi \models g_2$
- $M, \pi \models g_1 \wedge g_2 \Leftrightarrow M, s \models g_1$ and $M, s \models g_2$

semantics

- $M, s \models \mathbf{A}g_1 \quad \Leftrightarrow$ for every path π starting from s, $M, \pi \models g_1$.
- $M, \pi \models f_1 \Leftrightarrow s$ is the first state of π and $M, s \models f_1$
- $M, s \models \neg g_1 \Leftrightarrow M, s \not\models g_1$
- $M, \pi \models g_1 \vee g_2 \Leftrightarrow M, \pi \models g_1$ or $M, \pi \models g_2$
- $M, \pi \models g_1 \wedge g_2 \Leftrightarrow M, s \models g_1$ and $M, s \models g_2$

semantics

- $M, s \models \mathbf{A}g_1 \quad \Leftrightarrow$ for every path π starting from s, $M, \pi \models g_1$.
- $M, \pi \models f_1 \Leftrightarrow s$ is the first state of π and $M, s \models f_1$
- $M, s \models \neg g_1 \Leftrightarrow M, s \not\models g_1$
- $M, \pi \models g_1 \vee g_2 \Leftrightarrow M, \pi \models g_1$ or $M, \pi \models g_2$
- $M, \pi \models g_1 \wedge g_2 \Leftrightarrow M, s \models g_1$ and $M, s \models g_2$

semantics

- $M, s \models \mathbf{A}g_1 \quad \Leftrightarrow$ for every path π starting from s, $M, \pi \models g_1$.
- $M, \pi \models f_1 \Leftrightarrow s$ is the first state of π and $M, s \models f_1$
- $M, s \models \neg g_1 \Leftrightarrow M, s \not\models g_1$
- $M, \pi \models g_1 \vee g_2 \Leftrightarrow M, \pi \models g_1$ or $M, \pi \models g_2$
- $M, \pi \models g_1 \wedge g_2 \Leftrightarrow M, s \models g_1$ and $M, s \models g_2$

semantics

- $M, s \models \mathbf{A}g_1 \quad \Leftrightarrow$ for every path π starting from s, $M, \pi \models g_1$.
- $M, \pi \models f_1 \Leftrightarrow s$ is the first state of π and $M, s \models f_1$
- $M, s \models \neg g_1 \Leftrightarrow M, s \not\models g_1$
- $M, \pi \models g_1 \vee g_2 \Leftrightarrow M, \pi \models g_1$ or $M, \pi \models g_2$
- $M, \pi \models g_1 \wedge g_2 \Leftrightarrow M, s \models g_1$ and $M, s \models g_2$

semantics

- $M, \pi \models \mathbf{X} g_1 \;\; \Leftrightarrow M, \pi^1 \models g_1$.
- $M, \pi \models \mathbf{F} g_1 \;\; \Leftrightarrow$ there exists a k $\geq$ 0 such that $M, \pi^k \models g_1$.
- $M, \pi \models \mathbf{G} g_1 \Leftrightarrow$ for all i $\geq$0, $M, \pi^i \models g_1$.
- $M, \pi \models g_1 \mathbf{U} g_2 \;\; \Leftrightarrow$ there exists a k $\geq$0 such that $M, \pi^k \models g_2$ and for all 0$\leq j < k$, $M, \pi^j \models g_1$.
- $M, \pi \models g_1 \mathbf{R} g_2 \Leftrightarrow$ for all $j \geq 0$, if for every $i < j$, M, π^i un-satisfies g_1 then $M, \pi^j \models g_2$.

semantics

- $M, \pi \models \mathbf{X} g_1 \quad \Leftrightarrow M, \pi^1 \models g_1$.
- $M, \pi \models \mathbf{F} g_1 \quad \Leftrightarrow$ there exists a k $\geq$ 0 such that $M, \pi^k \models g_1$.
- $M, \pi \models \mathbf{G} g_1 \Leftrightarrow$ for all i $\geq$0, $M, \pi^i \models g_1$.
- $M, \pi \models g_1 \mathbf{U} g_2 \quad \Leftrightarrow$ there exists a k $\geq$0 such that $M, \pi^k \models g_2$ and for all 0$\leq j < k$, $M, \pi^j \models g_1$.
- $M, \pi \models g_1 \mathbf{R} g_2 \Leftrightarrow$ for all $j \geq 0$, if for every $i < j$, M, π^i un-satisfies g_1 then $M, \pi^j \models g_2$.

semantics

- $M, \pi \models \mathbf{X} g_1 \quad \Leftrightarrow M, \pi^1 \models g_1$.
- $M, \pi \models \mathbf{F} g_1 \quad \Leftrightarrow$ there exists a k $\geq$ 0 such that $M, \pi^k \models g_1$.
- $M, \pi \models \mathbf{G} g_1 \Leftrightarrow$ for all i $\geq$0, $M, \pi^i \models g_1$.
- $M, \pi \models g_1 \mathbf{U} g_2 \quad \Leftrightarrow$ there exists a k $\geq$0 such that $M, \pi^k \models g_2$ and for all 0$\leq j < k$, $M, \pi^j \models g_1$.
- $M, \pi \models g_1 \mathbf{R} g_2 \Leftrightarrow$ for all $j \geq 0$, if for every $i < j$, M, π^i un-satisfies g_1 then $M, \pi^j \models g_2$.

semantics

- $M, \pi \models \mathbf{X}g_1 \Leftrightarrow M, \pi^1 \models g_1$.
- $M, \pi \models \mathbf{F}g_1 \Leftrightarrow$ there exists a k $\geq$ 0 such that $M, \pi^k \models g_1$.
- $M, \pi \models \mathbf{G}g_1 \Leftrightarrow$ for all i $\geq$0, $M, \pi^i \models g_1$.
- $M, \pi \models g_1\mathbf{U}g_2 \Leftrightarrow$ there exists a k $\geq$0 such that $M, \pi^k \models g_2$ and for all 0$\leq j < k$, $M, \pi^j \models g_1$.
- $M, \pi \models g_1\mathbf{R}g_2 \Leftrightarrow$ for all $j \geq 0$, if for every $i < j$, M, π^i un-satisfies g_1 then $M, \pi^j \models g_2$.

semantics

- $M, \pi \models \mathbf{X}g_1 \quad \Leftrightarrow M, \pi^1 \models g_1$.
- $M, \pi \models \mathbf{F}g_1 \quad \Leftrightarrow$ there exists a k $\geq$ 0 such that $M, \pi^k \models g_1$.
- $M, \pi \models \mathbf{G}g_1 \Leftrightarrow$ for all i $\geq$0, $M, \pi^i \models g_1$.
- $M, \pi \models g_1 \mathbf{U} g_2 \quad \Leftrightarrow$ there exists a k $\geq$0 such that $M, \pi^k \models g_2$ and for all 0$\leq j < k$, $M, \pi^j \models g_1$.
- $M, \pi \models g_1 \mathbf{R} g_2 \Leftrightarrow$ for all $j \geq 0$, if for every $i < j$, M, π^i un-satisfies g_1 then $M, \pi^j \models g_2$.

semantics

CTL is the subset of CTL^* that is obtained by restricting the syntax of path formulas using the following rule.

- If f and g are state formulas, then $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$ and $f\mathbf{R}g$ are path formulas.

semantics

An LTL path formula is either£º

- If $p \in AP$, then p is a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$, and $f\mathbf{R}g$ are path formulas.

semantics

An LTL path formula is either£º

- If $p \in AP$, then p is a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$, and $f\mathbf{R}g$ are path formulas.

semantics

Most of the specifications in the following part of this article will be written in the logic CTL. There are ten basic CTL operators:

- **AX** and **EX**,
- **AF** and **EF**
- **AG** and **EG**
- **AU** and **EU**
- **AR** and **ER**

semantics

Most of the specifications in the following part of this article will be written in the logic CTL. There are ten basic CTL operators:

- **AX** and **EX**,
- **AF** and **EF**
- **AG** and **EG**
- **AU** and **EU**
- **AR** and **ER**

semantics

Most of the specifications in the following part of this article will be written in the logic CTL. There are ten basic CTL operators:

- **AX** and **EX**,
- **AF** and **EF**
- **AG** and **EG**
- **AU** and **EU**
- **AR** and **ER**

semantics

Most of the specifications in the following part of this article will be written in the logic CTL. There are ten basic CTL operators:

- **AX** and **EX**,
- **AF** and **EF**
- **AG** and **EG**
- **AU** and **EU**
- **AR** and **ER**

semantics

Most of the specifications in the following part of this article will be written in the logic CTL. There are ten basic CTL operators:

- **AX** and **EX**,
- **AF** and **EF**
- **AG** and **EG**
- **AU** and **EU**
- **AR** and **ER**

semantics

Each of the ten operators can be expressed in terms of three operators **EX**, **EG** and **EU**:

- $\mathbf{AX}f = \neg\ \mathbf{EX}(\neg f)$
- $\mathbf{EF}f = \mathbf{E}[\mathit{True}\,\mathbf{U}\,f]$
- $\mathbf{AG}f = \neg\ \mathbf{EF}(\neg f)$
- $\mathbf{A}[f\mathbf{U}g] = \neg\ \mathbf{E}[\neg g\mathbf{U}(\neg f \wedge \neg g)] \wedge \neg \mathbf{EG}\neg g$
- $\mathbf{A}[f\mathbf{R}g] = \neg\ \mathbf{E}[\neg f\mathbf{U}\neg g]$

semantics

Each of the ten operators can be expressed in terms of three operators **EX**, **EG** and **EU**:

- **AX**$f = \neg$ **EX**$(\neg f)$
- **EF**f=**E**[$True$**U**f]
- **AG**$f = \neg$ **EF**$(\neg f)$
- **A**[f**U**g]$= \neg$ **E**[$\neg g$**U**$(\neg f \wedge \neg g)$]$\wedge\neg$**EG**$\neg g$
- **A**[f**R**g]$= \neg$ **E**[$\neg f$**U**$\neg g$]

semantics

Each of the ten operators can be expressed in terms of three operators **EX**, **EG** and **EU**:

- $\mathbf{AX}f = \neg\,\mathbf{EX}(\neg f)$
- $\mathbf{EF}f = \mathbf{E}[\mathit{True}\,\mathbf{U}\,f]$
- $\mathbf{AG}f = \neg\,\mathbf{EF}(\neg f)$
- $\mathbf{A}[f\mathbf{U}g] = \neg\,\mathbf{E}[\neg g\mathbf{U}(\neg f \wedge \neg g)] \wedge \neg\mathbf{EG}\neg g$
- $\mathbf{A}[f\mathbf{R}g] = \neg\,\mathbf{E}[\neg f\mathbf{U}\neg g]$

semantics

Each of the ten operators can be expressed in terms of three operators **EX**, **EG** and **EU**:

- $\mathbf{AX}f = \neg\,\mathbf{EX}(\neg f)$
- $\mathbf{EF}f = \mathbf{E}[\mathit{True}\,\mathbf{U}\,f]$
- $\mathbf{AG}f = \neg\,\mathbf{EF}(\neg f)$
- $\mathbf{A}[f\mathbf{U}g] = \neg\,\mathbf{E}[\neg g\mathbf{U}(\neg f \wedge \neg g)] \wedge \neg\mathbf{EG}\neg g$
- $\mathbf{A}[f\mathbf{R}g] = \neg\,\mathbf{E}[\neg f\mathbf{U}\neg g]$

semantics

Each of the ten operators can be expressed in terms of three operators **EX**, **EG** and **EU**:

- $\mathbf{AX}f = \neg\, \mathbf{EX}(\neg f)$
- $\mathbf{EF}f = \mathbf{E}[True\,\mathbf{U} f]$
- $\mathbf{AG}f = \neg\, \mathbf{EF}(\neg f)$
- $\mathbf{A}[f\mathbf{U}g] = \neg\, \mathbf{E}[\neg g\mathbf{U}(\neg f \wedge \neg g)] \wedge \neg \mathbf{EG}\neg g$
- $\mathbf{A}[f\mathbf{R}g] = \neg\, \mathbf{E}[\neg f\mathbf{U}\neg g]$

Outline

1. Introduction
   - The Need for Formal Methods
   - Advantages that Model Checking Enjoys
   - disadvantages that Model Checking suffers from
2. How to Do Model Checking
3. Temporal Logic
   - The Computation Tree Logic CTL^*
     - syntax
     - semantics
   - CTL and LTL
4. **Breakthroughs on State Space Explosion**
   - Ordered Binary Decision Diagram
   - Symbolic Model Checking

Ordered Binary Decision Diagram

Every binary decision diagram B with root v determines a boolean function $f_v(x_1, \ldots, x_n)$ in the following manner:

- If v is a terminal vertex
  - If *value(v)* =1 then $f_v(x_1, \ldots, x_n)$=1.
  - If *value(v)* =0 then $f_v(x_1, \ldots, x_n)$=0.
- v is a nonterminal vertex with *var(v)*=x_i then f_v is the function
  - $f_v(x_1, \ldots, x_n) = (\neg x_i \wedge f_{low(v)}(x_1, \ldots, x_n)) \vee (x_i \wedge f_{high(v)}(x_1, \ldots, x_n))$.

Ordered Binary Decision Diagram

Every binary decision diagram B with root v determines a boolean function $f_v(x_1, \ldots, x_n)$ in the following manner:

- If v is a terminal vertex
  - If *value(v)* =1 then $f_v(x_1, \ldots, x_n)$=1.
  - If *value(v)* =0 then $f_v(x_1, \ldots, x_n)$=0.
- v is a nonterminal vertex with $var(v)=x_i$ then f_v is the function
  - $f_v(x_1, \ldots, x_n) = (\neg x_i \wedge f_{low(v)}(x_1, \ldots, x_n)) \vee (x_i \wedge f_{high(v)}(x_1, \ldots, x_n))$

Ordered Binary Decision Diagram

Every binary decision diagram B with root v determines a boolean function $f_v(x_1, \ldots, x_n)$ in the following manner:

- If v is a terminal vertex
  - If *value(v)* =1 then $f_v(x_1, \ldots, x_n)$=1.
  - If *value(v)* =0 then $f_v(x_1, \ldots, x_n)$=0.
- v is a nonterminal vertex with *var(v)*=x_i then f_v is the function
  - $f_v(x_1, \ldots, x_n)=(\neg\, x_i \wedge f_{low(v)}(x_1, \ldots, x_n)) \vee (\, x_i \wedge f_{high(v)}(x_1, \ldots, x_n))$

Ordered Binary Decision Diagram

Every binary decision diagram B with root v determines a boolean function $f_v(x_1, \ldots, x_n)$ in the following manner:

- If v is a terminal vertex
  - If *value(v)* =1 then $f_v(x_1, \ldots, x_n)$=1.
  - If *value(v)* =0 then $f_v(x_1, \ldots, x_n)$=0.
- v is a nonterminal vertex with *var(v)*=x_i then f_v is the function
  - $f_v(x_1, \ldots, x_n)=(\neg\, x_i \wedge f_{low(v)}(x_1, \ldots, x_n)) \vee (\, x_i \wedge f_{high(v)}(x_1, \ldots, x_n))$

Ordered Binary Decision Diagram

Every binary decision diagram B with root v determines a boolean function $f_v(x_1, \ldots, x_n)$ in the following manner:

- If v is a terminal vertex
  - If *value(v)* =1 then $f_v(x_1, \ldots, x_n)$=1.
  - If *value(v)* =0 then $f_v(x_1, \ldots, x_n)$=0.
- v is a nonterminal vertex with *var(v)*=x_i then f_v is the function
  - $f_v(x_1, \ldots, x_n)$=$(\neg\ x_i \wedge f_{low(v)}(x_1, \ldots, x_n)) \vee (\ x_i \wedge f_{high(v)}(x_1, \ldots, x_n))$

Ordered Binary Decision Diagram

- canonical representation for boolean functions by restricting OBDDs.
- represent kripke structures by OBDDs

Ordered Binary Decision Diagram

- canonical representation for boolean functions by restricting OBDDs.
- represent kripke structures by OBDDs

Symbolic Model Checking

mainly focus on the symbolic model checking algorithm for CTL

- Fixpoint Representations
- **AF** $f = \mu Z.\ f_1 \vee$ **AX**Z
- **EF** $f = \mu Z.\ f_1 \vee$ **EX**Z
- **AG** $f = \nu Z.\ f_1 \wedge$ **AX**Z
- **EG** $f = \nu Z.\ f_1 \wedge$ **EX**Z
- A[f_1 **U** f_2]= $\mu Z.\ f_2 \vee (f_1 \wedge$ **AX**Z)
- E[f_1 **U** f_2]= $\mu Z.\ f_2 \vee (f_1 \wedge$ **EX**Z)
- A[f_1 **R** f_2]= $\nu Z.\ f_2 \wedge (f_1 \vee$ **AX**Z)
- A[f_1 **R** f_2]= $\nu Z.\ f_2 \wedge (f_1 \vee$ **EX**Z)

Symbolic Model Checking

mainly focus on the symbolic model checking algorithm for CTL

- Fixpoint Representations
- **AF** $f = \mu Z.\ f_1 \vee$ **AX**Z
- **EF** $f = \mu Z.\ f_1 \vee$ **EX**Z
- **AG** $f = \nu Z.\ f_1 \wedge$ **AX**Z
- **EG** $f = \nu Z.\ f_1 \wedge$ **EX**Z
- $A[f_1$ **U** $f_2] = \mu Z.\ f_2 \vee (f_1 \wedge$ **AX**$Z)$
- $E[f_1$ **U** $f_2] = \mu Z.\ f_2 \vee (f_1 \wedge$ **EX**$Z)$
- $A[f_1$ **R** $f_2] = \nu Z.\ f_2 \wedge (f_1 \vee$ **AX**$Z)$
- $A[f_1$ **R** $f_2] = \nu Z.\ f_2 \wedge (f_1 \vee$ **EX**$Z)$

Symbolic Model Checking

mainly focus on the symbolic model checking algorithm for CTL

- Fixpoint Representations
- **AF** $f = \mu Z.\ f_1 \vee$ **AX**Z
- **EF** $f = \mu Z.\ f_1 \vee$ **EX**Z
- **AG** $f = \nu Z.\ f_1 \wedge$ **AX**Z
- **EG** $f = \nu Z.\ f_1 \wedge$ **EX**Z
- **A**$[f_1$ **U** $f_2] = \mu Z.\ f_2 \vee (f_1 \wedge$ **AX**Z$)$
- **E**$[f_1$ **U** $f_2] = \mu Z.\ f_2 \vee (f_1 \wedge$ **EX**Z$)$
- **A**$[f_1$ **R** $f_2] = \nu Z.\ f_2 \wedge (f_1 \vee$ **AX**Z$)$
- **A**$[f_1$ **R** $f_2] = \nu Z.\ f_2 \wedge (f_1 \vee$ **EX**Z$)$

Symbolic Model Checking

mainly focus on the symbolic model checking algorithm for CTL

- Fixpoint Representations
- $\mathbf{AF}\ f = \mu Z.\ f_1 \vee \mathbf{AX} Z$
- $\mathbf{EF}\ f = \mu Z.\ f_1 \vee \mathbf{EX} Z$
- $\mathbf{AG}\ f = \nu Z.\ f_1 \wedge \mathbf{AX} Z$
- $\mathbf{EG}\ f = \nu Z.\ f_1 \wedge \mathbf{EX} Z$
- $\mathbf{A}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{AX} Z)$
- $\mathbf{E}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{EX} Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{AX} Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{EX} Z)$

Symbolic Model Checking

mainly focus on the symbolic model checking algorithm for CTL

- Fixpoint Representations
- $\mathbf{AF}\ f = \mu Z.\ f_1 \vee \mathbf{AX} Z$
- $\mathbf{EF}\ f = \mu Z.\ f_1 \vee \mathbf{EX} Z$
- $\mathbf{AG}\ f = \nu Z.\ f_1 \wedge \mathbf{AX} Z$
- $\mathbf{EG}\ f = \nu Z.\ f_1 \wedge \mathbf{EX} Z$
- $\mathbf{A}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{AX} Z)$
- $\mathbf{E}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{EX} Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{AX} Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{EX} Z)$

Symbolic Model Checking

mainly focus on the symbolic model checking algorithm for CTL

- Fixpoint Representations
- $\mathbf{AF}\ f = \mu Z.\ f_1 \vee \mathbf{AX}Z$
- $\mathbf{EF}\ f = \mu Z.\ f_1 \vee \mathbf{EX}Z$
- $\mathbf{AG}\ f = \nu Z.\ f_1 \wedge \mathbf{AX}Z$
- $\mathbf{EG}\ f = \nu Z.\ f_1 \wedge \mathbf{EX}Z$
- $\mathbf{A}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{AX}Z)$
- $\mathbf{E}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{EX}Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{AX}Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{EX}Z)$

Symbolic Model Checking

mainly focus on the symbolic model checking algorithm for CTL

- Fixpoint Representations
- **AF** $f = \mu Z.\ f_1 \vee$ **AX**Z
- **EF** $f = \mu Z.\ f_1 \vee$ **EX**Z
- **AG** $f = \nu Z.\ f_1 \wedge$ **AX**Z
- **EG** $f = \nu Z.\ f_1 \wedge$ **EX**Z
- **A**$[f_1$ **U**$f_2] = \mu Z.\ f_2 \vee (f_1 \wedge$ **AX**$Z)$
- **E**$[f_1$ **U**$f_2] = \mu Z.\ f_2 \vee (f_1 \wedge$ **EX**$Z)$
- **A**$[f_1$ **R**$f_2] = \nu Z.\ f_2 \wedge (f_1 \vee$ **AX**$Z)$
- **A**$[f_1$ **R**$f_2] = \nu Z.\ f_2 \wedge (f_1 \vee$ **EX**$Z)$

Symbolic Model Checking

mainly focus on the symbolic model checking algorithm for CTL

- Fixpoint Representations
- $\mathbf{AF}\ f = \mu Z.\ f_1 \vee \mathbf{AX}Z$
- $\mathbf{EF}\ f = \mu Z.\ f_1 \vee \mathbf{EX}Z$
- $\mathbf{AG}\ f = \nu Z.\ f_1 \wedge \mathbf{AX}Z$
- $\mathbf{EG}\ f = \nu Z.\ f_1 \wedge \mathbf{EX}Z$
- $\mathbf{A}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{AX}Z)$
- $\mathbf{E}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{EX}Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{AX}Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{EX}Z)$

Symbolic Model Checking

mainly focus on the symbolic model checking algorithm for CTL

- Fixpoint Representations
- $\mathbf{AF}\ f = \mu Z.\ f_1 \vee \mathbf{AX}Z$
- $\mathbf{EF}\ f = \mu Z.\ f_1 \vee \mathbf{EX}Z$
- $\mathbf{AG}\ f = \nu Z.\ f_1 \wedge \mathbf{AX}Z$
- $\mathbf{EG}\ f = \nu Z.\ f_1 \wedge \mathbf{EX}Z$
- $\mathbf{A}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{AX}Z)$
- $\mathbf{E}[f_1\ \mathbf{U} f_2\ ] = \mu Z.\ f_2 \vee (f_1 \wedge \mathbf{EX}Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{AX}Z)$
- $\mathbf{A}[f_1\ \mathbf{R} f_2\ ] = \nu Z.\ f_2 \wedge (f_1 \vee \mathbf{EX}Z)$

Symbolic Model Checking

If f is an atomic proposition a, then *Check(f)* is the OBDD representing the set of states satisfying a. If $f = f_1 \wedge f_2$ or $f = \neg f_1$, then *Check(f)* will be easily obtained according to *Check(f_1)* and *Check(f_2)*. Formulas of the form **EX** f, **E**[f **U**g], and **EG**f are handled by the procedures:

- *Check*(**EX** f) = *CheckEX(Check(f))*,
- *Check*(**E** [f **U**g] = *CheckEU(Check(f), Check(g))*,
- *Check*(**EG** f) = *CheckEG(check(f))*.

Symbolic Model Checking

If f is an atomic proposition a, then *Check(f)* is the OBDD representing the set of states satisfying a. If $f = f_1 \wedge f_2$ or $f = \neg f_1$, then *Check(f)* will be easily obtained according to *Check*(f_1) and *Check*(f_2). Formulas of the form **EX** f, **E**[f **U**g], and **EG**f are handled by the procedures:

- *Check*(**EX** f) = *CheckEX(Check(f))*,
- *Check*(**E** [f **U**g] = *CheckEU(Check(f), Check(g))*,
- *Check*(**EG** f) = *CheckEG(check(f))*.

Symbolic Model Checking

If f is an atomic proposition a, then *Check(f)* is the OBDD representing the set of states satisfying a. If $f = f_1 \wedge f_2$ or $f = \neg f_1$, then *Check(f)* will be easily obtained according to *Check(f_1)* and *Check(f_2)*. Formulas of the form **EX** f, **E**[f **U**g], and **EG**f are handled by the procedures:

- *Check*(**EX** f) = *CheckEX(Check(f))*,
- *Check*(**E** [f **U**g] = *CheckEU(Check(f), Check(g))*,
- *Check*(**EG** f) = *CheckEG(check(f))*.

Reference

- Edmund M. Clarke, Jr., Oma Grumberg, and Doron A. Peled
  Model Checking.
  MIT Press, 1999.

- E. M. Clarke, E.A. Emerson, and A.P. Sistla.
  Automatic verification of finite-state concurrent systems using temporal logic specifications.
  In *Proceedings of the 10th Annual ACM Symposium on Principles of Programming Language*,, January 1983.

Reference

- Edmund M. Clarke, Jr., Oma Grumberg, and Doron A. Peled
  Model Checking.
  MIT Press, 1999.

- E. M. Clarke, E.A. Emerson, and A.P. Sistla.
  Automatic verification of finite-state concurrent systems using temporal logic specifications.
  In *Proceedings of the 10th Annual ACM Symposium on Principles of Programming Language,*, January 1983.

Reference

- Edmund M. Clarke, Jr., Oma Grumberg, and Doron A. Peled
  Model Checking.
  MIT Press, 1999.

- E. M. Clarke, E.A. Emerson, and A.P. Sistla.
  Automatic verification of finite-state concurrent systems using temporal logic specifications.
  In *Proceedings of the 10th Annual ACM Symposium on Principles of Programming Language,*, January 1983.

Questions or Comments?