Reasoning about Real-time Systems
Temporal Logics, Modeling Checking and Timed Automata

Huang-Ming Huang

Department of Computer Science and Engineering
Washington University in St. Louis

Mu-calculus and Model Checking
The Model Checker SPIN
Timed Automata: Semantics, Algorithms and Tools
Introduction

- Theme of the talk
  - Verify the correctness of computer systems
  - Especially about long-running systems
  - Extend to real-time systems

- System Verification Methods
  - Test based
  - Proof based
  - Model Checking

- Context of Model Checking
  - Concurrent and reactive systems

- Constituents of Model Checking
  - Framework for modeling systems
  - Property specification languages
  - Verification algorithms
Introduction

- Theme of the talk
  - Verify the correctness of computer systems
  - Especially about long-running systems
  - Extend to real-time systems

- System Verification Methods
  - Test based
  - Proof based
  - Model Checking

- Context of Model Checking
  - Concurrent and reactive systems

- Constituents of Model Checking
  - Framework for modeling systems
  - Property specification languages
  - Verification algorithms
Introduction

- Theme of the talk
  - Verify the correctness of computer systems
  - Especially about long-running systems
  - Extend to real-time systems

- System Verification Methods
  - Test based
  - Proof based
  - Model Checking

- Context of Model Checking
  - Concurrent and reactive systems

- Constituents of Model Checking
  - Framework for modeling systems
  - Property specification languages
  - Verification algorithms
Introduction

- Theme of the talk
  - Verify the correctness of computer systems
  - Especially about long-running systems
  - Extend to real-time systems

- System Verification Methods
  - Test based
  - Proof based
  - Model Checking

- Context of Model Checking
  - Concurrent and reactive systems

- Constituents of Model Checking
  - Framework for modeling systems
  - Property specification languages
  - Verification algorithms
Papers to discuss

Em97 *Model Checking and the Mu-calculus* by E. A. Emerson.
- Syntax and semantics of temporal logics
- Taxonomy and complexity of different model checking approaches

Ho97 *The Model Checker SPIN* by G. J. Holzmann
- The verification algorithm adopted by SPIN
- Optimizations used by SPIN model checker

BY04 *Timed Automata: Semantics, Algorithms and Tools* by J. Bengtsson and W. Yi
- Model checking with time
- Algorithms and optimization used by Uppaal model checker
Papers to discuss

Em97  *Model Checking and the Mu-calculus* by E. A. Emerson.
- Syntax and semantics of temporal logics
- Taxonomy and complexity of different model checking approaches

Ho97  *The Model Checker SPIN* by G. J. Holzmann
- The verification algorithm adopted by SPIN
- Optimizations used by SPIN model checker

BY04  *Timed Automata: Semantics, Algorithms and Tools* by J. Bengtsson and W. Yi
- Model checking with time
- Algorithms and optimization used by Uppaal model checker
A state transition system is a quadruple \((S, S_0, T, L)\), where

- \(S\) : set of states
- \(S_0\) : set of initial states
- \(T\) : set of transitions such that for each \(\alpha \in T\), \(\alpha \subseteq S \times S\)
- \(L : S \rightarrow 2^{AP}\)

A Kripke structure is a state transition system where,

- It has only one transition relation \(R\)
- \(\forall s \in S(\exists s'(R(s, s'))))\)

Example Adapted from “Model Checking” by Clark, Grumberg and Peled
A state transition system is a quadruple \((S, S_0, T, L)\), where

- **\(S\)**: set of states
- **\(S_0\)**: set of initial states
- **\(T\)**: set of transitions such that for each \(\alpha \in T, \alpha \subseteq S \times S\)
- **\(L\)**: \(S \rightarrow 2^{AP}\)

A Kripke structure is a state transition system where,
- It has only one transition relation \(R\)
- \(\forall s \in S(\exists s'(R(s, s')))\)

Example Adapted from “Model Checking” by Clark, Grumberg and Peled
A state transition system is a quadruple \((S, S_0, T, L)\), where

- \(S\) : set of states
- \(S_0\) : set of initial states
- \(T\) : set of transitions such that for each \(\alpha \in T, \alpha \subseteq S \times S\)
- \(L\) : \(S \rightarrow 2^{AP}\)

A Kripke structure is a state transition system where,

- It has only one transition relation \(R\)
- \(\forall s \in S(\exists s'(R(s, s'))))\)
A state transition system is a quadruple $\langle S, S_0, T, L \rangle$, where

- $S$: set of states
- $S_0$: set of initial states
- $T$: set of transitions such that for each $\alpha \in T$, $\alpha \subseteq S \times S$
- $L: S \rightarrow 2^{AP}$

A Kripke structure is a state transition system where,

- It has only one transition relation $R$
- $\forall s \in S (\exists s'(R(s, s')))$

Example Adapted from “Model Checking” by Clark, Grumberg and Peled
A state transition system is a quadruple \((S, S_0, T, L)\), where

- \(S\) : set of states
- \(S_0\) : set of initial states
- \(T\) : set of transitions such that for each \(\alpha \in T\), \(\alpha \subseteq S \times S\)
- \(L : S \rightarrow 2^{AP}\)

A Kripke structure is a state transition system where,

- It has only one transition relation \(R\)
- \(\forall s \in S (\exists s'(R(s, s')))\)

Example Adapted from “Model Checking” by Clark, Grumberg and Peled
System Models

- **A state transition system** is a quadruple \((S, S_0, T, L)\), where
  - \(S\): set of states
  - \(S_0\): set of initial states
  - \(T\): set of transitions such that for each \(\alpha \in T, \alpha \subseteq S \times S\)
  - \(L: S \rightarrow 2^{AP}\)

- **A Kripke structure** is a state transition system where,
  - It has only one transition relation \(R\)
  - \(\forall s \in S (\exists s'(R(s, s')))\)

Example Adapted from “Model Checking” by Clark, Grumberg and Peled
A state transition system is a quadruple \((S, S_0, T, L)\), where

- \(S\) : set of states
- \(S_0\) : set of initial states
- \(T\) : set of transitions such that for each \(\alpha \in T, \alpha \subseteq S \times S\)
- \(L : S \rightarrow 2^{AP}\)

A Kripke structure is a state transition system where,

- It has only one transition relation \(R\)
- \(\forall s \in S(\exists s'(R(s, s'))))\)

Example Adapted from “Model Checking” by Clark, Grumberg and Peled
A state transition system is a quadruple \((S, S_0, T, L)\), where

- \(S\) : set of states
- \(S_0\) : set of initial states
- \(T\) : set of transitions such that for each \(\alpha \in T, \alpha \subseteq S \times S\)
- \(L : S \rightarrow 2^{AP}\)

A Kripke structure is a state transition system where,

- It has only one transition relation \(R\)
- \(\forall s \in S(\exists s'(R(s, s'))))\)

Example Adapted from “Model Checking” by Clark, Grumberg and Peled
Property Specification Languages

Temporal Logics

- (Propositional) Linear Temporal Logic (LTL, PTL, PLTL)
- Branching Time Logic
  - CTL (Computational Tree Logic)
  - CTL*
  - $\mu$-calculus.
Temporal Operators
Describe the Properties of Paths Through Transition Systems

$Xp : \text{Next}$

$Gp : \text{Always}$

$Fp : \text{Eventually}$

$pUq : \text{Until}$
Temporal Operators
Describe the Properties of Paths Through Transition Systems

\( \text{Xp} : \text{Next} \)

\( \text{Gp} : \text{Always} \)

\( \text{Fp} : \text{Eventually} \)

\( \text{pUq} : \text{Until} \)
Temporal Operators

Describe the Properties of Paths Through Transition Systems

- $Xp : \text{Next}$
- $Gp : \text{Always}$
- $Fp : \text{Eventually}$
- $pUq : \text{Until}$
Temporal Operators

Describe the Properties of Paths Through Transition Systems

- $Xp : \text{Next}$
- $Gp : \text{Always}$
- $Fp : \text{Eventually}$
- $pUq : \text{Until}$
Path Quantifiers

Specify the Properties of *All* or *Some* of the Paths Starting from a State

\[ \text{EF} \cdot p \]
\[ \text{AF} \cdot p \]
\[ \text{EG} \cdot p \]
\[ \text{AG} \cdot p \]
Syntax of Temporal Logics

- **LTL**: Propositional logic with temporal operators
  \[ \phi ::= \text{true} \mid \text{false} \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid (\text{X}\phi) \mid (\text{F}\phi) \mid (\text{G}\phi) \mid (\phi \text{ U } \phi) \]

- **CTL**: Propositional logic with temporal operators prefixed with path quantifiers
  \[ \phi ::= \text{true} \mid \text{false} \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid (\text{AX}\phi) \mid (\text{EX}\phi) \mid (\text{AF}\phi) \mid (\text{EF}\phi) \mid (\text{AG}\phi) \mid (\text{EG}\phi) \mid (\text{A}[\phi \text{ U } \phi]) \mid (\text{E}[\phi \text{ U } \phi]) \]

- **CTL***: A temporal operator can be prefixed by another temporal operator
  An example: \[ (\text{EGF}\ p) \]
  \[ \phi ::= \text{true} \mid \text{false} \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid (\text{A}[\alpha]) \mid (\text{E}[\alpha]) \mid (\text{A}[\alpha \text{ U } \alpha]) \mid (\text{E}[\alpha \text{ U } \alpha]) \]

- \[ \alpha ::= \phi \mid (\neg \alpha) \mid (\alpha \land \alpha) \mid (\alpha \text{ U } \alpha) \mid (\text{G}\alpha) \mid (\text{F}\alpha) \mid (\text{X}\alpha) \]
Syntax of Temporal Logics

- **LTL**: Propositional logic with temporal operators
  \[ \phi ::= \text{true} \mid \text{false} \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid (X\phi) \mid (F\phi) \mid (G\phi) \mid (\phi \text{ U } \phi) \]

- **CTL**: Propositional logic with temporal operators prefixed with path quantifiers
  \[ \phi ::= \text{true} \mid \text{false} \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid (AX \phi) \mid (EX \phi) \mid (AF \phi) \mid (EF \phi) \mid (AG \phi) \mid (EG \phi) \mid (A[\phi \text{ U } \phi]) \mid (E[\phi \text{ U } \phi]) \]

- **CTL***: A temporal operator can be prefixed by another temporal operator
  - An example: \( \text{EGF } p \)
  \[ \phi ::= \text{true} \mid \text{false} \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid (A[\phi \text{ U } \phi]) \mid (E[\phi \text{ U } \phi]) \]
  \[ \alpha ::= \phi \mid (\neg \alpha) \mid (\alpha \land \alpha) \mid (\alpha \text{ U } \alpha) \mid (G\alpha) \mid (F\alpha) \mid (X\alpha) \]
Syntax of Temporal Logics

- **LTL**: Propositional logic with temporal operators
  \[ \phi ::= \text{true} \mid \text{false} \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid (X\phi) \mid (F\phi) \mid (G\phi) \mid (\phi \ U \ \phi) \]

- **CTL**: Propositional logic with temporal operators prefixed with path quantifiers
  \[ \phi ::= \text{true} \mid \text{false} \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid (AX \phi) \mid (EX \phi) \mid (AF \phi) \mid (EF \phi) \mid (AG \phi) \mid (EG \phi) \mid (A[\phi U \phi]) \mid (E[\phi U \phi]) \]

- **CTL\*: A temporal operator can be prefixed by another temporal operator**
  - An example: \( \text{EGF} \ p \)
  \[ \phi ::= \text{true} \mid \text{false} \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid (A[\alpha] \mid E[\alpha]) \]
  \[ \alpha ::= \phi \mid (\neg \alpha) \mid (\alpha \land \alpha) \mid (\alpha \ U \ alpha) \mid (G\alpha) \mid (F\alpha) \mid (X\alpha) \]
Fixpoints

- Given a domain $D$, a function $\tau : D \rightarrow D$, $v \in D$ is a fixpoint of $\tau$ iff $\tau(v) = v$

- Least fixpoint $\mu Z \cdot \tau(Z)$

- Greatest fixpoint $\nu Z \cdot \tau(Z)$
Fixpoints

- Given a domain $D$, a function $\tau : D \to D$, $v \in D$ is a fixpoint of $\tau$ iff $\tau(v) = v$

- Least fixpoint $\mu Z \cdot \tau(Z)$

- Greatest fixpoint $\nu Z \cdot \tau(Z)$
Fixpoints

Given a domain $D$, a function $\tau : D \rightarrow D$, $v \in D$ is a fixpoint of $\tau$ iff $\tau(v) = v$

**Least fixpoint $\mu Z \cdot \tau(Z)$**

**Greatest fixpoint $\nu Z \cdot \tau(Z)$**
$\mu$-Calculus Syntax and Its Expressibility

- **$\mu$-Calculus Syntax**: Propositional logic with fixpoint representations.
  - $\phi = true \mid false \mid p \mid \neg \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \phi \rightarrow \phi$
  - $Z \mid \mu Z \cdot \phi \mid \nu Z \cdot \phi \mid EX \phi \mid AX \phi$

- **$\mu$-calculus and CTL**
  - $EF p \equiv \mu Z \cdot p \lor EX Z$
  - $AG p \equiv \nu Z \cdot p \land AX Z$
  - $AF p \equiv \mu Z \cdot p \lor AX Z$
  - $EG p \equiv \nu Z \cdot p \land EX Z$
  - $Ap U q \equiv \mu Z \cdot q \lor (p \land AX Z)$
  - $Ep U q \equiv \mu Z \cdot q \lor (p \land EX Z)$

\[
\tau(Z) = p \lor EX Z
\]
\(\mu\)-Calculus Syntax and Its Expressibility

- \(\mu\)-Calculus Syntax: Propositional logic with fixpoint representations.
  \[ \phi = \text{true} \mid \text{false} \mid p \mid \neg \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \phi \rightarrow \phi \]
  \[ \mid Z \mid \mu Z \cdot \phi \mid \nu Z \cdot \phi \mid \text{EX} \phi \mid \text{AX} \phi \]

- \(\mu\)-calculus and CTL

  \[ \text{EF } p \equiv \mu Z \cdot p \lor \text{EX } Z \]
  \[ \text{AG } p \equiv \nu Z \cdot p \land \text{AX } Z \]
  \[ \text{AF } p \equiv \mu Z \cdot p \lor \text{AX } Z \]
  \[ \text{EG } p \equiv \nu Z \cdot p \land \text{EX } Z \]
  \[ \text{Ap } U q \equiv \mu Z \cdot q \lor (p \land \text{AX } Z) \]
  \[ \text{Ep } U q \equiv \mu Z \cdot q \lor (p \land \text{EX } Z) \]

\[ \tau(Z) = p \lor \text{EX } Z \]

\[ \tau(\text{false}) = p \lor \text{EX } \text{false} \]
\( \mu \)-Calculus Syntax and Its Expressibility

- \( \mu \)-Calculus Syntax: Propositional logic with fixpoint representations.
  \[ \phi = true \mid false \mid p \mid \neg \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \phi \rightarrow \phi \mid Z \mid \mu Z \cdot \phi \mid \nu Z \cdot \phi \mid \text{EX} \phi \mid \text{AX} \phi \]

- \( \mu \)-calculus and CTL

  \[ \begin{align*}
  \text{EF} \ p & \equiv \mu Z \cdot p \lor \text{EX} Z \\
  \text{AG} \ p & \equiv \nu Z \cdot p \land \text{AX} Z \\
  \text{AF} \ p & \equiv \mu Z \cdot p \lor \text{AX} Z \\
  \text{EG} \ p & \equiv \nu Z \cdot p \land \text{EX} Z \\
  A \ p \ U \ q & \equiv \mu Z \cdot q \lor (p \land \text{AX} Z) \\
  E \ p \ U \ q & \equiv \mu Z \cdot q \lor (p \land \text{EX} Z)
  \end{align*} \]

\[ \tau(Z) = p \lor \text{EX} Z \]

\[ \tau^2(false) = p \lor \text{EX} \tau(false) \]
\(\mu\)-Calculus Syntax and Its Expressibility

- \(\mu\)-Calculus Syntax: Propositional logic with fixpoint representations.
  \[\phi = \text{true} \mid \text{false} \mid p \mid \neg \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \phi \rightarrow \phi \mid Z \mid \mu Z \cdot \phi \mid \nu Z \cdot \phi \mid \text{EX} \phi \mid \text{AX} \phi\]

- \(\mu\)-calculus and CTL

  \[
  \begin{align*}
  \text{EF} p & \equiv \mu Z \cdot p \lor \text{EX} Z \\
  \text{AG} p & \equiv \nu Z \cdot p \land \text{AX} Z \\
  \text{AF} p & \equiv \mu Z \cdot p \lor \text{AX} Z \\
  \text{EG} p & \equiv \nu Z \cdot p \land \text{EX} Z \\
  \text{Ap U} q & \equiv \mu Z \cdot q \lor (p \land \text{AX} Z) \\
  \text{Ep U} q & \equiv \mu Z \cdot q \lor (p \land \text{EX} Z)
  \end{align*}
  \]

- Example graphs:

  \[
  \tau(Z) = p \lor \text{EX} Z \\
  \tau^3(\text{false}) = p \lor \text{EX} \tau^2(\text{false})
  \]
μ-Calculus Syntax and Its Expressibility

- **μ-Calculus Syntax**: Propositional logic with fixpoint representations.
  \[ \phi = \text{true} \mid \text{false} \mid p \mid \neg \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \phi \rightarrow \phi \mid Z \mid \mu Z \cdot \phi \mid \nu Z \cdot \phi \mid \text{EX} \phi \mid \text{AX} \phi \]

- **μ-calculus and CTL**

  - \( \text{EF} \ p \equiv \mu Z \cdot p \lor \text{EX} \ Z \)
  
  - \( \text{AG} \ p \equiv \nu Z \cdot p \land \text{AX} \ Z \)
  
  - \( \text{AF} \ p \equiv \mu Z \cdot p \lor \text{AX} \ Z \)
  
  - \( \text{EG} \ p \equiv \nu Z \cdot p \land \text{EX} \ Z \)
  
  - \( \text{Ap} \ U \ q \equiv \mu Z \cdot q \lor (p \land \text{AX} \ Z) \)
  
  - \( \text{Ep} \ U \ q \equiv \mu Z \cdot q \lor (p \land \text{EX} \ Z) \)

\[ \tau(Z) = p \lor \text{EX} \ Z \]

\[ \tau^4(\text{false}) = p \lor \text{EX} \ \tau^3(\text{false}) \]
μ-Calculus Syntax and Its Expressibility

- μ-Calculus Syntax: Propositional logic with fixpoint representations.
  - $\phi = true | false | p | \neg \phi | \phi \land \phi | \phi \lor \phi | \phi \rightarrow \phi$
  - $Z | \mu Z \cdot \phi | \nu Z \cdot \phi | \text{EX } \phi | \text{AX } \phi$

- μ-calculus and CTL

\[ EF \ p \equiv \mu Z \cdot p \lor \text{EX } Z \]
\[ AG \ p \equiv \nu Z \cdot p \land \text{AX } Z \]
\[ AF \ p \equiv \mu Z \cdot p \lor \text{AX } Z \]
\[ EG \ p \equiv \nu Z \cdot p \land \text{EX } Z \]
\[ A p \ U \ q \equiv \mu Z \cdot q \lor (p \land \text{AX } Z) \]
\[ E p \ U \ q \equiv \mu Z \cdot q \lor (p \land \text{EX } Z) \]

\[ \tau(Z) = p \lor \text{EX } Z \]
\[ \tau^4(false) = p \lor \text{EX } \tau^3(false) \]
Expressibility of Temporal Logics

μ-Calculus

CTL*

LTL

CTL
Expressibility of Temporal Logics

- **LTL**
  - Only able to assert properties that will hold for all runs of a transition system

- **CTL**
  - Unable to assert properties with fairness constraints

- **CTL***
  - \((LTL \cup CTL) \subset CTL***

- **\(\mu\)-calculus**
  - It is less human readable than LTL, CTL and CTL***
  - Its inductive definability is useful as bases for modeling checking algorithms
Expressibility of Temporal Logics

- **LTL**
  - Only able to assert properties that will hold for all runs of a transition system

- **CTL**
  - Unable to assert properties with fairness constraints

- **CTL**
  - \((LTL \cup CTL) \subseteq CTL^*\)

- **\(\mu\)-calculus**
  - It is less human readable than LTL, CTL and CTL*
  - Its inductive definability is useful as bases for modeling checking algorithms
Expressibility of Temporal Logics

- **LTL**
  - Only able to assert properties that will hold for all runs of a transition system

- **CTL**
  - Unable to assert properties with fairness constraints

- **CTL***
  - \((LTL \cup CTL) \subset CTL^*\)

- **\(\mu\)-calculus**
  - It is less human readable than LTL, CTL and CTL*
  - Its inductive definability is useful as bases for modeling checking algorithms
Expressibility of Temporal Logics

- **LTL**
  - Only able to assert properties that will hold for all runs of a transition system

- **CTL**
  - Unable to assert properties with fairness constraints

- **CTL**
  - $(LTL \cup CTL) \subset CTL^*$

- **$\mu$-calculus**
  - It is less human readable than LTL, CTL and CTL*
  - Its inductive definability is useful as bases for modeling checking algorithms
Taxonomy of Model Checking Algorithms

- Explicit State vs Symbolic State
- Global Calculation vs Local Search
- Monolithic Structures vs Incremental (On-the-fly) Algorithms
Taxonomy of Model Checking Algorithms

- Explicit State vs Symbolic State
- Global Calculation vs Local Search
- Monolithic Structures vs Incremental (On-the-fly) Algorithms

AG (Start ⇒ AF Heat)
Taxonomy of Model Checking Algorithms

- Explicit State vs Symbolic State
- Global Calculation vs Local Search
- Monolithic Structures vs Incremental (On-the-fly) Algorithms

\[ \text{AG} (\text{Start} \rightarrow \text{AF Heat}) \]
Taxonomy of Model Checking Algorithms

- Explicit State vs Symbolic State
- **Global Calculation** vs Local Search
- Monolithic Structures vs Incremental (On-the-fly) Algorithms
Taxonomy of Model Checking Algorithms

- Explicit State vs Symbolic State
- Global Calculation vs Local Search
- Monolithic Structures vs Incremental (On-the-fly) Algorithms

AG (\textit{Start} \rightarrow \textit{AF Heat})
Taxonomy of Model Checking Algorithms

- Explicit State vs Symbolic State
- Global Calculation vs Local Search
- Monolithic Structures vs Incremental (On-the-fly) Algorithms
Taxonomy of Model Checking Algorithms

- Explicit State vs Symbolic State
- Global Calculation vs Local Search
- **Monolithic Structures** vs Incremental (On-the-fly) Algorithms
Taxonomy of Model Checking Algorithms

- Explicit State vs Symbolic State
- Global Calculation vs Local Search
- Monolithic Structures vs Incremental (On-the-fly) Algorithms

Incremental Algorithm
Complexities of Model Checking Algorithms

- **Explicit State**
  - CTL: $O(|M||f|)$
  - LTL, CTL*: PSPACE complete, $O(|M|e^{|f|})$
  - General $\mu$-Calculus: NP ∩ co-NP

- **Symbolic**:
  - Theoretically, PSPACE complete
  - In practice: Good performance when $M$ can be represented in a small Ordered Binary Decision Diagram (OBDD)
  - Useful for hardware circuit design and debugging
Complexities of Model Checking Algorithms

- **Explicit State**
  - CTL: \(O(|M||f|)\)
  - LTL, CTL*: PSPACE complete, \(O(|M|e|f|)\)
  - General \(\mu\)-Calculus: \(\text{NP} \cap \text{co-NP}\)

- **Symbolic**:
  - Theoretically, PSPACE complete
  - In practice: Good performance when \(M\) can be represented in a small Ordered Binary Decision Diagram (OBDD)
  - Useful for hardware circuit design and debugging
State Explosion Problem

- Given a system composed by \( n \) processes running asynchronously, each is modeled by a transition system \( M_1, M_2, \cdots, M_n \)
  - The model of the entire system is \( M = M_1 \cup M_2 \cup \cdots \cup M_n \)
  - The number of global states grows exponentially in \( n \)
- Some optimizations techniques will be introduced in the next section
Papers to discuss

Em97 *Model Checking and the Mu-calculus* by E. A. Emerson.
- Syntax and semantics of temporal logics
- Taxonomy and complexity of different model checking approaches

Ho97 *The Model Checker SPIN* by G. J. Holzmann
- The verification algorithm adopted by SPIN
- Optimizations used by SPIN model checker

BY04 *Timed Automata: Semantics, Algorithms and Tools* by J. Bengtsson and W. Yi
- Model checking with time
- Algorithms and optimization used by Uppaal model checker
SPIN Introduction

- Developed from 1991 by Gerard Holzmann
- **System Model**: PROMELA language
  - Converted to Büchi automata by the model checker
- **Property Specification**: LTL
- **Model Checking Algorithm**: Explicit state, local search and on-the-fly algorithm
A Büchi Automaton

\[ A = (\Sigma, Q, \Delta, Q_0, F) \]

is a finite automaton on infinite words

- A run is an infinite path in the graph of the automaton
  - e.g. \( s_0 s_1 s_2 s_0 s_1 s_2 s_0 \cdots \)
- A run \( \rho \) is accepted by an automaton \( A \) iff there exists at least one state in \( F \) which appears infinitely often in \( \rho \)
- Büchi automata are closed under intersection and complementation

A Büchi Automaton
Büchi Automata

- A Büchi Automaton
  $A = (\Sigma, Q, \Delta, Q_0, F)$ is a finite automaton on infinite words
- A run is an infinite path in the graph of the automaton
  - e.g. $s_0s_1s_2s_0s_1s_2s_0 \cdots$
- A run $\rho$ is accepted by an automaton $A$ iff there exists at least one state in $F$ which appears infinitely often in $\rho$
- Büchi automata are closed under intersection and complementation
Büchi Automata

- A Büchi Automaton $A = (\Sigma, Q, \Delta, Q_0, F)$ is a finite automaton on infinite words.
- A run is an infinite path in the graph of the automaton.
  - e.g. $s_1s_2s_0s_1s_2s_0 \cdots$
- A run $\rho$ is accepted by an automaton $A$ iff there exists at least one state in $F$ which appears infinitely often in $\rho$.
- Büchi automata are closed under intersection and complementation.
Büchi Automata

- A Büchi Automaton
  \( A = (\Sigma, Q, \Delta, Q_0, F) \) is a finite automaton on infinite words
- A run is an infinite path in the graph of the automaton
  - e.g. \( s_0 s_1 s_2 s_0 s_1 s_2 s_0 \ldots \)
- A run \( \rho \) is accepted by an automaton \( A \) iff there exists at least one state in \( F \) which appears infinitely often in \( \rho \)
- Büchi automata are closed under intersection and complementation

A Büchi Automaton
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton $M$
- Transform the LTL specification into another Büchi automaton $M'$
- Use nested depth-first search to find if $M \cap \overline{M'} \neq \emptyset$
  - First DFS: looking for an accepting state
  - Second DFS: looking for a cycle through the accepting state
- On-the-fly algorithm: states of $M \cap \overline{M'}$ generated as needed
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton $M$
- Transform the LTL specification into another Büchi automaton $M'$
- Use nested depth-first search to find if $M \cap M' \neq \emptyset$
  - First DFS: looking for an accepting state
  - Second DFS: looking for a cycle through the accepting state
- On-the-fly algorithm: states of $M \cap \overline{M}'$ generated as needed

Kripke Structure

```
\begin{tabular}{c}
  $s_0$
  \\
  $s_1$
  \\
  $s_2$
\end{tabular}
```

Corresponding Büchi Automaton

```
\begin{tabular}{c}
  $p$
  \\
  $q$
  \\
  $p, q$
\end{tabular}
```
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton $M$
- Transform the LTL specification into another Büchi automaton $M'$
- Use nested depth-first search to find if $M \cap \overline{M'} \neq \emptyset$
  - First DFS: looking for an accepting state
  - Second DFS: looking for a cycle through the accepting state
- On-the-fly algorithm: states of $M \cap \overline{M'}$ generated as needed
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton $M$
- Transform the LTL specification into another Büchi automaton $M'$
- Use nested depth-first search to find if $M \cap M' \neq \emptyset$
  - First DFS: looking for an accepting state
  - Second DFS: looking for a cycle through the accepting state
- On-the-fly algorithm: states of $M \cap \overline{M'}$ generated as needed
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton $M$
- Transform the LTL specification into another Büchi automaton $M'$
- Use nested depth-first search to find if $M \cap M' \neq \emptyset$
  - First DFS: looking for an accepting state
  - Second DFS: looking for a cycle through the accepting state
- On-the-fly algorithm: states of $M \cap M'$ generated as needed
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton \( M \)
- Transform the LTL specification into another Büchi automaton \( M' \)
- Use nested depth-first search to find if \( M \cap M' \neq \emptyset \)
  - First DFS: looking for an accepting state
  - Second DFS: looking for a cycle through the accepting state
- On-the-fly algorithm: states of \( M \cap M' \) generated as needed

Kripke Structure

Corresponding Büchi Automaton
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton $M$
- Transform the LTL specification into another Büchi automaton $M'$
- Use nested depth-first search to find if $M \cap \overline{M'} \neq \emptyset$
  - First DFS: looking for an accepting state
  - Second DFS: looking for a cycle through the accepting state
- On-the-fly algorithm: states of $M \cap \overline{M'}$ generated as needed

![Automaton for $FG \ p$](image-url)
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton \( M \)
- Transform the LTL specification into another Büchi automaton \( M' \)
- Use nested depth-first search to find if \( M \cap \overline{M'} \neq \emptyset \)
  - First DFS: looking for an accepting state
  - Second DFS: looking for a cycle through the accepting state
- On-the-fly algorithm: states of \( M \cap \overline{M'} \) generated as needed
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton $M$
- Transform the LTL specification into another Büchi automaton $M'$
- Use nested depth-first search to find if $M \cap \overline{M'} \neq \emptyset$
  - First DFS: looking for an accepting state
  - Second DFS: looking for a cycle through the accepting state
- On-the-fly algorithm: states of $M \cap \overline{M'}$ generated as needed
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton $M$
- Transform the LTL specification into another Büchi automaton $M'$
- Use nested depth-first search to find if $M \cap \overline{M'} \neq \emptyset$
  - **First DFS**: looking for an accepting state
  - **Second DFS**: looking for a cycle through the accepting state
- **On-the-fly algorithm**: states of $M \cap \overline{M'}$ generated as needed
LTL Model Checking

- Transform the Kripke structure into an Büchi automaton $M$
- Transform the LTL specification into another Büchi automaton $M'$
- Use nested depth-first search to find if $M \cap \overline{M'} \neq \emptyset$
  - **First DFS**: looking for an accepting state
  - **Second DFS**: looking for a cycle through the accepting state
- **On-the-fly algorithm**: states of $M \cap \overline{M'}$ generated as needed
Partial Order Reduction
Reduce the number of reachable states

T1: $x := 1$  $g := g + 2$

T2: $y := 1$  $g := g \times 2$

Huang-Ming Huang (WUSTL)
Effect of Partial Order Reduction

The graph shows the number of states generated as a function of the problem size (number of processes). The graph compares standard search and reduced search methods.

- **Number of States Generated**
  - Standard search: Dashed line
  - Reduced search: Dotted line

- **Problem Size (Number of Processes)**
  - Range: 1 to 7

- **Y-axis**
  - Ranges from $10^0$ to $10^6$ states.
State Compression
Reduce the space need for a state
Effect of State Compression
Compression of 24535220 states

<table>
<thead>
<tr>
<th>Memory (Mb)</th>
<th>Standard</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>156.59</td>
<td>59.57</td>
</tr>
<tr>
<td>Compressed</td>
<td>107.56</td>
<td>123.46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Standard</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>107.56</td>
<td>123.46</td>
</tr>
<tr>
<td>Compressed</td>
<td>107.56</td>
<td>123.46</td>
</tr>
</tbody>
</table>
Bit-State Hashing

- Used for DFS to identify if a state has been visited
- Lossy
- Need good hash function
- No false positive

Standard Hashing
Bit-State Hashing

- Used for DFS to identify if a state has been visited
- Lossy
  - Need good hash function
  - No false positive
Bit-State Hashing

- Used for DFS to identify if a state has been visited
- Lossy
- Need good hash function
- No false positive

![Graph showing problem coverage in percent vs. maximum memory available (in bits)]

**Bitstate Search**

**Standard Search**

<table>
<thead>
<tr>
<th>Coverage in percent</th>
<th>0%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Coverage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bit-State Hashing

- Used for DFS to identify if a state has been visited
- Lossy
- Need good hash function
- No false positive
Papers to discuss

Em97 *Model Checking and the Mu-calculus* by E. A. Emerson.
- Syntax and semantics of temporal logics
- Taxonomy and complexity of different model checking approaches

Ho97 *The Model Checker SPIN* by G. J. Holzmann
- The verification algorithm adopted by SPIN
- Optimizations used by SPIN model checker

BY04 *Timed Automata: Semantics, Algorithms and Tools* by J. Bengtsson and W. Yi
- Model checking with time
- Algorithms and optimization used by Uppaal model checker
Model checking with timed automata

Developed by Uppsala University and Aalborg University, first released in 1995,

**System Model** : GUI to draw timed automata

**Property Specification** : TCTL
  - TCTL : CTL with clock constraints

**Model Checking Algorithm** : Explicit state, global search
A Timed Automaton $A = (S, S_0, X, I, T)$

- $S$ is a finite set of locations
- $S_0 \subseteq S$: a set of starting locations
- $X$: a set of clocks
- $I: S \rightarrow C(X)$ location invariants
- $T \subseteq S \times C(X) \times 2^X \times S$ a set of transitions
A Timed Automaton $A = (S, S_0, X, I, T)$

- $S$ is a finite set of locations
- $S_0 \subseteq S$: a set of starting locations
- $X$: a set of clocks
- $I: S \rightarrow C(X)$ location invariants
- $T \subseteq S \times C(X) \times 2^X \times S$ a set of transitions
A Timed Automaton $A = (S, S_0, X, I, T)$

- $S$ is a finite set of locations
- $S_0 \subseteq S$: a set of starting locations
- $X$: a set of clocks
- $I : S \rightarrow C(X)$ location invariants
- $T \subseteq S \times C(X) \times 2^X \times S$ a set of transitions
A Timed Automaton $A = (S, S_0, X, I, T)$

- $S$ is a finite set of locations
- $S_0 \subseteq S$ : a set of starting locations
- $X$ : a set of clocks
- $I : S \rightarrow C(X)$ location invariants
- $T \subseteq S \times C(X) \times 2^X \times S$ a set of transitions
A Timed Automaton $A = (S, S_0, X, I, T)$
- $S$ is a finite set of locations
- $S_0 \subseteq S$: a set of starting locations
- $X$: a set of clocks
- $I: S \rightarrow C(X)$ location invariants
- $T \subseteq S \times C(X) \times 2^X \times S$ a set of transitions
Transforming A Timed Automaton into Finite Automata

```plaintext
x > 10

press?

x := 0

x ≤ 10

press?

off

press?

dim

press?

bright

⟨⟨off, x = 0⟩⟩

⟨off, x ≥ 0⟩

⟨dim, x = 0⟩

⟨dim, x ≥ 0⟩

⟨off, x > 10⟩

⟨bright, x ≤ 10⟩

⟨bright, x ≥ 0⟩
```

Huang-Ming Huang (WUSTL)
Transforming A Timed Automaton into Finite Automata

\[
\begin{align*}
&x > 10 \\
&\text{press?} \\
&x := 0 \\
&\text{dim} \\
&x \leq 10 \\
&\text{press?} \\
&\text{bright}
\end{align*}
\]

\[
\begin{align*}
&\langle \text{off}, x = 0 \rangle \\
&\langle \text{off}, x \geq 0 \rangle \\
&\langle \text{dim}, x = 0 \rangle \\
&\langle \text{dim}, x \geq 0 \rangle \\
&\langle \text{off}, x > 10 \rangle \\
&\langle \text{bright}, x \leq 10 \rangle \\
&\langle \text{bright}, x \geq 0 \rangle
\end{align*}
\]
Transforming A Timed Automaton into Finite Automata

\[ \langle \text{off}, x = 0 \rangle \]
\[ \langle \text{off}, x \geq 0 \rangle \]
\[ \langle \text{dim}, x = 0 \rangle \]
\[ \langle \text{dim}, x \geq 0 \rangle \]
\[ \langle \text{bright}, x = 0 \rangle \]
\[ \langle \text{bright}, x \leq 10 \rangle \]
\[ \langle \text{bright}, x \geq 0 \rangle \]
\[ \langle \text{off}, x > 10 \rangle \]
Transforming A Timed Automaton into Finite Automata

\[
\begin{align*}
&\langle \text{off}, x = 0 \rangle \\
&\langle \text{off}, x \geq 0 \rangle \\
&\langle \text{dim}, x = 0 \rangle \\
&\langle \text{dim}, x \geq 0 \rangle \\
&\langle \text{bright}, x = 0 \rangle \\
&\langle \text{bright}, x \geq 0 \rangle \\
&\langle \text{bright}, x \leq 10 \rangle \\
&\langle \text{off}, x > 10 \rangle
\end{align*}
\]
Transforming A Timed Automaton into Finite Automata

\[
\begin{align*}
&\langle \text{off}, x = 0 \rangle \\
&\langle \text{dim}, x = 0 \rangle \\
&\langle \text{bright}, x = 0 \rangle \\
&\langle \text{off}, x > 10 \rangle \\
&\langle \text{dim}, x \geq 0 \rangle \\
&\langle \text{bright}, x \leq 10 \rangle \\
&\langle \text{bright}, x \geq 0 \rangle
\end{align*}
\]
Transforming A Timed Automaton into Finite Automata

\[ x > 10 \quad \text{press?} \]
\[ x \leq 10 \quad \text{press?} \]
\[ x := 0 \]

\[ \langle \text{off}, x = 0 \rangle \]
\[ \langle \text{off}, x \geq 0 \rangle \]
\[ \langle \text{dim}, x = 0 \rangle \]
\[ \langle \text{dim}, x \geq 0 \rangle \]
\[ \langle \text{bright}, x = 0 \rangle \]
\[ \langle \text{bright}, x \leq 10 \rangle \]
\[ \langle \text{bright}, x \geq 0 \rangle \]

\[ \langle \langle \text{off}, x = 0 \rangle \rangle \]
\[ \langle \langle \text{off}, x \geq 0 \rangle \rangle \]

Huang-Ming Huang (WUSTL)
Transforming A Timed Automaton into Finite Automata

```
x > 10
press?  
x := 0
bright  

dim

press?

x ≤ 10
press?

off

⟨⟨off, x = 0⟩⟩

⟨⟨off, x ≥ 0⟩⟩

⟨off, x > 10⟩

⟨bright, x ≤ 10⟩

⟨bright, x ≥ 0⟩

⟨dim, x = 0⟩

⟨dim, x ≥ 0⟩

⟨bright, x = 0⟩
```

Huang-Ming Huang (WUSTL)
Transforming A Timed Automaton into Finite Automata

\[
\begin{align*}
\langle \text{off}, x = 0 \rangle & \quad \langle \text{off}, x > 0 \rangle \\
\langle \text{dim}, x = 0 \rangle & \quad \langle \text{dim}, x > 0 \rangle \\
\langle \text{bright}, x = 0 \rangle & \quad \langle \text{bright}, x > 0 \rangle \\
\langle \text{off}, x = 0 \rangle & \quad \langle \text{off}, x > 0 \rangle \\
\langle \text{dim}, x = 0 \rangle & \quad \langle \text{dim}, x > 0 \rangle \\
\langle \text{bright}, x = 0 \rangle & \quad \langle \text{bright}, x > 0 \rangle \\
\langle \text{off}, x = 0 \rangle & \quad \langle \text{off}, x > 0 \rangle \\
\langle \text{dim}, x = 0 \rangle & \quad \langle \text{dim}, x > 0 \rangle \\
\langle \text{bright}, x = 0 \rangle & \quad \langle \text{bright}, x > 0 \rangle \\
\end{align*}
\]
Difference Bound Matrices (DBM)

Representing

\[ x_1 \geq 3 \quad \land \quad x_3 \leq 5 \quad \land \]
\[ x_3 - x_1 \leq 2 \quad \land \quad x_2 - x_3 < 2 \quad \land \]
\[ x_2 - x_1 < 10 \quad \land \quad x_1 - x_2 < -4 \]

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. \( O(n^3) \)
Difference Bound Matrices (DBM)

Representing

\[
\begin{align*}
0 - x_1 & \leq -3 \quad \land \quad x_3 - 0 & \leq 5 \\
x_3 - x_1 & \leq 2 \quad \land \quad x_2 - x_3 & < 2 \\
x_2 - x_1 & < 10 \quad \land \quad x_1 - x_2 & < -4
\end{align*}
\]

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. \(O(n^3)\)
## Difference Bound Matrices (DBM)

### Representing

\[
\begin{align*}
0 - x_1 &\leq -3 & \land & x_3 - 0 &\leq 5 & \land \\
x_3 - x_1 &\leq 2 & \land & x_2 - x_3 &< 2 & \land \\
x_2 - x_1 &< 10 & \land & x_1 - x_2 &< -4
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 - 0</td>
<td>(x_1 - 0)</td>
<td>(x_2 - 0)</td>
<td>(x_3 - 0)</td>
</tr>
<tr>
<td>(x_1)</td>
<td>0 - (x_1)</td>
<td>(x_1 - x_1)</td>
<td>(x_2 - x_1)</td>
<td>(x_3 - x_1)</td>
</tr>
<tr>
<td>(x_2)</td>
<td>0 - (x_2)</td>
<td>(x_1 - x_2)</td>
<td>(x_2 - x_2)</td>
<td>(x_3 - x_2)</td>
</tr>
<tr>
<td>(x_3)</td>
<td>0 - (x_3)</td>
<td>(x_1 - x_3)</td>
<td>(x_2 - x_3)</td>
<td>(x_3 - x_3)</td>
</tr>
</tbody>
</table>

**Canonical DBM:** using all pair shortest path (Floyd-Washall) algorithm. \(O(n^3)\)
Difference Bound Matrices (DBM)

Representing

\[
0 - x_1 \leq -3 \land x_3 - 0 \leq 5 \land \\
x_3 - x_1 \leq 2 \land x_2 - x_3 < 2 \land \\
x_2 - x_1 < 10 \land x_1 - x_2 < -4
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(≤, 0)</td>
<td>(x_1 - 0)</td>
<td>(x_2 - 0)</td>
<td>(x_3 - 0)</td>
</tr>
<tr>
<td>(x_1)</td>
<td>0 - (x_1)</td>
<td>(≤, 0)</td>
<td>(x_2 - x_1)</td>
<td>(x_3 - x_1)</td>
</tr>
<tr>
<td>(x_2)</td>
<td>0 - (x_2)</td>
<td>(x_1 - x_2)</td>
<td>(≤, 0)</td>
<td>(x_3 - x_2)</td>
</tr>
<tr>
<td>(x_3)</td>
<td>0 - (x_3)</td>
<td>(x_1 - x_3)</td>
<td>(x_2 - x_3)</td>
<td>(≤, 0)</td>
</tr>
</tbody>
</table>

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. \(O(n^3)\)
Difference Bound Matrices (DBM)

Representing

\[
\begin{align*}
0 - x_1 &\leq -3 & x_3 - 0 &\leq 5 \\
x_3 - x_1 &\leq 2 & x_2 - x_3 &< 2 \\
x_2 - x_1 &< 10 & x_1 - x_2 &< -4
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>((\leq, 0))</td>
<td>(x_1 - 0)</td>
<td>(x_2 - 0)</td>
<td>(x_3 - 0)</td>
</tr>
<tr>
<td>(x_1)</td>
<td>((\leq, -3))</td>
<td>((\leq, 0))</td>
<td>(x_2 - x_1)</td>
<td>(x_3 - x_1)</td>
</tr>
<tr>
<td>(x_2)</td>
<td>0 - (x_2)</td>
<td>(x_1 - x_2)</td>
<td>((\leq, 0))</td>
<td>(x_3 - x_2)</td>
</tr>
<tr>
<td>(x_3)</td>
<td>0 - (x_3)</td>
<td>(x_1 - x_3)</td>
<td>(x_2 - x_3)</td>
<td>((\leq, 0))</td>
</tr>
</tbody>
</table>

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. \(O(n^3)\)
Difference Bound Matrices (DBM)

Representing

\[ 0 - x_1 \leq -3 \wedge x_3 - 0 \leq 5 \wedge x_3 - x_1 \leq 2 \wedge x_2 - x_3 < 2 \wedge x_2 - x_1 < 10 \wedge x_1 - x_2 < -4 \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0-x</th>
<th>0-x</th>
<th>0-x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(\leq, 0)</td>
<td>x_1-0</td>
<td>x_2-0</td>
<td>(\leq, 5)</td>
</tr>
<tr>
<td>x_1</td>
<td>(\leq, -3)</td>
<td>(\leq, 0)</td>
<td>x_2-x_1</td>
<td>x_3-x_1</td>
</tr>
<tr>
<td>x_2</td>
<td>0-x_2</td>
<td>x_1-x_2</td>
<td>(\leq, 0)</td>
<td>x_3-x_2</td>
</tr>
<tr>
<td>x_3</td>
<td>0-x_3</td>
<td>x_1-x_3</td>
<td>x_2-x_3</td>
<td>(\leq, 0)</td>
</tr>
</tbody>
</table>

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. \( O(n^3) \)
Representing

\[0 - x_1 \leq -3 \quad \land \quad x_3 - 0 \leq 5 \quad \land \]
\[x_3 - x_1 \leq 2 \quad \land \quad x_2 - x_3 < 2 \quad \land \]
\[x_2 - x_1 < 10 \quad \land \quad x_1 - x_2 < -4\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(≤, 0)</td>
<td>x_1 - 0</td>
<td>x_2 - 0</td>
<td>(≤, 5)</td>
</tr>
<tr>
<td>x_1</td>
<td>(≤, -3)</td>
<td>(≤, 0)</td>
<td>x_2 - x_1</td>
<td>(≤, 2)</td>
</tr>
<tr>
<td>x_2</td>
<td>0 - x_2</td>
<td>x_1 - x_2</td>
<td>(≤, 0)</td>
<td>x_3 - x_2</td>
</tr>
<tr>
<td>x_3</td>
<td>0 - x_3</td>
<td>x_1 - x_3</td>
<td>x_2 - x_3</td>
<td>(≤, 0)</td>
</tr>
</tbody>
</table>

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. \(O(n^3)\)
**Difference Bound Matrices (DBM)**

Representing

\[
\begin{align*}
0 - x_1 & \leq -3 \quad \land \quad x_3 - 0 & \leq 5 \\
x_3 - x_1 & \leq 2 \quad \land \quad x_2 - x_3 & < 2 \\
x_2 - x_1 & < 10 \quad \land \quad x_1 - x_2 & < -4
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$(\leq, 0)$</td>
<td>$x_1 - 0$</td>
<td>$x_2 - 0$</td>
<td>$(\leq, 5)$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>$(\leq, -3)$</td>
<td>$(\leq, 0)$</td>
<td>$x_2 - x_1$</td>
<td>$(\leq, 2)$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$0 - x_2$</td>
<td>$x_1 - x_2$</td>
<td>$(\leq, 0)$</td>
<td>$x_3 - x_2$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$0 - x_3$</td>
<td>$x_1 - x_3$</td>
<td>$(&lt;, 2)$</td>
<td>$(\leq, 0)$</td>
</tr>
</tbody>
</table>

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. $O(n^3)$
Difference Bound Matrices (DBM)

Representing

\[
\begin{align*}
0 - x_1 &\leq -3 \quad \land \quad x_3 - 0 &\leq 5 \quad \land \\
x_3 - x_1 &\leq 2 \quad \land \quad x_2 - x_3 &< 2 \quad \land \\
x_2 - x_1 &< 10 \quad \land \quad x_1 - x_2 &< -4
\end{align*}
\]

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. \(O(n^3)\)
Difference Bound Matrices (DBM)

Representing

\[
0 - x_1 \leq -3 \quad \land \quad x_3 - 0 \leq 5 \quad \land \\
x_3 - x_1 \leq 2 \quad \land \quad x_2 - x_3 < 2 \quad \land \\
x_2 - x_1 < 10 \quad \land \quad x_1 - x_2 < -4
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>((\leq, 0))</td>
<td>(x_1 - 0)</td>
<td>(x_2 - 0)</td>
<td>((\leq, 5))</td>
</tr>
<tr>
<td>(x_1)</td>
<td>((\leq, -3))</td>
<td>((\leq, 0))</td>
<td>((&lt;, 10))</td>
<td>((\leq, 2))</td>
</tr>
<tr>
<td>(x_2)</td>
<td>(0 - x_2)</td>
<td>((&lt;, -4))</td>
<td>((\leq, 0))</td>
<td>(x_3 - x_2)</td>
</tr>
<tr>
<td>(x_3)</td>
<td>(0 - x_3)</td>
<td>(x_1 - x_3)</td>
<td>((&lt;, 2))</td>
<td>((\leq, 0))</td>
</tr>
</tbody>
</table>

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. \(O(n^3)\)
Difference Bound Matrices (DBM)

Representing

\[
\begin{align*}
0 - x_1 &\leq -3 & x_3 - 0 &\leq 5 \\
x_3 - x_1 &\leq 2 & x_2 - x_3 &< 2 \\
x_2 - x_1 &< 10 & x_1 - x_2 &< -4
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>((\leq, 0))</td>
<td>((\leq, \infty))</td>
<td>((\leq, \infty))</td>
<td>((\leq, 5))</td>
</tr>
<tr>
<td>(x_1)</td>
<td>((\leq, -3))</td>
<td>((\leq, 0))</td>
<td>((&lt;, 10))</td>
<td>((\leq, 2))</td>
</tr>
<tr>
<td>(x_2)</td>
<td>((\leq, \infty))</td>
<td>((&lt;, -4))</td>
<td>((\leq, 0))</td>
<td>((\leq, \infty))</td>
</tr>
<tr>
<td>(x_3)</td>
<td>((\leq, \infty))</td>
<td>((\leq, \infty))</td>
<td>((&lt;, 2))</td>
<td>((\leq, 0))</td>
</tr>
</tbody>
</table>

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. \(O(n^3)\)
Difference Bound Matrices (DBM)

Representing

\[
\begin{align*}
0 - x_1 & \leq -3 \quad \land \quad x_3 - 0 \leq 5 \\
 x_3 - x_1 & \leq 2 \quad \land \quad x_2 - x_3 < 2 \\
x_2 - x_1 & < 10 \quad \land \quad x_1 - x_2 < -4
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$(\leq, 0)$</td>
<td>$(\leq, \infty)$</td>
<td>$(\leq, \infty)$</td>
<td>$(\leq, 5)$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>$(\leq, -3)$</td>
<td>$(\leq, 0)$</td>
<td>$(&lt;, 10)$</td>
<td>$(\leq, 2)$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$(\leq, \infty)$</td>
<td>$(&lt;, -4)$</td>
<td>$(\leq, 0)$</td>
<td>$(\leq, \infty)$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$(\leq, \infty)$</td>
<td>$(\leq, \infty)$</td>
<td>$(&lt;, 2)$</td>
<td>$(\leq, 0)$</td>
</tr>
</tbody>
</table>

Canonical DBM: using all pair shortest path (Floyd-Washall) algorithm. $O(n^3)$
DBM Operations

- **Intersection** \((D^1 \land D^2)\)
- **Clock Reset** \(D[\lambda := 0]\)
- **Elapsing of time** \(D^{\uparrow}\)

\[
D^1 = \begin{bmatrix}
\cdots & x_i - x_j & \cdots \\
\vdots & & \vdots \\
\cdots & x_i - x_j & \cdots
\end{bmatrix},
\]

\[
D^2 = \begin{bmatrix}
\cdots & \cdots \\
\vdots & & \vdots \\
\cdots & x_i - x_j & \cdots
\end{bmatrix}
\]

\[
D^1 \land D^2 = \begin{bmatrix}
\cdots & \cdots & \cdots \\
\vdots & & \vdots \\
\cdots & (\ <, 3) & \cdots \\
\vdots
\end{bmatrix}
\]
**DBM Operations**

- Intersection ($D_1 \land D_2$)
- Clock Reset ($D[\lambda := 0]$)
- Elapsing of time ($D^{\uparrow}$)

\[
D_1 = \begin{bmatrix}
\cdots & (\leq, 5) & \cdots \\
\cdots & & \cdots \\
\end{bmatrix},
\]

\[
D_2 = \begin{bmatrix}
\cdots & (\lt, 3) & \cdots \\
\cdots & & \cdots \\
\end{bmatrix}
\]

\[
D_1 \land D_2 = \begin{bmatrix}
\cdots & (\lt, 3) & \cdots \\
\cdots & & \cdots \\
\end{bmatrix}
\]
**DBM Operations**

- **Intersection** \((D^1 \land D^2)\)
- **Clock Reset** \((D[\lambda := 0])\)
- **Elapsing of time** \((D^\uparrow)\)

\[
\begin{bmatrix}
0 & x_{1,0} & x_{2,0} & x_{3,0} \\
x_{0,1} & 0 & x_{2,1} & x_{3,1} \\
x_{0,2} & x_{1,2} & 0 & x_{3,2} \\
x_{0,3} & x_{1,3} & x_{2,3} & 0
\end{bmatrix}
\]

\[
x_2 := 0
\]
**DBM Operations**

- Intersection ($D^1 \land D^2$)
- Clock Reset ($D[\lambda := 0]$)
- Elapsing of time ($D^{\uparrow}$)

\[
x_2 := 0
\]

\[
\begin{bmatrix}
0 & x_{1,0} & x_{2,0} & x_{3,0} \\
 x_{0,1} & 0 & x_{2,1} & x_{3,1} \\
 0 & x_{1,0} & 0 & x_{3,0} \\
 x_{0,3} & x_{1,3} & x_{2,3} & 0
\end{bmatrix}
\]
DBM Operations

- Intersection ($D^1 \wedge D^2$)
- Clock Reset ($D[\lambda := 0]$)
- Elapsing of time ($D^\uparrow$)

\[
x_2 := 0
\]

\[
\begin{pmatrix}
0 & x_{1,0} & 0 & x_{3,0} \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
x_{0,1} & 0 & x_{0,1} & x_{3,1} \\
0 & x_{1,0} & 0 & x_{3,0} \\
x_{0,3} & x_{1,3} & x_{0,3} & 0 \\
\end{pmatrix}
\]
DBM Operations

- Intersection \((D^1 \land D^2)\)
- Clock Reset \((D[\lambda := 0])\)
- Elapsing of time \((D^\uparrow)\)

\[
D = \begin{bmatrix}
0 & x_{1,0} & x_{2,0} & x_{3,0} \\
x_{0,1} & 0 & x_{2,1} & x_{3,1} \\
x_{0,2} & x_{1,2} & 0 & x_{3,2} \\
x_{0,3} & x_{1,3} & x_{2,3} & 0
\end{bmatrix}
\]
DBM Operations

- Intersection \((D^1 \land D^2)\)
- Clock Reset \((D[\lambda := 0])\)
- Elapsing of time \((D^{\uparrow})\)

\[
D = \begin{bmatrix}
0 & x_{1,0} & x_{2,0} & x_{3,0} \\
x_{0,1} & 0 & x_{2,1} & x_{3,1} \\
x_{0,2} & x_{1,2} & 0 & x_{3,2} \\
x_{0,3} & x_{1,3} & x_{2,3} & 0
\end{bmatrix}
\]
DBM Operations

- Intersection ($D^1 \land D^2$)
- Clock Reset ($D[\lambda := 0]$)
- Elapsing of time ($D^{\uparrow}$)

$$D^{\uparrow} = \begin{bmatrix}
0 & \infty & \infty & \infty \\
x_{0,1} & 0 & x_{2,1} & x_{3,1} \\
x_{0,2} & x_{1,2} & 0 & x_{3,2} \\
x_{0,3} & x_{1,3} & x_{2,3} & 0
\end{bmatrix}$$
Computing zone successor

- \( \varphi \) is the current zone
  - e.g. \( \varphi = \{x = 0 \land y = 0\} \)
- Given a transition \( e = (s, \psi, \lambda, s') \) of a timed automaton
- \( I(s) \) is the invariant of state \( s \)

\[
\text{succ}(s, \varphi) = ((\varphi \land I(s)) \uparrow \land I(s) \land \psi)[\lambda := 0]
\]
Computing zone successor

- \( \varphi \) is the current zone
  - e.g. \( \varphi = \{ x = 0 \land y = 0 \} \)
- Given a transition \( e = (s, \psi, \lambda, s') \) of a timed automaton
  - \( I(s) \) is the invariant of state \( s \)
  
\[
\text{succ}(s, \varphi) = (\varphi \land I(s))^\uparrow \land I(s) \land \psi)[\lambda := 0]
\]
Computing zone successor

- $\varphi$ is the current zone
  - e.g. $\varphi = \{x = 0 \land y = 0\}$
- Given a transition $e = (s, \psi, \lambda, s')$ of a timed automaton
- $I(s)$ is the invariant of state $s$

$succ(s, \varphi) = ((\varphi \land I(s))^\uparrow \land I(s) \land \psi)[\lambda := 0]$
Computing zone successor

- \( \varphi \) is the current zone
  - e.g. \( \varphi = \{x = 0 \land y = 0\} \)
- Given a transition \( e = (s, \psi, \lambda, s') \) of a timed automaton
- \( I(s) \) is the invariant of state \( s \)

\[
succ(s, \varphi) = ((\varphi \land I(s))^\uparrow \land I(s) \land \psi) [\lambda := 0]
\]
Computing zone successor

- \( \varphi \) is the current zone
  - e.g. \( \varphi = \{x = 0 \land y = 0\} \)

- Given a transition \( e = (s, \psi, \lambda, s') \) of a timed automaton

- \( I(s) \) is the invariant of state \( s \)

- \( \text{succ}(s, \varphi) = (((\varphi \land I(s)) \uparrow \land I(s) \land \psi)[\lambda := 0] \)

\[
\begin{align*}
x &\leq 4 \land y \leq 5 \\
x &\geq 3 \land y \geq 2 \\
x &:= 0
\end{align*}
\]
Computing zone successor

- $\varphi$ is the current zone
  - e.g. $\varphi = \{x = 0 \land y = 0\}$

- Given a transition $e = (s, \psi, \lambda, s')$ of a timed automaton

- $I(s)$ is the invariant of state $s$

- $\text{succ}(s, \varphi) = ((\varphi \land I(s))^\uparrow \land I(s) \land \psi)[\lambda := 0]$
Computing zone successor

- \( \varphi \) is the current zone
  - e.g. \( \varphi = \{ x = 0 \land y = 0 \} \)
- Given a transition \( e = (s, \psi, \lambda, s') \) of a timed automaton
- \( I(s) \) is the invariant of state \( s \)
- \( \text{succ}(s, \varphi) = ((\varphi \land I(s))^\uparrow \land I(s) \land \psi)[\lambda := 0] \)
Computing zone successor

- $\varphi$ is the current zone
  - e.g. $\varphi = \{x = 0 \land y = 0\}$

- Given a transition $e = (s, \psi, \lambda, s')$ of a timed automaton

- $I(s)$ is the invariant of state $s$

- $\text{succ}(s, \varphi) = ((\varphi \land I(s))^\uparrow \land I(s) \land \psi)[\lambda := 0]$
Computing zone successor

- $\varphi$ is the current zone
  - e.g. $\varphi = \{x = 0 \land y = 0\}$

- Given a transition $e = (s, \psi, \lambda, s')$ of a timed automaton

- $I(s)$ is the invariant of state $s$

- $\text{succ}(s, \varphi) = ((\varphi \land I(s))^{\uparrow} \land I(s) \land \psi)[\lambda := 0]$
Computing zone successor

- $\varphi$ is the current zone
  - e.g. $\varphi = \{x = 0 \land y = 0\}$
- Given a transition $e = (s, \psi, \lambda, s')$ of a timed automaton
- $I(s)$ is the invariant of state $s$
- $\text{succ}(s, \varphi) = ((\varphi \land I(s))^\uparrow \land I(s) \land \psi)[\lambda := 0]$

```
x \leq 4 \land y \leq 5
```

```
\text{rel} \quad s
\text{rel} \quad e
\text{rel} \quad s'
```

```
x \geq 3 \land y \geq 2
x := 0
```

```
x \leq 4 \\
y \leq 5
```

```
x \geq 3 \\
y \geq 2
x := 0
```

```
x \leq 4 \\
y \leq 5
```

```
x \geq 3 \\
y \geq 2
x := 0
```

```
x \leq 4 \\
y \leq 5
```

```
x \geq 3 \\
y \geq 2
x := 0
```

```
x \leq 4 \\
y \leq 5
```

```
x \geq 3 \\
y \geq 2
x := 0
```
Conclusion
Summary

- **System Model**
  - Kripke Structures [Em97]
  - Transition Systems
  - Timed Automata [BY04]

- **Specify Properties**
  - Temporal Logics: LTL [Em97,Ho97], CTL, CTL*, $\mu$-calculus [Em97]
  - Timed Temporal Logics: TCTL [BY04], TLTL

- **Algorithms**
  - Symbolic [Em97] vs Explicit States [Em97,Ho97]
  - Global searching [Em97] vs On-the-fly [Em97,Ho97]
  - From timed to untimed: DBM [BY04]

- **Tools**
  - Untimed: SPIN [Ho97], Bogor, ...
  - Timed: Uppaal [BY04], IF, ...
Summary

- **System Model**
  - Kripke Structures [Em97]
  - Transition Systems
  - Timed Automata [BY04]

- **Specify Properties**
  - Temporal Logics : LTL [Em97,Ho97], CTL, CTL*, $\mu$-calculus [Em97]
  - Timed Temporal Logics : TCTL [BY04], TLTL

- **Algorithms**
  - Symbolic[Em97] vs Explicit States [Em97,Ho97]
  - Global searching [Em97] vs On-the-fly [Em97,Ho97]
  - From timed to untimed : DBM [BY04]

- **Tools**
  - Untimed : SPIN [Ho97], Bogor, ...
  - Timed : Uppaal [BY04], IF, ...
Summary

- **System Model**
  - Kripke Structures [Em97]
  - Transition Systems
  - Timed Automata [BY04]

- **Specify Properties**
  - Temporal Logics: LTL [Em97, Ho97], CTL, CTL*, μ-calculus [Em97]
  - Timed Temporal Logics: TCTL [BY04], TLTL

- **Algorithms**
  - Symbolic [Em97] vs Explicit States [Em97, Ho97]
  - Global searching [Em97] vs On-the-fly [Em97, Ho97]
  - From timed to untimed: DBM [BY04]

- **Tools**
  - Untimed: SPIN [Ho97], Bogor, ...
  - Timed: Uppaal [BY04], IF, ...
Summary

- **System Model**
  - Kripke Structures [Em97]
  - Transition Systems
  - Timed Automata [BY04]

- **Specify Properties**
  - Temporal Logics: LTL [Em97,Ho97], CTL, CTL*, \(\mu\)-calculus [Em97]
  - Timed Temporal Logics: TCTL [BY04], TLTL

- **Algorithms**
  - Symbolic[Em97] vs Explicit States [Em97,Ho97]
  - Global searching [Em97] vs On-the-fly [Em97,Ho97]
  - From timed to untimed: DBM [BY04]

- **Tools**
  - Untimed: SPIN [Ho97], Bogor, ...
  - Timed: Uppaal [BY04], IF, ...
Avoid State Explosion Problem
- Algorithm Optimizations [Ho97]
- Better System Abstraction

Model Checking Applications
- Untimed [Ho97]: Leader election problems, mutual exclusion, communication networks design ...
- Timed [BY04]: TDMA Protocol Start-up mechanism, Lip synchronization algorithms ...
Avoid State Explosion Problem
- Algorithm Optimizations [Ho97]
- Better System Abstraction

Model Checking Applications
- Untimed [Ho97]: Leader election problems, mutual exclusion, communication networks design ...
- Timed [BY04]: TDMA Protocol Start-up mechanism, Lip synchronization algorithms ...