Symbolic Model Checking of Real-Time Systems using Difference Decision Diagrams

Jesper Blak Møller
[jm@it.edu]

Department of Innovation
The IT University of Copenhagen
[www.it.edu/people/jm]

Outline of Talk

1. Background
2. Symbolic model checking
3. Difference decision diagrams
4. Summary of results

Background

- More and more systems contain **embedded computers:**
  - Mobile phones
  - Medical equipment
  - Cars
  - Satellites
  - Airplanes

Background

- More and more systems contain **embedded computers:**
  - Mobile phones
  - Medical equipment
  - Cars
  - Satellites
  - Airplanes
- Nobody’s perfect

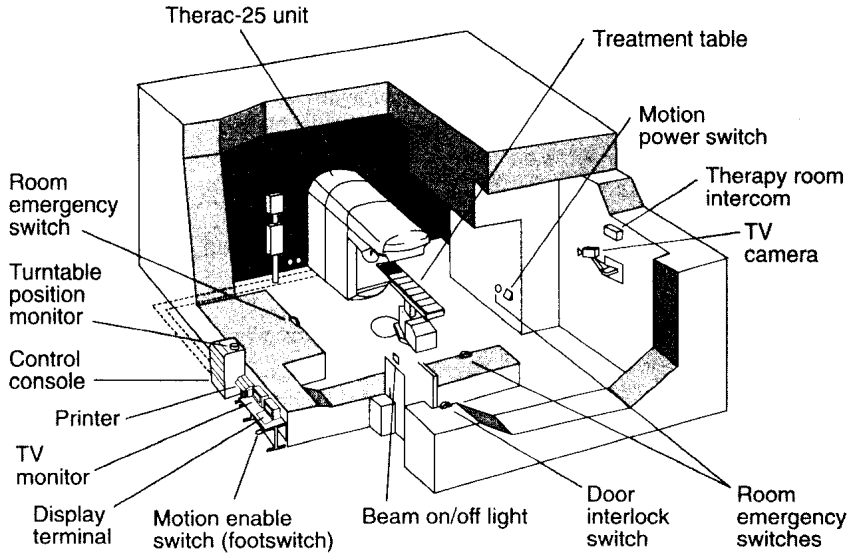
Background

- More and more systems contain **embedded computers:**
  - Mobile phones
  - Medical equipment
  - Cars
  - Satellites
  - Airplanes
- Nobody’s perfect
- Types of errors:
  - Irritating/harmless
  - Catastrophic

Background: Catastrophic Errors (1)

Therac-25 radiation therapy machine [1985–1987]

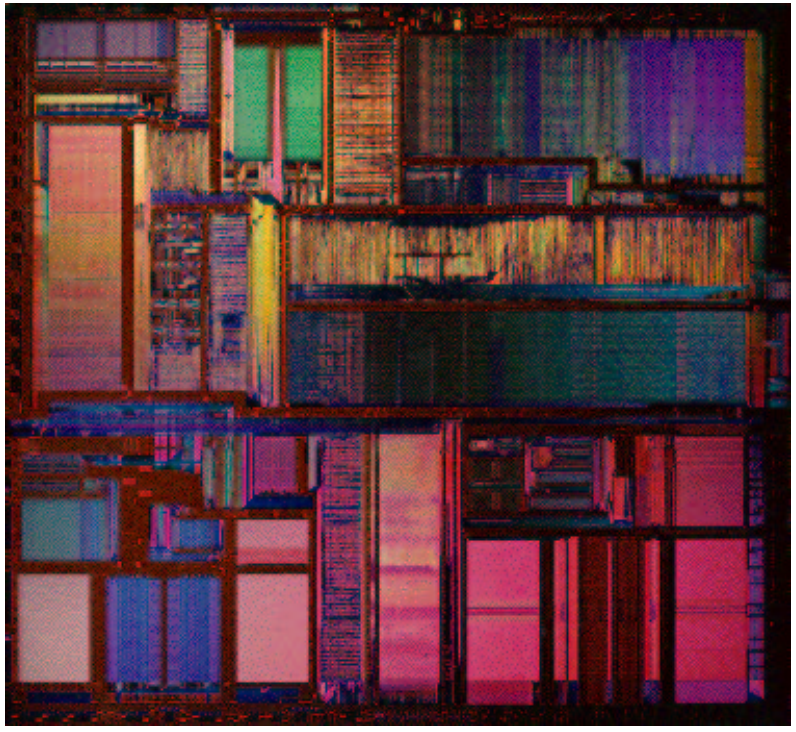
- Patients receive massive overdoses of X-rays
- Two people die, others are injured and handi-capped



Background: Catastrophic Errors (2)

Intel's Pentium division bug [Dec. 1994]

- 4.999999/14.999999 gives 0.333329, but should be 0.3333**33**29
- Intel have to replace millions of defect processors
- Costs several 100 million dollars



Background: Catastrophic Errors (3)

Ariane 5 rocket [4 June 1996]

- Floating-point conversion fails
- The computer crashes
- Rocket self-destructs 40 seconds after lift-off
- Costs ESA 600 million dollars

Background: Model Checking

- How do we ensure that a system is correct?
- Traditionally: Testing and simulation
- But testing only shows the presence, not the absence, of errors [Dijkstra 68]
- **Model checking** [Clarke & Emerson 81]:
  - Construct a *model* M of the system
  - Specify the *requirements* R
  - Find all states of M, check R in each state
- Gives a 100% guarantee of correctness
- State explosion problem

Background: Symbolic Model Checking

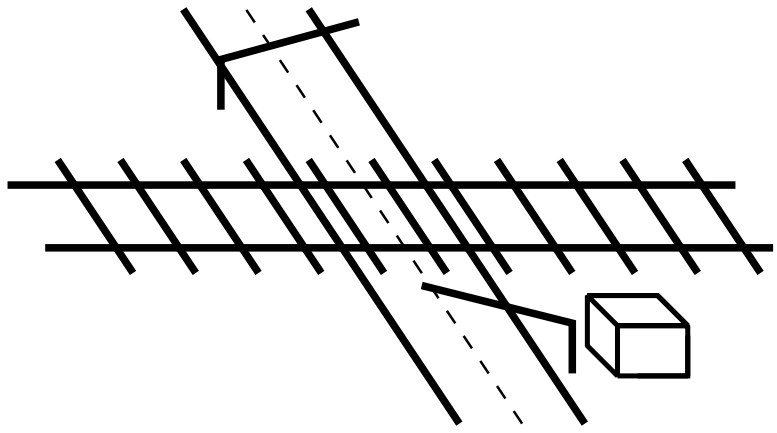
- A major breakthrough: **Binary Decision Diagrams** (BDDs) [Bryant 86]
- Efficient data structure with algorithms for manipulating **Boolean formulas**
- BDDs + model checking = **symbolic model checking** [Burch et al. 90]
- Basic idea: Represent a **set of states** as a Boolean formula
- Makes it possible to prove correctness of systems with Boolean variables

Background: Real-Time Systems

- Not all systems can be modeled using just Boolean variables
- **Real-time systems:** Boolean variables and real-valued variables (clocks)
- Clocks are used to express that events take time
- Let us consider a small example: a gate controller at a railroad crossing

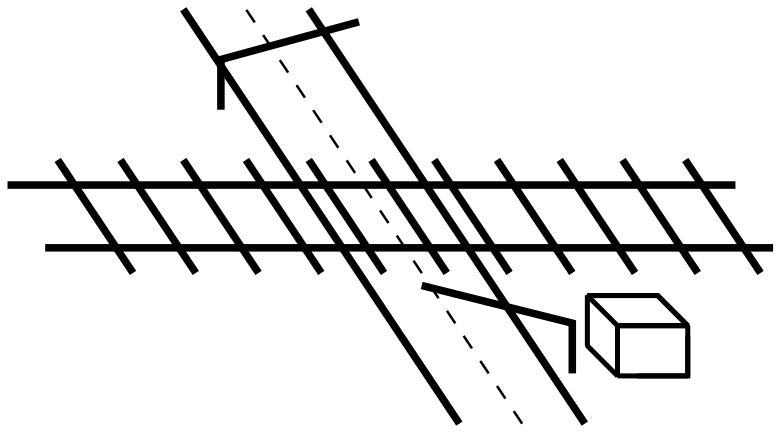
Background: An Example

- The train notifies the controller at least 2 minutes before entering the crossing, and will exit after at most 5 minutes.



Background: An Example

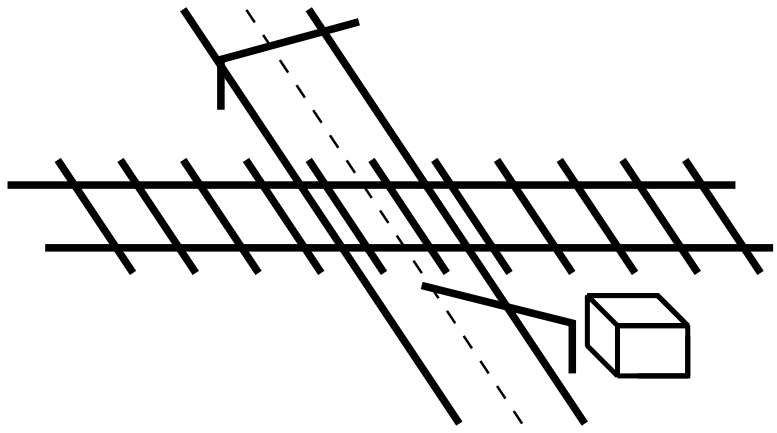
- The train notifies the controller at least 2 minutes before entering the crossing, and will exit after at most 5 minutes.



- After 1 minute, the controller will start lowering the gate. The gate will be down within 1 minute.

Background: An Example

- The train notifies the controller at least 2 minutes before entering the crossing, and will exit after at most 5 minutes.



- After 1 minute, the controller will start lowering the gate. The gate will be down within 1 minute.
- Within 1 minute after the train has exit, the controller will start raising the gate. The gate will be up within 1–2 minutes.

Background: An Example (2)

- The problem: Verify that the gate is:
  - Always closed when the train is inside
  - Never closed for more than 10 minutes

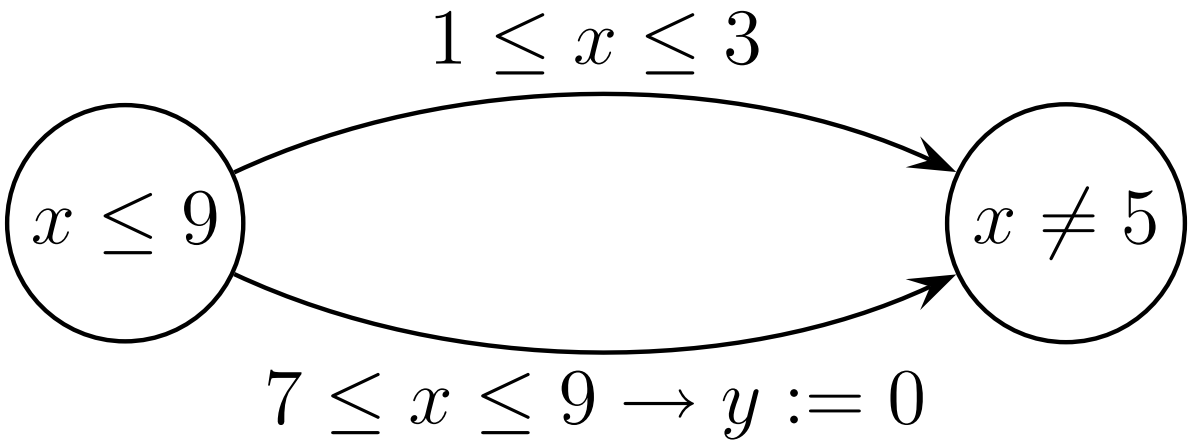
Background: An Example (2)

- The problem: Verify that the gate is:
  - Always closed when the train is inside
  - Never closed for more than 10 minutes
- To extend symbolic model checking to real-time systems, we need:
  1. A notation for building a model
  2. A logic for representing sets of states
  3. An algorithm for finding the reachable states
  4. A data structure for implementing this algorithm

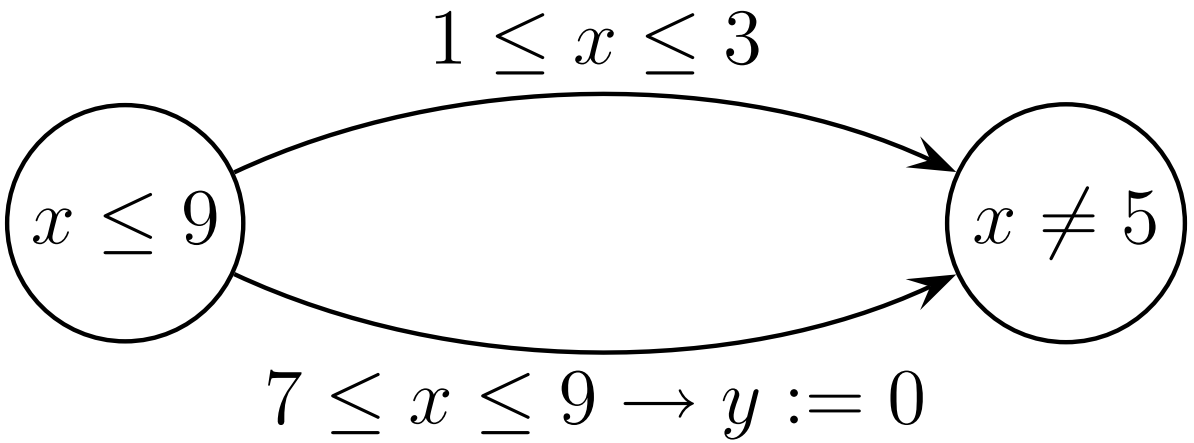
Outline of Talk

1. Background
2. **Symbolic model checking**
3. Difference decision diagrams
4. Summary of results

Symbolic Model Checking: Timed Guarded Commands



Symbolic Model Checking: Timed Guarded Commands



$$
\begin{aligned}
G &= (\{b\}, \{x, y\}, T, I) \\
T &= \left\{ \begin{array}{ll} b \wedge (1 \leq x \leq 3) \rightarrow b & := \mathbf{false} \\ b \wedge (7 \leq x \leq 9) \rightarrow b, y & := \mathbf{false}, 0 \end{array} \right\} \\
I &= \big(b \Rightarrow (x \leq 9)\big) \wedge \big(\neg b \Rightarrow (x \neq 5)\big)
\end{aligned}
$$

Symbolic Model Checking: States & Successors

- **A state** s is an interpretation of the variables:

$$s = [x \mapsto 8.0,\ y \mapsto 0.0,\ b \mapsto \mathit{false}]$$

- Executing $x < 5.0 \ \rightarrow \ b, x := \mathit{true}, 0.0$:

$$s' = [x \mapsto 0.0,\ y \mapsto 0.0,\ b \mapsto \mathit{true}]$$

- Advancing time by 2.5 :

$$s' = [x \mapsto 10.5,\ y \mapsto 2.5,\ b \mapsto \mathit{false}]$$

Symbolic Model Checking: States & Successors (2)

- **Discrete successors** for executing $g \longrightarrow \vec{v} := \vec{d}$:

$$\frac{s \models g \quad s[\vec{v} := \vec{d}] \models I}{s \longrightarrow s[\vec{v} := \vec{d}]}$$

Symbolic Model Checking: States & Successors (2)

- **Discrete successors** for executing $g \longrightarrow \vec{v} := \vec{d}$:

$$\frac{s \models g \quad s[\vec{v} := \vec{d}] \models I}{s \longrightarrow s[\vec{v} := \vec{d}]}$$

- **Timed successors** for advancing time by δ:

$$\frac{\delta \geq 0 \quad \forall \delta'.0 \leq \delta' \leq \delta : s[\vec{c} := \vec{c} + \delta'] \models I}{s \xrightarrow{\delta} s[\vec{c} := \vec{c} + \delta]}$$

Symbolic Model Checking: States & Successors (2)

- **Discrete successors** for executing $g \rightarrow \vec{v} := \vec{d}$:

$$\frac{s \models g \quad s[\vec{v} := \vec{d}] \models I}{s \rightarrow s[\vec{v} := \vec{d}]}$$

- **Timed successors** for advancing time by δ:

$$\frac{\delta \geq 0 \quad \forall \delta'.0 \leq \delta' \leq \delta : s[\vec{c} := \vec{c} + \delta'] \models I}{s \xrightarrow{\delta} s[\vec{c} := \vec{c} + \delta]}$$

✓ A notation for modeling real-time systems

Symbolic Model Checking: The z-Trick

- Let the variable z represent “zero”

Symbolic Model Checking: The z-Trick

- Let the variable z represent “zero”
- Replace $x \leq d$ by $x - z \leq d$, and
  Replace $x := d$ by $x := z + d$

Symbolic Model Checking: The z-Trick

- Let the variable z represent “zero”
- Replace $x \leq d$ by $x - z \leq d$, and
  Replace $x := d$ by $x := z + d$
- Replace b_i by $x_i - x_i' \leq 0$

Symbolic Model Checking: The z-Trick

- Let the variable z represent "zero"
- Replace $x \leq d$ by $x - z \leq d$, and
  Replace $x := d$ by $x := z + d$
- Replace b_i by $x_i - x_i' \leq 0$
- For example, the command $x \leq 5 \rightarrow y := 1$
  becomes $x - z \leq 5 \rightarrow y := z + 1$

Symbolic Model Checking: The z-Trick

- Let the variable z represent "zero"
- Replace $x \leq d$ by $x - z \leq d$, and
  Replace $x := d$ by $x := z + d$
- Replace b_i by $x_i - x_i' \leq 0$
- For example, the command $x \leq 5 \rightarrow y := 1$
  becomes $x - z \leq 5 \rightarrow y := z + 1$
- A set of states can now be represented by a formula of the form:

$$\phi \quad ::= \quad x - y \leq d \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

Symbolic Model Checking: The z-Trick

- Let the variable z represent "zero"
- Replace $x \leq d$ by $x - z \leq d$, and
  Replace $x := d$ by $x := z + d$
- Replace b_i by $x_i - x_i' \leq 0$
- For example, the command $x \leq 5 \rightarrow y := 1$
  becomes $x - z \leq 5 \rightarrow y := z + 1$
- A set of states can now be represented by a formula of the form:

$$\phi \quad ::= \quad x - y \leq d \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

- ✓ A logic for representing sets of states

Symbolic Model Checking: Reachable States

- Let $\mathbf{post}(\phi)$ be a function that returns a formula for the discrete and timed successors of ϕ

Symbolic Model Checking: Reachable States

- Let $\mathbf{post}(\phi)$ be a function that returns a formula for the discrete and timed successors of ϕ
- Computing the **reachable states**:

$$
\begin{array}{l}
\phi \leftarrow \phi_0 \\
\mathbf{repeat} \\
\quad \phi' \leftarrow \phi \\
\quad \phi \leftarrow \phi \vee \mathbf{post}(\phi) \\
\mathbf{until}\ \phi = \phi'
\end{array}
$$

Symbolic Model Checking: Reachable States

- Let $\mathbf{post}(\phi)$ be a function that returns a formula for the discrete and timed successors of ϕ
- Computing the **reachable states**:

$$
\begin{array}{l}
\phi \leftarrow \phi_0 \\
\mathbf{repeat} \\
\quad \phi' \leftarrow \phi \\
\quad \phi \leftarrow \phi \vee \mathbf{post}(\phi) \\
\mathbf{until}\ \phi = \phi'
\end{array}
$$

- How to define
  - $\mathbf{post}_{\mathrm{d}}(\phi)$ for executing commands?
  - $\mathbf{post}_{\mathrm{t}}(\phi)$ for advancing time?

Symbolic Model Checking: Discrete Successors

- Executing the command $g \rightarrow \vec{v} := \vec{d}$ according to the rule

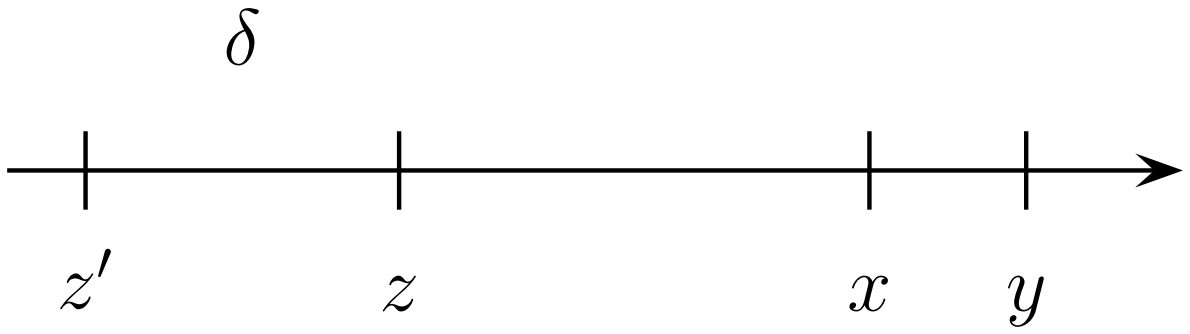
$$\frac{s \models g \quad s[\vec{v} := \vec{d}] \models I}{s \rightarrow s[\vec{v} := \vec{d}]}$$

can be defined as:

$$\mathbf{post}_{\mathrm{d}}(\phi, g \rightarrow \vec{v} := \vec{d}) \;=\; \big(\exists \vec{v}(\phi \wedge g)\big) \wedge \big(\vec{v} = \vec{d}\big) \wedge I$$

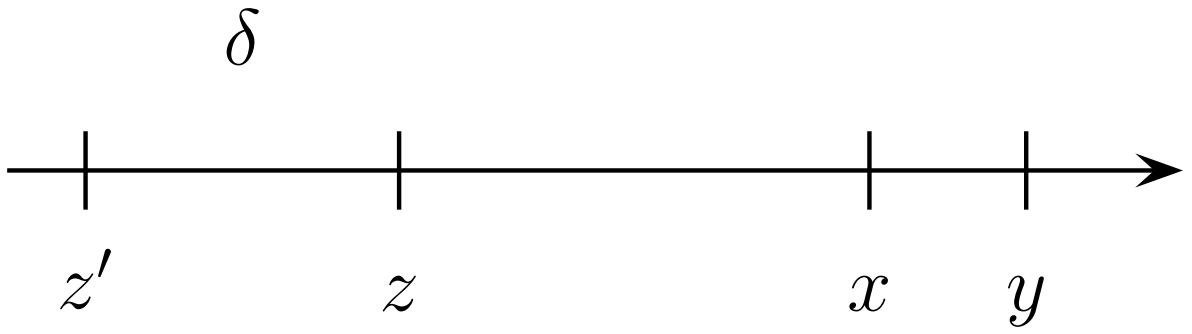
Symbolic Model Checking: Timed Successors

- Advancing all clocks by δ :



Symbolic Model Checking: Timed Successors

- Advancing all clocks by δ :



Symbolically:

$$\begin{aligned}\phi[\vec{c} := \vec{c} + \delta] &= \phi[z := z - \delta] \\ &= \big(\exists z.(\phi \wedge z' = z - \delta)\big)[z/z']\end{aligned}$$

Symbolic Model Checking: Timed Successors (2)

- Advancing time by δ according to the rule

$$\frac{\delta \geq 0 \quad \forall \delta'.0 \leq \delta' \leq \delta : s[\vec{c} := \vec{c} + \delta'] \models I}{s \xrightarrow{\delta} s[\vec{c} := \vec{c} + \delta]}$$

can be defined as:

$$\mathbf{post}_{\mathrm{t}}(\phi, \delta) \;=\; \big(\exists z.(\phi \wedge z' = z - \delta \wedge P_{\mathrm{post}})\big)[z/z']$$

Symbolic Model Checking: Timed Successors (2)

- Advancing time by δ according to the rule

$$\frac{\delta \geq 0 \quad \forall \delta'.0 \leq \delta' \leq \delta : s[\vec{c} := \vec{c} + \delta'] \models I}{s \xrightarrow{\delta} s[\vec{c} := \vec{c} + \delta]}$$

  can be defined as:

$$\mathbf{post}_{\mathrm{t}}(\phi, \delta) \;=\; \big(\exists z.(\phi \wedge z' = z - \delta \wedge P_{\mathrm{post}})\big)[z/z']$$

- $P_{\mathrm{post}} = \; z' \leq z \;\wedge\; \forall z''.(z' \leq z'' \leq z \Rightarrow I[z''/z])$
  specifies whether z' is a legal zero-point

Symbolic Model Checking: Timed Successors (3)

- Advancing time by an arbitrary amount can be defined as:

$$\begin{aligned} \mathbf{post}_{\mathrm{t}}(\phi) &= \bigvee_{\delta \in \mathbb{R}} \mathbf{post}_{\mathrm{t}}(\phi, \delta) \\ &= \big(\exists z.(\phi \wedge P_{\mathrm{post}})\big)[z/z'] \end{aligned}$$

Symbolic Model Checking: Timed Successors (3)

- Advancing time by an arbitrary amount can be defined as:

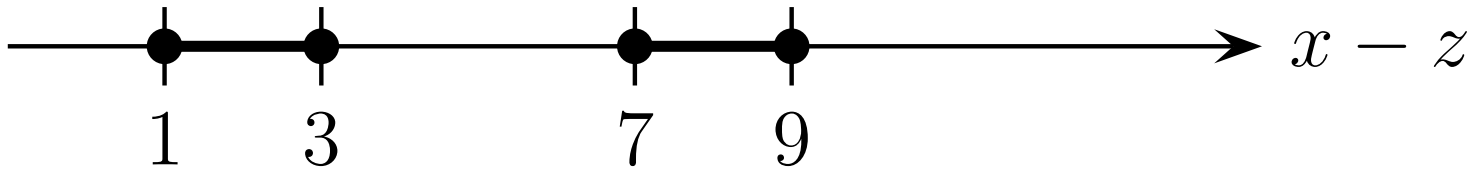
$$\begin{aligned}\mathbf{post}_{\mathrm{t}}(\phi) &= \bigvee_{\delta\in\mathbb{R}} \mathbf{post}_{\mathrm{t}}(\phi,\delta)\\ &= \big(\exists z.(\phi \wedge P_{\mathrm{post}})\big)[z/z']\end{aligned}$$

- ✓ A symbolic algorithm for constructing the reachable set of states

Symbolic Model Checking: Example

- Consider the formula

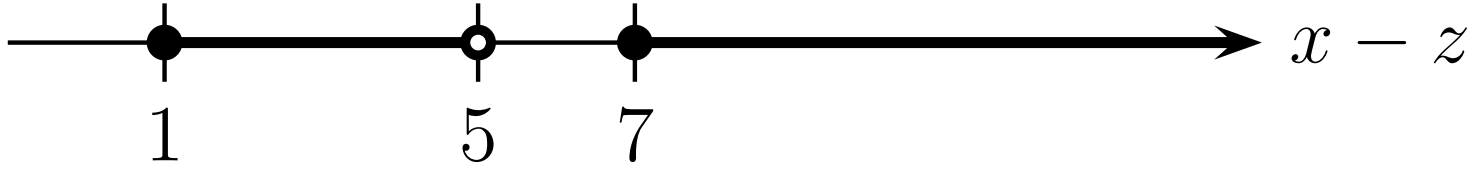
$$\phi \;=\; (1 \leq x - z \leq 3) \vee (7 \leq x - z \leq 9)$$



and the state-invariant $I \;=\; x - z \neq 5$.

- Advancing time from ϕ:

$$\mathbf{post}_{\mathrm{t}}(\phi) \;=\; (1 \leq x - z < 5) \vee (7 \leq x - z)$$



Symbolic Model Checking: Nondeterministic Assignments

- We can extend timed guarded commands with a nondeterministic assignment operator:

$$\vec{v} := \mathbf{any}\ \vec{v'}.\phi$$

- Ordinary assignments:

$$g \rightarrow x := y + d \quad \equiv \quad x := \mathbf{any}\ x' \,.\, (g \wedge x' = y + d)$$

- Advancing time: $z := \mathbf{any}\ z' \,.\, P_{\mathrm{post}}$
- Simplifies the semantics:

$$\mathbf{post}(\phi_0, \vec{v} := \mathbf{any}\ \vec{v'} \,.\, \phi) = \exists \vec{v}.(\phi_0 \wedge \phi)[\vec{v}/\vec{v'}]$$

Symbolic Model Checking: Related Work

Henzinger, Nicollin, Sifakis & Yovine [HNSY94] have also presented a symbolic model checking algorithm for real-time systems

1. Use two types of constraints: $x \leq d$ and $x - y \leq d$
2. Advance time using quantifier elimination in the more complicated first-order theory of reals with addition and order
3. Their version of timed guarded commands cannot model the progression of time

Symbolic Model Checking: Summary

- ✓ We have a notation for specifying real-time systems

Symbolic Model Checking: Summary

- ✓ We have a notation for specifying real-time systems
- ✓ We have a logic for representing sets of states:

$$\phi \quad ::= \quad x - y \leq d \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi$$

Symbolic Model Checking: Summary

- ✓ We have a notation for specifying real-time systems
- ✓ We have a logic for representing sets of states:

$$\phi \quad ::= \quad x - y \leq d \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi$$

- ✓ We have an algorithm for constructing the reachable set of states

Symbolic Model Checking: Summary

- ✓ We have a notation for specifying real-time systems
- ✓ We have a logic for representing sets of states:

$$\phi \quad ::= \quad x - y \leq d \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi$$

- ✓ We have an algorithm for constructing the reachable set of states
- 👉 Now we need a data structure for manipulating this logic (construct formulas, decide validity)

Outline of Talk

1. Background
2. Symbolic model checking
3. **Difference decision diagrams**
4. Summary of results

Difference Decision Diagrams: Introduction

- Implicit representation of the logic :

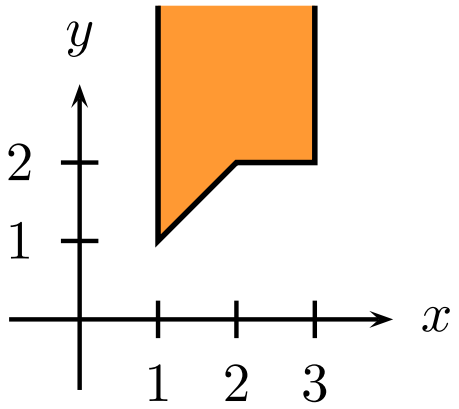
$$\phi \ ::= \ x - y \leq d \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists x.\phi \, .$$

Difference Decision Diagrams: Introduction

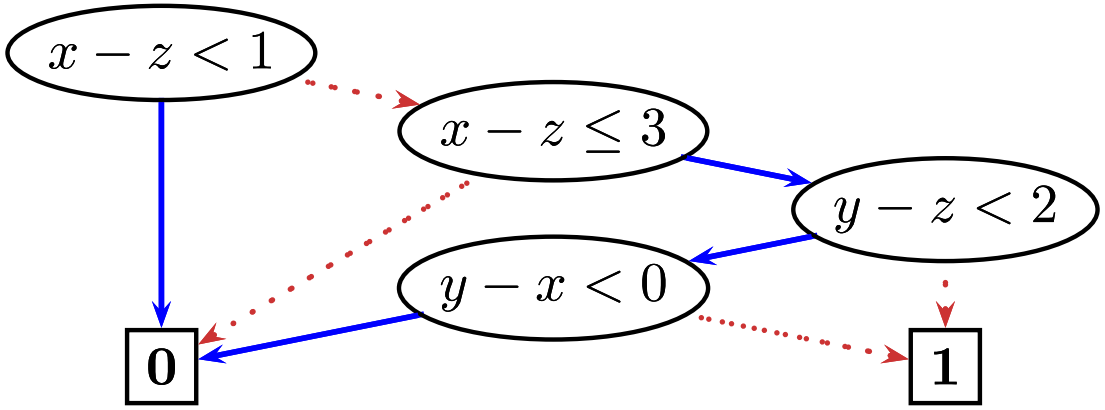
- Implicit representation of the logic :

$$\phi \ ::= \ x - y \leq d \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists x.\phi \, .$$

- $\phi \ = \ 1 \leq x - z \leq 3 \ \wedge \ (y - z \geq 2 \ \vee \ y - x \geq 0)$



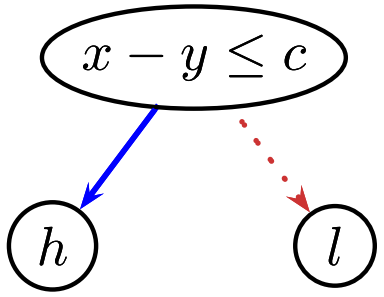
(x, y)-plot for $z = 0$



Difference decision diagram

DDDs: Local Reduction Rules

- A **difference decision diagram** is a directed, acyclic graph with two types of nodes:

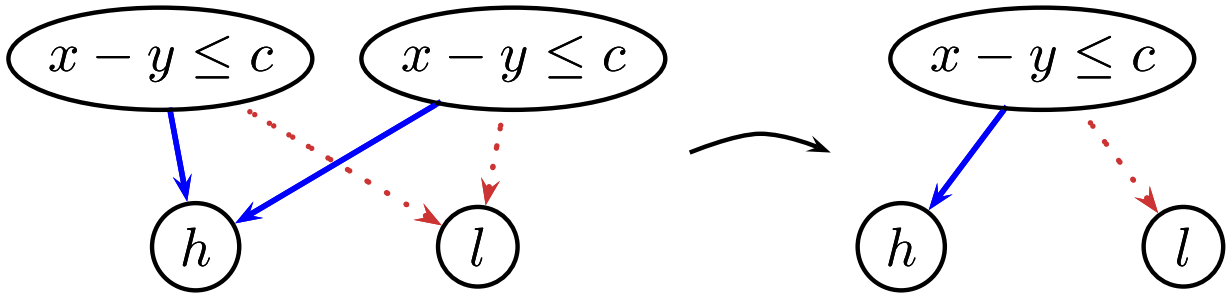


if $x - y \leq c$ **then** h **else** l

false respectively **true**

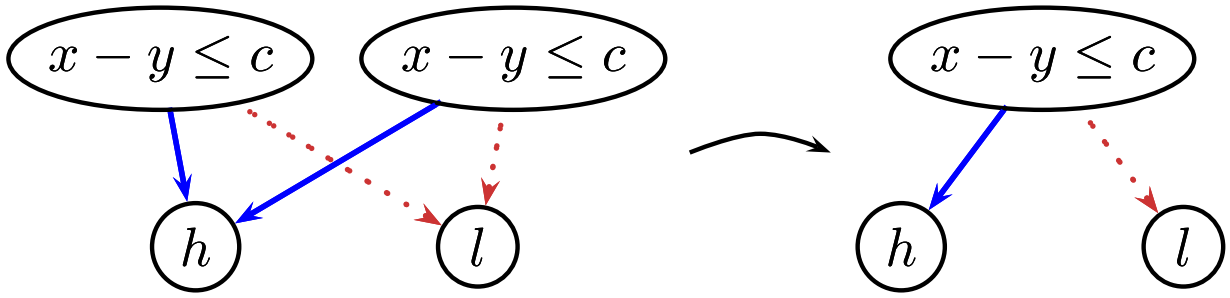
DDDs: Local Reduction Rules (2)

- No duplicate vertices:

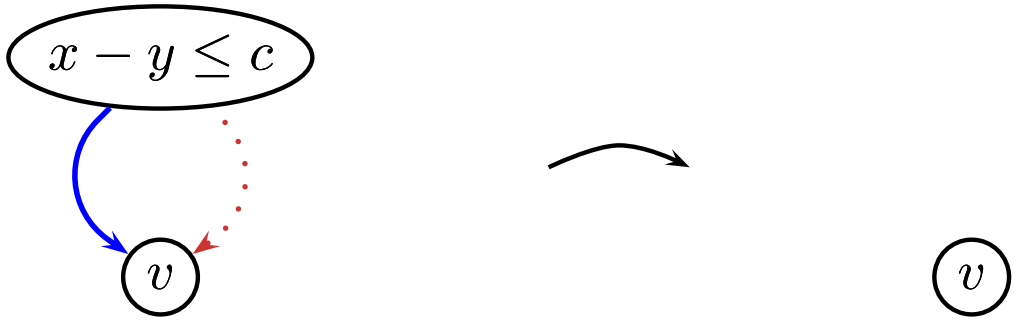


DDDs: Local Reduction Rules (2)

- No duplicate vertices:

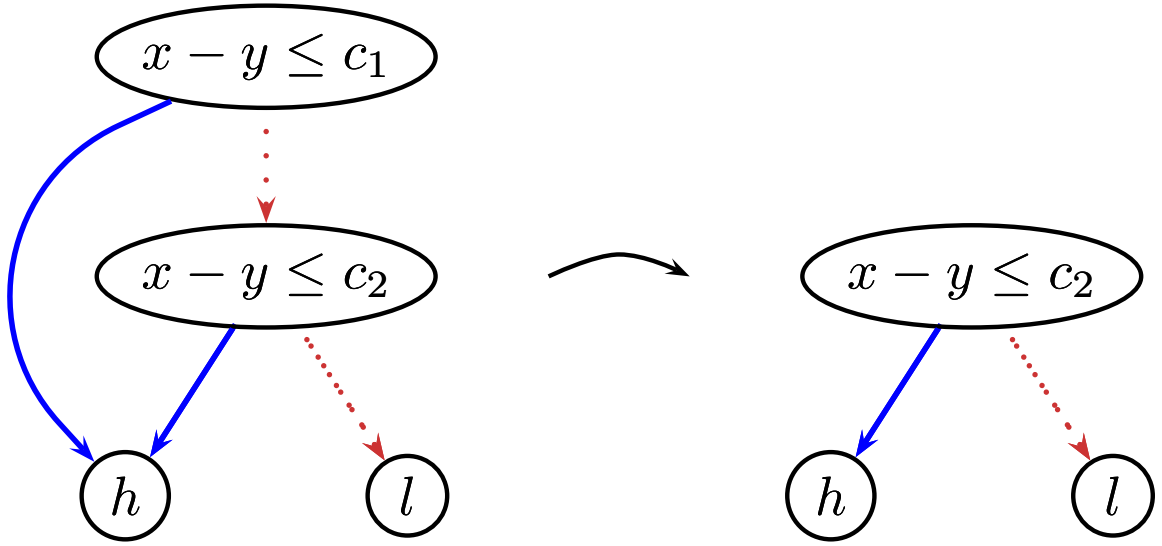


- Different high- and low-branches:



DDDs: Local Reduction Rules (3)

- No redundant tests:

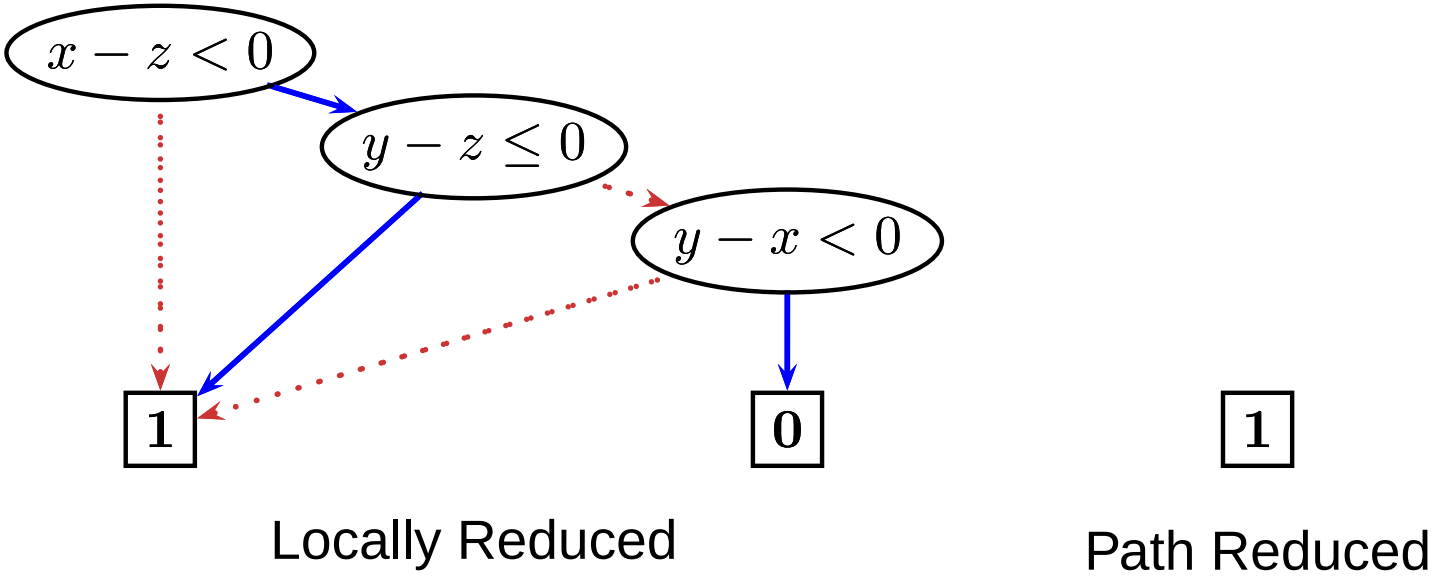


DDDs: Path Reduction

- A **path-reduced DDD** is a locally reduced DDD where all paths are feasible

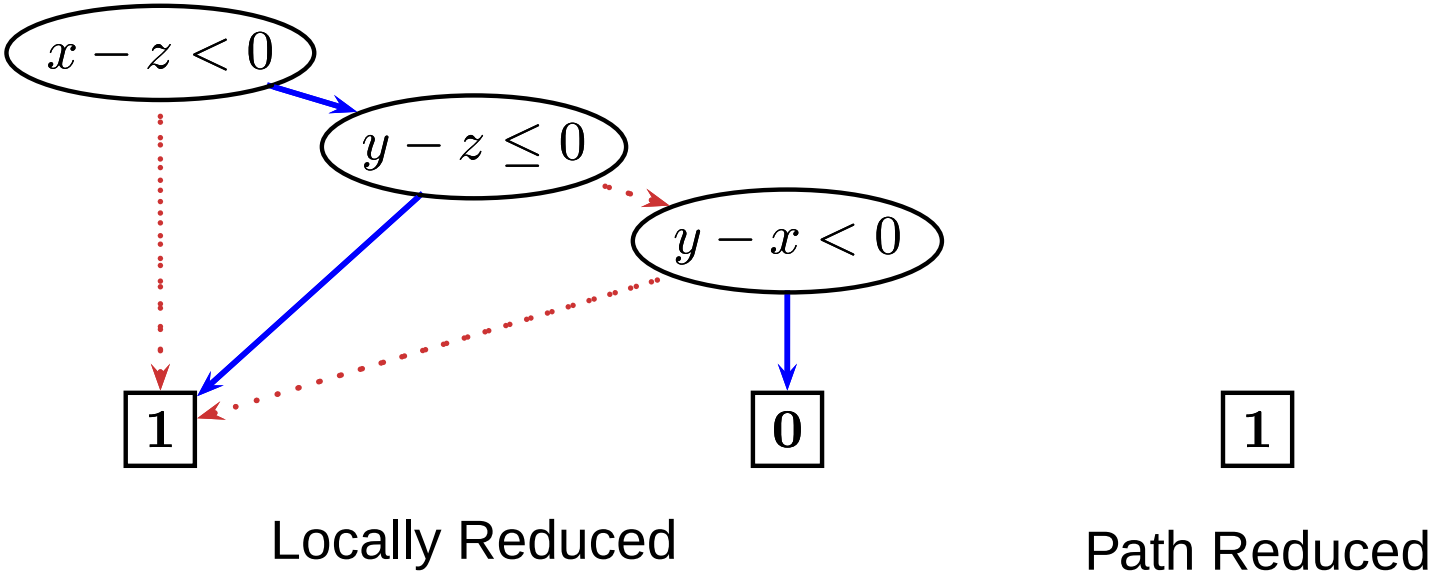
DDDs: Path Reduction

- A **path-reduced DDD** is a locally reduced DDD where all paths are feasible
- For example:



DDDs: Path Reduction

- A **path-reduced DDD** is a locally reduced DDD where all paths are feasible
- For example:



- **Theorem:** A path-reduced DDD is a tautology if and only if it is the terminal vertex 1.

DDDs: Construction & Manipulation

- DDD for $x - y \leq c$: $\mathit{mk}(x, y, c, 1, 0)$

DDDs: Construction & Manipulation

- DDD for $x - y \leq c$: ***mk***$(x, y, c, 1, 0)$
- DDD for $\phi_1 \wedge \phi_2$: ***apply***$(\wedge, u_1, u_2)$ using a generalization of the BDD *apply* algorithm

DDDs: Construction & Manipulation

- DDD for $x - y \leq c$: *mk*$(x, y, c, 1, 0)$
- DDD for $\phi_1 \wedge \phi_2$: *apply*$(\wedge, u_1, u_2)$ using a generalization of the BDD *apply* algorithm
- DDD for $\neg\phi$: *apply*$(\neg_1, u, \_)$

DDDs: Construction & Manipulation

- DDD for $x - y \leq c$: *mk*$(x, y, c, 1, 0)$
- DDD for $\phi_1 \wedge \phi_2$: *apply*$(\wedge, u_1, u_2)$
  using a generalization of the BDD *apply* algorithm
- DDD for $\neg\phi$: *apply*$(\neg_1, u, \_)$
- DDD for $\exists x.\phi$: *exists*(x, u)
  using Fourier–Motzkin quantifier elimination

DDDs: Construction & Manipulation

- DDD for $x - y \leq c$: *mk*$(x, y, c, 1, 0)$
- DDD for $\phi_1 \wedge \phi_2$: *apply*$(\wedge, u_1, u_2)$
  using a generalization of the BDD *apply* algorithm
- DDD for $\neg\phi$: *apply*$(\neg_1, u, \_)$
- DDD for $\exists x.\phi$: *exists*(x, u)
  using Fourier–Motzkin quantifier elimination
- DDD is a tautology: *path_reduce*$(u) = \mathbf{1}$
  using Bellman–Ford's algorithm

DDDs: Implementation

- **DDDLIB:** Kernel written in C with high-level interfaces for C++ and Standard ML
  [partly based on joint work with J. Lichtenberg]

DDDs: Implementation

- **DDDLIB:** Kernel written in C with high-level interfaces for C++ and Standard ML
  [partly based on joint work with J. Lichtenberg]
- Can be downloaded from [www.it.edu/research/ddd] for non-commercial use

DDDs: Implementation

- **DDDLIB:** Kernel written in C with high-level interfaces for C++ and Standard ML [partly based on joint work with J. Lichtenberg]
- Can be downloaded from [www.it.edu/research/ddd] for non-commercial use
- High-level interface for Objective Caml also available [Sorea 01a]

DDDs: Applications

- DDDs have been integrated in a verification tool for **infinite state systems** [Abdulla & Nylén 00]

DDDs: Applications

- DDDs have been integrated in a verification tool for **infinite state systems** [Abdulla & Nylén 00]
- DDDs are part of SRIs **TEMPO model checker** for event-recording automata [Sorea 01b]

DDDs: Applications

- DDDs have been integrated in a verification tool for **infinite state systems** [Abdulla & Nylén 00]
- DDDs are part of SRIs **TEMPO model checker** for event-recording automata [Sorea 01b]
- DDDs are used in the **CALCULEMUS EU project** for bounded model checking [Cimatti et al.]

Outline of Talk

1. Background
2. Symbolic model checking
3. Difference decision diagrams
4. **Summary of results**

Summary of Results

- **Symbolic model checking:** I have shown that advancing time in the post operator can be performed as an **existential quantification** in the simple logic:

$$\phi \quad ::= \quad x - y \leq d \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi$$

Summary of Results

- **Symbolic model checking:** I have shown that advancing time in the post operator can be performed as an **existential quantification** in the simple logic:

$$\phi \quad ::= \quad x - y \leq d \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi$$

- **Difference decision diagrams:** I have extended BDDs with real-valued variables and obtained a data structure for representing and deciding validity of formulas of the form ϕ

Summary of Results (2)

- **Timed Guarded Commands:** I have shown how to model real-time systems in terms of nondeterministic assignments $\vec{x} := \mathbf{any}\ \vec{x'}.\phi$.

Summary of Results (2)

- **Timed Guarded Commands:** I have shown how to model real-time systems in terms of nondeterministic assignments $\vec{x} := \mathbf{any}\ \vec{x'}.\phi$.
- **Experimental results:** Symbolic model checking using DDDs
  - Works well on fx scheduling protocols, comparable with KRONOS & UPPAAL
  - Works not quite as well on fx timed asynchronous circuits

- Questions?