On the Decidability of Metric Temporal Logic

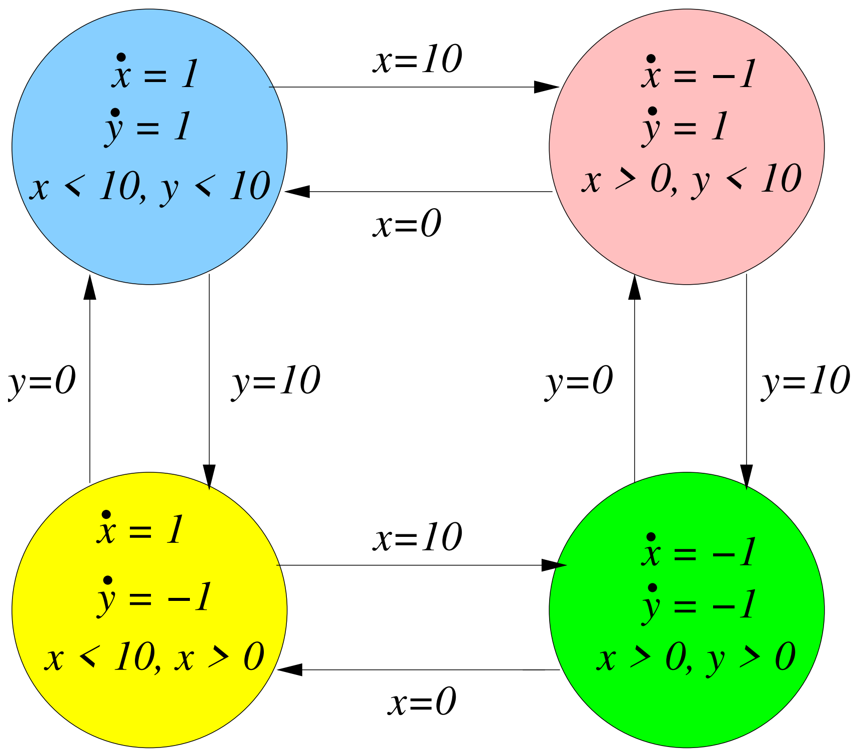
Joël Ouaknine and James Worrell

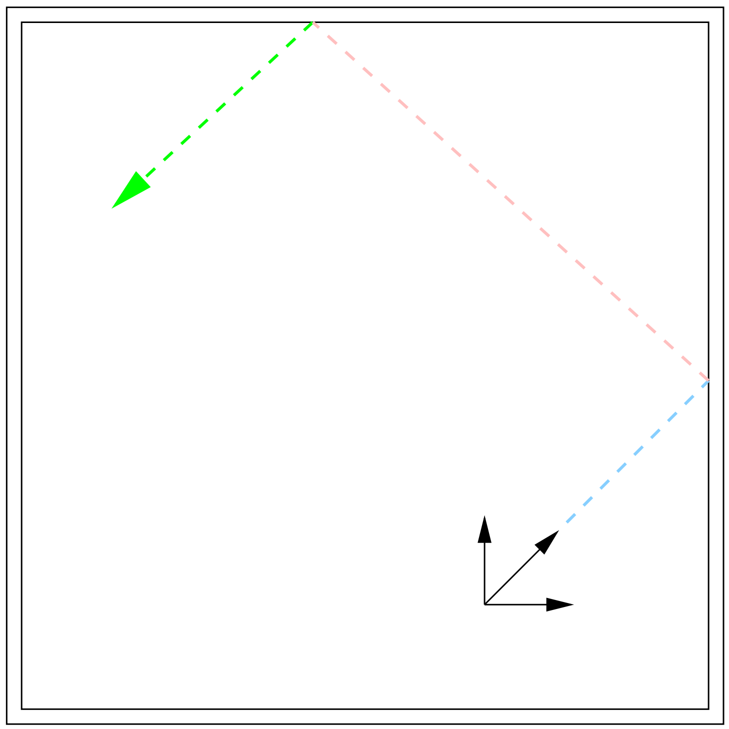
Oxford University Computing Laboratory

FOSSACS, March 2006

Real-time Models

– Rectangular hybrid automata (Alur, Courcoubetis, Henzinger):





Order-theoretic vs Metric Specifications

- "After a blue state, a pink state always precedes a green state."

$$\Box(\mathit{blue} \longrightarrow (\neg \mathit{green}\ \mathcal{U}\ \mathit{pink}))\ .$$

- "After entering a blue state, the ball enters a pink state within 25 time units."

$$\Box(\mathit{blue} \longrightarrow \Diamond_{[0,25]}\mathit{pink})$$

Metric Temporal Logic

- **Metric Temporal Logic (MTL).**
  - MTL = Linear temporal logic + constrained modalities
- $\varphi ::= a \mid \top \mid \varphi \wedge \varphi \mid \neg\varphi \mid \Box_I \varphi \mid \Diamond_I \varphi \mid \varphi\, \mathcal{U}_I\, \varphi \mid \ldots$
  - $\Box_I \varphi$: All events occurring in the time interval I satisfy φ.
  - $\Diamond_I \varphi$: There exists an event in the time interval I that satisfies φ.

Timed Words

- A **timed word** is an infinite sequence of **timed events**:

$$\langle (t_0, a_0), (t_1, a_1), (t_2, a_2), (t_3, a_3), \ldots \rangle$$

Timed Words

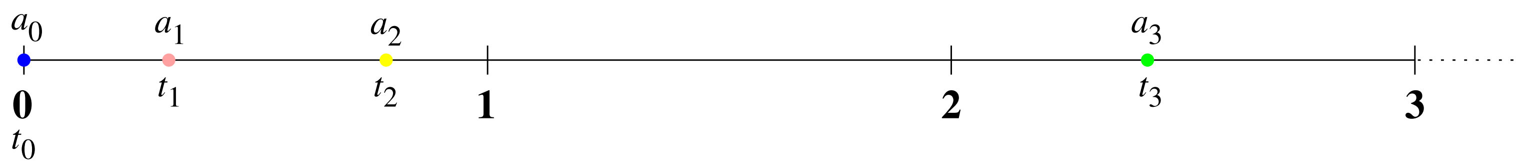
- A **timed word** is an infinite sequence of **timed events**:

$$\langle (t_0, a_0), (t_1, a_1), (t_2, a_2), (t_3, a_3), \ldots \rangle$$

Timed Words

- A **timed word** is an infinite sequence of **timed events**:

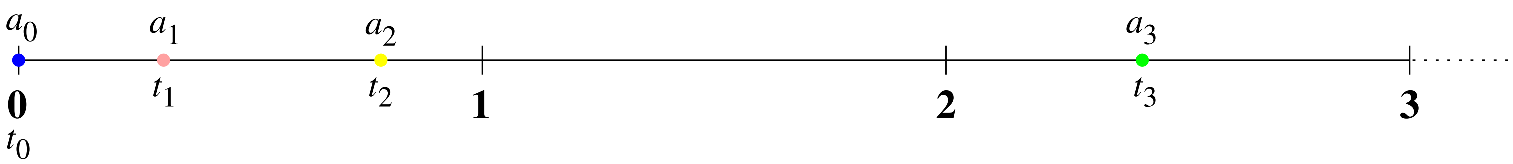
$$\langle (t_0, a_0), (t_1, a_1), (t_2, a_2), (t_3, a_3), \ldots \rangle$$



Timed Words

- A **timed word** is an infinite sequence of **timed events**:

$$\langle (t_0, a_0), (t_1, a_1), (t_2, a_2), (t_3, a_3), \ldots \rangle$$

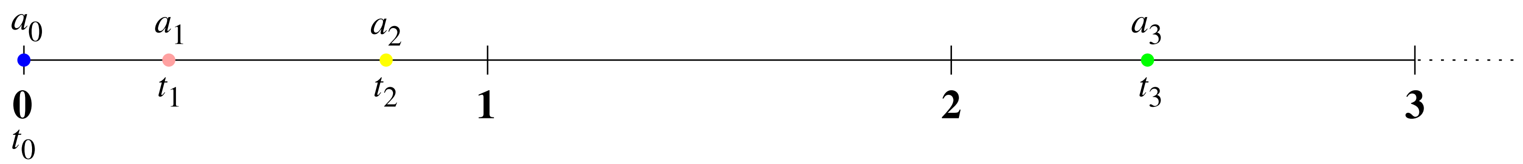


- Define satisfaction relation $w \models \varphi$.

Timed Words

- A **timed word** is an infinite sequence of **timed events**:

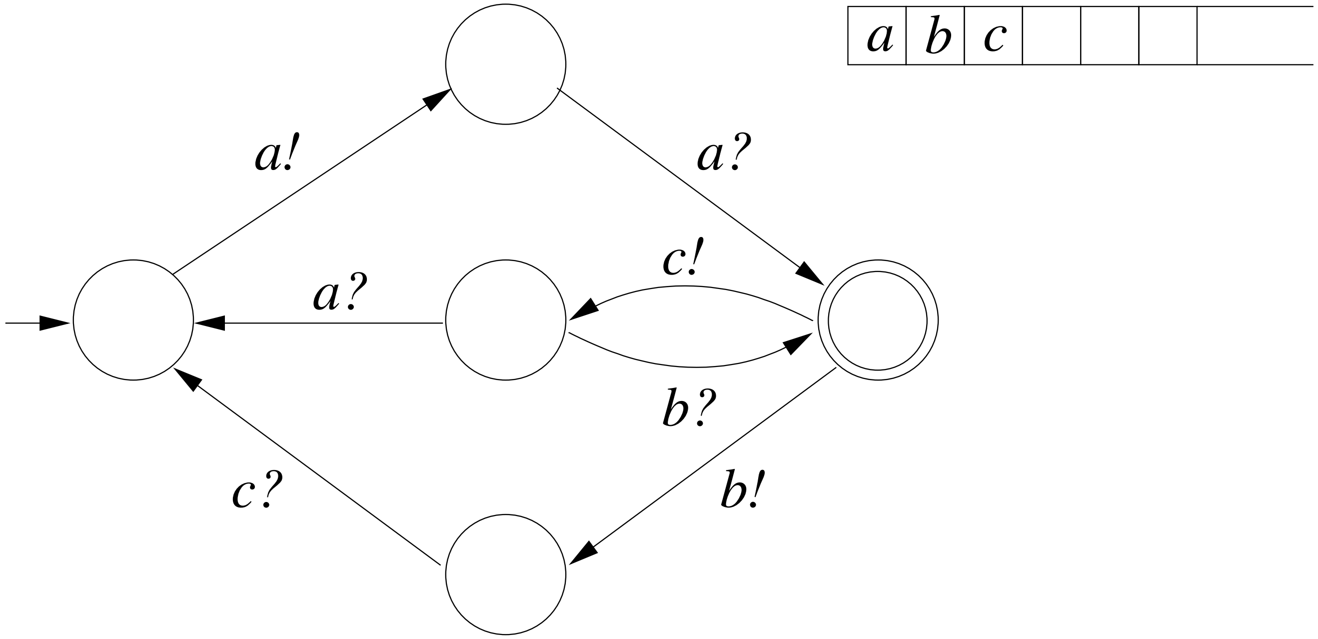
$$\langle (t_0, a_0), (t_1, a_1), (t_2, a_2), (t_3, a_3), \ldots \rangle$$



- Define satisfaction relation $w \models \varphi$.
- Investigate **satisfiability problem**.

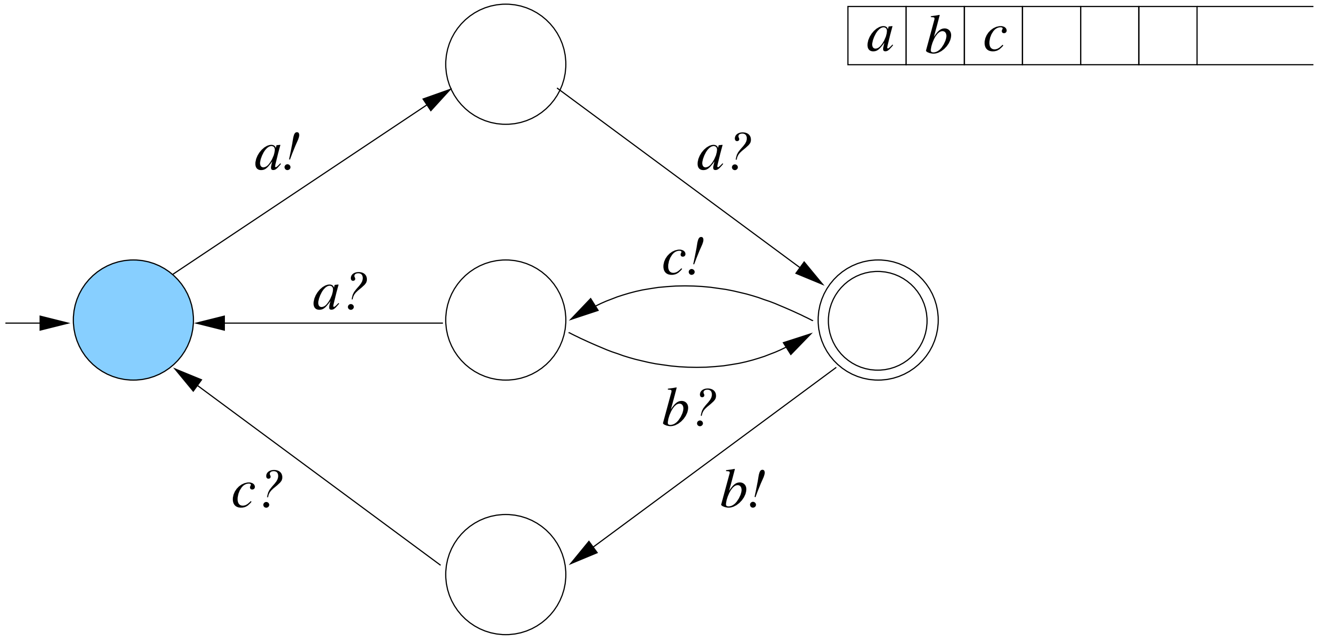
Channel Machines

Our undecidability result for MTL uses **channel machines**.



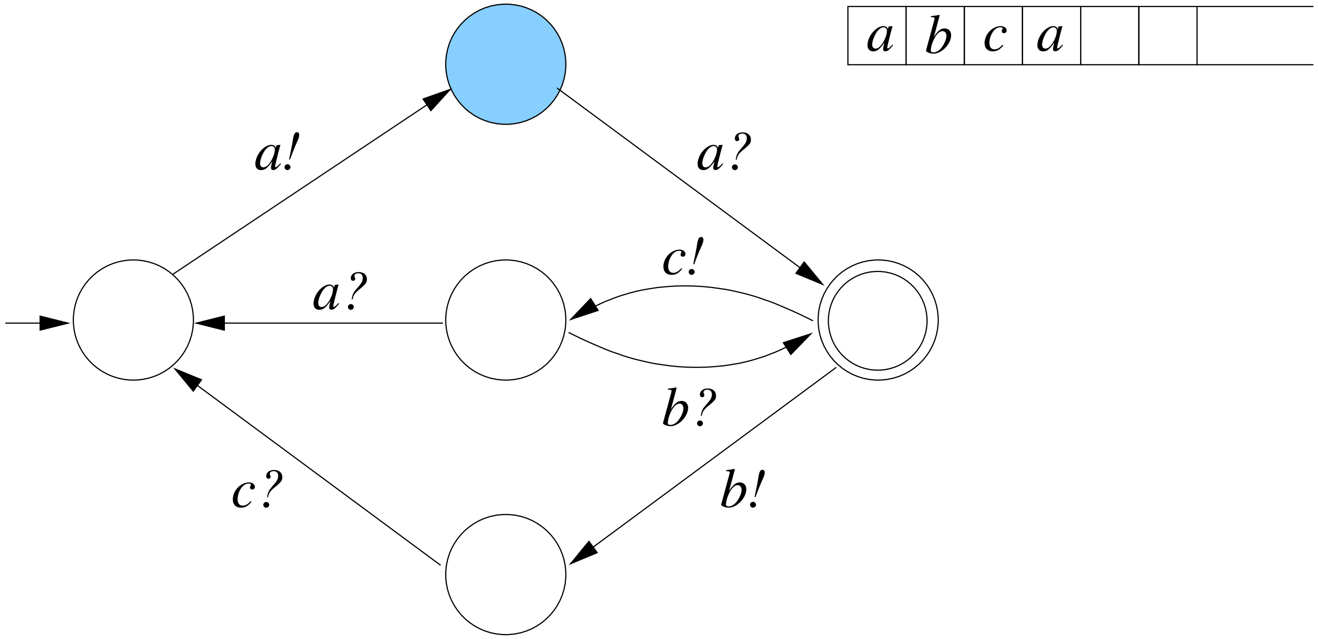
Channel Machines

Our undecidability result for MTL uses **channel machines**.



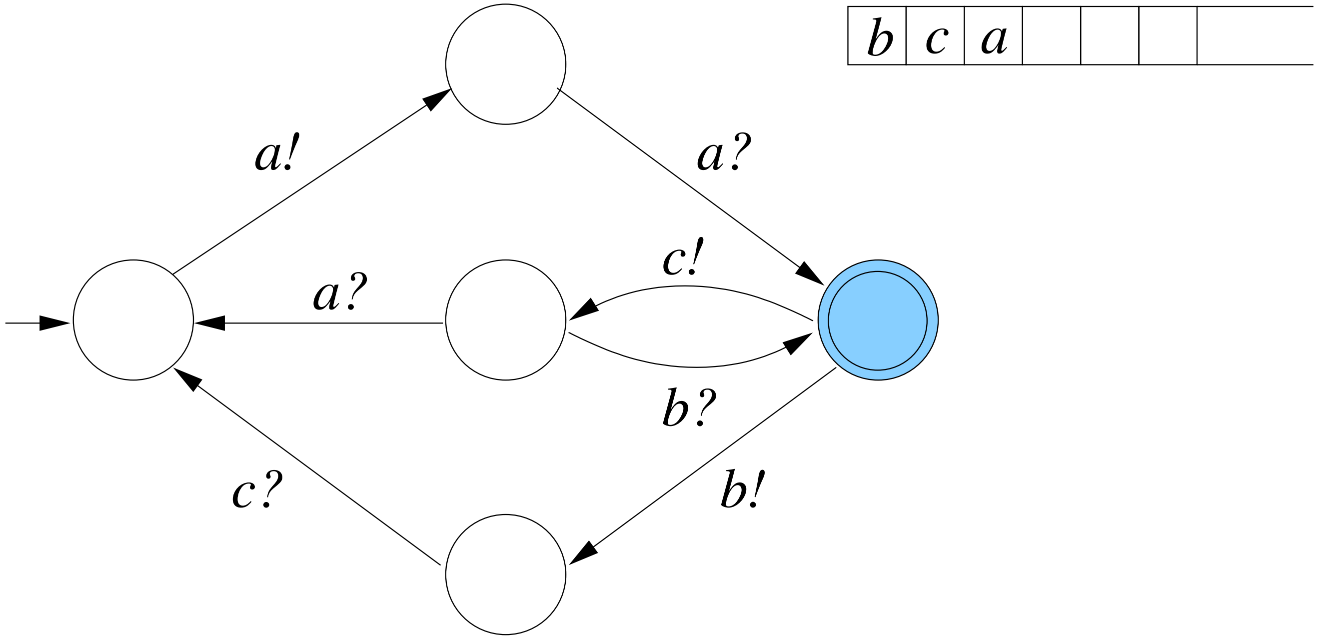
Channel Machines

Our undecidability result for MTL uses **channel machines**.



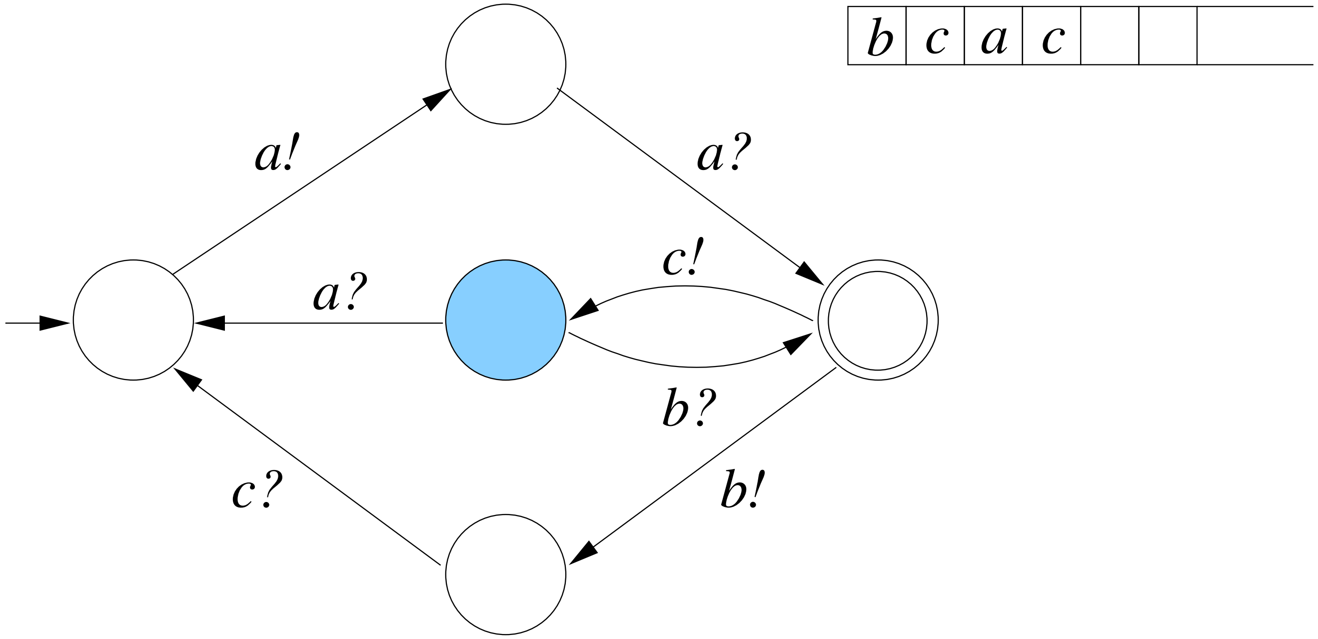
Channel Machines

Our undecidability result for MTL uses **channel machines**.



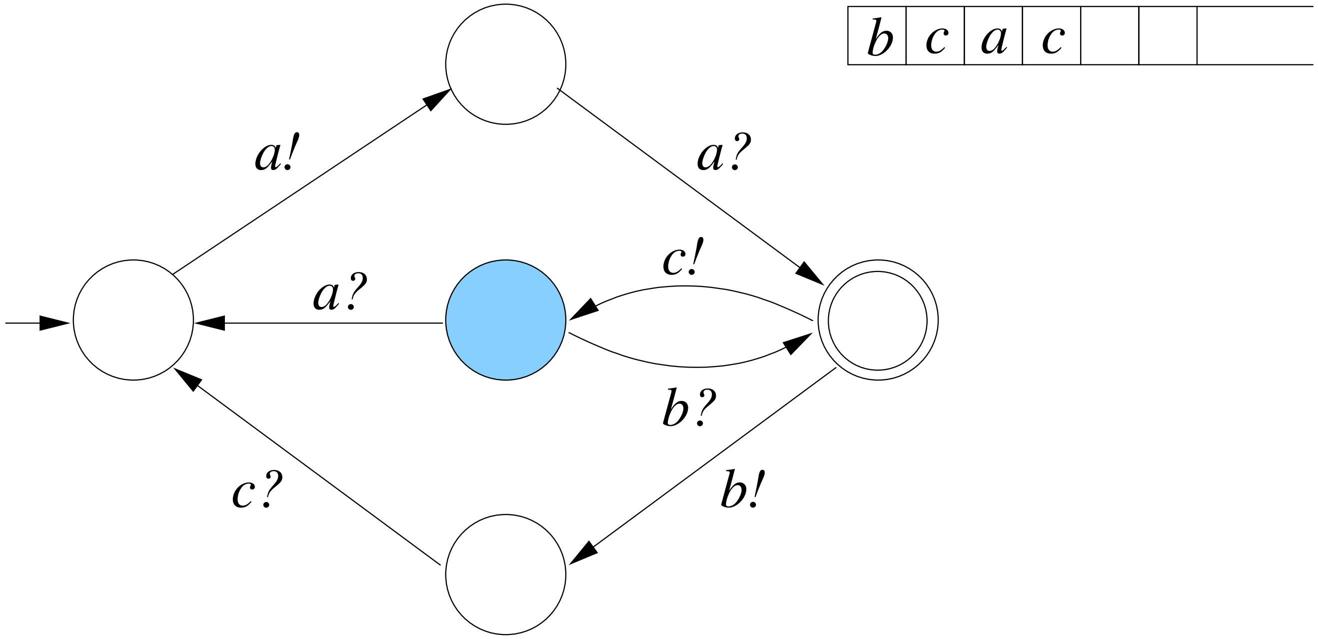
Channel Machines

Our undecidability result for MTL uses **channel machines**.



Channel Machines

Our undecidability result for MTL uses **channel machines**.



Thm. The halting problem for channel machines is undecidable.

Undecidability for MTL

Claim. MTL is **undecidable** (over both finite and infinite words).

Undecidability for MTL

Claim. MTL is **undecidable** (over both finite and infinite words).

Proof: Let $\mathcal{C}$ be a channel machine. Encode the computations of $\mathcal{C}$ as timed words:

Undecidability for MTL

Claim. MTL is **undecidable** (over both finite and infinite words).

Proof: Let $\mathcal{C}$ be a channel machine. Encode the computations of $\mathcal{C}$ as timed words:

- Successive control states of $\mathcal{C}$ are encoded by events with integer timestamps.

Undecidability for MTL

Claim. MTL is **undecidable** (over both finite and infinite words).

Proof: Let $\mathcal{C}$ be a channel machine. Encode the computations of $\mathcal{C}$ as timed words:

- Successive control states of $\mathcal{C}$ are encoded by events with integer timestamps.
- The channel contents of $\mathcal{C}$ is encoded as a sequence of events in the unit interval between successive integers.

Undecidability for MTL

Claim. MTL is **undecidable** (over both finite and infinite words).

Proof: Let $\mathcal{C}$ be a channel machine. Encode the computations of $\mathcal{C}$ as timed words:

- Successive control states of $\mathcal{C}$ are encoded by events with integer timestamps.
- The channel contents of $\mathcal{C}$ is encoded as a sequence of events in the unit interval between successive integers.
- Dense-time semantics allows us to encode unbounded channels.

Undecidability for MTL

Claim. MTL is **undecidable** (over both finite and infinite words).

Proof: Let $\mathcal{C}$ be a channel machine. Encode the computations of $\mathcal{C}$ as timed words:

- Successive control states of $\mathcal{C}$ are encoded by events with integer timestamps.
- The channel contents of $\mathcal{C}$ is encoded as a sequence of events in the unit interval between successive integers.
- Dense-time semantics allows us to encode unbounded channels.

Write down an MTL formula φ that captures precisely those timed words corresponding to halting computations of $\mathcal{C}$.

Then $\mathcal{C}$ halts iff φ is satisfiable.

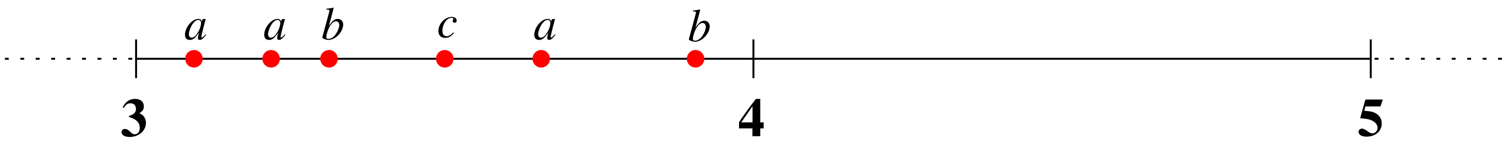
Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

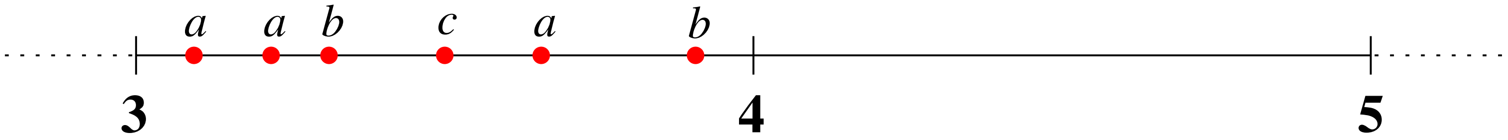
This is captured as:



Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

This is captured as:



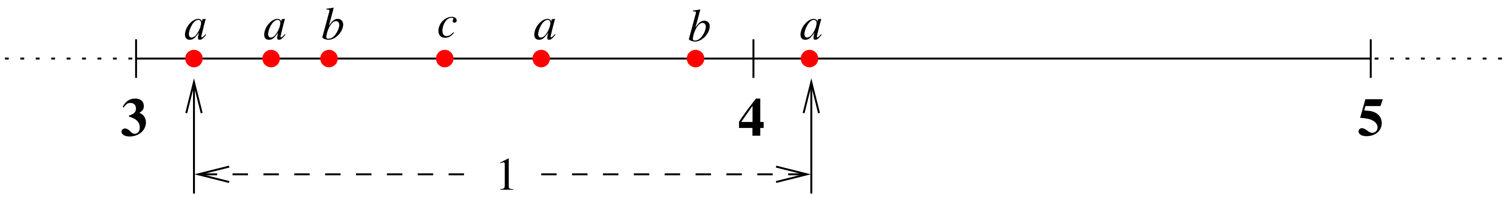
To correctly propagate the channel contents to the next time interval, we use the following formula:

$$\bigwedge_{x \in \Sigma(M)} \Box(x \longrightarrow \Diamond_{=1} x)$$

Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

This is captured as:



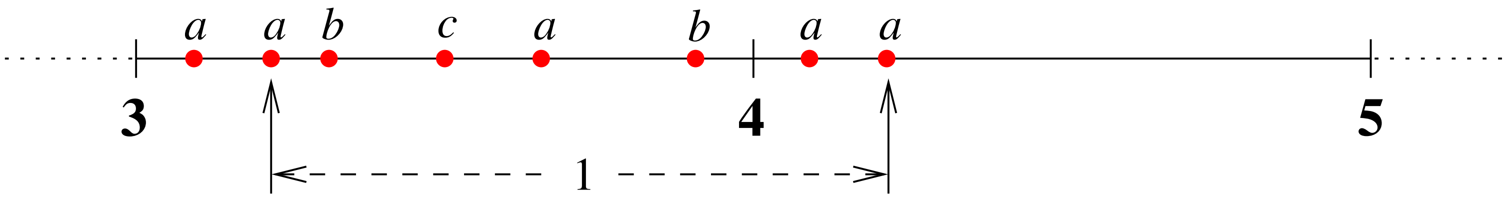
To correctly propagate the channel contents to the next time interval, we use the following formula:

$$\bigwedge_{x \in \Sigma(M)} \Box(x \longrightarrow \Diamond_{=1} x)$$

Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

This is captured as:



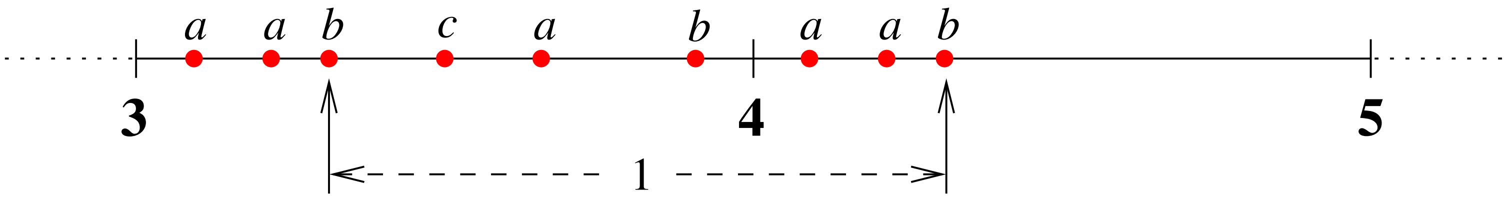
To correctly propagate the channel contents to the next time interval, we use the following formula:

$$\bigwedge_{x \in \Sigma(M)} \Box(x \longrightarrow \Diamond_{=1} x)$$

Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

This is captured as:



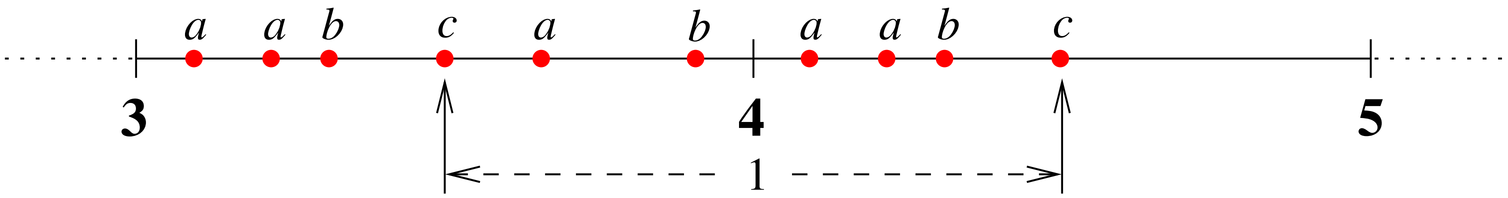
To correctly propagate the channel contents to the next time interval, we use the following formula:

$$\bigwedge_{x \in \Sigma(M)} \Box (x \longrightarrow \Diamond_{=1} x)$$

Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

This is captured as:



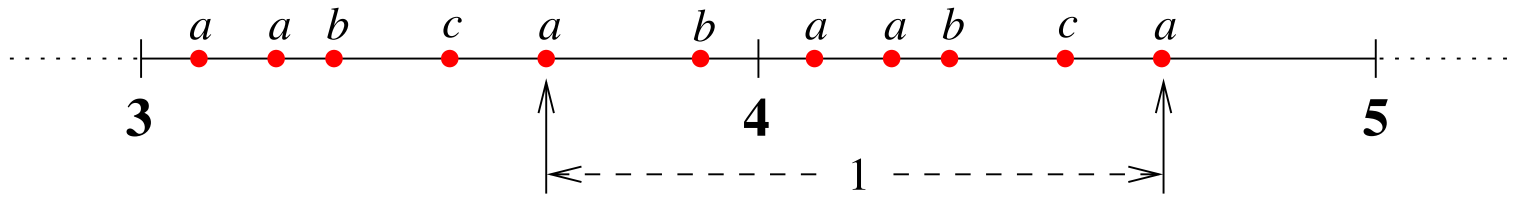
To correctly propagate the channel contents to the next time interval, we use the following formula:

$$\bigwedge_{x \in \Sigma(M)} \Box(x \longrightarrow \Diamond_{=1} x)$$

Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

This is captured as:



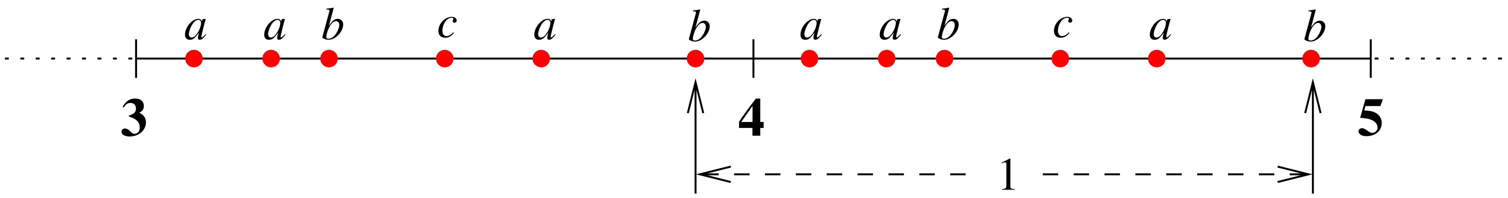
To correctly propagate the channel contents to the next time interval, we use the following formula:

$$\bigwedge_{x \in \Sigma(M)} \Box(x \longrightarrow \Diamond_{=1} x)$$

Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

This is captured as:



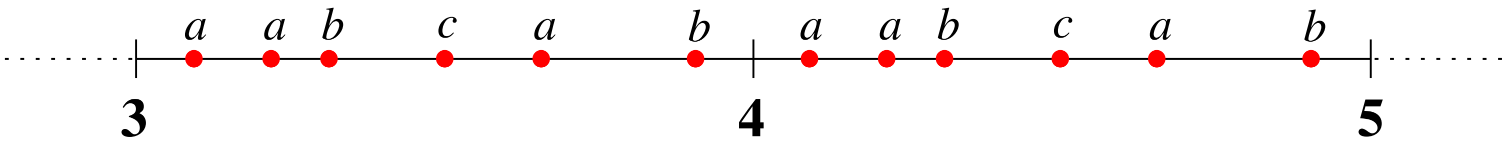
To correctly propagate the channel contents to the next time interval, we use the following formula:

$$\bigwedge_{x \in \Sigma(M)} \Box(x \longrightarrow \Diamond_{=1} x)$$

Propagating the Channel Contents

Suppose that at time 3, the current channel contents of $\mathcal{C}$ is $\langle aabcab \rangle$.

This is captured as:



To correctly propagate the channel contents to the next time interval, we use the following formula:

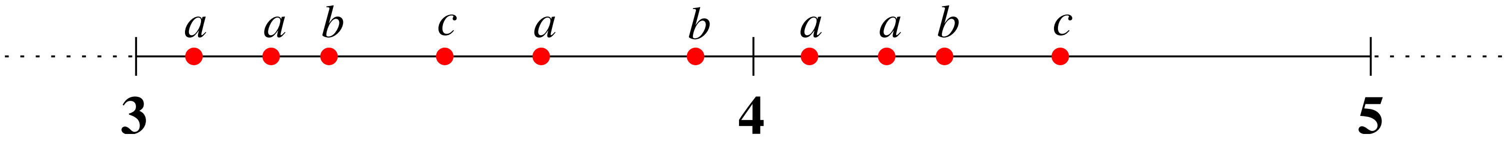
$$\bigwedge_{x \in \Sigma(M)} \Box(x \longrightarrow \Diamond_{=1} x)$$

Insertion Errors!

There is something missing — the formula

$$\bigwedge_{x \in \Sigma} \Box\left(x \longrightarrow \Diamond_{=1} x\right)$$

only enforces **forward** propagation of events:

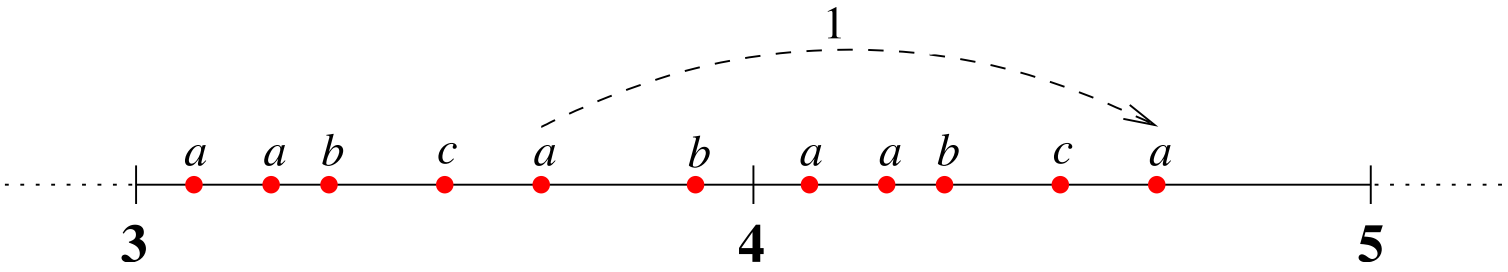


Insertion Errors!

There is something missing — the formula

$$\bigwedge_{x \in \Sigma} \Box (x \longrightarrow \Diamond_{=1} x)$$

only enforces **forward** propagation of events:

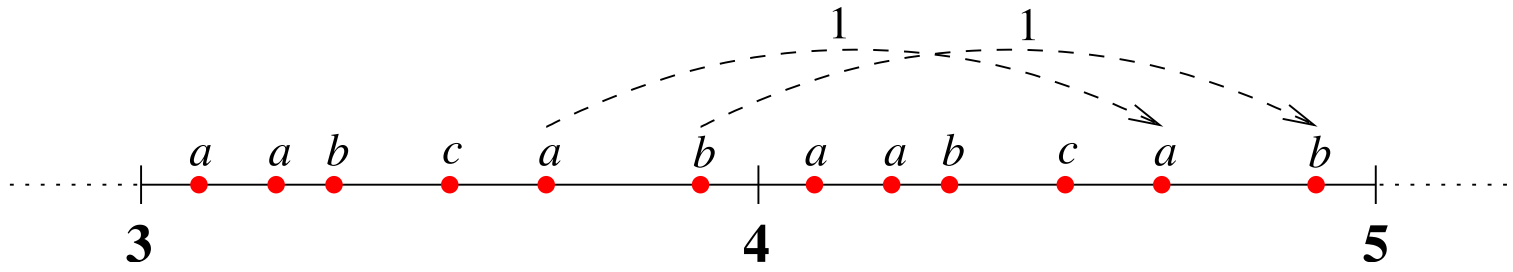


Insertion Errors!

There is something missing — the formula

$$\bigwedge_{x \in \Sigma} \Box(x \longrightarrow \Diamond_{=1} x)$$

only enforces **forward** propagation of events:

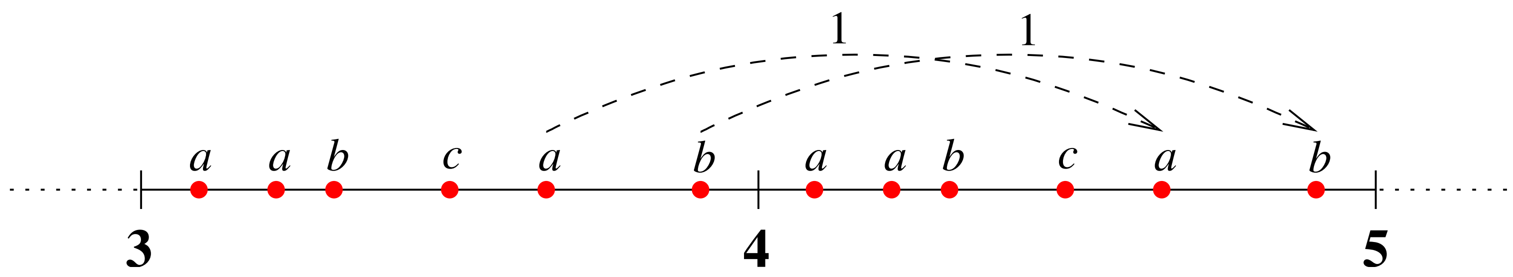


Insertion Errors!

There is something missing — the formula

$$\bigwedge_{x \in \Sigma} \Box(x \longrightarrow \Diamond_{=1} x)$$

only enforces **forward** propagation of events:



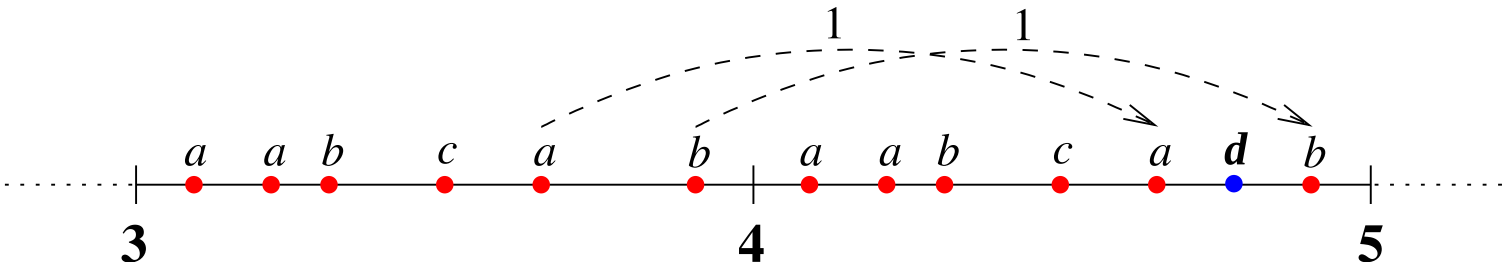
Nothing prevents, however, **insertion errors** from occurring.

Insertion Errors!

There is something missing — the formula

$$\bigwedge_{x \in \Sigma} \Box(x \longrightarrow \Diamond_{=1} x)$$

only enforces **forward** propagation of events:



Nothing prevents, however, **insertion errors** from occurring.

Channel Machines with Insertion Errors

We can only simulate channel machines with insertion errors in MTL.

Channel Machines with Insertion Errors

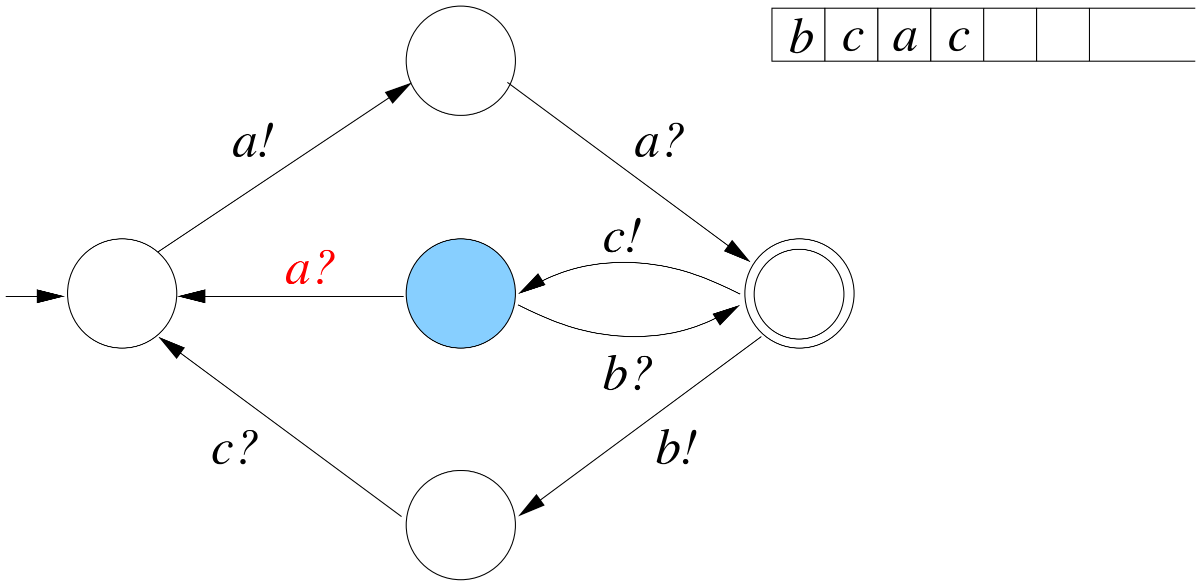
We can only simulate channel machines with insertion errors in MTL.

Read-transitions are always enabled.

Channel Machines with Insertion Errors

We can only simulate channel machines with insertion errors in MTL.

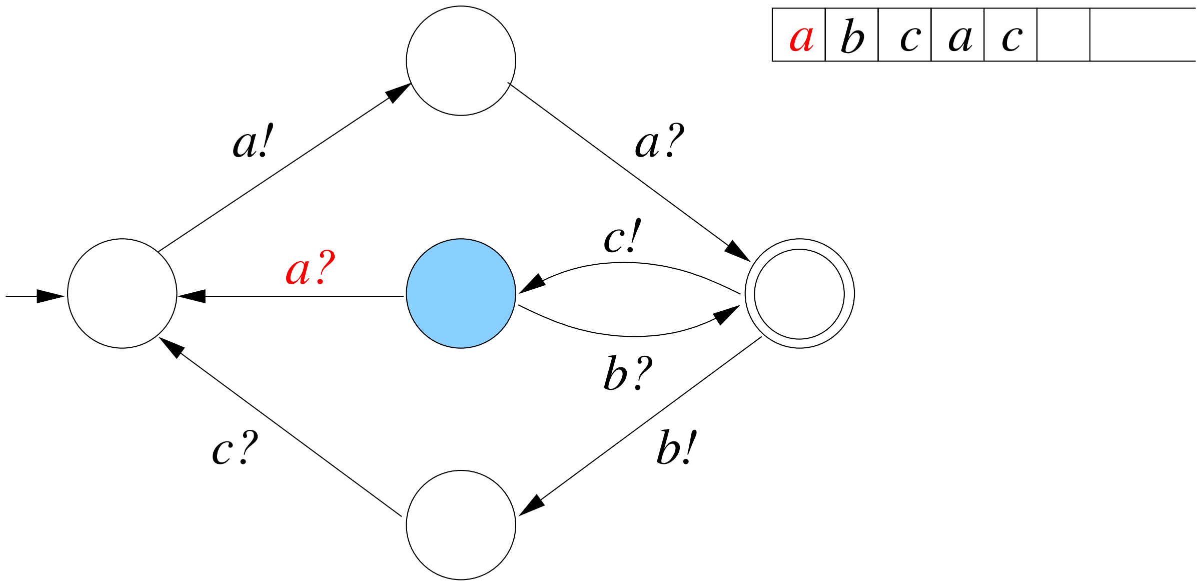
Read-transitions are always enabled.



Channel Machines with Insertion Errors

We can only simulate channel machines with insertion errors in MTL.

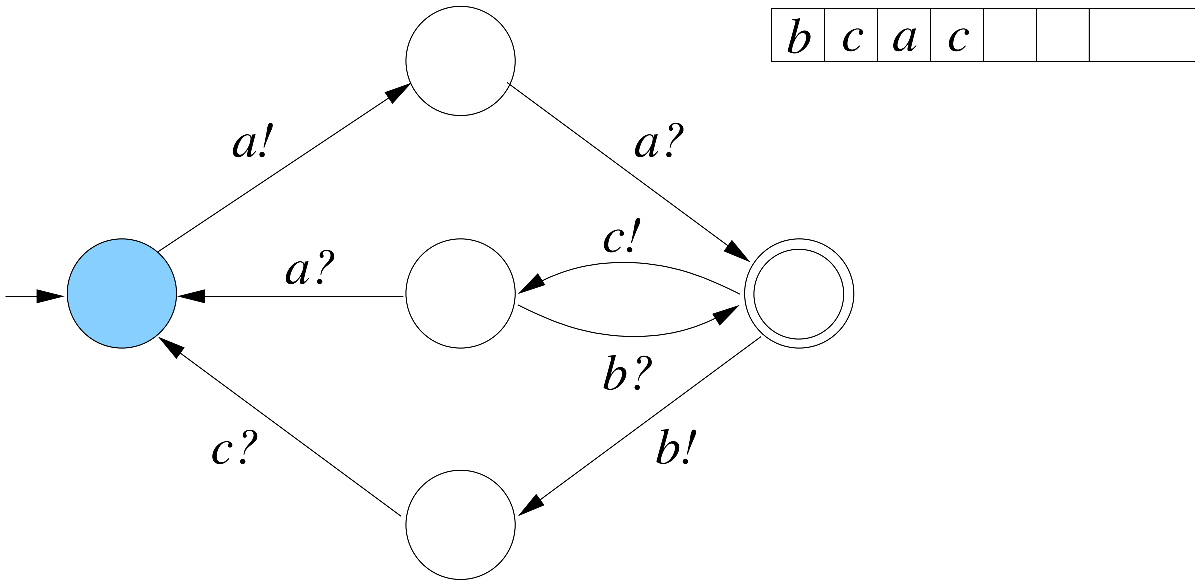
Read-transitions are always enabled.



Channel Machines with Insertion Errors

We can only simulate channel machines with insertion errors in MTL.

Read-transitions are always enabled.



Recurrent-state Problem

- Add emptiness testing via transitions $s \xrightarrow{\varepsilon?} t$

Recurrent-state Problem

- Add emptiness testing via transitions $s \xrightarrow{\varepsilon?} t$
- **Defn.** The **recurrent-state problem** asks if a given channel machine has an infinite computation that visits an accepting state infinitely often.

Recurrent-state Problem

- Add emptiness testing via transitions $s \xrightarrow{\varepsilon?} t$
- **Defn.** The **recurrent-state problem** asks if a given channel machine has an infinite computation that visits an accepting state infinitely often.
- **Thm.** The recurrent-state problem for channel machines with insertion errors and emptiness testing is undecidable.

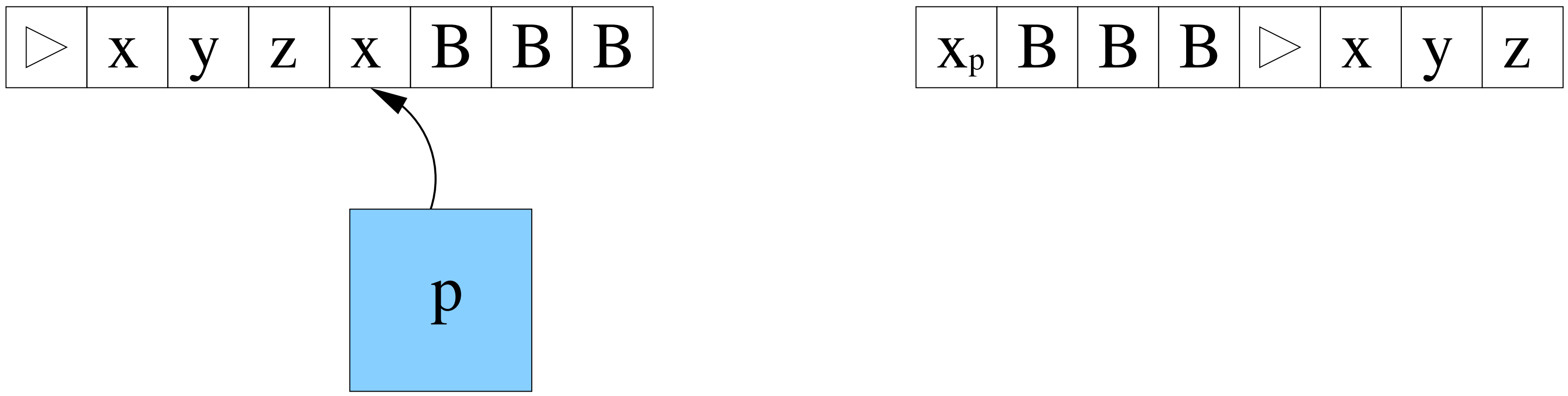
Recurrent-state Problem

- Add emptiness testing via transitions $s \xrightarrow{\varepsilon?} t$
- **Defn.** The **recurrent-state problem** asks if a given channel machine has an infinite computation that visits an accepting state infinitely often.
- **Thm.** The recurrent-state problem for channel machines with insertion errors and emptiness testing is undecidable.
  - Simulate Turing machine using faulty channel machine.

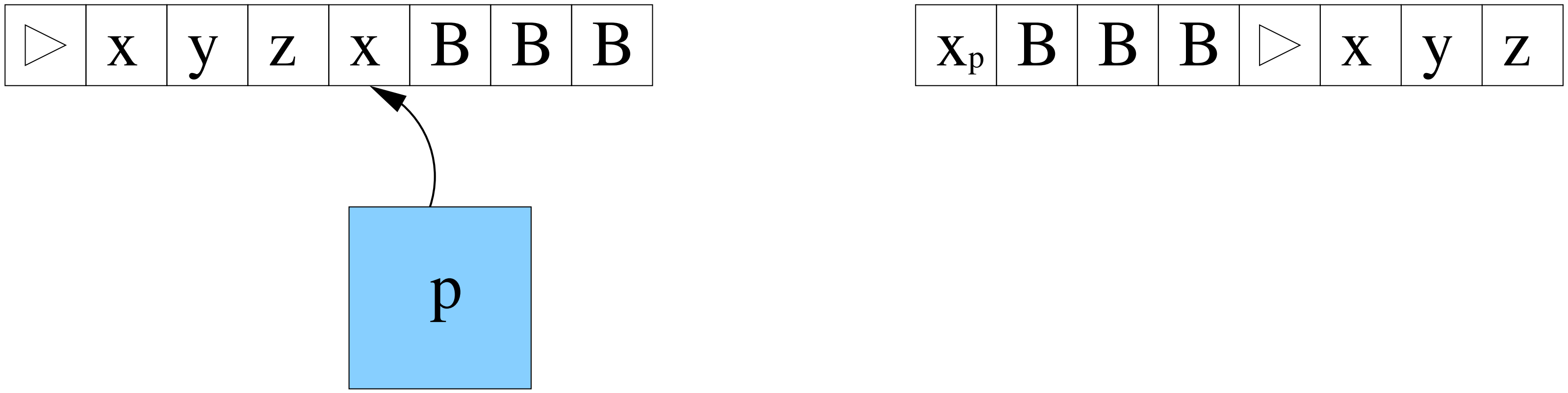
Simulating a Turing Machine

- Simulate a Turing machine $\mathcal{M}$ with a faulty channel machine $\mathcal{C}$.
- $\mathcal{C}$ has two channels c and d.
  - Store configuration of $\mathcal{M}$ on c.
  - Use d as a counter.
  - Both channels are subject to insertion errors.

Simulation Phase

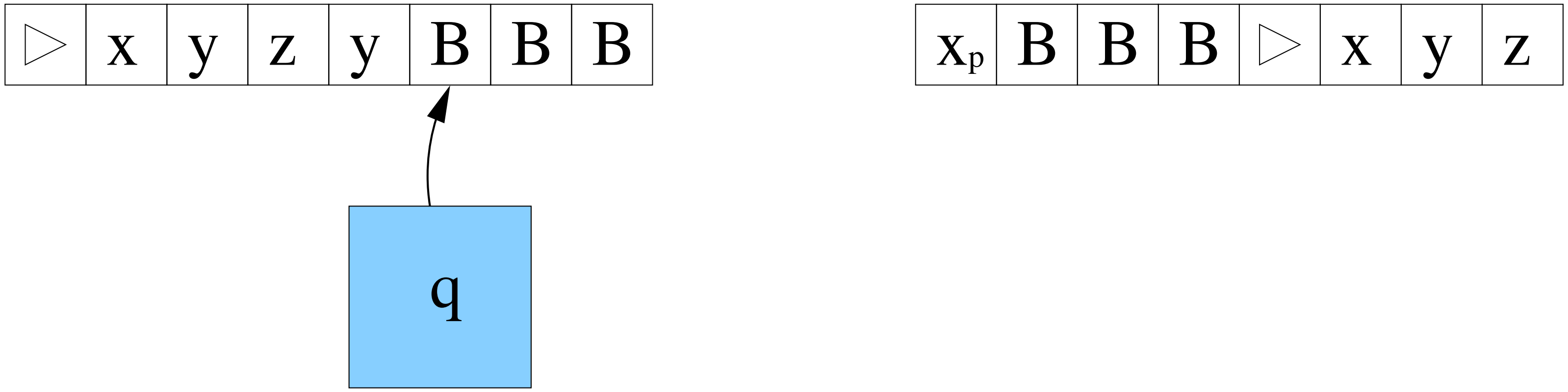


Simulation Phase



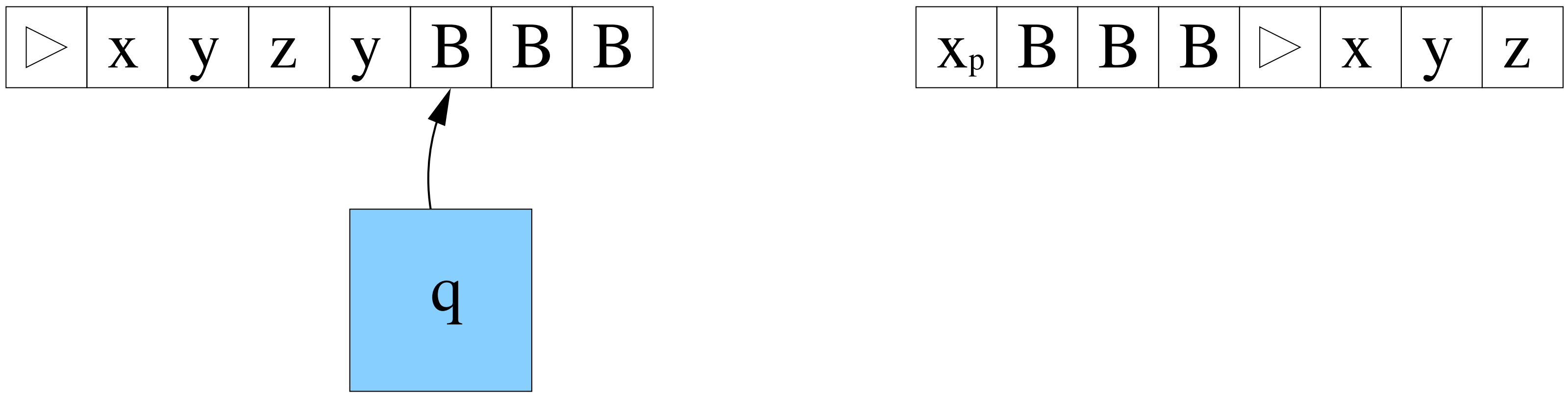
- Simulate transition $(p, x, q, y, \rightarrow)$ of $\mathcal{M}$.

Simulation Phase



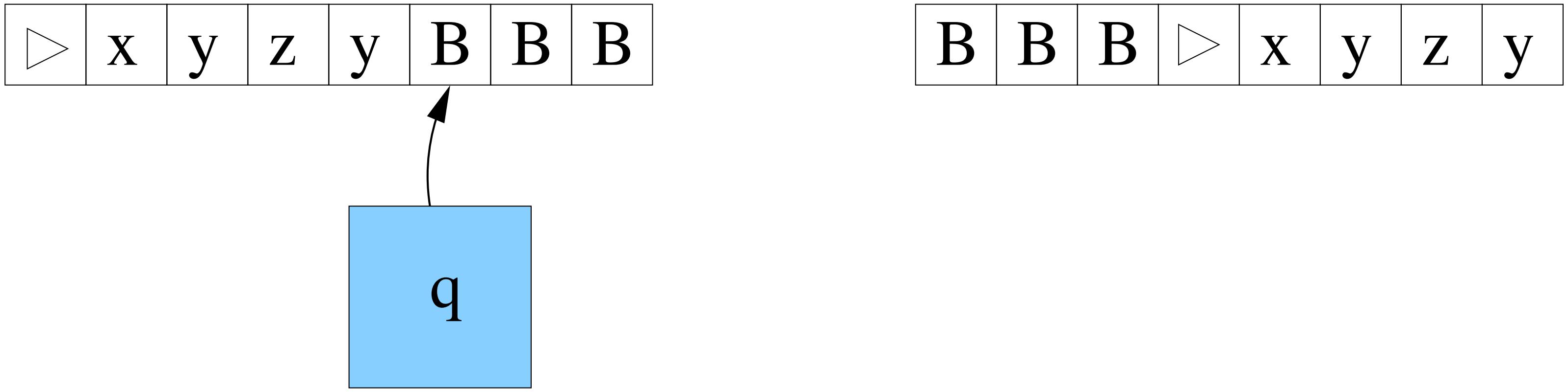
Simulate transition $(p, x, q, y, \rightarrow)$ of $\mathcal{M}$.

Simulation Phase



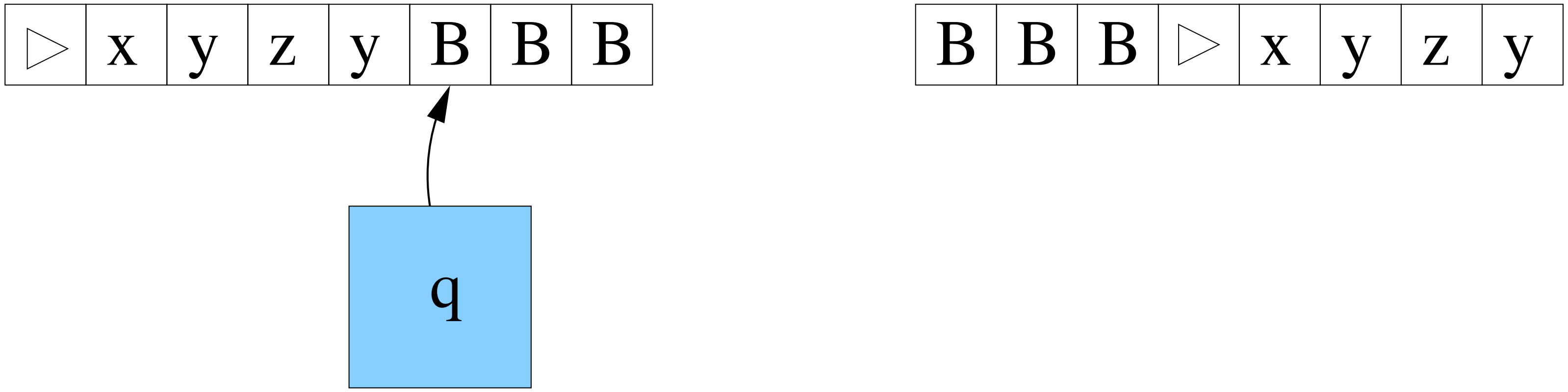
- Simulate transition $(p, x, q, y, \rightarrow)$ of $\mathcal{M}$.
  - Read x_q $\Rightarrow$ Write y

Simulation Phase



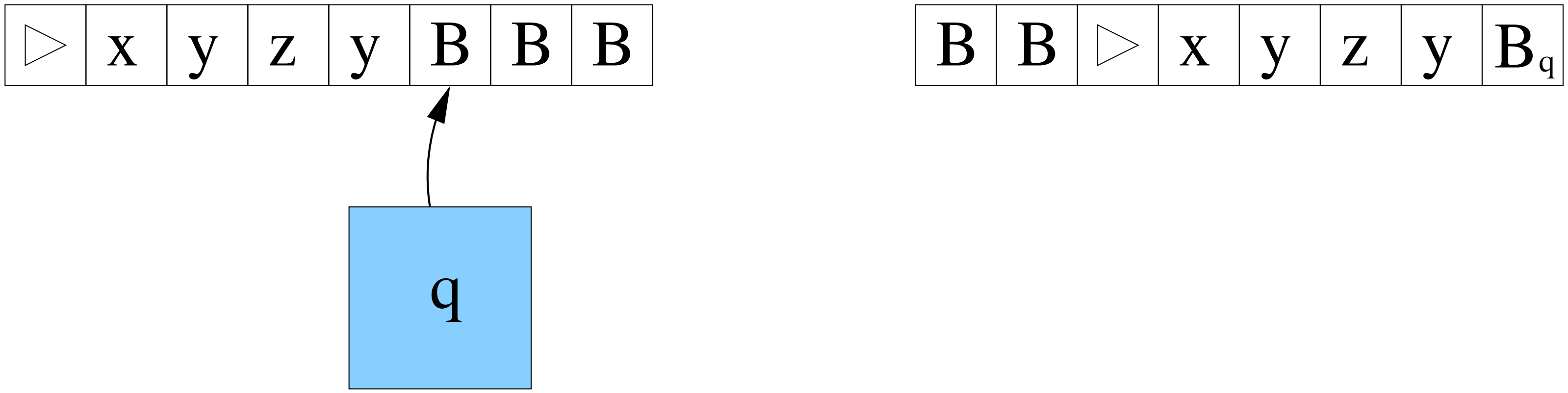
- Simulate transition $(p, x, q, y, \rightarrow)$ of $\mathcal{M}$.
  - Read x_q $\Rightarrow$ Write y

Simulation Phase



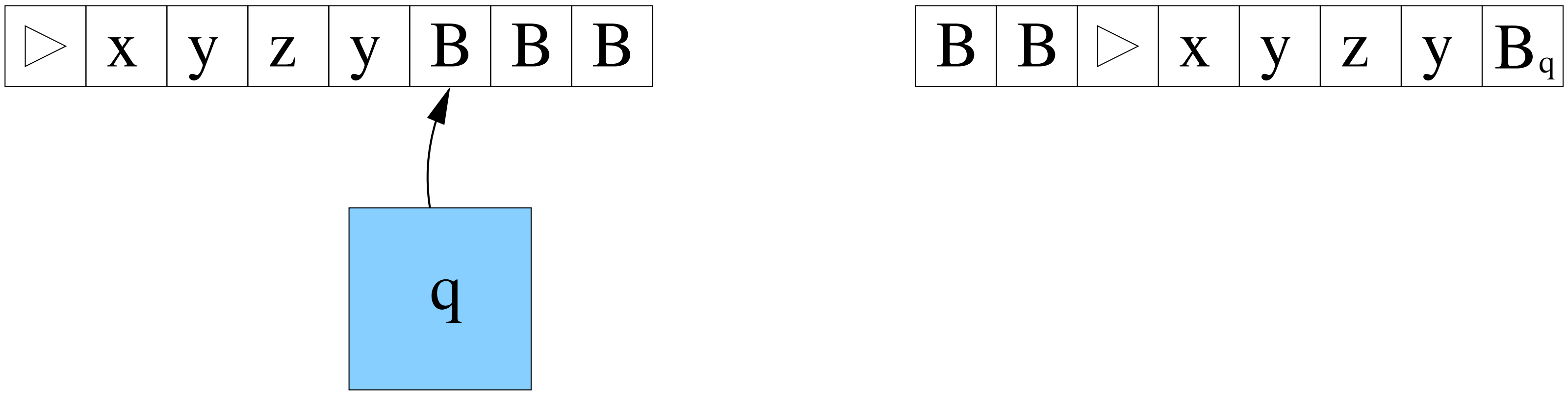
- Simulate transition $(p, x, q, y, \rightarrow)$ of $\mathcal{M}$.
  - Read $x_q \;\Rightarrow\;$ Write $y \;\Rightarrow\;$ Read $B \;\Rightarrow\;$ Write B_q.

Simulation Phase



- Simulate transition $(p, x, q, y, \rightarrow)$ of $\mathcal{M}$.
  - Read $x_q \;\Rightarrow\;$ Write $y \;\Rightarrow\;$ Read $B \;\Rightarrow\;$ Write B_q.

Simulation Phase

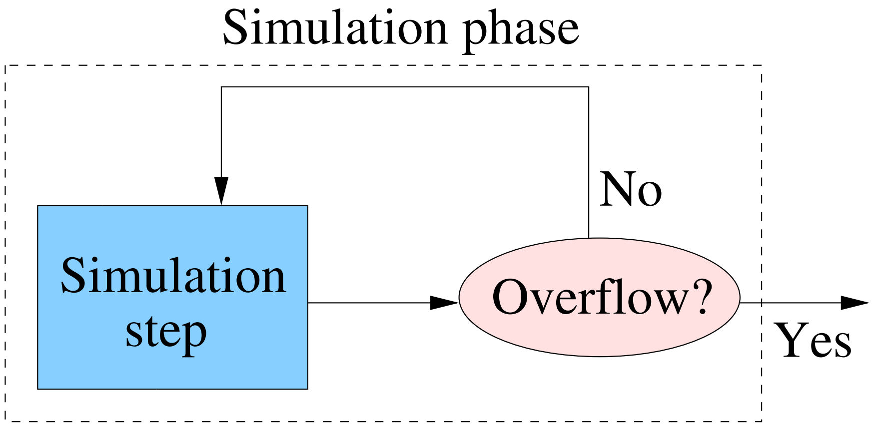


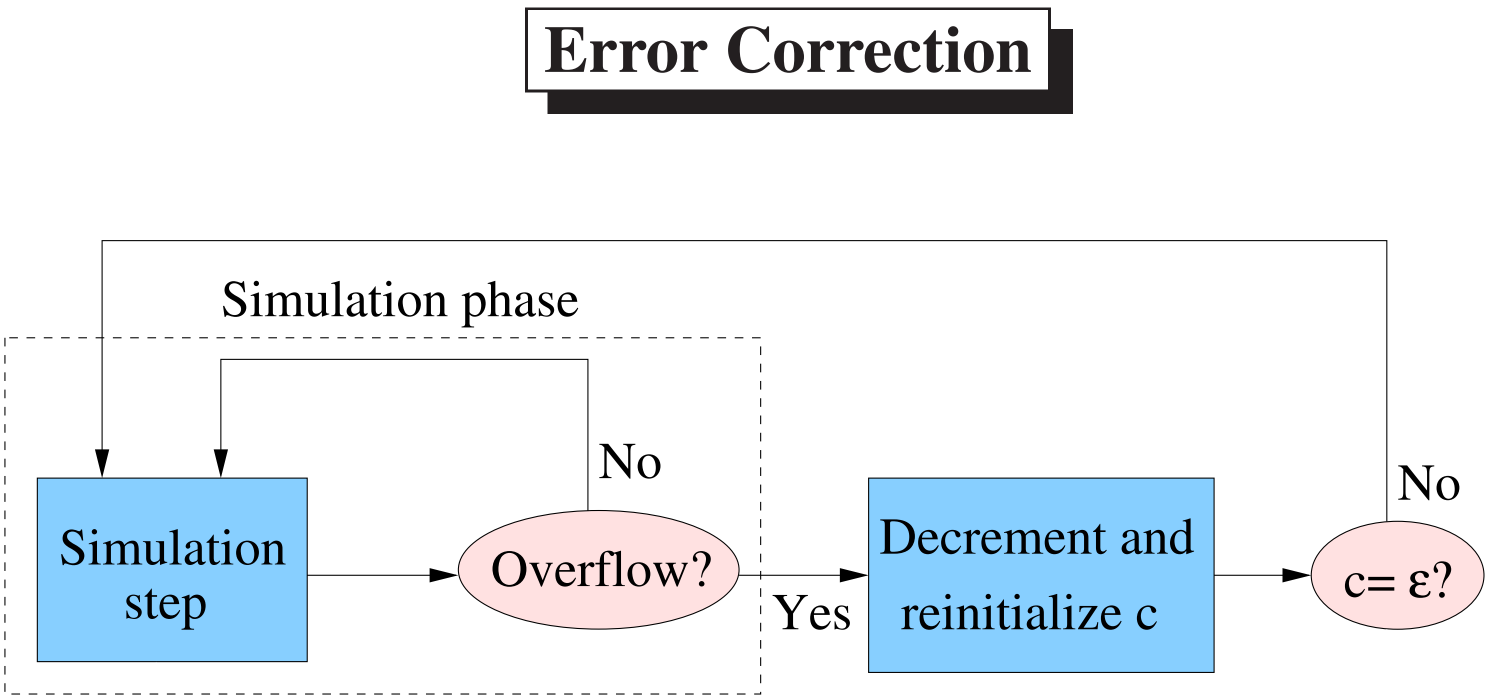
- Simulate transition $(p, x, q, y, \rightarrow)$ of $\mathcal{M}$.
  - Read x_q $\Rightarrow$ Write y $\Rightarrow$ Read B $\Rightarrow$ Write B_q.
- Notice the alternation between read and write events.

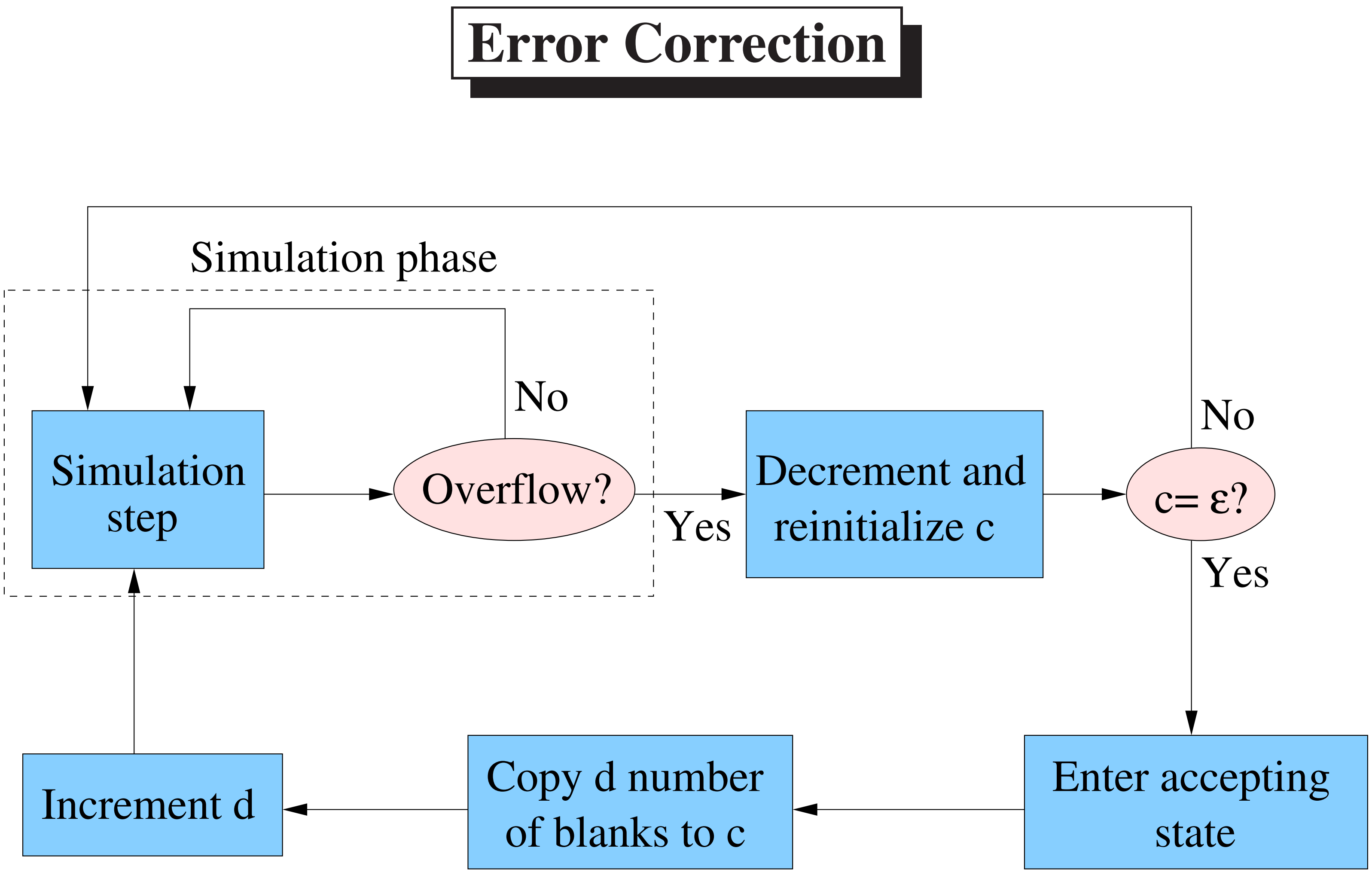
Simulation Phase

- Two problems:
  - Could run out of space during simulation phase.
  - Could also be computing gibberish!

Error Correction




Error Correction
Simulation phase
No
Simulation step
Overflow?
Yes
Decrement and reinitialize c
No
c= ε?



Error Correction
Simulation phase
Simulation step
Overflow?
No
Yes
Decrement and reinitialize c
c= ε?
No
Yes
Enter accepting state
Copy d number of blanks to c
Increment d

RSP is undecidable

Prop. $\mathcal{C}$ has a recurrent computation iff $\mathcal{M}$ **does not** halt.

Proof. ($\Longrightarrow$)

Suppose $\mathcal{M}$ has a halting computation using N tape cells. Once $|c| > N$, then $\mathcal{C}$ can never enter the accepting state again.

Theorem. The recurrent state problem for channel machines with insertion errors is undecidable.

MTL is Undecidable

- **Theorem.** MTL is undecidable.
  - **Proof.** By reduction from **recurrent-state problem** for channel machines with **insertion errors**.
  - The reduction involves the **recurrence** formula $\Box\Diamond p$ (p holds infinitely often).
  - Contrast with the result that MTL is decidable over finite timed words (OW05).

MTL is undecidable

- The decidability of a number of real-time logics over timed words was considered in *Real-time Logics: Complexity and Expressiveness* [Alur and Henzinger 93].
- The undecidability proof of Alur and Henzinger relied on encoding computations of **perfect** Turing machines as timed words. This proof only applies to richer formalisms than MTL (e.g., first-order monadic logic of timed words, or MTL with past connectives).
- Nonetheless it has become folklore that MTL is undecidable.

Summary

- We considered specifying **metric** properties of real-time systems under a **non-Z eno** semantics.
- **Thm.** Metric Temporal Logic is undecidable.
- Decidability of MTL differs over finite and infinite timed words.
- Undecidability of MTL in the flow semantics is much more straightforward.