Metric Temporal Logic and Faulty Turing Machines

James Worrell
Dept. of Mathematics, Tulane University, USA

Joël Ouaknine
Oxford University Computing Laboratory, UK

AVOCS 2005

Automated Verification

- Classical theory:
  - Finite automata, temporal logics, model checking, . . .
  - *Qualitative* model of time (ordered sequences of states/actions).
  - Theoretical foundations well-understood.

Automated Verification

- Classical theory:
  - Finite automata, temporal logics, model checking, ...
  - *Qualitative* model of time (ordered sequences of states/actions).
  - Theoretical foundations well-understood.

Trakhtenbrot [LICS 95]: "**Lift classical theory to real-time world.**"

Automated Verification

- Classical theory:
  - Finite automata, temporal logics, model checking, ...
  - *Qualitative* model of time (ordered sequences of states/actions).
  - Theoretical foundations well-understood.

Trakhtenbrot [LICS 95]: "**Lift classical theory to real-time world.**"

- Real-time theory:
  - Timed automata, quantitative temporal logic, model checking, ...
  - *Quantitative* and dense model of time.
  - Theoretical foundations under active development.

Timed and Untimed Specifications

- A standard (untimed) response property:
  - "Every request is eventually granted."

$$\Box(request \longrightarrow \Diamond grant)$$

Timed and Untimed Specifications

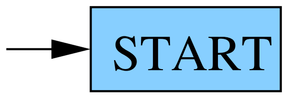
- A standard (untimed) response property:
  - "Every request is eventually granted."

$$\Box(\mathit{request} \longrightarrow \Diamond \mathit{grant})$$

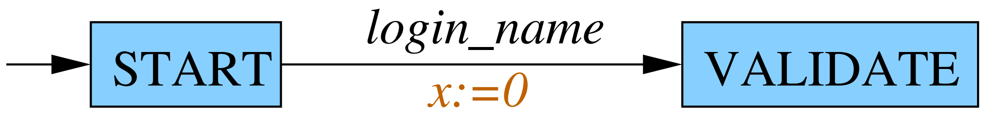
- A *time-bounded* response property:
  - "Every request is eventually granted *within 5 time units*."

$$\Box(\mathit{request} \longrightarrow \Diamond_{[0,5]} \mathit{grant})$$

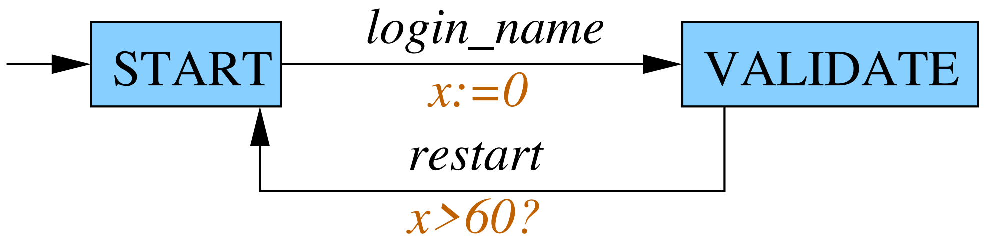
Example: A Login Protocol



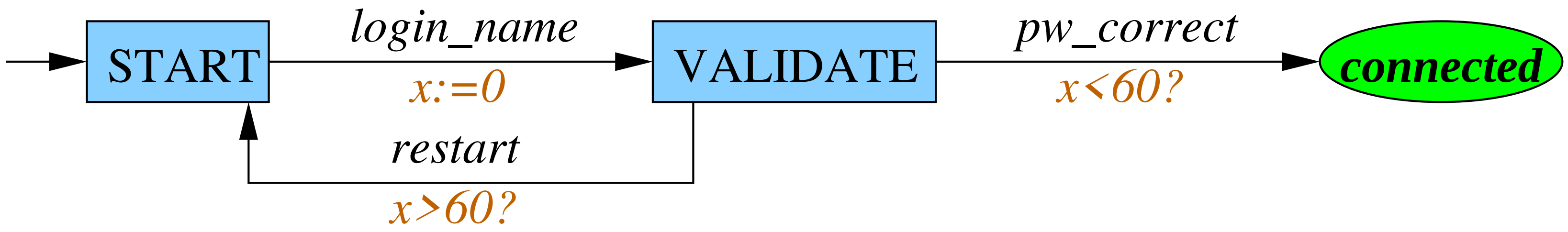
Example: A Login Protocol



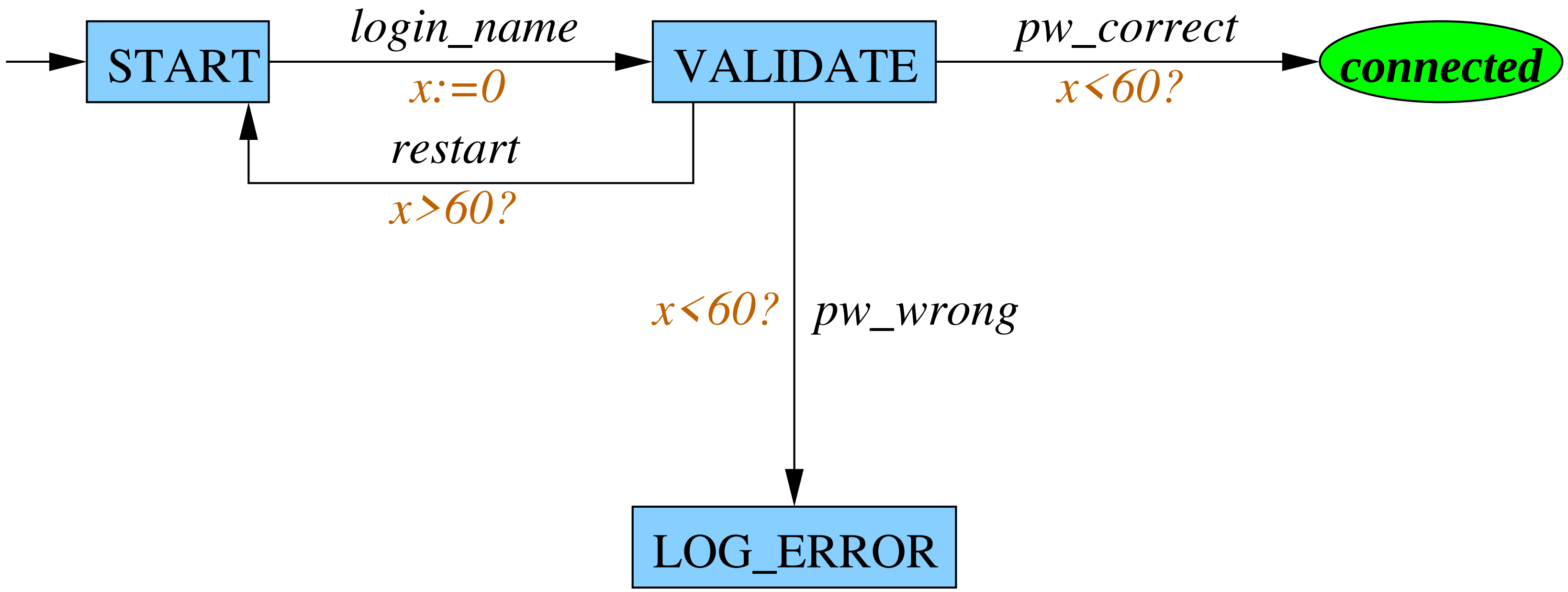
Example: A Login Protocol



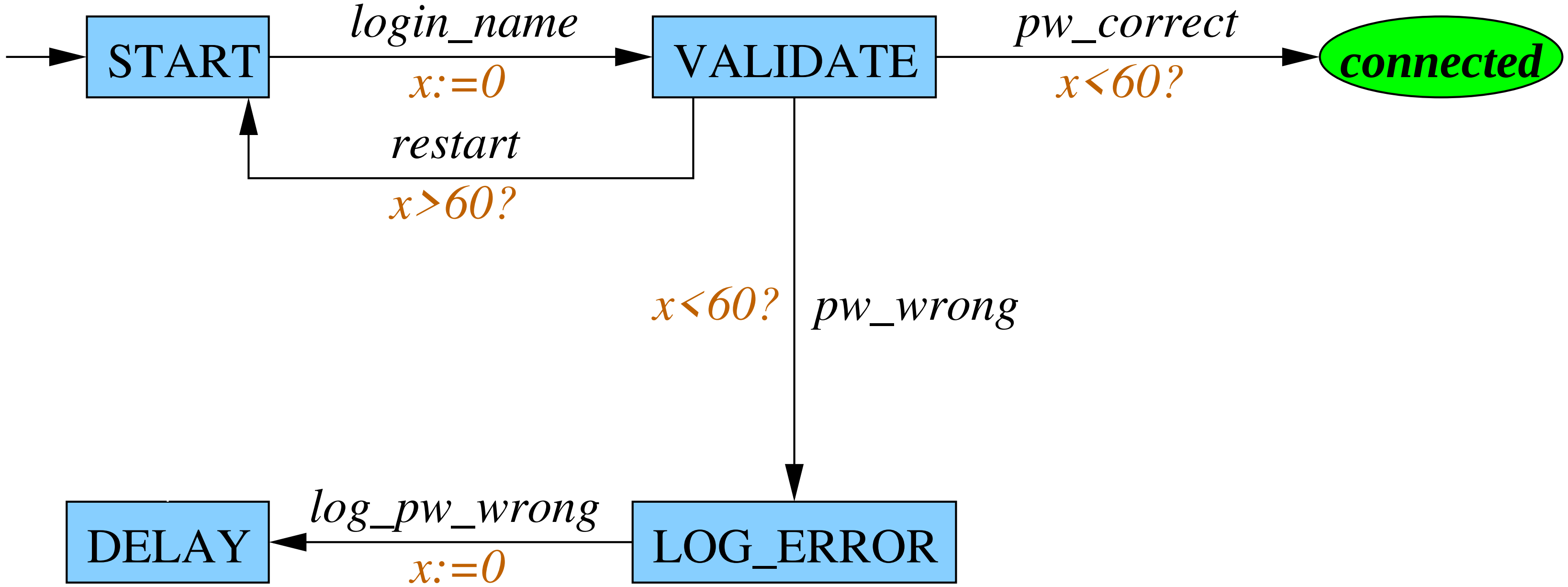
Example: A Login Protocol



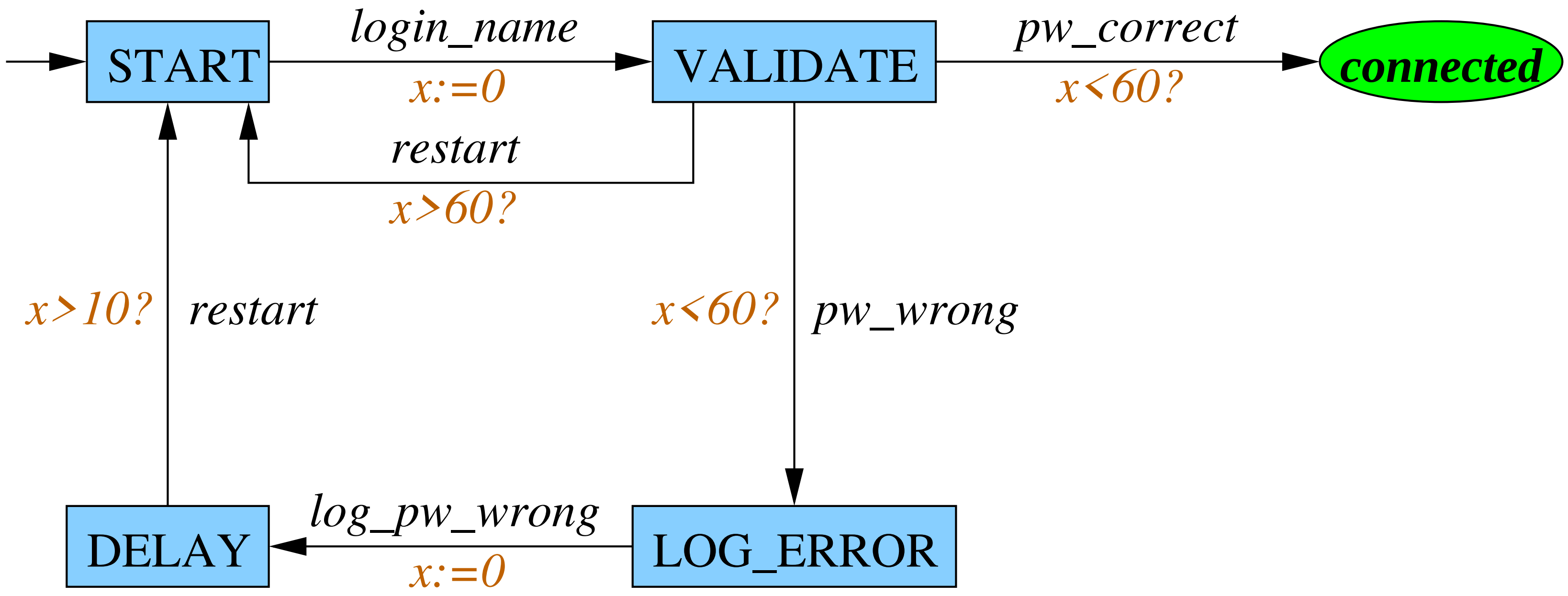
Example: A Login Protocol



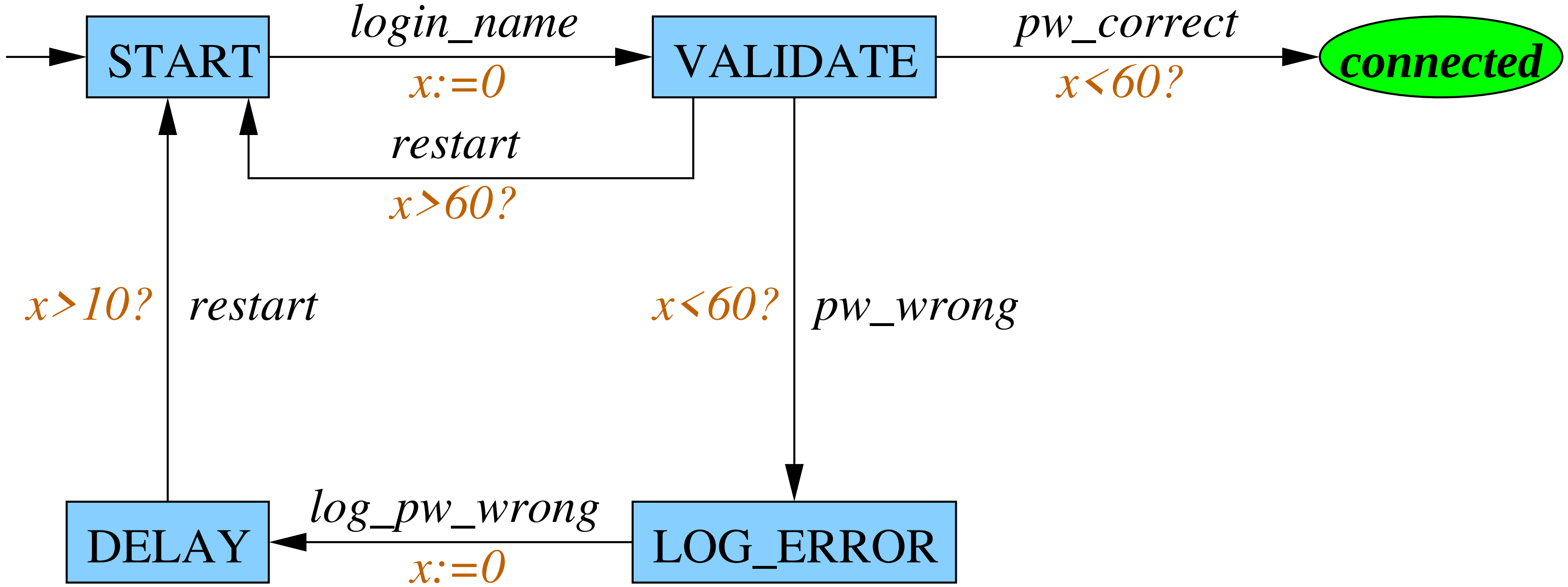
Example: A Login Protocol



Example: A Login Protocol



Example: A Login Protocol



SPECIFICATION: $\square(pw\_wrong \longrightarrow \square_{[0,10]} \neg restart)$

Metric Temporal Logic (MTL)

- To reason precisely about time, *dense-time* temporal logics were proposed by several researchers in the late 80s: de Roever, Emerson, Koymans, Pnueli, Sistla, . . .

Metric Temporal Logic (MTL)

- To reason precisely about time, *dense-time* temporal logics were proposed by several researchers in the late 80s: de Roever, Emerson, Koymans, Pnueli, Sistla, ...

- **Metric Temporal Logic (MTL)** is a prominent such dense-time specification formalism.
  - Over two hundred and fifty papers according to `scholar.google.com` !

Metric Temporal Logic (MTL)

- MTL = LTL + timing constraints on operators:
  - $\Diamond_{[3,5]} a$
  - $\Box(a \rightarrow \Diamond_{=1} b)$

Metric Temporal Logic (MTL)

- MTL = LTL + timing constraints on operators:
  - $\Diamond_{[3,5]} a$
  - $\Box(a \rightarrow \Diamond_{=1} b)$
- $\varphi ::= \mathit{TRUE} \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid a \mid \varphi_1 \, \mathcal{U}_I \, \varphi_2 \mid \bigcirc_I \, \varphi$

Metric Temporal Logic (MTL)

- MTL = LTL + timing constraints on operators:
  - $\Diamond_{[3,5]} a$
  - $\Box(a \to \Diamond_{=1} b)$
- $\varphi ::= \mathit{TRUE} \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid a \mid \varphi_1 \, \mathcal{U}_I \, \varphi_2 \mid \bigcirc_I \varphi$

  where a is atomic and $I \subseteq \mathbb{R}$ is an interval with integer (or unbounded) endpoints.

Metric Temporal Logic (MTL)

- MTL = LTL + timing constraints on operators:
  - $\Diamond_{[3,5]} a$
  - $\Box(a \to \Diamond_{=1} b)$
- $\varphi ::= \mathit{TRUE} \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid a \mid \varphi_1 \, \mathcal{U}_I \, \varphi_2 \mid \bigcirc_I \varphi$

  where a is atomic and $I \subseteq \mathbb{R}$ is an interval with integer (or unbounded) endpoints.
- Modal operators $\Box_I$ and $\Diamond_I$ are derived from $\mathcal{U}_I$ in the usual way.

Timed Words

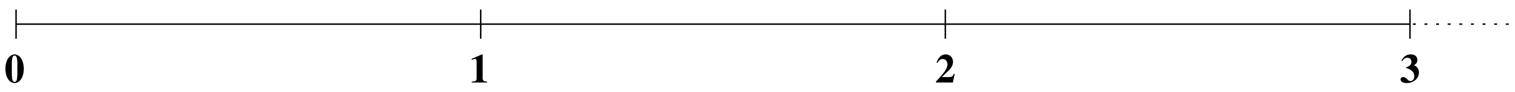
- A **timed word** is a finite or infinite sequence of **timed events**:

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

Timed Words

- A **timed word** is a finite or infinite sequence of **timed events**:

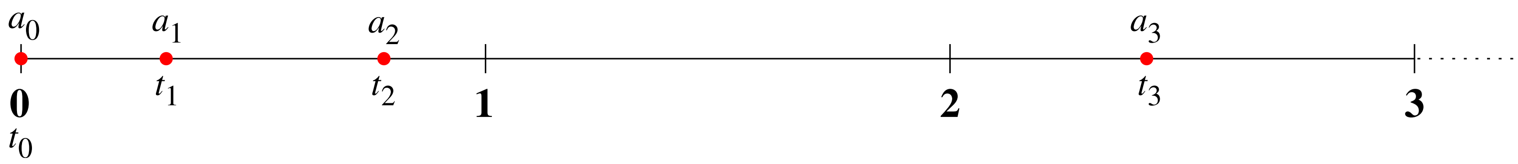
$$\langle (t_0, a_0), (t_1, a_1), (t_2, a_2), (t_3, a_3), \ldots \rangle$$



Timed Words

- A **timed word** is a finite or infinite sequence of **timed events**:

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



Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

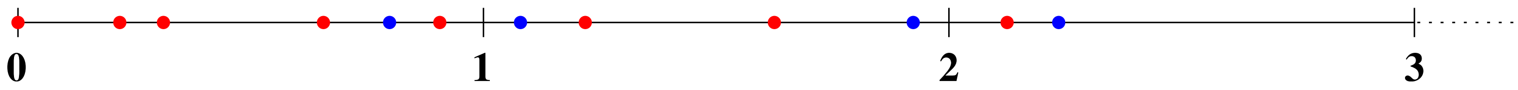
$$\Box(\mathit{request} \longrightarrow \Diamond_{[0,1]}\mathit{grant})$$

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\mathit{request} \longrightarrow \Diamond_{[0,1]}\mathit{grant})$$

  - Let w be:

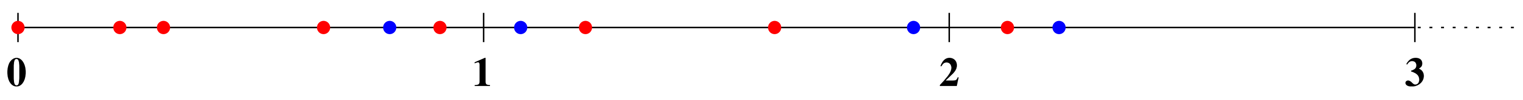


Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\textit{request} \longrightarrow \Diamond_{[0,1]}\textit{grant})$$

  - Let w be:



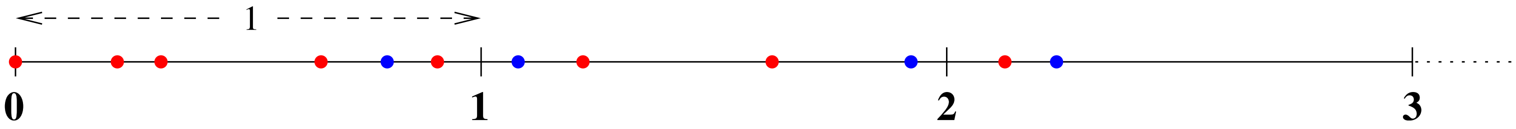
Does $w \models \varphi$?

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\textcolor{red}{request} \longrightarrow \Diamond_{[0,1]}\textcolor{blue}{grant})$$

  - Let w be:



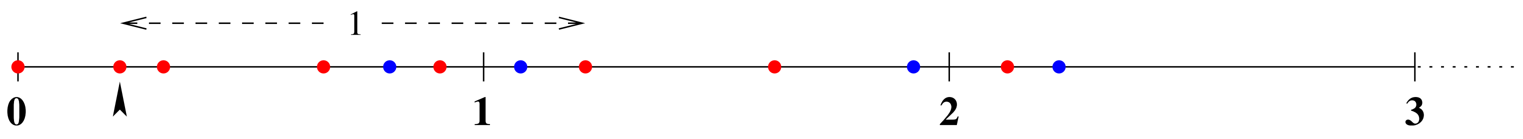
Does $w \models \varphi$?

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\textit{request} \longrightarrow \Diamond_{[0,1]}\textit{grant})$$

  - Let w be:



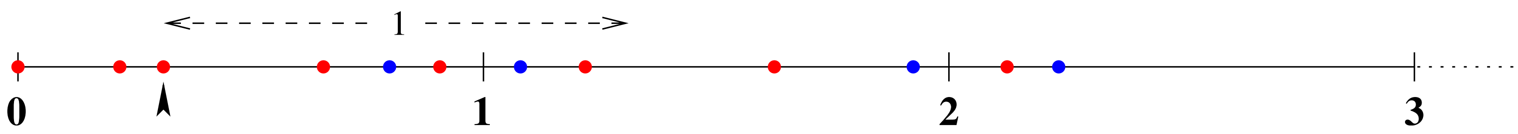
Does $w \models \varphi$?

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\textcolor{red}{request} \longrightarrow \Diamond_{[0,1]}\textcolor{blue}{grant})$$

  - Let w be:



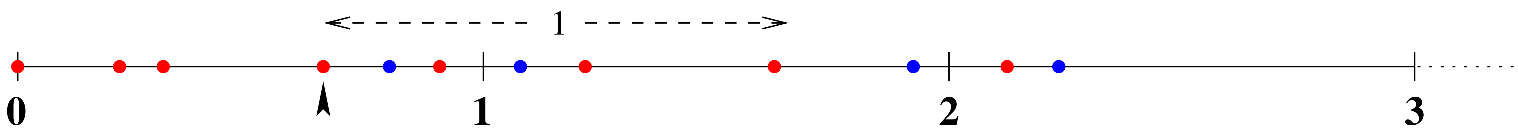
Does $w \models \varphi$?

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\textit{request} \longrightarrow \Diamond_{[0,1]}\textit{grant})$$

  - Let w be:



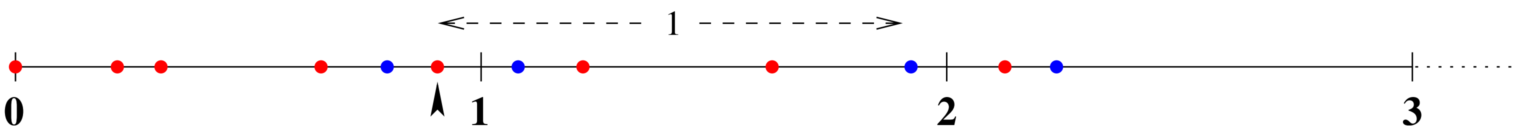
Does $w \models \varphi$?

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\textcolor{red}{request} \longrightarrow \Diamond_{[0,1]}\textcolor{blue}{grant})$$

  - Let w be:



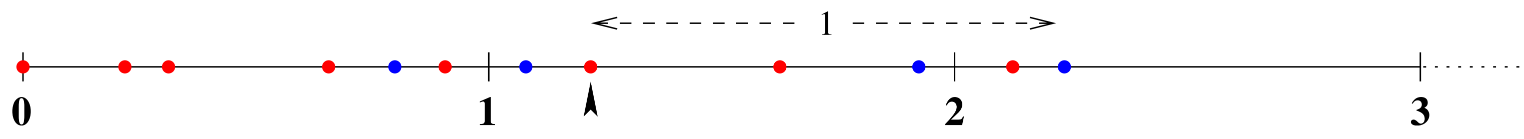
Does $w \models \varphi$?

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\mathit{request} \longrightarrow \Diamond_{[0,1]}\mathit{grant})$$

  - Let w be:



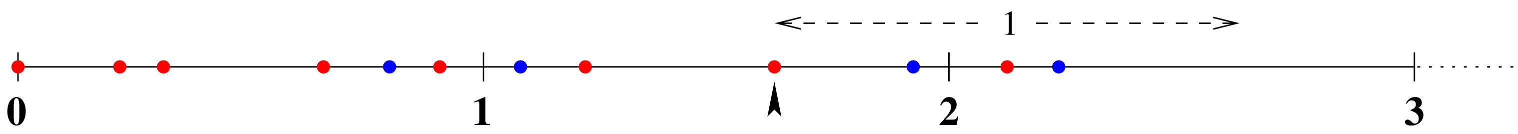
Does $w \models \varphi$?

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\mathit{request} \longrightarrow \Diamond_{[0,1]}\,\mathit{grant})$$

  - Let w be:



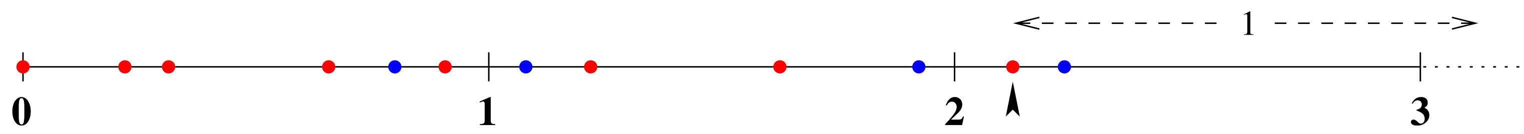
Does $w \models \varphi$?

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\mathit{request} \longrightarrow \Diamond_{[0,1]}\mathit{grant})$$

  - Let w be:



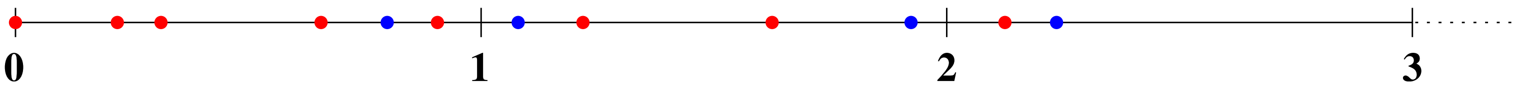
Does $w \models \varphi$?

Point-based Semantics for MTL

- The relationship $w \models \varphi$ is defined inductively as in LTL.
  - For example, let φ be:

$$\Box(\textcolor{red}{request} \longrightarrow \Diamond_{[0,1]} \textcolor{blue}{grant})$$

  - Let w be:



Indeed, $w \models \varphi$.

Point-based vs. Interval Semantics

- In the point-based semantics it is most natural to interpret atoms as events rather than states.

Point-based vs. Interval Semantics

- In the point-based semantics it is most natural to interpret atoms as events rather than states.
- The temporal quantifiers range over a countable set of points (where the system changes its discrete state).

Point-based vs. Interval Semantics

- In the point-based semantics it is most natural to interpret atoms as events rather than states.
- The temporal quantifiers range over a countable set of points (where the system changes its discrete state).
- In contrast, the *interval (or superdense) semantics* records the state of a system at every instant in time. This semantics differs in important respects from the point-based one and our results do not apply to it.

Point-based vs. Interval Semantics

- In the point-based semantics it is most natural to interpret atoms as events rather than states.
- The temporal quantifiers range over a countable set of points (where the system changes its discrete state).
- In contrast, the *interval (or superdense) semantics* records the state of a system at every instant in time. This semantics differs in important respects from the point-based one and our results do not apply to it.
- The dichotomy ‘point-based vs. interval semantics’ can be read as ‘event-based vs. state-based semantics’.

Satisfiability and Model Checking

- **Satisfiability:** Given φ, does there exist a timed word w such that $w \models \varphi$?
  - We will consider satisfiability both by *finite* and *infinite* words.

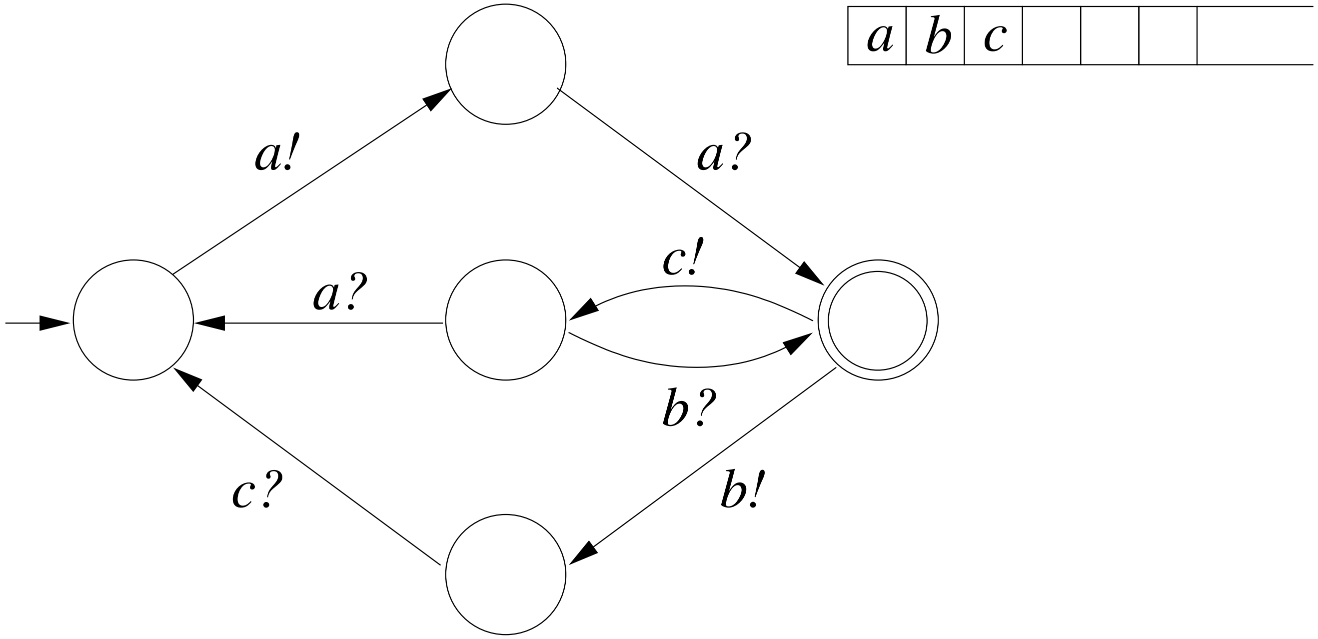
Satisfiability and Model Checking

- **Satisfiability:** Given φ, does there exist a timed word w such that $w \models \varphi$?
  - We will consider satisfiability both by *finite* and *infinite* words.

- **Model checking:** Given φ and a timed automaton A, does $w \models \varphi$ for all $w \in L(A)$?
  - For simplicity we mainly focus on satisfiability in this presentation.

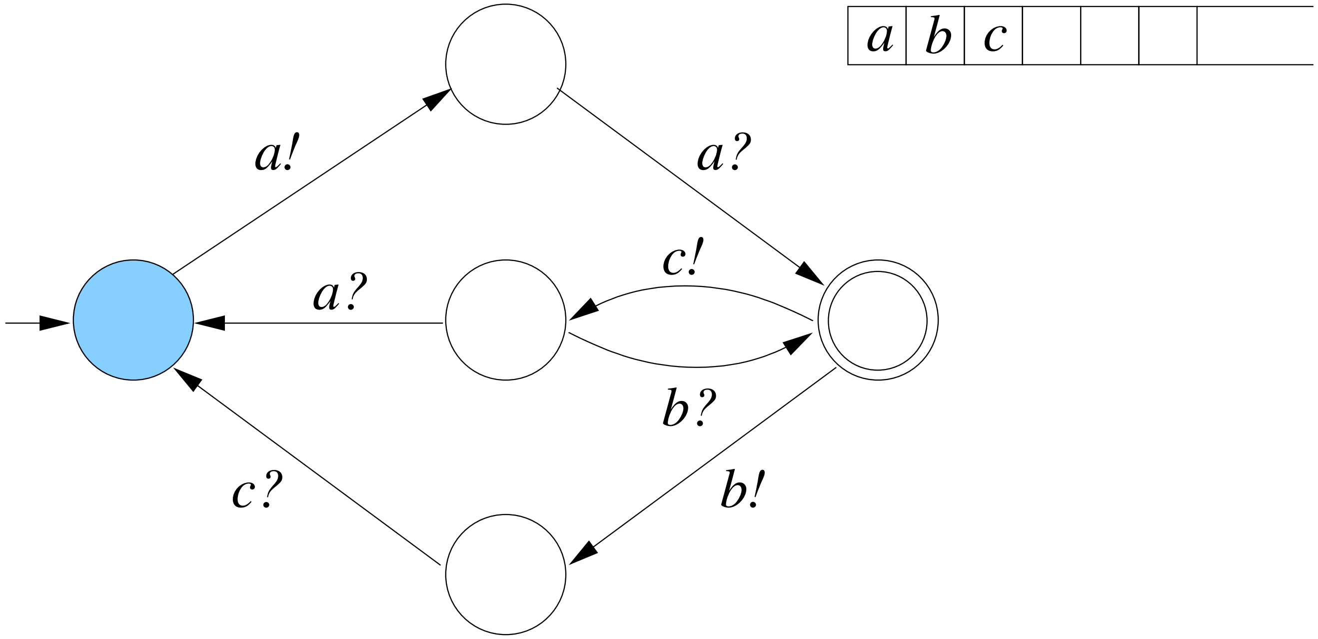
Interlude: Channel Machines

Our undecidability results use **channel machines**.



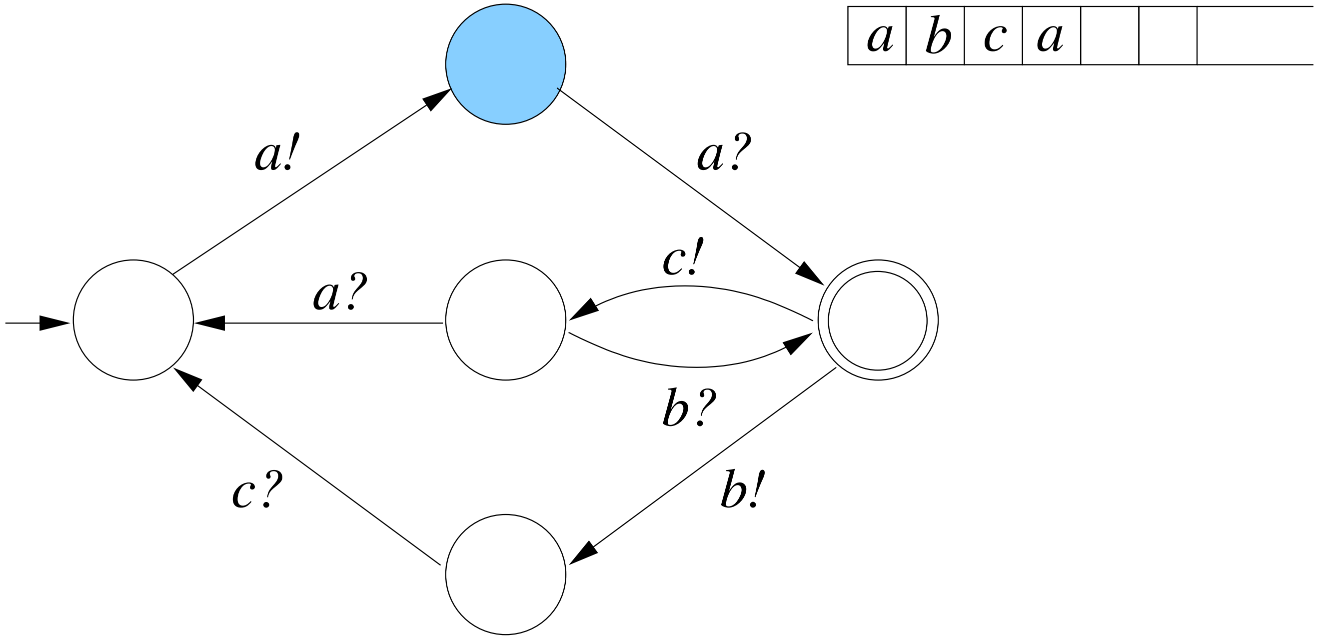
Interlude: Channel Machines

Our undecidability results use **channel machines**.



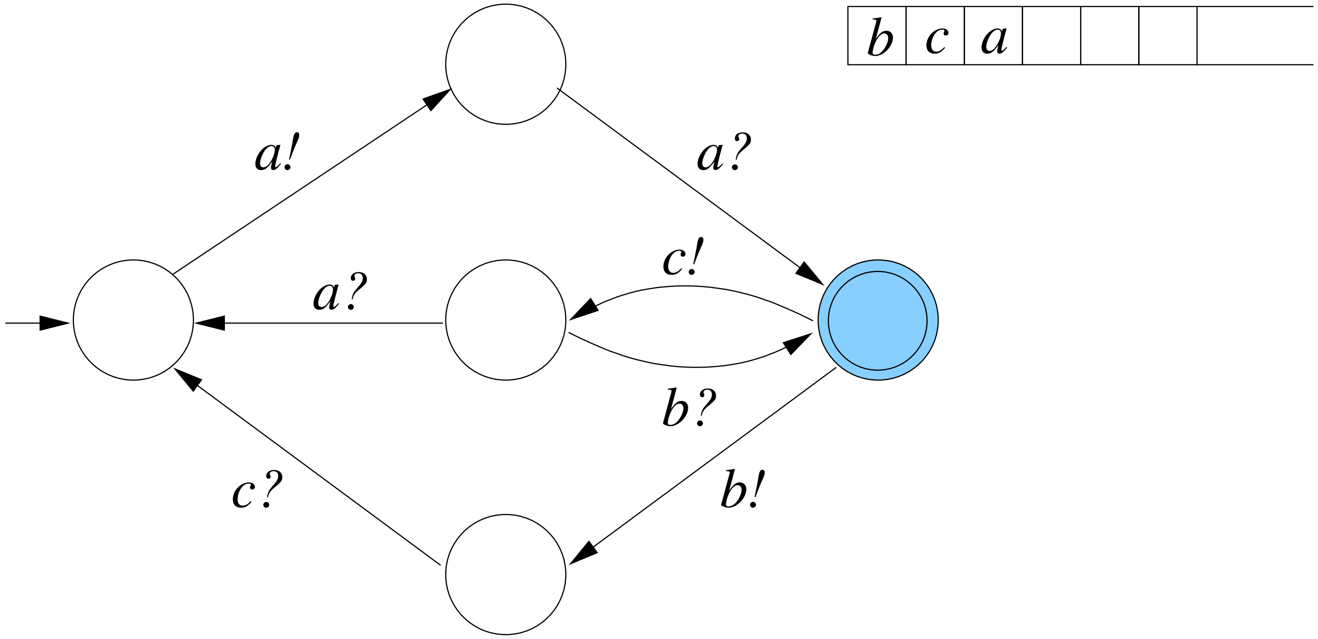
Interlude: Channel Machines

Our undecidability results use **channel machines**.



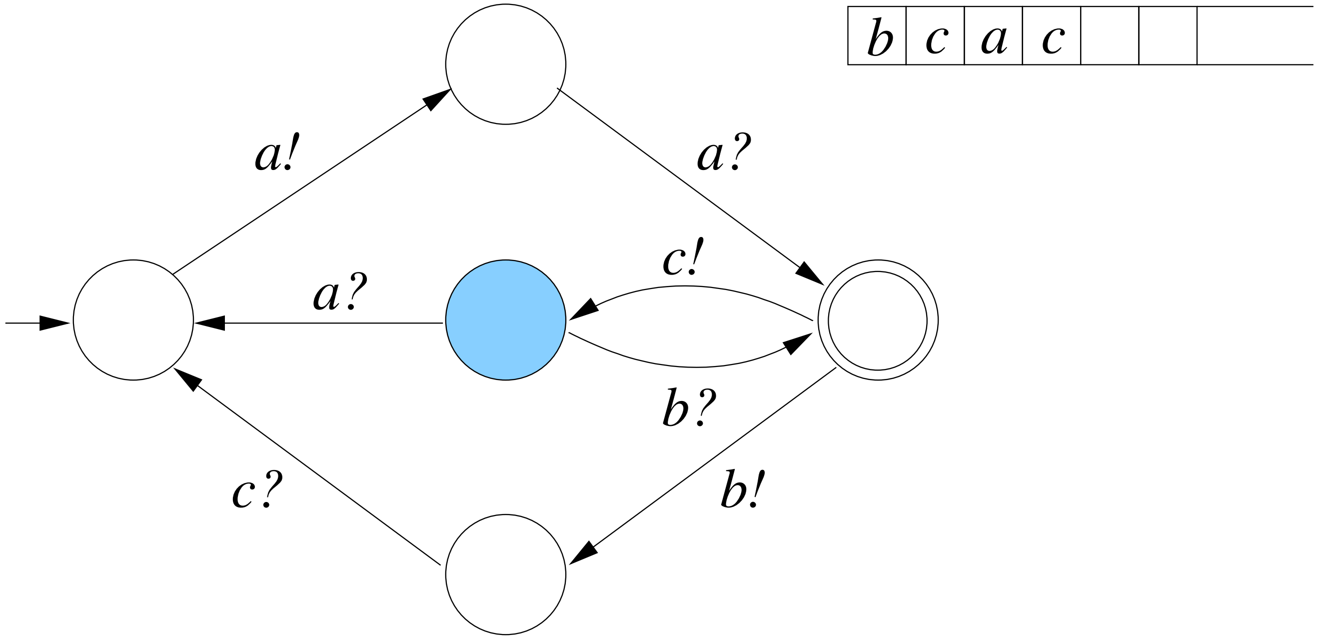
Interlude: Channel Machines

Our undecidability results use **channel machines**.



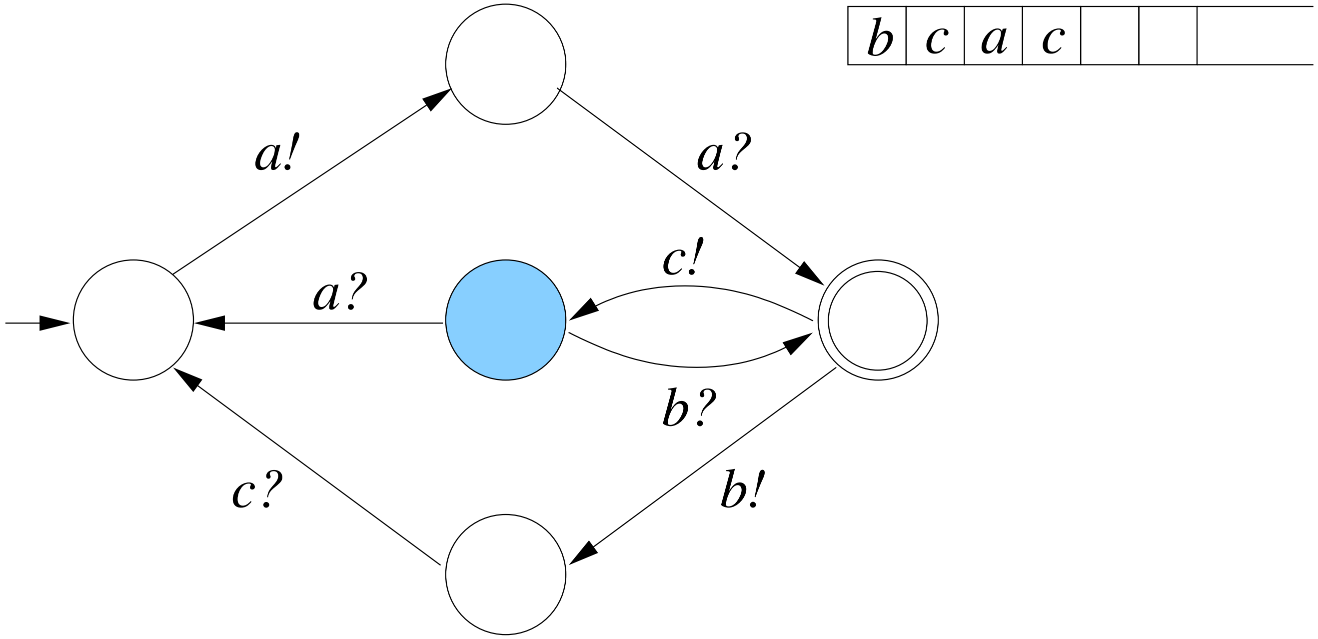
Interlude: Channel Machines

Our undecidability results use **channel machines**.



Interlude: Channel Machines

Our undecidability results use **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 C be a channel machine. Encode the computations of C as timed words:

Undecidability for MTL

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

Proof: Let C be a channel machine. Encode the computations of C as timed words:

- Successive control states of C are encoded by events with integer timestamps.

Undecidability for MTL

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

Proof: Let C be a channel machine. Encode the computations of C as timed words:

- Successive control states of C are encoded by events with integer timestamps.
- The channel contents of 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 C be a channel machine. Encode the computations of C as timed words:

- Successive control states of C are encoded by events with integer timestamps.
- The channel contents of 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 C be a channel machine. Encode the computations of C as timed words:

- Successive control states of C are encoded by events with integer timestamps.
- The channel contents of 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 C.

Then C halts iff φ is satisfiable.

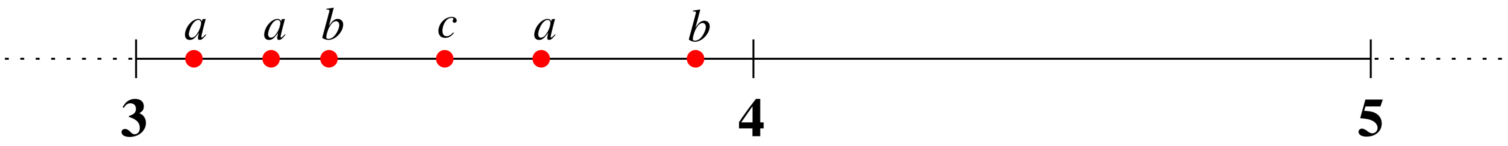
Propagating the Channel Contents

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

Propagating the Channel Contents

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

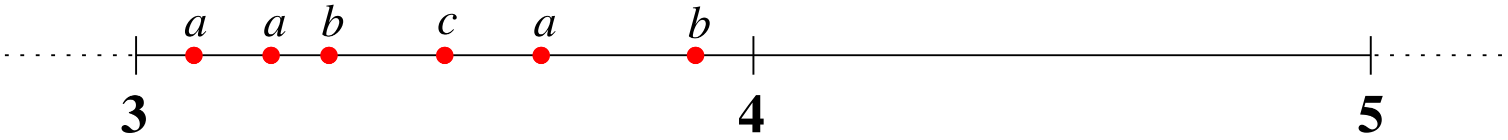
This is captured as:



Propagating the Channel Contents

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

This is captured as:



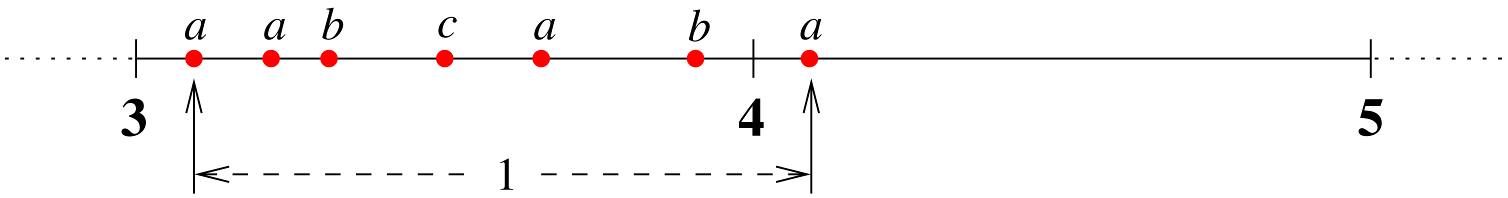
To correctly propagate the channel contents to the next time interval, we use the following formula:

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

Propagating the Channel Contents

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

This is captured as:



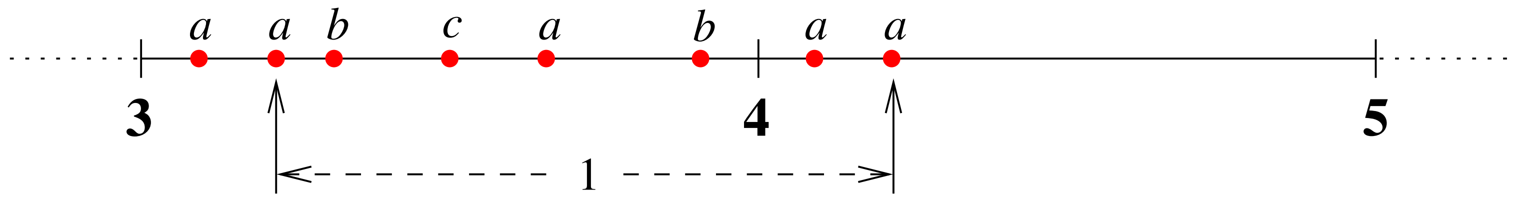
To correctly propagate the channel contents to the next time interval, we use the following formula:

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

Propagating the Channel Contents

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

This is captured as:



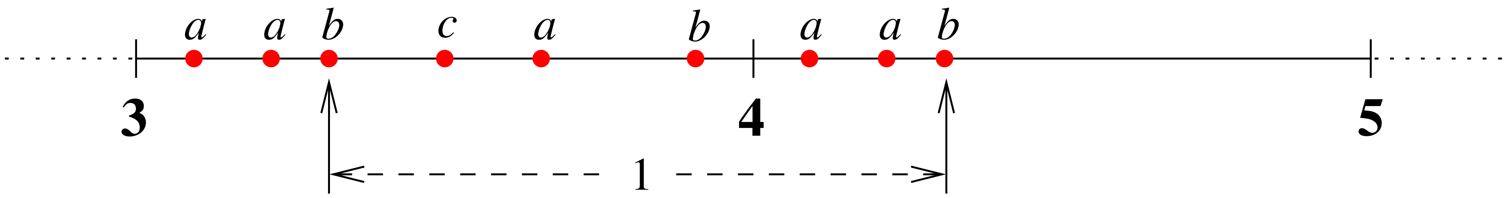
To correctly propagate the channel contents to the next time interval, we use the following formula:

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

Propagating the Channel Contents

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

This is captured as:



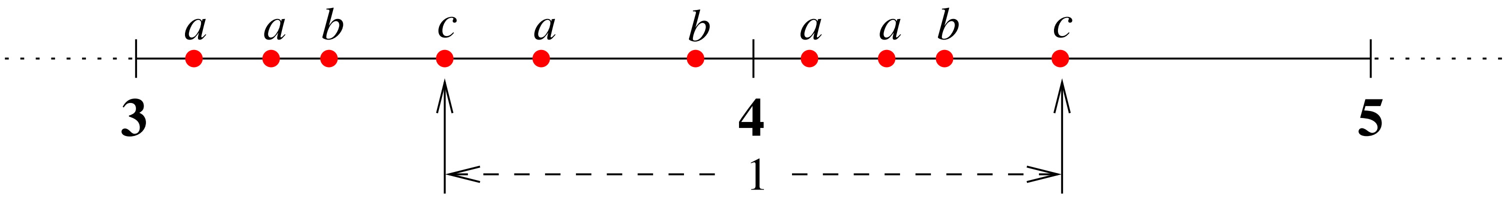
To correctly propagate the channel contents to the next time interval, we use the following formula:

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

Propagating the Channel Contents

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

This is captured as:



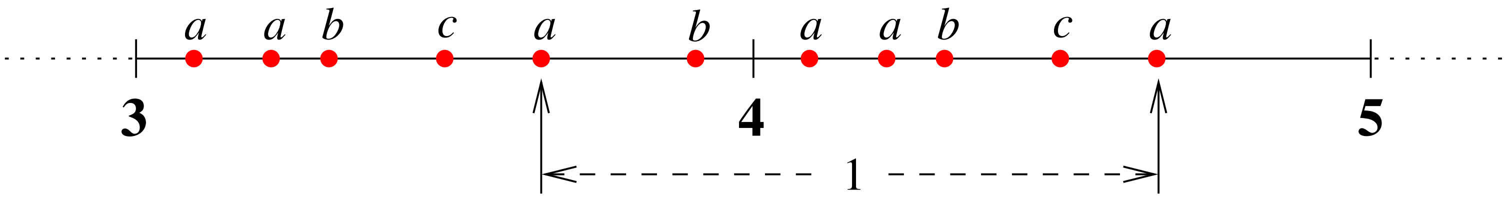
To correctly propagate the channel contents to the next time interval, we use the following formula:

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

Propagating the Channel Contents

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

This is captured as:



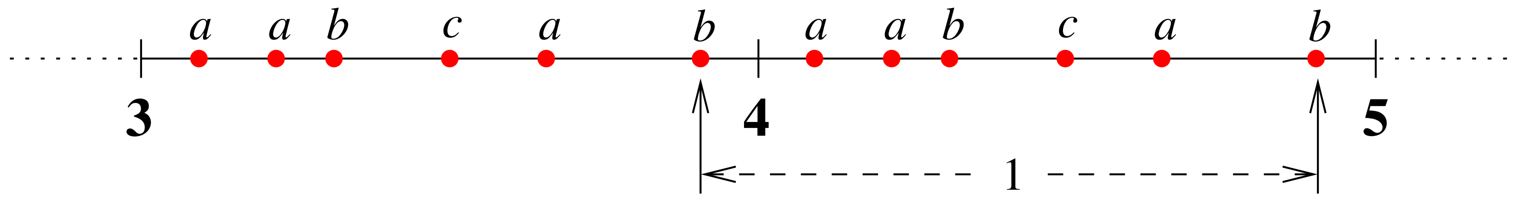
To correctly propagate the channel contents to the next time interval, we use the following formula:

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

Propagating the Channel Contents

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

This is captured as:



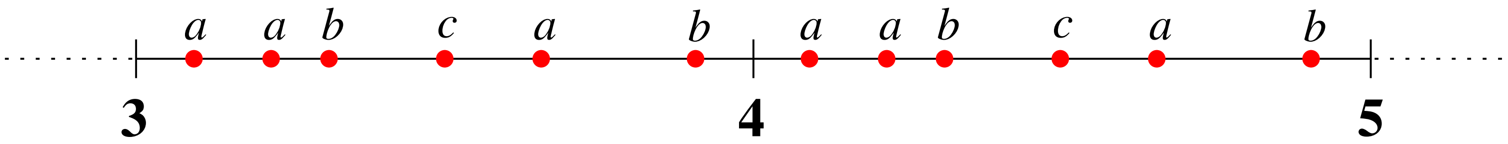
To correctly propagate the channel contents to the next time interval, we use the following formula:

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

Propagating the Channel Contents

Suppose that at time 3, the current channel contents of 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:

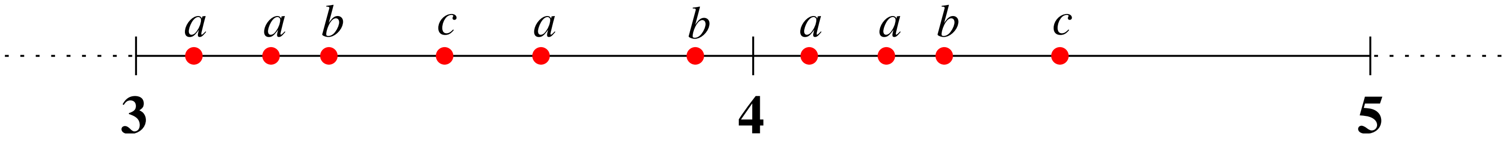
$$\bigwedge_{x \in \Sigma(M)} \Box (x \longrightarrow \Diamond_{=1} x)$$

Insertion Errors!

There is something missing — the formula

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

only enforces **forward** propagation of events:

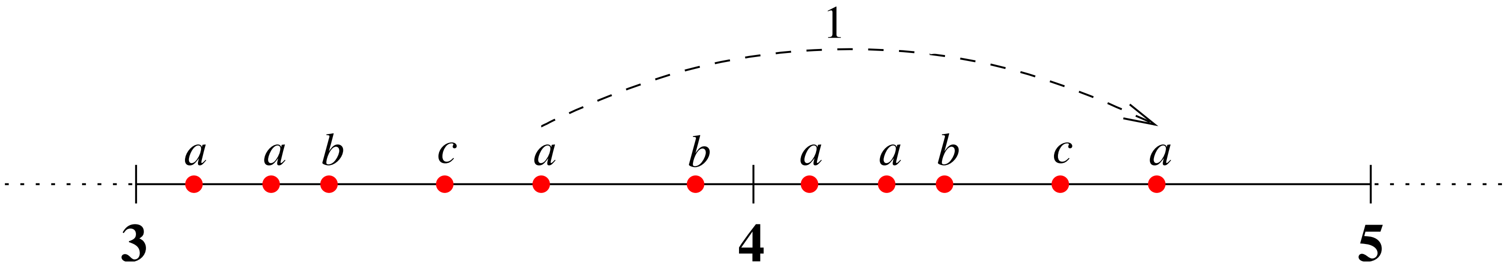


Insertion Errors!

There is something missing — the formula

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

only enforces **forward** propagation of events:

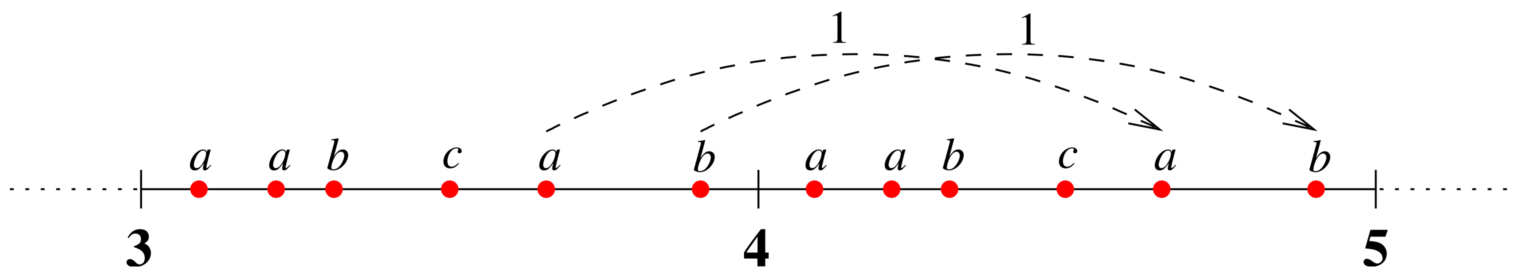


Insertion Errors!

There is something missing — the formula

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

only enforces **forward** propagation of events:

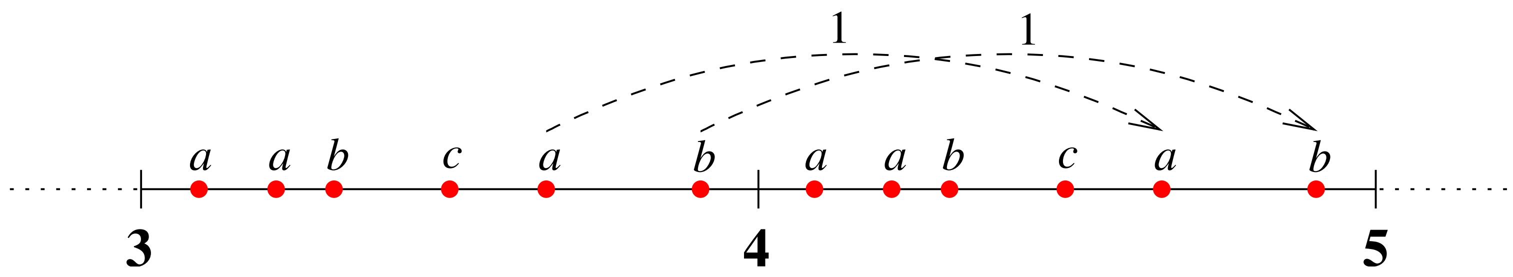


Insertion Errors!

There is something missing — the formula

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

only enforces **forward** propagation of events:



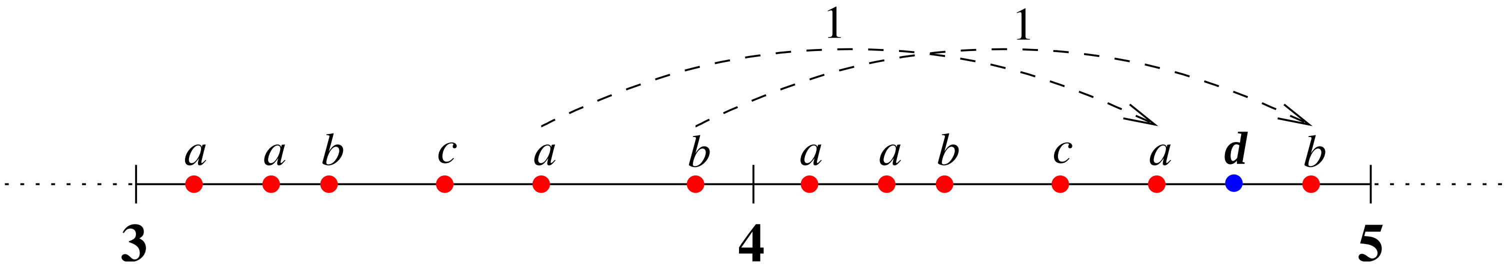
Nothing prevents, however, **insertion errors** from occurring.

Insertion Errors!

There is something missing — the formula

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

only enforces **forward** propagation of events:



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

Undecidability for MTL

- The decidability of a number of real-time logics was considered in *Real-time Logics: Complexity and Expressiveness* [Alur and Henzinger 93].
- The undecidability proofs of Alur and Henzinger rely on encoding computations of Turing machines as timed words. They only apply to richer formalisms than MTL.

Our Contribution

	Satisfiability	Model Checking
MTL (Inf.)	??	??
MTL (Fin.)	??	??
Safety MTL (Inf.)	??	??

Our Contribution

	Satisfiability	Model Checking
MTL (Inf.)	??	??
MTL (Fin.)	??	??
Safety MTL (Inf.)	??	??

- Safety MTL: bounded timing constraints on all eventuality operators.

Our Contribution

	Satisfiability	Model Checking
MTL (Inf.)	??	??
MTL (Fin.)	Dec. / NPR	Dec. / NPR
Safety MTL (Inf.)	??	Dec. / ??

- Safety MTL: bounded timing constraints on all eventuality operators.

MTL Satisfiability over Finite Words

Theorem. MTL satisfiability over finite timed words is **decidable**.

MTL Satisfiability over Finite Words

Theorem. MTL satisfiability over finite timed words is **decidable**.

Proof: Given φ, construct an equivalent **timed alternating automaton** with a *single* clock:

MTL Satisfiability over Finite Words

Theorem. MTL satisfiability over finite timed words is **decidable**.

Proof: Given φ, construct an equivalent **timed alternating automaton** with a *single* clock:

- States of A are the modal subformulas of φ.

MTL Satisfiability over Finite Words

Theorem. MTL satisfiability over finite timed words is **decidable**.

Proof: Given φ, construct an equivalent **timed alternating automaton** with a *single* clock:

- States of A are the modal subformulas of φ.
- A single clock is needed to enforce any single timing constraint.

MTL Satisfiability over Finite Words

Theorem. MTL satisfiability over finite timed words is **decidable**.

Proof: Given φ, construct an equivalent **timed alternating automaton** with a *single* clock:

- States of A are the modal subformulas of φ.
- A single clock is needed to enforce any single timing constraint.
- The conjunctive branching of A spawns a new computation thread (with a new fresh single clock) for every new timing requirement.

MTL Satisfiability over Finite Words

Theorem. MTL satisfiability over finite timed words is **decidable**.

Proof: Given φ, construct an equivalent **timed alternating automaton** with a *single* clock:

- States of A are the modal subformulas of φ.
- A single clock is needed to enforce any single timing constraint.
- The conjunctive branching of A spawns a new computation thread (with a new fresh single clock) for every new timing requirement.
- Satisfiability for φ reduces to language emptiness for A.

Punctual response: let φ be the formula $\Box(a \longrightarrow \Diamond_{=1} b)$.

Punctual response: let φ be the formula $\Box(a \longrightarrow \Diamond_{=1} b)$.

- $s_0 : \Box(a \longrightarrow \Diamond_{=1} b)$. Initial state, accepting.
- $s_1 : \Diamond_{=1} b$. Non-accepting.

Punctual response: let φ be the formula $\Box(a \longrightarrow \Diamond_{=1} b)$.

- $s_0 : \Box(a \longrightarrow \Diamond_{=1} b)$. Initial state, accepting.
- $s_1 : \Diamond_{=1} b$. Non-accepting.

A_φ:

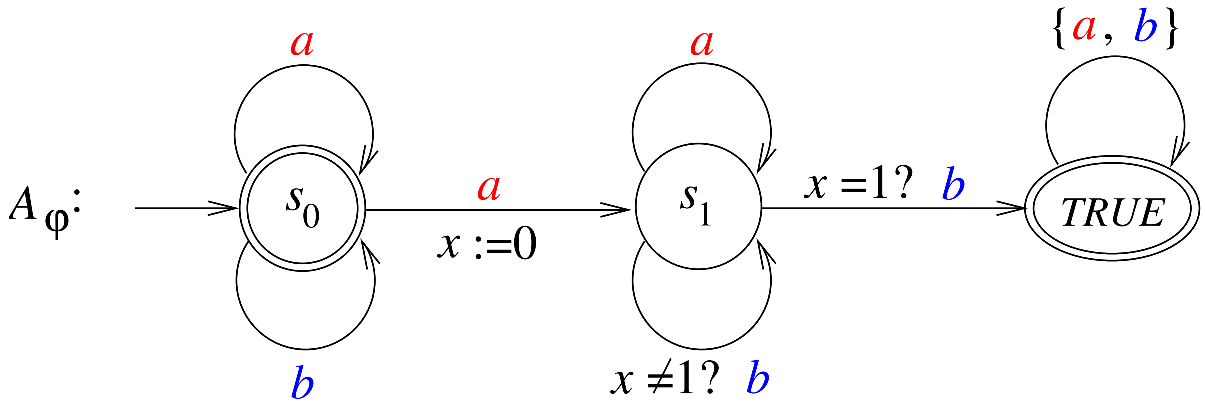
	a	b
s_0	$s_0 \wedge x.s_1$	s_0
s_1	s_1	$(x = 1) \vee s_1$

Punctual response: let φ be the formula $\Box(a \longrightarrow \Diamond_{=1} b)$.

- $s_0 : \Box(a \longrightarrow \Diamond_{=1} b)$. Initial state, accepting.
- $s_1 : \Diamond_{=1} b$. Non-accepting.

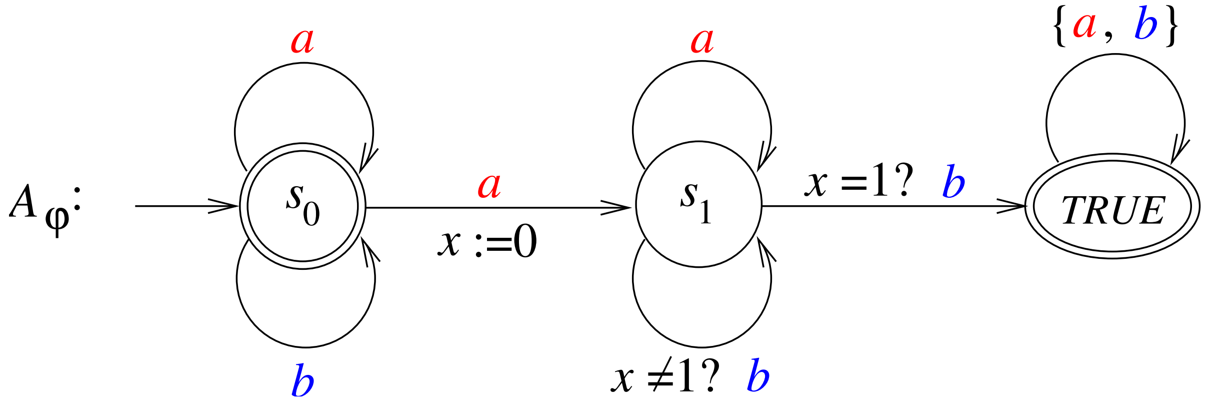
A_φ:

	a	b
s_0	$s_0 \wedge x.s_1$	s_0
s_1	s_1	$(x = 1) \vee s_1$

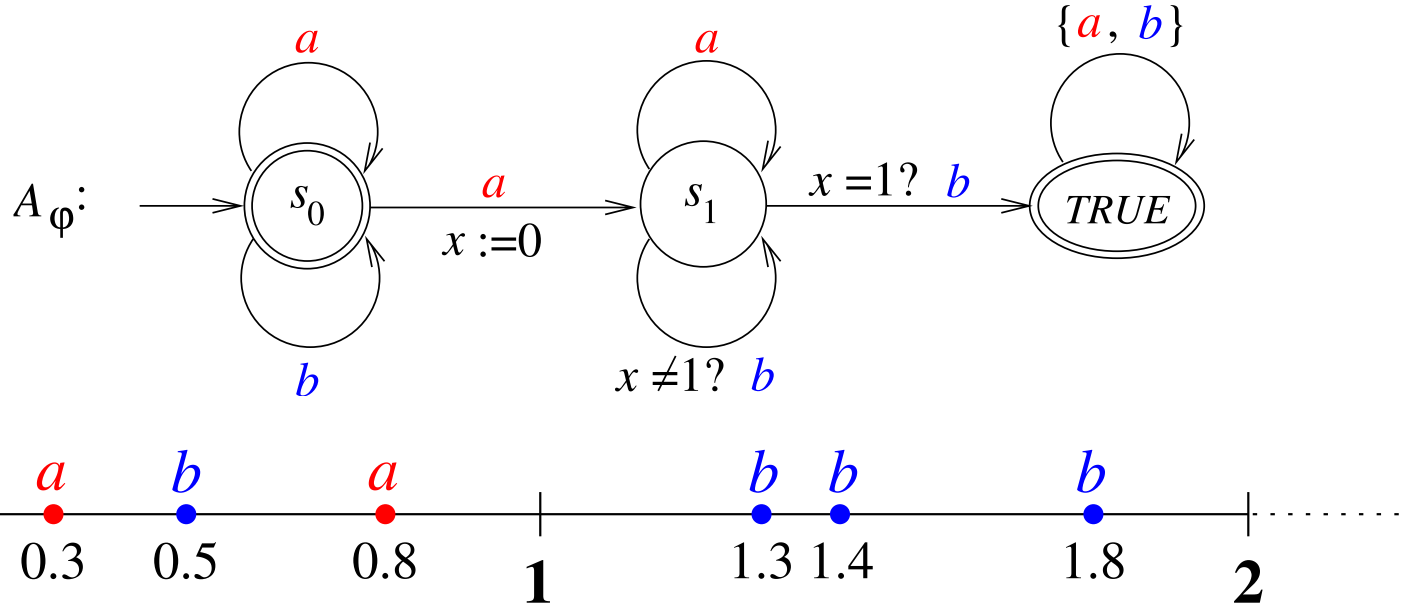


$$\varphi\colon\quad \Box(a \longrightarrow \Diamond_{=1} b)$$

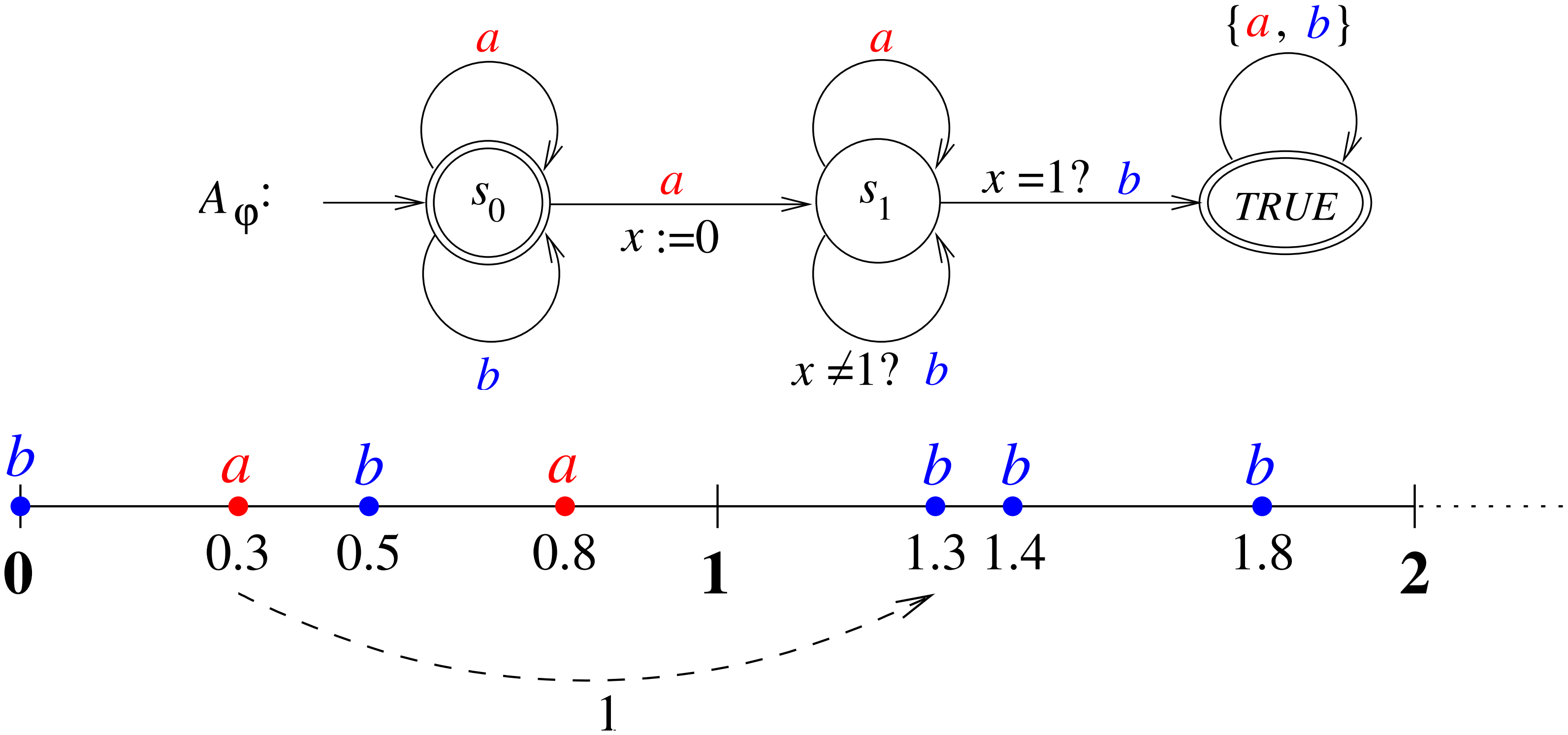
$$\varphi\colon\quad \Box(a \longrightarrow \Diamond_{=1} b)$$



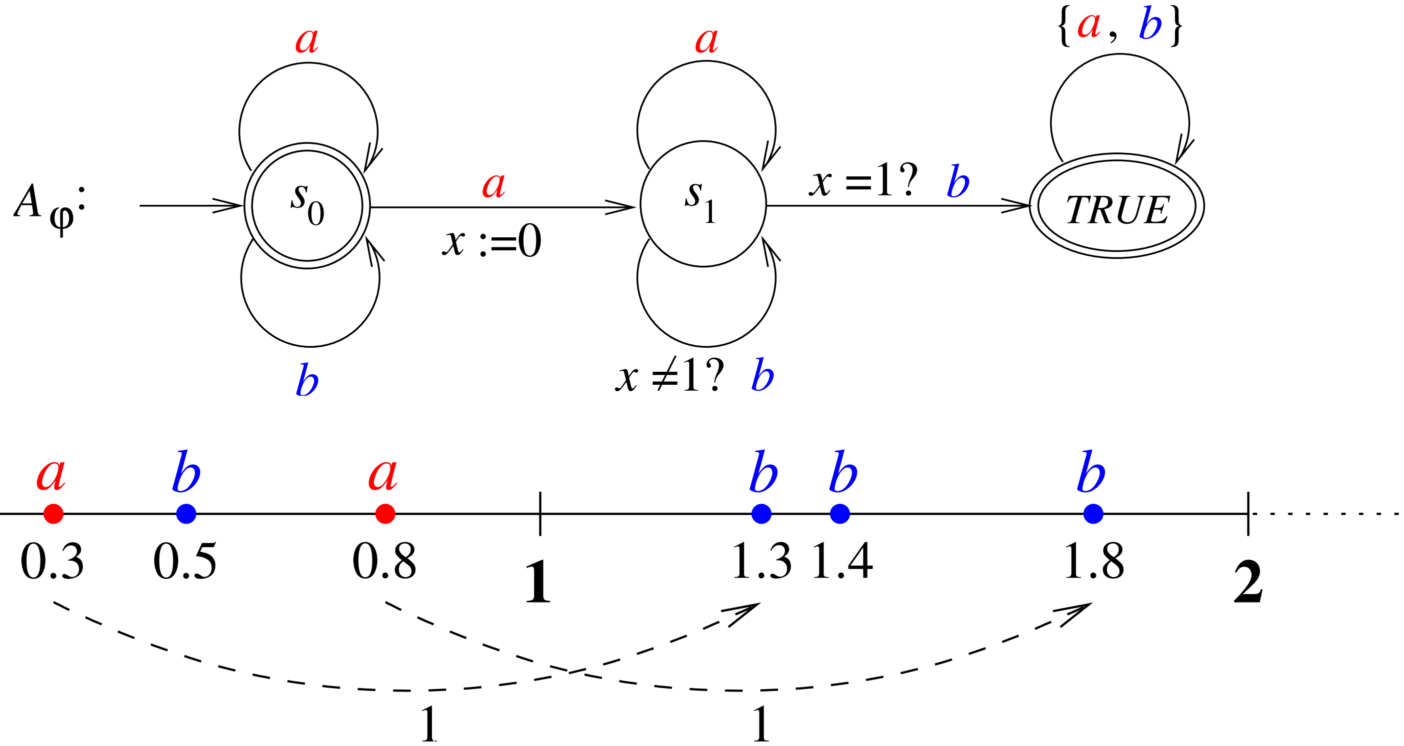
φ: $\square(a \longrightarrow \lozenge_{=1} b)$



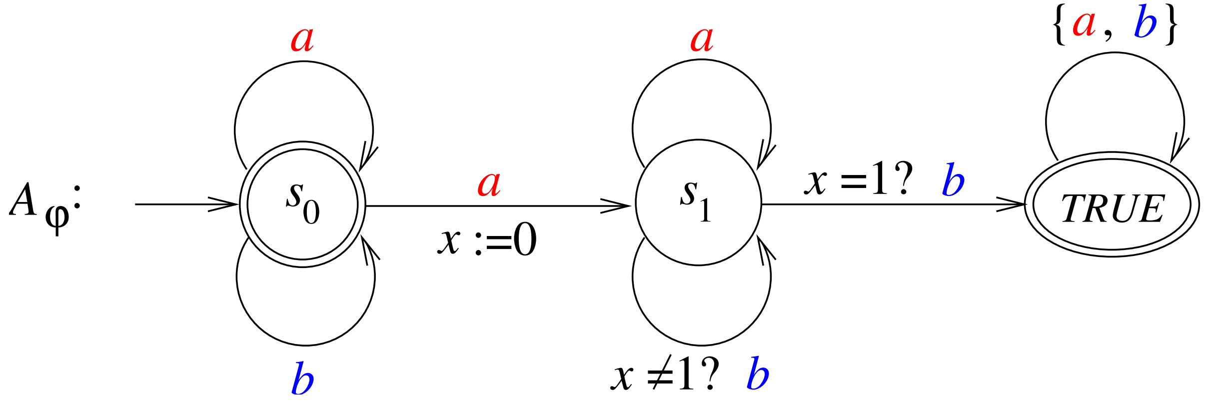
φ: $\Box(a \longrightarrow \Diamond_{=1} b)$

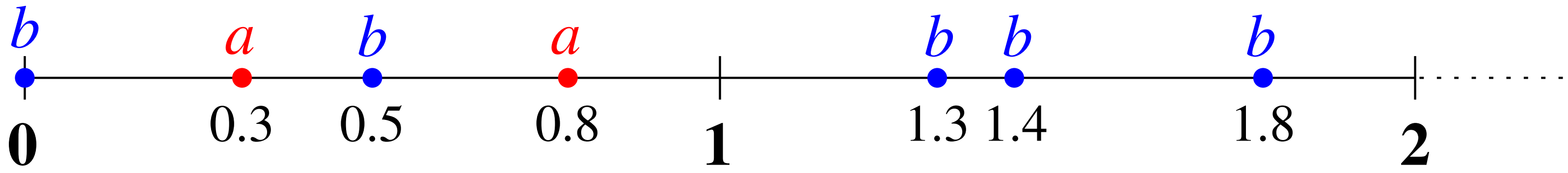


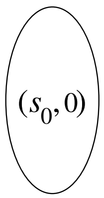
$$\varphi: \quad \Box(a \longrightarrow \Diamond_{=1} b)$$



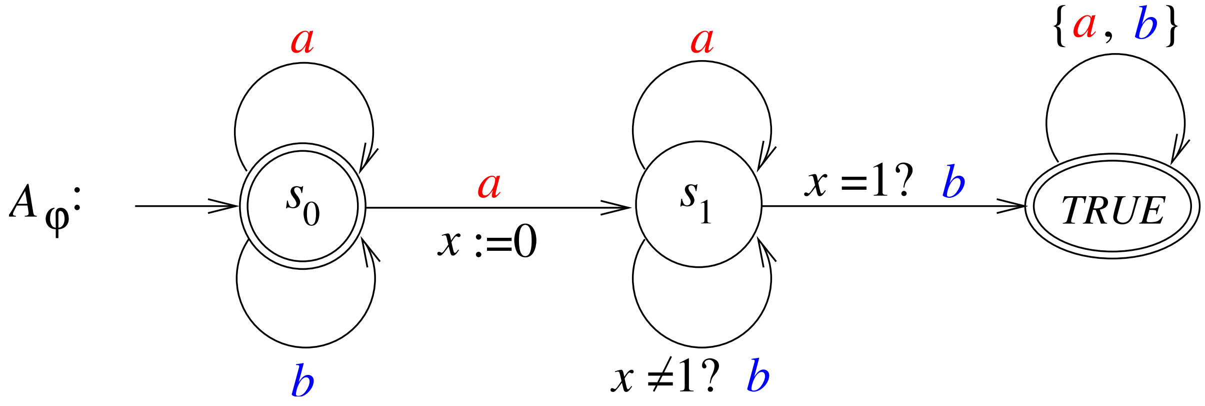
φ: $\Box(a \longrightarrow \Diamond_{=1} b)$

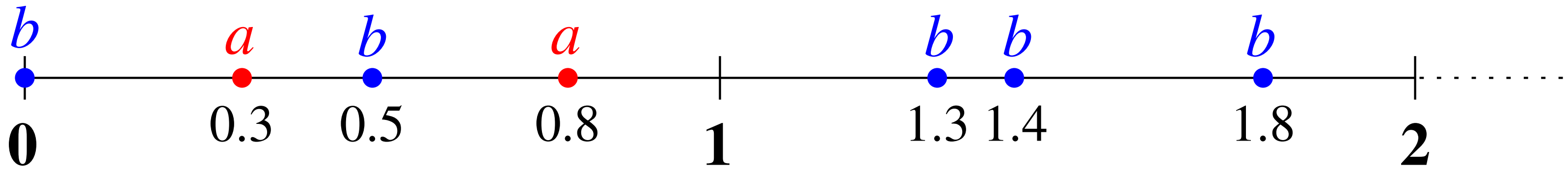


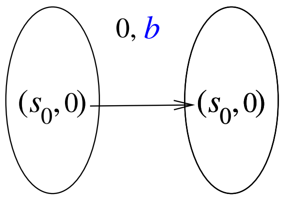




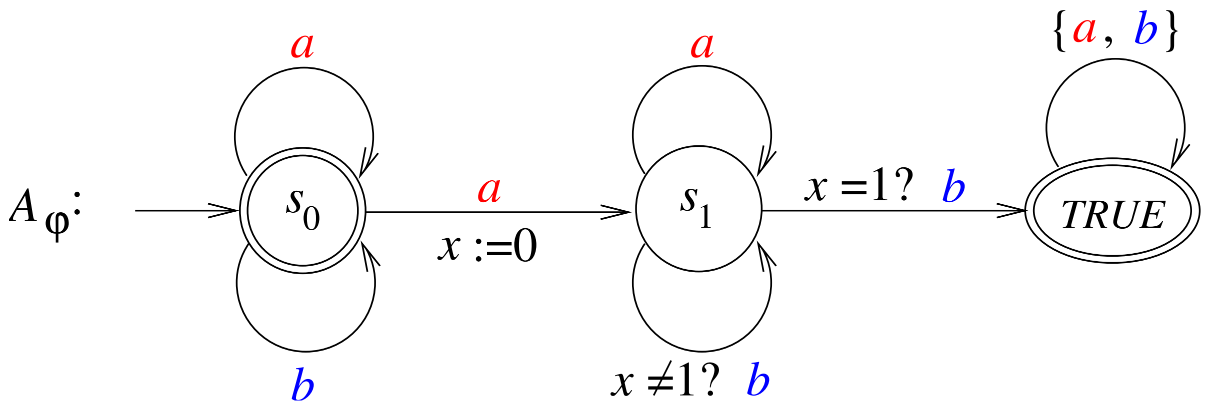
φ: $\Box(a \longrightarrow \Diamond_{=1} b)$

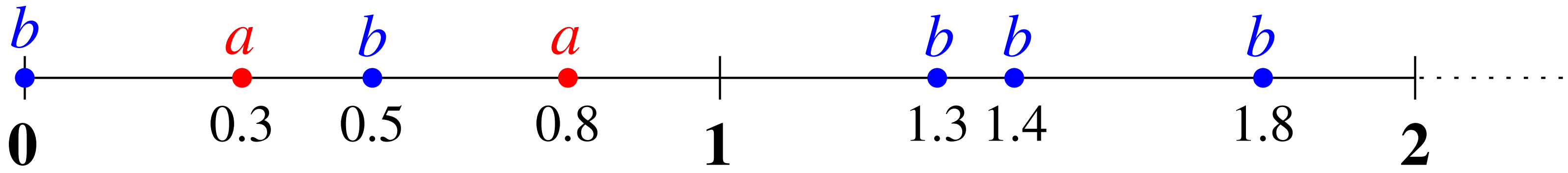


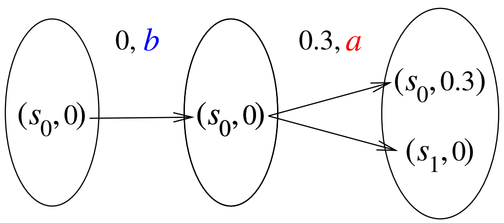




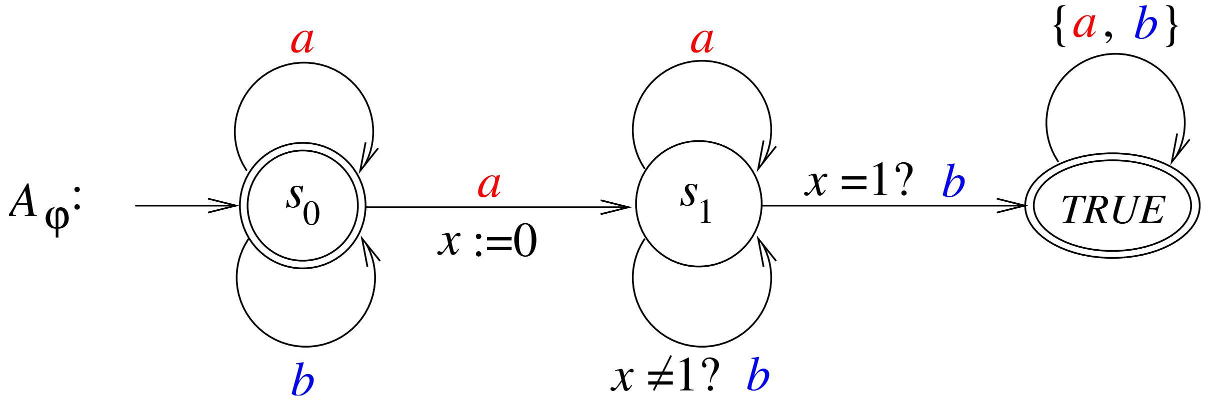
φ: $\Box(a \longrightarrow \Diamond_{=1} b)$

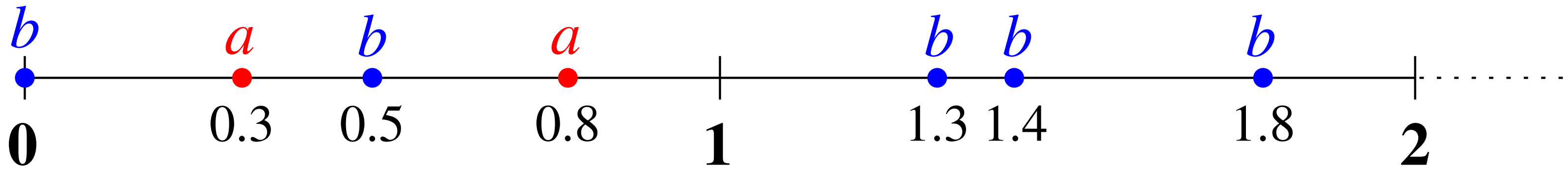


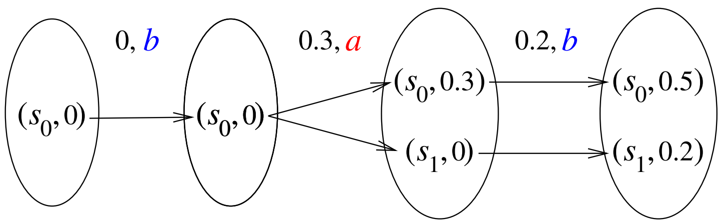




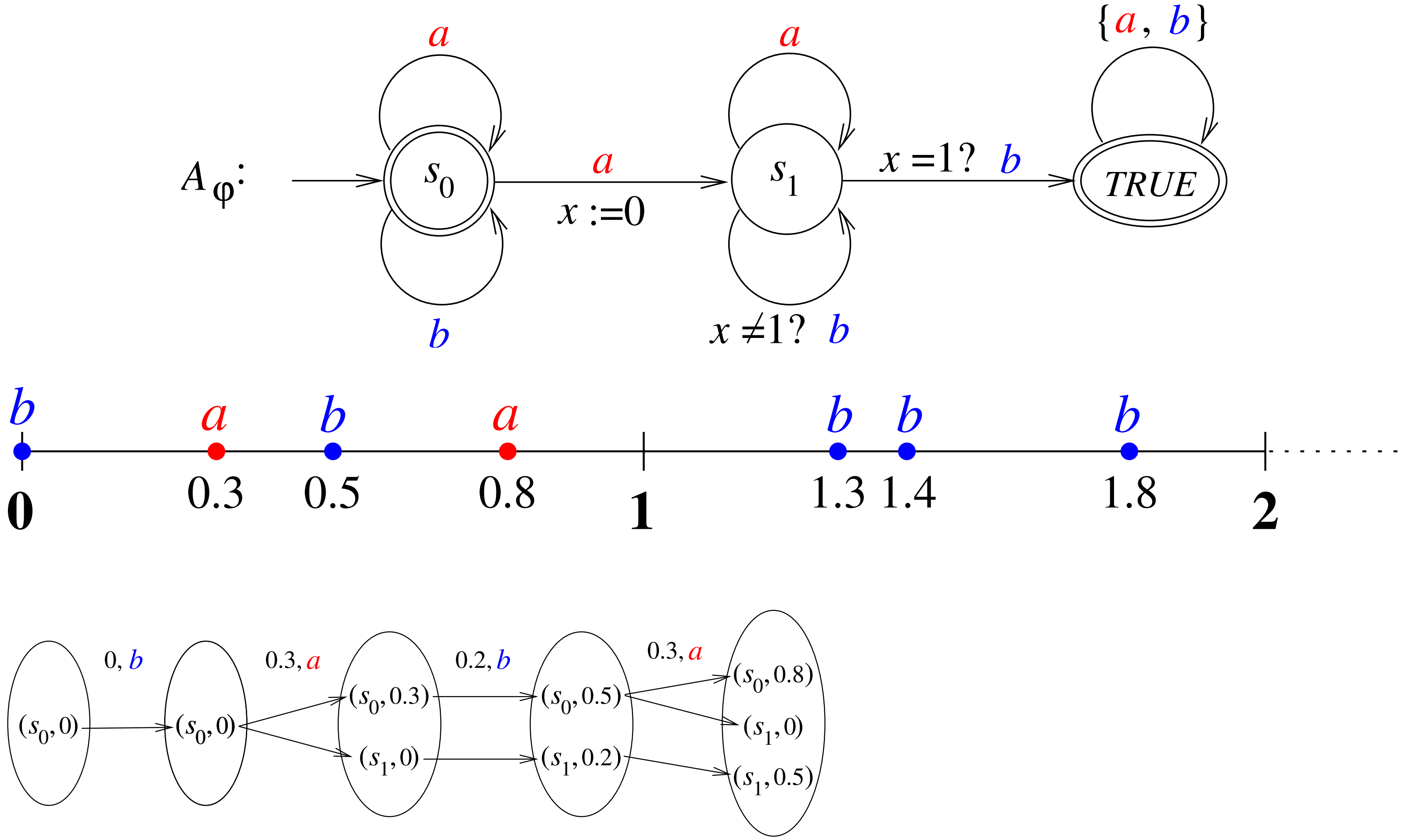
$$\varphi: \quad \Box(a \longrightarrow \Diamond_{=1} b)$$



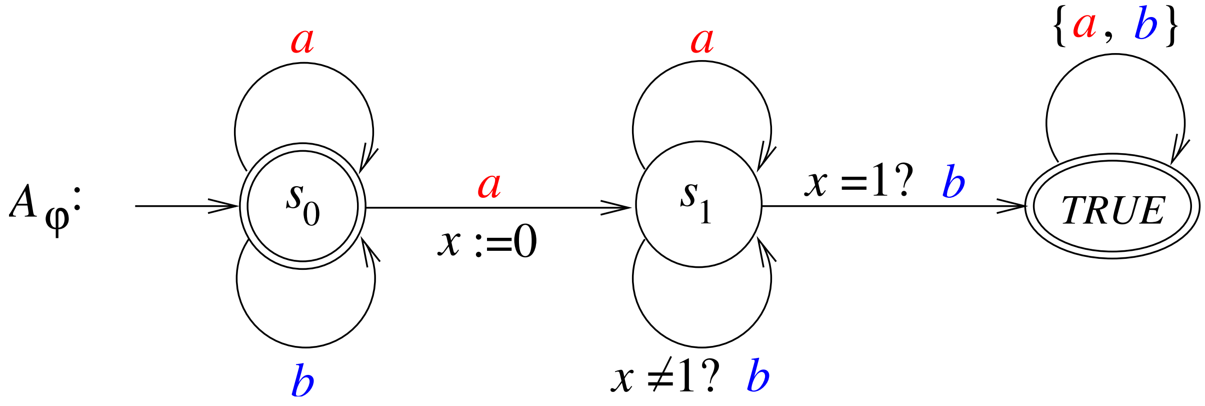


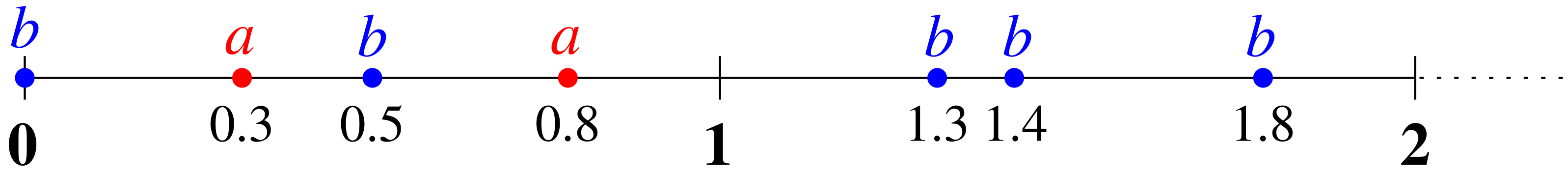


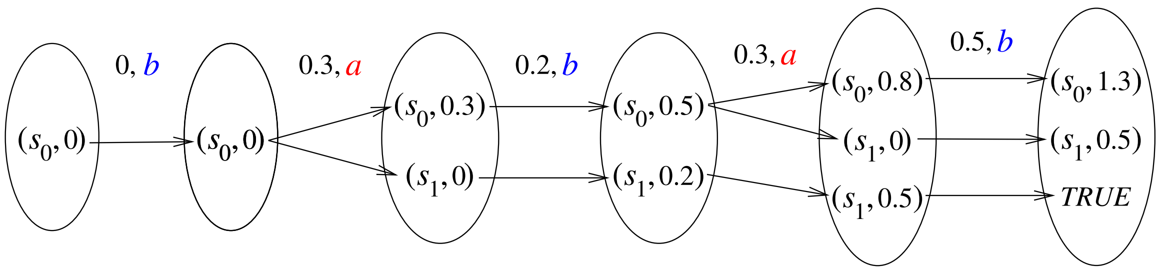
$$\varphi\colon\ \Box(a \longrightarrow \Diamond_{=1} b)$$



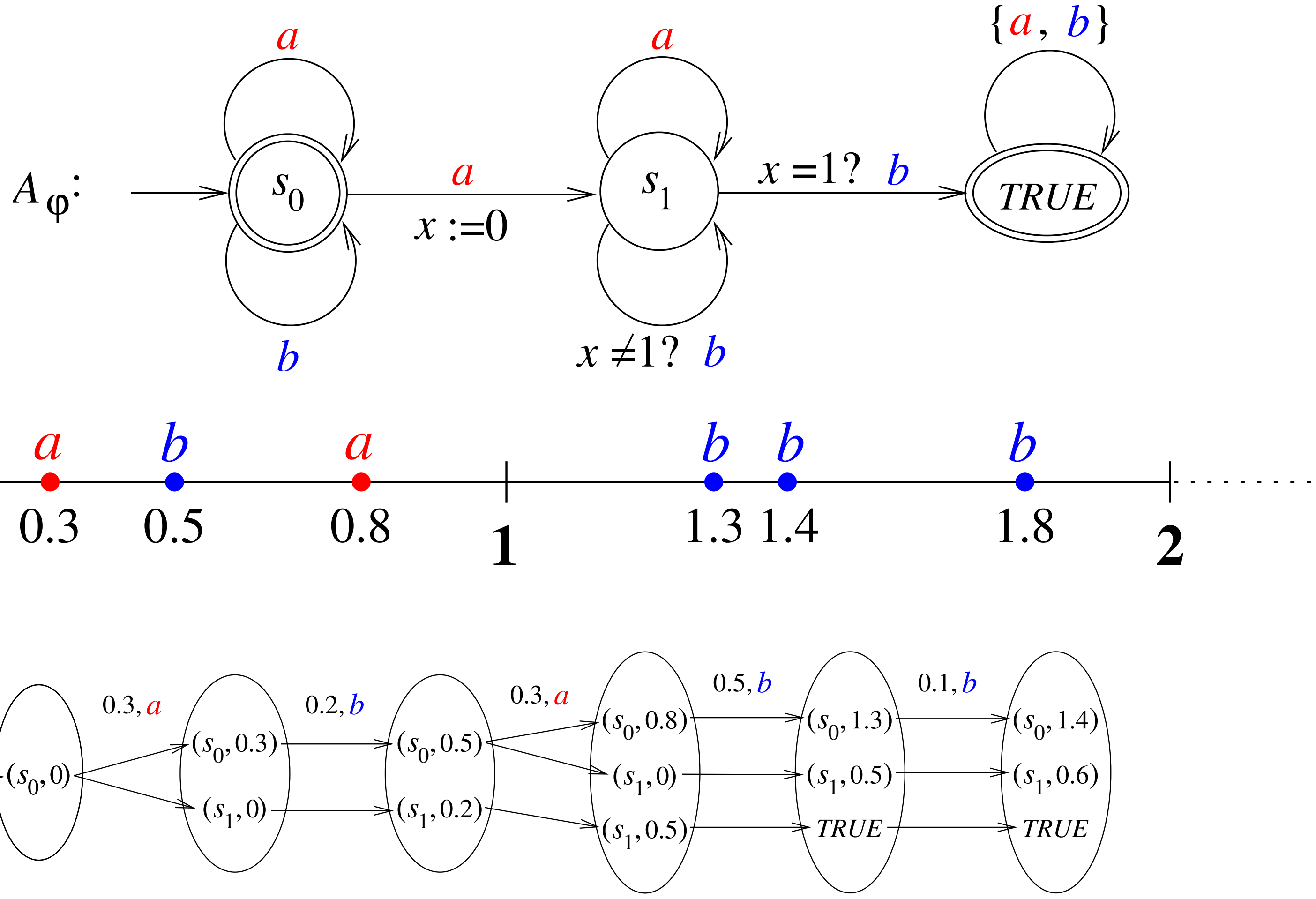
$$\varphi:\quad \Box(a \longrightarrow \Diamond_{=1} b)$$

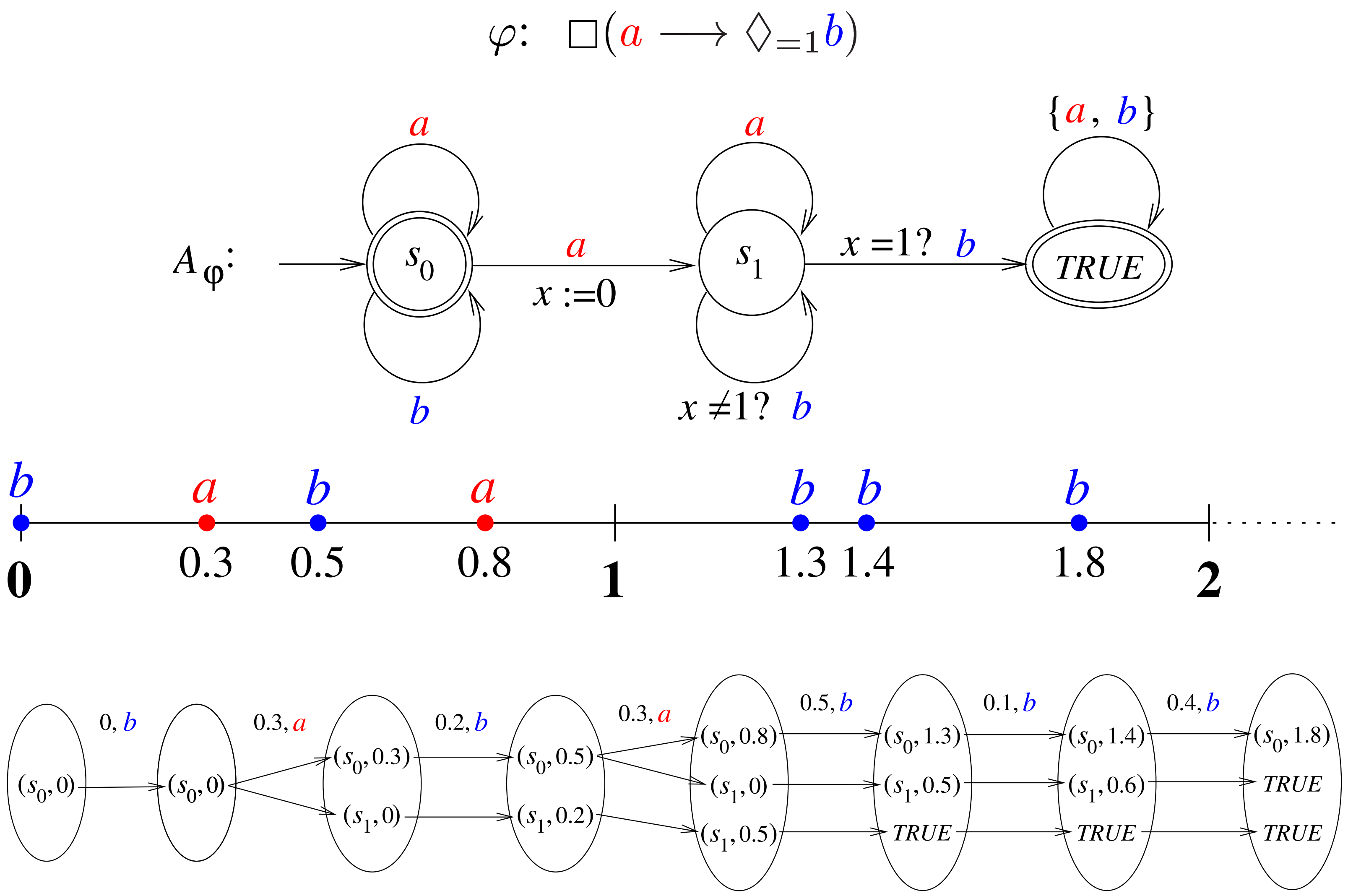






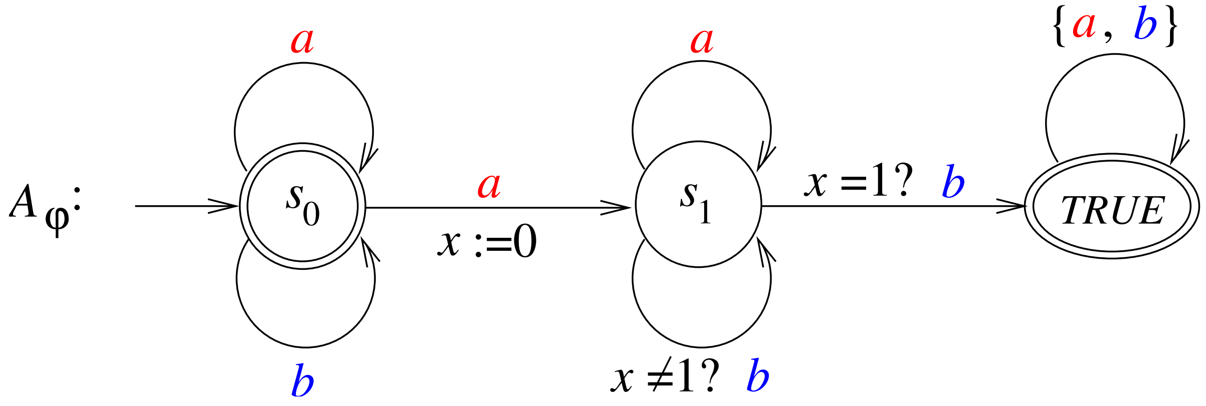
$$\varphi: \quad \Box(a \longrightarrow \Diamond_{=1} b)$$

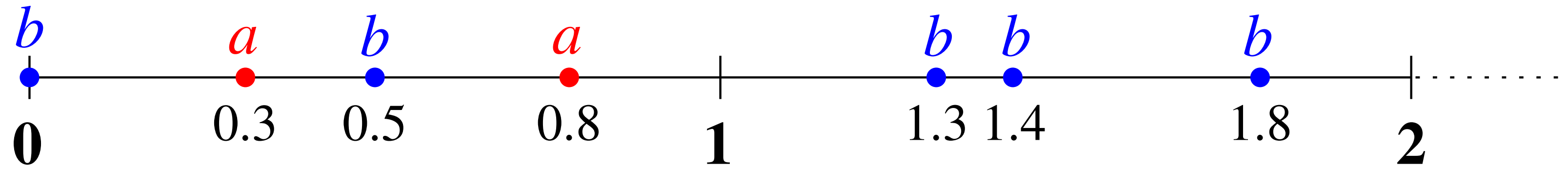


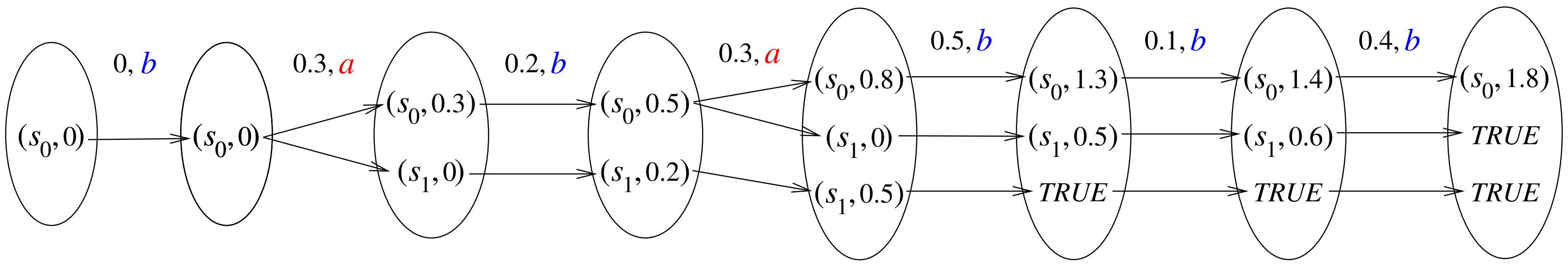


φ: □(a ⟶ ◊=1 b)
Aφ:
s0
s1
TRUE
a
b
x :=0
x =1? b
x ≠1? b
{a, b}
0
0.3
0.5
0.8
1
1.3
1.4
1.8
2
(s0, 0)
0, b
0.3, a
0.2, b
0.3, a
0.5, b
0.1, b
0.4, b
(s0, 0.3)
(s1, 0)
(s0, 0.5)
(s1, 0.2)
(s0, 0.8)
(s1, 0.5)
(s0, 1.3)
(s1, 0.5)
(s0, 1.4)
(s1, 0.6)
(s0, 1.8)
TRUE

$$\varphi: \quad \Box(a \longrightarrow \Diamond_{=1} b)$$







- Since all states in final slice are accepting, $w \vDash \varphi$.

Well-quasi-orders

- Deciding language-emptiness for A is non-trivial since it has infinitely many configurations and does not admit a finite quotient.

Well-quasi-orders

- Deciding language-emptiness for A is non-trivial since it has infinitely many configurations and does not admit a finite quotient.
- In lieu of finiteness we establish the existence of a **well-quasi-order (wqo)** on the configurations of A.

Well-quasi-orders

- Deciding language-emptiness for A is non-trivial since it has infinitely many configurations and does not admit a finite quotient.
- In lieu of finiteness we establish the existence of a **well-quasi-order (wqo)** on the configurations of A.
  - A quasi-order $(Q, \preccurlyeq)$ is a well-quasi-order if for every infinite sequence $q_0, q_1, q_2, \ldots$ in Q there exist indices $i < j$ such that $q_i \preccurlyeq q_j$.

Well-quasi-orders

- Deciding language-emptiness for A is non-trivial since it has infinitely many configurations and does not admit a finite quotient.
- In lieu of finiteness we establish the existence of a **well-quasi-order (wqo)** on the configurations of A.
  - A quasi-order $(Q, \preccurlyeq)$ is a well-quasi-order if for every infinite sequence $q_0, q_1, q_2, \ldots$ in Q there exist indices $i < j$ such that $q_i \preccurlyeq q_j$.
  - The subword order (e.g., $HIGMAN \preccurlyeq HIGHMOUNTAIN$) is a wqo.

Well-quasi-orders

- Deciding language-emptiness for A is non-trivial since it has infinitely many configurations and does not admit a finite quotient.
- In lieu of finiteness we establish the existence of a **well-quasi-order (wqo)** on the configurations of A.
  - A quasi-order $(Q, \preccurlyeq)$ is a well-quasi-order if for every infinite sequence $q_0, q_1, q_2, \ldots$ in Q there exist indices $i < j$ such that $q_i \preccurlyeq q_j$.
  - The subword order (e.g., $HIGMAN \preccurlyeq HIGHMOUNTAIN$) is a wqo.
- We solve the language-emptiness problem for A by a forward-reachability algorithm. Termination is guaranteed by the wqo on the configurations of A.

Our Contribution

	Satisfiability	Model Checking
MTL (Inf.)	??	??
MTL (Fin.)	Dec. / NPR	Dec. / NPR
Safety MTL (Inf.)	??	Dec. / ??

- Safety MTL: bounded timing constraints on all eventuality operators.

Our Contribution

	Satisfiability	Model Checking
MTL (Inf.)	Undec.	Undec.
MTL (Fin.)	Dec. / NPR	Dec. / NPR
Safety MTL (Inf.)	??	Dec. / ??

- Safety MTL: bounded timing constraints on all eventuality operators.

Channel Machines with Insertion Errors

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

Channel Machines with Insertion Errors

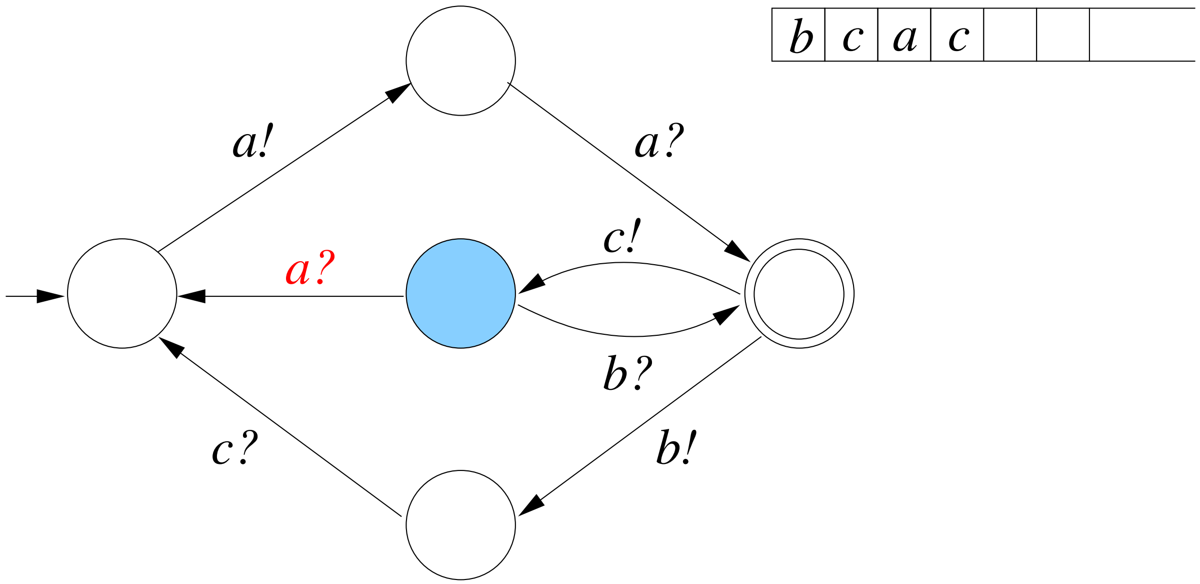
We can only simulate channel machines with insertion errors in MTL.

Read-transitions are always enabled.

Channel Machines with Insertion Errors

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

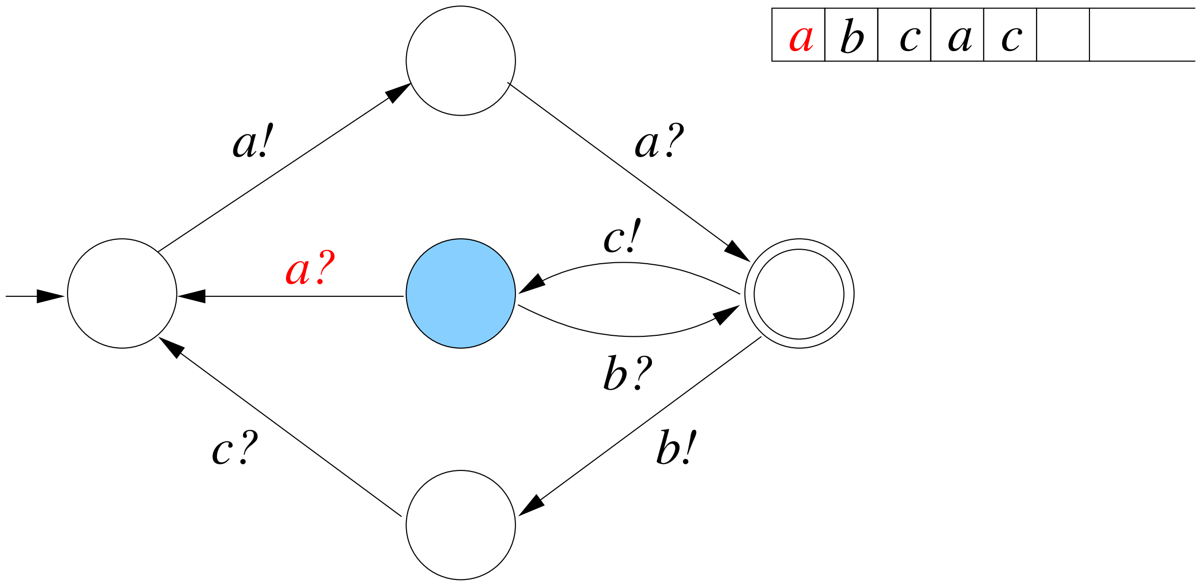
Read-transitions are always enabled.



Channel Machines with Insertion Errors

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

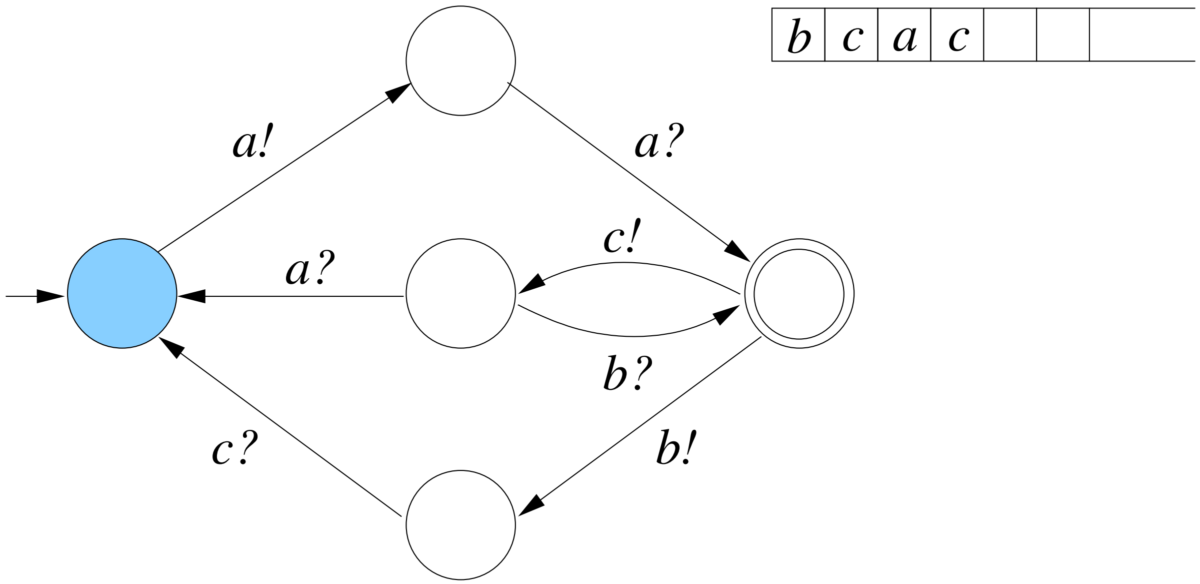
Read-transitions are always enabled.



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.

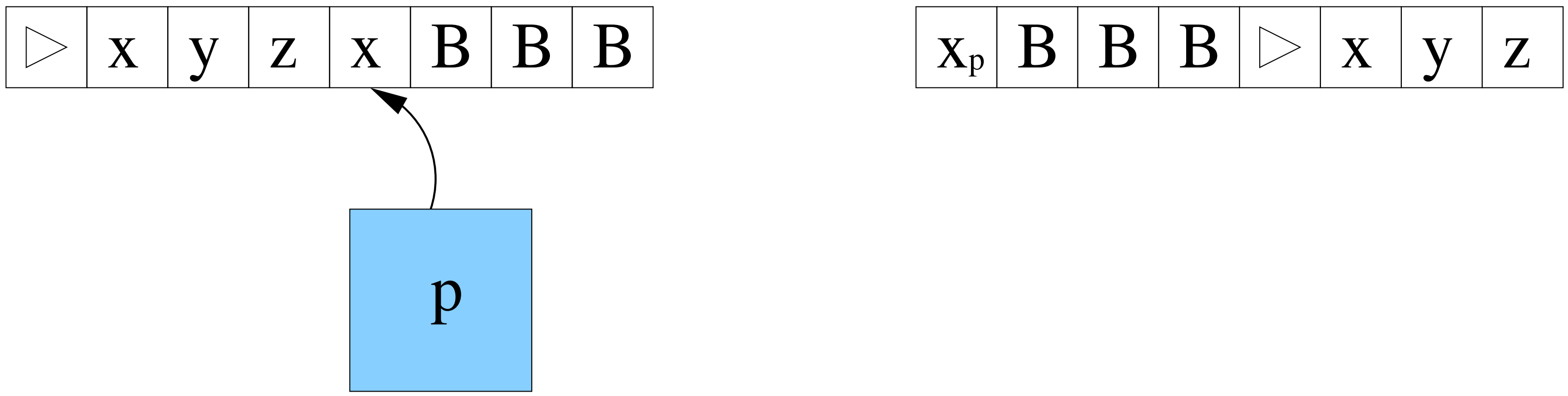
Recurrent-state Problem

- 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. Use liveness to implement error correction mechanism.

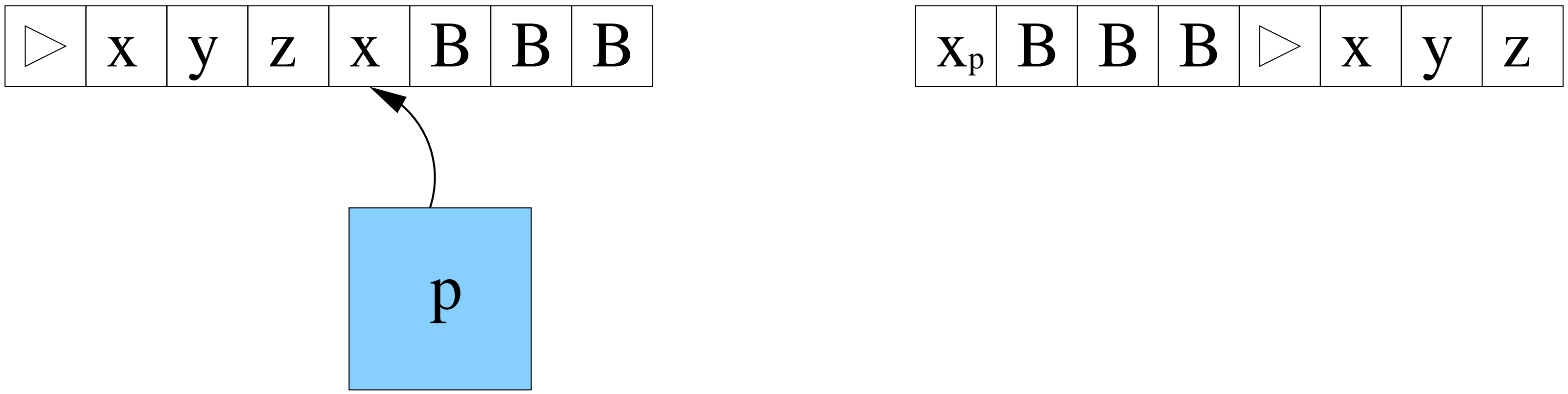
Simulating a Turing Machine

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

Simulation Phase

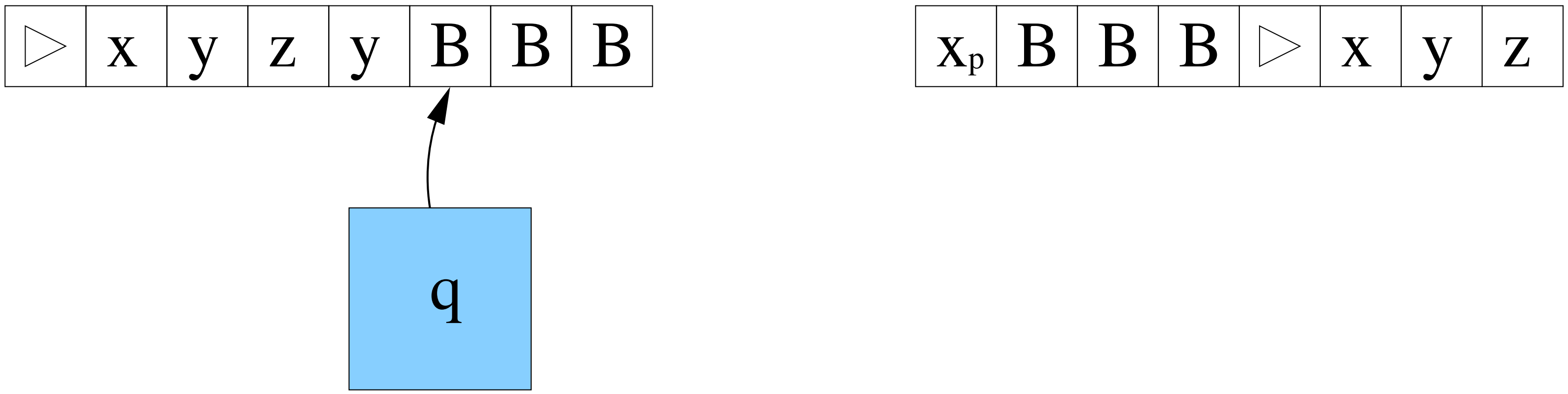


Simulation Phase



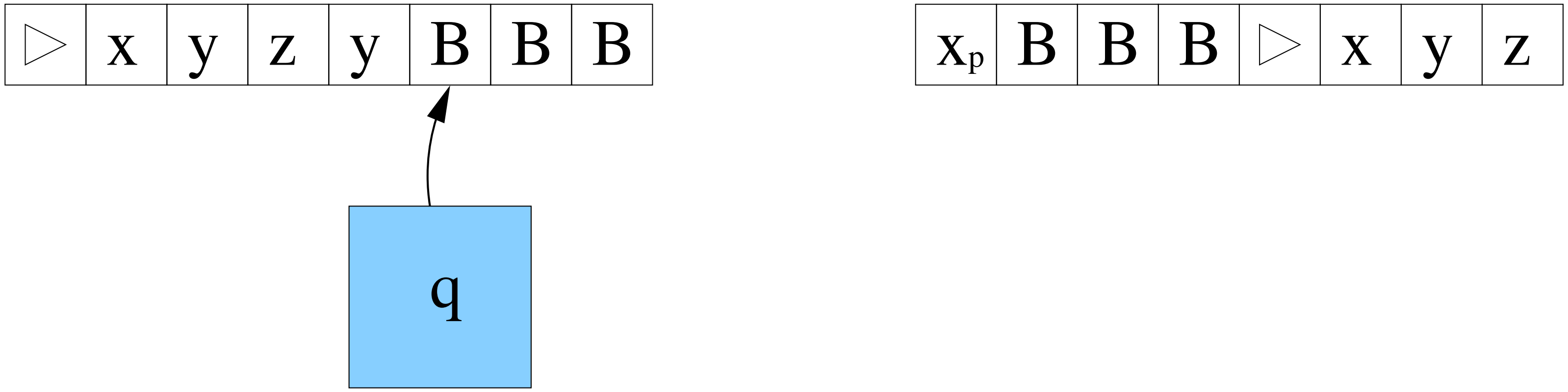
- Simulate transition $(p, x, q, y, \rightarrow)$ of M.

Simulation Phase



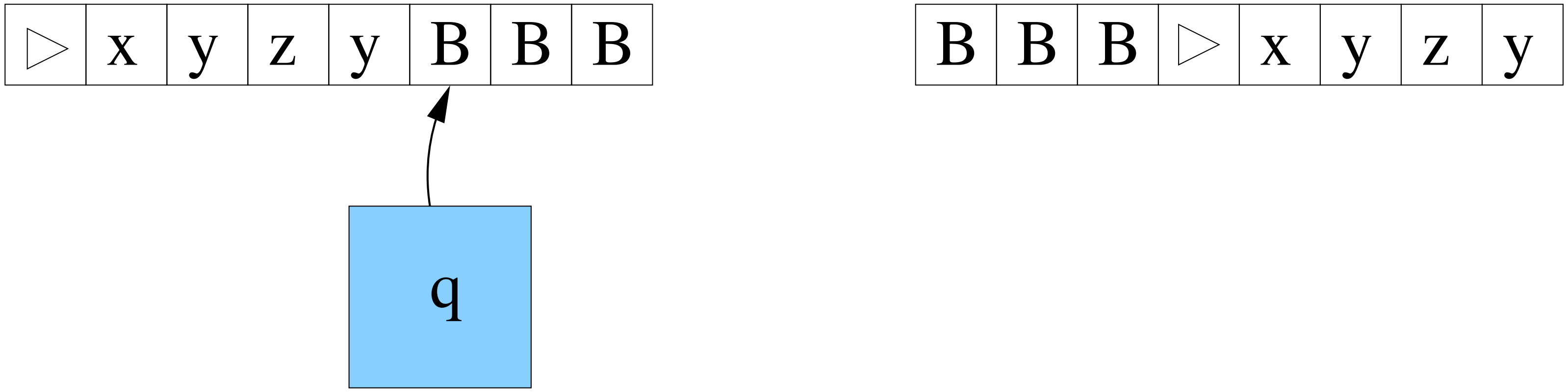
Simulate transition $(p, x, q, y, \rightarrow)$ of M.

Simulation Phase



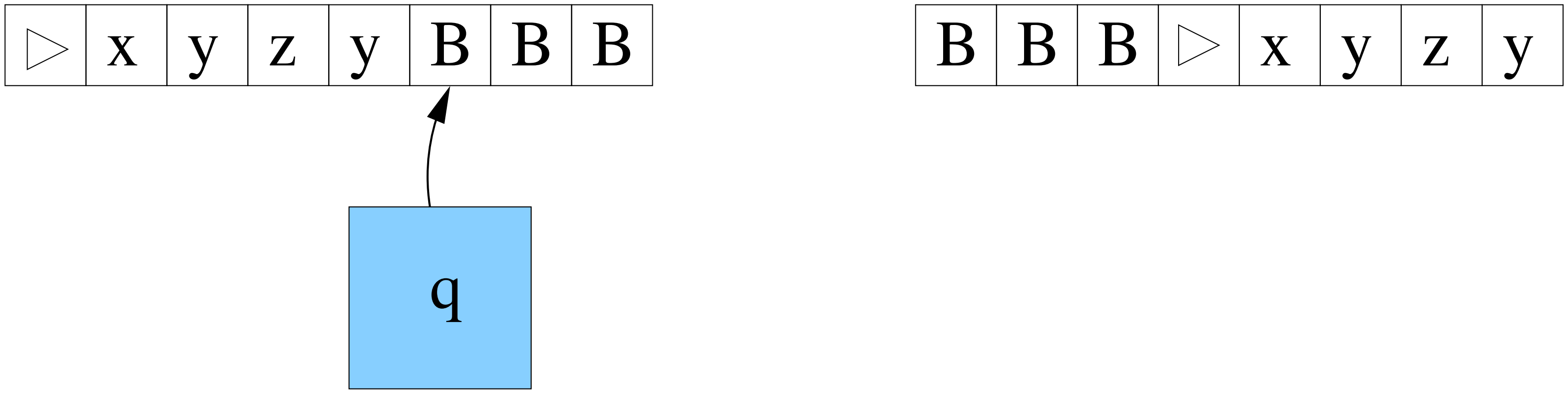
- Simulate transition $(p, x, q, y, \rightarrow)$ of M.
  - Read $x_q \;\Rightarrow\;$ Write y

Simulation Phase



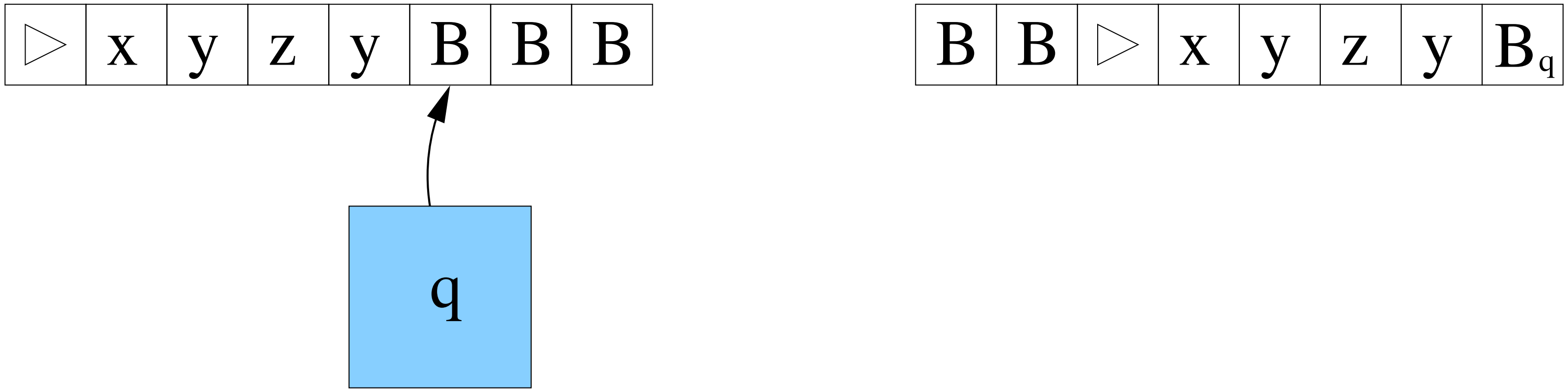
- Simulate transition $(p, x, q, y, \rightarrow)$ of M.
  - Read x_q $\Rightarrow$ Write y

Simulation Phase



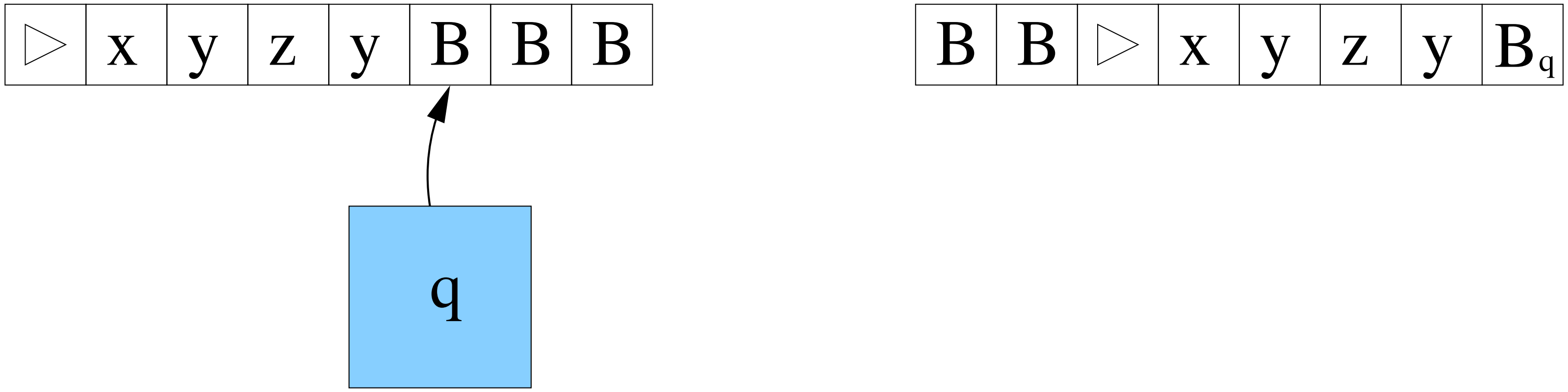
- Simulate transition $(p, x, q, y, \rightarrow)$ of M.
  - Read x_q $\Rightarrow$ Write y $\Rightarrow$ Read B $\Rightarrow$ Write B_q.

Simulation Phase



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

Simulation Phase

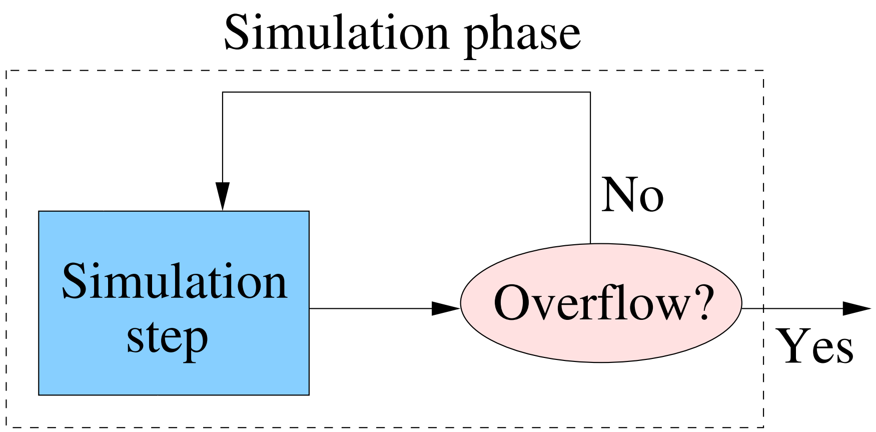


- Simulate transition $(p, x, q, y, \rightarrow)$ of M.
  - Read x_q $\Rightarrow$ Write y $\Rightarrow$ Read B $\Rightarrow$ Write B_q.
- Notice the alternation between read and write events.

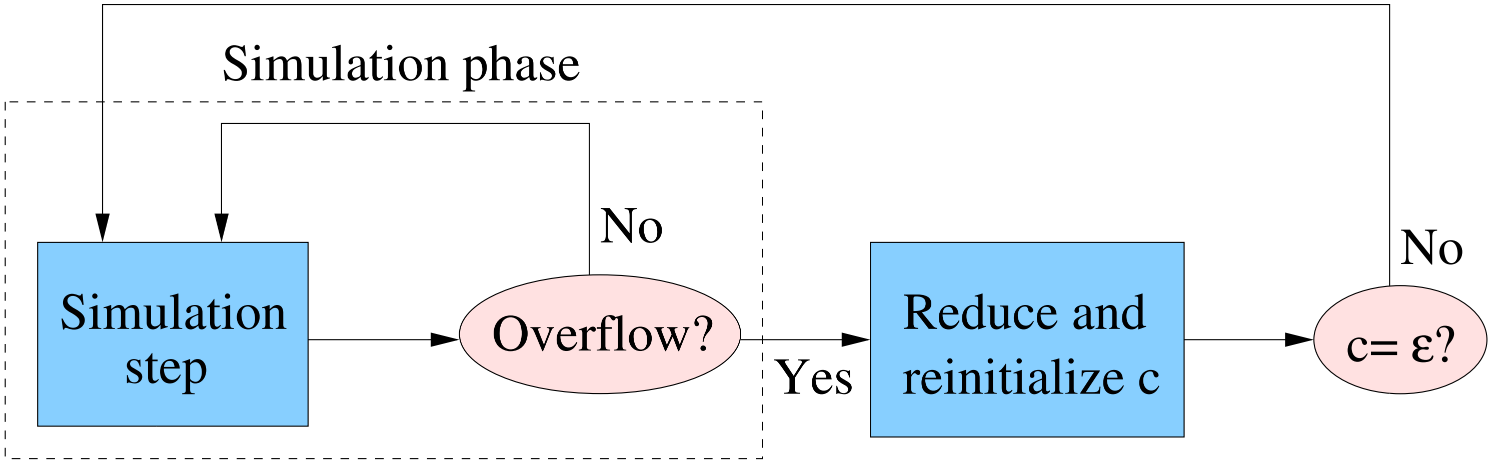
Simulation Phase

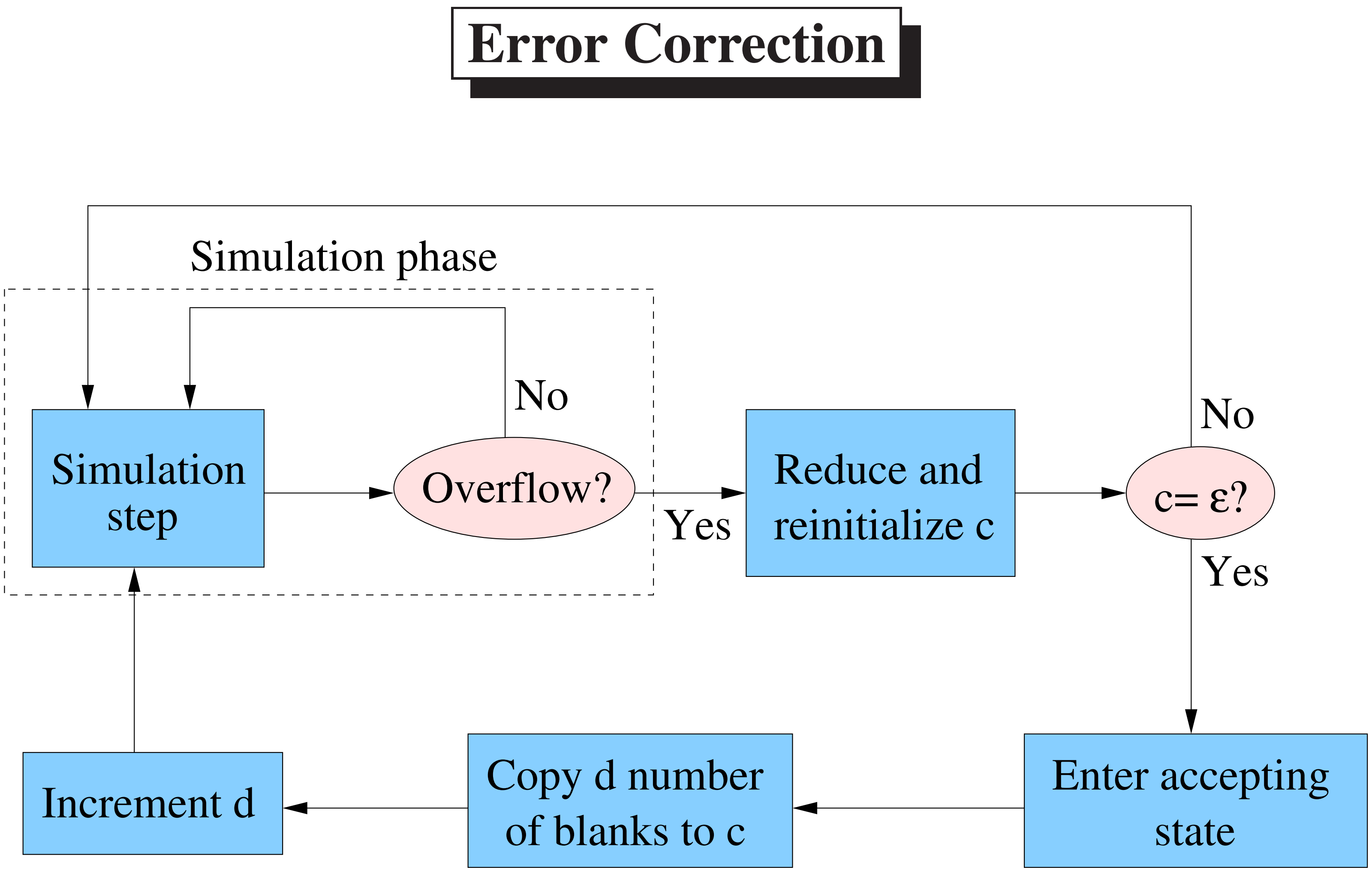
- Two problems:
  - Could run out of space during simulation phase.
  - Could also be computing gibberish!

Error Correction



Error Correction




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

Second Main Result

Lemma. The recurrent-state problem for channel machines with insertion errors and zero testing is undecidable.

Theorem. Both the satisfiability and model checking problems for MTL over infinite words are undecidable.

Our Contribution

	Satisfiability	Model Checking
MTL (Inf.)	Undec.	Undec.
MTL (Fin.)	Dec. / NPR	Dec. / NPR
Safety MTL (Inf.)	??	Dec. / ??

- Safety MTL: bounded timing constraints on all eventuality operators.

One Last Thing

	Satisfiability	Model Checking
MTL (Inf.)	Undec.	Undec.
MTL (Fin.)	Dec. / NPR	Dec. / NPR
Safety MTL (Inf.)	Dec. / NPR	Dec. / ??

- Safety MTL: bounded timing constraints on all eventuality operators.

Summary and Bibliography

- Decidability and complexity results for MTL:
  - Safety properties (e.g., invariance and time-bounded response) are decidable.
  - Liveness properties are undecidable.
- Bibliography:

  - J. Ouaknine and J. Worrell. On the Decidability of Metric Temporal Logic. *Proceedings of LICS'05, IEEE Computer Society Press*, 2005.
  - J. Ouaknine and J. Worrell. Metric Temporal Logic and Faulty Turing Machines. *In preparation.*