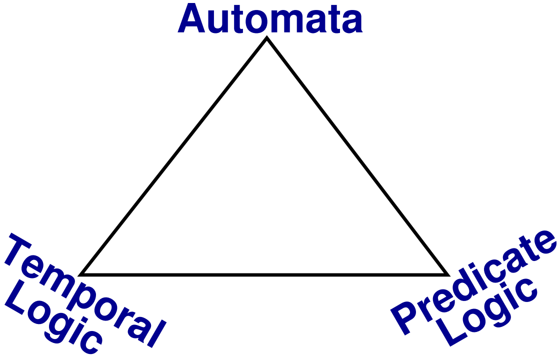
# Tutorial on Timed Systems Verification

James Worrell

Oxford University Computing Laboratory

MoVeP, July 2010

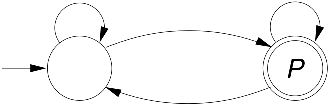
# The Classical Theory of Verification




- Qualitative (order-theoretic), rather than quantitative (metric).
- Time is modelled as the naturals $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.
- Note: focus on linear time (as opposed to branching time).

# A Simple Example

*'P occurs infinitely often'*




$$\Box\Diamond P$$
$$\forall x\, \exists y\, (x < y \wedge P(y))$$

# Specification and Verification

Assume the system is modelled by an automaton $M$.
The specification can be given by:

- A Linear Temporal Logic (LTL) formula $\theta$.

  $$\theta ::= P \mid \theta_1 \wedge \theta_2 \mid \theta_1 \vee \theta_2 \mid \neg\theta \mid \theta_1 \,\mathcal{U}\, \theta_2 \mid \theta_1 \,\mathcal{S}\, \theta_2$$

  For example, $\Box(REQ \rightarrow \Diamond ACK)$.

  Verification is then model checking: $M \models \theta$ ?

- A First-Order Logic (FO($<$)) formula $\varphi$.

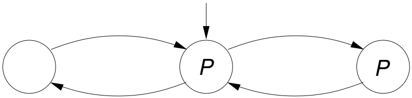
  $$\varphi ::= x < y \mid P(x) \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \forall x\, \varphi \mid \exists\, x \varphi$$

  For example, $\forall x\, (REQ(x) \rightarrow \exists y\, (x < y \wedge ACK(y)))$.

  Verification is again model checking: $M \models \varphi$ ?

# Another Example

*'P holds at every even position*
*(and may or may not hold at odd positions)'*




- It turns out it is impossible to capture this requirement using LTL or FO($<$).
- LTL and FO($<$) can however capture the specification: *'Q holds precisely at even positions'*:

$$Q \wedge \Box(Q \rightarrow \bigcirc\neg Q) \wedge \Box(\neg Q \rightarrow \bigcirc Q)$$

- So one way to capture the original specification would be to write: *'Q holds precisely at even positions* **and** $\Box(Q \rightarrow P)$*'*.
- Finally, need to existentially quantify $Q$ out:

$$\exists Q\ (Q \textit{ holds precisely at even positions } \textbf{and } \Box(Q \rightarrow P))$$

# More Specification and Verification

Monadic Second-Order Logic (MSO($<$)):

$$\varphi ::= x < y \mid P(x) \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \forall x\, \varphi \mid \exists x\, \varphi \mid \forall P\, \varphi \mid \exists P\, \varphi$$

## Theorem (Büchi 1960)

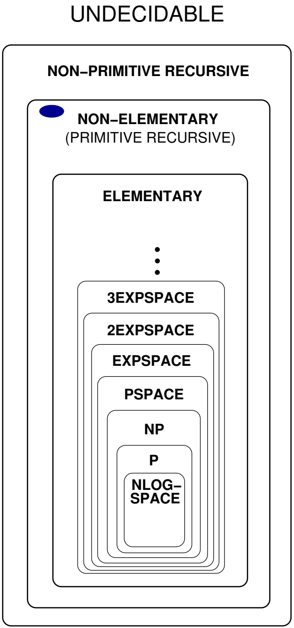
*Any MSO($<$) formula $\varphi$ can be effectively translated into an equivalent automaton $A_\varphi$.*

## Corollary (Church 1960)

*The model-checking problem for automata against MSO($<$) specifications is decidable:*

$$M \models \varphi \quad \textit{iff} \quad L(M) \cap L(A_{\neg\varphi}) = \emptyset$$

# Algorithmic Complexity




- Most problems in Computer Science sit within PSPACE.
- Hierarchy extends much beyond:
  - EXPSPACE: $2^{p(n)}$
  - 2EXPSPACE: $2^{2^{p(n)}}$
  - 3EXPSPACE: $2^{2^{2^{p(n)}}}$
  - $\ldots$
  - ELEMENTARY: $\bigcup_{k \in \mathbb{N}} \{k\text{EXPSPACE}\}$
  - NON-ELEMENTARY: $\underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^{n}}}}}}}_{n}$
  - NON-PRIMITIVE RECURSIVE:

    Ackerman: 3, 4, 8, 2048, $\underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}}_{2048}$, $\ldots$

# Complexity and Equivalence

In fact:

Theorem (Stockmeyer 1974)

*FO($<$) satisfiability has non-elementary complexity.*

Theorem (Kamp 1968;
Gabbay, Pnueli, Shelah, Stavi 1980)

*LTL and FO($<$) have precisely the same expressive power.*

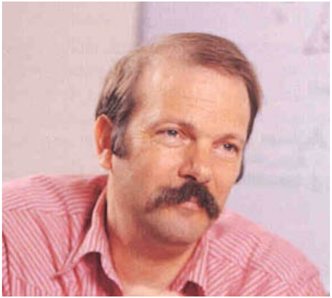
But amazingly:

Theorem (Sistla & Clarke 1982)

*LTL satisfiability and model checking are PSPACE-complete.*

# Logics and Automata

*"The paradigmatic idea of the automata-theoretic approach to verification is that we can compile high-level logical specifications into an equivalent low-level finite-state formalism."*

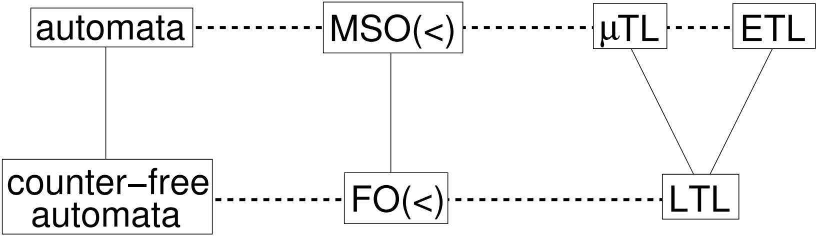


Moshe Vardi

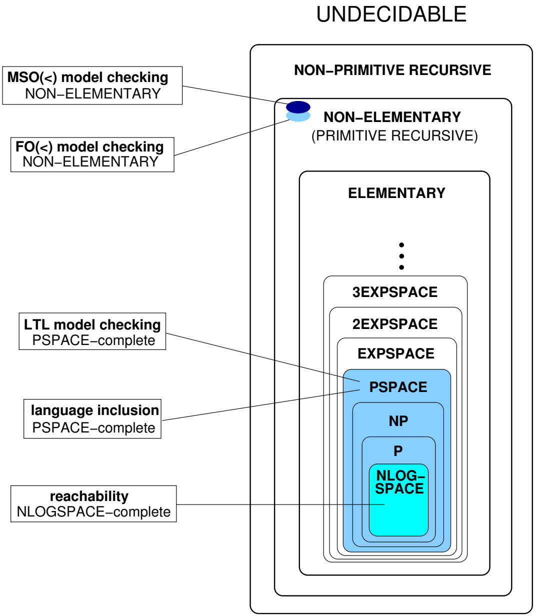
**Theorem**

*Automata are closed under all Boolean operations. Moreover, the language inclusion problem ( $L(A) \subseteq L(B)$ ?) is PSPACE-complete.*

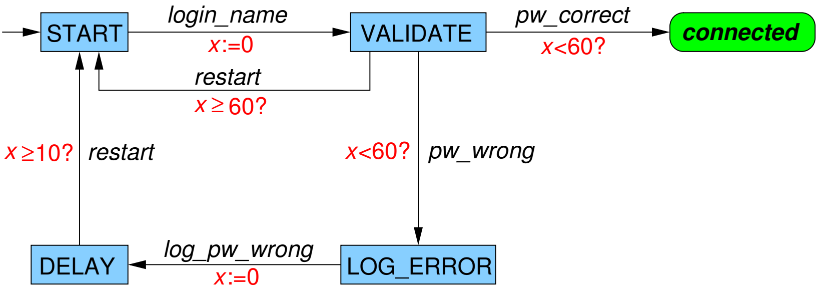
# The Classical Theory: Expressiveness



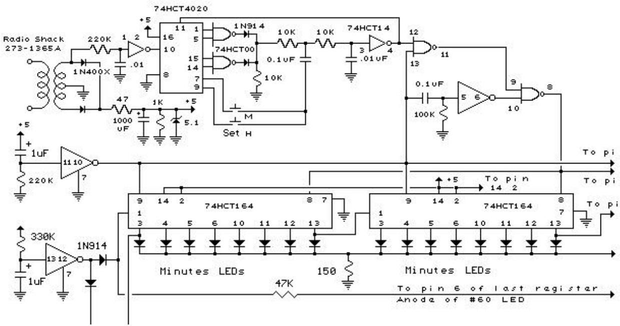
# The Classical Theory: Complexity



# A Login Protocol

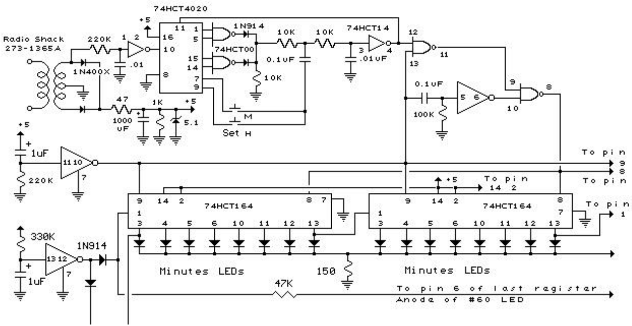



SPECIFICATION: $\square(\textit{pw\_wrong} \longrightarrow \square_{[0,10)} \neg \textit{restart})$




[. . . ] When power is applied, a single '1' bit is loaded into the first stage of both the minutes and hours registers. To accomplish this, a momentary low reset signal is sent to all the registers (at pin 9) and also a NAND gate to lock out any clock transitions at pin 8 of the minutes registers. At the same time, a high level is applied to the data input lines of both minutes and hours registers at pin 1. A single positive going clock pulse is generated at the end of the reset signal which loads a high level into the first stage of the minutes register. The rising edge of first stage output at pin 3 advances the hours and a single bit is also loaded into the hours register. Power should remain off for 3 seconds before being re-applied to allow the filter and timing capacitors to discharge. [. . . ]

(Bill Bowden, `www.circuitdb.com/circuits/id/98`)




[. . . ] When power is applied, a single ‘1’ bit is loaded into the first stage of both the minutes and hours registers. To accomplish this, a momentary low reset signal is sent to all the registers (at pin 9) and also a NAND gate to lock out any clock transitions at pin 8 of the minutes registers. At the same time, a high level is applied to the data input lines of both minutes and hours registers at pin 1. A single positive going clock pulse is generated at the end of the reset signal which loads a high level into the first stage of the minutes register. The rising edge of first stage output at pin 3 advances the hours and a single bit is also loaded into the hours register. Power should remain off for 3 seconds before being re-applied to allow the filter and timing capacitors to discharge. [. . . ]

(Bill Bowden, `www.circuitdb.com/circuits/id/98`)

# Timed Systems

Timed systems occur in:

- Hardware circuits
- Communication protocols
- Cell phones
- Plant controllers
- Aircraft navigation systems
- ...

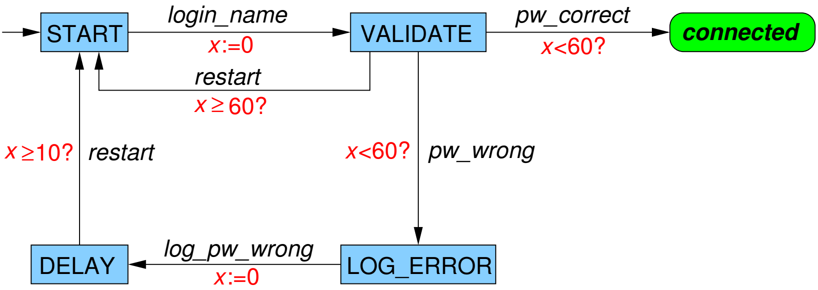
In many instances, it is **crucial** to accurately model the timed behaviour of the system.

# From Qualitative to Quantitative

*"Lift the classical theory*
*to the real-time world."*

Boris Trakhtenbrot, LICS 1995

# Timed Automata



# Timed Automata

Timed automata were introduced by Rajeev Alur at Stanford during his PhD thesis under David Dill:

- Rajeev Alur, David L. Dill: *Automata For Modeling Real-Time Systems*. ICALP 1990: 322-335
- Rajeev Alur, David L. Dill: *A Theory of Timed Automata*. TCS 126(2): 183-235, 1994

# 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 Automata

Timed automata are language acceptors for timed words

Theorem (Alur, Courcourbetis, Dill 1990)

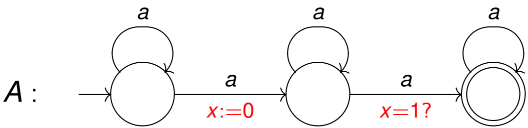
*Reachability is decidable, in fact PSPACE-complete.*

Unfortunately:

Theorem (Alur & Dill 1990)

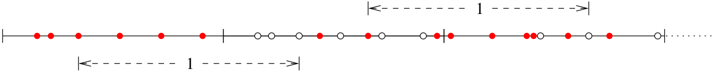
*Language inclusion is undecidable for timed automata.*

# An Uncomplementable Timed Automaton




*L(A)*:


$\overline{L(A)}$:




*A* cannot be complemented:
There is no timed automaton *B* with $L(B) = \overline{L(A)}$.

# Metric Temporal Logic

Metric Temporal Logic (MTL) [Koymans; de Roever; Pnueli $\sim$1990] is a central quantitative specification formalism for timed systems.

- MTL = LTL + timing constraints on operators:

$$\Box(\boxminus_{[0,1]} \mathit{PEDAL} \rightarrow \Diamond_{[5,10]} \mathit{BRAKE})$$

- Widely cited and used (over seven hundred papers according to `scholar.google.com`!).

Unfortunately:

**Theorem (Alur & Henzinger 1992)**

*MTL satisfiability and model checking are undecidable over $\mathbb{R}_{\geq 0}$. (Decidable but non-primitive recursive under certain semantic restrictions [Ouaknine & Worrell 2005].)*

# Metric Predicate Logic

The first-order metric logic of order (FO($<$,$+1$)) extends FO($<$) by the unary function '$+1$'.
For example, $\Box(PEDAL \rightarrow \Diamond_{[5,10]}\, BRAKE)$ becomes

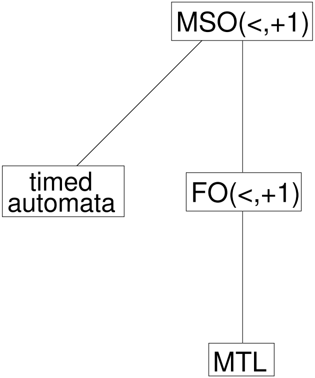
$$(PEDAL(x) \rightarrow \exists y\, (x + 5 \leq y \leq x + 10 \wedge BRAKE(y)))$$

## Theorem (Hirshfeld & Rabinovich 2007)

*FO($<$,$+1$) is strictly more expressive than MTL over $\mathbb{R}_{\geq 0}$.*

Corollary: FO($<$,$+1$) and MSO($<$,$+1$) satisfiability and model checking are undecidable over $\mathbb{R}_{\geq 0}$.

# The Real-Time Theory: Expressiveness



# Key Stumbling Blocks

**Theorem (Alur & Dill 1990)**

*Language inclusion is undecidable for timed automata.*

**Theorem (Hirshfeld & Rabinovich 2007)**

*FO($<$,$+1$) is strictly more expressive than MTL over $\mathbb{R}_{\geq 0}$.*

# Part II: Negative Results

# Undecidability

Theorem (Alur & Dill 1990)

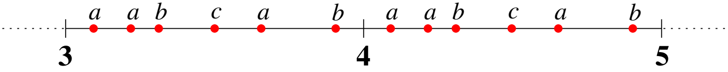
*Language inclusion is undecidable for timed automata.*

**Proof.**

- Encode halting computations of two-counter machine $M$ as timed language $L(M)$.
- Define timed automaton $A$ accepting the **complement** of $L(M)$.
- $A$ is universal if and only if $L(M)$ has no halting computation.

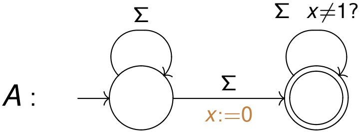
# Undecidability

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



# Undecidability

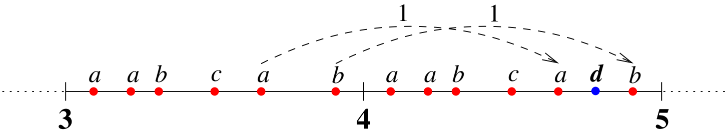
To correctly propagate the tape contents we require that every event in the current time interval have a matching event one time unit later.

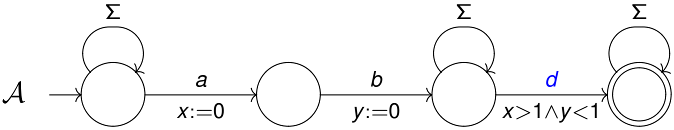



*A* accepts all timed words that *violate* this property

# Backward Propagation

This not sufficient: we have only enforced *forward* propagation of events.



# Observations

The undecidability proof required

- Dense Time
- Infinite Precision
- Two clocks
- Timed words of unbounded duration

# Inexpressiveness

Theorem (Hirshfeld & Rabinovich 2007)

*FO($<$,$+1$) is strictly more expressive than any temporal logic with finitely many modalities definable in FO($<$,$+1$) over $\mathbb{R}$.*

Build your own temporal logic:

- $(X\,\mathcal{U}\,Y)(t) \equiv \exists s > t\,(Y(s) \wedge \forall u\,(t < u < s \rightarrow X(u)))$
- $(X\,\mathcal{S}\,Y)(t) \equiv \exists s < t\,(Y(s) \wedge \forall u\,(s < u < t \rightarrow X(u)))$
- $C_n(X)(t) \equiv \exists x_1 \cdots \exists x_n$
  $(t < x_1 < \cdots < x_n < t + 1 \wedge X(x_1) \wedge \cdots \wedge X(x_n))$

# Inexpressiveness

Theorem (Hirshfeld & Rabinovich 2007)

*Let TL be a temporal logic with* ***finitely many modalities*** *definable in FO($<$,$+1$). Then TL is strictly less expressive than FO($<$,$+1$).*

- One free predicate variable $P$.
- Four **simple formulas** $P(t)$, $\neg P(t)$, True and False.
- Model $\mathcal{M}_k$ interprets $P$ as $\mathbb{N}/k$.
- In $\mathcal{M}_k$ every formula $\varphi(t)$ of FO($<$,$+1$) is equivalent to a simple formula.

# Inexpressiveness

- TL-modality $O(X_1, \ldots, X_n)$ interpreted by FO($<$,$+1$)-formula $\psi(X_1, \ldots, X_n, t)$.
- Semantics of $\psi$ in $\mathcal{M}_k$ defined by **truth table.**

| $X_1$ | $\cdots$ | $X_n$ | $\psi$ |
|---|---|---|---|
| $P$ | $\cdots$ | True | $\neg P$ |

- There exists $k \neq \ell$ such that any TL-formula is equivalent to the same simple formula on both $\mathcal{M}_k$ and $\mathcal{M}_\ell$.
- But $C_n$ distinguishes $\mathcal{M}_k$ from $\mathcal{M}_\ell$ for some $n$.

# Part II: One-Clock Automata

# Mind the Gap

Timed automata language inclusion: $L(B) \stackrel{?}{\subseteq} L(A)$

- $A$ has *no* clocks: PSPACE-Complete [Alur *et al*. 90]

- $A$ has *two* clocks: Undecidable [Alur, Dill 94]

## Mind the Gap

Timed automata language inclusion: $L(B) \stackrel{?}{\subseteq} L(A)$

- $A$ has *no* clocks: PSPACE-Complete [Alur *et al*. 90]
- $A$ has *one* clock: Decidable — in fact Non-Primitive Recursive
- $A$ has *two* clocks: Undecidable [Alur, Dill 94]

# Mind the Gap

Timed automata language inclusion: $L(B) \overset{?}{\subseteq} L(A)$

- $A$ has *no* clocks: PSPACE-Complete [Alur *et al.* 90]
- $A$ has *one* clock: Decidable — in fact Non-Primitive Recursive
- $A$ has *two* clocks: Undecidable [Alur, Dill 94]

This result is somewhat surprising: in most computational structures, deciding language inclusion normally uses:

$$L(B) \subseteq L(A) \iff L(B) \cap \overline{L(A)} = \emptyset$$

However, one-clock timed automata cannot be complemented . . .

## Some Applications

- Hardware and software systems are often described via high-level **functional specifications**, describing their intended global behavior.
- Functional specifications are often given as **finite-state machines**. A proposed implementation *IMP* meets its specification *SPEC* iff

$$L(IMP) \subseteq L(SPEC).$$

**Finite-state machines** are often used as **specifications** of systems:

$$\textit{IMP} \text{ meets } \textit{SPEC} \quad \text{iff} \quad L(IMP) \subseteq L(SPEC)$$

- Our work enables us to handle **timed** functional specifications: **timed automata with a single clock**.
- (Further potential applications to verification described later on.)

# Sketch of the Algorithm

- Reduce the language inclusion question $L(B) \stackrel{?}{\subseteq} L(A)$ to a **reachability** question on an infinite graph $\mathcal{H}$.

# Sketch of the Algorithm

- Reduce the language inclusion question $L(B) \stackrel{?}{\subseteq} L(A)$ to a **reachability** question on an infinite graph $\mathcal{H}$.
- Construct a compatible **well-quasi-order** $\preccurlyeq$ on $\mathcal{H}$:
  - Whenever $W \preccurlyeq W'$: if $W$ is safe, then $W'$ is safe.
  - Any infinite sequence $W_1, W_2, W_3, \ldots$ eventually saturates: there exists $i < j$ such that $W_i \preccurlyeq W_j$.

# Sketch of the Algorithm

- Reduce the language inclusion question $L(B) \stackrel{?}{\subseteq} L(A)$ to a **reachability** question on an infinite graph $\mathcal{H}$.
- Construct a compatible **well-quasi-order** $\preccurlyeq$ on $\mathcal{H}$:
  - Whenever $W \preccurlyeq W'$: if $W$ is safe, then $W'$ is safe.
  - Any infinite sequence $W_1, W_2, W_3, \ldots$ eventually saturates: there exists $i < j$ such that $W_i \preccurlyeq W_j$.
- Explore $\mathcal{H}$, looking for unsafe nodes. The search must eventually terminate.

# Sketch of the Algorithm

- Reduce the language inclusion question $L(B) \stackrel{?}{\subseteq} L(A)$ to a **reachability** question on an infinite graph $\mathcal{H}$.
- Construct a compatible **well-quasi-order** $\preccurlyeq$ on $\mathcal{H}$:
  - Whenever $W \preccurlyeq W'$: if $W$ is safe, then $W'$ is safe.
  - Any infinite sequence $W_1, W_2, W_3, \ldots$ eventually saturates: there exists $i < j$ such that $W_i \preccurlyeq W_j$.
- Explore $\mathcal{H}$, looking for unsafe nodes. The search must eventually terminate.
- For simplicity, we focus on **universality**: $L(A) \stackrel{?}{=} \mathbf{TT}$

# Higman's Lemma

Let $\Lambda = \{a_1, a_2, \ldots, a_n\}$ be an alphabet.
Let $\preccurlyeq$ be the **subword order** on $\Lambda^*$, the set of finite words over $\Lambda$.

Ex.: HIGMAN $\preccurlyeq$ HIGHMOUNTAIN

Then $\preccurlyeq$ is a **well-quasi-order** on $\Lambda^*$:

Any infinite sequence of words $W_1$, $W_2$, $W_3$, $\ldots$ must eventually have two words $W_i \preccurlyeq W_j$, with $i < j$.

---

G. Higman, "Ordering by divisibility in abstract algebras."
*Proceedings of the London Mathematical Society*, vol. 2, 1952.

# Higman's Lemma

Let $\Lambda = \{a_1, a_2, \ldots, a_n\}$ be an alphabet.

Let $\preccurlyeq$ be the **subword order** on $\Lambda^*$, the set of finite words over $\Lambda$.

Ex.: HIGMAN $\preccurlyeq$ HIGHMOUNTAIN

Then $\preccurlyeq$ is a **well-quasi-order** on $\Lambda^*$:

Any infinite sequence of words $W_1, W_2, W_3, \ldots$ must eventually have two words $W_i \preccurlyeq W_j$, with $i < j$.

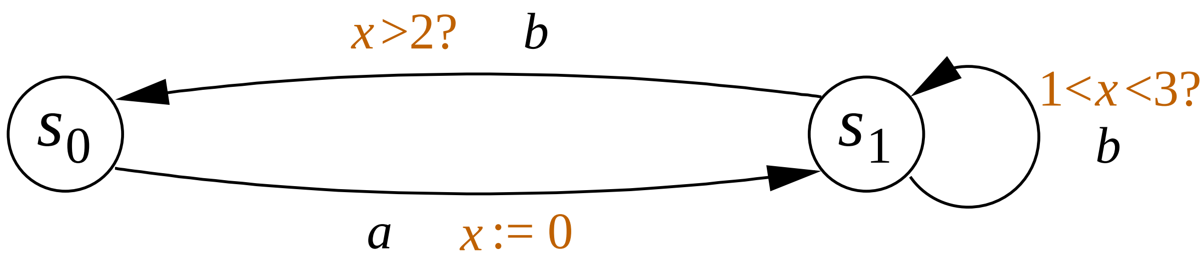
---

G. Higman, "Ordering by divisibility in abstract algebras."
*Proceedings of the London Mathematical Society*, vol. 2, 1952.

# Timed Automata Configurations

Let $A$ be a timed automaton with a single clock $x$,
and discrete **locations** $s_0, s_1, \ldots, s_n$.

- A **state** of $A$ is a pair $(s, v)$:
  - $s$ is a location.
  - $v \in \mathbb{R}^+$ is the value of clock $x$.
- A **configuration** of $A$ is a finite set of states.

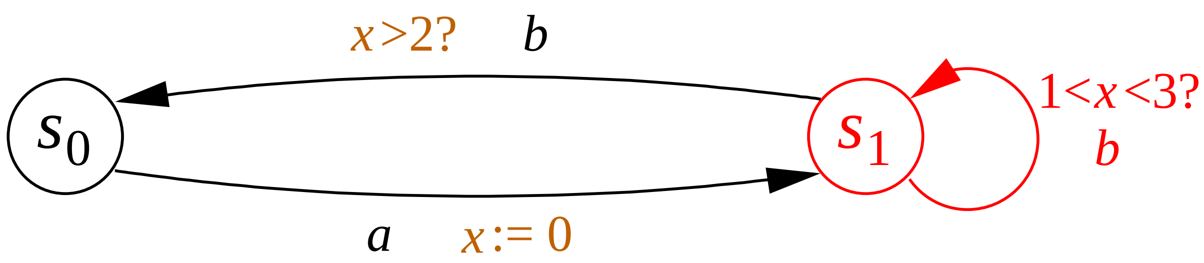


# Timed Automata Configurations

Let $A$ be a timed automaton with a single clock $x$, and discrete **locations** $s_0, s_1, \ldots, s_n$.

- A **state** of $A$ is a pair $(s, v)$:
  - $s$ is a location.
  - $v \in \mathbb{R}^+$ is the value of clock $x$.
- A **configuration** of $A$ is a finite set of states.

$(s_1, 1.4)$

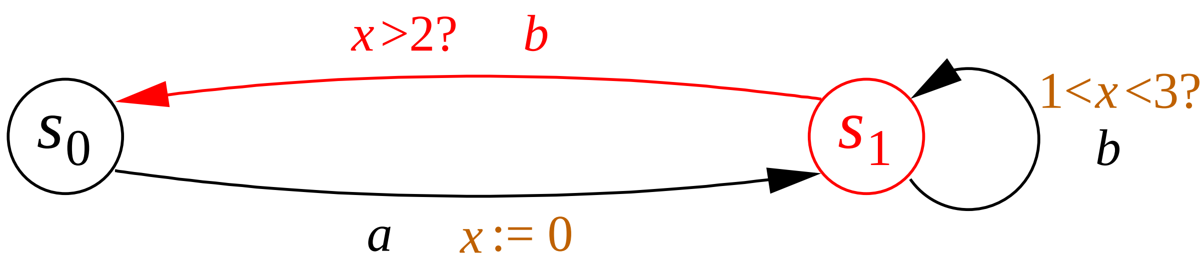


# Timed Automata Configurations

Let $A$ be a timed automaton with a single clock $x$,
and discrete **locations** $s_0$, $s_1$, $\ldots$, $s_n$.

- A **state** of $A$ is a pair $(s, v)$:
  - $s$ is a location.
  - $v \in \mathbb{R}^+$ is the value of clock $x$.
- A **configuration** of $A$ is a finite set of states.

$(s_1, 4.5)$

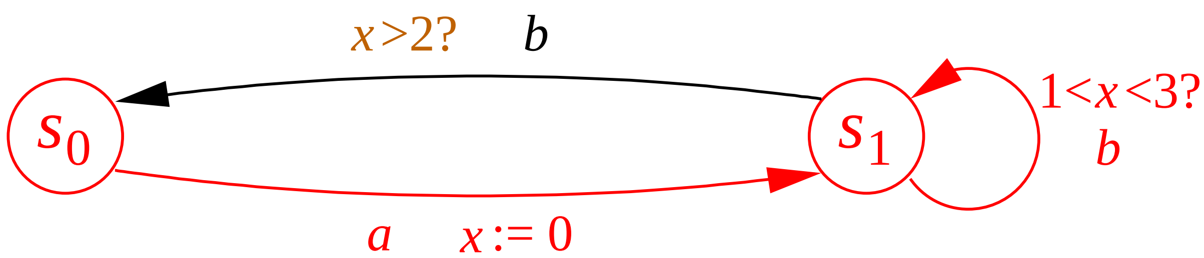


# Timed Automata Configurations

Let $A$ be a timed automaton with a single clock $x$, and discrete **locations** $s_0, s_1, \ldots, s_n$.

- A **state** of $A$ is a pair $(s, v)$:
  - $s$ is a location.
  - $v \in \mathbb{R}^+$ is the value of clock $x$.
- A **configuration** of $A$ is a finite set of states.

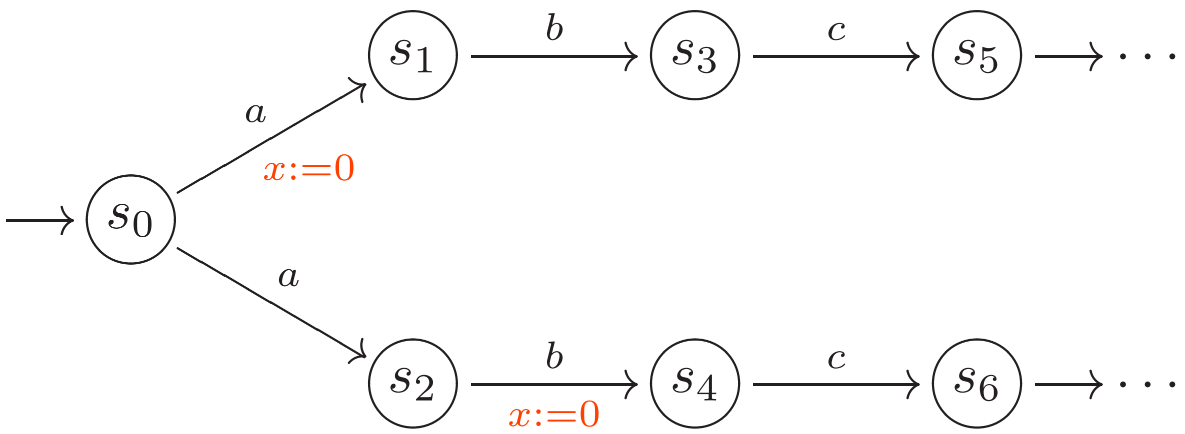
$\{(s_0, 2), (s_1, 1.4)\}$



# Timed Automata Configurations

Every timed trace $u$ gives rise to a configuration of $A$.

Ex.: $u = \langle 0.5, a, 0.2, b, 0.4, c \rangle$ leads to $\{(s_5, 0.6), (s_6, 0.4)\}$.



# Bisimilar Configurations

If $C$ is a configuration, let $A[C]$ be $A$ 'started' in configuration $C$.

**Definition.** A relation $\mathcal{R}$ on configurations is a **bisimulation** if, whenever $C_1 \; \mathcal{R} \; C_2$, then

- $\forall a \in \Sigma$, $\forall t_1 \in \mathbb{R}^+$, $\exists t_2 \in \mathbb{R}^+$ such that
  if $A[C_1] \xrightarrow{t_1, a} A[C_1']$, then $A[C_2] \xrightarrow{t_2, a} A[C_2']$, and $C_1' \; \mathcal{R} \; C_2'$.
- Vice-versa.

## Bisimilar Configurations

If $C$ is a configuration, let $A[C]$ be $A$ ‘started’ in configuration $C$.

**Definition.** A relation $\mathcal{R}$ on configurations is a **bisimulation** if, whenever $C_1 \; \mathcal{R} \; C_2$, then

- $\forall a \in \Sigma$, $\forall t_1 \in \mathbb{R}^+$, $\exists t_2 \in \mathbb{R}^+$ such that if $A[C_1] \xrightarrow{t_1, a} A[C_1']$, then $A[C_2] \xrightarrow{t_2, a} A[C_2']$, and $C_1' \; \mathcal{R} \; C_2'$.
- Vice-versa.

We say that $C_1$ and $C_2$ are **bisimilar**, written $C_1 \sim C_2$, if there exists some bisimulation relating them.

# Bisimilar Configurations

If $C$ is a configuration, let $A[C]$ be $A$ 'started' in configuration $C$.

**Definition.** A relation $\mathcal{R}$ on configurations is a **bisimulation** if, whenever $C_1 \; \mathcal{R} \; C_2$, then

- $\forall a \in \Sigma, \forall t_1 \in \mathbb{R}^+, \exists t_2 \in \mathbb{R}^+$ such that if $A[C_1] \xrightarrow{t_1, a} A[C_1']$, then $A[C_2] \xrightarrow{t_2, a} A[C_2']$, and $C_1' \; \mathcal{R} \; C_2'$.
- Vice-versa.

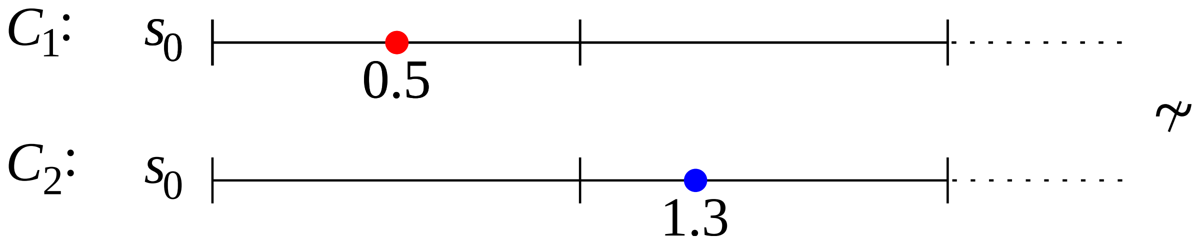
We say that $C_1$ and $C_2$ are **bisimilar**, written $C_1 \sim C_2$, if there exists some bisimulation relating them.

**Theorem.** If $C_1 \sim C_2$, then

$$A[C_1] \text{ is universal } \iff A[C_2] \text{ is universal.}$$

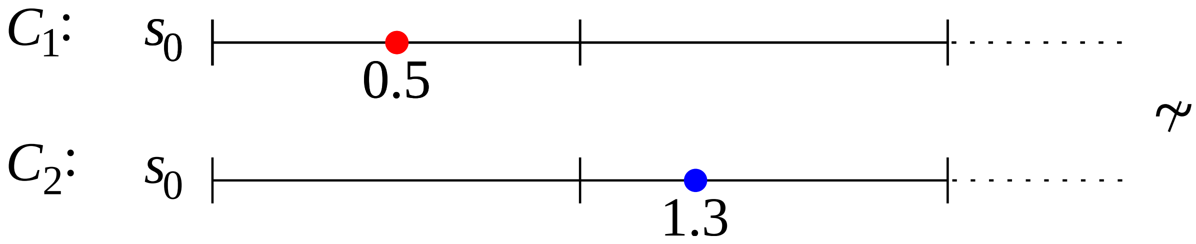
# Bisimilar Configurations: Examples

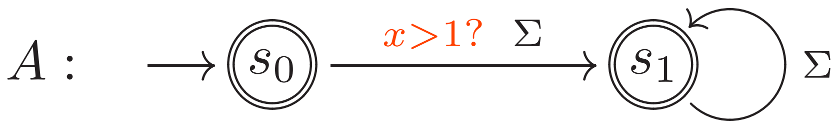
$$C_1 = \{(s_0, 0.5)\} \not\approx C_2 = \{(s_0, 1.3)\}.$$



# Bisimilar Configurations: Examples

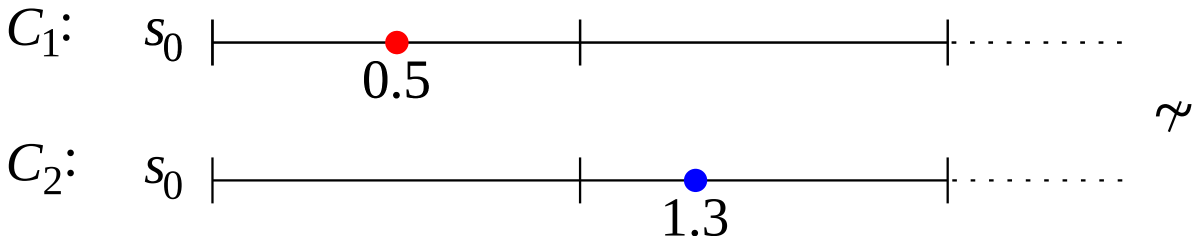
$$C_1 = \{(s_0, 0.5)\} \not\sim C_2 = \{(s_0, 1.3)\}.$$

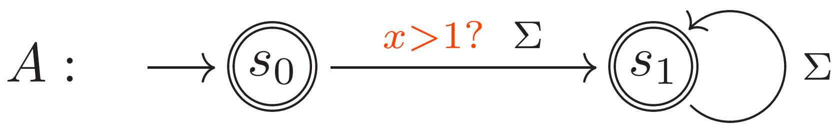


# Bisimilar Configurations: Examples

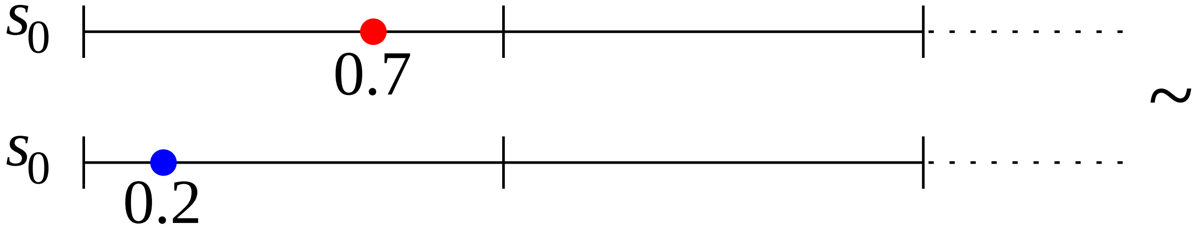
$$C_1 = \{(s_0, 0.5)\} \not\approx C_2 = \{(s_0, 1.3)\}.$$



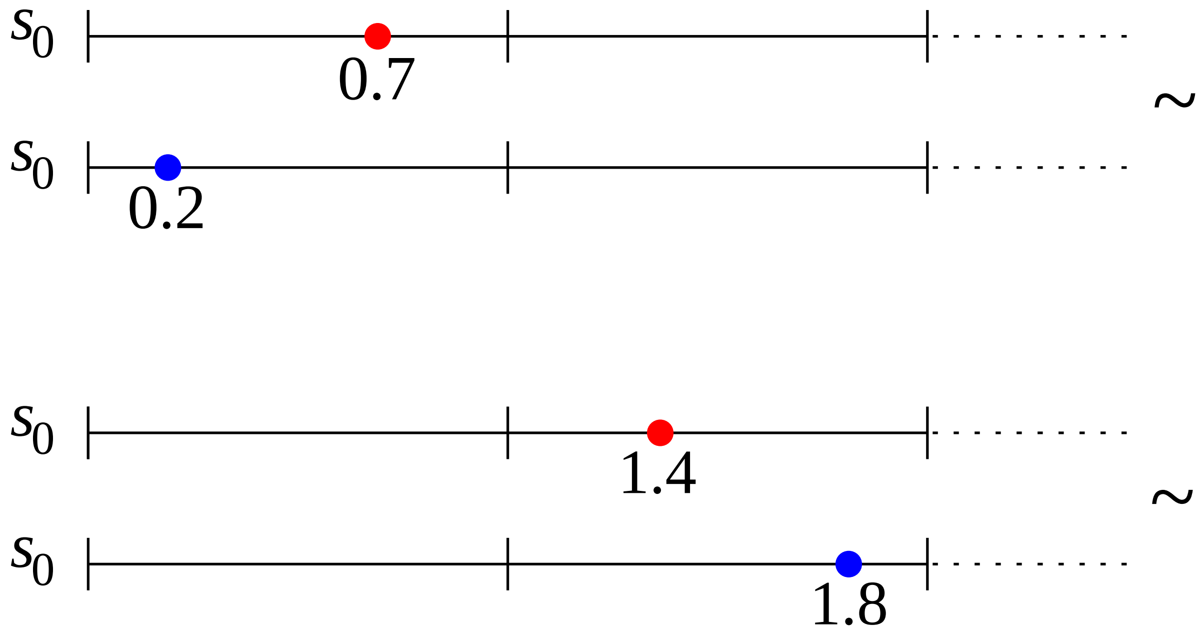




$A[C_2]$ is universal, but $A[C_1]$ rejects $\langle 0, a\rangle$.

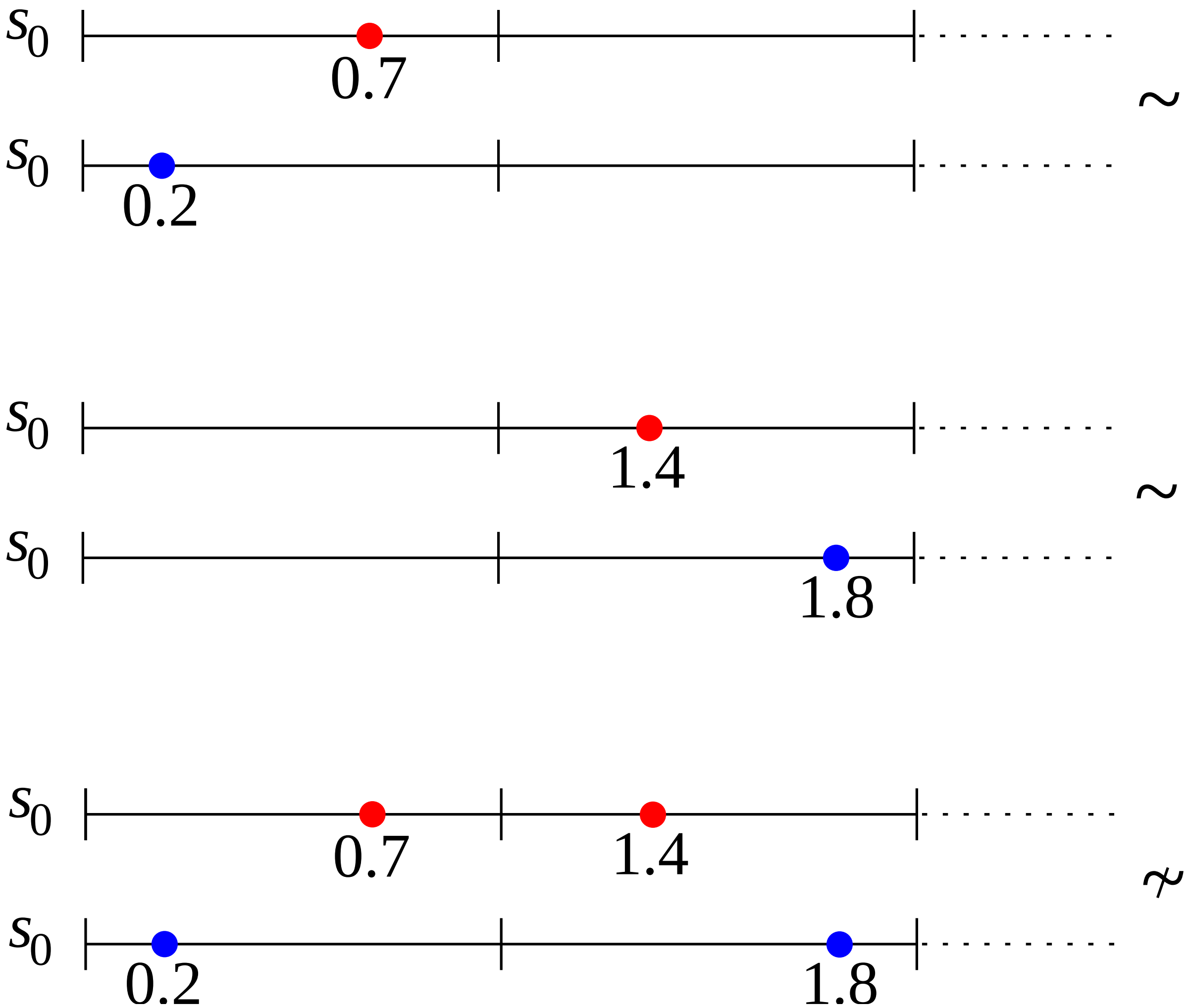
# Bisimilar Configurations: Examples



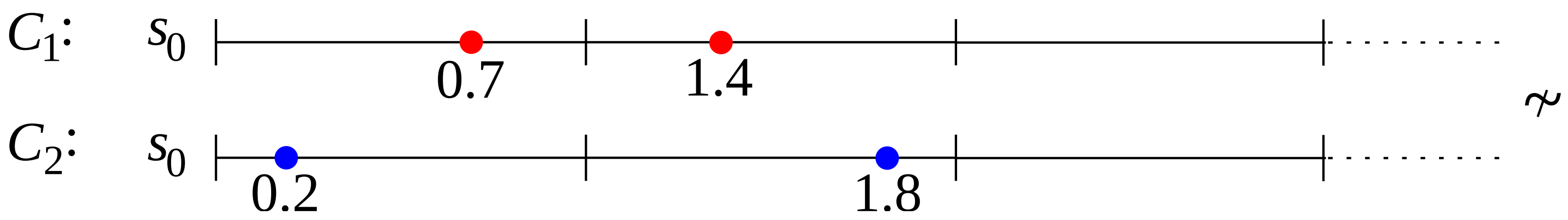
# Bisimilar Configurations: Examples



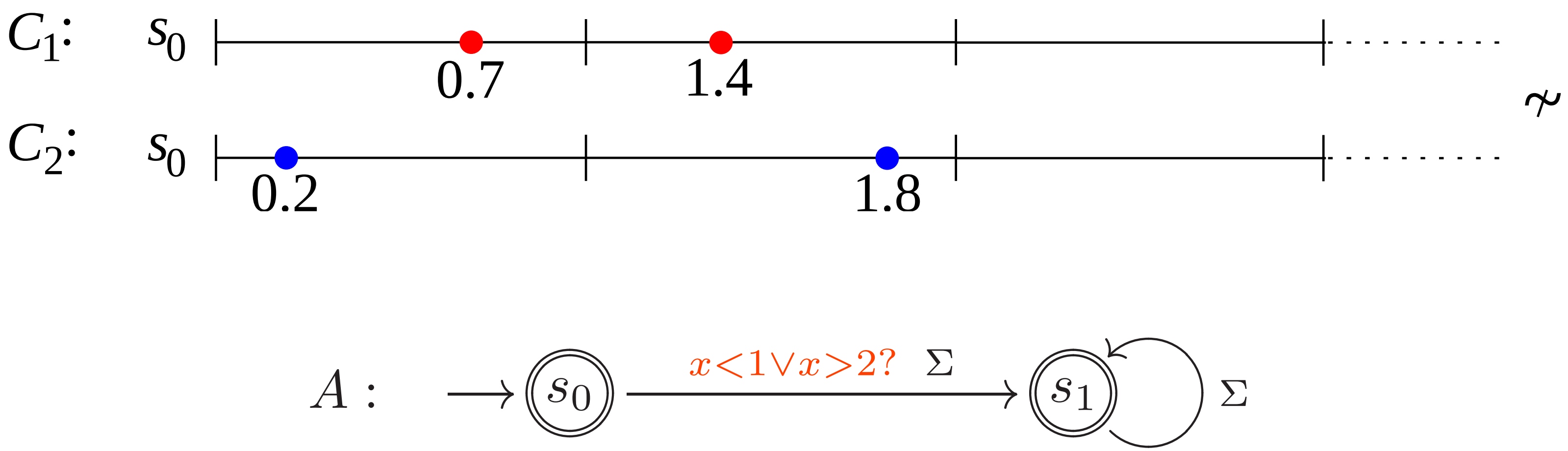
# Bisimilar Configurations: Examples



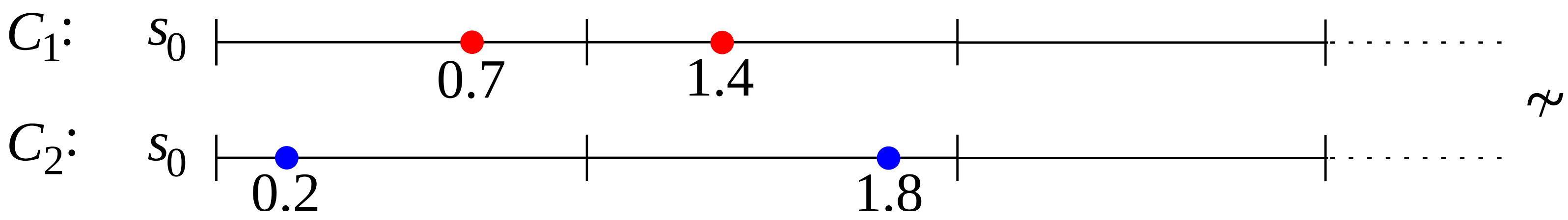
# Bisimilar Configurations: Examples

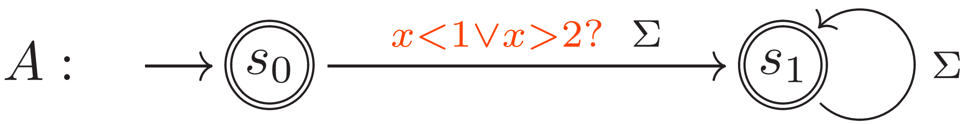


# Bisimilar Configurations: Examples



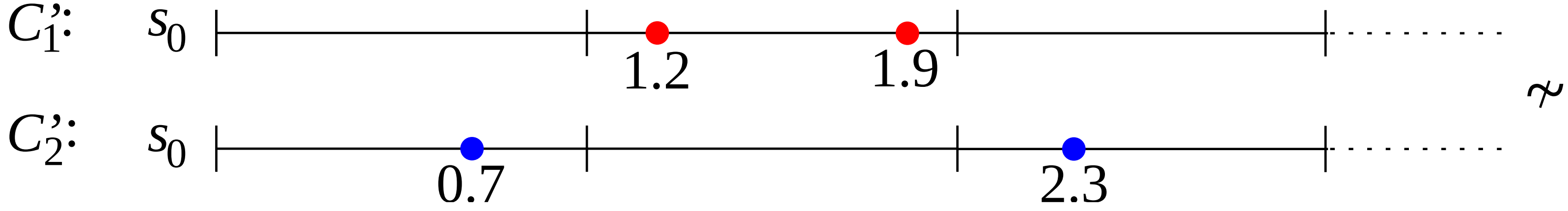
# Bisimilar Configurations: Examples

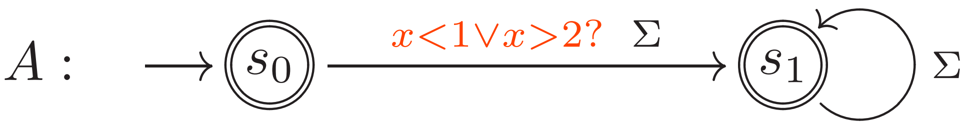



$A[C_2]$ is universal, but $A[C_1]$ rejects $\langle 0.5, a \rangle$.

# Bisimilar Configurations: Examples

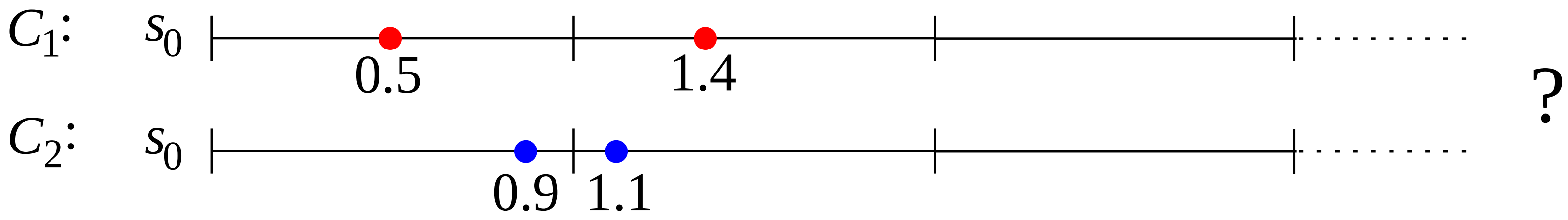



$A[C_2]$ is universal, but $A[C_1]$ rejects $\langle 0.5, a\rangle$.

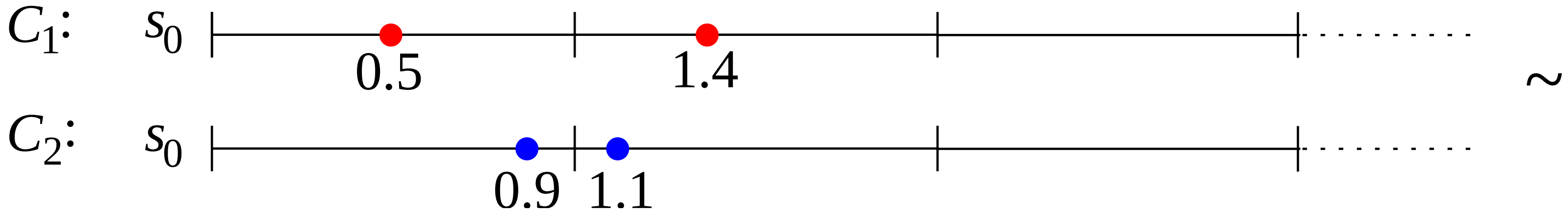
## Bisimilar Configurations: Examples

What about . . .



# Bisimilar Configurations: Examples

What about . . .




They **are** bisimilar: $C_1 \sim C_2$.

# Constructing a Decidable Bisimulation Relation

Let $K \in \mathbb{N}$ be the largest constant appearing in clock constraints of $A$.

**Theorem.** Let $C$ and $C'$ be configurations of $A$.
If there exists a bijection $f : C \to C'$ that preserves

- locations: $f(s, v) = (s', v') \implies s = s'$,
- integer parts of clock $x$, up to $K$:
  $f(s, v) = (s', v') \implies ((\lceil v \rceil = \lceil v' \rceil \wedge \lfloor v \rfloor = \lfloor v' \rfloor) \vee v, v' > K)$,
- the ordering of the fractional parts of clock $x$:
  $f(s_i, v_i) = (s'_i, v'_i) \implies (v_i < v_j \iff v'_i < v'_j)$,

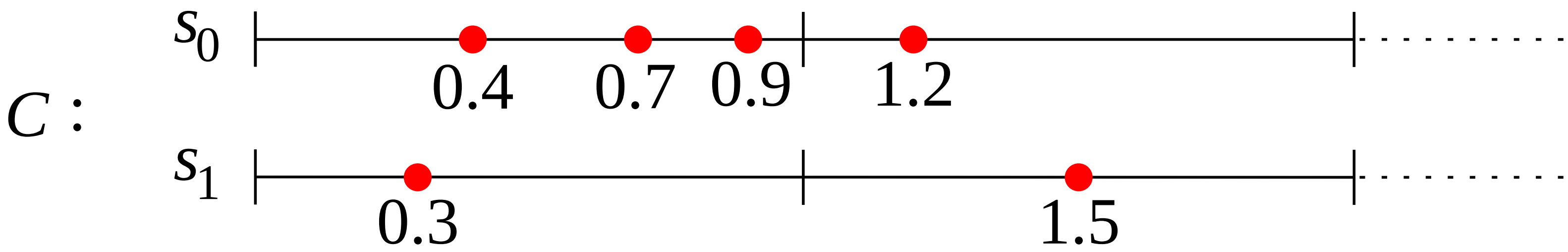
then $C \sim C'$.

# Constructing a Decidable Bisimulation Relation

- Let $K$ be the largest constant appearing in clock constraints of $A$.
- Let $REG = \Big\{\{0\}, (0,1), \{1\}, (1,2), \ldots, \{K\}, (K,\infty)\Big\}$ be the collection of 'one-dimensional regions' of $A$.
- Let $S = \{s_0, s_1, \ldots, s_n\}$ be the set of locations of $A$.
- Let $\Lambda = S \times REG$.
- Let $C$ be a configuration of $A$. For simplicity, assume all the fractional parts of states in $C$ are distinct.
- Note that each state in $C$ has a unique matching letter in $\Lambda$.
- Encode $C$ as a word $H(C) \in \Lambda^*$, ordered by increasing fractional parts of states.

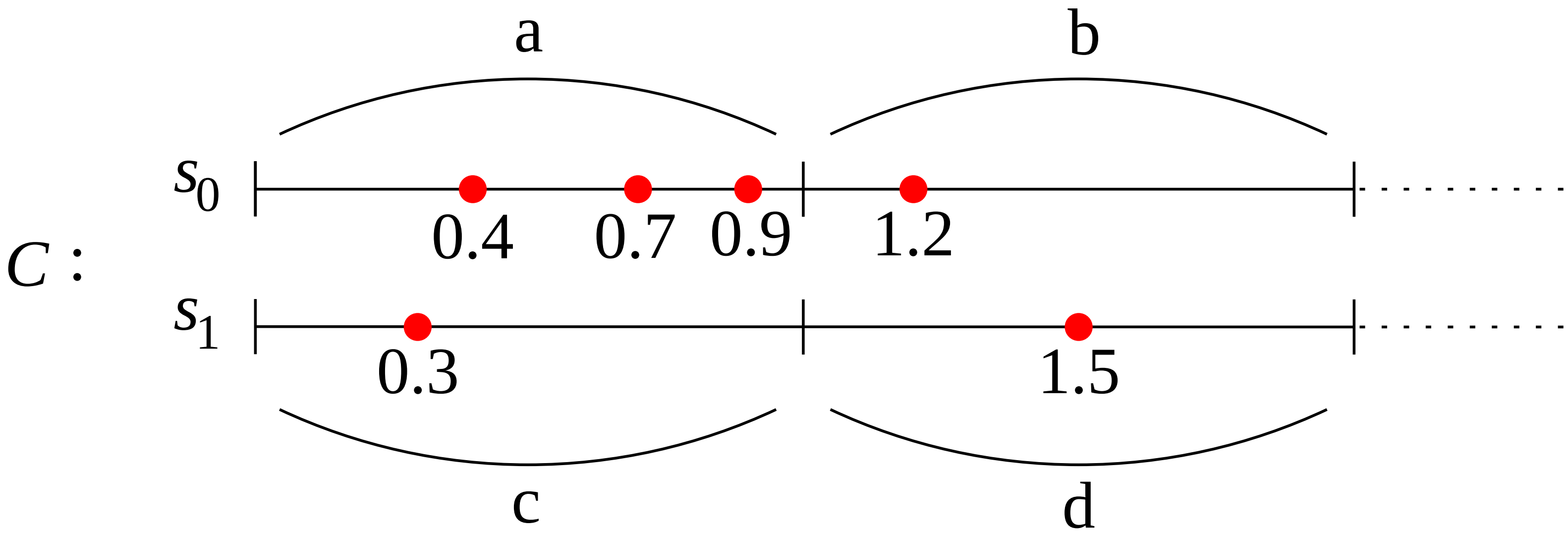
# Encoding Configurations as Words: Example

Consider the configuration $C$:



# Encoding Configurations as Words: Example

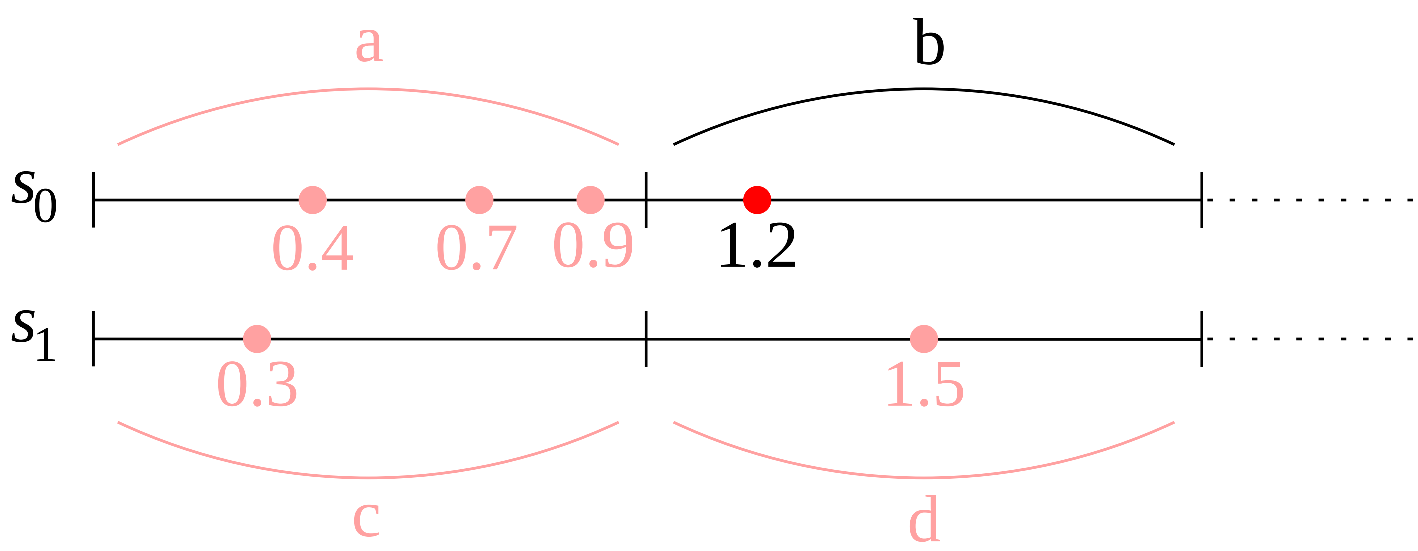
Consider the configuration $C$:



# Encoding Configurations as Words: Example

Consider the configuration $C$:

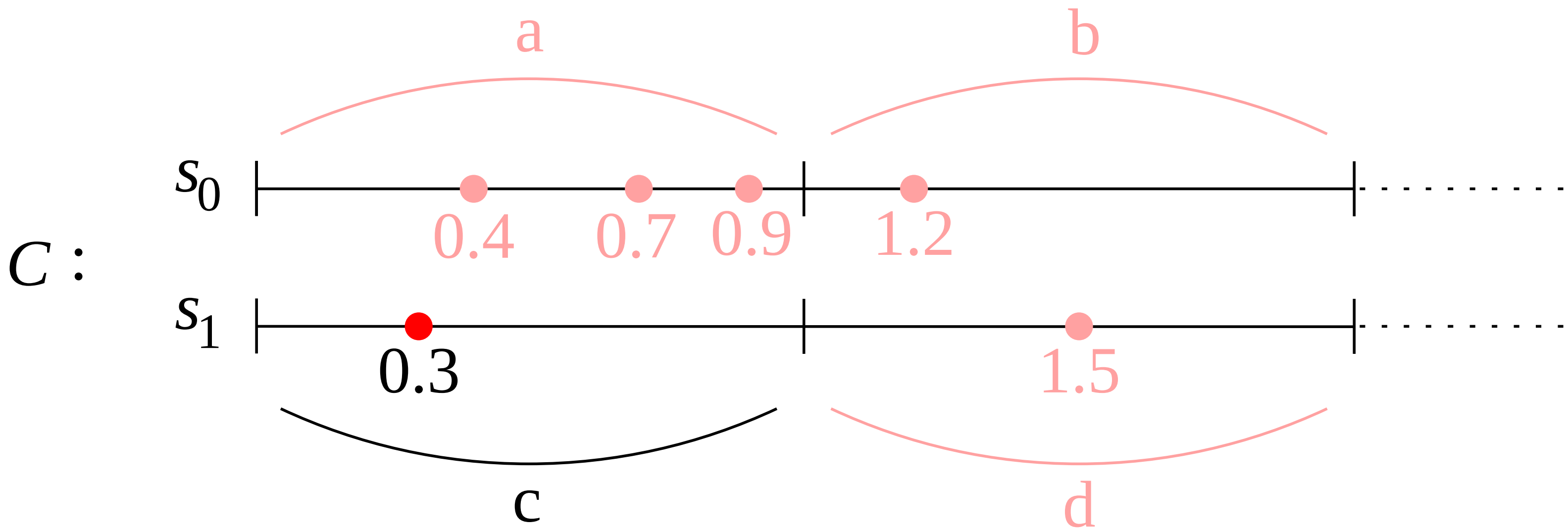
$C$ :




We encode $C$ as $H(C) = \mathrm{b} \ldots$

# Encoding Configurations as Words: Example

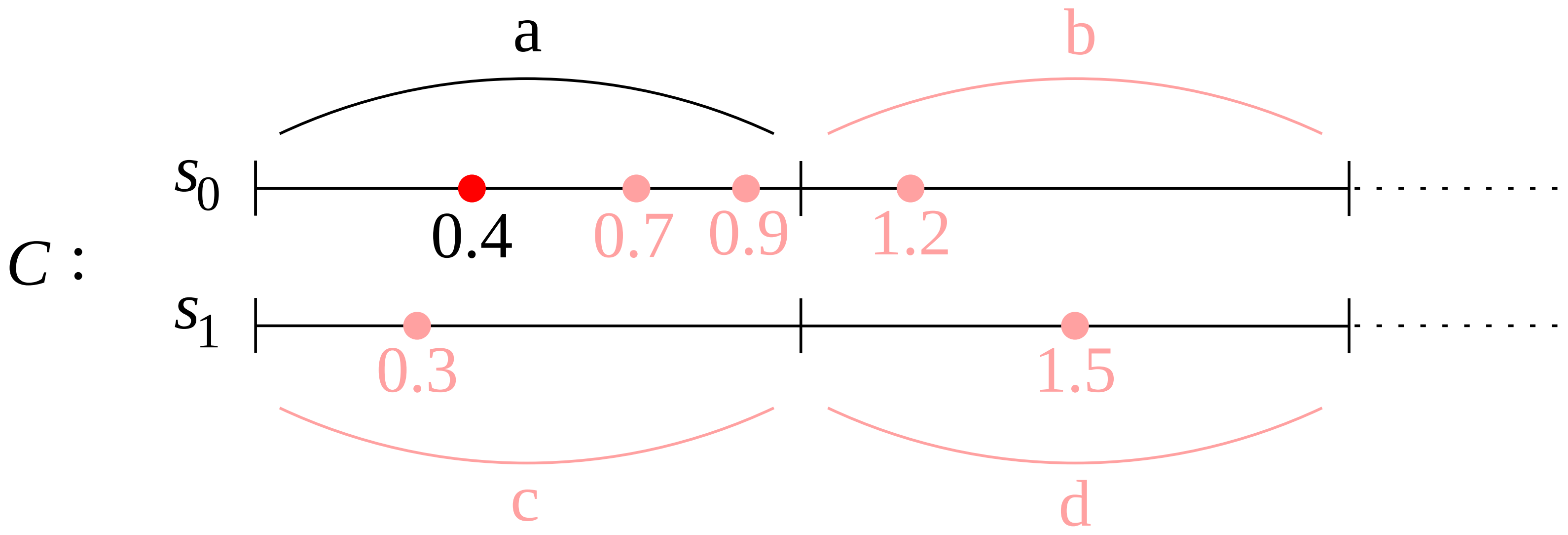
Consider the configuration $C$:




We encode $C$ as $H(C) = \mathrm{bc}\ldots$

## Encoding Configurations as Words: Example

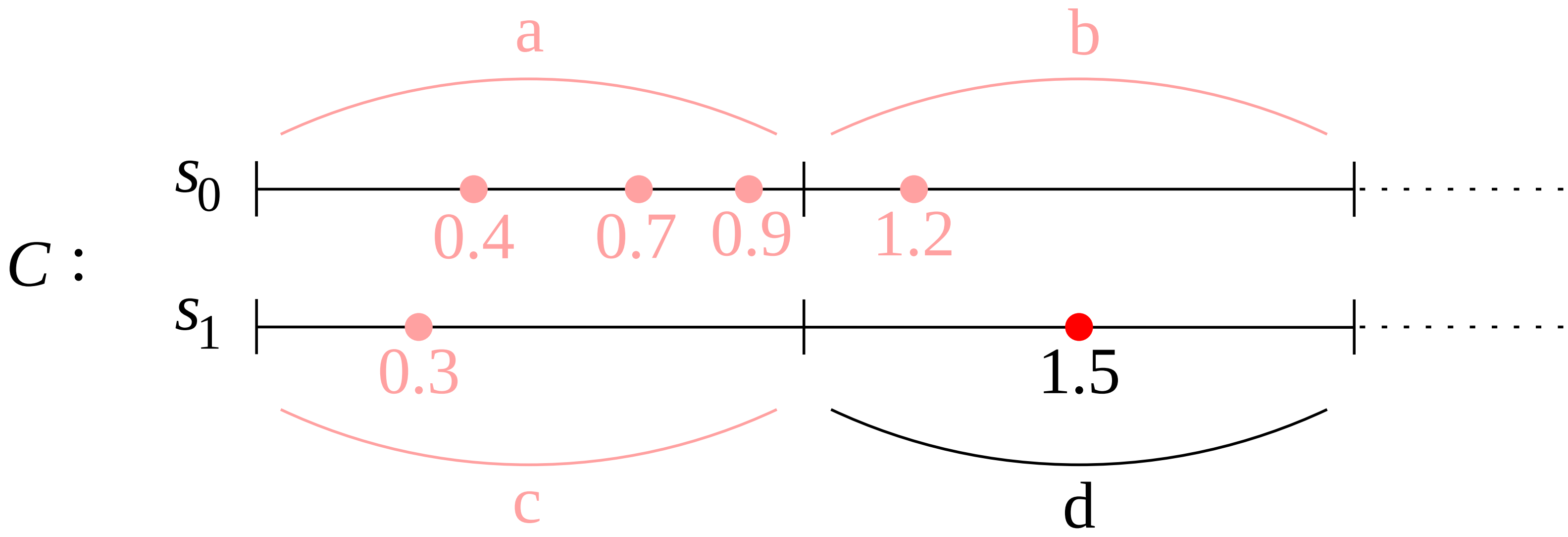
Consider the configuration $C$:




We encode $C$ as $H(C) = \mathrm{bca}\ldots$

# Encoding Configurations as Words: Example

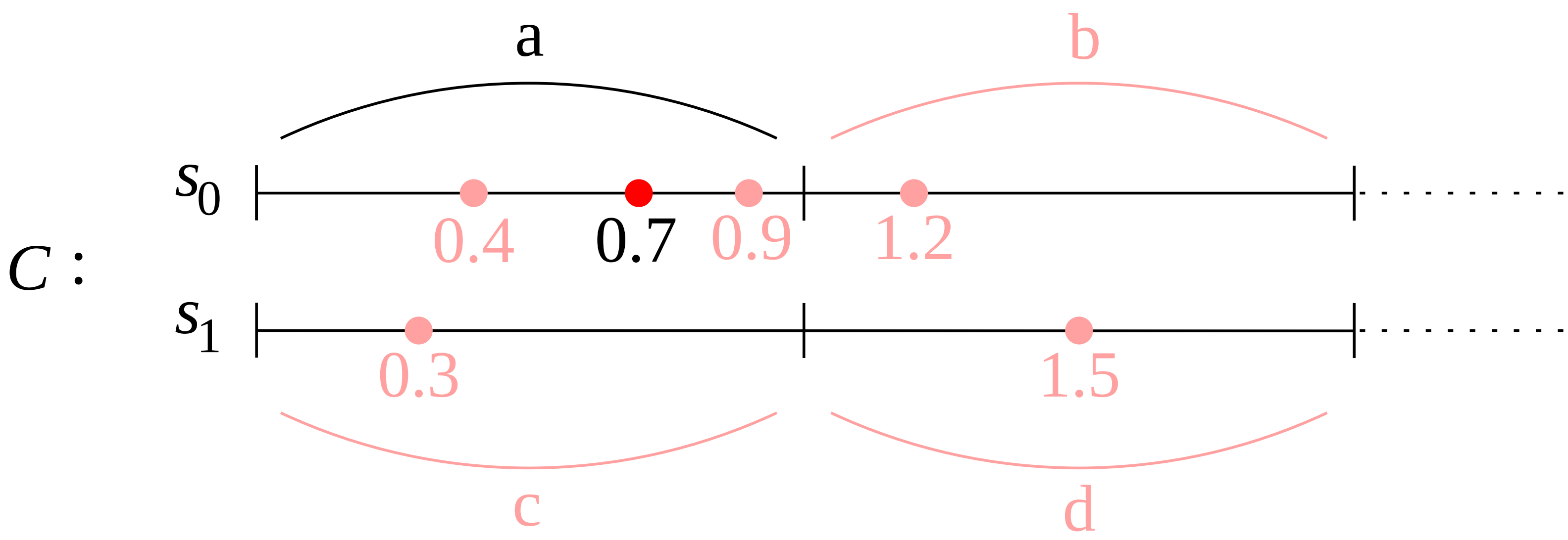
Consider the configuration $C$:




We encode $C$ as $H(C) = \mathrm{bcad}\ldots$

# Encoding Configurations as Words: Example

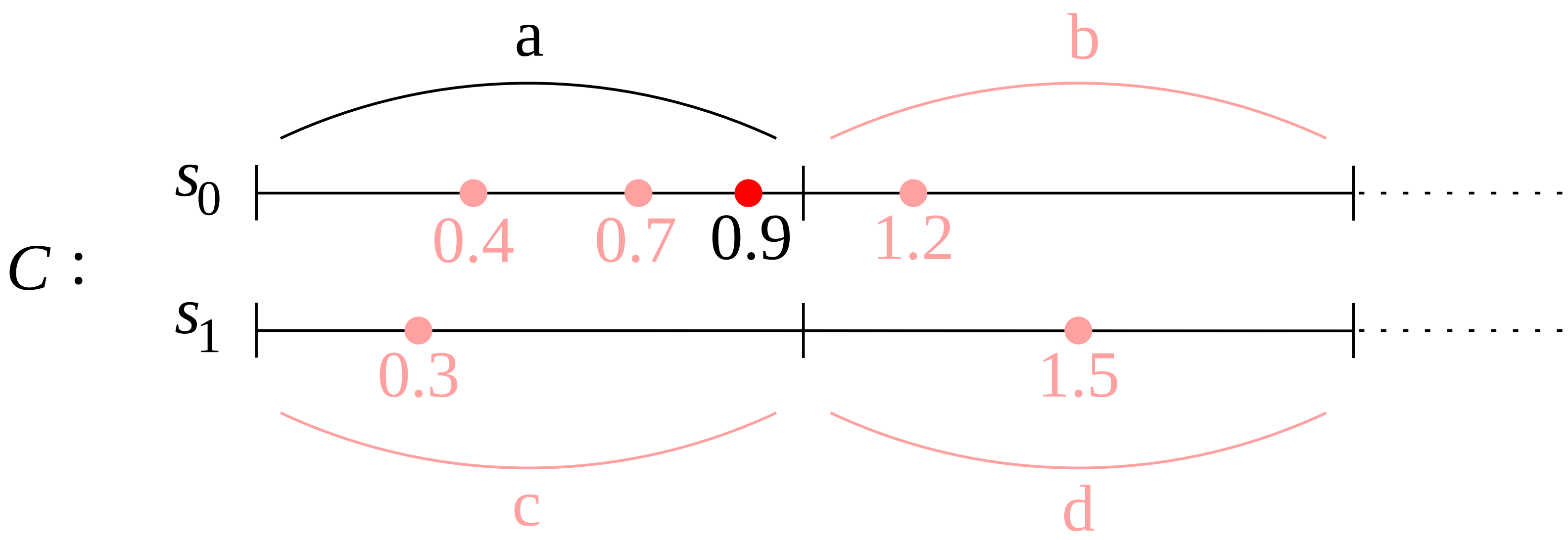
Consider the configuration $C$:




We encode $C$ as $H(C) = \text{bcada} \ldots$

# Encoding Configurations as Words: Example

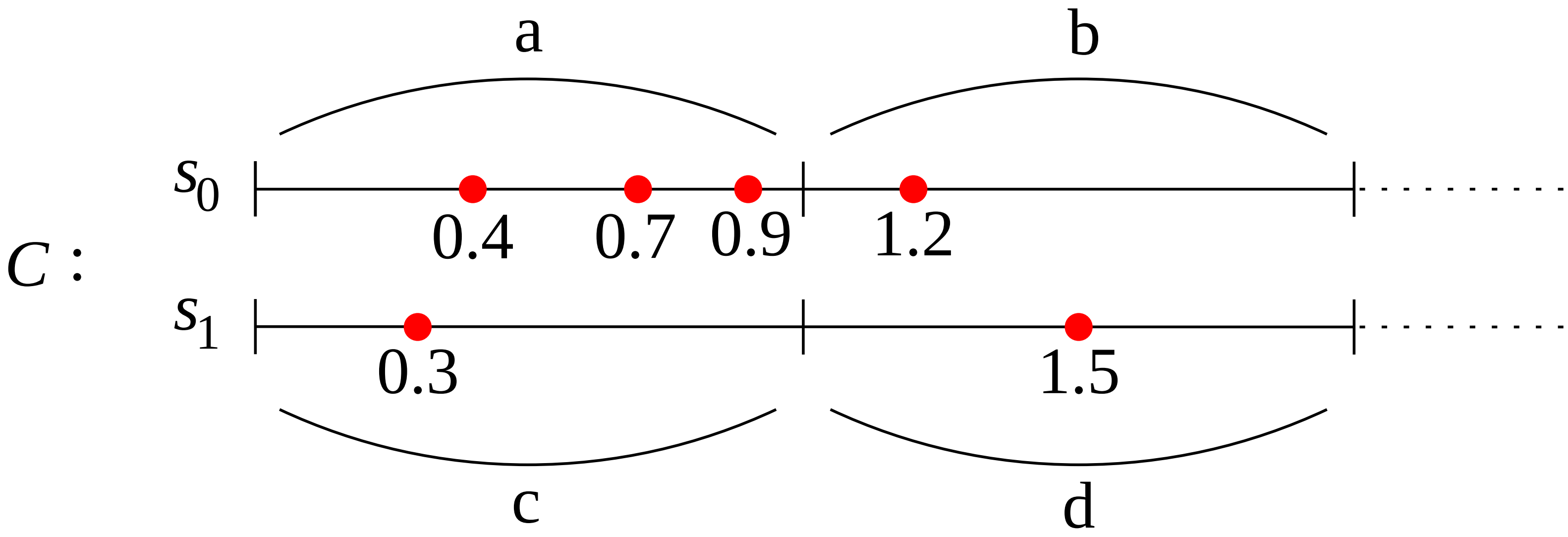
Consider the configuration $C$:




We encode $C$ as $H(C) = \mathrm{bcadaa}$

# Encoding Configurations as Words: Example

Consider the configuration $C$:




We encode $C$ as $H(C) = \text{bcadaa}$.

# From Bisimulation to Simulation

**Theorem.** If $H(C) = H(C')$, then $C \sim C'$.

# From Bisimulation to Simulation

**Theorem.** If $H(C) = H(C')$, then $C \sim C'$.

**Corollary.** If $H(C) = H(C')$, then

$$A[C] \text{ is universal} \iff A[C'] \text{ is universal.}$$

# From Bisimulation to Simulation

**Theorem.** If $H(C) = H(C')$, then $C \sim C'$.

**Corollary.** If $H(C) = H(C')$, then

$$A[C] \text{ is universal} \iff A[C'] \text{ is universal.}$$

**Corollary.** If $H(C) \preccurlyeq H(C')$, then

$$A[C] \text{ is universal} \implies A[C'] \text{ is universal.}$$

# The Algorithm: Recapitulation

- Reduce the universality question $L(A) \stackrel{?}{=} \mathbf{TT}$ to a reachability question on an infinite graph of words.
- The subword order $\preccurlyeq$ on this graph is a compatible well-quasi-order:
  - Whenever $H(C) \preccurlyeq H(C')$:
    if $A[C]$ is universal, then $A[C']$ is universal.
  - Any infinite sequence $H(C_1)$, $H(C_2)$, $H(C_3)$, $\ldots$ eventually saturates: there exists $i < j$ such that $H(C_i) \preccurlyeq H(C_j)$.
- Explore the graph, looking for a word/configuration from which $A$ cannot perform some event. The search must eventually terminate.

# Timed Automata Language Inclusion

**Theorem.** The language inclusion problem $L(B) \stackrel{?}{\subseteq} L(A)$ is **decidable**, provided $A$ has at most one clock.
The complexity is **non-primitive recursive**.

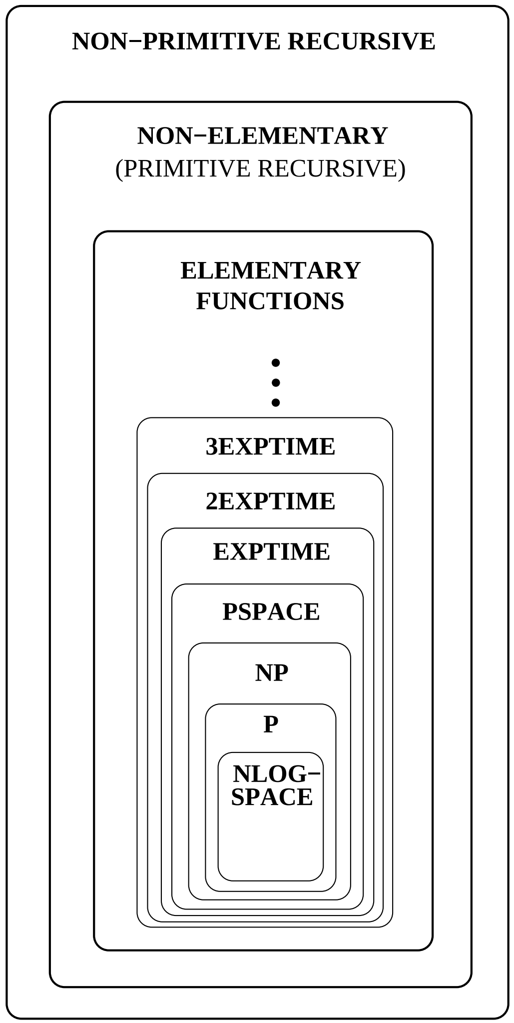
Non-primitive recursive complexity lower bound is established by reduction from reachability problem for lossy channel systems.

# Algorithmic Complexity

Emptiness/
Reachability

**UNDECIDABLE**

Universality/
Language Inclusion

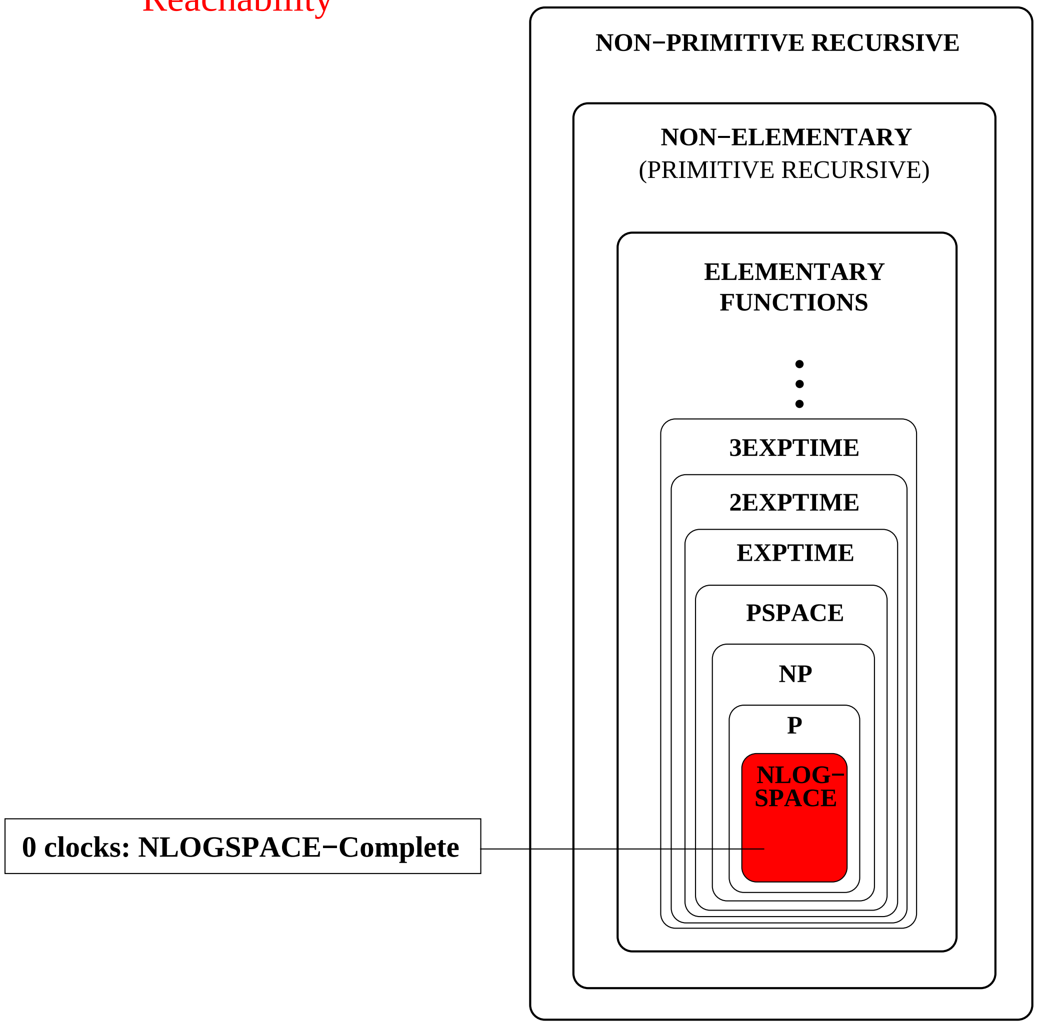


# Algorithmic Complexity

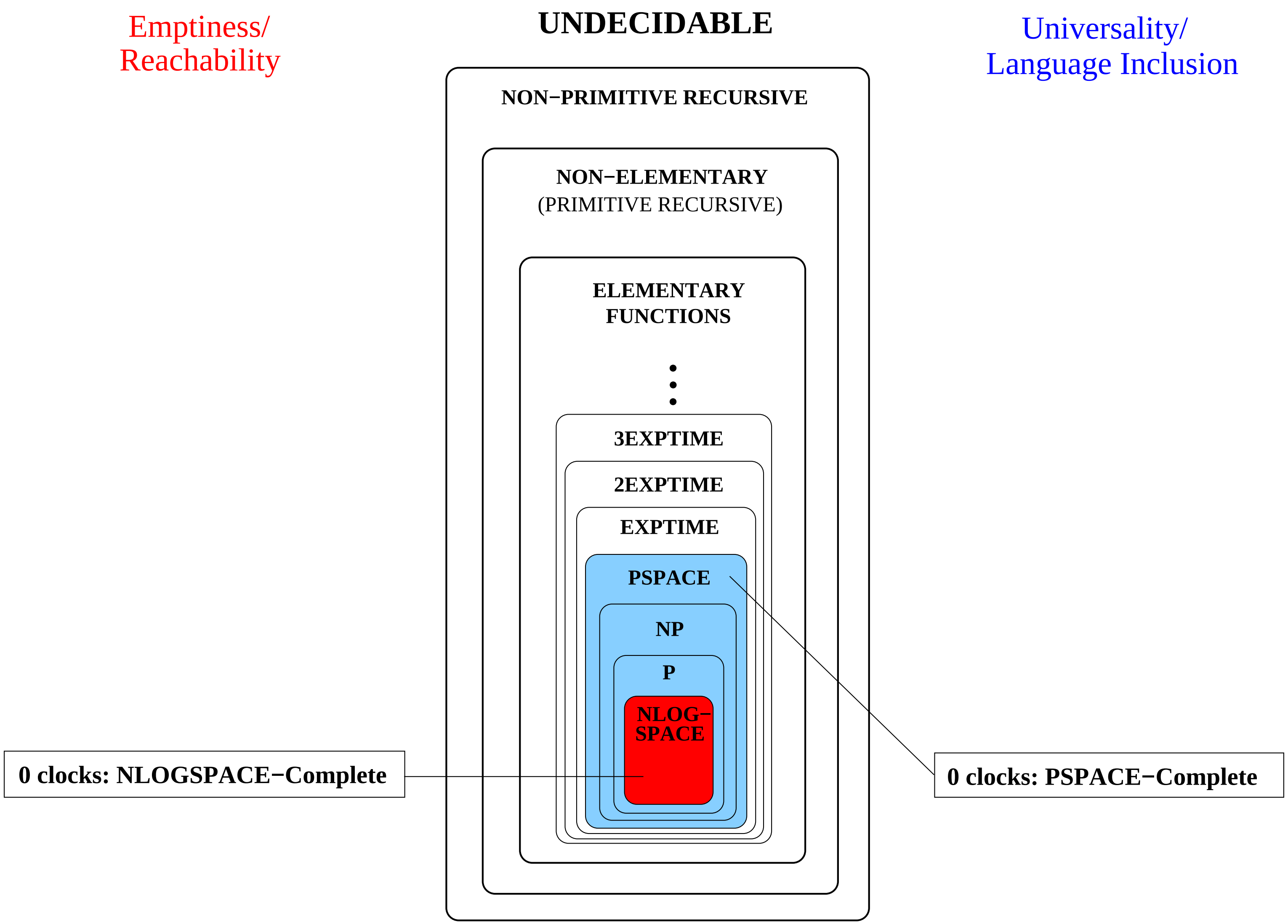
Emptiness/
Reachability

UNDECIDABLE

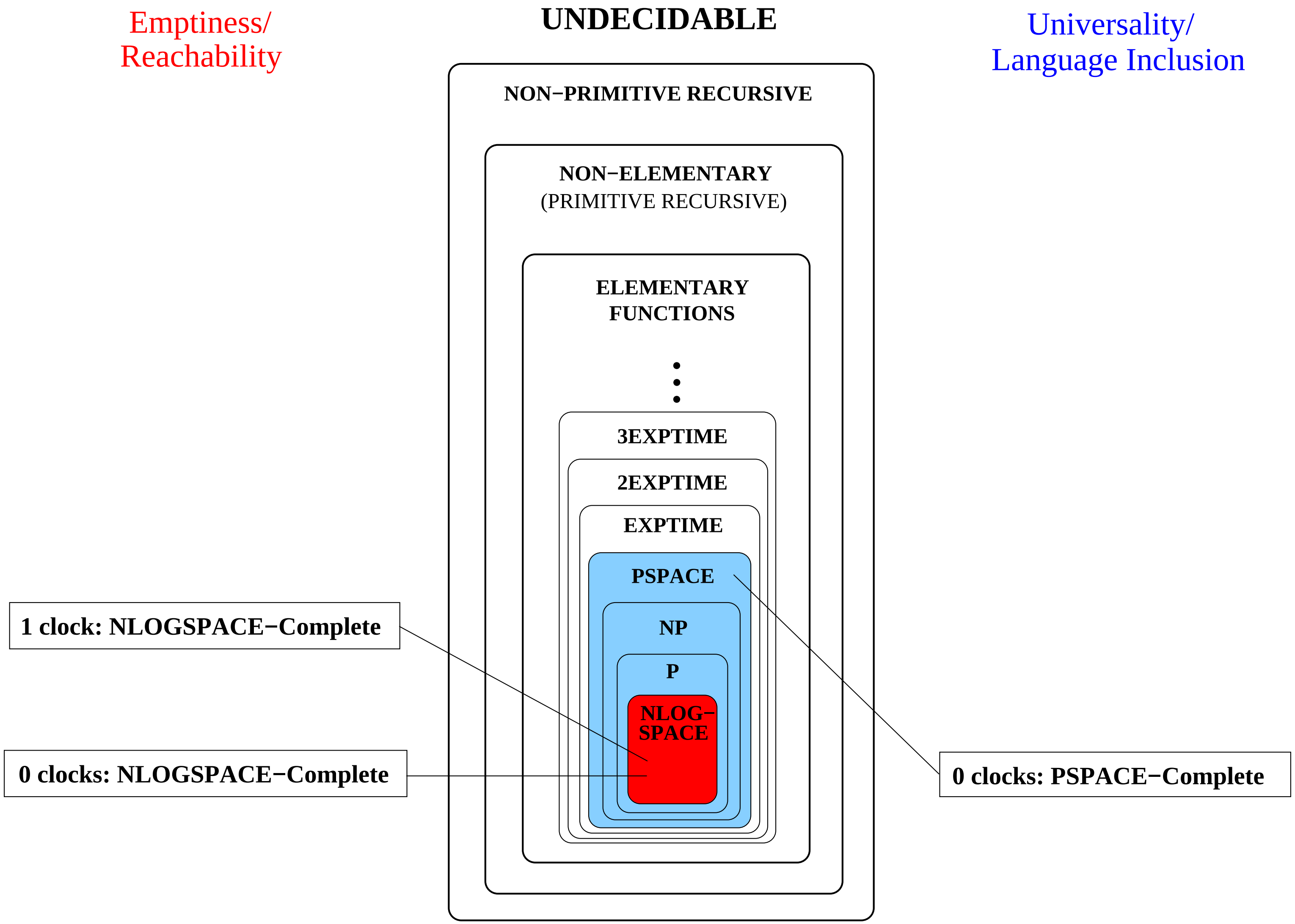
Universality/
Language Inclusion



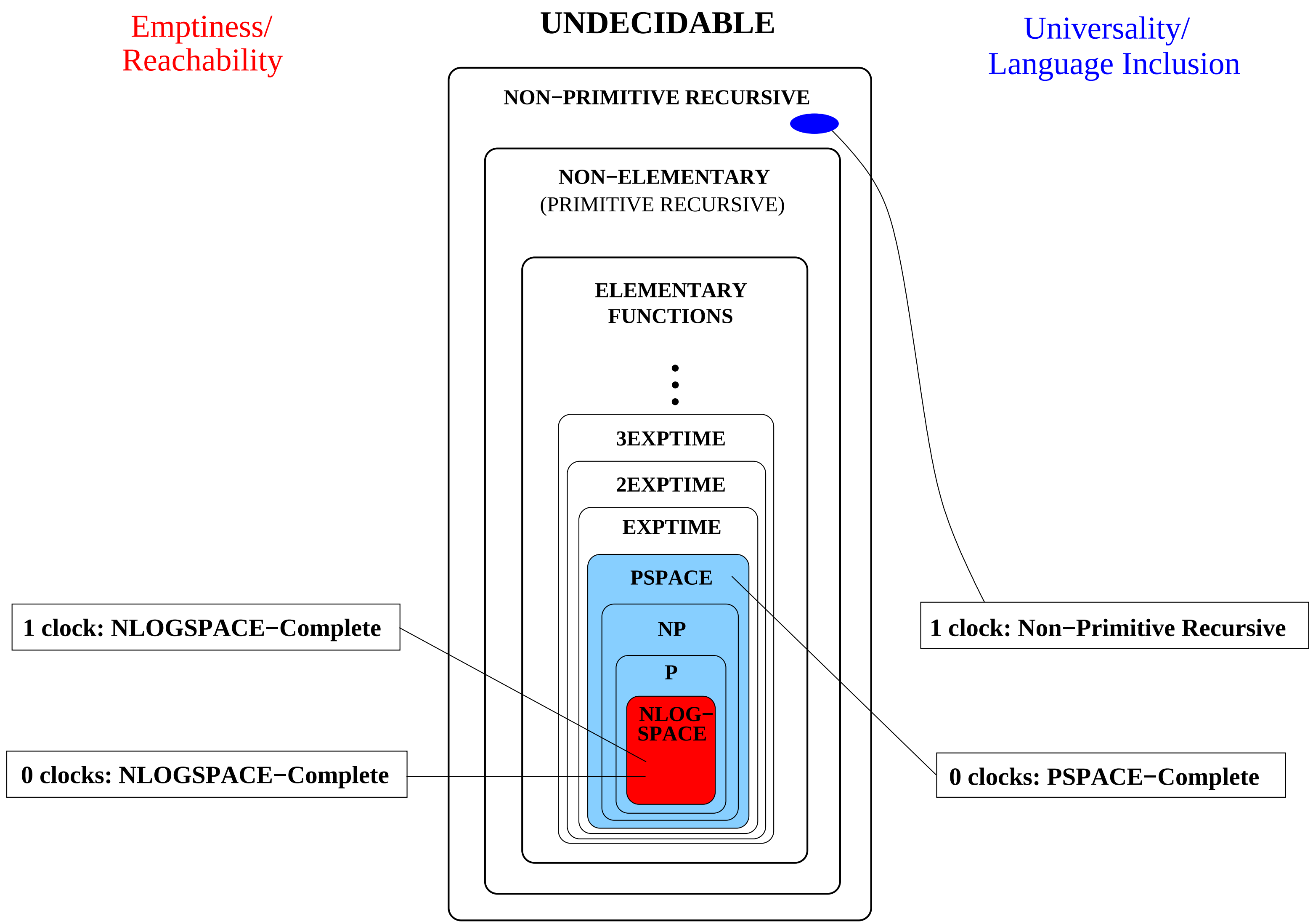
# Algorithmic Complexity



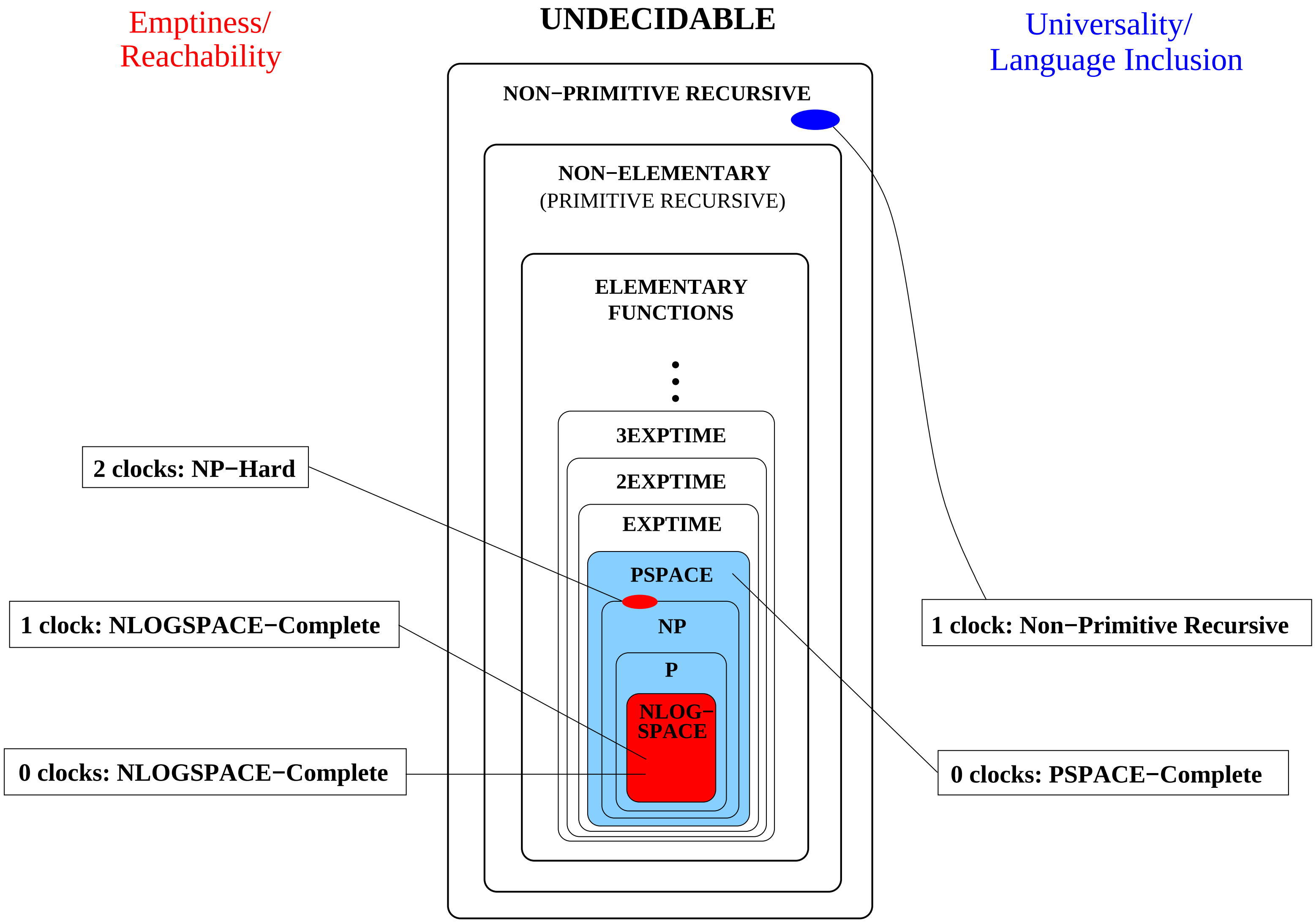
# Algorithmic Complexity

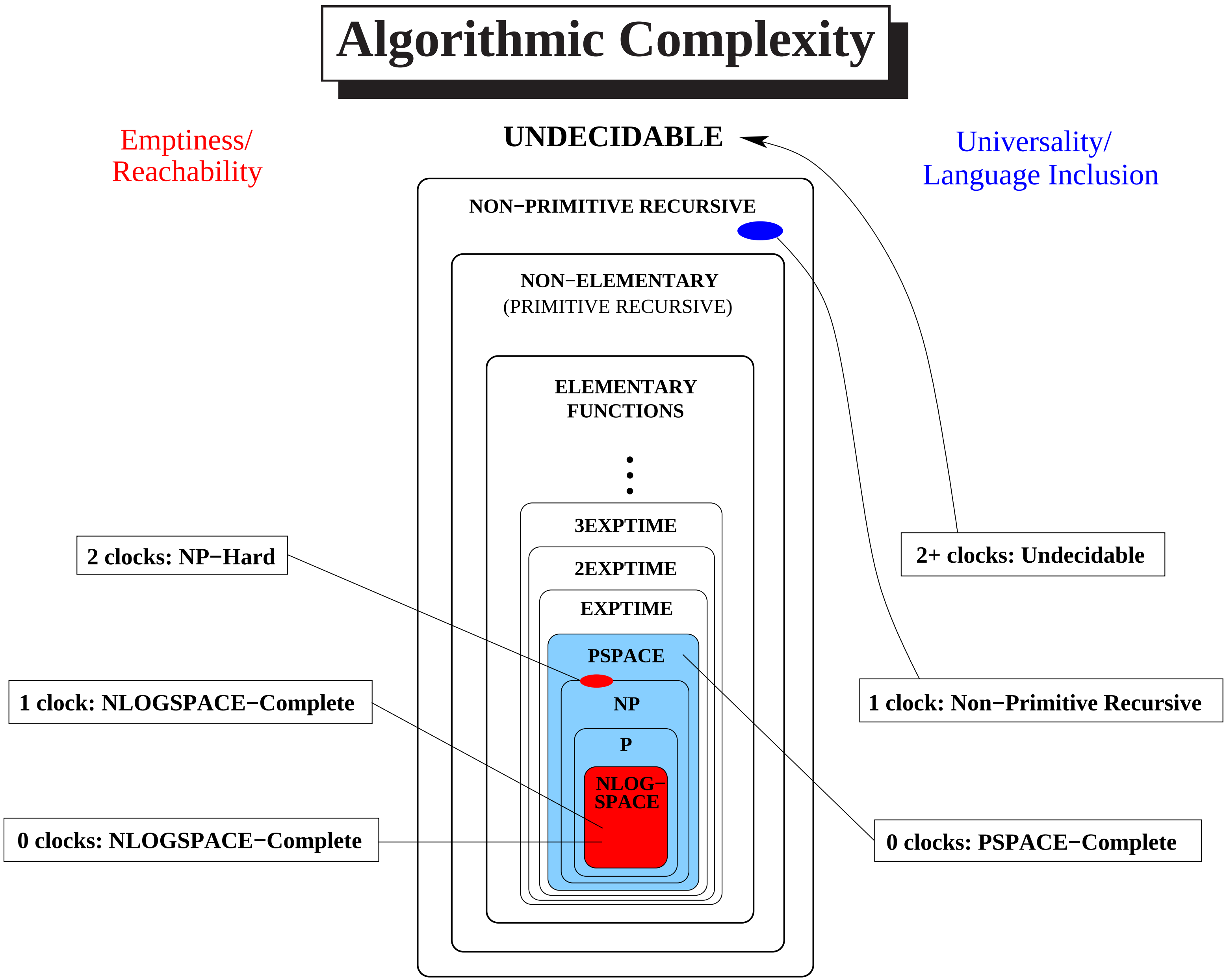


# Algorithmic Complexity

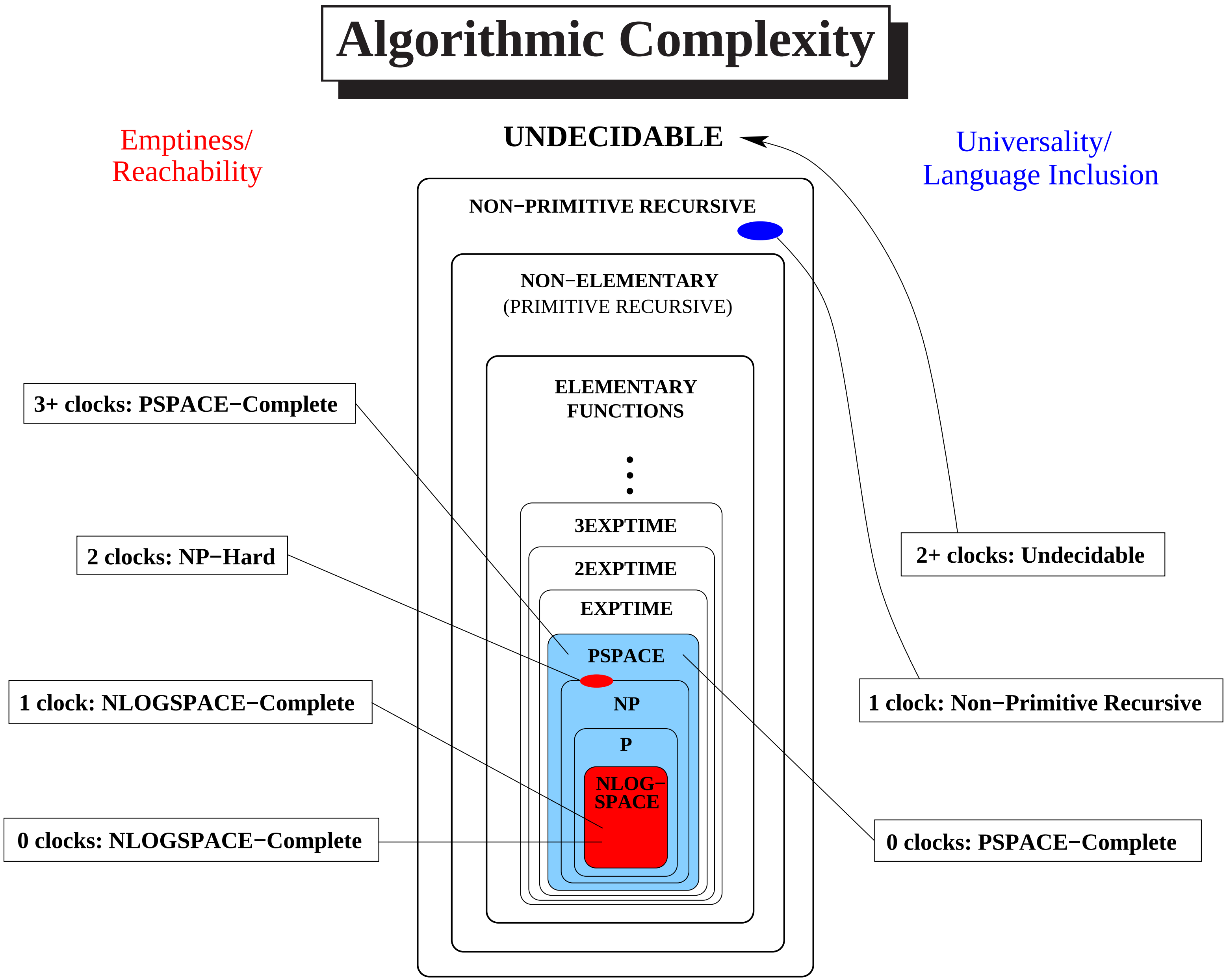


# Algorithmic Complexity





Algorithmic Complexity
Emptiness/
Reachability
UNDECIDABLE
Universality/
Language Inclusion
NON−PRIMITIVE RECURSIVE
NON−ELEMENTARY
(PRIMITIVE RECURSIVE)
ELEMENTARY
FUNCTIONS
3EXPTIME
2EXPTIME
EXPTIME
PSPACE
NP
P
NLOG−
SPACE
2 clocks: NP−Hard
1 clock: NLOGSPACE−Complete
0 clocks: NLOGSPACE−Complete
2+ clocks: Undecidable
1 clock: Non−Primitive Recursive
0 clocks: PSPACE−Complete



Algorithmic Complexity
Emptiness/
Reachability
UNDECIDABLE
Universality/
Language Inclusion
NON−PRIMITIVE RECURSIVE
NON−ELEMENTARY
(PRIMITIVE RECURSIVE)
ELEMENTARY
FUNCTIONS
3EXPTIME
2EXPTIME
EXPTIME
PSPACE
NP
P
NLOG−
SPACE
3+ clocks: PSPACE−Complete
2 clocks: NP−Hard
1 clock: NLOGSPACE−Complete
0 clocks: NLOGSPACE−Complete
2+ clocks: Undecidable
1 clock: Non−Primitive Recursive
0 clocks: PSPACE−Complete

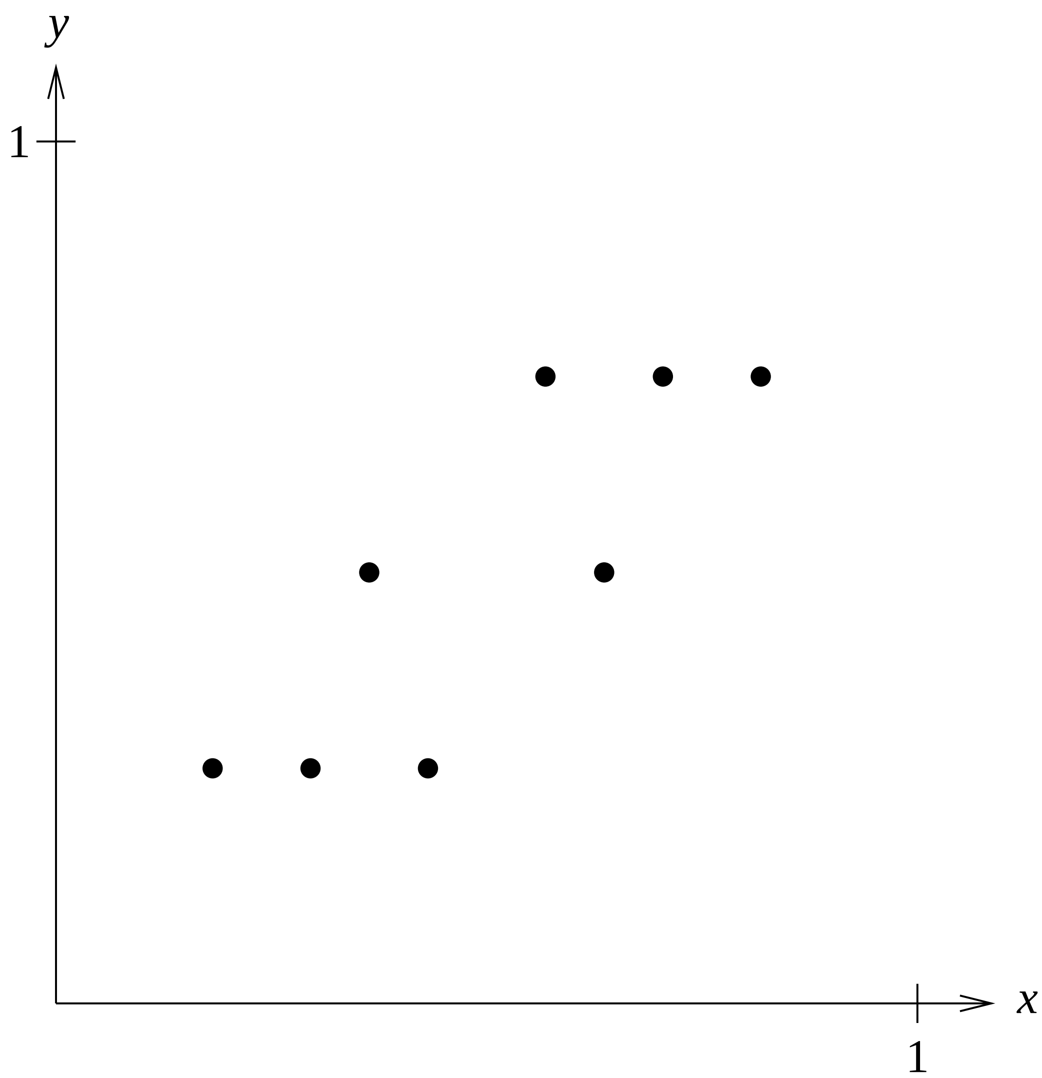
# Summary

- A single clock is surprisingly powerful . . .
  - can capture simple timed functional specifications
  - can capture substantial fragments of MTL

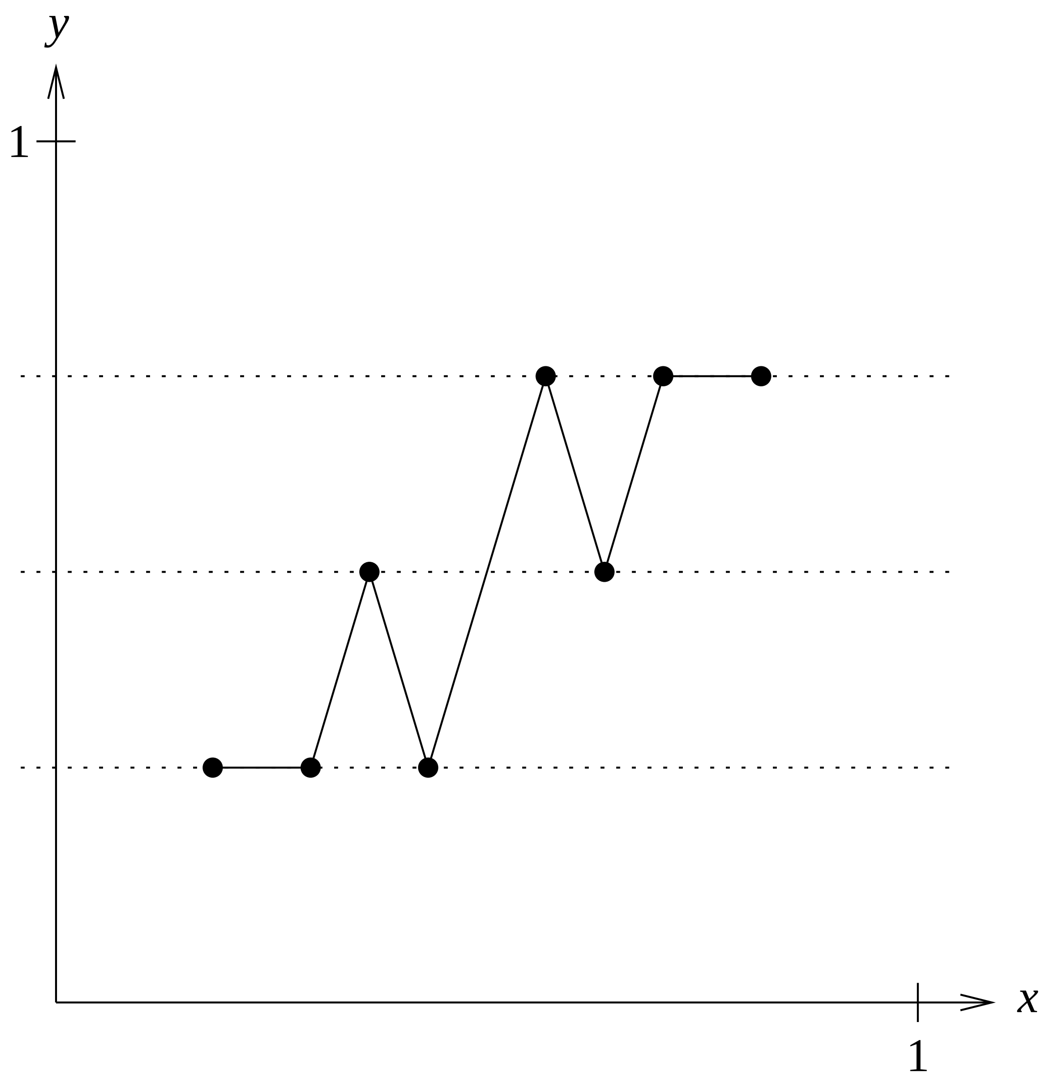
  . . . yet still lives in a decidable world.

- Punctuality not *quite* as noxious as previously thought:
  - but it does take language inclusion from PSPACE to Non-Primitive Recursive!

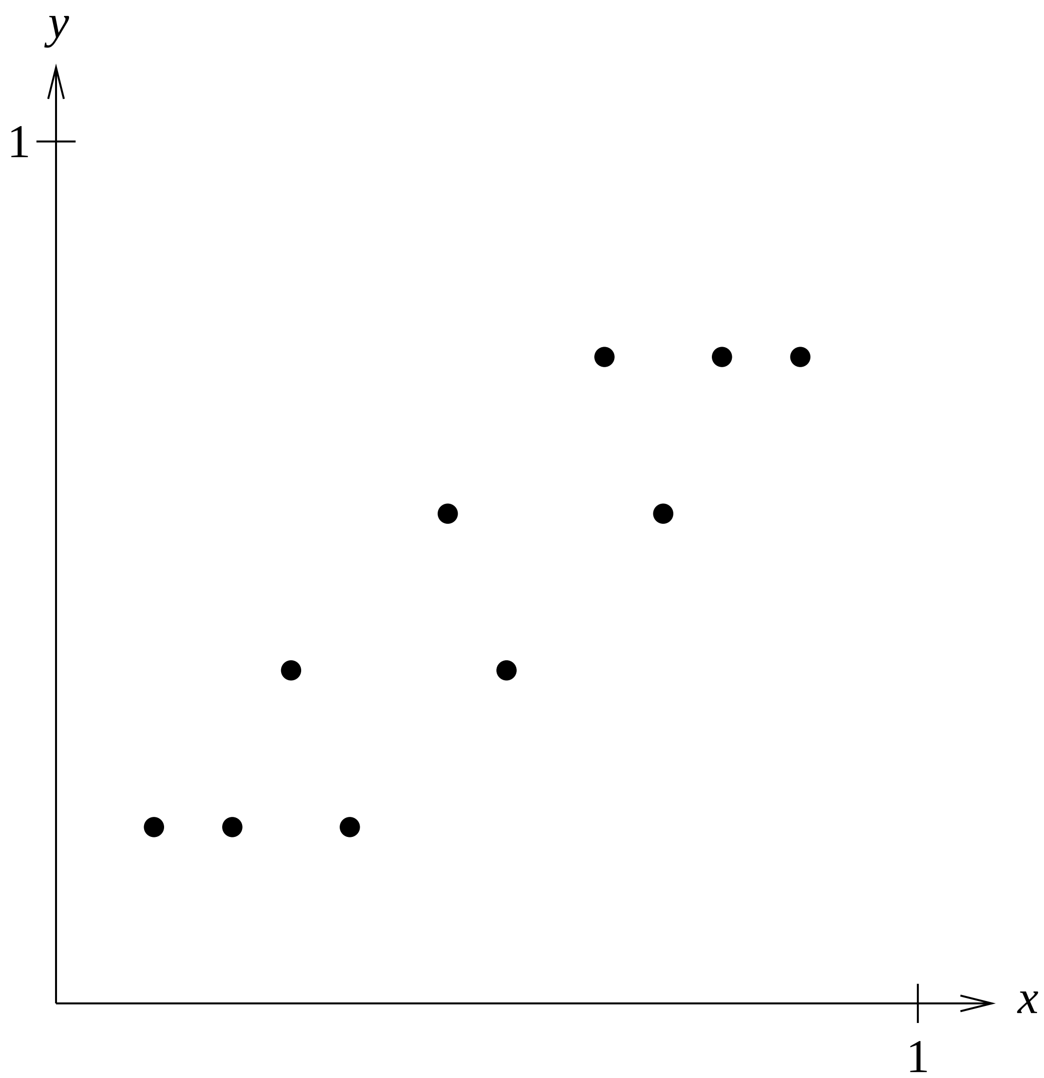
# Two Clocks: Configuration $G_3$



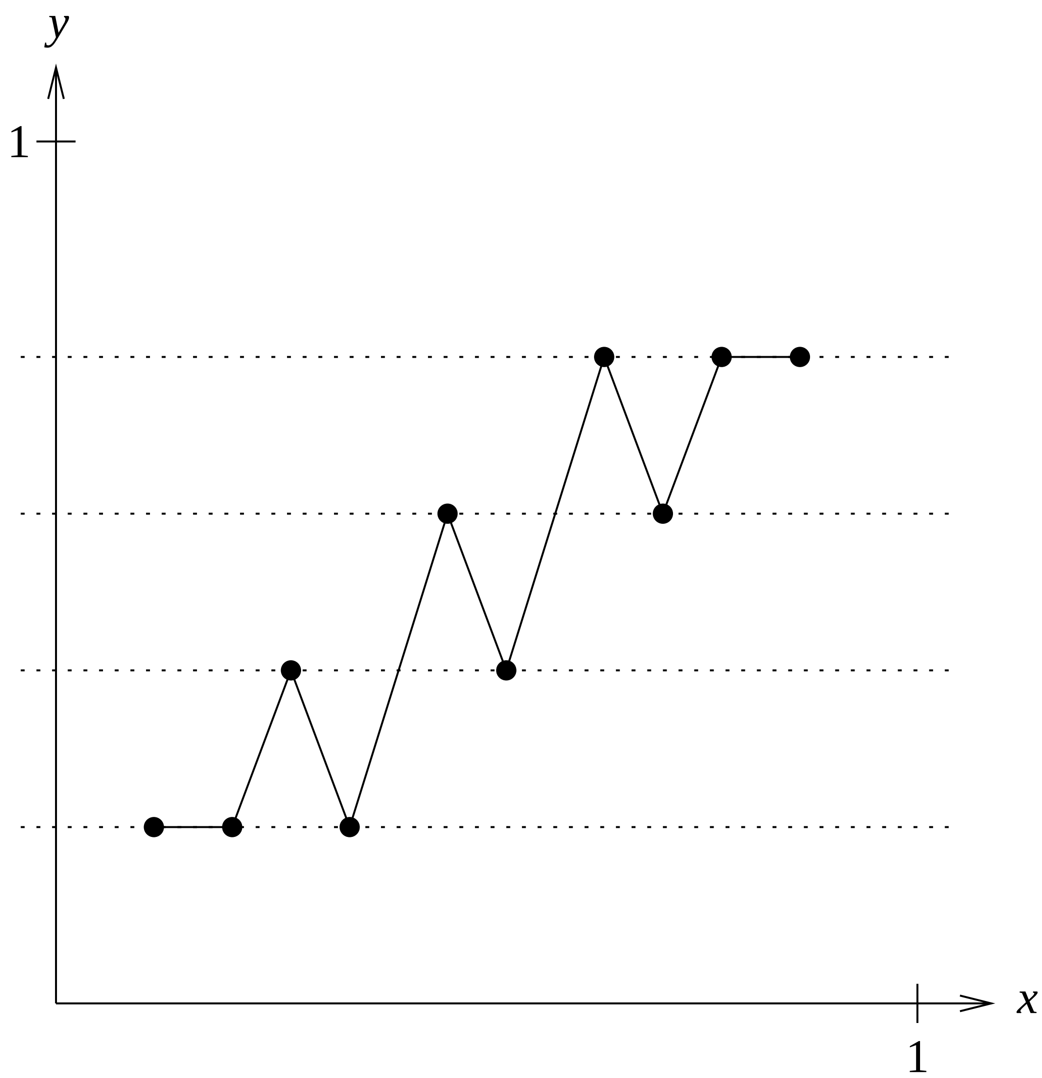
# Two Clocks: Configuration $G_3$



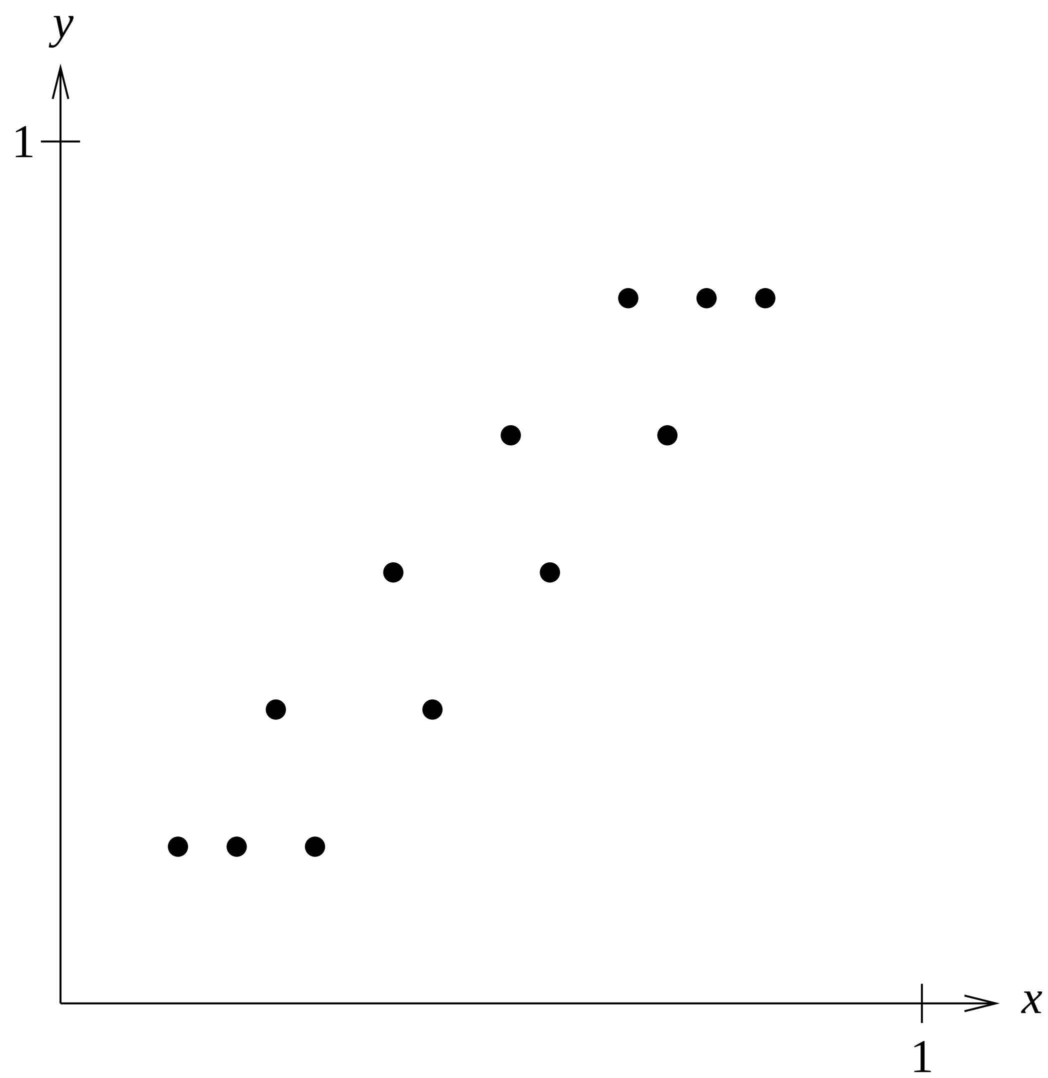
## Two Clocks: Configuration $G_4$



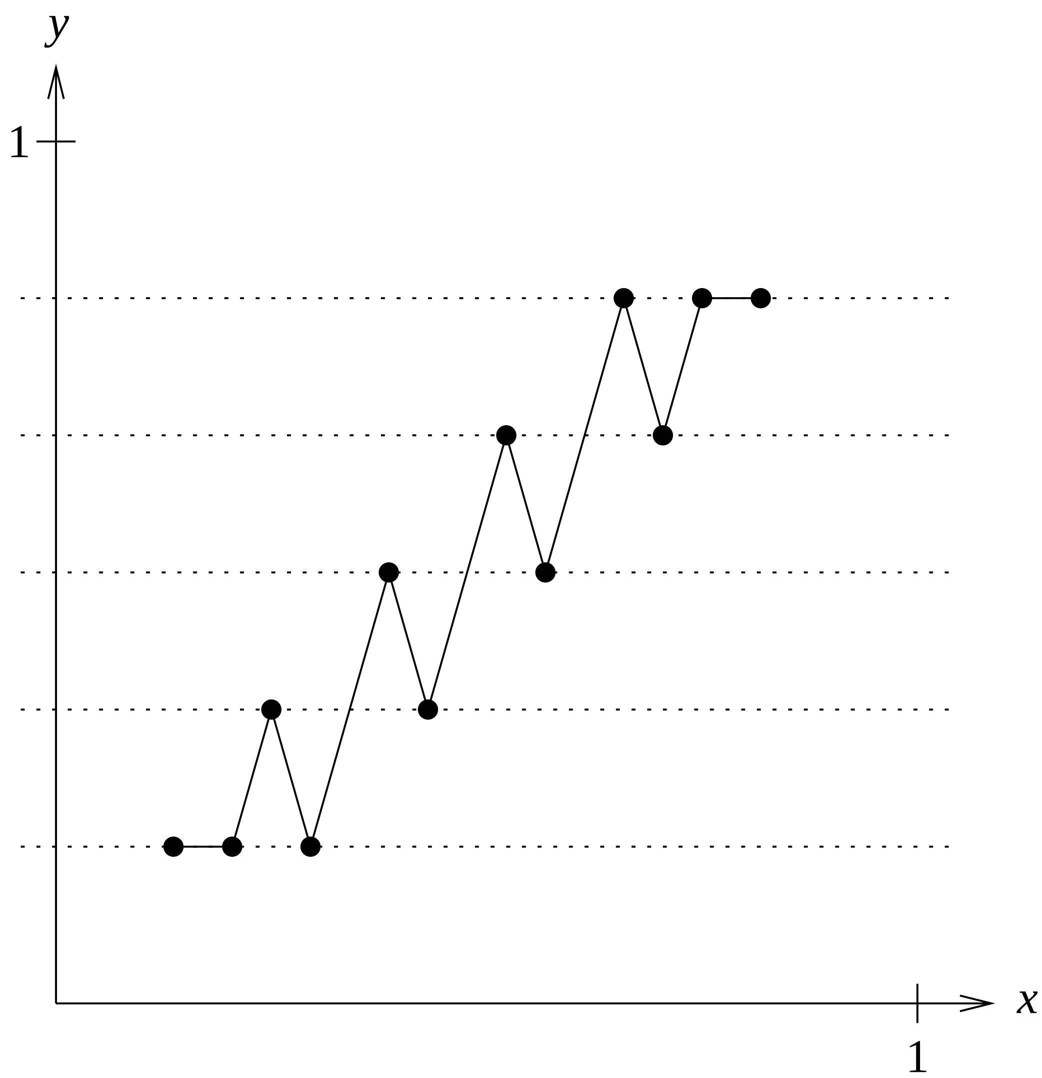
# Two Clocks: Configuration $G_4$



# Two Clocks: Configuration $G_5$



# Two Clocks: Configuration $G_5$



## Part II: Future Work

- Efficient implementation:
  - Symbolic algorithms — using better-quasi-orders?
  - Good conservative abstractions.
  - Counterexample-guided framework.
- Language inclusion when discounting the future and/or bounding time.
- Connections with lossy and insertion channel systems:
  - Logical characterization of the expressive power of one-clock timed alternating automata.

# Part IV: Time-Bounded Verification

James Worrell

Oxford University Computing Laboratory

MOVEP, July 2010

# A Long Time Ago, circa 2003. . .

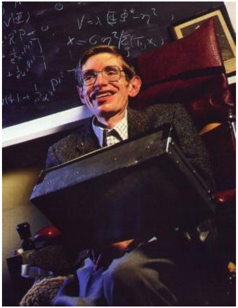
# Time-Bounded Language Inclusion

TIME-BOUNDED LANGUAGE INCLUSION PROBLEM

**Instance:** Timed automata $A$, $B$, and time bound $T \in \mathbb{N}$

**Question:** Is $L_T(A) \subseteq L_T(B)$ ?

- Inspired by Bounded Model Checking.
- Timed systems often have time bounds (e.g. timeouts), even if total number of actions is potentially unbounded.
- Universe's lifetime is believed to be bounded anyway…



# Timed Automata and Metric Logics

- Unfortunately, timed automata cannot be complemented even over bounded time…
- Key to solution is to translate problem into logic: Behaviours of timed automata can be captured in MSO($<$,$+1$) (in fact, even in $\exists$MTL [Henzinger, Raskin, Schobbens 1998]).
- This reverses Vardi's 'automata-theoretic approach to verification' paradigm!

# Monadic Second-Order Logic

More problems:

**Theorem (Shelah 1975)**

*MSO($<$) is undecidable over* $[0, 1)$.

By contrast,

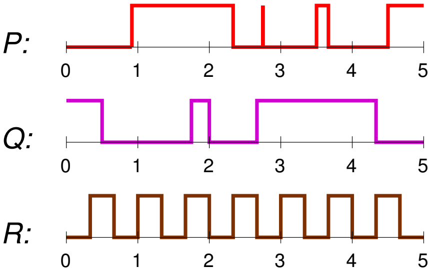
**Theorem**

- *MSO($<$) is decidable over* $\mathbb{N}$ *[Büchi 1960]*
- *MSO($<$) is decidable over* $\mathbb{Q}$*, via [Rabin 1969]*

# Finite Variability

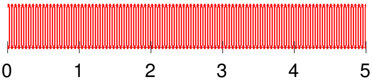
Timed behaviours are modelled as flows (or signals):

$f : [0, T) \to 2^{\mathbf{MP}}$




Predicates must have finite variability:

Disallow e.g. $\mathbb{Q}$:




Then:

## Theorem (Rabinovich 2002)

*MSO(<) satisfiability over finitely-variable flows is decidable.*

# The Time-Bounded Theory of Verification

## Theorem

*For any fixed bounded time domain $[0, T)$, the satisfiability and model-checking problems for MSO($<$,$+1$), FO($<$,$+1$), and MTL are all decidable, with the following complexities:*

| | |
|---|---|
| *MSO($<$,$+1$)* | *NON-ELEMENTARY* |
| *FO($<$,$+1$)* | *NON-ELEMENTARY* |
| *MTL* | *EXPSPACE-complete* |

## Theorem

*MTL and FO($<$,$+1$) are equally expressive over any fixed bounded time domain $[0, T)$.*

## Theorem

*Given timed automata A, B, and time bound $T \in \mathbb{N}$, the language inclusion problem $L_T(A) \subseteq L_T(B)$ is decidable and 2EXPSPACE-complete.*

# Time-Bounded Language Inclusion

- Let timed automata $A$, $B$, and time bound $T$ be given.
- Define formula $\varphi_A^{\mathrm{acc}}(\mathbf{W}, \mathbf{P})$ in MSO($<$,$+1$) such that:

$$A \text{ accepts timed word } w \iff \varphi_A^{\mathrm{acc}}(\mathbf{W}, \mathbf{P}) \text{ holds}$$

  where
  - $\mathbf{W}$ encodes $w$
  - $\mathbf{P}$ encodes a corresponding run of $A$.
- Define likewise $\varphi_B^{\mathrm{acc}}(\mathbf{W}, \mathbf{Q})$ for timed automaton $B$.
- Then $L_T(A) \subseteq L_T(B)$ iff:

$$\forall \mathbf{W} \, \forall \mathbf{P} \; (\varphi_A^{\mathrm{acc}}(\mathbf{W}, \mathbf{P}) \to \exists \mathbf{Q} \, \varphi_B^{\mathrm{acc}}(\mathbf{W}, \mathbf{Q}))$$

  holds over time domain $[0, T)$.
- This can be decided in 2EXPSPACE.

# MSO($<$,+1) Time-Bounded Satisfiability

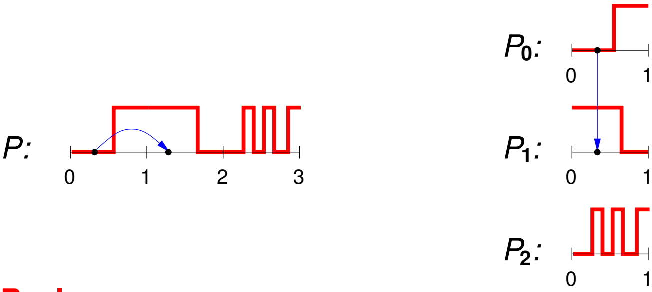
Key idea: eliminate the metric by 'vertical stacking'.

- Let $\varphi$ be an MSO($<$,$+$1) formula and let $T \in \mathbb{N}$.
- Construct an MSO($<$) formula $\overline{\varphi}$ such that:

$$\varphi \text{ is satisfiable over } [0, T) \iff \overline{\varphi} \text{ is satisfiable over } [0, 1)$$

- Conclude by invoking decidability of MSO($<$).

# From MSO(<,+1) to MSO(<)

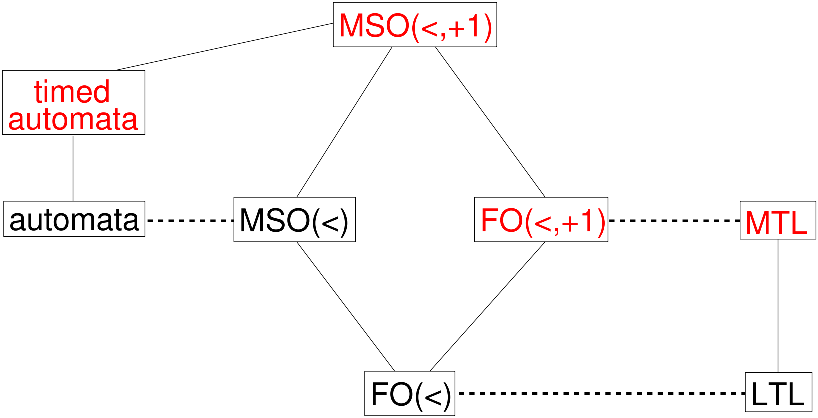



**Replace every:**

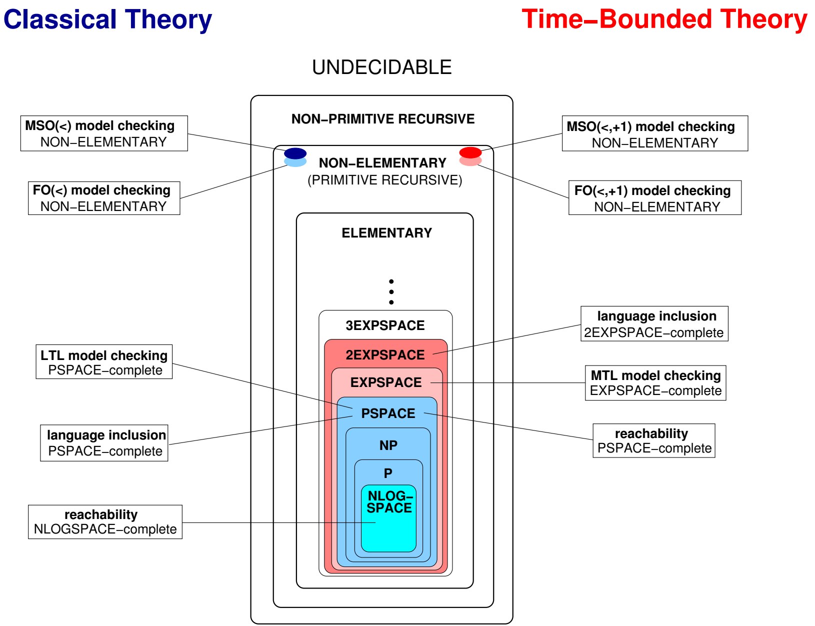
- $\forall x\, \psi(x)$ **by** $\forall x\ (\psi(x) \wedge \psi(x+1) \wedge \psi(x+2))$
- $x + k_1 < y + k_2$ **by** $\begin{cases} x < y & \text{if } k_1 = k_2 \\ \textbf{true} & \text{if } k_1 < k_2 \\ \textbf{false} & \text{if } k_1 > k_2 \end{cases}$
- $P(x+k)$ **by** $P_k(x)$
- $\forall P\, \psi$ **by** $\forall P_0\, \forall P_1\, \forall P_2\, \psi$

Then $\varphi$ is satisfiable over $[0, T)$ $\iff$ $\overline{\varphi}$ is satisfiable over $[0, 1)$.

# The Time-Bounded Theory: Expressiveness



# The Time-Bounded Theory: Complexity



# Part IV: Conclusion

- For specifying and verifying real-time systems, the time-bounded theory is much better behaved than the real-time theory.
- Original motivation for this work was the time-bounded language inclusion problem for timed automata.
  We used logic as a tool to solve this problem.

**Thank you for your attention!**