Computational Logic
Multi-modal Logics

Alexandre Madeira

Dep. Matemática, U. Aveiro

May 7, 2025

Motivations for the Section

Modal Logic is now used in the practices of **specification, modeling, and verification of complex systems**.

Specializations/adaptations to operationalize these practices:

1. Enrich the accessibility relations with actions:

   Modal → Multimodal

2. Interpret "programs," i.e., structured combinations of actions:

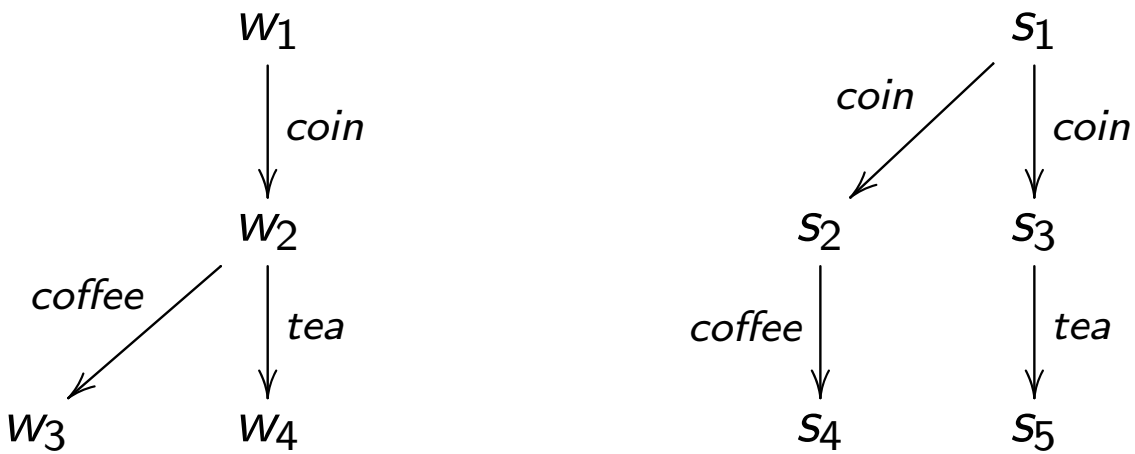
   Multimodal → Dynamic

3. Operational languages for defining the models: **Process Algebras**

Outline

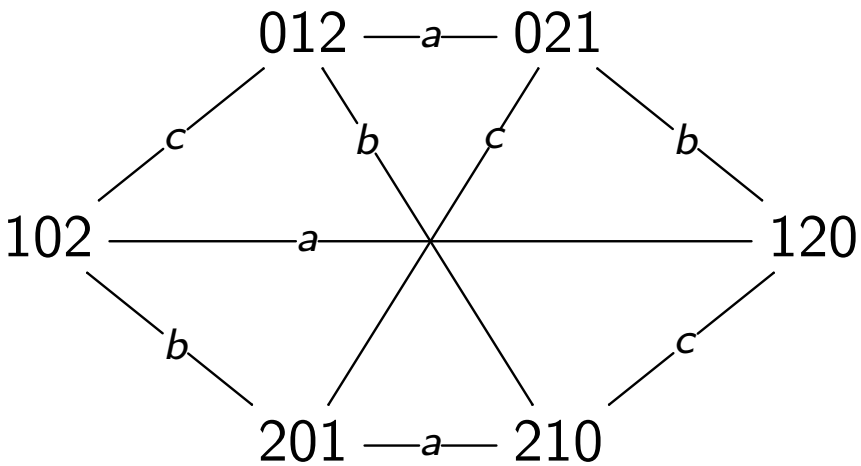
Processes are Transition Systems

Two Coffee Machines



Multi-agent Knowledge Systems are Transition Systems

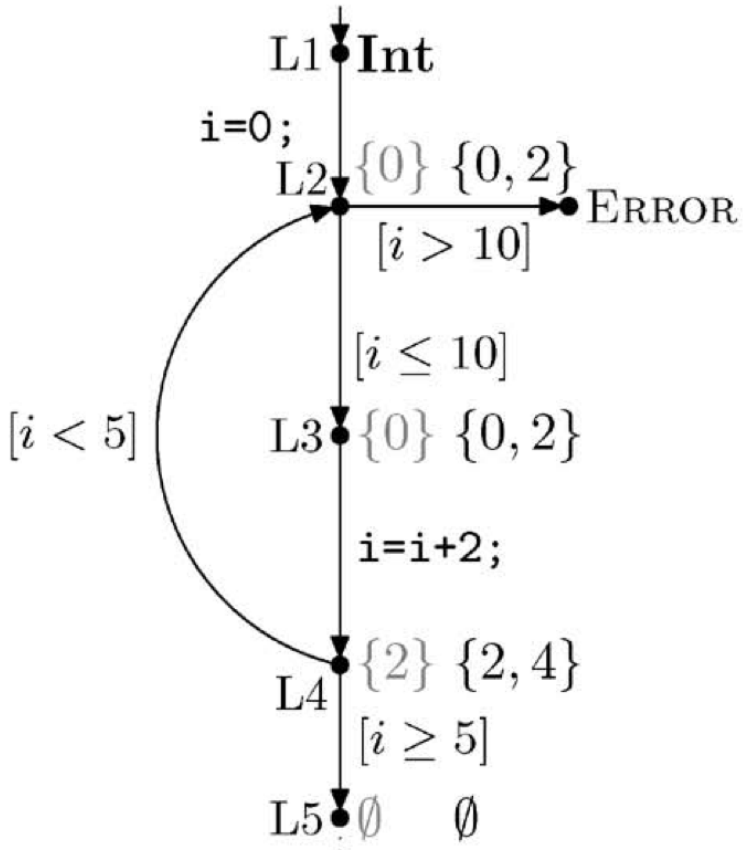
The Envelope Game



E.g., in state 012: Ana has the envelope with 0, Bob has the one with 1, and Clara has the one with 2.

Programs are Transition Systems

```
int i = 0;
do {
   assert(i <= 10);
   i = i+2;
} while (i < 5);
```



The Language

Multimodal Signature

A signature is a pair $(\mathrm{Prop}, \mathrm{Act})$ where Prop and Act are (disjoint) sets of **propositional symbols** and **action or modality symbols**, respectively.

The Language

Multimodal Signature

A signature is a pair $(\mathrm{Prop}, \mathrm{Act})$ where Prop and Act are (disjoint) sets of **propositional symbols** and **action or modality symbols**, respectively.

Formulas

Let $(\mathrm{Prop}, \mathrm{Act})$ be a multimodal signature. The set of multimodal formulas for $(\mathrm{Prop}, \mathrm{Act})$, denoted by $\mathrm{MFm}(\mathrm{Prop}, \mathrm{Act})$, is defined by the following grammar:

$$\phi ::= p \mid \bot \mid \phi \to \phi \mid [m]\,\phi$$

where $p \in \mathrm{Prop}$ and $m \in \mathrm{Act}$

The Language

Multimodal Signature

A signature is a pair $(\mathrm{Prop}, \mathrm{Act})$ where Prop and Act are (disjoint) sets of **propositional symbols** and **action or modality symbols**, respectively.

Formulas

Let $(\mathrm{Prop}, \mathrm{Act})$ be a multimodal signature. The set of multimodal formulas for $(\mathrm{Prop}, \mathrm{Act})$, denoted by $\mathrm{MFm}(\mathrm{Prop}, \mathrm{Act})$, is defined by the following grammar:

$$\phi ::= p \mid \bot \mid \phi \to \phi \mid [m]\,\phi$$

where $p \in \mathrm{Prop}$ and $m \in \mathrm{Act}$

Abbreviations

- $\langle m \rangle \phi := \neg [m] \neg \phi$
- ...

Multimodal Models and Structures

Multimodal Models and Structures

A **model** for a signature $(\mathrm{Prop}, \mathrm{Act})$ is a pair $\mathcal{M} = \langle \mathcal{F}, V \rangle$, where:

- $\mathcal{F} = \langle W, R \rangle$ is a **Kripke structure**, i.e.,
  - W is a non-empty set (of **states**)
  - $R = (R_m)_{m \in \mathrm{Act}}$ is a family of binary relations $R_m \subseteq W \times W$, one for each modality symbol $m \in \mathrm{Act}$
- $V : \mathrm{Prop} \to \mathcal{P}(W)$ is a **valuation**.

Multimodal Satisfaction Relation

Satisfaction in a model $\mathcal{M}$ at a state w

$\mathcal{M}, w \models \top$		
$\mathcal{M}, w \models p$	iff	$w \in V(p)$
$\mathcal{M}, w \models \phi_1 \rightarrow \phi_2$	iff	$\mathcal{M}, w \not\models \phi_1$ or $\mathcal{M}, w \models \phi_2$
$\mathcal{M}, w \models [m]\,\phi$	iff	$\forall v \in W.\ (w, v) \in R_m \Rightarrow \mathcal{M}, v \models \phi$

Multimodal Satisfaction Relation

Satisfaction in a model $\mathcal{M}$ at a state w

$$\begin{array}{lll}
\mathcal{M}, w \models \top & & \\
\mathcal{M}, w \models p & \text{iff} & w \in V(p) \\
\mathcal{M}, w \models \phi_1 \rightarrow \phi_2 & \text{iff} & \mathcal{M}, w \not\models \phi_1 \text{ or } \mathcal{M}, w \models \phi_2 \\
\mathcal{M}, w \models [m]\,\phi & \text{iff} & \forall v \in W.\ (w, v) \in R_m \Rightarrow \mathcal{M}, v \models \phi
\end{array}$$

Corollary:

$$\begin{array}{lll}
\mathcal{M}, w \not\models \bot & & \\
\mathcal{M}, w \models \neg\phi & \text{iff} & \mathcal{M}, w \not\models \phi \\
\mathcal{M}, w \models \phi_1 \wedge \phi_2 & \text{iff} & \mathcal{M}, w \models \phi_1 \text{ and } \mathcal{M}, w \models \phi_2 \\
\mathcal{M}, w \models \phi_1 \vee \phi_2 & \text{iff} & \mathcal{M}, w \models \phi_1 \text{ or } \mathcal{M}, w \models \phi_2 \\
\mathcal{M}, w \models \langle m \rangle\,\phi & \text{iff} & \exists v \in W \text{ such that } (w, v) \in R_m \text{ and } \mathcal{M}, v \models \phi
\end{array}$$

Multimodal Satisfaction

Satisfaction

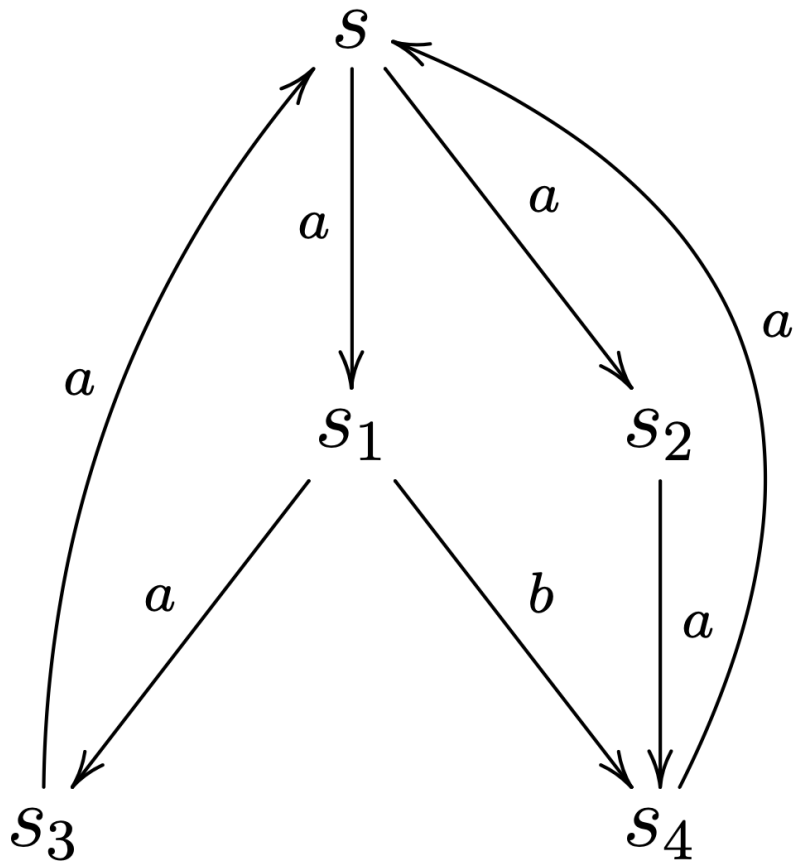
A formula ϕ in $\mathrm{MFm}(\mathrm{Prop}, \mathrm{Act})$ is:

- **satisfiable in** $\mathcal{M}$ if it is satisfied in some state w of $\mathcal{M}$
- **globally satisfiable** in $\mathcal{M}$ ($\mathcal{M} \models \phi$) if it is satisfied in every state of $\mathcal{M}$
- **valid** ($\models \phi$) if it is globally satisfied in all models over $(\mathrm{Prop}, \mathrm{Act})$
- **a semantic consequence** of a set of formulas Γ ($\Gamma \models \phi$) if for all models $\mathcal{M}$ and for all states w, if $\mathcal{M}, w \models \Gamma$ then $\mathcal{M}, w \models \phi$

Exercise

Verify whether:

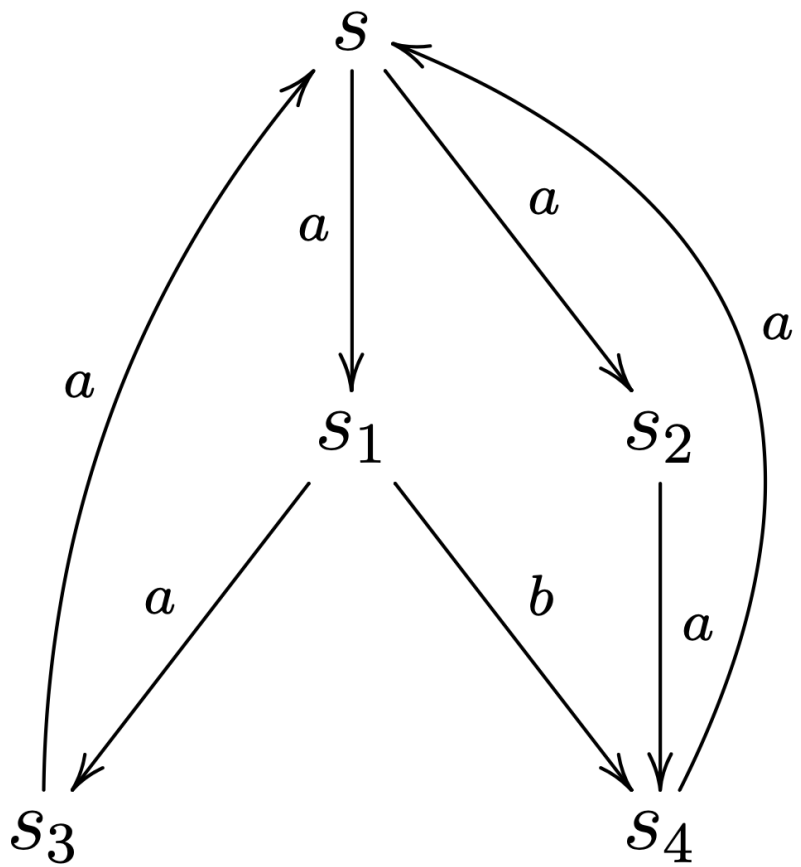
1. $M, s \models \langle a \rangle \top$
2. $M, s \models [a] \bot$
3. $M, s \models \langle b \rangle \top$
4. $M, s \models [b] \bot$
5. $M, s \models [a] \langle b \rangle \top$
6. $M, s \models \langle a \rangle \langle b \rangle \bot$
7. $M, s \models [a] \langle a \rangle [a] [b] \bot$
8. $M, s \models \langle a \rangle (\langle a \rangle \top \wedge \langle b \rangle \top)$
9. $M, s \models [a] (\langle a \rangle \top \vee \langle b \rangle \top)$
10. $M, s \models \langle a \rangle ([b][a] \bot \wedge \langle b \rangle \top)$



Exercício

Determine:

1. $[\![[a][b]\bot]\!]_M$
2. $[\![\langle a\rangle(\langle a\rangle\top \wedge \langle b\rangle\top)]\!]_M$
3. $[\![[a][a][b]\bot]\!]_M$
4. $[\![[a](\langle a\rangle\top \vee \langle b\rangle\top)]\!]_M$



Exercise

Find a model M for $(\{\}, \{a, b, c\})$ that has a state w such that simultaneously:

- $M, w \models \langle a \rangle (\langle b \rangle \langle c \rangle \top \wedge \langle c \rangle \top)$
- $M, w \models \langle a \rangle \langle b \rangle ([a] \bot \wedge [b] \bot \wedge [c] \bot)$
- $M, w \models [a] \langle b \rangle ([c] \bot \wedge \langle a \rangle \top)$

Bisimulation

Bisimulation (Multimodal version)

Let $M = (W, R, V)$ and $M' = (W', R', V')$ be two models for $(\mathrm{Prop}, \mathrm{Act})$. A **bisimulation between M and M'** is a relation $B \subseteq W \times W'$ such that, for any $(w, w') \in B$ **and for any** $a \in \mathrm{Act}$, the following conditions hold:

- (Atom) $w \in V(p)$ iff $w' \in V'(p)$, for all $p \in \mathrm{Prop}$
- (Zig) if $(w, v) \in R_a$ then there exists a $v' \in W'$ such that $(w', v') \in R'_a$ and $(v, v') \in B$
- (Zag) if $(w', v') \in R'_a$ then there exists a $v \in W$ such that $(w, v) \in R_a$ and $(v, v') \in B$

Hennessy-Milner Theorem (Multimodal Version)

Image-finite model

A model $M = (W, R, V)$ is called **image-finite** if for every $w \in W$, and for every $a \in \mathrm{Act}$, the set $R_a[w] = \{v \mid (w, v) \in R\}$ is finite.

Hennessy-Milner Theorem (Multimodal Version)

Image-finite model

A model $M = (W, R, V)$ is called **image-finite** if for every $w \in W$, and for every $a \in \mathrm{Act}$, the set $R_a[w] = \{v \mid (w, v) \in R\}$ is finite.

Hennessy-Milner Theorem

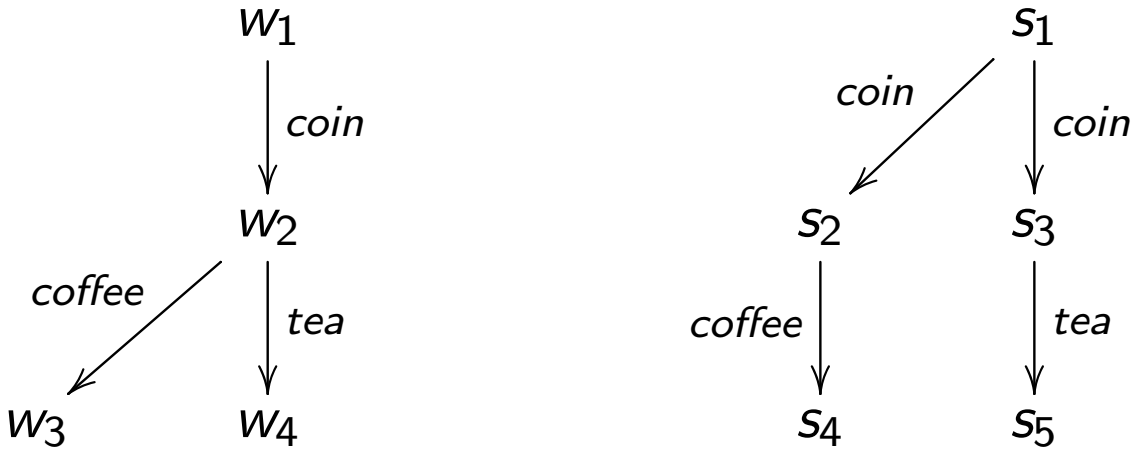
Let M and M' be two image-finite models for $(\mathrm{Prop}, \mathrm{Act})$. Then, for any $w \in W$ and $w' \in W'$, the following are equivalent:

1. There exists a bisimulation $B : M \rightleftharpoons M'$ such that $(w, w') \in B$
2. For every $\varphi \in \mathrm{MFm}(\mathrm{Prop}, \mathrm{Act})$,

$$M, w \models \varphi \text{ iff } M', w' \models \varphi$$

Examples

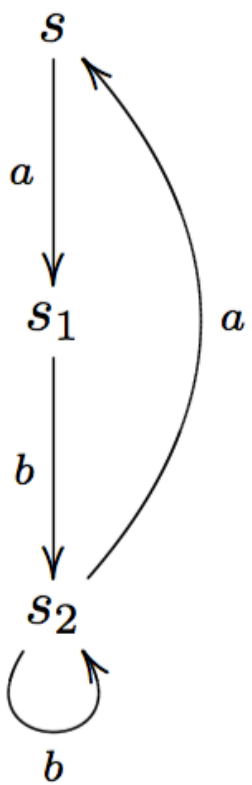
Two coffee machines

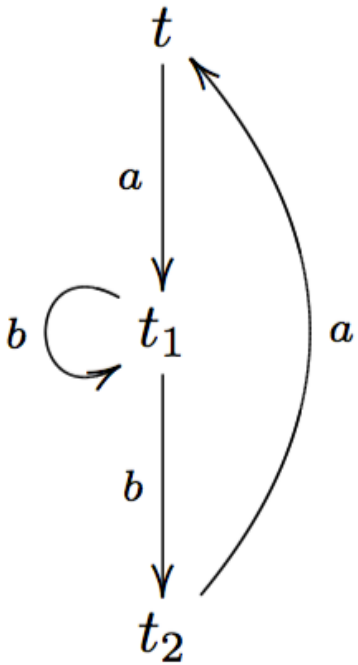


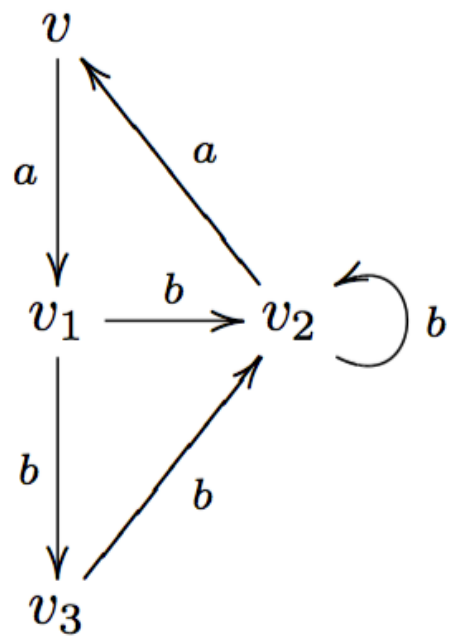
We have that $w_1 \not\sim s_1$, because:

- $M, w_1 \models [coin]\langle coffee\rangle\top$, and
- $N, s_1 \not\models [coin]\langle coffee\rangle\top$

Exercise







Show that $s \not\leftrightarrow t \not\leftrightarrow v$.

Example – Temporal Logics

Temporal Logic

- W is the set of time points
- There is a unique modality corresponding to the **transitive closure of the "next-time" relation**

Example – Temporal Logics

Temporal Logic

- W is the set of time points
- There is a unique modality corresponding to the **transitive closure of the "next-time" relation**

(Until) $\mathcal{M}, w \models \phi \mathcal{U} \psi$ iff

There exists a $v \in W$ such that $(w, v) \in R$ and $\mathcal{M}, v \models \psi$, and for all $u \in W$ such that $(w, u) \in R$ and $(u, v) \in R$, we have that $\mathcal{M}, u \models \phi$

(Since) $\mathcal{M}, w \models \phi \mathcal{S} \psi$ iff

There exists a $v \in W$ such that $(v, w) \in R$ and $\mathcal{M}, v \models \psi$, and for all u such that $(v, u) \in R$ and $(u, w) \in R$, we have that $\mathcal{M}, u \models \phi$

Operationalizing Multimodal Logic

We can use **sets of actions in modalities**

$$M, w \models \langle K \rangle \phi \quad \text{iff} \quad \exists_{w \in \{w' | (w,w') \in R_a \text{ and } a \in K\}} . M, w' \models \phi$$

$$M, w \models [K] \phi \quad \text{iff} \quad \forall_{w \in \{w' | (w,w') \in R_a \text{ and } a \in K\}} . M, w' \models \phi$$

Operationalizing Multimodal Logic

We can use **sets of actions in modalities**

$$M, w \models \langle K \rangle \phi \quad \text{iff} \quad \exists_{w \in \{w' \mid (w,w') \in R_a \text{ and } a \in K\}} . M, w' \models \phi$$

$$M, w \models [K] \phi \quad \text{iff} \quad \forall_{w \in \{w' \mid (w,w') \in R_a \text{ and } a \in K\}} . M, w' \models \phi$$

Notation

The following is used:

- The symbol $-$ to represent $K = \mathrm{Act}$.
  E.g. $\langle - \rangle \varphi$
- The expression $-A$ to represent $K = \mathrm{Act} \setminus A$.
  E.g. $[-A]\varphi$. Parentheses are omitted in singular sets. For example, $[-a]\varphi$ denotes $[-\{a\}]\varphi$

Operationalizing Multimodal Logic

Typical Properties

- **inevitability of** a: $\langle - \rangle \top$ and $[-a] \bot$

Operationalizing Multimodal Logic

Typical Properties

- **inevitability of** a: $\langle - \rangle \top$ and $[-a] \bot$
- **progress**: $\langle - \rangle \top$

Operationalizing Multimodal Logic

Typical Properties

- **inevitability of** a: $\langle - \rangle \top$ and $[-a] \bot$
- **progress**: $\langle - \rangle \top$
- **deadlock or termination**: $[-] \bot$

Illustration

Taxi Network Specification

Specify in Multimodal Logic the scenario of a taxi network described below. Pay attention to the signature definition and then to the specification of the relevant requirements.

- $\phi_0 =$ **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**
- $\phi_1 =$ **This property applies only to cars in service.**
- $\phi_2 =$ **If a car is allocated to a service, it must first pick up the passenger and then plan the route.**
- $\phi_3 =$ **When an emergency is detected, a taxi becomes inactive.**
- $\phi_4 =$ **A car in service is not inactive.**

Illustration

Taxi Network Specification

- $\phi_0 =$ **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**

Illustration

Taxi Network Specification

- $\phi_0 =$ **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**
  - $\phi_0 = \langle rec, alo \rangle \top$

Illustration

Taxi Network Specification

- $\phi_0 =$ **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**
  - $\phi_0 = \langle rec, alo \rangle \top$
- $\phi_1 =$ **This property applies only to cars in service.**

Illustration

Taxi Network Specification

- $\phi_0 =$ **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**
  - $\phi_0 = \langle rec, alo \rangle \top$
- $\phi_1 =$ **This property applies only to cars in service.**
  - $\phi_1 = [onservice] \langle rec, alo \rangle \top$ or $\phi_1 = [onservice] \phi_0$

Illustration

Taxi Network Specification

- ϕ_0 = **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**
  - $\phi_0 = \langle rec, alo \rangle \top$
- ϕ_1 = **This property applies only to cars in service.**
  - $\phi_1 = [onservice] \langle rec, alo \rangle \top$ or $\phi_1 = [onservice] \phi_0$
- ϕ_2 = **If a car is allocated to a service, it must first pick up the passenger and then plan the route.**

Illustration

Taxi Network Specification

- ϕ_0 = **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**
  - $\phi_0 = \langle rec, alo \rangle \top$
- ϕ_1 = **This property applies only to cars in service.**
  - $\phi_1 = [onservice] \langle rec, alo \rangle \top$ or $\phi_1 = [onservice] \phi_0$
- ϕ_2 = **If a car is allocated to a service, it must first pick up the passenger and then plan the route.**
  - $\phi_2 = [alo] \langle rec \rangle \langle plan \rangle \top$
- ϕ_3 = **When an emergency is detected, a taxi becomes inactive.**

Illustration

Taxi Network Specification

- ϕ_0 = **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**
  - $\phi_0 = \langle rec, alo \rangle \top$
- ϕ_1 = **This property applies only to cars in service.**
  - $\phi_1 = [onservice] \langle rec, alo \rangle \top$ or $\phi_1 = [onservice] \phi_0$
- ϕ_2 = **If a car is allocated to a service, it must first pick up the passenger and then plan the route.**
  - $\phi_2 = [alo] \langle rec \rangle \langle plan \rangle \top$
- ϕ_3 = **When an emergency is detected, a taxi becomes inactive.**
  - $\phi_3 = [sos] [-] \bot$

Illustration

Taxi Network Specification

- $\phi_0 =$ **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**
  - $\phi_0 = \langle rec, alo \rangle \top$
- $\phi_1 =$ **This property applies only to cars in service.**
  - $\phi_1 = [onservice] \langle rec, alo \rangle \top$ or $\phi_1 = [onservice] \phi_0$
- $\phi_2 =$ **If a car is allocated to a service, it must first pick up the passenger and then plan the route.**
  - $\phi_2 = [alo] \langle rec \rangle \langle plan \rangle \top$
- $\phi_3 =$ **When an emergency is detected, a taxi becomes inactive.**
  - $\phi_3 = [sos] [-] \bot$
- $\phi_4 =$ **A car in service is not inactive.**

Illustration

Taxi Network Specification

- ϕ_0 = **In a taxi network, a car can pick up a passenger or be allocated by the Dispatch to a pending service.**
  - $\phi_0 = \langle rec, alo \rangle \top$
- ϕ_1 = **This property applies only to cars in service.**
  - $\phi_1 = [onservice] \langle rec, alo \rangle \top$ or $\phi_1 = [onservice] \phi_0$
- ϕ_2 = **If a car is allocated to a service, it must first pick up the passenger and then plan the route.**
  - $\phi_2 = [alo] \langle rec \rangle \langle plan \rangle \top$
- ϕ_3 = **When an emergency is detected, a taxi becomes inactive.**
  - $\phi_3 = [sos] [-] \bot$
- ϕ_4 = **A car in service is not inactive.**
  - $\phi_4 = [onservice] \langle - \rangle \top$

Exercise

Formalise each of the following properties:

1. The occurrence of a and b is impossible.
2. The occurrence of a followed by b is impossible.
3. Only the occurrence of a is possible.
4. Once a occurred, b or c may occur.
5. After a occurred followed by b, c may occur.
6. Once a occurred, b or c may occur but not both.
7. a cannot occur before b.
8. There is only an initial transition labelled by a.

Exercise

Consider the following process

$$\begin{aligned} Start &=^{df} fw.Go + stop.\mathbf{0} \\ Go &=^{df} fw.bk.bk.Start + right.left.bk.Start \end{aligned}$$

Formalize the following properties

1. After *fw* another *fw* is immediately possible
2. After *fw* followed by *right*, *left* is possible but *bk* is not.
3. The action *fw* is the only possible one.
4. The third action is not *fw*.

Outline

A *Naïve* Approach

Is Multi-modal Logic suitable for reasoning about programs?

- considering one modality for each program in the language
- modeling the computation universe as the transition system that interprets all these programs

Dynamic Logic(s)

- Multi-modal logics designed to **work with actions in a structured way**
- Goal: **Reason about programs**
- Ingredients:
  - **Atomic program** notion
  - **Regular expressions** over atomic programs
  - **Testing mechanisms** for dealing with conditionals:
    e.g. *if _then_else_*
- These principles are sufficiently abstract to be **adapted to various computing paradigms** ...

Intuitions: Dynamic Logic for Verifying Imperative Programs

- To handle classical imperative programs, what is the notion of an atomic program, i.e., what is the set Π_0?
- What is the notion of **state**?
- What is the notion of **test**?

Let π be the following program:

```
while x<3 do
 x:=x+1
od
```

Example:

If $x = 0$, any execution of π, if it terminates, results in a state where $x = 2$

$$x = 0 \rightarrow [((x < 3?); x := x + 1)^*; (x \geq 3?)]x = 2$$

The Dynamic Logic We Will Consider in This Course:

- Atomic programs — sets of actions Act
- Tests — assertions in our logic
- Valuations allow the representation of **local observations**, i.e., what we can observe beyond the dynamics

Generic Notion of Program and Its Interpretation

Set of Programs for Atomic Program Set Act

$$\pi := a \mid \pi;\pi \mid \pi + \pi \mid \pi^* \mid \varphi?$$

$a \in \text{Act}$ and φ a **"state property"**

Interpretation of Programs

How do we interpret these programs in a model (Act, Prop)-model $M = (W, R, V)$?

- A program π will be interpreted as a relation $Pr_\pi \subseteq W \times W$ recursively.

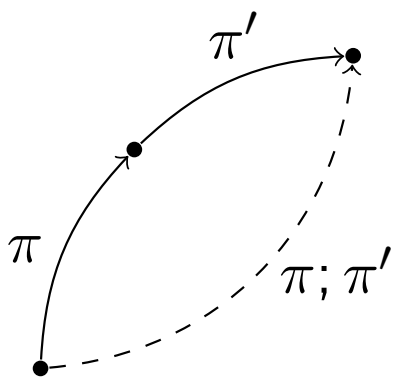
Atomic Program

$Pr_a = R_a,\ a \in \text{Act}$

Interpretation of Programs

Sequential Program

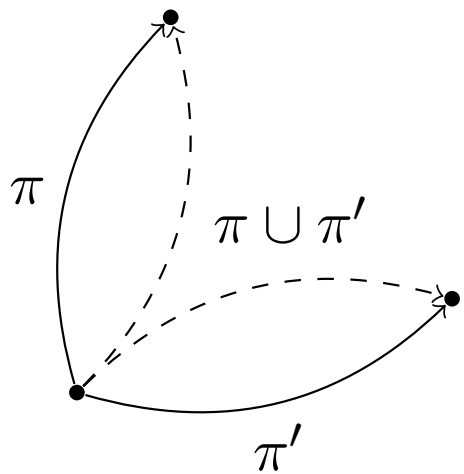
- $Pr_{\pi;\pi'} = Pr_{\pi} \circ Pr_{\pi'}$



Interpretation of Programs

Non deterministic choice

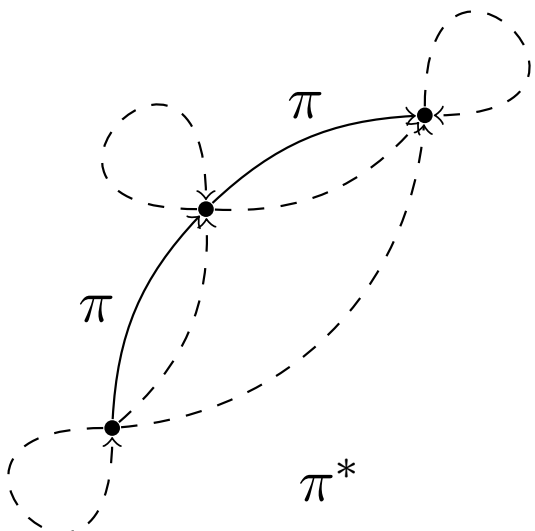
- $Pr_{\pi+\pi'} = Pr_{\pi} \cup Pr_{\pi'}$



Interpretation of Programs

Iteractive closure

- $Pr_{\pi^*} = (Pr_\pi)^*$, para $(Pr_\pi)^* = \bigcup_{n \geq 0} (Pr_\pi)^n$, onde
  - $(w, w') \in (Pr_\pi)^0$ se $w = w'$
  - $(w, w') \in (Pr_\pi)^{k+1}$ se $(w, w') \in (Pr_\pi)^k \circ (Pr_\pi)$



Interpretation of Programs

Test

- $Pr_{\varphi?} = \{(w, w) \mid M, w \models \varphi\}$

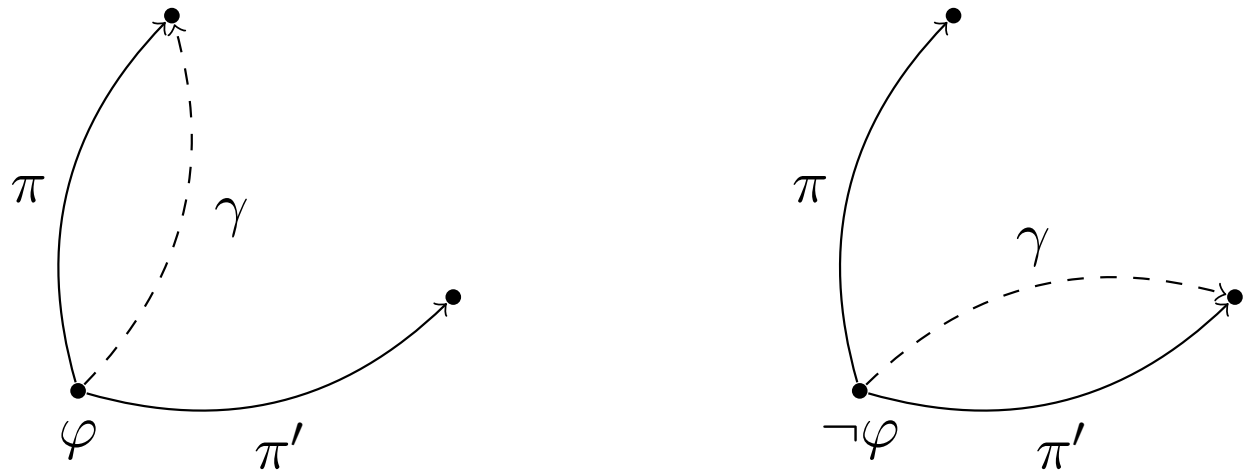
Exercise

Express the standard commands of imperative programming as terms of our algebra of programs. Namely:

- **if** φ **then** π **else** π'
- **while** φ **do** π **od**
- **repeat** π **until** φ

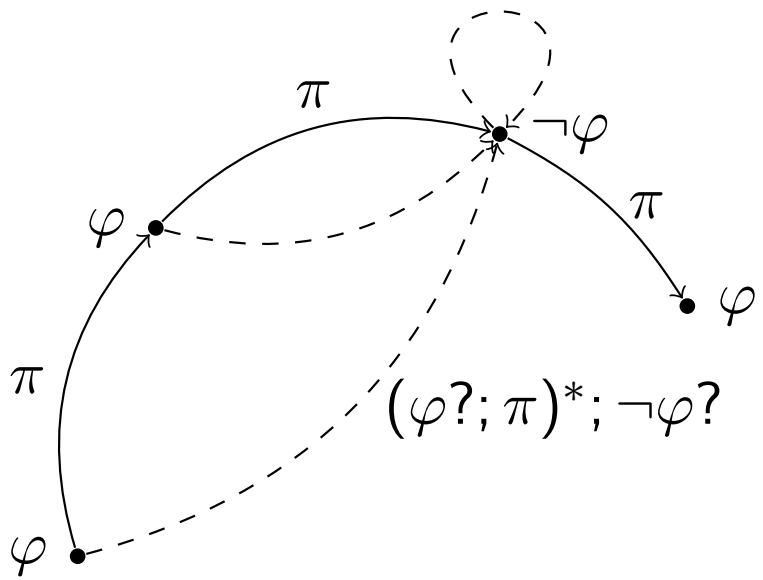
Programs interpretation - abbreviatures

if φ then π else π' $\equiv (\varphi?;\pi) + (\neg\varphi?;\pi')$



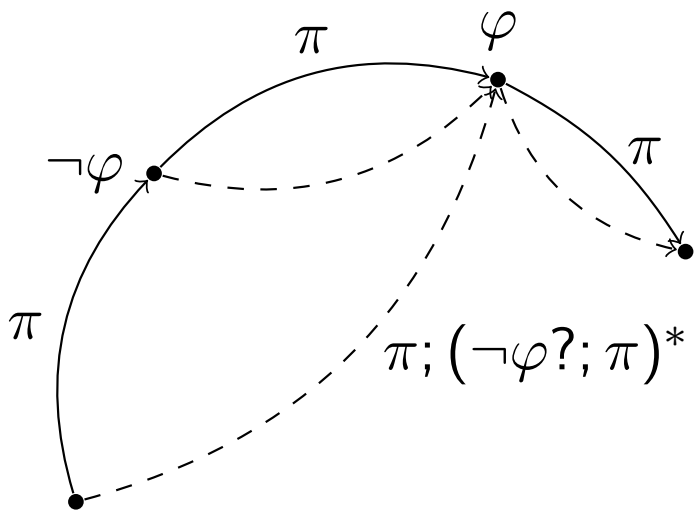
Programs interpretation - abbreviatures

while φ do π od $\equiv$ $(\varphi?;\pi)^*;\neg\varphi?$



Programs interpretation - abbreviatures

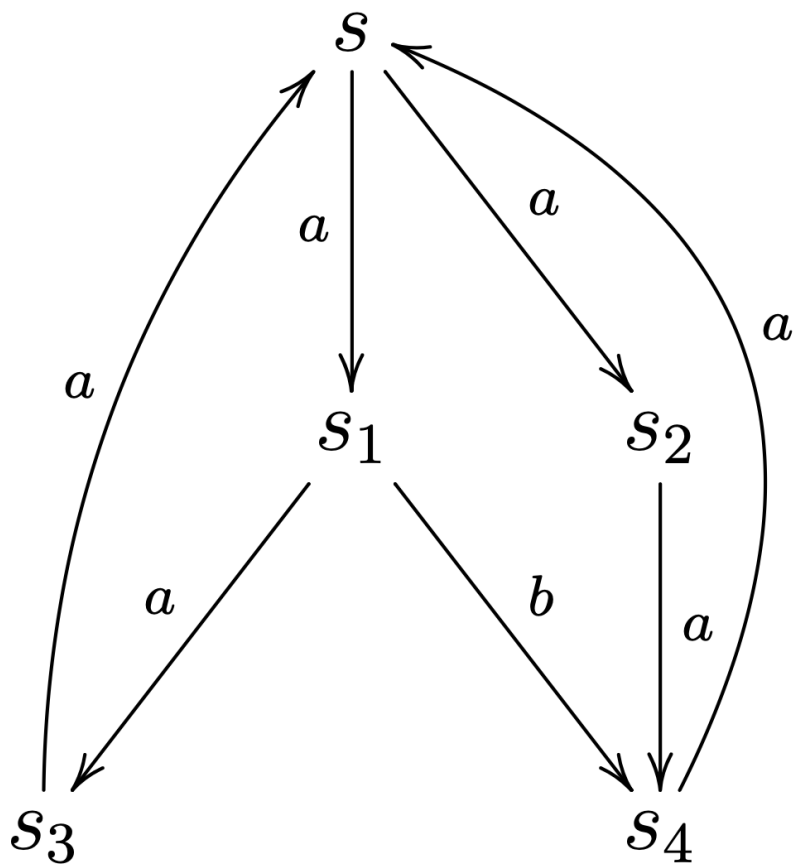
repeat π **until** $\varphi \equiv \quad \pi;(\neg\varphi?;\pi)^*$



Exercise

Consider the $(\{a, b\}, \{p, q\})$-model M represented in the left, such that $V(p) = \{s_1, s_3\}$ e $V(q) = \{s, s_2, s_4\}$.
Interpret the following programs in M:

- $a; b$
- $b; a$
- $a + b$
- $(a; b) + b$
- a^*
- $(p?); a$
- $(q?); a + (\neg q?)b$
- $(a + b)^*$
- $(p \wedge q)?$



Dynamic Logic

(Act, Prop)-formulas

Formulas:

$\varphi ::= \quad p \quad | \quad \bot \quad | \quad \varphi \to \varphi \quad | \quad [\pi]\varphi$, para $p \in \text{Prop}$

Programs:

$\pi ::= \quad a \quad | \quad \pi;\pi \quad | \quad \pi + \pi \quad | \quad \pi^* \quad | \quad \varphi?$, with $a \in \text{Act}$

Satisfaction Relation in DL

Satisfaction for a model $\mathcal{M}$ at state w

$$
\begin{array}{lll}
M, w \models \top & & \\
M, w \models p & \text{sse} & w \in V(p) \\
M, w \models \phi_1 \rightarrow \phi_2 & \text{iff} & M, w \not\models \phi_1 \text{ or } M, w \models \phi_2 \\
M, w \models [\pi]\,\phi & \text{iff} & \forall v \in W.\ (w, v) \in Pr_\pi \textbf{ implies } M, v \models \phi
\end{array}
$$

Satisfaction Relation in DL

Satisfaction for a model $\mathcal{M}$ at state w

$$
\begin{array}{lll}
M, w \models \top & & \\
M, w \models p & \text{sse} & w \in V(p) \\
M, w \models \phi_1 \rightarrow \phi_2 & \text{iff} & M, w \not\models \phi_1 \text{ or } M, w \models \phi_2 \\
M, w \models [\pi]\, \phi & \text{iff} & \forall v \in W.\ (w, v) \in Pr_\pi \textbf{ implies } M, v \models \phi
\end{array}
$$

Corollary:

$$
\begin{array}{lll}
M, w \not\models \bot & & \\
M, w \models \neg\phi & \text{iff} & M, w \not\models \phi \\
M, w \models \phi_1 \wedge \phi_2 & \text{iff} & M, w \models \phi_1 \text{ and } M, w \models \phi_2 \\
M, w \models \phi_1 \vee \phi_2 & \text{iff} & M, w \models \phi_1 \text{ or } M, w \models \phi_2 \\
M, w \models \langle\pi\rangle\, \phi & \text{iff} & \exists\, v \in W \textbf{ such that } (w, v) \in Pr_\pi \textbf{ e } M, v \models \phi
\end{array}
$$

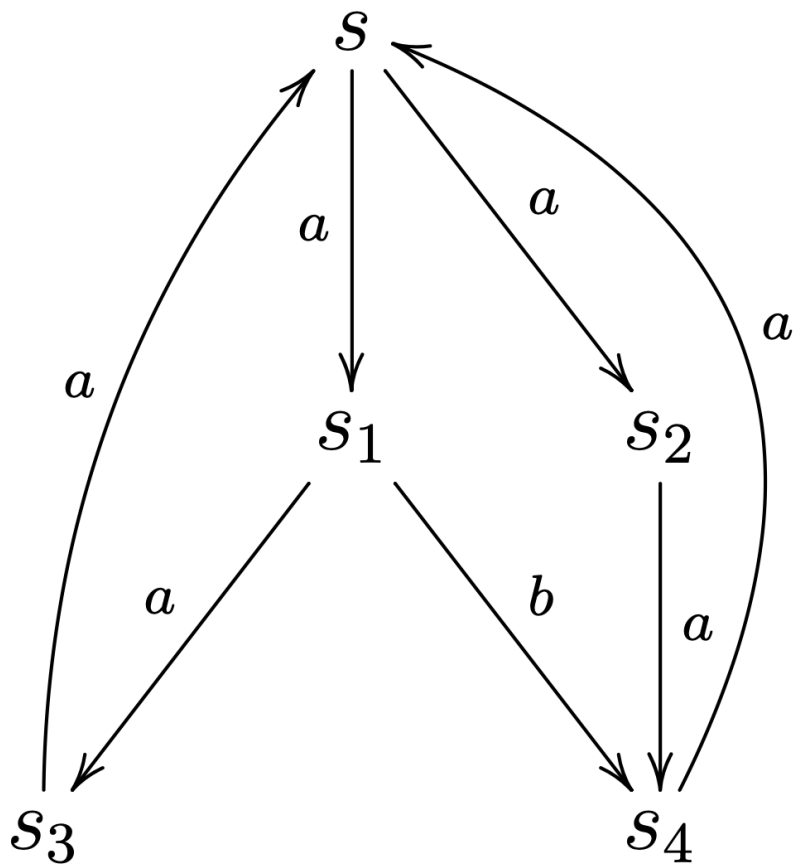
Exercise

Consider the $(\{a, b\}, \{p, q\})$-model M in the left, assuming $V(p) = \{s_1, s_3\}$ e $V(q) = \{s, s_2, s_4\}$.
What are the correct statements:

- $M, s \models \langle a^* \rangle q$
- $M, s_1 \models \langle p?; a; b + q?; a; b \rangle \top$
- $M \models [(p \wedge q)?] \bot$

Extend the operator $[\![\_]\!]_M$ to the multi-modal case and calculate:

- $[\![\langle p?; a; b + q?; a; b \rangle \top]\!]_M$
- $[\![[a^*]q]\!]_M$



Exercise

Consider the $(\{a, b, c\}, \{p, q\})$-model $M = (W, R, V)$, with $W = \{w_1, w_2, w_3, w_4, w_5\}$ and such that :

- $V(p) = \{w_1, w_3\}$ e $V(q) = W$,
- $R_a = \{(w_1, w_3), (w_1, w_4), (w_1, w_5), (w_2, w_3), (w_5, w_3)\}$
- $R_b = \{(x, y) \in W^2 | x = y\}$,
- $R_c = \{(w_1, x) | x \in W\}$

Check if:

- a) $M, w_1 \models [(a; b)]p \vee [b^* + c]q$
- b) $M, w_3 \models [q?; b]p \rightarrow [c]\neg q$

Exercise

Consider the $(\{a, b, c\}, \{p, q\})$-model $M = (W, R, V)$, with $W = \{-2, -1, 0, 1, 2\}$ and such that:

- $V(p) = \{x \in W | x > 0\}$ and $V(q) = \{x \in W | x \leq 1\}$,
- $R_a = \{(x, y) \in W^2 | x \leq 0, y \geq 0\}$
- $R_b = \{(x, y) \in W^2 | x = y\}$
- $R_c = \{(0, x) | x \in W\}$

Check if:

- a) $M, 0 \models [(a + b)]p \vee [b^* + c]q$
- b) $M, 2 \models [(p \rightarrow q)?; b]p \rightarrow [c]\neg q$

Exercise

Verify if the following properties are valid in DL

- $[\alpha;\beta]\varphi \leftrightarrow [\alpha][\beta]\varphi$
- $[\alpha+\beta]\varphi \leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi$
- $[\alpha^*]\varphi \rightarrow \varphi \wedge [\alpha][\alpha]^*\varphi$
- $[\alpha^*](\varphi \rightarrow [\alpha]\varphi) \rightarrow (\varphi \rightarrow [\alpha^*]\varphi)$
- $[\varphi?]\psi \leftrightarrow (\varphi \rightarrow \psi)$

An Hennessy Milner Theorem for PDL?

Exercise

- Observe that the semantics of dynamic logic for a $(\mathrm{Prop}, \mathrm{Act})$-model M is equivalent to the semantics of multimodal logic in the "respective" $(\mathrm{Prop}, \mathit{Prog}(\mathrm{Act}))$-model $\bar{M}$
- Verify that $B : M \rightleftharpoons N$ **if and only if** $B : \bar{M} \rightleftharpoons \bar{N}$
- Conclude with the knowledge you have of (multi)modal logic

More Operational Version: Action Sets as Atomic Programs

$$\alpha := K \mid K \cup K \mid K \cap K$$

for $K \subseteq \text{Act}$.

More Operational Version: Action Sets as Atomic Programs

$$\alpha := K \mid K \cup K \mid K \cap K$$

for $K \subseteq \mathrm{Act}$. Just like in the multimodal case, we represent:

- the set Act by $-$
- the set $A \setminus \{a\}$ by $-a$

Programs

$$R := \epsilon \mid \alpha \mid R.R \mid R + R \mid R^*$$

Hennessy-Milner with regular modalities

on regular modalities

$\langle R_1 + R_2 \rangle \varphi \leftrightarrow \langle R_1 \rangle \varphi \vee \langle R_2 \rangle \varphi$

$[R_1 + R_2] \varphi \leftrightarrow [R_1] \varphi \wedge [R_2] \varphi$

$\langle R_1 . R_2 \rangle \varphi \leftrightarrow \langle R_1 \rangle \langle R_2 \rangle \varphi$

$[R_1 . R_2] \varphi \leftrightarrow [R_1][R_2] \varphi$

Representation of More Complex Patterns

- The property φ is true in all reachable states.

Representation of More Complex Patterns

- The property φ is true in all reachable states.

$$[-^*]\varphi$$

- The property φ is always accessible through action a.

Representation of More Complex Patterns

- The property φ is true in all reachable states.

$$[-^*]\varphi$$

- The property φ is always accessible through action a.

$$[-^*]\langle a\rangle\varphi$$

- The property φ is inevitable.

Representation of More Complex Patterns

- The property φ is true in all reachable states.

$$[-^*]\varphi$$

- The property φ is always accessible through action a.

$$[-^*]\langle a\rangle\varphi$$

- The property φ is inevitable.

$$[-^*]\langle -\rangle\varphi$$

Representation of More Complex Patterns

- As long as an *error* does not happen, a **deadlock** will not occur.

Representation of More Complex Patterns

- As long as an *error* does not happen, a **deadlock** will not occur.

$$[(-error)^*]\langle - \rangle \top$$

Representation of More Complex Patterns

- As long as an *error* does not happen, a **deadlock** will not occur.

$$[(-error)^*]\langle - \rangle \top$$

- Whenever an a happens in a reachable state, an action b can be subsequently performed, unless an c happens, cancelling the need to perform b.

Representation of More Complex Patterns

- As long as an *error* does not happen, a **deadlock** will not occur.

$$[(-error)^*]\langle - \rangle \top$$

- Whenever an a happens in a reachable state, an action b can be subsequently performed, unless an c happens, cancelling the need to perform b.

$$[-^*.a]\langle -^*.(b \cup c)\rangle \top$$

Representation of More Complex Patterns

- As long as an *error* does not happen, a **deadlock** will not occur.

$$[(-error)^*]\langle - \rangle \top$$

- Whenever an a happens in a reachable state, an action b can be subsequently performed, unless an c happens, cancelling the need to perform b.

$$[-^*.a]\langle -^*.(b \cup c)\rangle \top$$

- Whenever action a occurs, it must always be possible to do b afterward, although doing b can be infinitely postponed.

Representation of More Complex Patterns

- As long as an *error* does not happen, a **deadlock** will not occur.

$$[(-error)^*]\langle - \rangle \top$$

- Whenever an a happens in a reachable state, an action b can be subsequently performed, unless an c happens, cancelling the need to perform b.

$$[-^*.a]\langle -^*.(b \cup c)\rangle \top$$

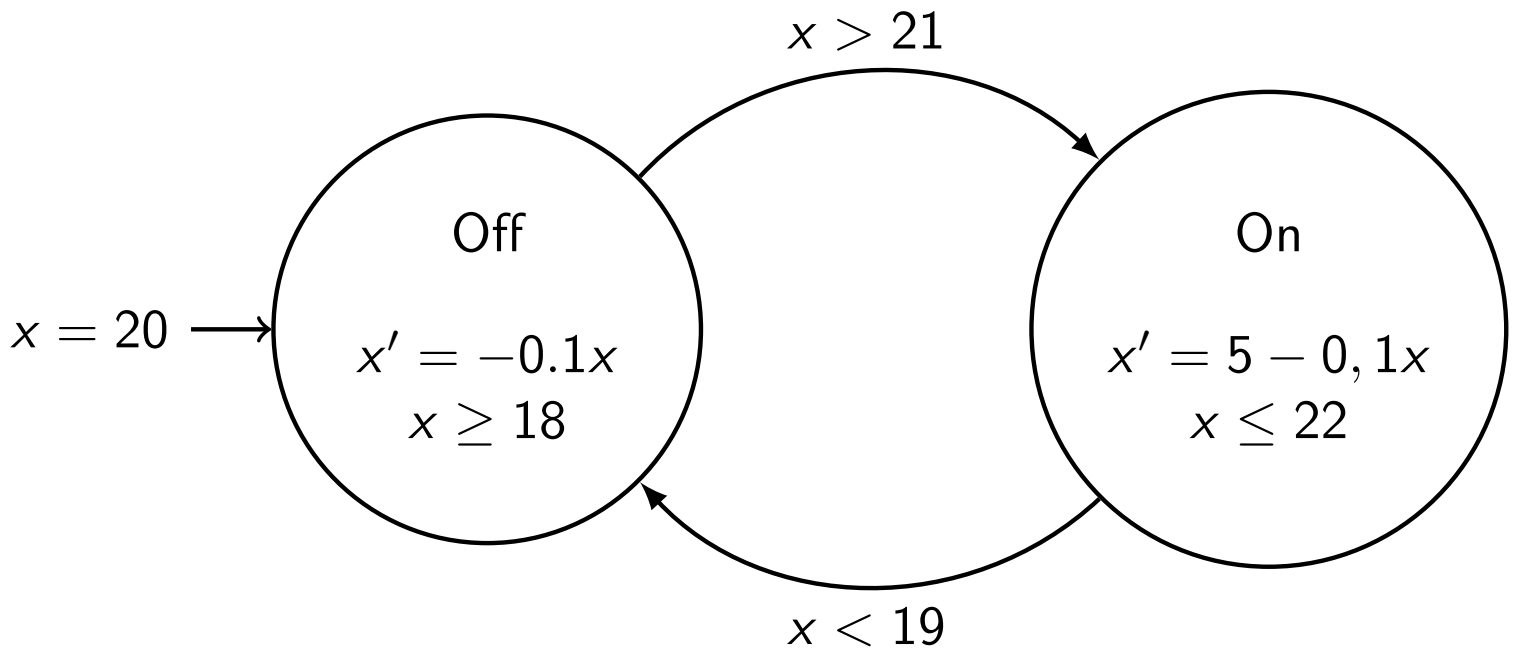
- Whenever action a occurs, it must always be possible to do b afterward, although doing b can be infinitely postponed.

$$[-^*.a.(-b)^*]\langle -^*.b\rangle \top$$

Outline

The Hybrid Automaton

The Thermostat

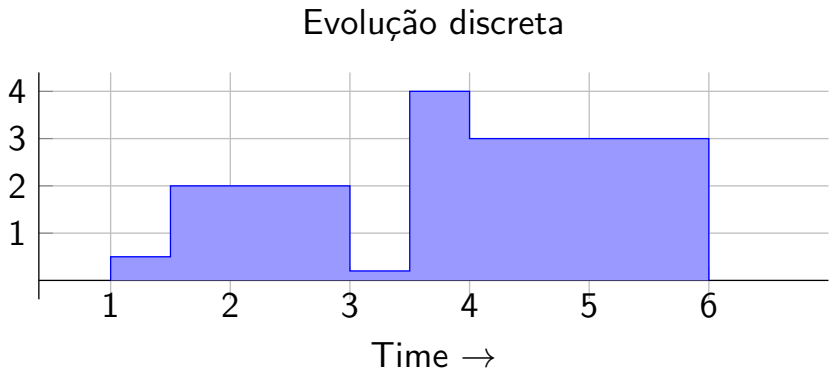


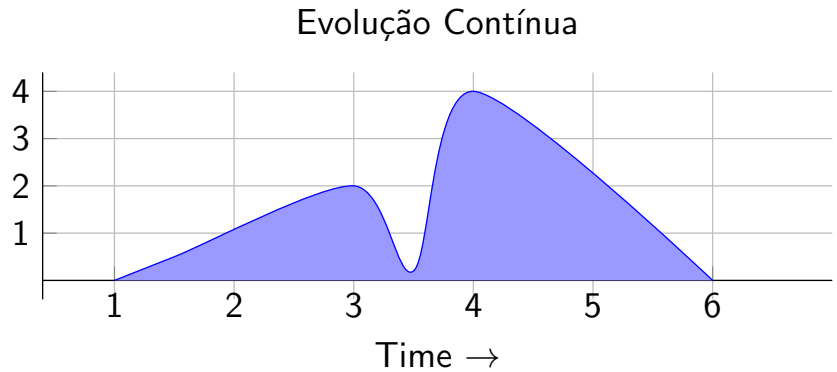
Dynamic Logic for Hybrid Systems?

André Platzer's Differential Dynamic Logic $d\mathcal{L}$

- A logic developed to **specify** and **verify properties** of hybrid systems
- It has a powerful **computational proof support** — KeYmaera

Evoluções Discretas vs. Contínuas





Hybrid = discretep + **contínuo**

- digital controller actions, discrete event interaction, etc
- **physics entities, analogic controller actions, etc**

Syntax of $d\mathcal{L}$

Hybrid Programs

$$\alpha, \beta \ni x := \theta \mid x' = \theta \,\&\, \chi \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \mid ?\chi$$

Syntax of $d\mathcal{L}$

Hybrid Programs

$$\alpha, \beta \ni x := \theta \mid x' = \theta \,\&\, \chi \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \mid ?\chi$$

$d\mathcal{L}$-formulas

$$\phi, \psi \ni \theta_1 = \theta_2 \mid \theta_1 \leq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid [\alpha]\phi$$

where θ, θ_1 and θ_2 are terms

Platzer's $d\mathcal{L}$ – semantics

States:

They are functions $\mathcal{V} \to \mathbb{R}$

Interpretation of Programs

The relation $\rho(\alpha) \subseteq \mathcal{S} \times \mathcal{S}$ is defined as in first-order DL with

- $\rho(x := \theta) = \{(u, v) | v(x) = \theta \text{ for all } y \in \mathcal{V} \setminus \{x\}, u(y) = v(y)\}$
- $\rho(x' = \theta \& \chi) = \{(\varphi(0), \varphi(r)) | \varphi(t) \models \chi, 0 \leq t \leq r, \text{ for every solution } \varphi : [0, r] \to \mathcal{S} \text{ with any duration } r\}$
- $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
- $\rho(\alpha; \beta) = \rho(\alpha) \circ \rho(\beta)$
- $\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n)$, where $\alpha^0 = id$ and $\alpha^{n+1} = \alpha; \alpha^n$
- $\rho(?\chi) = \{(v, v) | v \models \chi\}$

Platzer's $d\mathcal{L}$ – satisfaction

- $v \models (\theta_1 = \theta_2)$ iff $v_{\theta_1} = v_{\theta_2}$
- $v \models \neg\rho$ iff $v \not\models \rho$
- $v \models \rho \wedge \rho'$ iff $v \models \rho$ and $v \models \rho'$
- $v \models \rho \vee \rho'$ iff $v \models \rho$ or $v \models \rho'$
- $v \models [\alpha]\rho$ iff for every $(v, w) \in \rho(\alpha)$, $w \models \rho$
- $v \models \langle\alpha\rangle\rho$ iff there exists a $(v, w) \in \rho(\alpha)$, such that $w \models \rho$

Biology Case Study

Controller of a Biological System

Example

$\begin{cases} x' = 5 - x \\ y' = 6 - y + u \end{cases}$ $x < 3 \wedge y \geq 2$	$\begin{cases} x' = 6 - x \\ y' = 1 - y + u \end{cases}$ $x \geq 3 \wedge y \geq 2$
$\begin{cases} x' = -x \\ y' = 5 - y + u \end{cases}$ $x < 3 \wedge y < 2$	$\begin{cases} x' = 1 - x \\ y' = -y + u \end{cases}$ $x \geq 3 \wedge y < 2$

D. Figueiredo, Manuel Martins and M. Chaves.
Applying differential dynamic logic to reconfigurable biological networks,
Mathematical Biosciences, vol. 291, 10-20, 2017.

Biological Examples

Controller of a Biological System

We analyze the *steady states*

i.e., the values of x and y to which the system tends.

Control:

- $u = 2$, if $x \geq 3$ and $t \geq 2$
- $u = 0$, otherwise

Using numerical methods, we know that $(x, y) = (6, 3)$ is a candidate

Biological Example

Controller in a Biological System

$\begin{cases} x' = 5 - x \\ y' = 6 - y + u \end{cases}$ $x < 3 \wedge y \geq 2$	$\begin{cases} x' = 6 - x \\ y' = 1 - y + u \end{cases}$ $x \geq 3 \wedge y \geq 2$
$\begin{cases} x' = -x \\ y' = 5 - y + u \end{cases}$ $x < 3 \wedge y < 2$	$\begin{cases} x' = 1 - x \\ y' = -y + u \end{cases}$ $x \geq 3 \wedge y < 2$

- $\alpha_1 \equiv (?x < 3 \wedge y < 2; u := 0;$
  $(x' = -x, y' = 5 - y + u, \tau' = 1 \,\&\, x \leq 3 \wedge y \leq 2))$
- ...
- $\alpha_4 \equiv (?x \geq 3 \wedge y \geq 2; u := 2;$
  $(x' = 6 - x, y' = 1 - y + u, \tau' = 1 \,\&\, x \geq 3 \wedge y \geq 2))$

Biological Example

Controller in a Biological System

The evolution of the system can be described by the following hybrid program:

$$\alpha \equiv \alpha_1 \cup \alpha_2 \cup \alpha_3 \cup \alpha_4$$

Biological Example

Controller in a Biological System

The evolution of the system can be described by the following hybrid program:

$$\alpha \equiv \alpha_1 \cup \alpha_2 \cup \alpha_3 \cup \alpha_4$$

$(x, y) = (6, 3)$ is a steady state:

$$\exists c > 0(\forall 0 < k < c((x-6)^2+(y-3)^2 = k \wedge \tau = 0 \rightarrow [\alpha^*](\tau = 0 \vee (x-6)^2+(y-$$

Outline

The mCRL2 Toolset

The so-called **Process Algebras** are formalisms for the specification of complex transition systems (typically involving interaction and concurrency).
The **mCRL2** offers:

- a **process algebra**, based on ACP (Bergstra & Klop, 1982)

The mCRL2 Toolset

The so-called **Process Algebras** are formalisms for the specification of complex transition systems (typically involving interaction and concurrency).
The **mCRL2** offers:

- a **process algebra**, based on ACP (Bergstra & Klop, 1982)
- with an axiomatic semantics

THE MCRL2 TOOLSET

The so-called **Process Algebras** are formalisms for the specification of complex transition systems (typically involving interaction and concurrency).
The **mCRL2** offers:

- a **process algebra**, based on ACP (Bergstra & Klop, 1982)
- with an axiomatic semantics
- and a Dynamic Logic used for the **specification** of properties over these systems

The mCRL2 Toolset

The so-called **Process Algebras** are formalisms for the specification of complex transition systems (typically involving interaction and concurrency).
The **mCRL2** offers:

- a **process algebra**, based on ACP (Bergstra & Klop, 1982)
- with an axiomatic semantics
- and a Dynamic Logic used for the **specification** of properties over these systems
- tools for **simulation** and **verification**

`www.mcrl2.org`

ACTIONS

INTERACTION THROUGH SETS OF MULTI-ACTIONS

- A multi-action is the basic unit of interaction that **executes atomically**.

$$\alpha \quad ::= \quad \tau \;\mid\; a \;\mid\; a(d) \;\mid\; \alpha \mid \alpha$$

- Actions can be parameterized by data.
- The structure $\langle \mathrm{Act}, |, \tau \rangle$ forms an **Abelian monoid**.

SEQUENTIAL PROCESSES

NON-DETERMINISTIC SEQUENTIAL BEHAVIOR

The set of **processes** $\mathbb{P}$ is defined by the grammar:

$$p ::= \alpha \mid \delta \mid p + p \mid p \cdot p \mid \mathsf{P}(d)$$

- Choice (non-deterministic): $+$
- Sequential composition: $\cdot$
- Inaction or deadlock: δ
- Processes parameterized by data: $\mathsf{P}(x : D) = p$

Base axiomatics to model sequential behaviours

$$
\begin{array}{lcl}
A1 & x + y & = y + x \\
A2 & (x + y) + z & = x + (y + z) \\
A3 & x + x & = x \\
A4 & (x + y).z & = x.z + y.z \\
A5 & (x.y).z & = x.(y.z) \\
A6 & x + \delta & = x \\
A7 & \delta \cdot x & = \delta
\end{array}
$$

SEQUENTIAL PROCESSES

EXERCISE (AUTONOMOUS WORK)

Describe the following behaviours

- $a.b.\delta.c + a$
- $(a + b).\delta.c$
- $(a + b).e + \delta.c$
- $a + (\delta + a)$
- $a.(b + c).d.(b + c)$

USING THE AXIOMATICS , SHOW THAT:

- $\delta.(a + b) = \delta \cdot a + \delta \cdot b$
- $a + (\delta + a) = a$
- it is true that $a.(b + c) = a.b + a.c$?

Conditionals

We have also processes like:

$$c \rightarrow p \diamond q$$

where

- c is a condition
- p and q are processes

CONDITIONALS

WE HAVE ALSO PROCESSES LIKE:

$$c \to p \diamond q$$

where

- c is a condition
- p and q are processes

AXIOMS:

COND1 $\quad true \to x \diamond y = x$

COND2 $\quad false \to x \diamond y = y$

THEN $\quad c \to x = c \to x \diamond \delta$

mCRL2

Examples

```
act    order, receive, keep, refund, return;

proc   Buy = order.OrderedItem

       OrderedItem = receive.ReceivedItem + refund.Buy;
       ReceivedItem = return.OrderedItem + keep;

init   Buy;
```

Examples

Clock V1

```
act    set, alarm, reset;

proc   P = set.R
       R = reset.P + alarm.R

init   P
```

Examples

Clock V2

```
act    set:N, alarm, reset, tick;

proc   P = (sum n:N . set(n).R(n)) + tick.P
       R(n:N) = reset.P + ((n == 0) -> alarm.R(0) <> tick.R(n-1))

init   P
```

PARALLEL COMPOSITION

$\|$ = **interleaving** + **sincronization**

- **Interaction** is a basic element of systems design
- can be seen as black-boxs configurations
- mCRL2: discipline such **synchronization**

$$p ::= \cdots \mid p \parallel p \mid p \mid p \mid p \between p$$

Parallel Composition

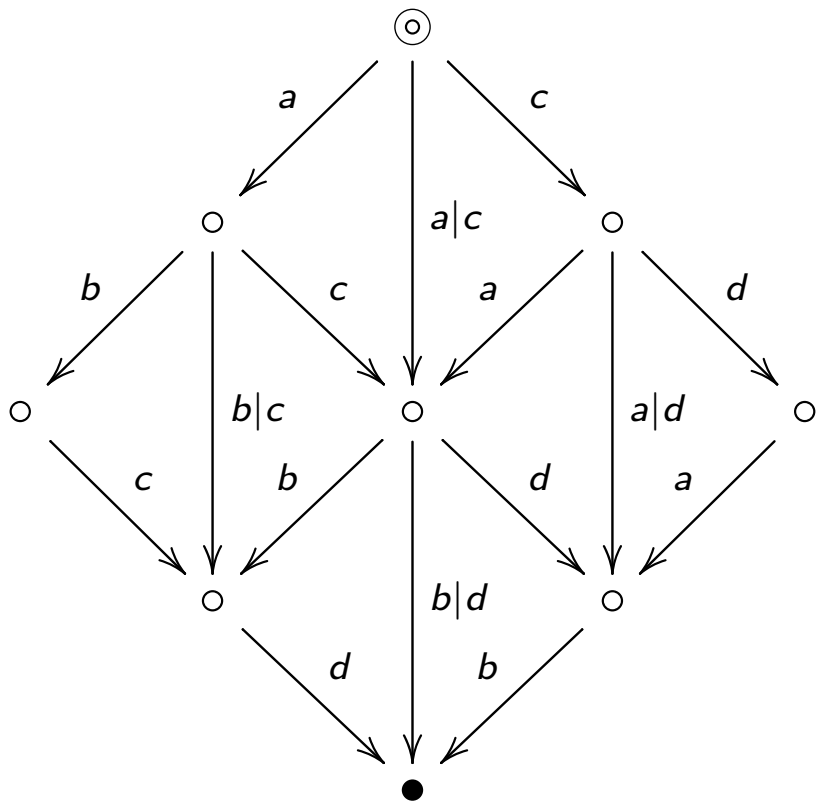
An example

$$a \cdot b \parallel c \cdot d$$

Parallel Composition

An example

$$a \cdot b \parallel c \cdot d$$



Parallel Composition

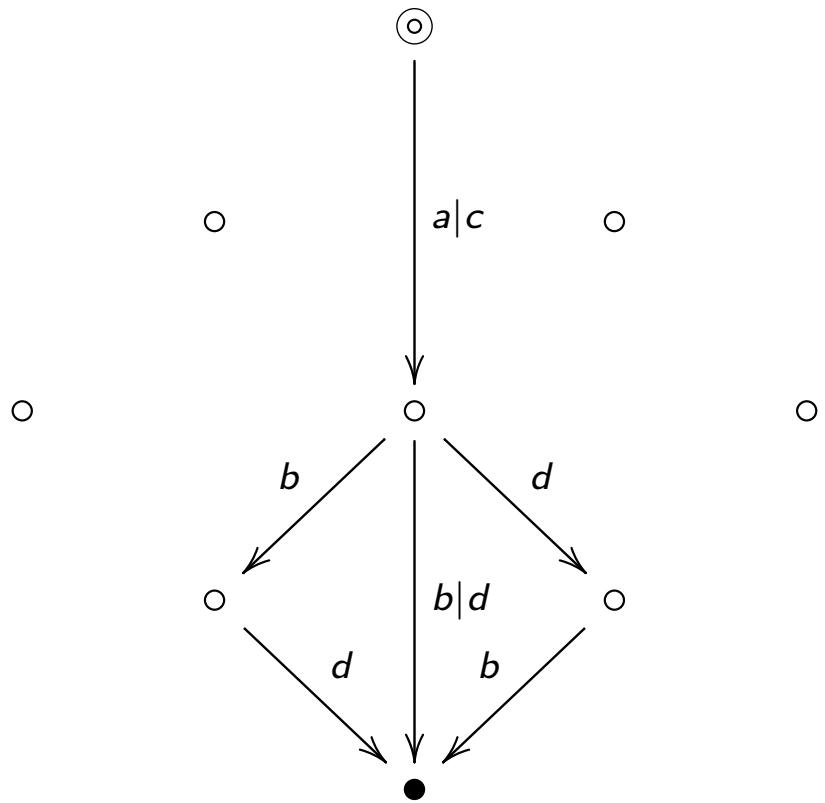
An example

$$a \cdot b \mid c \cdot d$$

Parallel Composition

An example

$$a \cdot b \mid c \cdot d$$



INTERACTION

COMMUNICATION $\Gamma_C(p)$ (COM)

- Applies the **communication function** C, forces synchronization and renames it to a new action:

$$a_1 \mid \cdots \mid a_n \rightarrow c$$

- Enforces communication via data parameters c, e.g.:

$$\Gamma_{\{a|b\rightarrow c\}}(a(8) \mid b(8)) = c(8)$$
$$\Gamma_{\{a|b\rightarrow c\}}(a(12) \mid b(8)) = a(12) \mid b(8)$$
$$\Gamma_{\{a|b\rightarrow c\}}(a(8) \mid a(12) \mid b(8)) = a(12) \mid c(8)$$

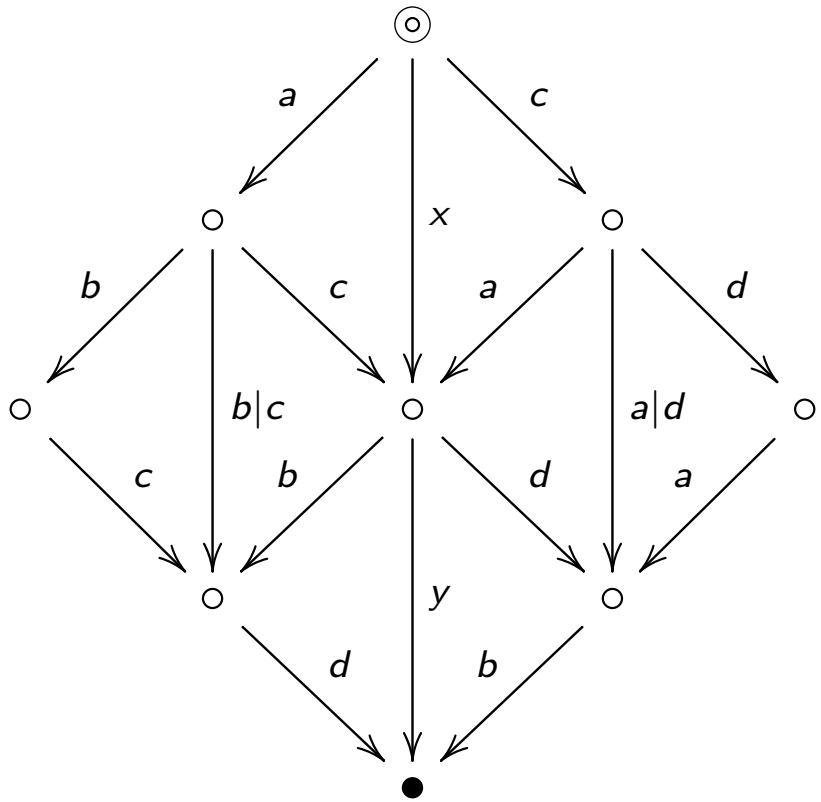
- The left-hand sides of C must be disjoint, e.g., $\{a \mid b \rightarrow c, a \mid d \rightarrow j\}$ is not allowed

Interface control

Exercise

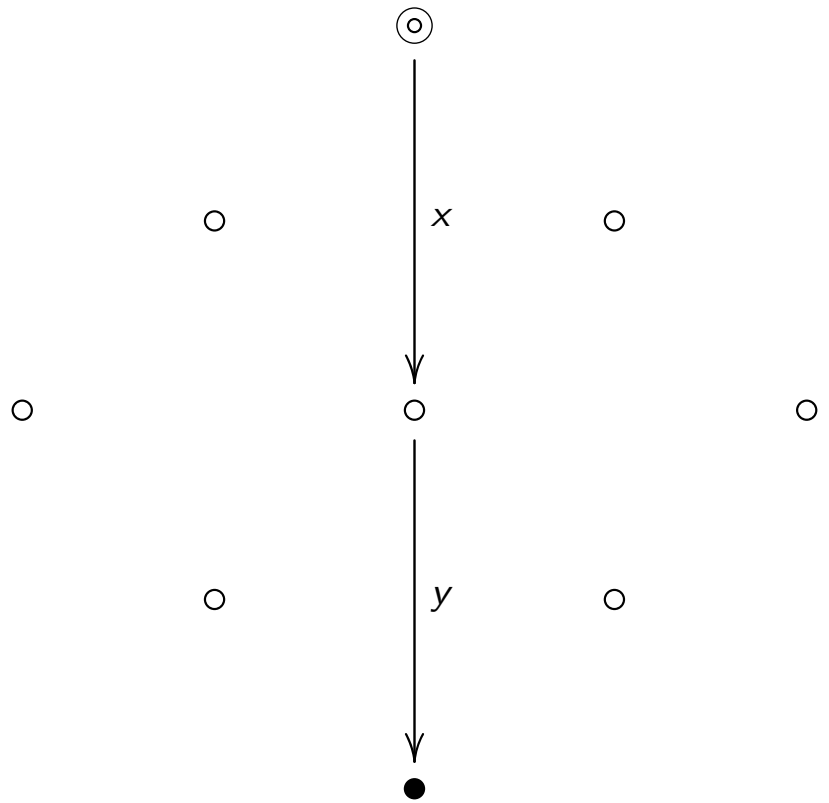
$$\nabla_{\{x,y\}}(\Gamma_{\{a|c->x,b|d->y\}}(a.b \parallel c.d))$$

Interface control



$$\boxed{\Gamma_{\{a|c->x,b|d->y\}}(a.b \parallel c.d)}$$

Interface control



$$\boxed{\nabla_{\{x,y\}}(\Gamma_{\{a|c->x,b|d->y\}}(a.b \parallel c.d))}$$

Interface Control

Block: $\partial_B(p)$ (block)

- Specifies which actions are **not** allowed to occur
- Data parameters do not interfere

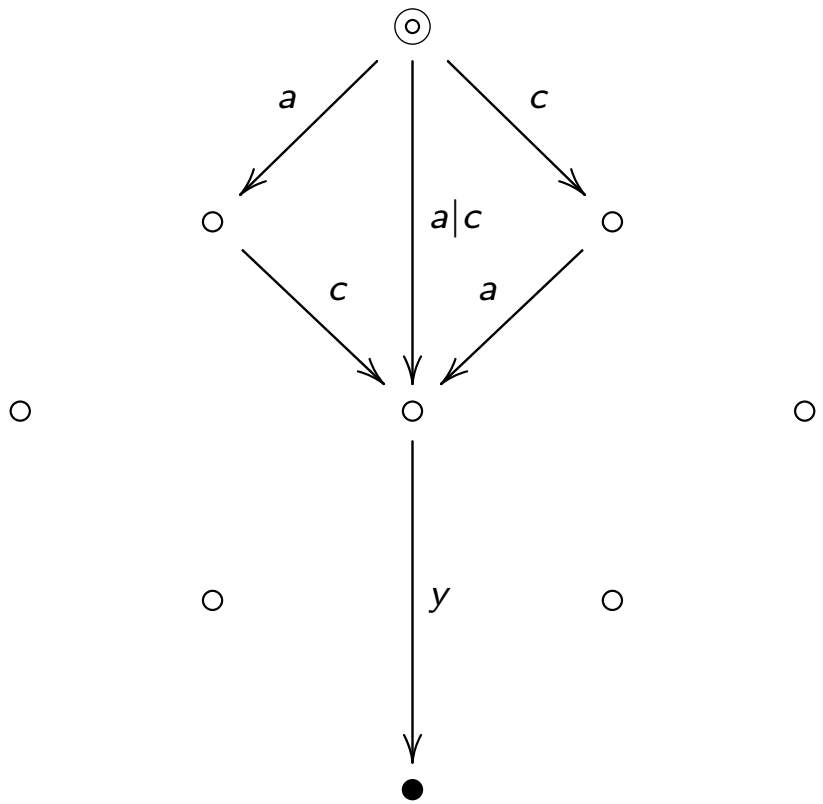
$$\partial_{\{b\}}(d(12) + a(8) + (b(false, 4) \mid c)) \;=\; d(12) + a(8)$$

Interface Control

$$\boxed{\partial_{\{b,d\}}(\Gamma_{\{b|d->y\}}(a.b \parallel c.d))}$$

Interface Control

$$\boxed{\partial_{\{b,d\}}(\Gamma_{\{b|d->y\}}(a.b \parallel c.d))}$$



Interface Control

Enforce communication

- $\nabla_{\{c\}}(\Gamma_{\{a|b\to c\}}(p))$

Interface Control

Enforce communication

- $\nabla_{\{c\}}(\Gamma_{\{a|b\to c\}}(p))$
- $\partial_{\{a,b\}}(\Gamma_{\{a|b\to c\}}(p))$

INTERFACE CONTROL

RENAMING $\rho_M(p)$ (RENAME)

- rename actions of p accordingly with a function M

$$\begin{aligned}\rho_{\{d\to h\}}(d(12)+s(8) \mid d(false)+d.a.d(7)) \\ = \; h(12)+s(8) \mid h(false)+h.a.h(7)\end{aligned}$$

INTERFACE CONTROL

HIDING $\tau_H(p)$ (HIDE)

- hide (i.e. rename to τ) all the actions in H in any multi-actions of p.

$$\tau_{\{d\}}(d(12) + s(8) \mid d(false) + h.a.d(7)) = \tau + s(8) \mid \tau + h.a.\tau = \tau + s(8) + h.a.\tau$$

- τ and δ can not be renamed

Interface Control

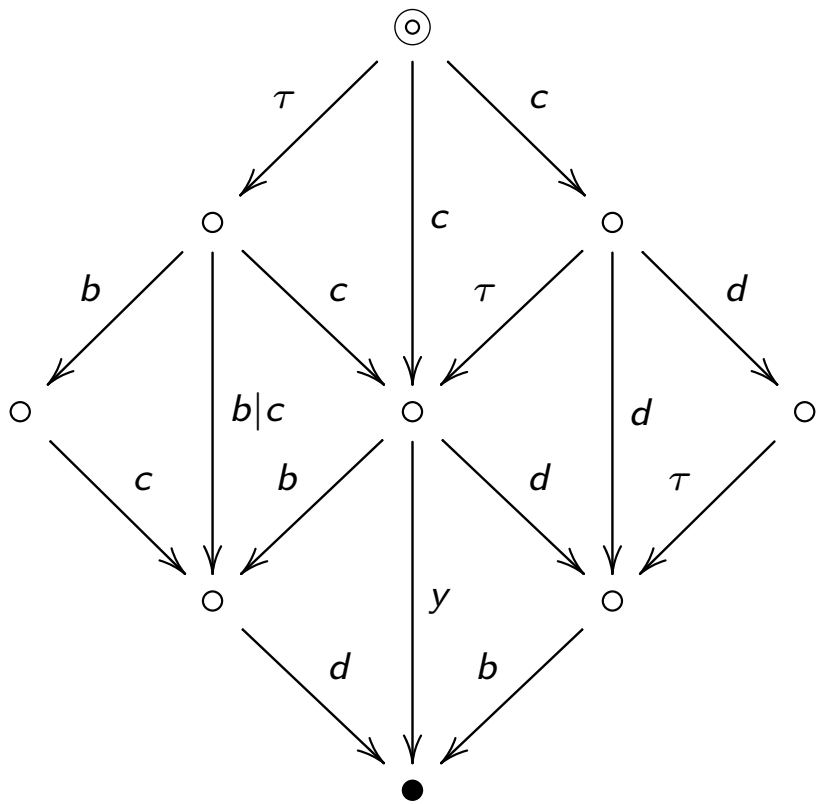
Example

$$\boxed{\tau_{\{a\}}(\Gamma_{\{b|d->y\}}(a.b \parallel c.d))}$$

Interface Control

Example

$$\boxed{\tau_{\{a\}}(\Gamma_{\{b|d->y\}}(a.b \parallel c.d))}$$



Example

Buffers

```
act    inn,outt,ia,ib,oa,ob,c :  Bool;

proc   BufferS = sum n:  Bool.inn(n).outt(n).BufferS;

       BufferA = rename({inn -> ia, outt -> oa}, BufferS);
       BufferB = rename({inn -> ib, outt -> ob}, BufferS);

       S = allow({ia,ob,c}, comm({oa|ib -> c}, BufferA || BufferB));

init   hide({c}, S);
```

Data types

- Equalities: Equations, inequations and conditionals (`if(-,-,-)`)
- Basic types: Booleans, natural, reals, integers, ... with the usual operators
- Sets, multisets, sequences ... with the usual operators
- Definitions of functions, including λ-notation
- Inductive types: such as

```
sort   BTree = struct leaf(Pos) | node(BTree, BTree)
```

Signatures and definitions

Sorts, functions, constants, variables ...

```
sort   S, A;

cons   s,t:S, b:set(A);

map    f:  S x S -> A;
       c:  A;

var    x:S;

eqn    f(x,s) = s;
```

Signatures and definitions

A functional language ...

```
sort   BTree = struct leaf(Pos) | node(BTree, BTree);

map    flatten:  BTree -> List(Pos);

var    n:Pos, t,r:BTree;

eqn    flatten(leaf(n)) = [n];
       flatten(node(t,r)) = flatten(t) ++ flatten(r);
```

Processes with data

Why?

- Data allows to make finite specifications of infinite systems
- data and parametrized processes
- sums with data types: $\sum_{n:N} s(n)$
- conditional processes $b \rightarrow p \diamond q$

Examples

Counter

```
act    up, down;
       setcounter:Pos;

proc   Ctr(x:Pos) = up.Ctr(x+1)
              + (x>0) -> down.Ctr(x-1)
              + sum m:Pos.(setcounter(m).Ctr(m))

init   Ctr(345);
```

Examples

Prime checkers

```
map    primes :  Set(N);
eqn    primes = n : N ∀p,q∈N p, q > 1 ⇒ (p * q) ≠ n;
act    yes, no;
       ask:N;

proc   Checker = sum n:N . ask(n) .  (n in primes -> yes <> no) .  Checker

init   Checker
```

Examples

Dynamic binary trees

```
act    left,right;

map    N:Pos;

eqn    N = 512;

proc   X(n:Pos)=(n<=N)->(left.X(2*n)+right.X(2*n+1))<>delta;

init   X(1);
```

Overview

The verification problem

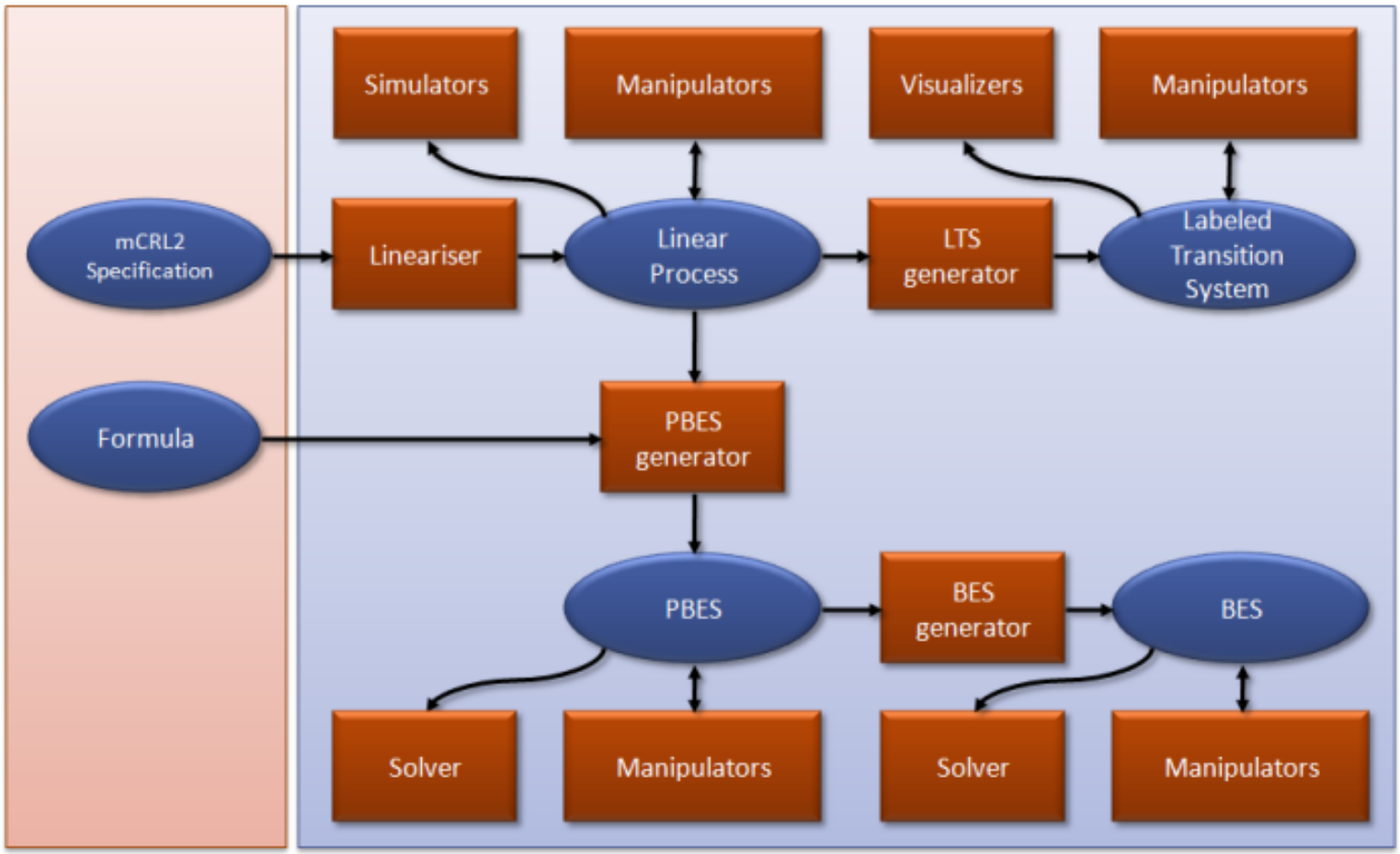
- Given a specification of the system's behaviour is in mCRL2
- and the system's requirements are specified as properties in a temporal logic,
- a model checking algorithm decides whether the property holds for the model: the property can be verified or refuted;
- sometimes, witnesses or counter examples can be provided

Toolset Functionality

Strategies to Handle Infinite Models and Specifications

- The model specification is described in mCRL2 (`x.mcrl2`)
- This specification is linearized into the **Linear Process Specification** format (`x.lps`)
- In this format, the specification can be transformed and simulated
- Specifically, we can generate the associated **Labeled Transition System** (`x.lts`), simulate it, and test properties using the tool's **boolean equation solvers**

Toolset Overview



www.mcrl2.org

Toolset Overview

Tool Tutorial

https://www.mcrl2.org/web/user_manual/tutorial/tutorial.html