ABSTRACT INTERPRETATION: THEORY AND APPLICATIONS

P. Cousot

Patrick.Cousot@ens.fr http://www.di.ens.fr/~cousot

Second International Summer School in Computational Logic, ISCL 2002

25th—30th August 2002, Acquafredda di Maratea (Basilicata, Italy)

LESSON 4

A Potpourri of Applications of Abstract Interpretation

Content of the Potpourri of Applications of Abstract Interpretation

1. ~~Syntax . . . . 47~~
2. ~~Semantics . . . . 51~~
3. Typing 62
4. Model Checking 76
5. ~~Program Transformations . . . . 89~~
6. ~~Static Program Analysis . . . . 93~~

Application to Typing

- P. Cousot, *Types as Abstract Interpretations*, ACM 24th POPL, 1997, pp. 316-331.

Syntax of the Eager Lambda Calculus

$\mathrm{x}, \mathrm{f}, \ldots \in \mathbb{X}$	:	variables
$e \in \mathbb{E}$	:	expressions
$e ::= \mathrm{x}$		variable
	$\mid \boldsymbol{\lambda}\mathrm{x} \cdot e$	abstraction
	$\mid e_1(e_2)$	application
	$\mid \boldsymbol{\mu}\mathrm{f} \cdot \boldsymbol{\lambda}\mathrm{x} \cdot e$	recursion
	$\mid \mathbf{1}$	one
	$\mid e_1 - e_2$	difference
	$\mid (e_1 \,\mathbf{?}\, e_2 \,\mathbf{:}\, e_3)$	conditional

Semantic Domains

Ω	wrong/runtime error value
$\bot$	non-termination
$\mathbb{W} \stackrel{\text{def}}{=} \{\Omega\}$	wrong
$z \in \mathbb{Z}$	integers
$u, f, \varphi \in \mathbb{U} \cong \mathbb{W}_\bot \oplus \mathbb{Z}_\bot \oplus [\mathbb{U} \mapsto \mathbb{U}]$[19]$_\bot$	values
$R \in \mathbb{R} \stackrel{\text{def}}{=} \mathbb{X} \mapsto \mathbb{U}$	environments
$\phi \in \mathbb{S} \stackrel{\text{def}}{=} \mathbb{R} \mapsto \mathbb{U}$	semantic domain

19 $[\mathbb{U} \mapsto \mathbb{U}]$: continuous, $\bot$-strict, Ω-strict functions from values $\mathbb{U}$ to values $\mathbb{U}$.

Denotational Semantics with Run-Time Type Checking

$$\mathsf{S}[\![\mathbf{1}]\!]\mathsf{R} \stackrel{\text{def}}{=} 1$$

$$\begin{aligned}\mathsf{S}[\![e_1 - e_2]\!]\mathsf{R} \stackrel{\text{def}}{=} (\, & \mathsf{S}[\![e_1]\!]\mathsf{R} = \bot \vee \mathsf{S}[\![e_2]\!]\mathsf{R} = \bot \;?\; \bot \\ \mid\, & \mathsf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z}_1 \wedge \mathsf{S}[\![e_2]\!]\mathsf{R} = \mathsf{z}_2 \;?\; \mathsf{z}_1 - \mathsf{z}_2 \\ \mid\, & \Omega \,)\end{aligned}$$

$$\begin{aligned}\mathsf{S}[\![(e_1 \,?\, e_2 : e_3)]\!]\mathsf{R} \stackrel{\text{def}}{=} (\, & \mathsf{S}[\![e_1]\!]\mathsf{R} = \bot \;?\; \bot \\ \mid\, & \mathsf{S}[\![e_1]\!]\mathsf{R} = 0 \;?\; \mathsf{S}[\![e_2]\!]\mathsf{R} \\ \mid\, & \mathsf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z} \neq 0 \;?\; \mathsf{S}[\![e_3]\!]\mathsf{R} \\ \mid\, & \Omega \,)\end{aligned}$$

$$\mathsf{S}[\![\mathrm{x}]\!]\mathsf{R} \stackrel{\text{def}}{=} \mathsf{R}(\mathrm{x})$$

$$\begin{aligned}\mathsf{S}[\![\boldsymbol{\lambda}\mathrm{x}\cdot e]\!]\mathsf{R} \stackrel{\text{def}}{=} \boldsymbol{\lambda}\,\mathsf{u}\bullet(\,&\mathsf{u}=\bot\ ?\ \bot\\ &|\ \mathsf{u}=\Omega\ ?\ \Omega\\ &|\ \mathsf{S}[\![e]\!]\mathsf{R}[\mathrm{x}\leftarrow\mathsf{u}]\ )\end{aligned}$$

$$\begin{aligned}\mathsf{S}[\![e_1(e_2)]\!]\mathsf{R} \stackrel{\text{def}}{=} (\ &\mathsf{S}[\![e_1]\!]\mathsf{R}=\bot \vee \mathsf{S}[\![e_2]\!]\mathsf{R}=\bot\ ?\ \bot\\ &|\ \mathsf{S}[\![e_1]\!]\mathsf{R}=\mathsf{f}\in[\mathbb{U}\mapsto\mathbb{U}]\ ?\ \mathsf{f}(\mathsf{S}[\![e_2]\!]\mathsf{R})\\ &|\ \Omega\ )\end{aligned}$$

$$\mathsf{S}[\![\boldsymbol{\mu}\mathrm{f}\cdot\boldsymbol{\lambda}\mathrm{x}\cdot e]\!]\mathsf{R} \stackrel{\text{def}}{=} \mathrm{lfp}^{\sqsubseteq}\boldsymbol{\lambda}\,\varphi\bullet\mathsf{S}[\![\boldsymbol{\lambda}\mathrm{x}\cdot e]\!]\mathsf{R}[\mathrm{f}\leftarrow\varphi]$$

Standard Denotational & Collecting Semantics

- The denotational semantics is:

$$\mathbf{S}[\![\bullet]\!] \in \mathbb{E} \mapsto \mathbb{S}$$

- A concrete property P of a program is a set of possible program behaviors:

$$P \in \mathbb{P} \stackrel{\text{def}}{=} \wp(\mathbb{S})$$

- The standard collecting semantics is the strongest concrete property:

$$\mathbf{C}[\![\bullet]\!] \in \mathbb{E} \mapsto \mathbb{P} \qquad \mathbf{C}[\![e]\!] \stackrel{\text{def}}{=} \{\mathbf{S}[\![e]\!]\}$$

Church/Curry Monotypes

- Simple types are monomorphic:

$$m \in \mathbb{M}^{c}, \quad m ::= \texttt{int} \mid m_1 \,\texttt{->}\, m_2 \qquad \text{monotype}$$

- A type environment associates a type to free program variables:

$$H \in \mathbb{H}^{c} \stackrel{\text{def}}{=} \mathbb{X} \mapsto \mathbb{M}^{c} \qquad \text{type environment}$$

Church/Curry Monotypes (continued)

- A typing $\langle H, m\rangle$ specifies a possible result type m in a given type environment H assigning types to free variables:

$$\theta \in \mathbb{I}^{\mathsf{c}} \stackrel{\text{def}}{=} \mathbb{H}^{\mathsf{c}} \times \mathbb{M}^{\mathsf{c}} \qquad \text{typing}$$

- An abstract property or program type is a set of typings;

$$T \in \mathbb{T}^{\mathsf{c}} \stackrel{\text{def}}{=} \wp(\mathbb{I}^{\mathsf{c}}) \qquad \text{program type}$$

Concretization Function

The meaning of types is a program property, as defined by the concretization function γ^{c}: [20]

- Monotypes $\gamma_1^{\mathsf{c}} \in \mathbb{M}^{\mathsf{c}} \mapsto \wp(\mathbb{U})$:

$$
\begin{aligned}
\gamma_1^{\mathsf{c}}(\mathtt{int}) &\stackrel{\text{def}}{=} \mathbb{Z} \cup \{\bot\} \\
\gamma_1^{\mathsf{c}}(m_1 \texttt{->} m_2) &\stackrel{\text{def}}{=} \{\varphi \in [\mathbb{U} \mapsto \mathbb{U}] \mid \\
&\qquad\qquad \forall \mathsf{u} \in \gamma_1^{\mathsf{c}}(m_1) : \varphi(\mathsf{u}) \in \gamma_1^{\mathsf{c}}(m_2)\} \\
&\quad \cup \{\bot\}
\end{aligned}
$$

20 For short up/down lifting/injection are omitted.

- type environment $\gamma_2^{\mathsf{c}} \in \mathbb{H}^{\mathsf{c}} \mapsto \wp(\mathbb{R})$:

$$\gamma_2^{\mathsf{c}}(H) \stackrel{\mathsf{def}}{=} \{\mathsf{R} \in \mathbb{R} \mid \forall \mathbf{x} \in \mathbb{X} : \mathsf{R}(\mathbf{x}) \in \gamma_1^{\mathsf{c}}(H(\mathbf{x}))\}$$

- type environment $\gamma_2^{\mathsf{c}} \in \mathbb{H}^{\mathsf{c}} \mapsto \wp(\mathbb{R})$:

$$\gamma_2^{\mathsf{c}}(H) \stackrel{\text{def}}{=} \{\mathsf{R} \in \mathbb{R} \mid \forall \mathbf{x} \in \mathbb{X} : \mathsf{R}(\mathbf{x}) \in \gamma_1^{\mathsf{c}}(H(\mathbf{x}))\}$$

- typing $\gamma_3^{\mathsf{c}} \in \mathbb{I}^{\mathsf{c}} \mapsto \mathbb{P}$:

$$\gamma_3^{\mathsf{c}}(\langle H, m\rangle) \stackrel{\text{def}}{=} \{\phi \in \mathbb{S} \mid \forall \mathsf{R} \in \gamma_2^{\mathsf{c}}(H) : \phi(\mathsf{R}) \in \gamma_1^{\mathsf{c}}(m)\}$$

- type environment $\gamma_2^{\mathsf{c}} \in \mathbb{H}^{\mathsf{c}} \mapsto \wp(\mathbb{R})$:

$$\gamma_2^{\mathsf{c}}(H) \stackrel{\mathsf{def}}{=} \{\mathsf{R} \in \mathbb{R} \mid \forall \mathrm{x} \in \mathbb{X} : \mathsf{R}(\mathrm{x}) \in \gamma_1^{\mathsf{c}}(H(\mathrm{x}))\}$$

- typing $\gamma_3^{\mathsf{c}} \in \mathbb{I}^{\mathsf{c}} \mapsto \mathbb{P}$:

$$\gamma_3^{\mathsf{c}}(\langle H, m\rangle) \stackrel{\mathsf{def}}{=} \{\phi \in \mathbb{S} \mid \forall \mathsf{R} \in \gamma_2^{\mathsf{c}}(H) : \phi(\mathsf{R}) \in \gamma_1^{\mathsf{c}}(m)\}$$

- program type $\gamma^{\mathsf{c}} \in \mathbb{T}^{\mathsf{c}} \mapsto \mathbb{P}$:

$$\gamma^{\mathsf{c}}(T) \stackrel{\mathsf{def}}{=} \bigcap_{\theta \in T} \gamma_3^{\mathsf{c}}(\theta)$$

$$\gamma^{\mathsf{c}}(\emptyset) \stackrel{\mathsf{def}}{=} \mathbb{S}$$

Program Types

- Galois connection:

$$\langle \mathbb{P}, \subseteq, \emptyset, \mathbb{S}, \cup, \cap \rangle \underset{\alpha^{\mathsf{c}}}{\overset{\gamma^{\mathsf{c}}}{\leftrightarrows}} \langle \mathbb{T}^{\mathsf{c}}, \supseteq, \mathbb{I}^{\mathsf{c}}, \emptyset, \cap, \cup \rangle$$

Program Types

- Galois connection:

$$\langle \mathbb{P}, \subseteq, \emptyset, \mathbb{S}, \cup, \cap \rangle \underset{\alpha^{\mathsf{c}}}{\overset{\gamma^{\mathsf{c}}}{\leftrightarrows}} \langle \mathbb{T}^{\mathsf{c}}, \supseteq, \mathbb{I}^{\mathsf{c}}, \emptyset, \cap, \cup \rangle$$

- Types $\mathbf{T}[\![e]\!]$ of an expression e:

$$\mathbf{T}[\![e]\!] \subseteq \alpha^{\mathsf{c}}(\mathbf{C}[\![e]\!]) = \alpha^{\mathsf{c}}(\{\mathbf{S}[\![e]\!]\})$$

Program Types

- Galois connection:

$$\langle \mathbb{P}, \subseteq, \emptyset, \mathbb{S}, \cup, \cap \rangle \underset{\alpha^{\mathsf{c}}}{\overset{\gamma^{\mathsf{c}}}{\leftrightarrows}} \langle \mathbb{T}^{\mathsf{c}}, \supseteq, \mathbb{I}^{\mathsf{c}}, \emptyset, \cap, \cup \rangle$$

- Types $\mathbf{T}[\![e]\!]$ of an expression e:

$$\mathbf{T}[\![e]\!] \subseteq \alpha^{\mathsf{c}}(\mathbf{C}[\![e]\!]) = \alpha^{\mathsf{c}}(\{\mathbf{S}[\![e]\!]\})$$

Typable Programs Cannot Go Wrong

$$\Omega \in \gamma^{\mathsf{c}}(\mathbf{T}[\![e]\!]) \quad \Longleftrightarrow \quad \mathbf{T}[\![e]\!] = \emptyset$$

Church/Curry Monotype Abstract Semantics

$$\mathsf{T}[\![\mathrm{x}]\!] \stackrel{\text{def}}{=} \{\langle H, H(\mathrm{x})\rangle \mid H \in \mathbb{H}^{\mathrm{c}}\} \qquad \text{(VAR)}$$

$$\mathsf{T}[\![\boldsymbol{\lambda}\mathrm{x} \cdot e]\!] \stackrel{\text{def}}{=} \{\langle H, m_1 \to m_2\rangle \mid \langle H[\mathrm{x} \leftarrow m_1], m_2\rangle \in \mathsf{T}[\![e]\!]\} \qquad \text{(ABS)}$$

$$\mathsf{T}[\![e_1(e_2)]\!] \stackrel{\text{def}}{=} \{\langle H, m_2\rangle \mid \langle H, m_1 \to m_2\rangle \in \mathsf{T}[\![e_1]\!] \land \langle H, m_1\rangle \in \mathsf{T}[\![e_2]\!]\} \qquad \text{(APP)}$$

$$\mathsf{T}[\![\mathbf{1}]\!] \stackrel{\text{def}}{=} \{\langle H, \mathtt{int}\rangle \mid H \in \mathbb{H}^c\} \qquad \text{(CST)}$$

$$\mathsf{T}[\![e_1 - e_2]\!] \stackrel{\text{def}}{=} \{\langle H, \mathtt{int}\rangle \mid \langle H, \mathtt{int}\rangle \in \mathsf{T}[\![e_1]\!] \cap \mathsf{T}[\![e_2]\!]\} \qquad \text{(DIF)}$$

$$\mathsf{T}[\![(e_1\, ?\, e_2 : e_3)]\!] \stackrel{\text{def}}{=} \{\langle H, m\rangle \mid \langle H, \mathtt{int}\rangle \in \mathsf{T}[\![e_1]\!] \wedge \langle H, m\rangle \in \mathsf{T}[\![e_2]\!] \cap \mathsf{T}[\![e_3]\!]\} \qquad \text{(CND)}$$

$$\mathsf{T}[\![\boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e]\!] \stackrel{\text{def}}{=} \{\langle H, m\rangle \mid \langle H[\mathtt{f} \leftarrow m], m\rangle \in \mathsf{T}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!]\} \qquad \text{(REC)}$$ [21]

21 The abstract fixpoint has been eliminated thanks to fixpoint induction: $\mathrm{lfp} F \sqsubseteq P \Leftrightarrow \exists I : F(I) \sqsubseteq I \wedge I \sqsubseteq P$.

The Herbrand Abstraction to Get Hindley's Unification-Based Type Inference Algorithm

$$\langle \wp(\mathbf{ground}(T)),\ \subseteq,\ \emptyset,\ \mathbf{ground}(T),\ \cup,\ \cap\rangle \underset{\mathbf{lcg}}{\overset{\mathbf{ground}}{\leftrightarrows}} \langle T^{\emptyset}/_{\equiv},\ \leq,\ \emptyset,\ [\text{'a}]_{\equiv},\ \mathbf{lcg},\ \mathbf{gci}\rangle$$

where:

- T: set of terms with variables 'a, . . . ,
- $\mathbf{lcg}$: least common generalization,
- $\mathbf{ground}$: set of ground instances,
- $\leq$: instance preordering,
- $\mathbf{gci}$: greatest common instance.

Application to Model Checking

- P. Cousot & R. Cousot, *Temporal Abstract Interpretation*, ACM 27th POPL, 2000, pp. 12-25.

Model Checking

1) Built a model M of the computer system;
2) Check (i.e. prove enumeratively) or semi-check (with semi-algorithms) that the model satisfies a specification given (as a set of traces φ) by a (linear) temporal formula: $M \subseteq \varphi$ or $M \cap \varphi \neq \emptyset$.

Objective of Model Checking

1) Built a model M of the computer system;
2) Check (i.e. prove enumeratively) or semi-check (with semi-algorithms) that the model satisfies a specification given (as a set of traces φ) by a (linear) temporal formula: $M \subseteq \varphi$ or $M \cap \varphi \neq \emptyset$.

- The model and specification should be proved to be correct abstractions of the computer system (often taken for granted, could be done by abstract interpretation);

Abstractions in Model Checking

Main abstractions in model checking:

- Implicit abstraction: to informally design the model of reference;
- Polyhedral abstraction (with widening): synchronous, real-time & hybrid system verification;
- Finitary abstraction (without widening): hardware & protocole verification [22];

[22] Abstracting concrete transition systems to abstract transition systems so as to reuse existing model checkers in the abstract.

Model-checking itself is an abstraction

- Universal abstraction:

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \supseteq \rangle \underset{\alpha_M^\forall}{\overset{\gamma_M^\forall}{\longleftarrow\!\!\!\longrightarrow\!\!\!\!\!\rightarrow}} \langle \wp(\Sigma), \supseteq \rangle$$

$$\alpha_M^\forall(\Phi) \stackrel{\text{def}}{=} \{s \mid \{\sigma \in M \mid \sigma_0 = s\} \subseteq \Phi\}$$

- Existential abstraction:

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \subseteq \rangle \underset{\alpha_M^\exists}{\overset{\gamma_M^\exists}{\longleftarrow\!\!\!\longrightarrow\!\!\!\!\!\rightarrow}} \langle \wp(\Sigma), \subseteq \rangle$$

$$\alpha_M^\exists(\Phi) \stackrel{\text{def}}{=} \{s \mid \{\sigma \in M \mid \sigma_0 = s\} \cap \Phi \neq \emptyset\}$$

Model-checking itself is an abstraction

- Universal abstraction:

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \supseteq \rangle \xrightleftharpoons[\alpha_M^\forall]{\gamma_M^\forall} \langle \wp(\Sigma), \supseteq \rangle$$

$$\alpha_M^\forall(\Phi) \stackrel{\text{def}}{=} \{s \mid \{\sigma \in M \mid \sigma_0 = s\} \subseteq \Phi\}$$

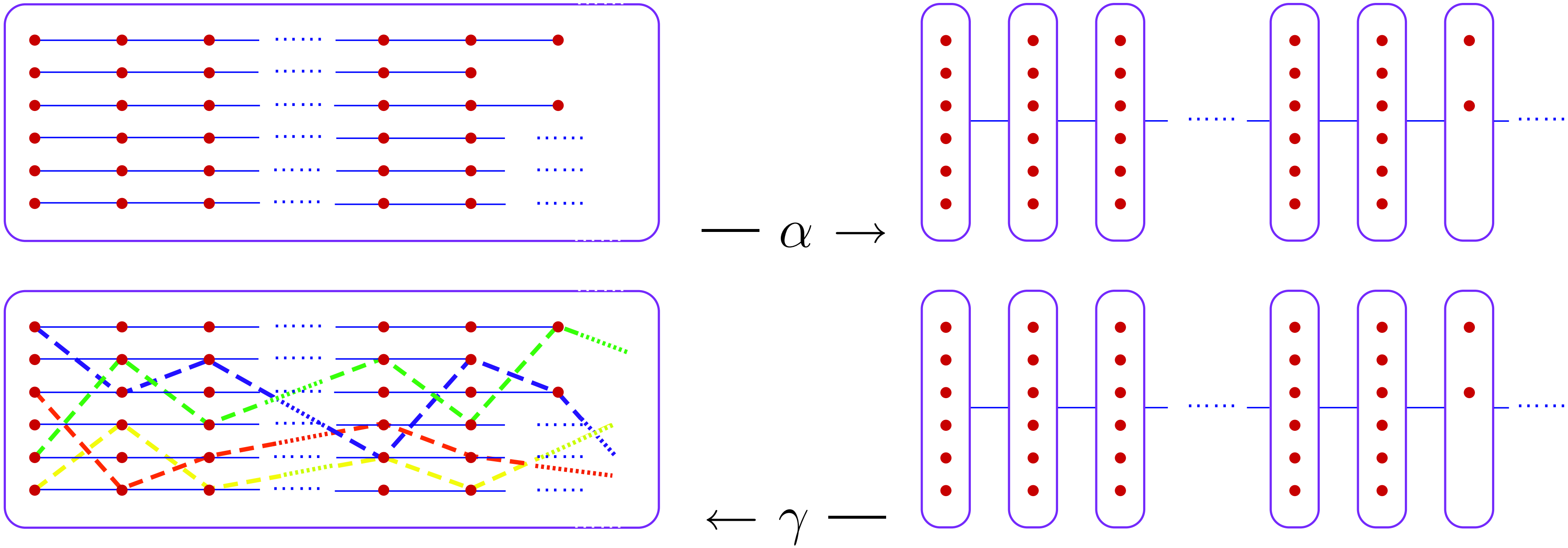
- Existential abstraction:

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \subseteq \rangle \xrightleftharpoons[\alpha_M^\exists]{\gamma_M^\exists} \langle \wp(\Sigma), \subseteq \rangle$$

$$\alpha_M^\exists(\Phi) \stackrel{\text{def}}{=} \{s \mid \{\sigma \in M \mid \sigma_0 = s\} \cap \Phi \neq \emptyset\}$$

These abstractions lead, by fixpoint approximation of the trace semantics, to the classical (finite-state or nonterminating) model-checking algorithms.

Implicit Abstraction in Model Checking



Spurious traces:  ;

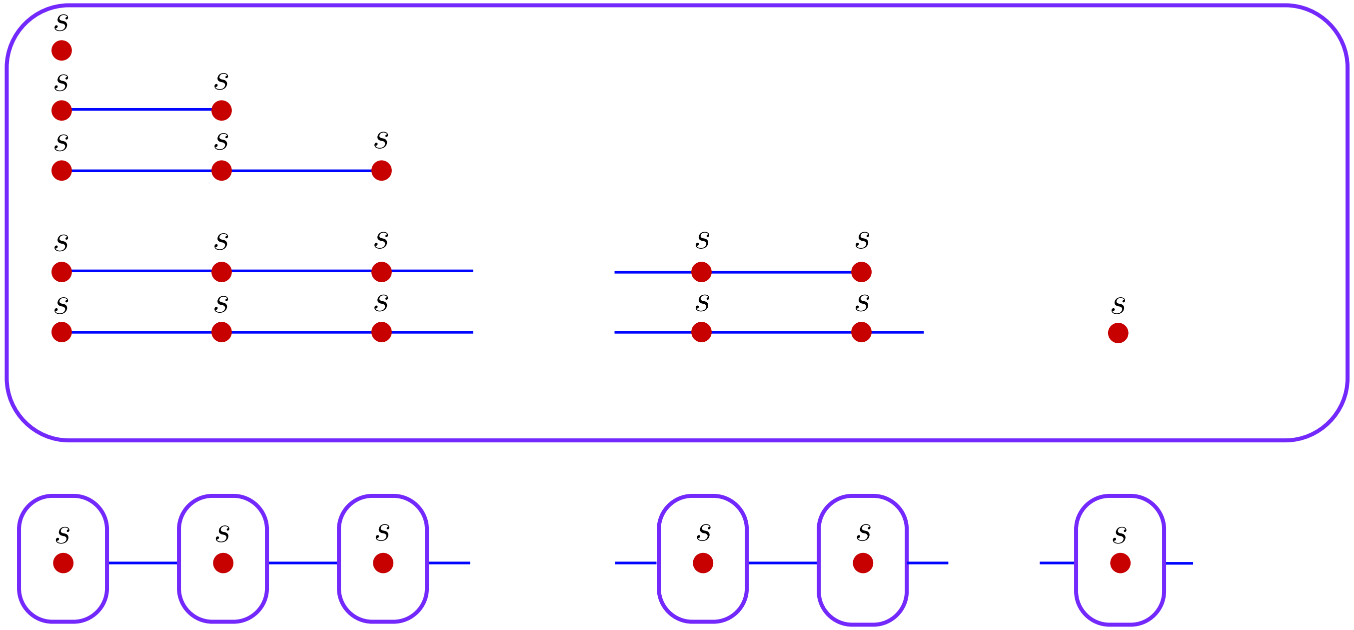
The semantics of the μ-calculus is closed under this abstraction.

Soundness

For a *given class* of properties, soundness means that:

> Any property (in the *given class*) of the abstract world must hold in the concrete world;

Example for <u>Un</u>soundness



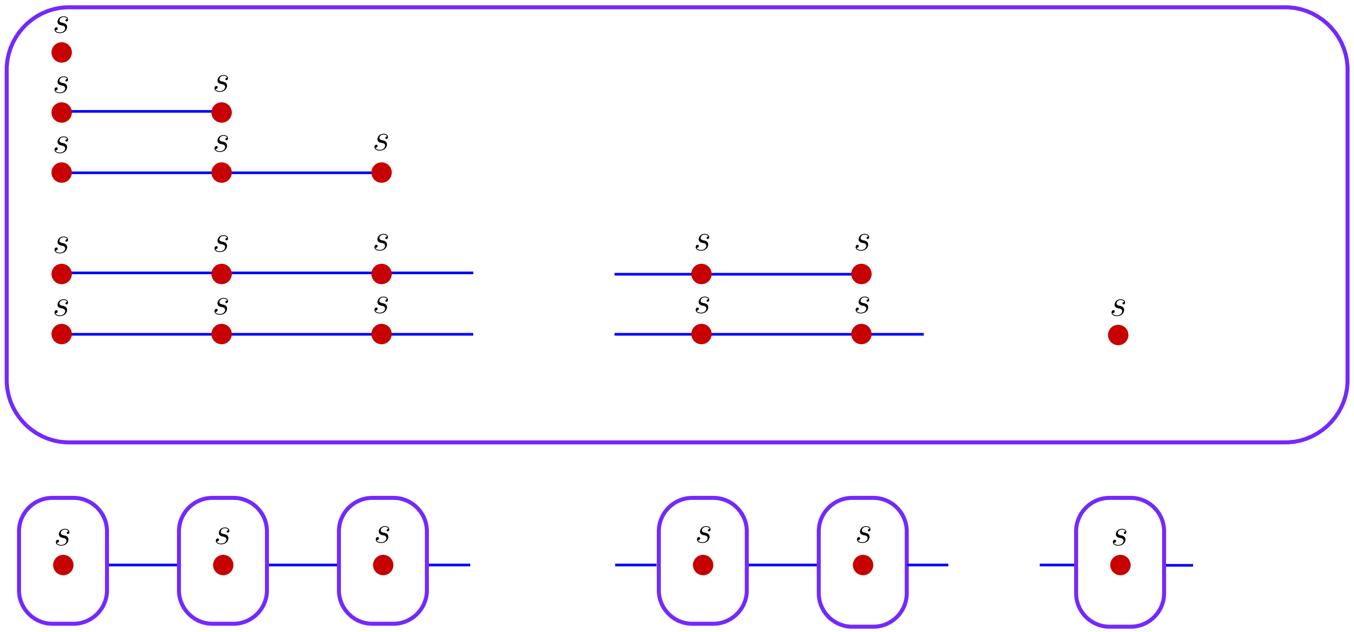
All abstract traces are infinite but not the concrete ones!

Completeness

For a *given class* of properties, completeness means that:

> Any property (in the *given class*) of the concrete world must hold in the abstract world;

Example for Incompleteness



All concrete traces are finite but not the abstract ones!

On the Completeness of Model-Checking

- Contrary to program analysis, model checking is complete;

On the Completeness of Model-Checking

- Contrary to program analysis, model checking is complete;
- Completeness is relative to the model, not the program semantics;

On the Completeness of Model-Checking

- Contrary to program analysis, model checking is complete;
- Completeness is relative to the model, not the program semantics;
- Completeness follows from restrictions on the models and specifications (e.g. closure under the implicit abstraction);

On the Completeness of Model-Checking

- Contrary to program analysis, model checking is complete;
- Completeness is relative to the model, not the program semantics;
- Completeness follows from restrictions on the models and specifications (e.g. closure under the implicit abstraction);
- There are models/specifications (such as the $\stackrel{\curvearrowleft}{\mu}{}^{\star}$-calculus using bidirectional traces) for which:
  - The implicit abstraction is incomplete (POPL'00),
  - Any abstraction is incomplete (Ranzato, ESOP'01).

On the Completeness of Model-Checking

- Contrary to program analysis, model checking is complete;
- Completeness is relative to the model, not the program semantics;
- Completeness follows from restrictions on the models and specifications (e.g. closure under the implicit abstraction);
- There are models/specifications (such as the $\overset{\curvearrowleft}{\mu}^{\star}$-calculus using bidirectional traces) for which:
  - The implicit abstraction is incomplete (POPL'00),
  - Any abstraction is incomplete (Ranzato, ESOP'01).

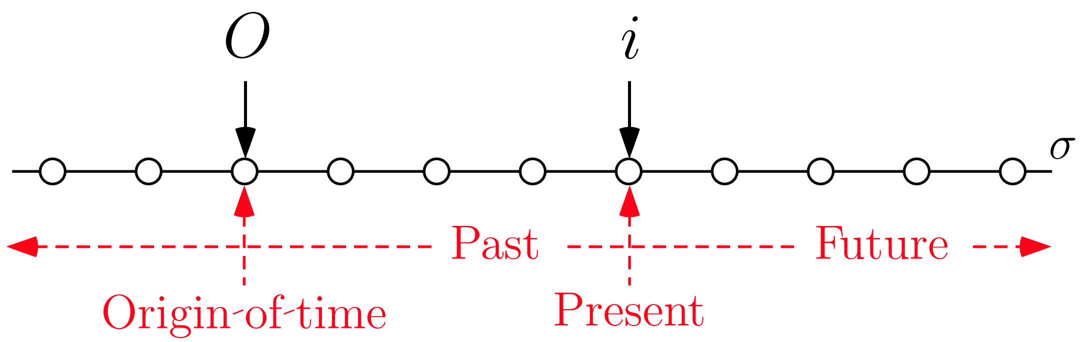
  in both cases, even for *finite* transition systems.

Bidirectional Traces

- $\langle i, \sigma \rangle$ — bidirectional trace

  $\sigma \in \mathbb{Z} \longmapsto \Sigma$ — trace

  $i \in \mathbb{Z}$ — present time



The reversible $\overset{\curvearrowleft\star}{\mu}$-calculus

$$
\begin{array}{lll}
\varphi ::= \boldsymbol{\sigma}_S \text{ [23]} & & [\![\boldsymbol{\sigma}_S]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \sigma_i \in S\} \\
\quad\mid \ \boldsymbol{\pi}_t \text{ [24]} & & [\![\boldsymbol{\pi}_t]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \langle \sigma_i, \sigma_{i+1}\rangle \in t\} \\
\quad\mid \ \oplus\varphi_1 \text{ [25]} & & [\![\oplus\varphi_1]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \langle i+1, \sigma\rangle \in [\![\varphi_1]\!]\rho \\
\quad\mid \ \varphi_1^{\curvearrowleft} & & [\![\varphi_1^{\curvearrowleft}]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \langle -i, \lambda j.\sigma_{-j}\rangle \in [\![\varphi_1]\!]\rho\} \\
\quad\mid \ \varphi_1 \vee \varphi_2 & & [\![\varphi_1 \vee \varphi_2]\!]\rho \stackrel{\text{def}}{=} [\![\varphi_1]\!]\rho \cup [\![\varphi_2]\!]\rho \\
\quad\mid \ \neg\,\varphi_1 & & [\![\neg\,\varphi_1]\!]\rho \stackrel{\text{def}}{=} \neg[\![\varphi_1]\!]\rho
\end{array}
$$

---

23 $S \in \wp(\Sigma)$.

24 $t \in \wp(\Sigma \times \Sigma)$.

25 $\oplus$ is next time.

The reversible $\overset{\curvearrowleft}{\mu}^{\star}$-calculus (cont'd)

$$
\begin{array}{lrcl}
\mid \ \ldots & & & \\
\mid \ X \,^{26} & [\![X]\!]\rho & \stackrel{\text{def}}{=} & \rho(X) \\
\mid \ \boldsymbol{\mu}\, X \cdot \varphi_1 & [\![\boldsymbol{\mu}\, X \cdot \varphi_1]\!]\rho & \stackrel{\text{def}}{=} & \operatorname{lfp}^{\subseteq} \boldsymbol{\lambda}\, x \cdot [\![\varphi_1]\!]\rho X x \\
\mid \ \boldsymbol{\nu}\, X \cdot \varphi_1 & [\![\boldsymbol{\nu}\, X \cdot \varphi_1]\!]\rho & \stackrel{\text{def}}{=} & \operatorname{gfp}^{\subseteq} \boldsymbol{\lambda}\, x \cdot [\![\varphi_1]\!]\rho X x \\
\mid \ \forall\, \varphi_1 : \varphi_2 \,^{27} & [\![\forall\, \varphi_1 : \varphi_2]\!]\rho & \stackrel{\text{def}}{=} & \{\langle i, \sigma\rangle \in [\![\varphi_1]\!]\rho \mid \\
 & & & \{\langle i, \sigma'\rangle \in [\![\varphi_1]\!]\rho \mid \sigma'_i = \sigma_i\} \subseteq [\![\varphi_2]\!]\rho\}
\end{array}
$$

---

26 variable.

27 The traces of φ_1 such that all traces of φ_1 with same present state satisfy φ_2.