« Program Properties: Semantics, Specifications and Logics »

Patrick Cousot
Jerome C. Hunsaker Visiting Professor
Massachusetts Institute of Technology
Department of Aeronautics and Astronautics

cousot@mit.edu
www.mit.edu/~cousot

Course 16.399: “Abstract interpretation”
http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www/

Introduction

In this lecture, our objective is
– to study program properties:
  - that the program does have (semantics)
  - that the program should have (specification)
– to formally specify program properties, essentially via logics

Program Semantics

Christopher Strachey

Dana S. Scott

Reference

[1] Scott, Dana and Strachey, Christopher “Toward a mathematical semantics for computer languages”, in Proc. Symp. on Computers and Automata vol. 21 (1971).

The Variety of Program Semantics

A *program semantics* is a formal description of the possible executions of a program, in interaction with an environment, at some level of abstraction/observation:

- The small-step operational semantics specifies the change of state for any elementary program computation step
- The big-step operational semantics specifies the change of state when executing several computation steps of a program command, ignoring possible nontermination

- The natural semantics specifies the change of initial/final states when completely executing a program command from entry states, ignoring possible nontermination
- The denotational semantics specifies the change of initial/final states when completely executing a program command from entry states, including possible nontermination
- The forward reachability semantics specifies which states can be reached during a program execution starting from given initial states

- The partial/maximal trace operational semantics specify the partial/maximal sequences of states resulting from the successive executions of elementary program computation step[1]
- ...

[1] In the "partial trace operational semantics" observations whence traces are finite sequences. In the "maximal trace operational semantics", traces are maximal whence finite terminating with a blocking states with no possible successors or infinite in case of nontermination.

The Small-Step Operational Semantics

The small-step operational semantics of a program P, as defined in lecture 5, is a transition system:

$$\langle \Sigma[\![P]\!],\ \tau[\![P]\!],\ \mathrm{Entry}[\![P]\!],\ \mathrm{Exit}[\![P]\!]\rangle$$

where:

- $\Sigma[\![P]\!]$ is the set of program states
- $\tau[\![P]\!] \in \wp(\Sigma[\![P]\!] \times \Sigma[\![P]\!])$ is the transition relation between a state and its possible successors
- $\mathrm{Entry}[\![P]\!] \in \wp(\Sigma[\![P]\!])$ is the set of entry/initial states
- $\mathrm{Exit}[\![P]\!] \in \wp(\Sigma[\![P]\!])$ is the set of exit/final states

Trace Semantics

A trace semantics records the sequence of states encountered during a partial or complete execution, maybe together with the action performed to move from one state to another.

```
1:
   X:=?;
2:
   while (X<>0) do
3:
      X:=X-1
4:
   od
5:
```

Example of partial trace (labelled with actions):

$[1{:}\,\Omega] \xrightarrow{\texttt{X:=?}} [2{:}\,5] \xrightarrow{\texttt{X<>0}} [3{:}\,5] \xrightarrow{\texttt{X:=X-1}} [4{:}\,4]$ (each state shows the value of X)

Does not necessarily starts from entry states


```
1:
   X:=?;
2:
   while (X>0) do
3:
      X:=X-1
4:
   od
5:
```

Example of finite maximal trace (labelled with actions):

$[1{:}\,\Omega] \xrightarrow{\texttt{X:=?}} [2{:}\,2] \xrightarrow{\texttt{X<>0}} [3{:}\,2] \xrightarrow{\texttt{X:=X-1}} \ldots$

$[4{:}\,1] \xrightarrow{\texttt{X<>0}} [3{:}\,1] \xrightarrow{\texttt{X:=X-1}} [4{:}\,0] \xrightarrow{\neg\texttt{(X<>0)}} [5{:}\,0]$

"Maximal" since does terminate with a final state (here defined as without any possible successor state $F \stackrel{\text{def}}{=} \{s \in \Sigma \mid \forall s' \in \Sigma : \neg\tau(s, s')\}$)

Does not necessarily starts from entry states


```
1:
   X:=?;
2:
   while (X>0) do
3:
      X:=X-1
4:
   od
5:
```

Example of infinite (maximal)[2] trace (labelled with actions):

$[1{:}\,\Omega] \xrightarrow{\texttt{X:=?}} [2{:}\,\text{-}1] \xrightarrow{\texttt{X<>0}} [3{:}\,\text{-}1] \xrightarrow{\texttt{X:=X-1}} \ldots$

$[4{:}\,\text{-}2] \xrightarrow{\texttt{X<>0}} [3{:}\,\text{-}2] \xrightarrow{\texttt{X:=X-1}} [4{:}\,\text{-}3] \xrightarrow{\texttt{X<>0}} \ldots$

$[3{:}\,\text{-}3] \xrightarrow{\texttt{X:=X-1}} [4{:}\,\text{-}4] \xrightarrow{\texttt{X<>0}} \ldots \;\ldots$

Does not necessarily starts from entry states

[2] Here infinite is maximal although it is mathematically conceivable to have transfinite traces.

The Partial Trace Semantics

Given a transition system $\langle \Sigma,\ \tau \rangle$, the corresponding partial trace semantics is

$$\{\sigma \in \Sigma^{\vec{n}} \mid n > 0 \land \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1})\}$$

Another common case is that of prefix traces starting from given initial states $I \in \wp(\Sigma)$:

$$\{\sigma \in \Sigma^{\vec{n}} \mid n > 0 \land \sigma_0 \in I \land \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1})\}$$

The Maximal Trace Semantics

Given a transition system $\langle \Sigma,\ \tau \rangle$, the corresponding maximal trace semantics is:

– Maximal finite execution traces:

$$\{\sigma \in \Sigma^{\vec{n}} \mid n > 0 \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \wedge \sigma_{n-1} \in F\}$$

where (e.g.) the final states are $F \stackrel{\text{def}}{=} \{s \in \Sigma \mid \forall s' \in \Sigma : \neg t(s, s')\}$

– Maximal infinite execution traces:

$$\{\sigma \in \Sigma^{\vec{\omega}} \mid \forall i \geq 0 : \tau(\sigma_i, \sigma_{i+1})\}$$

– Maximal bifinite execution traces:

$$\{\sigma \in \Sigma^{\vec{n}} \mid n > 0 \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \wedge \sigma_{n-1} \in F\}$$
$$\cup\ \{\sigma \in \Sigma^{\vec{\omega}} \mid \forall i \geq 0 : \tau(\sigma_i, \sigma_{i+1})\}$$

– A special case consists in restricting to initial states $\sigma_0 \in I$ where $I \in \wp(\Sigma)$

The Big-Step Operational Semantics

Given a transition system $\langle \Sigma,\ \tau \rangle$, the corresponding big-step operational semantic is [3]

$$\{\langle \sigma_0,\ \sigma_{n-1} \rangle \mid \exists n > 0 : \sigma \in \Sigma^{\vec{n}} \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1})\}$$
$$= \tau^*$$

– A special case consists in restricting to initial states $\sigma_0 \in I$ that is [4] $I \rceil \tau^*$

[3] A more rigorous but longer notation would be $\{\langle s,\ s' \rangle \mid \exists n > 0 : \exists \sigma \in \Sigma^{\vec{n}} : s = \sigma_0 \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \wedge s' = \sigma_{n-1}\}$

[4] The "big-step operational semantics" is often restricted to an entry/exit relation, which is then nothing but the "natural operational semantics", see page 17

The Natural Denotational Semantics

Given a transition system $\langle \Sigma,\ \tau \rangle$, initial states $I \in \wp(\Sigma)$, the corresponding natural denotational semantic is

$$\{\langle \sigma_0,\ \sigma_{n-1} \rangle \mid \exists n > 0 : \sigma \in \Sigma^{\vec{n}} \wedge \sigma_0 \in I \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \wedge \sigma_{n-1} \in F\}$$
$$\cup\ \{\langle \sigma_0,\ \bot^{5} \rangle \mid \sigma \in \Sigma^{\vec{\omega}} \wedge \forall i \geq 0 : \tau(\sigma_i, \sigma_{i+1})\}$$

where the final states are

$$F \stackrel{\text{def}}{=} \{s \in \Sigma \mid \forall s' \in \Sigma : \neg t(s, s')\}$$

i.e. blocking states

[5] $\bot$ is called "Scott bottom" (from Dana Scott).

The Natural Operational Semantics [6]

Given a transition system $\langle \Sigma,\ \tau\rangle$, $I \in \wp(\Sigma)$, $F \stackrel{\text{def}}{=} \{s \in \Sigma \mid \forall s' \in \Sigma : \neg t(s, s')\}$, the corresponding natural operational semantic is

$$\{\langle\sigma_0,\ \sigma_{n-1}\rangle \mid \exists n > 0 : \sigma \in \Sigma^{\vec{n}} \wedge \sigma_0 \in I \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \wedge \sigma_{n-1} \in F\}$$

[6] Also called the "Angelic Denotational Semantics", where "angelic" refers to the fact that nontermination is completely ignored.

The Demoniac Denotational Semantics [7]

Given a transition system $\langle \Sigma,\ \tau\rangle$, $I \in \wp(\Sigma)$, $F \stackrel{\text{def}}{=} \{s \in \Sigma \mid \forall s' \in \Sigma : \neg t(s, s')\}$, the corresponding demoniac denotational semantics is

$$\{\langle\sigma_0,\ \sigma_{n-1}\rangle \mid \exists n > 0 : \sigma \in \Sigma^{\vec{n}} \wedge \sigma_0 \in I \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \wedge \sigma_{n-1} \in F\}$$
$$\cup\ \{\langle\sigma_0,\ s'\rangle \mid \sigma \in \Sigma^{\vec{\omega}} \wedge \forall i \geq 0 : \tau(\sigma_i, \sigma_{i+1}) \wedge s' \in \Sigma \cup \{\bot\}\}$$

[7] The "demoniac" qualifier refers to the fact that a possibility of nontermination causes an erratic finite behavior (s' can be any state). It follows that conclusions can be drawn unpon final states only in case of definite termination.

Forward Reachability Semantics

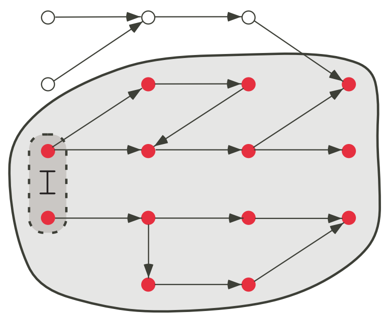
Given a transition system $\langle \Sigma,\ \tau\rangle$ and initial [8] states $I \in \wp(\Sigma)$, the corresponding forward reachability semantics is the set of descendants of the initial states

$$\{\sigma_{n-1} \mid \exists n > 0 : \sigma \in \Sigma^{\vec{n}} \wedge \sigma_0 \in I \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1})\}$$
$$= \text{post}[\tau^*]I$$

where $\text{post}[r]X \triangleq \{y \mid \exists x \in X : r(x, y)\}$.

[8] The "initial" states need not be the entry states but can be any given set of states, excluding maybe the empty set for which the definition is of poor interest!

Example of Forward Reachability Semantics



Backward Reachability Semantics

Given a transition system $\langle \Sigma, \tau \rangle$ and final states [9] $F \in \wp(\Sigma)$ $F \in \wp(\Sigma)$, the corresponding backward reachability semantics is the set of ascendants of the final states

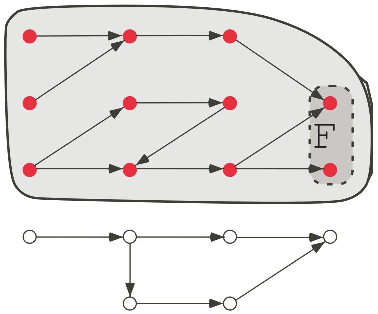
$$\{\sigma_0 \mid \exists n > 0 : \sigma \in \Sigma^{\vec{n}} \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \\ \wedge\, \sigma_{n-1} \in F\}$$

$$= \text{pre}[\tau^\star]F$$

where $\text{pre}[r]X \triangleq \text{post}[r^{-1}]X = \{x \mid \exists y \in X : r(x, y)\}$.

[9] Again, the final states need not be exit states

Example of Backward Reachability Semantics



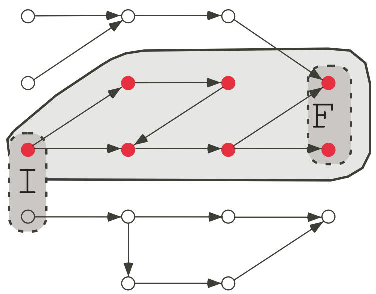
Bidirectional Reachability Semantics

Given a transition system $\langle \Sigma, \tau \rangle$, initial states $I \in \wp(\Sigma)$ and final states $F \in \wp(\Sigma)$, we can also be interested in the reachability semantics that is the set of descendants of the initial states which are ascendants of the final states

$$\{\sigma_j \mid \exists n > 0 : \sigma \in \Sigma^{\vec{n}} \wedge \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \wedge \\ \sigma_0 \in I \wedge 0 \leq j < n \wedge \sigma_{n-1} \in F\}$$

$$= \text{post}[\tau^\star]I \cap \text{pre}[\tau^\star]F$$

Example of Bidirectional Reachability Semantics

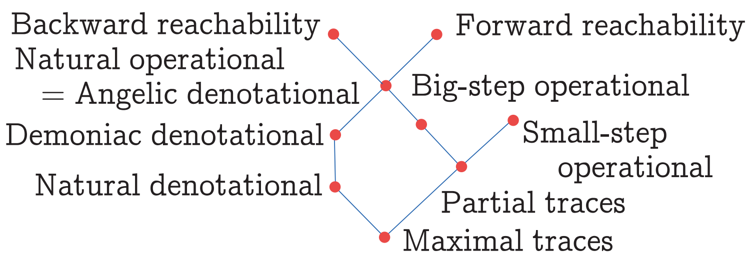


What is the best-fit semantics?

– None
– This depends on which kind of program properties we are interested in!

Hierarchy of Semantics

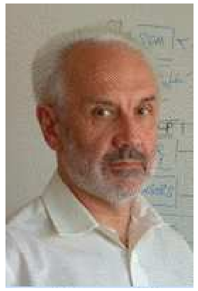
The abstract interpretation point of view [2] is that these semantics are abstractions of each other:



Reference

[2] P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science* 277(1–2):47–103, 2002.

Specification of Program Properties

Jean-Raymond Abrial   Cliff B. Jones

Reference

[3] Jean-Raymond Abrial. "Data Semantics". in J.W. Klimbie and K.L. Koffeman (eds.), IFIP Working Conference *Data Base Management* 1974, pp. 1–60.

[4] Cliff B. Jones. "Program Specifications and Formal Development". International Computing Symposium 1977

Program Specification, Semantics and Correctness

Semantic Domain and Program Semantics

- The semantic domain is $\mathcal{S}$ which elements describes possible program executions
- For example, if executions of a program are described by a set of traces on states Σ then $\mathcal{S} = \wp(\vec{\Sigma}^{\infty})$
- The semantics $\mathsf{Sem}[\![P]\!] \in \mathcal{S}$ of a program P describes effective program executions

Program Properties and Specifications

- A property is represented by the set of elements that have this property (e.g. even $= \{0, 2, 4, \ldots\}$)
- A program property is a set of possible semantics for programs with that property
- The set of program properties is therefore $\wp(\mathcal{S})$
- A program specification Spec is a formal description of desired program property so $\mathsf{Spec} \in \wp(\mathcal{S})$

Program Correctness

- The correctness of a program P with respect to a specification Spec is $\mathsf{Sem}[\![P]\!] \in \mathsf{Spec}$
- The intuition is that the program semantics has the desired property as stated by the specification

Trace properties

Trace properties

- $\mathcal{S} = \wp(\vec{\Sigma}^\infty)$
- $\mathsf{Sem}[\![P]\!] \in \mathcal{S}$ so $\mathsf{Sem}[\![P]\!] \in \wp(\vec{\Sigma}^\infty)$
- $\mathsf{Spec} \in \wp(\mathcal{S})$ so $\mathsf{Spec} \in \wp(\wp(\vec{\Sigma}^\infty))$
- Correctness: $\mathsf{Sem}[\![P]\!] \in \mathsf{Spec}$

In practice, a weaker form of correctness specification (called trace properties):

- $\mathsf{Spec} \in \wp(\vec{\Sigma}^\infty)$
- Universal correctness: $\mathsf{Sem}[\![P]\!] \subseteq \mathsf{Spec}$ [10]

[10] of the more general form $\mathsf{Sem}[\![P]\!] \in \{X \in \wp(\vec{\Sigma}^\infty) \mid \mathsf{Spec} \subseteq X\}$

- Existential correctness: $\mathsf{Sem}[\![P]\!] \cap \mathsf{Spec} \neq \emptyset$ [11]

Not all program trace properties can be expressed in these later form.

[11] of the more general form $\mathsf{Sem}[\![P]\!] \in \{X \in \wp(\vec{\Sigma}^\infty) \mid \mathsf{Spec} \cap X \neq \emptyset\}$

Relational properties

Relational properties

- $\mathcal{S} = \wp(\Sigma \times (\Sigma \cup \{\perp\}))$
- $\mathsf{Sem}[\![P]\!] \in \mathcal{S}$ so $\mathsf{Sem}[\![P]\!] \in \wp(\Sigma \times (\Sigma \cup \{\perp\}))$
- $\mathsf{Spec} \in \wp(\mathcal{S})$ so $\mathsf{Spec} \in \wp(\wp(\Sigma \times (\Sigma \cup \{\perp\})))$
- Correctness: $\mathsf{Sem}[\![P]\!] \in \mathsf{Spec}$

In practice, a weaker form of correctness specification (called relational properties):

- $\mathsf{Spec} \in \wp(\Sigma \times (\Sigma \cup \{\perp\}))$
- Correctness: $\mathsf{Sem}[\![P]\!] \subseteq \mathsf{Spec}$ or $\mathsf{Sem}[\![P]\!] \cap \mathsf{Spec} \neq \emptyset$

Not all program relational properties can be expressed in these later form.

Example of relational property: partial correctness

- Big-step operational semantics/Angelic denotational semantics: $\mathsf{Sem}[\![P]\!] \in \wp(\Sigma \times \Sigma)$
- Specification: $\mathsf{Spec} \in \wp(\Sigma \times \Sigma)$
- Partial correctness: $\mathsf{Sem}[\![P]\!] \subseteq \mathsf{Spec}$

Let any program execution be described by $\langle s,\ s'\rangle \in \mathsf{Sem}[\![P]\!]$. By partial correctness, $\langle s,\ s'\rangle \in \Sigma \times \Sigma$ is constrained to satisfy the specified input-output relation Spec, that is $\langle s,\ s'\rangle \in \mathsf{Spec}$.

Example of relational property: total correctness

- Denotational semantics: $\mathsf{Sem}[\![P]\!] \in \wp(\Sigma \times (\Sigma \cup \{\perp\}))$
- Specification: $\mathsf{Spec} \in \wp(\Sigma \times \Sigma)$
- Total correctness: $\mathsf{Sem}[\![P]\!] \subseteq \mathsf{Spec}$

Let any program execution be described by $\langle s,\ s'\rangle \in \mathsf{Sem}[\![P]\!]$. By total correctness, $\langle s,\ s'\rangle \in \Sigma \times \Sigma$ which excludes $s' = \perp$ that is program nontermination. Morever, an input-output relation must be satisfied as in the partial correctness case.

State properties

State properties

- $\mathcal{S} = \wp(\Sigma)$
- $\mathsf{Sem}[\![P]\!] \in \mathcal{S}$ so $\mathsf{Sem}[\![P]\!] \in \wp(\Sigma)$
- $\mathsf{Spec} \in \wp(\mathcal{S})$ so $\mathsf{Spec} \in \wp(\wp(\Sigma))$
- Correctness: $\mathsf{Sem}[\![P]\!] \in \mathsf{Spec}$

In practice, a weaker form of correctness specification (called state properties):

- $\mathsf{Spec} \in \wp(\Sigma)$
- Correctness: $\mathsf{Sem}[\![P]\!] \subseteq \mathsf{Spec}$ or $\mathsf{Sem}[\![P]\!] \cap \mathsf{Spec} \neq \emptyset$

Not all program state properties can be expressed in these later form.

Example of universal state property: invariance

- Forward reachability semantics:

$$\mathsf{Sem}[\![P]\!] \stackrel{\text{def}}{=} \mathsf{post}[\tau[\![P]\!]^*]I \in \wp(\Sigma)$$

- Specification: $\mathsf{Spec} \in \wp(\Sigma)$
- Invariance: $\mathsf{Sem}[\![P]\!] \subseteq \mathsf{Spec}$

All reachable states during execution must satisfy the specification (this is also called a *safety property* in that all reachable states not in Spec are excluded):

$$\mathsf{post}[\tau[\![P]\!]^*]I \subseteq \mathsf{Spec}$$
$$\Longleftrightarrow\ I \subseteq \widetilde{\mathsf{pre}}[\tau[\![P]\!]^*]\mathsf{Spec} \quad \text{where} \quad \widetilde{\mathsf{pre}}[r]X \triangleq \neg\mathsf{pre}[r](\neg X)$$
$$\Longleftrightarrow\ \forall s, s' \in \Sigma : [s \in I \wedge \tau[\![P]\!]^*(s, s')] \Longrightarrow s' \in \mathsf{Spec}$$

Example of existential state property: runtime error

- Forward reachability semantics:

$$\mathsf{Sem}[\![P]\!] \stackrel{\text{def}}{=} \mathsf{post}[\tau[\![P]\!]^*]I \in \wp(\Sigma)$$

- Specification: $\mathsf{Error} \in \wp(\Sigma)$ (erroneous states)
- Presence of run-time error: $\mathsf{Sem}[\![P]\!] \cap \mathsf{Error} \neq \emptyset$

  There is at least one possible execution of the program which will reach an erroneous state

- Absence of run-time error: $\mathsf{Sem}[\![P]\!] \subseteq \neg\mathsf{Error}$

  No possible possible execution of the program can reach an erroneous state

Program Logics

Formal descriptions of program properties

We have to look for notations that can describe program properties, that is:

- Sets of states ⇒ First-order logic
- Relations on states ⇒ First-order logic
- Traces (sequences of states) ⇒
  - First-order logic
  - Temporal logics
  - Synchronous languages

Set of States Predicate Logic

Formal description of *sets of states* by predicates

In lecture 5, we have defined the mini-language SIL, with:

- Values: $\mathbb{I}_\Omega$ (machine bounded integers and errors)
- Program variables: $\text{Var}[\![P]\!]$
- Environments: $\text{Env}[\![P]\!] \stackrel{\text{def}}{=} \text{Var}[\![P]\!] \mapsto \mathbb{I}_\Omega$
- Program components: $\text{Cmp}[\![P]\!]$
- labels: Lab
- Program labels: $\text{in}_P \in \text{Cmp}[\![P]\!] \mapsto \wp(\text{Lab})$

- States: $\Sigma[\![P]\!] \stackrel{\text{def}}{=} \text{in}_P[\![P]\!] \times \text{Env}[\![P]\!]$

We can describe sets of states by first order predicates, for example

```
1:
  X := 0;
2:
  while (X < 10) do
3:
    X := X + 1
4:
  od
5:
```

$$
\begin{array}{l}
\quad \mathbf{at}[\![1:]\!] \wedge \mathbf{Ierr}[\![\mathtt{X}]\!] \\
\vee\ \mathbf{at}[\![2:]\!] \wedge 0 \leq \mathtt{X} \wedge \mathtt{X} \leq 10 \\
\vee\ \mathbf{at}[\![3:]\!] \wedge 0 \leq \mathtt{X} \wedge \mathtt{X} \leq 9 \\
\vee\ \mathbf{at}[\![4:]\!] \wedge 1 \leq \mathtt{X} \wedge \mathtt{X} \leq 10 \\
\vee\ \mathbf{at}[\![5:]\!] \wedge \mathtt{X} = 10
\end{array}
$$

Extending the syntax of predicates

Formally, the syntax of terms in first order predicates is extended to include

- Program variables: $\mathrm{Var}[\![P]\!]$
- Errors: $\mathbf{Ierr}[\![\mathtt{X}]\!]$, $\mathbf{Aerr}[\![\mathtt{X}]\!]$

The syntax of atomic formulæ is extended with:

- Control atomic formulæ: $\mathbf{at}[\![\ell]\!]$, $\ell \in \mathrm{in}_P$

Extending assignments

Let $\mathfrak{C}$ be a fresh so-called *control variable* such that $\mathfrak{C} \notin \mathrm{Var}[\![P]\!]$.

An assignment ρ maps variables in $\mathrm{Var}[\![P]\!] \cup \{\mathfrak{C}\}$ as follows:

- $\rho(\mathtt{X}) \in \mathbb{I}_\Omega$, $\mathtt{X} \in \mathrm{Var}[\![P]\!]$, memory state
- $\rho(\mathfrak{C}) \in \mathrm{Lab}$, control state

Extending the semantics of predicates

The interpretation of terms defined in lecture 6 is extended with

- $\mathcal{S}^I[\![\mathbf{Ierr}[\![\mathtt{X}]\!]]\!]\rho \stackrel{\mathrm{def}}{=} (\rho(\mathtt{X}) = \Omega_{\mathrm{i}})$ (initialization error)
- $\mathcal{S}^I[\![\mathbf{Aerr}[\![\mathtt{X}]\!]]\!]\rho \stackrel{\mathrm{def}}{=} (\rho(\mathtt{X}) = \Omega_{\mathrm{a}})$ (arithmetic error)
- (and has $\mathcal{S}^I[\![\mathtt{X}]\!]\rho = \rho(\mathtt{X})$, as usual)

The interpretation of atomic formulæ is exetended with

- $\mathcal{S}^I[\![\mathbf{at}[\![\ell]\!]]\!]\rho \stackrel{\mathrm{def}}{=} (\rho(\mathfrak{C}) = \ell)$ (control is at ℓ)

Models of state predicates

Now a first-order formula Φ with a given interpretation I is understood to describe a set of states, as follows:

$$\{\langle \rho(\mathfrak{C}),\ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \rho(\mathtt{X})\rangle \mid \mathcal{S}^I[\![\Phi]\!]\rho = \mathtt{tt}\}$$

$$= \{\langle \rho(\mathfrak{C}),\ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \rho(\mathtt{X})\rangle \mid \rho \Vdash \Phi\}\ [12]$$

12 This is satisfiability $(\rho \Vdash \Phi) \stackrel{\mathrm{def}}{=} (\mathcal{S}^I[\![\Phi]\!]\rho = \mathtt{tt})$.

State Relation Predicate Logic

Formal description of *state relations* by predicates

- We need to be able to make statements about pairs of states
- One convention is to use [13]:
  - Unprimed variables and statements for the first state
  - Primed variables and statements for the second state

[13] Another inverse convention is primed variables for the first state and unprimed one for the second. Another convention is that of a preprime $'\mathtt{X}$ for the first state and a postprime $\mathtt{X}'$ for the second. One can also use indexes like $\mathtt{X}_0$ and $\mathtt{X}_1$, $\underline{\mathtt{X}}$, $\overline{\mathtt{X}}$, etc.

Extending the syntax of predicates

Formally, the syntax of terms in first order predicates is extended to include

- (Primed) program variables: $\mathrm{Var}[\![P]\!]$, $\{\mathtt{X}' \mid \mathtt{X} \in \mathrm{Var}[\![P]\!]\}$
- Mathemathical variables: $x \in \mathcal{V}$ [14]
- Errors: $\mathbf{Ierr}[\![\mathtt{X}]\!]$, $\mathbf{Aerr}[\![\mathtt{X}]\!]$

while atomic formulae also include

- Control atomic formulæ: $\mathbf{at}[\![\ell]\!]$, $\mathbf{at}'[\![\ell]\!]$, $\ell \in \mathrm{in}_P$

[14] different from the (primed) program variables in that $\mathcal{V} \cap (\mathrm{Var}[\![P]\!] \cup \{\mathtt{X}' \mid \mathtt{X} \in \mathrm{Var}[\![P]\!]\}) = \emptyset$

Extending assignments

To define the interpretation of formulæ, let $\mathfrak{C}$, $\mathfrak{C}'$ be a fresh so-called *control variables* [15].

An assignment ρ maps variables in $\mathrm{Var}[\![P]\!] \cup \{\mathtt{X}' \mid \mathtt{X} \in \mathrm{Var}[\![P]\!]\} \cup \{\mathfrak{C}, \mathfrak{C}'\} \cup \mathcal{V}$ as follows:

- $\rho(x) \in \mathbb{Z}$, $x \in \mathcal{V}$
- $\rho(\mathtt{X}), \rho(\mathtt{X}') \in \mathbb{I}_{\Omega}$, $\mathtt{X} \in \mathrm{Var}[\![P]\!]$, memory states
- $\rho(\mathfrak{C}), \rho(\mathfrak{C}') \in \mathrm{Lab}$, control states

[15] different from the (primed) program and mathematical variables in that $\{\mathfrak{C}, \mathfrak{C}'\} \cap (\mathrm{Var}[\![P]\!] \cup \{\mathtt{X}' \mid \mathtt{X} \in \mathrm{Var}[\![P]\!]\} \cup \mathcal{V}) = \emptyset$

Extending the semantics of predicates

The interpretation of terms defined in lecture 6 is extended with

- $\mathcal{S}^I[\![\mathbf{Ierr}[\![\mathtt{X}]\!]]\!]\rho \stackrel{\text{def}}{=} (\rho(\mathtt{X}) = \Omega_{\mathtt{i}})$ [16] (initialization error)
- $\mathcal{S}^I[\![\mathbf{Aerr}[\![\mathtt{X}]\!]]\!]\rho \stackrel{\text{def}}{=} (\rho(\mathtt{X}) = \Omega_{\mathtt{a}})$ [16] (arithmetic error)
- (and has $\mathcal{S}^I[\![\mathtt{X}]\!]\rho = \rho(\mathtt{X})$ [16], as usual)

while for atomic formulae, we have

- $\mathcal{S}^I[\![\mathbf{at}[\![\ell]\!]]\!]\rho \stackrel{\text{def}}{=} (\rho(\mathfrak{C}) = \ell)$ (control is first at ℓ)
- $\mathcal{S}^I[\![\mathbf{at}'[\![\ell]\!]]\!]\rho \stackrel{\text{def}}{=} (\rho(\mathfrak{C}') = \ell)$ (control is then at ℓ)

[16] $\mathtt{X} \in (\mathrm{Var}[\![P]\!] \cup \{\mathtt{X}' \mid \mathtt{X} \in \mathrm{Var}[\![P]\!]\})$

Models of state relation predicates

Now a first-order formula Φ with a given interpretation I is understood to describe a state relation (set of pairs of states), as follows:

$$\{\langle\langle\rho(\mathfrak{C}'),\ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \rho(\mathtt{X}')\rangle,\ \langle\rho(\mathfrak{C}),\ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \rho(\mathtt{X})\rangle\rangle \mid \mathcal{S}^I[\![\Phi]\!]\rho = \mathtt{tt}\}$$

$$= \{\langle\langle\rho(\mathfrak{C}'),\ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \rho(\mathtt{X}')\rangle,\ \langle\rho(\mathfrak{C}),\ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \rho(\mathtt{X})\rangle\rangle \mid \rho \Vdash \Phi\}$$ [17]

[17] Again, this is satisfiability $(\rho \Vdash \Phi) \stackrel{\text{def}}{=} (\mathcal{S}^I[\![\Phi]\!]\rho = \mathtt{tt})$.

Example of state relation described by a predicate

The classical program invariants:

```
1: { X = x0 & Y = y0 & x0 >= y0 }
   Z := X;
2:
   while (Z <> Y) do
3:    { X = x0 >= Z > Y = y0 }
      Z := Z - 1
4:
   od
5:
```

can be specified by the predicate:

$$(\mathbf{at}'[\![1:]\!] \wedge \mathtt{X}' = x_0 \wedge \mathtt{Y}' = y_0 \wedge x_0 \geq y_0 \wedge \mathbf{at}[\![3:]\!]) \Longrightarrow (\mathtt{X} = x_0 \geq \mathtt{Z} > \mathtt{Y} = y_0)$$

Trace Predicate Logic

Formal description of *traces* by predicates

- We use a discrete model for time (i.e. $\mathbb{N}$ instead of $\mathbb{R}_+$)
- All traces are infinite [18]
- We need to be able to make statements about states at a given time
  - We use $\mathtt{X}[t]$ to denote the value of the program variable $\mathtt{X}$ at time $t \in \mathbb{N}$
  - We use $\mathbf{at}[\![\ell]\!][t]$ to specify where the control stands at time $t \in \mathbb{N}$

18 Finite ones can be encoded using an undefined value $\bot$: $abc..xyz$ becomes $abc..xyz\bot\bot\bot\ldots\bot\bot\bot\ldots$

Extending the syntax of predicates

Formally, the syntax of first order predicates is extended as follows

- Mathematical variables: $x \in \mathcal{V}$
- Program variables: $\mathtt{X} \in \mathsf{Var}[\![P]\!]$ [19]
- Terms $t \in \mathcal{T}$:

$$
\begin{array}{rcl}
t & ::= & c \\
 & | & x \\
 & | & f(t_1, \ldots, t_n) \\
 & | & \mathtt{X}[t]
\end{array}
$$

19 Assuming $\mathcal{V} \cap \mathsf{Var}[\![P]\!] = \emptyset$

- Atomic formulæ $A \in \mathcal{A}$:

$$
\begin{array}{rcl}
A & ::= & r(t_1, \ldots, t_n) \\
 & | & \mathbf{at}[\![\ell]\!][t] \\
 & | & \mathbf{Aerr}[\![\mathtt{X}]\!][t] \\
 & | & \mathbf{Ierr}[\![\mathtt{X}]\!][t]
\end{array}
$$

- Formulæ $\Phi \in \mathcal{L}$:

$$
\begin{array}{rcll}
\Phi & ::= & A & A \in \mathcal{A} \\
 & | & \forall x : \Phi & x \in \mathcal{V} \\
 & | & \Phi_1 \vee \Phi_2 & \\
 & | & \neg\Phi &
\end{array}
$$

Note that quantification is over mathematical variables only

Extending assignments

To define the interpretation of formulæ, let $\mathfrak{C}$ be a fresh so-called *control variable* [20].

An assignment ρ maps variables in $\mathsf{Var}[\![P]\!] \cup \{\mathfrak{C}\} \cup \mathcal{V}$ as follows:

- $\rho(x) \in \mathcal{D}_I$, $x \in \mathcal{V}$
- $\rho(\mathtt{X}) \in \mathbb{N} \mapsto \mathbb{I}_\Omega$, $\mathtt{X} \in \mathsf{Var}[\![P]\!]$, timed memory states
- $\rho(\mathfrak{C}) \in \mathbb{N} \mapsto \mathsf{Lab}$, timed control states

20 different from the program and mathematical variables in that $\mathfrak{C} \notin (\mathsf{Var}[\![P]\!] \cup \mathcal{V})$

Extending the semantics of predicates

The interpretation of terms defined in lecture 6 is extended with

- $\mathcal{S}^I[\![\mathtt{X}[t]]\!]\rho \stackrel{\text{def}}{=} \rho(\mathtt{X})(\mathcal{S}^I[\![t]\!]\rho)$ [21]

while for atomic formulae, we have

- $\mathcal{S}^I[\![\mathbf{at}[\![\ell]\!][t]]\!]\rho \stackrel{\text{def}}{=} (\rho(\mathfrak{C})(\mathcal{S}^I[\![t]\!]\rho) = \ell)$
- $\mathcal{S}^I[\![\mathbf{Ierr}[\![\mathtt{X}]\!][t]]\!]\rho \stackrel{\text{def}}{=} (\rho(\mathtt{X})(\mathcal{S}^I[\![t]\!]\rho) = \Omega_{\mathrm{i}})$ [21] (initialization error)

[21] $\mathtt{X} \in (\mathrm{Var}[\![P]\!] \cup \{\mathtt{X}' \mid \mathtt{X} \in \mathrm{Var}[\![P]\!]\})$

- $\mathcal{S}^I[\![\mathbf{Aerr}[\![\mathtt{X}]\!][t]]\!]\rho \stackrel{\text{def}}{=} (\rho(\mathtt{X})(\mathcal{S}^I[\![t]\!]\rho) = \Omega_{\mathrm{a}})$ [21] (arithmetic error)
- (and has $\mathcal{S}^I[\![\mathtt{X}]\!]\rho = \rho(\mathtt{X})$ [21], as usual)

[21] $\mathtt{X} \in (\mathrm{Var}[\![P]\!] \cup \{\mathtt{X}' \mid \mathtt{X} \in \mathrm{Var}[\![P]\!]\})$

Models of trace predicates

Now a first-order formula Φ with a given interpretation I is understood to describe traces, as follows:

$$\{\lambda i \in \mathbb{N} \cdot \langle \rho(\mathfrak{C})(i),\ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \rho(\mathtt{X})(i)\rangle \mid \mathcal{S}^I[\![\Phi]\!]\rho = \mathtt{tt}\}$$

$$= \{\lambda i \in \mathbb{N} \cdot \langle \rho(\mathfrak{C})(i),\ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \rho(\mathtt{X})(i)\rangle \mid \rho \Vdash \Phi\}$$ [22]

[22] Again, this is satisfiability $(\rho \Vdash \Phi) \stackrel{\text{def}}{=} (\mathcal{S}^I[\![\Phi]\!]\rho = \mathtt{tt})$.

Example of trace description by a predicate

The decrementation of X over time in:

```
1:
   Z := ?;
2:
   while (Z > 0) do
3:
      Z := Z - 1
4:
   od
5:
```

can be specified by the first-order trace predicate:

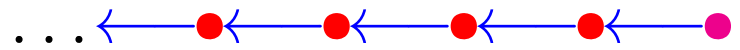
$$\forall i, j : ((i \leq j) \land \neg\mathbf{at}[\![1:]\!][i]) \Longrightarrow (\mathtt{Z}[i] \geq \mathtt{Z}[j])$$

Future, Past and Bidirectional Traces

– We have considered future traces

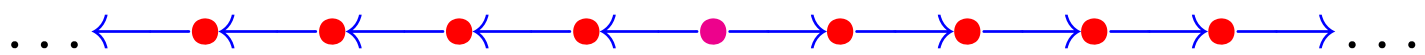
to specify what can happen from now on

– Past traces



are useful to describe the present state as a function of the past

– Bidirectional traces [5]

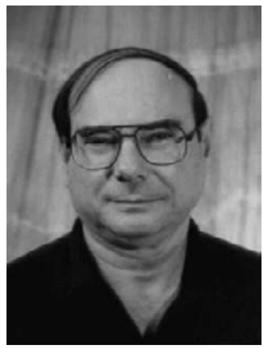


are useful to describe the future as a function of the past

Reference

[5] P. Cousot and R. Cousot. "Temporal abstract interpretation". In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Mass., January 2000. ACM Press, New York, NY.

Linear Time Temporal Logic



Amir Pnueli

Reference

[6] Amir Pnueli: "The Temporal Logic of Programs", In Proc. 18th Symp. Foundations of Computer Science, pages 46–57, 1977.

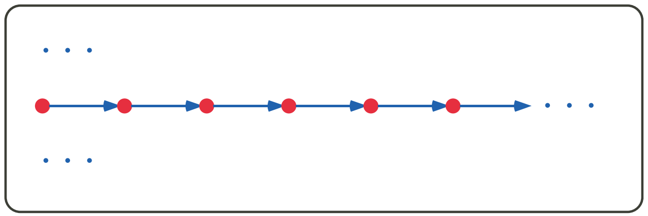
[7] Zohar Manna and Amir Pnueli: "The Temporal Logic of Reactive and Concurrent Systems: Specification". Springer-Verlag, 1992

Temporal logics

– Traces predicates are flexible but too general to be handled easily by computer-aided formal methods
– Other forms of logics, inspired by modal logic, have been designed to specify execution traces

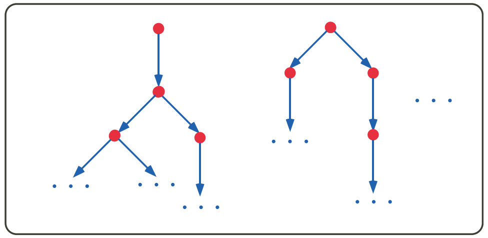
Linear-time temporal Logic [6]

– The set of execution traces is defined by describing traces one at a time



Branching-time temporal Logic

– The set of traces is defined by describing the nondeterministic interleaving of executions (like in Emerson's CTL* [8])

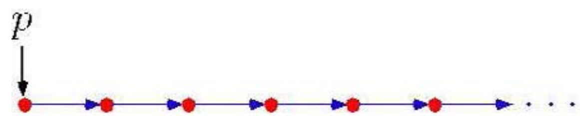


Reference

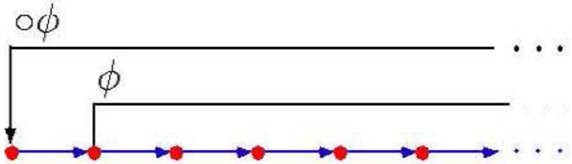
[8] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time. POPL 1983: Pages 127-140

Linear Temporal Operators

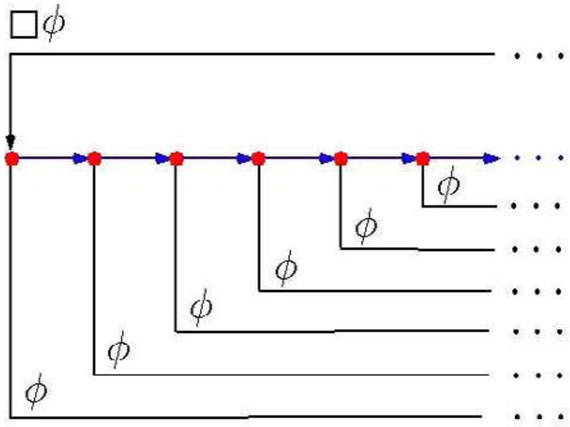
– An atomic predicate p means that the current state in the trace satisfies p



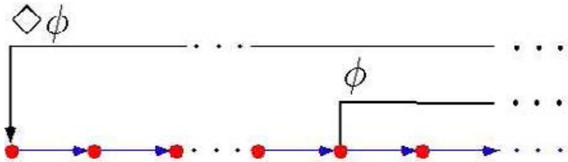
– $\circ\,\Phi$ means that Φ holds at next time in the trace



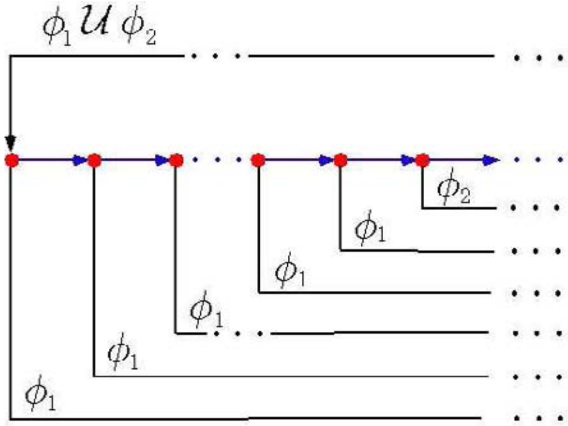
– $\Box\,\Phi$ means that from now on, the trace satisfies Φ



– $\Diamond\,\Phi$ means that some time in the future, the trace satisfies Φ



– $\Phi_1\,\mathcal{U}\,\Phi_2$ means that Φ_1 always holds until the trace satisfies Φ_2



Linear Temporal Logic Syntax

$v, u \in \mathcal{V}$	Variables
p	State formula (first order predicate)
$\Phi ::=$	LTL formula
$\quad p$	state formula
$\quad \mid \neg \Phi$	negation
$\quad \mid \Phi_1 \vee \Phi_2$	disjunction
$\quad \mid \exists u : \Phi$	existential quantification
$\quad \mid \circ \Phi$	next
$\quad \mid \Phi_1 \,\mathcal{U}\, \Phi_2$	until

Trace Suffix

Given an infinite trace $\sigma \in \Sigma^{\vec{\omega}}$, and $k \in \mathbb{N}$, we define the *suffix* $\sigma \nearrow k$ *of* σ *at* k as the infinite trace starting at k

$$\sigma \nearrow k \stackrel{\text{def}}{=} \sigma_k \sigma_{k+1} \sigma_{k+2} \ldots$$

In particular $\sigma \nearrow 0 = \sigma$

Linear Temporal Logic Semantics

Let I be an interpretation of the first-order logic (where $\Sigma \stackrel{\text{def}}{=} \mathcal{V} \mapsto D_I$), the semantics $\mathcal{S}^I[\![\Phi]\!]$ of a LTL formula Φ is

$$\begin{aligned}
\mathcal{S}^I[\![p]\!] &\stackrel{\text{def}}{=} \{\sigma \in \Sigma^{\vec{\omega}} \mid \mathcal{S}^I[\![p]\!]\sigma_0 = \mathtt{tt}\} \\
\mathcal{S}^I[\![\neg\Phi]\!] &\stackrel{\text{def}}{=} \Sigma^{\vec{\omega}} \setminus \mathcal{S}^I[\![\Phi]\!] \\
\mathcal{S}^I[\![\Phi_1 \vee \Phi_2]\!] &\stackrel{\text{def}}{=} \mathcal{S}^I[\![\Phi_1]\!] \cup \mathcal{S}^I[\![\Phi_2]\!] \\
\mathcal{S}^I[\![\exists u : \Phi]\!] &\stackrel{\text{def}}{=} \bigcup_{d \in D_I} \{\sigma[u := d] \mid \sigma \in \mathcal{S}^I[\![\Phi]\!]\}
\end{aligned}$$

where $\sigma[u := d] \stackrel{\text{def}}{=} \lambda i \in \mathbb{N} \cdot \sigma_i[u := d]$

$$\begin{aligned}
\mathcal{S}^I[\![\circ\,\Phi]\!] &\stackrel{\text{def}}{=} \{\sigma \in \Sigma^{\vec{\omega}} \mid \sigma \nearrow 1 \in \mathcal{S}^I[\![\Phi]\!]\} \\
\mathcal{S}^I[\![\Phi_1 \,\mathcal{U}\, \Phi_2]\!] &\stackrel{\text{def}}{=} \{\sigma \in \Sigma^{\vec{\omega}} \mid \exists k \in \mathbb{N} : \forall i \in [0, k-1] : \\
&\qquad \sigma \nearrow i \in \mathcal{S}^I[\![\Phi_1]\!] \wedge \sigma \nearrow k \in \mathcal{S}^I[\![\Phi_2]\!]\}
\end{aligned}$$

LTL Auxiliary Operators

- $\Diamond \Phi \stackrel{\text{def}}{=} \mathtt{tt}\,\mathcal{U}\,\Phi$ eventually/sometime
- $\Box \Phi \stackrel{\text{def}}{=} \neg(\Diamond \neg \Phi)$ always/henceforth
- $\Phi_1\,\mathcal{W}\,\Phi_2 \stackrel{\text{def}}{=} \Phi_1\,\mathcal{U}\,\Phi_2 \vee \Box \Phi_1$ waiting for/unless

Example of trace description by a LTL formula

The decrementation of X over time in:

```
1:
   Z := ?;
2:
   while (Z > 0) do
3:
      Z := Z - 1
4:
   od
5:
```

can be specified by the LTL formula:

$$\Box(\forall u : (\neg \mathbf{at}[\![1:]\!] \wedge \mathrm{Z} = u) \Longrightarrow (\Box(\mathrm{Z} \leq u)))$$

The semantics of the LTL formula

$$-\ \mathcal{S}^I[\![\mathrm{Z} = u]\!]$$
$$= \{\sigma \in \Sigma^{\vec{\omega}} \mid \mathcal{S}^I[\![\mathrm{Z} = u]\!]\sigma_0\}$$
$$= \{\sigma \mid \sigma_0(\mathrm{Z}) \leq \sigma_0(u)\}$$

$$-\ \mathcal{S}^I[\![\Box(\mathrm{Z} = u)]\!]$$
$$= \{\sigma \in \Sigma^{\vec{\omega}} \mid \forall k \in \mathbb{N} : \sigma \nearrow k \in \mathcal{S}^I[\![\mathrm{Z} = u]\!]\}$$
$$= \{\sigma \mid \forall k \in \mathbb{N} : \sigma_k(z) \leq \sigma_k(u)\}$$

$$-\ \mathcal{S}^I[\![\neg \mathbf{at}[\![1:]\!] \wedge \mathrm{Z} = u]\!]$$
$$= \{\sigma \in \Sigma^{\vec{\omega}} \mid \mathcal{S}^I[\![\neg \mathbf{at}[\![1:]\!] \wedge \mathrm{Z} = u]\!]\sigma_0\}$$
$$= \{\sigma \mid (\sigma_0(\mathfrak{C}) \neq 1: \wedge\, \sigma_0(\mathrm{Z}) = \sigma_0(u))\}$$

$$-\ \mathcal{S}^I[\![(\neg \mathbf{at}[\![1:]\!] \wedge \mathrm{Z} = u) \Longrightarrow (\Box(\mathrm{Z} \leq u))]\!]$$
$$= \{\sigma \in \Sigma^{\vec{\omega}} \mid \mathcal{S}^I[\![(\neg \mathbf{at}[\![1:]\!] \wedge \mathrm{Z} = u) \Longrightarrow (\Box(\mathrm{Z} \leq u))]\!]\sigma_0\}$$

$$= \{\sigma \mid (\sigma_0(\mathfrak{C}) \neq 1: \wedge\, \sigma_0(\mathrm{Z}) = \sigma_0(u)) \Longrightarrow (\forall k \in \mathbb{N} : \sigma_k(\mathrm{Z}) \leq \sigma_k(u))\}$$

$$-\ \mathcal{S}^I[\![\forall u : (\neg \mathbf{at}[\![1:]\!] \wedge \mathrm{Z} = u) \Longrightarrow (\Box(\mathrm{Z} \leq u))]\!])$$
$$= \bigcap_{d \in \mathbb{I}_\Omega} \{\sigma[u := d] \mid \sigma \in \mathcal{S}^I[\![(\neg \mathbf{at}[\![1:]\!] \wedge \mathrm{Z} = u) \Longrightarrow (\Box(\mathrm{Z} \leq u))]\!]\}$$
$$= \bigcap_{d \in \mathbb{I}_\Omega} \{\sigma[u := d] \mid (\sigma_0(\mathfrak{C}) \neq 1: \wedge\, \sigma_0(\mathrm{Z}) = \sigma_0(u)) \Longrightarrow (\forall k \in \mathbb{N} : \sigma_k(\mathrm{Z}) \leq \sigma_k(u))\}$$
$$= \bigcap_{d \in \mathbb{I}_\Omega} \{\sigma \mid (\sigma_0(\mathfrak{C}) \neq 1: \wedge\, \sigma_0(\mathrm{Z}) = \sigma_0(d)) \Longrightarrow (\forall k \in \mathbb{N} : \sigma_k(\mathrm{Z}) \leq \sigma_k(d))\}$$
$$= \{\sigma \mid \forall d \in \mathbb{I}_\Omega : (\sigma_0(\mathfrak{C}) \neq 1: \wedge\, \sigma_0(\mathrm{Z}) = d) \Longrightarrow (\forall k \in \mathbb{N} : \sigma_k(\mathrm{Z}) \leq d)\}$$
$$= \{\sigma \mid [\sigma_0(\mathfrak{C}) \neq 1:] \Longrightarrow [\forall d \in \mathbb{I}_\Omega : (\sigma_0(\mathrm{Z}) = d) \Longrightarrow (\forall k \in \mathbb{N} : \sigma_k(\mathrm{Z}) \leq d)]\}$$
$$= \{\sigma \mid [\sigma_0(\mathfrak{C}) \neq 1:] \Longrightarrow [\forall k \in \mathbb{N} : \sigma_k(\mathrm{Z}) \leq \sigma_0(\mathrm{Z})]\}$$

Intuitively, for all execution traces that do not start at 1:, the later values of Z are less than or equal to the current value of Z.

Expressing Simple Properties with LTL Formulæ

- $\Phi_1 \Longrightarrow \Diamond \Phi_2$
  if Φ_1 holds now then Φ_2 eventually holds later
- $\Box(\Phi_1 \Longrightarrow \circ \Phi_2)$
  whenever Φ_1 holds, Φ_2 holds in next state
- $\Box(\Phi_1 \Longrightarrow \Diamond \Phi_2)$
  once Φ_1 holds, Φ_2 eventually holds
- $\Box(\Phi_1 \Longrightarrow \Box \Phi_2)$
  once Φ_1 holds, Φ_2 always holds

- $\Box \Diamond \Phi$
  Φ holds infinitely often
- $\Diamond \Box \Phi$
  eventually Φ holds permanently
- $(\neg \Phi_1) \mathcal{W} \Phi_2$
  the first time Φ_1 holds, Φ_2 must hold now or previously
- $\Box \exists u : ((x = u) \wedge \circ (x = u + 1))$
  x increases by 1 from any state to the next

Temporal tautologies

- $\Box \Phi = \neg(\Diamond \neg \Phi)$
- $\Box \Phi = \Phi \mathcal{W} \mathrm{ff}$
- $\Phi_1 \mathcal{U} \Phi_2 = (\Phi_1 \mathcal{W} \Phi_2) \wedge \Diamond \Phi_2$
- $\Box \Phi = \Phi \wedge \circ (\Box \Phi)$
- $\Diamond \Phi = \Phi \vee \circ (\Diamond \Phi)$
- $\Phi_1 \mathcal{U} \Phi_2 = \Phi_2 \vee (\Phi_1 \wedge \circ (\Phi_1 \mathcal{U} \Phi_2))$
- $\Phi_1 \mathcal{W} \Phi_2 = \Phi_2 \vee (\Phi_1 \wedge \circ (\Phi_1 \mathcal{W} \Phi_2))$
- $\Phi = \mathrm{ff} \mathcal{U} \Phi$
- $\Box \Phi \Longrightarrow \Phi$
- $\Phi \Longrightarrow \Diamond \Phi$

- $\Phi_1 \mathcal{U} \Phi_2 \Longrightarrow (\Phi_1 \vee \Phi_2)$
- $\Phi_1 \mathcal{W} \Phi_2 \Longrightarrow (\Phi_1 \vee \Phi_2)$
- $\Phi_1 \mathcal{U} \Phi_2 \Longrightarrow \Phi_1 \mathcal{W} \Phi_2$
- $\Phi_2 \Longrightarrow \Phi_1 \mathcal{U} \Phi_2$
- $\Phi_2 \Longrightarrow \Phi_1 \mathcal{W} \Phi_2$
- $\neg(\Phi_1 \mathcal{U} \Phi_2) \iff (\neg \Phi_2) \mathcal{W} (\neg \Phi_1 \wedge \neg \Phi_2)$
- $\neg(\Phi_1 \mathcal{W} \Phi_2) \iff (\neg \Phi_2) \mathcal{U} (\neg \Phi_1 \wedge \neg \Phi_2)$
- $\neg(\circ \Phi) \iff \circ (\neg \Phi)$
- $\neg(\Box \Phi) \iff \Diamond (\neg \Phi)$
- $\neg(\Diamond \Phi) \iff \Box(\neg \Phi)$
- $\Box \Box \Phi \iff \Box \Phi$

- $\Diamond \Diamond \Phi \iff \Diamond \Phi$
- $\Phi_1 \mathcal{U} (\Phi_1 \mathcal{U} \Phi_2) \iff \Phi_1 \mathcal{U} \Phi_2$
- $\Phi_1 \mathcal{W} (\Phi_1 \mathcal{W} \Phi_2) \iff \Phi_1 \mathcal{W} \Phi_2$
- $(\Phi_1 \mathcal{U} \Phi_2) \mathcal{U} \Phi_2 \iff \Phi_1 \mathcal{U} \Phi_2$
- $(\Phi_1 \mathcal{W} \Phi_2) \mathcal{W} \Phi_2 \iff \Phi_1 \mathcal{W} \Phi_2$
- $\Diamond \Box \Diamond \Phi \iff \Box \Diamond \Phi$
- $\Box \Diamond \Box \Phi \iff \Diamond \Box \Phi$
- $\Phi_1 \mathcal{W} (\Phi_1 \mathcal{U} \Phi_2) \iff \Phi_1 \mathcal{W} \Phi_2$
- $(\Phi_1 \mathcal{U} \Phi_2) \mathcal{W} \Phi_2 \iff \Phi_1 \mathcal{U} \Phi_2$
- $\Phi_1 \mathcal{U} (\Phi_1 \mathcal{W} \Phi_2) \iff \Phi_1 \mathcal{W} \Phi_2$
- $(\Phi_1 \mathcal{W} \Phi_2) \mathcal{U} \Phi_2 \iff \Phi_1 \mathcal{U} \Phi_2$

- $\bigcirc (\Phi_1 \vee \Phi_2) \iff (\bigcirc \Phi_1) \vee (\bigcirc \Phi_2)$
- $\bigcirc (\Phi_1 \mathcal{W} \Phi_2) \iff (\bigcirc \Phi_1) \mathcal{W} (\bigcirc \Phi_2)$
- $\Diamond (\Phi_1 \vee \Phi_2) \iff (\Diamond \Phi_1) \vee (\Diamond \Phi_2)$
- $\Box (\Phi_1 \wedge \Phi_2) \iff (\Box \Phi_1) \wedge (\Box \Phi_2)$
- $\Phi_1 \mathcal{U} (\Phi_2 \vee \Phi_3) \iff (\Phi_1 \mathcal{U} \Phi_2) \vee (\Phi_1 \mathcal{U} \Phi_3)$
- $(\Phi_1 \wedge \Phi_2) \mathcal{U} \Phi_3 \iff (\Phi_1 \mathcal{U} \Phi_3) \wedge (\Phi_2 \mathcal{U} \Phi_3)$
- $\Phi_1 \mathcal{W} (\Phi_2 \vee \Phi_3) \iff (\Phi_1 \mathcal{W} \Phi_2) \vee (\Phi_1 \mathcal{W} \Phi_3)$
- $(\Phi_1 \wedge \Phi_2) \mathcal{W} \Phi_3 \iff (\Phi_1 \mathcal{W} \Phi_3) \wedge (\Phi_2 \mathcal{W} \Phi_3)$
- $\Box (\forall u : \Phi) \iff \forall u : \Box \Phi$
- $\Diamond (\exists u : \Phi) \iff \exists u : \Diamond \Phi$

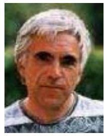
- $\Phi_1 \mathcal{U} (\exists u : \Phi_2) \iff \exists u : (\Phi_1 \mathcal{U} \Phi_2) \qquad u \notin \mathrm{FV}[\![\Phi_1]\!]$ [23]
- $(\forall u : \Phi_1) \mathcal{U} \Phi_2 \iff \forall u : \Phi_1 \mathcal{U} \Phi_2 \qquad u \notin \mathrm{FV}[\![\Phi_2]\!]$
- $\Phi_1 \mathcal{W} (\exists u : \Phi_2) \iff \exists u : (\Phi_1 \mathcal{W} \Phi_2) \qquad u \notin \mathrm{FV}[\![\Phi_1]\!]$
- $(\forall u : \Phi_1) \mathcal{W} \Phi_2 \iff \forall u : \Phi_1 \mathcal{W} \Phi_2 \qquad u \notin \mathrm{FV}[\![\Phi_2]\!]$

In general $\Diamond (\forall u : \Phi) \not\Longleftarrow\!\Longrightarrow \forall u : (\Diamond \Phi)$, a counter example is:

- $\sigma = \{\mathtt{X} \to 0, \mathtt{U} \to 0\}, \{\mathtt{X} \to 0, \mathtt{U} \to 1\}, \ldots, \{\mathtt{X} \to 0, \mathtt{U} \to n\}, \ldots$
- $\sigma \Vdash \forall u > 0 : (\Diamond (\mathtt{X} \neq \mathtt{U} \wedge \mathtt{U} = u))$
- $\sigma \nVdash \Diamond (\forall u > 0 : (\mathtt{X} \neq \mathtt{U} \wedge \mathtt{U} = u))$

[23] Recall that $\mathrm{FV}[\![\Phi]\!]$ is the set of free variables of Φ

Synchronous Languages



Gérard Berry Paul Caspi Nicolas Halbwachs

References

[9] Gérard Berry, Laurent Cosserat. "The ESTEREL Synchronous Programming Language and its Mathematical Semantics". Seminar on Concurrency, LNCS 187, Springer, 1984, pp. 389–448.

[10] J.L. Bergerand, P. Caspi, D. Pilaud, N. Halbwachs, E. Pilaud. "Outline of a Real Time Data Flow Language". IEEE Real-Time Systems Symposium, San Diego, 1985, pp. 33–42.

[11] N. Halbwachs. "Synchronous programming of reactive systems". Kluwer Academic Pub., 1993.

Stream/synchronous languages

– Stream/synchronous languages like Lucid [12] or Scade [24]/ Lustre [11], etc. can be used to specify sets of finite/ infinite traces (streams)

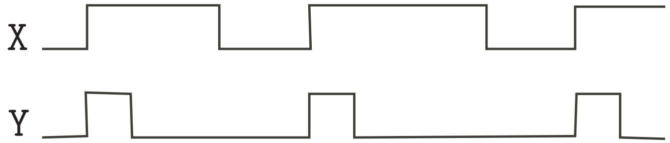
Reference

[12] William W. Wadge and Edward A. Ashcroft. "LUCID, the dataflow programming language". A.P.I.C. Studies In Data Processing; Vol. 22, 312 p., Academic Press Professional, Inc. San Diego, CA, USA, 1985,

[24] Commercialized by Esterel Technologies

Example

The time diagram



can be specified in LUSTRE [11] as

```
Y = X and not pre(X)
```

that is

$$\begin{cases} Y(0) \text{ is undefined} \\ Y(n+1) = X(n+1) \land \neg X(n) \quad n \geq 0 \end{cases}$$

or better

```
Y = false → X and not pre(X)
```

that is

$$\begin{cases} Y(0) = \mathrm{ff} \\ Y(n+1) = X(n+1) \land \neg X(n) \quad n \geq 0 \end{cases}$$

Syntax of a (subset [25]) of LUSTRE

$\mathtt{X}$	variables
$P ::= DP \mid D$	program
$D ::= X = E$	equational declaration
$E ::=$	expression
$\quad f\backslash n(E_1, \ldots, E_n)$	
$\quad \mid \ \mathtt{pre}(E)$	
$\quad \mid \ E_1 \rightarrow E_2$	
$\quad \mid \ \mathtt{X}$	

[25] The most important notions left out in the subset are that of *module* and of *clock*. Here all sequences are bases on the same clock (while in general there is a basic clock and all sequences are defined at given periods of the basic clock and constant in between).

title

The semantics of a program P is the set ρ of infinite execution traces coinductively defined by the equation [26]

$$\rho = \mathcal{S}^I[\![P]\!](\rho)$$

where

$$\mathcal{S}^I[\![\mathtt{X}_1 = E_1 \ldots \mathtt{X}_n = E_n]\!]\rho(\mathtt{X}_i) \stackrel{\text{def}}{=} \mathcal{S}^I[\![E_i]\!]\rho$$

(The value of variable $\mathtt{X}_i$ is given by expression E_i in equation $\mathtt{X}_i = E_i$) .../...

[26] There is a mathematical difficulty here that was we elucidate when studying fixpoint definitions. Here we choose the $\subseteq$-geratest fixpoint.

Semantics of a subset of LUSTRE

- let $\text{Var}[\![P]\!]$ be the set of variables in program P
- let I be an interpretation and D_I be the set of program variable values (including the booleans $\mathbb{B}$, ...)
- The values of the program variables are traces (or streams) in $\mathbb{N} \mapsto D_I$
- The semantics of a program maps variables to their value:

$$\mathcal{S}^I[\![P]\!] \in \wp\left(\prod_{\mathtt{X} \in \text{Var}[\![P]\!]} \mathbb{N} \mapsto D_I\right)$$

$$\begin{aligned}
\mathcal{S}^I[\![f\backslash n(E_1, \ldots, E_n)]\!]\rho &\stackrel{\text{def}}{=} \{\mathcal{I}^I[\![f\backslash n]\!](\sigma_1, \ldots, \sigma_n) \mid \\
&\qquad \forall i \in [1, n] : \sigma_i \in \mathcal{S}^I[\![E_i]\!]\rho\} \\
\mathcal{S}^I[\![\mathtt{pre}(E)]\!]\rho &\stackrel{\text{def}}{=} \{\sigma \mid \sigma \nearrow 1 \in \mathcal{S}^I[\![E]\!]\rho\} \\
\mathcal{S}^I[\![E_1 \rightarrow E_2]\!]\rho &\stackrel{\text{def}}{=} \{\sigma_0 \cdot \sigma' \nearrow 1 \mid \sigma \in \mathcal{S}^I[\![E_1]\!]\rho \wedge \\
&\qquad \sigma' \in \mathcal{S}^I[\![E_2]\!]\rho\} \\
\mathcal{S}^I[\![\mathtt{X}]\!]\rho &\stackrel{\text{def}}{=} \rho(X)
\end{aligned}$$

Semantics of an example program

`X = (0 → pre(X)+1)`

- $\mathcal{S}^I[\![0]\!]\rho = 0000000\ldots$
- $\mathcal{S}^I[\![\texttt{pre(X)}]\!]\rho = \{x \cdot \sigma \mid \sigma \in \mathcal{S}^I[\![\texttt{X}]\!]\rho\}$
  $= \{x \cdot \sigma \mid \sigma \in \rho(\texttt{X})\}$
- $\mathcal{S}^I[\![1]\!]\rho = 1111111\ldots$
- $\mathcal{S}^I[\![\texttt{pre(X)} + 1]\!]\rho = \{(x+1) \cdot \lambda i \cdot \sigma_i + 1 \mid \sigma \in \rho(\texttt{X})\}$
- $\mathcal{S}^I[\![0 \to \texttt{pre(X)} + 1]\!]\rho = \{0 \cdot \lambda i \cdot \sigma_i + 1 \mid \sigma \in \rho(\texttt{X})\}$

so letting $\varXi = \rho(\texttt{X})$, we must solve the equation

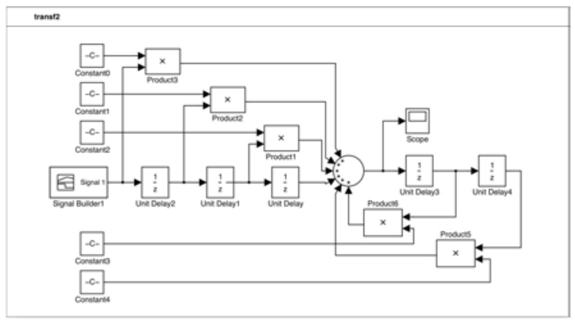
$$\varXi = \{0 \cdot \lambda i \cdot \sigma_i + 1 \mid \sigma \in \varXi\}$$

We proceed iteratively, starting from all possible traces:

$$\begin{aligned}
\varXi^0 &= \{\sigma \mid \sigma \in \mathbb{N} \mapsto \mathbb{Z}\}\\
\varXi^1 &= \{0 \cdot \lambda i \cdot \sigma_i + 1 \mid \sigma \in \varXi^0\}\\
&= \{0 \cdot \sigma \mid \sigma \in \mathbb{N} \mapsto \mathbb{Z}\}\\
\varXi^2 &= \{0 \cdot 1 \cdot \sigma \mid \sigma \in \mathbb{N} \mapsto \mathbb{Z}\}\\
&\ldots\\
\varXi^n &= \{0 \cdot 1 \cdot \ldots \cdot (n-1) \cdot \sigma \mid \sigma \in \mathbb{N} \mapsto \mathbb{Z}\}\\
\varXi^{n+1} &= \{0 \cdot \lambda i \cdot \sigma_i + 1 \mid \sigma \in \varXi^n\}\\
&= \{0 \cdot (0+1) \cdot (1+1) \ldots \cdot ((n-1)+1) \cdot \sigma \mid \sigma \in \mathbb{N} \mapsto \mathbb{Z}\}\\
&= \{0 \cdot 1 \cdot \ldots \cdot n \cdot \sigma \mid \sigma \in \mathbb{N} \mapsto \mathbb{Z}\}\\
&\ldots\\
X^\omega &= \bigcap_{n \geq 0} X^n = \{0 \cdot 1 \cdot \ldots \cdot n \cdot (n+1) \cdot \ldots\}
\end{aligned}$$

Comparative Example of Specification

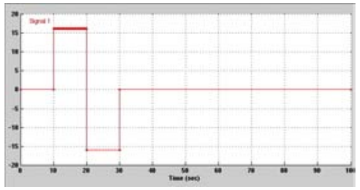
Example of 3-2 filter in Simulink [13]



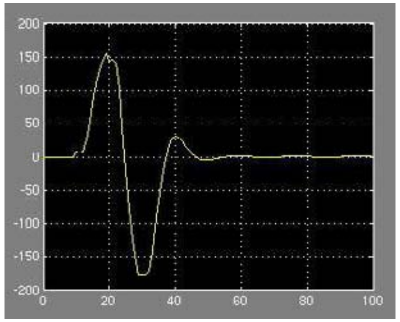
Reference

[13] Simulink®, The MathWorks, Inc.

Sample input



Sample output



3-2 filter specification with trace predicates

$$E[0] = E[1] = E[2] = S[0] = S[1] = S[2] = 0$$
$$\wedge\ \forall i \geq 3 : S[i] = C0 \times E[i-3] + C1 \times E[i-2] + E[i] + C2 \times E[i-1] + C3 \times S[i-1] + C4 \times S[i-2]$$

– Time appears explicitly, which is sometimes considered error-prone and is harmful for model-checking and automatic code generation

3-2 filter specification with temporal logic

$$(E = 0) \wedge \circ(E = 0) \wedge \circ\circ(E = 0) \wedge (S = 0) \wedge \circ(S = 0) \wedge \circ\circ(S = 0) \wedge \Box(\exists e_3 : (E = e_3 \wedge \circ(\exists e_2 : \exists s_2 : (E = e_2) \wedge (S = s_2) \wedge \circ(\exists e_1 : \exists s_1 : (E = e_1) \wedge (S = s_1) \wedge \circ(S = C0 \times e_3 + C1 \times e_2 + C2 \times e_1 + E + C3 \times s_1 + C4 \times s_2)))))$$

– Not really readable (a general default of temporal logics, for example in real life specifications, casual users just state many tautologies)
– Model-checkers (for finite state specifications)
– No automatic code generation tool

3-2 filter specification with a synchronous language

$$S = (0 \rightarrow (0 \rightarrow (0 \rightarrow (C0 \times \mathrm{pre}(\mathrm{pre}(\mathrm{pre}(E))) \; C1 \times \mathrm{pre}(\mathrm{pre}((E)) + C2 \times \mathrm{pre}(E) + E + C3 \times \mathrm{pre}(S) + C4 \times \mathrm{pre}(\mathrm{pre}(S)))))$$

– More readable
– Model-checkers (for finite state programs)
– Automatic code generation tools

THE END

My MIT web site is http://www.mit.edu/~cousot/

The course web site is http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www/.