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

Reference

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.

\[ \text{Entry}[P], \tau[P], \text{Exit}[P] \]

where:

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

\[ \text{The small-step operational semantics of a program } P, \text{ as defined in lecture 5, is a transition system:} \]

\[ \langle \Sigma[P], \tau[P], \text{Entry}[P], \text{Exit}[P] \rangle \]
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):

\[
\begin{align*}
1 & : x \Rightarrow x := ? \\
2 & : x \Rightarrow x < 0 \Rightarrow x := x - 1 \\
3 & : x \Rightarrow x < 0 \Rightarrow x := x - 1 \\
4 & : x \Rightarrow x < 0 \Rightarrow x := x - 1 \\
5 & :
\end{align*}
\]

Does not necessarily starts from entry states.

\[ ^{2} \text{Here infinite is maximal although it is mathematically conceivable to have transfinite traces.} \]

The Partial Trace Semantics

Given a transition system \((\Sigma, \tau)\), the corresponding partial trace semantics is

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

Another common case is that of prefix traces starting from given initial states \(I \in \mathcal{P}(\Sigma)\):

\[ \{ \sigma \in \Sigma^\mathbb{N} \mid n > 0 \land \sigma_0 \in I \land \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \} \]
The Maximal Trace Semantics

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

- **Maximal finite execution traces:**
  $$\{ \sigma \in \Sigma^n \mid n > 0 \land \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \land \sigma_{n-1} \in F \}$$
  where (e.g.) the final states are $F \overset{\text{def}}{=} \{ s \in \Sigma \mid \forall s' \in \Sigma : \neg t(s, s') \}$

- **Maximal infinite execution traces:**
  $$\{ \sigma \in \Sigma^\omega \mid \forall i \geq 0 : \tau(\sigma_i, \sigma_{i+1}) \}$$

- **Maximal bifinite execution traces:**
  $$\{ \sigma \in \Sigma^n \mid n > 0 \land \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \land \sigma_{n-1} \in F \} \cup \{ \sigma \in \Sigma^\omega \mid \forall i \geq 0 : \tau(\sigma_i, \sigma_{i+1}) \}$$

- 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 $(\Sigma, \tau)$, the corresponding big-step operational semantics is:

- **Maximal finite execution traces:**
  $$\{ (\sigma_0, \sigma_{n-1}) \mid \exists n > 0 : \sigma \in \Sigma^n \land \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \}$$

- **Maximal infinite execution traces:**
  $$\{ (\sigma_0, \bot) \mid \sigma \in \Sigma^\omega \land \forall i \geq 0 : \tau(\sigma_i, \sigma_{i+1}) \}$$

- A special case consists in restricting to initial states $\sigma_0 \in I$ that is $I \upharpoonright \tau^*$

The Natural Denotational Semantics

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

- **Maximal finite execution traces:**
  $$\{ \sigma_0 \in I \} \cup \{ (\sigma_0, \sigma_{n-1}) \mid \exists n > 0 : \sigma \in \Sigma^n \land \forall i \in [0, n-2] : \tau(\sigma_i, \sigma_{i+1}) \land \sigma_{n-1} \in F \}$$

- **Maximal infinite execution traces:**
  $$\{ \sigma \in \Sigma^\omega \mid \forall i \geq 0 : \tau(\sigma_i, \sigma_{i+1}) \}$$

where the final states are

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

i.e. blocking states

---

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

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 Operational Semantics

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

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

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

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

where $\text{post}[\tau]X \overset{\text{def}}{=} \{y \mid \exists x \in X : \tau(x, y)\}$.

The Demoniac Denotational Semantics

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

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

$\cup \{\langle \sigma_0, s' \rangle \mid \sigma \in \Sigma^\omega \land \forall i \geq 0 : \tau(\sigma_i, \sigma_{i+1}) \land s' \in \Sigma \cup \{\bot\}\}$$

TheDemoniac Denotational Semantics: where “demoniac” 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 upon final states only in case of definite termination.

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

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

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

where $\text{post}[\tau]X \overset{\text{def}}{=} \{y \mid \exists x \in X : \tau(x, y)\}$.
Backward Reachability Semantics

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

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

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

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

\[\text{Again, the final states need not be exit states}\]

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

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

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

Example of Backward Reachability Semantics

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:

- Backward reachability
- Forward reachability
- Natural operational
- Big-step operational
- Angelic denotational
- Demoniac denotational
- Small-step operational
- Natural denotational
- Partial traces
- Maximal traces

Reference

Program Specification, Semantics and Correctness

Semantic Domain and Program Semantics

- The **semantic domain** is \( S \) which elements describes possible program executions
- For example, if executions of a program are described by a set of traces on states \( \Sigma \) then \( S = \wp(\Sigma^{\infty}) \)
- The **semantics** \( \text{Sem}[P] \in S \) 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 \( \wp(S) \)
- A **program specification** \( \text{Spec} \) is a formal description of desired program property so \( \text{Spec} \in \wp(S) \)

Program Correctness

- The **correctness** of a program \( P \) with respect to a specification \( \text{Spec} \) is \( \text{Sem}[P] \subseteq \text{Spec} \)
- The intuition is that the program semantics has the desired property as stated by the specification
Trace properties

\[ S = \wp(\Sigma^{\infty}) \]
\[ \text{Sem}[P] \in S \text{ so } \text{Sem}[P] \in \wp(\Sigma^{\infty}) \]
\[ \text{Spec} \in \wp(S) \text{ so } \text{Spec} \in \wp(\wp(\Sigma^{\infty})) \]

Correctness: \( \text{Sem}[P] \in \text{Spec} \)

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

- Spec \( \in \wp(\Sigma^{\infty}) \)
- Universal correctness: \( \text{Sem}[P] \subseteq \text{Spec} \)

Existential correctness: \( \text{Sem}[P] \cap \text{Spec} \neq \emptyset \)

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

Relational properties

\[ \text{Sem}[P] \in \{X \in \wp(\Sigma^{\infty}) \mid \text{Spec} \cap X \neq \emptyset\} \]
Relational properties

- $S = \rho(\Sigma \times (\Sigma \cup \{\bot\}))$
- $\text{Sem}[P] \in S$ so $\text{Sem}[P] \in \rho(\Sigma \times (\Sigma \cup \{\bot\}))$
- $\text{Spec} \in \rho(S)$ so $\text{Spec} \in \rho(\rho(\Sigma \times (\Sigma \cup \{\bot\}))$
- Correctness: $\text{Sem}[P] \subseteq \text{Spec}$

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

- $\text{Spec} \in \rho(\rho(\Sigma \times (\Sigma \cup \{\bot\}))$
- Correctness: $\text{Sem}[P] \subseteq \text{Spec}$ or $\text{Sem}[P] \cap \text{Spec} ≠ \emptyset$

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

Example of relational property:

- Denotational semantics: $\text{Sem}[P] \in \rho(\Sigma \times (\Sigma \cup \{\bot\}))$
- Specification: $\text{Spec} \in \rho(\Sigma \times \Sigma)$
- Total correctness: $\text{Sem}[P] \subseteq \text{Spec}$

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

Example of relational property:

- Partial correctness

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

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

State properties
State properties

- $S = \rho(\Sigma)$
- $\text{Sem}[P] \in S$ so $\text{Sem}[P] \in \rho(\Sigma)$
- Spec $\in \rho(\Sigma)$ so Spec $\in \rho(\rho(\Sigma))$
- Correctness: $\text{Sem}[P] \subseteq \text{Spec}$

In practice, a weaker form of correctness specification (called \textit{state properties}):

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

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

Example of existential state property: runtime error

- Forward reachability semantics:
  \[
  \text{Sem}[P] \overset{\text{def}}{=} \text{post}[\tau[P]^*]I \in \rho(\Sigma)
  \]
- Specification: $\text{Error} \in \rho(\Sigma)$ (erroneous states)
- Presence of run-time error: $\text{Sem}[P] \cap \text{Error} \neq \emptyset$
  \text{There is at least one possible execution of the program which will reach an erroneous state}
- Absence of run-time error: $\text{Sem}[P] \subseteq \neg \text{Error}$
  \text{No possible possible execution of the program can reach an erroneous state}

Example of universal state property: invariance

- Forward reachability semantics:
  \[
  \text{Sem}[P] \overset{\text{def}}{=} \text{post}[\tau[P]^*]I \in \rho(\Sigma)
  \]
- Specification: $\text{Spec} \in \rho(\Sigma)$
- Invariance: $\text{Sem}[P] \subseteq \text{Spec}$

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

\[
\text{post}[\tau[P]^*]I \subseteq \text{Spec} \iff I \subseteq \text{pre}[\tau[P]^*]\text{Spec} \quad \text{where} \quad \text{pre}[\tau]X \triangleq \neg \text{pre}[\tau](\neg X)
\]
\[
\iff \forall s, s' \in \Sigma: [s \in I \land \tau[P]^*(s, s')] \implies s' \in \text{Spec}
\]

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

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: \( \text{Env}[P] \overset{\text{def}}{=} \text{Var}[P] \rightarrow \mathbb{I}_\Omega \)
- Program components: \( \text{Cmp}[P] \)
- labels: \( \text{Lab} \)
- Program labels: \( \text{in}_P \in \text{Cmp}[P] \rightarrow \varphi(\text{Lab}) \)

Set of States Predicate Logic

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

1: 
\[ X := 0; \]  
\[ \text{at}[1:] \land \text{Ierr}[X] \]

2: 
\[ \text{while} \ (X < 10) \ do \]
\[ \lor \ \text{at}[2:] \land 0 \leq X \land X \leq 10 \]

3: 
\[ X := X + 1 \]
\[ \lor \ \text{at}[3:] \land 0 \leq X \land X \leq 9 \]

4: 
\[ \text{od} \]
\[ \lor \ \text{at}[4:] \land 1 \leq X \land X \leq 10 \]

5: 
\[ \text{at}[5:] \land X = 10 \]
Extending the syntax of predicates

Formally, the syntax of terms in first order predicates is extended to include
- Program variables: \([\text{Var}[P]]\)
- Errors: \([\text{Ierr}[X]], \text{Aerr}[X]\)

The syntax of atomic formulæ is extended with:
- Control atomic formulæ: \([\text{at}[\ell]], \ell \in \text{in}_P\)

Extending assignments

Let \(\mathcal{C}\) be a fresh so-called control variable such that \(\mathcal{C} \notin \text{Var}[P]\).

An assignment \(\rho\) maps variables in \(\text{Var}[P] \cup \{\mathcal{C}\}\) as follows:
- \(\rho(X) \in \bigcup_i \Omega_i, X \in \text{Var}[P]\), memory state
- \(\rho(\mathcal{C}) \in \text{Lab}\), control state

Extending the semantics of predicates

The interpretation of terms defined in lecture 6 is extended with
- \(S^I[\text{Ierr}[X]]\rho \overset{\text{def}}{=} (\rho(X) = \Omega_1)\) (initialization error)
- \(S^I[\text{Aerr}[X]]\rho \overset{\text{def}}{=} (\rho(X) = \Omega_a)\) (arithmetic error)
- (and has \(S^I[X]\rho = \rho(X)\), as usual)

The interpretation of atomic formulæ is extended with
- \(S^I[\text{at}[\ell]]\rho \overset{\text{def}}{=} (\rho(\mathcal{C}) = \ell)\) (control is at \(\ell\))

Models of state predicates

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

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

\[
= \{\langle \rho(\mathcal{C}), \lambda X \in \text{Var}[P] \cdot \rho(X) \rangle \mid \rho \models \Phi \}\}
\]

\[\text{\textsuperscript{12}}\text{This is satisfiability } (\rho \models \Phi) \overset{\text{def}}{=} (S^I[\Phi]\rho = \text{tt}).\]
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:
  - Unprimed variables and statements for the first state
  - Primed variables and statements for the second state

Extending the syntax of predicates

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

- (Primed) program variables: \( \text{Var}[P], \{X' \mid X \in \text{Var}[P]\} \)
- Mathematical variables: \( x \in \mathcal{V} \)
- Errors: \( \text{Ierr}[X], \text{Aerr}[X] \)

while atomic formulae also include:

- Control atomic formulæ: \( \text{at}[\ell], \text{at}'[\ell], \ell \in \text{in}_P \)

Extending assignments

To define the interpretation of formulæ, let \( \mathcal{C}, \mathcal{C}' \) be a fresh so-called control variables.

An assignment \( \rho \) maps variables in \( \text{Var}[P] \cup \{X' \mid X \in \text{Var}[P]\} \cup \{\mathcal{C}, \mathcal{C}'\} \cup \mathcal{V} \) as follows:

- \( \rho(x) \in \mathbb{Z}, x \in \mathcal{V} \)
- \( \rho(X), \rho(X') \in \mathbb{I}_P, X \in \text{Var}[P] \), memory states
- \( \rho(\mathcal{C}), \rho(\mathcal{C}') \in \mathbb{L}_P \), control states

Footnotes:

13 Another inverse convention is primed variables for the first state and unprimed one for the second. Another convention is that of a prime 'X for the first state and a postprime 'X for the second. One can also use indexes like \( x_0, x_1, \ldots, X, \ldots \).

14 Different from the (primed) program variables in that \( \mathcal{V} \cap (\text{Var}[P] \cup \{X' \mid X \in \text{Var}[P]\}) = \emptyset \)

15 Different from the (primed) program and mathematical variables in that \( \{\mathcal{C}, \mathcal{C}'\} \cap (\text{Var}[P] \cup \{X' \mid X \in \text{Var}[P]\} \cup \mathcal{V}) = \emptyset \)
Extending the semantics of predicates

The interpretation of terms defined in lecture 6 is extended with

- \( S^I[I_{\text{err}}[X]]\rho \stackrel{\text{def}}{=} (\rho(X) = \Omega_1) \) (initialization error)
- \( S^I[A_{\text{err}}[X]]\rho \stackrel{\text{def}}{=} (\rho(X) = \Omega_2) \) (arithmetic error)
- (and has \( S^I[X]\rho = \rho(X) \), as usual)

while for atomic formulae, we have

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

\( \Omega_1, \Omega_2 \) are the interpretations of \( \text{err} \) and \( A_{\text{err}} \), respectively.

\( \Omega_1, \Omega_2 \) are the interpretations of \( \text{err} \) and \( A_{\text{err}} \), respectively.

Example of state relation described by a predicate

The classical program invariants:

1. \{ \( X = x_0 \ \& \ Y = y_0 \ \& \ x_0 \geq y_0 \) \}
   \( Z := X \)
2. \while \( (Z < y) \) do
3. \{ \( X = x_0 \geq Z > Y = y_0 \) \}
   \( Z := Z - 1 \)
4. \end
5. can be specified by the predicate:

\( \Rightarrow (X = x_0 \geq Z > Y = y_0) \)

Models of state relation predicates

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

\[
\{(\rho(\mathcal{C}'), \lambda X \in \text{Var}[P].\rho(X')), (\rho(\mathcal{C}), \lambda X \in \text{Var}[P].\rho(X)) \mid S^I[\Phi]\rho = \top\}
\]

\[
= \{(\rho(\mathcal{C}'), \lambda X \in \text{Var}[P].\rho(X')), (\rho(\mathcal{C}), \lambda X \in \text{Var}[P].\rho(X)) \mid \rho \models \Phi\}
\]

\( \top, \bot \) are truth and falsehood judgments, respectively.

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 \( X[t] \) to denote the value of the program variable \( X \) at time \( t \in \mathbb{N} \)
  - We use \( \text{at}[\ell][t] \) to specify where the control stands at time \( t \in \mathbb{N} \)

---

Extending the syntax of predicates

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

- Mathematical variables: \( x \in \mathcal{V} \)
- Program variables: \( X \in \mathcal{V} \) \( [P] \) \(^{19}\)
- Terms \( t \in \mathcal{T} \):
  \[
  t ::= c \\
  \quad | \ x \\
  \quad | \ f(t_1, \ldots, t_n) \\
  \quad | \ X[t]
  \]

---

Extending assignments

To define the interpretation of formulæ, let \( \mathcal{C} \) be a fresh so-called control variable\(^{20}\).

An assignment \( \rho \) maps variables in \( \mathcal{V} \cup \mathcal{P} \cup \mathcal{C} \) as follows:

- \( \rho(x) \in \mathcal{D}_I, \ x \in \mathcal{V} \)
- \( \rho(X) \in \mathcal{N} \mapsto \mathbb{I}_\Omega, \ X \in \mathcal{P} \), timed memory states
- \( \rho(\mathcal{C}) \in \mathcal{N} \mapsto \text{Lab} \), timed control states

---

\(^{18}\) Finite ones can be encoded using an undefined value \( \ddot{\ldots} \): \( \text{abc::xyz} \) becomes \( \text{abc::xyz} \ddot{\ldots} \ddot{\ldots} \ddot{\ldots} \)

\(^{19}\) Assuming \( \mathcal{V} \cap \mathbb{P}[P] = \emptyset \)

\(^{20}\) Different from the program and mathematical variables in that \( \mathcal{C} \not\in \mathcal{V} \cup \mathcal{P} \cup \mathcal{C} \)
Extending the semantics of predicates

The interpretation of terms defined in lecture 6 is extended with

\[- S_I[X|t]|\rho = \rho(X)(S_I[t]|\rho) \]  \(21\)

while for atomic formulae, we have

\[- S_I[\text{at}][t]|\rho = (\rho(\mathcal{C})(S_I[t]|\rho) = \ell) \]

\[- S_I[\text{err}[X][t]|\rho = (\rho(X)(S_I[t]|\rho) = \Omega_1) \]  \(21\) (initialization error)

\[- S_I[\text{err}[X][t]|\rho = (\rho(X)(S_I[t]|\rho) = \Omega_0) \]  \(21\) (arithmetic error)

\[- \text{(and has } S_I[X|t]|\rho = \rho(X), \text{ as usual)} \]

Models of trace predicates

Now a first-order formula \(\Phi\) with a given interpretation \(I\) is understood to describe traces, as follows:

\[\{\lambda i \in \mathbb{N}. (\rho(\mathcal{C})(i), \lambda X \in \text{Var}[P] . \rho(X)(i)) \mid S_I[\Phi]|\rho = t\} \]

\[= \{\lambda i \in \mathbb{N}. (\rho(\mathcal{C})(i), \lambda X \in \text{Var}[P] . \rho(X)(i)) \mid \rho \models \Phi\} \]  \(22\)

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:

\[\forall i, j : ((i \leq j) \land \neg\text{at}[1 : [i]]) \implies (Z[i] \geq Z[j])\]
Future, Past and Bidirectional Traces

- We have considered future traces

\[ \bullet \bullet \bullet \rightarrow \ldots \]

to specify what can happen from now on

- Past traces

\[ \ldots \bullet \rightarrow \bullet \bullet \bullet \bullet \rightarrow \bullet \]

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

– Bidirectional traces [5]

\[ \ldots \bullet \bullet \rightarrow \bullet \bullet \rightarrow \bullet \rightarrow \ldots \]

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

Reference


Linear Time Temporal Logic

Amir Pnueli

Reference

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


Linear Temporal Operators

- An atomic predicate $p$ means that the current state in the trace satisfies $p$
- $\Diamond \Phi$ means that $\Phi$ holds at next time in the trace.

- $\Box \Phi$ means that from now on, the trace satisfies $\Phi$.

- $\Phi_1 \cup \Phi_2$ means that $\Phi_1$ always holds until the trace satisfies $\Phi_2$.

- $\Diamond \Phi$ means that some time in the future, the trace satisfies $\Phi$. 

Diagram representations are not provided in the natural text format.
Linear Temporal Logic Syntax

\[ v, u \in \mathcal{V} \quad \text{Variables} \]
\[ p \quad \text{State formula (first order predicate)} \]
\[ \Phi ::= \begin{array}{c} p \\
\neg \Phi \\
\Phi_1 \lor \Phi_2 \\
\exists u : \Phi \\
\Diamond \Phi \\
\Phi_1 \mathcal{U} \Phi_2 \end{array} \quad \text{LTL formula} \]


Linear Temporal Logic Semantics

Let \( I \) be an interpretation of the first-order logic (where \( \Sigma \defeq \mathcal{V} \rightarrow D_I \)), the semantics \( S^I[\Phi] \) of a LTL formula \( \Phi \) is

\[
S^I[p] \defeq \{ \sigma \in \Sigma^\omega | S^I[p] \sigma_0 = \top \}
\]
\[
S^I[\neg \Phi] \defeq \Sigma^\omega \setminus S^I[\Phi]
\]
\[
S^I[\Phi_1 \lor \Phi_2] \defeq S^I[\Phi_1] \cup S^I[\Phi_2]
\]
\[
S^I[\exists u : \Phi] \defeq \bigcup_{d \in D_I} \{ \sigma[u := d] | \sigma \in S^I[\Phi] \}
\]

where \( \sigma[u := d] \defeq \lambda i \in \mathbb{N}. \sigma_i[u := d] \)

Trace Suffix

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

\[
\sigma \nearrow k \defeq \sigma_k \sigma_{k+1} \sigma_{k+2} \ldots
\]

In particular \( \sigma \nearrow 0 = \sigma \)

\[
S^I[\Diamond \Phi] \defeq \{ \sigma \in \Sigma^\omega | \sigma \nearrow 1 \in S^I[\Phi] \}
\]
\[
S^I[\Phi_1 \mathcal{U} \Phi_2] \defeq \{ \sigma \in \Sigma^\omega | \exists k \in \mathbb{N} : \forall i \in [0, k - 1] : \sigma \nearrow i \in S^I[\Phi_1] \wedge \sigma \nearrow k \in S^I[\Phi_2] \} \]
### LTL Auxiliary Operators

- $\Diamond \Phi \equiv \top \mathcal{U} \Phi$ 
  - eventually/sometime
- $\Box \Phi \equiv \neg(\Diamond \neg \Phi)$ 
  - always/henceforth
- $\Phi_1 \mathcal{W} \Phi_2 \equiv \Phi_1 \mathcal{U} \Phi_2 \lor \Box \Phi_1$ 
  - waiting for/unless

### The semantics of the LTL formula

- $S'[Z = u]$ 
  - $\{\sigma \mid \sigma_0(Z) \leq \sigma(u)\}$
- $S'[Z = u] \equiv \{\sigma \mid \sigma_0(Z) \leq \sigma(u)\}$
- $S'[\neg \text{at}[1:] \land Z = u]$ 
  - $\{\sigma_0(\mathcal{C}) \neq 1 : \land \sigma_0(Z) = \sigma_0(u)\}$
- $S'[\neg \text{at}[1:] \land Z = u] \implies [\Box(Z \leq u)]$ 
  - $\{\sigma_0(\mathcal{C}) \neq 1 : \land \sigma_0(Z) = \sigma_0(u)\}$

### 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:

$$\Box(\forall u : (\neg \text{at}[1:] \land Z = u) \implies [\Box(Z \leq u)])$$

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 Formulae

- $\Phi_1 \implies \Diamond \Phi_2$
  if $\Phi_1$ holds now then $\Phi_2$ eventually holds later

- $\Box(\Phi_1 \implies \Diamond \Phi_2)$
  whenever $\Phi_1$ holds, $\Phi_2$ holds in next state

- $\Box(\Phi_1 \implies \Phi_2)$
  once $\Phi_1$ holds, $\Phi_2$ eventually holds

- $\Box(\Phi_1 \implies \Box \Phi_2)$
  once $\Phi_1$ holds, $\Phi_2$ always holds

Temporal tautologies

- $\Box \Phi = \neg (\Diamond \neg \Phi)$

- $\Box \Phi = \Phi \lor \Box \Phi$

- $\Phi_1 \cup \Phi_2 = (\Phi_1 \lor \Phi_2) \land \Diamond \Phi_2$

- $\Box \Phi = \Phi \land \Box \Phi$

- $\Diamond \Phi = \Phi \lor \Box \Phi$

- $\Phi_1 \cup \Phi_2 = \Phi_2 \lor (\Phi_1 \land \Box \Phi_2)$

- $\Phi_1 \lor \Phi_2 = \Phi_2 \lor (\Phi_1 \land \Box \Phi_2)$

- $\Phi = \Box \Phi$

- $\Box \Phi \implies \Phi$

- $\Phi \implies \Diamond \Phi$

- $\Diamond \Phi \implies \Box \Phi$

- $\Box \Phi \implies \Diamond \Phi$

- $\Box \Phi \implies \Box \Phi$

- $\Box \Phi \implies \Diamond \Phi$

- $\Box \Phi \implies \Box \Phi$

- $\Box \Phi \implies \Diamond \Phi$

- $\Box \Phi \implies \Box \Phi$
- $\Diamond (\Phi_1 \land \Phi_2) \iff (\Diamond \Phi_1) \land (\Diamond \Phi_2)$
- $\Diamond (\Phi_1 \lor \Phi_2) \iff (\Diamond \Phi_1) \lor (\Diamond \Phi_2)$
- $\Diamond (\Phi_1 \land \Phi_2) \iff (\Diamond \Phi_1) \land (\Diamond \Phi_2)$
- $\Diamond (\Phi_1 \lor \Phi_2) \iff (\Diamond \Phi_1) \lor (\Diamond \Phi_2)$
- $\Diamond (\Phi_1 \lor \Phi_2) \land (\Phi_1 \land \Phi_2) \iff (\Phi_1 \lor \Phi_2) \land (\Phi_1 \land \Phi_2)$
- $\Diamond (\Phi_1 \land \Phi_2) \lor (\Phi_1 \lor \Phi_2) \iff (\Phi_1 \land \Phi_2) \lor (\Phi_1 \lor \Phi_2)$
- $\Diamond (\Phi_1 \lor \Phi_2) \lor (\Phi_1 \land \Phi_2) \iff (\Phi_1 \lor \Phi_2) \lor (\Phi_1 \land \Phi_2)$
- $\Diamond (\Phi_1 \lor \Phi_2) \land (\Phi_1 \land \Phi_2) \iff (\Phi_1 \lor \Phi_2) \land (\Phi_1 \land \Phi_2)$
- $\Diamond \Box (\Phi_1 \lor \Phi_2) \iff \Box (\Diamond \Phi_1) \lor (\Diamond \Phi_2)$
- Synchronous Languages

$\Phi_1 U (\exists u : \Phi_2) \iff \exists u : (\Phi_1 U \Phi_2)$ $u \not\in \text{FV}[\Phi_1]$ ²³

$- (\forall u : \Phi_1) U \Phi_2 \iff \forall u : (\Phi_1 U \Phi_2)$ $u \not\in \text{FV}[\Phi_2]$

$- (\forall u : \Phi_1) U \Phi_2 \iff \forall u : \Phi_1 U \Phi_2$ $u \not\in \text{FV}[\Phi_2]$

$- (\forall u : \Phi_1) U \Phi_2 \iff \forall u : \Phi_1 U \Phi_2$ $u \not\in \text{FV}[\Phi_2]$

In general $\Diamond (\forall u : \Phi) \iff \forall u : (\Diamond \Phi)$, a counter example is:

$\sigma = \{X \rightarrow 0, U \rightarrow 0\}, \{X \rightarrow 0, U \rightarrow 1\}, \ldots, \{X \rightarrow 0, \rightarrow n\}, \ldots$

$\sigma \models \forall u > 0 : (\Diamond (X \neq U \land U = u))$

²³ Recall that FV[\Phi] is the set of free variables of \Phi.
References


Stream/synchronous languages

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

Example

The time diagram

\[ X \]

\[ Y \]

can be specified in LUSTRE [11] as

\[ Y = X \text{ and not } \preceq(X) \]

that is

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

or better

\[ Y = \text{false} \rightarrow X \text{ and not } \preceq(X) \]

that is

\[
\begin{cases}
Y(0) = \text{ff} \\
Y(n+1) = X(n+1) \land \neg X(n) & n \geq 0
\end{cases}
\]

Reference

Syntax of a (subset\textsuperscript{25}) of LUSTRE

\begin{align*}
X & ::= D P | D \\
P & ::= X = E \\
D & ::= E \\
E & ::= \text{expression} \\
& ::= f/n(E_1, \ldots, E_n) \\
& \mid \text{pre}(E) \\
& \mid E_1 \to E_2 \\
& \mid X
\end{align*}

\textsuperscript{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).

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 \( \mathbb{B} \), \( \ldots \))
- 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:

\[
S^I[P] \in \rho \left( \prod_{X \in \text{Var}[P]} \mathbb{N} \mapsto D_I \right)
\]

\[
S^I[f/n(E_1, \ldots, E_n)] \rho \overset{\text{def}}{=} \{ T^I[f/n](\sigma_1, \ldots, \sigma_n) \mid \forall i \in [1, n] : \sigma_i \in S^I[E_i] \rho \}
\]

\[
S^I[\text{pre}(E)] \rho \overset{\text{def}}{=} \{ \sigma \mid \sigma \not\rightarrow 1 \in S^I[E] \rho \}
\]

\[
S^I[E_1 \to E_2] \rho \overset{\text{def}}{=} \{ \sigma \cdot \sigma' \not\rightarrow 1 \mid \sigma \in S^I[E_1] \rho \land \sigma' \in S^I[E_2] \rho \}
\]

\[
S^I[X] \rho \overset{\text{def}}{=} \rho(X)
\]

\textsuperscript{26} There is a mathematical difficulty here that was elucidated when studying fixpoint definitions. Here we choose the \textit{\( \leq \)-greatest fixpoint.}
Semantics of an example program

\[ X = (0 \rightarrow \text{pre}(X)+1) \]

\[ S^I[0] \rho = 0000000 \ldots \]
\[ S^I[\text{pre}(X)] \rho = \{ x \cdot \sigma \mid \sigma \in S^I[X] \rho \} = \{ x \cdot \sigma \mid \sigma \in \rho(X) \} \]
\[ S^I[1] \rho = 1111111 \ldots \]
\[ S^I[\text{pre}(X)+1] \rho = \{(x+1) \cdot \lambda i \cdot \sigma i + 1 \mid \sigma \in \rho(X)\} \]
\[ S^I[0 \rightarrow \text{pre}(X)+1] \rho = \{0 \cdot \lambda i \cdot \sigma i + 1 \mid \sigma \in \rho(X)\} \]

so letting \( \mathcal{E} = \rho(X) \), we must solve the equation
\[ \mathcal{E} = \{0 \cdot \lambda i \cdot \sigma i + 1 \mid \sigma \in \mathcal{E}\} \]

We proceed iteratively, starting from all possible traces:

\[ \mathcal{E}^0 = \{\sigma \mid \sigma \in \mathbb{N} \rightarrow \mathbb{Z}\} \]
\[ \mathcal{E}^1 = \{0 \cdot \lambda i \cdot \sigma i + 1 \mid \sigma \in \mathcal{E}^0\} = \{0 \cdot \sigma \mid \sigma \in \mathbb{N} \rightarrow \mathbb{Z}\} \]
\[ \mathcal{E}^2 = \{0 \cdot 1 \cdot \sigma \mid \sigma \in \mathbb{N} \rightarrow \mathbb{Z}\} \]

\[ \ldots \]
\[ \mathcal{E}^n = \{0 \cdot 1 \ldots \cdot (n-1) \cdot \sigma \mid \sigma \in \mathbb{N} \rightarrow \mathbb{Z}\} \]
\[ \mathcal{E}^{n+1} = \{0 \cdot \lambda i \cdot \sigma i + 1 \mid \sigma \in \mathcal{E}^n\} = \{0 \cdot (0 + 1) \cdot (1 + 1) \ldots \cdot ((n-1) + 1) \cdot \sigma \mid \sigma \in \mathbb{N} \rightarrow \mathbb{Z}\} = \{0 \cdot 1 \ldots \cdot n \cdot \sigma \mid \sigma \in \mathbb{N} \rightarrow \mathbb{Z}\} \]

\[ X^\omega = \bigcap_{n \geq 0} X^n = \{0 \cdot 1 \ldots \cdot n \cdot (n+1) \ldots \} \]

Comparative Example of Specification

Example of 3-2 filter in Simulink [13]

Reference

3-2 filter specification with temporal logic

\[(E = 0) \land \diamond (E = 0) \land \diamond \diamond (E = 0) \land (S = 0) \land \diamond (S = 0) \land \diamond (S = 0) \land \Box (\exists e_3 : (E = e_3 \land \diamond \exists e_2 : \exists s_2 : (E = e_2) \land (S = s_2) \land \diamond (\exists e_1 : \exists s_1 : (E = e_1) \land (S = s_1) \land \diamond (S = C_0 \times e_3 + C_1 \times e_2 + C_2 \times e_1 + E + C_3 \times s_1 + C_4 \times s_2))))\]

- 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 trace predicates

\[\land \forall i \geq 3 : S[i] = C_0 \times E[i - 3] + C_1 \times E[i - 2] + E[i] + C_2 \times E[i - 1] + C_3 \times S[i - 1] + C_4 \times S[i - 2]\]

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

3-2 filter specification with a synchronous language

\[S = (0 \rightarrow (0 \rightarrow (0 \rightarrow (C_0 \times \text{pre(pre(E))))))\]
\[C_1 \times \text{pre(E)} + C_2 \times \text{pre(E)} + E + C_3 \times \text{pre(S)} + C_4 \times \text{pre(S)}))\]

- 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/.