Semantics and Verification of Software

Summer Semester 2015

Lecture 19: Wrap-Up

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

http://moves.rwth-aachen.de/teaching/ss-15/sv-sw/

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Outline of Lecture 19

Recap: Correctness Properties for Execution Time

Soundness and Completeness

Outlook: Semantics of Functional Programming Languages

Outlook: Semantics of Logic Programming Languages

Miscellaneous

Recap: Correctness Properties for Execution Time

Semantics of Timed Correctness Properties

Definition (Semantics of timed correctness properties (extends Definition 11.1))

Let $A, B \in Assn$, $c \in Cmd$, and $e \in AExp$. Then $\{A\}\, c\, \{e \Downarrow B\}$ is called valid (notation: $\models \{A\}\, c\, \{e \Downarrow B\}$) if there exists $k \in \mathbb{N}$ such that for each $I \in Int$ and each $\sigma \models^I A$, there exist $\sigma' \in \Sigma$ and $\tau \leq k \cdot \mathfrak{A}[\![e]\!]\sigma$ such that $\langle c, \sigma \rangle \xrightarrow{\tau} \sigma'$ and $\sigma' \models^I B$

Note: e is evaluated in initial (rather than final) state

Proving Timed Correctness

Definition (Hoare Logic for timed correctness (extends Definition 11.3))

The Hoare rules for timed correctness are given by (where $i, u \in LVar$)

$$\text{(skip)}\ \frac{}{\{A\}\ \texttt{skip}\ \{1 \Downarrow A\}} \qquad \text{(asgn)}\ \frac{}{\{A[x \mapsto a]\}\ x := a\ \{1 \Downarrow A\}}$$

$$\text{(seq)}\ \frac{\{A \wedge e_2' = u\}\ c_1\ \{e_1 \Downarrow C \wedge e_2 \leq u\} \quad \{C\}\ c_2\ \{e_2 \Downarrow B\}}{\{A\}\ c_1 ; c_2\ \{e_1 + e_2' \Downarrow B\}}$$

$$\text{(if)}\ \frac{\{A \wedge b\}\ c_1\ \{e \Downarrow B\} \quad \{A \wedge \neg b\}\ c_2\ \{e \Downarrow B\}}{\{A\}\ \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}\ \{e \Downarrow B\}}$$

$$\text{(while)}\ \frac{\{i \geq 0 \wedge A(i+1) \wedge e' = u\}\ c\ \{e_0 \Downarrow A(i) \wedge e \leq u\}}{\{\exists i . i \geq 0 \wedge A(i)\}\ \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}\ \{e \Downarrow A(0)\}}$$

where $\models (i \geq 0 \wedge A(i+1)) \Rightarrow (b \wedge e \geq e_0 + e')$ and $\models A(0) \Rightarrow (\neg b \wedge e \geq 1)$

$$\text{(cons)}\ \frac{\models (A \Rightarrow (A' \wedge \exists k \in \mathbb{N} . e' \leq k \cdot e)) \quad \{A'\}\ c\ \{e' \Downarrow B'\} \quad \models (B' \Rightarrow B)}{\{A\}\ c\ \{\Downarrow e\} B}$$

Outline of Lecture 19

Recap: Correctness Properties for Execution Time

Soundness and Completeness

Outlook: Semantics of Functional Programming Languages

Outlook: Semantics of Logic Programming Languages

Miscellaneous

Soundness and Completeness

Theorem 19.1 (Soundness)

For every timed correctness property $\{A\}\, c\, \{e \Downarrow B\}$,

$$\vdash \{A\}\, c\, \{e \Downarrow B\} \quad \Rightarrow \quad \models \{A\}\, c\, \{e \Downarrow B\}.$$

Proof.

on the board (by structural induction on derivation; only (while) rule) □

Soundness and Completeness

Theorem 19.1 (Soundness)

For every timed correctness property $\{A\}\,c\,\{e \Downarrow B\}$,

$$\vdash \{A\}\,c\,\{e \Downarrow B\} \quad \Rightarrow \quad \models \{A\}\,c\,\{e \Downarrow B\}.$$

Proof.

on the board (by structural induction on derivation; only (while) rule) □

Theorem 19.2 (Relative completeness)

The Hoare Logic for timed correctness properties is relatively complete, i.e., for every $\{A\}\,c\,\{e \Downarrow B\}$:

$$\models \{A\}\,c\,\{e \Downarrow B\} \quad \Rightarrow \quad \vdash \{A\}\,c\,\{e \Downarrow B\}.$$

Proof.

omitted □

Outline of Lecture 19

Semantics of Functional Programming Languages I

- Program = list of function definitions

Semantics of Functional Programming Languages I

- Program = list of function definitions
- Simplest setting: first-order function definitions of the form

$$f(x_1, \ldots, x_n) = t$$

  - function name f
  - formal parameters $x_1, \ldots, x_n$
  - term t over (base and defined) function calls and $x_1, \ldots, x_n$

Semantics of Functional Programming Languages I

- Program = list of function definitions
- Simplest setting: first-order function definitions of the form

$$f(x_1, \ldots, x_n) = t$$

  - function name f
  - formal parameters $x_1, \ldots, x_n$
  - term t over (base and defined) function calls and $x_1, \ldots, x_n$
- Operational semantics (only function calls; for terms t_i, numbers z_j and variables x_k)
  - call-by-value case:

$$\frac{t_1 \to z_1 \quad \ldots \quad t_n \to z_n \quad t[x_1 \mapsto z_1, \ldots, x_n \mapsto z_n] \to z}{f(t_1, \ldots, t_n) \to z}$$

  - call-by-name case:

$$\frac{t[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n] \to z}{f(t_1, \ldots, t_n) \to z}$$

Semantics of Functional Programming Languages II

- Denotational semantics
  - program = equation system (for functions)
  - induces call-by-value and call-by-name functional
  - monotonic and continuous w.r.t. graph inclusion
  - semantics := least fixpoint (Tarski/Knaster Theorem)
  - coincides with operational semantics

Semantics of Functional Programming Languages II

- Denotational semantics
  - program = equation system (for functions)
  - induces call-by-value and call-by-name functional
  - monotonic and continuous w.r.t. graph inclusion
  - semantics := least fixpoint (Tarski/Knaster Theorem)
  - coincides with operational semantics
- Extensions: higher-order types, data types, ...

Semantics of Functional Programming Languages II

- Denotational semantics
  - program = equation system (for functions)
  - induces call-by-value and call-by-name functional
  - monotonic and continuous w.r.t. graph inclusion
  - semantics := least fixpoint (Tarski/Knaster Theorem)
  - coincides with operational semantics
- Extensions: higher-order types, data types, ...
- see [Winskel 1996, Sct. 9] and *Functional Programming* course [Giesl]

Outline of Lecture 19

Recap: Correctness Properties for Execution Time

Soundness and Completeness

Outlook: Semantics of Functional Programming Languages

Outlook: Semantics of Logic Programming Languages

Miscellaneous

Outlook: Semantics of Logic Programming Languages

Syntax of Logic Programming Languages

- Program = list of predicate definitions
- Predicate definition = sequence of clauses of the form $q_0 \colon - q_1, \ldots, q_n$ with atoms p, q_i
- Atom = predicate call $p(t_1, \ldots, t_k)$ with predicate p and terms t_i over variables, constants and function symbols

Outlook: Semantics of Logic Programming Languages

Syntax of Logic Programming Languages

- Program = list of predicate definitions
- Predicate definition = sequence of clauses of the form $q_0 \colon - q_1, \ldots, q_n$ with atoms p, q_i
- Atom = predicate call $p(t_1, \ldots, t_k)$ with predicate p and terms t_i over variables, constants and function symbols

Example 19.3

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Outlook: Semantics of Logic Programming Languages

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  ($\Rightarrow$ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  (⇒ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

```
sibling(sally, erica).
```

Outlook: Semantics of Logic Programming Languages

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  ($\Rightarrow$ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

`sibling(sally, erica).`

$\Leftarrow$ `parent(Z, sally), parent(Z, erica).`

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  (⇒ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

```
sibling(sally, erica).
⇐ parent(Z, sally), parent(Z, erica).
⇐ mother(Z, sally), parent(Z, erica).
```


Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  ($\Rightarrow$ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

```
sibling(sally, erica).
⇐ parent(Z, sally), parent(Z, erica).
⇐ mother(Z, sally), parent(Z, erica).
⇐ parent(anna, erica).
```


Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  (⇒ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

```
sibling(sally, erica).
⇐ parent(Z, sally), parent(Z, erica).
⇐ mother(Z, sally), parent(Z, erica).
⇐ parent(anna, erica).
⇐ mother(anna, erica). ↯
```

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  (⇒ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

```
sibling(sally, erica).
⇐ parent(Z, sally), parent(Z, erica).
⇐ mother(Z, sally), parent(Z, erica).
⇐ parent(anna, erica).
⇐ father(anna, erica). ↯
```

Outlook: Semantics of Logic Programming Languages

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  (⇒ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

```
sibling(sally, erica).
⇐ parent(Z, sally), parent(Z, erica).
⇐ father(Z, sally), parent(Z, erica).
```

Outlook: Semantics of Logic Programming Languages

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  ($\Rightarrow$ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

```
sibling(sally, erica).
⇐ parent(Z, sally), parent(Z, erica).
⇐ father(Z, sally), parent(Z, erica).
⇐ parent(tom, erica).
```

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  ($\Rightarrow$ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

```
sibling(sally, erica).
⇐ parent(Z, sally), parent(Z, erica).
⇐ father(Z, sally), parent(Z, erica).
⇐ parent(tom, erica).
⇐ mother(tom, erica). ↯
```

Outlook: Semantics of Logic Programming Languages

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  ($\Rightarrow$ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

```
sibling(sally, erica).
⇐ parent(Z, sally), parent(Z, erica).
⇐ father(Z, sally), parent(Z, erica).
⇐ parent(tom, erica).
⇐ father(tom, erica).
```

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Try to find refutation proof of negated query
  ($\Rightarrow$ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 19.4

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Refutation proof:

`sibling(sally, erica).`
$\Leftarrow$ `parent(Z, sally), parent(Z, erica).`
$\Leftarrow$ `father(Z, sally), parent(Z, erica).`
$\Leftarrow$ `parent(tom, erica).`
$\Leftarrow$ `father(tom, erica).`
$\Leftarrow \Box$

Outlook: Semantics of Logic Programming Languages

Denotational Semantics of Logic Programming Languages

- meaning of program = {fully instantiated valid atoms}
- fixpoint iteration:
  - start with empty set
  - 1st step: all instantiations of facts (i.e., clauses with empty RHS)
  - $i + 1$st step: all instantiations of facts that can be derived from known facts
- monotonic and continuous w.r.t. set inclusion
- semantics := least fixpoint (Tarski/Knaster Theorem)
- coincides with operational semantics

Denotational Semantics of Logic Programming Languages

- meaning of program = {fully instantiated valid atoms}
- fixpoint iteration:
  - start with empty set
  - 1st step: all instantiations of facts (i.e., clauses with empty RHS)
  - $i + 1$st step: all instantiations of facts that can be derived from known facts
- monotonic and continuous w.r.t. set inclusion
- semantics := least fixpoint (Tarski/Knaster Theorem)
- coincides with operational semantics

Example 19.5

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Fixpoint iteration:
$A_0 = \emptyset$

Outlook: Semantics of Logic Programming Languages

Denotational Semantics of Logic Programming Languages

- meaning of program = {fully instantiated valid atoms}
- fixpoint iteration:
  - start with empty set
  - 1st step: all instantiations of facts (i.e., clauses with empty RHS)
  - $i + 1$st step: all instantiations of facts that can be derived from known facts
- monotonic and continuous w.r.t. set inclusion
- semantics := least fixpoint (Tarski/Knaster Theorem)
- coincides with operational semantics

Example 19.5

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Fixpoint iteration:

$A_0 = \emptyset$

$A_1 = \{\mathtt{f}(\mathtt{t}, \mathtt{s}), \mathtt{f}(\mathtt{t}, \mathtt{e}), \mathtt{f}(\mathtt{m}, \mathtt{t}), \mathtt{m}(\mathtt{a}, \mathtt{s})\}$

Denotational Semantics of Logic Programming Languages

- meaning of program = {fully instantiated valid atoms}
- fixpoint iteration:
  - start with empty set
  - 1st step: all instantiations of facts (i.e., clauses with empty RHS)
  - $i + 1$st step: all instantiations of facts that can be derived from known facts
- monotonic and continuous w.r.t. set inclusion
- semantics := least fixpoint (Tarski/Knaster Theorem)
- coincides with operational semantics

Example 19.5

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Fixpoint iteration:

$A_0 = \emptyset$

$A_1 = \{\mathtt{f(t,s)}, \mathtt{f(t,e)}, \mathtt{f(m,t)}, \mathtt{m(a,s)}\}$

$A_2 = A_1 \cup \{\mathtt{p(t,s)}, \mathtt{p(t,e)}, \mathtt{p(m,t)}, \mathtt{p(a,s)}\}$

Denotational Semantics of Logic Programming Languages

- meaning of program = {fully instantiated valid atoms}
- fixpoint iteration:
  - start with empty set
  - 1st step: all instantiations of facts (i.e., clauses with empty RHS)
  - $i+1$st step: all instantiations of facts that can be derived from known facts
- monotonic and continuous w.r.t. set inclusion
- semantics := least fixpoint (Tarski/Knaster Theorem)
- coincides with operational semantics

Example 19.5

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Fixpoint iteration:

$A_0 = \emptyset$

$A_1 = \{\mathtt{f}(\mathtt{t},\mathtt{s}), \mathtt{f}(\mathtt{t},\mathtt{e}), \mathtt{f}(\mathtt{m},\mathtt{t}), \mathtt{m}(\mathtt{a},\mathtt{s})\}$

$A_2 = A_1 \cup \{\mathtt{p}(\mathtt{t},\mathtt{s}), \mathtt{p}(\mathtt{t},\mathtt{e}), \mathtt{p}(\mathtt{m},\mathtt{t}), \mathtt{p}(\mathtt{a},\mathtt{s})\}$

$A_3 = A_2 \cup \{\mathtt{s}(\mathtt{s},\mathtt{e}), \mathtt{s}(\mathtt{e},\mathtt{s})\}$

Denotational Semantics of Logic Programming Languages

- meaning of program = {fully instantiated valid atoms}
- fixpoint iteration:
  - start with empty set
  - 1st step: all instantiations of facts (i.e., clauses with empty RHS)
  - $i + 1$st step: all instantiations of facts that can be derived from known facts
- monotonic and continuous w.r.t. set inclusion
- semantics := least fixpoint (Tarski/Knaster Theorem)
- coincides with operational semantics

Example 19.5

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Fixpoint iteration:

$A_0 = \emptyset$

$A_1 = \{\mathtt{f}(\mathtt{t},\mathtt{s}), \mathtt{f}(\mathtt{t},\mathtt{e}), \mathtt{f}(\mathtt{m},\mathtt{t}), \mathtt{m}(\mathtt{a},\mathtt{s})\}$

$A_2 = A_1 \cup \{\mathtt{p}(\mathtt{t},\mathtt{s}), \mathtt{p}(\mathtt{t},\mathtt{e}), \mathtt{p}(\mathtt{m},\mathtt{t}), \mathtt{p}(\mathtt{a},\mathtt{s})\}$

$A_3 = A_2 \cup \{\mathtt{s}(\mathtt{s},\mathtt{e}), \mathtt{s}(\mathtt{e},\mathtt{s})\}$

$A_4 = A_3$

Outlook: Semantics of Logic Programming Languages

Further Topics in Logic Programming Languages

- (Prolog) extensions: arithmetic, lists, cut, I/O, ...
- see *Logic Programming* course [Giesl]

Outline of Lecture 19

Recap: Correctness Properties for Execution Time

Soundness and Completeness

Outlook: Semantics of Functional Programming Languages

Outlook: Semantics of Logic Programming Languages

Miscellaneous

Miscellaneous

- Oral exams:
  - Thu 23 July
  - Wed 26 August
  - Thu 24 September

  Registration via foodle poll (cf. course web page)

Miscellaneous

- Oral exams:
  - Thu 23 July
  - Wed 26 August
  - Thu 24 September

  Registration via foodle poll (cf. course web page)
- Master-level teaching in Winter 2015/16:
  - Course *Modelling and Verification of Probabilistic Systems* [Katoen]
  - Course *Concurrency Theory* [Katoen/Noll]
  - Seminar *Trends in Computer-Aided Verification* [Katoen/Noll/NN]