Semantics and Verification of Software

Winter Semester 2017/18

Lecture 20: Wrap-Up

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

http://moves.rwth-aachen.de/teaching/ws-1718/sv-sw/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Outline of Lecture 20

Operational Semantics of Functional Programming Languages

- Program = list of function definitions

Operational Semantics of Functional Programming Languages

- 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 of Functional Programming Languages

- 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:

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

Denotational Semantics of Functional Programming Languages

- 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

Denotational Semantics of Functional Programming Languages

- 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, ...

Denotational Semantics of Functional Programming Languages

- 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 20

Syntax of Logic Programming Languages

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

Example 20.1

```
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).
```

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
  ($\Rightarrow$ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 20.2

```
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).
```

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 20.2

```
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).
```

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 20.2

```
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).
```

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 20.2

```
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).
```

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 20.2

```
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
  ($\Rightarrow$ instantiated query is logical consequence of program)
- Involves backtracking if several clause heads match

Example 20.2

```
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). ↯
```

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 20.2

```
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).
```

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 20.2

```
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 20.2

```
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). ↯
```

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 20.2

```
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 20.2

```
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).
⇐ □
```

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 20.3

```
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).
```

Fixpoint iteration:
$A_0 = \emptyset$

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 20.3

```
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).
```

Fixpoint iteration:

$A_0 = \emptyset$

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

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 20.3

```
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).
```

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 20.3

```
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).
```

Fixpoint iteration:

$A_0 = \emptyset$

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

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

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

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 20.3

```
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).
```

Fixpoint iteration:

$A_0 = \emptyset$

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

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

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

$A_4 = A_3$

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 20

Summary I

Semantics of programming languages

- Models the computational meaning of each program
- Provides abstract entities that represent just the relevant features of all possible executions
  - relationship between input and output
  - whether execution terminates or not
- Ignores details that have no relevance to the correctness of implementations
- Equivalences identify programs with identical semantics

Summary I

Semantics of programming languages

- Models the computational meaning of each program
- Provides abstract entities that represent just the relevant features of all possible executions
  - relationship between input and output
  - whether execution terminates or not
- Ignores details that have no relevance to the correctness of implementations
- Equivalences identify programs with identical semantics

(Structural) operational semantics

- Uses syntax-guided rules to specify transition relations (for evaluation/execution)
- Represents scoping of identifiers by splitting of data state into environment and store part
- Handles interaction between concurrent processes by labels on transitions; requires small-step semantics

Summary II

Denotational semantics

- Denotations are defined inductively on program structure
- Recursion (loops, procedures) handled as least fixpoints of continuous functions on CCPOs

Summary II

Denotational semantics

- Denotations are defined inductively on program structure
- Recursion (loops, procedures) handled as least fixpoints of continuous functions on CCPOs

Axiomatic semantics

- A Hoare Logic gives rules for the relation between assertions about values of variables before and after execution of each construct
- Employs logical variables for remembering (initial) values of program variables
- Partial vs. total correctness
- Recursion handled by invariants

Outline of Lecture 20

Oral Exams

- Oral exams:
  - Thu Feb 8 morning
  - Tue Feb 20 morning
  - Thu Mar 8 morning
  - Thu Mar 15 morning
  - Thu Mar 29 morning

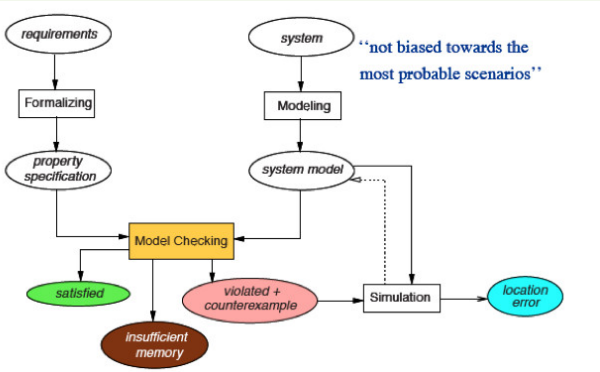
  Registration via foodle poll at

  https://terminplaner2.dfn.de/foodle/Oral-Exam-in-Semantics-Verification-5a5db

Miscellaneous

Master-Level Teaching in Summer 2018

Course *Introduction to Model Checking* [Katoen]



Course *Static Program Analysis* [Noll]

1. Dataflow analysis
2. Abstract interpretation
3. Interprocedural analysis
4. Analysis of heap data structures
5. Applications in optimising compilers and SW verification

Seminar *Formal Verification Meets Machine Learning* [Katoen, Noll, NN]

- Safety-related issues for machine learning
- Explainability in AI and in model checking
- Scalability and applicability of formal verification

Profillinie

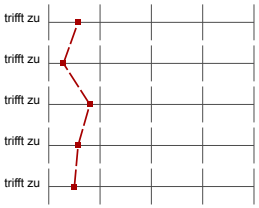
Teilbereich: Informatik
Name der/des Lehrenden: apl. Prof. Dr.rer.nat. Thomas Noll
Titel der Lehrveranstaltung: (Name der Umfrage) Semantik und Verifikation von Software (17ws-53304)

Verwendete Werte in der Profillinie: Mittelwert

Allgemein

1.6 Die Veranstaltung interessiert mich. n=8 mw=1,6 md=2,0 s=0,5

Konzept der Vorlesung



2.1 Die Lernziele der Vorlesung sind definiert. n=7 mw=1,6 md=1,0 s=0,8

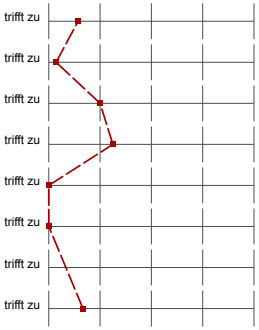
2.2 Die Vorlesung hat eine klar erkennbare Struktur. n=7 mw=1,3 md=1,0 s=0,5

2.3 Die zur Verfügung gestellten Materialien sind hilfreich. n=5 mw=1,8 md=2,0 s=0,8

2.4 Die ausgewählten Beispiele sind hilfreich. n=7 mw=1,6 md=1,0 s=0,8

2.5 Es werden Zusammenfassungen an sinnvollen Stellen gemacht. n=6 mw=1,5 md=1,5 s=0,5

Vermittlung und Verhalten



3.1 ... erklärt den Stoff verständlich. n=7 mw=1,6 md=2,0 s=0,5

3.2. ... geht auf Verständnisfragen ein. n=7 mw=1,1 md=1,0 s=0,4

3.3 ... berücksichtigt unterschiedliche Kenntnisstände der Studierenden. n=7 mw=2,0 md=2,0 s=1,2

3.4 ... schafft es, mich für den Vorlesungsstoff zu begeistern. n=8 mw=2,3 md=2,0 s=0,9

3.5 ... spricht angemessen laut und deutlich. n=7 mw=1,0 md=1,0 s=0,0

3.6 ... ist gut vorbereitet. n=7 mw=1,0 md=1,0 s=0,0

3.7 ... ist außerhalb der Vorlesung ansprechbar. (*)

3.8 ... setzt Medien ein, die zum Verständnis beitragen. n=6 mw=1,7 md=2,0 s=0,5

Rahmenbedingungen

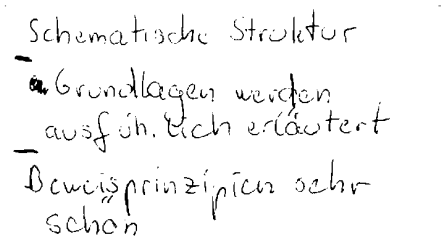
4.1 Der zeitliche Rahmen der Vorlesung wird eingehalten. n=7 mw=1,3 md=1,0 s=0,5

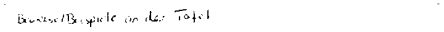
(*) Hinweis: Wenn die Anzahl der Antworten auf eine Frage zu gering ist, wird für die Frage keine Auswertung angezeigt.

Auswertungsteil der offenen Fragen

Besondere Anregungen / Kritik / Wünsche:

5.1 Was hat Ihnen an der Vorlesung besonders **gut** gefallen?





The professor takes his time to explain everything and has the time to answer questions

5.2 Was hat Ihnen an der Vorlesung **nicht** gefallen?

Symboltabelle / Symbole
manchmal inkonsistent
wlp, sp - Definitionen vor der Benutzung / zum Verständnis sinnvoller
bei zu vielen Beweisen unklar, ob diese relevant (in VL) sind.
white mit end updaten ?!
chain & Charakterisierung fix
w.r.t. chain anfügen
Bilder &

1

Maybe speed up tedious proofs using
a proof assistant like Coq a Isabelle

Sometimes lecture notes
instead of slides to understa
the lectures better would
be nice