FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Formal Semantics of Programming Languages

Florian Zuleger

SS 2014

Programming Languages

- **Syntax:** what sequences of symbols are valid programs? Grammars, lexical analyzers, parsers, …
- **Pragmatics:** Descriptions and examples of how features of the language should be used. Design issues. Implementation issues. …
- **Semantics:** The meaning of language's features; e.g. what should happen when a program is executed. When are two program fragments equivalent? When does a program satisfy a specification?

Formal Methods

The application of **maths** and **logic** to Computer Science:

- **Design of programming languages**: describe exact behavior of all programs from the language
- **Mathematical models of system behavior**: predict system behavior using mathematical reasoning
- **Program correctness:**
  - Tradition: use extensive test suites
  - The future: augment test suites with formal logic

Informal descriptions

Here is the informal definition of

```
while B do C
```

adapted from B.W. Kerrigan and D.M. Ritchie, *The C Programming Language*, Prentice-Hall, 1976, pp. 202:

The command C is executed repeatedly so long as the value of the expression B remains true. The test takes place before each execution of the command.

Informal descriptions

An extract from the Algol 60 report:

Finally the procedure body, modified as above, is inserted in place of the procedure statement and executed. If a procedure is called from a place outside the scope of any non-local quantity of the procedure body, the conflicts between the identifiers inserted through this process of body replacement and the identifiers whose declarations are valid at the place of the procedure statement or function designator will be avoided through suitable systematic changes of the latter identifiers.

What is it saying???

Designing a new language (http://esolangs.org/wiki/Brain**)

Instructions in the Brain** language given by the grammar:

I ::= '>' | '<' | '+' | '-' | '.' | ',' | '[' | ']' | $I_1\ I_2$

>	Move the pointer to the right
<	Move the pointer to the left
+	Increment the memory cell under the pointer
-	Decrement the memory cell under the pointer
.	Output the character signified by the cell at the pointer
,	Input a character and store it in the cell at the pointer
[	Jump past the matching] if the cell under pointer is 0
]	Jump back to the matching [if the cell under the pointer is nonzero
$I_1\ I_2$	Sequencing of Commands

What does programs written in Brain** do?

Benefits of Formal, Mathematical Semantics

- **Implementation**: machine-independent specification of behavior. Correctness of compilers, including optimizations, static analyses, …
- **Verification**: semantics supports reasoning about programs, specifications and other properties, both mechanically and by hand.
- **Language design**: subtle ambiguities in existing languages often come to light, and cleaner, clearer ways of organizing things can be discovered.

Compilation

Most compilers translate “higher-level” languages into “lower-level” languages, e.g., Java-code is translated into Java-bytecode

E.g.   z = a + b

is translated into

              push a, push b, add, store z

When is this translation correct?

Code Optimization

- Question: Can we replace code fragment C be code fragment d?

```
int add1(int x, int y)
{ return (x + y);
}
```

```
int add2(int x, int y)
{ return (y + x);
}
```

```
int add3(int [] a, int i)
{ return (a[i++] + a[i]);
}
```

```
int add4(int [] a, int i)
{ return (a[i] + a[i++]);
}
```

Code Optimization

- Question: Can we replace code fragment C be code fragment d?

```
if(a && b)
{  …
}
```

```
if (l != null && l.next  != null)
{  …
}
```

```
if(b && a)
{ …
}
```

```
if (l.next  != null && l != null)
{  …
}
```

Code Optimization

- Question: Can we replace code fragment C be code fragment d?

```
*p = 1;
*q = 2;
*p = *p + 3;
```

Can p and q point to the same location, i.e., $p = q$?

```
*q = 2;
*p = 1 + 3;
```

Corret only for $p \neq q$!

Concurrency

Two memory locations x and y initially 0.

Thread 1	Thread 2
x = 1;	y = 1;
print y;	print x;

What are the possible values for x and y when they are read?

Expected answer: (1,1), (1,0), (0,1) assuming sequential consistency (≈ statements are executed atomically in the order the appear in the threads)

We observe (0,0) every 630/100000 runs (on an Intel Core Duo x86)

Semantics: ???

Course Aims

To introduce students to formal methods for specifying the semantics of programming languages and formal techniques for verifying the behavior of programs.

Learning Outcomes:

- Specify the semantics for simple languages.
- **Reason formally** about the behavior of programs.

Lecture Dates

1.4.	lecture
3.4.	lecture/ex1 online
8.4.	exercise/ repetition
10.4.	lecture/hand-in ex1/ex2 online
29.4.	lecture
6.5.	exercise/ex1 discussed
8.5.	lecture/hand-in ex2/ex3 online
13.5.	lecture
15.5.	exercise/ex2 discussed
20.5.	lecture/hand-in ex3/ex4 online
22.5.	lecture
27.5.	exercise/ex3 discussed
3.6.	lecture/hand-in ex4/ex5 online
5.6.	lecture
13.6.	exercise/ex4 discussed
17.6.	lecture/hand-in ex5
24.6.	exercise/ex5 discussed
27.6.	exam

Homework

- There will be 5 exercise sheets.
- You can obtain 20 points for each exercise sheet.
- Your solutions will be corrected and returned to you.
- Each exercise sheet is discussed in the following exercise.
- You have to obtain 40% of the available points to be admitted to the exam (i.e., you have to obtain 40 points out of 100 total points).

Exam

- Written exam.
- On the last lecture date, i.e., 27.6., 12:00-14:00. **Note the changed room and time!**
- For admission you need 40 points of the exercises.
- The points of the exercises **do not** count for the final grade. The final grade depends solely on the final exam.
- Exam questions will be similar in style and level of difficulty as the questions on the exercise sheets.
- A second exam will be written or oral depending on the number of students who register. The date will be announced.

Literature

- Script by Matthew Hennessy, https://www.cs.tcd.ie/Matthew.Hennessy/teaching/2013/slexternal2013/resources/Guy.pdf, https://www.cs.tcd.ie/Matthew.Hennessy/teaching/2013/slexternal2013/lectures.php
- Book by Nielson and Nielson, *SEMANTICS WITH APPLICATIONS – a formal introduction*, John Wiley & Sons, 1999, available online at http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html
- Book by Glynn Winskel, *The Formal Semantics of Programming Languages*, MIT Press, 1993
- More literature to be announced..

Styles of Semantics

- **Operational**: a program’s meaning is given in terms of the steps of computation the program makes when you run it.
- **Denotational**: a program’s meaning is given abstractly as a mathematical function.
- **Axiomatic**: a program’s meaning is given indirectly in terms of the collection of properties it satisfies; these properties are defined via a collection of axioms and rules.

State of the Art

- **Operational**: most sequential languages and some concurrent languages can be given operational semantics; e.g. used by Xavier Leroy to prove the correctness of a compiler in Coq.
- **Denotational**: useful for proving the correctness of static analyses / program analyses; related to abstract interpretation, compare the work by Patrick Cousot.
- **Axiomatic**: weapon of choice in verification (Hoare Logic); has for example been used in proving the correctness of a simple garbage collector and a simple operating system inside Microsoft Research.

Arithmetic Expressions (*Exp*)

Syntax given by grammar in BNF:

$$E ::= n \mid E_1 + E_2 \mid E_1 * E_2 \mid \ldots$$

where

- n ranges over the numerals 0, 1, …
- +, *, … are symbols for operations

We will always work with **abstract syntax**.

Useful notions: abstract syntax tree (AST), well-bracketed expressions, arithmetic conventions, …

Alternative: Inference Rules

$$\text{NUM}\ \frac{}{n} \qquad \text{n is a numeral}$$

$$\text{ADD}\ \frac{E_1 \qquad E_2}{E_1 + E_2} \qquad \text{MUL}\ \frac{E_1 \qquad E_2}{E_1 * E_2} \qquad \ldots$$

Exp is the set of expressions that can be derived by these rules.

Anatomy of an Inference Rule

Finitely many hypotheses!

$$\text{NAME}\ \frac{\text{hypothesis}_1 \quad \ldots \quad \text{hypothesis}_n}{\text{conclusion}}\ \text{side condition}$$

Big-Step Semantics of *Exp*

Judgements:

$E \Downarrow n$

Meaning:

Evaluating expression E results in **number** n.

Big-Step Semantics of *Exp*

$$\text{B-NUM}\ \frac{}{\mathbf{n} \Downarrow \mathrm{n}} \quad \mathrm{n} = \text{number}(\mathbf{n})$$

$$\text{B-ADD}\ \frac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{E_1 + E_2 \Downarrow n_3} \quad n_3 = n_1 + n_2$$

+ is the arithmetic function that takes two **numbers** (not **numerals**!) and returns their sum

Similar rules for *, …

How to Read Axioms

The axiom

$$\text{B-NUM}\quad \frac{}{\mathbf{n} \Downarrow n} \quad n = \text{number}(\mathbf{n})$$

says:

for every numeral $\mathbf{n}$,
the numeral $\mathbf{n}$ evaluates to the number n.

In (B-NUM) n is a kind of variable: you can put any numeral you like in its place. These are called *meta-variables*.

Numbers and Numerals

$$\text{B-NUM} \quad \frac{}{\mathbf{n} \Downarrow n} \quad n = \text{number}(\mathbf{n})$$

Why do we make a difference between numbers and numerals?

E.g. 100 can be a number in base two or base eight, but is looks as number in base ten, too. Thus we can interpret 100 in different ways!

Because of this difficulty we assume a function that converts numerals into numbers.

How to Read Rules

The inference rule

$$\text{B-ADD}\ \frac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{E_1 + E_2 \Downarrow n_3} \quad n_3 = n_1 + n_2$$

says:

for any expressions E_1 and E_2,

if it is the case that $E_1 \Downarrow n_1$

and if it is the case that $E_2 \Downarrow n_2$

then it is the case that $E_1 + E_2 \Downarrow n_3$

where n_3 is the number such that $n_3 = n_1 + n_2$

In (B-NUM) E_1, E_2, n_1 , n_2, n_3 are *meta-variables*.

Rules are Schemas

Because the E’s and n’s in these rules are metavariables each rule is really a **schema** (pattern) for an infinite collection of rules. Some of these **instances** are a bit silly, for example:

$$\text{B-ADD}\ \frac{3 \Downarrow 4 \qquad 4 \Downarrow 5}{(3 + 4) \Downarrow 9}$$

This rule is **valid**, but is *useless*, because it is not the case that $3 \Downarrow 4$. That is to say, the hypotheses of the rule are never satisfied.

Example

A proof that $(3 + (2 + 1)) \Downarrow 6$

$$\text{B-ADD}\ \dfrac{\text{B-NUM}\ \dfrac{}{3 \Downarrow 3} \qquad \text{B-ADD}\ \dfrac{\text{B-NUM}\ \dfrac{}{2 \Downarrow 2} \qquad \text{B-NUM}\ \dfrac{}{1 \Downarrow 1}}{(2+1) \Downarrow 3}}{(3 + (2+1)) \Downarrow 6}$$

Notation: $\vdash_{\text{big}} (3 + (2 + 1)) \Downarrow 6$

Small-step semantics of *Exp*

<u>Judgements:</u>

$E_1 \to E_2$

<u>Meaning:</u>

After performing one step of evaluation of E_1 the expression E_2 remains to be evaluated.

Small-step semantics of *Exp*

$$\text{S-LEFT}\ \frac{E_1 \rightarrow E_1'}{E_1 + E_2 \rightarrow E_1' + E_2}$$

$$\text{S-RIGHT}\ \frac{E_2 \rightarrow E_2'}{n + E_2 \rightarrow n + E_2'}$$

$$\text{S-ADD}\ \frac{}{n_1 + n_2 \rightarrow n_3} \quad \begin{array}{l} \text{number}(\mathbf{n_3}) = \\ \text{number}(\mathbf{n_1}) + \text{number}(\mathbf{n_2}) \end{array}$$

Similar rules for *, …

Small-step semantics of *Exp*

$$\text{S-LEFT}\quad \frac{E_1 \rightarrow E_1'}{E_1 + E_2 \rightarrow E_1' + E_2}$$

$$\text{S-RIGHT}\quad \frac{E_2 \rightarrow E_2'}{n + E_2 \rightarrow n + E_2'}$$

Left-to-right evaluation!

$$\text{S-ADD}\quad \frac{}{n_1 + n_2 \rightarrow n_3}\quad \text{number}(\mathbf{n_3}) = \text{number}(\mathbf{n_1}) + \text{number}(\mathbf{n_2})$$

Similar rules for *, …

Examples

A derivation:

$$\dfrac{\text{S-ADD}\ \dfrac{}{3 + 7 \to 10}}{(3 + 7) + (8 + 1) \to 10 + (8 + 1)}\ \text{S-LEFT}$$

Another derivation:

$$\dfrac{\text{S-ADD}\ \dfrac{}{8 + 1 \to 9}}{10 + (8 + 1) \to 10 + 9}\ \text{S-RIGHT}$$

Notation: $\vdash_{sm} (3 + 7) + (8 + 1) \to 10 + (8 + 1)$

$\vdash_{sm} 10 + (8 + 1) \to 10 + 9$

Choice semantics of *Exp*

$$\text{S-LEFT}\ \frac{E_1 \rightarrow_{ch} E_1'}{E_1 + E_2 \rightarrow_{ch} E_1' + E_2}$$

$$\text{S-RIGHT}\ \frac{E_2 \rightarrow_{ch} E_2'}{E_1 + E_2 \rightarrow_{ch} E_1 + E_2'}$$

$$\text{S-ADD}\ \frac{}{n_1 + n_2 \rightarrow_{ch} n_3}\quad \begin{array}{l}\text{number}(\mathbf{n_3}) = \\ \text{number}(\mathbf{n_1}) + \text{number}(\mathbf{n_2})\end{array}$$

Examples

A derivation:

$$\dfrac{\dfrac{}{3 + 7 \rightarrow_{ch} 10}\ \text{S-ADD}}{(3 + 7) + (8 + 1) \rightarrow_{ch} (3 + 7) + 9}\ \text{S-LEFT}$$

Notation: $\vdash_{ch}\ (3 + 7) + (8 + 1) \rightarrow_{ch} 10 + (8 + 1)$

$\vdash_{ch}\ (3 + 7) + (8 + 1) \rightarrow_{ch} (3 + 7) + 9$

True or False?: $\vdash_{ch}\ E_1 \rightarrow_{ch} E_2$ implies $\vdash_{sm}\ E_1 \rightarrow E_2$

$\vdash_{sm}\ E_1 \rightarrow E_2$ implies $\vdash_{ch}\ E_1 \rightarrow_{ch} E_2$

Recap: Reflexive Transitive Closure

Given a relation $\rightarrow$ we define its **reflexive transitive closure** $\rightarrow^*$ as follows:

E $\rightarrow^*$ E' iff either

- E = E' (no steps of the relation are needed to get from E to E') or
- there is a finite sequence of relation steps $E \rightarrow E_1 \rightarrow E_2 \rightarrow \ldots \rightarrow E_k \rightarrow E'$.

Executing Small-step Semantics

We say E **evaluates to** n, if E $\rightarrow$* n.

Questions:

- E $\Downarrow$ n and E $\Downarrow$ m implies m=n (determinism)?
- Is there always an n s.t. E $\rightarrow$* n (termination)?
- E $\rightarrow$* n iff E $\Downarrow$ n (equivalence of semantics)?
- E $\rightarrow$* n iff E $\rightarrow_{ch}$* n (determinism of choice)?

Denotational Semantics

We will define the **denotational semantics** of expressions via a function:

$$[\![-]\!] : \mathit{Exp} \to \mathbb{N}.$$

Denotation of Expressions

Here is the definition of our semantic function. This is an example of definition by **structural induction**.

$$[\![\mathbf{n}]\!] = n, \qquad \text{using } n = \text{number}(\mathbf{n})$$

$$[\![E_1 + E_2]\!] = [\![E_1]\!] + [\![E_2]\!]$$

Remember the distinction between numbers and numerals. We only make this distinction on this slide.

Calculating Semantics

$$\begin{aligned}
[\![(1 + (2 + 3))]\!] &= [\![1]\!] + [\![(2 + 3)]\!] \\
&= 1 + [\![(2 + 3)]\!] \\
&= 1 + ([\![2]\!] + [\![3]\!]) \\
&= 1 + (2 + 3) \\
&= 1 + 5 \\
&= 6
\end{aligned}$$

Associativity of Addition

Theorem $[\![E_1 + (E_2 + E_3)]\!] = [\![(E_1 + E_2) + E_3]\!]$

Proof
$$\begin{aligned}
[\![E_1 + (E_2 + E_3)]\!] &= [\![E_1]\!] + [\![(E_2 + E_3)]\!] \\
&= [\![E_1]\!] + ([\![E_2]\!] + [\![E_3]\!]) \\
&= ([\![E_1]\!] + [\![E_2]\!]) + [\![E_3]\!] \\
&= ([\![E_1 + E_2]\!]) + [\![E_3]\!] \\
&= [\![(E_1 + E_2) + E_3]\!]
\end{aligned}$$

Exercise: Show a similar fact using the operational semantics (more cumbersome!).

Contextual Equivalence (Informal)

We shall now introduce a very important idea in semantics, that of **contextual equivalence**.

Intuition: Equivalent programs can be used interchangeably; if $P_1 \cong P_2$ and P_1 is used in some context, $C[P_1]$, then we should get the same effect if we replace P_1 with P_2 : we expect $C[P1] \cong C[P2]$.

To make this more precise, we say that a context $C[-]$ is a program with a hole where you would ordinarily expect to see a sub-program.

Some Exp contexts

$C_1[-] = -.$

$C_2[-] = (- + 2).$

$C_3[-] = ((- + 1) + -).$

Grammar for expression contexts

$$C ::= - \mid n \mid C_1 + C_2 \mid C_1 * C_2 \mid \ldots$$

Filling the holes

C_1[-] = -.

C_2[-] = (- + 2).

C_3[-] = ((- + 1) + -).

C_1[(3+4)] = (3+4).

C_2[(3+4)] = ((3+4) + 2).

C_3[(3+4)] = (((3+4) + 1) + (3+4)).

Contextual Equivalence

Definition

Expressions E_1 and E_2 are **contextually equivalent with respect to the big-step semantics**, if for all contexts C[−] and all numerals n,

$$C[E_1] \Downarrow n \quad \text{iff} \quad C[E_2] \Downarrow n.$$

Compositionality and Contextual Equivalence

Recall: the denotational semantics is compositional, i.e., the meaning of an expression phrase is built out of the meanings of its sub-expressions.

Thus, each context determines a “function between meanings”, i.e., for each C[−] there is a function $f_C \colon \mathbb{N} \to \mathbb{N}$ such that

$$[\![C[E]]\!] = f_C([\![E]\!])$$

for any expression E.

Compositionality and Contextual Equivalence

A consequence of such a function $f_C \colon \mathbb{N} \to \mathbb{N}$ for a context $C[-]$ is that

$$\text{if } [\![E_1]\!] = [\![E_2]\!] \text{ then}$$
$$[\![C[E_1]]\!] = f_C([\![E_1]\!]) = f_C([\![E_2]\!]) = [\![C[E_2]]\!]!$$

It remains to relate the structural and the denotational semantics, i.e., we want to establish that $[\![E]\!] = n$ iff $E \Downarrow n$ for all expressions E!

Equivalence Theorem

Theorem For all expressions E, $[\![E]\!] = n \Leftrightarrow E \Downarrow n$.

Proof

We split the proof into the two lemmas for the cases

$$[\![E]\!] = n \Leftarrow E \Downarrow n$$

and

$$[\![E]\!] = n \Rightarrow E \Downarrow n.$$

Equivalence Theorem

Lemma For all expressions E, $[\![E]\!] = n \Leftarrow E \Downarrow n$.

Proof By structural induction on the expression E.

Base Case: E is a numeral n. We have $E \Downarrow n$.
This can only be derived from the rule (B-NUM).

$$\text{B-NUM}\ \frac{}{n \Downarrow n}$$

By the definition of the denotational semantics we have $[\![E]\!] = n$, so the base case is solved.

Equivalence Theorem

Induction Step: Suppose E is of the form $(E_1 + E_2)$. We have $(E_1 + E_2) \Downarrow n$.

This can only be derived from the rule (B-ADD) using hypotheses $E_1 \Downarrow n_1$ and $E_2 \Downarrow n_2$ for some numbers n_1, n_2 with $n_3 = n_1+n_2$.

$$\text{B-ADD}\ \frac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{E_1 + E_2 \Downarrow n} \quad n = n_1 + n_2$$

By the inductive hypothesis, $[\![E_i]\!] = n_i$ for i = 1, 2, and therefore $[\![E]\!] = [\![E_1]\!] + [\![E_2]\!] = n_1 + n_2 = n$.

Equivalence Theorem

Lemma For all expressions E, $[\![E]\!] = n \Rightarrow E \Downarrow n$.

Proof By structural induction on the expression E.

<u>Base Case:</u> E is a numeral n. We have $[\![E]\!] = n$.
By the rule (B-NUM) we can derive $n \Downarrow n$.

$$\text{B-NUM}\ \frac{}{n \Downarrow n}$$

This solves the base case.

Equivalence Theorem

<u>Induction Step:</u> Suppose E is of the form $(E_1 + E_2)$. We have $[\![E_1 + E_2]\!] = n$.

By the definition of the denotational semantics we have $n = [\![E_1 + E_2]\!] = [\![E_1]\!] + [\![E_2]\!] = n_1 + n_2$, where n_i is the number such that $[\![E_i]\!] = n_i$.

From the induction hypothesis we get $E_i \Downarrow n_i$ for i=1,2.

Thus we derive $(E_1 + E_2) \Downarrow n$ from rule (B-ADD).

$$\text{B-ADD}\ \frac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{E_1 + E_2 \Downarrow n} \quad n = n_1 + n_2$$