Introduction to Semantics

Course 02240
spring 2005

Hanne Riis Nielson
riis@imm.dtu.dk
Informatics and Mathematical Modelling

Syntax of While

Syntactic categories

- Numerals — $n \in$ Num
- Variables — $x \in$ Var

not further specified

- Arithmetic expressions — $a \in$ AExp

$$a \;::=\; n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2$$

- Boolean expressions — $b \in$ BExp

$$b \;::=\; \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2$$

- Statements — $S \in$ Stm

$$\begin{aligned} S \;::=\;& x := a \mid \texttt{skip} \mid S_1 ; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \\ \mid\;& \texttt{while } b \texttt{ do } S \mid \texttt{repeat } S \texttt{ until } b \end{aligned}$$

Abstract vs Concrete syntax

- **Abstract Syntax**

  focusses on the *structure* of expressions, statements, etc and ignores the scanning and parsing aspects of the syntax

$$
\begin{aligned}
a ::= {} & n \mid x \\
& \mid a_1 + a_2 \\
& \mid a_1 * a_2
\end{aligned}
$$

- **Concrete Syntax**

  deals with scanning and parsing aspects

A ::= T+ A | T
T ::= F * T | F
F ::= N | X | (A)

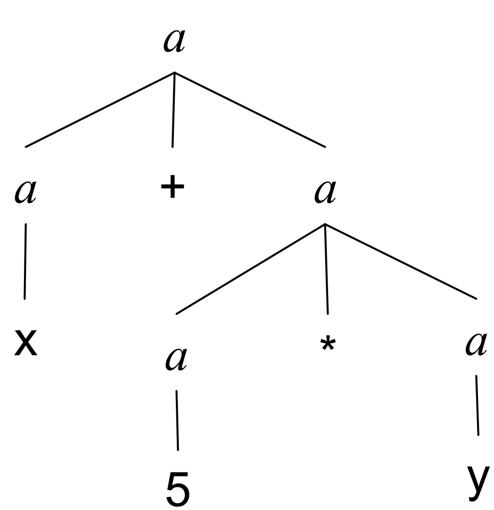
N: digit$^+$
X: letter (digit | letter)*

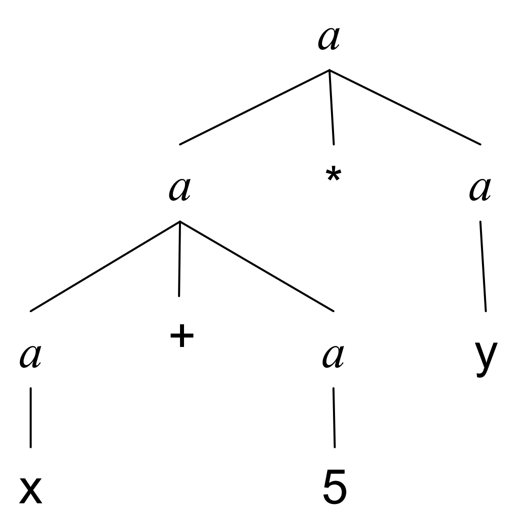
Example: x+5*y

$$a ::= n \mid x \mid a_1 + a_2 \mid a_1 * a_2$$

- **Abstract syntax:**
  - formalises the allowable parse trees
  - we use parentheses to disambiguate the syntax
  - we introduce defaults as e.g. * binds closer than +

default
x + (5 * y)



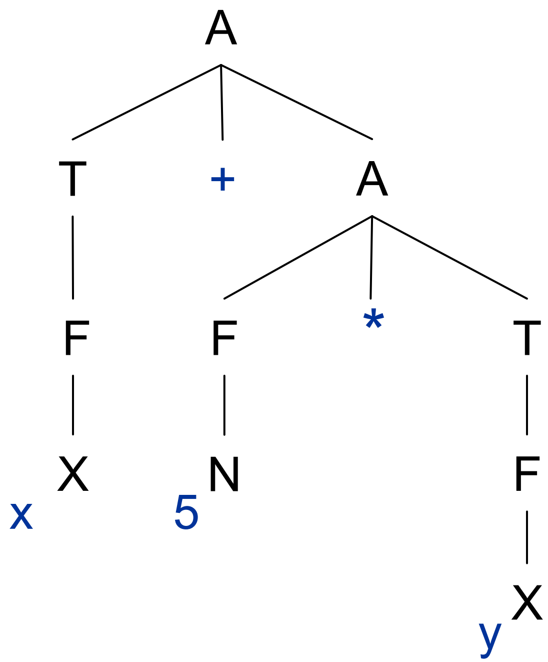


write
(x + 5) * y

Example: x+5*y

- Concrete syntax
  - parantheses disambiguate the syntax
  - the grammar captures aspects like the precedence and associativity rules

A ::= T+ A | T
T ::= F * T | F
F ::= N | X | (A)



Other ambiguities

$$
\begin{array}{lll}
S & ::= & x := a \mid \texttt{skip} \mid S_1\,;\,S_2 \mid \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \\
 & \mid & \texttt{while}\ b\ \texttt{do}\ S
\end{array}
$$

x := 1; y := 2; z := 3

x := 1; (y := 2; z := 3)

(x := 1; y := 2); z := 3

if x < y then x := 1; y := 2 else x := 3; y := 4

if x < y then (x := 1; y := 2) else x := 3; y := 4

if x < y then x := 1; y := 2 else (x := 3; y := 4)

while x < y do x := x+1; y := 0

while x < y do (x := x+1; y := 0)

Example programs

- factorial program:
  - if x = n initially then y= $n!$ when the program terminates
  - y := 1; while ¬(x=1) do (y:=y*x; x:=x-1)

- power function:

Exercise 1.2

  - if x = n and y = m initially then z = n^m when the program terminates
  - write the program in the while language

Semantics of expressions

Memory model: states

- the value of x+5*y depends on the values of the variables x and y
- these are determined by the current state
- operations on states:

lookup in a state: $s\ x$

update a state: $s' = s[y \mapsto n]$

$$s'\ x = \begin{cases} s\ x & \text{if } x \neq y \\ n & \text{if } x = y \end{cases}$$

variables

semantic values: numbers

x	2
y	4
z	0

the value of x+5*y is 22:

$\mathcal{A}$ [x+5*y]s = s(x)+5*s(y)
= 2+5*4
= 22

Semantic functions

- $\mathcal{A}$: AExp $\to$ (State $\to$ Z)

  for each arithmetic expression a and each state s the function determines the value (a number) $\mathcal{A}[a]s$ of a

- $\mathcal{B}$: BExp $\to$ (State $\to$ T)

  for each boolean expression b and each state s the function determines the value (true or false) $\mathcal{B}[b]s$ of b

$\mathcal{A}$: AExp → (State → Z)

one clause for each of the different forms of arithmetic expressions

$\mathcal{N}$: Num → Z
from numerals (syntax) to numbers (semantics)

$$
\left\{
\begin{array}{lcl}
\mathcal{A}[\![n]\!]s & = & \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s & = & s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s & = & \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s & = & \mathcal{A}[\![a_1]\!]s \star \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s & = & \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{array}
\right.
$$

symbols of syntax

semantic operators

$\mathcal{B}$: BExp → (State → T)

truth values:
tt (for true)
ff (for false)

$$
\begin{aligned}
\mathcal{B}[\![\texttt{true}]\!]s &= \mathbf{tt} \\
\mathcal{B}[\![\texttt{false}]\!]s &= \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s \neq \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s \leq \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg\, b]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]s = \mathbf{ff} \end{cases}
\end{aligned}
$$

one clause for each of the different forms of boolean expressions

The rules of the game

- the syntactic category is specified by giving
  - the basic elements
  - the composite elements; these have a unique decomposition into their immediate constituents

$$a \quad ::= \quad \boxed{n \mid x} \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2$$

$$b \quad ::= \quad \boxed{\texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2} \mid \neg b \mid b_1 \wedge b_2$$

The rules of the game

- The semantics is then defined by a compositional definition of a function:
  - there is a semantic clause for each of the basic elements of the syntactic category
  - there is a semantic clause for each of the ways for constructing composite elements; the clause is defined in terms of the semantics for the immediate constituents of the elements

$$\left.\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\ x
\end{aligned}\right\}\ \text{basic elements}$$

$$\left.\begin{aligned}
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \star \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{aligned}\right\}\ \text{composite elements}$$

A simple result

- We want to formalise the fact that the value of an arithmetic expression only depends on the values of the variables occurring in it
- **Definition:** FV(*a*) is the set of free variables in the arithmetic expression *a*

$$
\begin{array}{lcl}
\mathrm{FV}(n) & = & \emptyset \\
\mathrm{FV}(x) & = & \{\, x \,\} \\
\mathrm{FV}(a_1 + a_2) & = & \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(a_1 \star a_2) & = & \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(a_1 - a_2) & = & \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2)
\end{array}
$$

A simple result and its proof

- **Lemma**: Assume that s and s' are states satisfying $s(x) = s'(x)$ for all x in FV(a). Then $\mathcal{A}[a]s = \mathcal{A}[a]s'$.

- **Proof**: by Structural Induction
  - case n
  - case x
  - case $a_1 + a_2$
  - case $a_1 * a_2$
  - case $a_1 - a_2$

Structural Induction

To prove a property of all the elements of the syntactic category do the following:

- Prove that the property holds for all the basis elements of the syntactic category.
- Prove that the property holds for all the composite elements of the syntactic category: Assume that the property holds for all the immediate constituents of the element — this is called the induction hypothesis — and prove that it also holds for the element itself.

A substitution result

- We want to formalise the fact that a substitution within an expression can be mimicked by a similar change of the state.
- **Definition**: Replacing all occurrences of *y* within *a* with a_0:

$$
\begin{array}{lcl}
n[y \mapsto a_0] & = & n \\
x[y \mapsto a_0] & = & \begin{cases} a_0 & \text{if } x = y \\ x & \text{if } x \neq y \end{cases} \\
(a_1 + a_2)[y \mapsto a_0] & = & (a_1[y \mapsto a_0]) + (a_2[y \mapsto a_0]) \\
(a_1 \star a_2)[y \mapsto a_0] & = & (a_1[y \mapsto a_0]) \star (a_2[y \mapsto a_0]) \\
(a_1 - a_2)[y \mapsto a_0] & = & (a_1[y \mapsto a_0]) - (a_2[y \mapsto a_0])
\end{array}
$$

A substitution result and its proof

- **Lemma**: Let

$$(s[y \mapsto v])\ x = \begin{cases} v & \text{if } x = y \\ s\ x & \text{if } x \neq y \end{cases}$$

- then for all states *s*

$$\mathcal{A}[\![a[y \mapsto a_0]]\!]s = \mathcal{A}[\![a]\!](s[y \mapsto \mathcal{A}[\![a_0]\!]s])$$

- **Proof**: Exercise 1.13

Semantics of statements

Updating the states

- An assignment updates the state

state before executing z := x+5*y

x	2
y	4
z	0

- general formulation:

$$\langle x := a,\ s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$$

state before executing $x := a$

state after executing $x := a$

state after executing z := x+5*y

x	2
y	4
z	22

Two kinds of semantics

- Natural semantics (NS)
  - given a statement and a state in which it has to be executed, what is the resulting state (if it exists)?
- Structural operational semantics (SOS)
  - given a statement and a state in which it has to be executed, what is the next step of the computation (if it exists)?

Natural semantics

- the result of executing the assignment *x* := *a* in the state *s* is the state *s* updated such that *x* gets the value of *a*

- the result of executing the skip statement in the state *s* is simply the state *s*

$$\langle x := a,\ s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\langle \texttt{skip},\ s\rangle \to s$$

axiom schemas
they can be instantiated for particular choices of *x*, *a* and *s*

Natural semantics

- the result of executing S_1; S_2 from the state s is obtained by first executing the S_1 from s to obtain its resulting state s' and then to execute S_2 from that state to obtain its resulting state s'' and that will be the resulting state for S_1; S_2

$$\frac{\langle S_1,\, s\rangle \to s',\ \langle S_2,\, s'\rangle \to s''}{\langle S_1;S_2,\, s\rangle \to s''}$$

a rule with
- two premises

and
- one conclusion

Building a derivation tree

axiom schemas

$$\langle \texttt{skip}, s\rangle \to s \qquad \langle x := a, s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$$

assume x is 0 in s_0

instances

$$\langle \texttt{skip}, s_0\rangle \to s_0 \qquad \langle \texttt{x} := \texttt{x}+1, s_0\rangle \to s_0[\texttt{x} \mapsto \mathbf{1}]$$

the state that is as s_0 except that x is 1

$$\frac{\langle \texttt{skip}, s_0\rangle \to s_0,\ \langle \texttt{x} := \texttt{x}+1, s_0\rangle \to s_0[\texttt{x} \mapsto \mathbf{1}]}{\langle \texttt{skip};\ \texttt{x} := \texttt{x}+1, s_0\rangle \to s_0[\texttt{x} \mapsto \mathbf{1}]}$$

instance of rule: the premises are satisfied

$$\frac{\langle S_1, s\rangle \to s',\ \langle S_2, s'\rangle \to s''}{\langle S_1;S_2, s\rangle \to s''}$$

Natural semantics

- The result of executing if b then S_1 else S_2 from state s depends on the value of b in state s:
  - If it is **tt** then the result is the resulting state of S_1
  - If it is **ff** then the result is the resulting state of S_2

$$\frac{\langle S_1, s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, s\rangle \rightarrow s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$\frac{\langle S_2, s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, s\rangle \rightarrow s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

side conditions

must be computable

Natural semantics

- The result of executing while *b* do *S* from state *s* depends on the value of *b* in state *s*:
  - If it is **tt** then we first execute *S* from *s* to obtain its resulting state *s*' and then repeat the execution of while *b* do *S* but from *s*' in order to obtain its resulting state *s*'' which then will be the overall resulting state
  - If it is **ff** then the resulting state is simply *s*

$$\frac{\langle S, s\rangle \to s', \langle \texttt{while}\ b\ \texttt{do}\ S, s'\rangle \to s''}{\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \to s''} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \to s \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

side conditions

Summary: natural semantics

$$\langle x := a\,,\, s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\langle \texttt{skip}\,,\, s\rangle \to s$$

$$\frac{\langle S_1\,,\, s\rangle \to s' \qquad \langle S_2\,,\, s'\rangle \to s''}{\langle S_1; S_2\,,\, s\rangle \to s''}$$

$$\frac{\langle S_1\,,\, s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\,,\, s\rangle \to s'} \qquad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$\frac{\langle S_2\,,\, s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\,,\, s\rangle \to s'} \qquad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

$$\frac{\langle S\,,\, s\rangle \to s' \qquad \langle \texttt{while}\ b\ \texttt{do}\ S\,,\, s'\rangle \to s''}{\langle \texttt{while}\ b\ \texttt{do}\ S\,,\, s\rangle \to s''} \qquad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$\langle \texttt{while}\ b\ \texttt{do}\ S\,,\, s\rangle \to s \qquad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

Example

$s_{ij}(y) = i,\ s_{ij}(x) = j$
$s = s_{03}$

$$
\dfrac{
\langle \mathtt{y{:=}1},\ s\rangle \to s_{13}
\qquad
\dfrac{
\dfrac{
\langle \mathtt{y{:=}y\star x},\ s_{13}\rangle \to s_{33}
\qquad
\langle \mathtt{x{:=}x-1},\ s_{33}\rangle \to s_{32}
}{
\langle \mathtt{y{:=}y\star x;\ x{:=}x-1},\ s_{13}\rangle \to s_{32}
}
\qquad
\dfrac{
\dfrac{
\langle \mathtt{y{:=}y\star x},\ s_{32}\rangle \to s_{62}
\qquad
\langle \mathtt{x{:=}x-1},\ s_{62}\rangle \to s_{61}
}{
\langle \mathtt{y{:=}y\star x;\ x{:=}x-1},\ s_{32}\rangle \to s_{61}
}
\qquad
\langle \mathtt{while}\ \neg(\mathtt{x{=}1})\ \mathtt{do}\ (\mathtt{y{:=}y\star x;\ x{:=}x-1}),\ s_{61}\rangle \to s_{61}
}{
\langle \mathtt{while}\ \neg(\mathtt{x{=}1})\ \mathtt{do}\ (\mathtt{y{:=}y\star x;\ x{:=}x-1}),\ s_{32}\rangle \to s_{61}
}
}{
\langle \mathtt{while}\ \neg(\mathtt{x{=}1})\ \mathtt{do}\ (\mathtt{y{:=}y\star x;\ x{:=}x-1}),\ s_{13}\rangle \to s_{61}
}
}{
\langle \mathtt{y{:=}1;\ while}\ \neg(\mathtt{x{=}1})\ \mathtt{do}\ (\mathtt{y{:=}y\star x;\ x{:=}x-1}),\ s\rangle \to s_{61}
}
$$

Exercise 2.3

- Consider the statement

  z := 0; while y $\leq$ x do (z := z+1; x := x-y)

  Construct a derivation tree for the statement when executed in a state where x has the value 17 and y has the value 5.

Terminology

Derivation tree

- When we use the axioms and rules to derive a transition $\langle S,s\rangle \to s'$ we obtain a derivation tree:
  - the *root* of the tree is $\langle S,s\rangle \to s'$
  - the leaves of the tree are instances of the axioms
  - the *internal nodes* of the tree are the conclusions of instances of the rules; they have the corresponding instances of their premises as immediate sons
- The execution of S from s
  - *terminates* if there is a state s' such that $\langle S,s\rangle \to s'$
  - *loops* if there is *no* state s' such that $\langle S,s\rangle \to s'$

Exercise

Exercise 2.4

- Consider the following statements
  - while $\neg$ (x = 1) do (y := y*x; x := x-1)
  - while 1 $\leq$ x do (y := y*x; x := x-1)
  - while true do skip

  For each statement determine whether or not it always terminates or it always loops. Argue for your answer using the axioms and rules of the NS.

Semantic equivalence

- **Definition**: Two statements S_1 and S_2 are semantically equivalent if for all states *s* and *s'*

$$\langle S_1, s\rangle \rightarrow s' \text{ if and only if } \langle S_2, s\rangle \rightarrow s'$$

- **Lemma**: while *b* do *S* and if *b* then (*S*; while *b* do *S*) else skip are semantically equivalent

Lemma: while b do S and
if b then (S; while b do S) else skip
are semantically equivalent

Proof:
part 1

Assume
$\mathcal{B}$ [b]s = tt

$$\frac{\langle S,\ s\rangle \rightarrow s' \qquad \langle \texttt{while}\ b\ \texttt{do}\ S,\ s'\rangle \rightarrow s''}{\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \rightarrow s''}$$

$$\frac{\dfrac{\langle S,\ s\rangle \rightarrow s' \qquad \langle \texttt{while}\ b\ \texttt{do}\ S,\ s'\rangle \rightarrow s''}{\langle S;\ \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \rightarrow s''}}{\langle \texttt{if}\ b\ \texttt{then}\ (S;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else skip},\ s\rangle \rightarrow s''}$$

Lemma: while b do S and
if b then (S; while b do S) else skip
are semantically equivalent

Proof: part 2

Assume
$\mathcal{B}$ [b]s = ff

$\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s$

s=s'' must be the case

s=s'' must be the case

$$\frac{\langle \texttt{skip},\ s\rangle \to s''}{\langle \texttt{if}\ b\ \texttt{then}\ (S;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else skip},\ s\rangle \to s''}$$

Exercise

Exercise 2.6

- Prove that $(S_1; S_2); S_3$ and $S_1; (S_2; S_3)$ are semantically equivalent

Proof principle
for
natural semantics

Deterministic semantics

- **Definition**: the natural semantics is deterministic if for all statements *S* and states *s*, *s'* and *s''*

$$\langle S, s\rangle \to s' \text{ and } \langle S, s\rangle \to s'' \text{ imply } s' = s''$$

- **Lemma**: the natural semantics of the while language is deterministic

- **Proof**: by induction on the shape of the derivation tree

Induction on the shape of derivation trees

To prove a property of all the derivation trees of a natural semantics do the following:

- Prove that the property holds for all the simple derivation trees by showing that it holds for the axioms of the transition system.
- Prove that the property holds for all composite derivation trees: For each rule assume that the property holds for its premises — this is called the induction hypothesis — and prove that it also holds for the conclusion of the rule provided that the conditions of the rule are satisfied.

Summary: natural semantics

simple derivation trees

$$\langle x := a \,,\, s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\langle \texttt{skip} \,,\, s\rangle \to s$$

composite derivation trees

$$\frac{\langle S_1 \,,\, s\rangle \to s' \qquad \langle S_2 \,,\, s'\rangle \to s''}{\langle S_1; S_2 \,,\, s\rangle \to s''}$$

$$\frac{\langle S_1 \,,\, s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \,,\, s\rangle \to s'} \qquad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$\frac{\langle S_2 \,,\, s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \,,\, s\rangle \to s'} \qquad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

$$\frac{\langle S \,,\, s\rangle \to s' \qquad \langle \texttt{while}\ b\ \texttt{do}\ S \,,\, s'\rangle \to s''}{\langle \texttt{while}\ b\ \texttt{do}\ S \,,\, s\rangle \to s''} \qquad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$\langle \texttt{while}\ b\ \texttt{do}\ S \,,\, s\rangle \to s \qquad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

The exercise session today: Natural Semantics for the repeat construct

$$S \quad ::= \quad x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2$$
$$\quad \mid \quad \cancel{\texttt{while}\ b\ \texttt{do}\ S} \mid \boxed{\texttt{repeat}\ S\ \texttt{until}\ b}$$

The repeat construct

- The most complex construct of the While language is the while *b* do *S* construct.
- To improve your understanding of the material of the course you will get a sequence of exercises on the repeat *S* until *b* construct.

- These exercises will all be part of the first assignment to be handed in on March 1st.

Exercise 2.7 and 2.10

- Specify the semantics of the construct

  repeat S until b

  The specification is not allowed to rely on the existence of the while construct in the language.

  Prove that repeat S until b is semantically equivalent to S; if b then skip else (repeat S until b)
- Harder: Show that repeat S until b is semantically equivalent to S; while $\neg b$ do S

The rules of the game

several premises (n≥0)

$$\frac{\langle S_1, s_1\rangle \to s_1', \cdots, \langle S_n, s_n\rangle \to s_n'}{\langle S, s\rangle \to s'} \quad \text{if } \cdots$$

conclusion

computable side condition

OBS

- If we have derivation trees that matches the premises and if the side condition is fulfilled
- then we can construct a derivation tree for the conclusion

Preliminary plan

9. Feb 2005	**Lecture**: Natural semantics (NS); NN ch 1 and 2.1
	Exercises: 1.2, 1.13, 2.3, 2.4, 2.6, 2.7, 2.10
	Programming: A prototype interpreter for NS
16. Feb 2005	**Lecture**: Structural operational semantics (SOS); NN 2.2
	Exercises: more later
	Programming: A prototype interpreter for SOS
23. Feb 2005	**Lecture**: Comparison of NS and SOS; NN 2.3, 3.1
	Exercises: more later
	Programming: more later
1. March 2005	**Deadline for handing the first exercise in:**
	”Everything one would like to know about the repeat construct – and a little bit more”
	More details later …

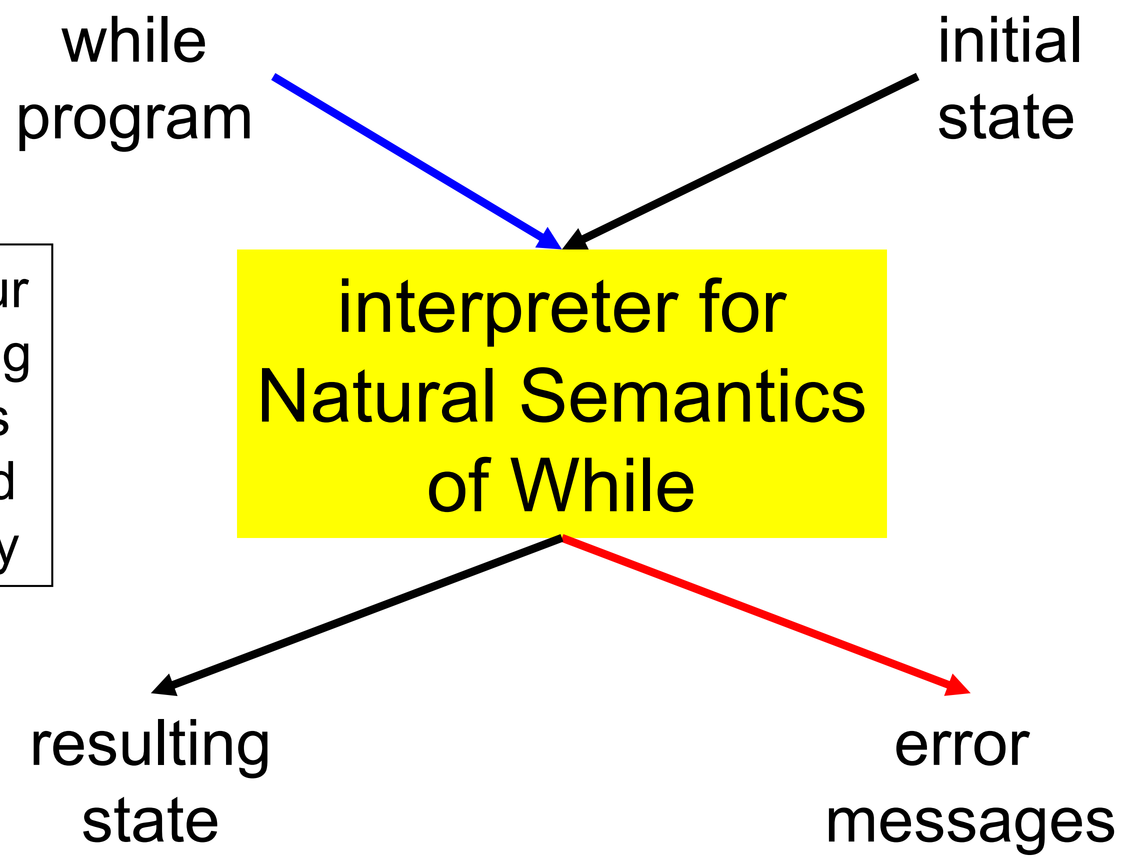
Programming exercises for today

We shall use SML

as this is very easy

but of course any language will do ...

Goal for today



improves your understanding of the axioms and rules and what they say review

Syntax in the theory

- Numerals
- Variables
- Arithmetic expressions

$$a \quad ::= \quad n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2$$

- Boolean expressions

$$b \quad ::= \quad \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2$$

- Statements

$$\begin{array}{lll} S & ::= & x := a \mid \texttt{skip} \mid S_1 ; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \\ & \mid & \texttt{while } b \texttt{ do } S \end{array}$$

- $n \in$ Num
- $x \in$ Var
- $a \in$ AExp
- $b \in$ BExp
- $S \in$ Stm

Syntax in SML

each syntactic category gives rise to a data type

```
type NUM = string
type VAR = string

datatype AEXP = Num   of NUM
              | Var   of VAR
              | Add   of AEXP * AEXP
              | Mult  of AEXP * AEXP
              | Sub   of AEXP * AEXP

datatype BEXP = tt
              | ff
              | …

datatype STM  = Ass   of VAR  * AEXP
              | Skip
              | …
```

Example: `y := y * x` becomes `Ass ("y", Mult (Var "y", Var "x"))`

Expressions

State = Var → Z

$\mathcal{N}$: Num → Z

$\mathcal{A}$: AExp → (State → Z)

$\mathcal{B}$: BExp → (State → T)

$$
\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \star \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{aligned}
$$

each semantic function gives rise to a SML function

```
type STATE = VAR -> int

(* N : NUM -> int *)

fun N n = valOf (Int.fromString n)

(* A: AEXP -> STATE -> int *)

fun A (Num n) s         = N n
 |  A (Var x) s         = s x
 |  A (Add (a1,a2)) s   = A a1 s + A a2 s
 |  A ...
```

Natural semantics

$$\langle x := a\ ,\ s\rangle \rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\langle \texttt{skip}\ ,\ s\rangle \rightarrow s$$

$$\frac{\langle S_1\ ,\ s\rangle \rightarrow s' \quad \langle S_2\ ,\ s'\rangle \rightarrow s''}{\langle S_1; S_2\ ,\ s\rangle \rightarrow s''}$$

$$\frac{\langle S_1\ ,\ s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ ,\ s\rangle \rightarrow s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$\frac{\langle S_2\ ,\ s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ ,\ s\rangle \rightarrow s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

$$\frac{\langle S\ ,\ s\rangle \rightarrow s' \quad \langle \texttt{while}\ b\ \texttt{do}\ S\ ,\ s'\rangle \rightarrow s''}{\langle \texttt{while}\ b\ \texttt{do}\ S\ ,\ s\rangle \rightarrow s''} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$\langle \texttt{while}\ b\ \texttt{do}\ S\ ,\ s\rangle \rightarrow s \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

the transition relation gives rise to a function in SML – why does that work, by the way?

```
datatype CONFIG
   = Inter of STM * STATE
   | Final of STATE
```

```
fun update x a s = ...

fun NS (Inter ((Ass (x,a)), s) )      = Final ...
 |  NS (Inter (Skip, s))              = Final s
 |  NS ..
```

Programming exercise

- Complete the SML implementation
- Test the implementations on programs like
  - y := 1; while ¬(x = 1) do (y := y * x; x := x – 1)
  - z := 0; while y ≤ x do (z := z+1; x := x-y)
  - while ¬ (x = 1) do (y := y*x; x := x-1)
  - while 1 ≤ x do (y := y*x; x := x-1)
  - while true do skip

  using a number of different states
- Extend the implementation to include the repeat construct