Programming Language Technology

Exam, 17 August 2016 at 14:00 – 18:00 in M

Course codes: Chalmers DAT151, GU DIT231.
Teacher: Fredrik Lindblad, will visit around 15:00 and 16:30. Phone: 031-7722038

Grading scale: Max = 60p, VG = 5 = 48p, 4 = 36p, G = 3 = 24p.
Allowed aid: an English dictionary.

Please answer the questions in English. Questions requiring answers in code can be answered in any of: C, C++, Haskell, Java, or precise pseudocode.

For any of the six questions, an answer of roughly one page should be enough.

Question 1 (Grammars): Write a labelled BNF grammar that covers the following constructs in a C-like imperative language: A program is a list of statements. Statement constructs are:

- `while` statements
- block statements (lists of statements surrounded by curly braces)
- expression statements (`E;`)

Expression constructs are:

- identifiers/variables
- integer literals
- function applications (`f(E,F,..)`)
- greater-than (`E > F`)
- multiplication (`E * F`)
- pre-decrement for variables (`--x`)

Operator precedences and associativity should follow the C standard. You can use the standard BNFC categories `Integer` and `Ident` as well as list shorthands, and `terminator`, `separator` and `coercions` rules. Note that function definitions should not be part of the grammar. (10p)

SOLUTION:

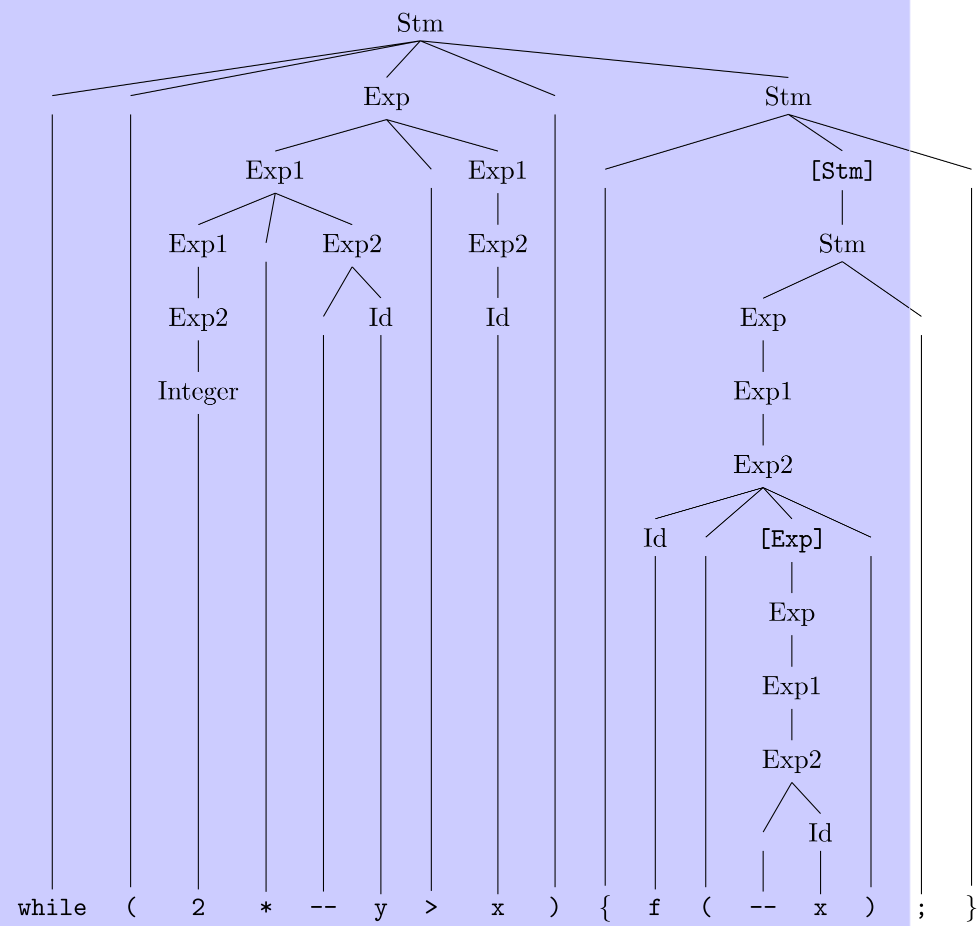
```
PStms.    Prg  ::= [Stm] ;

SWhile.   Stm  ::= "while" "(" Exp ")" Stm ;
SBlock.   Stm  ::= "{" [Stm] "}" ;
SExp.     Stm  ::= Exp ";" ;

EId.      Exp2 ::= Id ;
EInt.     Exp2 ::= Integer ;
EApp.     Exp2 ::= Id "(" [Exp] ")" ;
EPreDecr. Exp2 ::= "--" Id ;
ETimes.   Exp1 ::= Exp1 "*"  Exp2 ;
EGt.      Exp  ::= Exp1 ">"  Exp1 ;

coercions Exp 2 ;
terminator Stm "" ;
separator Exp "," ;

token Id (letter (letter | digit | '_')*) ;
```

Question 2 (Trees): Show the parse tree and the abstract syntax tree of the statement

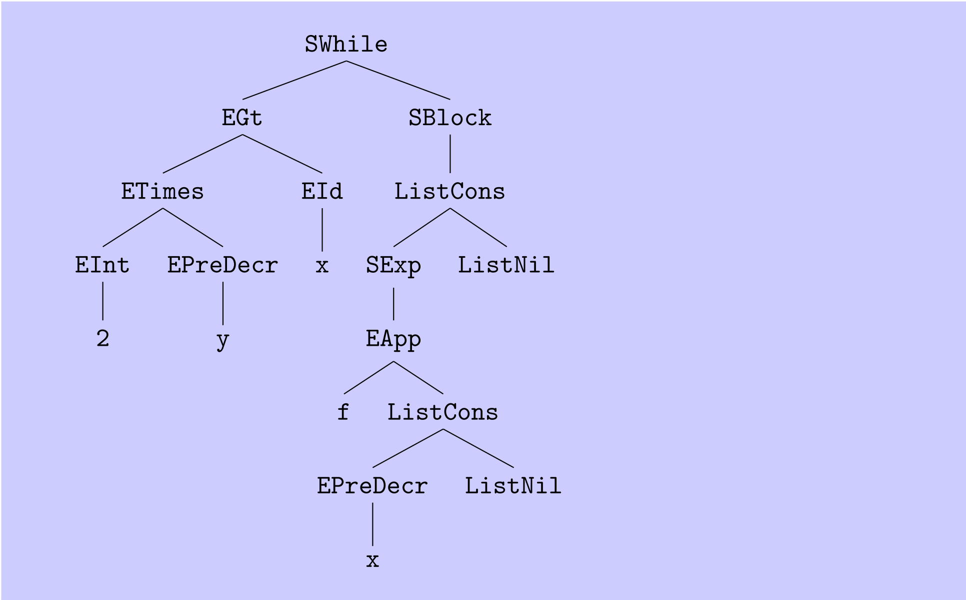
```
while (2 * --y > x) {f(--x);}
```

in the grammar that you wrote in question 1. In the parse tree show the coercions explicitly. (10p)

SOLUTION: Parse tree:



Abstract syntax tree:


SWhile
EGt
SBlock
ETimes
EId
ListCons
EInt
EPreDecr
x
SExp
ListNil
2
y
EApp
f
ListCons
EPreDecr
ListNil
x

Question 3 (Typing and evaluation):

A. Write standard typing rules or syntax-directed type-checking code (or pseudocode) for the following 5 constructs of the grammar in question 1: while-statements and expression forms variable/identifier, function application, pre-decrement and multiplication. The variable context must be made explicit. (5p)

SOLUTION:

$$\frac{\Gamma \vdash e : \texttt{bool} \qquad \Gamma \vdash s \ \mathit{valid}}{\Gamma \vdash \texttt{while (}e\texttt{)}\ s \ \mathit{valid}}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}$$

$$\frac{f : (T_1, \ldots, T_n) \rightarrow T \in \Gamma \qquad \Gamma \vdash e_1 : T_1 \qquad \ldots \qquad \Gamma \vdash e_n : T_n}{\Gamma \vdash f(e_1, \ldots, e_n) : T}$$

$$\frac{x : \texttt{int} \in \Gamma}{\Gamma \vdash \texttt{--}x : \texttt{int}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 * e_2 : \texttt{int}}$$

B. Write big-step operational semantic rules or syntax-directed interpretation code (or pseudocode) for the same 5 constructs as in part A. The environment must be made explicit. (5p)

SOLUTION:

$$\frac{\gamma \vdash e \Downarrow \langle \mathit{true}, \gamma' \rangle \qquad \gamma' \vdash s \Downarrow \gamma'' \qquad \gamma'' \vdash \texttt{while (}e\texttt{)}\ s \Downarrow \gamma'''}{\gamma \vdash \texttt{while (}e\texttt{)}\ s \Downarrow \gamma'''}$$

$$\frac{\gamma \vdash e \Downarrow \langle \mathit{false}, \gamma' \rangle}{\gamma \vdash \texttt{while (}e\texttt{)}\ s \Downarrow \gamma'}$$

$$\frac{x := v \in \gamma}{\gamma \vdash x \Downarrow \langle v, \gamma \rangle}$$

$$\frac{\begin{array}{c} \gamma \vdash e_1 \Downarrow \langle v_1, \gamma_1 \rangle \\ \gamma_1 \vdash e_2 \Downarrow \langle v_2, \gamma_2 \rangle \\ \ldots \\ \gamma_{n-1} \vdash e_n \Downarrow \langle v_n, \gamma_n \rangle \\ f(T_1\ x_1, \ldots, T_n\ x_n)\ \{s_1 \ldots s_m\} \in \gamma \\ \gamma.x_1 := v_1, \ldots, x_n := v_n \vdash s_1 \ldots s_m \Downarrow \langle v, \gamma' \rangle \end{array}}{\gamma \vdash f(e_1, \ldots, e_n) \Downarrow \langle v, \gamma_n \rangle}$$

$$\frac{x := v \in \gamma}{\gamma \vdash \texttt{--}x \Downarrow \langle v - 1, \gamma(x := v - 1) \rangle}$$

$$\frac{\gamma \vdash e_1 \Downarrow \langle v_1, \gamma' \rangle \qquad \gamma' \vdash e_2 \Downarrow \langle v_2, \gamma'' \rangle}{\gamma \vdash e_1 * e_2 \Downarrow \langle v_1 * v_2, \gamma'' \rangle}$$

Question 4 (Parsing):

A. Show a BNF grammar for expressions with the constructs boolean and, subtraction, less-than, variables and parentheses. Associativity and precedence should follow the C standard. The built-in BNFC `Ident` token type may be used, but no short-hands such as `coercions`. (4p)

B. Trace the LR-parsing of the expression `x && y - z < w`. Show how the stack and the input evolves and which actions are performed. (6p)

SOLUTION:

```
EId.    Exp3 ::= Ident ;
EMinus. Exp2 ::= Exp2 "-"  Exp3 ;
ELt.    Exp1 ::= Exp2 "<"  Exp2 ;
EAnd.   Exp  ::= Exp  "&&" Exp1 ;

Exp  ::= Exp1
Exp1 ::= Exp2
Exp2 ::= Exp3
Exp3 ::= "(" Exp ")"

stack                input            actions
                     x && y - z < w   s
x                    && y - z < w     r
Exp3                 && y - z < w     r
Exp2                 && y - z < w     r
Exp1                 && y - z < w     r
Exp                  && y - z < w     s
Exp &&               y - z < w        s
Exp && y             - z < w          r
Exp && Exp3          - z < w          r
Exp && Exp2          - z < w          s
Exp && Exp2 -        z < w            s
Exp && Exp2 - z      < w              r
Exp && Exp2 - Exp3   < w              r
Exp && Exp2          < w              s
Exp && Exp2 <        w                s
Exp && Exp2 < w                       r
Exp && Exp2 < Exp3                    r
Exp && Exp2 < Exp2                    r
Exp && Exp1                           r
Exp                                   accept
```

Question 5 (Compilation):

A. Write compilation schemes for each of the constructs of the grammar in question 1 except function application (in total 8 statement and expression constructs). It is not necessary to remember exactly the names of the JVM instructions – only what arguments they take and how they work. (6p)

SOLUTION:

```
compile(while (exp) stm) :
 TEST := newLabel()
 END := newLabel()
 emit(TEST:)
 compile(exp)
 emit(ifeq END)
 compile(stm)
 emit(goto TEST)
 emit(END:)

compile({stms}) :
 newBlock()
 foreach stm : stms
  compile(stm)
 exitBlock()

compile(exp;) :
 compile(exp)
 emit(pop)

compile(x) :
 emit(iload lookup(x))

compile(i) :
 emit(ldc i)

compile(exp1 > exp2) :
 TRUE := newLabel()
 emit(bipush 1)
 compile(exp1)
 compile(exp2)
 emit(if_icmpgt TRUE)
 emit(pop)
 emit(bipush 0)
 emit(TRUE:)
```

```
compile(exp1 * exp2) :
 compile(exp1)
 compile(exp2)
 emit(imul)

compile(--x) :
 emit(iload lookup(x))
 emit(bipush 1)
 emit(isub)
 emit(dup)
 emit(istore lookup(x))
```

B. Give the small-step semantics of the JVM instructions you used in the compilation schemes in part A. (4p)

SOLUTION: For each command, we give a transition $(P, V, S) \rightarrow (P', V', S')$ from old program counter P to its new value P', old variable store V to new store V', and old stack state S to new stack state S'. Stack $S.v$ shall mean that the top value on the stack is v, the rest is S. Jump targets L are used as instruction addresses, and $P+1$ is the instruction address following P.

instruction	state before		state after	
`goto` L	(P, V, S)	$\rightarrow$	(L, V, S)	
`ifeq` L	$(P, V, S.v)$	$\rightarrow$	(L, V, S)	if $v = 0$
`ifeq` L	$(P, V, S.v)$	$\rightarrow$	$(P+1, V, S)$	if $v \neq 0$
`if_icmpgt` L	$(P, V, S.v.w)$	$\rightarrow$	(L, V, S)	if $v > w$
`if_icmpgt` L	$(P, V, S.v.w)$	$\rightarrow$	$(P+1, V, S)$	if $v \leq w$
`iload` a	(P, V, S)	$\rightarrow$	$(P+1, V, S.V(a))$	
`istore` a	$(P, V, S.v)$	$\rightarrow$	$(P+1, V[a := v], S)$	
`ldc` i	(P, V, S)	$\rightarrow$	$(P+1, V, S.i)$	
`bipush` i	(P, V, S)	$\rightarrow$	$(P+1, V, S.i)$	(i is a byte)
`imul`	$(P, V, S.v.w)$	$\rightarrow$	$(P+1, V, S.(v * w))$	
`isub`	$(P, V, S.v.w)$	$\rightarrow$	$(P+1, V, S.(v - w))$	
`dup`	$(P, V, S.v)$	$\rightarrow$	$(P+1, V, S.v.v)$	
`pop`	$(P, V, S.v)$	$\rightarrow$	$(P+1, V, S)$	

Question 6 (Functional languages): Show the big-step operational semantics rules (not as code) for a functional language with the expression constructs function application, λ-abstraction, variables, integer literals and integer multiplication. The evaluation strategy should be call-by-value. Use closures and explicit environment. (6p)

SOLUTION:

$$\frac{\gamma \vdash f \Downarrow (\lambda x.e)\{\delta\} \qquad \gamma \vdash a \Downarrow u \qquad \delta, x := u \vdash e \Downarrow v}{\gamma \vdash f\, a \Downarrow v}$$

$$\frac{}{\gamma \vdash \lambda x.e \Downarrow (\lambda x.e)\{\gamma\}}$$

$$\frac{}{\gamma \vdash x \Downarrow v}\, x := v \in \gamma$$

$$\frac{}{\gamma \vdash i \Downarrow i}$$

$$\frac{\gamma \vdash e_1 \Downarrow i_1 \qquad \gamma \vdash e_2 \Downarrow i_2}{\gamma \vdash e_1 * e_2 \Downarrow i_1 \cdot i_2}$$

Show the derivation tree (using your operational semantics) of the evaluation of the expression

```
(\f -> f (f 5)) (\x -> 2 * x)
```

(4p)

SOLUTION:

$$\frac{\dfrac{}{\vdash (\lambda f.f\ (f\ 5)) \Downarrow (\lambda f.f\ (f\ 5))} \qquad \dfrac{}{\vdash (\lambda x.2 * x) \Downarrow (\lambda x.2 * x)} \qquad \dfrac{A}{f := (\lambda x.2 * x) \vdash f\ (f\ 5) \Downarrow 20}}{\vdash (\lambda f.f\ (f\ 5))\ (\lambda x.2 * x) \Downarrow 20}$$

Sub derivation A:

$$\frac{\dfrac{}{f := (\lambda x.2 * x) \vdash f \Downarrow (\lambda x.2 * x)} \qquad \dfrac{B}{f := (\lambda x.2 * x) \vdash f\ 5 \Downarrow 10} \qquad \dfrac{\dfrac{}{x := 10 \vdash 2 \Downarrow 2} \qquad \dfrac{}{x := 10 \vdash x \Downarrow 10}}{x := 10 \vdash 2 * x \Downarrow 20}}{f := (\lambda x.2 * x) \vdash f\ (f\ 5) \Downarrow 20}$$

Sub derivation B:

$$\dfrac{\dfrac{}{f := (\lambda x.2 * x) \vdash f \Downarrow (\lambda x.2 * x)} \qquad \dfrac{}{f := (\lambda x.2 * x) \vdash 5 \Downarrow 5} \qquad \dfrac{\dfrac{}{x := 5 \vdash 2 \Downarrow 2} \qquad \dfrac{}{x := 5 \vdash x \Downarrow 5}}{x := 5 \vdash 2 * x \Downarrow 10}}{f := (\lambda x.2 * x) \vdash f\ 5 \Downarrow 10}$$