Programming Language Technology

Exam, 6 April 2016 at 08:30 – 12:30 in M

Course codes: Chalmers DAT151, GU DIT231. As re-exam, also DAT150, TIN321 and DIT229/230.
Teacher: Fredrik Lindblad, will visit around 09:30 and 11:00.

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:

- `if` statements with non-optional `else` branch.
- block statements (lists of statements surrounded by curly braces)
- expression statements (`E;`)

Expression constructs are:

- identifiers/variables
- integer literals
- assignments of identifiers (`x = E`)
- addition (`E + F`)
- multiplication (`E * F`)

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. (10p)

SOLUTION:

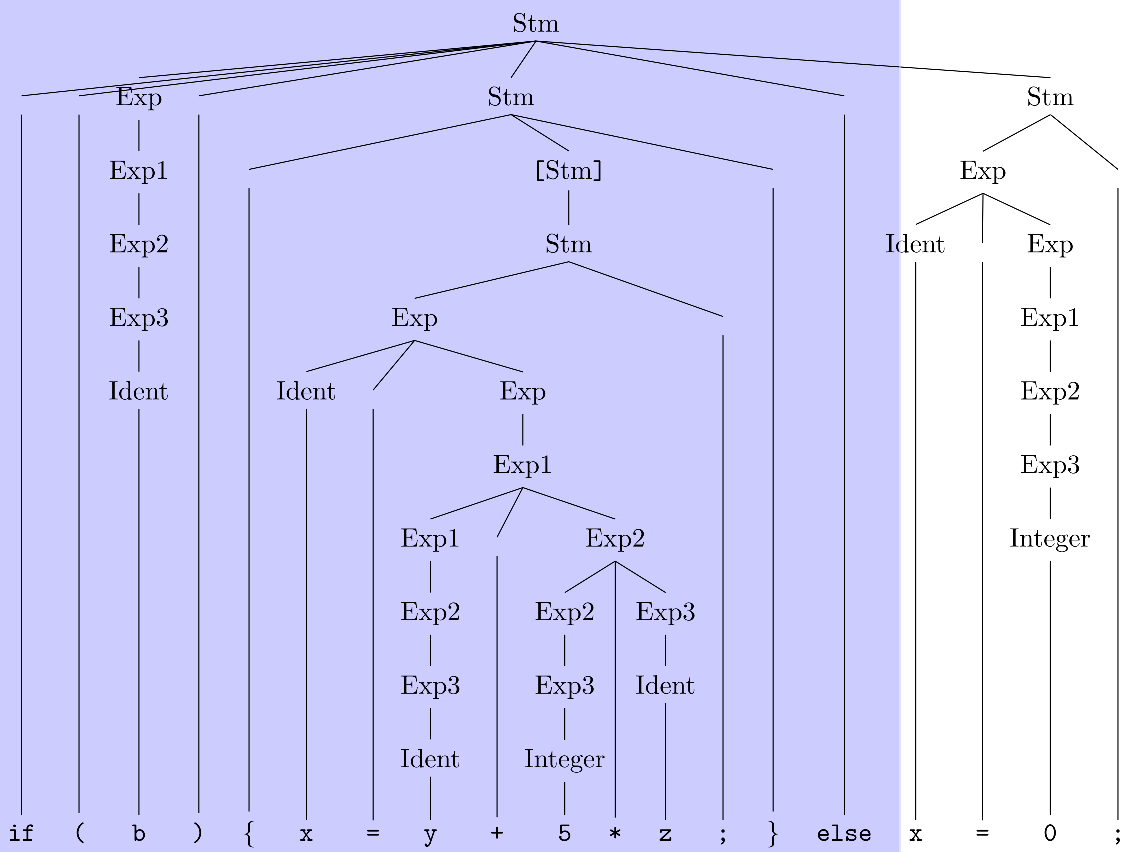
```
PStms.    Prg  ::= [Stm] ;

terminator Stm "" ;

SIf.      Stm  ::= "if" "(" Exp ")" Stm "else" Stm ;
SBlock.   Stm  ::= "{" [Stm] "}" ;
SExp.     Stm  ::= Exp ";" ;

EId.      Exp3 ::= Ident ;
EInt.     Exp3 ::= Integer ;
EMul.     Exp2 ::= Exp2 "*" Exp3 ;
EAdd.     Exp1 ::= Exp1 "+" Exp2 ;
EAss.     Exp  ::= Ident "=" Exp ;

coercions Exp 3 ;
```

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

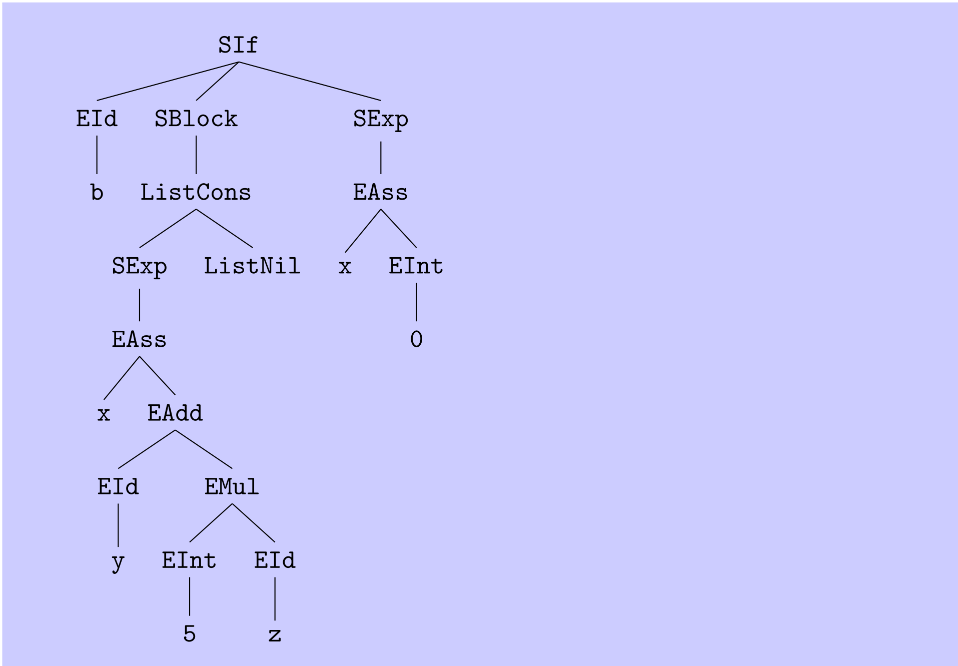
```
if (b) { x = y + 5 * z; } else x = 0;
```

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

SOLUTION: Parse tree:



Abstract syntax tree:


SIf
EId
SBlock
SExp
b
ListCons
EAss
SExp
ListNil
x
EInt
EAss
0
x
EAdd
EId
EMul
y
EInt
EId
5
z

Question 3 (Typing and evaluation):

A. Write standard typing rules or syntax-directed type-checking code (or pseudocode) for the *expression* constructs (5 constructs) of the grammar in question 1. The variable context must be made explicit. (5p)

SOLUTION:

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

$$\frac{}{\Gamma \vdash i : \texttt{int}}$$

$$\frac{x : T \in \Gamma \qquad \Gamma \vdash e : \texttt{T}}{\Gamma \vdash x = e : \texttt{T}}$$

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

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

B. Write big-step operational semantic rules or syntax-directed interpretation code (or pseudocode) for the expression constructs of the grammar in question 1. The environment must be made explicit. (5p)

SOLUTION:

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

$$\frac{}{\gamma \vdash i \Downarrow \langle i, \gamma \rangle}$$

$$\frac{\gamma \vdash e \Downarrow \langle v, \gamma' \rangle}{\gamma \vdash x = e \Downarrow \langle v, \gamma'(x := v) \rangle}$$

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

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

Question 4 (Regular expressions):

A. Write a regular expression the recognizes the following language (and only this): A string in the language is a sequence of tokens separated by one or more space-characters. A token is either of these two forms:

- Identifier: Any letter (a-z or A-Z) followed by any number of characters which are either a letter or a digit.
- String literal: A double quote ("), followed by any sequence of characters except double quote, followed by a double quote.

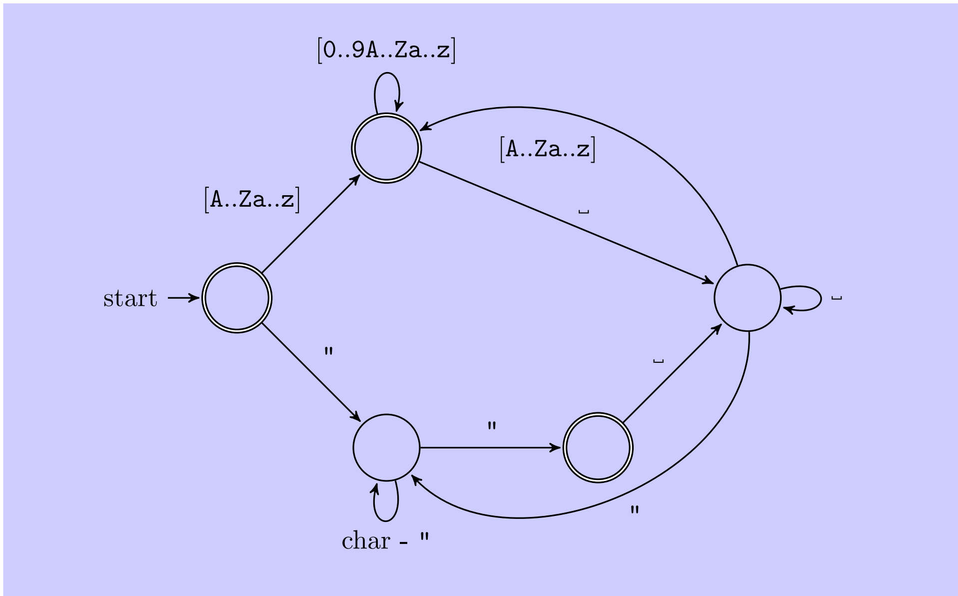
Do not use any short-hand regular expression constructs for letters and digits. You may refer to `char` as a short-hand for any character and `-` for which `A - B` represents the characters in `A` but not i `B`. (5p)

SOLUTION:

```
eps |
(((("A"|..|"Z"|"a"|..|"z")
   ("0"|..|"9"|"A"|..|"Z"|"a"|..|"z")*) |
  ('"' (char - '"')* '"'))
 (' ' ' '*
  ((("A"|..|"Z"|"a"|..|"z")
    ("0"|..|"9"|"A"|..|"Z"|"a"|..|"z")*) |
   ('"' (char - '"')* '"'))
)*)
```

B. Write a deterministic finite-state automaton (DFA) for the same language as in part A. (5p)

SOLUTION:


[0..9A..Za..z]
[A..Za..z]
[A..Za..z]
start
"
"
"
char - "

Question 5 (Compilation):

A. Write compilation schemes for each of the constructs (statement and expression, 8 in total) of the grammar in question 1. 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(if (exp) stm1 else stm2) :
 FALSE := newLabel()
 TRUE := newLabel()
 compile(exp)
 emit(ifeq FALSE)
 compile(stm1)
 emit(goto TRUE)
 emit(FALSE:)
 compile(stm2)
 emit(TRUE:)

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(x = exp) :
 compile(exp)
 emit(dup)
 emit(istore lookup(x))

compile(exp1 + exp2) :
 compile(exp1)
 compile(exp2)
 emit(iadd)
```

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

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 $v \neq 0$
`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)	$\rightarrow$	$(P + 1, V, S.i)$	
`iadd`	$(P, V, S.v.w)$	$\rightarrow$	$(P + 1, V, S.(v + w))$	
`imul`	$(P, V, S.v.w)$	$\rightarrow$	$(P + 1, V, S.(v * w))$	
`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, addition and multiplication. The evaluation strategy should be call-by-value. Use closures and explicit environment. (6p)

SOLUTION:

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

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

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

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

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

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

```
(\f -> x + f x) (\y -> x * y)
```

in the environment `{x := 3}`. (4p)

SOLUTION: Let γ be short-hand for $x := 3, f := (\lambda y.x * y)\{x := 3\}$.

$$\frac{\dfrac{}{x := 3 \vdash \lambda f.x + f\ x \Downarrow (\lambda f.x + f\ x)\{x := 3\}} \qquad \dfrac{}{x := 3 \vdash \lambda y.x * y \Downarrow (\lambda y.x * y)\{x := 3\}} \qquad \dfrac{\text{sub derivation}}{\gamma \vdash x + f\ x \Downarrow 12}}{x := 3 \vdash (\lambda f.x + f\ x)\ (\lambda y.x * y) \Downarrow 12}$$

sub derivation:

$$\frac{\dfrac{}{\gamma \vdash x \Downarrow 3} \qquad \dfrac{\dfrac{}{\gamma \vdash f \Downarrow (\lambda y.x * y)\{x := 3\}} \qquad \dfrac{}{\gamma \vdash x \Downarrow 3} \qquad \dfrac{\dfrac{}{x := 3, y := 3 \vdash x \Downarrow 3} \qquad \dfrac{}{x := 3, y := 3 \vdash y \Downarrow 3}}{x := 3, y := 3 \vdash x * y \Downarrow 9}}{\gamma \vdash f\ x \Downarrow 9}}{\gamma \vdash x + f\ x \Downarrow 12}$$