

## **CS 502 – Compiling and Programming Systems**

### **Mid-term Examination, 11/3/04**

**Instructions:** Read carefully through the whole exam first and plan your time. Note the relative weight of each question and part (as a percentage of the score for the whole exam). The total points is 100 (*ie*, your grade will be the percentage of your answers that are correct).

This exam is **open book, open notes**. You are free to refer to any book or other study materials you bring to the exam room.

You have **90 minutes** to complete all five (5) questions. Write your answers on this paper (use both sides if necessary).

**Name:**

**Student Number:**

**Signature**

1. (MiniJava front-end; 20%) Describe the changes you would need to make to each phase of the MiniJava compiler front-end in order to add support for Java native methods, which have syntax denoted in BNF as follows:

```
<Native> ::= static native <Type> <Id> ( <Formals> ) ;
```

where <Type> denotes a Java return type specifier (*eg*, `void`, `int` or some class name), <Id> denotes a Java identifier and <Formals> denotes the usual optional list of formal parameter declarations. A native method has no Java method body. Rather, the body is implemented in some other language (*eg*, C or assembler) and compiled separately from the native method's class. A call to the native method invokes that separately compiled code.

Your answer should specifically include descriptions of changes in:

- (a) (Scanning; 5%) **Answer:**

Add `native` keyword.

- (b) (Parsing; 5%) **Answer:**

Change syntax for method declarations to accept `native` modifier (like `static`), and tag `Absyn.MethodDec` nodes similarly. Allow method body to consist of empty body `;` as well as `{ ... }`

- (c) (Semantic processing; 5%) **Answer:**

Check that methods declared `native` have a empty method body, and that non-native methods have a non-empty method body. Visit and check method body only for non-native methods.

- (d) (Translation to IR trees; 5%) **Answer:**

Don't translate native method bodies (*i.e.*, don't generate code for them). Translate calls to `native` methods into external calls.

2. (Run-time representations; 10%) There are several issues that arise with native methods, since native methods are implemented in a language other than Java. When compiling a native method call, what precautions must the Java compiler take? When implementing a native method (*ie*, externally from Java), what precautions must the programmer take?

**Answer:**

Native method calls must be translated into calls using the native calling sequence (if different to the Java calling sequence). The Java compiler may need to convert method arguments appropriately. The programmer must be careful to define type structures in the native language that match the type structures of the Java implementation in order to access fields and invoke methods on Java objects received as parameters to native calls. The programmer must be careful to avoid hiding Java object references from the Java run-time system (especially the garbage collector); otherwise, the reference may become invalid (say because the GC moves the target of the reference) while the native code is executing if the garbage collector is allowed to run.

3. (Context free grammars, LL and LR parsing; 15%) Consider the following simple context free grammar:

$$\begin{aligned} S &\rightarrow AaAb \mid BbBa \\ A &\rightarrow \varepsilon \\ B &\rightarrow \varepsilon \end{aligned}$$

- (a) (5%) Is this grammar LL(1)? Why or why not?

**Answer:**

Yes, since  $\text{FIRST}(AaAb) \cap \text{FIRST}(BbBa) = \{a\} \cap \{b\} = \emptyset$ .

- (b) (5%) Is it SLR(1)? Why or why not?

**Answer:**

No, since  $\text{FOLLOW}(A) \cap \text{FOLLOW}(B) = \{a, b\} \cap \{a, b\} \neq \emptyset$ , so there will be a reduce/reduce conflict in the state corresponding to item set:

$$\begin{aligned} S &\rightarrow \bullet AaAb \\ S &\rightarrow \bullet BbBa \\ A &\rightarrow \bullet \\ B &\rightarrow \bullet \end{aligned}$$

- (c) (5%) Is it LR(1)? Why or why not?

[Hint: You need not build the LR(1) item sets to answer this]

**Answer:**

Yes, since  $\text{LL}(1) \subset \text{LR}(1)$

4. (Context free grammars, LR parsing; 20%)

(a) (10%) Consider the following simple context free grammar:

$$S \rightarrow Aa \mid bAc \mid dc \mid bda$$

$$A \rightarrow d$$

Is this grammar LALR(1)? Why or why not?

**Answer:**

Here are the LR(1) item sets:

$$\begin{array}{ll}
 I_1: S \rightarrow \bullet Aa, \$ & I_4: S \rightarrow d \bullet c, \$ \\
 S \rightarrow \bullet bAc, \$ & A \rightarrow d \bullet, a \\
 S \rightarrow \bullet dc, \$ & I_5: S \rightarrow Aa \bullet, \$ \\
 S \rightarrow \bullet bda, \$ & I_6: S \rightarrow bA \bullet c, \$ \\
 A \rightarrow \bullet d, a & I_7: S \rightarrow bd \bullet a, \$ \\
 I_2: S \rightarrow A \bullet a, \$ & A \rightarrow d \bullet, c \\
 I_3: S \rightarrow b \bullet Ac, \$ & I_8: S \rightarrow dc \bullet, \$ \\
 S \rightarrow b \bullet da, \$ & I_9: S \rightarrow bAc \bullet, \$ \\
 A \rightarrow \bullet d, c & I_{10}: S \rightarrow bda \bullet, \$
 \end{array}$$

There are no conflicts, so the grammar is LR(1). There are no common cores, so these are also the LALR(1) item sets. Since there are no conflicts the grammar is also LALR(1).

Is it SLR(1)? Why or why not?

**Answer:**

$\text{FOLLOW}(A) = \{a, c\}$ , so there is an SLR(1) shift-reduce conflict in states  $I_4$  and  $I_7$ ; the grammar is not SLR(1).

(b) (10%) Consider the following simple context free grammar for the same language:

$$\begin{aligned} S &\rightarrow Aa \mid bAc \mid Bc \mid bBa \\ A &\rightarrow d \\ B &\rightarrow d \end{aligned}$$

Is this grammar LR(1)? Why or why not?

**Answer:**

Here are the LR(1) item sets:

$$\begin{array}{ll} I_1: S \rightarrow \bullet Aa, \$ & I_4: S \rightarrow B \bullet c, \$ \\ S \rightarrow \bullet bAc, \$ & I_5: A \rightarrow d \bullet, a \\ S \rightarrow \bullet Bc, \$ & B \rightarrow d \bullet, c \\ S \rightarrow \bullet bBa, \$ & I_6: S \rightarrow Aa \bullet, \$ \\ A \rightarrow \bullet d, a & I_7: S \rightarrow bA \bullet c, \$ \\ B \rightarrow \bullet d, c & I_8: S \rightarrow bB \bullet a, \$ \\ I_2: S \rightarrow A \bullet a, \$ & I_9: A \rightarrow d \bullet, c \\ I_3: S \rightarrow b \bullet Ac, \$ & B \rightarrow d \bullet, a \\ S \rightarrow b \bullet Ba, \$ & I_{10}: S \rightarrow Bc \bullet, \$ \\ A \rightarrow \bullet d, c & I_{11}: S \rightarrow bAc \bullet, \$ \\ B \rightarrow \bullet d, a & I_{12}: S \rightarrow bBa \bullet, \$ \end{array}$$

Is it LALR(1)? Why or why not?

**Answer:**

Merging states  $I_5$  and  $I_9$  produces an LALR(1) reduce-reduce conflict, so the grammar is not LALR(1).

5. (Control flow graphs, liveness analysis, register allocation; 35%) The following program to compute the  $n^{\text{th}}$  number in the Fibonacci sequence has been compiled for a machine with 2 registers:

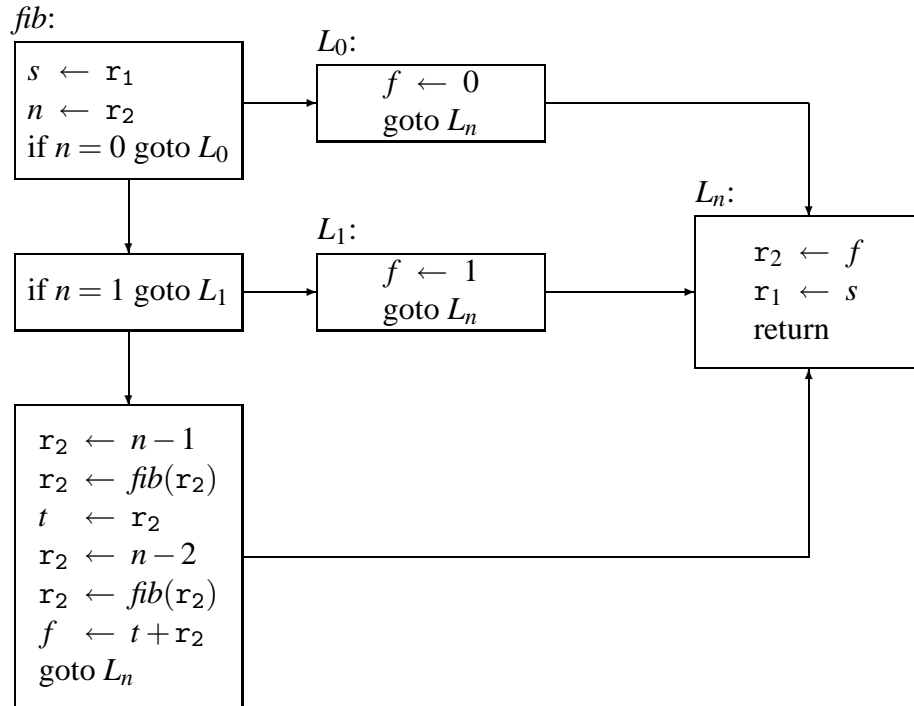
- $r_1$ : a callee-save register
- $r_2$ : a caller-save argument/result register

```
fib:  $s \leftarrow r_1$   
       $n \leftarrow r_2$   
      if  $n = 0$  goto  $L_0$   
      if  $n = 1$  goto  $L_1$   
       $r_2 \leftarrow n - 1$   
       $r_2 \leftarrow fib(r_2)$   
       $t \leftarrow r_2$   
       $r_2 \leftarrow n - 2$   
       $r_2 \leftarrow fib(r_2)$   
       $f \leftarrow t + r_2$   
      goto  $L_n$   
 $L_0$ :  $f \leftarrow 0$   
      goto  $L_n$   
 $L_1$ :  $f \leftarrow 1$   
      goto  $L_n$   
 $L_n$ :  $r_2 \leftarrow f$   
       $r_1 \leftarrow s$   
      return
```

(uses  $r_1, r_2$ )

- (a) (5%) Draw the control flow graph for this program, with nodes that are the program's *basic blocks* (ie, not individual instructions) and with edges representing the flow of control between the basic blocks.

**Answer:**



- (b) (10%) Annotate each *instruction* with the variables/registers live-out at that instruction.

	Def	Use	LiveOut
<i>fib</i> : $s \leftarrow r_1$			
$n \leftarrow r_2$			
if $n = 0$ goto $L_0$			
if $n = 1$ goto $L_1$			
$r_2 \leftarrow n - 1$			
$r_2 \leftarrow fib(r_2)$			
$t \leftarrow r_2$			
$r_2 \leftarrow n - 2$			
$r_2 \leftarrow fib(r_2)$			
$f \leftarrow t + r_2$			
goto $L_n$			
$L_0$ : $f \leftarrow 0$			
goto $L_n$			
$L_1$ : $f \leftarrow 1$			
goto $L_n$			
$L_n$ : $r_2 \leftarrow f$			
$r_1 \leftarrow s$			
return			

Answer:

	Def	Use	LiveOut
<i>fib</i> : $s \leftarrow r_1$	$s$	$r_1$	$sr_2$
$n \leftarrow r_2$	$n$	$r_2$	$sn$
if $n = 0$ goto $L_0$		$n$	$sn$
if $n = 1$ goto $L_1$		$n$	$sn$
$r_2 \leftarrow n - 1$	$r_2$	$n$	$snr_2$
$r_2 \leftarrow fib(r_2)$	$r_2$	$r_2$	$snr_2$
$t \leftarrow r_2$	$t$	$r_2$	$stn$
$r_2 \leftarrow n - 2$	$r_2$	$n$	$str_2$
$r_2 \leftarrow fib(r_2)$	$r_2$	$r_2$	$str_2$
$f \leftarrow t + r_2$	$f$	$tr_2$	$sf$
goto $L_n$			$sf$
$L_0$ : $f \leftarrow 0$	$f$		$sf$
goto $L_n$			$sf$
$L_1$ : $f \leftarrow 1$	$f$		$sf$
goto $L_n$			$sf$
$L_n$ : $r_2 \leftarrow f$	$r_2$	$f$	$sr_2$
$r_1 \leftarrow s$	$r_1$	$s$	$r_1r_2$
return		$r_1r_2$	

- (c) (5%) Fill in the following adjacency table representing the interference graph for the program; an entry in the table should contain an  $\times$  if the variable in the left column interferes with the corresponding variable/register in the top row. Since machine registers are pre-colored, we choose to omit adjacency information for them. Naturally, you must still record if a non-precolored node interferes with a pre-colored node; the columns for pre-colored nodes are there for that purpose.

Also, record the *unconstrained* move-related nodes in the table by placing an  $\circ$  in any empty entry where the variable in the left column is the source or target of any move involving the variable/register in the top row. **Remember that nodes that are move-related should not interfere if their live ranges overlap only starting at the move.**

	$r_1$	$r_2$	$s$	$n$	$t$	$f$
$s$						
$n$						
$t$						
$f$						

Answer:

	$r_1$	$r_2$	$s$	$n$	$t$	$f$
$s$	$\circ$	$\times$		$\times$	$\times$	$\times$
$n$		$\times$	$\times$		$\times$	
$t$		$\times$	$\times$	$\times$		
$f$		$\circ$	$\times$			

- (d) (15%) Show the steps of a coalescing graph-coloring register allocator as it assigns registers to the variables in the program. Use the *George criterion* for coalescing nodes: node  $a$  can be coalesced with node  $b$  only if all significant-degree (*ie*, degree  $\geq K$ ) neighbors of  $a$  already interfere with  $b$ . Show the final program, noting any redundant moves.

**Answer:**

- i. Coalesce  $f$  with  $r_2$ :

	$r_1$	$r_2$	$s$	$n$	$t$
$s$	○	×		×	×
$n$		×	×		×
$t$		×	×	×	

- ii. Possible spill  $t$ :

	$r_1$	$r_2$	$s$	$n$
$s$	○	×		×
$n$		×	×	

- iii. Possible spill  $s$ :

	$r_1$	$r_2$	$n$
$n$		×	

- iv. Simplify  $n$

- v. Color  $n \equiv r_1$

- vi. Actual spill  $s$

- vii. Actual spill  $t$

- viii. Retain coalesces from before first spill, rewrite program, recompute liveness:

	Def	Use	LiveOut
$fib: \mathcal{M}[s_{loc}] \leftarrow r_1$		$r_1$	$r_2$
$n \leftarrow r_2$	$n$	$r_2$	$n$
if $n = 0$ goto $L_0$		$n$	$n$
if $n = 1$ goto $L_1$		$n$	$n$
$r_2 \leftarrow n - 1$	$r_2$	$n$	$nr_2$
$r_2 \leftarrow fib(r_2)$	$r_2$	$r_2$	$nr_2$
$\mathcal{M}[t_{loc}] \leftarrow r_2$		$r_2$	$n$
$r_2 \leftarrow n - 2$	$r_2$	$n$	$r_2$
$r_2 \leftarrow fib(r_2)$	$r_2$	$r_2$	$r_2$
$t \leftarrow \mathcal{M}[t_{loc}]$	$t$		$tr_2$
$r_2 \leftarrow t + r_2$	$r_2$	$tr_2$	$r_2$
goto $L_n$			$r_2$
$L_0: r_2 \leftarrow 0$	$r_2$		$r_2$
goto $L_n$			$r_2$
$L_1: r_2 \leftarrow 1$	$r_2$		$r_2$
goto $L_n$			$r_2$
$L_n: r_1 \leftarrow \mathcal{M}[s_{loc}]$	$r_1$		$r_1r_2$
return		$r_1r_2$	

	$r_1$	$r_2$	$n$	$t$
$n$		×		
$t$		×		

- ix. Simplify  $n$
- x. Simplify  $t$
- xi. Color  $t \equiv r_1$
- xii. Color  $n \equiv r_1$
- xiii. Final program:

```

fib :  $\mathcal{M}[s_{loc}] \leftarrow r_1$ 
       $r_1 \leftarrow r_2$ 
      if  $r_1 = 0$  goto  $L_0$ 
      if  $r_1 = 1$  goto  $L_1$ 
       $r_2 \leftarrow r_1 - 1$ 
       $r_2 \leftarrow fib(r_2)$ 
       $\mathcal{M}[t_{loc}] \leftarrow r_2$ 
       $r_2 \leftarrow r_1 - 2$ 
       $r_2 \leftarrow fib(r_2)$ 
       $r_1 \leftarrow \mathcal{M}[t_{loc}]$ 
       $r_2 \leftarrow r_1 + r_2$ 
      goto  $L_n$ 
 $L_0$  :  $r_2 \leftarrow 0$ 
      goto  $L_n$ 
 $L_1$  :  $r_2 \leftarrow 1$ 
      goto  $L_n$ 
 $L_n$  :  $r_1 \leftarrow \mathcal{M}[s_{loc}]$ 
      return

```