Compiler Construction

Lab 3

Oct 19, FSU/CS

Your language is defined as before

$$
\begin{aligned}
\mathit{program} &\rightarrow (\texttt{define-fun}\ (\mathit{fun}\ (\mathit{type}\ \mathit{var})^*)\ \mathit{type}\ \mathit{expr})\ \mathit{program} \mid (\texttt{eval}\ \mathit{expr}) \\
\mathit{type} &\rightarrow \texttt{int} \mid \texttt{bool} \\
\mathit{expr} &\rightarrow \mathit{term} \mid \mathit{fla} \\
\mathit{term} &\rightarrow \mathit{const} \mid \mathit{var} \mid (\texttt{get-int}) \mid (\texttt{+}\ \mathit{term}\ \mathit{term}^+) \mid (\texttt{*}\ \mathit{term}\ \mathit{term}^+) \mid (\texttt{-}\ \mathit{term}\ \mathit{term}) \mid \\
&\quad (\texttt{div}\ \mathit{term}\ \mathit{term}) \mid (\texttt{mod}\ \mathit{term}\ \mathit{term}) \mid \\
&\quad (\texttt{if}\ \mathit{fla}\ \mathit{term}\ \mathit{term}) \mid (\mathit{fun}\ \mathit{expr}^*) \mid (\texttt{let}\ (\mathit{var}\ \mathit{expr})\ \mathit{term}) \\
\mathit{fla} &\rightarrow \texttt{true} \mid \texttt{false} \mid \mathit{var} \mid (\texttt{get-bool}) \mid \\
&\quad (\texttt{=}\ \mathit{term}\ \mathit{term}) \mid (\texttt{<}\ \mathit{term}\ \mathit{term}) \mid (\texttt{<=}\ \mathit{term}\ \mathit{term}) \mid (\texttt{>}\ \mathit{term}\ \mathit{term}) \mid (\texttt{>=}\ \mathit{term}\ \mathit{term}) \mid \\
&\quad (\texttt{not}\ \mathit{fla}) \mid (\texttt{and}\ \mathit{fla}\ \mathit{fla}^+) \mid (\texttt{or}\ \mathit{fla}\ \mathit{fla}^+) \mid (\texttt{if}\ \mathit{fla}\ \mathit{fla}\ \mathit{fla}) \mid \\
&\quad (\mathit{fun}\ \mathit{expr}^*) \mid (\texttt{let}\ (\mathit{var}\ \mathit{expr})\ \mathit{fla})
\end{aligned}
$$

Your task

Develop a translator from your AST to a CFG such that:

- It is written as a `visit_ast` (semantic actions in yacc won't be accepted)
- Virtual/input/output registers and names of basic blocks should be created on the fly
- The `entry` block for each function should begin with reading from input registers (`a1`, `a2`, etc..)
- The `exit` basic block for each function should have no instructions
- The `rv` register should be rewritten in every path of a CFG of every function
- Before every `call`, input registers should be rewritten
- After every `call`, the `rv` register should be read
- Every `if-then-else` (including nested) should be translated to at least four basic blocks: one for checking the guard (ending in a conditional `br`), the second two for the `then-` and the `else-`branches, respectively, both ending in the unconditional `br`s to the fourth basic block that merges the control flow
- While translating a `let`, translate the 2nd argument first, then assign the result to the 1st argument, and only then proceed to the 3rd argument

Your task (cont.)

You should create a CFG for input programs only if they pass all your semantic checks

- NOTE that since your language does not allow loops, your CFG should be acyclic

Add the pretty-printing functionality for your CFG (see examples in the next page and in the slides):

- Command-line pseudocode with all instructions
- A dot file `cfg.dot` (with the basic block dependencies only) that can be converted to a pdf file externally

- Printing should be done while visiting a CFG (not an AST or yacc productions)

As usual, please commit your test cases (semantically correct programs in the Lab-language, printed CFGs in text files and PDF files).

Example

Program:

```
(define-fun (t (bool a) (int b)) int
  (if a (if (= (get-int) 8) 50 44) b))
(eval (< 1 (t true 1)))
```

Printed instructions and CFG:

```
function t

entry:
v1 := a1
v2 := a2
br v1 bb2 bb3

bb2:
call GET-INT
v3 := rv
v4 := 8
v5 := v3 = v4
br v5 bb4 bb5

bb4:
v6 := 50
br bb6

bb5:
v6 := 44
br bb6

bb6:
v7 := v6
br bb7

bb3:
v7 := v2
br bb7

bb7:
rv := v7
br exit

PRINT

entry:
v1 := 1
a1 := TRUE
a2 := 1
call t
v2 := rv
v3 := v1 < v2
print v3
br exit
```