Class 36: Translating Conditionals (2)

Held: Wednesday, 23 November 2011

Summary: We continue our exploration of the translation of conditional statements.

Related Pages:
- EBoard.
- Due: Project, Phase 4: Type Checking.

Notes:
- Have a great Thanksgiving!
- But work on your homework.
- I should be on email off and on to answer questions.

Translating Short-Circuit Conditionals

- In short-circuit evaluation, your code needs to jump out of the expression evaluation code when the result of the boolean operator is known.
- Hence, we pass two extra parameters to the procedure for generating code for boolean expressions: the label to jump to if the expression is true and the label to jump to if the expression is false.
- The translation of the if statement is straightforward.

```c
code-for-short-circuit-expression
Jump label-for-truepart
label-for-truepart
  code-for-truepart
Jump label-for-end
label-for-falsepart
  code-for-falsepart
Jump label-for-end
label-for-end
```

- You may wonder what has happened to the first Jump-if-Zero. It’s now encapsulated in the code for the short-circuit expression.
  - Hence, we need to figure out how to translate boolean expressions.
- Here are some examples
/* Translate Boolean constants. */
translate (True, truelabel, falselabel)
    return new Instruction (Jump truelabel)
translate (False, truelabel, falselabel)
    return new Instruction (Jump falselabel)

/* Translate a less-than expression. */
translate (LessThan (Exp left, Exp right), truelabel, falselabel)
    CodeLoc l = translate (left);
    CodeLoc r = translate (right);
    return JoinCode (l.code, r.code,
        new Instruction (JumpLT truelabel l.loc r.loc),
        new Instruction (Jump falselabel));

/* Translate a not expression. */
translate (Not (subexp), truelabel, falselabel)
    return translate (not (subexp), falselabel, truelabel);

/* Translate an or expression. */
translate (Or (bexp1, bexp2), truelabel, falselabel)
    // The new label will fall between the code for bexp1
    // and the code to bexp2.
    Label newlabel = new Label ();
    // If bexp1 is true, jump to true label. Otherwise,
    // do the code for bexp2.
    Code code1 = translate (bexp1, truelabel, newlabel);
    // The result now depends only on bexp2.
    Code code2 = translate (bexp2, truelabel, falselabel);
    return JoinCode (code1, newlabel, code2);

/* Translate an and expression. */
translate (And (bexp1, bexp2), truelabel, falselabel)
    // The new label will fall between the code for bexp1
    // and the code to bexp2.
    Label newlabel = new Label ();
    // If bexp1 is false, jump to false label. Otherwise,
    // do the code for bexp2.
    Code code1 = translate (bexp1, newlabel, falselabel);
    // The result now depends only on bexp2.
    Code code2 = translate (bexp2, truelabel, falselabel);
    return JoinCode (code1, newlabel, code2);

/* Translate a Boolean variable. */
translate (BoolVar, truelabel, falselabel)
    return JoinCode (new Instruction (Jump-if-Zero falselabel BoolVar),
        new Instruction (Jump truelabel));

Short-Circuit Evaluation in Yacc/Bison

- How do we handle short-circuit evaluation in Yacc/Bison?
- That is, how do we handle short-circuit when we’re translating statements as we go, rather than after we build the tree?
- In effect, what we need to do is maintain the call stack ourselves.
if_then_else
 : _IF
 { 
  translating_boolean = 1; // Global flag
  Label *consequent_label = gen_label ();
  Label *alternate_label = gen_label ();
  push_label (consequent_label);
  push_label (alternate_label);
 }
 exp _THEN
 { 
  generate_instruction (_LABEL, consequent_label, NULL, NULL);
 }
 statement ELSE
 { 
  generate_instruction (_LABEL, alternate_label, NULL, NULL);
 }
 statement
 { 
  AttributeSet *attributes = new_attribute_set (0);
  $$ = new_internal_node (_if_then_else, attributes, 3, $3, $6, $9);
 }

• Then, the translation of the various things needs to grab those values.

exp : _TRUE
 { 
  Label *alternate_label = top_label ();
  Label *consequent_label = top2_label ();
  generate_instruction (JUMP, consequent_label, NULL, NULL);
  // ...
 }
 | _FALSE
 { 
  Label *alternate_label = top_label ();
  Label *consequent_label = top2_label ();
  generate_instruction (JUMP, alternate_label, NULL, NULL);
  // ...
 }
 |
 |
 { 
  Label *alternate_label = top_label ();
  Label *consequent_label = top2_label ();
  Label *new_consequent = gen_label ();
  push_label (new_consequent);
  push_label (alternate_label);
 } 
 exp _AND
 { 
  pop_label ();
  Label *new_consequent = pop_label ();
  generate_instruction (_LABEL, new_consequent, NULL, NULL);
 }
 exp ;
Translating Case Statements

- The simplest technique is to treat the case statement as a sequence of if s.
- However, if the cases are grouped together you can
  - Generate a label for each piece of code;
  - Make an array of those labels indexed by the corresponding value.
  - Do an indexed jump.
- An intermediate solution is to build a "search tree" for decisions.
  - We’ll try to look at some examples.