CS362 2011F *Compilers*

Class 35: Translating Conditionals (1)

Held: Monday, 21 November 2011

Summary: We begin our exploration of the translation of if-then statements and Boolean expressions.

Related Pages:

- EBoard.

Notes:

- I've made some changes to due dates, setting everything back by a week.
- No CS Extra this week.
- We'll spend a few minutes today considering issues related to type checking.

○

○ Basic Issues.

○ Translating with Strict Evaluation.

○ Translating with Short-Circuit Evaluation.

○ Short-Circuit Evaluation with Bison.

Basic Issues in Translating If Statements

- The first issue to consider in translating if statements is how to represent truth values. Typically, 0 is false and non-0 is true, but other options also exist.
  - We'll use the 0/non-0 form.
- It helps to remember that a basic if statement consists of a test expression, a true part, and a (possibly empty) false part.
- Surprisingly (or not so surprisingly) there are two common strategies for translating conditionals that are primarily related to the interpretation of boolean expressions involving `and` and `or`.
  - If you know that the first argument to `and` is false, do you need to evaluate the second?
  - If you know that the first argument to `or` is true, do you need to evaluate the second?
- If your language doesn't require the evaluation of all arguments to `and` and `or`, your language supports *short-circuit* evaluation.
- If your language requires the evaluation of all arguments to `and` and `or`, your language supports *strict* evaluation.
- Why would you choose short-circuit evaluation? Why would you choose strict evaluation?
- What does Pascal do?

Translating Strict Boolean Expressions

- Life is somewhat simpler with strict conditionals because you can use traditional expression evaluation strategies.
- Translate the expression, the true part, and the false part.
- Generate three new labels for the true part, the false part, and the end of the conditional.
- Join it all together with

```
  code-for-expression
  Jump-if-Zero label-for-falsepart location-for-expression-result
  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
```

- Why do we include that extra jumps at the beginning and end? Because they make it easier to rearrange the code, which we do in the optimization phase.

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.

```
  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, truelable, 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?
- We'll return to this question next class.