City University London

IN2009

Language Processors

Week 10

Translation to Intermediate Representation

Igor Siveroni

Session Plan

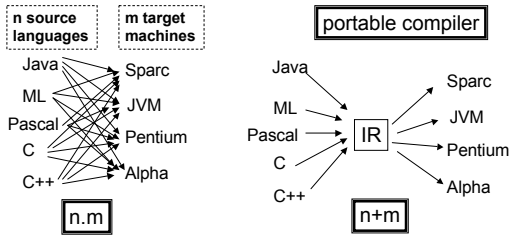
Session 7: Translation to intermediate representation

- Intermediate representation
- Why use IR
- Definition of an IR using trees
- Example translations
- See book for other translations.

Intermediate representation

- Semantic analysis
  - converts abstract syntax to abstract machine code (an intermediate rep)
- why an intermediate representation…?

Intermediate representation



Intermediate representation

- We represent the immediate output of the compilation using an *intermediate language* - an abstract machine language
  - Express target machine operations, but without machine-specific details
  - Source-language independent

Intermediate representation

- Compiler
  - front end: lexical analysis, parsing, semantic analysis
  - back end:
    - optimisation of IR (rewrite IR so as to improve execution speed)
    - translation to real machine language
    - (in case of Java, to another abstract machine language JVM)
- Many IRs
  - Appel uses *simple expression trees*

Real life IR and ILs

- A wide variety - for example gcc supports (amongst others):
  - RTL (register transfer language), GENERIC (tree-based), GIMPLE (a *static single assignment* language).
- Eiffel uses a simplified form of C.
- Java produces byte code.
- Microsoft .NET uses CIL.
- Other languages operate on pseudo-assembler or generic trees.

Intermediate representation

- Any good representation:
  - must be convenient for semantic analysis phase to produce.
  - must be convenient to translate to real (or virtual) machine language for target machines.
  - must have simple meaning for each construct that leads to simple operations on the IR to rewrite parts of it for optimisation etc.

Intermediate representation

- In any IR
  - individual components describe simple operations on the abstract machine represented by the IR instructions
  - each element of the complex abstract syntax is translated into a set of simple IR abstract machine instructions
  - groups of IR instructions will be grouped and regrouped to form real machine instructions

What does the IR abstract machine have?

- i.e. what do the trees represent/operate on?
  - integer constants
  - memory
  - registers [temporaries in the translation - infinite number in IR]
  - instruction set
  - sequential execution
  - labels and jumps

IR: tree expression operators

- `CONST(i)`
  - integer constant
- `NAME(n)`
  - symbolic constant (an assembly lang label)
- `TEMP(t)`
  - abstract register...infinite number!
- `BINOP(o,e1,e2)`
  - Where o = PLUS, MINUS, MUL, DIV, AND, OR, XOR, LSHIFT, RSHIFT, ARSHIFT

IR: tree expression operators

- `MEM(e)`
  - contents of wordSize bytes starting at addr e (means “store” if left child of `MOVE`, else “fetch”)
- `CALL(f,l)`
  - procedure call, applies f to list l
- `ESEQ(s,e)`
  - eval s for side-effects, eval e for result

IR: statements

- `MOVE(TEMP t,e)`
  - eval e & move into temp t
- `MOVE(MEM(e1),e2)`
  - eval e1 (⇒ addr a); eval e2 & store results into *wordsize* bytes of mem starting at a
- `EXP(e)`
  - eval e & discard the result

perform side effects & control flow

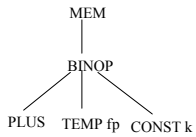
No provision for procedure and function defs - just body of each function

IR: statements

- `JUMP(e,labs)`
  - transfer control to addr e; labs specifies list of all poss locations e can eval to
- `CJUMP(o,e1,e2,t,f)`
  - eval e1 then e2, compare result with relational op o; if true jump to t else to f
- `SEQ(s1,s2)`
  - Execute statement s1 followed by s2
- `LABEL(n)`
  - defines const n to be current machine code address

Simple variables translation

- Simple variable v in current procedure or function stack frame
  - k: offset of v in frame
  - `TEMP fp`: frame pointer register



MEM(BINOP(PLUS,TEMP fp, CONST k))

Assignment Translation

- Assignment statement
  v = e
  - k: offset of v in frame
  - `TEMP fp`: frame pointer register
- Translated into:

MOVE(MEM(BINOP(PLUS,TEMP fp, CONST k)), e)

Boolean exp: Conditionals

- Boolean expressions are usually used as conditions for jumps.
- We will associate labels t and f to each boolean expression corresponding to jumps when the condition is true and false, respectively.
- Use `CJUMP`
  - x < 5 translates to **`CJUMP(LT, x, CONST(5), t, f)`**
  - for generated labels `t`, `f`

If-Statement translation

- If Statement:
  `if (e) then s1 else e2`
- Translated into:

```
SEQ(gen(e), // t,f labels of e
 SEQ(LABEL(t),
  SEQ(gen(s1),
   SEQ(JUMP(NAME(done)),
    SEQ(LABEL(f),
     gen(s2), LABEL(done)
```

If-Statement translation

- If Statement:
  `if (x < 5) then s1 else e2`
- Translated into:

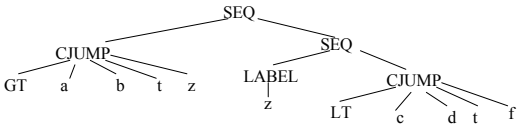
```
SEQ(CJUMP(LT,x,CONST(5),t,f)
 SEQ(LABEL(t),
  SEQ(gen(s1),
   SEQ(JUMP(NAME(done)),
    SEQ(LABEL(f),
     gen(s2), LABEL(done)
```


Conditionals: Example translation

a > b | c < d *translates to*

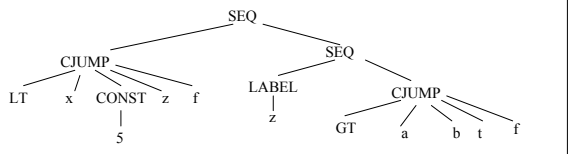
```
SEQ(CJUMP(GT,a,b,t,z),
     SEQ(LABEL z,
          CJUMP(LT,c,d,t,f)))
```

with t,f *labels*



Conditionals - (x<5)?(a>b):0

```
SEQ(CJUMP(LT,x,CONST(5),z,f),
  SEQ(LABEL z,
     CJUMP(GT,a,b,t [pick up val of a>b], f [pick up val 0]))
```



While Statements

- Statement: `while(e) s;`
- Translation sketch:
  test:
    if not(condition) goto *done*
    body
    goto *test*
  done:
- Translation?

What you should do now

- Figure out (check book) translations for:
  - for-loops
  - functions
  - declarations
- Complete the sample exam paper (Cityspace/Revision folder) before next week!

Schedule

- ***Module Summary & Exam Revision***
- Monday 27th April, 2009
  - 11:00 - 12:50
  - C.350