CSE 3302 Notes 6: Control Structure

(Last updated 10/21/15 8:11 PM)

References:

Gabbrielli: 6

6.1. EXPRESSIONS

Review items for expressions:

Position of operator (prefix, infix, postfix)
Precedence (C/C++/Java/JavaScript vs. Pascal)
Associativity
Arity - binary, unary, ternary (`?` `:`)
Dijkstra's shunting yard (`http://en.wikipedia.org/wiki/Shunting-yard_algorithm`)

C function call *arguments* are not required to be processed in a particular order (but is right-to-left for gcc, left-to-right for LLVM)

C etc. comma *operator* evaluates left-to-right (uses rightmost operand value as result)

see `http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES06/argOrder.c`

RELATIONAL BOOLEAN EXPRESSIONS

Fundamental difficulties with equality in logic & mathematics . . .

Notions of *equivalence* may be defined WRT a single function

Is an integer odd or even?
Is a function g in $\Theta(f)$?

What about *equality*?

Has to cover *all* notions of equivalence (addresses and references?)

For x and y to be equal, they are indistinguishable to *any* function

PLs

Shallow equality test - no dereferencing, tests whether values refer to same object?

Deep equality test - dereference and check values (cycles . . .)

ML = deep for *equality* types (more ML in notes 8)

Doesn't include `real`

```
- (1,2,3)=(1,2,3);
val it = true : bool

- [1,2,3]=[1,2,3];
val it = true : bool
```

(ML does allow `ref` types which function like pointers)

Scheme

`eq?` (shallow) and `equal?` (deep)

C

Besides comparing pointers with `==` and `!=`, can also use other comparisons (meaningful when dealing within same array, `struct`, etc.)

Pascal

Pointers may be compared only using equality comparisons (`=`, `<>`)

JavaScript: `==` vs. `===` and `!=` vs. `!==`

BOOLEAN EXPRESSIONS

Boolean operators to force sub-expression evaluation (for side effects)

C - Use `&` or `*` in place of `&&`, `|` or `+` in place of `||`

JavaScript `undefined`

Used when a property does not exist for an object.

To access `a.b.c.d` or get `undefined` (to avoid `TypeError`):

```
dCheck = a && a.b && a.b.c && a.b.c.d;
```

Based on short-circuit evaluation, JavaScript uses the last truthy/falsy value as result for `&&` and `||` (so do Scheme `and`/`or`, but `0` is truthy and only `#f` is falsy).

Misspelled property name vs. property with `undefined` as value . . .

`!!` sanitizes truthy/falsy value to `true` or `false`
`a || b` in JavaScript may be achieved in C using `a ? a : b`
`a && b` in JavaScript may be achieved in C using `a ? b : a`

SHORT-CIRCUIT BOOLEAN EVALUATION

C:

Left side of || and && is determined before right side, i.e. no portion of right side is evaluated before left side is determined.

"Give equivalent C code (e.g. using `if ... else ...`) to demonstrate the short-circuit nature of C boolean operators. Do not use `&&`, `||`, or `!` in your solution! Do not use work variables!"

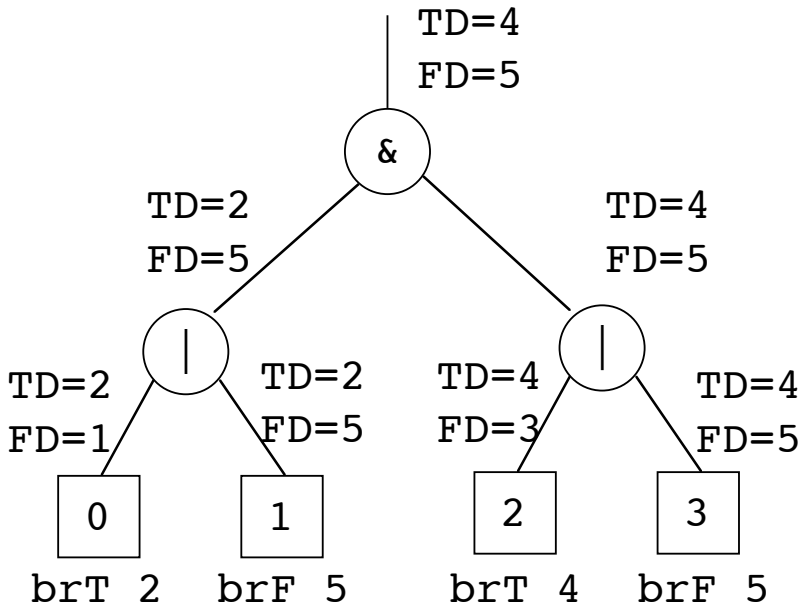
```
result = a < 13 && a > 10;
```

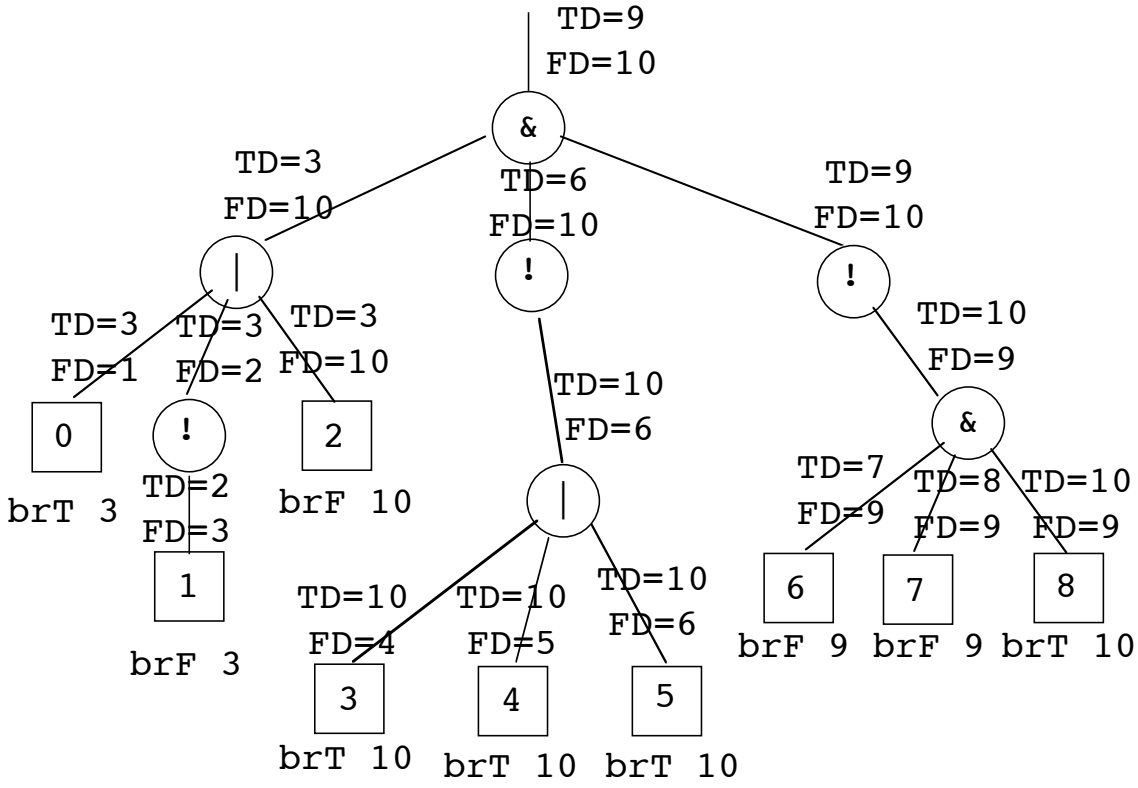
```
if (a < 13)
  if (a > 10)
    result = 1;
  else
    result = 0;
else
  result = 0;
```

```
result = e < 25 && !(f > 55 && g < 66);
```

```
if (e < 25)
  if (f <= 55)
    result = 1;
  else if (g >= 66)
    result = 1;
  else
    result = 0;
else
  result = 0;
```

http://ranger.uta.edu/~weems/NOTES3302/LAB/15SUM/LAB3/ - Conversion of expression with boolean result to jump-based code





6.2. COMMANDS (WITH SIDE EFFECTS)

l-value and r-value notions

Difference between reference model (Java) and modifiable variable model (C)

ASSIGNMENT

Shallow and *deep* differences again apply

Multiway (simultaneous, parallel) assignment

```
a, b = b, a;

i, j, a[i], a[j] = j, a[i], a[j], i;
```

What does this really save?

JavaScript - Destructuring assignment (also common in SML code, but strongly typed)

```
[a,b] = [1,2];
[a,b] = [b+1,a+3];
[a,a] = [b+2,a+1];   What happens?
```

6.3. Sequence Control

Explicit Sequencing

`;`, `{}`, `begin ... end`

Some much-maligned control structures:

`goto` (and its alterable versions - COBOL)

`break`/`continue`

`switch` (or long `if`/`else if` chains) - when used in superclass to avoid touching subclasses

> `http://www.amazon.com/gp/product/0321356683` - Item 20: Prefer class hierarchies to tagged classes

continuations (`goto` + state?, Notes 11)

(Aside: Knuth, "Structured Programming with go to Statements", esp. the acks on p. 296 `http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=356635.356640`)

Selection Statements

if ... then ... else ...

Switch

Generality of individual expressions

(small) integer values

JavaScript - general expressions and equality tests

Implementation

O(1) - table/hashtable
O(log n) - binary search
O(n) - like corresponding ifs (JavaScript)

Also, see Duff's device for exploiting C `case` fall-through property:

`http://en.wikipedia.org/wiki/Duff's_device`

ITERATIVE COMMANDS

Unbounded ("logically-controlled", `while`)

Bounded ("enumeration-controlled", `for`)

Just a special syntax for "while" *or* should number of iterations be predictable at onset?

Other issues:

Jumping into or out-of loop?

Is expression that index variable is tested against required to be constant?

Modifying index variable inside body?

Predictable value of index variable after loop termination?

Iterators - container abstraction (`foreach`)

Comparing two binary search trees?

Functional language iterators (see continuations in next section)

Aside: Backtrack programming and combinatorics
`http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=361219.361224`

6.4. STRUCTURED PROGRAMMING

p. 151 - Six elements from 1970s, but general support of types and abstraction came later.

`goto` may be acceptable as:

Multi-level `break`

In implementation of a state machine or statechart

6.5. TAIL RECURSION

Simplest form - activation record continues to exist only for passing back final value of recursive computation.

Accumulation/reduction - Procedure uses parameter to build result

```
(define (reverse l)
  (define (help l result)
    (cond
      ((empty? l) result)
      (else
       (help (cdr l) (cons (car l) result)))))
  (help l '()))
```

Scheme implementations are expected to treat tail recursion as iteration. Many can handle simple operators (e.g. `cons`) remaining after the call.