Algorithms and correctness

Formal correcness proofs

- Assertions
- Proof outline logic
- The inference rules
  - ∗ A system of reducing program correctness (P.O. validity) to the validity of a set of boolean expressions.
- Assignments: $P \Rightarrow R[x : e]$ ensures $\{P\}\, x := e\, \{R\}$
- Loops

  $$\{P\}\ A\ \{I\}\ \text{while}\ G\ \text{do}\ \{Q\}\ B\ \text{end while}\ \{R\}$$

  - ∗ The invariant must be established by initialization code: $\{P\}\, A\, \{I\}$ must be a valid PO
  - ∗ The invariant must be preserved by the loop body (assuming the guard true): $\{Q\}\ B\ \{I\}$ must be a valid PO
  - ∗ The invariant together with the negation of the guard imply the loop's postcondition $\neg G \wedge I \Rightarrow R$ must be a valid boolean expression.
- Loops should terminate. Use a variant: An expression such that $I \Rightarrow E \geq 0$ and that is decreased with each iteration.

Object invariant and data refinement

Using one set of variables to represent another

Contracts for procedures and recursion

Preconditions

- Obligation of caller
- Benefit to callee

Postconditions

- Obligation of the callee
- Benefit to the caller
- Conventions
  - Use x_0 for initial values and
  - result for the result.

Recursion

Make sure that some variant expression is getting smaller with each recursive call.

Typical pattern

- Divide the problem instance into (0 or more) subinstances
- Solve the subinstance
- Combine the solutions to the subinstances to make a solution to the instance.

In some cases we can just start by solving a lot of small instances and work our way up to bigger instances until we reach the one we really want to solve.

Loops and recursion

Every loop can be rewritten as a recursive routine. Thus techniques for writing recursive routines can be used instead of the invariant method.

Context free grammars

$(A, N, P, n_{\text{start}})$

One step replaces one nonterminal n with one string α such that $(n \to \alpha) \in P$.

The language defined by $(A, N, P, n_{\text{start}})$ is all strings in A^* reachable from n_{start} in one or more steps.

A parse tree gives a summary of a proof that its fringe is reachable from n_{start} in zero or more steps.

Context free grammars allow one to express the underlying tree structure of a text.

Recursive descent parsing

Assume $s \in A^*$ and $\$ \notin A$. Main code

$f := \text{true} \qquad s := s\hat{\ }[\$] \qquad \mathsf{n}_{\mathsf{start}}() \qquad f := f \wedge (s(0) = \$)$

Each nonterminal n becomes a routine n that tries to remove a prefix that matched n from s. And that may signal an error.

- Precondition: s is nonempty and ends with a $\$$.
- Postcondition: Either
  - Error: $f = \text{false}$ and s still ends with a $\$$.
  - Success: A prefix of s_0 that matches n has been

removed from s.

- How to choose.
  - $*$ If $f_0 = false$ then Error
  - $*$ If there is no prefix of s matching n then Error
  - $*$ If there is a *good* prefix of s matching n then Success (and the prefix removed from s must be good)
  - $*$ Otherwise: Either result is acceptable.
- A *good* prefix is one that will lead to a successful parse. Technically u is good if there are $v, t \in A^*$ so that $s_0 = ut\$$ and $n_{\text{start}}\$ \stackrel{*}{\Longrightarrow} vs_0$.
- The "Otherwise" case happens when there is a prefix that matches n but it is not good. Sometimes it is hard to tell that no prefix good. In this case, it is acceptable to just remove any prefix that matches n, because the error will be found eventually.