Synthesis of Sorting Algorithms

Lecture 9

Ras Bodik

CS294-2 Software Synthesis

Spring 2006

Administrativia

- Project proposals
  - Due in a week (Tue, Feb 21)
  - Work in pairs or alone
  - Format: ascii email
- Summaries
  - Preparing summaries of summaries takes too long
  - So, submit your summaries also to a web site
  - Anonymously if you prefer
    - I will email you a group anonymous account
  - Still submit also by email

Classification of sorting algorithms

- by insertion
  - insertion sort → shell sort
- by exchange
  - exchange sort → Quicksort
- by selection
  - selection sort → heap sort

- Classification idea:
  - by elementary operation
  - better algorithm viewed as optimization of simple one

Classify using synthesis

- due to Clark and Darlington

- sort(x1^x2, z1^z2)
  - perm(x1^x2, y1^y2) $\wedge$ sort(y1, z1) $\wedge$ sort(y2, z2)
    - quick sort
    - selection sort
  - sort(x1, y1) $\wedge$ sort(x2, y2) $\wedge$ perm(y1^y2, z1^z2)
    - merge sort
    - insertion sort

The specification language

- Two kinds of first-order logic sentences
- Implication
  - $p \Leftarrow q$
  - q contains no quantifiers
- Definition
  - $p \Leftrightarrow q$
  - q may contain quantifiers
  - Four forms:
    - conjunctive, disjunctive, existential, universal
    - to aid unfolding

Complete example: overview

- subset: specification

  subset(u, v) $\Leftrightarrow \forall$x . aux(x, v, u)

  aux(x, v, u) $\Leftrightarrow$ mem(x, v) $\vee$ $\neg$mem(x, u)

  mem(x, u^v) $\Leftrightarrow$ mem(x, u) $\vee$ mem(x, v)

- goal: desired recursive procedure (sketch)

  subset(u^v, …) $\Leftrightarrow$ … subset(u, …) … subset(v, …) …

- synthesized recursive procedure: implementation

  subset(u^v, s) $\Leftrightarrow$ subset(u, s) $\wedge$ subset(v, s)

  … base cases added manually

Goal decomposition, subsolution composition

- Want a solution to this goal:

  subset(a^b,) ⇐ … subset(a, …) … subset(b, …)

- Obtained from subproblems. Synthesize this first:

  aux(x, , a^b) ⇐ … aux(, , a) … aux(, , b) …

- Obtained how?

  subset(a^b,) subset(a,) subset(b,)

  ⇑ ⇓ ⇓

  aux(x, , a^b) ⇐ … aux(, , a) … aux(, , b)

Subproblem obtained automatically

subset(a^b, s) subset(a, s1) subset(b, s2)

⇑ ⇓ ⇓

aux(x, s , a^b) ⇐ … aux(x1, s1, a) … aux(x2, s2, b) …

- created with fold/unfold operation
  - break definitions into *if* and *only-if* parts, use substitutions from the higher-level goal
  - unfold formula: subset(a^b, s) ⇐ ∀x . aux(x, s, a^b)
  - fold formulae: subset(a, s1) ⇒ aux(x, s1, a)
    subset(b, s2) ⇒ aux(x, s2, b)

Subsolution composition (1)

- Assume we have solution to the subproblem:
  - our solution is simple:
    - bindings: x = x1 = x2; s = s1 = s2;
    - body: z1 ∧ z2
  - aux(x, s , a^b) ⇐ aux(x, s, a) ∧ aux(x, s, b)

- Simplify with bindings:

subset(a^b, s)		subset(a, s)		subset(b, s)
⇑		⇓		⇓
aux(x, s , a^b)	⇐	aux(x, s, a)	∧	aux(x, s, b)

Subsolution composition (2)

- solution:
  - aux(x, s , a^b) ⇐ aux(x, s, a) ∧ aux(x, s, b)
- unfold solution into unfold formula
  - subset(a^b, s) ⇐ ∀x . aux(x, s, a^b)
  - subset(a^b, s) ⇐ ∀x . (aux(x, s, a) ∧ aux(x, s, b))
- distribute the quantifier
  - subset(a^b, s) ⇐ (∀x . aux(x, s, a)) ∧ (∀x . aux(x, s, b))
- fold using folding formulae
  - subset(a^b, s) ⇐ (∀x . subset(a, s)) ∧ (∀x . subset(b, s))
- x does not appear in recursive call, so eliminate "∀x"
  - subset(a^b, s) ⇐ subset(a, s) ∧ subset(b, s)

Now come back to the aux subproblem

- **Folding/Unfolding to the subproblems:**

  aux(x, s , a^b) ⇐ … aux(x1, s1, a) … aux(x2, s2, b) …

  ⇑ ⇓ ⇓

  (1) mem(x, s) ⇐ … mem(x1, s1) … mem(x2, s2) …

  (2) ¬mem(x, a^b) ⇐ … ¬ mem(x1, a) … ¬mem(x2, b) …

- **Problem (1): directly solvable**

  mem(x, s) ⇐ mem(x, s), yielding bindings: x=x1=x2; s=s1=s2;

- **With bindings, Problem (2) becomes solvable**

  ¬mem(x, a^b) ⇐ … ¬ mem(x, a) … ¬mem(x, b) …

  ¬mem(x, a^b) ⇐ ¬ mem(x, a) ∧ ¬mem(x, b) (Def of mem)

Finish the aux subproblem

- **unfold two subsolutions into the unfold formula**

  aux(x, s, a^b) ⇐ mem(x, s) ∨ ¬mem(x, a^b)

  aux(x, s, a^b) ⇐ mem(x, s) ∨ ¬ mem(x, a) ∧ ¬mem(x, b)

- **convert to CNF**

  aux(x, s, a^b) ⇐

  (mem(x, s) ∨ ¬ mem(x, a)) ∧ (mem(x, s) ∨ ¬mem(x, b))

- **apply fold formulae**

  aux(x, s, a^b) ⇐ aux(x, s, a) ∧ aux(x, s, b)

Sort

$\text{sort}(a, b) \Leftrightarrow \text{perm}(a,b) \wedge \text{ord}(b)$

$\text{perm}(a, b) \Leftrightarrow \forall x . (\text{mem}(x, a) \Leftrightarrow \text{mem}(x, b))$

$\text{ord}(l) \Leftrightarrow \forall x \forall y . (x < y \Leftarrow \text{before}(x, y, l))$

The goal

sort(a1^a2, b)		… sort(a1, c) …		sort(a2,d) …
$\Uparrow$		$\Downarrow$		$\Downarrow$
perm(a1^a2, b)	$\Leftarrow$	… perm(a1, c) …		perm(a2,d) …
$\wedge$		$\wedge$		$\wedge$
ord(b)	$\Leftarrow$	… ord(c)	…	ord(d) …

Solve the perm subproblem

- (hierarchically obtained) solution to

  perm(a1^a2, b) ⇐ … perm(a1, c) … perm(a2,d) … is

  perm(a1^a2, b) ⇐ perm(a1, c) ∧ perm(a2,d) ∧ perm(c^d, b)
- unfold solution into unfold formula

  sort(a1^a2, b) ⇐ perm(a1^a2, b) ∧ ord(b)

  sort(a1^a2, b) ⇐ perm(a1, c) ∧ perm(a2,d) ∧ perm(c^d, b) ∧ ord(b)

  …

  sort(a1^a2, b) ⇐ sort(a1, c) ∧ sort(a2,d) ∧
  (perm(c^d, b) ∧ ord(b)) ⇐ (ord(c) ∧ ord(d))

The merge predicate

Give a name to the new predicate:

merge (c, d, b) ⇔ (perm(c^d, b) ∧ ord(b)) ⇐ (ord(c) ∧ ord(d))

Compare

- previous transformational paper

Example 1:

Spec:

- *fact(0)* $\Leftarrow$ *1*
- *fact(n+1)* $\Leftarrow$ *(n+1)*fact(n)*
- *factlist(0)* $\Leftarrow$ *nil*
- *factlist(n+1)* $\Leftarrow$ *cons(fact(n+1),factlist(n))*

Derivation:

5. *g(n)* $\Leftarrow \langle fact(n+1), factlist(n) \rangle$

   def (eureka)

6. *g(0)* $\Leftarrow \langle fact(1), factlist(0) \rangle$

   instantiate 5 with n=0

   $\Leftarrow \langle 1, nil \rangle$

   unfold 2, 1, law "*", unfold 4

Example 1, cont'd

7. *g(n+1)* ⇐ ⟨*fact(n+2),factlist(n+1)*⟩

inst. 5 with n=n+1

⇐ ⟨(n+2)**fact(n+1), cons(fact(n+1), factlist(n)*⟩

un 2,4

⇐ ⟨(n+2)**u, cons(u,v)* ⟩ where ⟨*u,v*⟩ = ⟨*fact(n+1),factlist(n)*⟩

abstract

⇐ ⟨(n+2)**u, cons(u,v)* ⟩ where ⟨*u,v*⟩ = *g(n)*

fold with 5

8. *factlist(n+1)* ⇐ *cons(fact(n+1),factlist(n))*

this is def 4, copied

⇐ *cons(u, v)* where ⟨*u,v*⟩ = ⟨*fact(n+1),factlist(n)*⟩

abstract

⇐ *cons(u, v)* where ⟨*u,v*⟩ = *g(n)*

fold with 5

Synthesis strategy

1. make necessary definitions
2. instantiate
3. for each instantiation unfold repeatedly, after each unfold:
   a. apply laws and where-abstraction
   b. fold repeatedly

User involvement:

- Invention needed in 1, 2.
- Discretion needed in a.
- rest is mechanical.