Equational Reasoning

Pete Manolios
Northeastern

Logic and Computation, 2/6/2019

Definitional Axioms

- When we admit a function with defunc, we get two axioms:
  - `ic ⇒ (f` x_1 `... ` x_n`) = body` (Recall binding power of =)
  - `ic ⇒ oc`
- Similarly for definec: generate the corresponding defunc
- In proofs we will not explicitly mention input contracts when using a function definition because contract completion (test?!)

Instantiation (PL)

- A substitution σ is a list of the form $((atom_1\ form_1) \dots (atom_n\ form_n))$
  - the atoms are the "targets" (no repetitions) and the forms are their "images"
  - by f|σ we mean, substitute every occurrence of a target by its image
  - e.g.: $(p \vee q \vee r)|((p\ q)\ (q\ (p \wedge s))\ (s\ u)) = q \vee (p \wedge s) \vee r$
- Instantiation: If f is valid, so is f|σ
  - e.g.: since $p \vee \neg p$ is valid, so is $(p \oplus q) \vee \neg(p \oplus q)$ (σ is $((p\ (p \oplus q)))$)
- A substitution σ is a list of the form $((var_1\ term_1) \dots (var_n\ term_n))$
  - the vars are the "targets" (no repetitions) and the terms are their "images"
  - by f|σ we mean, substitute every free occurrence of a target by its image
  - `(cons x (let ((y z)) y))|((x a) (y b) (z c) (w d)) = (cons a (let ((y c)) y))`
- Instantiation: If f is a *theorem*, so is f|σ
  - `(len (list x)) = 1` is theorem, so is `(len (list (list x y))) = 1`

Instantiation Examples

- A substitution σ is a list of the form ((var_1 $term_1$) … (var_n $term_n$))
  - the vars are the “targets” (no repetitions) and the terms are their “images”
  - by f|σ we mean, substitute every free occurrence of a target by its image
- Instantiation: If f is a *theorem*, so is f|σ
- Are the following substitutions correct? A:yes, B:no
- `(foo (cons (aapp w y) z))|((w (aapp b c)) (y (list a b)) (z (foo a)))`
  - `(foo (cons (aapp (aapp b c) (list a b)) (foo a)))`
- `(cons 'a b)|((a (cons a (list c))) (b (cons c nil)))`
  - `(cons 'a (cons c nil))`
- `(cons x (f x y f))|((x (cons a b)) (f x) (y (aapp y x)))`
  - `(cons (cons a b) (f (cons a b) (aapp y x) x))`

Example 1

- `(endp x) ⇒ (aapp (aapp x y) z) = (aapp x (aapp y z))`
- Proof?
- First, exportation: rewrite the conjecture so that we have as many hypotheses as possible (Propositional logic!). Nothing to do here.
- Then, contract completion: add needed hyps

  `(tlp x) ∧ (tlp y) ∧ (tlp z) ∧ (endp x) ⇒`

  `(aapp (aapp x y) z) = (aapp x (aapp y z))`
- Next, generate context, derived context, goal and proof

Example 1

Context

- C1:(tlp x)
- C2:(tlp y)
- C3:(tlp z)
- C4:(endp x)

Conjecture:

(tlp x) ∧ (tlp y) ∧ (tlp z) ∧ (endp x) ⇒

(aapp (aapp x y) z) = (aapp x (aapp y z))

Derived Context

- D1:x=nil { Def tlp C1, C4, cons axioms}

Goal:(aapp (aapp x y) z) = (aapp x (aapp y z))

Typically skip goal in hand proofs

```
  (aapp (aapp x y) z)
= { Def aapp, D1 }
  (aapp y z)
= { Def aapp, D1 }
  (aapp x (aapp y z))
```

Proof

```
(definec aapp (x :tl y :tl) :tl
 (if (endp x)
     y
     (cons (first x) (aapp (rest x) y)]
```

Notation to close off all parens

Feel free to use on exams

Example 2

- `(aapp (aapp x y) z) = (aapp x (aapp y z))`
- Exportation (nothing), contract completion:

  `(tlp x) ∧ (tlp y) ∧ (tlp z) ⇒`

  `(aapp (aapp x y) z) = (aapp x (aapp y z))`
- Proof?
- We can’t prove this right now. It will require induction
- What can we prove?

Example 3

```
(consp x) ∧
(aapp (aapp (rest x) y) z) = (aapp (rest x) (aapp y z))
⇒ (aapp (aapp x y) z) = (aapp x (aapp y z))
```

Exportation, Contract completion:

```
(tlp x) ∧ (tlp y) ∧ (tlp z) ∧ (consp x) ∧
(aapp (aapp (rest x) y) z) = (aapp (rest x) (aapp y z))
⇒ (aapp (aapp x y) z) = (aapp x (aapp y z))
```

Example 3

```
(definec aapp (x :tl y :tl) :tl
 (if (endp x)
     y
   (cons (first x) (aapp (rest x) y))]
```

- `C1:(tlp x)`
- `C2:(tlp y)`
- `C3:(tlp z)`
- `C4:(consp x)`
- `C5:(aapp (aapp (rest x) y) z) = (aapp (rest x) (aapp y z))`

Proof

```
   (aapp (aapp x y) z)
```

Example 3

```
(definec aapp (x :tl y :tl) :tl
 (if (endp x)
      y
    (cons (first x) (aapp (rest x) y)]
```

- C1:(tlp x)
- C2:(tlp y)
- C3:(tlp z)
- C4:(consp x)
- C5:(aapp (aapp (rest x) y) z) = (aapp (rest x) (aapp y z))

```
Proof
   (aapp (aapp x y) z)
= { Def of aapp, C4 }
   (aapp (cons (first x) (aapp (rest x) y)) z)
```

Example 3

```
(definec aapp (x :tl y :tl) :tl
 (if (endp x)
      y
    (cons (first x) (aapp (rest x) y)]
```

- C1:(tlp x)
- C2:(tlp y)
- C3:(tlp z)
- C4:(consp x)
- C5:(aapp (aapp (rest x) y) z) = (aapp (rest x) (aapp y z))

Proof

```
   (aapp (aapp x y) z)
= { Def of aapp, C4 }
   (aapp (cons (first x) (aapp (rest x) y)) z)
= { Def aapp, cons axioms }
   (cons (first x) (aapp (aapp (rest x) y) z))
```

Example 3

```
(definec aapp (x :tl y :tl) :tl
 (if (endp x)
     y
   (cons (first x) (aapp (rest x) y)]
```

- C1:(tlp x)
- C2:(tlp y)
- C3:(tlp z)
- C4:(consp x)
- C5:(aapp (aapp (rest x) y) z) = (aapp (rest x) (aapp y z))

Proof

```
   (aapp (aapp x y) z)
= { Def of aapp, C4 }
   (aapp (cons (first x) (aapp (rest x) y)) z)
= { Def aapp, cons axioms }
   (cons (first x) (aapp (aapp (rest x) y) z))
= { C5 }
   (cons (first x) (aapp (rest x) (aapp y z)))
```

Example 3

```
(definec aapp (x :tl y :tl) :tl
 (if (endp x)
      y
    (cons (first x) (aapp (rest x) y)]
```

- C1:(tlp x)
- C2:(tlp y)
- C3:(tlp z)
- C4:(consp x)
- C5:(aapp (aapp (rest x) y) z) = (aapp (rest x) (aapp y z))

Proof

```
   (aapp (aapp x y) z)
= { Def of aapp, C4 }
   (aapp (cons (first x) (aapp (rest x) y)) z)
= { Def aapp, cons axioms }
   (cons (first x) (aapp (aapp (rest x) y) z))
= { C5 }
   (cons (first x) (aapp (rest x) (aapp y z)))
= { Def aapp, C4 }
   (aapp x (aapp y z))
```

Example 4

- True or false?

```
(implies (consp x)
         (implies (and (tlp x)
                       (tlp y))
                  (implies (endp x)
                           (equal (aapp x y) (rrev y)))))
(definec rrev (x :tl) :tl
 (if (endp x)
     ()
   (aapp (rrev (rest x)) (list (first x)))))
```

- Prove/disprove it in groups.
  - A: we have a proof
  - B: we have a counterexample
  - C:not sure??

Propositional Skeleton

```
(implies (consp x)
         (implies (and (tlp x)
                       (tlp y))
                  (implies (endp x)
                           (equal (aapp x y) (rrev y)))))
```

A, B, C, D, E

$$A \Rightarrow (B \wedge C \Rightarrow (D \Rightarrow E))$$
$$= A \wedge B \wedge C \Rightarrow (D \Rightarrow E)$$
$$= A \wedge B \wedge C \wedge D \Rightarrow E$$

Exportation!

```
(implies (and (consp x)
              (tlp x)
              (tlp y)
              (endp x))
         (equal (aapp x y) (rev y)))
```

During exportation we only manipulate the propositional skelton

Context

```
(implies (and (consp x)
              (tlp x)
              (tlp y)
              (endp x))
         (equal (aapp x y) (rev y)))
```

```
C1.(consp x)
C2.(tlp x)
C3.(tlp y)
C4.(endp x)
------------------
D1. x=nil {Def tlp, C2, C4, cons axioms}
D2. nil   {C1, D1, cons axioms} (or {C1, C4, cons axioms})
```

Next Time

- More Equational Reasoning
- This is new for most of you, so start practicing