Lecture 3

Pete Manolios
Northeastern

Definitions Review

```
(defunc f (x1 … xn)
  :input-contract ic
  :output-contract oc
  body)
```

- f is *admissible* provided:
  - f is a new function symbol
  - the xi are distinct variable symbols
  - body is a term, possibly using f recursively as a function symbol, mentioning no variables freely other than the xi
  - the function is terminating
  - ic ⇒ oc is a theorem
  - the body contracts hold under the assumption that ic holds
- When we admit f, we get the following
  - Definitional axiom: `ic ⇒ (f x1 ... xn) = body`
  - Contract theorem: `ic ⇒ oc`

Measure Functions

- We use measure functions to prove termination.
- m is a measure function for f if all of the following hold.
  - m is an admissible function defined over the parameters of f;
  - m has the same input contract as f;
  - m has an output contract stating that it always returns a natural number; and
  - on every recursive call, m applied to the arguments to that recursive call decreases, under the conditions that led to the recursive call.

Measure Function Example

```
(defunc drop-last (x)
  :input-contract (true-listp x)
  :output-contract (true-listp (drop-last x))
  (cond ((endp x) nil)
        ((endp (rest x)) nil)
        (t (cons (first x) (drop-last (rest x))))))
```

- What is a measure function?
- `(len x)`

Measure Function Example

```
(defunc prefixes (l)
  :input-contract (true-listp l)
  :output-contract (true-listp (prefixes l))
  (cond ((endp l) '( () ))
        (t (cons l (prefixes (drop-last l))))))
```

- Is prefixes admissible?
- Yes. Use `(len l)`
- But, our main proof obligation is:

```
(implies (and (true-listp l)
              (not (endp l)))
         (< (len (drop-last l)) (len l)))
```

- This needs a proof by induction
- Common pattern: f’s definition uses g
  - to prove termination of f, we often need “size” theorems about g

ACL2-count

A very useful, built-in function, since ACL2s uses this function to build measure functions.

```
(defun acl2-count (x)
  (cond ((consp x)
         (+ 1 (acl2-count (car x))
            (acl2-count (cdr x))))
        ((integerp x) (integer-abs x))
        ((rationalp x)
         (+ (integer-abs (numerator x))
            (denominator x)))
        ((complex/complex-rationalp x)
         (+ 1 (acl2-count (realpart x))
            (acl2-count (imagpart x))))
        ((stringp x) (length x))
        (t 0)))
```

Observation

- We require a measure function to return a natural number
- But sometimes need more than a natural number to prove termination
- We need infinite numbers!
- An example is the "weird" function below (Ackermann)
- Try proving that is terminating and you'll see what I mean

```
(defunc weird (x y)
  :input-contract (and (natp x) (natp y))
  :output-contract (posp (weird x y))
  (cond ((equal x 0) (+ 1 y))
        ((equal y 0) (weird (- x 1) 1))
        (t (weird (- x 1) (weird x (- y 1))))))
```

Observation

- There are simple programs for which no one knows whether they terminate
- And no one has any good idea on how to prove that they do or don't
- Here is a simple, famous example

```
(defunc c (n)
  :input-contract (natp n)
  :output-contract (natp (c n))
  (cond ((< n 2) n)
        ((integerp  (/ n 2)) (c (/ n 2)))
        (t (c (+ 1 (* 3 n))))))
```

- The claim that it terminates is called the "Collatz conjecture."
- Paul Erdos: "Mathematics may not be ready for such problems."

Controlling ACL2s

- `:program` mode turns off theorem proving in ACL2s
  - no termination analysis is attempted
  - ACL2s will still *test* contracts and report any errors it finds
  - useful for prototyping & experimenting
- `:logic` mode is the default mode and allows you to switch back
  - you cannot define `:logic` mode functions if they depend on `:program` mode functions
- Other useful settings
  - `(acl2s-defaults :set testing-enabled nil)`
  - `(set-defunc-termination-strictp nil)`
  - `(set-defunc-function-contract-strictp nil)`
  - `(set-defunc-body-contracts-strictp nil)`

DEMO