Introduction to computer programming

Spring, 2013

Chen-Mou Cheng

National Taiwan University
Taipei, Taiwan
ccheng@cc.ee.ntu.edu.tw

February 28, 2013

Inductive specification of set S

Definition

A natural number n is in S if and only if

1. $n = 0$, or
2. $n - 3 \in S$.

Inductive specification of set S

Definition

A natural number n is in S if and only if

1. $n = 0$, or
2. $n - 3 \in S$.

```
def in_S(n):
  '''
  in_S(n) = True if n is in S, False otherwise
  '''
  if n == 0:
    return True
  elif n - 3 >= 0:
    return in_S(n - 3)
  else:
    return False
```

An alternative form

Definition

Define the set S to be the smallest set contained in $\mathbb{N}$ and satisfying the following two properties:

1. $0 \in S$, and
2. if $n \in S$, then $n + 3 \in S$.

Or in shorthand notation:

1. $\dfrac{}{0 \in S}$
2. $\dfrac{n \in S}{n + 3 \in S}$

Exercise (in class, perhaps using Excel)

Write inductive definitions of the following sets. Write each definition in all three styles (top-down, bottom-up, and rules of inference). Show some sample elements of each set using Excel.

1. $\{3n + 2 \mid n \in \mathbb{N}\}$
2. $\{2n + 3m + 1 \mid n, m \in \mathbb{N}\}$
3. $\{(n, 2n + 1)\} \mid n \in \mathbb{N}\}$
4. $\{(n, n^2) \mid n \in \mathbb{N}\}$ *(Do not mention squaring in your rules. As a hint, remember the equation* $(n + 1)^2 = n^2 + 2n + 1$.*)*

Exercise (in class)

Find a set T *of natural numbers such that* $0 \in T$*, and whenever* $n \in T$*, then* $n + 3 \in T$*, but* $T \neq S$*.*

List of integers, top-down

Definition

A Python list is a list of integers if and only if either

1. it is the empty list `[]`, or
2. it is a pair whose `car` is an integer and whose `cdr` is a list of integers.

```
car = lambda lst: lst[0]
cdr = lambda lst: lst[1:]
```

List of integers, bottom-up

Definition

The set List-of-Int is the smallest set of Python lists satisfying the following two properties:

1. [] $\in$ List-of-Int, and
2. if $n \in$ Int and $\ell \in$ List-of-Int, then $[n]+\ell \in$ List-of-Int.

Rules of inference:

1. $$\frac{}{[\,] \in \text{List-of-Int}}$$
2. $$\frac{n \in \text{Int} \qquad \ell \in \text{List-of-Int}}{[n]+\ell \in \text{List-of-Int}}$$

Chain of reasoning

Example

$$\dfrac{\text{-7} \in \mathsf{Int} \qquad \dfrac{3 \in \mathsf{Int} \qquad \dfrac{14 \in \mathsf{Int} \qquad [\,] \in \mathsf{List\text{-}of\text{-}Int}}{[14] + [\,] \in \mathsf{List\text{-}of\text{-}Int}}}{[3] + ([14] + [\,]) \in \mathsf{List\text{-}of\text{-}Int}}}{[\text{-}7] + ([3] + ([14] + [\,])) \in \mathsf{List\text{-}of\text{-}Int}}$$

Remark

[-7] $+$ *(*[3]* $+$ *(*[14]* $+$ *[]))* $=$ *[-7,3,14]*

List of integers using grammars

Definition

List-of-Int ::= []
List-of-Int ::= [Int] + List-of-Int

- **Nonterminal symbols** are the names of the sets being defined.
- **Terminal symbols** are the characters in the external representation, e.g., [,], +, etc.
- **Productions** are the *rules*. Each production has a left-hand side, which is a nonterminal symbol, and a right-hand side, which consists of terminal and nonterminal symbols. The left- and right-hand sides are usually separated by the symbol ::=, read *is* or *can be*.

Alternative forms

Definition

$$\begin{aligned} \text{List-of-Int} &::= [] \\ &::= [\text{Int}] + \text{List-of-Int} \end{aligned}$$

Definition

$\text{List-of-Int} ::= [] \mid [\text{Int}] + \text{List-of-Int}$

Definition

$\text{List-of-Int} ::= [\{\text{Int}\}^{*(,)}]$

- The notation $\{\ldots\}^{*(c)}$ is *Kleene star*. When this appears in a right-hand side, it indicates a sequence of any number of instances of whatever appears between the braces, separated by the nonempty character sequence c.
- A variant of the star notation is *Kleene plus* $\{\ldots\}^{+(c)}$, which indicates a sequence of one or more instances.

Exercise (in class)

Write a derivation from List-of-Int to [-7] + ([3] + ([14] + [])).

Binary tree using grammars

Definition (non-Python)

Bintree ::= Int | (Symbol Bintree Bintree)

Example

1
2
(foo 1 2)
(bar 1 (foo 1 2))
(baz (bar 1 (foo 1 2)) (biz 4 5))

Lambda expression using grammars

Definition (non-Python)

LcExp	::=	Identifier
	::=	(`lambda` (Identifier) LcExp)
	::=	(LcExp LcExp)

Example

(lambda (x) (+ x 5))
((lambda (x) (+ x 5)) (- x 7))

How do we do it in Python?

How do we do it in Python?

Will come back to it later in the semester.

Induction

Theorem

Let t be a binary tree. Then t contains an odd number of nodes.

Induction

Theorem

Let t be a binary tree. Then t contains an odd number of nodes.

Proof.

By induction on the size of `t`. The induction hypothesis, $\mathrm{IH}(k)$, is that any tree of size $\leq k$ has an odd number of nodes.

1. There are no trees with 0 nodes, so $\mathrm{IH}(0)$ holds trivially.
2. Let k be an integer such that $\mathrm{IH}(k)$ holds. If `t` has $\leq k+1$ nodes, there are exactly two possibilities:
    1. `t` could be of the form n, where n is an integer. In this case, `t` has exactly one node, and one is odd.
    2. `t` could be of the form `(sym t1 t2)`, where `sym` is a symbol, and `t1` and `t2` are trees. Since `t` has $\leq k+1$ nodes, `t1` and `t2` must have $\leq k$ nodes. Therefore by $\mathrm{IH}(k)$, they must each have an odd number of nodes, say $2n_1+1$ and $2n_2+1$ nodes, respectively. Hence the total number of nodes in the tree is $2(n_1+n_2+1)+1$, which is again odd.

□

Proof by structural induction

To prove that a proposition IH(s) is true for all structures s, prove the following:

1. IH is true on simple structures (those without substructures).
2. If IH is true on the substructures of s, then it is true on s itself.

The smaller-subproblem principle

If we can reduce a problem to a smaller subproblem, we can call the procedure that solves the problem to solve the subproblem.

Derivation of function "list length"

```
from car_cdr import *

def list_length(lst):
    '''
    list_length(lst) = the length of lst
    '''
    if lst == []:
        return 0
    else:
        return 1 + list_length(cdr(lst))
```

Example computation of list length

```
  list_length([a, [b, c] d])
= 1 + list_length([[b, c], d])
= 1 + (1 + list_length([d]))
= 1 + (1 + (1 + list_length([])))
= 1 + (1 + (1 + 0))
= 3
```

Derivation of function "n-th element"

```
from car_cdr import *

def nth_element(lst, n):
  '''
  nth_element(lst, n) = the n-th element of lst
  '''
  if lst == []:
    return report_list_too_short(n)
  if n == 0:
    return car(lst)
  else:
    return nth_element(cdr(lst), n - 1)

def report_list_too_short(n):
  print "List␣too␣short␣by␣%d␣elements." % (n + 1)
  return None
```

Example computation of n-th element

```
  nth_element([a, b, c, d, e], 3)
= nth_element([b, c, d, e], 2)
= nth_element([c, d, e], 1)
= nth_element([d, e], 0)
= d
```