Inductions and Strings

Mark Greenstreet, CpSc 421, Term 1, 2008/09

Lecture Outline

Mathematical background for the “Theory of Computing”

- Induction
- Strings
- An Example

Axioms for the Natural Numbers

Axiom 0: 0 is a natural number.

Axiom 1: if x is a natural number, so is $succ(x)$

Axiom 2: if x is a natural number, $succ(x) > x$.

Axiom 3: if x and y are natural numbers and $x > y$, then $succ(x) > y$.

Axiom 4: if x and y are natural numbers and $x > y$, then $x \neq y$.

We write $\mathbb{N}$ to denote the set of natural numbers.

Operations on the Natural Numbers

- Addition:

$$\begin{aligned} x + 0 &= x, \\ x + succ(y) &= succ(x + y). \end{aligned}$$

- Multiplication:

$$\begin{aligned} x * 0 &= 0, \\ x * succ(y) &= (x * y) + x. \end{aligned}$$

Two More Operations

- Division:

$$(x/y) = q \quad \Leftrightarrow \quad y * q = x.$$

- Exponentiation:

$$\begin{aligned} x^0 &= succ(0), \\ x^{succ(y)} &= (x^y) * x. \end{aligned}$$

Abbreviations

- Decimal digits:

$$1 = succ(0),\quad 2 = succ(1),\quad 3 = succ(2),\quad 4 = succ(3),$$
$$5 = succ(4),\quad 6 = succ(5),\quad 7 = succ(6),\quad 8 = succ(7),$$
$$9 = succ(8),\quad 10 = succ(9).$$

- Multidigit numbers:

$$1437 = 1*10^3 + 4*10^2 + 3*10^1 + 7*10^0$$
$$= \underbrace{succ(succ(succ(\ldots(succ(}_{1437\text{ ``succ(''s}}0\underbrace{))\ldots)))}_{1437\text{ ``)''s}}$$

0 is the primitive element for the naturals.

Lazy Proofs

To prove: For all natural numbers, n, $\sum_{k=0}^{n} k \;=\; \frac{n^2 + n}{2}$.

Strategy:

- Wait for you to propose a particular m.
- Ask you to prove that m is a natural number. You'll have to me you that

$$m \quad = \quad succ(succ(succ(\ldots succ(0) \ldots))).$$

- I'll Prove that the formula holds for $m = 0$.
- For each $succ$ in the formula for m, I'll show that the formula for the sum holds.

Visualize Laziness

If you show me:

m = succ(
succ(
succ(
⋮ ⋮
succ(
succ(
succ(
0)))...)))

then, I'll show you:

proof for m = succ(succ(succ(... succ(succ(succ(0)))...)))
proof for m = succ(succ(... succ(succ(succ(0)))...))
proof for m = succ(... succ(succ(succ(0)))...)
⋮
proof for m = succ(succ(succ(0)))
proof for m = succ(succ(0))
proof for m = succ(0)
proof for m = 0

Proof for $m = 0$

- $\sum_{k=0}^{0} k = 0$.
- $$\begin{aligned} \frac{0^2+0}{2} &= \frac{0^2}{2}, && \text{def. } + \\ &= \frac{0^{succ(succ(0))}}{2}, && \text{def. 2} \\ &= \frac{(0*0)*0}{2}, && \text{def. exponentiation} \\ &= \frac{0}{2}, && \text{def. multiplication} \\ &= 0, && 2*0=0, \text{ def. division} \end{aligned}$$
- $\square$

Proof for $succ(m)$

$$
\begin{aligned}
& \frac{succ(m)^2 + succ(m)}{2} & \\
= \quad & \tfrac{(m+1)^2+(m+1)}{2}, & m + 1 = succ(m) \\
= \quad & \tfrac{(m^2+2*m+1)+(m+1)}{2}, & \text{algebra} \\
= \quad & \tfrac{(m^2+m)+2*(m+1)}{2}, & \text{more algebra} \\
= \quad & \tfrac{m^2+m}{2} + \tfrac{2*(m+1)}{2}, & \text{more algebra} \\
= \quad & \tfrac{m^2+m}{2} + (m+1), \text{ def. division} & \\
= \quad & \left(\sum_{k=0}^{m} k\right) + (m+1), & \text{already shown: } \sum_{k=0}^{m} k = \frac{k^2+k}{2} \\
= \quad & \sum_{k=0}^{succ(m)} k, & \text{def. summation}
\end{aligned}
$$

Inductive Definitions

- Induction applies when the domain of interest is defined inductively.
- An inductive definition consists of a collection cases:
  - Primitive elements. We can write these cases as:

    $$s_0 \in S$$

    For example, $0 \in \mathbb{N}$.
  - Inductive cases that build larger elements from smaller ones. We can write:

    $$\forall s_1, s_2, \ldots s_k \in S.\ C(s_1, s_2, \ldots s_k) \in S$$

    For example, $\forall x \in \mathbb{N}.\ succ(x) \in \mathbb{N}$.

Proof by Induction

If S is a set that is defined inductively, and $P : S \rightarrow \{0, 1\}$ is a predicate over elements of S, then we can prove that P holds for all elements of S by showing

- For each primitive element, s_0, of S show that $P(s_0)$ is true.
- For each inductive case, show that for any non-primitive element of s, you can find s_1, s_2, $\ldots s_k$ such that $s = C(s_1, s_2, \ldots s_k)$, and that

$$(P(s_1) \wedge P(s_2) \wedge \ldots \wedge P(s_k)) \Rightarrow P(s)$$

Strong Induction

- Let $\mathcal{S}$ be the set such that $x \in \mathcal{S}$ iff
  - $x = 0$, or
  - $x = 1$, or
  - there are y and z in $\mathcal{S}$ such that $x = y + z$.

  It is straightforward to show that $S = \mathbb{N}$, the natural numbers as defined on slide 3.
- Proof by strong induction.
  To prove that $P(n)$ holds for all natural number, n, show:
  - $P(0)$, and
  - $P(1)$, and
  - for any natural number $x > 1$, we can find natural numbers $y < x$ and $z < x$ such that $x = y + z$, and $(P(y) \wedge P(z)) \implies P(x)$.
- There are many more ways we could generate the integers, and each leads to its own template for induction proofs.

Strings

Let Σ be a finite set of "symbols".

- Informal definition: a string is a sequence of zero or more elements from Σ.
- Inductive definition: $s \in \Sigma^*$ iff
  - $s = \epsilon$, the empty string.
  - There is a $w \in \Sigma^*$ and a $c \in \Sigma$ such that $s = w \cdot c$.
- Note: The operator $\cdot$ represents concatenation, and we often omit writing it, just like skipping the $*$ for multiplication.

Operations on Strings:

- String concatenation:

$$\begin{aligned} x \cdot \epsilon &= x \\ x \cdot (y \cdot c) &= (x \cdot y) \cdot c \end{aligned}$$

- Length:

$$\begin{aligned} length(\epsilon) &= 0 \\ length(w \cdot c) &= length(w) + 1 \end{aligned}$$

  We write $|w|$ as a shorthand for $length(w)$.

- Equality:

$$\begin{aligned} x = y \quad \leftrightarrow \quad & (x = \epsilon) \land (y = \epsilon) \\ \lor \quad & (x = u \cdot c) \land (y = v \cdot d) \land (u = v) \land (c = d) \end{aligned}$$

One More Operation:

- Ordering:

$$
\begin{aligned}
x < y \quad \leftrightarrow \quad & \qquad length(x) < length(y) \\
& \vee \quad (length(x) = length(y)) \wedge (x = c \cdot u) \\
& \qquad \wedge (y = d \cdot v) \wedge (c < d) \\
& \vee \quad (length(x) = length(y)) \wedge (x = c \cdot u) \\
& \qquad \wedge (y = c \cdot v) \wedge (u < v)
\end{aligned}
$$

Note that “zymurgy” $<$ “aardvark” by this ordering.

Putting it all together

- Let $\Sigma = \{0, 1\}$.
- Let $S \subseteq \Sigma^*$, such that w is in S iff
  - $w = \epsilon$; or
  - There is a string x in S such that $w = 0x1$ or $w = 1x0$; or
  - There are strings x and y in S with $x \neq \epsilon$ and $y \neq \epsilon$ such that $w = xy$.
- Prove that w is in S iff the number of 0's in w is equal to the number of 1's.

Proof strategy

- First show that if $w \in S$, then w has an equal number of `0`'s and `1`'s.
- Next, show that if w has an equal number of `0`'s and `1`'s, then $w \in S$.

Proof strategy

- First show that if $w \in S$, then w has an equal number of `0`'s and `1`'s.
  - Here, we consider each of the three rules for a string being in S.
  - We will show that each rule produces a string with an equal number of `0`'s and `1`'s.
  - This is a proof by induction according to the inductive definition of S.
- Next, show that if w has an equal number of `0`'s and `1`'s, then $w \in S$.

$(w \in S) \Rightarrow (\#0(w) = \#1(w))$

Let w be an arbitrary element of S. Appling induction over the definition of S we get:

- case $w = \epsilon$: $\#0(w) = \#1(w) = 0$. ✓
- case $\exists x \in S.(w = \texttt{0}x\texttt{1}) \vee (w = \texttt{1}x\texttt{0})$:
  1. Because $x \in S$, the induction hypothesis holds for x and $\#0(x) = \#1(x)$.
  2. If $w = \texttt{0}x\texttt{1}$, then $\#0(w) = \#0(x) + 1$, and $\#1(w) = \#1(x) + 1$.
  3. From (1) and (2), $\#1(w) = \#0(w)$. ✓
- case $\exists x, y \in S.w = xy$:

$$
\begin{aligned}
\#0(w) &= \#0(x) + \#0(y), && w = xy \\
&= \#1(x) + \#1(y), && x, y \in S, |x| < |w|, |y| < |w|, \text{ind. hyp.} \\
&= \#1(w), && w = xy
\end{aligned}
$$
✓

Proof strategy (revisited)

- We have just shown that if $w \in S$, then w has an equal number of `0`'s and `1`'s.
- We will now show that if w has an equal number of `0`'s and `1`'s. then $w \in S$.
  - The basic idea is that for any string with an equal number of we'll find a way to apply the rules defining S to show that it is in S.
  - We consider the empty string and then strings of the form $u \cdot c$. Thus, this is an inductive proof according to the definition of strings.
  - The tricky part is that if $w = u \cdot c$ has an equal number of `0`'s and `1`'s, then u definitely does not!

    To handle this, we'll introduce a function that keeps track of the difference between the number of `0`'s and `1`'s.

$(w \in S) \Leftarrow (\#0(w) = \#1(w))$

- case $w = \epsilon$: $w \in S$ by the first rule in the definition of S. ✓
- case $w = v \cdot c$ for some $c \in \Sigma$:

1. Assume $c =$ `0` (the other case is equivalent).
2. By definition $\#0(w) = \#0(v) + 1 \geq 1$, and $\#1(w) = \#1(v)$.
3. By the assumption that $\#0(w) = \#1(w)$, we get $\#1(v) \geq 1$, and thus $|v| \geq 1$.
4. Let $v = d \cdot u$ for some $d \in \Sigma$.
5. If $d =$ `1`, then
   a. $w = 1 \cdot u \cdot 0$; $\#0(u) = \#0(w) - 1$; and $\#1(u) = \#1(w) - 1$.
   b. Thus, $u \in S$ by the induction hypothesis.
   c. $w \in S$ by the second rule in the definition of S. ✓
6. If $d =$ `0`, then we go to the next slide

Defining f

$$
\begin{aligned}
f(\epsilon) &= 0 \\
f(s \cdot \mathtt{0}) &= f(s) - 1 \\
f(s \cdot \mathtt{1}) &= f(s) + 1
\end{aligned}
$$

Observations about f:

- $\#0(w) = \#1(w)$ iff $f(w) = 0$.
- $f(xy) = f(x) + f(y)$.
- If $f(s) > 0$ then for all $k \in [0 \ldots f(s)]$, s can be divided into two strings, x, and y, such that $s = xy$ and $f(x) = k$.
- If $f(s) < 0$ then for all $k \in [f(s) \ldots 0]$, s can be divided into two strings, x, and y, such that $s = xy$ and $f(x) = k$.

$(w \in S) \Leftarrow (\#0(w) = \#1(w))$ **(cont.)**

6. If $d = 0$, then $w = 0 \cdot u \cdot 0$.

   a. $f(w) = 0$, because $\#0(w) = \#1(w)$.

   b. $f(u) = 2$, because $(-1) + f(u) + (-1) = 0$.

   c. From the third observation on the previous slide, we conclude there are strings x and y such that $u = xy$ and $f(x) = 1$.

   d. By the construction x and y, we conclude $w = 0 \cdot x \cdot y \cdot 0$.

   e. From the second observation on the previous slide, we conclude $f(0 \cdot x) = 0$,and $f(y \cdot 0) = 0$.

   f. From the first observation on the previous slide, we conclude $\#0(0x) = \#1(0x)$,and $\#0(y0) = \#1(y0)$.

   g. The induction hypothesis yields: $0 \cdot x \in S$, and $y \cdot 0 \in S$.

   h. The third rule in the definition of S yields: $(0 \cdot x) \cdot (y \cdot 0) \in S$.

   i. Thus, $w \in S$ (see step d). $\checkmark$

Tuple-Terror

In this class, we will often get definitions along the lines of:

> A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
>
> 1. Q is a finite set called the states.
> 2. $\ldots$
>
> (From *Sipser*, Def. 1.5, p. 35)

"Tuples" are the mathematicians way of describing things that resemble what programmers call "data structures."

Type and Sets

- Programming language types correspond to sets.
- The java type boolean corresponds to the set $\{\text{true}, \text{false}\}$. A variable of type boolean can have either value from this set.
- A Java int corresponds to the set $[-2^{31}, \ldots 2^{31} - 1]$. A variable of type int can have any value from this set.
- A Java class corresponds to the set that is the cross-product of the sets for each of its fields (see note on slide 27). Let's do an example to see how this works.

class CourseSection (java version)

```
class CourseSection {
    String instructor;      // who teaches the class
    Set<int> students;      // who is taking the class (student #s)
    Department d;           // the department offering this course
    int courseNum;          // the course number
    int sectionNum;         // the section number

    . . .                   // constructors, methods, etc.
}
```

class CourseSection (tuple version)

In *Sipser*, a course section would be describe as:

A course section is a 5-tuple (I, S, d, c, x) where

1. $I \in \Sigma^*$ is the instructor of the course.
2. S is a set of integers, the student numbers of the students in the course.
3. $\ldots$

Notes:

- a tuple is an element of the set that is formed by the cross-product of the sets for each of its elements. This means that you can put a tuple together by choosing any value you like for each element from the corresponding set.
- In Java (and other programming languages), we'll often restrict this. The constructor for a class may insure that some relationship holds for the members of the class, and all methods of the class may preserve this relationship. These are the data invariants that you've seen (I assume) in earlier classes, but we won't go further into that (at least not now).

The coming week

Reading:

September 8 (Monday): *Sipser* 1.1.
Lecture will cover through Example 1.15 (i.e. pages 31–40).

September 10 (Wednesday): *Sipser* 1.1 (continued).
Lecture will cover the rest of the section (i.e. pages 40–47).

September 12 (Friday): *Sipser* 1.2.
Lecture will cover through Example 1.35 (i.e. pages 47–52).

Homework:

September 5 (today): Homework 0 goes out (due Sept. 12).

September 12 (Friday): Homework 0 due. Homework 1 goes out (due Sept. 19).

Have a good weekend!