CIS 500

Software Foundations

Fall 2002

16 September

Administrivia

- Recitations begin (in fact, have already begun!) this week.
- Section E (Tue 3:00 – 4:30) will meet in the IRCS "back conference room," room 413 in the 3401 Walnut building (above Starbucks). Check the course web page for detailed directions.
- If you are not sure which recitation you are in, email `bracy@gradient`.

Plan for this week

We've seen enough of OCaml for now. (If time permits, we'll come back later to some more advanced programming idioms later in the course.) Time to get started with the book.

General plan:

- discuss material in class [This week: induction, basic operational semantics]
- read in book [Chapters 3 and 4]
- homework assignment [covering chapters 3 and 4; handed out Thu, due next Thu]
- discuss in recitation
- finish off homework assignment

Reading Assignment

ASAP: Read Preface and Chapter 1 and skim Chapter 2 of TAPL on your own.

By Thursday or Friday: Read Chapters 3 and 4.

Review

Induction

Principle of ordinary induction on natural numbers

Suppose that P is a predicate on the natural numbers. Then:
If $P(0)$
and, for all i, $P(i)$ implies $P(i+1)$,
then $P(n)$ holds for all n.

Example

Theorem: $2^0 + 2^1 + ... + 2^n = 2^{n+1} - 1$, for every n.

Proof:

- Let $P(i)$ be "$2^0 + 2^1 + ... + 2^i = 2^{i+1} - 1$."
- Show $P(0)$:

$$2^0 = 1 = 2^1 - 1$$

- Show that $P(i)$ implies $P(i+1)$:

$$\begin{aligned} 2^0 + 2^1 + ... + 2^{i+1} &= (2^0 + 2^1 + ... + 2^i) + 2^{i+1} \\ &= (2^{i+1} - 1) + 2^{i+1} && \text{IH} \\ &= 2 \cdot (2^{i+1}) - 1 \\ &= 2^{i+2} - 1 \end{aligned}$$

- The result ($P(n)$ for all n) follows by the principle of induction.

Shorthand form

Theorem: $2^0 + 2^1 + ... + 2^n = 2^{n+1} - 1$, for every n.

Proof: By induction on n.

- Base case ($n = 0$):

$$2^0 = 1 = 2^1 - 1$$

- Inductive case ($n = i + 1$):

$$\begin{aligned} 2^0 + 2^1 + ... + 2^{i+1} &= (2^0 + 2^1 + ... + 2^i) + 2^{i+1} \\ &= (2^{i+1} - 1) + 2^{i+1} && \text{IH} \\ &= 2 \cdot (2^{i+1}) - 1 \\ &= 2^{i+2} - 1 \end{aligned}$$

Complete Induction

Principle of complete induction on natural numbers

Suppose that P is a predicate on the natural numbers. Then:
If, for each natural number n,
given $P(i)$ for all $i < n$
we can show $P(n)$,
then $P(n)$ holds for all n.

Ordinary and complete induction are interderivable — assuming one, we can prove the other.

Thus, the choice of which to use for a particular proof is purely a question of style.

We'll see some other (equivalent) styles as we go along.

Syntax

Simple Arithmetic Expressions

Here is a BNF grammar for a very simple language of arithmetic expressions:

`t` ::=		terms:
	`true`	constant true
	`false`	constant false
	`if t then t else t`	conditional
	`0`	constant zero
	`succ t`	successor
	`pred t`	predecessor
	`iszero t`	zero test

Terminology:

- `t` here is a metavariable

Abstract vs. concrete syntax

Q1: Does this grammar define a set of character strings, a set of token lists, or a set of abstract syntax trees?

Abstract vs. concrete syntax

Q1: Does this grammar define a set of character strings, a set of token lists, or a set of abstract syntax trees?

A: In a sense, all three. But we are primarily interested, here, in abstract syntax trees.

For this reason, grammars like the one on the previous slide are sometimes called abstract grammars. An abstract grammar defines a set of abstract syntax trees and suggests a mapping from character strings to trees.

We then write terms as character strings rather than trees simply for convenience. If there is any potential confusion about what tree is intended, we use parens to disambiguate.

Q: So, are

```
succ 0
succ (0)
(((succ (((((0))))))))
```

"the same term"?

What about

```
succ 0
pred (succ (succ 0))
```

?

A more explicit form of the definition

The set of terms is the smallest set $\mathcal{T}$ such that

1. $\{\texttt{true}, \texttt{false}, \texttt{0}\} \subseteq \mathcal{T}$;
2. if $\texttt{t}_1 \in \mathcal{T}$, then $\{\texttt{succ t}_1, \texttt{pred t}_1, \texttt{iszero t}_1\} \subseteq \mathcal{T}$;
3. if $\texttt{t}_1 \in \mathcal{T}$, $\texttt{t}_2 \in \mathcal{T}$, and $\texttt{t}_3 \in \mathcal{T}$, then $\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \in \mathcal{T}$.

Inference rules

An alternate notation for the same definition:

$$\texttt{true} \in \mathcal{T} \qquad \texttt{false} \in \mathcal{T} \qquad \texttt{0} \in \mathcal{T}$$

$$\frac{\texttt{t}_1 \in \mathcal{T}}{\texttt{succ t}_1 \in \mathcal{T}} \qquad \frac{\texttt{t}_1 \in \mathcal{T}}{\texttt{pred t}_1 \in \mathcal{T}} \qquad \frac{\texttt{t}_1 \in \mathcal{T}}{\texttt{iszero t}_1 \in \mathcal{T}}$$

$$\frac{\texttt{t}_1 \in \mathcal{T} \quad \texttt{t}_2 \in \mathcal{T} \quad \texttt{t}_3 \in \mathcal{T}}{\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \in \mathcal{T}}$$

Note that "the smallest set closed under..." is implied (but often not stated explicitly).

Terminology:

- axiom vs. rule
- concrete rule vs. rule scheme

Terms, concretely

For each natural number i, define a set $\mathcal{S}_i$ as follows:

$$\begin{array}{rcl} \mathcal{S}_0 & = & \emptyset \\ \mathcal{S}_{i+1} & = & \{\texttt{true}, \texttt{false}, \texttt{0}\} \\ & & \cup \ \{\texttt{succ t}_1, \texttt{pred t}_1, \texttt{iszero t}_1 \mid \texttt{t}_1 \in \mathcal{S}_i\} \\ & & \cup \ \{\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \mid \texttt{t}_1, \texttt{t}_2, \texttt{t}_3 \in \mathcal{S}_i\} \end{array}$$

Now let

$$\mathcal{S} = \bigcup_i \mathcal{S}_i$$

Equivalence of the definitions

We have seen two basic presentations of terms:

1. inductively ($\mathcal{T}$) as the smallest set closed under certain rules
   - explicit inductive definition
   - BNF shorthand
   - inference rule shorthand
2. concretely ($\mathcal{S}$) as the limit of a series of sets (of larger and larger terms)

Are these presentations equivalent? I.e., do we have $\mathcal{T} = \mathcal{S}$?

Inductive Function Definitions

The set of constants appearing in a term $\texttt{t}$, written $Consts(\texttt{t})$, is defined as follows:

$$\begin{array}{lcl} Consts(\texttt{true}) & = & \{\texttt{true}\} \\ Consts(\texttt{false}) & = & \{\texttt{false}\} \\ Consts(\texttt{0}) & = & \{\texttt{0}\} \\ Consts(\texttt{succ t}_1) & = & Consts(\texttt{t}_1) \\ Consts(\texttt{pred t}_1) & = & Consts(\texttt{t}_1) \\ Consts(\texttt{iszero t}_1) & = & Consts(\texttt{t}_1) \\ Consts(\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3) & = & Consts(\texttt{t}_1) \cup Consts(\texttt{t}_2) \cup Consts(\texttt{t}_3) \end{array}$$

Simple, right?

First question: In what sense is this a "definition"?

(Normally, a "definition" just assigns a convenient name to a previously-known thing. But here, the "thing" on the right-hand side involves the very name that we are "defining"!)

Second question: Suppose we had written this instead...

The set of constants appearing in a term `t`, written BadConsts(`t`), is defined as follows:

$$
\begin{array}{lcl}
\mathsf{BadConsts}(\texttt{true}) & = & \{\texttt{true}\} \\
\mathsf{BadConsts}(\texttt{false}) & = & \{\texttt{false}\} \\
\mathsf{BadConsts}(\texttt{0}) & = & \{\texttt{0}\} \\
\mathsf{BadConsts}(\texttt{0}) & = & \{\} \\
\mathsf{BadConsts}(\texttt{succ } \texttt{t}_1) & = & \mathsf{BadConsts}(\texttt{t}_1) \\
\mathsf{BadConsts}(\texttt{pred } \texttt{t}_1) & = & \mathsf{BadConsts}(\texttt{t}_1) \\
\mathsf{BadConsts}(\texttt{iszero } \texttt{t}_1) & = & \mathsf{BadConsts}(\texttt{iszero (iszero } \texttt{t}_1\texttt{))}
\end{array}
$$

What is the essential difference between these two definitions? How do we tell the difference between well-formed inductive definitions and ill-formed ones?

What, exactly, does a well-formed inductive definition mean?

First, recall that a function is just a two-place relation with certain properties:

- It is total: every element of its domain occurs at least once in its "graph"
- It is deterministic: every element of its domain occurs at mostd once in its graph.

We have seen how to define relations inductively. E.g....

Let Consts be the smallest two-place relation closed under the following rules:

$$(\texttt{true}, \{\texttt{true}\}) \in \textsf{Consts}$$

$$(\texttt{false}, \{\texttt{false}\}) \in \textsf{Consts}$$

$$(\texttt{0}, \{\texttt{0}\}) \in \textsf{Consts}$$

$$\frac{(\texttt{t}_1, \mathbf{C}) \in \textsf{Consts}}{(\texttt{succ t}_1, \mathbf{C}) \in \textsf{Consts}}$$

$$\frac{(\texttt{t}_1, \mathbf{C}) \in \textsf{Consts}}{(\texttt{pred t}_1, \mathbf{C}) \in \textsf{Consts}}$$

$$\frac{(\texttt{t}_1, \mathbf{C}) \in \textsf{Consts}}{(\texttt{iszero t}_1, \mathbf{C}) \in \textsf{Consts}}$$

$$\frac{(\texttt{t}_1, \mathbf{C}_1) \in \textsf{Consts} \qquad (\texttt{t}_2, \mathbf{C}_2) \in \textsf{Consts} \qquad (\texttt{t}_3, \mathbf{C}_3) \in \textsf{Consts}}{(\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3, (\textsf{Consts}(\texttt{t}_1) \cup \textsf{Consts}(\texttt{t}_2) \cup \textsf{Consts}(\texttt{t}_3))) \in \textsf{Consts}}$$

This definition certainly defines a relation (i.e., the smallest one with a certain property). How can we tell that this relation is a function?

This definition certainly defines a relation (i.e., the smallest one with a certain property). How can we tell that this relation is a function?

Prove it!

Theorem: The relation Consts defined by the inference rules a couple of slides ago is total and deterministic.

Proof: (Exercise.)

How about the bad definition?

$$(\texttt{true}, \{\texttt{true}\}) \in \text{BadConsts}$$

$$(\texttt{false}, \{\texttt{false}\}) \in \text{BadConsts}$$

$$(\texttt{0}, \{\texttt{0}\}) \in \text{BadConsts}$$

$$(\texttt{0}, \{\}) \in \text{BadConsts}$$

$$\frac{(\texttt{t}_1, \mathbf{C}) \in \text{BadConsts}}{(\texttt{succ t}_1, \mathbf{C}) \in \text{BadConsts}}$$

$$\frac{(\texttt{t}_1, \mathbf{C}) \in \text{BadConsts}}{(\texttt{pred t}_1, \mathbf{C}) \in \text{BadConsts}}$$

$$\frac{(\texttt{iszero (iszero t}_1\texttt{)}, \mathbf{C}) \in \text{BadConsts}}{(\texttt{iszero t}_1, \mathbf{C}) \in \text{BadConsts}}$$

Now, this set of rules does define a perfectly good relation — it's just that this relation does not happen to be a function!

Just for fun, let's calculate some cases of this relation...

- For what values of $\mathbf{C}$ do we have $(\texttt{false}, \mathbf{C}) \in$ Consts?
- For what values of $\mathbf{C}$ do we have $(\texttt{succ 0}, \mathbf{C}) \in$ Consts?
- For what values of $\mathbf{C}$ do we have $(\texttt{if false then 0 else 0}, \mathbf{C}) \in$ Consts?
- For what values of $\mathbf{C}$ do we have $(\texttt{iszero 0}, \mathbf{C}) \in$ Consts?

$$\boxed{\text{Operational Semantics}}$$

[Informal definition and examples; intuition of abstract machines]

[values]

[concrete definitions for booleans]

[Proof trees

Inference rules as an inductive definition of valid proof trees (vs. as a definition of a relation)]

[Normal forms.

Theorem: normal forms = values.]

[Multi-step evaluation.]