CIS 500

Software Foundations

Fall 2003

15 September

Administrivia

- If you still want to join a study group, send mail to `cis500@seas`. We currently have one pending request.
- Sign-up sheets for recitations are being passed around the room. Please write down your name and recitation preferences. We will announce assignments as soon as possible.
- HW3 is available now; due next Monday.
  - Can be handed in either physically or electronically
  - If you foresee a lot of technical writing in your future, you should consider learning how to use Latex now.
  - Please look at (at least) the first two problems from HW3 before coming to this week's recitation.

Where we're going

Going Meta...

The functional programming style used in OCaml is based on treating programs as data — i.e., on writing functions that manipulate other functions as their inputs and outputs.

Going Meta...

The functional programming style used in OCaml is based on treating programs as data — i.e., on writing functions that manipulate other functions as their inputs and outputs.

Everything in this course is based on treating programs as mathematical objects — i.e., we will be building mathematical theories whose basic objects of study are programs (and whole programming languages).

Going Meta...

The functional programming style used in OCaml is based on treating programs as data — i.e., on writing functions that manipulate other functions as their inputs and outputs.

Everything in this course is based on treating programs as mathematical objects — i.e., we will be building mathematical theories whose basic objects of study are programs (and whole programming languages).

Jargon: We will be studying the metatheory of programming languages.

Warning!

The material in the next couple of lectures is more slippery than it may first appear.

"I believe it when I hear it" is not a sufficient test of understanding.

A much better test is "I can explain it so that someone else believes it."

Basics of Induction (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 $P(i+1)$:

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

- 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 linear character strings rather than trees simply for convenience. If there is any potential confusion about what tree is intended, we use parentheses 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 $\mathcal{T}$ of terms is the smallest set such that

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

Inference rules

An alternate notation for the same definition:

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

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

$$\frac{\mathtt{t}_1 \in \mathcal{T} \qquad \mathtt{t}_2 \in \mathcal{T} \qquad \mathtt{t}_3 \in \mathcal{T}}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \in \mathcal{T}}$$

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

Define an infinite sequence of sets, $\mathcal{S}_0$, $\mathcal{S}_1$, $\mathcal{S}_2$, ..., as follows:

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

Now let

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

Comparing the definitions

We have seen two different presentations of terms:

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

Comparing the definitions

We have seen two different presentations of terms:

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

What does it mean to assert that “these presentations are equivalent”?

Induction on Syntax

Why two definitions?

The two ways of defining the set of terms are both useful:

1. the definition of terms as the smallest set with a certain closure property is compact and easy to read
2. the definition of the set of terms as the limit of a sequence gives us an induction principle for proving things about terms...

Induction on Terms

Definition: The depth of a term $\mathtt{t}$ is the smallest i such that $\mathtt{t} \in \mathcal{S}_i$.

From the definition of $\mathcal{S}$, it is clear that, if a term $\mathtt{t}$ is in $\mathcal{S}_i$, then all of its immediate subterms must be in $\mathcal{S}_{i-1}$, i.e., they must have strictly smaller depths.

This observation justifies the principle of induction on terms.

Let $\mathbf{P}$ be a predicate on terms.

> If, for each term $\mathtt{s}$,
> given $\mathbf{P}(\mathtt{r})$ for all immediate subterms $\mathtt{r}$ of $\mathtt{s}$
> we can show $\mathbf{P}(\mathtt{s})$,
> then $\mathbf{P}(\mathtt{t})$ holds for all $\mathtt{t}$.

Inductive Function Definitions

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

$$
\begin{array}{lcl}
Consts(\mathtt{true}) & = & \{\mathtt{true}\} \\
Consts(\mathtt{false}) & = & \{\mathtt{false}\} \\
Consts(\mathtt{0}) & = & \{\mathtt{0}\} \\
Consts(\mathtt{succ}\ \mathtt{t_1}) & = & Consts(\mathtt{t_1}) \\
Consts(\mathtt{pred}\ \mathtt{t_1}) & = & Consts(\mathtt{t_1}) \\
Consts(\mathtt{iszero}\ \mathtt{t_1}) & = & Consts(\mathtt{t_1}) \\
Consts(\mathtt{if}\ \mathtt{t_1}\ \mathtt{then}\ \mathtt{t_2}\ \mathtt{else}\ \mathtt{t_3}) & = & Consts(\mathtt{t_1}) \cup Consts(\mathtt{t_2}) \cup Consts(\mathtt{t_3})
\end{array}
$$

Simple, right?

First question:

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”!

So in what sense is this a definition??

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

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

$$
\begin{array}{lcl}
\text{BadConsts}(\texttt{true}) & = & \{\texttt{true}\} \\
\text{BadConsts}(\texttt{false}) & = & \{\texttt{false}\} \\
\text{BadConsts}(\texttt{0}) & = & \{\texttt{0}\} \\
\text{BadConsts}(\texttt{0}) & = & \{\} \\
\text{BadConsts}(\texttt{succ } t_1) & = & \text{BadConsts}(t_1) \\
\text{BadConsts}(\texttt{pred } t_1) & = & \text{BadConsts}(t_1) \\
\text{BadConsts}(\texttt{iszero } t_1) & = & \text{BadConsts}(\texttt{iszero (iszero } t_1\texttt{)})
\end{array}
$$

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 can be viewed as a two-place relation (called the "graph" of the function) 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 most 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:

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

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

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

$$\frac{(t_1, \mathbf{C}) \in \text{Consts}}{(\texttt{succ } t_1, \mathbf{C}) \in \text{Consts}}$$

$$\frac{(t_1, \mathbf{C}) \in \text{Consts}}{(\texttt{pred } t_1, \mathbf{C}) \in \text{Consts}}$$

$$\frac{(t_1, \mathbf{C}) \in \text{Consts}}{(\texttt{iszero } t_1, \mathbf{C}) \in \text{Consts}}$$

$$\frac{(t_1, \mathbf{C}_1) \in \text{Consts} \qquad (t_2, \mathbf{C}_2) \in \text{Consts} \qquad (t_3, \mathbf{C}_3) \in \text{Consts}}{(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3, (\text{Consts}(t_1) \cup \text{Consts}(t_2) \cup \text{Consts}(t_3))) \in \text{Consts}}$$

This definition certainly defines a relation (i.e., the smallest one with a certain closure property).

Q: How can we be sure that this relation is a function?

This definition certainly defines a relation (i.e., the smallest one with a certain closure property).

Q: How can we be sure that this relation is a function?

A: Prove it!

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

I.e., for each term t there is exactly *one* set of terms C such that $(t, C) \in$ Consts.

Proof:

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

I.e., for each term t there is exactly *one* set of terms C such that $(t, C) \in$ Consts.

Proof: By induction on t.

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

I.e., for each term t there is exactly *one* set of terms C such that $(t, C) \in$ Consts.

Proof: By induction on t.

To apply the induction principle for terms, we must show, for an arbitrary term t, that if

> for each immediate subterm s of t, there is exactly *one* set of terms C_s such that $(s, C_s) \in$ Consts

then

> there is exactly *one* set of terms C such that $(t, C) \in$ Consts.

Proceed by cases on the form of t.

- If t is `0`, `true`, or `false`, then we can immediately see from the definition of *Consts* that there is exactly *one* set of terms C (namely $\{t\}$) such that $(t, C) \in Consts$.

Proceed by cases on the form of t.

- If t is `0`, `true`, or `false`, then we can immediately see from the definition of *Consts* that there is exactly *one* set of terms C (namely $\{t\}$) such that $(t, C) \in Consts$.

- If t is `succ` t_1, then the induction hypothesis tells us that there is exactly *one* set of terms C_1 such that $(t_1, C_1) \in Consts$. But then it is clear from the definition of *Consts* that there is exactly *one* set C (namely C_1) such that $(t, C) \in Consts$.

Proceed by cases on the form of t.

- If t is `0`, `true`, or `false`, then we can immediately see from the definition of *Consts* that there is exactly *one* set of terms C (namely $\{t\}$) such that $(t, C) \in Consts$.

- If t is `succ` t_1, then the induction hypothesis tells us that there is exactly *one* set of terms C_1 such that $(t_1, C_1) \in Consts$. But then it is clear from the definition of *Consts* that there is exactly *one* set C (namely C_1) such that $(t, C) \in Consts$.

  Similarly when t is `pred` t_1 or `iszero` t_1.

- If t is `if` s_1 `then` s_2 `else` s_3, then the induction hypothesis tells us
  - there is exactly *one* set of terms C_1 such that $(t_1, C_1) \in Consts$
  - there is exactly *one* set of terms C_2 such that $(t_2, C_3) \in Consts$
  - there is exactly *one* set of terms C_3 such that $(t_3, C_3) \in Consts$

  But then it is clear from the definition of *Consts* that there is exactly *one* set C (namely $C_1 \cup C_2 \cup C_3$) such that $(t, C) \in Consts$.

How about the bad definition?

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

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

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

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

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

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

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

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

Another Inductive Definition

$$
\begin{aligned}
\text{size}(\texttt{true}) &= 1\\
\text{size}(\texttt{false}) &= 1\\
\text{size}(\texttt{0}) &= 1\\
\text{size}(\texttt{succ t}_1) &= \text{size}(\texttt{t}_1) + 1\\
\text{size}(\texttt{pred t}_1) &= \text{size}(\texttt{t}_1) + 1\\
\text{size}(\texttt{iszero t}_1) &= \text{size}(\texttt{t}_1) + 1\\
\text{size}(\texttt{if t}_1\texttt{ then t}_2\texttt{ else t}_3) &= \text{size}(\texttt{t}_1) + \text{size}(\texttt{t}_2) + \text{size}(\texttt{t}_3) + 1
\end{aligned}
$$

Another proof by induction

Theorem: The number of distinct constants in a term is at most the size of the term. I.e., $|\text{Consts}(\texttt{t})| \leq \text{size}(\texttt{t})$.

Proof:

Another proof by induction

Theorem: The number of distinct constants in a term is at most the size of the term. I.e., $|\mathsf{Consts}(t)| \leq \mathsf{size}(t)$.

Proof: By induction on t.

Another proof by induction

Theorem: The number of distinct constants in a term is at most the size of the term. I.e., $|\mathsf{Consts}(t)| \leq \mathsf{size}(t)$.

Proof: By induction on t.

Assuming the desired property for immediate subterms of t, we must prove it for t itself.

Another proof by induction

Theorem: The number of distinct constants in a term is at most the size of the term. I.e., $|\mathsf{Consts}(t)| \leq \mathsf{size}(t)$.

Proof: By induction on t.

Assuming the desired property for immediate subterms of t, we must prove it for t itself.

There are three cases to consider:

Case: t is a constant

Immediate: $|\mathsf{Consts}(t)| = |\{t\}| = 1 = \mathsf{size}(t)$.

Another proof by induction

Theorem: The number of distinct constants in a term is at most the size of the term. I.e., $|\mathsf{Consts}(t)| \leq \mathsf{size}(t)$.

Proof: By induction on t.

Assuming the desired property for immediate subterms of t, we must prove it for t itself.

There are three cases to consider:

Case: t is a constant

Immediate: $|\mathsf{Consts}(t)| = |\{t\}| = 1 = \mathsf{size}(t)$.

Case: $t = \texttt{succ } t_1$, $\texttt{pred } t_1$, or $\texttt{iszero } t_1$

By the induction hypothesis, $|\mathsf{Consts}(t_1)| \leq \mathsf{size}(t_1)$. We now calculate as follows: $|\mathsf{Consts}(t)| = |\mathsf{Consts}(t_1)| \leq \mathsf{size}(t_1) < \mathsf{size}(t)$.

Case: $\mathtt{t = if\ t_1\ then\ t_2\ else\ t_3}$

By the induction hypothesis, $|Consts(t_1)| \leq size(t_1)$, $|Consts(t_2)| \leq size(t_2)$, and $|Consts(t_3)| \leq size(t_3)$. We now calculate as follows:

$$\begin{aligned} |Consts(t)| &= |Consts(t_1) \cup Consts(t_2) \cup Consts(t_3)| \\ &\leq |Consts(t_1)| + |Consts(t_2)| + |Consts(t_3)| \\ &\leq size(t_1) + size(t_2) + size(t_3) \\ &< size(t). \end{aligned}$$

Operational Semantics

Abstract Machines

An abstract machine consists of:

- a set of states
- a transition relation on states, written $\longrightarrow$

A state records all the information in the machine at a given moment. For example, an abstract-machine-style description of a conventional microprocessor would include the program counter, the contents of the registers, the contents of main memory, and the machine code program being executed.

For the very simple languages we are considering at the moment, however, the term being evaluated is the whole state of the abstract machine.

Nb. Often, the transition relation is actually a partial function: i.e., from a given state, there is at most one possible next state. But in general there may be many.

Operational semantics for Booleans

Syntax of terms and values

`t ::=`		terms
	`true`	constant true
	`false`	constant false
	`if t then t else t`	conditional
`v ::=`		values
	`true`	true value
	`false`	false value

The evaluation relation $t \longrightarrow t'$ is the smallest relation closed under the following rules:

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \qquad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad \text{(E-If)}$$

Terminology

Computation rules:

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \qquad \text{(E-IfFalse)}$$

Congruence rule:

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad \text{(E-If)}$$

Computation rules perform “real” computation steps.

Congruence rules determine where computation rules can be applied next.

Evaluation, more explicitly

$\longrightarrow$ is the smallest two-place relation closed under the following rules:

$$((\texttt{if true then } t_2 \texttt{ else } t_3), t_2) \in \longrightarrow$$

$$((\texttt{if false then } t_2 \texttt{ else } t_3), t_3) \in \longrightarrow$$

$$\frac{(t_1, t_1') \in \longrightarrow}{((\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3), (\texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3)) \in \longrightarrow}$$