

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.

CIS 500, 15 September

4-a

XIS 500, 15 September

4-b

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
(Re∨iew)

Induction

Principle of ordinary induction on natural numbers

Suppose that P is a predicate on the natural numbers. Then: If $P(\mathbf{0})$

and, for all i, P(i) implies P(i+1), then P(n) holds for all n.

CIS 500, 15 September

7

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): $2^{0} + 2^{1} + ... + 2^{i+1} = (2^{0} + 2^{1} + ... + 2^{i}) + 2^{i+1}$ $= (2^{i+1} - 1) + 2^{i+1}$ by IH $= 2 \cdot (2^{i+1}) - 1$ $= 2^{i+2} - 1$

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

XIS 500, 15 September

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):

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

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 < nwe can show P(n),

then P(n) holds for all n.

8

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.

OIS 500, 15 September

11

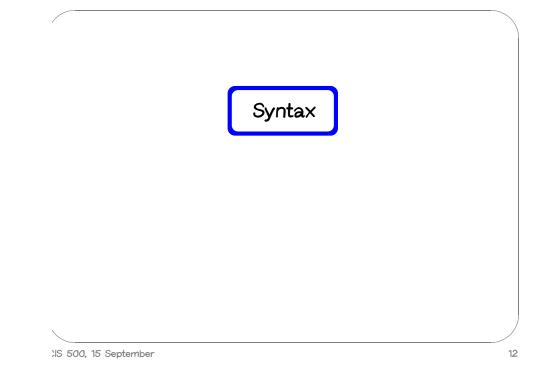
Simple Arithmetic Expressions

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.

OIS 500, 15 September

14-a

16

Q: So, are

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

"the same term"?

What about

succ 0
pred (succ (succ 0))

?

XIS 500, 15 September



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

1. {true, false, 0} $\subseteq \mathcal{T}$;

2. if $t_1 \in \mathcal{T}$, then {succ t_1 , pred t_1 , iszero t_1 } $\subseteq \mathcal{T}$;

3. if $t_1 \in \mathcal{T}$, $t_2 \in \mathcal{T}$, and $t_3 \in \mathcal{T}$, then if t_1 then t_2 else $t_3 \in \mathcal{T}$.

Inference rules An alternate notation for the same definition: true $\in \mathcal{T}$ false $\in \mathcal{T}$ $0 \in \mathcal{T}$ $t_1 \in \mathcal{T}$ false $\in \mathcal{T}$ $0 \in \mathcal{T}$ $t_1 \in \mathcal{T}$ false $\in \mathcal{T}$ $0 \in \mathcal{T}$ $t_1 \in \mathcal{T}$ false $\in \mathcal{T}$ $0 \in \mathcal{T}$ $t_1 \in \mathcal{T}$ false $\in \mathcal{T}$ $t_1 \in \mathcal{T}$ $succ t_1 \in \mathcal{T}$ $t_1 \in \mathcal{T}$ $t_2 \in \mathcal{T}$ $t_3 \in \mathcal{T}$ Succ that "the smallest set closed under..." is implied (but often not stated explicitly). Terminology: \bullet axiom vs. rule

♦ concrete rule vs. rule scheme

15

Terms, concretely

Define an infinite sequence of sets, S_0 , S_1 , S_2 , ..., as follows:

 $S_{i+1} = \emptyset$ $S_{i+1} = \{ \text{true, false, 0} \}$ $\cup \{ \text{succ } t_1, \text{ pred } t_1, \text{ iszero } t_1 \mid t_1 \in S_i \}$ $\cup \{ \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid t_1, t_2, t_3 \in S_i \}$

Now let

 $S = \bigcup_{i} S_{i}$

CIS 500, 15 September

18

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)

XIS 500, 15 September

19

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 (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...

OIS 500, 15 September

21

Inductive Function Definitions

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

Consts (true)	=	{true}
Consts (false)	=	{false}
Consts(0)	=	{0}
$Consts(succ t_1)$	=	$Consts(t_1)$
$Consts(pred t_1)$	=	$Consts(t_1)$
Consts(iszero t1)	=	$Consts(t_1)$
$Consts(if t_1 then t_2 else t_3)$	=	$\textbf{Consts}(\texttt{t}_1) \cup \textbf{Consts}(\texttt{t}_2) \cup \textbf{Consts}(\texttt{t}_3)$

Induction on Terms

Definition: The depth of a term t is the smallest i such that $t \in S_i$.

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

This observation justifies the principle of induction on terms.

Let P be a predicate on terms.

If, for each term s, given P(r) for all immediate subterms r of s we can show P(s), then P(t) holds for all t.

XIS 500, 15 September

22

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??

Simple, right?

We have seen how to define relations inductively. E.g.... Let Consts be the smallest two-place relation closed under the following $(true, \{true\}) \in Consts$ $(false, \{false\}) \in Consts$ $(0, \{0\}) \in Consts$ $(t_1, C) \in Consts$ (succ t₁, C $) \in$ Consts $(t_1, C) \in Consts$ (pred t₁, C) \in Consts $(t_1, C) \in Consts$ (iszero t_1, C) \in Consts $(t_2, C_2) \in Consts$ $(t_1, C_1) \in Consts$ $(t_3, C_3) \in Consts$ (if t₁ then t₂ else t₃, (Consts(t₁) \cup Consts(t₂) \cup Consts(t₃))) \in 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?

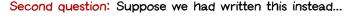
US 500, 15 September

rules:

25

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.



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

 $= \{true\}$

BadConsts (false)	=	{false}
BadConsts(0)	=	{0}
BadConsts(0)	=	8
$BadConsts(succ t_1)$	=	$BadConsts(t_1)$
$BadConsts(pred t_1)$	=	$BadConsts(t_1)$
$BadConsts(iszero t_1)$	=	$BadConsts(iszero (iszero t_1))$

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?

BadConsts(true)

27

XIS 500, 15 September

26

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 $(t,C)\in \text{Consts.}$

Proof:

OIS 500, 15 September

28-a

XIS 500, 15 September

29

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

i.e., for each term ${\tt t}$ there is exactly one set of terms C such that $({\tt t},C)\in {\sf Consts.}$

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 $(t,C)\in \text{Consts.}$

To apply the induction principle for terms, we must show, for an arbitrary term ${\tt t},$ 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 $(t, C) \in Consts$.

Proof: By induction on t.

US 500, 15 September

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 t₁, 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 $(t, C) \in Consts$.

30

30-b

XIS 500, 15 September

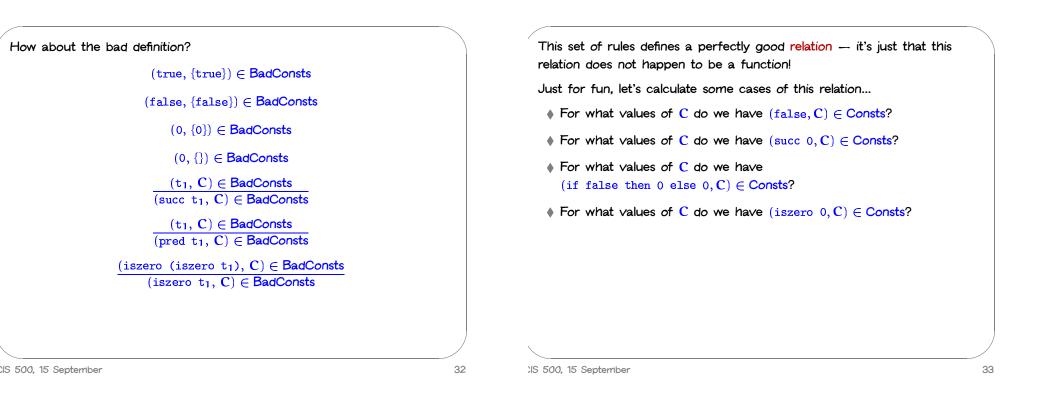
30-a

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 t₁, 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 $(t, C) \in Consts$.



_	
_	
	1
=	1
=	1
=	$size(t_1) + 1$
=	$size(t_1) + 1$
=	$size(t_1) + 1$
=	$\text{size}(\texttt{t}_1) + \text{size}(\texttt{t}_2) + \text{size}(\texttt{t}_3) + 1$
	=

Another proof by induction

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

Proof:

Another proof by induction Another proof by induction Theorem: The number of distinct constants in a term is at most the size Theorem: The number of distinct constants in a term is at most the size of the term. I.e., |Consts(t)| < size(t). of the term. I.e., |Consts(t)| < size(t). Proof: By induction on t. **Proof:** By induction on t. Assuming the desired property for immediate subterms of t, we must prove it for t itself. US 500, 15 September 35-a XIS 500, 15 September 35-b

Another proof by induction

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

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

```
\label{eq:limit} \mbox{Immediate: } |\mbox{Consts}(t)| = |\{t\}| = 1 = \mbox{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., $|Consts(t)| \le size(t)$.

Proof: By induction on t.

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

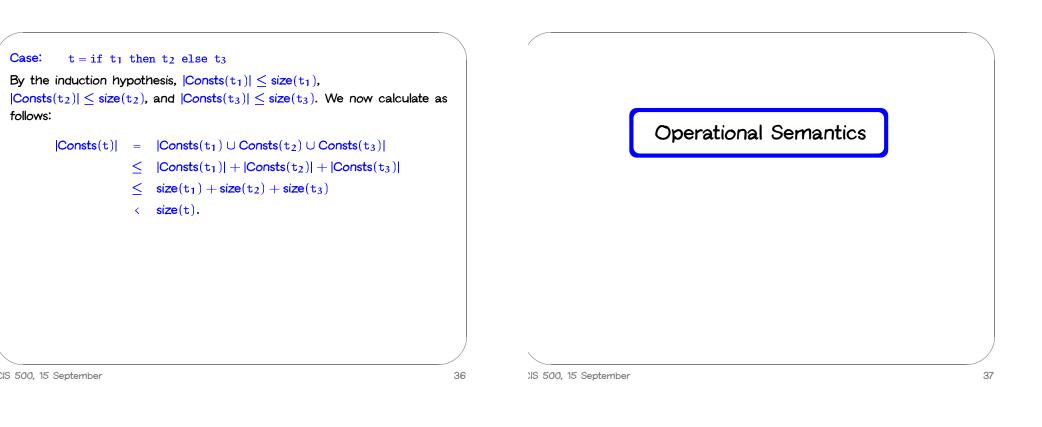
There are three cases to consider:

```
Case: t is a constant
```

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

Case: $t = succ t_1$, pred t₁, or iszero t₁

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



Abstract Machines

An abstract machine consists of:

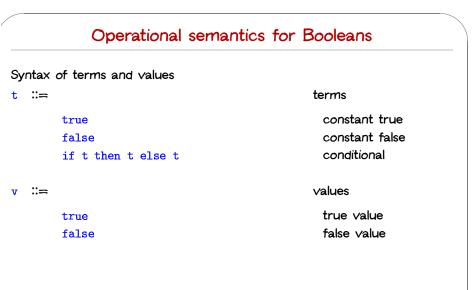
♦ a set of states

 \blacklozenge 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.



The evaluation relation $t \rightarrow t'$ is the smallest relation closed under the following rules: if true then t_2 else $t_3 \rightarrow t_2$ (E-IFTRUE) if false then t_2 else $t_3 \rightarrow t_3$ (E-IFFALSE) $\frac{t_1 \rightarrow t'_1}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t'_1 \text{ then } t_2 \text{ else } t_3}$ (E-IF) if t_1 then t_2 else $t_3 \rightarrow \text{if } t'_1$ then t_2 else t_3

