Programming Languages and Types

Klaus Ostermann

based on slides by Benjamin C. Pierce

Where we’re going

Type Systems...

Type systems are one of the most fascinating and powerful aspects of programming languages.

I could take for hours about why type systems are important etc., but instead we will skip directly to our first type system (after discussing some preliminaries).

Going Meta...

In this part of the course we will be more mathematical than in the first part.

We will define languages in terms of formal syntax, operational semantics, and type system.

We treat *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.

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 (ordinary) 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 ($n = i + 1$):

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

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 $i < n$*
> *we can show $P(n)$,*
> *then $P(n)$ holds for all n.*

Complete versus ordinary induction

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

Q: 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

Q: 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. $\{\texttt{true}, \texttt{false}, \texttt{0}\} \subseteq \mathcal{T}$;
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:

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

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

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

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

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

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

Now let

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

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 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.*

Inductive Function Definitions

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

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

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 $\mathtt{t}$, written $\mathit{BadConsts}(\mathtt{t})$, is defined as follows:

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

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?

What is a function?

Recall that a *function* f from A (its domain) to B (its co-domain) 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. More precisely:

  For every $a \in A$*, there exists some* $b \in B$ *such that* $(a, b) \in f$*.*

- It is *deterministic*: every element of its domain occurs at most once in its graph. More precisely:

  If $(a, b_1) \in f$ *and* $(a, b_2) \in f$*, then* $b_1 = b_2$*.*

We have seen how to define relations inductively. E.g....
Let *Consts* be the smallest two-place relation closed under the following rules:

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

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

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

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

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

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

$$\frac{(\texttt{t}_1,\, C_1) \in \mathit{Consts} \qquad (\texttt{t}_2,\, C_2) \in \mathit{Consts} \qquad (\texttt{t}_3,\, C_3) \in \mathit{Consts}}{(\texttt{if t}_1\ \texttt{then t}_2\ \texttt{else t}_3,\, (C_1 \cup C_2 \cup C_3)) \in \mathit{Consts}}$$

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 $\mathtt{t}$ there is exactly one set of terms C such that $(\mathtt{t}, C) \in \mathit{Consts}$.

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

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 $\mathtt{t}$ there is exactly one set of terms C such that $(\mathtt{t}, C) \in \mathit{Consts}$.

Proof: By induction on $\mathtt{t}$.
To apply the induction principle for terms, we must show, for an arbitrary term $\mathtt{t}$, that if

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

then

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

Proceed by cases on the form of $\mathtt{t}$.

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

Proceed by cases on the form of $\mathtt{t}$.

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

- If $\mathtt{t}$ is $\mathtt{succ\ t_1}$, then the induction hypothesis tells us that there is exactly one set of terms C_1 such that $(\mathtt{t}_1, C_1) \in \mathit{Consts}$. But then it is clear from the definition of *Consts* that there is exactly one set C (namely C_1) such that $(\mathtt{t}, C) \in \mathit{Consts}$.

Proceed by cases on the form of $\mathtt{t}$.

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

- If $\mathtt{t}$ is $\mathtt{succ}\ \mathtt{t}_1$, then the induction hypothesis tells us that there is exactly one set of terms C_1 such that $(\mathtt{t}_1, C_1) \in \mathit{Consts}$. But then it is clear from the definition of *Consts* that there is exactly one set C (namely C_1) such that $(\mathtt{t}, C) \in \mathit{Consts}$.

  Similarly when $\mathtt{t}$ is $\mathtt{pred}\ \mathtt{t}_1$ or $\mathtt{iszero}\ \mathtt{t}_1$.

- If $\mathtt{t}$ is $\mathtt{if}\ \mathtt{s}_1\ \mathtt{then}\ \mathtt{s}_2\ \mathtt{else}\ \mathtt{s}_3$, then the induction hypothesis tells us

  - there is exactly one set of terms C_1 such that $(\mathtt{t}_1, C_1) \in \mathit{Consts}$
  - there is exactly one set of terms C_2 such that $(\mathtt{t}_2, C_2) \in \mathit{Consts}$
  - there is exactly one set of terms C_3 such that $(\mathtt{t}_3, C_3) \in \mathit{Consts}$

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

How about the bad definition?

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

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

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

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

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

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

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

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 C do we have $(\texttt{false}, C) \in \mathit{BadConsts}$?

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 C do we have $(\texttt{false}, C) \in \mathit{BadConsts}$?
- For what values of C do we have $(\texttt{succ 0}, C) \in \mathit{BadConsts}$?

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

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

Another Inductive Definition

$$
\begin{array}{lcl}
\mathit{size}(\mathtt{true}) & = & 1 \\
\mathit{size}(\mathtt{false}) & = & 1 \\
\mathit{size}(\mathtt{0}) & = & 1 \\
\mathit{size}(\mathtt{succ}\ \mathtt{t}_1) & = & \mathit{size}(\mathtt{t}_1) + 1 \\
\mathit{size}(\mathtt{pred}\ \mathtt{t}_1) & = & \mathit{size}(\mathtt{t}_1) + 1 \\
\mathit{size}(\mathtt{iszero}\ \mathtt{t}_1) & = & \mathit{size}(\mathtt{t}_1) + 1 \\
\mathit{size}(\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3) & = & \mathit{size}(\mathtt{t}_1) + \mathit{size}(\mathtt{t}_2) + \mathit{size}(\mathtt{t}_3) + 1
\end{array}
$$

Another proof by induction

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

Proof:

Another proof by induction

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

Proof: By induction on $\mathtt{t}$.

Another proof by induction

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

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., $|\mathit{Consts}(\mathtt{t})| \leq \mathit{size}(\mathtt{t})$.

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

Another proof by induction

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

Proof: By induction on $\mathtt{t}$.
Assuming the desired property for immediate subterms of $\mathtt{t}$, we must prove it for $\mathtt{t}$ itself.

There are "three" cases to consider:

Case: $\mathtt{t}$ is a constant

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

Case: $\mathtt{t} = \mathtt{succ}\ \mathtt{t}_1$, $\mathtt{pred}\ \mathtt{t}_1$, or $\mathtt{iszero}\ \mathtt{t}_1$

By the induction hypothesis, $|\mathit{Consts}(\mathtt{t}_1)| \leq \mathit{size}(\mathtt{t}_1)$. We now calculate as follows:
$|\mathit{Consts}(\mathtt{t})| = |\mathit{Consts}(\mathtt{t}_1)| \leq \mathit{size}(\mathtt{t}_1) < \mathit{size}(\mathtt{t})$.

Case: $\quad \mathtt{t} = \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$

By the induction hypothesis, $|\mathit{Consts}(\mathtt{t}_1)| \leq \mathit{size}(\mathtt{t}_1)$, $|\mathit{Consts}(\mathtt{t}_2)| \leq \mathit{size}(\mathtt{t}_2)$, and $|\mathit{Consts}(\mathtt{t}_3)| \leq \mathit{size}(\mathtt{t}_3)$. We now calculate as follows:

$$\begin{array}{rcl}
|\mathit{Consts}(\mathtt{t})| & = & |\mathit{Consts}(\mathtt{t}_1) \cup \mathit{Consts}(\mathtt{t}_2) \cup \mathit{Consts}(\mathtt{t}_3)| \\
 & \leq & |\mathit{Consts}(\mathtt{t}_1)| + |\mathit{Consts}(\mathtt{t}_2)| + |\mathit{Consts}(\mathtt{t}_3)| \\
 & \leq & \mathit{size}(\mathtt{t}_1) + \mathit{size}(\mathtt{t}_2) + \mathit{size}(\mathtt{t}_3) \\
 & < & \mathit{size}(\mathtt{t}).
\end{array}$$

Structural Operational Semantics (SOS)

Abstract Machines

An *abstract machine* consists of:

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

We read "$\texttt{t} \longrightarrow \texttt{t}'$" as "$\texttt{t}$ evaluates to $\texttt{t}'$ in one step".

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.

Abstract Machines

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
```

Evaluation relation for Booleans

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

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

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

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

Terminology

Computation rules:

$$\texttt{if true then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_2 \qquad (\textsc{E-IfTrue})$$

$$\texttt{if false then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_3 \qquad (\textsc{E-IfFalse})$$

Congruence rule:

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{if } \texttt{t}_1' \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3} \quad (\textsc{E-If})$$

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:

$$((\texttt{if true then } \texttt{t}_2 \texttt{ else } \texttt{t}_3), \texttt{t}_2) \quad \in \quad \longrightarrow$$

$$((\texttt{if false then } \texttt{t}_2 \texttt{ else } \texttt{t}_3), \texttt{t}_3) \quad \in \quad \longrightarrow$$

$$\frac{(\texttt{t}_1, \texttt{t}_1') \quad \in \quad \longrightarrow}{((\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3), (\texttt{if } \texttt{t}_1' \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3)) \quad \in \quad \longrightarrow}$$

The notation $\texttt{t} \longrightarrow \texttt{t}'$ is short-hand for $(\texttt{t}, \texttt{t}') \in \longrightarrow$.

Digression

Suppose we wanted to change our evaluation strategy so that the `then` and `else` branches of an `if` get evaluated (in that order) before the guard. How would we need to change the rules?

Digression

Suppose we wanted to change our evaluation strategy so that the `then` and `else` branches of an `if` get evaluated (in that order) before the guard. How would we need to change the rules?

Suppose, moreover, that if the evaluation of the `then` and `else` branches leads to the same value, we want to immediately produce that value ("short-circuiting" the evaluation of the guard). How would we need to change the rules?

Digression

Suppose we wanted to change our evaluation strategy so that the `then` and `else` branches of an `if` get evaluated (in that order) before the guard. How would we need to change the rules?

Suppose, moreover, that if the evaluation of the `then` and `else` branches leads to the same value, we want to immediately produce that value ("short-circuiting" the evaluation of the guard). How would we need to change the rules?

Of the rules we just invented, which are computation rules and which are congruence rules?

Reasoning about Evaluation

Derivations

We can record the “justification” for a particular pair of terms that are in the evaluation relation in the form of a tree.

(on the board)

Terminology:

- These trees are called *derivation trees* (or just *derivations*).
- The final statement in a derivation is its *conclusion*.
- We say that the derivation is a *witness* for its conclusion (or a *proof* of its conclusion) — it records all the reasoning steps that justify the conclusion.

Observation

Lemma: Suppose we are given a derivation tree $\mathcal{D}$ witnessing the pair $(\mathtt{t},\, \mathtt{t}')$ in the evaluation relation. Then either

1. the final rule used in $\mathcal{D}$ is E-IfTrue and we have $\mathtt{t} = \mathtt{if\ true\ then\ t_2\ else\ t_3}$ and $\mathtt{t}' = \mathtt{t_2}$, for some $\mathtt{t_2}$ and $\mathtt{t_3}$, or
2. the final rule used in $\mathcal{D}$ is E-IfFalse and we have $\mathtt{t} = \mathtt{if\ false\ then\ t_2\ else\ t_3}$ and $\mathtt{t}' = \mathtt{t_3}$, for some $\mathtt{t_2}$ and $\mathtt{t_3}$, or
3. the final rule used in $\mathcal{D}$ is E-If and we have $\mathtt{t} = \mathtt{if\ t_1\ then\ t_2\ else\ t_3}$ and $\mathtt{t}' = \mathtt{if\ t_1'\ then\ t_2\ else\ t_3}$, for some $\mathtt{t_1}$, $\mathtt{t_1'}$, $\mathtt{t_2}$, and $\mathtt{t_3}$; moreover, the immediate subderivation of $\mathcal{D}$ witnesses $(\mathtt{t_1},\, \mathtt{t_1'}) \in \longrightarrow$.

Induction on Derivations

We can now write proofs about evaluation "by induction on derivation trees."

Given an arbitrary derivation $\mathcal{D}$ with conclusion $\mathtt{t} \longrightarrow \mathtt{t}'$, we assume the desired result for its immediate sub-derivation (if any) and proceed by a case analysis (using the previous lemma) of the final evaluation rule used in constructing the derivation tree.

E.g....

Induction on Derivations — Example

Theorem: If $\mathtt{t} \longrightarrow \mathtt{t}'$, i.e., if $(\mathtt{t}, \mathtt{t}') \in \longrightarrow$, then $\mathit{size}(\mathtt{t}) > \mathit{size}(\mathtt{t}')$.
Proof: By induction on a derivation $\mathcal{D}$ of $\mathtt{t} \longrightarrow \mathtt{t}'$.

1. Suppose the final rule used in $\mathcal{D}$ is E-IfTrue, with $\mathtt{t} = \mathtt{if\ true\ then\ t_2\ else\ t_3}$ and $\mathtt{t}' = \mathtt{t_2}$. Then the result is immediate from the definition of *size*.
2. Suppose the final rule used in $\mathcal{D}$ is E-IfFalse, with $\mathtt{t} = \mathtt{if\ false\ then\ t_2\ else\ t_3}$ and $\mathtt{t}' = \mathtt{t_3}$. Then the result is again immediate from the definition of *size*.
3. Suppose the final rule used in $\mathcal{D}$ is E-If, with $\mathtt{t} = \mathtt{if\ t_1\ then\ t_2\ else\ t_3}$ and $\mathtt{t}' = \mathtt{if\ t_1'\ then\ t_2\ else\ t_3}$, where $(\mathtt{t_1}, \mathtt{t_1'}) \in \longrightarrow$ is witnessed by a derivation $\mathcal{D}_1$. By the induction hypothesis, $\mathit{size}(\mathtt{t_1}) > \mathit{size}(\mathtt{t_1'})$. But then, by the definition of *size*, we have $\mathit{size}(\mathtt{t}) > \mathit{size}(\mathtt{t}')$.

Normal forms

A *normal form* is a term that cannot be evaluated any further — i.e., a term $\texttt{t}$ is a normal form (or "is in normal form") if there is no $\texttt{t}'$ such that $\texttt{t} \longrightarrow \texttt{t}'$.

A normal form is a state where the abstract machine is halted — i.e., it can be regarded as a "result" of evaluation.

Normal forms

A *normal form* is a term that cannot be evaluated any further — i.e., a term $\mathtt{t}$ is a normal form (or "is in normal form") if there is no $\mathtt{t}'$ such that $\mathtt{t} \longrightarrow \mathtt{t}'$.

A normal form is a state where the abstract machine is halted — i.e., it can be regarded as a "result" of evaluation.

Recall that we intended the set of *values* (the boolean constants `true` and `false`) to be exactly the possible "results of evaluation." Did we get this definition right?

Values $=$ normal forms

Theorem: A term `t` is a value iff it is in normal form.
Proof:
The $\Longrightarrow$ direction is immediate from the definition of the evaluation relation.

Values $=$ normal forms

Theorem: A term `t` is a value iff it is in normal form.
Proof:
The $\Longrightarrow$ direction is immediate from the definition of the evaluation relation.
For the $\Longleftarrow$ direction,

Values $=$ normal forms

Theorem: A term `t` is a value iff it is in normal form.
Proof:
The $\Longrightarrow$ direction is immediate from the definition of the evaluation relation.
For the $\Longleftarrow$ direction, it is convenient to prove the contrapositive: If `t` is *not* a value, then it is *not* a normal form.

Values $=$ normal forms

Theorem: A term $\mathtt{t}$ is a value iff it is in normal form.

Proof:

The $\Longrightarrow$ direction is immediate from the definition of the evaluation relation.

For the $\Longleftarrow$ direction, it is convenient to prove the contrapositive: If $\mathtt{t}$ is *not* a value, then it is *not* a normal form. The argument goes by induction on $\mathtt{t}$.

Note, first, that $\mathtt{t}$ must have the form $\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$ (otherwise it would be a value). If $\mathtt{t}_1$ is $\mathtt{true}$ or $\mathtt{false}$, then rule E-IfTrue or E-IfFalse applies to $\mathtt{t}$, and we are done.

Otherwise, $\mathtt{t}_1$ is not a value and so, by the induction hypothesis, there is some $\mathtt{t}_1'$ such that $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$. But then rule E-If yields

$$\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \longrightarrow \mathtt{if}\ \mathtt{t}_1'\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$$

i.e., $\mathtt{t}$ is not in normal form.

Numbers

New syntactic forms

```
t  ::=  ...                               terms
        0                                   constant zero
        succ t                              successor
        pred t                              predecessor
        iszero t                            zero test

v  ::=  ...                               values
        nv                                  numeric value

nv  ::=                                   numeric values
        0                                   zero value
        succ nv                             successor value
```

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{succ}\ \mathtt{t}_1 \longrightarrow \mathtt{succ}\ \mathtt{t}_1'} \qquad \text{(E-Succ)}$$

$$\mathtt{pred}\ \mathtt{0} \longrightarrow \mathtt{0} \qquad \text{(E-PredZero)}$$

$$\mathtt{pred}\ (\mathtt{succ}\ \mathtt{nv}_1) \longrightarrow \mathtt{nv}_1 \qquad \text{(E-PredSucc)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{pred}\ \mathtt{t}_1 \longrightarrow \mathtt{pred}\ \mathtt{t}_1'} \qquad \text{(E-Pred)}$$

$$\mathtt{iszero}\ \mathtt{0} \longrightarrow \mathtt{true} \qquad \text{(E-IszeroZero)}$$

$$\mathtt{iszero}\ (\mathtt{succ}\ \mathtt{nv}_1) \longrightarrow \mathtt{false} \qquad \text{(E-IszeroSucc)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{iszero}\ \mathtt{t}_1 \longrightarrow \mathtt{iszero}\ \mathtt{t}_1'} \qquad \text{(E-IsZero)}$$

Values are normal forms

Our observation a few slides ago that all values are in normal form still holds for the extended language.

Is the converse true? I.e., is every normal form a value?

Values are normal forms, but we have stuck terms

Our observation a few slides ago that all values are in normal form still holds for the extended language.

Is the converse true? I.e., is every normal form a value?
No: some terms are *stuck*.

Formally, a stuck term is one that is a normal form but not a value.
What are some examples?

Stuck terms model run-time errors.

Multi-step evaluation.

The *multi-step evaluation* relation, $\longrightarrow^*$, is the reflexive, transitive closure of single-step evaluation.

I.e., it is the smallest relation closed under the following rules:

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\mathtt{t} \longrightarrow^* \mathtt{t}'}$$

$$\mathtt{t} \longrightarrow^* \mathtt{t}$$

$$\frac{\mathtt{t} \longrightarrow^* \mathtt{t}' \qquad \mathtt{t}' \longrightarrow^* \mathtt{t}''}{\mathtt{t} \longrightarrow^* \mathtt{t}''}$$

Termination of evaluation

Theorem: For every $\mathtt{t}$ there is some normal form $\mathtt{t}'$ such that $\mathtt{t} \longrightarrow^* \mathtt{t}'$.

Proof:

Termination of evaluation

Theorem: For every $\texttt{t}$ there is some normal form $\texttt{t}'$ such that $\texttt{t} \longrightarrow^{*} \texttt{t}'$.

Proof:

- First, recall that single-step evaluation strictly reduces the size of the term:

$$\textit{if } t \longrightarrow t', \textit{ then } \textit{size}(t) > \textit{size}(t')$$

- Now, assume (for a contradiction) that

$$t_0,\ t_1,\ t_2,\ t_3,\ t_4,\ \ldots$$

  is an infinite-length sequence such that

$$t_0 \longrightarrow t_1 \longrightarrow t_2 \longrightarrow t_3 \longrightarrow t_4 \longrightarrow \cdots.$$

- Then

$$\textit{size}(t_0) > \textit{size}(t_1) > \textit{size}(t_2) > \textit{size}(t_3) > \ldots$$

- But such a sequence cannot exist — contradiction!

Termination Proofs

Most termination proofs have the same basic form:

Theorem: *The relation* $R \subseteq X \times X$ *is terminating — i.e., there are no infinite sequences* x_0, x_1, x_2, *etc. such that* $(x_i, x_{i+1}) \in R$ *for each* i.

Proof:

1. *Choose*
   - *a well-founded set* $(W, <)$ *— i.e., a set* W *with a partial order* $<$ *such that there are no infinite descending chains* $w_0 > w_1 > w_2 > \ldots$ *in* W
   - *a function* f *from* X *to* W
2. *Show* $f(x) > f(y)$ *for all* $(x, y) \in R$
3. *Conclude that there are no infinite sequences* x_0, x_1, x_2, *etc. such that* $(x_i, x_{i+1}) \in R$ *for each* i, *since, if there were, we could construct an infinite descending chain in* W.