CIS 500

Software Foundations

Fall 2002

18 September

Administrivia

- Last week's homework assignment is back in Christine's office if you want yours back. (But we didn't make any interesting marks on them.)
- We're going to try an electronic submission procedure for this week's homework assignment. Details will be given in the assignment.
- Make sure to answer the "debriefing" question!

Review (and a few more details)

Simple Arithmetic Expressions

The set $\mathcal{T}$ of terms is defined by the following abstract grammar:

`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

Inference Rule Notation

The set $\mathcal{T}$ is the smallest set closed under the following rules.

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

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

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

Each of these rules can be thought of as a generating function that, given some elements from $\mathcal{T}$, generates some new element of $\mathcal{T}$. Saying that $\mathcal{T}$ is closed under these rules means that $\mathcal{T}$ cannot be made any bigger using these generating functions — it already contains everything "justified" by its members.

Let's write these generating functions explicitly.

$$\begin{aligned}
F_1(U) &= \{\mathtt{true}\} \\
F_2(U) &= \{\mathtt{false}\} \\
F_3(U) &= \{\mathtt{0}\} \\
F_4(U) &= \{\mathtt{succ}\ \mathtt{t}_1 \mid \mathtt{t}_1 \in U\} \\
F_5(U) &= \{\mathtt{pred}\ \mathtt{t}_1 \mid \mathtt{t}_1 \in U\} \\
F_6(U) &= \{\mathtt{iszero}\ \mathtt{t}_1 \mid \mathtt{t}_1 \in U\} \\
F_7(U) &= \{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \mid \mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3 \in U\}
\end{aligned}$$

Each one takes a set of terms U as input and produces a set of "terms justified by U" as output.

If we now define

$$F(U) = F_1(U) \cup F_2(U) \cup F_3(U) \cup F_4(U) \cup F_5(U) \cup F_6(U) \cup F_7(U)$$

then we can restate the previous definition of the set of terms $\mathcal{T}$ like this...

Definition:

- A set U is said to be "closed under F" (or "F-closed") if $F(U) \subseteq U$.
- The set of terms $\mathcal{T}$ is the smallest F-closed set.

The concrete definition

Our other definition of the set of terms can also be stated using the generating function F:

$$\begin{aligned}
\mathcal{S}_0 &= \emptyset \\
\mathcal{S}_{i+1} &= F(\mathcal{S}_i) \\
\\
\mathcal{S} &= \bigcup_i \mathcal{S}_i
\end{aligned}$$

Compare this definition of $\mathcal{S}$ with the one we saw last time:

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

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

The only difference is that we have "pulled out" F and given it a name.

Note that our two definitions of terms characterize the same set $\mathcal{T}$ from different directions:

- "from above," as the intersection of all F-closed sets;
- "from below," as the limit (union) of a series of sets that start from $\emptyset$ and get "closer and closer to being F-closed."

Proposition 3.2.6 in the book (which we also stated in the last lecture, but did not prove) asserts that these two definitions actually define the same set.

An Inductive Function Definition

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

Another Inductive Definition

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

Proofs by Induction on Terms

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

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

This observation justifies a very common pattern of proofs "by induction on terms."

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

Proof:

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

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

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

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

Assuming the desired property for all terms of smaller depth than $\mathtt{t}$ (i.e., for all depths smaller than the depth of $\mathtt{t}$), we must prove it for $\mathtt{t}$ itself.

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: By induction on the depth of t.

Assuming the desired property for all terms of smaller depth than t (i.e., for all depths smaller than the depth of t), we must prove it for t itself.

There are three cases to consider:

Case: t is a constant

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

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: By induction on the depth of t.

Assuming the desired property for all terms of smaller depth than t (i.e., for all depths smaller than the depth 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 t_1, or iszero t_1

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

Case: t = if t_1 then t_2 else t_3

By the induction hypothesis [why does it apply??], $|Consts(t_1)| \leq size(t_1)$, $|Consts(t_2)| \leq size(t_2)$, and $|Consts(t_3)| \leq size(t_3)$. We now calculate as follows:

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

Structural Induction

The general principal underlying this proof is:

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.

Proofs based on this induction principle generally begin “By induction on the structure of t,” or just “By induction on t.”

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

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

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

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

Terminology

Computation rules:

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

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

Congruence rule:

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

Computation rules perform "real" computation steps.

Congruence rules determine where computation rules can be applied next.

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?

Evaluation, more explicitly

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

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

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

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

Even more explicitly...

What is the generating function corresponding to these rules?

Even more explicitly...

What is the generating function corresponding to these rules?

[on the board...]

Even more explicitly...

What is the generating function corresponding to these rules?

[on the board...]

Now we can write out a concrete version of the definition of $\longrightarrow$...

[on the board...]

Observations

As we did for terms, we can define the depth of a pair $(t, t') \in \longrightarrow$ as the smallest i such that $(t, t') \in \longrightarrow_i$.

Moreover, this formulation of the definition of evaluation immediately implies the following:

Lemma: If $(t, t') \in \longrightarrow_i$, then either

1. $t = \texttt{if true then } t_2 \texttt{ else } t_3$ and $t' = t_2$, for some t_2 and t_3, or
2. $t = \texttt{if false then } t_2 \texttt{ else } t_3$ and $t' = t_3$, for some t_2 and t_3, or
3. $t = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3$ and $t' = \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3$, for some t_1, t_1', t_2, and t_3 such that (t_1, t_1') is in $\longrightarrow_j$ for some $j < i$.

Together, these observations imply...

Induction on Evaluation

We can reason "by induction on evaluation" just as we did earlier on terms. For example...

Theorem: If $t \longrightarrow t'$ — i.e., if $(t, t') \in \longrightarrow$ — then $\text{size}(t) > \text{size}(t')$.

Proof: [...]

Aside

Q: Why are we bothering to prove all these completely obvious facts about terms and evaluation?

Aside

Q: Why are we bothering to prove all these completely obvious facts about terms and evaluation?

A: Suppose you told one of these facts to someone and they replied, "I don't believe it!" How would you convince them, aside from just saying, "Well, look at it again... isn't it obvious?"

I.e., we're trying to draw out why it is obvious.

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 the conclusion (or a proof of the conclusion) — it records all the reasoning steps that justify the conclusion.

Observation

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

1. the final rule used in $\mathcal{D}$ is E-IfTrue and we have $\texttt{t} = \texttt{if true then } \texttt{t}_2 \texttt{ else } \texttt{t}_3$ and $\texttt{t}' = \texttt{t}_2$, for some $\texttt{t}_2$ and $\texttt{t}_3$, or
2. the final rule used in $\mathcal{D}$ is E-IfFalse and we have $\texttt{t} = \texttt{if false then } \texttt{t}_2 \texttt{ else } \texttt{t}_3$ and $\texttt{t}' = \texttt{t}_3$, for some $\texttt{t}_2$ and $\texttt{t}_3$, or
3. the final rule used in $\mathcal{D}$ is E-If and we have $\texttt{t} = \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3$ and $\texttt{t}' = \texttt{if } \texttt{t}_1' \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3$, for some $\texttt{t}_1$, $\texttt{t}_1'$, $\texttt{t}_2$, and $\texttt{t}_3$; moreover, the immediate subderivation of $\mathcal{D}$ witnesses $(\texttt{t}_1, \texttt{t}_1') \in \longrightarrow$.

Induction on Derivations

Combining the previous ideas, we can write proofs about evaluation "By induction on derivation trees." E.g....

Theorem: If $\texttt{t} \longrightarrow \texttt{t}'$ — i.e., if $(\texttt{t}, \texttt{t}') \in \longrightarrow$ — then $size(\texttt{t}) > size(\texttt{t}')$.

Proof: By induction on a derivation of $\texttt{t} \longrightarrow \texttt{t}'$.

For step of the induction, we assume the desired result for all smaller derivations and proceed by a case analysis of the evaluation rule used at the root of the derivation tree.

[...]

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

$\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{succ\ t_1} \longrightarrow \mathtt{succ\ t_1'}} \quad \text{(E-Succ)}$$

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

$$\mathtt{pred\ (succ\ nv_1)} \longrightarrow \mathtt{nv_1} \quad \text{(E-PredSucc)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{pred\ t_1} \longrightarrow \mathtt{pred\ t_1'}} \quad \text{(E-Pred)}$$

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

$$\mathtt{iszero\ (succ\ nv_1)} \longrightarrow \mathtt{false} \quad \text{(E-IszeroSucc)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{iszero\ t_1} \longrightarrow \mathtt{iszero\ t_1'}} \quad \text{(E-IsZero)}$$

Aside

Q: Could we give the previous definition without bothering to introduce a separate category of numeric values?

Normal Forms

A normal form is a term $\mathtt{t}$ that does not evaluate to anything — i.e., such that there are no pairs of the form $(\mathtt{t}, \mathtt{t}')$ in $\longrightarrow$ for any $\mathtt{t}'$.

Normal Forms

A normal form is a term $\mathtt{t}$ that does not evaluate to anything — i.e., such that there are no pairs of the form $(\mathtt{t}, \mathtt{t}')$ in $\longrightarrow$ for any $\mathtt{t}'$.

Theorem: Every value $\mathtt{v}$ is a normal form.

Proof: [...]

Normal Forms

A normal form is a term $\mathtt{t}$ that does *not* evaluate to anything — i.e., such that there are *no* pairs of the form $(\mathtt{t}, \mathtt{t}')$ in $\longrightarrow$ for any $\mathtt{t}'$.

Theorem: Every value $\mathtt{v}$ is a normal form.

Proof: [...]

N.b.: When $\mathtt{t}$ is a normal form, we also say that $\mathtt{t}$ is "in normal form."

Stuck terms

Is the converse true?

Stuck terms

Is the converse true?

No: some terms are stuck.

Formally, a stuck term is one that is a normal form but not a value.

Stuck terms model run-time errors.

Multi-step evaluation.

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

That is, it is the smallest relation such that

1. if $\mathtt{t} \longrightarrow \mathtt{t}'$ then $\mathtt{t} \longrightarrow^* \mathtt{t}'$,
2. $\mathtt{t} \longrightarrow^* \mathtt{t}$ for all $\mathtt{t}$, and
3. if $\mathtt{t} \longrightarrow^* \mathtt{t}'$ and $\mathtt{t}' \longrightarrow^* \mathtt{t}''$, then $\mathtt{t} \longrightarrow^* \mathtt{t}''$.

Termination of evaluation

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

Proof:

Termination of evaluation

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

Proof: By induction on the number of steps in the derivation of $\mathtt{t} \longrightarrow^* \mathtt{t}'$....

More examples (time permitting)

- Nondeterministic choice (which properties are preserved when we add it?)
- A one-element memory
- A looping construct