CIS 500

Software Foundations

Fall 2005

Induction; Operational Semantics

Announcements

Review recitations start this week. You may go to any recitation section that you wish. You do not need to register for the section, nor do you need to attend the same section the entire semester. If you need help finding a study group, we will match people up in recitation sections this week.

Wed 3:30-5:00 PM	Levine 315	Bohannon
Thurs 10:30-12 PM	Levine 612	Aydemir
Thurs 1:30-3 PM	Levine 512	Bohannon
Fri 9:30-11 AM	Levine 512	Aydemir

First homework assignment is due one week from today.

Structural Induction

Boolean terms: Syntax

Recall the definition of the language $\mathcal{B}$:

```
t ::= true
      false
      not t
      if t then t else t
```

This was a short hand notation for the definition of the set $\mathcal{B}$.

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

1. $\{\texttt{true}, \texttt{false}\} \subseteq \mathcal{B}$;
2. if $\texttt{t}_1 \in \mathcal{B}$, then $\{\texttt{not t}_1\} \subseteq \mathcal{B}$;
3. if $\texttt{t}_1 \in \mathcal{B}$, $\texttt{t}_2 \in \mathcal{B}$, and $\texttt{t}_3 \in \mathcal{B}$, then $\texttt{if t}_1\ \texttt{then t}_2\ \texttt{else t}_3 \in \mathcal{B}$.

Boolean terms: Semantics

We defined the semantics of $\mathcal{B}$ using the relation ***Eval***. If $(\mathtt{t}_1, \mathtt{t}_2) \in \mathit{Eval}$ then $\mathtt{t}_2$ is the meaning of $\mathtt{t}_1$. Recall that ***Eval*** is the smallest set closed under the following rules:

1. $(\mathtt{true}, \mathtt{true}) \in \mathit{Eval}$
2. $(\mathtt{false}, \mathtt{false}) \in \mathit{Eval}$
3. $(\mathtt{not\ t}, \mathtt{true}) \in \mathit{Eval}$ when $(\mathtt{t}, \mathtt{false}) \in \mathit{Eval}$
4. $(\mathtt{not\ t}, \mathtt{false}) \in \mathit{Eval}$ when $(\mathtt{t}, \mathtt{true}) \in \mathit{Eval}$
5. $(\mathtt{if\ t_1\ then\ t_2\ else\ t_3}, \mathtt{t}) \in \mathit{Eval}$ when either:
   - $(\mathtt{t}_1, \mathtt{true}) \in \mathit{Eval}$ and $(\mathtt{t}_2, \mathtt{t}) \in \mathit{Eval}$
   - $(\mathtt{t}_1, \mathtt{false}) \in \mathit{Eval}$ and $(\mathtt{t}_3, \mathtt{t}) \in \mathit{Eval}$

Proving properties of programming languages

Suppose we want to prove that evaluation is deterministic. In other words: For all $\mathtt{t}$ there exists at most one $\mathtt{t'}$ such that $(\mathtt{t}, \mathtt{t'}) \in \boldsymbol{Eval}$.

Structural Induction

We can use induction for boolean terms. The way we have defined terms gives us an induction principle:

For all $\mathtt{t} \in \mathcal{B}$, $P(\mathtt{t})$ is true if and only if

- $P(\mathtt{true})$ and $P(\mathtt{false})$ hold
- for all $\mathtt{t}_1 \in \mathcal{B}$, if $P(\mathtt{t}_1)$ holds, then $P(\mathtt{not}\ \mathtt{t}_1)$ hold.
- for all $\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3 \in \mathcal{B}$, if $P(\mathtt{t}_1)$, $P(\mathtt{t}_2)$ and $P(\mathtt{t}_3)$ holds, then $P(\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3)$ holds.

Proofs by induction

We'll prove that evaluation is deterministic. In other words: For all $\texttt{t}$ there exists at most one $\texttt{t}'$ such that $(\texttt{t}, \texttt{t}') \in \boldsymbol{Eval}$.

This gives us the property:

$\boldsymbol{P}(\texttt{t})$ = exists at most one $\texttt{t}'$ such that $(\texttt{t}, \texttt{t}') \in \boldsymbol{Eval}$.

So we want to show:

- $\boldsymbol{P}(\texttt{true})$ (i.e. exists at most one $\texttt{t}'$ such that $(\texttt{true}, \texttt{t}') \in \boldsymbol{Eval}$)
- $\boldsymbol{P}(\texttt{false})$
- $\boldsymbol{P}(\texttt{not t}_1)$ given that $\boldsymbol{P}(\texttt{t}_1)$ holds.
- $\boldsymbol{P}(\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3)$ given that $\boldsymbol{P}(\texttt{t}_1)$, $\boldsymbol{P}(\texttt{t}_2)$ and $\boldsymbol{P}(\texttt{t}_3)$ all hold.

Boolean terms: Semantics

We defined the semantics of $\mathcal{B}$ using the relation ***Eval***. If $(\mathtt{t}_1, \mathtt{t}_2) \in \mathit{Eval}$ then $\mathtt{t}_2$ is the meaning of $\mathtt{t}_1$. Recall that ***Eval*** is the smallest set closed under the following rules:

1. $(\mathtt{true}, \mathtt{true}) \in \mathit{Eval}$
2. $(\mathtt{false}, \mathtt{false}) \in \mathit{Eval}$
3. $(\mathtt{not\ t}, \mathtt{true}) \in \mathit{Eval}$ when $(\mathtt{t}, \mathtt{false}) \in \mathit{Eval}$
4. $(\mathtt{not\ t}, \mathtt{false}) \in \mathit{Eval}$ when $(\mathtt{t}, \mathtt{true}) \in \mathit{Eval}$
5. $(\mathtt{if\ t_1\ then\ t_2\ else\ t_3}, \mathtt{t}) \in \mathit{Eval}$ when either:
   - $(\mathtt{t}_1, \mathtt{true}) \in \mathit{Eval}$ and $(\mathtt{t}_2, \mathtt{t}) \in \mathit{Eval}$
   - $(\mathtt{t}_1, \mathtt{false}) \in \mathit{Eval}$ and $(\mathtt{t}_3, \mathtt{t}) \in \mathit{Eval}$

Proof on chalkboard

Alternate notation: Inference rules

We can also define ***Eval*** using a shorthand notation. An alternate notation for the same definition:

$$(\texttt{true}, \texttt{true}) \in \textit{Eval} \qquad (\texttt{false}, \texttt{false}) \in \textit{Eval}$$

$$\frac{(\texttt{t}_1, \texttt{true}) \in \textit{Eval}}{(\texttt{not t}_1, \texttt{false}) \in \textit{Eval}} \qquad \frac{(\texttt{t}_1, \texttt{false}) \in \textit{Eval}}{(\texttt{not t}_1, \texttt{true}) \in \textit{Eval}}$$

$$\frac{(\texttt{t}_1, \texttt{true}) \in \textit{Eval} \qquad (\texttt{t}_2, \texttt{t}) \in \textit{Eval}}{(\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3, \texttt{t}) \in \textit{Eval}} \qquad \frac{(\texttt{t}_1, \texttt{false}) \in \textit{Eval} \qquad (\texttt{t}_3, \texttt{t}) \in \textit{Eval}}{(\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3, \texttt{t}) \in \textit{Eval}}$$

Note that, just in the BNF notation, "the smallest set closed under..." is implied (but often not stated explicitly).

Terminology:

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

Alternate notation: relational symbols

If we abbreviate $(\mathtt{t}, \mathtt{t}') \in \textbf{\textit{Eval}}$ as $\mathtt{t} \Downarrow \mathtt{t}'$ we can write these rules even more succinctly:

$$\mathtt{true} \Downarrow \mathtt{true} \qquad \mathtt{false} \Downarrow \mathtt{false}$$

$$\frac{\mathtt{t}_1 \Downarrow \mathtt{true}}{\mathtt{not}\ \mathtt{t}_1 \Downarrow \mathtt{false}} \qquad \frac{\mathtt{t}_1 \Downarrow \mathtt{false}}{\mathtt{not}\ \mathtt{t}_1 \Downarrow \mathtt{true}}$$

$$\frac{\mathtt{t}_1 \Downarrow \mathtt{true} \quad \mathtt{t}_2 \Downarrow \mathtt{t}}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \Downarrow \mathtt{t}} \qquad \frac{\mathtt{t}_1 \Downarrow \mathtt{false} \quad \mathtt{t}_3 \Downarrow \mathtt{t}}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \Downarrow \mathtt{t}}$$

The notation $\mathtt{t} \Downarrow \mathtt{t}'$ is read as "t evaluates to t'".

We will often abbreviate relations using symbols such as $\Downarrow$, $\rightarrow$, $\vdash$, etc.

Naming the rules

It is also useful to give names to each rule, so that we can refer to them later.

$$\texttt{true} \Downarrow \texttt{true} \qquad \text{B-True}$$

$$\texttt{false} \Downarrow \texttt{false} \qquad \text{B-False}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{true}}{\texttt{not t}_1 \Downarrow \texttt{false}} \qquad \text{B-NotTrue}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{false}}{\texttt{not t}_1 \Downarrow \texttt{true}} \qquad \text{B-NotFalse}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{true} \qquad \texttt{t}_2 \Downarrow \texttt{t}}{\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \Downarrow \texttt{t}} \qquad \text{B-IfTrue}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{false} \qquad \texttt{t}_3 \Downarrow \texttt{t}}{\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \Downarrow \texttt{t}} \qquad \text{B-IfFalse}$$

Derivations

The inference rule notation leads to a convenient notation for showing why a pair of terms is in the evaluation relation.

Say someone asked you to prove that

`if true then(not false) else (not true)` $\Downarrow$ `true`

Proving properties about evaluation

Last time we showed that the evaluation relation was a function.

i.e. for all `t` there is at most one `t'` such that `t` $\Downarrow$ `t'`.

Today we will show a related property: that evaluation is total.

i.e. for all `t` there is at least one `t'` such that `t` $\Downarrow$ `t'`.

How to prove this property?

Use structural induction

Again we will use the structural induction principle for terms in $\mathcal{B}$:

For all $\mathtt{t}$ in $\mathcal{B}$, $P(\mathtt{t})$ is true, if and only if

- $P(\mathtt{true})$ and $P(\mathtt{false})$ hold
- for all $\mathtt{t}_1 \in \mathcal{B}$, if $P(\mathtt{t}_1)$ holds, then $P(\mathtt{not}\ \mathtt{t}_1)$ hold.
- for all $\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3 \in \mathcal{B}$, if $P(\mathtt{t}_1)$, $P(\mathtt{t}_2)$ and $P(\mathtt{t}_3)$ holds, then $P(\mathtt{if}\ \mathtt{t}_1 \mathtt{then}\ \mathtt{t}_2 \mathtt{else}\ \mathtt{t}_3)$ holds.

To show that evaluation is total, we need $P(\mathtt{t})$ to be "there exists a $\mathtt{t}'$ such that $\mathtt{t} \Downarrow \mathtt{t}'$".

Strengthening the induction principle

We can not show that $P(\texttt{not t}_1)$, given $P(\texttt{t}_1)$.

$P(\texttt{t}_1)$ tells us that $\texttt{t}_1$ evaluates to some $\texttt{t}'$, but $\texttt{not t}_1$ only evaluates if $\texttt{t}'$ is `true` or `false`, and we don't know that.

What to do now? Are we stuck?

Strengthing the induction principle

The solution is to prove a property that implies the property that we want.

Instead of showing

"`t` there exists a `t'` such that `t` $\Downarrow$ `t'`"

we will show

"for all `t` either `t` $\Downarrow$ `true` or `t` $\Downarrow$ `false`"

Proving the second property implies that the first one is also true.

To show the second property we need `P(t)` to be "either `t` $\Downarrow$ `true` or `t` $\Downarrow$ `false`".

A larger language

Growing a language

The boolean language is an extremely simple language. There is not a lot that you can say with it.

At the same time, it is pretty easy to prove properties about it.

As we add to the expressiveness of a language, it usually becomes more difficult to show that the same properties are true.

In fact, some properties that are true for simple languages are not true for more expressive languages.

The language Arith

Consider a larger language, called Arith, that includes both booleans and natural numbers:

```
t ::= true
      false
      if t then t else t
      0
      succ t
      pred t
      iszero t
```

What is the structural induction principle for this language?

Language definability (informally)

This language does not include the term form `not t`.

However, all is not lost. Whenever we want to say `not t`, we can write: `if t then false else true`.

Because `not t` is definable, many of the same properties are true about Arith with `not t` as are true for Arith without `not t`.

Leaving out `not` means that our induction principle (and therefore our proofs) are shorter.

Semantics of Arith

To define the semantics of Arith, we will first define a subset of the terms of Arith that will be the result of evaluation.

These are called the values.

```
v  ::= bv
       nv
bv ::= true
       false
nv ::= 0
       succ nv
```

We use the metavariable `v` to indicate terms that are also values.

Semantics of Arith

Note: we are overloading the symbol $\Downarrow$ to refer to two different relations.

$$\texttt{true} \Downarrow \texttt{true} \qquad \text{B-True}$$

$$\texttt{false} \Downarrow \texttt{false} \qquad \text{B-False}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{true} \qquad \texttt{t}_2 \Downarrow \texttt{v}}{\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \Downarrow \texttt{v}} \qquad \text{B-IfTrue}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{false} \qquad \texttt{t}_3 \Downarrow \texttt{v}}{\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \Downarrow \texttt{v}} \qquad \text{B-IfFalse}$$

New rules:

$$\texttt{0} \Downarrow \texttt{0} \qquad \text{B-Zero}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{nv}}{\texttt{succ t}_1 \Downarrow \texttt{succ nv}} \qquad \text{B-Succ}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{0}}{\texttt{pred t}_1 \Downarrow \texttt{0}} \qquad \text{B-PredZero}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{succ nv}}{\texttt{pred t}_1 \Downarrow \texttt{nv}} \qquad \text{B-PredSucc}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{0}}{\texttt{iszero t}_1 \Downarrow \texttt{true}} \qquad \text{B-IsZeroZero}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{succ nv}}{\texttt{iszero t}_1 \Downarrow \texttt{false}} \qquad \text{B-IsZeroSucc}$$

Metavariables are useful

We can replace three rules:

$$\texttt{true} \Downarrow \texttt{true} \qquad \text{B-True}$$

$$\texttt{false} \Downarrow \texttt{false} \qquad \text{B-False}$$

$$\texttt{0} \Downarrow \texttt{0} \qquad \text{B-Zero}$$

With one rule:

$$\texttt{v} \Downarrow \texttt{v} \qquad \text{B-Value}$$

Properties of Arith

We showed that two properties were true of $\mathcal{B}$, are these same properties true of Arith?

- Evaluation is deterministic: for all `t`, there is at most one `t'` such that `t` $\Downarrow$ `t'`.
- Evaluation is total: for all `t`, either `t` $\Downarrow$ `true` or `t` $\Downarrow$ `false`.

The second is obviously false. What if we rephrase it as:

- Evaluation is total: for all `t`, `t` $\Downarrow$ `v`.

Evaluation is not total

- Evaluation is total: for all `t`, `t` $\Downarrow$ `v`.

There is a counterexample to this theorem. What does `succ false` evaluate to?

If we try to use induction to show this theorem, where does the proof break?

Some terms, like `succ false`, are "meaningless" in our semantics.

Stuck terms

It's a little unsettling that evaluation is not total.

- We want to give meanings to all terms.
- We want to (abstractly) describe the execution of a computer.
- Later: some languages contain infinite loops.
  - Those terms won't have meanings with this style of semantics either.
  - Want to distinguish loops from errors like `succ false`.

Small-step semantics

Small-step semantics

- Most of the semantics we will define in this course will be in a style called small-step operational semantics.
- Core idea: describe the "intermediate" steps of evaluation of an abstract machine.
- An abstract machine consists of:
  - a set of states
  - a transition relation on states, written $\rightarrow$

Small-step semantics

- Based on two relations between terms of Arith:
  - small-step evaluation: $\texttt{t} \rightarrow \texttt{t}'$
  - multi-step evaluation: $\texttt{t} \rightarrow^* \texttt{t}'$
- Small-step evaluation is the one step execution of the abstract machine. The states of the machine are terms.
- Multi-step evaluation is the reflexive, transitive closure of small-step evaluation. It describes execution sequences of the abstract machine.
  - $\texttt{t} \rightarrow^* \texttt{t}'$ is total (because of reflexivity).
  - $\texttt{t} \rightarrow \texttt{t}'$ may not be total (when the machine gets "stuck").

Normal forms

- A normal form is a term that cannot be evaluated any further – i.e. a term `t` is a normal form (or "is in normal form") is there is no `t'` such that `t` $\longrightarrow$ `t'`
- A normal form is a state where the abstract machine is halted – it can be regarded as a "result" of evaluation.
- The meaning of a term `t` with small-step semantics is a term `t'`, such that `t` $\longrightarrow^*$ `t'` and `t'` is a normal form.

  We say that `t'` "is the normal form of" `t`.

Normal forms

- For Arith, not all normal forms are values, but every value is a normal form.
- A term like `succ false` that is a normal form, but is not a value, is "stuck".

Small-step semantics

Booleans:

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3}$$

Natural numbers:

$$\frac{t_1 \longrightarrow t_1'}{\texttt{succ } t_1 \longrightarrow \texttt{succ } t_1'} \qquad \texttt{pred } 0 \longrightarrow 0 \qquad \texttt{pred } (\texttt{succ } nv_1) \longrightarrow nv_1$$

Both:

$$\texttt{iszero } 0 \longrightarrow \texttt{true} \qquad \texttt{iszero } (\texttt{succ } nv_1) \longrightarrow \texttt{false} \qquad \frac{t_1 \longrightarrow t_1'}{\texttt{iszero } t_1 \longrightarrow \texttt{iszero } t_1'}$$

What do all non-axiom rules in common?

Terminology

Computation rules:

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3$$

Congruence rules:

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3}$$

Computation rules perform "real" computation steps.

Congruence rules determine where computation rules can be applied next.

What about the other 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?

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?

Properties of this semantics

- (Homework): This small-step semantics "agrees" with the large-step semantics for terms that do not get stuck. In other words, $\texttt{t} \Downarrow \texttt{v}$ if and only if $\texttt{t} \rightarrow^* \texttt{v}$.
- The $\rightarrow$ relation is deterministic. If $\texttt{t} \rightarrow \texttt{t}'$ and $\texttt{t} \rightarrow \texttt{t}''$ then $\texttt{t}' = \texttt{t}''$.
- Evaluation is deterministic: There is at most one normal form for a term $\texttt{t}$. (Easy to prove: Follows because the $\rightarrow$ relation is deterministic).
- Evaluation is total: There is at least one normal form for a term $\texttt{t}$. (More difficult to prove: Must show that there are no infinite sequences of small-step evaluation.)

Reasoning about evaluation

Induction principles

We've seen three definitions of sets and their associated induction principles:

- Natural numbers
- Boolean terms
- Arithmetic terms

Given a set defined with BNF, it is not too hard to describe the structural induction principle for that set.

For example:

```
t ::= brillig
      tove
      snicker t
      gyre t gimble t
```

What is the structural induction principle for this language?

More induction principles

However, these are not the only sets that we've defined so far.

We defined the semantics of these languages using relations, and relations are just sets.

These sets also have induction principles.

Induction on evaluation

We can define an induction principle for small-step evaluation. Recall the definition (just for booleans, for now):

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

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

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

What is the induction principle for this relation?

Using this induction principle

For all $\mathtt{t}$, $\mathtt{t}'$, $\boldsymbol{P}(\mathtt{t} \rightarrow \mathtt{t}')$ if

- $\boldsymbol{P}(\mathtt{if\ true\ then\ t_2\ else\ t_3} \rightarrow \mathtt{t_2})$ and
- $\boldsymbol{P}(\mathtt{if\ false\ then\ t_2\ else\ t_3} \rightarrow \mathtt{t_3})$ and
- $\boldsymbol{P}(\mathtt{if\ t_1\ then\ t_2\ else\ t_3} \rightarrow \mathtt{if\ t_1'\ then\ t_2\ else\ t_3})$ given that $\boldsymbol{P}(\mathtt{t_1} \rightarrow \mathtt{t_1'})$

What does it mean to say
$\boldsymbol{P}(\mathtt{if\ t_1\ then\ t_2\ else\ t_3} \rightarrow \mathtt{if\ t_1'\ then\ t_2\ else\ t_3})$?

Derivations

Another way to look at it is in terms of derivations.

A derivation records the "justification" for a particular pair of terms that are in the evaluation relation, in the form of a tree. We've all ready seen one example:

(example on the board)

Terminology:

- These trees are called derivation trees (or just derivations)
- The final statement in a derivation is the conclusion
- We say that a derivation is a witness for its conclusion (or a proof of its conclusion) – it records the reasoning steps to justify the conclusion
- When we reason about the conclusions, we are reasoning about derivations

Observation

Lemma: Suppose we are given a derivation $\mathcal{D}$ witnessing the pair $(\mathtt{t}, \mathtt{t}')$ in the $\longrightarrow$ 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 $\mathtt{t_1} \longrightarrow \mathtt{t_1'}$.

Induction on Derivations

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

Given an arbitrary derivation $\mathcal{D}$ with conclusion $\texttt{t} \rightarrow \texttt{t}'$, 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 small-step evaluation

For example, we can show that small-step evaluation is deterministic.

Theorem: If $\mathtt{t} \to \mathtt{t}'$ then if $\mathtt{t} \to \mathtt{t}''$ then $\mathtt{t}' = \mathtt{t}''$.

Proof: By induction on a derivation $\mathcal{D}$ of $\mathtt{t} \to \mathtt{t}'$.

1. Suppose the final rule used in $\mathcal{D}$ is E-IfTrue, with $\mathtt{t} = \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$ and $\mathtt{t}_1 = \mathtt{true}$ and $\mathtt{t}' = \mathtt{t}_2$. Therefore, the last rule of the derivation of $\mathtt{t} \to \mathtt{t}'$ cannot be E-IfFalse, because $\mathtt{t}_1$ is not $\mathtt{false}$. Furthermore, the last rule cannot be E-If either, because this rule requires that $\mathtt{t}_1 \to \mathtt{t}_1'$, and $\mathtt{true}$ does not step to anything. So the last rule can only be E-IfTrue.

2. Suppose the final rule used in $\mathcal{D}$ is E-IfFalse, with $\mathtt{t} = \mathtt{if}\ \mathtt{false}\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$ and $\mathtt{t}' = \mathtt{t}_3$. This case is similar to the previous.

3. Suppose the final rule used in $\mathcal{D}$ is E-If, with $\mathtt{t} = \mathtt{if}\mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$ and $\mathtt{t}' = \mathtt{if}\mathtt{t}_1'\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$, where $\mathtt{t}_1 \to \mathtt{t}_1'$ is witnessed by a derivation

$\mathcal{D}_1$. The last rule in the derivation of $\mathtt{t} \to \mathtt{t}''$ can only be E-If, so it must be that $\mathtt{t}_1 \to \mathtt{t}_1''$. By induction $\mathtt{t}_1' = \mathtt{t}_1''$ so $\mathtt{t}' = \mathtt{t}''$.

What principle to use?

We've proven the same theorem using two different induction principles.

Q: Which one is the best one to use?

A: The one that works.

For these simple languages, anything you can prove by induction on $\texttt{t} \longrightarrow \texttt{t}'$, you can prove by structural induction on $\texttt{t}$. But that will not be the case for every language.

Well-founded induction

A Question

Why are any of these induction principles true? Why should I believe a proof that employs one?

Well-founded induction

Well-founded induction is a generalized form of all of these induction principles.

Let $\prec$ be a well-founded relation on a set $\mathbf{A}$. Let $\boldsymbol{P}$ be a property. Then $\forall \mathbf{a} \in \mathbf{A}.\boldsymbol{P}(\mathbf{a})$ iff

$$\forall \mathbf{a} \in \mathbf{A}.([\forall \mathbf{b} \prec \mathbf{a}.\boldsymbol{P}(\mathbf{b})] \Rightarrow \boldsymbol{P}(\mathbf{a})$$

Choosing the right set $\mathbf{A}$ and relation $\prec$ determines the induction principle.

Well-founded induction

For example, we let $A = \mathcal{N}$ and $n \prec m \stackrel{\text{def}}{=} m = n + 1$. In this case, we can rewrite previous principle as:

$\forall a \in \mathcal{N}.P(a)$ iff

$$\forall a \in \mathcal{N}.([\forall b \prec a.P(b)] \Rightarrow P(a)$$

Now, by definition a is either 0 or $i + 1$ for some i:

$\forall a \in \mathcal{N}.P(a)$ iff

$$[\forall b \prec 0.P(b)] \Rightarrow P(0) \wedge$$
$$\forall i \in \mathcal{N}.[\forall b \prec i + 1.P(b)] \Rightarrow P(i + 1)$$

Simplify to:

$\forall a \in \mathcal{N}.P(a)$ iff $P(0) \wedge \forall i \in \mathcal{N}.P(i) \Rightarrow P(i + 1)$

Strong induction

If $\prec$ is the "strictly less than" relation $<$, then the principle we get is strong induction.

$\forall \mathbf{a} \in \mathcal{N}.P(\mathbf{a})$ iff

$$\forall \mathbf{a} \in \mathcal{N}.([\forall \mathbf{b} < \mathbf{a}.P(\mathbf{b})] \Rightarrow P(\mathbf{a})$$

Well-founded relation

The induction principle holds only when the relation $\prec$ is well-founded.

Definition: A well-founded relation is a binary relation $\prec$ on a set $\mathbf{A}$ such that there are no infinite descending chains $\cdots \prec \mathbf{a_i} \prec \cdots \prec \mathbf{a_1} \prec \mathbf{a_0}$.

Are the successor and $<$ relations well-founded?

Proof of well-founded induction

We'd like to show that:

Theorem: Let $\prec$ is a well-founded relation on a set $\mathbf{A}$. Let $\boldsymbol{P}$ be a property. Then $\forall \mathbf{a} \in \mathbf{A}.\boldsymbol{P}(\mathbf{a})$ iff

$$\forall \mathbf{a} \in \mathbf{A}.([\forall \mathbf{b} \prec \mathbf{a}.\boldsymbol{P}(\mathbf{b})] \Rightarrow \boldsymbol{P}(\mathbf{a})$$

The ($\Rightarrow$) direction is trivial. We'll show the ($\Leftarrow$) direction.

First, observe that any nonempty subset Q of A has a minimal element, even if Q is infinite.

Now, suppose $\neg \boldsymbol{P}(\mathbf{a})$ for some $\mathbf{a}$ in $\mathbf{A}$. There must be a minimal element $\mathbf{m}$ of the set $\{\mathbf{a} \in \mathbf{A} | \neg \boldsymbol{P}(\mathbf{a})\}$. But then, $\neg \boldsymbol{P}(\mathbf{m})$ yet $[\forall \mathbf{b} \prec \mathbf{m}.\boldsymbol{P}(\mathbf{b})]$ which is a contradiction.

Structural induction

Well-founded induction also generalizes structural induction.

If $\prec$ is the "immediate subterm" relation for an inductively defined set, then the principle we get is structural induction.

For example, in Arith, the term $\texttt{t}_1$ is an immediate subterm of the term $\texttt{succ t}_1$.

Is the immediate subterm relation well-founded?

Yes, if all terms of Arith are finite.

Termination of evaluation

Termination of evaluation

Theorem: For every `t` there is some normal form `t'` such that `t` $\rightarrow^*$ `t'`.

An Inductive Definition

We can define the size of a term with the following relation:

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

Note: this is yet more shorthand. How would we write this definition with inference rules?

Induction on Derivations — Another 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 $\mathit{size}(\mathtt{t}) > \mathit{size}(\mathtt{t'})$.

Termination of evaluation

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

Proof:

Termination of evaluation

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

Proof:

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

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

- 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

  $size(t_0), size(t_1), size(t_2), size(t_3), size(t_4), \ldots$

  is an infinite, strictly decreasing, sequence of natural numbers.

- 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.