CIS 500

Software Foundations

Fall 2004

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	DRLB 4E9	review
Thurs 1:30-3 PM	Towne 321	review
Thurs 10:30-12 PM	Towne 307	review
Fri 9:30-11 AM	Towne 307	review

There will be no advanced recitation this week. It will start next week.

Wed 3:30-5:00 PM	DRLB 4C2	advanced

First homework assignment is due one week from today.

Structural Induction (continued)

Review: Proof by Induction

- Suppose we have a set S and we want to show that some property P holds of each of its members.
- Suppose that S is ordered, so that, for each element x of S, it makes sense to talk about "the elements of S immediately smaller than x."
- Prove the following implication for each element x:

  If P holds of all the elements of S immediately smaller than x, then it is also true of x.
- Apply the principle of induction to conclude that P holds of every element of S.

N.b.: this is a slightly rough sketch; we will see it more rigorously next time.

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

Boolean terms: Ordering

Note that the way we have constructed the set of boolean terms gives rise to a natural notion of "the set of terms smaller than a given term."

- the set of terms immediately smaller than `true` is $\{\}$ (the empty set)
- the set of terms immediately smaller than `false` is $\{\}$
- the set of terms immediately smaller than `not` t_1 is $\{t_1\}$
- the set of terms immediately smaller than `if` t_1 `then` t_2 `else` t_3 is $\{t_1, t_2, t_3\}$.

Boolean terms: Induction Principle (I)

Instantiating the general principle of induction with the specific ordering on boolean terms, we obtain a specific induction principle for boolean terms:

- Suppose we want to show that some property $\mathbf{P}$ holds of all boolean terms.
- Prove the following implication:
  - for each boolean term x, if $\mathbf{P}$ holds of every element of the set of terms immediately smaller than x, then it holds of x
- Conclude that $\mathbf{P}$ holds of all boolean terms.

Since the definition of the set of boolean terms has several cases, we can split the implication to be checked into four separate implications:

Boolean terms: Induction Principle (II)

- Prove the following implications:
  - for each boolean term x, if x has the form `true` and P holds of every element of the set of terms immediately smaller than x, then it holds of x
  - for each boolean term x, if x has the form `true` and P holds of every element of the set of terms immediately smaller than x, then it holds of x
  - for each boolean term x, if x has the form `not` t_1 and P holds of every element of the set of terms immediately smaller than x, then it holds of x
  - for each boolean term x, if x has the form `if` t_1 `then` t_2 `else` t_3 and P holds of every element of the set of terms immediately smaller than x, then it holds of x.
- Conclude that P holds of all boolean terms.

Combining this with the actual definition of “immediately smaller” from a few slides ago...

Boolean terms: Induction Principle (III)

- Prove the following:
  - for each boolean term x of the form `true`, P holds of x
  - for each boolean term x of the form `false`, P holds of x
  - for each boolean term x of the form `not` t_1, if P holds of t_1, then it holds of x
  - for each boolean term x of the form `if` t_1 `then` t_2 `else` t_3, if P holds of t_1, t_2, and t_3, then it holds of x.
- Conclude that P holds of all boolean terms.

Applying a few more logical simplifications...

Boolean terms: Induction Principle (Final Form)

- Prove the following:
  - P holds of `true`
  - P holds of `false`
  - for each boolean term t_1, if P holds of t_1, then it holds of `not` t_1
  - for each triple of boolean terms t_1, t_2, and t_3, if P holds of t_1, t_2, and t_3, then it holds of `if` t_1 `then` t_2 `else` t_3.
- Conclude that P holds of all boolean terms.

Boolean terms: Semantics

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

1. $(\texttt{true}, \texttt{true}) \in \text{Eval}$
2. $(\texttt{false}, \texttt{false}) \in \text{Eval}$
3. $(\texttt{not t}, \texttt{true}) \in \text{Eval}$ when $(\texttt{t}, \texttt{false}) \in \text{Eval}$
4. $(\texttt{not t}, \texttt{false}) \in \text{Eval}$ when $(\texttt{t}, \texttt{true}) \in \text{Eval}$
5. $(\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3, \texttt{t}) \in \text{Eval}$ when either:
   - ♦ $(\texttt{t}_1, \texttt{true}) \in \text{Eval}$ and $(\texttt{t}_2, \texttt{t}) \in \text{Eval}$
   - ♦ $(\texttt{t}_1, \texttt{false}) \in \text{Eval}$ and $(\texttt{t}_3, \texttt{t}) \in \text{Eval}$

Tidying the Notation

We're going to be writing a lot of definitions like Eval throughout the semester, so it's convenient to establish some shorthand notations for defining relations (just like we did with BNF for defining sets of abstract syntax trees).

Alternate notation: Inference rules

A more compact notation for the same definition:

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

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

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

Note that, just as 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 $(\mathtt{t}, \mathtt{t}') \in \mathsf{Eval}$ as $\mathtt{t} \Downarrow \mathtt{t}'$ we can write these rules even more succinctly:

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

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

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

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

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

Naming the rules

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

$\texttt{true} \Downarrow \texttt{true}$	B-True
$\texttt{false} \Downarrow \texttt{false}$	B-False
$\dfrac{\texttt{t}_1 \Downarrow \texttt{true}}{\texttt{not t}_1 \Downarrow \texttt{false}}$	B-NotTrue
$\dfrac{\texttt{t}_1 \Downarrow \texttt{false}}{\texttt{not t}_1 \Downarrow \texttt{true}}$	B-NotFalse
$\dfrac{\texttt{t}_1 \Downarrow \texttt{true} \qquad \texttt{t}_2 \Downarrow \texttt{t}}{\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \Downarrow \texttt{t}}$	B-IfTrue
$\dfrac{\texttt{t}_1 \Downarrow \texttt{false} \qquad \texttt{t}_3 \Downarrow \texttt{t}}{\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \Downarrow \texttt{t}}$	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

$$\texttt{if true then (not false) else (not true)} \Downarrow \texttt{true}$$

[on board]

Proving properties about evaluation

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

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

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

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

How to prove this property?

Use structural induction

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

P holds for all $\mathtt{t}$ in $\mathcal{B}$ if we can show

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

In this case, to show that evaluation is total, we choose the property $P(\mathtt{t})$ to be "there exists some $\mathtt{t}'$ such that $\mathtt{t} \Downarrow \mathtt{t}'$".

Instantiating the induction principle with this $\mathbf{P}$...

Use structural induction (II)

If we can prove...

- there is some $\mathtt{t}'$ such that $\mathtt{true} \Downarrow \mathtt{t}'$
- there is some $\mathtt{t}'$ such that $\mathtt{false} \Downarrow \mathtt{t}'$
- if there is some $\mathtt{t}_1'$ such that $\mathtt{t}_1 \Downarrow \mathtt{t}_1'$, then there is some $\mathtt{t}'$ such that $\mathtt{not}\ \mathtt{t}_1 \Downarrow \mathtt{t}'$
- if there are some $\mathtt{t}_1'$, $\mathtt{t}_2'$, and $\mathtt{t}_3'$ such that $\mathtt{t}_1 \Downarrow \mathtt{t}_1'$ and $\mathtt{t}_2 \Downarrow \mathtt{t}_2'$ and $\mathtt{t}_3 \Downarrow \mathtt{t}_3'$, then there is some $\mathtt{t}'$ such that $\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \Downarrow \mathtt{t}'$

...then we can conclude that, for every $\mathtt{t}$ in $\mathcal{B}$, there is some $\mathtt{t}'$ such that $\mathtt{t} \Downarrow \mathtt{t}'$.

[on board]

A Difficulty

Oops: We cannot show $P(\texttt{not t}_1)$, given $P(\texttt{t}_1)$.

$P(\texttt{t}_1)$ tells us that $\texttt{t}_1$ evaluates to some $\texttt{t}'$, but (according to the definition) $\texttt{not t}_1$ only evaluates further if $\texttt{t}'$ is `true` or `false`, and we haven't proved 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

> “for all $\mathtt{t}$ there exists a $\mathtt{t}'$ such that $\mathtt{t} \Downarrow \mathtt{t}'$”

we will show

> “for all $\mathtt{t}$ either $\mathtt{t} \Downarrow \mathtt{true}$ or $\mathtt{t} \Downarrow \mathtt{false}$”

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

To show the second property we choose $\mathtt{P(t)}$ to be “either $\mathtt{t} \Downarrow \mathtt{true}$ or $\mathtt{t} \Downarrow \mathtt{false}$”.

New instance of structural induction principle

If we can prove...

- `true` $\Downarrow$ `true` or `true` $\Downarrow$ `false`
- `false` $\Downarrow$ `true` or `false` $\Downarrow$ `false`
- if either $\texttt{t}_1$ $\Downarrow$ `true` or $\texttt{t}_1$ $\Downarrow$ `false`, then either `not` $\texttt{t}_1$ $\Downarrow$ `true` or `not` $\texttt{t}_1$ $\Downarrow$ `false`
- if either $\texttt{t}_1$ $\Downarrow$ `true` or $\texttt{t}_1$ $\Downarrow$ `false`, then either `not` $\texttt{t}_1$ $\Downarrow$ `true` or `not` $\texttt{t}_1$ $\Downarrow$ `false`
- if either $\texttt{t}_1$ $\Downarrow$ `true` or $\texttt{t}_1$ $\Downarrow$ `false` and either $\texttt{t}_2$ $\Downarrow$ `true` or $\texttt{t}_2$ $\Downarrow$ `false` and either $\texttt{t}_3$ $\Downarrow$ `true` or $\texttt{t}_3$ $\Downarrow$ `false`, then either `if` $\texttt{t}_1$ `then` $\texttt{t}_2$ `else` $\texttt{t}_3$ $\Downarrow$ `true` or `if` $\texttt{t}_1$ `then` $\texttt{t}_2$ `else` $\texttt{t}_3$ $\Downarrow$ `false`.

...then we can conclude that, for every `t` in $\mathcal{B}$, either `t` $\Downarrow$ `true` or `t` $\Downarrow$ `false`.

[on board]

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?

Derived forms (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 instead write `if t then false else true`.

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 can be the results of evaluation.

These are called the values.

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

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

Semantics of Arith

Old rules:

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

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

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

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

New rules:

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

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

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

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

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

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

Metavariables are useful

Since we have adopted the convention that the metavariable $\mathtt{v}$ denotes values, we can replace three rules

$\mathtt{true} \Downarrow \mathtt{true}$ B-True

$\mathtt{false} \Downarrow \mathtt{false}$ B-False

$\mathtt{0} \Downarrow \mathtt{0}$ B-Zero

with one rule:

$\mathtt{v} \Downarrow \mathtt{v}$ B-Value

Properties of Arith

We proved two properties of $\mathcal{B}$, above. Are these same properties true of Arith?

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

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

- Evaluation is total: for all $\mathtt{t}$, $\mathtt{t} \Downarrow \mathtt{v}$.

Evaluation is not total

$\boxed{\text{Evaluation is total: for all } \texttt{t},\ \texttt{t} \Downarrow \texttt{v}.}$

There is a counterexample to this claim: What does `succ false` evaluate to?

If we try to use induction to prove this claim, where does the proof break down?

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 know the meaning of all terms.
- We intend the evaluation relation to (abstractly) describe the execution of a computer. But then we would expect that

  ```
  succ false
  ```

  and

  ```
  if true then (succ false) else false
  ```

  should behave differently (the first is immediately stuck; the second does a little work and then gets stuck). But our evaluation relation treats them the same: neither evaluates to a value.
- Later: *Some languages contain infinite loops.*
  - Those terms won't have meanings either.
  - Want to distinguish loops from errors like `succ false`.

Small-step semantics

Small-step semantics

- Most of the evaluation relations we will define in this course will be in a style called small-step operational semantics.
- Core idea: describe evaluation as a sequence of “state changes” of an abstract machine.
- An abstract machine consists of:
  - a set of states
  - a transition relation on states, written $\longrightarrow$

Small-step semantics

- **Small-step evaluation**, written $\mathtt{t} \longrightarrow \mathtt{t}'$, is the *one-step* execution of the abstract machine. The states of the machine are just terms.
- **Multi-step evaluation**, written $\mathtt{t} \longrightarrow^* \mathtt{t}'$, is the reflexive, transitive closure of small-step evaluation. That is:
  - if $\mathtt{t} \longrightarrow \mathtt{t}'$ then $\mathtt{t} \longrightarrow^* \mathtt{t}'$
  - $\mathtt{t} \longrightarrow^* \mathtt{t}$ for every $\mathtt{t}$
  - if $\mathtt{t} \longrightarrow^* \mathtt{t}'$ and $\mathtt{t}' \longrightarrow^* \mathtt{t}''$ then $\mathtt{t} \longrightarrow^* \mathtt{t}''$.

  I.e., a term $\mathtt{t}$ is related (by the multi-step evaluation relation) to every term that can be obtained from $\mathtt{t}$ by "turning the crank" of single-step evaluation zero or more times.

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”) is there is no $\mathtt{t}'$ such that $\mathtt{t} \longrightarrow \mathtt{t}'$
- A normal form is a state where the abstract machine is halted – it can be regarded as a “result” of evaluation.
- We can say that the meaning of a term $\mathtt{t}$ in a small-step semantics is a term $\mathtt{t}'$, such that $\mathtt{t} \longrightarrow^* \mathtt{t}'$ and $\mathtt{t}'$ is a normal form.

  We say that $\mathtt{t}'$ “is the normal form of” $\mathtt{t}$.

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:

$$\texttt{if true then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \ \longrightarrow \texttt{t}_2 \qquad \texttt{if false then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \ \longrightarrow \texttt{t}_3$$

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

Natural numbers:

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{succ } \texttt{t}_1 \longrightarrow \texttt{succ } \texttt{t}_1'} \qquad \texttt{pred 0} \longrightarrow \texttt{0} \qquad \texttt{pred (succ } \texttt{nv}_1\texttt{)} \longrightarrow \texttt{nv}_1$$

Both:

$$\texttt{iszero 0} \longrightarrow \texttt{true} \quad \texttt{iszero (succ } \texttt{nv}_1\texttt{)} \longrightarrow \texttt{false} \quad \frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{iszero } \texttt{t}_1 \longrightarrow \texttt{iszero } \texttt{t}_1'}$$

Terminology

Computation rules:

$$\texttt{if true then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_2 \qquad \texttt{if false then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_3$$

Congruence rules:

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

Computation rules perform “real” computation steps.

Congruence rules determine where computation rules can be applied next.

What about the other rules?

Digression

The small-step semantics of Arith tells us exactly in which order the sub-terms of a given term will get evaluated.

Suppose we wanted to change the 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, $\mathtt{t} \Downarrow \mathtt{v}$ if and only if $\mathtt{t} \longrightarrow^{*} \mathtt{v}$.
- The $\longrightarrow$ relation is deterministic. If $\mathtt{t} \longrightarrow \mathtt{t}'$ and $\mathtt{t} \longrightarrow \mathtt{t}''$ then $\mathtt{t}' = \mathtt{t}''$.
- Evaluation is deterministic: There is at most *one* normal form for a term $\mathtt{t}$. (Easy to prove: Follows because the $\longrightarrow$ relation is deterministic).
- Evaluation is total: There is at least *one* normal form for a term $\mathtt{t}$. (More difficult to prove: Must show that there are *no* infinite sequences of small-step evaluation.)