Well-founded induction

Induction principles

What is the structural induction principle for this language?

```plaintext
    |     |     |
    v    v    v
  57te t 5ermte t

  snache t

  love
t ::= brillig
    tovesnickert
    gyretgimblet
```

For example:

Induction principle for that set.

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

- Arithmetic terms
- Boolean terms
- Natural numbers

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

Announcements

- Guest lectures for the next 3 weeks.
- No office hours 9/19, 9/26, 10/3.
- Pseudocode—cs500@cis.upenn.edu
- I will be reachable by email.

I will be away September 14-October 5.
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.

Choosing the right set \( A \) and relation \( \prec \) determines the induction principle:

\[
(\forall a \in A \cdot \forall b \prec a \cdot P(b)) \Rightarrow P(a)
\]

Well-founded induction

Well-founded induction

Let \( \prec \) be a well-founded relation on a set \( A \). Let \( \forall \cdot \exists \cdot \) be a property. Then well-founded induction is a generalized form of all of these induction principles.

Strong induction

If \( \prec \) is the strictly less than relation \( \prec \), then the principle we get is strong induction:

\[
(\forall a \in A \cdot \forall b \prec a \cdot P(b)) \Rightarrow P(a)
\]

Strong induction

For example, we let \( A = \mathbb{N} \) and \( n \prec m \) if \( m = n + 1 \). In this case, we can simplify to:

\[
(\forall a \in A \cdot \forall b \prec a \cdot P(b)) \Rightarrow P(a)
\]
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 \( A \) such that
there are no inﬁnite descending chains \( a_0 \succ a_1 \succ \cdots \succ a_n \succ a_{n+1} \).
Digression

Suppose we wanted to change our evaluation strategy so that the \texttt{then} and \texttt{else} branches of an \texttt{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 \texttt{then} and \texttt{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?
Normal forms

A normal form is a term that cannot be evaluated any further—i.e., a term
that is a normal form (or has a normal form) is haltable: it can be
regarded as a „result“ of evaluation.

For Arith, not all normal forms are values, but every value is a normal
form.

Properties of this semantics

We say that \( t \) is \( \downarrow \) the normal form of \( t' \).

The meaning of a term \( t \) with small-step semantics is a term \( t' \), such that
\( t \downarrow t' \).

\( t \downarrow t' \) if and only if \( t \) \( \rightarrow \) \( t' \).

The \( \rightarrow \) relation is deterministic: If \( t \rightarrow t_0 \) and \( t \rightarrow t_0' \) then \( t_0 = t_0' \).

Evaluation is total: There is at least one normal form for a term \( t \).

Evaluation is local: There is at least one normal form for a term \( t \).

A normal form is a state where the abstract machine is halted—i.e., a term
that is a normal form (or has a normal form) is haltable: it can be
regarded as a „result“ of evaluation.

Properties

small-step evaluation

Normal forms

For Arith, not all normal forms are values, but every value is a normal
form.

A term like \( \text{succ} \text{false} \) that is a normal form, but is not a value, is
„stuck.“

A term like \( \text{false} \) is a normal form, but is not a value, is
„false.“

Reversing about evaluation
Induction on evaluation

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

\[
\begin{align*}
\text{E-IfTrue} & : \text{if true then } t_2 \text{ else } t_3 \\
\text{E-IfFalse} & : \text{if false then } t_2 \text{ else } t_3 \\
\text{E-If} & : \text{if } t_1 \text{ then } t_2 \text{ else } t_3
\end{align*}
\]

We view reasoning about the conclusions as reasoning about derivations. It records the reasoning steps to justify the conclusion.

The final statement in a derivation is the conclusion.

These trees are called derivation trees (or just derivations).

Terminology:

Example on the board.

When we reason about the conclusions, we are reasoning about derivations.

\[
\begin{align*}
\text{If } t_1 \text{ then } t_2 \text{ else } t_3 
\end{align*}
\]

We say that a derivation is a witness for its conclusion (or a proof of its conclusion).

Lemma: Suppose we are given a derivation \(D\) witnessing the pair \((t, t_0)\) in the evaluation relation. Then either:

1. the final rule used in \(D\) is \text{E-IfTrue} and we have \(t = \text{if true then } t_2 \text{ else } t_3\) and \(t_0 = t_2\) for some \(t_2\) and \(t_3\), or
2. the final rule used in \(D\) is \text{E-IfFalse} and we have \(t = \text{if false then } t_2 \text{ else } t_3\) and \(t_0 = t_3\) for some \(t_2\) and \(t_3\), or
3. the final rule used in \(D\) is \text{E-If} and we have \(t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3\) and \(t_0 = \text{if } t_0 \text{ then } t_1 \text{ else } t_2\) for some \(t_1\) and \(t_0\) and \(t_2\) and \(t_3\). Moreover, the immediate subderivation of \(D\) at \((t, t_0)\) gives that \(t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3\) and \(t_0 = \text{if } t_0 \text{ then } t_1 \text{ else } t_2\) for all \(t_1\), \(t_0\), \(t_2\), and \(t_3\).
### Induction on Derivations

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

Given an arbitrary derivation \( D \) with conclusion \( t \downarrow t_0 \), we assume the desired result for its immediate sub-derivations (if any) and proceed by a case analysis (using the previous lemma) of the final evaluation rule used in constructing the derivation tree.

#### Every Language

You can prove by structural induction on \( t \). But then you will not be able to prove for every simple language. Anything you can prove by induction on \( t \downarrow t_1 \downarrow t_2 \),

\( A \) The one that works.

\( Q \) Which one is the best one to use?

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

---

### What Principle to Use?

#### Every Language

You can prove by structural induction on \( t \). But then you will not be able to prove for every simple language. Anything you can prove by induction on \( t \downarrow t_1 \downarrow t_2 \),

\( A \) The one that works.

\( Q \) Which one is the best one to use?

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

---

### Induction on Small-Step Evaluation

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

1. Suppose the final rule used in \( D \) is \( E \)-IFE.

2. Suppose the final rule used in \( D \) is \( E \)-IFE.

3. Suppose the final rule used in \( D \) is \( E \)-IFE.

**Note:** The last rule in the derivation of \( t \downarrow t_0 \) can only be \( E \)-IF, so it must be that \( t_1 \downarrow t_0 \). By induction on \( t_1 \), we proceed by a case analysis (using the previous lemma) of the final evaluation rule used in constructing the derivation tree.

#### Every Language

You can prove by structural induction on \( t \). But then you will not be able to prove for every simple language. Anything you can prove by induction on \( t \downarrow t_1 \downarrow t_2 \),

\( A \) The one that works.

\( Q \) Which one is the best one to use?

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

---

### Induction on Derivations

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

Theorem: For every $t$ there is some normal form $n$ such that $t \rightarrow^* n$.

Induction on Derivations — Another Example

Theorem: For every $t$ there is some normal form $n$ such that $t \rightarrow^* n$.

An Inductive Definition

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

$$\text{size}(\text{true}) = 1$$
$$\text{size}(\text{false}) = 1$$
$$\text{size}(\text{0}) = 1$$
$$\text{size}(\text{succ}(t)) = \text{size}(t) + 1$$
$$\text{size}(\text{iftrue}(t_1; t_2; t_3)) = \text{size}(t_1) + \text{size}(t_2) + \text{size}(t_3) + 1$$

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

Theorem:
For every \( t \) there is some normal form \( t_0 \) such that \( t \rightarrow^* t_0 \).

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

Let \( X \) be the set of all terms. Define a relation \( \rightarrow \) on \( X \) such that for all terms \( x \) and \( y \) in \( X \),

\[
\begin{align*}
(x \rightarrow y) & \iff (x \rightarrow^* y) \land \neg (y \rightarrow x),
\end{align*}
\]

where \( \rightarrow^* \) denotes the reflexive transitive closure of \( \rightarrow \).

Now, assume (for a contradiction) that \( (x \rightarrow^* y) \land \neg (y \rightarrow x) \) for each \( (x \rightarrow y) \). Since, if there were, we could construct an infinite descending chain in \( W \).

\( \rightarrow \) is an infinite strictly decreasing sequence of natural numbers.

Then

\[ x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \cdots \]

is an infinite strictly decreasing sequence such that

\[ \forall i, \forall x_i, \forall x_{i+1}, (x_i \rightarrow x_{i+1}) \]

Now, assume (for a contradiction) that

\[ \neg (x_n \rightarrow x_{n+1}) \]

If \( t \rightarrow t \), then \( size(t) \geq size(t) \). This contradicts the size of the term.

Proof:
For every \( t \) there is some normal form \( t_0 \) such that \( t \rightarrow t_0 \).