Lesson 1
Untyped Arithmetic Expressions

1/9/2002

Topics

• abstract syntax
• inductive definitions and proofs
• evaluation
• modeling runtime errors
Lesson 1: Untyped Arithmetic

An abstract syntax

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

Terms defined by a BNF style grammar.
Not worried about ambiguity.
t is a syntactic metavariable

example terms

true
0
succ 0
if false then 0
    else pred(if true then succ 0 else 0)
iszero true
if 0 then true else pred 0
Inductive defn of terms

Defn: The set of terms is the smallest set \( T \) such that

1. \( \{\text{true, false, 0}\} \subseteq T \)

2. if \( t_1 \subseteq T \), then \( \{\text{succ } t, \text{pred } t, \text{iszero } t\} \subseteq T \)

3. if \( t_1, t_2, t_3 \subseteq T \), then if \( t_1 \) then \( t_2 \) else \( t_3 \) \( \subseteq T \)

Terms defined using inference rules

Defn: The set of terms is defined by the following rules:

\[
\begin{align*}
\text{true} & \subseteq T \\
\text{false} & \subseteq T \\
0 & \subseteq T \\
\text{succ } t & \subseteq T \\
\text{pred } t & \subseteq T \\
\text{iszero } t & \subseteq T \\
\text{if } t_1 \text{ then } t_2 \text{ else } t_3 & \subseteq T
\end{align*}
\]
Definition by induction, concretely

Defn: For each i, define Si as follows

\[ S(0) = \emptyset \]
\[ S(i+1) = \{\text{true, false, 0}\} \]
\[ \big\{\text{succ } t, \text{ pred } t, \text{ iszero } t | t \in S(i)\} \]
\[ \big\{\text{if } t1 \text{ then } t2 \text{ else } t3 | t1,t2,t3 \in S(i)\} \]

Then let

\[ S = \big\{S(i) | i \in \text{Nat}\big\} \]

Proposition: \( S = T \)

---

Defining functions inductively

Constants appearing in a term

\[ \text{consts}(\text{true}) = \{\text{true}\} \]
\[ \text{consts}(\text{false}) = \{\text{false}\} \]
\[ \text{consts}(0) = \{0\} \]
\[ \text{consts}(\text{succ } t) = \text{consts}(t) \]
\[ \text{consts}(\text{pred } t) = \text{consts}(t) \]
\[ \text{consts}(\text{iszero } t) = \text{consts}(t) \]
\[ \text{consts}(\text{if } t1 \text{ then } t2 \text{ else } t3) = \]
\[ \text{consts}(t1) \big\{\text{consts}(t1) \big\} \text{consts}(t1) \]
Defining functions inductively

Size of a term:

\[
\begin{align*}
\text{size}(\text{true}) &= 1 \\
\text{size}(\text{false}) &= 1 \\
\text{size}(0) &= 1 \\
\text{size}(\text{succ } t) &= \text{size}(t) + 1 \\
\text{size}(\text{pred } t) &= \text{size}(t) + 1 \\
\text{size}(\text{iszero } t) &= \text{size}(t) + 1 \\
\text{size}(\text{if } t1 \text{ then } t2 \text{ else } t3) &= \text{size}(t1) + \text{size}(t2) + \text{size}(t3) + 1
\end{align*}
\]

Defining functions inductively

Depth of a term

\[
\begin{align*}
\text{depth}(\text{true}) &= 1 \\
\text{depth}(\text{false}) &= 1 \\
\text{depth}(0) &= 1 \\
\text{depth}(\text{succ } t) &= \text{depth}(t) + 1 \\
\text{depth}(\text{pred } t) &= \text{depth}(t) + 1 \\
\text{depth}(\text{iszero } t) &= \text{depth}(t) + 1 \\
\text{depth}(\text{if } t1 \text{ then } t2 \text{ else } t3) &= \max(\text{depth}(t1),\text{depth}(t2),\text{depth}(t3)) + 1
\end{align*}
\]
Proof by induction (on depth)

If, for each term \( s \),
   given \( P(r) \) for all terms with
   \( \text{depth}(r) < \text{depth}(s) \),
   we can show \( P(s) \)
then \( P(s) \) holds for all terms.

Proof by induction (on size)

If, for each term \( s \),
   given \( P(r) \) for all terms with
   \( \text{size}(r) < \text{size}(s) \),
   we can show \( P(s) \)
then \( P(s) \) holds for all terms.
**Proof by induction (on depth)**

If, for each term $s$,
given $P(r)$ for all immediate subterms of $S$,
we can show $P(s)$
then $P(s)$ holds for all terms.

---

**Operational semantics**

An abstract machine for with instructions on how to evaluate terms of the language.

In simple cases, the terms of the language can be interpreted as the instructions.

The values (results) can also be taken to be (simple) terms in the language.
Evaluation: booleans

Terms
\[ t ::= \text{true} \]
\[ | \text{false} \]
\[ | \text{if } t \text{ then } t \text{ else } t \]

Values
\[ v ::= \text{true} \]
\[ | \text{false} \]

Evaluation (reduction) relation

An evaluation relation is a binary relation
\[ t \rightarrow t' \]
on terms representing one step of evaluation.
This is known as a small-step or one-step evaluation relation.

A normal form is a term which is fully evaluated, i.e. for which no further evaluation is possible.
Thus, \( t \) is a normal form term if there is no term \( t' \) such that \( t \rightarrow t' \).
Evaluation rules for boolean terms

The evaluation relation $t \models t'$ is the least relation satisfying the following rules.

- $\text{if true then } t_2 \text{ else } t_3 \models t_2$
- $\text{if false then } t_2 \text{ else } t_3 \models t_3$
- $t_1 \models t_1'$
- $\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \models$
  $\text{if } t_1' \text{ then } t_2 \text{ else } t_3$

Evaluation strategy

Evaluation rules can determine an evaluation strategy that limits where evaluation takes place.

Example:
- $\text{if true then } (\text{if false then false else true}) \text{ else true}$
  $\models \text{if false then false else true}$
But not
- $\text{if true then } (\text{if false then false else true}) \text{ else true}$
  $\not\models \text{if true then true else true}$
Determinacy

Evaluation of boolean terms is deterministic. That is if \( t \downarrow t' \) and \( t \downarrow t'' \), then \( t' = t'' \).

Proof by induction on derivations of \( t \downarrow t' \).

Values and normal forms

Every value is a normal form (is in normal form).

For booleans, every normal form is a value.

But generally, not all normal forms are values.

E.g. \( \text{pred(true)} \)

Such non-value normal forms are called stuck.
Multistep evaluation

Defn: Let \( \mathcal{R}^* \) be the reflexive, transitive closure of \( \mathcal{R} \). I.e \( \mathcal{R}^* \) is the least reln such that

1. if \( t \mathcal{R} t' \) then \( t \mathcal{R}^* t' \)
2. \( t \mathcal{R}^* t \)
3. if \( t \mathcal{R}^* t' \) and \( t' \mathcal{R}^* t'' \) then \( t \mathcal{R}^* t'' \)

Boolean normal forms

Uniqueness of normal forms
Theorem: If \( t \mathcal{R}^* u \) and \( t \mathcal{R}^* u' \) where \( u \) and \( u' \) are normal forms, then \( u = u' \).
Proof: determinacy of \( \mathcal{R} \)

Existence of normal forms
Theorem: For any term \( t \), there is a normal form \( u \) such that \( t \mathcal{R}^* u \).
Proof: If \( t \mathcal{R} t' \), then \( t' \) is smaller than \( t \), i.e. \( \text{size}(t') < \text{size}(t) \).
Evaluation for arithmetic

Terms
\[ t ::= \ldots | 0 | \text{succ} \ t | \text{pred} \ t | \text{iszero} \ t \]

Values
\[ v ::= \ldots | \text{nv} \]
\[ \text{nv} ::= 0 | \text{succ} \ \text{nv} \]

Base computation rules

- \text{pred} 0 \rightarrow 0 \quad \text{E-PredZero}
- \text{pred} (\text{succ} \ \text{nv}) \rightarrow \text{nv} \quad \text{E-PredSucc}
- \text{iszero} 0 \rightarrow \text{true} \quad \text{E-IszeroZero}
- \text{iszero} (\text{succ} \ \text{nv}) \rightarrow \text{false} \quad \text{E-IszeroSucc}

Note that the E-PredSucc and E-IsZeroSucc rules are restricted to the case where the argument is a value (call-by-value).
Lesson 1: Untyped Arithmetic

**Congruence rules**

\[
\begin{align*}
\text{succ } t & \equiv \text{ succ } t' & \text{E-Succ} \\
\text{pred } t & \equiv \text{ pred } t' & \text{E-Pred} \\
iszero t & \equiv \text{ iszero } t' & \text{E-Iszero}
\end{align*}
\]

**Homework 1**

- Do exercises 3.5.13 and 3.5.14.
Stuck terms and runtime errors

Stuck terms
Defn: a closed term is stuck if it is a normal form but is not a value.

Examples:
- pred true
- if succ(0) then true else false

We can take stuck terms as representing runtime errors.