

A Language for Arithmetic Expressions

CS 565

Lecture 2

1/14/09

Preliminaries

- Familiarity with set notation
- Relations:
 - partial order (reflexive, transitive, antisymmetric)
 - joins and meets
- Induction
- Chapter 2, TAPL

Initial Goals

- Introduce a simple well-known language
 - basic arithmetic expressions
 - Study properties of this language via
 - abstract syntax
 - inductive definitions
 - proof strategies
 - Focus on techniques to reason about a language rather than the language itself
-

Syntax

BNF Grammar:

`t ::=`

`true`

`| false`

`| if t then t else t`

`| 0`

`| succ t`

`| pred t`

`| iszero t`

terms

constant true

constant false

conditional

constant 0

successor

predecessor

zero test

Terminology

`t` is a metavariable - it is a placeholder, but it is not a variable of the object language

Programs

- A program is just a term built from the grammar:
 - `true` \rightarrow `true`
 - `if false then 0 else 1` \rightarrow `1`
 - `iszero (pred (succ 0))` \rightarrow `true`
 - `succ(succ(succ 0))` \rightarrow `3`
 - Grammar does not prevent writing terms that may not make much sense:
 - `succ 0` \rightarrow ?
 - `if 0 then 0 else 0` \rightarrow ?
 - Grammar does not define rules to guide us in ascribing translation or meaning to terms
-

Abstract vs. concrete syntax

- What does the grammar actually define?
 1. a set of character strings
 2. a set of tokens
 3. a set of abstract syntax trees
 - It defines all three, but we are most interested in (3)
 - Call the grammar an “abstract grammar” because it defines a set of abstract syntax trees, along with a strategy for mapping character strings to these trees.
 - We use parentheses to disambiguate terms when the intended corresponding tree is not clear from context
-

Abstract vs. concrete syntax

□ Are:

- `succ 0`
- `succ (0)`
- `((succ (((0))))))`

"the same term"?

□ What about:

- `succ 0`
 - `pred (succ (succ 0))`
-

Syntax

The grammar is shorthand for the following inductive definition:

Definition: The set of terms is the smallest T such that

1. $\{\text{true}, \text{false}, 0\} \in T$
2. if $t_1 \in T$ then $\{\text{succ } t_1, \text{pred } t_1, \text{iszero } t_1\} \in T$
3. if $t_1 \in T, t_2 \in T, \text{ and } t_3 \in T, \text{ then}$
 $\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \in T$

First clause says there are three simple expressions (i.e., expressions that do not refer to meta-variables) in T

Second and third clauses says how compound expressions can be constructed from smaller constituent pieces

Alternative formulation (Inference Rules)

$\text{true} \in T$ $\text{false} \in T$ $0 \in T$

$\frac{t_1 \in T}{\text{succ } t_1 \in T}$ $\frac{t_1 \in T}{\text{pred } t_1 \in T}$ $\frac{t_1 \in T}{\text{iszero } t_1 \in T}$

$\frac{t_1 \in T, t_2 \in T, t_3 \in T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \in T}$

These rules are often referred to as "inference" rules
Rules without premises are called axioms

Each rule is read "If we have established the statements in the premises listed above the line, then we may conclude the statement listed below the line."

Alternative formulation (Concrete Terms)

For each natural number i , define set S_i as follows:

$$S_0 = \phi$$

$$S_{i+1} = \{\text{true}, \text{false}, 0\} \cup$$

$$[\{\text{succ } t_1, \text{pred } t_1, \text{iszero } t_1 \mid t_1 \in S_i\} \cup$$

$$[\{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid t_1, t_2, t_3 \in S_i\}$$

$$S = \bigcup_i S_i$$

This definition is constructive - it gives an explicit procedure for generating all the elements of T

Exercise: How many elements does S_3 have? What about S_i for arbitrary i ?

Exercise: Show that for any i , $S_i \subseteq S_{i+1}$.

Equivalence

- We have seen two basic ways for describing the language of simple arithmetic:
 - inductively where T is the smallest set closed under certain rules
 - explicit inductive definition
 - BNF shorthand
 - inference rule shorthand
 - concretely where S is the limit of a series of sets
 - None of the definitions actually describe the meaning of terms with respect to the values they represent
 - Are these different definitional styles equivalent?
-

Generating functions

- Each inference rule defining T can be thought of as a generating function that given some elements from T , generates new elements of T .
 - To say T is closed under these rules means that T cannot be made bigger using these generating functions.
-

Generating functions

$$F_1(U) = \{\text{true}\}$$

$$F_2(U) = \{\text{false}\}$$

$$F_3(U) = \{0\}$$

$$F_4(U) = \{\text{succ } t_1 \mid t_1 \in U\}$$

$$F_5(U) = \{\text{pred } t_1 \mid t_1 \in U\}$$

$$F_6(U) = \{\text{iszero } t_1 \mid t_1 \in U\}$$

$$F_7(U) = \{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid t_1, t_2, t_3 \in U\}$$

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

Relating back to T

We now define

$$F(U) = \bigcup_i F_i(U)$$

Definition:

- A set U is set to be closed under F (or F -closed) if $F(U) \subseteq U$
 - The set of terms T is the smallest F -closed set
-

Relating back to S

- We now have two constructive definitions that characterize the same set 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” to being F -closed
-

T = S

Proof: T is defined as the smallest set satisfying certain conditions. Suffice to show that (a) S satisfies these conditions and (b) any set satisfying these conditions has S as a subset

- Can prove (a) by inspection
 - Can prove (b) by complete induction on i .
 - Suppose S' satisfies the conditions defining T. Show that if for any i , $S_i \subseteq S'$, thus implying that $S \subseteq S'$
-

Induction on Terms

- if $t \in T$ then
 - t is a constant
 - t is of the form `succ t_1` , `pred t_1` , `iszero t_1` for some smaller term t_1
 - t is of the form `if t_1 then t_2 else t_3` for some smaller terms t_1 , t_2 , t_3
 - Can apply this observation to
 - define inductive definitions of functions over terms
 - inductive proofs of properties over terms
-

Example

The set of constants appearing in a term t written $\text{Consts}(t)$ is defined as:

$$\text{Consts}(\text{true}) = \{ \text{true} \}$$

$$\text{Consts}(\text{false}) = \{ \text{false} \}$$

$$\text{Consts}(0) = \{ 0 \}$$

$$\text{Consts}(\text{succ } t_1) = \text{Consts}(t_1)$$

$$\text{Consts}(\text{pred } t_1) = \text{Consts}(t_1)$$

$$\text{Consts}(\text{iszero } t_1) = \text{Consts}(t_1)$$

$$\text{Consts}(\text{if } t_1 \text{ then } t_2 \text{ else } t_3) =$$

$$\text{Consts}(t_1) \cup \text{Consts}(t_2) \cup \text{Consts}(t_3)$$

Inductive definitions

- In what sense is this a definition?
 - The thing we are defining is defined in terms of the thing we are defining
 - But, the specification has the essential trait of being unambiguous (it defines a function):
 - total: every element in the range is related by at least one element in its domain
 - deterministic: every element in the domain is related to at most one element in its range
-

Inductive definition

An alternative formulation:

$$\text{BadConsts}(\text{true}) = \{ \text{true} \}$$

$$\text{BadConsts}(\text{false}) = \{ \text{false} \}$$

$$\text{BadConsts}(0) = \{0\}$$

$$\text{BadConsts}(0) = \{ \}$$

$$\text{BadConsts}(\text{succ } t_1) = \text{BadConsts}(t_1)$$

$$\text{BadConsts}(\text{pred } t_1) = \text{BadConsts}(t_1)$$

$$\text{BadConsts}(\text{iszero } t_1) =$$

$$\text{BadConsts}(\text{iszero } (\text{iszero } t_1))$$

Inductive definitions

- BadConsts is not a well-formed inductive definition:
 - it is not deterministic (two rules for 0)
 - it is not total (no rule for if)
 - it is not inductive (rule for iszero)
-

Another inductive definition

$$\text{Size}(\text{true}) = 1$$

$$\text{Size}(\text{false}) = 1$$

$$\text{Size}(0) = 1$$

$$\text{Size}(\text{succ } t_1) = \text{Size}(t_1) + 1$$

$$\text{Size}(\text{pred } t_1) = \text{Size}(t_1) + 1$$

$$\text{Size}(\text{iszero } t_1) = \text{Size}(t_1) + 1$$

$$\text{then Consts}(\text{if } t_1 \text{ then } t_2 \text{ else } t_3) =$$

$$\text{Size}(t_1) \cup \text{Size}(t_2) \cup \text{Size}(t_3) + 1$$

The depth of a term t is the smallest i such that $t \in S_i$

Inductive proofs on terms

Lemma: The number of distinct constants in term t is no greater than the size of t (i.e., $|\text{Consts}(t)| \leq \text{Size}(t)$)

Proof: By induction on the depth of t .

Assuming the desired property for all terms of smaller depth than t holds, we must prove it for t itself.

Inductive proofs on terms

Case: t is a constant

$$|\text{Consts}(t)| = |\{t\}| = 1 = \text{Size}(t)$$

Case: $t = \text{succ}(t_1), \text{pred}(t_1), \text{iszero}(t_1)$

By the induction hypothesis,

$$|\text{Consts}(t_1)| \leq \text{Size}(t_1).$$

Now, $|\text{Consts}(t)| = |\text{Consts}(t_1)| \cdot$

$$\text{Size}(t_1) < \text{Size}(t)$$

Inductive proofs on terms

Case: $t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3$

By the induction hypothesis,

$$|\text{Consts}(t_i)| \leq \text{Size}(t_i) \quad 1 \leq i \leq 3.$$

Now, $|\text{Consts}(t)| =$

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

Structural induction

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 .

Variants:

Induction by depth:

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

Induction on size:

If for each term s , given $P(r)$ for all r such that $\text{size}(r) < \text{size}(s)$, we can show $P(s)$, then $P(s)$ holds for all s .

For next time...

- Chapter 3 TAPL
 - Download the Coq proof assistant 8.1
 - <http://coq.inria.fr>
 - Download and familiarize yourself with Ott
 - <http://www.cl.cam.ac.uk/~pes20/ott>
 - Exercise: use Ott to generate the language of arithmetic expressions
-