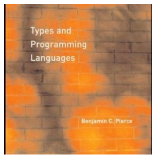
Preliminaries

CS 565

Lecture 2



Basics

Ordered sets:

- A binary relation R on a set S is:

  reflexive if $(s,s) \in R$

  symmetric if $(s,t) \in R \wedge (t,s) \in R$

  transitive if $(s,t), (t,u) \in R \Rightarrow (s,u) \in R$

  antisymmetric if $(s,t), (t,s) \in R \wedge s = t$

- A reflexive, transitive relation on S is a called a *preorder* on S
- A reflexive, transitive, antisymmetric relation on S is called a *partial order* ($\sqsubseteq$) on S

  A partial order is a *total order* if for each $s,t \in S$, either $s \sqsubseteq t$ or $t \sqsubseteq s$.

- A reflexive, transitive, symmetric relation on S is called an *equivalence relation* on S.

Basics

Suppose R is a binary relation on S.

- The *reflexive closure* of R is the smallest reflexive relation R' that contains R.
- The *transitive closure* of R is the smallest transitive relation R' that contains R.

Let R be a binary relation on S. Define R' as:

$R' = R \cup \{(s,s) \mid s \in S\}$

Show that R' is the reflexive closure of R.

Need to show that R' is a reflexive relation on R.

Need to show that R' is the smallest such relation.

Basics

Let S have preorder $\sqsubseteq$. We say $\sqsubseteq$ is *well-founded* if it contains no infinite decreasing chains.

A *preorder* is a relation that is reflexive and transitive.

- The preorder defining the natural numbers is well-founded.
- The preorder on the integers is not.

Induction

Principle of ordinary induction on natural numbers:

- Suppose that P is a predicate on N.
- Then if P(0) holds, and for all i, P(i) ⇒ P(i+1),

  P(n) holds for all n.

Example:

Theorem: $2^0 + 2^1 + \cdots + 2^{n-1} = 2^{n+1} - 1$ for all n

Proof

By induction on n.

- Base case (n = 0):

$$2^0 = 2^1 - 1$$

- Inductive case (n = i + 1):

$$\begin{aligned} 2^0 + 2^1 + \cdots + 2^{i+1} &= \\ (2^0 + 2^1 + \cdots + 2^i) + 2^{i+1} &= \\ 2^{i+1}-1 + 2^{i+1} &= \text{(induction hypothesis)} \\ 2 * 2^{i+1}-1 &= 2^{i+2} - 1 \end{aligned}$$

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 ::=`	terms
`true`	constant true
`| false`	constant false
`| if t then t else t`	Conditional
`| 0`	constant 0
`| succ t`	successor
`| pred t`	predecessor
`| iszero t`	zero test

Terminology:

`t` is a *metavariable*, not a variable of the object language

Programs

A program is just a term built from the grammar:

```
true → true
if false then 0 else 1 → 1
iszero (pred (succ 0)) → true
succ(succ(succ 0))) → 3
```

Grammar does not prevent writing terms that may not make much sense:

```
succ 0 → ?
if 0 then 0 else 0 → ?
```

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

`{true,false,0}` ⊆ T

if `t1` ∈ T then {`succ t1`,`pred t1`,`iszero t1` } ⊆ T

if `t1`, `t2` ∈ T, and `t3` ∈ T, then

if `t1` then `t2` else `t3` ∈ 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

$$\texttt{true} \in T \qquad \texttt{false} \in T \qquad \texttt{0} \in T$$

$$\frac{\texttt{t} \in T}{\texttt{succ t} \in T} \qquad \frac{\texttt{t} \in T}{\texttt{pred t} \in T}$$

$$\frac{\texttt{t} \in T}{\texttt{iszero t} \in T} \qquad \frac{\texttt{t}_1 \in T, \texttt{t}_2 \in T, \texttt{t}_3 \in T}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \in T}$$

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

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

$$\begin{aligned} S_0 &= \emptyset \\ S_{i+1} &= \{\texttt{true}, \texttt{false}, 0\} \cup \\ &= \{\texttt{succ } \texttt{t}_1, \texttt{pred } \texttt{t}_1, \texttt{iszero } \texttt{t}_1 | \texttt{t}_1 \in S_i\} \cup \\ & \quad \{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 | \texttt{t}_1, \texttt{t}_2, \texttt{t}_3, \in S_i\} \\ S &= \bigcup_i S_i \end{aligned}$$

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? 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
  - BNF shorthand
  - explicit inductive definition
  - inference rule shorthand
- concretely or constructively 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

F1(U) = {`true`}

F2(U) = {`false`}

F3(U) = {`0`}

F4(U) = {`succ t1` | `t1` ∈ U)

F5(U) = {`pred t1` | `t1` ∈ U)

F6(U) = {`iszero t1` | `t1` ∈ U)

F7(U) = {**`if`** `t1` **`then`** `t2` **`else`** `t3` | `t1,t2,t3` ∈ 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

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

Definition:

A set U is said 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 {} 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' satisfying the three conditions defining T. Show that for any i, $S_i \subseteq S'$, thus implying $S \subseteq S'$.
  Assume $S_j \subseteq S'$ for j < i and show that $S_i \subseteq S'$.

Induction on Terms

if `t` $\in$ T then

- `t` is a constant
- `t` is of the form `succ t'`, `pred t'`, `iszero t'` for some smaller term `t'`
- `t` is of the form `if t1 then t2 else t3` for some smaller terms `t1, t2, t3`

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 Consts(`t`) is defined as:

Consts(`true`) = { `true` }

Consts(`false`) = {`false`}

Consts(`0`) = {`0`}

Consts(`succ t`) = Consts(`t`)

Consts(`pred t`) = Consts(`t`)

Consts(`iszero t`) = Consts(`t`)

Consts(`if t1 then t2 else t3`) =
  Consts(`t1`) $\cup$ Consts(`t2`) $\cup$ Consts(`t3`)

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:

BadConsts(`true`) = { `true` }

BadConsts(`false`) = {`false`}

BadConsts(`0`) = {`0`}

BadConsts(`0`) = {}

BadConsts(`succ t`) = BadConsts(`t`)

BadConsts(`pred t`) = BadConsts(`t`)

BadConsts(`iszero t`) = BadConsts(`iszero (iszero t)`)

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

Size(`true`) = 1

Size(`false`) = 1

Size(`0`) = 1

Size(`succ t1`) = Size(t1) + 1

Size(`pred t1`) = Size(t1) + 1

Size(`iszero t1`) = Size(t1) + 1

Size(`if t1 then t2 else t3`) =
  Size(`t1`) + Size(`t2`) + Size(`t3`) + 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., |Consts(`t`)| ≤ 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

  | Consts(`t`) | = |{`t`}| = 1 = Size(`t`)

Case: `t` = `succ(t1)`, `pred(t1)`, `iszero(t1)`

  By the induction hypothesis,

  |Consts(`t1`) | ≤ Size(`t1`).

  Now, |Consts(`t`)| = |Consts(`t1`)| ·

   Size(`t1`) < Size(`t`)

Inductive proofs on terms

Case: `t = if t1 then t2 else t3`

By the induction hypothesis,

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

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

$|\text{Consts}(t1) \cup \text{Consts}(t2) \cup \text{Consts}(t3)|$

$\leq |\text{Consts}(t1)| + |\text{Consts}(t2)| + |\text{Consts}(t3)|$

$\leq \text{Size}(t1) + \text{Size}(t2) + \text{Size}(t3) < \text{Size}(t)$

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 depth(`r`) < 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 size(`r`) < size (`s`), we can show P(`s`), then P(`s`) holds for all `s`.