

Contents

1 Announcements

First homework is out; you can find it on the class web site. It is due October 11, before class. The homework is a programming homework, so start early.

There is also an exercise due this Thursday (see the last section).

2 Questions

What is a language? Should a language be understandable to be useful? How can we say a language is understandable (is a language understandable if it can potentially “dump core”)?

What is computation? Is there a mathematical meaning of what computation is?

Programs run the world. But they are almost never correct. Even your car may need a reboot. What if the robot that operates on you dumps core? Are there ways to prevent programs from “going wrong”?

Programs are complex objects and they get more complex. Why are they so complex? Is it the problems that we try to solve? Is it the set of primitives available to us? Is it the languages that we use?

Programming languages as a field concerns the answers to these types of problems. This class will give you an introduction to the field primarily by overviewing the basic concepts and techniques.

3 Syntax

Every language definition starts by defining syntax of the language. The syntax fixes the *object language* (the language of discourse). There are various ways to defining the syntax.

As an example, consider a language where we can write *terms* like this:

- true
- false
- 0

- `isZero 0`
- `succ 0`
- `pred succ 0`
- `if isZero (pred 0) then succ 0 else pred 0`

Let's define such a language.

3.1 The BNF Style.

This is perhaps the most common, succinct, way of defining syntax.

$$t ::= 0 \mid \text{true} \mid \text{false} \mid \\ \text{isZero } t \mid \text{succ } t \mid \text{pred } t \mid \\ \text{if } t \text{ then } t \text{ else } t$$

The variable t (stands for “term”) is said to be *meta variable*. It is *meta*, because its domain is the terms of the object language, as opposed to values of the object language.

Question: Which one of the following are legal terms of the language?

- `0`
- `true`
- `if true then true else true`
- `if isZero 0 then true else false`
- `if true then pred 0 else succ 0`
- `if succ 0 then pred 0 else succ 0`

Note: when t is used multiple times in a definition, it does not mean that all instances get the same term.

Note: some of the legal terms above make little sense. For example, `if succ 0 then ... else ...`

3.2 Inductive Definition

We can also define the syntax of a language inductively. Here is an inductive definition for our language:

The set of terms \mathcal{T} such that

1. $\{0, \text{true}, \text{false}\} \in \mathcal{T}$
2. If $t \in \mathcal{T}$, then $\{\text{succ } t, \text{pred } t, \text{isZero } t\} \subseteq \mathcal{T}$.

3. If $t_1 \in \mathcal{T} \wedge t_2 \wedge t_3 \in \mathcal{T}$, then **if** t_1 **then** t_2 **else** $t_3 \in \mathcal{T}$.

Question: Is this definition equivalent to the BNF definition?

Answer: Let T_{BNF} be the set of terms derivable from the BNF definition. To show equality we need to show that

1. $\forall t. t \in T_{BNF}. t \in \mathcal{T}$, and
2. $\forall t. t \in \mathcal{T}. t \in T_{BNF}$.

Let's consider the two cases separately.

1. For the first property, let's try an induction proof.

Solution:

2. **Solution:**

We have to change the definition to insist that the set be smallest. Thus, the correct inductive definition is, the smallest set of terms \mathcal{T} such that

1. $\{0, \text{true}, \text{false}\} \in \mathcal{T}$
2. If $t \in \mathcal{T}$, then $\{\text{succ } t, \text{pred } t, \text{isZero } t\} \subseteq \mathcal{T}$.
3. If $t_1 \in \mathcal{T} \wedge t_2 \wedge t_3 \in \mathcal{T}$, then **if** t_1 **then** t_2 **else** $t_3 \in \mathcal{T}$.

3.3 Inference Rules

Another way to define the syntax of a language is to use inference rules. Here is how we would define our language:

$$\begin{array}{c} \text{true} \in \mathcal{T} \quad \text{false} \in \mathcal{T} \quad 0 \in \mathcal{T} \\ \frac{t \in \mathcal{T}}{\text{succ } t \in \mathcal{T}} \quad \frac{t \in \mathcal{T}}{\text{pred } t \in \mathcal{T}} \quad \frac{t \in \mathcal{T}}{\text{isZero } t \in \mathcal{T}} \\ \frac{t_1 \in \mathcal{T} \quad t_2 \in \mathcal{T} \quad t_3 \in \mathcal{T}}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \in \mathcal{T}} \end{array}$$

The idea is that a terms is well defined if it is derivable by the application of these rules.

Some terminology: the terms without premises are called axioms. Each rule is called an *inference rule*.

Question: Is there a need to say that the syntax is the smallest set \mathcal{T} obtained by the inference rules?

Solution:

3.4 Concrete Syntax

For each $i \in \mathbb{N}$, define the set S_i as follows.

$$\begin{aligned} S_0 &= \emptyset \\ S_{i+1} &= \{\text{true, false, 0}\} \\ &\cup \{\text{succ } t, \text{pred } t, \text{isZero } t \mid t \in S_i\} \\ &\cup \{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid t_1, t_2, t_3 \in S_i\} \end{aligned}$$

and define $S = \bigcup_i S_i$.

Exercise: How many elements does S_3 have? Let's try to write a formula for $|S_i|$.

Solution:

Exercise: Prove that the set \mathcal{T} defined inductively and the set S defined concretely are equal, *i.e.*, $\mathcal{T} = S$.

Answer: : see the proof in the book. it is a good example of “complete induction”.

4 Induction on Terms

We often define properties of terms inductively.

4.1 Constants

We can define the constants of a term as follows.

$$\begin{aligned} \mathit{consts}(\mathbf{true}) &= \{\mathit{true}\} \\ \mathit{consts}(\mathbf{false}) &= \{\mathit{false}\} \\ \mathit{consts}(0) &= \{0\} \\ \mathit{consts}(\mathbf{succ } t) &= \mathit{consts}(t) \\ \mathit{consts}(\mathbf{pred } t) &= \mathit{consts}(t) \\ \mathit{consts}(\mathbf{isZero } t) &= \mathit{consts}(t) \\ \mathit{consts}(\mathbf{if } t_1 \mathbf{ then } t_2 \mathbf{ else } t_3) &= \mathit{consts}(t_1) \cup \mathit{consts}(t_2) \cup \mathit{consts}(t_3) \end{aligned}$$

4.2 Size

We define size of a term as the total size of the subterms plus one; constants have size one.

$$\begin{aligned} \mathit{size}(\mathbf{true}) &= 1 \\ \mathit{size}(\mathbf{false}) &= 1 \\ \mathit{size}(0) &= 1 \\ \mathit{size}(\mathbf{succ } t) &= 1 + \mathit{size}(t) \end{aligned}$$

$$\begin{aligned}
\text{size}(\text{pred } t) &= 1 + \text{size}(t) \\
\text{size}(\text{isZero } t) &= 1 + \text{size}(t) \\
\text{size}(\text{if } t_1 \text{ then } t_2 \text{ else } t_3) &= 1 + \text{size}(t_1) + \text{size}(t_2) + \text{size}(t_3)
\end{aligned}$$

The size of a term is the number of nodes in its *abstract syntax tree*.

Question: Define the depth of a term.

Answer: Very similar to size. Defined as the maximum depth of the sub-terms plus one.

4.3 Principles of Induction on Terms

Nearly all proofs in programming languages are by induction on terms. The following three forms of induction can all be used. They are indeed derivable from each other. Structural induction is the most direct technique of all and therefore is the most common.

To prove that predicate $P(\cdot)$ holds on all terms, we can use any of the following techniques.

1. Induction on depth: If for each term s of depth d , we can show that $P(s)$ assuming that $P(r)$ holds for all terms r of depth less than d , then $P(\cdot)$ holds.
2. Induction on size: If for each term s of size n , we can show that $P(s)$ assuming that $P(r)$ holds for all terms r of size less than n , then $P(\cdot)$ holds.
3. Structural induction: If for each term s , we can show that $P(s)$ holds assuming that $P(r)$ holds for each subterm of s , then $P(\cdot)$ holds.

5 Exercise (2 points)

1. (Exercise 3.2.5. from the text book)
For each $i \in \mathbb{N}$, define the set S_i as follows.

$$\begin{aligned}
S_0 &= \emptyset \\
S_{i+1} &= \{\text{true}, \text{false}, 0\} \\
&\cup \{\text{succ } t, \text{pred } t, \text{isZero } t \mid t \in S_i\} \\
&\cup \{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid t_1, t_2, t_3 \in S_i\}
\end{aligned}$$

Show that $\forall i. S_i \subseteq S_{i+1}$.

Solution:

2. Define $iValue(\cdot)$ over the terms as follows.

$$iValue(\mathbf{true}) = 0$$

$$iValue(\mathbf{false}) = 0$$

$$iValue(0) = 0$$

$$iValue(\mathbf{succ } t) = 1 + iValue(t)$$

$$iValue(\mathbf{pred } t) = iValue(t) - 1$$

$$iValue(\mathbf{isZero } t) = 0$$

$$iValue(\mathbf{if } t_1 \mathbf{ then } t_2 \mathbf{ else } t_3) = \max(iValue(t_2), iValue(t_3))$$

Show by structural induction that $iValue(\cdot) < size(\cdot)$.

Solution: