Lecture Notes on Inductive Definitions

15-411: Compiler Design
Jan Hoffmann

Lecture 1
August 29, 2017

1 Introduction

In this course and the lecture notes we will heavily rely on concepts like *inductive definitions*, *inference rules*, and *derivations* to define and formalize concepts and algorithms. The advantage of this formalism is that it is mathematically precise, flexible, concise, and can often be directly turned into a functional program using pattern matching and recursion.

It is important to understand these concepts since you will otherwise have trouble following the lectures and lecture notes. If you struggle then please review chapters 1 and 2 in Bob Harper's PFPL [Har12].

2 Judgments

A *judgment* (or predicate, assertion, ...) states a property of certain objects or relates two objects of potentially different classes. The objects are usually inductively defined (see next section). We do not use a general notation for judgments but use whatever notation is a good fit for the particular judgment we are formalizing.

Example judgments:

n even	n is an even number
type(τ)	τ is a type
$e : \tau$	expression e has type τ
live(ℓ, x)	variable x is live at line ℓ

One way to think of a judgment is of a set where the set consists of exactly the objects that have the respective property. For instance, you can think of the predicate *live* as defining a set L such that

$$(\ell, x) \in L \iff \text{live}(\ell, x) \; .$$

3 Inductive Definitions

Most judgments that you will see in this course talk about C0 programs and are defined inductively. It is likely that you are already familiar with inductive definitions. Most certainly, you have seen proofs by induction on the natural numbers: To prove a statement $A(n)$, you first prove that $A(n)$ holds for $n = 0$ and then you prove that $A(n)$ implies $A(n+1)$ for all natural numbers n. One way to look at such a proof is that you *inductively define* a proof for every number n. However, we cannot only inductively define proofs but also other objects.

For example, we can define the natural numbers themselves inductively. We say that set $\mathbb{N}$ of natural numbers is inductively defined by the following two rules.

1. $\text{nat}(0)$
2. If $\text{nat}(n)$ then $\text{nat}(n+1)$.

An inductive definition always implies that we are looking for the smallest set such that the given rules hold.[1] For example, the two previous rules hold for the natural numbers as well as for the integers. We have

$$\begin{array}{l} 0 \in \mathbb{N} \\ n \in \mathbb{N} \implies n+1 \in \mathbb{N} \end{array}$$

and

$$\begin{array}{l} 0 \in \mathbb{Z} \\ n \in \mathbb{Z} \implies n+1 \in \mathbb{Z} \; . \end{array}$$

But an inductive interpretation of the rules leads to the natural numbers since $\mathbb{N}$ is the smallest set for which rules (1) and (2) hold.

Another example of an inductive definition is given by the following rules.

$$\begin{array}{ll} (a) & \text{leq}(0,0) \\ (b) & \text{leq}(n,m) \implies \text{leq}(n+1,m+1) \\ (c) & \text{leq}(n,m) \implies \text{leq}(n,m+1) \end{array}$$

The first rule says that 0 is less or equal to 0. The second rule says that if n is less or equal to m then $n+1$ is less or equal to $m+1$. The judgment then formalizes the familiar relation $n \leq m$ on natural numbers.

Exercise: Prove that $\mathbb{N}$ is indeed the smallest set for which (1) and (2) hold. Hint: Prove by induction that if S satisfies (1) and (2) and $n \in \mathbb{N}$ then $n \in S$.

Exercise: Prove that $n \leq m$ iff $\text{leq}(n,m)$.

[1] We will later say that $\mathbb{N}$ is the smallest set that is closed under the two derivation rules.

4 Inference Rules

If inductive definitions are as simple as the examples nat(n) and leq(n, m) then it is straightforward to write them down in the way we have done in the previous section. However, inductive definitions that involve programming languages tend to be more complex. The have both more rules and the rules themselves have more preconditions. As a result, it is more concise to formulate inductive definitions using *inference rules*.

Assume a given inductive definition with a rules of the form

$$P_1 \text{ and } \ldots \text{ and } P_n \implies P\ .$$

From now on, we will write such rules as

$$\frac{\begin{array}{c}P_1\\ \ldots\\ P_n\end{array}}{P}\ (Rule_1)$$

We call $P_1, \ldots, P_n$ the premises of the rule and we call P the conclusion of the rule. If $n = 0$ then the rules does not have premises and we call it a leaf or an axiom.

Let us consider our two examples again. The judgment nat(n) is define by the following two rules.

$$\frac{}{\text{nat}(0)}\ N_1 \qquad\qquad \frac{\text{nat}(n)}{\text{nat}(n+1)}\ N_2$$

The judgment leq(n, m) can is defined by the following rules.

$$\frac{}{\text{leq}(0,0)}\ L_1 \qquad\qquad \frac{\text{leq}(n,m)}{\text{leq}(n+1,m+1)}\ L_2 \qquad\qquad \frac{\text{leq}(n,m)}{\text{leq}(n,m+1)}\ L_3$$

5 Derivations

If we inductively apply the inference rules then we obtain a *derivation* of a judgment. A derivation can be seen as a tree with axioms (or leaf rules) at the leaves. It is a proof that a particular judgment holds.

For example, we can derive the judgment nat(3) as follows.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\text{nat}(0)}\ N_1}{\text{nat}(1)}\ N_2}{\text{nat}(2)}\ N_2}{\text{nat}(3)}\ N_2$$

The derivation proofs that 3 is a natural number. Since we only have one or zero premises in our rules, the derivation is a degenerated tree with only one branch.

We can read such a derivation either from top to bottom or from bottom to top. If we start at the bottom then we construct the derivation starting with the judgment nat(3) that we would like to derive. Since $3 \neq 0$ the only rule we can apply is N_2. We then have to find a derivation for the premise nat(2) of the rule and so on.

Exercise: Use the rules L_1, L_2, and L_3 to derive the judgment leq$(3, 4)$.

6 Induction and Derivations

We have now seen to seemingly different interpretations of inductive rules. The first interpretation was that the rules define the smallest set that is closed under application of the rules (like $\mathbb{N}$ for the judgment nat$(\cdot)$). The second interpretation was that a judgment holds iff there is a derivation of that judgment using the rules.

Fortunately, it turns out that both interpretations are identical. In other words, there is a derivation of a judgment $P(o_1, \ldots, o_n)$ iff $\vec{o} \in S$ where S is the smallest set that is closed under the rules of the judgment.

7 Modes of a Judgment

As mentioned in the introduction, it is possible to define functions using rules. For every inductive definition we can define the *characteristic function* that determines for a given input if the judgment holds for this input or not. Consider again the inductive definition of of leq(n, m). We can turn it into a function LEQ : Nat $\to$ Nat $\to$ Bool that evaluates to true for inputs n, m iff leq(n, m).

```
LEQ(0,0) = true
LEQ(n,m) = (* What rule to implement? *)
```

But we run into a problem. To get an efficient and easily-implementable function we would like that at most one rule is applicable for a given input. Otherwise, we would need to backtrack in the translation of the inductive definition into a recursive function. So we define

$$\frac{}{\text{leq}(0,0)}\ L_1' \qquad\qquad \frac{\text{leq}(n,m)}{\text{leq}(n+1,m+1)}\ L_2' \qquad\qquad \frac{\text{leq}(0,m)}{\text{leq}(0,m+1)}\ L_3'$$

Now the direct translation yields the following function.

```
LEQ(0,0)       = true
LEQ(0,m+1)     = LEQ(0,m)
LEQ(n+1,m+1) = LEQ(n,m)
LEQ(_,_)       = false
```

Note that the last case can be changed to `LEQ(n+1,0) = false`.

In addition to the characteristic function, inference rules can sometimes also be turned into other functions by interpreting some arguments of the judgments as an input and other arguments as the output. We call such a view on the judgment the *mode of a judgment.* Some judgments allow for different modes.

Consider for example the judgment $\mathsf{add}(n_1, n_2, n)$ which that expresses that $n_1 + n_2 = n$. It is defined by the following inference rules.

$$\frac{}{\mathsf{add}(0, n, n)}\ A_1 \qquad\qquad \frac{\mathsf{add}(m, n, k)}{\mathsf{add}(m+1, n, k+1)}\ A_2$$

Like in the previous example, we could use the mode that all three arguments of the judgment are inputs. However, we know that addition is a function, that is, for a given natural numbers n_1 and n_2, there is at exactly on n such that $\mathsf{add}(n_1, n_2, n)$. So we can consider another mode at which the first two arguments of the judgment are the inputs and the third argument is the output. The rule $A1$ then reads at follows. Given the arguments 0 and n, we return n. The rule $A2$ reads: Given the arguments $m + 1$ and n we recursively apply the judgment to m and n, and obtain k such that $\mathsf{add}(m, n, k)$. We then return $k + 1$. In this way, the judgment describes the usual recursive addition function.

Exercise: Show that the rules L'_1, L'_2, and L'_3 are equivalent to the rules L_1, L_2, and L_3. Hint: You need to show that there exists a derivation of $\mathsf{leq}(n, m)$ using the primed rules iff there is a derivation of $\mathsf{leq}(n, m)$ using the original rules.

Exercise: Define the characteristic function of the judgment $\mathsf{add}(m, n, k)$.

8 Computing the Set of a Judgment

Sometimes we are also interested in computing the set of all objects for which a judgment holds. For example, for the judgment $\mathsf{live}(\ell, x)$ we would like to compute a set $S \subseteq D \times D$ such that

$$(\ell, x) \in S \iff \mathsf{live}(\ell, x)\ .$$

Here, we assume again that live is defined by the rules L_1, L_2, and L_3. However, we assume that the rules are restricted to some domain $D = [0, n] \subseteq \mathbb{N}$ that is a downward-closed subset of the natural numbers. (This example is slightly contrived but illustrates the idea.)

To compute S, we will start with the empty set $S_0 = \emptyset$ and successively construct sets $S_0, S_1, \dots, S_k$ so that $S_i \subseteq S_{i+1}$. To construct S_{i+1}, we will apply one of the rules to the elements of S_i. We will if there is no way to generate a new element using the rules.

Assume for instance, we have $D = \{0, 1, 2\}$. Then the process could look as follows.

S_0	$\emptyset$	apply L_1
S_1	$\{(0,0)\}$	apply L_2 to $(0,0)$
S_2	$\{(0,0),(1,1)\}$	apply L_2 to $(1,1)$
S_3	$\{(0,0),(1,1),(2,2)\}$	apply L_3 to $(0,0)$
S_4	$\{(0,0),(1,1),(2,2),(0,1)\}$	apply L_2 to $(0,1)$
S_5	$\{(0,0),(1,1),(2,2),(0,1),(1,2)\}$	apply L_3 to $(0,1)$
S_6	$\{(0,0),(1,1),(2,2),(0,1),(1,2),(0,2)\}$	

When we inspect the set S_6 we realize that we cannot generate new elements applying any of the rules. So we know $S = S_6$ and the computation ends.

References

[Har12] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2012.