Scribe: Shafquat Rahman, Dan Stowell | Lecturer: Andrew Myers

1 Well-founded relation

Definition: A well-founded relation is a binary relation $\prec$ on a set A such that there are no infinite descending chains. When $a \prec b$, element a is said to be a predecessor of element b. Note: A well-founded relation is irreflexive as otherwise there would be infinitely descending chains $\ldots \prec a \ldots \prec a$

The principle of well-founded relation states that

$$\frac{\forall e \in S \;.\; (\forall e' \prec e \;.\; P(e')) \Rightarrow P(e)}{\forall e \;.\; P(e)}$$

If e is such that there are no predecessors e', then it is necessary to simply prove $P(e)$. This corresponds to the base case of ordinary induction. The inductive step involves showing $P(e)$ holds assuming that $\forall e' \prec e.P(e)$ holds. For example, if the $\prec$ is defined as the successor relation $n \prec m \iff m = n+1$ on the natural numbers, the principle of well-founded induction specialises to ordinary mathematical induction. As another example, the $\prec$ is defined to be the subexpression relation such that $e' \prec e$ where e' is a subexpression of e. A proof by induction on this relation is called *structural induction*. Therefore, mathematical and structural induction are special cases of the more general well-founded induction.

Let us apply structural induction to the case of arithmetic evaluation.

Base case: Showing that the property holds for syntactic atoms, such as integers $1, 2, \ldots$ or a variable x
Inductive step: Showing that if $P(e)$ holds for subexpressions then it holds for e.

2 Rule induction

We want to be able to prove that certain properties hold for all the programs written in a programming language. In particular, we want to focus on the set of all semantically valid programs written in a programming language. What tools do we have to describe this set? Up to this point we have described semantics using rules, so we will use *rule induction* to write proofs about these semantics. The central insight of rule induction is that if you can prove that a property holds for the premise and conclusion of each rule in the set of rules, then the property must hold for all rule instances of those rules.

Consider an inference rule:

$$\frac{a_1 \Downarrow n_1 \quad a_2 \Downarrow n_2 \quad n_3 = n_1 + n_2}{a_1 + a_2 \Downarrow n_3}$$

Here is an example of a rule instance:

$$\frac{2 \Downarrow 2 \quad 3 \Downarrow 3}{2 + 3 \Downarrow 5}$$

In general, all rule instances take the following form, where x, x_i are possible set elements:

$$\frac{x_1 \quad x_2 \quad \ldots \quad x_n}{x}$$

If we want to prove a property $P(e)$ for all elements e of an inductively defined set, there is a natural choice for a well-founded relation on elements e. Consider that $e' \prec e$ if a derivation (proof tree) of e' is a subderivation of the smallest derivation of e. This relation cannot have an infinite descending chain because the proof trees must get smaller at each step along a chain, and proof trees have finite height.

This leads to the principle of rule induction, that we can prove a property $P(e)$ by showing that $P(e)$ holds assuming that $P(e')$ holds for all e' appearing in the premise of the rule deriving e.

Let I_R be the set of all rule instances for a set of rules R. Then we can express rule induction as:

$$\forall i \in I_R \,.\, P(i) \iff (\forall \frac{x_1, \ldots, x_n}{x} \in R \mid \{x_1, \ldots, x_n\} \subseteq I_R, \forall y \in \{x_1, \ldots, x_n\} \,.\, P(y) \Rightarrow P(y))$$

3 The rule operator

Let's look more closely at inductive definitions. Consider the *rule operator*, R defined by a set of rule instances U:

$$R(B) = \{x \mid \exists \{x_1, \ldots, x_n\} \subseteq B \,.\, \frac{x_1, \ldots, x_n}{x} \in U\}$$

The rule operator yields all the elements derivable from those rule instances in B. For example, $R(\emptyset)$ is the set of conclusions from the rule instances of the given axioms.

What properties should we expect of the set of elements A defined by the given rule instances?

- The set A should be *closed* under the rule operator. That is, applying the given rules should not derive any new elements not already in A. Formally, $R(A) \subseteq A$.
- The set A should be *consistent* with respect to the rule operator. That is, every element in A should be derivable from a rule. Formally, $A \subseteq R(A)$.

Putting these two properties together, we see that $A = R(A)$. In other words, the set A is a *fixed point* of the rule operator. The question, then, is *which* fixed point we want and how to obtain it.

An inductive definition is the *least fixed point* generated by the rule instances; that is, the smallest set. It turns out that we can Using the rule operator, we can construct the set of all valid programs A. Let $A_0 = \emptyset$, $A_1 = R(\emptyset)$, $A_2 = R(R(\emptyset))$, $\ldots$, $A_n = R^n(\emptyset)$. Then we can define the set of all valid programs as follows:

$$A = \bigcup_{n \in \omega} A_n$$

What's so special about A? It is closed under the rule operator (i.e., $R(A) \subseteq A$. Consider a rule instance with premises p and conclusion c. In order for the elements of p to be in A, there must be a derivation of height x for each element. Therefore, $p \subseteq A_x$. Applying the rule operator one more time will yield y from p: $y \in R(A_x)$. Both A_x and A_{x+1} are subsets of A by the definition given above. Therefore y must be in A, and A must be closed under R. Note that the set of premises p must be finite in order for this proof to hold.

The set A is also consistent (i.e., $A \subseteq R(A)$):

1. $y \in A$
2. $y \in A_x$ for some $x > 0$
3. $y \in R(A_{x-1})$
4. $\exists \frac{y_1, \ldots, y_n}{y} \,.\, \{y_1, \ldots, y_n\} \subseteq A_{x-1}$
5. $A_{x-1} \subseteq A$
6. $\{y_1, \ldots, y_n\} \subseteq A$
7. $y \in R(A)$
8. $A \subseteq R(A)$

If A closed and consistent under R, then $A = R(A)$. The set of all valid programs is the fixed point of the rule operator. Not only that, it is the *least* fixed point. We can use the monotonicity of R to prove this property. Suppose we have a fixed point B closed under R. Then $R(\emptyset) \subseteq R(B) = B$, $R(R(\emptyset)) \subseteq R(R(B)) = B$, and so on. At every stage, $A \subseteq B$.

It is interesting to note that the least fixed point is the set of all terminating evaluations in a programming language (since any terminating evaluation must have a finite derivation). The greatest fixed point is the set of all evaluations, even those ones which do not terminate.

4 Inductively defined function

Mathematically, *functions* can be defined as an ordered pair (a, b) or a mapping $a \mapsto b$. For example, we saw how to define the *free variable* function for the lambda calculus:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(e_0\ e_1) &= FV(e_0) \cup FV(e_1) \\ FV(\lambda x.\, e) &= FV(e) \setminus \{x\} \end{aligned}$$

We can interpret this function definition as a set of inference rules. The axiom is the case for x:

$$\frac{}{FV(x) = x}$$

The other rules are:

$$\frac{FV(e_1) = s_1 \quad FV(e_2) = s_2}{FV(e_1\ e_2) = s_3} \text{ (where } s_3 = s_1 \cup s_2)$$

$$\frac{FV(e) = s}{FV(\lambda x.\, e) = s'} \text{ (where } s' = s \setminus \{x\})$$

In order to ensure that f is a total function, it is enough to satisfy the following conditions, which happen to be true for FV:

- Every argument a matches exactly one conclusion.
- Rules respect a well-founded relation on a, that is the definition of $f(a)$ is in terms of other a' such that $a' \prec a$ for some well-founded relation $\prec$ on arguments.