Formal Methods / Nachum Dershowitz
Lecture 7, 2-05-2000
Notes by: Iddo Tzameret

Fixpoints

Introduction

We present here the concept of fixpoints, which is a method for proving properties of recursive programs. In discussing recursive problems the key problem is: what is the partial function f defined by a recursive program P? Let's look at the following recursive program, for example:

$$f := \lambda x, y.\ \text{if } x = y \text{ then } y + 1 \text{ else } f(x, f(x - 1, y + 1)) \qquad (1)$$

The question is what does it really mean?

This function, as well as many other computations may, in general, give results for some inputs, and may run indefinitely for other inputs. Thus, it defines a *partial function*. In order to be able to deal with *total functions* we use $\perp$ to denote an *undefined* value (which represents a non-terminating run). For any set S not containing $\perp$ let $S^\perp$ denote $S \cup \{\perp\}$.

Consider the partial function $f : D \to R$ to be a total function $f : D \to R^\perp$. However, since we shall consider composition of functions, an output of a function can be used also as an input. Hence, we should add $\perp$ to the domain as well: $f : D^\perp \to R^\perp$.

For example:

$f(0, y) \to 1$
$f(-1, y) \to f(-1, y)$
$f(1, y) \to f(0, f(-1, ...))$

Although the second term of $f(1, y)$ is $\perp$ (that is, $f(-1, ...)$), we could return the answer 1, since $f(0, y)$ yields 1 independently of y.

A *fixpoint* of such a definition as in (1) is a partial function that satisfies the equation. For example, the following are three fixpoints of (1):

$$f_1 = \lambda x, y.\ \text{if } x = y \text{ then } y + 1 \text{ else } x + 1$$

$$f_2 = \lambda x, y.\ \text{if } x \leq y \text{ then } y - 1 \text{ else } x + 1$$

$$f_3 = \lambda x, y.\ \text{if } x \geq y \wedge 2|(x-y) \text{ then } x+1 \text{ else } \bot$$

Among those three fixpoints, f_3 has a special property: for every x and y *in its domain* f_3 gives the same value as f_1 and f_2, and it can be shown that this happens for every other fixpoint of (1). In this case we say that f_3 is the *least fixpoint* of (1), that is, it is *less defined than or equal* to all other fixpoints of (1).

The fixpoint approach is one way to tackle the problem concerning the meaning of a recursive problem. This approach states that the function f, that is, the 'real meaning' of f, is the *least fixpoint*, denoted by f_∞. This is a reasonable approach since in practice many implementation of recursive programs indeed lead to the least fixpoint.

considering different types of fixpoints

In defining a function as above we are giving a formal definition which has no apparent *operational* meaning, that is, we do not know the way this function should actually be computed. What we certainly do know is that we should compute a fixpoint, since anything else would contradict the definition. However, we can consider many types of fixpoints, so the question arises: which one of them is the 'right' fixpoint?

Some of the fixpoint are not computable at all. It seems reasonable to compute the least fixpoint. We will see that given some reasonable conditions on the function definition, the least fixpoint exists and is unique and computable.

Besides the least fixpoint, there is another interesting fixpoint: the *optimal fixpoint*, which is a fixpoint function that has the maximal agreement with all the other fixpoints. That is, it returns $\bot$ for any input value for which other fixpoints return different values (or for the case they all return $\bot$); for every input value for which those fixpoints that are defined for it return *the same value*, the optimal fixpoint returns this value.

Most program languages compute something that is less than the least fixpoint (and never greater). ALGOL and HASKELL, for instance, compute the least fixpoint while LISP, SCHEME and C compute something which is less than the least fixpoint, that is, they have more points for which their result is undefined. Figure 1 shows the relations of different types of fixpoints; as we go upward from the central point (where the least fixpoint resides) the domain for which the fixpoints return a defined value is getting larger. Conversely, as we go downward from the least fixpoint, the undefined domain is getting larger, until we arrive at the least defined function Ω which is never defined.

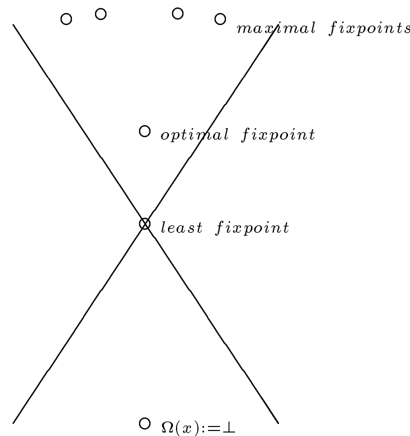

Figure 1: The relative place of different types of fixpoints. The upper part represents the fixpoints functions as their domains get larger (i.e., more input values in which the functions have a value other then $\perp$). Conversely, the lower part denotes the functions which are not fixpoints, and are less then the *least fixpoint* (according to the $\sqsubseteq$ order, introduced in the next section).

The Order of Functions

Partial ordering on $D^{\perp}$

In order to define the measure of definedness of a function (that is, the domain for which it has a value different from $\perp$), we first introduce the $\sqsubseteq$ order as a partial order on *domain elements* including $\perp$:

for all $x \in D^{\perp}$,

$\perp \sqsubseteq x$

$x \sqsubseteq x$.

Notice that distinct elements of D are unrelated by $\sqsubseteq$.

for $(D^{\perp})^n$, we define

$< x_1, ..., x_n > \sqsubseteq < y_1, ..., y_n > \Longleftrightarrow x_i \sqsubseteq y_i$, for all i. We are now ready to

define the $\sqsubseteq$ order as an order on *functions*:

$$f \sqsubseteq g \Longleftrightarrow \forall x \in D^{\perp}. f(x) \sqsubseteq g(x).$$

The $\sqsubseteq$ partial ordering is intended to correspond to the notion of *is less defined than or equal to.* This means that f is 'less than' g if f is not defined in some points where g is defined, and in the other points, both f and g are defined and have equal values. The $\sqsubseteq$ order also yields an *equality*

relation between functions: a function f is *equal* to g if f and g are defined for the same points and have the same values there, i.e, $f \sqsubseteq g$ and $g \sqsubseteq f$.

We denote the (n-ary) 'always undefined function' by
$\Omega(\bar{x}) := \perp$, for every $\bar{x} \in (D^{\perp})^n$.

Note that for any n-ary function f, we have $\Omega \sqsubseteq f$.

Monotonic functions

definition: *An n-ary function f from $(D^{\perp})^n$ to $R^{\perp}$ is monotonic if for all $\bar{x}, \bar{y} \in (D^{\perp})^n$:*

$$\bar{x} \sqsubseteq \bar{y} \Longrightarrow f(\bar{x}) \sqsubseteq f(\bar{y}).$$

This definition means intuitively that in order to be monotonic, we require from a function that if the input $\bar{x}$ is less defined than $\bar{y}$, then the output $f(\bar{x})$ is also less defined than $f(\bar{y})$.

Composition of monotonic functions is also monotonic, since if f and g are monotonic, then $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$, which in turn implies $g(f(x)) \sqsubseteq g(f(y))$.

Strict Functions

A function $f : (D^{\perp})^n \rightarrow (R^{\perp})$ is *strict* when
$f(x_1, ..., x_n) = \perp$ if $x_i = \perp$ for some $1 \leq i \leq n$.

Lemma: *Every strict function is monotonic.*

Exercise: Prove the lemma. (Hint: by contradiction, assume that $f(x_1, ..., x_n)$ is strict but not monotonic.)

The $\sqsubseteq$ order enables us to define the *least fixpoint* more formally: the least fixpoint of some definition of a function is the smallest fixpoint in the partial order of $\sqsubseteq$. It should be noted that the least fixpoint is computable in contrast to the optimal fixpoint. We shall see next a way to compute the least fixpoint, and we'll prove that given a continuous function(al), this computation indeed yields the *unique* least fixpoint.

Examples:

1. **Constants**: Every 0-ary function, i.e., a constant, is monotonic.

2. **if − then − else** is monotonic. We assume the following:

   if T then A else ... $\mapsto A$

   if F then ... else $A \mapsto A$

   if $\perp$ then ... else ... $\mapsto \perp$

   Exercise: Show that this definition of the if-then-else function is monotonic although it is *not* strict (consider the three cases where the premise is *true, false and* $\perp$).

3. We also assume $\perp + n \mapsto \perp$ for all natural n, as well as $\perp = \perp \mapsto \perp$.

4. **Identity**: The identity n-ary function mapping $\bar{x}$ into itself, is monotonic.

5. **Ω**: $\Omega(x) := \perp$ is monotonic.

The First Recursion Theorem (Kleene)

Let us turn back to our first example of a function definition: $f := \lambda x, y.$ if $x = y$ then $y + 1$ else $f(x, f(x - 1, y + 1))$. It is possible to see that this definition is of the form: $f := B[f, f]$. Hence, the function(al) B, is a function that maps the set of functions of $D^{\perp} \times D^{\perp} \rightarrow R^{\perp}$ into *itself*; that is, B takes a function f of the form $D^{\perp} \times D^{\perp} \rightarrow R^{\perp}$, and returns the function $B[f, f]$, which is of the same form, i.e., $D^{\perp} \times D^{\perp} \rightarrow R^{\perp}$.

In order to compute the least fixpoint of the definition we will build a chain of functions such that:

$$f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq ...$$

Where f_0 is the minimal function of $\sqsubseteq$, i.e., Ω. $f_1 = B[f_0, f_0]$, $f_2 = B[f_1, f_1]$, and so on. The least fixpoint will be the *limit* of this chain, which is actually the *least upper bound* of the chain. But let us turn first to more formal definitions.

Least upper bound

We denote an arbitrary *sequence* $f_0, f_1, f_2, ...$ by $\{f_i\}$. The sequence $\{f_i\}$ is called a *chain* only if $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq ...$. We say that f is the *upper bound of* $\{f_i\}$, if $f_i \sqsubseteq f$ for all f_i. If in addition $f \sqsubseteq g$ for every upper bound g, then we say that f is the *least upper bound of* $\{f_i\}$. We also denote the least upper bound of $\{f_i\}$ by $\lim_{i \rightarrow \infty} f_i$.

Lemma: *Every chain $\{f_i\}$ has a least upper bound.*

proof.
Consider the following function:

$$f(x) = \begin{cases} \bot & \forall i. f_i(x) = \bot; \\ a & \exists i. f_i(x) = a \neq \bot. \end{cases}$$

We shall show that f is the least upper bound of the chain $\{f_i\}$. first, we show that f is an *upper bound* of $\{f_i\}$. For this purpose we need to show that for all i $f_i \sqsubseteq f$; indeed, for all i and for all x there are two choices:

(1) $f_i(x) = \bot$ then clearly, $f_i(x) \sqsubseteq f(x)$.

(2) $f_i(x) = a$, for some a, and from the definition of f we also have that $f(x) = a$. Hence, $f_i(x) \sqsubseteq f(x)$.

From (1) and (2) we conclude that for all i $f_i \sqsubseteq f$.

Second, we show that f is the *least* upper bound of $\{f_i\}$. Let g be an upper bound of $\{f_i\}$; we need to show that for all x $f(x) \sqsubseteq g(x)$. Since g is an upper bound we have that for all i $f_i(x) \sqsubseteq g(x)$. If $f(x) = \bot$ then $f(x) \sqsubseteq g(x)$. Otherwise, $f(x) = a$, where, from the definition of f we know that $a = f_i(x)$ for some i. Since, $f_i(x) \sqsubseteq g(x)$ for all i, we have also that $f(x) \sqsubseteq g(x)$. Therefore $f \sqsubseteq g$.

Q.E.D.

Monotonicity and continuity

We now turn to describe two interesting properties of function(als), monotonicity and continuity.

monotonicity

A function(al), $B[f]$, is said to be monotonic if $f \sqsubseteq g$ implies that $B[f] \sqsubseteq B[g]$.

continuity

A monotonic function(al), $B[f]$, is said to be continuous if for any chain of functions $\{f_i\}$, we have

$$\lim_{i \to \infty} B[f_i] = B[\lim_{i \to \infty} f_i]$$

Note that $\{f_i\}$ is a *chain* and thus have a least upper bound, according to the last lemma. Hence, the term $\lim_{i \to \infty} f_i$ which denotes the least upper bound of the chain $\{f_i\}$, is legitimate here. Furthermore, since B is *monotonic* we have $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq ... \Longrightarrow B[f_0] \sqsubseteq B[f_1] \sqsubseteq B[f_2] \sqsubseteq ...$. Therefore, $\{B[f_i]\}$

is also a chain and its limit, $\lim_{i\to\infty} B[f_i]$, that is, its least upper bound, really exists. Hence, the continuity property can be written also as:

$$lub\{B[f_i]\} = B[lub\{f_i\}]$$

Where lub denotes the *least upper bound.*

Examples:

1. $\bot \cdot 0 = 0, \;\; 0 \cdot \bot = 0$ is not strict but continuous.

2. Composition of *if-then-else*, constants, monotonic base functions, f as a function variable, is always *continuous.*

   In general we have the following rule:

   Proposition (without a proof [1]): *any function(al) B defined by composition of monotonic functions and a function variable f, is continuous.*

We see next that the two properties we just described, monotonicity and continuity, are the conditions that allow us to deduce that the limit of a chain $\{f_i\}$, i.e., its lub, is indeed the least fixpoint we are seeking.

definition: *Given a function(al) B, we say that f is a fixpoint of B if $B[f] = f$, that is, if B maps the function f into itself. If for any other fixpoint of B, g, we have that $f \sqsubseteq g$, then we say that f is the least fixpoint of B.*

Notice that a function(al) B have at most one least fixpoint, f, since if both f and g are least fixpoints of B, then $f \sqsubseteq g$ and $g \sqsubseteq f$ and thus, $f \equiv g$.

Theorem 1 (**First Recursion Theorem** (**Kleene**)) *Every continuous function(al) $B[f]$ has a unique least fixpoint,* $\lim_{i\to\infty} B^i[\Omega]$.

proof.

1. Since B is continuous it is also monotonic and therefore we have for all i

   $$f_i \sqsubseteq B[f_i]$$

   , thus we have $f_0 = \Omega \sqsubseteq f_1 = B[\Omega] \sqsubseteq f_2 = B^2[\Omega] \sqsubseteq \ldots$.

[1] See "Mathematical Theory of Computation", by Zohar Manna, chap. 5, for a proof.

2. Denote $\lim_{i\to\infty} B^i[\Omega]$ by f_∞. We will prove that f_∞ is a fixpoint of B.

$$f_\infty = \lim_{i\to\infty} B^i[\Omega] = \lim_{i\to\infty} B^{i+1}[\Omega] = \lim_{i\to\infty} B[B^i[\Omega]] \tag{1}$$

   from continuity of B we have

$$\lim_{i\to\infty} B[B^i[\Omega]] = B[\lim_{i\to\infty} B^i[\Omega]] = B[f_\infty] \; . \tag{2}$$

   Hence, from (1) and (2) we have that $f_\infty = B[f_\infty]$.

3. Let g be a fixpoint of B, we'll show that for all i $f_i \sqsubseteq g$. By induction on i: $f_0 = \Omega \sqsubseteq g$, since Ω is the minimal function of $\sqsubseteq$. Assume that $f_i \sqsubseteq g$. Then, $f_{i+1} = B[f_i] \sqsubseteq B[g]$, from monotonicity of B and induction hypothesis. Because g is a fixpoint $g = B[g]$, and thus we have $f_{i+1} \sqsubseteq g$.

   This implies that g is an *upper bound* of the chain $\{B^i[\Omega]\}$ (we already proved in (1) that it is a chain). Since $\{B^i[\Omega]\}$ is a chain it has a least upper bound, which we denoted earlier by f_∞, and thus we conclude that $f_\infty \sqsubseteq g$ for any fixpoint g. Q.E.D.

Exercise:
Consider the following functional:

$$B[f](x) := \quad \text{if } x = 0 \text{ then } 1 \text{ else } f(x+1)$$

Where, x is over the naturals including $\perp$.
1) What are the fixpoints of B?
2) What is the *least fixpoint* among those fixpoints?

Computation Rules

What determines the operational semantics of different programming languages and distinguish in general the nature of a language from that of other languages, is its *rewriting strategy.* That is, the order by which terms are being replaced by other terms according to a rewrite rule.

A popular rewriting order, i.e., a computation rule, is *"call-by-value": innermost-leftmost.* That is, compute first the innermost and left-most subterm that has a rule. This, computation rule is adopted by C, PASCAL, SCHEME and LISP, for instance. However, this method leads to something less than the least fixpoint.

Another computation rule is the *"call by name": outermost-leftmost.* This rule gives the least fixpoint for a continuous function(al). HASKELL

and ALGOL use this rule. It is implemented by rewriting the outermost subterm, every step.

Also, another computation rule is the *"outermost-fair"*, which is similar to the "call by name" rule, but takes care that no subterms which are positioned right to the leftmost term will be neglected forever.

Example:

Consider the term: F(F(1), 0) + F(F(2),F(3)). If we use *innermost-leftmost* comutation rule, we shall first replace the term F(1). If, however, we use *outermost-leftmost* rule, we shall first replace the term F(F(1), 0), and then the outermost term would be F(1), and so on.