1 Set Operators

Recall from last time that a *rule instance* is of the form

$$\frac{X_1 \; X_2 \; \ldots \; X_n}{X}, \tag{1}$$

where X and the X_i are members of some set S. From this we defined a monotone set operator

$$R(B) \triangleq \{X \mid \{X_1, X_2, \ldots, X_n\} \subseteq B \text{ and } \frac{X_1 \; X_2 \; \ldots \; X_n}{X} \text{ is a rule instance}\}. \tag{2}$$

Recall that *monotone* means that if $B \subseteq C$, then $R(B) \subseteq R(C)$. We listed two desirable properties of the set $A \subseteq S$ defined by R, namely:

- A should be R-*consistent*: $A \subseteq R(A)$. We would like this to hold because we would like every element of A to be included in A only as a result of applying a rule.
- A should be R-*closed*: $R(A) \subseteq A$. We would like this to hold because we would like every element that the rules say should be in A to actually be in A.

These two properties together say that $A = R(A)$, or in other words, A should be a fixpoint of R. We then asked two natural questions:

- Does R actually have a fixpoint?
- Is the fixpoint unique? If not, which one should we take?

Before we answer these questions, let us look at a few examples to illustrate these concepts. Suppose that $S = \{x, y, z\}$ and consider the set operator R defined by the following rules:

$$\frac{}{x} \qquad \frac{x}{y} \qquad \frac{x \quad y}{z}$$

The R-closed and R-consistent subsets of S are as given by the following table:

B	$R(B)$	R-*closed*	R-*consistent*
$\{\}$	$\{x\}$	*No*	*Yes*
$\{x\}$	$\{x, y\}$	*No*	*Yes*
$\{y\}$	$\{x\}$	*No*	*No*
$\{z\}$	$\{x\}$	*No*	*No*
$\{x, y\}$	$\{x, y, z\}$	*No*	*Yes*
$\{x, z\}$	$\{x, y\}$	*No*	*No*
$\{y, z\}$	$\{x\}$	*No*	*No*
$\{x, y, z\}$	$\{x, y, z\}$	*Yes*	*Yes*

The operator R has just one fixed point, namely $\{x, y, z\}$.

For another example, consider a set operator R' defined by the following rules:

$$\frac{}{x} \qquad \frac{y}{z} \qquad \frac{x \quad z}{y}$$

The R'-closed and R'-consistent subsets of S are as given by the following table:

B	$R'(B)$	R'-*closed*	R'-*consistent*
$\{\}$	$\{x\}$	*No*	*Yes*
$\{x\}$	$\{x\}$	*Yes*	*Yes*
$\{y\}$	$\{x, z\}$	*No*	*No*
$\{z\}$	$\{x\}$	*No*	*No*
$\{x, y\}$	$\{x, z\}$	*No*	*No*
$\{x, z\}$	$\{x, y\}$	*No*	*No*
$\{y, z\}$	$\{x, z\}$	*No*	*No*
$\{x, y, z\}$	$\{x, y, z\}$	*Yes*	*Yes*

This operator has two fixed points, $\{x\}$ and $\{x, y, z\}$.

2 Least Fixpoints

It turns out that every monotone set operator has at least one fixpoint. As the second example above shows, it may have many fixed points.

But among all its fixpoints, it has a unique minimal one with respect to set inclusion $\subseteq$; that is, a fixpoint that is a subset of all other fixpoints of R. We call this the *least fixpoint* of R, and we take A to be this set.

The least fixpoint of R can be defined in two different ways, "from below" and "from above":

$$A_* \triangleq \bigcup \{R^n(\varnothing) \mid n \geq 0\} \quad = \quad R(\varnothing) \cup R(R(\varnothing)) \cup R(R(R(\varnothing))) \cup \cdots \tag{3}$$

$$A^* \triangleq \bigcap \{B \subseteq S \mid R(B) \subseteq B\}. \tag{4}$$

The set A_* is the union of all sets of the form $R^n(\varnothing)$, the sets obtained by applying R some finite number of times to the empty set. It consists of all elements of S that are in $R^n(\varnothing)$ for some $n \geq 0$.

The set A^* is the intersection of all the R-closed subsets of S. It consists of all elements of S that are in every R-closed set.

We will show that $A_* = A^*$ and that this set is the least fixpoint of R, so we will take $A \triangleq A_* = A^*$.

3 The Knaster–Tarski Theorem

The fact that $A_* = A^*$ is a special case of a more general theorem called the *Knaster–Tarski theorem*. It states that any monotone set operator R has a unique least fixpoint, and that this fixpoint can be obtained either "from below" by iteratively applying R to the empty set, or "from above" by taking the intersection of all R-closed sets.

For general monotone operators R, the "from below" construction may require iteration through transfinite ordinals. However, the operators R defined from rule systems as described above are *chain-continuous* (definition below). This is a stronger property than monotonicity. It guarantees that the "from below" construction converges to a fixpoint after only ω steps, where ω is the first transfinite ordinal.

3.1 Monotone, Continuous, and Finitary Operators

A set operator $R : 2^S \to 2^S$ is said to be

- *monotone* if $B \subseteq C$ implies $R(B) \subseteq R(C)$;
- *chain-continuous* if for any chain of sets $\mathcal{C}$,

$$R(\bigcup \mathcal{C}) \quad = \quad \bigcup \{R(B) \mid B \in \mathcal{C}\}$$

  (a *chain* is a set $\mathcal{C}$ of subsets of S that is linearly ordered by the set inclusion relation $\subseteq$; that is, for all $B, C \in \mathcal{C}$, either $B \subseteq C$ or $C \subseteq B$);
- *finitary* if for any set C, the value of $R(C)$ is determined by the values of $R(B)$ for finite subsets $B \subseteq C$ in the following sense:

$$R(C) \quad = \quad \bigcup \{R(B) \mid B \subseteq C,\ B \text{ finite}\}.$$

One can show

1. every rule-based operator of the form (2) is finitary;
2. every finitary operator is chain-continuous (in fact, the converse holds as well);
3. every chain-continuous operator is monotone.

The proofs of 1, 2, and 3 are fairly straightforward and we will leave them as exercises. The converse of 2 requires transfinite induction and is more difficult.

3.2 Proof of the Knaster–Tarski Theorem for Chain-Continuous Operators

Let us prove the Knaster–Tarski theorem in the special case of chain-continuous operators, which will allow us to avoid introducing transfinite ordinals (not that they are not worth introducing!), and that is all we need to handle rule-based inductive definitions.

Theorem (Knaster–Tarski) Let $R : 2^S \to 2^S$ be a chain-continuous set operator, and let A_* and A^* be defined as in (3) and (4), respectively. Then $A_* = A^*$, and this set is the $\subseteq$-least fixpoint of R.

Proof. The theorem follows from the following two facts:

(i) For every n and every R-closed set B, $R^n(\varnothing) \subseteq B$. This can be proved by induction on n. It follows that $A_* \subseteq A^*$.

(ii) A_* is a fixpoint of R, thus it is R-closed. As A^* is contained in every R-closed set we have $A^* \subseteq A_*$. Moreover, because every fixed point is R-closed, A_* is contained in every fixed point. That is, A_* is the least fixed point of R.

For (i), let B be an R-closed set. We proceed by induction on n. The basis for $n = 0$ is $\varnothing \subseteq B$, which is trivially true. Now suppose $R^n(\varnothing) \subseteq B$. We have

$$\begin{aligned} R^{n+1}(\varnothing) &= R(R^n(\varnothing)) \\ &\subseteq R(B) \quad \text{by the induction hypothesis and monotonicity} \\ &\subseteq B \quad \text{since } B \text{ is } R\text{-closed.} \end{aligned}$$

We conclude that for all n and all R-closed sets B, $R^n(\varnothing) \subseteq B$, therefore

$$A_* \quad = \quad \bigcup \{R^n(\varnothing) \mid n \geq 0\} \quad \subseteq \quad \bigcap \{B \subseteq S \mid R(B) \subseteq B\} \quad = \quad A^*.$$

For (ii), we want to show that $R(A_*) = A_*$. It can be proved by induction on n that the sets $R^n(\varnothing)$ form a chain:

$$\varnothing \quad \subseteq \quad R(\varnothing) \quad \subseteq \quad R^2(\varnothing) \quad \subseteq \quad R^3(\varnothing) \quad \subseteq \quad \cdots$$

We have $\varnothing \subseteq R(\varnothing)$ trivially, and by monotonicity, if $R^n(\varnothing) \subseteq R^{n+1}(\varnothing)$, then

$$R^{n+1}(\varnothing) \quad = \quad R(R^n(\varnothing)) \quad \subseteq \quad R(R^{n+1}(\varnothing)) \quad = \quad R^{n+2}(\varnothing).$$

Now by chain-continuity,

$$R(A_*) \quad = \quad R(\bigcup_{n\geq 0} R^n(\varnothing)) \quad = \quad \bigcup_{n\geq 0} R(R^n(\varnothing)) \quad = \quad \bigcup_{n\geq 0} R^{n+1}(\varnothing) \quad = \quad A_*.$$

as required. □

4 Rule Induction

Let us use our newfound wisdom on well-founded induction and least fixpoints of monotone maps to prove some properties of the reduction rules.

4.1 Example: CBV Preserves Closedness

Theorem 1. *If $e \to e'$ under the CBV reduction rules, then $FV(e') \subseteq FV(e)$. In other words, CBV reductions cannot introduce any new free variables.*

Proof. By induction on the CBV derivation of $e \to e'$. There is one case for each CBV rule, corresponding to each way $e \to e'$ could be derived.

Case 1 $\dfrac{e_1 \to e_1'}{e_1\, e_2 \to e_1'\, e_2}$.

We assume that the desired property is true of the premise—this is the induction hypothesis—and we wish to prove under this assumption that it is true for the conclusion. Thus we are assuming that $FV(e_1') \subseteq FV(e_1)$ and wish to prove that $FV(e_1'\, e_2) \subseteq FV(e_1\, e_2)$.

$$\begin{aligned} FV(e_1'\, e_2) &= FV(e_1') \cup FV(e_2) \quad \text{by the definition of } FV \\ &\subseteq FV(e_1) \cup FV(e_2) \quad \text{by the induction hypothesis} \\ &= FV(e_1\, e_2) \quad \text{again by the definition of } FV. \end{aligned}$$

Case 2 $\dfrac{e_2 \to e_2'}{v\, e_2 \to v\, e_2'}$.

This case is similar to Case 1, where now $e_2 \to e_2'$ is used in the induction hypothesis.

Case 3 $\dfrac{}{(\lambda x.\, e)\, v \to e\{v/x\}}$.

There is no induction hypothesis for this case, since there is no premise in the rule; thus this case constitutes the basis of our induction. We wish to show, independently of any inductive assumption, that $FV(e\{v/x\}) \subseteq FV((\lambda x.\, e)\, v)$.

This case requires a lemma, stated below, to show that $FV(e\{v/x\}) \subseteq (FV(e) - \{x\}) \cup FV(v)$. Once that is shown, we have

$$\begin{aligned} FV(e\{v/x\}) &\subseteq (FV(e) - \{x\}) \cup FV(v) \quad \text{by the lemma to be proved} \\ &= FV(\lambda x.\, e) \cup FV(v) \quad \text{by the definition of } FV \\ &= FV((\lambda x.\, e)\, v) \quad \text{again by the definition of } FV. \end{aligned}$$

We have now considered all three rules of derivation for the CBV λ-calculus, so the theorem is proved. □

Lemma 2. *$FV(e\{v/x\}) \subseteq (FV(e) - \{x\}) \cup FV(v)$ (this lemma is used by case 3 in the above theorem).*

Proof. By structural induction on e. There is one case for each clause in the definition of the substitution operator. We have assumed previously that values are closed terms, so $FV(v) = \varnothing$ for any value v; but actually we do not need this for the proof, and we do not assume it.

Case 1 $e = x$.

$$\begin{aligned} FV(e\{v/x\}) &= FV(x\{v/x\}) \\ &= FV(v) \quad \text{by the definition of the substitution operator} \\ &= (\{x\} - \{x\}) \cup FV(v) \\ &= (FV(x) - \{x\}) \cup FV(v) \quad \text{by the definition of } FV \\ &= (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

Case 2 $e = y$, $y \neq x$.

$$\begin{aligned} FV(e\{v/x\}) &= FV(y\{v/x\}) \\ &= FV(y) \quad \text{by the definition of the substitution operator} \\ &= \{y\} \quad \text{by the definition of } FV \\ &\subseteq (\{y\} - \{x\}) \cup FV(v) \\ &= (FV(y) - \{x\}) \cup FV(v) \quad \text{again by the definition of } FV \\ &= (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

Case 3 $e = e_1\, e_2$.

$$\begin{aligned} FV(e\{v/x\}) &= FV((e_1\, e_2)\{v/x\}) \\ &= FV(e_1\{v/x\}\, e_2\{v/x\}) \quad \text{by the definition of the substitution operator} \\ &\subseteq (FV(e_1) - \{x\}) \cup FV(v) \cup (FV(e_2) - \{x\}) \cup FV(v) \quad \text{by the induction hypothesis} \\ &= ((FV(e_1) \cup FV(e_2)) - \{x\}) \cup FV(v) \\ &= (FV(e_1\, e_2) - \{x\}) \cup FV(v) \quad \text{again by the definition of } FV \\ &= (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

Case 4 $e = \lambda x.\, e'$.

$$\begin{aligned} FV(e\{v/x\}) &= FV((\lambda x.\, e')\{v/x\}) \\ &= FV(\lambda x.\, e') \quad \text{by the definition of the substitution operator} \\ &= FV(\lambda x.\, e') - \{x\} \quad \text{because } x \notin FV(\lambda x.\, e') \\ &\subseteq (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

Case 5 $e = \lambda y.\, e'$, $y \neq x$. This is the most interesting case, because it involves a change of bound variable. Using the fact $FV(v) = \varnothing$ for values v would give a slightly simpler proof. Let v be a value and z a variable such that $z \neq x$, $z \notin FV(e')$, and $z \notin FV(v)$.

$$\begin{aligned} FV(e\{v/x\}) &= FV((\lambda y.\, e')\{v/x\}) \\ &= FV(\lambda z.\, e'\{z/y\}\{v/x\}) \quad \text{by the definition of the substitution operator} \\ &= FV(e'\{z/y\}\{v/x\}) - \{z\} \quad \text{by the definition of } FV \\ &= ((((FV(e') - \{y\}) \cup FV(z)) - \{x\}) \cup FV(v)) - \{z\} \quad \text{by the induction hypothesis twice} \\ &= (((FV(\lambda y.\, e') \cup \{z\}) - \{x\}) \cup FV(v)) - \{z\} \quad \text{by the definition of } FV \\ &= ((FV(\lambda y.\, e') - \{x\}) \cup FV(v) \cup \{z\}) - \{z\} \\ &= (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

□

There is a subtle point that arises in Case 5. We said at the beginning of the proof that we would be doing structural induction on e; that is, induction on the well-founded subterm relation $<$. This was a lie. Because of the change of bound variable necessary in Case 5, we are actually doing induction on the relation of subterm modulo α-equivalence:

$$e <_\alpha e' \quad \triangleq \quad \exists e''\ e'' < e' \wedge e =_\alpha e''.$$

But a moment's thought reveals that this relation is still well-founded, since α-reduction does not change the size or shape of the term, so we are ok.

4.2 Example: Agreement of Big-Step and Small-Step Semantics

We can express the idea that the two semantics should agree on terminating executions by connecting the relations $\to$ and $\Downarrow$:

$$\langle a, \sigma\rangle \to_a \overline{n} \iff \langle a, \sigma\rangle \Downarrow_a n \tag{5}$$

$$\langle b, \sigma\rangle \to_b \overline{t} \iff \langle b, \sigma\rangle \Downarrow_b t \tag{6}$$

$$\langle c, \sigma\rangle \to \langle \mathsf{skip}, \sigma'\rangle \iff \langle c, \sigma\rangle \Downarrow \sigma', \tag{7}$$

where $\overline{\mathit{false}} = \mathsf{false}$ and $\overline{\mathit{true}} = \mathsf{true}$. These can all be proved using induction. We can prove (5) and (6) as lemmas separately, then use these lemmas to prove (7).

Let us just show a few of the cases for (7). To prove the forward implication of (7), we can use structural induction on c. The converse requires induction on the derivation of the big-step evaluation.

Suppose we are given $\langle c, \sigma\rangle \Downarrow \sigma'$. The form of the derivation $\langle c, \sigma\rangle \Downarrow \sigma'$ depends on the form of c.

- Case skip. In this case we know $\sigma = \sigma'$, and trivially, $\langle \mathsf{skip}, \sigma\rangle \to \langle \mathsf{skip}, \sigma\rangle$.

- Case $x := a$. In this case we know from the premises that $\langle a, \sigma\rangle \Downarrow n$ for some n and that $\sigma' = \sigma[n/x]$.

  We will need a lemma to the effect that $\langle a, \sigma\rangle \to_a \overline{n}$ implies that $\langle x := a, \sigma\rangle \to \langle x := \overline{n}, \sigma\rangle$. This result can be proved easily using an induction on the number of steps in the derivation $\langle a, \sigma\rangle \to_a \overline{n}$. Given this and (5), we have that $\langle x := a, \sigma\rangle \to \langle x := \overline{n}, \sigma\rangle$ and $\langle x := \overline{n}, \sigma\rangle \xrightarrow{1} \langle \mathsf{skip}, \sigma[n/x]\rangle$, so $\langle x := a, \sigma\rangle \to \langle \mathsf{skip}, \sigma[n/x]\rangle$.

- Case $\mathsf{while}\ b\ \mathsf{do}\ c$, where $\langle b, \sigma\rangle \Downarrow_b \mathit{false}$. Then $\sigma = \sigma'$. In small-step semantics, we have an initial step

$$\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \xrightarrow{1} \langle \mathsf{if}\ b\ \mathsf{then}\ (c\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}, \sigma\rangle. \tag{8}$$

  By (6), $\langle b, \sigma\rangle \to_b \mathsf{false}$. We need another lemma to the effect that if $\langle b, \sigma\rangle \to_b t$, then

$$\langle \mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ c', \sigma\rangle \to \langle \mathsf{if}\ t\ \mathsf{then}\ c\ \mathsf{else}\ c', \sigma\rangle. \tag{9}$$

  In this case $t = \mathsf{false}$, thus (8) becomes $\langle \mathsf{skip}, \sigma\rangle$ as desired.

- Case $\mathsf{while}\ b\ \mathsf{do}\ c$, where $\langle b, \sigma\rangle \Downarrow_b \mathit{true}$. This is the most interesting case in the entire proof. In small-step semantics, we have the initial step (8), as in the previous case. From (9) with $t = \mathsf{true}$, (8) becomes $\langle c\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle$. We need one more lemma for stitching together small-step executions:

$$\langle c_1, \sigma\rangle \to \langle \mathsf{skip}, \sigma''\rangle \ \wedge\ \langle c_2, \sigma''\rangle \to \langle \mathsf{skip}, \sigma'\rangle \implies \langle c_1\ ;\ c_2, \sigma\rangle \to \langle \mathsf{skip}, \sigma'\rangle. \tag{10}$$

  This can be proved by induction on the number of steps.

  Now, because $\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \Downarrow \sigma'$ and $\langle b, \sigma\rangle \Downarrow_b \mathit{true}$, we know that $\langle c, \sigma\rangle \Downarrow \sigma''$ and $\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma''\rangle \Downarrow \sigma'$ for some σ''. Furthermore, because these two derivations are subderivations of the derivation $\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \Downarrow \sigma'$, the induction hypothesis gives us $\langle c, \sigma\rangle \to \langle \mathsf{skip}, \sigma''\rangle$ and $\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma''\rangle \to \sigma'$. Using (10), we have $\langle c\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \to \langle \mathsf{skip}, \sigma'\rangle$.

  We could not have used structural induction for this proof, because the induction step involved relating an evaluation of the command $\mathsf{while}\ b\ \mathsf{do}\ c$ to a different evaluation of the same command rather than to an evaluation of a subexpression.

5 Remark

These proofs may seem rather tedious, and one may wonder why we are doing them in such detail. But of course, this is exactly the point. Formal reasoning about the semantics of the λ-calculus, including such seemingly complicated notions as reductions and substitutions, can be reduced to the mindless application of a few simple rules. There is no hand-waving or magic involved. There is nothing hidden, it is all right there in front of you. To the extent that we can do this for real programming languages, we will be better able to understand what is going on.