Chapter 2

2.2.6 It is clear that R' is reflexive. Suppose R'' is some other reflexive relation containing R, which is to say $(s, s) \in R''$ for all $s \in S$ and $R \subseteq R''$. Hence

$$R' = R \cup \{(s, s) \mid s \in S\} \subseteq R'',$$

so R' is smallest.

2.2.7 First note that R is cumulative, i.e.

$$n \leq m \Rightarrow R_n \subseteq R_m. \tag{1}$$

(We leave it as an unproven fact that for any sequence s, the two propositions '$s_n \leq s_{n+1}$ for all n' and '$s_n \leq s_m$ for all $n \leq m$' are equivalent.)

Suppose $s\ R^+\ t$ and $t\ R^+\ u$. Since R^+ is a union, there must be m, n such that $s\ R_m\ t$ and $t\ R_n\ u$. From (1) it follows that $s\ R_{\max\{m,n\}}\ t$ and $t\ R_{\max\{m,n\}}\ u$, so by construction $s\ R_{\max\{m,n\}+1}\ u$, which implies $s\ R^+\ u$. Hence R^+ is transitive.

Suppose R'' is some other transitive relation containing R. Thus $R_0 = R \subseteq R''$, and $R_n \subseteq R'' \Rightarrow R_{n+1} \subseteq R''$ since R'' is transitive, so $R_n \subseteq R''$ for all n. Hence $R^+ \subseteq R''$, so R^+ is smallest.

2.2.8 Suppose P is preserved by R. Since $P(s) \Rightarrow P(s)$, adding (s, s) to R can never invalidate the preservation of P by R. In a similar fashion, if $P(s)$, $s\ R\ t$, and $t\ R\ u$, then $P(t)$ and so $P(u)$by the preservation of P by R, so adding (s, u) to R can do no harm. In conclusion, P is preserved by R^*.

Chapter 3

3.2.4 Let n_i be the size of S_i. Then $n_0 = 0$ and $n_{i+1} = 3 + 3n_i + n_i^3$, so $n_3 = 59\,439$.

3.2.5 Let F be the construction such that $S_{i+1} = F(S_i)$. We show that F is monotone with respect to inclusion, i.e.

$$A \subseteq B \Rightarrow F(A) \subseteq F(B). \tag{2}$$

Suppose $A \subseteq B$ and $t \in F(A)$. Then either

- $\mathtt{t} \in \{\mathtt{true}, \mathtt{false}, \mathtt{0}\}$, in which case $\mathtt{t} \in F(B)$, or
- $\mathtt{t} \in \{\mathtt{succ\ t}_1, \mathtt{pred\ t}_1, \mathtt{iszero\ t}_1\}$ for some $\mathtt{t}_1 \in A$, in which case $\mathtt{t}_1 \in B$, so $\mathtt{t} \in F(B)$, or
- $\mathtt{t} = \mathtt{if\ t}_1\ \mathtt{then\ t}_2\ \mathtt{else\ t}_3$ for $\{\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3\} \subseteq A$, in which case $\{\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3\} \subseteq B$, so $t \in F(B)$.

Hence $F(A) \subseteq F(B)$.

We can now see that S is cumulative, since $S_0 \subseteq S_1$ and also $S_i \subseteq S_{i+1} \Rightarrow S_{i+1} \subseteq S_{i+2}$, where the first fact follows from S_0 being the empty set, and the second fact is a substitution instance of (2).

3.3.4 Let $\bar{P}$ be the extension of the property P, i.e. $\bar{P} = \{\mathtt{s} \in \mathcal{T} \mid P(\mathtt{s})\} \subseteq \mathcal{T}$. The premise of structural induction assert that $\bar{P}$ satisfies conditions 1–3 in definition 3.2.1, and since $\mathcal{T}$ is the smallest such set, $\mathcal{T} \subseteq \bar{P}$. Thus $\bar{P} = \mathcal{T}$, so $P(\mathtt{s})$ for all $\mathtt{s} \in \mathcal{T}$, which is the conclusion of structural induction.

For induction on depth or size we argue as follows. Let f be either *depth* or *size*. Note that the inverse image of f partitions $\mathcal{T}$, i.e. if we let $F_i = \{\mathtt{s} \in \mathcal{T} \mid f(\mathtt{s}) = i\}$ then $\bigcup_i F_i = \mathcal{T}$ and $F_i \cap F_j = \varnothing$ whenever $i \neq j$. The premise of induction on depth or size may then be reformulated thusly: if $F_i \subseteq \bar{P}$ for every $i < j$, then $F_j \subseteq \bar{P}$. By strong induction on $\mathbb{N}$, we can conclude that $F_i \subseteq \bar{P}$ for all i, so $\mathcal{T} \subseteq \bar{P}$.

(As a side note, $F_i = S_i$ when $f = \mathit{depth}$.)

3.5.5 Structural induction.

3.5.10

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\mathtt{t} \longrightarrow^* \mathtt{t}'} \qquad \frac{}{\mathtt{t} \longrightarrow^* \mathtt{t}'} \qquad \frac{\mathtt{t} \longrightarrow^* \mathtt{t}' \qquad \mathtt{t}' \longrightarrow^* \mathtt{t}''}{\mathtt{t} \longrightarrow^* \mathtt{t}''}$$

3.5.13 E-FUNNY1 transforms positive conditionals into non-deterministic choice. Determinacy of one-step evaluation (3.5.4), as well as Uniqueness of normal forms (3.5.11), fail, since e.g. both

$$\mathtt{if\ true\ then\ true\ else\ false} \longrightarrow \mathtt{true}, \text{ and}$$
$$\mathtt{if\ true\ then\ true\ else\ false} \longrightarrow \mathtt{false}.$$

On the other hand, Every value is a normal form (3.5.7), If $\mathtt{t}$ is a normal form, then $\mathtt{t}$ is a value (3.5.8), and Termination of evaluation (3.5.12), remain valid.

E-FUNNY2 makes possible early evaluation of the then-branch of a conditional. Determinacy of one-step evaluation fails, since e.g. both

$\mathtt{if\ true\ then\ pred\ succ\ 0\ else\ 0} \longrightarrow \mathtt{if\ true\ then\ 0\ else\ 0}$, and
$\mathtt{if\ true\ then\ pred\ succ\ 0\ else\ 0} \longrightarrow \mathtt{pred\ succ\ 0}$.

Every value is a normal form (3.5.7), If $\mathtt{t}$ is a normal form, then $\mathtt{t}$ is a value (3.5.8), and Termination of evaluation (3.5.12), remain valid. Uniqueness of normal forms also remains valid, but the proof is no longer trivial.

Let $\mathtt{t}$ be a term, and $\mathtt{u}$ and $\mathtt{v}$ be normal forms. We prove that $\mathtt{t} \longrightarrow^* \mathtt{u}$ and $\mathtt{t} \longrightarrow^* \mathtt{v}$ implies $\mathtt{u} = \mathtt{v}$ by induction on $\mathtt{t}$.

- If $\mathtt{t} \in \{\mathtt{true}, \mathtt{false}, \mathtt{0}\}$, then clearly $\mathtt{t} = \mathtt{u} = \mathtt{v}$.
- If $\mathtt{t} = \mathtt{succ}\ \mathtt{t}_1$, then each step of the trace $\mathtt{t} \longrightarrow^* \mathtt{u}$ ($\mathtt{t} \longrightarrow^* \mathtt{v}$) must be derived by E-SUCC from steps of a corresponding trace $\mathtt{t}_1 \longrightarrow^* \mathtt{u}_1$ ($\mathtt{t}_1 \longrightarrow^* \mathtt{v}_1$), and $\mathtt{u}_1$ ($\mathtt{v}_1$) must be a value, since $\mathtt{u}$ ($\mathtt{v}$) is. By induction hypothesis $\mathtt{u}_1 = \mathtt{v}_1$, so $\mathtt{u} = \mathtt{v}$.
- If $\mathtt{t} = \mathtt{pred}\ \mathtt{t}_1$ the situation is similar, except the last step of the trace is either E-PREDZERO or E-PREDSUCC, depending on $\mathtt{u}_1 = \mathtt{v}_1$.
- If $\mathtt{t} = \mathtt{iszero}\ \mathtt{t}_1$, it is again similar, with the last step being either E-ISZEROZERO or E-ISZEROSUCC depending on $\mathtt{u}_1 = \mathtt{v}_1$.
- If $\mathtt{t} = \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$ the situation gets more interesting. As a subsequence of the trace $\mathtt{t} \longrightarrow^* \mathtt{u}$ ($\mathtt{t} \longrightarrow^* \mathtt{v}$), there must exist a trace $\mathtt{t}_1 \longrightarrow^* \mathtt{u}_1$ ($\mathtt{t}_1 \longrightarrow^* \mathtt{v}_1$) lifted by E-IF – to which we can apply the induction hypothesis to conclude $\mathtt{u}_1 = \mathtt{v}_1$ – followed by either E-IFTRUE or E-IFFALSE. We can divide the first trace into two parts,

  $$\mathtt{t} \overset{a}{\longrightarrow}^* \mathtt{t}' \overset{b}{\longrightarrow} \mathtt{t}'' \overset{c}{\longrightarrow}^* \mathtt{u},$$

  where the b step is this last application of either E-IFTRUE or E-IFFALSE, and similar for the second trace. The steps in a which are not part of the $\mathtt{t}_1$-trace must be a trace

  $$\mathtt{t}_2 \overset{d}{\longrightarrow}^* \mathtt{t}_2'$$

  lifted by E-FUNNY2. We now proceed by cases:

  * If $\mathtt{u}_1 = \mathtt{true}$ then

    $$\mathtt{t}_2 \overset{d}{\longrightarrow}^* \mathtt{t}_2' = \mathtt{t}'' \overset{c}{\longrightarrow}^* \mathtt{u}$$

    is a trace, and we have a similar one for $\mathtt{v}$, to which we may apply the induction hypothesis to conclude $\mathtt{u} = \mathtt{v}$.

* If $u_1 = \texttt{false}$ then we do the same thing with

$$t_3 = t'' \xrightarrow{c}{}^{*} u.$$

3.5.14 Idea: Since each form of term is matched by the conclusion of at most one rule, evaluation traces can never fork.

3.5.16 Proposition: For all terms t, $t \longrightarrow^* t'$ for a stuck term t' in the old sense if and only if $t \longrightarrow^* \texttt{wrong}$ in the new sense.

Proof idea: The conclusions of the added rules match exactly the forms of terms which are not covered by the old (normal forms), but are also not values (stuck terms).

3.5.17 By induction on t:

- If $t \in \{\texttt{true}, \texttt{false}, \texttt{0}\}$ then $t \longrightarrow^* t$ and also $t \Downarrow t$.
- If $t = \texttt{succ}\ t_1$ and $t_1 \longrightarrow^* v_1$ with v_1 a numeric value, then $t_1 \Downarrow v_1$ by induction hypothesis, $t \longrightarrow^* \texttt{succ}\ v_1$ by E-Succ, and $t \Downarrow \texttt{succ}\ v_1$ by B-Succ.
- If $t = \texttt{succ}\ t_1$ and $t_1 \not\longrightarrow^* v_1$ with v_1 a numeric value, then $t_1 \not\Downarrow v_1$ by induction hypothesis, so t is stuck in both systems.
- If $t = \texttt{pred}\ t_1$ and $t_1 \longrightarrow^* v_1$ with v_1 a numeric value, then $t_1 \Downarrow v_1$ by induction hypothesis. Depending on the form of v_1 we can apply either E-PredZero respectively B-PredZero, or E-PredSucc respectively B-PredSucc, to show $t \longrightarrow^* v$ and $t \Downarrow v$, for v of the appropriate form.
- If $t = \texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3$ and $t_1 \longrightarrow^* \texttt{true}$ and $t_2 \longrightarrow^* v_2$ where v_2 is a value, then $t_1 \Downarrow \texttt{true}$ and $t_2 \Downarrow v_2$ by induction hypotheses, $t \longrightarrow^* v_2$ by E-IfTrue, and $t \Downarrow v_2$ by B-IfTrue.
- The remaining cases are similar.

3.5.18 Dismiss the old rules E-If, E-IfTrue and E-IfFalse in favour of the following new rules:

$$\frac{t_2 \longrightarrow t_2'}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \longrightarrow \texttt{if}\ t_1\ \texttt{then}\ t_2'\ \texttt{else}\ t_3} \qquad \text{(E-IfThen)}$$

$$\frac{t_3 \longrightarrow t_3'}{\texttt{if}\ t_1\ \texttt{then}\ v_2\ \texttt{else}\ t_3 \longrightarrow \texttt{if}\ t_1\ \texttt{then}\ v_2\ \texttt{else}\ t_3'} \qquad \text{(E-IfElse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if}\ t_1\ \texttt{then}\ v_2\ \texttt{else}\ v_3 \longrightarrow \texttt{if}\ t_1'\ \texttt{then}\ v_2\ \texttt{else}\ v_3} \qquad \text{(E-IfCond)}$$

$$\texttt{if true then}\ v_2\ \texttt{else}\ v_3 \longrightarrow v_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then}\ v_2\ \texttt{else}\ v_3 \longrightarrow v_3 \qquad \text{(E-IfFalse)}$$

Chapter 4

4.2.1 Because it prevents tail-call optimization, and fills the stack with useless exception handlers; useless, because only the innermost one will ever be utilized.

A better way to do it is to use an option, either by rewriting `eval1` directly, or by proxying:

```
let rec eval t =
  let eval1' t =
    try Some(eval1 t)
    with NoRuleApplies → None in
  match eval1' t with
      Some(t') → eval t'
    | None → t
```

Another perhaps less elegant solution is an imperative loop:

```
let eval t =
  let t = ref t in
  try while true do
    t := eval1 !t
  done
  with NoRuleApplies → ();
  !t
```

4.2.2
```
let rec eval t = match t with
    v when isval v → v
  | TmIf(_,t1,t2,t3) → begin match eval t1 with
      TmTrue(_) → eval t2
    | TmFalse(_) → eval t3
    | _ → raise NoRuleApplies end
  | TmSucc(fi,t1) → TmSucc(fi,eval(t1))
  | TmPred(_,t1) → begin match eval t1 with
      TmZero(fi) → TmZero(fi)
    | TmSucc(_,nv1) when isnumericalval nv1 → nv1
    | _ → raise NoRuleApplies end
  | TmIsZero(_,t1) → begin match eval t1 with
      TmZero(fi) → TmTrue(dummyinfo)
    | TmSucc(_,nv1) when isnumericalval nv1 → TmFalse(dummyinfo)
    | _ → raise NoRuleApplies end
  | _ → raise NoRuleApplies (* Unreachable *)
```

Chapter 5

5.2.1 `or = λb. λc. b tru c;`
`not = λb. b fls tru;`

5.2.2 `succ' = λn. λs. λz. n s (s z);`

5.2.3 `times' = λm. λn. λs. λz. m (n s) z;`

5.2.4 `exp = λm. λn. n (times m) c1;`

5.2.5 `sub = λm. λn. n prd m;`

5.2.6 This of course depends on the reduction strategy, but for a rough estimate we can let the strategy be whatever I feel like. We assume, in contrast to the implementation, that nominal definitions are metamathematical constructs. Finally, let $\langle \mathtt{f}, \mathtt{s} \rangle$ stand for `λb. b f s`. Then we compute:

$$
\begin{aligned}
\mathtt{pair\ f\ s} &\longrightarrow^{2} \langle \mathtt{f}, \mathtt{s} \rangle, \\
\mathtt{fst}\ \langle \mathtt{f}, \mathtt{s} \rangle &\longrightarrow^{4} \mathtt{f}, \\
\mathtt{snd}\ \langle \mathtt{f}, \mathtt{s} \rangle &\longrightarrow^{4} \mathtt{s}, \\
\mathtt{plus}\ \mathtt{c}_m\ \mathtt{c}_n &\longrightarrow^{6} \mathtt{c}_{m+n}, \\
\mathtt{zz} &\longrightarrow^{2} \langle \mathtt{c}_0, \mathtt{c}_0 \rangle, \\
\mathtt{ss}\ \langle \mathtt{c}_m, \mathtt{c}_n \rangle &\longrightarrow^{1} \mathtt{pair\ (snd}\ \langle \mathtt{c}_m, \mathtt{c}_n \rangle\mathtt{)\ (plus\ c}_1\ \mathtt{(snd}\ \langle \mathtt{c}_m, \mathtt{c}_n \rangle\mathtt{)} \\
&\longrightarrow^{2+4+6+4} \langle \mathtt{c}_n, \mathtt{c}_{n+1} \rangle, \\
\mathtt{prd}\ \mathtt{c}_m &\longrightarrow^{1} \mathtt{fst\ (c}_m\ \mathtt{ss\ zz)} \\
&\longrightarrow^{2+2} \mathtt{fst\ (ss}^m\ \langle \mathtt{c}_0, \mathtt{c}_0 \rangle\mathtt{)} \\
&\longrightarrow^{17m} \mathtt{fst}\ \langle \mathtt{c}_{m-1}, \mathtt{c}_m \rangle \\
&\longrightarrow^{4} \mathtt{c}_{m-1}, \\
\mathtt{sub}\ \mathtt{c}_m\ \mathtt{c}_n &\longrightarrow^{2} \mathtt{c}_n\ \mathtt{prd}\ \mathtt{c}_m \\
&\longrightarrow^{2} \mathtt{prd}^n\ \mathtt{c}_m \\
&\longrightarrow^{(17m+9)n} \mathtt{c}_{m-n}.
\end{aligned}
$$

In conclusion, it takes roughly $17mn + 9n + 4$ evaluation steps to subtract n from m.

(It should be noted, though, that with a better encoding `prd` c_m can be evaluated in a constant number of steps, making subtraction linear. Thus the awful performance gleaned above is an artifact of that particular encoding, rather than encodings in general.)

5.2.7 `equal = λm. λn. and (iszro (sub m n)) (iszro (sub n m));`

5.2.8
```
/* [x, y, z] = λc. λn. c x (c y (c z n)); */
nil = λc. λn. n; /* Same as fls and c0. */
cons = λh. λt. λc. λn. c h (t c n);
head = λl. l tru fls;
tail = λl. fst (l
  (λh. λp. pair (snd p) (cons h (snd p)))
  (pair nil nil));
isnil = λl. l (λh. λt. fls) tru;
```

5.2.9 Because to use test, one would need to protect the then and else clauses from premature evaluation, complicating the presentation. Here's what it could look like:

```
factorial' = fix (λfactorial. λn.
  test (equal n c0)
    (λ_. c1)
    (λ_. times n (factorial (prd n)))
  nil);
```

5.2.10
```
churchnat = λn. λs. λz.
  fix (λf. λn. if iszero n then z else s (f (pred n))) n;
```

5.2.11 Sum is not a particularly good example, since it is more simply expressed as `λl. l plus c0`, but here goes:

```
sum = fix (λsum. λl.
  if realbool (isnil l) then c0
    else plus (head l) (sum (tail l)));
```

5.3.3 By induction on $\mathtt{t}$:

- If $\mathtt{t} = \mathtt{x}$, then $|FV(x)| = 1 = size(x)$.
- If $\mathtt{t} = \lambda\mathtt{x}.\mathtt{t}_1$, then $|FV(\lambda\mathtt{x}.\mathtt{t}_1)| \leq |FV(\mathtt{t}_1)| \leq size(\mathtt{t}_1) < size(\lambda\mathtt{x}.\mathtt{t}_1)$.
- If $\mathtt{t} = \mathtt{t}_1\mathtt{t}_2$, then $|FV(\mathtt{t}_1\mathtt{t}_2)| \leq |FV(\mathtt{t}_1)|+|FV(\mathtt{t}_2)| \leq size(\mathtt{t}_1)+size(\mathtt{t}_2) \leq size(\mathtt{t}_1\mathtt{t}_2)$.

5.3.6 Let $\mathtt{f}$ be a redex-free term, i.e. one of the form $\mathtt{x}$, $\mathtt{x\ f}$ or $\lambda\mathtt{x.\ f}$. I assume 'lazy' mean call-by-name here.

$$(\lambda\mathtt{x.\ t}_{12})\ \mathtt{t}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{12} \qquad \text{(F}\beta\text{-Abs)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \qquad \text{(F}\beta\text{-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1\ \mathtt{t}_2'} \qquad \text{(F}\beta\text{-App2)}$$

$$(\lambda \mathtt{x}.\ \mathtt{t}_{12})\ \mathtt{t}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{12} \qquad \text{(NO-ABS)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \qquad \text{(NO-APP1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{f}_1\ \mathtt{t}_2 \longrightarrow \mathtt{f}_1\ \mathtt{t}_2'} \qquad \text{(NO-APP2)}$$

$$(\lambda \mathtt{x}.\ \mathtt{t}_{12})\ \mathtt{t}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{12} \qquad \text{(CBN-ABS)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \qquad \text{(CBN-APP1)}$$

5.3.7

$$\mathtt{x}\ \mathtt{t}_2 \longrightarrow \mathtt{wrong} \qquad \text{(E-WRONG1)}$$

$$\mathtt{t}_1\ \mathtt{x} \longrightarrow \mathtt{wrong} \qquad \text{(E-WRONG2)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{wrong}}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{wrong}} \qquad \text{(E-WRONG3)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{wrong}}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{wrong}} \qquad \text{(E-WRONG4)}$$

5.3.8

$$\mathtt{v} \Downarrow \mathtt{v} \qquad \text{(B-VALUE)}$$

$$\frac{\mathtt{t}_1 \Downarrow \lambda \mathtt{x}.\ \mathtt{t}_{21} \qquad \mathtt{t}_2 \Downarrow \mathtt{v}_2 \qquad [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \Downarrow \mathtt{v}}{\mathtt{t}_1\ \mathtt{t}_2 \Downarrow \mathtt{v}} \qquad \text{(B-APP)}$$

Chapter 6

6.1.1
```
c0 = λ λ 0;
c2 = λ λ 1 (1 0);
plus = λ λ λ λ 3 1 (2 1 0);
fix = λ (λ 1 (λ 1 1 0)) (λ 1 (λ 1 1 0));
foo = (λ λ 0) (λ 0);
```