Figure 5.3

IV. Text file Domain $f \in File = Record^*$

V. File system

Domain $s \in File\text{-}system = Id \rightarrow File$

Operations

$access : Id \times File\text{-}system \rightarrow File$

$access = \lambda(i,s).\ s(i)$

$update : Id \times File \times File\text{-}system \rightarrow File\text{-}system$

$update = \lambda(i,f,s).\ [\, i \mapsto f\,]s$

VI. Open file

Domain $p \in Openfile = Record^* \times Record^*$

Operations

$newfile : Openfile$

$newfile = (nil,nil)$

$copyin : File \rightarrow Openfile$

$copyin = \lambda f\,.\,(nil,f)$

$copyout : Openfile \rightarrow File$

$copyout = \lambda p.$ "appends $fst(p)$ to $snd(p)$— defined later"

$forwards : Openfile \rightarrow Openfile$

$forwards = \lambda(front, back).\ null\ back \rightarrow (front, back)$
$\quad [\!]\ ((hd\ back)\ cons\ front, (tl\ back))$

$backwards : Openfile \rightarrow Openfile$

$backwards = \lambda(front, back).\ null\ front \rightarrow (front, back)$
$\quad [\!]\ (tl\ front, (hd\ front)\ cons\ back)$

$insert : Record \times Openfile \rightarrow Openfile$

$insert = \lambda(r\ , (front, back)).\ null\ back \rightarrow (front,\ r\ cons\ back)$
$\quad [\!]\ ((hd\ back)\ cons\ front),\ r\ cons\ (tl\ back))$

$delete : Openfile \rightarrow Openfile$

$delete = \lambda(front, back).\ (front,\ (null\ back \rightarrow back\ [\!]\ tl\ back))$

$at\text{-}first\text{-}record : Openfile \rightarrow Tr$

$at\text{-}first\text{-}record = \lambda(front, back).\ null\ front$

$at\text{-}last\text{-}record : Openfile \rightarrow Tr$

$at\text{-}last\text{-}record = \lambda(front, back).\ null\ back \rightarrow true$
$\quad [\!]\ (null\ (tl\ back) \rightarrow true\ [\!]\ false)$

$isempty : Openfile \rightarrow Tr$

$isempty = \lambda(front, back).\ (null\ front)\ and\ (null\ back)$

Figure 5.4

Abstract syntax:

P ∈ Program-session
S ∈ Command-sequence
C ∈ Command
R ∈ Record
I ∈ Identifier

P ::= **edit** I **cr** S
S ::= C **cr** S | **quit**
C ::= **newfile** | **moveforward** | **moveback** | **insert** R | **delete**

Semantic algebras:

I. Truth values
Domain $t \in Tr$
Operations

$true, false : Tr$

$and : Tr \times Tr \rightarrow Tr$

II. Identifiers
Domain $i \in Id$ = Identifier

III. Text records
Domain $r \in Record$

IV. - VI. defined in Figure 5.3

VII. Character Strings (defined in Example 3.3 of Chapter 3)

VIII. Output terminal log
Domain $l \in Log = String^*$

Valuation functions:

P: Program-session $\rightarrow File\text{-}system \rightarrow (Log \times File\text{-}system)$
$\mathbf{P}[\![\textbf{edit I cr S}]\!] = \lambda s.$ let $p = copyin\ (access(\ [\![\mathrm{I}]\!], s))$ in
("edit I" *cons* $fst(\mathbf{S}[\![\mathrm{S}]\!]p\ ),\ update(\ [\![\mathrm{I}]\!],\ copyout\ (snd\ (\mathbf{S}[\![\mathrm{S}]\!]p\ )),\ s)$)

S: Command-sequence $\rightarrow Openfile \rightarrow (Log \times Openfile)$
$\mathbf{S}[\![\mathrm{C\ cr\ S}]\!] = \lambda p.$ let $(l', p') = \mathbf{C}[\![\mathrm{C}]\!]p$ in $((l'\ cons\ fst(\mathbf{S}[\![\mathrm{S}]\!]p')), snd\ (\mathbf{S}[\![\mathrm{S}]\!]p'))$
$\mathbf{S}[\![\textbf{quit}]\!] = \lambda p.$ ("quit" *cons* *nil* , p)

Figure 5.4 (continued)

```
C: Command →Openfile→(String × Openfile)
  C[[newfile]] = λp. ("newfile", newfile)
  C[[moveforward]] = λp. let (k',p') = isempty (p) →("error: file is empty", p)
                                 [] ( at-last-record(p) →("error: at back already", p)
                                      [] ("", forwards(p)) )
                           in ("moveforward" concat k', p'))
  C[[moveback]] = λp. let (k',p') = isempty (p) →("error: file is empty", p)
                                 [] ( at-first-record(p) →("error: at front already", p) )
                                      [] ("", backwards(p))
                           in ("moveback" concat k', p')
  C[[insert R]] = λp. ("insert R", insert (R[[R]], p))
  C[[delete]] = λp. let (k',p') = isempty (p) →("error: file is empty", p)
                                 [] ("", delete (p))
                           in ("delete" concat k', p')
```

of this list. The equation for S[[C cr S]] deserves a bit of study. It says to:

1. Evaluate C[[C]]p to obtain the next log entry l' plus the updated open file p'.
2. Cons l' to the log list and pass p' onto S[[S]].
3. Evaluate S[[S]]p' to obtain the meaning of the remainder of the program, which is the rest of the log output plus the final version of the updated

is created. The + domain builder attaches a "type tag" to a value. The *Store* domain becomes:

$$Store = Id \rightarrow Storable\text{-}value$$

The type tags are stored with the truth values and numbers for later reference. Since storable values are used in arithmetic and logical expressions, type errors

Figure 5.5

V. Values that may be stored
Domain $v \in Storable\text{-}value = Tr + Nat$

VI. Values that expressions may denote
Domain $x \in Expressible\text{-}value = Storable\text{-}value + Errvalue$
where $Errvalue = Unit$
Operations

$check\text{-}expr : (Store \rightarrow Expressible\text{-}value) \times$
$(Storable\text{-}value \rightarrow Store \rightarrow Expressible\text{-}value) \rightarrow (Store \rightarrow Expressible\text{-}value)$
$f_1\ check\text{-}expr\ f_2 = \lambda s.$ cases $(f_1\ s)$ of
is$Storable\text{-}value(v) \rightarrow (f_2\ v\ s)$
[] is$Errvalue() \rightarrow$ in$Errvalue()$
end

VII. Input buffer
Domain $i \in Input = Expressible\text{-}value^*$
Operations

$get\text{-}value : Input \rightarrow (Expressible\text{-}value \times Input)$
$get\text{-}value = \lambda i.\ null\ i \rightarrow (\text{in}Errvalue(), i)$ [] $(hd\ i, tl\ i)$

Figure 5.5 (continued)

VIII. Output buffer

Domain $o \in Output = (Storable\text{-}value + String)^*$

Operations

$empty : Output$

$empty = nil$

$put\text{-}value : Storable\text{-}value \times Output \rightarrow Output$

$put\text{-}value = \lambda(v,o).\ \text{in}Storable\text{-}value(v)\ cons\ o$

$put\text{-}message : String \times Output \rightarrow Output$

$put\text{-}message = \lambda(t,o).\ \text{in}String(t)\ cons\ o$

IX. Store

Domain $s \in Store = \text{Id} \rightarrow Storable\text{-}value$

Operations

$newstore : Store$

$access : Id \rightarrow Store \rightarrow Storable\text{-}value$

$update : Id \rightarrow Storable\text{-}value \rightarrow Store \rightarrow Store$

X. Program State

Domain $a \in State = Store \times Input \times Output$

XI. Post program state

Domain $z \in Post\text{-}state = OK + Err$

where $OK = State$

and $Err = State$

Operations

$check\text{-}result : (Store \rightarrow Expressible\text{-}value) \times (Storable\text{-}value \rightarrow State \rightarrow Post\text{-}state_\perp) \rightarrow (State \rightarrow Post\text{-}state_\perp)$

$f\ check\text{-}result\ g = \lambda(s,i,o).\ \text{cases}\ (f\ s)\ \text{of}$
$\quad \text{is}Storable\text{-}value(v) \rightarrow (g\ v\ (s,i,o))$
$\quad [\,]\ \text{is}Errvalue() \rightarrow \text{in}Err(s, i, put\text{-}message(\text{"type error"}, o))\ \text{end}$

$check\text{-}cmd : (State \rightarrow Post\text{-}state_\perp) \times (State \rightarrow Post\text{-}state_\perp) \rightarrow (State \rightarrow Post\text{-}state_\perp)$

$h_1\ check\text{-}cmd\ h_2 = \lambda a.\ \text{let}\ z = (h_1\ a)\ \text{in cases}\ z\ \text{of}$
$\quad \text{is}OK(s,i,o) \rightarrow h_2(s,i,o)$
$\quad [\,]\ \text{is}Err(s,i,o) \rightarrow z\ \text{end}$

Figure 5.6

Abstract syntax:

P∈ Program
C∈ Command
E∈ Expression
I ∈ Id
N∈ Numeral

P ::= C.

C ::= C_1;C_2 | I:=E | **if** E **then** C_1 **else** C_2 | **read** I | **write** E | **diverge**

E ::= E_1+E_2 | E_1=E_2 | ¬ E | (E) | I | N | **true**

Semantic algebras:

I. Truth values (defined in Figure 5.1)

II. Natural numbers (defined in Figure 5.1)

III. Identifiers (defined in Figure 5.1)

IV. Character strings (defined in Example 3.5 of Chapter 3)

V. - XI. (defined in Figure 5.5)

Valuation functions:

P: Program→*Store* →*Input* →*Post-state*$_\perp$
P⟦C.⟧ = λ*s*.λ*i*. C⟦C⟧ (*s*, *i*, *empty*)

C: Command→*State* →*Post-state*$_\perp$
C⟦C_1;C_2⟧ = C⟦C_1⟧ *check-cmd* C⟦C_2⟧
C⟦I:=E⟧ = E⟦E⟧ *check-result* (λ*v*.λ(*s*,*i*,*o*). in*OK*((*update* ⟦I⟧ *v* *s*), *i*, *o*))
C⟦**if** E **then** C_1 **else** C_2⟧ = E⟦E⟧ *check-result*
(λ*v*.λ(*s*,*i*,*o*). cases *v* of
is*Tr*(*t*)→(*t* →C⟦C_1⟧ [] C⟦C_2⟧)(*s*,*i*,*o*)
[] is*Nat*(*n*)→in*Err*(*s*,*i*, *put-message*("bad test", *o*)) end)
C⟦**read** I⟧ = λ(*s*,*i*,*o*). let (*x*,*i'*) = *get-value*(*i*) in
cases *x* of
is*Storable-value*(*v*) → in*OK*((*update* ⟦I⟧ *v* *s*), *i'*, *o*)
[] is*Errvalue*() → in*Err*(*s*, *i'*, *put-message*("bad input", *o*)) end
C⟦**write** E⟧ = E⟦E⟧ *check-result* (λ*v*.λ(*s*,*i*,*o*). in*OK*(*s*, *i*, *put-value*(*v*,*o*)))
C⟦**diverge**⟧ = λ*a*. ⊥

Figure 5.6

Abstract syntax:

P∈ Program
C∈ Command
E∈ Expression
I ∈ Id
N∈ Numeral

P ::= C.
C ::= C_1;C_2 | I:=E | **if** E **then** C_1 **else** C_2 | **read** I | **write** E | **diverge**
E ::= E_1+E_2 | E_1=E_2 | ¬ E | (E) | I | N | **true**

Semantic algebras:

I. Truth values (defined in Figure 5.1)

II. Natural numbers (defined in Figure 5.1)

III. Identifiers (defined in Figure 5.1)

IV. Character strings (defined in Example 3.5 of Chapter 3)

V. - XI. (defined in Figure 5.5)

Valuation functions:

P: Program→*Store* →*Input* →*Post-state*$_\perp$
P⟦C.⟧ = λ*s*.λ*i*. C⟦C⟧ (*s*, *i*, *empty*)

C: Command→*State* →*Post-state*$_\perp$
C⟦C_1;C_2⟧ = C⟦C_1⟧ *check-cmd* C⟦C_2⟧
C⟦I:=E⟧ = E⟦E⟧ *check-result* (λ*v*.λ(*s*,*i*,*o*). in*OK*((*update* ⟦I⟧ *v* *s*), *i*, *o*))
C⟦**if** E **then** C_1 **else** C_2⟧ = E⟦E⟧ *check-result*
(λ*v*.λ(*s*,*i*,*o*). cases *v* of
is*Tr*(*t*)→(*t* →C⟦C_1⟧ [] C⟦C_2⟧)(*s*,*i*,*o*)
[] is*Nat*(*n*)→in*Err*(*s*,*i*, *put-message*("bad test", *o*)) end)
C⟦**read** I⟧ = λ(*s*,*i*,*o*). let (*x*,*i'*) = *get-value*(*i*) in
cases *x* of
is*Storable-value*(*v*) → in*OK*((*update* ⟦I⟧ *v* *s*), *i'*, *o*)
[] is*Errvalue*() → in*Err*(*s*, *i'*, *put-message*("bad input", *o*)) end
C⟦**write** E⟧ = E⟦E⟧ *check-result* (λ*v*.λ(*s*,*i*,*o*). in*OK*(*s*, *i*, *put-value*(*v*,*o*)))
C⟦**diverge**⟧ = λ*a*. ⊥

Figure 5.6 (continued)

```
E: Expression→Store→Expressible-value
  E[[E1+E2]] = E[[E1]] check-expr
          (λv.λs. cases v of
                  isTr(t)→inErrvalue()
                  [] isNat(n)→E[[E2]]s check-expr
                      (λv'.λs. cases v' of
                          isTr(t')→inErrvalue()
                          [] isNat(n')→inStorable-value(inNat(n plus n')) end)
                  end
  E[[E1=E2]] = "similar to above equation"
  E[[¬ E]] = E[[E]] check-expr
          (λv.λs. cases v of
                  isTr(t)→inStorable-value(inTr(not t))
                  [] isNat(n)→inErrvalue() end)
  E[[(E)]] = E[[E]]
  E[[I]] = λs. inStorable-value(access [[I]] s)
  E[[N]] = λs. inStorable-value(inNat(N[[N]]))
  E[[true]] = λs. inStorable-value(inTr(true))
N:Numeral→Nat (omitted)
```