1 Languages

1.1 Flexible Language

Flexible Language:

- Non deterministic.
- Type error means stuck.
- Without type annotation in lambda functions and mlet.
- Introduce the explicit substitution in mlet, and the substitution in lambdas, the difference.
- Characterize errors.
- Fix Index.

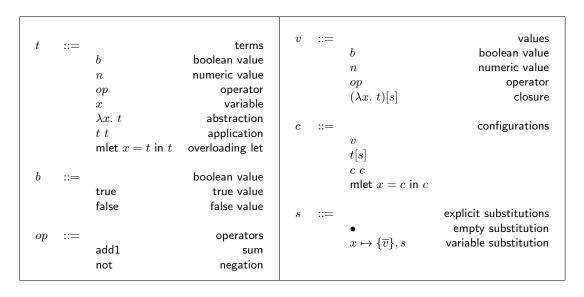


Figure 1: Syntax of the Flexible Language.

Definition 1 (\oplus). Given an environment s and a variable binding $x \mapsto v_1$, the operator \oplus is defined as follows:

$$s \oplus x \mapsto v_1 = \begin{cases} x \mapsto \{v_1\} & s = \emptyset \\ x \mapsto \{\overline{v}\} \cup \{v_1\}, s' & s = x \mapsto \{\overline{v}\}, s' \\ y \mapsto \{\overline{v}\}, s' \oplus x \mapsto v_1 & s = y \mapsto \{\overline{v}\}, s' \end{cases}$$

$$b[s] \longrightarrow b \qquad \qquad (\text{False})$$

$$n[s] \longrightarrow n \qquad \qquad (\text{Num})$$

$$op[s] \longrightarrow op \qquad \qquad (\text{Op})$$

$$x[x \mapsto \{\overline{v}\}, s] \longrightarrow v_i \qquad \qquad (\text{VarOk})$$

$$\frac{x \neq y}{x[y \mapsto \{\overline{v}\}, s] \longrightarrow x[s]} \qquad (\text{VarNext})$$

$$(\text{mlet } x = t_1 \text{ in } t_2)[s] \longrightarrow \text{mlet } x = t_1[s] \text{ in } t_2[s] \qquad (\text{LetSub})$$

$$(t_1 \ t_2)[s] \longrightarrow t_1[s] \ t_2[s] \qquad \qquad (\text{AppSub})$$

$$\text{mlet } x = v \text{ in } t_2[s] \longrightarrow t_2[x \mapsto v \oplus s] \qquad \qquad (\text{Let})$$

$$(\lambda x. \ t_2)[s] \ v \longrightarrow ([x \mapsto v]t_2)[s] \qquad \qquad (\text{App})$$

$$\text{add1} \ n \longrightarrow n + 1 \qquad \qquad (\text{Sum})$$

$$\text{not } b \longrightarrow \neg b \qquad \qquad (\text{Negation})$$

$$\frac{c_1 \longrightarrow c'_1}{\text{mlet } x = c_1 \text{ in } c_2 \longrightarrow \text{mlet } x = c'_1 \text{ in } c_2} \qquad \qquad (\text{Let1})$$

$$\frac{c_1 \longrightarrow c'_1}{c_1 \ c_2 \longrightarrow c'_1 \ c_2} \qquad \qquad (\text{App1})$$

$$\frac{c \longrightarrow c'}{v \ c \longrightarrow v \ c'} \qquad (\text{App2})$$

Figure 2: Reduction rules for Flexible Language.

1.2 Tag Driven Language

Tag Driven Language:

- Non deterministic.
- Type error means stuck.
- Dispatch error means stuck.
- Without type annotation in lambda functions and mlet.
- Semantic "tag driven", introducing flat tag in the environment.

 $\textbf{Definition 2} \ (\mathsf{lookup}). \ \ \mathit{The \ relation \ lookup} \ \mathit{is \ defined \ as \ follows:}$

$$\mathsf{lookup} = \{(x, s, S', v) \mid x \mapsto v \in \mathsf{flat}(s) \land \mathsf{tag}(v) = S\}$$

Definition 3 (flat). The function flat is defined as follows:

$$\mathsf{flat}(s) = \begin{cases} \varnothing & s = \varnothing \\ x \mapsto v_1 \cdots, x \mapsto v_n, \mathsf{flat}(s') & s = x \mapsto \{\overline{v}\}, s' \end{cases}$$

$$S ::=$$
 tags Int integer tag Bool boolean tag Fun function tag ...

Figure 3: Syntax of the Tag Driven Language (Extends Flexible Language).

Figure 4: Reduction rules for Tag Driven Language (Extends Flexible Language).

Definition 4 (tag). The function tag is defined as follows:

$$\mathsf{tag}(v) = \begin{cases} \mathsf{Int} & v = n \\ \mathsf{Bool} & v = b \\ \mathsf{Fun} & v = \lambda x. \ t \end{cases}$$

1.3 Tag Driven Language with ascription

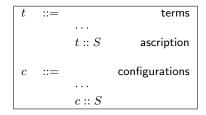


Figure 5: Syntax of the Tag Driven Languagewith ascriptions.

$$c \longrightarrow c$$

$$v :: S \longrightarrow v \qquad \text{(Asc)}$$

$$v = \mathsf{lookup}(x, [s], S)$$

$$x[s] :: S \longrightarrow v :: T$$

$$c \longrightarrow c' \quad \mathsf{notVal_Var}(c)$$

$$c :: S \longrightarrow c' :: S \qquad \text{(Asc1)}$$

Figure 6: Reduction rules for Tag Driven Language with ascriptions.

1.4 Strict Language

$$\begin{array}{ccc} & & \dots & \\ w & ::= & & \mathsf{multi-value} \\ & & \{\overline{v}\} & \mathsf{set of values} \end{array}$$

Figure 7: Syntax of the Strict Language (Extends Tag Driven Language with ascriptions).

1.5 Overloading Language

$$w[s] \longrightarrow w \qquad \qquad \text{(MultiValue)}$$

$$x[x \mapsto w, s] \longrightarrow w \qquad \qquad \text{(VarOk)}$$

$$\frac{x \neq y}{x[y \mapsto w, s] \longrightarrow x[s]} \qquad \text{(VarNext)}$$

$$(mlet \ x = t_1 \ \text{in} \ t_2)[s] \longrightarrow \text{mlet} \ x = t_1[s] \ \text{in} \ t_2[s] \qquad \qquad \text{(LetSub)}$$

$$(t_1 \ t_2)[s] \longrightarrow t_1[s] \ t_2[s] \qquad \qquad \text{(AppSub)}$$

$$mlet \ x = v \ \text{in} \ t_2[s] \longrightarrow t_2[x \mapsto v \oplus s] \qquad \qquad \text{(Let)}$$

$$(\lambda x. \ t_2)[s] \ v \longrightarrow ([x \mapsto v]t_2)[s] \qquad \qquad \text{(App)}$$

$$add1 \ n \longrightarrow n+1 \qquad \qquad \text{(Sum)}$$

$$not \ b \longrightarrow \neg \ b \qquad \qquad \text{(Negation)}$$

$$\frac{c_1 \longrightarrow c_1'}{\text{mlet} \ x = c_1 \ \text{in} \ c_2 \longrightarrow \text{mlet} \ x = c_1' \ \text{in} \ c_2} \qquad \qquad \text{(Let1)}$$

$$\frac{c_1 \longrightarrow c_1'}{c_1 \ c_2 \longrightarrow c_1' \ c_2} \qquad \qquad \text{(App1)}$$

$$\frac{c \longrightarrow c'}{v \ c \longrightarrow v \ c'} \qquad \qquad \text{(App2)}$$

Figure 8: Reduction rules for Strict Language.