Simple Extensions

Principles of Programming Languages

CSE 526

Compiled at 09:00 on 2020/04/23

Typed arithmetic expressions

$$
\begin{array}{rcll}
t & ::= & \texttt{true} & \text{Terms} \\
 & | & \texttt{false} & \\
 & | & \texttt{if}(t, t, t) & \\
 & | & \texttt{0} & \\
 & | & \texttt{succ}\ \ t & \\
 & | & \texttt{pred}\ \ t & \\
 & | & \texttt{iszero}\ \ t &
\end{array}
$$

$$
\begin{array}{rcll}
T & ::= & & \text{Types} \\
 & & \texttt{Bool} & \\
 & | & \texttt{Nat} &
\end{array}
$$

Typing relation for arithmetic expressions

The smallest binary relation ":" between types and terms satisfying all instances of the following inference rules:

$$\texttt{true} : \texttt{Bool} \quad \text{T-True}$$

$$\texttt{false} : \texttt{Bool} \quad \text{T-False}$$

$$\frac{t_1 : \texttt{Bool} \quad t_2 : T \quad t_3 : T}{\texttt{if}(t_1, t_2, t_3) : T} \quad \text{T-If}$$

$$\texttt{0} : \texttt{Nat} \quad \text{T-Zero}$$

$$\frac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}} \quad \text{T-Succ}$$

$$\frac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}} \quad \text{T-Pred}$$

$$\frac{t_1 : \texttt{Nat}}{\texttt{iszero}\ t_1 : \texttt{Bool}} \quad \text{T-IsZero}$$

λ-NB

t	::=		Terms
		...	as in NB
		x	Variable
	|	$\lambda x : T.\ t$	Abstraction
	|	$t\ t$	Application
T	::=		Types
		...	as in NB
	|	$T \to T$	type of functions
Γ	::=		Contexts
		$\emptyset$	Empty Context
	|	$\Gamma, x : T$	Variable Binding

Extended Typing Relation

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{T-Var}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.\ t_2 \ : \ T_1 \to T_2} \quad \text{T-Abs}$$

$$\frac{\Gamma \vdash s : T_1 \to T_2 \qquad \Gamma \vdash t : T_1}{\Gamma \vdash (s\ t) : T_2} \quad \text{T-App}$$

Unit type

- A single constant `unit` that has type `Unit`
- The constant `unit` is analogous to `void`: the return value from "procedures".
- Main application will be when we introduce side-effects later

Syntax: $t ::= \ldots \mid$ `unit` constant `unit`

Values: $v ::= \ldots \mid$ `unit` constant `unit`

Types: $T ::= \ldots \mid$ `Unit` `Unit` type

Unit type: Additional Rules

Typing relation:

$$\Gamma \vdash \texttt{unit} : \texttt{Unit} \qquad \text{T-Unit}$$

Derived Form:

$$t_1 ; t_2 \stackrel{def}{=} (\lambda x.t_2)\ t_1, \quad \text{where } x \notin FV(t_2)$$

- Note that derived forms introduce new syntactic constructs, but their semantics is defined in terms of existing constructs.
- Derived forms are also known as *syntactic sugar*.
- Derived forms are widely used to extend the usable features of a language without changing its internals (e.g. SML and Core-ML).

Let Bindings

- Used to name sub-expressions in a complex expression
- OCAML example:

```
let x=3
in let y = x+2
   in y*y;;
```

- Addition to λNB:

Syntax: $t ::= \ldots \mid \texttt{let}\ x = t\ \texttt{in}\ t$ let binding

No additions to values or types.

Let: Additional Rules

Single-step Evaluation:

$$\texttt{let}\ x = v_1\ \texttt{in}\ t_2 \quad \rightarrow \quad [x \mapsto v_1]t_2 \qquad \text{E-LetVal}$$

$$\frac{t_1 \rightarrow t_1'}{\texttt{let}\ x = t_1\ \texttt{in}\ t_2 \quad \rightarrow \quad \texttt{let}\ x = t_1'\ \texttt{in}\ t_2} \qquad \text{E-Let}$$

Typing Relation:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : T_2} \qquad \text{T-Let}$$

Pairs: Syntax

Syntax:

$$\begin{array}{rcll} t & ::= & \ldots & \\ & | & \{t, t\} & \text{pair} \\ & | & t.1 & \text{first projection} \\ & | & t.2 & \text{second projection} \end{array}$$

Values:

$$\begin{array}{rcll} v & ::= & \ldots & \\ & | & \{v, v\} & \text{pair values} \end{array}$$

Types:

$$\begin{array}{rcll} T & ::= & \ldots & \\ & | & T \times T & \text{product type} \end{array}$$

Pairs: Single-step evaluation rules

Additional Rules:

$$\{v_1, v_2\}.1 \to v_1 \qquad \text{E-ProjVal1}$$

$$\{v_1, v_2\}.2 \to v_2 \qquad \text{E-ProjVal2}$$

$$\frac{t_1 \to t_1'}{t_1.1 \to t_1'.1} \qquad \text{E-Proj1}$$

$$\frac{t_1 \to t_1'}{t_1.2 \to t_1'.2} \qquad \text{E-Proj2}$$

$$\frac{t_1 \to t_1'}{\{t_1, t_2\} \to \{t_1', t_2\}} \qquad \text{E-Pair1}$$

$$\frac{t_2 \to t_2'}{\{v_1, t_2\} \to \{v_1, t_2'\}} \qquad \text{E-Pair2}$$

Pairs: Typing Rules

Additional Rules:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2} \qquad \text{T-Pair}$$

$$\frac{\Gamma \vdash t : T_{11} \times T_{12}}{\Gamma \vdash t.1 : T_{11}} \qquad \text{T-Proj1}$$

$$\frac{\Gamma \vdash t : T_{11} \times T_{12}}{\Gamma \vdash t.2 : T_{12}} \qquad \text{T-Proj2}$$

Records: Syntax

Syntax:

$$\begin{array}{rcll} t & ::= & \ldots & \\ & | & \{l_i = t_i{}^{i \in 1 \ldots n}\} & \text{record} \\ & | & t\,.\,l & \text{projection} \end{array}$$

Values:

$$\begin{array}{rcll} v & ::= & \ldots & \\ & | & \{l_i = v_i{}^{i \in 1 \ldots n}\} & \text{record values} \end{array}$$

Types:

$$\begin{array}{rcll} T & ::= & \ldots & \\ & | & \{l_i : T_i{}^{i \in 1 \ldots n}\} & \text{type of records} \end{array}$$

Records: Single-step evaluation rules

Additional Rules:

$$\{l_i = v_i{}^{i \in 1 \ldots n}\}\,.\,l_j \to v_j \qquad \text{E-ProjRcd}$$

$$\frac{t_1 \to t_1'}{t_1\,.\,l \to t_1'\,.\,l} \qquad \text{E-Proj}$$

$$\frac{t_j \to t_j'}{\begin{array}{c}\{l_i = v_i{}^{i \in 1..(j-1)}, l_j = t_j, l_k = t_k{}^{k \in (j+1) \ldots n}\} \to \\ \{l_i = v_i{}^{i \in 1..(j-1)}, l_j = t_j', l_k = t_k{}^{k \in (j+1) \ldots n}\}\end{array}} \qquad \text{E-Rcd}$$

Records: Typing Rules

Additional Rules:

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i = t_i{}^{i \in 1 \ldots n}\} : \{l_i : T_i{}^{i \in 1 \ldots n}\}} \quad \text{T-Rcd}$$

$$\frac{\Gamma \vdash t : \{l_i : T_i{}^{i \in 1 \ldots n}\}}{\Gamma \vdash t.l_j : T_j} \quad \text{T-Proj}$$

Records: Summary

- Labels in a record are distinct
- Tuples can be considered as special case of records, i.e. with integer labels
- Fields in a record are treated as *ordered*
- Ordered field semantics is common in lower-level languages (e.g. C)

Variants: Syntax

Syntax:

$$\begin{array}{rcll} t & ::= & \ldots & \\ & | & \texttt{<}l\texttt{=}t\texttt{>}\ \texttt{as}\ T & \text{tagging} \\ & | & \texttt{case}\ t\ \texttt{of}\ \texttt{<}l_i\texttt{=}x_i\texttt{>} \Rightarrow t_i^{\,i\in 1\ldots n} & \text{case} \end{array}$$

Values:

$$\begin{array}{rcll} v & ::= & \ldots & \\ & | & \texttt{<}l\texttt{=}v\texttt{>}\ \texttt{as}\ T & \text{tagged values} \end{array}$$

Types:

$$\begin{array}{rcll} T & ::= & \ldots & \\ & | & \texttt{<}l_i : T_i^{\,i\in 1\ldots n}\texttt{>} & \text{variant type} \end{array}$$

Variants: Single step evaluation rules

Additional Rules:

$$\texttt{case}\ (\texttt{<}l_j = v_j\texttt{>}\ \texttt{as}\ T)\ \texttt{of}\ \texttt{<}l_i\texttt{=}x_i\texttt{>} \Rightarrow t_i^{\,i\in 1\ldots n} \to [x_j \mapsto v_j]t_j \qquad \text{E-CaseVariant}$$

$$\frac{t_0 \to t_0'}{\begin{array}{l}\texttt{case}\ t_0\ \texttt{of}\ \texttt{<}l_i\texttt{=}x_i\texttt{>} \Rightarrow t_i^{\,i\in 1\ldots n} \\ \to\ \texttt{case}\ t_0'\ \texttt{of}\ \texttt{<}l_i\texttt{=}x_i\texttt{>} \Rightarrow t_i^{\,i\in 1\ldots n}\end{array}} \qquad \text{E-Case}$$

$$\frac{t_i \to t_i'}{\texttt{<}l_i = t_i\texttt{>}\ \texttt{as}\ T \to \texttt{<}l_i = t_i'\texttt{>}\ \texttt{as}\ T} \qquad \text{E-Variant}$$

Variants: Typing rules

Additional rules:

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash \texttt{<}l_j = t_j\texttt{>}\ \texttt{as}\ \texttt{<}l_i : T_i^{\,i\in 1\ldots n}\texttt{>}\ :\ \texttt{<}l_i : T_i^{\,i\in 1\ldots n}\texttt{>}} \quad \text{T-Variant}$$

$$\frac{\Gamma \vdash t_0 : \texttt{<}l_i : T_i^{\,i\in 1\ldots n}\texttt{>} \qquad \text{for each } i \quad \Gamma, x_i : T_i \vdash t_i : T}{\Gamma \vdash \texttt{case}\ t_0\ \texttt{of}\ \texttt{<}l_i\texttt{=}x_i\texttt{>} \Rightarrow t_i^{\,i\in 1\ldots n}\ :\ T} \quad \text{T-Case}$$

Recursion: Syntax

Syntax:

$$\begin{array}{rcll} t & ::= & \ldots & \\ & | & \texttt{fix}\ t & \text{fixed point} \end{array}$$

Derived Form:

$$\begin{array}{l} \texttt{letrec}\ f : T_1 = t_1\ \texttt{in}\ t_2 \\ \stackrel{def}{=} \texttt{let}\ f = \texttt{fix}\ (\lambda\ f : T_1.\ t_1)\ \texttt{in}\ t_2 \end{array}$$

Recursion: Single-step evaluation and typing rules

Additional Rules:

- Single-step evaluation rules:

$$\frac{t \to t'}{\texttt{fix}\ t \to \texttt{fix}\ t'} \quad \text{E-Fix}$$

$$\texttt{fix}\ (\lambda f : T_1.\ t_1) \ \to [f \mapsto \texttt{fix}\ (\lambda f : T_1.\ t_1)\ ]t_1 \qquad \text{E-FixBeta}$$

- Typing rule:

$$\frac{\Gamma \vdash t_1 : T_1 \to T_1}{\Gamma \vdash \texttt{fix}\ t_1 : T_1} \quad \text{T-Fix}$$