

Simple Extensions

Principles of Programming Languages

CSE 526

- 1 Recall
- 2 Simple Extensions
- 3 Datatypes
- 4 Recursion

Typed arithmetic expressions

```
 $t ::=$  true  
      | false  
      | if( $t, t, t$ )  
      | 0  
      | succ  $t$   
      | pred  $t$   
      | iszero  $t$ 
```

Terms

```
 $T ::=$   
      Bool  
      | Nat
```

Types

Typing relation for arithmetic expressions

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

$$\text{true} : \text{Bool} \quad \text{T-TRUE}$$

$$\text{false} : \text{Bool} \quad \text{T-FALSE}$$

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

$$0 : \text{Nat} \quad \text{T-ZERO}$$

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

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

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

λ -NB

 $t ::=$

...

 x | $\lambda x : T. t$ | $t t$

Terms

as in NB

Variable

Abstraction

Application

λ -NB

$t ::=$		Terms
	...	as in NB
	x	Variable
	$\lambda x : T. t$	Abstraction
	$t t$	Application
$T ::=$		Types
	...	as in NB
	$T \rightarrow T$	type of functions

λ -NB

 $t ::=$

...

 x $|\ \lambda x : T. t$ $|\ t t$

Terms

as in NB

Variable

Abstraction

Application

 $T ::=$

...

 $|\ T \rightarrow T$

Types

as in NB

type of functions

 $\Gamma ::=$ \emptyset $|\ \Gamma, x : T$

Contexts

Empty Context

Variable Binding

Extended Typing Relation

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

Extended Typing Relation

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

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

Extended Typing Relation

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

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

$$\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash (s \ t) : T_2} \quad \text{T-APP}$$

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: $t ::= \dots$
 | `unit` constant unit

Values: $v ::= \dots$
 | `unit` constant unit

Types: $T ::= \dots$
 | `Unit` Unit type

Unit type: Additional Rules

Typing relation:

$$\Gamma \vdash \mathbf{unit} : \mathbf{Unit} \quad \text{T-UNIT}$$

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: $t ::= \dots$
 | `let` $x = t$ `in` t **let binding**

No additions to values or types.

Let: Additional Rules

Single-step Evaluation:

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

$$\frac{t_1 \rightarrow t'_1}{\text{let } x = t_1 \text{ in } t_2 \rightarrow \text{let } x = t'_1 \text{ in } t_2} \quad \text{E-LET}$$

Typing Relation:

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

Pairs: Syntax

Syntax:

$$\begin{array}{l} t ::= \dots \\ \quad | \{t, t\} \quad \text{pair} \\ \quad | t.1 \quad \text{first projection} \\ \quad | t.2 \quad \text{second projection} \end{array}$$

Values:

$$\begin{array}{l} v ::= \dots \\ \quad | \{v, v\} \quad \text{pair values} \end{array}$$

Types:

$$\begin{array}{l} T ::= \dots \\ \quad | T \times T \quad \text{product type} \end{array}$$

Pairs: Single-step evaluation rules

Additional Rules:

$$\{v_1, v_2\}.1 \rightarrow v_1 \quad \text{E-PROJVAL1}$$

$$\{v_1, v_2\}.2 \rightarrow v_2 \quad \text{E-PROJVAL2}$$

$$\frac{t_1 \rightarrow t'_1}{t_1.1 \rightarrow t'_1.1} \quad \text{E-PROJ1}$$

$$\frac{t_1 \rightarrow t'_1}{t_1.2 \rightarrow t'_1.2} \quad \text{E-PROJ2}$$

$$\frac{t_1 \rightarrow t'_1}{\{t_1, t_2\} \rightarrow \{t'_1, t_2\}} \quad \text{E-PAIR1}$$

$$\frac{t_2 \rightarrow t'_2}{\{v_1, t_2\} \rightarrow \{v_1, t'_2\}} \quad \text{E-PAIR2}$$

Pairs: Typing Rules

Additional Rules:

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

$$\frac{\Gamma \vdash t : T_{11} \times T_{12}}{\Gamma \vdash t.1 : T_{11}} \quad \text{T-PROJ1}$$

$$\frac{\Gamma \vdash t : T_{11} \times T_{12}}{\Gamma \vdash t.2 : T_{12}} \quad \text{T-PROJ2}$$

Records: Syntax

Syntax:

$$\begin{array}{l} t ::= \dots \\ \quad | \{l_i = t_i^{i \in 1 \dots n}\} \quad \text{record} \\ \quad | t.l \quad \text{projection} \end{array}$$

Values:

$$\begin{array}{l} v ::= \dots \\ \quad | \{l_i = v_i^{i \in 1 \dots n}\} \quad \text{record values} \end{array}$$

Types:

$$\begin{array}{l} T ::= \dots \\ \quad | \{l_i : T_i^{i \in 1 \dots n}\} \quad \text{type of records} \end{array}$$

Records: Single-step evaluation rules

Additional Rules:

$$\{l_i = v_i^{i \in 1..n}\}.l_j \rightarrow v_j \quad \text{E-PROJRC D}$$

$$\frac{t_1 \rightarrow t'_1}{t_1.l \rightarrow t'_1.l} \quad \text{E-PROJ}$$

$$\frac{t_j \rightarrow t'_j}{\{l_i = v_i^{i \in 1..(j-1)}, l_j = t_j, l_k = t_k^{k \in (j+1)..n}\} \rightarrow \{l_i = v_i^{i \in 1..(j-1)}, l_j = t'_j, l_k = t_k^{k \in (j+1)..n}\}} \quad \text{E-RCD}$$

Records: Typing Rules

Additional Rules:

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

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

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:

$$\begin{aligned} t &::= \dots \\ &| \langle l=t \rangle \text{ as } T && \text{tagging} \\ &| \text{ case } t \text{ of } \langle l_i=x_i \rangle \Rightarrow t_i^{i \in 1 \dots n} && \text{case} \end{aligned}$$

Values:

$$\begin{aligned} v &::= \dots \\ &| \langle l=v \rangle \text{ as } T && \text{tagged values} \end{aligned}$$

Types:

$$\begin{aligned} T &::= \dots \\ &| \langle l_i : T_i^{i \in 1 \dots n} \rangle && \text{variant type} \end{aligned}$$

Variants: Single step evaluation rules

Additional Rules:

$$\text{case } (\langle l_j = v_j \rangle \text{ as } T) \text{ of } \langle l_i = x_i \rangle \Rightarrow t_i^{i \in 1 \dots n} \rightarrow [x_j \mapsto v_j] t_j \quad \text{E-CASEVARIANT}$$

$$\frac{t_0 \rightarrow t'_0}{\text{case } t_0 \text{ of } \langle l_i = x_i \rangle \Rightarrow t_i^{i \in 1 \dots n} \rightarrow \text{case } t'_0 \text{ of } \langle l_i = x_i \rangle \Rightarrow t_i^{i \in 1 \dots n}} \quad \text{E-CASE}$$

$$\frac{t_i \rightarrow t'_i}{\langle l_i = t_i \rangle \text{ as } T \rightarrow \langle l_i = t'_i \rangle \text{ as } T} \quad \text{E-VARIANT}$$

Variants: Typing rules

Additional rules:

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash \langle l_j = t_j \rangle \text{ as } \langle l_i : T_i^{i \in 1 \dots n} \rangle : \langle l_i : T_i^{i \in 1 \dots n} \rangle} \quad \text{T-VARIANT}$$

$$\frac{\begin{array}{c} \Gamma \vdash t_0 : \langle l_i : T_i^{i \in 1 \dots n} \rangle \\ \text{for each } i \quad \Gamma, x_i : T_i \vdash t_i : T \end{array}}{\Gamma \vdash \text{case } t_0 \text{ of } \langle l_i = x_i \rangle \Rightarrow t_i^{i \in 1 \dots n} : T} \quad \text{T-CASE}$$

Recursion: Syntax

Syntax:

$$t ::= \dots$$

| `fix t` fixed point

Derived Form:

$$\text{letrec } f : T_1 = t_1 \text{ in } t_2$$
$$\stackrel{\text{def}}{=} \text{let } f = \text{fix } (\lambda f : T_1. t_1) \text{ in } t_2$$

Recursion: Single-step evaluation and typing rules

Additional Rules:

- Single-step evaluation rules:

$$\frac{t \rightarrow t'}{\mathbf{fix} \ t \rightarrow \mathbf{fix} \ t'} \quad \text{E-FIX}$$

$$\mathbf{fix} \ (\lambda f : T_1. t_1) \rightarrow [f \mapsto \mathbf{fix} \ (\lambda f : T_1. t_1)] t_1 \quad \text{E-FIXBETA}$$

- Typing rule:

$$\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_1}{\Gamma \vdash \mathbf{fix} \ t_1 : T_1} \quad \text{T-FIX}$$