Language #1: Expressions

• Simple arithmetic and boolean operations
• Every term computes to a *value*, either int or bool

• Arithmetic operators: plus, minus, times
• Boolean operators: and, or, not, comparison, if-then-else

• $3 + 5 \times 9$
• if $1 = 0$ or $1 = 1$ then $2$ else $4$
Expressions: Syntax

• Arithmetic operators: plus, minus, times
• Boolean operators: and, or, not, comparison, if-then-else
Expressions: Syntax

\[ E ::= \langle\#\rangle \]

\[ \mid E + E \mid E - E \mid E \ast E \]

\[ \mid \langle\text{bool}\rangle \]

\[ \mid E \text{ and } E \mid E \text{ or } E \]

\[ \mid \text{not } E \]

\[ \mid E = E \]

\[ \mid \text{if } E \text{ then } E \text{ else } E \]

type exp = \text{Int of int}

\[ \mid \text{Add of exp} \ast \text{exp} \mid \ldots \]

\[ \mid \text{Bool of bool} \]

\[ \mid \text{And of exp} \ast \text{exp} \mid \ldots \]

\[ \mid \text{Not of exp} \]

\[ \mid \text{Eq of exp} \ast \text{exp} \]

\[ \mid \text{If of exp} \ast \text{exp} \ast \text{exp} \]
Expressions: Interpreter

• Every term computes to a \textit{value}, either int or bool

\texttt{type value = IntV of int | BoolV of bool}

\texttt{let rec eval (e : exp) : value = \texttt{(* let rec eval e = *)}}

\texttt{match e with}

| \texttt{Int i} ->
| \texttt{Add (e1, e2)} ->
| ...
Expressions: Interpreter

• Every term computes to a value, either int or bool

```ocaml
type value = IntV of int | BoolV of bool

let rec eval (e : exp) : value = (* let rec eval e = *)
match e with
| Int i -> IntV i
| Add (e1, e2) ->
| ...
```

Expressions: Interpreter

• Every term computes to a value, either int or bool

`type value = IntV of int | BoolV of bool`

```ocaml
let rec eval (e : exp) : value = (* let rec eval e = *)
    match e with
    | Int i -> IntV i
    | Add (e1, e2) -> eval e1 + eval e2
    | ...  Type error!
```
let rec eval (e : exp) : value =
  match e with
  | Int i -> IntV i
  | Add (e1, e2) ->
    (match eval e1, eval e2 with
     | IntV i1, IntV i2 -> IntV (i1 + i2)
     | _, _ -> ?)
Expressions: Interpreter

let rec eval (e : exp) : value =
  match e with
  | Int i -> IntV i
  | Add (e1, e2) ->
    (match eval e1, eval e2 with
     | IntV i1, IntV i2 -> IntV (i1 + i2)
     | _, _ -> None)

  type 'a option = Some of 'a | None
Expressions: Interpreter

let rec eval (e : exp) : value option =
  match e with
  | Int i -> IntV i
  | Add (e1, e2) ->
    (match eval e1, eval e2 with
    | IntV i1, IntV i2 -> IntV (i1 + i2)
    | _, _ -> None)

type 'a option = Some of 'a | None
let rec eval (e : exp) : value option =
  match e with
  | Int i -> Some (IntV i)
  | Add (e1, e2) ->
    (match eval e1, eval e2 with
     | Some (IntV i1), Some (IntV i2) -> Some (IntV (i1 + i2))
     | _, _ -> None)

type 'a option = Some of 'a | None
Expressions: Interpreter

let rec eval (e : exp) : value option =
match e with
| ... |
| Bool b -> Some (BoolV b) |
| And (e1, e2) ->
  (match eval e1, eval e2 with
   | Some (BoolV b1), Some (BoolV b2) -> Some (BoolV (b1 && b2)) |
   | _, _ -> None)
Expressions: Interpreter

let rec eval (e : exp) : value option =
  match e with
  | ... | ...
  | Eq (e1, e2) ->
    (match eval e1, eval e2 with
     | Some (IntV i1), Some (IntV i2) -> Some (BoolV (i1 = i2))
     | ...)

Expressions: Interpreter

let rec eval (e : exp) : value option =
  match e with
  | ...|
  | Eq (e1, e2) ->
    (match eval e1, eval e2 with
     | Some v1, Some v2 -> Some (BoolV (v1 = v2))
     | ...)

Expressions: Interpreter

let rec eval (e : exp) : value option =
  match e with
  | ...
  | Eq (e1, e2) -> Some (BoolV (eval e1 = eval e2))

• Should we be able to compare ints and bools? Should two erroneous expressions be equal? Depends on the language!
Expressions: Interpreter

let rec eval (e : exp) : value option =
  match e with
  | ...  
  | If (e, e1, e2) ->
    (match eval e with
     | Some (BoolV b) -> if b then eval e1 else eval e2 
     | _ -> None)
Expressions: Int-only Interpreter

let rec eval (e : exp) : int =
  match e with
  | Int i -> i
  | Add (e1, e2) -> eval e1 + eval e2
  | ...  
  | If (e, e1, e2) -> if eval e <> 0 then eval e1 else eval e2
  
• Simpler interpreter, but behavior may surprise programmers!
Structure of a language

• Syntax
  — Concrete: what do programs look like?
  — Abstract: what are the pieces of a program?

➢ Semantics
  — Static: which programs make sense?
  — Dynamic: what do programs do when we run them?

• Pragmatics
  — Implementation: how can we actually make the semantics happen?
  — IDE, tool support, etc.
Static Semantics: Types

• A type system is a relation between terms and types
• We write $t : \tau$ for “term $t$ has type $\tau$”
• We define type systems using inference rules:

\[
\frac{P_1 \ldots P_n}{t : \tau} \quad \frac{e_1 : \text{int} \quad e_2 : \text{int}}{e_1 + e_2 : \text{int}}
\]

• Often sound: $t : \tau$ implies that $t$ evaluates to a value of type $\tau$
• Often conservative: not all programs that evaluate are well-typed
Expressions: Types

- Types: int, bool
- Rules:
  - \((n \text{ is a number})\)
    \[
    \frac{n : \text{int}}{n : \text{int}}
    \]
  - \((b \text{ is a boolean})\)
    \[
    \frac{b : \text{bool}}{b : \text{bool}}
    \]
  - \(e_1 : \tau\) \(e_2 : \tau\)
    \[
    \frac{e_1 = e_2 : \text{bool}}{e_1 = e_2 : \text{bool}}
    \]
Expressions: Types

• Types: int, bool

• Rules:

  \[(n \text{ is a number})\]
  \[
  \frac{n : \text{int}}{}
  \]

  \[(b \text{ is a boolean})\]
  \[
  \frac{b : \text{bool}}{}
  \]

  \[e_1 : \tau_1 \quad e_2 : \tau_2\]
  \[
  \frac{e_1 = e_2 : \text{bool}}{}
  \]

  \[e_1 : \text{int} \quad e_2 : \text{int}\]
  \[
  \frac{e_1 + e_2 : \text{int}}{}
  \]

  \[e_1 : \text{bool} \quad e_2 : \text{bool}\]
  \[
  \frac{e_1 \text{ and } e_2 : \text{bool}}{}
  \]
Expressions: Types

• Types: int, bool

• Rules:
  
  (n is a number)  
  \[n : \text{int}\]

  (b is a boolean)  
  \[b : \text{bool}\]

  \[e_1 : \tau \quad e_2 : \tau\]
  \[e_1 = e_2 : \text{bool}\]

  \[e_1 : \text{int} \quad e_2 : \text{int}\]
  \[e_1 + e_2 : \text{int}\]

  \[e_1 : \text{bool} \quad e_2 : \text{bool}\]
  \[e_1 \text{ and } e_2 : \text{bool}\]

  \[e : \text{bool} \quad e_1 : \tau \quad e_2 : \tau\]
  \[\text{if } e \text{ then } e_1 \text{ else } e_2 : \tau\]