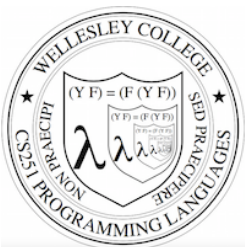


Interpreting and Compiling Intex



CS251 Programming Languages
 Fall 2017, Lyn Turbak

Department of Computer Science
 Wellesley College

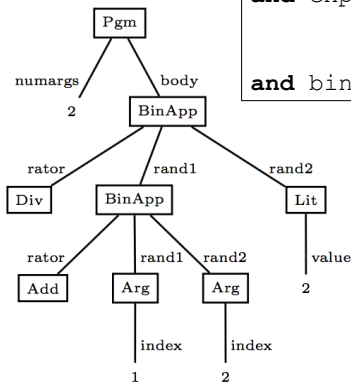
A New Mini-Language: Intex

Intex programs are simple arithmetic expressions on integers that can refer to integer arguments.

Intex is the first in a sequence of mini-languages that can be extended to culminate in something that is similar to Racket. At each step along the way, we can add features that allow us to study different programming language dimensions.

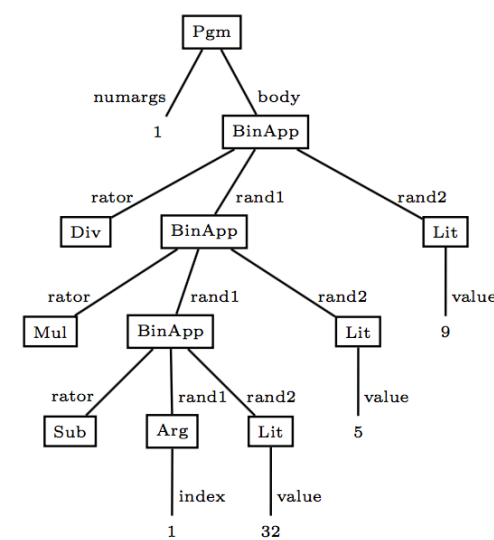
Intex Syntax Trees & Syntactic Data Types

```
datatype pgm = Intex of int * exp
and exp = Int of int
        | Arg of int
        | BinApp of binop * exp * exp
and binop = Add | Sub | Mul | Div | Rem
```



```
val avg = Intex(2, BinApp(Div, BinApp(Add, Arg 1, Arg 2), Int 2))
```

How do we write this Intex program in SML?



Intex Implementation #1: Intex Interpreter in SML

Given an [avg-in-Intex program](#), how can we execute it?

- avg machine (I)
- [avg-in-Intex program](#)
- Intex interpreter machine (I)
 - ✧ [Intex-in-SML-interpreter program](#)
 - ✧ SML interpreter machine in wx VM (ignore details)

Intex 5

Intex Interpreter Without Error Checking: Skeleton

```
(* Intex.pgm -> int list -> int *)
fun run (Intex(numargs, exp)) args =
  ??

(* Intex.exp -> int list -> int *)
and eval (Int i) args = ??
  | eval (Arg index) args = ??
  | eval (BinApp(binop, exp1, exp2)) args =
    ??

(* Intex.binop -> int * int -> int *)
and binopToFun Add = op+
  | binopToFun Mul = op*
  | binopToFun Sub = op-
  | binopToFun Div = (fn(x,y) => x div y)
  | binopToFun Rem = (fn(x,y) => x mod y)
```

Intex 6

Intex Interpreter Without Error Checking: Solutions

```
(* Intex.pgm -> int list -> int *)
fun run (Intex(numargs, exp)) args =
  eval exp args

(* Intex.exp -> int list -> int *)
and eval (Int i) args = i
  | eval (Arg index) args = List.nth(args, index-1)
  | eval (BinApp(binop, exp1, exp2)) args =
    (binopToFun binop)(eval exp1 args, eval exp2 args)

(* Intex.binop -> int * int -> int *)
and binopToFun Add = op+
  | binopToFun Mul = op*
  | binopToFun Sub = op-
  | binopToFun Div = (fn(x,y) => x div y)
  | binopToFun Rem = (fn(x,y) => x mod y)
```

Intex 7

Intex Interpreter With Error Checking

```
exception EvalError of string

(* Intex.pgm -> int list -> int *)
fun run (Intex(numargs, exp)) args =
  if numargs <> length args
  then raise EvalError
    "Mismatch between expected and actual number of args"
  else eval exp args

(* Intex.exp -> int list -> int *)
and eval (Int i) args = i
  | eval (Arg index) args =
    if (index <= 0) orelse (index > length args)
    then raise EvalError "Arg index out of bounds"
    else List.nth(args, index-1)
  | eval (BinApp(binop, exp1, exp2)) args =
    let val i1 = eval exp1 args
        val i2 = eval exp2 args
    in (case (binop, i2) of
        (Div, 0) => raise EvalError "Division by 0"
        | (Rem, 0) => raise EvalError "Remainder by 0"
        | _ => (binopToFun binop)(i1, i2))
    end
```

Intex 8

Try it out

```
- run (Intex(1, BinApp(Mul, Arg 1, Arg 1))) [5];
val it = 25 : int

- run (Intex(1, BinApp(Div, Arg 1, Arg 1))) [5];
val it = 1 : int

- run (Intex(1, BinApp(Div, Arg 1, Arg 1))) [0];
uncaught exception EvalError

- run avg [5,15];
val it = 10 : int

- map (run f2c) [[~40], [0], [32], [98], [212]];
val it = [~40,~18,0,36,100] : int list
```

Intex 9

Handling Errors

```
(* Intex.pgm -> int list -> string *)
fun testRun pgm args =
  Int.toString (run pgm args) (* Convert to string so
  same type as error messages below *)
  handle EvalError msg => "EvalError: " ^ msg
  | other => "Unknown exception: " ^ (exnMessage other)
```

```
- testRun (Intex(1, BinApp(Div, Arg 1, Arg 1))) [5];
val it = "1" : string

- testRun (Intex(1, BinApp(Div, Arg 1, Arg 1))) [0];
val it = "EvalError: Division by 0" : string

- map (testRun f2c) [[~40], [0], [32], [98], [212]];
val it = ["~40","~18","0","36","100"] : string list
```

Intex 10

Intex programs as S-expression strings

```
Intex(1, BinApp(Mul, Arg 1, Arg 1))
```

```
"(intex 1 (* ($ 1) ($ 1))"
```

```
Intex(2,
  BinApp(Div,
    BinApp(Add, Arg 1, Arg 2),
    Int 2))
```

```
"(intex 2 (/ (+ ($ 1) ($ 2)) 2))"
```

```
Intex(1,
  BinApp(Div,
    BinApp(Mul,
      BinApp(Sub, Arg 1, Int 32),
      Int 5),
    Int 9))
```

```
"(intex 1 (/ (* (- ($ 1) 32) 5) 9))"
```

Intex 11

Running Intex programs as S-expression strings

```
(* string -> string -> string *)
fun testRun' pgmSexpString argsSexpString =
  testRun (stringToPgm pgmSexpString)
  (sexpStringToIntList argsSexpString)
  handle SexpError (msg, sexp) =>
    ("SexpError: " ^ msg ^ " " ^ (Sexp.sexpToString sexp))
  | Sexp.IllFormedSexp msg =>
    ("SexpError: Ill-formed sexp " ^ msg)
  | other => "Unknown exception: " ^ (exnMessage other)
```

```
- testRun' "(intex 2 (/ (+ ($ 1) ($ 2)) 2))" "(5 15)";
val it = "10" : string

- map (testRun' "(intex 1 (/ (* (- ($ 1) 32) 5) 9))")
= ["(-40)", "(0)", "(32)", "(98)", "(212)"];
val it = ["~40","~18","0","36","100"] : string list

- map (testRun' "(intex 1 (/ ($ 1) ($ 1)))"=
= ["(-17)", "(0)", "(42)"];
val it = ["1","EvalError: Division by 0","1"] : string list
```

Intex 12

A Read-Eval-Print Loop (REPL) in Intex

```

- repl();

intex> (+ 1 2)
3

intex> (#args 6 7)

intex> ( * ($ 1) ($ 2) )
42

intex> (#run (intex 2 (/ (+ ($ 1) ($ 2)) 2)) 5 15)
10

intex> (#run "avg.itx" 5 15)
10

intex> (#run avg.itx 5 15)
10

intex> (#quit)
Moriturus te saluto!

```

Intex 13

What do we know about this program?

```

val test = Intex(2,
                  BinApp(Sub,
                          BinApp(Mul, Arg 1, Arg 3),
                          Arg 2))

```

Intex 14

Dynamic vs. Static Checking: Arg Indices

Dynamic check (at runtime) :

```

| eval (Arg index) args =
  if (index <= 0) orelse (index > length args)
  then raise EvalError "Arg index out of bounds"
  else List.nth(args, index-1)

```

Static check (at compile time or checking time, before runtime) :

Idea: We know numargs from program, so can use this to check all argument references without running the program.

Such checks are done by examining the program syntax tree. Often there is a choice between a *bottom-up* and *top-down* approach to processing the tree.

You will do both approaches for Arg index checking in PS9.

Intex 15

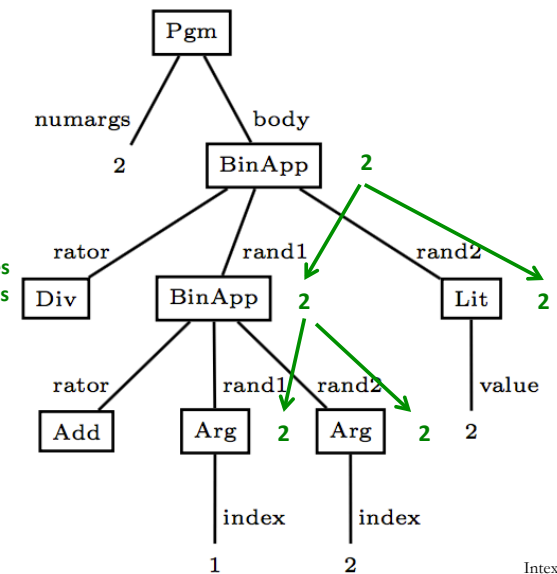
Static Arg Index Checking: Top Down

In top-down phase, pass numargs to every subexpression in program.

Check against every arg Index.

Return true for Arg indices that pass test and subexps Without arg indices

Return false if any Arg index fails test.



Intex 16

How to define intexToPostFix? Skeleton

```
fun intexToPostFix (Intex.Intex(numargs, exp)) =
  PostFix.PostFix(numargs, expToCmds exp ???)
  (* 0 is a depth argument that statically tracks
     how many values are on stack above the arguments *)

and expToCmds (Intex.Int i) depth = ???
  | expToCmds (Intex.Arg index) depth = ???

  | expToCmds (Intex.BinApp(binop,exp1,exp2)) depth = ???

and binopToArithop Intex.Add = PostFix.Add
  | binopToArithop Intex.Sub = PostFix.Sub
  | binopToArithop Intex.Mul = PostFix.Mul
  | binopToArithop Intex.Div = PostFix.Div
  | binopToArithop Intex.Rem = PostFix.Rem
```

Intex 21

How to define intexToPostFix? Solution

```
fun intexToPostFix (Intex.Intex(numargs, exp)) =
  PostFix.PostFix(numargs, expToCmds exp 0)
  (* 0 is a depth argument that statically tracks
     how many values are on stack above the arguments *)

and expToCmds (Intex.Int i) depth = [PostFix.Int i]
  | expToCmds (Intex.Arg index) depth =
    [PostFix.Int (index + depth), PostFix.Nget]
    (* specified argument is on stack at index + depth *)
  | expToCmds (Intex.BinApp(binop,exp1,exp2)) depth =
    (expToCmds exp1 depth)
    (* 1st rand is at same depth as whole binapp *)
    @ (expToCmds exp2 (depth + 1))
    (* for 2nd rand, add 1 to depth to account for 1st rand *)
    @ [PostFix.Arithop (binopToArithop binop)]

and binopToArithop Intex.Add = PostFix.Add
  | binopToArithop Intex.Sub = PostFix.Sub
  | binopToArithop Intex.Mul = PostFix.Mul
  | binopToArithop Intex.Div = PostFix.Div
  | binopToArithop Intex.Rem = PostFix.Rem
```

Intex 22