CSCI: 4500/6500 Programming Languages

SML

Standard ML

- General purpose, modular functional programming language developed a team in the 1970s at the University of Edinburgh, headed by Robin Milner (polymorphism paper in reading list).
- First language to include polymorphic type inference together with a type-safe exception handling mechanism
- ML - historically stands for Meta Language. ML was a meta language for expressing and manipulating logical proofs.

Standard ML

- Static-scoped with a syntax that is closer to Pascal than to LISP
  - » e.g. infix arithmetic expressions
- Uses type declarations, but also does type inferencing to determine the types of undeclared variables
  - » type of all variables can be determined at compile time.
- It is strongly typed (whereas Scheme is essentially typeless) and has no type coercions (talk more about this later in the Semester)
- Includes exception handling
- Module facility for implementing abstract data types.
- Permits side-effect (therefore an impure functional language)

Standard ML (cont)

- Standard ML is a domain-specific language that is appropriate for building compilers
- Support for
  - » Complex data structures (abstract syntax, compiler intermediate forms)
  - » Memory management like Java
  - » Large projects with many modules
  - » Advanced type system for error detection

Some sad news…

- You will be responsible for learning SML on your own.
- Today we will cover some basics
- Resources:
  - » Robert Harper's
    - – http://www.cs.cmu.edu/People/rwh/introsml/overview.htm
  - » Peter Lee's:
    - – http://www.cs.cmu.edu/~petel/smlguide/smlnj.htm
  - » SML/NJ Literature:
    - – http://www.smlnj.org/doc/literature.html#tutorials
    - – Runs on Microsoft Windows, MacOS X (yay!), UNIX,
  - » Short and comprehensive tutorial:
    - – http://cs.wwc.edu/Environment/SML-Tutorial.html
  - » more… (and more will be provided).

Installation

Distribution (SML of New Jersey):

- http://www.smlnj.org/dist/working/110.57/index.html
- Developed at Bell laboratories and Princeton University
- Installation: Set your `PATH` variable where you install it.
- Run:

```
{saffron:ingrid:219} sml
Standard ML of New Jersey v110.57 [built: Mon
  Nov 21 09:35:29 2005]
-
```

Interactive ML

- Type in expressions
- Evaluate and print type and result
- Compiles as well

Hello word! in SML

```
- print("Hello world!\n");
Hello world
val it = () : unit
-
```


Preliminaries

- Read – Eval – Print – Loop:

```
- 3+2;
```


Preliminaries

- Read – Eval – Print – Loop:

```
- 3+2;
val it = 5 : int
```


Preliminaries

- Read – Eval – Print – Loop:

```
- 3+2;
val it = 5 : int
- it + 7 ;
```


Preliminaries

- Read – Eval – Print – Loop:

```
- 3+2;
val it = 5 : int
- it + 7 ;
val it = 12 : int
```

Preliminaries

- Read – Eval – Print – Loop:

```
- 3+2;
val it = 5 : int
- it + 7 ;
val it = 12 : int
- it - 3 ;
```


Preliminaries

- Read – Eval – Print – Loop:

```
- 3+2;
val it = 5 : int
- it + 7 ;
val it = 12 : int
- it - 3 ;
val it = 9 : int
```


Preliminaries

- Read – Eval – Print – Loop:

```
- 3+2;
val it = 5 : int
- it + 7 ;
val it = 12 : int
- it - 3 ;
val it = 9 : int
- 4 + true
```


Preliminaries

- Read – Eval – Print – Loop:

```
- 3+2;
val it = 5 : int
- it + 7 ;
val it = 12 : int
- it - 3 ;
val it = 9 : int
- 4 + true
= ;
stdIn:14.1-14.9 Error: operator and operand don't
  agree [literal]
  operator domain: int * int
  operand:         int * bool
  in expression:
    4 + true
```


- Copy and paste the following text into a Standard ML window:

```
2+2;          (* note semicolon at end*)
3*4;
4/3;          (* an error! *)
6 div 2;      (* integer division *)
7 div 3;
```



```
- 4/3 ;
stdIn:20.1-20.4 Error: operator and operand
  don't agree [literal]
  operator domain: real * real
  operand:         int * int
  in expression:
    4 / 3
- 4.0 / 3.0 ;
val it = 1.33333333333 : real
```

- Includes lists and list operations
- The `val` statement binds a name to a value (similar to `DEFINE` in Scheme)
- Function declaration form:

  `fun` function_name (formal_parameters) =

  function_body_expression;

e.g.,

```
fun cube(x : int) = x * x * x ;
fun square( x: int) : int = x * x ;
```