A Brief Introduction to Standard ML

Specification and Verification with Higher-Order Logic

Arnd Poetzsch-Heffter
(Slides by Jens Brandt)

Software Technology Group
Fachbereich Informatik
Technische Universität Kaiserslautern

Sommersemester 2008

Outline

Functional Programming

Fact

- *functional program = expression*

Functional Programs

- do not have
  - variables
  - assigments
  - sequences
  - repetition
- instead
  - let-expressions
  - recursive functions
  - higher-order functions

Functional Programming

Advantages

- clearer semantics
- corresponds more directly to abstract mathematical objects
- more freedom in implementation

The SML Programming Language

Overview

- functional programming language
- normally used interactively
- strongly typed, with:
  - type inference
  - abstract data types
  - polymorphic types
- exception-handling mechanisms

Motivation

- SML is the meta-language for the HOL theorem prover

Outline

Evaluation and Bindings

Example (Evaluation)

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

- rev [1,2,3,4,5];
val it = [5,4,3,2,1] : int list
```

Example (Simple Bindings)

```
- val n = 8 * 2 + 5;
val n = 21 : int

- n * 2;
val it = 42 : int
```

Bindings

Example (Special Identifier **it**)

```
- it;
val it = 42 : int
```

Example (Multiple Bindings)

```
- val one = 1 and two = 2;
val one = 1 : int
val two = 2 : int
```

Local Bindings

Example (Simple Local Binding)

```
- val n = 0;
val n = 0 : int

- let val n = 12 in n div 6 end;
val it = 2 : int

- n;
val it = 0 : int
```

Example (Multiple Local Bindings)

```
- let val n = 5 val m = 6 in n + m end;
val it = 11 : int
```

Booleans

Example (Operations)

```
- val b1 = true and b2 = false;
val b1 = true : bool
val b2 = false : bool

- 1 = (1 + 1);
val it = false : bool

- not (b1 orelse b2);
val it = false : bool

- (7 < 3) andalso (false orelse 2 > 0);
val it = false : bool
```

Integers

Example (Operations)

```
- val n = 2 + (3 * 4);
val n = 14 : int

- val n = (10 div 2) - 7;
val n = ~2 : int
```

Applying Functions

General Rules

- type of functions from σ_1 to σ_2 is $\sigma_1 \rightarrow \sigma_2$
- application $f\ \ x$ applies function f to argument x
- call-by-value (obvious!)
- left associative: $m\ \ n\ \ o\ \ p = (((m\ \ n)o)p)$

Defining Functions

Example (One Argument)

```
- fun f n = n + 2;
val f = fn : int -> int

- f 22;
val it = 24 : int
```

Example (Two or More Arguments)

```
- fun plus n (m:int) = n + m;
val plus = fn : int -> int -> int

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

Currying

Example (Curried Addition)

```
- fun plus n (m:int) = n + m;
val plus = fn : int -> int -> int

- plus 1 2;
val it = 3 : int
```

Example (Partial Application)

```
- val inc = plus 1;
val inc = fn : int -> int

- inc 7;
val it = 8 : int
```

Higher-Order Functions

Example (Higher-Order Functions)

```
- fun foo f n = (f(n+1)) div 2 ;
val foo = fn : (int -> int) -> int -> int

- foo inc 3;
val it = 2 : int
```

Recursive Functions

Example (Defining Recursive Functions)

```
- fun f n = if (n=0) then 1 else n * f(n-1);
val f = fn : int -> int

- f 3;
val it = 6 : int

- fun member x [] = false |
      member x (h::t) = (x=h) orelse (member x t);
val member = fn : ''a -> ''a list -> bool

- member 3 [1,2,3,4];
val it = true : bool
```

Lambda Abstractions

Example (The Increment Function)

```
- fn x=> x + 1;
val it = fn : int -> int

- (fn x=> x + 1) 2;
val it = 3 : int
```

Lambda Abstractions

Example (Curried Multiplication)

```
- fn x=> fn y=> x * (y:int );
val it = fn : int -> int -> int

- val double = (fn x=> fn y=> x * (y:int )) 2;
val double = fn : int -> int

- double 22;
val it = 44 : int
```

Clausal Definitions

Example (Fibonacci)

```
fun fib 0 = 1
  | fib 1 = 1
  | fib n = fib (n−1) + fib(n−2);
```

Exceptions

Example (Failure)

```
– hd [];
uncaught exception Hd

– 1 div 0;
uncaught exception Div
```

User-Defined Exceptions

Example (Explicitly Generating Failure)

```
- exception negative_argument_to_fact;
exception negative_argument_to_fact

- fun fact n = if (n<0) then raise negative_argument_to_fact
                        else if (n=0) then 1 else n * fact (n-1);
val fact = fn : int -> int

- fact (~1);
uncaught exception negative_argument_to_fact
```

Example (Exception Handling)

```
- (fact (~1)) handle negative_argument_to_fact => 0;
val it = 0 : int
```

Unit

Example (Unit)

```
- ();
val it = () : unit

- close_theory;
val it = fn : unit -> unit
```

Character Strings

Example (String Operations)

```
- "abc";
val it = "abc" : string

- chr;
val it = fn : int -> string

- chr 97;
val it = "a" : string
```

List Constructors

Example (Empty Lists)

```
- null l;
val it = false : bool

- null [];
val it = true : bool
```

Example (Construction and Concatenation)

```
- 9 :: l;
val it = [9,2,3,5] : int list

- [true,false] @ [false,true];
val it = [true,false,false,true] : bool list
```

List Operations

Example (Head and Tail)

```
- val l = [2,3,2+3];
val l = [2,3,5] : int list

- hd l;
val it = 2 : int

- tl l;
val it = [3,5] : int list
```

Pattern Matching

Example (Pattern Matching and Lists)

```
- fun bigand [] = true
    | bigand (h::t) = h andalso bigand t;
val bigand = fn : bool list -> bool
```

Pairs

Example (Pair Functions)

```
- val p = (2,3);
val p = (2,3) : int * int

- fst p;
val it = 2 : int

- snd p;
val it = 3 : int
```

Records

Example (Date Record)

```
val date = {day=4,month="february",year=1967}
  : {day:int, month:string, year:int}

- val {day=d,month=m,year=y} = date;
val d = 4 : int
val m = "february" : string
val y = 1967 : int

- #month date;
val it = "february" : string
```

Polymorphism

Example (Head Function)

```
- hd [2,3];
val it = 2 : int

- hd [true, false];
val it = true : bool
```

Problem

What is the type of hd?

int list -> int or bool list -> bool

Polymorphism

Example (Type of Head Function)

```
- hd;
val it = fn : 'a list -> 'a
```

Example (Polymorphic Head Function)

- head function has both types
- 'a is a type variable.
- hd can have any type of the form σ list -> σ
  (where σ is an SML type)

Type Inference

Example (Mapping Function)

```
- fun map f l =
        if (null l)
          then []
          else f(hd l)::(map f (tl l));
val map = fn : ('a -> 'b) -> 'a list -> 'b list

- map (fn x=>0);
val it = fn : 'a list -> int list
```

Fact (ML Type Inference)

SML infers the most general type.

Standard List Operations

Example (Mapping)

```
- fun map f [] = []
    | map f (h::t) = f h :: map f t;
val ('a, 'b) map = fn : ('a -> 'b) -> 'a list -> 'b list
```

Example (Filtering)

```
- fun filter P [] = []
    | filter P (h::t) = if P h then h:: filter P t
                        else filter P t;
val 'a filter = fn : ('a -> bool) -> 'a list -> 'a list
```

Type Inference

Example (Function Composition)

```
- fun comp f g x = f(g x);
val comp = fn:('a -> 'b) -> ('c -> 'a) -> 'c -> 'b

- comp null (map (fn y=> y+1));
val it = fn : int list -> bool
```

Some System Functions

Example (Load a file called file.sml)

```
- use;
val it = fn : string -> unit

- use "file.sml";
[opening file.sml]
...
```

Key Commands

- terminate the session: <Ctrl> D
- interrupt: <Ctrl> C

Summary

- functional programming in Standard ML
- evaluation and bindings
- defining Functions
- standard data types
- type inference