EBook Gratis

APRENDIZAJE sml

Free unaffiliated eBook created from
Stack Overflow contributors.

#sml

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sml

It is an unofficial and free sml ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sml.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con sml

Observaciones

Esta sección proporciona una descripción general de qué es sml y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de sml, y vincular a los temas relacionados. Dado que la Documentación para sml es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Examples

Instalación

Hay una docena de implementaciones de Standard ML. MLton produce código muy optimizado, pero no tiene REPL . SML / NJ es el más utilizado, pero tiene mensajes de error ligeramente difíciles para fines de aprendizaje. Moscow ML y Poly / ML son fáciles de comenzar, pero no son compatibles con el formato de paquete .mlb. Sin embargo, eso no es esencial para empezar.

Aquí hay instrucciones para instalar cada uno de SML / NJ, Moscow ML y Poly / ML divididos por sistema operativo.

En Windows

SML / NJ:

- Vaya a http://www.smlnj.org/dist/working/ y encuentre la última versión, por ejemplo, Archivos de distribución 110.80 .
- Desplácese hacia abajo y encuentre el instalador de MS Windows, por ejemplo, smlnj-110.80.msi . Ejecuta el instalador.
- Ahora tiene un REPL en, por ejemplo, `C:\Program Files (x86)\SML NJ\bin\sml.bat` .

Moscú ML:

- Vaya a http://mosml.org/ y haga clic en "Descargar Win. Installer". Ejecuta el instalador.
- Ahora tiene un REPL en, por ejemplo, `C:\Program Files (x86)\mosml\bin\mosml.exe` .

Usando Homebrew en MacOS

SML / NJ:

- Ejecute `brew install smlnj` como su propio usuario. Prueba REPL con `smlnj` .

Moscú ML:

- Vaya a http://mosml.org/ y haga clic en "Descargar archivo PKG". Ejecuta el instalador.
- *Falta ... Prueba REPL ¿cómo? ¿Está en `$PATH` ahora?*

En Ubuntu / Debian Linux

SML / NJ:

- Ejecute `sudo apt-get install smlnj` como superusuario. Prueba REPL con `smlnj` .

Moscú ML:

- *(Ubuntu)* Agrega el PPA como el superusuario. Prueba REPL con `mosml` .

```
sudo add-apt-repository ppa:kflarsen/mosml
sudo apt-get update
sudo apt-get install mosml
```

Añadiendo soporte de readline

Para poder utilizar las teclas de flecha para navegar por las líneas que se escribieron previamente en el REPL, la mayoría de los compiladores SML pueden beneficiarse del programa `rlwrap` . Usando Homebrew en MacOS, instálalo con `brew install rlwrap` , y en Ubuntu / Debian Linux, instálalo con `sudo apt-get install rlwrap` . Luego en la terminal, intente lo siguiente:

```
alias mosml='rlwrap mosml -P full'
alias sml='rlwrap sml'
alias poly='rlwrap poly'
```

Estos alias se pueden agregar, por ejemplo, a su `~/.bashrc` para que funcionen de forma predeterminada.

La tecla de flechas ahora debería funcionar mejor.

Lea Empezando con sml en línea: https://riptutorial.com/es/sml/topic/6953/empezando-con-sml

Capítulo 2: Comentarios

Sintaxis

- (* abre un comentario de bloque
- *) cierra un comentario de bloque
- (* y *) debe ser equilibrado en número

Examples

Todos los comentarios son comentarios en bloque.

```
(***********************************************
*  All comments in SML are block comments
*  Block Comments begin with '(*'
*  Block Comments end with '*)'
*  (* Block Comments can be nested *)
*  The additional framing asterisks at the beginning
*  and end of this block comment are common to languages
*  of SML's vintage.
*  Likewise the asterisk at the start of each line
*  But this is solely a matter of style.
***********************************************)

val _ = print "Block comment example\n" (* line ending block comment *)
```

Comentarios anidados

```
(* The block comment syntax allows nested comments
(* whether or not this is a good thing is probably
a matter of personal opinion (*or coding standards*)*)*)

val _ = print "Nested comment example\n" (* line ending block comment *)
```

Lea Comentarios en línea: https://riptutorial.com/es/sml/topic/6976/comentarios

Capítulo 3: Programación interactiva utilizando el REPL

Sintaxis

- A diferencia de los archivos de código fuente, el punto y coma ';' Es obligatorio terminar cada expresión en el REPL.

Examples

Iniciando el SMLNJ REPL

REPL significa 'Leer Evaluar Imprimir Bucle'. El REPL se puede usar para escribir y ejecutar código una línea a la vez y es una alternativa a escribir código en un archivo y luego compilar o interpretar todo el archivo antes de la ejecución.

Para iniciar SMLNJ REPL desde un símbolo del sistema:

```
smluser> sml
Standard ML of New Jersey v110.78 [built: Thu Jul 23 11:21:58 2015]
- 3+4;
val it = 7 : int
- (*a comment: press contrl-d to exit *)
smluser>
```

En las carcasas de comandos Bash y similares, la funcionalidad de línea de lectura GNU se puede agregar a la REPL SML usando el comando del sistema `rlwrap sml` .

```
smluser> rlwrap sml
Standard ML of New Jersey v110.78 [built: Thu Jul 23 11:21:58 2015]
- 3+4;
val it = 7 : int
- (* pressing the up arrow recalls the previous input *)
- 3+4;
val it = 7 : int
-
smluser>
```

Usando 'it'

Todas las expresiones SML devuelven un valor. El REPL almacena el valor de retorno de la última expresión evaluada. `it` proporciona el valor de la última expresión evaluada en el REPL.

```
smluser> sml
Standard ML of New Jersey v110.78 [built: Thu Jul 23 11:21:58 2015]
- 3+4;
val it = 7 : int
```

```
- it;
val it = 7 : int
- it + 1;
val it = 8 : int
-

[1]+  Stopped                 sml
smluser>
```

Efectivamente, los comentarios no son evaluados por el REPL y no cambian su valor.

```
smluser> sml
Standard ML of New Jersey v110.78 [built: Thu Jul 23 11:21:58 2015]
- 3+4;
val it = 7 : int
- (* a comment *);
- it;
val it = 7 : int

[1]+  Stopped                 sml
smluser>
```

Lea Programación interactiva utilizando el REPL en línea:
https://riptutorial.com/es/sml/topic/6975/programacion-interactiva-utilizando-el-repl

Capítulo 4: Sistema de módulos

Examples

Evaluación perezosa

Standard ML no tiene soporte incorporado para la evaluación perezosa. Algunas implementaciones, especialmente SML / NJ, tienen primitivas de evaluación diferida no estándar, pero los programas que usan esas primitivas no serán portátiles. Las suspensiones perezosas también pueden implementarse de manera portátil, utilizando el sistema de módulos Standard ML.

Primero definimos una interfaz, o *firma* , para manipular suspensiones perezosas:

```
signature LAZY =
sig
  type 'a lazy

  val pure : 'a -> 'a lazy
  val delay : ('a -> 'b) -> 'a -> 'b lazy
  val force : 'a lazy -> 'a

  exception Diverge

  val fix : ('a lazy -> 'a) -> 'a
end
```

Esta firma indica que:

- El constructor de tipos de suspensiones perezosas es *abstracto* : su representación interna está oculta (e irrelevante para) los usuarios.
- Hay dos formas de crear una suspensión: envolviendo directamente su resultado final y retrasando una aplicación de función.
- Lo único que podemos hacer con una suspensión es forzarla. Cuando se fuerza una suspensión retrasada por primera vez, su resultado se memoriza, por lo que la próxima vez no será necesario volver a calcular el resultado.
- Podemos crear valores autorreferenciales, donde la autorreferencia pasa por una suspensión. De esta manera podemos crear, por ejemplo, un flujo lógicamente infinito que contiene el mismo elemento repetido, como en el siguiente fragmento de código de Haskell:

```
-- Haskell, not Standard ML!
xs :: [Int]
xs = 1 : xs
```

Después de definir la interfaz, tenemos que proporcionar una implementación real, también conocida como módulo o *estructura* :

```
structure Lazy :> LAZY =
struct
  datatype 'a state
    = Pure of 'a
    | Except of exn
    | Delay of unit -> 'a

  type 'a lazy = 'a state ref

  fun pure x = ref (Pure x)
  fun delay f x = ref (Delay (fn _ => f x))
  fun compute f = Pure (f ()) handle e => Except e
  fun force r =
    case !r of
        Pure x => x
      | Except e => raise e
      | Delay f => (r := compute f; force r)

  exception Diverge

  fun fix f =
    let val r = ref (Except Diverge)
    in r := compute (fn _ => f r); force r end
end
```

Esta estructura indica que una suspensión se representa internamente como una célula mutable, cuyo estado interno es uno de los siguientes:

- `Pure x` , si la suspensión ya fue forzada, y su resultado final es `x` .
- `Except e` , si la suspensión ya fue forzada, y se lanzó una excepción en el proceso.
- `Delay f` , si la suspensión no fue forzada todavía, y su resultado final se puede obtener evaluando `f ()` .

Además, debido a que utilizamos una *suscripción opaca* (`:>`), la representación interna del tipo de suspensiones está oculta fuera del módulo.

Aquí está nuestro nuevo tipo de suspensiones perezosas en acción:

```
infixr 5 :::
datatype 'a stream = NIL | ::: of 'a * 'a stream Lazy.lazy

(* An infinite stream of 1s, as in the Haskell example above *)
val xs = Lazy.fix (fn xs => 1 ::: xs)

(* Haskell's Data.List.unfoldr *)
fun unfoldr f x =
  case f x of
      NONE => NIL
    | SOME (x, y) => x ::: Lazy.delay (unfoldr f) y

(* Haskell's Prelude.iterate *)
fun iterate f x = x ::: Lazy.delay (iterate f o f) x

(* Two dummy suspensions *)
val foo = Lazy.pure 0
val bar = Lazy.pure 1
```

```
(* Illegal, foo and bar have type `int Lazy.lazy`,
 * whose internal representation as a mutable cell is hidden *)
val _ = (foo := !bar)
```

Lea Sistema de módulos en línea: https://riptutorial.com/es/sml/topic/7013/sistema-de-modulos

Capítulo 5: Tipos numericos

Sintaxis

- Los números reales deben comenzar con uno o más dígitos seguidos de un período seguido de uno o más dígitos.
- **~** es el operador para denotar números negativos
- **div** es el operador para la división entera.
- **/** Es el operador de la división real.

Examples

Entero

Conceptos básicos de enteros

```
Standard ML of New Jersey v110.78 [built: Thu Jul 23 11:21:58 2015]
- 6;
val it = 6 : int
- ~6;
val it = ~6 : int
- 6 + ~6;
val it = 0 : int
```

División entera

```
- 6 div 3;
val it = 2 : int
- 6 div 4;
val it = 0 : int
- 3 div 6;
val it = 0 : int
```

Límites de valor entero

Uso de funciones de biblioteca de base de enteros

```
- Int.maxInt;
val it = SOME 1073741823 : int option
- Int.minInt;
val it = SOME ~1073741824 : int option
```

Real

Conceptos básicos de números reales

```
- 6.0;
val it = 6.0 : real
```

```
- ~6.0;
val it = ~6.0 : real
- 6.0 + ~6.0;
val it = 0.0 : real
- 6.0 / 3.0;
val it = 2.0 : real
- 4.0 / 6.0;
val it = 0.666666666667 : real
```

Límites de valor real

Uso de las funciones de la biblioteca Real Basis

```
- Real.maxFinite;
val it = 1.79769313486E308 : real
- Real.minPos;
val it = 4.94065645841E~324 : real
- Real.minNormalPos;
val it = 2.22507385851E~308 : real
```

infinito

```
- Real.posInf;
val it = inf : real
- Real.negInf;
val it = ~inf : real
```

Coerción de valores reales a enteros

Redondeo

Los valores intermedios entre dos enteros van hacia el valor par más cercano.

```
- round(4.5);
val it = 4 : int
- round(3.5);
val it = 4 : int
```

Truncamiento

```
val it = 4 : int
- trunc(4.5);
val it = 4 : int
- trunc(3.5);
val it = 3 : int
```

Piso y techo

```
- ceil(4.5);
val it = 5 : int
- floor(4.5);
val it = 4 : int
```

Error de operador aritmético con tipos numéricos mixtos

No se puede agregar Integer y Real *

```
- 5 + 1.0;
stdIn:1.2-10.4 Error: operator and operand don't agree [overload conflict]
  operator domain: [+ ty] * [+ ty]
  operand:         [+ ty] * real
  in expression:
    5 + 1.0
```

Coersión de valor entero a real

```
- real(6);
val it = 6.0 : real
```

Lea Tipos numericos en línea: https://riptutorial.com/es/sml/topic/7010/tipos-numericos

Creditos

S. No	Capítulos	Contributors
1	Empezando con sml	4444, ben rudgers, Community, Simon Shine
2	Comentarios	ben rudgers
3	Programación interactiva utilizando el REPL	ben rudgers, Nick
4	Sistema de módulos	pyon
5	Tipos numericos	ben rudgers, pyon