eBook Gratuit

APPRENEZ sml

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#sml

Table des matières

À propos

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

Chapitre 1: Commencer avec sml

Remarques

Cette section fournit une vue d'ensemble de ce qu'est sml et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans sml, et établir un lien avec les sujets connexes. La documentation de sml étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Installation

Il existe une douzaine de mises en œuvre de la norme ML. MLton produit un code très optimisé, mais n'a pas de REPL . SML / NJ est le plus utilisé, mais présente des messages d'erreur légèrement difficiles à apprendre. Moscow ML et Poly / ML sont faciles à utiliser, mais ne prennent pas en charge le format de package .mlb. Ce n'est pas essentiel pour commencer, cependant.

Vous trouverez ci-dessous des instructions pour l'installation de SML / NJ, Moscow ML et Poly / ML en fonction du système d'exploitation.

Sous Windows

SML / NJ:

- Rendez-vous sur http://www.smlnj.org/dist/working/ et trouvez la dernière version, par exemple 110.80 Fichiers de distribution .
- Faites défiler la liste et recherchez le programme d'installation Windows, par exemple smlnj-110.80.msi . Exécutez le programme d'installation.
- Vous avez maintenant une REPL dans par exemple `C:\Program Files (x86)\SML NJ\bin\sml.bat` .

Moscou ML:

- Allez sur http://mosml.org/ et cliquez sur "Télécharger Win. Installer". Exécutez le programme d'installation.
- Vous avez maintenant une REPL dans par exemple `C:\Program Files (x86)\mosml\bin\mosml.exe` .

Utiliser Homebrew sur MacOS

SML / NJ:

- Exécutez `brew install smlnj` tant que votre propre utilisateur. Testez REPL avec `smlnj` .

Moscou ML:

- Allez à http://mosml.org/ et cliquez sur "Télécharger le fichier PKG". Exécutez le programme d'installation.
- *Missing ... Test REPL Comment? Est-ce dans `$PATH` maintenant?*

Sur Ubuntu / Debian Linux

SML / NJ:

- Exécutez `sudo apt-get install smlnj` tant que super utilisateur. Testez REPL avec `smlnj` .

Moscou ML:

- *(Ubuntu)* Ajoutez le PPA en tant que super utilisateur. Testez REPL avec `mosml` .

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

Ajout du support readline

Afin de pouvoir utiliser les touches fléchées pour naviguer dans les lignes précédemment saisies dans la REPL, la plupart des compilateurs SML peuvent bénéficier du programme `rlwrap` . En utilisant Homebrew sur MacOS, installez-le par `brew install rlwrap` , et sur Ubuntu / Debian Linux, installez-le par `sudo apt-get install rlwrap` . Ensuite, dans le terminal, essayez ce qui suit:

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

Ces alias peuvent être ajoutés par exemple à votre `~/.bashrc` pour qu'ils fonctionnent par défaut.

La touche fléchée devrait maintenant fonctionner mieux.

Lire Commencer avec sml en ligne: https://riptutorial.com/fr/sml/topic/6953/commencer-avec-sml

Chapitre 2: commentaires

Syntaxe

- (* ouvre un commentaire de bloc
- *) ferme un commentaire de bloc
- (* et *) doit être équilibré en nombre

Examples

Tous les commentaires sont des commentaires de bloc

```
(***********************************************
*  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 *)
```

Commentaires imbriqués

```
(* 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 *)
```

Lire commentaires en ligne: https://riptutorial.com/fr/sml/topic/6976/commentaires

Chapitre 3: Programmation interactive utilisant le REPL

Syntaxe

- Contrairement aux fichiers de code source, le point-virgule ';' est obligatoire pour terminer chaque expression dans le REPL.

Examples

Démarrer le SMLNJ REPL

REPL signifie «Read Evaluate Print Loop». Le REPL peut être utilisé pour écrire et exécuter du code sur une ligne à la fois et constitue une alternative à l'écriture de code dans un fichier, puis à la compilation ou à l'interprétation de l'intégralité du fichier avant son exécution.

Pour lancer SMLNJ REPL à partir d'une invite de commande:

```
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>
```

Dans les shells de commande Bash et similaires, la fonctionnalité readline GNU peut être ajoutée à la REPL SML à l'aide de la commande système `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>
```

En l'utilisant'

Toutes les expressions SML renvoient une valeur. Le REPL stocke la valeur de retour de la dernière expression évaluée. `it` fournit la valeur de la dernière expression évaluée dans la 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>
```

En effet, les commentaires ne sont pas évalués par la REPL et n'en changent pas la valeur.

```
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>
```

Lire Programmation interactive utilisant le REPL en ligne:
https://riptutorial.com/fr/sml/topic/6975/programmation-interactive-utilisant-le-repl

Chapitre 4: Système de module

Examples

Évaluation paresseuse

La norme ML ne prend pas en charge l'évaluation paresseuse. Certaines implémentations, notamment SML / NJ, ont des primitives d'évaluation paresseuse non standard, mais les programmes utilisant ces primitives ne seront pas portables. Les suspensions paresseuses peuvent également être implémentées de manière portable, en utilisant le système de module standard de ML.

Nous définissons d'abord une interface, ou *signature* , pour manipuler des suspensions paresseuses:

```
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
```

Cette signature indique que:

- Le constructeur de type des suspensions paresseuses est *abstrait* - sa représentation interne est cachée (et non pertinente) aux utilisateurs.
- Il existe deux manières de créer une suspension: en encapsulant directement son résultat final et en retardant une application.
- La seule chose que nous pouvons faire avec une suspension est de la forcer. Lorsqu'une suspension retardée est forcée pour la première fois, son résultat est mémorisé, de sorte que la prochaine fois que le résultat n'aura pas à être recalculé.
- Nous pouvons créer des valeurs auto-référentielles, où la référence à soi-même passe par une suspension. De cette façon, nous pouvons créer, par exemple, un flux logiquement infini contenant le même élément répété, comme dans l'extrait de code Haskell suivant:

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

Après avoir défini l'interface, nous devons fournir une implémentation réelle, également appelée module ou *structure* :

```
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
```

Cette structure indique qu'une suspension est représentée en interne en tant que cellule mutable, dont l'état interne est l'un des suivants:

- `Pure x` , si la suspension était déjà forcée, et son résultat final est `x` .
- `Except e` , si la suspension était déjà forcée, et une exception a été lancée dans le processus.
- `Delay f` , si la suspension n'a pas encore été forcée, et son résultat final peut être obtenu en évaluant `f ()` .

De plus, comme nous avons utilisé *une attribution opaque* (`:>`), la représentation interne du type de suspension est cachée en dehors du module.

Voici notre nouveau type de suspensions paresseuses en action:

```
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)
```

Lire Système de module en ligne: https://riptutorial.com/fr/sml/topic/7013/systeme-de-module

Chapitre 5: Types numériques

Syntaxe

- Les nombres réels doivent commencer par un ou plusieurs chiffres, suivis d'un point suivi d'un ou de plusieurs chiffres.
- **~** est l'opérateur pour dénoter les nombres négatifs
- **div** est l'opérateur de la division entière.
- **/** est l'opérateur pour la division réelle.

Examples

Entier

Notions de base

```
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
```

Division entière

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

Entier Valeur Limite

Utilisation des fonctions de bibliothèque de base entières

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

Réal

Notions de base sur les nombres réels

```
- 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
```

Valeur réelle des limites

Utilisation des fonctions de bibliothèque de base réelle

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

Infini

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

Coercition des valeurs réelles aux entiers

Arrondi

Les valeurs à mi-chemin entre deux nombres entiers vont vers la valeur paire la plus proche.

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

Troncature

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

Plancher et plafond

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

Erreur d'opérateur arithmétique avec des types numériques mixtes

Impossible d'ajouter des nombres entiers et réels *

```
- 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
```

Cohésion de la valeur entière à la valeur réelle

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

Lire Types numériques en ligne: https://riptutorial.com/fr/sml/topic/7010/types-numeriques

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec sml	4444, ben rudgers, Community, Simon Shine
2	commentaires	ben rudgers
3	Programmation interactive utilisant le REPL	ben rudgers, Nick
4	Système de module	pyon
5	Types numériques	ben rudgers, pyon