Compilation

Le langage Hopix

Adrien Guatto

Master 1 Informatique
2022–2023

Le langage Hopix

Les grandes lignes

- Fonctionnel : fonctions anonymes, d'ordre supérieur ; "orienté valeur"
- Impératif : évaluer une expression produit une valeur et *a un effet*.
- Typé : les programmes mal typés sont rejetés *avant* l'exécution.

Le langage Hopix

Les grandes lignes

- Fonctionnel : fonctions anonymes, d'ordre supérieur ; "orienté valeur"
- Impératif : évaluer une expression produit une valeur et *a un effet*.
- Typé : les programmes mal typés sont rejetés *avant* l'exécution.

Les détails : MiniML

- Du polymorphisme de rang 1 (schémas de types).
- Des types enregistrement, des types sommes et du filtrage (`match`).
- Des références modifiables allouées dynamiquement.
- Des boucles dénombrées, "tant que" et "jusqu'à ce que".

Le langage Hopix

Les grandes lignes

- Fonctionnel : fonctions anonymes, d'ordre supérieur ; "orienté valeur"
- Impératif : évaluer une expression produit une valeur et *a un effet*.
- Typé : les programmes mal typés sont rejetés *avant* l'exécution.

Les détails : MiniML

- Du polymorphisme de rang 1 (schémas de types).
- Des types enregistrement, des types sommes et du filtrage (`match`).
- Des références modifiables allouées dynamiquement.
- Des boucles dénombrées, "tant que" et "jusqu'à ce que".

On va découvrir le langage par parcours descendant de sa syntaxe.

Qu'est-ce qu'un programme Hopix ?

- Une liste de définitions de types et valeurs, et de déclaration externes.
- L'exécuter, c'est évaluer dans l'ordre les définitions de valeur.
- Les valeurs définies sont invisibles, seul importe l'effet de la définition.

Qu'est-ce qu'un programme Hopix ?

- Une liste de définitions de types et valeurs, et de déclaration externes.
- L'exécuter, c'est évaluer dans l'ordre les définitions de valeur.
- Les valeurs définies sont invisibles, seul importe l'effet de la définition.

```
type program = definition list

and definition =
  | DefineValue of value_definition
  | DefineType of type_constructor
                * type_variable list
                * type_definition
  | DeclareExtern of identifier * type_scheme
```

Qu'est-ce qu'un programme Hopix ?

- Une liste de définitions de types et valeurs, et de déclaration externes.
- L'exécuter, c'est évaluer dans l'ordre les définitions de valeur.
- Les valeurs définies sont invisibles, seul importe l'effet de la définition.

```
type program = definition list

and definition =
  | DefineValue of value_definition
  | DefineType of type_constructor
                * type_variable list
                * type_definition
  | DeclareExtern of identifier * type_scheme
```

Un point de vue commun sur le typage

Le sens ("la sémantique") d'un programme ne dépend pas des types.

Qu'est-ce qu'une définition de type ?

```
type type_definition =
  (** A sum type for tagged values
      [K₁ (ty₁₁, ..., ty₁ ) | ... | K  (ty ₁, ..., ty  )]. *)
  | DefineSumType of (constructor * ty list) list
  (** A record type { l₁ : ty₁, ..., l  : ty }. *)
  | DefineRecordType of (label * ty) list
  (** A type with no visible definition. *)
  | Abstract
```

- Des types somme (“algèbriques”) récursifs.
- Des types enregistrements, formés par des listes de champs typés.
- Des types abstraits, c'est-à-dire sans définition.

Qu'est-ce qu'un type ? Qu'un schéma de type ?

```
type ty =
  (** An instantiated type constructor [t <ty₁, ..., ty >]. *)
  | TyCon of type_constructor * ty list
  (** A function type [ty₁ → ty₂]. *)
  | TyArrow of ty * ty
  (** A tuple type [ty₁ * ... * ty ]. *)
  | TyTuple of ty list
  (** A type variable ['a]. *)
  | TyVar of type_variable

type type_scheme =
  ForallTy of type_variable list * ty located
```

Qu'est-ce qu'un type ? Qu'un schéma de type ?

```
type ty =
  (** An instantiated type constructor [t <ty₁, ..., ty >]. *)
  | TyCon of type_constructor * ty list
  (** A function type [ty₁ → ty₂]. *)
  | TyArrow of ty * ty
  (** A tuple type [ty₁ * ... * ty ]. *)
  | TyTuple of ty list
  (** A type variable ['a]. *)
  | TyVar of type_variable

type type_scheme =
  ForallTy of type_variable list * ty located
```

Le polymorphisme de rang 1

Les types de la forme $(\forall \alpha.\tau_1) \to \tau_2$ ne font pas partie du langage.

Qu'est-ce qu'un définition de valeur ?

```
type value_definition =
  (** A toplevel definition for a value. *)
  | SimpleValue of expression polymorphic_definition
  (** A toplevel definition for mutually recursive functions. *)
  | RecFunctions of function_definition polymorphic_definition list

and 'a polymorphic_definition =
  identifier * type_scheme option * 'a
```

Qu'est-ce qu'une expression ? Qu'un motif ?

Voir `hopixAST.ml`.