Compilation

Le compilateur flap

Adrien Guatto

Master 1 Informatique
2022–2023

La compilation, d'hier...

> *COMPILER, verbe trans. : rassembler en un seul ouvrage des extraits provenant de sources différentes.*
>
> – *Trésor de la langue française informatisé*

La compilation, d'hier...

COMPILER, verbe trans. : rassembler en un seul ouvrage des extraits provenant de sources différentes.

– Trésor de la langue française informatisé

© *Computer History Museum*

Le premier compilateur : A-0

- *Arithmetic Language version 0*
- Grace Murray Hopper, 1951-1952, UNIVAC I
- Amalgame des sous-routines

The Education of a Computer – Grace Murray Hopper

La compilation, d'hier...

COMPILER, verbe trans. : rassembler en un seul ouvrage des extraits provenant de sources différentes.

– Trésor de la langue française informatisé

© *Computer History Museum*

Le premier compilateur : A-0

- *Arithmetic Language version 0*
- Grace Murray Hopper, 1951-1952, UNIVAC I
- Amalgame des sous-routines

The Education of a Computer – Grace Murray Hopper

`https://en.wikipedia.org/wiki/History_of_compiler_construction`

... à aujourd'hui ...

... à aujourd'hui ...

GCC 13.2

- Sources : C, C++, Objective-C, FORTRAN, Ada, Go, D, Modula-2.
- Cibles : x86, ARM, POWER, RISC-V... (26 architectures).
- Implémentation : C++ et C++ ; **4** MLoC.

... à aujourd'hui ...

GCC 13.2

- Sources : C, C++, Objective-C, FORTRAN, Ada, Go, D, Modula-2.
- Cibles : x86, ARM, POWER, RISC-V... (26 architectures).
- Implémentation : C++ et C++ ; **4** MLoC.

Clang + LLVM 17.0

- Sources : C, C++, Objective-C.
- Cibles : x86, ARM, POWER.
- Implémentation : C++ ; **1,2+2,3** MLoC.

... à aujourd'hui ...

GCC 13.2

- Sources : C, C++, Objective-C, FORTRAN, Ada, Go, D, Modula-2.
- Cibles : x86, ARM, POWER, RISC-V... (26 architectures).
- Implémentation : C++ et C++ ; **4** MLoC.

Clang + LLVM 17.0

- Sources : C, C++, Objective-C.
- Cibles : x86, ARM, POWER.
- Implémentation : C++ ; **1,2+2,3** MLoC.

CompCert 3.13.1

- Source : C.
- Cibles : x86, ARM, POWER.
- Implémentation : Coq et OCaml ; **33** KLoC.

... et maintenant

flap 1.0

- Source : Hopix.
- Cible : x86-64.
- Implémentation : OCaml ; 11 KLoC pour le code complet.

Comment aborder flap ?

... et maintenant

flap 1.0

- Source : Hopix.
- Cible : x86-64.
- Implémentation : OCaml ; 11 KLoC pour le code complet.

Comment aborder flap ? En première approche, ce n'est qu'une fonction

```
val flap : string -> string.
```

En pratique, les chaînes d'entrée sont lues/écrites dans des fichiers.

… et maintenant

flap 1.0

- Source : Hopix.
- Cible : x86-64.
- Implémentation : OCaml ; 11 KLoC pour le code complet.

Comment aborder flap ? En première approche, ce n'est qu'une fonction

```
val flap : string -> string.
```

En pratique, les chaînes d'entrée sont lues/écrites dans des fichiers.

Une spécification informelle pour flap

"Si le compilateur termine sans erreur, la chaîne fournie contenait du code source valide, et la chaîne produite contient du code cible valide qui est équivalent au code source."

Langage source et langage cible

Langage source et langage cible

Hopix

- Un langage fonctionnel et impératif typé.
  - Fonctions d'ordre supérieur, types algébriques, références, boucles.
- Y penser comme à un petit sous-ensemble d'OCaml.
  - Manquent tableaux, exceptions, modules, objets, variants polymorphes, arguments nommés et optionnels, effets algébriques...

Langage source et langage cible

Hopix

- Un langage fonctionnel et impératif typé.
  - Fonctions d'ordre supérieur, types algébriques, références, boucles.
- Y penser comme à un petit sous-ensemble d'OCaml.
  - Manquent tableaux, exceptions, modules, objets, variants polymorphes, arguments nommés et optionnels, effets algébriques...

Langage d'assemblage ("assembleur") x86-64

- Langage natif des processeurs Intel et AMD utilisés dans les PC.
- Instructions élémentaires très explicites pour :
  - les calculs arithmétiques et logiques ;
  - la manipulation du flot de contrôle ;
  - l'accès à la mémoire et l'administration du processeur.
- En pratique, plutôt désagréable, même pour un langage de sa famille.

Un programme Hopix simple et sa traduction optimisée

Un programme Hopix simple et sa traduction optimisée

```
fun fib n =
  if n < 1
  then 1 else fib(n-1)+fib(n-2)
```

Un programme Hopix simple et sa traduction optimisée

```
fun fib n =
  if n < 1
  then 1 else fib(n-1)+fib(n-2)
```

```
f1:     pushq   %rbp
        xorl    %ebp, %ebp
        pushq   %rbx
        movq    %rdi, %rbx
        pushq   %rcx
.L3:    testq   %rbx, %rbx
        js      .L5
        movq    %rbx, %rdi
        decq    %rbx
        call    f1
        addl    %eax, %ebp
        jmp     .L3
.L5:    leal    1(%rbp), %eax
        popq    %rdx
        popq    %rbx
        popq    %rbp
        ret
```

Un programme Hopix simple et sa traduction optimisée

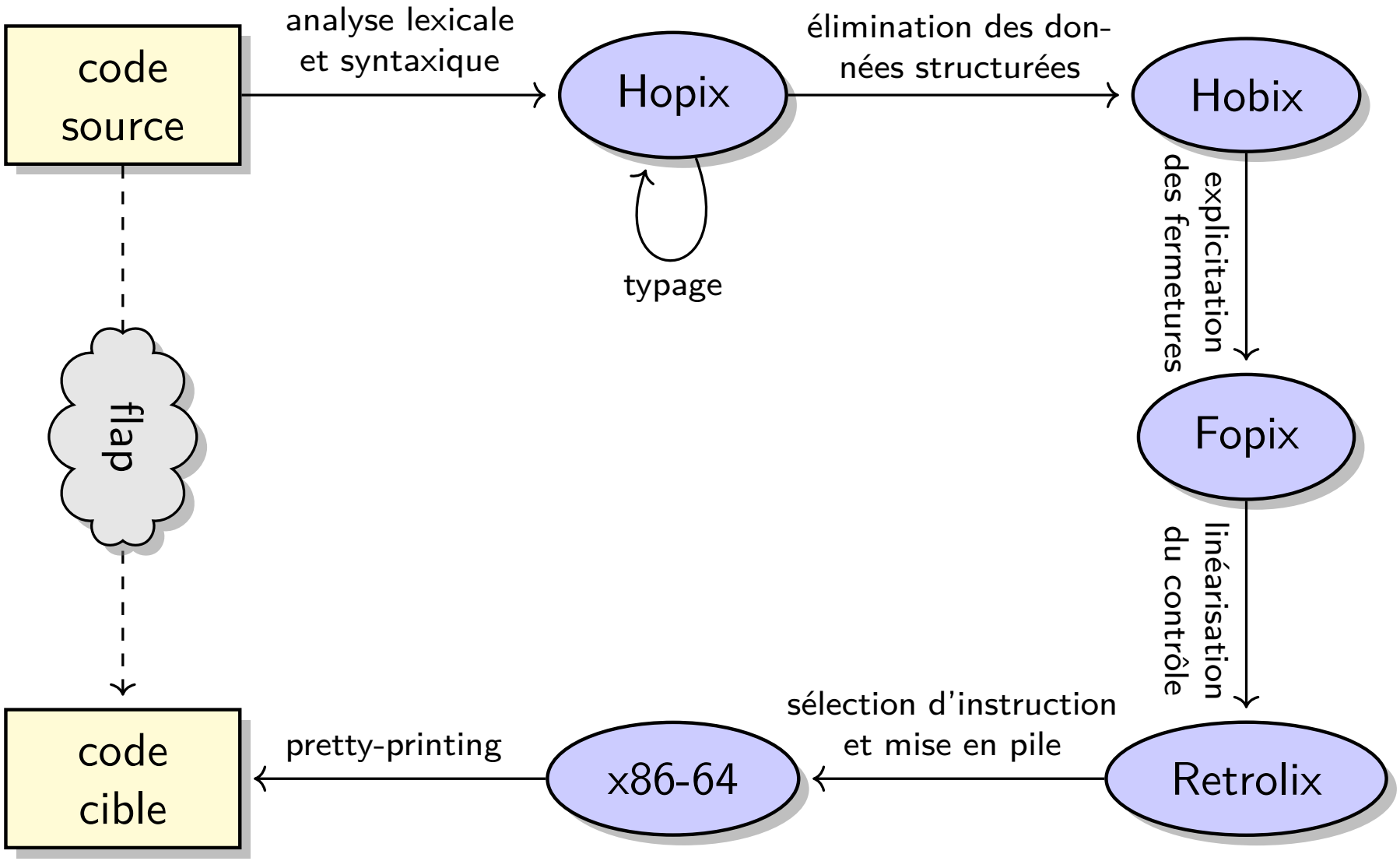
```
fun fib n =
  if n < 1
  then 1 else fib(n-1)+fib(n-2)
```

Morale

- La distance est déjà importante malgré la simplicité de l'exemple.
- On va donc procéder par étape.

```
f1:     pushq   %rbp
        xorl    %ebp, %ebp
        pushq   %rbx
        movq    %rdi, %rbx
        pushq   %rcx
.L3:    testq   %rbx, %rbx
        js      .L5
        movq    %rbx, %rdi
        decq    %rbx
        call    f1
        addl    %eax, %ebp
        jmp     .L3
.L5:    leal    1(%rbp), %eax
        popq    %rdx
        popq    %rbx
        popq    %rbp
        ret
```

Le flot de compilation de flap



L'infrastructure de flap

Principes généraux

- Tous les langages, intermédiaires ou non, sont sur un pied d'égalité.
- Ils viennent avec un interprète qui permet de tester la compilation.
- Le flot de compilation peut être librement interrompu et repris.

L'infrastructure de flap

Principes généraux

- Tous les langages, intermédiaires ou non, sont sur un pied d'égalité.
- Ils viennent avec un interprète qui permet de tester la compilation.
- Le flot de compilation peut être librement interrompu et repris.

Conséquences

- Tous les langages doivent être traités uniformément.
  - On doit pouvoir consommer, produire, interpréter du code source.
- Flap définit donc une interface pour les langages et les compilateurs.

Un langage de programmation, pour flap (1/3)

```
module type Language = sig
  (** A language has a [name]. *)
  val name : string

  (** A language has a file [extension]. *)
  val extension : string

  (** {1 Syntax} *)

  (** A syntax is defined by the type of abstract syntax trees. *)
  type ast

  (** [parse_filename f] turns the content of file [f] into an abstract syntax
      tree. An {! Error} is raised if this content is syntactically invalid. *)
  val parse_filename : string -> ast

  (** [parse_string c] is the same as [parse_filename] except that the source
      code is directly given as a string. *)
  val parse_string : string -> ast

  (** [print_ast] turns an abstract syntax tree into a human-readable string. *)
  val print_ast : ast -> string

  ...
```

Un langage de programmation, pour flap (2/3)

```
...

(** {2 Semantics} *)

(** A [runtime] environment contains all the information necessary
    to evaluate a program. *)
type runtime

(** An [observable] is the type of program results, as returned in the
    interactive loop. *)
type observable

(** Program evaluation begins in the [initial_runtime]. *)
val initial_runtime : unit -> runtime

(** [evaluate runtime p] executes the program [p] and produces a new runtime
    as well as the observable resulting from this runtime. *)
val evaluate : runtime -> ast -> runtime * observable

(** [print_observable] turns an observable into a human-readable string. *)
val print_observable : runtime -> observable -> string

...
```

Un langage de programmation, pour flap (3/3)

```
  ...

  (** {3 Type Checking} *)

  (** A typing environment stores static information about the program. *)
  type typing_environment

  (** The initial typing environment contains predefined static information,
      like the type of constants. *)
  val initial_typing_environment : unit -> typing_environment

  (** [typecheck env p] checks whether the program [p] is well-formed and
      extends the typing environment accordingly. If [p] is not well-formed an
      {!Error} is issued. *)
  val typecheck : typing_environment -> ast -> typing_environment

  (** [print_typing_environment] returns a human-readable representation of a
      typing environment. *)
  val print_typing_environment : typing_environment -> string
end
```

Un compilateur, pour flap

```
module type Compiler = sig
  module Source : Language
  module Target : Language

  (** It is often necessary to maintain some information about a program during
      its compilation. The [environment] type describes this information. *)
  type environment

  (** Program compilation begins in the result of [initial_environment ()]. *)
  val initial_environment : unit -> environment

  (** [translate source env] returns a [target] program semantically equivalent
      to [source], as a well as an enriched environment [env] that contains
      information produced during the compilation of [source]. *)
  val translate : Source.ast -> environment -> Target.ast * environment
end

(** [register compiler] integrates [compiler] is the set of flap's compilers. *)
val register : (module Compiler) -> unit
```

Organisation de flap

```
$ tree src
src/
├── commandLineOptions.ml
├── common
│   ├── compilers.mli
│   ├── languages.mli
│   ├── ...
├── dune
├── flap.ml
├── hopix
│   ├── hopix.ml
│   ├── hopixAST.ml
│   ├── hopixLexer.mll
│   ├── hopixParser.mly
│   ├── hopixInterpreter.ml
│   ├── hopixTypechecker.ml
│   ├── ...
├── hobix
│   ├── hobix.ml
│   ├── hobixAST.ml
│   ├── hobixInterpreter.ml
│   ├── ...
│   └── hopixToHobix.ml
├── fopix
│   ├── fopix.ml
│   ├── fopixAST.ml
│   ├── ...
│   ├── hobixToFopix.ml
├── utilities
│   ├── dict.ml
│   ├── dict.mli
│   ├── digraph.ml
│   ├── digraph.mli
│   ├── error.ml
│   ├── error.mli
│   ├── ...
└── version.ml
```

Conclusion

- Flap est une base de code fortement structurée.
- Vous devez la lire et jouer avec pour bien la comprendre.
- Posez des questions en cours, TP ou sur la liste de diffusion.

Conclusion

- Flap est une base de code fortement structurée.
- Vous devez la lire et jouer avec pour bien la comprendre.
- Posez des questions en cours, TP ou sur la liste de diffusion.

La suite

Assez de logistique, place au jalon 1 !