Programming Language Semantics and Compiler Design

(Sémantique des Langages de Programmation et Compilation)

Preamble

Frédéric Lang & Laurent Mounier
firstname.lastname@univ-grenoble-alpes.fr
Univ. Grenoble Alpes, Inria,
Laboratoire d'Informatique de Grenoble & Verimag

Master of Sciences in Informatics at Grenoble (MoSIG)
Master 1 info

Univ. Grenoble Alpes - UFR IM^{2}AG
www.univ-grenoble-alpes.fr — im2ag.univ-grenoble-alpes.fr

Some practical information

6 ECTS (60 hours).

Lecture sessions: 2 × 90 min / week

- Frédéric Lang (INRIA - Convecs): language semantics
- Laurent Mounier (UGA - Verimag): compiler design
- Henri-Pierre Charles (CEA Leti): guest lecture

Exercise sessions: 2 × 90 min / week

- Gwenaël Delaval (M1 Info - Group 1)
- Zachary Assoumani (M1 Info - Group 2)
- Alexandre Bérard (M1 MoSiG - Group 1)
- Cristian Ené (M1 MoSiG - Group 2)
- Ghilhem Lacombe Ené (M1 MoSiG - Group 3)

Emails: `FirstName.LastName@univ-grenoble-alpes.fr`

Meetings are possible (on appointment).

Assessment (the rules of the game)

Final Exam (FE)

- coefficient: 1.4
- date: week 49 (December)
- 3 hours

Assessment (the rules of the game)

Final Exam (FE)

- coefficient: 1.4
- date: week 49 (December)
- 3 hours

Mid-term exam (ME)

- coefficient: 2/3 of 0.6
- program: language semantics
- date: first half of October.
- 1.5 hours

Assessment (the rules of the game)

Final Exam (FE)

- coefficient: 1.4
- date: week 49 (December)

- 3 hours

Mid-term exam (ME)

- coefficient: 2/3 of 0.6
- program: language semantics
- date: first half of October.

- 1.5 hours

Assessment (the rules of the game)

Final Exam (FE)

- coefficient: 1.4
- date: week 49 (December)

- 3 hours

Mid-term exam (ME)

- coefficient: 2/3 of 0.6
- program: language semantics
- date: first half of October.

- 1.5 hours

$$\boxed{\text{Final Grade} = \frac{1.4 \times \mathrm{FE} + 0.6 \times \mathrm{ME}}{2}}$$

References

Pedagogical Resources

All pedagogical resources are on the Moodle (slides and tutorials):

`https://im2ag-moodle.univ-grenoble-alpes.fr`

Some classical books:

A. Aho, R. Sethi and J. Ullman
Compilers: Principles, techniques and tools
InterEditions, 1989

H. R. Nielson and F. Nielson.
Semantics with Applications: An Appetizer.
Springer, March 2007. ISBN 978-1-84628-691-9

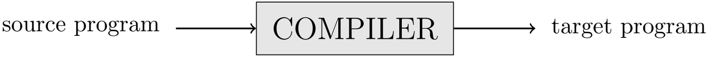
W. Waite and G. Goos.
Compiler Construction
Springer Verlag, 1984

R. Wilhelm and D. Maurer.
Compilers - Theory, construction, generation
Masson 1994

Compilers: what you surely already know…

A compiler is a language processor: it transforms a program:

- from a language we can understand: the programming language,
- to a language the machine can understand: the target language.



Global objectives of the course

- Programming languages, and the formalization of their meaning :
  - static and dynamic language semantics
- General compiler architecture.
- Some more detailed compiler techniques.

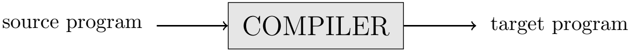
Global objectives of the course

- Programming languages, and the formalization of their meaning :
  - static and dynamic language semantics
- General compiler architecture.
- Some more detailed compiler techniques.

Basic objective

Study the translation performed by a compiler:

- the questions raised by the translation;
- the expected properties of this translation;
- how to perform this translation.



source program

COMPILER

target program

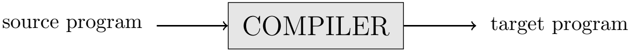
Global objectives of the course

- Programming languages, and the formalization of their meaning :
  - static and dynamic language semantics
- General compiler architecture.
- Some more detailed compiler techniques.

Basic objective

Study the translation performed by a compiler:

- the questions raised by the translation;
- the expected properties of this translation;
- how to perform this translation.



The algorithms and design principles used in compilers are generic and used in many domains of computer science and software engineering

Many programming language paradigms . . .

Imperative languages

- ex: FORTRAN, C, Ada, Java, Python, Rust, etc.
- notions: control structure, (explicit) memory assignment, expressions, types, . . .

Functional languages

- ex: ML, CAML, LISP, Scheme, etc
- notions: term reduction, function evaluation, recursion, high-order, types, . . .

Object-oriented languages

- ex: Java, Ada, Eiffel, C++, etc.
- notions: objects, classes, types, inheritance, polymorphism, . . .

Logical languages

- ex: Prolog
- notions: resolution, unification, predicate calculus, . . .

Web languages

- ex: JavaScript, PHP, HTML
- notions: scripts, markers, . . .

etc.

. . . and many architectures to target ...

- Complex instruction set computer (CISC)
- Reduced instruction set computer (RISC)
- VLIW, multi-processor & multi-core architectures
- dedicated processors (DSP, . . .)
- embedded systems (mobile phones, IoT, . . .)
- industrial systems (SCADA, Programmable Logic Controller)
- etc.

. . . and many architectures to target ...

- Complex instruction set computer (CISC)
- Reduced instruction set computer (RISC)
- VLIW, multi-processor & multi-core architectures
- dedicated processors (DSP, . . .)
- embedded systems (mobile phones, IoT, . . .)
- industrial systems (SCADA, Programmable Logic Controller)
- etc.

... while addressing numerous (conflicting ?) challenges:

- reliability
- performances
- low energy and resource consumption
- security
- etc.

We will mainly focus on:

Imperative languages

- data structures
  - basic types (integers, characters, pointers, etc)
  - user-defined types (enumeration, unions, arrays, . . .)
- control structures
  - assignments
  - iterations, conditionals, sequence
  - nested blocks, sub-programs

We will mainly focus on:

Imperative languages

- data structures
  - basic types (integers, characters, pointers, etc)
  - user-defined types (enumeration, unions, arrays, . . .)
- control structures
  - assignments
  - iterations, conditionals, sequence
  - nested blocks, sub-programs

"Standard" general-purpose machine architecture: (e.g. ARM, iX86)

- heap, stack and registers
- arithmetic and logical binary operations
- conditional branches

Course programme (overview)

1. Introduction
2. Natural operational semantics
3. Structural operational semantics
4. Axiomatic semantics
5. compiler architecture
6. Static semantics of a language and type analysis
7. Intermediate-code generation
8. Code optimization
9. Machine-code generation
10. Some security issues
11. Dynamic Compilation and compilation for embedded systems (Henri-Pierre Charles, CEA Grenoble)

Credits

The content and materials used in this course have been mostly provided, improved and maintained by Yliès Falcone (INRIA Corse).

Many thanks as well to all the teachers and students (and students who became teachers!) who helped us to improve this course during the last years . . .