Introduction to the Tiger Project

Akim Demaille Étienne Renault Roland Levillain
first.*last*@lrde.epita.fr

EPITA — École Pour l'Informatique et les Techniques Avancées

February 5th, 2015

Context and Motivation

The Needs

- 15 years ago, EPITA asked for a long and challenging project.
- Should virtually be a *potpourri* of every subject from computer science courses taught in third year.

A (Miraculous) Solution

A compiler construction project.

Context and Motivation

The Needs

- 15 years ago, EPITA asked for a long and challenging project.
- Should virtually be a *potpourri* of every subject from computer science courses taught in third year.

A (Miraculous) Solution

A compiler construction project.

Goals

Aim

Compiler construction as a by-product

Complete Project

Specifications, implementation, documentation, testing, distribution.

Several iterations

7 (optionally up to 11) steps, for 3 (resp. up to 6) months.

Algorithmically challenging

Applied use of well known data structures and algorithms.

Team Management

Project conducted in group of four students.

Goals

Aim

Compiler construction as a by-product

Complete Project

Specifications, implementation, documentation, testing, distribution.

Several iterations

7 (optionally up to 11) steps, for 3 (resp. up to 6) months.

Algorithmically challenging

Applied use of well known data structures and algorithms.

Team Management

Project conducted in group of four students.

Goals

Aim

Compiler construction as a by-product

Complete Project

Specifications, implementation, documentation, testing, distribution.

Several iterations

7 (optionally up to 11) steps, for 3 (resp. up to 6) months.

Algorithmically challenging

Applied use of well known data structures and algorithms.

Team Management

Project conducted in group of four students.

Goals

Aim

Compiler construction as a by-product

Complete Project

Specifications, implementation, documentation, testing, distribution.

Several iterations

7 (optionally up to 11) steps, for 3 (resp. up to 6) months.

Algorithmically challenging

Applied use of well known data structures and algorithms.

Team Management

Project conducted in group of four students.

Goals

Aim

Compiler construction as a by-product

Complete Project

Specifications, implementation, documentation, testing, distribution.

Several iterations

7 (optionally up to 11) steps, for 3 (resp. up to 6) months.

Algorithmically challenging

Applied use of well known data structures and algorithms.

Team Management

Project conducted in group of four students.

Goals (cont.)

C++

Expressive power; uses both low- and high-level constructs; industry standard.

Object-Oriented (OO) Design and Design Pattern (DP)

Practice common OO idioms, apply DPs.

Development Tools

Autotools, Doxygen, Flex, Bison, GDB, Valgrind, Git, etc.

Understanding Computers

Compiler and languages are tightly related to computer architecture.

English

Everything is to be written in English (code, documentation, tests).

Goals (cont.)

C++
: Expressive power; uses both low- and high-level constructs; industry standard.

Object-Oriented (OO) Design and Design Pattern (DP)
: Practice common OO idioms, apply DPs.

Development Tools
: Autotools, Doxygen, Flex, Bison, GDB, Valgrind, Git, etc.

Understanding Computers
: Compiler and languages are tightly related to computer architecture.

English
: Everything is to be written in English (code, documentation, tests).

Goals (cont.)

C++
: Expressive power; uses both low- and high-level constructs; industry standard.

Object-Oriented (OO) Design and Design Pattern (DP)
: Practice common OO idioms, apply DPs.

Development Tools
: Autotools, Doxygen, Flex, Bison, GDB, Valgrind, Git, etc.

Understanding Computers
: Compiler and languages are tightly related to computer architecture.

English
: Everything is to be written in English (code, documentation, tests).

Goals (cont.)

C++
: Expressive power; uses both low- and high-level constructs; industry standard.

Object-Oriented (OO) Design and Design Pattern (DP)
: Practice common OO idioms, apply DPs.

Development Tools
: Autotools, Doxygen, Flex, Bison, GDB, Valgrind, Git, etc.

Understanding Computers
: Compiler and languages are tightly related to computer architecture.

English
: Everything is to be written in English (code, documentation, tests).

Goals (cont.)

C++
: Expressive power; uses both low- and high-level constructs; industry standard.

Object-Oriented (OO) Design and Design Pattern (DP)
: Practice common OO idioms, apply DPs.

Development Tools
: Autotools, Doxygen, Flex, Bison, GDB, Valgrind, Git, etc.

Understanding Computers
: Compiler and languages are tightly related to computer architecture.

English
: Everything is to be written in English (code, documentation, tests).

Non Goal

Writing a Compiler Paradoxically!

Well, at least considered a *secondary* issue.

- Why?

  The vast majority of [computer science] students are unlikely to ever design a compiler [Debray, 2002].

- But... Students interested in compiler construction should be given the opportunity to work on challenging, optional assignments.

Non Goal

Writing a Compiler Paradoxically!

Well, at least considered a *secondary* issue.

- Why?

  The vast majority of [computer science] students are unlikely to ever design a compiler [Debray, 2002].

- But... Students interested in compiler construction should be given the opportunity to work on challenging, optional assignments.

Non Goal

Writing a Compiler Paradoxically!
Well, at least considered a secondary issue.

- Why?

  The vast majority of [Computer Science] students are unlikely to ever design a compiler [Debray, 2002].

- But... Students interested in compiler construction should be given the opportunity to work on challenging, optional assignments.

Non Goal

Writing a Compiler Paradoxically!
Well, at least considered a *secondary* issue.

- Why?

> *The vast majority of [Computer Science] students are unlikely to ever design a compiler [Debray, 2002].*

- But... Students interested in compiler construction should be given the opportunity to work on challenging, optional assignments.

Non Goal

Writing a Compiler Paradoxically!
Well, at least considered a *secondary* issue.

- Why?

  The vast majority of [Computer Science] students are unlikely to ever design a compiler [Debray, 2002].

- But... Students interested in compiler construction should be given the opportunity to work on challenging, optional assignments.

Non Goal

Writing a Compiler Paradoxically!
Well, at least considered a *secondary* issue.

- Why?

  The vast majority of [Computer Science] students are unlikely to ever design a compiler [Debray, 2002].

- But... Students interested in compiler construction should be given the opportunity to work on challenging, optional assignments.

Non Goal

Writing a Compiler Paradoxically!
Well, at least considered a *secondary* issue.

- Why?

  The vast majority of [Computer Science] students are unlikely to ever design a compiler [Debray, 2002].

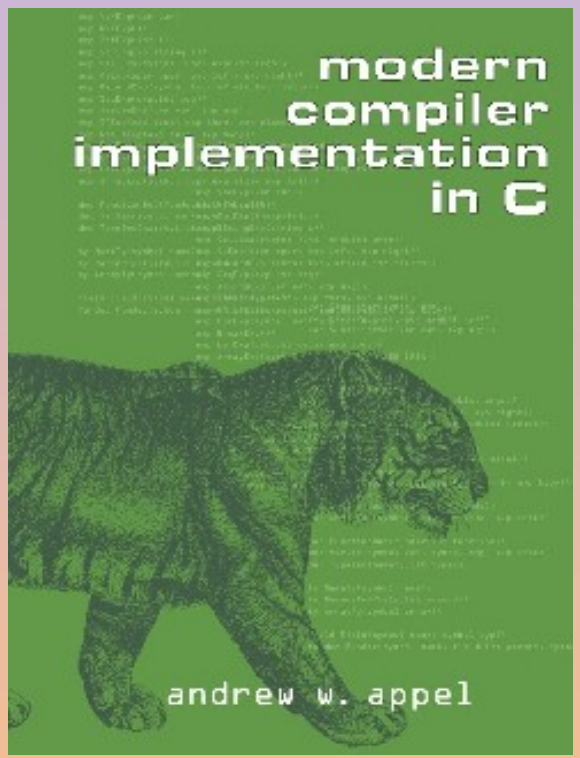
- But... Students interested in compiler construction should be given the opportunity to work on challenging, optional assignments.

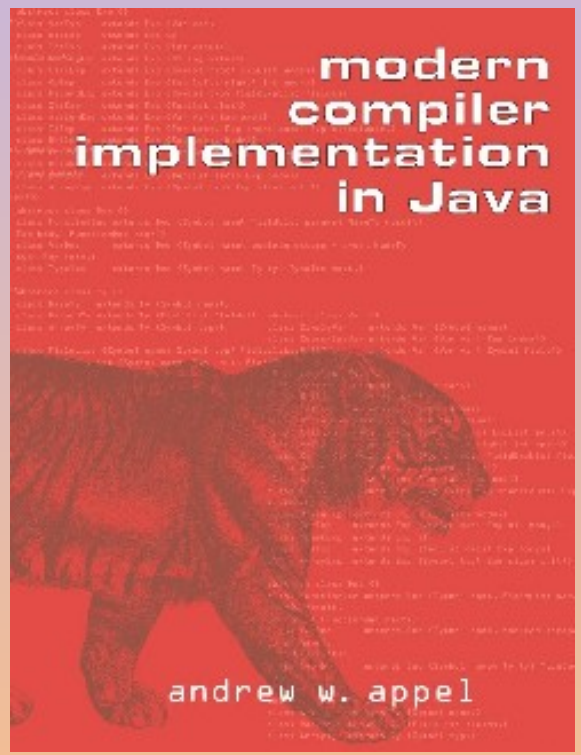
Introduction to the Tiger Project

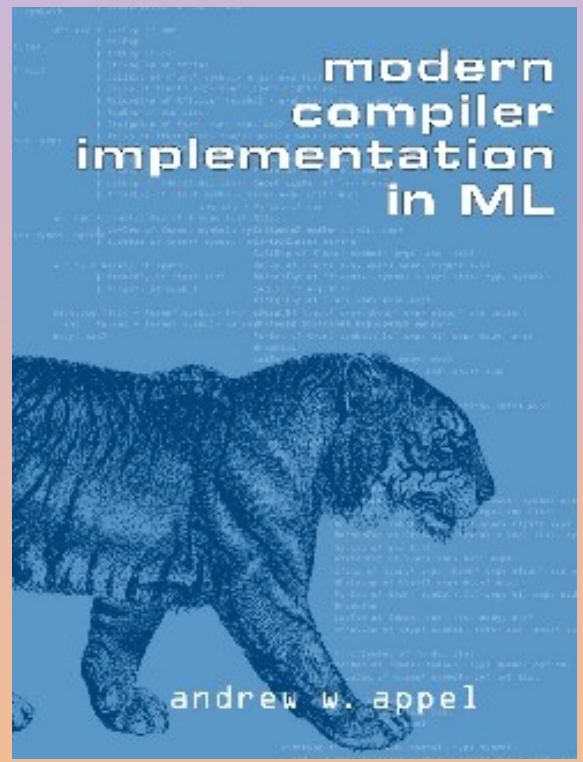
Overview of the Tiger Project

A Core-Curriculum Compiler-Construction Project

Based on Andrew Appel's Tiger language and *Modern Compiler Implementation* books [Appel, 1998].







A Core-Curriculum Compiler-Construction Project (cont.)

...and largely adapted [Demaille, 2005].

- Compiler (to be) written in C++.
- Initial Tiger language definition (a Pascal-descendant language, dressed in a clean ML-like syntax).
- Augmented with `import` statements, adjustable prelude, OO constructs, etc.
- Better defined (no implementation-defined behavior left).
- More compiler modules and features than in the initial design.
- In particular more *tools* to both help students develop and improve their compiler and make the maintenance easier to teachers and assistants.

A Core-Curriculum Compiler-Construction Project (cont.)

...and largely adapted [Demaille, 2005].

- Compiler (to be) written in C++.
- Initial Tiger language definition (a Pascal-descendant language, dressed in a clean ML-like syntax).
- Augmented with `import` statements, adjustable prelude, OO constructs, etc.
- Better defined (no implementation-defined behavior left).
- More compiler modules and features than in the initial design.
- In particular more *tools* to both help students develop and improve their compiler and make the maintenance easier to teachers and assistants.

A Core-Curriculum Compiler-Construction Project (cont.)

...and largely adapted [Demaille, 2005].

- Compiler (to be) written in C++.
- Initial Tiger language definition (a Pascal-descendant language, dressed in a clean ML-like syntax).
- Augmented with `import` statements, adjustable prelude, OO constructs, etc.
- Better defined (no implementation-defined behavior left).
- More compiler modules and features than in the initial design.
- In particular more *tools* to both help students develop and improve their compiler and make the maintenance easier to teachers and assistants.

A Core-Curriculum Compiler-Construction Project (cont.)

...and largely adapted [Demaille, 2005].

- Compiler (to be) written in C++.
- Initial Tiger language definition (a Pascal-descendant language, dressed in a clean ML-like syntax).
- Augmented with `import` statements, adjustable prelude, OO constructs, etc.
- Better defined (no implementation-defined behavior left).
- More compiler modules and features than in the initial design.
- In particular more *tools* to both help students develop and improve their compiler and make the maintenance easier to teachers and assistants.

Project's *Modus Operandi*

- The compiler is designed as a long pipe composed of several modules.
- The project is divided in several steps, where students have to implement one (or two) module(s).
- Code with gaps.
- Work is evaluated by a program at each delivery.
- Students defend their work every two steps in front of a teaching assistant.
- Several optional assignments are given as extra modules.
- Motivated students can choose to proceed with the implementation of the back-end of the compiler.

Project's *Modus Operandi*

- The compiler is designed as a long pipe composed of several modules.
- The project is divided in several steps, where students have to implement one (or two) module(s).
- Code with gaps.
- Work is evaluated by a program at each delivery.
- Students defend their work every two steps in front of a teaching assistant.
- Several optional assignments are given as extra modules.
- Motivated students can choose to proceed with the implementation of the back-end of the compiler.

Project's *Modus Operandi*

- The compiler is designed as a long pipe composed of several modules.
- The project is divided in several steps, where students have to implement one (or two) module(s).
- Code with gaps.
- Work is evaluated by a program at each delivery.
- Students defend their work every two steps in front of a teaching assistant.
- Several optional assignments are given as extra modules.
- Motivated students can choose to proceed with the implementation of the back-end of the compiler.

Project's *Modus Operandi*

- The compiler is designed as a long pipe composed of several modules.
- The project is divided in several steps, where students have to implement one (or two) module(s).
- Code with gaps.
- Work is evaluated by a program at each delivery.
- Students defend their work every two steps in front of a teaching assistant.
- Several optional assignments are given as extra modules.
- Motivated students can choose to proceed with the implementation of the back-end of the compiler.

Figures

- 14 years of existence.
- 250 students per year (on average).
- Project done in groups of 4 (formerly 6) students.
- 7 mandatory steps (compiler front-end).
- 4 optional steps (compiler back-end).
- Reference compiler: 25KLOC.
- Students are expected to write about 5500 lines (or about 7000 lines, with the optional assignments).
- 250+ pages of documentation (reference manual [Demaille and Levillain, 2007b] and project assignments [Demaille and Levillain, 2007a]).
- 3 papers in international conferences [Demaille, 2005, Demaille et al., 2008, Demaille et al., 2009].

History I

2000 Beginning of the Tiger Project: a front-end, a single teacher, no assistant.

2001 Have students learn and use the Autotools for project maintenance.

2002 Teaching Assistants involved in the project.
Interpreter for the Intermediate Representation (IR) language (HAVM).

2003 Addition of a MIPS back-end, partly from the work of motivated students.
Interpreter for the MIPS language (Nolimips).
The structures of the Abstract Syntax Tree (AST) and a visitor are generated from a description file.

History II

2005 A second teacher in the project maintenance and supervision.
First uses of some Boost libraries (Boost.Variant, Boost Graph Library (BGL), Smart Pointers).
First use of concrete-syntax program transformations (code generation) using TWEASTs

2007 Tiger becomes an Object-Oriented Language (OOL).

2009 C++ objects on the parser stack.

2011 Extension of Bison's grammar to handle named parameters.

2012 Conversion of `tc` to C++11.

2015 Even more C++11/C++14, aiming at C++17.
OO is now optional.
Much simpler/faster build system.

Practical information

Entry Points

- Home page of the project:
  `http://tiger.lrde.epita.fr`
- Resources for the project:
  `http://www.lrde.epita.fr/~tiger`
- `news.epita.fr` newsgroup:
  `epita.cours.compile`

Reference Documents

- Tiger Compiler Project Assignments:
  `www.lrde.epita.fr/~tiger/assignments.html`
  `www.lrde.epita.fr/~tiger/assignments.split`
  `www.lrde.epita.fr/~tiger/assignments.pdf`
  `www.lrde.epita.fr/~tiger/assignments.info`
  `www.lrde.epita.fr/~tiger/assignments.txt`
- Tiger Compiler Reference Manual (TCRM):
  `www.lrde.epita.fr/~tiger/tiger.html`
  `www.lrde.epita.fr/~tiger/tiger.split`
  `www.lrde.epita.fr/~tiger/tiger.pdf`
  `www.lrde.epita.fr/~tiger/tiger.info`
  `www.lrde.epita.fr/~tiger/tiger.txt`

Lectures notes

Slides (like these ones):
`http://www.lrde.epita.fr/~tiger/lecture-notes/slides/`

Handouts (if you need to print lectures, please use these and save a tree):

- **1-up** `http://www.lrde.epita.fr/~tiger/lecture-notes/handouts/`
- **4-up** `http://www.lrde.epita.fr/~tiger/lecture-notes/handouts-4/`

Subdirectories:

- `ccmp/` CCMP lecture notes
  - `tc/` TC-*n* presentations, given by the Assistants
- `tyla/` TYLA lecture notes

Other Useful Resources

- Doxygen Documentation:
  `http://www.lrde.epita.fr/~tiger/tc-doc/`
- Past exams: `http://www.lrde.epita.fr/~tiger/exams/`
- Some artwork from previous classes:
  `http://www.lrde.epita.fr/~tiger/humor/`

Rules of the Game

Nul n'est censé ignorer la loi.

`www.lrde.epita.fr/~tiger/assignments.html`

1. **No copy between groups.**
2. Tests are part of the project
   (test cases and frameworks should not be exchanged).
3. Fixing mistakes earlier is better (and less expensive).
4. Work between groups is encouraged!
   As long as they don't cheat.

Nul n'est censé ignorer la loi.

`www.lrde.epita.fr/~tiger/assignments.html`

1. **No copy between groups.**
2. **Tests are part of the project**
   (test cases and frameworks should not be exchanged).
3. Fixing mistakes earlier is better (and less expensive).
4. Work between groups is encouraged!
   As long as they don't cheat.

Nul n'est censé ignorer la loi.

`www.lrde.epita.fr/~tiger/assignments.html`

1. **No copy between groups.**
2. **Tests are part of the project**
   (test cases and frameworks should not be exchanged).
3. Fixing mistakes earlier is better (and less expensive).
4. Work between groups is encouraged!
   As long as they don't cheat.

Nul n'est censé ignorer la loi.

`www.lrde.epita.fr/~tiger/assignments.html`

1. **No copy between groups.**
2. **Tests are part of the project**
   (test cases and frameworks should not be exchanged).
3. Fixing mistakes earlier is better (and less expensive).
4. Work between groups is encouraged!
   As long as they don't cheat.

The Tiger Compiler

The C Compilation Model

Which (coarse-grained) steps can we find in gcc?

The C Compilation Model

Which (coarse-grained) steps can we find in gcc?

- cpp (preprocessor)

The C Compilation Model

Which (coarse-grained) steps can we find in gcc?

- cpp (preprocessor)
- cc1 (actual C compiler)

The C Compilation Model

Which (coarse-grained) steps can we find in `gcc`?

- `cpp` (preprocessor)
- `cc1` (actual C compiler)
- `as` (assembler)

The C Compilation Model

Which (coarse-grained) steps can we find in `gcc`?

- `cpp` (preprocessor)
- `cc1` (actual C compiler)
- `as` (assembler)
- `ld` (linker)

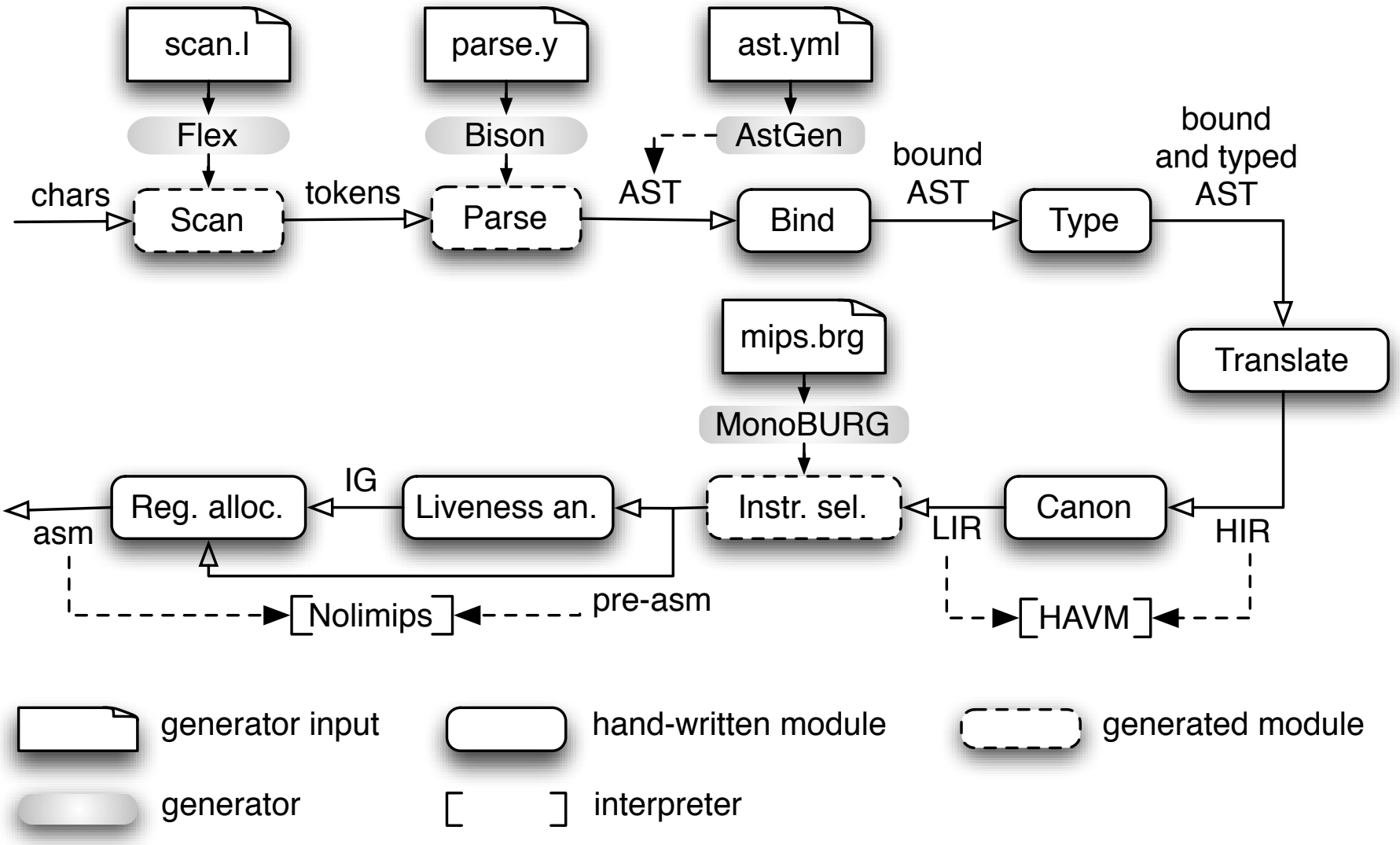
The C Compilation Model

Which (coarse-grained) steps can we find in `gcc`?

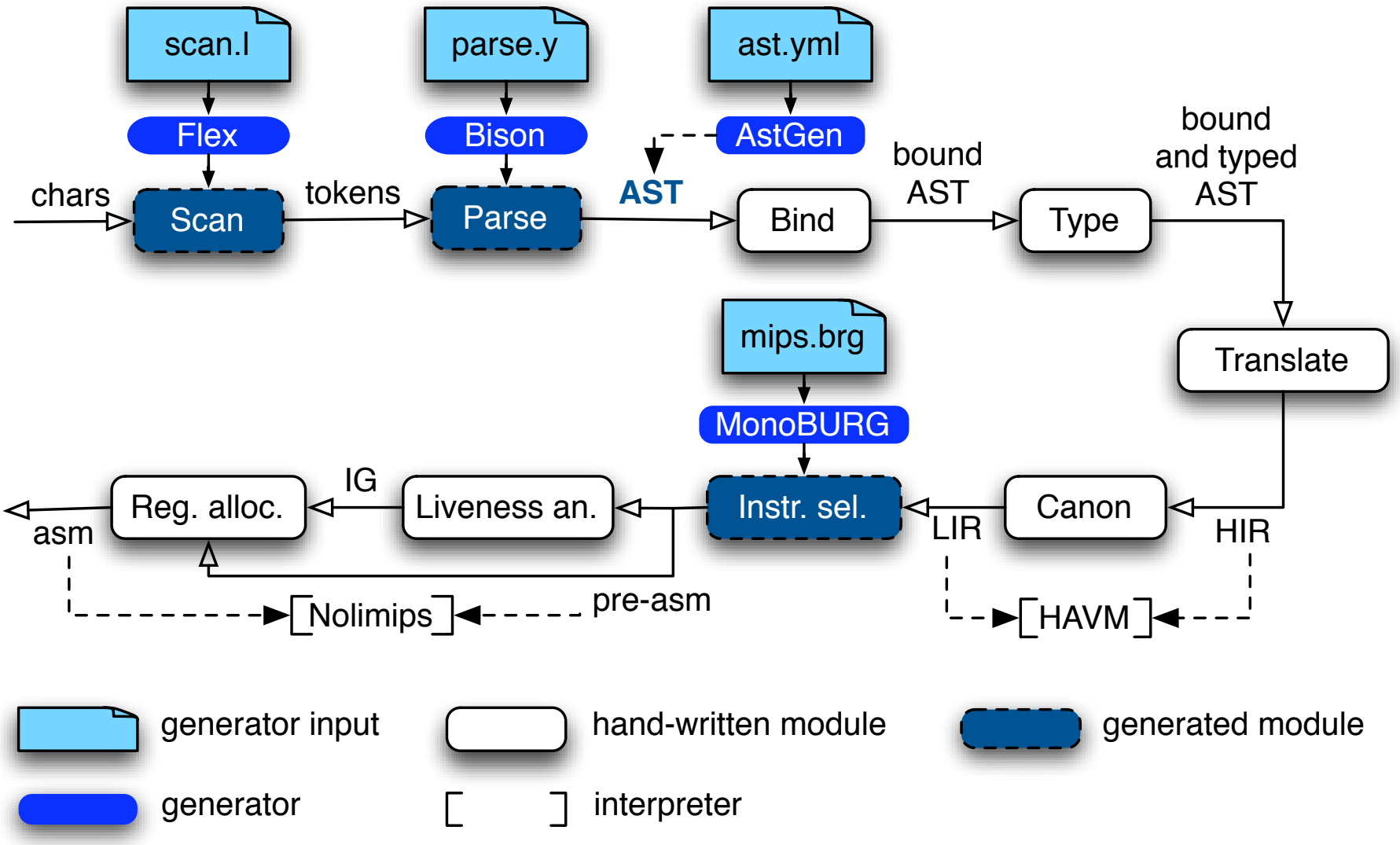
- `cpp` (preprocessor)
- `cc1` (actual C compiler)
- `as` (assembler)
- `ld` (linker)

$\rightarrow$ A *pipe*.

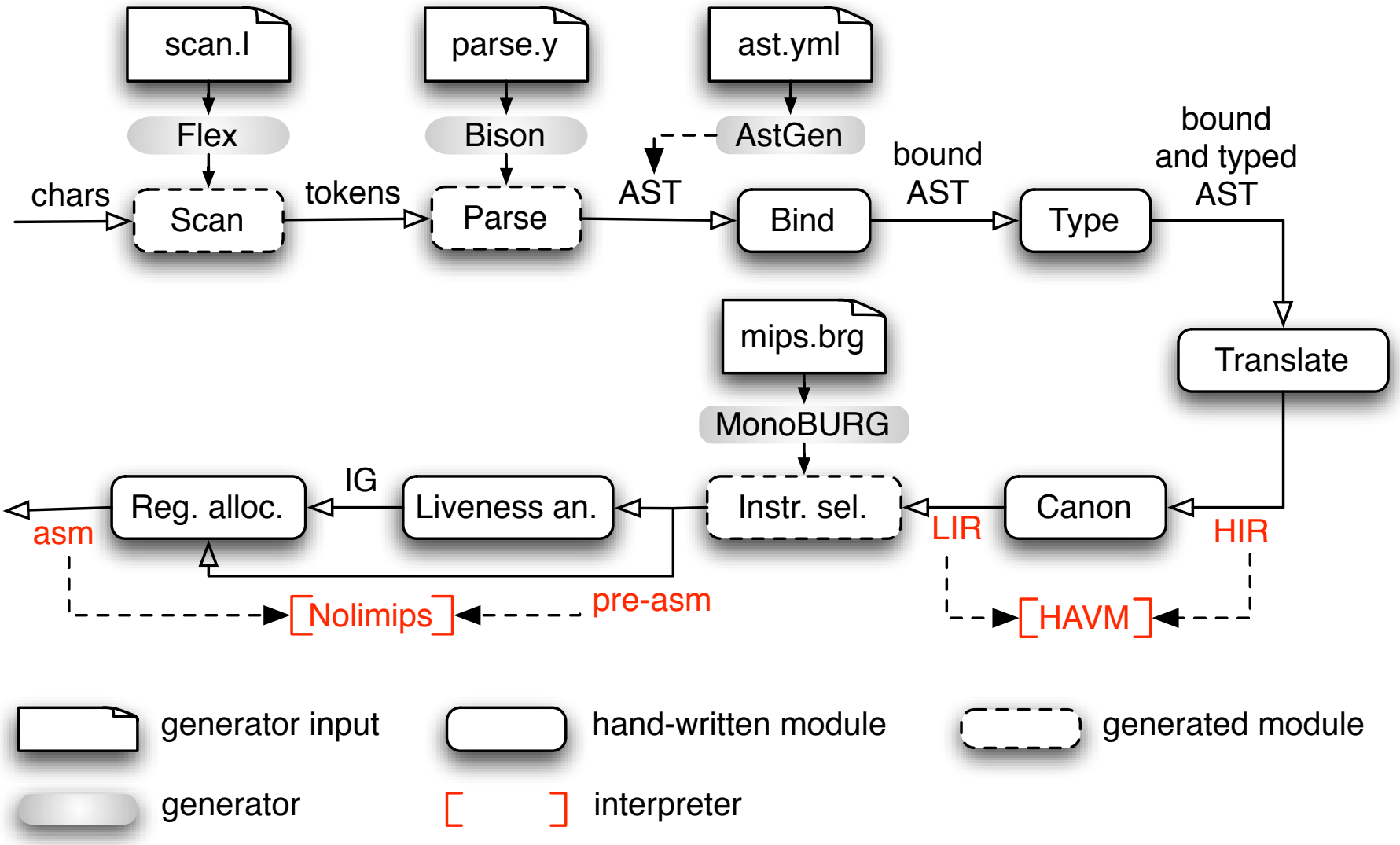
tc: A Compiler as A Long Pipe



tc: A Compiler as A Long Pipe



tc: A Compiler as A Long Pipe



Other Compiling Strategies

- Intermediate language-based strategy: SmartEiffel, GHC
- Bytecode strategy: Java bytecode (JVM), CIL (.NET)
- Hybrid approaches: GCJ (Java bytecode or native code)
- Retargetable optimizing back ends: MLRISC, VPO (Very Portable Optimizer), and somehow C-- (Quick C--).
- Modular systems: LLVM (compiler as a library, centered on a typed IR). Contains the LLVM core libraries, Clang, LLDB, etc. Also:
  - VMKit: a substrate for virtual machines (JVM, etc.).
  - Emscripten: an LLVM-to-JavaScript compiler. Enables C/C++ to JS compilation.

Intermediate Representations (IR) are fundamental.

The Tiger Language

Two flavors

Appel's Defined in *Modern Compiler Implementation* books (see the Appendix).

Ours Defined in the Tiger Compiler Reference Manual (TCRM).

- Features many extensions: `import` keyword, overloading, OOP, ...
- Implemented by LRDE's reference compiler.
- This is the target language for your project.

Two flavors

Appel's Defined in *Modern Compiler Implementation* books (see the Appendix).

Ours Defined in the Tiger Compiler Reference Manual (TCRM).

- Features many extensions: `import` keyword, overloading, OOP, ...
- Implemented by LRDE's reference compiler.
- This is the target language for your project.

Two flavors

Appel's Defined in *Modern Compiler Implementation* books (see the Appendix).

Ours Defined in the Tiger Compiler Reference Manual (TCRM).

- Features many extensions: `import` keyword, overloading, OOP, ...
- Implemented by LRDE's reference compiler.
- This is the target language for your project.

Two flavors

Appel's Defined in *Modern Compiler Implementation* books (see the Appendix).

Ours Defined in the Tiger Compiler Reference Manual (TCRM).

- Features many extensions: `import` keyword, overloading, OOP, ...
- Implemented by LRDE's reference compiler.
- This is the target language for your project.

Two flavors

Appel's Defined in *Modern Compiler Implementation* books (see the Appendix).

Ours Defined in the Tiger Compiler Reference Manual (TCRM).

- Features many extensions: `import` keyword, overloading, OOP, ...
- Implemented by LRDE's reference compiler.
- This is the target language for your project.

Tiger

- Toy language (not industry-proof)...

  ...but still effective.
- Imperative language, descendant of Algol.
- Functional flavour.
  - `a := if 1 then 2 else 3`
  - `function incr(x : int) : int = x + 1`
- Simple and sound grammar.
- Well defined semantics.

Tiger

- Toy language (not industry-proof)...

  ...but still effective.
- Imperative language, descendant of Algol.
- Functional flavour.
  - `a := if 1 then 2 else 3`
  - `function incr(x : int) : int = x + 1`
- Simple and sound grammar.
- Well defined semantics.

Tiger

- Toy language (not industry-proof)...

  ...but still effective.
- Imperative language, descendant of Algol.
- Functional flavour.
  - `a := if 1 then 2 else 3`
  - `function incr(x : int) : int = x + 1`
- Simple and sound grammar.
- Well defined semantics.

Tiger

- Toy language (not industry-proof)...

  ...but still effective.
- Imperative language, descendant of Algol.
- Functional flavour.
  - `a := if 1 then 2 else 3`
  - `function incr(x : int) : int = x + 1`
- Simple and sound grammar.
- Well defined semantics.

Tiger

- Toy language (not industry-proof)...

  ...but still effective.
- Imperative language, descendant of Algol.
- Functional flavour.
  - `a := if 1 then 2 else 3`
  - `function incr(x : int) : int = x + 1`
- Simple and sound grammar.
- Well defined semantics.

Tiger

- Toy language (not industry-proof)...

  ...but still effective.
- Imperative language, descendant of Algol.
- Functional flavour.
  - `a := if 1 then 2 else 3`
  - `function incr(x : int) : int = x + 1`
- Simple and sound grammar.
- Well defined semantics.

Tiger

- Toy language (not industry-proof)...

  ...but still effective.
- Imperative language, descendant of Algol.
- Functional flavour.
  - `a := if 1 then 2 else 3`
  - `function incr(x : int) : int = x + 1`
- Simple and sound grammar.
- Well defined semantics.

Tiger

- Toy language (not industry-proof)...

  ...but still effective.
- Imperative language, descendant of Algol.
- Functional flavour.
  - `a := if 1 then 2 else 3`
  - `function incr(x : int) : int = x + 1`
- Simple and sound grammar.
- Well defined semantics.

Your first Tiger Program

```
print("Hello World!\n")
```

Your second Tiger program

```
let
  function hello(name : string) =
    print(concat("Hello ", name))
in
  hello("You");
  print("\n")
end
```

The classic Factorial function

```
let
  /* Compute n!  */
  function fact(n : int) : int =
    if n = 0
      then 1
      else n * fact(n - 1)
in
  print_int(fact(10));
  print("\n")
end
```

Bibliography I


Appel, A. W. (1998).
Modern Compiler Implementation in C, Java, ML.
Cambridge University Press.


Debray, S. (2002).
Making compiler design relevant for students who will (most likely) never design a compiler.
In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 341–345. ACM Press.


Demaille, A. (2005).
Making compiler construction projects relevant to core curriculums.
In *Proceedings of the Tenth Annual Conference on Innovation and Technology in Computer Science Education (ITICSE'05)*, pages 266–270, Universidade Nova de Lisboa, Monte da Pacarita, Portugal.

Bibliography II


Demaille, A. and Levillain, R. (2007a).
The Tiger Compiler Project Assignment.
EPITA Research and Development Laboratory (LRDE), 14-16 rue Voltaire, FR-94270 Le Kremlin-Bicêtre, France.
http://www.lrde.epita.fr/~akim/ccmp/assignments.pdf.


Demaille, A. and Levillain, R. (2007b).
The Tiger Compiler Reference Manual.
EPITA Research and Development Laboratory (LRDE), 14-16 rue Voltaire, FR-94270 Le Kremlin-Bicêtre, France.
http://www.lrde.epita.fr/~akim/ccmp/tiger.pdf.

Bibliography III


Demaille, A., Levillain, R., and Perrot, B. (2008).
A set of tools to teach compiler construction.
In *Proceedings of the Thirteenth Annual Conference on Innovation and Technology in Computer Science Education (ITICSE'08)*, pages 68–72, Universidad Politécnica de Madrid, Spain.


Demaille, A., Levillain, R., and Sigoure, B. (2009).
TWEAST: A simple and effective technique to implement concrete-syntax AST rewriting using partial parsing.
In *Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC'09)*, pages 1924–1929, Waikiki Beach, Honolulu, Hawaii, USA.