Compilation Project

Philippe Bidinger

philippe.bidinger@univ-grenoble-alpes.fr

Université Grenoble Alpes

December 11th, 2017

Contents

Contents

Project setup

- Home page of the project `www-verimag.imag.fr/~bidinger/compilation`.
- All announcements will be made on this page. Check everyday.
- Download `archive.tar.gz` from and untar it (`tar zxvf archive.tar.gz`).
- All the instructions and documentation for the project is accessible from `doc/index.html`.
- Depending on the language you will use for the project, you will start with the code given in java, ocaml or c.
- See README for the organization of the archive.

Git setup

- Each student creates an account on github
  - Good habit to use it for all your programming projects
  - Showcase of your work
  - Use something else if you don't like github
- One student creates a project and upload the initial files
- Add all other students as collaborators
- Add me as collaborator On github user "phlalx"
- Project has to be set to private eventually
- Go to education.github.com to request free account upgrade
  - If problem, use `bitbucket.com`
- Everbody in the group clones the project
- Set up your git locally (see provided doc)
  - gitconfig
  - upload public key

Important dates

- Mon. 11 - 13h30 - 16h45 Presentation of the project / git / ocaml refresher
- Fri. 15/12, 22/12, 12/01 - 19h weekly report (mid-project submission on 22/12)
- Wed. 17/01 - 12h00 Final project submission
- Friday. 19 - 09h00 - 17h00 Project defense (maybe Thursday)
- Most days, I'll be answering questions in F204, either morning or afternoon.
- We may schedule extra lectures if needed
- All missed deadlines will be penalized
- Project will be retrieved from git repo master branch on time

Supervision

- You should be *autonomous*,
  - E.g. set up a github/bitbucket project
  - follow an ocaml tutorial,
  - find an ARM instruction,
  - write a bash script or compile a java program ...
- However, ask for help if:
  - you are stuck (don't wait for the last day!)
  - need clarification on provided material
  - want to make sure you're on the right path
- You're expected to work 7-8 hours a day for 4 weeks.
- Use internet with good judgment
  - Stack overflow for all git / programming questions
  - But MinCaml, ASML only used in this project
  - Careful with compilation resources

Communication

- Email
    - Clear and precise questions
    - Try to think first
    - answer by email, or during presence hour, or on the FAQ
- Updates, corrections
    - Email to leaders
    - Update on webpage

Group organization

- Leader
  - Global view of the project
  - Communication with supervisor
  - Check deadlines and deliverables
- Difficulties
  - Distribution of tasks (efficiency vs interest, dynamic)
  - Heterogeneity (skills, motivation)
- Daily meeting
- Don't stay in a bubble

Weekly Organization

Team organization

Highly recommended: every day, ≈10 minutes "stand-up meeting" at fixed hour.

For each task:

- - new task: description of task
  - existing task: status of task (work done, work remaining...)
- people assigned on task
- expected time of completion

Assign the "scribe" role: takes log of meeting (must not change during week).

Weekly Organization

Team organization

Highly recommended: every day, ≈10 minutes “stand-up meeting” at fixed hour.

For each task:

- - new task: description of task
  - existing task: status of task (work done, work remaining. . .)
- people assigned on task
- expected time of completion

Assign the “scribe” role: takes log of meeting (must not change during week).

Weekly report is the list of all five logs of the week.

Group Organization

Send me an email TODAY or TOMORROW with:

- team members (4-5 people)
- team name
- team “leader” (recommended)
- programming language. OCaml, Java or something else if you know what you are doing.
- Git repository that I can clone

- I’ll confirm every email
- Without this, you will not be part of the project

Grade

Grade will take into account:

- Respect of requirements
- Code:
  - features
  - writing quality (adhere to coding conventions, write well-documented and modular programs)
  - robustness
  - test suite
- Documentation:
  - "User documentation": what we need to use your compiler
  - "Developer documentation": what a new developer would need, including documentation of test suite
  - Recommended: javadoc (or ocamldoc) + markdown
- Demo:
  - 20 min. presentation (technical & commercial).
  - 10 min. questions
- Team management (weekly reports,...)
- Personal modifier

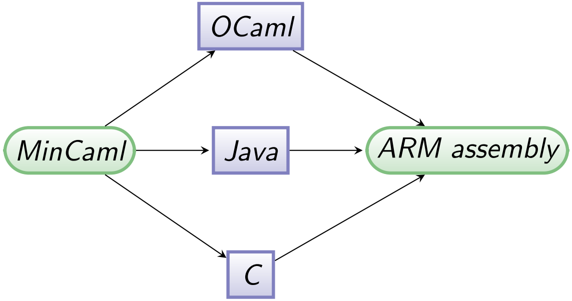
Cheating

- You must write all the code you hand in, except for code that was provided. You are not allowed to look at anyone else's solution or use code written by someone else.
- but you may discuss your assignments with other students. This is encouraged.
- Plagiarism will be detected (detection tools).
- we will follow you during the project.

Contents

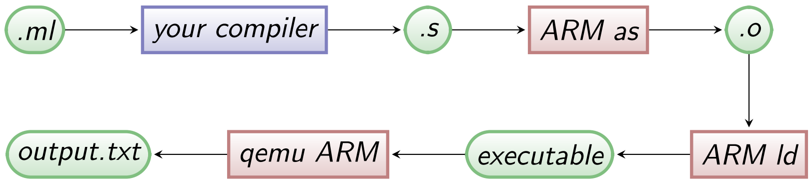
The Compiler

Skeletons given in OCaml, Java, C, with parser and minimal tests.


$$\begin{tikzcd}
 & \text{OCaml} \arrow[dr] & \\
\text{MinCaml} \arrow[ur] \arrow[r] \arrow[dr] & \text{Java} \arrow[r] & \text{ARM assembly} \\
 & \text{C} \arrow[ur] &
\end{tikzcd}$$

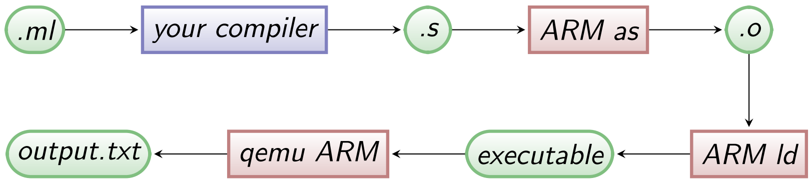
Toolchain

Compiler Toolchain:

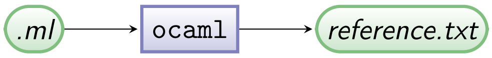


Toolchain

Compiler Toolchain:



Comparison:



Warning: `ocaml` interprets the code (no compilation).

Toolchain Example

```
$ mincamlc gcd.ml -o gcd.s
$ arm-none-eabi-as -o gcd.o gcd.s libmincaml.S
$ arm-none-eabi-ld -o gcd gcd.o
$ qemu-arm ./gcd
```

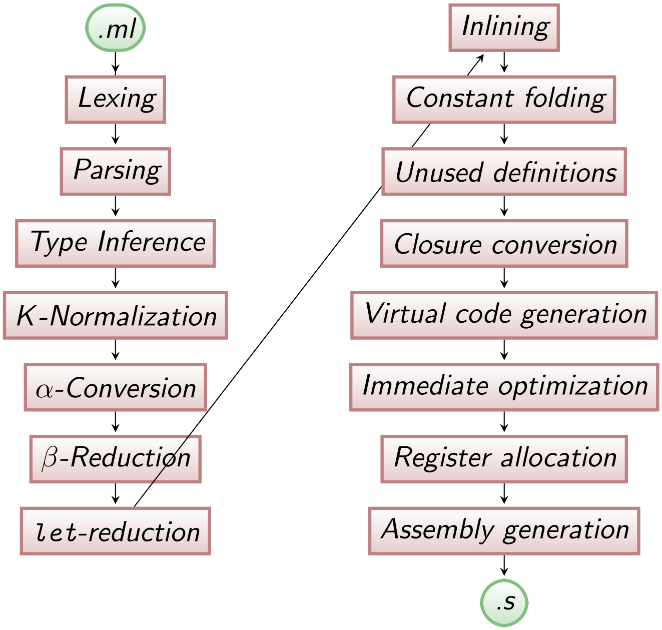
See in the archive for other examples.

GCD Example

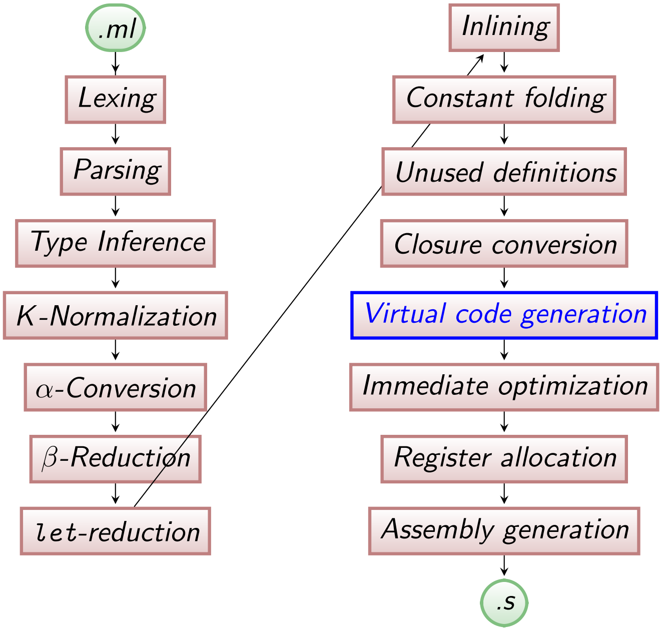
```
let rec gcd m n =
  if m = 0 then n else
  if m <= n then gcd m (n - m) else
  gcd n (m - n)
```

```
        .text
gcd.7:
        cmp   r0, #0
        bne   be_else.17
        mov   r0, r1
        bx    lr
be_else.17:
        cmp   r0, r1
        bgt   ble_else.18
        sub   r1, r1, r0
        b     gcd.7
        nop
ble_else.18:
        sub   r0, r0, r1
        mov   r14, r1
        mov   r1, r0
        mov   r0, r14
        b     gcd.7
        nop
```

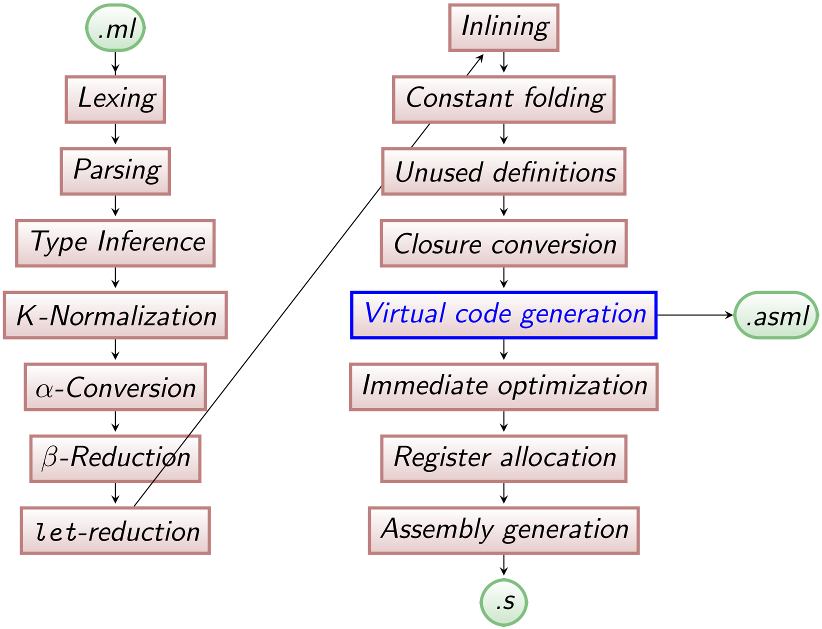
Compiler Passes



Compiler Passes



Compiler Passes



Tasks and Incremental Development

Decide what you want to do!

- Front-end (35%)
- Type inference and type checking (15%)
- Back-end (35%)
- Testing (15%)

Each task can be subdivided furthermore.

- Don't work one pass a at a time, but one feature at a time
- Integrate ASAP

Incremental development

- simple arithmetic expression
- call to external functions
- if-then-else
- functions (let rec)
- tuples and arrays
- closures
- floats

The ASML intermediate representation

ASML is at the interface between front-end and back-end.
It is a *Virtual Machine Code*.

Features

- values stored in temporaries ("registers")
- functions represented by labels
- no nested definitions
- memory accesses are explicit
- if-then-else constructs
- operators for memory allocations

The ASML intermediate representation

ASML is at the interface between front-end and back-end.
It is a *Virtual Machine Code*.

ASML example

```
let _ =
  let size = 2 in
  let init = 0 in
  let arr = call _min_caml_create_array size init in
  let i0 = 0 in
  let v0 = 1 in
  let tu = mem(arr + i0) <- v0 in
  let i1 = 1 in
  let v1 = mem(arr + i0) in
  mem(arr + i1) <- v1
```

The ASML intermediate representation

- We provide an interpeter for ASML. You can test the front-end independently of the back-end!
  - Compare the behavior of the ASML program with the behavior of the source file (using ocaml)
  - But you need to output ASML (easy)
- you can test the back-end independently of the front-end too
  - But you need to be able to parse ASML (more difficult!). Maybe a simpler representation (JSON?)
- In any case, you will have to write a datatype that encode ASML programs.

The Command Line

We will use our test suite on your compilers.

Return value

0	on success
1	on error, plus error message on stderr

Required command-line options

-o <file>	: output file
-h	: display help
-v	: display version
-t	: only type checking
-p	: parse only
-asml	: print ASML

The Command Line

We will use our test suite on your compilers.

Return value

0	on success
1	on error, plus error message on stderr

Required command-line options

-o <file>	: output file
-h	: display help
-v	: display version
-t	: only type checking
-p	: parse only
-asml	: print ASML
-my-opt	: you can add personal options (optimizations, etc.)

Contents