

LeTSeE

The LEgal Transformation SpacE Explorer
Edition 0.1, for LeTSeE 0.1.0
February 16th 2008

Louis-Noël Pouchet

This manual is dedicated to LeTSeE version 0.1.0, a platform for computing and manipulating legal affine schedules of a program.

Copyright © 2006-2008 Louis-Noël Pouchet.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 published by the Free Software Foundation. To receive a copy of the GNU Free Documentation License, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

Table of Contents

1	Introduction	1
1.1	A Complete Toolchain	1
1.1.1	LeTSeE as a Program Optimization Platform	1
1.1.2	LeTSeE as a Legal Affine Scheduling Library	1
1.2	Installation of LeTSeE	1
1.3	Test Run of LeTSeE	2
1.4	If Anything Goes Wrong...	2
2	Polyhedral Representation of Programs	3
2.1	Input of LeTSeE	3
3	Running LeTSeE to optimize code	5
3.1	Introduction	5
3.2	Guided tour	5
3.2.1	Check available tests	5
3.2.2	Check available options	5
3.2.3	A first run of LeTSeE	6
3.2.4	A first exhaustive exploration through the tester script	8
3.2.5	Inspecting the results	11
3.2.6	Cleaning the transformations	12
3.2.7	Cleaning the results	12
3.2.8	Heuristic exploration of the space	12
3.2.9	Dealing with larger programs	13
3.3	Adding new test files	13
4	Programming with the LeTSeE library	15
4.1	A Tour of the Functionality	15
4.2	A Tour of the Structures	15
4.3	A Tour of the Files	15
4.4	The Workflow	16
5	Theoretical Corner	17
6	References	19

1 Introduction

LeTSeE is a platform dedicated to computing and exploring the legal affine scheduling space of a statically controlled program. It is programmed as a library, offering services such as:

- a tunable algorithm for legal transformation space construction,
- various heuristics to traverse legal spaces,
- many convenience functions (graph manipulation, transformation generation, etc.)

1.1 A Complete Toolchain

The LeTSeE software is a part of a larger toolchain (often also referred as LeTSeE by the author!) containing a set of free softwares for program transformation and optimization in the polyhedral model. It comprises:

- **FM**, the Fourier-Motzkin Library, for manipulating \mathbb{Q} -polyhedra
- **PipLib**, a parametric integer programming solver
- **Polylib**, a library for parametric integer polyhedra computation
- **CLooG**, a code generator
- **Candl**, a dependence analyzer

To enjoy full features of LeTSeE, it is *strongly recommended* to install all the abovementioned softwares before installing LeTSeE.

1.1.1 LeTSeE as a Program Optimization Platform

The LeTSeE software (especially through the offered binaries) can be used to fully optimize a program, the output being an optimized C code corresponding to the polyhedral representation given as an input (plus some extra info, roughly the C code corresponding to the input program separated in several blocks).

1.1.2 LeTSeE as a Legal Affine Scheduling Library

The LeTSeE core is a library offering many facilities to compute legal affine schedules of a program, provided a simple list of dependence polyhedra.

1.2 Installation of LeTSeE

The installation of LeTSeE follows the classical scheme. Please specify the path of the 2 mandatory dependencies of LeTSeE (FM and candl) to the configure script¹. Please note that the FM library must have been compiled with the PIPLib support (it is turned on by default in the LeTSeE Toolkit package)

```
$> tar xzf letsee-0.1.0.tar.gz
$> cd letsee-0.1.0
$> ./configure --prefix=/path/to/be/installed --with-fm=/path/to/fm/install
-w-with-candl=/path/to/candl/install
$> make
```

¹ The configure script will automatically detect the location of dependencies if `-prefix=path/to/dir` is specified and all dependencies were installed in `path/to/dir`

1.3 Test Run of LeTSeE

The `make check` command can be used to make a test run of LeTSeE. It will launch the space computation of the *matmult* code fragment, generate the corresponding 912 transformations, compile and run them on the target machine (provided it is a Linux embedding GCC).

```
$> make check
```

It uses the `src/scripts/tester2.sh` script, which will be detailed later.

1.4 If Anything Goes Wrong...

There are several installation errors which may happen, beside a bug in the software itself. Please check the following before contacting the author:

1. All dependencies are correctly installed, and were checked to work.
2. All dynamic libraries are accessible for the LeTSeE binary. Typically, you need to define the `LD_LIBRARY_PATH` variable with the paths to *all* libraries on which LeTSeE depends on. You may also need to specify in the `PATH` environment variable the path to access the `CLooG` and `Candl` binaries.
3. The base tester scripts of LeTSeE rely on the fact that you're using UNIX-like computer, offering a few typical instructions (`set_scheduler`, and assembly access to the Time Stamp Counter of the processor). If you don't meet these requirements, you cannot use directly `make check`. Don't worry, LeTSeE will work, you will just have to specify some extra options, as detailed later.

2 Polyhedral Representation of Programs

2.1 Input of LeTSeE

Explain the `.candl`, and the `.program`

This chapter will be completed for the next release. In the meantime, please refer to our papers, and to the examples in `tests/battery`.

3 Running LeTSeE to optimize code

3.1 Introduction

The LeTSeE binary offers access to most of the functions of the library for program optimization in the polyhedral model. In addition, several scripts and test files are available to get a full tour of the software's possibilities.

3.2 Guided tour

3.2.1 Check available tests

(only provide .candl files to LeTSeE)

```
$> ls -R tests/battery

battery/  rr/  utdsp/

./battery:
bastoul1.candl      edge_detect.program  h264.candl          matvect.program
bastoul1.program   fft_256_2.candl     h264.program        mvt.candl
croust.candl       fft_256_2.program  matmult.candl       mvt.program
croust.program     gosser.candl        matmult.program     template.candl
edge_detect.candl  gosser.program      matvect.candl       template.program

./rr:
rr.candl  rr.program

./utdsp:
compress/  edge_detect/  fir/  lpc/  mult/

./utdsp/compress:
compress-dct-opt.candl  compress-dct-opt.program

./utdsp/edge_detect:
edge_detect-convolve2d.candl  edge_detect-convolve2d.program

./utdsp/fir:
fir.candl  fir.program

./utdsp/lpc:
lpc-LPC_analysis.candl  lpc-LPC_analysis.program

./utdsp/mult:
mult.candl  mult-opt.candl  mult-opt.program  mult.program
```

3.2.2 Check available options

```
$> src/letsee -h
```

Options for letsee are:

```

-a      --ilb      Lower iterator coefficient bound [-1]
-b      --iub      Upper iterator coefficient bound [ 1]
-c      --plb      Lower parameter coefficient bound [-1]
-d      --pub      Upper parameter coefficient bound [ 1]
-e      --clb      Lower constant coefficient bound [-1]
-f      --cub      Upper constant coefficient bound [ 1]
-o      --output   Output file [stdout]
-n      --no-files Don't create CLoog files for drawn schedules
-z      --transfo-dir Directory for CLoog files [transformations]
-t      --type     Legal space computation algorithm ([ospace], multi)
-w      --walk     Exploration heuristic ([exhaust], h1, r1, r1m, m1)
-s      --spacenorm Normalize legal space
-x      --execute  Compilation line
-r      --rate     Threshold for heuristic filter (in %) [5]
-q      --rtries   Number of tries for heuristic [20]
-l      --le-fur   Use Le Fur redundancy elimination
-L      --load-space Load space from file
-y      --backtrack Enable backtrack mode
-p      --dep-order Enable dependence ordering w.r.t. traffic
-S      --scheme-m1 Specify scheme (imply -w m1): "i+p+c,i+p,i,0"
-v      --verbose  Verbose output
-h      --help     Print this help

```

3.2.3 A first run of LeTSeE

```
$> src/letsee -s -v -t multi tests/battery/matmult.candl
```

This command generates the set of legal schedules, for the matmult example, within the bounds [-1,1] (default value) for all schedule coefficients. Note that this command only generate the schedules as CLoog files, in the transformations/ directory. Note also we provided the verbose option (-v), as well as the multidimensional schedules support (-t multi), which is now recommended by default and mandatory for using heuristics. Also, the -s option is specified, in order to normalize and simplify the outputted space. This option is required most of the time if you want LeTSeE to traverse the generated polytopes efficiently.

We now detail the output, step by step.

This first section shows the initialization of LeTSeE, and the result of the call to Candl: the dependence graph.

```

$> src/letsee -s -v -t multi tests/battery/matmult.candl
. Calling candl
. Building legal transformation space
.. Starting space computation
... Build and solve local systems
digraph G {
# Data Dependence Graph
# Generated by Candl baa 12 bits
S0 -> S1 [label=" RAW depth 2, ref 0->0 "];

```

```

    S0 -> S1 [label=" WAW depth 2, ref 0->0 "];
    S1 -> S1 [label=" RAW depth 3, ref 0->0 "];
    S1 -> S1 [label=" WAW depth 3, ref 0->0 "];
    S1 -> S1 [label=" WAR depth 3, ref 0->0 "];
# Number of edges = 5
}

```

This section shows the core of the method used for the computation of the legal space in the multidimensional case. It orders the dependences with respect to a main criterion: pairwise strong unsatisfiability (that is, the impossibility to strongly solve 2 given dependences at the same schedule depth). The result is a colored conflict graph where each node is a dependence.

```

.... Compute internal dependence graph
nb_scc: 2
0: S0
1: S1
.... Compute dependence pairwise conflict graph
..... Check conflicts for dependence 1
..... Check conflicts for dependence 2
..... Check conflicts for dependence 3
..... Check conflicts for dependence 4
..... Check conflicts for dependence 5
.... Conflict graph computed (0 conflicts)
digraph G {
# Graph output
# Generated by LetSee
D5 [label="D5:c=-1,s=-1"];
D4 [label="D4:c=-1,s=-1"];
D3 [label="D3:c=-1,s=-1"];
D2 [label="D2:c=-1,s=-1"];
D1 [label="D1:c=-1,s=-1"];
# Number of vertices = 5;
# Number of edges = 0;
}

```

This section shows the actual construction of the legal space. Here only one sequential dimension is needed. The initialization step creates a polytope where all 5 dependences are weakly solved. The compaction step performs an implicit equalities detection, and a Gaussian elimination to reduce the actual number of dimensions of the system. Then, for each dependence, we test if it possible to strongly solve it at that schedule dimension, while keeping at least one point in the space.

```

.... Initializing schedule dimension 1
..... 00000
.... Compaction: using 9/9 dimensions
.... Check consistency: yes
.... Solving dependence 1 (id: 5)
..... Strong dependence satisfaction: yes
.... Solving dependence 2 (id: 4)

```


Where `ilb`, `iUb` are the bounds for the iterator coefficients, `plb`, `pUb` those for parameter coefficients, and `clb`, `cUb` those for the constant coefficients. The compilation line is typically the content of the `$(CC)` plus `$(CFLAGS)` environment variables (that is, compiler executable call and optimization options), but the following additional options are required:

- `-DPARVALX=123`: means that the `X`th global parameter of the program will have value 123. This must be specified for each global parameter. `X` takes values between 1 and `nb_parameters`.
- `-Dtest_malloc`: means that you use `malloc` to allocate arrays. If unspecified, static (i.e., on the stack) arrays are used.

An important warning: *You must have `bash` installed on your system*, as the script uses special features of this shell. Use the `SH_BINARY` variable at the beginning of the `src/scripts/cloog2test/cloog2test` shell script to specify the path to `bash`, if necessary, and the shebang accordingly for this script and `src/scripts/tester2.sh`.

Also, please visit the beginning of `src/scripts/tester2.sh` to set the `LETSEE_OPTS` global variable with the appropriate options you want to provide to LeTSeE.

Finally, to enjoy full features of the tester script, you must have `gnuplot` and `ps2pdf` installed.

```
$> src/scripts/tester2.sh -1 1 -1 1 -1 1 "gcc -O2 -DPARVAL1=64 \
-Dtest_malloc" tests/battery/matmult.candl
=> Testing tests/battery/matmult
= Test parameters:
  tlb: -1 tUb: 1
  plb: -1 pUb: 1
  clb: -1 cUb: 1
  Compilation line: gcc -O2 -DPARVAL1=64 -Dtest_malloc
```

Starting transformations generation...

```
real    0m0.948s
user    0m0.085s
sys     0m0.248s
. Calling candl
. Building legal transformation space
.. Starting space computation
... Build and solve local systems
digraph G {
# Data Dependence Graph
# Generated by Candl baa 12 bits
  S0 -> S1 [label=" RAW depth 2, ref 0->0 "];
  S0 -> S1 [label=" WAW depth 2, ref 0->0 "];
  S1 -> S1 [label=" RAW depth 3, ref 0->0 "];
  S1 -> S1 [label=" WAW depth 3, ref 0->0 "];
  S1 -> S1 [label=" WAR depth 3, ref 0->0 "];
# Number of edges = 5
}
```

```

.... Compute internal dependence graph
nb_scc: 2
0: S0
1: S1
.... Compute dependence pairwise conflict graph
..... Check conflicts for dependence 1
..... Check conflicts for dependence 2
..... Check conflicts for dependence 3
..... Check conflicts for dependence 4
..... Check conflicts for dependence 5
.... Conflict graph computed (0 conflicts)
digraph G {
# Graph output
# Generated by LetSee
D5 [label="D5:c=-1,s=-1"];
D4 [label="D4:c=-1,s=-1"];
D3 [label="D3:c=-1,s=-1"];
D2 [label="D2:c=-1,s=-1"];
D1 [label="D1:c=-1,s=-1"];
# Number of vertices = 5;
# Number of edges = 0;
}
.... Initializing schedule dimension 1
..... 00000
.... Compaction: using 9/9 dimensions
.... Check consistency: yes
.... Solving dependence 1 (id: 5)
..... Strong dependence satisfaction: yes
.... Solving dependence 2 (id: 4)
..... Strong dependence satisfaction: yes
.... Solving dependence 3 (id: 3)
..... Strong dependence satisfaction: yes
.... Solving dependence 4 (id: 2)
..... Strong dependence satisfaction: yes
.... Solving dependence 5 (id: 1)
..... Strong dependence satisfaction: yes
Solved 5/5 dependences (no backtrack)
... All systems solved
.. Legal space computed
. Exploring legal transformation space
.. Legal space polytope exploration
*****
*****
*****
*****
*****
*****

```



```
$> cat transformations/transformation_xxx.c
[...]
```

look at its schedule (the 'SCATTERING' section: 1 matrix per statement, skip the k+1 first columns (k is the number of rows) to see the actual schedule used for the statement):

```
$> cat transformations/transformation_xxx.cloog
[...]
```

Re-run a single transformation, given a schedule:

argument syntax: `cloog_schedule_file "compilation line" test_file \ directory_for_binary`
output: the number of cycles

```
$> src/scripts/onetest.sh transformations/transformation_838.cloog "gcc \
-02 -DPARVAL1=128 -Dtest_malloc" tests/battery/matmult.candl \
transformations
19015496
```

Re-run the original code, for another parameters' value (must be performed before the first run of any transformation):

argument syntax: `directory_for_binary "compilation line" test_file` output: the number of cycles

```
$> src/scripts/origtest.sh transformations "gcc -02 -DPARVAL1=128 \
-Dtest_malloc" tests/battery/matmult.candl
67872812
```

3.2.6 Cleaning the transformations

```
$> make clean-transfo
```

3.2.7 Cleaning the results

```
$> make clean-results
```

3.2.8 Heuristic exploration of the space

LeTSeE provides a powerful set of functionalities to build heuristics to traverse the space. Two heuristics are currently available in the official release: the Decoupling Heuristic and a purely random one.

To use the heuristics, please respect the following:

- Provide option `-x "compilation line"` (see above for the description of what is “compilation line”)
- Provide option `-s`, to simplify the space
- Provide option `-w XX` where `XX` is one of the following:
 - `exhaust`: exhaustive traversal heuristic
 - `h1`: decoupling heuristic for one-dimensional schedules [2]
 - `m1`: decoupling heuristic for multidimensional schedules [3,4]
 - `r1`: random heuristic for one-dimensional schedules
 - `r1m`: decoupling heuristic for multidimensional schedules

Note that the decoupling heuristic for multidimensional schedules can be parametrized through the `-S "class,class,class"` option. You specify, for each dimension of the schedule, the classes which will be explored, while the non traversed ones will be automatically chosen [4]. `class` takes values in:

- `i`: traverse only distinct values for iterator coefficients (distinct combinations of interchange, skewing and reversal)
- `i+p`: traverse only distinct values for iterator plus parameters coefficients (same plus fusion and distribution)
- `i+p+c`: traverse distinct values for all coefficients (same plus shifting and peeling)
- `0`: don't traverse distinct solutions, just pick one

The reader is strongly encouraged to go through the bibliography to have a detailed explanation of the heuristic behavior.

3.2.9 Dealing with larger programs

Provide options `'-l'` to help the solver to scale, for the larger codes.

3.3 Adding new test files

A test is composed of three files:

1. a reference file for the original program, exactly named `transformations/mynewtest.c`
2. a Candl input file for the test, exactly named `tests/battery/mynewtest.candl`
3. a tool file for the test, exactly named `tests/battery/mynewtest.program`

Use other tests as reference (matmult for instance). Don't forget to always use `PAR-VALX` (where X starts at 1) to refer to the global parameters of loops. Also, the program is supposed to check an output against the original code output, so don't forget to fill the `EPILOGUE` variable of `mynewtest.program` accordingly to what you've written in `mynewtest.c`.

You can use the test called *template* as a basis.

4 Programming with the LeTSeE library

A Doxygen documentation of the API is available in the `doc/html/doc.tar.gz` archive.

4.1 A Tour of the Functionality

To be done.

4.2 A Tour of the Structures

The following structures are used inside LeTSeE:

- **CandlProgram**: The Candl representation of a program. It contains a list of statements, and for each of them its iteration domain plus the array access functions (one for the read cells, one for the written ones).
- **CandlDependence**: The Candl representation of a dependence. It contains a pointer to the source and the target statements of the dependence, plus a polytope representing the dependence domain (a subset of the Cartesian product of the iteration domains of the source and target of the dependence), represented in Piplib/Polylib format.
- **s_graph_t**: A generic graph structure, mostly used to create a graph of dependences.
- **s_fm_solution_t**: A Q-polyhedra, represented by affine constraints (typically the output of the Fourier-Motzkin algorithm). See the FM library documentation.
- **s_fm_system_t**: A matrix of rationals. See the FM library documentation.
- **s_ls_space_t**: The structure holding the legal transformation space of a program.
- **s_ls_explorer_t**: The structure holding the mandatory information to traverse the legal transformation space of a program.

4.3 A Tour of the Files

All the files for the library are in the `letsee/` directory.

- **letsee/error.c**: Convenience functions to handle internal errors in LeTSeE.
- **letsee/explorer.c**: Unit dedicated to polytope traversal, and convenience functions for the search space exploration (transformation generation and execution).
- **letsee/farkas.c**: All functions related to the construction of the legal transformation space except the algorithm itself.
- **letsee/graph.c**: Graph manipulation unit.
- **letsee/heuristic-exhaust.c**: Exhaustive traversal heuristic.
- **letsee/heuristic-h1.c**: Decoupling traversal heuristic [2].
- **letsee/heuristic-m1.c**: Decoupling traversal heuristic [3,4].
- **letsee/heuristic-random.c**: Random traversal heuristic.
- **letsee/heuristics.c**: Convenience functions for the heuristic traversal.
- **letsee/lspc.c**: Main unit for the construction of the legal transformation space for one-dimensional case (outdated).
- **letsee/options.c**: Option management unit.

- **letsee/schedspace.c:** Main unit for the construction of the legal transformation space for multidimensional case.
- **letsee/space.c:** Convenience functions for the space_t structure.
- **letsee/transformations.c:** Convenience functions for transformation generation.
- **letsee/xmalloc.c:** Safe wrapper for malloc.

4.4 The Workflow

input: A Candl program

1. Build the list of dependences (call Candl), see `src/letsee.c:main`
2. Build the legal transformation space, see `letsee/schedspace.c:ls_schedspace_build`
3. Traverse the legal transformation space, see `letsee/explorer.c:ls_explorer`

And, in more details, to build the legal space:

input: A list of dependences (in Candl format)

1. Build the dependence conflict graph: a graph where each node is a dependence, and each edge represent a conflict between the 2 dependences (that is, it is impossible to strongly solve them at the same time dimension). See `letsee/farkas.c:ls_farkas_build_conflicts`.
2. Color the conflict graph. See `letsee/graph.c:ls_graph_color`.
3. Build the legal space. See `letsee/schedspace.c:ls_schedspace_compute_solution`.
 1. Initialize the solution space. See `letsee/farkas.c:ls_farkas_initialize_solution`.
 2. Compute, for all dependences, the constraints for weak satisfaction. For each dependence, the constraint system to solve is built (see `letsee/farkas.c:ls_farkas_build_system`) then intersected in the global polytope (see `letsee/farkas.c:ls_farkas_intersect_cs`).
 3. Iteratively try to strongly solve each possible dependences.

5 Theoretical Corner

This chapter will be completed for the next release. In the meantime, please refer to our papers.

6 References

- [1] Louis-Noël Pouchet, Cédric Bastoul and Albert Cohen. LetSee: the LEgal Transformation SpacE Explorator. In *Third International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems*, pp 247–251, L’Aquila, Italia, July 2007. Extended abstract.
- [2] Louis-Noël Pouchet, Cédric Bastoul, Albert Cohen and Nicolas Vasilache. Iterative optimization in the polyhedral model: Part I, one-dimensional time. In *IEEE/ACM Fifth International Symposium on Code Generation and Optimization (CGO’07)*, pp 144–156, IEEE Computer Society Press, San Jose, California, March 2007.
- [3] Louis-Noël Pouchet, Cédric Bastoul, John Cavazos and Albert Cohen. A Note on the Performance Distribution of Affine Schedules. In *2nd Workshop on Statistical and Machine learning approaches to ARchitectures and compilaTion (SMART’08)*, Gotenborg, Sweden, January 2008.
- [4] Louis-Noël Pouchet, Cédric Bastoul, Albert Cohen and John Cavazos. Iterative optimization in the polyhedral model: Part II, multidimensional time. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI’08)*, ACM Press, Tucson, Arizona, June 2008. To appear.

