Loop distribution in GCC

Adding a new optimization pass at the GIMPLE SSA level in GCC

Georges-André SILBER
Centre de recherche en informatique
École nationale supérieure des mines de Paris
Georges-Andre.Silber@ensmp.fr

May 2006
Grenoble, France

Outline

- Getting and compiling GCC
- How to add an optimization pass in GCC
- Case study: loop distribution

Getting and compiling GCC

Getting GCC

- Main website: http://gcc.gnu.org
- Use Subversion: http://subversion.tigris.org
- svn co svn://gcc.gnu.org/svn/gcc/trunk gcc
- 'svn up' in the directory 'gcc' to get in sync

About GCC

Mission Statement
Releases
Snapshots
Mailing lists
Contributors
Steering Committee

Documentation

Installation
· Platforms
· Testing
Manual
FAQ
Wiki
Further Readings

Download

Mirror sites
Binaries

"Live" Sources

SVN read access
Rsync read access
SVN write access

Development

Development Plan
· (tentative timeline)
Contributing
...Why?
Open projects
Front ends
Back ends
Extensions

Welcome to the GCC home page!

GCC, the GNU Compiler Collection, includes front ends for C, C++, Objective-C, Fortran, Java, and Ada, as well as libraries for these languages (libstdc++, libgcj,...).

We strive to provide regular, high quality releases, which we want to work well on a variety of native and cross targets (including GNU/Linux), and encourage everyone to contribute changes and help testing GCC. Our sources are readily and freely available via SVN and weekly snapshots.

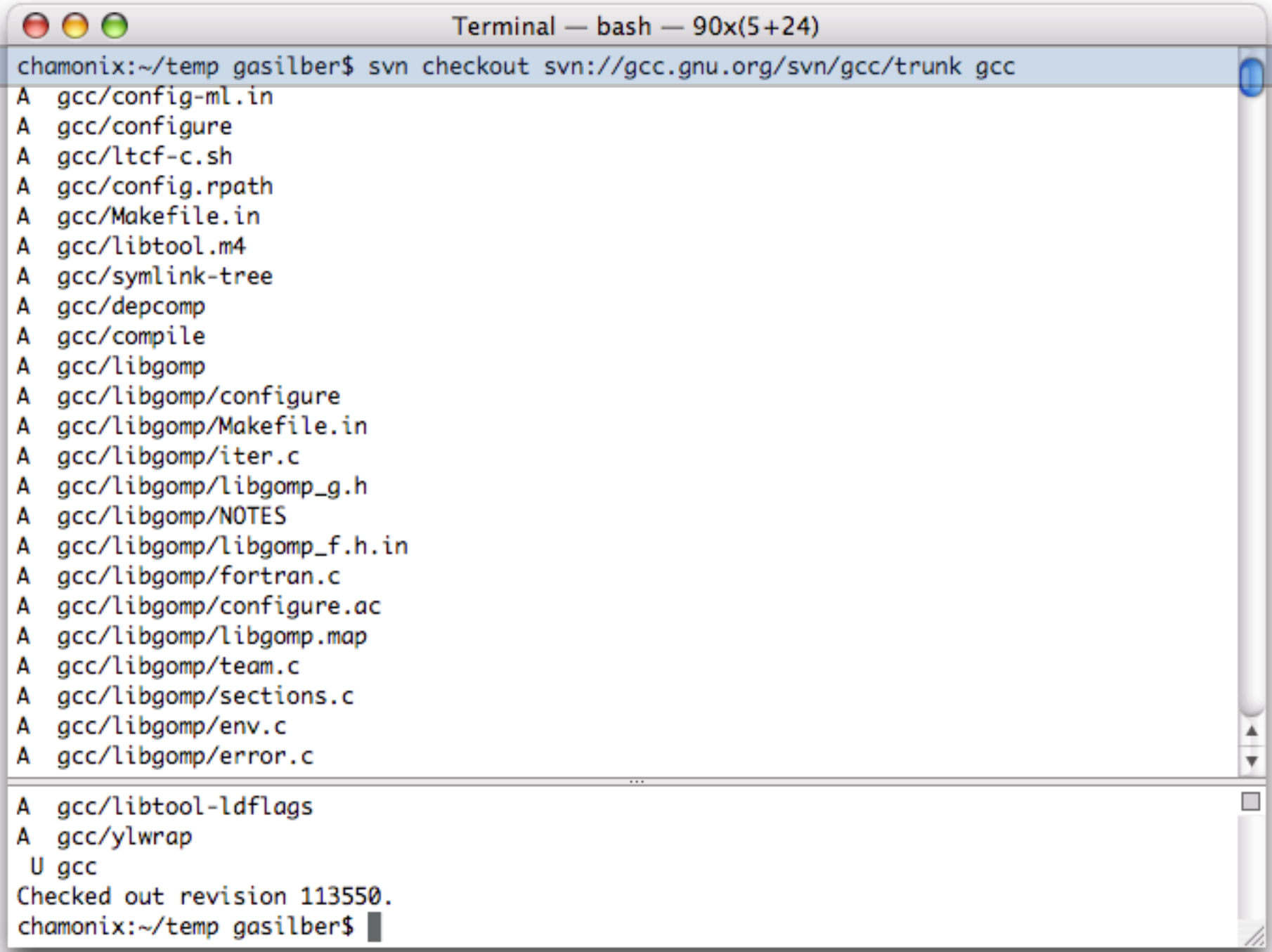
Major decisions about GCC are made by the steering committee, guided by the mission statement.

Current release series: GCC 4.1.0 (released 2006-02-28)
Branch status: 2006-04-16 (open for regression and documentation fixes only). Serious regressions . All known regressions.
Previous release series: GCC 4.0.3 (released 2006-03-10)
Branch status: 2006-03-10 (open for regression and documentation fixes only). Serious Regressions.
Previous release series: GCC 3.4.5 (released 2005-11-30)
Branch status: GCC 3.4.6 is the last release from the 3.4 series; the branch has been closed after the release.
Active development (mainline): will become GCC 4.2.0 (current changes)
Status: Stage 3; 2006-04-19 (open for bugfixes).

News/Announcements

March 10, 2006
GCC 4.0.3 has been released.
March 9, 2006
Richard Henderson, Jakub Jelinek and Diego Novillo of Red Hat Inc, and Dmitry Kurochkin have contributed an implementation of the OpenMP v2.5 parallel programming interface for C, C++ and Fortran.
February 28, 2006
GCC 4.1.0 has been released.
November 30, 2005
GCC 3.4.5 has been released.
October 26, 2005
GCC has moved from CVS to SVN for revision control.
September 28, 2005
GCC 4.0.2 has been released.

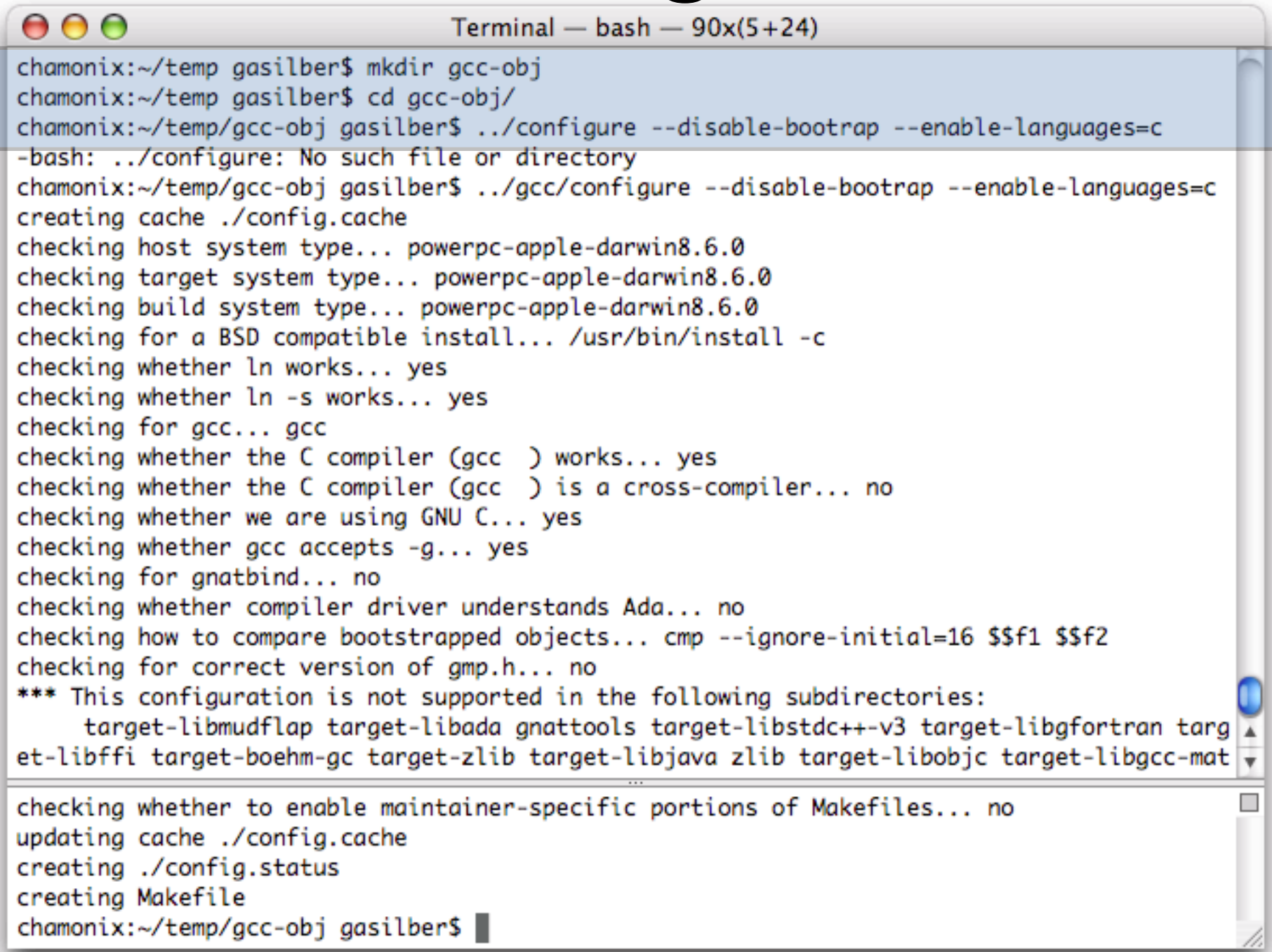
SVN checkout



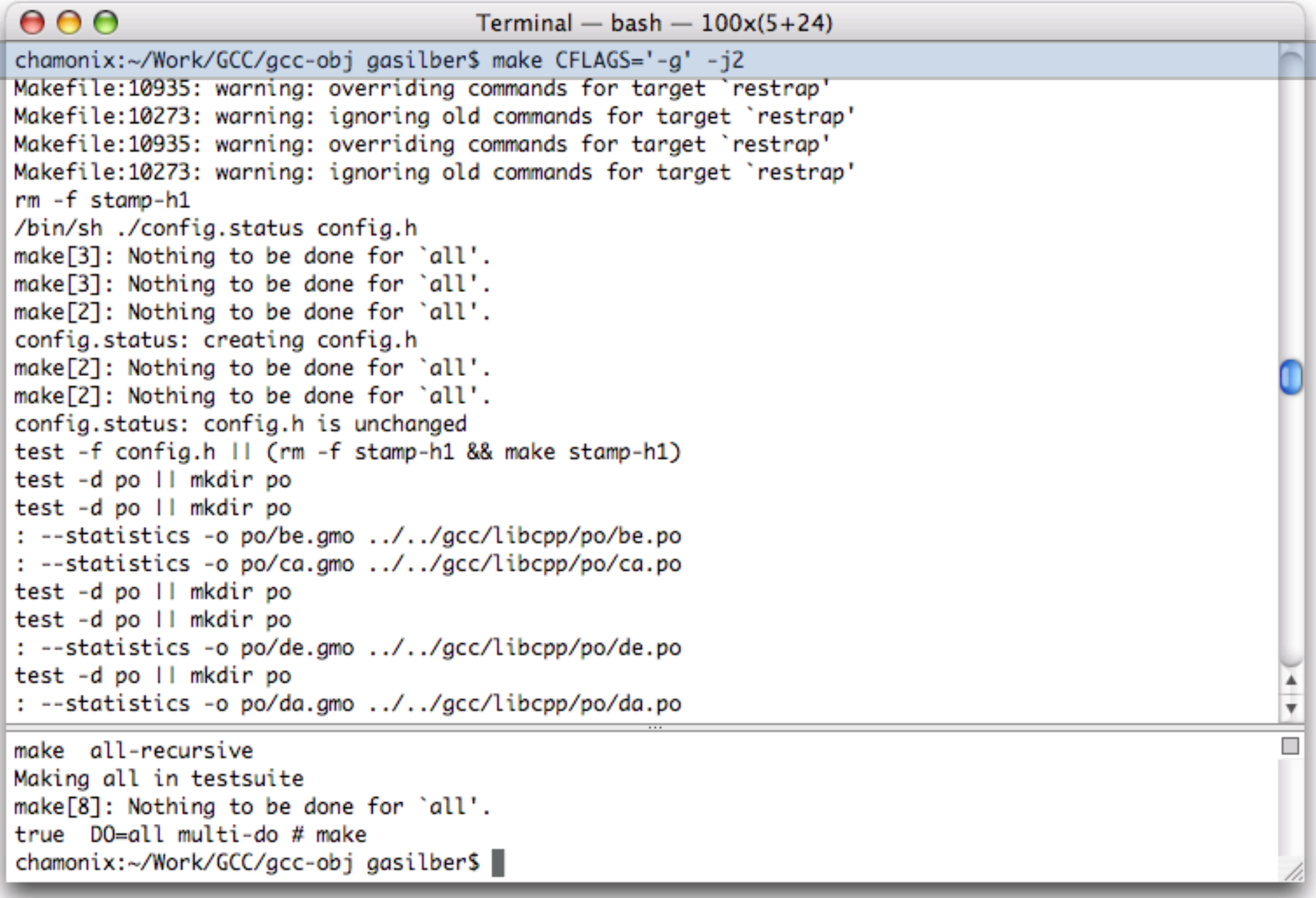
Compiling GCC

- Create a directory, for instance ‘gcc-obj’
- This is the target of the compilation
- cd gcc-obj
- $GCCSRC/configure
- Use ‘--enable-languages=c’ for C only
- Use ‘--disable-bootstrap’ for slow machines
- make

Configure


Terminal — bash — 90x(5+24)
chamonix:~/temp gasilber$ mkdir gcc-obj
chamonix:~/temp gasilber$ cd gcc-obj/
chamonix:~/temp/gcc-obj gasilber$ ../configure --disable-bootrap --enable-languages=c
-bash: ../configure: No such file or directory
chamonix:~/temp/gcc-obj gasilber$ ../gcc/configure --disable-bootrap --enable-languages=c
creating cache ./config.cache
checking host system type... powerpc-apple-darwin8.6.0
checking target system type... powerpc-apple-darwin8.6.0
checking build system type... powerpc-apple-darwin8.6.0
checking for a BSD compatible install... /usr/bin/install -c
checking whether ln works... yes
checking whether ln -s works... yes
checking for gcc... gcc
checking whether the C compiler (gcc) works... yes
checking whether the C compiler (gcc) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for gnatbind... no
checking whether compiler driver understands Ada... no
checking how to compare bootstrapped objects... cmp --ignore-initial=16 $$f1 $$f2
checking for correct version of gmp.h... no
*** This configuration is not supported in the following subdirectories:
target-libmudflap target-libada gnattools target-libstdc++-v3 target-libgfortran targ
et-libffi target-boehm-gc target-zlib target-libjava zlib target-libobjc target-libgcc-mat
checking whether to enable maintainer-specific portions of Makefiles... no
updating cache ./config.cache
creating ./config.status
creating Makefile
chamonix:~/temp/gcc-obj gasilber$

Make


Terminal — bash — 100x(5+24)
chamonix:~/Work/GCC/gcc-obj gasilber$ make CFLAGS='-g' -j2
Makefile:10935: warning: overriding commands for target `restrap'
Makefile:10273: warning: ignoring old commands for target `restrap'
Makefile:10935: warning: overriding commands for target `restrap'
Makefile:10273: warning: ignoring old commands for target `restrap'
rm -f stamp-h1
/bin/sh ./config.status config.h
make[3]: Nothing to be done for `all'.
make[3]: Nothing to be done for `all'.
make[2]: Nothing to be done for `all'.
config.status: creating config.h
make[2]: Nothing to be done for `all'.
make[2]: Nothing to be done for `all'.
config.status: config.h is unchanged
test -f config.h || (rm -f stamp-h1 && make stamp-h1)
test -d po || mkdir po
test -d po || mkdir po
: --statistics -o po/be.gmo ../../gcc/libcpp/po/be.po
: --statistics -o po/ca.gmo ../../gcc/libcpp/po/ca.po
test -d po || mkdir po
test -d po || mkdir po
: --statistics -o po/de.gmo ../../gcc/libcpp/po/de.po
test -d po || mkdir po
: --statistics -o po/da.gmo ../../gcc/libcpp/po/da.po
make all-recursive
Making all in testsuite
make[8]: Nothing to be done for `all'.
true DO=all multi-do # make
chamonix:~/Work/GCC/gcc-obj gasilber$

Using GCC

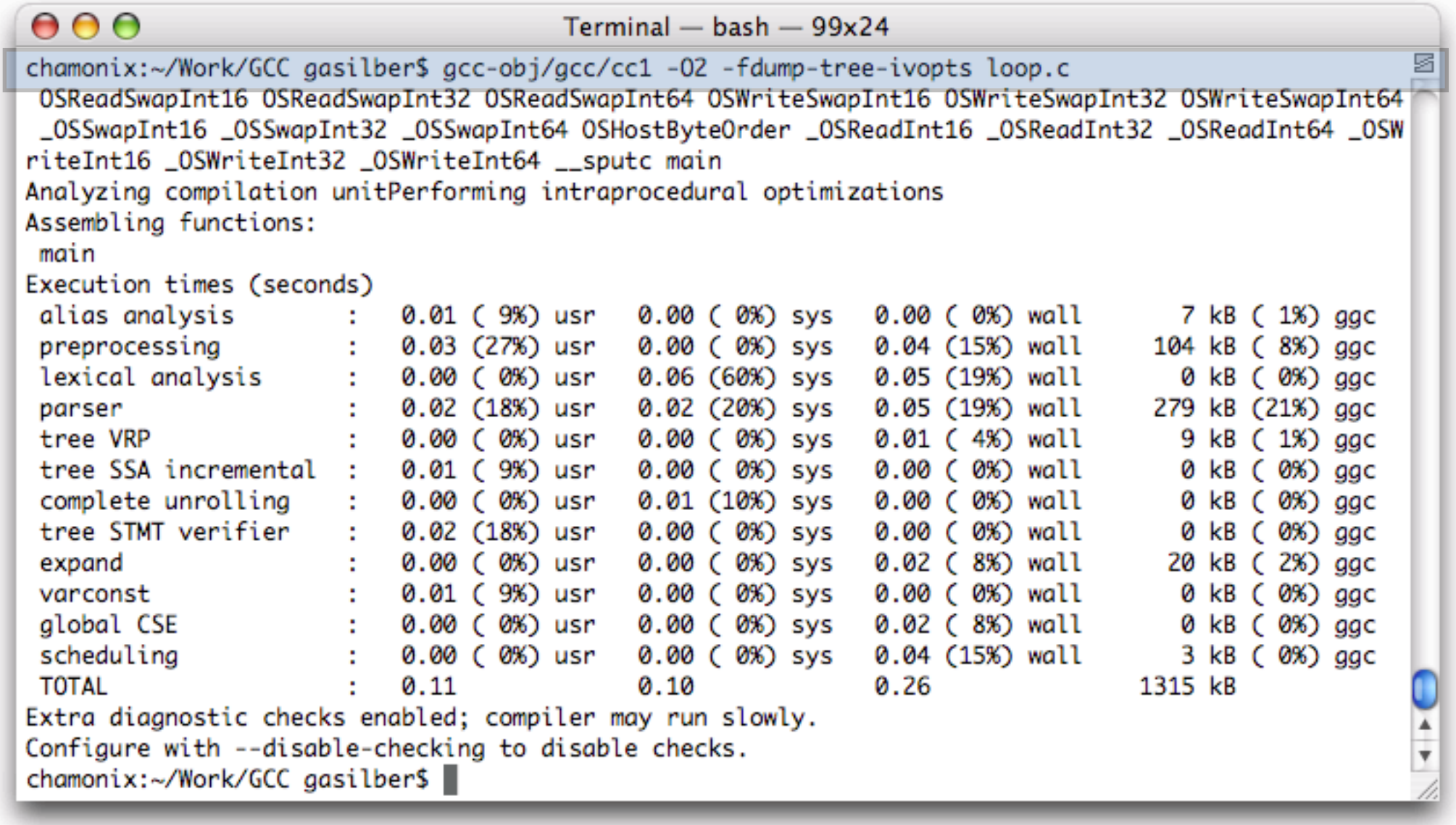
- Considering 'loop.c' is an example code
- The 'cc1' (cc *one*) executable is in 'gcc-obj/gcc'
- gcc-obj/gcc/cc1 -O2 -fdump-tree-ivopts loop.c
- It generates 'loop.s'
- Dump after 'ivopts' pass in 'loop.c.086t.ivopts'
- Note: use 'make install' for full installation

Example

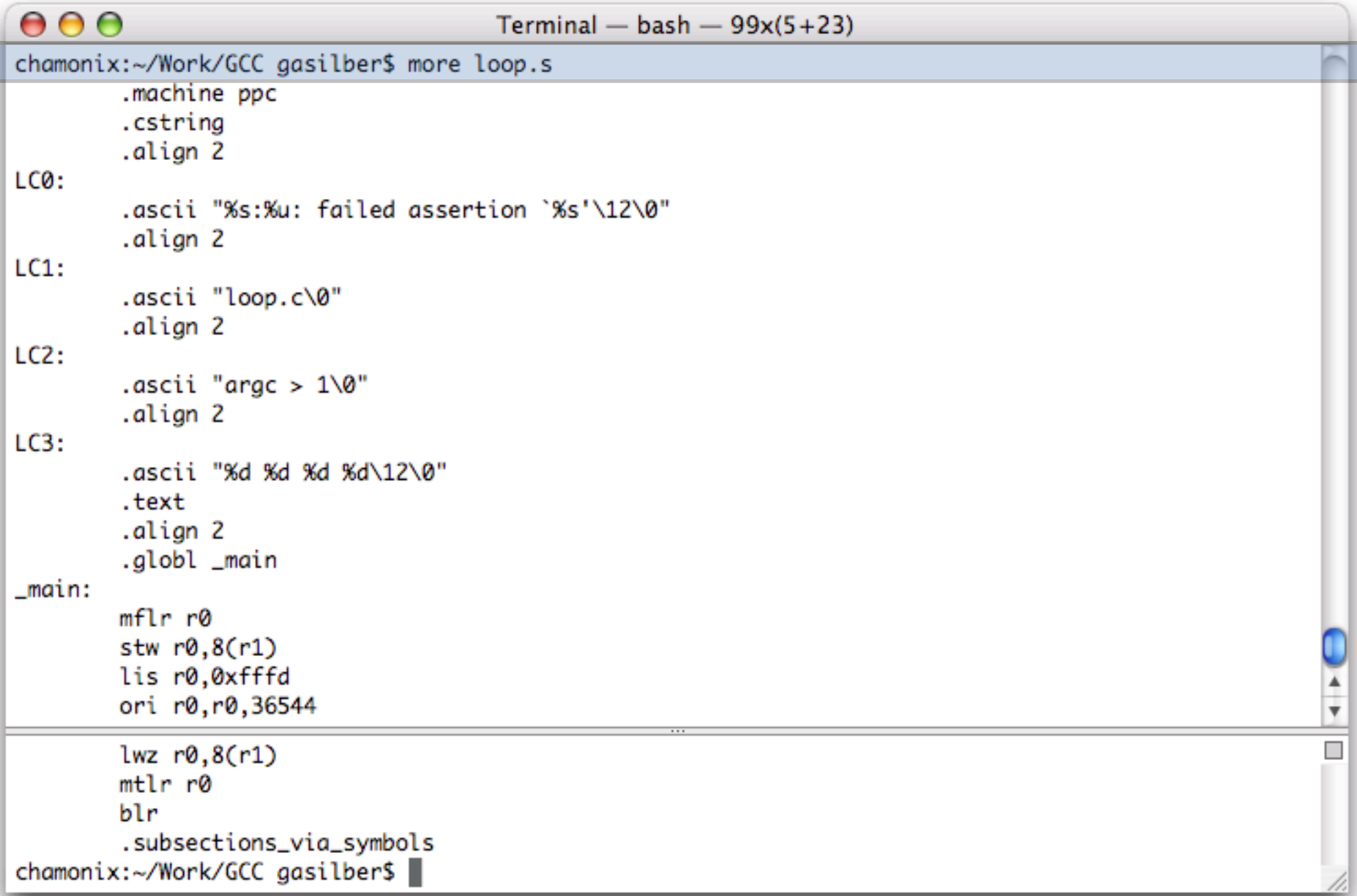
Terminal — bash — 99x28

```
chamonix:~/Work/GCC gasilber$ more loop.c
#include <stdlib.h>
#include <assert.h>
#include <stdio.h>
#define N 10000

int
main (int argc, char const* argv[])
{
  unsigned int i;
  int a[N], b[N], c[N], d[N];
  int k;

  assert (argc > 1);
  k = atoi (argv[1]);
  a[0] = k;
  a[3] = k+1;
  c[1] = k*2;
  for (i = 2; i < (N-1); i ++)
    {
      a[i] = k * i;
      b[i] = a[i-2] + k;
      c[i] = b[i] + a[i+1];
      d[i] = c[i-1] + k + i;
    }
  printf ("%d %d %d %d\n", a[N-2], b[N-1], c[N-2], d[N-2]);
  return 0;
}
```

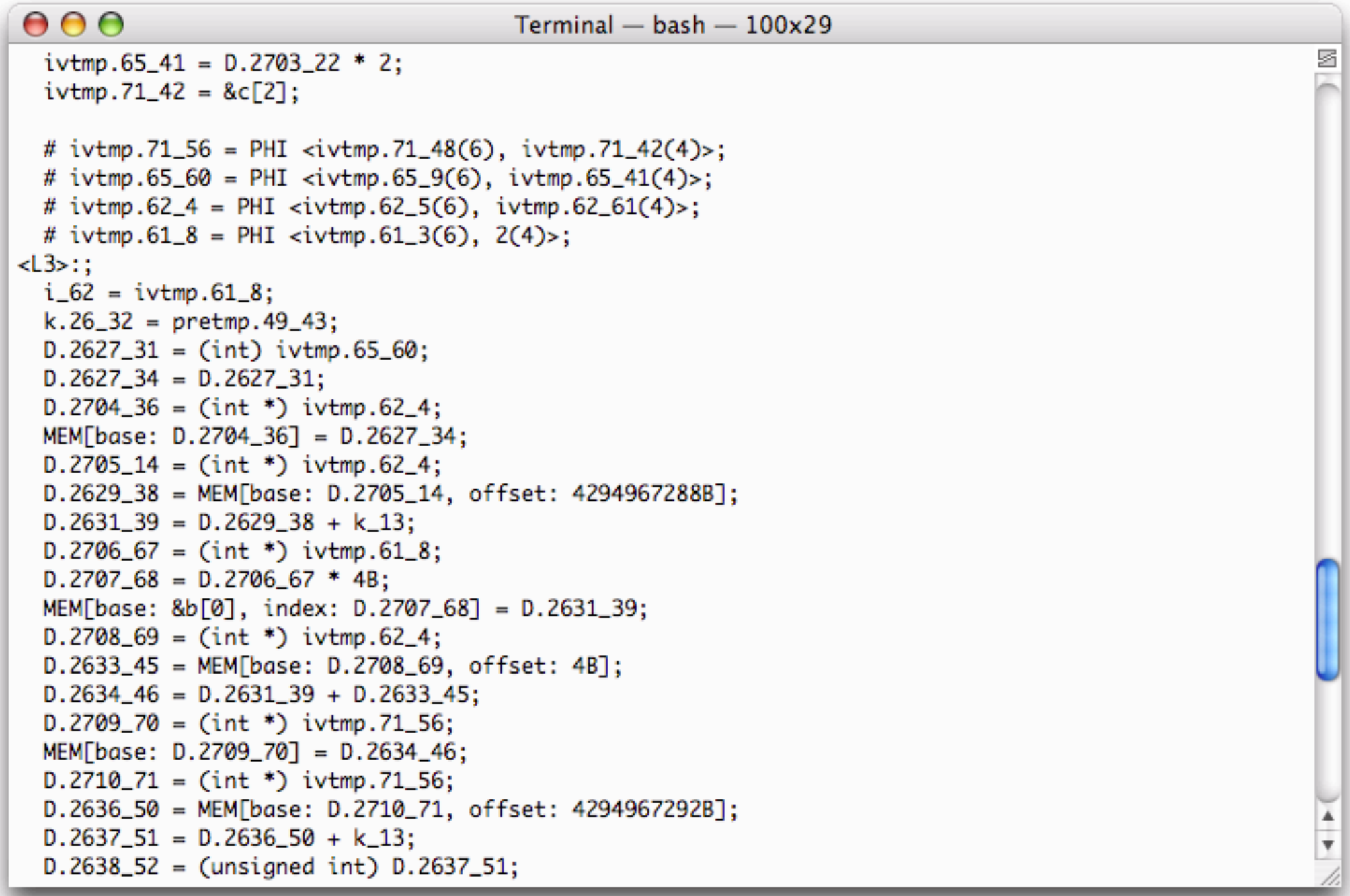
Execute cc1


Terminal — bash — 99x24
chamonix:~/Work/GCC gasilber$ gcc-obj/gcc/cc1 -O2 -fdump-tree-ivopts loop.c
 OSReadSwapInt16 OSReadSwapInt32 OSReadSwapInt64 OSWriteSwapInt16 OSWriteSwapInt32 OSWriteSwapInt64
 _OSSwapInt16 _OSSwapInt32 _OSSwapInt64 OSHostByteOrder _OSReadInt16 _OSReadInt32 _OSReadInt64 _OSW
riteInt16 _OSWriteInt32 _OSWriteInt64 __sputc main
Analyzing compilation unitPerforming intraprocedural optimizations
Assembling functions:
 main
Execution times (seconds)
 alias analysis : 0.01 (9%) usr 0.00 (0%) sys 0.00 (0%) wall 7 kB (1%) ggc
 preprocessing : 0.03 (27%) usr 0.00 (0%) sys 0.04 (15%) wall 104 kB (8%) ggc
 lexical analysis : 0.00 (0%) usr 0.06 (60%) sys 0.05 (19%) wall 0 kB (0%) ggc
 parser : 0.02 (18%) usr 0.02 (20%) sys 0.05 (19%) wall 279 kB (21%) ggc
 tree VRP : 0.00 (0%) usr 0.00 (0%) sys 0.01 (4%) wall 9 kB (1%) ggc
 tree SSA incremental : 0.01 (9%) usr 0.00 (0%) sys 0.00 (0%) wall 0 kB (0%) ggc
 complete unrolling : 0.00 (0%) usr 0.01 (10%) sys 0.00 (0%) wall 0 kB (0%) ggc
 tree STMT verifier : 0.02 (18%) usr 0.00 (0%) sys 0.00 (0%) wall 0 kB (0%) ggc
 expand : 0.00 (0%) usr 0.00 (0%) sys 0.02 (8%) wall 20 kB (2%) ggc
 varconst : 0.01 (9%) usr 0.00 (0%) sys 0.00 (0%) wall 0 kB (0%) ggc
 global CSE : 0.00 (0%) usr 0.00 (0%) sys 0.02 (8%) wall 0 kB (0%) ggc
 scheduling : 0.00 (0%) usr 0.00 (0%) sys 0.04 (15%) wall 3 kB (0%) ggc
 TOTAL : 0.11 0.10 0.26 1315 kB
Extra diagnostic checks enabled; compiler may run slowly.
Configure with --disable-checking to disable checks.
chamonix:~/Work/GCC gasilber$

Assembly code



GIMPLE dump


Terminal — bash — 100x29
ivtmp.65_41 = D.2703_22 * 2;
ivtmp.71_42 = &c[2];
# ivtmp.71_56 = PHI <ivtmp.71_48(6), ivtmp.71_42(4)>;
# ivtmp.65_60 = PHI <ivtmp.65_9(6), ivtmp.65_41(4)>;
# ivtmp.62_4 = PHI <ivtmp.62_5(6), ivtmp.62_61(4)>;
# ivtmp.61_8 = PHI <ivtmp.61_3(6), 2(4)>;
<L3>:;
i_62 = ivtmp.61_8;
k.26_32 = pretmp.49_43;
D.2627_31 = (int) ivtmp.65_60;
D.2627_34 = D.2627_31;
D.2704_36 = (int *) ivtmp.62_4;
MEM[base: D.2704_36] = D.2627_34;
D.2705_14 = (int *) ivtmp.62_4;
D.2629_38 = MEM[base: D.2705_14, offset: 4294967288B];
D.2631_39 = D.2629_38 + k_13;
D.2706_67 = (int *) ivtmp.61_8;
D.2707_68 = D.2706_67 * 4B;
MEM[base: &b[0], index: D.2707_68] = D.2631_39;
D.2708_69 = (int *) ivtmp.62_4;
D.2633_45 = MEM[base: D.2708_69, offset: 4B];
D.2634_46 = D.2631_39 + D.2633_45;
D.2709_70 = (int *) ivtmp.71_56;
MEM[base: D.2709_70] = D.2634_46;
D.2710_71 = (int *) ivtmp.71_56;
D.2636_50 = MEM[base: D.2710_71, offset: 4294967292B];
D.2637_51 = D.2636_50 + k_13;
D.2638_52 = (unsigned int) D.2637_51;

Debugging GCC

A lot of people frequently have questions about debugging GCC. In particular, how to debug the compiler itself, instead of the driver.

Here is a quick rundown:

Assuming you've produced preprocessed source (see the bug reporting directions for how to do this), and have the compiler built somewhere, you can simply do

```
gdb --args <location of cc1, cc1plus, or whatever compiler
   for the language the preprocessed source file is in> <flags passed to compiler>
```

This will enable you to debug the compiler itself, instead of the driver.

You can also use the driver's `-###` option which writes the commands that the driver would execute. For example,

```
gdb --args $(./xgcc -### <parameters to the driver> 2>&1 | fgrep cc1)
```

There are scripts that automate all of this for you here that make debugging a front-end much simpler.

While stepping through a front-end within a debugger, you can use the `debug_tree()` and `debug_rtx()` functions to print out the structure of a tree node or RTL expression respectively.

GCC itself is normally compiled at `-O2` which makes stepping through code a bit difficult. You should use GDB 6.3 (or a newer version), which can work properly with location lists generated by newer GCCs that help in debugging in such cases. Another useful trick is to only compile the particular module you are interested in at a lower optimisation level. For example, if you are debugging `parse.y` in the Java front-end, you can use:

```
$ touch $GCC_SRC_DIR/gcc/java/parse.y
$ make BOOT_CFLAGS='-O0 -g3' bubblestrap
```

If you use GDB to debug GCC and you run the debugger from within the `$GCC_BUILD_DIR/gcc` folder, you get to automatically use the `.gdbinit` file created there by the build process. It defines a few handy macros to help debug GCC. See the file `$GCC_SRC_DIR/gcc/gdbinit.in` for details.

Randomization

You may want to read up on Randomization and disable it if you would like reproducible results

How to add an optimization pass in GCC

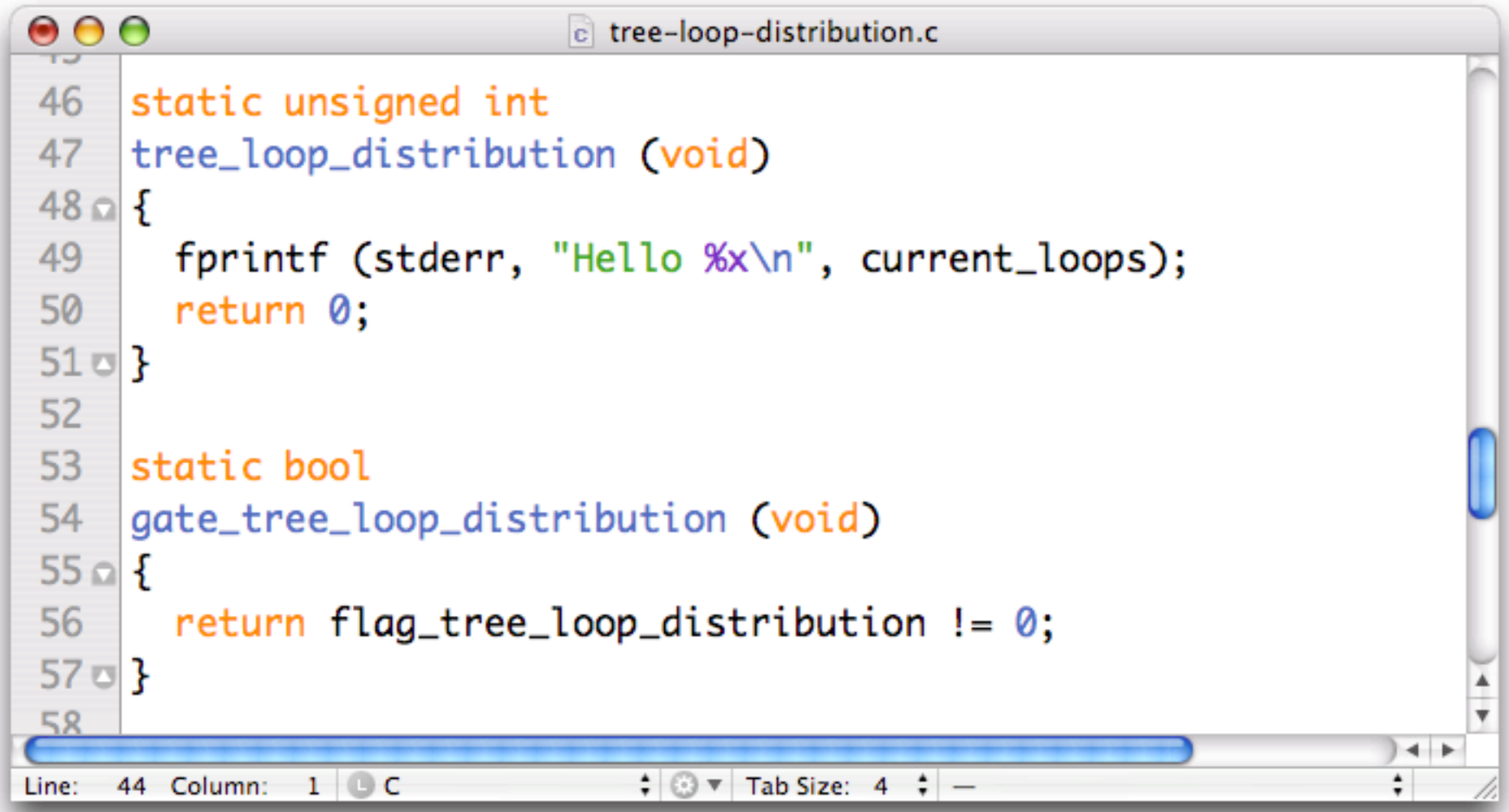
Adding a pass: checklist

- ✓ New pass in file 'gcc/gcc/*mynewpass*.c'
- ✓ Edit 'gcc/gcc/passes.c' (new pass)
- ✓ Edit 'gcc/gcc/tree-flow.h' (prototype)
- ✓ Edit 'gcc/gcc/tree-pass.h' (pass prototype)
- ✓ Edit 'gcc/gcc/common.opt' (new option)
- ✓ Edit 'gcc/gcc/doc/invoke.texi' (doc)
- ✓ Edit 'gcc/gcc/timevar.def' (timing)
- ✓ Edit 'gcc/gcc/Makefile.in' (new target)

New pass file

- New C file in gcc/gcc
- Name: tree-loop-distribution.c
- Pass gate
- Pass function
- Pass structure describing the pass
- The pass is executed for each function

Pass function and gate



Pass structure

tree-loop-distribution.c

```
59  struct tree_opt_pass pass_loop_distribution =
60  {
61    "ldist",                          /* name */
62    gate_tree_loop_distribution,      /* gate */
63    tree_loop_distribution,           /* execute */
64    NULL,                             /* sub */
65    NULL,                             /* next */
66    0,                                /* static_pass_number */
67    TV_TREE_LOOP_DISTRIBUTION,        /* tv_id */
68    PROP_cfg | PROP_ssa,              /* properties_required */
69    0,                                /* properties_provided */
70    0,                                /* properties_destroyed */
71    0,                                /* todo_flags_start */
72    TODO_dump_func | TODO_verify_loops, /* todo_flags_finish */
73    0                                 /* letter */
74  };
75
```

Line: 55 Column: 2 C Tab Size: 4 gate_tree_loop_distribution

Terse name

```
struct tree_opt_pass pass_loop_distribution =
{
  "ldist",                              /* name */
  gate_tree_loop_distribution,          /* gate */
  tree_loop_distribution,               /* execute */
  NULL,                                 /* sub */
  NULL,
  0,
  TV_TREE_LOOP_DISTR
  PROP_cfg | PROP_ss
  0,
  0,
  0,
  TODO_dump_func | TODO_verify_loops,   /* todo_flags_finish */
  0                                     /* letter */
};
```

const char *name;

Terse name of the pass used as a fragment of the dump file name.

gcc/cc1 -O -ftree-loop-distribution -fdump-tree-**ldist**

Gate function

```
struct tree_opt_pass pass_loop_distribution =
{
  "ldist",                          /* name */
  gate_tree_loop_distribution,      /* gate */
  tree_loop_distribution,           /* execute */
  NULL,
  NULL,
  0,
  TV_TREE_LOOP_DIST
  PROP_cfg | PROP_s
  0,
  0,
  0,
  TODO_dump_func |                                  /
  0
};
```

bool (*gate) (void)

If non-null, this pass and all sub-passes are executed only if the function returns true.

Function for execution

```
struct tree_opt_pass pass_loop_distribution =
{
  "ldist",                          /* name */
  gate_tree_loop_distribution,      /* gate */
  tree_loop_distribution,           /* execute */
  NULL,                             /* sub */
  NULL,
  0,
  TV_TREE_LOOP_DISTRI
  PROP_cfg | PROP_ssa
  0,
  0,
  0,
  TODO_dump_func | TO
  0
};
```

unsigned int (*execute) (void)

This is the code to run. If null, then there should be sub-passes otherwise this pass does nothing.

Hierarchy of passes

```
struct tree_opt_pass pass_loop_distribution =
{
  "ldist",                          /* name */
  gate_tree_loop_distribution,      /* gate */
  tree_loop_distribution,           /* execute */
  NULL,                             /* sub */
  NULL,                             /* next */
  0,
  TV_TREE_LOOP_D
  PROP_cfg | PRO
  0,
  0,
  0,
  TODO_dump_func
  0
};
```

struct tree_opt_pass *...;

Passes are chained and can have sub-passes.

Pass number

```
struct tree_opt_pass pass_loop_distribution =
{
  "ldist",                              /* name */
  gate_tree_loop_distribution,          /* gate */
  tree_loop_distribution,               /* execute */
  NULL,                                 /* sub */
  NULL,                                 /* next */
  0,                                    /* static_pass_number */
  TV_TREE_LOOP_DISTRIBUTION,            /* tv_id */
  PROP_cfg | PROP_ssa                   /* properties_required */
  0,
  0,
  0,
  TODO_dump_fu
  0
};
```

int static_pass_number;

Used as a fragment of the dump file name.

Variable for timing

```
struct tree_opt_pass pass_loop_distribution =
{
  "ldist",                              /* name */
  gate_tree_loop_distribution,          /* gate */
  tree_loop_distribution,               /* execute */
  NULL,                                 /* sub */
  NULL,                                 /* next */
  0,                                    /* static_pass_number */
  TV_TREE_LOOP_DISTRIBUTION,            /* tv_id */
  PROP_cfg | PROP_ssa,                  /* properties_required */
  0,
  0,
  0,
  TODO_dump_func | TODO
  0
};
```

unsigned int tv_id;

The timevar id associated with this pass.

Properties

```
struct tree_opt_pass pass_loop_distribution =
{
  "ldist",
  gate_tree_loop_d
  tree_loop_distri
  NULL,
  NULL,
  0,
  TV_TREE_LOOP_DISTRIBUTION,      /* tv_id */
  PROP_cfg | PROP_ssa,            /* properties_required */
  0,                              /* properties_provided */
  0,                              /* properties_destroyed */
  0,                              /* todo_flags_start */
  TODO_dump_func | TODO_verify_loops, /* todo_flags_finish */
  0                               /* letter */
};
```

unsigned int ...;

Passes can require, provide and/or destroy some properties.

Things to do...

```
struct tree_opt_pass pass_loop_distribution =
{
  "ldist",                              /* name */
  gate_tree_loop_distribution,          /* gate */
  tree_loop_distribution,               /* execute */
  NULL,
  NULL,
  0,
  TV_TREE_LOOP_DISTRI
  PROP_cfg | PROP_ssa
  0,
  0,                                    /* properties_destroyed */
  0,                                    /* todo_flags_start */
  TODO_dump_func | TODO_verify_loops,   /* todo_flags_finish */
  0                                     /* letter */
};
```

unsigned int ...;

Things to do before and after the pass.

Letter for RTL dumps

```
struct tree_opt_pass pass_loop_distribution =
{
  "ldist",                              /* name */
  gate_tree_loop_distribution,          /* gate */
  tree_loop_distribution,               /* execute */
  NULL,                                 /* sub */
  NULL,                                 /* next */
  0,
  TV_TREE_LOOP
  PROP_cfg | PR
  0,
  0,
  0,
  TODO_dump_func | TODO_verify_loops, /* todo_flags_finish */
  0                                     /* letter */
};
```

char letter;

Letter used for RTL dumps.

GCC file: passes.c

Adding the pass in the pass hierarchy

init_optimization_passes()

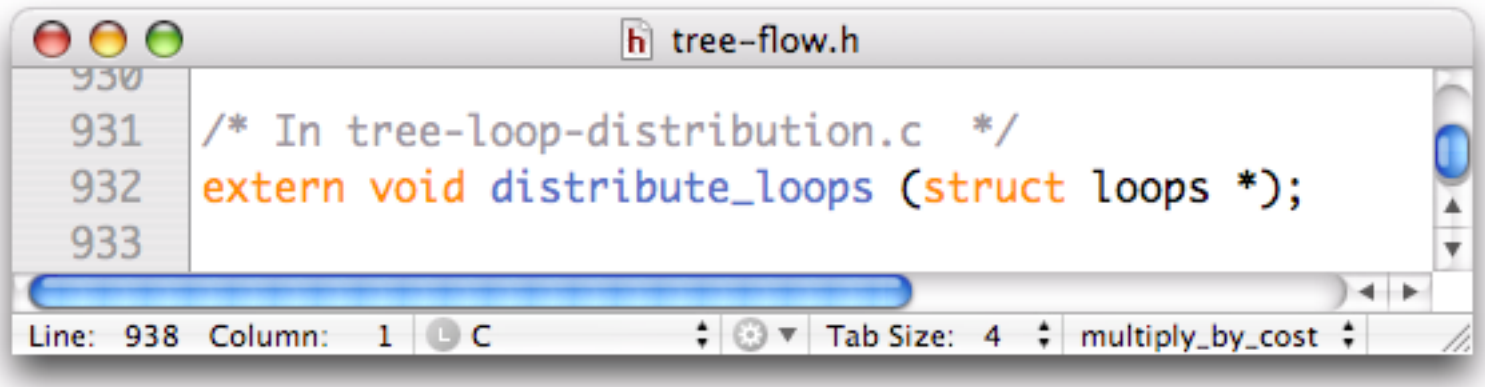
passes.c

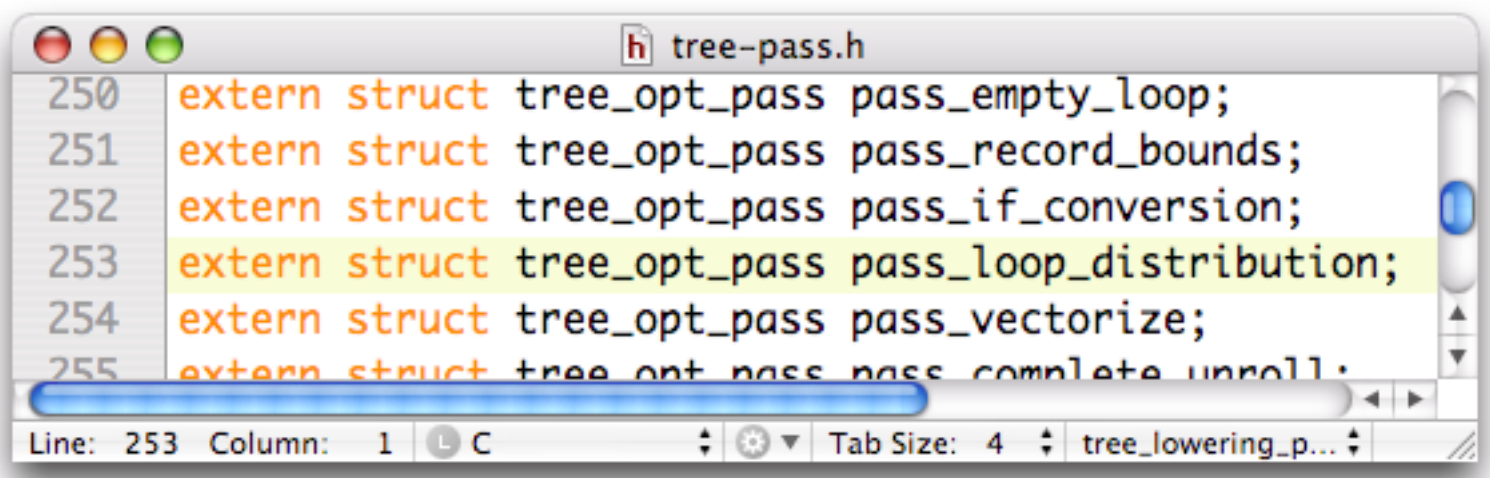
```
593   p = &pass_tree_loop.sub;
594   NEXT_PASS (pass_tree_loop_init);
595   NEXT_PASS (pass_copy_prop);
596   NEXT_PASS (pass_lim);
597   NEXT_PASS (pass_tree_unswitch);
598   NEXT_PASS (pass_scev_cprop);
599   NEXT_PASS (pass_empty_loop);
600   NEXT_PASS (pass_record_bounds);
601   NEXT_PASS (pass_linear_transform);
602   NEXT_PASS (pass_loop_distribution);
603   NEXT_PASS (pass_iv_canon);
604   NEXT_PASS (pass_if_conversion);
605   NEXT_PASS (pass_vectorize);
606   NEXT_PASS (pass_complete_unroll);
607   NEXT_PASS (pass_loop_prefetch);
608   NEXT_PASS (pass_iv_optimize);
609   NEXT_PASS (pass_tree_loop_done);
610   *p = NULL;
```

Line: 602 Column: 3 C Tab Size: 4 NEXT_PASS

tree-flow.h / tree-pass.h

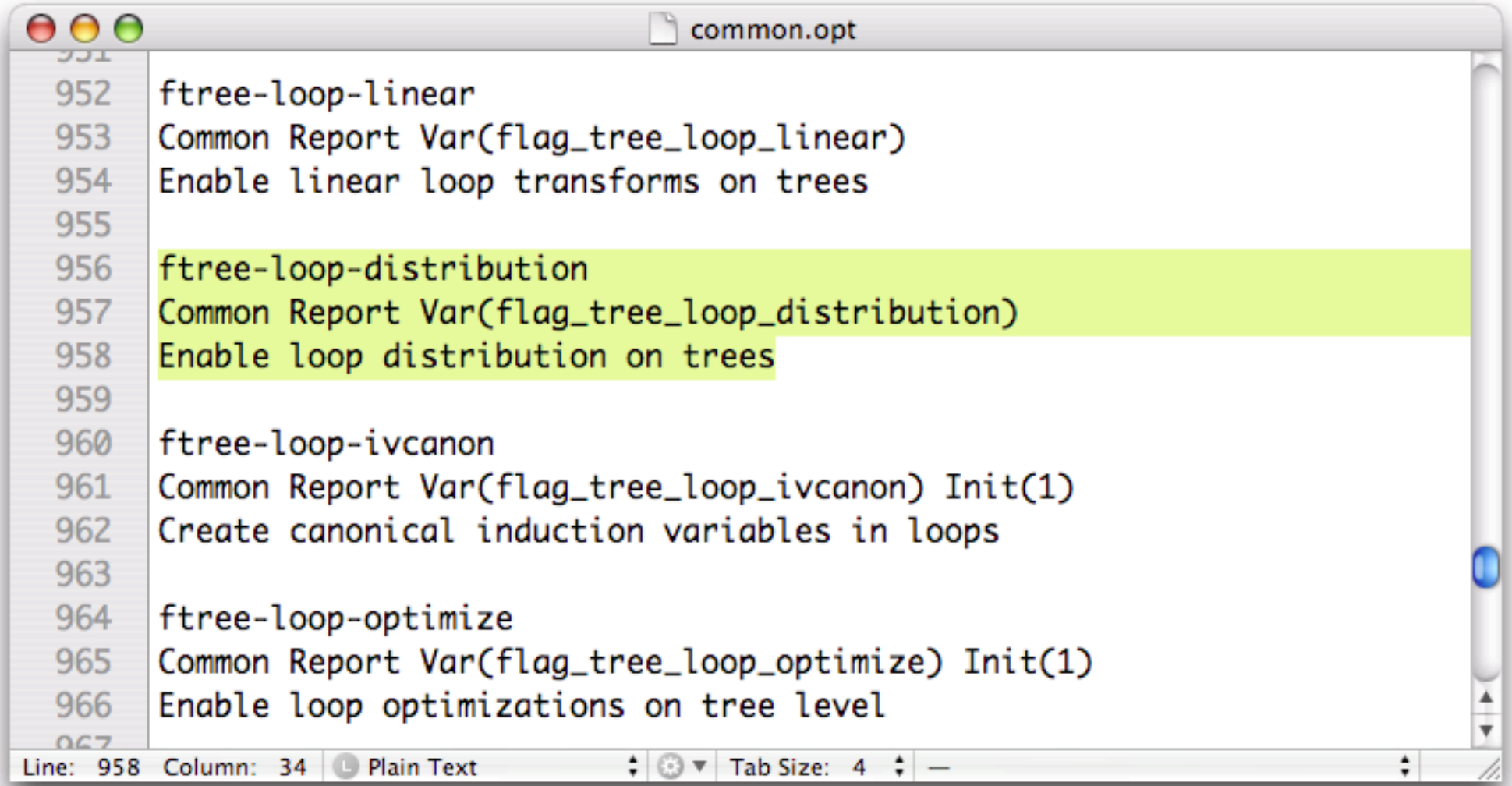
Prototypes for pass function and structure.





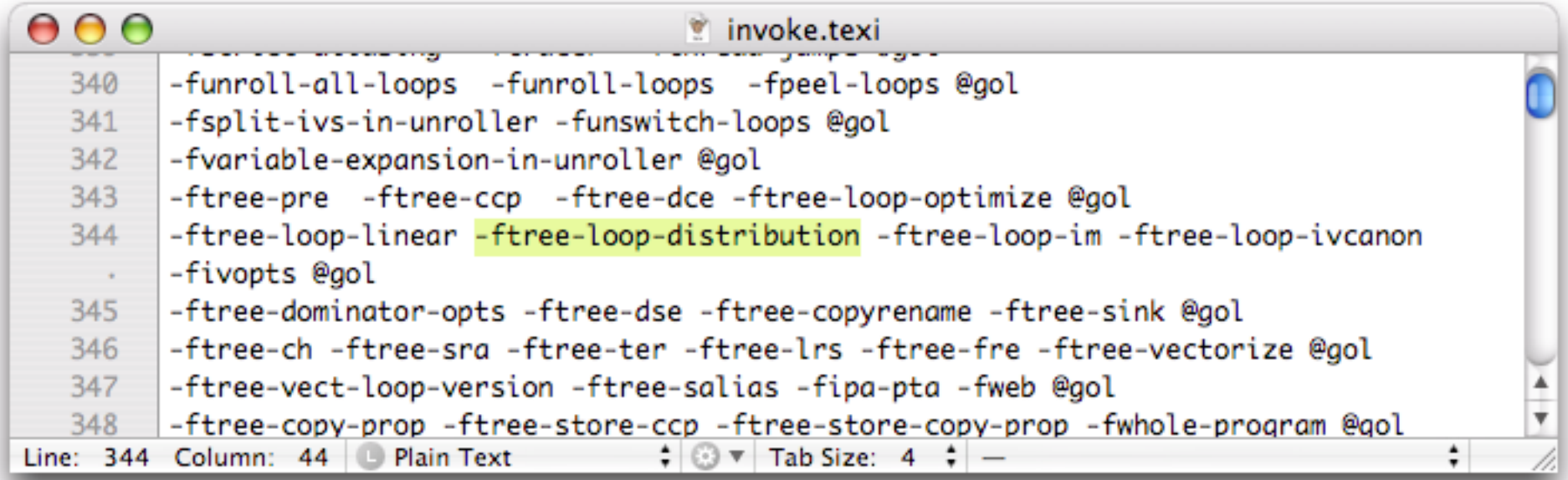
common.opt

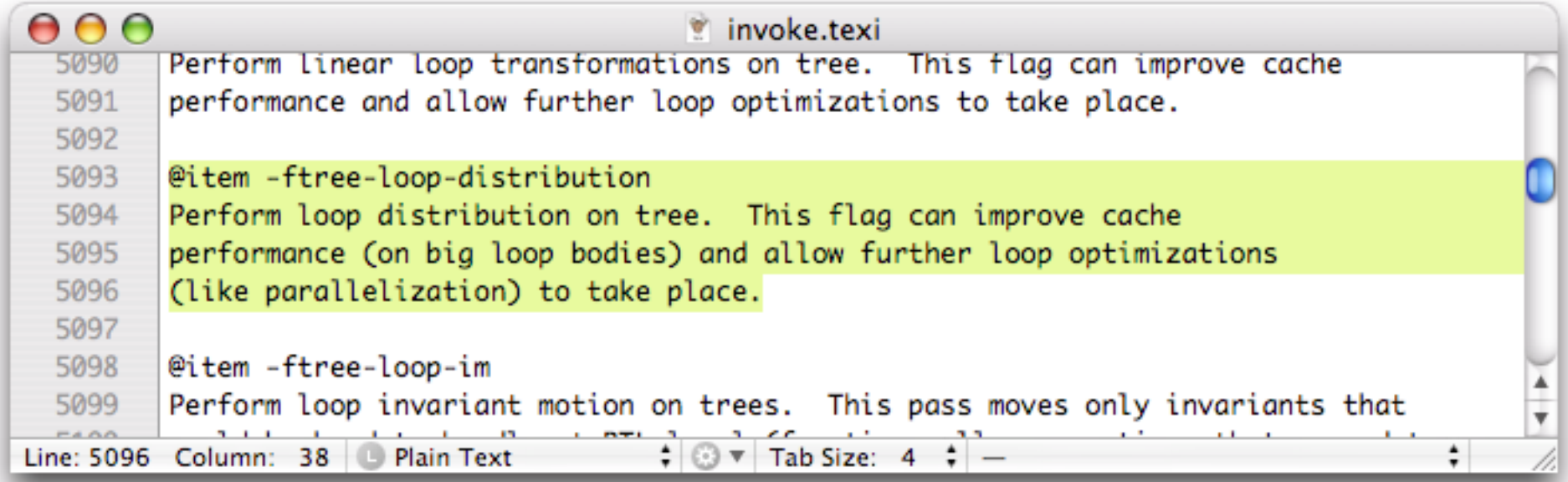
Command line option and internal flag.



Doc: invoke.texi

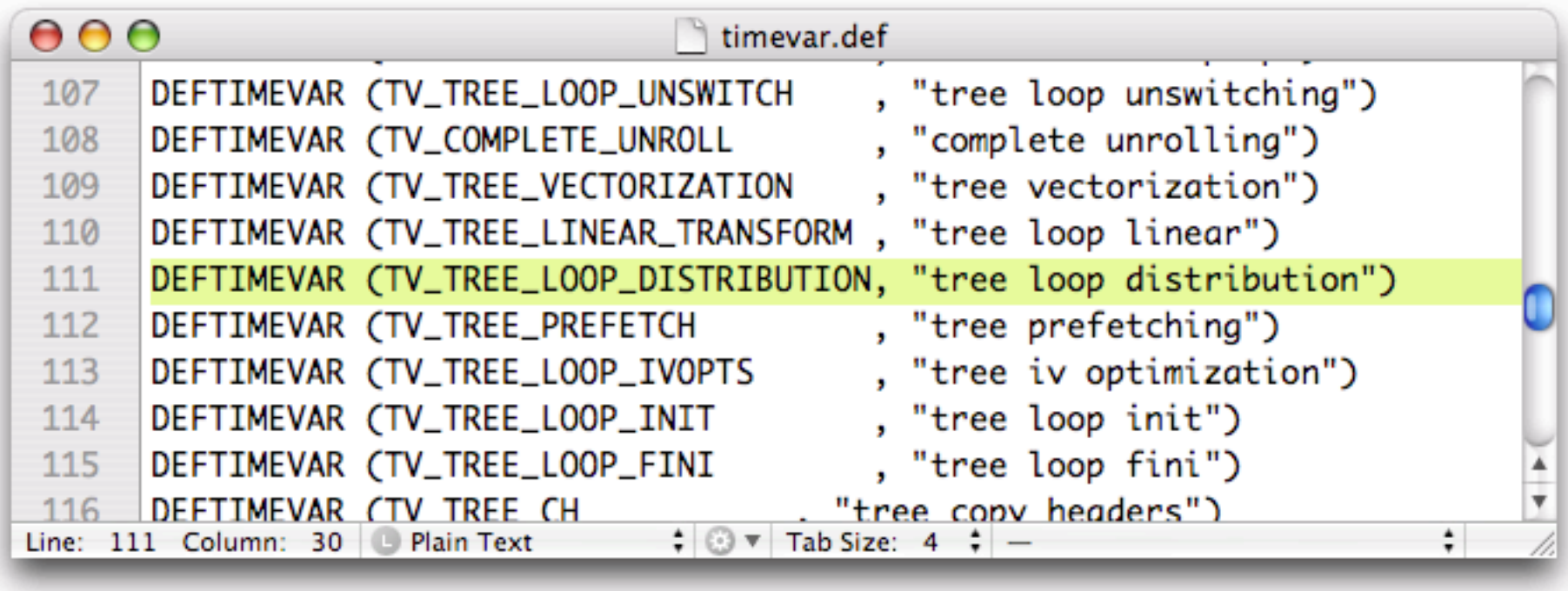
Documenting the pass for the GCC manual.



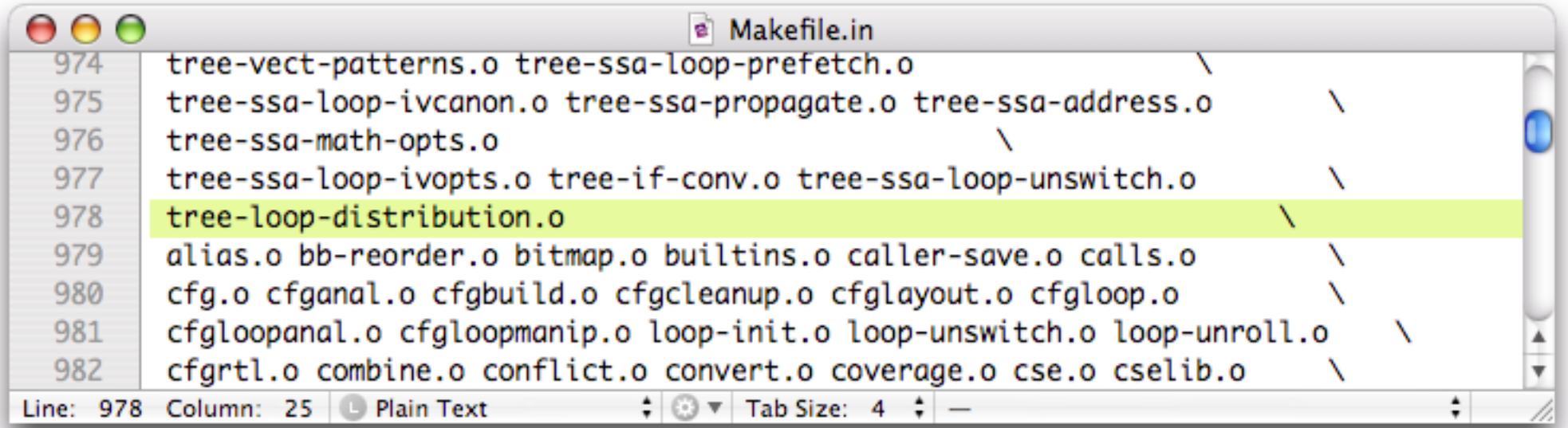


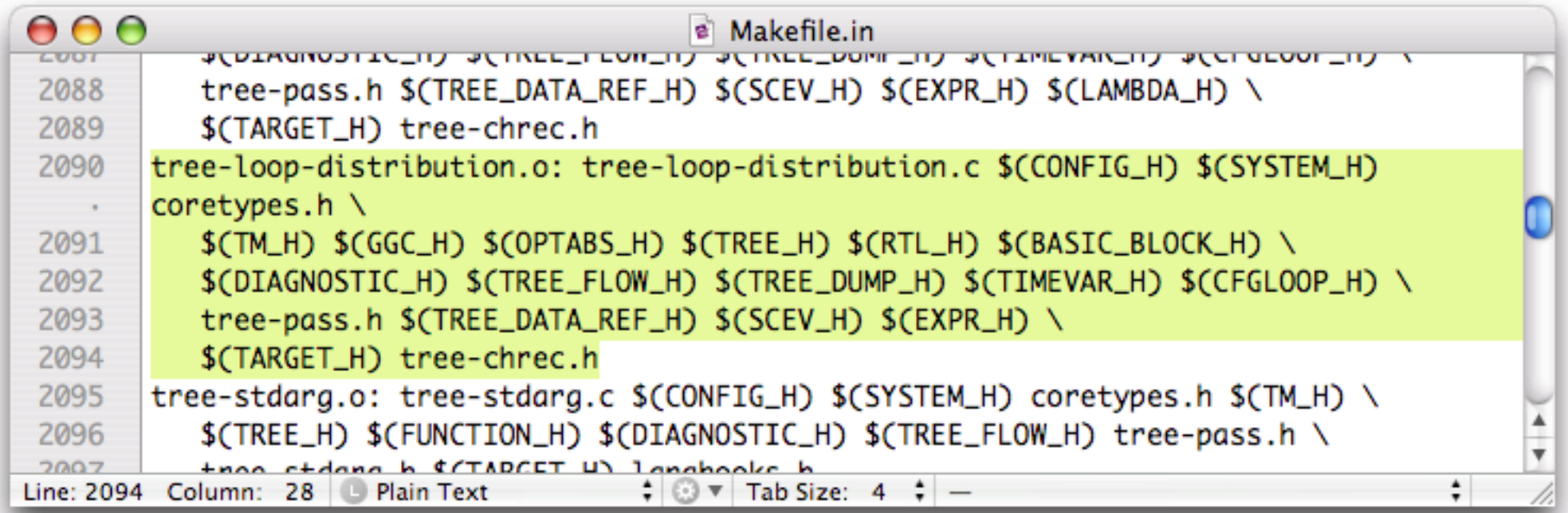
Timing: timevar.def

Variable used for timing and identification in the timing report.

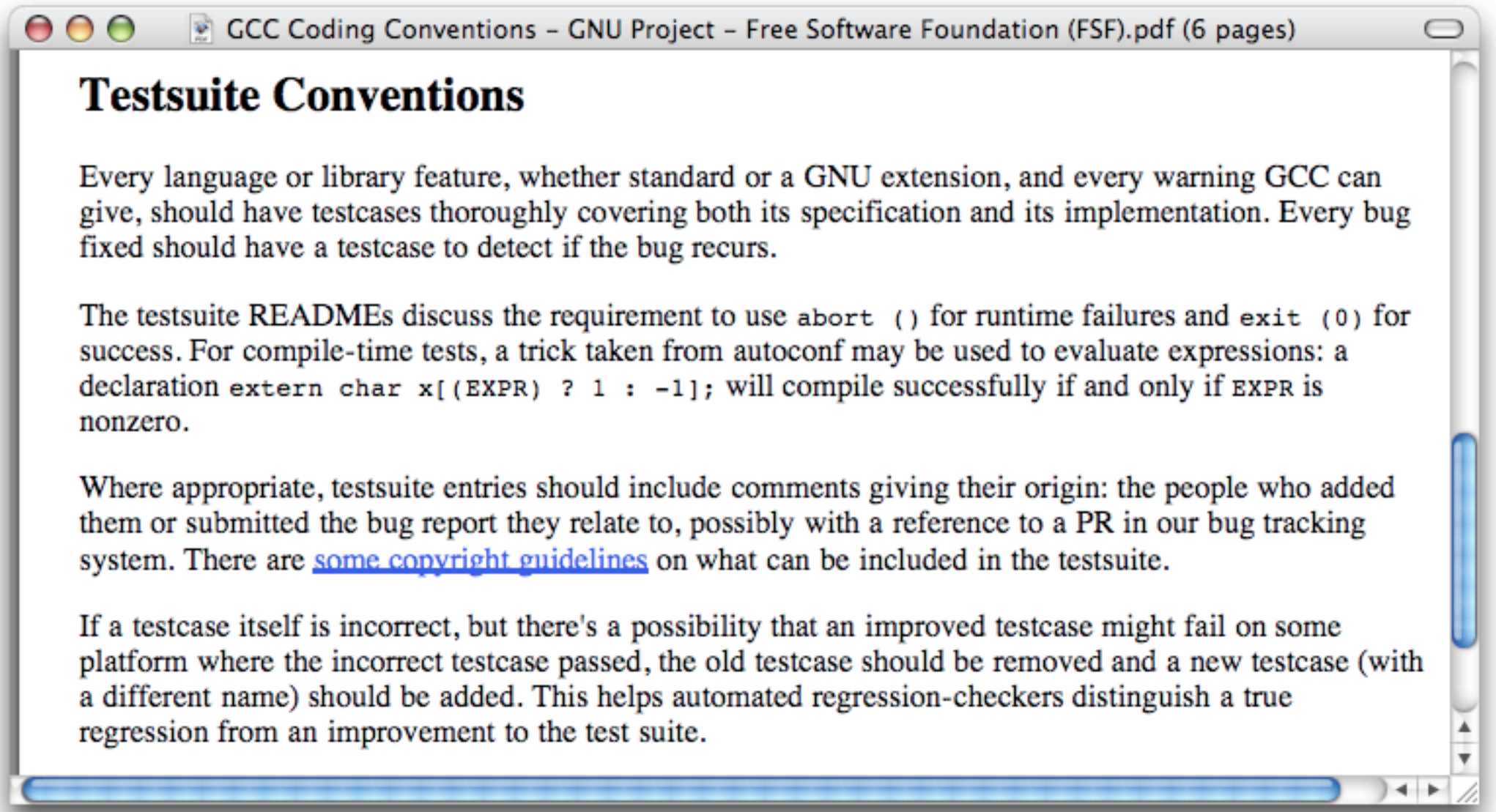


Makefile.in


Makefile.in
974 tree-vect-patterns.o tree-ssa-loop-prefetch.o \
975 tree-ssa-loop-ivcanon.o tree-ssa-propagate.o tree-ssa-address.o \
976 tree-ssa-math-opts.o \
977 tree-ssa-loop-ivopts.o tree-if-conv.o tree-ssa-loop-unswitch.o \
978 tree-loop-distribution.o \
979 alias.o bb-reorder.o bitmap.o builtins.o caller-save.o calls.o \
980 cfg.o cfganal.o cfgbuild.o cfgcleanup.o cfglayout.o cfgloop.o \
981 cfgloopanal.o cfgloopmanip.o loop-init.o loop-unswitch.o loop-unroll.o \
982 cfgrtl.o combine.o conflict.o convert.o coverage.o cse.o cselib.o \
Line: 978 Column: 25 Plain Text Tab Size: 4

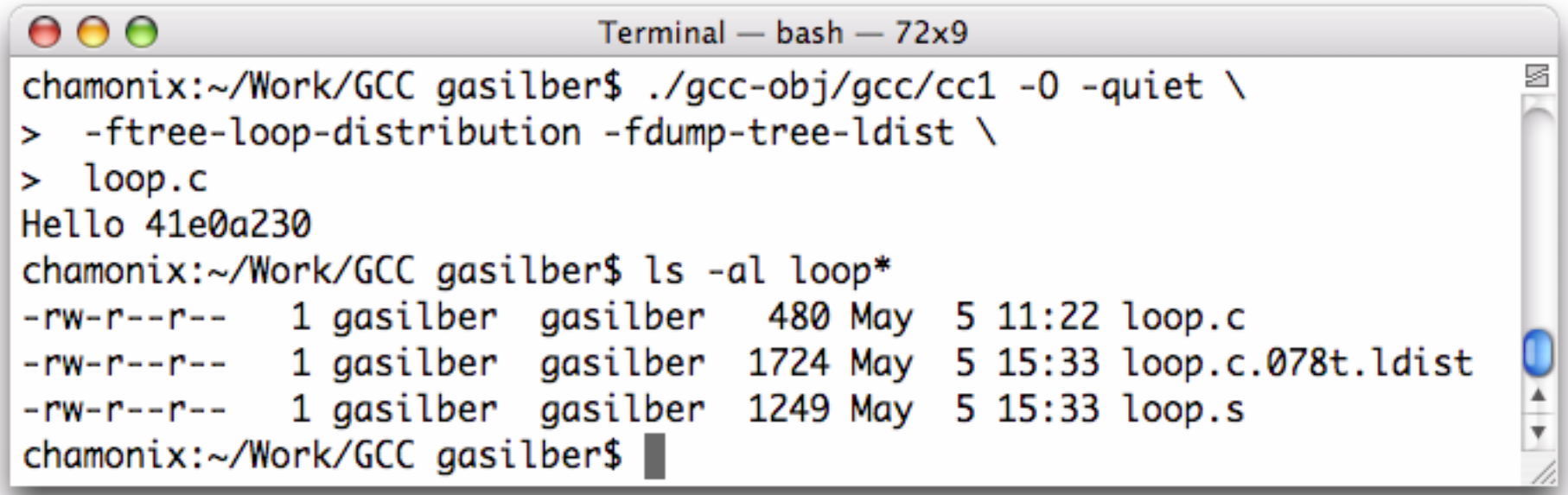

Makefile.in
2088 tree-pass.h $(TREE_DATA_REF_H) $(SCEV_H) $(EXPR_H) $(LAMBDA_H) \
2089 $(TARGET_H) tree-chrec.h
2090 tree-loop-distribution.o: tree-loop-distribution.c $(CONFIG_H) $(SYSTEM_H)
coretypes.h \
2091 $(TM_H) $(GGC_H) $(OPTABS_H) $(TREE_H) $(RTL_H) $(BASIC_BLOCK_H) \
2092 $(DIAGNOSTIC_H) $(TREE_FLOW_H) $(TREE_DUMP_H) $(TIMEVAR_H) $(CFGLOOP_H) \
2093 tree-pass.h $(TREE_DATA_REF_H) $(SCEV_H) $(EXPR_H) \
2094 $(TARGET_H) tree-chrec.h
2095 tree-stdarg.o: tree-stdarg.c $(CONFIG_H) $(SYSTEM_H) coretypes.h $(TM_H) \
2096 $(TREE_H) $(FUNCTION_H) $(DIAGNOSTIC_H) $(TREE_FLOW_H) tree-pass.h \
Line: 2094 Column: 28 Plain Text Tab Size: 4

Testsuite


GCC Coding Conventions – GNU Project – Free Software Foundation (FSF).pdf (6 pages)
Testsuite Conventions
Every language or library feature, whether standard or a GNU extension, and every warning GCC can give, should have testcases thoroughly covering both its specification and its implementation. Every bug fixed should have a testcase to detect if the bug recurs.
The testsuite READMEs discuss the requirement to use abort () for runtime failures and exit (0) for success. For compile-time tests, a trick taken from autoconf may be used to evaluate expressions: a declaration extern char x[(EXPR) ? 1 : -1]; will compile successfully if and only if EXPR is nonzero.
Where appropriate, testsuite entries should include comments giving their origin: the people who added them or submitted the bug report they relate to, possibly with a reference to a PR in our bug tracking system. There are some copyright guidelines on what can be included in the testsuite.
If a testcase itself is incorrect, but there's a possibility that an improved testcase might fail on some platform where the incorrect testcase passed, the old testcase should be removed and a new testcase (with a different name) should be added. This helps automated regression-checkers distinguish a true regression from an improvement to the test suite.

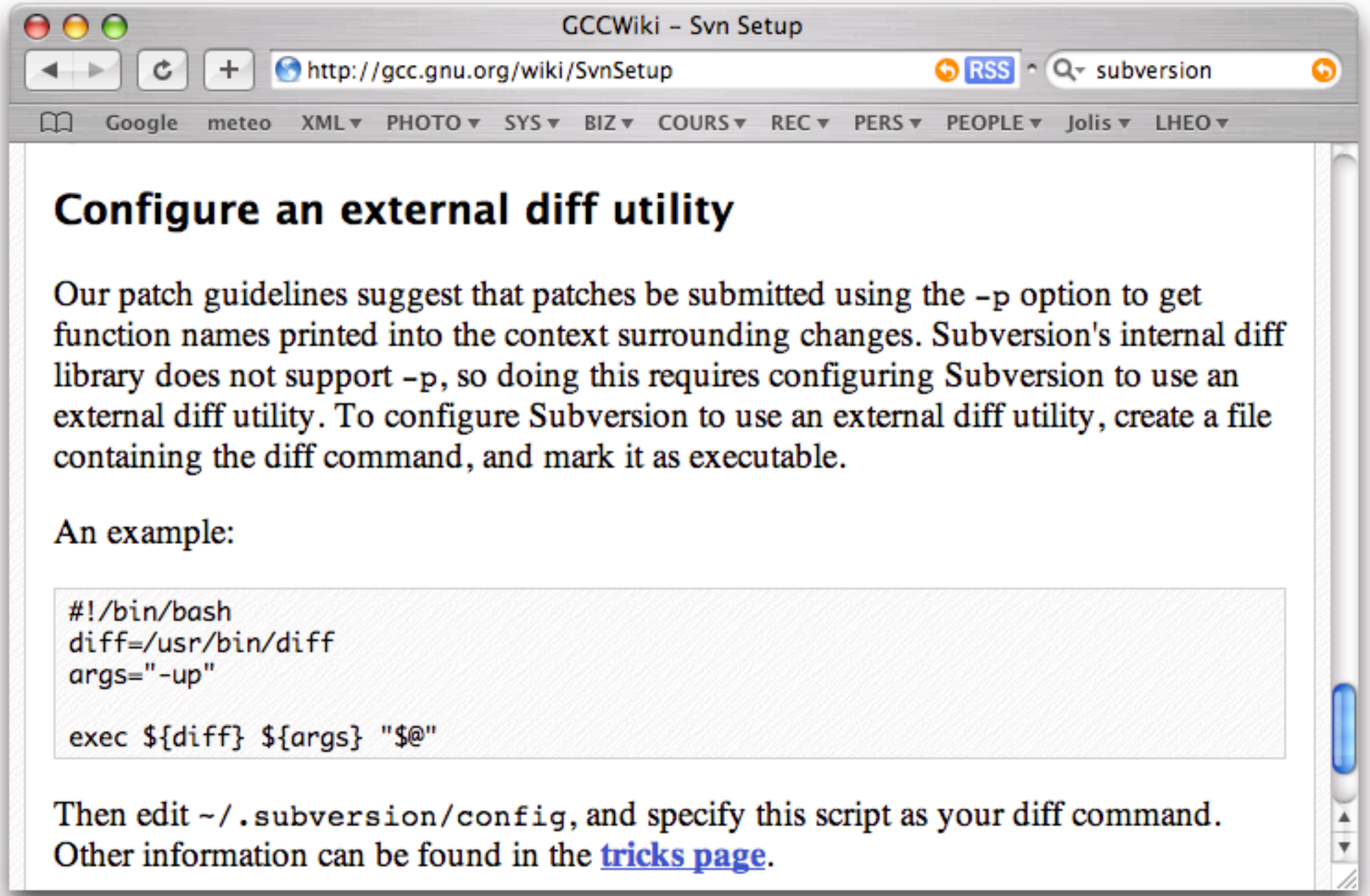
Test

gcc-obj/gcc/cc1 -O
-ftree-loop-distribution
-fdump-tree-ldist
loop.c



Preparing a patch

- In ‘gcc/gcc’ issue a ‘svn diff > mypatch’
- Edit the patch to add a ‘Changelog’
- Apply: patch -p0 < mypatch



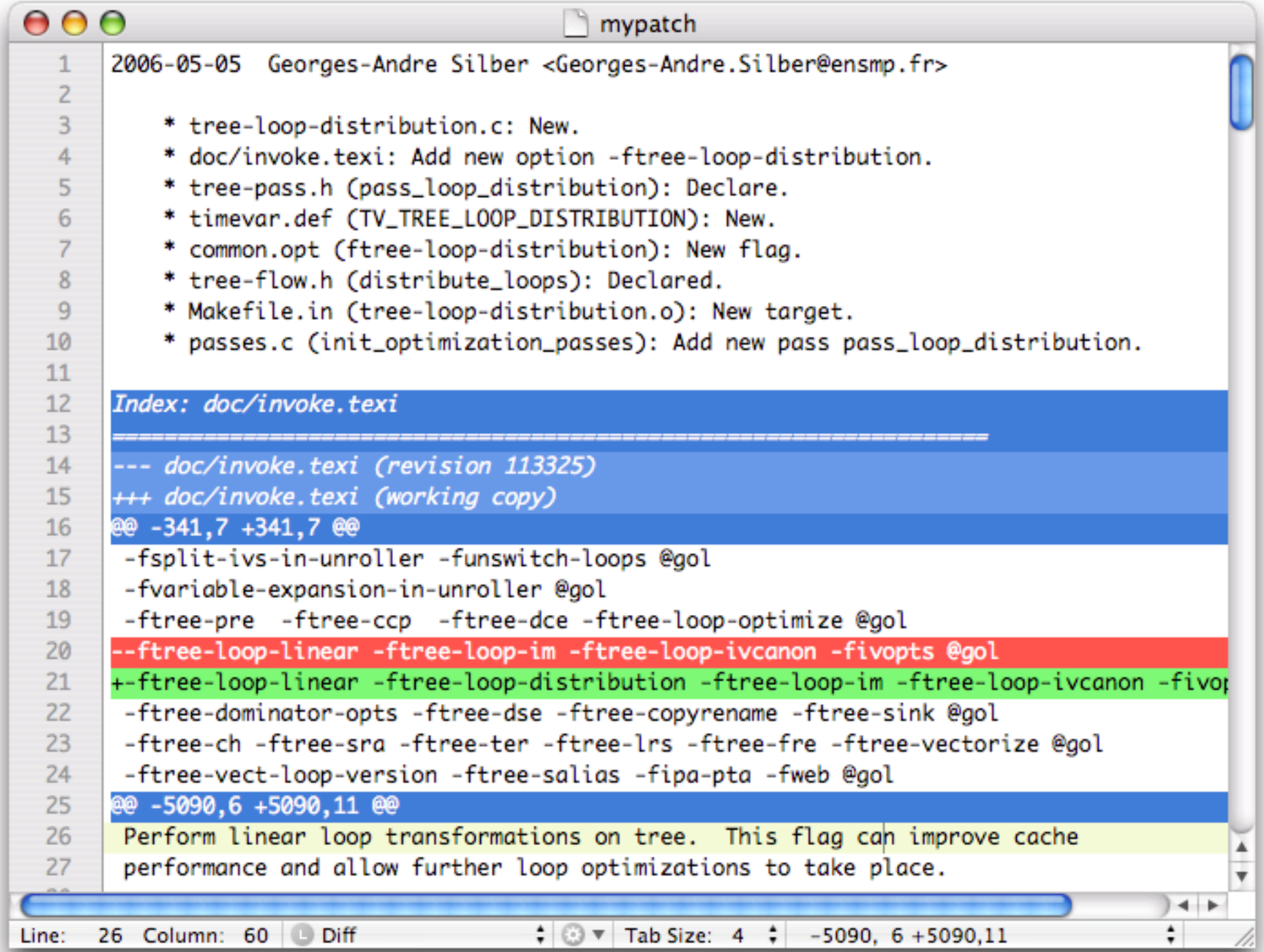
Configure an external diff utility

Our patch guidelines suggest that patches be submitted using the `-p` option to get function names printed into the context surrounding changes. Subversion's internal diff library does not support `-p`, so doing this requires configuring Subversion to use an external diff utility. To configure Subversion to use an external diff utility, create a file containing the diff command, and mark it as executable.

An example:

```
#!/bin/bash
diff=/usr/bin/diff
args="-up"

exec ${diff} ${args} "$@"
```

Then edit `~/.subversion/config`, and specify this script as your diff command. Other information can be found in the tricks page.


mypatch
2006-05-05 Georges-Andre Silber <Georges-Andre.Silber@ensmp.fr>
* tree-loop-distribution.c: New.
* doc/invoke.texi: Add new option -ftree-loop-distribution.
* tree-pass.h (pass_loop_distribution): Declare.
* timevar.def (TV_TREE_LOOP_DISTRIBUTION): New.
* common.opt (ftree-loop-distribution): New flag.
* tree-flow.h (distribute_loops): Declared.
* Makefile.in (tree-loop-distribution.o): New target.
* passes.c (init_optimization_passes): Add new pass pass_loop_distribution.
Index: doc/invoke.texi
--- doc/invoke.texi (revision 113325)
+++ doc/invoke.texi (working copy)
@@ -341,7 +341,7 @@
-fsplit-ivs-in-unroller -funswitch-loops @gol
-fvariable-expansion-in-unroller @gol
-ftree-pre -ftree-ccp -ftree-dce -ftree-loop-optimize @gol
--ftree-loop-linear -ftree-loop-im -ftree-loop-ivcanon -fivopts @gol
+-ftree-loop-linear -ftree-loop-distribution -ftree-loop-im -ftree-loop-ivcanon -fivop
-ftree-dominator-opts -ftree-dse -ftree-copyrename -ftree-sink @gol
-ftree-ch -ftree-sra -ftree-ter -ftree-lrs -ftree-fre -ftree-vectorize @gol
-ftree-vect-loop-version -ftree-salias -fipa-pta -fweb @gol
@@ -5090,6 +5090,11 @@
Perform linear loop transformations on tree. This flag can improve cache
performance and allow further loop optimizations to take place.
Line: 26 Column: 60 Diff Tab Size: 4 -5090, 6 +5090,11

Case study: loop distribution

```
DO i=2,N
  S1
  S2
ENDDO
```

```
DO i=2,N
  S1
ENDDO
DO i=2,N
  S2
ENDDO
```

```
DO i=2,N
  S2
ENDDO
DO i=2,N
  S1
ENDDO
```

```
DO i=2,N
  S2
  S1
ENDDO
```

Why loop distribution?

- Typical pass in compiling technology
  - Especially for the source-to-source community
- Can increase parallelism and cache hits
- (Can decrease performance and cache hits)
- Goal: help the vectorizer of GCC

How to distribute?

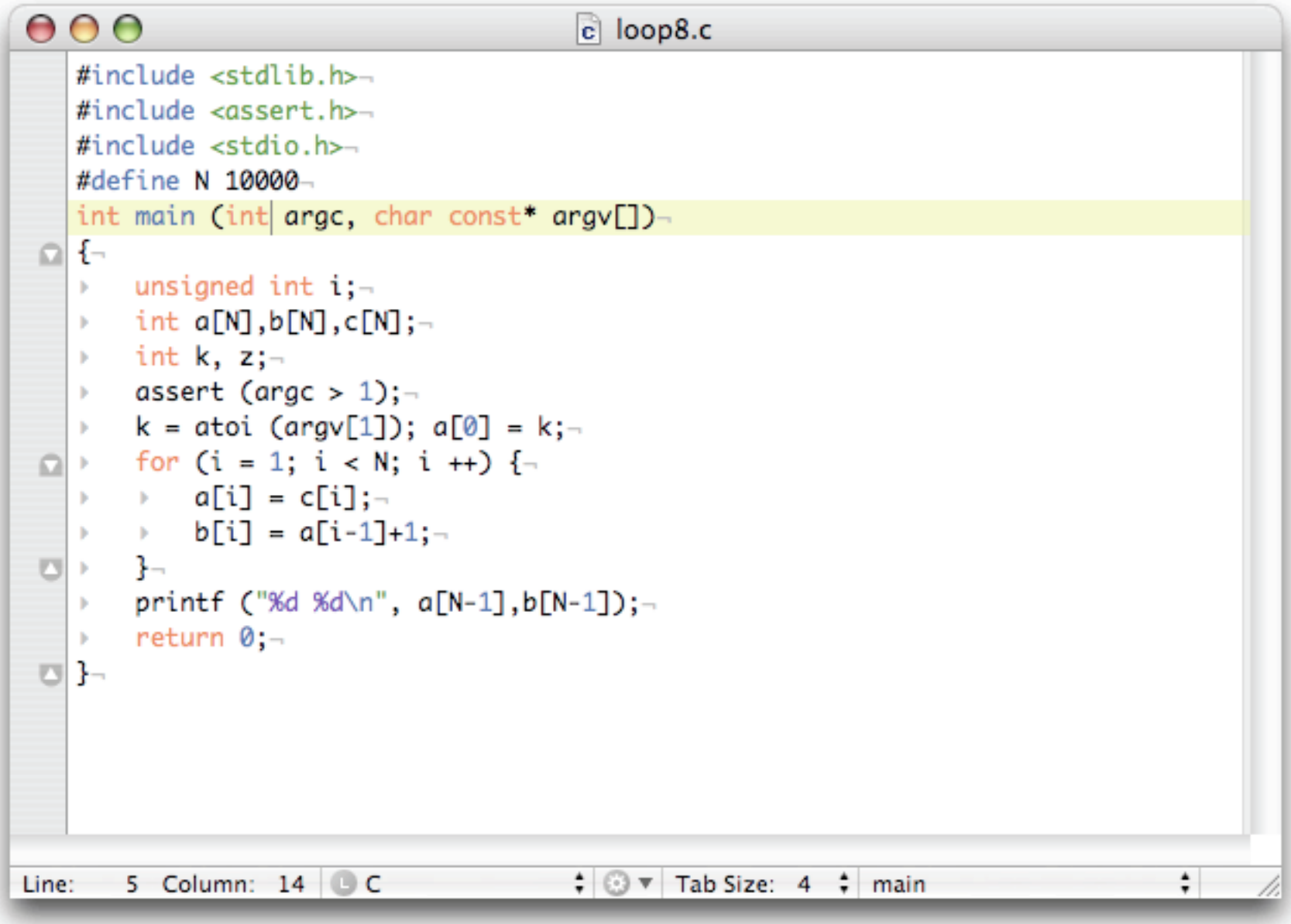
Algorithm by Allen, Callahan, and Kennedy
(simplified version for a loop nest of depth one)

- Build a data dependence graph with levels
- Find the Strongly Connected Components
- Rewrite the new loops according to a topological sort of the SCCs
- Produces the maximal number of parallel loops for a data dependence graph with levels

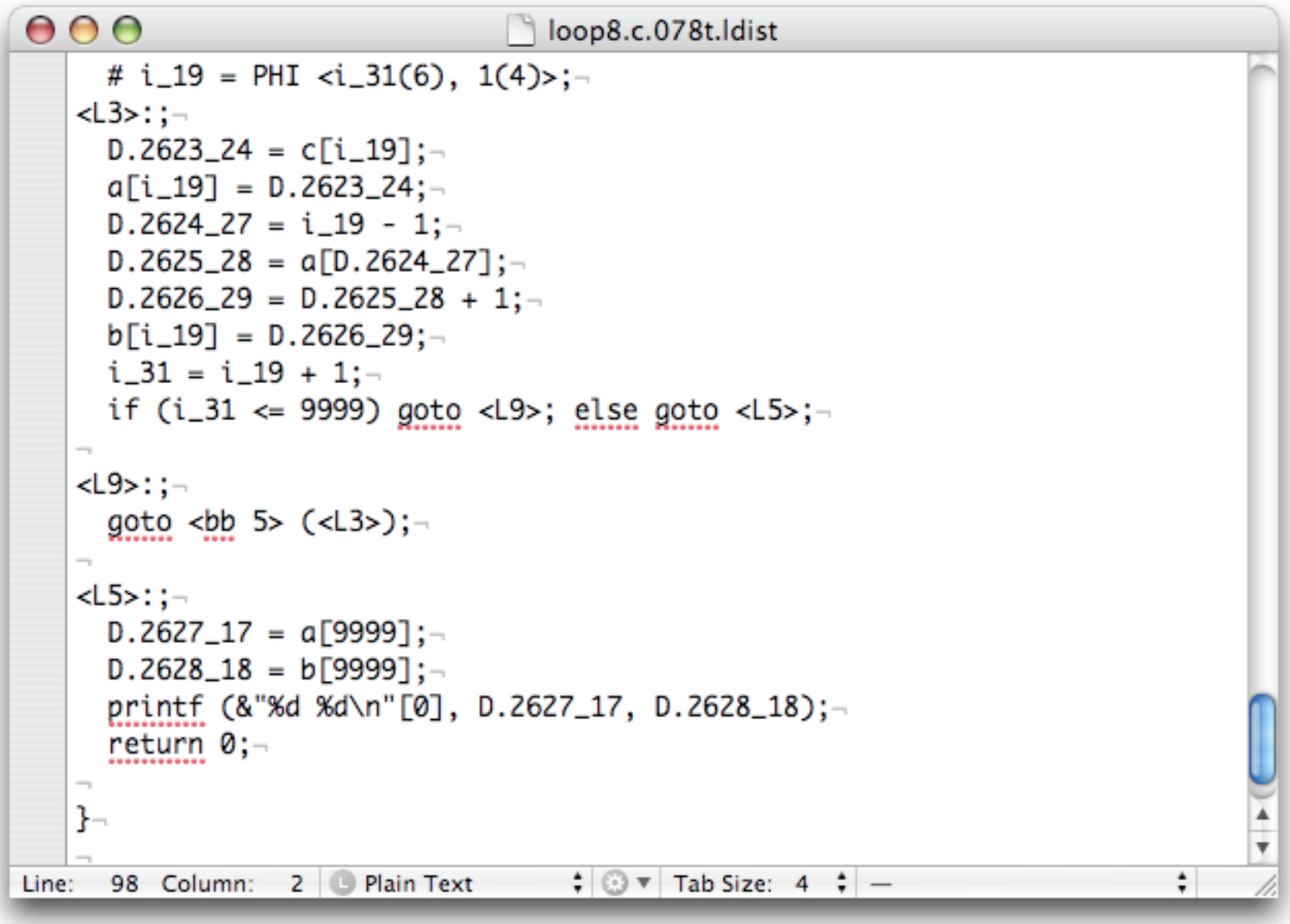
How in GCC?

- Use of existing GCC infrastructure
  - Loops + Dependences + SSA graph + GIMPLE
- New algorithms and data structures in GCC
  - Data dependence graph + SCC computation
- Manipulating GCC trees for code generation
  - Distributed loops

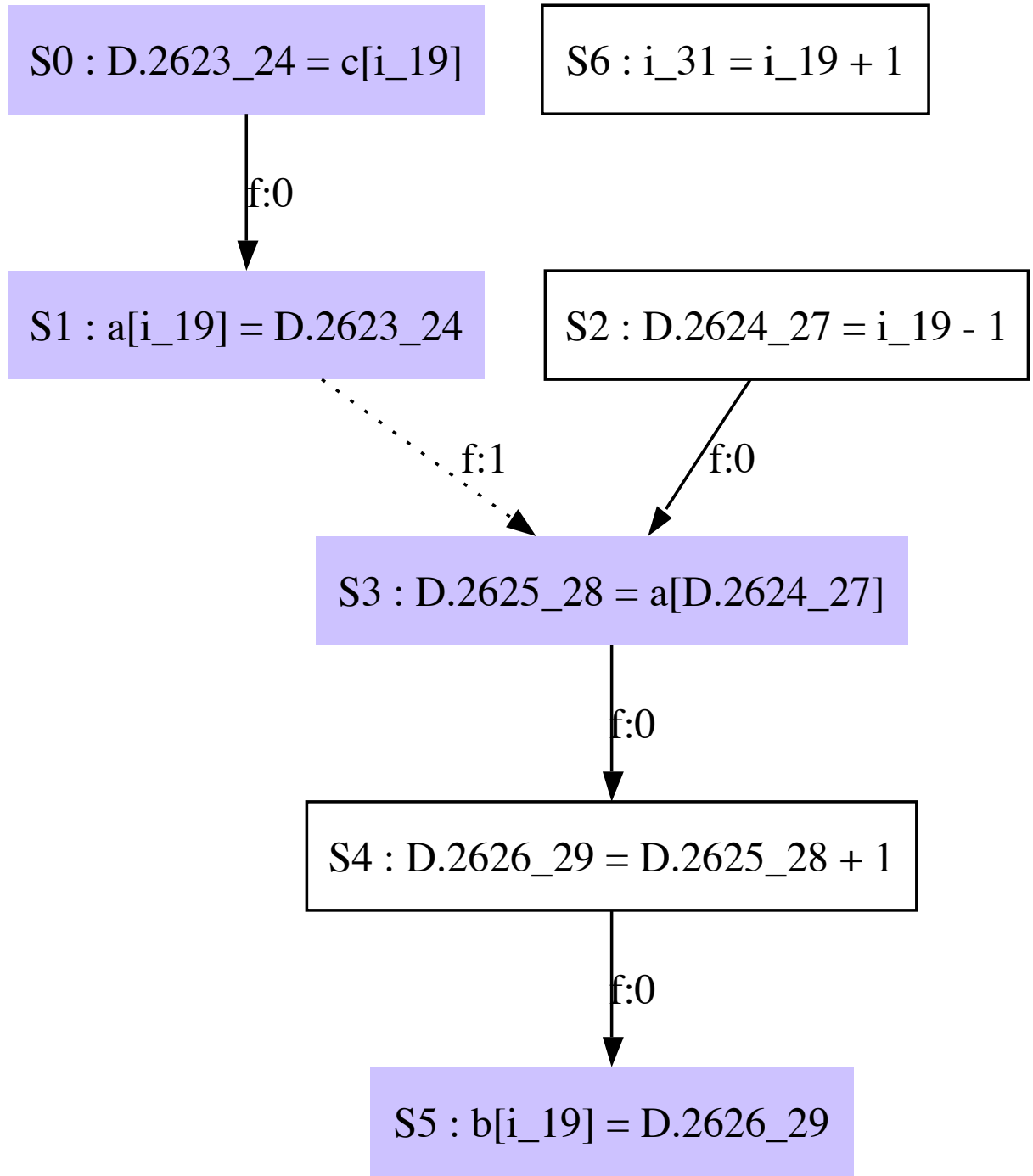
Example 1: C code



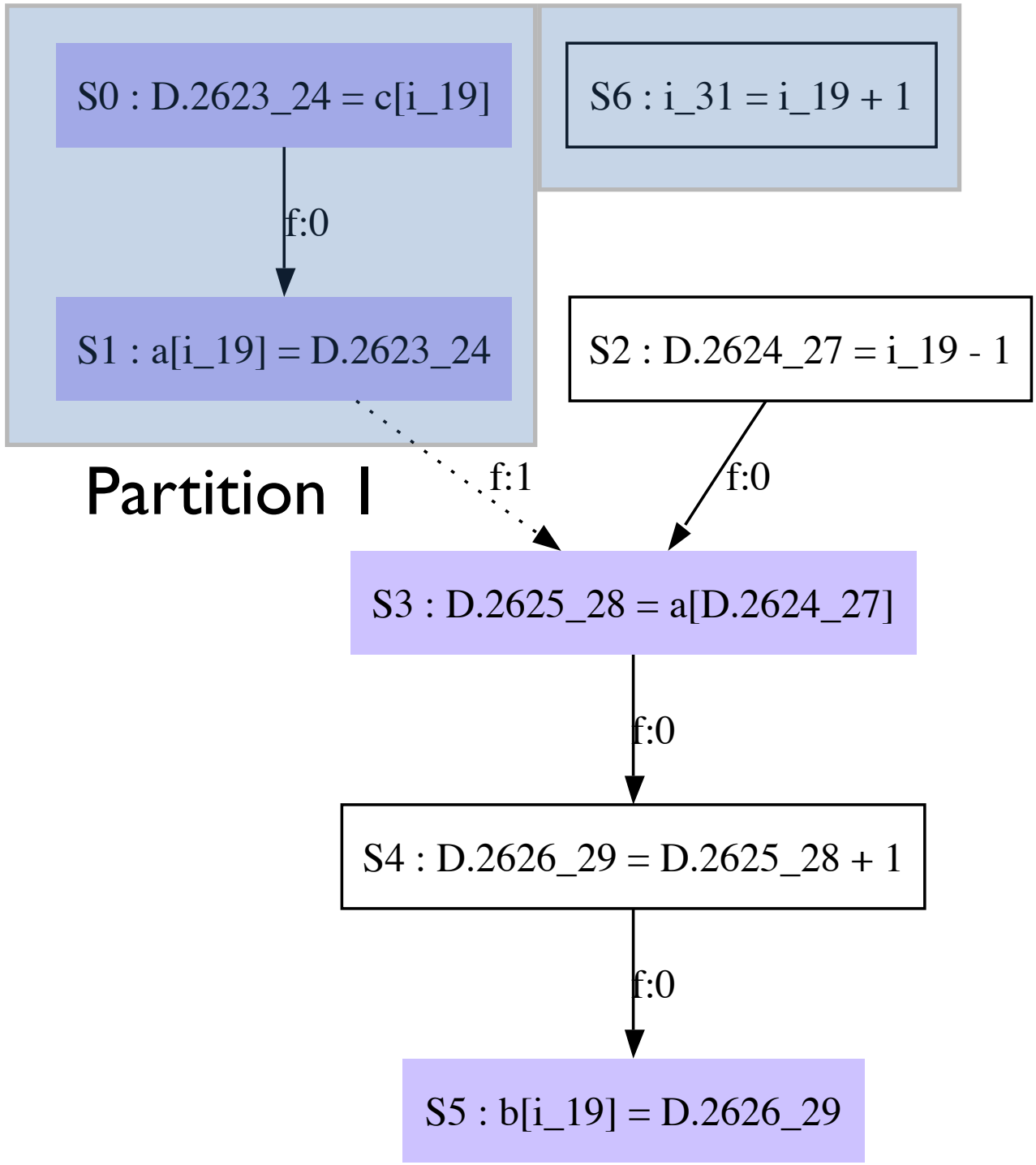
Example 1: GIMPLE dump



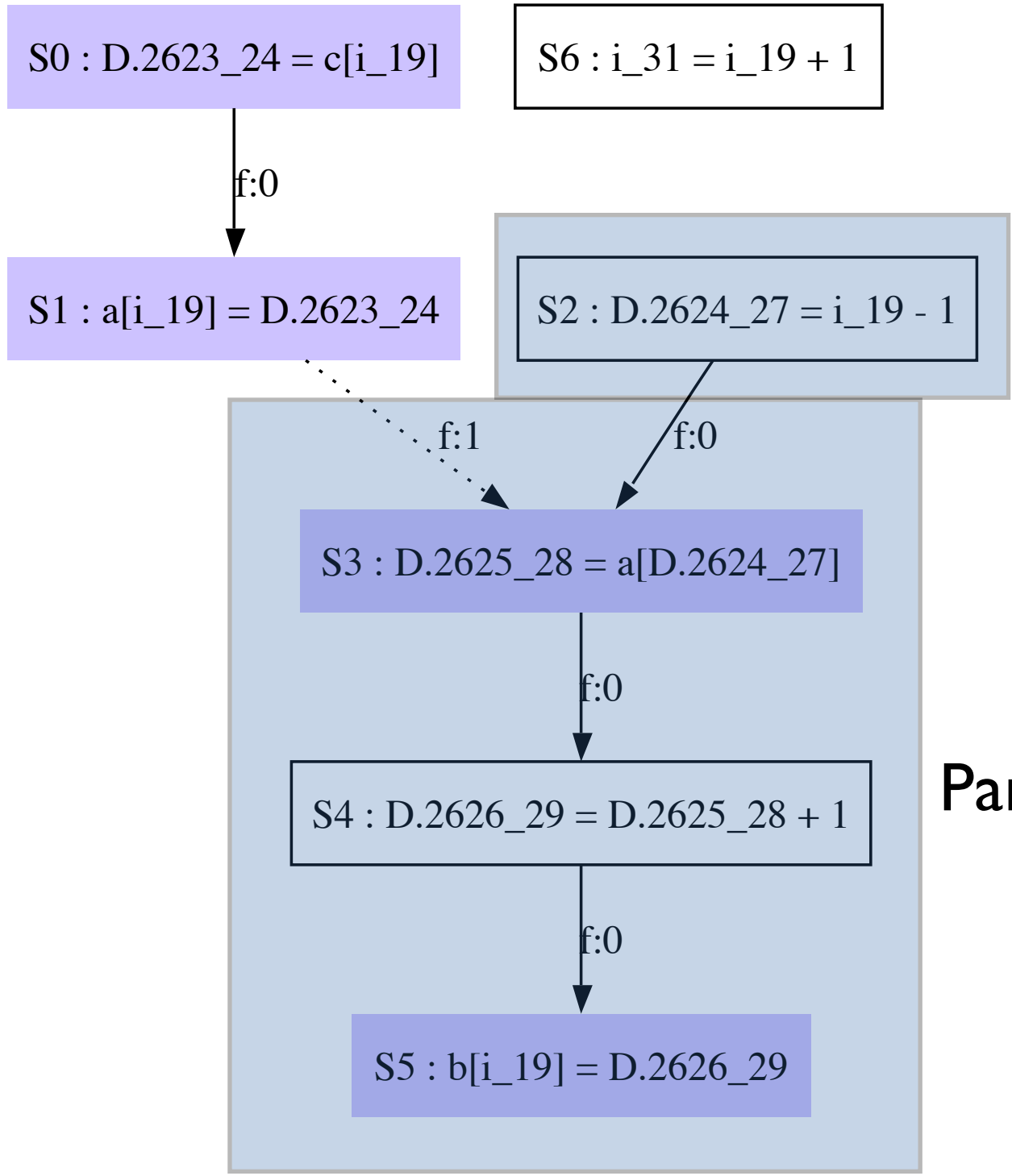
Example I : RDG



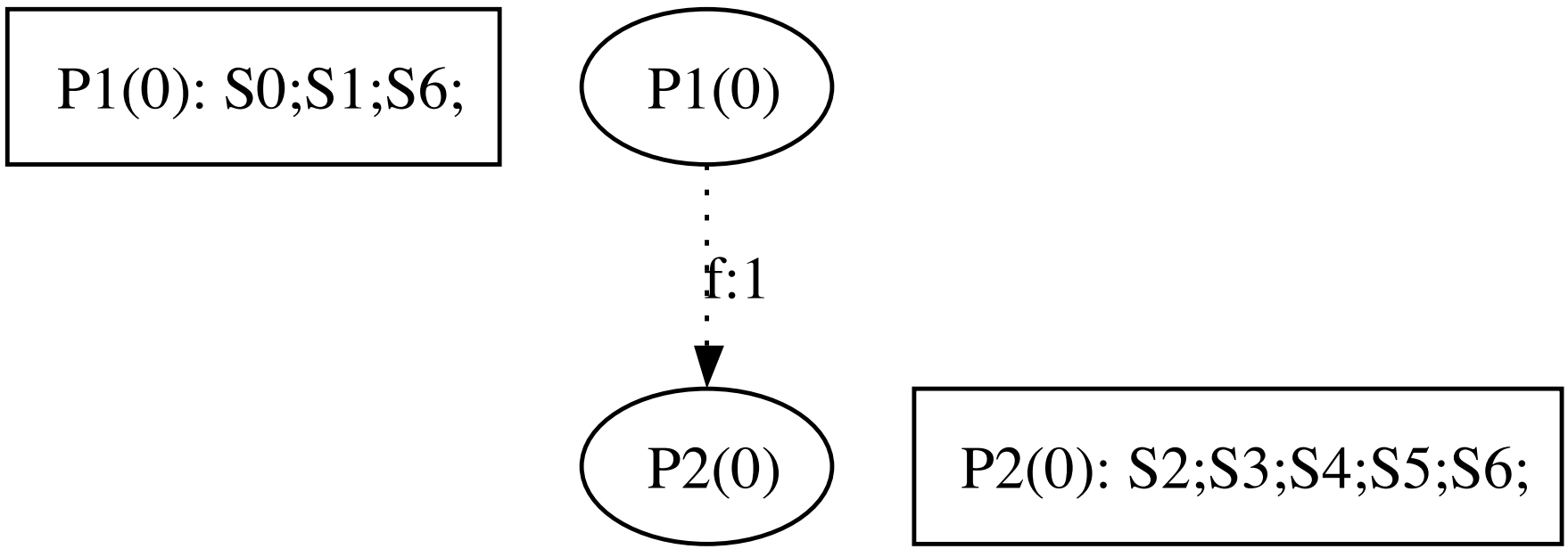
Example 1 : RDG



Example 1 : RDG

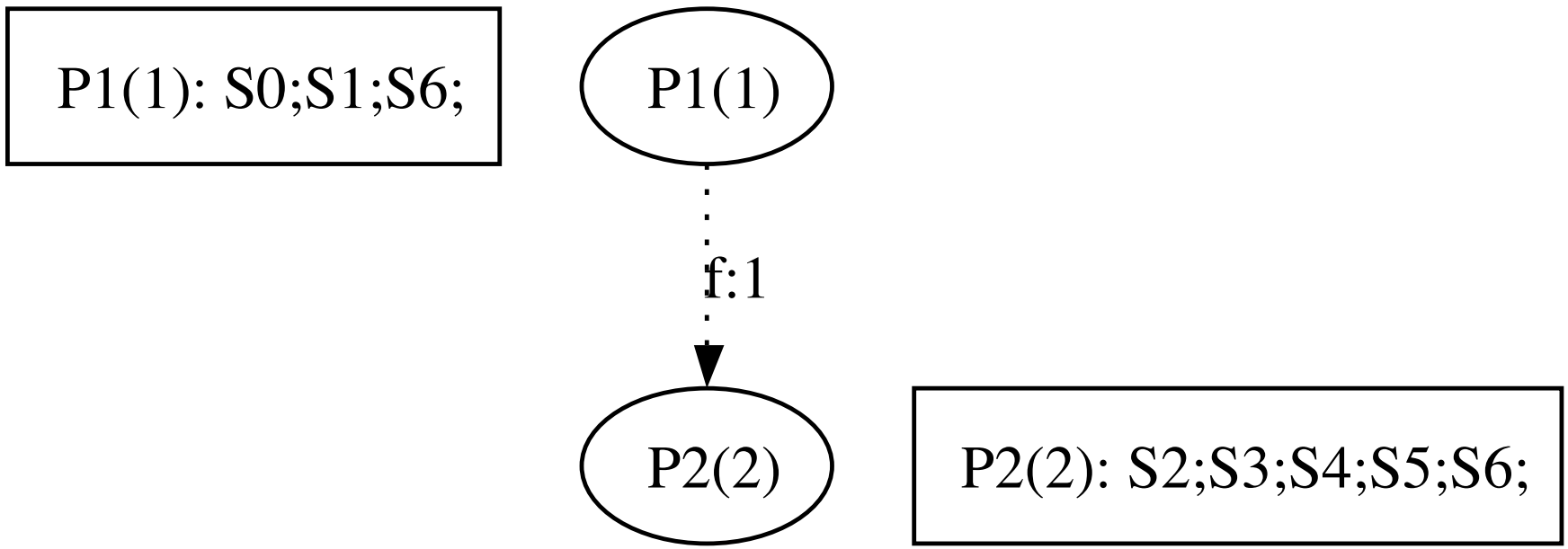


Example 1: partition graph



Example 1: SCC graph

Strongly Connected Components



Two parallel loops

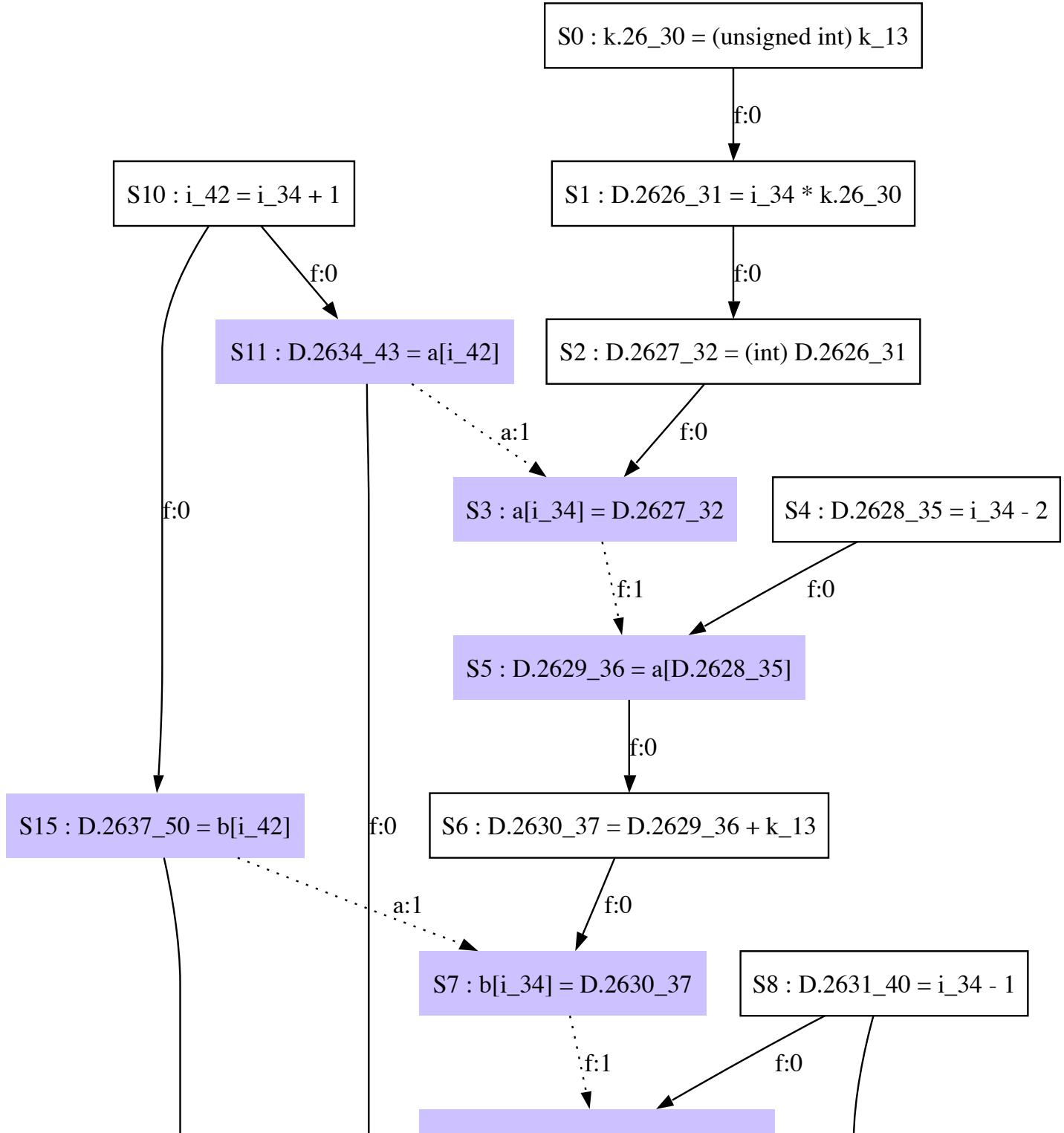
Example 2: C code

loop6.c

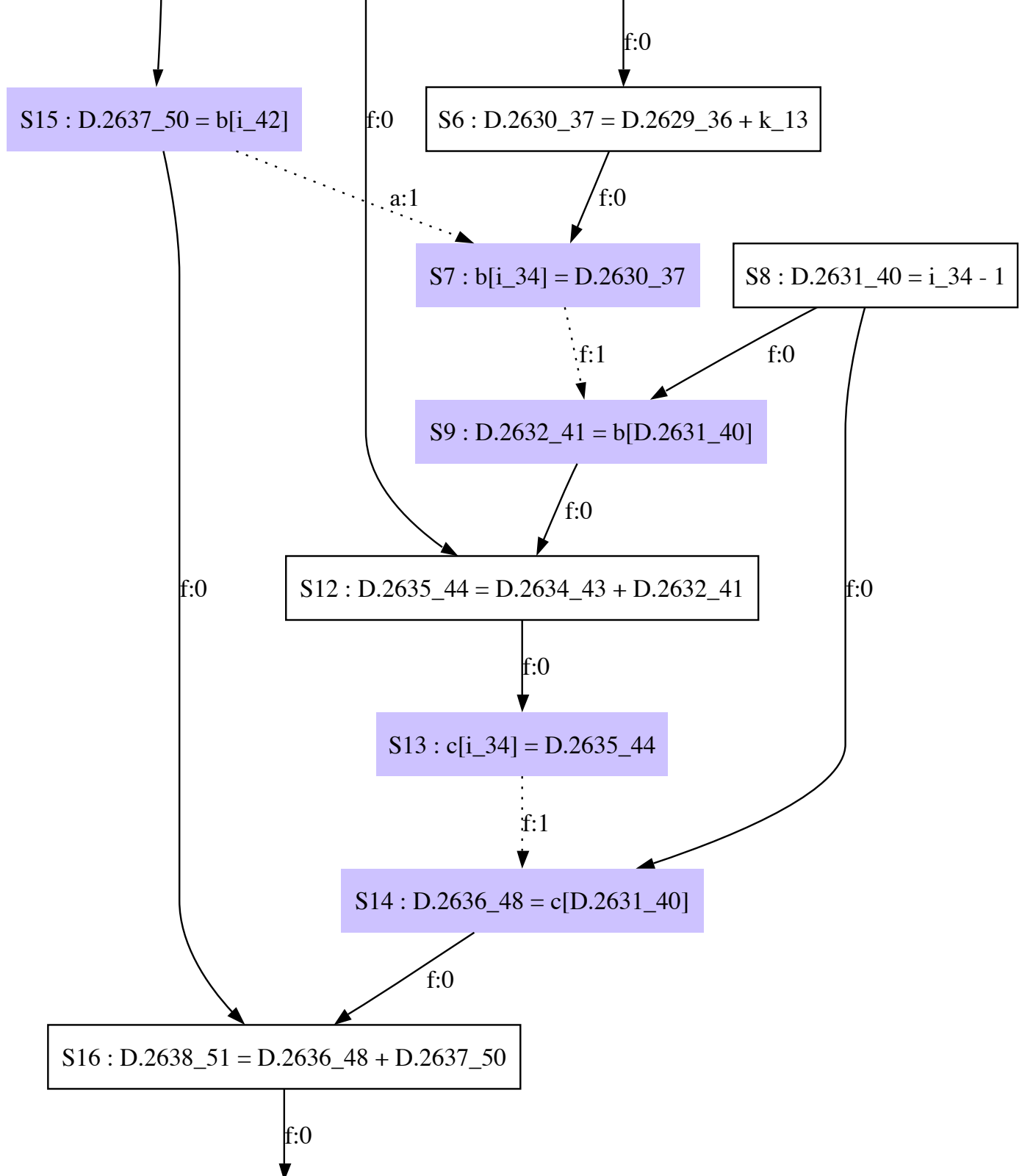
```c
#include <stdlib.h>
#include <assert.h>
#include <stdio.h>
#define N 10000
int main (int argc, char const* argv[])
{
    unsigned int i;
    int a[N], b[N], c[N], d[N];
    int k, z;
    assert (argc > 1);
    k = atoi (argv[1]); a[0] = k; a[3] = k * 2;
    for (i = 2; i < (N-1); i ++) {
        a[i] = k * i;
        b[i] = a[i-2] + k;
        c[i] = b[i-1] + a[i+1];
        d[i] = c[i-1] + b[i+1] + k + i;
    }
    printf ("%d %d %d %d\n", a[N-2], b[N-1], c[N-2], d[N-2]);
    return 0;
}
```

Line: 15 Column: 32 C Tab Size: 4 main

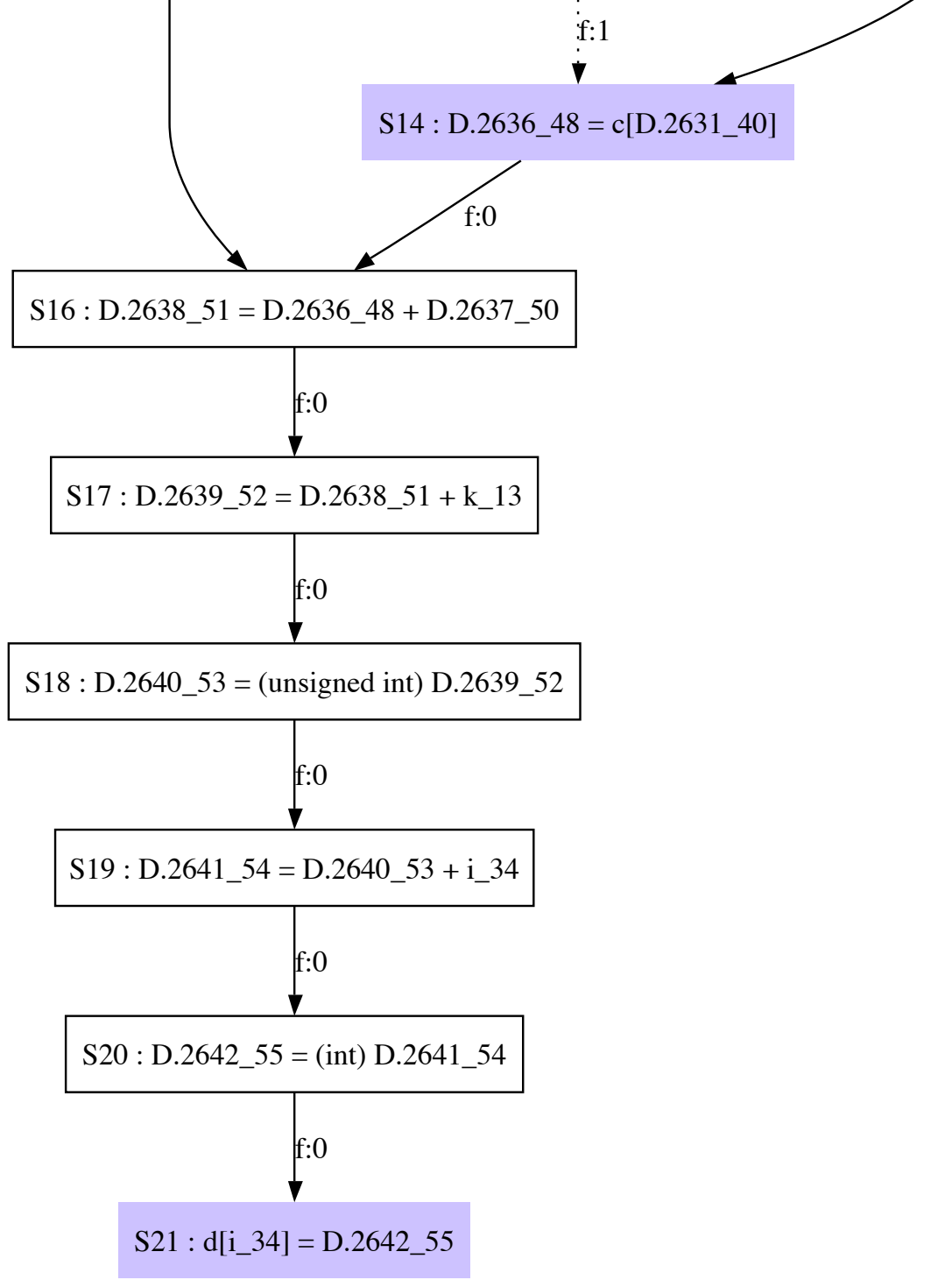
Example 2: RDG (I /3)



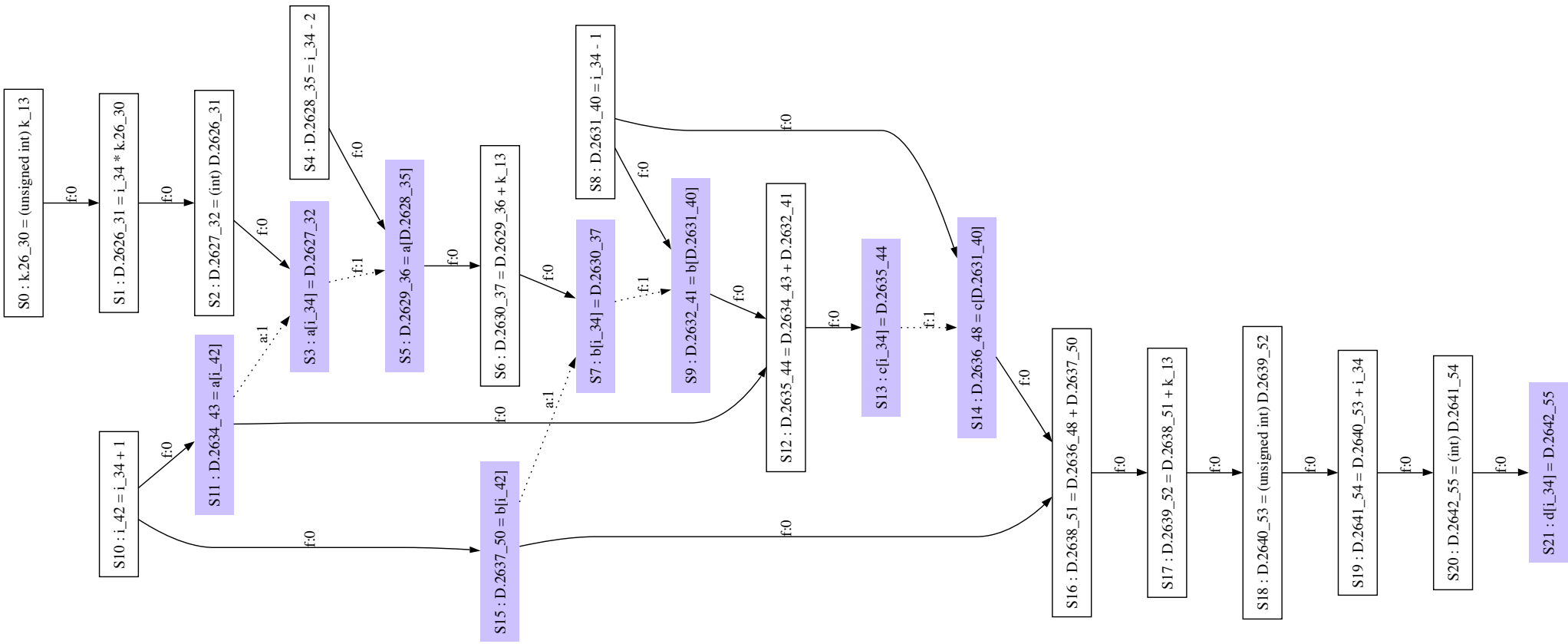
Example 2: RDG (2/3)



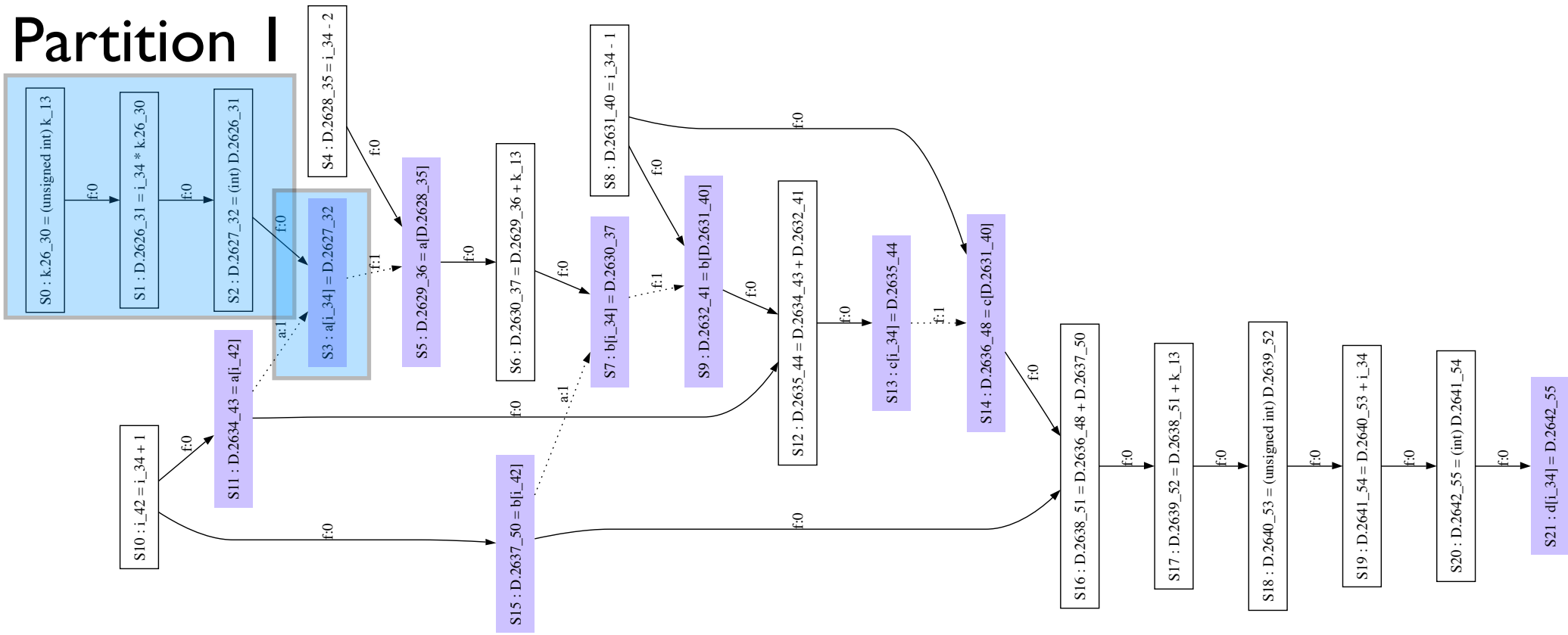
Example 2: RDG (3/3)



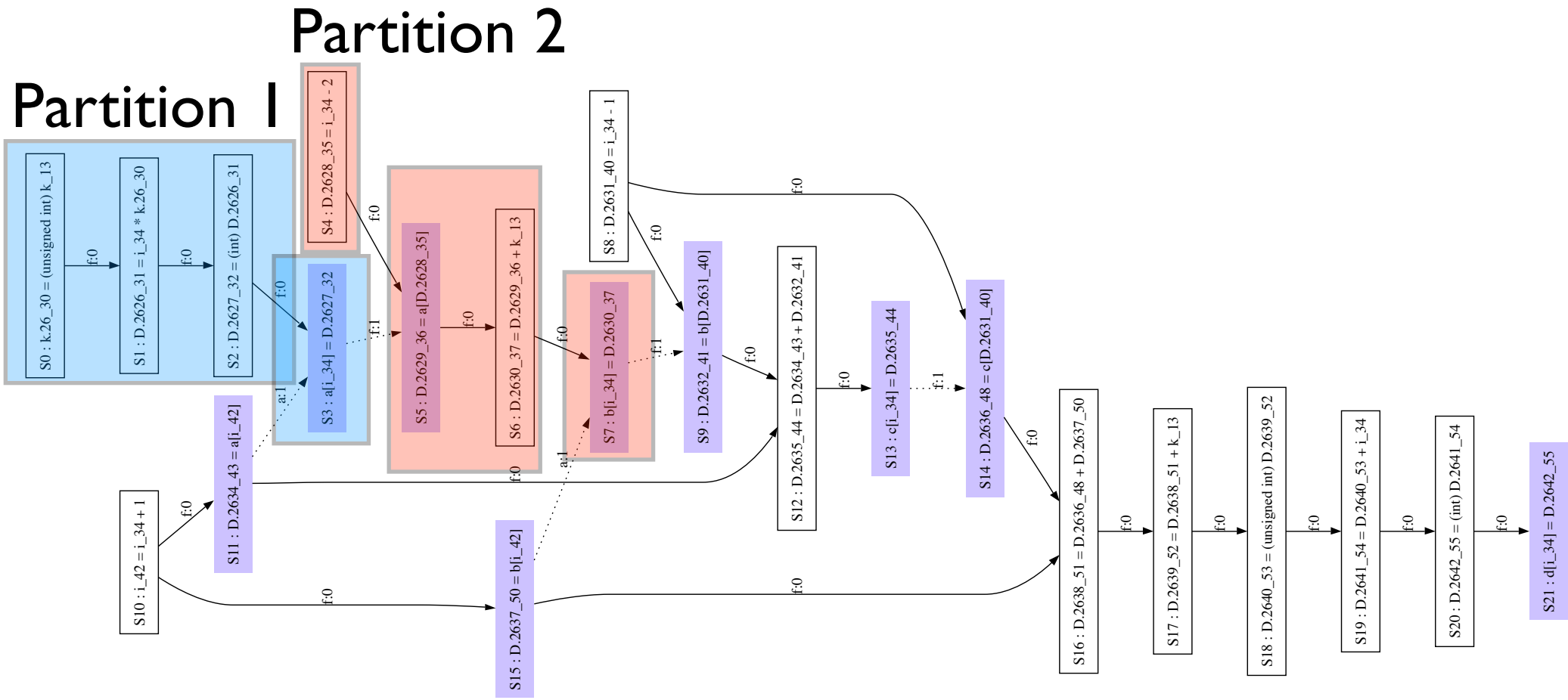
Exemple 2: RDG



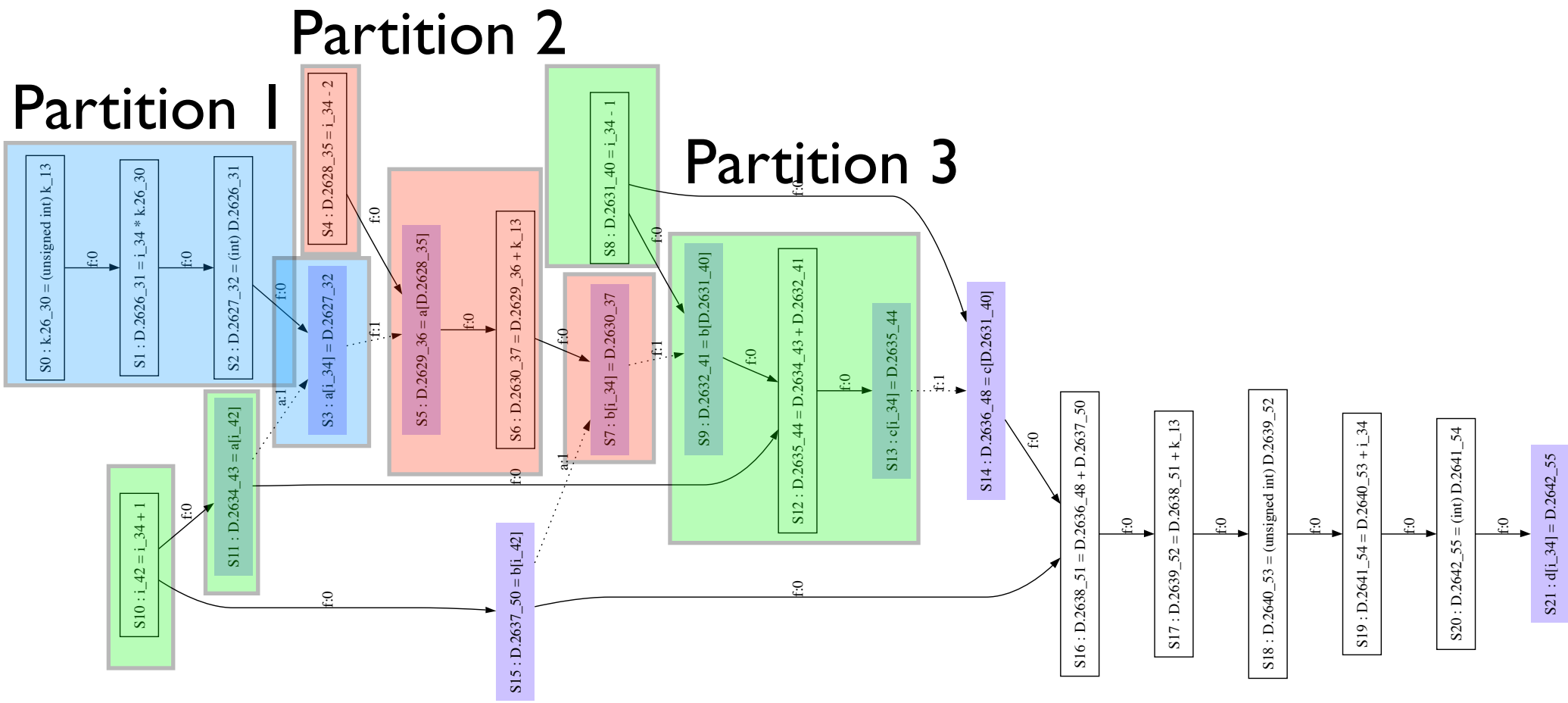
Exemple 2: RDG



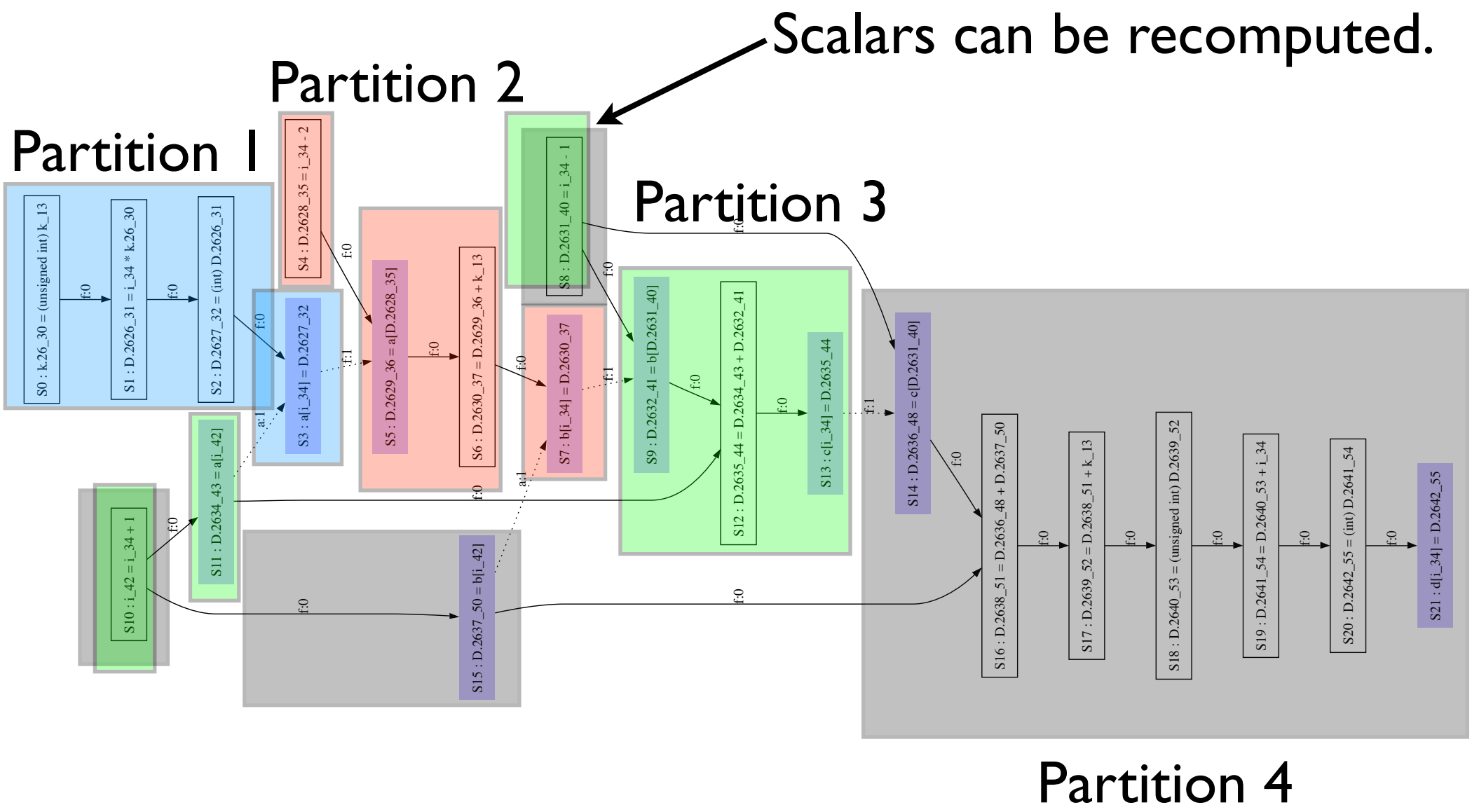
Exemple 2: RDG



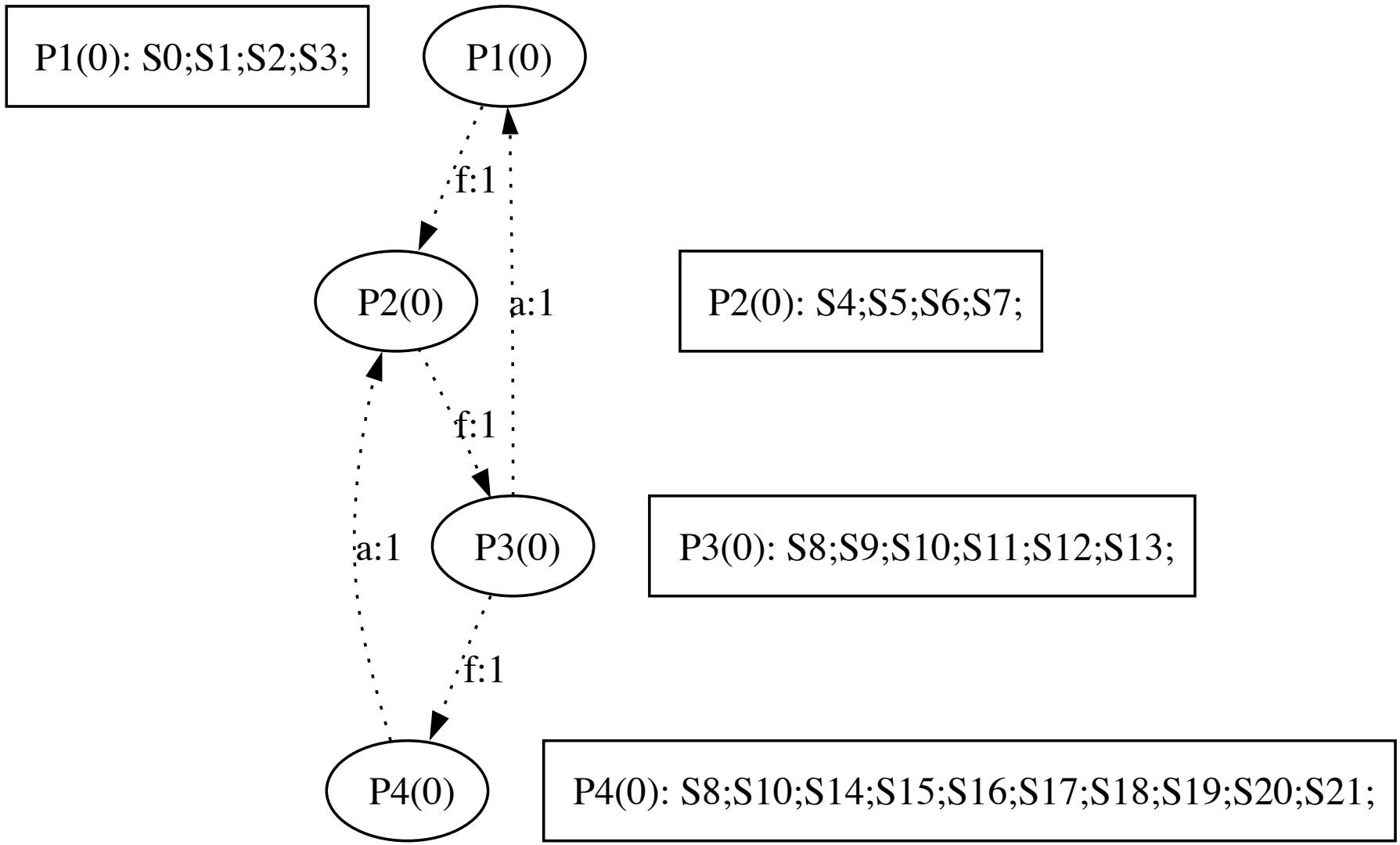
Exemple 2: RDG



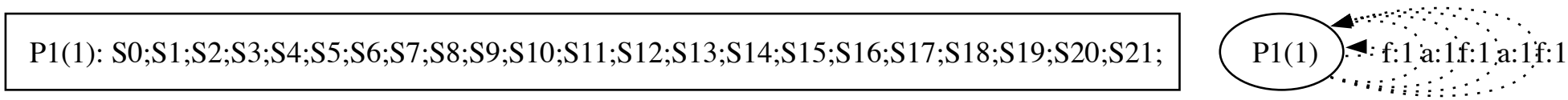
Exemple 2: RDG



Example 2: partition graph



Example 2: SCC graph



One sequential loop

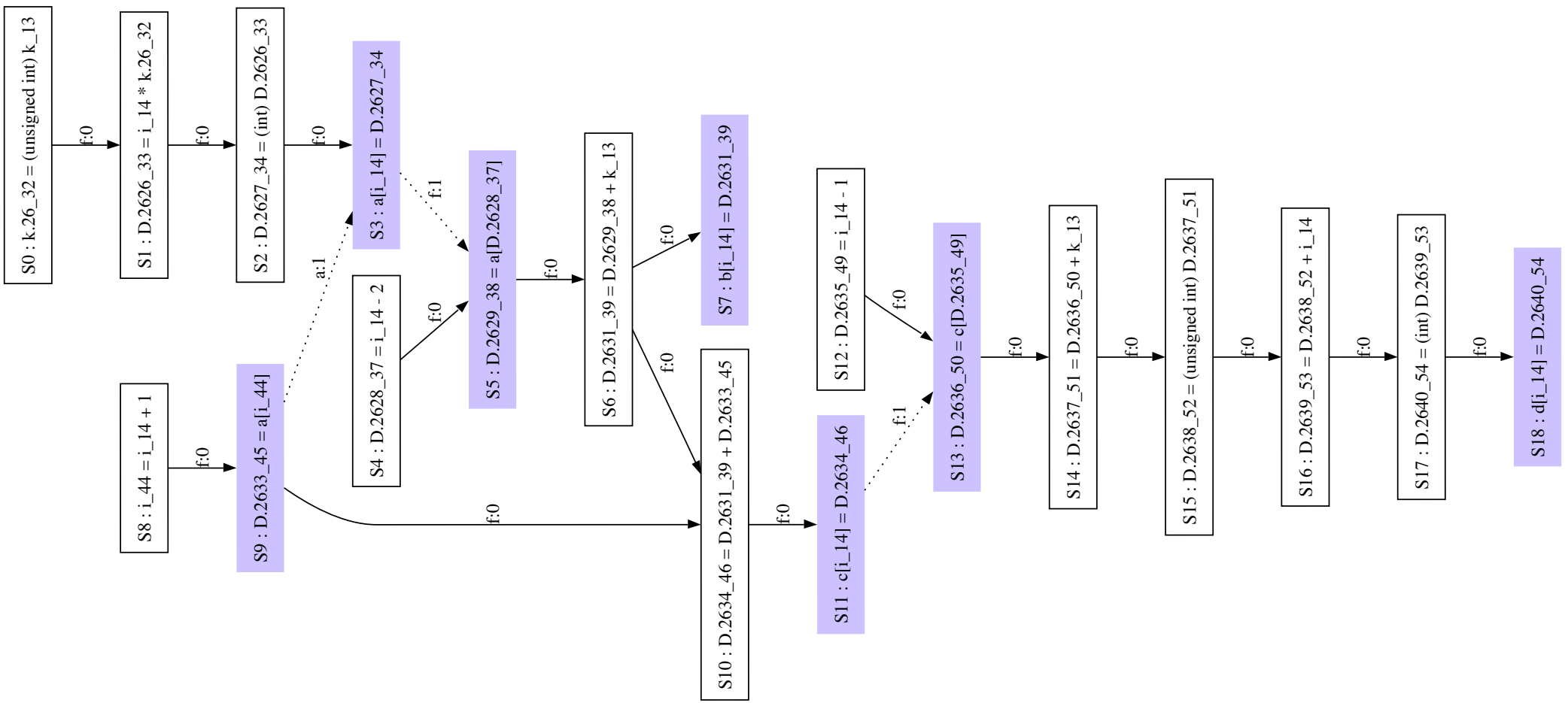
Example 3: C code

loop1.c

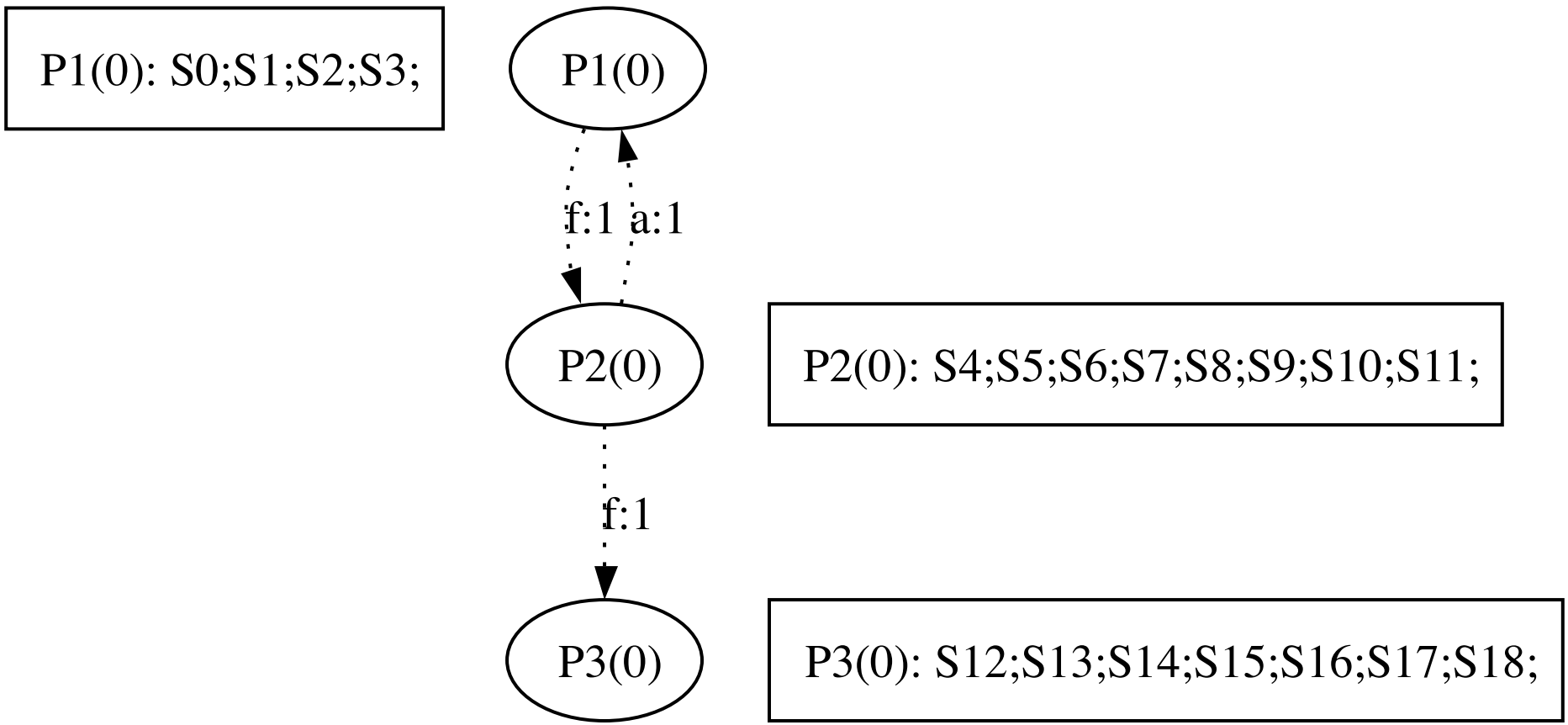
```c
int
main (int argc, char const* argv[])
{
    unsigned int i;
    int a[N], b[N], c[N], d[N];
    int k;

    assert (argc > 1);
    k = atoi (argv[1]);
    a[0] = k; a[3] = k*2;
    c[1] = k+1;
    for (i = 2; i < (N-1); i ++)
        {
            a[i] = k * i; /* S1 */
            b[i] = a[i-2] + k; /* S2 */
            c[i] = b[i] + a[i+1]; /* S3 */
            d[i] = c[i-1] + k + i; /* S4 */
        }
    printf ("%d %d %d %d\n", a[N-2], b[N-1], c[N-2], d[N-2]);
    return 0;
}
```

Line: 23 Column: 10 C Tab Size: 4 main

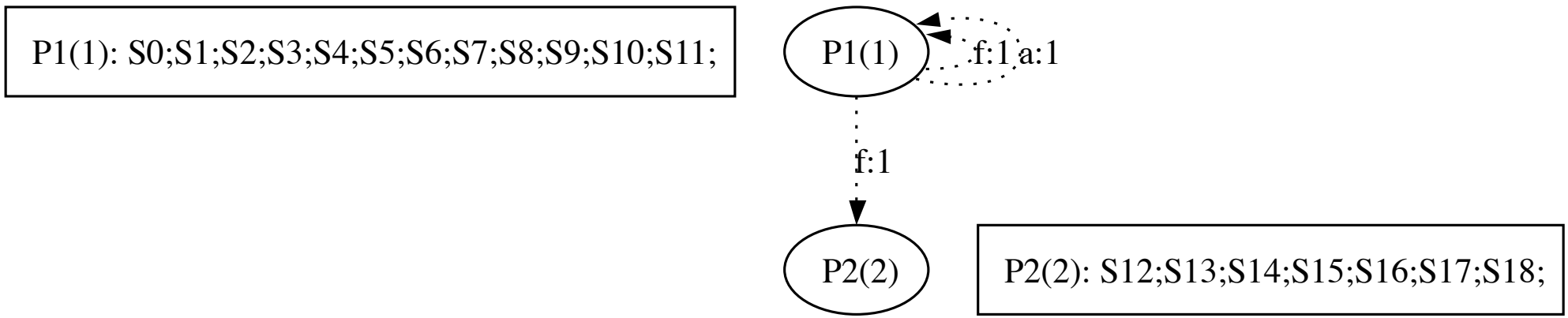
Exemple 3: RDG



Exemple 3: partition graph



Exemple 3: SCC graph

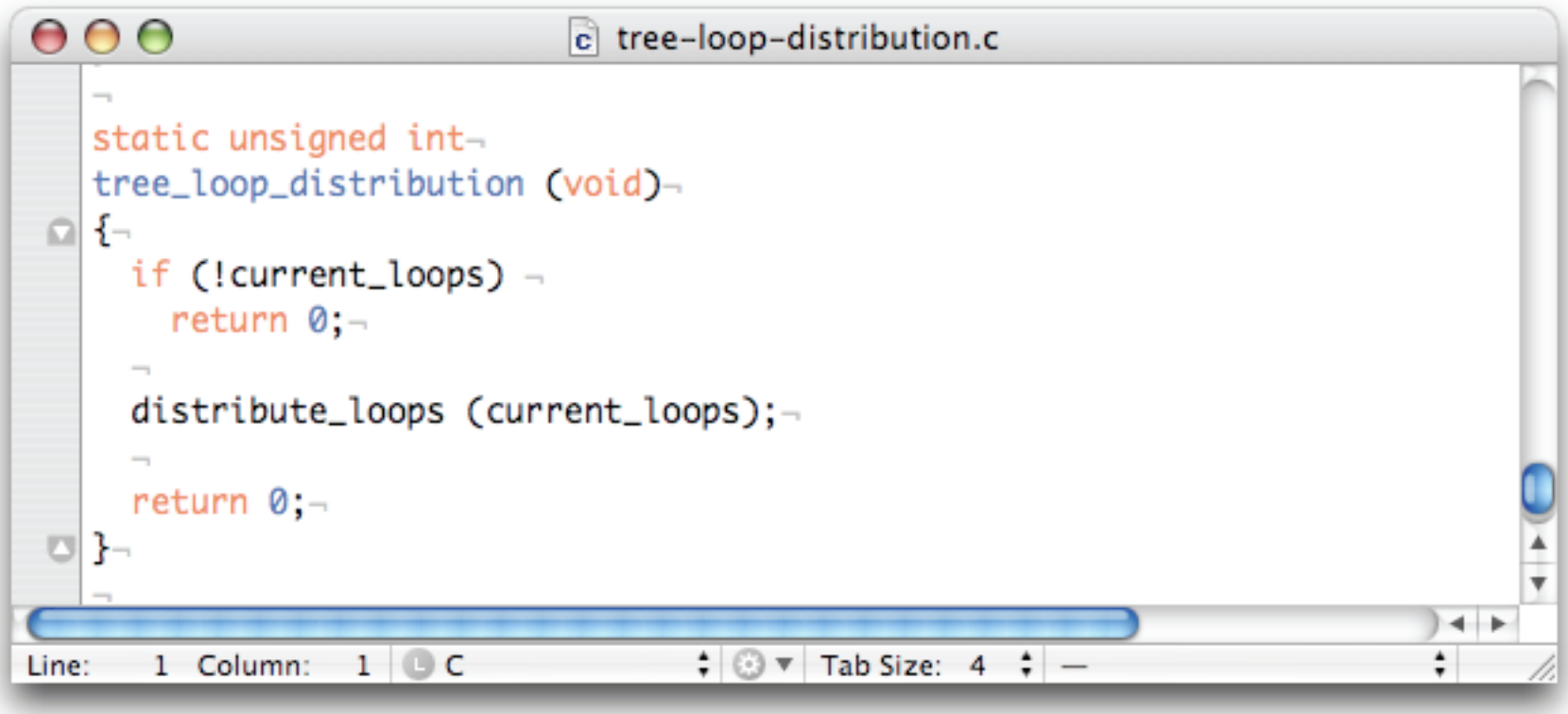


One sequential loop
One parallel loop

Sketch of the implementation

- ☝ The code presented here is simplified
- ☝ This is not part of GCC now
- The complete patch can be found on:
  - http://www.hipeac.net

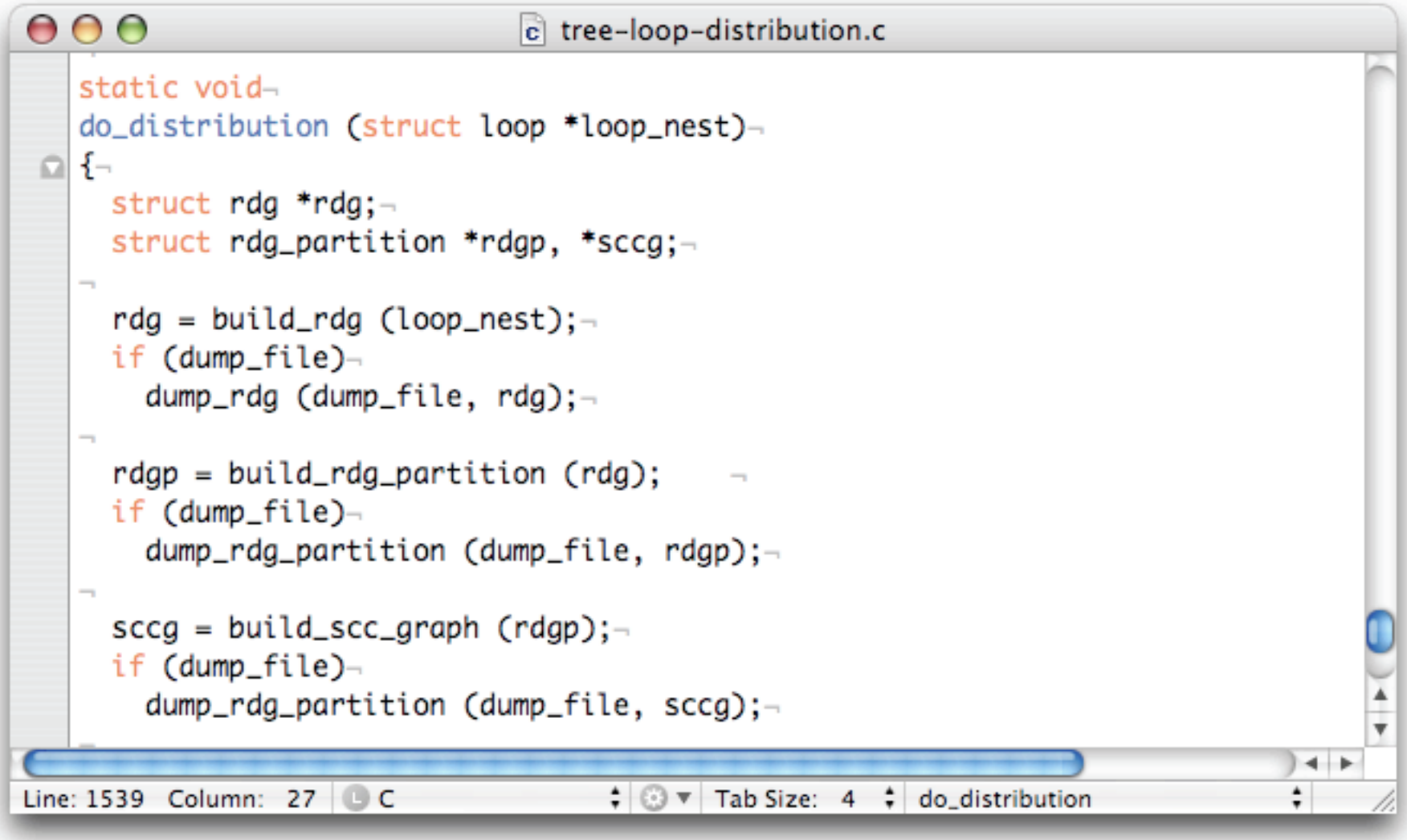
tree_loop_distribution ()



tree_loop_distribution is called for each function

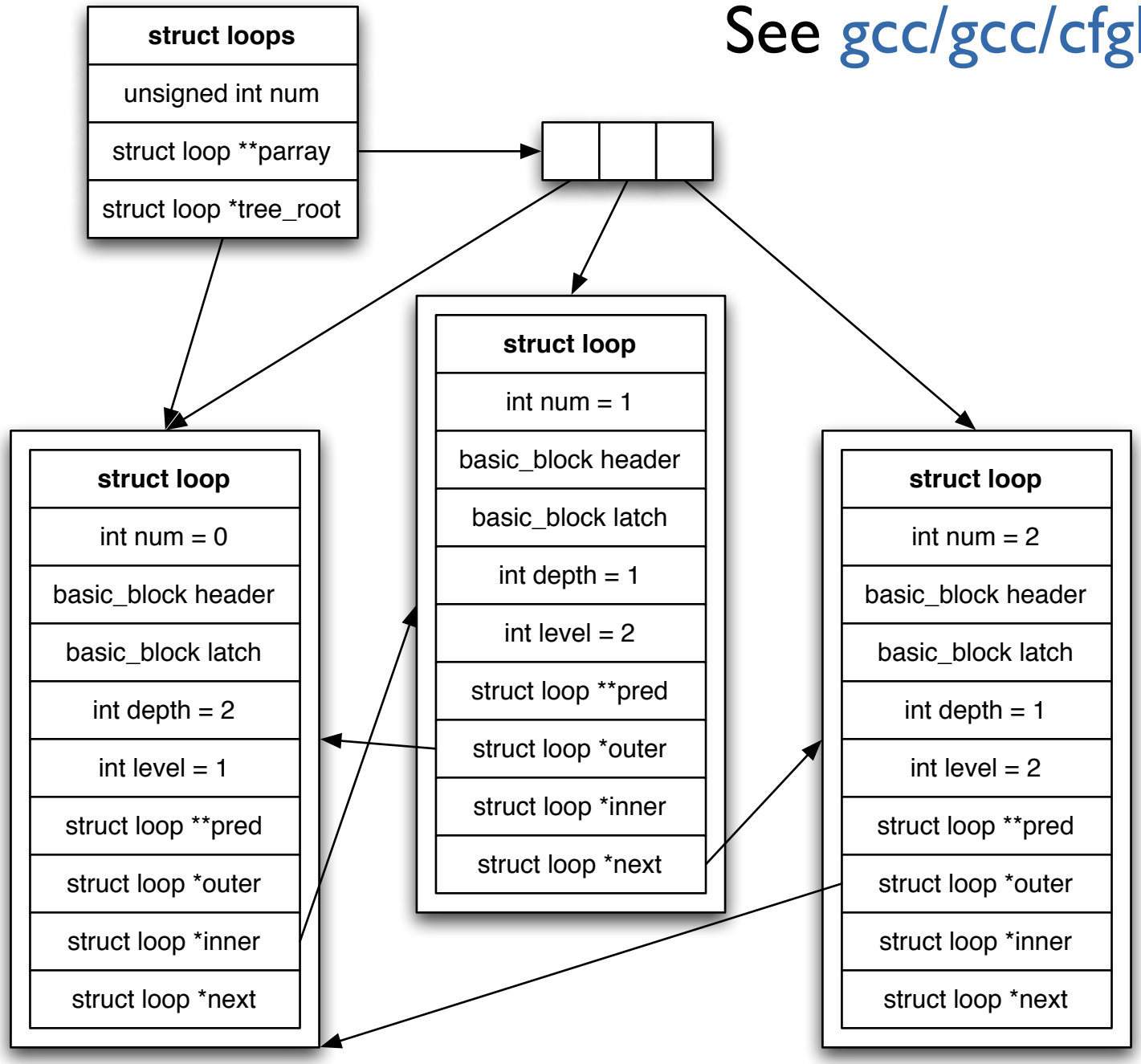
current_loops contains the loops of the function

Building dependence graphs



Loop structure of GCC

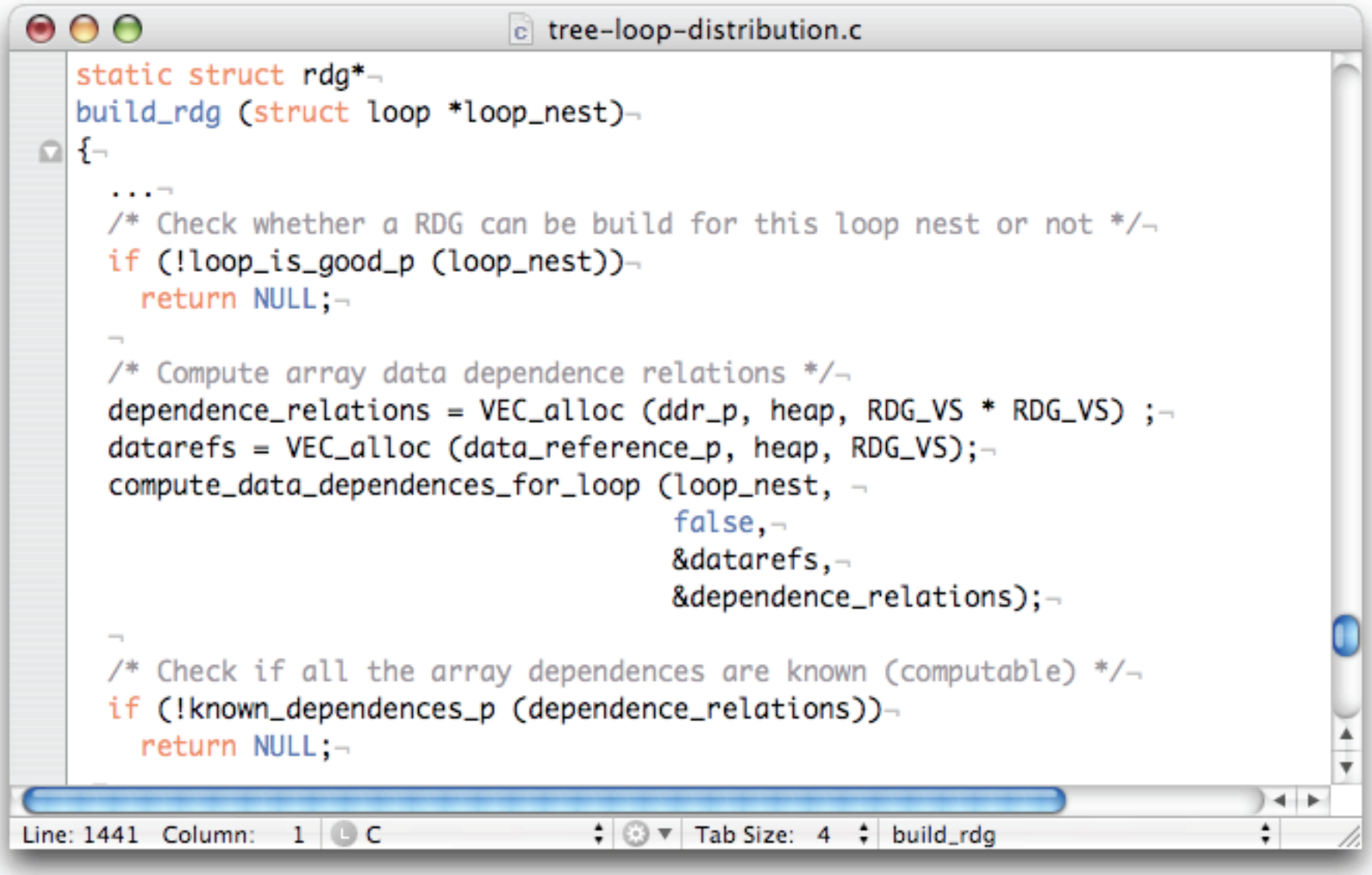
See gcc/gcc/cfgloop.h



RDG computation

- One vertex per statement of the loop body
- One edge for each scalar dependence (SSA)
- One edge for each data dependence (SCEV)

build_rdg () [1/2]



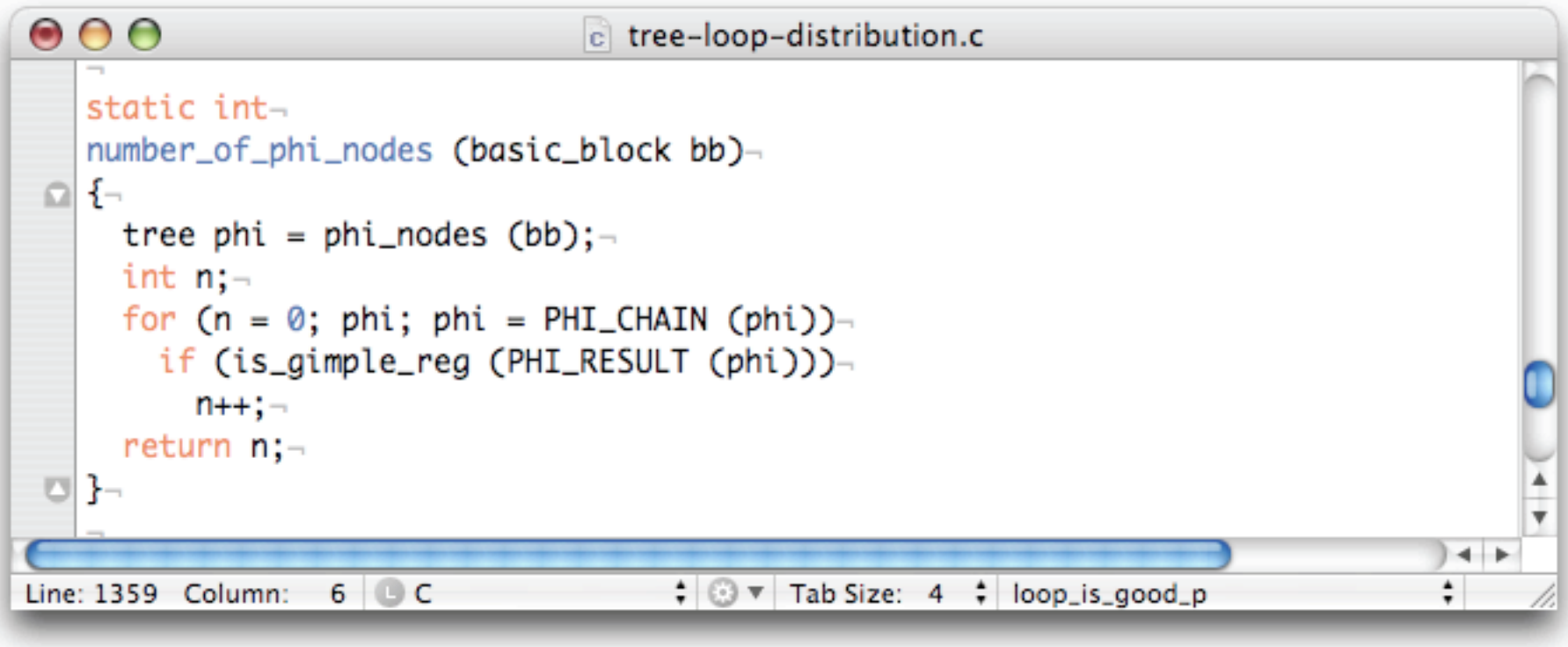
VEC_alloc () : gcc/gcc/vec.h

compute_data_dependences_for_loop () : gcc/gcc/tree-data-ref.c

loop_is_good_p ()

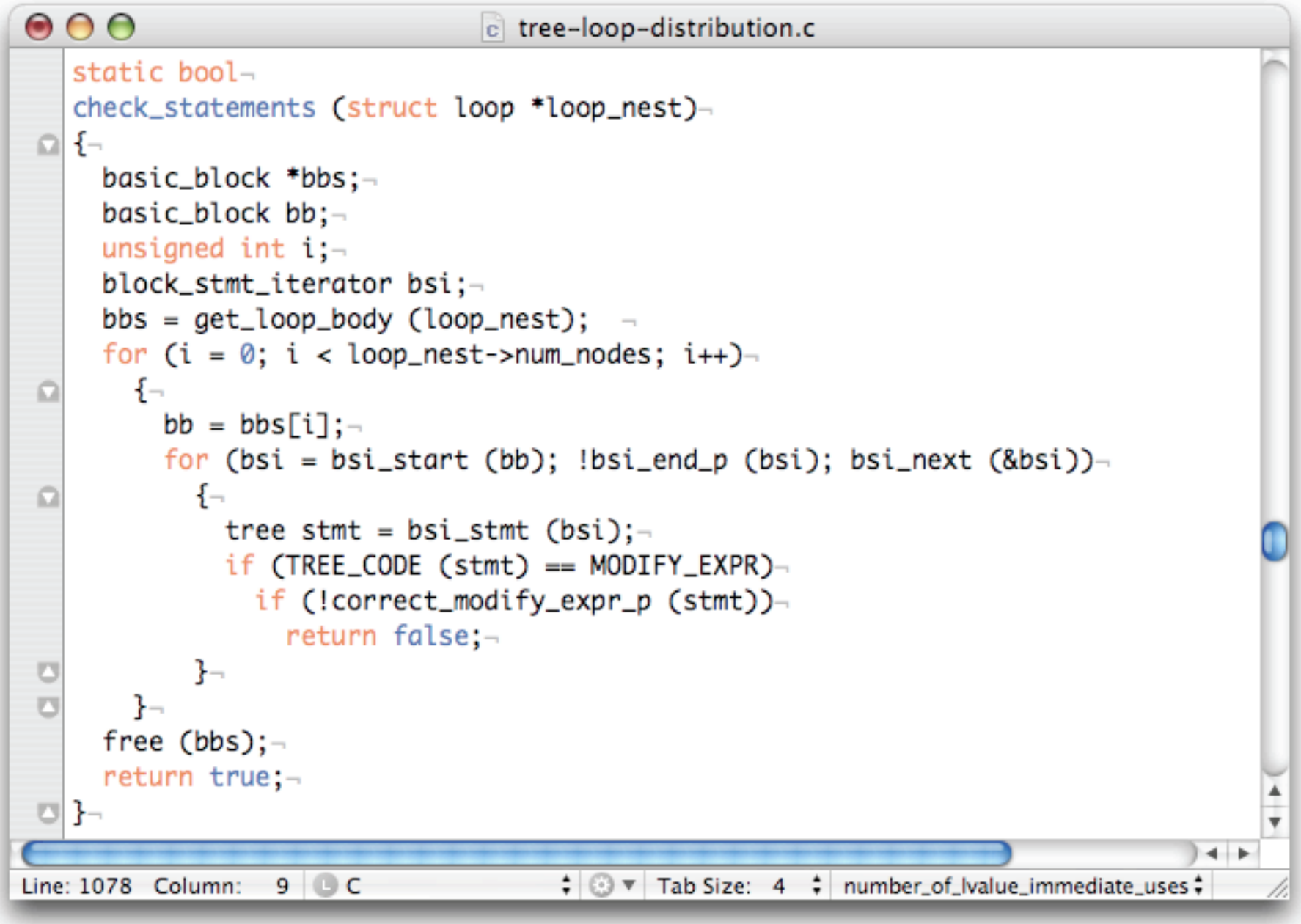
get_loop_exit_condition () : gcc/gcc/tree-scalar-evolution.c

number_of_phi_nodes ()

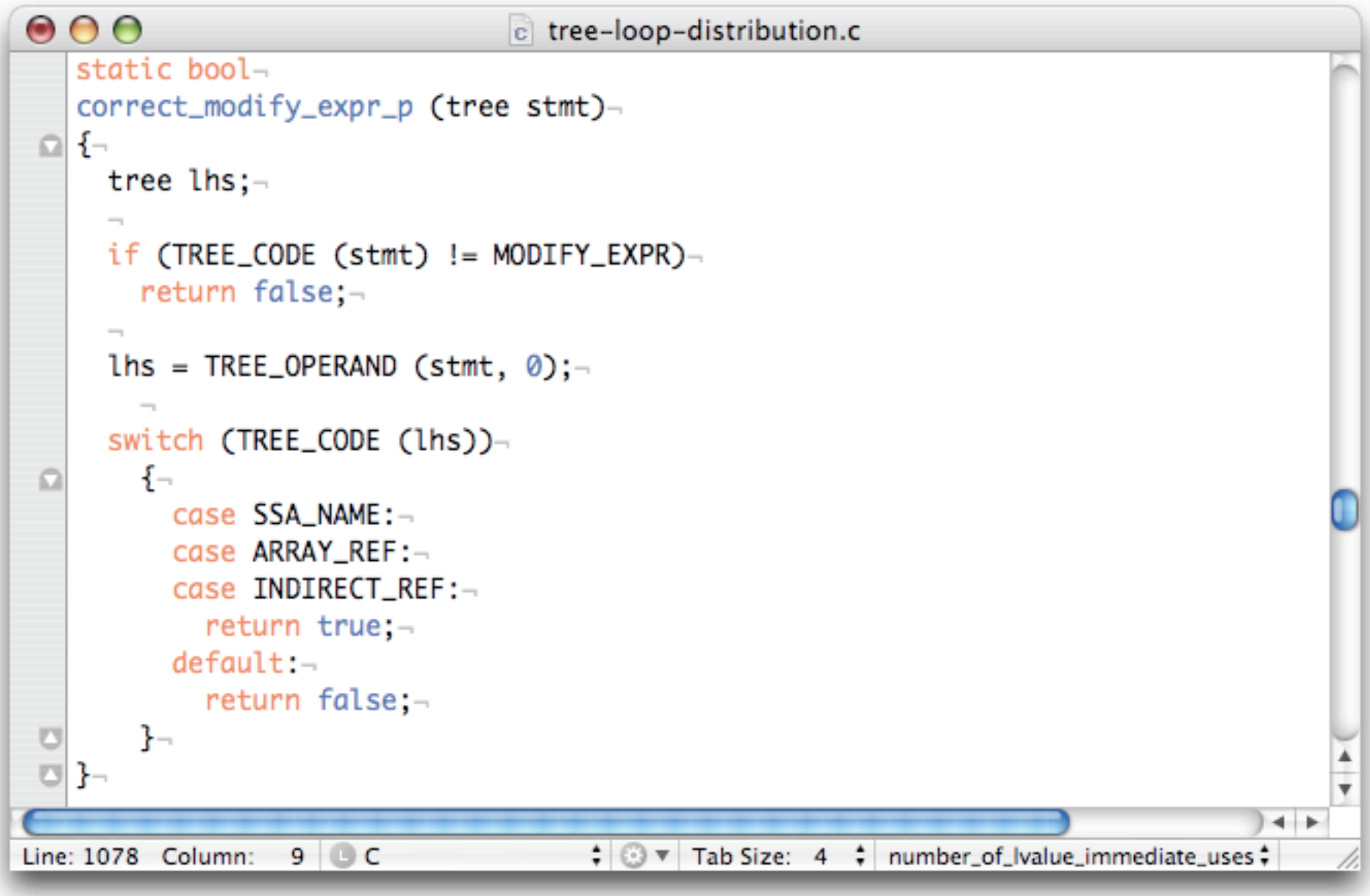


phi_nodes () : gcc/gcc/tree-flow-inline.h
PHI_CHAIN () : gcc/gcc/tree.h
is_gimple_reg () : gcc/gcc/tree-gimple.c

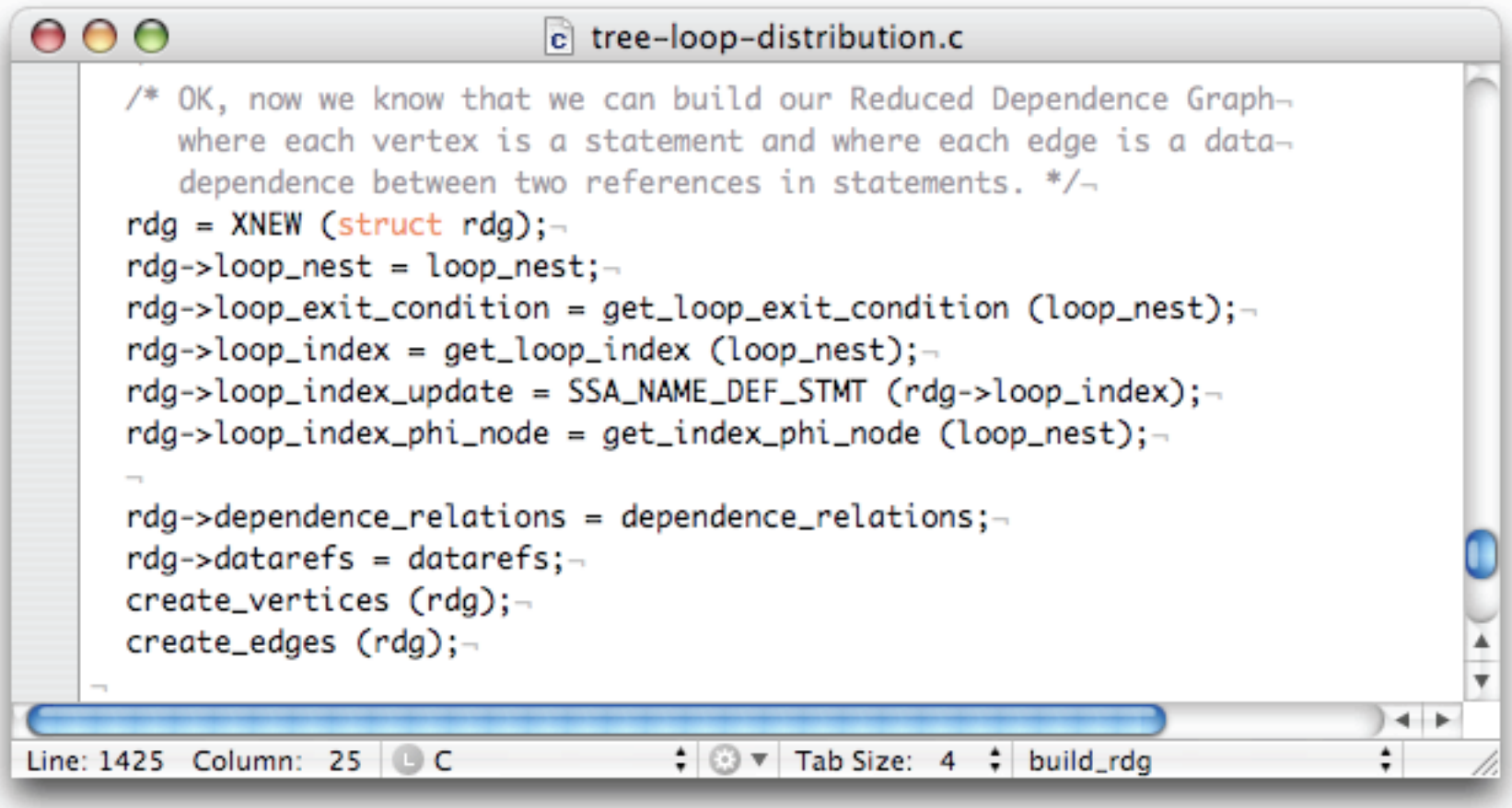
check_statements ()



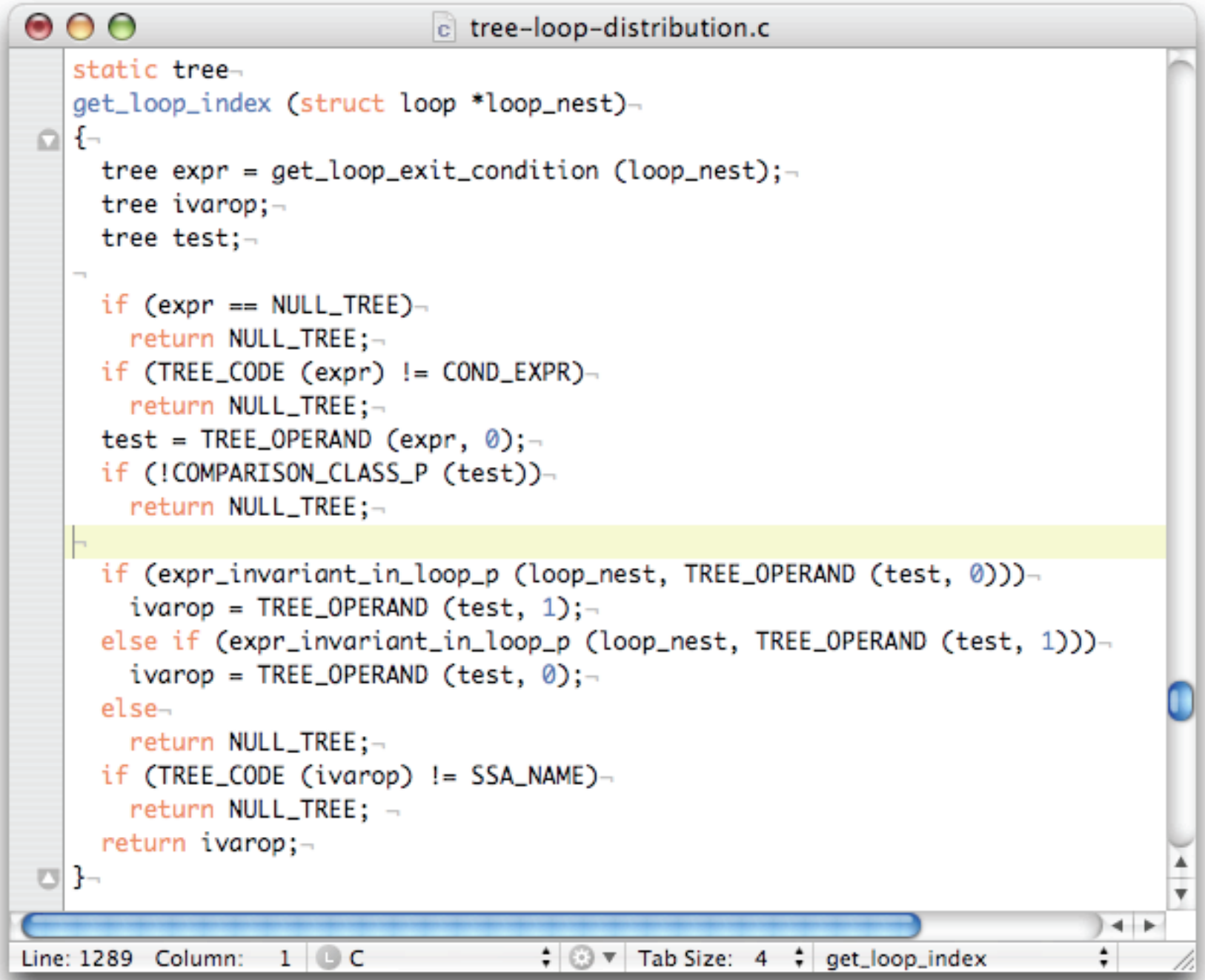
correct_modify_expr_p ()



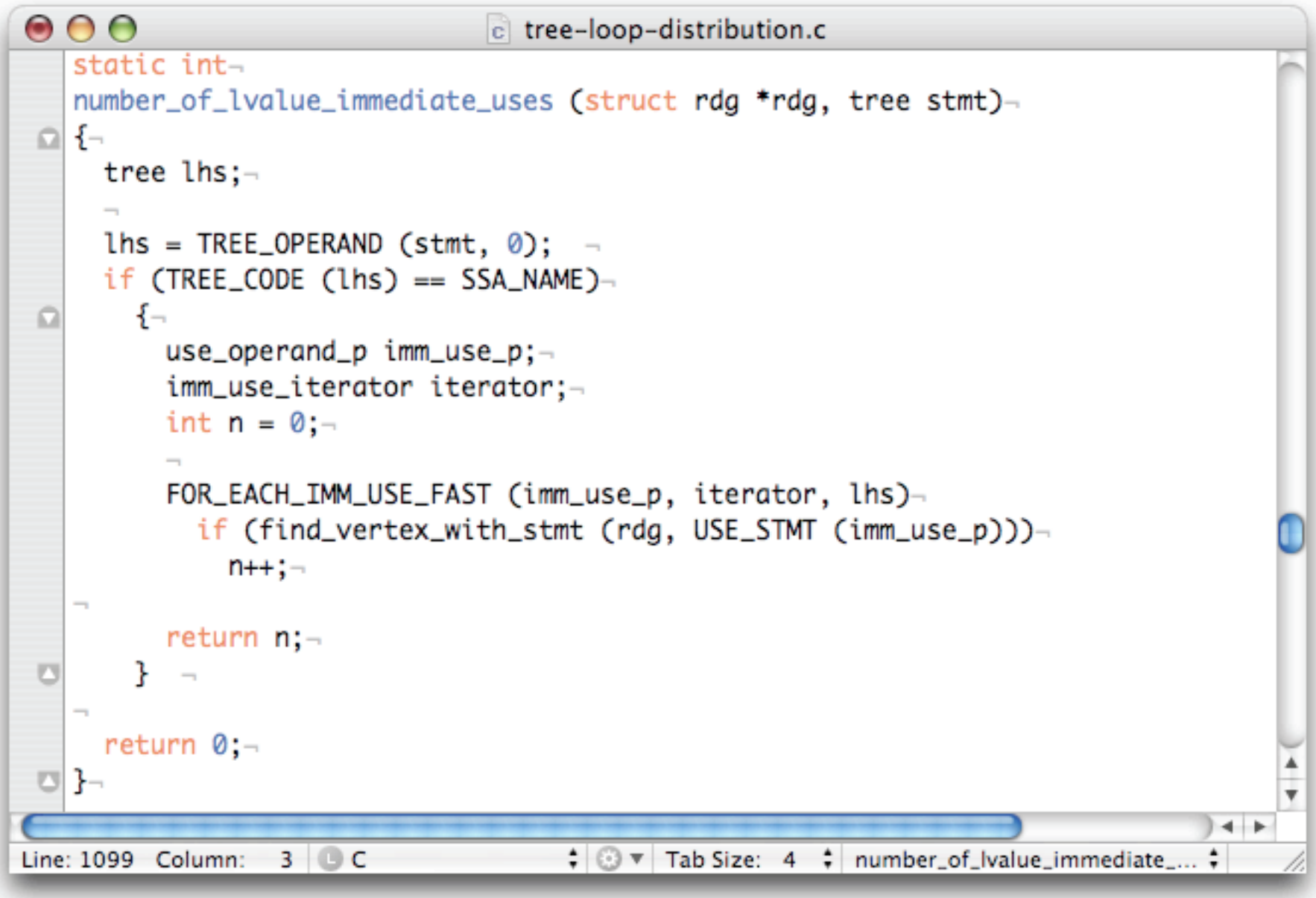
build_rdg () [2/2]


tree-loop-distribution.c
/* OK, now we know that we can build our Reduced Dependence Graph
where each vertex is a statement and where each edge is a data
dependence between two references in statements. */
rdg = XNEW (struct rdg);
rdg->loop_nest = loop_nest;
rdg->loop_exit_condition = get_loop_exit_condition (loop_nest);
rdg->loop_index = get_loop_index (loop_nest);
rdg->loop_index_update = SSA_NAME_DEF_STMT (rdg->loop_index);
rdg->loop_index_phi_node = get_index_phi_node (loop_nest);
rdg->dependence_relations = dependence_relations;
rdg->datarefs = datarefs;
create_vertices (rdg);
create_edges (rdg);
Line: 1425 Column: 25
C
Tab Size: 4
build_rdg

get_loop_index ()



SSA def-use


tree-loop-distribution.c
static int
number_of_lvalue_immediate_uses (struct rdg *rdg, tree stmt)
{
tree lhs;
lhs = TREE_OPERAND (stmt, 0);
if (TREE_CODE (lhs) == SSA_NAME)
{
use_operand_p imm_use_p;
imm_use_iterator iterator;
int n = 0;
FOR_EACH_IMM_USE_FAST (imm_use_p, iterator, lhs)
if (find_vertex_with_stmt (rdg, USE_STMT (imm_use_p)))
n++;
return n;
}
return 0;
}
Line: 1099 Column: 3 C Tab Size: 4 number_of_lvalue_immediate_...

dump_rdg ()

tree-loop-distribution.c

```
static void
dump_rdg (FILE *outf, struct rdg *rdg)
{
  unsigned int i;
  rdg_vertex_p vertex;

  fprintf (outf, "<graphviz><![CDATA[\n");
  fprintf (outf, "digraph ");
  print_generic_expr (outf, rdg->loop_index, 0);
  fprintf (outf, " {\n");
  for (i = 0; i < rdg->nb_vertices; i++)
    {
      fprintf (outf, " v%d [ label = \"", rdg->vertices[i].number);
      fprintf (outf, "S%d : ", rdg->vertices[i].number);
      print_generic_expr (outf, rdg->vertices[i].stmt, 0);
      fprintf (outf, "\"");
      if (rdg->vertices[i].has_dd_p)
    fprintf (outf, " shape=rect style=filled color=\".7 .3 1.0\"]");
      else
    fprintf (outf, " shape=rect]");
      fprintf (outf, ";\n");
    }
```

Line: 877 Column: 1 C Tab Size: 4 dump_rdg

Next?

- Topological sort of SCC graph
- For each SCC, create a loop
- Mark it parallel or not according to loop carried dependences

More

- http://gcc.gnu.org
- gcc@gcc.gnu.org
- gcc-patches@gcc.gnu.org