USAGE OF DIY TOOLS

INDEX:

1. herdtools7

herdtools7 is a suite of tools designed for reasoning about memory models. It is commonly used for running litmus tests, which are small parallel programs that test the memory consistency models of various architectures (such as x86, ARM, RISC-V, etc.). Here's an explanation of the components and how to set it up:

1.1 Components of herdtools7

1. **herd7**: A tool for running memory model simulations.
2. **litmus7**: A tool for running litmus tests in assembler for various architectures.
3. **diy7**: A tool for generating litmus tests.
4. **jingle**: A tool for combining litmus tests.
5. **other libraries and tools**: Supporting the main tools in performing their tasks.

2. Setting Up herdtools7 with OPAM

To work with herdtools7, you need to use OPAM to handle the installation of dependencies and the correct OCaml compiler version

2.1 OPAM (OCaml Package Manager)

It is the package manager for the OCaml programming language. It is designed to handle the installation of OCaml libraries and applications, providing an easy way to manage dependencies, install packages, and manage multiple OCaml compiler versions and switch between them.

2.2 Installation process of herdtools

Install OPAM: If you don't already have OPAM installed, install it using your package manager. For example, on Ubuntu:

```
sudo apt-get install opam
```

Initialize OPAM: Initialize OPAM and set up your environment:

```
opam init
eval $(opam env)
```

Create a Switch with the Correct OCaml Version: Ensure you are using a compatible OCaml version. For herdtools7, OCaml 4.12.0 or greater should be appropriate. Create a new switch:

```
opam switch create 4.12.0
eval $(opam env)
```

Clone the herdtools7 Repository: If you haven't already cloned the repository, do so:

```
git clone https://github.com/herd/herdtools7.git
cd herdtools7
```

Install Required Dependencies: Install all necessary dependencies listed in the herdtools7.opam file:

```
opam install . --deps-only
```

Install Specific Libraries: Specifically, ensure that zarith , dune and menhirare installed:

```
opam install zarith dune menhir
```

Build herdtools7: After setting up the environment and dependencies, build herdtools7:

```
make
```

Set Environment Variable for litmus7: Ensure the LITMUSDIR environment variable points to the directory containing header.txt and _show.awk:

```
export LITMUSDIR=$HOME/Desktop/herdtools7/litmus/libdir
```

Run litmus7: Now you can run litmus7 with your test files:

```
litmus7 test.litmus
```

3. Litmus Tests

A litmus test is a small parallel program designed to exercise the memory model of a parallel, shared-memory, computer. Given a litmus test in assembler (X86, X86_64, Power, ARM, MIPS, RISC-V), litmus7 runs the test.

Using litmus7 thus requires a parallel machine, which must additionally feature gcc and the pthreads library. Our tool litmus7 has some limitations especially as regards recognised instructions.

A litmus test source has three main sections:

1. The initial state defines the initial values of registers and memory locations. Initialisation to zero may

   be omitted.

2. The code section defines the code to be run concurrently — above there are two threads. Yes we know, our X86 assembler syntax is a mistake.

3. The final condition applies to the final values of registers and memory locations.

The provided text is a description of a litmus test for x86 memory models, often used to verify the correctness of concurrent memory operations and memory ordering in processors. Here's a detailed explanation of each part:

3.1 A simple run

Consider the following (rather classical, store buffering) SB.litmus litmus test for X86:

X86 SB
"Fre PodWR Fre PodWR"
{ x=0; y=0; }
P0
| P1
;
MOV [x],$1 | MOV [y],$1 ;
MOV EAX,[y] | MOV EAX,[x] ;
locations [x;y;]
exists (0:EAX=0 /\ 1:EAX=0)

3.1.2 Breakdown of the Litmus Test

1. X86 SB

   Meaning: This indicates the architecture and memory model being tested. In this case, it's x86 with a focus on the "SB" (Store Buffer) memory model.

2. "Fre PodWR Fre PodWR"

   Meaning: This specifies the constraints or conditions on memory operations:

   Fre (Freely): Indicates that the memory operations are not constrained by any specific ordering rules beyond those given.

   PodWR: Refers to a condition where a store operation might be reordered with respect to other operations but must appear to be executed after a write operation.

   Po: Stands for "Program Order." It indicates that the order of memory operations (reads and writes) should follow the order they appear in the program.

   dWR: Refers to "Data Writes Reordering." It constrains how data writes are ordered. Specifically:

   d: Refers to data operations or data-level constraints.

   WR: Stands for "Writes," indicating that the constraint focuses on how write operations are handled.

3. { x=0; y=0; }

   Meaning: This is the initial state of the memory locations involved in the test:

   x=0: The memory location x is initialized to 0.

   y=0: The memory location y is initialized to 0.

4. P0 | P1 ;

   Meaning: This denotes two concurrent threads or processes in the test:

P0: The operations performed by the first process/thread.

P1: The operations performed by the second process/thread.

The | symbol represents concurrency between these two processes.

5. MOV [x],$1 | MOV [y],$1 ;

   Meaning: These are the memory operations executed by the two processes:

   MOV [x],$1: In process P0, store the value 1 into memory location x.

   MOV [y],$1: In process P1, store the value 1 into memory location y.

6. MOV EAX,[y] | MOV EAX,[x] ;

   Meaning: These are the load operations executed by the two processes:

   MOV EAX,[y]: In process P0, load the value from memory location y into the register EAX.

   MOV EAX,[x]: In process P1, load the value from memory location x into the register EAX.

7. locations [x;y;]

   Meaning: Lists the memory locations involved in the test:

   [x;y;]: The test involves memory locations x and y.

8. exists (0:EAX=0 /\ 1:EAX=0)

   Meaning: This specifies the condition to check in the test:

   exists: Indicates that we are looking for a scenario where the following condition is true.

   0:EAX=0: In process P0, the register EAX holds the value 0.

   1:EAX=0: In process P1, the register EAX holds the value 0.

   /\: Logical AND operator, meaning both conditions must hold true.

A litmus test source has three main sections:
1. The initial state defines the initial values of registers and memory locations. Initialisation to zero may be omitted.
2. The code section defines the code to be run concurrently — above there are two threads. Yes we know, our X86 assembler syntax is a mistake.
3. The final condition applies to the final values of registers and memory locations.

3.1.3 Test run

Run the test by:

```
% litmus7 SB.litmus
```

```
vlab@HYVLAB8:~/Desktop$ litmus7 test.litmus
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Results for test.litmus %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X86 SB
"Fre PodWR Fre PodWR"

{
 [x]=0;
 [y]=0;
}
 P0          | P1          ;
 MOV [x],$1  | MOV [y],$1  ;
 MOV EAX,[y] | MOV EAX,[x] ;

locations [x; y;]
exists (0:EAX=0 /\ 1:EAX=0)
Generated assembler
#START _litmus_P1
            movl $1,(%rdi,%rcx)
            movl (%rdx,%rcx),%eax
#START _litmus_P0
            movl $1,(%rdx,%rcx)
            movl (%rdi,%rcx),%eax

Test SB Allowed
Histogram (3 states)
6     *>0:EAX=0; 1:EAX=0; [x]=1; [y]=1;
499998:>0:EAX=1; 1:EAX=0; [x]=1; [y]=1;
499996:>0:EAX=0; 1:EAX=1; [x]=1; [y]=1;
Ok

Witnesses
Positive: 6, Negative: 999994
Condition exists (0:EAX=0 /\ 1:EAX=0) is validated
Hash=2d53e83cd627ba17ab11c875525e078b
Observation SB Sometimes 6 999994
Time SB 0.14
```

Machine:HYVLAB8

```
processor	: 0
vendor_id	: GenuineIntel
cpu family	: 6
model		: 60
model name	: Intel(R) Core(TM) i3-4160T CPU @ 3.10GHz
stepping	: 3
microcode	: 0x28
cpu MHz		: 800.000
cache size	: 3072 KB
physical id	: 0
siblings	: 4
core id		: 0
cpu cores	: 2
apicid		: 0
initial apicid	: 0
fpu		: yes
fpu_exception	: yes
cpuid level	: 13
wp		: yes
flags		: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm arat pln pts md_clear flush_l1d
vmx flags	: vnmi preemption_timer invvpid ept_x_only ept_ad ept_1gb flexpriority tsc_offset vtpr mtf vapic ept vpid unrestricted_guest ple
bugs		: cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit srbds mmio_unknown
bogomips	: 6185.83
clflush size	: 64
cache_alignment	: 64
address sizes	: 39 bits physical, 48 bits virtual
power management:

processor	: 1
```

```
vendor_id	: GenuineIntel
cpu family	: 6
model		: 60
model name	: Intel(R) Core(TM) i3-4160T CPU @ 3.10GHz
stepping	: 3
microcode	: 0x28
cpu MHz		: 800.000
cache size	: 3072 KB
physical id	: 0
siblings	: 4
core id		: 1
cpu cores	: 2
apicid		: 2
initial apicid	: 2
fpu		: yes
fpu_exception	: yes
cpuid level	: 13
wp		: yes
flags		: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm arat pln pts md_clear flush_l1d
vmx flags	: vnmi preemption_timer invvpid ept_x_only ept_ad ept_1gb flexpriority tsc_offset vtpr mtf vapic ept vpid unrestricted_guest ple
bugs		: cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit srbds mmio_unknown
bogomips	: 6185.83
clflush size	: 64
cache_alignment	: 64
address sizes	: 39 bits physical, 48 bits virtual
power management:

processor	: 2
vendor_id	: GenuineIntel
cpu family	: 6
model		: 60
```

```
model name	: Intel(R) Core(TM) i3-4160T CPU @ 3.10GHz
stepping	: 3
microcode	: 0x28
cpu MHz		: 2713.474
cache size	: 3072 KB
physical id	: 0
siblings	: 4
core id		: 0
cpu cores	: 2
apicid		: 1
initial apicid	: 1
fpu		: yes
fpu_exception	: yes
cpuid level	: 13
wp		: yes
flags		: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm arat pln pts md_clear flush_l1d
vmx flags	: vnmi preemption_timer invvpid ept_x_only ept_ad ept_1gb flexpriority tsc_offset vtpr mtf vapic ept vpid unrestricted_guest ple
bugs		: cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit srbds mmio_unknown
bogomips	: 6185.83
clflush size	: 64
cache_alignment	: 64
address sizes	: 39 bits physical, 48 bits virtual
power management:

processor	: 3
vendor_id	: GenuineIntel
cpu family	: 6
model		: 60
model name	: Intel(R) Core(TM) i3-4160T CPU @ 3.10GHz
stepping	: 3
microcode	: 0x28
```

```
cpu MHz                 : 1862.119
cache size      : 3072 KB
physical id     : 0
siblings : 4
core id         : 1
cpu cores       : 2
apicid          : 3
initial apicid  : 3
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi
mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good
nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg
fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm
abm cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad
fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm arat pln pts md_clear flush_l1d
vmx flags       : vnmi preemption_timer invvpid ept_x_only ept_ad ept_1gb flexpriority tsc_offset vtpr mtf
vapic ept vpid unrestricted_guest ple
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit
srbds mmio_unknown
bogomips        : 6185.83
clflush size    : 64
cache_alignment         : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

Revision d21b2c0b0da97f29bba573c500894f3dd0e10378, version 7.57+1
Command line: litmus7 test.litmus
Parameters
#define SIZE_OF_TEST 100000
#define NUMBER_OF_RUN 10
#define AVAIL 1
#define STRIDE (-1)
#define MAX_LOOP 0
/* gcc options: -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread */
/* barrier: user */
/* launch: changing */
/* affinity: none */
/* memory: direct */
/* safer: write */
/* preload: random */
/* speedcheck: no */
/* alloc: dynamic */
```

3.2 Cross compilation

With option -o <name.tar> , litmus7 does not run the test. Instead, it produces a tar archive that contains the C sources for the test.

Consider SB-PPC.litmus , a Power version of the previous test:

```
PPC SB-PPC
"Fre PodWR Fre PodWR"
{ 0:r2=x; 0:r4=y; 1:r2=y; 1:r4=x; }
P0 | P1 ;
li r1,1 | li r1,1 ;
stw r1,0(r2) | stw r1,0(r2) ;
lwz r3,0(r4) | lwz r3,0(r4) ;
exists (0:r3=0 /\ 1:r3=0)
```

3.2.1 Compilation process

- create a tar using litmus7 and untar it in other directory to get the corresponding scripts of the litmus tests

```
vlab@HYVLAB8:~/Desktop/Litmus$ litmus7 -o power.tar power.litmus
File "power.litmus" Cannot find file header.txt
Fatal error: exception Misc.Fatal("Cannot find file _show.awk")
vlab@HYVLAB8:~/Desktop/Litmus$export
lITMUSDIR=/home/vlab/Desktop/herdtools7/litmus/libdir
vlab@HYVLAB8:~/Desktop/Litmus$ litmus7 -o power.tar power.litmus
vlab@HYVLAB8:~/Desktop/Litmus$ ls
power.litmus  power.tar
vlab@HYVLAB8:~/Desktop/Litmus$ mkdir power_test
vlab@HYVLAB8:~/Desktop/Litmus$ cd power_test/
vlab@HYVLAB8:~/Desktop/Litmus/power_test$ tar xf ../power.tar
```

```
vlab@HYVLAB8:~/Desktop/Litmus/power_test$ ls
comp.sh        litmus_rand.h  outs.c  power.c     run.sh    utils.c
litmus_rand.c  Makefile       outs.h  README.txt  show.awk  utils.h
```

- Test is compiled by the shell script comp.sh (or by (Gnu) make, at user's choice) and run by the shell script run.sh:

```
vlab@HYVLAB8:~/Desktop/Litmus/power_test$ sh comp.sh
power.c: In function ‘P1’:
power.c:223:1: error: unknown register name ‘r0’ in ‘asm’
  223 | asm __volatile__ (
      | ^~~
power.c: In function ‘P0’:
power.c:193:1: error: unknown register name ‘r0’ in ‘asm’
  193 | asm __volatile__ (
      | ^~~
power.c: In function ‘P1’:
power.c:223:1: error: unknown register name ‘r0’ in ‘asm’
  223 | asm __volatile__ (
      | ^~~
power.c: In function ‘P0’:
power.c:193:1: error: unknown register name ‘r0’ in ‘asm’
  193 | asm __volatile__ (
      | ^~~
```

- The error you're seeing indicates that the compiler does not recognize the r0 register name in the inline assembly code. This error is common when the code is written for a different architecture than the one targeted by the compiler. In this case, it appears the code is written for the PowerPC architecture but is being compiled on a system that does not support or recognize PowerPC inline assembly syntax.
- To resolve this issue, you need to ensure that your compiler setup is configured to target the PowerPC architecture. Here are the steps you can follow:

1. Install a Cross-Compiler for PowerPC

You need to install a cross-compiler that can compile code for the PowerPC architecture. On Ubuntu, you can install the gcc-powerpc-linux-gnu package:

```
sudo apt-get update
sudo apt-get install gcc-powerpc-linux-gnu
```

2. Modify the Compilation Script

Make sure that your compilation script (comp.sh) uses the PowerPC cross-compiler. Open the comp.sh script and replace the compiler invocation with the PowerPC cross-compiler:

```
#!/bin/sh
GCC=powerpc-linux-gnu-gcc
GCCOPTS="-Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread"
LINKOPTS=""
/bin/rm -f *.exe *.s
$GCC $GCCOPTS -O2 -c outs.c
$GCC $GCCOPTS -O2 -c utils.c
$GCC $GCCOPTS -O2 -c litmus_rand.c
$GCC $GCCOPTS $LINKOPTS -o power.exe outs.o utils.o litmus_rand.o power.c
$GCC $GCCOPTS -S power.c && awk -f show.awk power.s > power.t && /bin/rm power.s
```

3. Modify the Run Script: Ensure run.sh also uses the correct executable name and path.

vlab@HYVLAB8:~/Desktop/Litmus/power_test$ sh run.sh

```
Wednesday 07 August 2024 09:41:11 AM IST
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Results for power.litmus %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PPC SB-PPC
"Fre PodWR Fre PodWR"


{
 0:r2=x; 0:r4=y;
 1:r2=y; 1:r4=x;
}
 P0          | P1          ;
 li r1,1     | li r1,1     ;
 stw r1,0(r2) | stw r1,0(r2) ;
 lwz r3,0(r4) | lwz r3,0(r4) ;


exists (0:r3=0 /\ 1:r3=0)
Generated assembler
#START _litmus_P1
        li 3,1
        stw 3,0(9)
        lwz 6,0(10)
#START _litmus_P0
        li 3,1
        stw 3,0(9)
        lwz 6,0(10)
/lib/ld.so.1: No such file or directory
```

```
Revision d21b2c0b0da97f29bba573c500894f3dd0e10378, version 7.57+1
Command line: litmus7 -o power.tar power.litmus
Parameters
#define SIZE_OF_TEST 100000
#define NUMBER_OF_RUN 10
#define AVAIL 1
#define STRIDE (-1)
#define MAX_LOOP 0
/* gcc options: -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread */
/* barrier: user */
/* launch: changing */
/* affinity: none */
/* memory: direct */
/* safer: write */
/* preload: random */
/* speedcheck: no */
/* alloc: dynamic */
GCC=powerpc-linux-gnu-gcc
LITMUSOPTS=
Wednesday 07 August 2024 09:41:11 AM IST
vlab@HYVLAB8:~/Desktop/Litmus/power_test$ ls
comp.sh        litmus_rand.o  outs.h   power.exe   run.sh    utils.h
litmus_rand.c  Makefile       outs.o   power.t     show.awk  utils.o
litmus_rand.h  outs.c         power.c  README.txt  utils.c
```

- As we see, the condition validates also on Power. Notice that compilation produces an executable le, **power.exe**, which can be run directly, for a less verbose output.

3.3 Running several tests at once

1. To compile the two tests together, we can give two le names as arguments to litmus:

```
vlab@HYVLAB8:~/Desktop/Litmus$ ls
power.litmus  power.tar  power_test  test.litmus
vlab@HYVLAB8:~/Desktop/Litmus$ litmus7 -o two_tests.tar test.litmus power.litmus
vlab@HYVLAB8:~/Desktop/Litmus$ ls
power.litmus  power.tar  power_test  test.litmus  two_tests.tar
```

2. Or, more conveniently, list the litmus sources in a le whose name starts with @: take two ppc tests

- PPC tests refer to tests for systems using PowerPC (PPC) architecture. These tests are typically used to verify the behavior and correctness of software running on PowerPC-based systems

```
vlab@HYVLAB8:~/Desktop/Litmus$ cat @all
SB000.litmus
power.litmus
vlab@HYVLAB8:~/Desktop/Litmus$ mkdir all_test_runs
vlab@HYVLAB8:~/Desktop/Litmus$ cd all_test_runs/
vlab@HYVLAB8:~/Desktop/Litmus/all_test_runs$ tar xf ../@all.tar
vlab@HYVLAB8:~/Desktop/Litmus/all_test_runs$ ls
comp.sh       Makefile  power.c     SB000.c   utils.h
litmus_rand.c  outs.c    README.txt  show.awk
litmus_rand.h  outs.h    run.sh      utils.c
```

- change the default gcc option in Makefile to **"powerpc-linux-gnu-gcc"**

```
GCC = powerpc-linux-gnu-gcc
vlab@HYVLAB8:~/Desktop/Litmus/all_test_runs$export
PATH=$PATH:/usr/bin/powerpc-linux-gnu-gcc
vlab@HYVLAB8:~/Desktop/Litmus/all_test_runs$ make
powerpc-linux-gnu-gcc -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread -O2 -c outs.c
powerpc-linux-gnu-gcc -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread -O2 -c utils.c
powerpc-linux-gnu-gcc -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread -O2 -c litmus_rand.c
powerpc-linux-gnu-gcc -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread -S SB000.c
powerpc-linux-gnu-gcc -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread  -o SB000.exe outs.o
utils.o litmus_rand.o SB000.s
powerpc-linux-gnu-gcc -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread -S power.c
powerpc-linux-gnu-gcc -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread  -o power.exe outs.o
utils.o litmus_rand.o power.s
awk -f show.awk SB000.s > SB000.t
awk -f show.awk power.s > power.t
rm power.s SB000.s
vlab@HYVLAB8:~/Desktop/Litmus/all_test_runs$ ls
comp.sh       Makefile  power.c    run.sh     show.awk
litmus_rand.c  outs.c    power.exe  SB000.c    utils.c
litmus_rand.h  outs.h    power.t    SB000.exe  utils.h
litmus_rand.o  outs.o    README.txt  SB000.t    utils.o
vlab@HYVLAB8:~/Desktop/Litmus/all_test_runs$ sh run.sh
Wednesday 07 August 2024 10:32:13 AM IST
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Results for SB000.litmus %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PPC SB000
"PodWR Fre PodWR Fri PodWR Fre"

{
 0:r2=x; 0:r4=y;
 1:r2=y; 1:r4=z; 1:r7=x;
}
 P0           | P1           ;
 li r1,1      | li r1,1      ;
 stw r1,0(r2) | stw r1,0(r2) ;
 lwz r3,0(r4) | lwz r3,0(r4) ;
              | li r5,1      ;
              | stw r5,0(r4) ;
              | lwz r6,0(r7) ;

exists (0:r3=0 /\ 1:r3=0 /\ 1:r6=0)
Generated assembler
#START _litmus_P1
        li 30,1
        stw 30,0(9)
        lwz 5,0(8)
        li 12,1
        stw 12,0(8)
        lwz 4,0(7)
#START _litmus_P0
        li 3,1
        stw 3,0(9)
        lwz 6,0(10)
/lib/ld.so.1: No such file or directory
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Results for power.litmus %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PPC SB-PPC
"Fre PodWR Fre PodWR"
```

```
{
 0:r2=x; 0:r4=y;
 1:r2=y; 1:r4=x;
}
 P0          | P1          ;
 li r1,1     | li r1,1     ;
 stw r1,0(r2) | stw r1,0(r2) ;
 lwz r3,0(r4) | lwz r3,0(r4) ;

exists (0:r3=0 /\ 1:r3=0)
Generated assembler
#START _litmus_P1
        li 3,1
        stw 3,0(9)
        lwz 6,0(10)
#START _litmus_P0
        li 3,1
        stw 3,0(9)
        lwz 6,0(10)
/lib/ld.so.1: No such file or directory
Revision d21b2c0b0da97f29bba573c500894f3dd0e10378, version 7.57+1
Command line: litmus7 -o @all.tar @all
Parameters
#define SIZE_OF_TEST 100000
#define NUMBER_OF_RUN 10
#define AVAIL 1
#define STRIDE (-1)
#define MAX_LOOP 0
/* gcc options: -Wall -std=gnu99 -fomit-frame-pointer -O2 -pthread */
/* barrier: user */
/* launch: changing */
/* affinity: none */
/* memory: direct */
/* safer: write */
/* preload: random */
/* speedcheck: no */
/* alloc: dynamic */
GCC=gcc
```

```
LITMUSOPTS=
Wednesday 07 August 2024 10:32:13 AM IST
```

4. Generating tests

4.1 diy7

diy7 is used for generating tests. It can be configured to test various memory models and architectures.

4.1.1 Basic Command Syntax

```
diy7 -arch ARCH
```

- -arch ARCH: Specify the target architecture (e.g., X86, ARM, PowerPC).

4.1.2 Example Command

```
vlab@HYVLAB8:~/Desktop/diy7$ diy7 -arch X86
Generator produced 24 tests
Relaxations tested: {ACMFencedRW} {ACMFencedRR} {PodWW} {PodWR} {MFencedWW} {MFencedWR}
vlab@HYVLAB8:~/Desktop/diy7$ ls
2+2W000.litmus  3.SB000.litmus   R000.litmus   test.litmus
2+2W001.litmus  3.SB001.litmus   R001.litmus   W+RR+WW+RR000.litmus
2+2W002.litmus  4.LB000.litmus   SB000.litmus  W+RW+RW+RW000.litmus
2+2W003.litmus  @all             SB001.litmus  WWC000.litmus
3.2W000.litmus  IRIW000.litmus   SB002.litmus  WW+RR+WW+RR000.litmus
3.2W001.litmus  IRWIW000.litmus  SB003.litmus
3.LB000.litmus  LB000.litmus     SB004.litmus
```

example :

```
X86 SB
"Fre PodWR Fre PodWR"
{ x=0; y=0; }
P0 | P1 ;
MOV [y],$1 | MOV [x],$1 ;          #(a)Wy1 | (c)Wx1
MOV EAX,[x] | MOV EAX,[y] ;  #(b)Rx0 | (d)Ry0
exists (0:EAX=0 /\ 1:EAX=0)
```

To focus on interaction through shared memory, let us consider memory accesses, or memory events. A memory event will hold a direction (write, written W, or read, written R), a memory location (written x, y) a value and a unique label.
In any run of the simple example above, four memory events occur:

1. **Write Events:**
    - (a)Wy1: Write value 1 to memory location y
    - (c)Wx1: Write value 1 to memory location x
2. **Read Events:**
    - (b)Rx0: Read value from memory location x into EAX (in processor P0)
    - (d)Ry0: Read value from memory location y into EAX (in processor P1)

If the program's behaviour is modelled by the interleaving of its events, the rst event must be a write of value 1 to location x or y and at least one of the loads must see a 1. Thus, a SC machine would exhibit only three possible outcomes for this test:

Allowed: 0:EAX = 0 ∧ 1:EAX = 1

Allowed: 0:EAX = 1 ∧ 1:EAX = 0

Allowed: 0:EAX = 1 ∧ 1:EAX = 1

On x86 architecture, due to relaxed memory ordering:

- Additional Allowed Outcome: **Allowed: 0:EAX = 0 ∧ 1:EAX = 0** (Processor P0 reads 0 from x and 1 from y, Processor P1 reads 0 from x and 0 from y)

4.2 Introduction to candidate relaxations

Sequential Consistency (SC) requires that the memory operations of different processors appear in a globally consistent order. For SC, the order of reads and writes must be consistent with some interleaving of all operations. (In the context of memory models, the term **"interleaving of all operations"** refers to how various memory operations (reads and writes) from different threads or processes are ordered relative to each other.)

7.1 Relaxation of Sequential Consistency

- Relaxation is one of the key concepts behind simple analysis of weak memory models.
- In the context of memory consistency models, relaxation refers to the process of weakening the constraints of a more restrictive model (like Sequential Consistency) to allow for greater concurrency and potentially improve performance

4.2.1 Candidate Relaxations

A **candidate relaxation** is a specific type of relaxation applied to memory operations between threads. It is a way to describe and analyze how certain memory operations can be reordered or observed differently from what Sequential Consistency (SC) would enforce.

Candidate relaxations are used to analyze how different memory operations can be relaxed while still maintaining certain guarantees. This is crucial for:

- **Understanding Performance**: By allowing certain relaxations, systems can potentially execute more operations in parallel, improving overall performance.
- **Testing and Verification**: Tools like diy7 and diyone7 use candidate relaxations to generate tests that evaluate how well a system adheres to different memory models and how these relaxations impact program behavior.

4.2.2 Relaxations of SC

Relaxations of SC allow more flexibility in the ordering and visibility of memory operations. They define how and when updates to memory locations become visible to other threads. These relaxations can lead to increased performance but may also make the behavior of programs more complex to understand and predict.

4.2.3 Example of Candidate Relaxations

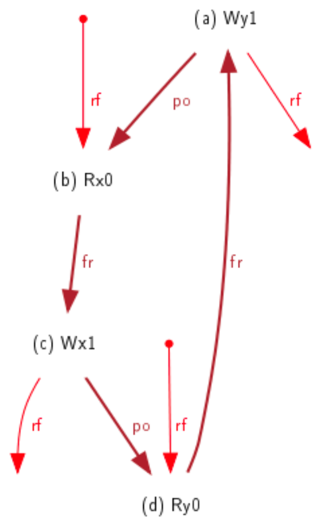
Here's how some candidate relaxations might be defined in terms of their effects:

- **Rfi (Read From Invalid)**: Reads a value from a location that has been written but is now invalid.
- **Rfe (Read From Existing)**: Reads a value from a location that was written and is still valid.
- **Coi (Coherence Order Invalid)**: The coherence order between writes might be relaxed.
- **Coe (Coherence Order Exists)**: Maintains coherence order between writes.
- **Fri (Forwarded Read Invalid)**: A read operation sees a value from a write operation that was expected to be invalidated.
- **Fre (Forwarded Read Exists)**: A read operation sees a value from a write operation in a coherence-preserving manner.

Consider again our classical example, from a SC perspective. We briefly argued that the outcome 0:EAX = 0 $\wedge$ 1:EAX = 0 is forbidden by SC. We now present a more complete reasoning:

- From the condition on outcome, we get the values in read events: (b) Rx0 and (d) Ry0.
- Because of these values, (b) Rx0 must precede the write (c)Wx1 in the final interleaving of SC. Similarly, (d) Ry0 must precede the write (a)Wy1. This we note (b) fr → (c) and (d) fr → (a).

- Because of sequential execution order on one single processor (a.k.a. program order), (a)Wy1 must precede (b) Rx0 (first processor); while (c)Wx1 must precede (d) Ry0 (second processor). This we note (a) po ⟶ (b) and (c) po ⟶ (d)
- We synthesise the four constraints above as the following graph:



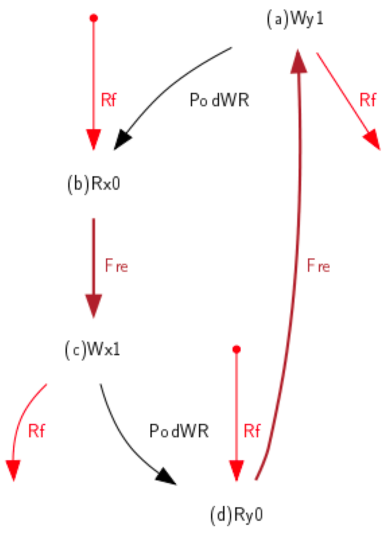
Constraint arrows or global arrows are shown in brown colour. As the graph contains a cycle of brown arrows, the events cannot be ordered
The key idea of diy7 resides in producing programs from similar cycles. To that aim, the edges in cycles must convey additional information:

- For **po ⟶ edges**, we consider whether the locations of the events on both sides of the edge are the same or not **('s' or 'd')**; and the direction of these events **(W or R)**. For instance the two po ⟶ edges in the example are **PodWR** (program order edge between a write and a read whose locations are different).
- For **fr ⟶ edges,** we consider whether the processor of the events on both sides of the edge are the same processor **(i for internal)** or different processors **(e for external).**
  For instance the two fr ⟶ edges in the example are Fre.

Order Consideration:

- **fr** considers the coherence order and indicates that the read gets its value from a write that happens before it in coherence order.
- **rf** focuses on the direct dependency of a read on a write without necessarily considering the coherence order.

So far so good, but our x86 machine produced the outcome 0:EAX=0 ⋀ 1:EAX=0. Loads may be reordered with older stores to different locations, which we rephrase as: PodWR is relaxed. Considering Fre to be safe,we have the graph: And the brown sub-graph becomes acyclic.



The diy7 suite precisely provides tools for this approach. As a first example, SB.litmus can be created as follows: % diyone7 -arch X86 -name SB Fre PodWR Fre PodWR

4.2.4 generating single test using diyone7

```
vlab@HYVLAB8:~/Desktop/diy7$ diyone7 -arch X86 -name SB Fre PodWR Fre PodWR
vlab@HYVLAB8:~/Desktop/diy7$ ls
SB.litmus
vlab@HYVLAB8:~/Desktop/diy7$ cat SB.litmus
X86 SB
"Fre PodWR Fre PodWR"
Generator=diyone7 (version 7.57+1)
Prefetch=0:x=F,0:y=T,1:y=F,1:x=T
Com=Fr Fr
Orig=Fre PodWR Fre PodWR
{
}
 P0          | P1          ;
 MOV [x],$1  | MOV [y],$1  ;
 MOV EAX,[y] | MOV EAX,[x] ;
exists (0:EAX=0 /\ 1:EAX=0)
```

4.2.5 generating tests using diy7

As a second example, we can produce several similar tests as follows:

```
vlab@HYVLAB8:~/Desktop/diy7$  diy7 -arch X86 -safe Fre -relax PodWR -name SB
Generator produced 2 tests
Relaxations tested: {PodWR}
vlab@HYVLAB8:~/Desktop/diy7$ ls
@all  SB000.litmus  SB001.litmus
```

```
vlab@HYVLAB8:~/Desktop/diy7$ cat @all
# diy7 -arch X86 -safe Fre -relax PodWR -name SB
# Version 7.57+1, Revision: d21b2c0b0da97f29bba573c500894f3dd0e10378
SB000.litmus
SB001.litmus
```

4.3 More candidate relaxations

We assume the memory to be coherent. Coherence implies that, in a given execution, the writes to a given location are performed by following a sequence, or coherence order, and that all processors see the same sequence.

4.3.1 Coherence Order

This is the sequence in which writes to a specific memory location are observed by all processors. Coherence order must be preserved such that writes to the same memory location are observed in the same order by all processors.

In diy7, coherence orders are specified indirectly through different types of edges in the graph:

4.3.1.1 Coe (Coherence Order External)

- **Definition**: Represents a coherence order edge between two writes (W) to the same memory location performed by different processors. The first write precedes the second write in the coherence order of the memory location.
- **Usage**: Ensures that all processors will observe the writes in the same order. If the coherence order is respected, the test generated by diy7 will reflect this constraint.

4.3.1.2 Coi (Coherence Order Internal)

- **Definition**: Represents a coherence order edge between two writes to the same memory location performed by the same processor. The first write precedes the second write in the coherence order of the memory location.
- **Usage**: Ensures that within the same processor, writes to a memory location follow a coherent order.

Let's consider an example to clarify:

- **Scenario**: Suppose there are two writes to the same memory location ℓ, one by processor P1 and another by processor P2.

- **Coe Example**: If the first write by P1 should be seen before the second write by P2, you would use a Coe edge to specify this coherence order. This ensures that all processors observe the writes to ℓ in the order P1 -> P2.
- **Coi Example**: If both writes are by the same processor, say P1, and you want to ensure that the first write to ℓ is observed before the second write by P1, you would use a Coi edge to specify this order.

Consider an instance:

```
vlab@HYVLAB8:~/Desktop/diy7$ diyone7 -arch X86 -name x86-2+2W Coe PodWW Coe PodWW
vlab@HYVLAB8:~/Desktop/diy7$ ls
@all  SB000.litmus  SB001.litmus  x86-2+2W.litmus
vlab@HYVLAB8:~/Desktop/diy7$ cat x86-2+2W.litmus
X86 x86-2+2W
"Coe PodWW Coe PodWW"
Generator=diyone7 (version 7.57+1)
Prefetch=0:x=F,0:y=W,1:y=F,1:x=W
Com=Co Co
Orig=Coe PodWW Coe PodWW
{
}
 P0         | P1         ;
 MOV [x],$2 | MOV [y],$2 ;
 MOV [y],$1 | MOV [x],$1 ;
exists ([x]=2 /\ [y]=2)
```

By the coherence hypothesis, checking the final value of locations suffices to characterise those two coherence orders, as expressed by the final condition of x86-2+2W: **exists (x=2 /\ y=2)**

4.3.2 Candidate Relaxations: Rfe and Rfi

4.3.2.1 Rfe (Read From External)

- **Definition**: Represents a read (R) from a different processor that reads its value from a write (W) on another processor. This is used to model communication between different processors through shared memory.
- **Usage**: In a graph, an Rfe edge shows that the read operation on one processor loads the value written by another processor.
- **Example**: If processor P1 writes a value to location x and processor P2 reads from x, the read would be considered an Rfe edge if it reads the value written by P1.

4.3.2.2 Rfi (Read From Internal)

- **Definition**: Represents a read (R) from the same processor that reads its value from a write (W) on the same processor. This is used to model communication within the same processor.

- **Usage**: In a graph, an Rfi edge shows that the read operation on the same processor loads the value written by an earlier write on the same processor.
- **Example**: If processor P1 writes a value to location x and later reads from x, the read would be considered an Rfi edge if it reads the value written by the earlier write on P1.

4.4 Generating the iriw Test

The IRIW test (Independent Reads of Independent Writes) is used to explore the behavior of memory models in multiprocessor systems. It examines how independent reads and writes on different processors can lead to unexpected outcomes due to memory reordering or relaxations.
Here's how you can generate it using diyone7:

4.4.1 External IRIW Test

This test involves independent reads and writes across different processors, modeled with Rfe edges

```
% diyone7 -arch X86 Rfe PodRR Fre Rfe PodRR Fre -name iriw
```

```
vlab@HYVLAB8:~/Desktop/diy7$ diyone7 -arch X86 Rfe PodRR Fre Rfe PodRR Fre -name iriw
vlab@HYVLAB8:~/Desktop/diy7$ ls
@all  iriw.litmus  SB000.litmus  SB001.litmus  x86-2+2W.litmus
vlab@HYVLAB8:~/Desktop/diy7$ cat iriw.litmus
X86 iriw
"Rfe PodRR Fre Rfe PodRR Fre"
Generator=diyone7 (version 7.57+1)
Prefetch=1:x=F,1:y=T,3:y=F,3:x=T
Com=Rf Fr Rf Fr
Orig=Rfe PodRR Fre Rfe PodRR Fre
{
}
 P0        | P1         | P2        | P3         ;
 MOV [x],$1 | MOV EAX,[x] | MOV [y],$1 | MOV EAX,[y] ;
           | MOV EBX,[y] |           | MOV EBX,[x] ;
exists (1:EAX=1 /\ 1:EBX=0 /\ 3:EAX=1 /\ 3:EBX=0)
vlab@HYVLAB8:~/Desktop/diy7$
```

Explanation:

- **Rfe**: Read from an external write.
- **PodRR**: Program order dependency between a read and another read, where the locations of the reads are different.
- **Fre**: From-read edge, where the read loads the value of a write from another processor.
- This combination ensures that the generated test captures the behavior where reads from different processors read values written by other processors, checking the order of these reads and writes.

4.4.2 Internal IRIW Test

This variation replaces Rfe with Rfi, modeling interactions within the same processor.

We generate its internal variation (i.e. where all Rfe are replaced by R) as easily:

```
% diyone7 -arch X86 Rfi PodRR Fre Rfi PodRR Fre -name iriw-internal
```

```
vlab@HYVLAB8:~/Desktop/diy7$ diyone7 -arch X86 Rfi PodRR Fre Rfi PodRR Fre -name
iriw-internal
vlab@HYVLAB8:~/Desktop/diy7$ ls
@all               iriw.litmus   SB001.litmus
iriw-internal.litmus  SB000.litmus  x86-2+2W.litmus
vlab@HYVLAB8:~/Desktop/diy7$ cat iriw-internal.litmus
X86 iriw-internal
"Rfi PodRR Fre Rfi PodRR Fre"
Generator=diyone7 (version 7.57+1)
Prefetch=0:x=F,0:y=T,1:y=F,1:x=T
Com=Fr Fr
Orig=Rfi PodRR Fre Rfi PodRR Fre
{
}
 P0          | P1          ;
 MOV [x],$1  | MOV [y],$1  ;
 MOV EAX,[x] | MOV EAX,[y] ;
 MOV EBX,[y] | MOV EBX,[x] ;
exists (0:EAX=1 /\ 0:EBX=0 /\ 1:EAX=1 /\ 1:EBX=0)
```

- **Explanation**:
    - **Rfi**: Read from an internal write.
    - **PodRR**: Program order dependency between a read and another read, where the locations of the reads are different.
    - **Fre**: From-read edge, where the read loads the value of a write from another processor.
    - This combination ensures that the generated test captures the behavior where reads and writes occur within the same processor, checking the order of these operations.

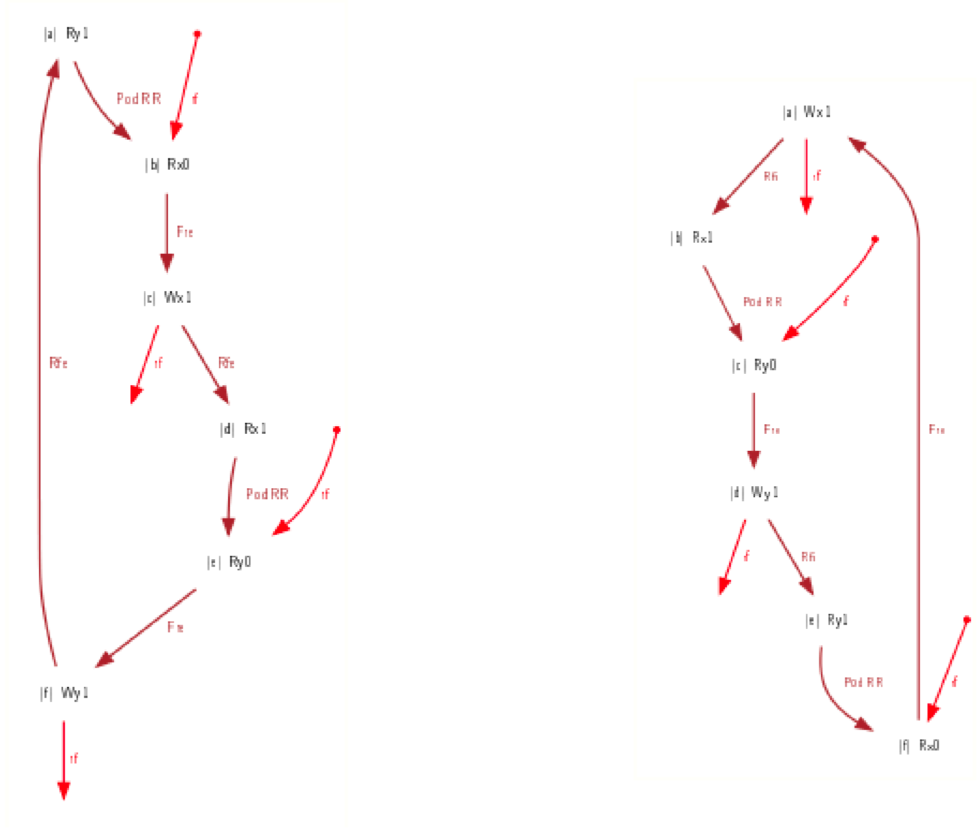
4.4.3 Generating IRIW Test for Power (PPC)

The IRIW test (Independent Reads of Independent Writes) can be generated for the Power architecture by specifying the appropriate candidate relaxations:

```
% diyone7 -arch PPC Rfe PodRR Fre Rfe PodRR Fre -name iriw_power
```

```
vlab@HYVLAB8:~/Desktop/diy7$ diyone7 -arch PPC Rfe PodRR Fre Rfe PodRR Fre -name
iriw_power
vlab@HYVLAB8:~/Desktop/diy7$ cat iriw_power.litmus
PPC iriw_power
"Rfe PodRR Fre Rfe PodRR Fre"
Generator=diyone7 (version 7.57+1)
Prefetch=1:x=F,1:y=T,3:y=F,3:x=T
Com=Rf Fr Rf Fr
Orig=Rfe PodRR Fre Rfe PodRR Fre
{
0:r2=x;
1:r2=x; 1:r4=y;
2:r2=y;
3:r2=y; 3:r4=x;
}
 P0          | P1          | P2          | P3          ;
 li r1,1     | lwz r1,0(r2) | li r1,1     | lwz r1,0(r2) ;
 stw r1,0(r2) | lwz r3,0(r4) | stw r1,0(r2) | lwz r3,0(r4) ;
exists (1:r1=1 /\ 1:r3=0 /\ 3:r1=1 /\ 3:r3=0)
```

Figure 1: Cycles for iriw and iriw-internal



4.5 Summary of simple candidate relaxations

4.5.1.Communication candidate relaxations

We call communication candidate relaxations the relations between two events communicating through memory, though they could belong to the same processor. Thus, these events operate on the same memory location.

diy7 syntax	Source	Target	Processor	Additional property
`Rfi`	W	R	Same	Target reads its value from source
`Rfe`	W	R	Different	Target reads its value from source
`Coi`	W	W	Same	Source precedes target in coherence order
`Coe`	W	W	Different	Source precedes target in coherence order
`Fri`	R	W	Same	Source reads a value from a write that precedes target in coherence order
`Fre`	R	W	Different	Source reads a value from a write that precedes target in coherence order

** Notice that Ws is a deprecated synonym of Co

main difference between the Rf and Fr

Rf: Direct read from a write on the same processor, typically indicating an immediate read following a write.

Fr: Read observes a write that precedes it in the coherence order, which might involve other operations in between.

4.5.2 Program order candidate relaxations

We call program order candidate relaxations each relation between two events in the program order. These events are on the same processor, since they are in program order. As regards code output, diy7 interprets a program order candidate relaxation by generating two memory instructions (load or store) following one another.

Program order candidate relaxations have the following syntax: **Po(s|d)(R|W)(R|W)**

where:

- s (resp. d) indicates that the two events are the same (resp. different) location(s);
- R (resp. W) indicates an event to be a read (resp. a write);

diy7 syntax	Source	Target	Location
`PosRR`	R	R	Same
`PodRR`	R	R	Diff
`PosRW`	R	W	Same
`PodRW`	R	W	Diff
`PosWW`	W	W	Same
`PodWW`	W	W	Diff
`PosWR`	W	R	Same
`PodWR`	W	R	Diff

It is to be noticed that PosWR, PosWW and PosRW are similar to R, Coi and Fri, respectively. More precisely, diy7 is unable to consider a PosWR (or PosWW, or PosRW) candidate relaxation as not being also a R (or Coi, or Fri) candidate relaxation. However, litmus test conditions may be more informative in the case of R and Fri.

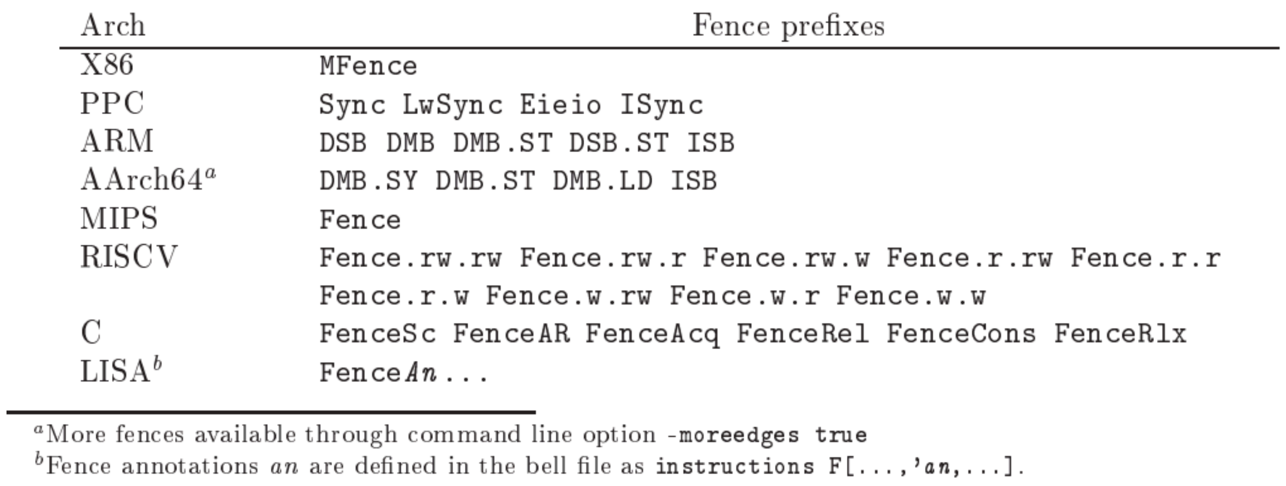
4.5.3 Fence candidate relaxations

Relaxed architectures provide specific instructions, namely barriers or fences, to enforce order of memory accesses. In diy7 the presence of a fence instruction is specified with fence candidate relaxations, similar to program order candidate relaxations, except that a fence instruction is inserted. Hence we have **FencesRR, FencedRR**. etc. The inserted fence is the strongest fence provided by the architecture, that is **mfence for x86 dmb for ARM , fence for riscv and sync for Power.**

Fences can also be specified by using specific names. For instance, we have **MFence for x86**; while on **PPC we have Sync, LwSync, Eieio and ISync.**

Hence, to yield two reads to different locations and separated by the lightweight PPC barrier lwsync, we specify **LwSyncdRR**

Figure 3: Fence prefixes per architecture



Arch	Fence prefixes
X86	`MFence`
PPC	`Sync LwSync Eieio ISync`
ARM	`DSB DMB DMB.ST DSB.ST ISB`
AArch64[a]	`DMB.SY DMB.ST DMB.LD ISB`
MIPS	`Fence`
RISCV	`Fence.rw.rw Fence.rw.r Fence.rw.w Fence.r.rw Fence.r.r Fence.r.w Fence.w.rw Fence.w.r Fence.w.w`
C	`FenceSc FenceAR FenceAcq FenceRel FenceCons FenceRlx`
LISA[b]	`Fence`*An*`...`

[a]More fences available through command line option `-moreedges true`

[b]Fence annotations *an* are defined in the bell file as `instructions F[...,'an,...]`.

```
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ diy7 -show fences
 Eieio LwSync Sync ISync
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ diy7 -arch RISCV -show fences
 Fence.i Fence.r.r Fence.r.w Fence.r.rw Fence.w.r Fence.w.w Fence.w.rw Fence.rw.r Fence.rw.w Fence.rw.rw
Fence.tso Fence.iorw.iorw
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ diy7 -arch X86 -show fences
 MFence
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ diy7 -arch MIPS -show fences
 Sync
```

```
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ diy7 -arch ARM -show fences
 DSB.ST DSB DMB.ST DMB ISB
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ diy7 -arch AArch64 -show fences
IC.IVAUp DC.CVAUp IC.IVAUn DC.CVAUn DSB.SY DMB.SY DSB.ST DMB.ST DSB.LD DMB.LD ISB
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ diy7 -arch C -show fences
Warning: optimised conditions are not supported by C arch
 FenceAcq FenceRel FenceAR FenceSc FenceRlx FenceCon
```

4.6 Fence prefixes

4.6.1 Mfence (X86)

An MFENCE (Memory Fence) is an instruction used in computer architecture to enforce memory ordering constraints. It is a type of memory barrier that ensures all memory read and write operations that were issued before the MFENCE instruction are completed before any memory read and write operations issued after the MFENCE instruction can begin. This is crucial for maintaining memory consistency, especially in multi-processor systems where memory operations can be reordered for performance optimization.

On x86 architectures, MFENCE is used to enforce ordering of both read and write operations. Here is a simple example in x86 assembly:

```
; Assume RAX and RBX are general-purpose registers
MOV [x], RAX     ; Write to memory location x
MFENCE           ; Memory fence
MOV RBX, [y]     ; Read from memory location y
```

In this example:

- The write to memory location x must complete before the MFENCE instruction.
- The MFENCE ensures that the read from memory location y happens only after the write to x is complete.

4.6.1.1 Usage in Multi-threading

In a multi-threaded environment, MFENCE can be used to synchronize memory access between threads. Consider the following pseudo-code for two threads:

Thread 1:

```
MOV [x], 1      ; Write 1 to x
```

```
MFENCE         ; Memory fence
MOV [flag], 1   ; Set flag to indicate x is updated
```

Thread 2:

```
; Busy-wait until flag is set

LOOP:
   CMP [flag], 1
   JNE LOOP

MFENCE         ; Memory fence
MOV EAX, [x]    ; Read value from x
```

In this scenario:

- Thread 1 writes to x and then sets a flag to indicate that the value at x is updated. The MFENCE ensures that the write to x completes before the flag is set.
- Thread 2 waits until the flag is set. The MFENCE ensures that once the flag is observed as set, the read from x reflects the updated value.

4.6.1.2 Using the diy7 tool to generate tests involving FENCE

- list the available fences using the command **"diy7 -show fences"**

  vlab@HYVLAB8:~/Desktop/diy7$ diy7 -show fences

  ```
  Eieio LwSync Sync ISync
  ```
- use diyone7 and generate the Sync fence test

  ```
  % diy7 -arch X86 -name mytest -o mytest_tests.tar -optcond -cond observe
  ```

  - **diyone7**: The tool being used. diyone7 is a test generator for litmus tests, focusing on memory models and their relaxations.
  - **-arch X86**: Specifies the target architecture for which the tests should be generated. In this case, it's X86, indicating that the generated tests will be suitable for the x86 architecture.
  - **-name mytest**:The generated tests will be named using this base name (mytest), with possible suffixes or numbers appended as needed.
  - **-o mytest_tests.tar**: The results of the test generation will be archived into a file named mytest_tests.tar. This file will contain the litmus tests in a tarball format.
  - **-optcond** : This option enables optimization of the conditions in the tests, which can make the tests more concise or efficient. It helps in generating more relevant test cases by optimizing the conditions under which certain memory ordering constraints are checked.

- **-cond observe:** Sets the final condition style to observe. This option configures the generator to produce tests with conditions that reflect the observation of specific behaviors or interactions.

```
vlab@HYVLAB8:~/Desktop/diy7$ diy7 -arch X86 -name fence1 -o fence_tests1.tar -optcond -cond observe
Generator produced 42 tests
Relaxations tested: {ACMFencedRW} {ACMFencedRR} {PodWW} {PodWR} {MFencedWW} {BCMFencedWW} {MFencedWR}
vlab@HYVLAB8:~/Desktop/diy7$ ls
@all fence_tests1.tar iriw-internal.litmus iriw.litmus iriw_power.litmus SB000.litmus SB001.litmus x86-2+2W.litmus
vlab@HYVLAB8:~/Desktop/diy7$ mkdir fence_test1
vlab@HYVLAB8:~/Desktop/diy7$ cd fence_test1/
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ tar -xvf ../fence_tests1.tar
./
./fence1041.litmus
./fence1037.litmus
./@all
./fence1025.litmus
./fence1002.litmus
./fence1027.litmus
./fence1038.litmus
./fence1014.litmus
./fence1039.litmus
./fence1010.litmus
./fence1022.litmus
./fence1032.litmus
./fence1030.litmus
./fence1019.litmus
./fence1001.litmus
./fence1029.litmus
./fence1009.litmus
./fence1035.litmus
./fence1018.litmus
./fence1005.litmus
./fence1007.litmus
./fence1023.litmus
```

```
./fence1034.litmus
./fence1026.litmus
./fence1031.litmus
./fence1003.litmus
./fence1013.litmus
./fence1028.litmus
./fence1016.litmus
./fence1021.litmus
./fence1033.litmus
./fence1024.litmus
./fence1040.litmus
./fence1004.litmus
./fence1000.litmus
./fence1015.litmus
./fence1006.litmus
./fence1012.litmus
./fence1011.litmus
./fence1008.litmus
./fence1036.litmus
./fence1020.litmus
./fence1017.litmus
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ ls
@all            fence1005.litmus  fence1011.litmus  fence1017.litmus  fence1023.litmus  fence1029.litmus
fence1035.litmus  fence1041.litmus
fence1000.litmus    fence1006.litmus    fence1012.litmus    fence1018.litmus    fence1024.litmus
fence1030.litmus  fence1036.litmus
fence1001.litmus    fence1007.litmus    fence1013.litmus    fence1019.litmus    fence1025.litmus
fence1031.litmus  fence1037.litmus
fence1002.litmus    fence1008.litmus    fence1014.litmus    fence1020.litmus    fence1026.litmus
fence1032.litmus  fence1038.litmus
fence1003.litmus    fence1009.litmus    fence1015.litmus    fence1021.litmus    fence1027.litmus
fence1033.litmus  fence1039.litmus
fence1004.litmus    fence1010.litmus    fence1016.litmus    fence1022.litmus    fence1028.litmus
fence1034.litmus  fence1040.litmus
```

4.6.2 RISCV fence prefixes

The RISC-V FENCE instruction is used to ensure ordering of memory operations. The format of the FENCE instruction is as follows:

FENCE [pred],[succ]

Where pred specifies the set of operations that must be completed before the fence, and succ specifies the set of operations that must be started after the fence.

Each pred and succ is a combination of the following bits:

- **r:** Read operations
- **w:** Write operations
- **o:** Other operations
- **i:** Input/output operations

Below are explanations and examples for the different FENCE instructions mentioned:

1. **FENCE.rw.rw**:
   - Ensures that all read and write operations before the fence are completed before any read or write operations after the fence are started.
2. **FENCE.rw.r**:
   - Ensures that all read and write operations before the fence are completed before any read operations after the fence are started.
3. **FENCE.rw.w**:
   - Ensures that all read and write operations before the fence are completed before any write operations after the fence are started.
4. **FENCE.r.rw**:
   - Ensures that all read operations before the fence are completed before any read or write operations after the fence are started.
5. **FENCE.r.r**:
   - Ensures that all read operations before the fence are completed before any read operations after the fence are started.
6. **FENCE.r.w**:
   - Ensures that all read operations before the fence are completed before any write operations after the fence are started.
7. **FENCE.w.rw**:
   - Ensures that all write operations before the fence are completed before any read or write operations after the fence are started.

8. **FENCE.w.r**:
    - Ensures that all write operations before the fence are completed before any read operations after the fence are started.
9. **FENCE.w.w**:
    - Ensures that all write operations before the fence are completed before any write operations after the fence are started.

4.7 Annotations

Annotations in the context of litmus testing and memory models are used to specify the behavior of certain operations in terms of atomicity, ordering, and visibility. They help define how operations should be treated and how they interact with each other within a memory model.

As annotations apply to events, they are pseudo-candidates that appear in-between actual candidate relaxations. For instance, consider the AArch64 architecture that features primitive store release (annotation L) and load acquire (annotation A) instructions.

For AArch64, you can use annotations to specify a release-acquire idiom in your tests. The release-acquire idiom typically involves:

- **Store Release (L)**: A store that releases a memory barrier.
- **Load Acquire (A)**: A load that acquires a memory barrier.

One may specify the well known message-passing release-acquire idiom as follows:

```
% diyone7 -arch AArch64 PodWW L Rfe A PodRR Fre
```

4.7.1 Command to observe the annotations of specific architecture

```
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ diy7 -show annotations
 R A w0 h0 h2
```

General Symbols:

- **R**: Represents a read operation.
- **A**: Represents an acquire operation, which typically ensures that subsequent operations are not reordered before this operation.
- **w0**: Likely represents a write operation at a specific location or index.
- **h0, h2**: These could denote specific events or operations, such as read or write events at certain indices or points in time.

4.7.2 Types of annotations

4.7.2.1 X or A (Atomic)

This annotation is used to denote atomic accesses. Atomic operations typically include load-reserve/store-conditional pairs (lr/sc) or read-modify-write instructions. In the context of architectures and litmus testing, these annotations help in generating tests that reflect the atomicity requirements of the system.

4.7.2.1.1 Atomicity

- An operation is atomic if it appears to be instantaneous to other threads or processes. This means that once an atomic operation starts, it runs to completion without being interrupted.
- Atomicity prevents race conditions where multiple threads or processes access shared data simultaneously in an inconsistent manner.

4.7.2.1.2 Load-Reserve and Store-Conditional (LR/SC)

- **Load-Reserve (LR)**: Reads a value from memory and reserves the location for a potential write. This reservation prevents other processors or threads from writing to that location until the reservation is either confirmed or invalidated.
- **Store-Conditional (SC)**: Attempts to write a value to the memory location reserved by a previous LR operation. The write is successful only if no other thread has modified the location since the LR operation.

4.7.2.1.3 Read-Modify-Write (RMW)

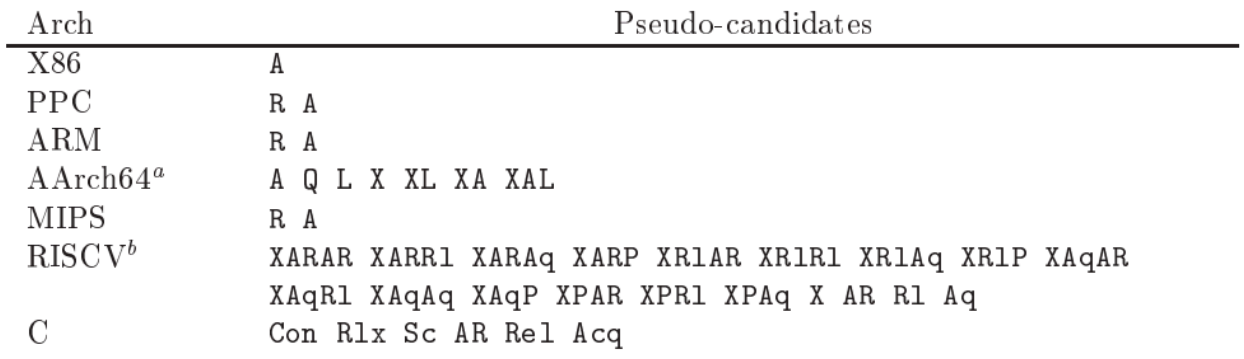
- An operation that reads a value from memory, modifies it, and writes it back to memory as a single atomic step. Ensures that no other thread can change the value between reading and writing, thus preventing data races.

4.7.2.2 L (Store Release)

Indicates a store operation that acts as a release barrier, ensuring that previous writes are visible before the store operation.

4.7.2.3 A (Load Acquire)

Indicates a load operation that acts as an acquire barrier, ensuring that previous writes are visible before the load operation.



Arch	Pseudo-candidates
X86	A
PPC	R A
ARM	R A
AArch64[a]	A Q L X XL XA XAL
MIPS	R A
RISCV[b]	XARAR XARRl XARAq XARP XRlAR XRlRl XRlAq XRlP XAqAR XAqRl XAqAq XAqP XPAR XPRl XPAq X AR Rl Aq
C	Con Rlx Sc AR Rel Acq

```
vlab@HYVLAB8:~/Desktop/diy7/fence_test1$ diy7 -arch RISCV -show annotations
 XARAR XARRl XARAq XARP XRlAR XRlRl XRlAq XRlP XAqAR XAqRl XAqAq XAqP XPAR XPRl XPAq X AR Rl Aq w0 h0 h2
```

4.7.2.4 General Letter Meanings

1. **X**: This is typically used to denote **General/Atomic Operation** or a placeholder for various operations. It might represent any memory operation or a generalized atomic operation.
2. **R**: Stands for **Read** operation. It indicates that the memory operation in this position is a read from memory.
3. **W**: Stands for **Write** operation. It denotes a memory write operation.
4. **A**: Indicates an **Acquire** operation. Acquire operations enforce ordering constraints to ensure that subsequent operations cannot be reordered before this operation, often used to synchronize memory.
5. **R**: Represents a **Release** operation. Release operations also enforce ordering constraints but typically focus on ensuring that preceding operations are completed before subsequent operations.
6. **P**: Denotes a **Predicate** or **Predicate Operation**. IRefers to conditions or logical checks that influence when and how memory operations occur. It deals with ensuring certain conditions before proceeding with memory operations.
7. **l**: Refers to **Location** or **Latency**.Refers to the specific memory address being accessed (location) or the delay associated with completing a memory operation (latency). It helps in defining where memory operations occur and understanding timing aspects.
8. **q**: Represents **Queue** or could be used to denote **Query** or **Query Operation**. It often indicates operations related to queue management or querying state.
9. **h**: Denotes **High-level Events** or **Specific Events**. It may refer to specific event points or high-level events in the model.
10. **0, 1, 2**: These typically refer to **Indices** or **Specific Instances**. For example, h0 might denote the first high-level event, while h2 might denote the third.

4.8 Use of conf file to generate the litmus tests

Repeating command line options is painful and error prone. Besides, configuration parameters may get lost. Thus, we regroup those in configuration files that simply list the options to be passed to diy7, one option per line.

For instance here is the configuration file for testing the safe relaxations of x86,

4.8.1 x86-safe.conf.

```
#safe x86 conf file
-arch X86
#Generate tests on four processors or less
-nprocs 4
#From cycles of size at most six
-size 6
#With names safe000, safe0001,...
-name safe
#List of safe relaxations
-safe PosR* PodR* PodWW PosWW Rfe Wse Fre FencesWR FencedWR
```

Observe that the syntax of candidate relaxations allows one shortcut: the wildcard * stands for W and R. Thus PodR* gets expanded to the two candidate relaxations PodRR and PodRW.

We get safe tests by issuing the following command, preferably in a specific directory, say safe.

```
% diy7 -conf x86-safe.conf
```

```
vlab@HYVLAB8:~/Desktop/diy7$ mkdir conf_tests
vlab@HYVLAB8:~/Desktop/diy7$ cd conf_tests/
vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ gvim x86-safe.conf
vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ cat x86-safe.conf
#safe x86 conf file
-arch X86
#Generate tests on four processors or less
-nprocs 4
#From cycles of size at most six
-size 6
#With names safe000, safe0001,...
-name safe
#List of safe relaxations
-safe PosR* PodR* PodWW PosWW Rfe Wse Fre FencesWR FencedWR
vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ diy7 -conf x86-safe.conf
```

```
Generator produced 38 tests
```

vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ ls

```
@all            safe009.litmus  safe019.litmus  safe029.litmus
safe000.litmus  safe010.litmus  safe020.litmus  safe030.litmus
safe001.litmus  safe011.litmus  safe021.litmus  safe031.litmus
safe002.litmus  safe012.litmus  safe022.litmus  safe032.litmus
safe003.litmus  safe013.litmus  safe023.litmus  safe033.litmus
safe004.litmus  safe014.litmus  safe024.litmus  safe034.litmus
safe005.litmus  safe015.litmus  safe025.litmus  safe035.litmus
safe006.litmus  safe016.litmus  safe026.litmus  safe036.litmus
safe007.litmus  safe017.litmus  safe027.litmus  safe037.litmus
safe008.litmus  safe018.litmus  safe028.litmus  x86-safe.conf
```

Here are the configuration files for confirming that R and PodWR are relaxed, x86-rfi.conf and x86-podwr.conf.

4.8.2 x86-rfi.conf

```
#rfi x86 conf file
-arch X86
-nprocs 4
-size 6
-name rfi
-safe PosR* PodR* PodWW PosWW Rfe Wse Fre FencesWR FencedWR
-relax Rfi
#At most three "instructions" per thread.
-ins 3
```

vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ gvim x86-rfi.conf

vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ cat x86-rfi.conf

```
#rfi x86 conf file
-arch X86
-nprocs 4
-size 6
-name rfi
-safe PosR* PodR* PodWW PosWW Rfe Wse Fre FencesWR FencedWR
-relax Rfi
#At most three "instructions" per thread.
```

```
-ins 3
vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ diy7 -conf x86-rfi.conf
Generator produced 28 tests
Relaxations tested: {Rfi}
vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ ls
@all           rfi006.litmus  rfi013.litmus  rfi020.litmus  rfi027.litmus
rfi000.litmus  rfi007.litmus  rfi014.litmus  rfi021.litmus  x86-rfi.conf
rfi001.litmus  rfi008.litmus  rfi015.litmus  rfi022.litmus
rfi002.litmus  rfi009.litmus  rfi016.litmus  rfi023.litmus
rfi003.litmus  rfi010.litmus  rfi017.litmus  rfi024.litmus
rfi004.litmus  rfi011.litmus  rfi018.litmus  rfi025.litmus
rfi005.litmus  rfi012.litmus  rfi019.litmus  rfi026.litmus
```

4.8.3 x86-PodWR.conf

```
#podrw x86 conf file
-arch X86
-nprocs 4
-size 6
-name podwr
-safe Fre
-relax PodWR
```

```
vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ gvim x86-podwr.conf
vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ diy7 -conf x86-podwr.conf
Generator produced 2 tests
Relaxations tested: {PodWR}
vlab@HYVLAB8:~/Desktop/diy7/conf_tests$ ls
@all             podwr001.litmus  x86-rfi.conf
podwr000.litmus  x86-podwr.conf   x86-safe.conf
```

4.9 Test variations with diycross7

The tool diycross7 has an interface similar to diyone7, except it accepts a list of candidate relaxations where diyone7 accepts single candidate relaxations. The new tool produces the test resulting by cross-producing the lists.

For instance, one can generate all variations on the IRIW test that involve data dependencies and the lightweight barrier lwsync as follows:

```
% diycross7 -arch PPC -name IRIW Rfe DpdR,LwSyncdRR Fre Rfe DpdR,LwSyncdRR Fre
```

```
vlab@HYVLAB8:~/Desktop/diy7/diycross7$ diycross7 -arch PPC -name IRIW Rfe DpdR,LwSyncdRR Fre Rfe DpdR,LwSyncdRR Fre
Generator produced 3 tests
vlab@HYVLAB8:~/Desktop/diy7/diycross7$ ls
@all  IRIW+addrs.litmus  IRIW+lwsync+addr.litmus  IRIW+lwsyncs.litmus
vlab@HYVLAB8:~/Desktop/diy7/diycross7$ cat IRIW+lwsync+addr.litmus
PPC IRIW+lwsync+addr
"Rfe LwSyncdRR Fre Rfe DpAddrdR Fre"
Cycle=Rfe LwSyncdRR Fre Rfe DpAddrdR Fre
Generator=diycross7 (version 7.57+1)
Prefetch=1:x=F,1:y=T,3:y=F,3:x=T
Com=Rf Fr Rf Fr
Orig=Rfe LwSyncdRR Fre Rfe DpAddrdR Fre
{
0:r2=x;
1:r2=x; 1:r4=y;
2:r2=y;
3:r2=y; 3:r5=x;
}
 P0           | P1           | P2           | P3           ;
 li r1,1      | lwz r1,0(r2) | li r1,1      | lwz r1,0(r2) ;
 stw r1,0(r2) | lwsync       | stw r1,0(r2) | xor r3,r1,r1 ;
              | lwz r3,0(r4) |              | lwzx r4,r3,r5 ;
exists (1:r1=1 /\ 1:r3=0 /\ 3:r1=1 /\ 3:r4=0)
```

4.10 Identifying coherence orders with observers

- We first produce the four writes test 2+2W for Power:

  ```
  % diyone7 -name 2+2W -arch PPC PodWW Coe PodWW Coe
  ```

- Test 2+2W is the Power version of the x86 test x86-2+2W

```
vlab@HYVLAB8:~/Desktop/diy7/observers$ diyone7 -name 2+2W -arch PPC PodWW Coe PodWW Coe
vlab@HYVLAB8:~/Desktop/diy7/observers$ ls
```

```
2+2W.litmus
vlab@HYVLAB8:~/Desktop/diy7/observers$ cat 2+2W.litmus
PPC 2+2W
"PodWW Coe PodWW Coe"
Generator=diyone7 (version 7.57+1)
Prefetch=0:x=F,0:y=W,1:y=F,1:x=W
Com=Co Co
Orig=PodWW Coe PodWW Coe
{
0:r2=x; 0:r4=y;
1:r2=y; 1:r4=x;
}
 P0           | P1           ;
 li r1,2      | li r1,2      ;
 stw r1,0(r2) | stw r1,0(r2) ;
 li r3,1      | li r3,1      ;
 stw r3,0(r4) | stw r3,0(r4) ;
exists ([x]=2 /\ [y]=2)
```

4.10.1 Simple observers

Observers are additional threads introduced into a test to observe and validate the ordering of memory operations. The core idea is to monitor how loads from the same memory location are handled by the system. By executing multiple loads in sequence from the same location, observers can help determine the order in which memory operations are visible, which is essential for understanding coherence and memory consistency models.

4.10.2 How Observers Work

1. **Define the Observer Thread:** An observer thread is added to the test suite to load from a specific memory location multiple times.
2. **Monitor Coherence Orders:** By analyzing the values loaded by the observer, you can identify the coherence order of memory operations. For instance, if the observer loads a value and sees a certain sequence of values, it helps in determining the coherence order.
3. **Example of Coherence Orders:** If an observer performs a sequence of loads (e.g., load 1, then load 2) from a memory location x, and you observe the order of these loads, you can infer how memory operations are ordered (coherence order 0, 1, 2, etc.).

4.10.3 -obs Option and Its Arguments

The -obs option in the diy7 tool controls the use of observer threads in the test generation process. Observers are threads that help test and validate memory consistency models by performing loads from shared memory locations. Here's a breakdown of the -obs option and its arguments:

- **accept**: This setting allows the use of observers if they are beneficial for the test generation process. Observers will be included if they are useful for achieving the desired test cases.
- **avoid**: This is the default setting. It avoids using observers in the test generation process. The tool will generate tests without adding observer threads.
- **force**: This setting forces the inclusion of observers in all generated tests. It ensures that observers are used to validate coherence orders and memory operations, regardless of whether they are deemed necessary.
- **local**: This setting enables local observers. Local observers are used to test memory consistency at a more localized level and are not intended for broader coherence order testing.

4.10.4 -obstype Option and Its Arguments

The -obstype option in diy7 specifies the style of observers used in the test generation process. Observers are additional threads used to check memory consistency and coherence orders. Each style affects how observers are used and how they interact with the main threads. Here's a breakdown of the styles available:

- **fenced**:
  - **Description**: In the fenced style, observers are placed with memory fences or barriers around their operations. This means that the observer's actions are separated from the rest of the system by explicit synchronization points.
  - **Usage**: This style is useful when you want to ensure that the observers do not interfere with other memory operations and that their results are clearly separated from other threads' results.
- **loop**:
  - **Description**: The loop style involves observers that perform their operations in a loop, continuously accessing memory locations. This can help test how repeated accesses by observers affect the memory consistency model.
  - **Usage**: This style is useful for examining the behavior of systems under repeated load conditions and can help identify issues related to repeated access patterns.
- **straight**:

  - **Description**: In the straight style, observers perform their memory accesses in a straightforward, linear fashion without additional synchronization or looping constructs. Observers simply perform sequential loads or stores.
  - **Usage**: This style is useful for basic testing where observers perform their operations in a simple sequence, allowing you to observe basic coherence behavior without additional complexity.

```
vlab@HYVLAB8:~/Desktop/diy7/observers$  diyone7 -name 2+2WObs -obs force -obstype straight -arch PPC PodWW Coe PodWW Coe
vlab@HYVLAB8:~/Desktop/diy7/observers$ ls
2+2W.litmus  2+2WObs.litmus
vlab@HYVLAB8:~/Desktop/diy7/observers$ cat 2+2WObs.litmus
PPC 2+2WObs
"PodWW Coe PodWW Coe"
Generator=diyone7 (version 7.57+1)
Prefetch=2:x=F,2:y=W,3:y=F,3:x=W
Com=Co Co
Orig=PodWW Coe PodWW Coe
{
0:r2=x;
1:r2=y;
2:r2=x; 2:r4=y;
3:r2=y; 3:r4=x;
}
 P0           | P1           | P2           | P3           ;
 lwz r1,0(r2) | lwz r1,0(r2) | li r1,2      | li r1,2      ;
 lwz r3,0(r2) | lwz r3,0(r2) | stw r1,0(r2) | stw r1,0(r2) ;
              |              | li r3,1      | li r3,1      ;
              |              | stw r3,0(r4) | stw r3,0(r4) ;
exists (0:r1=1 /\ 0:r3=2 /\ 1:r1=1 /\ 1:r3=2)
```

5. A note on test names

The concept of "family names" in testing with diy7 helps categorize tests based on their structure, which is determined by the communication relaxations used and the sequence of read and write events in the test cycles. Here’s a detailed explanation of how family names are derived and their significance:

5.1 Understanding Family Names

Family names represent the structure of tests by summarizing the types of operations (reads and writes) and their sequence in the test cycles. These names are important because they help group similar tests together, which is useful for understanding and analyzing the behavior of memory models.

5.2 Key Points for Family Names

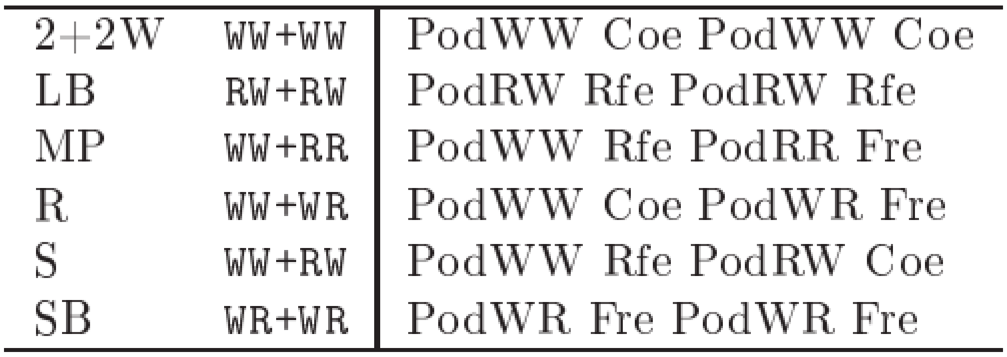
1. **External Communication Candidate Relaxations**:
   - These relaxations determine the directions of the first and last events in the threads of a test. They help in identifying the basic operations that frame the test cycles.
   - Common examples of external communication relaxations are W (write), R (read), WW (write-write), RR (read-read), RW (read-write), and WR (write-read).
2. **Determining Test Structure**:
   - The structure of the test is based on the external communication relaxations. For example, if a cycle starts and ends with writes and the other thread starts and ends with reads, the family name reflects these operations.
3. **Normalization of Family Names**:
   - To standardize family names, a minimal representation is chosen among possible representations of a cycle. This minimal representation follows a lexical order derived from the sequence of operations.
   - The lexical order is: W < WW < RR < RW < WR < R.
   - This means that if multiple representations are possible, the one that comes first lexically is selected as the family name.

Example : For instance, consider the cycle **PodWW Rfe PodRR Fre**

There are two threads in the corresponding test (as there are two external communication candidate relaxations), one thread starts and ends with a write (written WW), while the other thread starts and ends with a read (written RR). The family name is thus WW+RR, (or RR+WW, but we choose the former)

5.3 Common Nicknames and Family Names

Here is the list of nicknames and family names for two thread tests:



2+2W	WW+WW	PodWW Coe PodWW Coe
LB	RW+RW	PodRW Rfe PodRW Rfe
MP	WW+RR	PodWW Rfe PodRR Fre
R	WW+WR	PodWW Coe PodWR Fre
S	WW+RW	PodWW Rfe PodRW Coe
SB	WR+WR	PodWR Fre PodWR Fre

1. **SB (Store-Buffering)**
   - **Family Name**: WR+WR
   - **Cycle Example**: PodWR Fre PodWR Fre
   - **Description**: This family consists of two threads, each starting with a write and ending with a read. This pattern is known as store-buffering.
2. **LB (Load-Buffering)**
   - **Family Name**: RW+RW
   - **Cycle Example**: PodRW Rfe PodRW Rfe
   - **Description**: This family involves two threads, each starting with a read and ending with a write. This pattern is often referred to as load-buffering.
3. **MP (Mixed-Pattern)**
   - **Family Name**: WW+RR
   - **Cycle Example**: PodWW Rfe PodRR Fre
   - **Description**: This family includes one thread starting and ending with writes and the other starting and ending with reads. This pattern represents a mixed read-write behavior.
4. **R (Read-Write)**
   - **Family Name**: WW+WR
   - **Cycle Example**: PodWW Coe PodWR Fre
   - **Description**: This family consists of one thread starting and ending with writes and the other starting with a write and ending with a read. It highlights the interaction between read and write operations.
5. **S (Store)**
   - **Family Name**: WW+RW
   - **Cycle Example**: PodWW Rfe PodRW Coe

    - **Description**: This family includes one thread starting and ending with writes and the other starting with a write and ending with a read. This pattern is commonly associated with store behavior.

solated Writes and Reads are individual memory operations that appear in tests and can affect how we name and categorize test cycles. These isolated operations are often identified based on specific relaxation patterns in memory models. Here's a detailed explanation of how isolated writes and reads contribute to family names and nicknames:

5.4 Isolated Writes and Reads

- **Isolated Writes**: These are writes that do not immediately follow another write or are separated by different types of operations. They often appear as W in family names.
- **Isolated Reads**: These are reads that do not immediately follow another read or are separated by different types of operations. They often appear as R in family names.

Example Breakdown

Consider the cycle Rfe PodRR Fre Rfe PodRR Fre:

1. **Family Name**: W+RR+W+RR
    - **Isolated Writes**: There are two isolated writes (W), one at the beginning (Rfe PodRR) and one in the middle (Fre Rfe).
    - **Isolated Reads**: Reads are grouped with other operations to form RR pairs.
2. **Cycle Explanation**:
    - **Initial State**: Rfe (read from an external source)
    - **Operations**:
        - PodRR (an operation that can be classified under RR)
        - Fre (an operation that introduces a fresh state)
        - Rfe (another read from an external source)
        - PodRR (another RR operation)
    - The sequence shows isolated writes (W) and read pairs (RR), leading to the family name W+RR+W+RR.
3. **Nickname**: IRIW (Independent Reads of Independent Writes)
    - **Description**: The nickname IRIW reflects the pattern where reads are independent of the writes, indicating a scenario where reads and writes are isolated from each other

note : Typically involve operations where a thread performs a write that is independent of other operations in the test cycle. For example, Fre (fresh state) or Pod (partial order dependency) can result in isolated writes.

6. Additional tools: extracting cycles and classification

When non-standard family names or numeric names are used, it proves convenient to rename tests with the standard naming scheme. We provide two tools to do so: mcycle7 that extracts cycles from litmus source les and classify7 that normalises and renames cycles

6.1. mcycles7

Purpose: Extracts cycles from litmus source files.

Usage: This tool processes litmus source files to extract the memory access patterns (cycles) used in the tests. It helps in identifying and analyzing the cycles defined in the source files.

6.2. classify7

Purpose: Normalizes and renames cycles according to a standard naming scheme.

Usage: This tool takes the cycles extracted by mcycle7 or directly from litmus files and assigns them standard names according to predefined naming conventions. It helps in organizing and standardizing test names for consistency.

The tool classify7 reads its standard input, interpreting is as a list of cycles in the output format of mcycle7. It normalises and classies those cycles. The tool classify7 accepts the following documented options:

-arch (X86|PPC|ARM|AArch64|RISCV|LISA|C) Set architecture. Default is PPC.

-bell <name> Read bell le <name>, implies -arch LISA.

-lowercase <bool> Use lowercase familly names, default false.

-u Instruct classify7 to fail when two tests have the same normalised name. Otherwise classify7 will output one line per test, regardless of duplicate names.

-diyone Output a normalised list of names and cycles, which is legal input for diyone7.

6.3 Usage of mcycles7 and classify7 on generated litmus tests

- locate whether mcycles7 package is present or not

```
vlab@HYVLAB8:~/sudo apt install mlocate
vlab@HYVLAB8:~/Desktop/diy7/readRelax$ locate mcycles7
/home/vlab/.opam/4.02.3/bin/mcycles7
/home/vlab/.opam/default/bin/mcycles7
/home/vlab/Desktop/herdtools7/_build/install/default/bin/mcycles7
```

- use mcycles7 on a litmus test to extract the memory access pattern

```
vlab@HYVLAB8:~/Desktop/diy7/readRelax$ mcycles7 test_suite000.litmus
test_suite000: PodWR Fre PodWR Fre
vlab@HYVLAB8:~/Desktop/diy7$ mkdir mcycles7-classify7
vlab@HYVLAB8:~/Desktop/diy7$ cd mcycles7-classify7/
```

- create a conf file using which we generate the tests using diy7 tool

```
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ gvim X.conf
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ cat X.conf
-arch X86
-name X
-nprocs 3
-size 6
-safe Pod**,Fre,Rfe,Wse
-mode critical

vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ diy7 -conf X.conf
Generator produced 23 tests
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ ls
@all         X004.litmus  X009.litmus  X014.litmus  X019.litmus
X000.litmus  X005.litmus  X010.litmus  X015.litmus  X020.litmus
X001.litmus  X006.litmus  X011.litmus  X016.litmus  X021.litmus
X002.litmus  X007.litmus  X012.litmus  X017.litmus  X022.litmus
```

```
X003.litmus  X008.litmus  X013.litmus  X018.litmus  X.conf
```

- use mcycles7 for extracting memory access patterns of all tests at once

```
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ mcycles7 @all
X000: Rfe PodRR Fre PodWR Fre
X001: Rfe PodRW Coe PodWR Fre
X002: Rfe PodRW Rfe PodRR Fre
X003: Rfe PodRR Fre PodWW Coe
X004: Rfe PodRW Coe PodWW Coe
X005: Rfe PodRW Rfe PodRW Coe
X006: PodWW Rfe PodRR Fre PodWR Fre
X007: PodWW Rfe PodRW Coe PodWR Fre
X008: PodWW Rfe PodRR Fre
X009: PodWW Rfe PodRW Rfe PodRR Fre
X010: PodWW Coe PodWW Rfe PodRR Fre
X011: PodWW Coe PodWW Rfe PodRW Coe
X012: PodWW Rfe PodRW Coe
X013: PodWW Rfe PodRW Rfe PodRW Coe
X014: PodRW Rfe PodRW Rfe
X015: PodRW Rfe PodRW Rfe PodRW Rfe
X016: PodWR Fre PodWR Fre
X017: PodWR Fre PodWR Fre PodWR Fre
X018: PodWW Coe PodWR Fre PodWR Fre
X019: PodWW Coe PodWR Fre
X020: PodWW Coe PodWW Coe PodWR Fre
X021: PodWW Coe PodWW Coe
X022: PodWW Coe PodWW Coe PodWW Coe
```

- The tool mcycle7 has no options and takes litmus source les or index les as arguments. It outputs a list of lines to standard output. Each line starts with a test name, suxed by :, then the cycle of the named test.

```
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ classify7 @all
classify7: No argument.
Usage classify7 [options] [arg]*
```

```
options are:
  -diyone  generate input for diyone
  -lowercase <bool> use lowercase familly names, default false
  -u  reject duplicate normalised names
  -map  <name> save renaming map into file <name>
  -bell <name> read bell file <name>
    -arch  <C|CPP|LISA|JAVA|ASL|AArch64|ARM|BPF|MIPS|PPC|X86|RISCV|X86_64>  specify architecture
  -help  Display this list of options
  --help  Display this list of options
```

- The output of mcycle7 can be piped into classify7 for family classication:

vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ mcycles7 @all | classify7 -arch X86

```
2+2W
  X021 -> 2+2W: PodWW Coe PodWW Coe
3.2W
  X022 -> 3.2W: PodWW Coe PodWW Coe PodWW Coe
3.LB
  X015 -> 3.LB: PodRW Rfe PodRW Rfe PodRW Rfe
3.SB
  X017 -> 3.SB: PodWR Fre PodWR Fre PodWR Fre
ISA2
  X009 -> ISA2: PodWW Rfe PodRW Rfe PodRR Fre
LB
  X014 -> LB: PodRW Rfe PodRW Rfe
MP
  X008 -> MP: PodWW Rfe PodRR Fre
R
  X019 -> R: PodWW Coe PodWR Fre
RWC
  X000 -> RWC: Rfe PodRR Fre PodWR Fre
S
  X012 -> S: PodWW Rfe PodRW Coe
SB
  X016 -> SB: PodWR Fre PodWR Fre
```

```
W+RWC
  X006 -> W+RWC: PodWW Rfe PodRR Fre PodWR Fre
WRC
  X002 -> WRC: Rfe PodRW Rfe PodRR Fre
WRR+2W
  X003 -> WRR+2W: Rfe PodRR Fre PodWW Coe
WRW+2W
  X004 -> WRW+2W: Rfe PodRW Coe PodWW Coe
WRW+WR
  X001 -> WRW+WR: Rfe PodRW Coe PodWR Fre
WWC
  X005 -> WWC: Rfe PodRW Rfe PodRW Coe
Z6.0
  X007 -> Z6.0: PodWW Rfe PodRW Coe PodWR Fre
Z6.1
  X011 -> Z6.1: PodWW Coe PodWW Rfe PodRW Coe
Z6.2
  X013 -> Z6.2: PodWW Rfe PodRW Rfe PodRW Coe
Z6.3
  X010 -> Z6.3: PodWW Coe PodWW Rfe PodRR Fre
Z6.4
  X018 -> Z6.4: PodWW Coe PodWR Fre PodWR Fre
Z6.5
  X020 -> Z6.5: PodWW Coe PodWW Coe PodWR Fre
```

- Notice that classify7 accepts the arch option, as it needs to parse cycles.

- Finally, one can normalise tests, using normalised names by piping mcycle7 output into diyone7 with options -norm -num false:
- Use the -norm and -num false options for diyone7 to normalize the names.

```
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ mkdir normalized_op
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ mcycles7 @all | diyone7 -arch X86 -norm -num false -o normalized_op
Generator produced 23 tests
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ cd normalized_op/
```

```
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7/normalized_op$ ls
2+2W.litmus  ISA2.litmus  SB.litmus      W+RWC.litmus   Z6.2.litmus
3.2W.litmus  LB.litmus    S.litmus       WRW+WR.litmus  Z6.3.litmus
3.LB.litmus  MP.litmus    WRC.litmus     WWC.litmus     Z6.4.litmus
3.SB.litmus  R.litmus     WRR+2W.litmus  Z6.0.litmus    Z6.5.litmus
@all         RWC.litmus   WRW+2W.litmus  Z6.1.litmus
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7/normalized_op$ cd ..
```

- Alternatively, one may instruct classify7 to produce output for diyone7. In that case one should pass option -diyone to classify7 so as to instruct it to produce output that is parsable by diyone7:

```
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ rm -r normalized_op/ && mkdir normalized_op/
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ mcycles7 @all | classify7 -arch X86 -diyone | diyone7 -arch X86 -o normalized_op/
Generator produced 23 tests
vlab@HYVLAB8:~/Desktop/diy7/mcycles7-classify7$ ls normalized_op/
2+2W.litmus  ISA2.litmus  SB.litmus      W+RWC.litmus   Z6.2.litmus
3.2W.litmus  LB.litmus    S.litmus       WRW+WR.litmus  Z6.3.litmus
3.LB.litmus  MP.litmus    WRC.litmus     WWC.litmus     Z6.4.litmus
3.SB.litmus  R.litmus     WRR+2W.litmus  Z6.0.litmus    Z6.5.litmus
@all         RWC.litmus   WRW+2W.litmus  Z6.1.litmus
```

7. Simulating memory models with herd7

The tool herd7 is a memory model simulator. Users may write simple, single events, axiomatic models of their own and run litmus tests on top of their model. The herd7 distribution already includes some models.
The authors of herd7 are Jade Alglave and Luc Maranget

7.1 Sequential consistency

The simulator herd7 accepts models written in text les. For instance here is sc.cat, the denition of the

sequentially consistent (SC) model in the partial-order style: SC.cat

```
SC
include "fences.cat"
include "cos.cat"
(* Atomic *)
empty rmw & (fre;coe) as atom
(* Sequential consistency *)
show sm\id as si
acyclic po | ((fr | rf | co);sm) as sc
```

The model above illustrates some features of model definitions:

1. A model file starts with a tag (here SC), which can also be a string (in double quotes) in case the tag includes special characters or spaces.
2. Pre-defined bindings. Here po (program order) and rf (read from) are pre-defined. The remaining two communication relations (co and fr) are computed by the included le cos.cat, For simplicity, we may as well assume that co and fr are pre-dened.
3. The computation of new relations from other relations, and their binding to a name with the let construct. Here, a new relation com is the union | of the three pre-dened communication relations.
4. The performance of some checks. Here the relation po | com (i.e. the union of program order po and of communication relations) is required to be acyclic. Checks can be given names by suffixing them with a name.

```
vlab@HYVLAB8:~/Desktop/herd7$ herd7 -model ./SC.cat test.litmus
File "_none_", line 1, characters -1--1: Cannot find file stdlib.cat
```

```
vlab@HYVLAB8:~/Desktop/herd7$ find / -name "stdlib.cat" 2>/dev/null
/home/vlab/.opam/default/.opam-switch/sources/herdtools7.7.57/herd/libdir/stdlib.cat
/home/vlab/.opam/default/share/herdtools7/herd/stdlib.cat
/home/vlab/.opam/4.02.3/.opam-switch/sources/herdtools7.7.52/herd/libdir/stdlib.cat
/home/vlab/.opam/4.02.3/share/herdtools7/herd/stdlib.cat
```

```
/home/vlab/Desktop/herdtools7/herd/libdir/stdlib.cat
/home/vlab/Desktop/herdtools7/herd-www/cat_includes/stdlib.cat
```
vlab@HYVLAB8:~/Desktop/herd7$ herd7 -I /home/vlab/.opam/default/share/herdtools7/herd -model ./SC.cat test.litmus
```
Test SB Allowed
States 3
0:EAX=0; 1:EAX=1; [x]=1; [y]=1;
0:EAX=1; 1:EAX=0; [x]=1; [y]=1;
0:EAX=1; 1:EAX=1; [x]=1; [y]=1;
No
Witnesses
Positive: 0 Negative: 3
Condition exists (0:EAX=0 /\ 1:EAX=0)
Observation SB Never 0 3
Time SB 0.00
Hash=2d53e83cd627ba17ab11c875525e078b
```

The output of herd7 mainly consists in the list of final states that are allowed by the simulated model. Additional output relates to the test condition. One sees that the test condition does not validate on top of SC, as No appears just after the list of final states and as there is no Positive witness. Namely, the condition exists (0:EAX=0 /\ 1:EAX=0) reflects a non-SC behaviour

7.2 Total Store Order (TSO)

However, the non-SC execution shows up on x86 machines, whose memory model is TSO. As TSO relaxes
the write-to-read order, we attempt to write a TSO model tso-00.cat, by simply removing write-to-read
pairs from the acyclicity check:

```
"A first attempt for TSO"
include "cos.cat"
(* Communication relations that order events*)
let com-tso = rf | co | fr
```

(* Program order that orders events *)
let po-tso = po & (W*W | R*M)
(* TSO global-happens-before *)
let ghb = po-tso | com-tso
acyclic ghb as tso
show ghb

This model illustrates several features of model definitions:

- New predefined sets: W, R and M, which are the sets of write events, read events and of memory events,
  respectively.
- The cartesian product operator * that returns the cartesian product of two event sets as a relation.
- The intersection operator & that operates on sets and relations.

Include Statement:
include "cos.cat"

- This line includes the file cos.cat, which presumably defines additional relations and constructs used in the model. It is necessary for defining communication relations like rf (read-from), co (coherence order), and fr (from-read).

Communication Relations:
let com-tso = rf | co | fr

- Defines a new relation com-tso that combines the read-from (rf), coherence order (co), and from-read (fr) relations. This represents the communication relations that affect the TSO model.

Program Order:
let po-tso = po & (W*W | R*M)

- Defines a new program order relation po-tso. This combines the existing program order po with a condition that involves write-write (W*W) and read-modify (R*M) interactions. This is a typical way to model program order constraints specific to TSO.

Global Happens-Before (GHB):

```
let ghb = po-tso | com-tso
```

- Defines the ghb (global-happens-before) relation as the union of po-tso (program order specific to TSO) and com-tso (communication relations). This relation represents the total order of events in TSO.

Acyclic Check:

```
acyclic ghb as tso
```

- Checks that the ghb relation is acyclic, meaning there are no cycles in the global happens-before order, which is essential for the validity of the model. The result of this check is named tso.

Show Statement:

```
show ghb
```

- Displays the ghb relation, which provides information about the global happens-before relation defined by the model.

```
vlab@HYVLAB8:~/Desktop/herd7$ gvim tso.cat
vlab@HYVLAB8:~/Desktop/herd7$ herd7 -I /home/vlab/.opam/default/share/herdtools7/herd
-model ./tso.cat test.litmus
Test SB Allowed
States 4
0:EAX=0; 1:EAX=0; [x]=1; [y]=1;
0:EAX=0; 1:EAX=1; [x]=1; [y]=1;
0:EAX=1; 1:EAX=0; [x]=1; [y]=1;
0:EAX=1; 1:EAX=1; [x]=1; [y]=1;
Ok
Witnesses
Positive: 1 Negative: 3
Condition exists (0:EAX=0 /\ 1:EAX=0)
Observation SB Sometimes 1 3
Time SB 0.00
Hash=2d53e83cd627ba17ab11c875525e078b
```

7.3 Producing pictures of executions

- The simulator herd7 can be instructed to produce pictures of executions. Those pictures are instrumental in understanding and debugging models. It is important to understand that herd7 does not produce pictures by default.
- To get pictures one must instruct herd7 to produce pictures of some executions with the -show option. This option accepts specific keywords, its default being none, instructing herd7 not to produce any picture.
- A frequently used keyword is prop that means show the executions that validate the proposition in the final condition. Namely, the final condition in the litmus test is a quantified boolean proposition as for instance exists (0:EAX=0 /\ 1:EAX=0) at the end of test SB.
- But this is not enough, users also have to specify what to do with the picture: save it in le in the DOT format of the graphviz graph visualization software, or display the image,15 or both.
- One instructs herd7 to save images with the -o dirname option, where dirname is the name of a directory, which must exist. Then, when processing the le name.litmus, herd7 will create a le name.dot into the directory dirname

As an example, so as to display the image of the non-SC behaviour of SB, one should invoke herd7 as:

```
vlab@HYVLAB8:~/Desktop/herd7$ herd7 -I /home/vlab/.opam/default/share/herdtools7/herd
-model tso2.cat -show prop -dotmode plain -o img test.litmus
```

```
Test SB Allowed
States 4
0:EAX=0; 1:EAX=0; [x]=1; [y]=1;
0:EAX=0; 1:EAX=1; [x]=1; [y]=1;
0:EAX=1; 1:EAX=0; [x]=1; [y]=1;
0:EAX=1; 1:EAX=1; [x]=1; [y]=1;
Ok
Witnesses
Positive: 1 Negative: 3
Condition exists (0:EAX=0 /\ 1:EAX=0)

Observation SB Sometimes 1 3

Time SB 0.00
Hash=2d53e83cd627ba17ab11c875525e078b
```

- This command analyzes the test.litmus file using the tso2.cat model and saves the output in .dot format in the img directory.

-I /home/vlab/.opam/default/share/herdtools7/herd: Adds the specified directory to the search path for model files.

-model tso2.cat: Specifies the model file to use for checking the test.

-show prop: Instructs herd7 to display properties of the test.

-dotmode plain: Specifies that the output should be in plain .dot format.

-o img: Sets the output directory to img where the generated files will be stored.

test.litmus: The litmus test file to analyze.

```
vlab@HYVLAB8:~/Desktop/herd7$ cd img/
vlab@HYVLAB8:~/Desktop/herd7/img$ ls
test.dot
vlab@HYVLAB8:~/Desktop/herd7/img$ dot -Tpng test.dot -o test.png
```

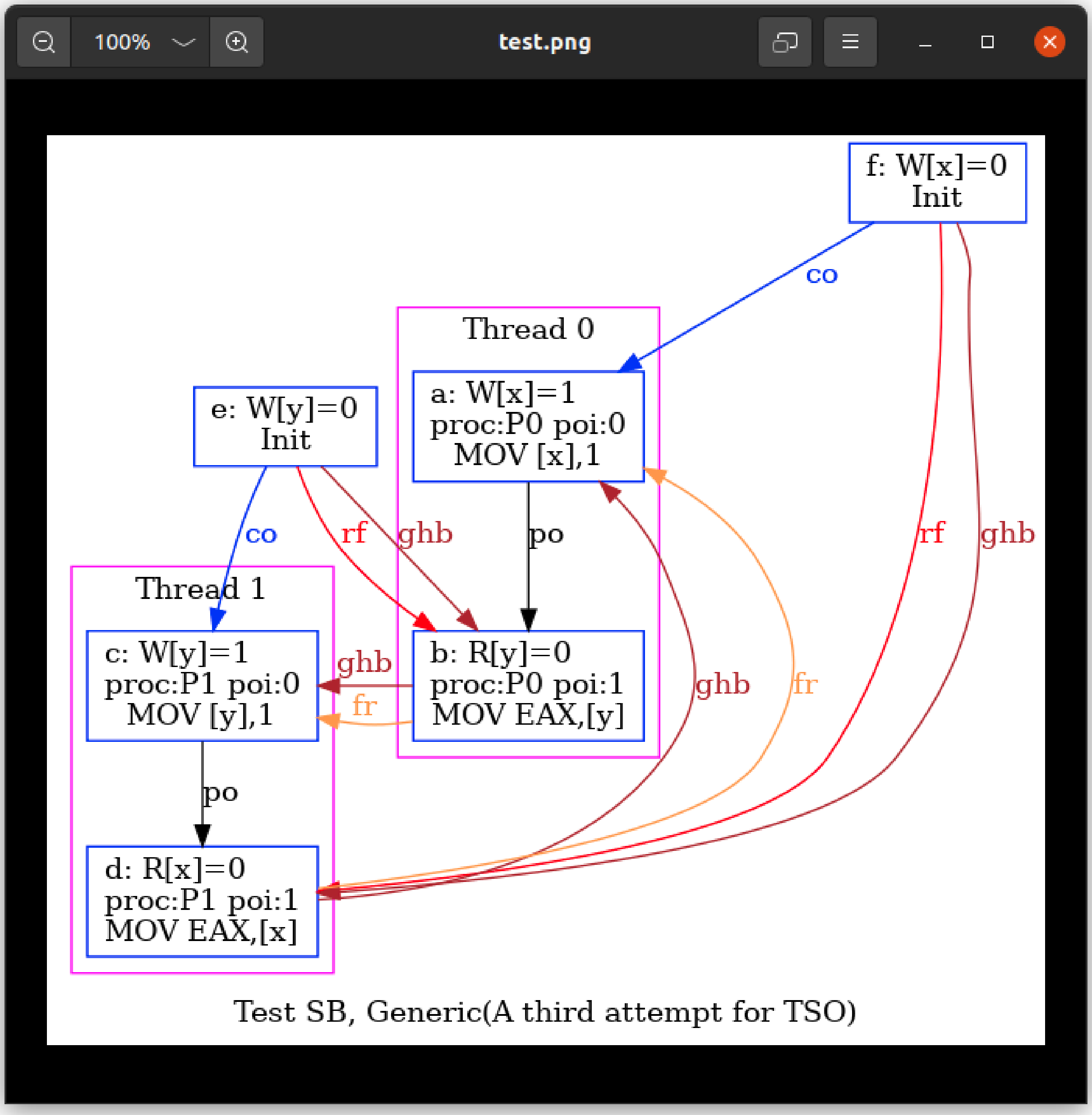
This command converts the .dot file into a PNG image format.

- **dot:** The Graphviz tool for processing .dot files.
- **-Tpng:** Specifies that the output format should be PNG.
- **test.dot:** The input .dot file to be converted.
- **-o test.png:** Specifies the output file name for the converted PNG image.

```
vlab@HYVLAB8:~/Desktop/herd7/img$ xdg-open test.png
```

This command tries to open the test.png image using the default image viewer.

- **xdg-open:** Opens a file or URL in the default application associated with the file type.
- **test.png:** The PNG file you want to open.



7.4 Graph modes

herd7 can produce three styles of pictures, dot clustered pictures, dot free pictures, and neato pictures with explicit placement of the events of one thread as a column. The style is commanded by the -graph option that accepts three possible arguments: **cluster (default), free and columns**

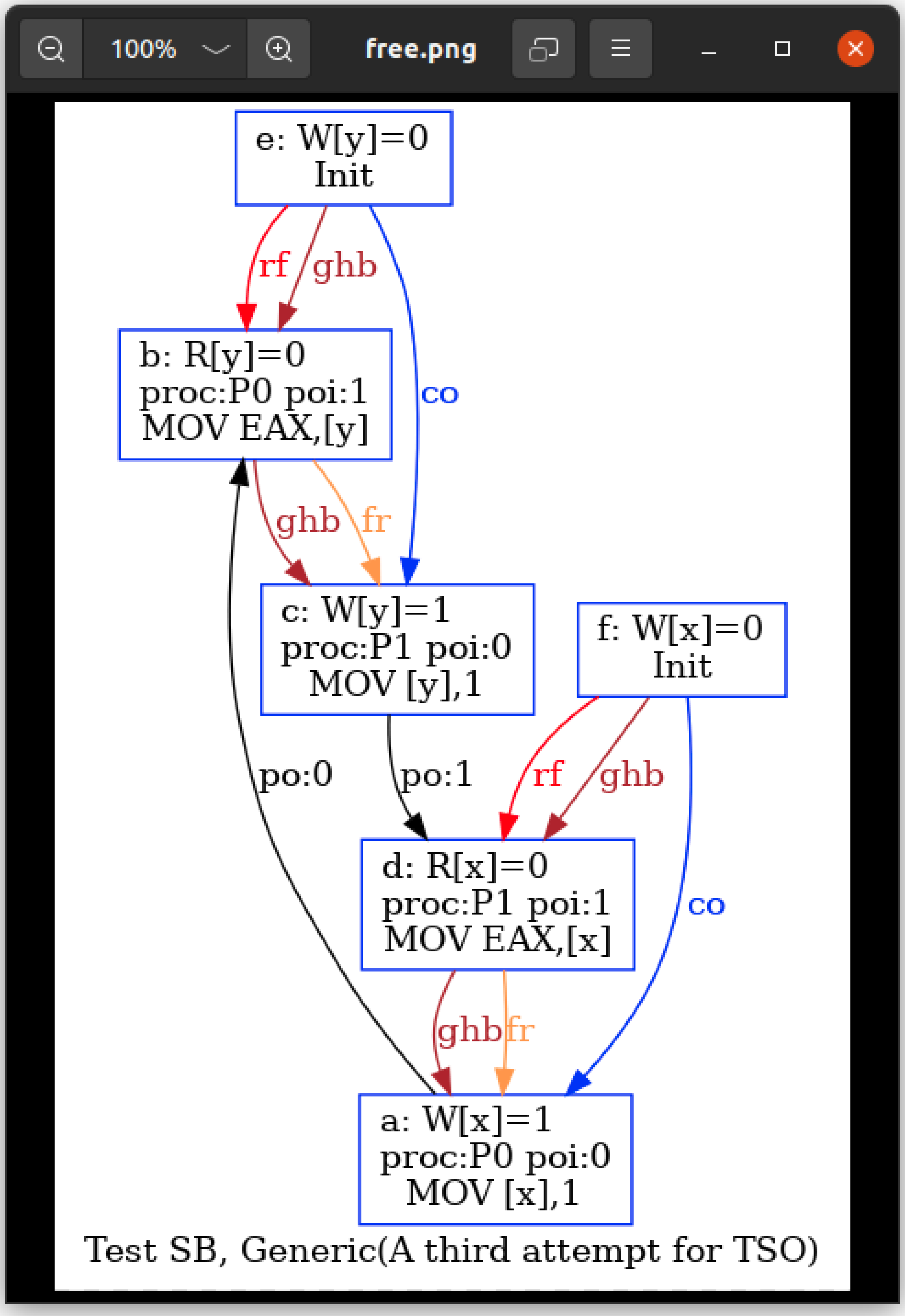
7.4.1 -graph free

```
vlab@HYVLAB8:~/Desktop/herd7$ herd7 -I /home/vlab/.opam/default/share/herdtools7/herd -model tso2.cat -show prop -dotmode plain -graph free -o img test.litmus
```

Test SB Allowed

```
States 4
0:EAX=0; 1:EAX=0; [x]=1; [y]=1;
0:EAX=0; 1:EAX=1; [x]=1; [y]=1;
0:EAX=1; 1:EAX=0; [x]=1; [y]=1;
0:EAX=1; 1:EAX=1; [x]=1; [y]=1;
Ok
Witnesses
Positive: 1 Negative: 3
Condition exists (0:EAX=0 /\ 1:EAX=0)
Observation SB Sometimes 1 3
Time SB 0.00
Hash=2d53e83cd627ba17ab11c875525e078b
```

```
vlab@HYVLAB8:~/Desktop/herd7$ ls
img  output.txt  SC.cat  test.litmus  tso2.cat  tso.cat
vlab@HYVLAB8:~/Desktop/herd7$ cd img
vlab@HYVLAB8:~/Desktop/herd7/img$ ls
test.dot  test.png
vlab@HYVLAB8:~/Desktop/herd7/img$ mv test.dot free.dot
vlab@HYVLAB8:~/Desktop/herd7/img$ ls
free.dot  test.png
vlab@HYVLAB8:~/Desktop/herd7/img$ dot -Tpng free.dot -o free.png
vlab@HYVLAB8:~/Desktop/herd7/img$ xdg-open free.png
```


100%
free.png
e: W[y]=0
Init
rf
ghb
co
b: R[y]=0
proc:P0 poi:1
MOV EAX,[y]
ghb
fr
c: W[y]=1
proc:P1 poi:0
MOV [y],1
f: W[x]=0
Init
po:0
po:1
rf
ghb
d: R[x]=0
proc:P1 poi:1
MOV EAX,[x]
co
ghb
fr
a: W[x]=1
proc:P0 poi:0
MOV [x],1
Test SB, Generic(A third attempt for TSO)

7.4.2 -graph columns

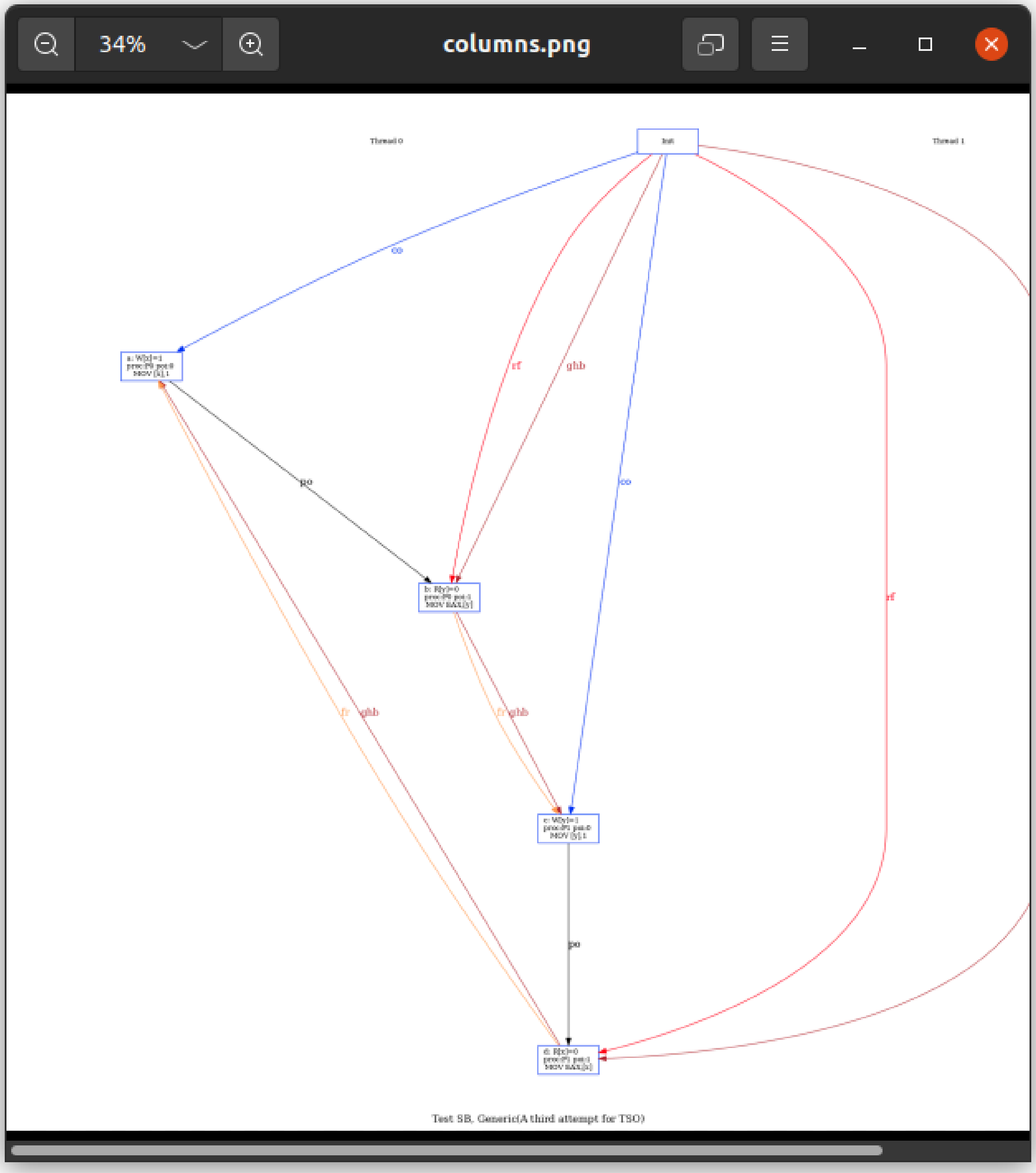
```
vlab@HYVLAB8:~/Desktop/herd7$ herd7 -I /home/vlab/.opam/default/share/herdtools7/herd
-model tso2.cat -show prop -dotmode plain -graph columns -o img test.litmus
Test SB Allowed
States 4
0:EAX=0; 1:EAX=0; [x]=1; [y]=1;
0:EAX=0; 1:EAX=1; [x]=1; [y]=1;
0:EAX=1; 1:EAX=0; [x]=1; [y]=1;
0:EAX=1; 1:EAX=1; [x]=1; [y]=1;
Ok
Witnesses
Positive: 1 Negative: 3
Condition exists (0:EAX=0 /\ 1:EAX=0)
Observation SB Sometimes 1 3
Time SB 0.00
Hash=2d53e83cd627ba17ab11c875525e078b

vlab@HYVLAB8:~/Desktop/herd7$ cd img
vlab@HYVLAB8:~/Desktop/herd7/img$ ls
free.dot  free.png  test.dot  cluster.png
vlab@HYVLAB8:~/Desktop/herd7/img$ mv test.dot columns.dot
vlab@HYVLAB8:~/Desktop/herd7/img$ ls
cluster.png  columns.dot  free.dot  free.png
vlab@HYVLAB8:~/Desktop/herd7/img$ dot -Tpng columns.dot -o columns.png
Warning: node 'proc0_label_node', graph 'G' size too small for label
Warning: node 'proc1_label_node', graph 'G' size too small for label
vlab@HYVLAB8:~/Desktop/herd7/img$ cat columns.dot
digraph G {
/* legend */
label="Test SB, Generic(A third attempt for TSO)";
/* init events */
eiidinit [label="Init", shape="box", color="blue"];
/* the unlocked events */
proc0_label_node [shape=none, label="Thread 0", pos="1.000000,1.700003!", fixedsize=true,
width=0.650000, height=0.187500]
```

```
eiid0 [label="a: W[x]=1\lproc:P0 poi:0\lMOV [x],1", shape="box", color="blue"];
eiid1 [label="b: R[y]=0\lproc:P0 poi:1\lMOV EAX,[y]", shape="box", color="blue"];
proc1_label_node [shape=none, label="Thread 1", pos="2.000000,1.700003!", fixedsize=true,
width=0.650000, height=0.187500]
eiid2 [label="c: W[y]=1\lproc:P1 poi:0\lMOV [y],1", shape="box", color="blue"];
eiid3 [label="d: R[x]=0\lproc:P1 poi:1\lMOV EAX,[x]", shape="box", color="blue"];

/* the intra_causality_data edges */

/* the intra_causality_control edges */

/* the poi edges */
eiid0 -> eiid1 [label="po", color="black", fontcolor="black"];
eiid2 -> eiid3 [label="po", color="black", fontcolor="black"];
/* the rfmap edges */
eiidinit -> eiid1 [label="rf", color="red", fontcolor="red"];
eiidinit -> eiid3 [label="rf", color="red", fontcolor="red"];


/* The viewed-before edges */
eiid1 -> eiid2 [label="ghb", color="brown", fontcolor="brown"];
eiid3 -> eiid0 [label="ghb", color="brown", fontcolor="brown"];
eiidinit -> eiid1 [label="ghb", color="brown", fontcolor="brown"];
eiidinit -> eiid3 [label="ghb", color="brown", fontcolor="brown"];
eiid1 -> eiid2 [label="fr", color="#ffa040", fontcolor="#ffa040"];
eiid3 -> eiid0 [label="fr", color="#ffa040", fontcolor="#ffa040"];
eiidinit -> eiid2 [label="co", color="blue", fontcolor="blue"];
eiidinit -> eiid0 [label="co", color="blue", fontcolor="blue"];
}
vlab@HYVLAB8:~/Desktop/herd7/img$ gvim columns.dot
vlab@HYVLAB8:~/Desktop/herd7/img$ cat columns.dot
digraph G {
graph [size="20,20", ratio="fill"];
node [fontsize=10, width=1.2, height=0.5];
```

proc0_label_node [shape=none, label="Thread 0", pos="1.000000,1.700003!", width=1.0, height=0.2];

……..

```
vlab@HYVLAB8:~/Desktop/herd7/img$ dot -Tpng columns.dot -o columns.png
vlab@HYVLAB8:~/Desktop/herd7/img$ xdg-open columns.png
```



Test SB, Generic(A third attempt for TSO)

7.4.3 -graph cluster

It is a default mode of the graph option which is produce image same as the figure in 7.3 section

7.5 Configuration files

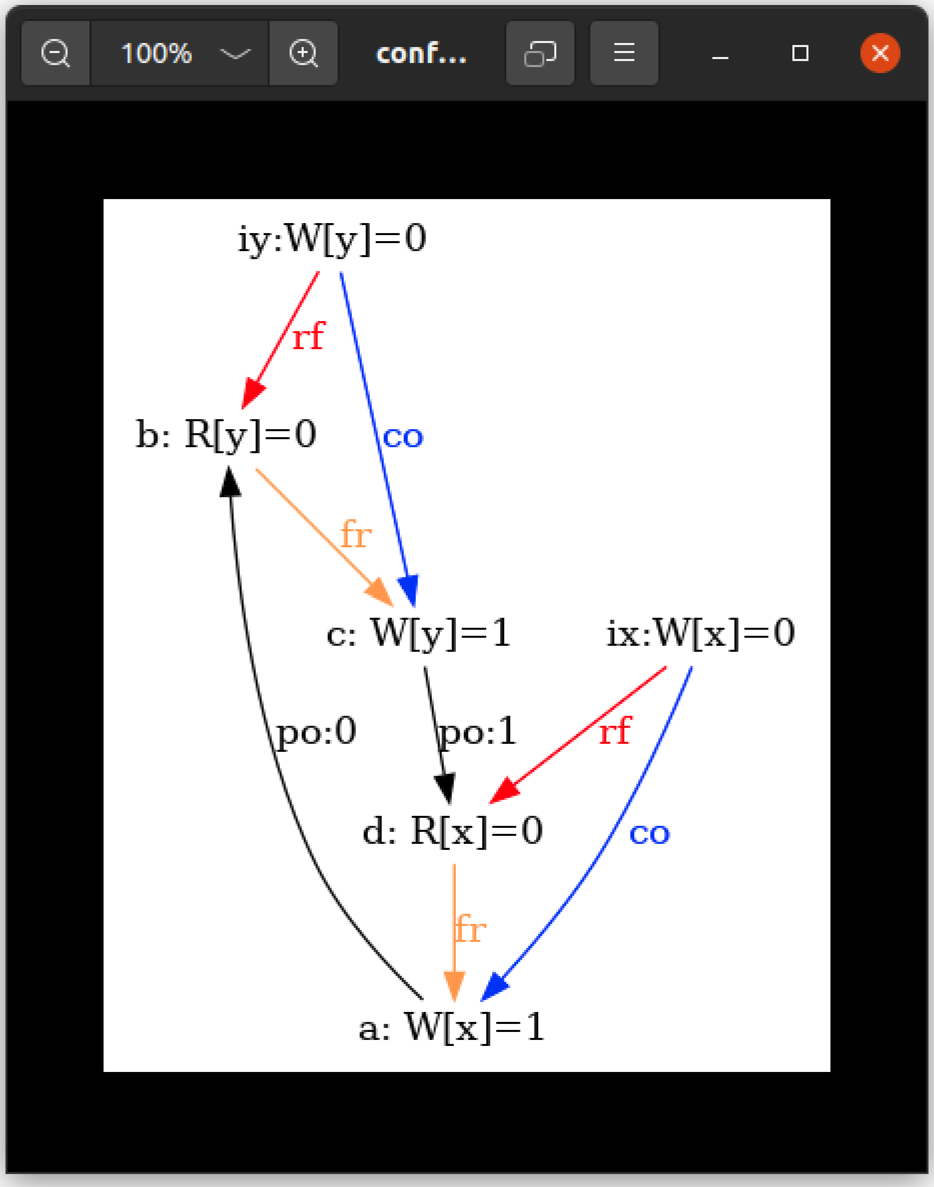
The syntax of configuration files is minimal: lines key arg are interpreted as setting the value of parameter key to arg. Each parameter has a corresponding option, usually -key, except for the single letter option -v whose parameter is verbose.
As command line options are processed left-to-right, settings from a configuration file (option -conf) can be overridden by a later command line option. Configuration les will be used mostly for controlling pictures. Some configuration files are present in the distribution.
As an example, here is the configuration le conf_file.conf, which can be used to display images in free mode.

```
#Main graph mode
graph free
#Show memory events only
showevents memory
#Minimal information in nodes
squished true
#Do not show a legend at all
showlegend false
```

```
vlab@HYVLAB8:~/Desktop/herd7$ herd7 -I /home/vlab/.opam/default/share/herdtools7/herd -conf free.conf -doshow prop -show prop -o img test.litmus
Test SB Allowed
States 4
0:EAX=0; 1:EAX=0; [x]=1; [y]=1;
0:EAX=0; 1:EAX=1; [x]=1; [y]=1;
0:EAX=1; 1:EAX=0; [x]=1; [y]=1;
0:EAX=1; 1:EAX=1; [x]=1; [y]=1;
Ok
Witnesses
Positive: 1 Negative: 3
Condition exists (0:EAX=0 /\ 1:EAX=0)
Observation SB Sometimes 1 3
Time SB 0.00
```

Hash=2d53e83cd627ba17ab11c875525e078b

```
vlab@HYVLAB8:~/Desktop/herd7$ ls
free.conf  img  output.txt  SC.cat  test.litmus  tso2.cat  tso.cat
vlab@HYVLAB8:~/Desktop/herd7$ cd img
vlab@HYVLAB8:~/Desktop/herd7/img$ ls
cluster.png  columns.dot  columns.png  free.dot  free.png  test.dot
vlab@HYVLAB8:~/Desktop/herd7/img$ mv test.dot conf_fig.dot
vlab@HYVLAB8:~/Desktop/herd7/img$ dot -Tpng conf_fig.dot -o conf_fig.png
vlab@HYVLAB8:~/Desktop/herd7/img$ ls
cluster.png  columns.dot  columns.png  conf_fig.dot  conf_fig.png  free.dot  free.png
vlab@HYVLAB8:~/Desktop/herd7/img$ xdg-open conf_fig.png
```


100%
conf...
iy:W[y]=0
rf
co
b: R[y]=0
fr
c: W[y]=1
ix:W[x]=0
po:0
po:1
rf
d: R[x]=0
co
fr
a: W[x]=1