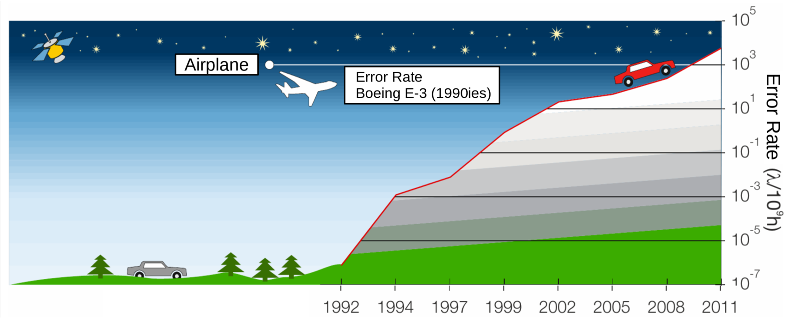
Data-Flow–Sensitive Fault-Space Pruning for the Injection of Transient Hardware Faults

Oskar Pusz, Christian Dietrich, Daniel Lohmann

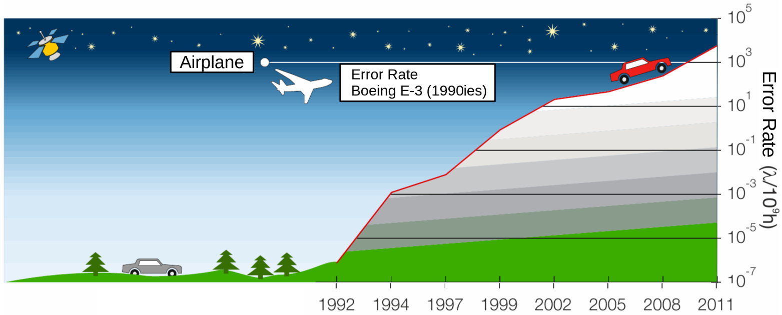
June 22, 2021

Motivation



Airplane
Error Rate
Boeing E-3 (1990ies)
Error Rate (λ/10^9h)
10^5
10^3
10^1
10^-1
10^-3
10^-5
10^-7
1992
1994
1997
1999
2002
2005
2008
2011

Motivation



"Toyota claimed the 2005 Camry's main CPU had error detecting and correcting RAM. ***It didn't.****"*

Source: Investigation Report, EDN Network, 28 Oct 2013

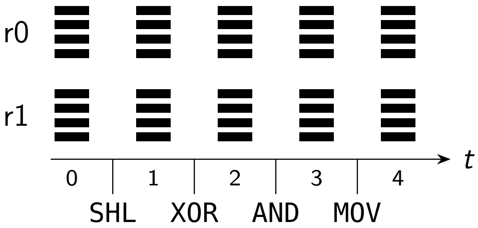
The Fault Space

- Fault injection campaign for a given program (execution)
  - **FM**: Uniformly-distributed soft errors in registers and memory
  - **Goal:** Quantify the failure-behavior of a single program execution.

The Fault Space

- Fault injection campaign for a given program (execution)
  - **FM**: Uniformly-distributed soft errors in registers and memory
  - **Goal:** Quantify the failure-behavior of a single program execution.

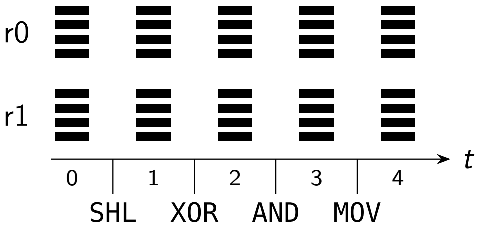
```
// initial r0=5, r1=11
//////////////////////
// shift-left by 1
r0 := SHL r0, 1  //r0=10
// bit-wise XOR with 7
r1 := XOR r1, 7  //r1=12
// bit-wise AND
r1 := AND r0, r1 //r1=8
// move result to r0
r0 := MOV r1     //r0=8
// result in r0
```



The Fault Space

- Fault injection campaign for a given program (execution)
  - **FM**: Uniformly-distributed soft errors in registers and memory
  - **Goal:** Quantify the failure-behavior of a single program execution.

```
// initial r0=5, r1=11
//////////////////////
// shift-left by 1
r0 := SHL r0, 1  //r0=10
// bit-wise XOR with 7
r1 := XOR r1, 7  //r1=12
// bit-wise AND
r1 := AND r0, r1 //r1=8
// move result to r0
r0 := MOV r1     //r0=8
// result in r0
```

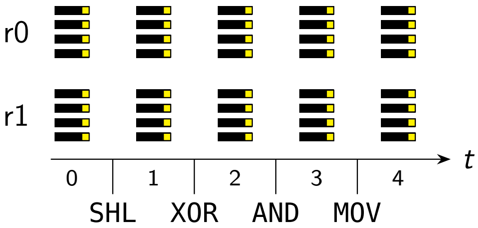


- Plan and inject!
  - Record a fault-free execution of the program-under-test.

The Fault Space

- Fault injection campaign for a given program (execution)
  - **FM**: Uniformly-distributed soft errors in registers and memory
  - **Goal:** Quantify the failure-behavior of a single program execution.

```
// initial r0=5, r1=11
//////////////////////
// shift-left by 1
r0 := SHL r0, 1  //r0=10
// bit-wise XOR with 7
r1 := XOR r1, 7  //r1=12
// bit-wise AND
r1 := AND r0, r1 //r1=8
// move result to r0
r0 := MOV r1     //r0=8
// result in r0
```

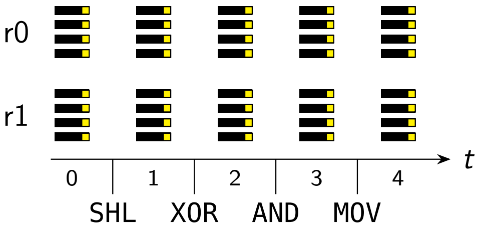


- Plan and inject!
  - Record a fault-free execution of the program-under-test.
  - Inject every memory location in each processor cycle.

The Fault Space

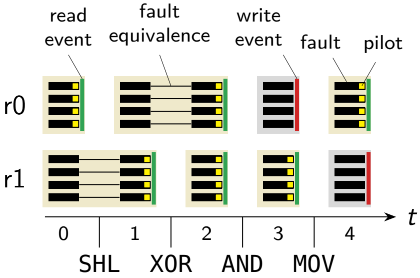
- Fault injection campaign for a given program (execution)
  - **FM**: Uniformly-distributed soft errors in registers and memory
  - **Goal:** Quantify the failure-behavior of a single program execution.

```
// initial r0=5, r1=11
//////////////////////
// shift-left by 1
r0 := SHL r0, 1  //r0=10
// bit-wise XOR with 7
r1 := XOR r1, 7  //r1=12
// bit-wise AND
r1 := AND r0, r1 //r1=8
// move result to r0
r0 := MOV r1     //r0=8
// result in r0
```



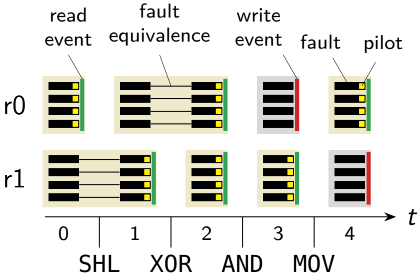
- Plan and inject!
  - Record a fault-free execution of the program-under-test.
  - Inject every memory location in each processor cycle.
  - Wait.... (40 injections)

- Def-Use Pruning
  - **Observation**: Faults between read/write events have equivalent behavior
  - Faults only become active on a read; a write makes it benign.
  - Select one *fault-injection pilot* for each equivalence interval



Fault Space Pruning

- Def-Use Pruning
  - **Observation**: Faults between read/write events have equivalent behavior
  - Faults only become active on a read; a write makes it benign.
  - Select one *fault-injection pilot* for each equivalence interval



- Significantly reduces number of injections (40 $\rightarrow$ 24), but...
  - Equivalences are only formed horizontally, not vertically.
  - Some instructions mask errors or only propagate them.

In a nutshell: Data-flow-Sensitive Pruning

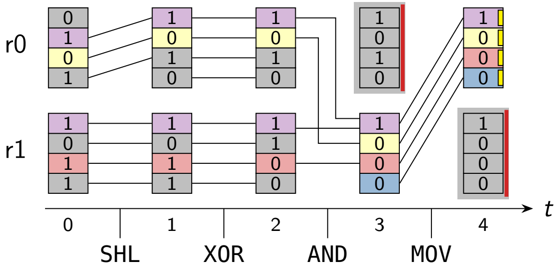
Basic principle

As long as an single-bit error does not escalate to a multi-bit error or becomes visible, we can extend the equivalence set.

In a nutshell: Data-flow-Sensitive Pruning

Basic principle

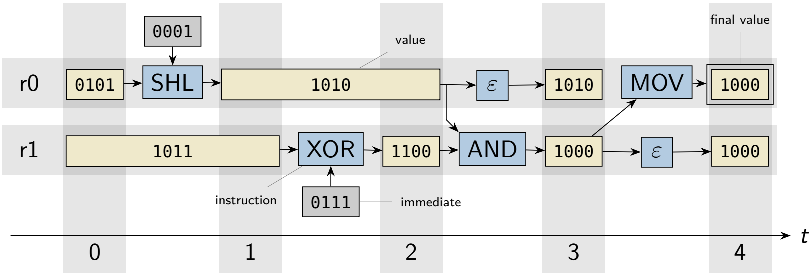
As long as an single-bit error does not escalate to a multi-bit error or becomes visible, we can extend the equivalence set.



- Golden run is one path through the program
  - **Knowledge**: instructions, register values, instruction semantic

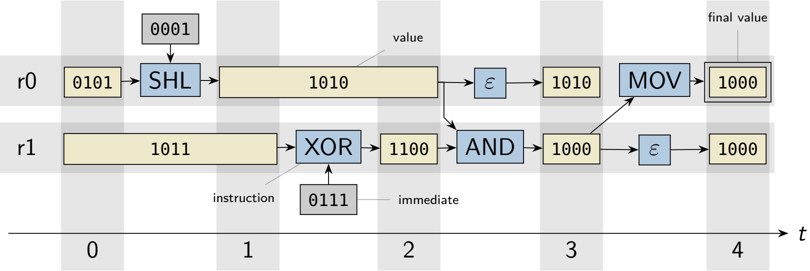
→ We can calculate masking and propagation behavior.

Step 1: Build a data-flow graph



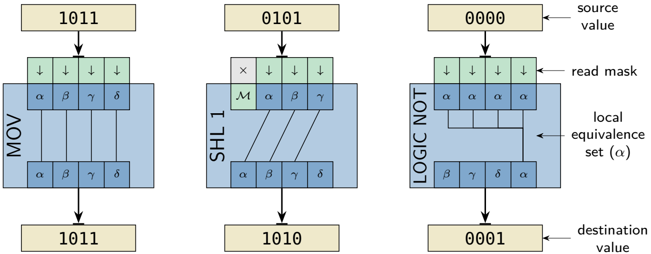
- Directed graph of operations (blue) and operands (yellow)
  - All values and operations are known from the golden run
  - Artificial ε-nodes model the influence of read events

Step 1: Build a data-flow graph



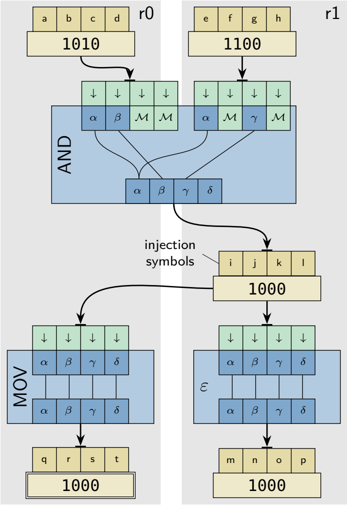
- Directed graph of operations (blue) and operands (yellow)
  - All values and operations are known from the golden run
  - Artificial ε-nodes model the influence of read events
- Choosing read or final value nodes for injection leads to Def-Use pilots

Step 2: Form local fault equivalences



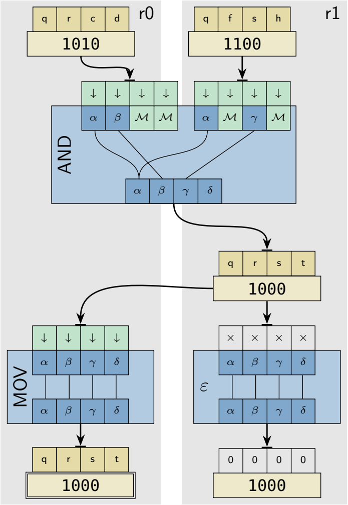
- Error propagation of a single instruction
  - **Assumption**: Exactly one input bit is faulty
  - Combine instruction semantic and operand values

Step 3: Propagate equivalences globally



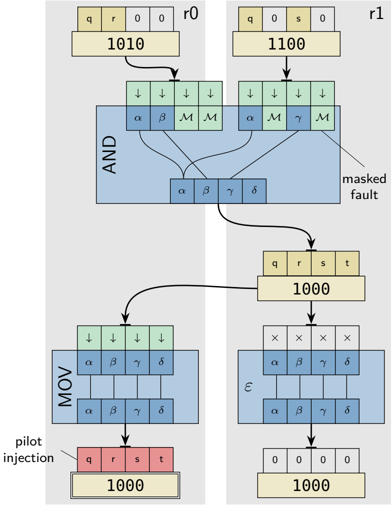
- One FI symbol per operand bit
  - All occurrences are equivalent
  - **Goal**: Propagate symbols

Step 3: Propagate equivalences globally



- One FI symbol per operand bit
  - All occurrences are equivalent
  - **Goal**: Propagate symbols
- Propagation Phase
  - readers $= 0 \longrightarrow$ mark benign
  - readers $= 1 \longrightarrow$ propagate back
  - readers $> 1 \longrightarrow$ do nothing

Step 3: Propagate equivalences globally



- One FI symbol per operand bit
  - All occurrences are equivalent
  - **Goal**: Propagate symbols
- Propagation Phase
  - readers $= 0 \rightarrow$ mark benign
  - readers $= 1 \rightarrow$ propagate back
  - readers $> 1 \rightarrow$ do nothing
- Mask and Plan
  - Operation can mask faults
  - One injection per symbol

Evaluation: MiBench and Microbenchmarks

	#Faults [10^6]	Def-Use #Inj. [10^4]	DFPrune #Inj. [10^4]	Δ Inj. [%]
mi/BC	70.33	222.40	181.43	-18.42
mi/BFD	1894.82	331.38	295.95	-10.69
mi/BFE	1880.82	326.93	292.97	-10.39
mi/QSORT	1623.90	270.58	234.31	-13.40
mi/RDD	3506.17	397.60	345.37	-13.13
mi/RDE	3457.59	397.99	351.90	-11.58
mi/SHA	242.63	252.79	219.74	-13.07
μ/FIB	1.15	8.87	7.56	-14.78
μ/LSUM	0.02	0.26	0.26	0.00
μ/MIXED	0.03	0.45	0.40	-11.83
μ/QSort	0.18	1.27	1.22	-4.36
μ/QSortIter	1.20	4.23	3.88	-8.18

Summary

- Def-Use Pruning is one-dimensional
  - Equivalences are only formed along the time axis
  - Instruction can mask errors benign or propagate them
- DFPrune: Data-Flow–Sensitive Fault Space Pruning
  - Faults are equivalent as long as the error does not escape!
  - Propagate FI Symbols on the Data-Flow Graph
  - Instruction-local Fault Equivalences
- DFPrune reduces the number of required injections
  - Between 10 and 18 percent reduction for MiBench
  - Reductions across all failure classes
  - At least as good as Def-Use Pruning