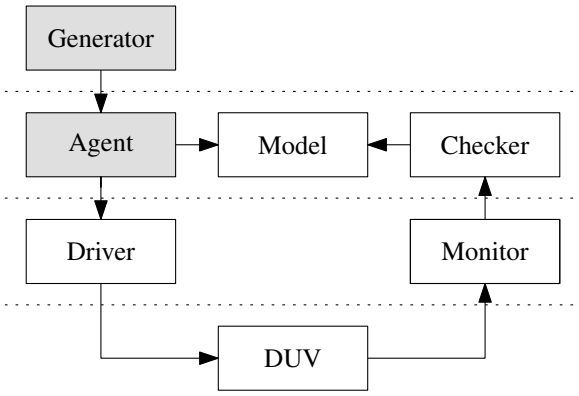
1 Constrained Random Stimulus

Das *Coverage* -Ziel wird Entwicklern meist durch das Management vorgegeben und liegt dadurch in fast allen Fällen sehr nah an 100%. Durch einen gerichteten Test ist dies bei komplexen Entwürfen meist kaum möglich. Eine Abhilfe für dieses Problem besteht in der automatisierten Generierung der Testfälle.

Stellen Sie sicher, dass die (*Functional* !) *Coverage* -Regeln aus der letzten Übung richtig in Ihre PROL16-Verifikationsumgebung aus den letzten Übungen integriert sind. Erweitern Sie außerdem die Klasse `Prol16Opcode` um *Constraints* , sodass nur gültige Opcodes und Registeradressen generiert werden.

Passen Sie Ihre Verifikationsumgebung nun so an, dass anstatt des gerichteten Tests Befehle randomisiert und dann auf dem Modell und dem DUV ausgeführt werden:



Simulieren Sie so eine gewisse Anzahl von Befehlen und dokumentieren Sie die erreichte *Functional Coverage* ! Erhöhen Sie die Anzahl der simulierten Befehle solange, bis Sie die maximal mögliche *Coverage* erreichen – wieviele Befehle werden dafür (ungefähr) benötigt? Erstellen Sie ein Diagramm, in dem Sie die erreichte Coverage über die Anzahl der simulierten Befehle darstellen. Nehmen Sie dazu mindestens zehn Messpunkte auf. Erklären Sie
□ den Kurvenverlauf!

□ Beantworten Sie die folgenden Fragen:

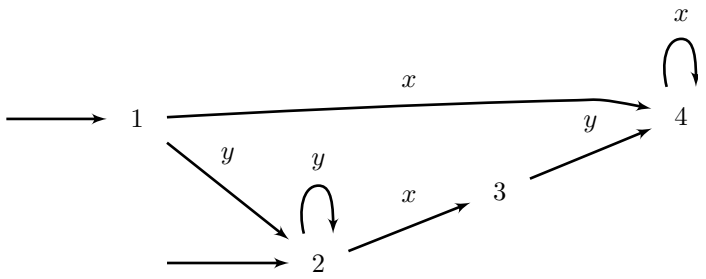
- Sind die randomisierten Werte vorhersehbar? Begründen Sie Ihre Antwort! Welchen Vorteil haben/hätten vorhersehbare Werte?
- Welcher mathematischen Funktion folgt die erreichte *Functional Coverage* in Abhängigkeit der Anzahl der durchgeführten Tests? Begründen Sie diesen Verlauf!
- Wie würden Sie ihre *Constraints* der Klasse `Prol16Opode` wählen, wenn:
  - 50% der generierten Registeradressen 0 sein sollen?
  - die beiden Registeradressen für **NOP** immer 0 sein müssen?
- Was ist der Unterschied zwischen `rand` und `randc`?
- Wieviele Befehle werden mindestens benötigt, um das folgende Covarage-Ziel zu erreichen?
  - Jeder Befehl soll mit jeder Kombination von Carry und Zero ausgeführt werden und jede dieser Ausführungen soll jede mögliche Kombination von Carry und Zero erzeugen. Beachten Sie dabei, dass nicht jeder Befehl jedes Flag beeinflusst – **AND** setzt bspw. das Carry-Flag nie, diese Kombinationen sollen also nicht berücksichtigt werden. Die Befehle **LOAD**, **STORE** und **SLEEP** sollen ebenfalls nicht berücksichtigt werden.

  Sie können dabei vereinfachend annehmen, dass vor jedem Befehl alle Registerinhalte und Flags beliebig verändert werden können.

  Wieviele Befehle benötigt Ihr Test aus dieser Übung? Erklären Sie, warum die Redundanz auftritt!

2 Modellierung

1. Gegeben sei das im Folgenden abgebildete LTS. Geben Sie die formale Darstellung des LTS als 4-Tupel, inklusive seiner Komponenten, an!



2. Konstruieren Sie eine Kripke-Struktur zu dem oben gegebenen LTS!

> *"The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at and repair."*
>
> Douglas Adams

1 Beispiel 1

1.1 Diagramm

Anzahl	Coverage (%)
1	15
10	52.3
50	72.8
100	78.4
500	87.7
1000	90.1
5000	95
10000	95.2
100000	95.2
1000000	95.2

Table 1: Coverage Tabelle

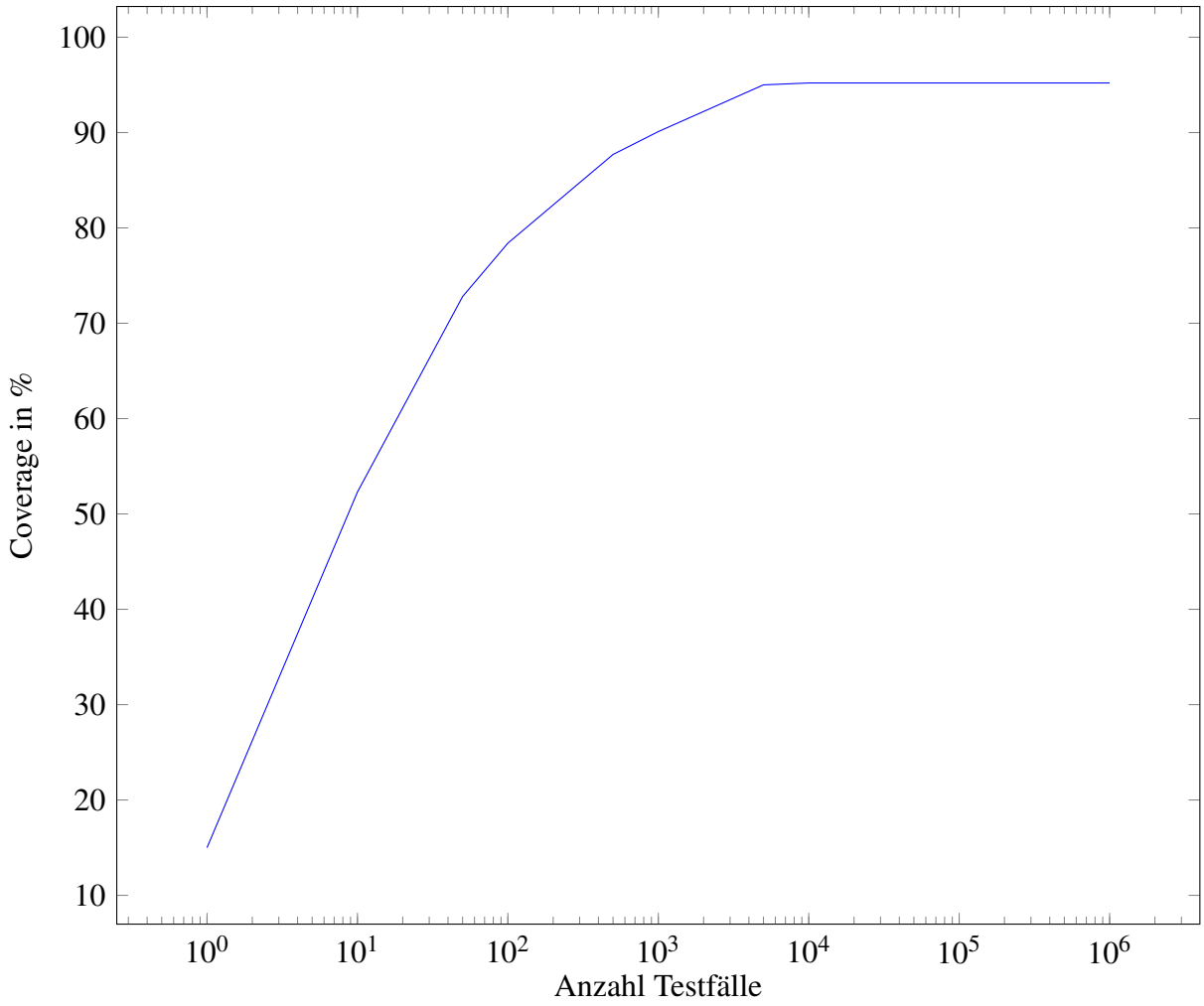


Figure 1: Coverage Graph

1.2 Beantwortung der Fragen

- Randomisierte Werte sind vorhersehbar, weil der Randomgenerator einen Seed hat. Die Testfälle sind dadurch reproduzierbar.

- Logarithmische Funktion. Je mehr Kombinationen ausgeführt werden desto geringer ist die Chance, dass eine Kombinationen ausgeführt wird, die noch nicht ausgeführt wurde bzw. sind ab einem bestimmten Zeitpunkt alle möglichen Kombinationen ausgeführt worden.

- ```
  constraint ra_0 {
     ra dist { [0:0]:/50, [1:31]:/50 };
  }

  constraint rb_0 {
     rb dist { [0:0]:/50, [1:31]:/50 };
  }

  constraint nop_0 {
     (cmd == Nop) -> (ra == 0) && (rb == 0);
  }
  ```

- `randc` wiederholt Werte erst dann wenn alle ausgeschöpft wurden.

- Befehle: 21
  Befehle, die keine Flags beeinflussen: 6
  Befehle, die beide Flags beeinflussen: 11
  Befehle, Carry Flag 0 und Z Flag beeinflussen: 4

  Berechnung:
  keine Flags: 6 * 4 = 24
  Flags: 11 * 4 * 4 = 176
  Carry Flag 0: 4 * 4 * 2 = 32
  Summe: 232

  Test aus der Übung: Der Test braucht ca. 5000 Kombinationen, dann ist die maximale Coverage erreicht. Die Redundanz tritt auf, da mehrere Register verwendet werden und dadurch sehr spezielle Kombinationen von Befehlen ausgeführt werden müssen um alle Tests zu covern. Zum Beispiel müssen für ein ADDC sehr spezielle Werte vorhanden sein, damit dieser das Zero und Carry Flag setzt.
  ```

1.3 Source Code

Der Sourcecode des Prol16 wurde nicht hinzugefügt, da der von der Elearning Plattform verwendet wurde.

../../src/sv/ifProl16.sv

```
1  interface ifProl16 #(parameter int gDataWidth = 16);
2
3     logic [gDataWidth-1:0] mem_addr;     // unused
4     logic [gDataWidth-1:0] mem_data_cpu;    // data from cpu
5     logic [gDataWidth-1:0] mem_data_tb;     // data from tb
6     logic mem_ce_n;
7     logic mem_oe_n;
8     logic mem_we_n;                      // unused
9     logic illegal_inst;
10    logic cpu_halt;
11
12    modport master (
13       output mem_data_tb,
14       input mem_data_cpu, mem_ce_n, mem_oe_n, illegal_inst, cpu_halt
15    );
16
17 endinterface
```

../../src/sv/pkgProl16.sv

```
1  package pkgProl16;
2     typedef bit [15:0] data_v;
3  endpackage
```

../../src/sv/Prol16Command.sv

```
1  typedef enum bit [5:0]
2     {
3        Nop=6'b000000,
4        Loadi=6'b000010,
5        Jump=6'b001000,
6        Jumpc=6'b001010,
7        Jumpz=6'b001011,
8        Move=6'b001100,
9        And=6'b010000,
10       Or=6'b010001,
11       Xor=6'b010010,
12       Not=6'b010011,
13       Add=6'b010100,
14       Addc=6'b010101,
15       Sub=6'b010110,
16       Subc=6'b010111,
17       Comp=6'b011000,
18       Inc=6'b011010,
19       Dec=6'b011011,
20       Shl=6'b011100,
21       Shr=6'b011101,
22       Shlc=6'b011110,
23       Shrc=6'b011111
24       //Invalid=6'b111111
25    } Prol16Command;
```

../../src/sv/Prol16Opcode.sv

```
1  `include "Prol16Command.sv"
2
```

```
3  class Prol16Opcode;
4     rand int ra;
5     rand int rb;
6     randc Prol16Command cmd;
7     rand pkgProl16::data_v data;
8
9     constraint ra_size {ra >= 0 && ra <= 31;};
10    constraint rb_size {rb >= 0 && rb <= 31;};
11
12    constraint ra_0 {
13       ra dist { [0:0]:/50, [1:31]:/50 };
14    }
15
16    constraint rb_0 {
17       rb dist { [0:0]:/50, [1:31]:/50 };
18    }
19
20    constraint nop_0 {
21       (cmd == Nop) -> (ra == 0) && (rb == 0);
22    }
23
24    function new(int ra, int rb, Prol16Command cmd, pkgProl16::data_v data);
25       this.ra = ra;
26       this.rb = rb;
27       this.cmd = cmd;
28       this.data = data;
29    endfunction
30 endclass
```

../../src/sv/Prol16State.sv

```
1  class Prol16State #(parameter int gRegs = 32);
2     pkgProl16::data_v regs[gRegs];
3     pkgProl16::data_v programCounter;
4     bit cFlag;
5     bit zFlag;
6
7     function new();
8        for (int i = 0; i < gRegs; i++) begin
9           regs[i] = '0;
10       end
11       programCounter = 0;
12       cFlag = 0;
13       zFlag = 0;
14    endfunction
15
16    task reset();
17       for (int i = 0; i < gRegs; i++) begin
18          regs[i] = '0;
19       end
20       programCounter = 0;
21       cFlag = 0;
22       zFlag = 0;
23    endtask
24 endclass
```

../../src/sv/Prol16Model.sv

```
1  `include "Prol16State.sv"
2  `include "Prol16Opcode.sv"
3
4  class Prol16Model#(parameter int gNumRegs);
```

```
5     Prol16State state;
6
7     function new(Prol16State _state);
8        state = _state;
9     endfunction
10
11    task reset();
12       state.reset();
13    endtask
14
15    task calcCarryFlag(int data);
16       if (data > (2**$size(pkgProl16::data_v) - 1) || data < 0)
17          state.cFlag = 1;
18       else
19          state.cFlag = 0;
20    endtask
21
22    task calcZeroFlag(pkgProl16::data_v data);
23       if (data == '0)
24          state.zFlag = 1;
25       else
26          state.zFlag = 0;
27    endtask
28
29    task execute(Prol16Opcode opcode);
30       int data = 0;  // tmp data for carry calc
31       bit tmpCFlag = 0; // tmp carry flag
32
33       // inc pc here, overwrite in jumps
34       state.programCounter++;
35
36      if (opcode.ra > (gNumRegs - 1) || opcode.ra < 0 ||
37          opcode.rb > (gNumRegs - 1) || opcode.rb < 0)
38          $display("Illegal register, doing nop");
39      else
40       begin
41
42       case (opcode.cmd)
43          Nop:
44          begin
45          end
46
47          Loadi:
48          begin
49             state.regs[opcode.ra] = opcode.data;
50             state.programCounter++; // TODO: necessary?
51          end
52
53          Jump:
54          begin
55             state.programCounter = state.regs[opcode.ra];
56          end
57
58          Jumpc:
59          begin
60             if (state.cFlag == 1)
61                state.programCounter = state.regs[opcode.ra];
62          end
63
64          Jumpz:
```

```
65          begin
66             if (state.zFlag == 1)
67                state.programCounter = state.regs[opcode.ra];
68          end
69
70          Move:
71          begin
72             state.regs[opcode.ra] = state.regs[opcode.rb];
73          end
74
75          And:
76          begin
77             state.regs[opcode.ra] = state.regs[opcode.ra] & state.regs[
                  opcode.rb];
78             state.cFlag = 0;
79             calcZeroFlag(state.regs[opcode.ra]);
80          end
81
82          Or:
83          begin
84             state.regs[opcode.ra] = state.regs[opcode.ra] | state.regs[
                  opcode.rb];
85             state.cFlag = 0;
86             calcZeroFlag(state.regs[opcode.ra]);
87          end
88
89          Xor:
90          begin
91             state.regs[opcode.ra] = state.regs[opcode.ra] ^ state.regs[
                  opcode.rb];
92             state.cFlag = 0;
93             calcZeroFlag(state.regs[opcode.ra]);
94          end
95
96          Not:
97          begin
98             state.regs[opcode.ra] = ~(state.regs[opcode.ra]);
99             state.cFlag = 0;
100            calcZeroFlag(state.regs[opcode.ra]);
101         end
102
103         Add:
104         begin
105            data = state.regs[opcode.ra] + state.regs[opcode.rb];
106            calcCarryFlag(data);
107            state.regs[opcode.ra] = data;
108            calcZeroFlag(state.regs[opcode.ra]);
109         end
110
111         Addc:
112         begin
113            data = state.regs[opcode.ra] + state.regs[opcode.rb] + state.
                  cFlag;
114            calcCarryFlag(data);
115            state.regs[opcode.ra] = data;
116            calcZeroFlag(state.regs[opcode.ra]);
117         end
118
119         Sub:
120         begin
```

```
121             data = state.regs[opcode.ra] - state.regs[opcode.rb];
122             calcCarryFlag(data);
123             state.regs[opcode.ra] = data;
124             calcZeroFlag(state.regs[opcode.ra]);
125          end
126
127          Subc:
128          begin
129             data = state.regs[opcode.ra] - state.regs[opcode.rb] - state.
                   cFlag;
130             calcCarryFlag(data);
131             state.regs[opcode.ra] = data;
132             calcZeroFlag(state.regs[opcode.ra]);
133          end
134
135          Comp:
136          begin
137             data = state.regs[opcode.ra] - state.regs[opcode.rb];
138             calcCarryFlag(data);
139             calcZeroFlag(data);
140          end
141
142          Inc:
143          begin
144             data = state.regs[opcode.ra] + 1;
145             calcCarryFlag(data);
146             state.regs[opcode.ra] = data;
147             calcZeroFlag(state.regs[opcode.ra]);
148          end
149
150          Dec:
151          begin
152             data = state.regs[opcode.ra] - 1;
153             calcCarryFlag(data);
154             state.regs[opcode.ra] = data;
155             calcZeroFlag(state.regs[opcode.ra]);
156          end
157
158          Shl:
159          begin
160             if (state.regs[opcode.ra][$size(pkgProl16::data_v) - 1] == 1)
161               state.cFlag = 1;
162             else
163               state.cFlag = 0;
164
165             state.regs[opcode.ra] = state.regs[opcode.ra] << 1;
166             calcZeroFlag(state.regs[opcode.ra]);
167          end
168
169          Shr:
170          begin
171                if (state.regs[opcode.ra][0] == 1)
172               state.cFlag = 1;
173             else
174               state.cFlag = 0;
175
176                state.regs[opcode.ra] = state.regs[opcode.ra] >> 1;
177             calcZeroFlag(state.regs[opcode.ra]);
178          end
179
```

```
180           Shlc:
181           begin
182             if (state.regs[opcode.ra][$size(pkgProl16::data_v) - 1] == 1)
183               tmpCFlag = 1;
184              else
185               tmpCFlag = 0;
186
187              state.regs[opcode.ra] = state.regs[opcode.ra] << 1;
188              state.regs[opcode.ra][0] = state.cFlag;
189              state.cFlag = tmpCFlag;
190              calcZeroFlag(state.regs[opcode.ra]);
191           end
192
193           Shrc:
194           begin
195             if (state.regs[opcode.ra][0] == 1)
196               tmpCFlag = 1;
197              else
198               tmpCFlag = 0;
199
200              state.regs[opcode.ra] = state.regs[opcode.ra] >> 1;
201              state.regs[opcode.ra][$size(pkgProl16::data_v) - 1] = state.
                    cFlag;
202              state.cFlag = tmpCFlag;
203              calcZeroFlag(state.regs[opcode.ra]);
204           end
205
206           default : $display("Wrong opcode: doing nop");
207         endcase
208         end
209      endtask
210   endclass
```

../../src/sv/testProl16Rand.sv

```
1  `include "Prol16Model.sv"
2
3  class TestClass;
4     task assertWithoutFlags(int expectedRegisterValue, int expectedPCValue,
          Prol16State state, int register, string text);
5        assert (expectedRegisterValue == state.regs[register])
6        else $error("Expected Register Value: %d, Actual Register Value: %d,
             Info: %s", expectedRegisterValue, state.regs[register], text);
7
8        assert (expectedPCValue == state.programCounter)
9        else $error("Expected PC Value: %d, Actual PC Value: %d, Info: %s",
             expectedPCValue, state.programCounter, text);
10    endtask
11
12    task assertWithFlags(int expectedRegisterValue, int expectedPCValue, int
           expectedCarryFlag, int expectedZeroFlag, Prol16State state, int
          register, string text);
13       assert (expectedRegisterValue == state.regs[register])
14       else $error("Expected Register Value: %d, Actual Register Value: %d,
             Info: %s", expectedRegisterValue, state.regs[register], text);
15
16       assert (expectedPCValue == state.programCounter)
17       else $error("Expected PC Value: %d, Actual PC Value: %d, Info: %s",
             expectedPCValue, state.programCounter, text);
18
19       assert (expectedCarryFlag == state.cFlag)
```

```
20        else $error("Expected Carry Flag Value: %d, Actual Carry Flag Value:
              %d, Info: %s", expectedCarryFlag, state.cFlag, text);
21
22        assert (expectedZeroFlag == state.zFlag)
23        else $error("Expected Zero Flag Value: %d, Actual Zero Flag Value: %d
              , Info: %s", expectedZeroFlag, state.zFlag, text);
24     endtask
25
26     task assertWithDuv(Prol16State state, int register, logic [15:0] cpuRegs
           [31:0], logic[15:0] cpuPc, logic cpuCFlag, logic cpuZFlag,
          Prol16Command cmd);
27        assert (state.regs[register] == cpuRegs[register])
28        else $error("WithDuv: Expected Register Value: %d, Actual Register
              Value: %d, Info: %s", state.regs[register], cpuRegs[register], cmd
              .name());
29
30        assert (state.programCounter == cpuPc)
31        else $error("WithDuv: Expected PC Value: %d, Actual PC Value: %d,
              Info: %s", state.programCounter, cpuPc, cmd.name());
32
33        assert (state.cFlag == cpuCFlag)
34        else $error("WithDuv: Expected Carry Flag Value: %d, Actual Carry
              Flag Value: %d, Info: %s", state.cFlag, cpuCFlag, cmd.name());
35
36        assert (state.zFlag == cpuZFlag)
37        else $error("WithDuv: Expected Zero Flag Value: %d, Actual Zero Flag
              Value: %d, Info: %s", state.zFlag, cpuZFlag, cmd.name());
38     endtask
39  endclass
40
41  program testProl16Rand(ifProl16.master cpu, output logic rst, input logic
        clk);
42     logic [15:0] cpuRegs [31:0];
43     logic [15:0] cpuPc;
44     logic cpuCFlag;
45     logic cpuZFlag;
46     logic [5:0] opcodeDUV;
47     logic [4:0] ra;
48     logic [4:0] rb;
49
50     logic [5:0] prevOpcodeDUV;
51
52     const int numTestCases = 100000;
53
54     event CommandStart;
55     event End;
56
57     bit LoadiOccurred = 0;
58
59     Prol16Opcode opcode = new(0, 0, Nop, 0);
60
61     task trigger();
62        while (!End.triggered) begin
63          @(negedge cpu.mem_oe_n)
64          begin
65            //$display("negEdge oe");
66           if (LoadiOccurred == 0)
67           begin
68              cpu.mem_data_tb[15:10] <= opcode.cmd;
69              cpu.mem_data_tb[9:5] <= opcode.ra;
```

```
70              cpu.mem_data_tb[4:0] <= opcode.rb;
71
72           prevOpcodeDUV <= opcodeDUV;
73
74              if (opcode.cmd == Loadi)
75              begin
76                 LoadiOccurred = 1;
77              end
78           end
79           else
80           begin
81              cpu.mem_data_tb <= opcode.data;
82              LoadiOccurred = 0;
83           end
84          end
85          @(posedge cpu.mem_oe_n)
86          begin
87            //$display("posEdge oe");
88           -> CommandStart;
89          end
90        end
91     endtask
92
93     //always_ff @(negedge clk) begin
94    //  previous_opcode <= opcode;
95    //end
96
97     covergroup CoverCpu @(negedge cpu.mem_oe_n);  // TODO, the right signal?
98        option.per_instance = 1;
99
100       cmd: coverpoint opcodeDUV iff (rst) {
101       bins NOP   = {0};
102       ignore_bins SLEEP = {1};
103       bins LOADI = {2};
104       ignore_bins LOAD  = {3};
105       ignore_bins STORE = {4};
106       bins JUMP  = {8};
107       bins JUMPC = {10};
108       bins JUMPZ = {11};
109       bins MOVE  = {12};
110       bins AND   = {16};
111       bins OR    = {17};
112       bins XOR   = {18};
113       bins NOT   = {19};
114       bins ADD   = {20};
115       bins ADDC  = {21};
116       bins SUB   = {22};
117       bins SUBC  = {23};
118       bins COMP  = {24};
119       bins INC   = {26};
120       bins DEC   = {27};
121       bins SHL   = {28};
122       bins SHR   = {29};
123       bins SHLC  = {30};
124       bins SHRC  = {31};
125       bins Invalid = {63};
126       illegal_bins other = default;
127     }
128
129       prevCmd: coverpoint prevOpcodeDUV iff (rst) {
```

```
130         bins NOP   = {0};
131         ignore_bins SLEEP = {1};
132         bins LOADI = {2};
133         ignore_bins LOAD  = {3};
134         ignore_bins STORE = {4};
135         bins JUMP  = {8};
136         bins JUMPC = {10};
137         bins JUMPZ = {11};
138         bins MOVE  = {12};
139         bins AND   = {16};
140         bins OR    = {17};
141         bins XOR   = {18};
142         bins NOT   = {19};
143         bins ADD   = {20};
144         bins ADDC  = {21};
145         bins SUB   = {22};
146         bins SUBC  = {23};
147         bins COMP  = {24};
148         bins INC   = {26};
149         bins DEC   = {27};
150         bins SHL   = {28};
151         bins SHR   = {29};
152         bins SHLC  = {30};
153         bins SHRC  = {31};
154         bins Invalid = {63};
155         //illegal_bins other = default;
156       }
157
158         ra: coverpoint ra;
159         rb: coverpoint rb;
160
161         crs_cmd_ra: cross cmd, ra;
162         crs_cmd_rb: cross cmd, rb;
163
164         cFlag: coverpoint cpuCFlag {
165            bins set = {1};
166            bins notset = {0};
167         }
168
169         zFlag: coverpoint cpuZFlag {
170            bins set = {1};
171            bins notset = {0};
172         }
173
174         crs_cmd_c: cross cmd, cFlag;
175         crs_cmd_z: cross cmd, zFlag;
176
177         trans_c: coverpoint cpuCFlag {
178         bins from_0_to_1 = (0 => 1);
179         bins from_1_to_0 = (1 => 0);
180         bins from_0_to_0 = (0 => 0);
181         bins from_1_to_1 = (1 => 1);
182       }
183
184         trans_z: coverpoint cpuZFlag {
185         bins from_0_to_1 = (0 => 1);
186         bins from_1_to_0 = (1 => 0);
187         bins from_0_to_0 = (0 => 0);
188         bins from_1_to_1 = (1 => 1);
189       }
```

```
190
191       prevCmdChangeFlags: coverpoint prevOpcodeDUV {
192       bins doesnt_change_flags = { 0, 1, 2, 3, 4, 8, 10, 11, 12};
193       bins c_flag_0_and_z_flag_x = { 16, 17, 18, 19 };
194       bins c_flag_x_and_z_flag_x = { 20, 21, 22, 23, 24, 26, 27, 28, 29,
              30, 31 };
195     }
196
197       crs_cmd_c_change: cross prevCmdChangeFlags, trans_c {
198       illegal_bins ill = binsof(prevCmdChangeFlags.doesnt_change_flags) &&
              (binsof(trans_c.from_0_to_1) || binsof(trans_c.from_1_to_0));
199
200     }
201
202       crs_cmd_z_change: cross prevCmdChangeFlags, trans_z {
203       illegal_bins ill = binsof(prevCmdChangeFlags.doesnt_change_flags) &&
              (binsof(trans_z.from_0_to_1) || binsof(trans_z.from_1_to_0));
204     }
205
206      crs_cmd_c_is_0: cross prevCmdChangeFlags, trans_c {
207       illegal_bins ill = binsof(prevCmdChangeFlags.c_flag_0_and_z_flag_x)
              && (binsof(trans_c.from_0_to_1) || binsof(trans_c.from_1_to_1));
208     }
209
210   endgroup
211
212   initial begin : stimuli
213      Prol16State state = new();
214      Prol16Model#(32) model = new(state);
215
216      TestClass testClass = new();
217
218      CoverCpu coverCpu = new;
219
220      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(0)"
             , "/top/TheTest/cpuRegs(0)");
221      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(1)"
             , "/top/TheTest/cpuRegs(1)");
222      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(2)"
             , "/top/TheTest/cpuRegs(2)");
223      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(3)"
             , "/top/TheTest/cpuRegs(3)");
224      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(4)"
             , "/top/TheTest/cpuRegs(4)");
225      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(5)"
             , "/top/TheTest/cpuRegs(5)");
226      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(6)"
             , "/top/TheTest/cpuRegs(6)");
227      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(7)"
             , "/top/TheTest/cpuRegs(7)");
228      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(8)"
             , "/top/TheTest/cpuRegs(8)");
229      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(9)"
             , "/top/TheTest/cpuRegs(9)");
230      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(10)
             ", "/top/TheTest/cpuRegs(10)");
231      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(11)
             ", "/top/TheTest/cpuRegs(11)");
232      $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(12)
             ", "/top/TheTest/cpuRegs(12)");
```

```
233         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(13)
                ", "/top/TheTest/cpuRegs(13)");
234         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(14)
                ", "/top/TheTest/cpuRegs(14)");
235         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(15)
                ", "/top/TheTest/cpuRegs(15)");
236         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(16)
                ", "/top/TheTest/cpuRegs(16)");
237         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(17)
                ", "/top/TheTest/cpuRegs(17)");
238         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(18)
                ", "/top/TheTest/cpuRegs(18)");
239         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(19)
                ", "/top/TheTest/cpuRegs(19)");
240         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(20)
                ", "/top/TheTest/cpuRegs(20)");
241         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(21)
                ", "/top/TheTest/cpuRegs(21)");
242         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(22)
                ", "/top/TheTest/cpuRegs(22)");
243         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(23)
                ", "/top/TheTest/cpuRegs(23)");
244         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(24)
                ", "/top/TheTest/cpuRegs(24)");
245         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(25)
                ", "/top/TheTest/cpuRegs(25)");
246         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(26)
                ", "/top/TheTest/cpuRegs(26)");
247         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(27)
                ", "/top/TheTest/cpuRegs(27)");
248         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(28)
                ", "/top/TheTest/cpuRegs(28)");
249         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(29)
                ", "/top/TheTest/cpuRegs(29)");
250         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(30)
                ", "/top/TheTest/cpuRegs(30)");
251         $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(31)
                ", "/top/TheTest/cpuRegs(31)");
252
253         $init_signal_spy("/top/TheCpu/datapath_inst/RegPC", "/top/TheTest/
                cpuPc");
254         $init_signal_spy("/top/TheCpu/control_inst/Carry", "/top/TheTest/
                cpuCFlag");
255         $init_signal_spy("/top/TheCpu/control_inst/Zero", "/top/TheTest/
                cpuZFlag");
256         $init_signal_spy("/top/TheCpu/datapath_inst/RegOpcode", "/top/TheTest
                /opcodeDUV");
257         $init_signal_spy("/top/TheCpu/datapath_inst/RegAIdx", "/top/TheTest/
                ra");
258         $init_signal_spy("/top/TheCpu/datapath_inst/RegBIdx", "/top/TheTest/
                rb");
259
260         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(0)", "
                16#0000", 0, 1);
261         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(1)", "
                16#0000", 0, 1);
262         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(2)", "
                16#0000", 0, 1);
263         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(3)", "
                16#0000", 0, 1);
```

```
264         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(4)", "
                16#0000", 0, 1);
265         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(5)", "
                16#0000", 0, 1);
266         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(6)", "
                16#0000", 0, 1);
267         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(7)", "
                16#0000", 0, 1);
268         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(8)", "
                16#0000", 0, 1);
269         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(9)", "
                16#0000", 0, 1);
270         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(10)",
                "16#0000", 0, 1);
271         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(11)",
                "16#0000", 0, 1);
272         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(12)",
                "16#0000", 0, 1);
273         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(13)",
                "16#0000", 0, 1);
274         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(14)",
                "16#0000", 0, 1);
275         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(15)",
                "16#0000", 0, 1);
276         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(16)",
                "16#0000", 0, 1);
277         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(17)",
                "16#0000", 0, 1);
278         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(18)",
                "16#0000", 0, 1);
279         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(19)",
                "16#0000", 0, 1);
280         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(20)",
                "16#0000", 0, 1);
281         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(21)",
                "16#0000", 0, 1);
282         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(22)",
                "16#0000", 0, 1);
283         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(23)",
                "16#0000", 0, 1);
284         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(24)",
                "16#0000", 0, 1);
285         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(25)",
                "16#0000", 0, 1);
286         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(26)",
                "16#0000", 0, 1);
287         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(27)",
                "16#0000", 0, 1);
288         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(28)",
                "16#0000", 0, 1);
289         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(29)",
                "16#0000", 0, 1);
290         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(30)",
                "16#0000", 0, 1);
291         $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(31)",
                "16#0000", 0, 1);
292
293
294         // generate reset
                ------------------------------------------------------
```

```
295         rst = 1;
296         #10 rst = 0;
297         #20 rst = 1;
298 
299         //Reset
300         model.reset();
301         //testClass.assertWithoutFlags(0, 0, model.state, 12, "Reset test");
302 
303         fork
304           trigger();
305         join_none
306 
307       //Nop
308         @(CommandStart);
309         testClass.assertWithDuv(model.state, 12, cpuRegs, cpuPc, cpuCFlag,
              cpuZFlag, opcode.cmd);
310 
311 
312         for (int i = 0; i < numTestCases; i++) begin
313           if (opcode.cmd == Loadi)
314           begin
315              @(CommandStart);
316           end
317           model.execute(opcode);
318           opcode.randomize();
319           //$display("Command %b", opcode.cmd);
320           //$display("CommandDUV %b", opcodeDUV);
321            @(CommandStart);
322            testClass.assertWithDuv(model.state, opcode.ra, cpuRegs, cpuPc,
                 cpuCFlag, cpuZFlag, opcode.cmd);
323         end
324 
325         -> End;
326         $stop;
327     end : stimuli
328 endprogram
```

../../src/sv/top.sv

```
1  `include "testProl16Rand.sv"
2 
3  module top;
4     logic clk = 0, rst;
5     always #10 clk = ~clk;
6 
7     ifProl16 ifCpu();
8     cpu TheCpu(clk, rst, ifCpu.mem_addr, ifCpu.mem_data_cpu, ifCpu.
          mem_data_tb, ifCpu.mem_ce_n, ifCpu.mem_oe_n,
9                ifCpu.mem_we_n, ifCpu.illegal_inst, ifCpu.cpu_halt);
10    testProl16Rand TheTest(ifCpu.master, rst, clk);
11 endmodule
```

../../sim/CompileSimRandCov.do

```
1 vlib work
2 
3 vlog ../src/sv/ifProl16.sv
4 vlog ../src/sv/pkgProl16.sv
5 vlog ../src/sv/Prol16Command.sv
6 vlog ../src/sv/Prol16Opcode.sv +incdir+../src/sv
7 vlog ../src/sv/Prol16State.sv +incdir+../src/sv
8 vlog ../src/sv/Prol16Model.sv +incdir+../src/sv
```

```
vlog ../src/sv/testProl16Rand.sv +incdir+../src/sv
vlog ../src/sv/top.sv +incdir+../src/sv

vcom -87 ../src/vhdl/prol16_pack.vhd
vcom -87 ../src/vhdl/reg_file.vhd
vcom -87 ../src/vhdl/datapath.vhd
vcom -87 ../src/vhdl/control.vhd
vcom -87 ../src/vhdl/alu.vhd
vcom -93 ../src/vhdl/memory.vhd
vcom -87 ../src/vhdl/cpu.vhd
#vcom -87 ../src/vhdl/cpu_tb.vhd

vsim -novopt top
run -all
```

2 Beispiel 2

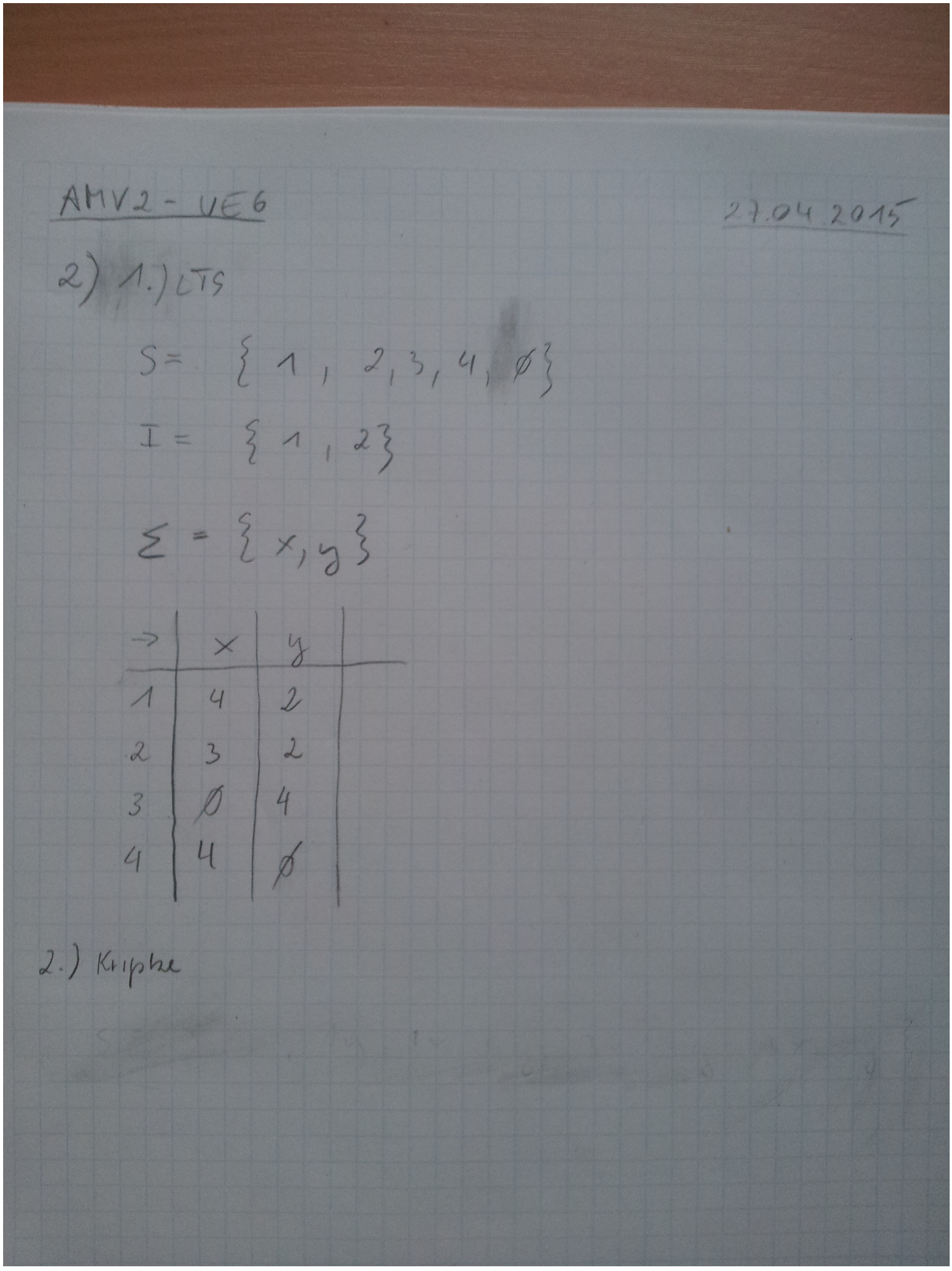


Figure 2: LTS

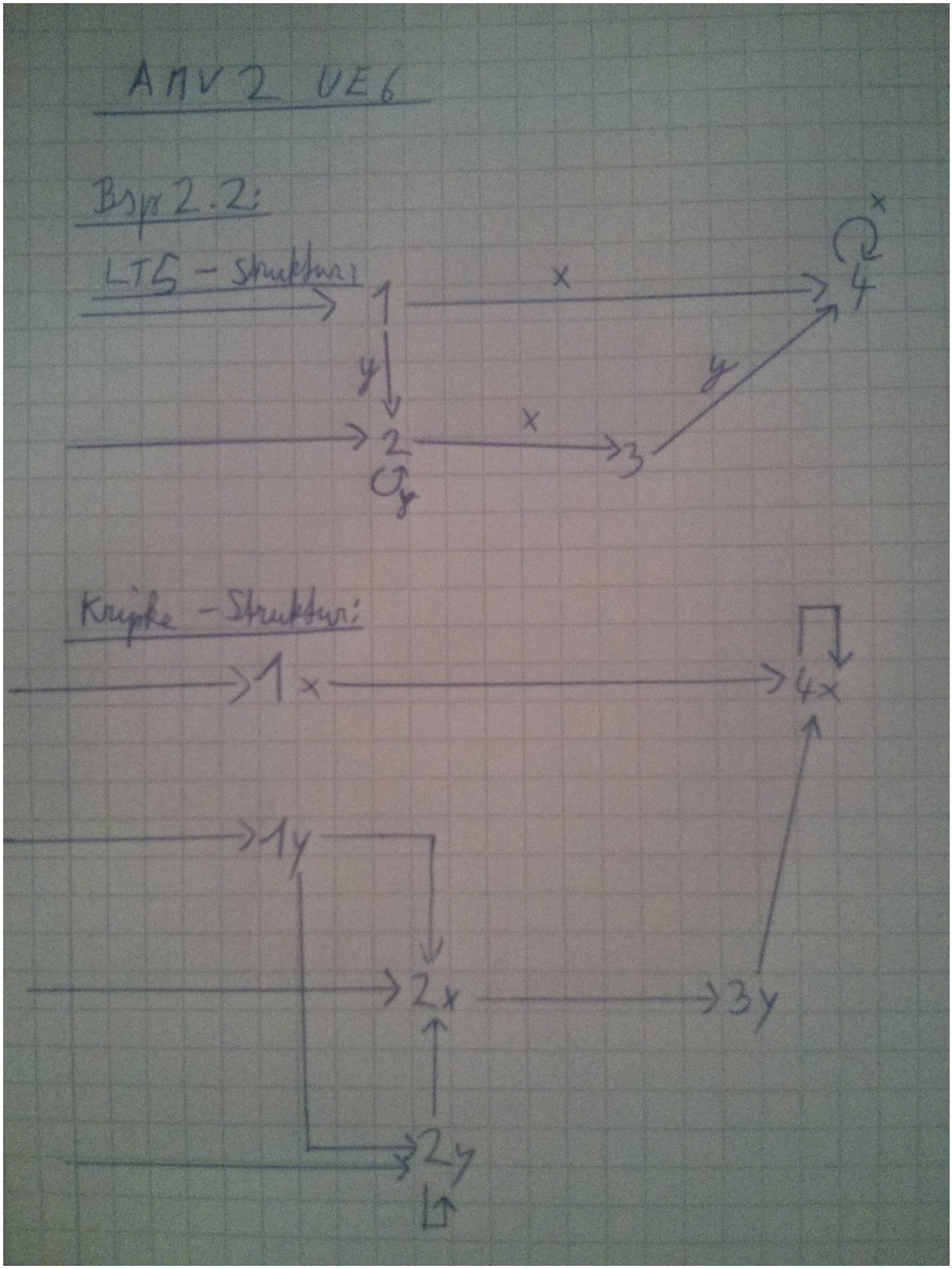


Figure 3: Kripke