Delayed Nondeterminism in Model Checking Embedded Systems Assembly Code

Thomas Noll[1] Bastian Schlich[2]

[1]Software Modelling and Verification Group
[2]Software for Embedded Systems
RWTH Aachen University
noll@cs.rwth-aachen.de

Haifa Verification Conference, October 25, 2007

C Code vs. Assembly Code

- Most microcontroller software written in C
- Many model checkers accept (ANSI) C programs: BLAST, CBMC, MAGIC, ...
- Often restricted support of C constructs:
  - no expressions with side effects
  - no recursion
  - only integer variables
  - ...
- Many microcontroller features not considered in ANSI C:
  - direct hardware access (registers, ...)
  - embedded assembly code
  - interrupts
  - ...
- Case studies from industrial applications contain errors which
  - passed all tests and reviews
  - are only observable on assembly level
  - caused by forgotten interrupt enabling/disabling, reentrance problems, ...

$\Longrightarrow$ Base model checking on assembly code

C Code vs. Assembly Code

- Most microcontroller software written in C
- Many model checkers accept (ANSI) C programs: BLAST, CBMC, MAGIC, ...
- Often restricted support of C constructs:
  - no expressions with side effects
  - no recursion
  - only integer variables
  - ...
- Many microcontroller features not considered in ANSI C:
  - direct hardware access (registers, ...)
  - embedded assembly code
  - interrupts
  - ...
- Case studies from industrial applications contain errors which
  - passed all tests and reviews
  - are only observable on assembly level
  - caused by forgotten interrupt enabling/disabling, reentrance problems, ...

$\Longrightarrow$ Base model checking on assembly code

C Code vs. Assembly Code

- Most microcontroller software written in C
- Many model checkers accept (ANSI) C programs: BLAST, CBMC, MAGIC, ...
- Often restricted support of C constructs:
  - no expressions with side effects
  - no recursion
  - only integer variables
  - ...
- Many microcontroller features not considered in ANSI C:
  - direct hardware access (registers, ...)
  - embedded assembly code
  - interrupts
  - ...
- Case studies from industrial applications contain errors which
  - passed all tests and reviews
  - are only observable on assembly level
  - caused by forgotten interrupt enabling/disabling, reentrance problems, ...

$\Longrightarrow$ Base model checking on assembly code

C Code vs. Assembly Code

- Most microcontroller software written in C
- Many model checkers accept (ANSI) C programs: BLAST, CBMC, MAGIC, ...
- Often restricted support of C constructs:
  - no expressions with side effects
  - no recursion
  - only integer variables
  - ...
- Many microcontroller features not considered in ANSI C:
  - direct hardware access (registers, ...)
  - embedded assembly code
  - interrupts
  - ...
- Case studies from industrial applications contain errors which
  - passed all tests and reviews
  - are only observable on assembly level
  - caused by forgotten interrupt enabling/disabling, reentrance problems, ...

$\Longrightarrow$ Base model checking on assembly code

C Code Before Preprocessing

```
int main (void) {
  init();// call initialization
  sei();
  while(1) {
    inputs = PINA & 0x0F;
    cli();
    if (direction < 5) {
      if (inputs & (1 << 1)) {// down
        if (direction = 2 || direction = 3) {
          TCCR1B = 0x00;
          TIFR = 0xFF;
          TCNT1 = 0x00;
          TIMSK = (1<<OCIE1A);
          TCCR1B = 0x05;
          direction = 1;
        }
```

C Code After Preprocessing

```
int main (void) {
  init();
  __asm__ __volatile__ ("sei" ::);
  while(1) {
    inputs = (*(volatile uint8_t *)((0x19) + 0x20)) & 0x0F;
    asm__ __volatile__ ("cli" ::);
    if (direction < 5) {
      if (inputs & (1 << 1)) {
        if (direction = 2 || direction = 3) {
          (*(volatile uint8_t *)((0x2E) + 0x20)) = 0x00;
          (*(volatile uint8_t *)((0x38) + 0x20)) = 0xFF;
          (*(volatile uint16_t *)((0x2C) + 0x20)) = 0x00;
          (*(volatile uint8_t *)((0x39) + 0x20)) = (1<<4);
          (*(volatile uint8_t *)((0x2E) + 0x20)) = 0x05;
          direction = 1;
        }
```

Pros and Cons of Using Assembly Code

Advantages:

- Errors of all development stages detectable:
  - (C) programming errors
  - compiler errors
  - errors invisible in the C code (reentrance problems, ...)
  - errors in handling the hardware (interrupts, ...)
- Instructions (relatively) easy to handle
- Clean and well documented semantics
- Implementation close to actual execution

Disadvantages:

- Hardware dependency
  $\Longrightarrow$ compiler-generating approach
- Bigger state spaces (finer granularity)
  $\Longrightarrow$ abstraction techniques

Pros and Cons of Using Assembly Code

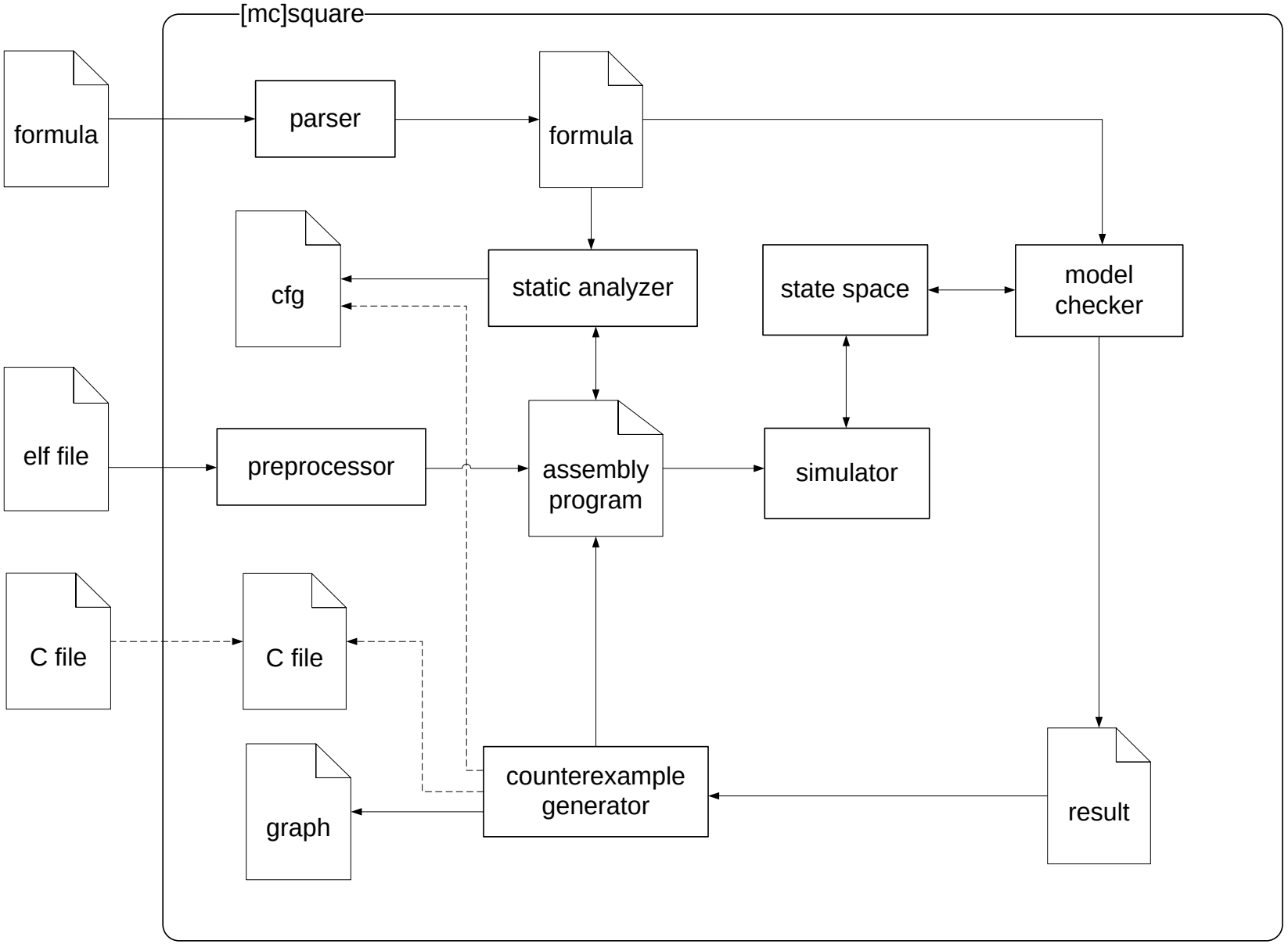
Advantages:

- Errors of all development stages detectable:
  - (C) programming errors
  - compiler errors
  - errors invisible in the C code (reentrance problems, ...)
  - errors in handling the hardware (interrupts, ...)
- Instructions (relatively) easy to handle
- Clean and well documented semantics
- Implementation close to actual execution

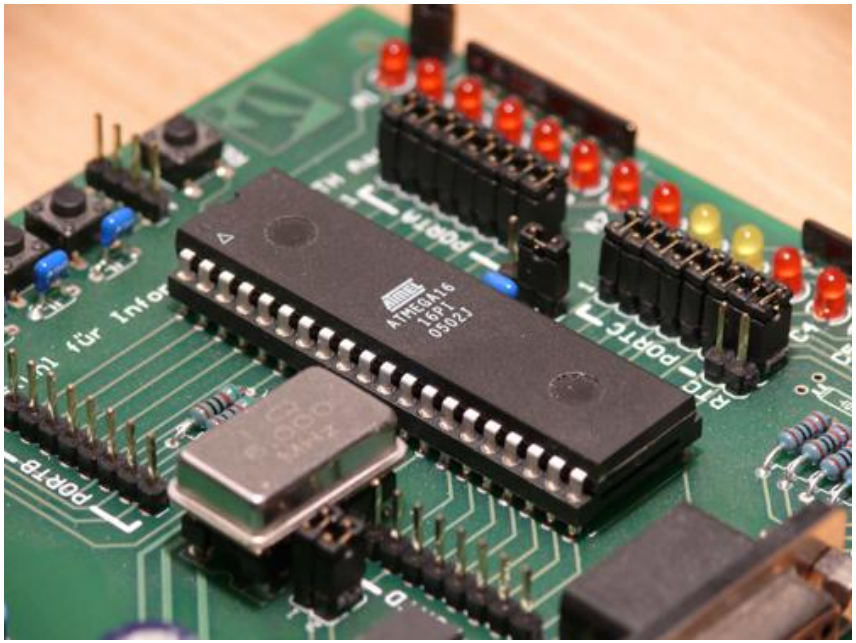
Disadvantages:

- Hardware dependency
  $\Longrightarrow$ compiler-generating approach
- Bigger state spaces (finer granularity)
  $\Longrightarrow$ abstraction techniques

The *[mc]square* Model Checker

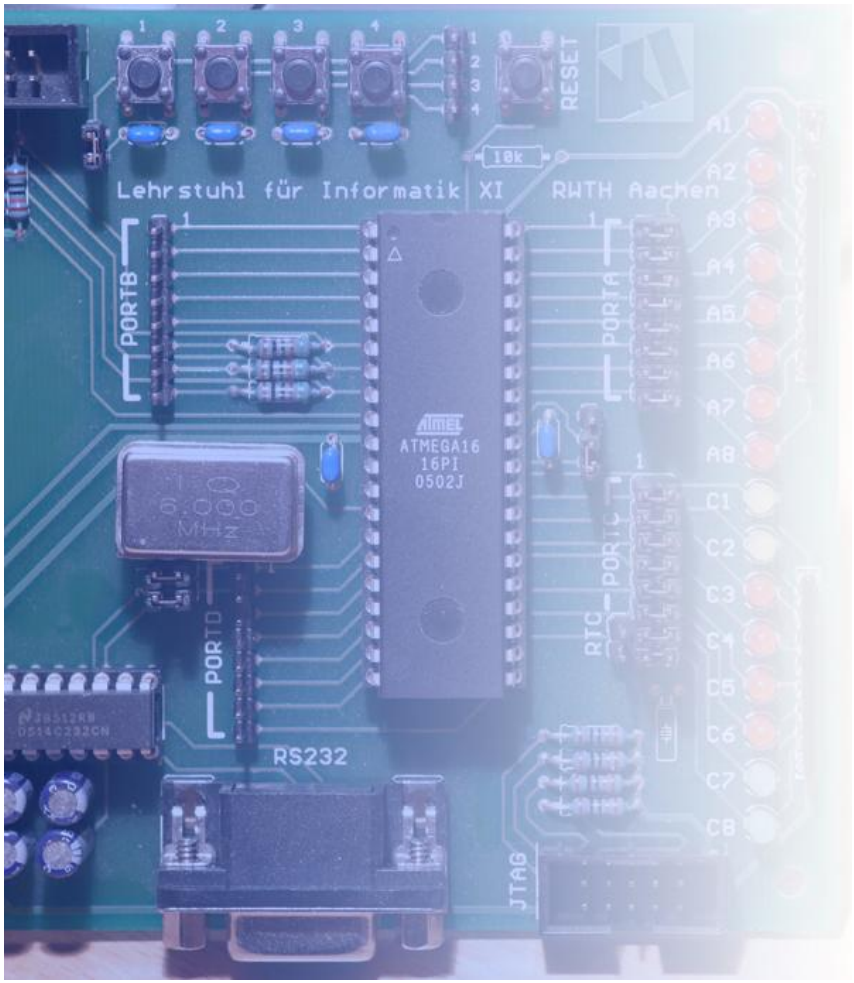


The ATMEL ATmega16 Microcontroller



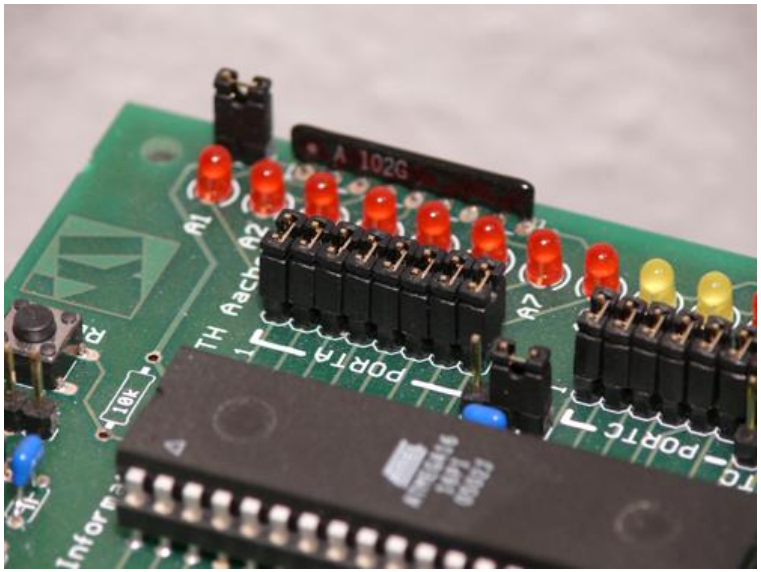
- 8-bit microcontroller
- 16 KB flash memory
- 1KByte internal SRAM
- 512 bytes EEPROM
- 3 timer/counter units
- 4 I/O Ports 8-bit
- 20 vectorized interrupts

Sources of Nondeterminism



- I/O ports
- Timer
- SPI
- TWI
- USART
- ...

I/O Ports

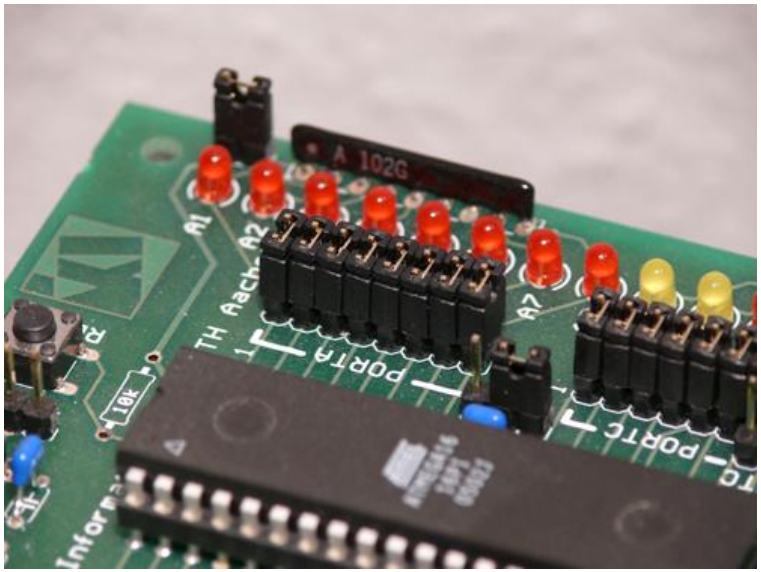


- Basic means to monitor and control external hardware
- Each 8 I/O pins (byte access)
- Bidirectional

Monitoring, access and control of I/O ports via three special registers for each port:

- Data Direction Register (`DDR`): specifies input (= 0) or output (= 1) property of corresponding pin
- Port Register (`PORT`): specifies values of output pins
- Port Input Register (`PIN`): contains values of input pins (read-only)

I/O Ports

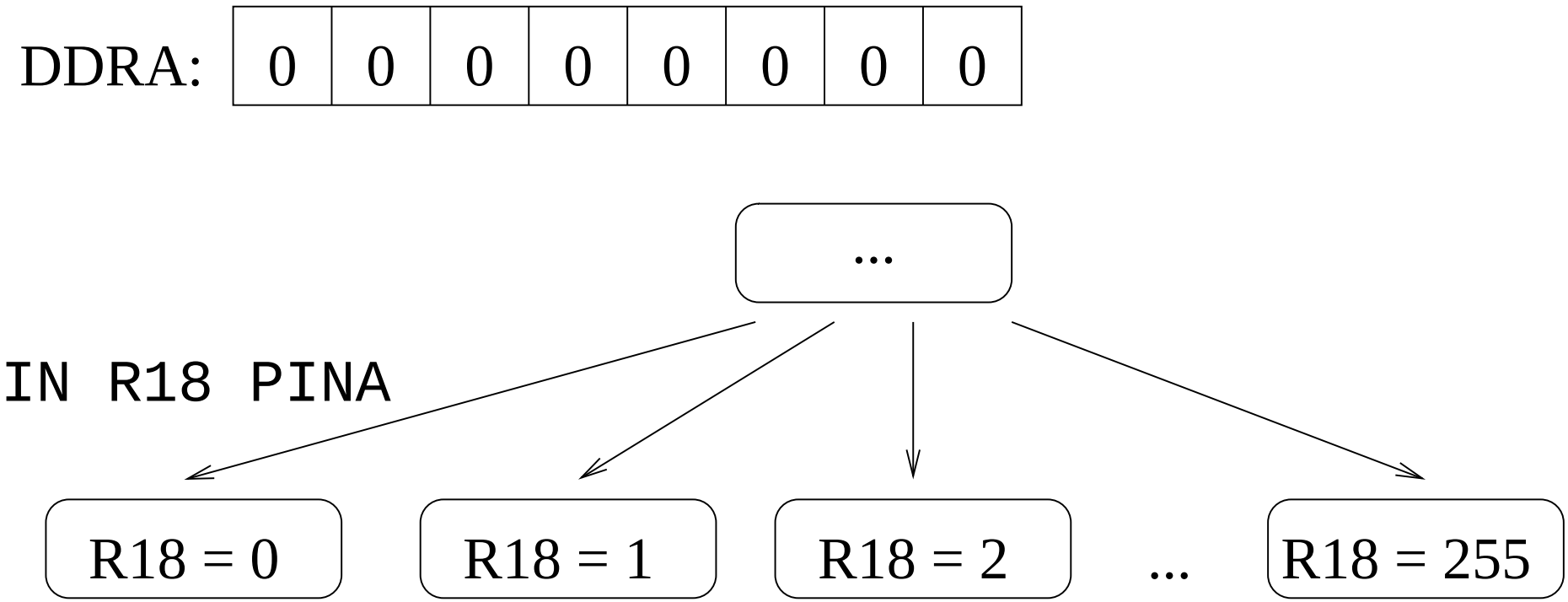


- Basic means to monitor and control external hardware
- Each 8 I/O pins (byte access)
- Bidirectional

Monitoring, access and control of I/O ports via three special registers for each port:

- Data Direction Register (`DDR`): specifies input (= 0) or output (= 1) property of corresponding pin
- Port Register (`PORT`): specifies values of output pins
- Port Input Register (`PIN`): contains values of input pins (read-only)

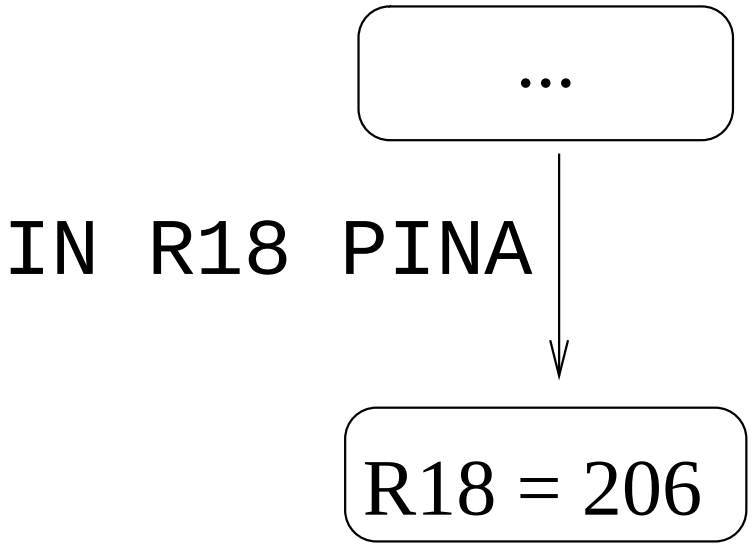
Impact on State Space I



Impact on State Space II

DDRA:	1	1	1	1	1	1	1	1

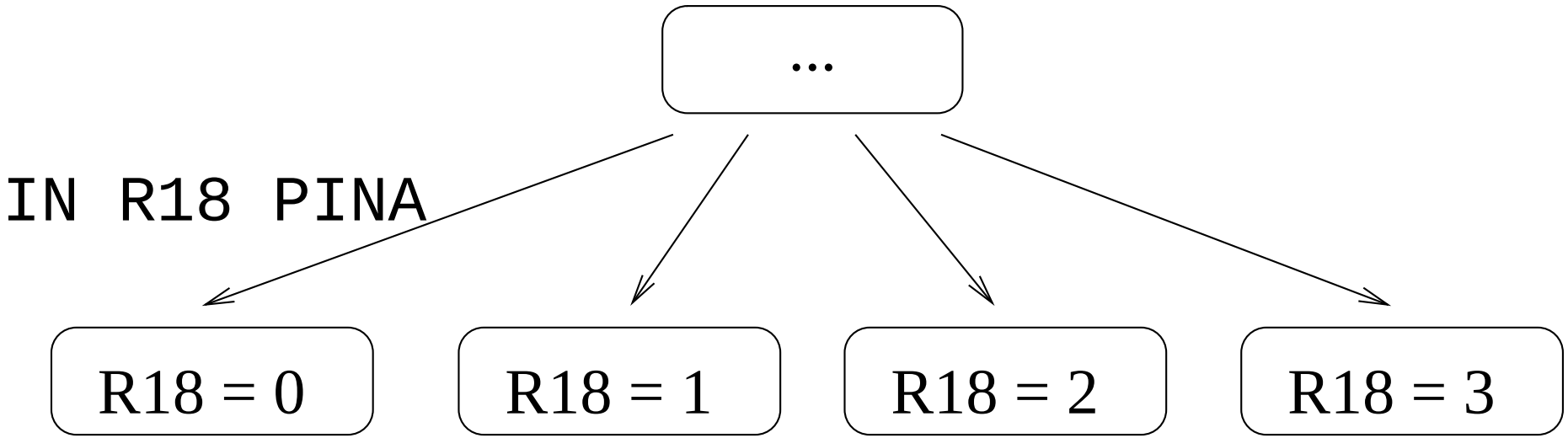
PORTA:	1	1	0	0	1	1	1	0



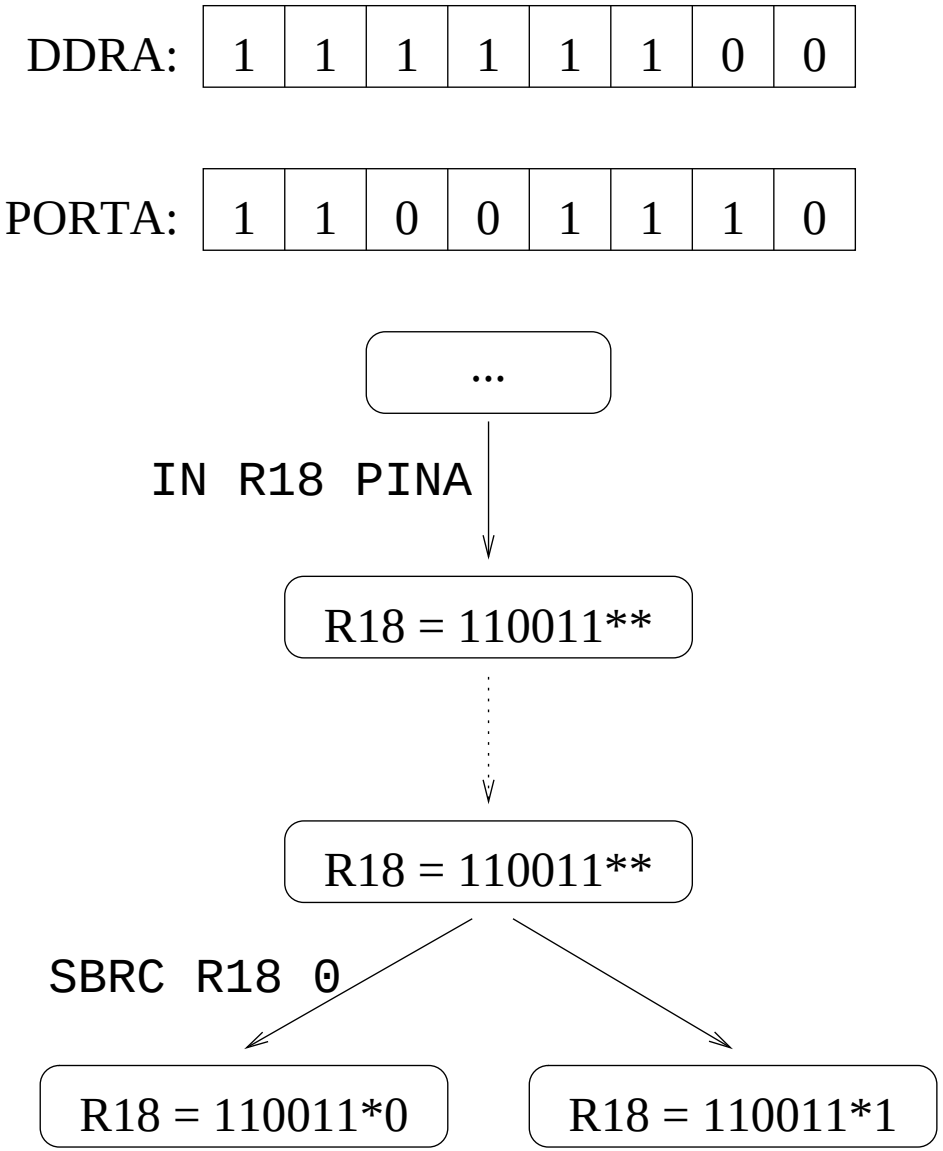
Impact on State Space III

DDRA:	1	1	1	1	1	1	0	0

PORTA:	0	0	0	0	0	0	0	0



Delayed Nondeterminism



The Formal Model: State Space

- Bits: $\mathbb{B} := \{0, 1\}$
- Bytes: $\mathbb{C} := \mathbb{B}^8$
- Memory addresses: $A := \mathbb{C}^m$ (here: $m = 2$)
- Nondeterministic bit value: $*$
- $\mathbb{B}_* := \mathbb{B} \cup \{*\}$, $\mathbb{C}_* := \mathbb{B}_*^8$
- Deterministic addresses $D \subseteq A$
  (certain registers, variables in formula, ...)
- Memory states: $V := \{v \mid v : A \to \mathbb{C}_*\}$ where $v(a) \in \mathbb{C}$ for every $a \in D$
- Control locations: Q (here: program counter)
- (System) states: $S := Q \times V$

Flow of Control

In every cycle:

1. run environment handler $g_1; \ldots; g_k$
   (introduces nondeterministic values where necessary),
2. if necessary, apply interrupt dispatcher $e_1 : q_1 > \ldots > e_l : q_l$
   (reaction to extraordinary events such as interrupts); otherwise
3. apply instruction handler $q : h_1 : q'_1 > \ldots > h_m : q'_m$
   for current location $q \in Q$
   (normal execution of machine instructions)

Here each g_i, h_i is a guarded assignment of the form

$$e_0 \to x_1 := e_1, \ldots, x_n := e_n$$

(e_j value expressions, x_j address expressions)

Flow of Control

In every cycle:

1. run environment handler $g_1; \ldots; g_k$
   (introduces nondeterministic values where necessary),
2. if necessary, apply interrupt dispatcher $e_1 : q_1 > \ldots > e_l : q_l$
   (reaction to extraordinary events such as interrupts), otherwise
3. apply instruction handler $q : h_1 : q'_1 > \ldots > h_m : q'_m$
   for current location $q \in Q$
   (normal execution of machine instructions)

Here each g_i, h_i is a guarded assignment of the form

$$e_0 \to x_1 := e_1, \ldots, x_n := e_n$$

(e_j value expressions, x_j address expressions)

Flow of Control

In every cycle:

1. run environment handler $g_1; \ldots; g_k$
   (introduces nondeterministic values where necessary),
2. if necessary, apply interrupt dispatcher $e_1 : q_1 > \ldots > e_l : q_l$
   (reaction to extraordinary events such as interrupts); otherwise
3. apply instruction handler $q : h_1 : q'_1 > \ldots > h_m : q'_m$
   for current location $q \in Q$
   (normal execution of machine instructions)

Here each g_i, h_i is a guarded assignment of the form

$$e_0 \to x_1 := e_1, \ldots, x_n := e_n$$

(e_j value expressions, x_j address expressions)

Flow of Control

In every cycle:

1. run environment handler $g_1; \ldots; g_k$
   (introduces nondeterministic values where necessary),
2. if necessary, apply interrupt dispatcher $e_1 : q_1 > \ldots > e_l : q_l$
   (reaction to extraordinary events such as interrupts); otherwise
3. apply instruction handler $q : h_1 : q'_1 > \ldots > h_m : q'_m$
   for current location $q \in Q$
   (normal execution of machine instructions)

Here each g_i, h_i is a guarded assignment of the form

$$e_0 \to x_1 := e_1, \ldots, x_n := e_n$$

(e_j value expressions, x_j address expressions)

Example: Environment Handler

$$\texttt{TCCR0[CS02]} = 1 \vee \texttt{TCCR0[CS01]} = 1 \vee \texttt{TCCR0[CS00]} = 1$$
$$\rightarrow \texttt{TIFR[TOV0]} := nd(\texttt{TIFR[TOV0]});$$
$$\texttt{DDRB[DDB2]} = 0 \rightarrow \texttt{GIFR[INTF2]} := nd(\texttt{GIFR[INTF2]}); \ldots$$

- Timer overflow interrupt possible if timer activated
- External interrupt possible if input enabled
- Here: $nd(*) := *$, $nd(0) := *$, and $nd(1) := 1$

Example: Environment Handler

$$\texttt{TCCR0[CS02]} = 1 \vee \texttt{TCCR0[CS01]} = 1 \vee \texttt{TCCR0[CS00]} = 1$$
$$\rightarrow \texttt{TIFR[TOV0]} := nd(\texttt{TIFR[TOV0]});$$
$$\texttt{DDRB[DDB2]} = 0 \rightarrow \texttt{GIFR[INTF2]} := nd(\texttt{GIFR[INTF2]}); \ldots$$

- Timer overflow interrupt possible if timer activated
- External interrupt possible if input enabled
- Here: $nd(*) := *$, $nd(0) := *$, and $nd(1) := 1$

Example: Interrupt Dispatcher

$$\texttt{SREG[I]} = 1 \wedge \texttt{TIMSK[TOIE0]} = 1 \wedge \texttt{TIFR[TOV0]} = 1 : 18 >$$
$$\texttt{SREG[I]} = 1 \wedge \texttt{GICR[INT2]} = 1 \wedge \texttt{GIFR[INTF2]} = 1 : 36 > \ldots$$

- Timer interrupt raised if
  - interrupts are globally enabled and
  - timer interrupt not masked and
  - timer overflow has occurred
- Effect: jump to interrupt handler at address 18

Example: Interrupt Dispatcher

$$\texttt{SREG[I]} = 1 \wedge \texttt{TIMSK[TOIE0]} = 1 \wedge \texttt{TIFR[TOV0]} = 1 : 18 >$$
$$\texttt{SREG[I]} = 1 \wedge \texttt{GICR[INT2]} = 1 \wedge \texttt{GIFR[INTF2]} = 1 : 36 > \ldots$$

- Timer interrupt raised if
  - interrupts are globally enabled and
  - timer interrupt not masked and
  - timer overflow has occurred
- Effect: jump to interrupt handler at address 18

Example: Interrupt Dispatcher

$$\texttt{SREG[I]} = 1 \wedge \texttt{TIMSK[TOIE0]} = 1 \wedge \texttt{TIFR[TOV0]} = 1 : 18 >$$
$$\texttt{SREG[I]} = 1 \wedge \texttt{GICR[INT2]} = 1 \wedge \texttt{GIFR[INTF2]} = 1 : 36 > \ldots$$

- Timer interrupt raised if
  - interrupts are globally enabled and
  - timer interrupt not masked and
  - timer overflow has occurred
- Effect: jump to interrupt handler at address 18

Example: Interrupt Dispatcher

$$\texttt{SREG}[\texttt{I}] = 1 \wedge \texttt{TIMSK}[\texttt{TOIE0}] = 1 \wedge \texttt{TIFR}[\texttt{TOV0}] = 1 : 18 >$$
$$\texttt{SREG}[\texttt{I}] = 1 \wedge \texttt{GICR}[\texttt{INT2}] = 1 \wedge \texttt{GIFR}[\texttt{INTF2}] = 1 : 36 > \ldots$$

- Timer interrupt raised if
  - interrupts are globally enabled and
  - timer interrupt not masked and
  - timer overflow has occurred
- Effect: jump to interrupt handler at address 18

Example: Adding Instruction

ADD Ri, Rj at address q:

$$q : \mathtt{R}i := \mathtt{R}i + \mathtt{R}j, \mathtt{SREG}[\mathtt{Z}] := (\mathtt{R}i + \mathtt{R}j = 0), \mathtt{SREG}[\mathtt{C}] := \ldots, \ldots : q + 2$$

- Adds contents of registers Ri and Rj and stores result in Ri
- Sets flags in status register SREG:
  - zero flag Z (= 0)
  - carry flag C (= 1)
  - ...
- $. + . : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$
- $. = 0 : \mathbb{C} \to \mathbb{B}$

Example: Adding Instruction

ADD Ri, Rj at address q:

$$q : \mathtt{R}i := \mathtt{R}i + \mathtt{R}j, \mathtt{SREG}[\mathtt{Z}] := (\mathtt{R}i + \mathtt{R}j = 0), \mathtt{SREG}[\mathtt{C}] := \ldots, \ldots : q + 2$$

- Adds contents of registers Ri and Rj and stores result in Ri
- Sets flags in status register SREG:
  - zero flag Z (= 0)
  - carry flag C (= 1)
  - ...
- $. + . : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$
- $. = 0 : \mathbb{C} \to \mathbb{B}$

Example: Input Instruction

`IN Ri,A` at address q:

$$q : \texttt{R}i := (\texttt{DDRA} \wedge \texttt{PORTA}) \vee (\neg\texttt{DDRA} \wedge \texttt{PINA}) : q + 2$$

- Copies contents of registers `PORTA`/`PINA` according to mask `DDRA`
- $. \wedge . : \mathbb{C} \times \mathbb{C}_* \to \mathbb{C}_*$
- $. \vee . : \mathbb{C}_* \times \mathbb{C}_* \to \mathbb{C}_*$
- $\neg . : \mathbb{C} \to \mathbb{C}$

Example: "Skip If Bit Cleared" Instruction

`SBRC Ri,b` at address q:

$$\begin{array}{rl} q : \mathtt{R}i[b] = 0 \rightarrow : q+3 & > \\ \mathtt{R}i[b] = 1 \rightarrow : q+2 & \end{array}$$

- Branches control according to bth bit in register `Ri`

Immediate Instantiation I

Immediate Instantiation

Each assignment of nondeterministic bit values is resolved by assigning all possible combinations of concrete values.

- Thus: only deterministic values are allowed to be stored
- Still $v(a) \in \mathbb{C}_* \setminus \mathbb{C}$ possible for specific addresses $a \in A \setminus D$ (e.g., $a = \texttt{PINA}$)
- Guarded assignment $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ enabled in state $(q, v) \in S$ if $[\![e_0]\!]_v = 1$
- Gives rise to concrete transition $(q, v) \xrightarrow{g} (q', v')$ for every $v' \in V$ obtained by
  1. evaluating every right-hand side expression e_i
  2. taking every possible instantiation of nondeterministic bit values
  3. updating v accordingly
- Formally:
  - $v' := v[[\![x_i]\!]_v \mapsto c_i; 1 \leq i \leq n]$ with $\mathbb{C} \cup \mathbb{B} \ni c_i \sqsubseteq [\![e_i]\!]_v$ for all $1 \leq i \leq n$
  - $\sqsubseteq \subseteq \mathbb{B}_* \times \mathbb{B}_*$ given by $0 \sqsubseteq *$ and $1 \sqsubseteq *$ (pointwise lifted to $\mathbb{C}_*$ and V)

$\Longrightarrow$ Yields concrete transition system $T^c = (S, \bigcup_{g \in G} \xrightarrow{g}, s_0)$

Immediate Instantiation I

Immediate Instantiation

Each assignment of nondeterministic bit values is resolved by assigning all possible combinations of concrete values.

- Thus: only deterministic values are allowed to be stored
- Still $v(a) \in \mathbb{C}_* \setminus \mathbb{C}$ possible for specific addresses $a \in A \setminus D$ (e.g., $a = \texttt{PINA}$)
- Guarded assignment $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ enabled in state $(q, v) \in S$ if $[\![e_0]\!]_v = 1$
- Gives rise to concrete transition $(q, v) \xrightarrow{g} (q', v')$ for every $v' \in V$ obtained by
  1. evaluating every right-hand side expression e_i
  2. taking every possible instantiation of nondeterministic bit values
  3. updating v accordingly
- Formally:
  - $v' := v[[\![x_i]\!]_v \mapsto c_i; 1 \leq i \leq n]$ with $\mathbb{C} \cup \mathbb{B} \ni c_i \sqsubseteq [\![e_i]\!]_v$ for all $1 \leq i \leq n$
  - $\sqsubseteq \subseteq \mathbb{B}_* \times \mathbb{B}_*$ given by $0 \sqsubseteq *$ and $1 \sqsubseteq *$ (pointwise lifted to $\mathbb{C}_*$ and V)

$\Longrightarrow$ Yields concrete transition system $T^c = (S, \bigcup_{g \in G} \xrightarrow{g}, s_0)$

Immediate Instantiation I

Immediate Instantiation

Each assignment of nondeterministic bit values is resolved by assigning all possible combinations of concrete values.

- Thus: only deterministic values are allowed to be stored
- Still $v(a) \in \mathbb{C}_* \setminus \mathbb{C}$ possible for specific addresses $a \in A \setminus D$ (e.g., $a = \texttt{PINA}$)
- Guarded assignment $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ enabled in state $(q, v) \in S$ if $[\![e_0]\!]_v = 1$
- Gives rise to concrete transition $(q, v) \xrightarrow{g} (q', v')$ for every $v' \in V$ obtained by
  1. evaluating every right-hand side expression e_i
  2. taking every possible instantiation of nondeterministic bit values
  3. updating v accordingly
- Formally:
  - $v' := v[[\![x_i]\!]_v \mapsto c_i; 1 \leq i \leq n]$ with $\mathbb{C} \cup \mathbb{B} \ni c_i \sqsubseteq [\![e_i]\!]_v$ for all $1 \leq i \leq n$
  - $\sqsubseteq \subseteq \mathbb{B}_* \times \mathbb{B}_*$ given by $0 \sqsubseteq *$ and $1 \sqsubseteq *$ (pointwise lifted to $\mathbb{C}_*$ and V)

$\Longrightarrow$ Yields concrete transition system $T^c = (S, \bigcup_{g \in G} \xrightarrow{g}, s_0)$

Immediate Instantiation I

Immediate Instantiation

Each assignment of nondeterministic bit values is resolved by assigning all possible combinations of concrete values.

- Thus: only deterministic values are allowed to be stored
- Still $v(a) \in \mathbb{C}_* \setminus \mathbb{C}$ possible for specific addresses $a \in A \setminus D$ (e.g., $a = \texttt{PINA}$)
- Guarded assignment $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ enabled in state $(q, v) \in S$ if $[\![e_0]\!]_v = 1$
- Gives rise to concrete transition $(q, v) \xrightarrow{g} (q', v')$ for every $v' \in V$ obtained by
  1. evaluating every right-hand side expression e_i
  2. taking every possible instantiation of nondeterministic bit values
  3. updating v accordingly
- Formally:
  - $v' := v[[\![x_i]\!]_v \mapsto c_i; 1 \leq i \leq n]$ with $\mathbb{C} \cup \mathbb{B} \ni c_i \sqsubseteq [\![e_i]\!]_v$ for all $1 \leq i \leq n$
  - $\sqsubseteq \subseteq \mathbb{B}_* \times \mathbb{B}_*$ given by $0 \sqsubseteq *$ and $1 \sqsubseteq *$ (pointwise lifted to $\mathbb{C}_*$ and V)

$\Longrightarrow$ Yields concrete transition system $T^c = (S, \bigcup_{g \in G} \xrightarrow{g}, s_0)$

Immediate Instantiation I

Immediate Instantiation

Each assignment of nondeterministic bit values is resolved by assigning all possible combinations of concrete values.

- Thus: only deterministic values are allowed to be stored
- Still $v(a) \in \mathbb{C}_* \setminus \mathbb{C}$ possible for specific addresses $a \in A \setminus D$ (e.g., $a = \texttt{PINA}$)
- Guarded assignment $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ enabled in state $(q, v) \in S$ if $[\![e_0]\!]_v = 1$
- Gives rise to concrete transition $(q, v) \xrightarrow{g} (q', v')$ for every $v' \in V$ obtained by
  1. evaluating every right-hand side expression e_i
  2. taking every possible instantiation of nondeterministic bit values
  3. updating v accordingly
- Formally:
  - $v' := v[[\![x_i]\!]_v \mapsto c_i; 1 \leq i \leq n]$ with $\mathbb{C} \cup \mathbb{B} \ni c_i \sqsubseteq [\![e_i]\!]_v$ for all $1 \leq i \leq n$
  - $\sqsubseteq \subseteq \mathbb{B}_* \times \mathbb{B}_*$ given by $0 \sqsubseteq *$ and $1 \sqsubseteq *$ (pointwise lifted to $\mathbb{C}_*$ and V)

$\Longrightarrow$ Yields concrete transition system $T^c = (S, \bigcup_{g \in G} \xrightarrow{g}, s_0)$

Immediate Instantiation I

Immediate Instantiation

Each assignment of nondeterministic bit values is resolved by assigning all possible combinations of concrete values.

- Thus: only deterministic values are allowed to be stored
- Still $v(a) \in \mathbb{C}_* \setminus \mathbb{C}$ possible for specific addresses $a \in A \setminus D$ (e.g., $a = \texttt{PINA}$)
- Guarded assignment $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ enabled in state $(q, v) \in S$ if $[\![e_0]\!]_v = 1$
- Gives rise to concrete transition $(q, v) \xrightarrow{g} (q', v')$ for every $v' \in V$ obtained by
  1. evaluating every right-hand side expression e_i
  2. taking every possible instantiation of nondeterministic bit values
  3. updating v accordingly
- Formally:
  - $v' := v[[\![x_i]\!]_v \mapsto c_i; 1 \leq i \leq n]$ with $\mathbb{C} \cup \mathbb{B} \ni c_i \sqsubseteq [\![e_i]\!]_v$ for all $1 \leq i \leq n$
  - $\sqsubseteq \subseteq \mathbb{B}_* \times \mathbb{B}_*$ given by $0 \sqsubseteq *$ and $1 \sqsubseteq *$ (pointwise lifted to $\mathbb{C}_*$ and V)

$\Longrightarrow$ Yields concrete transition system $T^c = (S, \bigcup_{g \in G} \xrightarrow{g}, s_0)$

Immediate Instantiation II

Example

- $v(\texttt{DDRA}) = 11111100$, $v(\texttt{PORTA}) = 00000000$, $v(\texttt{PINA}) = ********$
- Execution of `IN R1,A` at address q:

$$q : \texttt{R1} := (\texttt{DDRA} \wedge \texttt{PORTA}) \vee (\neg\texttt{DDRA} \wedge \texttt{PINA}) : q + 2$$

- Transitions from (q, v) to
  1. $(q + 2, v[\texttt{R1} \mapsto 00000000])$
  2. $(q + 2, v[\texttt{R1} \mapsto 00000001])$
  3. $(q + 2, v[\texttt{R1} \mapsto 00000010])$
  4. $(q + 2, v[\texttt{R1} \mapsto 00000011])$

Immediate Instantiation II

Example

- $v(\texttt{DDRA}) = 11111100$, $v(\texttt{PORTA}) = 00000000$,
  $v(\texttt{PINA}) = *********$
- Execution of `IN R1,A` at address q:

$$q : \texttt{R1} := (\texttt{DDRA} \wedge \texttt{PORTA}) \vee (\neg\texttt{DDRA} \wedge \texttt{PINA}) : q + 2$$

- Transitions from (q, v) to
  1. $(q + 2, v[\texttt{R1} \mapsto 00000000])$
  2. $(q + 2, v[\texttt{R1} \mapsto 00000001])$
  3. $(q + 2, v[\texttt{R1} \mapsto 00000010])$
  4. $(q + 2, v[\texttt{R1} \mapsto 00000011])$

Immediate Instantiation II

Example

- $v(\texttt{DDRA}) = 11111100$, $v(\texttt{PORTA}) = 00000000$,
  $v(\texttt{PINA}) = *********$
- Execution of `IN R1,A` at address q:

$$q : \texttt{R1} := (\texttt{DDRA} \wedge \texttt{PORTA}) \vee (\neg\texttt{DDRA} \wedge \texttt{PINA}) : q + 2$$

- Transitions from (q, v) to
  1. $(q + 2, v[\texttt{R1} \mapsto 00000000])$
  2. $(q + 2, v[\texttt{R1} \mapsto 00000001])$
  3. $(q + 2, v[\texttt{R1} \mapsto 00000010])$
  4. $(q + 2, v[\texttt{R1} \mapsto 00000011])$

Delayed Nondeterminism I

Delayed Nondeterminism

Replace nondeterministic by concrete values only if and when this is required by a subsequent computation step.

Delayed Nondeterminism II

Informally: instantiation of $v(a)[b] = *$ in $(q, v) \in S$ required when ex. $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ s.t.

- (a, b) referred by e_0, or
- g enabled and some e_i refers to (a, b) in a deterministic argument, or
- some x_i dereferences a, or
- some e_i yields $*$ and x_i deterministic

Formally: g induces abstract transition $(q, v) \overset{g}{\Longrightarrow} (q', v')$ if ex. $v_1, v_2, v_3, v_4 \in V$ s.t.

1. $v_1 \sqsubseteq v$ with $v_1(a, b) \neq *$ if (a, b) referred by e_0, and
2. $[\![e_0]\!]_{v_1} = 1$, and
3. $v_2 \sqsubseteq v_1$ with $v_2(a, b) \neq *$ if some $e_i = op(y_1, \ldots, y_n)$, $op : T_1 \times \ldots \times T_n \to T_0$, and (a, b) referred by some y_j where $T_j \in \{\mathbb{C}, \mathbb{B}\}$, and
4. $v_3 \sqsubseteq v_2$ with $v_3(a, b) \neq *$ if some $x_i = a\downarrow + d$, and
5. $v_4 := v_3[[\![x_i]\!]_{v_3} \mapsto [\![e_i]\!]_{v_3}; 1 \leq i \leq n]$, and
6. $v' \leq v_4$ with $v'(a, b) \neq *$ if $[\![x_i]\!]_{v_4} \in \{a, (a, b)\}$ for some i, $a \in D$

$\Longrightarrow$ Yields abstract transition system $T^a = (S, \bigcup_{g \in G} \overset{g}{\Longrightarrow}, s_0)$

Delayed Nondeterminism II

Informally: instantiation of $v(a)[b] = *$ in $(q, v) \in S$ required when ex. $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ s.t.

- (a, b) referred by e_0, or
- g enabled and some e_i refers to (a, b) in a deterministic argument, or
- some x_i dereferences a, or
- some e_i yields $*$ and x_i deterministic

Formally: g induces abstract transition $(q, v) \overset{g}{\Longrightarrow} (q', v')$ if ex. $v_1, v_2, v_3, v_4 \in V$ s.t.

1. $v_1 \sqsubseteq v$ with $v_1(a, b) \neq *$ if (a, b) referred by e_0, and
2. $[\![e_0]\!]_{v_1} = 1$, and
3. $v_2 \sqsubseteq v_1$ with $v_2(a, b) \neq *$ if some $e_i = op(y_1, \ldots, y_n)$, $op : T_1 \times \ldots \times T_n \to T_0$, and (a, b) referred by some y_j where $T_j \in \{\mathbb{C}, \mathbb{B}\}$, and
4. $v_3 \sqsubseteq v_2$ with $v_3(a, b) \neq *$ if some $x_i = a\downarrow + d$, and
5. $v_4 := v_3[[\![x_i]\!]_{v_3} \mapsto [\![e_i]\!]_{v_3}; 1 \leq i \leq n]$, and
6. $v' \leq v_4$ with $v'(a, b) \neq *$ if $[\![x_i]\!]_{v_4} \in \{a, (a, b)\}$ for some $i, a \in D$

$\Longrightarrow$ Yields abstract transition system $T^a = (S, \bigcup_{g \in G} \overset{g}{\Longrightarrow}, s_0)$

Delayed Nondeterminism II

Informally: instantiation of $v(a)[b] = *$ in $(q, v) \in S$ required when ex. $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ s.t.

- (a, b) referred by e_0, or
- g enabled and some e_i refers to (a, b) in a deterministic argument, or
- some x_i dereferences a, or
- some e_i yields $*$ and x_i deterministic

Formally: g induces abstract transition $(q, v) \overset{g}{\Longrightarrow} (q', v')$ if ex. $v_1, v_2, v_3, v_4 \in V$ s.t.

1. $v_1 \sqsubseteq v$ with $v_1(a, b) \neq *$ if (a, b) referred by e_0, and
2. $[\![e_0]\!]_{v_1} = 1$, and
3. $v_2 \sqsubseteq v_1$ with $v_2(a, b) \neq *$ if some $e_i = op(y_1, \ldots, y_n)$, $op : T_1 \times \ldots \times T_n \to T_0$, and (a, b) referred by some y_j where $T_j \in \{\mathbb{C}, \mathbb{B}\}$, and
4. $v_3 \sqsubseteq v_2$ with $v_3(a, b) \neq *$ if some $x_i = a\downarrow + d$, and
5. $v_4 := v_3[[\![x_i]\!]_{v_3} \mapsto [\![e_i]\!]_{v_3}; 1 \leq i \leq n]$, and
6. $v' \leq v_4$ with $v'(a, b) \neq *$ if $[\![x_i]\!]_{v_4} \in \{a, (a, b)\}$ for some $i, a \in D$

$\Longrightarrow$ Yields abstract transition system $T^a = (S, \bigcup_{g \in G} \overset{g}{\Longrightarrow}, s_0)$

Delayed Nondeterminism II

Informally: instantiation of $v(a)[b] = *$ in $(q, v) \in S$ required when ex. $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ s.t.

- (a, b) referred by e_0, or
- g enabled and some e_i refers to (a, b) in a deterministic argument, or
- some x_i dereferences a, or
- some e_i yields $*$ and x_i deterministic

Formally: g induces abstract transition $(q, v) \xRightarrow{g} (q', v')$ if ex. $v_1, v_2, v_3, v_4 \in V$ s.t.

1. $v_1 \sqsubseteq v$ with $v_1(a, b) \neq *$ if (a, b) referred by e_0, and
2. $[\![e_0]\!]_{v_1} = 1$, and
3. $v_2 \sqsubseteq v_1$ with $v_2(a, b) \neq *$ if some $e_i = op(y_1, \ldots, y_n)$, $op : T_1 \times \ldots \times T_n \to T_0$, and (a, b) referred by some y_j where $T_j \in \{\mathbb{C}, \mathbb{B}\}$, and
4. $v_3 \sqsubseteq v_2$ with $v_3(a, b) \neq *$ if some $x_i = a\downarrow + d$, and
5. $v_4 := v_3[[\![x_i]\!]_{v_3} \mapsto [\![e_i]\!]_{v_3}; 1 \leq i \leq n]$, and
6. $v' \leq v_4$ with $v'(a, b) \neq *$ if $[\![x_i]\!]_{v_4} \in \{a, (a, b)\}$ for some $i, a \in D$

$\Longrightarrow$ Yields abstract transition system $T^a = (S, \bigcup_{g \in G} \xRightarrow{g}, s_0)$

Delayed Nondeterminism II

Informally: instantiation of $v(a)[b] = *$ in $(q, v) \in S$ required when ex. $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ s.t.

- (a, b) referred by e_0, or
- g enabled and some e_i refers to (a, b) in a deterministic argument, or
- some x_i dereferences a, or
- some e_i yields $*$ and x_i deterministic

Formally: g induces abstract transition $(q, v) \overset{g}{\Longrightarrow} (q', v')$ if ex. $v_1, v_2, v_3, v_4 \in V$ s.t.

1. $v_1 \sqsubseteq v$ with $v_1(a, b) \neq *$ if (a, b) referred by e_0, and
2. $[\![e_0]\!]_{v_1} = 1$, and
3. $v_2 \sqsubseteq v_1$ with $v_2(a, b) \neq *$ if some $e_i = op(y_1, \ldots, y_n)$, $op : T_1 \times \ldots \times T_n \to T_0$, and (a, b) referred by some y_j where $T_j \in \{\mathbb{C}, \mathbb{B}\}$, and
4. $v_3 \sqsubseteq v_2$ with $v_3(a, b) \neq *$ if some $x_i = a\downarrow + d$, and
5. $v_4 := v_3[[\![x_i]\!]_{v_3} \mapsto [\![e_i]\!]_{v_3}; 1 \leq i \leq n]$, and
6. $v' \leq v_4$ with $v'(a, b) \neq *$ if $[\![x_i]\!]_{v_4} \in \{a, (a, b)\}$ for some i, $a \in D$

$\Longrightarrow$ Yields abstract transition system $T^a = (S, \bigcup_{g \in G} \overset{g}{\Longrightarrow}, s_0)$

Delayed Nondeterminism II

Informally: instantiation of $v(a)[b] = *$ in $(q, v) \in S$ required when ex. $g = q : e_0 \to x_1 := e_1, \ldots, x_n := e_n : q'$ s.t.

- (a, b) referred by e_0, or
- g enabled and some e_i refers to (a, b) in a deterministic argument, or
- some x_i dereferences a, or
- some e_i yields $*$ and x_i deterministic

Formally: g induces abstract transition $(q, v) \overset{g}{\Longrightarrow} (q', v')$ if ex. $v_1, v_2, v_3, v_4 \in V$ s.t.

1. $v_1 \sqsubseteq v$ with $v_1(a, b) \neq *$ if (a, b) referred by e_0, and
2. $[\![e_0]\!]_{v_1} = 1$, and
3. $v_2 \sqsubseteq v_1$ with $v_2(a, b) \neq *$ if some $e_i = op(y_1, \ldots, y_n)$, $op : T_1 \times \ldots \times T_n \to T_0$, and (a, b) referred by some y_j where $T_j \in \{\mathbb{C}, \mathbb{B}\}$, and
4. $v_3 \sqsubseteq v_2$ with $v_3(a, b) \neq *$ if some $x_i = a\downarrow + d$, and
5. $v_4 := v_3[[\![x_i]\!]_{v_3} \mapsto [\![e_i]\!]_{v_3}; 1 \leq i \leq n]$, and
6. $v' \leq v_4$ with $v'(a, b) \neq *$ if $[\![x_i]\!]_{v_4} \in \{a, (a, b)\}$ for some i, $a \in D$

$\Longrightarrow$ Yields abstract transition system $T^a = (S, \bigcup_{g \in G} \overset{g}{\Longrightarrow}, s_0)$

Delayed Nondeterminism III

Example

- $v(\texttt{DDRA}) = 11111100$, $v(\texttt{PORTA}) = 00000000$,
  $v(\texttt{PINA}) = ********$
- Execution of `IN R1,A` at address q:
  - $q : \texttt{R1} := (\texttt{DDRA} \wedge \texttt{PORTA}) \vee (\neg\texttt{DDRA} \wedge \texttt{PINA}) : q + 2$
  - yields transition $(q, v) \Longrightarrow (\underbrace{q + 2}_{q'}, \underbrace{v[\texttt{R1} \mapsto 000000 * *]}_{v'})$
- Execution of `SBRC R1,0` at address q':
  - $q' : \texttt{R1}[0] = 0 \rightarrow : q' + 3$
    $q' : \texttt{R1}[0] = 1 \rightarrow : q' + 2$
  - yields $(q', v') \Longrightarrow (q' + 3, v[\texttt{R1} \mapsto 000000 * 0])$
    and $(q', v') \Longrightarrow (q' + 2, v[\texttt{R1} \mapsto 000000 * 1])$

Delayed Nondeterminism III

Example

- $v(\texttt{DDRA}) = 11111100$, $v(\texttt{PORTA}) = 00000000$,
  $v(\texttt{PINA}) = ********$
- Execution of `IN R1,A` at address q:
  - $q : \texttt{R1} := (\texttt{DDRA} \wedge \texttt{PORTA}) \vee (\neg\texttt{DDRA} \wedge \texttt{PINA}) : q + 2$
  - yields transition $(q, v) \Longrightarrow (\underbrace{q+2}_{q'}, \underbrace{v[\texttt{R1} \mapsto 000000 * *]}_{v'})$
- Execution of `SBRC R1,0` at address q':
  - $q' : \texttt{R1}[0] = 0 \rightarrow : q' + 3$
    $q' : \texttt{R1}[0] = 1 \rightarrow : q' + 2$
  - yields $(q', v') \Longrightarrow (q' + 3, v[\texttt{R1} \mapsto 000000 * 0])$
    and $(q', v') \Longrightarrow (q' + 2, v[\texttt{R1} \mapsto 000000 * 1])$

Delayed Nondeterminism III

Example

- $v(\texttt{DDRA}) = 11111100$, $v(\texttt{PORTA}) = 00000000$,
  $v(\texttt{PINA}) = * * * * * * * *$
- Execution of `IN R1,A` at address q:
  - $q : \texttt{R1} := (\texttt{DDRA} \wedge \texttt{PORTA}) \vee (\neg\texttt{DDRA} \wedge \texttt{PINA}) : q + 2$
  - yields transition $(q, v) \Longrightarrow (\underbrace{q+2}_{q'}, \underbrace{v[\texttt{R1} \mapsto 000000 * *]}_{v'})$
- Execution of `SBRC R1,0` at address q':
  - $q' : \texttt{R1}[0] = 0 \rightarrow : q' + 3$
    $q' : \texttt{R1}[0] = 1 \rightarrow : q' + 2$
  - yields $(q', v') \Longrightarrow (q' + 3, v[\texttt{R1} \mapsto 000000 * 0])$
    and $(q', v') \Longrightarrow (q' + 2, v[\texttt{R1} \mapsto 000000 * 1])$

Soundness of Abstraction I

- Property specification given by temporal formula φ over set P of bit value expressions
- Defines labeling $\lambda : S \to 2^P : (q, v) \mapsto \{p \in P \mid [\![p]\!]_v = 1\}$
  - concrete LTS: $L^c = (S, \bigcup_{g \in G} \xrightarrow{g}, s_0, \lambda)$
  - abstract LTS: $L^a = (S, \bigcup_{g \in G} \stackrel{g}{\Longrightarrow}, s_0, \lambda)$
  - note: $[\![p]\!]_v$ always defined since $Var(\varphi) \subseteq D$
- Connection between L^c and L^a given by simulation:
  a binary relation $\rho \subseteq S \times S$ such that $s_0 \rho s_0$ and, whenever $s_1 \rho s_2$,
  - $\lambda(s_1) = \lambda(s_2)$ and
  - for every transition $s_1 \xrightarrow{g} s_1'$
    there exists $s_2' \in S$
    such that $s_2 \stackrel{g}{\Longrightarrow} s_2'$ and $s_1' \rho s_2'$

Soundness of Abstraction I

- Property specification given by temporal formula φ over set P of bit value expressions
- Defines labeling $\lambda : S \to 2^P : (q, v) \mapsto \{p \in P \mid [\![p]\!]_v = 1\}$
  - concrete LTS: $L^c = (S, \bigcup_{g \in G} \xrightarrow{g}, s_0, \lambda)$
  - abstract LTS: $L^a = (S, \bigcup_{g \in G} \stackrel{g}{\Longrightarrow}, s_0, \lambda)$
  - note: $[\![p]\!]_v$ always defined since $Var(\varphi) \subseteq D$
- Connection between L^c and L^a given by simulation:
  a binary relation $\rho \subseteq S \times S$ such that $s_0 \rho s_0$ and, whenever $s_1 \rho s_2$,
  - $\lambda(s_1) = \lambda(s_2)$ and
  - for every transition $s_1 \xrightarrow{g} s_1'$
    there exists $s_2' \in S$
    such that $s_2 \stackrel{g}{\Longrightarrow} s_2'$ and $s_1' \rho s_2'$

Soundness of Abstraction II

Theorem

L^a simulates L^c.

Proof.

Simulation relation given by partial order on bit values:

$$(q_1, v_1)\rho(q_2, v_2) \text{ iff } q_1 = q_2 \text{ and } v_1 \sqsubseteq v_2$$

□

Corollary

Delayed nondeterminism is sound w.r.t. "path-universal" temporal logics such as LTL or ACTL:

$$L^a \models \varphi \text{ implies } L^c \models \varphi$$

(i.e., no false positives)

Soundness of Abstraction II

Theorem

L^a simulates L^c.

Proof.

Simulation relation given by partial order on bit values:

$$(q_1, v_1)\rho(q_2, v_2) \text{ iff } q_1 = q_2 \text{ and } v_1 \sqsubseteq v_2$$

□

Corollary

Delayed nondeterminism is sound w.r.t. "path-universal" temporal logics such as LTL or ACTL:

$$L^a \models \varphi \text{ implies } L^c \models \varphi$$

(i.e., no false positives)

Soundness of Abstraction II

Theorem

L^a simulates L^c.

Proof.

Simulation relation given by partial order on bit values:

$$(q_1, v_1)\rho(q_2, v_2) \text{ iff } q_1 = q_2 \text{ and } v_1 \sqsubseteq v_2$$

□

Corollary

Delayed nondeterminism is sound w.r.t. "path-universal" temporal logics such as LTL or ACTL:

$$L^a \models \varphi \text{ implies } L^c \models \varphi$$

(i.e., no false positives)

Empirical Results

Program	Instantiation	# states stored	# states created	Size [MB]	Time [s]	Reduction
light switch	immediate	6,624	9,050	2	0.2	-
	delayed	352	380	< 1	0.01	95%
plant	immediate	801,616	854,203	256	23.19	-
	delayed	188,404	195,955	61	5.05	76%
reentrance problem	immediate	107,649	110,961	33	2.8	-
	delayed	107,649	110,961	33	2.8	0%
traffic light	immediate	35,613	38,198	11	0.92	-
	delayed	10,004	10,520	3	0.28	72%
window lift	immediate	10,100,400	11,196,174	2,049	416.98	-
	delayed	323,450	444,191	102	10.78	97%

Conclusion

- Delayed Nondeterminism: abstraction technique for explicit-state model checking of microcontroller code
  (similar to “X-valued simulation” in hardware verification)
- Significant reduction of state space in concrete examples
- Over-approximation:
  - sound for path-universal logics (simulation)
  - generally incomplete (false negatives due to copying of nondeterminism)
- Future work:
  - handle more microcontrollers (Infineon Tricore, Intel MC, ...)
  - establish bisimulation by introducing identities for nondeterministic values
  - implement compiler generator for state-space evaluators

Conclusion

- Delayed Nondeterminism: abstraction technique for explicit-state model checking of microcontroller code
  (similar to “X-valued simulation” in hardware verification)
- Significant reduction of state space in concrete examples
- Over-approximation:
  - sound for path-universal logics (simulation)
  - generally incomplete (false negatives due to copying of nondeterminism)
- Future work:
  - handle more microcontrollers (Infineon Tricore, Intel MC, ...)
  - establish bisimulation by introducing identities for nondeterministic values
  - implement compiler generator for state-space evaluators

Conclusion

- Delayed Nondeterminism: abstraction technique for explicit-state model checking of microcontroller code
  (similar to "X-valued simulation" in hardware verification)
- Significant reduction of state space in concrete examples
- Over-approximation:
  - sound for path-universal logics (simulation)
  - generally incomplete (false negatives due to copying of nondeterminism)
- Future work:
  - handle more microcontrollers (Infineon Tricore, Intel MC, ...)
  - establish bisimulation by introducing identities for nondeterministic values
  - implement compiler generator for state-space evaluators

Conclusion

- Delayed Nondeterminism: abstraction technique for explicit-state model checking of microcontroller code
  (similar to "X-valued simulation" in hardware verification)
- Significant reduction of state space in concrete examples
- Over-approximation:
  - sound for path-universal logics (simulation)
  - generally incomplete (false negatives due to copying of nondeterminism)
- Future work:
  - handle more microcontrollers (Infineon Tricore, Intel MC, ...)
  - establish bisimulation by introducing identities for nondeterministic values
  - implement compiler generator for state-space evaluators

Incompleteness of Abstraction

Observation: over-approximation due to copying of nondeterminism

Example

$$
\begin{array}{lll}
q_1: & & a[0] := * \;\; : q_2 \\
q_2: & & a[1] := a[0] : q_3 \\
q_3: a[0] \wedge \neg a[1] \rightarrow & & \;\; : q_4
\end{array}
$$

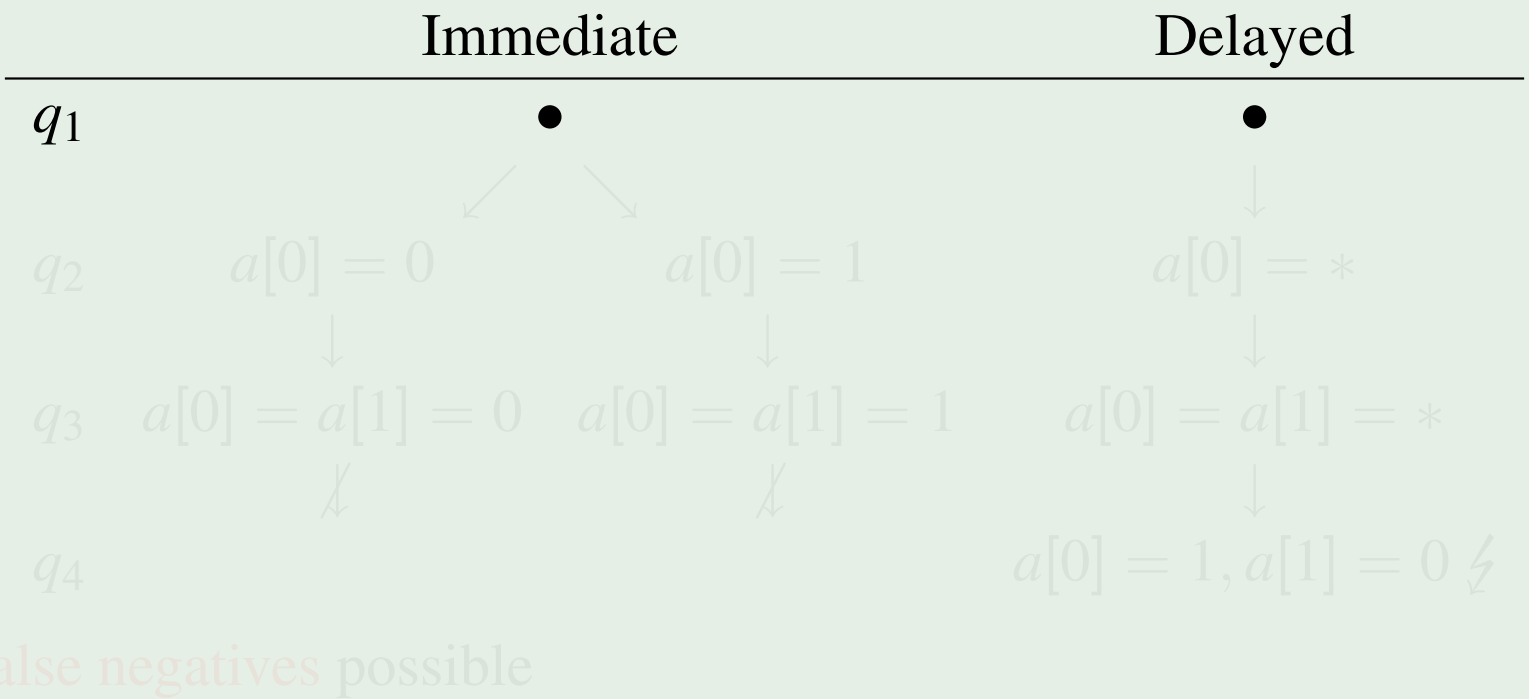
$\Longrightarrow$ false negatives possible

Incompleteness of Abstraction

Observation: over-approximation due to copying of nondeterminism

Example

$$
\begin{array}{lll}
q_1: & a[0] := * & : q_2 \\
q_2: & a[1] := a[0] & : q_3 \\
q_3: a[0] \wedge \neg a[1] \rightarrow & & : q_4
\end{array}
$$



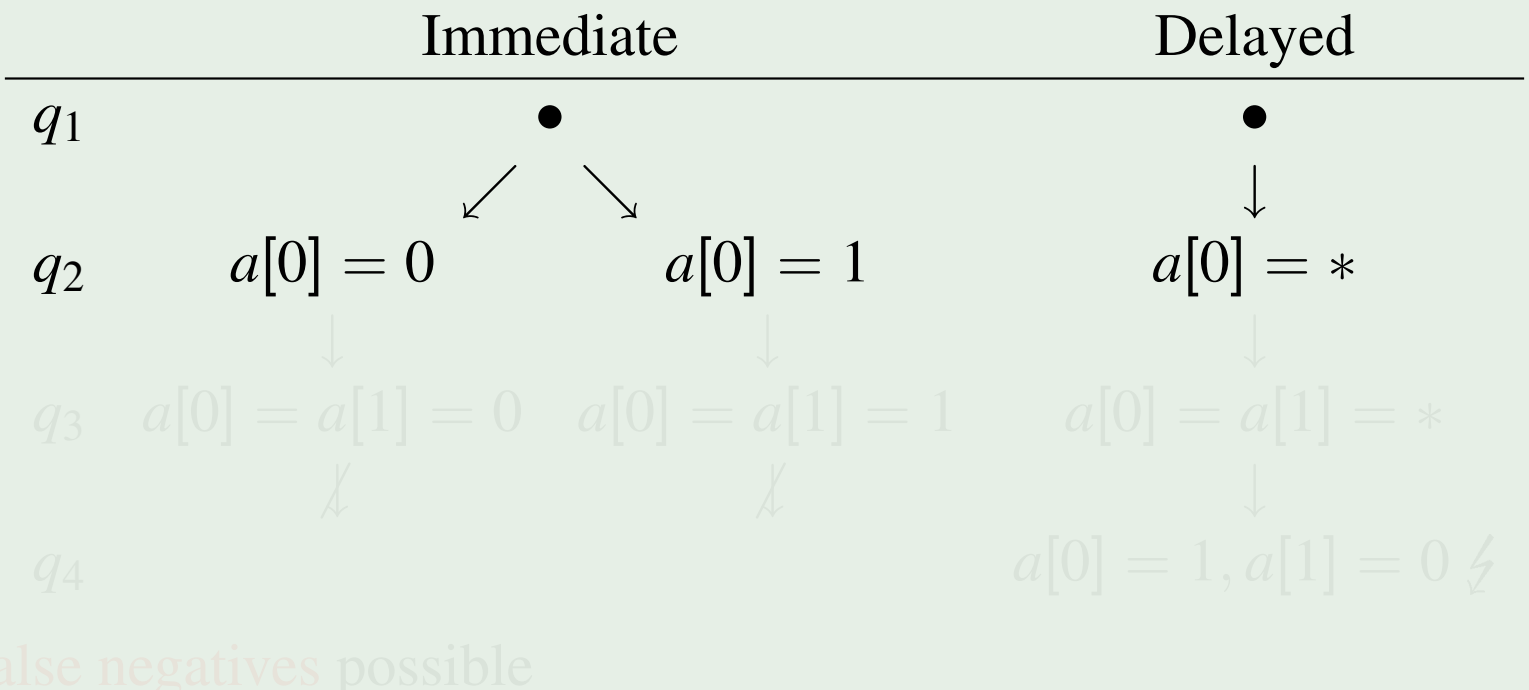
$\Longrightarrow$ false negatives possible

Incompleteness of Abstraction

Observation: over-approximation due to copying of nondeterminism

Example

$$
\begin{array}{lll}
q_1\colon & a[0] := * & \colon q_2 \\
q_2\colon & a[1] := a[0] & \colon q_3 \\
q_3\colon a[0] \wedge \neg a[1] \rightarrow & & \colon q_4
\end{array}
$$



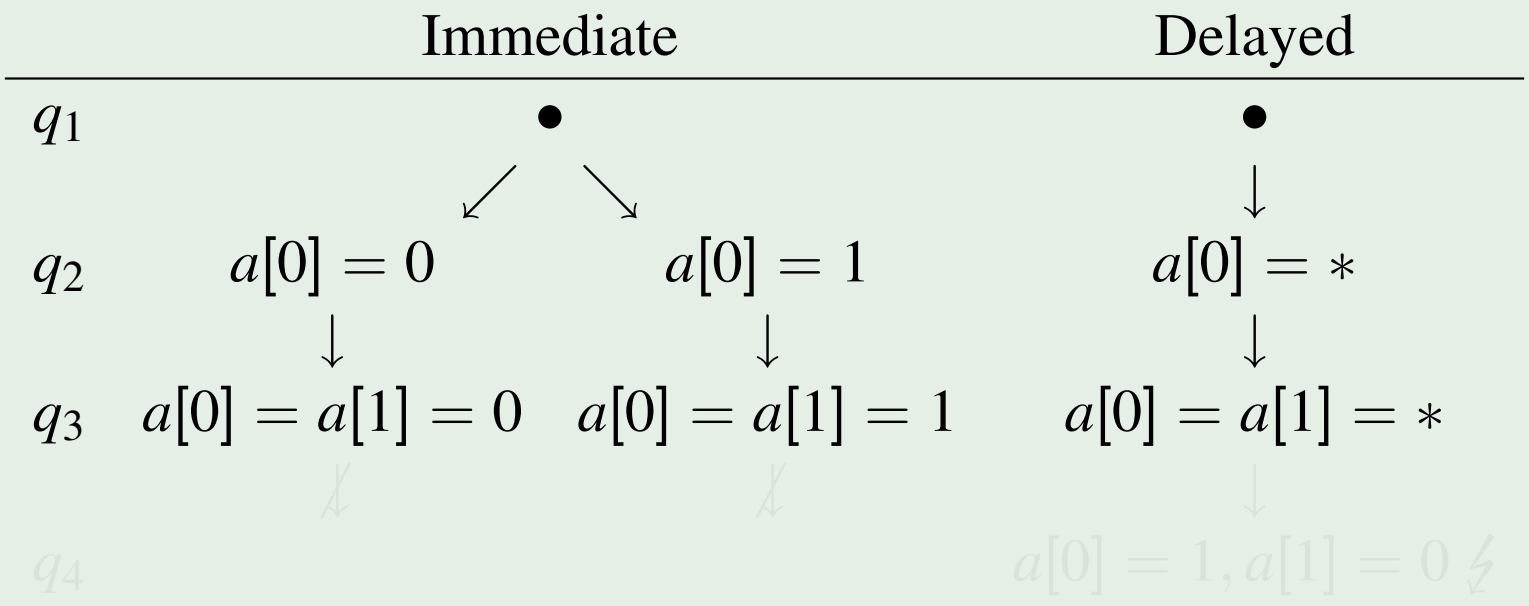
$\Longrightarrow$ false negatives possible

Incompleteness of Abstraction

Observation: over-approximation due to copying of nondeterminism

Example

$$
\begin{array}{lll}
q_1: & a[0] := * & : q_2 \\
q_2: & a[1] := a[0] & : q_3 \\
q_3: a[0] \wedge \neg a[1] \rightarrow & & : q_4
\end{array}
$$



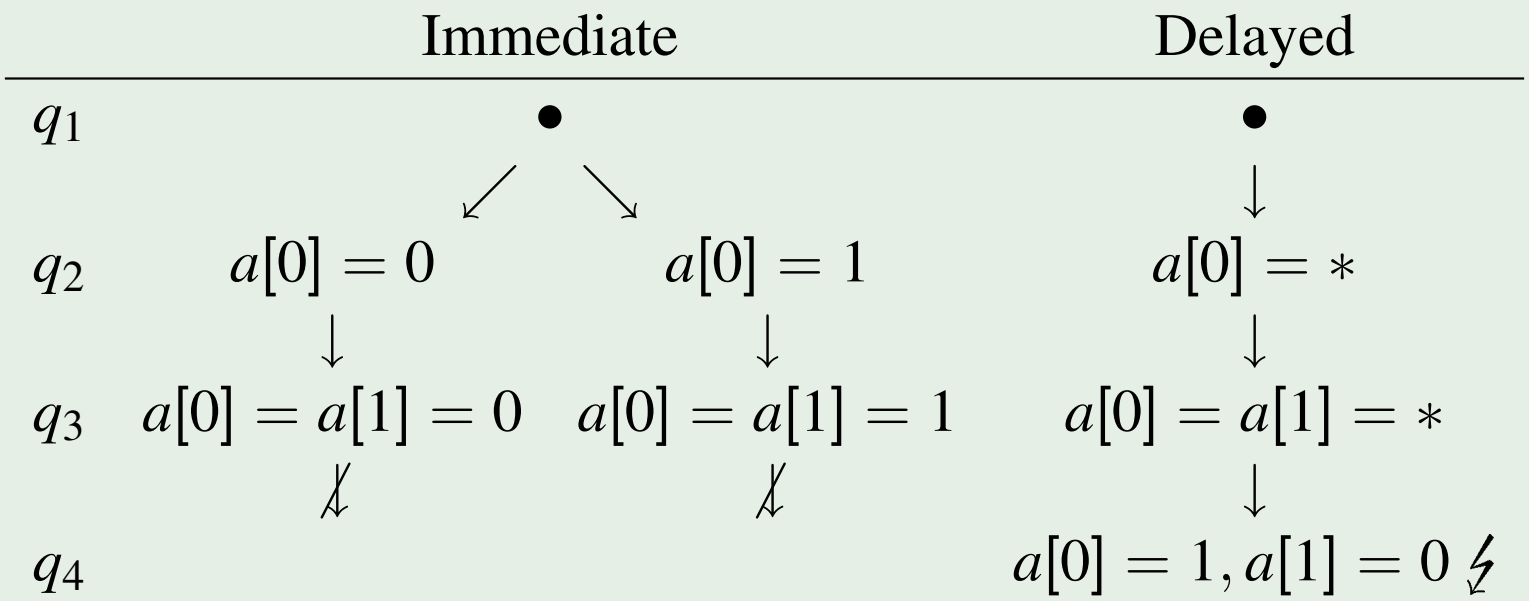
$\Longrightarrow$ false negatives possible

Incompleteness of Abstraction

Observation: over-approximation due to copying of nondeterminism

Example

$$
\begin{array}{lll}
q_1: & a[0] := * & : q_2 \\
q_2: & a[1] := a[0] & : q_3 \\
q_3: a[0] \wedge \neg a[1] \rightarrow & & : q_4
\end{array}
$$



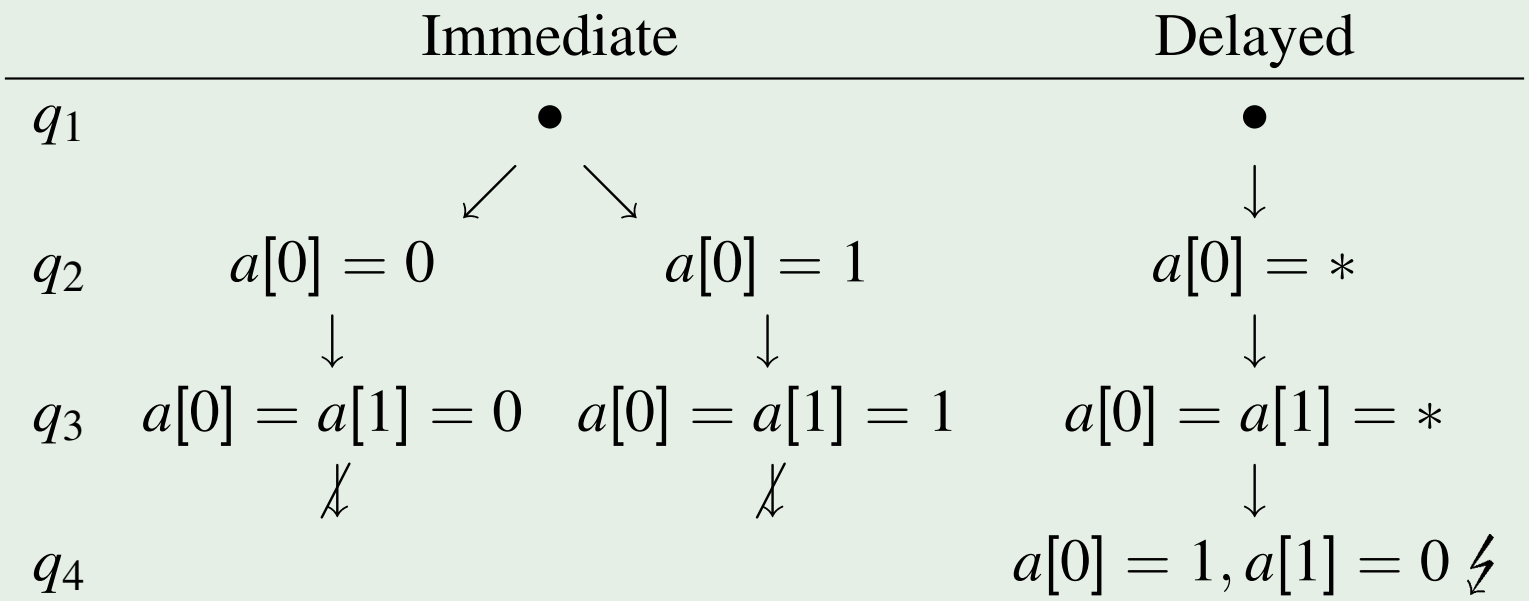
$\Longrightarrow$ false negatives possible

Incompleteness of Abstraction

Observation: over-approximation due to copying of nondeterminism

Example

$$
\begin{array}{lll}
q_1\colon & & a[0] := * \;\; : q_2 \\
q_2\colon & & a[1] := a[0] : q_3 \\
q_3\colon a[0] \wedge \neg a[1] \rightarrow & & \;\; : q_4
\end{array}
$$



$\Longrightarrow$ false negatives possible