

Data Transformation

user's variables u

implementer's variables old

Data Transformation

user's variables u

implementer's variables old

new implementer's variables new

Data Transformation

user's variables u

implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new. \exists old. D$

Data Transformation

user's variables u

implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new \cdot \exists old \cdot D$

specification S is transformed to $\forall old \cdot D \Rightarrow \exists old' \cdot D' \wedge S$

Data Transformation

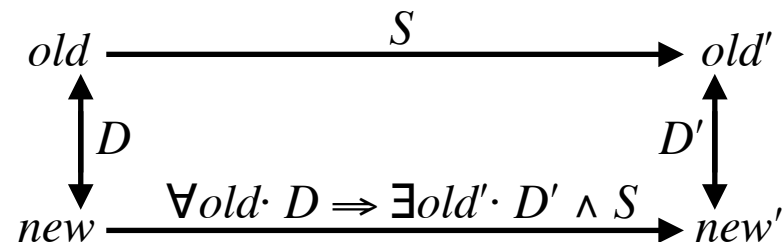
user's variables u

implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new. \exists old. D$

specification S is transformed to $\forall old. D \Rightarrow \exists old'. D' \wedge S$



Data Transformation

user's variables u

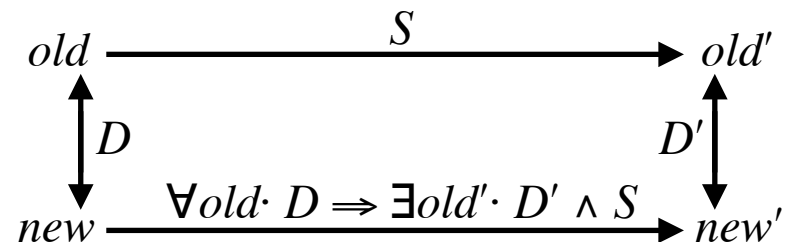
implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new \cdot \exists old \cdot D$



specification S is transformed to $\forall old \cdot D \Rightarrow \exists old' \cdot D' \wedge S$



Data Transformation

user's variables u

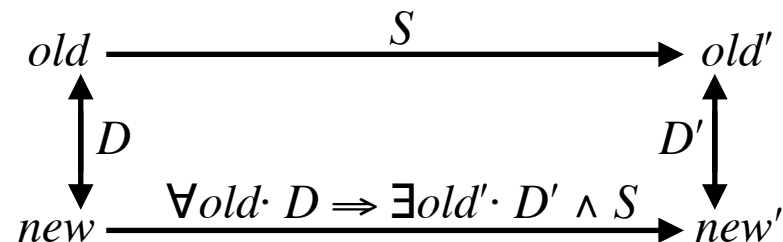
implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new \cdot \exists old \cdot D$



specification S is transformed to $\forall old \cdot D \Rightarrow \exists old' \cdot D' \wedge S$



Data Transformation

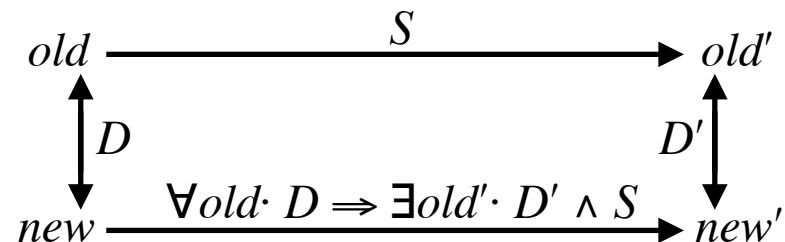
user's variables u

implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new. \exists old. D$

specification S is transformed to $\forall old. D \Rightarrow \exists old'. D' \wedge S$ ←



Data Transformation

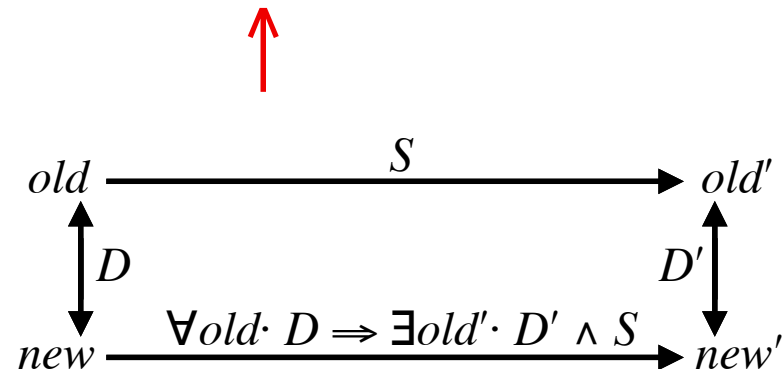
user's variables u

implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new \cdot \exists old \cdot D$

specification S is transformed to $\forall old \cdot D \Rightarrow \exists old' \cdot D' \wedge S$



Data Transformation

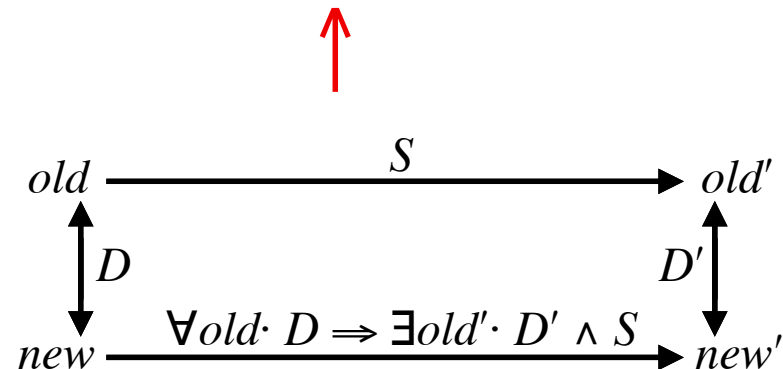
user's variables u

implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new \cdot \exists old \cdot D$

specification S is transformed to $\forall old \cdot D \Rightarrow \exists old' \cdot D' \wedge S$



Data Transformation

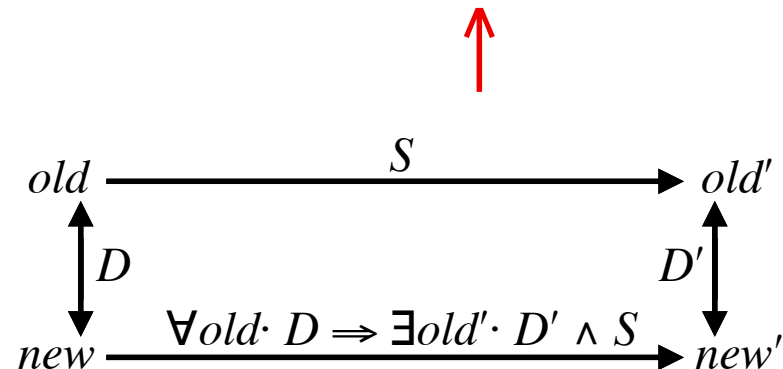
user's variables u

implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new. \exists old. D$

specification S is transformed to $\forall old. D \Rightarrow \exists old'. D' \wedge S$



Data Transformation

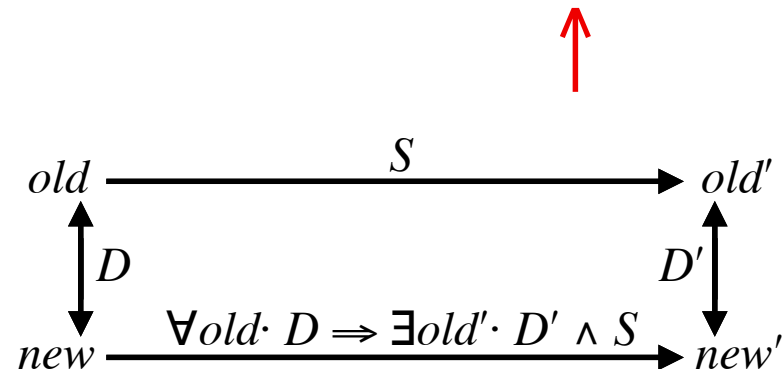
user's variables u

implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new \cdot \exists old \cdot D$

specification S is transformed to $\forall old \cdot D \Rightarrow \exists old' \cdot D' \wedge S$



Data Transformation

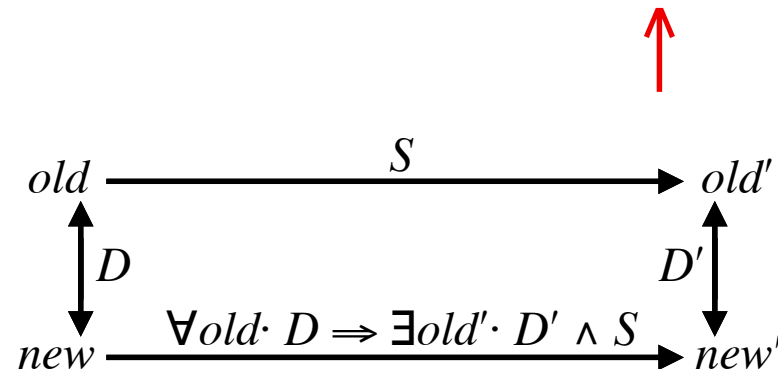
user's variables u

implementer's variables old

new implementer's variables new

data transformer D relates old and new such that $\forall new \cdot \exists old \cdot D$

specification S is transformed to $\forall old \cdot D \Rightarrow \exists old' \cdot D' \wedge S$



Data Transformation

example

user's variable u : *bin*

implementer's variable v : *nat*

Data Transformation

example

user's variable u : *bin*

implementer's variable v : *nat*

operations

$zero = v := 0$

$increase = v := v + 1$

$inquire = u := \text{even } v$

Data Transformation

example

user's variable $u: bin$

implementer's variable $v: nat$

operations

$zero = v := 0$

$increase = v := v + 1$

$inquire = u := even\ v$

new implementer's variable $w: bin$

data transformer $w = even\ v$

Data Transformation

$$\forall v \cdot D \Rightarrow \exists v' \cdot D' \wedge \text{zero}$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{zero}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := 0)$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{zero}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := 0)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = u \wedge v' = 0$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{zero}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := 0)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = u \wedge v' = 0$$



1-pt

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{zero}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := 0)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = u \wedge v' = 0$$

1-pt

$$= \forall v. w = \text{even } v \Rightarrow w' = \text{even } 0 \wedge u' = u$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{zero}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := 0)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = u \wedge v' = 0$$

1-pt

$$= \forall v. w = \text{even } v \Rightarrow w' = \text{even } 0 \wedge u' = u$$

change variable

$$\forall v: \text{domain}. (\text{substitute } f v \text{ for } r \text{ in } b)$$

$$= \forall r: f \text{ domain}. b$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{zero}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := 0)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = u \wedge v' = 0$$

1-pt

$$= \forall v. w = \text{even } v \Rightarrow w' = \text{even } 0 \wedge u' = u$$

change variable

$$= \forall r: \text{even nat}. w = r \Rightarrow w' = \top \wedge u' = u$$

$$\forall v: \text{domain}. (\text{substitute } f v \text{ for } r \text{ in } b)$$

$$= \forall r: f \text{ domain}. b$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{zero}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := 0)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = u \wedge v' = 0$$

1-pt

$$= \forall v. w = \text{even } v \Rightarrow w' = \text{even } 0 \wedge u' = u$$

change variable

$$= \forall r: \text{even nat}. w = r \Rightarrow w' = \top \wedge u' = u$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{zero}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := 0)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = u \wedge v' = 0 \quad \text{1-pt}$$

$$= \forall v. w = \text{even } v \Rightarrow w' = \text{even } 0 \wedge u' = u \quad \text{change variable}$$

$$= \forall r: \text{even nat}. w = r \Rightarrow w' = \top \wedge u' = u \quad \text{1-pt}$$

$$= w' = \top \wedge u' = u$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{zero}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := 0)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = u \wedge v' = 0 \quad \text{1-pt}$$

$$= \forall v. w = \text{even } v \Rightarrow w' = \text{even } 0 \wedge u' = u \quad \text{change variable}$$

$$= \forall r: \text{even nat}. w = r \Rightarrow w' = \top \wedge u' = u \quad \text{1-pt}$$

$$= w' = \top \wedge u' = u$$

$$= w := \top$$

Data Transformation

$$\forall v \cdot D \Rightarrow \exists v' \cdot D' \wedge \textit{increase}$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \textit{increase}$$

$$= \forall v. w = \textit{even } v \Rightarrow \exists v'. w' = \textit{even } v' \wedge (v := v+1)$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \textit{increase}$$

$$= \forall v. w = \textit{even } v \Rightarrow \exists v'. w' = \textit{even } v' \wedge (v := v+1)$$

$$= \forall v. w = \textit{even } v \Rightarrow \exists v'. w' = \textit{even } v' \wedge u'=u \wedge v'=v+1$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \textit{increase}$$

$$= \forall v. w = \textit{even } v \Rightarrow \exists v'. w' = \textit{even } v' \wedge (v := v+1)$$

$$= \forall v. w = \textit{even } v \Rightarrow \exists v'. w' = \textit{even } v' \wedge u'=u \wedge v'=v+1$$

1-pt

$$= \forall v. w = \textit{even } v \Rightarrow w' = \textit{even } (v+1) \wedge u'=u$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \textit{increase}$$

$$= \forall v. w = \textit{even } v \Rightarrow \exists v'. w' = \textit{even } v' \wedge (v := v+1)$$

$$= \forall v. w = \textit{even } v \Rightarrow \exists v'. w' = \textit{even } v' \wedge u'=u \wedge v'=v+1$$

1-pt

$$= \forall v. w = \textit{even } v \Rightarrow w' = \textit{even } (v+1) \wedge u'=u$$

change var

$$= \forall r: \textit{even nat}. w=r \Rightarrow w' = \neg r \wedge u'=u$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{increase}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := v+1)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u'=u \wedge v'=v+1 \quad \text{1-pt}$$

$$= \forall v. w = \text{even } v \Rightarrow w' = \text{even } (v+1) \wedge u'=u \quad \text{change var}$$

$$= \forall r: \text{even nat}. w=r \Rightarrow w' = \neg r \wedge u'=u \quad \text{1-pt}$$

$$= w' = \neg w \wedge u'=u$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{increase}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (v := v+1)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = u \wedge v' = v+1 \quad \text{1-pt}$$

$$= \forall v. w = \text{even } v \Rightarrow w' = \text{even } (v+1) \wedge u' = u \quad \text{change var}$$

$$= \forall r: \text{even nat}. w = r \Rightarrow w' = \neg r \wedge u' = u \quad \text{1-pt}$$

$$= w' = \neg w \wedge u' = u$$

$$= w := \neg w$$

Data Transformation

$$\forall v \cdot D \Rightarrow \exists v' \cdot D' \wedge \textit{inquire}$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{inquire}$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge (u := \text{even } v)$$

$$= \forall v. w = \text{even } v \Rightarrow \exists v'. w' = \text{even } v' \wedge u' = \text{even } v \wedge v' = v \quad \text{1-pt}$$

$$= \forall v. w = \text{even } v \Rightarrow w' = \text{even } v \wedge u' = \text{even } v \quad \text{change var}$$

$$= \forall r: \text{even nat}. w=r \Rightarrow w'=r \wedge u'=r \quad \text{1-pt}$$

$$= w'=w \wedge u'=w$$

$$= u := w$$

Data Transformation

example

user's variable $u: bin$

implementer's variable $v: bin$

operations

$set = v := \top$

$flip = v := \neg v$

$ask = u := v$

new implementer's variable $w: nat$

data transformer $v = even\ w$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{set}$$

$$= \forall v. v = \text{even } w \Rightarrow \exists v'. v' = \text{even } w' \wedge (v := \top)$$

$$= \text{even } w' \wedge u' = u$$

$$\Leftarrow w := 0$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{set}$$

$$= \forall v. v = \text{even } w \Rightarrow \exists v'. v' = \text{even } w' \wedge (v := \top)$$

$$= \text{even } w' \wedge u' = u \quad \leftarrow$$

$$\leftarrow w := 0$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{set}$$

$$= \forall v. v = \text{even } w \Rightarrow \exists v'. v' = \text{even } w' \wedge (v := \top)$$

$$= \text{even } w' \wedge u' = u$$

$$\Leftarrow w := 0 \quad \leftarrow$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge \text{flip}$$

$$= \forall v. v = \text{even } w \Rightarrow \exists v'. v' = \text{even } w' \wedge (v := \neg v)$$

$$= \text{even } w' \neq \text{even } w \wedge u' = u$$

$$\Leftarrow w := w + 1$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge ask$$

$$= \forall v. v = even\ w \Rightarrow \exists v'. v' = even\ w' \wedge (u := v)$$

$$= even\ w' = even\ w = u'$$

$$\Leftarrow u := even\ w$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge ask$$

$$= \forall v. v = even\ w \Rightarrow \exists v'. v' = even\ w' \wedge (u := v)$$

$$= even\ w' = even\ w = u' \quad \leftarrow$$

$$\Leftarrow u := even\ w$$

Data Transformation

$$\forall v. D \Rightarrow \exists v'. D' \wedge ask$$

$$= \forall v. v = even\ w \Rightarrow \exists v'. v' = even\ w' \wedge (u := v)$$

$$= even\ w' = even\ w = u'$$

$$\Leftarrow u := even\ w \quad \leftarrow$$

Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

binary implementer's variables

A records the state of input a at last output change

B records the state of input b at last output change

Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**



Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**



Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**



Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**



Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**



Security Switch

A security switch has three binary user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first user changes back before the second user changes, the output does not change.

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

$b := \neg b$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

Security Switch

replace old implementer's variables A and B with nothing!

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

$b := \neg b$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

Security Switch

replace old implementer's variables A and B with nothing!

data transformer

$$A=B=c$$

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

$b := \neg b$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

Security Switch

replace old implementer's variables A and B with nothing!

data transformer

$$A=B=c$$

proof

$$\forall new. \exists old. \text{transformer}$$

operations

$a := \neg a.$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c.$ $A := a.$ $B := b$ **else ok fi**

$b := \neg b.$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c.$ $A := a.$ $B := b$ **else ok fi**

Security Switch

replace old implementer's variables A and B with nothing!

data transformer

$$A=B=c$$

proof

$$\exists A, B. A=B=c$$

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

$b := \neg b$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

Security Switch

replace old implementer's variables A and B with nothing!

data transformer

$$A=B=c$$

proof

$$\exists A, B. A=B=c$$

generalization, using c for both A and B

$\Leftarrow \top$

operations

$a := \neg a. \text{ if } a \neq A \wedge b \neq B \text{ then } c := \neg c. A := a. B := b \text{ else ok fi}$

$b := \neg b. \text{ if } a \neq A \wedge b \neq B \text{ then } c := \neg c. A := a. B := b \text{ else ok fi}$

Security Switch

operations

$a := \neg a$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

$b := \neg b$. **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c$. $A := a$. $B := b$ **else ok fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

operations

$a := \neg a.$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$ **else ok fi**

$b := \neg b.$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$ **else ok fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else *ok* **fi**

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
then $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else *ok* **fi**

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
 \uparrow \uparrow **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
 \uparrow \uparrow **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
then $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**

use one-point law for A and B , and for A' and B'

= **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=a \wedge c'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
then $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**



use one-point law for A and B , and for A' and B'

= **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=a \wedge c'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
then $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**



use one-point law for A and B , and for A' and B'

= **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=a \wedge c'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**



Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
then $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**



use one-point law for A and B , and for A' and B'

= **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=a \wedge c'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
then $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**



use one-point law for A and B , and for A' and B'

= **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=a \wedge c'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**



Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
then $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**

use one-point law for A and B , and for A' and B'

= **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=a \wedge c'=b$ use context
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**

= **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=\neg c \wedge c'=\neg c$
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

expand assignments, sequential compositions, and *ok*

= $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
then $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**

use one-point law for A and B , and for A' and B'

= **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=a \wedge c'=b$ use context
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**

= **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=\neg c \wedge c'=\neg c$
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**

= **if** $a \neq c \wedge b \neq c$ **then** $c := \neg c$ **else ok fi**

Security Switch

$\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$ **then** $c := \neg c. A := a. B := b$
else ok fi

expand assignments, sequential compositions, and *ok*

$=$ $\forall A, B. A=B=c \Rightarrow \exists A', B'. A'=B'=c' \wedge$ **if** $a \neq A \wedge b \neq B$
then $a'=a \wedge b'=b \wedge c'=\neg c \wedge A'=a \wedge B'=b$
else $a'=a \wedge b'=b \wedge c'=c \wedge A'=A \wedge B'=B$ **fi**

use one-point law for A and B , and for A' and B'

$=$ **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=a \wedge c'=b$ use context
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**

$=$ **if** $a \neq c \wedge b \neq c$ **then** $a'=a \wedge b'=b \wedge c'=\neg c \wedge c'=\neg c \wedge c'=\neg c$
else $a'=a \wedge b'=b \wedge c'=c \wedge c'=c \wedge c'=c$ **fi**

$=$ **if** $a \neq c \wedge b \neq c$ **then** $c := \neg c$ **else ok fi**

$=$ $c := (a \neq c \wedge b \neq c) \neq c$