

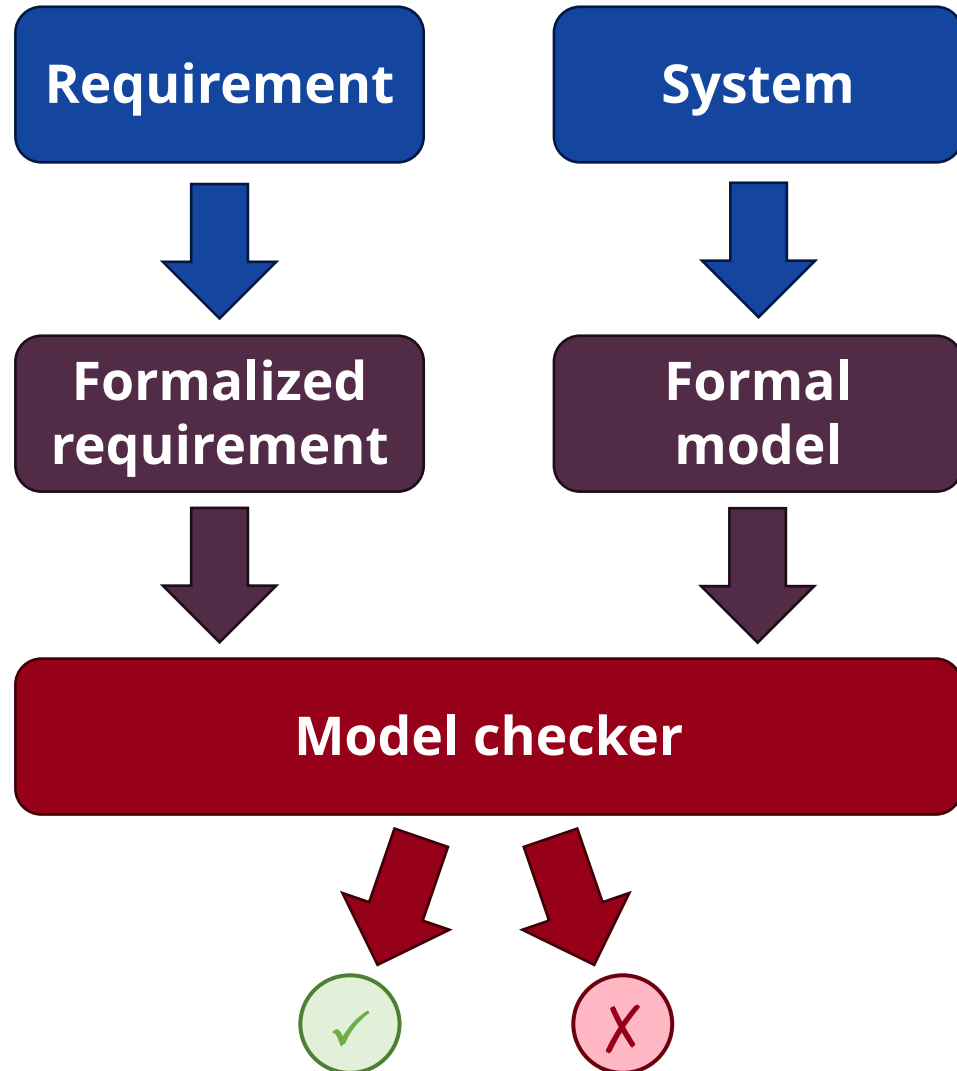
Abstraction-based model checking for real-time software-intensive system models

Dóra Cziborová

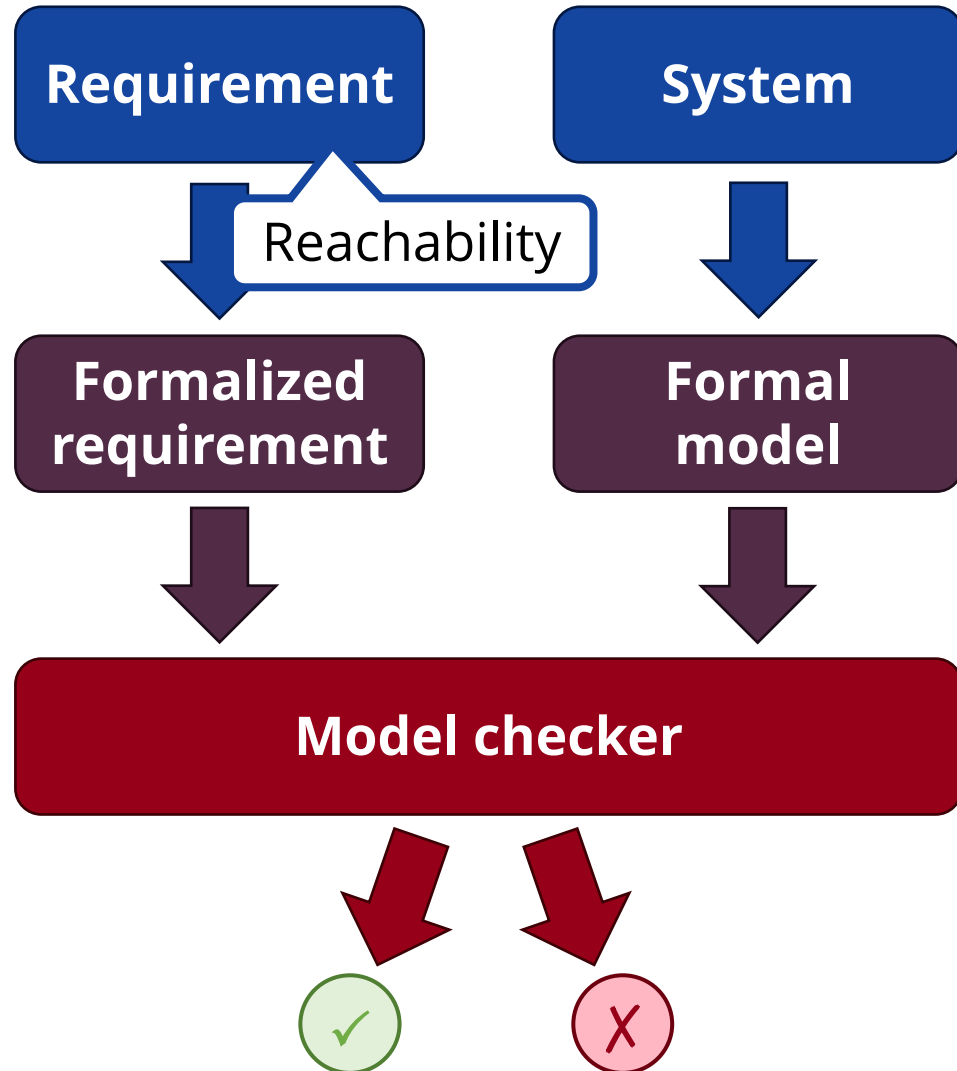


**Critical Systems
Research Group**

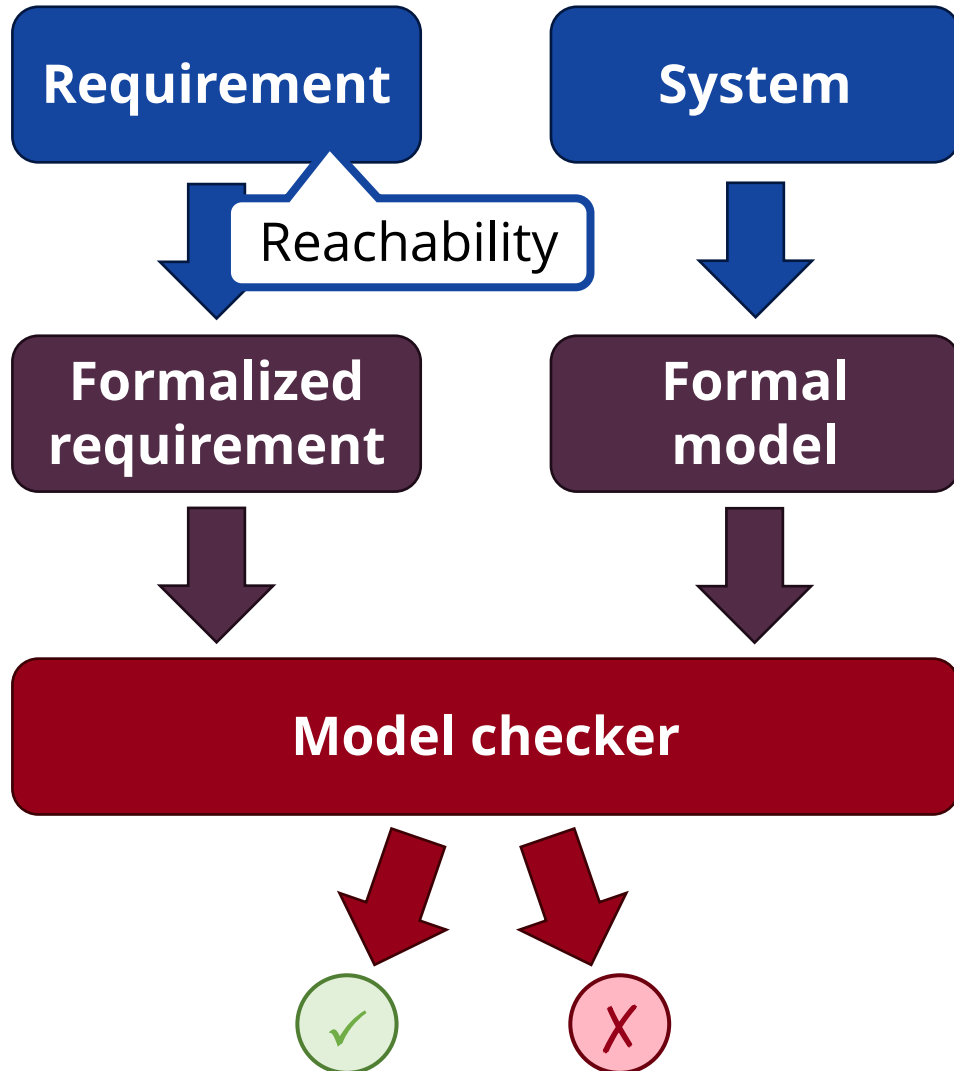
Model checking



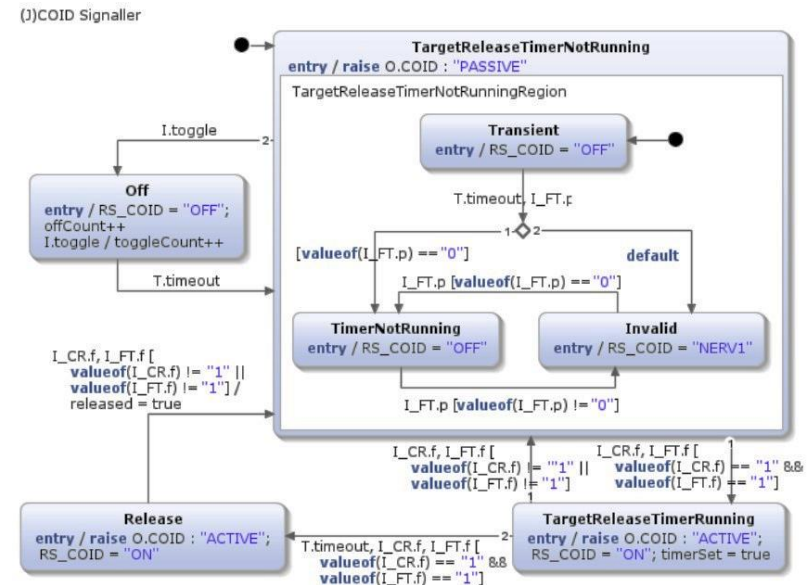
Model checking



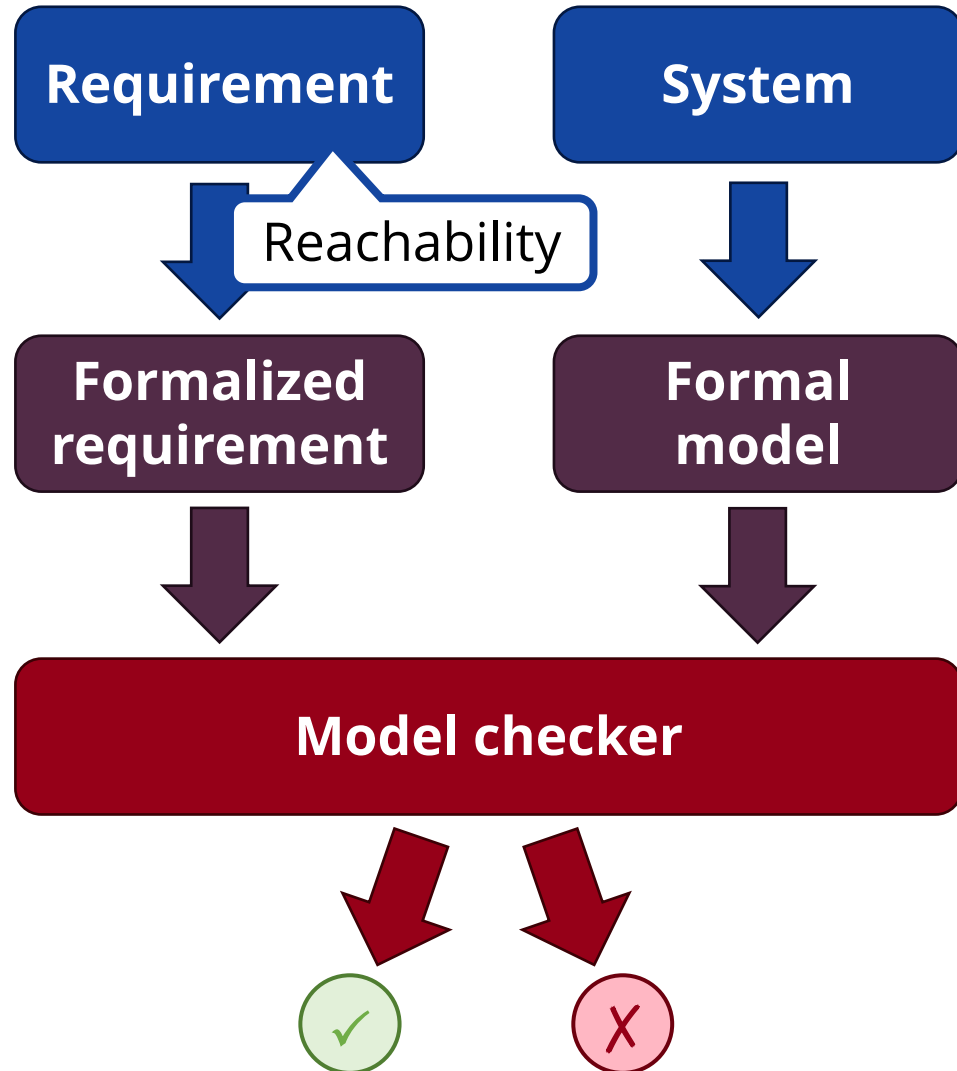
Model checking - system models



- Focus: **real-time software-intensive** systems
- State-based representation, e.g. **statecharts**

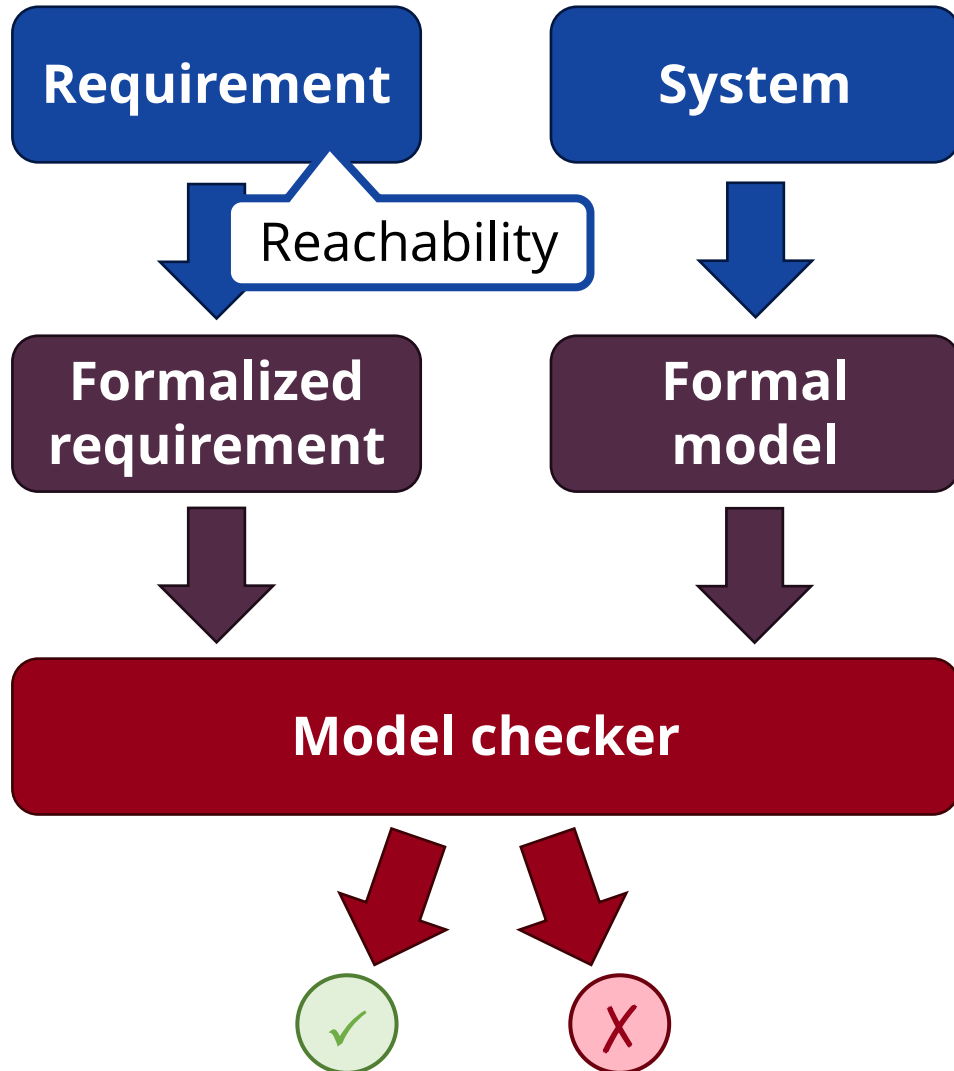


Model checking - formal models



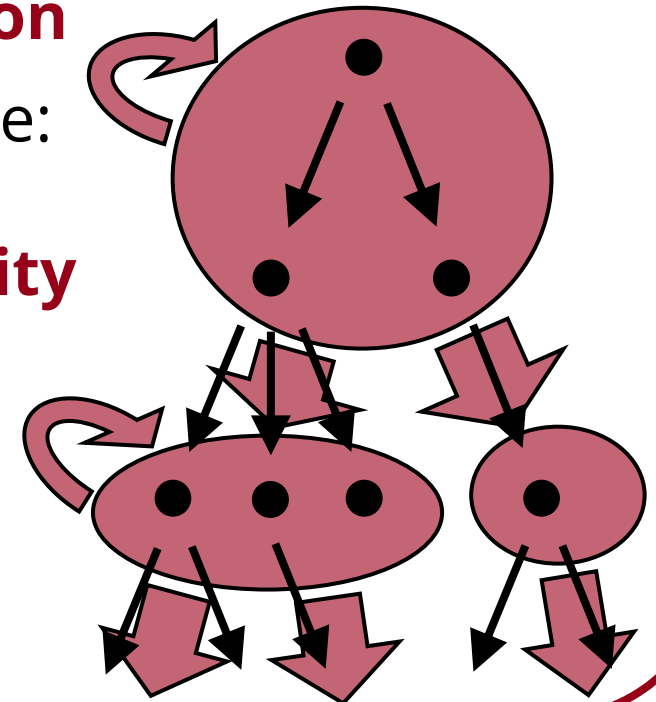
- **Intermediate formalisms:**
 - **High-level** language constructs
 - More **expressive** than low-level formal models
 - Easier mapping from system models
- The **XSTS** formalism – e**X**tended **S**ymbolic **T**ransition **S**ystem

Model checking - abstraction



- # of data variables
 - Continuous time
- } state space explosion

- **Abstraction**
- State space: **abstract reachability graph (ARG)**



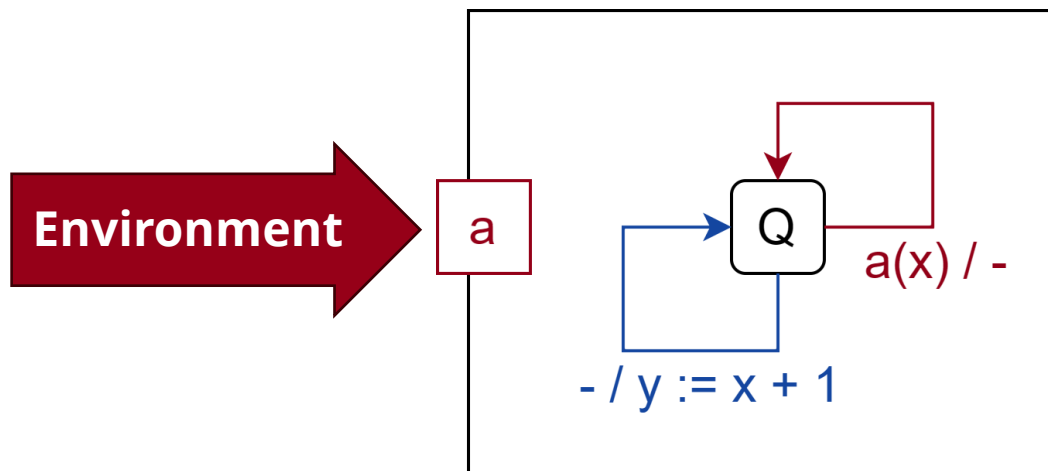
Steps to verify timed software-intensive models

- I. An **intermediate formalism** is required
 - Existing formalism: timed automata
 - Extending the **XSTS** formalism by **timing**

- II. Supporting the **verification of timed XSTS models**
 - Usual challenges of timed verification
 - Challenges specific to the timed XSTS formalism

The XSTS formalism

Simple statechart model



XSTS representation

```
type State : {Q}
ctrl var state : State = Q
var x : integer = 0
var y : integer = 0
```

```
trans {
  if (state == Q) {
    choice {
      havoc x;
    } or {
      y := x + 1;
    }
  }
}
```


XSTS language constructs

XSTS language constructs

Assumption

```
assume y > x;
```

XSTS language constructs

Assumption

```
assume y > x;
```

Assignment

```
y := x + 1;
```

XSTS language constructs

Assumption

`assume y > x;`

Assignment

`y := x + 1;`

**Non-deterministic
assignment**

`havoc x;`

XSTS language constructs

Assumption

```
assume y > x;
```

Assignment

```
y := x + 1;
```

Non-deterministic assignment

```
havoc x;
```

Conditional operation

```
if (x > 0) {  
    ...  
} else {  
    ...  
}
```

XSTS language constructs

Assumption

```
assume y > x;
```

Assignment

```
y := x + 1;
```

Non-deterministic assignment

```
havoc x;
```

Conditional operation

```
if (x > 0) {  
    ...  
} else {  
    ...  
}
```

Non-deterministic operation

```
choice {  
    ...  
} or {  
    ...  
} or {  
    ...
```


XSTS language constructs

Assumption

```
assume y > x;
```

Assignment

```
y := x + 1;
```

Non-deterministic assignment

```
havoc x;
```

Conditional operation

```
if (x > 0) {  
    ...  
} else {  
    ...  
}
```

Non-deterministic operation

```
choice {  
    ...  
} or {  
    ...  
} or {  
    ...
```

Counting loop

```
for i from 0 to x do {  
    ...  
}
```

The TXSTS formalism - Timed XSTS

- XSTS extended by **clock variables** and **clock operations**

The TXSTS formalism - Timed XSTS

- XSTS extended by **clock variables** and **clock operations**

Clock set / reset

```
c := 0;
```

```
c := 500;
```

The TXSTS formalism - Timed XSTS

- XSTS extended by **clock variables** and **clock operations**

Clock set / reset

```
c := 0;
```

```
c := 500;
```

Clock constraints

```
assume c1 - c2 > 0;
```

```
if (c > 500 || ...) ...
```

The TXSTS formalism - Timed XSTS

- XSTS extended by **clock variables** and **clock operations**

Clock set / reset

```
c := 0;  
c := 500;
```

Clock constraints

```
assume c1 - c2 > 0;  
if (c > 500 || ...) ...
```

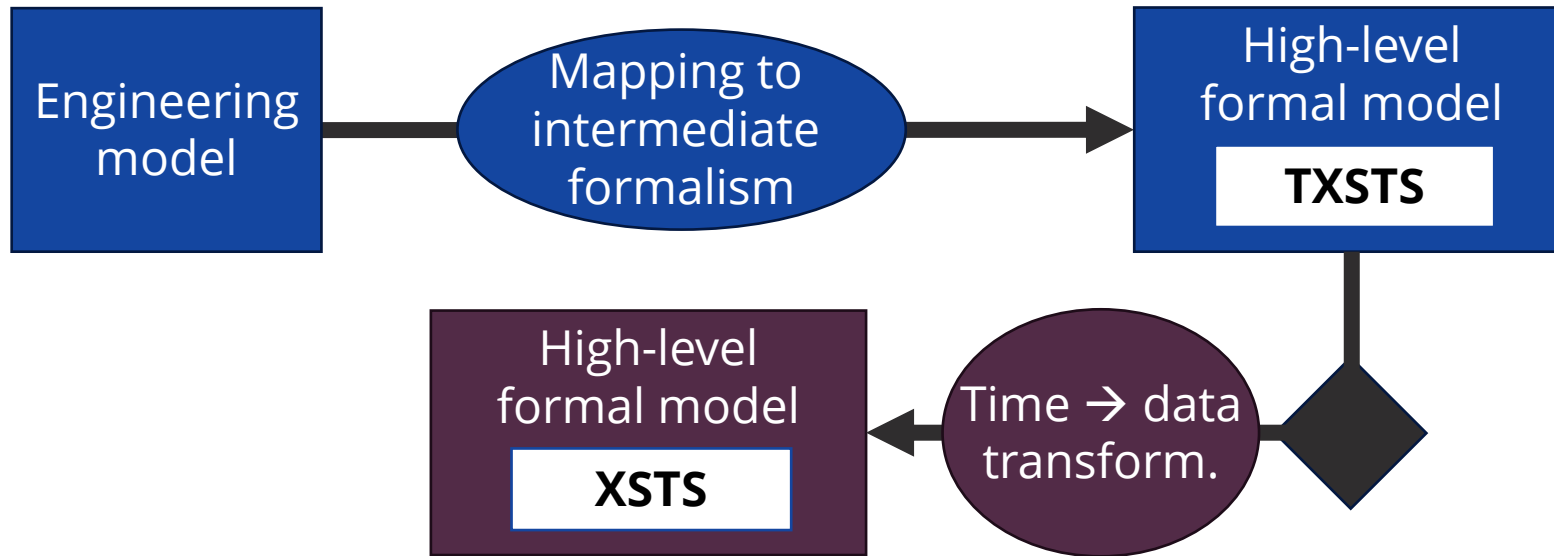
Increment all clocks

```
__delay;
```

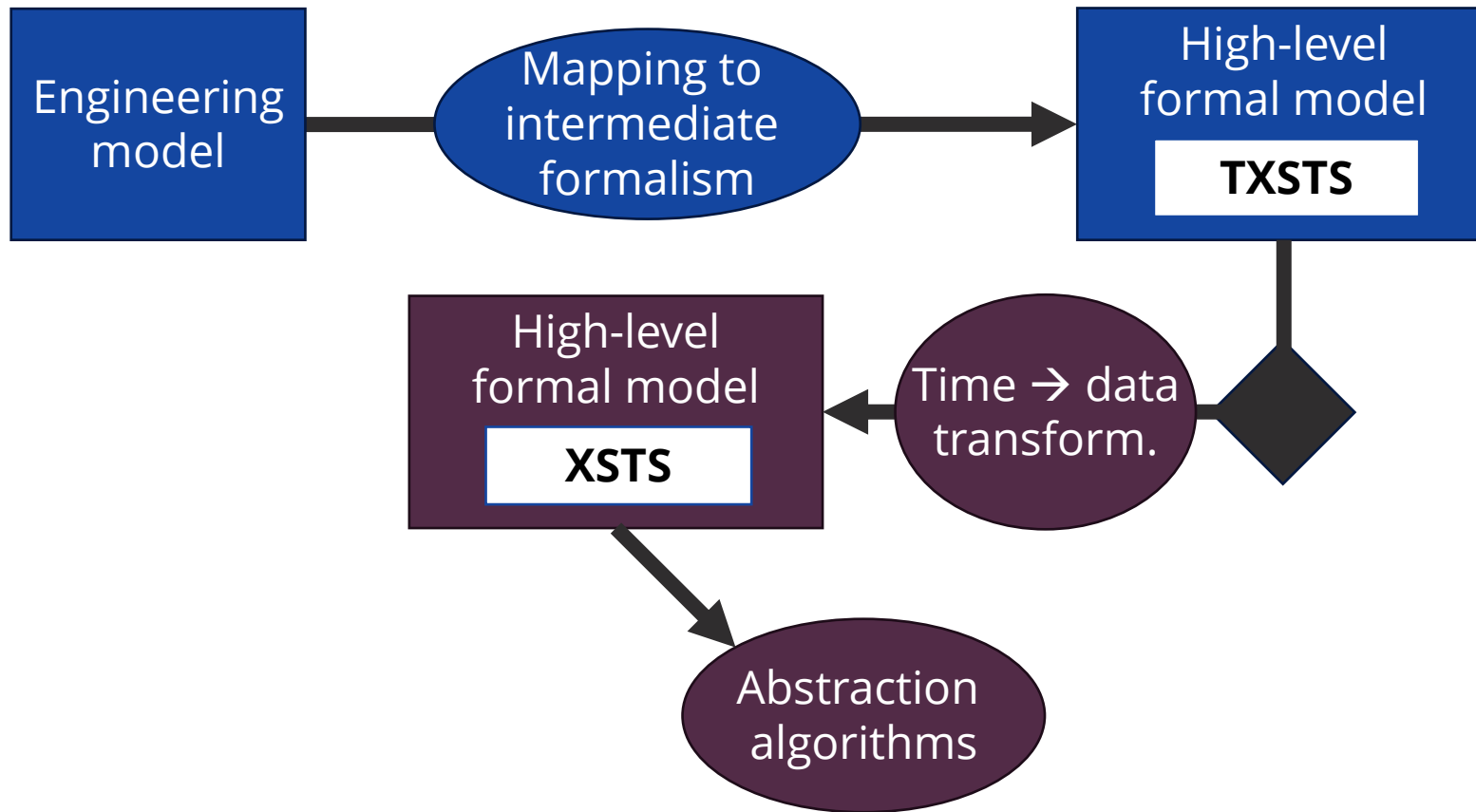
Verification approaches for TXSTS models



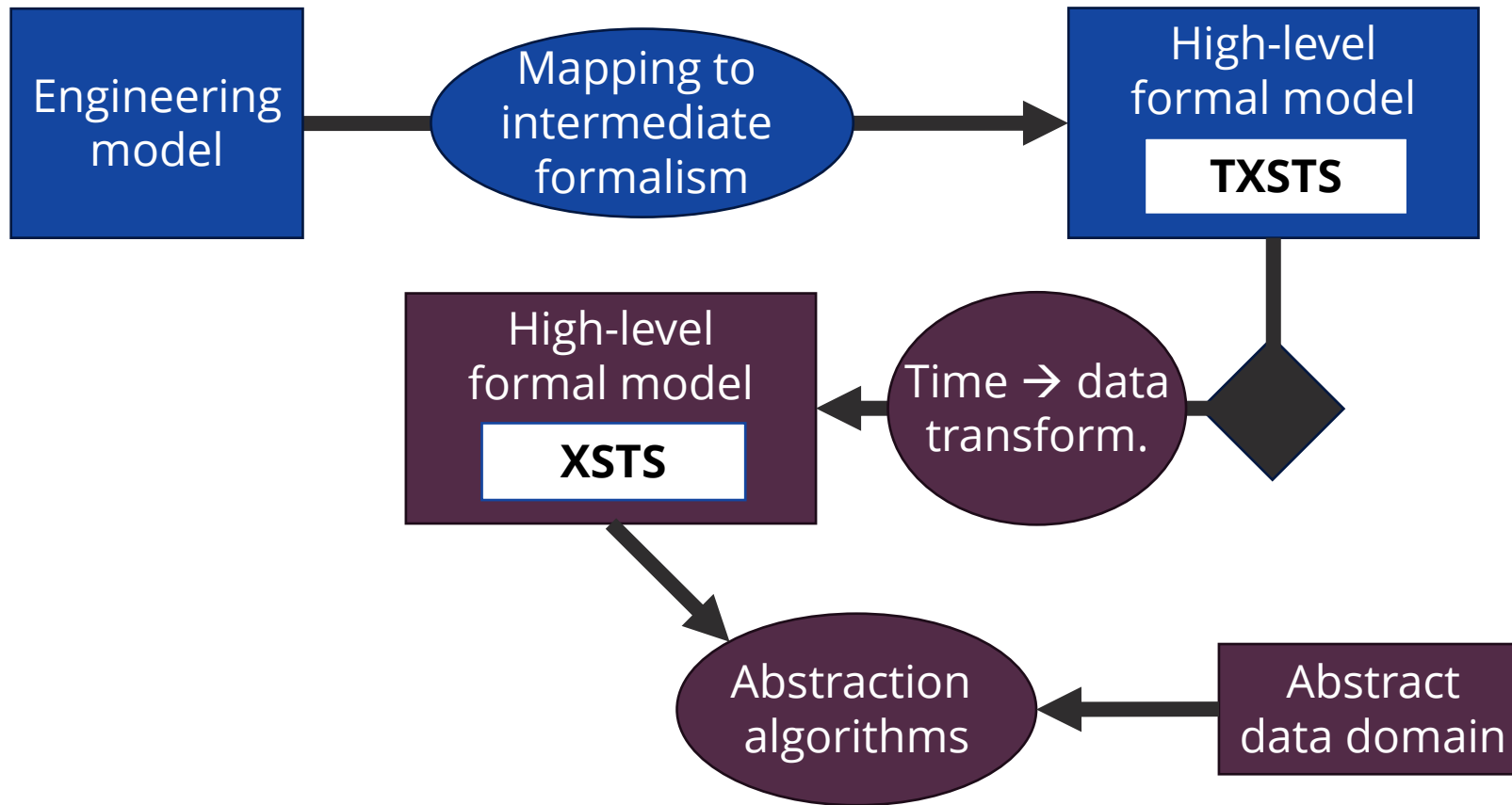
Verification approaches for TXSTS models



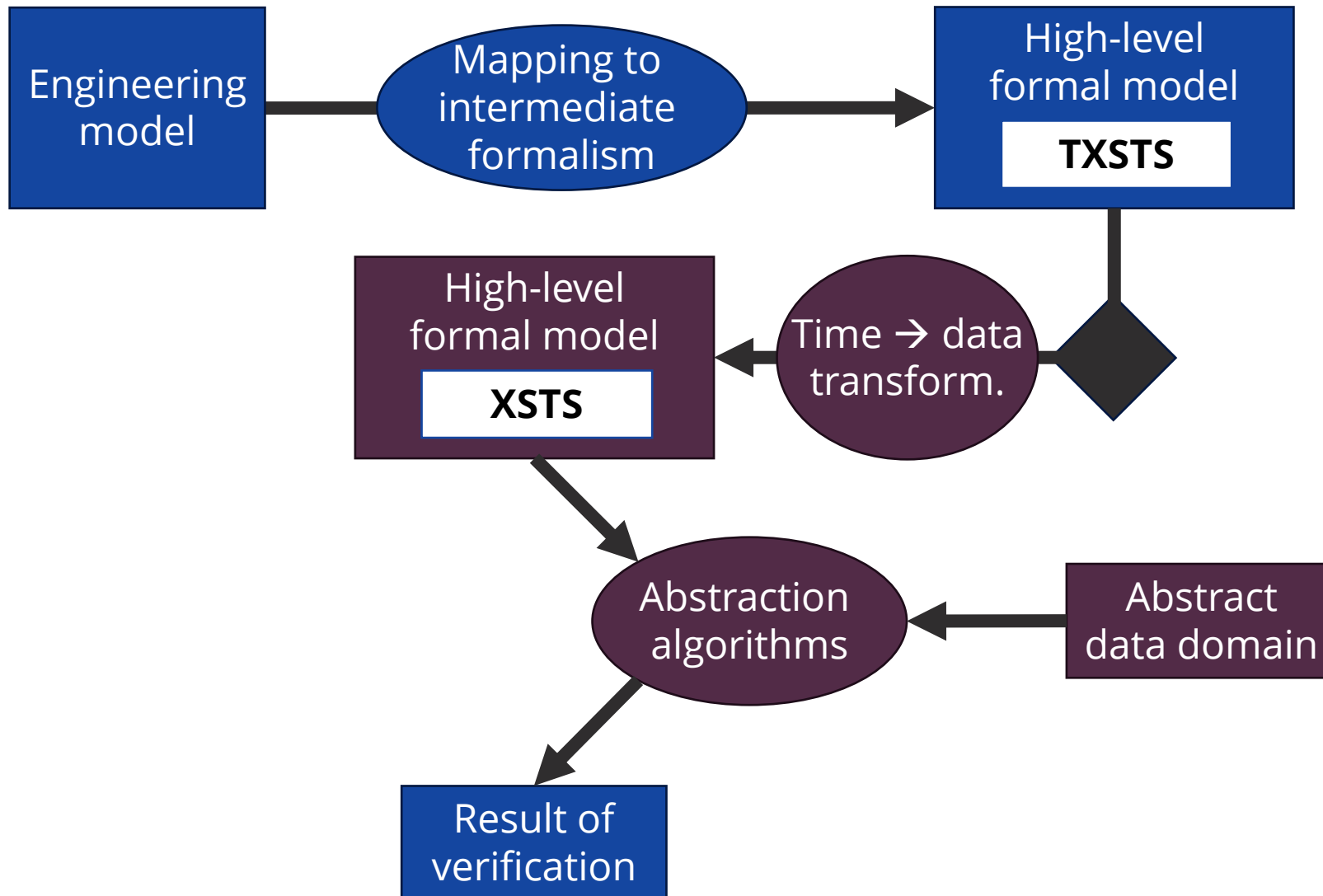
Verification approaches for TXSTS models



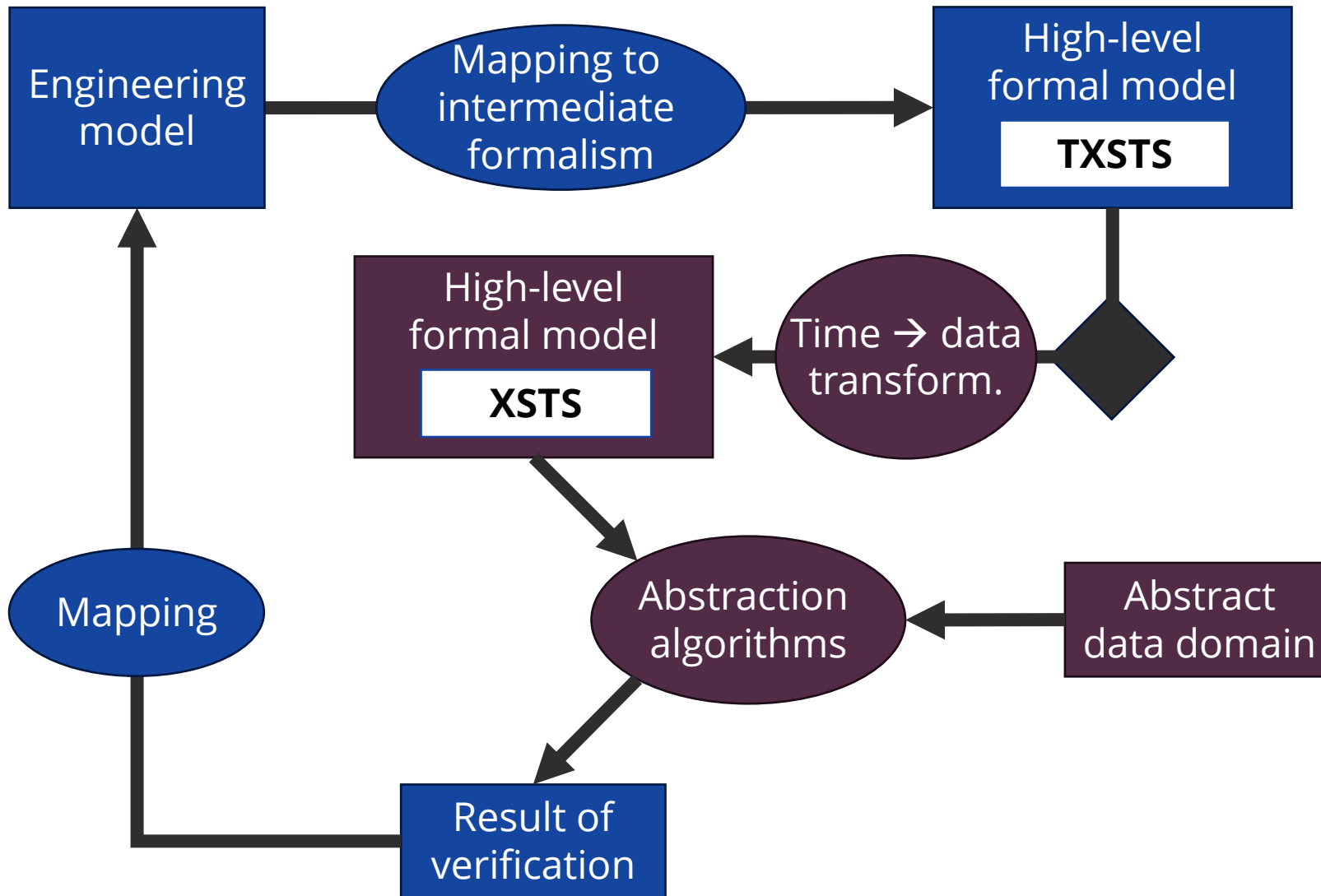
Verification approaches for TXSTS models



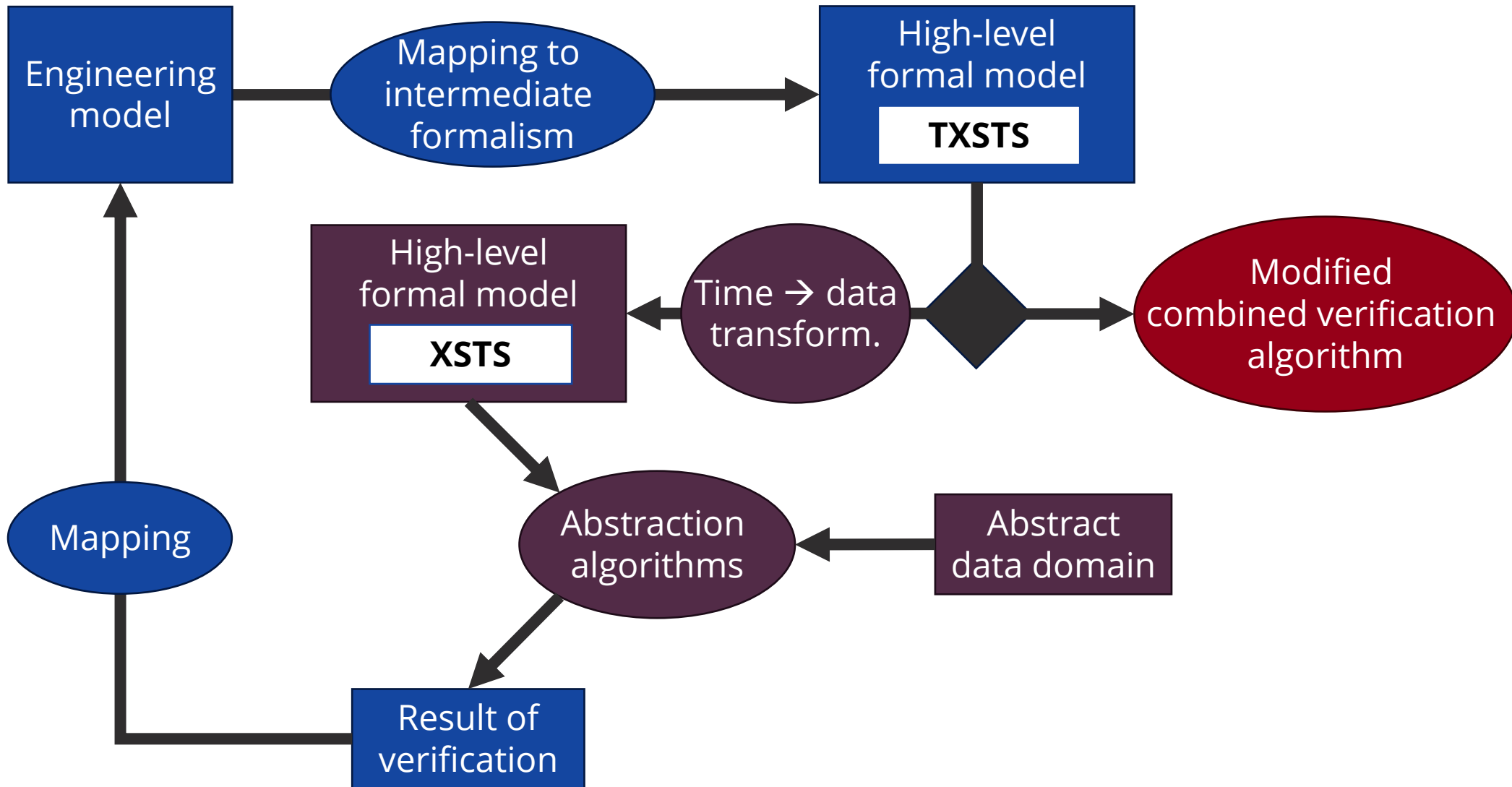
Verification approaches for TXSTS models



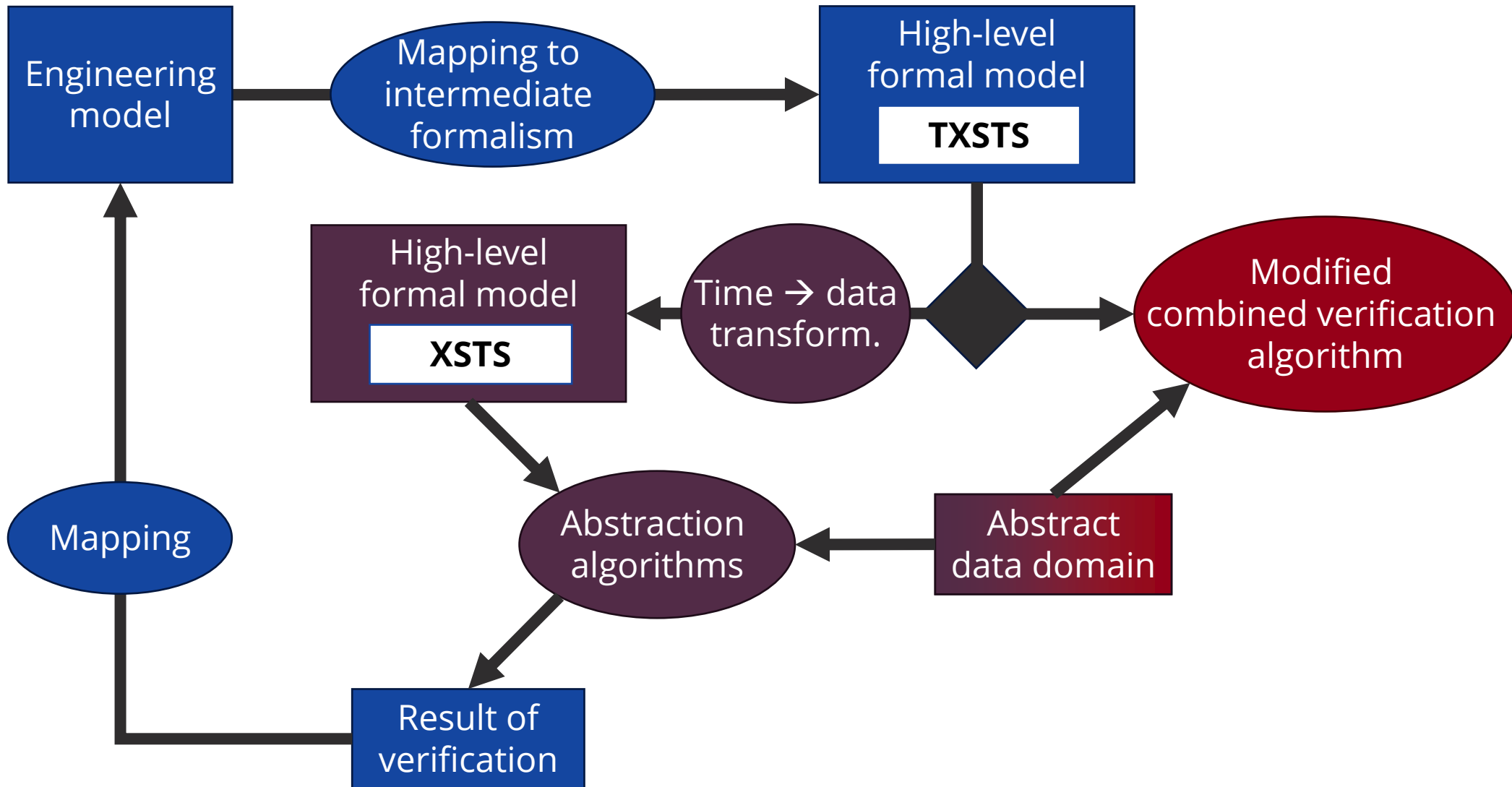
Verification approaches for TXSTS models



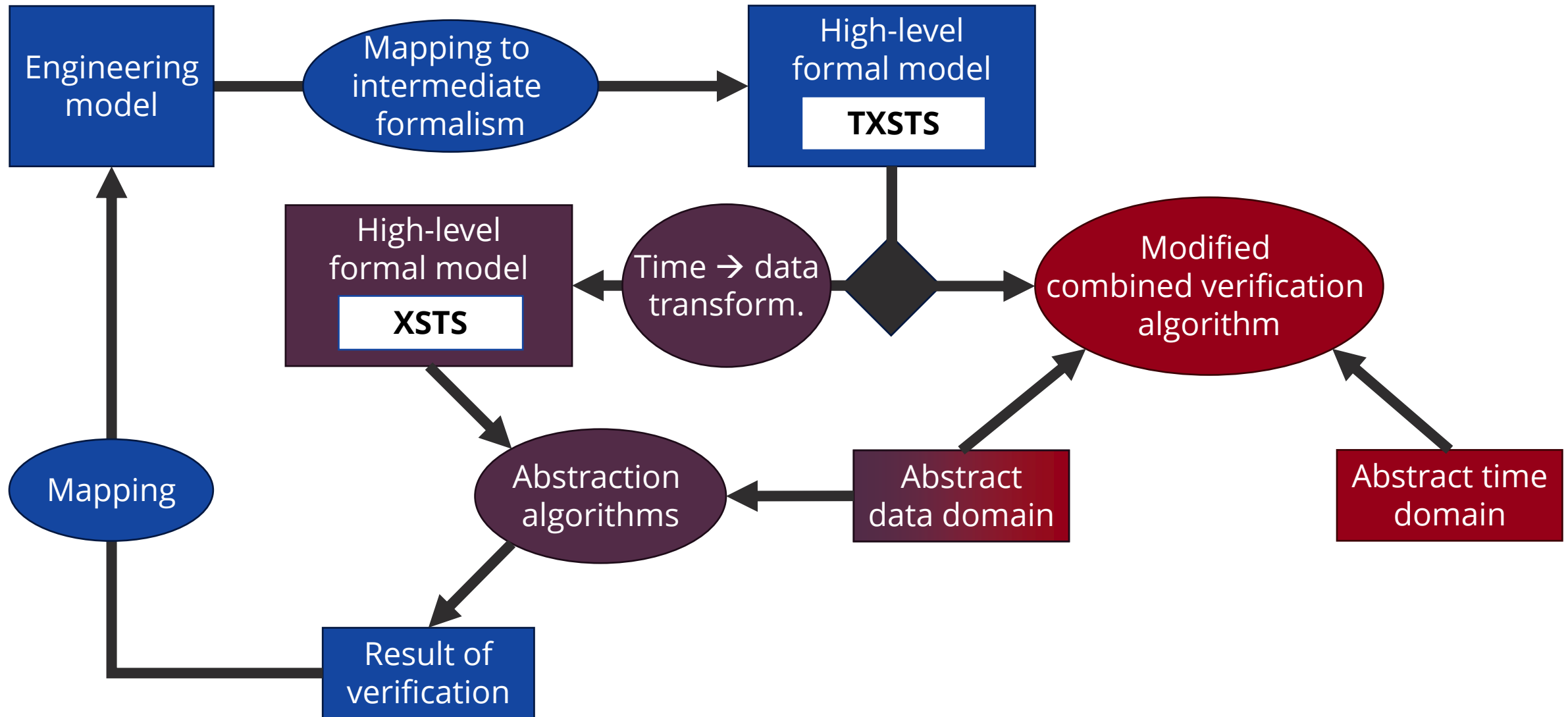
Verification approaches for TXSTS models



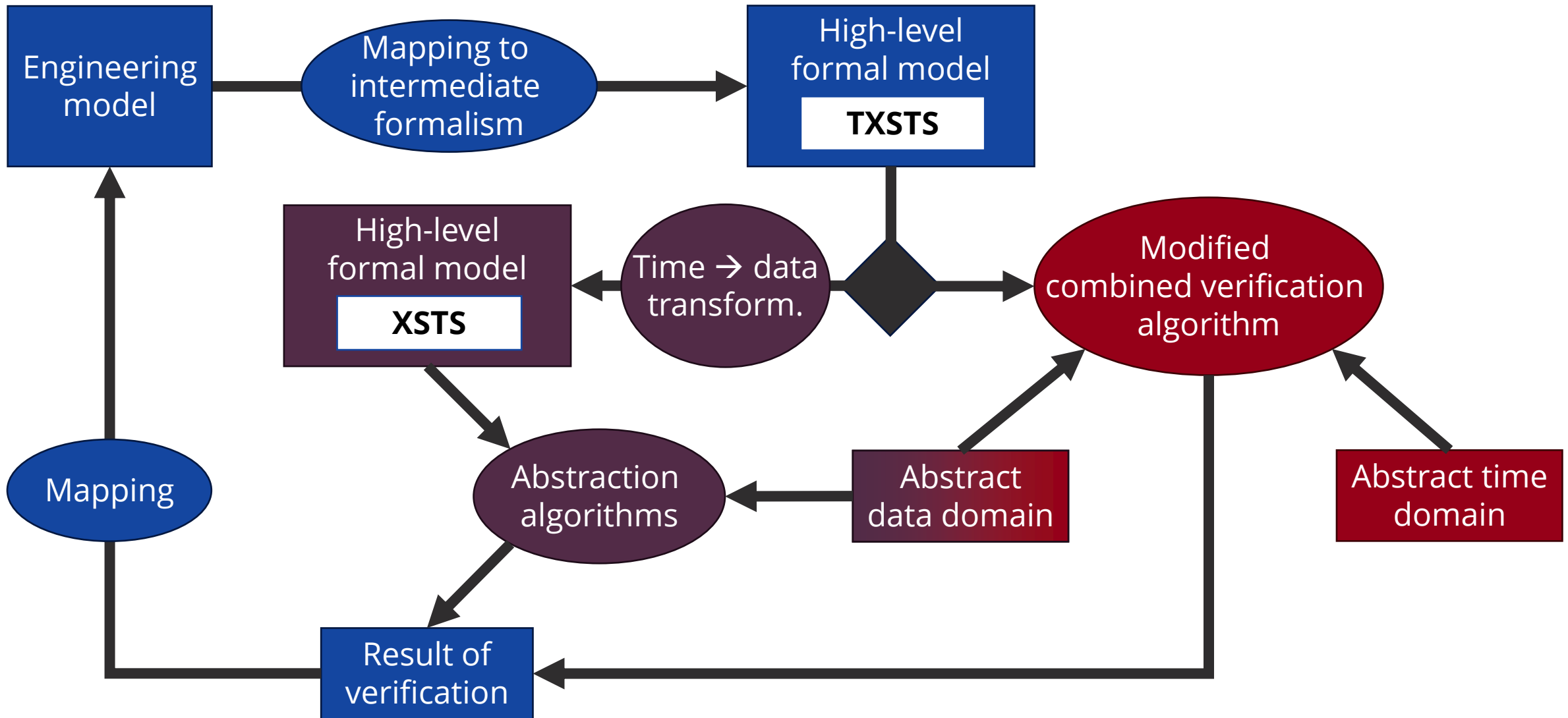
Verification approaches for TXSTS models



Verification approaches for TXSTS models

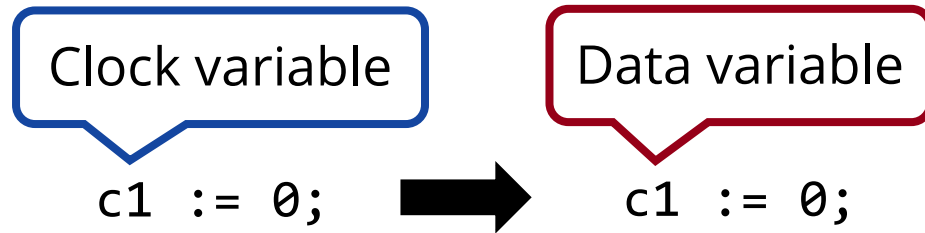


Verification approaches for TXSTS models



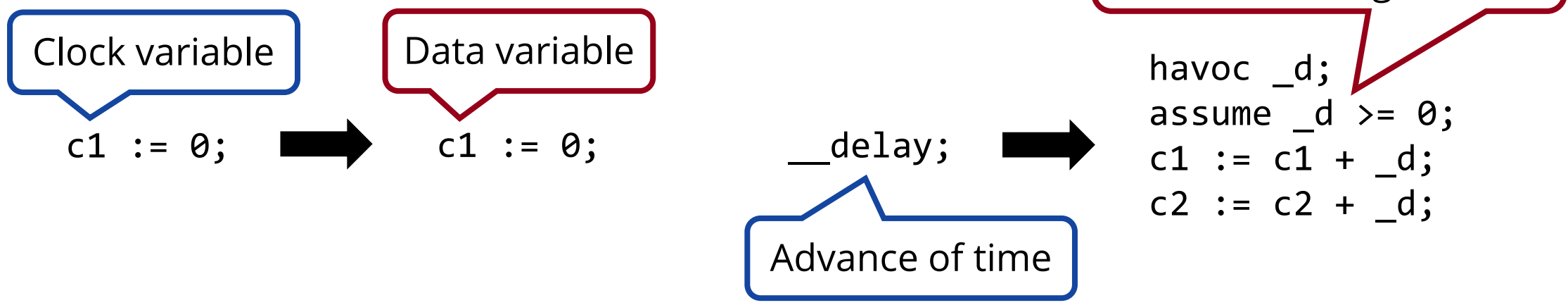
1st approach: transformation of TXSTS to XSTS

- Clocks to rational variables
- Clock operations to data operations



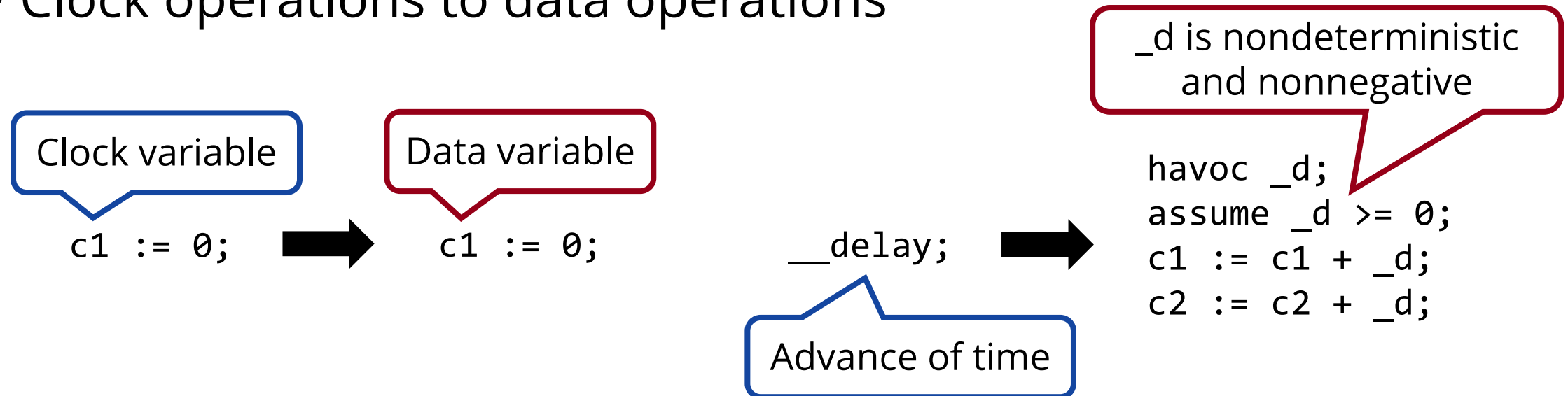
1st approach: transformation of TXSTS to XSTS

- Clocks to rational variables
- Clock operations to data operations



1st approach: transformation of TXSTS to XSTS

- Clocks to rational variables
- Clock operations to data operations



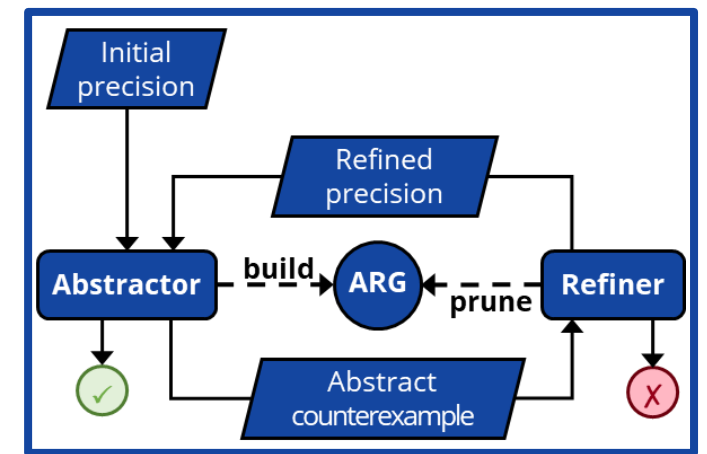
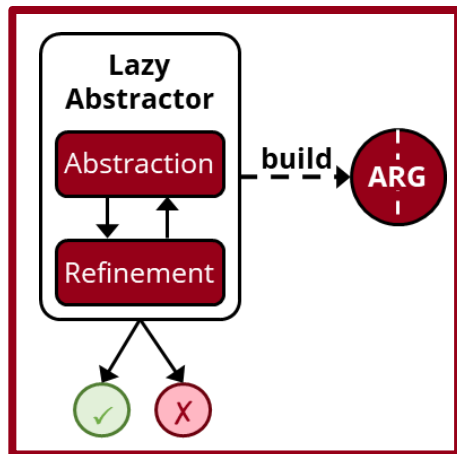
- Existing algorithms can be used without modification
- Efficient time abstraction techniques cannot be used

2nd approach: verification of TXSTS models

- Existing abstraction-based techniques: **lazy abstraction**, **CEGAR**
- Building on **combined abstraction**
 - Lazy abstraction for timing, CEGAR for data

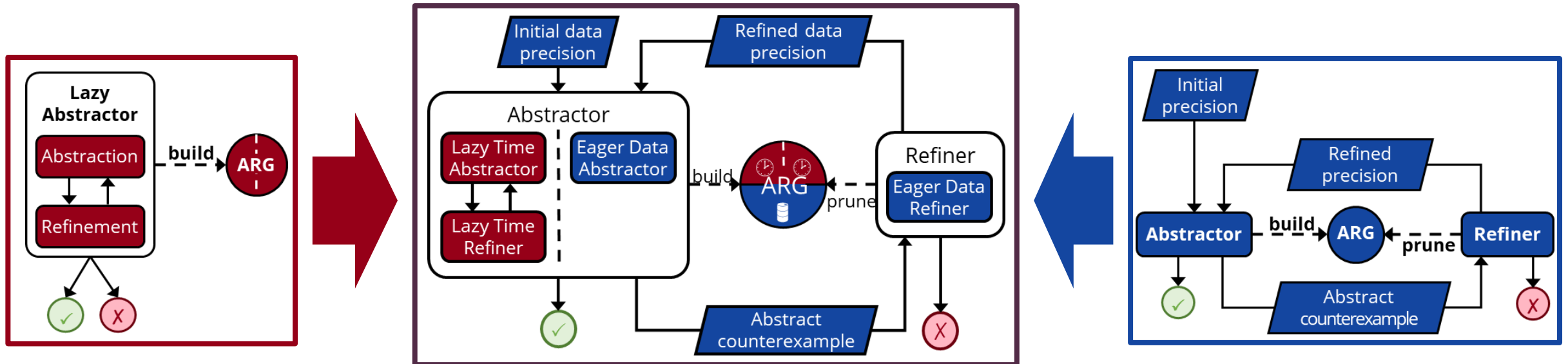
2nd approach: verification of TXSTS models

- Existing abstraction-based techniques: **lazy abstraction**, **CEGAR**
- Building on **combined abstraction**
 - Lazy abstraction for timing, CEGAR for data



2nd approach: verification of TXSTS models

- Existing abstraction-based techniques: **lazy abstraction**, **CEGAR**
- Building on **combined abstraction**
 - Lazy abstraction for timing, CEGAR for data



2nd approach: verification of TXSTS models

- Existing abstraction-based techniques: **lazy abstraction**, **CEGAR**
- Building on **combined abstraction**
 - Lazy abstraction for timing, CEGAR for data



Existing algorithms presume that the results of operations can be computed individually for timing and data

2nd approach: verification of TXSTS models

- Existing abstraction-based techniques: **lazy abstraction**, **CEGAR**
- Building on **combined abstraction**
 - Lazy abstraction for timing, CEGAR for data



Existing algorithms presume that the results of operations can be computed individually for timing and data

- A **problematic example**, with data variable **x** and clock variable **c**

```
if ((x == 0 && c > 500) || (x == 1 && c < 400))  
{ ... }
```

2nd approach: verification of TXSTS models

- Existing abstraction-based techniques: **lazy abstraction**, **CEGAR**
- Building on **combined abstraction**
 - Lazy abstraction for timing, CEGAR for data



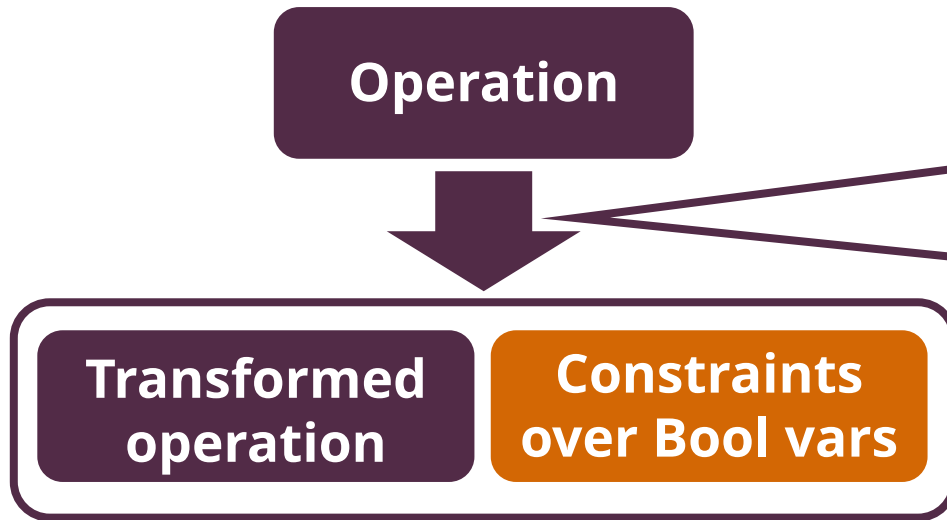
Existing algorithms presume that the results of operations can be computed individually for timing and data

- A **problematic example**, with data variable **x** and clock variable **c**

```
if ((x == 0 && c > 500) || (x == 1 && c < 400))  
{ ... }
```

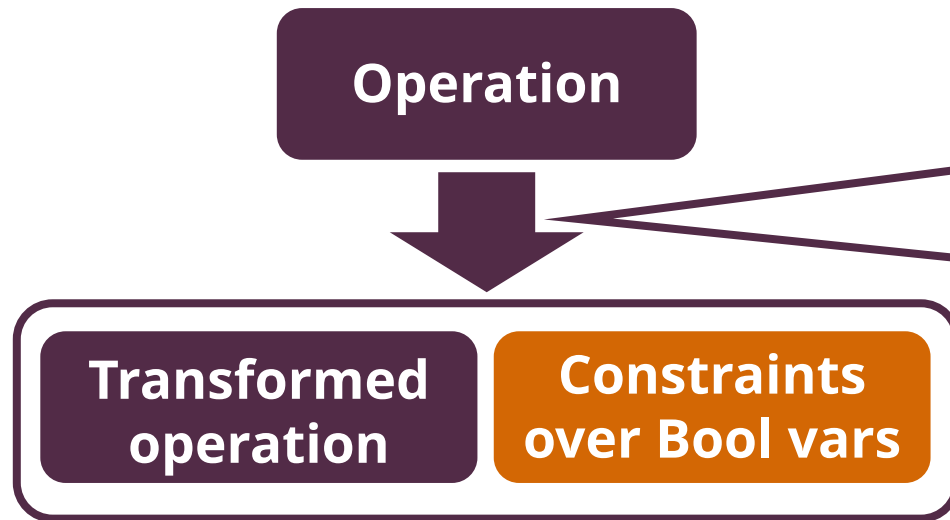
- Solution: **control flow splitting**

Control flow splitting



Transformation:
introducing new **Boolean variables**,
and **constraints** on these variables

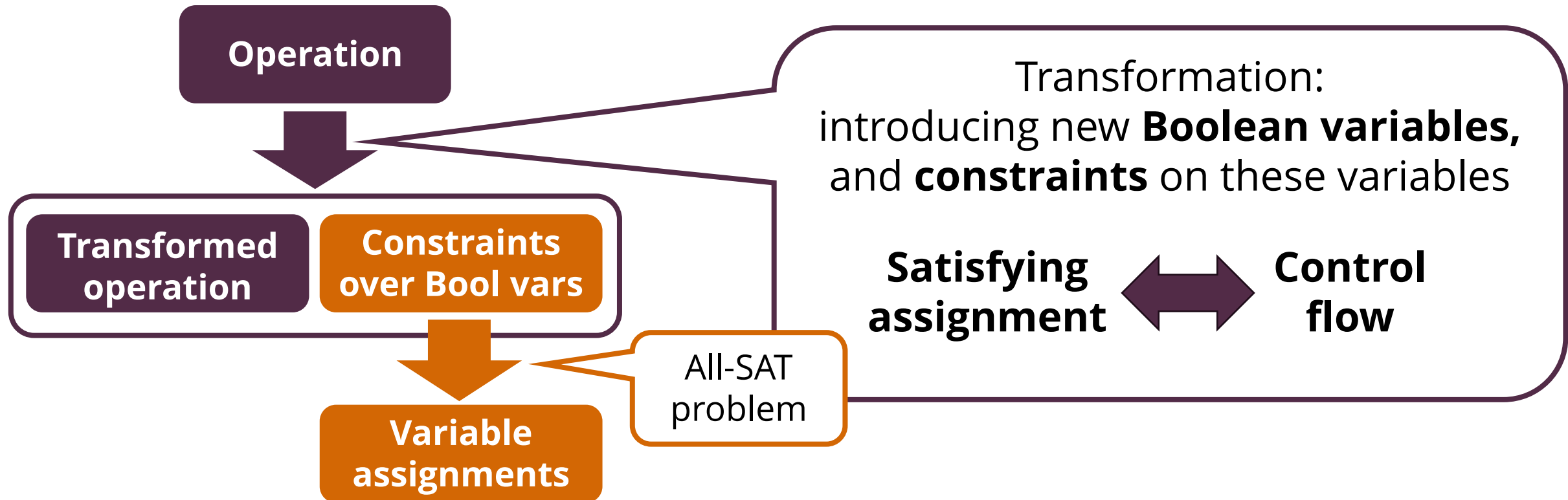
Control flow splitting



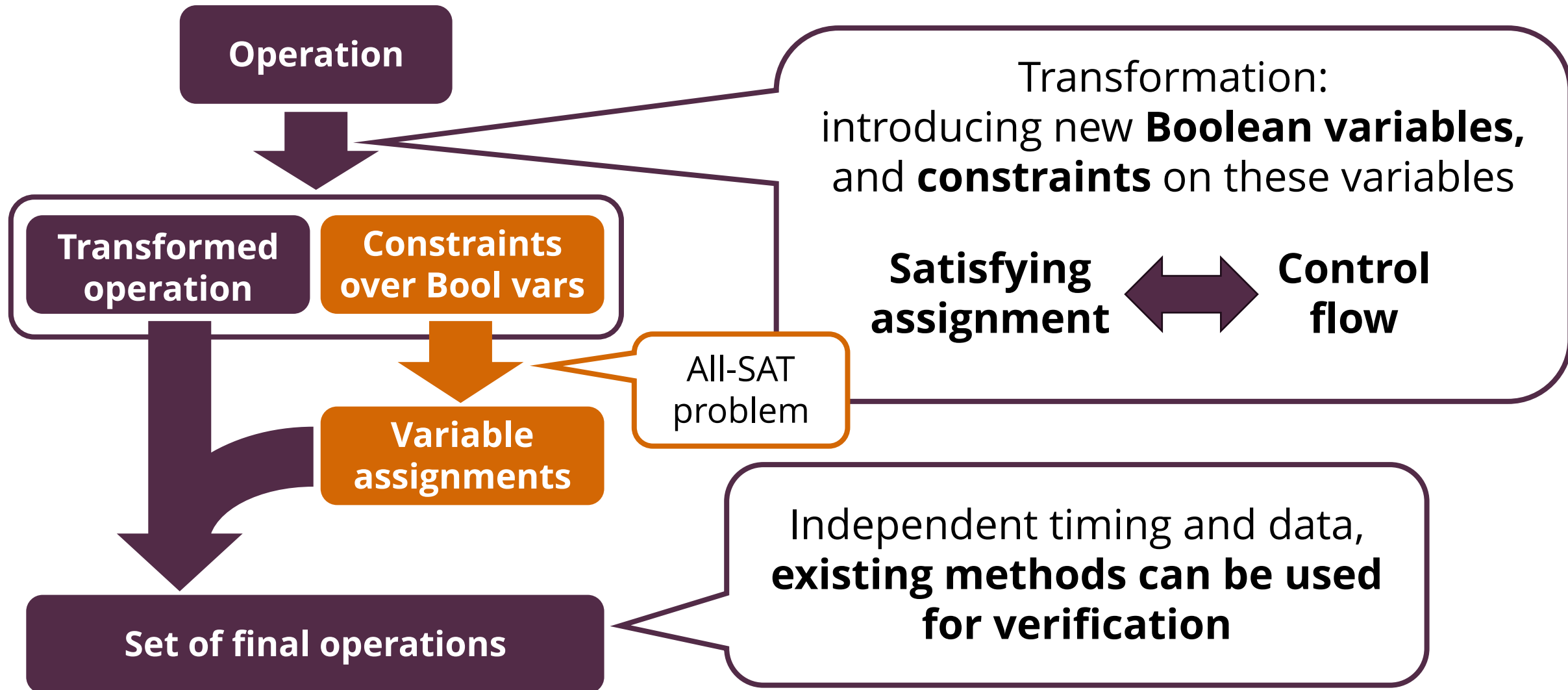
Transformation:
introducing new **Boolean variables**,
and **constraints** on these variables

Satisfying assignment ↔ **Control flow**

Control flow splitting

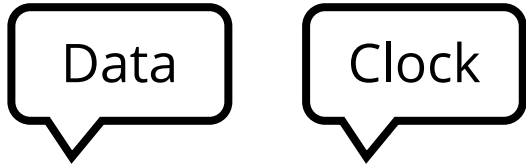


Control flow splitting



Example - operation transformation

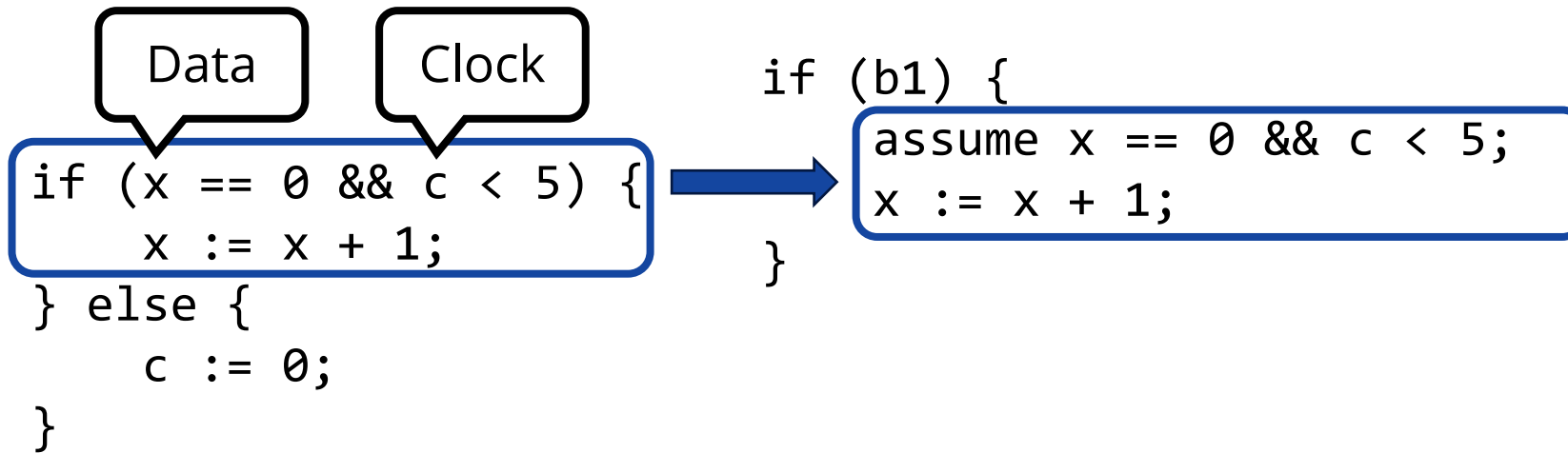
Boolean vars + constraints: satisfying assignment \leftrightarrow control flow



```
if (x == 0 && c < 5) {  
    x := x + 1;  
} else {  
    c := 0;  
}
```

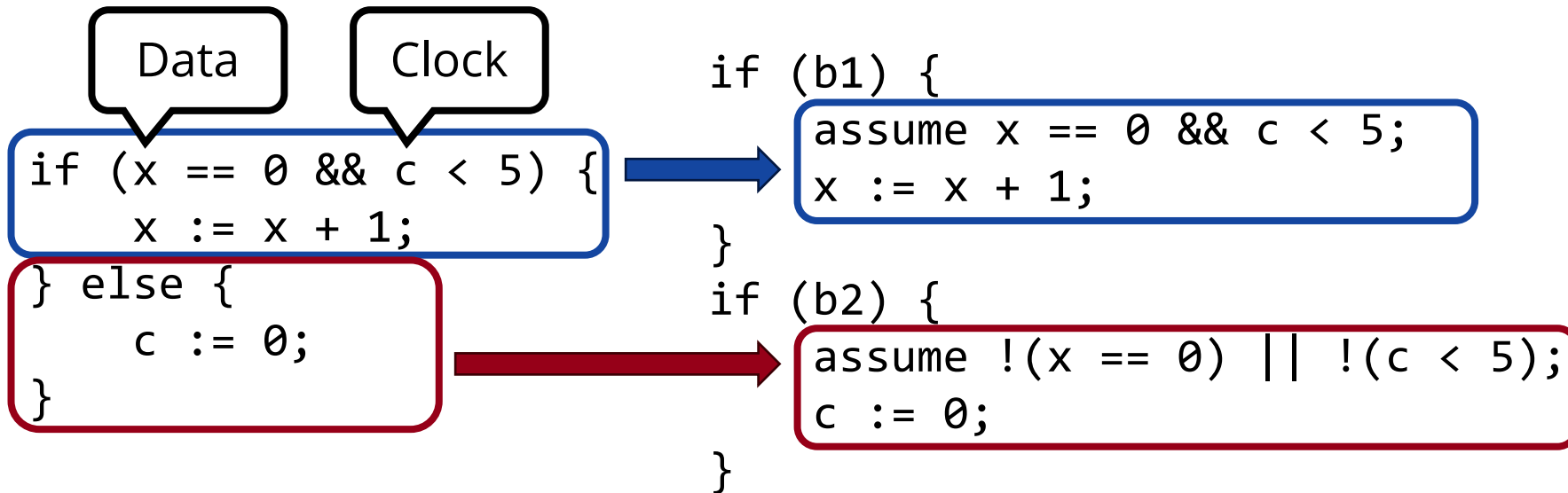
Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow



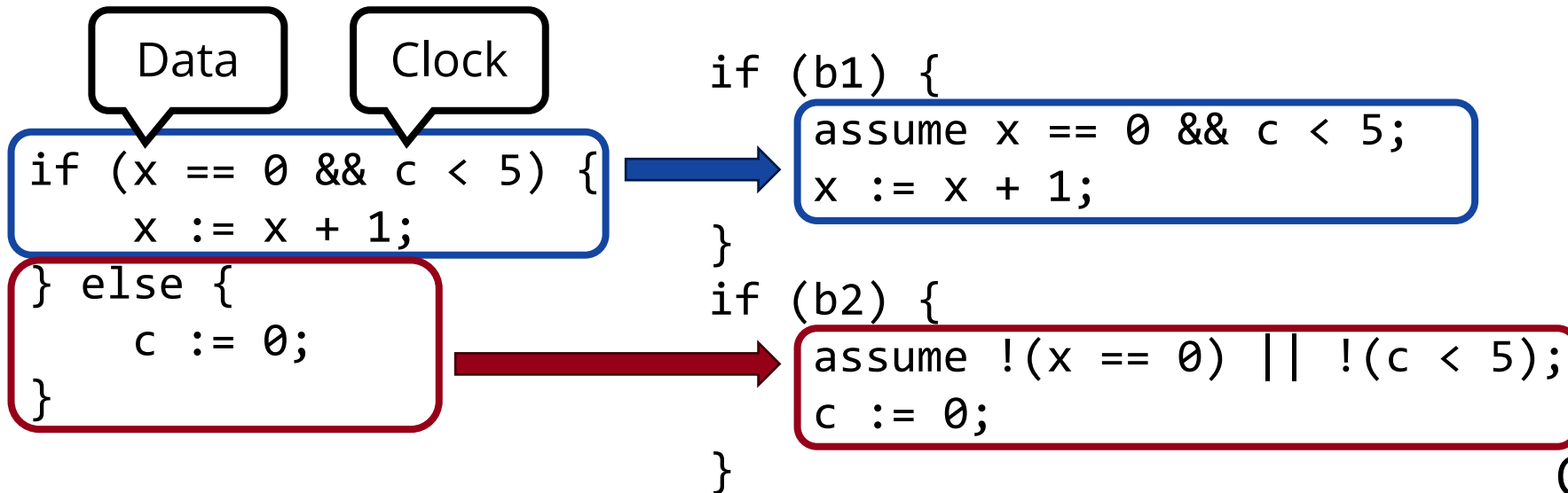
Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow



Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow



Constraints:

- b1 xor b2

Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow

```
if (x == 0 && c < 5) {  
    x := x + 1;  
} else {  
    c := 0;  
}
```

```
if (b1) {  
    assume x == 0 && c < 5;  
    x := x + 1;  
}  
if (b2) {  
    assume !(x == 0) || !(c < 5);  
    c := 0;  
}
```

Note: In the original image, callout boxes labeled 'Data' and 'Clock' point to the expressions $x == 0$ and $c < 5$ in both code blocks.

Constraints:

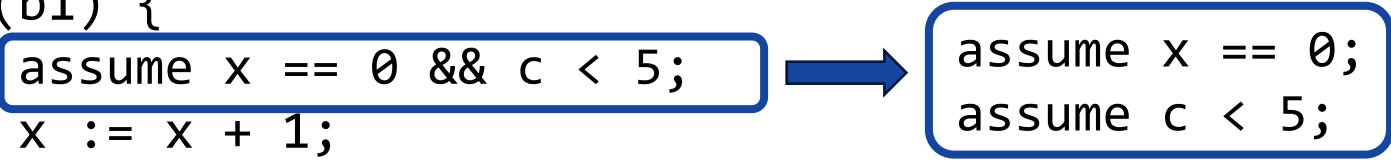
- b1 xor b2

Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow

```
if (x == 0 && c < 5) {  
    x := x + 1;  
} else {  
    c := 0;  
}
```

```
if (b1) {  
    assume x == 0 && c < 5;  
    x := x + 1;  
}  
if (b2) {  
    assume !(x == 0) || !(c < 5);  
    c := 0;  
}
```



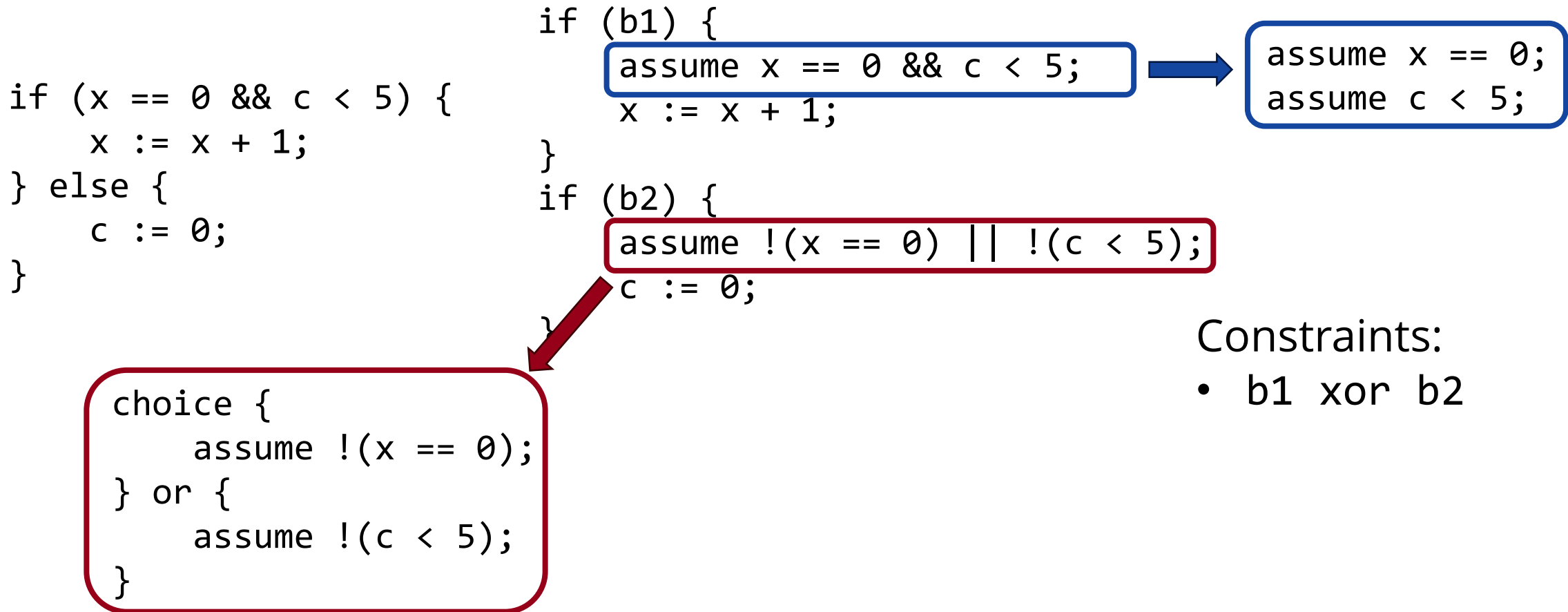
```
assume x == 0;  
assume c < 5;
```

Constraints:

- b1 xor b2

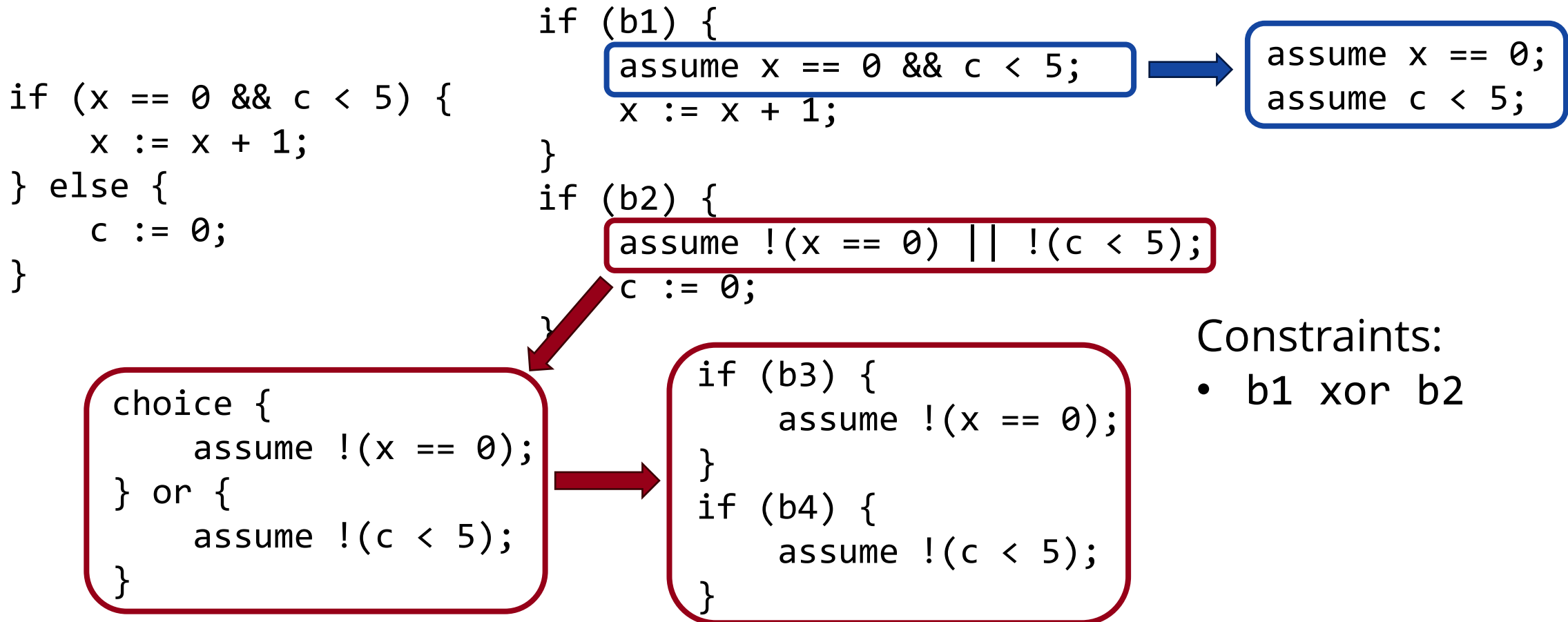
Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow



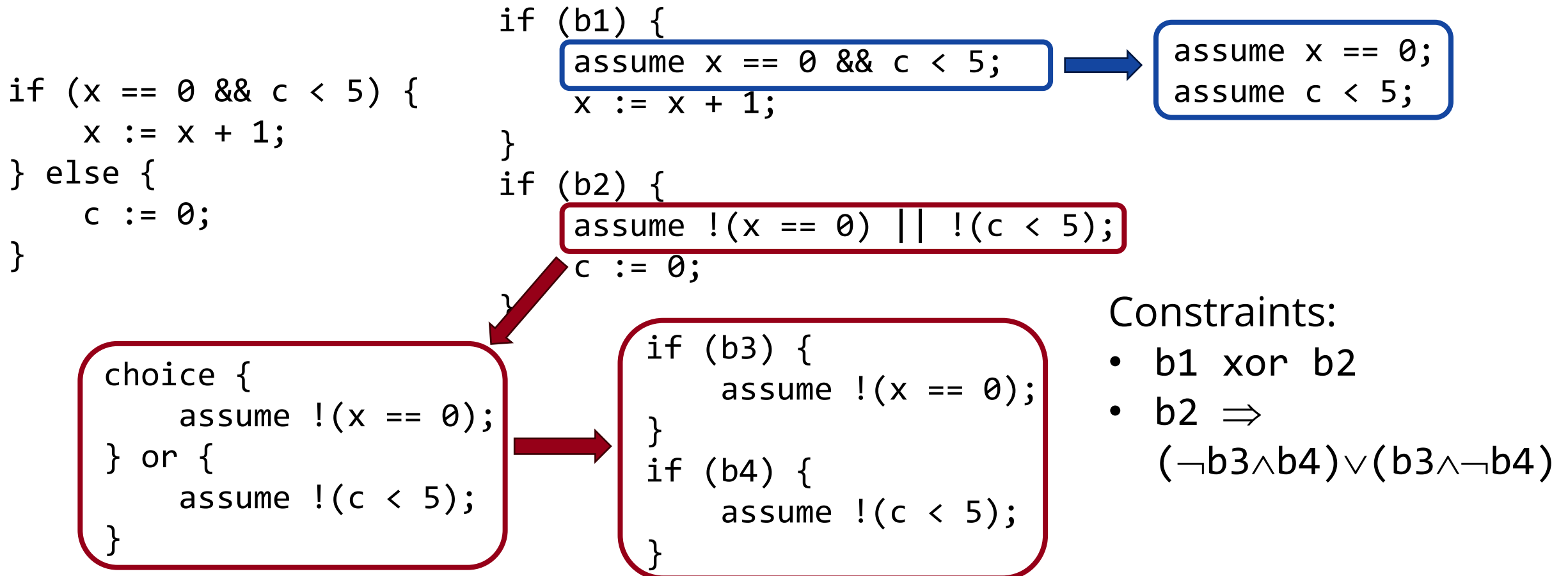
Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow



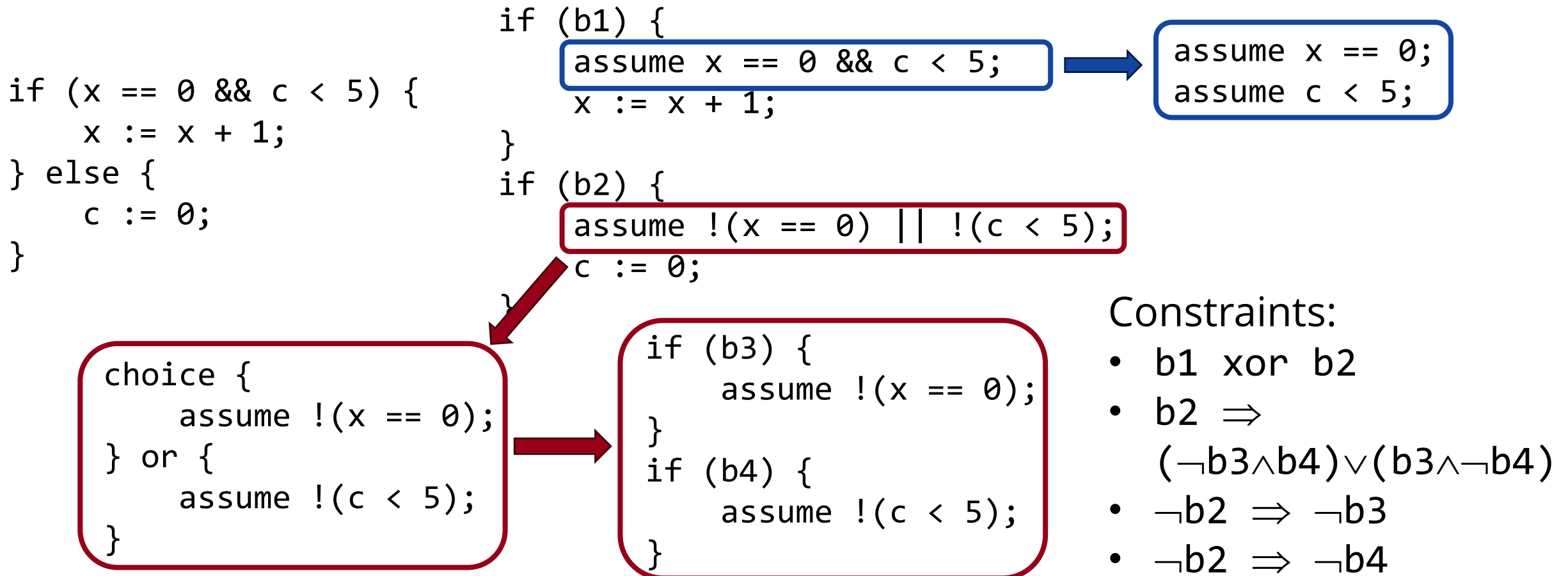
Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow



Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow



Example - operation transformation

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow

```
if (x == 0 && c < 5) {
    x := x + 1;
} else {
    c := 0;
}

if (b1) {
    assume x == 0;
    assume c < 5;
    x := x + 1;
}
if (b2) {
    if (b3) {
        assume !(x == 0);
    }
    if (b4) {
        assume !(c < 5);
    }
    c := 0;
}
```

Constraints:

- $b1 \text{ xor } b2$
- $b2 \Rightarrow (\neg b3 \wedge b4) \vee (b3 \wedge \neg b4)$
- $\neg b2 \Rightarrow \neg b3$
- $\neg b2 \Rightarrow \neg b4$

Example – control flow with an SMT solver

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow

```
if (x == 0 && c < 5) {  
    x := x + 1;  
} else {  
    c := 0;  
}  
  
if (b1) {          b1 = false  
    assume x == 0;  
    assume c < 5;  
    x := x + 1;  
}  
if (b2) {          b2 = true  
    if (b3) {      b3 = true  
        assume !(x == 0);  
    }  
    if (b4) {      b4 = false  
        assume !(c < 5);  
    }  
    c := 0;  
}
```

Constraints:

- $b1 \text{ xor } b2$
- $b2 \Rightarrow (\neg b3 \wedge b4) \vee (b3 \wedge \neg b4)$
- $\neg b2 \Rightarrow \neg b3$
- $\neg b2 \Rightarrow \neg b4$

Example – control flow with an SMT solver

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow

```
if (x == 0 && c < 5) {  
    x := x + 1;  
} else {  
    c := 0;  
}
```

```
assume !(x == 0);  
c := 0;
```

```
if (b1) {          b1 = false  
    assume x == 0;  
    assume c < 5;  
    x := x + 1;  
}  
if (b2) {          b2 = true  
    if (b3) {      b3 = true  
        assume !(x == 0);  
    }  
    if (b4) {      b4 = false  
        assume !(c < 5);  
    }  
    c := 0;  
}
```

Constraints:

- $b1 \text{ xor } b2$
- $b2 \Rightarrow (\neg b3 \wedge b4) \vee (b3 \wedge \neg b4)$
- $\neg b2 \Rightarrow \neg b3$
- $\neg b2 \Rightarrow \neg b4$

Example - control flow with an SMT solver

Boolean vars + constraints: satisfying assignment \leftrightarrow control flow

```
if (x == 0 && c < 5) {  
    x := x + 1;  
} else {  
    c := 0;  
}
```

```
assume x == 0;  
assume c < 5;  
x := x + 1;
```

```
assume !(x == 0);  
c := 0;
```

```
assume !(c < 5);  
c := 0;
```

```
if (b1) {          b1 = false  
    assume x == 0;  
    assume c < 5;  
    x := x + 1;  
}  
if (b2) {          b2 = true  
    if (b3) {      b3 = true  
        assume !(x == 0);  
    }  
    if (b4) {      b4 = false  
        assume !(c < 5);  
    }  
    c := 0;  
}
```

Constraints:

- $b1 \text{ xor } b2$
- $b2 \Rightarrow (\neg b3 \wedge b4) \vee (b3 \wedge \neg b4)$
- $\neg b2 \Rightarrow \neg b3$
- $\neg b2 \Rightarrow \neg b4$

Preliminary evaluation of the approaches

- Implemented in the **Theta** open source verification framework
- Two TXSTS models from Gamma engineering models:
 - Example model demonstrating the capabilities of Gamma: **crossroad**
 - Industrial case study: model of a **safety-critical railway protocol**
- 30 reachability properties, analyzed in two ways:
 - **Reachability** of a given state
 - **Timed reachability**: reachability of given state **under a given time limit**

Preliminary evaluation of the approaches

- 3 CPU cores, time limit of 20 minutes, memory limit of 15 GB
- Best configurations of both approaches compared: number of verified properties, with mean CPU time

Approach	Verified properties with time limit of 20 minutes	
	Reachability	Timed reachability
Time → data transformation	30/30 (100%) 7.48 s	12/30 (40%) 2.26 s
Combined abstraction with control flow splitting	30/30 (100%) 11.09 s	18/30 (60%) 40.99 s

- **Reachability**: same success rate, **time→data transf.** is faster
- **Timed** reachability: **control flow splitting** is more successful

Summary

