# Witnesses: From Proof Certificates to Cooperation Data

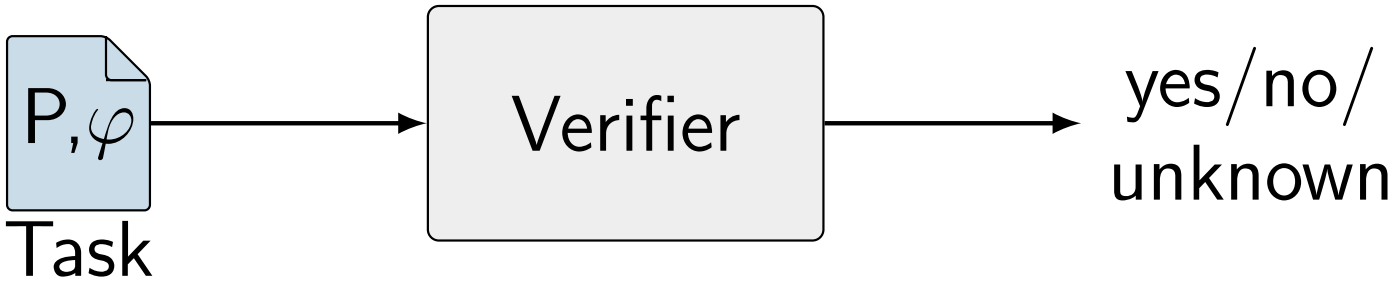
Heike Wehrheim

University of Oldenburg
Germany

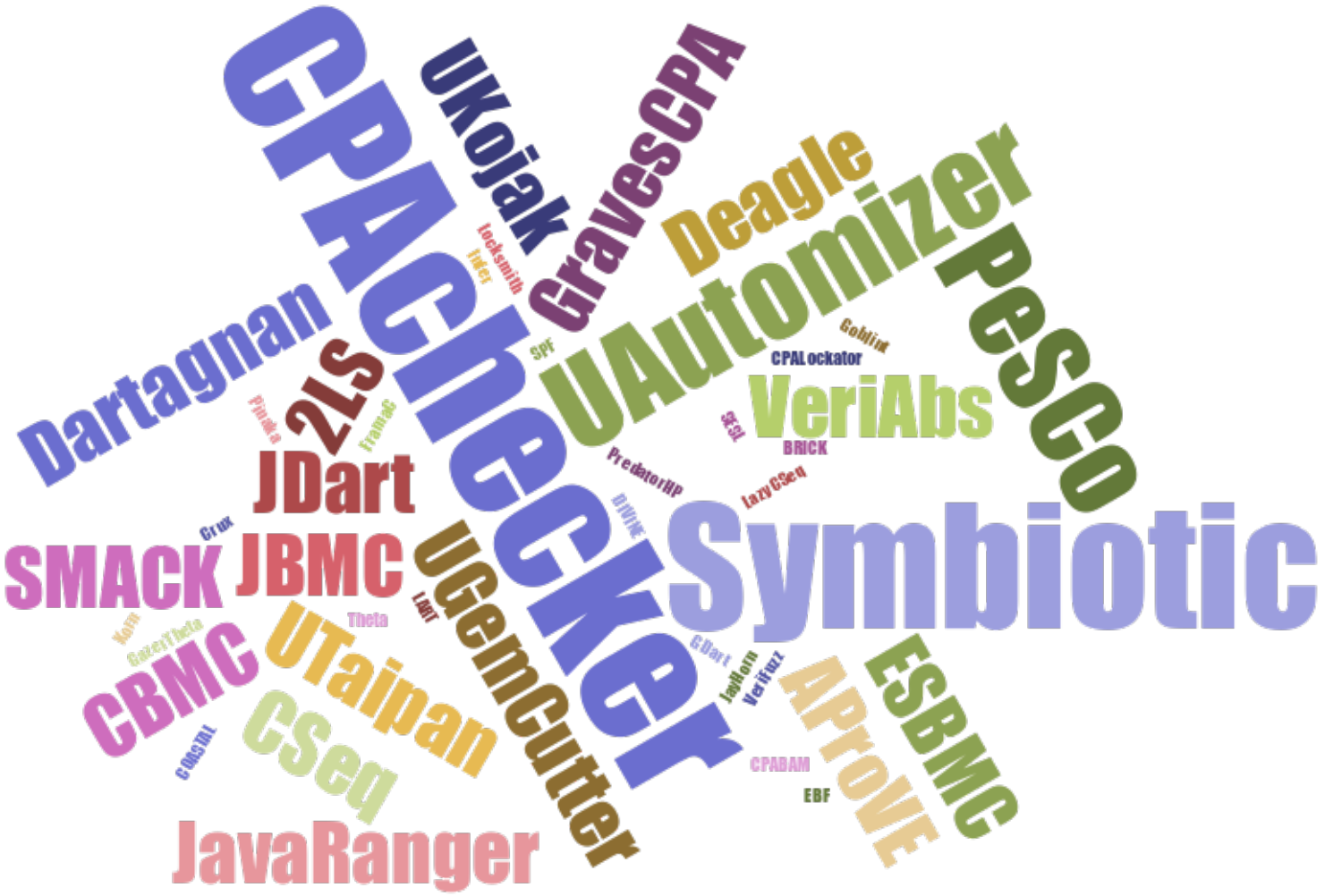
Joint work with:
Marie-Christine Jakobs, Jan Haltermann, Dirk Beyer, Thomas Lemberger

# Automatic Software Verification

Task: $P$ - program, $\varphi$ - specification



# Plethora of Verifiers



# Verification Artefacts

Verifiers produce numerous sorts of data about programs

- proofs,
- counterexamples,
- test cases,
- precisions,
- ...

$\Rightarrow$ verification artefacts

# Verification Artefacts

Verifiers produce numerous sorts of data about programs

- proofs,
- counterexamples,
- test cases,
- precisions,
- ...

$\Rightarrow$ verification artefacts

witness = artefact witnessing some verification result

# This Talk

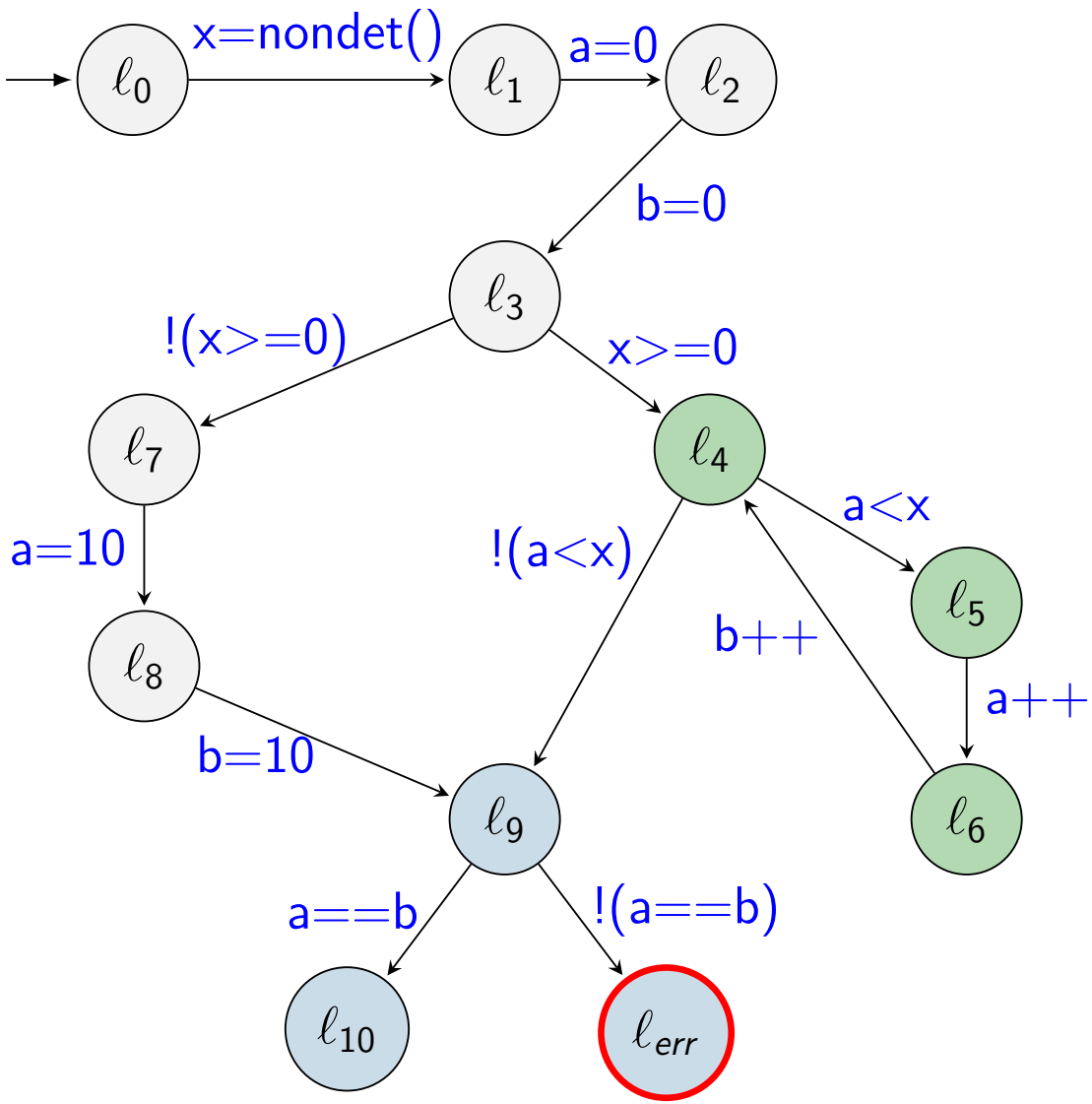
Witnesses in software verification used as ...

- ... proof certificates
- ... cooperation data

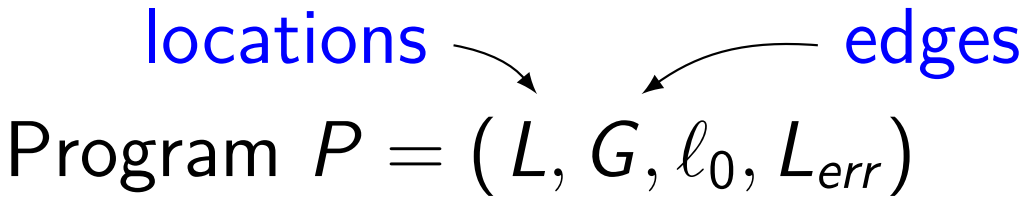
# Basics

# Programs and CFAs

```
int x = nondet();
int a = 0;
int b = 0;
if (x >= 0)
    while (a < x)
        a++;
        b++;
else
    a = 10;
    b = 10;
assert (a==b);
```



# Safety



# Safety

locations edges

Program $P = (L, G, \ell_0, L_{err})$

Semantics: transition system $T(P) = (C, \rightarrow, c_0)$

- $C$: set of concrete states, $c_0 \in C$: initial state,
- $\rightarrow \subseteq C \times G \times C$: transitions.

# Safety

locations edges

Program $P = (L, G, \ell_0, L_{err})$

Semantics: transition system $T(P) = (C, \rightarrow, c_0)$

- $C$: set of concrete states, $c_0 \in C$: initial state,
- $\rightarrow \subseteq C \times G \times C$: transitions.

Program safe if

$$c_0 \xrightarrow{g_1} c_1 \xrightarrow{g_2} \ldots \xrightarrow{g_n} c_{err}$$

$(c_{err} = (\ell_{err}, \cdot))$ not possible

# Abstract Analysis

Abstract domain $\mathbb{A}$:

- lattice $(E_{\mathbb{A}}, \sqsubseteq_{\mathbb{A}})$ of abstract states,
- concretisation function $[\![\cdot]\!] : E_{\mathbb{A}} \to 2^C$,
- transfer relation $\rightsquigarrow_{\mathbb{A}} \subseteq E_{\mathbb{A}} \times G \times E_{\mathbb{A}}$.

Termination:

stop : $E_{\mathbb{A}} \times 2^{E_{\mathbb{A}}} \to \mathbb{B}$ (can we stop now?)

# Two Analyses

Value analysis:

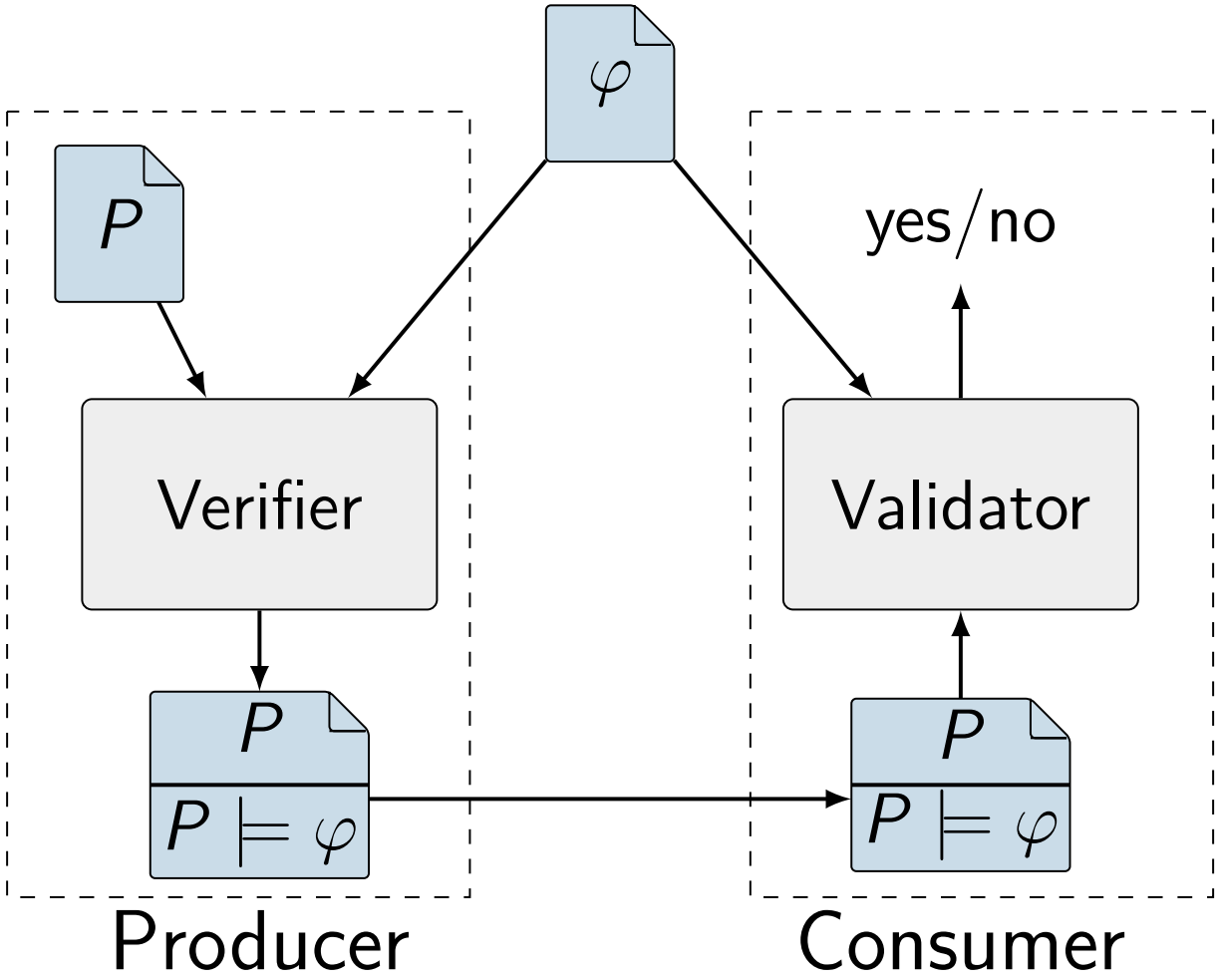
- track values of (some) variables
- e.g. $a : 10, b : ?, x : ?$

Predicate analysis:

- track predicates about (some) variables
- e.g. $a - 1 = b$

# Topic 1:
# Proof-Carrying Code (PCC)

# PCC Scheme




[Necula: Proof-Carrying Code. POPL 1997.]

# PCC from Software Verification

Idea:

- employ data computed by verifier as proof
- design a validation technique for checking whether proof guarantees safety
- evaluate whether proof validation is simpler than proof generation

# Abstract Reachability Graph ARG

Reminder: $E_{\mathbb{A}}$: abstract states

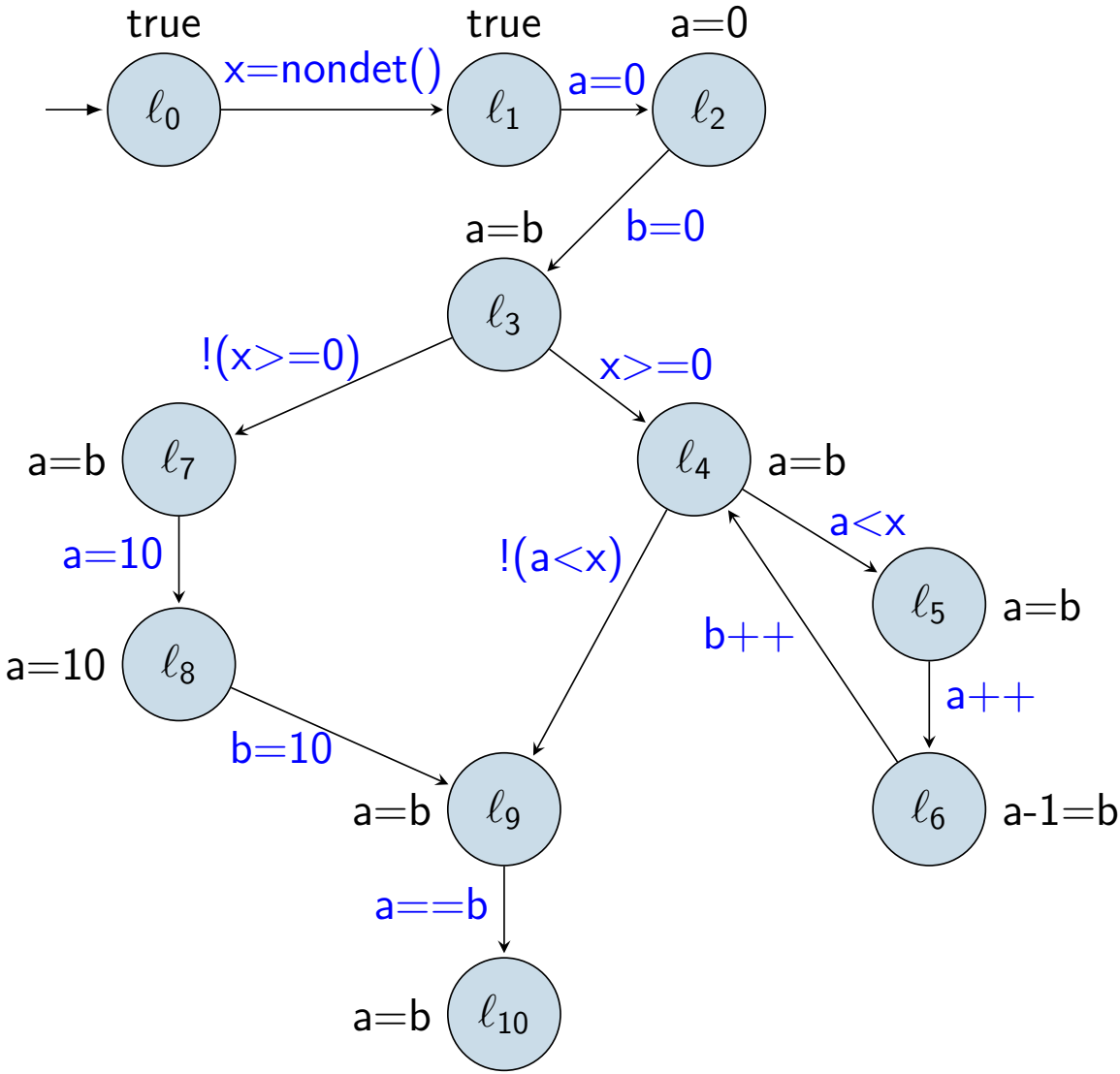
## Definition

ARG $A = (N, \delta, n_0)$ for program $P = (L, G, \ell_0, L_{err})$ wrt. analysis $\mathbb{A}$

- Nodes $N \subseteq E_{\mathbb{A}}$, $n_0 \in N$ initial node,
- edges $\delta \subseteq N \times G \times N$.

# ARG of Example ($\mathbb{A}$=predicate analysis)

```
int x = nondet();
int a = 0;
int b = 0;
if (x >= 0)
    while (a < x)
        a++;
        b++;
else
    a = 10;
    b = 10;
assert (a==b);
```



# Using ARGs as Proofs

Reminder: $[\![\cdot]\!] : E_{\mathbb{A}} \to 2^C$ concretisation

An ARG $A = (N, \delta, n_0)$ for program $P = (L, G, \ell_0, L_{err})$ wrt. analysis $\mathbb{A}$ is ...

**Rooted** $c_0 \in [\![n_0]\!]$

**Sound** $\forall n \in N, g \in G :$
$n \overset{g}{\rightsquigarrow}_{\mathbb{A}} e$ implies $\mathsf{stop}(e, \{n' \mid (n, g, n') \in \delta\}$

**Safe** $c_{err} \notin [\![N]\!]$.

# Using ARGs as Proofs

Reminder: $[\![\cdot]\!] : E_{\mathbb{A}} \to 2^C$ concretisation

An ARG $A = (N, \delta, n_0)$ for program $P = (L, G, \ell_0, L_{err})$ wrt. analysis $\mathbb{A}$ is ...

- Rooted $c_0 \in [\![n_0]\!]$
- Sound $\forall n \in N, g \in G :$
  $n \overset{g}{\rightsquigarrow}_{\mathbb{A}} e$ implies $\mathsf{stop}(e, \{n' \mid (n, g, n') \in \delta\}$
- Safe $c_{err} \notin [\![N]\!]$.

An ARG is a proof witness if it is rooted, sound and safe.

# Using ARGs as Proofs

Reminder: $[\![\cdot]\!] : E_{\mathbb{A}} \to 2^C$ concretisation

An ARG $A = (N, \delta, n_0)$ for program $P = (L, G, \ell_0, L_{err})$ wrt. analysis $\mathbb{A}$ is ...

Rooted $c_0 \in [\![n_0]\!]$

Sound $\forall n \in N, g \in G :$
$n \overset{g}{\rightsquigarrow}_{\mathbb{A}} e$ implies $\mathsf{stop}(e, \{n' \mid (n, g, n') \in \delta\}$

Safe $c_{err} \notin [\![N]\!]$.

An ARG is a proof witness if it is rooted, sound and safe.

**Theorem**

*A proof witness ensures program safety.*

# Applied in PCC

$\mathbb{A}$-Verifier:

- Construct ARG (i.e., proof $P \models \varphi$) for $P$ and $\varphi$ using analysis $\mathbb{A}$,
- attach ARG to program.

$\mathbb{A}$-Validator:

- Check rootedness, soundness and safety.

[Jakobs, Wehrheim: Certification for configurable program analysis. SPIN 2014]

# What if the verifier cannot complete verification?

Partial ARG:

ARG $A = (N, \delta, n_0)$ for program $P = (L, G, \ell_0, L_{err})$ wrt. $\mathbb{A}$ which is

Rooted $c_0 \in [\![n_0]\!]$

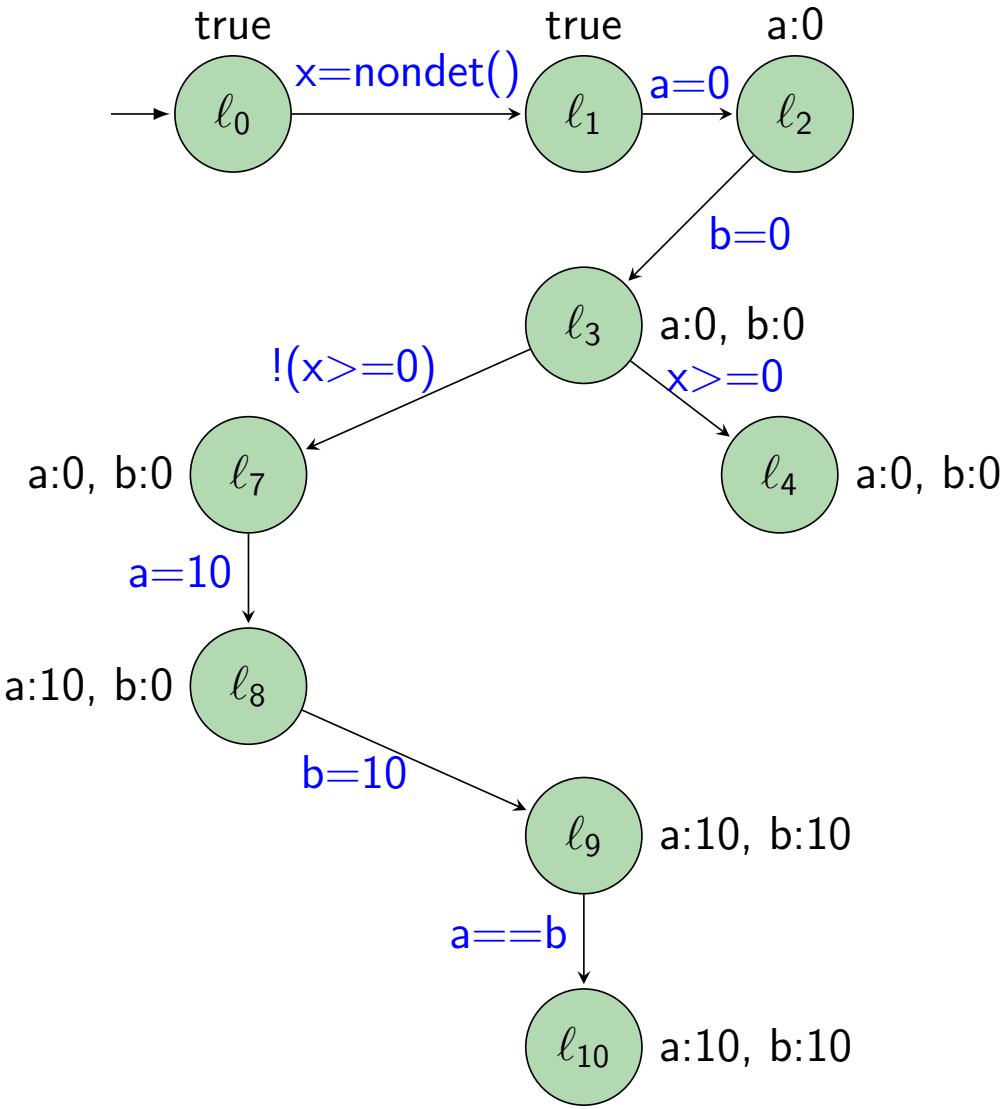
Partially sound $\forall n \in N, g \in G$ :

$$n \overset{g}{\rightsquigarrow}_{\mathbb{A}} e \text{ implies } \mathsf{stop}(e, \{n' \mid (n, g, n') \in \delta\} \vee \neg\exists(n, g, \cdot) \in \delta$$

Safe $c_{err} \notin [\![N]\!]$.

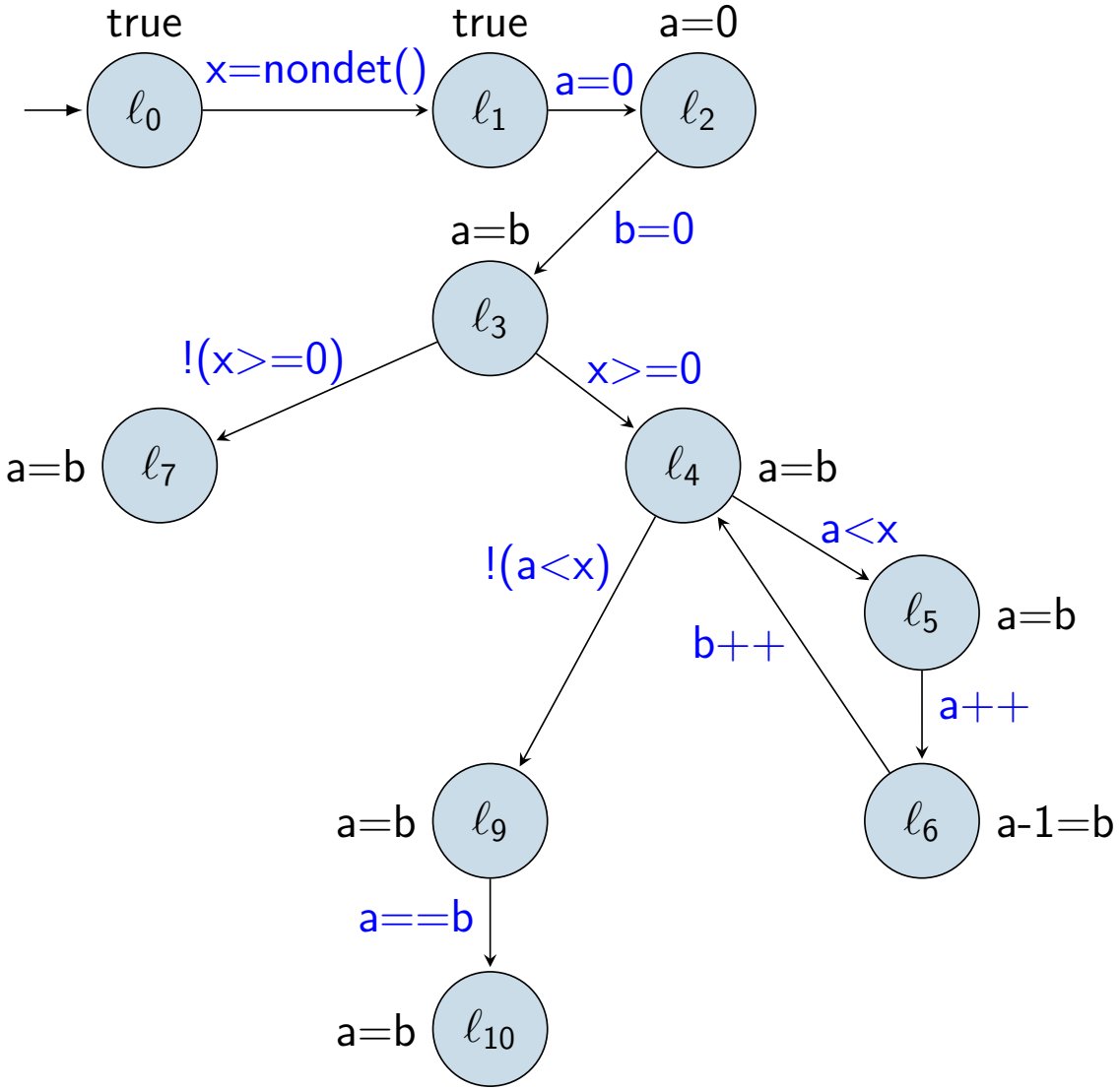
# Partial ARG for Example I

## Partial ARG for $\mathbb{A}$=value analysis



# Partial ARG for Example II

## Partial ARG for $\mathbb{A}$=predicate analysis



# Completion

Incomplete path:
a path $n_0\ g_1\ n_1\ g_2 \ldots n$ in a partial ARG with
$n \overset{g}{\rightsquigarrow}_{\mathbb{A}} e$ but $\neg\exists(n, g, \cdot) \in \delta$

A pair of partial ARGs $A_1$ and $A_2$ is jointly complete if for every incomplete path $\rho$ in $A_1$ there exists a complete path $\rho'$ with $\rho \preceq \rho'$ in $A_2$ and vice versa.

# Completion

Incomplete path:
a path $n_0\ g_1\ n_1\ g_2 \ldots n$ in a partial ARG with
$n \overset{g}{\rightsquigarrow}_{\mathbb{A}} e$ but $\neg\exists(n, g, \cdot) \in \delta$

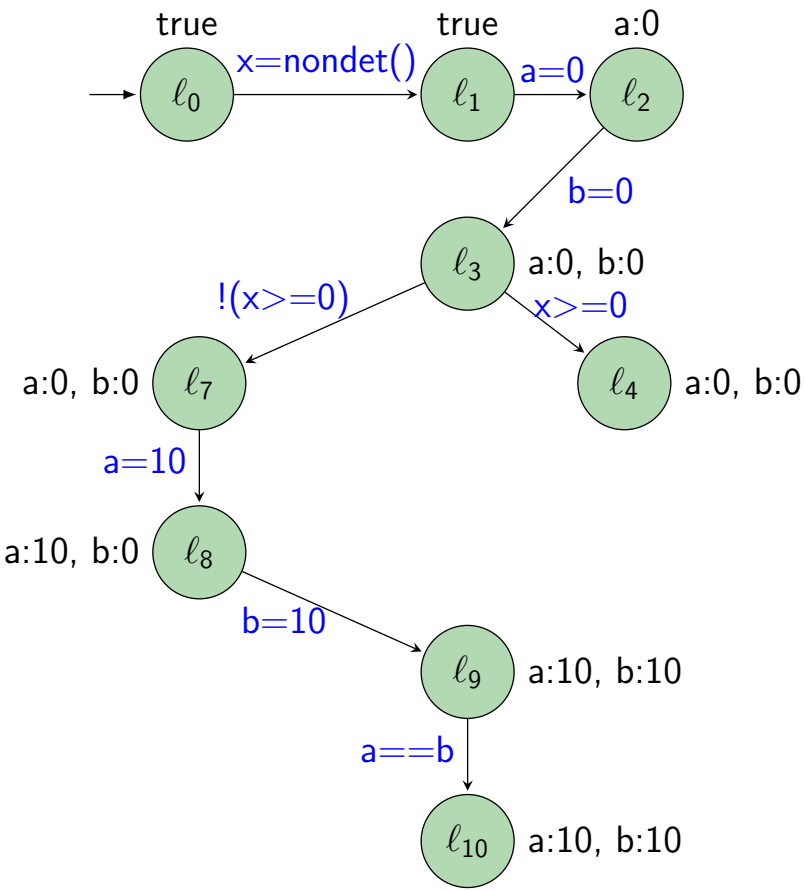
A pair of partial ARGs $A_1$ and $A_2$ is jointly complete if for every incomplete path $\rho$ in $A_1$ there exists a complete path $\rho'$ with $\rho \preceq \rho'$ in $A_2$ and vice versa.

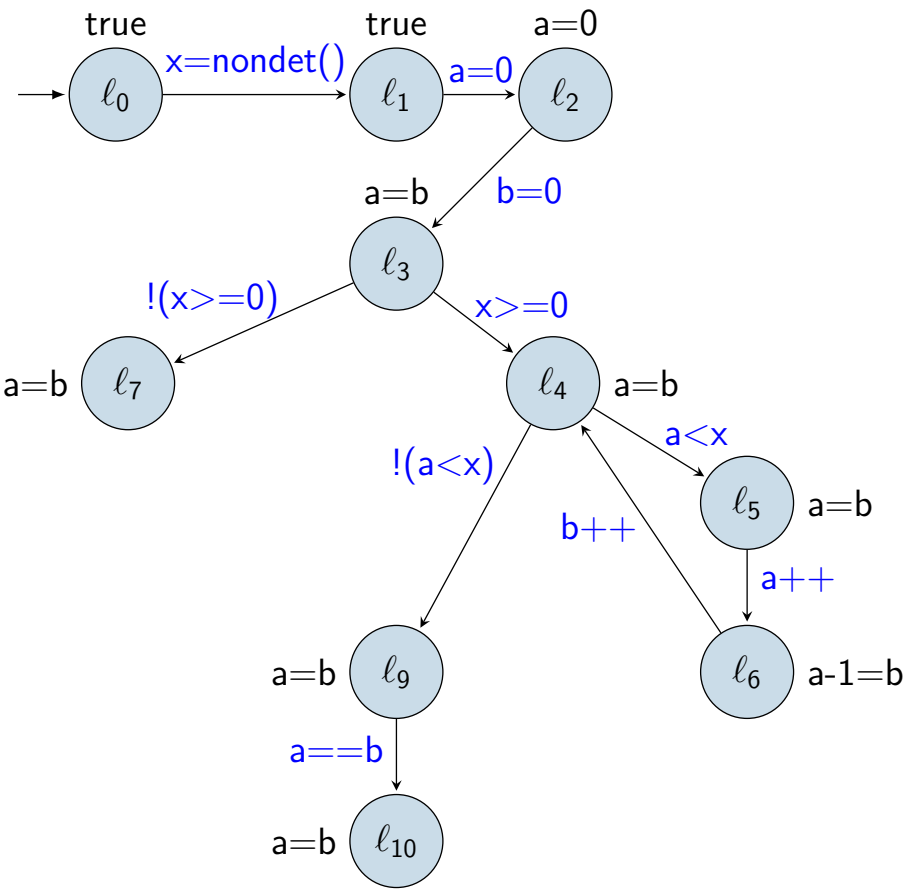
**Theorem**

*A jointly complete pair of partial ARGs ensures program safety.*

# Witness Joining I

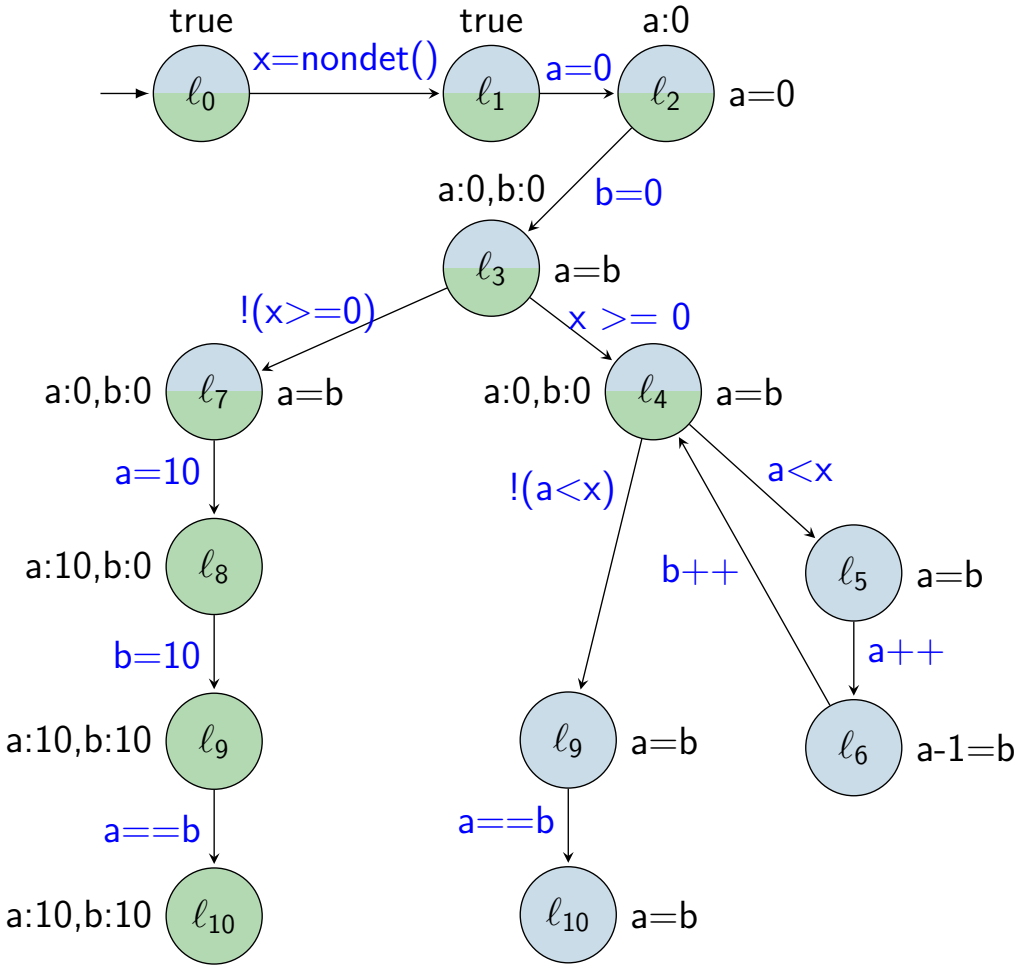
## Joining partially complete ARGs: product construction



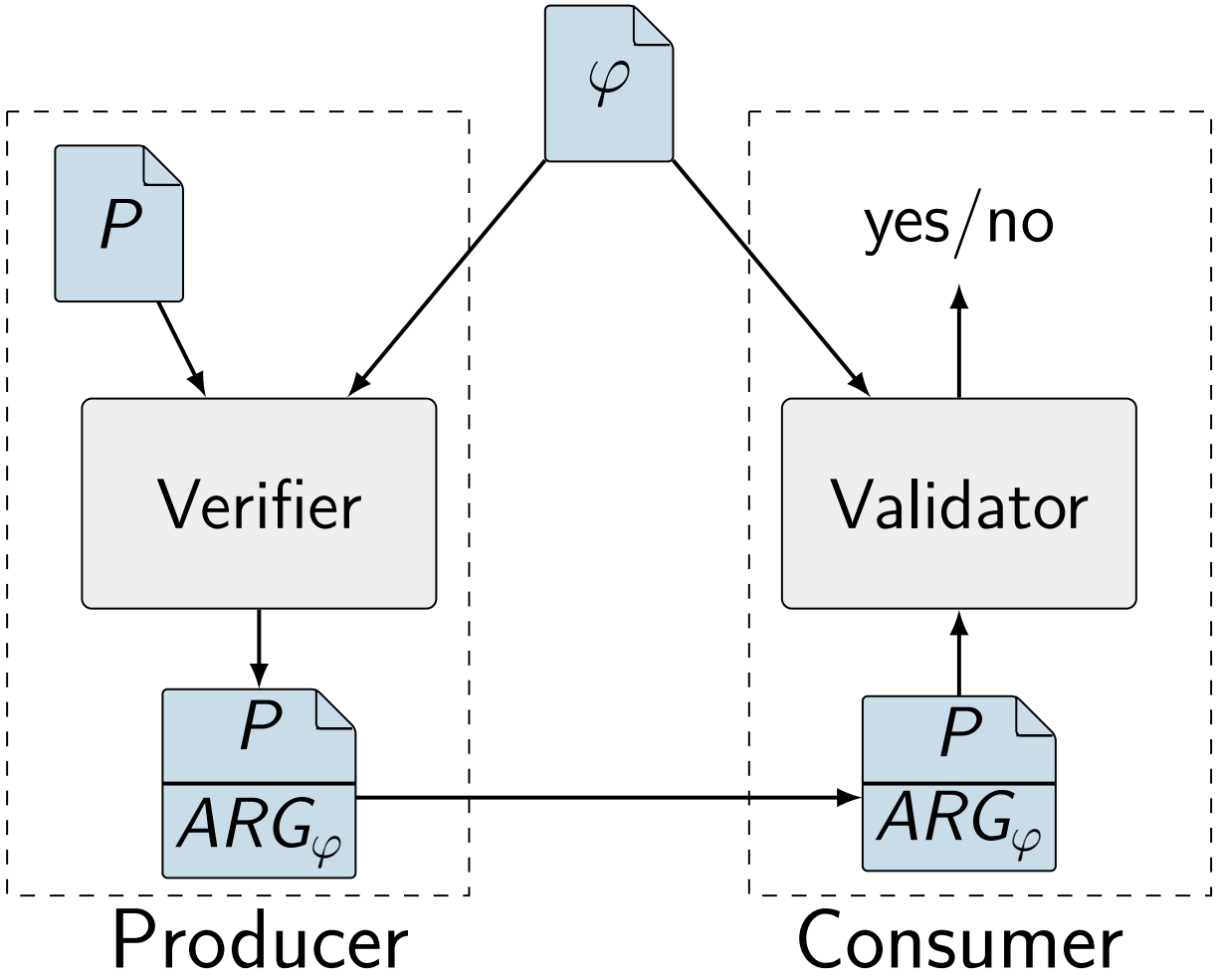
# Witness Joining II

## Result




[Jakobs: PART$_{\mathrm{PW}}$: From Partial Analysis Results to a Proof Witness. SEFM 2017]

# Witnesses in PCC



# Topic 2:
# Cooperative Verification

# Idea

Verifiers have strengths and weaknesses

Cooperative verification:

- verifiers cooperate on a verification task,
- and exchange information (as witnesses).

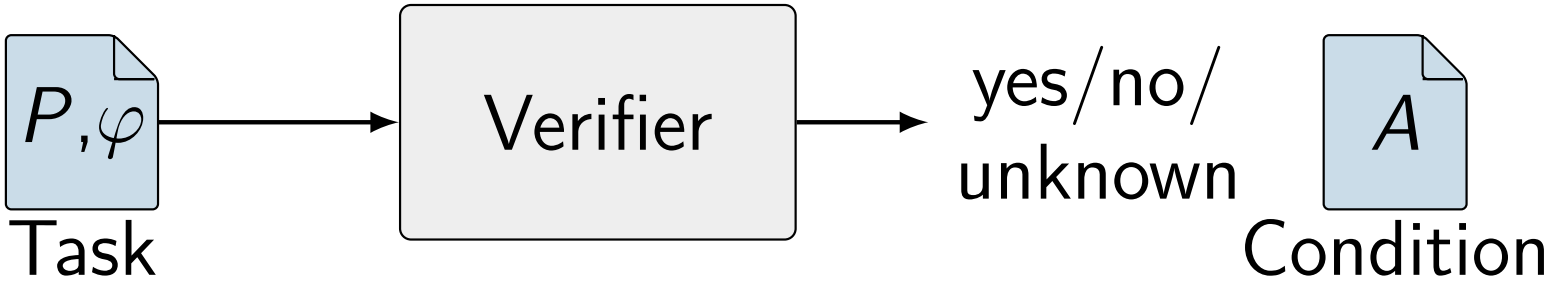
Two applications:

- sequential (conditional model checking)
- cyclic (component-based CEGAR)

# Topic 2a:
# Sequential Cooperation

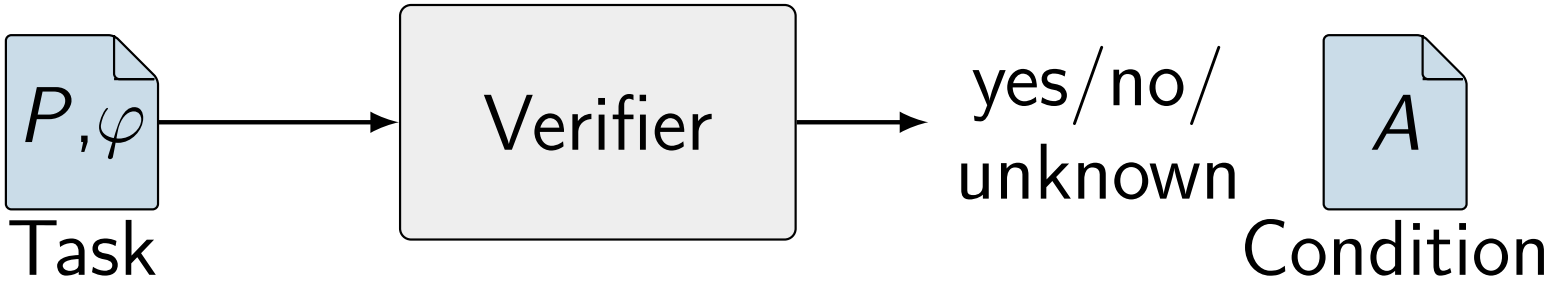
# Conditional Model Checking (CMC)

CMC:

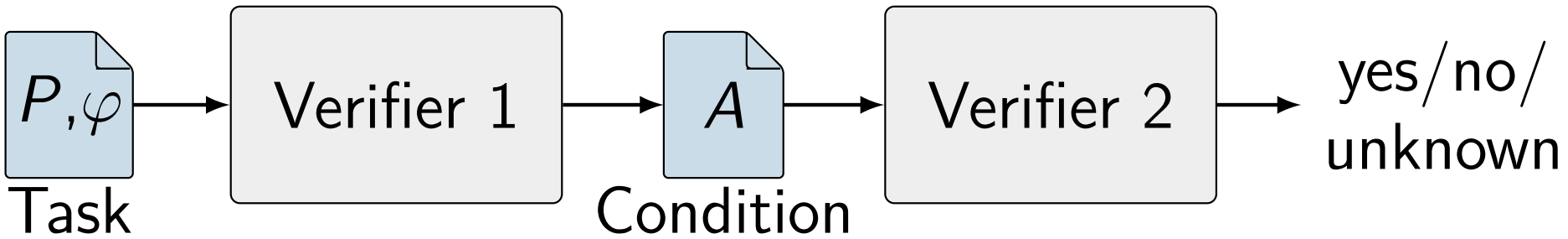


# Conditional Model Checking (CMC)

CMC:



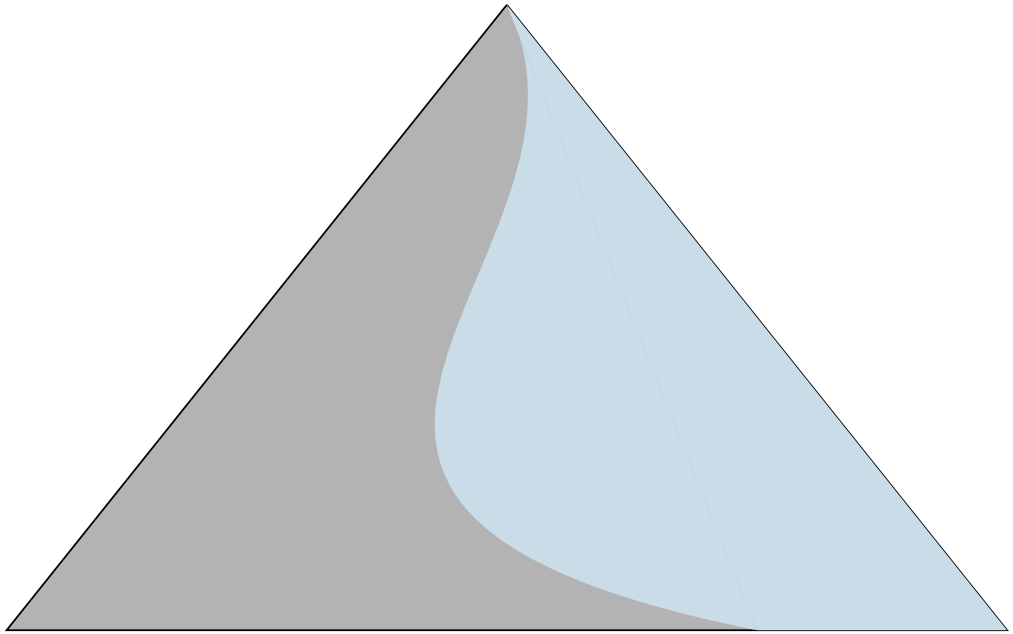

Usage:




[Beyer et al.: Conditional model checking: a technique to pass information between verifiers. FSE 2012]

# CMC Visualized

Idea: State space of program $P$



Condition describes part verified by Verifier 1 and Verifier 2 has to verify rest.

# Condition Automata

$\Phi$: assumptions on program variables

## Definition

Condition automaton $A = (Q, \Sigma, \delta, q_0, F)$ for program $P = (L, \ell_0, G, L_{err})$:

- set of states $Q$, $q_0 \in Q$ initial state,
- alphabet $\Sigma \subseteq 2^G \times \Phi$,
- transition relation $\delta \subseteq Q \times \Sigma \times Q$ and
- final states $F \subseteq Q$.

# Path Covering

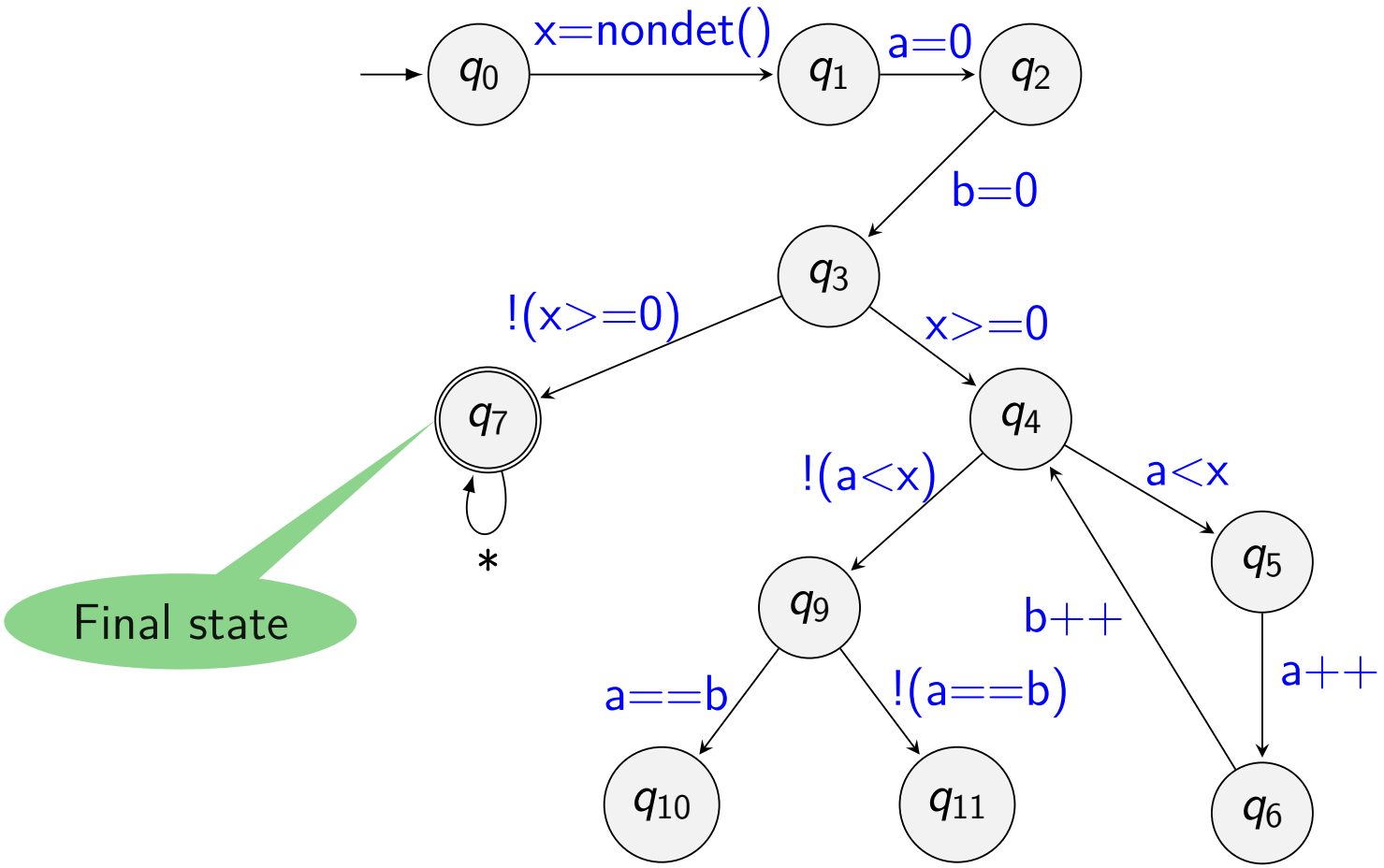
Condition automaton $A = (Q, \Sigma, \delta, q_0, F)$
program $P = (L, G, \ell_0, L_{err})$

$A$ covers a path $c_0 \xrightarrow{g_1} c_1 \xrightarrow{g_2} \ldots \xrightarrow{g_n} c_n$ of $P$ if there is a run $q_0 \xrightarrow{(G_1,\phi_1)} q_1 \xrightarrow{(G_2,\phi_2)} \ldots \xrightarrow{(G_k,\phi_k)} q_k$, $1 \leq k \leq n$, if

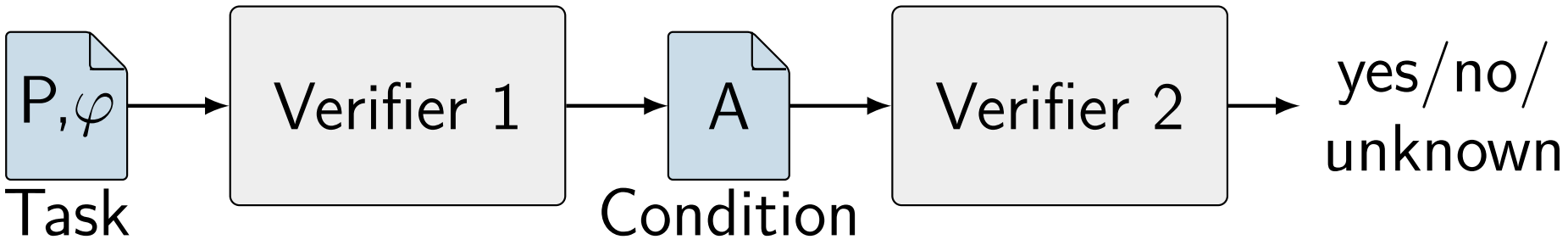
- $g_i \in G_i$,
- $c_i \models \phi_i$,
- $q_k \in F$.

# Condition Automaton of Example

Condition automaton produced by value analysis



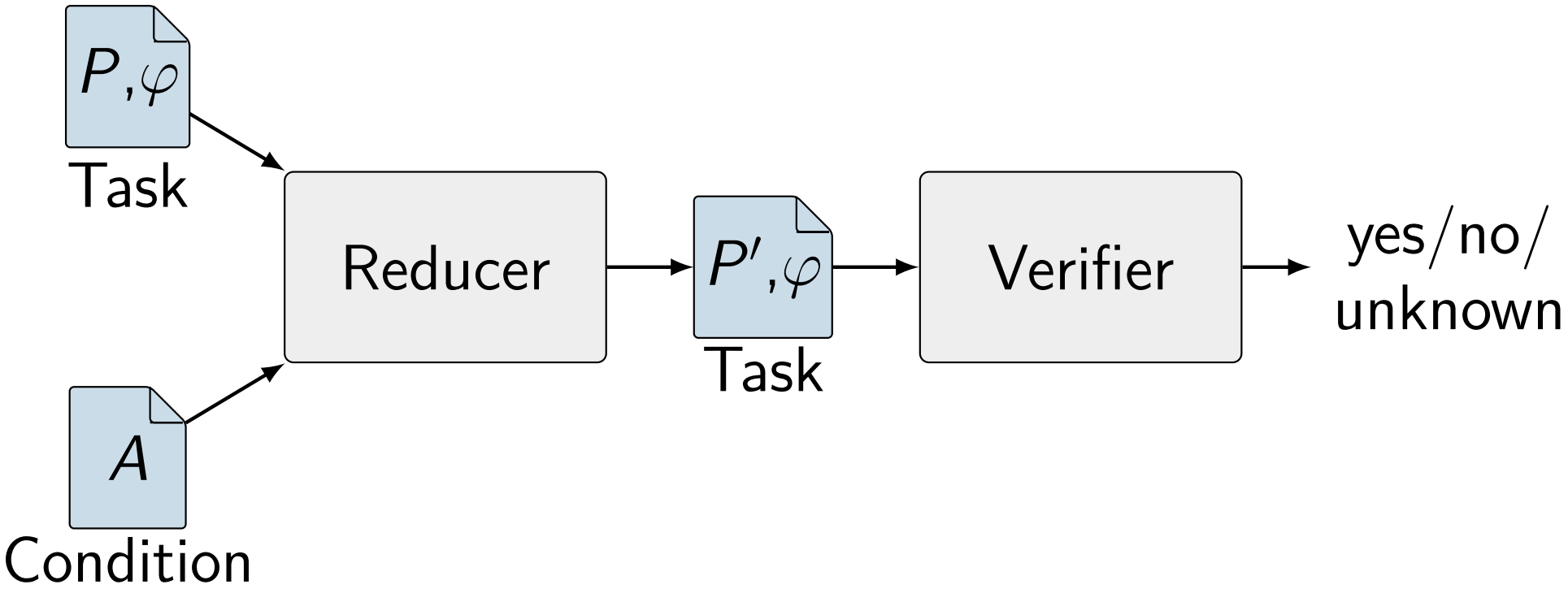
# Conditional Model Checking Revisited




What if Verifier 2 does not understand conditions?

# Reducer I

Idea:
take condition and transform it into residual program




[Beyer, Jakobs, Lemberger, Wehrheim: Reducer-based construction of conditional verifiers. ICSE 2018.]

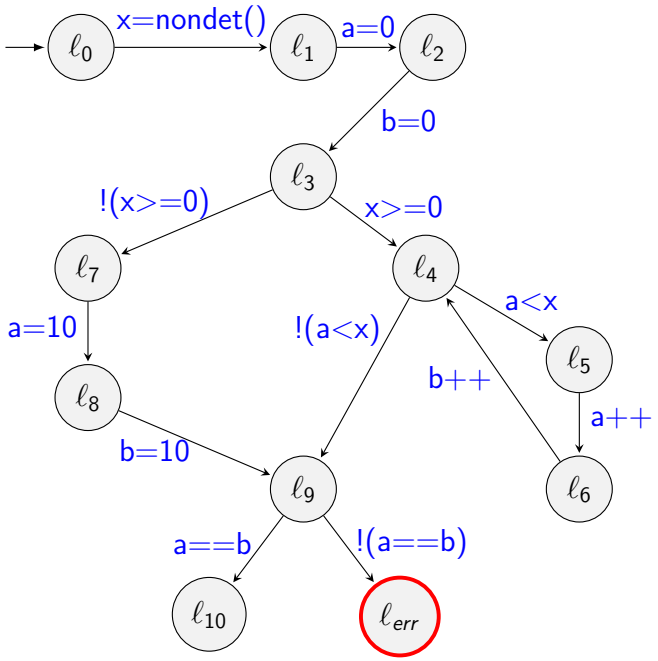
# Reducer II

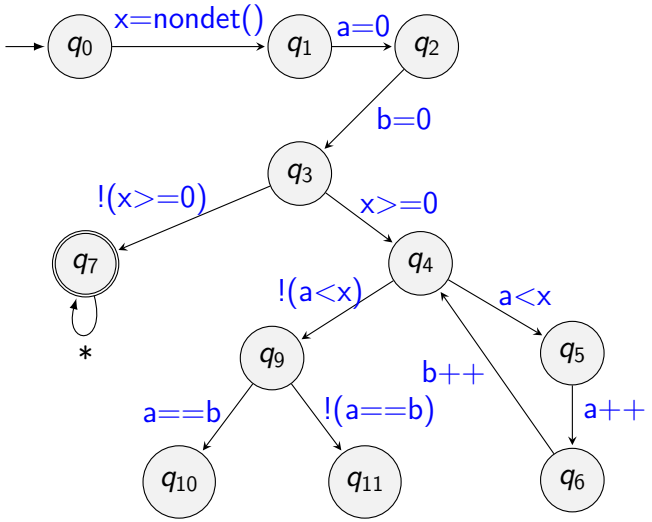
Reducer $red : (P, A) \mapsto P'$ such that

$$paths(P) \setminus covers(A) \subseteq paths(P') \subseteq paths(P)$$

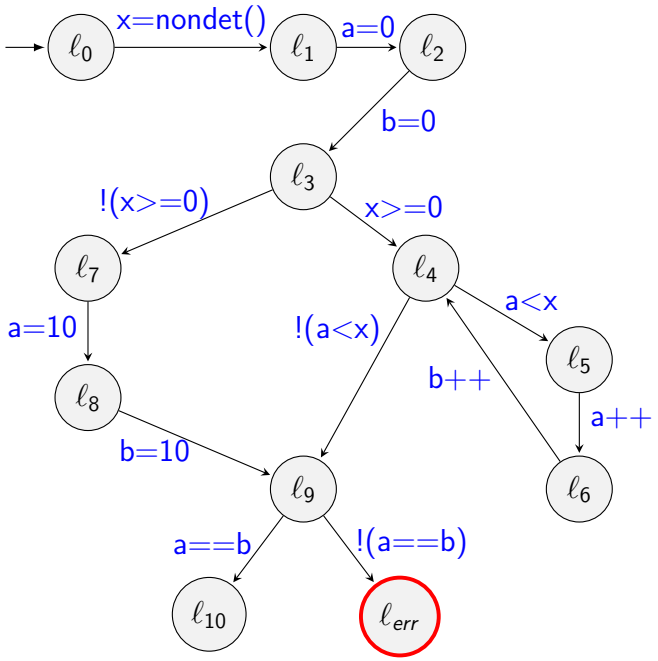
Technically:
build product of condition and program

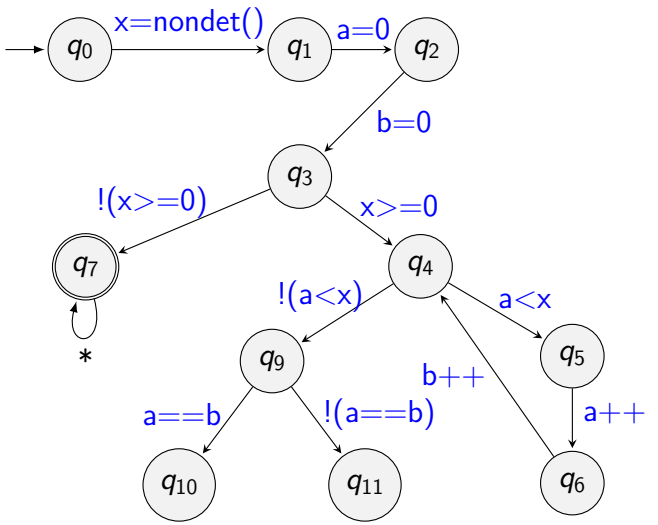
# Residual Program of Example: CFA

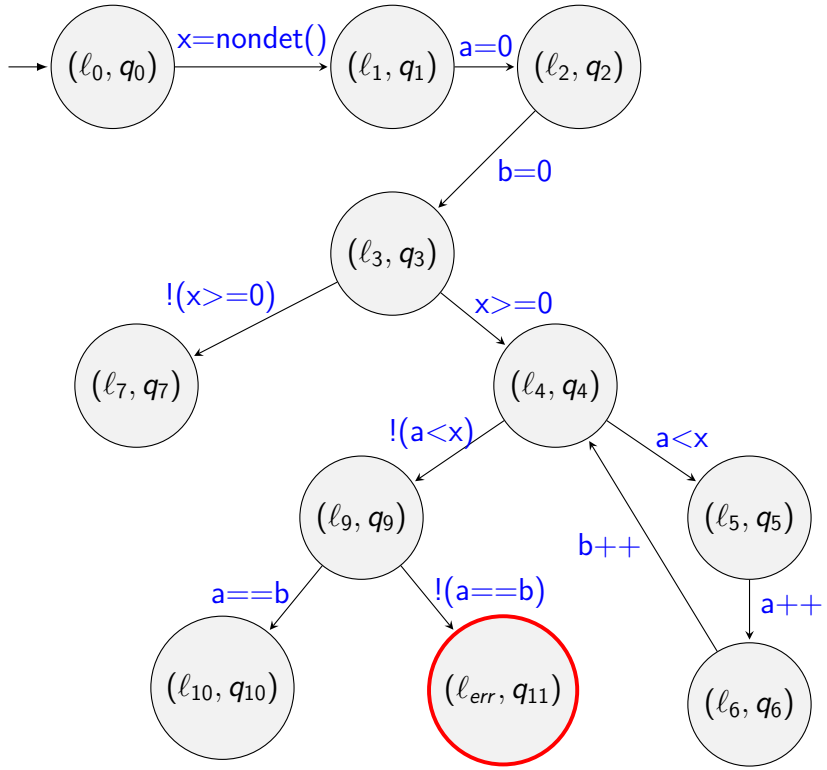


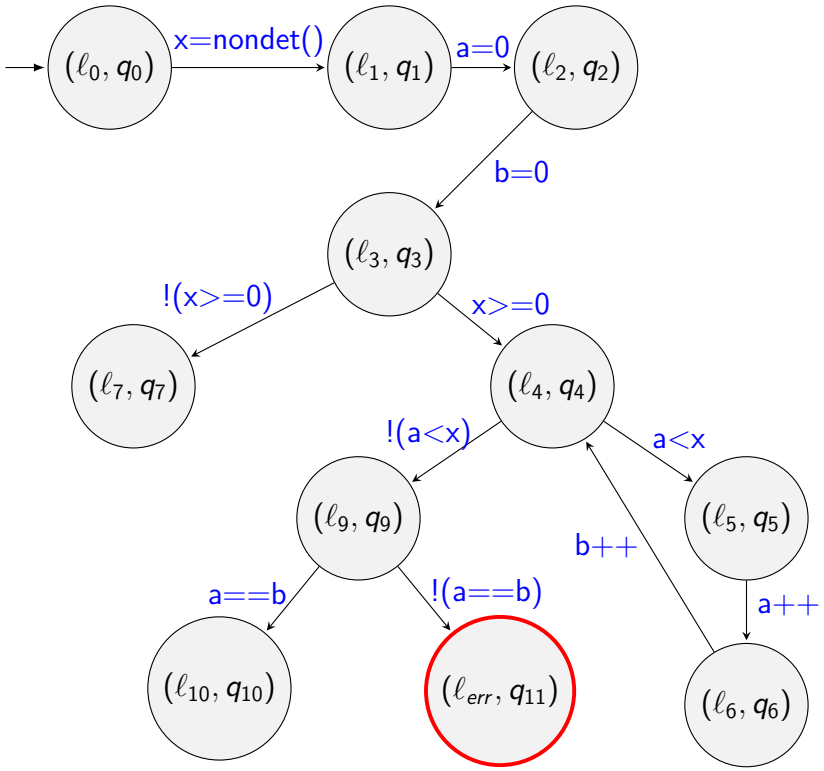
# Residual Program of Example: CFA



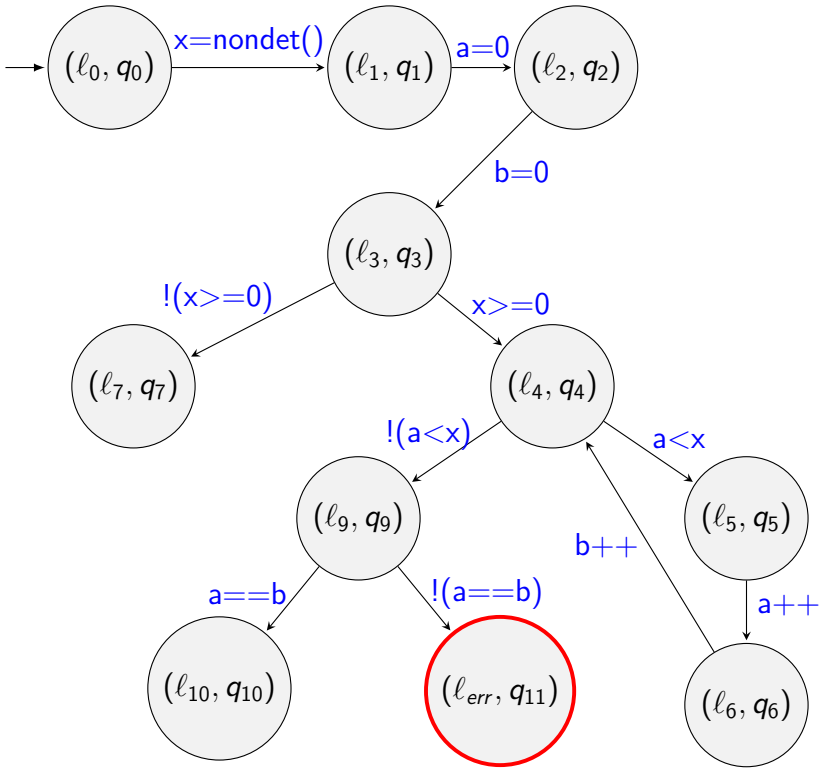



# Residual Program of Example: Code



# Residual Program of Example: Code




```
int x = nondet();
int a = 0;
int b = 0;
if (x >= 0)
    while (a < x)
        a++;
        b++;
    assert (a==b);
else
    skip
```

# Does it payoff?

Predicate Analysis + CPASeq/UAutomizer (820 tasks)

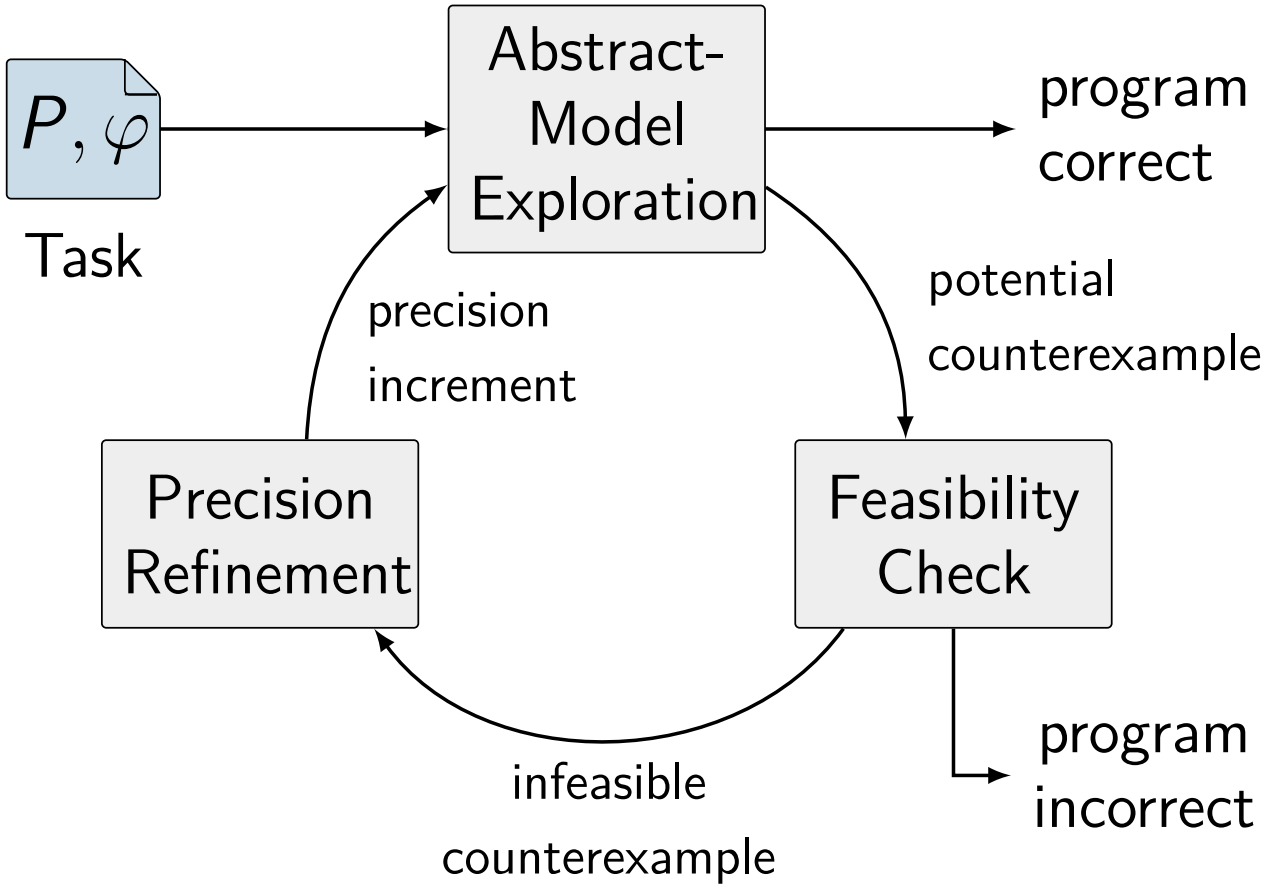
| | | correct | | incorrect | |
|---|---|---|---|---|---|
| | **overall** | **proof** | **alarm** | **proof** | **alarm** |
| **CPASeq** | 513 | 265 | 248 | 0 | 0 |
| **Pred+CPASeq** | 789 | 387 | 402 | 0 | 0 |
| **UAuto** | 238 | 170 | 68 | 4 | 3 |
| **Pred+UAuto** | 789 | 386 | 403 | 0 | 4 |

(similar results for Smack)

# Topic 2b:
# Cyclic Cooperation

# CEGAR

## CEGAR (counterexample-guided abstraction refinement)



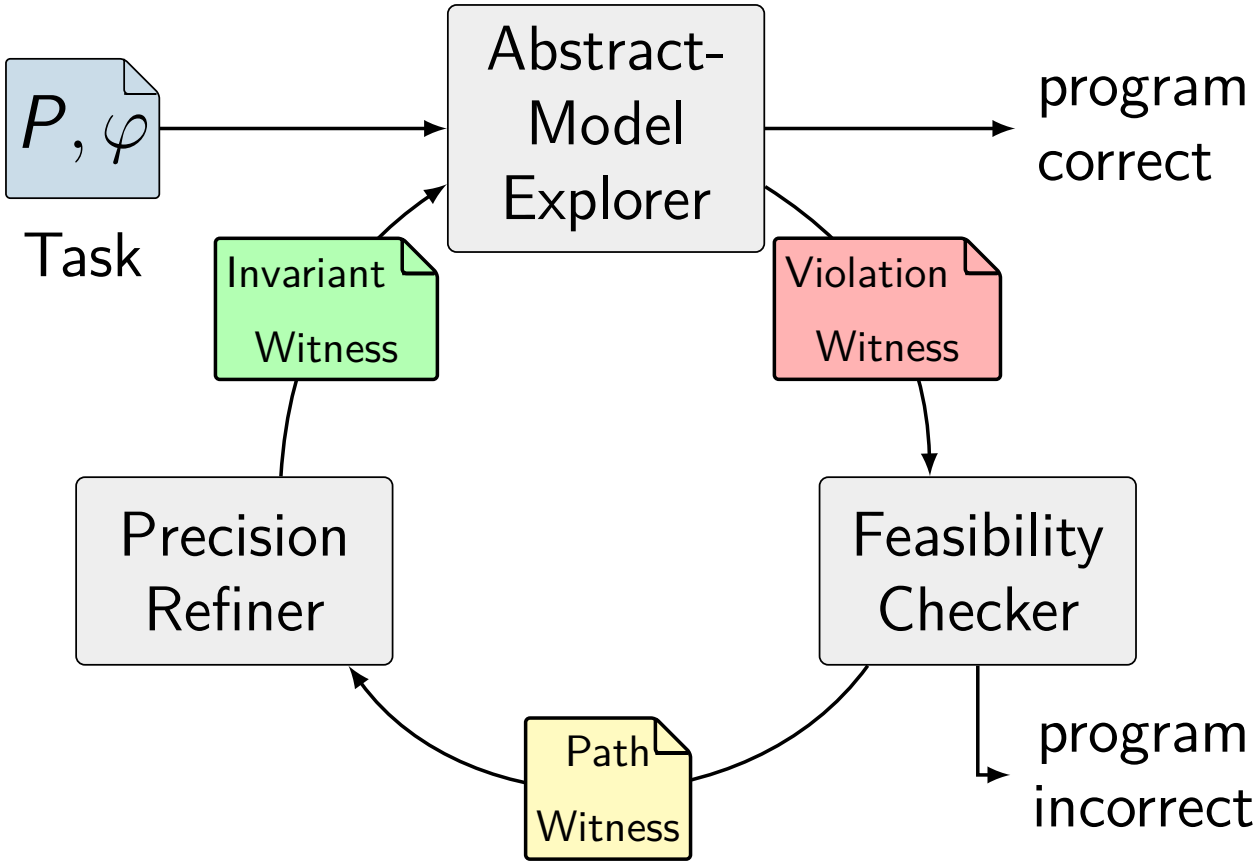
# Decomposition

Idea:

- decompose into independent components
- define interfaces
- exchange information via witnesses

$\Rightarrow$ enables usage of off-the-shelf tools

# Component-Based CEGAR



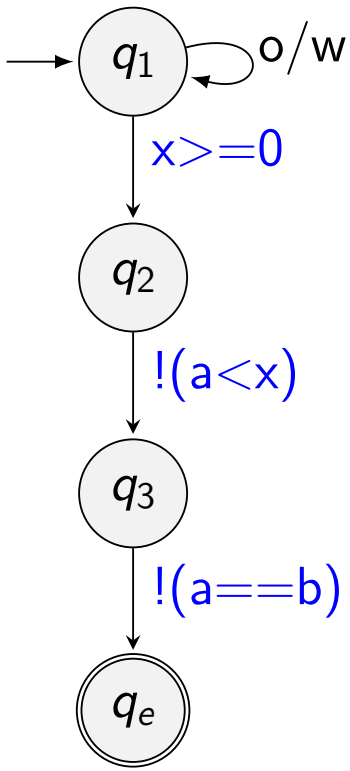

[Beyer, Haltermann, Lemberger, Wehrheim: Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR. ICSE 2022.]

# On Example I

## Program

```
int x = nondet();
int a = 0;
int b = 0;
if (x >= 0)
    while (a < x)
          a++;
          b++;
else
    a = 10;
    b = 10;
assert (a==b);
```

## Violation witness

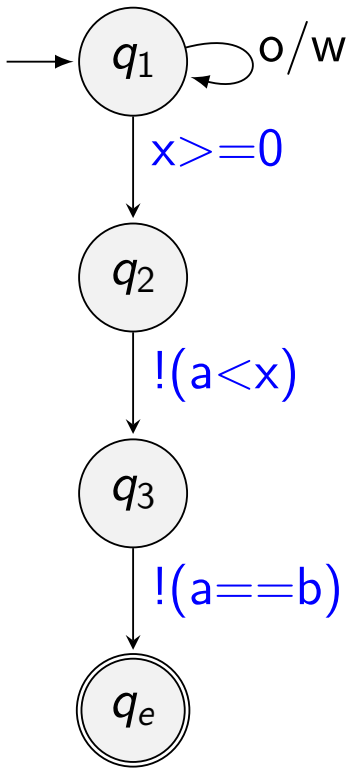


# On Example I

Program

```
int x = nondet();
int a = 0;
int b = 0;
if (x >= 0)
    while (a < x)
          a++;
          b++;
else
    a = 10;
    b = 10;
assert (a==b);
```

Violation witness



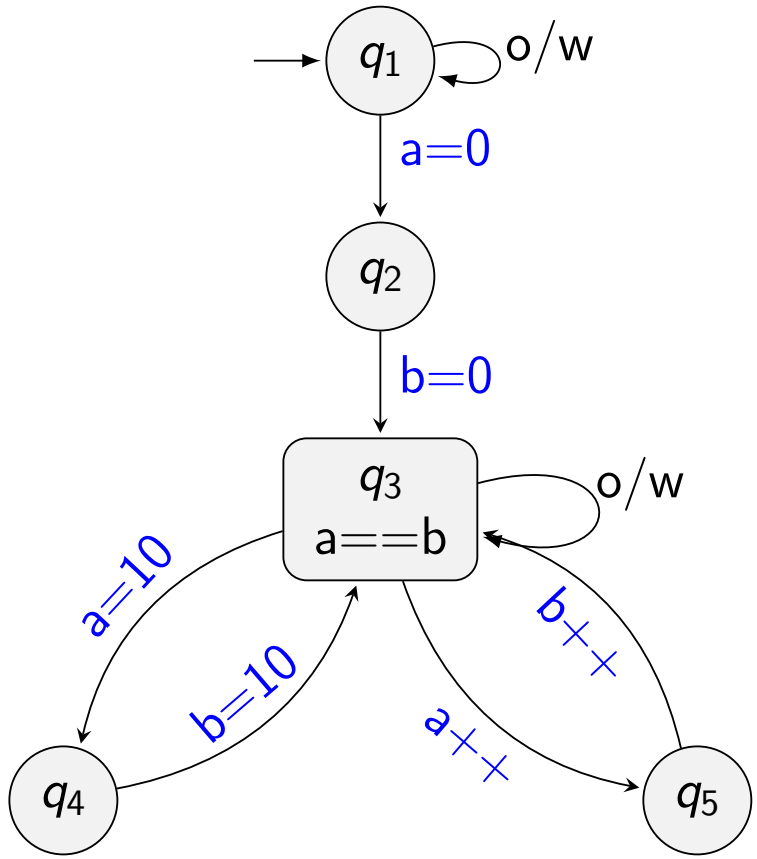

spurious

# On Example II

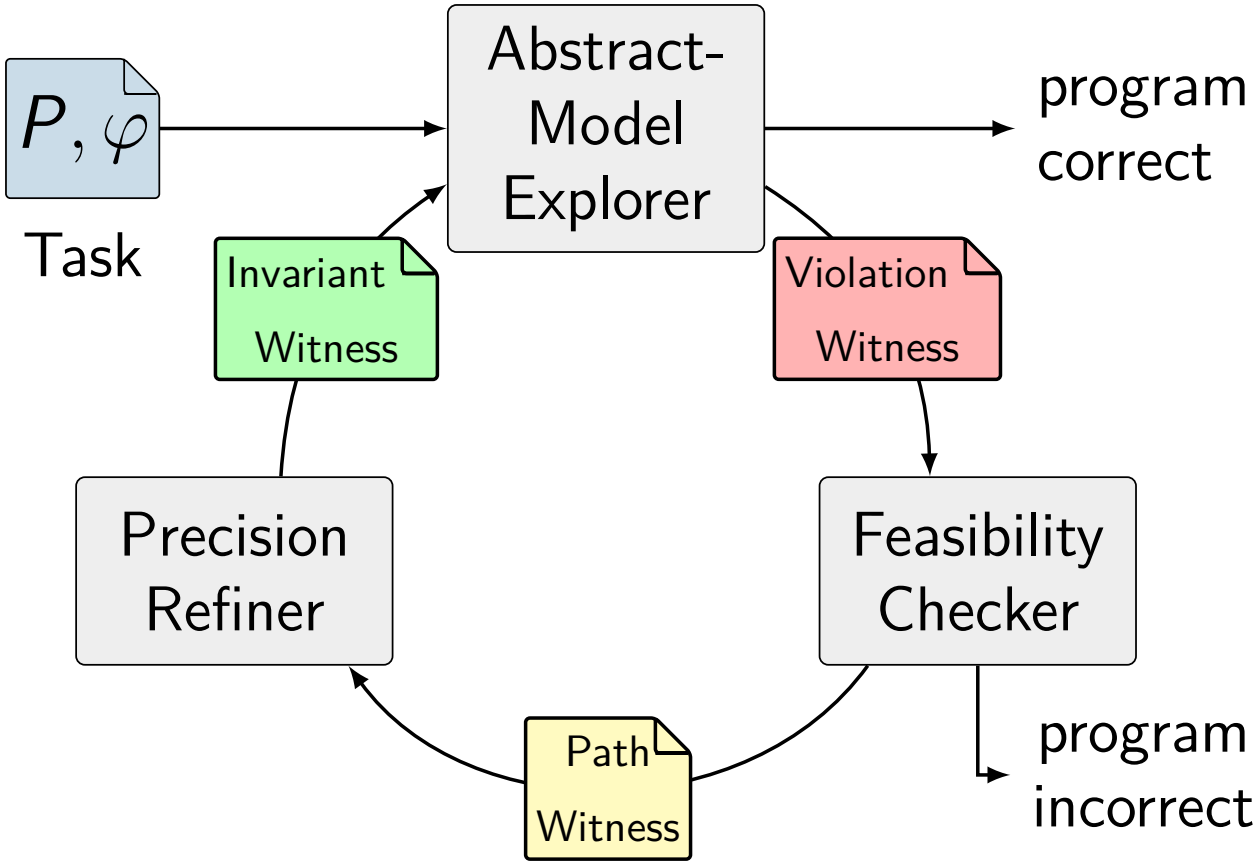
Program

```
int x = nondet();
int a = 0;
int b = 0;
if (x >= 0)
    while (a < x)
          a++;
          b++;
else
    a = 10;
    b = 10;
assert (a==b);
```

Invariant witness



# Component-Based CEGAR




[Beyer, Haltermann, Lemberger, Wehrheim: Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR. ICSE 2022.]

# Does it payoff?

Decomposed CPAchecker predicate analysis: (8347 tasks)

| | | **correct** | | **incorrect** | |
|---|---|---|---|---|---|
| | **overall** | **proof** | **alarm** | **proof** | **alarm** |
| **Pred** | 3769 | 2556 | 1213 | 3 | 9 |
| **PredComp** | 3524 | 2450 | 1074 | 0 | 3 |
| **PredCompWit** | 2854 | 2110 | 744 | 0 | 1 |

# Does it payoff?

Decomposed CPAchecker predicate analysis: (8347 tasks)

| | | **correct** | | **incorrect** | |
|---|---|---|---|---|---|
| | **overall** | **proof** | **alarm** | **proof** | **alarm** |
| **Pred** | 3769 | 2556 | 1213 | 3 | 9 |
| **PredComp** | 3524 | 2450 | 1074 | 0 | 3 |
| **PredCompWit** | 2854 | 2110 | 744 | 0 | 1 |

Reasons:

- exchange of information takes times
- standardized formats $\Rightarrow$ loss of information

# Some Conclusions

Use cooperative verification when needed!

# Some Conclusions

Use cooperative verification when needed!

Wish to tool developers:

Provide computed information in witnesses

# Summary

Witnesses everywhere in software verification!

As ...

- ... proofs
- ... conditions
- ... counterexamples
- ... precisions