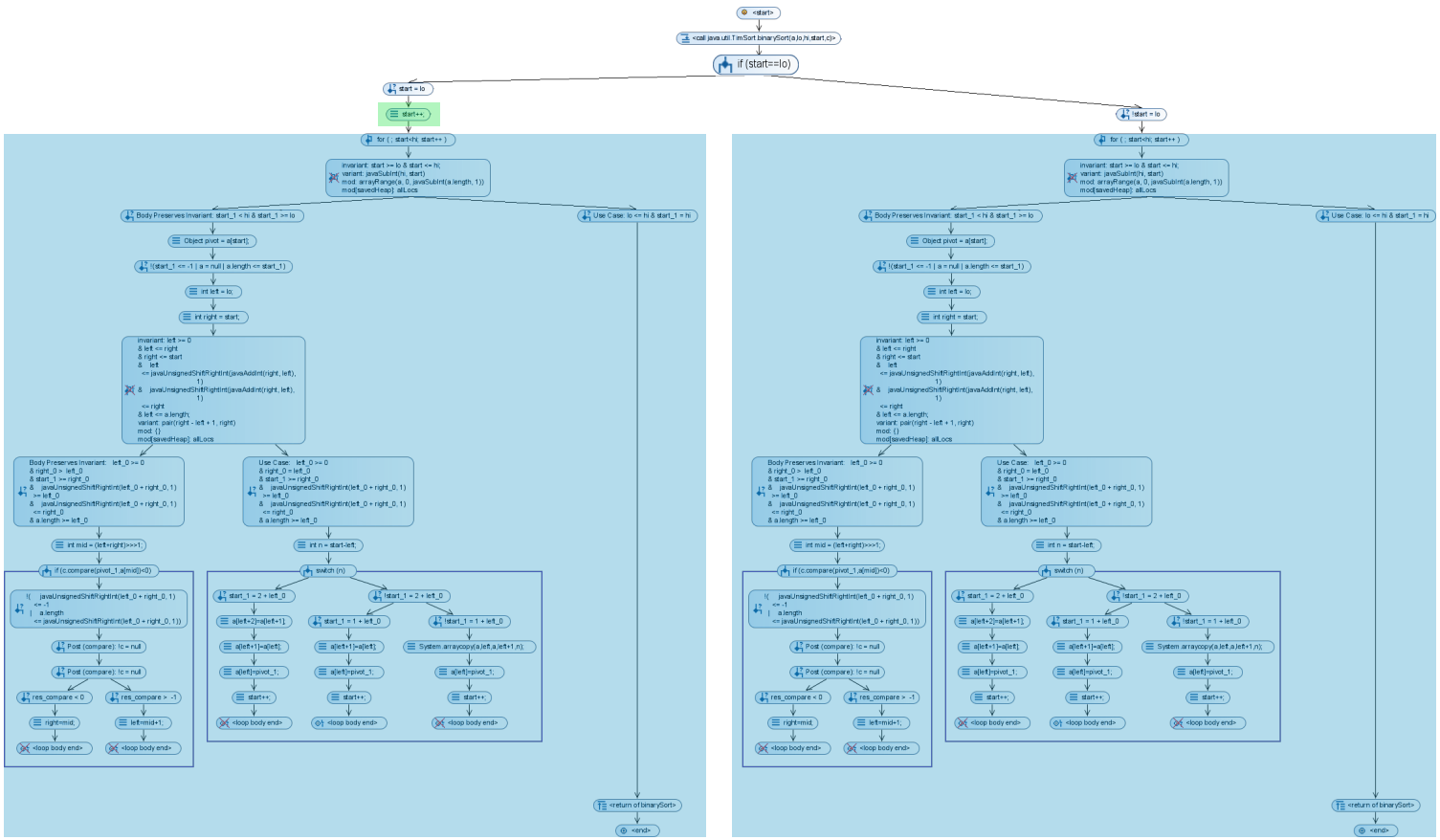
A General Lattice Model for Merging Symbolic Execution Branches

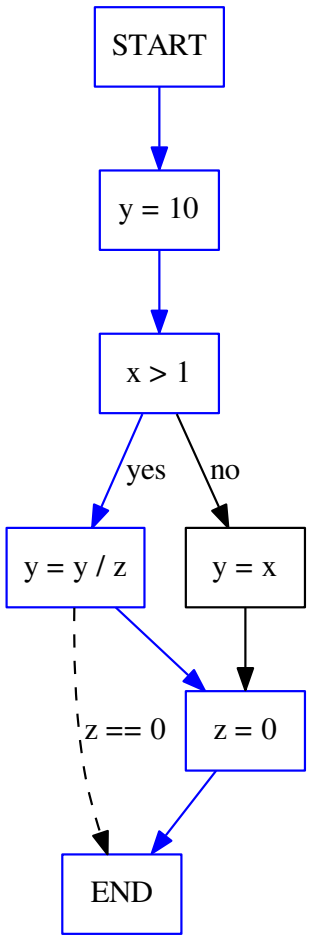
Dominic Scheurer

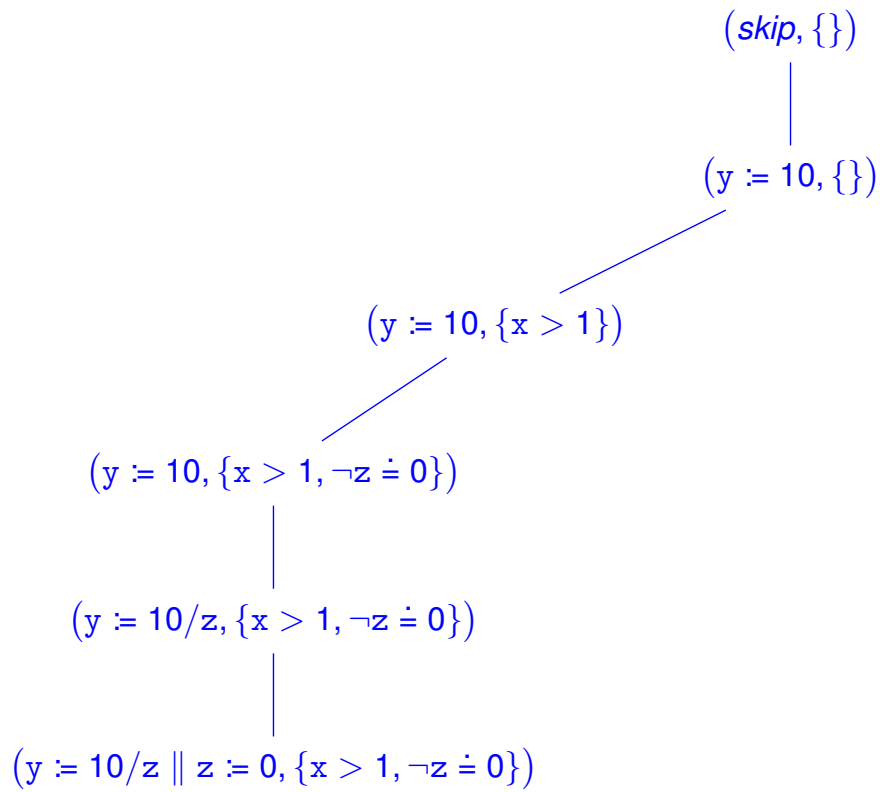
scheurer@cs.tu-darmstadt.de

TECHNISCHE UNIVERSITÄT DARMSTADT

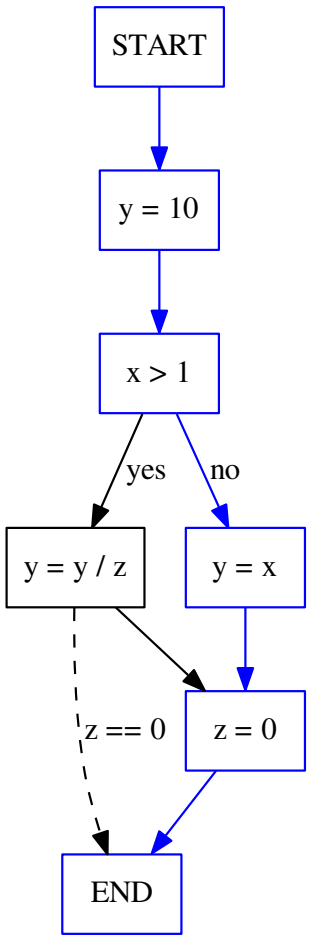


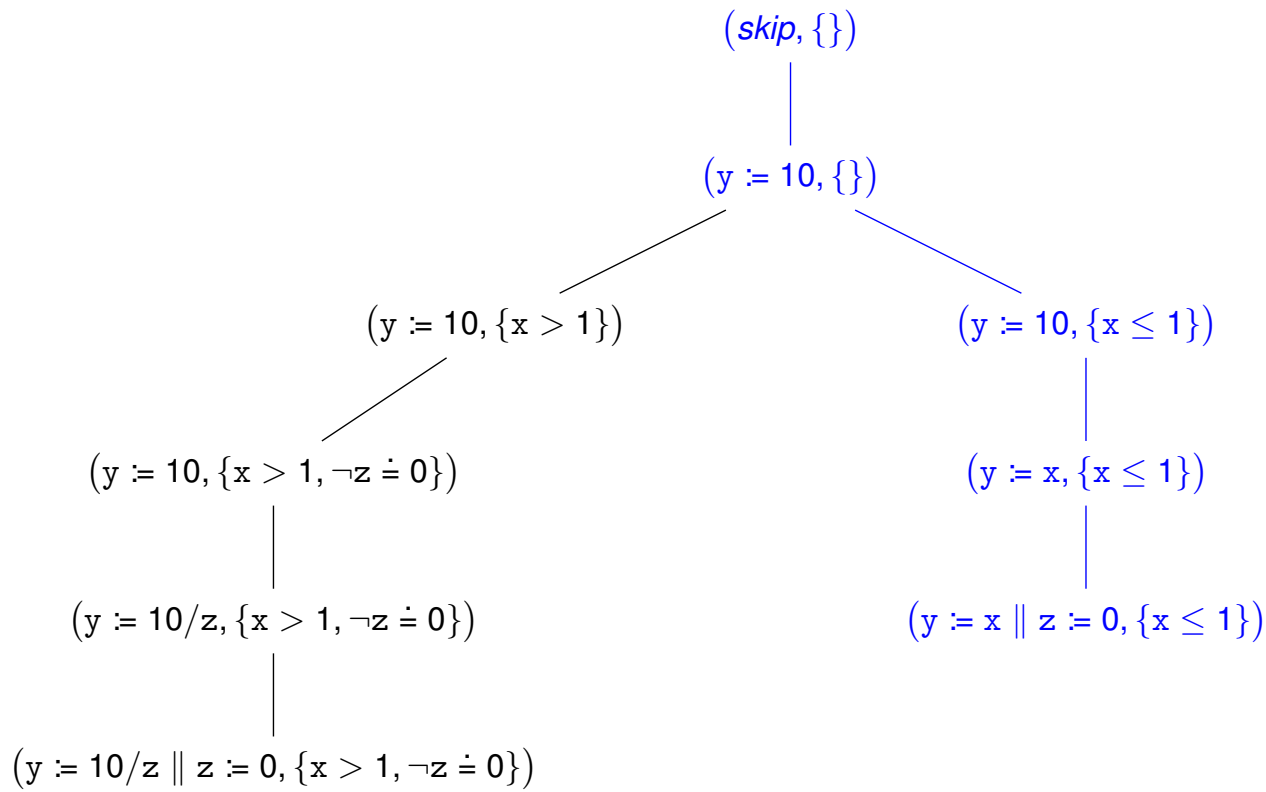
Symbolic Execution follows all possible execution paths





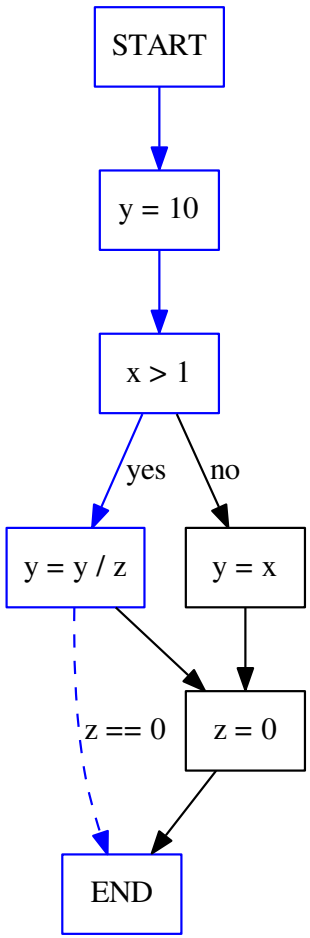
Symbolic Execution follows all possible execution paths

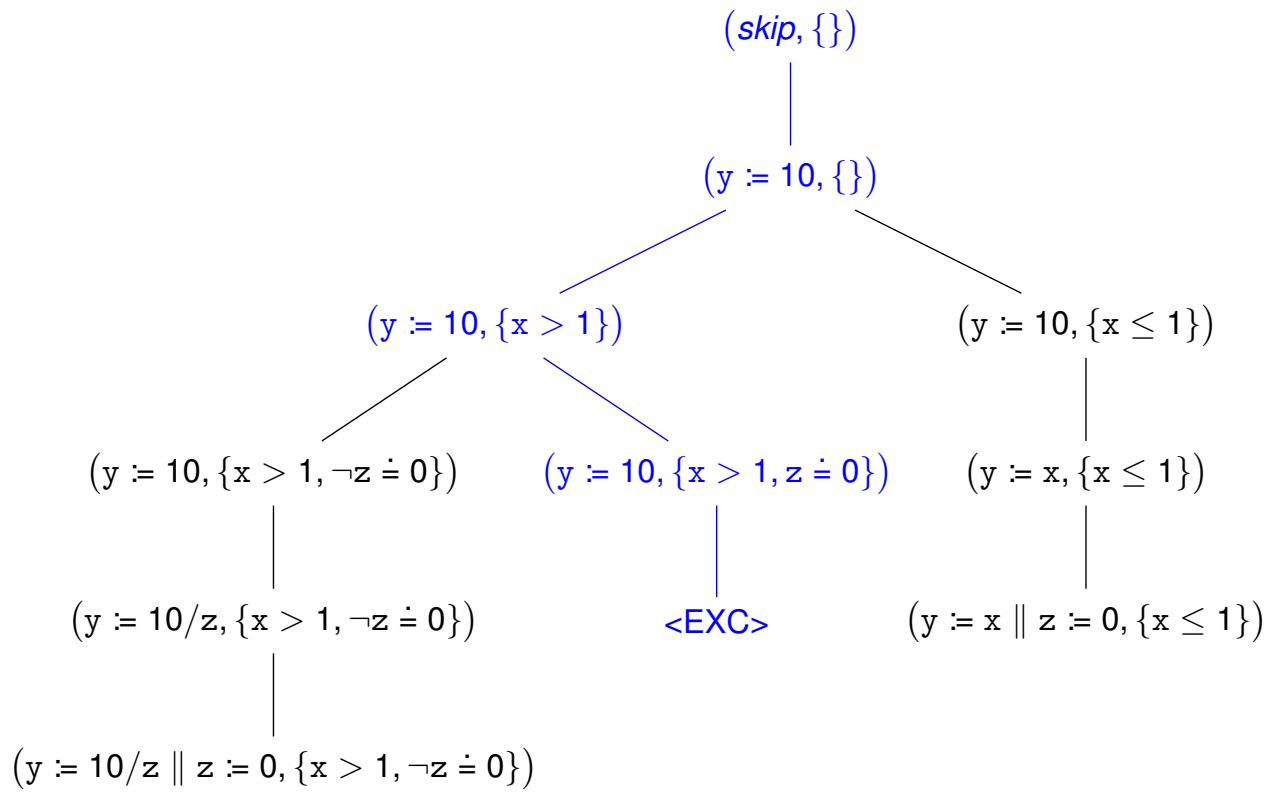




Symbolic Execution follows all possible execution paths

TECHNISCHE UNIVERSITÄT DARMSTADT





Software Engineering Group

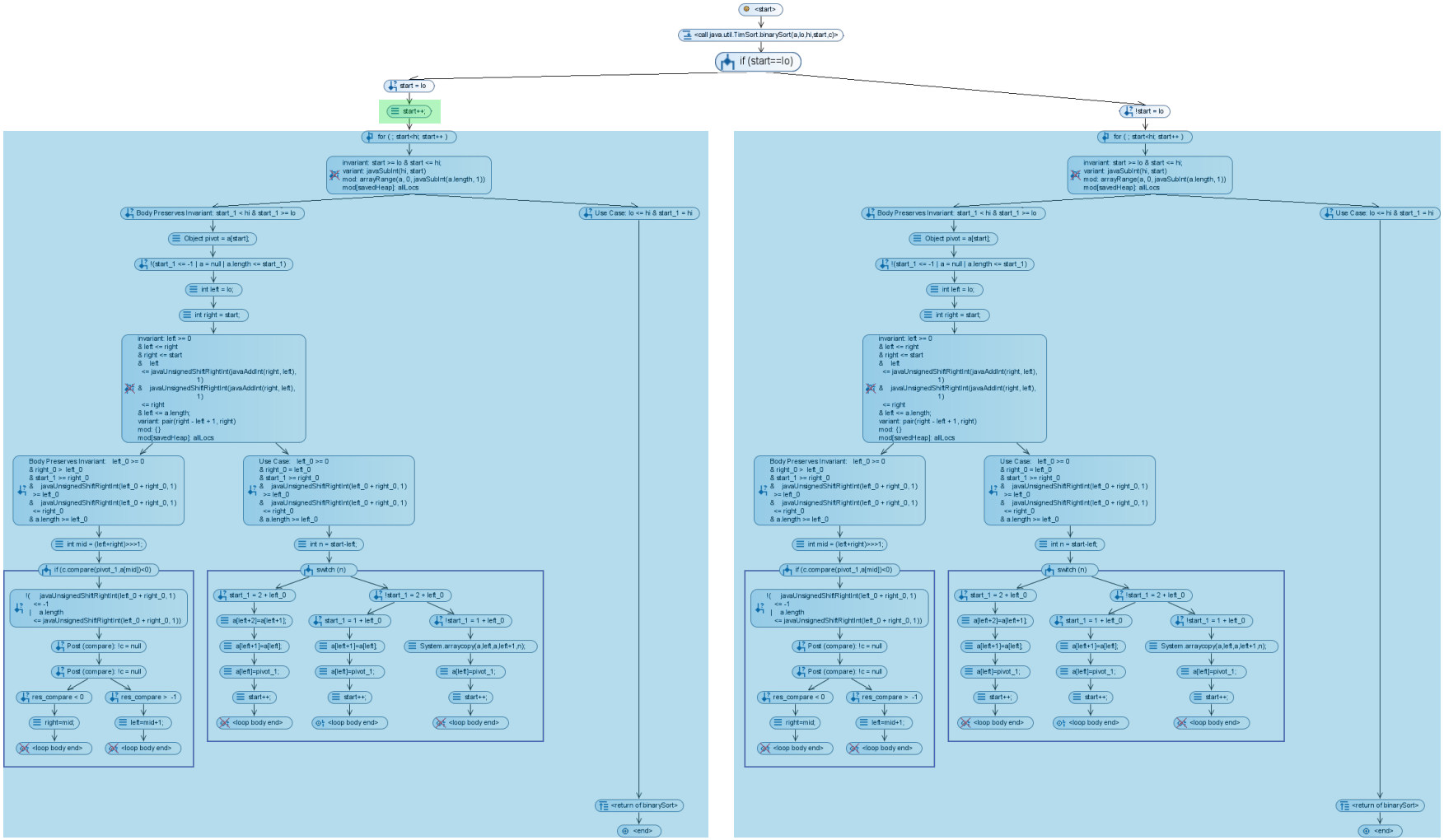
Beware of the *Path Explosion Problem*! Example: TimSort.binarySort

TECHNISCHE UNIVERSITÄT DARMSTADT

```
private static void binarySort(Object[] a, /* ... */) {
    assert lo <= start && start <= hi;

    if (start == lo)
        start++;

    for ( ; start < hi; start++) {
        // ...
        while (left < right) { /* ... */ }
        // ...

        int n = start - left;
        switch (n) { /* ... */ }
        a[left] = pivot;
    }
}
```

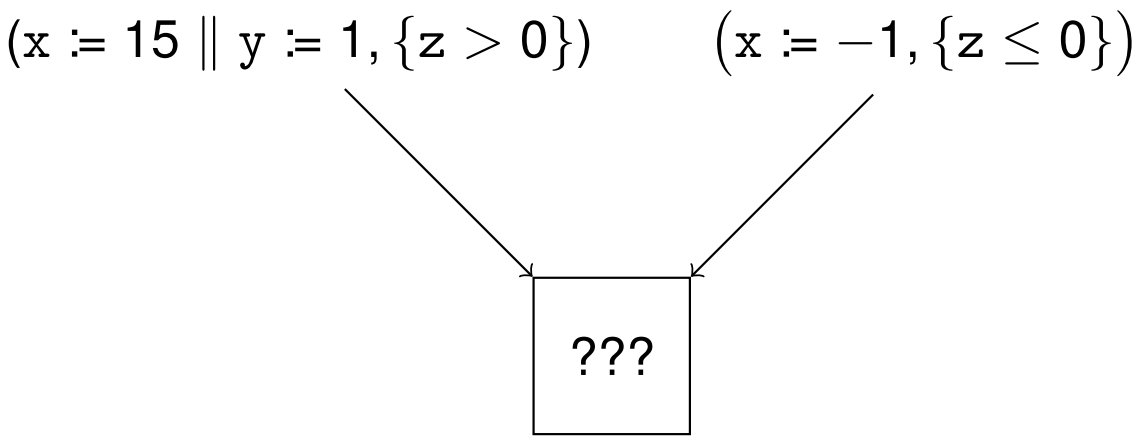
Beware of the *Path Explosion Problem*!
TimSort.binarySort – Symbolic Execution Tree

TECHNISCHE UNIVERSITÄT DARMSTADT



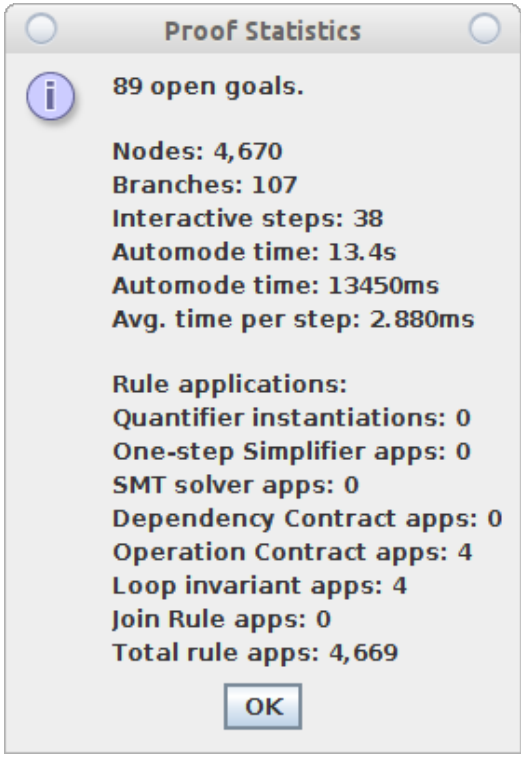
Software Engineering Group

Goals



TimSort.binarySort – Proof statistics after SE

TECHNISCHE UNIVERSITÄT DARMSTADT

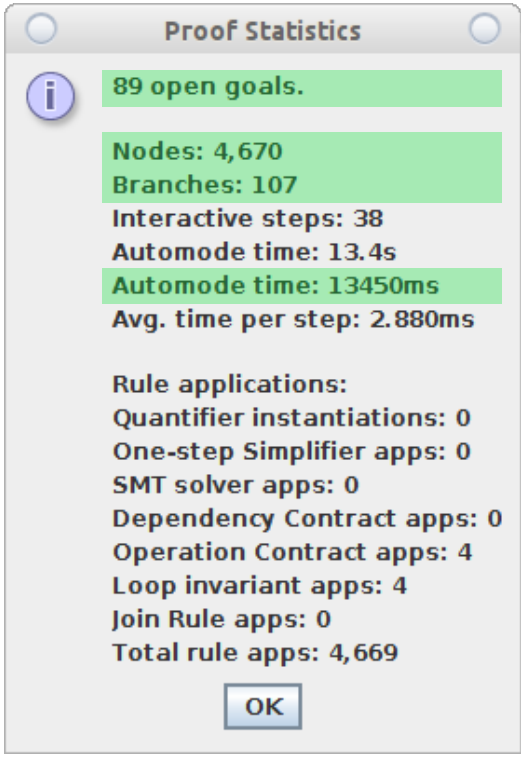


Statistics after finishing Symbolic Execution

Software Engineering Group

TimSort.binarySort – Proof statistics after SE

TECHNISCHE UNIVERSITÄT DARMSTADT



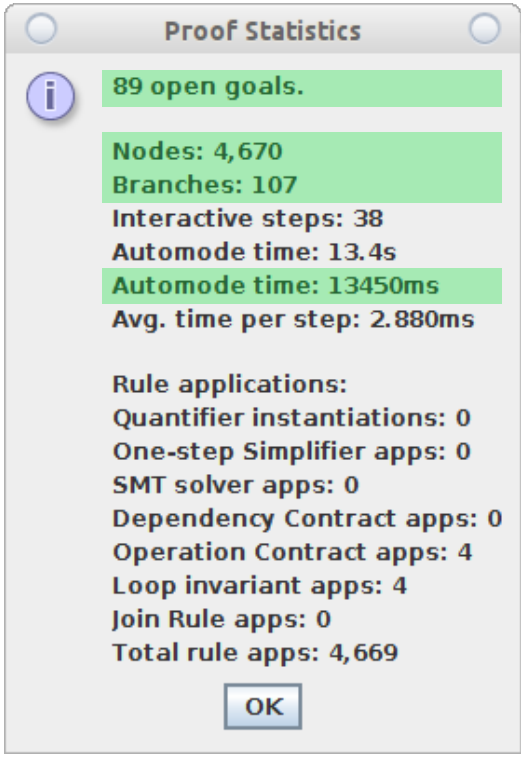
Statistics after finishing Symbolic Execution

Statistics after finishing Symbolic Execution *with one merge*

Software Engineering Group

TimSort.binarySort – Proof statistics after SE

TECHNISCHE UNIVERSITÄT DARMSTADT



Statistics after finishing Symbolic Execution

Statistics after finishing Symbolic Execution *with one merge*

Software Engineering Group

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Preliminary Considerations

Java Dynamic Logic (Java DL): A program logic for proving functional properties about Java programs

TECHNISCHE UNIVERSITÄT DARMSTADT

Syntax

$$\mathtt{x} = \mathit{true} \rightarrow \underbrace{\left\{ \mathtt{y} := c \parallel \underbrace{\mathtt{x} := \mathit{if}\,(\varphi)\,\mathit{then}\,(1)\,\mathit{else}\,(-c)}_{\text{elementary update}} \right\}}_{\text{parallel update}} \underbrace{\langle \mathtt{y}{+}{+}; \mathtt{return};\rangle\, \mathtt{y} > 0}_{\text{Modality}}$$

Semantics

$$\mathit{val}\left(K, \sigma; \{\mathtt{y} := 1\}\right) = \sigma_{[\mathtt{y} \mapsto 1]}$$

$$\mathit{val}\,(K, \sigma; \exists x; x > 0) = \mathit{tt} \iff (K, \sigma) \models \exists x; x > 0$$

Sequent Calculus

$$\frac{\Delta \implies \Gamma, \{\mathtt{y} := 1\}\,\langle p \rangle\,\varphi}{\Delta \implies \Gamma, \langle \mathtt{y} = 1;\ p \rangle\,\varphi}\ \text{assignment}$$

Software Engineering Group

Java Dynamic Logic (Java DL): A program logic for proving functional properties about Java programs

TECHNISCHE UNIVERSITÄT DARMSTADT

Syntax

$$\mathtt{x} = \mathit{true} \rightarrow \underbrace{\left\{ \mathtt{y} := c \parallel \underbrace{\mathtt{x} := \mathit{if}\,(\varphi)\,\mathit{then}\,(1)\,\mathit{else}\,(-c)}_{\text{elementary update}} \right\}}_{\text{parallel update}} \underbrace{\langle \mathtt{y}{+}{+}; \mathtt{return}; \rangle\, \mathtt{y} > 0}_{\text{Modality}}$$

Semantics

$$\mathit{val}\left(K, \sigma; \{\mathtt{y} := 1\}\right) = \sigma_{[\mathtt{y} \mapsto 1]}$$

$$\mathit{val}\,(K, \sigma; \exists x; x > 0) = \mathit{tt} \iff (K, \sigma) \models \exists x; x > 0$$

Sequent Calculus

$$\frac{\Delta \Longrightarrow \Gamma, \{\mathtt{y} := 1\}\,\langle p \rangle\,\varphi}{\Delta \Longrightarrow \Gamma, \langle \mathtt{y} = 1;\ p \rangle\,\varphi}\ \text{assignment}$$

Software Engineering Group

Java Dynamic Logic (Java DL): A program logic for proving functional properties about Java programs

TECHNISCHE UNIVERSITÄT DARMSTADT

Syntax

$$\mathtt{x} = \mathit{true} \rightarrow \underbrace{\left\{ \mathtt{y} := c \parallel \underbrace{\mathtt{x} := \mathit{if}\,(\varphi)\,\mathit{then}\,(1)\,\mathit{else}\,(-c)}_{\text{elementary update}} \right\}}_{\text{parallel update}} \underbrace{\langle \mathtt{y}{+}{+}; \mathtt{return}; \rangle\, \mathtt{y} > 0}_{\text{Modality}}$$

Semantics

$$\mathit{val}\left(K, \sigma; \{\mathtt{y} := 1\}\right) = \sigma_{[\mathtt{y} \mapsto 1]}$$

$$\mathit{val}\,(K, \sigma; \exists x; x > 0) = \mathit{tt} \iff (K, \sigma) \models \exists x; x > 0$$

Sequent Calculus

$$\frac{\Delta \Longrightarrow \Gamma, \{\mathtt{y} := 1\}\,\langle p \rangle\,\varphi}{\Delta \Longrightarrow \Gamma, \langle \mathtt{y} = 1; \; p \rangle\,\varphi}\ \text{assignment}$$

Software Engineering Group

An SE state is a triple

corresponding to a Java DL sequent

TECHNISCHE UNIVERSITÄT DARMSTADT

SE State $(\underbrace{\mathtt{x} := 15 \parallel \mathtt{y} := 1}_{U}, \underbrace{\{\mathtt{z} > 0, !(\mathtt{o} \doteq \mathit{null})\}}_{C}, \underbrace{\langle p \rangle \varphi}_{\text{PC}})$

Symbolic State U: current value of program variables

Path Condition C: (Set of) constraints leading to the current execution path

Program Counter: remaining program to execute p and post condition φ

Sequent $\bigwedge C \Longrightarrow \{U\} \langle p \rangle \varphi$

An SE state is a triple corresponding to a Java DL sequent

TECHNISCHE UNIVERSITÄT DARMSTADT

SE State $(\underbrace{\mathtt{x} := 15 \parallel \mathtt{y} := 1}, \quad \underbrace{\{\mathtt{z} > 0, !(\mathtt{o} \doteq \mathit{null})\}}, \quad \underbrace{\langle p \rangle \varphi} \quad)$

Symbolic State U: current value of program variables

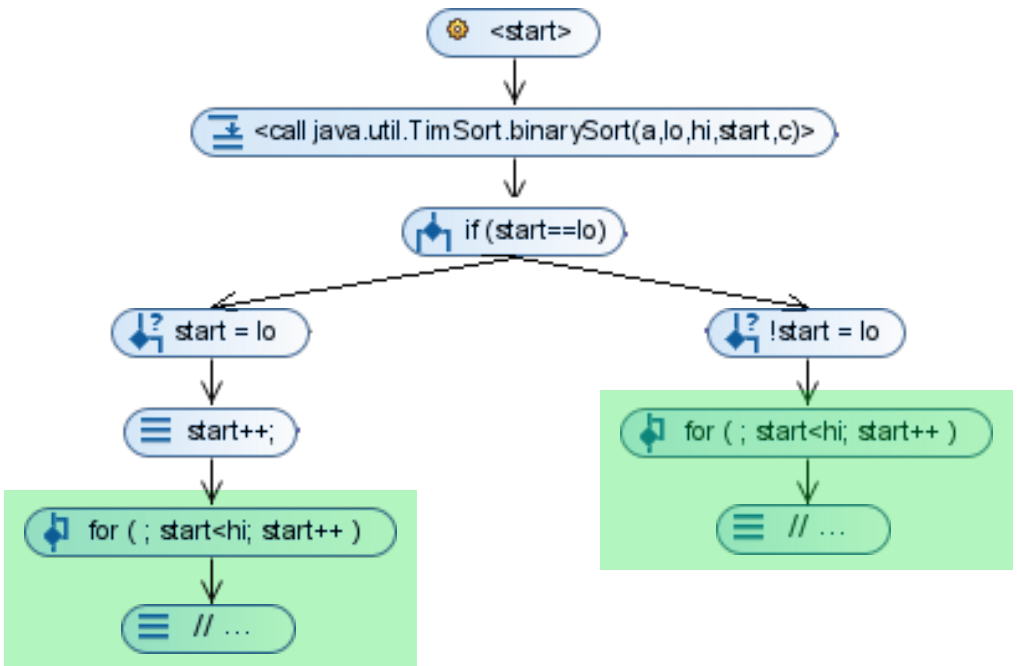
Path Condition C: (Set of) constraints leading to the current execution path

Program Counter: remaining program to execute p and post condition φ

Sequent $\bigwedge C \implies \{U\} \langle p \rangle \varphi$

Software Engineering Group

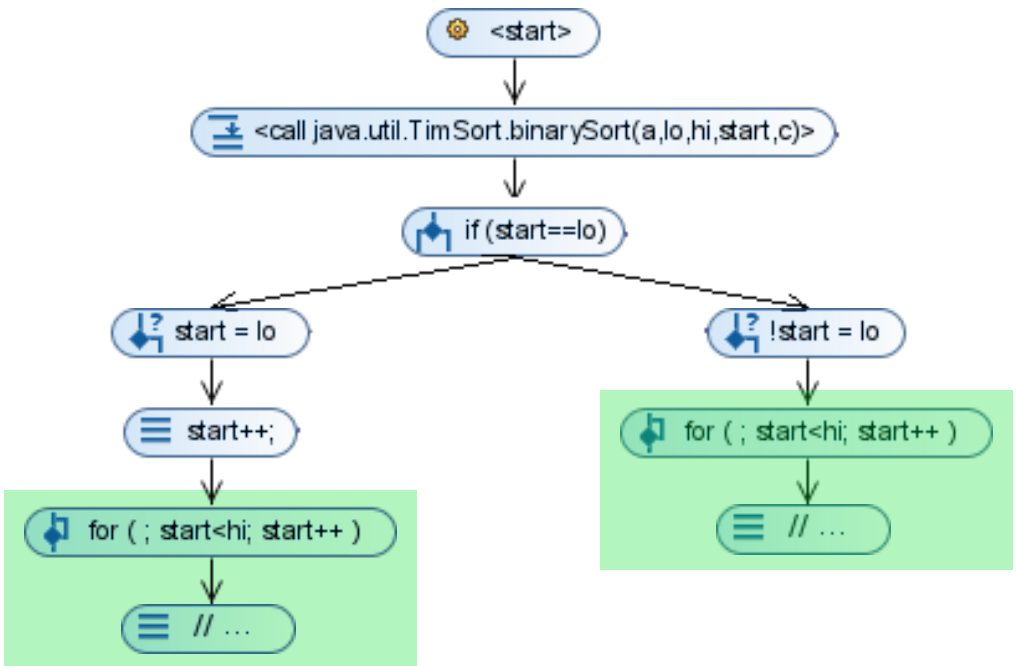
Only choose states with the *same program counter*



(U_1, C_1, φ) (U_2, C_2, φ)

(U^*, C^*, φ)

Only choose states with the *same program counter*



(U_1, C_1, φ) (U_2, C_2, φ)

(U^*, C^*, φ)

TECHNISCHE
UNIVERSITÄT
DARMSTADT

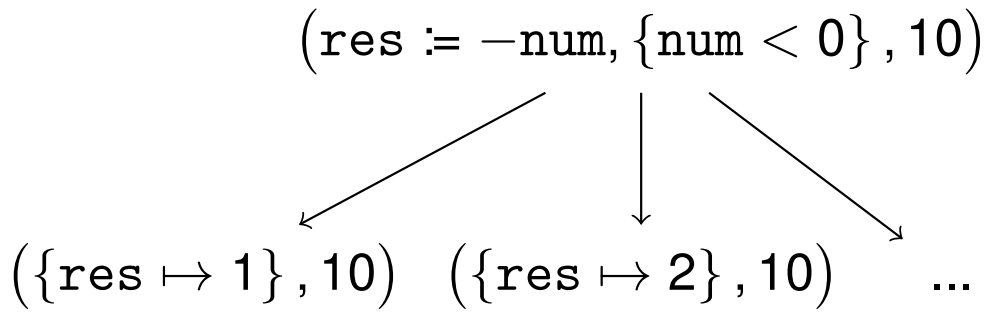
Merging Branches in SE Trees: The formal framework

Software Engineering Group

An SE state represents many *concrete states...*

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   public int abs(int num) {
2       int res;
3
4       if (num < 0) {
5           res = -num;
6       } else {
7           res = num;
8       }
9
10      return res;
11  }
```



Software Engineering Group

...and some are “weaker” than others

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

The *concretization function concr* maps an SE state $s = (U, C, \varphi)$ to a set of concrete states, where

$$concr(s) := \left\{ (\sigma', \varphi) : \underbrace{\sigma' = val(K, \sigma; U) \wedge (K, \sigma) \models C}_{\text{All values satisfying the path condition}} \right\}$$

Definition

Let s_1, s_2 be two SE states. We say that s_2 is *weaker* than s_1 (“$s_1 \lesssim s_2$”) if and only if $concr(s_1) \subseteq concr(s_2)$.

...and some are “weaker” than others

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

The *concretization function concr* maps an SE state $s = (U, C, \varphi)$ to a set of concrete states, where

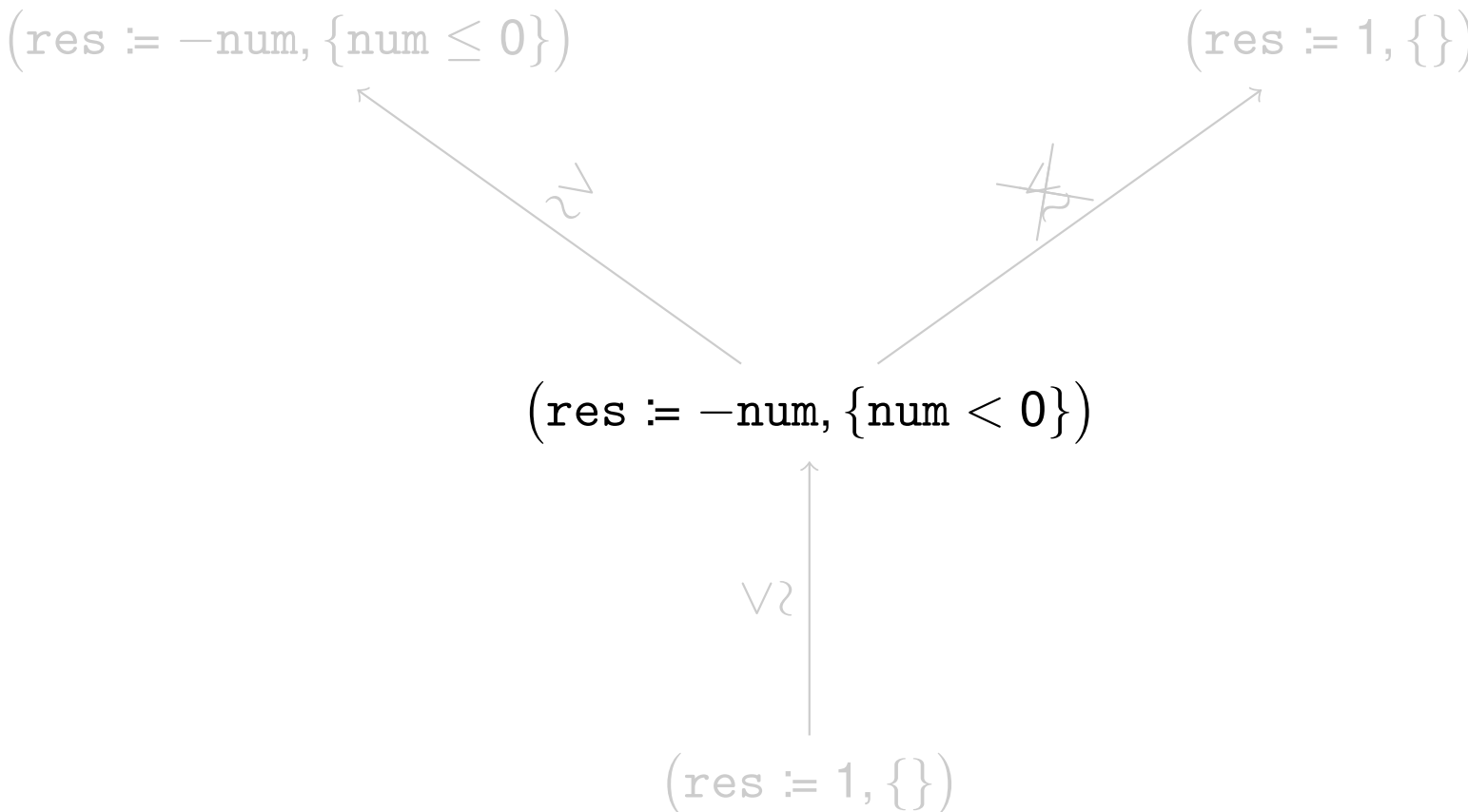
$$concr(s) := \left\{ (\sigma', \varphi) : \underbrace{\sigma' = val(K, \sigma; U) \wedge (K, \sigma) \models C}_{\text{All values satisfying the path condition}} \right\}$$

Definition

Let s_1, s_2 be two SE states. We say that s_2 is *weaker* than s_1 (“$s_1 \lesssim s_2$”) if and only if $concr(s_1) \subseteq concr(s_2)$.

Some simple examples for weakening

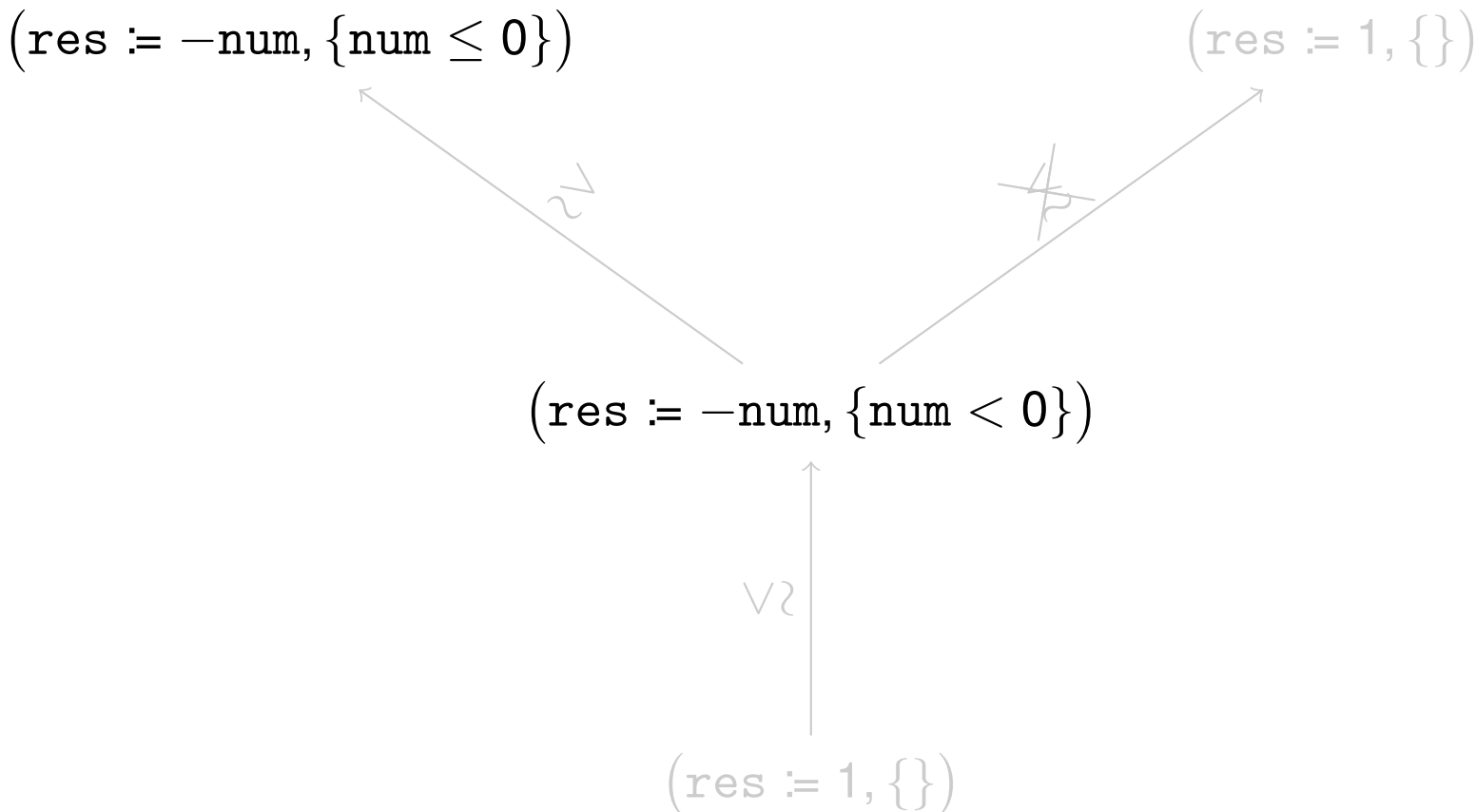
TECHNISCHE UNIVERSITÄT DARMSTADT



Software Engineering Group

Some simple examples for weakening

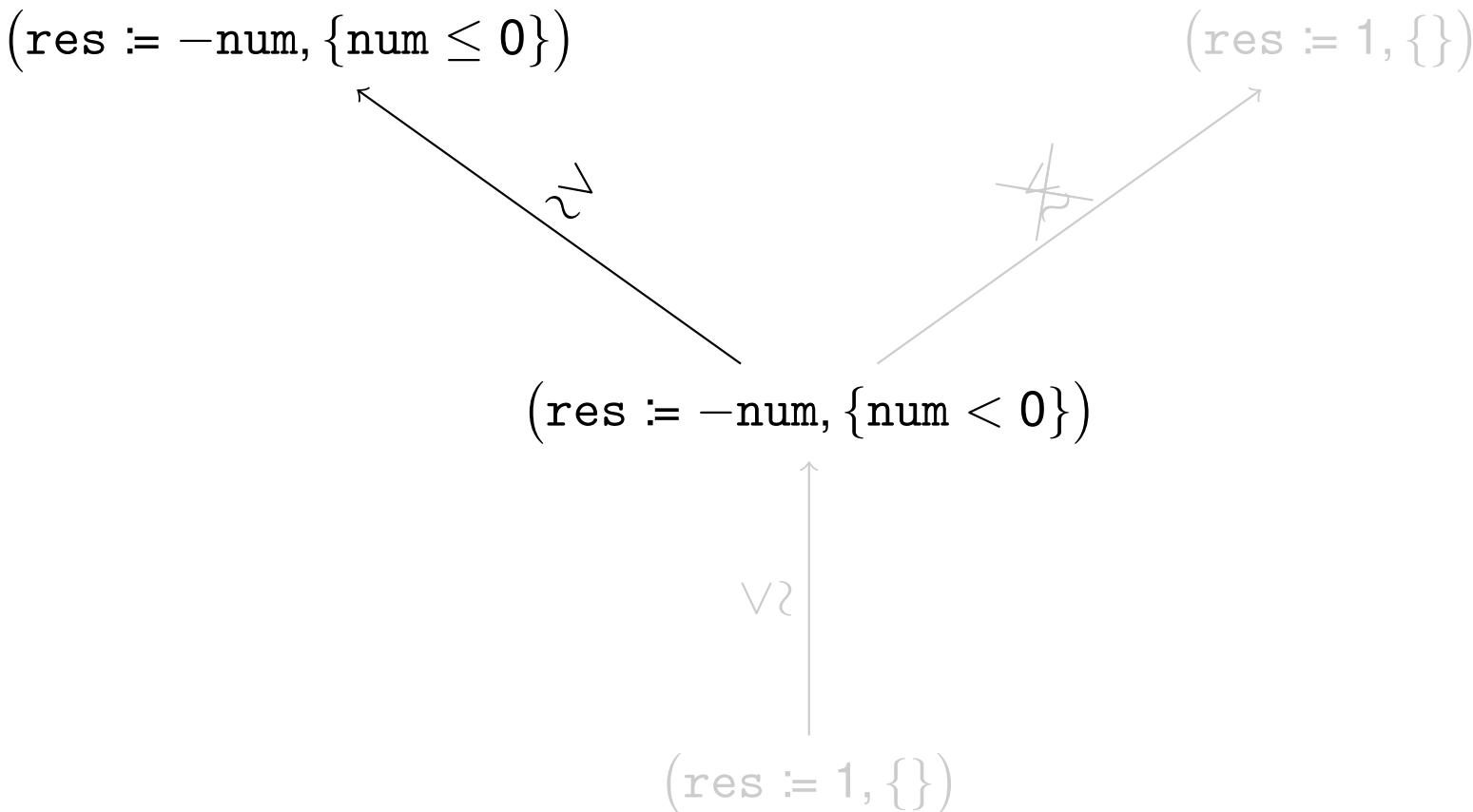
TECHNISCHE UNIVERSITÄT DARMSTADT



Software Engineering Group

Some simple examples for weakening

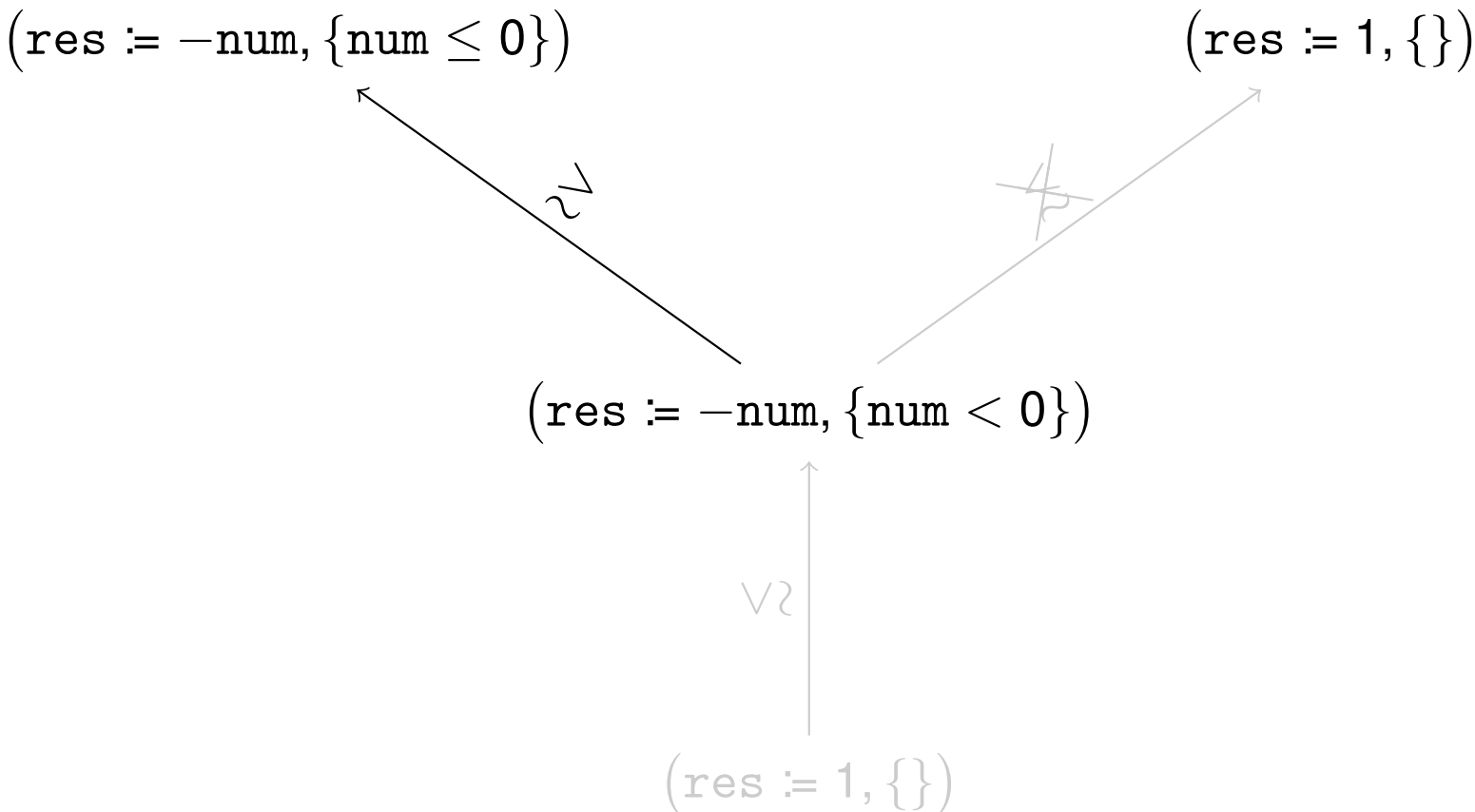
TECHNISCHE UNIVERSITÄT DARMSTADT



Software Engineering Group

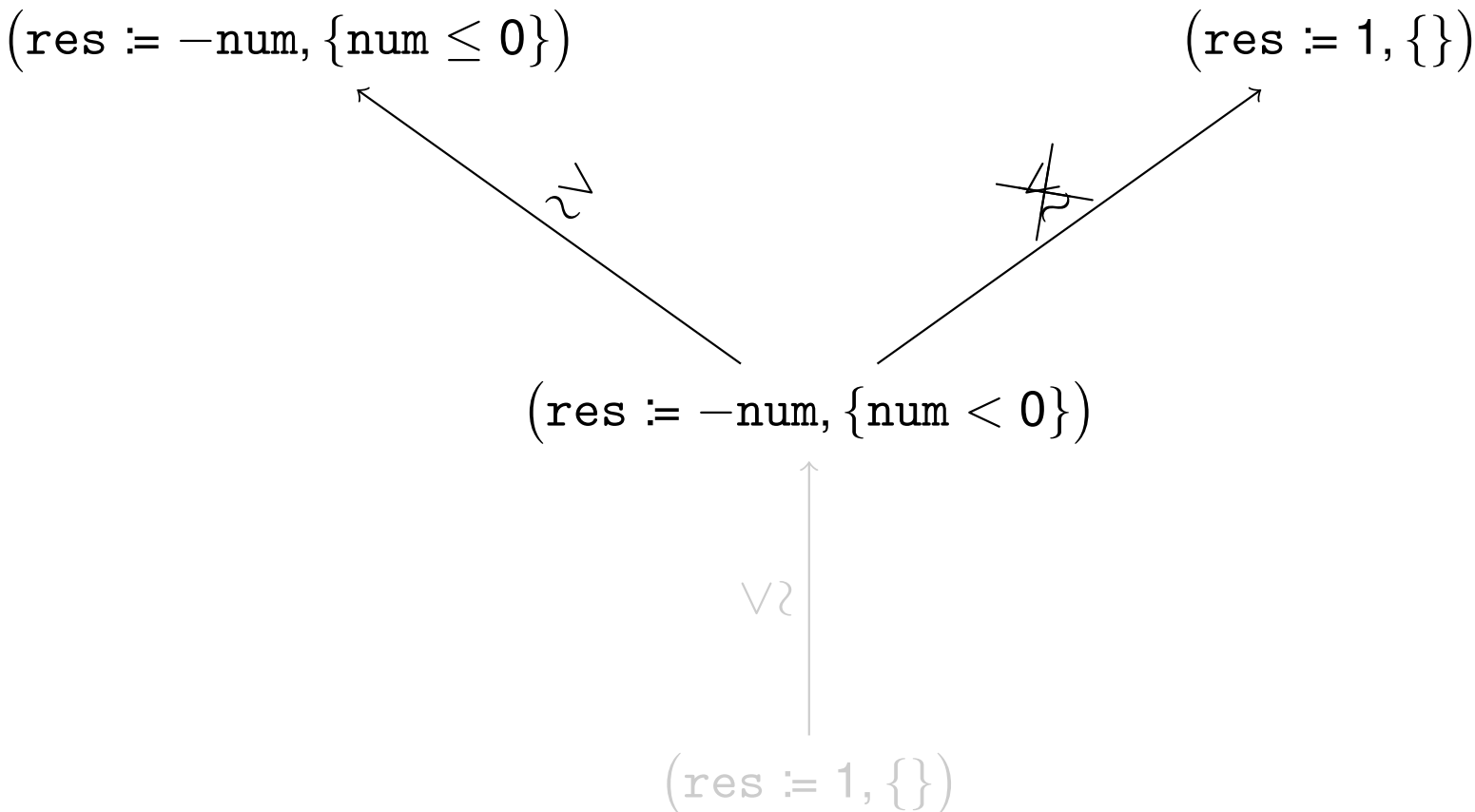
Some simple examples for weakening

TECHNISCHE UNIVERSITÄT DARMSTADT



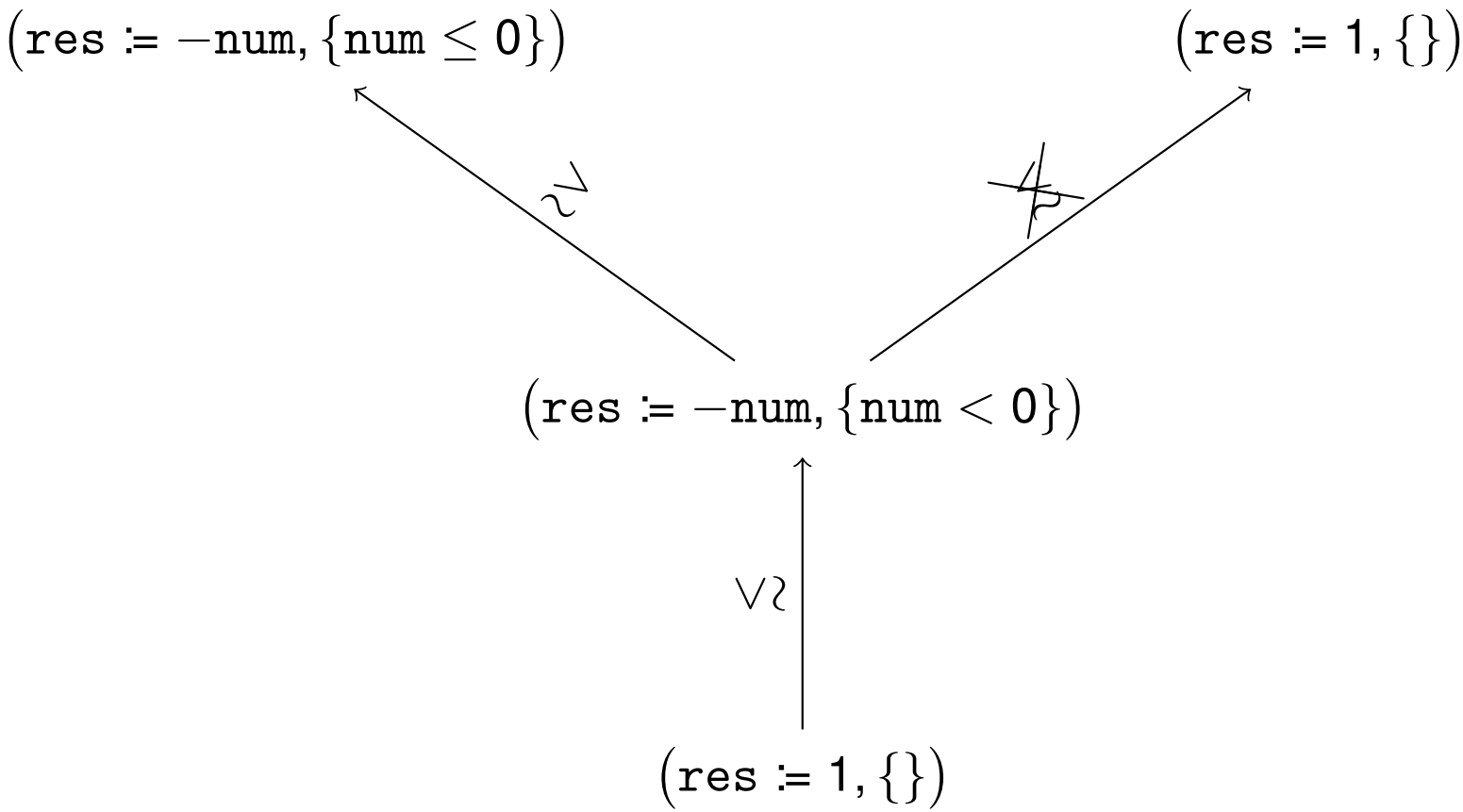
Software Engineering Group

Some simple examples for weakening



Some simple examples for weakening

TECHNISCHE UNIVERSITÄT DARMSTADT



Software Engineering Group

Closing the gap to abstract interpretation: The general lattice model for SE

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

Let $\dot{\sqcup}$ be an operation on SE states. We call the family of structures

$$\{\mathbb{L}_\varphi\}_{\varphi \in \mathrm{Fml}} := \left\{\left(S_\varphi, \dot{\sqcup}\right)\right\}_\varphi \text{ where}$$

$$S_\varphi := \{(U, C, \varphi) \mid (U, C, \varphi) \text{ is an SE state}\}$$

the *induced family of join-semilattices for* $\dot{\sqcup}$ iff $\dot{\sqcup}$ is *idempotent*, *commutative* and *associative* for states with the *same program counter*, its induced partial order relation $\preceq$ *is consistent with* "$\lesssim$", and both operands $\subseteq$ of the result *have the same program counter*.

Closing the gap to abstract interpretation: The general lattice model for SE

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

Let $\dot{\sqcup}$ be an operation on SE states. We call the family of structures

$$\{\mathbb{L}_\varphi\}_{\varphi \in \mathsf{Fml}} := \left\{\left(S_\varphi, \dot{\sqcup}\right)\right\}_\varphi \text{ where}$$

$$S_\varphi := \{(U, C, \varphi) \mid (U, C, \varphi) \text{ is an SE state}\}$$

the *induced family of join-semilattices for* $\dot{\sqcup}$ iff $\dot{\sqcup}$ is *idempotent*, *commutative* and *associative* for states with the *same program counter*, its induced partial order relation $\preceq$ *is consistent with* "$\lesssim$", and path conditions C^* of merged states obey certain *constraints*.

Closing the gap to abstract interpretation: The general lattice model for SE

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

Let $\dot{\sqcup}$ be an operation on SE states. We call the family of structures

$$\{\mathbb{L}_\varphi\}_{\varphi \in \mathsf{Fml}} := \left\{\left(S_\varphi, \dot{\sqcup}\right)\right\}_\varphi \text{ where}$$

$$S_\varphi := \{(U, C, \varphi) \mid (U, C, \varphi) \text{ is an SE state}\}$$

the *induced family of join-semilattices for* $\dot{\sqcup}$ iff $\dot{\sqcup}$ is *idempotent*, *commutative* and *associative* for states with the *same program counter*, its induced partial order relation $\preceq$ *is consistent with* "$\lesssim$", and path conditions C^* of merged states obey certain *constraints*.

Closing the gap to abstract interpretation: The general lattice model for SE

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

Let $\dot{\sqcup}$ be an operation on SE states. We call the family of structures

$$\{\mathbb{L}_\varphi\}_{\varphi \in \mathsf{Fml}} := \left\{\left(S_\varphi, \dot{\sqcup}\right)\right\}_\varphi \text{ where}$$

$$S_\varphi := \{(U, C, \varphi) \mid (U, C, \varphi) \text{ is an SE state}\}$$

the *induced family of join-semilattices for* $\dot{\sqcup}$ iff $\dot{\sqcup}$ is *idempotent*, *commutative* and *associative* for states with the *same program counter*, its induced partial order relation $\preceq$ *is consistent with* "$\lesssim$", and path conditions C^* of merged states obey certain *constraints*.

The general lattice model cont'd: Constraints on merge path conditions

TECHNISCHE UNIVERSITÄT DARMSTADT

$(U_1, C_1, \varphi) \preceq (U_2, C_2, \varphi)$ implies

- C_2 is logically equivalent to a formula $C \wedge Ax_{\overline{v}} \left[\overline{c} \,/\, \overline{v}\right]$ ($\overline{c}$ are "new" Skolem constants in (U_2, C_2, φ)), where
- $\bigwedge C_1 \rightarrow C$ is provable and
- $\exists \overline{v}; Ax_{\overline{v}}$ is provable

Lemma

If a formula φ does not contain any of the $\overline{c}$ and is true in all models (in a signature with $\overline{c}$) that satisfy C and $Ax_{\overline{v}} \left[\overline{c} \,/\, \overline{v}\right]$, then φ is also true in all models that only satisfy C (in the signature without $\overline{c}$).

The general lattice model cont'd: Constraints on merge path conditions

TECHNISCHE UNIVERSITÄT DARMSTADT

$(U_1, C_1, \varphi) \preceq (U_2, C_2, \varphi)$ implies

- C_2 is logically equivalent to a formula $C \wedge Ax_{\overline{v}} \left[\overline{c} \,/ \overline{v}\right]$ ($\overline{c}$ are "new" Skolem constants in (U_2, C_2, φ)), where
- $\bigwedge C_1 \rightarrow C$ is provable and
- $\exists \overline{v}; Ax_{\overline{v}}$ is provable

Lemma

If a formula φ does not contain any of the $\overline{c}$ and is true in all models (in a signature with $\overline{c}$) that satisfy C and $Ax_{\overline{v}} \left[\overline{c} \,/ \overline{v}\right]$, then φ is also true in all models that only satisfy C (in the signature without $\overline{c}$).

The general lattice model cont'd: Constraints on merge path conditions

TECHNISCHE UNIVERSITÄT DARMSTADT

$(U_1, C_1, \varphi) \preceq (U_2, C_2, \varphi)$ implies

- C_2 is logically equivalent to a formula $C \wedge Ax_{\overline{v}} \left[\overline{c} \,/ \overline{v}\right]$ ($\overline{c}$ are "new" Skolem constants in (U_2, C_2, φ)), where
- $\bigwedge C_1 \rightarrow C$ is provable and
- $\exists \overline{v}; Ax_{\overline{v}}$ is provable

Lemma

If a formula φ does not contain any of the $\overline{c}$ and is true in all models (in a signature with $\overline{c}$) that satisfy C and $Ax_{\overline{v}} \left[\overline{c} \,/ \overline{v}\right]$, then φ is also true in all models that only satisfy C (in the signature without $\overline{c}$).

The general lattice model cont'd: Constraints on merge path conditions

TECHNISCHE UNIVERSITÄT DARMSTADT

$(U_1, C_1, \varphi) \preceq (U_2, C_2, \varphi)$ implies

- C_2 is logically equivalent to a formula $C \wedge Ax_{\overline{v}} \left[\overline{c} \,/\, \overline{v}\right]$ ($\overline{c}$ are "new" Skolem constants in (U_2, C_2, φ)), where
- $\bigwedge C_1 \rightarrow C$ is provable and
- $\exists \overline{v}; Ax_{\overline{v}}$ is provable

Lemma

If a formula φ does not contain any of the $\overline{c}$ and is true in all models (in a signature with $\overline{c}$) that satisfy C and $Ax_{\overline{v}} \left[\overline{c} \,/\, \overline{v}\right]$, then φ is also true in all models that only satisfy C (in the signature without $\overline{c}$).

An example for constraints on merge path conditions

TECHNISCHE UNIVERSITÄT DARMSTADT

$$(\mathtt{x} \coloneqq 42, C_1, \varphi) \dot{\sqcup} (\mathtt{x} \coloneqq 17, C_2, \varphi) =$$

$$\left(\mathtt{x} \coloneqq c, \left\{ \left(\bigwedge C_1 \vee \bigwedge C_2 \right), c \geq 0 \right\}, \varphi \right)$$

$$(\mathtt{x} \coloneqq 42, C_1, \varphi) \preceq \left(\mathtt{x} \coloneqq c, \underbrace{\left\{ \left(\bigwedge C_1 \vee \bigwedge C_2 \right), c \geq 0 \right\}}_{\equiv C \wedge Ax_{\overline{v}}[\overline{c}/\overline{v}]}, \varphi \right)$$

Software Engineering Group

An example for constraints on merge path conditions

TECHNISCHE UNIVERSITÄT DARMSTADT

$$(\mathtt{x} \coloneqq 42, C_1, \varphi) \dot{\sqcup} (\mathtt{x} \coloneqq 17, C_2, \varphi) =$$

$$\left(\mathtt{x} \coloneqq c, \left\{ \left(\bigwedge C_1 \vee \bigwedge C_2 \right), c \geq 0 \right\}, \varphi \right)$$

$$(\mathtt{x} \coloneqq 42, C_1, \varphi) \preceq \left(\mathtt{x} \coloneqq c, \underbrace{\left\{ \left(\bigwedge C_1 \vee \bigwedge C_2 \right), c \geq 0 \right\}}_{\equiv C \wedge Ax_{\overline{v}}\left[\overline{c}/\overline{v}\right]}, \varphi \right)$$

Software Engineering Group

Merging with join-semilattices is sound

TECHNISCHE UNIVERSITÄT DARMSTADT

Theorem

Let $\varphi \in \mathsf{Fml}$ *be a formula and* $\dot{\sqcup}$ *be a join operation of an induced join-semilattice* $\mathbb{L}_\varphi$ *. Then, merging two SE states*

$$s_i = (U_i, C_i, \varphi)\,, \quad i = 1, 2,$$

to a state

$$s^* = s_1 \dot{\sqcup} s_2$$

is sound*, i.e. if* s^* *is valid, then both* s_1 *and* s_2 *are valid.*

TECHNISCHE
UNIVERSITÄT
DARMSTADT

$2\frac{1}{2}$ popular merge techniques

Maximum precision with *if-then-else*

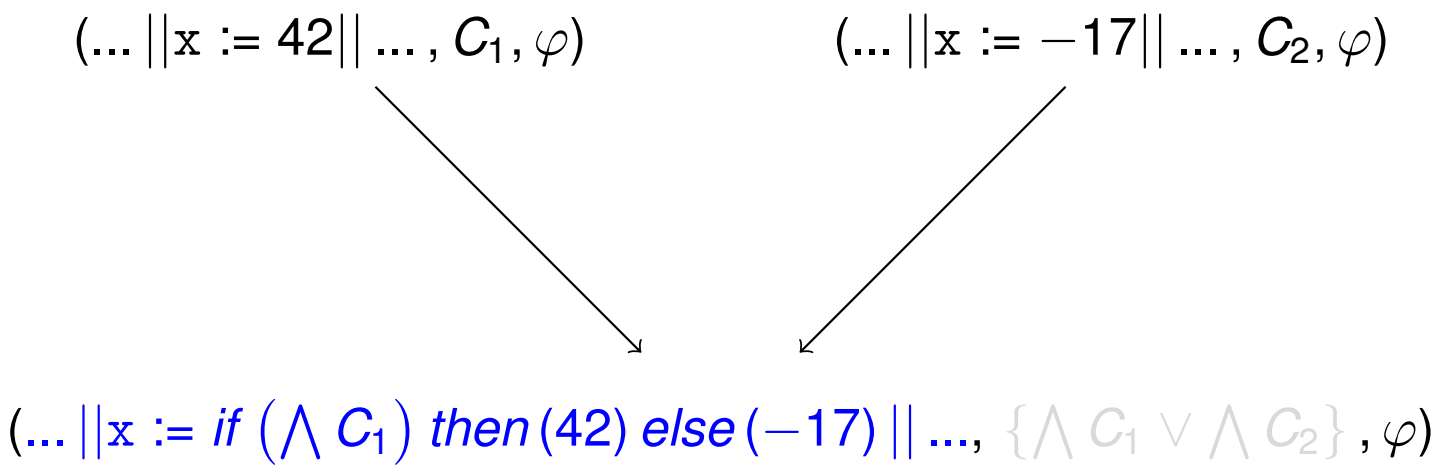
TECHNISCHE UNIVERSITÄT DARMSTADT

$$(\ldots \,||\mathtt{x} := 42||\, \ldots, C_1, \varphi) \qquad (\ldots \,||\mathtt{x} := -17||\, \ldots, C_2, \varphi)$$

$$(\ldots \,||\mathtt{x} := \mathit{if}\ \left(\bigwedge C_1\right)\ \mathit{then}\ (42)\ \mathit{else}\ (-17)\,||\, \ldots, \left\{\bigwedge C_1 \vee \bigwedge C_2\right\}, \varphi)$$

Maximum precision with *if-then-else*

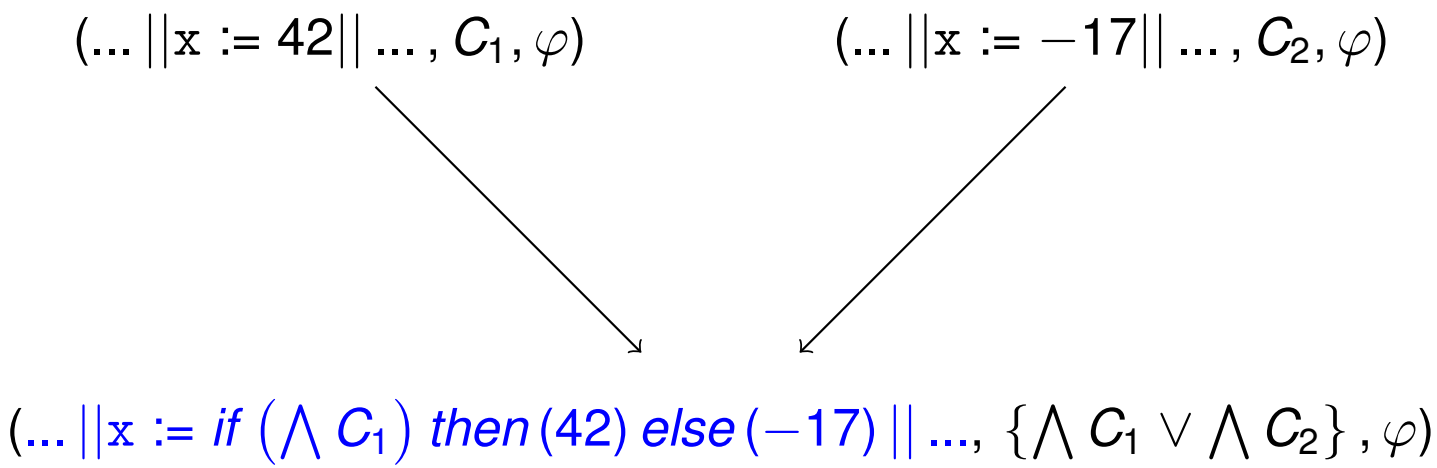
TECHNISCHE UNIVERSITÄT DARMSTADT



Software Engineering Group

Maximum precision with *if-then-else*

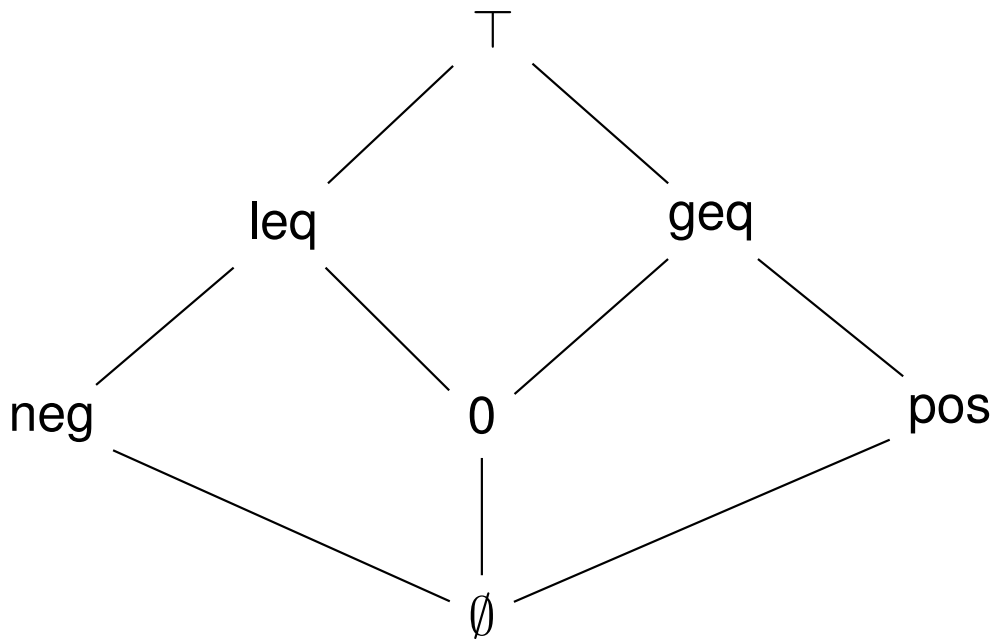
TECHNISCHE UNIVERSITÄT DARMSTADT



Software Engineering Group

Symbolic Execution meets Abstract Interpretation

TECHNISCHE UNIVERSITÄT DARMSTADT



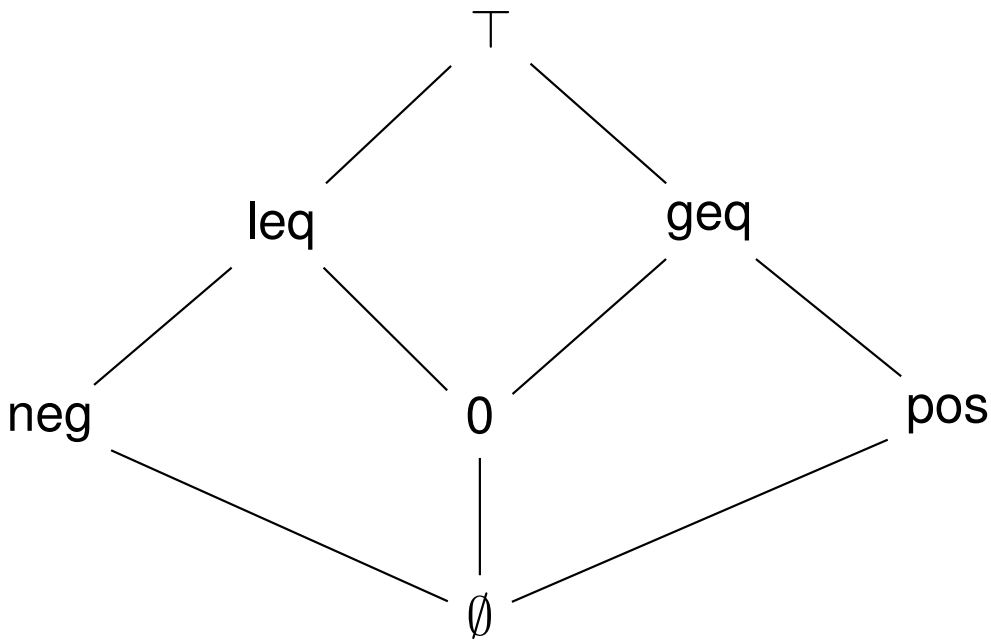
$$\gamma(\top) = \mathbb{Z}$$
$$\gamma(\text{leq}) = \{i \in \mathbb{Z} | i \leq 0\}$$
$$\gamma(\text{geq}) = \{i \in \mathbb{Z} | i \geq 0\}$$
$$\gamma(\text{neg}) = \{i \in \mathbb{Z} | i < 0\}$$
$$\gamma(\text{pos}) = \{i \in \mathbb{Z} | i > 0\}$$
$$\gamma(0) = \{0\}$$
$$\gamma(\emptyset) = \{\}$$

Example from [BHW09]

Software Engineering Group

Symbolic Execution meets Abstract Interpretation

TECHNISCHE UNIVERSITÄT DARMSTADT



$$
\begin{aligned}
\gamma(\top) &= \mathbb{Z} \\
\gamma(\text{leq}) &= \{i \in \mathbb{Z} | i \leq 0\} \\
\gamma(\text{geq}) &= \{i \in \mathbb{Z} | i \geq 0\} \\
\gamma(\text{neg}) &= \{i \in \mathbb{Z} | i < 0\} \\
\gamma(\text{pos}) &= \{i \in \mathbb{Z} | i > 0\} \\
\gamma(0) &= \{0\} \\
\gamma(\emptyset) &= \{\}
\end{aligned}
$$

Example from [BHW09]

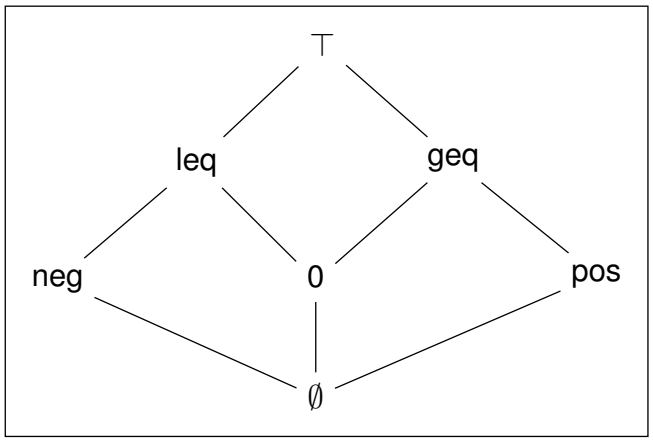
Software Engineering Group

Symbolic Execution meets Abstract Interpretation

TECHNISCHE UNIVERSITÄT DARMSTADT

$$(\ldots \,||\mathtt{x} := 0||\, \ldots, C_1, \varphi) \qquad (\ldots \,||\mathtt{x} := -17||\, \ldots, C_2, \varphi)$$

$$(\ldots \,||\mathtt{x} := c_{leq}||\, \ldots, \{\textstyle\bigwedge C_1 \vee \bigwedge C_2, c_{leq} \leq 0\}, \varphi)$$



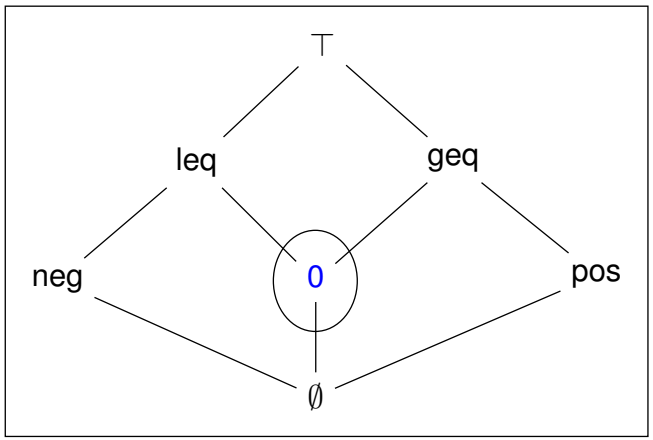
Software Engineering Group

Symbolic Execution meets Abstract Interpretation

TECHNISCHE UNIVERSITÄT DARMSTADT

$$(\ldots \,||x := 0||\, \ldots, C_1, \varphi) \qquad (\ldots \,||x := -17||\, \ldots, C_2, \varphi)$$

$$(\ldots \,||x := c_{leq}||\, \ldots, \{\textstyle\bigwedge C_1 \vee \bigwedge C_2, c_{leq} \leq 0\}, \varphi)$$



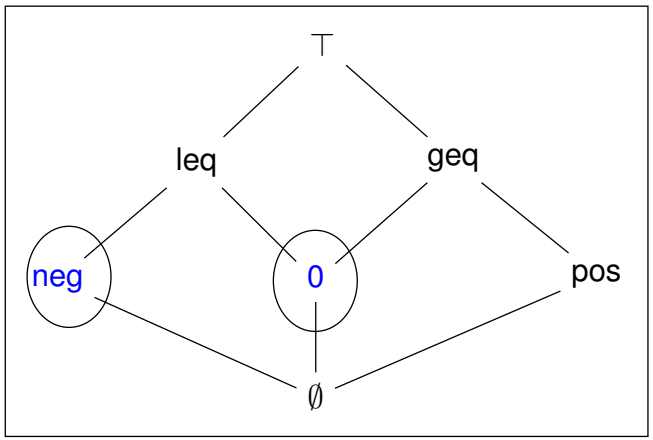
Software Engineering Group

Symbolic Execution meets Abstract Interpretation

TECHNISCHE UNIVERSITÄT DARMSTADT

$$(\ldots \,||\mathtt{x} := 0||\, \ldots, C_1, \varphi) \qquad (\ldots \,||\mathtt{x} := -17||\, \ldots, C_2, \varphi)$$

$$(\ldots \,||\mathtt{x} := c_{leq}||\, \ldots, \{\textstyle\bigwedge C_1 \vee \bigwedge C_2, c_{leq} \leq 0\}, \varphi)$$



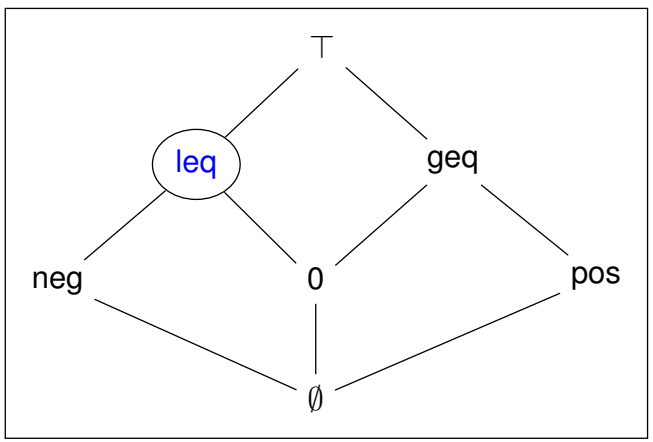
Software Engineering Group

Symbolic Execution meets Abstract Interpretation

TECHNISCHE UNIVERSITÄT DARMSTADT

$$(\ldots \,||x := 0||\, \ldots, C_1, \varphi) \qquad (\ldots \,||x := -17||\, \ldots, C_2, \varphi)$$

$$(\ldots \,||x := c_{leq}||\, \ldots, \{\textstyle\bigwedge C_1 \vee \bigwedge C_2, c_{leq} \leq 0\}, \varphi)$$

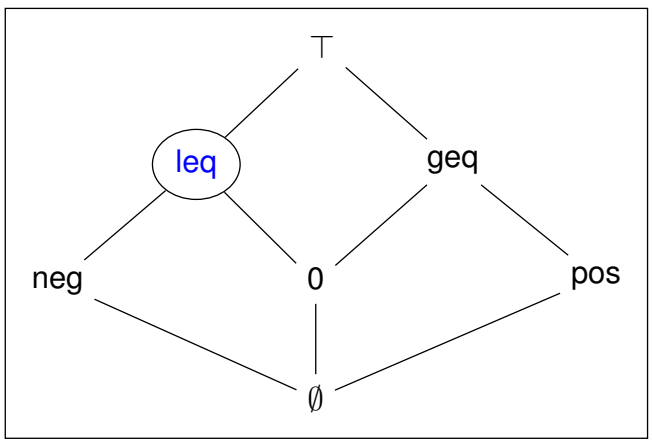


Software Engineering Group

Symbolic Execution meets Abstract Interpretation

TECHNISCHE UNIVERSITÄT DARMSTADT

$$(\ldots \,||\mathtt{x} := 0||\, \ldots, C_1, \varphi) \qquad (\ldots \,||\mathtt{x} := -17||\, \ldots, C_2, \varphi)$$



$$(\ldots \,||\mathtt{x} := c_{leq}||\, \ldots, \{\textstyle\bigwedge C_1 \vee \bigwedge C_2, c_{leq} \leq 0\}, \varphi)$$

Software Engineering Group

Predicate Abstraction facilitates the ad-hoc construction of abstract domains tailored to a *specific problem*

TECHNISCHE UNIVERSITÄT DARMSTADT

$preds := \{\square \geq x,\ \square \geq 0\}$

```
1   // prove: result >= 0
2   if (y < x) {
3       int tmp = x;
4       x = y;
5       y = tmp;
6   }
7
8   return y - x;
```


Software Engineering Group

Predicate Abstraction facilitates the ad-hoc construction of abstract domains tailored to a *specific problem*

TECHNISCHE UNIVERSITÄT DARMSTADT

$preds := \{\square \geq x,\ \square \geq 0\}$

```
1   // prove: result >= 0
2   if (y < x) {
3       int tmp = x;
4       x = y;
5       y = tmp;
6   }
7
8   return y - x;
```

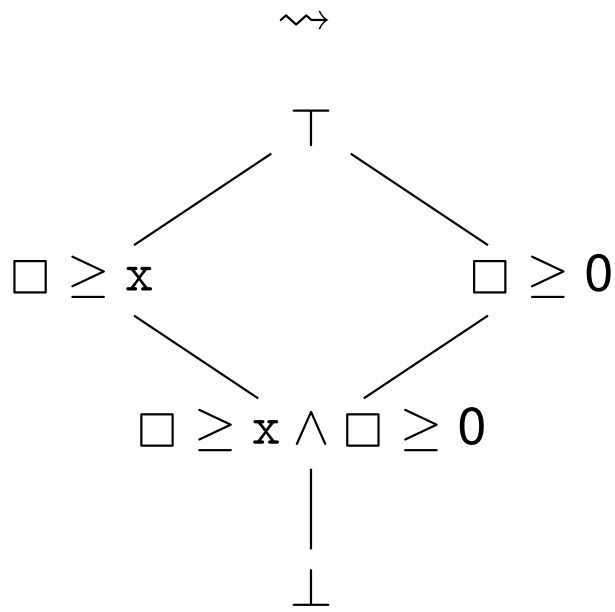

Software Engineering Group

Predicate Abstraction facilitates the ad-hoc construction of abstract domains tailored to a *specific problem*

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   // prove: result >= 0
2   if (y < x) {
3       int tmp = x;
4       x = y;
5       y = tmp;
6   }
7
8   return y - x;
```

$preds := \{\square \geq x,\ \square \geq 0\}$



Software Engineering Group

TECHNISCHE
UNIVERSITÄT
DARMSTADT

State merging in the SE workflow

A semi-automatic approach: *Merge annotations* in block contracts

TECHNISCHE UNIVERSITÄT DARMSTADT

```
/*@ join_proc "JoinByIfThenElse"; @*/
{
    if (current.parent == null}
        root = newParent;
        newParent.parent = null;
    } else {
        current.parent.right = newParent;
        newParent.parent = current.parent;
    }
}
current.parent = newParent;
//...
```

A semi-automatic approach: *Merge annotations* in block contracts

TECHNISCHE UNIVERSITÄT DARMSTADT

```
/*@ join_proc "JoinByIfThenElse"; @*/
{
                                                                    ;
}
c
// ...
```

Supported join_proc values

- "JoinByFullAnonymization"
- "JoinByIfThenElse"
- "JoinByIfThenElseAntecedent"
- "JoinBySignLatticeAbstraction"

Software Engineering Group

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Evaluation, Conclusion and Outlook

Results of the TimSort case study: Merging can shorten proofs significantly...

TECHNISCHE UNIVERSITÄT DARMSTADT

Value measured	With merging	Without merging
Nodes	88,333	303,716
Branches	324	632
Automode time	33min	28min
Merge Rule apps	6	0

⇒ **71%** improvement in #nodes

Timsort.gallopLeft

Value measured	With merging	Without merging
Nodes	63,309	279,156
Branches	1,576	1,804
Automode time	32min	41min
Merge Rule apps	1	0

⇒ **77%** improvement in #nodes

Timsort.mergeAt

Results of the TimSort case study: Merging can shorten proofs significantly...

TECHNISCHE UNIVERSITÄT DARMSTADT

Value measured	With merging	Without merging
Nodes	88,333	303,716
Branches	324	632
Automode time	33min	28min
Merge Rule apps	6	0

⇒ **71%** improvement in #nodes

Timsort.gallopLeft

Value measured	With merging	Without merging
Nodes	63,309	279,156
Branches	1,576	1,804
Automode time	32min	41min
Merge Rule apps	1	0

⇒ **77%** improvement in #nodes

Timsort.mergeAt

Results of the TimSort case study: ...facilitates symbolic execution of previously infeasible methods...

TECHNISCHE UNIVERSITÄT DARMSTADT

Value measured	With merging	Without merging
Nodes	460,410	N/A
Branches	2,310	N/A
Automode time	2h 29min	N/A
Merge Rule apps	5	0

Timsort.mergeHi

Value measured	With merging	Without merging
Nodes	1,455,919	N/A
Branches	2,906	N/A
Automode time	14h 20min	N/A
Merge Rule apps	12	0

Timsort.mergeLo

Results of the TimSort case study: ...and makes some complicated block contracts superfluous.

TECHNISCHE UNIVERSITÄT DARMSTADT

```
/*@ normal_behavior
  @ ensures
  @       (i == (\bigint)stackSize-3 ==>
  @          (    runLen[(\bigint)i+1]  == \old(runLen[(\bigint)i+2])
  @           && runBase[(\bigint)i+1] == \old(runBase[(\bigint)i+2])))
  @   &&
  @       (i == (\bigint)stackSize-2 ==>
  @          (    runLen[(\bigint)i+1]  == \old(runLen[(\bigint)i+1])
  @           && runBase[(\bigint)i+1] == \old(runBase[(\bigint)i+1])));
  @ assignable
  @   runBase[i+1], runLen[i+1];
  @*/
```



```
//@ join_proc "JoinByIfThenElse";
```


Software Engineering Group

Results of the TimSort case study: ...and makes some complicated block contracts superfluous.

TECHNISCHE UNIVERSITÄT DARMSTADT

```
/*@ normal_behavior
  @ ensures
  @       (i == (\bigint)stackSize-3 ==>
  @          (    runLen[(\bigint)i+1]  == \old(runLen[(\bigint)i+2])
  @           && runBase[(\bigint)i+1] == \old(runBase[(\bigint)i+2])))
  @   &&
  @       (i == (\bigint)stackSize-2 ==>
  @          (    runLen[(\bigint)i+1]  == \old(runLen[(\bigint)i+1])
  @           && runBase[(\bigint)i+1] == \old(runBase[(\bigint)i+1])));
  @ assignable
  @   runBase[i+1], runLen[i+1];
  @*/
```

```
//@ join_proc "JoinByIfThenElse";
```


Software Engineering Group

Examples of Ongoing and Future Work

TECHNISCHE UNIVERSITÄT DARMSTADT

```
/*@ merge contract;
  @ \method "Predicate Abstraction";
  @ \predicates (v, v >= 0),
  @             (v, v >= x); */
if (y < x) {
    int tmp = x;
    x = y;
    y = tmp;
}

return y - x;
```

Fully fledged block contracts for state merging

```
/*@ merge contract;
  @ \method "Predicate Abstraction";
  @ \predicates \automatic;
  @*/
if (y < x) {
    int tmp = x;
    x = y;
    y = tmp;
}

return y - x;
```

Automatic inference of abstraction predicates

Examples of Ongoing and Future Work

TECHNISCHE UNIVERSITÄT DARMSTADT

```
/*@ merge contract;
  @ \method "Predicate Abstraction";
  @ \predicates (v, v >= 0),
  @             (v, v >= x); */
if (y < x) {
    int tmp = x;
    x = y;
    y = tmp;
}

return y - x;
```

Fully fledged block contracts for state merging

```
/*@ merge contract;
  @ \method "Predicate Abstraction";
  @ \predicates \automatic;
  @*/
if (y < x) {
    int tmp = x;
    x = y;
    y = tmp;
}

return y - x;
```

Automatic inference of abstraction predicates

A general framework for merging SE states with KeY

TECHNISCHE UNIVERSITÄT DARMSTADT

- ✓ Parametric merge rule with soundness result
- ✓ General lattice framework for admissible join operations
- ✓ Instantiations for the two most prominent join operations
- ✓ Implementation for KeY theorem prover
- ✓ Evaluation with Timsort: Merging can significantly reduce size of proof trees

A general framework for merging SE states with KeY

TECHNISCHE UNIVERSITÄT DARMSTADT

- ✓ Parametric merge rule with soundness result
- ✓ General lattice framework for admissible join operations
- ✓ Instantiations for the two most prominent join operations
- ✓ Implementation for KeY theorem prover
- ✓ Evaluation with Timsort: Merging can significantly reduce size of proof trees

A general framework for merging SE states with KeY

TECHNISCHE UNIVERSITÄT DARMSTADT

- ✓ Parametric merge rule with soundness result
- ✓ General lattice framework for admissible join operations
- ✓ Instantiations for the two most prominent join operations
- ✓ Implementation for KeY theorem prover
- ✓ Evaluation with Timsort: Merging can significantly reduce size of proof trees

A general framework for merging SE states with KeY

TECHNISCHE UNIVERSITÄT DARMSTADT

- ✓ Parametric merge rule with soundness result
- ✓ General lattice framework for admissible join operations
- ✓ Instantiations for the two most prominent join operations
- ✓ Implementation for KeY theorem prover
- ✓ Evaluation with Timsort: Merging can significantly reduce size of proof trees

A general framework for merging SE states with KeY

TECHNISCHE UNIVERSITÄT DARMSTADT

- ✓ Parametric merge rule with soundness result
- ✓ General lattice framework for admissible join operations
- ✓ Instantiations for the two most prominent join operations
- ✓ Implementation for KeY theorem prover
- ✓ Evaluation with Timsort: Merging can significantly reduce size of proof trees

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Literature

Literature I

TECHNISCHE UNIVERSITÄT DARMSTADT

Bernhard Beckert, Reiner Hähnle, and Peter Schmitt.
Verification of Object-Oriented Software: The KeY Approach.
Springer Berlin Heidelberg, 2009.

Benjamin Weiß.
Deductive Verification of Object-Oriented Software: Dynamic Frames, Dynamic Logic and Predicate Abstraction.
PhD thesis. Karlsruhe Institute of Technology, 2011.

Richard Bubel, Reiner Hähnle, and Benjamin Weiß.
Abstract Interpretation of Symbolic Execution with Explicit State Updates.
In Frank de Boer, Marcello Bonsangue, and Eric Madelaine, editors, Formal Methods for Components and Objects.
Springer Berlin Heidelberg, 2009.

Literature II

TECHNISCHE UNIVERSITÄT DARMSTADT

Trevor Hansen, Peter Schachte, and Harald Sondergaard.
State Joining and Splitting for the Symbolic Execution of Binaries.
In Saddek Bensalem and Doron Peled, editors, LNCS 5779. Springer Berlin Heidelberg, 2009.

Volodymyr Kuznetsov et al.
Efficient State Merging in Symbolic Execution.
In: Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, 2012.

Patrick Cousot and Radhia Cousot.
Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.
In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. New York, 1977.