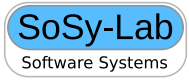
# Formale Spezifikation und Verifikation

## Wintersemester 2024

Prof. Dr. Gidon Ernst
gidon.ernst@lmu.de

Software and Computational Systems Lab
Ludwig-Maximilians-Universität München, Germany

September 30, 2024

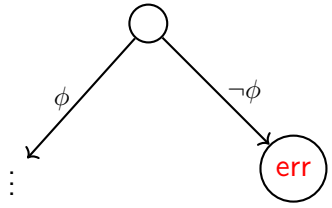
# Nichtdeterminismus und Annahmen

# Erinnerung: assert im Kontrollflussautomat

**Spezifikation durch Zusicherung**

`assert` $\phi$;

in einem Programm $P$




**Verifikation**

- Berechnung der erreichbaren Zustände $\sigma^R$ von $P$: mit Hilfe des Transitionssystems und der expliziten Erreichbarkeitsanalyse
- Falls es einen Zustand $s \in \sigma^R$ gibt mit $s \models \mathsf{pc} = \mathsf{err}$, dann wurde ein Fehler gefunden

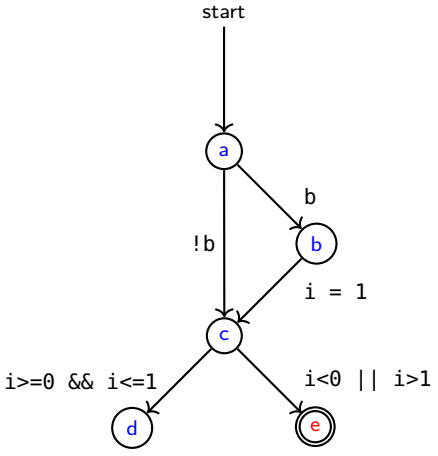
# Heute

- Nichtdeterministische Auswahl und Annahmen als Modellierungtechnik
  - Methodenparameter: alle möglichen Aufrufe (`@ForAll`)
  - Benutzereingaben: alle möglichen Eingaben (`__VERIFIER_nondet_int()`)
  - Abstraktion: alle möglichen Verhalten ($post(\text{on}) = \{\text{off}, \text{broken}\}$)

  $\rightarrow$ Nichtdeterminismus $\approx$ Mengen von Zuständen
- Zustandsmengen als Auswertung von Formeln
- Explizite Erreichbarkeitsanalyse auf Zustandsmengen
- Symbolische Repräsentation von Zustandsmengen
- Symbolische Erreichbarkeitsanalyse

# Beispiel: viele Startzustände

**Kontrollflussautomat**




**Transitionssystem**

$$\{\{\texttt{pc} \mapsto \textcircled{a}, \texttt{b} \mapsto b, \texttt{i} \mapsto 0\} \mid b \in \mathbb{B}\}$$

$$\{\textcircled{a}, \mathit{false}, 0\} \quad \{\textcircled{a}, \mathit{true}, 0\}$$

$$\{\textcircled{c}, \mathit{false}, 0\} \quad \{\textcircled{b}, \mathit{true}, 0\}$$

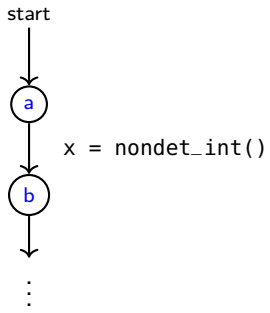
$$\{\textcircled{d}, \mathit{false}, 0\} \quad \{\textcircled{c}, \mathit{true}, 1\}$$

$$\{\textcircled{d}, \mathit{true}, 1\}$$

✓ kompakte Darstellung

# Beispiel: viele Nachfolgerzustände

**Kontrollflussautomat**



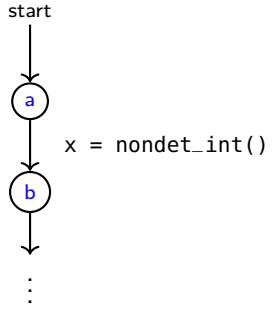

**Programm**

```
int x = nondet_int();
...
```

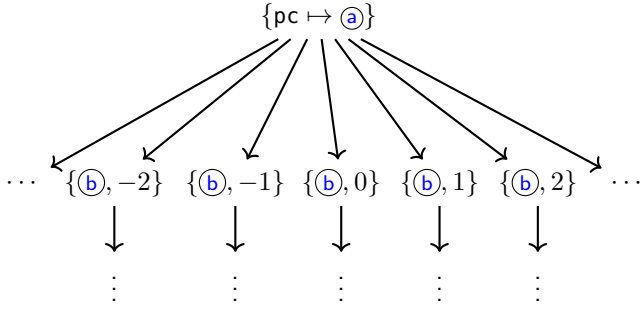
Idee: x ist *uneingeschränkt*

# Beispiel: viele Nachfolgerzustände

**Kontrollflussautomat**




**Transitionssystem**




Ein Nachfolger pro $x \in [\![\texttt{int}]\!]$

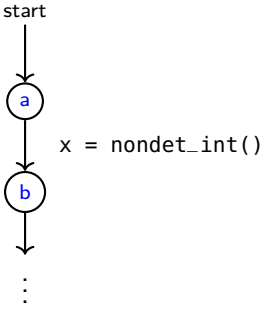
**Programm**

```
int x = nondet_int();
...
```

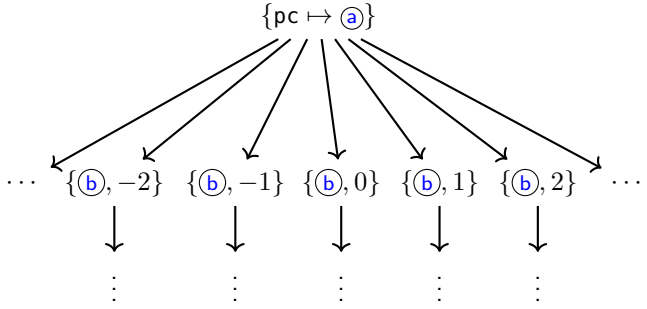
Idee: x ist *uneingeschränkt*

# Beispiel: viele Nachfolgerzustände

**Kontrollflussautomat**




**Transitionssystem**




Ein Nachfolger pro $x \in [\![\texttt{int}]\!]$

**Programm**

```
int x = nondet_int();
...
```

Idee: x ist *uneingeschränkt*

- Auf 32-Bit Rechner: $[\![\texttt{int}]\!] = \{-2^{31}, \cdots, 2^{31}-1\}$
- Auf 64-Bit Rechner: $[\![\texttt{int}]\!] = \{-2^{63}, \cdots, 2^{63}-1\}$
- Idealisiert/Mathematisch: $[\![\texttt{int}]\!] = \mathbb{Z}$
- Mit `@ForAll` Zufallsauswal

# Äquivalente Spezifikationsansätze

Mit jqwik

```
void test(@ForAll
          @Positive int n)
{
    int m = sqrt(n);
    assert m <= n;
}
```

# Äquivalente Spezifikationsansätze

Mit jqwik

```
void test(@ForAll
          @Positive int n)
{
    int m = sqrt(n);
    assert m <= n;
}
```

Mit Annahmen
(✗ Testen: vergebliche Versuche)

```
void test(@ForAll int n) {
    assume n > 0; // Filter
    int m = sqrt(n);
    assert m <= n;
}
```

# Äquivalente Spezifikationsansätze

Mit jqwik

```
void test(@ForAll
          @Positive int n)
{
    int m = sqrt(n);
    assert m <= n;
}
```

Mit Annahmen
(✗ Testen: vergebliche Versuche)

```
void test(@ForAll int n) {
    assume n > 0; // Filter
    int m = sqrt(n);
    assert m <= n;
}
```

Mit nichtdeterministischer Auswahl

```
void test() {
    // cf. Arbitrary.sample()
    int n = nondet_int();
    assume n > 0;
    int m = sqrt(n);
    assert m <= n;
}
```

# Äquivalente Spezifikationsansätze

Mit jqwik

```
void test(@ForAll
          @Positive int n)
{
    int m = sqrt(n);
    assert m <= n;
}
```

Mit Annahmen
(✗ Testen: vergebliche Versuche)

```
void test(@ForAll int n) {
    assume n > 0; // Filter
    int m = sqrt(n);
    assert m <= n;
}
```

Mit nichtdeterministischer Auswahl

```
void test() {
    // cf. Arbitrary.sample()
    int n = nondet_int();
    assume n > 0;
    int m = sqrt(n);
    assert m <= n;
}
```

Semantische Aussage über Programm

- $P$: `int m = sqrt(n);`
- Sei $T$ Transitionssystem von $P$
- mit $\sigma^I = \left\{ \{\mathtt{n} \mapsto n\} \mid n > 0 \right\}$
- für alle Spuren $(s_1, \ldots, s_n)$ durch $T$
- wobei $s_1 \models \mathtt{n > 0}$ gegeben
- ist zu prüfen $s_n \models \mathtt{m <= n}$

# Weitere Beispiele

Modulare Spezifikation und Verifikation

**Implementierung**

```
class List<T extends Comparable> {
  void sort() {
    ...
    assert list.isSorted();
  }
}
```

# Weitere Beispiele

Modulare Spezifikation und Verifikation

**Implementierung**

```
class List<T extends Comparable> {
  void sort() {
    ...
    assert list.isSorted();
  }
}
```

**Aufrufer**

```
List<String> names = ...;
names.sort();
assume names.isSorted();
String s0 = names.get(0);
String s1 = names.get(1);
assert s0.compareTo(s1) <= 0;
```

# Weitere Beispiele

Modulare Spezifikation und Verifikation

**Implementierung**

```
class List<T extends Comparable> {
  void sort() {
    ...
    assert list.isSorted();
  }
}
```

**Aufrufer**

```
List<String> names = ...;
names.sort();
assume names.isSorted();
String s0 = names.get(0);
String s1 = names.get(1);
assert s0.compareTo(s1) <= 0;
```

- Der Aufrufer braucht genaue Implementierung nicht zu kennen!
  Hier: `assume` ist immer gerechtfertigt.
- Ausblick: Design by Contract [Meyer, 1992]:
  Typischerweise spezielle Sprachkonstrukte (vgl. JML)
- Zum Nachdenken: welche Probleme verbleiben in obigem Code?

# Weitere Beispiele

"Mathematische Beweise" als Programme

$$\forall\, n \in \mathbb{N}.\ \sum_{i=1..n} n = \frac{n \cdot (n+1)}{2}$$

```
int n = nondet_int(), sum = 0;
assume n >= 0;
for(int i=1; i<=n; i++) sum += n;
assert sum == n*(n+1) / 2;
```

# Weitere Beispiele

Nicht-algorithmische Beschaffung von Werten mit bestimmten Eigenschaften, z.B.: Primfaktorzerlegung von Kryptographischem Schlüssel k:

Problembeschreibung:

```
int p = nondet_int();
int q = nondet_int();
int b = prime(p) && prime(q) && p*q == k;
```

Lösung erzwingen:

```
assume b;
```

# Nichtdeterminismus und Annahmen

```
x = nondet_int()
```

- Ergebnis x völlig beliebig
- wir können *nicht selbst* wählen
- Spezifikation von *Möglichkeiten*

# Nichtdeterminismus und Annahmen

`x = nondet_int()`

- Ergebnis x völlig beliebig
- wir können *nicht selbst* wählen
- Spezifikation von *Möglichkeiten*

`assume` $\phi$`;`

- nicht erfüllte Annahmen $s \not\models \phi$ brechen die Ausführung einfach ab
- man *gibt vor* dass *nur bestimmte Fälle* betrachtet werden sollen

# Nichtdeterminismus und Annahmen

`x = nondet_int()`

- Ergebnis x völlig beliebig
- wir können *nicht selbst* wählen
- Spezifikation von *Möglichkeiten*

`assume` $\phi$`;`

- nicht erfüllte Annahmen $s \not\models \phi$ brechen die Ausführung einfach ab
- man *gibt vor* dass *nur bestimmte Fälle* betrachtet werden sollen

Beide Konzepte sind *mirakulös* (nicht implementierbar) und erlauben es, zu *raten*!
"Data refinement by miracles" [Morgan, 1988]

# Weitere Beispiele

Spezifikationen mit Annahmen können nicht-berechenbare Probleme ausdrücken

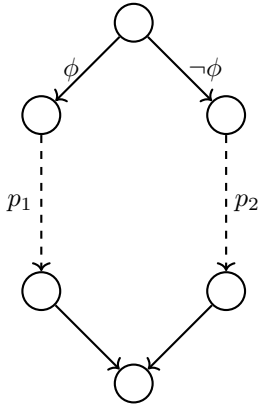
```
void maybeExecute(TuringMachine M, Input in) {
    boolean b = nondet_bool();
    boolean steps = nondet_int();
    assume b == M.halts(in, steps);

    if(b) {
        M.run(in, steps);
    } else {
        throw new Exception("Would not terminate");
    }
}
```

Beachte: Damit das Beispiel funktioniert, muss `assume` auch fordern dass `M.halts(in, steps)` terminiert. Diese Frage betrachten wir jetzt nicht und nehmen im Folgenden an, dass $\phi$ in `assume` $\phi$ keine solchen Probleme verursacht.

# Annahmen im Kontrollflussautomat

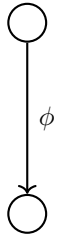
**Bedingung**

`if(`$\phi$`)` $p_1$ `else` $p_2$




**Annahme**

`assume` $\phi$`;`




Keine Kante für $\neg\phi$

# Nichtdeterminismus im Transitionssystem

Gegeben $P$ als Kontrollflussautomat $(L, \ell_0, G)$ mit Kanten $\ell \xrightarrow{op} \ell'$

Konstruktion von Transitionen in $T = (\Sigma, \sigma^I, \rightarrow)$

- Transition $s \rightarrow s'$ mit
  - $s = \{\mathsf{pc} \mapsto \ell, \vec{\mathsf{x}} \mapsto \vec{v}\}$
  - $s' = \{\mathsf{pc} \mapsto \ell', \vec{\mathsf{x}} \mapsto \vec{v}'\}$,

  für Kanten in $\ell \xrightarrow{op} \ell' \in G$

# Nichtdeterminismus im Transitionssystem

Gegeben $P$ als Kontrollflussautomat $(L, \ell_0, G)$ mit Kanten $\ell \xrightarrow{op} \ell'$

Konstruktion von Transitionen in $T = (\Sigma, \sigma^I, \rightarrow)$

- Transition $s \rightarrow s'$ mit
  - $s = \{\mathsf{pc} \mapsto \ell, \vec{\mathsf{x}} \mapsto \vec{v}\}$
  - $s' = \{\mathsf{pc} \mapsto \ell', \vec{\mathsf{x}} \mapsto \vec{v}'\}$,

  für Kanten in $\ell \xrightarrow{op} \ell' \in G$, sofern gilt dass
- Zuweisung $\ell \xrightarrow{\mathsf{x}_i = e} \ell'$: dann $v'_i = [\![e]\!]_s$ (Auswertung von $e$ in $s$), $v'_{j \neq i} = v_j$
- Bedingung $\ell \xrightarrow{\phi} \ell'$: dann $s \models \phi$ und $\vec{v}' = \vec{v}$

# Nichtdeterminismus im Transitionssystem

Gegeben $P$ als Kontrollflussautomat $(L, \ell_0, G)$ mit Kanten $\ell \xrightarrow{op} \ell'$

Konstruktion von Transitionen in $T = (\Sigma, \sigma^I, \rightarrow)$

- Transition $s \rightarrow s'$ mit
  - $s = \{\mathsf{pc} \mapsto \ell, \vec{\mathsf{x}} \mapsto \vec{v}\}$
  - $s' = \{\mathsf{pc} \mapsto \ell', \vec{\mathsf{x}} \mapsto \vec{v}'\}$,

  für Kanten in $\ell \xrightarrow{op} \ell' \in G$, sofern gilt dass
- Zuweisung $\ell \xrightarrow{\mathsf{x}_i = e} \ell'$: dann $v_i' = [\![e]\!]_s$ (Auswertung von $e$ in $s$), $v'_{j \neq i} = v_j$
- Bedingung $\ell \xrightarrow{\phi} \ell'$: dann $s \models \phi$ und $\vec{v}' = \vec{v}$

- Nichtdet. Zuweisung $\ell \xrightarrow{\mathsf{x}_i \text{ = nondet\_int();}} \ell'$: dann $v_i' \in [\![\mathtt{int}]\!]$ beliebig

# Nichtdeterminismus im Transitionssystem

Gegeben $P$ als Kontrollflussautomat $(L, \ell_0, G)$ mit Kanten $\ell \xrightarrow{op} \ell'$

Konstruktion von Transitionen in $T = (\Sigma, \sigma^I, \rightarrow)$

- Transition $s \rightarrow s'$ mit
  - $s = \{\mathsf{pc} \mapsto \ell, \vec{\mathsf{x}} \mapsto \vec{v}\}$
  - $s' = \{\mathsf{pc} \mapsto \ell', \vec{\mathsf{x}} \mapsto \vec{v}'\}$,

  für Kanten in $\ell \xrightarrow{op} \ell' \in G$, sofern gilt dass
- Zuweisung $\ell \xrightarrow{\mathsf{x}_i = e} \ell'$: dann $v_i' = [\![e]\!]_s$ (Auswertung von $e$ in $s$), $v'_{j \neq i} = v_j$
- Bedingung $\ell \xrightarrow{\phi} \ell'$: dann $s \models \phi$ und $\vec{v}' = \vec{v}$

- Nichtdet. Zuweisung $\ell \xrightarrow{\mathsf{x}_i \texttt{ = nondet\_int();}} \ell'$: dann $v_i' \in [\![\mathtt{int}]\!]$ beliebig
  Problem: $|post(s)| = |[\![\mathtt{int}]\!]|$ sind sehr viele Nachfolger

# Nichtdeterminismus vs Zufall

Eine *zufällige* Auswahl folgt immer einer *Wahrscheinlichkeitsverteilung*.
Typische Frage: bei 1000 Experimenten, wie oft kommt Zahl 1?

Bei Nichtdeterminismus kann man diese Frage garnicht stellen!
Es geht nur darum, dass wir als mögliche Ergebnisse $\{1, 2, 3, 4, 5, 6\}$ haben.

# Was man wissen sollte

- Wofür werden Nichtdeterminismus und Annahmen verwendet?
  - ✓ Spezifikation von Schnittstellen und Eingaben
  - ✓ Modellierung von Systemen
  - ✓ Modulare Beweise via Design by Contract
- Welche Probleme treten dann bei der expliziten Erreichbarkeitsanalyse auf?
  - ✗ Nichtdeterminismus: Zustandsexplosion
  - ✗ Annahmen: abgebrochene Ausführungen
  - ✗ Annahmen: müssen ausführbar sein
- Wie sind Annahmen im Kontrollflussautomat repräsentiert?
- Wie ist nichtdeterministische Auswahl im Transitionssystem repräsentiert?

# Symbolische Erreichbarkeitsanalyse

# Heute

✓ Nichtdeterministische Auswahl und Annahmen als Modellierungtechnik

- Methodenparameter: alle möglichen Aufrufe (`@ForAll`)
- Benutzereingaben: alle möglichen Eingaben (`__VERIFIER_nondet_int()`)
- Abstraktion: alle möglichen Verhalten ($post(\text{on}) = \{\text{off}, \text{broken}\}$)

→ Nichtdeterminismus ≈ Mengen von Zuständen

- Zustandsmengen als Auswertung von Formeln
- Explizite Erreichbarkeitsanalyse auf Zustandsmengen
- Symbolische Repräsentation von Zustandsmengen
- Symbolische Erreichbarkeitsanalyse

# Repräsentation von Zustandsmengen

Auszählung konkreter Zustände: $\left\{\{\mathsf{x} \mapsto 1\}, \{\mathsf{x} \mapsto 2\}, \ldots\right\}$

- ✓ konkret, Ausführungsnah
- ✗ skaliert nicht, nur endliche Mengen

# Repräsentation von Zustandsmengen

Auszählung konkreter Zustände: $\left\{\{\mathsf{x} \mapsto 1\}, \{\mathsf{x} \mapsto 2\}, \ldots\right\}$

- ✓ konkret, Ausführungsnah
- ✗ skaliert nicht, nur endliche Mengen

Durch Formeln beschreiben: $\left\{s \mid s \models \phi\right\}$

- ✓ kompakt
- ✓ effiziente Datenstrukturen und Algorithmen verfügbar

Notation für $\phi$ z.B. Java Ausrücke vom Typ `boolean`

# Repräsentation von Zustandsmengen

Auszählung konkreter Zustände: $\left\{\{\mathsf{x} \mapsto 1\}, \{\mathsf{x} \mapsto 2\}, \ldots\right\}$

- ✓ konkret, Ausführungsnah
- ✗ skaliert nicht, nur endliche Mengen

Durch Formeln beschreiben: $\left\{s \mid s \models \phi\right\}$

- ✓ kompakt
- ✓ effiziente Datenstrukturen und Algorithmen verfügbar

Notation für $\phi$ z.B. Java Ausrücke vom Typ **`boolean`**

Beispiel $\phi$: `x > 0` für **`int`** `x`

- $\left\{s \mid s \models \phi\right\} = \left\{s \mid s \models \mathtt{x > 0}\right\} = \left\{\{\mathsf{x} \mapsto x, \ldots\} \mid x \in [\![\mathtt{int}]\!] \text{ und } x > 0\right\}$

# Beispiele für Symbolische Zustandsmengen

Gegeben: `int x,y;`

| **Formel** $\phi$ | **Informell** | **Beispiele** $s \models \phi$ |
|---|---|---|
| $x = 9 \wedge y = x$ | beide Variablen haben Wert 9 | $\{x \mapsto 9, y \mapsto 9\}$ |
| $x > 0$ | x positiv, y *beliebig* | $\{x \mapsto 2, y \mapsto -23\}$ |
| $x > 0 \wedge y \leq 10$ | x positiv, y höchtens 10 | $\{x \mapsto 2, y \mapsto -23\}$ |
| $x > 0 \Longrightarrow y > 0$ | wenn x positiv, dann auch y | $\{x \mapsto 2, y \mapsto 1\}$<br>$\{x \mapsto 0, y \mapsto ?\}$ |
| $x > \lvert y \rvert$ | x größer als Betrag von y | $\{x \mapsto 2, y \mapsto -1\}$ |

# Beispiele für Symbolische Zustandsmengen

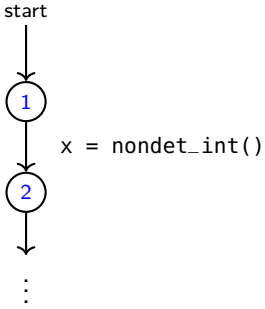
Gegeben: `int x,y;`

| **Formel** $\phi$ | **Informell** | **Beispiele** $s \models \phi$ |
|---|---|---|
| $x = 9 \wedge y = x$ | beide Variablen haben Wert 9 | $\{x \mapsto 9, y \mapsto 9\}$ |
| $x > 0$ | x positiv, y *beliebig* | $\{x \mapsto 2, y \mapsto -23\}$ |
| $x > 0 \wedge y \leq 10$ | x positiv, y höchtens 10 | $\{x \mapsto 2, y \mapsto -23\}$ |
| $x > 0 \Longrightarrow y > 0$ | wenn x positiv, dann auch y | $\{x \mapsto 2, y \mapsto 1\}$<br>$\{x \mapsto 0, y \mapsto ?\}$ |
| $x > \lvert y \rvert$ | x größer als Betrag von y | $\{x \mapsto 2, y \mapsto -1\}$ |

- Größe der symbolische Notation *nicht* proportional zur beschriebenen Zustandsmenge: 5 Symbole "$0 \leq x < 10000$" beschreiben $10^5$ Zustände
- Wir kommen ganz ohne *explizite* Zustände $s$ aus!

# Beispiel: Nichtdeterministische Auswahl

**Kontrollflussautomat**

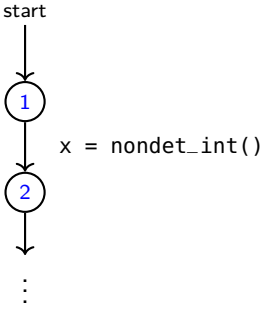



**Programm**

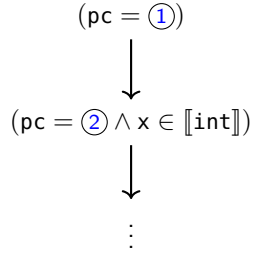
```
int x = nondet_int();
...
```

# Beispiel: Nichtdeterministische Auswahl

**Kontrollflussautomat**




**Erreichbare Zustände**




Ein symbolischer Nachfolger

**Programm**

```
int x = nondet_int();
...
```

# Explizite Erreichbarkeit mit Zustandsmengen

**Bisheriger Algorithmus**

$\sigma^R := \varnothing$ schon erreicht
$\tau := \sigma^I$ noch zu besuchen
**while** $\tau \neq \varnothing$ **do**
  **choose** $s$ **with** $s \in \tau$
  $\tau := \tau \setminus \{s\}$
  **if** $s \notin \sigma^R$ **then**
    $\sigma^R := \sigma^R \cup \{s\}$
    $\tau := \tau \cup post(s)$
  **end if**
**end while**

# Explizite Erreichbarkeit mit Zustandsmengen

**Bisheriger Algorithmus**

$\sigma^R := \varnothing$ schon erreicht
$\tau := \sigma^I$ noch zu besuchen
**while** $\tau \neq \varnothing$ **do**
  **choose** $s$ **with** $s \in \tau$
  $\tau := \tau \setminus \{s\}$
  **if** $s \notin \sigma^R$ **then**
    $\sigma^R := \sigma^R \cup \{s\}$
    $\tau := \tau \cup post(s)$
  **end if**
**end while**

**Mit Mengenoperationen**

$\sigma^R := \varnothing$ schon erreicht
$\tau := \sigma^I$ noch zu besuchen
**while** not $\tau \subseteq \sigma^R$ **do**
  $\sigma^R := \sigma^R \cup \tau$
  $\tau := Post(\tau)$
**end while**

Wir benötigen

- Operationen $\varnothing$, $\subseteq$, $\cup$
- $Post(\sigma) = \bigcup_{s \in \sigma} post(s)$

# Symbolische Erreichbarkeitsanalyse

**Neuer Algorithmus**

$\sigma^R := \varnothing$ schon erreicht

$\tau := \sigma^I$ noch zu besuchen

**while** not $\tau \subseteq \sigma^R$ **do**

  $\sigma^R := \sigma^R \cup \tau$

  $\tau := Post(\tau)$

**end while**

**Repräsentation (Grundidee):**
Verwende Formeln statt Zustandsmengen!
Wie sind $\varnothing, \cup, \subseteq$, und $Post$ realisiert?

# Symbolische Erreichbarkeitsanalyse

**Neuer Algorithmus**

$\sigma^R := \varnothing$ schon erreicht

$\tau := \sigma^I$ noch zu besuchen

**while** not $\tau \subseteq \sigma^R$ **do**

  $\sigma^R := \sigma^R \cup \tau$

  $\tau := Post(\tau)$

**end while**

**Repräsentation (Grundidee):**
Verwende Formeln statt Zustandsmengen!
Wie sind $\varnothing, \cup, \subseteq$, und $Post$ realisiert?

Einfach:

- $\varnothing$ $\rightsquigarrow$ `false`
- $\phi_1 \cup \phi_2$ $\rightsquigarrow$ $\phi_1 \vee \phi_2$
- $\phi_1 \subseteq \phi_2$ $\rightsquigarrow$ $\phi_1 \Longrightarrow \phi_2$
  $\rightsquigarrow$ $\phi_1 \wedge \neg\phi_2$ unerfüllbar

# Symbolische Erreichbarkeitsanalyse

**Neuer Algorithmus**

$\sigma^R := \varnothing$ schon erreicht

$\tau := \sigma^I$ noch zu besuchen

**while** not $\tau \subseteq \sigma^R$ **do**

  $\sigma^R := \sigma^R \cup \tau$

  $\tau := Post(\tau)$

**end while**

**Repräsentation (Grundidee):**
Verwende Formeln statt Zustandsmengen!
Wie sind $\varnothing, \cup, \subseteq$, und $Post$ realisiert?

Einfach:

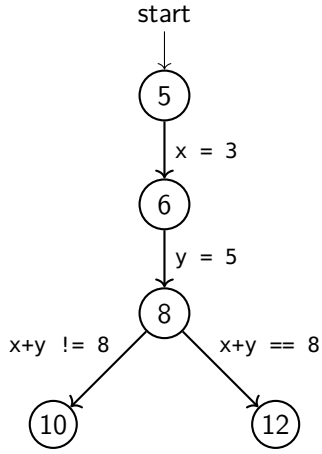
- $\varnothing$ $\rightsquigarrow$ `false`
- $\phi_1 \cup \phi_2$ $\rightsquigarrow$ $\phi_1 \vee \phi_2$
- $\phi_1 \subseteq \phi_2$ $\rightsquigarrow$ $\phi_1 \Longrightarrow \phi_2$
  $\rightsquigarrow$ $\phi_1 \wedge \neg\phi_2$ unerfüllbar

Noch unklar:

- $Post(\phi)$

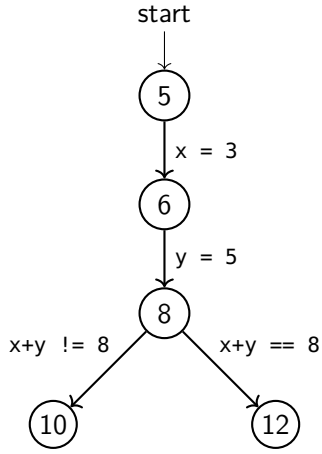
# Beispiel: Symbolische Erreichbarkeit

$$\sigma^R := \quad (\mathsf{pc} = 5)$$



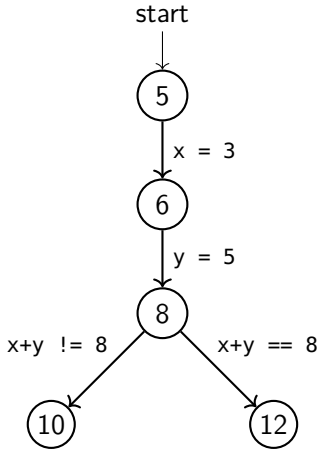
# Beispiel: Symbolische Erreichbarkeit

$$\sigma^R := \quad (\mathsf{pc} = 5) \\ \vee\, (\mathsf{pc} = 6 \wedge \mathsf{x} = 3)$$



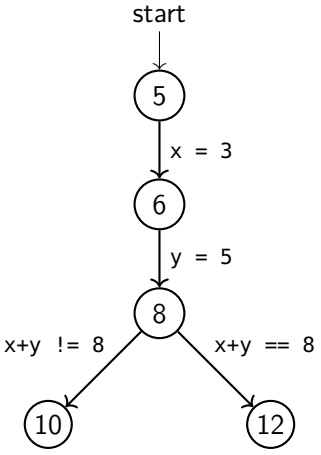
# Beispiel: Symbolische Erreichbarkeit

$$\begin{aligned}\sigma^R := \quad & (\mathsf{pc} = 5) \\ & \vee (\mathsf{pc} = 6 \wedge \mathsf{x} = 3) \\ & \vee (\mathsf{pc} = 8 \wedge \mathsf{x} = 3 \wedge \mathsf{y} = 5)\end{aligned}$$



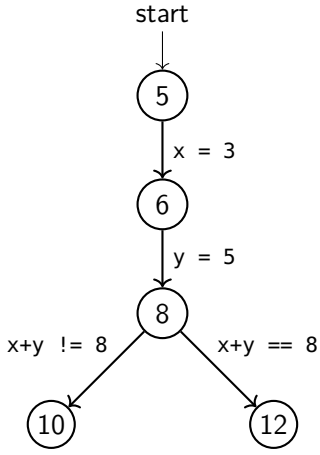
# Beispiel: Symbolische Erreichbarkeit

$$\begin{aligned}\sigma^R := \quad & (\mathsf{pc} = 5) \\ & \vee (\mathsf{pc} = 6 \wedge \mathsf{x} = 3) \\ & \vee (\mathsf{pc} = 8 \wedge \mathsf{x} = 3 \wedge \mathsf{y} = 5) \\ & \vee (\mathsf{pc} = 10 \wedge \mathsf{x} = 3 \wedge \mathsf{y} = 5 \wedge \mathsf{x} + \mathsf{y} \neq 8)\end{aligned}$$



# Beispiel: Symbolische Erreichbarkeit

$$
\begin{aligned}
\sigma^R := \quad & (\mathsf{pc} = 5) \\
& \vee (\mathsf{pc} = 6 \wedge \mathsf{x} = 3) \\
& \vee (\mathsf{pc} = 8 \wedge \mathsf{x} = 3 \wedge \mathsf{y} = 5) \\
& \vee (\mathsf{pc} = 10 \wedge \mathsf{x} = 3 \wedge \mathsf{y} = 5 \wedge \mathsf{x} + \mathsf{y} \neq 8) \\
& \vee (\mathsf{pc} = 12 \wedge \mathsf{x} = 3 \wedge \mathsf{y} = 5 \wedge \mathsf{x} + \mathsf{y} = 8)
\end{aligned}
$$

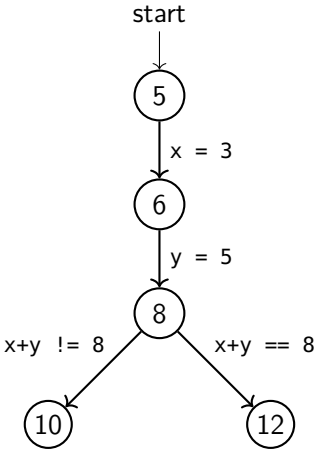


# Beispiel: Symbolische Erreichbarkeit

$$
\begin{aligned}
\sigma^R := \quad & (\mathsf{pc} = 5) \\
& \vee (\mathsf{pc} = 6 \wedge \mathsf{x} = 3) \\
& \vee (\mathsf{pc} = 8 \wedge \mathsf{x} = 3 \wedge \mathsf{y} = 5) \\
& \vee (\mathsf{pc} = 10 \wedge \mathsf{x} = 3 \wedge \mathsf{y} = 5 \wedge \mathsf{x} + \mathsf{y} \neq 8) \\
& \vee (\mathsf{pc} = 12 \wedge \mathsf{x} = 3 \wedge \mathsf{y} = 5 \wedge \mathsf{x} + \mathsf{y} = 8)
\end{aligned}
$$

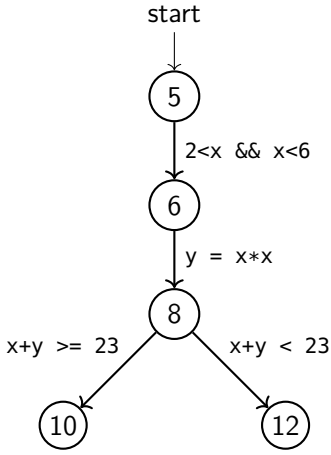
Analog zur expliziten Erreichbarkeit, allerdings

- $\mathsf{pc} = 5$ entspricht $\mathsf{pc} = 5 \wedge \mathsf{x} \in \mathbb{Z} \wedge \mathsf{y} \in \mathbb{Z}$: Startwerte können beliebig sein
- Zustand mit $\mathsf{pc} = 10$ ist nicht erreichbar: Teilformel ist unerfüllbar ( $\Longleftrightarrow$ `false`)



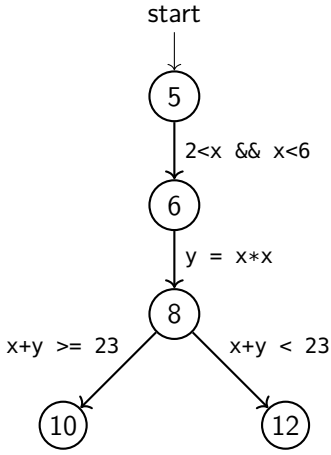
# Beispiel: Symbolische Erreichbarkeit

$$\sigma^R := \quad (\mathsf{pc} = 5)$$



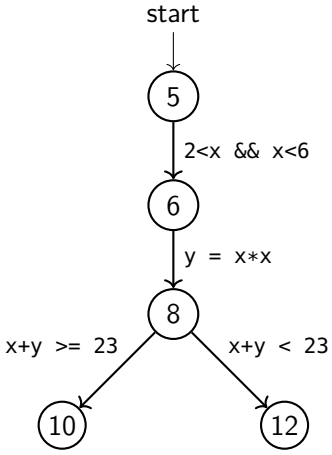
# Beispiel: Symbolische Erreichbarkeit

$$\sigma^R := \quad (\mathsf{pc} = 5)$$
$$\vee\, (\mathsf{pc} = 6 \wedge 2 < \mathsf{x} < 6)$$



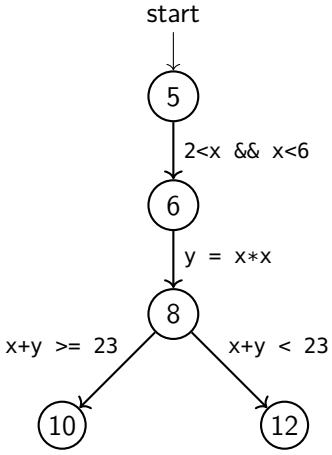
# Beispiel: Symbolische Erreichbarkeit

$$\begin{aligned}\sigma^R := \quad & (\mathsf{pc} = 5) \\ & \vee (\mathsf{pc} = 6 \wedge 2 < \mathsf{x} < 6) \\ & \vee (\mathsf{pc} = 8 \wedge 2 < \mathsf{x} < 6 \wedge \mathsf{y} = \mathsf{x} * \mathsf{x})\end{aligned}$$



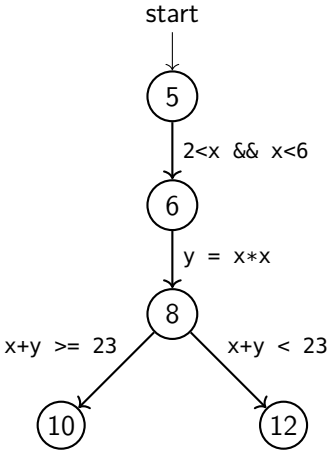
# Beispiel: Symbolische Erreichbarkeit

$$
\begin{aligned}
\sigma^R := \quad & (\mathsf{pc} = 5) \\
& \vee (\mathsf{pc} = 6 \wedge 2 < \mathsf{x} < 6) \\
& \vee (\mathsf{pc} = 8 \wedge 2 < \mathsf{x} < 6 \wedge \mathsf{y} = \mathsf{x} * \mathsf{x}) \\
& \vee (\mathsf{pc} = 10 \wedge 2 < \mathsf{x} < 6 \wedge \mathsf{y} = \mathsf{x} * \mathsf{x} \wedge \mathsf{x} + \mathsf{y} \geq 23)
\end{aligned}
$$



# Beispiel: Symbolische Erreichbarkeit

$$
\begin{aligned}
\sigma^R := \quad & (\mathsf{pc} = 5) \\
& \vee (\mathsf{pc} = 6 \wedge 2 < \mathsf{x} < 6) \\
& \vee (\mathsf{pc} = 8 \wedge 2 < \mathsf{x} < 6 \wedge \mathsf{y} = \mathsf{x} * \mathsf{x}) \\
& \vee (\mathsf{pc} = 10 \wedge 2 < \mathsf{x} < 6 \wedge \mathsf{y} = \mathsf{x} * \mathsf{x} \wedge \mathsf{x} + \mathsf{y} \geq 23) \\
& \vee (\mathsf{pc} = 12 \wedge 2 < \mathsf{x} < 6 \wedge \mathsf{y} = \mathsf{x} * \mathsf{x} \wedge \mathsf{x} + \mathsf{y} < 23)
\end{aligned}
$$



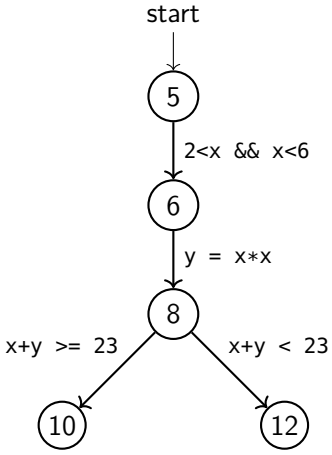
# Beispiel: Symbolische Erreichbarkeit

$$
\begin{aligned}
\sigma^R := \quad & (\mathsf{pc} = 5) \\
& \vee (\mathsf{pc} = 6 \wedge 2 < \mathsf{x} < 6) \\
& \vee (\mathsf{pc} = 8 \wedge 2 < \mathsf{x} < 6 \wedge \mathsf{y} = \mathsf{x} * \mathsf{x}) \\
& \vee (\mathsf{pc} = 10 \wedge 2 < \mathsf{x} < 6 \wedge \mathsf{y} = \mathsf{x} * \mathsf{x} \wedge \mathsf{x} + \mathsf{y} \geq 23) \\
& \vee (\mathsf{pc} = 12 \wedge 2 < \mathsf{x} < 6 \wedge \mathsf{y} = \mathsf{x} * \mathsf{x} \wedge \mathsf{x} + \mathsf{y} < 23)
\end{aligned}
$$

Finde Lösungen mit Constraint-Solvern, z.B.:

- $\sigma^R \wedge \mathsf{pc} = 10 \rightsquigarrow s = \{\mathsf{x} \mapsto 5, \mathsf{y} \mapsto 25\}$
- $\sigma^R \wedge \mathsf{pc} = 12 \rightsquigarrow s = \{\mathsf{x} \mapsto 3, \mathsf{y} \mapsto 9\}$

Skaliert für *linearer Arithmetik* auf sehr große Probleme



# Beispiel: Lösen von Constraints mit SMT Solvern

## Eingabedatei `constraints.smt2`

```
(set-logic ALL)
(set-option :produce-models true)

(declare-const pc Int)
(declare-const x  Int)
(declare-const y  Int)

(assert
    (or (and (= pc  5))
        (and (= pc  6) (< 2 x) (< x 6))
        (and (= pc  8) (< 2 x) (< x 6) (= y (* x x)))
        (and (= pc 10) (< 2 x) (< x 6) (= y (* x x)) (>= (+ x y) 23))
        (and (= pc 12) (< 2 x) (< x 6) (= y (* x x)) (<  (+ x y) 23))))

(push)
    (assert (= pc 10))
    (check-sat)
    (get-model)
(pop)

(push)
    (assert (= pc 12))
    (check-sat)
    (get-model)
(pop)
```

## Ausführung

```
z3 constraints.smt2
```

Ausgabe:

```
sat
(model
  (define-fun pc () Int
    10)
  (define-fun y () Int
    25)
  (define-fun x () Int
    5)
)
sat
(model
  (define-fun pc () Int
    12)
  (define-fun y () Int
    9)
  (define-fun x () Int
    3)
)
```

# Informell: Symbolische Berechnung von Nachfolgezuständen

Offene Frage: Berechnung von $Post$

# Informell: Symbolische Berechnung von Nachfolgezuständen

Offene Frage: Berechnung von $Post$

Zuweisung als Gleichung?

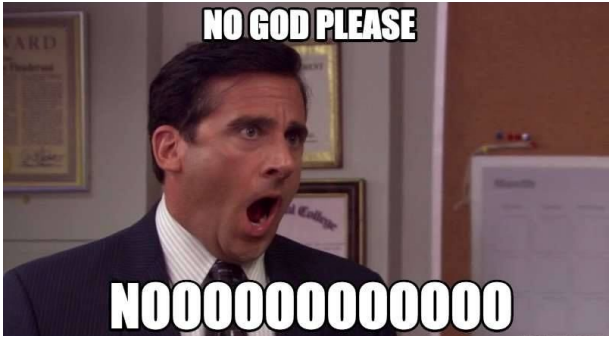
$$\mathsf{x} \geq 0 \xrightarrow{\texttt{x = x+1}} \mathsf{x} \geq 0 \wedge \mathsf{x} = \mathsf{x} + 1$$

# Informell: Symbolische Berechnung von Nachfolgezuständen

Offene Frage: Berechnung von $Post$

Zuweisung als Gleichung?

$$\mathsf{x} \geq 0 \xrightarrow{\mathsf{x = x+1}} \mathsf{x} \geq 0 \wedge \mathsf{x} = \mathsf{x} + 1$$




Problem: unterschiedliche Versionen von x verglichen → Formel wird unerfüllbar

# Informell: Symbolische Berechnung von Nachfolgezuständen

## Mögliche Ideen (Zuweisung)

Intuitiv

$$x \geq 0 \xrightarrow{x = x+1} x > 0 \qquad \text{oder} \qquad x \geq 0 \xrightarrow{x = x+1} x - 1 \geq 0$$

0

# Informell: Symbolische Berechnung von Nachfolgezuständen

**Mögliche Ideen (Zuweisung)**

Intuitiv (✓ reicht für Übung, ✗ nicht systematisch)

$x \geq 0 \xrightarrow{\texttt{x = x+1}} x > 0$ oder $x \geq 0 \xrightarrow{\texttt{x = x+1}} x - 1 \geq 0$

0

# Informell: Symbolische Berechnung von Nachfolgezuständen

**Mögliche Ideen (Zuweisung)**

Intuitiv (✓ reicht für Übung, ✗ nicht systematisch)

$$x \geq 0 \xrightarrow{\texttt{x = x+1}} x > 0 \qquad \text{oder} \qquad x \geq 0 \xrightarrow{\texttt{x = x+1}} x - 1 \geq 0$$

Eine Variable pro Programmstelle

$$x_\ell \geq 0 \xrightarrow{\texttt{x = x+1}} x_\ell > 0 \wedge x_{\ell'} = x_\ell + 1$$

1

# Informell: Symbolische Berechnung von Nachfolgezuständen

**Mögliche Ideen (Zuweisung)**

Intuitiv (✓ reicht für Übung, ✗ nicht systematisch)

$$x \geq 0 \xrightarrow{\texttt{x = x+1}} x > 0 \qquad \text{oder} \qquad x \geq 0 \xrightarrow{\texttt{x = x+1}} x - 1 \geq 0$$

Eine Variable pro Programmstelle (✗ Schleifen gehen so nicht[1])

$$x_\ell \geq 0 \xrightarrow{\texttt{x = x+1}} x_\ell > 0 \wedge x_{\ell'} = x_\ell + 1$$

[1] Zum Nachdenken: welcher (ganz einfache) Schleifencode zeigt das Problem?

# Informell: Symbolische Berechnung von Nachfolgezuständen

**Mögliche Ideen (Zuweisung)**

Intuitiv (✓ reicht für Übung, ✗ nicht systematisch)

$$x \geq 0 \xrightarrow{x = x+1} x > 0 \qquad \text{oder} \qquad x \geq 0 \xrightarrow{x = x+1} x - 1 \geq 0$$

Eine Variable pro Programmstelle (✗ Schleifen gehen so nicht[1])

$$x_\ell \geq 0 \xrightarrow{x = x+1} x_\ell > 0 \wedge x_{\ell'} = x_\ell + 1$$

Neue Variablen für vorige Werte einführen

$$x \geq 0 \xrightarrow{x = x+1} x_0 > 0 \wedge x = x_0 + 1$$

[1] Zum Nachdenken: welcher (ganz einfache) Schleifencode zeigt das Problem?

# Informell: Symbolische Berechnung von Nachfolgezuständen

**Mögliche Ideen (Zuweisung)**

Intuitiv (✓ reicht für Übung, ✗ nicht systematisch)

$$x \geq 0 \xrightarrow{x = x+1} x > 0 \qquad \text{oder} \qquad x \geq 0 \xrightarrow{x = x+1} x - 1 \geq 0$$

Eine Variable pro Programmstelle (✗ Schleifen gehen so nicht[1])

$$x_\ell \geq 0 \xrightarrow{x = x+1} x_\ell > 0 \wedge x_{\ell'} = x_\ell + 1$$

Neue Variablen für vorige Werte einführen (**ok** aber viele Variablen)

$$x \geq 0 \xrightarrow{x = x+1} x_0 > 0 \wedge x = x_0 + 1$$

[1] Zum Nachdenken: welcher (ganz einfache) Schleifencode zeigt das Problem?

# Informell: Symbolische Berechnung von Nachfolgezuständen

**Mögliche Ideen (Zuweisung)**

Intuitiv (✓ reicht für Übung, ✗ nicht systematisch)

$$x \geq 0 \xrightarrow{\texttt{x = x+1}} x > 0 \qquad \text{oder} \qquad x \geq 0 \xrightarrow{\texttt{x = x+1}} x - 1 \geq 0$$

Systematisch:

- Symbolic execution (nicht hier)
- Hoare-Logik ($\rightarrow$ später)

0

# Informell: Symbolische Berechnung von Nachfolgezuständen

**Bedingungen** (von `if`, `while`, `assume`)

# Informell: Symbolische Berechnung von Nachfolgezuständen

**Bedingungen** (von `if`, `while`, `assume`)

Beachte: keine Änderung der Variablen!

$$\phi \xrightarrow{\psi} ???$$

# Informell: Symbolische Berechnung von Nachfolgezuständen

**Bedingungen** (von `if`, `while`, `assume`)

Beachte: keine Änderung der Variablen!

Einfache Hinzunahme der neuen Bedingungen ✓

$$\phi \xrightarrow{\psi} \phi \wedge \psi$$

# Was man wissen sollte

- Welchen Vorteil hat die symbolische Zustandsrepräsentation?
- Wie werden Mengenoperationen auf Formeln abgebildet?
- In wie fern ist $Post$ auf Formeln etwas schwierig?
- Welche Tools können Lösungen für Formeln finden?
- Zum Nachdenken: Wie könnte man die Algorithmen zur expliziten und symbolischen Suche optimieren, wenn eine *Zielregion* als Spezifikation bekannt ist z.B. bestimmte Zustände, oder *Invarianten* die in jedem Zustand gelten sollen?

# License

These slides are licensed under the creative commons license:
https://creativecommons.org/licenses/by-nc-nd/4.0/

Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0)

- give appropriate credit
- distribute without modifications
- do not use for commercial purposes