# Formale Spezifikation und Verifikation

## Wintersemester 2024

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

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

November 4, 2024

# Testen

Testen

# Kontrollflussautomaten und Abdeckungsmaße

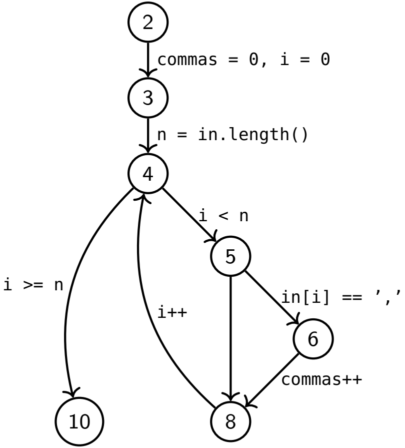
# Repräsentation von Programmen

# Repräsentation von Programmen

## Programm

```java
int countCommas(String in) {
  int commas = 0, i = 0
  int n = in.length();
  while(i < n) {
    if(in.charAt(i) == ',') {
      commas++;
    }
    i++;
  }
  return commas ;
}
```

## Kontrollflussautomat



# Repräsentation von Programmen

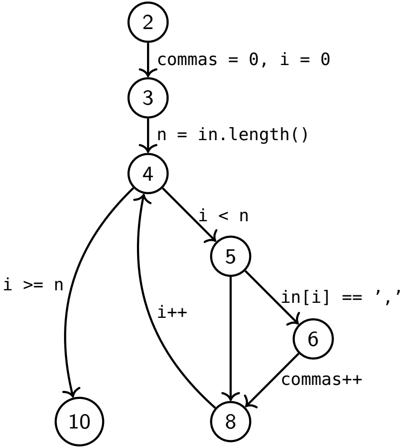
### Systematische Konstruktion

- Knoten = Programmstellen
- Kanten = Anweisungen & Tests
- Schleifen → Zyklen

✓ Gut geeignet, um die Ausführung von Programmen zu beschreiben und zu analysieren!

(Werbung für den Master
Beyer: Software Analysis and Verification)

### Kontrollflussautomat

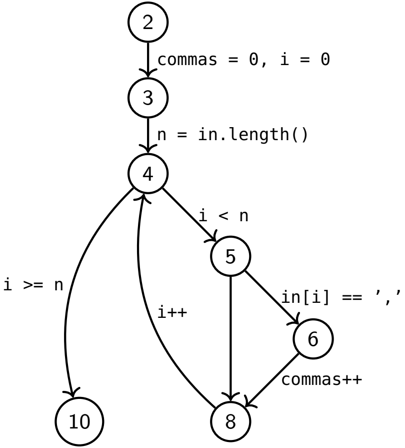


# Verfolgen von Ausführungen

## Für gegebene Eingaben

- `in = ""`

## Kontrollflussautomat

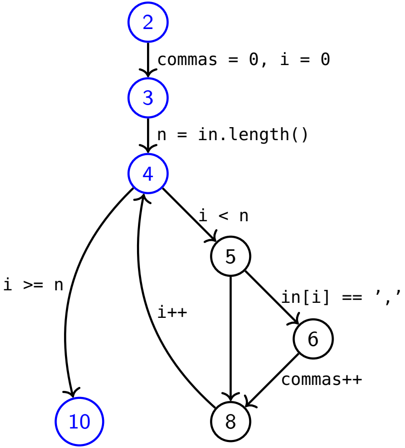


# Verfolgen von Ausführungen

## Für gegebene Eingaben

- `in = ""`
  - besuchte Zeilen: 4

## Kontrollflussautomat

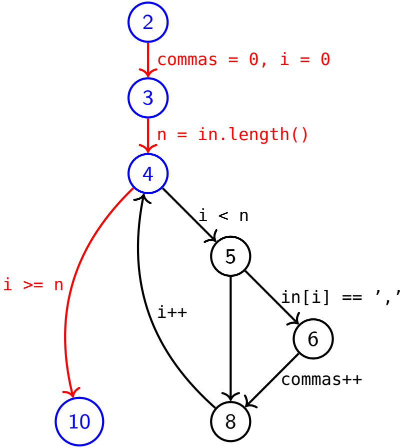


# Verfolgen von Ausführungen

## Für gegebene Eingaben

- `in = ""`
  - besuchte Zeilen: 4
  - verfolgte Kanten: 3

## Kontrollflussautomat

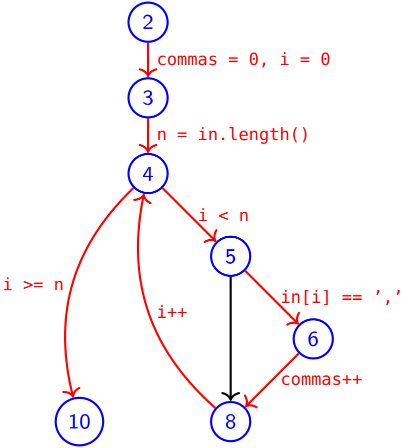


# Verfolgen von Ausführungen

## Für gegebene Eingaben

- `in = ""`
  - besuchte Zeilen: 4
  - verfolgte Kanten: 3
- `in = ","`
  - besuchte Zeilen: 7 (100%)
  - verfolgte Kanten: 7

## Kontrollflussautomat

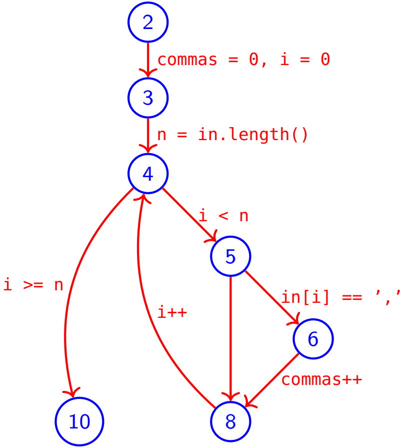


# Verfolgen von Ausführungen

## Für gegebene Eingaben

- `in = ""`
  - besuchte Zeilen: 4
  - verfolgte Kanten: 3
- `in = ","`
  - besuchte Zeilen: 7 (100%)
  - verfolgte Kanten: 7
- `in = "0,nix"`
  - besuchte Zeilen: 7 (100%)
  - verfolgte Kanten: 8 (100%)

## Kontrollflussautomat



# Überdeckungsmaße (Coverage)

- Anweisungsüberdeckung (statement coverage):
  wurden alle Programmstellen wurden besucht? (= Knoten im CFA)
- Zweigüberdeckung (branch coverage):
  wurden alle Fallunterscheidungen genommen? (= Kanten im CFA)
- Pfadüberdeckung (path coverage):
  wurden alle möglichen Pfade durch das Programm ausgeführt?
  (allgemein: unendlich viele)

# Überdeckungsmaße (Coverage)

- Anweisungsüberdeckung (statement coverage):
  wurden alle Programmstellen wurden besucht? (= Knoten im CFA)
- Zweigüberdeckung (branch coverage):
  wurden alle Fallunterscheidungen genommen? (= Kanten im CFA)
- Pfadüberdeckung (path coverage):
  wurden alle möglichen Pfade durch das Programm ausgeführt?
  (allgemein: unendlich viele)

Achtung: Oft wird coverage auf dem Kontrollfluss*graph* gemessen (Anweisungen in den Knoten).
Dabei sind die Definitionen leicht anders, aber die Maße im Wesentlichen gleich.

# Diskussion: Abdeckungsmaße

- Anweisungsüberdeckung & Zweigüberdeckung
  (+ Varianten wie Bedingungsüberdeckung)
  - ✓ verständlich
  - ✓ messbar: tatsächliche und bestmögliche Abdeckung
  - ✗ keine Aussagen über funktionale Eigenschaften

  Beachte: 100% nicht immer erreichbar (dead code)

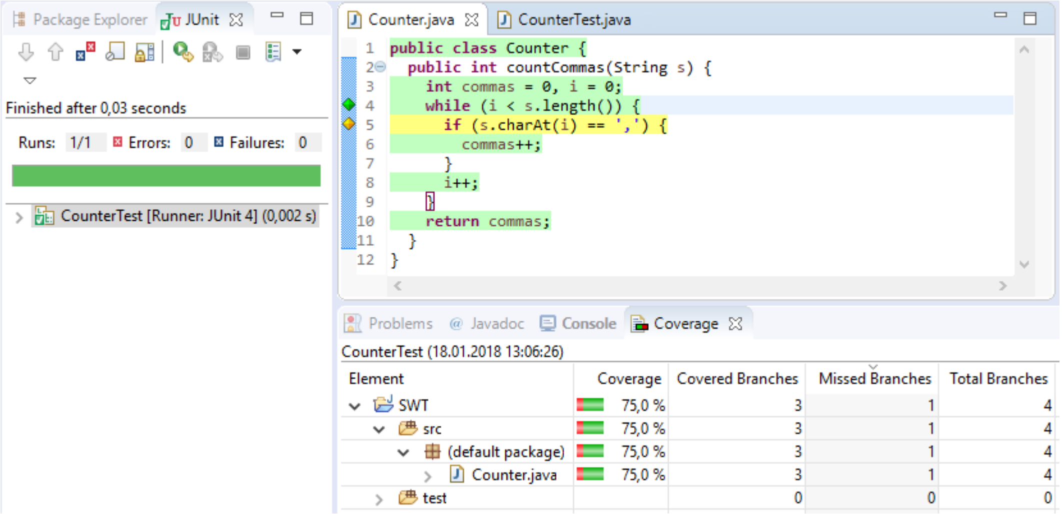
# Diskussion: Abdeckungsmaße

- Anweisungsüberdeckung & Zweigüberdeckung
  (+ Varianten wie Bedingungsüberdeckung)
  - ✓ verständlich
  - ✓ messbar: tatsächliche und bestmögliche Abdeckung
  - ✗ keine Aussagen über funktionale Eigenschaften

  Beachte: 100% nicht immer erreichbar (dead code)

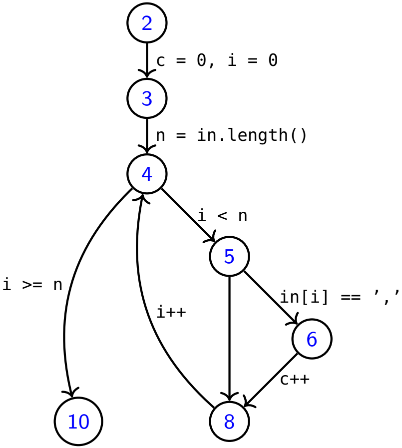
- Pfadüberdeckung:
  - ✓ semantisches Kriterium, genau die tatsächlich möglichen Ausführungen
  - ✓ Grundlage für *beweisbare Korrektheit*
  - ✗ für die meisten Systeme nicht erreichbar (Schleifen, Eingaben)

# Kontrollflussbasiertes Testen in Eclipse

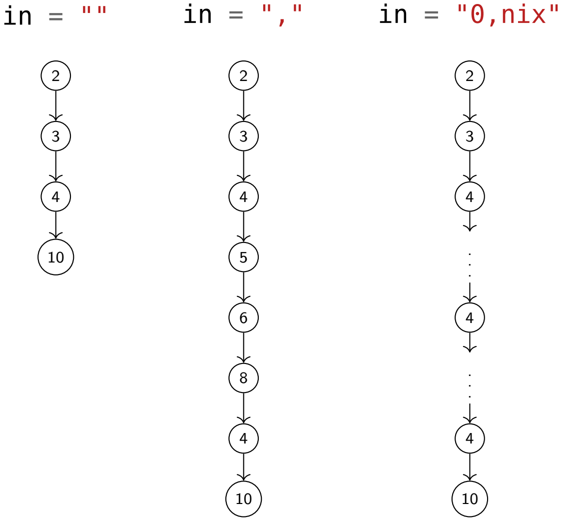


# Programmausführungen: veranschaulicht

## Kontrollflussautomat

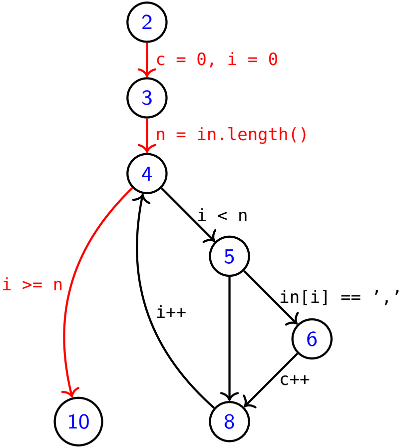



## Pfade



# Programmausführungen: veranschaulicht

**Kontrollflussautomat**

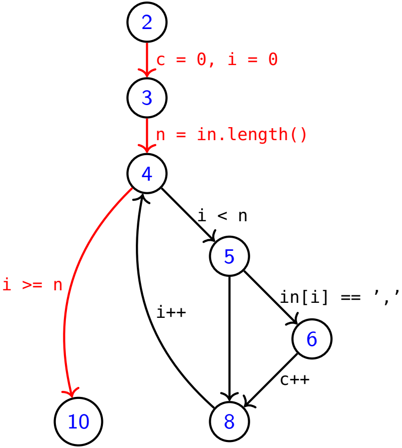



**Pfade & Zustände zur Laufzeit**

$$\{\texttt{pc} \mapsto 2, \texttt{in} \mapsto \texttt{""}\}$$

# Programmausführungen: veranschaulicht

## Kontrollflussautomat



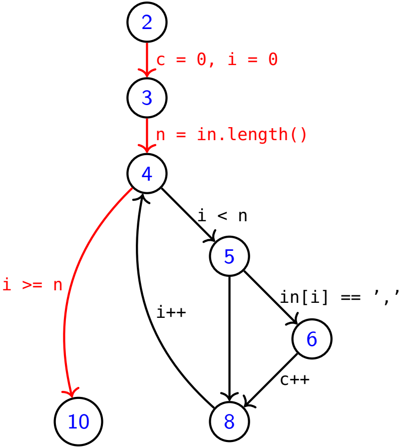

## Pfade & Zustände zur Laufzeit

$$\{\texttt{pc} \mapsto 2, \texttt{in} \mapsto \texttt{""}\}$$

$$\{\texttt{pc} \mapsto 3, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0\}$$

# Programmausführungen: veranschaulicht

## Kontrollflussautomat




## Pfade & Zustände zur Laufzeit

$$\{\texttt{pc} \mapsto 2, \texttt{in} \mapsto \texttt{""}\}$$

$$\downarrow$$

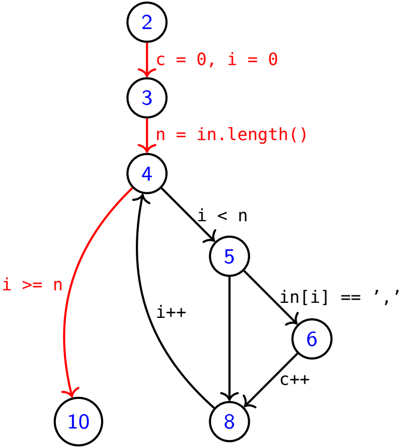
$$\{\texttt{pc} \mapsto 3, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0\}$$

$$\downarrow$$

$$\{\texttt{pc} \mapsto 4, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0, \texttt{n} \mapsto 0\}$$

# Programmausführungen: veranschaulicht

## Kontrollflussautomat




## Pfade & Zustände zur Laufzeit

$$\{\texttt{pc} \mapsto 2, \texttt{in} \mapsto \texttt{""}\}$$

$$\downarrow$$

$$\{\texttt{pc} \mapsto 3, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0\}$$

$$\downarrow$$

$$\{\texttt{pc} \mapsto 4, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0, \texttt{n} \mapsto 0\}$$

$$\downarrow$$

$$\{\texttt{pc} \mapsto 10, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0, \texttt{n} \mapsto 0\}$$

# Was man wissen sollte

- Programme $\leftrightarrow$ Kontrollflussautomaten:
  Knoten $\approx$ Programmzeilen, Kanten $=$ Tests & Anweisungen
- Welche Abdeckungsmaße gibts es? Vor/Nachteile?
- Wie sind die Abdeckungsmaße bezgl Kontrollflussautomaten definiert?
- *Zum Nachdenken*: Bei welchen Programmen/Systemen macht 100% Pfadabdeckung Sinn?
- Ausblick: Formale Definition von
  - Programm als Kontrollflussautomat
  - Ausführungen deines Programms als Transitionssystem
  - Zustände und Ausführungspfade

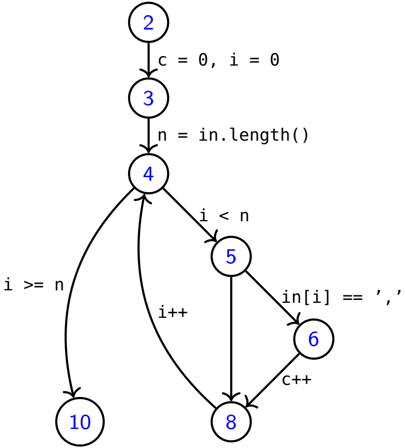
# Erreichbarkeitsanalysen

Erreichbarkeitsanalysen

# Transitionssysteme & Erreichbarkeit

# Repräsentation von Programmen

## Kontrollflussautomat

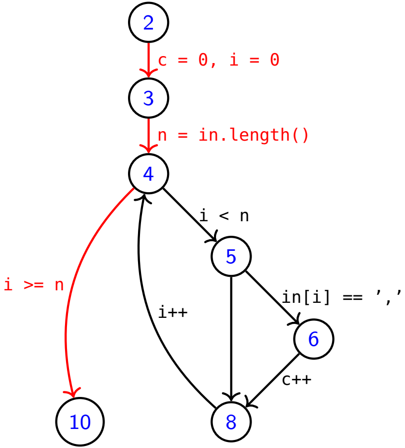



## Programm

```
int countCommas(String in) {
  int commas = 0, i = 0
  int n = in.length();
  while(i < n) {
    if(in.charAt(i) == ',') {
      commas++;
    }
    i++;
  }
  return commas ;
}
```

# Ausführung von Programmen

**Kontrollflussautomat**
statische Sicht




**Pfade & Zustände zur Laufzeit**
dynamische Sicht

$$\{\texttt{pc} \mapsto 2, \texttt{in} \mapsto \texttt{""}\}$$

$$\downarrow$$

$$\{\texttt{pc} \mapsto 3, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0\}$$

$$\downarrow$$

$$\{\texttt{pc} \mapsto 4, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0, \texttt{n} \mapsto 0\}$$

$$\downarrow$$

$$\{\texttt{pc} \mapsto 10, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0, \texttt{n} \mapsto 0\}$$

# Heute

*Formale* Definition von

- Syntax ($=$ statische Sicht):
  Kontrollflussautomaten als Programmrepräsentation
- Semantik ($=$ dynamische Sicht):
  Transitionssysteme zur Beschreibung von *Ausführungen*

# Heute

*Formale* Definition von

- Syntax (= statische Sicht):
  Kontrollflussautomaten als Programmrepräsentation
- Semantik (= dynamische Sicht):
  Transitionssysteme zur Beschreibung von *Ausführungen*

*Spezifikation* von Eigenschaften

- Invariante = erwünschte Eigenschaft aller *erreichbaren* Zustände
- insbesondere: Zielzustände/Fehlerzustände

# Heute

*Formale* Definition von

- Syntax (= statische Sicht):
  Kontrollflussautomaten als Programmrepräsentation
- Semantik (= dynamische Sicht):
  Transitionssysteme zur Beschreibung von *Ausführungen*

*Spezifikation* von Eigenschaften

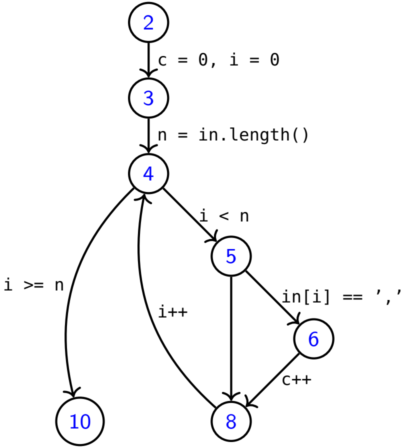
- Invariante = erwünschte Eigenschaft aller *erreichbaren* Zustände
- insbesondere: Zielzustände/Fehlerzustände

*Verifikation*

- *Erreichbarkeitsanalyse* durch explizites Aufzählen (2. Vorlesungsteil)
  (klassisches "model checking", [Emerson&Clarke; Sifakis, 1981])

# Kontrollflussautomaten

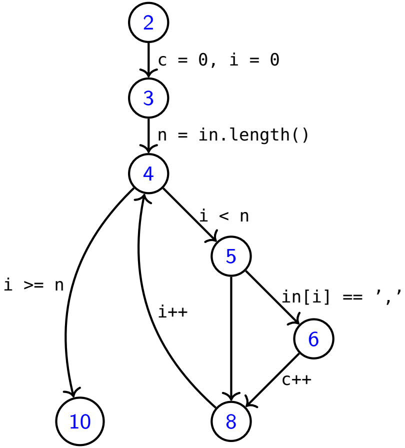
**Grafische Darstellung**




**Formale Definition** $P = (L, \ell_0, G)$

# Kontrollflussautomaten

**Grafische Darstellung**

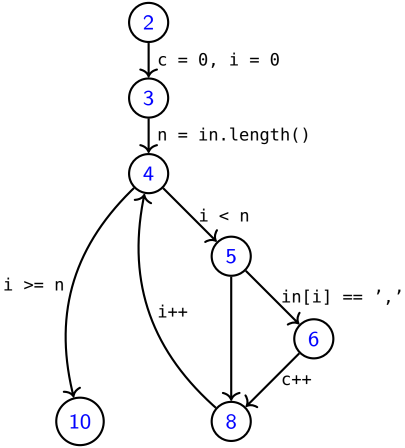



**Formale Definition** $P = (L, \ell_0, G)$

- Programmstellen (Knoten, $\approx$ Zeilen): $L = \{\ell_0, \ell_1, \dots, \ell_n\}$
  Beispiel: $L = \{2, 3, 4, 5, 6, 8, 10\}$
- Startknoten: $\ell_0$, z.B. $\ell_0 = 2$

# Kontrollflussautomaten

**Grafische Darstellung**

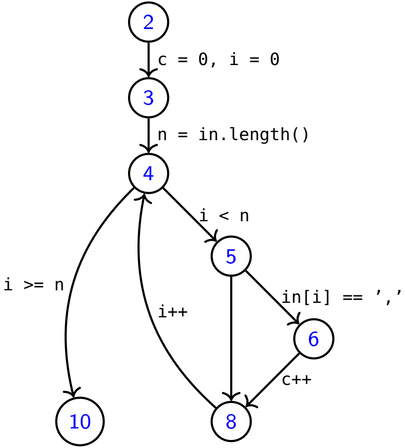



**Formale Definition** $P = (L, \ell_0, G)$

- Programmstellen (Knoten, $\approx$ Zeilen):
  $L = \{\ell_0, \ell_1, \ldots, \ell_n\}$
  Beispiel: $L = \{$2, 3, 4, 5, 6, 8, 10$\}$
- Startknoten: $\ell_0$, z.B. $\ell_0 =$ 2
- Anweisungen & Tests (Kanten):
  $G \subseteq L \times Ops \times L$
  - Zuweisungen $\ell_i \xrightarrow{x=e} \ell_j \in G$
  - Bedingungen $\ell_i \xrightarrow{\phi} \ell_j \in G$

# Kontrollflussautomaten

## Grafische Darstellung




## Mengen (formal, implementierungsnah)

$$P = (L, \ell_0, G), \quad \text{wobei}$$

$$L = \{2, 3, 4, 5, 6, 8, 10\} \quad \text{mit} \quad \ell_0 = 2$$

$$G = \Big\{ 2 \xrightarrow{\texttt{c = 0, i = 0}} 3,$$

$$3 \xrightarrow{\texttt{n = in.length}} 4,$$

$$4 \xrightarrow{\texttt{i < n}} 5, \quad 4 \xrightarrow{\texttt{i >= n}} 10,$$

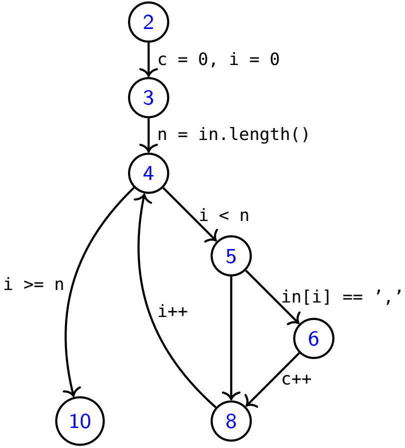
$$5 \xrightarrow{\texttt{in[i] == ','}} 6,$$

$$5 \xrightarrow{\texttt{in[i] != ','}} 8,$$

$$6 \xrightarrow{\texttt{c++}} 8, \quad 8 \xrightarrow{\texttt{i++}} 4 \Big\}$$

# Kontrollflussautomaten

## Grafische Darstellung




## Mengen (formal, implementierungsnah)

$$P = (L, \ell_0, G), \quad \text{wobei}$$

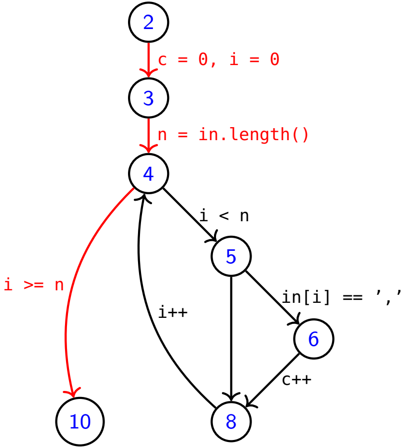
$$L = \{2, 3, 4, 5, 6, 8, 10\} \quad \text{mit} \quad \ell_0 = 2$$

$$\begin{aligned} G = \{ & 2 \xrightarrow{\texttt{c = 0, i = 0}} 3, \\ & 3 \xrightarrow{\texttt{n = in.length}} 4, \\ & 4 \xrightarrow{\texttt{i < n}} 5, \quad 4 \xrightarrow{\texttt{i >= n}} 10, \\ & 5 \xrightarrow{\texttt{in[i] == ','}} 6, \\ & 5 \xrightarrow{\texttt{in[i] != ','}} 8, \\ & 6 \xrightarrow{\texttt{c++}} 8, \quad 8 \xrightarrow{\texttt{i++}} 4 \} \end{aligned}$$

Beachte: Kontrollflussautomaten sind eine *endliche* Beschreibung der Programmstruktur

# Ausführung von Programmen

Startzustand $\{\mathtt{pc} \mapsto (2), \mathtt{in} \mapsto \mathtt{""}\}$

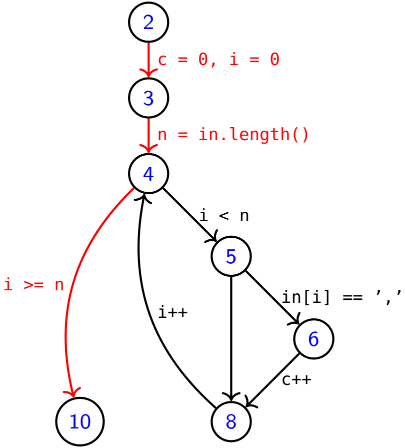



Endzustand

$\{\mathtt{pc} \mapsto (10), \mathtt{in} \mapsto \mathtt{""}, \mathtt{i} \mapsto 0, \mathtt{c} \mapsto 0, \mathtt{n} \mapsto 0\}$

# Ausführung von Programmen

Startzustand $\{\mathsf{pc} \mapsto ②, \mathsf{in} \mapsto \texttt{""}\}$




Endzustand
$\{\mathsf{pc} \mapsto ⑩, \mathsf{in} \mapsto \texttt{""}, \mathsf{i} \mapsto 0, \mathsf{c} \mapsto 0, \mathsf{n} \mapsto 0\}$

**Transitionssystem (informell)**

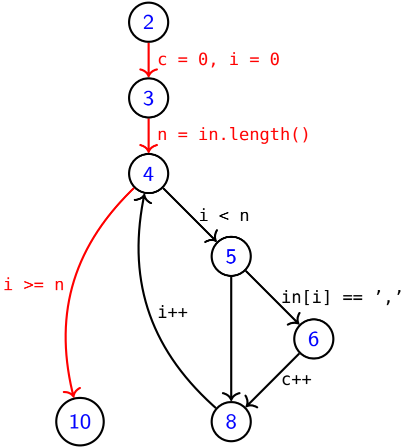
- Potentielle *Zustände*: Programmzähler pc und Variablenbelegungen

$$s = \{\mathsf{pc} \mapsto \ell, \mathsf{in} \mapsto in, \mathsf{i} \mapsto i, \mathsf{c} \mapsto c, \mathsf{n} \mapsto n\}$$

mit $\ell \in L$, $i, c, n \in \mathbb{Z}$, $in$ String,

# Ausführung von Programmen

Startzustand $\{$pc $\mapsto$ 2, in $\mapsto$ ""$\}$




Endzustand
$\{$pc $\mapsto$ 10, in $\mapsto$ "", i $\mapsto 0$, c $\mapsto 0$, n $\mapsto 0\}$

**Transitionssystem (informell)**

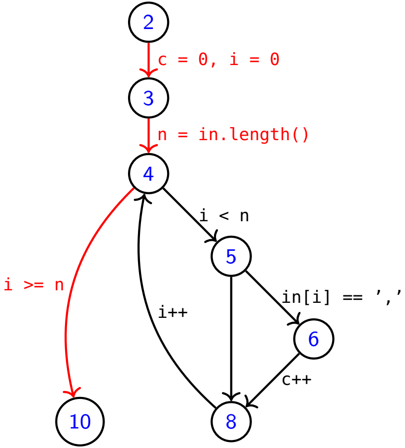
- Potentielle *Zustände*: Programmzähler pc und Variablenbelegungen

  $$s = \{\texttt{pc} \mapsto \ell, \texttt{in} \mapsto in, \texttt{i} \mapsto i, \texttt{c} \mapsto c, \texttt{n} \mapsto n\}$$

  mit $\ell \in L$, $i, c, n \in \mathbb{Z}$, $in$ String,
- Initial $s_0 = \{\texttt{pc} \mapsto \ell_0, \ldots\}$

# Ausführung von Programmen

Startzustand $\{\texttt{pc} \mapsto ②, \texttt{in} \mapsto \texttt{""}\}$




Endzustand
$\{\texttt{pc} \mapsto ⑩, \texttt{in} \mapsto \texttt{""}, \texttt{i} \mapsto 0, \texttt{c} \mapsto 0, \texttt{n} \mapsto 0\}$

**Transitionssystem (informell)**

- Potentielle *Zustände*: Programmzähler pc und Variablenbelegungen

$$s = \{\texttt{pc} \mapsto \ell, \texttt{in} \mapsto in, \texttt{i} \mapsto i, \texttt{c} \mapsto c, \texttt{n} \mapsto n\}$$

  mit $\ell \in L$, $i, c, n \in \mathbb{Z}$, $in$ String,

- Initial $s_0 = \{\texttt{pc} \mapsto \ell_0, \ldots\}$
- Potentielle *Transitionen* $s \longrightarrow s'$ entlang Kanten $\ell \xrightarrow{op} \ell'$

# Transitionssysteme (allgemein & formal)

Transitionssystem $T = (\Sigma, \sigma^I, \to)$ gegeben durch

- Menge $\Sigma$ aller (potentiellen) *Zustände*
- Menge $\sigma^I \subseteq \Sigma$ aller *Startzustände*
- $\to \subseteq \Sigma \times \Sigma$ *Transitionsrelation*

# Transitionssysteme (allgemein & formal)

Transitionssystem $T = (\Sigma, \sigma^I, \rightarrow)$ gegeben durch

- Menge $\Sigma$ aller (potentiellen) *Zustände*
- Menge $\sigma^I \subseteq \Sigma$ aller *Startzustände*
- $\rightarrow \subseteq \Sigma \times \Sigma$ *Transitionsrelation*

*Fundamentaler* Ansatz zur Beschreibung von dynamischen Systemen

- als Semantik von Programmen $P = (L, \ell_0, G)$
  “$\rightarrow$” als Interpreter, z.B.: (stackless) Python
- zur Modellierung, z.B. Kaffeemaschinen
- Varianten: mit Kantenmarkierungen, endliche Automaten

# Transitionssysteme (allgemein & formal)

Transitionssystem $T = (\Sigma, \sigma^I, \to)$ gegeben durch

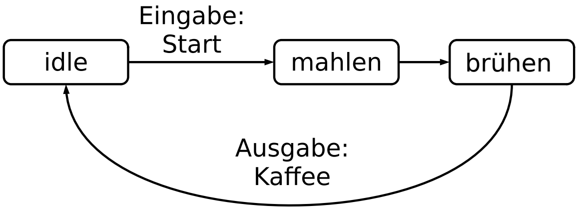
- Menge $\Sigma$ aller (potentiellen) *Zustände*
- Menge $\sigma^I \subseteq \Sigma$ aller *Startzustände*
- $\to \subseteq \Sigma \times \Sigma$ *Transitionsrelation*

*Fundamentaler* Ansatz zur Beschreibung von dynamischen Systemen

- als Semantik von Programmen $P = (L, \ell_0, G)$
  "$\to$" als Interpreter, z.B.: (stackless) Python
- zur Modellierung, z.B. Kaffeemaschinen
- Varianten: mit Kantenmarkierungen, endliche Automaten

Bei interessanten Systemen ist $\Sigma$ sehr groß oder unendlich.
Welche Zustände sind von $\sigma^I$ ausgehend via Transitionen "$\to$" *erreichbar*?

# Beispiel: Kaffeemaschine



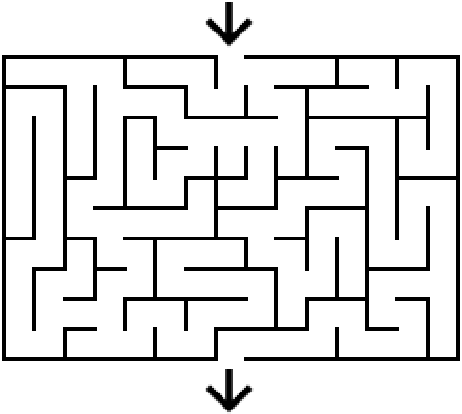

$$T = (\Sigma, \sigma^I, \rightarrow) \quad \text{mit}$$

$$\Sigma = \{\text{idle}, \text{mahlen}, \text{brühen}\}$$

$$\sigma^I = \{\text{idle}\}$$

$$\rightarrow = \{(\text{idle}, \text{mahlen}), (\text{mahlen}, \text{brühen}), (\text{brühen}, \text{idle})\}$$

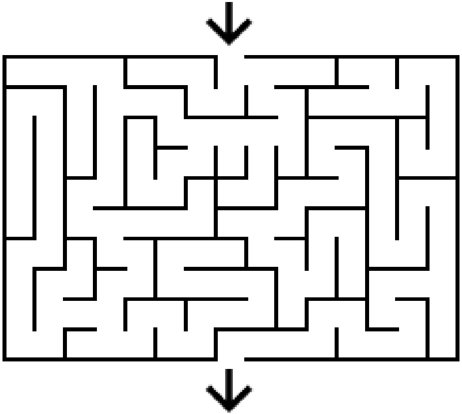
# Beispiel: Labyrinth



Modellierung als Transititionssystem

- $\Sigma =$ Positionen
- $\sigma^I = \{\text{entry}\}$: Eingang
- $\rightarrow$: Korridore

# Beispiel: Labyrinth

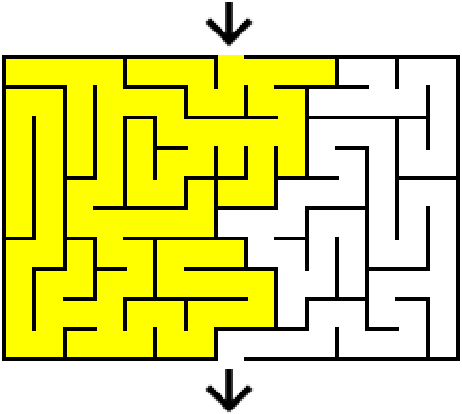


Modellierung als Transititionssystem

- $\Sigma =$ Positionen
- $\sigma^I = \{\text{entry}\}$: Eingang
- $\rightarrow$: Korridore

Gibt es einen Weg durch das Labyrinth?

# Beispiel: Labyrinth



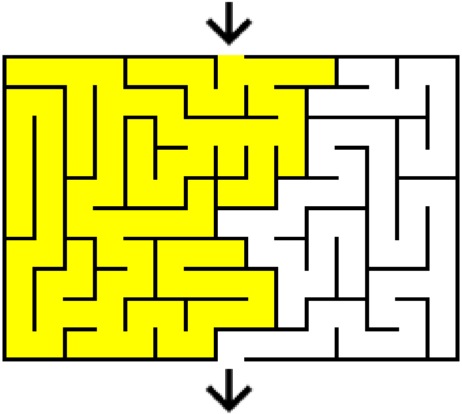
Modellierung als Transititionssystem

- $\Sigma =$ Positionen
- $\sigma^I = \{\text{entry}\}$: Eingang
- $\rightarrow$: Korridore

Gibt es einen Weg durch das Labyrinth?

1. Berechnung *erreichbarer* Zustände: $\sigma^R \subseteq \Sigma$

# Beispiel: Labyrinth



Modellierung als Transititionssystem

- $\Sigma =$ Positionen
- $\sigma^I = \{\text{entry}\}$: Eingang
- $\rightarrow$: Korridore

Gibt es einen Weg durch das Labyrinth?

1. Berechnung *erreichbarer* Zustände: $\sigma^R \subseteq \Sigma$
2. Ist der Ausgang erreichbar: exit $\in \sigma^R$?

Viele Eigenschaften von Systemen lassen sich mit Hilfe von $\sigma^R$ ausdrücken und oft auch einfach bestimmen

# Anwendung auf Programmverifikation

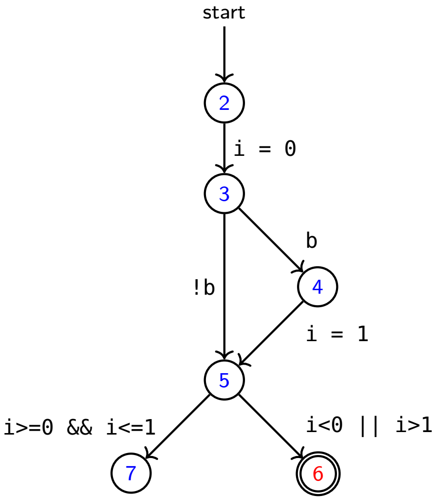
- Gegeben: Programm als Kontrollflussautomat $P = (L, \ell_0, G)$
- Gegeben: Fehlerstelle(n) $\ell_{\text{error}}$
  - Direkt: `throw new Exception()`
  - zusätzliche Fehlertransitionen für Standardfehler
- Verifikation: gilt $\ell_{\text{error}} \in \sigma^R$ im *zugehörigen Transitionssystem*?
  - Ja: Programm fehlerhaft
  - Nein: Programm korrekt
  - Evtl Nichtterminierung wenn unendlich viele Zustände erreichbar sind

# Beispiel: Programmverifikation

```java
int toInt(boolean b) {
    int i = 0;
    if(b)
        i = 1;
    if(i < 0 || i > 1)
        throw new Exception();
    return i;
}
```

# Beispiel: Programmverifikation

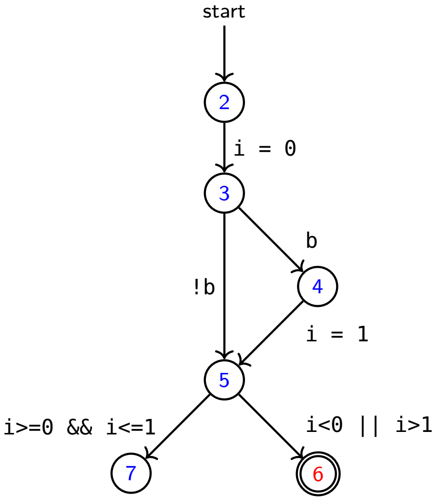
```java
int toInt(boolean b) {
    int i = 0;
    if(b)
        i = 1;
    if(i < 0 || i > 1)
        throw new Exception();
    return i;
}
```



# Beispiel: Programmverifikation

**Initiale Zustände $\sigma^I$**

$\{\mathsf{pc} \mapsto ②, \mathsf{b} \mapsto b\}$ mit $b \in \mathbb{B}$



# Beispiel: Programmverifikation

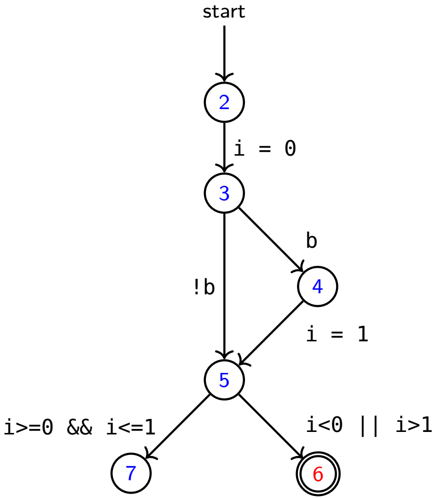
**Initiale Zustände $\sigma^I$**

$\{\texttt{pc} \mapsto ②, \texttt{b} \mapsto b\}$ mit $b \in \mathbb{B}$

**Erreichbare Zustände $\sigma^R$**

$\{\texttt{pc} \mapsto ②, \texttt{b} \mapsto \mathit{false}\}$

$\{\texttt{pc} \mapsto ②, \texttt{b} \mapsto \mathit{true}\}$



# Beispiel: Programmverifikation

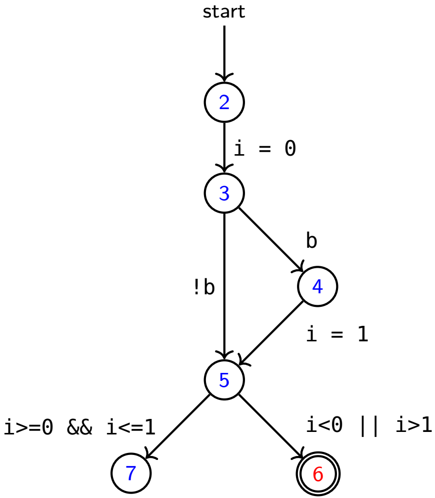
**Initiale Zustände $\sigma^I$**

$\{\texttt{pc} \mapsto ②, \texttt{b} \mapsto b\}$ mit $b \in \mathbb{B}$

**Erreichbare Zustände $\sigma^R$**

$\{\texttt{pc} \mapsto ②, \texttt{b} \mapsto \mathit{false}\}$
$\{\texttt{pc} \mapsto ③, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$

$\{\texttt{pc} \mapsto ②, \texttt{b} \mapsto \mathit{true}\}$



# Beispiel: Programmverifikation

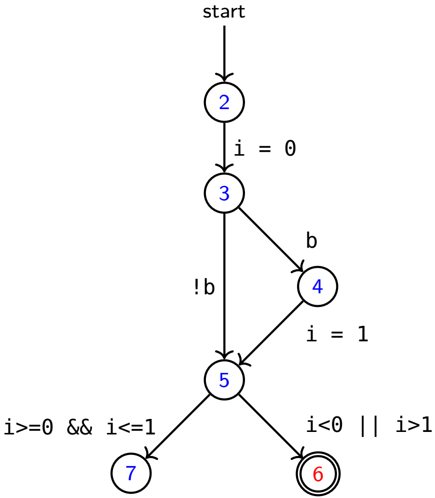
## Initiale Zustände $\sigma^I$

$\{\texttt{pc} \mapsto ②, \texttt{b} \mapsto b\}$ mit $b \in \mathbb{B}$

## Erreichbare Zustände $\sigma^R$

$\{\texttt{pc} \mapsto ②, \texttt{b} \mapsto \mathit{false}\}$
$\{\texttt{pc} \mapsto ③, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto ⑤, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$

$\{\texttt{pc} \mapsto ②, \texttt{b} \mapsto \mathit{true}\}$



# Beispiel: Programmverifikation

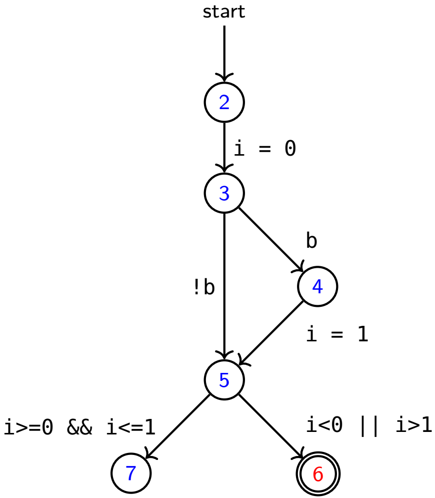
### Initiale Zustände $\sigma^I$

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto b\}$ mit $b \in \mathbb{B}$

### Erreichbare Zustände $\sigma^R$

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto \mathit{false}\}$
$\{\texttt{pc} \mapsto 3, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 5, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 7, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto \mathit{true}\}$



# Beispiel: Programmverifikation

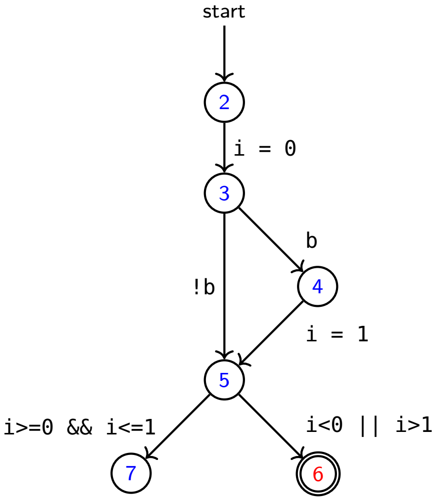
**Initiale Zustände $\sigma^I$**

$\{\mathtt{pc} \mapsto 2, \mathtt{b} \mapsto b\}$ mit $b \in \mathbb{B}$

**Erreichbare Zustände $\sigma^R$**

$\{\mathtt{pc} \mapsto 2, \mathtt{b} \mapsto \mathit{false}\}$
$\{\mathtt{pc} \mapsto 3, \mathtt{b} \mapsto \mathit{false}, \mathtt{i} \mapsto 0\}$
$\{\mathtt{pc} \mapsto 5, \mathtt{b} \mapsto \mathit{false}, \mathtt{i} \mapsto 0\}$
$\{\mathtt{pc} \mapsto 7, \mathtt{b} \mapsto \mathit{false}, \mathtt{i} \mapsto 0\}$

$\{\mathtt{pc} \mapsto 2, \mathtt{b} \mapsto \mathit{true}\}$
$\{\mathtt{pc} \mapsto 3, \mathtt{b} \mapsto \mathit{true}, \mathtt{i} \mapsto 0\}$



# Beispiel: Programmverifikation

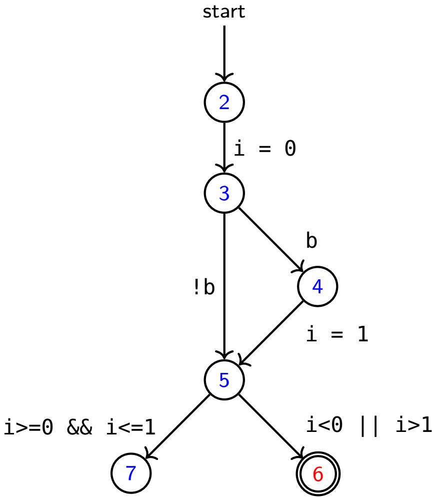
### Initiale Zustände $\sigma^I$

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto b\}$ mit $b \in \mathbb{B}$

### Erreichbare Zustände $\sigma^R$

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto \mathit{false}\}$
$\{\texttt{pc} \mapsto 3, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 5, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 7, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto \mathit{true}\}$
$\{\texttt{pc} \mapsto 3, \texttt{b} \mapsto \mathit{true}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 4, \texttt{b} \mapsto \mathit{true}, \texttt{i} \mapsto 0\}$



# Beispiel: Programmverifikation

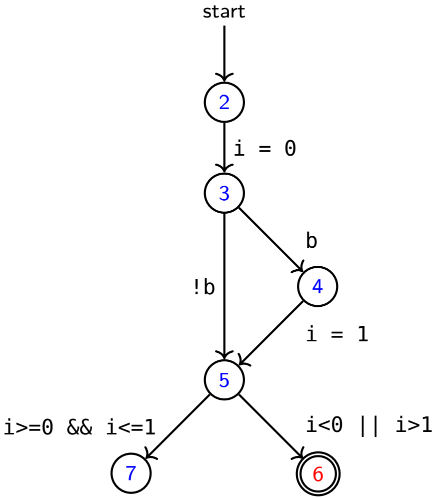
**Initiale Zustände $\sigma^I$**

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto b\}$ mit $b \in \mathbb{B}$

**Erreichbare Zustände $\sigma^R$**

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto \mathit{false}\}$
$\{\texttt{pc} \mapsto 3, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 5, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 7, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto \mathit{true}\}$
$\{\texttt{pc} \mapsto 3, \texttt{b} \mapsto \mathit{true}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 4, \texttt{b} \mapsto \mathit{true}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 5, \texttt{b} \mapsto \mathit{true}, \texttt{i} \mapsto 1\}$



# Beispiel: Programmverifikation

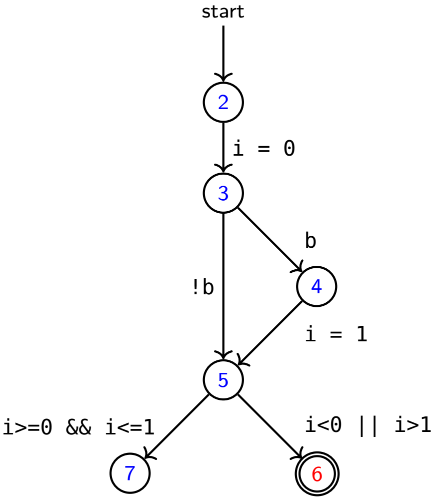
**Initiale Zustände $\sigma^I$**

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto b\}$ mit $b \in \mathbb{B}$

**Erreichbare Zustände $\sigma^R$**

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto \mathit{false}\}$
$\{\texttt{pc} \mapsto 3, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 5, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 7, \texttt{b} \mapsto \mathit{false}, \texttt{i} \mapsto 0\}$

$\{\texttt{pc} \mapsto 2, \texttt{b} \mapsto \mathit{true}\}$
$\{\texttt{pc} \mapsto 3, \texttt{b} \mapsto \mathit{true}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 4, \texttt{b} \mapsto \mathit{true}, \texttt{i} \mapsto 0\}$
$\{\texttt{pc} \mapsto 5, \texttt{b} \mapsto \mathit{true}, \texttt{i} \mapsto 1\}$
$\{\texttt{pc} \mapsto 7, \texttt{b} \mapsto \mathit{true}, \texttt{i} \mapsto 1\}$



# Beispiel: Programmverifikation

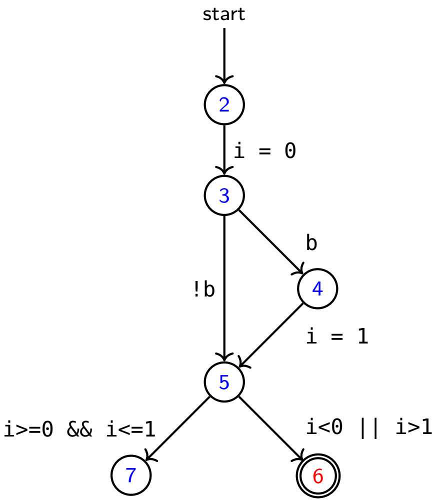
**Initiale Zustände** $\sigma^I$

$\{\mathtt{pc} \mapsto ②, \mathtt{b} \mapsto b\}$ mit $b \in \mathbb{B}$

**Erreichbare Zustände** $\sigma^R$

$\{\mathtt{pc} \mapsto ②, \mathtt{b} \mapsto \mathit{false}\}$
$\{\mathtt{pc} \mapsto ③, \mathtt{b} \mapsto \mathit{false}, \mathtt{i} \mapsto 0\}$
$\{\mathtt{pc} \mapsto ⑤, \mathtt{b} \mapsto \mathit{false}, \mathtt{i} \mapsto 0\}$
$\{\mathtt{pc} \mapsto ⑦, \mathtt{b} \mapsto \mathit{false}, \mathtt{i} \mapsto 0\}$

$\{\mathtt{pc} \mapsto ②, \mathtt{b} \mapsto \mathit{true}\}$
$\{\mathtt{pc} \mapsto ③, \mathtt{b} \mapsto \mathit{true}, \mathtt{i} \mapsto 0\}$
$\{\mathtt{pc} \mapsto ④, \mathtt{b} \mapsto \mathit{true}, \mathtt{i} \mapsto 0\}$
$\{\mathtt{pc} \mapsto ⑤, \mathtt{b} \mapsto \mathit{true}, \mathtt{i} \mapsto 1\}$
$\{\mathtt{pc} \mapsto ⑦, \mathtt{b} \mapsto \mathit{true}, \mathtt{i} \mapsto 1\}$




✓ Kein Zustand mit pc $\mapsto$ ⑥ erreichbar $\rightsquigarrow$ Programm ist korrekt

# Was man können und wissen sollte

- Wie repräsentiert man ein Programm als Kontrollflussautomaten, formal?
- Benennen der Bestandteile eines Transitionssystems $T = (\Sigma, \sigma^I, \rightarrow)$
- Was ist der Zusammenhang von Kontrollflussautomaten und den zugehörigen Transitionssystemen
- Welchen Nutzen hat die Menge der erreichbaren Zustände für die Verifikation?

Zum Nachdenken

- Muss man das zu einem Programm gehörige Transitionssystem vollständig berechnen, *bevor* man eine Erreichbarkeitsanalyse zur Verifikation durchführt?
- Welches Kriterium der Test-Abdeckung passt zur Berechnung von $\sigma^R$?

Erreichbarkeitsanalysen

# Explizite Erreichbarkeitsanalyse

# Heute

Bis jetzt

- ✓ Transitionssysteme als Semantik für Programme
- ✓ Verifikation anhand erreichbarer Zustände
- ✓ Analyse am Beispiel (Modelle, Programme)

# Heute

Bis jetzt

- ✓ Transitionssysteme als Semantik für Programme
- ✓ Verifikation anhand erreichbarer Zustände
- ✓ Analyse am Beispiel (Modelle, Programme)

Dieser Teil

- Begriffe, formale Definition von Erreichbarkeit
- Algorithmus: explizites Aufzählen von $\sigma^R$
- Formale Definition der Programmsemantik

# Transitionssysteme

Transitionssystem $T = (\Sigma, \sigma^I, \rightarrow)$ gegeben durch

- Zustandsmenge $\Sigma$, Startzustände $\sigma^I \subseteq \Sigma$
- Transitionsrelation $\rightarrow \subseteq \Sigma \times \Sigma$
  Schreibweise: $s \rightarrow s'$ oder $(s, s') \in \rightarrow$ für Zustände $s, s' \in \Sigma$

# Transitionssysteme

Transitionssystem $T = (\Sigma, \sigma^I, \rightarrow)$ gegeben durch

- Zustandsmenge $\Sigma$, Startzustände $\sigma^I \subseteq \Sigma$
- Transitionsrelation $\rightarrow \subseteq \Sigma \times \Sigma$
  Schreibweise: $s \rightarrow s'$ oder $(s, s') \in \rightarrow$ für Zustände $s, s' \in \Sigma$

*Nachfolger* von $s$: $post(s) := \{s' \mid s \rightarrow s'\}$
$T$ ist *nichtdeterministisch* falls $|post(s)| > 1$ für irgendeinen $s \in \Sigma$

# Transitionssysteme

Transitionssystem $T = (\Sigma, \sigma^I, \to)$ gegeben durch

- Zustandsmenge $\Sigma$, Startzustände $\sigma^I \subseteq \Sigma$
- Transitionsrelation $\to \subseteq \Sigma \times \Sigma$
  Schreibweise: $s \to s'$ oder $(s, s') \in \to$ für Zustände $s, s' \in \Sigma$

*Nachfolger* von $s$: $post(s) := \{s' \mid s \to s'\}$
$T$ ist *nichtdeterministisch* falls $|post(s)| > 1$ für irgendeinen $s \in \Sigma$

*Spur* durch $T$: endliche Folge von Zuständen $\overline{s} = \langle s_0, s_1, \ldots, s_n \rangle$, sodass

- $s_0 \in \sigma^I$
- $s_i \to s_{i+1}$ für alle $0 \leq i < n$

# Transitionssysteme

Transitionssystem $T = (\Sigma, \sigma^I, \to)$ gegeben durch

- Zustandsmenge $\Sigma$, Startzustände $\sigma^I \subseteq \Sigma$
- Transitionsrelation $\to \subseteq \Sigma \times \Sigma$
  Schreibweise: $s \to s'$ oder $(s, s') \in \to$ für Zustände $s, s' \in \Sigma$

*Nachfolger* von $s$: $post(s) := \{s' \mid s \to s'\}$
$T$ ist *nichtdeterministisch* falls $|post(s)| > 1$ für irgendeinen $s \in \Sigma$

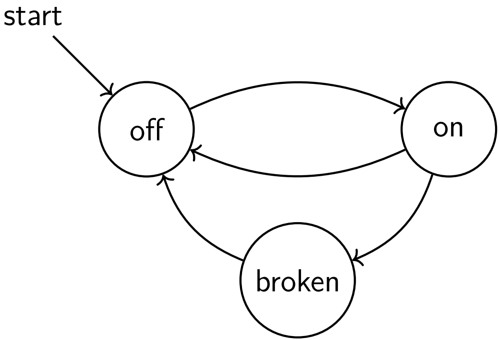
*Spur* durch $T$: endliche Folge von Zuständen $\overline{s} = \langle s_0, s_1, \dots, s_n \rangle$, sodass

- $s_0 \in \sigma^I$
- $s_i \to s_{i+1}$ für alle $0 \leq i < n$

*Erreichbare Zustände* $\sigma^R := \{s_n \mid \langle s_0, s_1, \dots, s_n \rangle$ Spur durch $T\}$

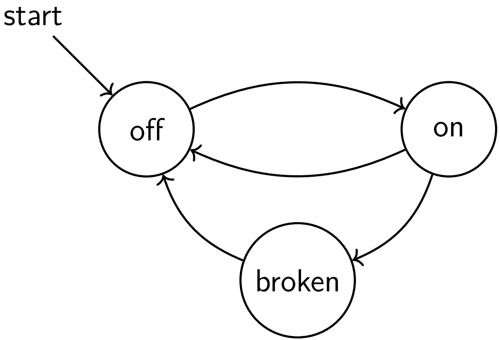
# Beispiel: Definitionen

Modell eines Projektors
(mit Defekten und Reparatur)




# Beispiel: Definitionen

Modell eines Projektors
(mit Defekten und Reparatur)




Nachfolger:

$$\begin{aligned} post(\text{off}) &= \{\text{on}\} \\ post(\text{on}) &= \{\text{off}, \text{broken}\} \\ post(\text{broken}) &= \{\text{off}\} \end{aligned}$$

Beispiele für Spuren:

$\langle \text{off} \rangle$

$\langle \text{off}, \text{on} \rangle$

$\langle \text{off}, \text{on}, \text{off} \rangle$

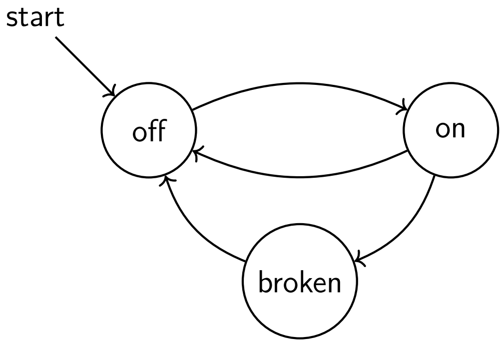
$\langle \text{off}, \text{on}, \text{broken}, \text{off}, \text{on} \rangle$

$\dots$

$\sigma^R = \Sigma$ (alle Zustände sind erreichbar)

# Beispiel: Definitionen

Modell eines Projektors
(mit Defekten und Reparatur)




Nichtdeterministische Auswahl der Nachfolger im Zustand “on”: Wir legen nicht fest *wann* bzw. *wodurch* eine Transition ausgeführt wird (✓ Abstraktion!)

Nachfolger:

$$post(\text{off}) = \{\text{on}\}$$
$$post(\text{on}) = \{\text{off}, \text{broken}\}$$
$$post(\text{broken}) = \{\text{off}\}$$

Beispiele für Spuren:

$\langle \text{off} \rangle$

$\langle \text{off}, \text{on} \rangle$

$\langle \text{off}, \text{on}, \text{off} \rangle$

$\langle \text{off}, \text{on}, \text{broken}, \text{off}, \text{on} \rangle$

$\ldots$

$\sigma^R = \Sigma$ (alle Zustände sind erreichbar)

# **Algorithmus**: Aufzählende Suche in Transitionssystemen

$\sigma^R := \varnothing$ 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**

# **Algorithmus**: Aufzählende Suche in Transitionssystemen

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

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

**while** $\tau \neq \emptyset$ **do**

  **choose** $s$ **with** $s \in \tau$

  $\tau := \tau \setminus \{s\}$

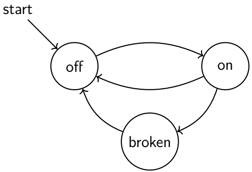
  **if** $s \notin \sigma^R$ **then**

    $\sigma^R := \sigma^R \cup \{s\}$

    $\tau := \tau \cup post(s)$

  **end if**

**end while**




1. $\tau = \{\text{off}\}$    $\sigma^R = \{\}$

# **Algorithmus**: Aufzählende Suche in Transitionssystemen

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

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

**while** $\tau \neq \varnothing$ **do**

  **choose** $s$ **with** $s \in \tau$

  $\tau := \tau \setminus \{s\}$

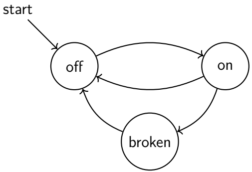
  **if** $s \notin \sigma^R$ **then**

    $\sigma^R := \sigma^R \cup \{s\}$

    $\tau := \tau \cup post(s)$

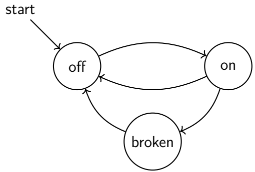
  **end if**

**end while**




1. $\tau = \{\text{off}\}$   $\sigma^R = \{\}$
2. $\tau = \{\text{on}\}$   $\sigma^R = \{\text{off}\}$

# **Algorithmus**: Aufzählende Suche in Transitionssystemen




$\sigma^R := \varnothing$ 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**

1. $\tau = \{\text{off}\}$  $\sigma^R = \{\}$
2. $\tau = \{\text{on}\}$  $\sigma^R = \{\text{off}\}$
3. $\tau = \{\text{broken}, \text{off}\}$  $\sigma^R = \{\text{off}, \text{on}\}$

# **Algorithmus**: Aufzählende Suche in Transitionssystemen

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

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

**while** $\tau \neq \varnothing$ **do**

  **choose** $s$ **with** $s \in \tau$

  $\tau := \tau \setminus \{s\}$

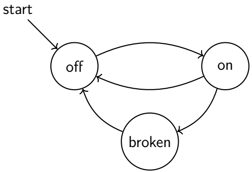
  **if** $s \notin \sigma^R$ **then**

    $\sigma^R := \sigma^R \cup \{s\}$

    $\tau := \tau \cup post(s)$

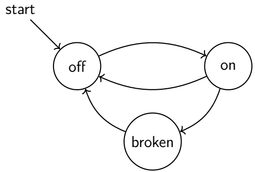
  **end if**

**end while**




1. $\tau = \{\text{off}\}$ $\quad\sigma^R = \{\}$
2. $\tau = \{\text{on}\}$ $\quad\sigma^R = \{\text{off}\}$
3. $\tau = \{\text{broken}, \text{off}\}$ $\quad\sigma^R = \{\text{off}, \text{on}\}$
4. $\tau = \{\text{broken}\}$ $\quad\sigma^R = \{\text{off}, \text{on}\}$

# **Algorithmus**: Aufzählende Suche in Transitionssystemen




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

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

**while** $\tau \neq \emptyset$ **do**

  **choose** $s$ **with** $s \in \tau$

  $\tau := \tau \setminus \{s\}$

  **if** $s \notin \sigma^R$ **then**

    $\sigma^R := \sigma^R \cup \{s\}$

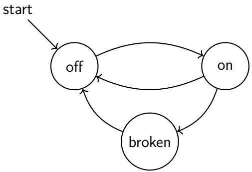
    $\tau := \tau \cup post(s)$

  **end if**

**end while**

1. $\tau = \{\text{off}\}$  $\sigma^R = \{\}$
2. $\tau = \{\text{on}\}$  $\sigma^R = \{\text{off}\}$
3. $\tau = \{\text{broken}, \text{off}\}$  $\sigma^R = \{\text{off}, \text{on}\}$
4. $\tau = \{\text{broken}\}$  $\sigma^R = \{\text{off}, \text{on}\}$
5. $\tau = \{\text{off}\}$  $\sigma^R = \{\text{off}, \text{on}, \text{broken}\}$

# **Algorithmus**: Aufzählende Suche in Transitionssystemen




$\sigma^R := \varnothing$ 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**

        $\sigma^R := \sigma^R \cup \{s\}$

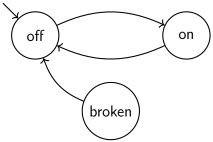
        $\tau := \tau \cup post(s)$

    **end if**

**end while**

1. $\tau = \{\text{off}\}$ $\quad\sigma^R = \{\}$
2. $\tau = \{\text{on}\}$ $\quad\sigma^R = \{\text{off}\}$
3. $\tau = \{\text{broken}, \text{off}\}$ $\quad\sigma^R = \{\text{off}, \text{on}\}$
4. $\tau = \{\text{broken}\}$ $\quad\sigma^R = \{\text{off}, \text{on}\}$
5. $\tau = \{\text{off}\}$ $\quad\sigma^R = \{\text{off}, \text{on}, \text{broken}\}$
6. $\tau = \{\}$ $\quad\sigma^R = \{\text{off}, \text{on}, \text{broken}\}$

# **Algorithmus**: Aufzählende Suche in Transitionssystemen




$\sigma^R := \varnothing$ 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**

    $\sigma^R := \sigma^R \cup \{s\}$

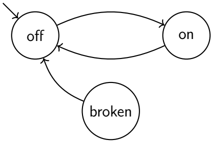
    $\tau := \tau \cup post(s)$

  **end if**

**end while**

1. $\tau = \{\text{off}\}$ $\sigma^R = \{\}$

# **Algorithmus**: Aufzählende Suche in Transitionssystemen




$\sigma^R := \varnothing$ 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**

    $\sigma^R := \sigma^R \cup \{s\}$

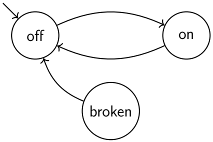
    $\tau := \tau \cup post(s)$

  **end if**

**end while**

1. $\tau = \{\text{off}\}$  $\sigma^R = \{\}$
2. $\tau = \{\text{on}\}$  $\sigma^R = \{\text{off}\}$

# **Algorithmus**: Aufzählende Suche in Transitionssystemen




$\sigma^R := \varnothing$ 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**

    $\sigma^R := \sigma^R \cup \{s\}$

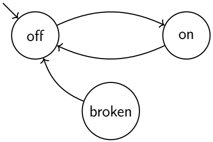
    $\tau := \tau \cup post(s)$

  **end if**

**end while**

1. $\tau = \{\text{off}\}$ $\quad\sigma^R = \{\}$
2. $\tau = \{\text{on}\}$ $\quad\sigma^R = \{\text{off}\}$
3. $\tau = \{\text{off}\}$ $\quad\sigma^R = \{\text{off}, \text{on}\}$

# **Algorithmus**: Aufzählende Suche in Transitionssystemen




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

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

**while** $\tau \neq \emptyset$ **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**

1. $\tau = \{\text{off}\}$ $\quad$ $\sigma^R = \{\}$
2. $\tau = \{\text{on}\}$ $\quad$ $\sigma^R = \{\text{off}\}$
3. $\tau = \{\text{off}\}$ $\quad$ $\sigma^R = \{\text{off}, \text{on}\}$
4. $\tau = \{\}$ $\quad$ $\sigma^R = \{\text{off}, \text{on}\}$

# Korrektheit

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

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

**while** $\tau \neq \emptyset$ **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**

- Es gilt immer: alle Zustände in $\tau$ und in $\sigma^R$ sind erreichbar
  - für $s \in \tau$ gibt es Spur $\langle s_0, \ldots, s \rangle$
  - analog: $s \in \sigma^R$

# Korrektheit

$\sigma^R := \varnothing$ 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**

- Es gilt immer: alle Zustände in $\tau$ und in $\sigma^R$ sind erreichbar
  - für $s \in \tau$ gibt es Spur $\langle s_0, \ldots, s \rangle$
  - analog: $s \in \sigma^R$
- Beweis der Korrektheit:
  - initial: $\langle s_0 \rangle$ mit $s_0 \in \sigma^I$
  - für $s' \in post(s)$ via Def $post$: $s \to s'$ und $\langle s_0, \ldots, s \rangle$ ist Spur $\Longrightarrow \langle s_0, \ldots, s, s' \rangle$ ist Spur □

# Korrektheit

$\sigma^R := \varnothing$ 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**

- Es gilt immer: alle Zustände in $\tau$ und in $\sigma^R$ sind erreichbar
  - für $s \in \tau$ gibt es Spur $\langle s_0, \ldots, s \rangle$
  - analog: $s \in \sigma^R$
- Beweis der Korrektheit:
  - initial: $\langle s_0 \rangle$ mit $s_0 \in \sigma^I$
  - für $s' \in post(s)$ via Def $post$: $s \to s'$ und $\langle s_0, \ldots, s \rangle$ ist Spur $\implies \langle s_0, \ldots, s, s' \rangle$ ist Spur $\square$
- kein falsches Ergebnis

# Laufzeit

$\sigma^R := \varnothing$ 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**

- In jedem Durchlauf wird ein Zustand aus $\tau$ herausgenommen
- Schleifeniterationen proportional dazu, wie viele Zustände insgesamt zu $\tau$ hinzugefügt werden

# Laufzeit

$\sigma^R := \varnothing$ 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**

- In jedem Durchlauf wird ein Zustand aus $\tau$ herausgenommen
- Schleifeniterationen proportional dazu, wie viele Zustände insgesamt zu $\tau$ hinzugefügt werden
  - initial: $|\sigma^I|$
  - pro Schleifendurchlauf: $|post(s)|$ für erreichbare $s$ einmal
- Laufzeit: proportional zur Anzahl der erreichbaren Transitionen
- Vollständigkeit: alle erreichbaren Zustände werden gefunden (wenn endlich erreichbar, ohne Beweis)

# Bemerkungen zur Implementierung

Welche Datenstruktur für $\tau$?

- Stack (LIFO) $\to$ DFS (depth-first search) Tiefensuche
  - lange Spuren zuerst verfolgen
  - ✓ rekursive Implementierung DFS gibt uns sofort den Fehlerpfad and
- Queue (FIFO) $\to$ BFS (breadth-first search) Breitensuche
  - ✓ findet Fehler in unendlichen großen Systemen
  - ✗ erhöhter Speicherplatzverbrauch
- Priority Queue: vielversprechende Spuren zuerst (Heuristiken zur Fehlersuche)

# Programmverifikation mit expliziter Erreichbarkeitsanalyse

Grundidee:

1. Übersetze Programm $P$ in Transitionssystem $T$
2. Berechne erreichbare Zustände $\sigma^R$
3. Teste Korrektheitseigenschaften $\varphi$ bezglüglich $\sigma^R$
   $\forall\ s \in \sigma^R.\ s \models \varphi \quad (s$ "erfüllt" $\varphi$, bzw. $\varphi$ "gilt in" $s)$

# Programmverifikation mit expliziter Erreichbarkeitsanalyse

Grundidee:

1. Übersetze Programm $P$ in Transitionssystem $T$
2. Berechne erreichbare Zustände $\sigma^R$
3. Teste Korrektheitseigenschaften $\varphi$ bezglüglich $\sigma^R$
   $\forall\ s \in \sigma^R.\ s \models \varphi \quad$ ($s$ "erfüllt" $\varphi$, bzw. $\varphi$ "gilt in" $s$)

Beispiele für Korrektheitseigenschaften

- $\varphi$: $\mathsf{pc} = \ell_{\mathsf{end}} \implies \mathsf{x} \geq 0$
- $\psi$: $\mathsf{pc} \neq \ell_{\mathsf{error}}$

# Programmverifikation mit expliziter Erreichbarkeitsanalyse

Grundidee:

1. Übersetze Programm $P$ in Transitionssystem $T$
2. Berechne erreichbare Zustände $\sigma^R$
3. Teste Korrektheitseigenschaften $\varphi$ bezglüglich $\sigma^R$
   $\forall\ s \in \sigma^R.\ s \models \varphi \quad$ ($s$ "erfüllt" $\varphi$, bzw. $\varphi$ "gilt in" $s$)

Beispiele für Korrektheitseigenschaften

- $\varphi$: $\mathsf{pc} = \ell_{\mathsf{end}} \Longrightarrow \mathsf{x} \geq 0$
- $\psi$: $\mathsf{pc} \neq \ell_{\mathsf{error}}$

Auswertung von $s \models \varphi$ durch Einzsetzen der Variablenwerte, z.B.:

- Für $s = \{\mathsf{pc} \mapsto \ell_{\mathsf{start}}, \mathsf{x} \mapsto -1\}$
  $\ell_{\mathsf{start}} = \ell_{\mathsf{end}} \Longrightarrow -1 \geq 0 \quad$ gdw. $\quad \texttt{false} \Longrightarrow -1 \geq 0$ ✓

# Übersetzungsschema Programme nach Transitionssysteme

Gegeben

- $P$ als Kontrollflussautomat $(L, \ell_0, G)$
- $\mathsf{x}_1 : t_1, \ldots, \mathsf{x}_n : t_n$ typisierte Variablen für $P$

# Übersetzungsschema Programme nach Transitionssysteme

Gegeben

- $P$ als Kontrollflussautomat $(L, \ell_0, G)$
- $\mathsf{x}_1 : t_1, \ldots, \mathsf{x}_n : t_n$ typisierte Variablen für $P$

Sei $[\![t]\!]$ der Wertebereich des Typs $t$

- $[\![\texttt{int}]\!] = \mathbb{Z}$ oder $[\![\texttt{int}]\!] = \{-\texttt{INT\_MIN} \cdots \texttt{INT\_MAX}\}$
- $[\![\texttt{boolean}]\!] = \mathbb{B}$ $[\![\texttt{String}]\!]$: alle Zeichenketten

Vergleiche: jqwick Generatoren

# Übersetzungsschema Programme nach Transitionssysteme

Gegeben

- $P$ als Kontrollflussautomat $(L, \ell_0, G)$
- $\mathsf{x}_1 : t_1, \dots, \mathsf{x}_n : t_n$ typisierte Variablen für $P$

Sei $[\![t]\!]$ der Wertebereich des Typs $t$

- $[\![\texttt{int}]\!] = \mathbb{Z}$ oder $[\![\texttt{int}]\!] = \{-\texttt{INT\_MIN} \cdots \texttt{INT\_MAX}\}$
- $[\![\texttt{boolean}]\!] = \mathbb{B}$ $\qquad$ $[\![\texttt{String}]\!]$: alle Zeichenketten

Vergleiche: jqwick Generatoren

Konstruiere $T = (\Sigma, \sigma^I, \rightarrow)$

- $\Sigma \;= \Big\{\{\mathsf{pc} \mapsto \ell, \mathsf{x}_1 \mapsto v_1, \dots, \mathsf{x}_n \mapsto v_n\}\Big\}$

# Übersetzungsschema Programme nach Transitionssysteme

Gegeben

- $P$ als Kontrollflussautomat $(L, \ell_0, G)$
- $\mathsf{x}_1 : t_1, \ldots, \mathsf{x}_n : t_n$ typisierte Variablen für $P$

Sei $[\![t]\!]$ der Wertebereich des Typs $t$

- $[\![\texttt{int}]\!] = \mathbb{Z}$ oder $[\![\texttt{int}]\!] = \{-\texttt{INT\_MIN} \cdots \texttt{INT\_MAX}\}$
- $[\![\texttt{boolean}]\!] = \mathbb{B}$ $\qquad$ $[\![\texttt{String}]\!]$: alle Zeichenketten

Vergleiche: jqwick Generatoren

Konstruiere $T = (\Sigma, \sigma^I, \rightarrow)$

- $\Sigma \; = \Big\{ \{\mathsf{pc} \mapsto \ell, \mathsf{x}_1 \mapsto v_1, \ldots, \mathsf{x}_n \mapsto v_n\} \mid \ell \in L, v_1 \in [\![t_1]\!], \ldots, v_n \in [\![t_n]\!] \Big\}$

# Übersetzungsschema Programme nach Transitionssysteme

Gegeben

- $P$ als Kontrollflussautomat $(L, \ell_0, G)$
- $\mathsf{x}_1 : t_1, \dots, \mathsf{x}_n : t_n$ typisierte Variablen für $P$

Sei $[\![t]\!]$ der Wertebereich des Typs $t$

- $[\![\texttt{int}]\!] = \mathbb{Z}$ oder $[\![\texttt{int}]\!] = \{-\texttt{INT\_MIN} \cdots \texttt{INT\_MAX}\}$
- $[\![\texttt{boolean}]\!] = \mathbb{B}$ $[\![\texttt{String}]\!]$: alle Zeichenketten

Vergleiche: jqwick Generatoren

Konstruiere $T = (\Sigma, \sigma^I, \to)$

- $\Sigma = \Big\{\{\mathsf{pc} \mapsto \ell, \mathsf{x}_1 \mapsto v_1, \dots, \mathsf{x}_n \mapsto v_n\} \mid \ell \in L, v_1 \in [\![t_1]\!], \dots, v_n \in [\![t_n]\!]\Big\}$
- $\sigma^I = \Big\{\{\mathsf{pc} \mapsto \ell_0, \mathsf{x}_1 \mapsto v_1, \dots, \mathsf{x}_n \mapsto v_n\} \mid v_i \text{ gültiger Initialwert von } \mathsf{x}_i\Big\}$

# Übersetzungsschema Programme nach Transitionssysteme

Gegeben

- $P$ als Kontrollflussautomat $(L, \ell_0, G)$
- $\mathsf{x}_1 : t_1, \ldots, \mathsf{x}_n : t_n$ typisierte Variablen für $P$

Konstruiere $T = (\Sigma, \sigma^I, \rightarrow)$

- $\Sigma \; = \Big\{ \{\mathsf{pc} \mapsto \ell, \vec{\mathsf{x}} \mapsto \vec{v}\} \mid \ell \in L, \vec{v} \in [\![\vec{t}]\!] \Big\}$

  (Abkürzende Schreibweise $\vec{a}$ für $a_1, \ldots, a_n$)

# Übersetzungsschema Programme nach Transitionssysteme

Gegeben

- $P$ als Kontrollflussautomat $(L, \ell_0, G)$
- $\mathsf{x}_1 : t_1, \ldots, \mathsf{x}_n : t_n$ typisierte Variablen für $P$

Konstruiere $T = (\Sigma, \sigma^I, \rightarrow)$

- $\Sigma \ = \Big\{\{\mathsf{pc} \mapsto \ell, \vec{\mathsf{x}} \mapsto \vec{v}\} \mid \ell \in L, \vec{v} \in [\![\vec{t}]\!]\Big\}$
  (Abkürzende Schreibweise $\vec{a}$ für $a_1, \ldots, a_n$)
- 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 $\ell \xrightarrow{op} \ell' \in G$

# Übersetzungsschema Programme nach Transitionssysteme

Gegeben

- $P$ als Kontrollflussautomat $(L, \ell_0, G)$
- $\mathsf{x}_1 : t_1, \ldots, \mathsf{x}_n : t_n$ typisierte Variablen für $P$

Konstruiere $T = (\Sigma, \sigma^I, \rightarrow)$

- $\Sigma \ = \Big\{ \{\mathsf{pc} \mapsto \ell, \vec{\mathsf{x}} \mapsto \vec{v}\} \mid \ell \in L, \vec{v} \in [\![\vec{t}]\!] \Big\}$
  (Abkürzende Schreibweise $\vec{a}$ für $a_1, \ldots, a_n$)
- 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 $\ell \xrightarrow{op} \ell' \in G$, solange 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 $\vec{v}' = \vec{v}$

# Implementierung: Erreichbarkeitsanalyse von Programmen

Mit dem Übersetzungsschema von $P$ nach $T$ können wir uns *mathematisch* überzeugen, wie Programme analysiert werden können.
Für eine *praktische* Umsetzung können wir uns direkt auf $post(s)$ stützen, z.B.:

```
class State          { int pc; Map<String,Integer> v; ...  }
interface Transition { Set<State> post(State s);           }
```

# Implementierung: Erreichbarkeitsanalyse von Programmen

Mit dem Übersetzungsschema von $P$ nach $T$ können wir uns *mathematisch* überzeugen, wie Programme analysiert werden können.
Für eine *praktische* Umsetzung können wir uns direkt auf $post(s)$ stützen, z.B.:

```
class State          { int pc; Map<String,Integer> v; ...  }
interface Transition { Set<State> post(State s);           }
```

```
class Assignment
  extends Transition {
    String x;
    Expression e;

    Set<State> post(State s) {
      State t = s.clone();
      t.v.put(x, e.eval(s));
      return Set.of(t);
    }
}
```

```
class Condition
  extends Transition {
    Formula phi;

    Set<State> post(State s) {
      if(phi.eval(s))
        return Set.of(s);
      else
        return Set.of();
    }
}
```

# Was man wissen und können sollte

- Begriffe: Nachfolgerzustände $post$, Spuren, Definition von $\sigma^R$
- Übersetzung Programme $\to$ Transitionssysteme:
  - Wie werden Programmzustände im Transitionssystem repräsentiert?
  - Wie werden Ausdrücke und Bedingungen über Zuständen ausgewertet?
  - Wie ergeben sich die Transitionen aus den Kanten im Kontrollflussautomaten als Ausführung der Zuweisungen bzw Überprüfung der Bedingungen?
- Grundidee hinter Korrektheit, Laufzeit, Vollständigkeit des Algorithmus

- Explizite Erreichbarkeitsanalyse auf Modellen und <u>Programmen</u> ausführen

- Zum Nachdenken: Welche Gemeinsamkeiten und Unterschiede finden sich zwischen der expliziten Erreichbarkeitsanalyse und den kennengelernten Ansätzen zum Testen?

# License