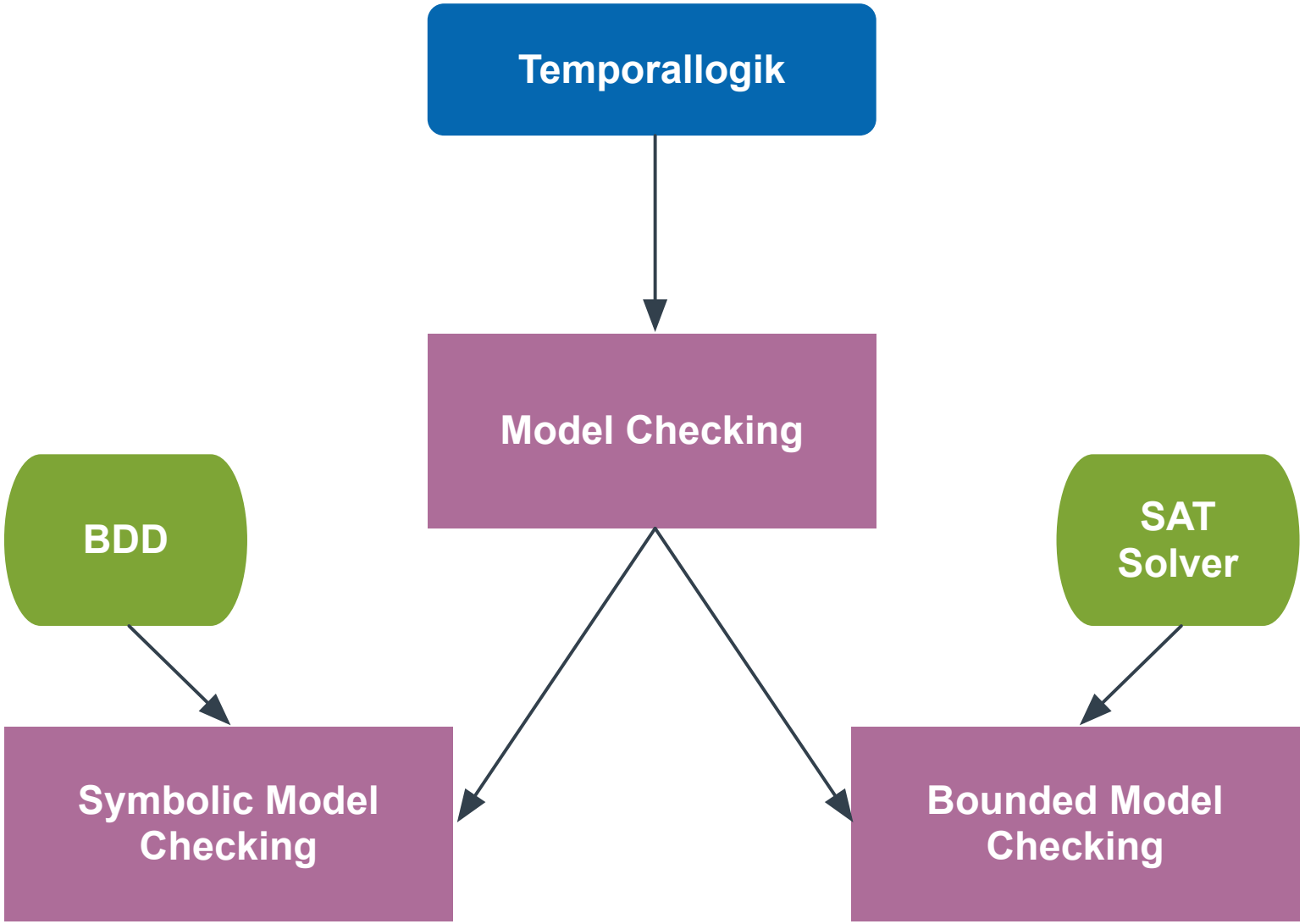
Automatisches Beweisen—Vertiefung
CTL Model Checking & Symbolic Model Checking

Christoph Zengler

Arbeitsbereich Symbolisches Rechnen
Prof. Dr. Wolfgang Küchlin
Universität Tübingen

15. Januar 2013

Der Plan für die nächsten drei Wochen



Motivation

Model Checking

CTL Model Checking

Symbolic Model Checking

BDD

Der Algorithmus

Literatur

Ein Beispiel für Model Checking

Gegenseitiger Ausschluss von Prozessen

- Gleichzeitiger Zugriff auf Ressourcen muss gekapselt werden
- Prozess darf nur in seinem kritischen Bereich (KB) darauf zugreifen

Eigenschaften

1. Sicherheit/Safety: Nur ein Prozess kann pro Zeitschritt in seinem kritischen Bereich sein.
2. Lebendigkeit/Liveness: Wann immer ein Prozess beantragt, in seinen KB zu kommen, wird ihm dies irgendwann erlaubt.
3. Nicht-Blockierend/non-blocking: Ein Prozess kann jederzeit beantragen in seinen KB zu kommen.

Motivation

Model Checking

CTL Model Checking

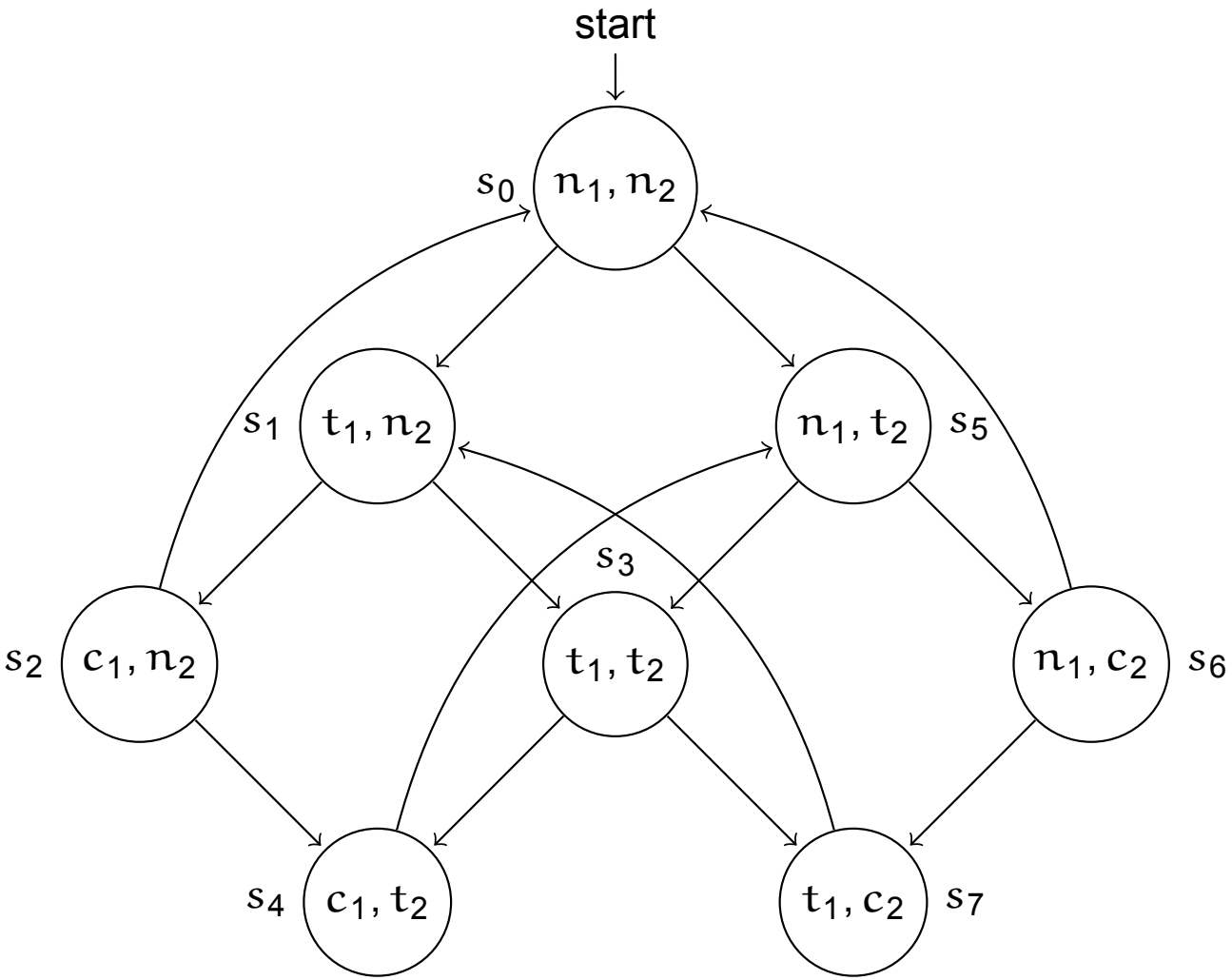
Symbolic Model Checking

BDD

Der Algorithmus

Literatur

1. Modellierungsversuch



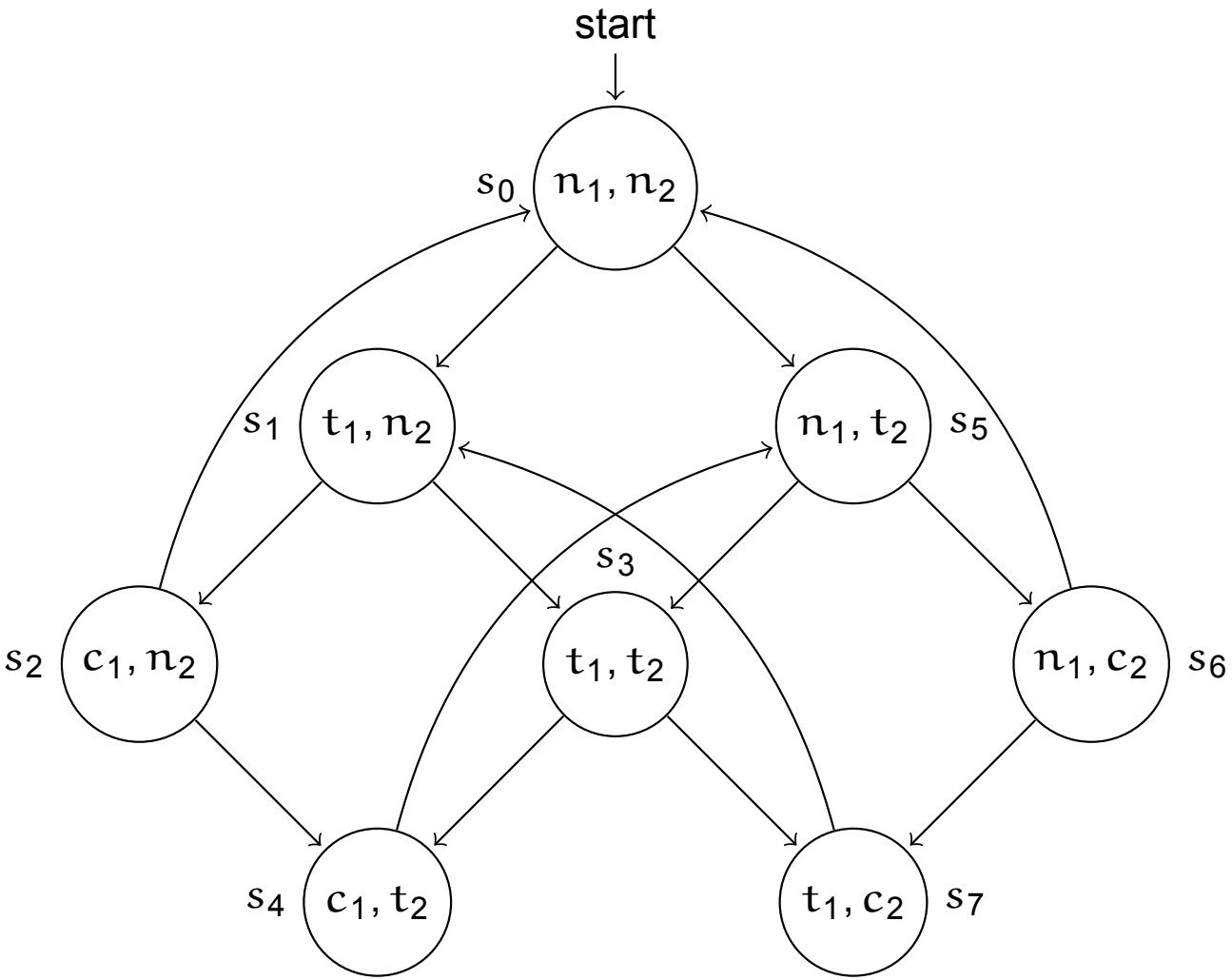
Zwei Prozesse, die entweder

(n) in ihrem nichtkritischen Bereich sind,

(t) beantragen in ihren KB zu kommen, oder

(c) in ihrem KB sind

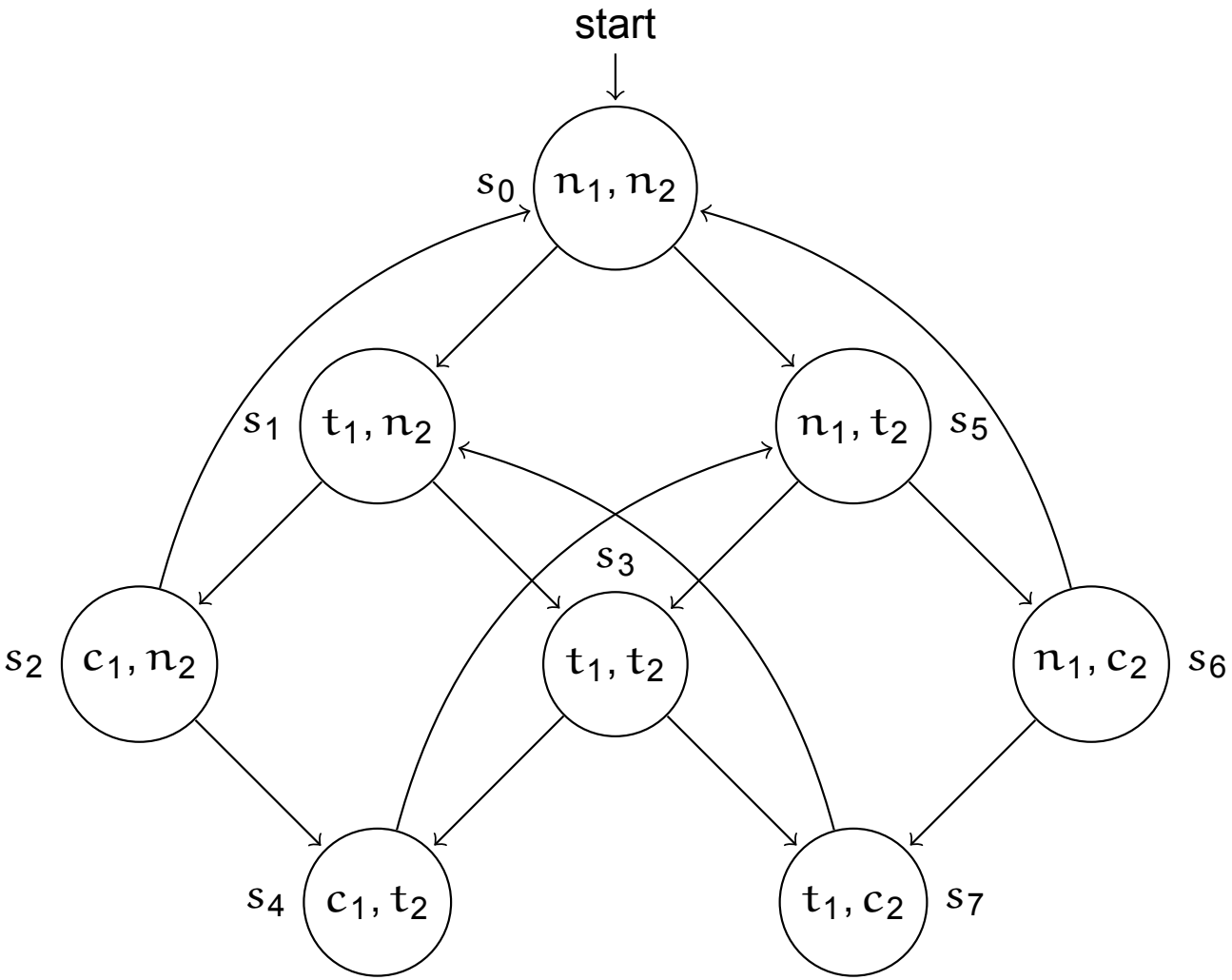
1. Modellierungsversuch



Safety in CTL

$\mathbf{AG}\,\neg(c_1 \wedge c_2)$ ✓

1. Modellierungsversuch

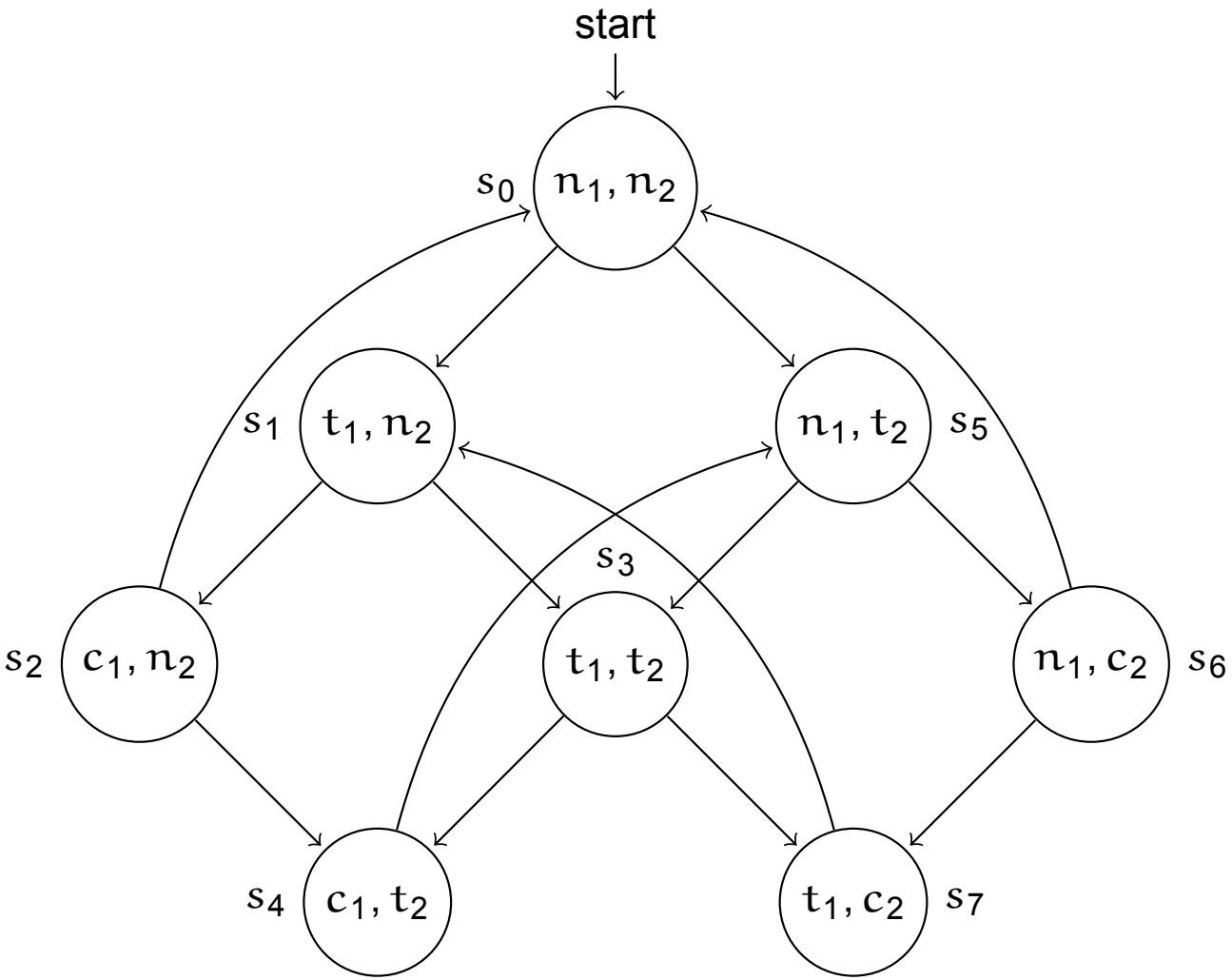


Liveness in CTL

$$\mathbf{AG}(t_1 \rightarrow \mathbf{AF}\, c_1) \wedge \mathbf{AG}(t_2 \rightarrow \mathbf{AF}\, c_2)$$ ↯

Problem: Nicht-Determinismus in s_3

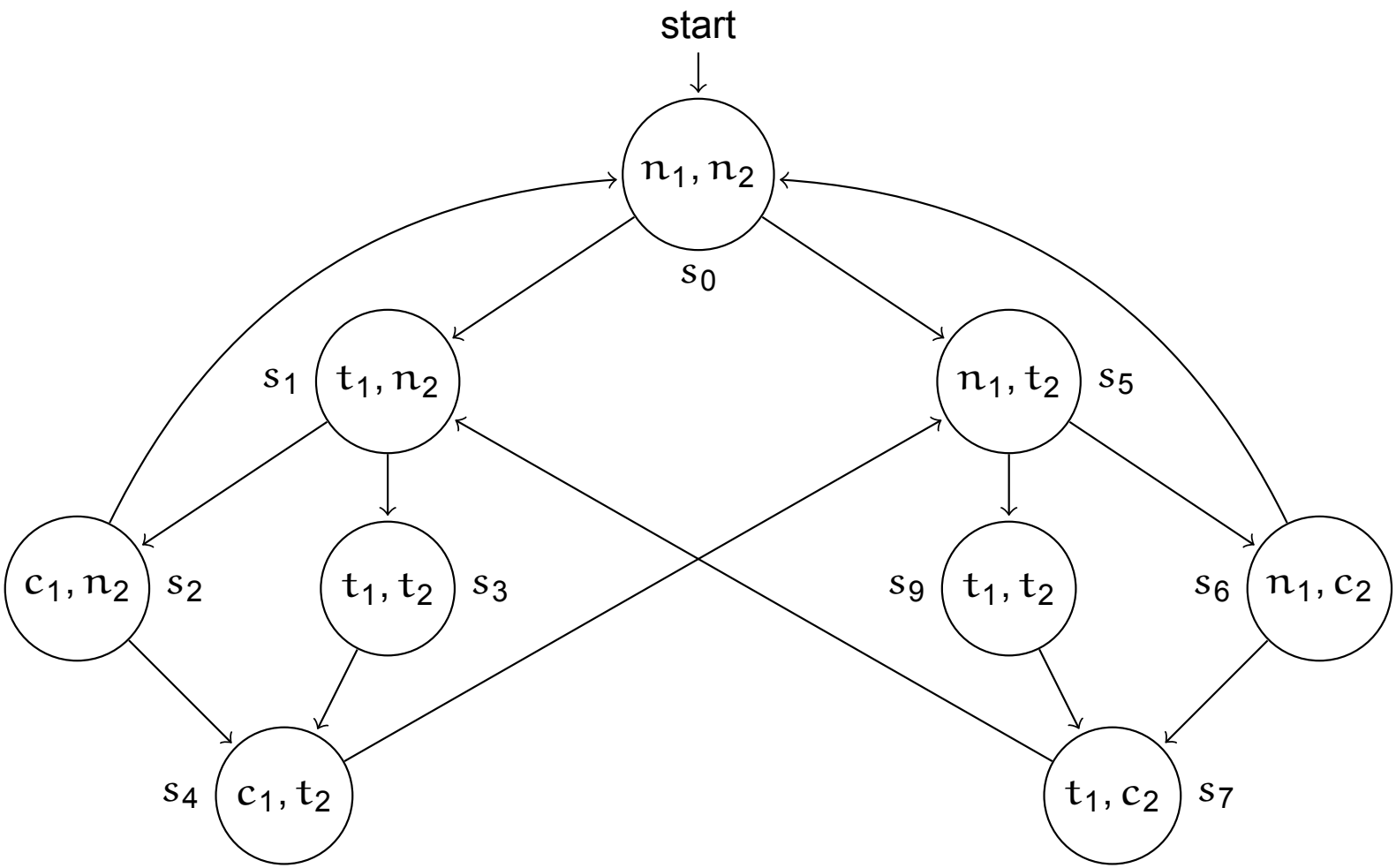
1. Modellierungsversuch



Non-blocking in CTL

$$\mathbf{AG}(n_1 \rightarrow \mathbf{EF}\, t_1) \wedge \mathbf{AG}(n_2 \rightarrow \mathbf{EF}\, t_2) \qquad \checkmark$$

2. Modellierungsversuch



Motivation

Model Checking

CTL Model Checking

Symbolic Model Checking

BDD

Der Algorithmus

Literatur

Model Checking

Definition (Model Checking)

Gegeben sei eine Kripke-Struktur $\mathcal{M}$, ein Zustand $s \in S$ und eine temporallogische Formel φ. Die Frage, ob $\texttt{eval}(\mathcal{M}, s, \varphi)$ wahr ist, wird als Model Checking bezeichnet.

- Wir können keine unendlichen Bäume aus der Kripke Struktur abwickeln, d.h. Check muss auf der Struktur selber stattfinden
- Im Falle, dass φ nicht gilt, kann Model Checking einen Pfad in $\mathcal{M}$ angeben, der φ verletzt (Gegenbeispiel)

Alternative Definition (Model Checking)

Gegeben sei eine Kripke Struktur $\mathcal{M}$ und eine temporallogische Formel φ. Model Checking beantwortet die Frage, in welchen Zuständen $s \in S$ die Formel φ gilt.

- Die zweite Fragestellung schließt die erste ein
- Die alternative Definition ist algorithmisch leichter lösbar

Motivation

Model Checking

CTL Model Checking

Symbolic Model Checking

BDD

Der Algorithmus

Literatur

Der Labelling Algorithmus für CTL

Idee!

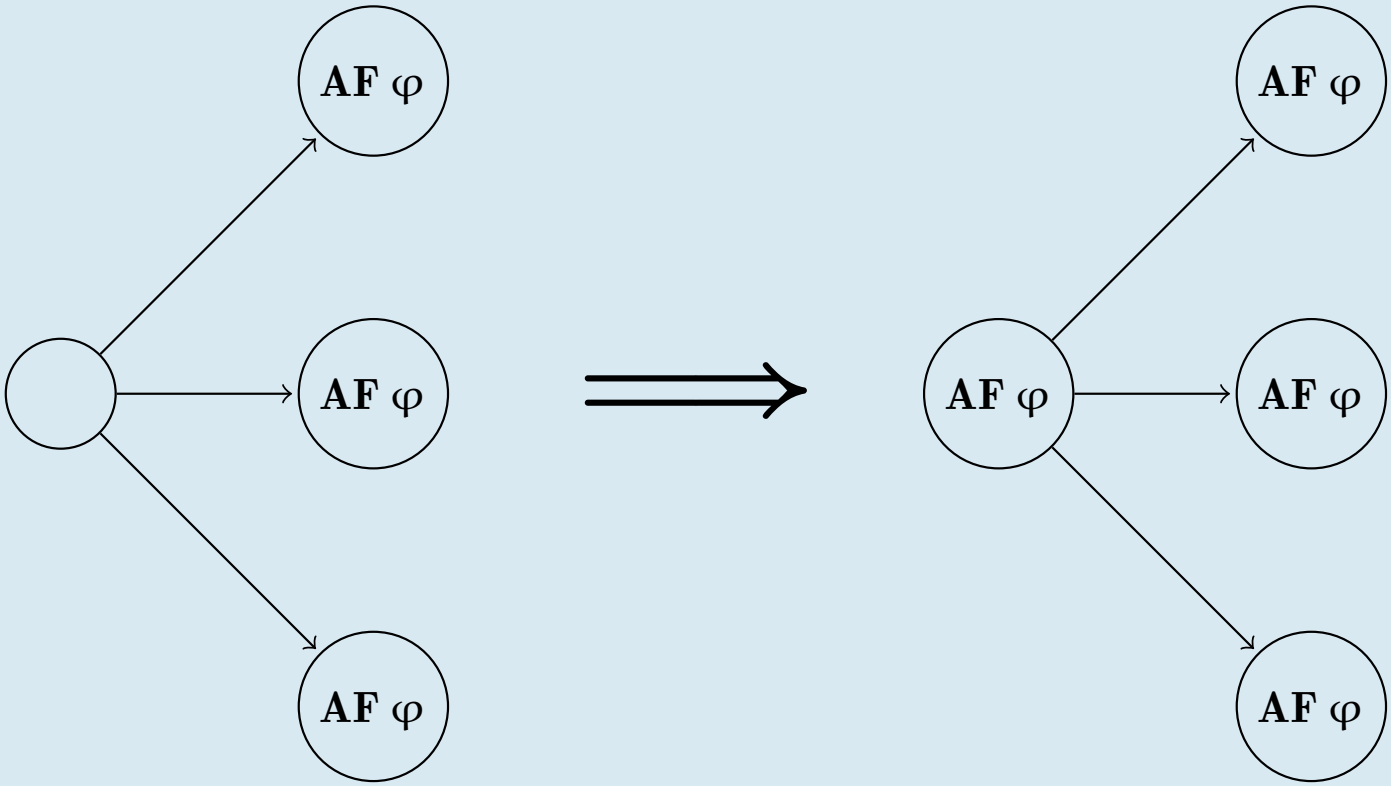
Beschrifte alle Zustände mit Teilformeln der relevanten Formeln (von den kleinsten Formeln nach außen), die an ihnen gelten. Schaue am Ende, in welchen Zuständen die relevante Formel gilt.

Wir brauchen nicht alle Operatoren zu betrachten

- Bei den aussagenlogischen Operatoren reichen $\bot$, $\neg$ und $\wedge$
- Bei den temporallogischen Operatoren reichen $\mathbf{AF}$, $\mathbf{EU}$ und $\mathbf{EX}$
  - $\mathbf{EF}\,\varphi \equiv \mathbf{E}[\top \mathbf{U} \varphi]$
  - $\mathbf{AG}\,\varphi \equiv \neg\,\mathbf{EF}\,\neg\varphi$
  - $\mathbf{EG}\,\varphi \equiv \neg\mathbf{AF}\neg\varphi$
  - $\mathbf{AX}\,\varphi \equiv \neg\mathbf{EX}\neg\varphi$
  - $\mathbf{A}[\varphi\,\mathbf{U}\,\psi] \equiv \neg(\mathbf{E}[\neg\psi\mathbf{U}(\neg\varphi \wedge \neg\psi)] \vee \mathbf{EG}\,\neg\psi)$

Grundidee des Labelling Algorithmus

Beispiel (Beschriftung von Zuständen mit $\mathbf{AF}$)



Implementierung

- keine explizite Annotation der Zustände

Der Algorithmus

Hauptalgorithmus

Algorithmus: $\texttt{modelCheckCTL}(\mathcal{M}, \psi)$

Eingabe: Kripke-Struktur $\mathcal{M}$ und CTL Formel ψ
Ausgabe: Menge an Zuständen $\subseteq S$, an denen ψ gilt

1. Reduziere ψ auf die Operatoren $\bot$, $\neg$, $\wedge$ $\mathbf{AF}$, $\mathbf{EU}$ und $\mathbf{EX}$
2. $\texttt{modelCheckCTL}(\mathcal{M}, \psi) = \psi$ match

$$
\begin{aligned}
&\bot && \rightsquigarrow \varnothing \\
\mid\ &v \in \mathcal{V} && \rightsquigarrow \{s \in S \mid v \in L(s)\} \\
\mid\ &\neg\varphi && \rightsquigarrow S - \texttt{modelCheckCTL}(\mathcal{M}, \varphi) \\
\mid\ &\varphi_1 \wedge \varphi_2 && \rightsquigarrow \texttt{modelCheckCTL}(\mathcal{M}, \varphi_1) \cap \texttt{modelCheckCTL}(\mathcal{M}, \varphi_2) \\
\mid\ &\mathbf{EX}\,\varphi && \rightsquigarrow \texttt{mc}_{\mathbf{EX}}(\mathcal{M}, \varphi) \\
\mid\ &\mathbf{AF}\,\varphi && \rightsquigarrow \texttt{mc}_{\mathbf{AF}}(\mathcal{M}, \varphi) \\
\mid\ &\mathbf{E}[\varphi_1\,\mathbf{U}\,\varphi_2] && \rightsquigarrow \texttt{mc}_{\mathbf{EU}}(\mathcal{M}, \varphi_1, \varphi_2)
\end{aligned}
$$

Existentielle und Universelle Urbilder

Zwei Funktionen zum Berechnen von Urbildern:

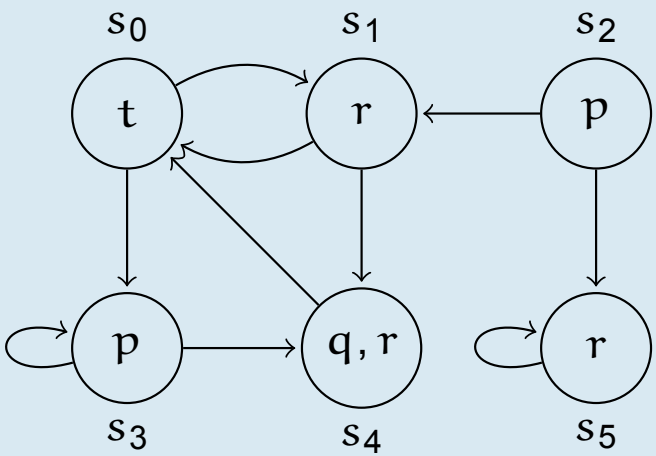
- $\mathtt{pre}_\exists(Y)$: Berechnet für eine Menge von Zuständen Y alle Zustände, die einen Übergang in Y machen *können*

$$\mathtt{pre}_\exists(Y) = \{s \in S \mid \text{exists } s' \text{ with } s \longrightarrow s' \text{ and } s' \in Y\}$$

- $\mathtt{pre}_\forall(Y)$: Berechnet für eine Menge von Zuständen Y alle Zustände, die *nur* Übergange in Y machen

$$\mathtt{pre}_\forall(Y) = \{s \in S \mid \text{for all } s' : s \longrightarrow s' \text{ implies } s' \in Y\}$$

Beispiel ($\mathtt{pre}_\exists$ und $\mathtt{pre}_\forall$)



- $\mathtt{pre}_\exists(\{s_3, s_4\}) = \{s_0, s_1, s_3\}$
- $\mathtt{pre}_\forall(\{s_3, s_4\}) = \{s_3\}$
- $\mathtt{pre}_\exists(\{s_5\}) = \{s_2, s_5\}$
- $\mathtt{pre}_\forall(\{s_5\}) = \{s_5\}$

Der Unteralgorithmus $\mathtt{mc}_{\mathbf{EX}}$

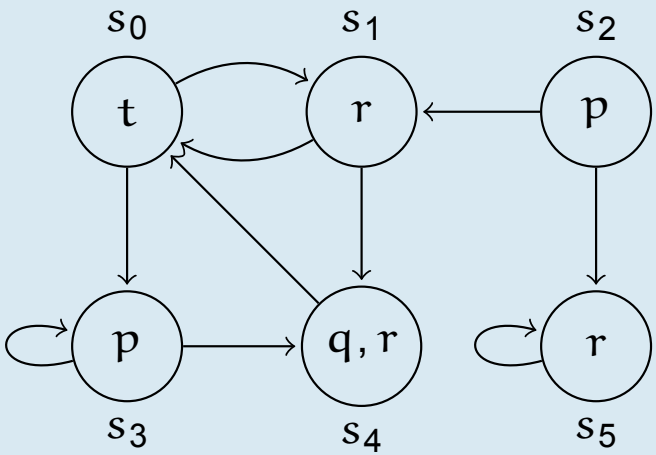
Algorithmus: $\mathtt{mc}_{\mathbf{EX}}(\mathcal{M}, \varphi)$

Eingabe: Kripke-Struktur $\mathcal{M}$ und CTL Formel φ
Ausgabe: Menge an Zuständen $\subseteq S$, an denen $\mathbf{EX}\,\varphi$ gilt

---

$X = \mathtt{modelCheckCTL}(\mathcal{M}, \varphi)$
$Y = \mathtt{pre}_{\exists}(X)$
return Y

Beispiel ($\mathtt{mc}_{\mathbf{EX}}$)



$\mathtt{mc}_{\mathbf{EX}}(\mathcal{M}, r)$:

1. $X = \mathtt{modelCheckCTL}(\mathcal{M}, r) = \{s_1, s_4, s_5\}$
2. $Y = \mathtt{pre}_{\exists}(X) = \{s_0, s_1, s_2, s_3, s_5\}$
3. **return** Y

Der Unteralgorithmus $\mathtt{mc}_{\mathbf{AF}}$

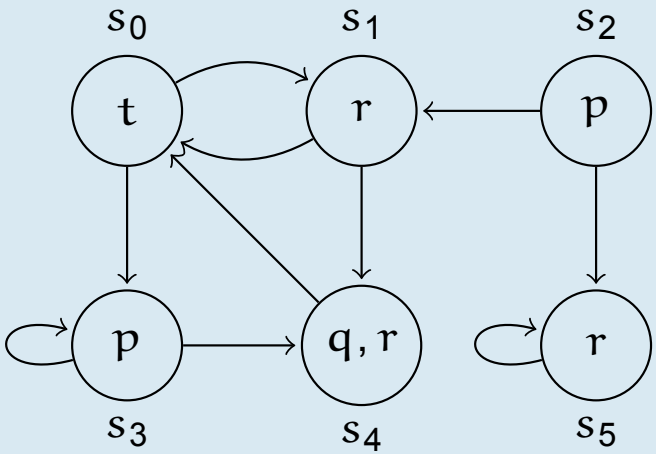
Algorithmus: $\mathtt{mc}_{\mathbf{AF}}(\mathcal{M}, \varphi)$

Eingabe: Kripke-Struktur $\mathcal{M}$ und CTL Formel φ
Ausgabe: Menge an Zuständen $\subseteq S$, an denen $\mathbf{AF}\,\varphi$ gilt

---

$X = S$
$Y = \mathtt{modelCheckCTL}(\mathcal{M}, \varphi)$
while $X \neq Y$ **do**
  $X = Y$
  $Y = Y \cup \mathtt{pre}_{\forall}(Y)$
return Y

Beispiel ($\mathtt{mc}_{\mathbf{AF}}$)



$\mathtt{mc}_{\mathbf{AF}}(\mathcal{M}, p)$:

1. $X = S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$
2. $Y = \{s_2, s_3\}$
3. $X = \{s_2, s_3\}$, $Y = \{s_2, s_3\}$
4. **return** Y

Der Unteralgorithmus $\mathtt{mc}_{\mathbf{EU}}$

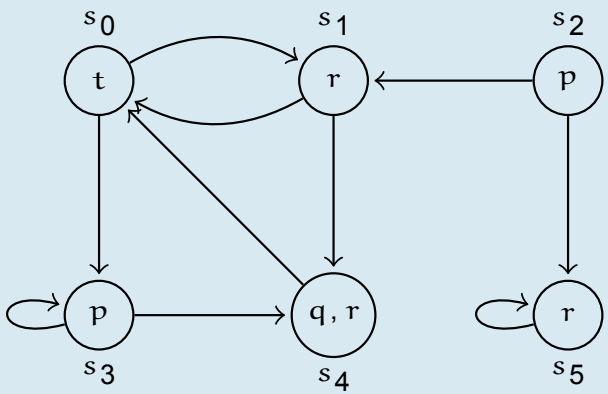
Algorithmus: $\mathtt{mc}_{\mathbf{EU}}(\mathcal{M}, \varphi_1, \varphi_2)$

Eingabe: Kripke-Struktur $\mathcal{M}$ und CTL Formeln φ_1 und φ_2
Ausgabe: Menge an Zuständen $\subseteq S$, an denen $\mathbf{E}[\varphi_1 \,\mathbf{U}\, \varphi_2]$ gilt

---

$W = \mathtt{modelCheckCTL}(\mathcal{M}, \varphi_1)$
$X = S$
$Y = \mathtt{modelCheckCTL}(\mathcal{M}, \varphi_2)$
while $X \neq Y$ **do**
  $X = Y$
  $Y = Y \cup (W \cap \mathtt{pre}_{\exists}(Y))$
return Y

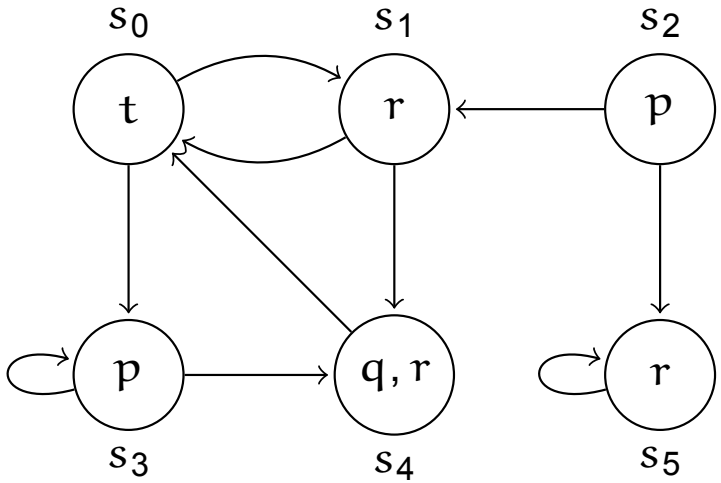
Beispiel ($\mathtt{mc}_{\mathbf{EU}}$)



$\mathtt{mc}_{\mathbf{EU}}(\mathcal{M}, t, r)$:

1. $W = \{s_0\}, X = S, Y = \{s_1, s_4, s_5\}$
2. $X = \{s_1, s_4, s_5\}, Y = \{s_0, s_1, s_4, s_5\}$
3. $X = \{s_0, s_1, s_4, s_5\}, Y = \{s_0, s_1, s_4, s_5\}$
4. **return** Y

Beispiel



Beispiel ($\mathbf{AG}(\mathbf{AF}(p \vee t))$)

1. Reduktion: $\varphi = \neg\,\mathbf{E}[\neg\bot\,\mathbf{U}\,\neg(\mathbf{AF}(p \vee t))]$
2. $\mathtt{modelCheckCTL}(\mathcal{M}, \varphi)$

   $= S - \mathtt{modelCheckCTL}(\mathcal{M}, \mathbf{E}[\neg\bot\,\mathbf{U}\,\neg(\mathbf{AF}(p \vee t))])$
   $= S - \mathtt{mc}_{\mathbf{EU}}(\mathcal{M}, \neg\bot, \neg(\mathbf{AF}(p \vee t)))$
3. $\mathtt{mc}_{\mathbf{EU}}(\mathcal{M}, \neg\bot, \neg(\mathbf{AF}(p \vee t)))$
   - $W = \mathtt{modelCheckCTL}(\mathcal{M}, \neg\bot) = \{s_0, s_1, s_2, s_3, s_4, s_5\}$
   - $X = \{s_0, s_1, s_2, s_3, s_4, s_5\}$
   - $Y = \mathtt{modelCheckCTL}(\mathcal{M}, \neg(\mathbf{AF}(p \vee t))) = S - \mathtt{mc}_{\mathbf{AF}}(\mathcal{M}, p \vee t)$
   - *... (an der Tafel)*

Komplexität

Komplexität

$$O(f \cdot V \cdot (V + E))$$

mit

- f Anzahl der Konnektive in der Formel
- V Anzahl der Zustände
- E Anzahl der Zustandsübergänge

d.h. linear in der Formel, quadratisch in der Kripke-Struktur

Verbesserung

Durch spezielle Behandlung von $\mathbf{EG}$ kann der Algorithmus auch linear in der Struktur sein, d.h. $O(f \cdot (V + E))$

Zustandsexplosion

- Modelle sind oft exponentiell in der Anzahl der Variablen (d.h. eine neue Variable verdoppelt das Modell)
- Umgehen durch z.B. Symbolic Model Checking (nächste VL)

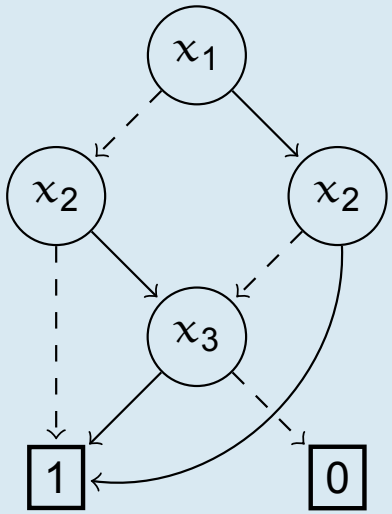
Motivation
Model Checking
CTL Model Checking
Symbolic Model Checking
BDD
Der Algorithmus
Literatur

Kurze Wiederholung: BDD

- Knowledge Compilation Format (Arbeit steckt im Aufbauen der Datenstruktur)
- Datenstruktur DAG
  - Knoten sind Variablen
  - Blätter sind die Terminale $\boxed{0}$ (`false`) und $\boxed{1}$ (`true`)
  - Kanten sind Belegung der Variable mit `true` oder `false`

Beispiel (BDD)

Formel: $(x_1 \leftrightarrow x_2) \vee x_3$, Variablenordnung $x_1 < x_2 < x_3$



ROBDD

- Normalerweise betrachten wir immer ROBDD:
  - reduziert: keine gleichen Teilbäume im BDD, bei keinem Knoten ist der rechte gleich dem linken Teilbaum
  - geordnet: auf allen Pfaden gilt die gleiche Variablenreihenfolge
- Kanonische Normalform für aussagenlogische Formeln
  - $\Rightarrow$ Tautologie entspricht BDD $\boxed{1}$
  - $\Rightarrow$ Kontradiktion entspricht BDD $\boxed{0}$
  - $\Rightarrow$ Erfüllbare Formel entspricht BDD ungleich $\boxed{0}$
- Graphalgorithmen können benutzt werden
  - Finden einer erfüllenden Belegung $\Rightarrow$ ein Pfad zu $\boxed{1}$
  - Auflisten aller erfüllenden Belegungen $\Rightarrow$ alle Pfade zu $\boxed{1}$
  - ...

Ab jetzt: BDD = ROBDD

Algorithmen auf ROBDD

Werden hier nur überblicksweise behandelt, Details siehe in ABG.

- $\mathcal{B}(\varphi)$: ROBDD, das die aussagenlogische Formel φ repräsentiert
- Jeder Knoten hat eine positive (high) und eine negative (low) Kante

Drei Algorithmen sind von Interesse

1. `apply`$(\mathcal{B}_1, \mathcal{B}_2, \circ)$: Kombiniert die beiden ROBDD $\mathcal{B}_1$ und $\mathcal{B}_2$ mit dem Booleschen Operator $\circ$, d.h

$$\mathcal{B}(\varphi_1 \circ \varphi_2) = \texttt{apply}(\mathcal{B}(\varphi_1), \mathcal{B}(\varphi_2), \circ)$$

2. `restrict`$(\mathcal{B}, v, b)$: Berechnet das ROBDD, bei dem die Variable v mit $b \in \{\bot, \top\}$ belegt ist (Restriktion), d.h.

$$\mathcal{B}(\varphi[b/v]) = \texttt{restrict}(\mathcal{B}(\varphi), v, b)$$

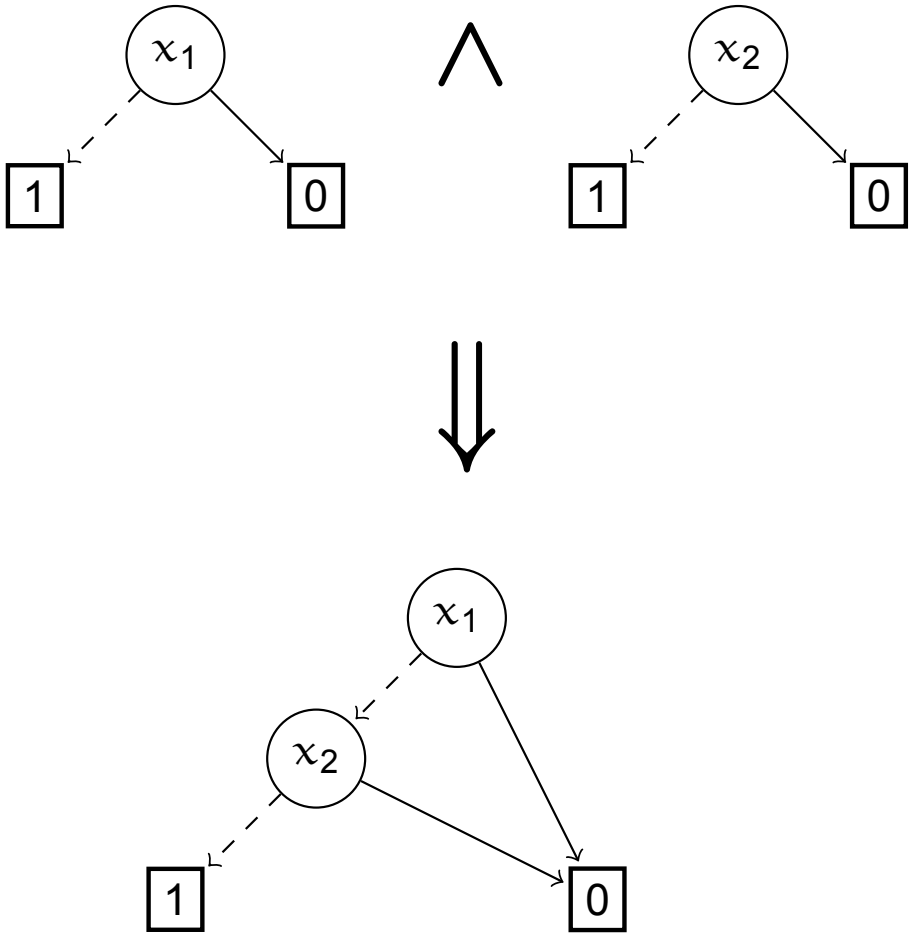
3. `exists`$(\mathcal{B}, v)$: Berechnet eine Projektion des ROBDD auf alle Variablen außer v, also eine existenzielle QE von v, d.h.

$$\mathcal{B}(\varphi[\top/v] \vee \varphi[\bot/v]) = \texttt{exists}(\mathcal{B}(\varphi), v)$$

Algorithmen auf ROBDD

Der `apply` Algorithmus

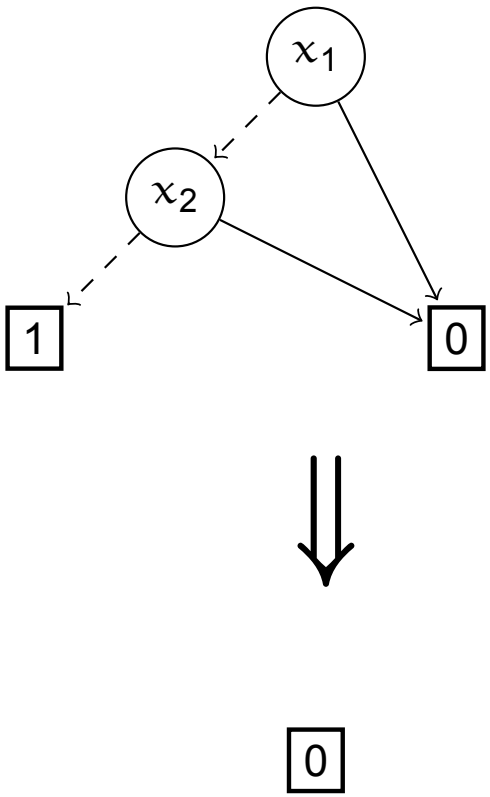
$$\mathcal{B}(\neg x_1 \wedge \neg x_2) = \texttt{apply}(\mathcal{B}(\neg x_1), \mathcal{B}(\neg x_2), \wedge)$$



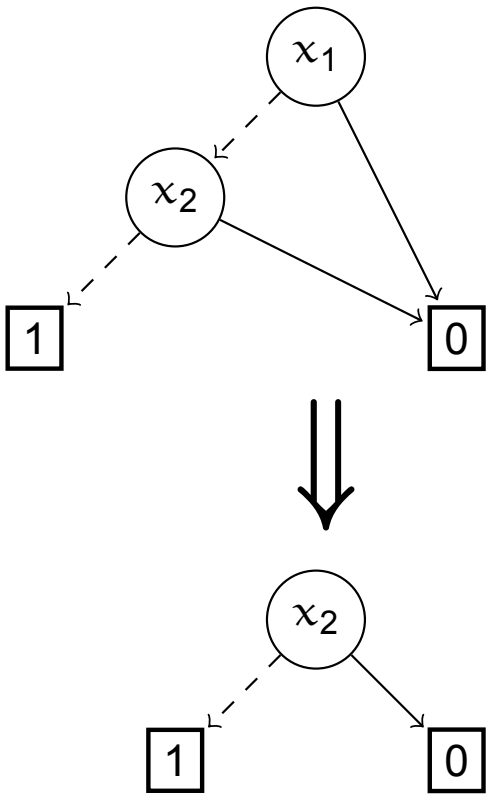
Algorithmen auf ROBDD

Der `restrict` Algorithmus

$$\mathcal{B}((\neg x_1 \wedge \neg x_2)[\top/x_1]) =$$
$$\texttt{restrict}(\mathcal{B}(\neg x_1 \wedge \neg x_2), x_1, \top)$$



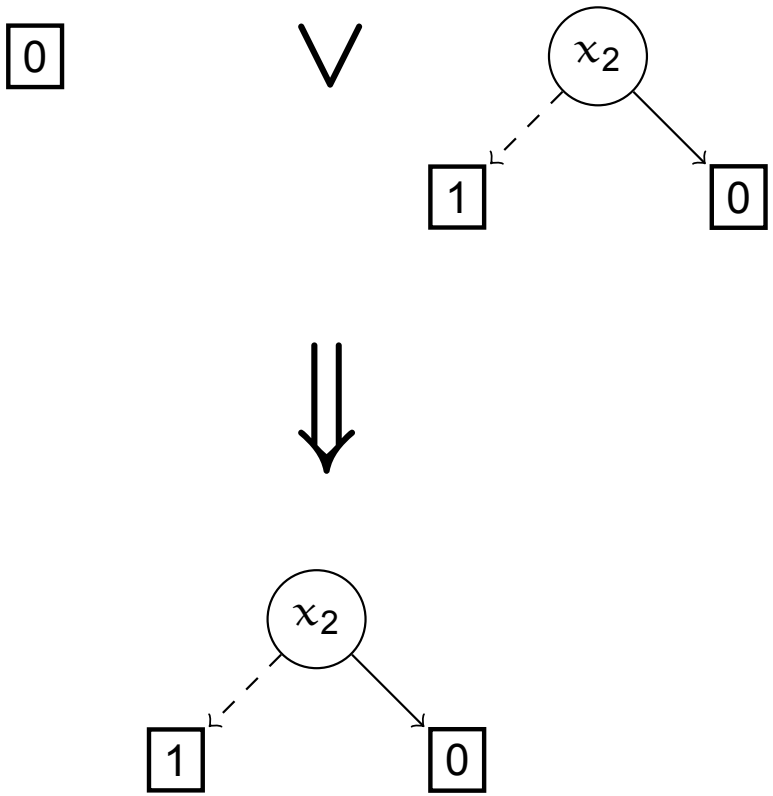
$$\mathcal{B}((\neg x_1 \wedge \neg x_2)[\bot/x_1]) =$$
$$\texttt{restrict}(\mathcal{B}(\neg x_1 \wedge \neg x_2), x_1, \bot)$$



Algorithmen auf ROBDD

Der `exists` Algorithmus

$$\mathcal{B}((\neg x_1 \wedge \neg x_2)[\top/x_1] \vee (\neg x_1 \wedge \neg x_2)[\bot/x_1]) = \texttt{exists}(\mathcal{B}(\neg x_1 \wedge \neg x_2), x_1)$$



Erweiterung: Elimination mehrerer Variablen $X = \{x_0, \ldots, x_n\}$ mit

$$\texttt{exists}(\mathcal{B}, X)$$

Grundidee des Algorithmus

1. Konstruiere ein BDD, das die Menge aller Zustände der Kripke-Struktur beschreibt
2. Konstruiere ein BDD, das die Zustandsübergänge beschreibt
3. Kombiniere die beiden BDD zu einem BDD, dass die gesamte Kripke-Struktur beschreibt
4. Führe den Algorithmus `modelCheckCTL` nun statt auf der Kripke-Struktur auf dem BDD aus

Definition (Mengenoperationen)

Beschreiben die beiden Formeln φ_1 und φ_2 jeweils die Zustandsmengen S_1 und S_2, so können wir die Mengenoperationen wie folgt realisieren

- $S_1 \cup S_2$ entspricht $\varphi_1 \vee \varphi_2$
- $S_1 \cap S_2$ entspricht $\varphi_1 \wedge \varphi_2$
- $S_1 - S_2$ entspricht $\varphi_1 \wedge \neg\varphi_2$

Darstellung von Zustandsmengen

Übersetzen eines einzelnen Zustandes s

Konjunktion aller Label, die an dem Zustand vorkommen (positive Literale) und die nicht vorkommen (negative Literale):

$$\mathcal{T}_S(s) = \bigwedge_{l \in \mathcal{L}} \lambda(l) \text{ mit } \lambda(l) = l, \text{ falls } l \in L(s), \text{ sonst } \neg l$$

Übersetzen einer Zustandsmenge

Für eine Zustandsmenge $\{s_0, \ldots, s_n\}$: Disjunktion über alle $\mathcal{T}_S(s_i)$:

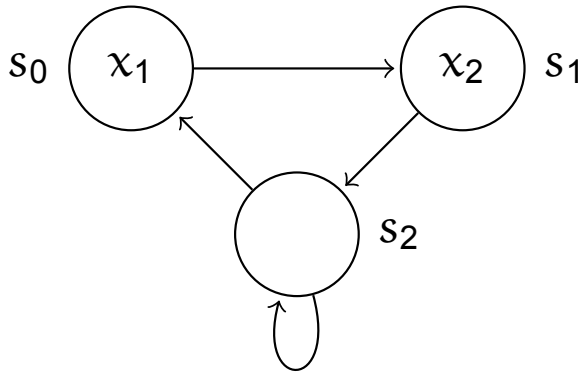
$$\mathcal{T}_S(\{s_0, \ldots, s_n\}) = \bigvee_{i \in \{0,\ldots,n\}} \mathcal{T}_S(s_i)$$

Hinweis

Zustände werden eindeutig über ihre Labelmengen identifiziert. Gibt es mehrere Zustände mit gleichen Beschriftungen, so müssen diese durch das Einführen von Hilfsvariablen unterschieden werden.

Darstellung von Zustandsmengen

Beispiel



Beispiel (Zustandsmengen als Boolesche Formel)

Menge der Zustände M	$\mathcal{T}_S(M)$
$\varnothing$	$\bot$
$\{s_0\}$	$x_1 \wedge \neg x_2$
$\{s_1\}$	$\neg x_1 \wedge x_2$
$\{s_2\}$	$\neg x_1 \wedge \neg x_2$
$\{s_0, s_1\}$	$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$
$\{s_0, s_2\}$	$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge \neg x_2)$
$\{s_1, s_2\}$	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$
S	$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$

Darstellung von Zustandsübergängen

- Führe eine Kopie x' jeder Variable x ein
- x ist die Belegung der Variable *vor* dem Übergang
- x' ist die Belegung der Variable *nach* dem Übergang
- Wir bezeichnen die Kopie eines Zustandes s mit s'

Übersetzen eines Zustandsübergangs

Der Übergang $s_0 \to s_1$ wird dann dargestellt als:

$$\mathcal{T}_T(s_0 \to s_1) = \mathcal{T}_S(s_0) \wedge \mathcal{T}_S(s_1')$$

Übersetzen einer Menge an Zustandsübergängen

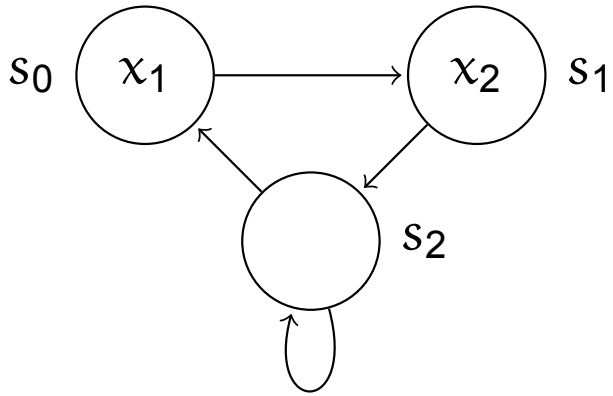
Für Zustandsübergänge $t_0, \ldots, t_n$ ist die Übersetzung

$$\mathcal{T}_T(\{t_0, \ldots, t_n\}) = \bigvee_{i \in \{0,\ldots,n\}} \mathcal{T}_T(t_i)$$

Übersetzung alle Zustandsübergänge: $\mathcal{T}_T(\longrightarrow)$

Darstellung von Zustandsübergängen

Beispiel



Beispiel (Zustandsübergänge als Formel)

$(x_1 \wedge \neg x_2 \wedge \neg x_1' \wedge x_2') \vee$	$s_0 \to s_1$
$(\neg x_1 \wedge x_2 \wedge \neg x_1' \wedge \neg x_2') \vee$	$s_1 \to s_2$
$(\neg x_1 \wedge \neg x_2 \wedge \neg x_1' \wedge \neg x_2') \vee$	$s_2 \to s_2$
$(\neg x_1 \wedge \neg x_2 \wedge x_1' \wedge \neg x_2')$	$s_2 \to s_0$

Implementierung der Urbilder

Das Universelle Urbild $\texttt{pre}_\forall$

$$\texttt{pre}_\forall(Y) = S - \texttt{pre}_\exists(S - Y)$$

Das Existenzielle Urbild $\texttt{pre}_\exists$

Berechnung von $\texttt{pre}_\exists(Y)$

- Wir suchen das Urbild, d.h. Vorgänger in der Übergangsrelation

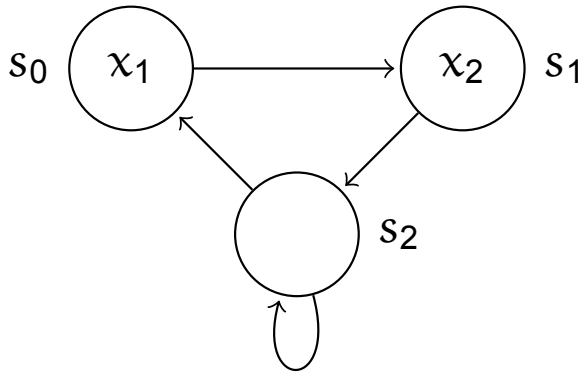
$\Rightarrow$ die Variablen in Y werden durch ihre Kopien ersetzt (Y')

- Bilde die Formel für $\mathcal{T}_T(\longrightarrow) \wedge \mathcal{T}_S(Y')$
- Eliminiere alle Kopien der Variablen aus der Formel

$$\texttt{exists}(\texttt{apply}(\mathcal{B}(\mathcal{T}_T(\longrightarrow)), \mathcal{B}(\mathcal{T}_S(Y')), \wedge), \text{vars}(Y'))$$

Implementierung der Urbilder

Beispiel



Beispiel (Berechnung von $\mathtt{pre}_\exists(\{s_2\})$)

$$\begin{aligned}\mathcal{T}_T(\longrightarrow) \wedge \mathcal{T}_S(Y') &= [(x_1 \wedge \neg x_2 \wedge \neg x_1' \wedge x_2') \vee (\neg x_1 \wedge x_2 \wedge \neg x_1' \wedge \neg x_2') \vee \\ &\quad (\neg x_1 \wedge \neg x_2 \wedge \neg x_1' \wedge \neg x_2') \vee (\neg x_1 \wedge \neg x_2 \wedge x_1' \wedge \neg x_2')] \wedge \\ &\quad \neg x_1' \wedge \neg x_2' \\ &= [(\neg x_1 \wedge x_2 \wedge \neg x_1' \wedge \neg x_2') \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_1' \wedge \neg x_2')] \wedge \\ &\quad \neg x_1' \wedge \neg x_2'\end{aligned}$$

- Elimination von x_1': $[(\neg x_1 \wedge x_2 \wedge \neg x_2') \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_2')] \wedge \neg x_2'$
- Elimination von x_2': $(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$ (*beschreibt* $\{s_1, s_2\}$)

Der Algorithmus

Wir wissen, wie wir Zustände, $\cup, \cap, -, \mathtt{pre}_\forall$ und $\mathtt{pre}_\exists$ mit BDD implementieren können, d.h. unveränderter Algorithmus

Algorithmus: $\mathtt{modelCheckCTL}(\mathcal{M}, \psi)$

Eingabe: Kripke-Struktur $\mathcal{M}$ und CTL Formel ψ
Ausgabe: Menge an Zuständen $\subseteq S$, an denen ψ gilt

1. Reduziere ψ auf die Operatoren $\bot$, $\neg$, $\wedge$ $\mathbf{AF}$, $\mathbf{EU}$ und $\mathbf{EX}$
2. $\mathtt{modelCheckCTL}(\mathcal{M}, \psi) = \psi$ $\mathtt{match}$

$$
\begin{aligned}
&\bot && \rightsquigarrow \varnothing \\
\mid\ &v \in \mathcal{V} && \rightsquigarrow \{s \in S \mid v \in L(s)\} \\
\mid\ &\neg\varphi && \rightsquigarrow S - \mathtt{modelCheckCTL}(\mathcal{M}, \varphi) \\
\mid\ &\varphi_1 \wedge \varphi_2 && \rightsquigarrow \mathtt{modelCheckCTL}(\mathcal{M}, \varphi_1) \cap \mathtt{modelCheckCTL}(\mathcal{M}, \varphi_2) \\
\mid\ &\mathbf{EX}\,\varphi && \rightsquigarrow \mathtt{mc}_{\mathbf{EX}}(\mathcal{M}, \varphi) \\
\mid\ &\mathbf{AF}\,\varphi && \rightsquigarrow \mathtt{mc}_{\mathbf{AF}}(\mathcal{M}, \varphi) \\
\mid\ &\mathbf{E}[\varphi_1\,\mathbf{U}\,\varphi_2] && \rightsquigarrow \mathtt{mc}_{\mathbf{EU}}(\mathcal{M}, \varphi_1, \varphi_2)
\end{aligned}
$$

Komplexität

Komplexität wichtiger BDD Operationen

- Äquivalenztest auf zwei BDD (Fixpunkt finden): $\mathrm{O}(|\mathcal{B}|)$
- Reduzieren eines BDD: $\mathrm{O}(|\mathcal{B}| \cdot \log |\mathcal{B}|)$
- $\texttt{apply}(\mathcal{B}_1, \mathcal{B}_2, \circ)$: $\mathrm{O}(|\mathcal{B}_1| \cdot |\mathcal{B}_2|)$
- $\texttt{restrict}(\mathcal{B}, v, \mathrm{b})$: $\mathrm{O}(|\mathcal{B}| \cdot \log |\mathcal{B}|)$
- $\texttt{exists}(\mathcal{B}, X)$: NP-vollständig

Existentielle Quantorenelimination auf Booleschen Formeln

Aktuelle Forschung am Arbeitsbereich:

- DNNF kompilieren & projizieren
- Projizierte Model Enumeration
- Portfolio Ansatz

Literaturtipps

Literaturhinweis

- *M. Huth & M. Ryan.* **Logic in Computer Science** **Chapter 3.** Cambridge University Press, 2004.
- *E. M. Clarke, O. Grumberg & D. A. Peled.* **Model Checking.** The MIT Press, 1999.

Web Links

- `http://nusmv.fbk.eu/` — Model Checker NuSMV

Motivation

Model Checking

CTL Model Checking

Symbolic Model Checking

BDD

Der Algorithmus

Literatur