Model-Checking

Markus Lohrey

Universität Siegen

http://www.eti.uni-siegen.de/ti/lehre/ws1718/model-checking/index.html?lang=de

WS 2017/2018

Allgemeines

Überblick:

1. Einführung
2. LTL: linear temporal logic
3. CTL: computation tree logic
4. Symbolisches Model-Checking
5. Model-Checking von Pushdownsystemen

Litratur:

- Baier, Katoen. Principles of Model Checking, MIT Press 2008
- Clarke, Grumberg, Peled. Model-Checking. MIT-Press 1999
- Hofmann, Lange. Automatentheorie und Logik, Springer 2011

Korrekte Software/Hardware

Drei berühmte Software/Hardware Fehler:

- Pentium-Fehler (1993): Kosten ca 475 Millionen Dollar
- Ariane 5 (1996): Kosten ca 600 Millionen Dollar
- Mars Orbiter (1999): 125 Millionen Dollar

Wie können solche Fehler verhindert werden?

- Fehlervermeidung durch Verwendung von Software-Engineering-Methoden
- Testen
- Deduktive Verifikation (z.B: Hoare-Kalkül)
- Model-Checking

Idee des Model-Checking

Eigenschaften von Systemen (Hardware oder Software) sollen automatisch verifiziert werden.

Beispiele für Systemeigenschaften:

- Für jede Eingabe terminiert das Programm P.
- Jeder Druckjob wird irgendwann auch bedient.
- In jedem Systemablauf gilt: Buffergröße $\leq$ 1GB

In der Informatik haben wir es mit zwei Typen von Systemen zu tun:

- Hardwaresysteme: typischerweise endlicher aber extrem großer Zustandsraum ($2^{\text{Speichergröße}}$)
- Softwaresysteme: typischerweise unendlicher Zustandsraum, z. B. aufgrund von Variablen mit unendlichen Wertebereichen (integers, reals)

Transitionsgraphen

Systeme (endlich oder unendlich) können abstrakt als Transitionsgraphen modelliert werden.

Definition (Transitionsgraph)

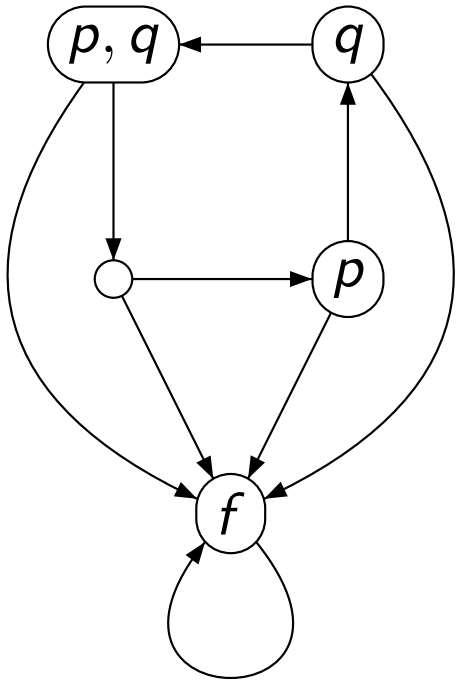
Ein Transitionsgraph ist ein Tupel $T = (V, E, \Pi, \pi)$ mit:

- V ist ein beliebige Menge (die Zustandsmenge oder Menge der Knoten).
- $E \subseteq V \times V$ ist die Menge der Kanten (Transitionen oder Systemübergänge).
- Π ist eine endliche Menge von Knotenmarkierungen (Propositionen).
- $\pi : V \to 2^{\Pi}$, $\pi(v)$ für $v \in V$ ist die Menge der in v geltenden Propositionen (2^{Π} = Menge aller Teilmengen von Π).

Alternative Bezeichnungen: Kripke-Struktur, knotenbeschrifteter (gerichteter) Graph.

Transitionsgraphen

Beispiel: Die Symbole in den Knoten sind die Propositionen, die dem Knoten zugeordnet sind.



Transitionsgraphen

Beispiel: Sequentieller Schaltkreis C mit

- n Inputbits $x_1, \ldots, x_n$,
- m Outputbits $y_1, \ldots, y_m$ und
- k internen Bits (Register) $z_1, \ldots, z_k$

Die Dynamik von C kann durch zwei Funktionen $f_y : \{0,1\}^{n+k} \to \{0,1\}^m$ und $f_z : \{0,1\}^{n+k} \to \{0,1\}^k$ dargestellt werden.

Falls $x_1, \ldots, x_n$ die aktuellen Werte der Inputbits sind und $z_1, \ldots, z_k$ die aktuellen Werte der internen Bits sind, dann sind

- $f_y(x_1, \ldots, x_n, z_1, \ldots, z_k)$ sind die Werte der m Outputbits $y_1, \ldots, y_m$, und
- $f_z(x_1, \ldots, x_n, z_1, \ldots, z_k)$ die neuen Werte der k internen Bits $z_1, \ldots, z_k$.

Transitionsgraphen

C kann durch den folgenden Transitionsgraphen $T = (V, E, \Pi, \pi)$ modelliert werden:

- $V = \{0,1\}^{n+k}$.

  Ein Knoten $v \in V$ repräsentiert die aktuellen Werte der n Inputbits und k internen Bits.

- $E = \{(\langle \overline{a}, \overline{c} \rangle, \langle \overline{a}', f_z(\overline{a}, \overline{c}) \rangle) \mid \overline{a}, \overline{a}' \in \{0,1\}^n, \overline{c} \in \{0,1\}^k\}$

  Beachte: Die Inputbits werden bei einer Transition nichtdeterministisch neu gesetzt.

  Idee: Inputbits können nicht kontrolliert werden.

- $\Pi = \{x_1, \ldots, x_n, y_1, \ldots, y_m, z_1, \ldots, z_k\}$
- $\pi(\overline{a}, \overline{c}) = \{x_i \mid \overline{a}_i = 1\} \cup \{y_i \mid f_y(\overline{a}, \overline{c})_i = 1\} \cup \{z_i \mid \overline{c}_i = 1\}$ für alle $\overline{a} \in \{0,1\}^n$, $\overline{c} \in \{0,1\}^k$.

Transitionsgraphen

Sei $T = (V, E, \Pi, \pi)$ ein Transitionsgraph.

Zur Erinnerung: Für Knoten $u, v \in V$ gilt $(u, v) \in E^*$ genau dann, wenn es $n \geq 0$ und Knoten $v_0, v_1, \ldots, v_n \in V$ gibt mit:

$$u = v_0, \;\; v = v_n \;\; \text{und} \bigwedge_{0 \leq i \leq n-1} (v_i, v_{i+1}) \in E.$$

Für $v \in V$ sei:

$$\begin{aligned}
\mathrm{pre}_T(v) &= \{u \in V \mid (u, v) \in E\} \\
\mathrm{suc}_T(v) &= \{u \in V \mid (v, u) \in E\} \\
\mathrm{pre}^*_T(v) &= \{u \in V \mid (u, v) \in E^*\} \\
\mathrm{suc}^*_T(v) &= \{u \in V \mid (v, u) \in E^*\}
\end{aligned}$$

Für $U \subseteq V$ und $X \in \{\mathrm{pre}, \mathrm{suc}, \mathrm{pre}^*, \mathrm{suc}^*\}$ sei $X_T(U) = \bigcup_{v \in U} X_T(v)$.

Repräsentation endlicher Transitionsgraphen

Die Standardrepräsentation eines endlichen Transitionsgraphen $T = (V, E, \Pi, \pi)$ besteht aus Adjazenzlisten.

Wir speichern für jeden Knoten $v \in V$

- eine Liste mit allen Knoten aus $\mathrm{suc}_T(v)$, sowie
- eine Liste mit allen Propositionen in $\pi(v)$.

Für manche Anwendungen ist es nützlich, für jeden Knoten $v \in V$ eine weitere Liste mit allen Knoten in $\mathrm{pre}_T(v)$ zu haben.

Diese Vorgängerlisten können in Zeit $O(|V| + |E|)$ aus den Adjazenzlisten erzeugt werden.

Später werden wir kompaktere Repräsentationen sehr großer Transitionsgraphen, wie sie in der Hardwareverifikation auftreten, kennenlernen (ordered binary decision diagrams).

Abstraktion

Transitionsgraphen sind ein abstrahiertes Modell für reale Systeme:

- Systemzustände (Momentaufnahmen) werden auf endlich viele Systemeigenschaften (= Propositionen) reduziert.
- Konkrete Werte von z.B. Integervariablen gehen dabei verloren.
- Dies ist der Preis, den man zahlen muss, um der Unentscheidbarkeit zu entgehen.

Erreichbarkeit

Die wohl wichtigste Frage, für die man sich beim Model-Checking interessiert, ist Erreichbarkeit.

Genauer: Für einen Transitionsgraphen $T = (V, E, \Pi, \pi)$, einen initialen Knoten $v \in V$, und eine Zielmenge $U \subseteq V$, will man wissen, ob es einen Pfad von v in die Menge U gibt (kurz: U ist von v erreichbar).

Formal: $v \in \mathrm{pre}_T^*(U)$

Die Menge U könnte z.B. eine Menge von kritischen Systemzuständen, die auf jeden Fall vermieden werden soll, repräsentieren.

Wenn dann U von v erreichbar ist, heißt das, dass v nicht sicher ist.

Erreichbarkeit

Durch Tiefensuche kann Erreichbarkeit in linearer Zeit $O(|V| + |E|)$ überprüft werden.

Aber:

- Wenn z.B. $|V| = 2^{1024}$ (Hardware mit 1024 bits) ist dies nicht mehr praktikabel.
- Ist V gar unendlich, so muss Erreichbarkeit nicht mehr entscheidbar sein.
- In der Tat: Das unentscheidbare Halteproblem für Turingmaschinen (oder z.B. C-Programme) kann als ein Erreichbarkeitsproblem betrachtet werden:

  Kann von der Anfangskonfiguration einer Turingmaschine eine Konfiguration, wo der Zustand ein Endzustand ist, erreicht werden?

Logiken

Komplexere Eigenschaften von Systemen (= Transitionsgraphen) werden wir durch Logiken beschreiben.

Typische Logiken zur Beschreibung von Systemeigenschaften:

- LTL: linear temporal logic
- CTL: computation tree logic

Das Model-Checking Problem:
Für eine Formel φ einer Logik L und einen Transitionsgraphen T wollen wir überprüfen, ob φ in T gilt, kurz $T \models \varphi$.

Vorteile der Beschreibung von Systemeigenschaften mittels Logik:

- Formal definierte Semantik
- Modulare Definition von Systemeigenschaften

Erfolge des Model-Checking

Es gibt mittlerweile eine Reihe von (zum Teil auch industriell eingesetzten) Model-Checking Wergzeugen, siehe z.B.

`https://en.wikipedia.org/wiki/List_of_model_checking_tools`.

Einige Erfolge:

- 1992: Validierung des Cache-Kohärenz-Protokolls des IEEE-Futurebus+ mittels SMV (Symbolic Model Verifier), siehe

  `http://www.cs.cmu.edu/~emc/papers/Conference%20Papers/95_verification_fbc_protocol.pdf`.

- 2001: Verifikation der Fließkomma-Einheit des Pentium 4. Fehler im FADD-Befehl gefunden, siehe

  `http://www.cs.rice.edu/~vardi/comp607/bentley.pdf`.

- 2000-2004: SLAM Projekt zur automatischen Verifikation von Windows XP Gerätetreibern, siehe

  `http://research.microsoft.com/pubs/70038/tr-2004-08.pdf`

Syntax von LTL

LTL (linear temporal logic) ist eine beliebte Logik zur Beschreibung (un)erwünschter Systemabläufe.

Sei Π eine endliche Menge von Propositionen und sei $\Sigma = 2^{\Pi}$.

Definition (LTL-Formeln)

Die Menge LTL(Π) aller LTL-Formeln über Π ist die bezüglich Inklusion kleineste Menge von Formeln mit:

- $\Pi \subseteq \text{LTL}(\Pi)$
- Wenn $\varphi, \psi \in \text{LTL}(\Pi)$, dann auch $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi) \in \text{LTL}(\Pi)$
- Wenn $\varphi, \psi \in \text{LTL}(\Pi)$, dann auch $\mathsf{X}\varphi$, $(\varphi \,\mathsf{U}\, \psi) \in \text{LTL}(\Pi)$.

Überflüssige Klammern lassen wir auch weg.

Unendliche Wörter

Wir interpretieren LTL-Formeln über unendlichen Wörtern über dem Alphabet Σ (auch bekannt als ω-Wörter).

Definition (ω-Wörter)

Mit Σ^ω bezeichnen wir die Menge aller ω-Wörter über dem Alphabet Σ:

$$\Sigma^\omega = \{a_1 a_2 a_3 a_4 \cdots \mid \forall i \geq 1 : a_i \in \Sigma\}$$

Formal kann man ein ω-Wort $w = a_1 a_2 a_3 a_4 \cdots$ auch als eine Funktion $w : \mathbb{N} \to \Sigma$ definieren (hier: $w(i) = a_i$).

Für ein ω-Wort $w = a_1 a_2 a_3 a_4 \cdots$ und $i \geq 1$ definieren wir das ω-Wort

$$w^i = a_i a_{i+1} a_{i+2} \cdots .$$

Insbesondere gilt $w^1 = w$.

Semantik von LTL auf ω-Wörtern

Für $w = (a_1 a_2 \cdots) \in \Sigma^\omega$ und $\varphi \in \mathrm{LTL}(\Pi)$ gilt $w \models \varphi$ (φ gilt in w) genau dann, wenn einer der folgenden Fälle gilt:

- $\varphi = p \in \Pi$ und $p \in a_1$
- $\varphi = \neg\psi$ und $w \not\models \psi$
- $\varphi = \psi \wedge \theta$ und ($w \models \psi$ und $w \models \theta$)
- $\varphi = \psi \vee \theta$ und ($w \models \psi$ oder $w \models \theta$)
- $\varphi = \mathrm{X}\psi$ und $w^2 \models \psi$
- $\varphi = \psi \,\mathrm{U}\, \theta$ und es existiert ein $k \geq 1$ mit
  - $w^k \models \theta$ und
  - $w^i \models \psi$ für alle $1 \leq i \leq k-1$

Intuitive Bedeutung von

- $\mathrm{X}\psi$ (NEXT ψ): Im nächsten Schritt gilt ψ.
- $\psi \,\mathrm{U}\, \theta$ (ψ UNTIL θ): Irgendwann gilt θ und bis dahin gilt ψ.

Wichtige LTL-Abkürzungen

- $\mathsf{F}\psi := (p \vee \neg p)\,\mathsf{U}\,\psi$: irgendwann (finally) gilt ψ

  Für jedes ω-Wort w gilt:

$$\begin{aligned} & w \models \mathsf{F}\psi \\ \iff\ & w \models (p \vee \neg p)\,\mathsf{U}\,\psi \\ \iff\ & \exists k \geq 1 : w^k \models \psi \text{ und } \forall 1 \leq i \leq k-1 : w^i \models (p \vee \neg p) \\ \iff\ & \exists k \geq 1 : w^k \models \psi \end{aligned}$$

- $\mathsf{G}\psi := \neg\mathsf{F}(\neg\psi)$: ψ gilt immer (globally).

  Für jedes ω-Wort w gilt:

$$\begin{aligned} & w \models \mathsf{G}\psi \\ \iff\ & \forall k \geq 1 : w^k \not\models \neg\psi \\ \iff\ & \forall k \geq 1 : w^k \models \psi \end{aligned}$$

Wichtige LTL-Abkürzungen

- $\varphi \,\mathsf{R}\, \psi := \neg(\neg\varphi \,\mathsf{U}\, \neg\psi)$: Release- oder Ablösungs-Operator

  Für jedes ω-Wort w gilt

$$
\begin{aligned}
& w \models \varphi \,\mathsf{R}\, \psi \\
\iff\ & w \models \neg(\neg\varphi \,\mathsf{U}\, \neg\psi) \\
\iff\ & \forall k \geq 1 : w^k \not\models \neg\psi \text{ oder } \exists 1 \leq i \leq k-1 : w^i \not\models \neg\varphi \\
\iff\ & \forall k \geq 1 : w^k \models \psi \text{ oder } \exists 1 \leq i \leq k-1 : w^i \models \varphi \\
\iff\ & (\forall k \geq 1 : w^k \models \psi) \text{ oder } \exists k \geq 1 : w^k \models \varphi \text{ und} \\
& \qquad\qquad \forall 1 \leq i \leq k : w^i \models \psi
\end{aligned}
$$

  Intuition: ψ wird durch φ abgelöst.

LTL-Beispiele

Wir schreiben im folgenden p (bzw. q) für die Menge $\{p\}$ (bzw. $\{q\}$) in ω-Wörtern.

Für $u \in \Sigma^+$ schreiben wir u^ω für das unendliche Wort $uuuuuu\cdots$.

- FGp: Ab einen bestimmten Zeitpunkt gilt p für immer.

  Es gilt z.B. $qpqpqp^\omega \models \mathsf{FG}p$.

- GFp: p gilt unendlich oft.

  Es gilt z.B. $\prod_{i\geq 1}(q^i p) \models \mathsf{GF}p$.

- G$(q \rightarrow \mathsf{F}p)$: Wann immer q gilt, gilt zu einem späteren Zeitpunkt irgendwann p.

  Es gilt z.B. $\prod_{i\geq 1}(q^i p) \models \mathsf{G}(q \rightarrow \mathsf{F}p)$ aber auch $\emptyset^\omega \models \mathsf{G}(q \rightarrow \mathsf{F}p)$.

Größe einer LTL-Formel

Für genaue Aussagen über die Laufzeit von Algorithmen müssen wir die Größe einer LTL-Formel sinnvoll definieren.

Definition (Größe einer LTL-Formel)

Für eine LTL-Formel $\varphi \in \mathrm{LTL}(\Pi)$ definieren wir die Größe $|\varphi|$ induktiv wie folgt:

- $|p| = 1$ für $p \in \Pi$
- $|\neg\psi| = |\mathsf{X}\psi| = |\mathsf{F}\psi| = |\mathsf{G}\psi| = 1 + |\psi|$
- $|\theta \wedge \psi| = |\theta \vee \psi| = |\theta\,\mathsf{U}\,\psi| = |\theta\,\mathsf{R}\,\psi| = 1 + |\theta| + |\psi|$

Äquivalenz

Definition (Äquivalenz von LTL-Formeln)

Zwei LTL-Formeln $\varphi, \psi \in \mathrm{LTL}(\Pi)$ sind äquivalent, kurz $\varphi \equiv \psi$, genau dann, wenn für alle ω-Wörter $w \in (2^{\Pi})^{\omega}$ gilt:

$$w \models \varphi \iff w \models \psi.$$

Beispiele:

- $\mathsf{X}(\varphi \wedge \psi) \equiv \mathsf{X}\varphi \wedge \mathsf{X}\psi$
- $\mathsf{X}(\varphi \vee \psi) \equiv \mathsf{X}\varphi \vee \mathsf{X}\psi$
- $\mathsf{G}(\varphi \wedge \psi) \equiv \mathsf{G}\varphi \wedge \mathsf{G}\psi$
- $\mathsf{F}(\varphi \vee \psi) \equiv \mathsf{F}\varphi \vee \mathsf{F}\psi$
- $\mathsf{FF}\varphi \equiv \mathsf{F}\varphi$
- $\mathsf{GG}\varphi \equiv \mathsf{G}\varphi$
- $\theta \,\mathsf{R}\, \psi \equiv \mathsf{G}\psi \vee (\psi \,\mathsf{U}\, (\theta \wedge \psi))$

Abwicklung von U and R

Die beiden folgenden Äquivalenzen sind besonders wichtig, daher formulieren wir sie als Lemma:

Lemma 1 (Expansionsgesetze)

Folgende Äquivalenzen gelten für alle LTL-Formeln φ und ψ:

$$\begin{aligned} \varphi \,\mathsf{U}\, \psi &\equiv \psi \vee (\varphi \wedge \mathsf{X}(\varphi \,\mathsf{U}\, \psi)) \\ \varphi \,\mathsf{R}\, \psi &\equiv \psi \wedge (\varphi \vee \mathsf{X}(\varphi \,\mathsf{R}\, \psi)) \end{aligned}$$

Beweis: Übung

Positive Normalform

Im folgenden erlauben wir den release-Operator R in LTL-Formeln.

Definition (positive Normalform)

Eine LTL-Formel ist in positive Normalform, falls das Negationssymbol $\neg$ nur direkt vor Propositionen aus Π auftaucht.

Beispiel: Die Formel

$$\neg((p \wedge \neg q)\,\mathsf{U}\,(\mathsf{X}p))$$

ist nicht in positiver Normalform.

Sie ist aber äquivalent zu folgender Formel in positiver Normalform:

$$(\neg p \vee q)\,\mathsf{R}\,\mathsf{X}(\neg p).$$

Positive Normalform

Lemma 2

Zu jeder LTL-Formel existiert eine äquivalente LTL-Formel in positiver Normalform.

Beweis: Negationen können durch Ausnutzen folgender Äquivalenzen direkt vor Propositionen gedrückt werden.

- $\neg\neg\psi \equiv \psi$
- $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$
- $\neg \mathsf{X}\psi \equiv \mathsf{X}\neg\psi$
- $\neg(\varphi \,\mathsf{U}\, \psi) \equiv \neg\varphi \,\mathsf{R}\, \neg\psi$
- $\neg(\varphi \,\mathsf{R}\, \psi) \equiv \neg\varphi \,\mathsf{U}\, \neg\psi$

Semantik von LTL auf Transitionsgraphen

Definition (Menge der in v beginnenden unendlichen Pfade)

Für einen Transitionsgraphen $T = (V, E, \Pi, \pi)$ und $v \in V$ sei

$$\text{path}(T, v) = \{(v_1 v_2 v_3 \cdots) \in V^\omega \mid v_1 = v, (v_i, v_{i+1}) \in E \text{ für alle } i \geq 1\}$$

Definition (Menge der in v beginnenden ω-traces (Spuren))

Für einen Transitionsgraphen $T = (V, E, \Pi, \pi)$ und $v \in V$ sei

$$\omega\text{-tr}(T, v) = \{\pi(v_1)\pi(v_2)\pi(v_3)\cdots \mid (v_1 v_2 v_3 \cdots) \in \text{path}(T, v)\} \subseteq \Sigma^\omega.$$

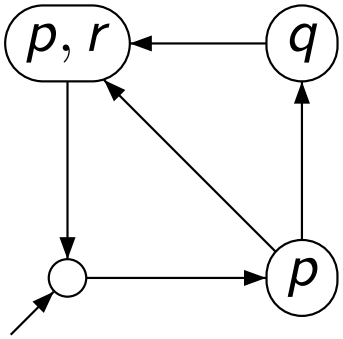
Definition (Semantik von LTL auf Transitionsgraphen)

Für $\varphi \in \text{LTL}(\Pi)$ schreiben wir $(T, v) \models \varphi$, falls $w \models \varphi$ für alle $w \in \omega\text{-tr}(T, v)$ gilt.

Vorsicht: $(T, v) \not\models \varphi$ ist nicht äquivalent zu $(T, v) \models \neg\varphi$!

Semantik von LTL auf Transitionsgraphen

Beispiel: Sei T folgender Transitionsgraph und v_0 der mit dem eingehenden Pfeil markierte Knoten.



Dann gilt z.B.

- $(T, v_0) \models \mathsf{GF}p$
- $(T, v_0) \models \mathsf{G}(p \vee \mathsf{X}p)$
- $(T, v_0) \models \mathsf{GF}r$

Aber $(T, v_0) \models \mathsf{GF}q$ gilt nicht.

LTL-Model-Checking

Definition (LTL-Model-Checking auf endlichen Transitionsgraphen)

INPUT: Ein endlicher Transitionsgraph $T = (V, E, \Pi, \pi)$, ein Knoten $v \in V$, und eine LTL-Formel $\varphi \in \mathrm{LTL}(\Pi)$.

FRAGE: Gilt $(T, v) \models \varphi$?

Apriori ist nicht klar, ob das LTL-Model-Checking Problem überhaupt entscheidbar ist.

Unser Ziel im folgenden ist zu zeigen, dass LTL-Model-Checking entscheidbar ist.

Wir verwenden hierzu den automatentheoretischen Ansatz und übersetzen eine LTL-Formel in einen sogenannten Büchiautomaten.

Büchiautomaten

Rein syntaktisch ist ein Büchiautomat genau das gleiche wie ein nichtdeterministischer endlicher Automat (siehe GTI).

Definition (nichtdeterministischer Büchiautomat, kurz NBA)

Ein nichtdeterministischer Büchiautomat (über dem Alphabet Σ) ist ein Tupel $B = (S, \Sigma, \delta, s_0, F)$, wobei gilt:

- S ist eine endliche Menge von Zuständen.
- Σ ist ein endliches Alphabet.
- $\delta \subseteq S \times \Sigma \times S$ ist die Transitionsrelation.
- s_0 ist der Anfangszustand.
- $F \subseteq S$ ist die Menge der Endzustände.

Büchiautomaten

Definition (Lauf und akzeptierender Lauf)

Sei $B = (S, \Sigma, \delta, s_0, F)$ ein NBA und $w = (a_1 a_2 a_3 \cdots) \in \Sigma^\omega$.

Ein Lauf von B auf w ist ein ω-Wort $(s_0 s_1 s_2 \cdots) \in S^\omega$ mit $(s_i, a_{i+1}, s_{i+1}) \in \delta$ für alle $i \geq 0$.

Dieser Lauf ist ein akzeptierender Lauf von B auf w, falls es unendlich viele $i \geq 0$ mit $s_i \in F$ gibt (oder äquivalent: es gibt ein $q \in F$ so dass $s_i = q$ für unendlich viele i gilt).

Definition (die von einem NBA akzeptierte Sprache)

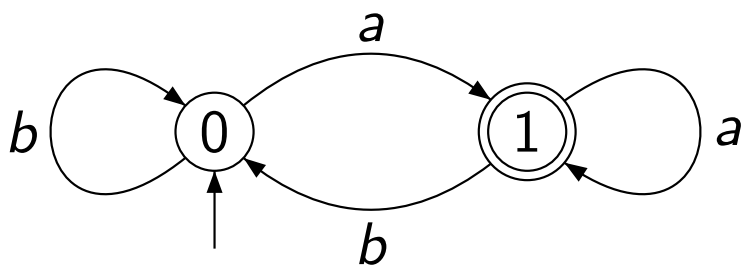
Die von dem NBA $B = (S, \Sigma, \delta, s_0, F)$ akzeptierte Sprache ist

$$L(B) = \{w \in \Sigma^\omega \mid \text{ es existiert ein akzeptierender Lauf von } B \text{ auf } w\}.$$

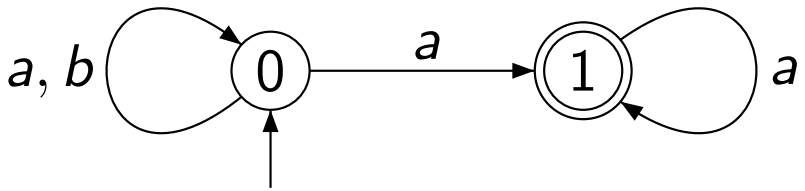
Beispiele für Büchiautomaten

Wir verwenden zum Zeichnen von Büchiautomaten die gleichen Konventionen wie bei normalen endlichen Automaten.

Beispiel 1: Dieser NBA akzeptierte alle ω-Wörter, die unendlich viele a's enthalten.



Beispiel 2: Dieser NBA akzeptierte $\{a, b\}^* a^\omega$.



Reguläre ω-Sprachen

Definition (reguläre ω-Sprachen)

Eine Sprache $L \subseteq \Sigma^\omega$ ist ω-regulär, falls ein NBA B mit $L(B) = L$ existiert.

Lemma 3

Eine Sprache $L \subseteq \Sigma^\omega$ ist ω-regulär genau dann, wenn

- $n \geq 0$ und
- reguläre Sprachen $U_1, V_1 \ldots, U_n, V_n \subseteq \Sigma^+$

existieren mit

$$L = \bigcup_{i=1}^{n} U_i V_i^\omega.$$

Beweis: Übung.

LTL $\rightarrow$ BA

Satz 4 (Vardi, Wolper 1994)

Für eine gegebene Formel $\varphi \in \mathrm{LTL}(\Pi)$ kann man einen NBA B über dem Alphabet $\Sigma := 2^{\Pi}$ mit $1 + |\varphi| \cdot 2^{2|\varphi|}$ vielen Zuständen konstruieren, so dass $L(B) = \{w \in \Sigma^{\omega} \mid w \models \varphi\}$ gilt.

Für den Beweis von Satz 4 ist es hilfreich, mit sogenannten verallgemeinerten Büchiautomaten zu arbeiten.

Definition (verallgemeinerter Büchiautomat, kurz VBA)

Ein verallgemeinerter Büchiautomat (über dem Alphabet Σ) ist ein Tupel $B = (S, \Sigma, \delta, I, F_0, \ldots F_{k-1})$, wobei gilt:

- S, Σ und δ wie bei einem (normalen) Büchiautomaten,
- $I, F_0, \ldots, F_{k-1} \subseteq S$, und
- $k \geq 1$.

Verallgemeinerte Büchiautomaten

Definition (Lauf und akzeptierender Lauf eines VBA)

Sei $B = (S, \Sigma, \delta, I, F_0, \ldots, F_{k-1})$ ein VBA und $w = (a_1 a_2 a_3 \cdots) \in \Sigma^\omega$.

Ein Lauf von B auf w ist ein ω-Wort $(s_0 s_1 s_2 \cdots) \in S^\omega$ mit $s_0 \in I$ und $(s_i, a_{i+1}, s_{i+1}) \in \delta$ für alle $i \geq 0$.

Dieser Lauf ist ein akzeptierender Lauf von B auf w, falls es für alle $0 \leq j \leq k-1$ unendlich viele $i \geq 0$ mit $s_i \in F_j$ gibt.

Definition (die von einem VBA akzeptierte Sprache)

Die von dem VBA $B = (S, \Sigma, \delta, I, F_0, \ldots, F_{k-1})$ akzeptierte Sprache ist

$$L(B) = \{w \in \Sigma^\omega \mid \text{ es existiert ein akzeptierender Lauf von } B \text{ auf } w\}.$$

VBA $\rightarrow$ NBA

Lemma 5

Aus einem VBA $B = (S, \Sigma, \delta, I, F_0, \ldots, F_{k-1})$ kann ein NBA A mit $1 + |S| \cdot k$ vielen Zuständen und $L(A) = L(B)$ konstruierte werden.

Beweis: Sei $k \geq 2$.

Schritt 1. Wir reduzieren zunächst k auf 1.

Definiere den VBA

$$B' = (S \times \{0, \ldots, k-1\}, \Sigma, \delta', I \times \{0\}, F_0 \times \{0\})$$

wobei $\delta' \subseteq (S \times \{0, \ldots, k-1\}) \times \Sigma \times (S \times \{0, \ldots, k-1\})$ wie folgt definiert ist:

$$\begin{aligned} \delta' \ = \ & \{\big((s,i), a, (p,i)\big) \mid (s,a,p) \in \delta, s \notin F_i\} \cup \\ & \{\big((s,i), a, (p, i+1 \bmod k)\big) \mid (s,a,p) \in \delta, s \in F_i\} \end{aligned}$$

VBA $\rightarrow$ NBA

Behauptung 1: $L(B) \subseteq L(B')$.

Sei $w \in L(B)$.

Also hat B einen akzeptierenden Lauf $r = (s_0 s_1 s_2 \cdots)$ auf w.

Für alle $0 \leq j \leq k-1$ existieren also unendlich viele $i \geq 0$ mit $s_i \in F_j$.

Wir wählen nun Positionen $0 = i_{-1} < i_0 < i_1 < i_2 < \cdots$ wie folgt aus:

$$i_p = \min\{j > i_{p-1} \mid s_j \in F_{p \bmod k}\} \text{ für alle } p \geq 0$$

Also kann der VBA B' zum Zeitpunkt i_p die Transition $\big((s_{i_p}, p \bmod k), a, (s_{i_p+1}, p+1 \bmod k)\big)$ durchführen.

Damit durchläuft B' unendlich oft die Menge $F_0 \times \{0\}$, d.h. $w \in L(B')$.

VBA $\rightarrow$ NBA

Behauptung 2: $L(B') \subseteq L(B)$.

Sei $w \in L(B')$.

Also hat B' einen akzeptierenden Lauf $r = (s_0, j_0)(s_1, j_1)(s_2, j_2) \cdots$ auf w.

Es existieren also unendlich viele $i \geq 0$ mit $s_i \in F_0$ und $j_i = 0$.

Damit aber die zweite Komponente j_i in r unendlich oft 0 durchläuft, muss unendlich oft der Zyklus $0, 1, 2, \ldots, k-1$ durchlaufen werden.

Insbesondere gibt es für alle $0 \leq j \leq k-1$ unendlich viele $i \geq 0$ mit $s_i \in F_j$.

Also ist $s_0 s_1 s_2 \cdots$ ein akzeptierender Lauf von B auf w, d.h. $w \in L(B)$.

VBA $\rightarrow$ NBA

Schritt 2: Wir reduzieren die Menge der Anfangszustände I auf einen Zustand.

Sei hierzu $B = (S, \Sigma, \delta, I, F)$ ein VBA mit einer einzigen Endzustandsmenge.

Sei $s_0 \notin S$ ein neuer Zustand.

Definiere den NBA

$$A = (S \cup \{s_0\}, \Sigma, \delta', s_0, F)$$

mit

$$\delta' = \delta \cup \{(s_0, a, p) \mid \exists s \in I : (s, a, p) \in \delta\}.$$

Dann gilt in der Tat $L(A) = L(B)$.

Schritt 1 und 2 kombiniert führen auf einen NBA mit $1 + |S| \cdot k$ vielen Zuständen. □

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

Sei $\varphi \in \mathrm{LTL}(\Pi)$, wobei alle $p \in \Pi$ auch in φ vorkommen sollen. Wegen Lemma 2 können wir voraussetzen, dass φ in positiver Normalform ist.

Definition (Fischer-Ladner-Abschluss)

Der Fischer-Ladner-Abschluss $\mathrm{FL}(\varphi)$ von φ ist die bezüglich Inklusion kleinste Menge von LTL-Formeln mit folgenden Eigenschaften:

- $\{\varphi\} \cup \Pi \cup \{\neg p \mid p \in \Pi\} \subseteq \mathrm{FL}(\varphi)$
- Wenn $\theta \vee \psi \in \mathrm{FL}(\varphi)$, dann $\theta \in \mathrm{FL}(\varphi)$ und $\psi \in \mathrm{FL}(\varphi)$.
- Wenn $\theta \wedge \psi \in \mathrm{FL}(\varphi)$, dann $\theta \in \mathrm{FL}(\varphi)$ und $\psi \in \mathrm{FL}(\varphi)$.
- Wenn $\mathsf{X}\psi \in \mathrm{FL}(\varphi)$, dann $\psi \in \mathrm{FL}(\varphi)$.
- Wenn $\theta \,\mathsf{U}\, \psi \in \mathrm{FL}(\varphi)$, dann $\mathsf{X}(\theta \,\mathsf{U}\, \psi) \in \mathrm{FL}(\varphi)$, $\theta \in \mathrm{FL}(\varphi)$, $\psi \in \mathrm{FL}(\varphi)$.
- Wenn $\theta \,\mathsf{R}\, \psi \in \mathrm{FL}(\varphi)$, dann $\mathsf{X}(\theta \,\mathsf{R}\, \psi) \in \mathrm{FL}(\varphi)$, $\theta \in \mathrm{FL}(\varphi)$, $\psi \in \mathrm{FL}(\varphi)$.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

Beispiel: $\mathrm{FL}\big((\neg p)\,\mathsf{U}\,(q \wedge r)\big)$ besteht aus folgenden Formeln:

$$\mathsf{X}((\neg p)\,\mathsf{U}\,(q \wedge r)),\ (\neg p)\,\mathsf{U}\,(q \wedge r),\ q \wedge r,\ p,\ \neg p,\ q,\ \neg q,\ r,\ \neg r.$$

Beachte: $|\mathrm{FL}(\varphi)| \leq 2 \cdot |\varphi|$, denn für jede Teilformel ψ von φ enthält $\mathrm{FL}(\varphi)$ höchstens zwei Formeln (ψ und $\mathsf{X}\psi$, bzw. p und $\neg p$).

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

Definition (Hintikka-Mengen)

Sei $\varphi \in \mathrm{LTL}(\Pi)$ wieder in positiver Normalform, und alle $p \in \Pi$ kommen in φ vor.

Eine Hintikka-Menge für φ ist eine Menge $M \subseteq \mathrm{FL}(\varphi)$ mit folgenden Eigenschaften.

- Für alle $p \in \Pi$ gilt entweder $p \in M$ oder $\neg p \in M$
- Wenn $\theta \wedge \psi \in M$ dann ($\theta \in M$ und $\psi \in M$).
- Wenn $\theta \vee \psi \in M$ dann ($\theta \in M$ oder $\psi \in M$).
- Wenn $\theta \,\mathsf{U}\, \psi \in M$, dann ($\psi \in M$ oder ($\theta \in M$ und $\mathsf{X}(\theta \,\mathsf{U}\, \psi) \in M$)).
- Wenn $\theta \,\mathsf{R}\, \psi \in M$, dann ($\psi \in M$ und ($\theta \in M$ oder $\mathsf{X}(\theta \,\mathsf{R}\, \psi) \in M$)).

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

$\mathcal{H}(\varphi) \subseteq 2^{\mathsf{FL}(\varphi)}$ sei die Menge aller Hintikka-Mengen für φ.

Intuition: Sei $w \in \Sigma^\omega$. Sei weiterhin $M = \{\psi \in \mathsf{FL}(\varphi) \mid w \models \psi\}$.
Offensichtlich muss M eine Hintikka-Mengen für φ sein.

Die Mengen in $\mathcal{H}(\varphi)$ sind also die Teilmengen des Fischer-Ladner-Abschlusses, die in einem ω-Wort wahr sind.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

Wir konstruieren nun aus einer LTL-Formel φ (wie immer in positiver Normalform) einen äquivalenten VBA B_φ.

Sei $\theta_1 \,\mathsf{U}\, \psi_1, \theta_2 \,\mathsf{U}\, \psi_2 \ldots, \theta_k \,\mathsf{U}\, \psi_k$ eine Auflistung aller in $\mathrm{FL}(\varphi)$ vorkommenden U-Formeln (o.B.d.A. $k \geq 1$).

Dann ist B_φ wie folgt definiert, wobei wie immer $\Sigma = 2^\Pi$:

$$B_\varphi = (\mathcal{H}(\varphi), \Sigma, \delta, I, F_1, \ldots, F_k)$$

mit

$$\begin{aligned} I &= \{M \mid \varphi \in M\} \\ F_j &= \{M \mid \text{ wenn } \theta_j \,\mathsf{U}\, \psi_j \in M, \text{ dann auch } \psi_j \in M\} \\ \delta &= \{(M, a, M') \mid a = \Pi \cap M, \text{ wenn } \mathsf{X}\psi \in M, \text{ dann } \psi \in M'\}. \end{aligned}$$

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

Sei nun $w = a_1 a_2 a_3 \cdots \in \Sigma^\omega$.

Behauptung 1: Wenn $w \models \varphi$, dann gilt $w \in L(B_\varphi)$.

Gelte also $w \models \varphi$.

Wir definieren einen akzeptierenden Lauf von B_φ auf w indem wir für alle $i \geq 1$ definieren:

$$M_i = \{\psi \in \mathsf{FL}(\varphi) \mid w^i \models \psi\}.$$

Dann gilt:

1. Für alle $i \geq 1$ gilt $M_i \in \mathcal{H}(\varphi)$.
2. $\varphi \in M_1$ da $w = w^1$ und $w \models \varphi$.
3. Für alle $i \geq 1$ gilt: Wenn $\mathsf{X}\psi \in M_i$, dann $\psi \in M_{i+1}$.
4. Für alle $i \geq 1$ gilt: $a_i = \Pi \cap M_i$.
5. Für alle $1 \leq j \leq k$ und $i \geq 1$ gilt: Wenn $\theta_j \,\mathsf{U}\, \psi_j \in M_i$ dann existiert ein $i' \geq i$ mit $\psi_j \in M_{i'}$.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

Dann ist $M_1 M_2 M_3 \cdots$ in der Tat ein akzeptierender Lauf von B_φ auf w:

- Wegen Punkt 1 ist jedes M_i ein Zustand von B_φ.
- Wegen Punkt 2 ist M_1 ein Anfangszustand von B_φ.
- Wegen Punkt 3 und 4 gilt $(M_i, a_i, M_{i+1}) \in \delta$ für alle $i \geq 1$.
- Wegen Punkt 5 wird jede Endzustandsmenge F_j $(1 \leq j \leq k)$ unendlich oft besucht:

  Sei hierzu $i \geq 1$ beliebig.

  Wir zeigen, dass ein $i' \geq i$ mit $M_{i'} \in F_j$ existiert.

  Falls $\theta_j \,\mathsf{U}\, \psi_j \notin M_i$, gilt $M_i \in F_j$.

  Falls $\theta_j \,\mathsf{U}\, \psi_j \in M_i$, gibt es ein $i' \geq i$ mit $\psi_j \in M_{i'}$.

  Dann gilt $M_{i'} \in F_j$.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

Behauptung 2: Wenn $w \in L(B_\varphi)$, dann gilt $w \models \varphi$.

Gelte $w \in L(B_\varphi)$ und sei $M_1 M_2 M_3 M_4 \cdots$ ein akzeptierender Lauf von B_φ auf w.

Wegen $\varphi \in M_1$ genügt es folgende Behauptung zu zeigen.

Behauptung 3: Für alle $i \geq 1$ und alle $\psi \in M_i$ gilt $w^i \models \psi$.

Wir zeigen Behauptung 3 durch Induktion über den Aufbau der Formel ψ.

1. Fall: $\psi = p \in \Pi \cap M_i$.

Da $(M_i, a_i, M_{i+1}) \in \delta$ gelten muss, folgt aus der Definition von δ:
$a_i = \Pi \cap M_i$ und damit $p \in a_i$.

Also gilt $w^i \models p$.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

2. Fall: $\psi = \neg p \in M_i$ mit $p \in \Pi$.

Wieder gilt $a_i = \Pi \cap M_i$.

Da $\neg p \in M_i \in \mathcal{H}(\varphi)$ gilt $p \notin M_i$, d.h. $p \notin a_i$ und $w^i \models \neg p$.

3. Fall: $\psi = \psi' \wedge \psi'' \in M_i$.

Da $\psi \in M_i \in \mathcal{H}(\varphi)$ gilt $\psi' \in M_i$ und $\psi'' \in M_i$.

Mit Induktion folgt $w^i \models \psi'$ und $w^i \models \psi''$, d.h. $w^i \models \psi$.

4. Fall: $\psi = \psi' \vee \psi'' \in M_i$.

Kann analog zu Fall 3 erledigt werden.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

5. Fall: $\psi = \mathsf{X}\psi' \in M_i$.

Da $M_1 M_2 M_3 \cdots M_i M_{i+1} \cdots$ ein Lauf von B_φ auf w ist, muss $\psi' \in M_{i+1}$ gelten.

Mit Induktion folgt $w^{i+1} \models \psi'$.

Also gilt $w^i \models \psi$.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

6. Fall: $\psi = \theta_j \,\mathsf{U}\, \psi_j$ für ein $1 \leq j \leq k$.

O.B.d.A. sei $j = 1$.

Aus $\theta_1 \,\mathsf{U}\, \psi_1 \in M_i \in \mathcal{H}(\varphi)$ folgt:

- $\psi_1 \in M_i$ oder
- $\theta_1 \in M_i$ und $\mathsf{X}(\theta_1 \,\mathsf{U}\, \psi_1) \in M_i$. d.h. $\theta_1 \,\mathsf{U}\, \psi_1 \in M_{i+1}$.

Falls $\psi_1 \in M_i$ folgt mit Induktion $w^i \models \psi_1$ und damit $w^i \models \theta_1 \,\mathsf{U}\, \psi_1$ und wir sind fertig.

Gelte also $\psi_1 \notin M_i$, $\theta_1 \in M_i$ und $\theta_1 \,\mathsf{U}\, \psi_1 \in M_{i+1}$.

Wir können nun das obige Argument wiederholen, und erhalten

- $\psi_1 \in M_{i+1}$ oder
- $\theta_1 \in M_{i+1}$ und $\theta_1 \,\mathsf{U}\, \psi_1 \in M_{i+2}$.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

Durch iterierte Anwendung des Arguments kommen wir auf einen der beiden folgenden Fälle:

- Es gibt ein $i' \geq i$ mit $\psi_1 \in M_{i'}$ und $\theta_1 \in M_j$ für alle $i \leq j < i'$.

  Mit Induktion erhalten wir $w^{i'} \models \psi_1$ und $w^j \models \theta_1$ für alle $i \leq j < i'$.

  Also gilt $w^i \models \theta_1 \,\mathsf{U}\, \psi_1$.
- Für alle $i' \geq i$ gilt $\psi_1 \notin M_{i'}$ und $\theta_1 \,\mathsf{U}\, \psi_1 \in M_{i'}$.

  Dann würde aber $M_{i'} \notin F_1$ für alle $i' \geq i$ gelten.

  Dann wäre aber der Lauf $M_1 M_2 M_3 \cdots$ nicht akzeptierend.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

7. Fall: $\psi = \psi' \,\mathsf{R}\, \psi'' \in M_i$.

Aus $\psi' \,\mathsf{R}\, \psi'' \in M_i \in \mathcal{H}(\varphi)$ folgt:

- $\psi'' \in M_i$ und
- $\psi' \in M_i$ oder $\mathsf{X}(\psi' \,\mathsf{R}\, \psi'') \in M_i$. d.h. $\psi' \,\mathsf{R}\, \psi'' \in M_{i+1}$.

Durch iterierte Anwendung dieses Arguments erhalten wir einen der beiden folgenden Fälle:

- $\psi'' \in M_{i'}$ für alle $i' \geq i$.
- Es gibt ein $i' \geq i$ mit $\psi' \in M_{i'}$ und $\psi'' \in M_j$ für alle $i \leq j \leq i'$.

Mit Induktion erhalten wir einen der beiden folgenden Fälle:

- $w^{i'} \models \psi''$ für alle $i' \geq i$.
- Es gibt ein $i' \geq i$ mit $w^{i'} \models \psi'$ und $w^j \models \psi''$ für alle $i \leq j \leq i'$.

Also gilt $w^i \models \psi' \,\mathsf{R}\, \psi''$.

Beweis von Satz 4 (LTL $\rightarrow$ VBA)

Wir haben nun gezeigt, dass $L(B_\varphi) = \{w \mid w \models \varphi\}$ gilt.

Außerdem hat B_φ höchstens $2^{2|\varphi|}$ viele Zustände, da es $2^{2|\varphi|}$ viele Teilmengen von $\mathrm{FL}(\varphi)$ gibt.

Die Anzahl der Endzustandsmengen von B_φ ist höchstens $|\varphi|$.

Nach Lemma 5 kann B_φ in einen äquivalenten NBA mit $1 + |\varphi| \cdot 2^{2|\varphi|}$ vielen Zuständen umgewandelt werden.

Dies beendet den Beweis von Satz 4. $\square$

Bemerkungen zu Satz 4

- Der exponentielle Blow-Up ist unvermeidbar.
- Ein in der Praxis relativ effizienties Verfahren zur Umwandlung einer LTL-Formel in einen NBA findet sich in: Gastin, Oddoux. Fast LTL to Büchi automata translation. In Proceedings of CAV'01, Lecture Notes in Computer Science 2102, S. 53–65, Springer 2001.
- Es gibt einen NBA, für den es keine äquivalente LTL-Formel gibt.

LTL-Model-Checking

Wir können nun folgenden Satz beweisen:

Satz 6 (Entscheidbarkeit von LTL-Model-Checking)

Für eine LTL-Formel $\varphi \in \mathrm{LTL}(\Pi)$, einen endlichen Transitionsgraphen $T = (V, E, \Pi, \pi)$ und einen Knoten $v_0 \in V$ kann in Zeit $2^{O(|\varphi|)} \cdot (|V| + |E|)$ entschieden werden, ob $(T, v_0) \models \varphi$ gilt.

Beweis: Sei $\Sigma = 2^{\Pi}$.

Zunächst wandeln wir mit Satz 4 die Formel $\neg\varphi$ in einen NBA $B = (S, \Sigma, \delta, s_0, F)$ mit folgenden Eigenschaften um:

- B hat $2^{O(|\varphi|)}$ viele Zustände
- $L(B) = \{w \in \Sigma^* \mid w \models \neg\varphi\}$

LTL-Model-Checking

Es gilt folgende Äquivalenz:

$$
\begin{aligned}
& (T, v_0) \not\models \varphi \\
\iff & \exists w \in \omega\text{-tr}(T, v_0) : w \not\models \varphi \\
\iff & \exists w \in \omega\text{-tr}(T, v_0) : w \models \neg\varphi \\
\iff & \exists w \in \omega\text{-tr}(T, v_0) : w \in L(B) \\
\iff & L(B) \cap \omega\text{-tr}(T, v_0) \neq \emptyset
\end{aligned}
$$

Wir definieren nun einen gerichteten endlichen Graphen $G = (V', E')$ und eine Teilmenge $U \subseteq V'$ der Knoten wie folgt:

- $V' = S \times V$
- $E' = \{((s, v), (s', v')) \mid (v, v') \in E, (s, \pi(v), s') \in \delta\}$
- $U = F \times V$

LTL-Model-Checking

Es gilt dann folgendes:

- G hat $2^{O(|\varphi|)} \cdot |V|$ viele Knoten und $2^{O(|\varphi|)} \cdot |E|$ viele Kanten.
- $L(B) \cap \omega\text{-tr}(T, v_0) \neq \emptyset$ genau dann, wenn in G ein in (s_0, v_0) beginnender unendlicher Pfad existiert, der unendlich oft einen Knoten aus U besucht.

  Letzteres ist äquivalent zu:

  $$\exists (s, v) \in U : ((s_0, v_0), (s, v)) \in (E')^* \text{ und } ((s, v), (s, v)) \in (E')^+$$

  Informel: Es existiert ein von (s_0, v_0) erreichbarer Knoten, der auf einem Zyklus liegt.

  Man nennt dies auch rekurrente Erreichbarkeit.

Rekurrente Erreichbarkeit

Es genügt also, den folgenden Satz zu beweisen. □

Satz 7

Für einen gerichteten Graphen $G = (V, E)$, eine Knotenmenge $U \subseteq V$ und einem Startknoten $v_0 \in V$ kann in Zeit $O(|V| + |E|)$ überprüft werden, ob ein $v \in U$ mit $(v_0, v) \in E^*$ und $(v, v) \in E^+$ existiert.

Beweis: Wir führen eine Variation von Tiefensuche in G durch.

Rekurrente Erreichbarkeit

Algorithmus Tiefensuche

procedure dfs($G = (V, E), v_0 \in V$)
  var S : stack of nodes; R : set of nodes;
  $R := \{v_0\}$; $S := v_0$
  while $S \neq \varepsilon$ **do**
    v := top(S)
    if $\mathrm{suc}_G(v) \cap R \neq \emptyset$ **then**
      nimm einen beliebigen Knoten $v' \in \mathrm{suc}_G(v) \cap R$ (†)
      push(S, v'); $R := R \cup \{v'\}$
    else
      pop(S);
    endif
  endwhile
endprocedure

Rekurrente Erreichbarkeit

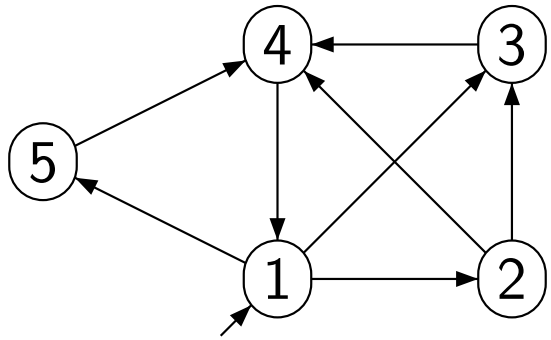
Tiefensuche erzeugt einen Tiefensuchbaum T_G mit der Knotenmenge $V' = \{v \in V \mid (v_0, v) \in E^*\}$ und den Kanten $(v, v') \in E \cap (V' \times V')$, für die (†) gilt.

Anhand dieses Baums T_G kann man die Kanten von G in 4 Klassen einteilen:

- Baumkanten: Die Kanten von T_G.
- Vorwärtskanten: Die Nicht-Baumkanten, die zu einem weiter unten im Baum liegenden Knoten führen.
- Rückwärtskanten: Die Kanten, die von einem Knoten zu einem Vorgänger in T_G führen.
- Querkanten: Alle anderen Kanten (führen zu einem weiter links in T_G liegenden Knoten).

Rekurrente Erreichbarkeit

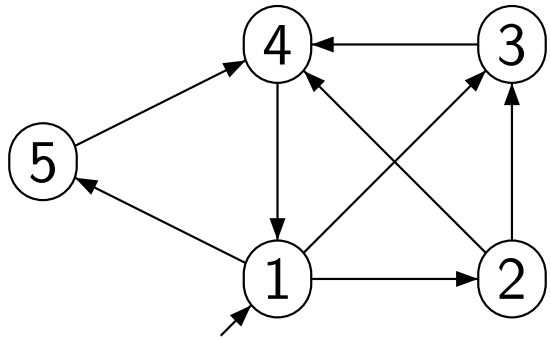
Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.



Dann wird der Tiefensuchbaum wie folgt aufgebaut:

Rekurrente Erreichbarkeit

Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.

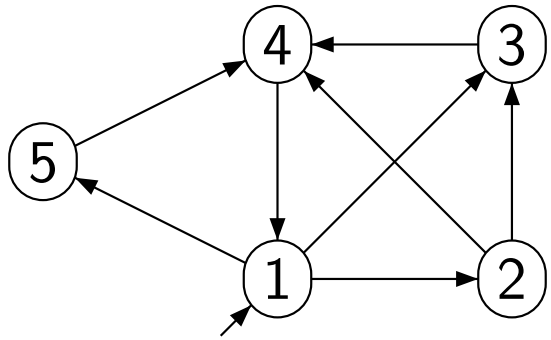


Dann wird der Tiefensuchbaum wie folgt aufgebaut:

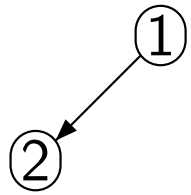
①

Rekurrente Erreichbarkeit

Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.

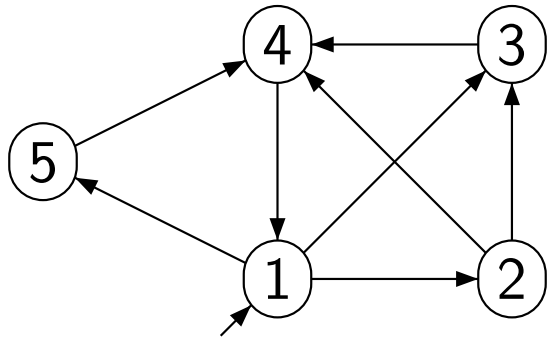


Dann wird der Tiefensuchbaum wie folgt aufgebaut:

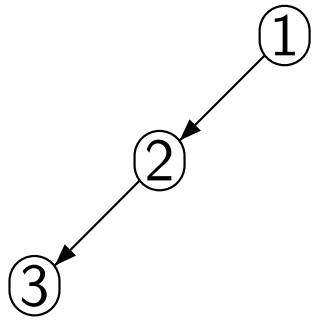


Rekurrente Erreichbarkeit

Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.

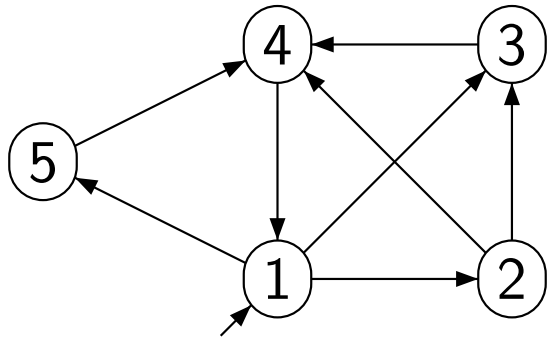


Dann wird der Tiefensuchbaum wie folgt aufgebaut:

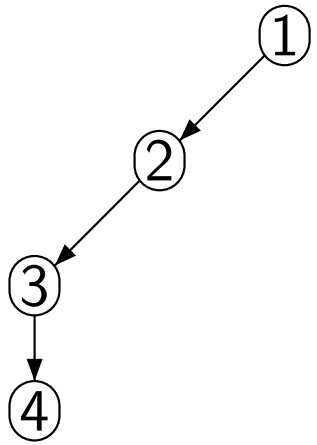


Rekurrente Erreichbarkeit

Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.

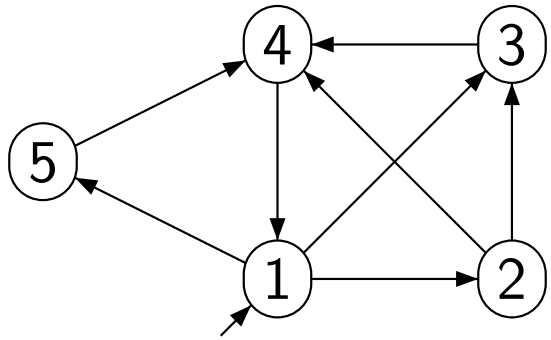


Dann wird der Tiefensuchbaum wie folgt aufgebaut:

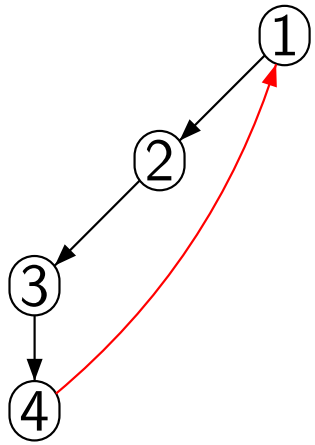


Rekurrente Erreichbarkeit

Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.

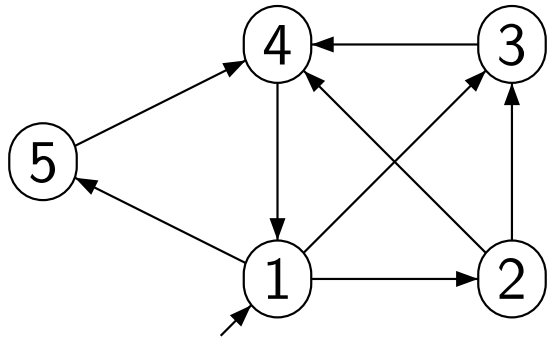


Dann wird der Tiefensuchbaum wie folgt aufgebaut:

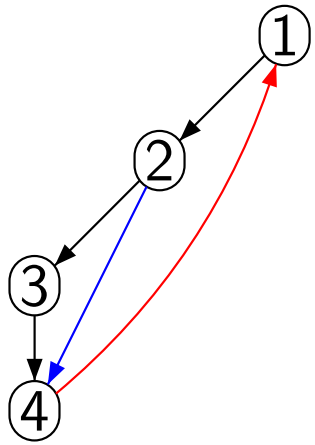


Rekurrente Erreichbarkeit

Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.

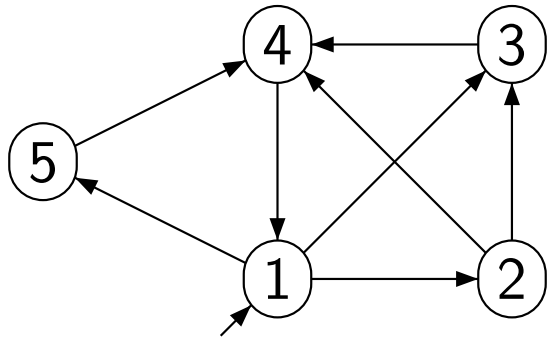


Dann wird der Tiefensuchbaum wie folgt aufgebaut:

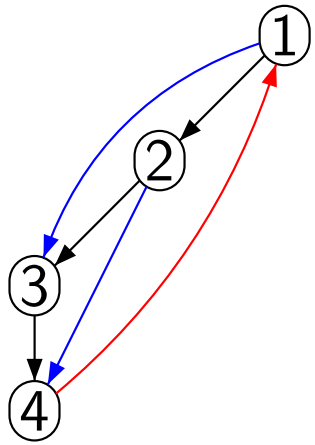


Rekurrente Erreichbarkeit

Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.

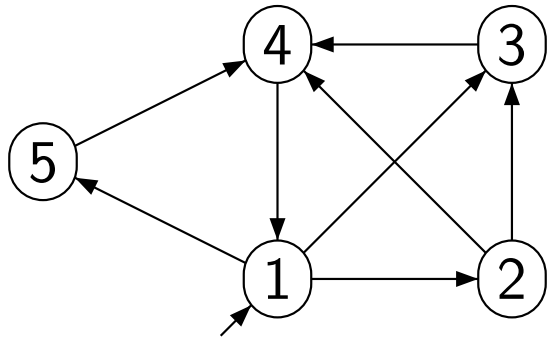


Dann wird der Tiefensuchbaum wie folgt aufgebaut:

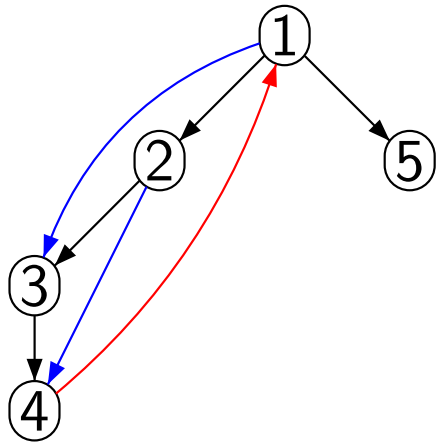


Rekurrente Erreichbarkeit

Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.

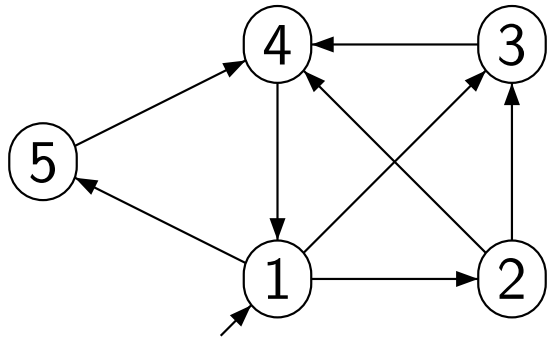


Dann wird der Tiefensuchbaum wie folgt aufgebaut:

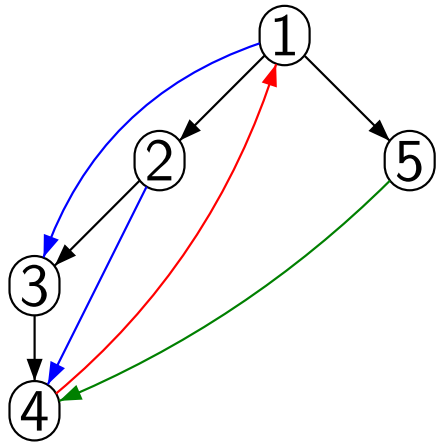


Rekurrente Erreichbarkeit

Beispiel: Sei $G = (V, E)$ der folgende Graph, v_0 der mit dem eingehenden Pfeil markierte Knoten.



Dann wird der Tiefensuchbaum wie folgt aufgebaut:



Rekurrente Erreichbarkeit

Tiefensuche kann verwendet werden, um herauszufinden, ob ein Knoten u auf einem Zyklus liegt.

Dies ist genau dann der Fall, wenn während $\mathrm{dfs}(G, u)$ für $v = \mathrm{top}(S)$ irgendwann $u \in \mathrm{suc}_G(v)$ gilt.

In der Prozedur cycle-search auf der folgenden Folie würde man R' normalerweise als eine mit $\emptyset$ initialisierte lokale Variable definieren.

Später benötigen wir jedoch die Variante, wo R' eine globale Variable ist.

Rekurrente Erreichbarkeit

Algorithmus Suche nach Zyklus

procedure cycle-search($G = (V, E), u \in V$)
  var S' : stack of nodes;
  $R' := R' \cup \{u\}$; $S' := u$;
  while $S' \neq \varepsilon$ **do**
    v := top(S')
    if $u \in \mathrm{suc}_G(v)$ **then return** (true) **endif**
    if $\mathrm{suc}_G(v) \cap R' \neq \emptyset$ **then**
      nimm einen beliebigen Knoten $v' \in \mathrm{suc}_G(v) \cap R'$
      push(S', v'); $R' := R' \cup \{v'\}$
    else
      pop(S')
    endif
  endwhile
  return (false)
endprocedure

Rekurrente Erreichbarkeit

Um zu testen, ob ein Knoten $v \in U$ mit $(v_0, v) \in E^*$ und $(v, v) \in E^+$ existiert, könnten wir den Algorithmus cycle-search von der letzten Folie in eine Tiefensuche schachteln:

- Starte dfs(G, v_0)
- Jedes mal, wenn der in Zeile (†) ausgewählte Knoten v' zu U gehört, starte cycle-search(G, v'), wobei R' mit $\emptyset$ initialisiert wird.

Dies würde jedoch zu einer quadratischen Laufzeit führen.

Um eine lineare Laufzeit zu erhalten, definieren wir R' als eine globale Variable, die nur zu Beginn mit $\emptyset$ initialisiert wird.

Rekurrente Erreichbarkeit

Algorithmus verschachtelte Tiefensuche

procedure ndfs($G = (V, E), v_0 \in V,\ U \subseteq V$)
  var S : stack of nodes; R, R' : set of nodes; cycle-found : boolean
  $R := \{v_0\}$; $R' := \emptyset$; $S := v_0$; $S' := \varepsilon$; cycle-found := false
  while $S \neq \varepsilon$ and not cycle-found **do**
    v := top(S)
    if $\mathrm{suc}_G(v) \cap R \neq \emptyset$ **then**
      nimm einen beliebigen Knoten $v' \in \mathrm{suc}_G(v) \cap R$
      push(S, v'); $R := R \cup \{v'\}$
    else (†)
      pop(S);
      if $v \in U$ and $v \notin R'$ **then** cycle-found := cycle-search(G, v)
    endif
  endwhile
  return (cycle-found)
endprocedure

Rekurrente Erreichbarkeit

Wir zeigen die folgende Äquivalenz:

$$\text{ndfs}(G, v_0) \text{ gibt true zurück } \iff \exists v \in U : (v_0, v) \in E^* \text{ und } (v, v) \in E^+$$

Die Richtung $\Longrightarrow$ is klar.

Für den Beweis von $\Longleftarrow$ genügt es, dass folgende Lemma zu zeigen:

Lemma 8

Sei v ein Knoten, für den während $\text{ndfs}(G, v_0, U)$ der else-Zweig in (†) betreten wird. Dann gibt es zu diesem Zeitpunkt keinen Zyklus $(u_1, u_2, \ldots, u_k)$ mit folgenden Eigenschaften:

- $v \in \{u_1, u_2, \ldots, u_k\}$
- $R' \cap \{u_1, u_2, \ldots, u_k\} \neq \emptyset$

Rekurrente Erreichbarkeit

Beweis des Lemmas:

Sei v ein Knoten, für den während ndfs(G, v_0, U) der else-Zweig in (†) betreten wird.

Angenommen es gibt zu diesem Zeitpunkt (wir nennen in t im weiteren) einen Zyklus $(u_1, u_2, \ldots, u_k)$ mit folgenden Eigenschaften:

- $v \in \{u_1, u_2, \ldots, u_k\}$
- $R'(t) \cap \{u_1, u_2, \ldots, u_k\} \neq \emptyset$ $\quad (R'(t) = R'$ zum Zeitpunkt $t)$

Sei $u \in R'(t) \cap \{u_1, u_2, \ldots, u_k\}$.

O.B.d.A. können wir folgendes annehmen: Für alle Knoten v' für die der else-Zweig in (†) zu einen Zeitpunkt $t' < t$ betreten wird, existiert kein Zyklus $(u'_1, u'_2, \ldots, u'_k)$ mit folgenden Eigenschaften:

- $v' \in \{u'_1, u'_2, \ldots, u'_k\}$
- $R'(t') \cap \{u'_1, u'_2, \ldots, u'_k\} \neq \emptyset$

Rekurrente Erreichbarkeit

Da $u \in R'(t)$ gilt, muss es einen Knoten $v' \neq v$ geben, so dass cycle-search(G, v') zu einem Zeitpunkt $t' < t$ gestartet wird und u während cycle-search(G, v') zu R' hinzugefügt wird.

Also gilt $v' \in U$, $v' \notin R'(t')$ und $(v', u) \in E^*$.

Da u und v auf einem gemeinsamen Zyklus liegen, gilt auch $(v', v) \in E^*$.

Wir betrachten nun folgende zwei Fälle:

Fall 1: v' wird in ndfs(G, v_0, U) vor v auf den Stack S gelegt.

Wegen $(v', v) \in E^*$, wird v vor v' vom Stack S genommen. Dies widerspricht aber $t' < t$.

Rekurrente Erreichbarkeit

Fall 2: v wird in ndfs(G, v_0, U) vor v' auf den Stack S gelegt.

Wegen $t' < t$ ist v noch auf dem Stack, wenn v' vom Stack entfernt wird.

Also gilt $(v, v') \in E^*$.

Da auch $(v', v) \in E^*$ und $v \neq v'$ gilt, liegen v und v' auf einem gemeinsamen Zyklus $(u'_1, u'_2, \ldots, u'_k)$.

Wegen $t' < t$ gilt weiterhin $R'(t') \cap \{u'_1, u'_2, \ldots, u'_k\} = \emptyset$.

Also wird dieser (oder ein anderer) Zyklus während cycle-search(G, v') gefunden.

Also liefert cycle-search(G, v') true zurück, und der Algorithmus terminiert vor Zeitpunkt t. Widerspruch! □

Rekurrente Erreichbarkeit

Für den Beweis von Satz 7 genügt es nun zu zeigen, dass ndfs(G, v_0, U) in Zeit $O(|V| + |E|)$ läuft.

Dazu genügt es zu beobachten, dass

- während ndfs(G, v_0, U) jede Kante $(v, v') \in E$ höchstens einmal exploriert wird und ebenso
- während aller Aufrufe von cycle-search jede Kante $(v, v') \in E$ höchstens einmal exploriert wird.

Der zweite Punkt gilt, da die globale Variable R' nur wachsen kann. □

Komplexität von LTL-Model-Checking

- Die Zeitschranke $2^{O(|\varphi|)} \cdot (|V| + |E|)$ in Satz 6 ist exponentiell in der Eingabelänge $|\varphi| + |V| + |E|$.

  Dies lässt sich wohl auch nicht vermeiden: LTL-Model-Checking ist ein PSPACE-vollständiges Problem.

- Andererseits: Die Zeitschranke $2^{O(|\varphi|)} \cdot (|V| + |E|)$ ist nur exponentiell in der Formelgröße $|\varphi|$.

  Ist also die Formel nicht allzu groß, so ist die Laufzeit $2^{O(|\varphi|)} \cdot (|V| + |E|)$ noch vertretbar.

  Dies hat praktische Relevanz:

  - In der Praxis ist zwar der Transitionsgraph häufig sehr groß (2^n Zustände für Schaltkreise mit n Bits), aber
  - die LTL-Formel ist häufig relativ klein, das sie von einem Menschen aufgeschrieben wurde.

SPIN

Einer der bekanntesten open-source LTL-Model-Checker ist SPIN (Simple PROMELA Interpreter).

- Wurde Anfang der 1980'er Jahre bei Bell Labs entwickelt.
- Das zu verifizierende System wird in der Modellbeschreibungssprache PROMELA (Process Meta Language) beschrieben.
- Die zu verifizierende Systemeigenschaft wird in LTL spezifiziert und in einen NBA transformiert.
- Viele Optimierungen
- Hauptanwendung: multi-threaded software
- Wurde z.B. zur Verifikation der Kontrollalgorithmen der Überflutungsbarrieren bei Rotterdam verwendet.
- 2002: ACM System Software Award
- Mehr Infos: `http://spinroot.com/spin/whatispin.html`

Erfüllbarkeit für LTL

Satz 4 hat noch weitere wichtige Konsequenzen:

Satz 9 (Entscheidbarkeit der Erfüllbarkeit für LTL)

Das folgende Problem ist entscheidbar:

INPUT: Eine LTL-Formel $\varphi \in \mathrm{LTL}(\Pi)$.

FRAGE: Existiert ein ω-Wort $w \in (2^\Pi)^\omega$ mit $w \models \varphi$ (φ ist erfüllbar)?

Beweis: Konstruiere aus φ einen NBA $B = (S, 2^\Pi, \delta, s_0, F)$ mit

$$L(B) = \{w \in (2^\Pi)^\omega \mid w \models \varphi\}.$$

Teste, dann ob ein Endzustand $s \in F$ von s_0 erreichbar ist, der auf einem Zyklus liegt. □

Äquivalenz für LTL

Satz 10 (Entscheidbarkeit der Äquivalenz für LTL)

Für zwei LTL-Formeln φ und ψ kann entschieden werden, ob $\varphi \equiv \psi$ gilt.

Beweis: Es gilt $\varphi \equiv \psi$ genau dann, wenn $(\varphi \wedge \neg\psi) \vee (\neg\varphi \wedge \psi)$ nicht erfüllbar ist. □

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Zur Erinnerung: Eine Sprache $L \subseteq \Sigma^\omega$ ist ω-regulär, falls ein NBA mit $L(B) = L$ existiert.

Abschluss der ω-regulären Sprachen unter Vereinigung und Schnitt ist relativ einfach zu zeigen.

Satz 11

Aus NBAs B_1 und B_2 kann ein NBA B mit $L(B) = L(B_1) \cup L(B_2)$ konstruiert werden.

Beweis: Sei $B_i = (S_i, \Sigma, \delta_i, s_i, F_i)$ mit $S_1 \cap S_2 = \emptyset$.

Sei $s_0 \notin S_1 \cup S_2$ und $B = (S_1 \cup S_2 \cup \{s_0\}, \Sigma, \delta, s_0, F_1 \cup F_2)$, wobei

$$\delta = \delta_1 \cup \delta_2 \cup \{(s_0, a, s) \mid (s_1, a, s) \in \delta_1 \text{ oder } (s_2, a, s) \in \delta_2\}.$$

Dann gilt in der Tat $L(B) = L(B_1) \cup L(B_2)$. □

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Satz 12

Aus NBAs B_1 und B_2 kann ein NBA B mit $L(B) = L(B_1) \cap L(B_2)$ konstruiert werden.

Beweis: Sei $B_i = (S_i, \Sigma, \delta_i, s_i, F_i)$.

Konstruiere den VBA $B = (S_1 \times S_2, \Sigma, \delta, \{(s_1, s_2)\}, F_1 \times S_2, S_1 \times F_2)$ mit

$$\delta = \{((p_1, p_2), a, (q_1, q_2)) \mid (p_1, a, q_1) \in \delta_1, (p_2, a, q_2) \in \delta_2\}.$$

Dann gilt $L(B) = L(B_1) \cap L(B_2)$.

Wegen Lemma 5 kann B in einen äquivalenten NBA umgewandelt werden. □

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Wir wollen nun zeigen, dass aus einem NBA B über dem Alphabet Σ ein NBA B' mit $L(B') = \Sigma^\omega \setminus L(B)$ konstruiert werden kann.

Anders gesagt: Die regulären ω-Sprachen sind unter Komplement abgeschlossen.

Erinnerung an GTI: Auch die reguläre Sprachen (von endlichen Wörtern) sind unter Komplement abgeschlossen.

Zum Beweis haben wir aus einem nichtdeterministischen Automaten einen äquivalenten deterministischen Automaten konstruiert (Potenzmengenkonstruktion).

Deterministische Büchiautomaten (DBA) definiert man wie bei normalen endlichen Automaten (δ ist eine Funktion $\delta : S \times \Sigma \to S$).

Aber: Es gibt eine reguläre ω-Sprache L, die nicht durch einen DBA akzeptiert werden kann: $L = \{a, b\}^* a^\omega$.

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Sei $B = (S, \Sigma, \delta, s_0, F)$ ein NBA.

Für $s, t \in S$ und $w \in \Sigma^+$ schreiben wir

- $s \xrightarrow{w} t$, falls ein mit w beschrifteter Pfad in B von s nach t existiert, und
- $s \xrightarrow{w}_F t$, falls ein mit w beschrifteter Pfad in B von s nach t existiert, welcher einen Zustand aus F besucht.

Wir definieren eine Äquivalenzrelation $\equiv_B$ auf der Menge Σ^+ wie folgt:
Für $u, v \in \Sigma^+$ gilt $u \equiv_B v$ genau dann, wenn gilt:

$$\forall s, t \in S : (s \xrightarrow{u} t) \Leftrightarrow (s \xrightarrow{v} t) \text{ und}$$

$$\forall s, t \in S : (s \xrightarrow{u}_F t) \Leftrightarrow (s \xrightarrow{v}_F t)$$

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Lemma 13

Die Relation $\equiv_B$ ist eine Äquivalenzrelation, welche nur endlich viele Äquivalenzklassen hat, und jede dieser Äquivalenzklassen ist eine reguläre Sprache.
Aus B kann eine Liste von endlichen Automaten $A_1, \ldots, A_k$ berechnet werden, so dass $L(A_1), \ldots, L(A_k)$ eine Liste aller Äquivalenzklassen von $\equiv_B$ ist.

Beweis: einfache Übung.

Lemma 14

Seien $U, V \subseteq \Sigma^+$ Äquivalenzklassen von $\equiv_B$. Falls $UV^\omega \cap L(B) \neq \emptyset$ gilt, so gilt bereits $UV^\omega \subseteq L(B)$.

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Beweis: Sei $w \in UV^\omega \cap L(B)$.

Wegen $w \in UV^\omega$ gibt es $u \in U$ und $v_1, v_2, v_3, \ldots \in V$ mit

$$w = uv_1v_2v_3\cdots .$$

Wegen $w \in L(B)$ gibt es Zustände $s_1, s_2, s_3, \ldots \in S$ mit

$$s_0 \xrightarrow{u} s_1, \forall i \geq 1 : s_i \xrightarrow{v_i} s_{i+1}, \text{ und } s_i \xrightarrow{v_i}_F s_{i+1} \text{ für } \infty \text{ viele } i \geq 1.$$

Sei nun $w' \in UV^\omega$ beliebig.

Dann können wir w' schreiben als

$$w' = u'v_1'v_2'v_3'\cdots$$

mit $u \equiv_B u'$ und $v_i \equiv_B v_i'$ für alle $i \geq 1$.

Also gilt

$$s_0 \xrightarrow{u'} s_1, \forall i \geq 1 : s_i \xrightarrow{v_i'} s_{i+1}, \text{ und } s_i \xrightarrow{v_i'}_F s_{i+1} \text{ für } \infty \text{ viele } i \geq 1.$$

Dies impliziert $w' \in L(B)$. □

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Lemma 15

Für jedes Wort $w \in \Sigma^\omega$ gibt es Äquivalenzklassen $U, V \subseteq \Sigma^+$ von $\equiv_B$ mit $w \in UV^\omega$.

Für den Beweis von Lemma 15 verwenden wir einen berühmten Satz aus der Kombinatorik.

Erinnerung: $\binom{\mathbb{N}}{2}$ ist die Menge aller 2-elementigen Teilmengen von $\mathbb{N}$.

Eine Färbung von $\binom{\mathbb{N}}{2}$ ist eine Abbildung $\chi : \binom{\mathbb{N}}{2} \to C$, wobei C eine endliche Menge (von Farben) ist.

Satz 16 (Satz von Ramsey, 1930)

Sei $\chi : \binom{\mathbb{N}}{2} \to C$ eine Färbung. Dann existiert eine unendliche Teilmenge $M \subseteq \mathbb{N}$, so dass für alle $A, B \in \binom{M}{2}$ gilt: $\chi(A) = \chi(B)$.

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Beweis von Lemma 15: Sei $w = a_1 a_2 a_3 \cdots$ mit $a_1, a_2, a_3, \ldots \in \Sigma$.

Für $i < j$ sei $w[i,j] = a_i a_{i+1} \cdots a_{j-1} \in \Sigma^+$.

Wir definieren nun eine Färbung χ der Menge $\binom{\mathbb{N}}{2}$ wie folgt: Für $i < j$ sei

$$\chi(\{i,j\}) = \text{ die Äquivalenzklasse } C \text{ von } \equiv_B \text{ mit } w[i,j] \in C.$$

Dies ist eine Färbung mit endlich vielen Farben.

Aus Ramseys Theorem folgt die Existenz einer unendlichen Menge $I \subseteq \mathbb{N}$, so dass für alle $i,j,k,\ell \in I$ mit $i < j$ und $k < \ell$ gilt: $\chi(\{i,j\}) = \chi(\{k,\ell\})$.

Sei $I = \{i_1, i_2, i_3, \ldots\}$ mit $1 < i_1 < i_2 < i_3 < \cdots$.

Es gibt also eine Äquivalenzklasse V von $\equiv_B$ mit

$$\forall k \geq 1 : w[i_k, i_{k+1}] = a_{i_k} \cdots a_{i_{k+1}-1} \in V.$$

Sei ausserdem U die Äquivalenzklasse von $\equiv_B$ mit $w[1, i_1] = a_1 a_2 \cdots a_{i_1-1} \in U$.

Dann gilt $w \in UV^\omega$. □

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Satz 17 (Büchi 1962)

Aus einem NBA B über Σ kann ein NBA B' mit $L(B') = \Sigma^\omega \setminus L(B)$ konstruiert werden.

Beweis: Wir zeigen zunächst die Existenz von B' und zeigen dann, wie man B' algorithmisch konstruieren kann.

Seien $(U_1, V_1), \ldots, (U_k, V_k)$ alle Paare, wobei

- $U_1, V_1, \ldots, U_k, V_k$ Äquivalenzklassen von $\equiv_B$ sind, und
- $U_i V_i^\omega \cap L(B) = \emptyset$ für alle $1 \leq i \leq k$.

Wegen Lemma 13 gibt es tatsächlich nur endlich viele solcher Paare.

Sei $L' = \bigcup_{i=1}^{k} U_i V_i^\omega$.

Wegen Lemma 3 is L' ω-regulär.

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Behauptung 1: $L' = \Sigma^\omega \setminus L(B)$.

Da $U_i V_i^\omega \cap L(B) = \emptyset$ für alle $1 \leq i \leq k$ gilt, folgt $L' \subseteq \Sigma^\omega \setminus L(B)$.

Sei umgekehrt $w \in \Sigma^\omega \setminus L(B)$.

Wegen Lemma 15 gibt es Äquivalenzklassen U und V von $\equiv_B$ mit $w \in UV^\omega$.

Aus $UV^\omega \cap L(B) \neq \emptyset$ würde wegen Lemma 14 $UV^\omega \subseteq L(B)$ folgen, was jedoch $w \in UV^\omega \cap (\Sigma^\omega \setminus L(B))$ widerspricht.

Also gilt $UV^\omega \cap L(B) = \emptyset$ und es existiert ein $1 \leq i \leq k$ mit $U = U_i$ und $V = V_i$.

Also gilt $w \in U_i V_i^\omega \subseteq L'$.

Einschub: Abschlusseigenschaften regulärer ω-Sprachen

Behauptung 2: Ein NBA B' mit $L(B') = L'$ kann effektiv konstruiert werden.

Nach Lemma 13 kann eine Liste $A_1, \ldots, A_k$ von endlichen Automaten für die Äquivalenzklassen von $\equiv_B$ berechnet werden.

Es genügt somit für gegebene $1 \leq i, j \leq k$ effektiv zu überprüfen, ob $L(A_i)L(A_j)^\omega \cap L(B) \neq \emptyset$ gilt.

Wegen Lemma 14 genügt es für beliebige Wörter $u \in L(A_i)$ und $v \in L(A_j)$ zu überprüfen, ob $uv^\omega \in L(B)$ gilt.

Es gilt $uv^\omega \in L(B)$ genau dann, wenn ein Zustand $s \in S$ und $m, n \leq |S|$ mit $s_0 \xrightarrow{uv^m} s$ und $s \xrightarrow{v^n}_F s$ existieren. □

Computation Tree Logic (CTL)

LTL kann nur Eigenschaften von (unendlichen) Pfaden in Transitionsgraphen ausdrücken.

Aber viele wichtige Systemeigenschaften gehen darüber hinaus und machen Aussagen über das Verzweigungsverhalten.

Beispiel: Sei $T = (V, E, \Pi, \pi)$ ein Transitionsgraph und $v_0 \in V$.

Folgende Eigenschaft kann nicht in LTL ausgedrückt werden:

Jeder von v_0 aus erreichbare Knoten v hat die Eigenschaft, dass man von v aus einen Knoten u mit $p \in \pi(u)$ erreichen kann.

Formal: $\forall v \in V : (v_0, v) \in E^* \rightarrow \exists u \in V : (v, u) \in E^* \wedge p \in \pi(u)$

Solche Eigenschaften lassen sich in CTL ausdrücken.

Syntax von CTL

In CTL gibt es zwei Typen von Formeln: Zustandsformeln und Pfadformeln, die sich wechselseitig rekursive definieren.

Sei Π wieder eine endliche Menge von Propositionen.

Definition (Syntax von CTL)

Die Mengen $\mathrm{CTL}_p(\Pi)$ (CTL-Pfadformeln) und $\mathrm{CTL}_s(\Pi)$ (CTL-Zustandsformeln) sind die bezüglich Inklusion kleinsten Mengen von Formeln mit:

- $\Pi \subseteq \mathrm{CTL}_s(\Pi)$
- Wenn $\varphi, \psi \in \mathrm{CTL}_s(\Pi)$, dann $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi \in \mathrm{CTL}_s(\Pi)$.
- Wenn $\varphi, \psi \in \mathrm{CTL}_s(\Pi)$, dann $\mathsf{X}\varphi, \varphi\,\mathsf{U}\,\psi \in \mathrm{CTL}_p(\Pi)$.
- Wenn $\varphi \in \mathrm{CTL}_p(\Pi)$, dann $\forall\varphi, \exists\varphi \in \mathrm{CTL}_s(\Pi)$.

Semantik von CTL

Sei $T = (V, E, \Pi, \pi)$ ein Transitionsgraph.

CTL-Zustandsformeln (CTL-Pfadformeln) werten wir in Knoten (Pfaden) von T aus.

Definition (Semantik von CTL-Zustandsformeln)

Sei $v \in V$ ein Knoten von T.

- $(T, v) \models p$ falls $p \in \pi(v)$.
- $(T, v) \models \neg\varphi$, falls $(T, v) \models \varphi$ nicht gilt.
- $(T, v) \models \varphi \wedge \psi$, falls $(T, v) \models \varphi$ und $(T, v) \models \psi$ gilt.
- $(T, v) \models \varphi \vee \psi$, falls $(T, v) \models \varphi$ oder $(T, v) \models \psi$ gilt.
- $(T, v) \models \forall\varphi$, falls $(T, p) \models \varphi$ für alle Pfade $p \in \mathrm{path}(T, v)$ gilt.
- $(T, v) \models \exists\varphi$, falls ein Pfad $p \in \mathrm{path}(T, v)$ mit $(T, p) \models \varphi$ existiert.

Semantik von CTL

Definition (Semantik von CTL-Pfadformeln)

Sei $p = (v_1 v_2 v_3 \cdots) \in \text{path}(T, v_1)$ ein Pfad in T.

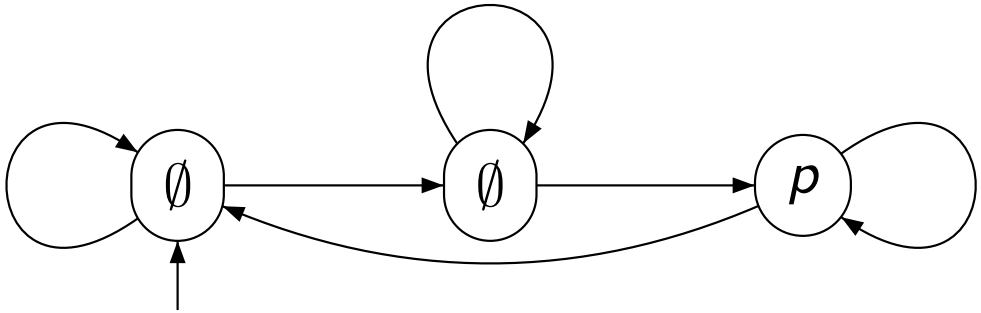
- $(T, p) \models X\varphi$, falls $(T, v_2) \models \varphi$
- $(T, p) \models \varphi \,\mathsf{U}\, \psi$, falls ein $i \geq 1$ existiert mit:
  - $(T, v_i) \models \psi$ und
  - $(T, v_j) \models \varphi$ für alle $1 \leq j < i$.

Wir verwenden ähnliche Abkürzungen wie für LTL:

- $\forall \mathsf{F}\varphi = \forall(\text{true}\,\mathsf{U}\,\varphi)$: Auf allen Pfaden gilt irgendwann φ.
- $\exists \mathsf{F}\varphi = \exists(\text{true}\,\mathsf{U}\,\varphi)$: Es gibt einen Pfad, auf dem φ irgendwann gilt.
- $\forall \mathsf{G}\varphi = \neg(\exists \mathsf{F} \neg\varphi)$: Auf allen Pfaden gilt φ immer.
- $\exists \mathsf{G}\varphi = \neg(\forall \mathsf{F} \neg\varphi)$: Es gibt einen Pfad auf dem φ immer gilt.

CTL: Beispiel 1

Betrachte den folgenden Transitionsgraphen T und sei v_0 der mit dem eingehenden Pfeil markierte Knoten.

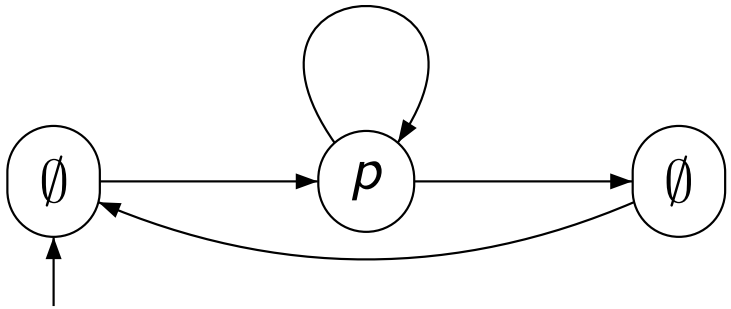


Dann gilt:

- $(T, v_0) \models \exists \mathsf{F} p$
- $(T, v_0) \not\models \forall \mathsf{F} p$
- $(T, v_0) \not\models \exists \mathsf{G} p$
- $(T, v_0) \not\models \forall \mathsf{G} p$
- $(T, v_0) \not\models \exists \mathsf{F} \forall \mathsf{G} p$

CTL: Beispiel 2

Betrachte den folgenden Transitionsgraphen T und sei v_0 der mit dem eingehenden Pfeil markierte Knoten.

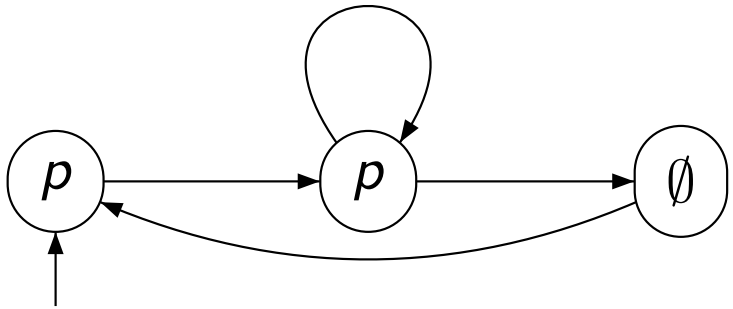


Dann gilt:

- $(T, v_0) \models \exists \mathsf{F} p$
- $(T, v_0) \models \forall \mathsf{F} p$
- $(T, v_0) \not\models \exists \mathsf{G} p$
- $(T, v_0) \not\models \forall \mathsf{G} p$
- $(T, v_0) \not\models \exists \mathsf{F} \forall \mathsf{G} p$

CTL: Beispiel 3

Betrachte den folgenden Transitionsgraphen T und sei v_0 der mit dem eingehenden Pfeil markierte Knoten.

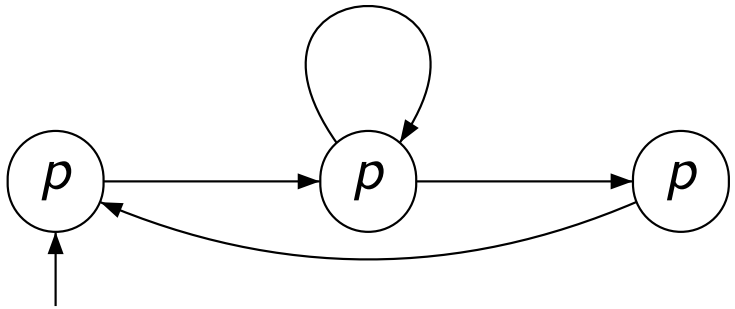


Dann gilt:

- $(T, v_0) \models \exists \mathsf{F} p$
- $(T, v_0) \models \forall \mathsf{F} p$
- $(T, v_0) \models \exists \mathsf{G} p$
- $(T, v_0) \not\models \forall \mathsf{G} p$
- $(T, v_0) \not\models \exists \mathsf{F} \forall \mathsf{G} p$

CTL: Beispiel 4

Betrachte den folgenden Transitionsgraphen T und sei v_0 der mit dem eingehenden Pfeil markierte Knoten.

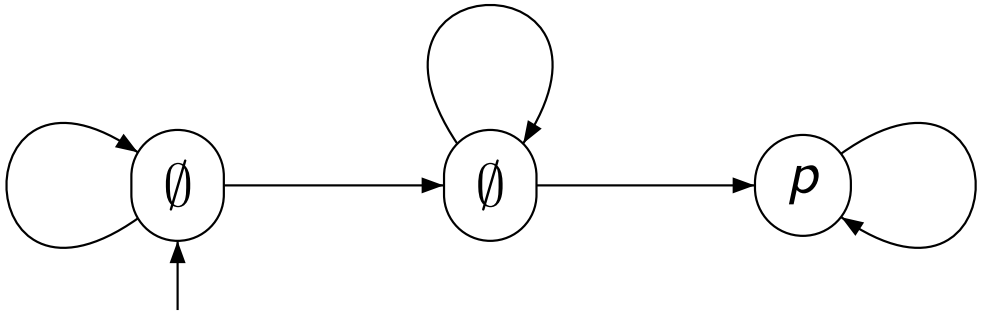


Dann gilt:

- $(T, v_0) \models \exists \mathsf{F} p$
- $(T, v_0) \models \forall \mathsf{F} p$
- $(T, v_0) \models \exists \mathsf{G} p$
- $(T, v_0) \models \forall \mathsf{G} p$
- $(T, v_0) \models \exists \mathsf{F} \forall \mathsf{G} p$

CTL: Beispiel 5

Betrachte den folgenden Transitionsgraphen T und sei v_0 der mit dem eingehenden Pfeil markierte Knoten.



Dann gilt:

- $(T, v_0) \models \exists \mathsf{F} p$
- $(T, v_0) \not\models \forall \mathsf{F} p$
- $(T, v_0) \not\models \exists \mathsf{G} p$
- $(T, v_0) \not\models \forall \mathsf{G} p$
- $(T, v_0) \models \exists \mathsf{F} \forall \mathsf{G} p$

Alternative Syntax für CTL

Beachte: Nach einem Pfadquantor ($\forall$ oder $\exists$) muss stets ein temporaler Operator (X oder U) folgen.

Z.B. ist $\exists((p \,\mathsf{U}\, q) \wedge (r \,\mathsf{U}\, s))$ kein gültige CTL-Formel.

Die Syntax von CTL lässt sich daher auch nur über Zustandsformeln definieren:

- $\Pi \subseteq \mathsf{CTL}_s(\Pi)$
- Wenn $\varphi \in \mathsf{CTL}_s(\Pi)$, dann $\neg\varphi \in \mathsf{CTL}_s(\Pi)$.
- Wenn $\varphi, \psi \in \mathsf{CTL}_s(\Pi)$, dann $\varphi \wedge \psi, \varphi \vee \psi \in \mathsf{CTL}_s(\Pi)$.
- Wenn $\varphi \in \mathsf{CTL}_s(\Pi)$, dann $\forall(\mathsf{X}\varphi), \exists(\mathsf{X}\varphi) \in \mathsf{CTL}_s(\Pi)$.
- Wenn $\varphi, \psi \in \mathsf{CTL}_s(\Pi)$, dann $\forall(\varphi \,\mathsf{U}\, \psi), \exists(\varphi \,\mathsf{U}\, \psi) \in \mathsf{CTL}_s(\Pi)$.

Äquivalenz von CTL-Formeln

Definition (Äquivalenz von CTL-Formeln)

Zwei CTL-Zustandsformeln $\varphi, \psi \in \mathrm{CTL}_s(\Pi)$ sind äquivalent, kurz $\varphi \equiv \psi$, falls für alle Transitionsgraphen $T = (V, E, \Pi, \pi)$ und alle $v \in V$ gilt:

$$(T, v) \models \varphi \iff (T, v) \models \psi.$$

Einige wichtige CTL-Äquivalenzen:

- $\forall \mathsf{G}\varphi \equiv \varphi \wedge \forall \mathsf{X} \forall \mathsf{G}\varphi$
- $\forall \mathsf{F}\varphi \equiv \varphi \vee \forall \mathsf{X} \forall \mathsf{F}\varphi$
- $\exists \mathsf{G}\varphi \equiv \varphi \wedge \exists \mathsf{X} \exists \mathsf{G}\varphi$
- $\exists \mathsf{F}\varphi \equiv \varphi \vee \exists \mathsf{X} \exists \mathsf{F}\varphi$
- $\forall(\varphi \,\mathsf{U}\, \psi) \equiv \psi \vee (\varphi \wedge \forall \mathsf{X} \forall(\varphi \,\mathsf{U}\, \psi))$
- $\exists(\varphi \,\mathsf{U}\, \psi) \equiv \psi \vee (\varphi \wedge \exists \mathsf{X} \exists(\varphi \,\mathsf{U}\, \psi))$

Äquivalenz von CTL-Formeln

- $\exists \mathsf{F}(\varphi \vee \psi) \equiv (\exists \mathsf{F}\varphi) \vee (\exists \mathsf{F}\psi)$
- $\forall \mathsf{G}(\varphi \wedge \psi) \equiv (\forall \mathsf{G}\varphi) \wedge (\forall \mathsf{G}\psi)$
- $\forall \mathsf{X}\varphi \equiv \neg\exists \mathsf{X}\neg\varphi$
- $\forall \mathsf{G}\varphi \equiv \neg\exists \mathsf{F}\neg\varphi$
- $\forall \mathsf{F}\varphi \equiv \neg\exists \mathsf{G}\neg\varphi$
- $\forall(\varphi \,\mathsf{U}\, \psi) \equiv (\neg\exists \mathsf{G}\neg\psi) \wedge \neg\exists(\neg\psi \,\mathsf{U}\, (\neg\varphi \wedge \neg\psi))$

Die letzte Äquivalenz sollten wir beweisen.

Unter Benutzung des release-Operators aus LTL können wir schreiben:

$$\forall(\varphi \,\mathsf{U}\, \psi) \equiv \neg\exists(\neg\varphi \,\mathsf{R}\, \neg\psi).$$

Weiter haben wir für den release-Operators folgende LTL-Äquivalenz gezeigt:

$$\varphi \,\mathsf{R}\, \psi \equiv \mathsf{G}\psi \vee (\psi \,\mathsf{U}\, (\varphi \wedge \psi)).$$

Äquivalenz von CTL-Formeln

Aber Vorsicht: $\neg\exists((\mathsf{G}\neg\psi) \vee (\neg\psi\,\mathsf{U}\,(\neg\varphi \wedge \neg\psi)))$ ist keine CTL-Formel.

Folgende beiden Aussagen sind äquivalent:

- Es existiert ein Pfad mit der Eigenschaft $A \vee B$.
- Es existiert ein Pfad mit der Eigenschaft A oder es existiert ein Pfad mit der Eigenschaft B.

Damit folgt:

$$\begin{aligned}
\forall(\varphi\,\mathsf{U}\,\psi) &\equiv \neg\exists(\neg\varphi\,\mathsf{R}\,\neg\psi) \\
&\equiv \neg((\exists\mathsf{G}\neg\psi) \vee \exists(\neg\psi\,\mathsf{U}\,(\neg\varphi \wedge \neg\psi))) \\
&\equiv (\neg\exists\mathsf{G}\neg\psi) \wedge \neg\exists(\neg\psi\,\mathsf{U}\,(\neg\varphi \wedge \neg\psi))
\end{aligned}$$

Existentielle CTL-Formeln

Für CTL-Model-Checking ist es hilfreich, nur CTL-Formeln ohne den universellen Pfad-Quantor $\forall$ zu betrachten.

Erlauben wir Formeln vom Typ $\exists \mathsf{G}\varphi$, so können wir in der Tat den universellen Pfad-Quantor eliminieren.

Definition (existentielle CTL-Formeln)

Die Menge der $\mathrm{CTL}_\exists(\Pi)$ aller existentiellen CTL-Zustandsformeln ist die kleinste Menge mit:

- $\Pi \subseteq \mathrm{CTL}_\exists(\Pi)$
- Wenn $\varphi \in \mathrm{CTL}_\exists(\Pi)$, dann $\neg\varphi \in \mathrm{CTL}_\exists(\Pi)$.
- Wenn $\varphi, \psi \in \mathrm{CTL}_\exists(\Pi)$, dann $\varphi \wedge \psi, \varphi \vee \psi \in \mathrm{CTL}_\exists(\Pi)$.
- Wenn $\varphi \in \mathrm{CTL}_\exists(\Pi)$, dann $\exists(\mathsf{X}\varphi), \exists(\mathsf{G}\varphi) \in \mathrm{CTL}_\exists(\Pi)$.
- Wenn $\varphi, \psi \in \mathrm{CTL}_\exists(\Pi)$, dann $\exists(\varphi \,\mathsf{U}\, \psi) \in \mathrm{CTL}_\exists(\Pi)$.

Existentielle CTL-Formeln

Lemma 18

Zu einer CTL-Zustandsformel $\varphi \in \mathrm{CTL}_s(\Pi)$ kann eine äquivalente existentielle CTL-Zustandsformel $\psi \in \mathrm{CTL}_\exists(\Pi)$ konstruiert werden.

Beweis: Ergibt sich durch Anwendung folgender Äquivalenzen:

- $\forall \mathsf{X}\varphi \equiv \neg\exists \mathsf{X}\neg\varphi$
- $\forall(\varphi \,\mathsf{U}\, \psi) \equiv (\neg\exists \mathsf{G}\neg\psi) \wedge \neg\exists(\neg\psi \,\mathsf{U}\, (\neg\varphi \wedge \neg\psi))$ □

Wichtige Beobachtung: Die obige Umwandlung macht i.A. die Formel exponentiell größer, da ψ in $(\neg\exists \mathsf{G}\neg\psi) \wedge \neg\exists(\neg\psi \,\mathsf{U}\, (\neg\varphi \wedge \neg\psi))$ drei mal vorkommt.

Aber: Die Anzahl der verschiedenen Teilformeln in der zu φ äquivalenten existentielle CTL-Zustandsformel ψ ist beschränkt durch $O(|\varphi|)$.

Existentielle CTL-Formeln

Andere Sichtweise: Der Syntaxbaum der existentiellen CTL-Zustandsformel ψ enthält viele identische Teilbäume.

Indem diese durch Mehrfachverweise nur einmal hingeschrieben werden erhält man eine Repäsentation von ψ durch einen sogenannten DAG (directed acyclic graph) der Größe $O(|\varphi|)$.

Beispiel:

$$\begin{aligned} \forall((p \wedge q)\,\mathsf{U}\,(\neg q \vee r)) \quad &\equiv \quad (\neg\exists\mathsf{G}\neg(\neg q \vee r)) \wedge \\ &\qquad \neg\exists(\neg(\neg q \vee r)\,\mathsf{U}\,(\neg(\neg q \vee r) \wedge \neg(p \wedge q))) \end{aligned}$$

Vergleich von LTL und CTL

Wir sagen, dass eine LTL Formel φ und eine CTL-Zustandsformel ψ äquivalent sind, falls für jeden Transitionsgraphen $T = (V, E, \Pi, \pi)$ und alle $v \in V$ gilt:

$$(T, v) \models \varphi \iff (T, v) \models \psi.$$

Auf den Beweis der folgenden beiden Sätze verzichten wir (siehe z.B. das Buch von Baier und Katoen, S. 337):

Satz 19

Für die LTL-Formeln $\mathsf{FG}p$ und $\mathsf{F}(p \wedge \mathsf{X}p)$ existieren keine äquivalenten CTL-Formeln.

Satz 20

Für die CTL-Formeln $\forall\mathsf{F}\forall\mathsf{G}p$, $\forall\mathsf{F}(p \wedge \forall\mathsf{X}p)$ und $\forall\mathsf{G}\exists\mathsf{F}p$ existieren keine äquivalenten LTL-Formeln.

CTL-Model-Checking

Satz 21 (Entscheidbarkeit von CTL-Model-Checking)

Für eine CTL-Zustandsformel $\varphi \in \mathrm{CTL}_s(\Pi)$ und einen endlichen Transitionsgraphen $T = (V, E, \Pi, \pi)$ kann in Zeit $O(|\varphi| \cdot (|V| + |E|))$ die Menge $\{v \in V \mid (T, v) \models \varphi\}$ berechnet werden.

Beweis: Wir wandeln zunächst in Zeit $O(|\varphi|)$ die Formel φ in eine äquivalente existentielle CTL-Formel ψ um.

Hierbei wird ψ durch einen DAG der Größe $O(|\varphi|)$ repräsentiert.

Die Knoten des DAGs repräsentieren die Teilformeln von ψ.

Für jede solche Teilformel θ berechnen wir die Menge

$$[\![\theta]\!]_T = \{v \in V \mid (T, v) \models \theta\},$$

wobei diese Menge durch ein Bitarray der Größe $|V|$ repräsentiert wird.

CTL-Model-Checking

Wir machen dies induktiv, beginnend bei den Propositionen aus Π.

Am Ende haben wir $[\![\psi]\!]_T = [\![\varphi]\!]_T$ berechnet.

Fall 1. $[\![p]\!]_T = \{v \in V \mid p \in \pi(v)\}$ für alle $p \in \Pi$, die in φ vorkommen.

Da in φ nur $|\varphi|$ viele Propositionen vorkommen können, ist der Aufwand hierfür durch $|\varphi| \cdot |V|$ beschränkt.

Fall 2. $[\![\neg\theta]\!]_T = V \setminus [\![\theta]\!]_T$.

Fall 3. $[\![\theta_1 \wedge \theta_2]\!]_T = [\![\theta_1]\!]_T \cap [\![\theta_2]\!]_T$.

Fall 4. $[\![\theta_1 \vee \theta_2]\!]_T = [\![\theta_1]\!]_T \cup [\![\theta_2]\!]_T$.

In Fall 2–4 kann die Zielmenge in Zeit $O(|V|)$ berechnet werden.

Fall 5. $[\![\exists \mathrm{X}\theta]\!]_T = \{v \in V \mid \exists u \in [\![\theta]\!]_T : (v,u) \in E\}$.

Zur Berechnung dieser Menge durchlaufen wir alle in Knoten aus $[\![\theta]\!]_T$ eingehenden Kanten rückwärts.

Dies kostet Zeit $O(|V| + |E|)$.

CTL-Model-Checking

Fall 6. $[\![\exists(\theta_1 \,\mathsf{U}\, \theta_2)]\!]_T$

Sei $W = [\![\theta_1]\!]_T$ und $U = [\![\theta_2]\!]_T$.

Wir müssen alle Knoten v_0 finden, bei denen ein Pfad $v_0, v_1, \ldots, v_n$ beginnt mit $v_n \in U$ und $v_0, \ldots, v_{n-1} \in W$.

Berechne den gerichteteten Graphen

$$G = (V, \underbrace{E^{-1} \cap (V \times W)}_{=:F}).$$

Dann berechnen wir in Zeit $O(|V| + |E|)$ alle Knoten von G, die im Graphen G von U aus erreichbar sind.

CTL-Model-Checking

Algorithmus $\exists U$

procedure $\exists U(G = (V, F), U \subseteq V)$
begin
  $K := U$;
  while $K \neq \emptyset$ **do**
    wähle einen beliebigen Knoten $v \in K$;
    $K := K \setminus \{v\}$;
    forall $u \in V$ mit $(v, u) \in F$ **do**
      if $u \notin U$ **then**
        $U := U \cup \{u\}$; $K := K \cup \{u\}$
      endif
    endfor
  endwhile
  return (U)
endprocedure

CTL-Model-Checking

Fall 7. $[\![\exists \mathrm{G}\theta]\!]_T$.

Sei $U = [\![\theta]\!]_T$.

Wir müssen also alle Knoten finden, in denen ein unendlicher Pfad beginnt, wo jeder Knoten in U liegt.

Berechne den gerichteteten Graphen

$$G = (U, E \cap (U \times U)),$$

indem alle Knoten aus $V \setminus U$ entfernt werden.

Wir müssen nun in Zeit $O(|V| + |E|)$ alle Knoten berechnen, in denen in G ein unendlicher Pfad beginnt.

Hierzu entfernen wir in G alle Knoten mit Ausgangsgrad 0 (d.h. aus denen keine Kante rausgeht) und wiederholen dies, bis jeder Knoten mindestens eine ausgehende Kante hat.

Die übrig gebliebenden Knoten bilden dann genau die Menge $[\![\exists \mathrm{G}\theta]\!]_T$.

CTL-Model-Checking

Algorithmus $\exists$G

procedure $\exists \mathrm{G}(G = (U, E))$
begin
  forall $u \in U$ **do** $d[u] :=$ Ausgangsgrad(u) **endfor**
  $D := \{u \mid d[u] = 0\}$;
  while $D \neq \emptyset$ **do**
    wähle einen beliebigen Knoten $v \in D$;
    $D := D \setminus \{v\}$; $U := U \setminus \{v\}$;
    forall $u \in U$ mit $(u, v) \in E$ **do**
      $E := E \setminus \{(u, v)\}$; $d[u] := d[u] - 1$;
      if $d[u] = 0$ **then** $D := D \cup \{u\}$ **endif**
    endfor
  endwhile
  return (U)
endprocedure

CTL-Model-Checking

Die Prozeduren zur Berechnung von $[\![\exists(\theta_1 \,\mathsf{U}\, \theta_2)]\!]_T$ und $[\![\exists \mathsf{G}\theta]\!]_T$ können auch als Fixpunktberechnungen betrachtet werden.

Für $[\![\exists(\theta_1 \,\mathsf{U}\, \theta_2)]\!]_T$:

- Sei $U_0 = [\![\theta_2]\!]_T$ und $W = [\![\theta_1]\!]_T$
- Für $i \geq 0$ definiere

$$U_{i+1} = U_i \cup (W \cap \{v \in V \mid \exists v' \in V : (v, v') \in E \wedge v' \in U_i\}).$$

- Dann gilt $U_0 \subseteq U_1 \subseteq U_2 \subseteq U_3 \subseteq \cdots$.
- Da V endlich ist, gibt es ein $i \geq 0$ (sogar $i \leq |V|$) mit $U_i = U_j$ für alle $j \geq i$ und es gilt $[\![\exists(\theta_1 \,\mathsf{U}\, \theta_2)]\!]_T = U_i$

CTL-Model-Checking

Für $[\![\exists \mathrm{G}\theta]\!]_T$:

- Sei $U_0 = [\![\theta]\!]_T$
- Für $i \geq 0$ definiere

$$U_{i+1} = U_i \cap \{v \in V \mid \exists v' \in V : (v, v') \in E \wedge v' \in U_i\}.$$

- Dann gilt $U_0 \supseteq U_1 \supseteq U_2 \supseteq U_3 \supseteq \cdots$.
- Da V endlich ist, gibt es ein $i \geq 0$ (sogar $i \leq |V|$) mit $U_i = U_j$ für alle $j \geq i$ und es gilt $[\![\exists \mathrm{G}\theta]\!]_T = U_i$

CTL*: Eine Verallgemeinerung von LTL und CTL

Zur Erinnerung (Folie 102): LTL und CTL sind in ihrer Ausdrucksstärke unvergleichbar.

CTL* ist eine Logik, die sowohl LTL als auch CTL verallgemeinert.

Sei Π wieder eine endliche Menge von Propositionen.

Definition (Syntax von CTL*)

$\mathrm{CTL}^*_p(\Pi)$ (CTL*-Pfadformeln) und $\mathrm{CTL}^*_s(\Pi)$ (CTL*-Zustandsformeln) sind die bezüglich Inklusion kleinsten Mengen von Formeln mit:

- $\Pi \subseteq \mathrm{CTL}^*_s(\Pi) \subseteq \mathrm{CTL}^*_p(\Pi)$
- Wenn $\varphi, \psi \in \mathrm{CTL}^*_s(\Pi)$, dann $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi \in \mathrm{CTL}^*_s(\Pi)$.
- Wenn $\varphi \in \mathrm{CTL}^*_p(\Pi)$, dann $\forall\varphi, \exists\varphi \in \mathrm{CTL}^*_s(\Pi)$.
- Wenn $\varphi, \psi \in \mathrm{CTL}^*_p(\Pi)$, dann $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi \in \mathrm{CTL}^*_p(\Pi)$.
- Wenn $\varphi, \psi \in \mathrm{CTL}^*_p(\Pi)$, dann $\mathsf{X}\varphi, \varphi\,\mathsf{U}\,\psi \in \mathrm{CTL}^*_p(\Pi)$.

Semantik von CTL*

Sei $T = (V, E, \Pi, \pi)$ ein Transitionsgraph.

CTL*-Zustandsformeln (CTL*-Pfadformeln) werten wir in Knoten (Pfaden) von T aus.

Definition (Semantik von CTL*-Zustandsformeln)

Sei $v \in V$ ein Knoten von T.

- $(T, v) \models p$ falls $p \in \pi(v)$.
- $(T, v) \models \neg\varphi$, falls $(T, v) \models \varphi$ nicht gilt.
- $(T, v) \models \varphi \wedge \psi$, falls $(T, v) \models \varphi$ und $(T, v) \models \psi$ gilt.
- $(T, v) \models \varphi \vee \psi$, falls $(T, v) \models \varphi$ oder $(T, v) \models \psi$ gilt.
- $(T, v) \models \forall\varphi$, falls $(T, p) \models \varphi$ für alle Pfade $p \in \text{path}(T, v)$ gilt.
- $(T, v) \models \exists\varphi$, falls ein Pfad $p \in \text{path}(T, v)$ mit $(T, p) \models \varphi$ existiert.

Semantik von CTL*

Definition (Semantik von CTL*-Pfadformeln)

Sei $p = (v_1 v_2 v_3 \cdots) \in \text{path}(T, v_1)$ ein Pfad in T.

- $(T, p) \models \varphi$, falls $(T, v_1) \models \varphi$ und φ eine CTL*-Zustandsformel ist.
- $(T, p) \models \neg\varphi$, falls $(T, p) \models \varphi$ nicht gilt.
- $(T, p) \models \varphi \wedge \psi$, falls $(T, p) \models \varphi$ und $(T, p) \models \psi$ gilt.
- $(T, p) \models \varphi \vee \psi$, falls $(T, p) \models \varphi$ oder $(T, p) \models \psi$ gilt.
- $(T, p) \models X\varphi$, falls $(T, p^2) \models \varphi$ (wobei $p^i = (v_i v_{i+1} v_{i+2} \cdots)$)
- $(T, p) \models \varphi \,\mathsf{U}\, \psi$, falls ein $i \geq 1$ existiert mit:
  - $(T, p^i) \models \psi$ und
  - $(T, p^j) \models \varphi$ für alle $1 \leq j < i$.

Wir verwenden die gleichen Abkürzungen wie für LTL und CTL.

CTL* ist ausdrucksstärker als LTL und CTL

Offensichtlich können wir eine CTL-Formel in eine äquivalente CTL*-Formel übersetzen.

Ausserdem gilt für jede LTL-Formel $\varphi \in \mathrm{LTL}(\Pi)$, jeden Transitionsgraphen $T = (V, E, \Pi, \pi)$ und jeden Knoten $v \in V$:

$$(T, v) \models_{\mathrm{LTL}} \varphi \iff (T, v) \models_{\mathrm{CTL}^*} \forall\varphi$$

Wir verzichten auf den Beweis des folgenden Satzes (siehe z.B. das Buch von Baier und Katoen).

Satz 22

Für die CTL**-Formel* $(\forall \mathrm{FG} p) \vee (\forall \mathrm{G} \exists \mathrm{F} q)$ *gibt es weder eine äquivalente CTL-Formel, noch eine äquivalente LTL-Formel.*

CTL*-Model-Checking

Satz 23

Für eine CTL*-*Zustandsformel* φ*, einen endlichen Transitionsgraphen* $T = (V, E, \Pi, \pi)$ *und einen Knoten* $v \in V$ *kann in Zeit* $2^{O(|\varphi|)} \cdot (|V| + |E|)$ *entschieden werden, ob* $(T, v) \models \varphi$ *gilt.*

Beweis: Wie im CTL-Model-Checking Algorithmus berechnen wir für jede Teilzustandsformel θ von φ die Menge $[\![\theta]\!]_T = \{v \in V \mid (T, v) \models \theta\}$.

Fall 1. $[\![p]\!]_T = \{v \in V \mid p \in \pi(v)\}$ für alle $p \in \Pi$, die in φ vorkommen.

Da in φ nur $|\varphi|$ viele Propositionen vorkommen können, ist der Aufwand hierfür durch $|\varphi| \cdot |V|$ beschränkt.

Fall 2. $[\![\neg\theta]\!]_T = V \setminus [\![\theta]\!]_T$.

Fall 3. $[\![\theta_1 \wedge \theta_2]\!]_T = [\![\theta_1]\!]_T \cap [\![\theta_2]\!]_T$.

Fall 4. $[\![\theta_1 \vee \theta_2]\!]_T = [\![\theta_1]\!]_T \cup [\![\theta_2]\!]_T$.

In Fall 2–4 kann die Zielmenge in Zeit $O(|V|)$ berechnet werden.

CTL*-Model-Checking

Fall 5. $[\![\forall\psi]\!]_T$:

Dann ist ψ eine Pfadformel.

Für jede maximale (maximal bezüglich der Teilformelrelation) Teilzustandsformel θ in ψ haben wir die Menge $[\![\theta]\!]_T$ bereits berechnet.

Wir führen nun für jede solche maximale Zustandsformel θ eine neue Proposition $p_\theta \notin \Pi$ ein.

Sei Π_ψ die Menge aller dieser neuen Propositionen.

Wir definieren den Transitionsgraphen $T' = (V, E, \Pi \cup \Pi_\psi, \pi')$, wobei

$$\pi'(v) = \pi(v) \cup \{p_\theta \in \Pi_\psi \mid v \in [\![\theta]\!]_T\}$$

Nun ersetzen wir in ψ jedes maximale Vorkommen von θ durch die Proposition p_θ.

CTL*-Model-Checking

Auf diese Weise erhalten wir aus ψ eine LTL-Formel ψ' und es gilt:

$$[\![\forall\psi]\!]_T = [\![\forall\psi']\!]_{T'} = \{v \in V \mid (T', v) \models_{\mathsf{LTL}} \psi'\}.$$

Die Menge $\{v \in V \mid (T, v) \models_{\mathsf{LTL}} \psi'\}$ kann in Zeit $2^{O(|\psi'|)} \cdot (|V| + |E|)$ mittels einer Variation des LTL-Model-Checking Algorithmus (Folie 58–70) berechnet werden.

CTL*-Model-Checking

Fall 6. $[\![\exists\psi]\!]_T$

Wegen $\exists\psi \equiv \neg\forall\neg\psi$ gilt $[\![\exists\psi]\!]_T = V \setminus [\![\forall\neg\psi]\!]_T$.

Wir berechnen $[\![\forall\neg\psi]\!]_T$ wie in Fall 5, und danach $V \setminus [\![\forall\neg\psi]\!]_T$.

Der Zeitaufwand ist wieder $2^{O(|\psi'|)} \cdot (|V| + |E|)$, wobei ψ' aus ψ wie in Fall 5 konstruiert wird.

Addiert man die Beiträge für den Zeitaufwand aus Fall 1–6, so kommt man insgesamt auf die Zeitschranke $2^{O(|\varphi|)} \cdot (|V| + |E|)$. □

Bisimulation

Idea: Versuche Zustände, die die gleichen CTL*-Zustandsformeln erfüllen, zu verschmelzen, um den Transitionsgraphen für das Model-Checking kleiner zu machen.

Bisimulation

Sei $T = (V, E, \Pi, \pi)$ ein Transitionsgraph.

Eine binäre Relation $R \subseteq V \times V$ ist eine Bisimulation auf T, falls folgende Bedingungen für alle $(u, v) \in R$ gelten:

- $\pi(u) = \pi(v)$.
- Für alle $u' \in \mathrm{suc}_T(u)$ existiert $v' \in \mathrm{suc}_T(v)$ mit $(u', v') \in R$.
- Für alle $v' \in \mathrm{suc}_T(v)$ existiert $u' \in \mathrm{suc}_T(u)$ mit $(u', v') \in R$.

Manchmal wollen wir auch eine Bisimulation zwischen zwei verschiedenen Transitionsgraphen T und T' definieren. Dies ist dann eine Bisimulation auf $T \uplus T'$ (disjunkte Vereinigung von T und T').

Bisimulationsäquivalenz

Bisimulationsäquivalenz

Die Relation

$$\sim_T \;=\; \bigcup_{R \text{ is Bisimulation auf } T} R$$

ist die Bisimulationsäquivalenz (bzw. Verhaltensäquivalenz) auf T.

Wir schreiben auch $\sim$ anstatt $\sim_T$ wenn T aus dem Kontext klar ist.

Satz 24

Die Relation $\sim_T$ ist (i) eine Bisimulation auf T und (ii) eine Äquivalenzrelation.

Bisimulationsäquivalenz

Beweis:

Zu (i): Es genügt zu zeigen: Wenn $\mathcal{B}$ eine Menge von Bisimulationen auf T ist, dann ist auch

$$R = \bigcup_{P \in \mathcal{B}} P$$

eine Bisimulation auf T.

Gelte $(u, v) \in R$, d.h. es existiert ein $P \in \mathcal{B}$ mit $(u, v) \in P$.

- Da P ein Bisimulation ist, gilt $\pi(u) = \pi(v)$.
- Sei $u' \in \mathrm{suc}_T(u)$. Da P ein Bisimulation ist, existiert $v' \in \mathrm{suc}_T(v)$ mit $(u', v') \in P$ und damit auch $(u', v') \in R$.
- Analog existiert für jeden Knoten $v' \in \mathrm{suc}_T(v)$ ein $u' \in \mathrm{suc}_T(u)$ mit $(u', v') \in R$.

Bisimulationsäquivalenz

Zu (ii). Wir zeigen, dass $\sim$ reflexiv, symmetrisch und transitiv ist.

$\sim$ ist reflexiv: Offensichtlich ist $\mathrm{Id}_V = \{(u,u) \mid u \in V\}$ eine Bisimulation.

Also gilt $\mathrm{Id}_V \subseteq \sim$, d.h. $\sim$ ist reflexiv.

$\sim$ ist symmetrisch: Gelte $u \sim v$.

Also gibt es eine Bisimulation P mit $(u,v) \in P$.

Dann ist auch $P^{-1} = \{(y,x) \mid (x,y) \in P\}$ eine Bisimulation.

Wegen $(v,u) \in P^{-1}$ gilt also $v \sim u$.

$\sim$ ist transitiv: Gelte $u \sim v \sim w$.

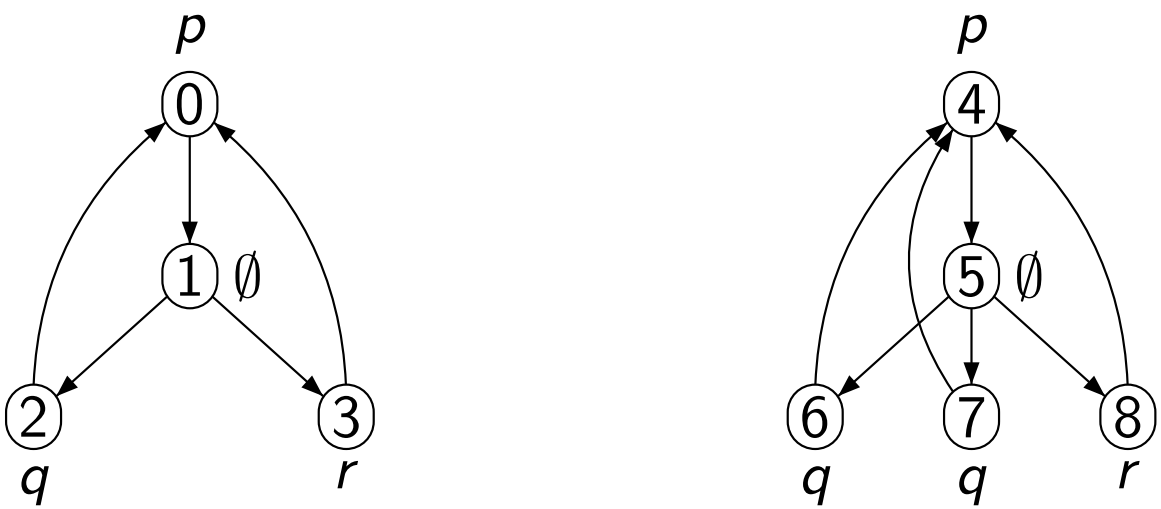
Also gibt es Bisimulationen P und Q mit $(u,v) \in P$ und $(v,w) \in Q$.

Dann ist auch $P \circ Q = \{(x,z) \mid \exists y \in V : (x,y) \in P \wedge (y,z) \in Q\}$ eine Bisimulation.

Wegen $(u,w) \in P \circ Q$ gilt also $u \sim w$. □

Bisimulationsäquivalenz

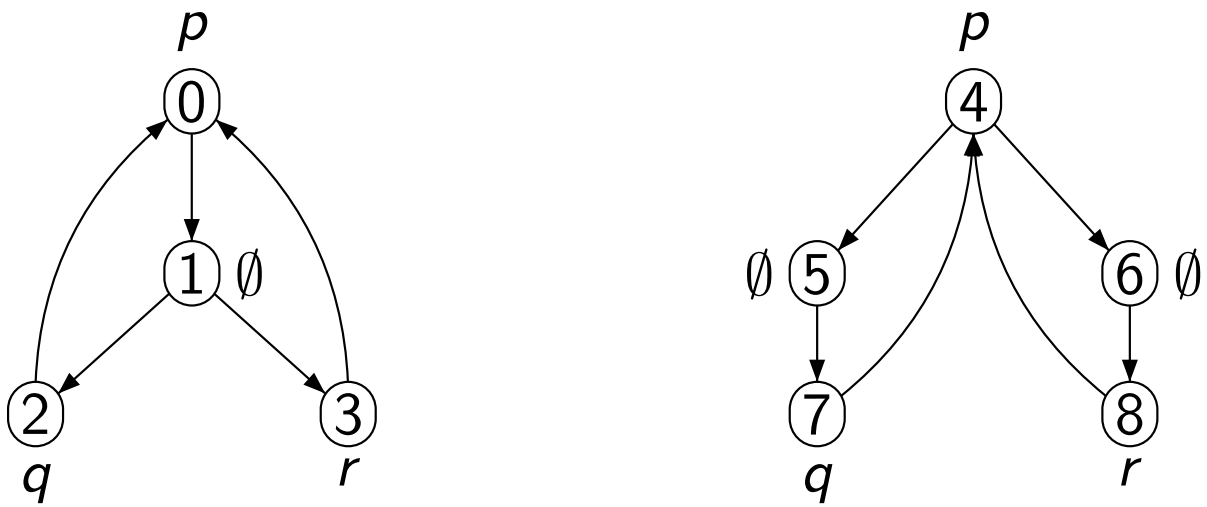
Beispiel 1:



Es gilt $0 \sim 4$: Eine Bisimulation ist $\{(0,4),(1,5),(2,6),(2,7),(3,8)\}$.

Bisimulationsäquivalenz

Beispiel 2:



Es gilt $0 \not\sim 4$:

Angenommen R wäre eine Bisimulation mit $(0, 4) \in R$.

Dann müsste $(1, 5) \in R$ oder $(1, 6) \in R$ gelten.

Angenommen es gilt $(1, 5) \in R$ (der Fall $(1, 6) \in R$ ist analog).

Dann müsste $(3, 7) \in R$ gelten.

Dies ist jedoch nicht der Fall, da $\pi(3) = \{r\} \neq \{q\} = \pi(7)$ gilt.

Bisimulationsäquivalenz versus Trace-Äquivalenz

Sei $T = (V, E, \Pi, \pi)$ ein Transitionsgraph, wobei wir annehmen, dass $\mathrm{suc}_T(v) \neq \emptyset$ für alle $v \in V$ gelte.

Die Menge $\omega\text{-tr}(T, u)$ wurde auf Folie 27 definiert.

Zwei Knoten $u, v \in V$ sind trace-äquivalent, kurz $u \equiv_{\mathrm{tr}} v$, falls gilt:

$$\omega\text{-tr}(T, u) = \omega\text{-tr}(T, v)$$

Es ist leicht zu sehen, dass $u \equiv_{\mathrm{tr}} v$ aus $u \sim v$ folgt.

Die Umkehrung ist falsch: In Beispiel 2 auf der vorherigen Folie gilt $0 \equiv_{\mathrm{tr}} 4$.

Bisimulationsquotient

Für einen Transitionsgraphen $T = (V, E, \Pi, \pi)$ definieren wir den Bisimulationsquotienten

$$T/_\sim = (V', E', \Pi, \pi')$$

wobei gilt:

- $V' = \{[v]_\sim \mid v \in V\}$ ist die Menge der Äquivalenklassen von $\sim$
- $E' = \{([u]_\sim, [v]_\sim) \mid (u, v) \in E\}$
- $\pi'([u]_\sim) = \pi(u)$

Beachte: Aus $u \sim v$ folgt $\pi(u) = \pi(v)$; dies ist für die Definition von π' wichtig.

Bisimulationsquotient

Lemma 25

Sei $T = (V, E, \Pi, \pi)$ ein Transitionsgraph und $T' = T/_\sim = (V', E', \Pi, \pi')$.
Die Relation $R = \{(u, [u]_\sim) \mid u \in V\}$ ist eine Bisimulation (auf $T \uplus T'$).

Beweis: Betrachte ein Paar $(u, [u]_\sim) \in R$.

Es gilt $\pi(u) = \pi'([u]_\sim)$.

Sei $v \in \mathrm{suc}_T(u)$. Dann gilt $[v]_\sim \in \mathrm{suc}_{T'}([u]_\sim)$ und $(v, [v]_\sim) \in R$.

Sei $[v]_\sim \in \mathrm{suc}_{T'}([u]_\sim)$.

Also gibt es eine Kante $(x, y) \in E$ mit $x \sim u$ und $y \sim v$.

Also gibt es $w \in \mathrm{suc}_T(u)$ mit $v \sim y \sim w$, d.h. $v \sim w$.

Wir erhalten $(w, [v]_\sim) = (w, [w]_\sim) \in R$. □

Bisimulationsinvarianz von CTL*

Satz 26

Sei T ein Transitionsgraph und seien u und v Knoten mit $u \sim_T v$. Dann gilt für jede CTL**-Zustandsformel φ:* $(T, u) \models \varphi \iff (T, v) \models \varphi$.

Beweis: Wir zeigen die Aussage durch Induktion über den Aufbau der CTL*-Zustandsformel φ.

Da wir während der Induktion auch eine Aussage für CTL*-Pfadformeln beweisen müssen, benötigen wir die folgenden Begriffe.

Für Pfade $p = (u_1 u_2 u_3 \cdots) \in \text{path}(T, u_1)$, $q = (v_1 v_2 v_3 \cdots) \in \text{path}(T, v_1)$ schreiben wir $p \sim q$ genau dann, wenn $u_i \sim v_i$ für alle $i \geq 1$ gilt.

Bisimulationsinvarianz von CTL*

Behauptung: Wenn $u_1 \sim v_1$, dann existiert für jeden Pfad $p = (u_1 u_2 u_3 \cdots) \in \text{path}(T, u_1)$ ein Pfad $q = (v_1 v_2 v_3 \cdots) \in \text{path}(T, v_1)$ mit $p \sim q$ (und umgekehrt ebenso).

Beweis der Behauptung:

Angenommen $v_1, \ldots, v_n$ sind bereits konstruiert.

Es gilt $u_n \sim v_n$ und $u_{n+1} \in \text{suc}_T(u_n)$.

Also existiert ein $v_{n+1} \in \text{suc}_T(v_n)$ mit $u_{n+1} \sim v_{n+1}$.

Wir können nun die beiden folgenden Aussagen simultan durch eine Induktion über den Formelaufbau zeigen:

- Für jede CTL*-Zustandsformel φ und alle Knoten u, v mit $u \sim v$ gilt: $(T, u) \models \varphi \iff (T, v) \models \varphi$.
- Für jede CTL*-Pfadformel ψ und alle Pfade p, q mit $p \sim q$ gilt: $(T, p) \models \psi \iff (T, q) \models \psi$.

Bisimulationsinvarianz von CTL*

Zunächst betrachten wir Zustandsformeln φ, wobei $u \sim v$.

Fall 1: $\varphi = p \in \Pi$.

Dann gilt $\pi(u) = \pi(v)$ und damit $(T, u) \models p \iff (T, v) \models p$.

Fall 2: $\varphi = \neg\varphi'$, $\varphi = \varphi_1 \wedge \varphi_2$

Direkte Anwendung der Induktionshypothese für φ', φ_1 und φ_2.

Fall 3: $\varphi = \exists\psi$ für eine Pfadformel ψ. Dann gilt:

$$\begin{aligned}
(T, u) \models \varphi &\iff \exists p \in \mathrm{path}(T, u) : (T, p) \models \psi \\
&\iff \exists q \in \mathrm{path}(T, v) : (T, q) \models \psi \\
&\iff (T, v) \models \varphi
\end{aligned}$$

Die zweite Äquivalenz folgt dabei aus der Behauptung von der vorherigen Folie sowie der Induktionshypothese für die Pfadformel ψ.

Bisimulationsinvarianz von CTL*

Nun betrachten wir Pfadformeln ψ.

Sei $p \in \text{path}(T, u)$, $q \in \text{path}(T, v)$, und $p \sim q$.

Fall 4: ψ ist eine Zustandsformel.

Da wir Zustandsformeln bereits abgehandelt haben, gilt:

$$\begin{aligned} (T, p) \models \psi &\iff (T, u) \models \psi \\ &\iff (T, v) \models \psi \\ &\iff (T, q) \models \psi \end{aligned}$$

Fall 5: $\psi = \neg\psi'$, $\psi = \psi_1 \wedge \psi_2$

Direkte Anwendung der Induktionshypothese für ψ', ψ_1 und ψ_2.

Bisimulationsinvarianz von CTL*

Fall 7: $\psi = \mathsf{X}\psi'$ für eine Pfadformel ψ'.

Offensichtlich folgt aus $p \sim q$, dass $p^i \sim q^i$ für alle $i \geq 1$ gilt.

Also erhalten wir:

$$\begin{aligned} (T, p) \models \psi &\iff (T, p^2) \models \psi' \\ &\iff (T, q^2) \models \psi' \\ &\iff (T, q) \models \psi \end{aligned}$$

Fall 8: $\psi = \psi_1 \,\mathsf{U}\, \psi_2$ für Pfadformeln ψ_1, ψ_2: analog

CTL*-Äquivalenz und CTL-Äquivalenz

Für einen Transitionsgraphen $T = (V, E, \Pi, \pi)$ definieren wir die folgenden beiden Äquivalenzrelationen auf der Knotenmenge V:

$$u \equiv_{\mathsf{CTL}} v \iff \forall \varphi \in \mathsf{CTL}_s(\Pi) : (T, u) \models \varphi \Leftrightarrow (T, v) \models \varphi$$

$$u \equiv_{\mathsf{CTL}^*} v \iff \forall \varphi \in \mathsf{CTL}_s(\Pi) : (T, u) \models \varphi \Leftrightarrow (T, v) \models \varphi$$

Satz 26 kann dann auch kurz wie folgt formuliert werden: $\sim \subseteq \equiv_{\mathsf{CTL}^*}$.

Wir zeigen nun folgenden Satz:

Satz 27

Für jeden endlichen Transitionsgraphen T gilt: $\sim = \equiv_{\mathsf{CTL}^*} = \equiv_{\mathsf{CTL}}$.

CTL*-Äquivalenz und CTL-Äquivalenz

Beweis:

Da wir $\sim \subseteq \equiv_{\mathrm{CTL}^*}$ bereits gezeigt haben, und $\equiv_{\mathrm{CTL}^*} \subseteq \equiv_{\mathrm{CTL}}$ offensichtlich gilt, genügt es $\equiv_{\mathrm{CTL}} \subseteq \sim$ zu zeigen.

Wir zeigen, dass $\equiv_{\mathrm{CTL}}$ eine Bisimulation ist.

Seien $u, v \in V$ mit $u \equiv_{\mathrm{CTL}} v$.

Insbesondere gilt für jede Proposition $p \in \Pi$: $(T, u) \models p \Leftrightarrow (T, v) \models p$.

Also gilt $\pi(u) = \pi(v)$.

Sei nun $u' \in \mathrm{suc}_T(u)$.

Wir zeigen, dass ein $v' \in \mathrm{suc}_T(v)$ mit $u' \equiv_{\mathrm{CTL}} v'$ existiert.

CTL*-Äquivalenz und CTL-Äquivalenz

Sei $V' = V/_{\equiv_{\mathrm{CTL}}}$ die Menge der Aquivalenzklassen von $\equiv_{\mathrm{CTL}}$.

Seien $C, D \in V'$ mit $C \neq D$.

Dann gibt es eine Formel $\varphi_{C,D}$ so dass für alle $x \in C$ und $y \in D$ gilt:

$$(T, x) \models \varphi_{C,D} \text{ und } (T, y) \not\models \varphi_{C,D}.$$

Sei nun

$$\varphi_C = \bigwedge_{D \in V' \setminus \{C\}} \varphi_{C,D}.$$

Dann gilt für alle $x \in C$ und alle $y \in V \setminus C$:

$$(T, x) \models \varphi_C \text{ und } (T, y) \not\models \varphi_C.$$

Sei nun C (bzw. C') die Äquivalenzklasse mit $u \in C$ (bzw. $u' \in C'$).

CTL*-Äquivalenz und CTL-Äquivalenz

Dann gilt $(T, u) \models \exists \mathsf{X} \varphi_{C'}$.

Wegen $u \equiv_{\mathsf{CTL}} v$ gilt auch $(T, v) \models \exists \mathsf{X} \varphi_{C'}$.

Sei $v' \in \mathrm{suc}_T(v)$ mit $(T, v') \models \varphi_{C'}$.

Dann gilt $u' \equiv_{\mathsf{CTL}} v'$. □

Bemerkungen: Der obige Beweis verwendet nur die temporalen Operatoren $\exists \mathsf{X}$ und $\forall \mathsf{X}$ (aber kein $\exists\,\mathsf{U}$ oder $\forall\,\mathsf{U}$).

Das CTL-Fragment, dass nur die temporalen Operatoren $\exists \mathsf{X}$ und $\forall \mathsf{X}$ benutzt, ist auch als Modallogik bekannt.

Satz 27 ist für unendliche Transitionsgraphen im Allgemeinen falsch (siehe das Buch von Baier und Katoen für eine Gegenbeispiel).

CTL*-Model-Checking auf dem Bisimulationsquotienten

Lemma 25 und Satz 26 erlauben uns, beim CTL*-Model-Checking den Transitionsgraphen T durch den (im Allgemeinen kleineren) Bisimulationsquotienten $T/_\sim$ zu ersetzen.

Genauer: Sei T ein Transitionsgraph, v ein Knoten von T und φ eine CTL*-Zustandsformel.

Dann können wir $(T, v) \models \varphi$ wie folgt überprüfen:

1. Berechne die Bisimulationsäquivalent $\sim$.
2. Berechne den Bisimulationsquotienten $T/_\sim$, indem alle Äquivalenzklassen von $\sim$ in einen Zustand kollabiert werden.
3. Überprüfe mittels des CTL*-Model-Checking Algorithmus, ob $(T/_\sim, [v]_\sim) \models \varphi$ gilt.

Berechnung der Bisimulationsäquivalenz

Unser Ziel im Folgenden: Effizienter Algorithmus zur Berechnung der Bisimulationsäquivalent $\sim$.

Lösung: Partitionsverfeinerung ähnlich zur Minimierung von endlichen DFAs (GTI).

Sei $T = (V, E, \Pi, \pi)$ der Transitionsgraph, wobei o.B.d.A. $V = \{1, \ldots, n\}$ für ein $n \geq 1$ gelte.

Wir erstellen eine Tabelle, die für jedes Paar (u, v) mit $u, v \in V$, $u < v$, einen Eintrag enthält. Dieser Eintrag kann markiert oder nicht markiert sein.

Idee: Sobald ein Paar (u, v) markiert ist, haben wir $u \not\sim v$ nachgewiesen.

Berechnung der Bisimulationsäquivalenz

Partitionsverfeinerungsalgorithmus:

1. Zu Beginn: (u, v) ist markiert genau dann, wenn $\pi(u) \neq \pi(v)$ gilt.
2. Falls (u, v) ein noch unmarkiertes Paar mit einer der folgenden Eigenschaften ist, wird (u, v) markiert:
   - Es existiert $u' \in \mathrm{suc}_T(u)$, so dass für alle $v' \in \mathrm{suc}_T(v)$ gilt: $u' < v'$ und (u', v') ist bereits markiert, oder $v' < u'$ und (v', u') ist bereits markiert.
   - Es existiert $v' \in \mathrm{suc}_T(v)$, so dass für alle $u' \in \mathrm{suc}_T(u)$ gilt: $u' < v'$ und (u', v') ist bereits markiert, oder $v' < u'$ und (v', u') ist bereits markiert.
3. Sobald keine neuen Paare mehr markiert werden, stoppt der Algorithmus.

Berechnung der Bisimulationsäquivalenz

Behauptung 1: Die Relation $R = \mathrm{Id}_V \cup R_0 \cup R_1$ mit

$$\begin{aligned} R_0 &= \{(u,v) \mid u,v \in V, u < v, (u,v) \text{ ist am Ende nicht markiert}\}, \\ R_1 &= \{(u,v) \mid u,v \in V, u > v, (v,u) \text{ ist am Ende nicht markiert}\} \end{aligned}$$

ist eine Bisimulation auf T.

Beweis von Behauptung 1

Wegen Schritt 1 aus dem Partitionsverfeinerungsalgorithmus gilt $\pi(u) = \pi(v)$ für alle $(u,v) \in R$.

Sei nun $(u,v) \in R$ und $u' \in \mathrm{suc}_T(u)$.

Wir müssen $v' \in \mathrm{suc}_T(v)$ mit $(u',v') \in R$ finden.

Fall 1: $u = v$. Trivial.

Berechnung der Bisimulationsäquivalenz

Fall 2: $(u, v) \in R_0$.

Da (u, v) nicht markiert wird, muss ein $v' \in \mathrm{suc}_T(v)$ mit einer der drei folgenden Eigenschaften existieren:

- $u' = v'$, d.h. $(u', v') \in \mathrm{Id}_V$,
- $u' < v'$ und (u', v') ist nicht markiert, d.h. $(u', v') \in R_0$,
- $v' < u'$ und (v', u') ist nicht markiert, d.h. $(u', v') \in R_1$.

Also gilt $(u', v') \in R$.

Fall 3: $(u, v) \in R_1$, d.h. $(v, u) \in R_0$. Analoge Argumentation wie in Fall 2.

Analog zeigt man, dass für alle $(u, v) \in R$ und $v' \in \mathrm{suc}_T(v)$ ein $u' \in \mathrm{suc}_T(v)$ mit $(u', v') \in R$ existiert. □

Aus Behauptung 1 folgt $R \subseteq \sim_T$.

Wir zeigen nun die andere Implikation.

Berechnung der Bisimulationsäquivalenz

Behauptung 2: Wenn ein Paar (u, v) markiert wird, gilt $u \not\sim_T v$.

Beweis von Behauptung 2:

Für Paare (u, v), die in Schritt 1 des Paritionsverfeinerungsalgorithmus markiert werden, gilt offensichtlich $u \not\sim_T v$.

Angenommen (u, v) wird in irgendwann in Schritt 2 des Paritionsverfeinerungsalgorithmus markiert.

Mit Induktion, können wir annehmen, dass für alle vorher markierten Paare (u', v') gilt: $u' \not\sim_T v'$.

Da (u, v) markiert wird, muss einer der beiden folgenden Fälle existieren:

Berechnung der Bisimulationsäquivalenz

- Es existiert $u' \in \mathrm{suc}_T(u)$, so dass für alle $v' \in \mathrm{suc}_T(v)$ gilt: $u' < v'$ und (u', v') ist bereits markiert, oder $v' < u'$ und (v', u') ist bereits markiert.

  In beiden Fällen gilt $u' \not\sim_T v'$ nach Induktion.

- Es existiert $v' \in \mathrm{suc}_T(v)$, so dass für alle $u' \in \mathrm{suc}_T(u)$ gilt: $u' < v'$ und (u', v') ist bereits markiert, oder $v' < u'$ und (v', u') ist bereits markiert.

  In beiden Fällen gilt $u' \not\sim_T v'$ nach Induktion.

Wieder ergibt sich $u \not\sim_T v$. □

Die berechnete Relation R ist also die Bisimulationsäquivalenz.

Berechnung der Bisimulationsäquivalenz

Die Laufzeit unseres Bisimulationsverfeinerungsalgorithmus ist zwar polynomiell in $|V| + |E|$, aber bei naiver Implementierung nicht besonders gut.

Bester bekannter Algorithmus zur Berechnung der Bisimulationsäquivalenz: $O(|E| \cdot \log |V|)$.

(Paige, Tarjan, Three Partition Refinement Algorithms, SIAM J. Comput., 16(6), 973–989)

Boolesche Funktionen

Normalerweise definiert man eine Boolesche Funktion als eine Funktion $f : \{0,1\}^m \to \{0,1\}$ für ein $m \geq 1$.

Für unsere Zwecke ist eine alternative Definition nützlicher.

Definition (Belegung)

Sei X eine endliche Menge von Booleschen Variablen.

Dann ist $\{0,1\}^X$ die Menge aller Funktionen von X nach $\{0,1\}$.

Eine Element $\beta \in \{0,1\}^X$ ist eine Belegung für X.

Definition (Boolesche Funktion)

Eine Boolesche Funktion über der Variablenmenge X ist eine Funktion $f : \{0,1\}^X \to \{0,1\}$.

Boolesche Funktionen

Im folgenden sei $X' = \{x' \mid x \in X\}$ stets eine disjunkte Kopie von X.

Für $\beta \in \{0,1\}^X$ sei $\beta' \in \{0,1\}^{X'}$ definiert durch $\beta'(x') = \beta(x)$ für alle $x \in X$.

Bemerkungen:

- Eine Belegung $\alpha \in \{0,1\}^{X \cup X'}$ kann mit dem Paar $(\alpha|_X, \alpha|_{X'}) \in \{0,1\}^X \times \{0,1\}^{X'}$ identifiziert werden.

  Hierbei ist $\alpha|_X$ die Einschränkung der Funktion $\alpha : X \cup X' \to \{0,1\}$ auf die Menge X und analog für $\alpha|_{X'}$.

- Eine Boolesche (oder aussagenlogische) Formel F, in der die Variablen aus X vorkommen, kann mit einer Booleschen Funktion $f_F : \{0,1\}^X \to \{0,1\}$ identifiziert werden.

  Hierbei ist $f_F(\beta)$ der Wahrheitswert von F unter der Belegung β.

Symbolische Repräsentation von Transitionsgraphen

Ein Transitionsgraph $T = (V, E, \Pi, \pi)$ kann symbolisch durch Boolesche Funktionen kodiert werden (wir nehmen an, dass $|V|$ eine Zweierpotenz ist):

- $V = \{0,1\}^X$ für eine Variablenmenge X.
- $f_E : \{0,1\}^{X \cup X'} \to \{0,1\}$ mit:

$$\forall \beta, \gamma \in \{0,1\}^X : f_E(\beta, \gamma') = 1 \iff (\beta, \gamma) \in E.$$

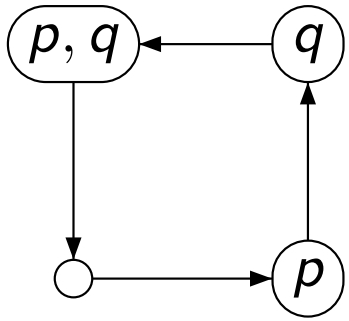
- $f_p : \{0,1\}^X \to \{0,1\}$ für $p \in \Pi$ mit:

$$\forall \beta \in \{0,1\}^X : f_p(\beta) = 1 \iff p \in \pi(\beta).$$

Daher sind wir an kompakten Darstellungen von Booleschen Funktionen interessiert.

Beispiel für symbolische Repräsentation

Sei $T = (V, E, \{p, q\}, \pi)$ der folgende Transitionsgraph (Zähler modulo 4):



Dann kann T durch die folgenden Booleschen Funktionen repräsentiert werden:

$$f_E(x_1, x_2, x_1', x_2') = (x_1 \leftrightarrow \neg x_1') \wedge (\neg x_1 \rightarrow ((x_2 \leftrightarrow x_2')) \wedge (x_1 \rightarrow ((x_2 \leftrightarrow \neg x_2'))$$

$$f_p(x_1, x_2) = x_1,\ f_q(x_1, x_2) = x_2.$$

Entscheidungsbäume

Entscheidungsbaum

Ein Entscheidungsbaum (über der endlichen Variablenmenge X, $m = |X|$) ist ein Knotenbeschrifteter Binärbaum T mit folgenden Eigenschaften

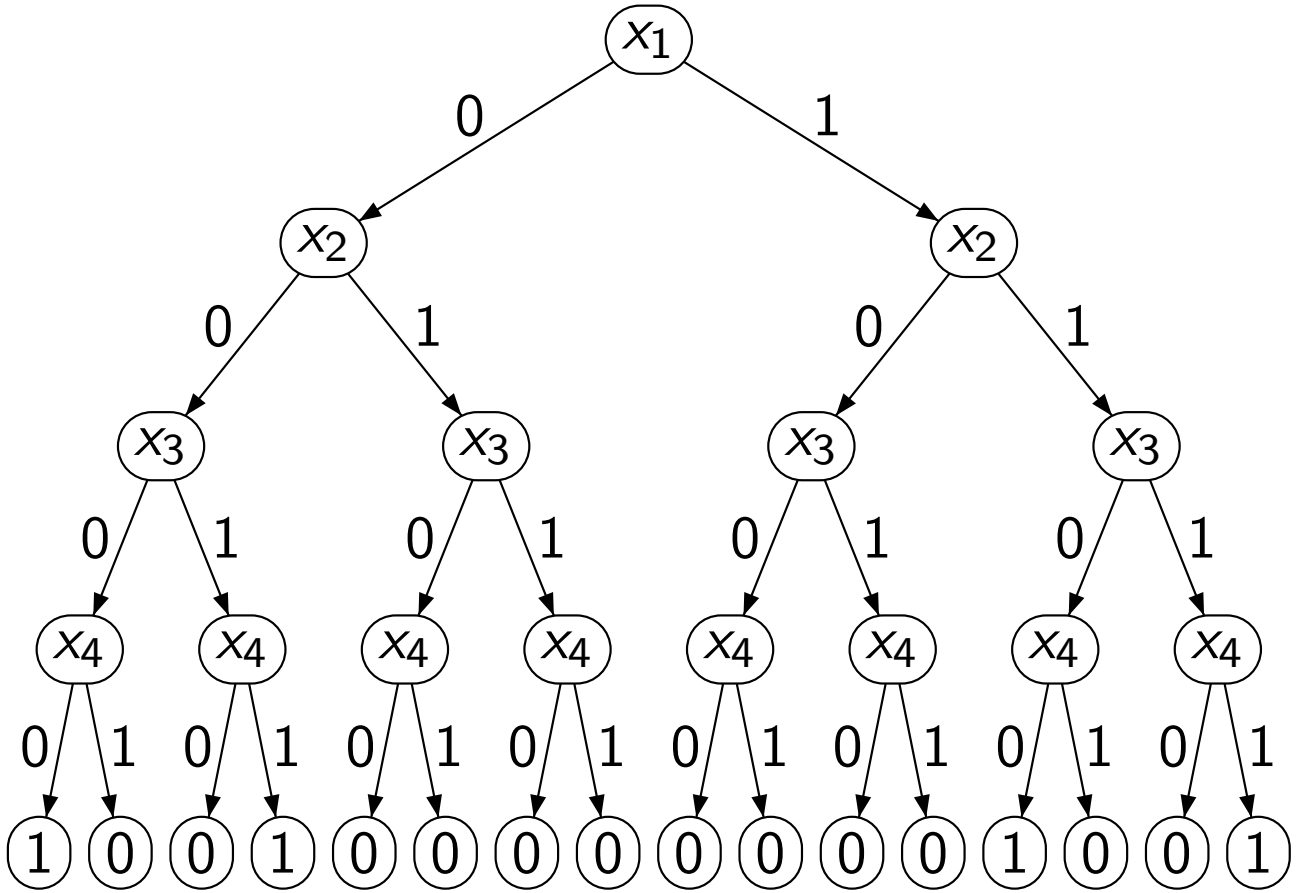
- Die Knoten sind alle Bitstrings $v \in \{0,1\}^*$ mit $|v| \leq m$.
- Das linke (bzw. rechte) Kind von v ist $v0$ (bzw. $v1$).
- Für alle $0 \leq i \leq m-1$ existiert eine Variable $x_i \in X$, so dass alle Knoten v mit $|v| = i$ mit x_i beschriftet sind.
- $x_i \neq x_j$ für $i \neq j$.
- Jedes Blatt v ($|v| = m$) ist mit einem Bit $b(v) \in \{0,1\}$ beschriftet.

T definiert eine Boolesche Funktion $f_T : \{0,1\}^X \to \{0,1\}$ durch:

$$\forall \beta \in \{0,1\}^X : f_T(\beta) = b(\beta(x_0)\beta(x_1)\cdots\beta(x_{m-1}))$$

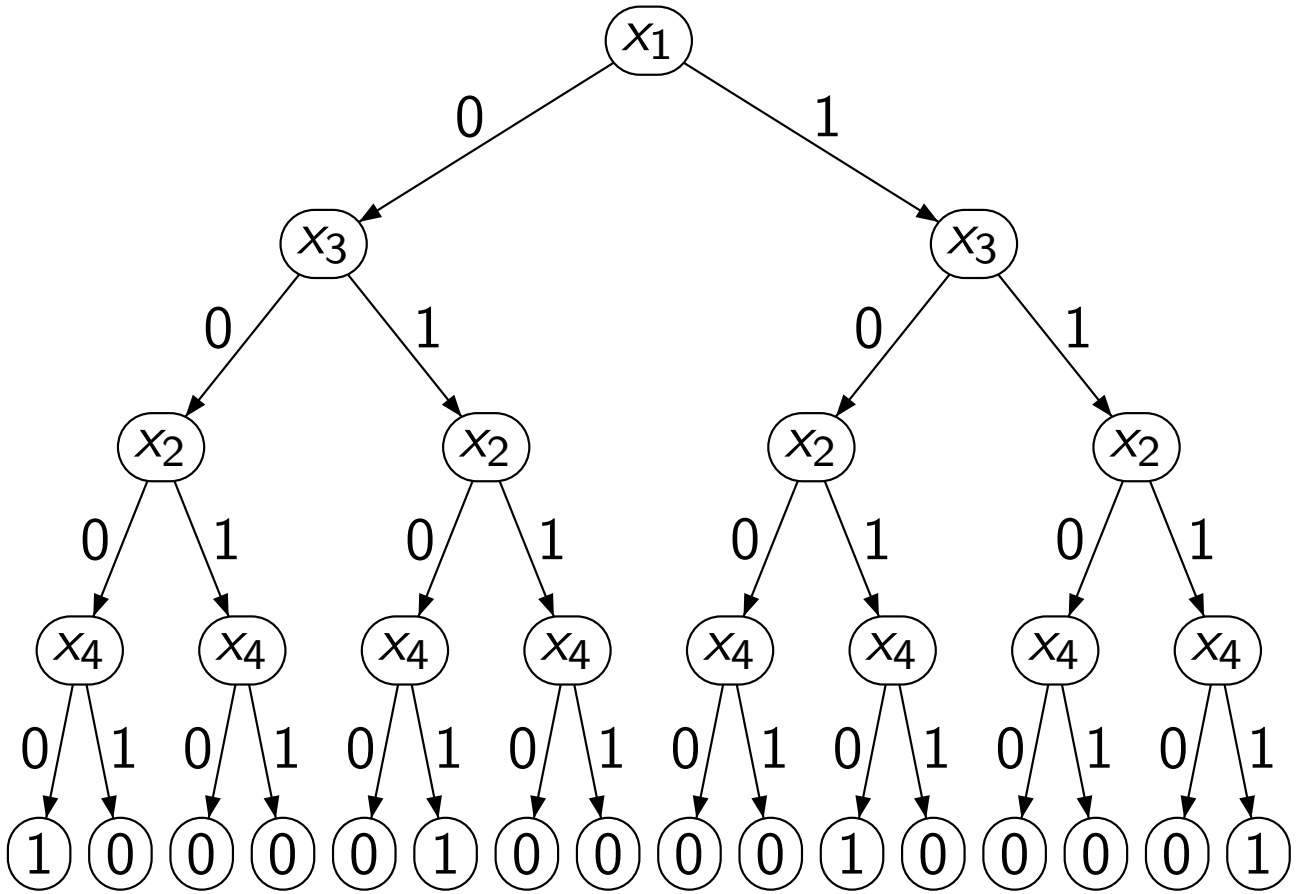
Entscheidungsbäume: Beispiel 1

Ein Entscheidungsbaum für die Boolesche Funktion $(x_1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4)$ (die Knotennamen $v \in \{0,1\}^*$ lassen wir hier weg, dafür haben wir die Kanten von v nach vb mit $b \in \{0,1\}$ beschriftet):



Entscheidungsbäume: Beispiel 2

Und ein zweiter Entscheidungsbaum für die gleiche Boolesche Funktion:



OBDDs

Geordnete binäre Entscheidungsdiagramme (ordered binary decision diagrams, kurz OBDDs) liefern eine kompakte Darstellung von Entscheidungsbäumen.

Idee: Identische Teilbäume eines Entscheidungsbaums werden nur einmal dargestellt.

Definition (Variablenordnung)

Sei X eine endliche Menge von Booleschen Variablen. Eine Variablenordnung auf X ist eine lineare Ordnung $<$ auf X.

OBDDs

Definition (OBDD)

Ein OBDD (bzgl. der Variablenordnung $<$ auf X) ist ein Tupel $D = (V, \text{out}_0, \text{out}_1, r, \lambda)$ mit folgenden Eigenschaften:

- V ist eine endliche Menge von Knoten mit $0, 1 \in V$.
- $r \in V$ ist der Wurzelknoten.
- $\text{out}_0 : V \setminus \{0,1\} \to V$ und $\text{out}_1 : V \setminus \{0,1\} \to V$
- $\lambda : V \setminus \{0,1\} \to X$ ist die Knotenbeschriftungsfunktion.
- Für alle $v \in V \setminus \{0,1\}$ und $i \in \{0,1\}$ gilt:
  Wenn $\text{out}_i(v) \notin \{0,1\}$ dann $\lambda(v) < \lambda(\text{out}_i(v))$.

OBDD als Graph

Beachte: Sei $E_D := \{(v, \mathrm{out}_i(v)) \mid v \in V \setminus \{0,1\}, i \in \{0,1\}\}$.

- Der Graph (V, E_D) ist azyklisch.
- Die beiden Knoten $0, 1 \in V$ (die Senken) sind die einzigen Knoten ohne ausgehende Kanten in (V, E_D), und jeder andere Knoten hat genau zwei ausgehende Kanten.

Navigation in OBDD

Sei $D = (V, \text{out}_0, \text{out}_1, r, \lambda)$ ein OBDD bzgl. der Variablenordnung $<$ auf der Variablenmenge $X = \{x_1, \ldots, x_n\}$ mit $x_1 < x_2 < \cdots < x_n$.

Für $s = a_1 a_2 \cdots a_k \in \{0,1\}^k$ $(0 \leq k \leq n)$ und $v \in V$ definieren wir den Knoten v^s induktiv wie folgt:

$$v^s = \begin{cases} v & \text{falls } v \in \{0,1\} \text{ oder } \lambda(v) = x_i \text{ und } i > k \\ (\text{out}_{a_i}(v))^s & \text{falls } \lambda(v) = x_i \text{ und } i \leq k \end{cases}$$

Intuition: v^s für $s = a_1 a_2 \cdots a_k$ berechnet sich wie folgt.

- Starte im Knoten v.
- Interpretiere a_i als den Wert der Variablen x_i und steige im OBDD nach folgender Regel ab: Wenn wir aktuell im Knoten u sind, $\lambda(u) = x_i$ und $i \leq k$, dann folge der mit a_i beschrifteten Kante.
- Wiederhole dies so lange wie möglich, d.h. bis der aktuelle Knoten u zu $\{0,1\}$ gehört, oder $\lambda(u) = x_i$ mit $i > k$ gilt.

Durch einen OBDD berechnete Boolesche Funktion

Definition (Boolesche Funktion f_D)

D definiert eine Boolesche Funktion $f_D : \{0,1\}^X \to \{0,1\}$ wie folgt:

$$\forall \beta \in \{0,1\}^X : f_D(\beta) = r^{\beta(x_1)\beta(x_2)\cdots\beta(x_n)}$$

Alternative Definition der Funktion f_D:

Definition (die im Knoten v berechnete Funktion f_v)

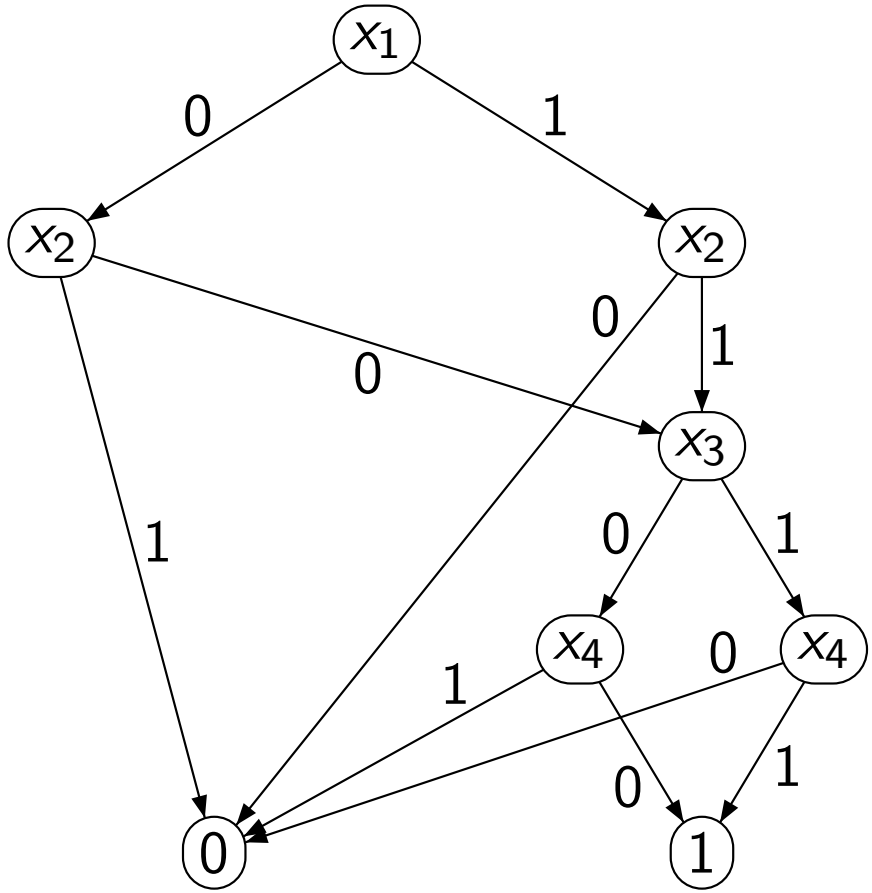
Wir ordnen jedem Knoten $v \in V$ eine Boolesche Funktion (die im Knoten v berechnete Funktion) $f_v : \{0,1\}^X \to \{0,1\}$ zu:

- $f_0 = 0$ und $f_1 = 1$
- Sei $v \in V \setminus \{0,1\}$ mit $\lambda(v) = x_j$ und $v_i = \text{out}_i(v)$, $i \in \{0,1\}$.

  Dann gilt $f_v = (\neg x_j \wedge f_{v_0}) \vee (x_j \wedge f_{v_1})$.
- Schließlich ist $f_D = f_r$

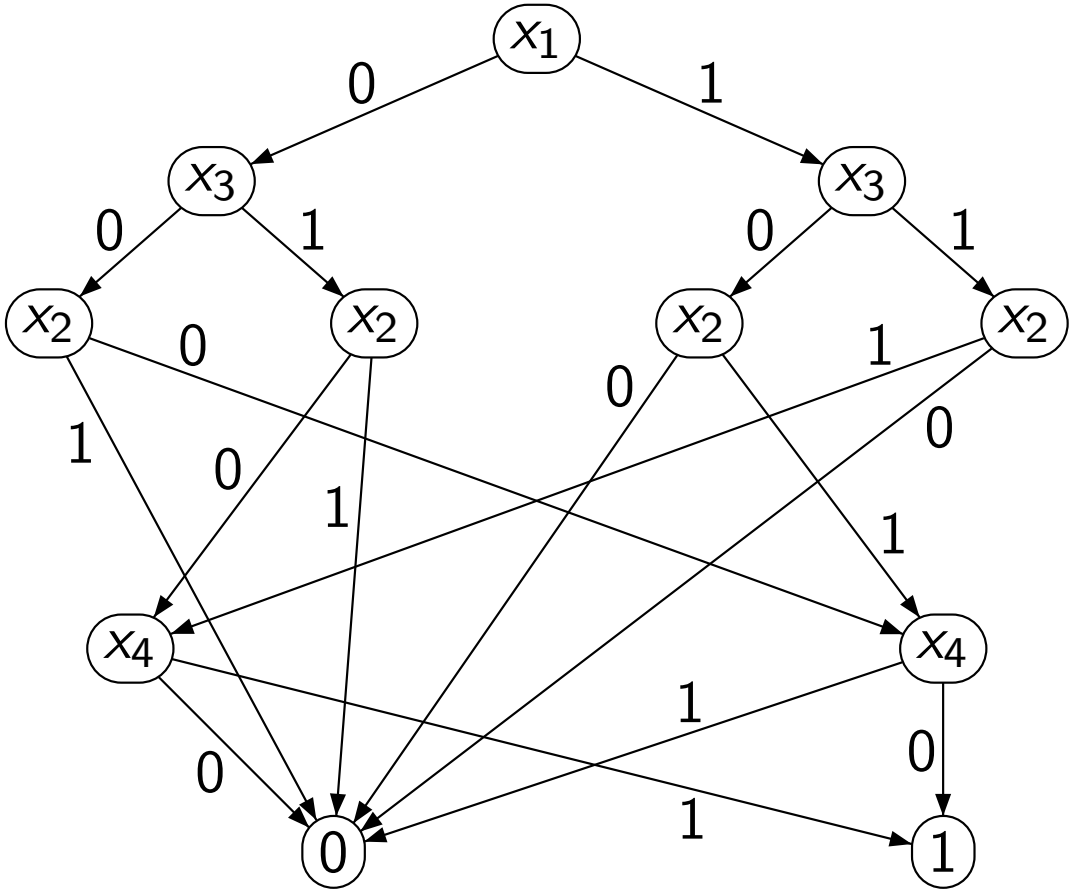
OBDDs: Beispiel 1

Ein OBDD für die Boolesche Funktion $(x_1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4)$:



OBDDs: Beispiel 2

Ein weiterer OBDD für die gleiche Boolesche Funktion:



Isomorphie von OBDDs

Definition (Isomorphie von OBDDs)

Zwei OBDDs $D = (V, \text{out}_0, \text{out}_1, r, \lambda)$ und $D' = (V', \text{out}'_0, \text{out}'_1, r', \lambda')$ sind isomorph, falls eine Bijektion $h : V \to V'$ mit folgenden Eigenschaften existiert:

- $h(r) = r'$, $h(0) = 0$, $h(1) = 1$,
- $\lambda'(h(v)) = \lambda(v)$ für alle $v \in V \setminus \{0, 1\}$,
- $\text{out}'_a(h(v)) = h(\text{out}_a(v))$ für alle $v \in V \setminus \{0, 1\}$, $a \in \{0, 1\}$.

Die Abbildung h ist dann ein Isomorphismus zwischen D und D'.

Dies bedeutet, dass D' aus D durch Umbenennung der Knoten gebildet werden kann.

Beachte: D und D' sind bezüglich der gleichen Variablenordnung definiert.

Restriktion von Booleschen Funktionen

Das weitere Ziel:

- Für jede Boolesche Funktion f und jede Variablenordnung existiert ein bis auf Isomorphie eindeutig bestimmter minimaler OBDD D_f für f.
- Aus einem beliebigen OBDD D für f kann D_f effizient durch Anwendung zweier lokaler Reduktionsregeln berechnet werden.

Fixiere die endliche Variablenmenge $X = \{x_1, \ldots, x_n\}$.

Definition (Restriktion einer Belegung)

Für eine Belegung $\beta : X \to \{0,1\}$, $1 \leq i \leq n$ und $a \in \{0,1\}$ sei $\beta[x_i = a] : X \to \{0,1\}$ die Belegung mit

$$\beta[x_i = a](x_j) = \begin{cases} a & \text{falls } i = j \\ \beta(x_j) & \text{sonst} \end{cases}$$

Restriktion von Booleschen Funktionen

Sei $f : \{0,1\}^X \to \{0,1\}$ eine Boolesche Funktion mit $X = \{x_1, \ldots, x_n\}$.

Definition (Restriktion einer Booleschen Funktion)

Für $1 \leq i \leq n$ und $a \in \{0,1\}$ ist $f[x_i = a] : \{0,1\}^X \to \{0,1\}$ die Boolesche Funktion mit:

$$\forall \beta \in \{0,1\}^X : f[x_i = a](\beta) = f(\beta[x_i = a]).$$

Intuition: Die Variable x_i wird auf a festgesetzt.

Beispiel: Sei f die durch den Booleschen Ausdruck $(x_1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4)$ berechnete Funktion über der Variablenmenge $\{x_1, x_2, x_3, x_4\}$.

Dann wird $f[x_2 = 0]$ durch den Ausdruck $\neg x_1 \wedge (x_3 \leftrightarrow x_4)$ berechnet.

Beachte: Wir betrachten $f[x_i = a]$ noch immer als eine Boolesche Funktion über der gesamten Variablenmenge X, obwohl sie von x_i nicht mehr abhängt.

Shannon-Expansion

Lemma 28 (Shannon's Expansionsgesetz)

Für jede Boolesche Funktion $f : \{0,1\}^X \to \{0,1\}$ *und jede Variable* $x_i \in X$ *gilt:*

$$f = (\neg x_i \wedge f[x_i = 0]) \vee (x_i \wedge f[x_i = 1])$$

Beweis: Sei $\beta : X \to \{0,1\}$ eine Belegung.

Fall 1. $\beta(x_i) = 0$: Dann gilt:

$$\begin{aligned}
(\neg x_i \wedge f[x_i = 0]) \vee (x_i \wedge f[x_i = 1])(\beta) &= \\
(1 \wedge f[x_i = 0](\beta)) \vee (0 \wedge f[x_i = 1](\beta)) &= \\
f[x_i = 0](\beta) &= \\
f(\beta[x_i = 0]) &= \\
f(\beta)
\end{aligned}$$

Fall 2. $\beta(x_i) = 1$: Analog □

Variablenabhängigkeiten

Definition (Abhängigkeit von einer Variablen x_i)

Die Boolesche Funktion f hängt von x_i ab, falls gilt: $f[x_i = 0] \neq f[x_i = 1]$.

Beobachtungen:

- $f[x_i = b]$ hängt nicht von x_i ab.
- Sei $D = (V, \text{out}_0, \text{out}_1, r, \lambda)$ ein OBDD bzgl. der Variablenordnung $x_1 < x_2 < \cdots < x_n$. Sei $v \in V$ ein Knoten mit $\lambda(v) = x_j$, $j > i$.

  Dann hängt die im Knoten v berechnete Funktion f_v nicht von x_i ab.

Variablenabhängigkeiten

Lemma 29

Folgende Aussagen sind äquivalent für jede Boolesche Funktion f:

- $f[x_i = 0] = f[x_i = 1]$
- $f = f[x_i = 0]$
- $f = f[x_i = 1]$

Beweis:

Aus $f = f[x_i = 0]$ folgt:

$$\begin{aligned} f[x_i = 1](\beta) &= f(\beta[x_i = 1]) = f[x_i = 0](\beta[x_i = 1]) \\ &= f(\beta[x_i = 1][x_i = 0]) = f(\beta[x_i = 0]) = f[x_i = 0](\beta). \end{aligned}$$

Aus $f[x_i = 0] = f[x_i = 1]$ folgt:

$$f(\beta) = f(\beta[x_i = \beta(x_i)]) = f[x_i = \beta(x_i)](\beta) = f[x_i = 0](\beta).$$

Restriktion und OBDDs

Analog kann gezeigt werden: $f = f[x_i = 1] \Leftrightarrow f[x_i = 0] = f[x_i = 1]$. □

Insbesondere: $f = f[x_i = 0] = f[x_i = 1]$ falls f nicht von x_i abhängt.

Lemma 30

Sei $D = (V, \text{out}_0, \text{out}_1, r, \lambda)$ ein OBDD, $v \in V$, $\lambda(v) = x_i$, $\text{out}_0(v) = v_0$ und $\text{out}_1(v) = v_1$.

Dann gilt für die in den Knoten v, v_0 und v_1 berechneten Funktionen: $f_{v_0} = f_v[x_i = 0]$ und $f_{v_1} = f_v[x_i = 1]$.

Beweis: Es gilt $f_v = (\neg x_i \wedge f_{v_0}) \vee (x_i \wedge f_{v_1})$.

Also gilt:

- $f_v[x_i = 0] = (\neg 0 \wedge f_{v_0}) \vee (0 \wedge f_{v_1}) = f_{v_0}$.
- $f_v[x_i = 1] = (\neg 1 \wedge f_{v_0}) \vee (1 \wedge f_{v_1}) = f_{v_1}$. □

Restriktion und OBDDs

Sei nun $x_1 < x_2 < \cdots < x_n$ die Variablenordnung auf $X = \{x_1, \ldots, x_n\}$.

Für $s = a_1 a_2 \cdots a_i \in \{0,1\}^i$ $(0 \leq i \leq n)$ sei $f^s : \{0,1\}^X \to \{0,1\}$ die Boolesche Funktion $f^s = f[x_1 = a_1][x_2 = a_2] \cdots [x_i = a_i]$.

Lemma 31

Sei $D = (V, \text{out}_0, \text{out}_1, r, \lambda)$ ein OBDD bzgl. der Variablenordnung $<$ auf $X = \{x_1, \ldots, x_n\}$ mit $x_1 < x_2 < \cdots < x_n$.

Sei $s = a_1 a_2 \cdots a_i \in \{0,1\}^i$ $(0 \leq i \leq n)$ und $v = r^s$.

Dann ist f_D^s die im Knoten v berechnete Funktion f_v.

Restriktion und OBDDs

Beweis: Induktion über i.

Induktionsanfang $i = 0$: $f_D^\varepsilon = f_D = f_r$ und $r = r^\varepsilon$.

Induktionsschritt: Sei $s = s'a$ mit $a \in \{0,1\}$, $|s'| = i - 1$ und $v' = r^{s'}$.

Nach Induktion gilt $f_D^s = f_D^{s'a} = f_D^{s'}[x_i = a] = f_{v'}[x_i = a]$.

1. Fall: $\lambda(v') = x_i$.

Dann gilt $v = r^{s'a} = (v')^a = \mathrm{out}_a(v')$ und $f_{v'}[x_i = a] = f_v$ wegen Lemma 30.

2. Fall: $v' \in \{0,1\}$ oder $\lambda(v') = x_j$ mit $j > i$.

Dann gilt $r^s = v'$.

Ausserdem hängt $f_{v'}$ nicht von x_i ab.

Aus Lemma 29 folgt $f_{v'}[x_i = a] = f_{v'}$. □

Reduktionsregeln

Seien $D = (V, \text{out}_0, \text{out}_1, r, \lambda)$ und $D' = (V', \text{out}'_0, \text{out}'_1, r', \lambda')$ OBDDs bzgl. der gleichen Variablenordnung.

Wir schreiben $D \to D'$ falls einer der beiden folgenden Fälle gilt:

- 1. Reduktionsregel:
  Es gibt Knoten $v, v' \in V$ mit $\text{out}_0(v) = v' = \text{out}_1(v)$ und D' entsteht durch Verschmelzen von v und v'.

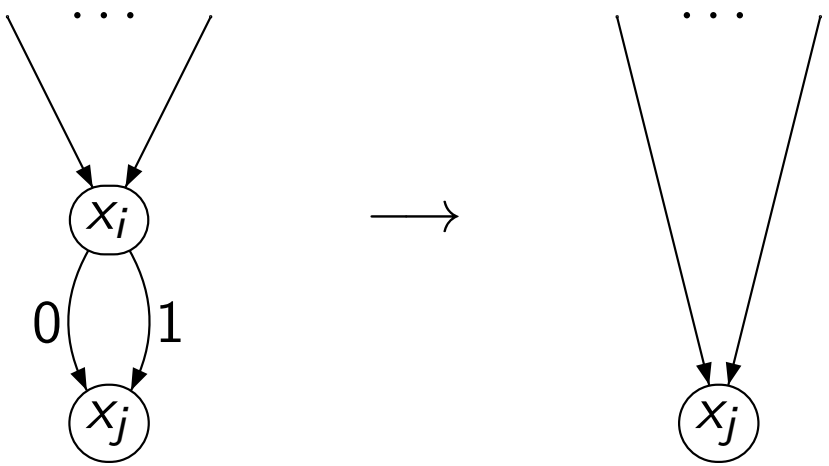
  Formal: $V' = V \setminus \{v\}$, $r' = r$ falls $r \in V'$, $r' = v'$ falls $r = v$, und für alle $u \in V'$:

$$\lambda'(u) = \lambda(u) \quad \text{und} \quad \text{out}'_a(u) = \begin{cases} \text{out}_a(u) & \text{falls } \text{out}_a(u) \neq v \\ v' & \text{falls } \text{out}_a(u) = v \end{cases}$$

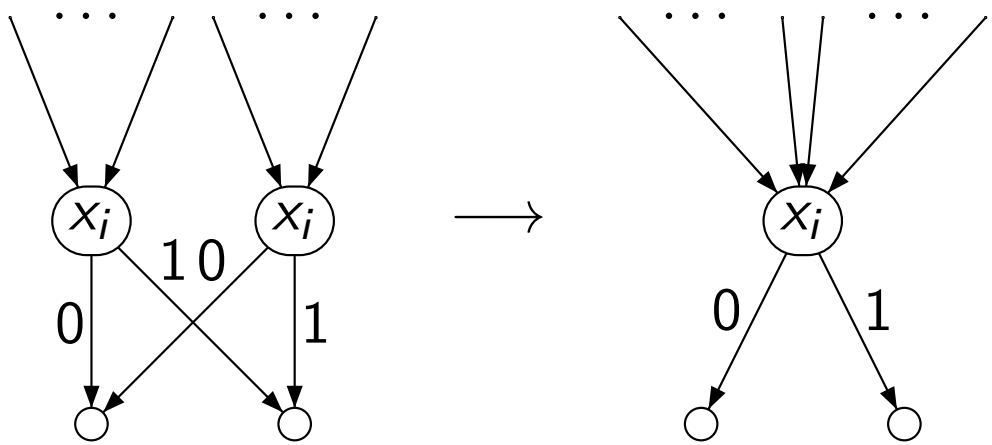
- 2. Reduktionsregel:
  Es gibt $v, v' \in V$ mit $v \neq v'$, $\lambda(v) = \lambda(v')$, $\text{out}_0(v) = \text{out}_0(v')$, $\text{out}_1(v) = \text{out}_1(v')$, und D' entsteht durch Verschmelzen von v und v' (wie oben).

Graphische Darstelllung der Reduktionsregeln

1. Reduktionsregel:



2. Reduktionsregel:



Reduktionsregeln erhalten die berechnete Funktion

Lemma 32

Gelte $D \to D'$ für OBDDs D und D'. Dann gilt $f_D = f_{D'}$.

Beweis:

Angenommen die Knoten v und v' von D werden bei der Reduktion $D \to D'$ verschmolzen.

Es genügt zu zeigen: $f_v = f_{v'}$.

Wird die zweite Reduktionsregel angewendet, so ist dies klar.

Bei Anwendung der ersten Reduktionsregel gilt $\text{out}_0(v) = v' = \text{out}_1(v)$.

Sei $\lambda(v) = x_i$.

Aus Lemma 30 folgt $f_v[x_i = 0] = f_{v'} = f_v[x_i = 1]$.

Aus Lemma 29 folgt schließlich $f_v = f_{v'}$. □

Der minimale OBDD

Im folgenden sei $|D|$ die Anzahl der Knoten des OBDDs D.

Definition (minimaler OBDD)

Sei $<$ eine Variablenordnung auf X. Ein OBDD D bzgl. der Variablenordnung $<$ ist minimal, falls für jeden OBDD D' bzgl. der gleichen Variablenordnung $<$ gilt: Wenn $f_D = f_{D'}$ dann $|D| \leq |D'|$.

Beachte: Wir definieren minimale OBDDs für eine feste Variablenordnung.

Sind D_1 und D_2 minimale OBDDs für die gleiche Funktion f aber bzgl. verschiedener Variablenordnungen, so können D_1 und D_2 unterschiedlich viele Knoten haben.

Beispiel: Die beiden OBDDs auf den Folien 157 und 158 sind beide minimal für die Funktion $(x_1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4)$.

Der minimale OBDD

Wir bezeichnen im weiteren die konstante 0-Funktion $f : \{0,1\}^X \to \{0,1\}$ (d.h. $f(\beta) = 0$ für alle Belegungen β) mit 0, und ebenso für die konstante 1-Funktion.

Der minimale OBDD für 0 sowie 1 ist offensichtlich der OBDD, der nur aus den Knoten 0 und 1 besteht.

Im weiteren gehen wir stets davon aus, dass für die Boolesche Funktion $f : \{0,1\}^X \to \{0,1\}$ gilt: $0 \neq f \neq 1$.

Dies bedeutet, dass f von wenigstens einer Variable abhängt.

Außerdem betrachten wir im folgenden nur OBDDs D, wo jeder Knoten von der Wurzel r erreichbar ist (mittels der Kantenrelation E_D).

Der minimale OBDD

Definition (OBDD D_f)

Sei $f : \{0,1\}^X \to \{0,1\}$ eine Boolesche Funktion, $X = \{x_1, \ldots, x_n\}$, $x_1 < x_2 < \cdots < x_n$.

Wir definieren den OBDD $D_f = (V_f, \text{out}_0, \text{out}_1, r, \lambda)$ bzgl. der Variablenordnung $x_1 < x_2 < \cdots < x_n$ wie folgt:

- $V_f = \{f^s \mid s \in \{0,1\}^*, |s| \leq n\}$
- $r = f$
- $\lambda(g) = x_i$ falls i der kleinste Index ist, so dass g von x_i abhängt.
- $\text{out}_a(g) = g[x_i = a]$, falls $g \in V_f$, $\lambda(g) = x_i$ und $a \in \{0,1\}$.

Beachte: Wegen $0 \neq f \neq 1$, gilt $0, 1 \in V_f$.

Der minimale OBDD

Lemma 33

$f_{D_f} = f$, d.h. D_f ist ein OBDD für f.

Beweis: Sei $g \in V_f$ ein Knoten des OBDD D_f.

Wir zeigen mit Induktion, dass die Funktion g die im Knoten g berechnete Boolesche Funktion ist.

1. Fall: $g = 0$ oder $g = 1$. Klar

2. Fall: $\lambda(g) = x_i$ für ein $1 \leq i \leq n$.

Es gilt $g[x_i = 0] = \text{out}_0(g)$ und $g[x_i = 1] = \text{out}_1(g)$.

Mit Induktion folgt, dass $g[x_i = a]$ die im Knoten $g[x_i = a]$ berechnete Funktion ist.

Also wird die Funktion $(\neg x_i \wedge g[x_i = 0]) \vee (x_i \wedge g[x_i = 1]) = g$ (siehe Lemma 28) im Knoten g berechnet. □

Der minimale OBDD

Lemma 34

D_f ist ein minimaler OBDD bzgl. der Variablenordnung $<$ für f.

Beweis:

Sei D ein beliebiger OBDD bzgl. der Variablenordnung $<$ für f.

Sei V die Knotenmenge von D.

Aus Lemma 31 folgt, dass jede der Funktionen f^s $(s \in \{0,1\}^*, |s| \leq n)$ in einem Knoten aus V berechnet wird.

Also gilt $|V| \geq |V_f|$. □

Der minimale OBDD

Lemma 35

Ein OBDD D für die Funktion f ist genau dann nicht minimal, wenn Knoten u, v mit $u \neq v$ und $f_u = f_v$ existieren.

Beweis:

Sei V die Knotenmenge von D.

Angenommen es gilt $f_u \neq f_v$ für alle Knoten $u, v \in V$ mit $u \neq v$.

Aus Lemma 31 folgt, dass jede der Funktionen $f^s \in V_f$ in genau einem Knoten von D berechnet wird.

Also gilt $|V| = |V_f|$ und D ist minimal (da D_f minimal ist).

Falls es Knoten $u, v \in V$ mit $u \neq v$ und $f_u = f_v$ gibt, muss wieder wegen Lemma 31 $|V| > |V_f|$ gelten,

Also ist D nicht minimal. □

Der minimale OBDD

Lemma 36

Ein OBDD D ist minimal (bzgl. der Variablenordnung $<$) genau dann, wenn keine der beiden Reduktionsregeln auf D anwendbar ist.

Beweis:

Wenn auf D eine Reduktionsregel anwendbar ist, ist D nach Lemma 32 nicht minimal.

Sei nun D nicht minimal.

Wegen Lemma 35 gibt es Knoten $u \neq v$ mit $f_u = f_v$.

Wir berechnen nun eine Folge von Knotenpaaren, wobei für das aktuelle Knotenpaar (u', v') stets $u' \neq v'$ und $f_{u'} = f_{v'}$ gilt.

Der minimale OBDD

Wir starten mit dem Knotenpaar (u, v).

Fall 1: $\lambda(u) = x_i = \lambda(v)$.

Sei $u_i = \text{out}_i(u)$ und $v_i = \text{out}_i(v)$ für $i \in \{0, 1\}$.

Dann gilt $f_{u_0} = f_{v_0}$ und $f_{u_1} = f_{v_1}$.

Falls $u_0 = v_0$ und $u_1 = v_1$ gilt, können wir die 2. Reduktionsregel anwenden, und wir sind fertig.

Ansonsten gilt $u_i \neq v_i$ für ein $i \in \{0, 1\}$.

Wir machen dann mit dem Paar (u_i, v_i) weiter.

Der minimale OBDD

Fall 2: $\lambda(u) = x_i < x_j = \lambda(v)$ oder $(\lambda(u) = x_i$ und $v \in \{0,1\})$.

Dann hängt f_u nicht von x_i ab.

Sei $u_i = \text{out}_i(u)$ für $i \in \{0,1\}$.

Dann gilt $f_{u_0} = f_{u_1}$.

Falls $u_0 = u_1$ gilt, können wir die 1. Reduktionsregel anwenden, und wir sind fertig.

Falls $u_1 \neq u_2$ machen wir mit dem Paar (u_1, u_2) weiter.

Obiger Prozess muss mit der Anwendung einer Reduktionsregel terminieren:

Sei (u', v') das aktuelle Paar und sei $\lambda(u') = x_i$ und $\lambda(v') = x_j$.

Dann wird in jedem Schritt $\min\{x_i, x_j\}$ echt größer. □

Der minimale OBDD

Lemma 37

Jeder minimale OBDD für f bzgl. der Variablenordnung $<$ ist isomorph zu dem OBDD D_f.

Beweis:

Sei $D' = (V', \text{out}'_0, \text{out}'_1, r', \lambda')$ ein OBDD für f mit $|V_f|$ vielen Knoten.

Dann gibt es für jedes $g \in V_f$ genau einen Knoten $v_g \in V'$, so dass g im Knoten v_g berechnet wird, d.h. $h : g \mapsto v_g$ is bijektiv.

Wir zeigen, dass h ein Isomorphismus zwischen D_f und D' ist.

Offensichtlich gilt $h(f) = v_f = r'$, $h(0) = 0$ und $h(1) = 1$.

Angenommen es gilt nun $\lambda(g) = x_i$, d.h. i ist der kleinste Index, so dass g von x_i abhängt.

Der minimale OBDD

Wir zeigen:

- $\lambda'(h(g)) = \lambda'(v_g) = x_i$.
- $\text{out}'_a(h(g)) = \text{out}'_a(v_g) = v_{g[x_i=a]} = h(\text{out}_a(g))$ für $a \in \{0,1\}$

Gelte $\lambda'(v_g) = x_j$.

Da D' minimal ist, hängt g von x_j ab (sonst wäre die erste Reduktionsregel anwendbar).

Also gilt $j \geq i$.

Würde $j > i$ gelten, so würde g nicht von x_i abhängen – ein Widerspruch.

Also gilt $\lambda'(v_g) = x_i$.

Sei nun $a \in \{0,1\}$.

Die im Knoten $\text{out}'_a(v_g)$ berechnete Funktion ist $g[x_i = a]$ nach Lemma 30.

Also gilt $\text{out}'_a(v_g) = v_{g[x_i=a]}$. □

Der minimale OBDD

Lemma 38

Sei D ein OBDD mit $f_D = f$. Durch iteriertes Anwenden der beiden Reduktionsregeln (in beliebiger Reihenfolge) erhalten wir schließlich einen zu D_f isomorphen OBDD.

Beweis: Sei D ein OBDD für f.

Sei D' ein OBDD der durch wiederholtes Anwenden der beiden Reduktionsregeln aus D entsteht, und so dass auf D' keine der beiden Regeln angewendet werden kann.

Aus Lemma 32 folgt $f_{D'} = f$.

Ausserdem ist D' nach Lemma 36 minimal.

Wegen Lemma 37 ist D' isomorph zu D_f. □

Der minimale OBDD

Aus den Lemmata 33, 34, 37 und 38 folgt nun:

Satz 39

Für jede Boolesche Funktion f und jede feste Variablenordnung $<$ gibt es einen bis auf Isomorphie eindeutig bestimmten minimalen OBDD D_f bzgl. $<$. Diesen erhält man aus einem beliebigen OBDD für f durch wiederholtes Anwenden der beiden Reduktionsregeln.

Für einen OBDD D kann man den Reduktionsalgorithmus (welcher die Reduktionsregeln solange wie möglich anwendet) in Zeit $O(|D|)$ implementieren.

Hierzu geht man bottom-up vor:

Ist $x_1 < x_2 < \cdots < x_n$ die Variablenordnung.

Dann wendet man zunächst Reduktionsregeln auf mit x_n beschriftete Knoten an, gefolgt von mit x_{n-1} beschrifteten Knoten, u.s.w.

Größe des minimalen OBDD

Satz 40 (Liaw, Lin 1992)

Sei $X = \{x_1, \ldots, x_n\}$ und $<$ eine beliebige Variablenordnung auf X. Für jede Boolesche Funktion $f : \{0,1\}^X \to \{0,1\}$ existiert ein OBDD D bzgl. $<$ mit $f_D = f$ und $|D| = (2 + \varepsilon_n) \cdot \frac{2^n}{n}$, wobei $\varepsilon_n \to 0$ für $n \to \infty$.

Für einen Bitstring $s = a_0 a_1 \cdots a_{n-1}$ sei

$$(a_0 a_1 \cdots a_{n-1})_2 = \sum_{i=0}^{n-1} a_i \cdot 2^i$$

die durch s repräsentierte Zahl.

Untere Schranke für die Multiplikation

Definition (mittlere Bit der Multiplikationsfunktion)

Sei f_{mult} die folgende Boolesche Funktion auf den $2n$ vielen Variablen $x_0, \ldots, x_{2n-1}$:

$f_{\text{mult}}(x_0, \ldots, x_{n-1}, x_n, \ldots, x_{2n-1})$ ist das Bit an Position $n-1$ in dem Produkt $(x_0 \cdots x_{n-1})_2 \cdot (x_n \cdots x_{2n-1})_2$.

Satz 41 (Bryant 1991)

Sei $<$ eine beliebige Variablenordnung auf $\{x_0, \ldots, x_{2n-1}\}$. Jeder OBDD für f_{mult} bzgl. der Variablenordnung $<$ hat mindestens $2^{(n/8)}$ viele Knoten.

Äquivalenz von OBDDs

Satz 42

Für OBDDs D, D' (bzgl. der gleichen Variablenordnung) kann in Zeit $O(|D| + |D'|)$ entschieden werden, ob $f_D = f_{D'}$ gilt.

Beweis: Reduziere D und D' so lange wie möglich.

Seien $\hat{D}$ und $\hat{D}'$ die resultierenden OBBDs.

Teste nun, ob $\hat{D}$ und $\hat{D}'$ isomorph sind. □

Bemerkung: Für Boolesche Funktionen f und g, die durch Boolesche Ausdrücke (über $\neg$, $\wedge$ und $\vee$) gegeben sind, ist es recht schwierig (**coNP**-vollständig) zu testen, ob $f = g$ gilt.

Boolesche Operationen auf OBDDs

Lemma 43

Sei D ein OBDD. In Zeit $O(|D|)$ kann ein OBDD für die Boolesche Funktion $\neg f_D$ berechnet werden.

Beweis: Vertausche die Rollen des 0-Knotens und des 1-Knotens.

Lemma 44

Seien D_1 und D_2 OBDDs über der gleichen Variablenmenge X und bzgl. der gleichen Variablenordnung.

In Zeit $O(|D_1| \cdot |D_2|)$ können OBDDs für die Boolesche Funktionen $f_{D_1} \wedge f_{D_2}$ und $f_{D_1} \vee f_{D_2}$ berechnet werden.

Boolesche Operationen auf OBDDs

Beweis:

Sei $D_i = (V_i, \text{out}_{i,0}, \text{out}_{i,1}, r_i, \lambda_i)$.

Sei $\circ$ eine beliebige binäre Boolesche Operation (z.B. $\wedge$ oder $\vee$).

Nach Shannon's Expansionsgesetz (Lemma 28) gilt für alle Booleschen Funktionen $f, g : \{0,1\}^X \to \{0,1\}$ und alle Variablen $x_i \in X$:

$$f \circ g = (\neg x_i \wedge (f[x_i = 0] \circ g[x_i = 0])) \vee (x_i \wedge (f[x_i = 1] \circ g[x_i = 1])) \quad (1)$$

Dies motiviert den folgenden Algorithmus apply$(u, v, \circ)$ zur Erzeugung eines OBDDs D für $f_{D_1} \circ f_{D_2}$, wobei $u \in V_1$ und $v \in V_2$.

Boolesche Operationen auf OBDDs

Algorithmus apply

procedure apply$(u, v, \circ)$
begin
  if Knoten (u, v) existiert bereits **then**
    exit
  endif
  erzeuge Knoten (u, v);
  if $\lambda_1(u) = x_i = \lambda_2(v)$ **then**
    $\lambda((u, v)) := x_i$;
    let $u_0 = \text{out}_{1,0}(u)$; $u_1 = \text{out}_{1,1}(u)$; $v_0 = \text{out}_{2,0}(v)$; $v_1 = \text{out}_{2,1}(v)$
    apply$(u_0, v_0, \circ)$;
    apply$(u_1, v_1, \circ)$;
    $\text{out}_0((u, v)) := (u_0, v_0)$;
    $\text{out}_1((u, v)) := (u_1, v_1)$;
  endif

Boolesche Operationen auf OBDDs

Algorithmus apply (Fortsetzung)

if $\lambda_1(u) = x_i$ und $(\lambda_2(v) = x_j > x_i$ oder $v \in \{0,1\})$ **then**
  $\lambda((u,v)) := x_i$;
  let $u_0 = \text{out}_{1,0}(u)$; $u_1 = \text{out}_{1,1}(u)$;
  apply$(u_0, v, \circ)$;
  apply$(u_1, v, \circ)$;
  $\text{out}_0((u,v)) := (u_0, v)$;
  $\text{out}_1((u,v)) := (u_1, v)$;
endif

Boolesche Operationen auf OBDDs

Algorithmus apply (Fortsetzung)

if $\lambda_2(v) = x_i$ und $(\lambda_1(u) = x_j > x_i$ oder $u \in \{0,1\})$ **then**
  $\lambda((u,v)) := x_i$;
  let $v_0 = \text{out}_{2,0}(v)$; $v_1 = \text{out}_{2,1}(v)$;
  apply$(u, v_0, \circ)$;
  apply$(u, v_1, \circ)$;
  $\text{out}_0((u,v)) := (u, v_0)$;
  $\text{out}_1((u,v)) := (u, v_1)$;
endif

Boolesche Operationen auf OBDDs

Algorithmus apply (Fortsetzung)

if $u \in \{0,1\}$ und $v \in \{0,1\}$ **then**
  verschmelze Knoten (u, v) mit dem Knoten $u \circ v$
endif
endprocedure

Initialer Aufruf: apply$(r_1, r_2, \circ)$.

Korrektheit: Für jeden von apply erzeugten Knoten (u, v) gilt:

$$f_{(u,v)} = f_u \circ f_v.$$

Beweis: Induktion unter Verwendung von (1).

Boolesche Operationen auf OBDDs

Seien u_0, u_1, v_0, v_1 wie im apply-Algorithmus definiert.

Fall 1. $u \in \{0,1\}$ und $v \in \{0,1\}$: klar

Fall 2. $\lambda_1(u) = x_i = \lambda_2(v)$.

Es gilt (nK = nach Konstruktion, L = Lemma, IH = Induktionshypothese):

$$
\begin{aligned}
f_{(u,v)} &\overset{\text{nK}}{=} (\neg x_i \wedge f_{(u_0,v_0)}) \vee (x_i \wedge f_{(u_1,v_1)}) \\
&\overset{\text{IH}}{=} (\neg x_i \wedge (f_{u_0} \circ f_{v_0})) \vee (x_i \wedge (f_{u_1} \circ f_{v_1})) \\
&\overset{\text{L30}}{=} (\neg x_i \wedge (f_u[x_i = 0] \circ f_v[x_i = 0])) \vee (x_i \wedge f_u[x_i = 1] \circ f_v[x_i = 1])) \\
&\overset{(1)}{=} f_u \circ f_v
\end{aligned}
$$

Boolesche Operationen auf OBDDs

Fall 3. $\lambda_1(u) = x_i$ und $(\lambda_2(v) = x_j > x_i$ oder $v \in \{0,1\})$

Dann hängt f_v nicht von x_i ab.

Also gilt $f_v[x_i = 0] = f_v = f_v[x_i = 1]$ und damit:

$$\begin{aligned}
f_{(u,v)} &\overset{\text{nK}}{=} (\neg x_i \wedge f_{(u_0,v)}) \vee (x_i \wedge f_{(u_1,v)}) \\
&\overset{\text{IH}}{=} (\neg x_i \wedge (f_{u_0} \circ f_v)) \vee (x_i \wedge (f_{u_1} \circ f_v)) \\
&\overset{\text{L30}}{=} (\neg x_i \wedge (f_u[x_i = 0] \circ f_v)) \vee (x_i \wedge f_u[x_i = 1] \circ f_v)) \\
&= (\neg x_i \wedge (f_u[x_i = 0] \circ f_v[x_i = 0])) \vee (x_i \wedge f_u[x_i = 1] \circ f_v[x_i = 1])) \\
&\overset{(1)}{=} f_u \circ f_v
\end{aligned}$$

Fall 4. $\lambda_2(v) = x_i$ und $(\lambda_1(u) = x_j > x_i$ oder $u \in \{0,1\})$:
Analog zu Fall 3. □

Boolesche Operationen auf OBDDs

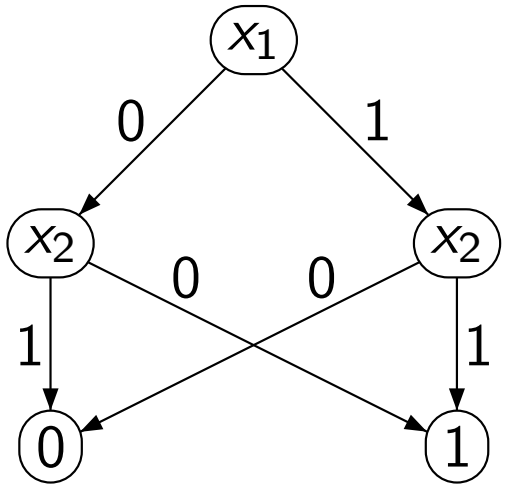
Beispiel: Sei $f_1 = (x_1 \leftrightarrow x_2)$ und $f_2 = (x_3 \leftrightarrow x_4)$, wobei wir f_1 und f_2 als Boolesche Funktionen über $\{x_1, x_2, x_3, x_4\}$ mit $x_1 < x_2 < x_3 < x_4$ betrachten:

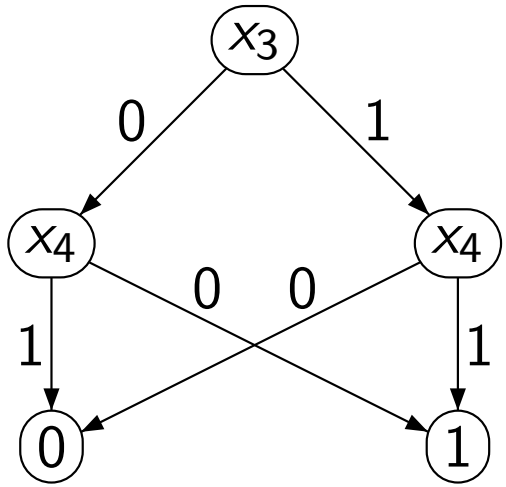
Hier sind OBDDs für f_1 und f_2:

Boolesche Operationen auf OBDDs

Beispiel: Sei $f_1 = (x_1 \leftrightarrow x_2)$ und $f_2 = (x_3 \leftrightarrow x_4)$, wobei wir f_1 und f_2 als Boolesche Funktionen über $\{x_1, x_2, x_3, x_4\}$ mit $x_1 < x_2 < x_3 < x_4$ betrachten:

Hier sind OBDDs für f_1 und f_2:

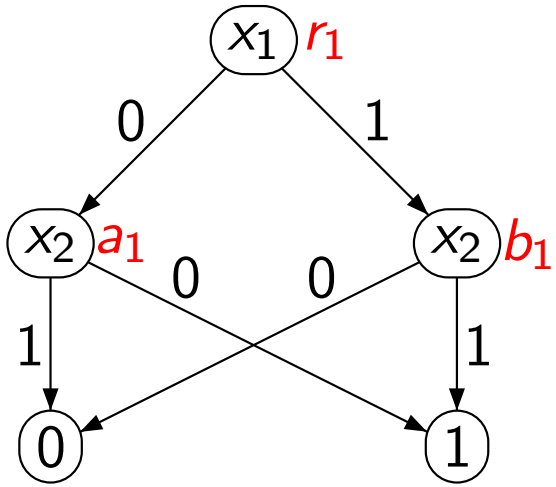


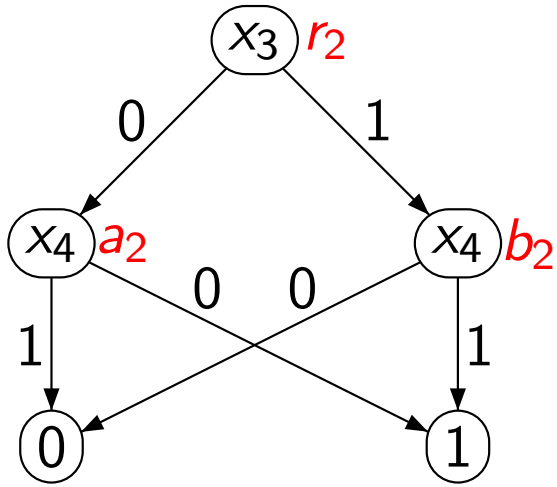


Boolesche Operationen auf OBDDs

Beispiel: Sei $f_1 = (x_1 \leftrightarrow x_2)$ und $f_2 = (x_3 \leftrightarrow x_4)$, wobei wir f_1 und f_2 als Boolesche Funktionen über $\{x_1, x_2, x_3, x_4\}$ mit $x_1 < x_2 < x_3 < x_4$ betrachten:

Hier sind OBDDs für f_1 und f_2:



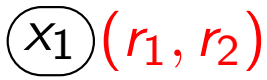


Boolesche Operationen auf OBDDs

Dann liefert apply den folgenden OBDD für $f_1 \wedge f_2$:

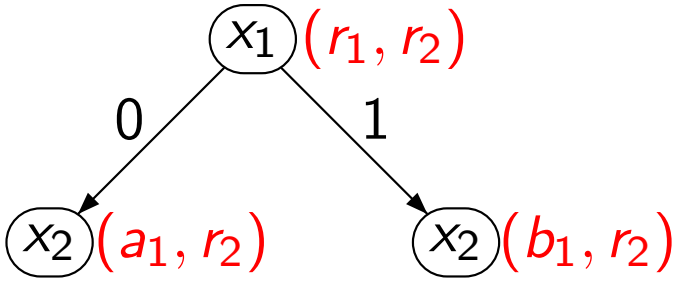
Boolesche Operationen auf OBDDs

Dann liefert apply den folgenden OBDD für $f_1 \wedge f_2$:



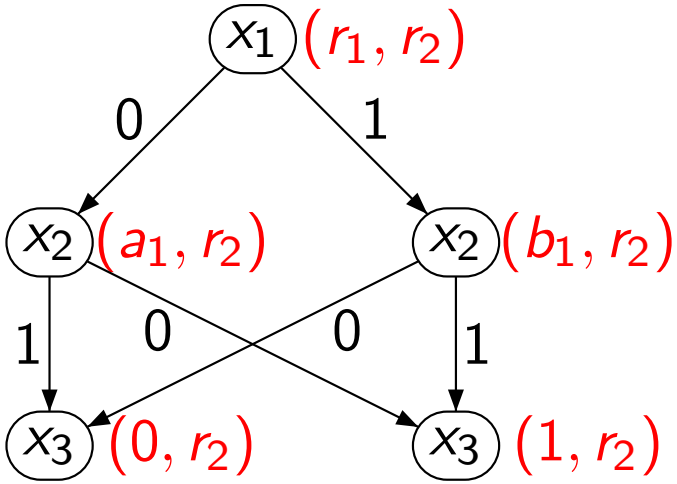
Boolesche Operationen auf OBDDs

Dann liefert apply den folgenden OBDD für $f_1 \wedge f_2$:



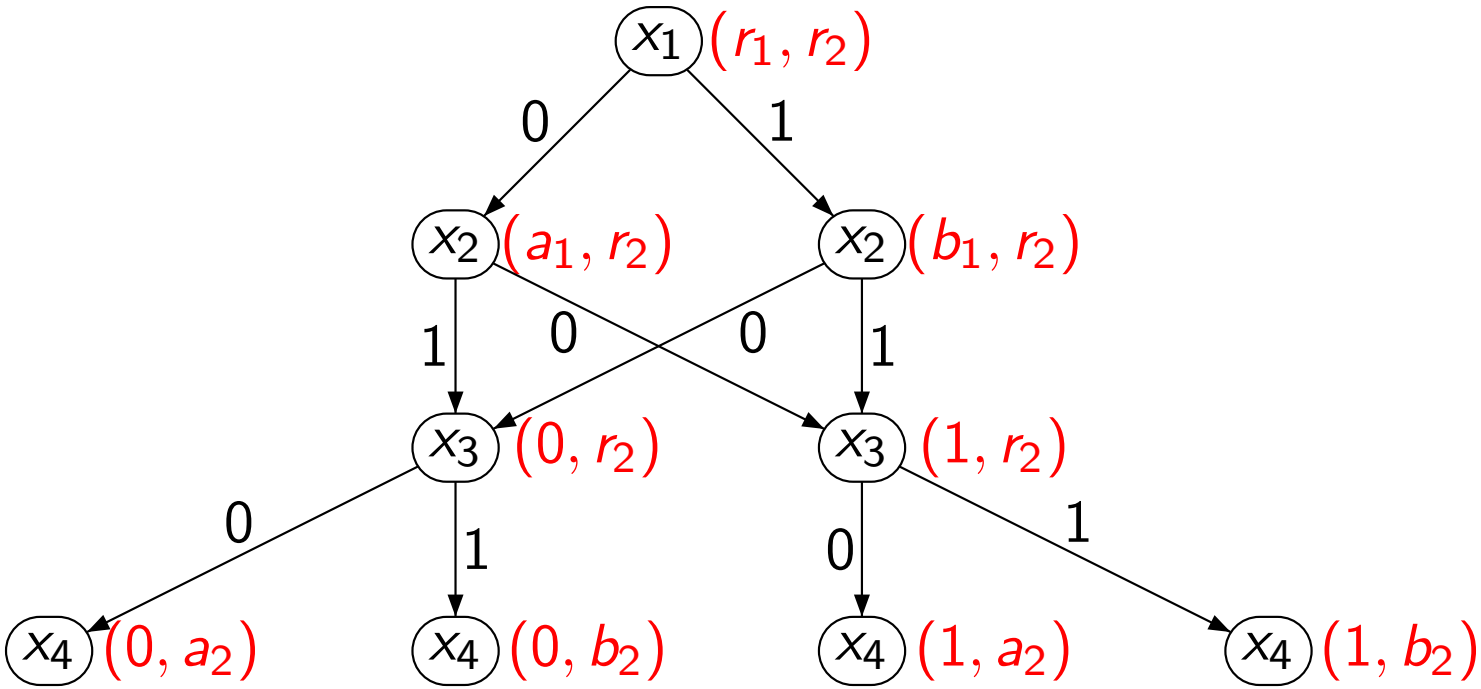
Boolesche Operationen auf OBDDs

Dann liefert apply den folgenden OBDD für $f_1 \wedge f_2$:



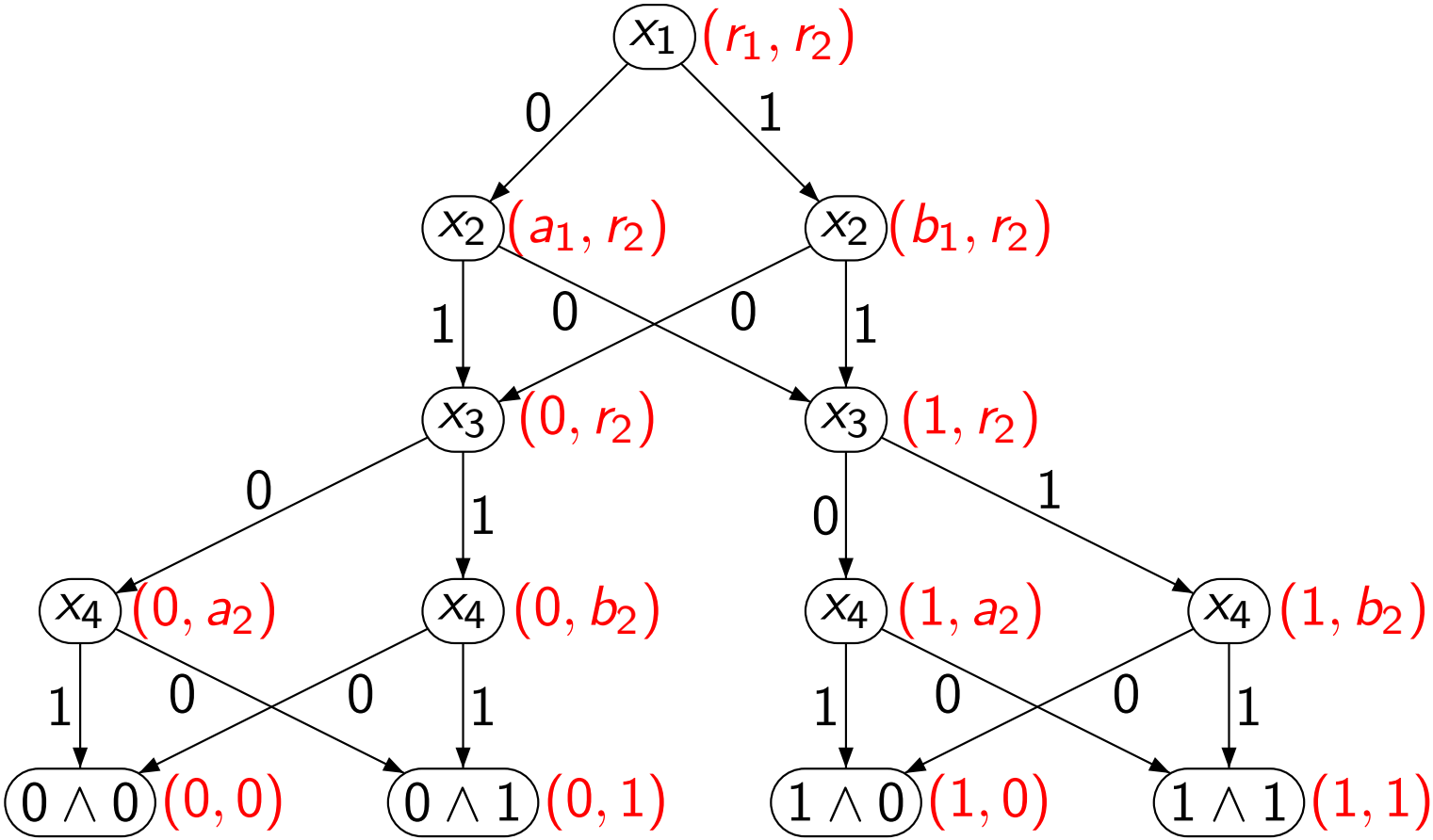
Boolesche Operationen auf OBDDs

Dann liefert apply den folgenden OBDD für $f_1 \wedge f_2$:



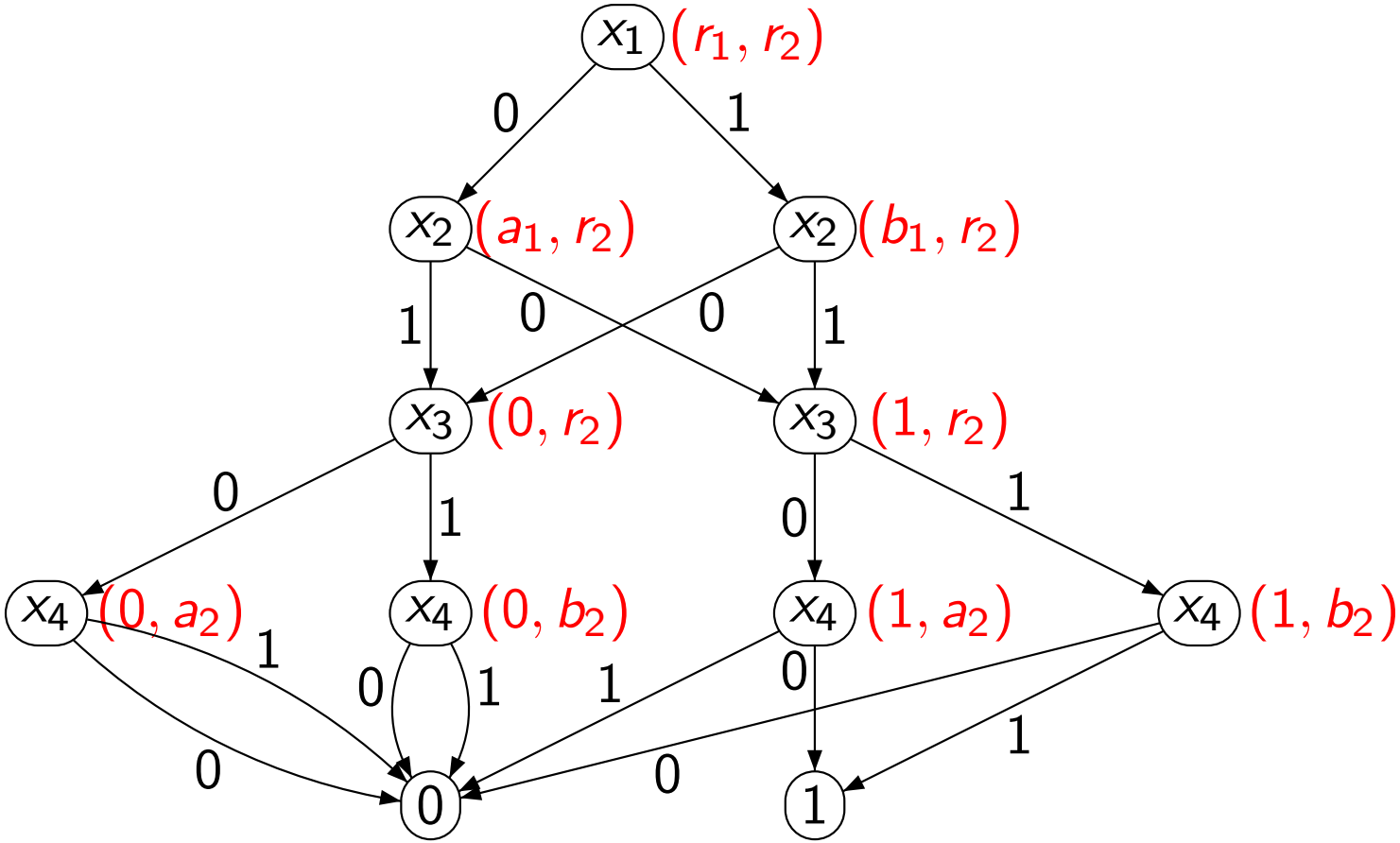
Boolesche Operationen auf OBDDs

Dann liefert apply den folgenden OBDD für $f_1 \wedge f_2$:



Boolesche Operationen auf OBDDs

Dann liefert apply den folgenden OBDD für $f_1 \wedge f_2$:



Boolesche Operationen auf OBDDs

Bemerkung: Mehrfache Anwendungen von apply führt zu i.A. sehr großen OBDDs.

Dies ist nicht überraschend: Erfüllbarkeit für eine boolesche Formel F (NP-vollständig) kann wie folgt gelöst werden:

- Konstruiere einen OBDD D für F unter Verwendung von apply.
- Teste Erfüllbarkeit, indem überprüft wird, ob in D ein Pfad von der Wurzel zu der 1-Senke führt.

Boolesche Operationen auf OBDDs

Definition (Existentielle Quantifizierung einer Booleschen Funktion)

Für eine Boolesche Funktion $f : \{0,1\}^X \to \{0,1\}$ und $x \in X$ ist $(\exists x.f) : \{0,1\}^X \to \{0,1\}$ die Boolesche Funktion

$$(\exists x.f) = f[x = 0] \vee f[x = 1]$$

Bemerkung: Wir könnten $(\exists x.f)$ auch als eine Boolesche Funktion über $X \setminus \{x\}$ betrachten.

Lemma 45

Sei D ein OBDD für die Boolesche Funktion $f : \{0,1\}^X \to \{0,1\}$ und sei $x \in X$.

Aus D kann ein OBDD der Größe $|D|^2$ für $(\exists x.f)$ berechnet werden.

Boolesche Operationen auf OBDDs

Beweis:

Wir konstruieren einen OBDD für $f[x = a]$, indem wir jeden Knoten v mit $\lambda(v) = x$ löschen, und alle Kanten, die nach v gehen, nach $\text{out}_a(v)$ richten.

Dann wenden wir die apply-Prozedur für $\vee$ auf die beiden OBDDs für $f[x = 0]$ und $f[x = 1]$ an. □

Bemerkung: Eine einfachere Konstruktion ist möglich, wenn x die letzte Variable in der Variablenordnung ist.

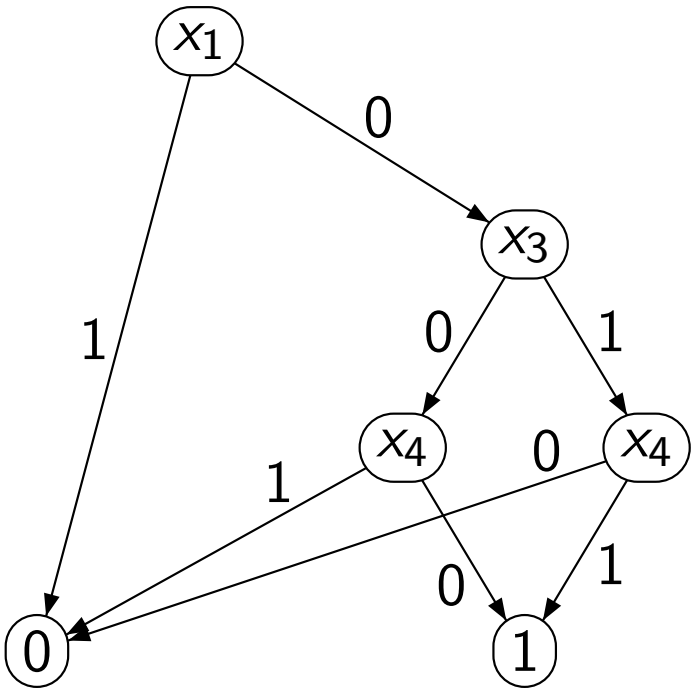
Dann identifizieren wir einfach jeden Knoten v mit $\lambda(v) = x$

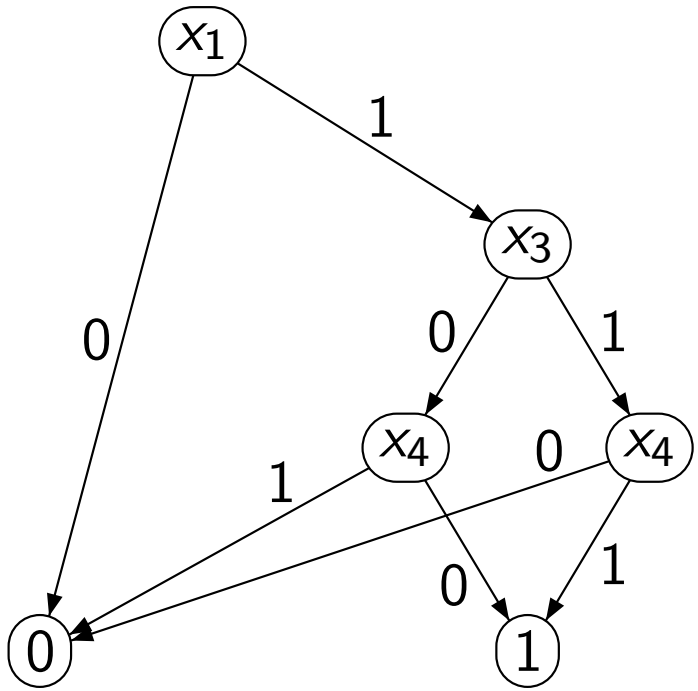
- der eine Kante zur 1-Senke hat mit der 1-Senke, bzw.
- der keine Kante zur 1-Senke hat mit der 0-Senke.

Kann zur Konstruktion eines OBDDs für $\exists x_i \exists x_{i+1} \cdots \exists x_n . f$ benutzt werden, falls die Variablenordnung $x_1 < x_2 < \cdots < x_n$ ist.

Boolesche Operationen auf OBDDs

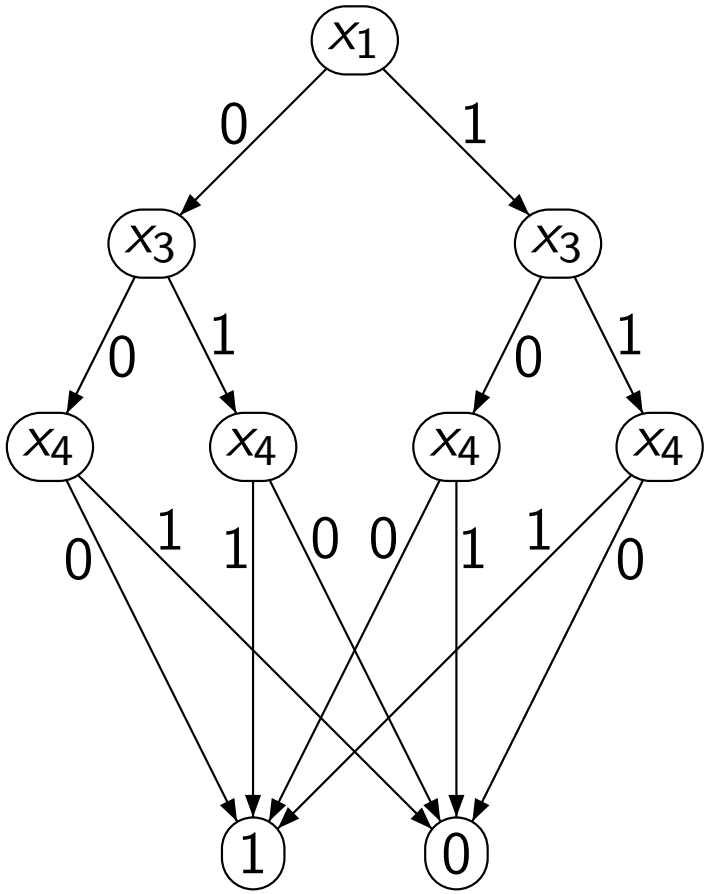
Beispiel: Diese beiden OBDDs entstehen aus dem OBDD von Folie 157 durch die Restriktionen $[x_2 = 0]$ bzw. $[x_2 = 1]$:





Boolesche Operationen auf OBDDs

Beispiel (Fortsetzung): apply für $\vee$ ergibt folgenden OBDD für $\exists x_2.(x_1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4) = x_3 \leftrightarrow x_4$:



Symbolisches CTL-Model-Checking mit OBDDs

Sei der Transitionsgraph $T = (V, E, \Pi, \pi)$ symbolisch durch Boolesche Funktionen kodiert (siehe Folie 147):

- $V = \{0,1\}^X$ für eine Variablenmenge X.
- $f_E : \{0,1\}^{X \cup X'} \to \{0,1\}$ mit:

$$\forall \beta, \gamma \in \{0,1\}^X : f_E(\beta, \gamma') = 1 \iff (\beta, \gamma) \in E.$$

- $f_p : \{0,1\}^X \to \{0,1\}$ für $p \in \Pi$ mit:

$$\forall \beta \in \{0,1\}^X : f_p(\beta) = 1 \iff p \in \pi(\beta).$$

Sei ψ eine existentielle CTL-Zustandsformel.

Wir berechnen für jede Teilformel θ von ψ einen OBDD D_θ mit:

$$\forall \beta \in \{0,1\}^X : f_{D_\theta}(\beta) = 1 \iff \beta \in [\![\theta]\!]_T.$$

Symbolisches CTL-Model-Checking mit OBDDs

Anders gesagt: Für eine Menge $U \subseteq V$ von Belegungen (also Knoten des Transitionsgraphen T) sei f_U die charakteristische Funktion für U.

Wir berechnen dann OBBDs für die charakteristischen Funktionen der Mengen $[\![\theta]\!]_T$.

Hierzu verwenden wir die Algorithmen von Folie 104 – 110.

Sei $X = \{x_1, \ldots, x_n\}$ und fixiere eine Variablenordnung auf $X \cup X' = \{x_1, \ldots, x_n, x'_1, \ldots, x'_n\}$.

Wichtig: $x_i < x_j \iff x'_i < x'_j$ für alle $1 \leq i, j \leq n$.

Mögliche Variablenordnungen:

- $x_1 < x'_1 < x_2 < x'_2 < \cdots < x_n < x'_n$
- $x_1 < x_2 < \cdots < x_n < x'_1 < x'_2 < \cdots < x'_n$.

Symbolisches CTL-Model-Checking mit OBDDs

Definition (Umbenennung von Variablen)

Für eine Belegung $\beta : X \to \{0,1\}$ der Variablen aus X sei $\beta' : X' \to \{0,1\}$ die Belegung mit $\beta'(x_i') = \beta(x_i)$.

Für eine Boolesche Funktion $f : \{0,1\}^X \to \{0,1\}$ sei $f' : \{0,1\}^{X'} \to \{0,1\}$ die Boolesche Funktion mit $f'(\beta') = f(\beta)$ für alle $\beta : X \to \{0,1\}$.

Einen Booleschen Ausdruck für f' erhält man also aus einem Booleschen Ausdruck für f durch Ersetzen jeder Variable x_i durch x_i'.

Einen OBDD für f' erhält man aus einem OBDD für f, indem jeder mit x_i markierte Knoten mit x_i' markiert wird.

Wichtig: Auf Grund der Annahme $(x_i < x_j \iff x_i' < x_j')$ wird beim Umbenennen die Variablenordnung erhalten.

Symbolisches CTL-Model-Checking mit OBDDs

Fall 1. $\theta = p \in \Pi$:

Konstruiere OBDD für f_p.

Fall 2. $\theta = \neg\theta'$:

Wir erhalten D_θ indem wir im OBDD $D_{\theta'}$ die 0-Senke mit der 1-Senke vertauschen (siehe Lemma 43).

Fall 3. $\theta = \theta_1 \wedge \theta_2$.

Wende apply-Prozedur für $\wedge$ auf die OBDDs D_{θ_1} und D_{θ_2} an (Lemma 44).

Fall 4. $\theta = \theta_1 \vee \theta_2$: analog zu Fall 3.

Symbolisches CTL-Model-Checking mit OBDDs

Fall 5. $\theta = \exists \mathrm{X}\, \xi$:

Wir haben den OBDD D_ξ (über der Variablenmenge $\{x_1, \ldots, x_n\}$) für die charakteristische Funktion von $[\![\xi]\!]_T$ bereits konstruiert.

Sei D_E ein OBDD für die Funktion $f_E : \{0,1\}^{X \cup X'} \to \{0,1\}$ (Kantenrelation).

Sei $f' := f'_{D_\xi} : \{0,1\}^{X'} \to \{0,1\}$ (Variablenumbenennung).

Einen OBDD D' für f' erhalten wir durch Ersetzen jeder Variable x_i in D_ξ durch x'_i.

Konstruiere aus D' und D_E unter Verwendung von Lemma 44 und 45 (existentielle Quantifizierung) einen OBDD für die Boolesche Funktion

$$\exists x'_1 \cdots \exists x'_n \,.\, (f_E(x_1, \ldots, x_n, x'_1, \ldots, x'_n) \wedge f'(x'_1, \ldots, x'_n)).$$

Symbolisches CTL-Model-Checking mit OBDDs

Fall 6. $\theta = \exists(\theta_1 \,\mathsf{U}\, \theta_2)$:

Zur Erinnerung (Folie 109):

- Sei $U_0 = [\![\theta_2]\!]_T$ und $W = [\![\theta_1]\!]_T$
- Für $i \geq 0$ definiere

$$U_{i+1} = U_i \cup (W \cap \{v \in V \mid \exists v' \in V : (v, v') \in E \wedge v' \in U_i\}).$$

- Dann gilt $U_0 \subseteq U_1 \subseteq U_2 \subseteq U_3 \subseteq \cdots$.
- Da V endlich ist, gibt es ein $i \geq 0$ (sogar $i \leq |V|$) mit $U_i = U_j$ für alle $j \geq i$ und es gilt $[\![\exists(\theta_1 \,\mathsf{U}\, \theta_2)]\!]_T = U_i$.

Wir haben die OBDDs $D' := D_{\theta_1}$ und $D_0 := D_{\theta_2}$ also bereits konstruiert.

Dann sind $f_0 := f_{D_0}$ und $g := f_{D'}$ die charakteristischen Funktionen von U_0 bzw. W.

Symbolisches CTL-Model-Checking mit OBDDs

Wir können dann OBBDs D_i $(i \geq 0)$ für die charakteristischen Funktionen der Mengen U_i berechnen (sei $\overline{x} = (x_1, \ldots, x_n)$ und $\overline{x}' = (x'_1, \ldots, x'_n)$):

$$f_{i+1}(\overline{x}) = f_i(\overline{x}) \ \vee \ (g(\overline{x}) \wedge \exists \overline{x}' \,.\, (f_E(\overline{x}, \overline{x}') \wedge f'_i(\overline{x}'))).$$

Wir minimieren den OBDD D_i mittels der Reduktionsregeln und stoppen, wenn D_i und D_{i+1} (beide minimal) isomorph sind.

Symbolisches CTL-Model-Checking mit OBDDs

Fall 7. $\theta = \exists \mathrm{G} \xi$:

Erinnerung (Folie 110)

- Sei $U_0 = [\![\xi]\!]_T$
- Für $i \geq 0$ definiere

$$U_{i+1} = U_i \cap \{v \in V \mid \exists v' \in V : (v, v') \in E \wedge v' \in U_i\}.$$

- Dann gilt $U_0 \supseteq U_1 \supseteq U_2 \supseteq U_3 \supseteq \cdots$.
- Da V endlich ist, gibt es ein $i \geq 0$ (sogar $i \leq |V|$) mit $U_i = U_j$ für alle $j \geq i$ und es gilt $[\![\exists \mathrm{G} \xi]\!]_T = U_i$.

Wir haben den OBDD $D_0 := D_\xi$ also bereits konstruiert.

Dann ist $f_0 := f_{D_0}$ die charakteristische Funktion von U_0.

Symbolisches CTL-Model-Checking mit OBDDs

Wir können dann OBBDs D_i $(i \geq 0)$ für die charakteristischen Funktionen der Mengen U_i berechnen (sei $\overline{x} = (x_1, \ldots, x_n)$ und $\overline{x}' = (x'_1, \ldots, x'_n)$):

$$f_{i+1}(\overline{x}) = f_i(\overline{x}) \ \wedge \ \exists \overline{x}' \,.\, (f_E(\overline{x}, \overline{x}') \wedge f'_i(\overline{x}')).$$

Wir minimieren den OBDD D_i mittels der Reduktionsregeln und stoppen, wenn D_i und D_{i+1} (beide minimal) isomorph sind.

Dies beendet die induktive Konstruktion der OBDDs für die Teilformeln θ.

Nach jedem Schritt sollten die konstruierten OBDDs minimiert werden.

Symbolisches CTL-Model-Checking mit OBDDs

Kritisch ist die Wahl der Variablenordnung $<$ auf $\{x_1, \ldots, x_n, x'_1, \ldots, x'_1\}$.

- $x_1 < x_2 < \cdots < x_n < x'_1 < x'_2 < \cdots < x'_n$:

  Macht die existentielle Quantifizierung $\exists x'_1 \cdots \exists x'_n$ einfach für OBDDs.

- $x_1 < x'_1 < x_2 < x'_2 < \cdots < x_n < x'_n$ (verschränkte Variablenordnung):

  Führt häufig zu einfachen OBDD für f_E (Kantenrelation).

  Häufig ergibt sich E als direktes Produkt mehrerer Kantenrelationen $E_1, \ldots, E_k$ (synchrones Produkt von Transitionsgraphen):

  $$E = \{(\langle u_1, \ldots, u_k \rangle, \langle v_1, \ldots, v_k \rangle) \mid (u_1, v_1) \in E_1, \ldots (u_k, v_k) \in E_k\}$$

  Hat man OBDDs für $f_{E_1}, \ldots, f_{E_k}$ so kann man unter der verschränkten Variablenordnung leichter einen OBDD für f_E konstruieren.

SMV

SMV (symbolic model verifier), siehe
`http://www.cs.cmu.edu/~modelcheck/smv.html`:

- Entwickelt von Ken McMillan an der Carnegie Mellon Universität Anfang der 1990'er Jahre.
- symbolischer CTL model checker basierend auf OBDDs
- Weiterentwicklung: nuSMV (new SMV), siehe `http://nusmv.fbk.eu`.

Unendliche Transitionsgraphen

Bisher haben wir in der Vorlesung nur Transitionsgraphen mit endlich vielen Knoten betrachtet.

Eignen sich z.B. zur Modellierung von Hardware.

Aber: Transitionsgraphen, welche Softwaresysteme modellieren sind typischerweise unendlich auf Grund von z.B.

- Variablen mit unendlichen Wertebereichen (integers, reals),
- dynamischer Erzeugung beliebig vieler Prozesse, oder
- beliebig großen Programmstacks

Unendliche Transitionsgraphen

Bisher haben wir in der Vorlesung nur Transitionsgraphen mit endlich vielen Knoten betrachtet.

Eignen sich z.B. zur Modellierung von Hardware.

Aber: Transitionsgraphen, welche Softwaresysteme modellieren sind typischerweise unendlich auf Grund von z.B.

- Variablen mit unendlichen Wertebereichen (integers, reals),
- dynamischer Erzeugung beliebig vieler Prozesse, oder
- beliebig großen Programmstacks $\rightarrow$ Pushdownsysteme

Pushdownsysteme

Pushdowngraphen können zur Modellierung von rekursiven Programmen eingesetzt werden, falls alle lokalen und globalen Variablen nur endliche Wertebereiche haben (mehr dazu später).

Pushdowngraphen werden durch Pushdownsysteme (Kellersysteme) beschrieben.

Definition

Ein Pushdownsystem ist ein Tupel $P = (Q, \Gamma, \Delta, \Pi, \rho)$ mit:

- Q ist eine endliche Menge von Zuständen.
- Γ ist eine endliche Menge von Kellersymbolen ($Q \cap \Gamma = \emptyset$).
- Δ ist eine endliche Teilmenge von $(Q \times \Gamma) \times (Q \times \Gamma^*)$.
- Π ist eine endliche Menge von Propositionen.
- $\rho : Q \to 2^{\Pi}$

Der Pushdowngraph $T(P)$

Für ein Pushdownsystem $P = (Q, \Gamma, \Delta, \Pi, \rho)$ definieren wir den Transitionsgraphen

$$T(P) = (Q\Gamma^*, \Rightarrow_P, \Pi, \pi),$$

wobei

- $\Rightarrow_P = \{(qAu, pvu) \mid u \in \Gamma^*, (q, A, p, v) \in \Delta\}$ und
- $\pi(qu) = \rho(q)$ für $q \in Q$, $u \in \Gamma^*$.

Elemente von $Q\Gamma^*$ werden auch als Konfigurationen von P bezeichnet.

Ein Pushdowngraph ist ein Transitionsgraph der Form $T(P)$, wobei P ein Pushdownsystem ist.

Beispiel

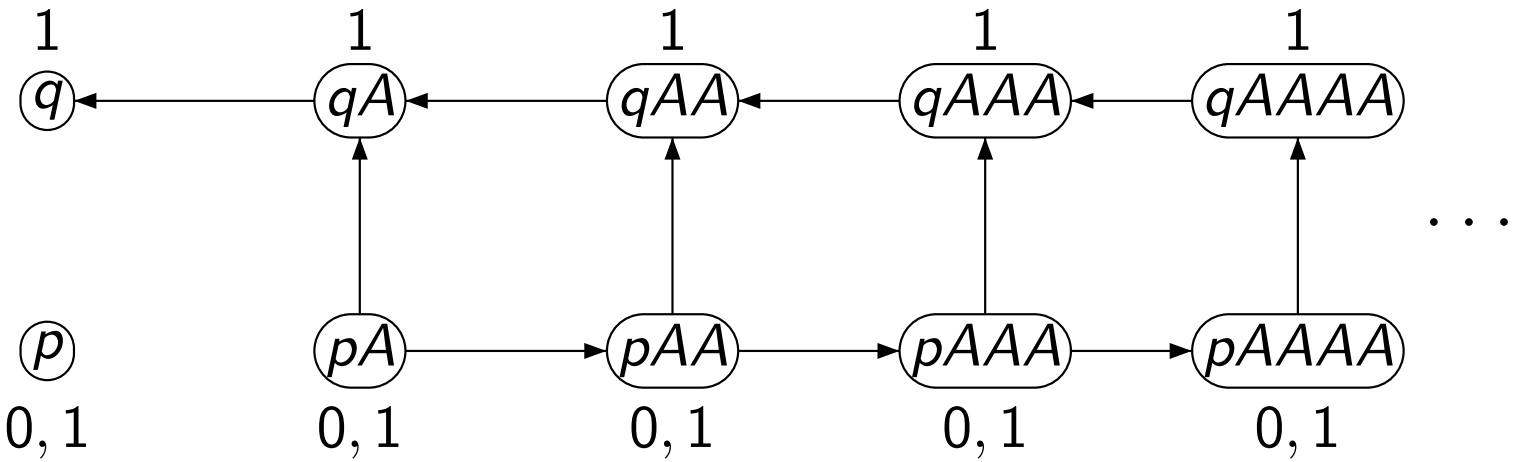
Beispiel 1: Sei

$$P = (\{p, q\}, \{A\}, \Delta, \{0, 1\}, \rho)$$

wobei

- $\Delta = \{(p, A, p, AA), (p, A, q, A), (q, A, q, \varepsilon)\}$ und
- $\rho(p) = \{0, 1\}, \rho(q) = \{1\}$

Dann sieht $T(P)$ wie folgt aus:



Erreichbarkeit in Pushdowngraphen

Im folgenden sei ein Pushdownsystem

$$P = (Q, \Gamma, \Delta, \Pi, \rho)$$

fixiert, und

$$T = T(P) = (Q\Gamma^*, \Rightarrow_P, \Pi, \pi)$$

sei der zugehörige Transitionsgraph.

Unser Ziel ist die Berechnung von $\mathrm{pre}^*_T(U)$ für eine Teilmenge $U \subseteq Q\Gamma^*$ von Konfigurationen.

Problem: Wie soll eine Teilmenge $U \subseteq Q\Gamma^*$ repräsentiert werden?

Wir betrachten reguläre Mengen von Konfigurationen, diese werden durch sogenannte P-Multiautomaten definiert.

Erreichbarkeit in Pushdowngraphen

Definition

Ein P-Multiautomat ist ein Tripel $M = (S, \delta, F)$, wobei gilt:

- S ist eine endliche Menge von Zuständen mit $Q \subseteq S$
- $\delta \subseteq S \times \Gamma \times S$
- $F \subseteq S$ (die Menge der Endzustände)

Für $w \in \Gamma^*$ definieren wir die Relation $\xrightarrow{w}_M \subseteq S \times S$ wie folgt induktiv über $|w|$:

- $\xrightarrow{\varepsilon}_M = \mathrm{Id}_S = \{(s, s) \mid s \in S\}$
- $\xrightarrow{Av}_M = \{(s_1, s_2) \mid \exists s \in S : (s_1, A, s) \in \delta, s \xrightarrow{v}_M s_2\}$ für $A \in \Gamma$, $v \in \Gamma^*$

Die durch M akzeptierte Konfigurationsmenge ist

$$L(M) = \{qw \mid q \in Q, w \in \Gamma^*, \exists f \in F : q \xrightarrow{w}_M f\}.$$

Eine Menge $U \subseteq Q\Gamma^*$ ist regulär falls ein P-Multiautomat M mit $U = L(M)$ existiert.

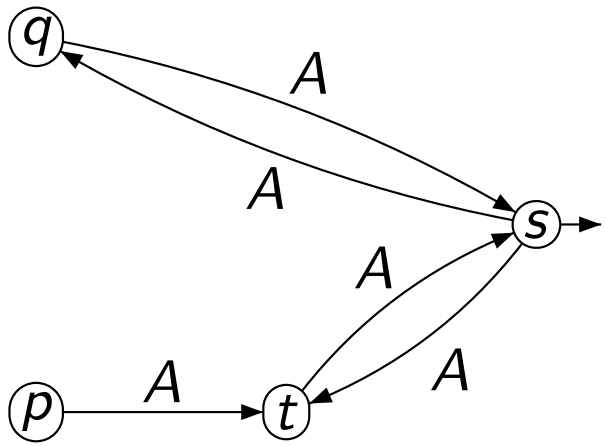
Erreichbarkeit in Pushdowngraphen

Beispiel 2:

Sei P das Pushdownsystem aus Beispiel 1. Dann ist

$$M = (\{p, q, s, t\}, \{(q, A, s), (s, A, q), (p, A, t), (t, A, s), (s, A, t)\}, \{s\})$$

ein P-Multiautomat, der durch das folgende Diagramm dargestellt werden kann:



Dieser Automat akzeptiert die Menge von Konfigurationen

$$L(M) = \{qA^{2n+1} \mid n \geq 0\} \cup \{pA^{2n} \mid n \geq 1\}.$$

Erreichbarkeit in Pushdowngraphen

Im Folgenden verlangen wir für einen P-Multiautomaten $M = (S, \delta, F)$ (wobei $P = (Q, \Gamma, \Delta, \Pi, \rho)$), dass $\delta \subseteq S \times \Gamma \times (S \setminus Q)$ gilt.

Es gibt also keine Transitionen, die in einen Zustand aus Q führen.

Beachte: Aus einem P-Multiautomaten $M = (S, \delta, F)$ kann man durch Hinzufügen von höchstens $|Q|$ vielen Zuständen einen P-Multiautomaten $M' = (S', \delta', F')$ mit $L(M) = L(M')$ und $\delta' \subseteq S' \times \Gamma \times (S' \setminus Q)$ konstruieren.

Sei $|\Delta| = \sum_{(q,A,p,v) \in \Delta} (|v| + 1)$.

Satz 46 (Bouajjani, Esparza, Maler 1997)

Für ein Pushdownsystem $P = (Q, \Gamma, \Delta, \Pi, \rho)$ und einen P-Multiautomaten $M = (S, \delta, F)$ kann in Zeit $O(|S|^2 \cdot |\Gamma| \cdot |\Delta|)$ ein P-Multiautomat $M' = (S, \delta', F)$ mit $L(M') = \text{pre}^*_{T(P)}(L(M))$ berechnet werden.

Erreichbarkeit in Pushdowngraphen

Korollar aus Satz 46

Für ein gegebenes Pushdownsystem $P = (Q, \Gamma, \Delta, \Pi, \rho)$ und zwei Konfigurationen $qu, pv \in Q\Gamma^*$ kann man in Polynomialzeit entscheiden, ob $qu \Rightarrow_P^* pv$ gilt.

Beweis:

1. Konstruiere einen P-Multiautomaten M mit $L(M) = \{pv\}$ (ist einfach)
2. Konstruiere einen P-Multiautomaten M' mit $L(M') = \text{pre}^*_{T(P)}(L(M))$ (geht in Polynomialzeit nach Satz 46)
3. Teste, ob $qu \in L(M')$ (ist einfach) □

Erreichbarkeit in Pushdowngraphen

Beweis von Satz 46:

Sei $P = (Q, \Gamma, \Delta, \Pi, \rho)$ ein Pushdownsystem und $M = (S, \delta, F)$ ein P-Multiautomat.

Beachte: Es gibt keine Transition $(s, A, q) \in \delta$ mit $q \in Q$.

Definiere Mengen X_i für $i \geq 0$ wie folgt induktiv:

$$\begin{aligned} X_0 &= L(M) \\ X_{i+1} &= X_i \cup \{qu \in Q\Gamma^* \mid \exists pv \in X_i : qu \Rightarrow_P pv\} \end{aligned}$$

X_i ist die Menge aller Konfigurationen, von denen man in höchstens i vielen $\Rightarrow_P$-Schritten nach $L(M)$ kommt.

Somit gilt:

$$\bigcup_{i \geq 0} X_i = \text{pre}^*_{T(P)}(L(M))$$

Erreichbarkeit in Pushdowngraphen

Wir konstruieren P-Multiautomaten $M_i = (S, \delta_i, F)$ $(i \geq 0)$ mit den folgenden drei Eigenschaften:

(P1) $\exists k \geq 0 : \delta_0 \subsetneq \delta_1 \subsetneq \cdots \subsetneq \delta_k = \delta_{k+1} = \delta_{k+2} = \cdots$

(P2) $\forall i \geq 0 : X_i \subseteq L(M_i)$

(P3) $\forall i \geq 0 : L(M_i) \subseteq \bigcup_{j \geq 0} X_j = \text{pre}^*_{T(P)}(L(M))$

(P2) und (P3) $\rightsquigarrow$:

$$\text{pre}^*_{T(P)}(L(M)) = \bigcup_{i \geq 0} X_i \subseteq \bigcup_{i \geq 0} L(M_i) \subseteq \text{pre}^*_{T(P)}(L(M))$$

und somit

$$\bigcup_{i \geq 0} L(M_i) = \text{pre}^*_{T(P)}(L(M)). \tag{2}$$

Erreichbarkeit in Pushdowngraphen

(P1) $\rightsquigarrow$ (k ist der Wert aus (P1)):

$$L(M_0) \subseteq L(M_1) \subseteq \cdots \subseteq L(M_k) = L(M_{k+1}) = L(M_{k+2}) = \cdots$$

Hieraus folgt mit (2)

$$L(M_k) = \bigcup_{i \geq 0} L(M_i) = \text{pre}^*_{T(P)}(L(M)).$$

Wir können also $M' = M_k$ setzen.

Konstruktion der M_i:

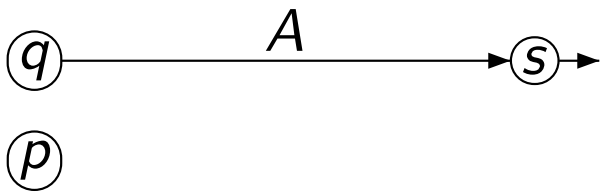
- Es sei $M_0 = M$.
- Sei $M_i = (S, \delta_i, F)$ bereits konstruiert. Dann ist

$$\delta_{i+1} = \delta_i \cup \{(q, A, s) \in Q \times \Gamma \times S \mid \exists (q, A, p, v) \in \Delta : p \xrightarrow{v}_{M_i} s\}.$$

Erreichbarkeit in Pushdowngraphen

Beispiel zur Konstruktion:

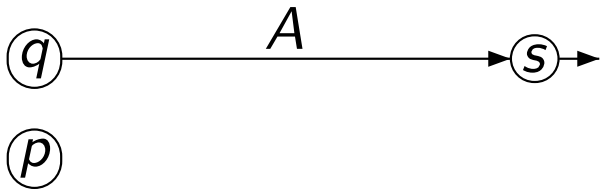
Sei P wieder das Pushdownsystem aus Beispiel 1 und betrachte den folgenden P-Multiautomaten M (es gilt $L(M) = \{qA\}$):



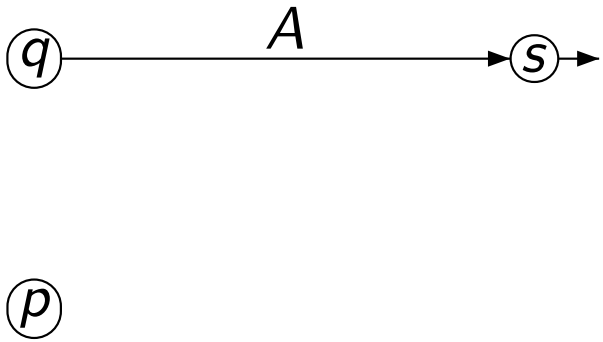
Erreichbarkeit in Pushdowngraphen

Beispiel zur Konstruktion:

Sei P wieder das Pushdownsystem aus Beispiel 1 und betrachte den folgenden P-Multiautomaten M (es gilt $L(M) = \{qA\}$):



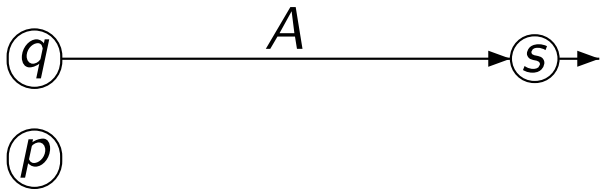
P-Multiautomat M_0:



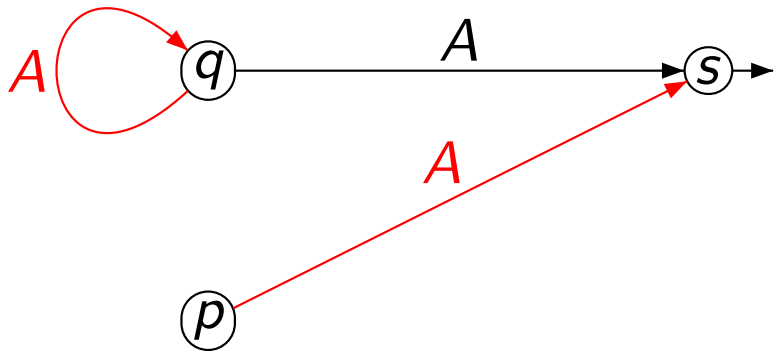
Erreichbarkeit in Pushdowngraphen

Beispiel zur Konstruktion:

Sei P wieder das Pushdownsystem aus Beispiel 1 und betrachte den folgenden P-Multiautomaten M (es gilt $L(M) = \{qA\}$):



P-Multiautomat M_1:

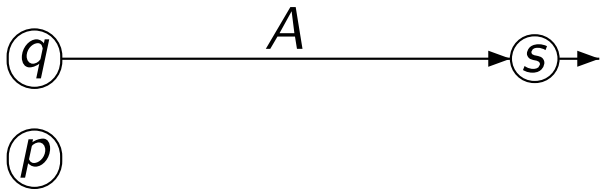


$$\left.\begin{array}{l}(q, A, q, \varepsilon) \in \Delta \\ q \xrightarrow{\varepsilon}_{M_0} q\end{array}\right\} \rightsquigarrow q \xrightarrow{A}_{M_1} q \qquad \left.\begin{array}{l}(p, A, q, A) \in \Delta \\ q \xrightarrow{A}_{M_0} s\end{array}\right\} \rightsquigarrow p \xrightarrow{A}_{M_1} s$$

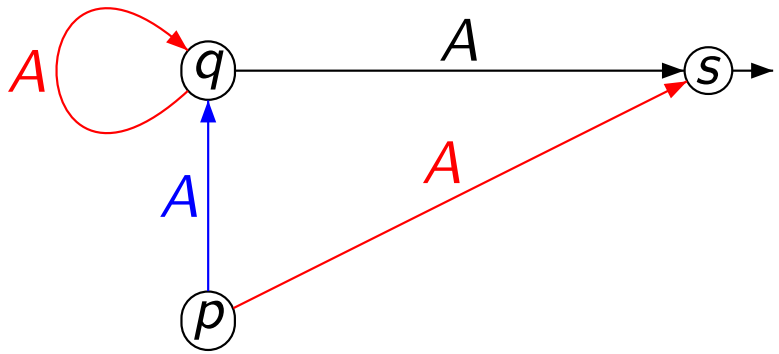
Erreichbarkeit in Pushdowngraphen

Beispiel zur Konstruktion:

Sei P wieder das Pushdownsystem aus Beispiel 1 und betrachte den folgenden P-Multiautomaten M (es gilt $L(M) = \{qA\}$):



P-Multiautomat M_2:

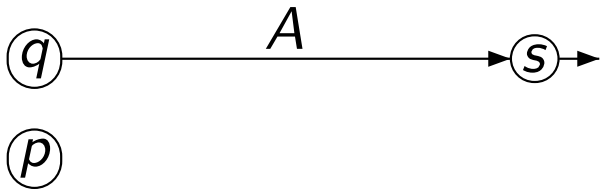


$$\left.\begin{array}{l}(p, A, q, A) \in \Delta \\ q \xrightarrow{A}_{M_1} q\end{array}\right\} \rightsquigarrow p \xrightarrow{A}_{M_2} q$$

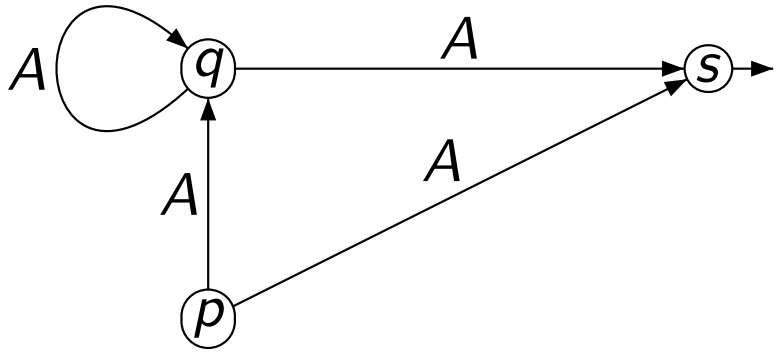
Erreichbarkeit in Pushdowngraphen

Beispiel zur Konstruktion:

Sei P wieder das Pushdownsystem aus Beispiel 1 und betrachte den folgenden P-Multiautomaten M (es gilt $L(M) = \{qA\}$):



P-Multiautomat M_2:



Der Automat M_2 ist saturiert, und es gilt (wie erhofft)
$L(M_2) = \{xA^n \mid x \in \{p, q\}, n \geq 1\} = \text{pre}^*_{T(P)}(\{qA\})$.

Erreichbarkeit in Pushdowngraphen

Beweis von (P1): $\exists k \geq 0 : \delta_0 \subsetneq \delta_1 \subsetneq \cdots \subsetneq \delta_k = \delta_{k+1} = \delta_{k+2} = \cdots$

Da jeder Automat M_i die gleiche Zustandsmenge S und das gleiche Alphabet Γ hat, kann ein δ_i höchstens $|S|^2 \cdot |\Gamma|$ viele Transitionen enthalten.

Dies impliziert (P1) für ein $k \leq |S|^2 \cdot |\Gamma|$.

Beweis von (P2): $\forall i \geq 0 : X_i \subseteq L(M_i)$

Induktion über $i \geq 0$:

IA: $X_0 = L(M) = L(M_0)$

IS: Sei $i \geq 0$ und $qu \in X_{i+1}$.

Also müssen $(q, A, p, v) \in \Delta$, $w \in \Gamma^*$ mit $u = Aw$ und $pvw \in X_i$ existieren.

Induktionshypothese $\rightsquigarrow pvw \in L(M_i)$, d.h.

$$p \xrightarrow{v}_{M_i} s \xrightarrow{w}_{M_i} f \in F.$$

Erreichbarkeit in Pushdowngraphen

Konstruktion von $M_{i+1} \rightsquigarrow$

$$q \xrightarrow{A}_{M_{i+1}} s \xrightarrow{w}_{M_{i+1}} f \in F,$$

d.h. $qu = qAw \in L(M_{i+1})$.

Beweis von (P3): Wir benötigen das folgende Lemma:

Lemma 47

Gelte $q \xrightarrow{w}_{M_i} s$ für ein $q \in Q$, $s \in S$. Dann gilt:

$$\exists p \in Q, v \in \Gamma^* : qw \Rightarrow_P^* pv, \; p \xrightarrow{v}_M s.$$

Beweis von Lemma 47:

Induktion über $i \geq 0$:

Erreichbarkeit in Pushdowngraphen

IA: Gelte $q \xrightarrow{w}_{M_0} s$.

$M_0 = M \quad \rightsquigarrow \quad q \xrightarrow{w}_{M} s$ und $qw \Rightarrow_P^* qw$.

Wir können also $p = q$ und $v = w$ wählen.

IS: Gelte $q \xrightarrow{w}_{M_i} s$ für $i > 0$.

Sei $w = A_1 A_2 \cdots A_k$. Dann existiert ein Pfad

$$q = s_0 \xrightarrow{A_1}_{M_i} s_1 \xrightarrow{A_2}_{M_i} s_2 \cdots \xrightarrow{A_k}_{M_i} s_k = s. \tag{3}$$

Induktion über Anzahl der Transitionen $(s_{k-1}, A_k, s_k) \in \delta_i \setminus \delta_{i-1}$:

Falls diese Anzahl $= 0$ ist, d. h. $q \xrightarrow{w}_{M_{i-1}} s$, so erhalten wir das Lemma mit Induktion (nach i).

Erreichbarkeit in Pushdowngraphen

Wenn diese Anzahl > 0 ist, können wir den Pfad (3) faktorisieren als

$$q \xrightarrow{w_1}_{M_i} p \xrightarrow{A}_{M_i} s' \xrightarrow{w_2}_{M_{i-1}} s \text{ mit } (p, A, s') \in \delta_i \backslash \delta_{i-1},\ w = w_1 A w_2. \quad (4)$$

Beachte: aus der Konstruktion von M_i folgt $p \in Q$.

Da entlang des Pfades $q \xrightarrow{w_1}_{M_i} p$ weniger Transitionen aus $\delta_i \setminus \delta_{i-1}$ als in (3) vorkommen, erhalten wir induktiv:

$$\exists q' \in Q, w' \in \Gamma^* : q w_1 \Rightarrow_P^* q' w', \quad q' \xrightarrow{w'}_M p.$$

Da $p \in Q$ gilt, hat p in M keine eingehenden Transitionen.
Also muss $q' = p$ und $w' = \varepsilon$ gelten, d. h.

$$q w_1 \Rightarrow_P^* p\varepsilon. \quad (5)$$

$(p, A, s') \in \delta_i \setminus \delta_{i-1}$ und Konstruktion von $M_i \rightsquigarrow$

$$\exists (p, A, p', v) \in \Delta : p' \xrightarrow{v}_{M_{i-1}} s' \quad (6)$$

Erreichbarkeit in Pushdowngraphen

(6) und Induktion (nach i) $\rightsquigarrow$

$$\exists r' \in Q, v' \in \Gamma^* : p'v \Rightarrow_P^* r'v', \; r' \xrightarrow{v'}_M s' \tag{7}$$

(4) und (7) $\rightsquigarrow$

$$r' \xrightarrow{v'}_M s' \xrightarrow{w_2}_{M_{i-1}} s, \quad \text{d.h.} \quad r' \xrightarrow{v'w_2}_{M_{i-1}} s$$

Induktion nach i $\rightsquigarrow$

$$\exists r \in Q, u \in \Gamma^* : \; r'v'w_2 \Rightarrow_P^* r\,u, \quad r \xrightarrow{u}_M s \tag{8}$$

(5)–(8) $\rightsquigarrow$

$$q\,w = q\,w_1 A\,w_2 \Rightarrow_P^* p\,A\,w_2 \Rightarrow_P p'v\,w_2 \Rightarrow_P^* r'v'w_2 \Rightarrow_P^* r\,u$$

wobei $r \xrightarrow{u}_M s$. Dies zeigt Lemma 47. □

Erreichbarkeit in Pushdowngraphen

Zurück zum Beweis von Eigenschaft (P3):

$$\forall i \geq 0 : L(M_i) \subseteq \bigcup_{i \geq 0} X_i = \text{pre}^*_{T(P)}(L(M))$$

Sei $q\,w \in L(M_i)$.

$\rightsquigarrow$ $q \xrightarrow{w}_{M_i} f$ für ein $f \in F$.

Lemma 47 $\rightsquigarrow$ $\exists p \in Q, v \in \Gamma^* : \; q\,w \;\Rightarrow^*_P \; p\,v, \quad p \xrightarrow{v}_M f$

$\rightsquigarrow$ $p\,v \in L(M)$

$\rightsquigarrow$ $q\,w \in \text{pre}^*_{T(P)}(L(M))$. $\square$

LTL-Model-Checking in Pushdowngraphen

Satz 48 (Bouajjani, Esparza, Maler 1997)

Für eine gegebene Formel $\varphi \in \mathrm{LTL}(\Pi)$ und ein Pushdownsystem $P = (Q, \Gamma, \Delta, \Pi, \rho)$ kann in exponentieller Zeit ein P-Multiautomat M berechnet werden mit: $L(M) = \{qu \in Q\Gamma^* \mid (T(P), qu) \not\models \varphi\}$.

Bemerkung: Die exponentielle Zeitschranke in Satz 48 kann nicht verbessert werden, da bereits das folgende Problem EXPTIME-vollständig ist:
EINGABE: Ein Pushdownsystem $P = (Q, \Gamma, \Delta, \Pi, \rho)$, eine Konfiguration $c \in Q\Gamma^*$ und eine LTL-Formel $\varphi \in \mathrm{LTL}(\Pi)$.
FRAGE: Gilt $(T(P), c) \models \varphi$.

LTL-Model-Checking in Pushdowngraphen

Beweis von Satz 48:

Wir gehen wie folgt vor:

Schritt 1:

Konstruiere aus φ einen Büchiautomaten

$$B = (S, 2^{\Pi}, \delta, s_0, F)$$

mit

$$L(B) = \{w \in (2^{\Pi})^{\omega} \mid w \models \neg\varphi\}.$$

Die hierfür benötigte Zeit ist nach Satz 4 durch $2^{O(|\varphi|)}$ beschränkt.

Ausserdem ist die Anzahl der Zustände $|S|$ von B auch durch $2^{O(|\varphi|)}$ beschränkt.

LTL-Model-Checking in Pushdowngraphen

Schritt 2:

Synchronisiere das Pushdownsystem P mit dem Büchiautomaten B. Dieser Schritt ist analog zur Konstruktion eines Pushdownautomaten für $L(A) \cap L(A')$ für einen (gewöhnlichen) endlichen Automaten A und einen Pushdownautomaten A'; siehe Vorlesung *Automaten und Sprachen*.

Formal: Definiere das Pushdownsystem

$$P \times B = (Q \times S, \Gamma, \Delta', \emptyset, \emptyset)$$

wobei für alle $(q, s), (p, t) \in Q \times S$, $A \in \Gamma$, $u \in \Gamma^*$ gilt:

$$((q, s), A, (p, t), u) \in \Delta' \iff \left\{ \begin{array}{l} (q, A, p, u) \in \Delta \text{ und} \\ (s, \rho(q), t) \in \delta \end{array} \right\}$$

LTL-Model-Checking in Pushdowngraphen

Dann gilt für alle $q_0 \in Q$ und alle $u_0 \in \Gamma^*$:

$$
\begin{aligned}
& (T(P), q_0 u_0) \not\models \varphi \\
\iff & \exists \text{ Pfad } q_0 u_0 \Rightarrow_P q_1 u_1 \Rightarrow_P q_2 u_2 \Rightarrow_P \cdots \\
& \qquad \text{mit } (\rho(q_0)\rho(q_1)\rho(q_2)\cdots) \not\models \varphi \\
\iff & \exists \text{ Pfad } q_0 u_0 \Rightarrow_P q_1 u_1 \Rightarrow_P q_2 u_2 \Rightarrow_P \cdots \\
& \qquad \text{mit } (\rho(q_0)\rho(q_1)\rho(q_2)\cdots) \models \neg\varphi \\
\iff & \exists \text{ Pfad } q_0 u_0 \Rightarrow_P q_1 u_1 \Rightarrow_P q_2 u_2 \Rightarrow_P \cdots \\
& \qquad \text{mit } (\rho(q_0)\rho(q_1)\rho(q_2)\cdots) \in L(B) \\
\iff & \exists \text{ Pfad } (q_0, s_0) u_0 \Rightarrow_{P\times B} (q_1, s_1) u_1 \Rightarrow_{P\times B} (q_2, s_2) u_2 \Rightarrow_{P\times B} \cdots \\
& \qquad \text{so dass unendlich viele } i \geq 0 \text{ mit } s_i \in F \text{ existieren}
\end{aligned}
$$

LTL-Model-Checking in Pushdowngraphen

Schritt 3:

Konstruiere (in Polynomialzeit) einen $(P \times B)$-Multiautomaten N mit

$$L(N) = \{(q,s)u \mid \exists \text{ Pfad } (q,s)u \Rightarrow_{P\times B} (q_1,s_1)u_1 \Rightarrow_{P\times B} (q_2,s_2)u_2 \cdots \text{ so dass } \infty \text{ viele } i \geq 1 \text{ mit } s_i \in F \text{ existieren}\}$$

(siehe nächste Folie)

Dann gilt:

$$\forall q \in Q \, \forall u \in \Gamma^* : (T(P), qu) \not\models \varphi \iff (q, s_0)u \in L(N)$$

Aus N können wir leicht (in Polynomialzeit) einen P-Multiautomaten M konstruieren mit:

$$\forall q \in Q \, \forall u \in \Gamma^* : qu \in L(M) \iff (q, s_0)u \in L(N)$$

Damit gilt wie gewünscht:

$$\forall q \in Q \, \forall u \in \Gamma^* : (T(P), qu) \not\models \varphi \iff qu \in L(M)$$

LTL-Model-Checking in Pushdowngraphen

Wir müssen noch den folgenden Satz beweisen:

Satz 49

Für ein gegebenes Pushdownsystem $P = (Q, \Gamma, \Delta, \Pi, \rho)$ und eine Teilmenge $R \subseteq Q$ können wir in Polynomialzeit einen P-Multiautomaten M konstruieren mit:

$$L(M) = \{c \in Q\Gamma^* \mid \exists \text{ Pfad } c = q_0 u_0 \Rightarrow_P q_1 u_1 \Rightarrow_P q_2 u_2 \Rightarrow_P \cdots \text{ so dass } \infty \text{ viele } i \geq 0 \text{ mit } q_i \in R \text{ existieren}\}$$

Der Beweis von Satz 49 basiert auf dem folgenden Lemma:

LTL-Model-Checking in Pushdowngraphen

Lemma 50

Sei $P = (Q, \Gamma, \Delta, \Pi, \rho)$ ein Pushdownsystem, $R \subseteq Q$ und $c \in Q\Gamma^*$.
Folgende beiden Aussagen sind äquivalent:

(1) Es existiert ein Pfad $c = q_0 u_0 \Rightarrow_P q_1 u_1 \Rightarrow_P q_2 u_2 \Rightarrow_P \cdots$, so dass ∞ viele $i \geq 0$ mit $q_i \in R$ existieren.

(2) Es gibt $q \in Q$, $r \in R$ und $A \in \Gamma$ mit:
   (i) $\exists w \in \Gamma^* : c \Rightarrow_P^* q A w$
   (ii) $\exists u, v \in \Gamma^* : q A \Rightarrow_P^+ r u \Rightarrow_P^* q A v$

Beweis:

(2) $\Rightarrow$ (1): Aus (2) erhalten wir den folgenden unendlichen Pfad:

$$c \Rightarrow_P^* q A w \Rightarrow_P^+ r u w \Rightarrow_P^* q A v w =$$
$$q A v w \Rightarrow_P^+ r u v w \Rightarrow_P^* q A v v w =$$
$$q A v v w \Rightarrow_P^+ r u v v w \Rightarrow_P^* q A v v v w \cdots$$

LTL-Model-Checking in Pushdowngraphen

Auf diesem Pfad wird der Zustand $r \in R$ unendlich oft besucht.
Also gilt (1).

(1) $\Rightarrow$ (2): Sei

$$c = q_0 u_0 \Rightarrow_P q_1 u_1 \Rightarrow_P q_2 u_2 \Rightarrow_P \cdots$$

ein Pfad, so dass ∞ viele $i \geq 0$ mit $q_i \in R$ existieren.

Dann gibt es ein $r \in R$, so dass $q_i = r$ für ∞ viele $i \geq 0$ gilt.

Sei $I = \{i \in \mathbb{N} \mid \forall j \geq i : |u_j| \geq |u_i|\}$.

Da $(\mathbb{N}, \leq)$ eine Wohlordnung ist, muss I unendlich sein.

Sei $I = \{i_0, i_1, i_2, \ldots\}$ mit $i_0 < i_1 < i_2 < \cdots$.

$\rightsquigarrow$ Wenn $u_{i_k} = A\,w$ (mit $A \in \Gamma$, $w \in \Gamma^*$), dann existiert für jedes $j > i_k$ ein $u \in \Gamma^+$ mit $u_j = u\,w$ und $q_{i_k} A \Rightarrow_P^+ q_j u$.

LTL-Model-Checking in Pushdowngraphen

Da Q und Γ endlich sind, existieren $q \in Q$, $A \in \Gamma$ und eine Teilfolge $j_0 < j_1 < j_2 < \cdots$ von $i_0 < i_1 < i_2 < \cdots$, so dass

$$\forall k \geq 0 : q_{j_k} = q, \ u_{j_k} \in A\Gamma^*.$$

Da es ∞ viele $i \geq 0$ mit $q_i = r$ gibt, existiert ein $k \geq 1$ mit

$$c = q_0 u_0 \ \Rightarrow_P^* \ q_{j_0} u_{j_0} = q u_{j_0} \ \Rightarrow_P^+ \ r u_\ell \ \Rightarrow_P^* \ q_{j_k} u_{j_k} = q u_{j_k}.$$

Sei $u_{j_0} = A\,w$.

Dann gilt also $c \ \Rightarrow_P^* \ q\,A\,w$ (dies ist (i)).

Ausserdem erhalten wir Faktorisierungen $u_\ell = u\,w$ und $u_{j_k} = A\,v\,w$, so dass

$$q\,A \ \Rightarrow_P^+ \ r\,u \ \Rightarrow_P^* \ q\,A\,v$$

(dies ist (ii)). □

LTL-Model-Checking in Pushdowngraphen

Nun können wir den Beweis von Satz 49 (und damit von Satz 48) beenden.

Aufgrund von Lemma 50 genügt es, in Polynomialzeit einen P-Multiautomaten M zu konstruieren, so dass gilt:

$c \in L(M)$ genau dann, wenn $q \in Q$ und $A \in \Gamma$ existieren mit:

(i) $\exists w \in \Gamma^* : c \Rightarrow_P^* q\,A\,w$

(ii) $\exists r \in R\, \exists u, v \in \Gamma^* : q\,A \Rightarrow_P^+ r\,u \Rightarrow_P^* q\,A\,v$

Bedingung (i) und (ii) sind äquivalent zu

(i') $c \in \mathrm{pre}^*(q\,A\,\Gamma^*)$

(ii') $q\,A \in \mathrm{pre}^+(R\,\Gamma^* \cap \mathrm{pre}^*(q\,A\,\Gamma^*))$

Hierbei ist $\mathrm{pre}^+(U) = \{p\,w \in Q\Gamma^* \mid \exists x \in U : p\,w \Rightarrow_P^+ x\}$.

Also soll für M gelten:

$$L(M) = \bigcup \{\mathrm{pre}^*(q\,A\,\Gamma^*) \mid q \in Q, A \in \Gamma,\ q\,A \in \mathrm{pre}^+(R\,\Gamma^* \cap \mathrm{pre}^*(q\,A\,\Gamma^*))\}$$

LTL-Model-Checking in Pushdowngraphen

Einen solchen P-Multiautomaten können wir wie folgt in Polynomialzeit konstruieren.

(1) Teste für alle $q \in Q$ und $A \in \Gamma$, ob $q\,A \in \mathrm{pre}^+(R\,\Gamma^* \cap \mathrm{pre}^*(q\,A\,\Gamma^*))$.

Diese Bedingung kann wie folgt in Polynomialzeit überprüft werden:

(a) Berechne in Poly.zeit einen P-Multiautomaten M_1 für $\mathrm{pre}^*(q\,A\,\Gamma^*)$.
Beachte: $q\,A\,\Gamma^*$ ist eine reguläre Menge von Konfigurationen.

(b) Berechne in Poly.zeit einen P-Multiautomaten M_2 für $R\,\Gamma^* \cap L(M_1)$.

(c) Berechne in Poly.zeit einen P-Multiautomaten M_3 für $\mathrm{pre}^+(L(M_2))$.

(d) Teste in Poly.zeit, ob $q\,A \in L(M_3)$.

(2) Falls der Test aus (1) positiv ist, notiere den P-Multiautomaten M_1.

(3) Konstruiere in Polynomialzeit einen P-Multiautomaten für die Vereinigung aller in Schritt (2) notierten Automaten.

Dies ist der gesuchte P-Multiautomat M. □

CTL-Model-Checking in Pushdowngraphen

Auch das Model-Checking Problem für CTL ist entscheidbar für Pushdowngraphen:

Satz 51 (Walukiewicz 2000)

Für eine gegebene

- CTL-Zustandsformel $\varphi \in \mathrm{CTL}_s(\Pi)$,
- ein Pushdownsystem $P = (Q, \Gamma, \Delta, \Pi, \rho)$
- und eine Konfiguration $qu \in Q\Gamma^*$

kann in exponentieller Zeit überprüft werden, ob $(T(P), qu) \models \varphi$ gilt.

Bemerkung: Das Model-Checking Problem für CTL und Pushdownsysteme ist EXPTIME-vollständig.

Wohlquasiordnungen

Eine Quasiordnung (kurz QO) ist ein Paar $(A, \leq)$, wobei $\leq$ reflexiv und transitiv (aber nicht unbedingt antisymmetrisch) ist.

Sei $(A, \leq)$ eine Quasiordnung. Wir schreiben $a < b$, falls $a \leq b \not\leq a$.

Übung: $<$ ist transitiv!

Die Relation $\equiv \subseteq A \times A$ mit $a \equiv b \iff a \leq b \leq a$ ist offensichtlich eine Äquivalenzrelation.

$(A, \leq)$ ist eine Wohlquasiordnung (kurz WQO), falls für jede Folge $a_1, a_2, a_3, \ldots$ von Elementen aus A gilt: $\exists i < j : a_i \leq a_j$.

Lemma 52 (Erdös, Rado)

Sei $(A, \leq)$ eine WQO. Dann gilt für jede Folge $a_1, a_2, a_3, \ldots$ von Elementen aus A: $\exists i_0 < i_1 < i_2 \cdots : a_{i_0} \leq a_{i_1} \leq a_{i_2} \leq \cdots$.

Wohlquasiordnungen

Beweis: Sei $M = \{i \geq 1 \mid \forall j > i : a_i \not\leq a_j\}$.

Wäre M unendlich, so könnten wir aus M eine Teilfolge $a_{i_0}, a_{i_1}, a_{i_2}, \ldots$ mit $\forall p < q : a_{i_p} \not\leq a_{i_q}$ konstruieren (Widerspruch zu WQO).

$\rightsquigarrow$ M endlich.

Wähle ein $i \geq 1$ mit $i > m$ für alle $m \in M$.

Für jedes $j \geq i$ existiert also ein $k > j$ mit $a_j \leq a_k$.

Dies erlaubt uns, eine unendliche Folge $a_{i_0} \leq a_{i_1} \leq a_{i_3} \leq \cdots$ mit $i = i_0$ zu beginnen. □

Wohlquasiordnungen

Lemma 53

Seien $(A_1, \leq_1)$ und $(A_2, \leq_2)$ WQOs.
Definiere die "Produktordnung" auf $A_1 \times A_2$:

$$(a_1, a_2) \leq (a_1', a_2') \quad \Longleftrightarrow \quad a_1 \leq_1 a_1' \ \wedge \ a_2 \leq_2 a_2'$$

Dann ist $(A_1 \times A_2, \leq)$ wieder eine WQO.

Beweis:

Offensichtlich ist $(A_1 \times A_2, \leq)$ eine Quasiordnung.

Sei nun $(a_{1,1}, a_{2,1}), (a_{1,2}, a_{2,2}), (a_{1,3}, a_{2,3}), \ldots$ eine unendliche Folge in $A_1 \times A_2$.

Wohlquasiordnungen

Da $(A_1, \leq_1)$ eine WQO ist, gibt es nach Lemma 52 in der Folge $a_{1,1}, a_{1,2}, a_{1,3}, \ldots$ eine unendliche Teilfolge

$$a_{1,i_1} \leq_1 a_{1,i_2} \leq_1 a_{1,i_3} \leq_1 \cdots$$

$(i_1 < i_2 < i_3 < \cdots)$.

Da $(A_2, \leq_2)$ eine WQO ist, gibt es in der Folge $a_{2,i_1}, a_{2,i_2}, a_{2,i_3}, \ldots$ Indizes $k < \ell$ mit $a_{2,i_k} \leq_2 a_{2,i_\ell}$.

$\rightsquigarrow$ $(a_{1,i_k}, a_{2,i_k}) \leq (a_{1,i_\ell}, a_{2,i_\ell})$ und $i_k < i_\ell$. $\square$

Wohlquasiordnungen

Beispiel:

Sei $k \geq 1$ und betrachte auf $\mathbb{N}^k$ die folgende Relation $\leq$:

$$(a_1, \ldots, a_k) \leq (b_1, \ldots, b_k) \quad \Longleftrightarrow \quad \forall 1 \leq i \leq k : a_i \leq b_i$$

Da $(\mathbb{N}, \leq)$ offensichtlich eine WQO ist, folgt aus Lemma 53 mit Induktion über k sofort:

Satz 54 (Dickson, 1913)

Für jedes $k \geq 1$ ist $(\mathbb{N}^k, \leq)$ eine WQO.

Wohlquasiordnungen

Eine Menge $I \subseteq A$ ist nach oben abgeschlossen, falls gilt:

$$\forall a \in I \, \forall b \in A : a \leq b \rightarrow b \in I.$$

Für $a \in A$ definieren wir $\uparrow a = \{b \in A \mid a \leq b\}$.

Eine Basis einer nach oben abgeschlossenen Menge I ist eine Menge $B \subseteq I$ mit $I = \bigcup_{a \in B} \uparrow a$.

Lemma 55 (Higman 1952)

Sei $(A, \leq)$ eine WQO. Jede nach oben abgeschlossene Menge $I \subseteq A$ hat eine endliche Basis.

Wohlquasiordnungen

Beweis:

Definiere $C = \{c \in I \mid \forall a \in I : a \not\leq c\}$.

Dann ist C eine Vereinigung von Äquivalenzklassen von $\equiv$:

Sei $c \in C$ und $b \equiv c$, d. h. $b \leq c \leq b$ (insbesondere $b \in I$)

Angenommen $b \notin C$, d. h. es gibt ein $a \in I$ mit $a \leq b$ und $b \not\leq a$.

Dann gilt $a \leq b \leq c$ (d. h. $a \leq c$) und $c \not\leq a$ (sonst würde $b \leq c \leq a$ gelten), d. h. $a < c$, ein Widerspruch zu $c \in C$.

Also gilt $b \in C$ und C ist in der Tat eine Vereinigung von Äquivalenzklassen von $\equiv$.

Sei nun $B \subseteq C$ eine Menge von Repräsentanten der Äquivalenzklassen $\{[c]_{\equiv} \mid c \in C\}$, d.h.

$$C = \{c \mid \exists b \in B : b \equiv c\} \text{ und } \forall b_1, b_2 \in B : b_1 = b_2 \Leftrightarrow b_1 \equiv b_2$$

Wohlquasiordnungen

Behauptung: B ist eine Basis von I.

Sei hierzu $a \in I$ beliebig.

Angenommen $\forall b \in B : b \not\leq a$.

$\rightsquigarrow \quad \forall c \in C : c \not\leq a$.

Wir definieren eine unendliche Folge $a = a_0 > a_1 > \cdots$ mit $a_i \in I$ (widerspricht WQO) induktiv wie folgt:

Seien $a_0, \ldots, a_n \in I$ schon definiert. Dann gilt insbesondere $a = a_0 \geq a_n$.

Da $c \not\leq a$ für alle $c \in C$ muss $a_n \in I \setminus C$ gelten.

Also existiert ein $a_{n+1} < a_n$ mit $a_{n+1} \in I$, was die Behauptung beweist.

Wohlquasiordnungen

Angenommen B wäre unendlich.

Seien $b_0, b_1, b_2, \ldots \in B \subseteq C \subseteq I$ mit $b_i \neq b_j$ für $i \neq j$.

$\rightsquigarrow$ $\forall i < j : b_i \not\leq b_j \wedge b_j \not\leq b_i$.

$\rightsquigarrow$ $\forall i < j : b_i \leq b_j \leq b_i \vee b_i \not\leq b_j \not\leq b_i$.

$\rightsquigarrow$ $\forall i < j : b_i \equiv b_j \vee b_i \not\leq b_j \not\leq b_i$.

$\rightsquigarrow$ $\forall i < j : b_i \not\leq b_j \not\leq b_i$ (da $b_i \not\equiv b_j$ für $i \neq j$).

$\rightsquigarrow$ Widerspruch zu WQO. $\square$

Wohlquasiordnungen

Lemma 56

Sei $(A, \leq)$ eine WQO und seien $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots \subseteq A$ nach oben abgeschlossen. Dann existiert ein $k \geq 0$ mit $I_k = I_{k+1} = I_{k+2} = \cdots$.

Beweis:

Angenommen es gibt $n_0 < n_1 < n_2 < \cdots$ mit $I_{n_0} \subsetneq I_{n_1} \subsetneq I_{n_2} \subsetneq \cdots$.

Für alle $i \geq 1$ sei $a_i \in I_{n_i} \setminus I_{n_{i-1}}$.

Da $(A, \leq)$ eine WQO ist, existieren $i < j$ mit $a_i \leq a_j$.

$\rightsquigarrow \quad a_j \in \uparrow a_i \subseteq I_{n_i}$

Dies widerspricht aber $a_j \notin I_{n_{j-1}} \supseteq I_{n_i}$. □

Wohlstrukturierte Transitionsgraphen

Ein wohlstrukturierter Transitionsgraph (kurz WTG) ist ein Tupel $T = (V, E, \Pi, \pi, \leq)$, wobei

- (V, E, Π, π) ein Transitionsgraph ist,
- $(V, \leq)$ ist eine WQO, und
- $\leq$ ist aufwärts-kompatibel mit E, d.h.

$$\forall (u, u') \in E, v \in V : u \leq v \;\Rightarrow\; \exists v' \in V : (v, v') \in E^* \wedge u' \leq v'$$

Die Komponenten Π und π (Markierung der Knoten mit Propositionen) sind im folgenden nicht wichtig, weshalb wir diese Komponenten weglassen, d.h. ein WTG ist einfach ein Tupel $T = (V, E, \leq)$.

Wohlstrukturierte Transitionsgraphen

Beispiel:

Ein Vektoradditionssystem ist ein endliche Menge von Paaren $S \subseteq \mathbb{N}^k \times \mathbb{N}^k$. Die Zahl k ist die Dimension von S.

Sei $T(S) = (\mathbb{N}^k, \Rightarrow_S, \leq)$, wobei

$$\Rightarrow_S = \{(a, a') \in \mathbb{N}^k \times \mathbb{N}^k \mid \exists (\ell, r) \in S : a \geq \ell, a' = a - \ell + r\}.$$

Dann ist $T(S)$ ein WTG:

- Satz 54 $\rightsquigarrow \leq$ ist eine WQO.
- Gelte $a \Rightarrow_S a'$ und sei $a \leq b$.

  Dann existiert $(\ell, r) \in S$ mit $a \geq \ell$ und $a' = a - \ell + r$.

  $\rightsquigarrow \; b \Rightarrow_S b - \ell + r \geq a - \ell + r = a'$.

Die Saturierungsmethode

Sei $T = (V, E, \leq)$ ein WTG im Folgenden.

Satz 57

Sei $I \subseteq V$ nach oben abgeschlossen. Dann ist auch $\text{pre}_T^*(I)$ nach oben abgeschlossen.

Beweis:

Sei $u \in \text{pre}_T^*(I)$ und $v \in V$ mit $u \leq v$.

Sei $u' \in I$ so, dass $(u, u') \in E^*$.

Aufwärts-Kompatibilität $\rightsquigarrow$ es existiert $v' \in V$ mit $u' \leq v', (v, v') \in E^*$.

I nach oben abgeschlossen $\rightsquigarrow$ $v' \in I$.

$\rightsquigarrow$ $v \in \text{pre}_T^*(I)$. □

Die Saturierungsmethode

Der WTG T hat effektive Pred-Basen, falls ein Algorithmus für das folgende Problem existiert (macht nur Sinn, falls V abzählbar ist):

EINGAGE: $v \in V$.

AUSGABE: eine endliche Basis für $\uparrow \mathrm{pre}(\uparrow v)$.

Wir bezeichnen im Folgenden mit $\mathrm{pb}(v)$ eine endliche Basis für $\uparrow \mathrm{pre}(\uparrow v)$.

Für $U \subseteq V$ sei $\mathrm{pb}(U) = \bigcup_{v \in U} \mathrm{pb}(v)$.

Falls U endlich ist, ist dies eine endliche Basis für $\uparrow \mathrm{pre}(\uparrow U)$.

Sei nun $T = (V, E, \leq)$ ein WTG mit effektiven Pred-Basen.

Wir wollen für eine nach oben abgeschlossene Menge $I \subseteq V$ (gegeben durch eine endliche Basis) eine Basis für die nach Satz 57 nach oben abgeschlossene Menge $\mathrm{pre}^*(I)$ berechnen.

Die Saturierungsmethode

Sei also $B \subseteq I$ eine endliche Basis von I.

Wir definieren endliche Mengen $K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq V$ induktiv:

$$\begin{aligned} K_0 &= B \\ K_{n+1} &= K_n \cup \mathrm{pb}(K_n) \end{aligned}$$

Wegen Lemma 56 gibt es ein kleinstes $m \geq 0$ mit:

$$\uparrow K_0 \subseteq \uparrow K_1 \subseteq \cdots \subseteq \uparrow K_{m-1} \subseteq \uparrow K_m = \uparrow K_{m+1} = \uparrow K_{m+2} = \cdots .$$

Lemma 58

$\uparrow K_m = \uparrow \bigcup_{i \geq 0} K_i$

Beweis: Es gilt

$$\uparrow \bigcup_{i \geq 0} K_i = \bigcup_{i \geq 0} \uparrow K_i = \bigcup_{i \geq 0}^{m} \uparrow K_i = \uparrow K_m.$$

Die Saturierungsmethode

Lemma 59

$\uparrow \bigcup_{i \geq 0} K_i = \mathrm{pre}^*(I)$

Beweis:

$\subseteq$: Es genügt $\uparrow K_i \subseteq \mathrm{pre}^*(I)$ für alle $i \geq 0$ zu zeigen.

Da $\mathrm{pre}^*(I)$ nach oben abgeschlossen ist (Satz 57), genügt es $K_i \subseteq \mathrm{pre}^*(I)$ zu zeigen.

Induktion über $i \geq 0$:

IA: $i = 0$.

$K_0 = B \subseteq I \subseteq \mathrm{pre}^*(I)$

IS: Sei $i \geq 0$.

$\rightsquigarrow \quad K_{i+1} = K_i \cup \mathrm{pb}(K_i)$.

IH $\rightsquigarrow \quad K_i \subseteq \mathrm{pre}^*(I)$.

Die Saturierungsmethode

Wir müssen also noch $\mathrm{pb}(K_i) \subseteq \mathrm{pre}^*(I)$ zeigen.

Da $\mathrm{pb}(K_i)$ eine (endliche) Basis für $\uparrow\mathrm{pre}(\uparrow K_i)$ ist, genügt es $\uparrow\mathrm{pre}(\uparrow K_i) \subseteq \mathrm{pre}^*(I)$ zu zeigen.

Es gilt:

$$\begin{aligned} K_i \subseteq \mathrm{pre}^*(I) &\rightsquigarrow \uparrow K_i \subseteq \mathrm{pre}^*(I) \\ &\rightsquigarrow \mathrm{pre}(\uparrow K_i) \subseteq \mathrm{pre}^*(I) \\ &\rightsquigarrow \uparrow\mathrm{pre}(\uparrow K_i) \subseteq \mathrm{pre}^*(I) \end{aligned}$$

$\supseteq$: Sei $v \in \mathrm{pre}^*(I)$.

Dann gibt es ein $n \geq 0$ und ein $u \in I$ mit $(v, u) \in E^n$.

Wir zeigen $v \in \uparrow K_n$ durch Induktion über $n \geq 0$.

IA: $n = 0$.

$\rightsquigarrow \quad v \in I \; = \uparrow B \; = \uparrow K_0$

Die Saturierungsmethode

IS: Sei $n \geq 1$.

Dann gibt es ein $v' \in \mathrm{pre}^*(I)$ mit $v \in \mathrm{pre}(v')$ und $(v', u) \in E^{n-1}$.

IH $\rightsquigarrow$ $v' \in \uparrow K_{n-1}$

$\rightsquigarrow$ $v \in \mathrm{pre}(\uparrow K_{n-1}) \subseteq \uparrow \mathrm{pre}(\uparrow K_{n-1}) = \uparrow \mathrm{pb}(K_{n-1}) \subseteq \uparrow K_n$. □

Satz 60

Sei $T = (V, E, \leq)$ ein WTG mit den folgenden Eigenschaften:

- T hat effektive Pred-Basen.
- Die Relation $\leq$ ist entscheidbar.

Dann kann man für eine gegebene endliche Menge $B \subseteq V$ eine Basis für $\mathrm{pre}^*(\uparrow B)$ berechnen.

Die Saturierungsmethode

Beweis:

Sei $I = \uparrow B$ und definiere die Folge $K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots$ von endlichen Mengen (Folie 106).

Da T effektive Pred-Basen hat, können wir die Mengen $K_0, K_1, K_2, \ldots$ effektiv berechnen.

Wir machen dies so lange, bis ein m mit $\uparrow K_m = \uparrow K_{m+1}$ erreicht wird (dies muss irgendwann geschehen).

Beachte: Es gilt

$$\begin{aligned} \uparrow K_m = \uparrow K_{m+1} &\iff K_{m+1} \subseteq \uparrow K_m \\ &\iff \forall u \in K_{m+1}\, \exists v \in K_m : v \leq u. \end{aligned}$$

Da $\leq$ entscheidbar ist, können wir diese Eigenschaft effektiv überprüfen.

Lemma 58 und 59 $\rightsquigarrow$ K_m ist eine endliche Basis für $\mathrm{pre}^*(I)$. □

Das Überdeckungsproblem

Das Überdeckungsproblem ist das folgende Problem:

EINGABE: Zustände $u, v \in V$.

FRAGE: Gibt es einen Knoten $w \in V$ mit $(u, w) \in E^*$ und $v \leq w$?

Satz 61

Sei $T = (V, E, \leq)$ ein WTG mit den folgenden Eigenschaften:

- T hat effektive Pred-Basen.
- Die Relation $\leq$ ist entscheidbar.

Dann ist das Überdeckungsproblem für T entscheidbar.

Beweis:

Seien $u, v \in V$.

Berechne eine endliche Basis B für $\text{pre}^*(\uparrow v)$ (geht nach Satz 60).

Überprüfe, ob $u' \in B$ mit $u' \leq u$ existiert (geht, da $\leq$ entscheidbar). □

Das Überdeckungsproblem

Korollar

Das Überdeckungsproblem für Vektoradditionssysteme ist entscheidbar.

Beweis: Sei $S \subseteq \mathbb{N}^k \times \mathbb{N}^k$ ein Vektoradditionssystem.

Die komponentenweise Ordnung $\leq$ auf $\mathbb{N}^k$ ist offensichtlich entscheidbar.

Wir müssen also noch zeigen, dass Vektoradditionssysteme effektive Pred-Basen haben.

Definiere für Vektoren $a = (a_1, \ldots, a_k) \in \mathbb{N}^k$ und $b = (b_1, \ldots, b_k) \in \mathbb{N}^k$ das Maximum

$$\max(a, b) = (\max(a_1, b_1), \ldots, \max(a_k, b_k)).$$

Dann ist für $s \in \mathbb{N}^k$ die endliche Menge

$$B(s) = \{\max(r, s) - r + \ell \mid (\ell, r) \in S\}$$

eine endliche Basis für $\uparrow \mathrm{pre}(\uparrow s)$:

Das Überdeckungsproblem

Offensichtlich gilt $B(s) \subseteq \mathsf{pre}(\uparrow s)$.

Gilt $b \in \uparrow\mathsf{pre}(\uparrow s)$, so gibt es $(\ell, r) \in S$ und Vektoren $c, d \in \mathbb{N}^k$ mit

$$b = d + \ell + c \quad \text{und} \quad r + c \geq s.$$

Also gilt:

$$\begin{aligned} r + c \geq \max(s, r) &\rightsquigarrow c \geq \max(s, r) - r \\ &\rightsquigarrow b \geq \ell + c \geq \max(r, s) - r + \ell \in B(s) \end{aligned}$$

Also: $b \in \uparrow B(s)$. □

Die Baumsaturierungsmethode

Sei $T = (V, E, \leq)$ ein WTG und sei $v \in V$.

Wir konstruieren den Erreichbarkeitsbaum $\mathrm{RT}(v)$ wie folgt (Knoten sind mit Elementen aus V markiert):

- Die Wurzel r von $\mathrm{RT}(v)$ ist mit v markiert.
- Sei x ein Knoten von $\mathrm{RT}(v)$, der mit $v' \in V$ markiert ist.
  - Wenn auf dem Pfad von der Wurzel r zu x ein Knoten $y \neq x$ mit einer Markierung $u \leq v'$ existiert, dann ist x ein Blatt von $\mathrm{RT}(v)$.
  - Wenn auf dem Pfad von der Wurzel r zu x kein Knoten $y \neq x$ mit einer Markierung $u \leq v'$ existiert, dann hat x für jedes $v'' \in \mathrm{suc}_T(v')$ einen mit v'' markierten Kindknoten.

Der WTG T ist endlich-verzweigend, falls für jeden Knoten $v \in V$ gilt: $\mathrm{suc}_T(v)$ ist endlich.

Die Baumsaturierungsmethode

Satz 62

Für alle $v \in V$ gilt:

- In RT(v) gibt es keinen unendlichen Pfad.
- Wenn der WTG T endlich verzweigend ist, dann ist RT(v) endlich.

Beweis:

Angenommen in dem Baum RT(v) gäbe es einen unendlichen Pfad $r = x_0 \to x_1 \to x_2 \to \cdots$.

Sei $v_i \in V$ die Markierung des Baumknoten x_i.

$\rightsquigarrow \ \forall i < j : v_i \not\leq v_j$. Widerspruch zu WQO!

Also gibt es in RT(v) keine unendlichen Pfade.

Die Baumsaturierungsmethode

Sei nun T endlich verzweigend.

$\rightsquigarrow$ Jeder Knoten in $\mathrm{RT}(v)$ hat nur endlich viele Kinder ($\mathrm{RT}(v)$ ist endlich verzweigend).

$\rightsquigarrow$ $\mathrm{RT}(v)$ ist ein endlich-verzweigender Baum ohne unendliche Pfade.

Königs Lemma $\rightsquigarrow$ $\mathrm{RT}(v)$ endlich. □

Bemerkungen:

- Für den Beweis von Satz 62 haben wir die Aufwärtskompatibilität von E und $\leq$ nicht benutzt.
- Wenn (i) T endlich verzweigend ist, (ii) die Abbildung $v \mapsto \mathrm{suc}_T(v)$ berechenbar ist, und (iii) die Relation $\leq$ entscheidbar ist, dann ist die Abbildung $v \mapsto \mathrm{RT}(v)$ berechenbar.

Die Baumsaturierungsmethode

Stärkere Formen der Aufwärtskompatibilität:

- T ist stark-kompatibel, falls für alle $(u, u') \in E$, $v \geq u$ gilt:

$$\exists v' \in V : (v, v') \in E \wedge u' \leq v'$$

- T ist transitiv-kompatibel, falls für alle $(u, u') \in E$, $v \geq u$ gilt:

$$\exists v' \in V : (v, v') \in E^+ \wedge u' \leq v'$$

- T ist stotternd-kompatibel, falls für alle $(u, u') \in E$, $v_0 \geq u$ gilt:

$$\exists n \geq 1, v_1, \ldots, v_n \in V : \bigwedge_{i=0}^{n-1} ((v_i, v_{i+1}) \in E \wedge u \leq v_i) \wedge u' \leq v_n$$

- T ist reflexiv-kompatibel, falls für alle $(u, u') \in E$, $v \geq u$ gilt:

$$\exists v' \in V : (v, v') \in (E \cup \mathrm{id}_V) \wedge u' \leq v'$$

Die Baumsaturierungsmethode

Es gelten offensichtlich die folgenden Implikationen:

$$
\begin{aligned}
T \text{ stark kompatibel} &\Longrightarrow T \text{ stotternd-kompatibel} \\
&\Longrightarrow T \text{ transitiv-kompatibel} \\
&\Longrightarrow T \text{ aufwärts-kompatibel}
\end{aligned}
$$

$$
\begin{aligned}
T \text{ stark-kompatibel} &\Longrightarrow T \text{ reflexiv-kompatibel} \\
&\Longrightarrow T \text{ aufwärts-kompatibel}
\end{aligned}
$$

Ausserdem ist der WTG $T^* = (V, E^*, \leq)$ stark kompatibel.

Die Baumsaturierungsmethode: Termination

Lemma 63

Sei der WTG $T = (V, E, \leq)$ transitiv-kompatibel und sei $v \in V$.
Dann beginnt in v ein unendlicher Pfad genau dann, wenn es in $\mathrm{RT}(v)$ ein mit v' markiertes Blatt gibt, so dass $\mathrm{suc}_T(v') \neq \emptyset$ gilt.

Beweis:

"$\Rightarrow$": Angenommen in v beginnt ein unendlicher Pfad:

$$v_0 \, E \, v_1 \, E \, v_2 \, E \, \cdots, \quad v_0 = v$$

Dann gibt es in $\mathrm{RT}(v)$ einen maximalen Pfad

$$r = x_0 \to x_1 \to x_2 \to \cdots \to x_n$$

(r ist die Wurzel, x_n ist ein Blatt), so dass x_i mit v_i beschriftet ist.

$\rightsquigarrow$ x_n ist ein mit v_n markiertes Blatt und $\mathrm{suc}_T(v_n) \neq \emptyset$.

Die Baumsaturierungsmethode: Termination

"$\Leftarrow$": Angenommen in $\mathrm{RT}(v)$ existiert ein mit u_1 markiertes Blatt x, so dass $\mathrm{suc}_T(u_1) \neq \emptyset$ gilt.

$\rightsquigarrow$ auf dem Pfad von der Wurzel r nach x existiert ein Knoten $y \neq x$ mit einer Markierung $u_0 \leq u_1$.

Es gilt: $v\, E^*\, u_0\, E^+\, u_1$.

Aus $(u_0, u_1) \in E^+$ und $u_0 \leq u_1$ folgt mit der transitiven Kompatibilität von T:

$$\exists u_2 \in V : (u_1, u_2) \in E^+ \ \wedge \ u_1 \leq u_2$$

Transitive Kompatibilität nochmals angewendet liefert:

$$\exists u_3 \in V : (u_2, u_3) \in E^+ \ \wedge \ u_2 \leq u_3$$

Allgemein erhalten wir Knoten $u_i \in V$ $(i \geq 0)$ mit $(u_i, u_{i+1}) \in E^+$ und $u_i \leq u_{i+1}$ für alle $i \geq 0$.

$\rightsquigarrow$ $v\, E^*\, u_0\, E^+\, u_1\, E^+\, u_2\, E^+\, u_3 \cdots$

Also beginnt in v ein unendlicher Pfad. □

Die Baumsaturierungsmethode: Termination

Satz 64

Sei T ein WTG mit folgenden Eigenschaften:

1. T ist endlich verzweigend.
2. $\leq$ ist entscheidbar.
3. Die Abbildung $v \mapsto \mathrm{suc}_T(v)$ ist berechenbar.
4. T ist transitiv-kompatibel.

Dann gibt es einen Algorithmus, der für einen gegebenen Knoten $v \in V$ entscheidet, ob in v ein unendlicher Pfad beginnt.

Beweis: Sei $v \in V$ gegeben.

Wegen (1), (2) und (3) können wir $\mathrm{RT}(v)$ effektiv berechnen.

Wegen (4) und Lemma 63 beginnt in v ein unendlicher Pfad genau dann, wenn in $\mathrm{RT}(v)$ ein mit v' markiertes Blatt existiert, so dass $\mathrm{suc}_T(v') \neq \emptyset$.

Wegen (3) ist letzteres entscheidbar. □

Die Baumsaturierungsmethode: Termination

Korollar

Für ein gegebenes Vektoradditionssystem $S \subseteq \mathbb{N}^k \times \mathbb{N}^k$ und $s \in \mathbb{N}^k$ kann man effektiv entscheiden, ob in s ein unendlicher $\Rightarrow_S$-Pfad beginnt.

Beweis:

Für ein Vektoradditionssystem S gilt offensichtlich:

1. $T(S)$ ist endlich verzweigend.
2. $\leq$ ist entscheidbar.
3. Die Abbildung $v \mapsto \mathrm{suc}_{T(S)}(v)$ ist berechenbar.
4. $T(S)$ ist transitiv-kompatibel (sogar stark-kompatibel). □

Die Baumsaturierungsmethode: control-state maintainability

Lemma 65

Sei $T = (V, E, \leq)$ ein stotternd-kompatibler WTG, sei $I \subseteq V$ nach oben abgeschlossen, und $v \in I$. Dann sind folgende Aussagen äquivalent:

1. In v beginnt ein maximaler (endlicher oder unendlicher) E-Pfad, dessen Zustände alle in I liegen.
2. In RT(v) existiert ein maximaler Pfad (von der Wurzel zu einem Blatt), dessen Markierungen alle in I liegen.

Beweis:

(1) $\Rightarrow$ (2): offensichtlich

(2) $\Rightarrow$ (1): Sei $r = x_0 \to x_1 \to \cdots \to x_n$ ein max. Pfad in RT(v) (r ist die Wurzel, x_n ist ein Blatt), so dass x_i mit $v_i \in I$ markiert ist ($v = v_0$).

$\rightsquigarrow \quad v = v_0 \, E \, v_1 \, E \, v_2 \, E \cdots E \, v_n$

Die Baumsaturierungsmethode: control-state maintainability

Fall 1: $\mathrm{suc}_T(v_n) = \emptyset$.

$\rightsquigarrow$ $v = v_0\, E\, v_1\, E\, v_2\, E \cdots E\, v_n$ ist ein max. E-Pfad mit $v_0, \ldots, v_n \in I$.

Fall 2: $\mathrm{suc}_T(v_n) \neq \emptyset$.

Da x_n ein Blatt ist, existiert ein $p < n$ mit $v_p \leq v_n$.

Wir konstruieren nun induktiv einen unendlichen Pfad

$$v = u_0\, E\, u_1\, E\, u_2\, E \cdots ,$$

so dass jeder Knoten u_i größer als einer der Zustände $v_j \in I$ ist.

Dies impliziert dann $u_i \in I$ für alle $i \geq 0$, da I nach oben abgeschlossen ist.

IA: $u_0 = v_0 = v$.

IS: Seien $u_0, \ldots, u_k$ bereits konstruiert.

Dann gilt $u_k \geq v_j$ für ein $0 \leq j \leq n$.

Die Baumsaturierungsmethode: control-state maintainability

Fall 2.1: $j < n$.

Es gilt $(v_j, v_{j+1}) \in E$.

Mit $v_j \leq u_k$ und der stotternden Kompatibilität von T folgt:

$$\exists \ell > k, u_{k+1}, \ldots, u_\ell \in V : \bigwedge_{i=k}^{\ell-1} ((u_i, u_{i+1}) \in E \wedge u_i \geq v_j) \wedge u_\ell \geq v_{j+1}$$

Wir können also den Pfad $u_0 \, E \, u_1 \, E \, u_2 \, E \cdots E \, u_k$ verlängern zu: $u_0 \, E \, u_1 \, E \, u_2 \, E \cdots E \, u_\ell$.

Fall 2.2: $j = n$.

Es gilt $u_k \geq v_j = v_n \geq v_p$ mit $p < n$.

Wir können deshalb mit $p = j$ wieder zu Fall 2.1 gehen. □

Die Baumsaturierungsmethode: control-state maintainability

Satz 66

Sei $T = (V, E, \leq)$ ein WTG mit folgenden Eigenschaften:

1. T ist endlich verzweigend.
2. $\leq$ ist entscheidbar.
3. Die Abbildung $v \mapsto \mathrm{suc}_T(v)$ ist berechenbar.
4. T ist stotternd-kompatibel.

Dann kann man folgendes Problem effektiv entscheiden (control-state maintainability):

EINGABE: $v \in V$ und eine endliche Menge $Q \subseteq V$.

FRAGE: Beginnt in v ein maximaler E-Pfad, so dass für jeden Knoten u auf diesem Pfad gilt: $\exists q \in Q : q \leq u$?

Die Baumsaturierungsmethode: control-state maintainability

Beweis:

(1), (2) und (3) $\rightsquigarrow$ RT(v) kann effektiv berechnet werden.

Wegen (4) und Lemma 65 genügt es zu überprüfen, ob in dem endlichen Baum RT(v) ein maximaler Pfad existiert, dessen Markierungen alle zu $\uparrow Q$ gehören.

Da Q endlich ist, ist dies mit (2) entscheidbar. □

Die Baumsaturierungsmethode: Beschränktheit

Ein WTG $T = (V, E, \leq)$ ist strikt kompatibel, falls für alle $u, u', v \in V$ gilt:

$$(u < v \wedge (u, u') \in E) \Rightarrow \exists v' : (u' < v' \wedge (v, v') \in E^*) \tag{9}$$

Beachte:

- Da T ein WTG ist, müssen E und $\leq$ auch im gewöhnlichen Sinne kompatibel sein, d.h für alle $u, u', v \in V$ gilt:

$$(u \leq v \wedge (u, u') \in E) \Rightarrow \exists v' : (u' \leq v' \wedge (v, v') \in E^*) \tag{10}$$

- Falls $\leq$ eine partielle Ordnung ist (d.h. $\leq$ ist auch antisymmetrisch), so gilt (9) $\Rightarrow$ (10).

Die Baumsaturierungsmethode: Beschränktheit

Lemma 67

Sei $T = (V, E, \leq)$ ein WTG mit folgenden Eigenschaften:

- T ist endlich verzweigend.
- T ist strikt kompatibel.
- $\leq$ ist eine partielle Ordnung.

Dann sind für alle $v \in V$ die beiden folgenden Eigenschaften äquivalent:

1. $\mathrm{suc}_T^*(v)$ ist unendlich.
2. In $\mathrm{RT}(v)$ existiert ein mit v_1 markiertes Blatt x, so dass auf dem Pfad von der Wurzel zu x ein mit $v_0 < v_1$ markierter Knoten $y \neq x$ existiert.

Die Baumsaturierungsmethode: Beschränktheit

Beweis:

(1) $\Leftarrow$ (2):

Es gilt $v\, E^*\, v_0\, E^+\, v_1$ und $v_0 < v_1$.

Aus der strikten Kompatibilität von T folgt die Existenz eines $v_2 > v_1$ mit $v_1\, E^*\, v_2$.

Aus $v_1 < v_2$ folgt $v_1 \neq v_2$ und damit $v_1\, E^+\, v_2$.

Durch Wiederholung dieses Arguments erhalten wir Zustände $v_0 < v_1 < v_2 < \cdots$ mit

$$v\, E^*\, v_0\, E^+\, v_1\, E^+\, v_2\, E^+\, v_3\, E^+ \cdots .$$

Für $i < j$ gilt $v_i < v_j$ und damit $v_i \neq v_j$.

$\rightsquigarrow$ $\mathrm{suc}_T^*(v)$ ist unendlich.

Die Baumsaturierungsmethode: Beschränktheit

(2) $\Leftarrow$ (1): Sei $\mathrm{suc}_T^*(v)$ unendlich.

Behauptung: Es existiert ein unendlicher Pfad

$$v = v_0 \, E \, v_1 \, E \, v_2 \, E \, \cdots$$

mit $v_i \neq v_j$ für $i \neq j$.

Betrachte hierzu den Baum B mit der Knotenmenge

$$\{u_0 u_1 \cdots u_n \in V^* \mid u_0 = v, \bigwedge_{i \neq j} u_i \neq u_j, \bigwedge_{i=0}^{n-1} (u_i, u_{i+1}) \in E\}$$

und allen Kanten der Form $u_0 u_1 \cdots u_n \to u_0 u_1 \cdots u_n u_{n+1}$.

Da $\mathrm{suc}_T^*(v)$ unendlich ist, und jeder Knoten in $\mathrm{suc}_T^*(v)$ durch einen zyklenfreien Pfad erreicht werden kann, ist B ein unendlicher Baum.

Da T endlich verzweigend ist, ist auch B endlich verzweigend.

Königs Lemma $\rightsquigarrow$ B hat einen unendlichen Pfad.

Die Baumsaturierungsmethode: Beschränktheit

Dieser unendliche Pfad liefert einen unendlichen Pfad $v = v_0\, E\, v_1\, E\, v_2\, E\, \cdots$ mit $v_i \neq v_j$ für $i \neq j$.

Im Baum $\mathrm{RT}(v)$ existiert ein maximaler Pfad $x_0 \to x_1 \to \cdots \to x_n$, so dass x_i mit v_i markiert ist (x_n ist ein Blatt).

Da $\mathrm{suc}_T(v_n) \neq \emptyset$, existiert ein $j < n$ mit $v_j \leq v_n$.

Da $v_j \neq v_n$ und $\leq$ eine partielle Ordnung ist, folgt $v_j < v_n$. □

Die Baumsaturierungsmethode: Beschränktheit

Satz 68

Sei $T = (V, E, \leq)$ ein WTG mit folgenden Eigenschaften:

1. T ist endlich verzweigend.
2. $\leq$ ist entscheidbar und eine partielle Ordnung.
3. Die Abbildung $v \mapsto \mathrm{suc}_T(v)$ ist berechenbar.
4. T ist strikt kompatibel.

Dann gibt es einen Algorithmus, der für einen gegebenen Knoten $v \in V$ entscheidet, ob $\mathrm{suc}_T^*(v)$ unendlich ist.

Die Baumsaturierungsmethode: Beschränktheit

Beweis:

(1), (2) und (3) $\rightsquigarrow$ RT(v) kann effektiv berechnet werden.

Wegen (2), (4) und Lemma 67 genügt es zu überprüfen, ob in RT(v) ein mit u markiertes Blatt x existiert, so dass auf dem Pfad von der Wurzel zu x ein mit $u' < u$ markierter Knoten $y \neq x$ liegt.

Da $\leq$ (und damit $<$) entscheidbar ist, kann dies effektiv überprüft werden. □

Korollar

Für ein gegebenes Vektoradditionssystem $S \subseteq \mathbb{N}^k \times \mathbb{N}^k$ und $s \in \mathbb{N}^k$ kann man effektiv entscheiden, ob $\mathrm{suc}^*_{T(S)}(s)$ unendlich ist.

Erweiterungen von Vektoradditionssystemen (Petrinetzen)

Ein affines Vektoradditionssystem S der Dimension k ist eine endliche Menge von Tripeln der Form (ℓ, r, R), wobei

- $\ell, r \in \mathbb{N}^k$ und
- $R \in \mathbb{N}^{k \times k}$ (d.h. R ist eine $(k \times k)$-Matrix über $\mathbb{N}$).

Für ein affines Vektoradditionssystem S definieren wir die Relation $\Rightarrow_S \subseteq \mathbb{N}^k \times \mathbb{N}^k$ wie folgt:

$$a \Rightarrow_S a' \quad \Longleftrightarrow \quad \exists c \in \mathbb{N}^k \; \exists (\ell, r, R) \in S : a = c + \ell \wedge a' = c \cdot R + r$$

Äquivalent: $\exists (\ell, r, R) \in S : a \geq \ell \wedge a' = (a - \ell) \cdot R + r$

Sei wieder $T(S) = (\mathbb{N}^k, \Rightarrow_S, \leq)$.

Erweiterungen von Vektoradditionssystemen (Petrinetzen)

Affine Vektoradditionssysteme verallgemeinern Transfer-Petrinetze und Reset-Petrinetze:

- In Transfer-Petrinetzen können Transitionen zusätzlich zu ihrer normalen Wirkung noch den kompletten Inhalt (= alle Token) einer Stelle in eine andere Stelle übertragen.
- In Reset-Petrinetzen können Transitionen zusätzlich zu ihrer normalen Wirkung noch den kompletten Inhalt (= alle Token) einer Stelle löschen.

Lemma 69

Sei S ein affines Vektoradditionssystem. Dann ist $T(S)$ ein stark-kompatibler WTG mit effektiven Pred-Basen.

Erweiterungen von Vektoradditionssystemen (Petrinetzen)

Beweis:

Gelte $a \Rightarrow_S a'$ und sei $b \geq a$.

$\rightsquigarrow$ es existiert $(\ell, r, R) \in S$ mit $a \geq \ell$ und $a' = (a - \ell) \cdot R + r$.

$\rightsquigarrow$ $b \geq \ell$ und $b \Rightarrow_S (b - \ell) \cdot R + r =: b'$.

Da die Matrix R nur Einträge ≥ 0 hat, folgt aus $a \leq b$:

$$a' = (a - \ell) \cdot R + r \leq (b - \ell) \cdot R + r = b'$$

Dies zeigt die starke Kompatibilität.

Wir müssen noch die Existenz effektiver Pred-Basen zeigen.

Sei hierzu $s \in \mathbb{N}^k$.

Wir müssen eine endliche Basis von

$$\uparrow \mathsf{pre}(\uparrow s) = \{x \in \mathbb{N}^k \mid \exists y \in \mathbb{N}^k \bigvee_{(\ell,r,R) \in S} x \geq y \geq \ell \wedge (y - \ell) \cdot R + r \geq s\}$$

berechnen.

Erweiterungen von Vektoradditionssystemen (Petrinetzen)

Dies ist mittels linearer Algebra möglich.

Alternativ (aber weniger effizient): Schreibe eine prädikatenlogische Formel $\varphi(x_1, \ldots, x_k)$ über der Struktur $(\mathbb{N}, +)$, so dass für alle $v \in \mathbb{N}^k$ gilt:

$$(\mathbb{N}, +) \models \varphi(v) \iff v \in \uparrow \mathrm{pre}(\uparrow s)$$

In der Formel φ dürfen wir ruhig $\leq$ und Konstanten aus $\mathbb{N}$ benutzen, denn für alle $x, y \in \mathbb{N}$ gilt:

- $x \leq y \iff \exists z : x + z = y$
- $x = 0 \iff x + x = x$
- $x < y \iff \exists z : x + z = y \wedge z \neq 0$
- $y = x + 1 \iff x < y \wedge \neg \exists z : x < z < y$
- Mit Formeln für $x = 0$ und $y = x + 1$ lassen sich dann beliebige Konstanten $n \in \mathbb{N}$ definieren.

Erweiterungen von Vektoradditionssystemen (Petrinetzen)

Es gibt einen Algorithmus, der für eine prädikatenlogische Formel entscheidet, ob sie in der Struktur $(\mathbb{N}, +)$ wahr ist (Entscheidbarkeit der sogenannten Presburger-Arithmetik, siehe Logik II).

Also können wir eine endliche Basis für $\uparrow \mathrm{pre}(\uparrow s)$ wie folgt berechnen:

- Setze $B := \emptyset$.
- Zähle alle Vektoren $v \in \mathbb{N}^k$ systematisch auf.

  Falls $(\mathbb{N}, +) \models \varphi(v)$ (d.h. $v \in \uparrow \mathrm{pre}(\uparrow s)$) gilt, setze $B := B \cup \{v\}$.
- Stoppe, sobald

$$(\mathbb{N}, +) \models \forall x_1 \cdots \forall x_k : \varphi(x_1, \ldots, x_k) \rightarrow \bigvee_{s \in B} s \leq (x_1, \ldots, x_k)$$

  gilt. Dann ist $\uparrow \mathrm{pre}(\uparrow s) = \uparrow B$ und B ist eine endliche Basis von $\uparrow \mathrm{pre}(\uparrow s)$.

Erweiterungen von Vektoradditionssystemen (Petrinetzen)

Korollar

Für ein gegebenes affines Vektoradditionssystem S sind folgende Eigenschaften entscheidbar:

1. Überdeckungsproblem: Gegeben $s, t \in \mathbb{N}^k$, gilt $\exists x \in \mathbb{N}^k : s \Rightarrow_S^* x \geq t$?
2. Termination: Gegeben $s \in \mathbb{N}^k$, startet in s ein unendlicher $\Rightarrow_S$-Pfad?
3. Control-state maintainability: Gegeben $s \in \mathbb{N}^k$ und eine endliche Teilmenge $Q \subseteq \mathbb{N}^k$, beginnt in s ein maximaler $\Rightarrow_S$-Pfad, der vollständig in $\uparrow Q$ enthalten ist?

Beweis: Mit Lemma 69 ergibt sich Aussage

- (1) aus Satz 61,
- (2) aus Satz 64, und
- (3) aus Satz 66.

Für (2) und (3) muss man noch beachten, dass $T(S)$ endlich verzweigend ist, und die Abbildung $s \mapsto \mathrm{suc}_{T(S)}(s)$ berechenbar ist. □

Erweiterungen von Vektoradditionssystemen (Petrinetzen)

Beachte: Für ein affines Vektoradditionssystem S ist $T(S)$ im Allgemeinen nicht strikt kompatibel.

Beispiel: Sei $S = \{(0,0,0)\}$. Dies ist ein affines Vektoradditionssystem in der Dimension 1; 0 bezeichnet hierbei die (1×1)-dimensionale 0-Matrix.

$\rightsquigarrow$ $T(S) = (\mathbb{N}, \{(n,0) \mid n \geq 0\}, \leq)$ ist nicht strikt kompatibel.

Ein strikt affines Vektoradditionssystem (in der Dimension k) ist ein affines Vektoradditionssystem S, so dass für alle $(\ell, r, R) \in S$ und alle $1 \leq i \leq k$ gilt: $\exists j : R_{i,j} > 0$ (in jeder Zeile existiert ein von Null verschiedener Eintrag).

Lemma 70

Sei S ein strikt affines Vektoradditionssystem. Dann ist $T(S)$ ein WTG mit strikt starker Kompatibilität.

Erweiterungen von Vektoradditionssystemen (Petrinetzen)

Beweis:

Wir müssen Folgendes zeigen:

Sei $R \in \mathbb{N}^{k \times k}$ mit $\forall i \exists j : R_{i,j} > 0$ und seien $a, b \in \mathbb{N}^k$ mit $a < b$.
Dann gilt $a \cdot R < b \cdot R$.

Hierfür genügt es offenbar zu zeigen:

Sei $R \in \mathbb{N}^{k \times k}$ mit $\forall i \exists j : R_{i,j} > 0$ und sei $a \in \mathbb{N}^k$ mit $a > 0$.
Dann gilt $a \cdot R > 0$.

Wegen $a > 0$ gibt es ein i mit $a_i > 0$.

Da $\forall i \exists j : R_{i,j} > 0$ gilt, gibt es ein j mit $R_{i,j} > 0$.

Dann gilt:

$$(a \cdot R)_j = \sum_{n=1}^{k} a_n \cdot R_{n,j} \geq a_i \cdot R_{i,j} > 0.$$

$\rightsquigarrow \quad a \cdot R > 0.$ □

Erweiterungen von Vektoradditionssystemen (Petrinetzen)

Korollar

Für ein gegebenes strikt affines Vektoradditionssystem S und $s \in \mathbb{N}^k$ kann man effektiv entscheiden, ob $\mathrm{suc}^*_{T(S)}(s)$ endlich ist.

D.h. Beschränktheit ist entscheidbar für strikt affine Vektoradditionssysteme.

Beweis:

Folgt aus Lemma 70 und Satz 68. □

Bemerkungen: Beschränktheit ist unentscheidbar für affine Vektoradditionssysteme (sogar für Reset-Petrinetze), siehe z. B. Dufourd, Schnoebelen, Jancar. Boundedness of Reset P/T Nets. In Proceedings of ICALP 1999, Lecture Notes in Computer Science 1644, S. 301–310, Springer 1999.

WQOs auf Wörtern

Sei Σ ein endliches Alphabet.

Für Wörter $u, v \in \Sigma^*$ schreiben wir $u \preceq v$ genau dann, wenn $u = a_1 a_2 \cdots a_n$ für $a_1, \ldots, a_n \in \Sigma$ und $v \in \Sigma^* a_1 \Sigma^* a_2 \cdots \Sigma^* a_n \Sigma^*$ (u ist ein Teilwort von v).

Satz 71 (Higman, 1952)

$(\Sigma^*, \preceq)$ ist eine WQO.

Beweis:

Offensichtlich ist $(\Sigma^*, \preceq)$ eine Quasiordnung (sogar eine partielle Ordnung).

Angenommen es gibt eine Folge $u_0, u_1, u_2, \ldots \in \Sigma^*$, so dass $u_i \not\preceq u_j$ für alle $i < j$.

Wir nennen im folgenden eine solche Folge schlecht.

WQOs auf Wörtern

Wir definieren nun eine neue schlechte Folge $v_0, v_1, v_2, \ldots \in \Sigma^+$ wie folgt:

1. v_0 sei ein kürzestes Wort, so dass eine mit v_0 beginnende schlechte Folge existiert.
2. v_1 sei ein kürzestes Wort, so dass eine mit v_0, v_1 beginnende schlechte Folge existiert.
3. v_2 sei ein kürzestes Wort, so dass eine mit v_0, v_1, v_2 beginnende schlechte Folge existiert.

   $\vdots$

Da Σ endlich ist gibt es ein Symbol $a \in \Sigma$ und eine Teilfolge $v_{i_0}, v_{i_1}, v_{i_2}, \ldots$ $(i_0 < i_1 < i_2 < \cdots)$ mit $v_{i_j} \in a\,\Sigma^*$ für alle $j \geq 0$.

Sei $v_{i_j} = a\, w_{i_j}$.

Die Folge $v_0, v_1, \ldots, v_{i_0-1}, w_{i_0}, w_{i_1}, w_{i_2}, \ldots$ ist dann wieder schlecht:

Würde $w_{i_j} \preceq w_{i_k}$ für $j < k$ gelten, so würde $v_{i_j} = a w_{i_j} \preceq a w_{i_k} = v_{i_k}$ gelten.

Widerspruch zur Wahl von v_{i_0}. □

Kanalsysteme (channel systems)

Sei $n \geq 1$. Ein n-Kanalsystem ist ein Tupel $K = (Q, \Sigma, \Delta)$ mit:

- Q ist eine endliche Menge von Zuständen,
- Σ ist ein endliches Alphabet von Nachrichtensymbolen,
- $\Delta \subseteq Q \times \{i!a, i?a \mid 1 \leq i \leq n, a \in \Sigma\} \times Q$ ist die Menge der Transitionen

Interpretation:

- $(p, i!a, q)$: Wenn der aktuelle Zustand p ist, dann kann die Nachricht a in die Warteschlange für Kanal i hinten angehängt werden. Der neue Zustand ist dann q.
- $(p, i?a, q)$: Wenn der aktuelle Zustand p ist, und in der Warteschlange für Kanal i die Nachricht a ganz vorne liegt, dann kann diese Nachricht aus dem Kanal entfernt werden und in den Zustand q gewechselt werden.

Kanalsysteme (channel systems)

Formal:

- Eine Konfiguration für K ist ein $(n+1)$-Tupel $(p, u_1, u_2, \ldots, u_n) \in Q \times (\Sigma^*)^n$.
- Für zwei Konfigurationen $(p, u_1, u_2, \ldots, u_n)$ und $(q, v_1, v_2, \ldots, v_n)$ und eine Transition $\delta = (p, c, q) \in \Delta$ schreiben wir

$$(p, u_1, u_2, \ldots, u_n) \Rightarrow_\delta (q, v_1, v_2, \ldots, v_n),$$

falls einer der beiden folgenden Fälle gilt:

- $c = i!a,\ v_i = au_i, v_j = u_j$ falls $j \neq i$
- $c = i?a,\ u_i = v_i a, v_j = u_j$ falls $j \neq i$

Es sei $\Rightarrow_K = \bigcup_{\delta \in \Delta} \Rightarrow_\delta$.

Kanalsysteme (channel systems)

Bemerkung:

Ein zu Kanalsystemen äquivalentes Modell ist das der kommunizierenden Automaten. Hierbei hat man n endliche Automaten, wobei das Alphabet für den i-ten Automaten $\{j!a, j?a \mid 1 \leq j \leq n, j \neq i, a \in \Sigma\}$ ist.

$j!a$ (bzw. $j?a$) bedeutet hierbei, dass Automat i an den Automaten j die Nachricht a sendet (bzw. vom Automaten j die Nachricht a empfängt).

Solch ein kommunizierender Automat entspricht einem $n(n-1)$-Kanalsystem mit Zustandsmenge $\prod_{i=1}^{n} Q_i$ (wobei Q_i die Zustandsmenge des i-ten Automaten ist.

Kanalsysteme (channel systems)

Zunächst die schlechten Nachrichten:

Satz 72 (Brand, Zafiropulo, 1983)

Es existiert ein 1-Kanalsystem $K = (Q, \Sigma, \Delta)$, für welches folgendes Problem (Erreichbarkeitsproblem) unentscheidbar ist:

INPUT: Zwei Konfigurationen $c_1, c_2 \in Q \times \Sigma^*$.

FRAGE: Gilt $c_1 \Rightarrow_K^* c_2$?

Beweis:

Sei $T = (Q, \Gamma, \delta, q_0, q_f)$ eine 1-Band-Turingmaschine (mit einem einseitig-unendlichen Band), die eine unentscheidbare Menge akzeptiert (z. B. das Halteproblem).

Kanalsysteme (channel systems)

- Q ist die endliche Menge der Zustände.
- Γ ist das Bandalphabet.
- $\square \in \Gamma$ ist das Blanksymbol. Alle Bandzellen, die nicht im Bereich des Eingabestrings $w \in \Gamma^*$ liegen, enthalten zu Beginn $\square$.
- $\delta \subseteq (Q \setminus \{q_f\}) \times \Gamma \times Q \times \Gamma \times \{\rightarrow, \leftarrow\}$ ist die Übergangsrelation.
- $q_0 \in Q$ ist der Anfangszustand, $q_f \in Q$ ist der Endzustand.

Eine Konfiguration von T kann durch ein Wort $c = uqv \in \Gamma^* Q \Gamma^+$ repräsentiert werden:

- u ist der Inhalt des Bandabschnitts links vom Schreiblesekopf.
- q ist der aktuelle Zustand.
- v ist der Inhalt des Bandabschnitts rechts vom Schreiblesekopf, inklusive des gerade gelesenen Symbols.

Bei Eingabe $w \in \Gamma^*$ ist $q_0 w \square$ die initiale Konfiguration.

T terminiert genau dann, wenn der Endzustand q_f erreicht wird.

Kanalsysteme (channel systems)

Wir simulieren T durch das 1-Kanalsystem

$$K = (Q \cup \{\alpha, \omega\}, \Gamma \cup \{\widehat{a} \mid a \in \Gamma\} \cup \{\#\}, \Delta)$$

mit den folgenden Transitionen:

$$\alpha \xrightarrow{!\square} \alpha$$

$$\alpha \xrightarrow{?\# \; !\#} \alpha$$

$$\alpha \xrightarrow{!\square} q_0$$

$$p \xrightarrow{?a \; !a} p \qquad \text{für alle } p \in Q \setminus \{q_f\}, a \in \Gamma \cup \{\#\}$$

$$p \xrightarrow{?\widehat{a} \; !b \; ?c \; !\widehat{c}} q \quad \text{für alle } (p, a, q, b, \leftarrow) \in \delta, c \in \Gamma$$

$$p \xrightarrow{?c \; !\widehat{c} \; ?\widehat{a} \; !b} q \quad \text{für alle } (p, a, q, b, \rightarrow) \in \delta, c \in \Gamma$$

$$q_f \xrightarrow{?x} \omega \qquad \text{für alle } x \in \Gamma \cup \{\widehat{a} \mid a \in \Gamma\} \cup \{\#\}$$

$$\omega \xrightarrow{?x} \omega \qquad \text{für alle } x \in \Gamma \cup \{\widehat{a} \mid a \in \Gamma\} \cup \{\#\}$$

Kanalsysteme (channel systems)

Dann gilt für alle Eingaben $a_1 a_2 \cdots a_n$ ($n \geq 1, a_1, \ldots, a_n \in \Gamma$):

T akzeptiert die Eingabe $a_1 a_2 \cdots a_n \iff (\alpha, \widehat{a_1} a_2 \cdots a_n \#) \Rightarrow_K^* (\omega, \varepsilon)$.

Intuition:

Eine Konfiguration von K der Form

$$(p, u_1 \widehat{a} u_2 \# v) \quad \text{bzw.} \quad (p, u \# v_1 \widehat{a} v_2)$$

repräsentiert die folgende Konfiguration der Turingmaschine T:

$$v u_1 p a u_2 \quad \text{bzw.} \quad v_1 q a v_2 u$$

Das Symbol $\#$ markiert also das Bandende, das Symbol $\widehat{a}$ markiert die aktuelle Position des Schreiblesekopf.

Kanalsysteme (channel systems)

Mit den Transitionen $\alpha \xrightarrow{!\square} \alpha$ und $\alpha \xrightarrow{?\# \; !\#} \alpha$ können beliebig viele $\square$'s am Bandanfang und Bandende eingefügt werden.

Mit den Transitionen $p \xrightarrow{?a \; !a} p$ ($p \in Q \setminus \{q_f\}$, $a \in \Gamma \cup \{\#\}$) kann das Band solange rotiert werden, bis ein Symbol der Form $\widehat{a}$ ganz oben in der Warteschlange ist.

Mit den Transitionen

$$p \xrightarrow{?\widehat{a} \; !b \; ?c \; !\widehat{c}} q \text{ für alle } (p, a, q, b, \leftarrow) \in \delta, c \in \Gamma$$

können dann Linksbewegungen von T simuliert werden, während mit den Transitionen

$$p \xrightarrow{?c \; !\widehat{c} \; ?\widehat{a} \; !b} q \text{ für alle } (p, a, q, b, \rightarrow) \in \delta, c \in \Gamma$$

Rechtsbewegungen von T simuliert werden können.

Bei der zweiten Transition muss K nichtdeterministisch raten, dass das zweitoberste Symbol in der Warteschlange von der Form $\widehat{a}$ ist. $\square$

Verlustbehaftete Kanalsysteme (lossy channel systems)

Verlustbehaftete n-Kanalsysteme modellieren kommunizierende Systeme mit unzuverlässigen Kanälen:
Nachrichten können in Kanälen verloren gehen.

Syntaktisch ist ein verlustbehaftetes n-Kanalsysteme $K = (Q, \Sigma, \Delta)$ wie ein (zuverlässiges) n-Kanalsystem definiert.

Für Konfigurationen $(p, u_1, \ldots, u_n)$ und $(q, v_1, \ldots, v_n)$ schreiben wir $(p, u_1, \ldots, u_n) \dashrightarrow_K (q, v_1, \ldots, v_n)$, falls Konfigurationen $(p, u'_1, \ldots, u'_n)$ und $(q, v'_1, \ldots, v'_n)$ existieren mit:

$$(p, u'_1, \ldots, u'_n) \Rightarrow_K (q, v'_1, \ldots, v'_n) \text{ und } \forall i \in \{1, \ldots, n\} : u'_i \preceq u_i, v_i \preceq v'_i.$$

Systemübergänge der Form $(p, u_1, \ldots, u_n) \Rightarrow_K (q, v_1, \ldots, v_n)$ nennen wir perfekte Systemübergänge

Idee hinter der Definition von $\dashrightarrow_K$:
Nachrichten können vor sowie nach perfekten Systemübergängen in Kanälen verloren gehen.

Verlustbehaftete Kanalsysteme (lossy channel systems)

Wir erweitern die Ordnung $\preceq$ von Σ^* auf die Menge aller Konfigurationen $Q \times (\Sigma^*)^n$ wie folgt:

$$(p, u_1, \ldots, u_n) \preceq (q, v_1, \ldots, v_n) \iff p = q \wedge \forall i \in \{1, \ldots, n\} : u_i \preceq v_i$$

Wegen Lemma 53 ist $(Q \times (\Sigma^*)^n, \preceq)$ eine WQO.

Satz 73

Sei $K = (Q, \Sigma, \Delta)$ ein verlustbehaftetes n-Kanalsystem.
Dann ist $T(K) = (Q \times (\Sigma^*)^n, \dashrightarrow_K, \preceq)$ ein WTG mit

1. *starker Kompatibilität und*
2. *effektiven Pred-Basen.*

Verlustbehaftete Kanalsysteme (lossy channel systems)

Beweis:

(1) $T(K)$ ist stark kompatibel: trivial

(2) $T(K)$ hat effektive Pred-Basen.

Für $(q, w_1, \ldots, w_n) \in Q \times (\Sigma^*)^n$ müssen wir eine endliche Basis von $\uparrow \mathrm{pre} \uparrow (q, w_1, \ldots, w_n)$ finden. Hierbei wird pre bezüglich $\dashrightarrow_K$ berechnet.

Für $U \subseteq Q \times (\Sigma^*)^n$ und $\delta \in \Delta$ definiere $\mathrm{pre}_\delta(U) = \{x \mid \exists y \in U : x \Rightarrow_\delta y\}$ (Menge aller Vorgänger von U bezüglich perfekter Systemübergänge mittels der Transition δ). Dann gilt:

$$\begin{aligned} \uparrow \mathrm{pre} \uparrow (q, w_1, \ldots, w_n) &= \bigcup_{\delta \in \Delta} \uparrow \mathrm{pre}_\delta \uparrow (q, w_1, \ldots, w_n) \\ &= \bigcup_{\delta \in \Delta} \uparrow \mathrm{pre}_\delta (q, \uparrow w_1, \ldots, \uparrow w_n) \end{aligned}$$

Hierbei ist $(q, \uparrow w_1, \ldots, \uparrow w_n)$ eine Abkürzung für $\{q\} \times \prod_{i=1}^{n} \uparrow w_i$.

Verlustbehaftete Kanalsysteme (lossy channel systems)

Wir berechnen eine endliche Basis für $\uparrow \mathrm{pre}_\delta(q, \uparrow w_1, \ldots, \uparrow w_n)$:

Fall 1: $\delta = (p, i!a, q)$

Dann gilt

$$\mathrm{pre}_\delta(q, \uparrow w_1, \ldots, \uparrow w_n) = \mathrm{pre}_\delta(q, \uparrow w_1, \ldots, \uparrow w_{i-1}, (\uparrow w_i) \cap a\Sigma^*, \uparrow w_{i+1}, \ldots \uparrow w_n).$$

Fall 1.1: $w_i \notin a\Sigma^*$.

Dann gilt $(\uparrow w_i) \cap a\Sigma^* = a \uparrow w_i$ und damit

$$\begin{aligned} \uparrow \mathrm{pre}_\delta(q, \uparrow w_1, \ldots, \uparrow w_n) &= \uparrow \mathrm{pre}_\delta(q, \uparrow w_1, \ldots, \uparrow w_{i-1}, a \uparrow w_i, \uparrow w_{i+1}, \ldots, \uparrow w_n) \\ &= \uparrow (p, \uparrow w_1, \ldots, \uparrow w_{i-1}, \uparrow w_i, \uparrow w_{i+1}, \ldots, \uparrow w_n) \\ &= \uparrow (p, w_1, \ldots, w_n). \end{aligned}$$

Verlustbehaftete Kanalsysteme (lossy channel systems)

Fall 1.2: $w_i = au$.

Dann gilt $(\uparrow w_i) \cap a\Sigma^* = (\uparrow(au)) \cap a\Sigma^* = a{\uparrow}u$ und damit

$$\begin{aligned}
\uparrow \mathrm{pre}_\delta(q, \uparrow w_1, \ldots, \uparrow w_n) &= \uparrow \mathrm{pre}_\delta(q, \uparrow w_1, \ldots, \uparrow w_{i-1}, a{\uparrow}u, \uparrow w_{i+1}, \ldots \uparrow w_n) \\
&= \uparrow(p, \uparrow w_1, \ldots, \uparrow w_{i-1}, \uparrow u, \uparrow w_{i+1}, \ldots, \uparrow w_n) \\
&= \uparrow(p, w_1, \ldots, w_{i-1}, u, w_{i+1}, \ldots, w_n).
\end{aligned}$$

Fall 2: $\delta = (p, i?a, q)$

$$\begin{aligned}
\uparrow \mathrm{pre}_\delta(q, \uparrow w_1, \ldots, \uparrow w_n) &= \uparrow(p, \uparrow w_1, \ldots, \uparrow w_{i-1}, (\uparrow w_i)a, \uparrow w_{i+1}, \ldots, \uparrow w_n) \\
&= (p, \uparrow w_1, \ldots, \uparrow w_{i-1}, \uparrow((\uparrow w_i)a), \uparrow w_{i+1}, \ldots, \uparrow w_n) \\
&= (p, \uparrow w_1, \ldots, \uparrow w_{i-1}, \uparrow(w_i a), \uparrow w_{i+1}, \ldots, \uparrow w_n) \\
&= \uparrow(p, w_1, \ldots, w_{i-1}, w_i a, w_{i+1}, \ldots, w_n).
\end{aligned}$$

In jedem der drei Fälle erhalten wir also eine endliche Basis für $\uparrow \mathrm{pre}_\delta(q, \uparrow w_1, \ldots, \uparrow w_n)$, die nur aus einer Konfiguration besteht. □

Verlustbehaftete Kanalsysteme (lossy channel systems)

Korollar

Für ein gegebenes verlustbehaftetes n-Kanalsystem $K = (Q, \Sigma, \Delta)$ sind folgende Eigenschaften entscheidbar:

1. Erreichbarkeit: Gilt $(p, u_1, \ldots, u_n) \dashrightarrow_K^* (q, v_1, \ldots, v_n)$ für gegebene Konfigurationen $(p, u_1, \ldots, u_n)$ und $(q, v_1, \ldots, v_n)$?
2. Termination: Beginnt in $(p, u_1, \ldots, u_n)$ ein unendlicher $\dashrightarrow_K$ Pfad für eine gegebene Konfigurationen $(p, u_1, \ldots, u_n)$?
3. Control-state maintainability

Beweis:

Offensichtlich ist für $T(K) = (Q \times (\Sigma^*)^n, \dashrightarrow_K, \preceq)$ die Abbildung $(p, u_1, \ldots, u_n) \mapsto \mathrm{suc}_{T(K)}(p, u_1, \ldots, u_n)$ berechenbar und die Ordnung $\preceq$ entscheidbar.

Zu (1): Satz 61 $\rightsquigarrow$ Überdeckungsproblem für $T(K)$ entscheidbar.

Verlustbehaftete Kanalsysteme (lossy channel systems)

Für Konfigurationen $(p, u_1, \ldots, u_n), (q, v_1, \ldots, v_n)$ gilt jedoch

$$(p, u_1, \ldots, u_n) \dashrightarrow_K^* (q, v_1, \ldots, v_n)$$

genau dann, wenn

$$\exists (r, w_1, \ldots, w_n) : (p, u_1, \ldots, u_n) \dashrightarrow_K^* (r, w_1, \ldots, w_n) \succeq (q, v_1, \ldots, v_n).$$

zu (2): Folgt aus Satz 64

zu (3): Folgt aus Satz 66 □

Bemerkung: $T(K)$ (für ein verlustbehaftetes n-Kanalsystem) ist i.A. nicht strikt kompatibel.