Hauptseminar 2006 - Model Checking

Benedikt Meurer

Fachbereich Elektrotechnik und Informatik
Universität Siegen

Übersicht

Einleitung

Ziel: Verifikation eines Systems anhand einer PLTL-Formel.

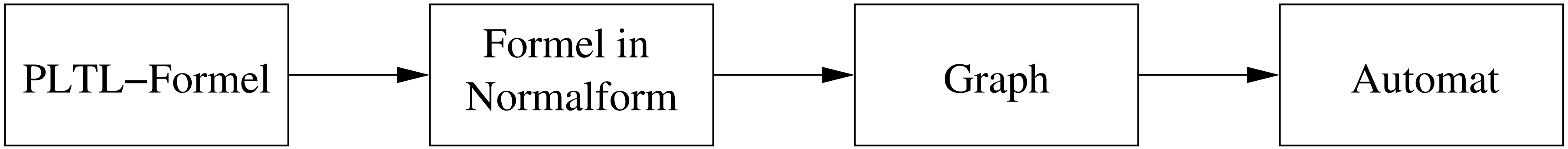
Dazu:

1. Erstellen eines Automaten A_{sys} für das System
2. Erstellen eines Automaten A_ϕ für ϕ
3. Verifizieren, daß A_{sys} der Spezifikation A_ϕ entspricht.

Konstruktion von A_{sys} i.d.R. möglich durch Werkzeugunterstützung direkt aus der Implementation o.ä. Entwurfsdokumenten.

Konstruktion von A_ϕ und Verifikation

Konstruktion von A_ϕ erfolgt in mehreren Schritten:



Die einzelnen Schritte werden im folgenden erläutert.

Verifikation von A_{sys} anhand von A_ϕ im Anschluß.

Übersicht

1. Einleitung
2. Normalform von PLTL-Formeln
3. Labelled Büchi Automata
4. LBAs und PLTL-Formeln
5. Automatenkonstruktion
6. Abschließende Bemerkungen

Normalform von PLTL-Formeln

Sei $p \in AP$, dann ist durch

$$\phi ::= p \mid \neg p \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathbf{X}\phi \mid \phi \mathbf{U} \phi \mid \phi \bar{\mathbf{U}} \phi$$

die Menge der gültigen PLTL-Formeln in Normalform definiert.

Das heißt im einzelnen:

1. $\mathbf{F}\phi$ wird ersetzt durch $\mathit{true}\mathbf{U}\phi$
2. $\mathbf{G}\phi$ wird ersetzt durch $\neg\mathbf{F}\neg\phi$
3. true wird ersetzt durch $p \vee \neg p$ (für irgendein $p \in AP$)
4. false wird ersetzt durch $\neg\mathit{true}$
5. Negation steht nur noch vor Aussagezeichen (*atomic propositions*)

Normalform von PLTL-Formeln

Um die letzte Bedingung zu erfüllen müssen Ausdrücke ggfs. umgeformt werden:

$$\begin{aligned}\neg(\phi \vee \psi) &= \neg\phi \wedge \neg\psi \\ \neg(\phi \wedge \psi) &= \neg\phi \vee \neg\psi \\ \neg\mathbf{X}\phi &= \mathbf{X}\neg\phi\end{aligned}$$

Zur Auflösung von $\neg(\phi\mathbf{U}\psi)$ wird ein Hilfsoperator $\bar{\mathbf{U}}$ eingeführt:

$$\begin{aligned}\neg(\phi\mathbf{U}\psi) &= \neg\phi\bar{\mathbf{U}}\neg\psi \\ \neg(\phi\bar{\mathbf{U}}\psi) &= \neg\phi\mathbf{U}\neg\psi\end{aligned}$$

Übersicht

Labelled Finite State Automata (LFSA)

PLTL-Model Checking basiert auf endlichen Automaten, die unendliche Wörter akzeptieren, den sog. Labelled Büchi Automata.

Ein Labelled Büchi Automaton (LBA) ist eine Variante eines Labelled Finite State Automaton (LFSA) mit einem speziellen Akzeptanzkriterium für Wörter (Büchi Kriterium).

Deshalb: Zunächst Erläuterung der Labelled Finite State Automata (LFSA).

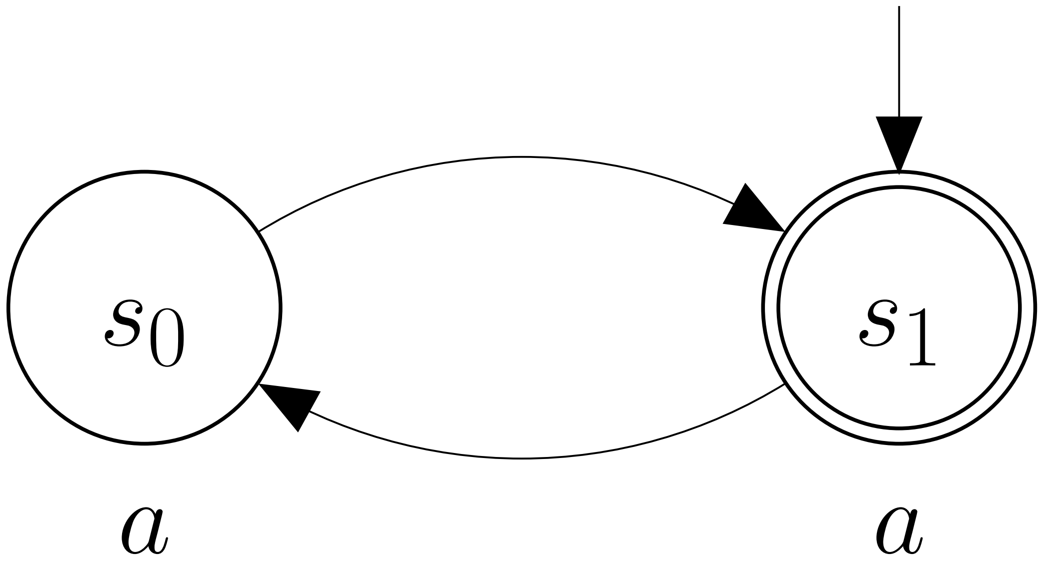
Definition eines LFSA

Ein LFSA ist definiert als $A = (\Sigma, S, S_0, \rho, F, l)$ mit:

- Σ ist das (nicht leere) Eingabealphabet
- S ist der endliche Zustandsraum
- $S_0 \subseteq S$ ist die (nicht leere) Menge von Startzuständen
- $\rho : S \to \mathcal{P}(S)$ ist die Übergangsfunktion
- $F \subseteq S$ ist die Menge der akzeptierenden Endzustände
- $l : S \to \Sigma$ ordnet jedem Zustand einen Bezeichner aus Σ zu (*labelling function*)

Graphische Darstellung eines LFSA

Ein LFSA $A = \big(\{a\}, \{s_0, s_1\}, \{s_1\}, \rho, \{s_1\}, l\big)$ wird wie folgt dargestellt:



Hier wird den Zuständen s_0 und s_1 der Bezeichner a zugeordnet, und s_1 ist einziger Start- und Endzustand.

Der Lauf eines LFSA

Bezeichne Σ^* die Menge der endlichen Wörter über Σ. Ferner schreiben wir $s \to s'$ gdw. $s' \in \rho(s)$.

Der Lauf (engl. *Run*) eines LFSA A, der endl. Wörter akzeptiert, ist eine endl. Folge $\sigma = s_0 \ldots s_n$ mit $s_0 \in S_0$ und $s_i \to s_{i+1}$ für $0 \leq i < n$. σ akzeptiert gdw. $s_n \in F$.

Die von einem LFSA akzeptierte Sprache

Wir sagen, ein LFSA A akzeptiert das endl. Wort $w = a_0 \ldots a_n$ gdw. ein Lauf $\sigma = s_0 \ldots s_n$ existiert mit $l(s_i) = a_i$ für $0 \leq i \leq n$.

Die von A akzeptierte Sprache $L(A)$ ist dann definiert durch:

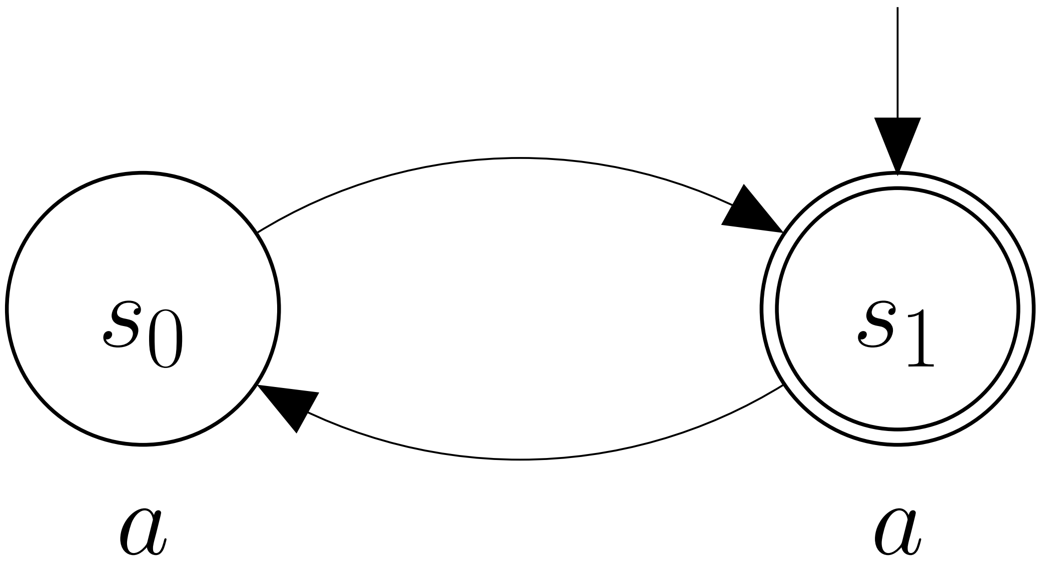
$$L(A) = \left\{ w \in \Sigma^* \mid A \text{ akzeptiert } w \right\}$$

$L(A)$ enthält also alle endl. Wörter, die A akzeptiert.

Beispiel eines LFSA für endl. Wörter

Sei $A = \big(\{a\}, \{s_0, s_1\}, \{s_1\}, \rho, \{s_1\}, l\big)$ mit

$$\begin{aligned} \rho:\ & s_0 \to \{s_1\} \\ & s_1 \to \{s_0\} \\ l:\ & s_0 \to a \\ & s_1 \to a \end{aligned}$$



A akzeptiert offensichtlich die Menge der endl. Wörter über $\{a\}$, deren Länge ein Vielfaches von 2 ist, also $L(A) = \big\{a^{2n} | n \in \mathbb{N}\big\}$.

Labelled Büchi Automata (LBA)

Problem: Für Model Checking allerdings Betrachtung von unendl. Wörtern ($\rightarrow$ Programme terminieren i.d.R. nicht)

$\Rightarrow$ kein Endzustand

$\Rightarrow$ Akzeptanzkriterium nicht anwendbar.

Neues Akzeptanzkriterium: (das sog. Büchi-Kriterium)

Der Lauf eines LBA A ist eine unendl. Folge $\sigma = s_0 s_1 \ldots$ mit $s_0 \in S_0$ und $s_i \rightarrow s_{i+1}$ für $i \geq 0$. Sei $lim(\sigma)$ die Menge der Zustände, die durch σ unendl. oft besucht werden. σ akzeptiert gdw. $lim(\sigma) \cap F \neq \emptyset$.

Die von einem LBA akzeptierte Sprache

Sei Σ^ω die Menge der unendl. Wörter über Σ. A akzeptiert $w = a_0 a_1 \cdots \in \Sigma^\omega$ gdw. ein akzeptierender Lauf $\sigma = s_0 s_1 \ldots$ für A existiert mit $l(s_i) = a_i$ für $i \geq 0$.

Die Menge der von A akzeptieren unendl. Wörter ist dann:

$$L_\omega(A) = \left\{ w \in \Sigma^\omega \mid A \text{ akzeptiert } w \right\}$$

$L_\omega(A)$ enthält also alle unendl. Wörter, die A akzeptiert.

Übersicht

LBAs und PLTL-Formeln

Ziel ist es Automaten für beliebige PLTL-Formeln angeben zu können.

Dazu: Verwendung von LBAs, da

- betrachtete Programme i.d.R. nicht terminieren ($\rightarrow$ unendl. Wörter)
- der Zustandsraum endlich sein muss, sonst $L_\omega(A_{sys}) \subseteq L_\omega(A_\phi)$ nicht entscheidbar.

Im Folgenden wird kurz erläutert, wie PLTL-Formeln durch LBAs dargestellt werden können.

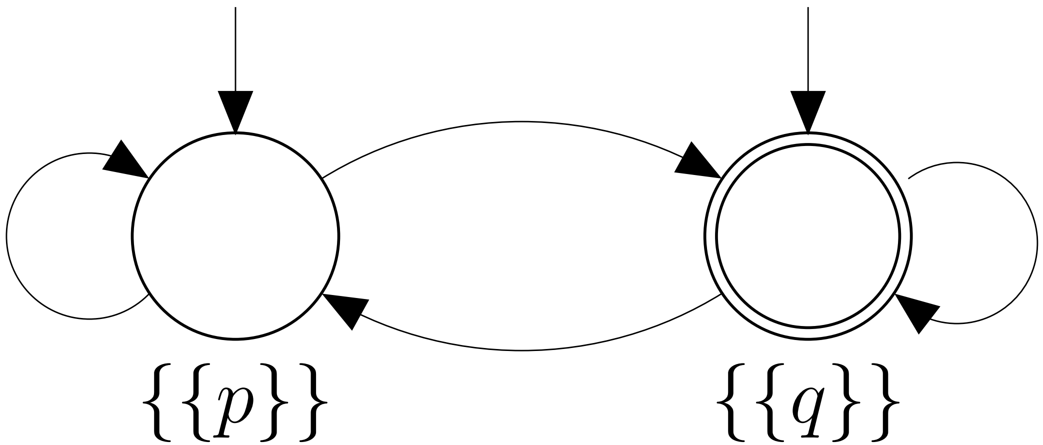
Adaption von LBAs für PLTL-Formeln

Sei $\Sigma = \mathcal{P}(AP)$ und $l : S \to \mathcal{P}(\Sigma)$, dann wird $w = a_0 a_1 \ldots$ akzeptiert gdw. ein akzeptierender Lauf $\sigma = s_0 s_1 \ldots$ existiert mit $s_0 \in S_0$ und $a_i \in l(s_i)$ für $i \geq 0$.

Wörter bestehen also aus Mengen von Aussagezeichen (*atomic propositions*). Der Grund wird anhand der nachfolgenden Beispiele klar.

A_ϕ akzeptiert also alle Folgen von Mengen von Aussagezeichen, für die ϕ gilt.

Beispiel 1: $\phi = \mathbf{G}(p\mathbf{U}q)$ mit $p, q \in AP$



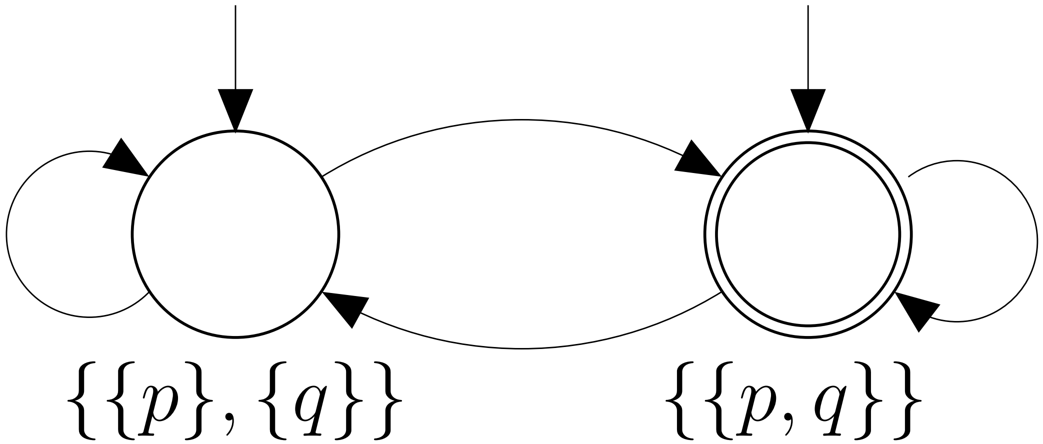
Die von A_ϕ akzeptierte unendl. Sprache ist

$$L_\omega(A_\phi) = \Big(\{p\}^*\{q\}\Big)^\omega$$

Vereinfacht ausgedrückt: q muß unendl. oft gelten, und zwischendurch darf jeweils p beliebig oft gelten, dann akzeptiert A_ϕ.

Beispiel 2: $\psi = \mathbf{G}\Big((p \vee q)\mathbf{U}(p \wedge q)\Big)$

Beispiel 2: $\psi = \mathbf{G}\Big((p \vee q)\mathbf{U}(p \wedge q)\Big)$ mit $p, q \in AP$



Die von A_ψ akzeptierte unendl. Sprache ist

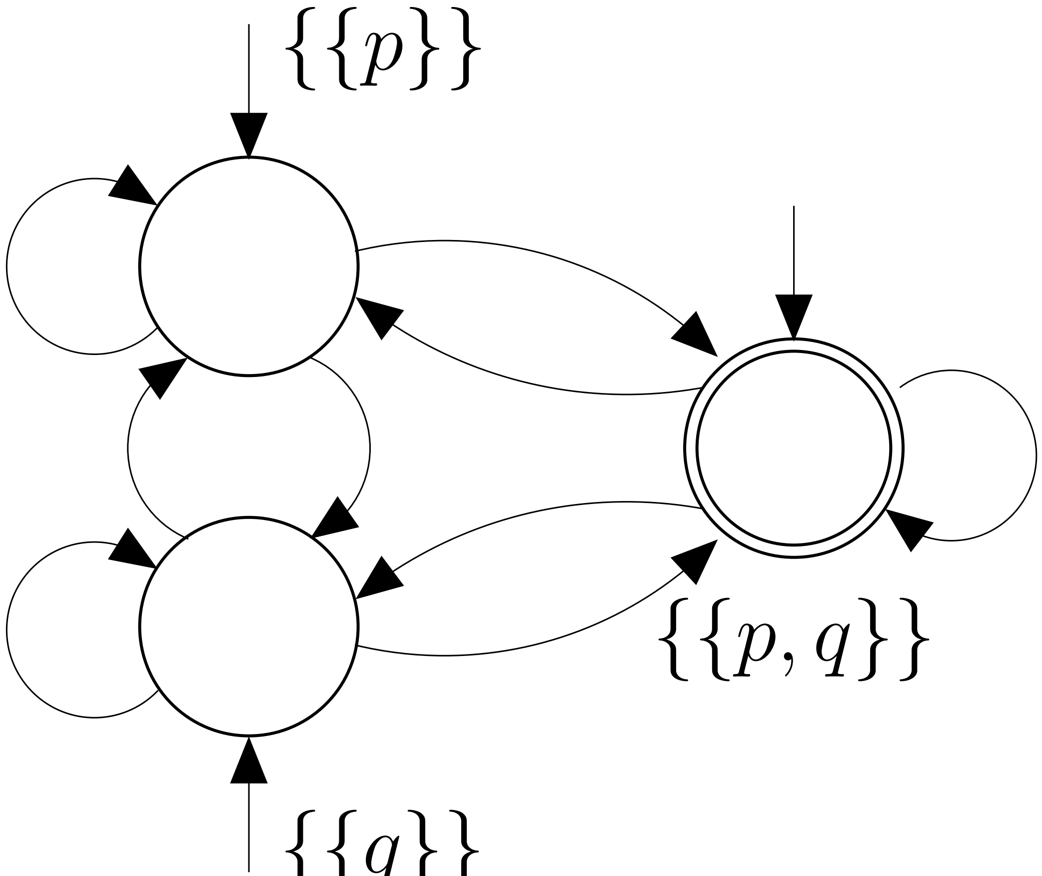
$$L_\omega(A_\psi) = \Big((\{p\}|\{q\})^*\{p,q\}\Big)^\omega$$

Anhand der Konjunktion $p \wedge q$ wird klar, warum $l : S \to \mathcal{P}(\mathcal{P}(AP))$.

Alternative Darstellung für Disjunktion möglich?

Beispiel 2: Äquivalenter Automat A'_ψ

$\psi = \mathbf{G}\Big((p \vee q)\mathbf{U}(p \wedge q)\Big)$ kann auch durch den folgenden Automaten A'_ψ dargestellt werden.



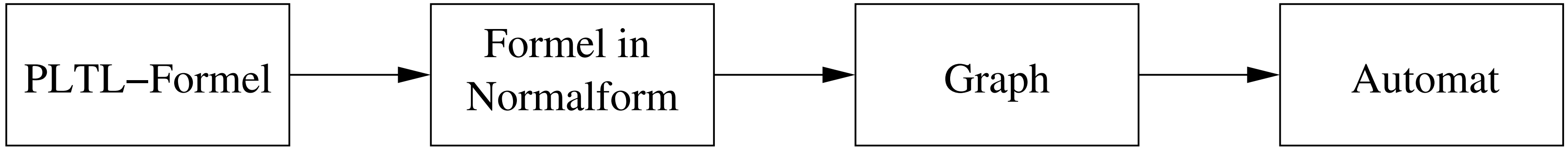
A'_ψ akzeptiert ebenfalls:
$L_\omega(A'_\psi) = \Big((\{p\}|\{q\})^*\{p,q\}\Big)^\omega$

Übersicht

1. Einleitung
2. Normalform von PLTL-Formeln
3. Labelled Büchi Automata
4. LBAs und PLTL-Formeln
5. Automatenkonstruktion
6. Abschließende Bemerkungen

Automatenkonstruktion

Zur Erinnerung: Der vorgeschlagene Algorithmus zur Konstruktion eines Automaten A_ϕ für eine PLTL-Formel ϕ:



Als nächstes: Betrachtung der Graphkonstruktion für ϕ.

Definition eines Graphen

Ein Graph $G_\phi = (V, E)$ für ϕ besteht aus einer Menge von Knoten V und einer Menge von Kanten $E \subseteq V \times V$. Ein Knoten v ist ein Quadrupel (P, N, O, Sc) mit

- einer Menge von Vorgängern $P \subseteq (V \cup \{init\})$ und
- Mengen von PLTL-Formeln N, O und Sc.

$N(v)$ ist die Menge der noch zu prüfenden Formeln. $O(v)$ ist die Menge der bereits geprüften Formeln. $Sc(v)$ enthält Formeln, die für alle direkten Nachfolger gelten müssen.

Algorithmus zur Graphkonstruktion

Die Funktion $CreateGraph(\phi)$ liefert eine Menge von Knoten - und implizite Kanteninformationen - die den Graph für ϕ repräsentiert.

V ist die Menge von Knoten des Graphen und S ein Stack mit den noch zu bearbeitenden Knoten. Anfangs ist $V = \emptyset$ und S enthält einen Knoten $(\{init\}, \{\phi\}, \emptyset, \emptyset)$.

Iteratives Abarbeiten bis S leer. Anschließend enthält V alle Knoten des Graphen.

until $is\_empty(S)$ **do**
  $v := top(S)$
  (v bearbeiten *)*
  $\vdots$
od

Algorithmus zur Graphkonstruktion

Falls v vollständig bearbeitet, $N(v) = \emptyset$, dann v aus S entfernen, und

- falls schon ein Knoten $u \in V$ existiert, der bzgl. O und Sc mit v übereinstimmt, werden nur neue Vorgänger ($\rightarrow$ Kanten) zu u hinzugefügt, $P(u) := P(u) \cup P(v)$,
- sonst v zu V hinzufügen, und $(\{v\}, Sc(v), \emptyset, \emptyset)$ auf S legen, da ggfs. noch nicht alle Nachfolger von v erforscht.

Algorithmus zur Graphkonstruktion

Ansonsten, wähle ein $\psi \in N(v)$ und entferne ψ aus $N(v)$:

- $\psi \in AP \vee (\neg\psi) \in AP$, dann falls ψ schon negiert in $O(v)$ vorkommt v aus S entfernen ($\rightarrow$ Widerspruch), sonst ψ ignorieren
- $\psi = (\psi_1 \wedge \psi_2)$, dann $N(v) := N(v) \cup \{\psi_1, \psi_2\}$ sofern nicht in $O(v)$
- $\psi = \mathbf{X}\psi_1$, dann $Sc(v) := Sc(v) \cup \{\psi_1\}$
- ansonsten, also ψ entweder $\psi_1 \vee \psi_2$, $\psi_1 \mathbf{U} \psi_2$ oder $\psi_1 \bar{\mathbf{U}} \psi_2$, v aufspalten in v_1 und v_2 mit initial $v_1 = v, v_2 = v$:
  $\rightarrow$ v aus S entfernen, dafür v_2 und v_1 auf S legen
  $\rightarrow$ ψ als erledigt für v_1 und v_2 vermerken
  $\rightarrow$ anschließend N für v_1, v_2 bestimmen

Fallunterscheidung $\psi \in \{\psi_1 \vee \psi_2, \psi_1 \mathbf{U} \psi_2, \psi_1 \bar{\mathbf{U}} \psi_2\}$

Für $\psi = \psi_1 \vee \psi_2$ werden einfach ψ_1 und ψ_2 als *noch zu prüfen* auf v_1 und v_2 verteilt, also $N(v_1) := N(v_1) \cup \Big(\{\psi_1\} \setminus O(v_1)\Big)$ und $N(v_2) := N(v_2) \cup \Big(\{\psi_2\} \setminus O(v_2)\Big)$.

Die übrigen Fälle $\psi = \psi_1 \mathbf{U} \psi_2$ und $\psi = \psi_1 \bar{\mathbf{U}} \psi_2$ lassen sich auf Disjunktion zurückführen:

- $\psi_1 \mathbf{U} \psi_2 \Leftrightarrow \psi_2 \vee \Big[\psi_1 \wedge \mathbf{X}(\psi_1 \mathbf{U} \psi_2)\Big]$
- $\psi_1 \bar{\mathbf{U}} \psi_2 \Leftrightarrow \psi_2 \wedge \Big[\psi_1 \vee \mathbf{X}(\psi_1 \bar{\mathbf{U}} \psi_2)\Big]$
  $\Leftrightarrow (\psi_1 \wedge \psi_2) \vee \Big[\psi_2 \wedge \mathbf{X}(\psi_1 \bar{\mathbf{U}} \psi_2)\Big]$

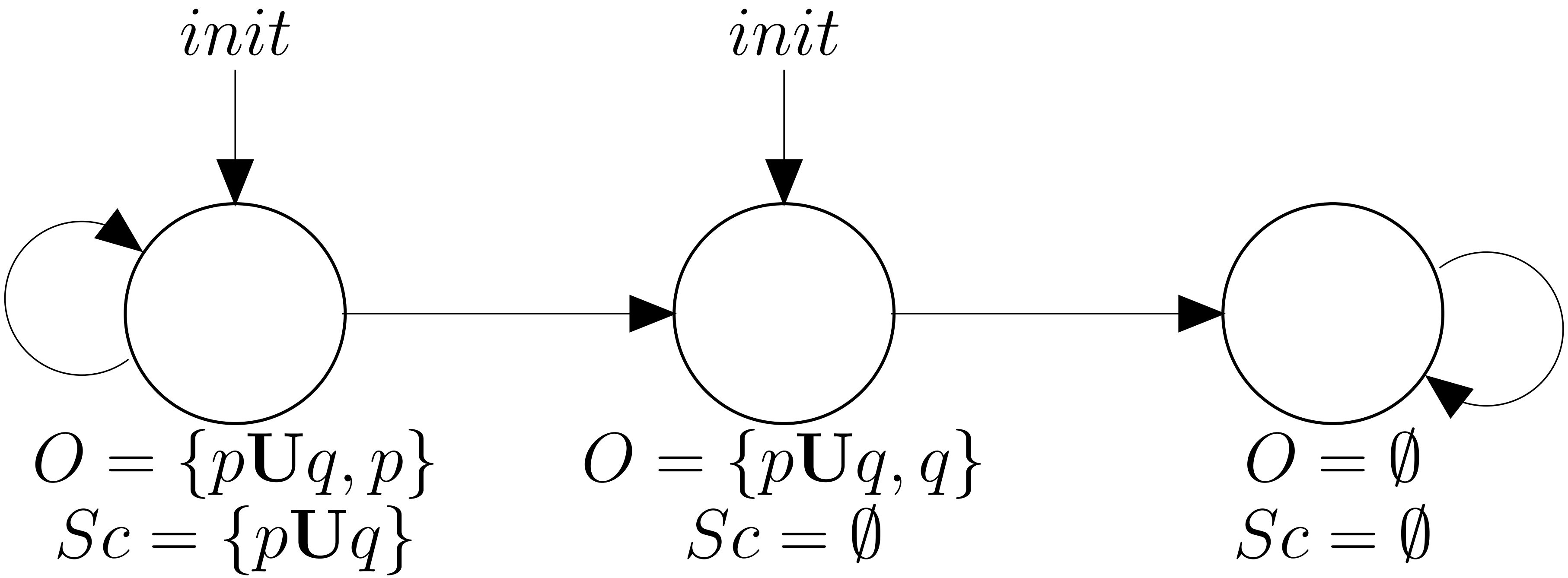
Algorithmus zur Graphkonstruktion

Abschließend, falls $N(v) \neq \emptyset$ war, noch ψ als bearbeitet in v markieren, also:

$$O(v) := O(v) \cup \{\psi\}$$

Falls S immer noch Knoten enthält, einen weiteren Durchlauf starten. Ansonsten $CreateGraph$ beenden und V zurückgeben.

Beispiel: Graph G_ϕ für $\phi = p\mathrm{U}q$ mit $p, q \in AP$.



Definition eines GLBA

Der so konstruierte Graph soll nun in einen LBA umgewandelt werden.

Zunächst: Konstruktion eines Generalized LBA (GLBA).

Ein GLBA $A = (\Sigma, S, S_0, \rho, \mathcal{F}, l)$ entspricht einem LBA, außer daß $\mathcal{F}$ eine Menge von *Endzustandsmengen* $\{F_1, \ldots, F_n\}$ mit $n \in \mathbb{N}$ und $F_i \subseteq S$ für $0 < i \leq n$, also $\mathcal{F} \subseteq \mathcal{P}(S)$.

Ein Lauf $\sigma = s_0 s_1 \ldots$ ist akzeptierend wenn aus jeder Menge $F_i \in \mathcal{F}$ zumindest ein Zustand unendl. oft durch σ besucht wird, also:

$$\forall 0 < i \leq n . lim(\sigma) \cap F_i \neq \emptyset$$

Konstruktion eines GLBA

Die Konstruktion eines GLBA A_ϕ für ϕ gestaltet sich nun einfach:

- $\Sigma = \mathcal{P}(AP)$
- $S = CreateGraph(\phi)$
- $S_0 = \{s \in S \mid init \in P(s)\}$
- $s \to s' \Leftrightarrow s \in P(s') \wedge s \neq init$
- $\mathcal{F} = \Big\{\{s \in S \mid \phi_1 \mathbf{U} \phi_2 \notin O(s) \vee \phi_2 \in O(s)\} \mid \phi_1 \mathbf{U} \phi_2 \in Sub(\phi)\Big\}$
- $l(s) = \{AP' \subseteq AP \mid Pos(s) \subseteq AP' \wedge Neg(s) \cap AP' = \emptyset\}$

Hierbei sei

- $Sub(\phi)$ die Menge der Teilformeln von ϕ,
- $Pos(s) = O(s) \cap AP$ die in s gültigen Ausdruckszeichen und entsprechend $Neg(s) = \{p \in AP \mid (\neg p) \in O(s)\}$.

Konstruktion eines GLBA

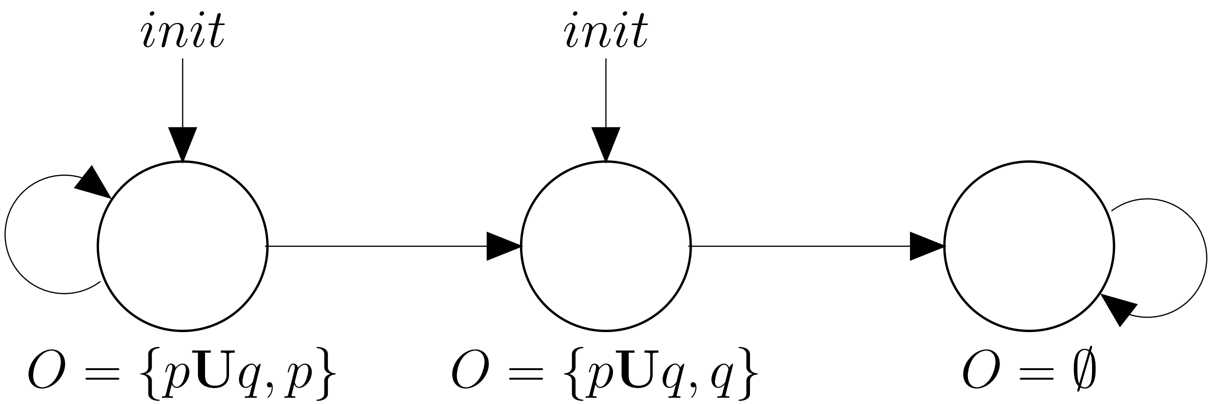
$\mathcal{F}$ enthält also eine Menge $F_i = \{s \in S \mid \phi_1 \mathbf{U} \phi_2 \notin O(s) \vee \phi_2 \in O(s)\}$ für jede Teilformel $\phi_1 \mathbf{U} \phi_2$ in ϕ. F_i enthält alle s, für die entweder $\phi_1 \mathbf{U} \phi_2$ nicht in $O(s)$ enthalten ist, oder ϕ_2 Teil von $O(s)$ ist.

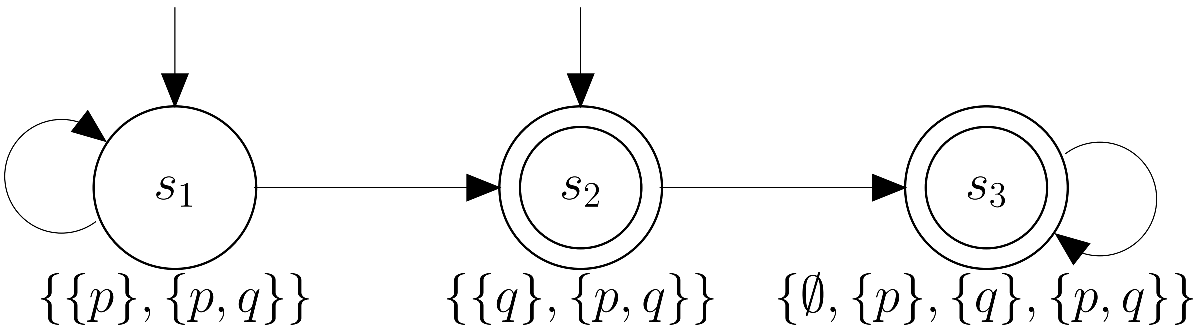
Falls keine Teilformel $\phi_1 \mathbf{U} \phi_2$ existiert, ist $\mathcal{F} = \emptyset$, also sind alle Läufe akzeptierend (z.B. $p \wedge \mathbf{X} q$).

Als Bezeichner für s wird die Menge der Mengen von Ausdruckszeichen verwendet, von denen $Pos(s)$ eine Teilmenge ist, und die nicht in $Neg(s)$ vorkommen.

Für $O(s) = \{p \mathbf{U} q, p\}$ ist $Pos(s) = \{p\}$ und $Neg(s) = \emptyset$, also $l(s) = \{\{p\}, \{p, q\}\}$.

Beispiel: Graph G_ϕ und GLBA A_ϕ für $\phi = p\mathbf{U}q$ mit $p, q \in AP$.





Konstruktion eines LBA aus einem GLBA

Um einen LBA A' aus einem GLBA A zu konstruieren, erstellt man im Prinzip n Kopien von A, eine für jede *Endzustandsmenge* F_i. Aus jedem Zustand s wird ein Paar (s, i) mit $0 < i \leq n$.

Auf die weiteren Details der Transformation wird hier aus Zeitgründen nicht eingegangen.

Übersicht

Konkretisierung des Verfahrens

1. Erstellen eines Automaten A_{sys} für das Modell des Systems
2. Erstellen eines Automaten A_{ϕ} für die Spezifikation
3. Verifizieren, dass die Implementation der Spezifikation entspricht, formal: $L_{\omega}(A_{sys}) \subseteq L_{\omega}(a_{\phi})$

Verfahren für Schritt 2 wurde erläutert, allerdings nur durch Werkzeugunterstützung zu lösen, da selbst kurze Formeln große Anzahl von Zuständen erzeugen. Z.B. enthält der Graph für $\neg(\mathbf{FF}p \Leftrightarrow \mathbf{F}p)$ bereits 22 Knoten und 41 Kanten.

Abschließende Bemerkungen

Die Entscheidung für Schritt 3 ist NP-Vollständig, kann aber umformuliert werden:

$$L_\omega(A_{sys}) \subseteq L_\omega(A_\phi) \;\Leftrightarrow\; L_\omega(A_{sys}) \cap L_\omega(\bar{A}_\phi) = \emptyset$$

Konstruktion von $\bar{A}_\phi$ allerdings aufwendig; hat A_ϕ n Zustände, so hat $\bar{A}_\phi$ i.a. c^{n^2} Zustände für irgendein $c > 1$. Deshalb statt $\bar{A}_\phi$ besser $A_{\neg\phi}$.

Also Schritt 3 damit reduziert auf $L_\omega(A_{sys}) \cap L_\omega(A_{\neg\phi}) = \emptyset$.

Wird im Anschluß erläutert.