Model Checking mit Büchi Automaten

Ingo Weigelt

Softwaretechnik 3

16.05.2007

Model Checking mit Büchi Automaten
Übersicht

ω-Automaten

ω-Automaten sind endliche Automaten, die unendlich lange Wörter akzeptieren.

- Unendliche Wörter repräsentieren nicht-terminierende Abläufe eines Systems.
- Endzustände werden durch andere Akzeptanzbedingungen ersezt.
- Beispiel: Büchi-Automaten.

Büchi-Automaten:

Formelle Definition:

Ein *Büchi Automat* ist ein 5-Tupel $(\Sigma, Q, \Delta, Q^0, F)$ mit:

- Σ : endliches Alphabet
- Q : endliche Menge von Zuständen
- $\Delta \subseteq Q \times \Sigma \times Q$: Transitionsrelation
- $Q^0 \subseteq$: Startzustände
- $F \subseteq Q$: Akteptierende Zustände

Akzeptanzverhalten:

- Sei $inf(\rho)$ die Menge der Zustände, die beim Lauf ρ unendlich oft besucht werden.
- Dann ist ρ akzeptierend gdw. $inf(\rho) \cap F \neq \emptyset$.

verallgemeinerte Büchi-Automaten

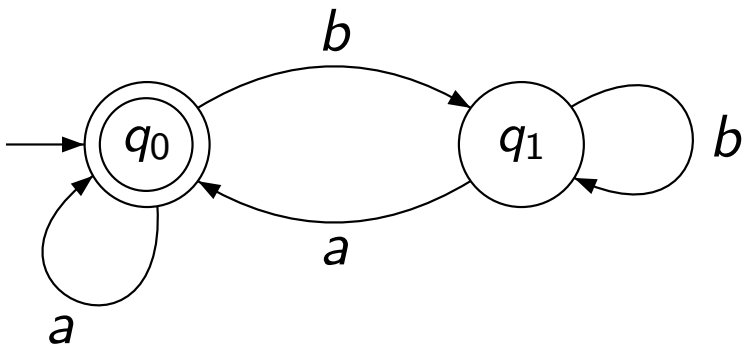
verallgemeinerter Büchi Automat

- Ein *verallgemeinerter Büchi Automat* ist ein Büchi-Automat bei dem F aus Mengen von Mengen von Zuständen besteht.
- Ein Lauf ρ ist akzeptierend gdw. $inf(\rho) \cap P_i \neq \emptyset$ für alle $P_i \in F$.

Zu jedem verallgemeinerten Büchi-Automaten $\mathcal{B}$ existiert ein Büchi-Automat $\mathcal{B}'$ mit $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}')$.

Beispiel

Büchi Automat $\mathcal{A}$: mit $\mathcal{L}(\mathcal{A}) = (b^* a)^\omega$



Bemerkungen

- Die von Büchi-Automaten erkannte Sprache heisst ω-*reguläre Sprache*.
- Nicht jede ω-reguläre Sprache wird auch von einem deterministischen Büchi-Automaten erkannt.
  z.B. $\mathcal{L} = \{\alpha \in \{a, b, c\} \mid \alpha \text{ enthält nur endlich viele } b's\}$
- D.h. nicht jeder nicht-deterministische Büchi-Automat kann in einen deterministischen Büchi-Automaten umgewandelt werden.

Abschlusseigenschaften

- deterministische Büchi-Automaten sind abgeschlossen unter Vereinigung und Schnitt.
- nichtdeterministische Büchi-Automaten sind zusätzlich abgeschlossen unter Komplement.

Leerheitsproblem

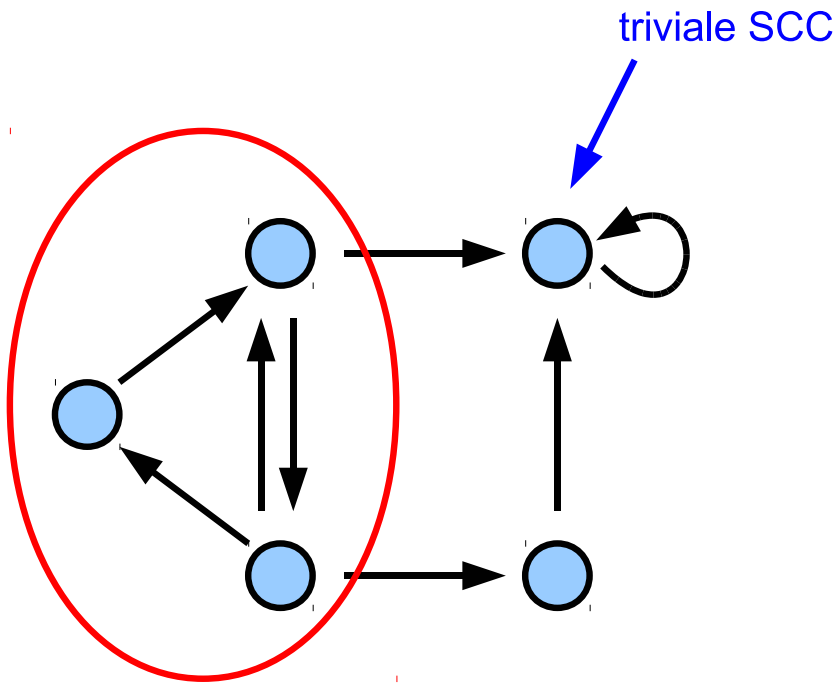
Entscheidbarkeit des Leerheitsproblems:

Es ist entscheidbar, ob ein Büchi-Automat die leere Sprache akzeptiert.

- Bestimme die SCCs des Automaten.
- Die akzeptierte Sprache ist nicht Leer genau dann, wenn es eine SCC gibt, die von einem Startzustand erreichbar ist und einen akzeptierenden Zustand enthält.

SCC - Strongly Connected Components

- Für die Labelfunktion $EG\psi$ müssen zuerst alle SCCs bestimmt werden
- Wiederholung: Eine starke Zusammenhangskomponente ist ein Teilgraph in dem jeder Knoten von jedem anderen Knoten über einen Pfad innerhalb des Graphen erreichbar ist.
- Beispiel:



Model Checking mit Büchi Automaten

Model Checking Problem:

- Gegeben: Kripke Struktur $\mathcal{M}$ und Spezifikation S
- Gesucht: Gilt $\mathcal{M} \models S$? Wenn nicht ist ein Gegenbeispiel gesucht

Idee:

- konstruiere Automat $\mathcal{A}$ zu $\mathcal{M}$
- konstruiere Automat $\mathcal{S}$ der $\mathcal{L}(S)$ erkennt
- Das Model $\mathcal{A}$ erfüllt die Spezifikation S gdw.

$$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$$

oder

$$\mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(S)} = \emptyset$$

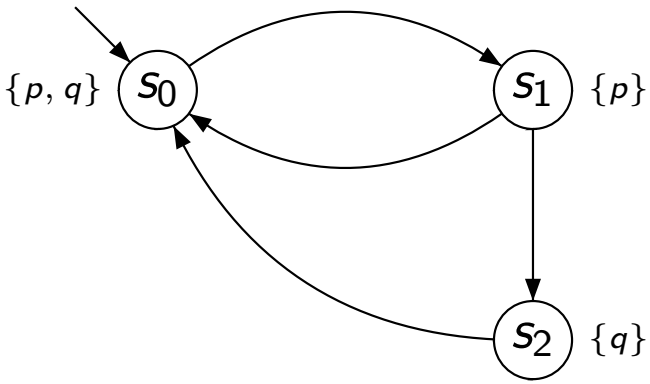
Model Checking mit Büchi Automaten

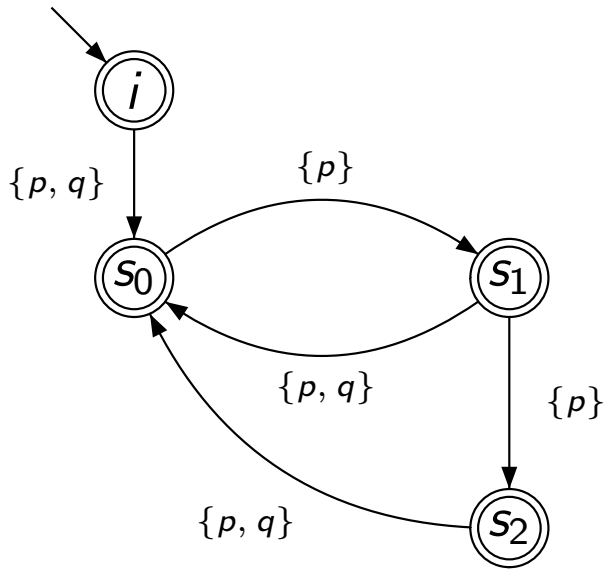
Model Checking Algorithmus:

- konstruiere Automat $\mathcal{A}$ zu $\mathcal{M}$
- konstruiere Automat $\mathcal{S}$ der $\overline{\mathcal{L}(S)}$ erkennt
- Bilde den Durchschnitt von $\mathcal{A}$ und $\mathcal{S}$
- Prüfe, ob der Automat $\mathcal{A} \cap \mathcal{S}$ leer ist.
  Falls $\mathcal{A} \cap \mathcal{S} \neq \emptyset$: Gegenbeispiel ausgeben

Konstruktion von $\mathcal{A}$

- Zustände der Kripke Struktur sind Zustände des Automaten
- Label der Zustände werden zu Label der Transitionen
- daher neuer Startzustand i
- alle Zustände sind akzeptierend





Konstruktion von $\mathcal{S}$

Model Checking Algorithmus:

- konstruiere Automat $\mathcal{A}$ zu $\mathcal{M}$
- konstruiere Automat $\mathcal{S}$ der $\overline{\mathcal{L}(S)}$ erkennt
- Bilde den Durchschnitt von $\mathcal{A}$ und $\mathcal{S}$
- Prüfe, ob der Automat $\mathcal{A} \cap \mathcal{S}$ leer ist.
  Falls $\mathcal{A} \cap \mathcal{S} \neq \emptyset$: Gegenbeispiel ausgeben

Konstruktion von $\mathcal{S}$

Gesucht ist ein Büchi Automat $\mathcal{S}$ der $\overline{\mathcal{L}(S)}$ erkennt.

- $\mathcal{S}$ kann direkt angegeben werden (SPIN)
- Es gibt ω-Automaten, bei denen das Komplement leicht berechnet werden kann.
- $\mathcal{S}$ wird aus einer Sprache wie LTL abgeleitet:
  $S = \varphi \Rightarrow$ bilde $\mathcal{S}$ zu $\neg\varphi$

Algorithmus von Gerth, Peled, Vardi und Wolper

Algorithmus:

Eingabe: Eigentschaft φ als LTL-Formel in negierter Normalform
D.h. nur Operatoren $U, R, \wedge und \vee$ erlaubt

- $F\psi \Rightarrow true\, U\, \psi$
- $G\psi \Rightarrow false\, R\, \psi$
- usw.

und Negation nur vor Propositionen:

- $\neg(\mu U \eta) = (\neg\mu) R (\neg\eta)$
- $\neg(\mu R \eta) = (\neg\mu) U (\neg\eta)$
- $\neg X \mu = X \neg\mu$

Ausgabe: verallgemeinerter Büchi Automat $\mathcal{B}$ zu φ

Algorithmus von Gerth, Peled, Vardi und Wolper

Datenstruktur des Algorithmus:

$node = [ID, incoming, old, new, next]$

- *ID*: Id des Knotens
- *incoming*: Liste von IDs von Knoten mit Kante zu diesem Knoten
- *old*, *new*, *next*: Listen von Formeln (Teilformeln von φ)

  die Teilformeln in *old* wurden bereits berechnet
  die Teilformeln in *new* werden im aktuellen Schritt berechnet
  die Teilformeln in *next* müssen danach noch berechnet werden

Algorithmus von Gerth, Peled, Vardi und Wolper

Initialisierung:

$node_0 := [ID \Leftarrow new\_id(),$
$incoming \Leftarrow \{init\},$
$old \Leftarrow \emptyset,$
$new \Leftarrow \{\varphi\},$
$next \Leftarrow \emptyset];$

$Nodes := \text{expand}(node_0, \emptyset);$

Algorithmus von Gerth, Peled, Vardi und Wolper

Die Funktion *expand*() ersetzt den Knoten q mit $\varphi \in new(q)$ rekursiv durch neue Knoten q', so dass $new(q')$ Teilformeln von φ enthält:

$\varphi = \mu \vee \psi$

q wird ersetzt durch zwei neue Knoten q_1, q_2:
$new(q) := new(q) \setminus \{\varphi\}$

$$q_1 := [ID \Leftarrow new\_id(),\ incoming \Leftarrow incoming(q),\ old \Leftarrow old(q) \cup \{\varphi\},\ new \Leftarrow new(q) \cup \{\mu\},\ next \Leftarrow next(q)]$$

$$q_2 := [ID \Leftarrow new\_id(),\ incoming \Leftarrow incoming(q),\ old \Leftarrow old(q) \cup \{\varphi\},\ new \Leftarrow new(q) \cup \{\psi\},\ next \Leftarrow next(q)]$$

Algorithmus von Gerth, Peled, Vardi und Wolper

$\varphi = \mu \wedge \psi$

q wird ersetzt durch einen neuen Knoten q':
$new(q) := new(q) \setminus \{\varphi\}$

$q' := [ID \Leftarrow new\_id(),$
$incoming \Leftarrow incoming(q),$
$old \Leftarrow old(q) \cup \{\varphi\},$
$new \Leftarrow new(q) \cup \{\mu, \psi\},$
$next \Leftarrow next(q)]$

Algorithmus von Gerth, Peled, Vardi und Wolper

$\varphi = \mu U \psi$

$\varphi \Leftrightarrow (\mu \wedge X(\mu U \psi)) \vee \psi$

q wird ersetzt durch zwei neue Knoten q_1, q_2:

$new(q) := new(q) \setminus \{\varphi\}$

$q_1 := [ID \Leftarrow new\_id(),$
$incoming \Leftarrow incoming(q),$
$old \Leftarrow old(q) \cup \{\varphi\},$
$new \Leftarrow new(q) \cup \{\mu\},$
$next \Leftarrow next(q) \cup \{\mu U \psi\}]$

$q_2 := [ID \Leftarrow new\_id(),$
$incoming \Leftarrow incoming(q),$
$old \Leftarrow old(q) \cup \{\varphi\},$
$new \Leftarrow new(q) \cup \{\psi\},$
$next \Leftarrow next(q)]$

Algorithmus von Gerth, Peled, Vardi und Wolper

$\varphi = \mu R \psi$

$\varphi \Leftrightarrow \psi \wedge (\mu \vee X(\mu R \psi)) \quad \Leftrightarrow \quad (\psi \wedge \mu) \vee (\psi \wedge X(\mu R \psi))$

q wird ersetzt durch zwei neue Knoten q_1, q_2:

$new(q) := new(q) \setminus \{\varphi\}$

$q_1 := [ID \Leftarrow new\_id(),$
$incoming \Leftarrow incoming(q),$
$old \Leftarrow old(q) \cup \{\varphi\},$
$new \Leftarrow new(q) \cup \{\psi, \mu\},$
$next \Leftarrow next(q)]$

$q_2 := [ID \Leftarrow new\_id(),$
$incoming \Leftarrow incoming(q),$
$old \Leftarrow old(q) \cup \{\varphi\},$
$new \Leftarrow new(q) \cup \{\psi\},$
$next \Leftarrow next(q) \cup \{\mu R \psi\}]$

Algorithmus von Gerth, Peled, Vardi und Wolper

$\varphi = X\mu$

q wird ersetzt durch einen neuen Knoten q':
$new(q) := new(q) \setminus \{\varphi\}$

$q' := [ID \Leftarrow new\_id(),$
$incoming \Leftarrow incoming(q),$
$old \Leftarrow old(q) \cup \{\varphi\},$
$new \Leftarrow new(q),$
$next \Leftarrow next(q) \cup \{\mu\}]$

Algorithmus von Gerth, Peled, Vardi und Wolper

Falls φ (negierte) Proposition oder *true* ist:

q wird ersetzt durch einen neuen Knoten q':

$new(q) := new(q) \setminus \{\varphi\}$

$$
\begin{aligned}
q' := [&ID \Leftarrow new\_id(),\\
&incoming \Leftarrow \{ID()\},\\
&old \Leftarrow old(q) \cup \{\varphi\},\\
&new \Leftarrow new(q),\\
&next \Leftarrow next(q)]
\end{aligned}
$$

Algorithmus von Gerth, Peled, Vardi und Wolper

Algorithmus:

expand (q, Nodes) {
  if $\mathit{new}(q) \neq \emptyset$
    sei $\eta \in \mathit{new}(q)$;
    if $\eta \in \mathit{old}(q)$ /* η wurde schon berechnet */
      $\mathit{new}(q) := \mathit{new}(q) \setminus \{\eta\}$;
      expand(q, Nodes);
    if $\neg\eta \in \mathit{old}(q) \vee \eta = \mathit{False}$ /* Widerspruch */
      return *Nodes*;
    else
      erzeuge Knoten q' bzw. q_1 und q_2 wie beschrieben;
      expand(q',*Nodes*); bzw. expand(q_2,expand(q_1,*Nodes*));

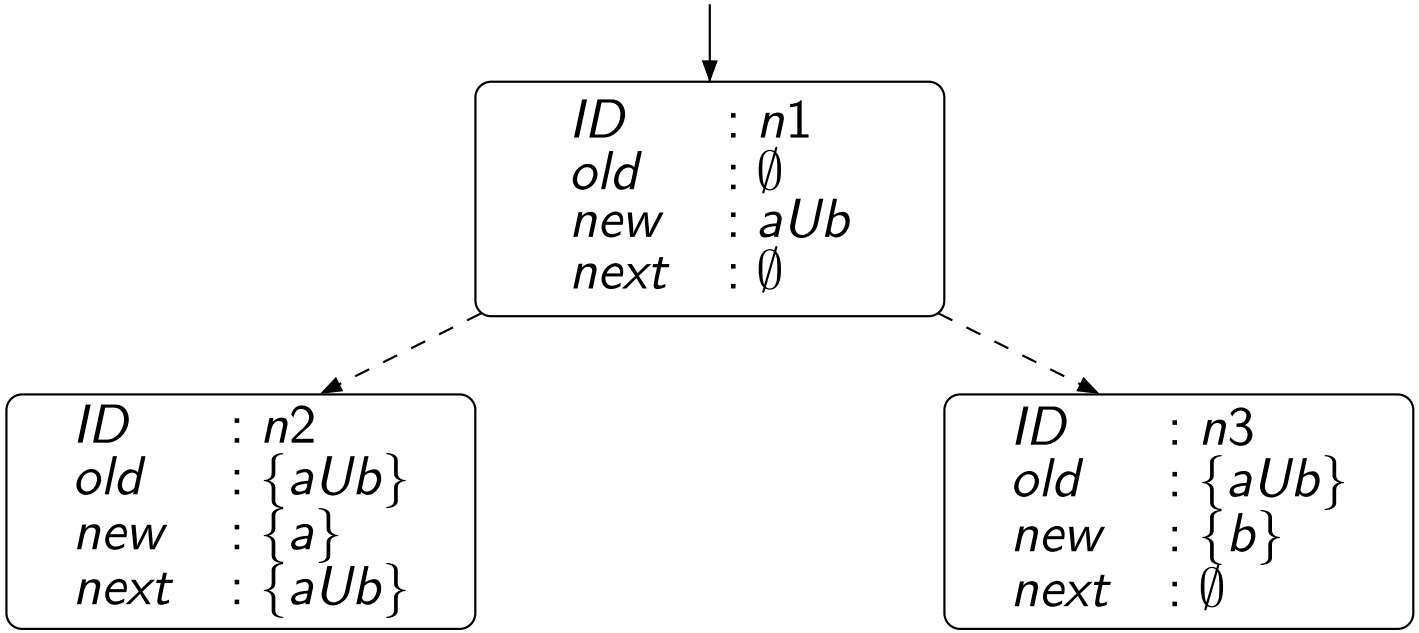
Algorithmus von Gerth, Peled, Vardi und Wolper

Algorithmus:

else /* $new(q) = \emptyset$ */
  if $\exists\ r \in Nodes$: $old(r) = old(q) \wedge next(r) = next(q)$ {
    $incoming(r) := incoming(r) \cup incoming(q)$;
    return $Nodes$;
  else /* neuen Knoten einfügen */
    expand([$ID \Leftarrow new\_id()$,
            $incoming \Leftarrow \{ID(q)\}$,
            $old \Leftarrow \emptyset$,
            $new \Leftarrow next(q)$,
            $next \Leftarrow \emptyset$], $Nodes \cup \{q\}$);
}

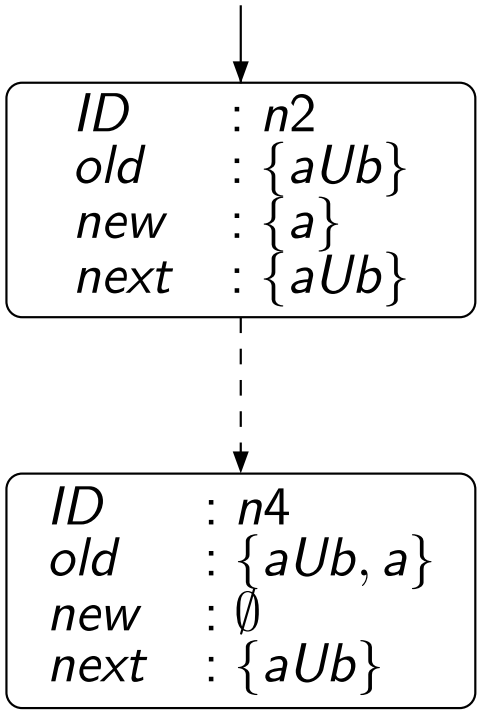
Algorithmus von Gerth, Peled, Vardi und Wolper

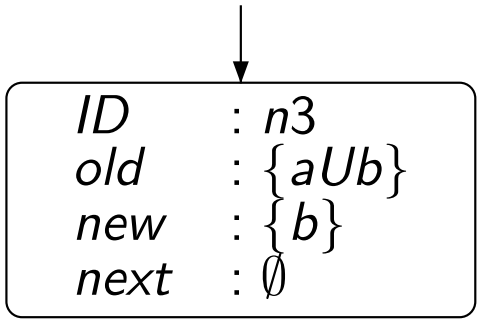
Beispiel: $\varphi = aUb$



Algorithmus von Gerth, Peled, Vardi und Wolper

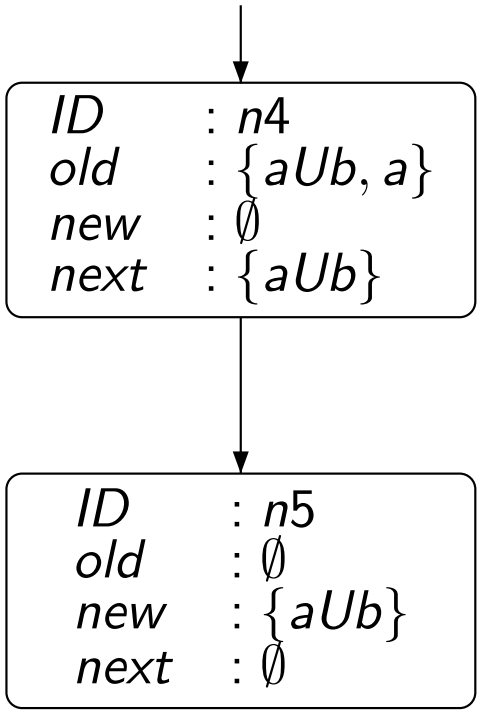
Beispiel: $\varphi = aUb$

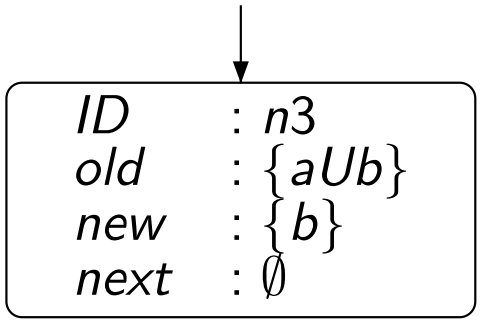




Algorithmus von Gerth, Peled, Vardi und Wolper

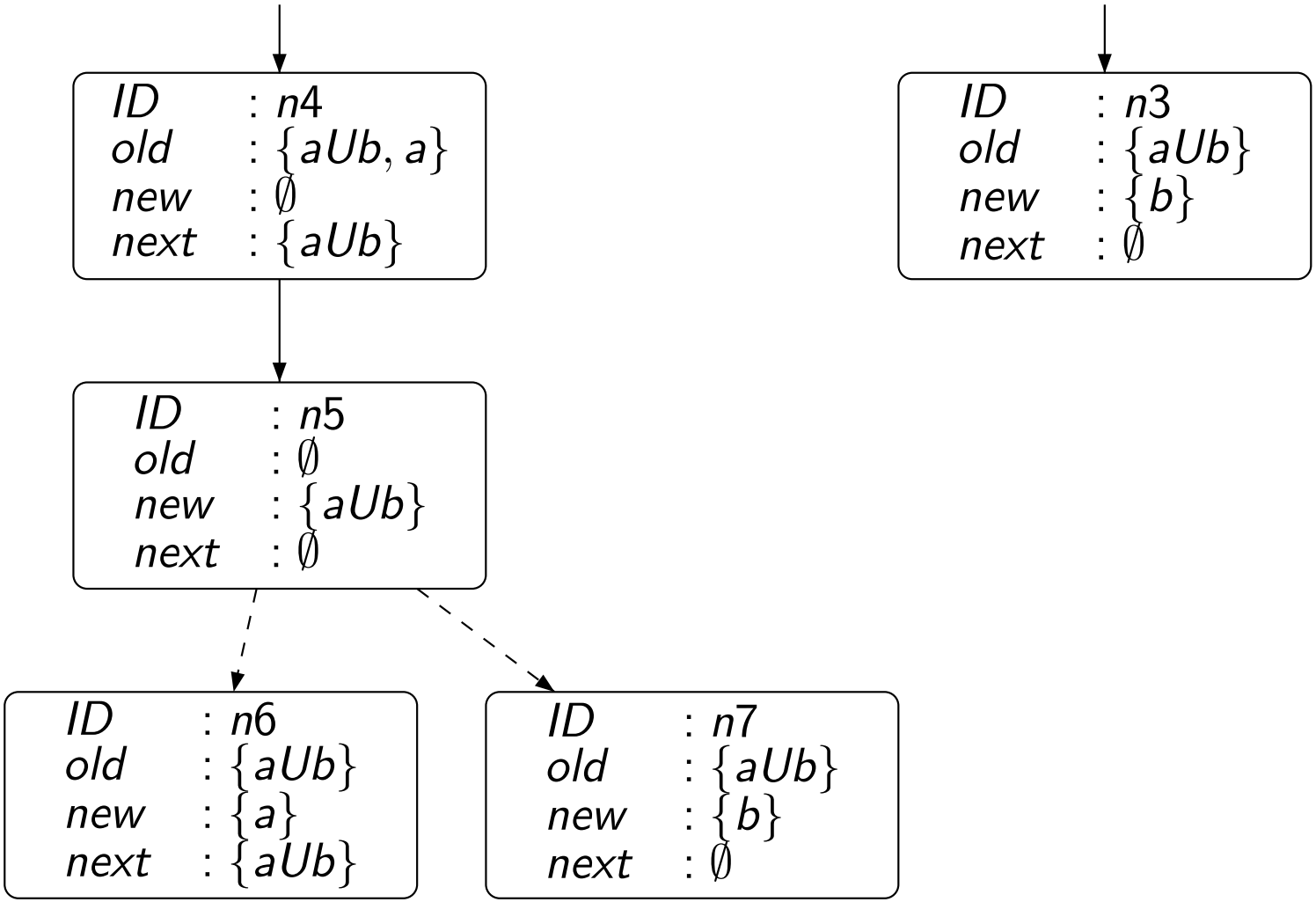
Beispiel: $\varphi = aUb$





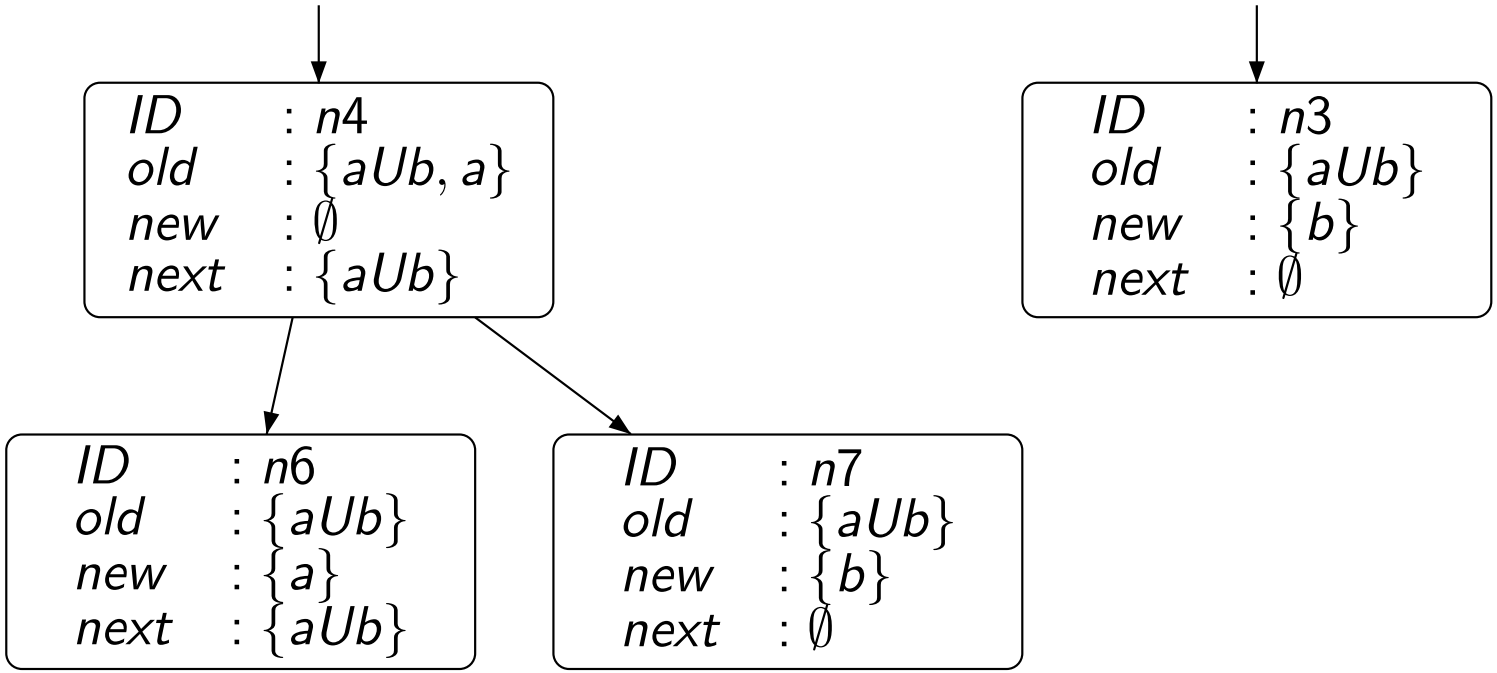
Algorithmus von Gerth, Peled, Vardi und Wolper

Beispiel: $\varphi = aUb$



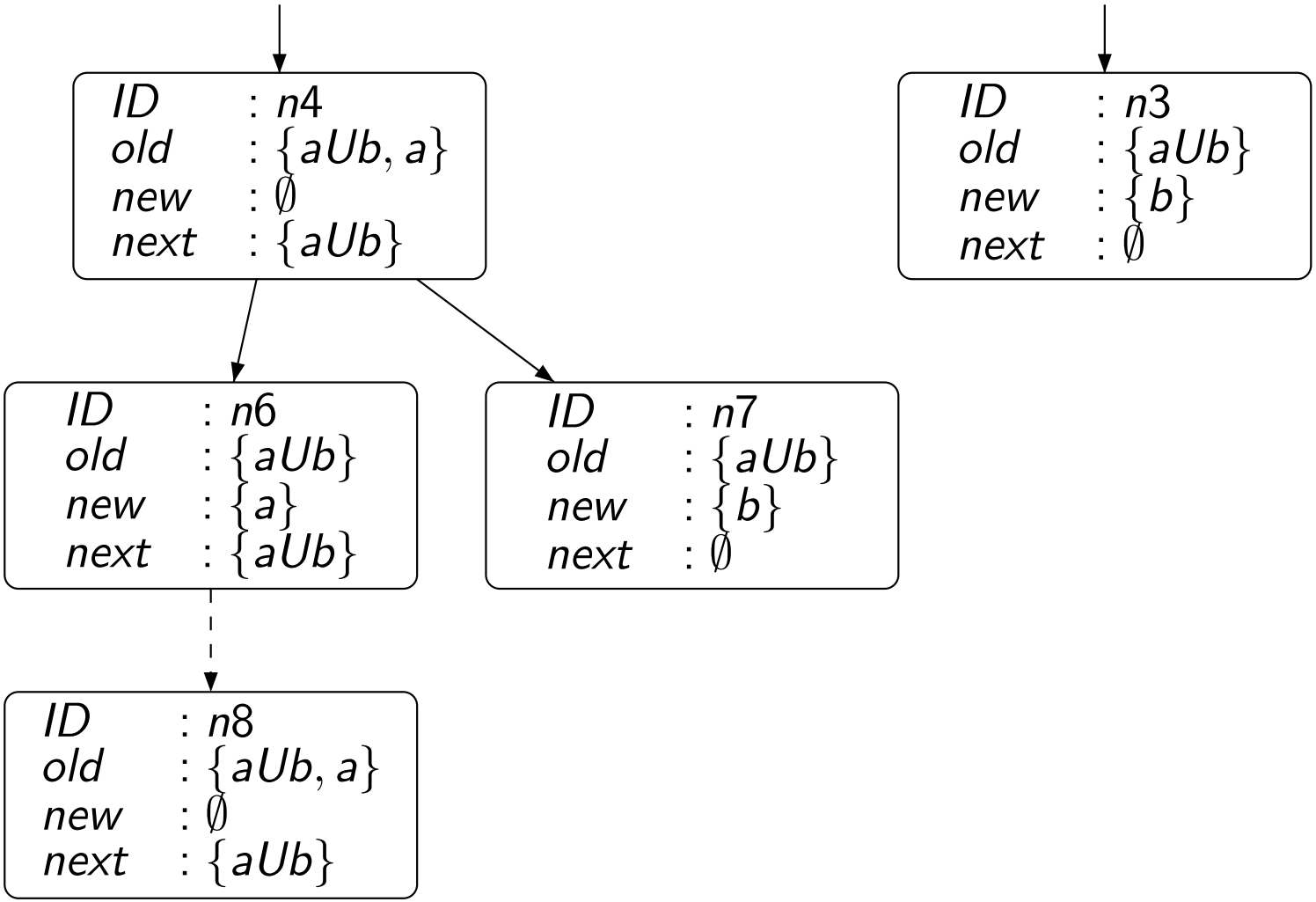
Algorithmus von Gerth, Peled, Vardi und Wolper

Beispiel: $\varphi = aUb$



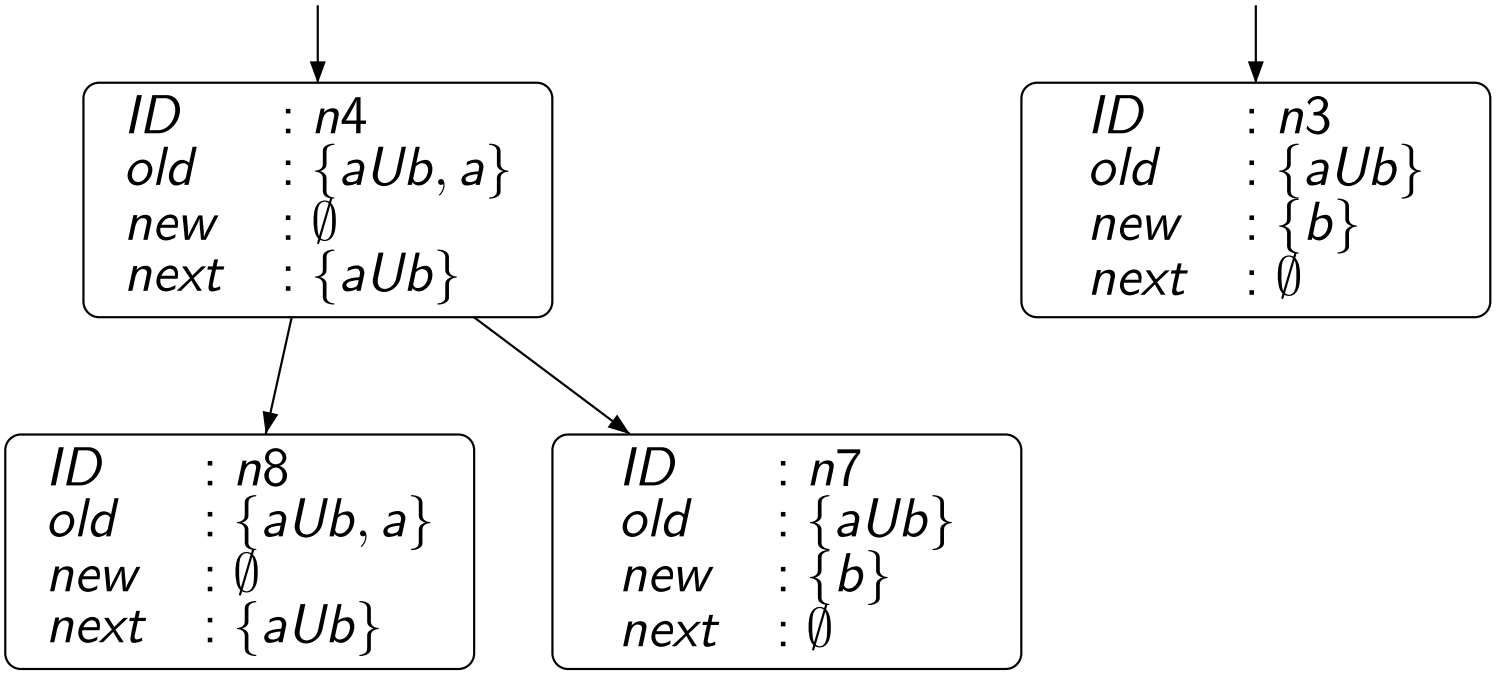
Algorithmus von Gerth, Peled, Vardi und Wolper

Beispiel: $\varphi = aUb$



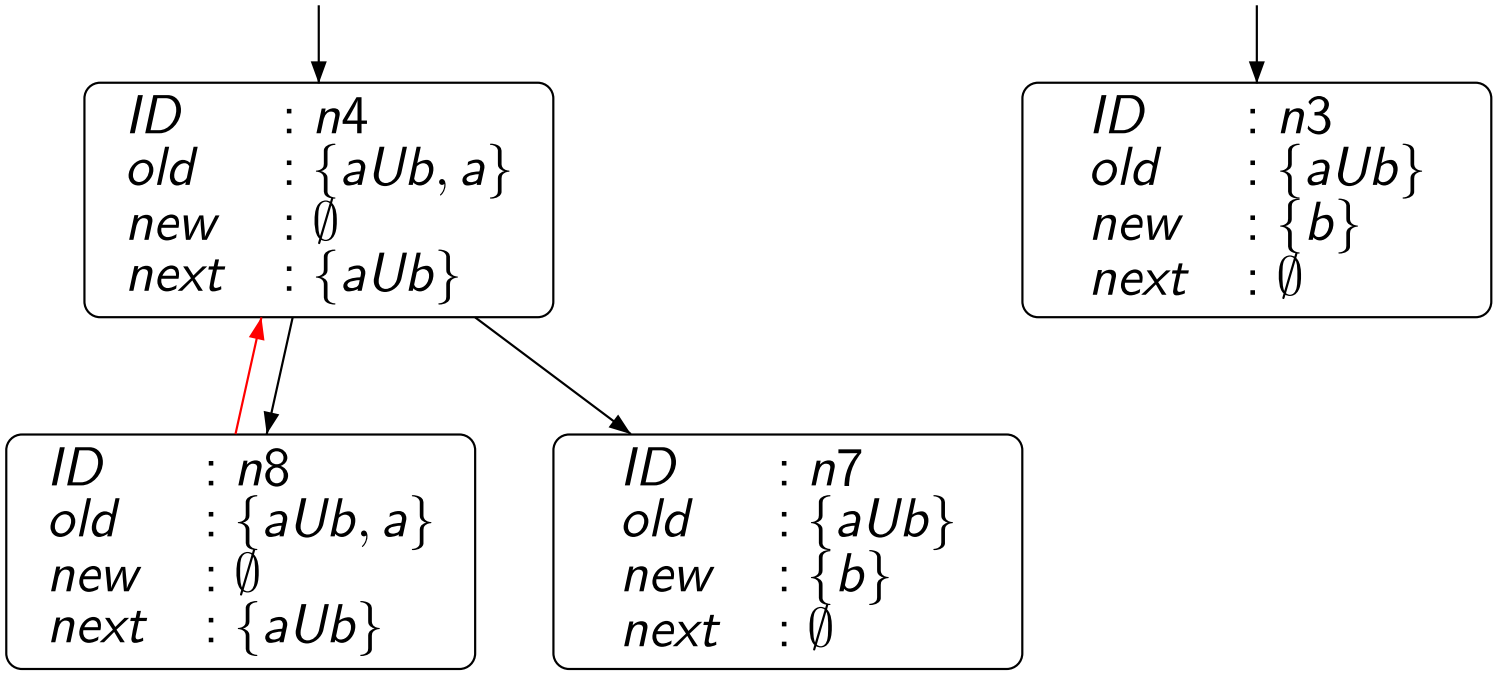
Algorithmus von Gerth, Peled, Vardi und Wolper

Beispiel: $\varphi = aUb$



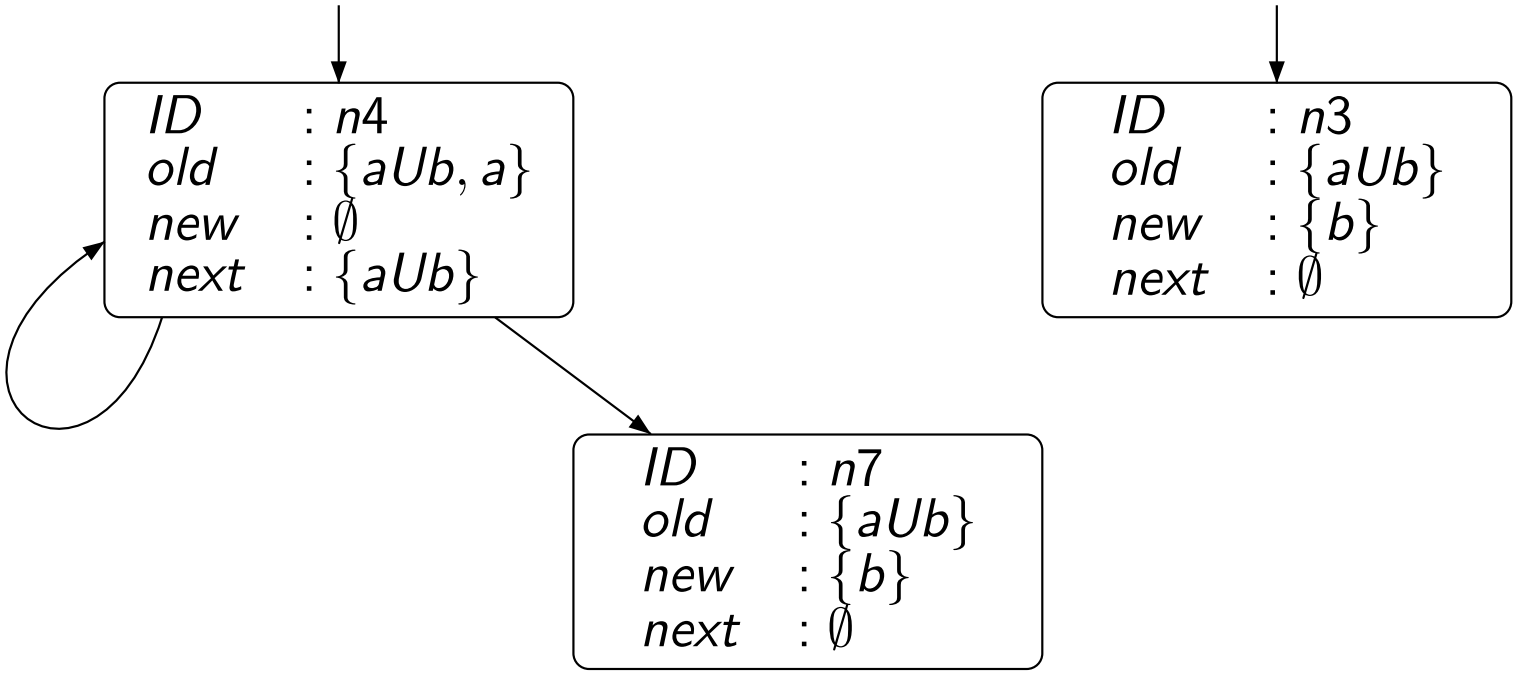
Algorithmus von Gerth, Peled, Vardi und Wolper

Beispiel: $\varphi = aUb$



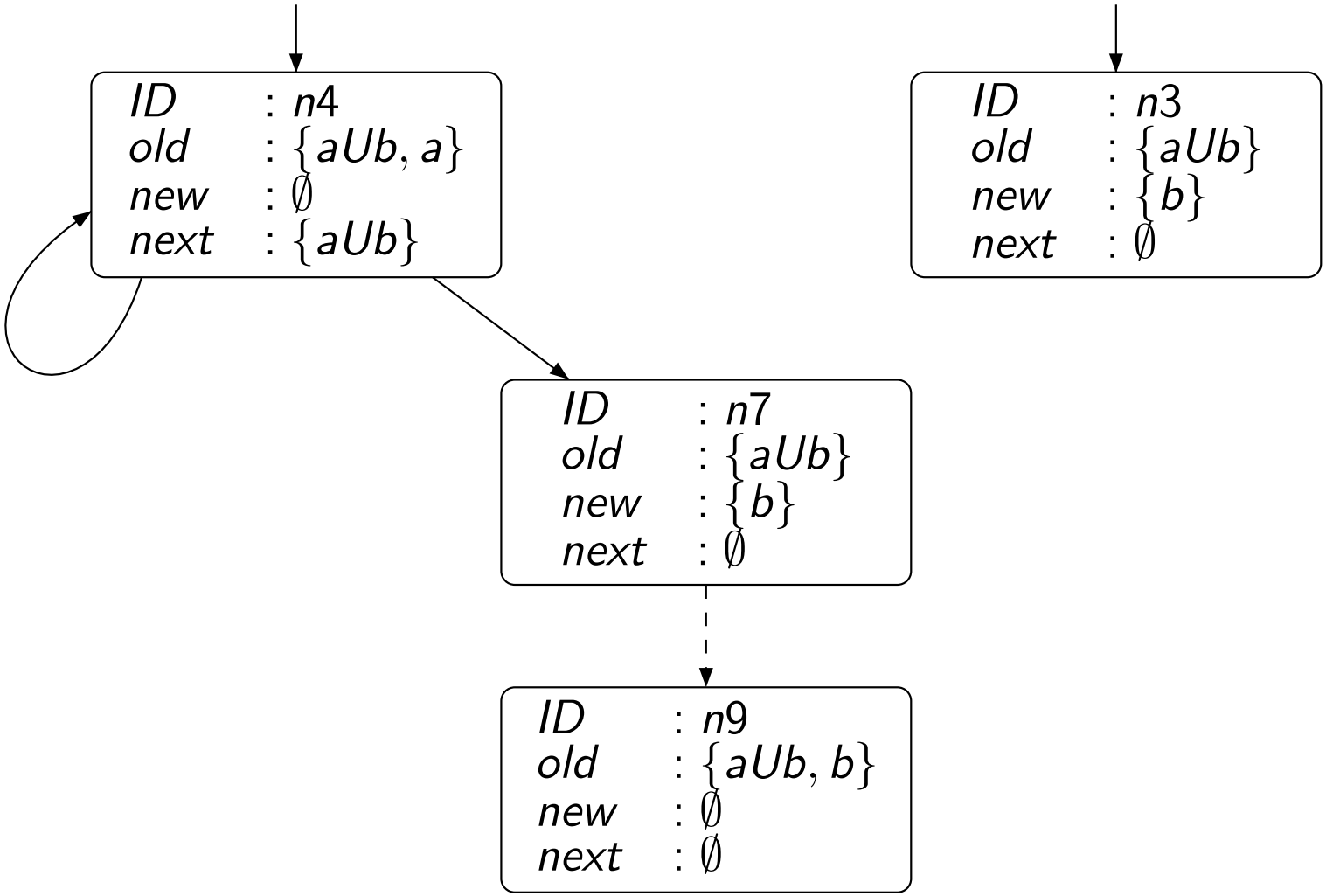
Algorithmus von Gerth, Peled, Vardi und Wolper

Beispiel: $\varphi = aUb$



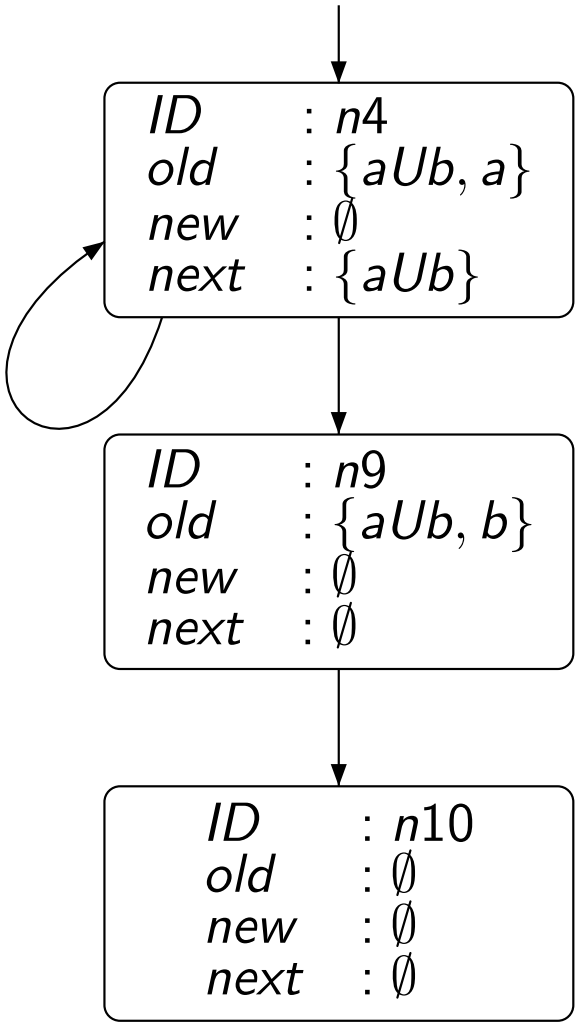
Algorithmus von Gerth, Peled, Vardi und Wolper

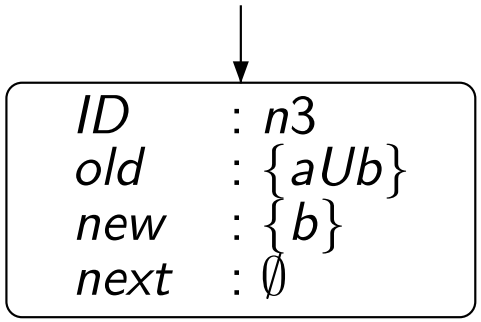
Beispiel: $\varphi = aUb$



Algorithmus von Gerth, Peled, Vardi und Wolper

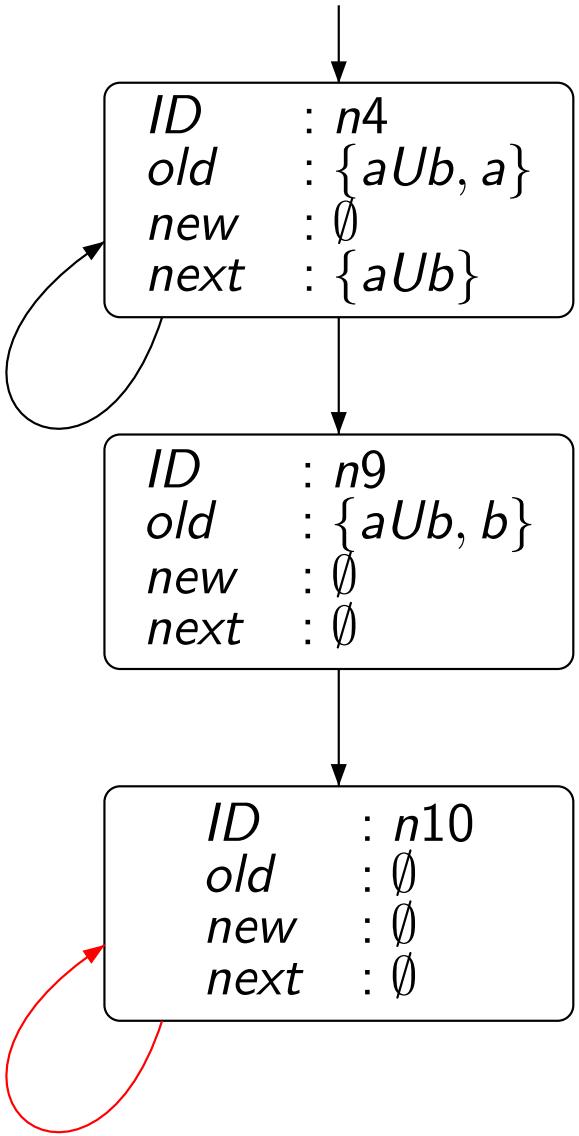
Beispiel: $\varphi = aUb$

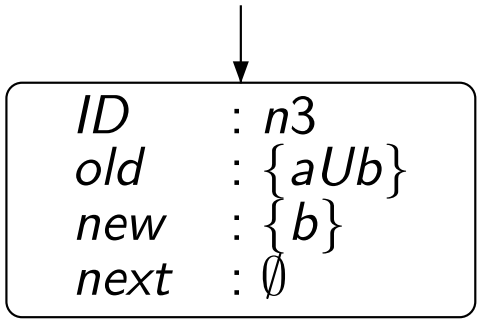




Algorithmus von Gerth, Peled, Vardi und Wolper

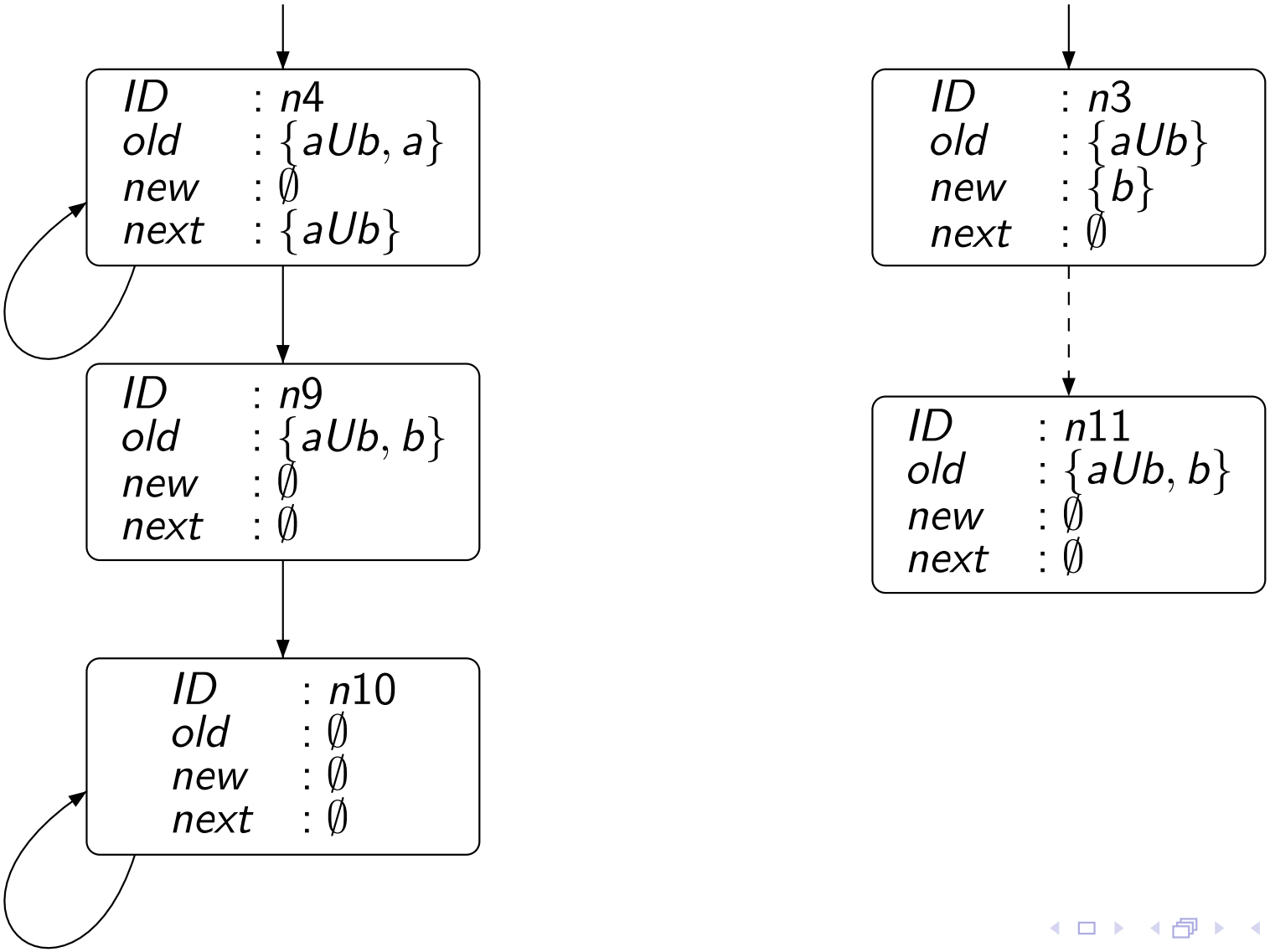
Beispiel: $\varphi = aUb$





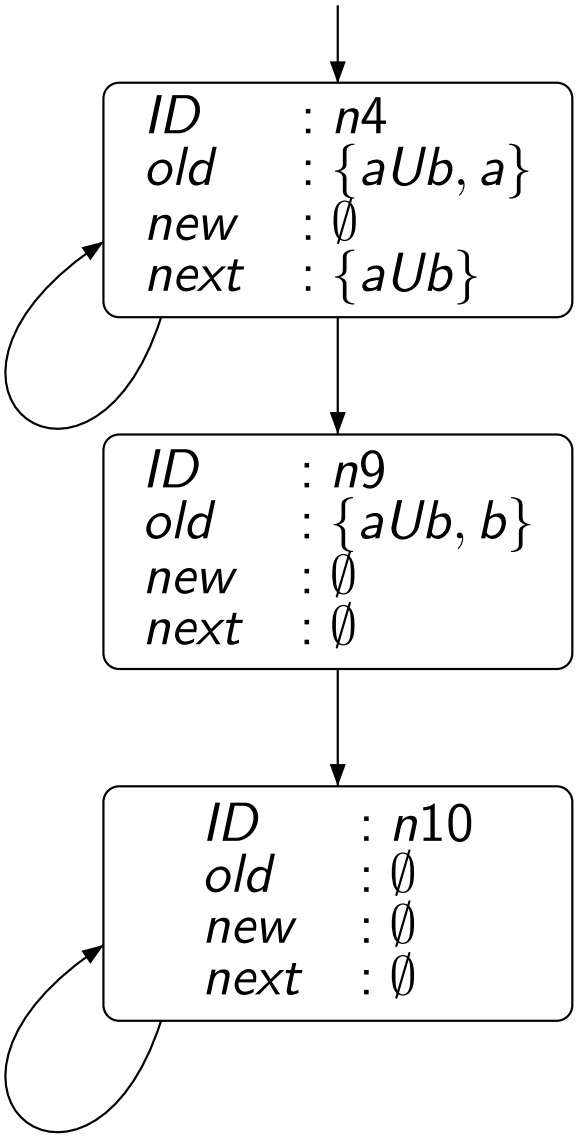
Algorithmus von Gerth, Peled, Vardi und Wolper

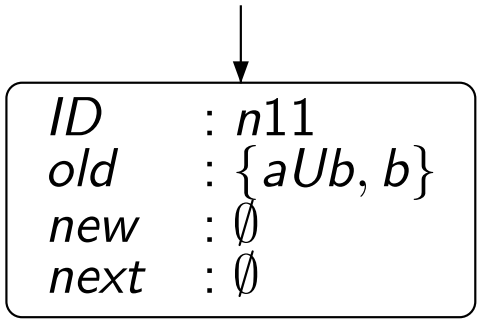
Beispiel: $\varphi = aUb$



Algorithmus von Gerth, Peled, Vardi und Wolper

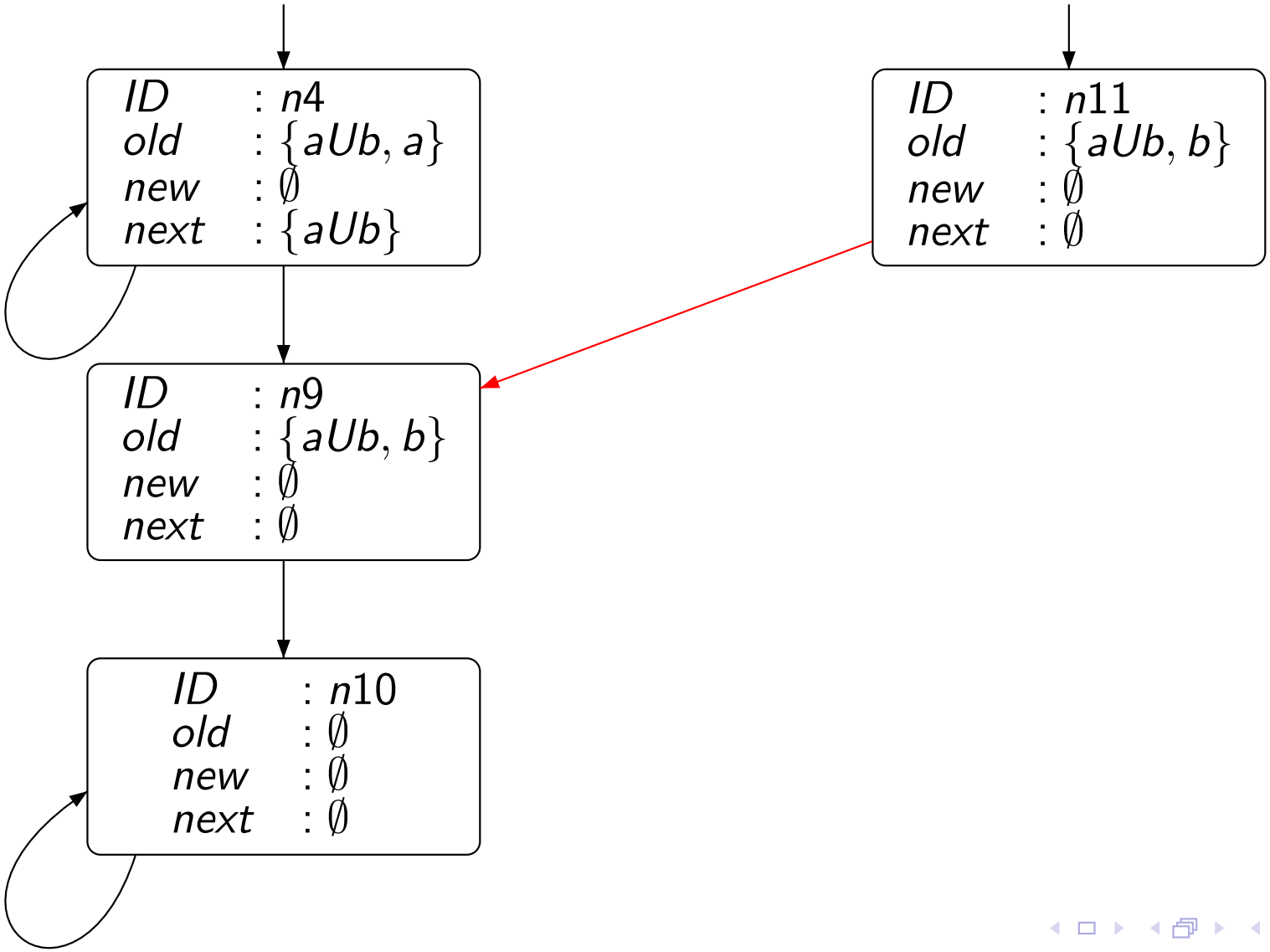
Beispiel: $\varphi = aUb$





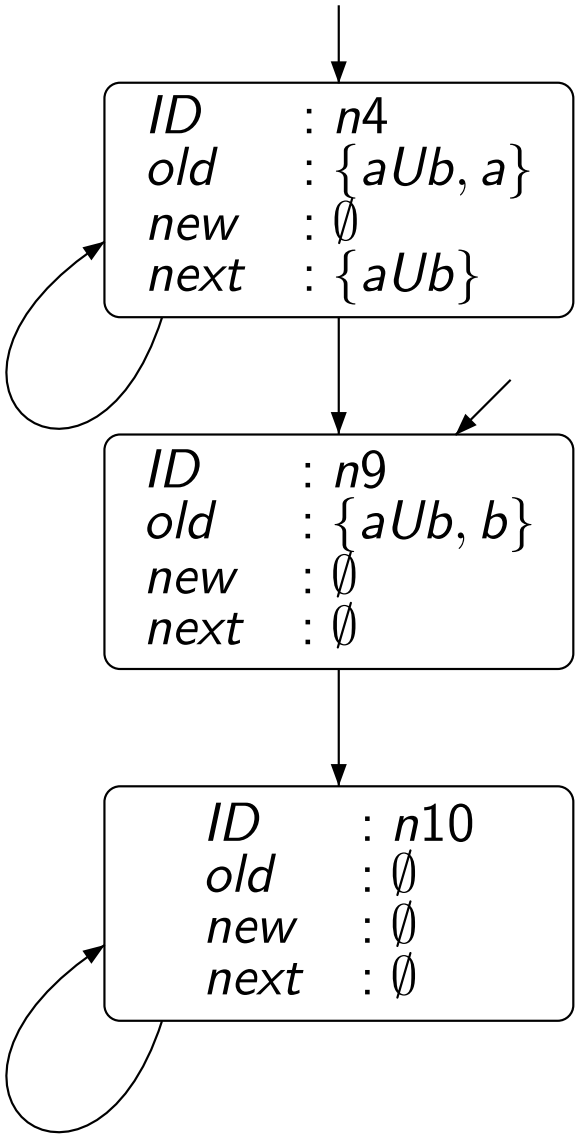
Algorithmus von Gerth, Peled, Vardi und Wolper

Beispiel: $\varphi = aUb$



Algorithmus von Gerth, Peled, Vardi und Wolper

Beispiel: $\varphi = aUb$



Rekursion beendet

Algorithmus von Gerth, Peled, Vardi und Wolper

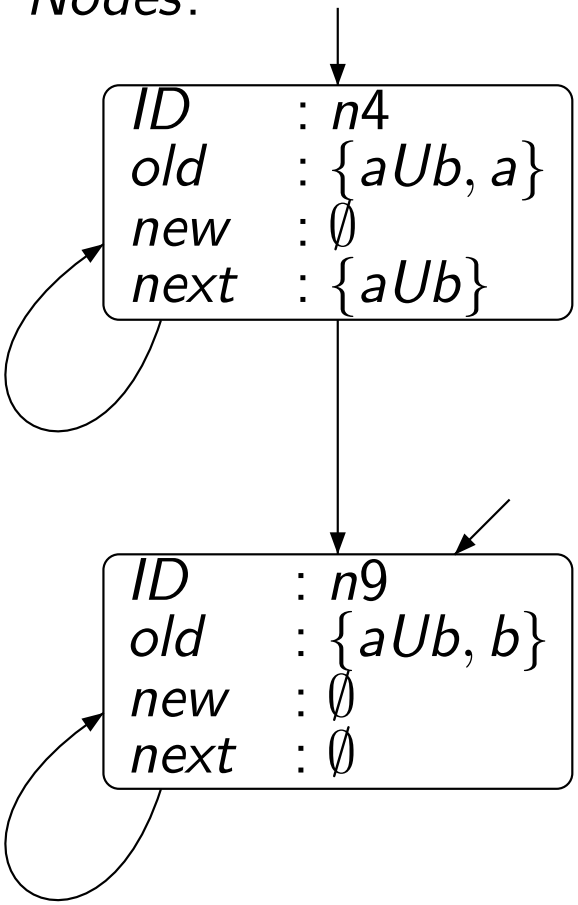
Konstruktion des verallgemeinerten Büchi Automaten:

- $\Sigma =$ Menge der Propositionen
- $Q =$ Menge der Knoten in *Nodes*
- $Q^0 = \mathit{init}$
- $\Delta =$ Menge aller Tripel (r, α, r') mit: $r \in \mathit{incoming}(r')$ und α erfüllt alle nicht-/negierten Propositionen in $\mathit{old}(r')$
- Akzeptanz: Für jede Teilformel der Form $\mu U \psi$ gibt es eine Akzeptanzmenge E_i mit:
  $E_i = \{r \mid \psi \in \mathit{old}(r) \text{ oder } \mu U \psi \notin \mathit{old}(r)\}$

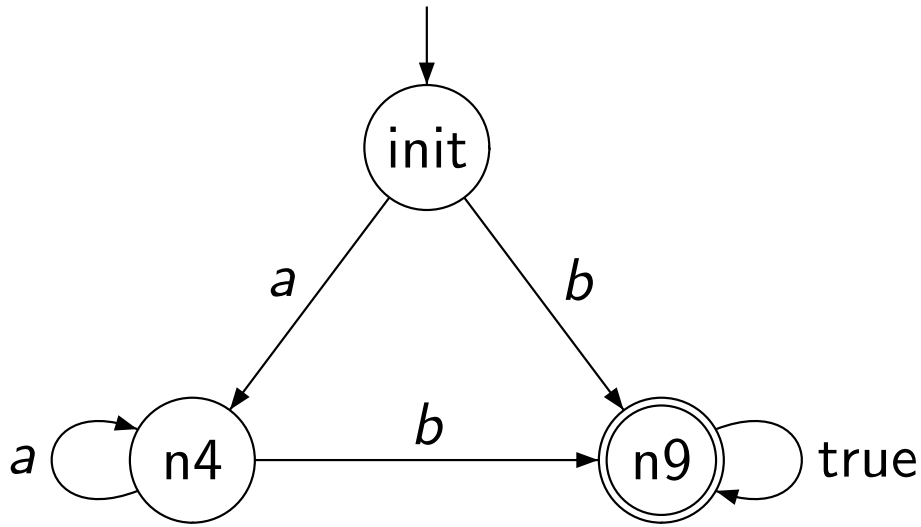
Algorithmus von Gerth, Peled, Vardi und Wolper

Beispiel: $\varphi = aUb$

Nodes:



$\mathcal{B}(Nodes)$:



$\mathcal{B}_1 \cap \mathcal{B}_2$

Modelchecking Algorithmus:

- konstruiere Automat $\mathcal{A}$ zu gegebener Kripke Struktur
- konstruiere Automat $\mathcal{S}$ der $\overline{\mathcal{L}(S)}$ erkennt
- Bilde den Durchschnitt von $\mathcal{A}$ und $\mathcal{S}$
- Prüfe, ob der Automat $\mathcal{A} \cap \mathcal{S}$ leer ist.
  Falls $\mathcal{A} \cap \mathcal{S} \neq \emptyset$: Gegenbeispiel ausgeben

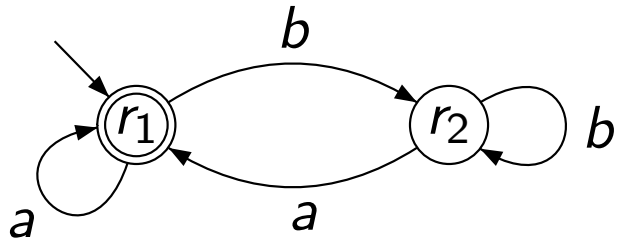
$\mathcal{B}_1 \cap \mathcal{B}_2$

Schnitt zweier Büchi Automaten:

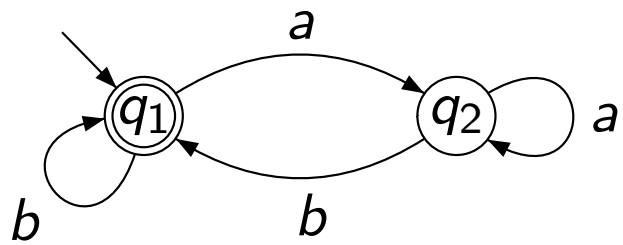
- $\mathcal{B}_1 = \langle \Sigma, Q_1, \Delta_1, Q_1^0, F_1 \rangle, \mathcal{B}_2 = \langle \Sigma, Q_2, \Delta_2, Q_2^0, F_2 \rangle$
- $\mathcal{B}_1 \cap \mathcal{B}_2$ akzeptiert gdw. akzeptierende Zustände von $\mathcal{B}_1$ **und** $\mathcal{B}_2$ unendlich oft durchlaufen werden.
- $\mathcal{B}_1 \cap \mathcal{B}_2 =$
  $\langle \Sigma,\ Q_1 \times Q_2 \times \{0,1,2\},\ \Delta,\ Q_1^0 \times Q_2^0 \times \{0\},\ Q_1 \times Q_2 \times \{2\} \rangle$
- $(\langle r_i, q_j, x \rangle, a, \langle r_m, q_n, y \rangle) \in \Delta$ gdw.
  $(r_i, a, r_m) \in \Delta_1$ und $(q_j, a, q_n) \in \Delta_2$
  Für x und y gilt:
  - $x = 0 \wedge r_m \in F_1 \ \Rightarrow\ y = 1$
  - $x = 1 \wedge q_m \in F_2 \ \Rightarrow\ y = 2$
  - $x = 2 \Rightarrow y = 0$
  - sonst $y = x$

$\mathcal{B}_1 \cap \mathcal{B}_2$ - Beispiel

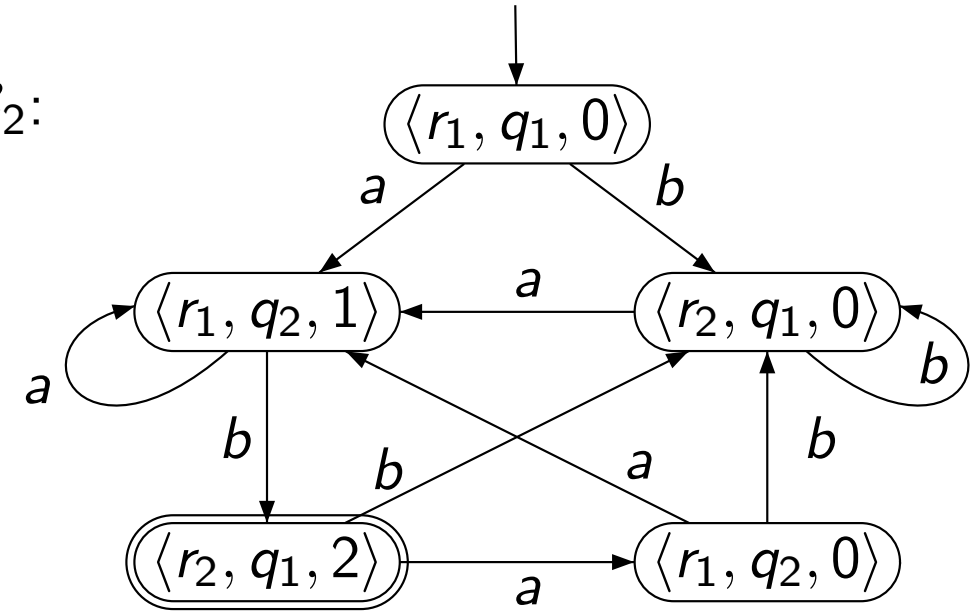
$\mathcal{B}_1$:



$\mathcal{B}_2$:



$\mathcal{B}_1 \cap \mathcal{B}_2$:



Leerheitstest

Model Checking Algorithmus:

- konstruiere Automat $\mathcal{A}$ zu $\mathcal{M}$
- konstruiere Automat $\mathcal{S}$ der $\overline{\mathcal{L}(S)}$ erkennt
- Bilde den Durchschnitt von $\mathcal{A}$ und $\mathcal{S}$
- Prüfe, ob der Automat $\mathcal{A} \cap \mathcal{S}$ leer ist.
  Falls $\mathcal{A} \cap \mathcal{S} \neq \emptyset$: Gegenbeispiel ausgeben

Leerheitstest

Idee des Algorithmus:

Sei ρ ein akzeptierender Lauf über $\mathcal{B}$

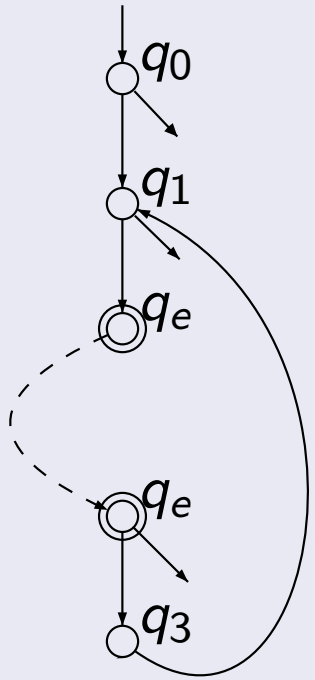
- dann enthält ρ unendlich viele Akzeptanzzustände.
- Da die Menge der Zustände endlich ist, muss ρ von der Form $\rho_1 {\rho_2}^{\omega}$ sein.
- Der Algorithmus sucht einen Pfad ρ_1 zu einem akzeptierenden Zustand q_e und von da einen Pfad ρ_2 zurück zu einem Knoten in ρ_1.
- Der Pfad $\rho_1\rho_2$ besitzt dann einen zyklischen Suffix über q_a und ist das gewünschte Gegenbeispiel.

Leerheitstest

double depth-first-search Algorithmus:

dfs1(q) {
  hash(q);
  for all Nachfolger q' von q **do**
    if $q' \notin$ *hashtable*
      *dfs*1(q');
  if *accept*(q)
    *dfs*2(q);
}

dfs2(q) {
  flag(q);
  for all Nachfolger q' von q **do**
    if $q' \in$ *dfs*1_*stack*
      terminate(*TRUE*);
    else if q' not flagged
      *dfs*2(q');
}



Zyklus gefunden

Leerheitstest

double DFS Algorithmus:

emptiness() {
  for all $q_0 \in Q^0$ **do**
    dfs1(q_0);
  terminate(False);
}

- Der Algorithmus terminiert mit TRUE wenn ein akzeptierender Pfad gefunden wurde.
- Der Stack von *dfs1* enthält den Pfad ρ_1.
- Der Stack von *dfs2* enthält den Pfad ρ_2.
- Worst-Case-Laufzeit: $\mathcal{O}(|Q| + |\Delta|)$

Model Checking mit Büchi Automaten

Zusammenfassung:

- Konstruktion von $\mathcal{A}$: nahezu kein Aufwand, falls die Kripke Struktur schon berechnet ist
- Konstruktion von $\mathcal{S}$: exponentiell zu φ
- Konstruktion von $\mathcal{A}$ und $\mathcal{S}$: $\mathcal{O}(|\mathcal{A}| \times |\mathcal{S}|)$
- Leerheitstest: $\mathcal{O}(|Q| + |\Delta|)$ im Worst-Case

On-the-Fly Model Checking

Idee:

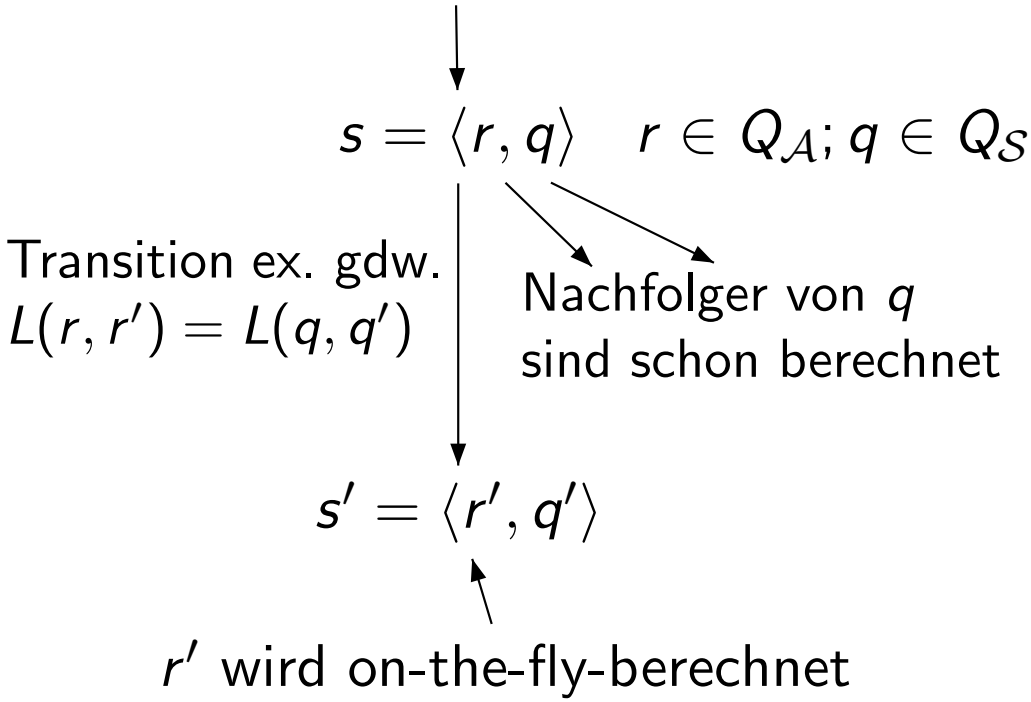
- Zuerst $\mathcal{S}$ erstellen
- Zustände von $\mathcal{S} \cup \mathcal{A}$ so berechnen, wie sie der double-DFS-Algorithmus benötigt.
- Dabei werden immer nur die nötigen Zustände aus $\mathcal{A}$ berechnet.

Vorteil:

- viele Zustände aus $\mathcal{A}$ werden nie erzeugt.
- während der Schnittberechnung werden Zyklen erkannt.

Verfahren:

Sei s der aktuelle Zustand, in dem sich der Suchalgorithmus befindet



- Falls $L(r, r') \neq L(q, q')$ wird abgebrochen, und die Nachfolger von r' werden nicht berechnet.
- Wird vor dem Backtracking zu s ein Zyklus gefunden wird abgebrochen, und die Nachfolger von s werden nicht berechnet.