UNIVERSITÄT ZU LÜBECK
INSTITUTE OF SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

isp

Transformation von regulärer Linearzeit-Temporallogik zu Paritätsautomaten

Kolloquium zur Bachelorarbeit von

Malte Schmitz

9. Februar 2012

ausgegeben und betreut von

Prof. Dr. Martin Leucker, ISP, Uni Lübeck

Einleitung

- Reguläre Linearzeit-Temporallogik (RLTL) ist neue Temporallogik.
- RLTL vereint LTL und ω-reguläre Ausdrücke.
  - Verallgemeinerung der LTL-Operatoren
  - volle Mächtigkeit ω-regulärer Ausdrücke
  - Negation und Konjunktion
- Ziel: Einsatz von RLTL in
  - Model Checking
  - und Runtime Verification.
- Umwandlung von RLTL zu Büchi-Automaten
  - Malte Schmitz: RLTL zu 2APW
  - Torben Scheffel: APW zu Büchi-Automaten
  - Johannes Thorn: Zusammenbau als Webservice

Gliederung

1. **RLTL**
   - Definition
   - Beispiel
2. **APWs**
   - Definitionen
   - Beispiel
3. **Umwandlung**
   - reguläre Ausdrücke
   - RLTL
   - Beispiel
4. **Implementierung**
   - Datenstrukturen
   - Anwendung

Idee

- Power-Operator verallgemeinert bekannte Operatoren.
  - Die Pflicht muss
  - mit jedem Lesen der Verzögerung gelten,
  - bis der Versuch diese Schleife beendet.

- LTL-Ausdruck $x \mathcal{U} y$
  - Pflicht: x
  - (implitite) Verzögerung: ein Zeichen
  - Versuch: y
- ω-regulärer Ausdruck $a; b^\omega$
  (Konkatenation von a und)
  - keine Pflicht (alles wird akzeptiert),
  - Verzögerung: b
  - kein Versuch (unendliche Wiederholung)

Veröffentlichungen

Regular Linear Temporal Logic*

Martin Leucker[1] and César Sánchez[2,3]

[1] Institut für Informatik
TU München, Germany
[2] Computer Science Department
Stanford University, Stanford, USA
[3] Computer Engineering Department
University of California, Santa Cruz, USA

Abstract. We present regular linear temporal logic (RLTL), a logic that generalizes linear temporal logic with the ability to use regular expressions arbitrarily as sub-expressions. Every LTL operator can be defined as a context in regular linear temporal logic. This implies that there is a (linear) translation from LTL to RLTL.
Unlike LTL, regular linear temporal logic can define all ω-regular languages, while still keeping the satisfiability problem in PSPACE. Unlike the extended temporal logics ETL_*, RLTL is defined with an algebraic signature. In contrast to the linear time μ-calculus, RLTL does not depend on fix-points in its syntax.

1 Introduction

We present *regular linear temporal logic* (RLTL), a formalism to express properties of infinite traces by conveniently *fusing* regular-expressions and linear-temporal logic. Moreover, we show that the satisfiability and equivalence of RLTL expressions are PSPACE-complete problems.

The linear temporal logic (LTL) [19, 16] is a modal logic over a linear frame, whose formulas express properties of infinite traces using two modalities: *next-time* and *until*. LTL is a widely accepted formalism for the specification and verification of concurrent and reactive systems. However, Wolper [26] showed

Veröffentlichungen

Regular Linear Temporal Logic*

Regular Linear Temporal Logic with Past

César Sánchez[1,2] and Martin Leucker[3]

[1] Madrid Institute for Advanced Studies (IMDEA Software), Spain
[2] Spanish Council for Scientific Research (CSIC), Spain
[3] Technische Universität München, Germany

Abstract. This paper upgrades Regular Linear Temporal Logic (RLTL) with past operators and complementation. RLTL is a temporal logic that extends the expressive power of linear temporal logic (LTL) to all ω-regular languages. The syntax of RLTL consists of an algebraic signature from which expressions are built. In particular, RLTL does not need or expose fix-point binders (like linear time μ-calculus), or automata to build and instantiate operators (like ETL_*).

Past operators are easily introduced in RLTL via a single previous-step operator for basic state formulas. The satisfiability and model checking problems for RLTL are PSPACE-complete, which is optimal for extensions of LTL. This result is shown using a novel linear size translation of RLTL expressions into 2-way alternating parity automata on words. Unlike previous automata-theoretic approaches to LTL, this construction is compositional (bottom-up). As alternating parity automata can easily be complemented, the treatment of negation is simple and does not require an upfront transformation of formulas into any normal form.

1

W
er
te
R

w
ti

1 Introduction

Veröffentlichungen

Regular Linear Temporal Logic*

Regular Linear Temporal Logic with Past

Efficient Regular Linear Temporal Logic using Dualization and Stratification

César Sánchez[1,2] and Julián Samborski-Forlese[1]

[1]IMDEA Software Institute, Madrid, Spain
[2]Institute for Applied Physics, CSIC, Spain

Abstract. We study efficient translations of Regular Linear Temporal Logic (RLTL) into automata on infinite words.
RLTL is a temporal logic that fuses Linear Temporal Logic (LTL) with regular expressions, extending its expressive power to all ω-regular languages.
The first contribution of this paper is a novel bottom up translation from RLTL into alternating parity automata of linear size that requires only colors 0, 1 and 2, and that enjoy the stratified internal structure of hesitant automata. Our translation is defined inductively for every operator, and does not require an upfront transformation of the expression into a normal form. Our construction builds at every step two automata: one equivalent to the formula and another to its complement. Inspired by this construction, the second contribution of this paper is to extend RLTL with new operators, including universal sequential composition,

1

W
er
te
R

w
ti

1

reguläre Linearzeit-Temporallogik (RLTL)

Definition (Syntax von RLTL in EBNF)

```
RLTL = B { "∨" B }                    // Disjunktion
   B = O { "∧" O }                    // Konjunktion
   O = E "/" RE "⟩⟩" E                // Power-Operatoren
     | E "/" RE "⟩" E | E
   E = RE ";" E | C                   // Konkatenation
   C = "∅" | "¬" C                    // Komplement
     | "(" RLTL ")"
```

- `RE` ist regulärer Ausdruck, der leeres Wort nicht akzeptiert.
- Semantik von Disjunktion, Konjunktion und Konkatenation „wie immer“.

Semantik der Power-Operatoren

$(w, i) \models_{\mathtt{RLTL}} a/x\rangle\rangle b$

- entweder $(w, i) \models_{\mathtt{RLTL}} b$ (Versuch)
- oder Folge $(i, i_1, \ldots, i_m)$ existiert, sodass
  - $(w, i_k, i_{k+1}) \models_{\mathtt{RE}} x$ (Verzögerung)
  - und $(w, i_k) \models_{\mathtt{RLTL}} a$ (Pflicht)
  - und $(w, i_m) \models_{\mathtt{RLTL}} b$ (Versuch)

$(w, i) \models_{\mathtt{RLTL}} a/x\rangle b$ auch, wenn
Folge $(i, i_1, \ldots)$ existiert, sodass

- $(w, i_k, i_{k+1}) \models_{\mathtt{RE}} x$ (Verzögerung)
- und $(w, i_k) \models_{\mathtt{RLTL}} a$ (Pflicht)
- (Versuch wird nicht erreicht.)

Beispiel

- RLTL-Formel $a; \neg\emptyset / \operatorname{true}\operatorname{true}\rangle\emptyset$
  über Alphabet $\Sigma = \{a, b\}$
- entspricht LTL-Formel $\Box a$ mit angepasster Verzögerung
- Prüft, ob jedes zweite Zeichen $a; \neg\emptyset$ erfüllt.

akzeptierte Worte

$$
\begin{aligned}
(ab)^\omega &= ababab\ldots, \\
a^\omega &= aaa\ldots, \\
(abaa)^\omega &= abaaabaa\ldots, \\
&\vdots
\end{aligned}
$$

nicht akzeptierte Worte

$$
\begin{aligned}
(ba)^\omega &= bababa\ldots, \\
(baaa)^\omega &= baaabaaa\ldots, \\
b^\omega &= bbb\ldots, \\
&\vdots
\end{aligned}
$$

alternierende Paritätsautomaten (APWs)

- APWs sind auf unendlichen Worten definiert
- APWs sind Verallgemeinerung von nicht-deterministischen und universellen Automaten.
- Pfad akzeptiert, wenn maximale Parität aller unendliche oft besuchter Zustände gerade ist.
- alle Pfade akzeptierend $\Longrightarrow$ Lauf akzeptiert
- ein Lauf akzeptierend $\Longrightarrow$ APW akzeptiert

positive boolsche Formel

Transitionen bilden auf $\mathcal{B}^+(Q)$ statt auf Q ab.

Beispiel

$\psi = (q_1 \vee q_2) \wedge q_3$

minimale Modelle von ψ: $\{q_1, q_3\}, \{q_2, q_3\}$

alternierende Paritätsautomaten (APWs)

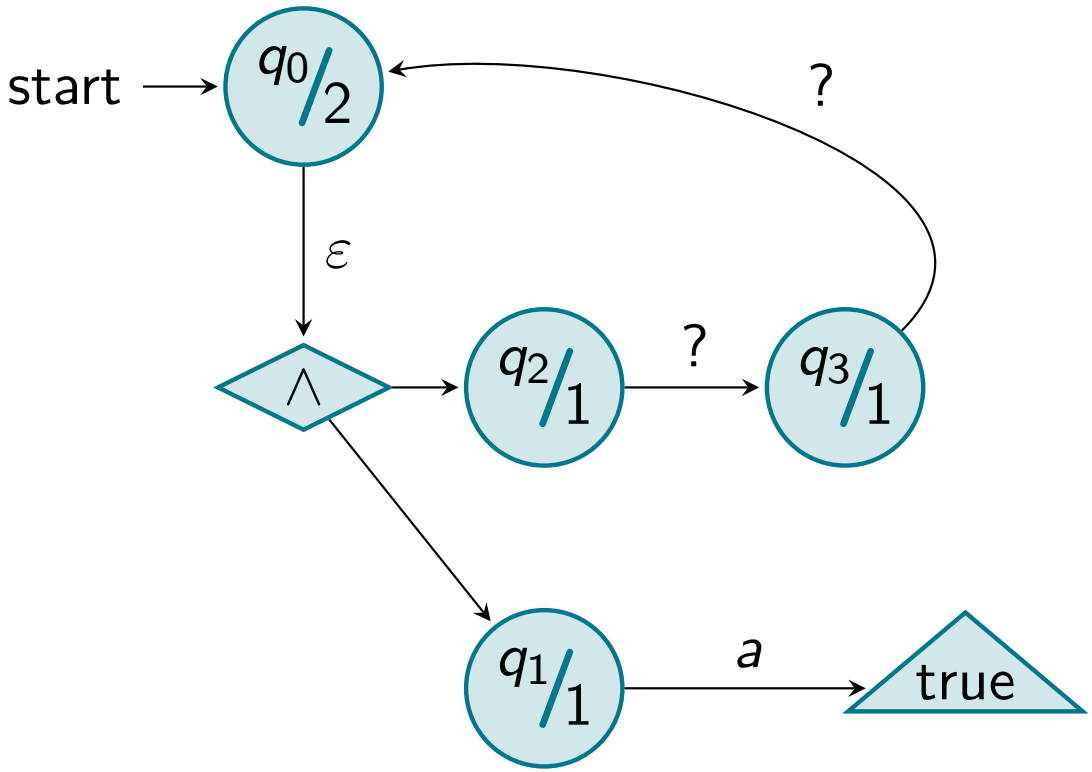
Ein APW $(\Sigma, Q, Q_0, \delta, F)$ besteht aus

- einem Eingabealphabet Σ,
- einer Menge Q von Zuständen,
- der initialen Belegung $Q_0 \in \mathcal{B}^+(Q)$,
- der Transitionsfunktion $\delta : (Q \times \Sigma) \rightsquigarrow \mathcal{B}^+(Q)$
- und der Paritätsfunktion $F : Q \to \mathbb{N}$.

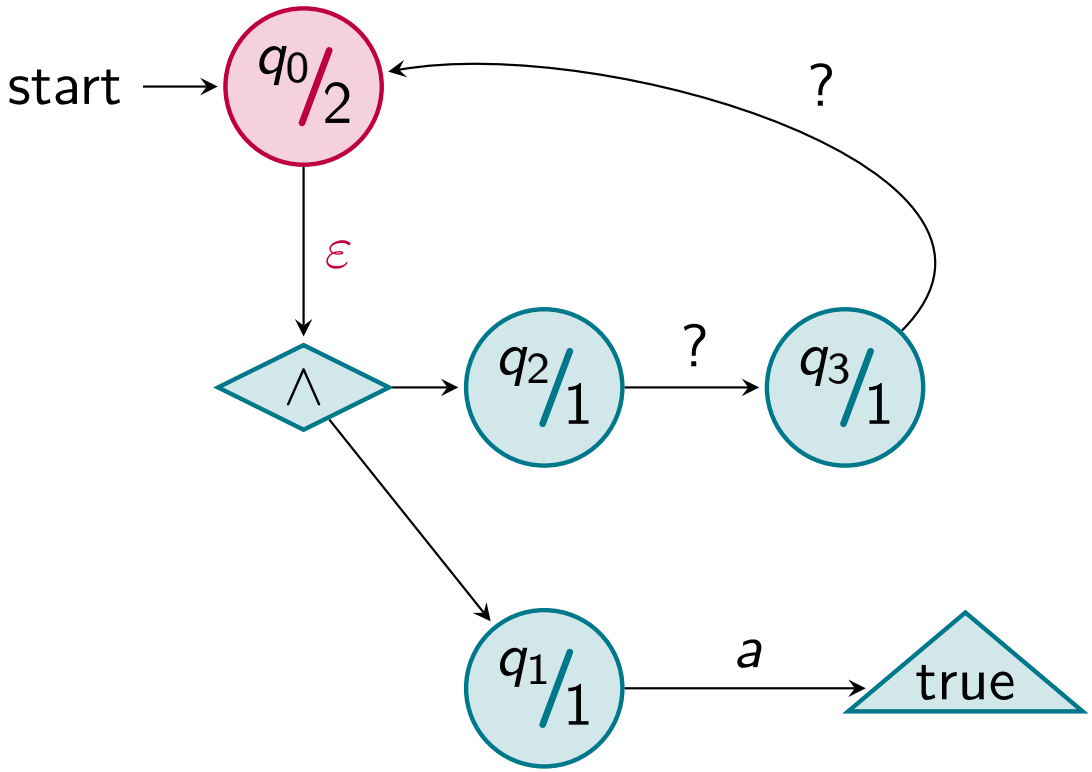
Definition (Lauf eines APWs)

- Lauf ist DAG über Konfigurationen.
- Wurzeln sind minimales Modell von Q_0.
- Nachfolger eines Knotens sind minimales Modell der entsprechenden Transition.

Beispiel eines APWs



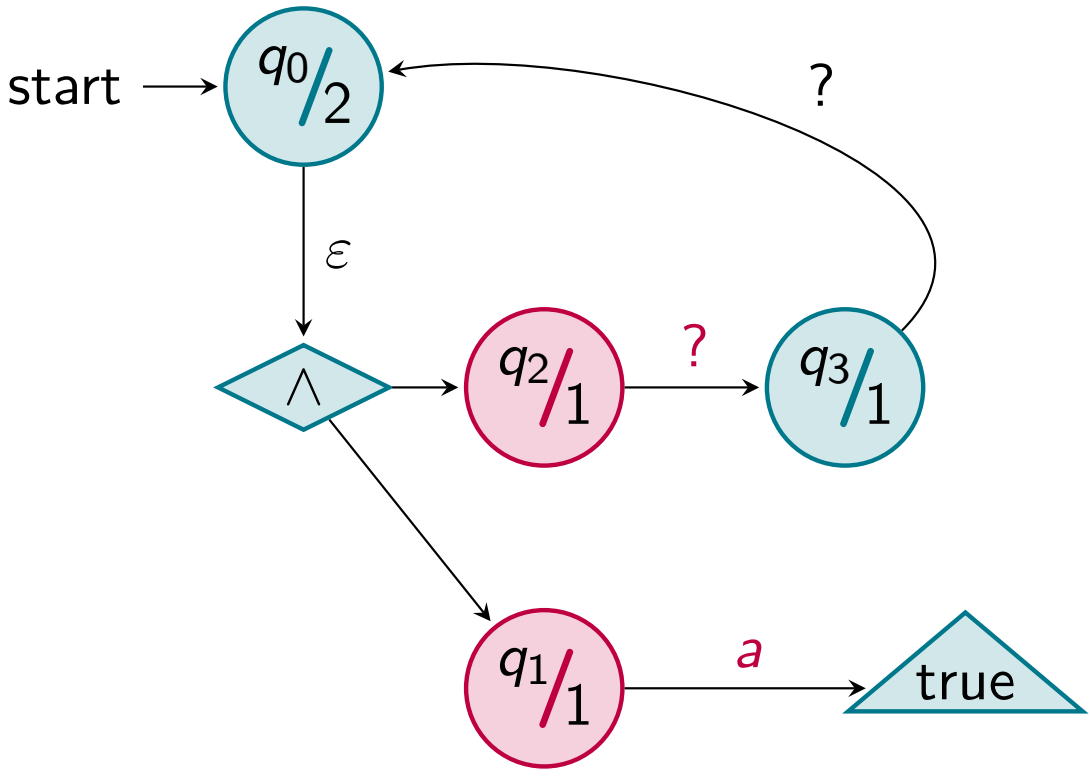
Beispiel eines APWs



Beispiel

a b a a b . . .

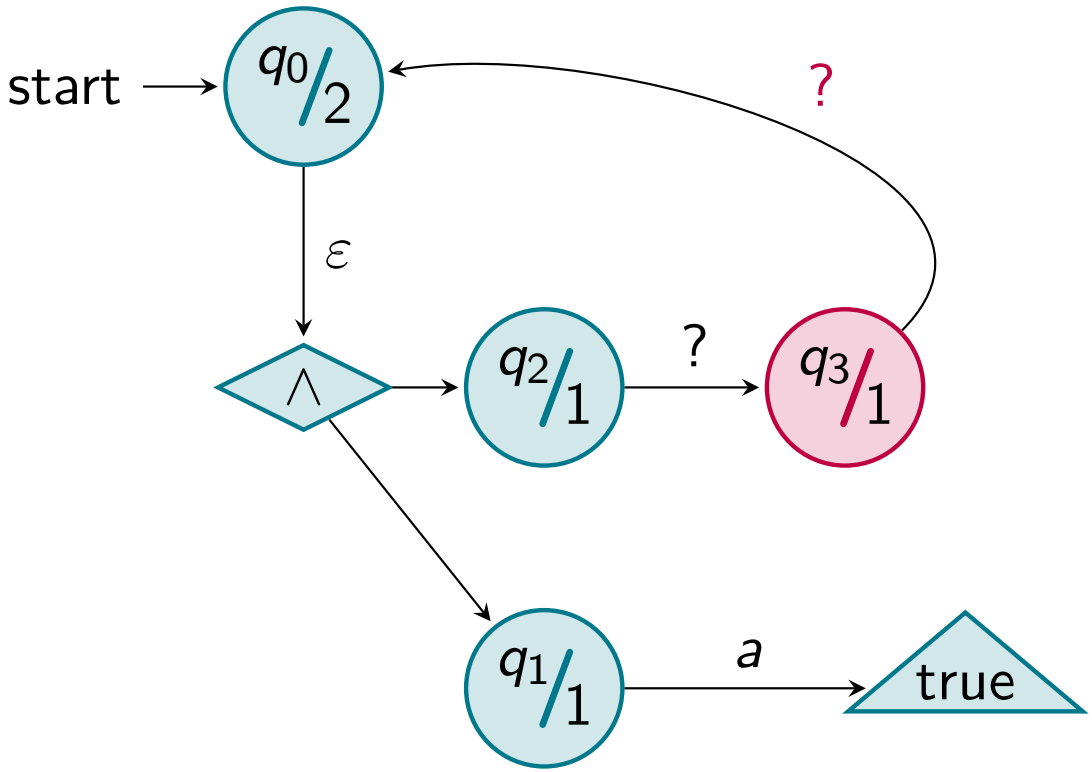
Beispiel eines APWs



Beispiel

a b a a b . . .

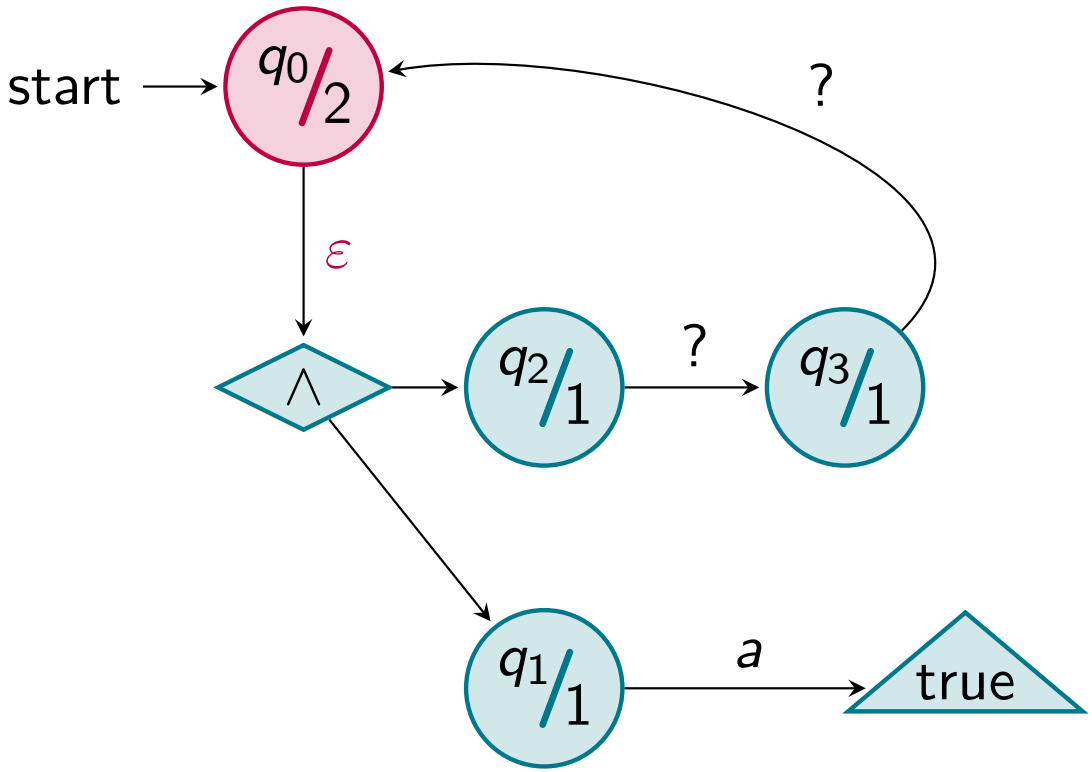
Beispiel eines APWs



Beispiel

a b a a b . . .

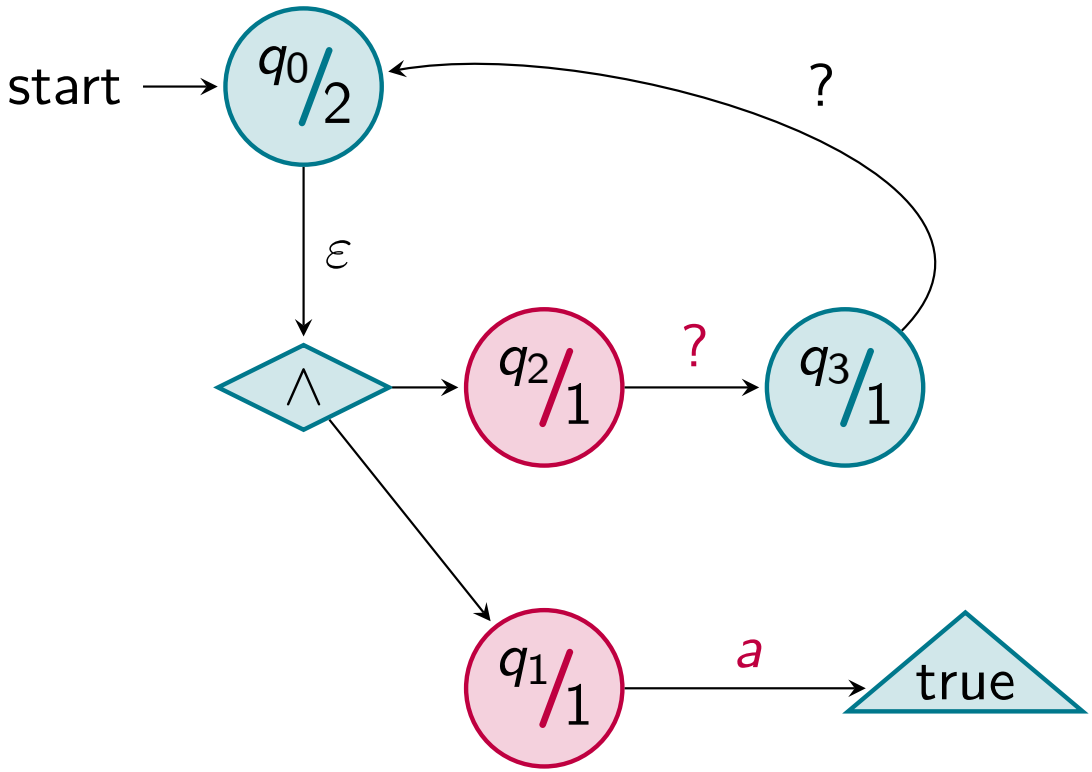
Beispiel eines APWs



Beispiel

a b a a b . . .

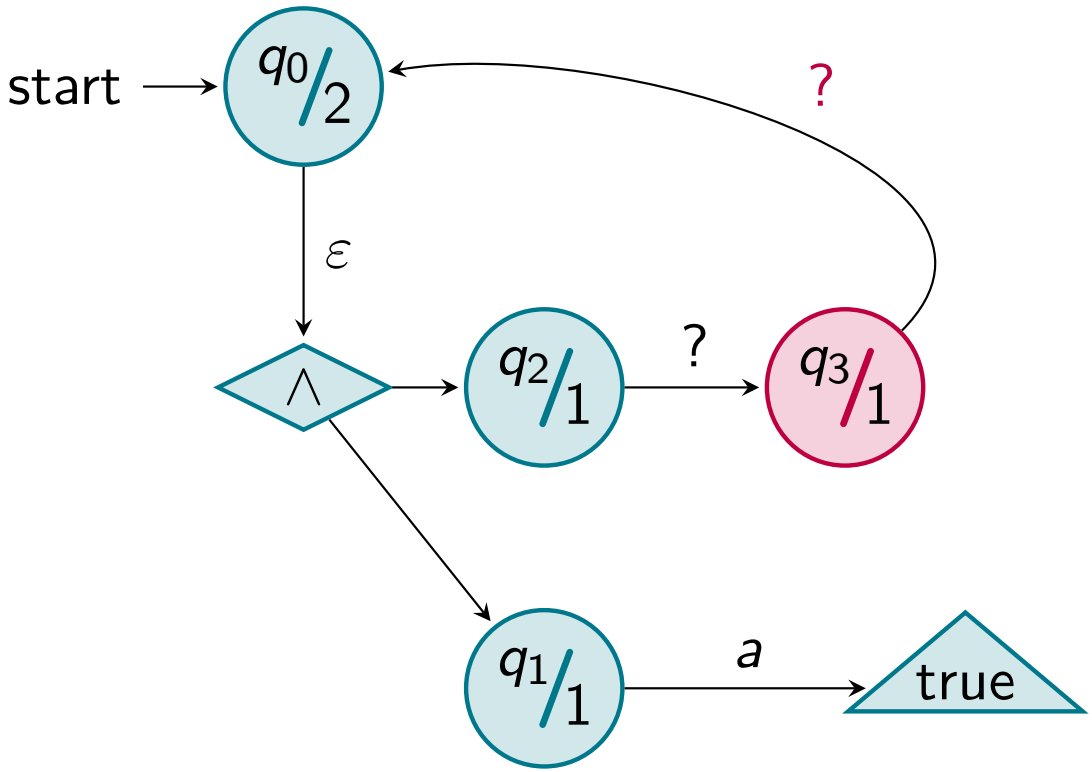
Beispiel eines APWs



Beispiel

a b a a b . . .

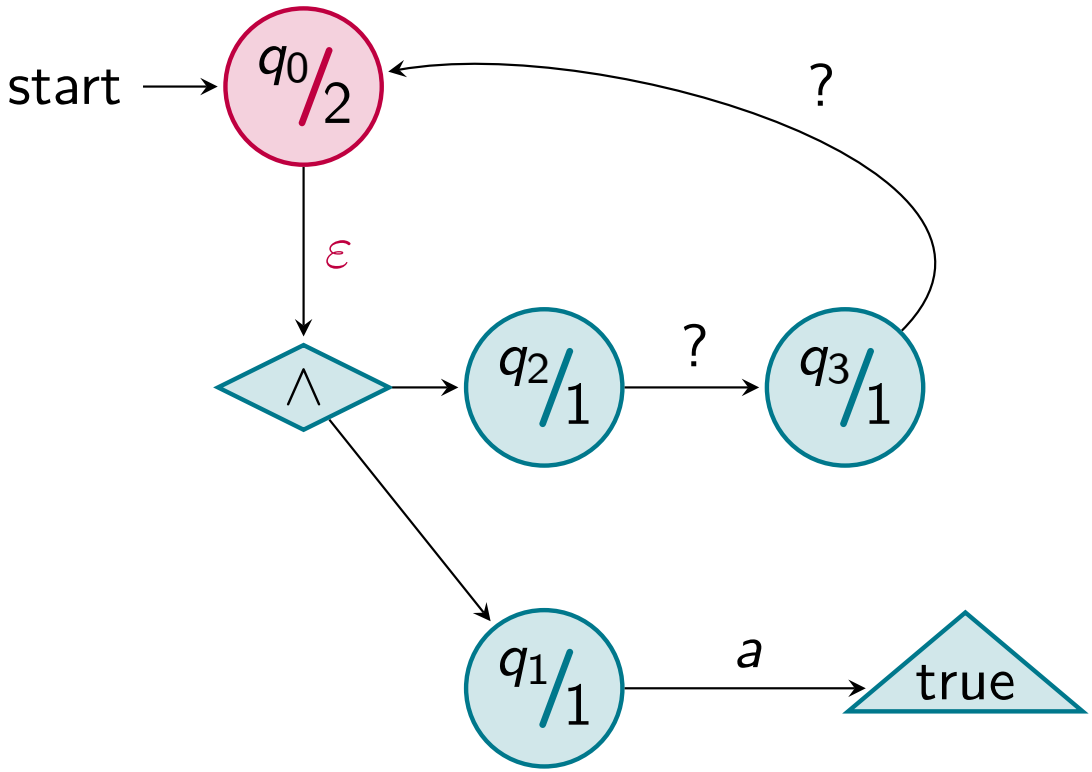
Beispiel eines APWs



Beispiel

a b a a b . . .

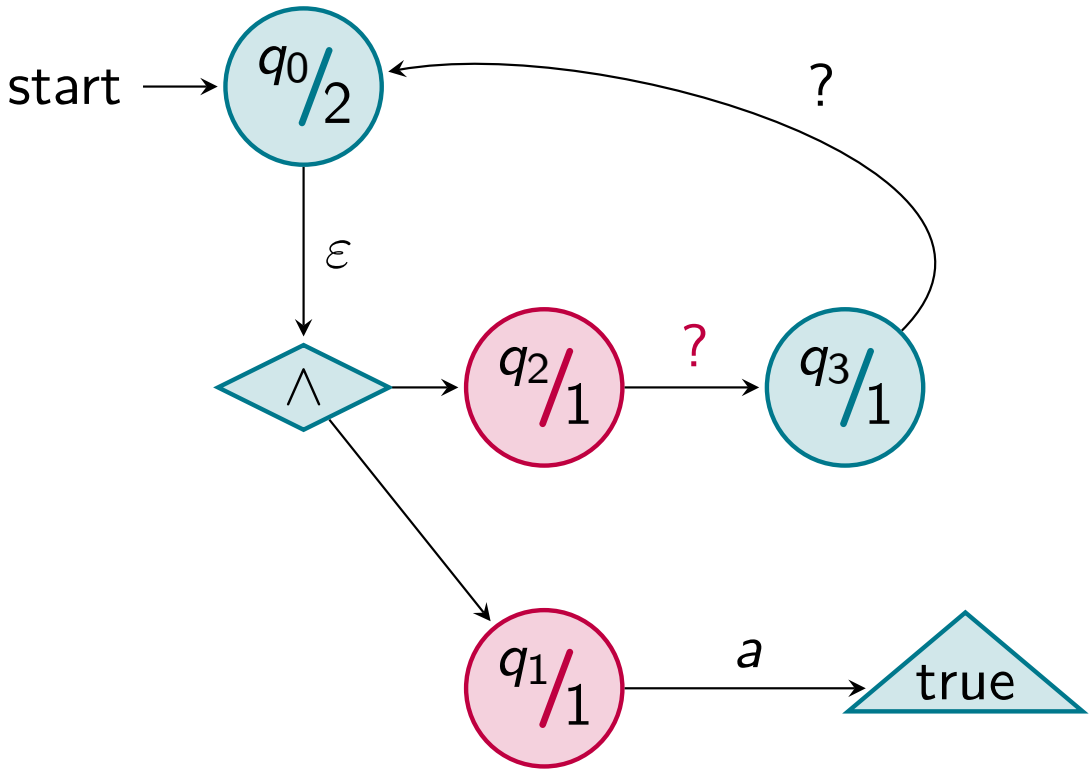
Beispiel eines APWs



Beispiel

a b a a b . . .

Beispiel eines APWs



Beispiel

a b a a b . . .

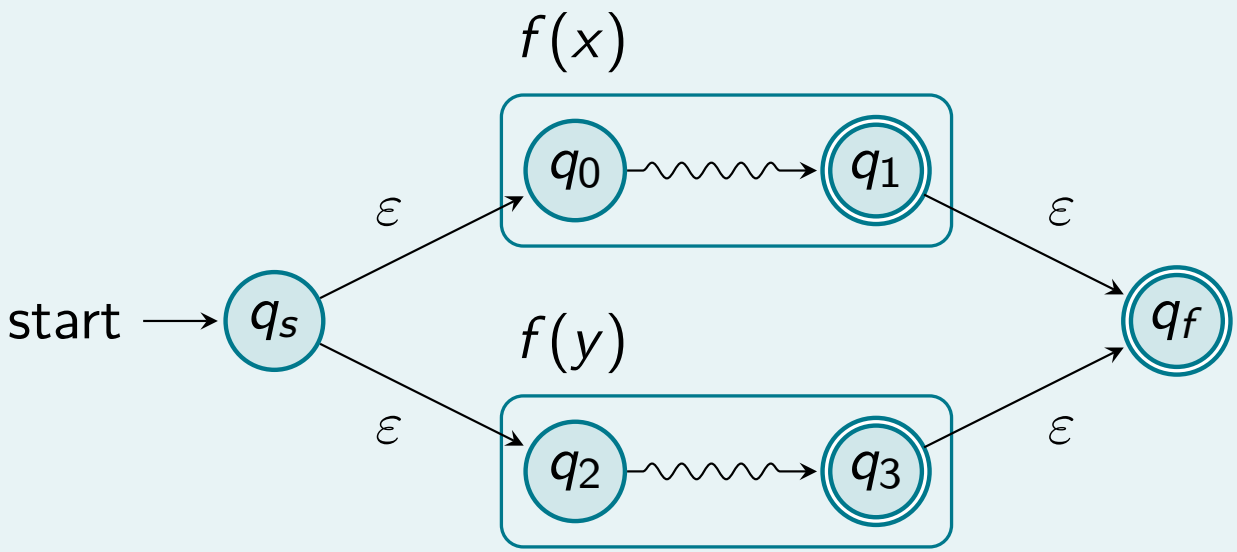
Idee der Umwandlung

1. Reguläre Ausdrücke werden zu NFAs.
2. ε-Transitionen werden entfernt.
3. RLTL-Formeln werden zu APWs.
   (NFAs werden in APWs eingebaut.)
4. ε-Transitionen werden entfernt.

Übersetzung regulärer Ausdrücke

f übersetzt reguläre Ausdrücke mit der Thompson-Konstruktion in NFAs.

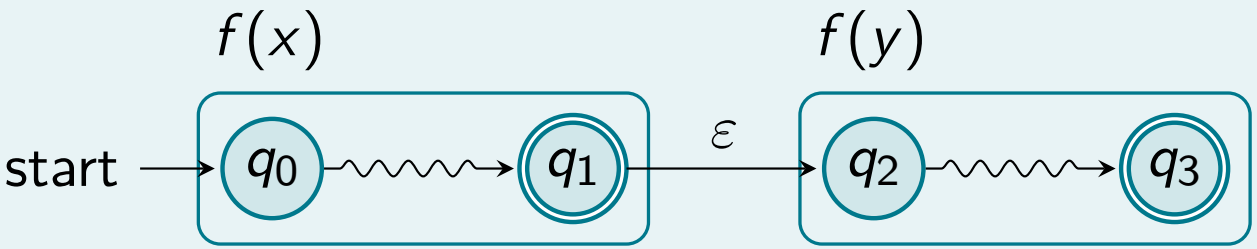
Disjunktion $f(x|y)$



Übersetzung regulärer Ausdrücke

f übersetzt reguläre Ausdrücke mit der Thompson-Konstruktion in NFAs.

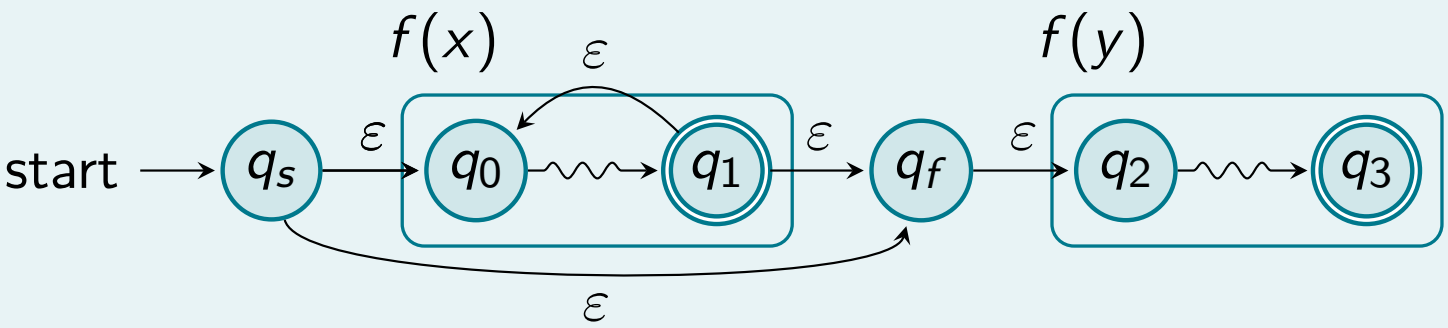
Konkatenation $f(xy)$



Übersetzung regulärer Ausdrücke

f übersetzt reguläre Ausdrücke mit der Thompson-Konstruktion in NFAs.

Kleene-Operator $f(x * y)$



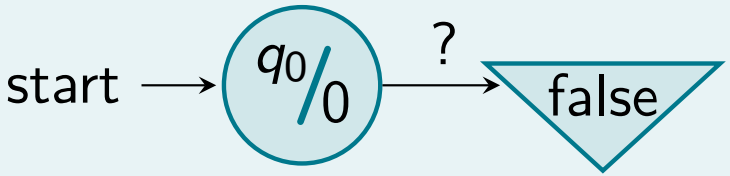
Entfernen der ε-Transitionen

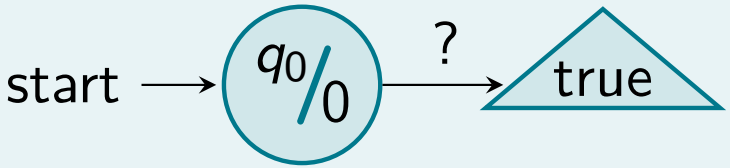
1. Tabelle aller ε-Transitionen erstellen.
2. Hülle der Tabelle bilden.
3. Akzeptanzbedingungen erhalten durch neue Zustände.
4. Alle ε-Transitionen entfernen
   und Tabelle auf Transitionsziele anwenden.
5. Tabelle auf Q_0 anwenden.

Übersetzung regulärer Ausdrücke

g übersetzt RLTL-Ausdrücke mit der Sánchez-Konstruktion in APW-Paare.

leere Sprache $g(\emptyset)$





Übersetzung regulärer Ausdrücke

g übersetzt RLTL-Ausdrücke mit der Sánchez-Konstruktion in APW-Paare.

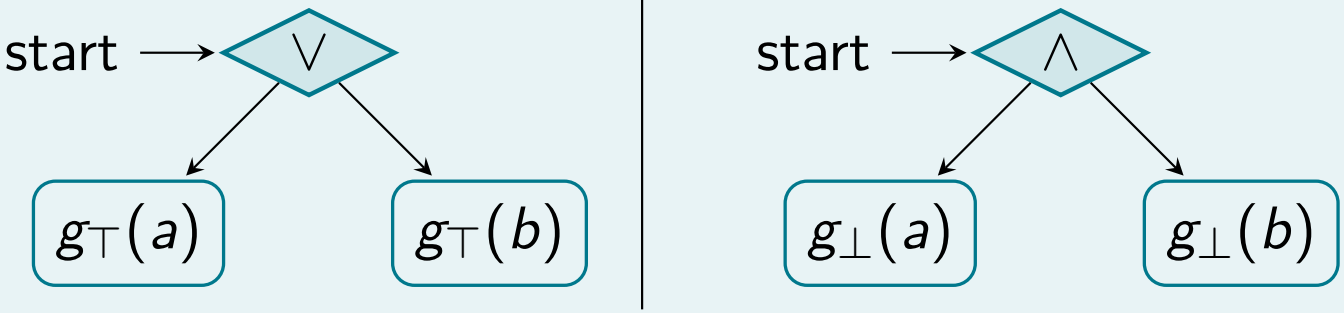
Komplement $g(\neg a)$

$g_\bot(a)$ | $g_\top(a)$

Übersetzung regulärer Ausdrücke

g übersetzt RLTL-Ausdrücke mit der Sánchez-Konstruktion in APW-Paare.

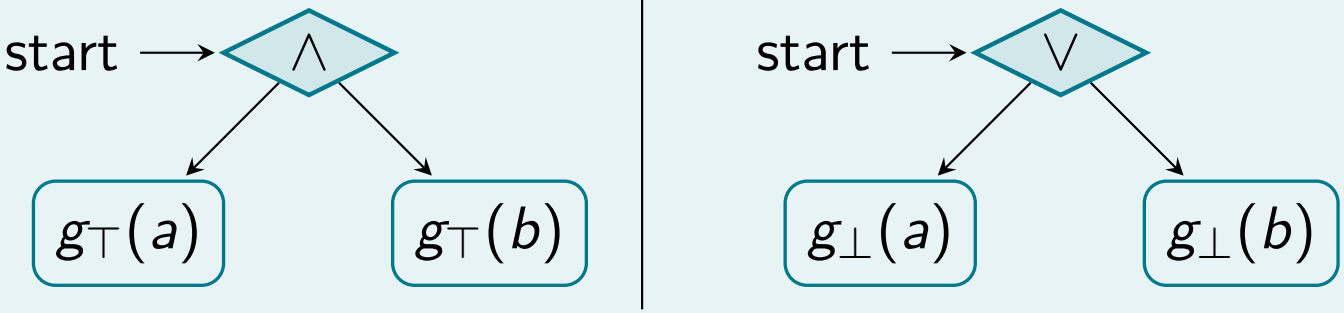
Disjunktion $g(a \vee b)$



Übersetzung regulärer Ausdrücke

g übersetzt RLTL-Ausdrücke mit der Sánchez-Konstruktion in APW-Paare.

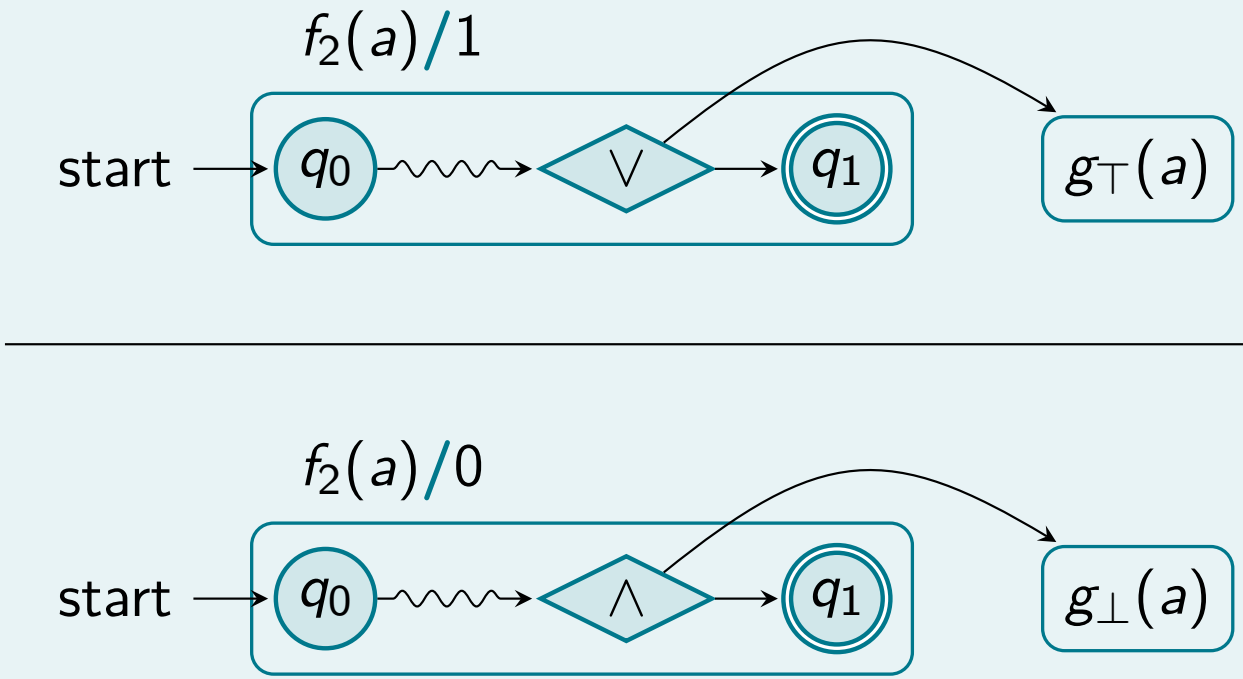
Konjunktion $g(a \wedge b)$



Übersetzung regulärer Ausdrücke

g übersetzt RLTL-Ausdrücke mit der Sánchez-Konstruktion in APW-Paare.

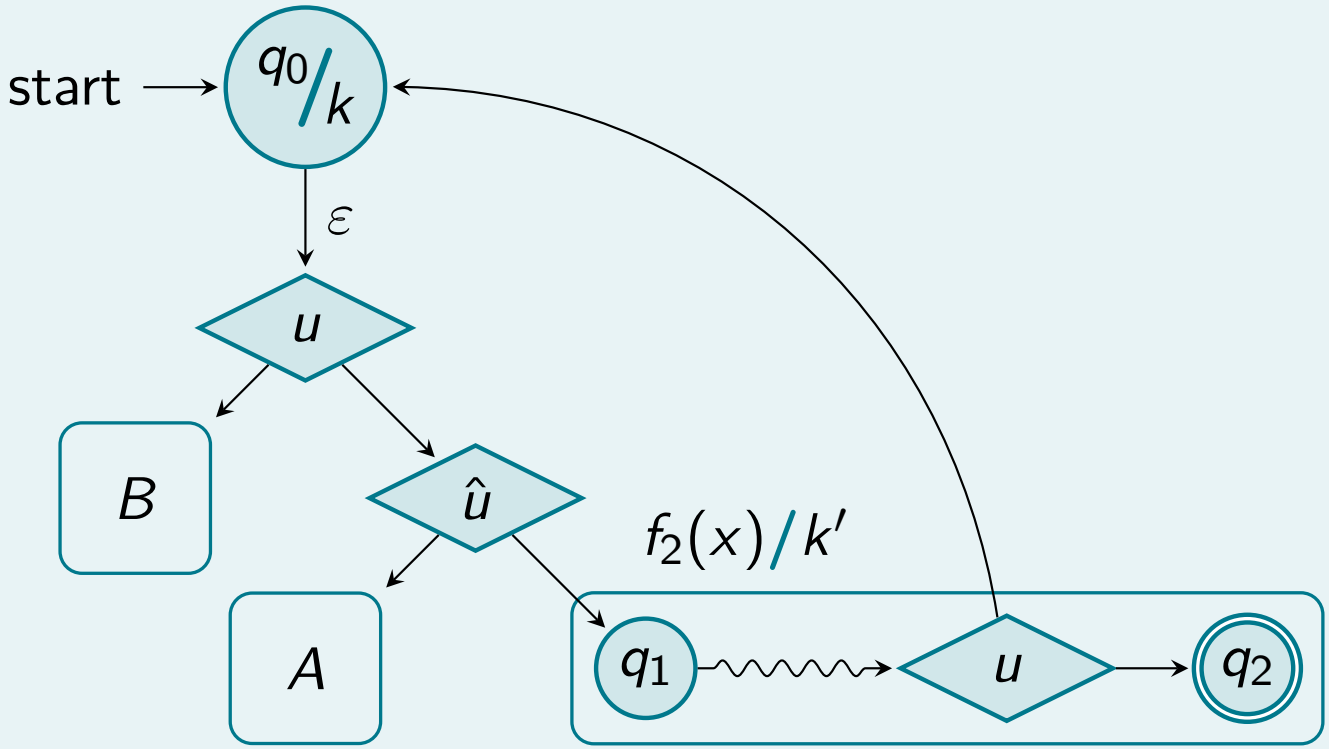
Konkatenation $g(x; a)$



Übersetzung regulärer Ausdrücke

g übersetzt RLTL-Ausdrücke mit der Sánchez-Konstruktion in APW-Paare.

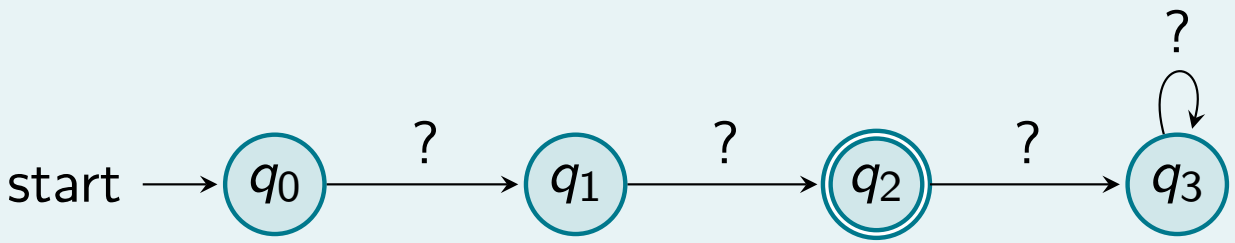
Power $g(a/x\rangle\rangle b)$ und schwache Power $g(a/x\rangle b)$



Beispiel

Anwendung von g (unter Verwendung von f)
auf $a; \neg\emptyset / \mathrm{true}\,\mathrm{true}\rangle\emptyset$

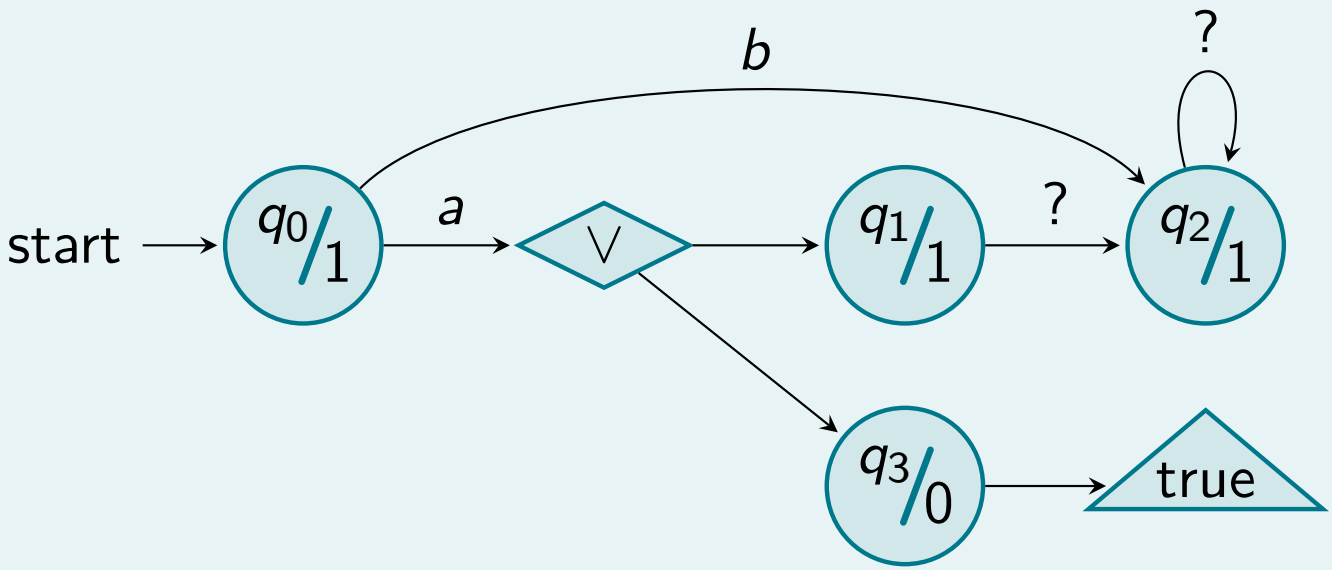
1. $f(\mathrm{true}\,\mathrm{true})$ als totaler Automat



Beispiel

Anwendung von g (unter Verwendung von f)
auf $a; \neg\emptyset / \mathrm{true}\,\mathrm{true} \rangle \emptyset$

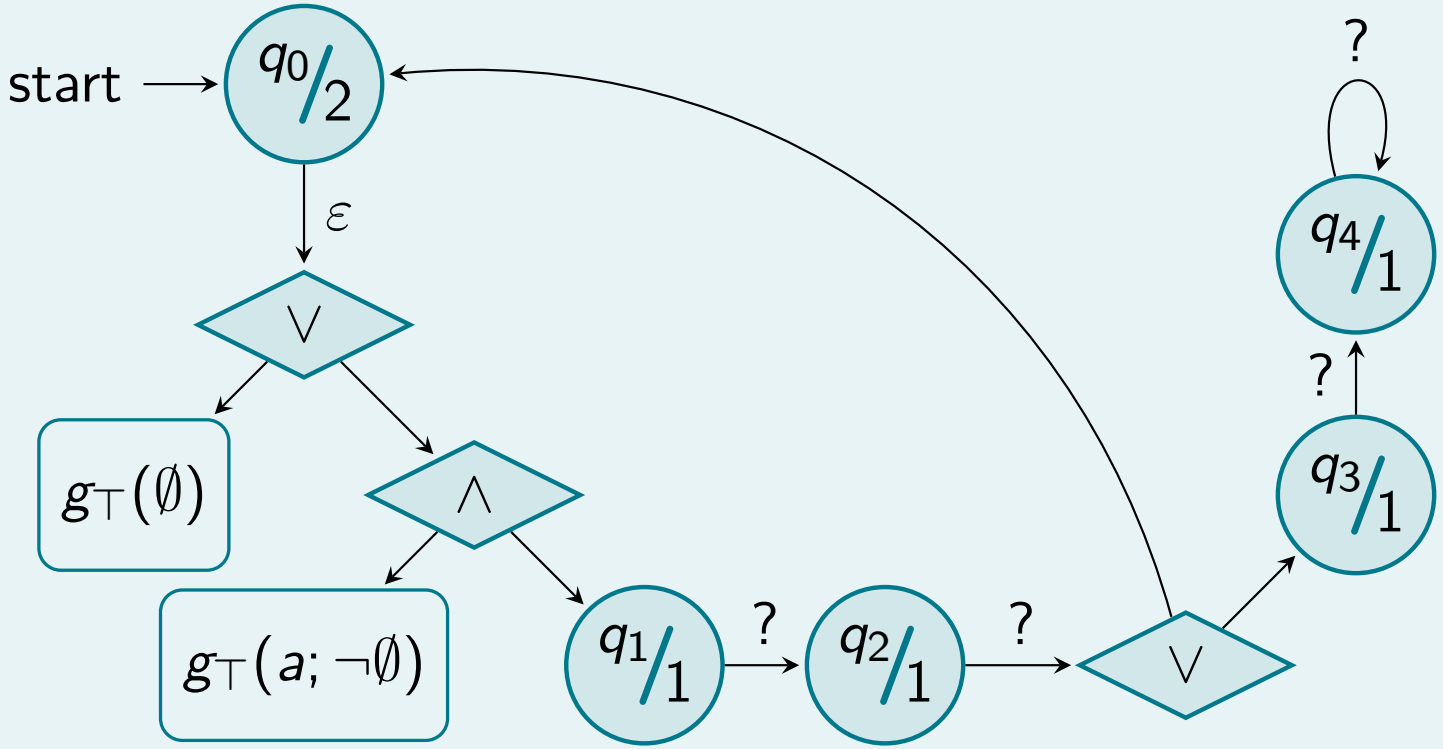
2. $g_\top(a; \neg\emptyset)$



Beispiel

Anwendung von g (unter Verwendung von f)
auf $a; \neg\emptyset / \operatorname{true}\operatorname{true}\rangle\emptyset$

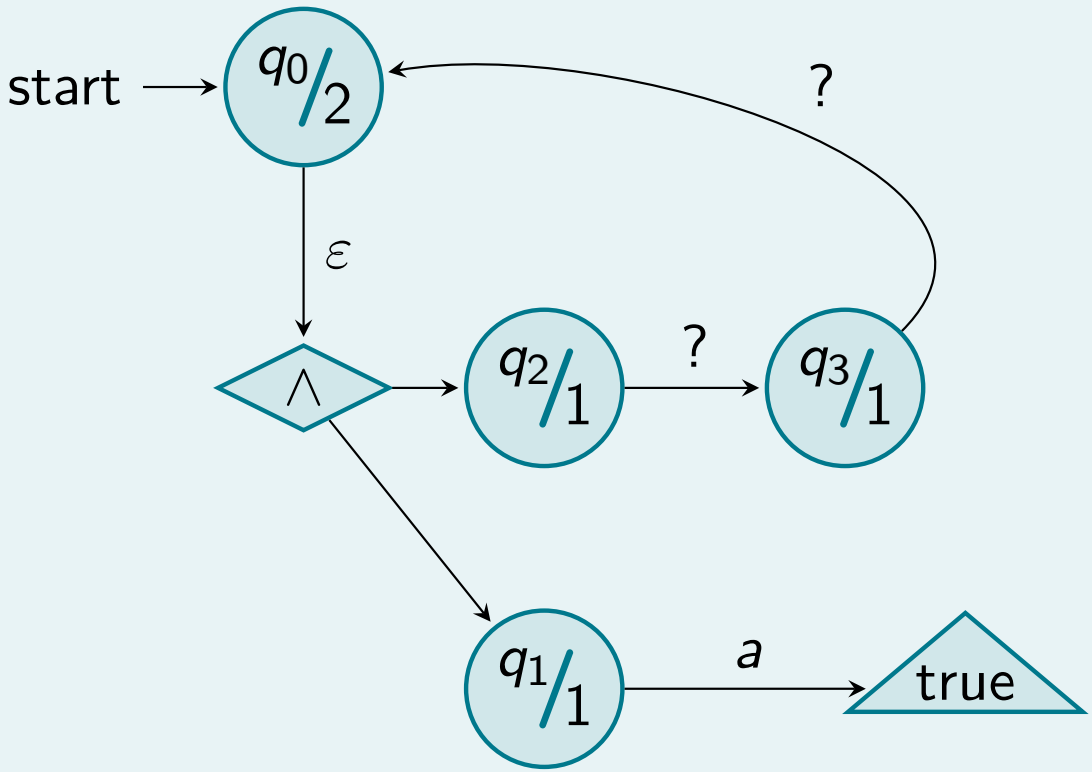
3. $g_\top(a; \neg\emptyset / \operatorname{true}\operatorname{true}\rangle\emptyset)$



Beispiel

Anwendung von g (unter Verwendung von f)
auf a; $\neg\emptyset / \mathrm{true}\,\mathrm{true}\rangle\emptyset$

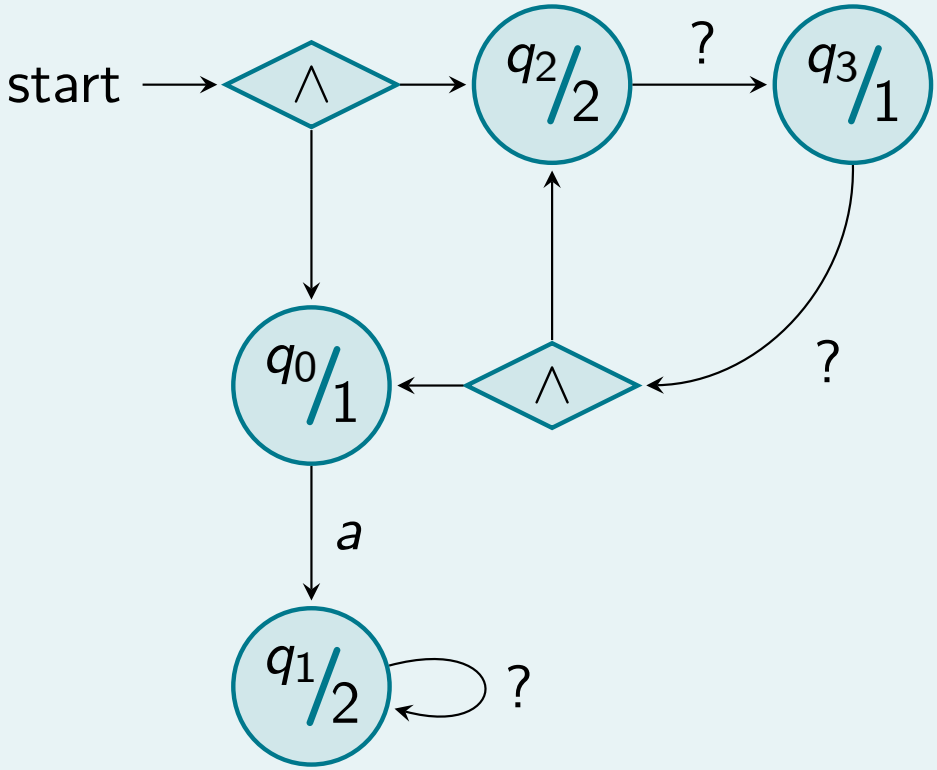
4. leichte Optimierung



Beispiel

Anwendung von g (unter Verwendung von f)
auf $a; \neg\emptyset / \mathrm{true}\,\mathrm{true} \rangle \emptyset$

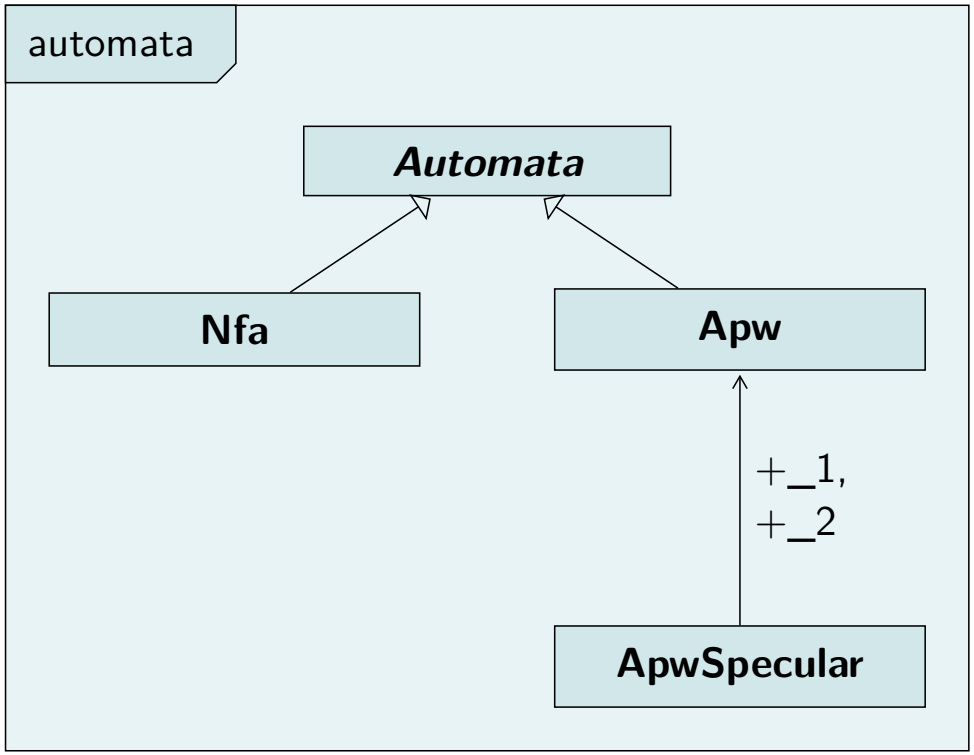
5. starke Optimierung



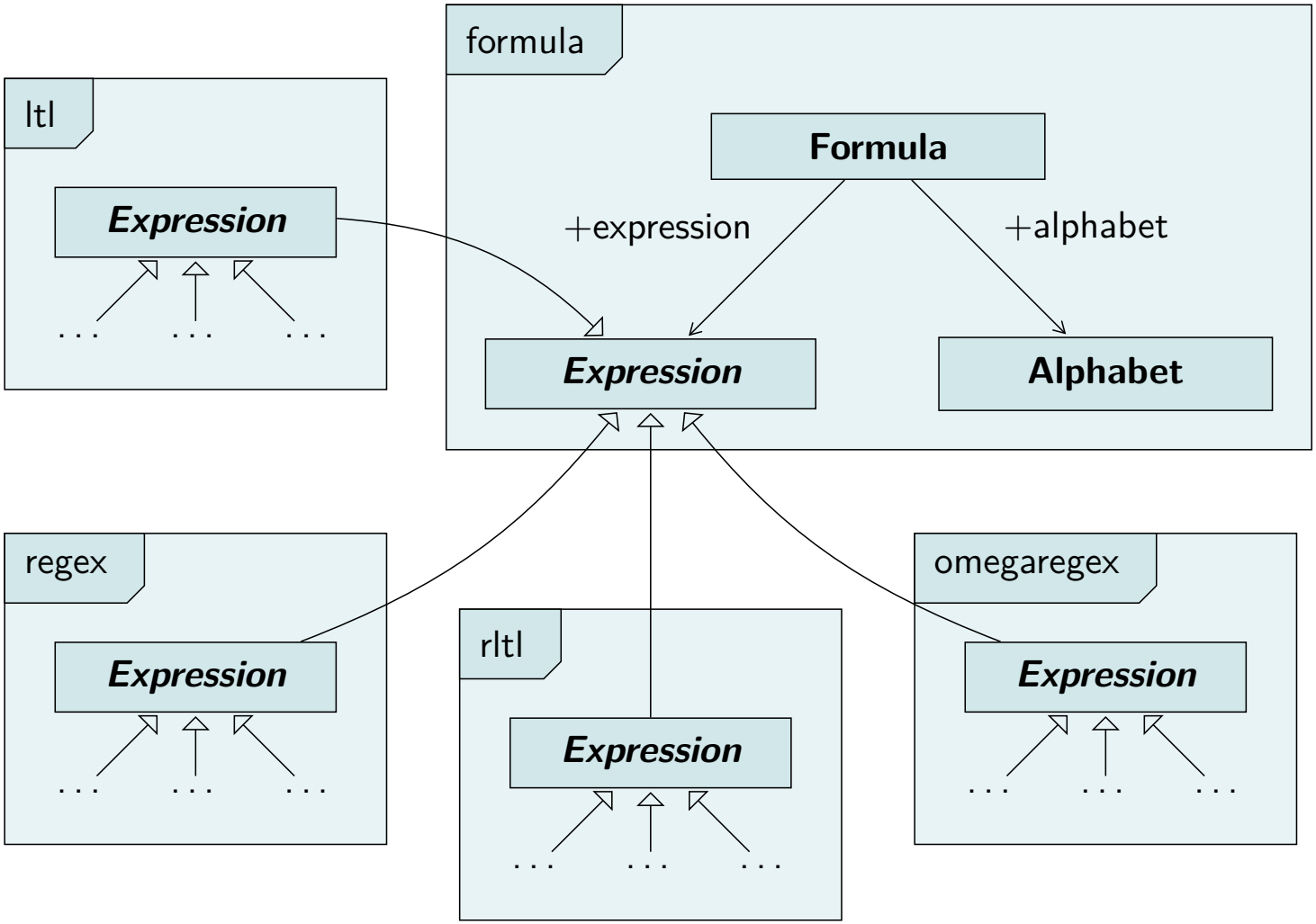
Implementierung in Scala

- Scala compiliert für JVM.
- Scala verbindet OOP und funktionale Programmierung.
  - Pattern Matching
  - map und reduce
  - Klassen, Objekte und Singletons
- Scala kennt Mengen, Listen und Tupel.
- Scala bietet Unterstützung für Parser-Kombinatoren.

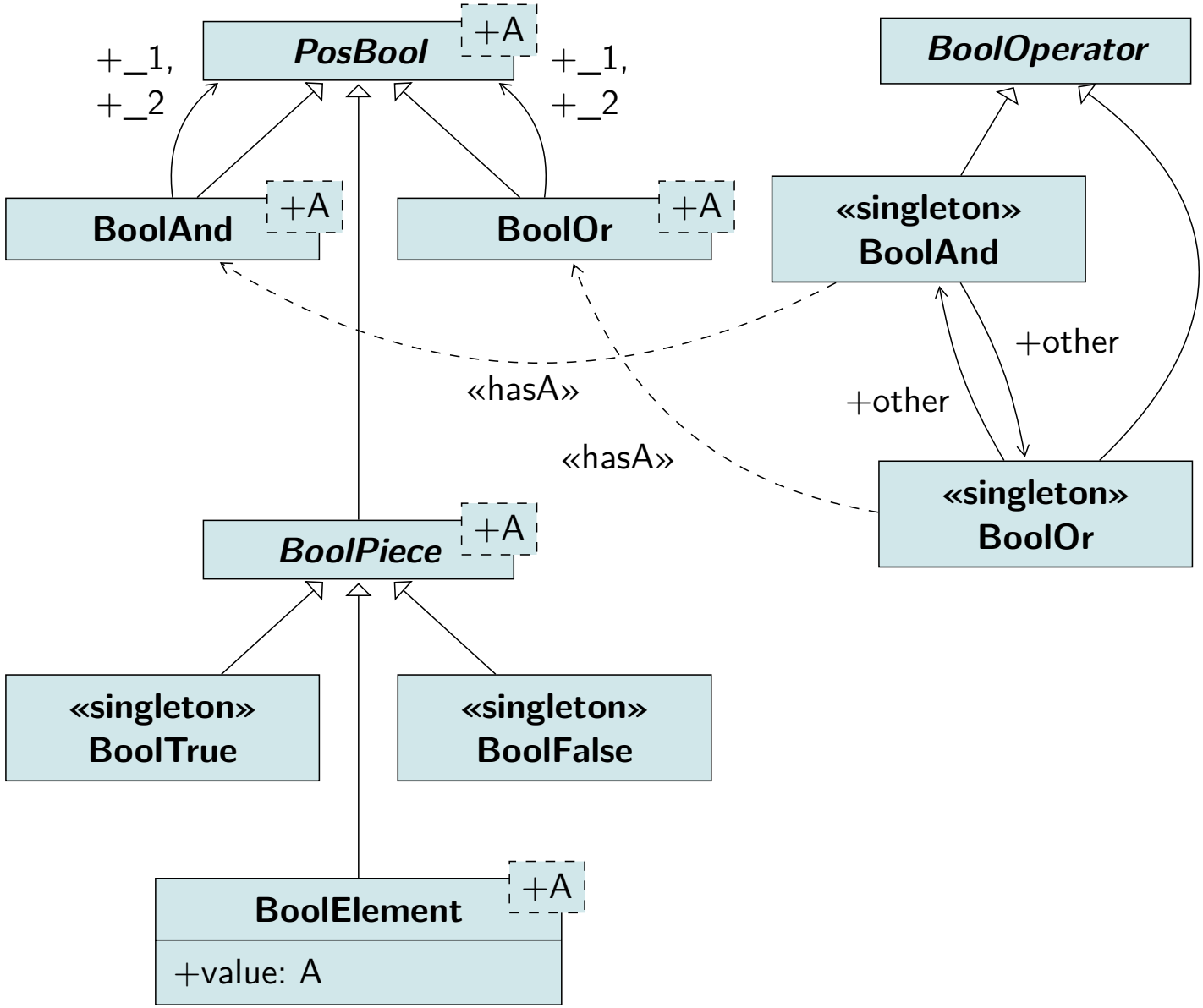
Automaten



Formeln



positive boolsche Formeln



Format der RLTL-Formeln

RLTL-Formel

$$a; \neg\emptyset / \operatorname{true}\operatorname{true}\rangle\emptyset$$

über Alphabet

$$\Sigma = \{a, b\}$$

wird zu

```
RLTL = a ; !% / TRUE TRUE > %,
ALPHABET = [a, b]
```

oder

```
RLTL = "a"; NOT EMPTY / TRUE TRUE > EMPTY,
ALPHABET = ["a", "b"]
```

Austauschformat von APWs

Beispielautomat $g(a; \neg\emptyset / \text{true}\,\text{true}\rangle\emptyset)$
im eigenen Austauschformat:

```
APW {
   ALPHABET = ["a", "b", EPSILON]
   STATES = [q0:2, q1:1, q2:1, q3:1]
   START = q0
   DELTA(q3, ?) = q0
   DELTA(q2, ?) = q3
   DELTA(q1, "a") = TRUE
   DELTA(q0, EPSILON) = q1 AND q2
}
```

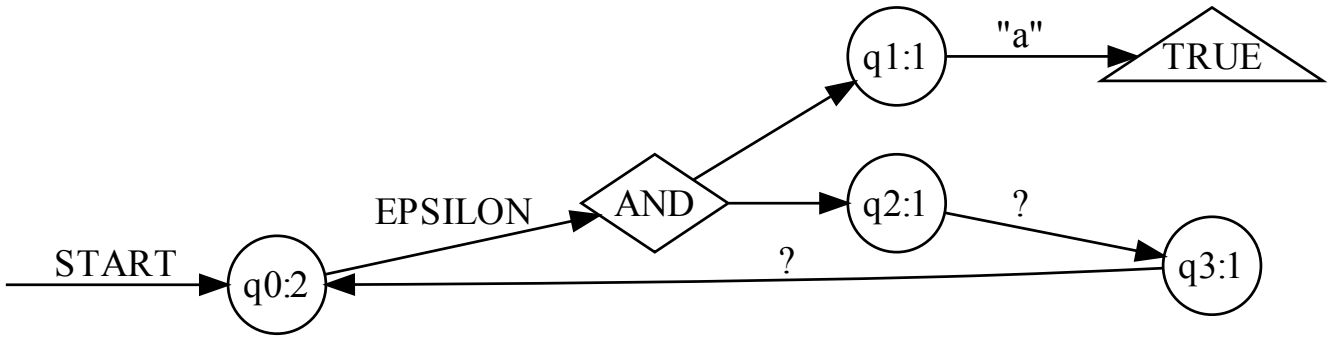
Dot-Format von APWs

Beispielautomat $g(a; \neg\emptyset / \text{true}\,\text{true}\rangle\emptyset)$
im Dot-Format:

```
digraph apw {
  rankdir=LR;
  q0 [shape=circle, label="q0:2", margin=0];
  q1 [shape=circle, label="q1:1", margin=0];
  q2 [shape=circle, label="q2:1", margin=0];
  q3 [shape=circle, label="q3:1", margin=0];
  start [shape=none, label=""];
  start -> q0 [label="START"];
  and1 [shape=diamond, label="AND", margin=0];
  and1 -> q1;
  and1 -> q2;
  q0 -> and1 [label="EPSILON"];
  q3 -> q0 [label="?"];
  q2 -> q3 [label="?"];
  true2 [shape=triangle, label="TRUE", margin=0];
  q1 -> true2 [label="\"a\""];
}
```

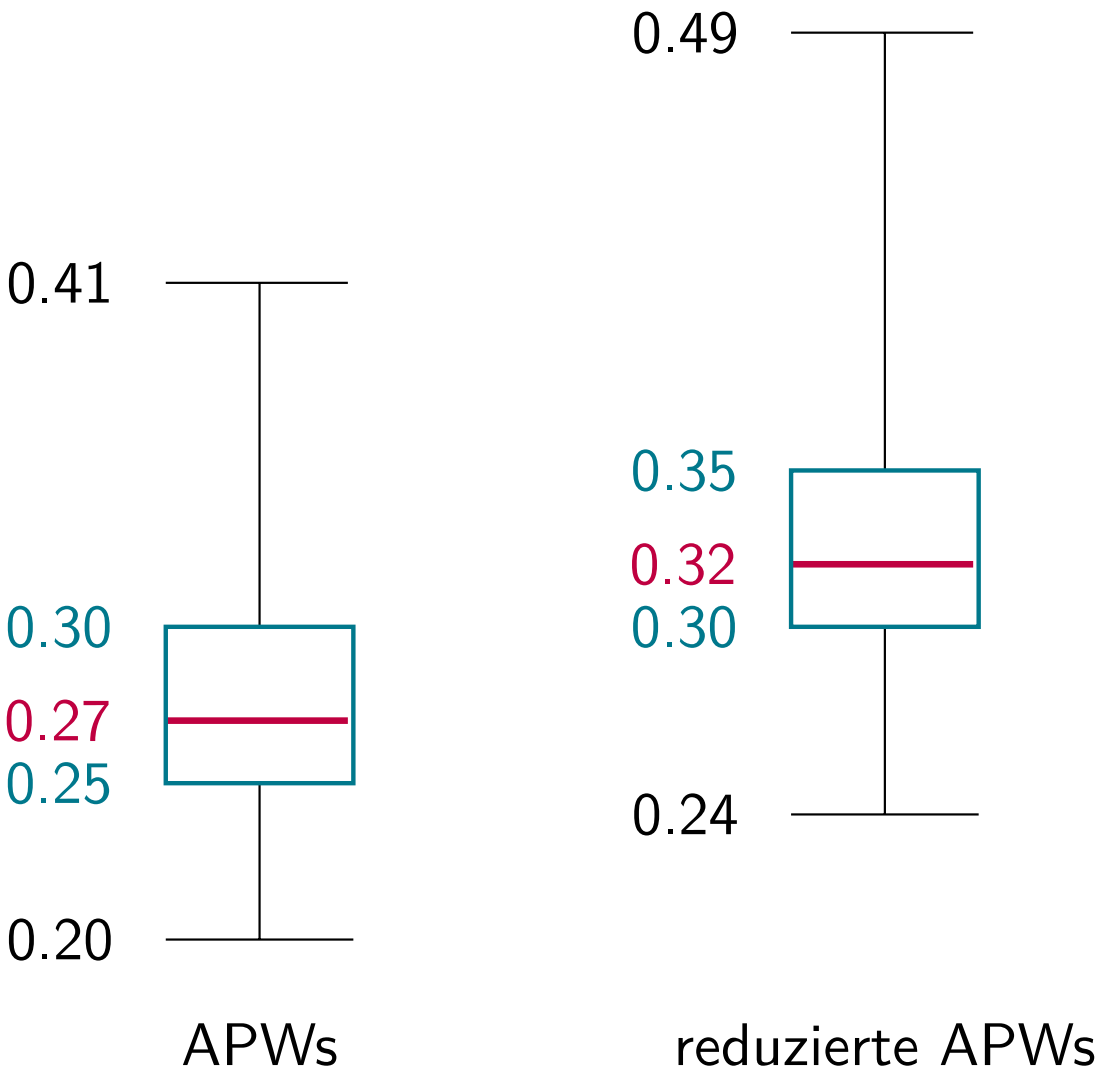
Dot-Format von APWs

Beispielautomat $g(a; \neg\emptyset / \text{true}\,\text{true}\rangle\emptyset)$
im Dot-Format:



Messung der Laufzeit

Test mit 97 LTL-Pattern aus der Praxis



(Plus ein Ausreißer mit 1.20 s und 1.27 s.)

Zusammenfassung

1. RLTL erweitert LTL um Konjunktion und Negation und verwendet reguläre Ausdrücke als Verzögerung.
2. Kombination mit der Arbeit von Torben Scheffel liefert Umwandlung von RLTL zur Büchiautomaten. RLTL mit Vergangenheit liefert 2APWs. Umwandlung von 2APWs zu APWs fehlt noch.
3. Umwandlung von RLTL in (2)APWs kann als CLI oder Java-Bibliothek verwendet werden. Die Laufzeit kann vernachlässigt werden. Die Umwandlung von APWs in Büchiautomaten dauert deutlich länger.
4. Alle Zwischenschritte der Umwandlung können einzeln über das CLI verwendet werden. Umwandlung von LTL und ω-regulären Ausdrücken in RLTL ist möglich.

Syntax regulärer Ausdrücke in EBNF

Definition (Syntax regulärer Ausdrücke in EBNF)

```
RE = T { "|" T }          // Disjunktion
 T = K { K }              // Konkatenation
 K = E "*" E | E          // Kleene-Operator
 E = F "-1" | "_" P | F   // Vergangenheit
 F = P | "(" RE ")"       // Klammerung
```

- Die Semantik von Disjunktion und Konkatenation sei „wie immer“.
- Binärer Kleene-Operator verhindert die Akzeptanz des leeren Wortes,
- denn das leere Wort ist als Verzögerung nicht geeignet.

reguläre Ausdrücke

Definition (Segment)

- Ein Segment ist ein Tupel (w, i, j) aus
  - einem unendlichen Wort w,
  - der Position i des ersten Zeichens
  - und der Position j des letzten Zeichens.

- Disjunktion, Konkatenation und Kleene-Operation „wie immer“.
- Formale Semantik: Relation $\models_{\mathrm{RE}}$ zwischen Segmenten und regulären Ausdrücken.

positive boolsche Formel

Transitionen bilden auf $\mathcal{B}^+(Q)$ statt auf Q ab.

Definition (positive boolsche Formel)

Für Formeln $\mathcal{B}^+(Q)$ von einer Menge Q gilt

- $\{\text{true}, \text{false}\} \subset \mathcal{B}^+(Q)$,
- $Q \subset \mathcal{B}^+(Q)$
- und $\forall q, r \in \mathcal{B}^+(Q) : q \vee r \in \mathcal{B}^+(Q), q \wedge r \in \mathcal{B}^+(Q)$.

- $M \subseteq Q$ ist Modell einer Formel, wenn $\beta(x) = 1$ für alle $x \in M$ die Formel erfüllt.
- M ist minimales Modell einer Formel, wenn kein Modell $M' \subsetneq M$ der Formel existiert.

Vergangenheit regulärer Ausdrücke

Es gilt $(w, i, j) \models_{\mathrm{RE}} -p$, genau dann wenn $w[i]$ den Basisausdruck p erfüllt und $j = i - 1$.

Der Operator $\cdot^{-1}$ wird induktiv über folgende Äquivalenzen definiert. Als Induktionsanfang gilt für einen Basisausdruck p

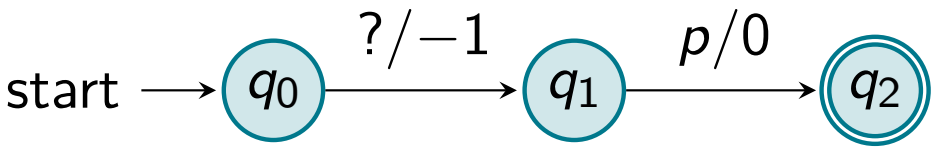
$$p^{-1} :\equiv -p$$
$$(-p)^{-1} :\equiv p$$

und als Induktionsschritt gilt für die regulären Ausdrücke a und b

$$(a \mid b)^{-1} :\equiv a^{-1} \mid b^{-1}$$
$$(ab)^{-1} :\equiv b^{-1}a^{-1}$$
$$(a * b)^{-1} :\equiv b^{-1} \mid b^{-1}a^{-1} * a^{-1}$$

RLTL mit Vergangenheit

- Verwendung von regulären Ausdrücken mit Vergangenheit liefert RLTL mit Vergangenheit.
- RLTL hat keinen eigenen Vergangenheits-Operator.
- Allgemeine Vergangenheit wird über Vergangenheit auf Basis-Ausdrücken erreicht.
- Umwandlung von regulären Ausdrücken mit Vergangenheit liefert 2NFAs, sodass schließlich 2APWs entstehen.
- Umwandlung von $-p$ für einen Basisausdruck p liefert diesen Automaten:



Implementierung in Scala

Beispiel (Ein NFA in Scala)

```
Tuple5[Set[Sign],  // Eingabealphabet
  List[State],     // Zustaende
  List[State],     // Startzustaende
  Map[Pair[State, Sign], List[State]],
                   // Transitionsfunktion
  List[State]]     // akzeptierende Zustaende
```

Kommandozeilenanwendung (CLI)

Umwandlungen werden der Reihe nach auf Eingabe angewendet

`--future` Allgemeine Vergangenheits-Operatoren in regulären Ausdrücken entfernen.

`--positive` Negationsoperatoren in RLTL entfernen.

`--dot` Automat in Dot-Format exportieren.

`--slim` APW optimieren.

`--reduce` APW oder NFA stark optimieren. (z. B. ε-Transitionen entfernen.)

`--complete` NFA zum totalen Automaten erweitern.

`--nfa` Regulären Ausdruck in NFA umwandeln.

`--apw` RLTL-Ausdruck in APW umwandeln.

`--!apw` Negierten RLTL-Ausdruck in APW umwandeln.

`--apws` RLTL-Ausdruck in zwei APWs umwandeln.

`--rltl` LTL oder ω-regulären Ausdruck in RLTL umwandeln.

`--profile` Zeitmessung starten.