Optimizing Schema Languages for XML: Numerical Constraints and Interleaving

Wouter Gelade[1], Wim Martens[2], and Frank Neven[1]

[1]Hasselt University
[2]University of Dortmund

Januari 11, 2007

XML

XML documents

XML documents are abstracted by trees.

Schema

A schema defines a tree language.

Advantages

- automatic validation
- automatic integration of data
- query optimization
- provides a user with a concrete semantics of the document
- aids in the specification of meaningful queries over XML data

Problems involving schema's

Minimization

- Given a schema, create an equivalent smaller schema.
- Equivalence problem: Given two schema's, do they define the same set of XML documents?

Problems involving schema's

Minimization

- Given a schema, create an equivalent smaller schema.
- Equivalence problem: Given two schema's, do they define the same set of XML documents?

Data integration

- Integration of XML data of different sources.
- Can be done on *schema level*.
- Inclusion problem: Given two schema's s, s', are all documents defined by s also defined by s'?
- Intersection (non-emptiness) problem: Given schema's $s_1, \ldots, s_n$, does there exist a document defined by all schema's?

Basic decision problems for XML schema languages

Building blocks

Equivalence, inclusion, and intersection problem are the building blocks of optimization problems.

Question

What is the complexity of the basic decision problems for XML schema languages?

Previous results

Reduce complexity to the complexity of the basic decision problems for regular expressions and tree automata.

Schema's and regular expressions

Schema languages

- DTD, XML Schema, and Relax NG
- Abstracted by grammars with regular expressions as right-hand sides.

Standard regular expressions

- DTDs only allow standard regular expressions using concatenation ($\cdot$), disjunction ($+$), Kleene-star ($*$), and question-mark ($?$).
- Standard regular expressions only form a subset of the actual regular expressions used in other XML schema languages.

Numerical occurrence operator

Definition

- If r is a RE(#), then $r^{[i,j]}$, with $i \leq j$ ($i, j \in \mathbb{N}$), also is a RE(#).
- Example: $a^{[3..5]}b + (cd)^{[0,10]}$
- Additional operator in XML Schema (minoccurs and maxoccurs).

Properties

- Defines only regular languages. Every numerical occurrence predicate can be *unfolded*.
- Example: $a^{[2,5]} = aaa?a?a?$.
- Is exponentially succinct with regard to standard regular expressions because integers are encoded in binary.
- Example $a^{[0..2^n]} = \underbrace{a? \cdots a?}_{2^n}$

Interleaving or shuffle operator

Definition

- For words w, u, v, and symbols a, b:
- $w\&\varepsilon = \varepsilon\&w = w$, and
- $au\&bv = (a(u\&bv)) \cup (b(au\&v))$
- Allows the words of its operands to be *shuffled*.
- Example: $r = ab\&cd$
- $abcd, cdab, acbd \in L(r)$, $bacd \notin L(r)$
- $L(r\&s) = \{w | u \in L(r), v \in L(s), w \in L(u\&v)\}$
- Additional operator in Relax NG.

Interleaving or shuffle operator

Properties

- Defines only regular languages. Can be translated into NFA using product construction.
- Is exponentially succinct with regard to standard regular expressions.
- Example: $a_1 \& a_2 \& \cdots \& a_n$.

XML Schema and SGML

- XML Schema has a restricted shuffle operator (ALL).
- SGML shuffle only allows unordered concatenation: $r \& s = rs + sr$.

Goal

Complexity basic decision problems

What is the complexity of the equivalence, inclusion, and intersection problem for XML schema languages using these new operators?

Outline

Document Type Definition (DTD)

DTD abstraction (start symbol store)

$$
\begin{aligned}
\mathit{store} &\rightarrow \mathit{dvd}\ \mathit{dvd}^* \\
\mathit{dvd} &\rightarrow \mathit{title}\ \mathit{price}\ (\mathit{discount} + \varepsilon) \\
\mathit{title} &\rightarrow \mathit{DATA} \\
\mathit{price} &\rightarrow \mathit{DATA} \\
\mathit{discount} &\rightarrow \mathit{DATA}
\end{aligned}
$$

XML Schema and Relax NG

Extended DTDs (EDTDs)

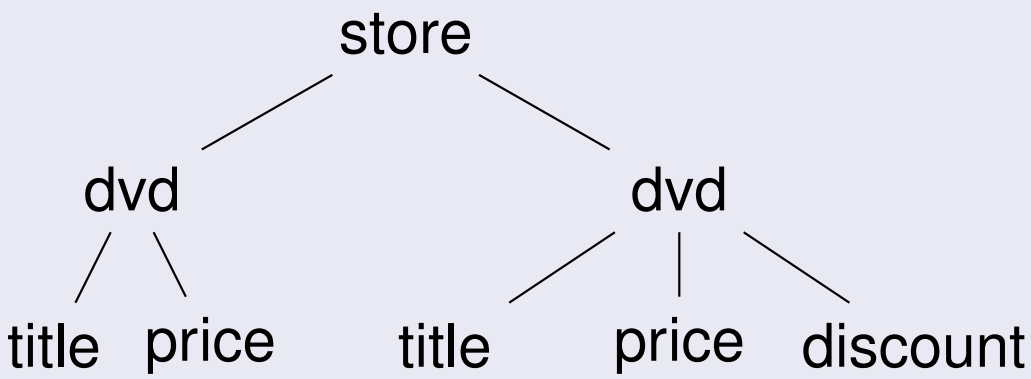
- XML Schema and Relax NG allow the use of types.
- Extend power of DTDs to extended DTDs.
- EDTDs are equivalent to regular unranked tree automata.

Extended DTDs

Example

$$
\begin{aligned}
\mathit{store} &\rightarrow (\mathit{dvd}^1 + \mathit{dvd}^2)^* \, \mathit{dvd}^2 \, (\mathit{dvd}^1 + \mathit{dvd}^2)^* \\
\mathit{dvd}^1 &\rightarrow \mathit{title}\ \mathit{price} \\
\mathit{dvd}^2 &\rightarrow \mathit{title}\ \mathit{price}\ \mathit{discount}
\end{aligned}
$$

Typed tree

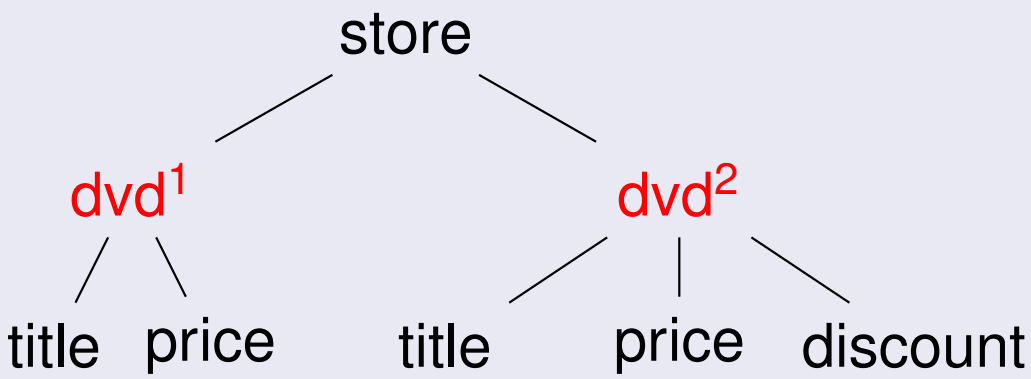


Extended DTDs

Example

$$
\begin{aligned}
\mathit{store} &\rightarrow (\mathit{dvd}^1 + \mathit{dvd}^2)^* \, \mathit{dvd}^2 \, (\mathit{dvd}^1 + \mathit{dvd}^2)^* \\
\mathit{dvd}^1 &\rightarrow \mathit{title}\ \mathit{price} \\
\mathit{dvd}^2 &\rightarrow \mathit{title}\ \mathit{price}\ \mathit{discount}
\end{aligned}
$$

Typed tree



XML Schema

Element Declarations Consistent (EDC) constraint

It is illegal to have two elements of the same name [...] but different types in a content model [...].

Single-type EDTD

A single-type EDTD is an EDTD for which in no regular expression two types b^i and b^j with $i \neq j$ occur.

Example is not single-type

$$
\begin{aligned}
\mathit{store} &\rightarrow (\mathit{dvd}^1 + \mathit{dvd}^2)^* \; \mathit{dvd}^2 \; (\mathit{dvd}^1 + \mathit{dvd}^2)^* \\
\mathit{dvd}^1 &\rightarrow \mathit{title}\ \mathit{price} \\
\mathit{dvd}^2 &\rightarrow \mathit{title}\ \mathit{price}\ \mathit{discount}
\end{aligned}
$$

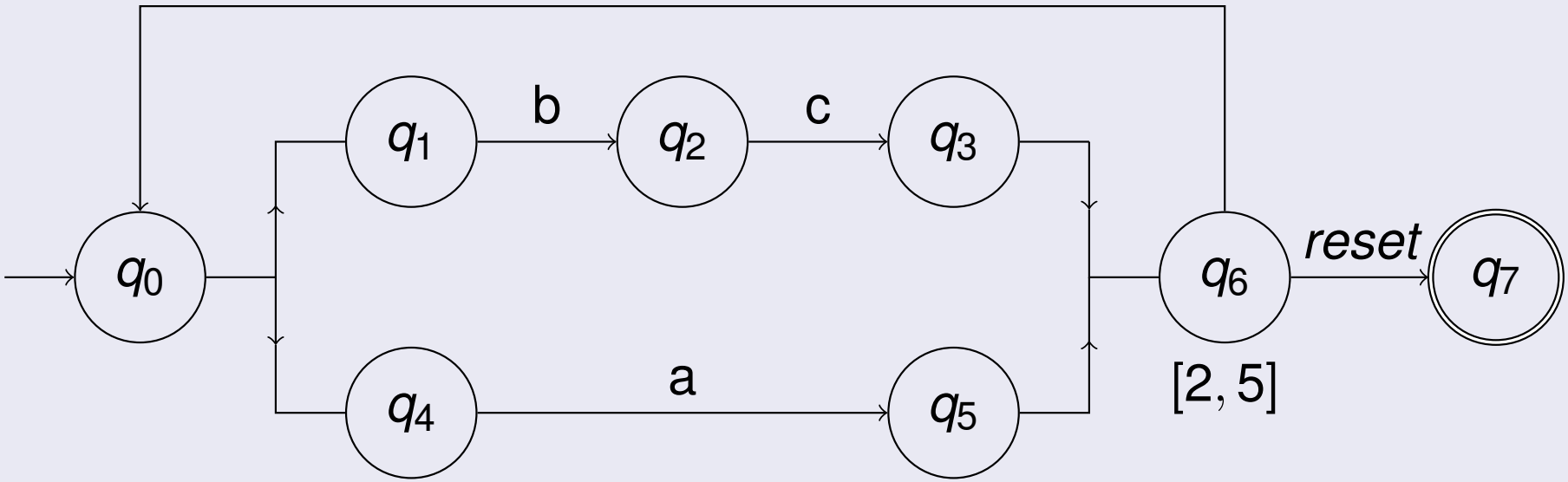
Outline

NFA(#, &) recognizing RE(#, &)

Features

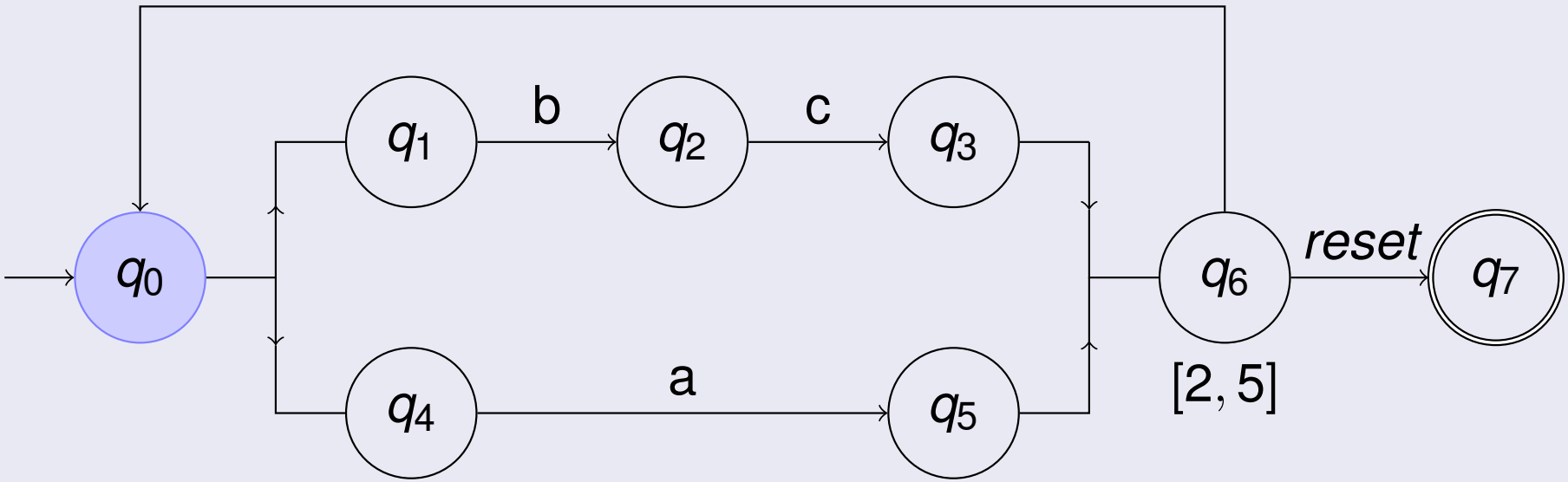
- Add counters to states to handle numerical occurrence operators.
- Allow automaton to be in more than one state at once to handle shuffle operator. (Cfr. Jedrzejowicz, Szepietowski 2001)

NFA(#, &) for $(a\&bc)^{[2,5]}$



$abcbac \in (a\&bc)^{[2,5]}$?

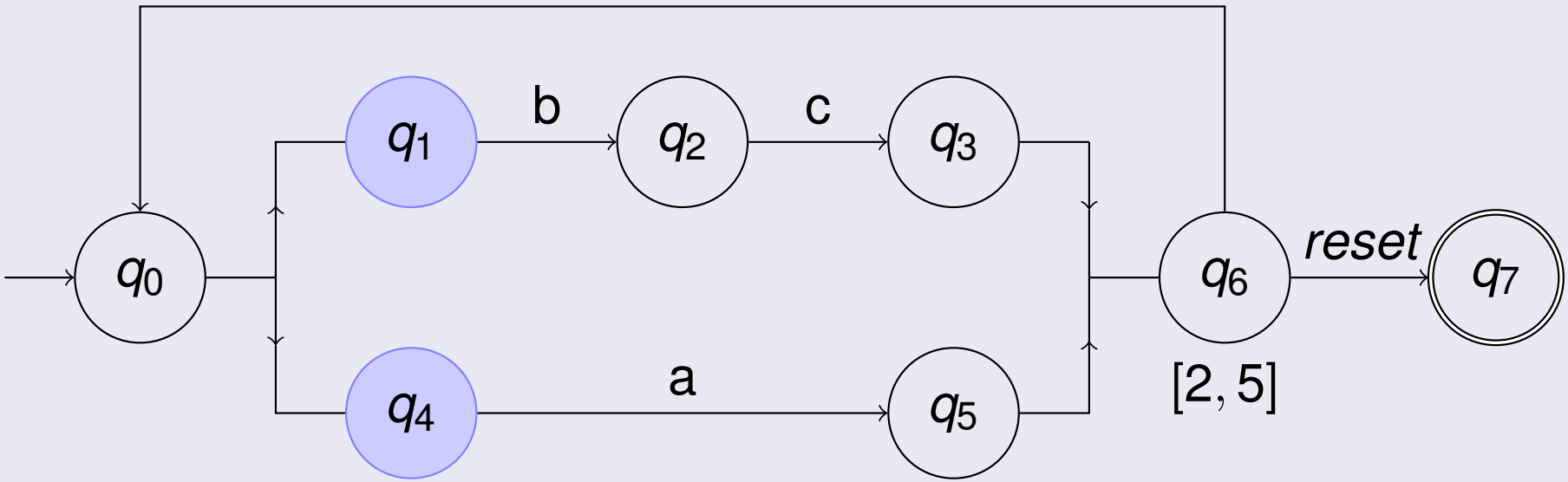
Input: abcbac



Counter $q_6 = 0$

$abcbac \in (a\&bc)^{[2,5]}$?

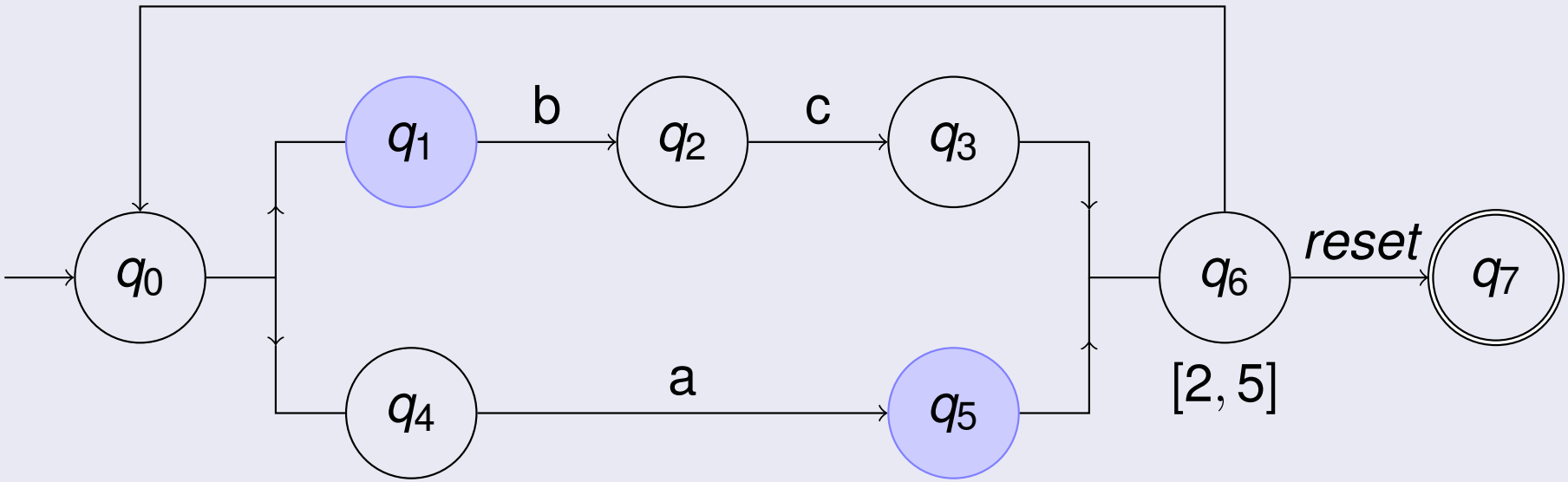
Input: abcbac



Counter $q_6 = 0$

$abcbac \in (a\&bc)^{[2,5]}$?

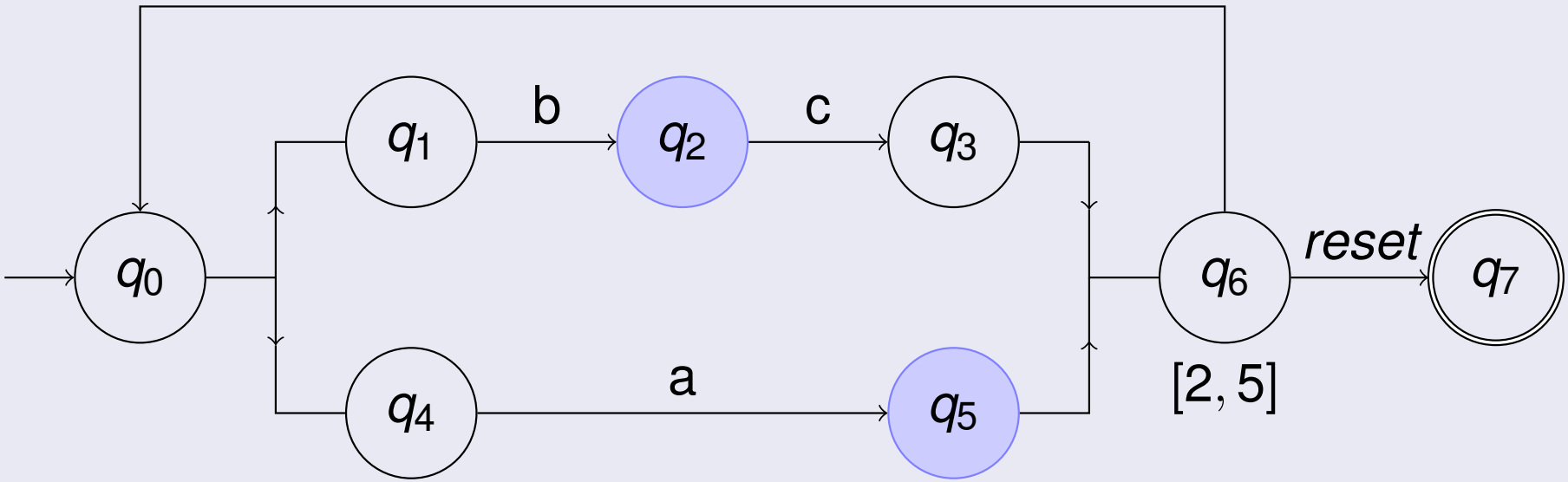
Input: abcbac



Counter $q_6 = 0$

$abcbac \in (a\&bc)^{[2,5]}$?

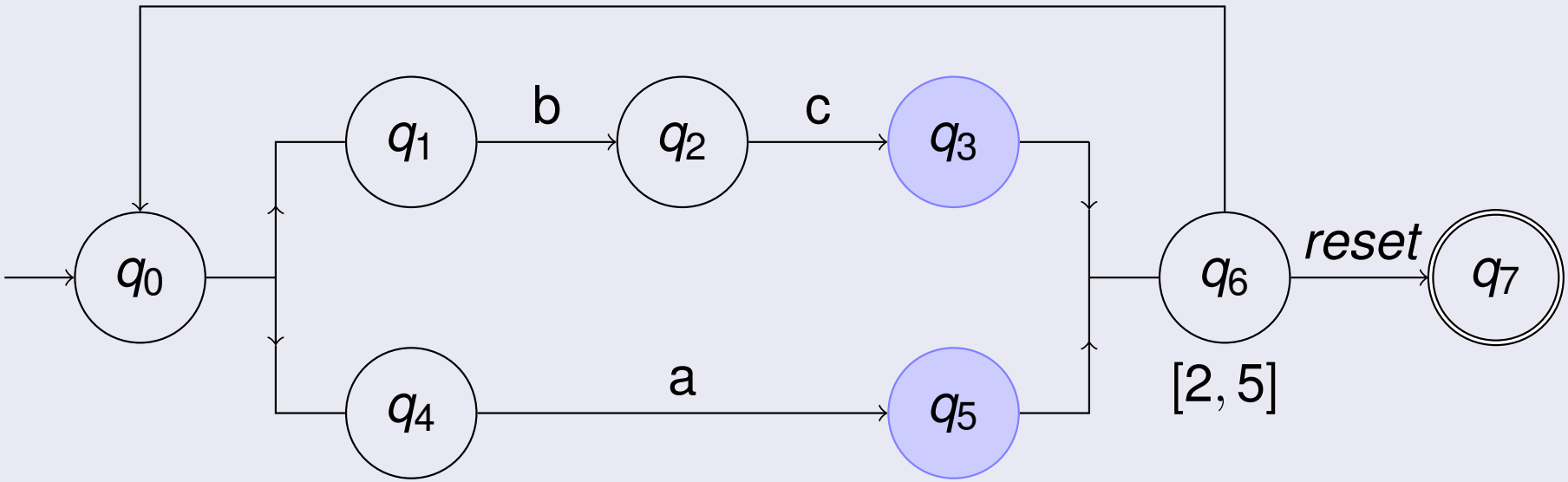
Input: abcbac



Counter $q_6 = 0$

$abcbac \in (a\&bc)^{[2,5]}$?

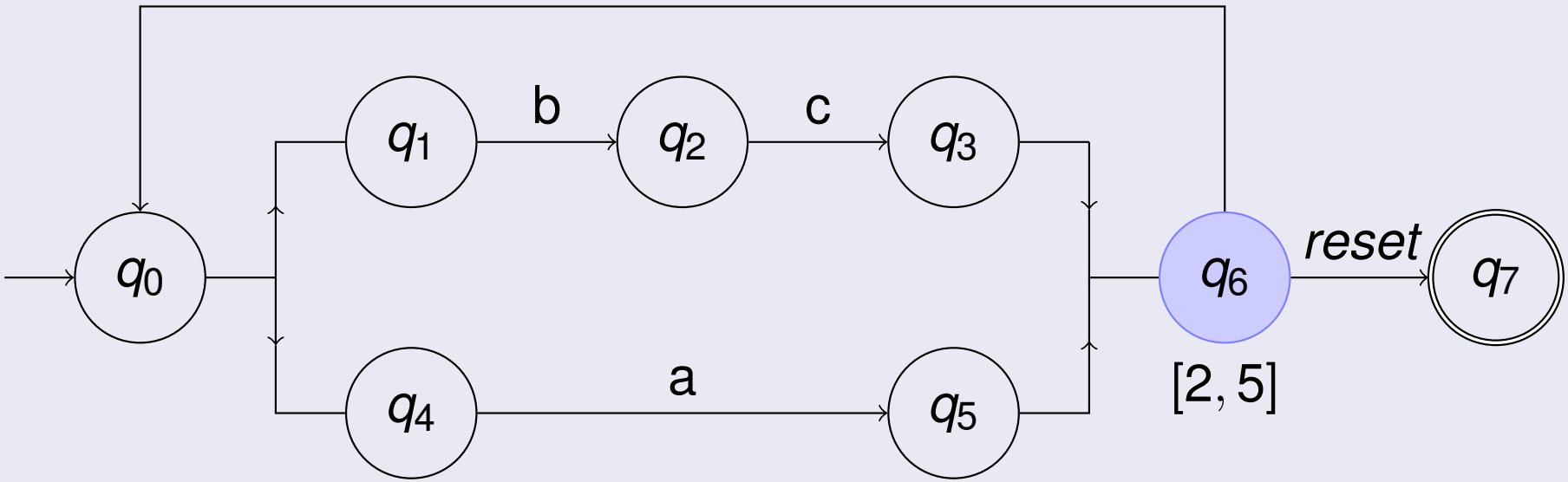
Input: abcbac



Counter $q_6 = 0$

$abcbac \in (a\&bc)^{[2,5]}$?

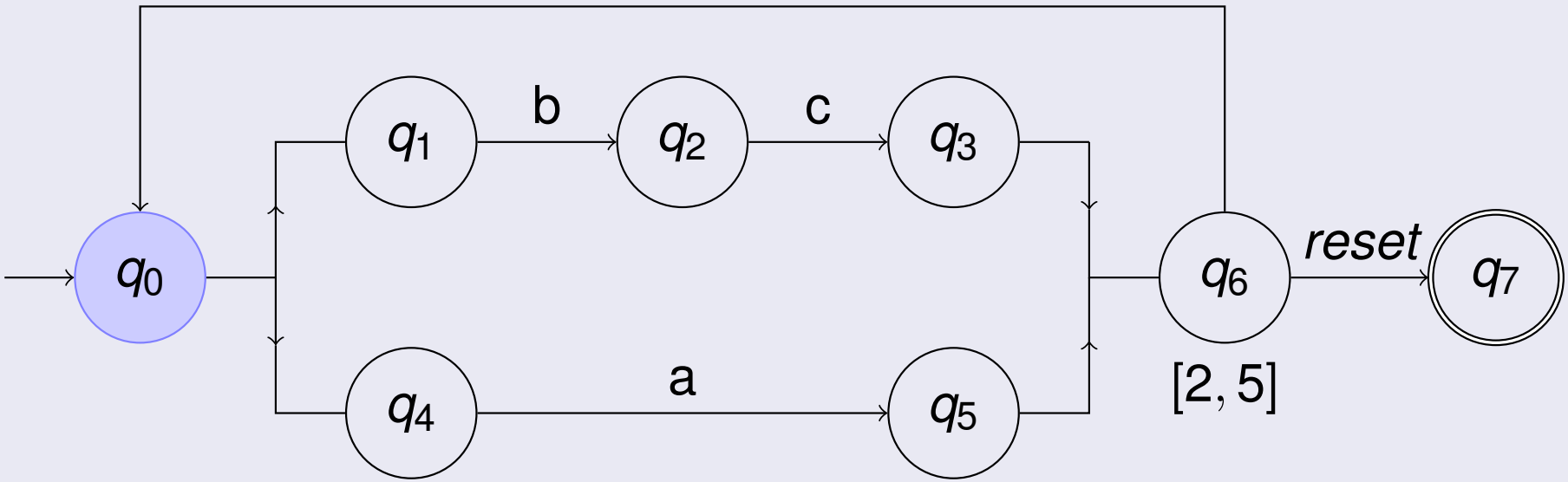
Input: abcbac



Counter $q_6 = 1$

$abcbac \in (a\&bc)^{[2,5]}$?

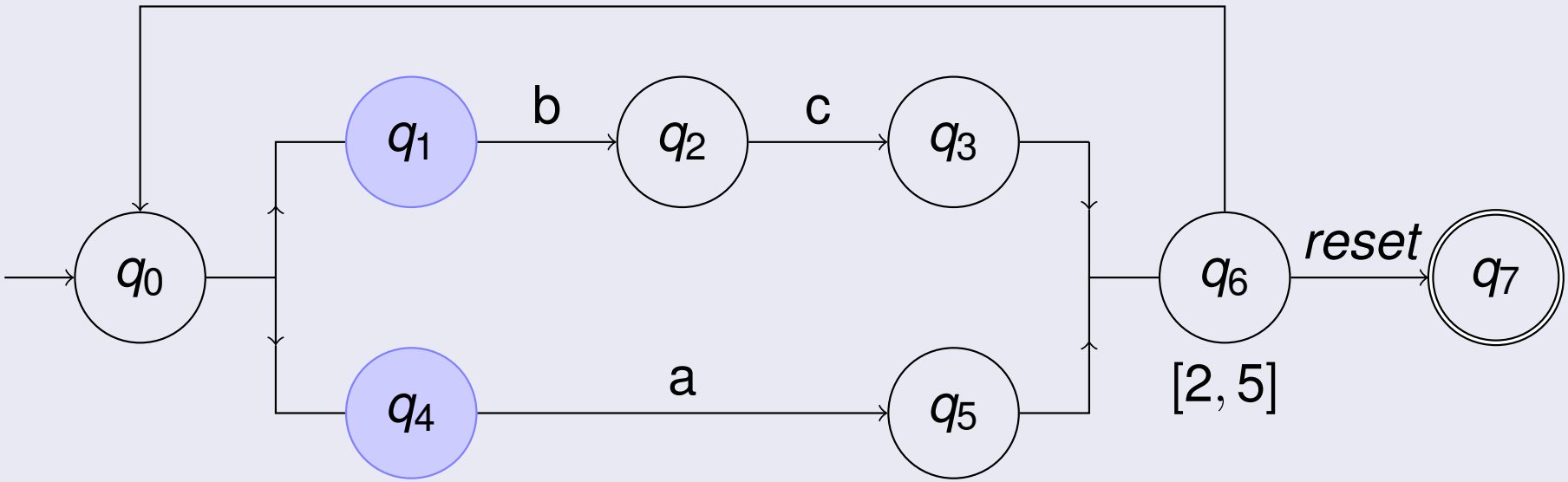
Input: abcbac



Counter $q_6 = 1$

$abcbac \in (a\&bc)^{[2,5]}$?

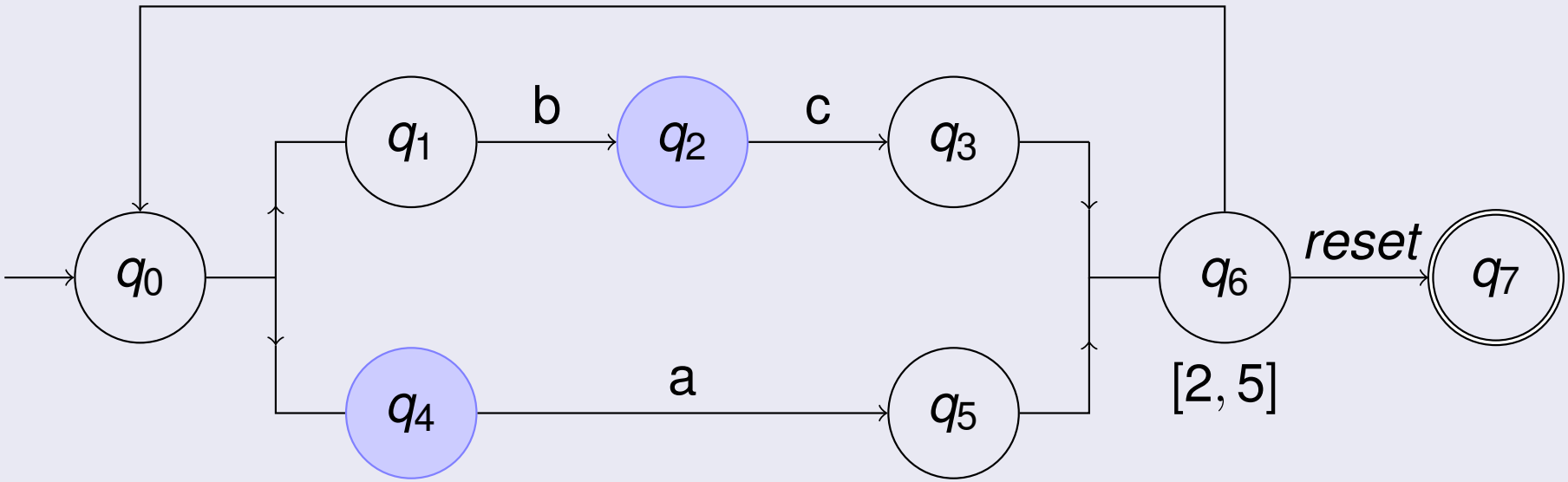
Input: abcbac



Counter $q_6 = 1$

$abcbac \in (a\&bc)^{[2,5]}$?

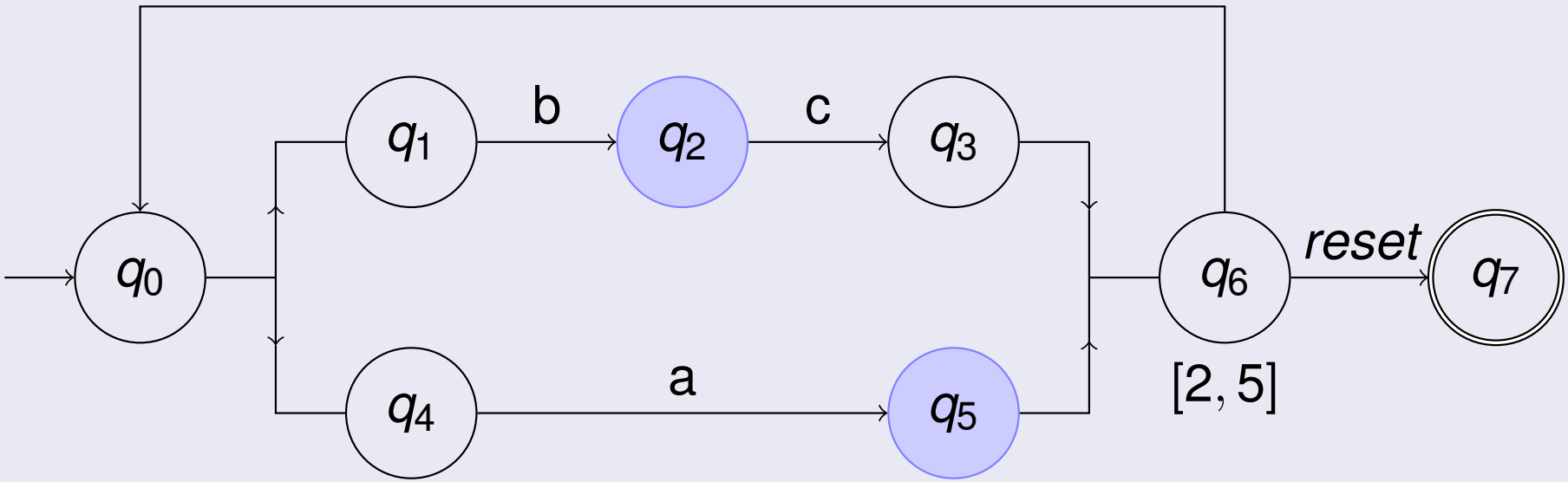
Input: abcbac



Counter $q_6 = 1$

$abcbac \in (a\&bc)^{[2,5]}$?

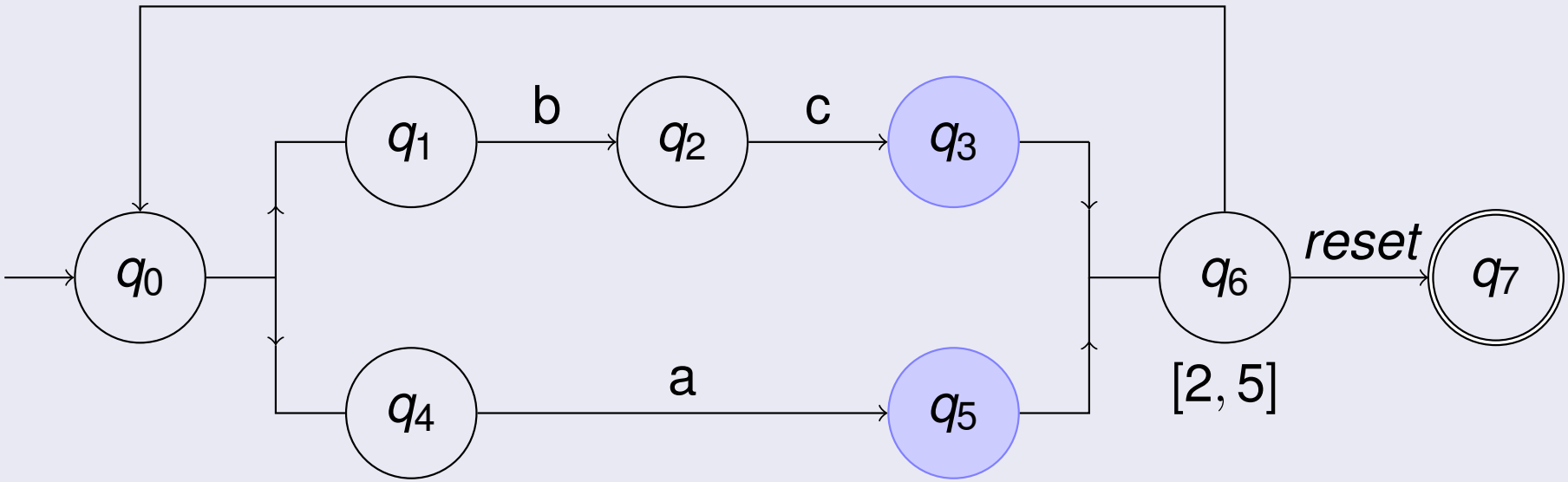
Input: abcbac



Counter $q_6 = 1$

$abcbac \in (a\&bc)^{[2,5]}$?

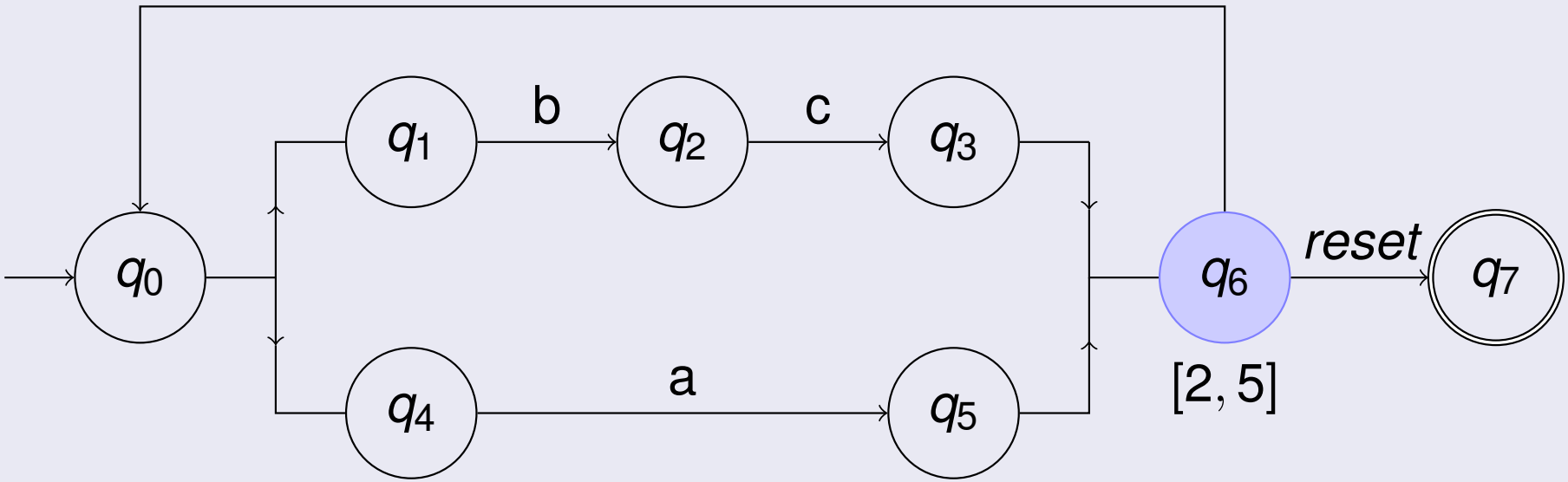
Input: abcbac



Counter $q_6 = 1$

$abcbac \in (a\&bc)^{[2,5]}$?

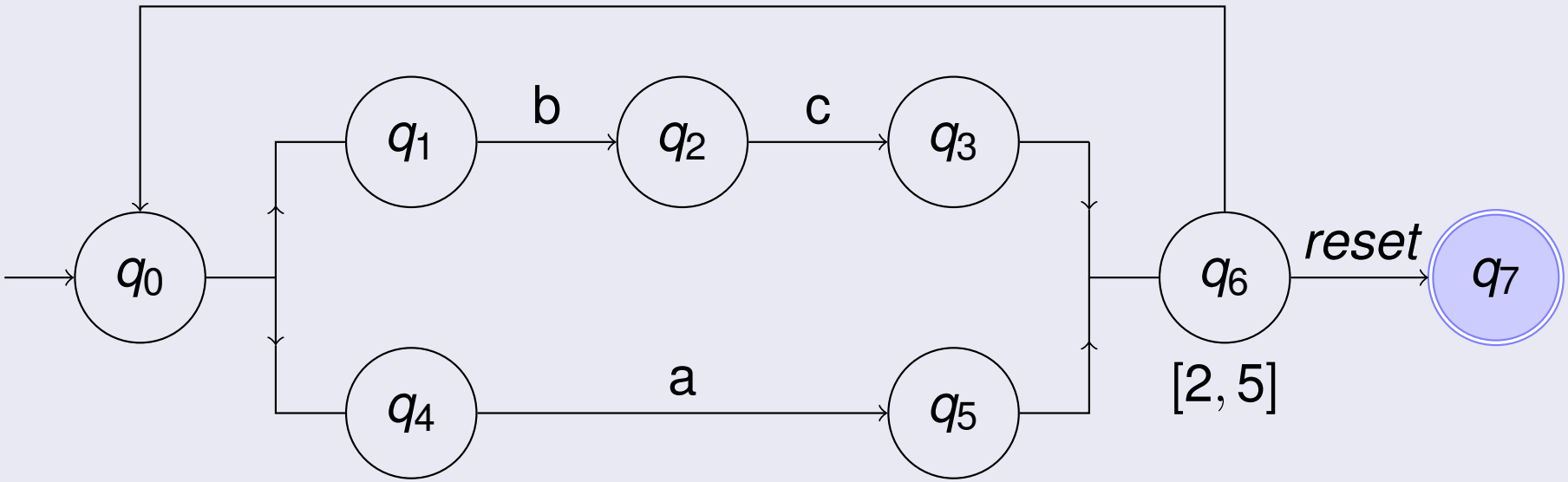
Input: abcbac



Counter $q_6 = 2$

$abcbac \in (a\&bc)^{[2,5]}$?

Input: abcbac



Counter $q_6 = 0$

NFA(#, &)

Proposition

Any RE(#, &) can be translated into an NFA(#, &) of polynomial size.

Corollary

Upperbounds on the complexity of the basic decision problems for NFA(#, &) immediately carry over to the corresponding problems for regular expressions.

NFA(#, &)

Proposition

Any RE(#, &) can be translated into an NFA(#, &) of polynomial size.

Corollary

Upperbounds on the complexity of the basic decision problems for NFA(#, &) immediately carry over to the corresponding problems for regular expressions.

Configurations

- A *configuration* of an NFA(#, &) consists of a number of states and a value for every counter.
- A configuration of an NFA(#, &) is polynomial in the size of the NFA(#, &).

Complexity of NFA(#, &)

Theorem

	equivalence	inclusion	intersection	membership
NFA	PSPACE	PSPACE	PSPACE	PTIME
NFA(#)	EXPSPACE	EXPSPACE	PSPACE	NP-hard in PSPACE
NFA(&)	EXPSPACE	EXPSPACE	PSPACE	PSPACE
NFA(#, &)	EXPSPACE	EXPSPACE	PSPACE	PSPACE

Complexity of NFA(#, &)

Theorem

	equivalence	inclusion	intersection	membership
NFA	PSPACE	PSPACE	PSPACE	PTIME
NFA(#)	EXPSPACE	EXPSPACE	PSPACE	NP-hard in PSPACE
NFA(&)	EXPSPACE	EXPSPACE	PSPACE	PSPACE
NFA(#, &)	EXPSPACE	EXPSPACE	PSPACE	PSPACE

Membership of RE(#, &)

- Membership for RE(#) is in PTIME (Kilpelainen 2003).
- Membership for RE(&) is NP-complete (Mayer, Stockmeyer 1994).

Outline

Complexity of regular expressions

Theorem

	equivalence	inclusion	intersection
RE	PSPACE	PSPACE	PSPACE
RE(#)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
RE(&)	EXPSPACE	EXPSPACE	**PSPACE**
RE(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**

Complexity of regular expressions

Theorem

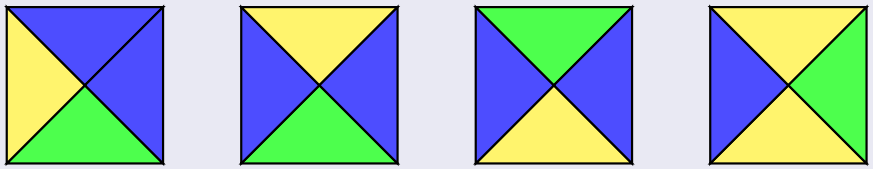
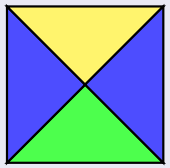
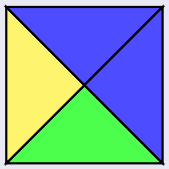
	equivalence	inclusion	intersection
RE	PSPACE	PSPACE	PSPACE
RE(#)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
RE(&)	EXPSPACE	EXPSPACE	**PSPACE**
RE(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**

Intersection in PSPACE

- Translation of RE(#, &) to NFA gives EXPSPACE upperbound.
- Translation of RE(#, &) to NFA(#, &) gives PSPACE upperbound.

Equivalence of RE(#) is EXPSPACE-hard

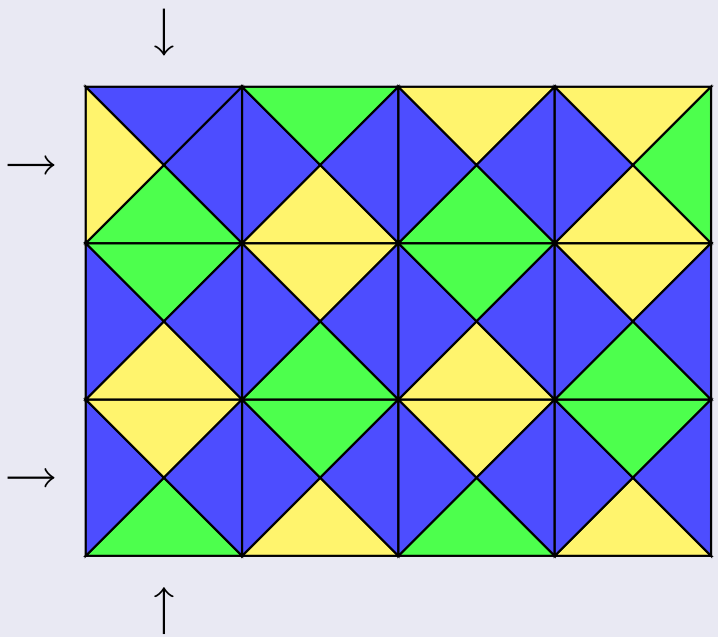
EXP-Corridor tiling instance

- Set of tiles T:  w x y z
- A number n : 2.
- A tile t_{bot}: 
- A tile t_{top}: 

EXP-Corridor tiling

EXP-Corridor tiling problem

- Given a tiling instance, does there exist a tiling of the $m \times 2^n$ corridor, for some m, with t_{bot} and t_{top} as the first tiles of bottom and top row?



- EXP-corridor tiling is EXPSPACE-complete.
- String representation: $\triangle xyxy \triangle yxyx \triangle wyxz \triangle$.

Reduction of EXP-corridor tiling

Reduction

- Construct a regular expression r which accepts all strings except those strings which encode a valid tiling.
- $L(r) = \Sigma^*$ iff there does not exist a valid tiling.
- Expression r consists of disjunction of expressions which capture all possible mistakes in a string.

Reduction of EXP-corridor tiling

Possible mistakes

- There are no 2^n tiles between two successive delimiters: $\Sigma^* \triangle (T^{[0,2^{n-1}]} + T^{[2^{n+1},2^{n+1}]} T^*) \triangle \Sigma^*$.
- Two tiles with different left and right colours are next to each other: $\Sigma^* t_1 t_2 \Sigma^*$, for all t_1, t_2 with different left and right colours.
- Two tiles with different top and bottom colours are on top of each other: $\Sigma^* t_1 \Sigma^{[2^n,2^n]} t_2 \Sigma^*$, for all t_1, t_2 with different top and bottom colors.
- ...

Chain regular expressions (CHAREs)

Practical study (Bex et. al. 2004)

- Majority of regular expressions used in practical DTDs and XSDs are very simple: CHain Regular Expressions (CHAREs).
- A CHARE is a sequence of factors where a factor is $(a_1 + \ldots + a_n)$, $(a_1 + \ldots + a_n)^*$, $(a_1 + \ldots + a_n)^+$, or $(a_1 + \ldots + a_n)?$.

CHARE(#)s

- We extend CHAREs by allowing factors $(a_1 + \ldots + a_n)^{[k,l]}$.
- $(a + b)^*(a + d)^{[3,5]}(b + c + d)?$ is a CHARE(#).
- $(a + b) + (a^* b^*)$ is not a CHARE.

Complexity of CHARE(#)s

Theorem

	INCLUSION	EQUIVALENCE	INTERSECTION
CHARE	PSPACE	in PSPACE	PSPACE
CHARE(#)	**EXPSPACE**	**in EXPSPACE**	**PSPACE**
CHARE(a, $a?$)	coNP	in PTIME	NP
CHARE(a, a^*)	coNP	in PTIME	NP
CHARE(a, $a?$, $a\#$)	**coNP**	**in PTIME**	**NP**
CHARE(a, $a\#^{>0}$)	**in PTIME**	**in PTIME**	**in PTIME**

Outline

Theorem

Theorem equivalence and inclusion problem (Martens et. al. 2004)

Let R be a class of regular expressions and C a complexity class. Then the following are equivalent:

- Inclusion for R is in C.
- Inclusion for DTD(R) is in C.
- Inclusion for single-type EDTD(R) is in C.

The same holds for equivalence.

Theorem intersection problem (Martens et. al. 2004)

Let R be a class of regular expressions and C a complexity class. Then the following are equivalent:

- Intersection for R is in C.
- Intersection for DTD(R) is in C.

Complexity of DTD and XML Schema

Results

	equivalence	inclusion	intersection
DTDs with RE	PSPACE	PSPACE	PSPACE
DTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
single-type EDTDs with RE	PSPACE	PSPACE	
single-type EDTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	
EDTDs with RE			
EDTDs with RE(#), RE(&), or RE(#, &)			

Complexity of all schema languages

Results

	equivalence	inclusion	intersection
DTDs with RE	PSPACE	PSPACE	PSPACE
DTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
single-type EDTDs with RE	PSPACE	PSPACE	EXPTIME
single-type EDTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**EXPTIME**
EDTDs with RE	EXPTIME	EXPTIME	EXPTIME
EDTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**EXPTIME**

Upperbounds EDTDs

Equivalence and inclusion problem for EDTDs in EXPSPACE

- Translation of RE(#, &) to NFA gives 2EXPTIME upperbound.
- Translation of RE(#, &) to NFA(#, &) gives EXPSPACE upperbound.

Intersection problem for EDTDs in EXPTIME

- Translation of RE(#, &) to NFA gives 2EXPTIME upperbound.
- Translation of RE(#, &) to NFA(#, &) gives EXPTIME upperbound.

Outline

XML Schema constraints

Unique Particle Attribution (UPA)

- UPA requires the used regular expressions to be deterministic or one-unambiguous.
- Our model for XML Schema does not incorporate that constraint.

Reason

- One-unambiguity is not yet fully understood for RE(#, &). (Preliminary work by Bruggeman-Klein (1994) and Kilpelaïnen (2005,2006)).
- One-unambiguity is not the right subclass of RE to get tractable results for the decision problems.
  - Intersection of one-unambiguous standard REs is PSPACE-complete (Martens et. al.).
  - Inclusion of one-unambiguous RE(#)s is coNP-hard (Kilpelaïnen).

Conclusion

Simplification

The simplification problem goes from EXPTIME to EXPSPACE when using RE(#, &) instead of RE.

Conclusion

- Complexity increase by adding the numerical occurrence and shuffle operator ranges from no increase to one exponential.
- Even for small subclasses of the regular expressions, the basic decision problems remain intractable.
- Is there a robust subclass of the schema languages for which the basic decision problems are in PTIME?