From Ambiguous Regular Expressions to Deterministic Parsing Automata

Angelo Borsotti Luca Breveglieri Stefano Crespi Reghizzi
Angelo Morzenti

Politecnico di Milano - Piazza Leonardo Da Vinci n. 32 - I-20133 Milano - Italy

angelo.borsotti@mail.polimi.it luca.breveglieri@polimi.it
stefano.crespireghizzi@polimi.it angelo.morzenti@polimi.it

CIAA - 17-21 Aug. 2015 - Umeå - Sweden

downloads

paper: full text - *LNCS* book *SW* tool: short note - GitHub

Table of Contents

Problem Statement and State of the Art

On the status of parsing Regular Expressions (*RE*s)

Given a *RE*, it is well known that *checking* if a string belongs to the language $L(RE)$ can be done by a *DFA* (in real-time).

A well known method to build a recognizer *DFA* for a given *RE* is the Berry-Sethi algorithm (1986).

Parsing a *RE*, i.e., recognizing a string and *building* a syntax tree thereof, has been tackled (and solved) more recently:

- by a Thompson-based *NFA* recognizer (Dubè and Feeley, 2000)
- by the Brzozowski derivative method (Sulzmann and Lu, 2014)

On a different line, an inspiring work (Okui and Suzuki, 2010) resorts to a *NFA* that incorporates the metasymbolic structure of the *RE*, and that has a linear-time complexity for parsing a string.

Research Direction and Objectives

Objectives of the present research work

Develop a *DFA*-based method to (efficiently) parse a *RE*.

Generalize the classical Berry-Sethi algorithm (*BS*).

Obtain a Berry-Sethi Parser (*BSP*) that:

- for non-ambiguous *RE*s, efficiently recognizes a string, and outputs the unique syntax tree of the string
- for ambiguous *RE*s, represents all the syntax trees, and outputs one selected tree, e.g., *Greedy* or *POSIX*

Our method merges the metasymbolic structure of the *RE* and the Berry-Sethi approach, to construct a *DFA* that works both as a pure recognizer and as a parser able to build syntax trees.

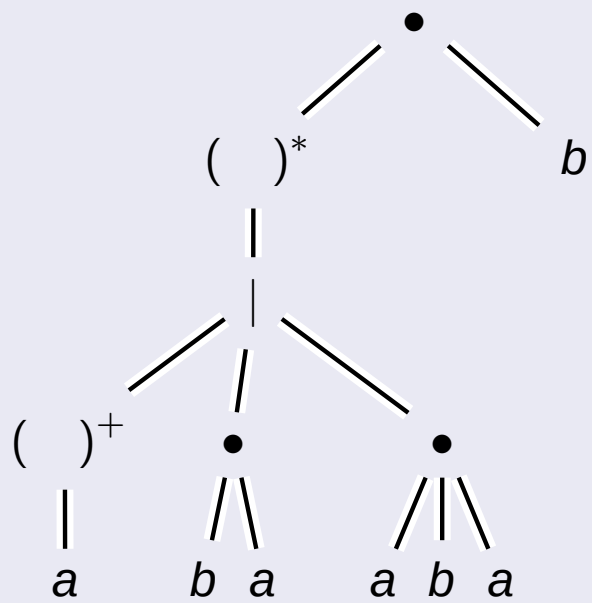
It is grounded in the theory of the locally testable languages.

Consider the following ambiguous *RE* e (it is the running example):

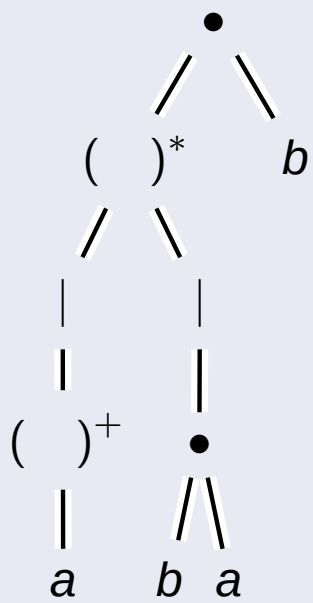
$$e = \left((a)^+ \mid b \cdot a \mid a \cdot b \cdot a \right)^* \cdot b$$

Abstract Syntax Tree (*AST*) samples

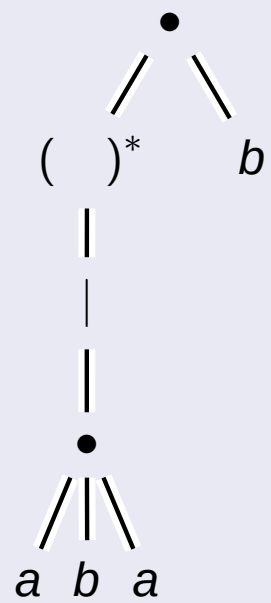
abstract syntax tree of the *RE* e



syntax trees of the string $abab \in L(e)$



Greedy

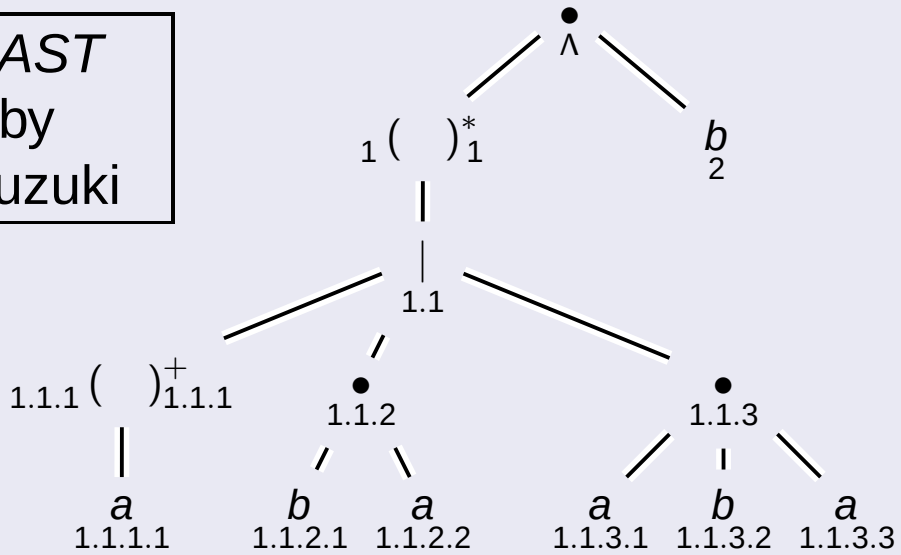


POSIX

Correspondence between a *structurally marked RE* $\hat{e}$ and its *AST*

$$\hat{e} = {}_{1}\Big({}_{1.1.1}\big(a \big)^{+}_{1.1.1} \;\Big|_{1.1}\; b_{1.1.2.1} \cdot_{1.1.2} a_{1.1.2.2} \;\Big|_{1.1}\; a_{1.1.3.1} \cdot_{1.1.3} b_{1.1.3.2} \cdot_{1.1.3} a_{1.1.3.3} \Big)^{*}_{1} \cdot_{\Lambda} b_{2}$$

annotated *AST* notation by Okui and Suzuki



Linearized Syntax Tree (*LST*) of the non-ambiguous string $ab \in L(e)$

$$\Lambda\Big({}_{1}\big({}_{1.1}\big({}_{1.1.1}\big({}_{1.1.1.1}\big(a \big)_{1.1.1.1} \big)_{1.1.1} \big)_{1.1} \big)_{1} \;\; {}_{2}\big(b \big)_{2} \Big)_{\Lambda}$$

Structural marking is hardly readable, a lighter marking helps.

Numerically marked *RE* $\hat{e}$ with *partial* marking

$$\hat{e} = {}_1\big({}_2(a_3)^{+}_{2} \mid b_4\, a_5 \mid a_6\, b_7\, a_8 \big)^{*}_{1}\, b_9$$

The metasymbols '$|$' and '$\cdot$' are left unmarked[1].

Numerically marked *RE* $\hat{e}$ with *terminal* marking

$$\hat{e} = \big((a_3)^{+} \mid b_4\, a_5 \mid a_6\, b_7\, a_8 \big)^{*}\, b_9$$

This is used in the classical Berry-Sethi algorithm.

Numerically partially marked *LST*s of the ambiguous string *abab*

Greedy: $${}_1\Big({}_2\Big(\underset{3}{a} \Big)_2 \; \underset{4}{b}\; \underset{5}{a} \Big)_1 \; \underset{9}{b}$$

POSIX: $${}_1\Big(\underset{6}{a}\; \underset{7}{b}\; \underset{8}{a} \Big)_1 \; \underset{9}{b}$$

A *SW* tool should use the structural representation of the *LST*.

[1] They can be easily inferred from their neighbours.

Use square brackets ‘ [’ and ‘] ’ as the new metasymbolic parentheses.

Reintroduction rules of the metasymbols - from $\hat{e}$ to $\check{e}$

parenthesis pair	reintroduction rule
$_h(\hat{e})_h$	$_h\big(\ [\hat{e}]\ \big)_h$
$_h(\hat{e})^*_h$	$\Big[\ {}_h\big(\ [\hat{e}]^+\ \big)_h\ \mid 1_h\ \Big]$
$_h(\hat{e})^+_h$	$_h\big(\ [\hat{e}]^+\ \big)_h$

Reintroduction example - *RE*s: e, $\hat{e}$ (marked) and $\check{e}$ (reparenthetized)

$$e = \Big((a)^+ \mid ba \mid aba\Big)^* b \qquad \hat{e} = {}_1\Big({}_2(\,a_3\,)^+_2 \mid b_4\,a_5 \mid a_6\,b_7\,a_8\Big)^*_1\ b_9$$

$$\check{e} = \Big[\ {}_1\big(\ \big[\ {}_2(\ [\,a_3\,]^+\ )_2 \mid b_4\,a_5 \mid a_6\,b_7\,a_8\ \big]^+\ \big)_1 \mid 1_1\ \Big]\ b_9$$

$$abab \in L(e) \iff {}_1\Big({}_2\Big(\begin{matrix}a\\3\end{matrix}\Big)_2\ \begin{matrix}b\\4\end{matrix}\ \begin{matrix}a\\5\end{matrix}\Big)_1\ \begin{matrix}b\\9\end{matrix} \in L(\check{e})$$

Alphabets

$$
\begin{array}{lcll}
\Sigma & = & \{\, a, b, \ldots \,\} & \text{terminals} \\
\widehat{\Sigma} & = & \{\, a_1, a_2, \ldots \,\} & \text{marked terminals} \\
\widehat{M} & = & \{\, 1_h{}^{a},\ {}_h(,)_h \mid \quad 1 \leq h \leq \text{a bound} \,\} & \text{marked metasymbols} \\
\widehat{\Omega} & = & \widehat{\Sigma} \cup \widehat{M} & \text{all marked symbols}
\end{array}
$$

[a] Symbol 1_h represents the empty string ε at position h.

Languages

original: $L(e) \subseteq \Sigma^*$ reparenthetized: $L(\check{e}) \subseteq \widehat{\Omega}^+$

Important observation

Given a *RE* e, it is evident that the reparenthetized language $L(\check{e})$ is the language of all and only the *LST*s of the strings generated by e.

Let a, b be terminals, x, y strings (possibly empty) and L a language.

Classical local sets of a (generic) language L

$$Ini(L) = \{ a \mid ax \in L \} \qquad Fin(L) = \{ b \mid xb \in L \}$$
$$Dig(L) = \{ ab \mid xaby \in L \} \qquad Fol(L, a) = \{ b \mid ab \in Dig(L) \}$$

Locally testable language L

$$L = Ini \cdot \Sigma^* \cap \Sigma^* \cdot Fin \cap \{ \text{strings with only digrams} \in Dig \}$$

A locally testable language is characterized by its *Ini*, *Fin* and *Dig* sets.

The local sets of a regular language are easily computable[2].

Important observation

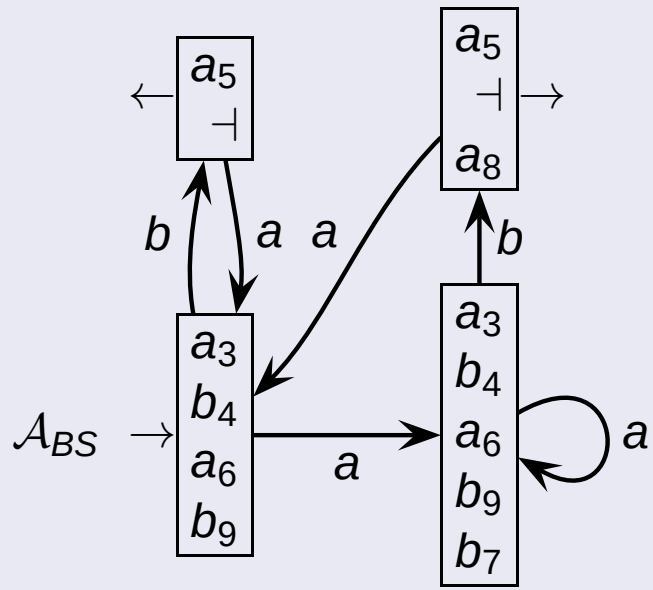
Since the reparenthetized *RE* $\check{e}$ is *linear* (no repeated symbols), the (reparenthetized) language $L(\check{e}) \subseteq \widehat{\Omega}^+$ of the *LST*s is *locally testable* !

[2] See e.g., Berstel and Pin (1996).

$$\hat{e} = \left(\left(a_3 \right)^+ \mid b_4\, a_5 \mid a_6\, b_7\, a_8 \right)^*\, b_9\, \dashv$$

Classical Berry-Sethi recognizer *DFA* $\mathcal{A}_{BS}$

Followers	Initials	a_3	b_4	a_6	b_9
	a_3	a_3	b_4	a_6	b_9
	b_4	a_5			
	a_5	a_3	b_4	a_6	b_9
	a_6	b_7			
	b_7	a_8			
	a_8	a_3	b_4	a_6	b_9
	b_9	$\dashv$			



It holds: $L\left(\mathcal{A}_{BS}\right) = L\left(e\right)$ (Berry and Sethi, 1986).

Let $\mu \in \widehat{M}^*$ be a (possibly empty) sequence of marked *metasymbols*, without repetitions[3]. Let a_h, $b_k \in \widehat{\Sigma}$ be marked *terminal* symbols.

Extended set of the initial terminals of the *RE e* - Ini_E

$$Ini_E\ (e) = \{\ \mu\, a_h \mid \quad Ini\ (\mu\, a_h) \subseteq Ini\ (\check{e})\ \wedge\ Dig\,(\mu\, a_h) \subseteq Dig\ (\check{e})\ \}$$

Extended set of the terminals following a_h in the *RE e* - Fol_E

$$Fol_E\ (e,\, a_h) = \{\ \mu\, b_k \mid \quad Dig\,(a_h\, \mu\, b_k) \subseteq Dig\ (\check{e})\ \}$$

Intuitively, such sets represent the acyclic metasymbolic paths in the *RE* to reach the initial terminals or between any two adjacent ones.

The extended local sets of a *RE e* can be easily computed (see paper), starting from the classical local sets of the reparenthetized *RE* $\check{e}$.

[3] So the number of μ prefixes is finite.

$$\hat{e} = {}_1\Big({}_2\big(a_3 \big)^{+}_{2} \;\Big|\; b_4\, a_5 \;\Big|\; a_6\, b_7\, a_8 \Big)^{*}_{1}\; b_9 \;\dashv$$

Extended sets of the initials and followers of the *RE e*

Initials	${}_1({}_2(a_3$	${}_1(b_4$	${}_1(a_6$	$1_1\, b_9$	
Followers					
a_3	a_3	$)_2\, {}_2(a_3$	$)_2\, b_4$	$)_2\, a_6$	$)_2\,)_1\, b_9$
b_4	a_5				
a_5	${}_2(a_3$	b_4	a_6	$)_1\, b_9$	
a_6	b_7				
b_7	a_8				
a_8	${}_2(a_3$	b_4	a_6	$)_1\, b_9$	
b_9	$\dashv$				

Item - a fragment of a syntax tree

$$\text{item id}: \left\langle \mu\, a_h,\ \overbrace{\{\,\text{item id.s}\,\}}^{\text{links to predecessors}} \right\rangle \qquad \mu \in \widehat{M}^* \quad a_h \in \widehat{\Sigma} \cup \dashv$$

The prefix μ must not contain any pair of repeated metasymbols.

The states of the *BSP DFA* $\mathcal{A}_{BSP}$ are sets (non-empty) of items.

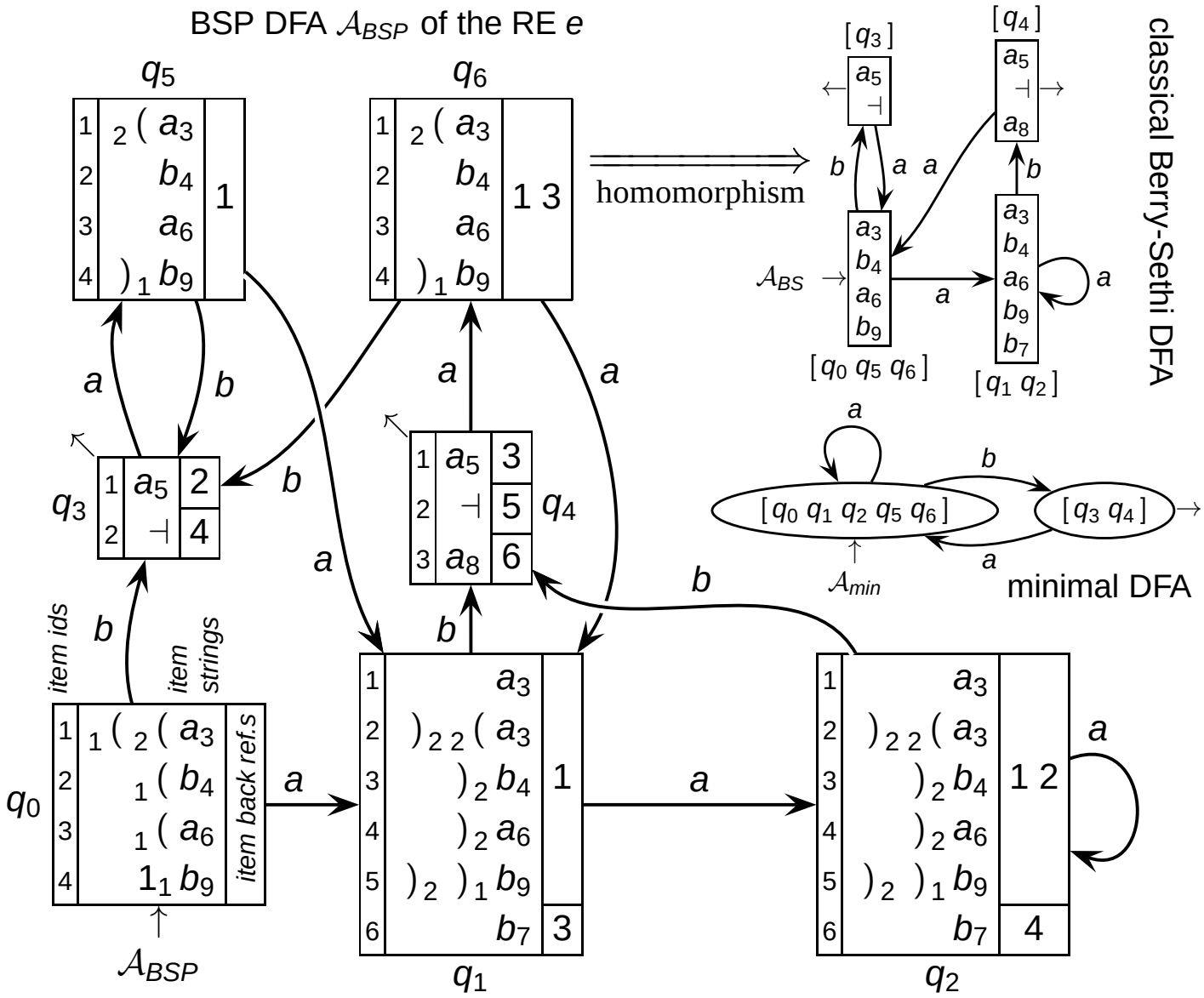
Sketch of the *BSP* construction algorithm

Input: extended local sets Ini_E and Fol_E of a *RE* e.
Output: state-transition graph of the *DFA* $\mathcal{A}_{BSP}$ of e.

Set the initial state of $\mathcal{A}_{BSP}$ to the set $Ini_E\,(e\,\dashv)$.
For each not yet examined state $p \in \mathcal{A}_{BSP}$, for each unmarked terminal $a \in \Sigma$
  for all the (source) items $id: \left\langle \mu\, a_h, \{\ldots\} \right\rangle \in p$ such that $unmark\,(a_h) = a$
    create[a] a new state q with all the (target) items in the sets $Fol_E\,(e\,\dashv, a_h)$,
    link such targets to their sources id, and create a new transition $p \xrightarrow{a} q$.
Set the final states of $\mathcal{A}_{BSP}$ to those that contain $\dashv$.

---

[a] Unless it was already created.


BSP DFA $\mathcal{A}_{BSP}$ of the RE e
q_5
q_6
homomorphism
classical Berry-Sethi DFA
$[q_3]$
$[q_4]$
$\mathcal{A}_{BS}$
$[q_0\ q_5\ q_6]$
$[q_1\ q_2]$
q_3
q_4
$[q_0\ q_1\ q_2\ q_5\ q_6]$
$[q_3\ q_4]$
$\mathcal{A}_{min}$
minimal DFA
item ids
item strings
item back ref.s
q_0
$\mathcal{A}_{BSP}$
q_1
q_2

Equivalence of the *BSP* and *BS DFA*s

For a *RE* e, the two *DFA*s $\mathcal{A}_{BSP}$ (parser) and $\mathcal{A}_{BS}$ (classical recognizer) of e are equivalent, i.e., both recognize the same language $L(e)$.

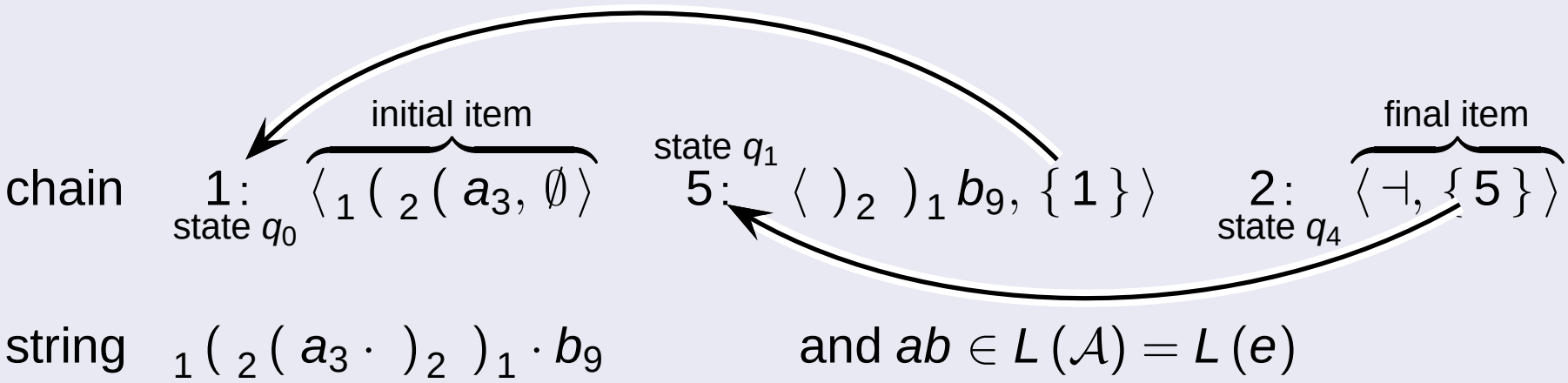
Sketch of proof

The classical recognizer $\mathcal{A}_{BS}$ is a homomorphic image of the new parser $\mathcal{A}_{BSP}$. By canceling the item metasymbols, as well as the references to the predecessor items, the *DFA* $\mathcal{A}_{BSP}$ (after merging the states that have turned identical) becomes exactly the *DFA* $\mathcal{A}_{BS}$.

Further observations

In general neither $\mathcal{A}_{BSP}$ nor $\mathcal{A}_{BS}$ are minimal. The sets of backward references in a state of $\mathcal{A}_{BSP}$ are either disjoint or coincident. This helps us optimize the parsing. The time complexity of the *BSP* construction is (at least) exponential in the size of the alphabet $\widehat{\Omega}$.

Item chain and string - sample from the running example



Extended language of the *BSP*

Given a *BSP* $\mathcal{A}$, define the *extended language* $\widehat{L}(\mathcal{A}) \subseteq \widehat{\Omega}^+$ to be the set of the strings (over marked symbols) of all the item chains in $\mathcal{A}$.

Infinite ambiguity - Kleene star (or cross) with a nullable argument

A string is infinitely ambiguous if and only if it has a *LST* that contains a substring of type $1_h \, \mu \, 1_h$, for some mark h and metasymbols $\mu \in \widehat{M}^*$.

Main theorem - The *BSP* extended language *is the set* of *RE LSTs*

Let e be a *RE* and $\mathcal{A}$ the *BSP* of e. The extended language $\hat{L}(\mathcal{A})$ of $\mathcal{A}$ coincides with the language $L(\check{e})$ of the *LSTs* of e, removing from $L(\check{e})$ the strings that contain a substring of type $1_h \, \mu \, 1_h$ (infinite ambiguity).

Sketch of proof

Language $\hat{L}(\mathcal{A})$ is locally testable and defined by the extended local sets[a] Ini_E and Fol_E of e, characterized by the classical local sets *Ini*, *Fin* and *Dig* of $\check{e}$. Language $L(\check{e})$ is also locally testable and defined by the same classical local sets. Thus $\hat{L}(\mathcal{A})$ and $L(\check{e})$ coincide (see note).

[a] Their definitions rule out any pair of repeated metasymbols, e.g., $1_h \, \mu \, 1_h$.

How to check the ambiguity of a *RE*

A *RE* is non-ambiguous if and only if every item of its *BSP* has at most one predecessor and every state of its *BSP* has at most one final item.

The same holds for a string, by restricting to the string state sequence.

How to parse a non-ambiguous string $w \in L(e)$ - two phases

First recognize w by using the *BSP* $\mathcal{A}$ of e and find the state sequence s of w (s is unique as $\mathcal{A}$ is a *DFA*). **Second** follow back the unique item chain of w in s. The string of such a chain is the unique *LST* of w.

Parsing complexity (non-ambiguous case)

The time complexity of parsing a non-ambiguous string $w \in \Sigma^*$ and building its *LST* is linear in the string length $|w|$.

Notice a string state sequence contains the whole string tree forest[4].

Disambiguation criteria - *Greedy* and *POSIX*

To select a prior *LST* for an ambiguous string, *Greedy*[a] makes the *RE* leftmost choices, *POSIX* (an *IEEE* standard) makes the *RE* longest matches (both criteria are formalized by Frisch and Cardelli, 2004).

[a] Diverging interpretations exist, e.g., Frisch and Cardelli vs *Java.regex*.

How to parse an ambiguous string $w \in L(e)$ - three phases (diff.s red)

First phase as before. **Second** follow back all the item chains of w in s, and tag the touched items. **Third** navigate such tagged items from the initial ones, and at each bifurcation point make the prior *LST* choice.

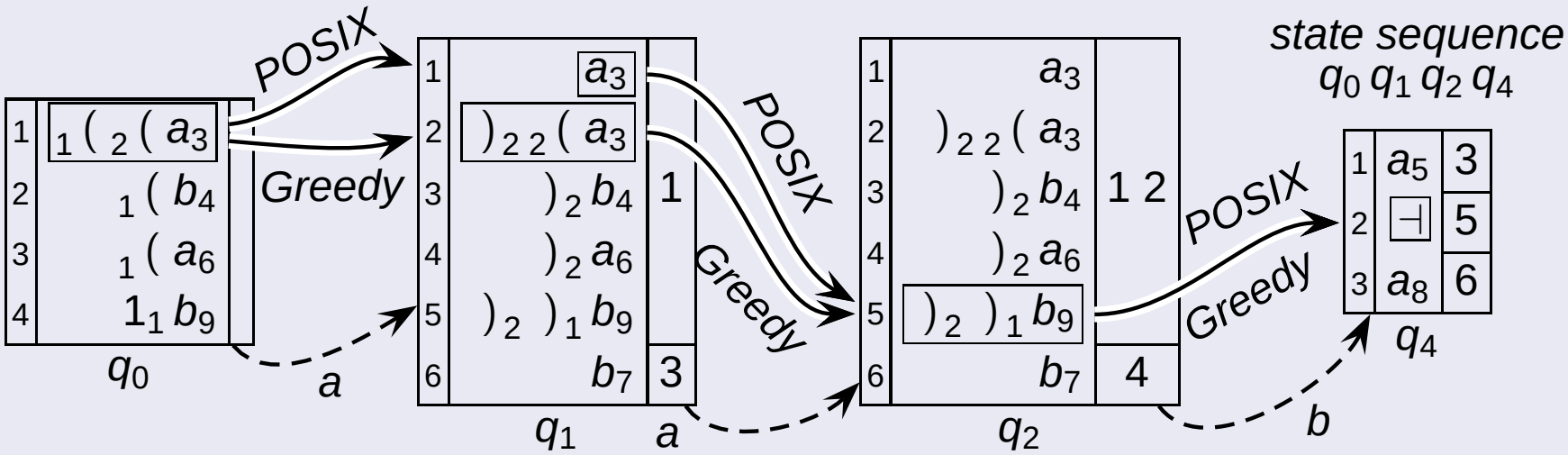
Parsing complexity (ambiguous case)

The time complexity of parsing an ambiguous string $w \in \Sigma^*$ and building its *Greedy* or *POSIX LST* is linear in the string length $|w|$.

[4] In a packed form, with common tree parts shared.

$$\hat{e} = {}_1\left({}_2\left(a_3 \right)_2^+ \mid b_4\, a_5 \mid a_6\, b_7\, a_8 \right)_1^*\, b_9\, \dashv$$

Parsing the ambiguous string $aab \in L(e)$



Here *Greedy* is computed according to the class *Java.regex*.

About the *POSIX* criterion

In the third parsing phase, for *POSIX* we use the Okui and Suzuki algorithm (2010), which dynamically bookkeeps information while navigating the items, and reads it back to make the prior choice.

Current results

A det. parser (*BSP*) for any *RE*, even ambiguous, such that:

- it generalizes the Berry-Sethi pure recognizer
- disambiguation criteria can be easily plugged in
- parsing a string (building its *LST*) is linear-time

The *BSP* also provides a simple test of the *RE* ambiguity.

A demonstrative *SW* tool (in *JavaScript*) is available (GitHub).

Future research

Complexity analytical evaluation vs other approaches.

Experimental performance evaluation (on benchmarks).

Ongoing research . . .

Experimental performance evaluation (using *Java*).

Promising figures (speed-up vs Okui-Suzuki *NFA*).

Three significant references

G. Berry and R. Sethi.
From regular expressions to deterministic automata.
Theor. Comput. Sci., 48(1):117–126, 1986.

A. Frisch and L. Cardelli.
Greedy regular expression matching.
In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *ICALP*, volume 3142 of *LNCS*, pages 618–629. Springer, 2004.

S. Okui and T. Suzuki.
Disambiguation in regular expression matching via position automata with augmented transitions.
In M. Domaratzki and K. Salomaa, editors, *CIAA*, volume 6482 of *LNCS*, pages 231–240. Springer, 2010.