Enumerated BSP automata

Gaétan Hains
Huawei France R&D Center (FRC)
gaetan.hains@huawei.com

GDR-GPL/LAMHA, Paris, Novembre 2015

- BSP automata are finitely-defined systems, but
- finite alphabet $\rightarrow$ regular alphabet ...
- two-level nature of BSP computation

1. BSP words and automata
2. Sequentialization and parallelization
3. BSP regular expressions
4. Minimization and cost model
5. Parallel acceleration
6. Intensional BSP automata

Concurrent systems & theories	Bulk-synchronous parallelism
Arbitrary asynchronism	Structured asynchronism
High-complexity	Low-complexity
Distributed computing	Parallel computing
Unpredictable performance	Predictable performance
Endless processes like servers	Finite processes like algorithms
Not scalable	Massively scalable
Pairwise synchronizations	Collective synchronizations
Implicit shared memory	Explicit distributed memory
Implicit processes	Explicit processes (pid variable)

Bulk-synchronous words and languages

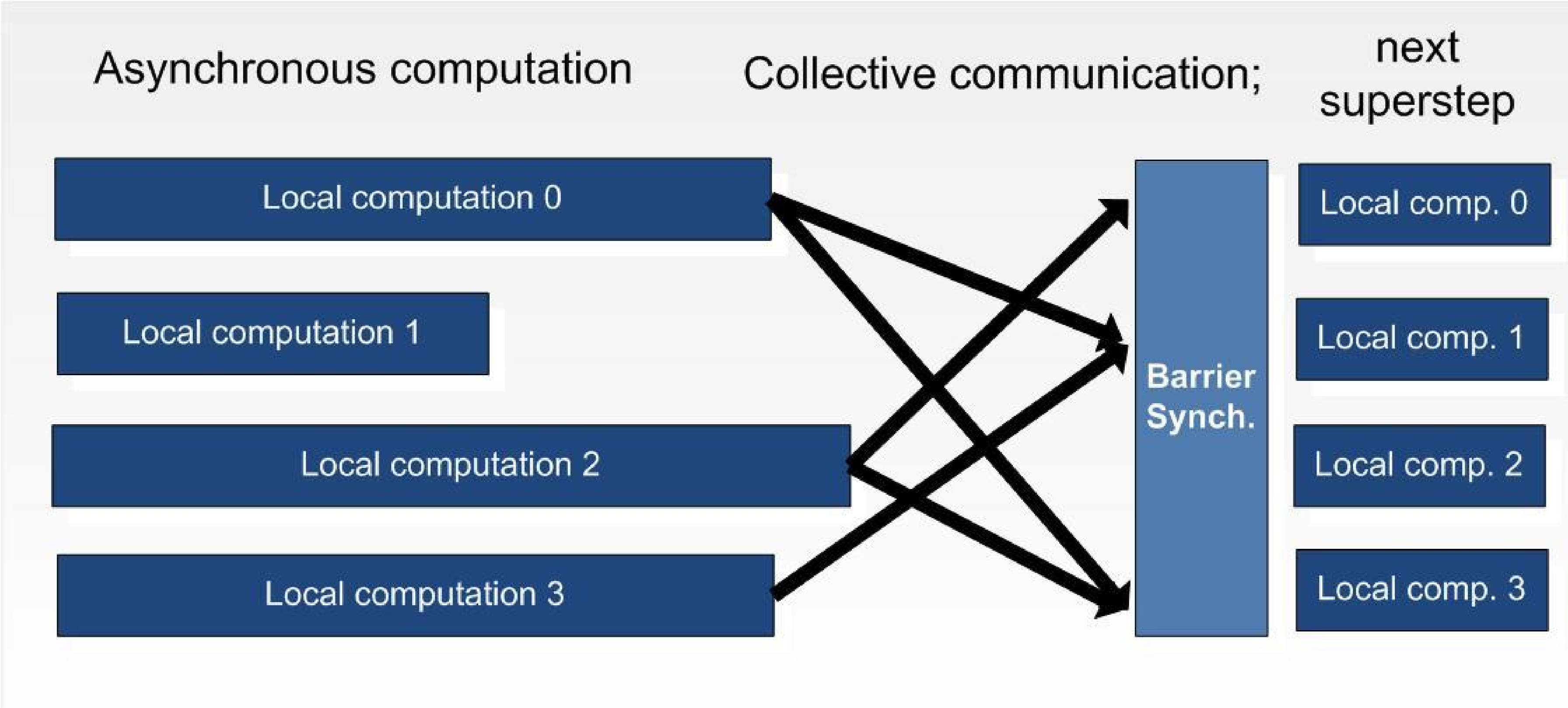
Definition 1. *Elements of* $(\Sigma^*)^p$ *are called* word-vectors*.*

A BSP word *over* Σ *is a sequence of word-vectors*

i.e. a sequence of $((\Sigma^*)^p)^*$*.*

A BSP language *over* Σ *is a set of BSP words over* Σ*.*

Figure 1: A BSP superstep



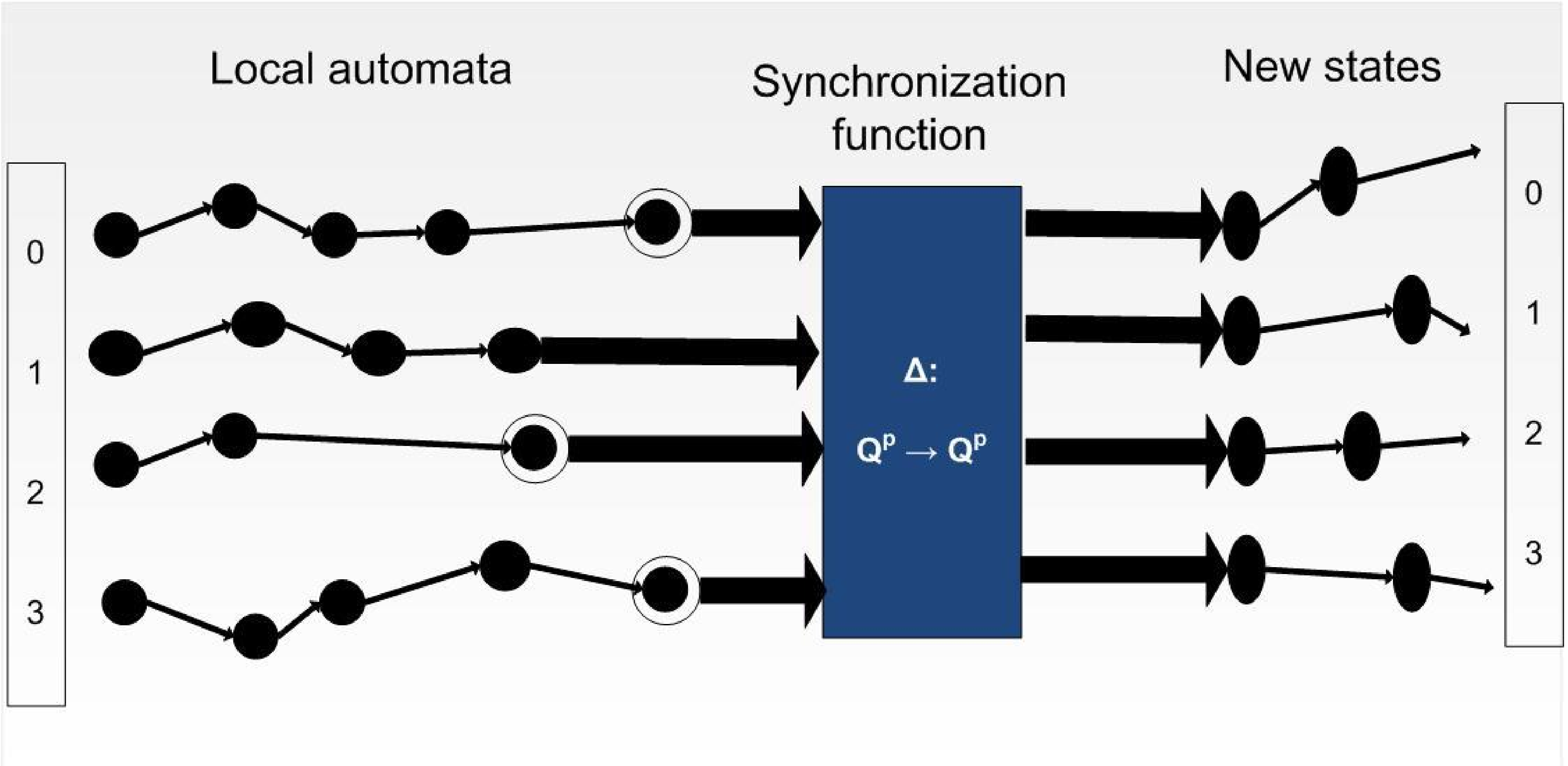
Bulk-synchronous automata

Definition 2. *BSP automaton* $\vec{A} = (\{Q^i\}_{i\in[p]}, \Sigma, \{\delta^i\}_{i\in[p]}, \{q_0^i\}_{i\in[p]}, \{F^i\}_{i\in[p]}, \Delta)$ *with* $(Q^i, \Sigma, \delta^i, q_0^i, F^i)$ *a DFA, and* $\Delta : \vec{Q} \to \vec{Q}$ *is called the* synchronization function *where* $\vec{Q} = (Q^0 \times ... \times Q^{(p-1)})$ *is the set of* global states*.*

1. If the sequence of word vectors is empty, stop; otherwise continue.
2. If $< w^0, \ldots, w^{p-1} >$ is the first word vector. Local automaton i applies w^i to its initial state and transition function to reach some state q^i, **not necessarily an accepting**.
3. The synchronization function maps $\Delta :< q^0, \ldots, q^{p-1} > \to < q'^0, \ldots, q'^{p-1} >$.
4. If there are no more word vectors, and $\forall i.\ q'^i \in F^i$, the BSP word is accepted.
5. If there are no more word vectors, and $\exists i.\ q'^i \notin F^i$, the BSP word is rejected.
6. If there are more word vectors, control returns to step 2. but with local automaton i in state q'^i, for every location i.

Proposition 1. *A BSP automaton is equivalent to a deterministic automaton over (the infinite alphabet of) word-vectors.*

Figure 2: A BSP automaton



Non-determinism and empty transitions

Definition 3. *A non-deterministic BSP automaton (NBSPA) is a BSP automaton whose local automata are of type*

$$Q \times \Sigma \to \mathcal{P}(Q)$$

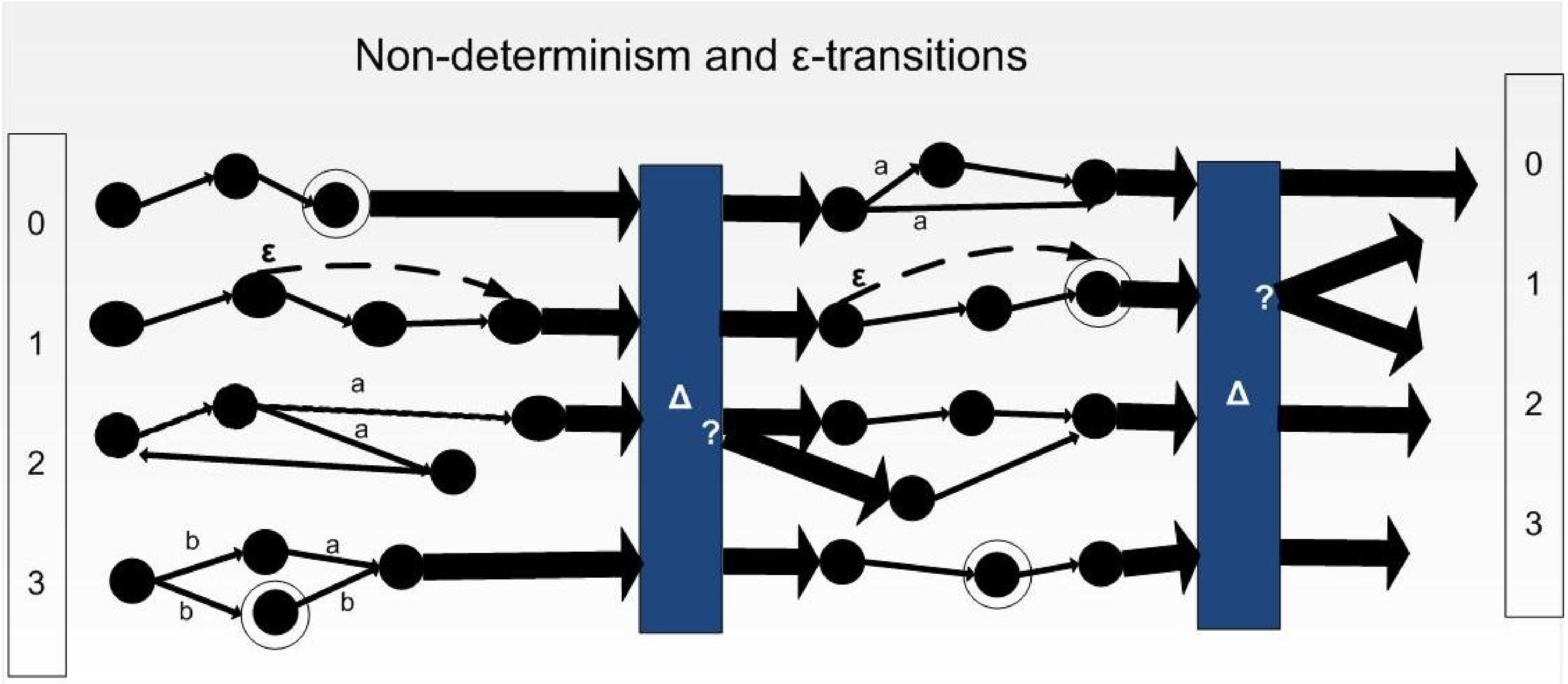
and whose synchronization function $\Delta : \vec{Q} \to \mathcal{P}(\vec{Q})$*.*

Definition 4. *A non-deterministic BSP automaton with empty transitions (ϵ-NBSPA) is a NBSPA with local ϵ-NFA .*

Proposition 2. *The language of a NBSPA can be accepted by a deterministic BSP automaton.*

Proposition 3. *The language of an ϵ-NBSPA can be recognized by a NBSPA.*

Figure 3: An ϵ-NBSPA



Sequentialization

Definition 5. *Word vectors sequentialization, add locations* Seq : $(\Sigma^*)^p \to (\Sigma \times [p])^*$ *BSP words, add semicolons for barriers:*

$$\begin{aligned} \textit{Seq}(\epsilon) &= \epsilon \\ \textit{Seq}(\vec{v}_1 \ldots \vec{v}_n) &= \textit{Seq}(\vec{v}_1); \ldots ; \textit{Seq}(\vec{v}_n); \end{aligned}$$

NOTE: $\vec{\epsilon} =< \epsilon, \ldots, \epsilon > \neq \epsilon$

Seq $< \epsilon, \ldots, \epsilon >= (;)$ (one barrier)

Proposition 4. $\forall$ *BSP automaton* A, $\exists$ *DFA* Seq(A) *on* $(\Sigma \times [p]) \cup \{;\}$ *such that* Seq$(L(A)) = L($Seq$(A))$.

BSP element: type	$\longrightarrow$	**local / sequential element**
$\epsilon : \Sigma^*$	$\xrightarrow{@i}$	ϵ
$a : \Sigma^*$	$\xrightarrow{@i}$	(a,i)
$abaa : \Sigma^*$	$\xrightarrow{@i}$	$(a,i)(b,i)(a,i)(a,i)$
$\vec{\epsilon} =< \epsilon, \epsilon, \epsilon, \epsilon >: (\Sigma^*)^p$	$\xrightarrow{\mathsf{Seq}}$	ϵ
$\vec{v}_1 =< aba, b, bbb, a >: (\Sigma^*)^p$	$\xrightarrow{\mathsf{Seq}}$	$(a,0)(b,0)(a,0)(b,1)(b,2)(b,2)(b,2)(a,3)$
$\vec{v}_2 =< a, \epsilon, bbb, \epsilon >: (\Sigma^*)^p$	$\xrightarrow{\mathsf{Seq}}$	$(a,0)(b,2)(b,2)(b,2)$
$\vec{\epsilon} =< \epsilon, \epsilon, \epsilon, \epsilon >: (\Sigma^*)^p$	$\xrightarrow{\mathsf{Seq}}$	ϵ
$\epsilon : ((\Sigma^*)^p)^*$	$\xrightarrow{\mathsf{Seq}}$	ϵ
$\vec{\epsilon} =< \epsilon, \epsilon, \epsilon, \epsilon >: ((\Sigma^*)^p)^*$	$\xrightarrow{\mathsf{Seq}}$	$(\epsilon;) \quad = \quad ;$
$\vec{v}_2\ \vec{\epsilon} : ((\Sigma^*)^p)^*$	$\xrightarrow{\mathsf{Seq}}$	$(a,0)(b,2)(b,2)(b,2);;$
$\vec{\epsilon}\ \vec{v}_2 : ((\Sigma^*)^p)^*$	$\xrightarrow{\mathsf{Seq}}$	$;(a,0)(b,2)(b,2)(b,2);$
$< \epsilon, a, \epsilon, a >< b, b, b, b >: ((\Sigma^*)^p)^*$	$\xrightarrow{\mathsf{Seq}}$	$(a,1)(a,3);(b,0)(b,1)(b,2)(b,3);$

Parallelization

Lemma 1. *Parallelization is the left-inverse of sequentialization on word-vectors* $(\Sigma^*)^p$*:*

$$\mathit{Par}(\mathit{Seq}(\vec{v})) = \vec{v}.$$

- To parallelize localized letters Par : $(\Sigma \times [p]) \to (\Sigma^*)^p$.
- To parallelize semicolon-free words Par : $(\Sigma \times [p])^* \to (\Sigma^*)^p$.
- To parallelize localized words with semicolons Par : $((\Sigma \times [p]) \cup \{;\})^* \to ((\Sigma^*)^p)^*$.

local / sequential element: type	$\longrightarrow$	**vector/BSP element: type**
$(a,1) : \Sigma \times [p]$	$\xrightarrow{\text{Par}}$	$< \epsilon, a, \epsilon, \epsilon >: (\Sigma^*)^p$
$\epsilon : (\Sigma \times [p])^*$	$\xrightarrow{\text{Par}}$	$< \epsilon, \epsilon, \epsilon, \epsilon >: (\Sigma^*)^p$
$(a,1)(b,3)(a,1) : (\Sigma \times [p])^*$	$\xrightarrow{\text{Par}}$	$< \epsilon, aa, \epsilon, b >: (\Sigma^*)^p$
$(a,0)(b,0)(a,0)(b,1)(b,2)(b,2)(b,2)(a,3)$	$\xrightarrow{\text{Par}}$	$< aba, b, bbb, a >: (\Sigma^*)^p$
$(a,0)(b,0)(b,2)(a,3);(a,0)(b,1)(b,2)(b,2);$	$\xrightarrow{\text{Par}}$	$< ab, \epsilon, b, a >< a, b, bb, \epsilon >: ((\Sigma^*)^p)^*$

Definition 6. $\Sigma_{p;} = (((\Sigma \times [p])^*);)$

$\Sigma_{p;}^* =$ sequential localized words, without non-empty semicolon-free words.

Definition 7. *For* $w \in ((\Sigma\times[p])^*)\cup\{;\}$, w' **over-sychronizes** w $(w \leq_; w')$ *if* w' *is* w *with interleaved semicolons.*
Lift the same definition to languages and automata.

Theorem 1. $\forall$ *automaton* A *on* $(\Sigma\times[p])\cup\{;\}$ $\exists$ *DFA* $A' \geq_; A$, *such that* $L(\mathit{Par}(A)) = \mathit{Par}(L(A'))$.

Bulk-synchronous regular expressions

A BSP regular expression is an expression R from the following grammar:

$$R ::= \emptyset \mid \epsilon \mid < r^0, \ldots, r^{p-1} > \mid R; R \mid R* \mid R + R$$

where r^i is any (scalar) regular expression.

R	$L(R)$
$\emptyset$	$\{ \}$
ϵ	$\{\epsilon\}$
$< r^0, \ldots, r^{p-1} >$	$L(r^0) \times \ldots \times L(r^{p-1})$
$R_1; R_2$	$L(R_1)L(R_2)$
R^*	$L(R)^*$
$R_1 + R_2$	$L(R_1) \cup L(R_2)$

Theorem 2. *For* $R \in$ *BSPRE* $\exists$ *a BSP automaton* A_R *such that* $L(A_R) = L(R)$.

Theorem 3. *For* A *a BSP automaton* $\exists$ $R_A \in$ *BSPRE such that* $L(R_A) = L(A)$.

Proposition 5. *If A is a deterministic BSP automaton on Σ then there exists a sequential automaton Min$($Seq$(A))$ that accepts the same Seq$(L(A))$ and is of minimal size.*

Figure 4: Automaton $\vec{A}_a$

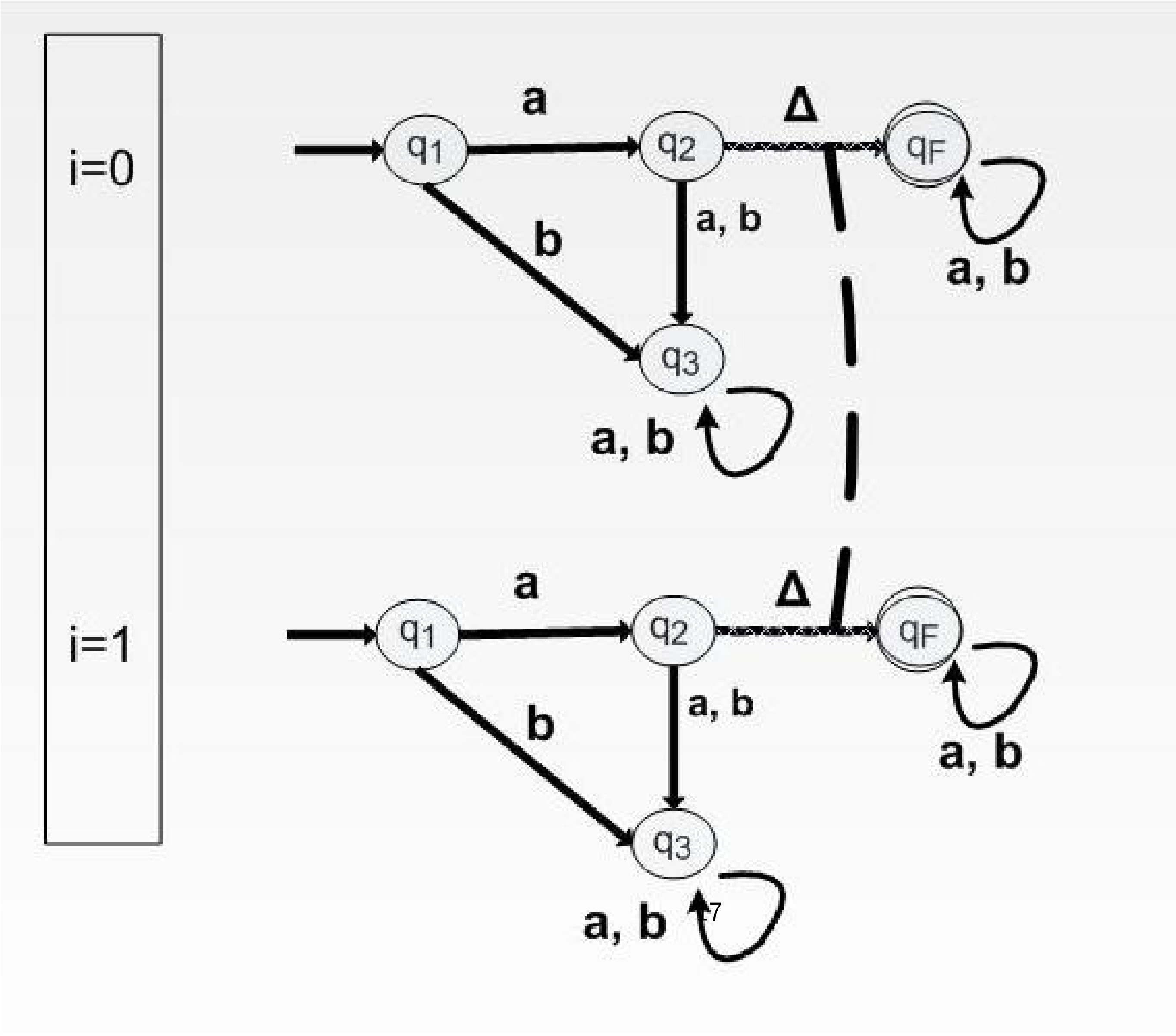


Figure 5: Locally minimal automaton Min(A_a)

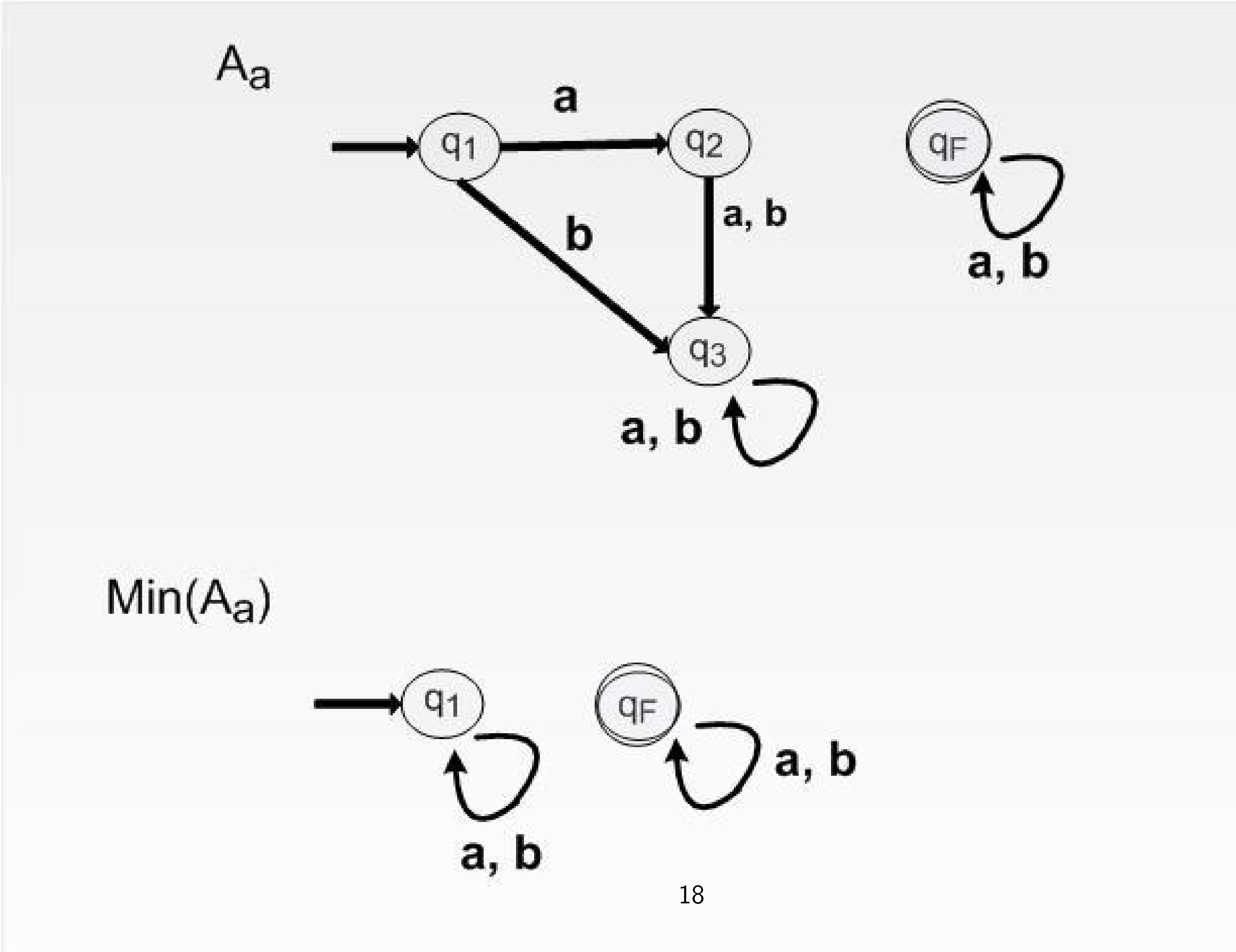
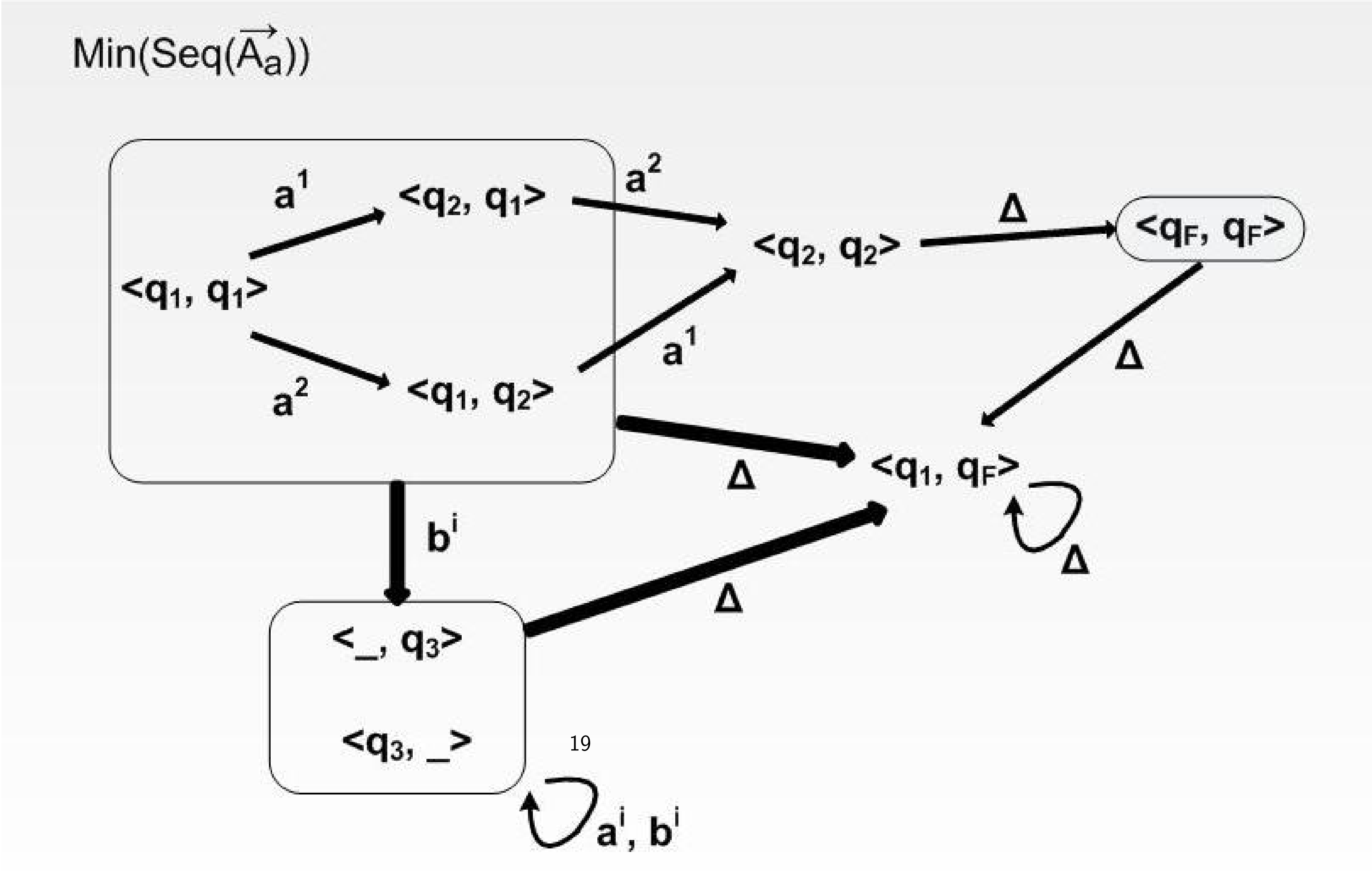


Figure 6: Sequential minimization of BSP automaton $\vec{A}_a$



Cost-model

Definition 8. *A* factorization function *on* Σ *words is a function* $\Phi : \Sigma^* \to (\Sigma^+)^*$ *such that*

$$\Phi(\epsilon) = \epsilon$$
$$|w| > 0 \;\Rightarrow\; |\Phi(w)| > 0$$
$$\Phi(w) = w_1, w_2 \ldots, w_n \;\Rightarrow\; w_1 w_2 \ldots w_n = w$$

Definition 9. *Given a factorization function* Φ *on* Σ *words, a* distribution function *based on* Φ *is a* $D_\Phi : \Sigma^* \to (\Sigma_{p;})^*$ *such that*

$$D_\Phi(\epsilon) = \epsilon$$
$$\Phi(w) = w_1, w_2 \ldots, w_n \;\Rightarrow\; D_\Phi(w) = w'_1; w'_2; \ldots w'_n;$$
$$w_t = a_1 \ldots a_k \;\Rightarrow\; w'_t = (a_1, i_1) \ldots (a_k, i_k)$$
$$i_1, \ldots, i_k \in [p]$$

Definition 10. *Let* $\vec{v} \in (\Sigma^*)^p$ *be a word vector. Its* BSP cost *cost*$(\vec{v}) = \max_i \left|v^i\right|$ *is the length of its longest element. Define also* $l \in \mathbf{N}^+$, *the* barrier synchronization cost *constant. For a BSP word* $w = \vec{v}_1 \ldots \vec{v}_S \in ((\Sigma^*)^p)^*$, *its BSP cost is*

$$\textit{cost}(w) = \Sigma_{t=1}^{S}(\textit{cost}(\vec{v}_t) + l) = Sl + \Sigma_{t=1}^{S}\textit{cost}(\vec{v}_t).$$

Definition 11. *For a given distribution function* D_Φ *of factorization* Φ, *the BSP cost of a sequential word* $w \in \Sigma^*$ *with respect to* D_Φ *is defined as the BSP cost of the parallelization of its distribution:*

$$\textit{cost}_{D_\Phi}(w) = \textit{cost}(\textit{Par}(D_\Phi(w)))$$

Problem 1. BSP-PARALLELIZE-WORDWISE
Input: *A regular language L given by a regular expression r or DFA A.*
Goal: *Find a distribution D_Φ and BSP automaton A_D such that $L(A) = \mathit{Par}(D_\Phi(L))$ and $|A_D| \in O(|A|)$.*
Subject to: $\forall w \in \Sigma^*$. *$\mathit{cost}_{D_\Phi}(w)$ is minimal over* $\{(\Phi, D_\Phi, A_D) \mid L(A) = \mathit{Par}(D_\Phi(L))\}$.

Problem 2. BSP-PARALLELIZE
Input: *A regular language* L *given by a regular expression or DFA.*
Goal: *Find a distribution* D_Φ *and BSP automaton* A_D *such that* $L(A) = \textit{Par}(D_\Phi(L))$ *and* $|A_D| \in O(|A|)$.
Subject to: $T_{D_\Phi}(n) = \max\{\textit{cost}_{D_\Phi}(w) \mid |w| = n\}$ *is minimal over* $\{(\Phi, D_\Phi, A_D) \mid L(A) = \textit{Par}(D_\Phi(L))\}$, *for all* $n \geq 0$.

Definition 12. *Let L be a regular language and (Φ, D_Φ, A_D) a factorization, distribution and BSP automaton for L i.e. Par$(D_\Phi(L))$. The parallel* speedup *obtained by (Φ, D_Φ, A_D) on a given word size n is the ratio*

$$\textit{speedup}(\Phi, D_\Phi, A_D, n) = \min\{n/\textit{cost}_{D_\Phi}(w) \mid |w| = n\}$$

$$L_1 = L(a^*),\ L_2 = L(a^*b^*),\ L_3 = L((a+b)^*bbb(a+b)^*)$$

Parallel recognition of L_1, L_2, L_3 ...

Problem 3. *OPEN PROBLEM: does every instance of BSP-PARALLELIZE have a one-superstep solution ?*

Answer "yes" if the number of states in the BSP automaton solution allowed to grow exponentially. Construction for showing this is very different from that of our above examples.

Proposition 6. *Every regular language* L *of regular expression* r *has a one-superstep parallelization* $(\Phi_1, D_{\div p}, A)$ *that can be constructed in time exponential in* $|r|$ *and such that* $|A|$ *is also exponential in* $|r|$.

Intensional notations for BSP automata

Write locations numbers $i \in [p]$ in binary.
Encode sets of locations with binary regular expressions.
e.g. $(0+1)^*1 =$ odd-rank locations,
$0(0+1)(0+1) =$ four first locations when $p = 8$ etc.
Define

$$\vec{r} ::= [\text{pid} \in b]\, r \;\mid\; \vec{r} + \vec{r}$$

where r is a normal reg.exp.
$[\text{pid} \in b]\, r$ is the vector of regular expressions s.t. value r at locations $i \in L(b)$ and ϵ elsewhere.
$\vec{r} + \vec{r} =$ pointwise (location by location) sum of regular expressions.

Intensional BSP regular expressions:

$$R ::= \emptyset \mid \epsilon \mid \vec{r} \mid R;R \mid R* \mid R + R.$$

Assume a BSPRE of the form

$$R = \vec{r} = [\mathsf{pid} \in b]\, r_1$$

and a location i that wishes to communication with a subset of locations.
Process i computes $b' =$ complement of b and also

$$r_1^+ = r_1 \cap (a+b)(a+b)^*$$

The required set of locations is

$$([\mathsf{pid} \in b']\,(a+b)^+) + [\mathsf{pid} \in b]\, r_1^+.$$

The automates the conversion of **get** operations into more efficient **put** operations.

Conclusions and future work

BSP automata and BSP langages preserve all the classical closure properties: non-determinism, ϵ-transitions and determinization, but break the classical properties of minimization. The interaction between state-minimization and BSP cost optimization remains to be understood.

Future work

1. BSP regular grammars and generalization to BSP context-free languages
2. parallel text processing and parsing,
3. pattern matching and data structure parallelization (tries etc).