Shift-Reduce Parsers for Transition Networks

Luca Breveglieri Stefano Crespi Reghizzi Angelo Morzenti

Politecnico di Milano

LATA 2014 - 10-14 March - Madrid

Problem statement and research objectives

On the status of the *LR* or bottom-up syntax analysis

LR (bottom-up) is an established methodology for syntax analysis.

Theory is mostly developed for grammars in Backus-Naur Form (*BNF*).

There are automated tools for compiler design that use it (e.g., Bison).

Extended *BNF* (*EBNF*) grammars (rules contain regular expressions) are widely used for specifying technical languages of all sorts.

Usually *EBNF* rules are reduced to *BNF* ones and then analyzed !

Objectives of the present research work

Develop an Extended *LR* (*ELR*) methodology to generalize the *LR* one.

Applicable to *EBNF* grammars represented as Transition Networks (*TN*).

Table of contents

State of the art in the *LR* syntax analysis

Classical $LR(k)$ theory for *BNF* grammars is well developed.

Compiler design tools for $LR(k)$ parsers exist (e.g., Bison).

EBNF grammars are popular for describing technical languages (e.g., syntax charts), but then little used to obtain the parser.

More recently attention has focused on representing an *EBNF* grammar in the equivalent form of a Transition Network (*TN*).

For *EBNF* grammars (or their *TN*'s) there are many attempts to apply *LR* analysis, but no simple and standard solution:

- regular expressions are annotated and manipulated directly $\Rightarrow$ this approach is somewhat distant from practical parsing
- *EBNF* is turned into *BNF* $\Rightarrow$ grammar is obscured and larger
- *EBNF* rules are processed directly $\Rightarrow$ parser is complicated due to the reduction move (at least in the current solutions)

There are also incomplete or even wrong solutions proposed.

EBNF grammar and recursive transition network

An *EBNF* grammar may have a regular expr. in a rule right part:

$$A \to \left(a \mid b\, B^* \, c \right)^+ \qquad \text{in general } A \to r.e.\left(a, b, \ldots, A, B \ldots \right)$$

and such an extended rule is interpreted as ∞-many *BNF* rules:

$$A \to a \mid b\, c \mid b\, B\, c \mid b\, B\, B\, c \mid \ldots \mid a\, b\, c \mid b\, c\, a \mid \ldots$$

Thus stipulate *EBNF* may have only one rule per each nonterminal.

Represent a grammar by a Transition Network (*TN*): a set of *DFA*'s.
Each *DFA* is equivalent to the regular expression in a rule right part.
The *TN* has a single *DFA* (called *machine*) per each nonterminal.
A transition with a nonterminal label is a *call site* for another machine.
So any machine can invoke any other one recursively (even itself).

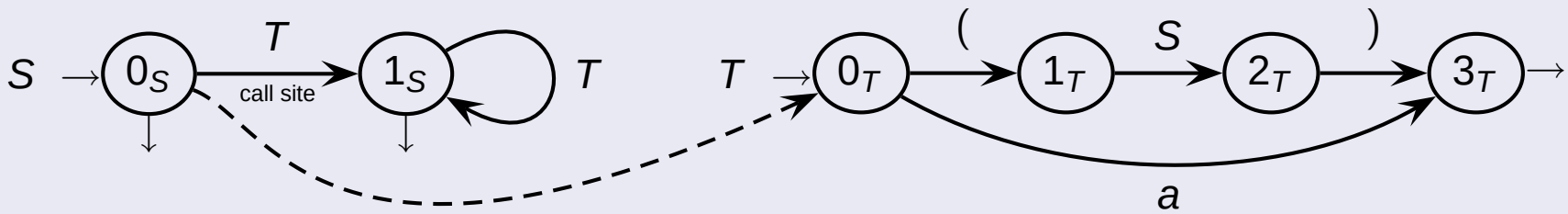
Sample transition network

EBNF grammar G of a simple language of expressions (axiom S)

$$\Sigma = \{ a, \text{`(`}, \text{`)`} \} \qquad V = \{ S, T \}$$

$$G \begin{cases} S \rightarrow T^* \\ T \rightarrow \text{`(`}\, S\, \text{`)`} \mid a \end{cases}$$

Transition network of G with a machine for S (axiomatic) and one for T



A machine of the *TN* is a *DFA* over the alphabet union of Σ and V.

But the initial state of a machine must not have any ingoing arcs.

A machine may be in the minimal form (except the initial state).

BNF: machine modeled as tree with no loops or confluent paths.

Right linearized grammar of a *TN*

A right linearized grammar is a piecewise right linear grammar.

Rules are parted into right linear groups, which call one another.

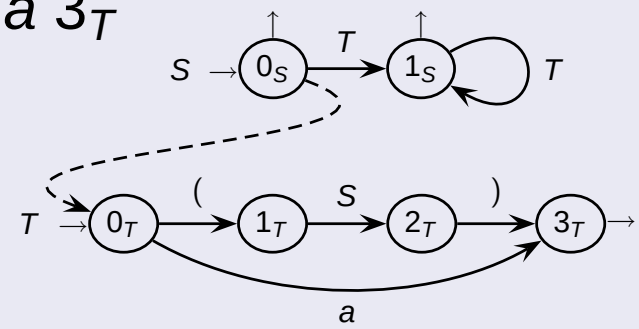
Each such group has only terminal or right recursive linear rules.

So a right linearized grammar maps a *TN*[1], but is purely *BNF*.

It is a useful theoretical representation, yet unfit for parsing.

Right linearized grammar G_{RL} of the sample *TN* (axiom 0_S)

$$G_{RL}\begin{cases} 0_S \rightarrow 0_T\ 1_S \mid \varepsilon \\ 1_S \rightarrow 0_T\ 1_S \mid \varepsilon \end{cases} \qquad \begin{aligned} 0_T &\rightarrow \text{`(' } 1_T \mid a\ 3_T \\ 1_T &\rightarrow 0_S\ 2_T \\ 2_T &\rightarrow \text{`)' } 3_T \\ 3_T &\rightarrow \varepsilon \end{aligned}$$



[1] Heilbrunner defined G_{RL} ('79), unrelated to *TN*.

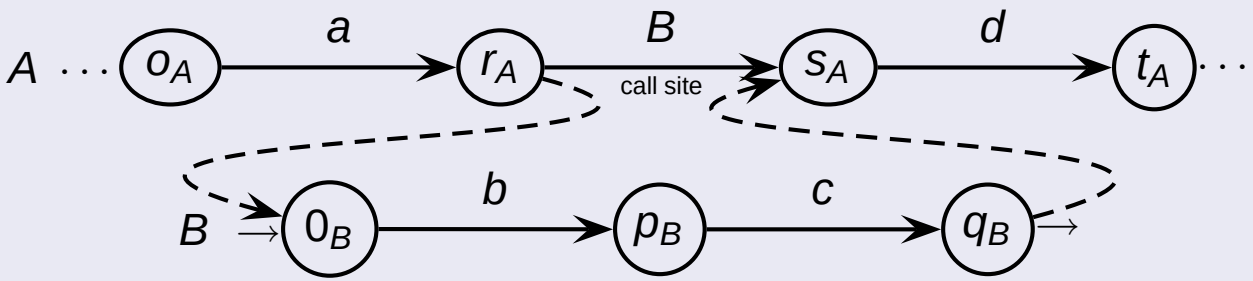
Item structure and its meaning

An *item* is a pair $\langle p, \pi \rangle$ (called *state*, *look-ahead*), such that:

- p is a state of (a machine of) the transition network
- $\pi \subseteq \left(\Sigma \cup \{ \dashv \} \right)$ is a subset of terminals ($\pi \neq \emptyset$)

An item represents an analysis point reached by the parser:

- a machine (i.e., a rule) matches the input as far as state p
- π contains the terminals expected after the machine ends



If the string to parse is $\ldots a\, b\, c\, d \ldots$, then the item means that machine B (called at site $r_A \xrightarrow{B} s_A$) has matched symbol b and now is at state p_B, and that when it ends, symbol d is expected.

Item shift: evolving an existing item

Suppose $\langle p, \pi \rangle$ is an existing item, with state p and look-ahead π.

Define an item *shift* (partial) function:

$$\mathit{shift} : \text{set of all items} \times \left(\Sigma \cup V \right) \rightarrow \text{set of all items}$$

which works on an item as follows:

$$\mathit{shift}\left(\langle p, \pi \rangle, X \right) = \langle q, \pi \rangle \qquad \text{if arc } p \xrightarrow{X} q \text{ is in the } TN$$

where X is any grammar symbol (terminal or nonterminal).

Using the *TN*, the *shift* function matches an item and a grammar symbol, and goes to the next item on the same machine.

Since the machine of the *TN* remains the same for the shifted item, the *shift* function does not change the item look-ahead.

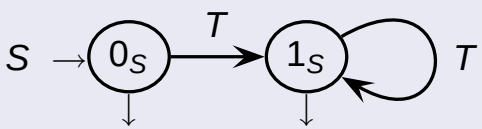
Item closure: creating a new look-ahead

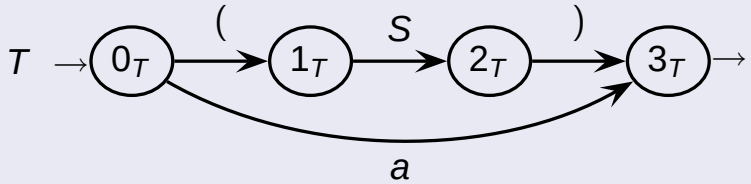
Closure of a (non-empty) set I of items

$$closure\,(I) = I \;\cup \left\{ \langle\, 0_B,\, \pi \,\rangle \;\middle|\; \begin{array}{l} \exists\, item\ \langle\, r,\, \rho \,\rangle \in closure\,(I) \\ \textbf{and}\ \exists\, arc\ \left(r \xrightarrow{B} s \right) \in TN \\ \textbf{and}\ \pi = initials\left(L\,(s) \cdot \rho \right) \end{array} \right\}$$

Closure examples (sample TN of G)

set I of items	new items added to I by closure
$\langle\, 1_T,\, \{\, \ldots \,\} \rangle$	$\langle\, 0_S,\, \{\, \text{`)'} \,\} \,\rangle$ $\langle\, 0_T,\, \{\, \text{`('}, a, \text{`)'} \,\} \rangle$





Closure may create items with initial TN state and new look-ahead.

Macro-state (m-state) and parser pilot

A *macro-state* (m-state) is a non-empty set of items, which represent possible analysis points reached by the parser.

The *pilot* of a *TN* (grammar) is a finite directed graph, where:

- the nodes are the m-states reachable by the parser
- the arcs connect m-states through grammar symbols

Extend the item shift function *shift* to the macro-states:

$$\boxed{\begin{array}{c}\langle p, \pi \rangle \\ \langle r, \rho \rangle \\ \dots\end{array}} = \boxed{\begin{array}{c|c} p & \pi \\ r & \rho \\ \dots & \dots \end{array}}$$

m-state I graphic form

$$shift\ (I, X) = \boxed{\begin{array}{l} shift\ (\langle p, \pi \rangle, X) \\ shift\ (\langle r, \rho \rangle, X) \\ shift\ (\dots, X) \end{array}} = \boxed{\begin{array}{c} \langle q, \pi \rangle \\ \langle s, \rho \rangle \\ \dots \end{array}}$$

For *BNF* grammars, items are often denoted as marked rules:

$$B \to \beta \bullet \gamma, \pi \qquad \Leftrightarrow \qquad \langle p_B, \pi \rangle$$

String β is the path from state 0_B to state p_B in the machine B.

Algorithm for building the pilot graph

pilot *DFA* $\mathcal{P} = (\Sigma \cup V,\ R,\ \vartheta,\ I_0)$

m-state set $R = \{ I_0, I_1, \ldots \}$

transition function $\vartheta \colon R \times (\Sigma \cup V) \to R$

Pilot graph algorithm - computes R and ϑ of $\mathcal{P}$

$R := closure \left(\left\{ \langle 0_S, \{ \dashv \} \rangle \right\} \right)$ - - initial m-state I_0

repeat

  for each m-s. $I \in R$ **and sym.** $X \in \Sigma \cup V$ **do**

    $I' := closure \left(shift\ (I, X) \right)$

    add m-state I' **to the m-state set** R

    add arc $I \xrightarrow{X} I'$ **to the transition function** ϑ

  end for

until R **does not change any more**

Algorithm for building the pilot graph

pilot *DFA* $\mathcal{P} = (\Sigma \cup V,\ R,\ \vartheta,\ I_0)$

m-state set $R = \{ I_0, I_1, \ldots \}$

transition function $\vartheta \colon R \times (\Sigma \cup V) \to R$

Pilot graph algorithm - computes R and ϑ of $\mathcal{P}$

$R := closure \left(\left\{ \langle 0_S, \{ \dashv \} \rangle \right\} \right)$ - - initial m-state I_0

repeat

  for each m-s. $I \in R$ **and sym.** $X \in \Sigma \cup V$ **do**

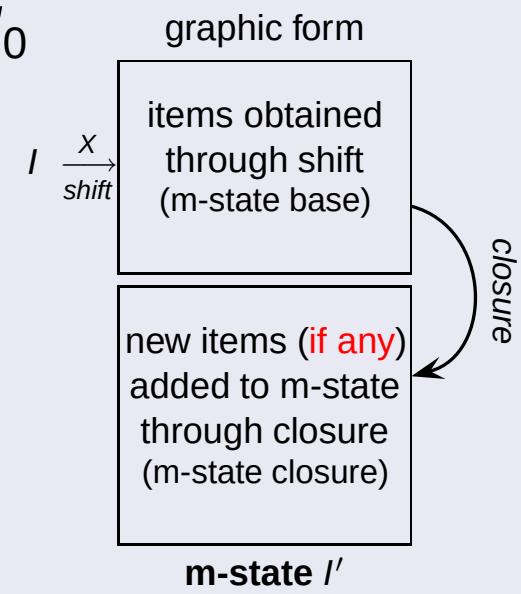
    $I' := closure \left(shift \ (I, X) \right)$

    add m-state I' **to the m-state set** R

    add arc $I \xrightarrow{X} I'$ **to the transition function** ϑ

  end for

until R **does not change any more**

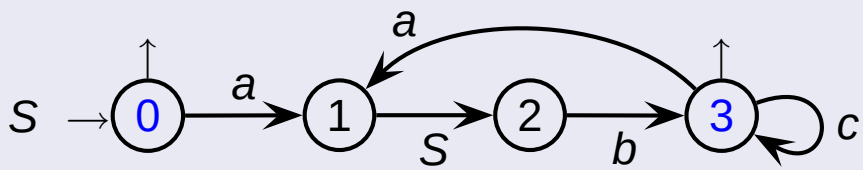


Sample pilot graph

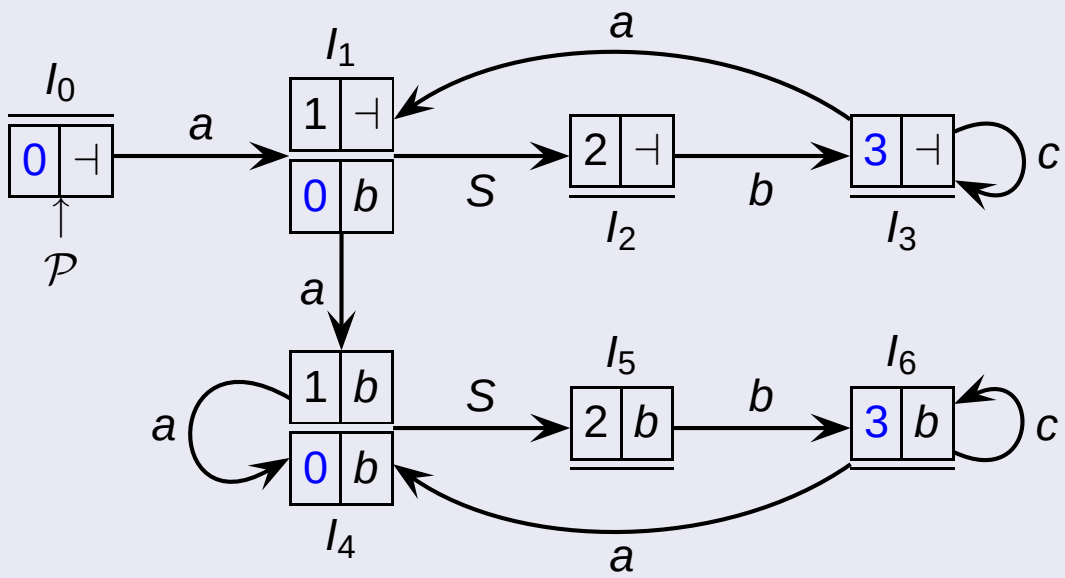
EBNF grammar of a (sort of) Dyck language

$$S \to \left(a\, S\, b\, c^* \right)^*$$

$$\varepsilon,\ a\,b,\ a\,a\,b\,b,\ a\,b\,c\,a\,b,\ \ldots$$



Pilot $\mathcal{P}$ of the *EBNF* grammar above

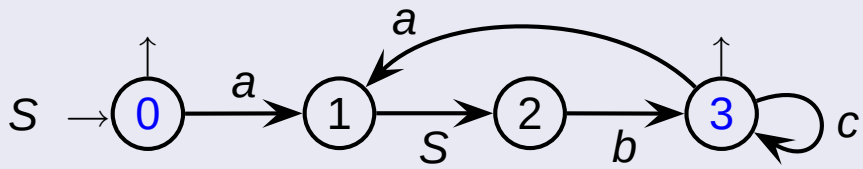


Sample pilot graph

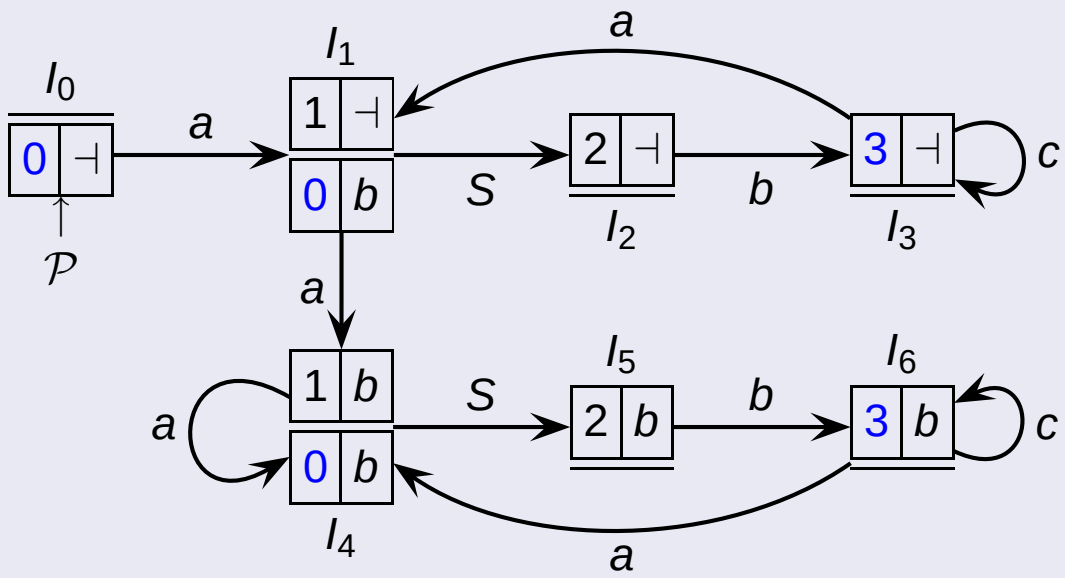
EBNF grammar of a (sort of) Dyck language

$$S \rightarrow \left(a\, S\, b\, c^* \right)^*$$

$\varepsilon,\ ab,\ aabb,\ abcab,\ \ldots$



Pilot $\mathcal{P}$ of the *EBNF* grammar above



I_0 is initial (by definition it is only closure); I_1, I_4 have both a base and a closure; I_3, I_6 are entered with two different labels (a classical pilot for *BNF* cannot do so)

Conflicts in the parser pilot

The pilot is meant to drive the parser deterministically.

Yet the pilot may fail to do so if it has internal conflicts.

The conflicts possible for *BNF* grammars are known.

Suppose I is the m-state currently under examination.

Shift-Reduce conflict (*SR*)

$\exists$ item $\langle p, \pi \rangle \in I$ s.t. state p is final and $\exists$ arc $I \xrightarrow{e} I'$: $e \in \pi$

Reduce-Reduce conflict (*RR*)

$\exists$ items $\langle p, \pi \rangle, \langle r, \rho \rangle \in I$ s.t. states p, r are final: $\pi \cap \rho \neq \emptyset$

But the pilot of a generic *TN* (with loops and confluent paths), i.e., that of an *EBNF* grammar, deserves a closer scrutiny ...

Multiple transition with convergence and conflict

δ: *TN* transition function

ϑ: pilot $\mathcal{P}$ transition function

Multiple transition / convergence / conflict - a new conflict type

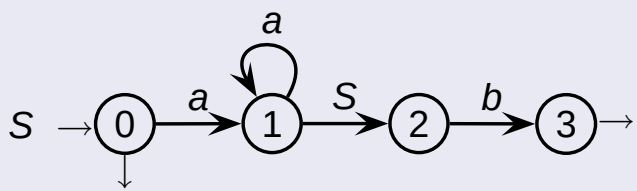
an m-state *I* has a multiple transition if it has items $\langle p, \pi \rangle$, $\langle r, \rho \rangle$ s.t. for a grammar symbol *X* both next states $\delta(p, X)$, $\delta(r, X)$ are defined

a multiple transition $\vartheta(I, X)$ is convergent if it holds: $\delta(p, X) = \delta(r, X)$

a convergent transition is conflicting if look-aheads overlap: $\pi \cap \rho \neq \emptyset$

Case of a small *EBNF* grammar $\{ \varepsilon, a^n b^m \mid n \geq m \geq 1 \}$ is det. !

$$S \rightarrow a^+ S\, b \mid \varepsilon$$



Multiple transition with convergence and conflict

δ: *TN* transition function

ϑ: pilot $\mathcal{P}$ transition function

Multiple transition / convergence / conflict - a new conflict type

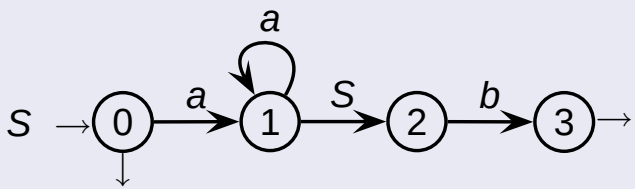
an m-state I has a multiple transition if it has items $\langle p, \pi \rangle$, $\langle r, \rho \rangle$ s.t. for a grammar symbol X both next states $\delta(p, X)$, $\delta(r, X)$ are defined

a multiple transition $\vartheta(I, X)$ is convergent if it holds: $\delta(p, X) = \delta(r, X)$

a convergent transition is conflicting if look-aheads overlap: $\pi \cap \rho \neq \emptyset$

Case of a small *EBNF* grammar $\{ \varepsilon, a^n b^m \mid n \geq m \geq 1 \}$ is det. !

$$S \rightarrow a^+ S\, b \mid \varepsilon$$



0	⊣

Multiple transition with convergence and conflict

δ: *TN* transition function

ϑ: pilot $\mathcal{P}$ transition function

Multiple transition / convergence / conflict - a new conflict type

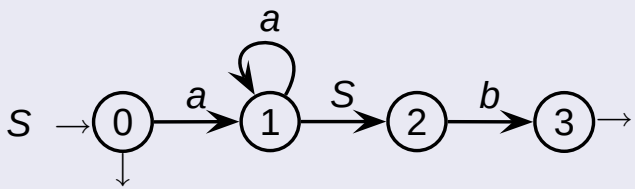
an m-state I has a multiple transition if it has items $\langle p, \pi \rangle$, $\langle r, \rho \rangle$ s.t. for a grammar symbol X both next states $\delta(p, X)$, $\delta(r, X)$ are defined

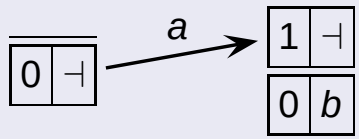
a multiple transition $\vartheta(I, X)$ is convergent if it holds: $\delta(p, X) = \delta(r, X)$

a convergent transition is conflicting if look-aheads overlap: $\pi \cap \rho \neq \emptyset$

Case of a small *EBNF* grammar $\{ \varepsilon, a^n b^m \mid n \geq m \geq 1 \}$ is det. !

$$S \rightarrow a^+ S\, b \mid \varepsilon$$





Multiple transition with convergence and conflict

δ: *TN* transition function

ϑ: pilot $\mathcal{P}$ transition function

Multiple transition / convergence / conflict - a new conflict type

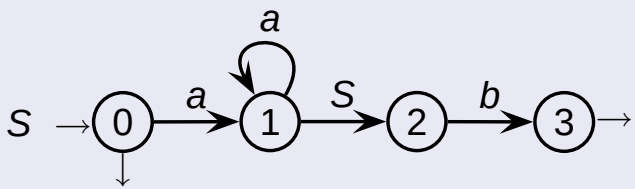
an m-state *I* has a multiple transition if it has items $\langle p, \pi \rangle$, $\langle r, \rho \rangle$ s.t. for a grammar symbol X both next states $\delta(p, X)$, $\delta(r, X)$ are defined

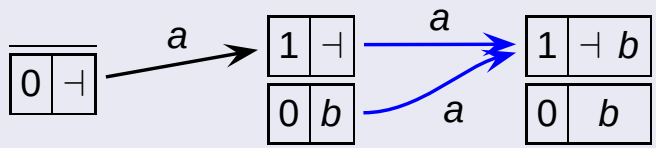
a multiple transition $\vartheta(I, X)$ is convergent if it holds: $\delta(p, X) = \delta(r, X)$

a convergent transition is conflicting if look-aheads overlap: $\pi \cap \rho \neq \emptyset$

Case of a small *EBNF* grammar $\{ \varepsilon, a^n b^m \mid n \geq m \geq 1 \}$ is det. !

$$S \rightarrow a^+ S\, b \mid \varepsilon$$





Multiple transition with convergence and conflict

δ: *TN* transition function

ϑ: pilot $\mathcal{P}$ transition function

Multiple transition / convergence / conflict - a new conflict type

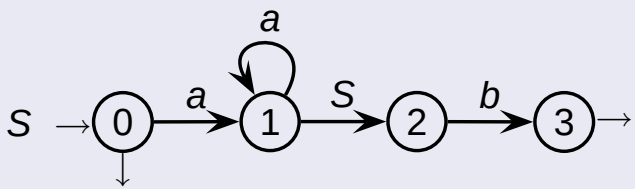
an m-state I has a multiple transition if it has items $\langle p, \pi \rangle$, $\langle r, \rho \rangle$ s.t. for a grammar symbol X both next states $\delta(p, X)$, $\delta(r, X)$ are defined

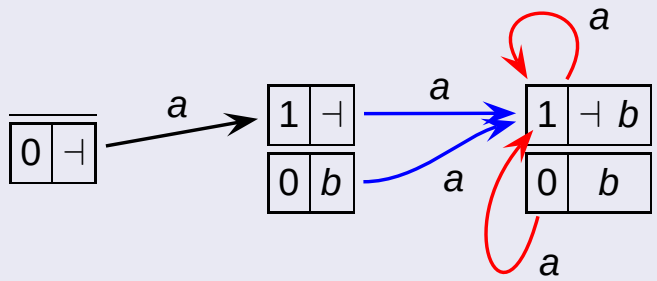
a multiple transition $\vartheta(I, X)$ is convergent if it holds: $\delta(p, X) = \delta(r, X)$

a convergent transition is conflicting if look-aheads overlap: $\pi \cap \rho \neq \emptyset$

Case of a small *EBNF* grammar $\{ \varepsilon, a^n b^m \mid n \geq m \geq 1 \}$ is det. !

$$S \rightarrow a^+ S\, b \mid \varepsilon$$





Determinism conditions for the parser pilot

(classical) *LR* (1) condition for a *BNF* grammar

no pilot m-state has any conflict of type *SR*

no pilot m-state has any conflict of type *RR*

(new) *ELR* (1) condition for a *TN* (i.e., an *EBNF* grammar)

no pilot m-state has any conflicts of type *SR* or *RR*

no pilot transition has any *Convergence* conflict

The *ELR* (1) condition reduces to the *LR* (1) one for *BNF*, since:

- *TN* is modeled as tree forest (no loops or confluent paths)
- pilot has no convergence, so has no conv. conflicts either

In the practical *EBNF* grammars, convergence may be frequent.

Relation between *ELR* cond. and (classical) *LR* cond.

Let T be any transition network that represents an *EBNF* grammar G, and let G_{RL} be the right linearized (*BNF*) grammar associated with T.

Main theorem

A network T meets the *ELR* (1) condition if, and only if, the right linearized grammar G_{RL} of net T meets the *LR* (1) condition.

Sketch of the proof (technical details in the paper)

There is a relation between the m-states of the *ELR* pilot $\mathcal{P}$ of T and those of the (classical) *LR* pilot $\mathcal{P}_{RL}$ of G_{RL}, which helps to match their conflicts:

"if" part: an *SR* conflict in $\mathcal{P}_{RL}$, shows as an *SR* one in $\mathcal{P}$; and an *RR* conflict in $\mathcal{P}_{RL}$, shows as an *RR* or a *Conv.* one in $\mathcal{P}$

"only if" part: an *SR* or *RR* conflict in $\mathcal{P}$, shows as an *SR* or *RR* one in $\mathcal{P}_{RL}$, resp.; and a *Conv.* conflict in $\mathcal{P}$, shows as an *RR* one in $\mathcal{P}_{RL}$

About the (classical) *LR* property

Converting *EBNF* into *BNF*

Suppose an *EBNF* grammar is transformed into an equivalent *BNF* one, not necessarily of right linearized (*RL*) type, by substituting recursive nonterminals to the iteration operators (star and cross).

A previous result (Heilbrunner '79)

If such a *BNF* grammar happens to be *LR*, then any *RL* representation of the original *EBNF* grammar is *LR* as well.

Combining with our Main Theorem . . .

Our definition of *LR* for an *EBNF* grammar, based on *TN* (i.e., *RL*), captures at least as much determinism as conversion methods do.

A common transformation from *EBNF* to *BNF*

Replace each iterator (star) by an additional nonterminal.

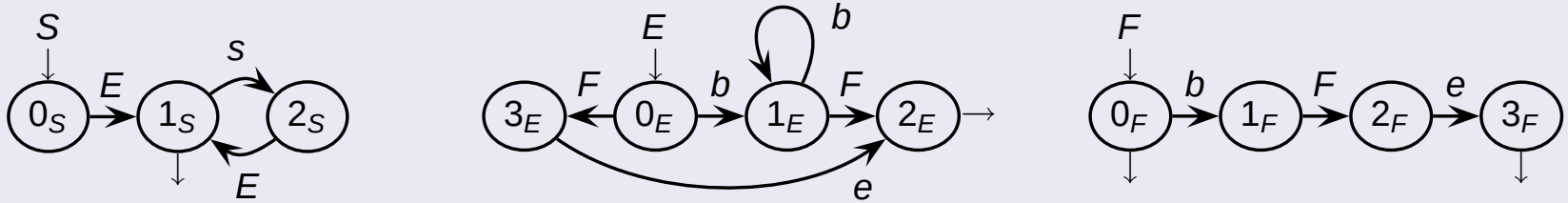
Add a right recursive rule to reproduce the removed iterator.

This transformation may not yield a deterministic grammar.

However it is widely used to design compilers for *EBNF*.

EBNF grammar with conflicts after transforming into *BNF*

$S \to E \left(s\ E \right)^*$ $\qquad E \to b^+ F \mid F\ e$ $\qquad F \to b\ F\ e \mid \varepsilon$



$S \to E\ s\ S \mid E$ $\qquad E \to B\ F \mid F\ e$ $\qquad F \to b\ F\ e \mid \varepsilon$

plus a new nonterm. B and a right recursive rule $B \to b\ B \mid b$

Nevertheless this *EBNF* grammar meets the *ELR* (1) condition !

Parsing algorithm

Parser structure and stack alphabet

The parser is a pushdown automaton and has a compound stack:

symbols: these are the grammar symbols (terminal and non)

sms: these are modified m-states, with a certain annotation

Parser control and move types

The pilot controls the parser and operates two types of move:

shift: follow a pilot transition, and push a *symbol* and a *sms*

reduce: on a final item, if the look-ahead matches the input, pop a series of *symbols* and *sms*'s (called *reduction handle*)

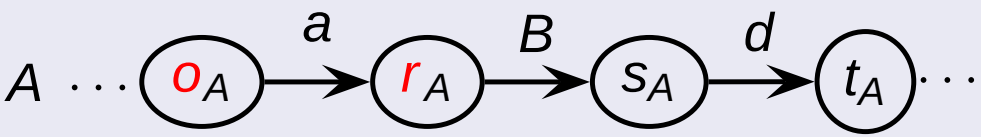
The item annotation is used to identify the reduction handle.

Acceptance is by empty stack (reject at stop or stack not empty).

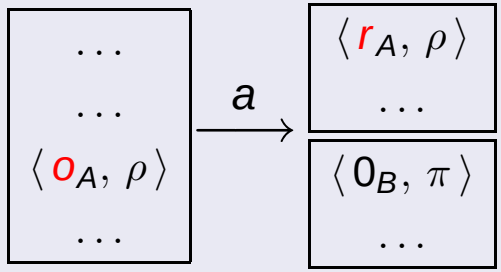
Pushdown stack

TN states and pilot m-states with look-ahead sets

machine A of a TN

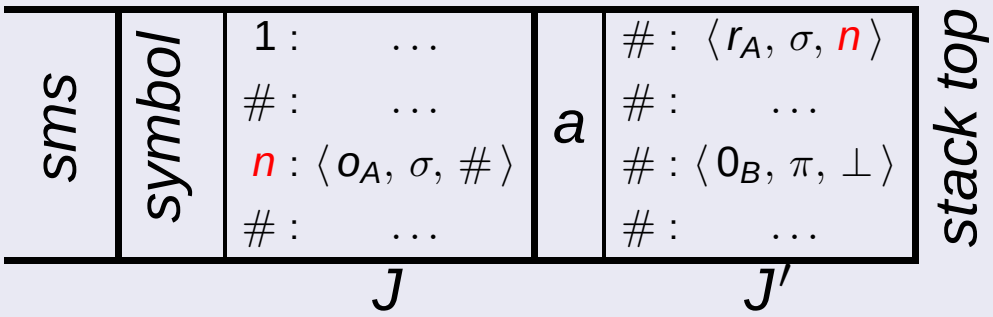


pilot transition $\vartheta(I, a) = I'$



Parser stack with alternation of symbols and *stack m-states* (*sms*)

parser pushdown stack $\ldots J\, a\, J'$

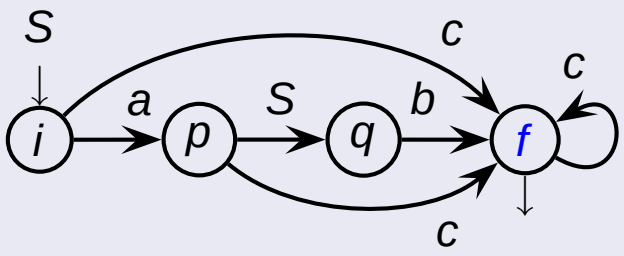


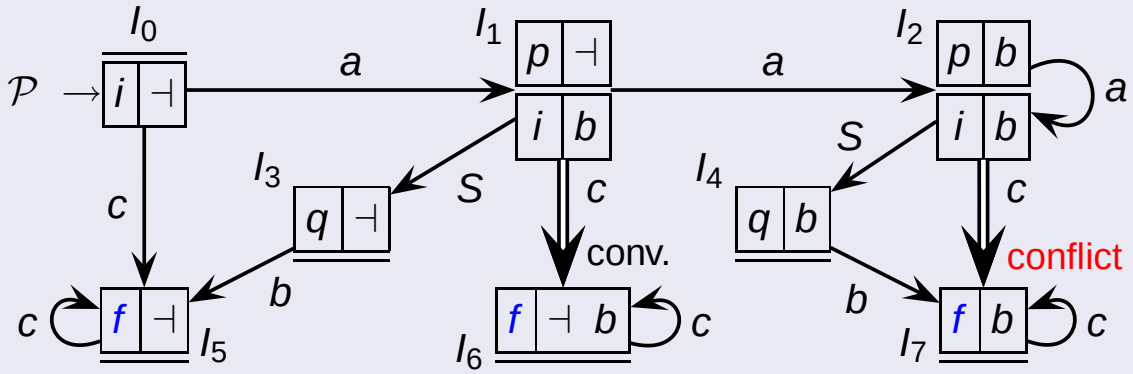
after splitting *convergent* items, the *sms* look-ahead σ may differ from the *m-state* look-ahead ρ of the same item

the stack alphabet is finite !

Simulation of stack operations

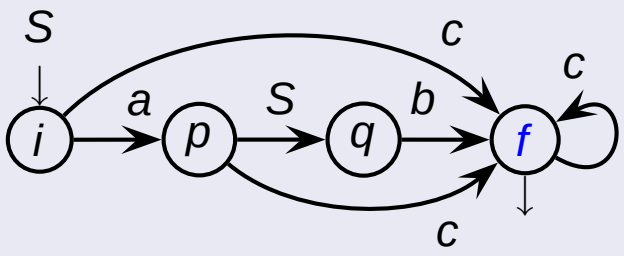
TN and its pilot (the part with a Conv. conflict is not simulated)

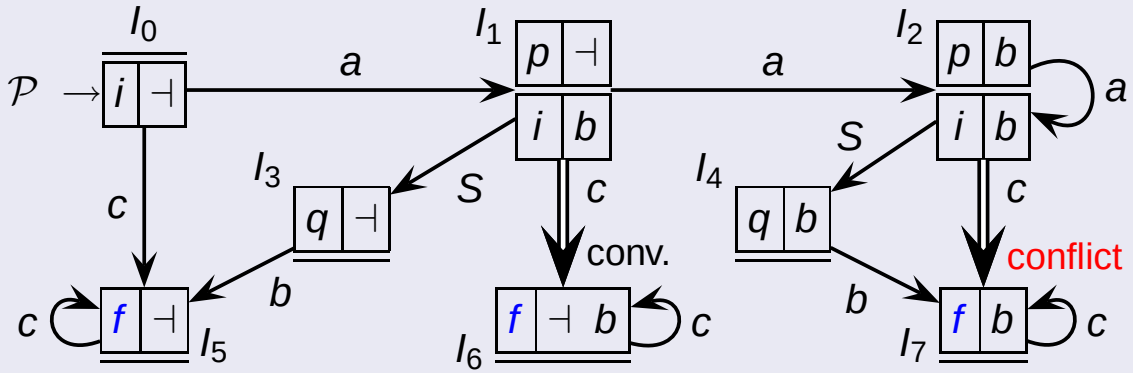




Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

input:

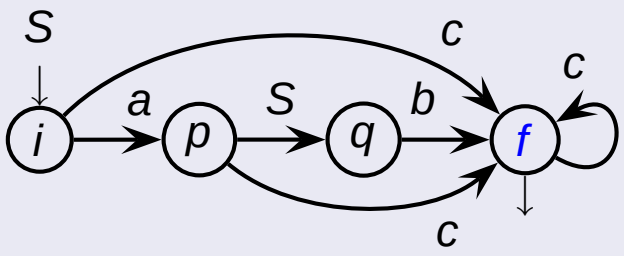
a	c	c	b	⊣

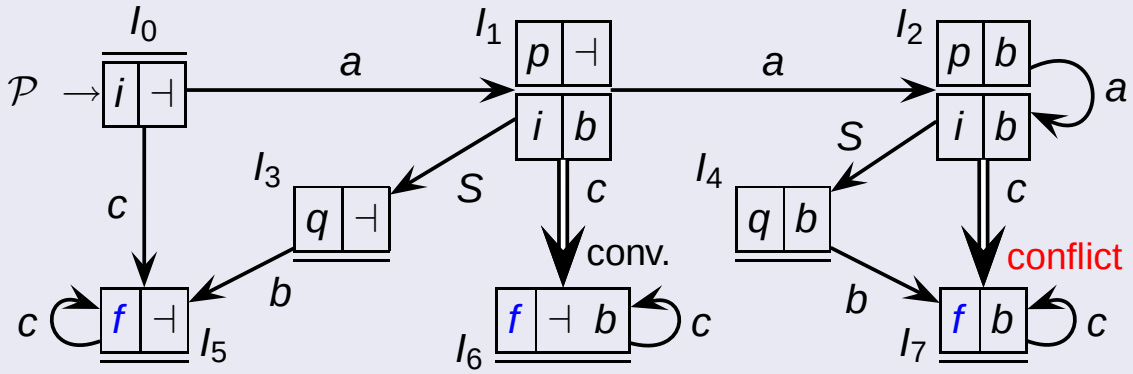
$1\,\langle i, \dashv, \perp \rangle$

J_0

Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

shift *a c c*

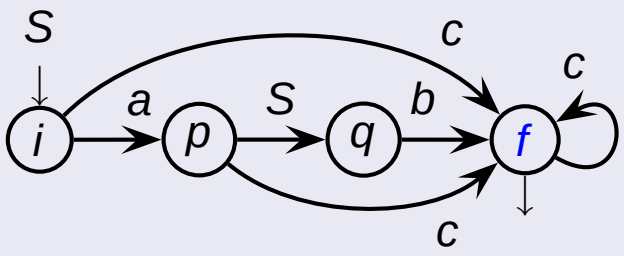
a	*c*	*c*	*b*	⊣

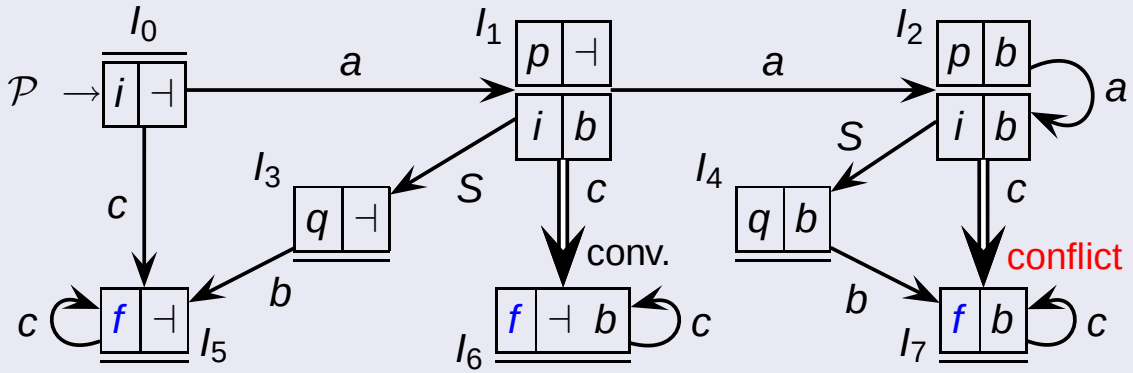
$\boxed{1\ \langle i, \dashv, \perp \rangle}$

J_0

Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

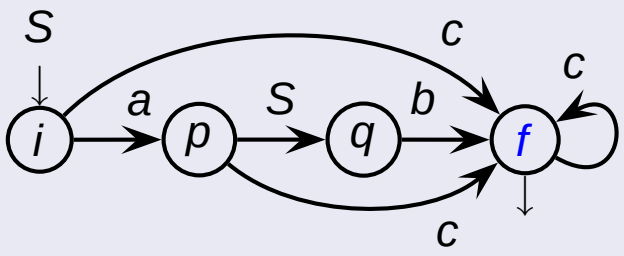
done

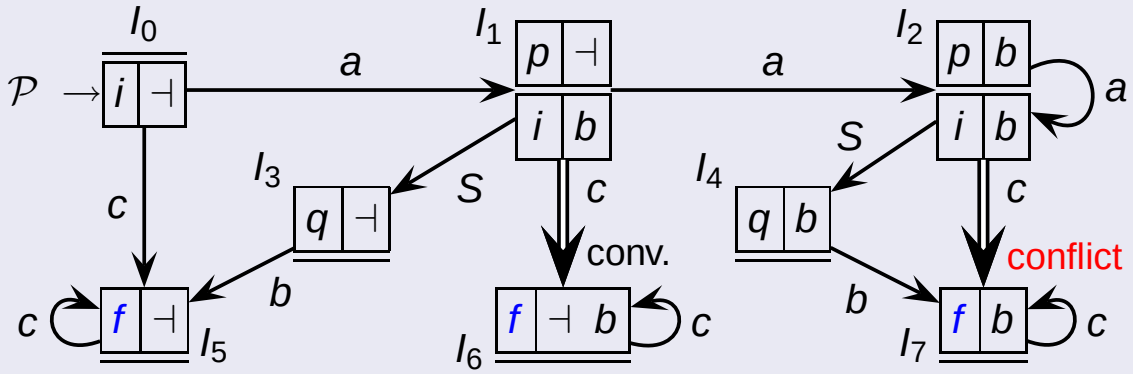
a	c	c	b	⊣

$1\langle i, \dashv, \perp\rangle$	a	$1\langle p, \dashv, 1\rangle$ $2\langle i, b, \perp\rangle$	c	$1\langle f, \dashv, 1\rangle$ $2\langle f, b, 2\rangle$	c	$1\langle f, \dashv, 1\rangle$ $2\langle f, b, 2\rangle$
J_0		J_1		J_6		J_6

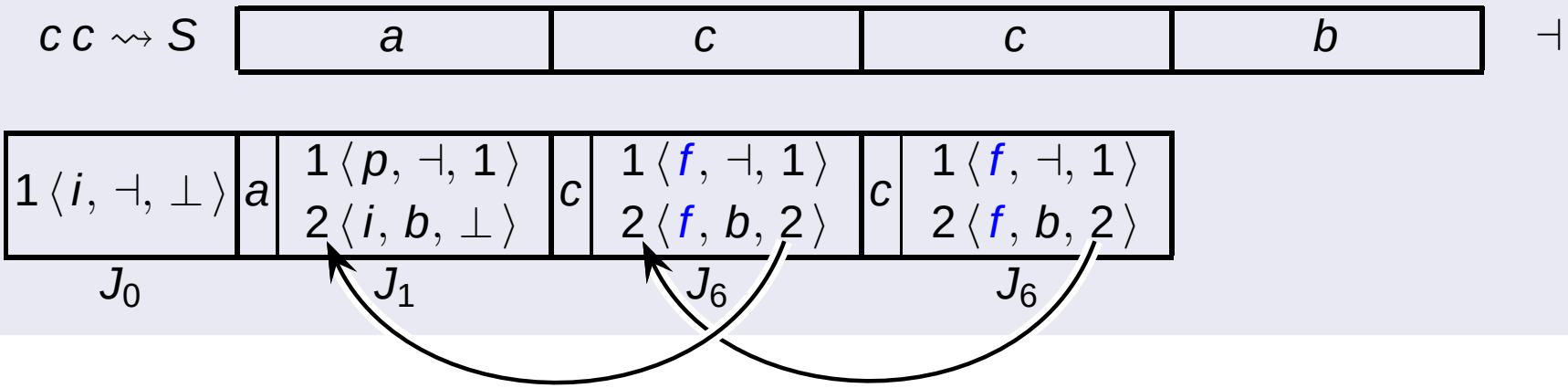
Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)



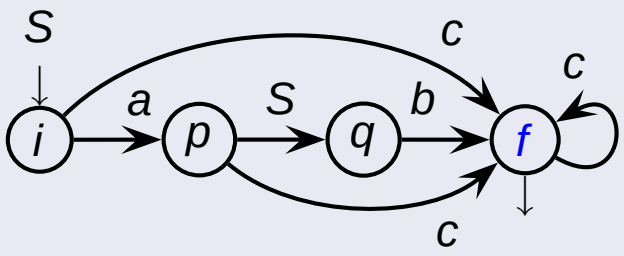


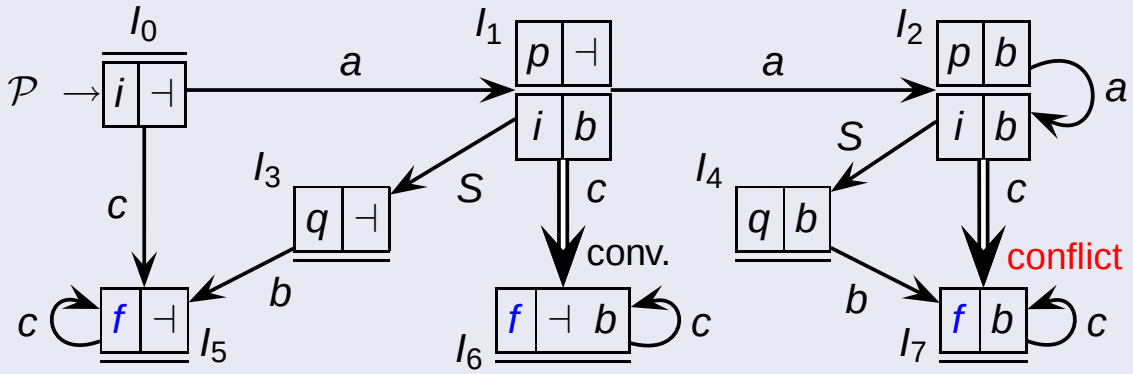
Parser stack simulation (shift and reduction operations)



Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

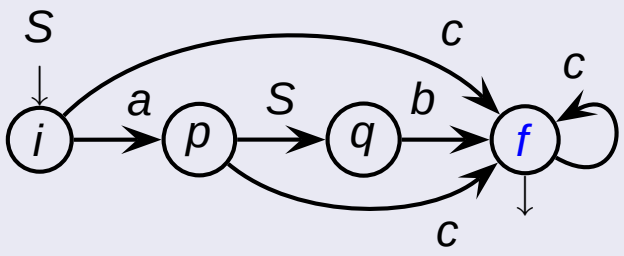
done

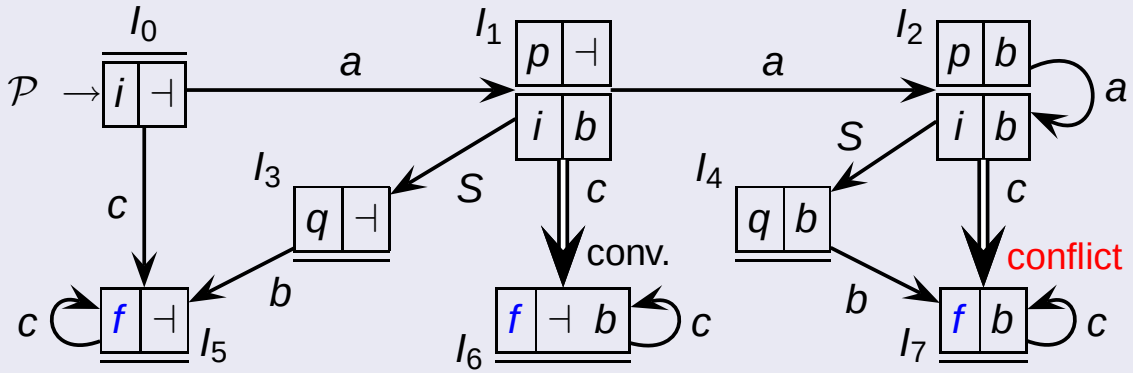
a	S	b	⊣

$1\langle i, \dashv, \perp\rangle$	a	$1\langle p, \dashv, 1\rangle$ $2\langle i, b, \perp\rangle$
J_0		J_1

Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

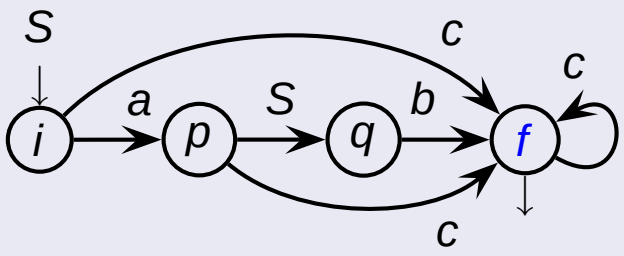
shift *S*

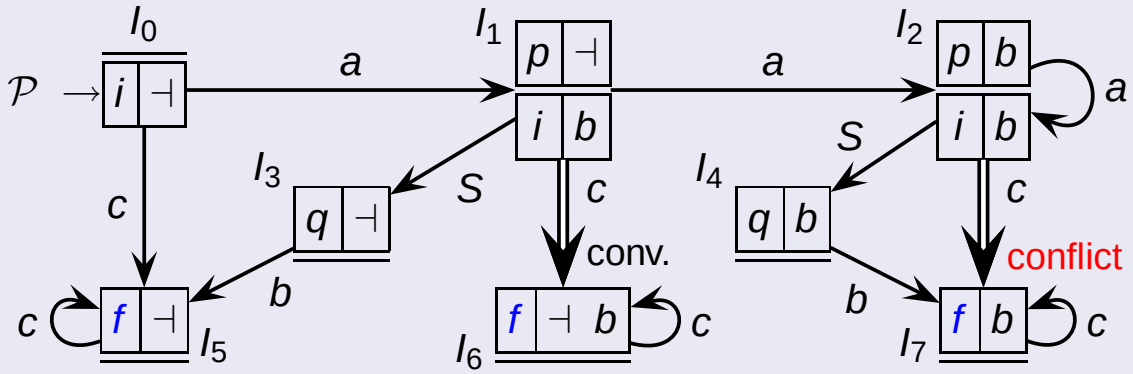
a	*S*	*b*	⊣

$1\langle i, \dashv, \perp\rangle$	*a*	$1\langle p, \dashv, 1\rangle$ $2\langle i, b, \perp\rangle$
J_0		J_1

Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

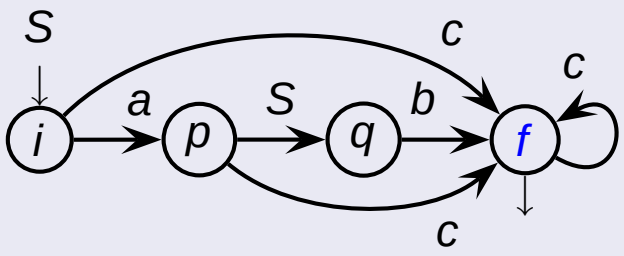
done

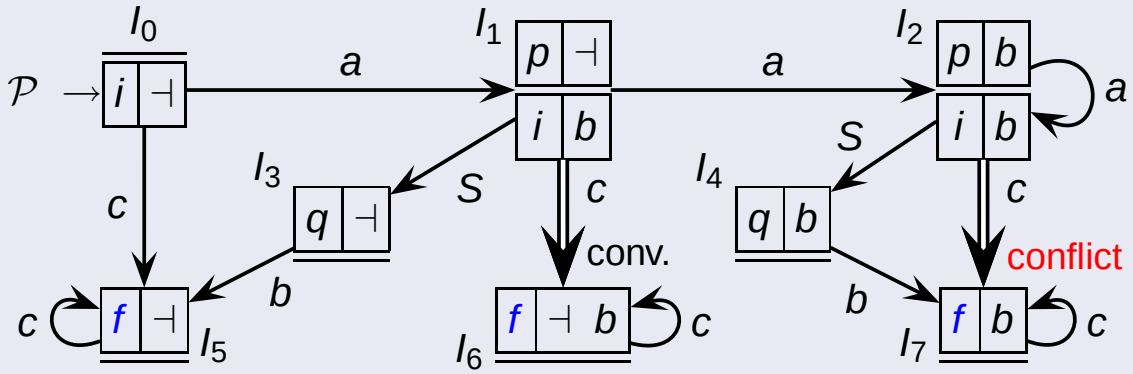
a	S	b	⊣

$1\langle i, \dashv, \perp\rangle$	a	$1\langle p, \dashv, 1\rangle$ $2\langle i, b, \perp\rangle$	S	$1\langle q, \dashv, 1\rangle$
J_0		J_1		J_3

Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

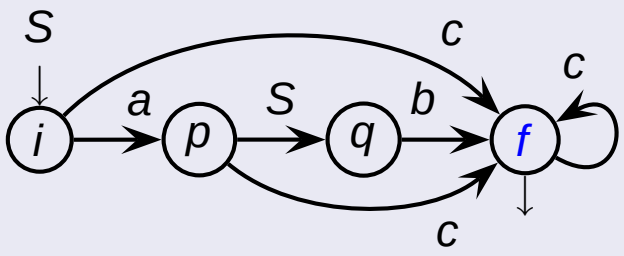
shift *b*

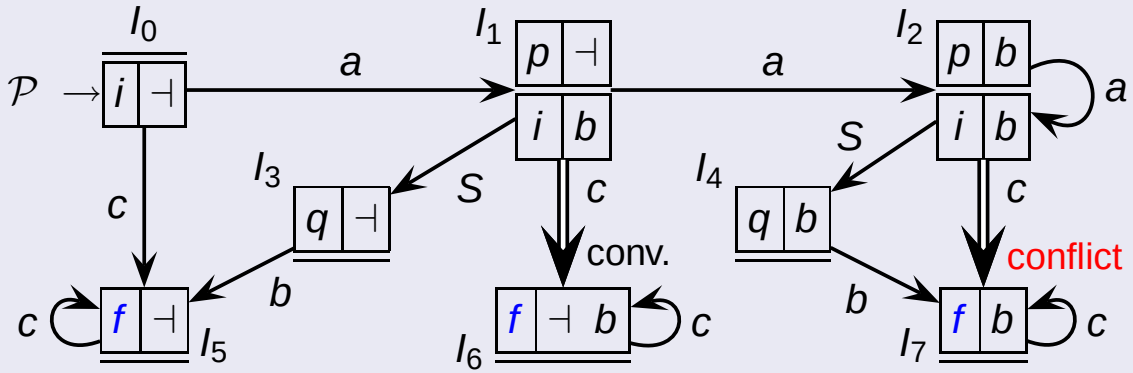
a	*S*	*b*	⊣

$1\langle i, \dashv, \perp\rangle$	*a*	$1\langle p, \dashv, 1\rangle$ $2\langle i, b, \perp\rangle$	*S*	$1\langle q, \dashv, 1\rangle$
J_0		J_1		J_3

Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

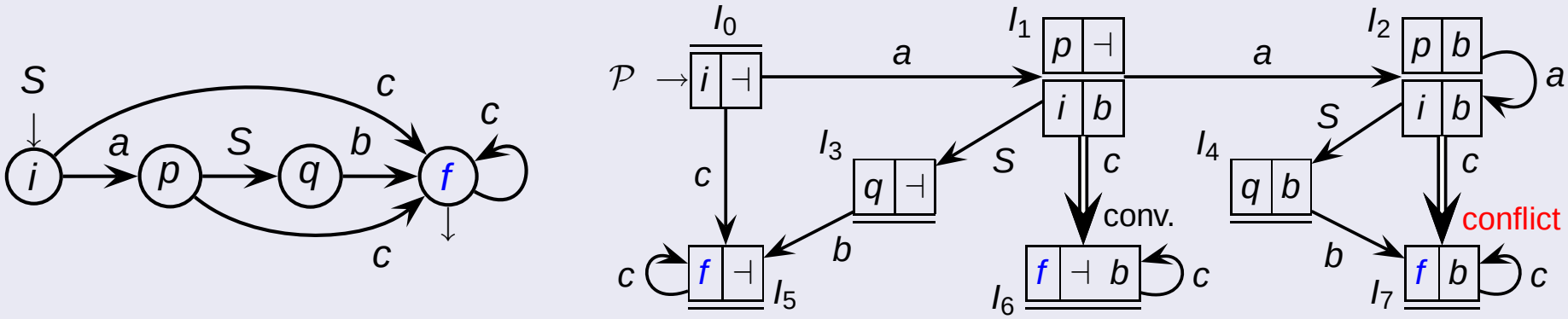
done

a	S	b	⊣

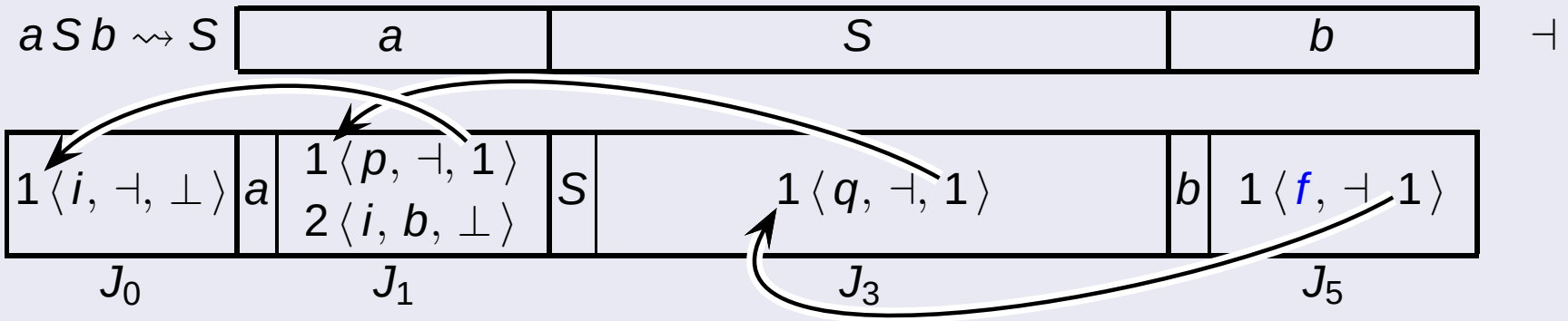
$1\langle i, \dashv, \perp\rangle$	a	$1\langle p, \dashv, 1\rangle$ $2\langle i, b, \perp\rangle$	S	$1\langle q, \dashv, 1\rangle$	b	$1\langle f, \dashv, 1\rangle$
J_0		J_1		J_3		J_5

Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)

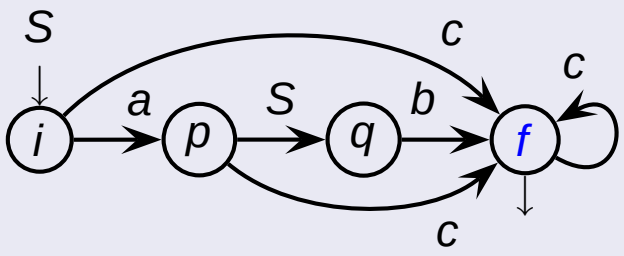


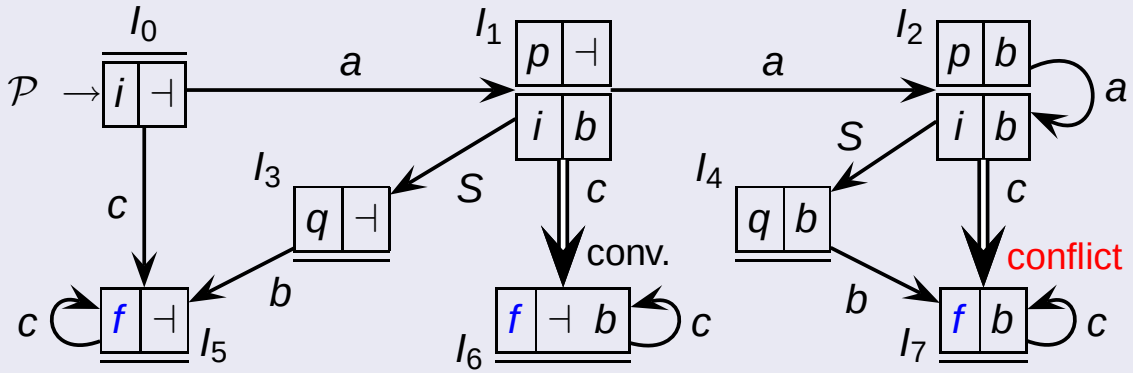
Parser stack simulation (shift and reduction operations)



Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

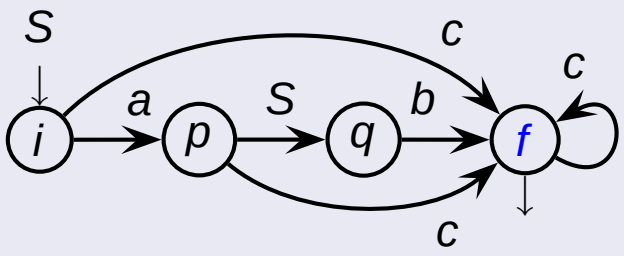
done

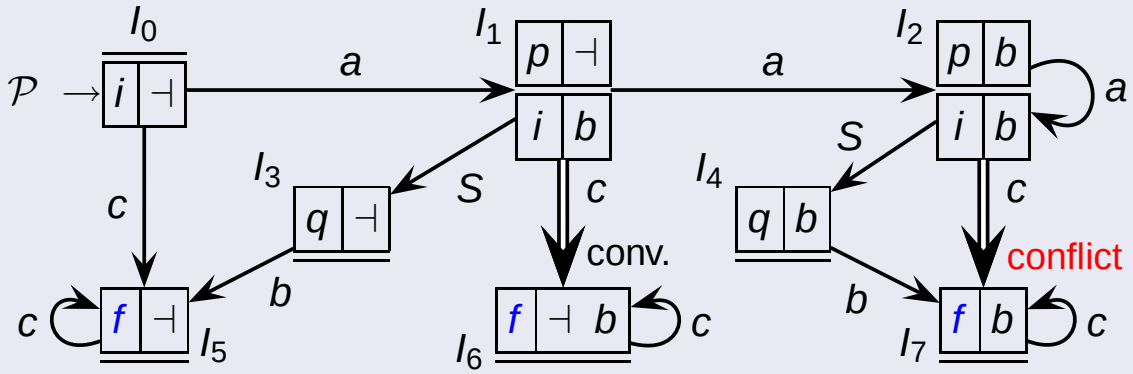
$1\,\langle i, \dashv, \perp \rangle$
J_0

whole input scanned and stack empty

Simulation of stack operations

TN and its pilot (the part with a Conv. conflict is not simulated)





Parser stack simulation (shift and reduction operations)

accepted

$1\,\langle i, \dashv, \perp \rangle$
J_0

whole input scanned and stack empty

Pilot graph optimization

The pilot graph can be optimized for two purposes:

- compacting the parser stack and reducing its memory occupation
- speeding up the parser stack operations, in particular the pop one

For reducing the memory occupation of the stack:

- include item links directly in the pilot m-states
- split the convergence items in the pilot m-states
- so the stack just contains symbols and m-state ids

For speeding up the pop of the reduction handle:

- include the reduction handle length in the pilot m-states
- so the reduction handle can be popped as a whole object

The number of pilot m-states may get larger, yet not too much.

Parser performances (all the measures are $\times 10^3$)

TN and pilot size for the *Java* language (*EBNF* grammar by Oracle)

graph	(m-)states	trans.	conv. trans.
TN	0.53	0.59	–
LR pilot[a]	2.9	25.8	–
ELR pilot	1.9	25.2	4.3

[a] *EBNF* grammar converted into *BNF* with more nonterminals.

Parser performance in language tokens / 10^{-3} s (on a consumer *PC*)

Java	*LR*	*ELR*
	0.94	0.95

JSON	*LR*	*ELR*
	2.2	2.3

item identifiers are annotated in the pilot, not kept on stack

Results and future research

About the *LR* parsing theory for *EBNF* grammars

Defined a new condition for *LR* parsing of *EBNF* grammars.

This condition only implies the new convergence conflict and so generalizes the classical *LR* condition for *BNF* grammars.

About the direct parsing of *EBNF* grammars

Defined a new *LR* parser model for *EBNF* grammars, which:

- is not less deterministic than converting the grammar into *BNF*
- with optimization, is as fast as the classical *LR* parser for *BNF*

Ongoing research . . .

More optimization of the parser pilot graph.

Experimentation and comparison with Bison.

Three significant references

J. C. Beatty.
On the relationship between the LL(1) and LR(1) grammars.
JACM, 29(4):1007–1022, 1982.

S. Heilbrunner.
On the definition of ELR(k) and ELL(k) grammars.
Acta Inform., 11:169–176, 1979.

S. Crespi Reghizzi, L. Breveglieri, and A. Morzenti.
Formal languages and compilation.
Springer, London, 2nd edition, 2013.