

Linguaggi di Floyd

Violetta Lonati

DSI - Università degli studi di Milano

PRIN Workshop Varese 2011

*Joint work with DEI - Politecnico di Milano
Stefano Crespi Reghizzi, Dino Mandrioli, Matteo Pradella*

Historical background and motivation

Operator-precedence languages, or Floyd languages

Automata for Floyd languages

Ω -languages and model checking

Historical background and motivation

Operator-precedence languages, or Floyd languages

Automata for Floyd languages

Ω -languages and model checking

Floyd languages

- Operator precedence grammars, inspired by the structure of arithmetic expressions, and aimed at deterministic (hence efficient) parsing [Floyd 1963]
- Dyck languages and the fundamental theorem by Chomsky and Shützenberger [1963]
- Parenthesis grammars [McNaughton 1967]
- Tree automata [Thatcher 1967]
- Some early investigation on the algebraic properties of operator precedence grammars and closure properties of context-free languages compared with regular languages [Crespi Reghizzi, Mandrioli and Martin 1978]

Historical background (2)

After 40 years and more

- Visibly Pushdown Languages [Alur and Madhusudan 2004]
 - suitable to model structured mark-up languages such as XML and apply model checking techniques
 - closure properties of regular languages (boolean, concatenation, kleene *, ...)
- various related families have been then defined and studied [Nowotka and Srba 2007, Caucal 2008]

FLs are back

**FLs actually include VPLs
and enjoy the same closure properties**

[Crespi Reghizzi and Mandrioli 2009]

Historical background and motivation

Operator-precedence languages, or Floyd languages

Automata for Floyd languages

Ω -languages and model checking

Let's start with an example

We have an arithmetic expression

$$4 + 5 \times 6$$

we want to evaluate it respecting the precedence among operators

Let's start with an example

We have an arithmetic expression

$$4 + 5 \times 6$$

we want to evaluate it respecting the precedence among operators

$$\begin{array}{r} 4 + \frac{5 \times 6}{30} \\ \hline 34 \end{array}$$

Let's start with an example

We have an arithmetic expression

$$4 + 5 \times 6$$

we want to evaluate it respecting the precedence among operators

$$\begin{array}{r} 4 + \frac{5 \times 6}{30} \\ \hline 34 \end{array}$$

It's like we add some parentheses

$$(4 + (5 \times 6))$$

How does the parsing work?

Implicitly, we refer to a context-free grammar:

$$S \rightarrow E$$

$$E \rightarrow E + T$$

$$E \rightarrow T \times n$$

$$E \rightarrow n$$

$$T \rightarrow T \times n$$

$$T \rightarrow n$$

How does the parsing work?

Implicitly, we refer to a context-free grammar:

$$S \rightarrow E$$

$$E \rightarrow E + T$$

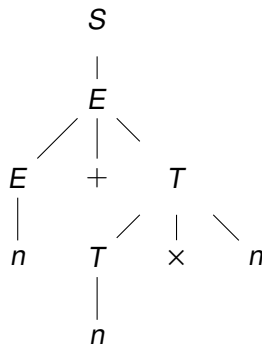
$$E \rightarrow T \times n$$

$$E \rightarrow n$$

$$T \rightarrow T \times n$$

$$T \rightarrow n$$

$n + n \times n$



How does the parsing work?

Implicitly, we refer to a context-free grammar:

$$S \rightarrow E$$

$$E \rightarrow E + T$$

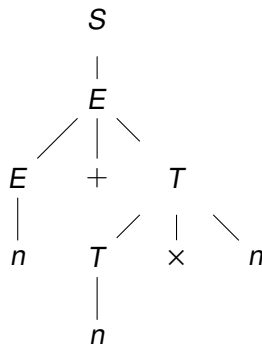
$$E \rightarrow T \times n$$

$$E \rightarrow n$$

$$T \rightarrow T \times n$$

$$T \rightarrow n$$

$$n + n \times n$$



Suppose you have a method for inserting the proper parentheses into an expression ... you get the **stencil** of the parsing tree

$$(n + (n \times n))$$

How does the parsing work?

Implicitly, we refer to a context-free grammar:

$$S \rightarrow E$$

$$E \rightarrow E + T$$

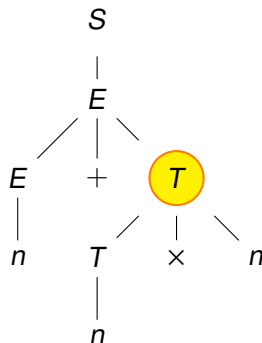
$$E \rightarrow T \times n$$

$$E \rightarrow n$$

$$T \rightarrow T \times n$$

$$T \rightarrow n$$

$$n + n \times n$$



Suppose you have a method for inserting the proper parentheses into an expression ... you get the **stencil** of the parsing tree

$$(n + (n \times n))$$

How does the parsing work?

Implicitly, we refer to a context-free grammar:

$$S \rightarrow E$$

$$E \rightarrow E + T$$

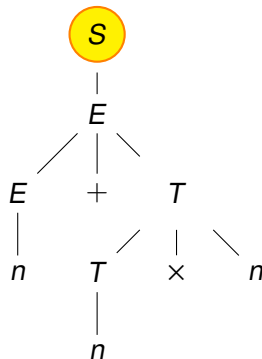
$$E \rightarrow T \times n$$

$$E \rightarrow n$$

$$T \rightarrow T \times n$$

$$T \rightarrow n$$

$n + n \times n$



Suppose you have a method for inserting the proper parentheses into an expression ... you get the **stencil** of the parsing tree

$(n + (n \times n))$

Precedence table

Whenever a parenthesis could be inserted between two symbol, we introduce a **parentheses generator**

- open-parentheses generator \langle
- closed-parentheses generator \rangle
- no parentheses \doteq

If $a \langle b$ we say that **a yields precedence**

if $a \rangle b$ we say that **a takes precedence**

For each pair of symbols we determine the admissible parentheses generator and obtain the **precedence table**

Precedence table - example

Grammar

$$S \rightarrow E$$

$$E \rightarrow E + T$$

$$E \rightarrow T \times n$$

$$E \rightarrow n$$

$$T \rightarrow T \times n$$

$$T \rightarrow n$$

Precedence table - example

Grammar

$$S \rightarrow E$$

$$E \rightarrow E + T$$

$$E \rightarrow T \times n$$

$$E \rightarrow n$$

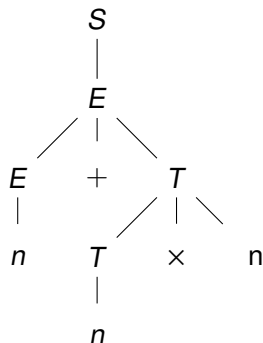
$$T \rightarrow T \times n$$

$$T \rightarrow n$$

Precedence table

	n	$+$	\times
n			
$+$			
\times			

Derivation tree



Precedence table - example

Grammar

$$S \rightarrow E$$

$$E \rightarrow E + T$$

$$E \rightarrow T \times n$$

$$E \rightarrow n$$

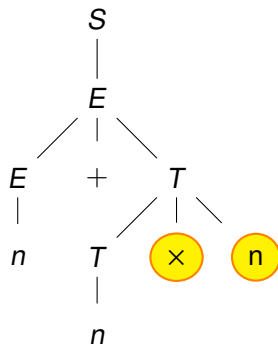
$$T \rightarrow T \times n$$

$$T \rightarrow n$$

Precedence table

	n	$+$	\times
n			
$+$			
\times	$\dot{=}$		

Derivation tree



Precedence table - example

Grammar

$$S \rightarrow E$$

$$E \rightarrow E + T$$

$$E \rightarrow T \times n$$

$$E \rightarrow n$$

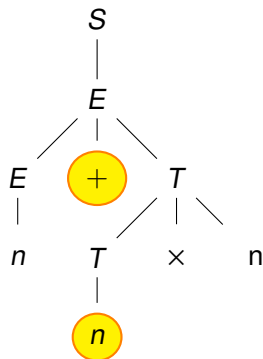
$$T \rightarrow T \times n$$

$$T \rightarrow n$$

Precedence table

	n	$+$	\times
n			
$+$	$<$		
\times	\doteq		

Derivation tree



Precedence table - example

Grammar

$$S \rightarrow E$$

$$E \rightarrow E + T$$

$$E \rightarrow T \times n$$

$$E \rightarrow n$$

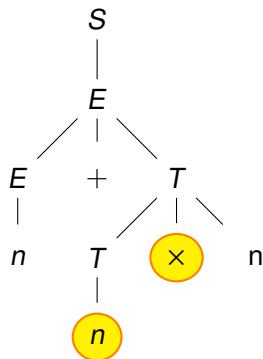
$$T \rightarrow T \times n$$

$$T \rightarrow n$$

Precedence table

	n	$+$	\times
n			$>$
$+$	$<$		
\times	\doteq		

Derivation tree



Precedence table - example

Grammar

$$S \rightarrow E$$

$$E \rightarrow E + T$$

$$E \rightarrow T \times n$$

$$E \rightarrow n$$

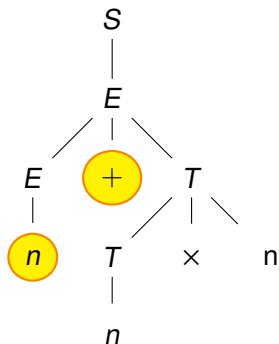
$$T \rightarrow T \times n$$

$$T \rightarrow n$$

Precedence table

	n	$+$	\times
n		$>$	$>$
$+$	$<$		
\times	\doteq		

Derivation tree



Precedence table - example

Grammar

$$S \rightarrow E$$

$$E \rightarrow E + T$$

$$E \rightarrow T \times n$$

$$E \rightarrow n$$

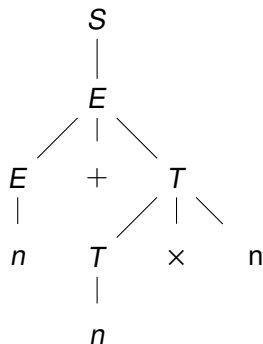
$$T \rightarrow T \times n$$

$$T \rightarrow n$$

Precedence table

	n	$+$	\times
n		\succ	\succ
$+$	\prec	\succ	\prec
\times	\doteq		

Derivation tree



Operator precedence grammars, or Floyd grammars

- A CF grammar is an **operator grammar** if each right-hand side is not empty and no right-hand side contains two consecutive terminals. (Every grammar admits an equivalent operator precedence grammar.)
- An operator grammar is a **operator-precedence grammar** if there are no conflicts in its precedence matrix (given a pair of subsequent symbols, there is at most one parenthesis generator).

We name them **Floyd grammar**
in honour of their inventor Robert W. Floyd.

Historical background and motivation

Operator-precedence languages, or Floyd languages

Automata for Floyd languages

Ω -languages and model checking

Automata for Floyd languages

The stack alphabet consists of pairs $[a \quad q]$ where

a is a symbol from the input alphabet (sort of),

q is a state.

Automata for Floyd languages


The stack alphabet consists of pairs [a' q] where

a is a symbol from the input alphabet (sort of),

q is a state.

Variant: the symbol may be marked

Automata for Floyd languages

The stack alphabet consists of pairs [ q] where

a is a symbol from the input alphabet (sort of),

q is a state.

Variant: special starting symbol $\#$

Automata for Floyd languages

The stack alphabet consists of pairs $[a \quad q]$ where

a is a symbol from the input alphabet (sort of),

q is a state.

3 kinds of moves, depending on the precedence relation between the symbol a on top of the stack and next input symbol b :

Automata for Floyd languages

The stack alphabet consists of pairs $[a \quad q]$ where

a is a symbol from the input alphabet (sort of),

q is a state.

3 kinds of moves, depending on the precedence relation between the symbol a on top of the stack and next input symbol b :

push move when $a \doteq b$
 b is pushed on the stack

Automata for Floyd languages

The stack alphabet consists of pairs $[a \quad q]$ where

a is a symbol from the input alphabet (sort of),

q is a state.

3 kinds of moves, depending on the precedence relation between the symbol a on top of the stack and next input symbol b :

push move when $a \doteq b$
 b is pushed on the stack

mark move when $a < b$
 b' is pushed on the stack

Automata for Floyd languages

The stack alphabet consists of pairs $[a \quad q]$ where

a is a symbol from the input alphabet (sort of),

q is a state.

3 kinds of moves, depending on the precedence relation between the symbol a on top of the stack and next input symbol b :

push move when $a \doteq b$
 b is pushed on the stack

mark move when $a < b$
 b' is pushed on the stack

flush move when $a > b$
symbols are popped from the stack until a marked symbol is found; b is not consumed

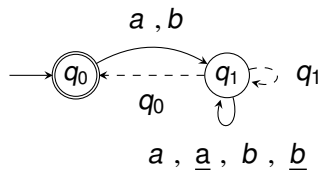
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	\langle	$\dot{=}$	\langle		
\underline{a}	\langle	\rangle	\langle	\rangle	\rangle
b	\langle		\langle	$\dot{=}$	
\underline{b}	\langle	\rangle	\langle	\rangle	\rangle
$\#$	\langle		\langle		$\dot{=}$



$aba\underline{ab}aa\underline{\underline{a}}\#$

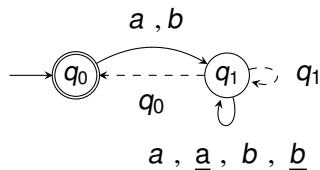
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0]$

$aba\underline{ab}aa\#$

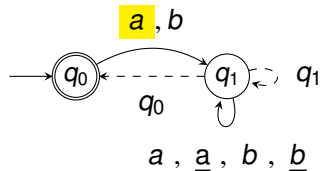
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	$\dot{=}$	<		
\underline{a}	<	>	<	>	>
b	<		<	$\dot{=}$	
\underline{b}	<	>	<	>	>
$\#$	<		<		$\dot{=}$



mark

$[\# q_0]$

$aba\underline{ab}aa\underline{a}\#$

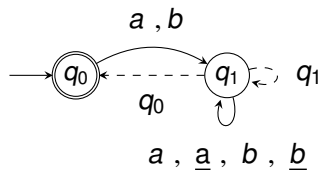
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1]$

$aba\underline{a}ba\underline{a}\#$

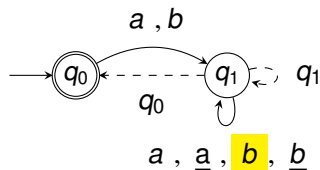
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	\langle	$\dot{=}$	\langle		
\underline{a}	\langle	\rangle	\langle	\rangle	\rangle
b	\langle		\langle	$\dot{=}$	
\underline{b}	\langle	\rangle	\langle	\rangle	\rangle
$\#$	\langle		\langle		$\dot{=}$



mark

$[\# q_0][a' q_1]$

$aba\underline{ab}aaa\#$

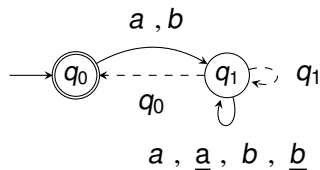
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1][b' q_1]$

$aba\underline{ab}aaa\#$

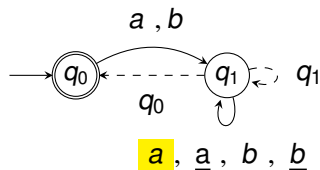
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	$\dot{=}$	<		
\underline{a}	<	>	<	>	>
b	<		<	$\dot{=}$	
\underline{b}	<	>	<	>	>
$\#$	<		<		$\dot{=}$



mark

$[\# q_0][a' q_1][b' q_1]$

$aba\underline{ab}aaa\#$

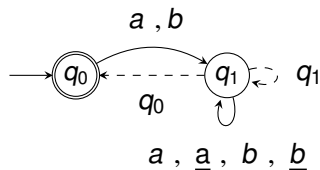
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1][b' q_1][a' q_1]$

$aba\underline{abaaa}\#$

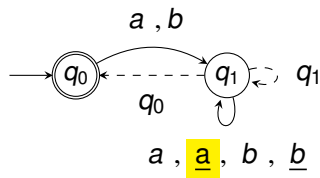
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	⋮	<		
\underline{a}	<	>	<	>	>
b	<		<	⋮	
\underline{b}	<	>	<	>	>
$\#$	<		<		⋮



push

$[\# q_0][a' q_1][b' q_1][a' q_1]$

$aba\underline{abaaa}\#$

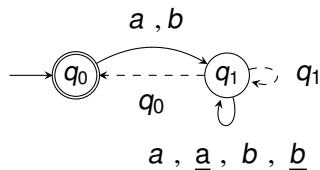
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1][b' q_1][a' q_1][\underline{a} q_1]$

$aba\underline{a}baaa\#$

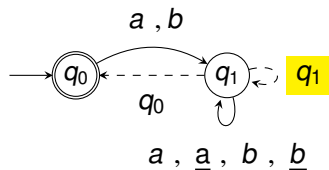
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



flush

$[\# q_0][a' q_1][b' q_1][a' q_1][\underline{a} q_1]$

$aba\underline{a}baaa\#$

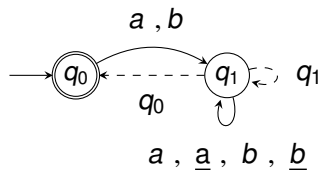
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1][b' q_1]$

$aba\underline{a}baaa\#$

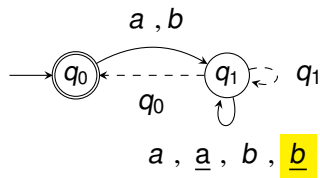
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	\langle	$\dot{=}$	\langle		
\underline{a}	\langle	\rangle	\langle	\rangle	\rangle
b	\langle		\langle	$\dot{=}$	
\underline{b}	\langle	\rangle	\langle	\rangle	\rangle
$\#$	\langle		\langle		$\dot{=}$



push

$[\# q_0][a' q_1][b' q_1]$

$aba\underline{a}baaa\#$

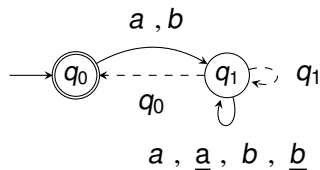
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1][b' q_1][\underline{b} q_1]$

$abaab\underline{a}aa\#$

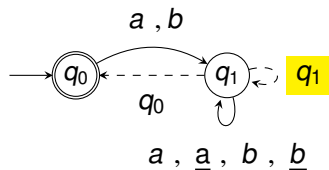
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



flush

$[\# q_0][a' q_1][b' q_1][\underline{b} q_1]$

$abaab\underline{a}aa\underline{b}\#$

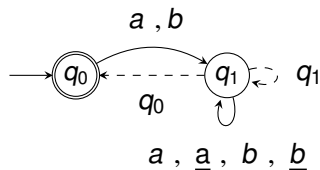
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1]$

$abaab\underline{aaa}\#$

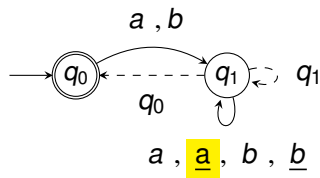
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	⋮	<		
\underline{a}	<	>	<	>	>
b	<		<	⋮	
\underline{b}	<	>	<	>	>
$\#$	<		<		⋮



push

$[\# q_0][a' q_1]$

$abaab\underline{a}aa\underline{b}\#$

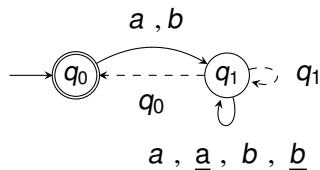
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1][\underline{a} q_1]$

$aba\underline{a}baa\underline{a}\#$

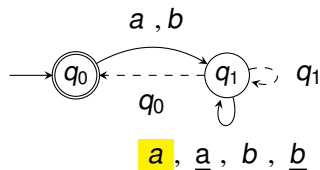
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	$\dot{=}$	<		
\underline{a}	>		<	>	>
b	<		<	$\dot{=}$	
\underline{b}	<	>	<	>	>
$\#$	<		<		$\dot{=}$



mark

$[\# q_0][a' q_1][\underline{a} q_1]$

$aba\underline{a}ba\underline{a}\#$

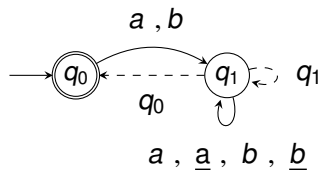
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≡	<		
\underline{a}	<	>	<	>	>
b	<		<	≡	
\underline{b}	<	>	<	>	>
$\#$	<		<		≡



$[\# q_0][a' q_1][\underline{a} q_1][a' q_1]$

$aba\underline{abaa}\underline{a}\#$

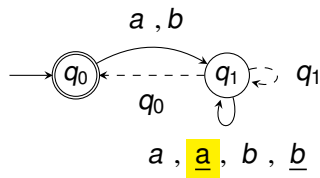
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	⋮	<		
\underline{a}	<	>	<	>	>
b	<		<	⋮	
\underline{b}	<	>	<	>	>
$\#$	<		<		⋮



push

$[\# q_0][a' q_1][\underline{a} q_1][a' q_1]$

$aba\underline{a}ba\underline{a}\#$

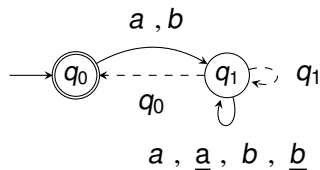
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1][\underline{a} q_1][a' q_1][\underline{a} q_1]$

$aba\underline{a}baaa\#$

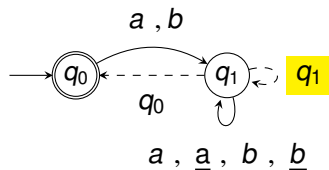
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



flush

$[\# q_0][a' q_1][\underline{a} q_1][a' q_1][\underline{a} q_1]$

$aba\underline{ab}aaa\#$

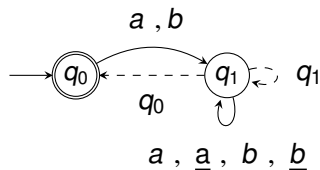
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0][a' q_1][\underline{a} q_1]$

$aba\underline{a}baaa\#$

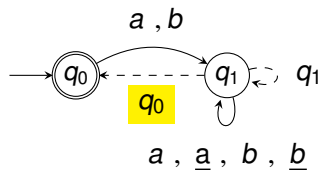
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



flush

$[\# q_0][a' q_1][\underline{a} q_1]$

$aba\underline{a}ba\underline{a}\#$

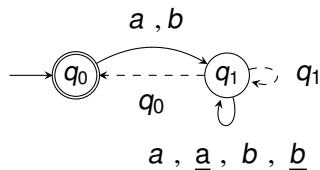
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	≐	<		
\underline{a}	<	>	<	>	>
b	<		<	≐	
\underline{b}	<	>	<	>	>
$\#$	<		<		≐



$[\# q_0]$

$aba\underline{ab}aaa\#$

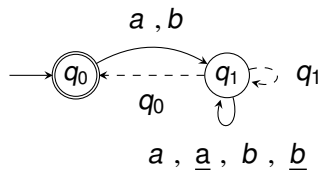
An automata model for Floyd languages - example

◀ restart

▶ skip

Dyck language of balanced strings of parenthesis,
with 2 pairs a, \underline{a} and b, \underline{b} .

	a	\underline{a}	b	\underline{b}	$\#$
a	<	$\dot{=}$	<		
\underline{a}	<	>	<	>	>
b	<		<	$\dot{=}$	
\underline{b}	<	>	<	>	>
$\#$	<		<		$\dot{=}$



accept!

$[\# q_0]$

$aba\underline{a}ba\underline{a}\#$

Theorem

It is easy to define a deterministic version of Floyd automata (as usual: require that the transition function is deterministic).
Deterministic Floyd automata are equivalent to nondeterministic ones.

Theorem

Any language generated by a Floyd grammar can be recognized by a Floyd automaton.

Theorem

Any language recognized by a Floyd automaton can be generated by a Floyd grammar.

Outline

Historical background and motivation

Operator-precedence languages, or Floyd languages

Automata for Floyd languages

Ω -languages and model checking

Having an operational model that defines FLs, it is now easy to introduce extensions to infinite words, i.e. to **ω -languages**.

The classical **Büchi condition of acceptance** can be adapted to FAs: an infinite word x is accepted if and only if configurations with stack $[a q_f]$, where q_f is final, occur infinitely often in the infinite computation on x .

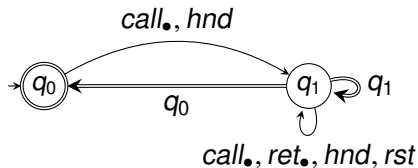
Example

We can model the stack management of a simple programming language that is able to handle nested exceptions

Ω -language - Example

- there are only two procedures, called a and b
- calls and returns are denoted by $call_a$, $call_b$, ret_a , ret_b
- during execution, it is possible to install an exception handler hnd
- signal rst is issued when an exception occur, or after a correct execution to uninstall the handler (with a rst the stack is “flushed”, restoring the state right before the last hnd).

	$call_a$	ret_a	$call_b$	ret_b	hnd	rst
$call_a$	<	$\dot{=}$	<		<	>
ret_a	<	>	<	>		>
$call_b$	<		<	$\dot{=}$	<	>
ret_b	<	>	<	>		>
hnd	<	<	<	<		$\dot{=}$
rst	>	>	>	>		
#	<		<		<	



Conclusions

Floyd grammars and languages

deserve renewed attention in the realm of formal languages

- FL family properly includes VPL
- FL family enjoys all closure properties of VPL and regular languages

Automaton

that perfectly matches the generative power of FGs

- extension to Ω -languages

Prototypical tool called *Flup*

<http://home.dei.polimi.it/pradella>

- interpreter for non-deterministic Floyd Automata
- FG to Floyd Automata translator

Parallel parsing

FLs can be parsed without applying a strictly left-to-right order
[Google grant]

Logic characterization

In order to apply strong model checking techniques to this class of languages