Visibly Pushdown Automata and Streaming

Olivier Serre

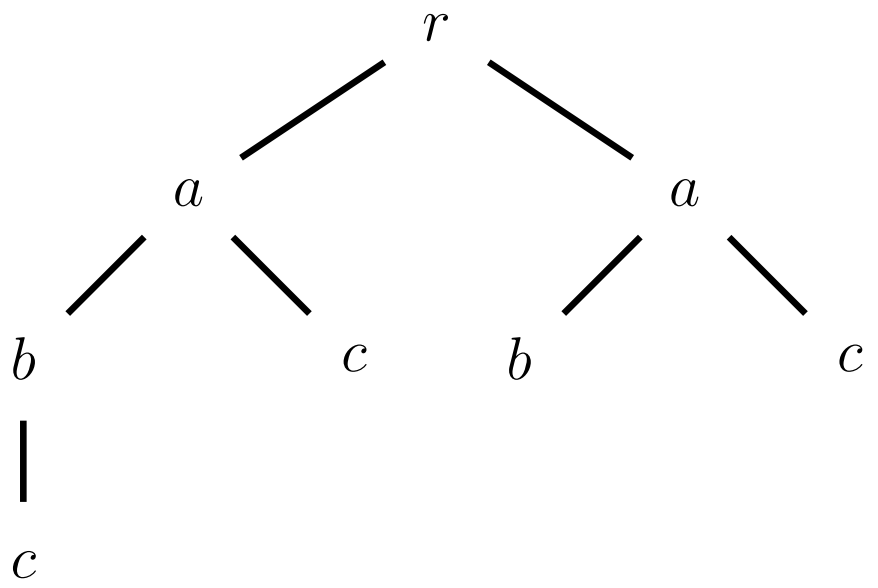
LIAFA, CNRS & Université Paris 7

www.liafa.jussieu.fr/~serre

serre@liafa.jussieu.fr

Streaming

Fact: In real word XML documents are not trees but words obtained by streaming the document.



Associated stream: $r\ a\ b\ c\ \overline{c}\ \overline{b}\ c\ \overline{c}\ \overline{a}\ a\ b\ \overline{b}\ c\ \overline{c}\ \overline{a}\ \overline{r}$

Validation problems for streaming tree documents (1/2)

Adapted from [Segoufin&Vianu, 2002]

Given a Document Type Defintion (DTD) d, equiv. a regular tree language L_d, one sets $Streams(L_d)$ for the set of streaming of trees from L_d.

Validation problems for streaming tree documents (1/2)

Adapted from [Segoufin&Vianu, 2002]

Given a Document Type Defintion (DTD) d, equiv. a regular tree language L_d, one sets $Streams(L_d)$ for the set of streaming of trees from L_d.

Strong validation / Strongly recognizable DTDs:

- Given a DTD d, can one check that a stream belongs to $Streams(L_d)$ using finite memory.
  Equiv. Is $Streams(L_d)$ a regular language?

Validation problems for streaming tree documents (1/2)

Adapted from [Segoufin&Vianu, 2002]

Given a Document Type Defintion (DTD) d, equiv. a regular tree language L_d, one sets $Streams(L_d)$ for the set of streaming of trees from L_d.

Strong validation / Strongly recognizable DTDs:

- Given a DTD d, can one check that a stream belongs to $Streams(L_d)$ using finite memory.
  Equiv. Is $Streams(L_d)$ a regular language?
- Strong validation is decidable [Segoufin& Vianu, 2002].

Validation problems for streaming tree documents (1/2)

Adapted from [Segoufin&Vianu, 2002]

Given a Document Type Defintion (DTD) d, equiv. a regular tree language L_d, one sets $Streams(L_d)$ for the set of streaming of trees from L_d.

Strong validation / Strongly recognizable DTDs:

- Given a DTD d, can one check that a stream belongs to $Streams(L_d)$ using finite memory.
  Equiv. Is $Streams(L_d)$ a regular language?
- Strong validation is decidable [Segoufin& Vianu, 2002].
- There exists DTDs that cannot be strongly validated, *e.g.* trees with a single arbitrary long branched made only of nodes labeled by a: $Streams(L_d) = \{ra^n\overline{a}^n\overline{r} \mid n \geq 0\}$.

Validation problems for streaming tree documents (2/2)

Assume that one proceeds only well formed streams: one can now validate the previous DTD with a finite automaton accepting $ra^*\overline{a}^*\overline{r}$.

Validation problems for streaming tree documents (2/2)

Assume that one proceeds only well formed streams: one can now validate the previous DTD with a finite automaton accepting $ra^*\overline{a}^*\overline{r}$.

Validation / Recognizable DTDs:

- Given a DTD d, can one check that a stream in $Streams(Trees)$ belongs to $Streams(L_d)$ using finite memory.
  Equiv. Is there a regular language R such that

$$Streams(L_d) = R \cap Streams(Trees)?$$

Validation problems for streaming tree documents (2/2)

Assume that one proceeds only well formed streams: one can now validate the previous DTD with a finite automaton accepting $ra^*\overline{a}^*\overline{r}$.

Validation / Recognizable DTDs:

- Given a DTD d, can one check that a stream in $Streams(Trees)$ belongs to $Streams(L_d)$ using finite memory.
  Equiv. Is there a regular language R such that

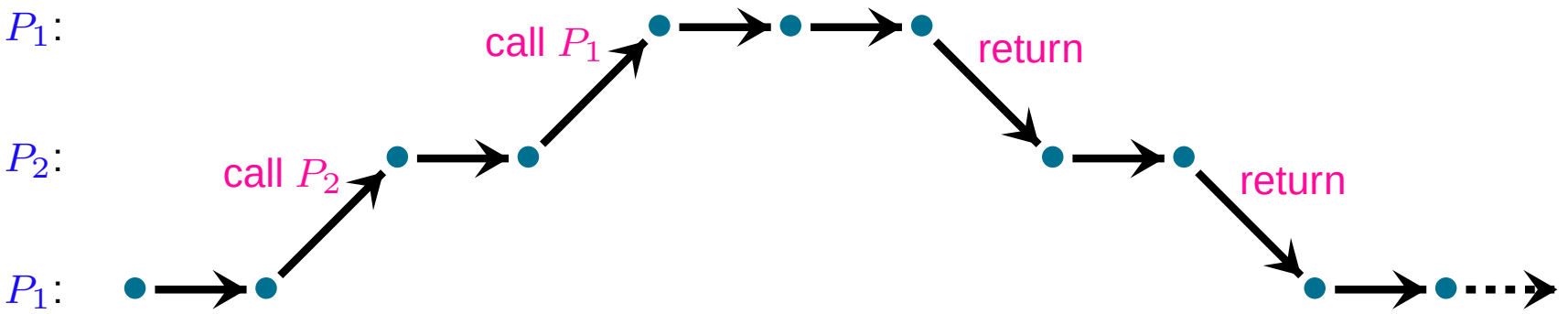
$$Streams(L_d) = R \cap Streams(Trees)?$$

- DTDs that can be validated are said to be recognizable.
- [Segoufin& Vianu, 2002] provides necessary conditions and also sufficient conditions (but no characterization is known).

Formal languages: regularity is not enough...

Regularity does not capture all we would like:

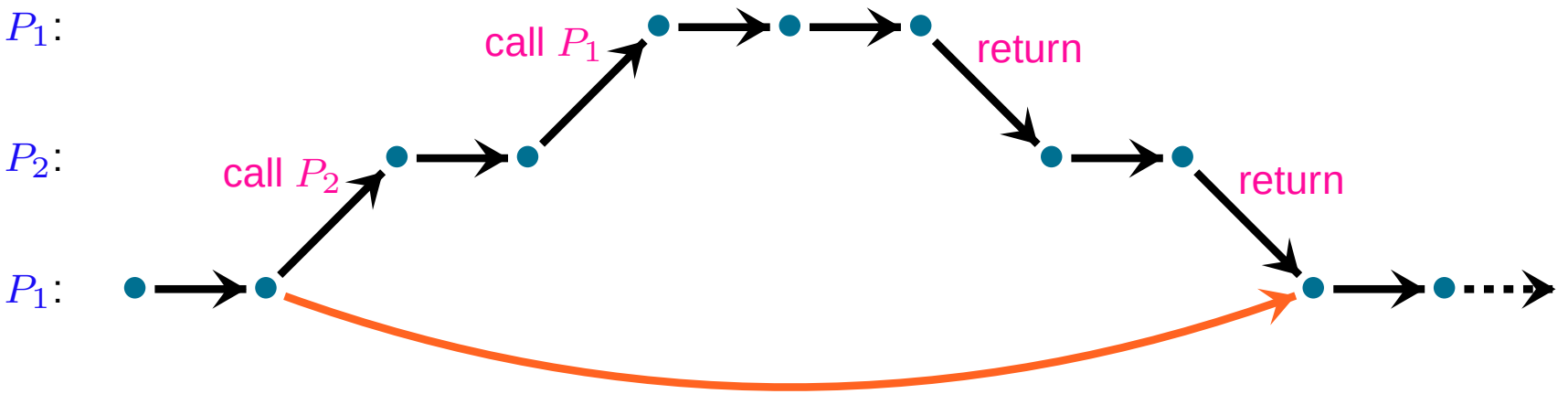
- Specifications for verification/games.



Formal languages: regularity is not enough...

Regularity does not capture all we would like:

- Specifications for verification/games.



Formal languages: regularity is not enough...

Regularity does not capture all we would like:

- Specifications for verification/games.
- To work with semi-structure data.

Example: HTML/XML document using opening/closing tags

```
<html>...<ul> <li> ...</li>...<li> ...</li></ul>...</html>
```

. . . but context free languages are not very handy

Closure properties:

	$\cup$	$\cap$	complement	concatenation	Kleene-$\star$
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No

. . . but context free languages are not very handy

Closure properties:

	$\cup$	$\cap$	complement	concatenation	Kleene-$\star$
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No

Decision problems:

	emptiness	equivalence	inclusion
Regular (NFA)	NLOG	PSPACE	PSPACE
CFL	PTIME	undecidable	undecidable
DCFL	PTIME	decidable	undecidable

. . . but context free languages are not very handy

Closure properties:

	$\cup$	$\cap$	complement	concatenation	Kleene-$\star$
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No

Decision problems:

	emptiness	equivalence	inclusion
Regular (NFA)	NLOG	PSPACE	PSPACE
CFL	PTIME	undecidable	undecidable
DCFL	PTIME	decidable	undecidable

Something in between?

. . . Visibly pushdown languages are good

Closure properties:

	$\cup$	$\cap$	complement	concatenation	Kleene-$\star$
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No
VPL	Yes	Yes	Yes	Yes	Yes

. . . Visibly pushdown languages are good

Closure properties:

	$\cup$	$\cap$	complement	concatenation	Kleene-$\star$
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No
VPL	Yes	Yes	Yes	Yes	Yes

Decision problems:

	emptiness	equivalence	inclusion
Regular (NFA)	NLOG	PSPACE	PSPACE
CFL	PTIME	undecidable	undecidable
DCFL	PTIME	decidable	undecidable
VPL	PTIME	EXPTIME	EXPTIME

Why are context-free languages not robust?

Intersection/union for finite automata:

Why are context-free languages not robust?

Intersection/union for finite automata:

- Use a product construction.
- Adapt the acceptance condition.

Why are context-free languages not robust?

Intersection/union for finite automata:

- Use a product construction.
- Adapt the acceptance condition.

Intersection/union of deterministic context-free languages: may not be context-free.

Why are context-free languages not robust?

Intersection/union for finite automata:

- Use a product construction.
- Adapt the acceptance condition.

Intersection/union of deterministic context-free languages: may not be context-free.

- $\{a^n b^n c^m \mid n, m \geq 0\} \cap \{a^n b^m c^m \mid n, m \geq 0\} = \{a^n b^n c^n \mid n \geq 0\}$
- Where is the problem in the product construction for pushdown automata?

Why are context-free languages not robust?

Intersection/union for finite automata:

- Use a product construction.
- Adapt the acceptance condition.

Intersection/union of deterministic context-free languages: may not be context-free.

- $\{a^n b^n c^m \mid n, m \geq 0\} \cap \{a^n b^m c^m \mid n, m \geq 0\} = \{a^n b^n c^n \mid n \geq 0\}$
- Where is the problem in the product construction for pushdown automata? The stacks are not synchronized.

Solution

Stack is input driven: for each input letter is associated a stack operation.

- Call: push.
- Return: pop.
- Internal action: skip.

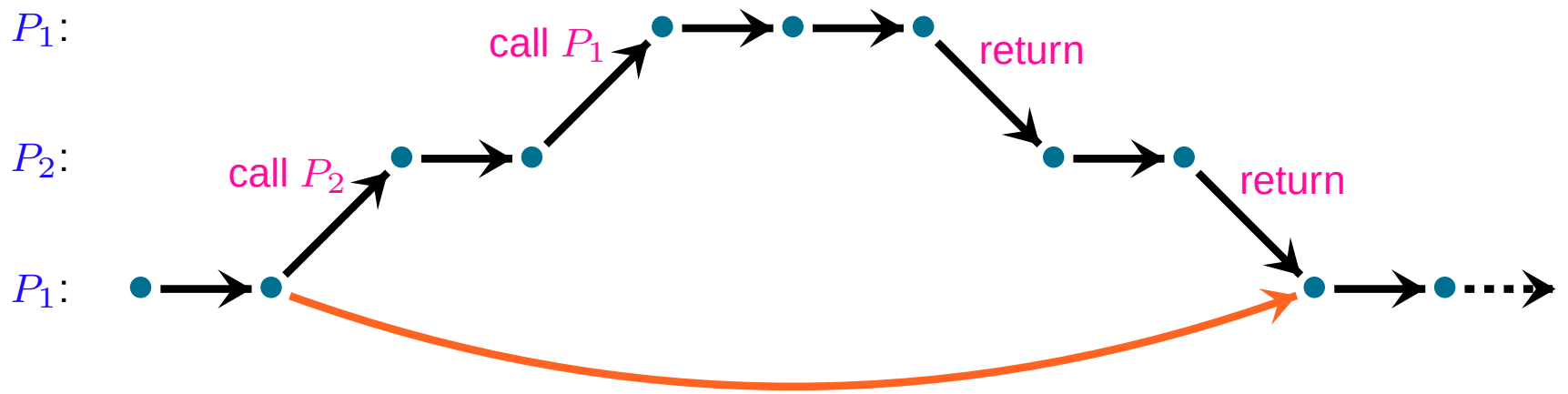
Solution

Stack is input driven: for each input letter is associated a stack operation.

- Call: push.
- Return: pop.
- Internal action: skip.

This restriction is natural:

- Verification of recursive programs.



Solution

Stack is input driven: for each input letter is associated a stack operation.

- Call: push.
- Return: pop.
- Internal action: skip.

This restriction is natural:

- Semi-structure data.

```
<html>...<ul> <li> ...</li>...<li> ...</li></ul>...</html>
```

Visibly pushdown automata

Partitioned alphabet : $A = \langle A_c, A_r, A_{int} \rangle$

- A_c : calls.
- A_r : returns.
- A_{int} : internal actions.

Visibly pushdown automata

Visibly pushdown automaton (VPA) $\mathcal{M} = (Q, A, \Gamma, \bot, Q_{in}, Acc, \Delta)$ **:**

- Q : finite set of states.
- A : partitioned input alphabet.
- Γ : finite stack alphabet.
- $\bot$: bottom of stack symbols.
- Q_{in} : set of initial states.
- Acc : acceptance condition.
- Δ : transition function.

Visibly pushdown automata

Visibly pushdown automaton (VPA) $\mathcal{M} = (Q, A, \Gamma, \bot, Q_{in}, Acc, \Delta)$ **:**

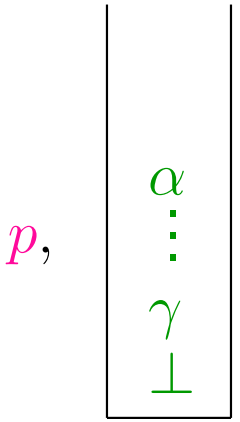
- Q : finite set of states.
- A : partitioned input alphabet.
- Γ : finite stack alphabet.
- $\Delta \subseteq \begin{cases} Q \times A_c \times Q \times (\Gamma \setminus \{\bot\}) \\ Q \times A_r \times (\Gamma \setminus \{\bot\}) \times Q \\ Q \times A_{int} \times Q. \end{cases}$

Visibly pushdown automata

Visibly pushdown automaton (VPA) $\mathcal{M} = (Q, A, \Gamma, \bot, Q_{in}, Acc, \Delta)$ **:**

- Q : finite set of states.
- A : partitioned input alphabet.
- Γ : finite stack alphabet.
- $\Delta \subseteq \begin{cases} Q \times A_c \times Q \times (\Gamma \setminus \{\bot\}) \\ Q \times A_r \times (\Gamma \setminus \{\bot\}) \times Q \\ Q \times A_{int} \times Q. \end{cases}$

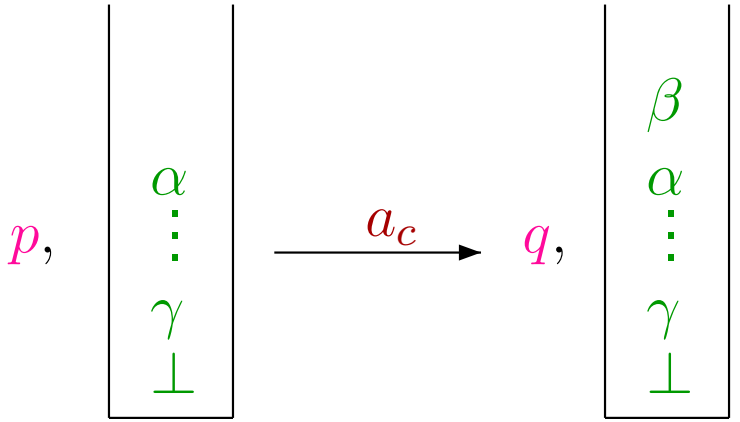
$(p, a_c, q, \beta) \in \Delta$:



Visibly pushdown automaton (VPA) $\mathcal{M} = (Q, A, \Gamma, \bot, Q_{in}, Acc, \Delta)$ **:**

- Q : finite set of states.
- A : partitioned input alphabet.
- Γ : finite stack alphabet.
- $\Delta \subseteq \begin{cases} Q \times A_c \times Q \times (\Gamma \setminus \{\bot\}) \\ Q \times A_r \times (\Gamma \setminus \{\bot\}) \times Q \\ Q \times A_{int} \times Q. \end{cases}$

$(p, a_c, q, \beta) \in \Delta$:

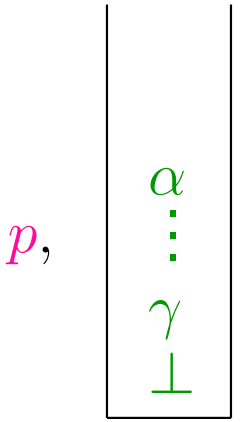


Visibly pushdown automata

Visibly pushdown automaton (VPA) $\mathcal{M} = (Q, A, \Gamma, \bot, Q_{in}, Acc, \Delta)$ **:**

- Q : finite set of states.
- A : partitioned input alphabet.
- Γ : finite stack alphabet.
- $\Delta \subseteq \begin{cases} Q \times A_c \times Q \times (\Gamma \setminus \{\bot\}) \\ Q \times A_r \times (\Gamma \setminus \{\bot\}) \times Q \\ Q \times A_{int} \times Q. \end{cases}$

$(p, a_r, \alpha, q) \in \Delta$:

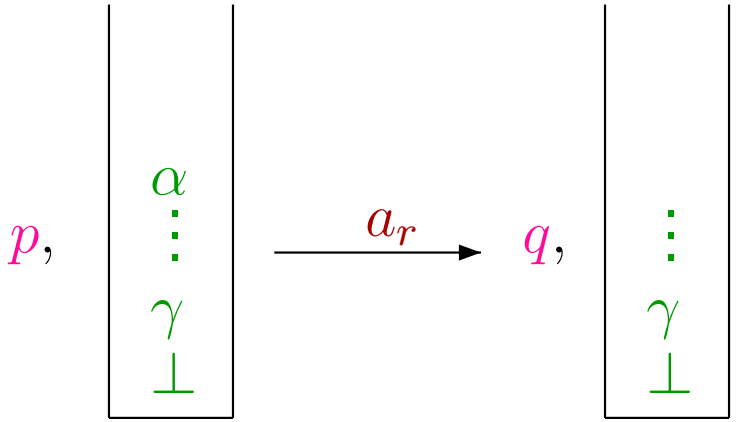


Visibly pushdown automata

Visibly pushdown automaton (VPA) $\mathcal{M} = (Q, A, \Gamma, \bot, Q_{in}, Acc, \Delta)$ **:**

- Q : finite set of states.
- A : partitioned input alphabet.
- Γ : finite stack alphabet.
- $\Delta \subseteq \begin{cases} Q \times A_c \times Q \times (\Gamma \setminus \{\bot\}) \\ Q \times A_r \times (\Gamma \setminus \{\bot\}) \times Q \\ Q \times A_{int} \times Q. \end{cases}$

$(p, a_r, \alpha, q) \in \Delta$:

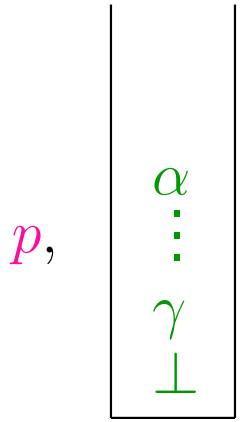


Visibly pushdown automata

Visibly pushdown automaton (VPA) $\mathcal{M} = (Q, A, \Gamma, \bot, Q_{in}, Acc, \Delta)$ **:**

- Q : finite set of states.
- A : partitioned input alphabet.
- Γ : finite stack alphabet.
- $\Delta \subseteq \begin{cases} Q \times A_c \times Q \times (\Gamma \setminus \{\bot\}) \\ Q \times A_r \times (\Gamma \setminus \{\bot\}) \times Q \\ Q \times A_{int} \times Q. \end{cases}$

$(p, a_i, q) \in \Delta$:

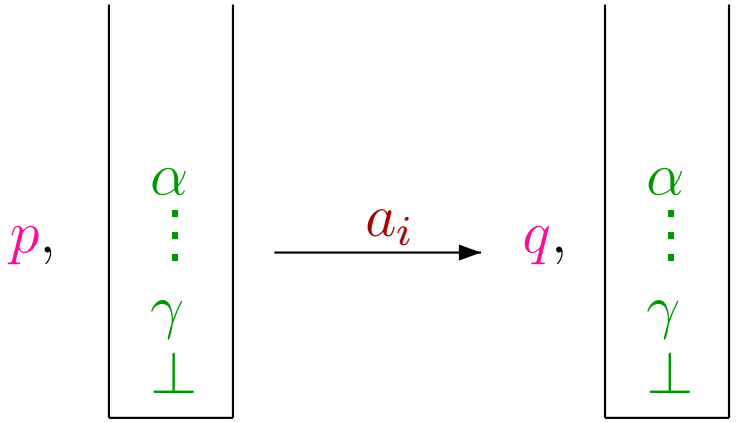


Visibly pushdown automata

Visibly pushdown automaton (VPA) $\mathcal{M} = (Q, A, \Gamma, \bot, Q_{in}, Acc, \Delta)$ **:**

- Q : finite set of states.
- A : partitioned input alphabet.
- Γ : finite stack alphabet.
- $\Delta \subseteq \begin{cases} Q \times A_c \times Q \times (\Gamma \setminus \{\bot\}) \\ Q \times A_r \times (\Gamma \setminus \{\bot\}) \times Q \\ Q \times A_{int} \times Q. \end{cases}$

$(p, a_i, q) \in \Delta$:



Associated languages (VPL)

Visibly pushdown automaton $\mathcal{M} = (Q, A, \Gamma, \perp, Q_{in}, Acc, \Delta)$:

- Finite words case:
  - $Acc = F \subseteq Q$, set of final states.
  - $w \in A^*$ is accepted by $\mathcal{M}$ iff there exists a run that ends in a configuration (q, σ), $q \in F$.
  - $L(\mathcal{M}) = \{w \in A^* \mid w \text{ accepted by } \mathcal{M}\}$.

- Other extensions exist:
  - infinite words,
  - trees.

Matching positions

$A_c = \{c_1, c_2\}$, $A_c = \{r_1, r_2, r_3\}$ and $A_{int} = \{i_1, i_2\}$

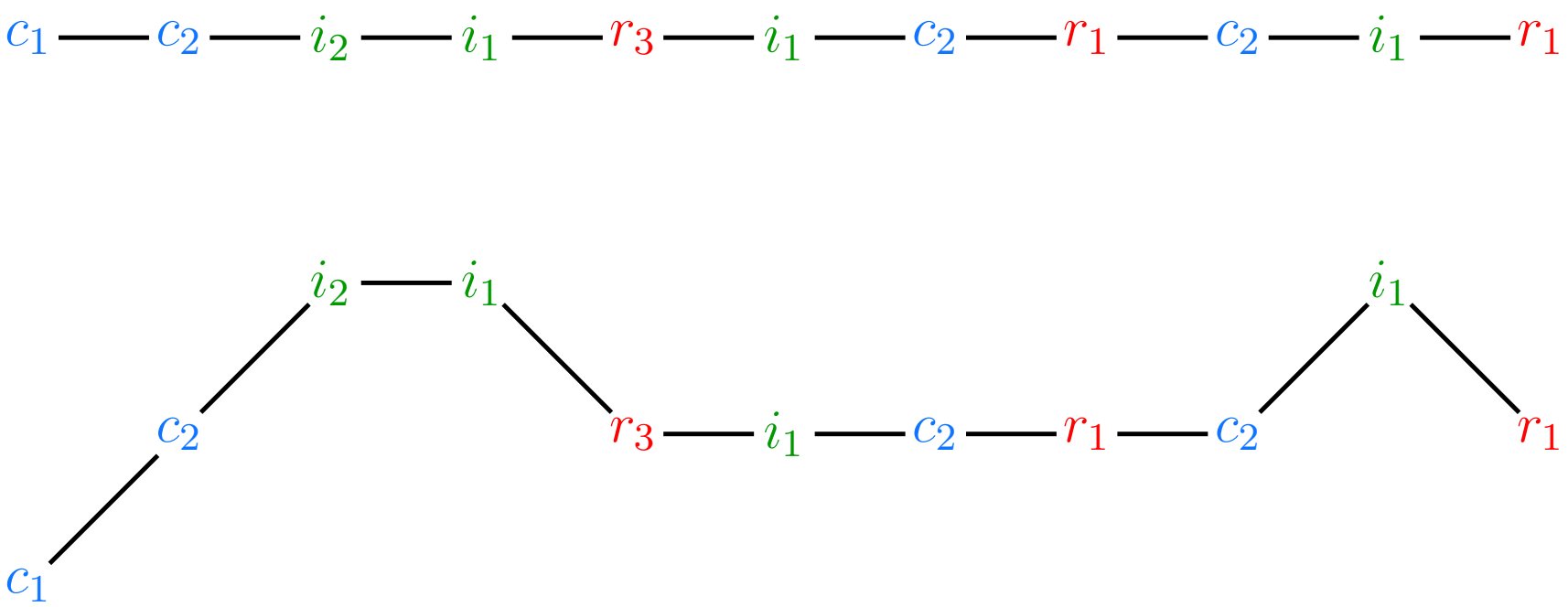
Matching positions

$A_c = \{c_1, c_2\}$, $A_c = \{r_1, r_2, r_3\}$ and $A_{int} = \{i_1, i_2\}$

$c_1 \text{——} c_2 \text{——} i_2 \text{——} i_1 \text{——} r_3 \text{——} i_1 \text{——} c_2 \text{——} r_1 \text{——} c_2 \text{——} i_1 \text{——} r_1$

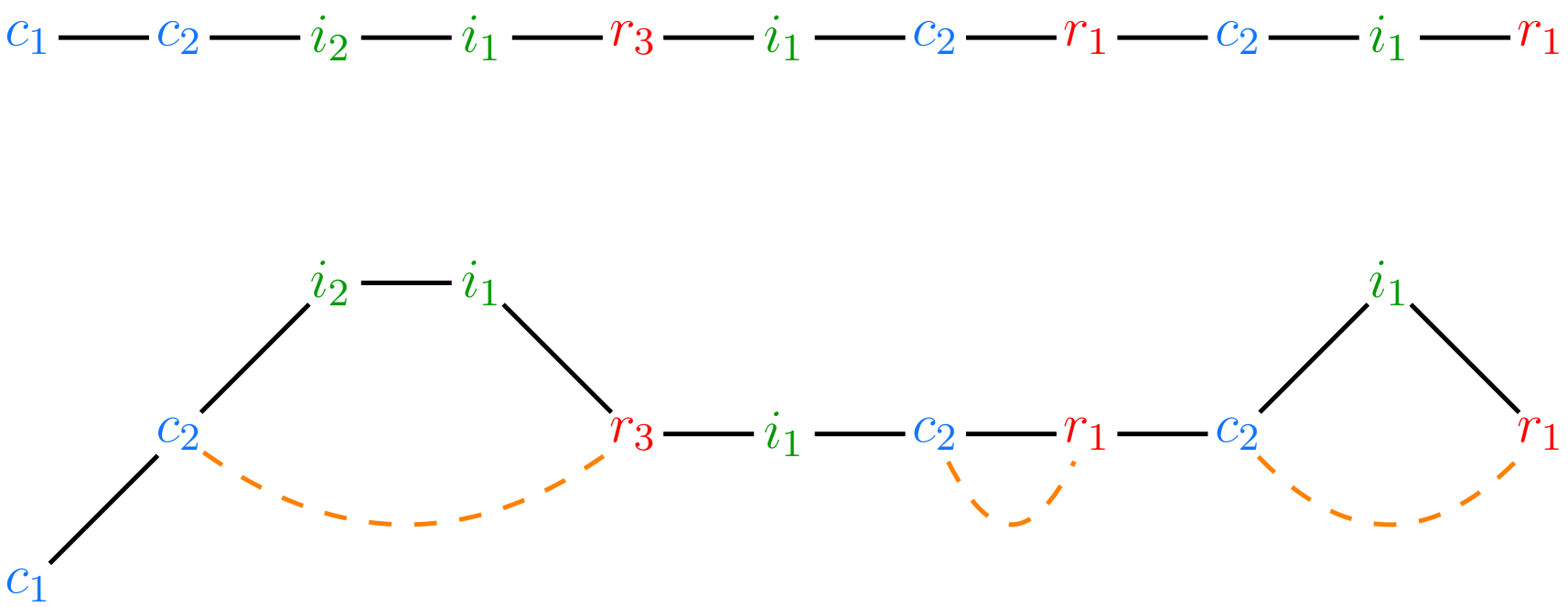
Matching positions

$A_c = \{c_1, c_2\}$, $A_c = \{r_1, r_2, r_3\}$ and $A_{int} = \{i_1, i_2\}$



Matching positions

$A_c = \{c_1, c_2\}$, $A_c = \{r_1, r_2, r_3\}$ and $A_{int} = \{i_1, i_2\}$

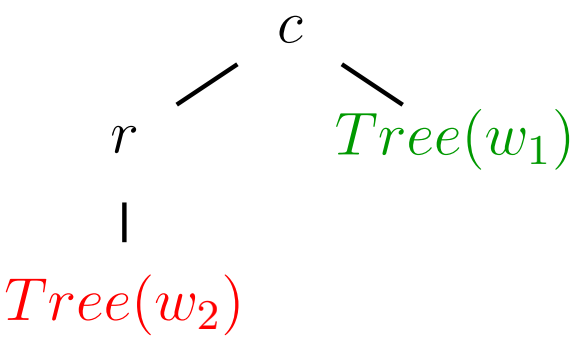


From words to trees

General rule:

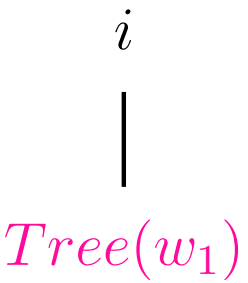
$w = cw_1rw_2$

$Tree(w)$

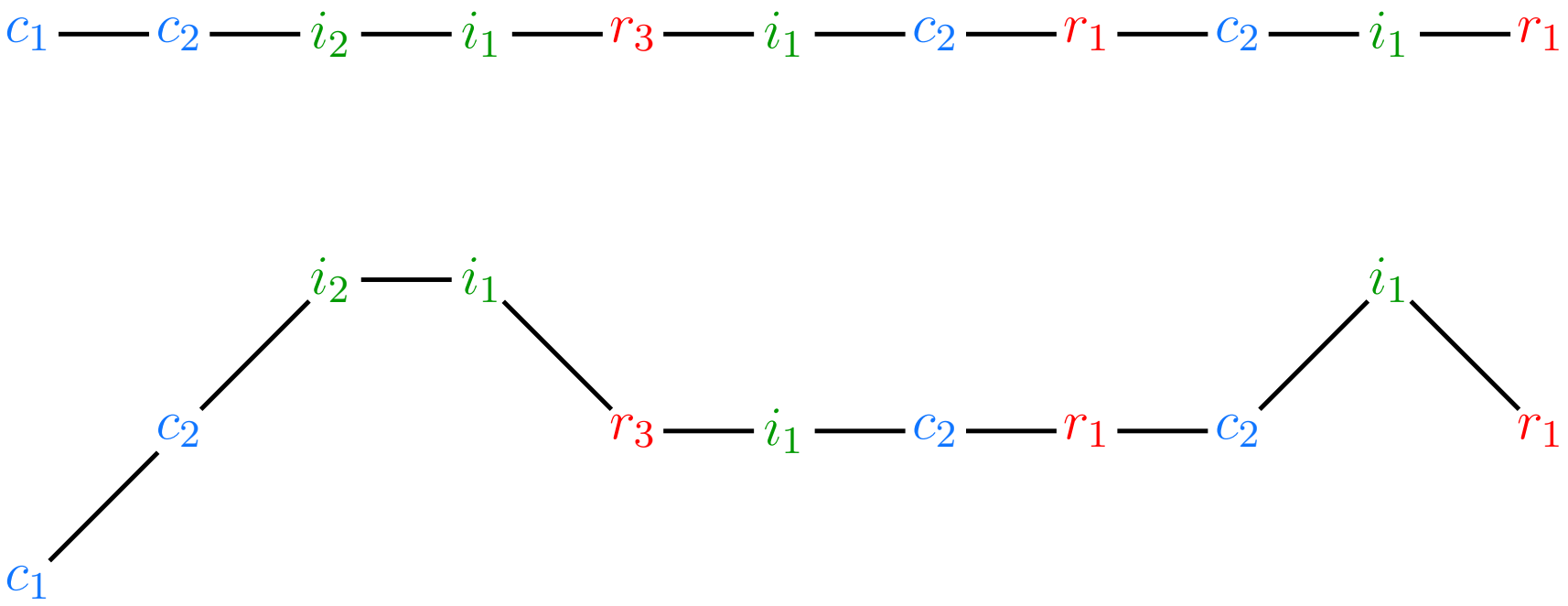


$w = iw_1$

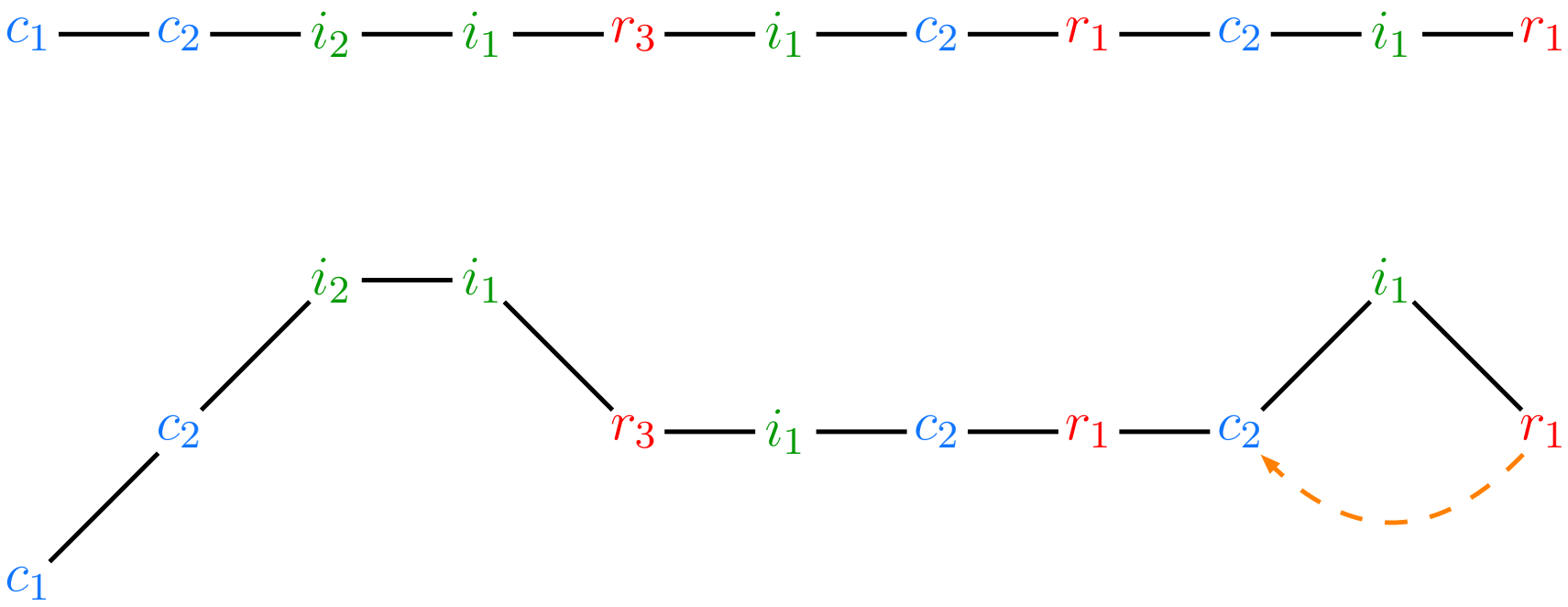
$Tree(w)$



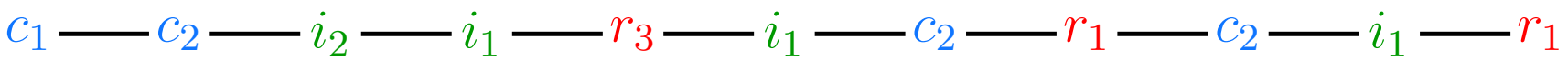
From words to trees: example

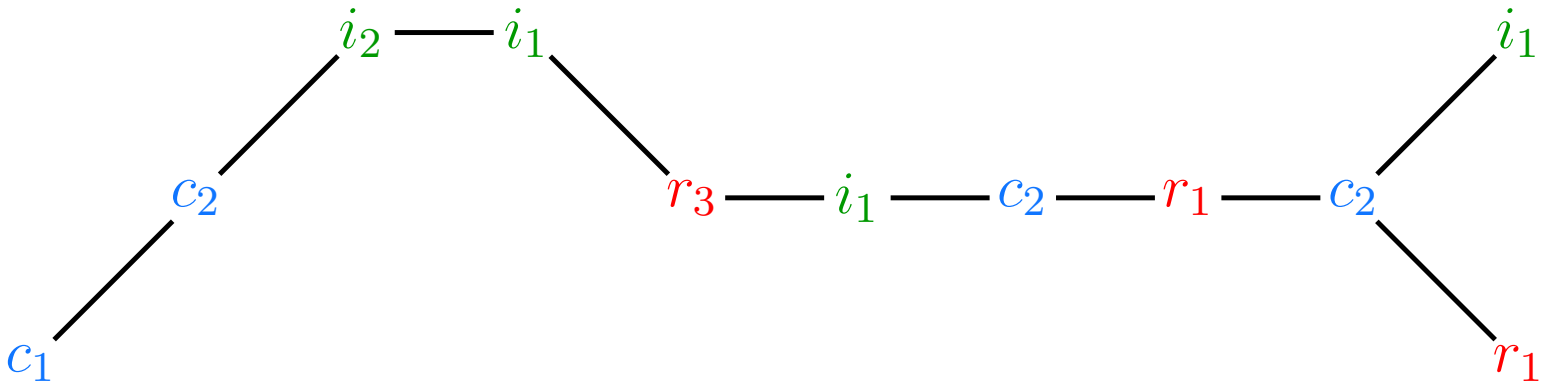


From words to trees: example

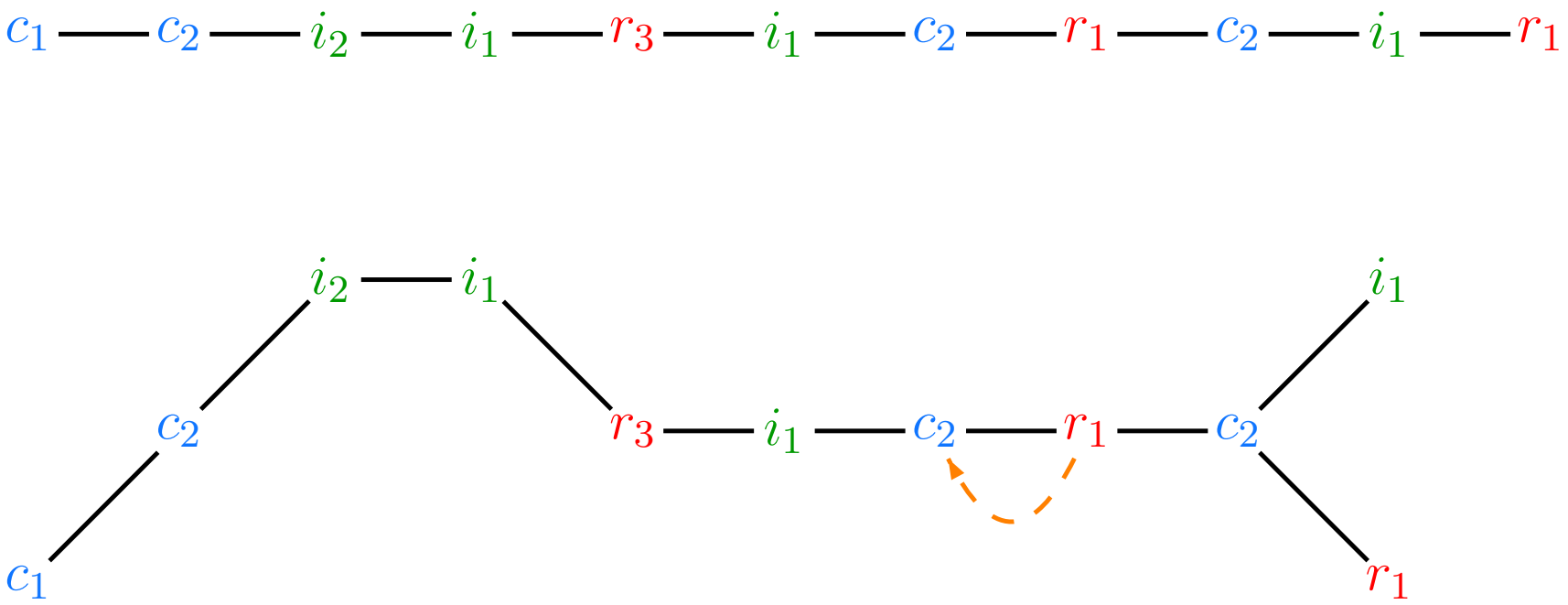


From words to trees: example

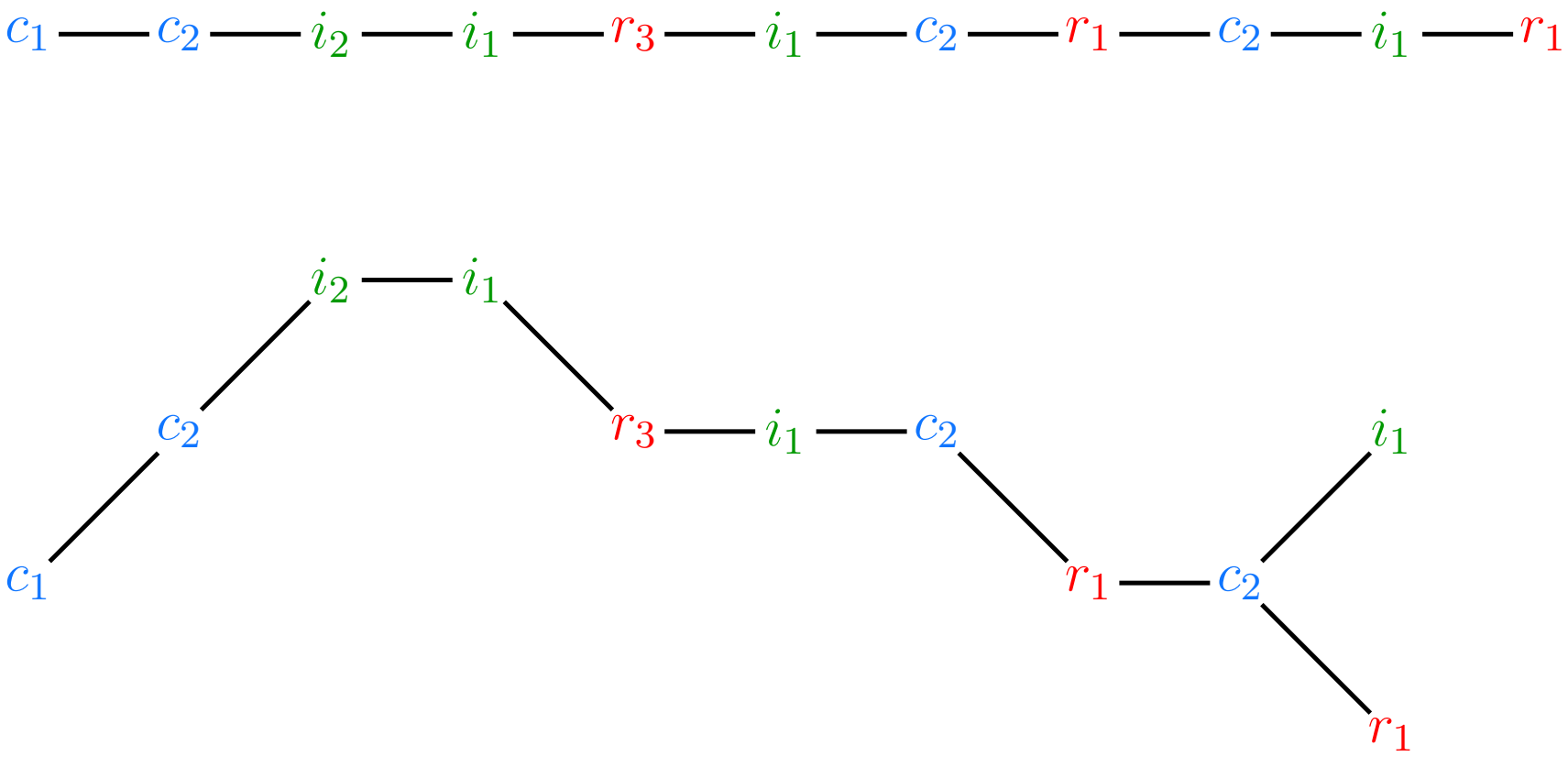




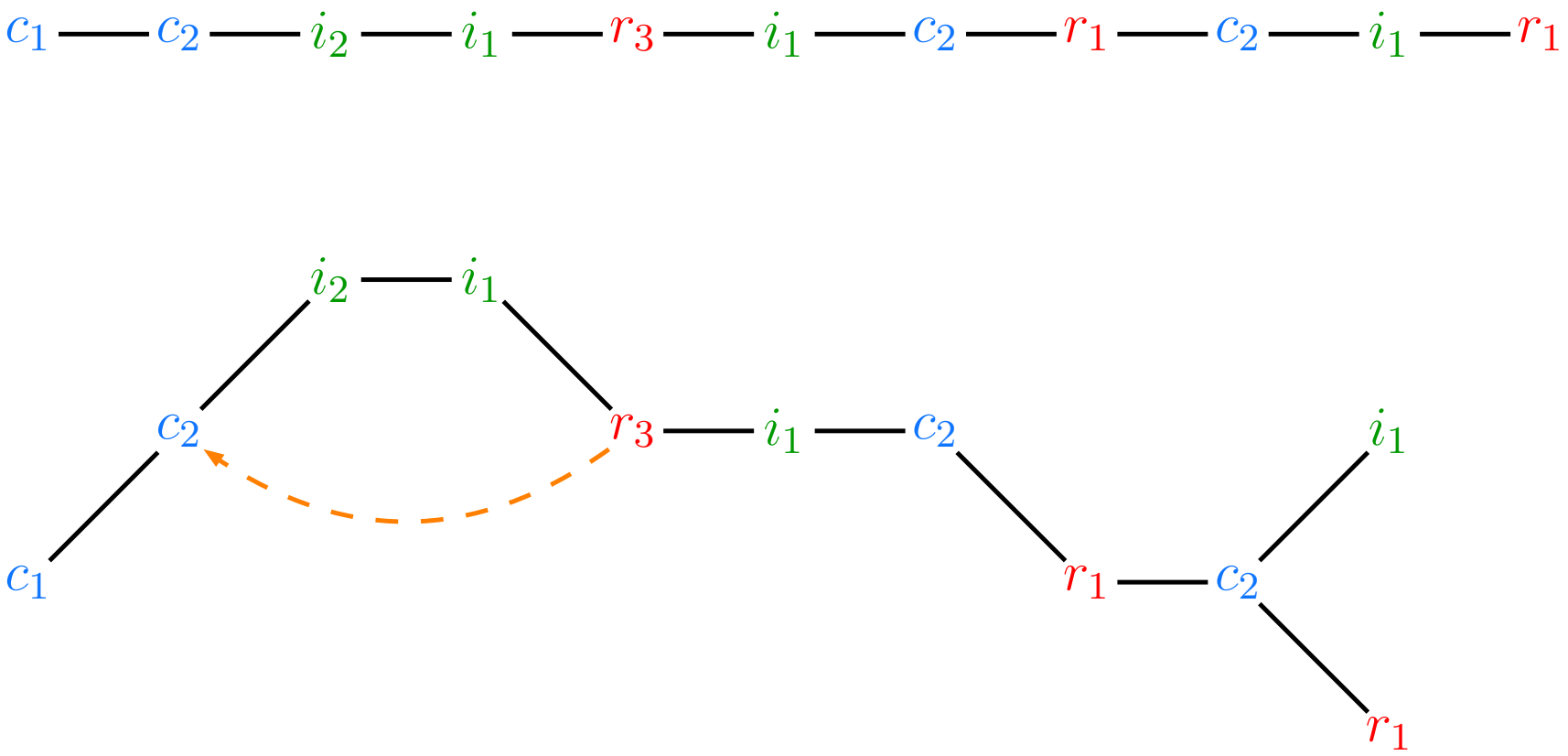
From words to trees: example



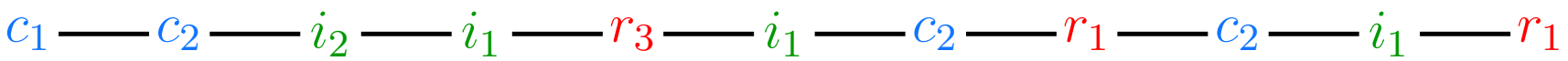
From words to trees: example

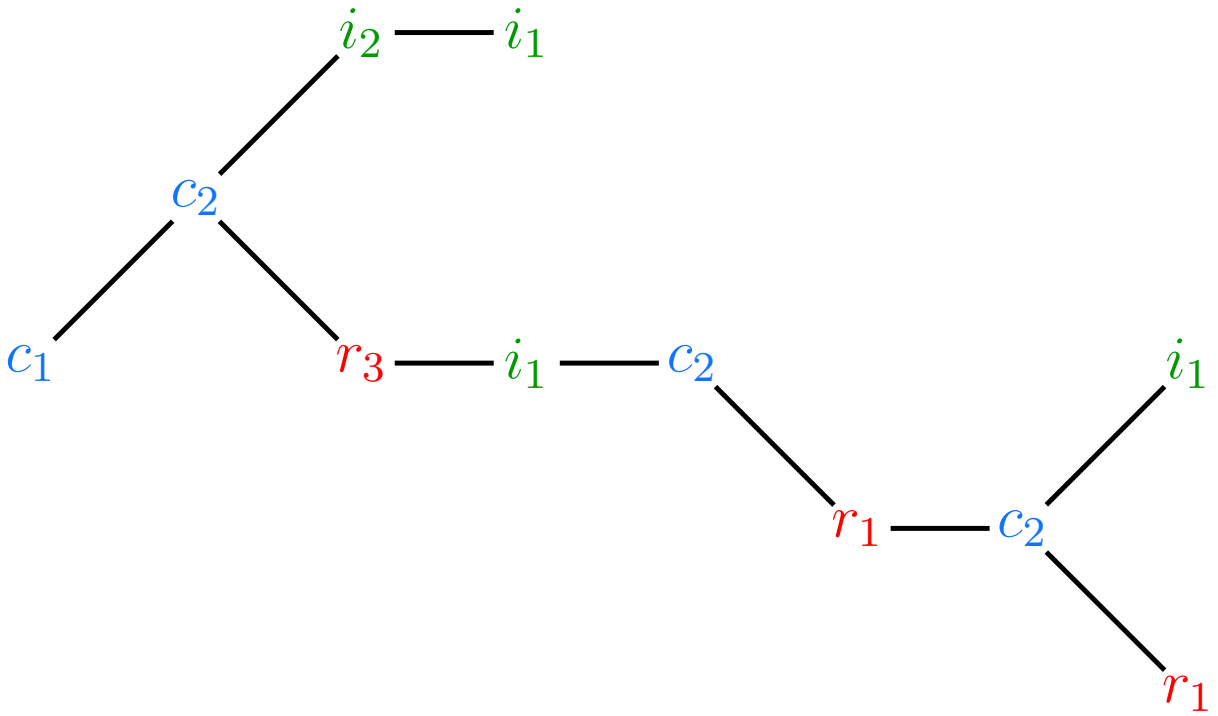


From words to trees: example

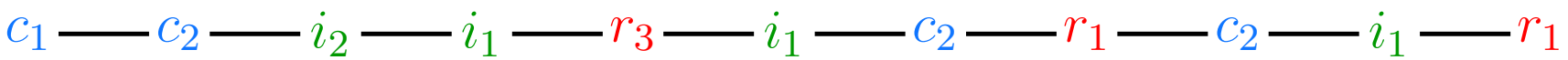


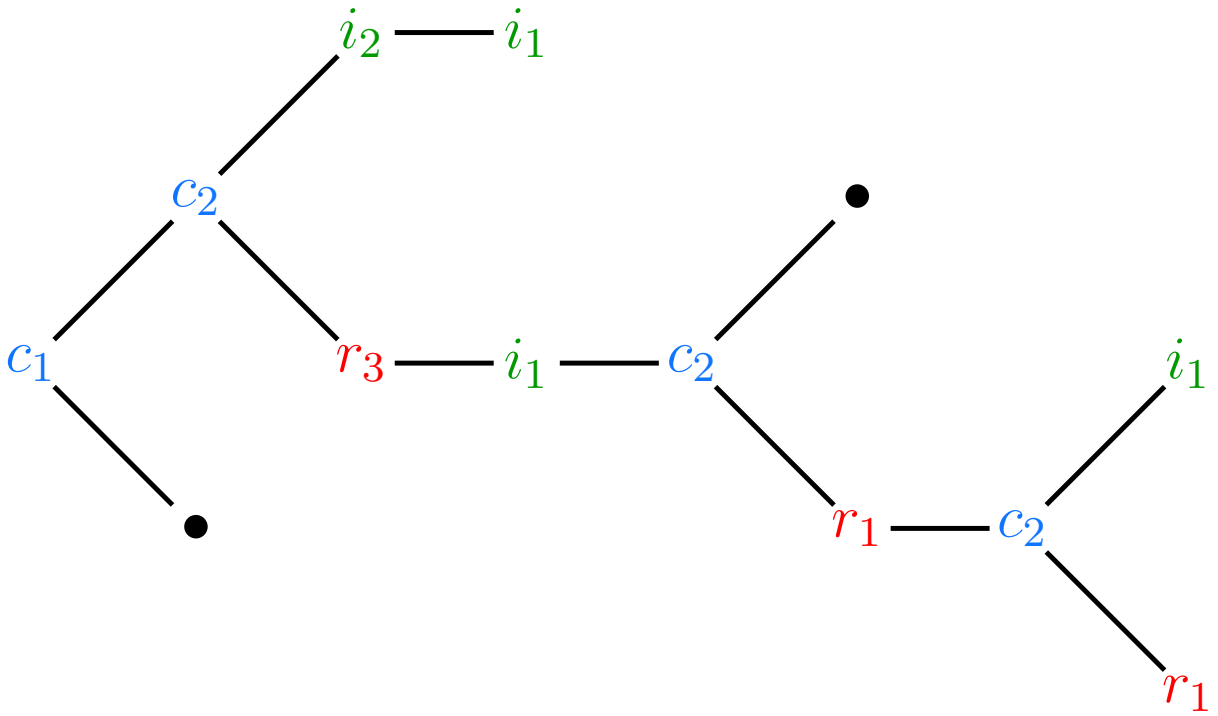
From words to trees: example





From words to trees: example

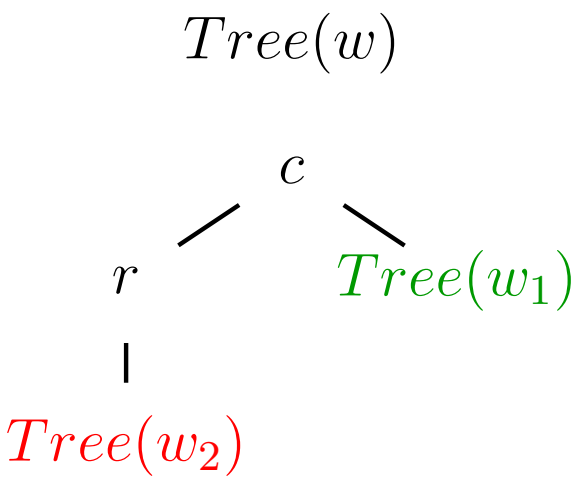




Relation to tree languages

Theorem.[Alur&Madhusudan] *A language L is VPL iff $Tree(L)$ is regular.*

Sketch of Proof. $(\Rightarrow)$ $(q_0, \gamma) \xrightarrow{c} (q_1, \gamma') \xrightarrow{w_1} (q_2, \gamma') \xrightarrow{r} (q_3, \gamma) \xrightarrow{w_2}$



$(\Leftarrow)$ Store the left son's state in the stack and retrieve it later.

Theorem. *A tree language L is regular iff $Streams(L)$ is VPL.*

Restating validation

Strong validation

- Input: a visibly pushdown language L.
- Output: is L regular?

Restating validation

Strong validation

- Input: a visibly pushdown language L.
- Output: is L regular?

Validation

- Input: a visibly pushdown language L.
- Output: is there R Regular such that $L = R \cap L_{Strwmw}$?

Related results

Theorem[Stearns'67]. *Regularity is decidable for visibly pushdown languages.*

Related results

Theorem[Stearns'67]. *Regularity is decidable for visibly pushdown languages.*

Theorem[Bárány, Löding,Serre]. *Given a VPL language L, one can decide whether there exists a regular language R such that*

$$L = R \cap L_{wmw}$$

Related results

Theorem[Stearns'67]. *Regularity is decidable for visibly pushdown languages.*

Theorem[Bárány, Löding,Serre]. *Given a VPL language L, one can decide whether there exists a regular language R such that*

$$L = R \cap L_{wmw}$$

A general problem

- Input: a VPL language K
- Is the following decidable: given a visibly pushdown language L is there R regular such that $L = R \cap K$?