

# **Using Finite Transducers for Describing and Synthesising Structural Time-Series Constraints**

Octobre 2017

Nicolas Beldiceanu  
IMT Atlantique

# Take-away points

- Constraints defined by **function composition**
- New family of constraints for **time-series**
- Applications in **data analysis** as well as **optimization**

# Outline

- **Background**
- Using finite transducers for capturing patterns
- Types of time-series constraints

# Time-series constraints

- Introduced 30 constraints to describe properties of time-series
- Constraints were implemented as **automata with counters**
- Also use **specialized** constraint checkers for analysis
- Hard to correctly implement/**maintain**
- Always **new** constraints to add (*business rules*)

# Key observations (*at the root of this research*)

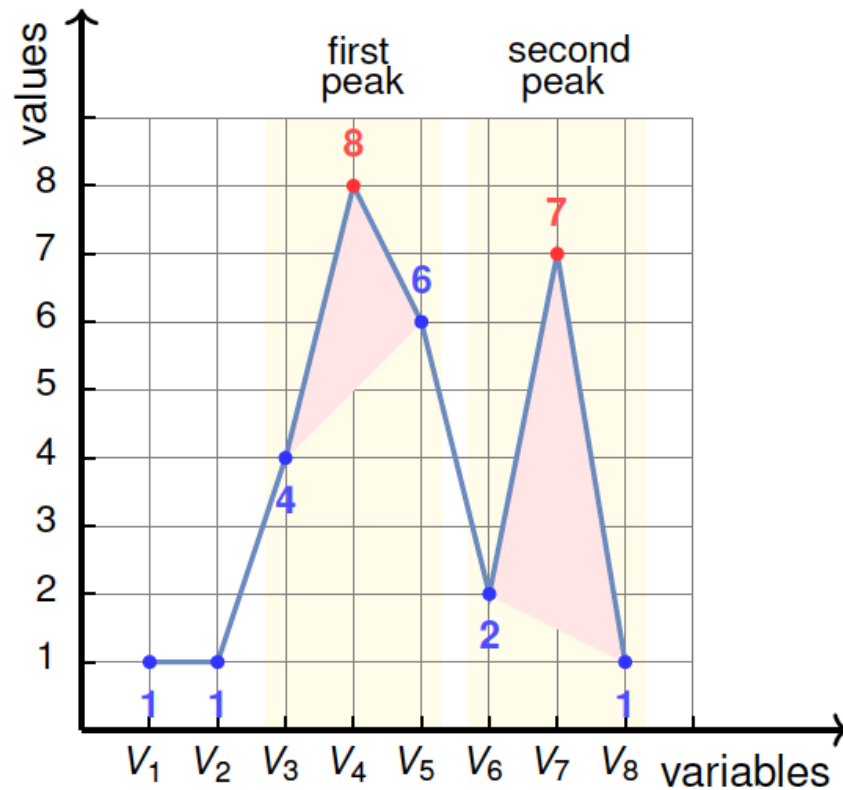
- Almost all automata based constraints of the catalog deal with **restricting a quantity computed** from occurrences of **non-overlapping patterns**
- Transitions of the counter automata have an **implicit semantics** (*steps for identifying a pattern*)

# Key observations (*at the root of this research*)

- Almost all automata based constraints of the catalog deal with **restricting a quantity computed** from occurrences of **non-overlapping patterns**
- Transitions of the counter automata have an **implicit semantics** (*steps for identifying a pattern*)

*Make **explicit** the implicit semantics attached to the transitions by using **the output alphabet of a transducer** that controls the **generation** of the counter updates used for computing the quantity to restrict*

# Example: the peak constraint



peak  $(2, \langle 1, 1, 4, 8, 6, 2, 7, 1 \rangle)$

# Automaton with counters: peak constraint in Global Constraint Catalog

## STATE SEMANTICS

$s$  : stationary/decreasing mode  $(\{> | =\}^*)$   
 $u$  : increasing mode  $(< \{< | =\}^*)$

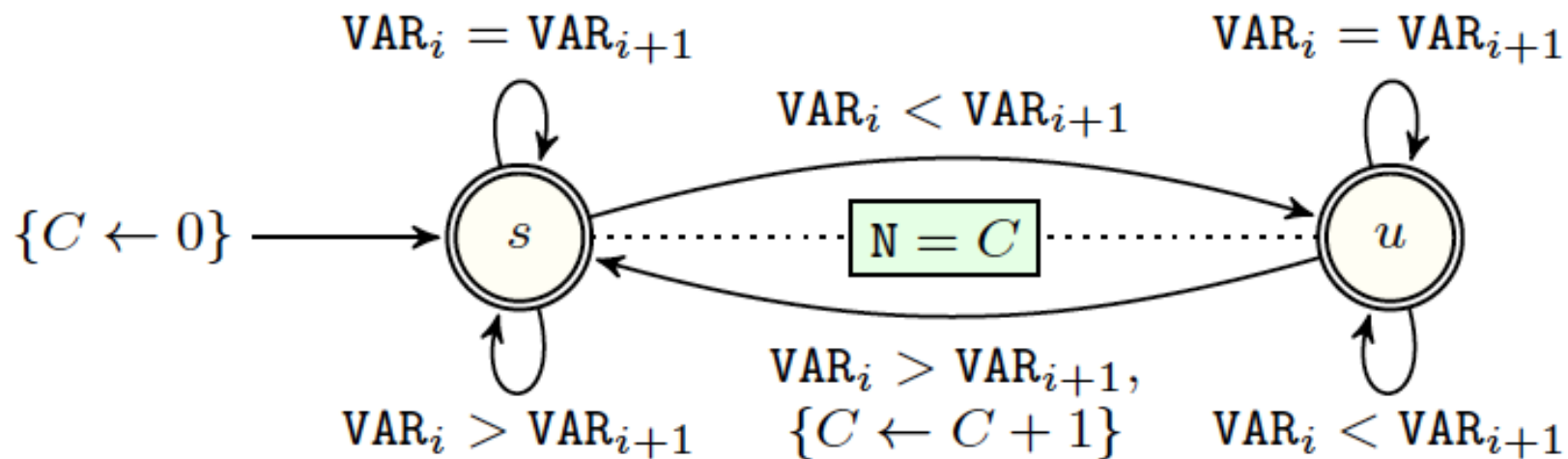


Figure 5.689: Automaton of the PEAK constraint



# Outline

- Background
- **Using finite transducers for capturing patterns**
- Types of time-series constraints

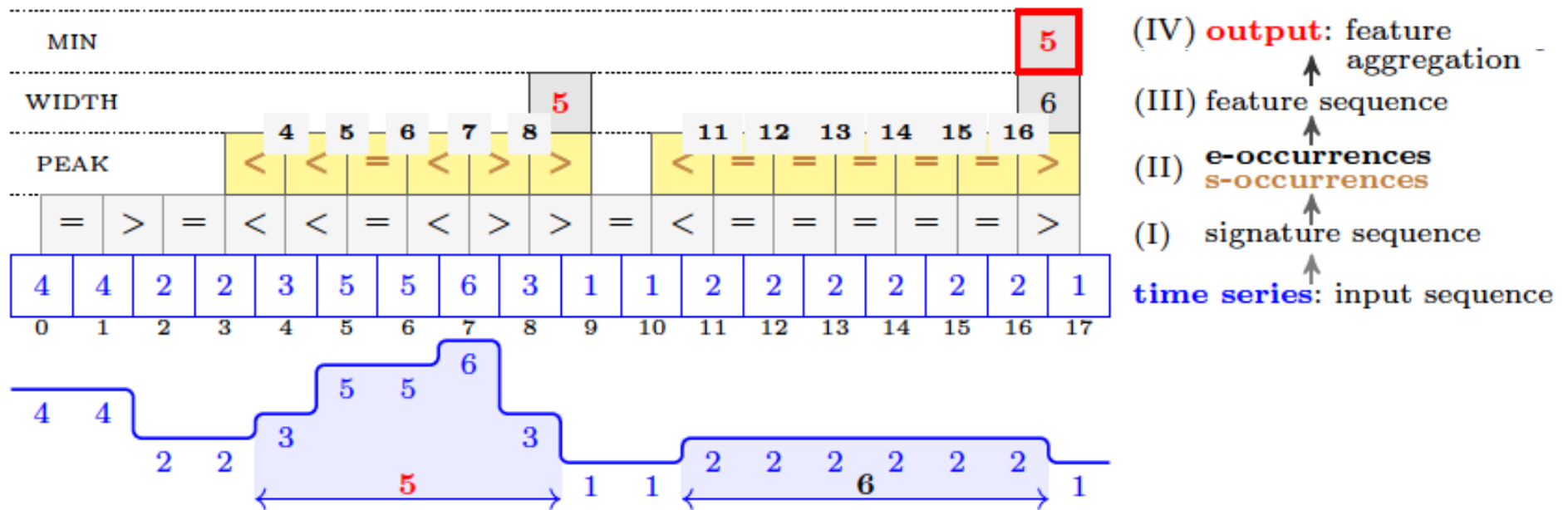
# Work program: define the family of structural time-series constraints

- Define constraints in a **systematic** way **independantly** from a given technology (CP, LP, LS)
- Derive **all** documentation/code **automatically** from compact description
- Apply optimizations **consistently**
- Use for **multiple** use cases

# Decomposing the definition of a constraint

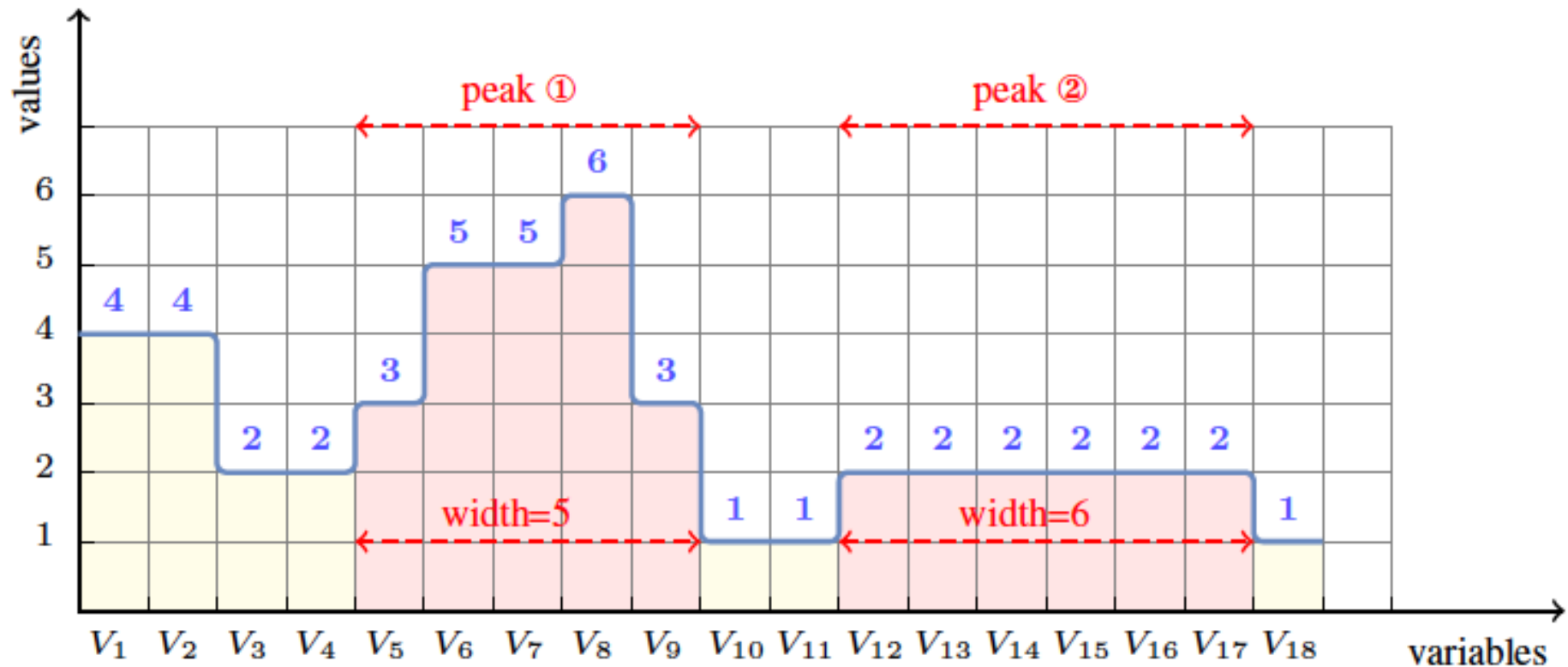
- Constraints are **pure functional dependencies** or **predicates** (*see predicates later on*)
- Implemented as **automata with counters**
- **Four steps** (layers) in definition
  - Building **signature**
  - Recognize **pattern occurrences** in sequence
  - Extract **feature** per pattern
  - **Aggregate** features

# Example: min\_width\_peak



`MIN_WIDTH_PEAK(5, <4, 4, 2, 2, 3, 5, 5, 6, 3, 1, 1, 2, 2, 2, 2, 2, 2, 1>)`

# Example: min\_width\_peak (continued)



`MIN_WIDTH_PEAK(5, <4, 4, 2, 2, 3, 5, 5, 6, 3, 1, 1, 2, 2, 2, 2, 2, 2, 1>)`

# Signature

- Convert (integer) value to finite alphabet
- Signature links two consecutive entries in time-series
- We use  $<, =, >$  with their natural semantics
- Other signatures possible

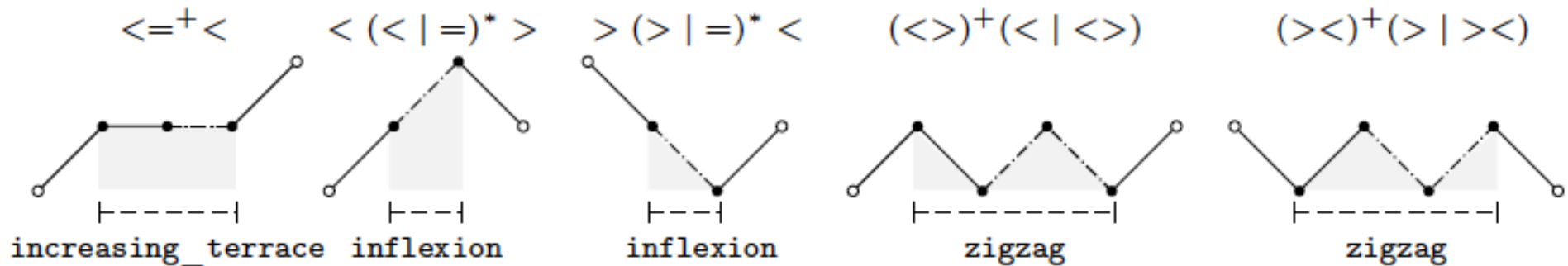
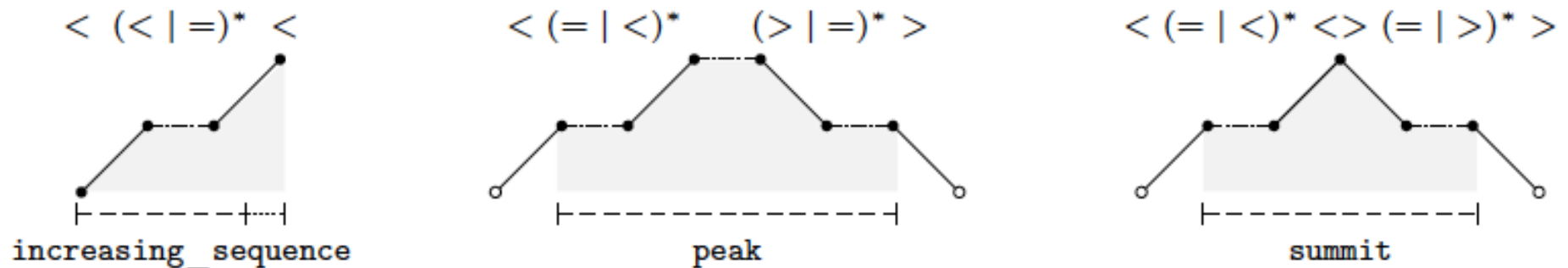
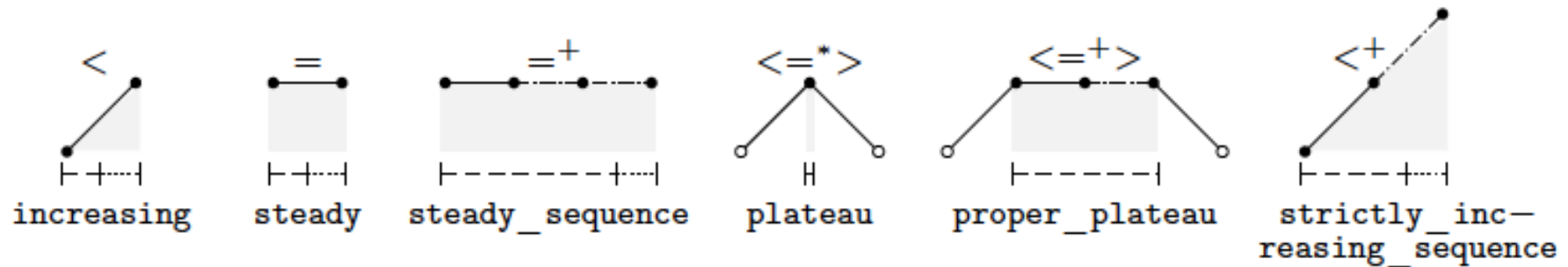
# Patterns

- **Identify a pattern** we are looking for  $r$
- **Extract subpart** for which computes feature  $a, b$

pattern	regular expression $r$	before $b$	after $a$
increasing	$<$	0	0
increasing_sequence	$< (<   =)^* <   <$	0	0
increasing_terrace	$< =^+ <$	1	1
summit	$(<   (< (=   <)^* <)) (>   (> (=   >)^* >))$	1	1
plateau	$< =^* >$	1	1
proper_plateau	$< =^+ >$	1	1
strictly_increasing_sequence	$<^+$	0	0
peak	$< (=   <)^* (>   =)^* >$	1	1
inflexion	$< (<   =)^* >   > (>   =)^* <$	1	1
steady	$=$	0	0
steady_sequence	$=^+$	0	0
zigzag	$(< >)^+ (<   < >)   (> <)^+ (>   > <)$	1	1

Find **maximal** words matching regular expression  $r$

# Patterns (continued)





# Definition of $\{s|i|e\}$ -occurrences of an occurrence of pattern

Given

an input sequence  $X_0, X_1, \dots, X_{n-1}$ ,

its signature sequence  $S_0, S_1, \dots, S_{n-2}$ ,

a pattern  $(r, a, b)$ ,

a non-empty signature subsequence  $s_i, s_{i+1}, \dots, s_j$

forming a maximum word matching  $r$

the  $s$ -occurrence  $(i..j)$  is the index sequence  $i, \dots, j$ ,

the  $i$ -occurrence  $[(i+b)..j]$  is the index sequence  $i+b, \dots, j$ ,

the  $e$ -occurrence  $[(i+b)..(j+1-a)]$  is the index sequence  
 $i+b, \dots, j+1-a$ .

# Definition of $\{s|i|e\}$ -occurrences of an occurrence of pattern

Given

an input sequence  $X_0, X_1, \dots, X_{n-1}$ ,  
its signature sequence  $S_0, S_1, \dots, S_{n-2}$ ,  
a pattern  $(r, a, b)$ ,  
a non-empty signature subsequence  $s_i, s_{i+1}, \dots, s_j$   
forming a maximum word matching  $r$

the  $s$ -occurrence  $(i..j)$  is the index sequence  $i, \dots, j$ ,  
the  $i$ -occurrence  $[(i+b)..j]$  is the index sequence  $i+b, \dots, j$ ,  
the  $e$ -occurrence  $[(i+b)..(j+1-a)]$  is the index sequence  
 $i+b, \dots, j+1-a$ .

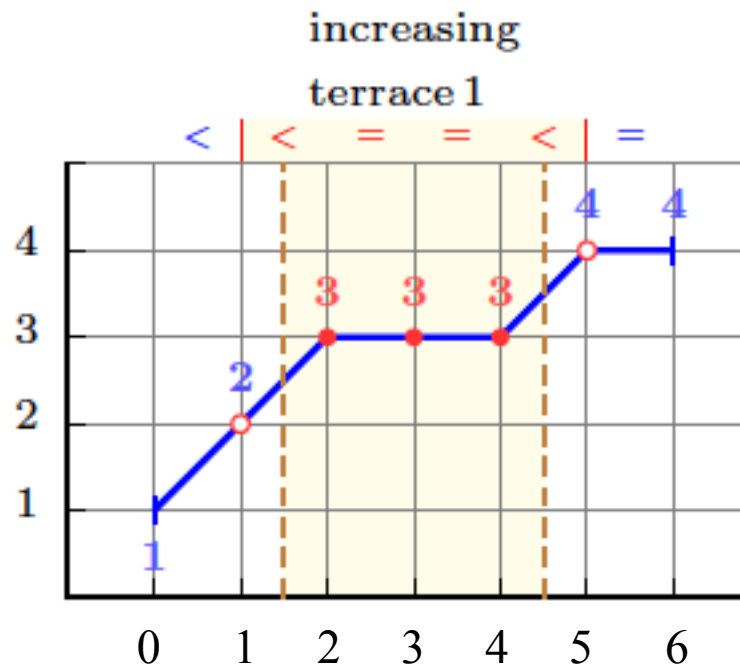
$s$ -occurrences : maximal signature sequence matching  $r$

$i$ -occurrences : do not overlap (footprint of the pattern)

$e$ -occurrences : used to compute the feature value

# Example of {s|i|e}-occurrences for the **increasing\_terrace** pattern

pattern	regular expression $r$	before $b$	after $a$
increasing terrace	$<=^+<$	1	1



## Indices of

s-occurrences : **1..4** ( $<==<$ )

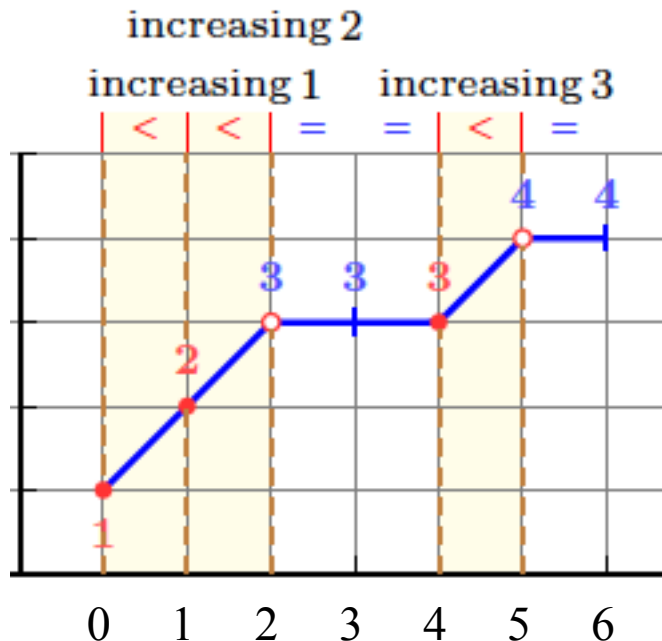
i-occurrences : **[2..4]** (333)

e-occurrences : **[[2..4]]** (333)

$a=1=b$  excludes first and last input values **2** and **4**

# Example of $\{s|i|e\}$ -occurrences for the **increasing** pattern

pattern	regular expression $r$	before $b$	after $a$
increasing	$<$	0	0



## Indices of

s-occurrences : **0..0**      **1..1**      **4..4**

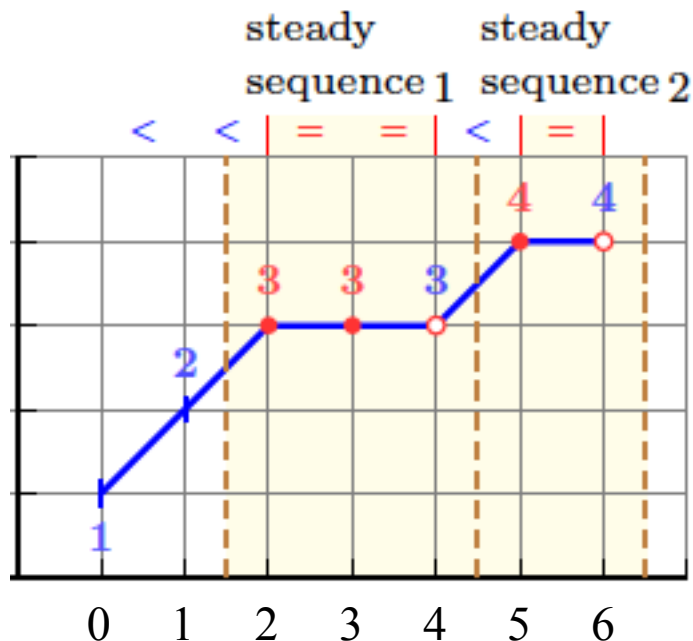
$i$ -occurrences : **[0..0]**      **[1..1]**      **[4..4]**

e-occurrences : **[[0..1]]**      **[[1..2]]**      **[[4..5]]**

since  $b=0$  s-occurrences and  $i$ -occurrences match  
 since  $a=0=b$  the 1 and 2 input values are part of e-occurrences

# Example of $\{s|i|e\}$ -occurrences for the **steady\_sequence** pattern

pattern	regular expression $r$	before $b$	after $a$
steady_sequence	$\cdot =^+ \cdot$	0	0



## Indices of

s-occurrences : **2..3**      **5..5**  
*i*-occurrences : **[2..3]**      **[5..5]**  
e-occurrences : **[[2..4]]**      **[[5..6]]**

since  $b=0$  s-occurrences and *i*-occurrences match  
since  $a=0=b$  the 1 and 2 input values are part of e-occurrences

# Features (*computed from e-occurrences*)

- `one` : value 1
- `width` : number of positions of the e-occurrence
- `surf` : sum of the values of the e-occurrence
- `max` : maximum value of the e-occurrence
- `min` : minimum value of the e-occurrence
- `range` : range of the e-occurrence: `max-min`

# Aggregators (*computed from sequence of features*)

- `max` : largest value of a sequence of features
- `min` : smallest value of a sequence of features
- `sum` : sum of the features of a sequence of feature

# Device for recognizing *i*-occurrences of a pattern: a seed transducer

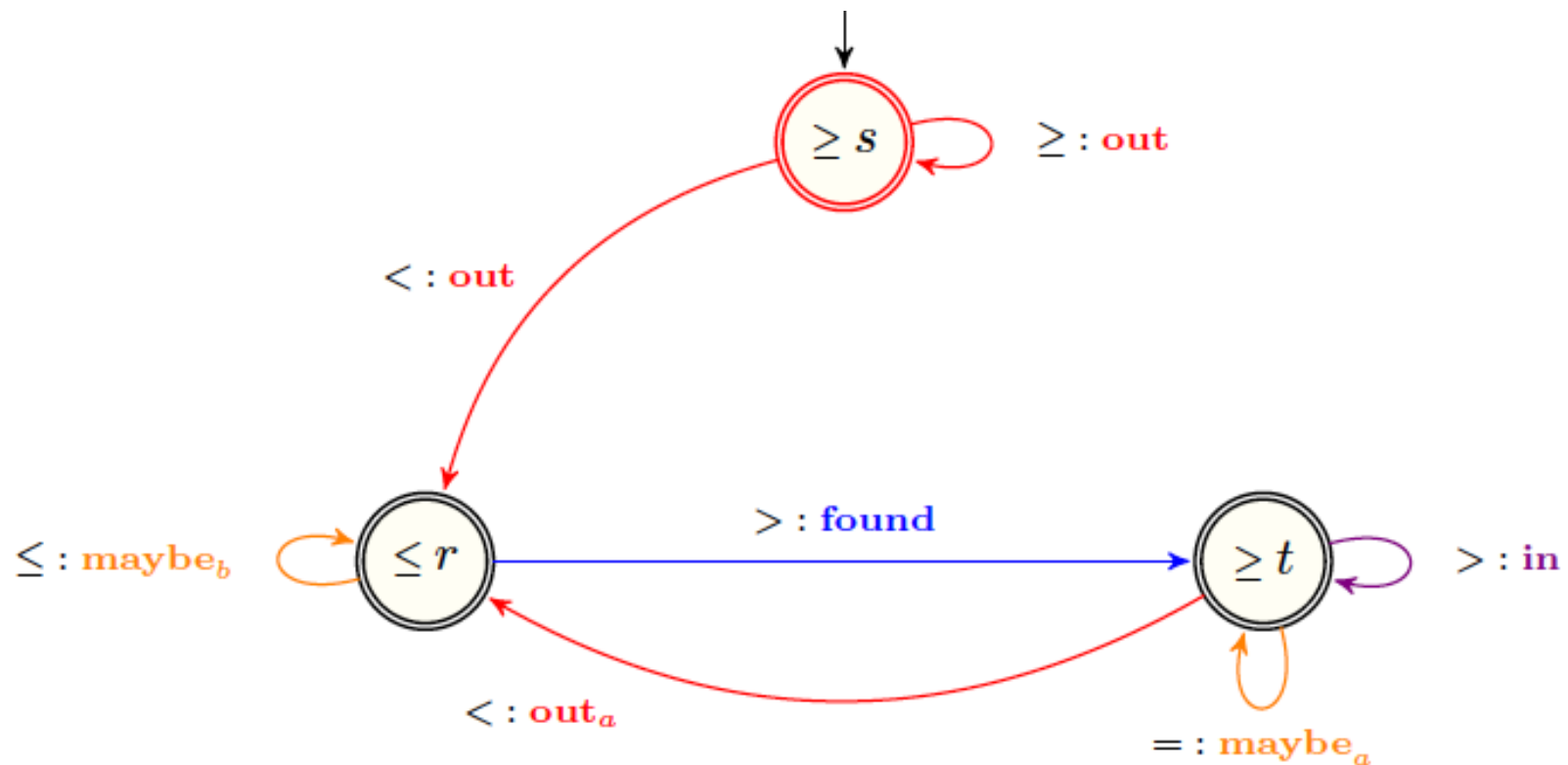
- Define a pattern by a **transducer**  
(*reading/writing regular language*)
- **Input**: signature sequence
- **Output**: word of a semantic alphabet with letters:
  - **out** we are **outside** the pattern
  - **maybe** we are **possibly in** the pattern  
(*must be confirmed later on*)
  - **found** **first place** we know **we are in** the pattern
  - **in** we are **still** in the pattern

**everything** *will be synthesized from the seed transducer*

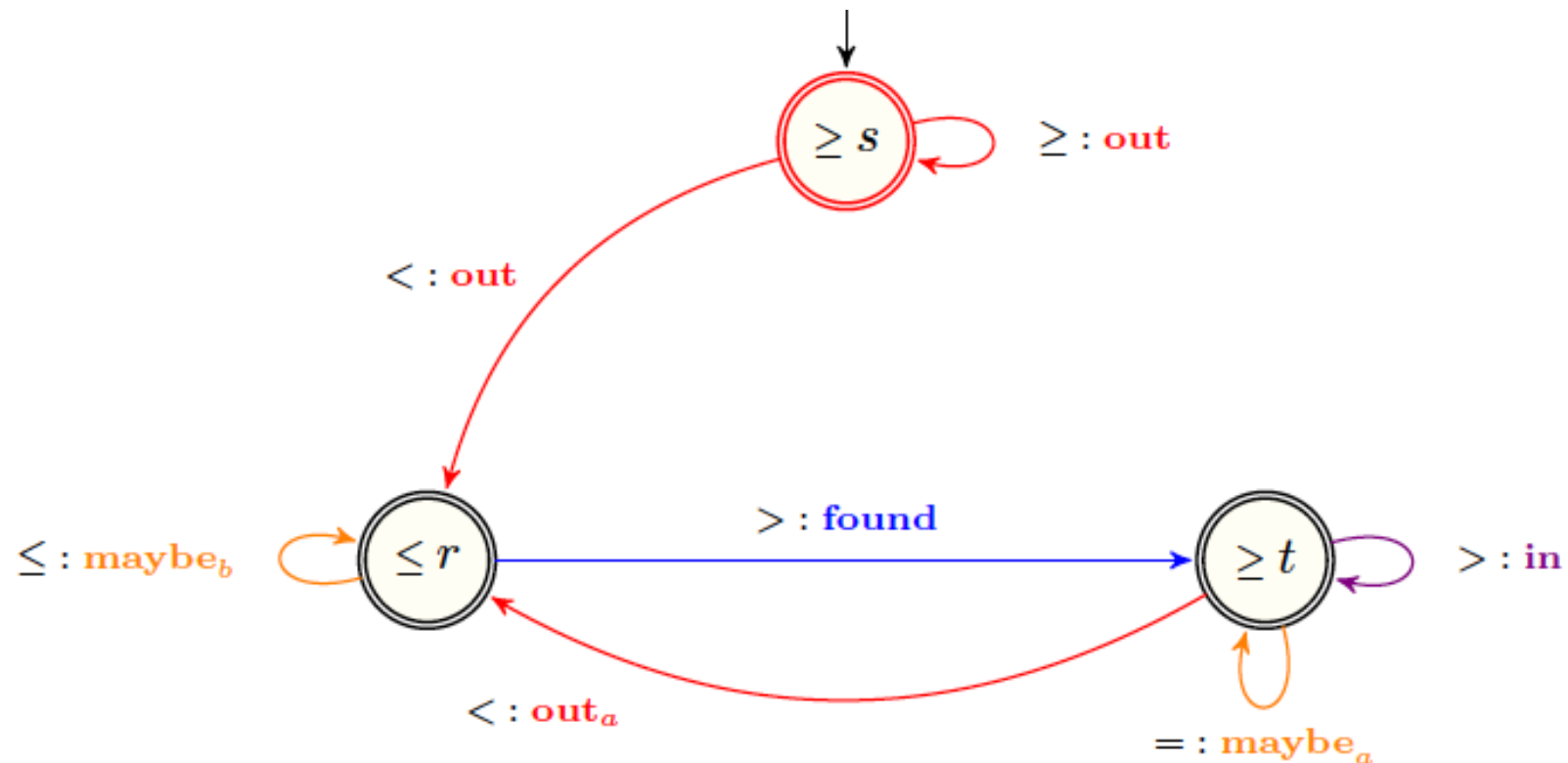


# Example: transducer for the **peak** pattern

pattern	regular expression $r$	before $b$	after $a$
peak	$< (=   <)^* (>   =)^* >$	1	1



# Example: transducer for the **peak** pattern



	o	o	o	o	m <sub>b</sub>	m <sub>b</sub>	m <sub>b</sub>	f	in	m <sub>a</sub>	o <sub>a</sub>	m <sub>b</sub>	m <sub>b</sub>	m <sub>b</sub>	m <sub>b</sub>	m <sub>b</sub>	f	output: semantic string	
	=	>	=	<	<	=	<	>	>	=	<	=	=	=	=	=	>	input: signature string	
states:	s	s	s	s	r	r	r	r	t	t	t	r	r	r	r	r	r	t	
	4	4	2	2	3	5	5	6	3	1	1	2	2	2	2	2	2	1	input: integer sequence

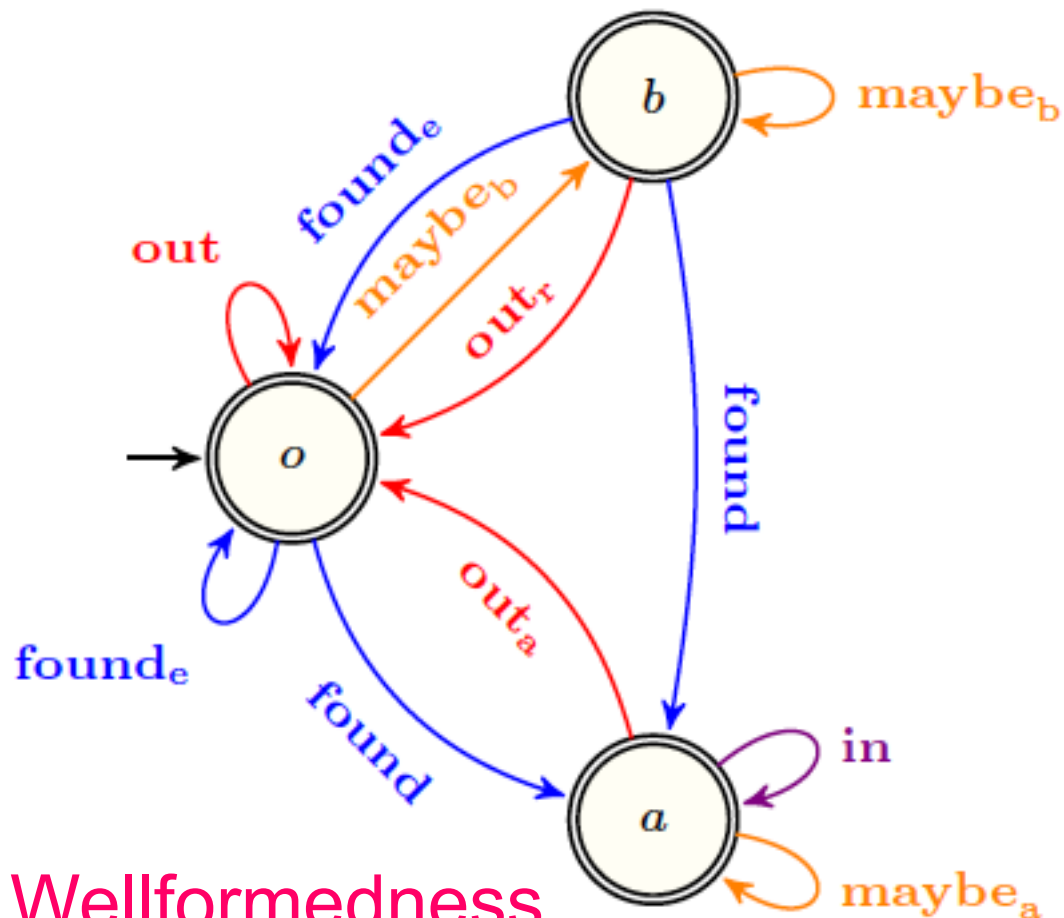
# Well-formed seed transducer (*language of the output*)

state semantics

$o$  : outside or after the end of a pattern

$b$  : potentially inside (before a **found**/**found<sub>e</sub>**)

$a$  : potentially inside (after a **found**)



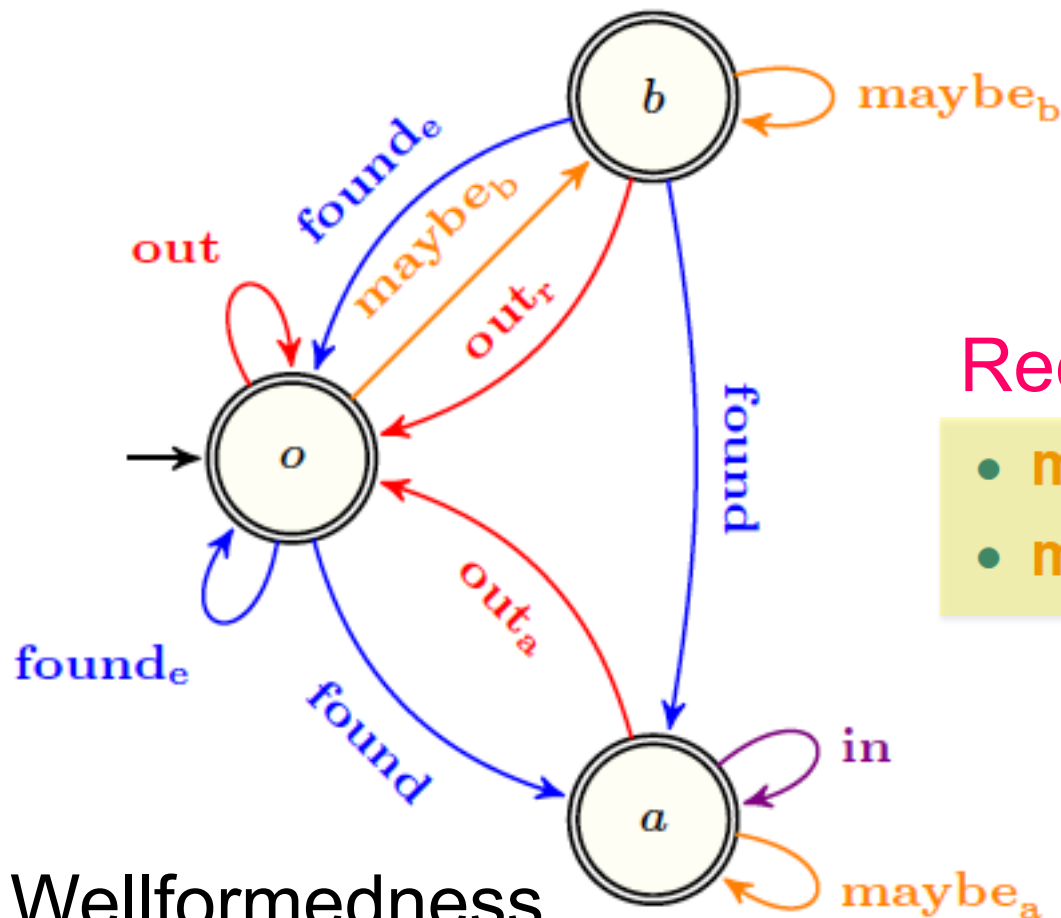
# Well-formed seed transducer (*language of the output*)

state semantics

$o$  : outside or after the end of a pattern

$b$  : potentially inside (before a **found**/**found<sub>e</sub>**)

$a$  : potentially inside (after a **found**)



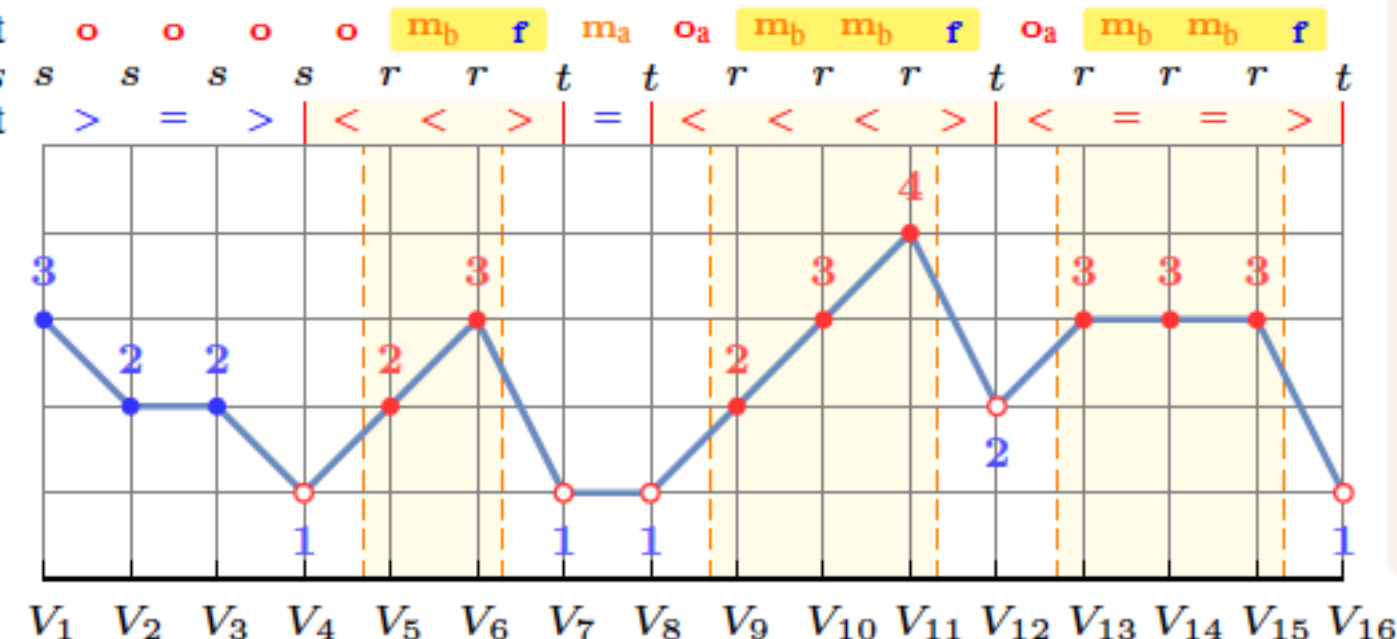
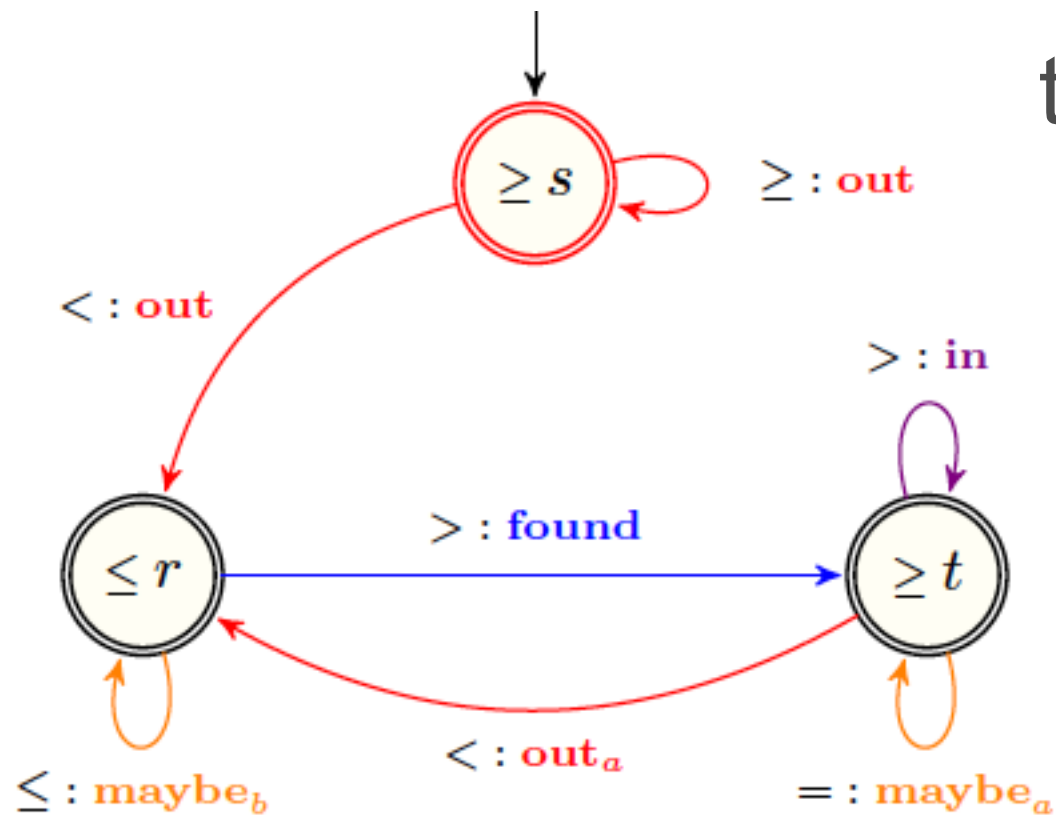
Recognizing pattern

- $\text{maybe}_b^* \text{found}_e$
- $\text{maybe}_b^* \text{found} \{ \text{maybe}_a^* \text{in}^+ \}^*$

Wellformedness

# transducer for peak

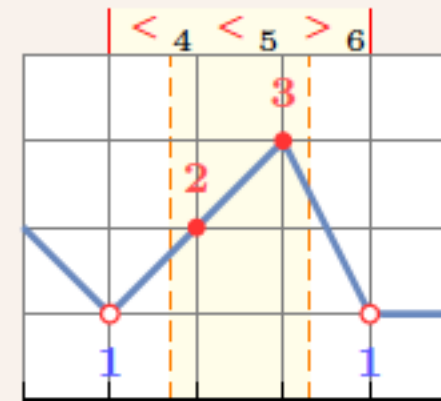
(A)



(C)

s-occurrence

$(i..j) = (4..6)$



$V_5 \quad V_6$

i-occurrence (•)

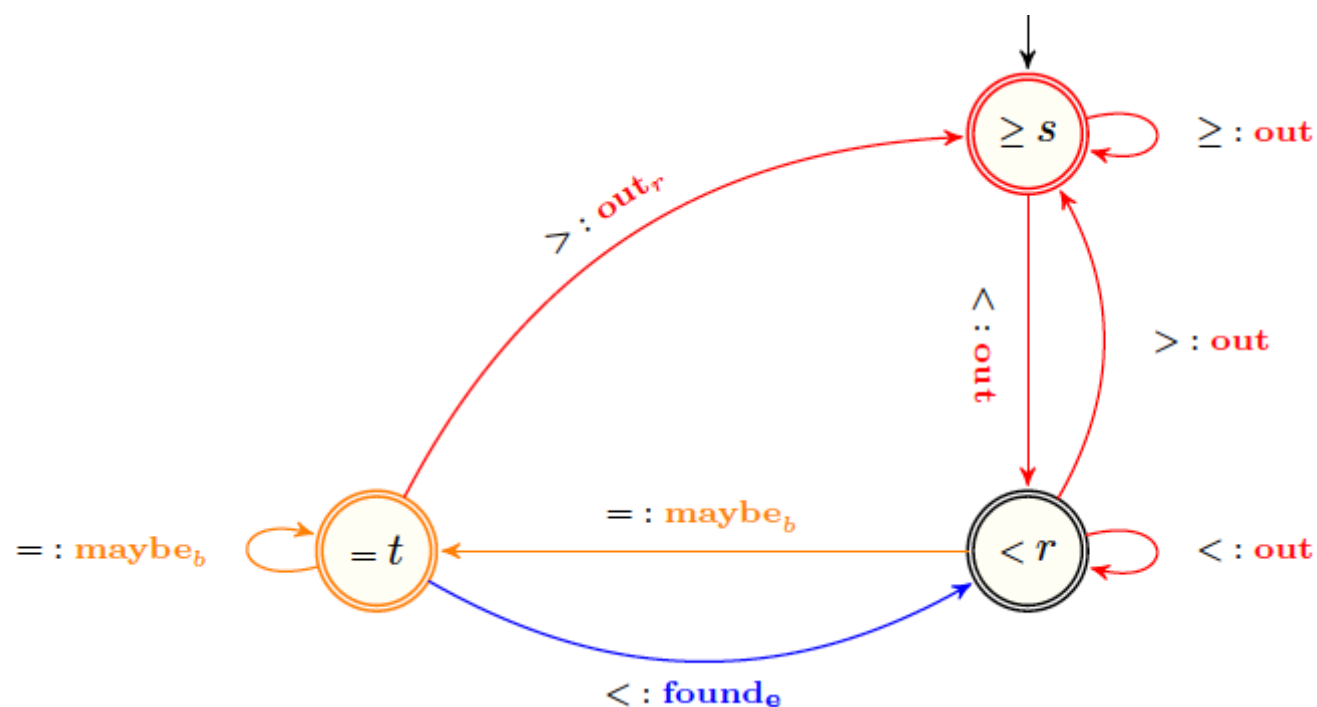
$[(i + \text{before})..j] = [(4 + 1)..6]$

e-occurrence (•)

$[[ (i + \text{before})..(j + 1 - \text{after}) ]]$   
 $= [[ (4 + 1)..(6 + 1 - 1) ]]$

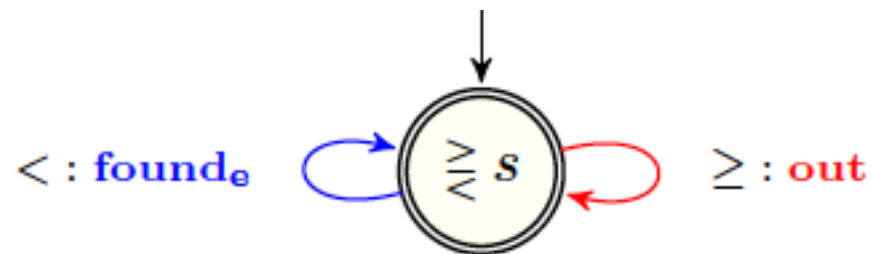
# Transducer for **increasing\_terrace**

pattern	regular expression $r$	before $b$	after $a$
increasing terrace	$<=^+ <$	1	1



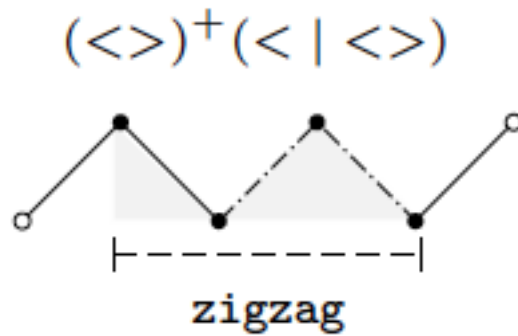
# Transducer for **increasing**

pattern	regular expression $r$	before $b$	after $a$
increasing	$<$	0	0

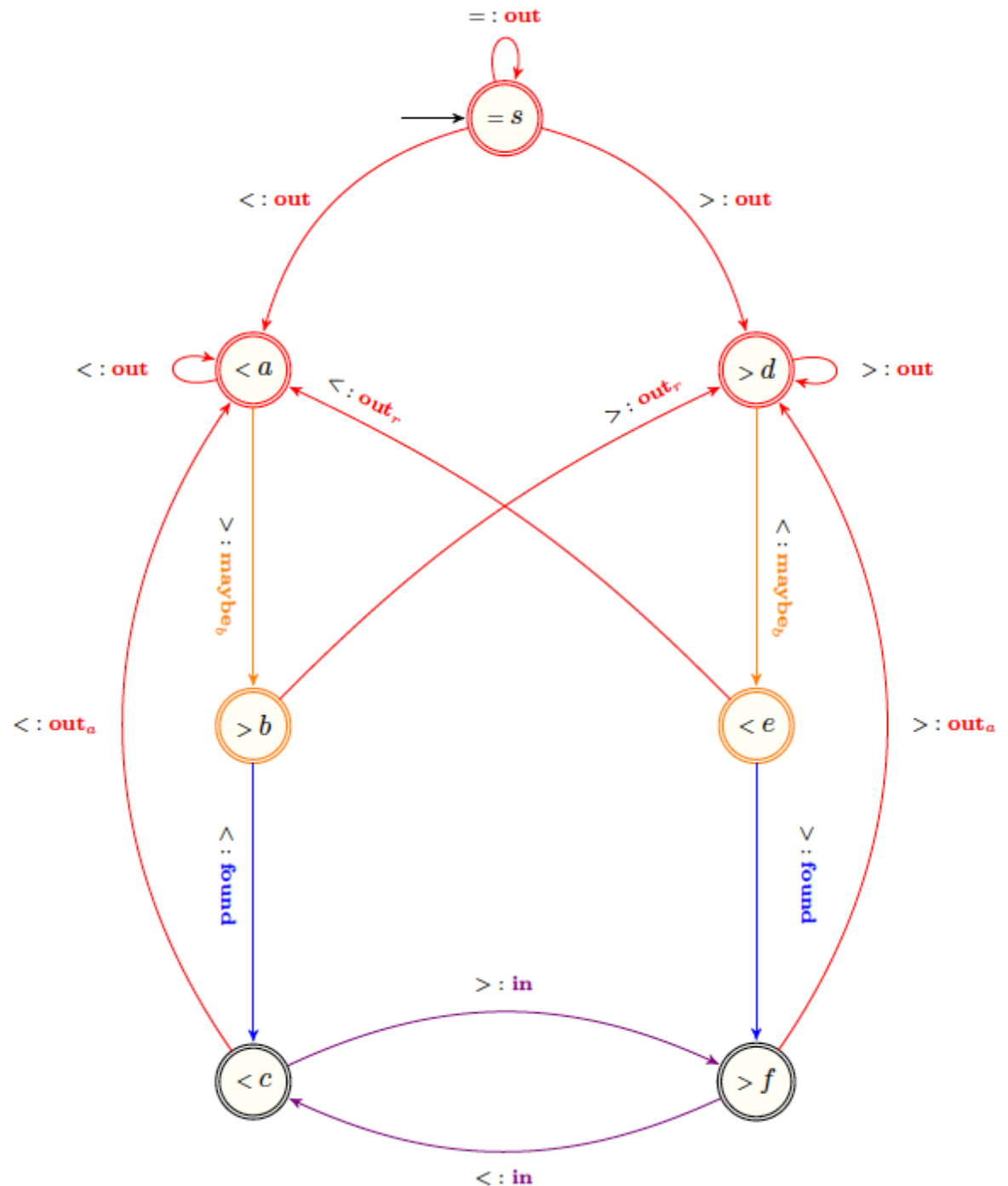
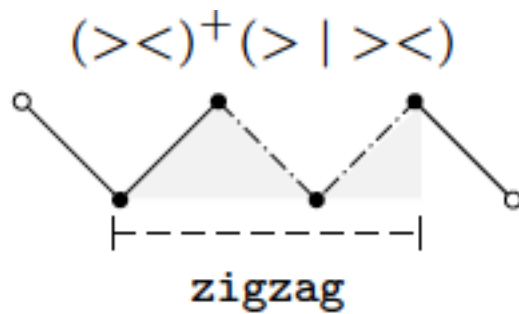


# Transducer for zigzag

$$a=1=b$$



or

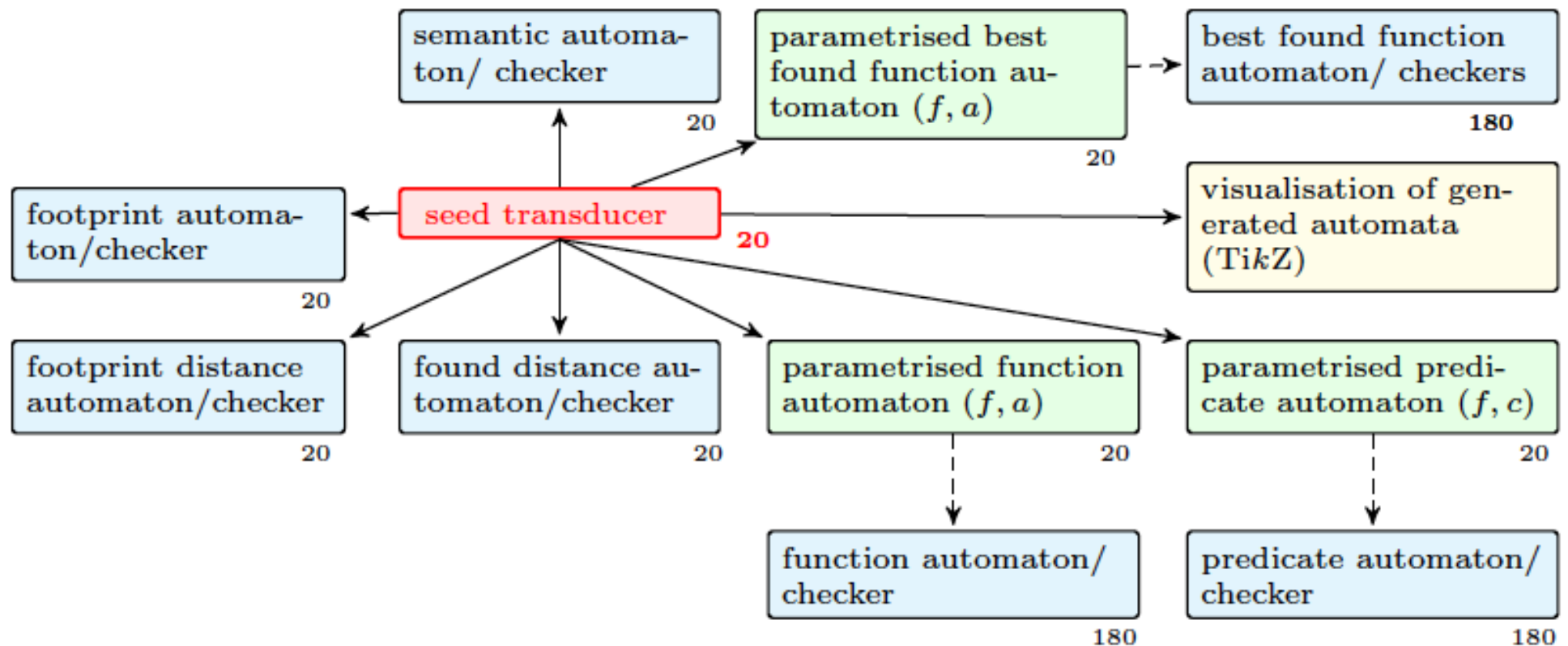




# Outline

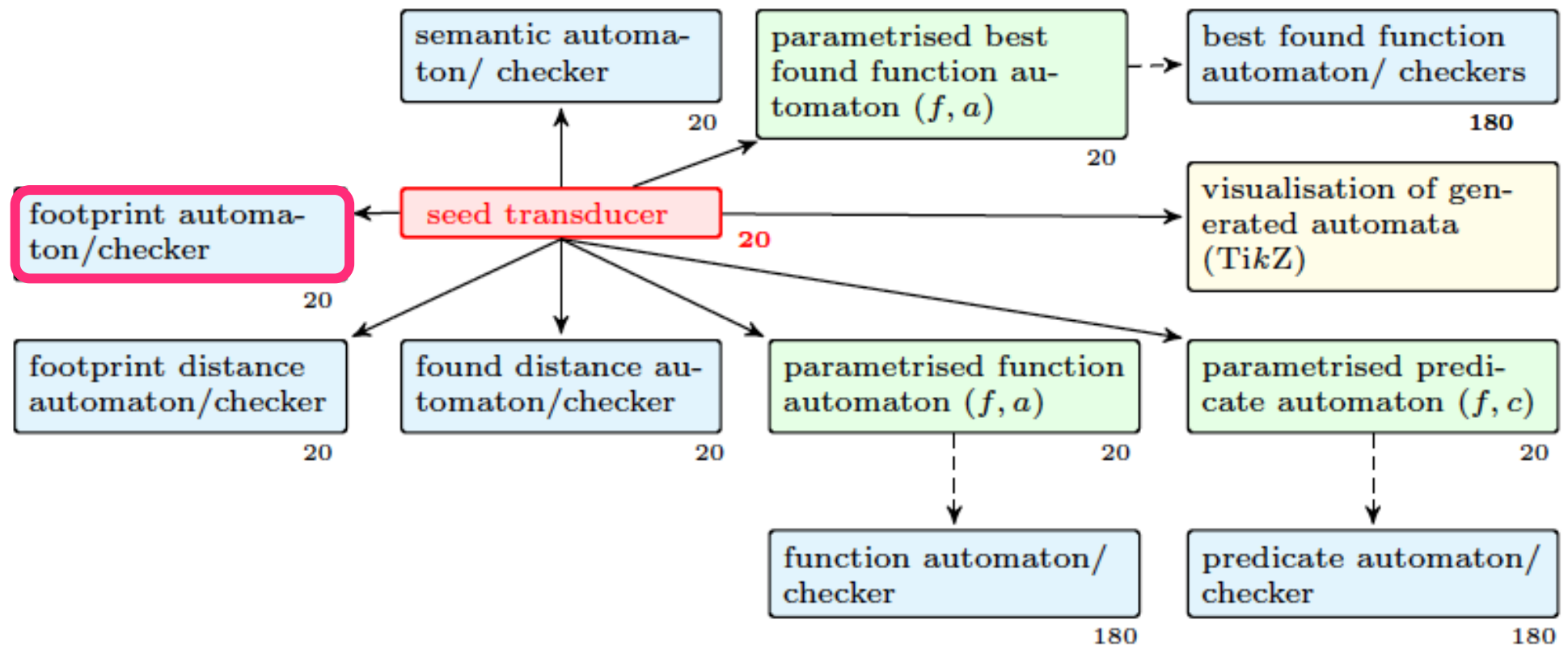
- Background
- Using finite transducers for capturing patterns
- **Types of time-series constraints**

# Producing different types of constraints



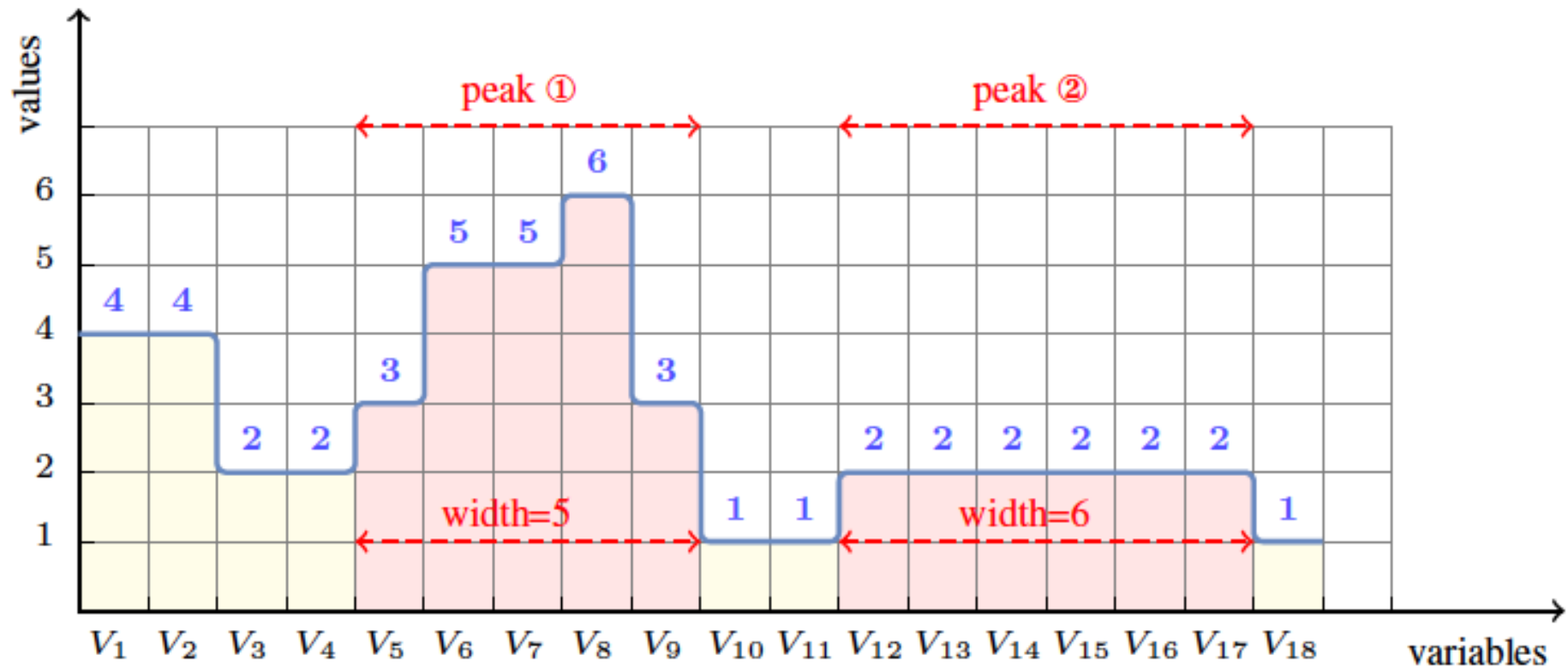
use an appropriate decoration table for synthesizing the corresponding automaton with counters

# Identifying where the occurrences of a pattern are



use an appropriate decoration table for synthesizing  
the corresponding automaton with counters

# Footprint constraint: identifying $i$ -occurrences of a pattern

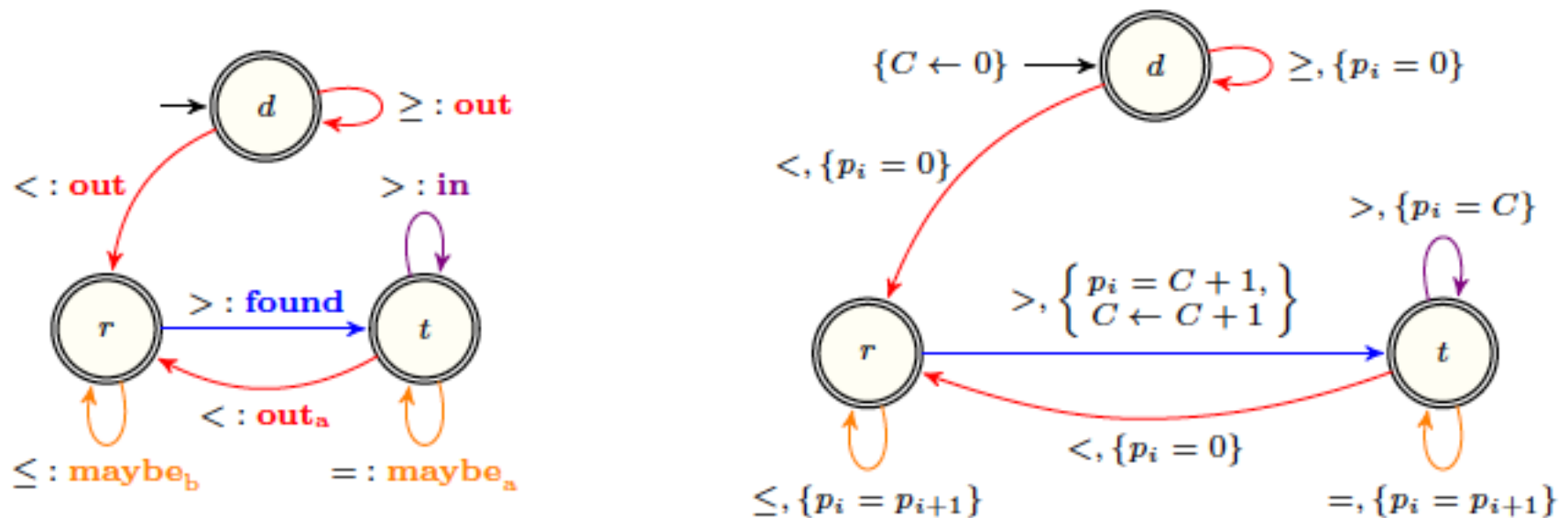


footprint(peak, [4,4,2,2,3,5,5,6,3,1,1,2,2,2,2,2,2,1],  
[0,0,0,0,1,1,1,1,1,0,0,2,2,2,2,2,2,0] )

# Decoration table for the footprint constraint (*generating counters updates*)

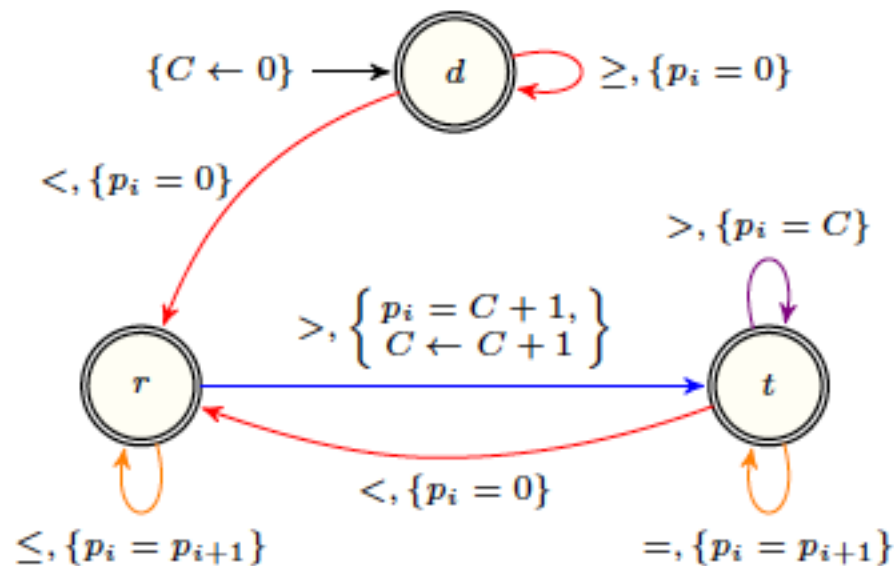
<b>initialisation</b>	$C \leftarrow 0$
<b>return</b>	$p_n = 0$
<b>semantic letters</b>	<b>annotations</b>
	<i>guard</i> <i>update of C</i>
<b>out</b>	$p_i = 0$
<b>out<sub>r</sub></b>	$p_i = 0$
<b>out<sub>a</sub></b>	$p_i = 0$
<b>maybe<sub>b</sub></b>	$p_i = p_{i+1}$
<b>maybe<sub>a</sub></b>	$p_i = p_{i+1}$
<b>found<sub>e</sub></b>	$p_i = C + 1 \quad C \leftarrow C + 1$
<b>found</b>	$p_i = C + 1 \quad C \leftarrow C + 1$
<b>in</b>	$p_i = C$

# Example: synthesizing the footprint constraint for the **peak** pattern



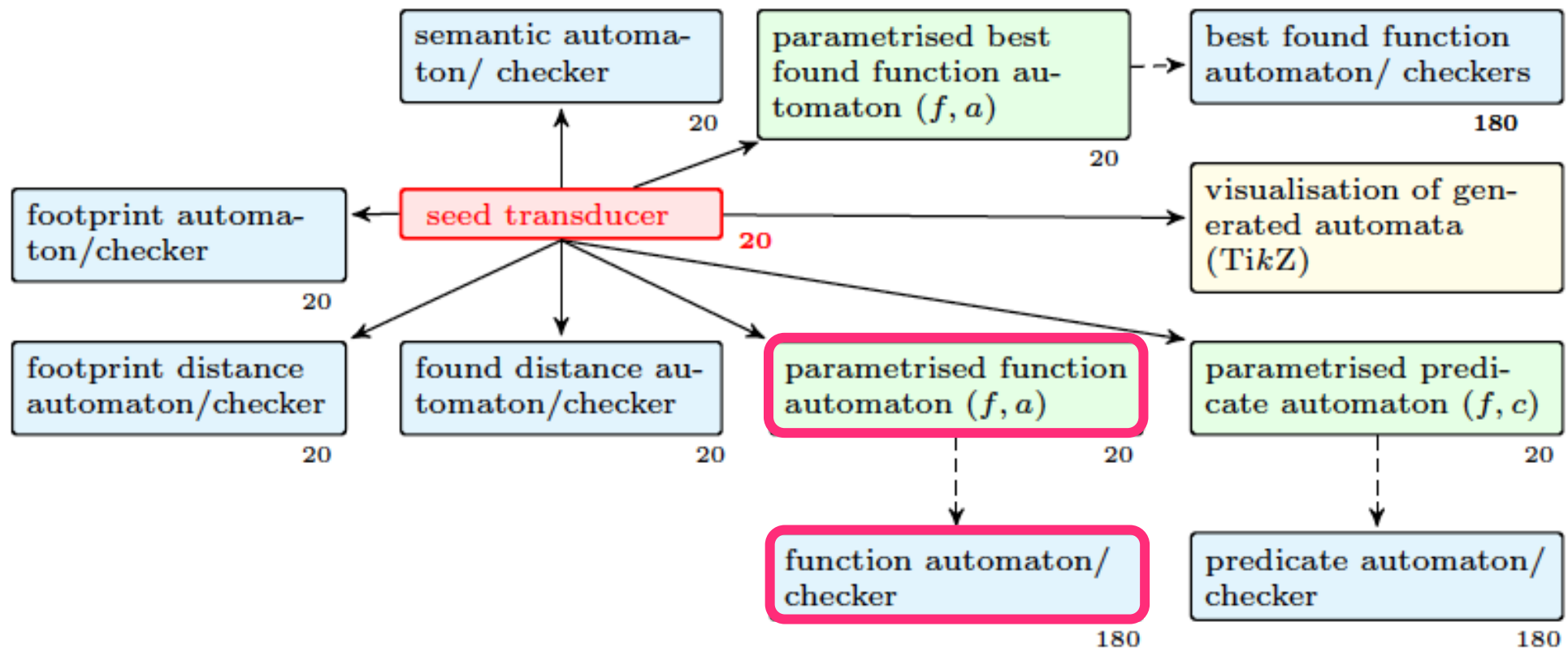
<b>out</b>	$p_i = 0$
<b>out<sub>r</sub></b>	$p_i = 0$
<b>out<sub>a</sub></b>	$p_i = 0$
<b>maybe<sub>b</sub></b>	$p_i = p_{i+1}$
<b>maybe<sub>a</sub></b>	$p_i = p_{i+1}$
<b>found<sub>e</sub></b>	$p_i = C + 1 \quad C \leftarrow C + 1$
<b>found</b>	$p_i = C + 1 \quad C \leftarrow C + 1$
<b>in</b>	$p_i = C$

# Example: executing the synthesized footprint automaton of the **peak** pattern



	$p_0 = 0$ $p_1 = 0$ $p_2 = 0$ $p_3 = 0$ $p_4 = p_5$ $p_5 = p_6$ $p_6 = p_7$ $p_7 = 1$ $p_8 = 1$ $p_9 = p_{10}$ $p_{10} = 0$ $p_{11} = p_{12}$ $p_{12} = p_{13}$ $p_{13} = p_{14}$ $p_{14} = p_{15}$ $p_{15} = p_{16}$ $p_{16} = 2$ $p_{17} = 0$																	
$C_i$	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	
$\tau_i$	$o$	$o$	$o$	$o$	$m_b$	$m_b$	$m_b$	$f$	$in$	$m_a$	$o_a$	$m_b$	$m_b$	$m_b$	$m_b$	$m_b$	$f$	
$q_i$	$d$	$d$	$d$	$d$	$r$	$r$	$r$	$r$	$t$	$t$	$t$	$r$	$r$	$r$	$r$	$r$	$t$	
$s_i$	$=$	$>$	$=$	$<$	$<$	$=$	$<$	$>$	$>$	$=$	$<$	$=$	$=$	$=$	$=$	$=$	$>$	
$x_i$	4	4	2	2	3	5	5	6	3	1	1	2	2	2	2	2	1	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

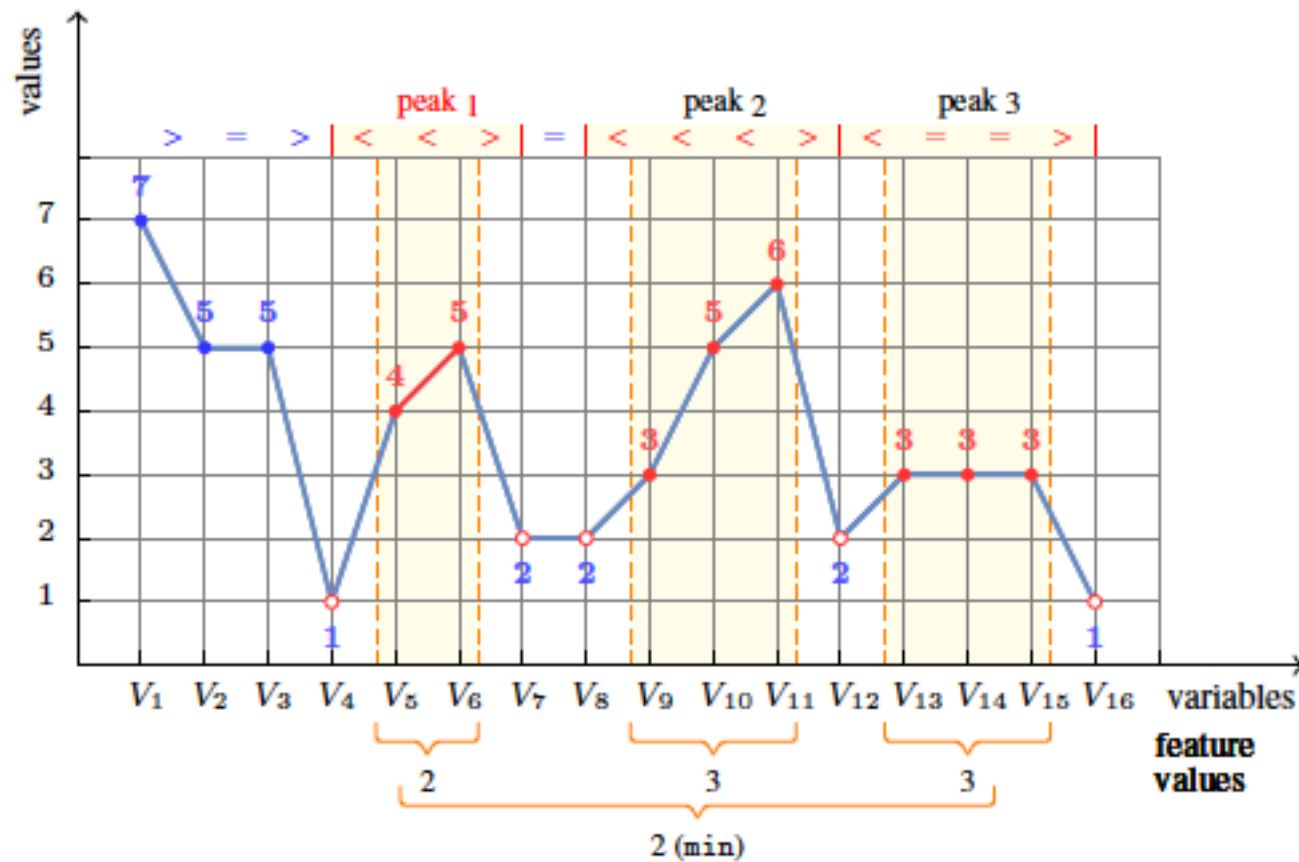
# Restricting feature values of occurrences of a pattern



use an appropriate decoration table for synthesizing the corresponding automaton with counters



# Feature constraints (example)



MIN\_WIDTH\_PEAK (2, [7, 5, 5, 1, 4, 5, 2, 2, 3, 5, 6, 2, 3, 3, 3, 1])

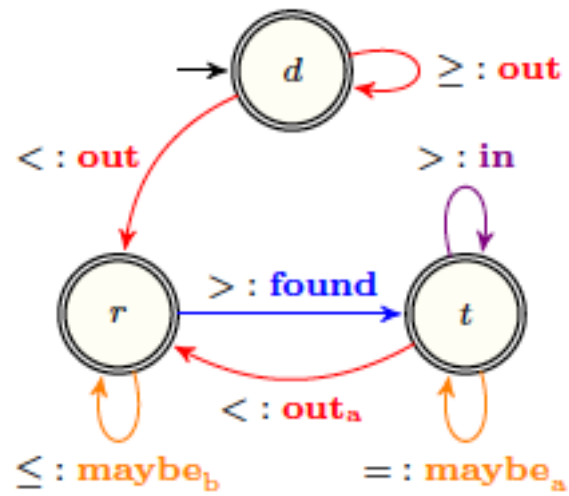
# Decoration table for the feature constraint

Feature $f$	$\text{neutral}_f$	$\text{min}_f$	$\text{max}_f$	$\phi_f$	$\delta_f$
one	1	1	1	max	0
width	0	0	$n$	+	1
surface	0	$-\infty$	$+\infty$	+	$x_t$
max	$-\infty$	$-\infty$	$+\infty$	max	$x_t$
min	$+\infty$	$-\infty$	$+\infty$	min	$x_t$
range	0	0	$+\infty$		$x_t$

Aggregator $g$	$\text{default}_{g,f}$
Max	$\text{min}_f$
Min	$\text{max}_f$
Sum	0

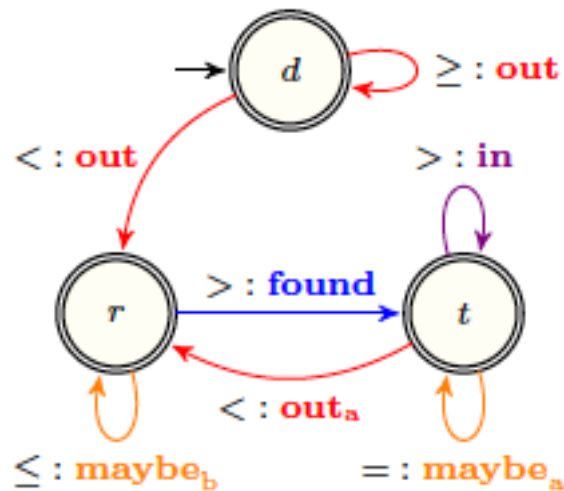
initialization	$C \leftarrow \text{default}_{af}$		$D \leftarrow \text{neutral}_f$	$R \leftarrow \text{default}_{af}$
return	$a(R, C)$			
Semantic Letter	Update of $C$		Update of $D$	Update of $R$
	After			
out				
out <sub>r</sub>			$D \leftarrow \text{neutral}_f$	
out <sub>a</sub>		$C \leftarrow \text{default}_{af}$	$D \leftarrow \text{neutral}_f$	$R \leftarrow a(R, C)$
maybe <sub>b</sub>			$D \leftarrow \phi_f(D, \delta_f)$	
maybe <sub>a</sub>	0		$D \leftarrow \phi_f(D, \delta'_f)$	
maybe <sub>a</sub>	1		$D \leftarrow \phi_f(D, \delta_f)$	
found	0	$C \leftarrow \phi_f(\phi_f(D, \delta_f), \delta'_f)$	$D \leftarrow \text{neutral}_f$	
found	1	$C \leftarrow \phi_f(D, \delta_f)$	$D \leftarrow \text{neutral}_f$	
in	0	$C \leftarrow \phi_f(C, \phi_f(D, \delta'_f))$	$D \leftarrow \text{neutral}_f$	
in	1	$C \leftarrow \phi_f(C, \phi_f(D, \delta_f))$	$D \leftarrow \text{neutral}_f$	
found <sub>e</sub>	0		$D \leftarrow \text{neutral}_f$	$R \leftarrow a(R, \phi_f(\phi_f(D, \delta_f), \delta'_f))$
found <sub>e</sub>	1		$D \leftarrow \text{neutral}_f$	$R \leftarrow a(R, \phi_f(D, \delta_f))$

# Example: synthesizing the automaton for min\_width\_peak

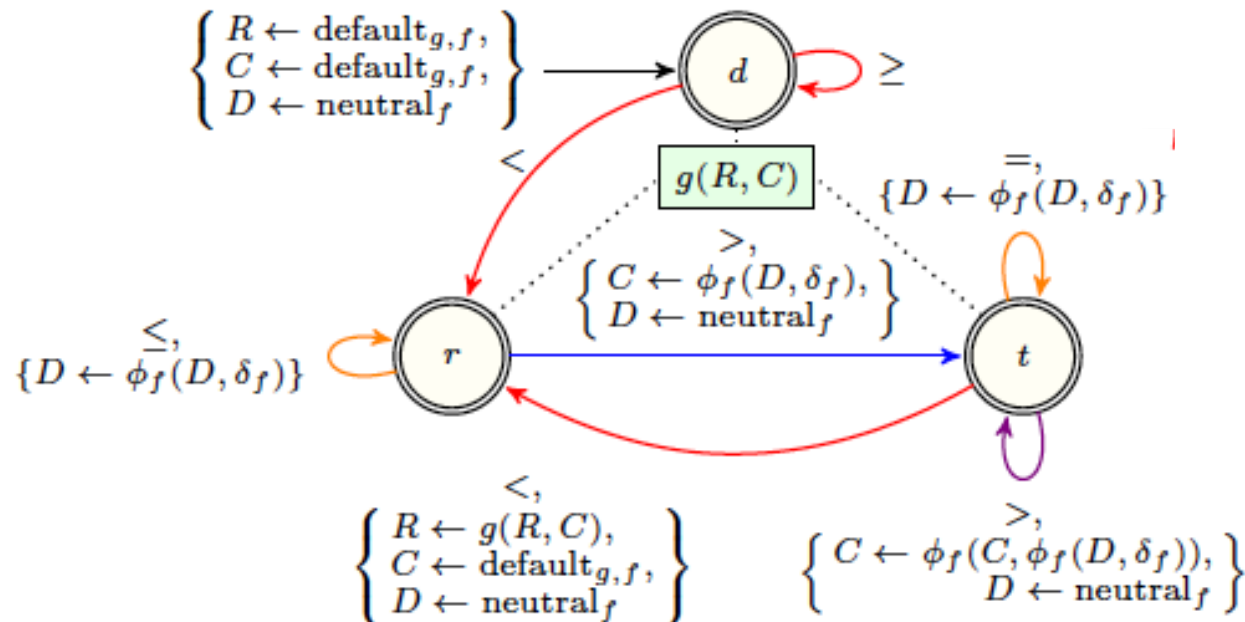


seed transducer

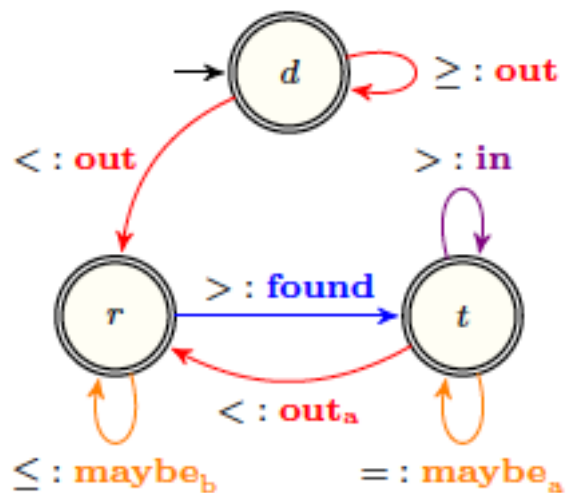
# Example: synthesizing the automaton for min\_width\_peak



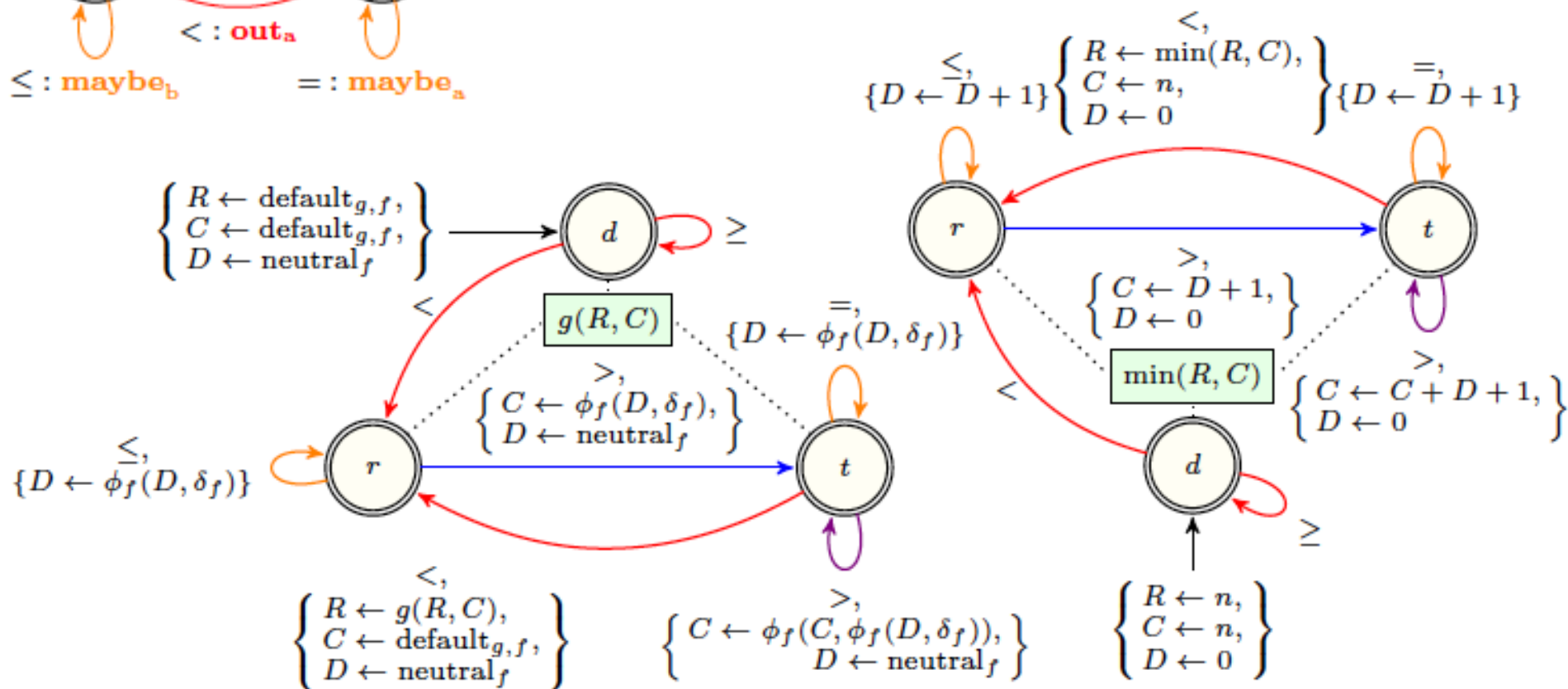
parametrized constraint  
(feature f, aggregator g)



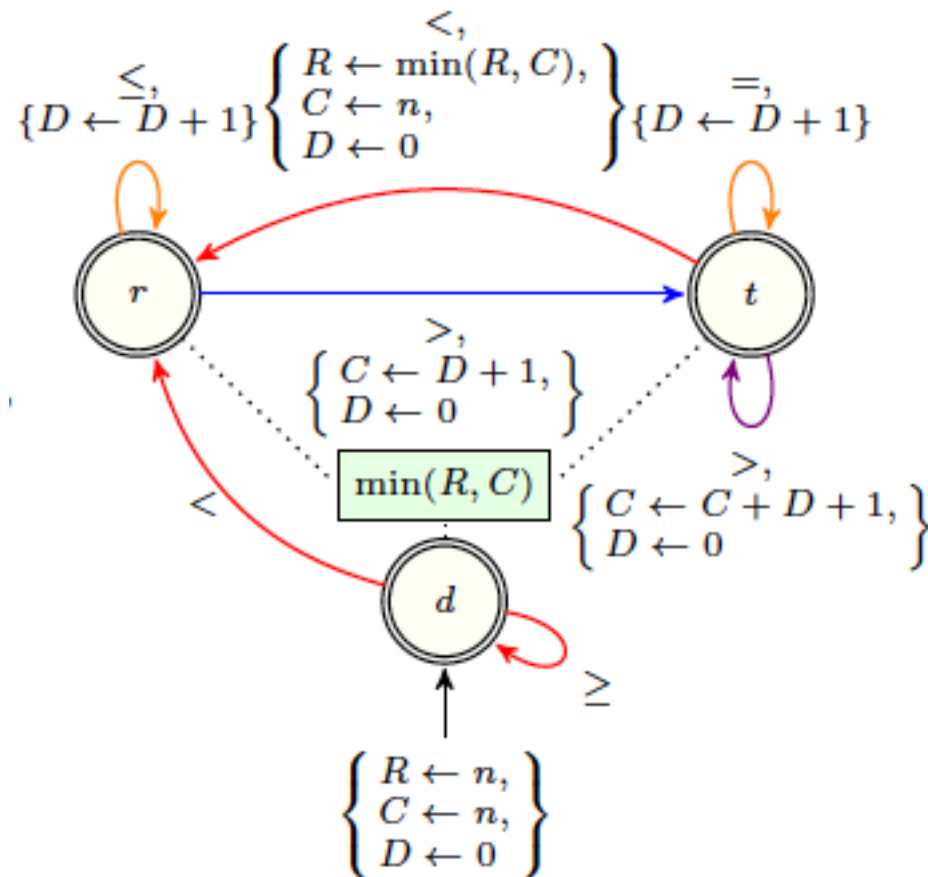
# Example: synthesizing the automaton for min\_width\_peak



specific constraint  
(min\_width\_peak)



# Running min\_width\_peak

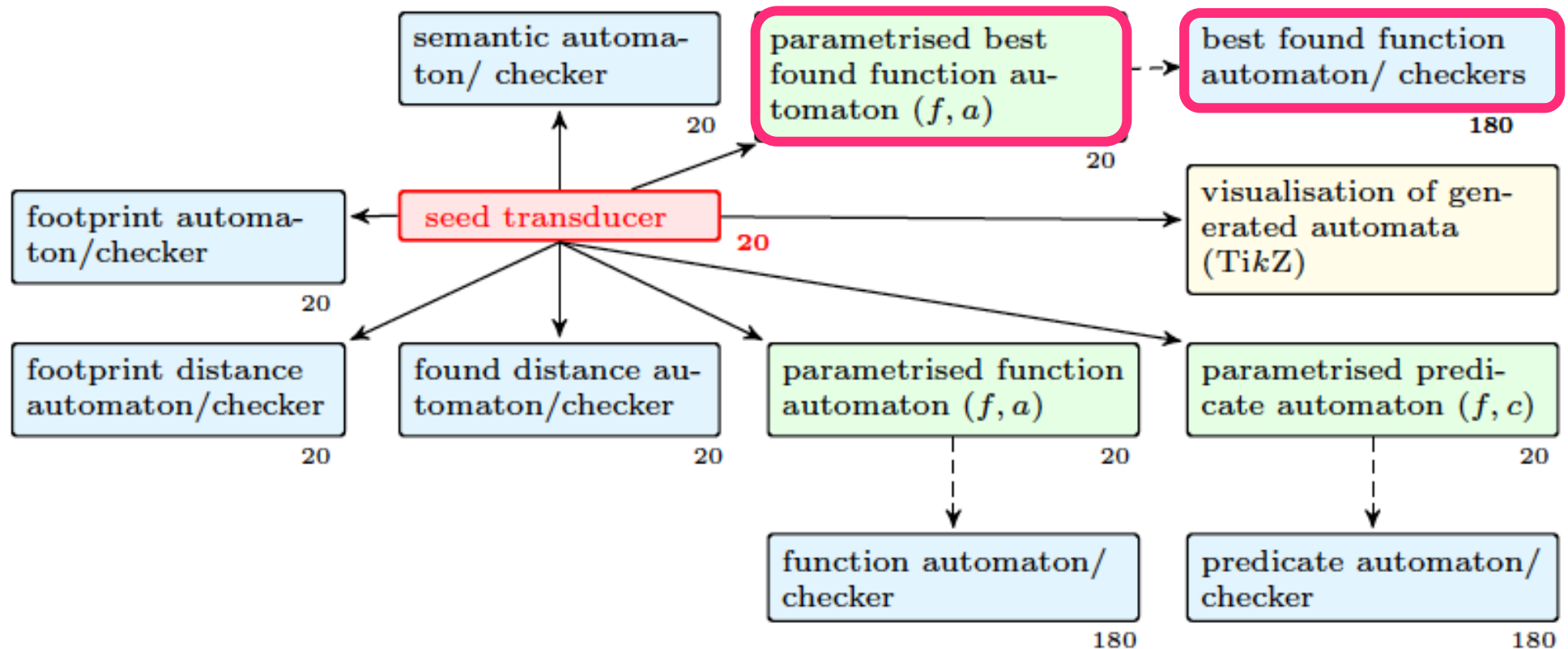


aggregate $R$	18	18	18	18	18	18	18	18	18	5	5	5	5	5	5	5	5	$\min(5, 6)$
current $C$	18	18	18	18	18	18	18	4	5	5	18	18	18	18	18	18	6	
potential $D$	0	0	0	0	0	1	2	3	0	0	1	0	1	2	3	4	5	0
semantic action	$\circ$	$\circ$	$\circ$	$\circ$	$m_b$	$m_b$	$m_b$	$f$	$in$	$m_a$	$o_a$	$m_b$	$m_b$	$m_b$	$m_b$	$m_b$	$f$	
signatures	$=$	$>$	$=$	$<$	$<$	$=$	$<$	$>$	$>$	$=$	$<$	$=$	$=$	$=$	$=$	$=$	$>$	
states	$s$	$s$	$s$	$s$	$r$	$r$	$r$	$r$	$t$	$t$	$t$	$r$	$r$	$r$	$r$	$r$	$r$	$t$
input	4	4	2	2	3	5	5	6	3	1	1	2	2	2	2	2	2	1

# Generated constraint for min\_width\_peak

```
min_width_peak_a(Value, VARIABLES):-
    collection(VARIABLES, [dvar]),
    get_attr1(VARIABLES, Xs), length(Xs, N),
    signature(Xs, Signature), gen_pairs(Xs, XPairs),
    Xs = [First|_], LT = 0, EQ = 1, GT = 2,
    automaton(XPairs, Xi-Xj, Signature,
    [source(s), sink(s), sink(r), sink(t)],
    [arc(s, GT, s, ([C, D, R])),
    arc(s, EQ, s, ([C, D, R])),
    arc(s, LT, r, ([C, D, R])),
    arc(r, GT, t, ([D + 1, 0, R])),
    arc(r, LT, r, ([C, D + 1, R])),
    arc(r, EQ, r, ([C, D + 1, R])),
    arc(t, GT, t, ([C + D + 1, 0, R])),
    arc(t, EQ, t, ([C, D + 1, R])),
    arc(t, LT, r, ([N, 0, min(R, C)])]),
    [C, D, R], [N, 0, N], [CLast, DLast, RLast]),
    Value #= min(RLast, CLast).
```

# Identifying occurrence of pattern with extreme value

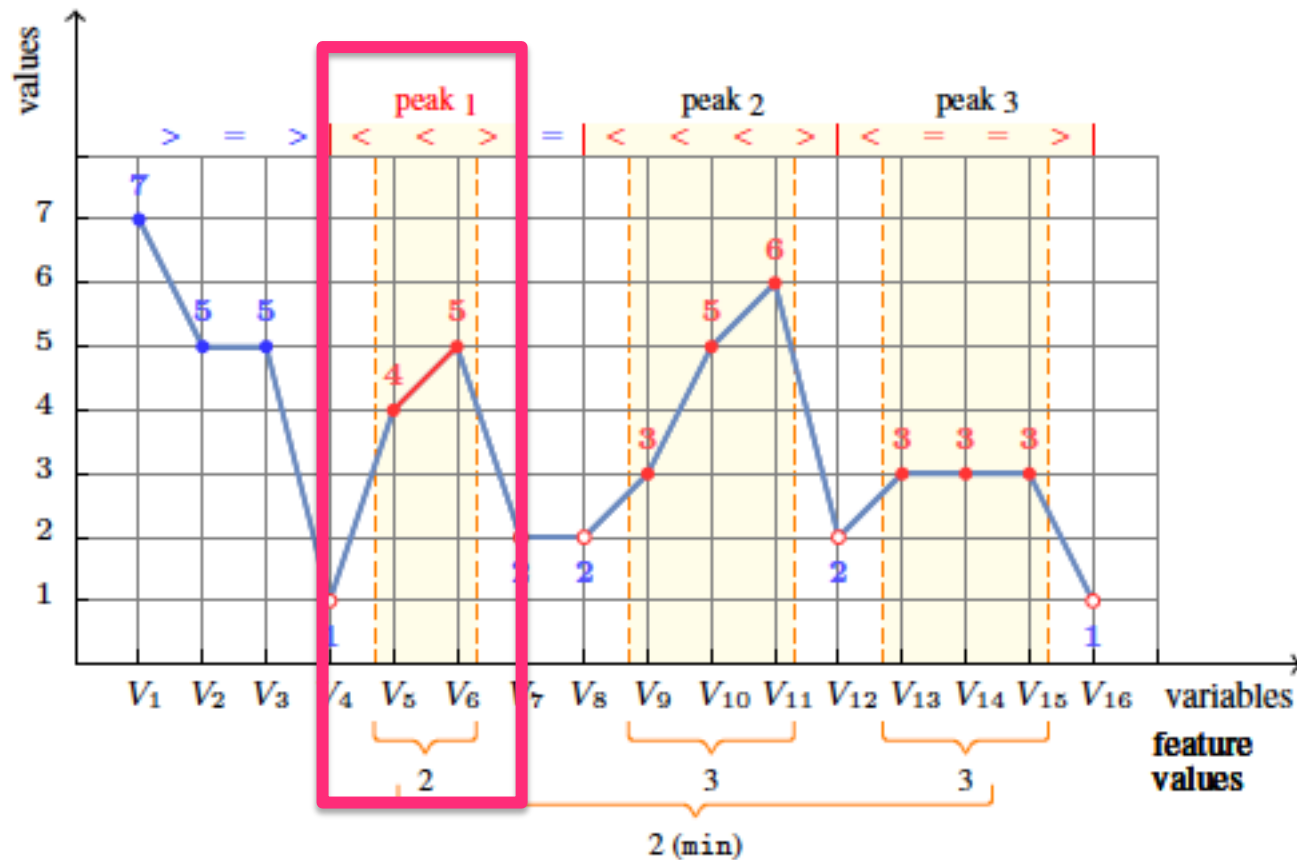




# Best found feature constraints

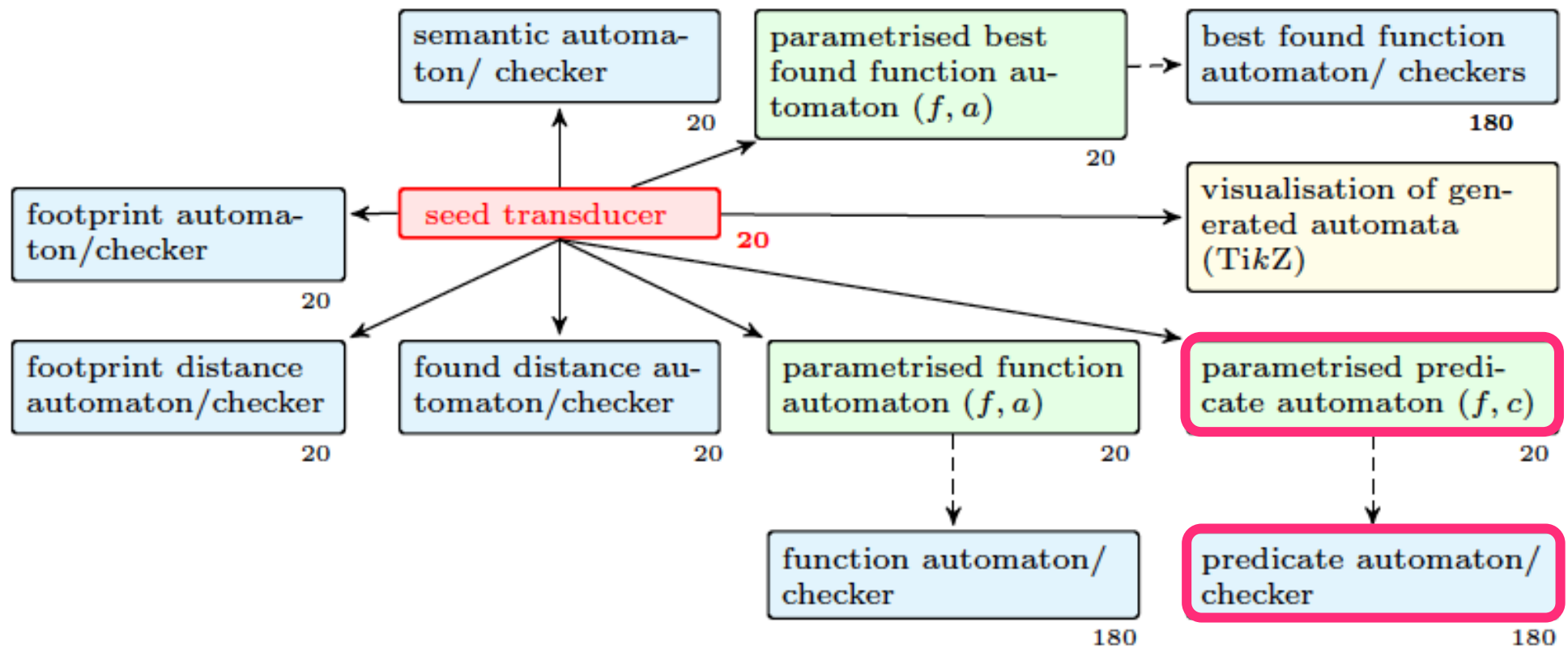
- Used only with min/max aggregators
- Smallest provide footprint of those i-occurrences for which the feature value corresponds to the minimum/maximum value
- Possible use: find most problematic patterns occurrences (e.g., longest zigzag)

# Best found feature constraints (example)

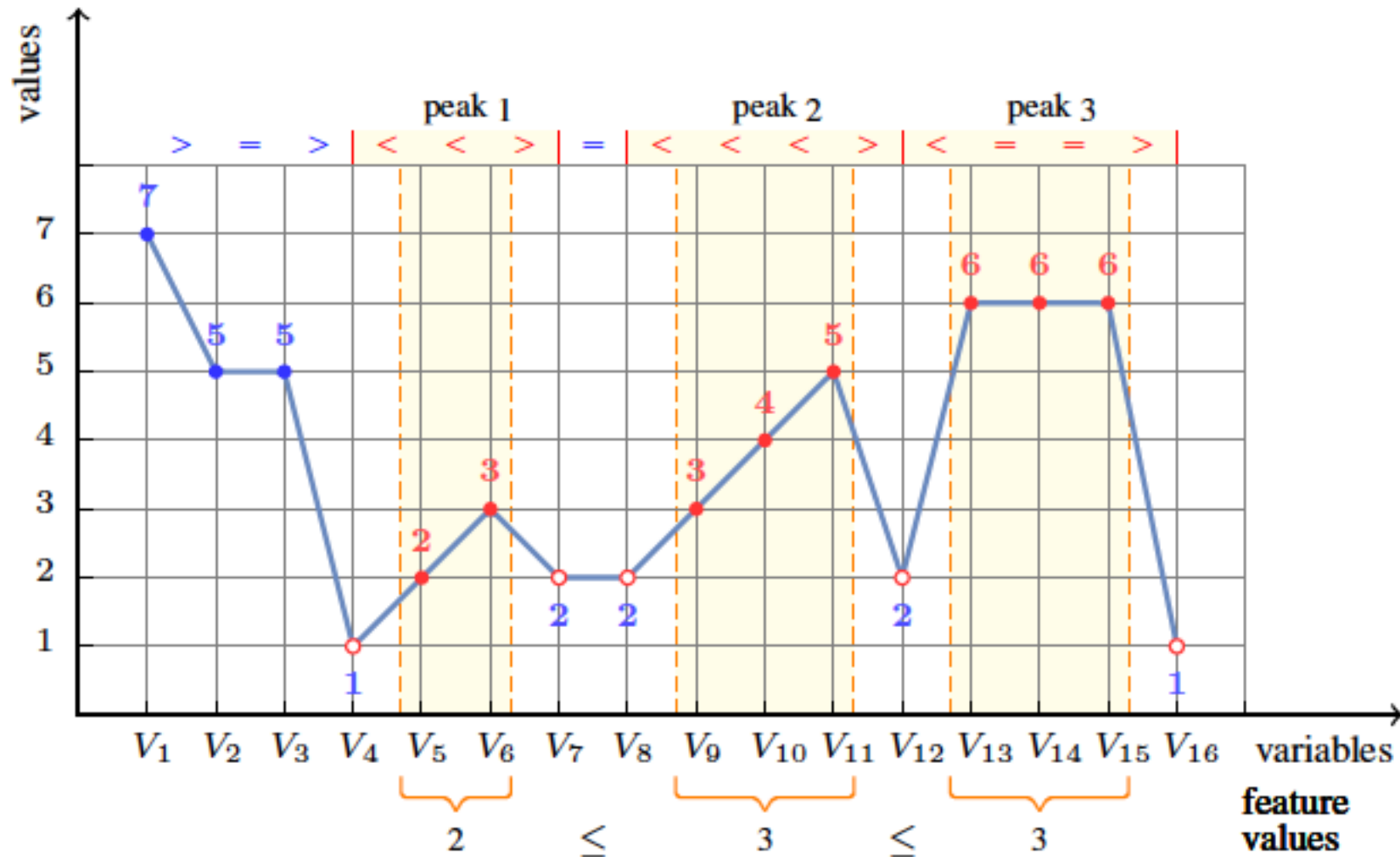


```
best_found( min_width_peak, 2,
            [7,5,5,1,4,5,2,2,3,5,6,2,3,3,3,1],
            [0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0] )
```

# Predicate constraints

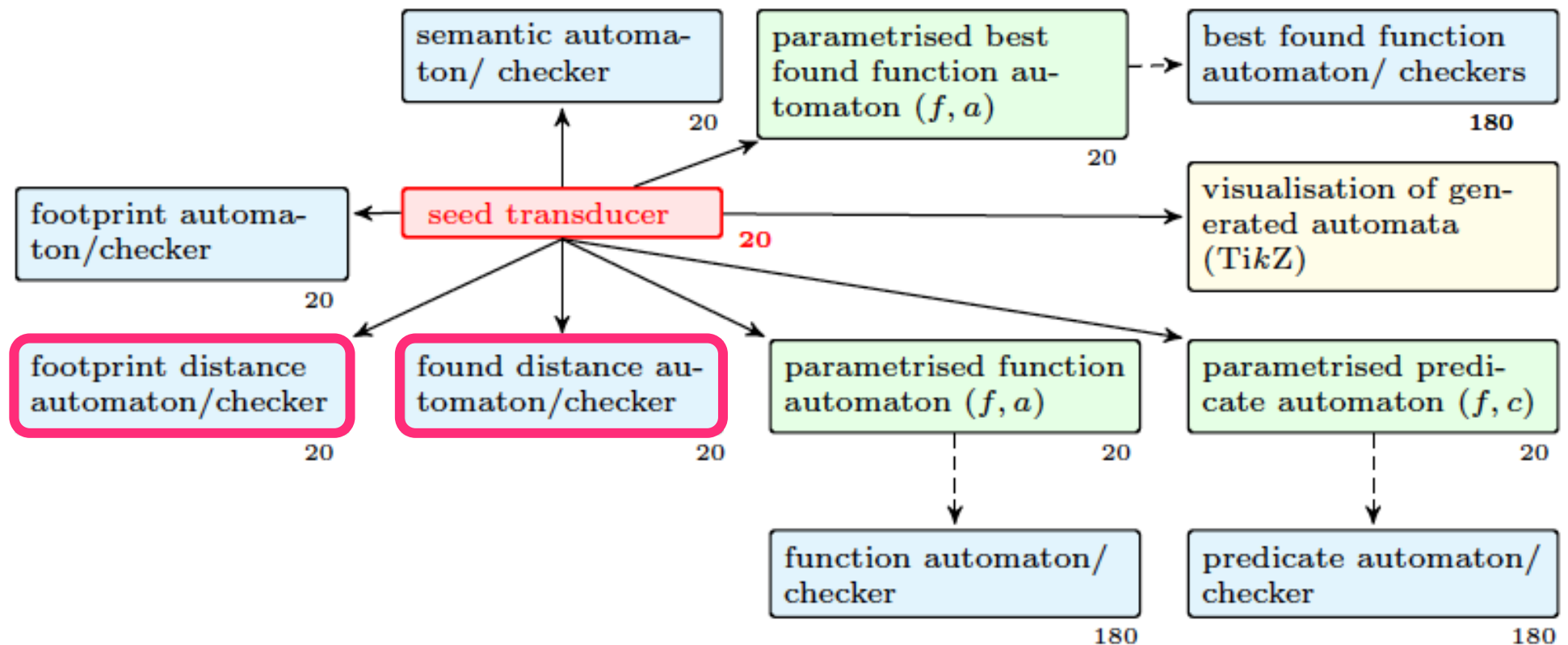


# Predicate constraints (example)



INCREASING\_WIDTH\_PEAK ([7, 5, 5, 1, 2, 3, 2, 2, 3, 4, 5, 2, 6, 6, 6, 1])

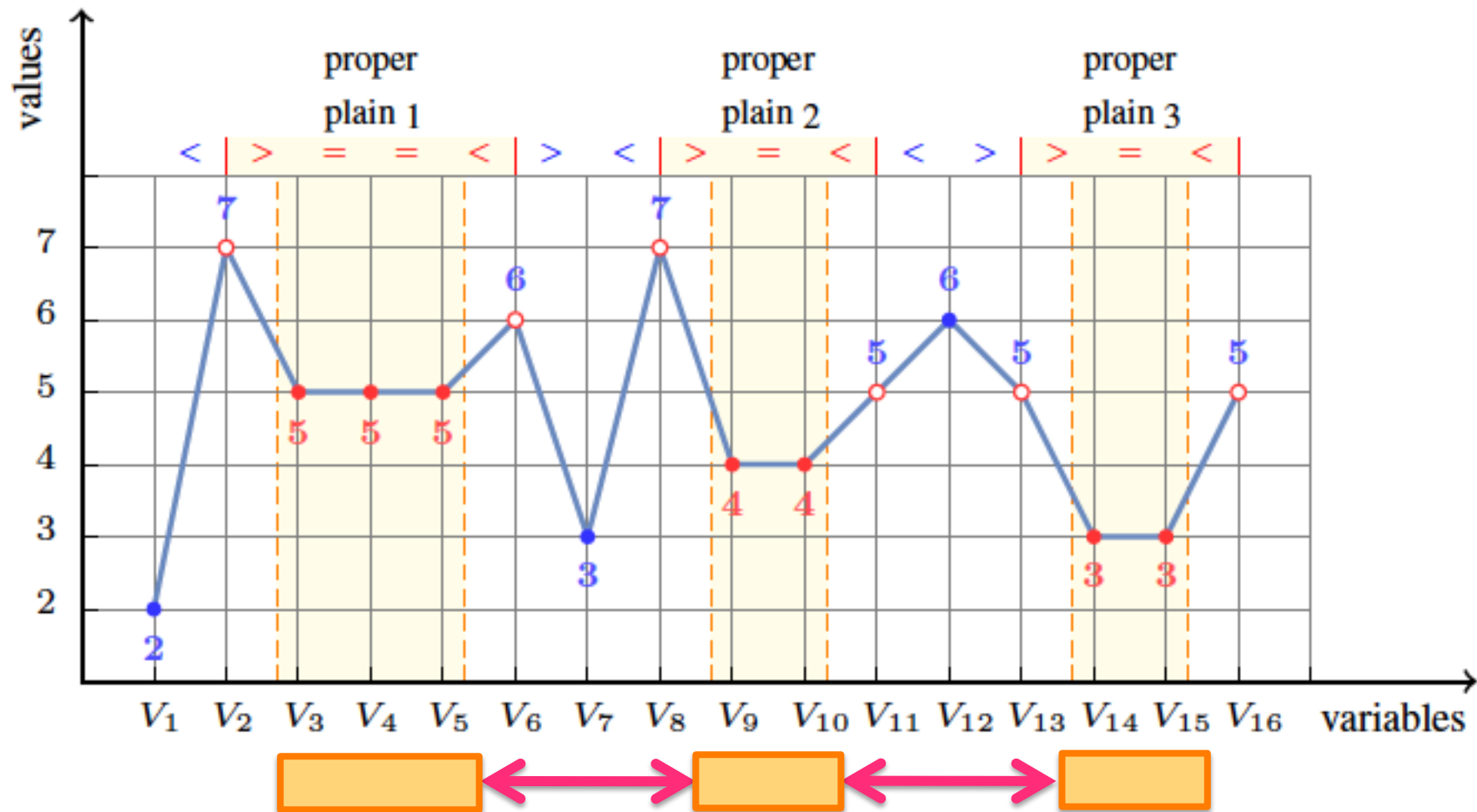
# Min/max distance between pattern occurrences



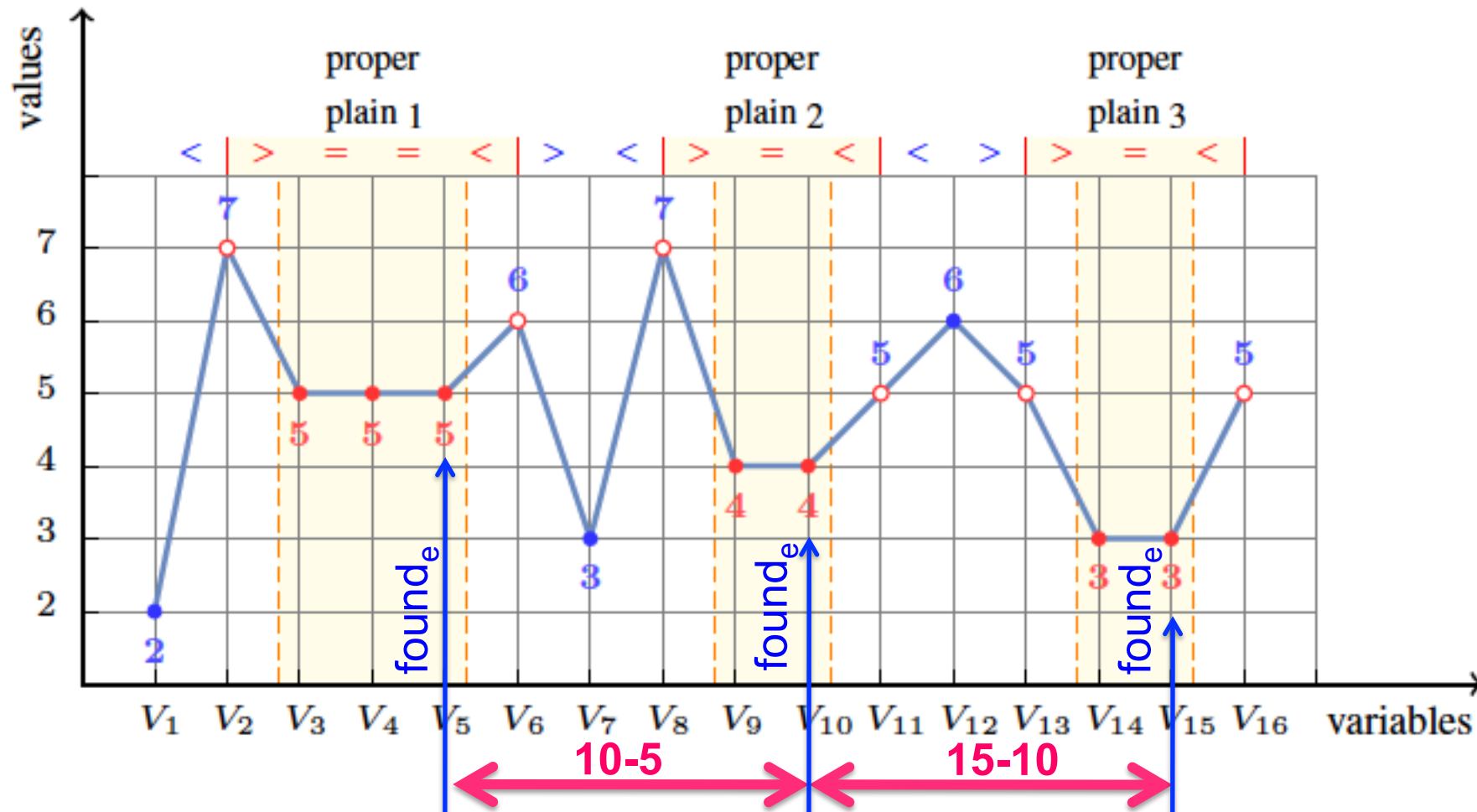
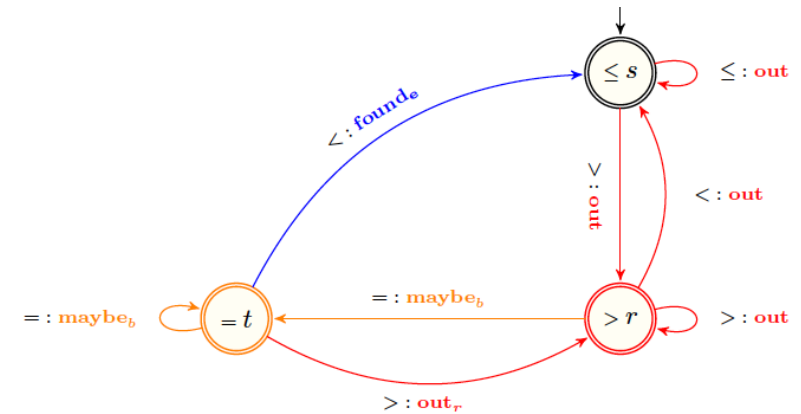
# Min/max distance between occurrences of patterns

- **Found** distance:  
distance between two consecutive found events
- **Footprint** distance:  
distance between two consecutive *i*-occurrences

# Footprint distance example (proper\_plain)



# Found distance example (proper\_plain)





# Existing constraints of the catalog

---

**constraint**

---

all\_equal\_peak  
all\_equal\_valley  
decreasing\_peak  
decreasing\_valley  
deepest\_valley  
highest\_peak  
increasing\_peak  
increasing\_valley  
inflexion  
longest\_decreasing\_sequence  
longest\_increasing\_sequence  
max\_decreasing\_slope  
max\_increasing\_slope  
min\_decreasing\_slope  
min\_dist\_between\_inflexion  
min\_increasing\_slope  
min\_surf\_peak  
min\_width\_peak  
min\_width\_plateau  
min\_width\_valley  
peak  
valley

---

# Reformulation of automata constraints in LP

- Two initial papers on automata constraints

G. Pesant (without counters) CP 2004

N. Beldiceanu *et al.* (with counters) CP 2004

- Reformulation of regular in LP

L.M. Rousseau *et al.* (without counters) CPAIOR 2007

# Reformulation of times-series constraints (function) in LP

## Main result

- Reformulation as LP of all time-series constraints
- Reformulation size (number of variables/number of linear constraints) is in  $O(n)$  where  $n$  is the sequence size

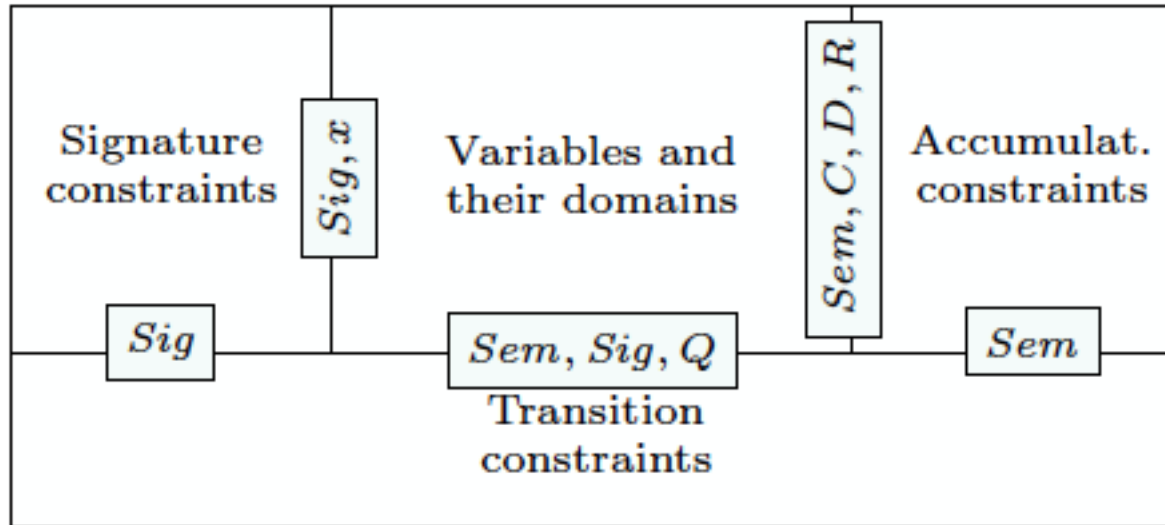
# Reformulation of times-series constraints in LP

- Two step process
  - First have a logical model encoding the behaviour of the transducer and of the counter updates
  - Second linearize the logical model using standard modelling techniques

# Variables of the logical model

- States variables  $Q_i$  (i in  $[0, n-1]$ )
- Signature variables  $Sig_i$  (i in  $[0, n-2]$ )
- Semantic variables  $Sem_i$  (i in  $[0, n-2]$ )
- Accumulator variables  $C_i, D_i, R_i$  (i in  $[0, n-2]$ )

# Constraints of the logical model



example

---

## Accumulator constraints

$$\begin{aligned}
 Sem_i = \text{found} &\Rightarrow R_{i+1} = R_i + 1, \\
 Sem_i = \sigma' &\Rightarrow R_{i+1} = R_i, \\
 &\forall \sigma' \in \{\text{out}_a, \text{out}, \text{in}\}, \forall i \in [0, n - 2].
 \end{aligned}$$


---

## Transition constraints

$$\begin{aligned}
 Q_i = s \wedge Sig_i = '>' &\Rightarrow Q_{i+1} = s \wedge Sem_i = \text{out}, \\
 Q_i = s \wedge Sig_i = '=' &\Rightarrow Q_{i+1} = s \wedge Sem_i = \text{out}, \\
 Q_i = s \wedge Sig_i = '<' &\Rightarrow Q_{i+1} = t \wedge Sem_i = \text{found}, \\
 Q_i = t \wedge Sig_i = '<' &\Rightarrow Q_{i+1} = t \wedge Sem_i = \text{in}, \\
 Q_i = t \wedge Sig_i = '>' &\Rightarrow Q_{i+1} = s \wedge Sem_i = \text{out}_a, \\
 Q_i = t \wedge Sig_i = '=' &\Rightarrow Q_{i+1} = s \wedge Sem_i = \text{out}_a, \\
 &\forall i \in [0, n - 2].
 \end{aligned}$$


---

# Use

## Context of use

time-series that **partly depends on some underlying structural constraints** that are linked to some infrastructure  
(*e.g. temperature measurement in a building,  
energy production by some unit*)

# Building models

**Learning models** from structured time series data,  
where model is a conjunction of time-series constraints.

➔ *Compute features and select the most relevant ones*  
not just classifying things,  
can be used as a constraint model to generate new solutions.



# Identifying automatically problems in data

- **Locate extreme feature values** for some pattern  
(e.g. *zizag, bump\_on\_decreasing\_sequence, dip\_on\_increasing\_sequence*)
- **Repair identified problems**  
(*by solving a small constraint problem*)

# Conclusion

1. Everything synthesised from the seed transducer  
*(decoration tables independent from the transducer)*
2. Some new work *(oriented toward practice)* regarding automata constraints *(not just picking the next item in the language hierarchy)*
3. On going work to extend, reinforce things along different lines
4. Automata for which the generator is very compact/versus automata for which you need to write ad-hoc code to generate them

# Conclusion

- Second volume of the global constraint catalog devoted to time-series constraints
  - use exactly the **same format** as the current catalog,
  - with the difference that **everything is synthesized** (*text, figures, code*).

## Global Constraint Catalog Volume II Time-Series Constraints

Nicolas Beldiceanu<sup>1</sup>

TASC (CNRS/INRIA) Mines Nantes, FR-44307 Nantes, France

Mats Carlsson

SICS, Box 1263, SE-16 429 Kista, Sweden

Helmut Simonis

Insight Centre for Data Analytics, University College Cork, Ireland

**Abstract:** This report first presents a restricted set of finite transducers used to synthesise structural time-series constraints described by means of a multi-layered functions composition scheme. Second it provides the corresponding synthesised catalogue of structural time-series constraints where each constraint is explicitly described in terms of automata with accumulators.