

Computing by Only Observing

M. Cavaliere

Microsoft Research - University of Trento, Italy

in collaboration with:

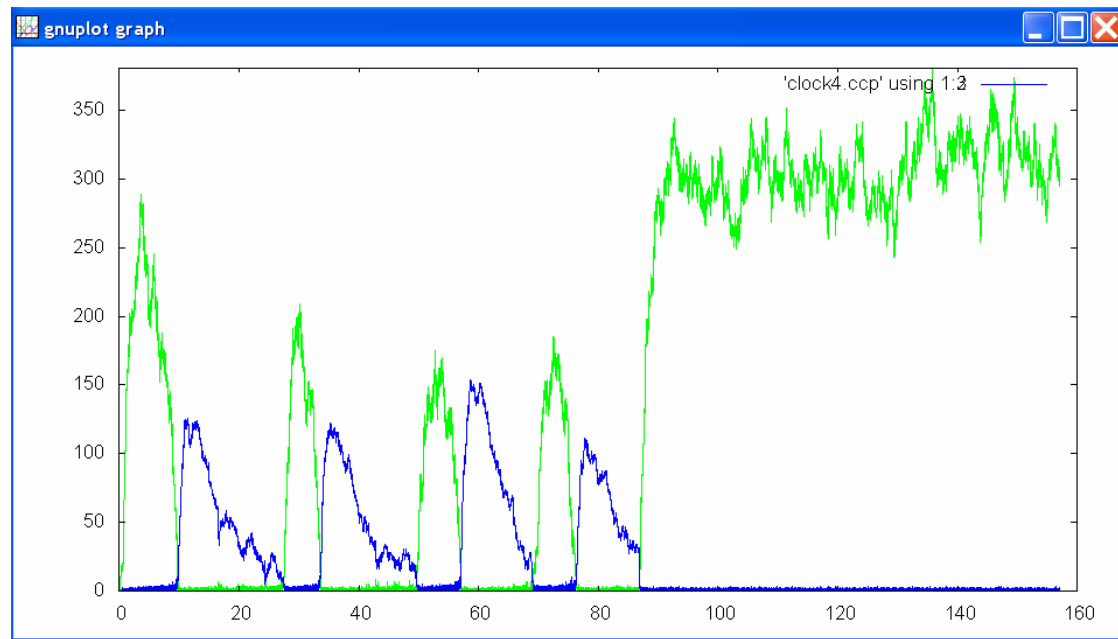
P. Frisco *Heriot Watt University, Edinburgh, U.K.*

H.J. Hoogeboom *Leiden University, The Netherlands*

From Biology...

In biology:

(often) the “result” of an experiment
is the observation (interpretation) of a
process



...to Computer Science

In computer science:

the “result” of a computation

is the observation (interpretation) of a
process

...to Computer Science

In computer science:

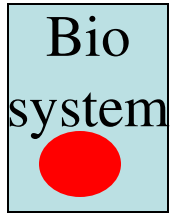
the “result” of a computation

is the observation (interpretation) of a
process

Introduced in membrane systems (Cavaliere, Leupold, 2003).

Extended to natural computing and formal languages.

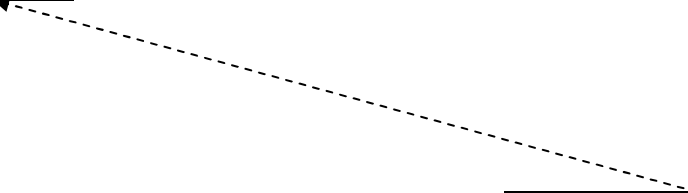
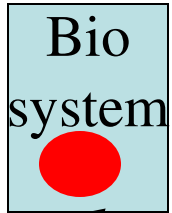
Observing the evolution of the bio-system



Observer

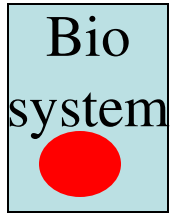
Observing the evolution of the bio-system

Step 1



Observing the evolution of the bio-system

Step 1

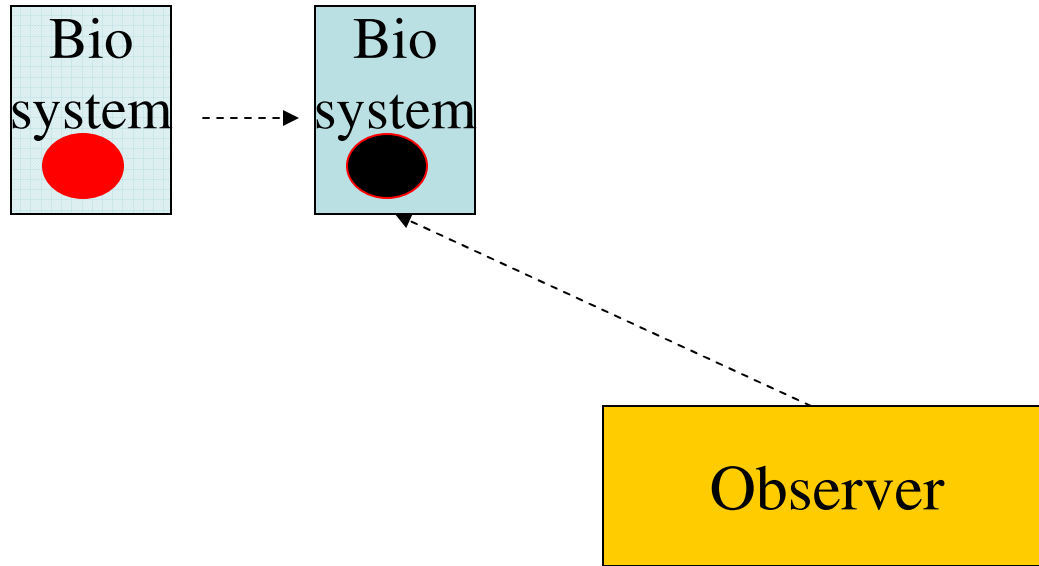


R



Observing the evolution of the bio-system

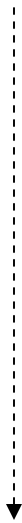
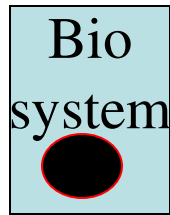
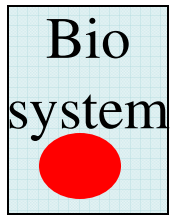
Step 2



R

Observing the evolution of the bio-system

Step 2

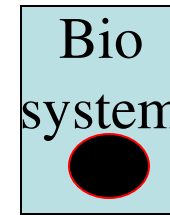
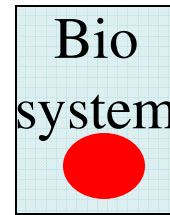
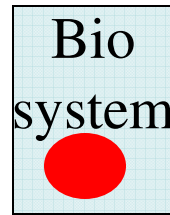
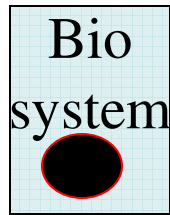
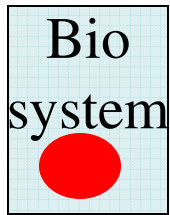


R

B

Observing the evolution of the bio-system

Step 5



R

B

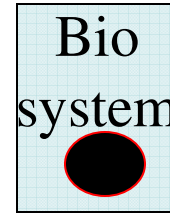
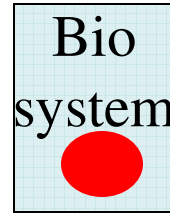
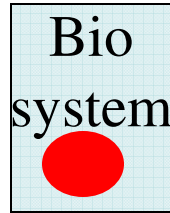
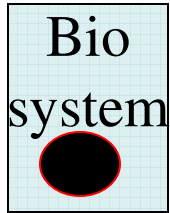
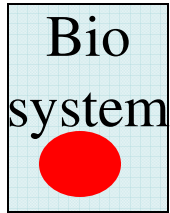
R

R

B



Result = Observed Evolution



R

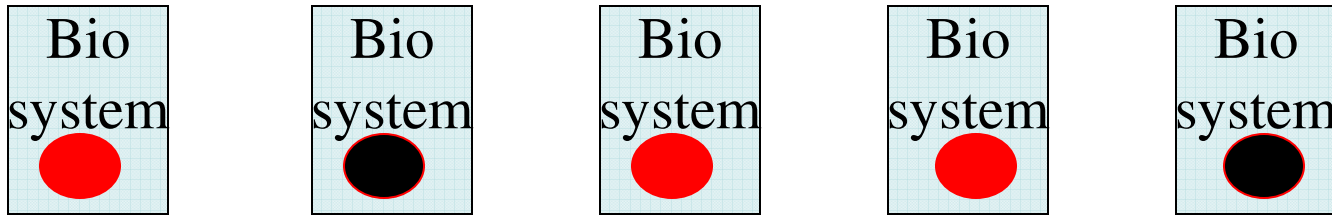
B

R

R

B

Result = Observed Evolution



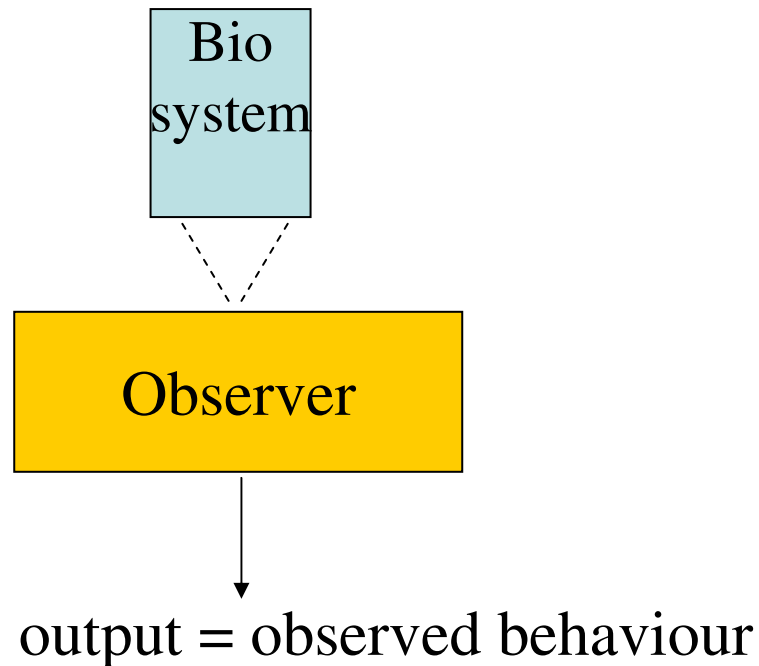
Result of the computation



Bio-system + Observer = Generative Computing Device.

We have two components:

- a **bio-system** - evolves.
- an **observer** - extracts a formal behaviour.



Constructing Generative Devices

It has been proved that

“simple” bio-systems **observed** by “simple” observers are computationally universal.

Ex:

sticker systems + regular observers are universal (2004)

finite splicing systems + regular observers are universal (2005)

recognizing devices (2006)

Constructing Generative Devices

The “usual” procedure:

One designs a specific bio-system and a specific observer to produce a specific result.

There is a problem:

Many bio-systems cannot be (easily) programmed. They “just” exists.

However:

The evolution of a bio-system can be monitored by using different observers.

Then:

How much is it possible to compute by fixing a basic bio-system and only choosing different observers?

Constructing Generative Devices

The “usual” procedure:

One designs a specific bio-system and a specific observer to produce a specific result.

There is a problem:

Many bio-systems cannot be (easily) programmed. They “just” exist.

However:

The evolution of a bio-system can be monitored by using different observers.

Then:

How much is it possible to compute by fixing a basic bio-system and only choosing different observers?

Constructing Generative Devices

The “usual” procedure:

One designs a specific bio-system and a specific observer to produce a specific result.

There is a problem:

Many bio-systems cannot be (easily) programmed. They “just” exist.

However:

The evolution of a bio-system can be monitored by using different observers.

Then:

How much is it possible to compute by fixing a basic bio-system and only choosing different observers?

Constructing Generative Devices

The “usual” procedure:

One designs a specific bio-system and a specific observer to produce a specific result.

There is a problem:

Many bio-systems cannot be (easily) programmed. They “just” exist.

However:

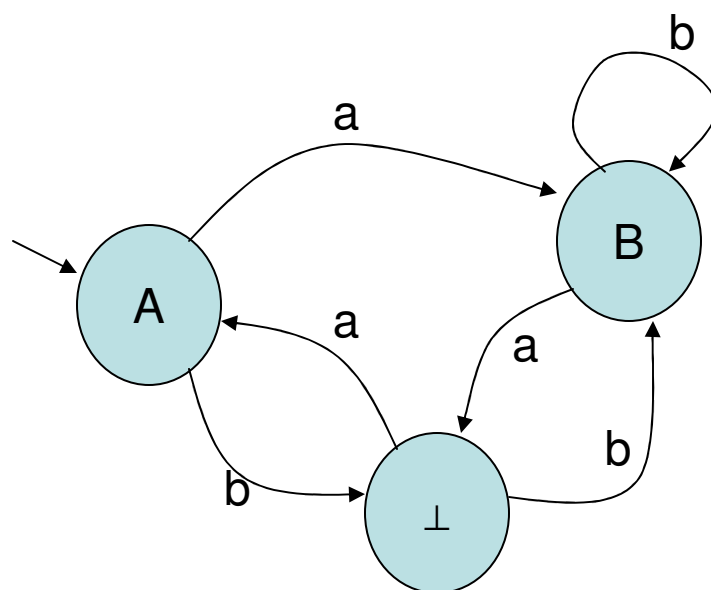
The evolution of a bio-system can be monitored by using different observers.

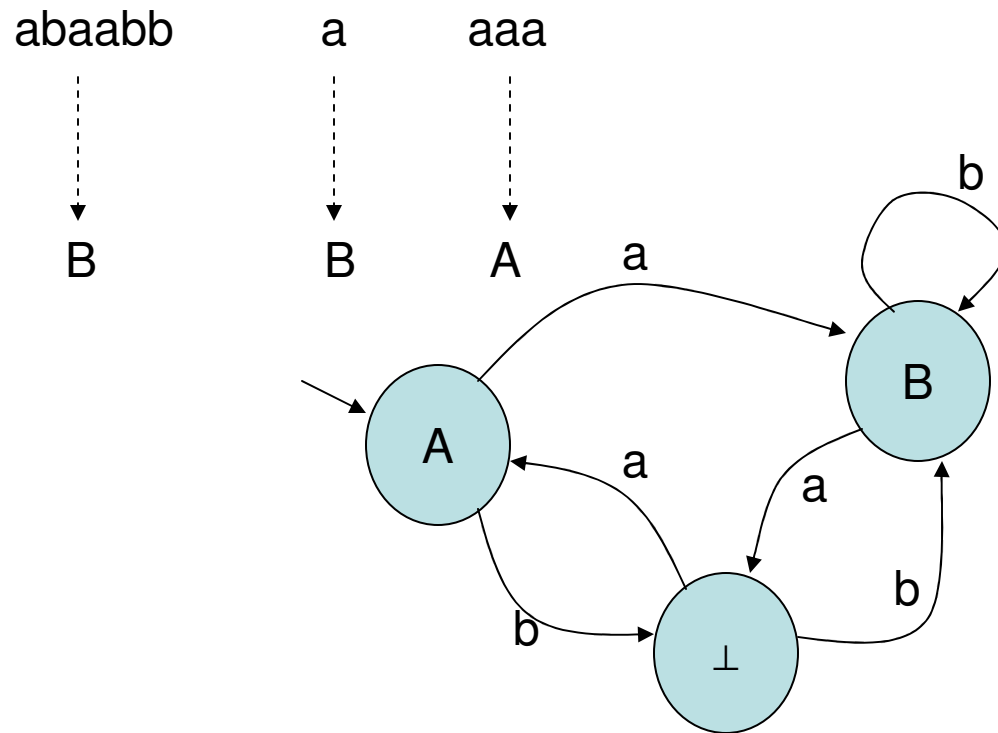
Then:

How much is it possible to compute by fixing a basic bio-system and only choosing different observers?

Composing Grammars and Observers

- Observer is a finite state automaton whose set of states is labeled with the symbols of an output alphabet.
- $O = (Z, V, \Sigma, z_0, \delta, \sigma)$
- Z states
- V input alphabet
- Σ output alphabet
- z_0 initial state
- $\delta: Z \times V \rightarrow Z$ transition function
- $\sigma: Z \rightarrow \Sigma \cup \{\lambda, \perp\}$





It associates to the strings a label (“meaning”)

G/O Systems

A G/O system is a pair

$$\Omega = (G, O)$$

$$G = (N, T, S, P)$$

$$O = (Z, V, \Sigma, z_0, \delta, \sigma)$$

$O(w)$ output of O over w

$$w_0, w_1, w_2, \dots, w_n$$

$$O(w_0) \cdot O(w_1) \cdot O(w_2) \cdots O(w_n) = O(w_0, w_1, \dots, w_n)$$

G/O Systems

A G/O system is a pair

$$\Omega = (G, O)$$

$$G = (N, T, S, P)$$

$$O = (Z, V, \Sigma, z_0, \delta, \sigma)$$

$O(w)$ output of O over w

$$w_0, w_1, w_2, \dots, w_n \qquad O(w_0) \cdot O(w_1) \cdot O(w_2) \cdots O(w_n) = O(w_0, w_1, \dots, w_n)$$

$$L(\Omega) = \{ O(w_0, w_1, \dots, w_n) \in \Sigma^* \mid S = w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n, \quad w_n \in T^* \}$$

G/O Systems: an Example

$$G = (N, T, S, P)$$

$$N = \{S, A, B, C\} \quad T = \{t, p\}$$

$$P = \{ S \rightarrow pS, S \rightarrow p, S \rightarrow A, \\ A \rightarrow AB, A \rightarrow C, B \rightarrow C, C \rightarrow t \}$$

$$O(w) = a \quad \text{if } w \in \{S, A, B, C, t, p\}^+$$

G/O Systems: an Example

$$G = (N, T, S, P)$$

$$N = \{S, A, B, C\} \quad T = \{t, p\}$$

$$P = \{ S \rightarrow pS, S \rightarrow p, S \rightarrow A, \\ A \rightarrow AB, A \rightarrow C, B \rightarrow C, C \rightarrow t \}$$

$$O(w) = a \quad \text{if } w \in \{S, A, B, C, t, p\}^+$$

$$\begin{array}{ccccccc} S & \Rightarrow & pS & \Rightarrow^{n-2} & p^{n-1}S & \Rightarrow & p^{n-1} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a & & a & \cdots & a & & a \end{array}$$

G/O Systems: an Example

$$G = (N, T, S, P)$$

$$N = \{S, A, B, C\} \quad T = \{t, p\}$$

$$P = \{ S \rightarrow pS, S \rightarrow p, S \rightarrow A, \\ A \rightarrow AB, A \rightarrow C, B \rightarrow C, C \rightarrow t \}$$

$$O(w) = a \quad \text{if } w \in \{S, A, B, C, t, p\}^+$$

$$\begin{array}{ccccccc} S & \Rightarrow & pS & \Rightarrow^{n-2} & p^{n-1}S & \Rightarrow & p^{n-1} \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \\ a & & a & \cdots & a & & a \end{array}$$

$$L(G, O) = \{a^i \mid i \geq 2\}$$

G/O Systems: an Example

$$G = (N, T, S, P)$$

$$N = \{S, A, B, C\} \quad T = \{t, p\}$$

$$P = \{ S \rightarrow pS, S \rightarrow p, S \rightarrow A, \\ A \rightarrow AB, A \rightarrow C, B \rightarrow C, C \rightarrow t \}$$

$$O'(w) = \begin{cases} \lambda & \text{if } w = S \\ a & \text{if } w \in AB^* \\ b & \text{if } w \in C^+B^* \\ c & \text{if } w \in t^+B^* \\ \perp & \text{else} \end{cases}$$

Same grammar – different observer

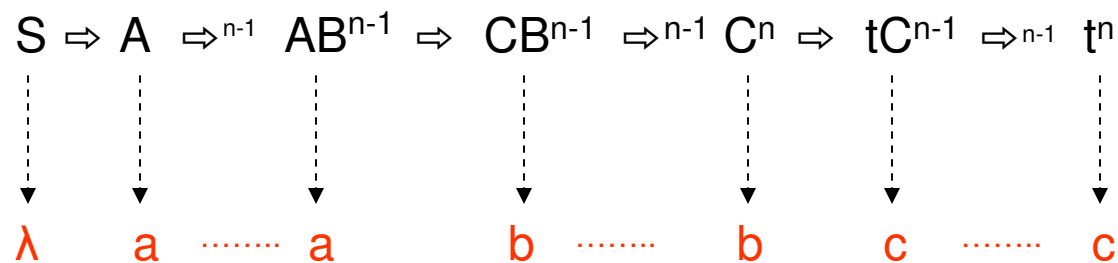
G/O Systems: an Example

$$G = (N, T, S, P)$$

$$N = \{S, A, B, C\} \quad T = \{t, p\}$$

$$P = \{ S \rightarrow pS, S \rightarrow p, S \rightarrow A, \\ A \rightarrow AB, A \rightarrow C, B \rightarrow C, C \rightarrow t \}$$

$$O'(w) = \begin{cases} \lambda & \text{if } w = S \\ a & \text{if } w \in AB^* \\ b & \text{if } w \in C^+B^* \\ c & \text{if } w \in t^+B^* \\ \perp & \text{else} \end{cases}$$



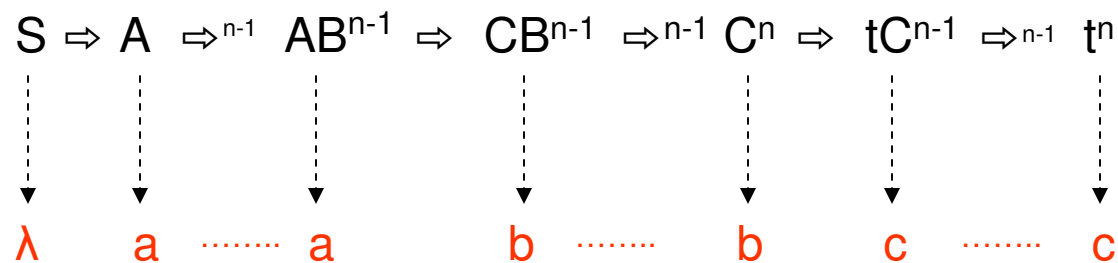
G/O Systems: an Example

$$G = (N, T, S, P)$$

$$N = \{S, A, B, C\} \quad T = \{t, p\}$$

$$P = \{ S \rightarrow pS, S \rightarrow p, S \rightarrow A, \\ A \rightarrow AB, A \rightarrow C, B \rightarrow C, C \rightarrow t \}$$

$$O'(w) = \begin{cases} \lambda & \text{if } w = S \\ a & \text{if } w \in AB^* \\ b & \text{if } w \in C^+B^* \\ c & \text{if } w \in t^+B^* \\ \perp & \text{else} \end{cases}$$



$$L(G, O') = \{a^n b^n c^n \mid n > 0\}$$

How much can we compute by only changing the observer and not the (observed) grammar?

How much can we compute by only changing the observer and not the (observed) grammar?

$\mathcal{L}^G(\text{FA})$

*families of languages defined by G/O systems using **the grammar G** and an observer that is a finite state automaton.*

How much can we compute by only changing the observer and not the (observed) grammar?

$\mathcal{L}^G(\text{FA})$

*families of languages defined by G/O systems using **the grammar G** and an observer that is a finite state automaton.*

A grammar G is universal modulo observation for a family of languages L if $L = \mathcal{L}^G(\text{FA})$.

How much can we compute by only changing the observer and not the (observed) grammar?

$\mathcal{L}^G(\text{FA})$

*families of languages defined by G/O systems using **the grammar G** and an observer that is a finite state automaton.*

A grammar G is universal modulo observation for a family of languages L if $L = \mathcal{L}^G(\text{FA})$.

G can generate every language in L when G's evolutions (derivations) are observed in an appropriate way.

How much can we compute by only changing the observer and not the (observed) grammar?

$\mathcal{L}^G(\text{FA})$

*families of languages defined by G/O systems using **the grammar G** and an observer that is a finite state automaton.*

A grammar G is universal modulo observation for a family of languages L if $L = \mathcal{L}^G(\text{FA})$.

G can generate every language in L when G's evolutions (derivations) are observed in an appropriate way.

Theorem

There exists a context-free grammar that is universal m.o. for RE

CF grammar + FA observer can “simulate” type-0 productions.

Type-0 grammar G_1 in Kuroda n.f.

$A \rightarrow BC, A \rightarrow a, A \rightarrow \lambda, \mathbf{AB \rightarrow CD}$

CF grammar G_2 with

$A \rightarrow BC, A \rightarrow a, A \rightarrow \lambda$

and

$A \rightarrow Ar, Ar \rightarrow Cr, Cr \rightarrow C,$
 $B \rightarrow Br, Br \rightarrow Dr, Dr \rightarrow D \quad \text{for } r: AB \rightarrow CD$

A regular observer can check that the rewriting of nonterminals A, B is done in a synchronized way (use \perp to “trash” wrong computation).

A specific type-0 grammar can be “simulated” by a specific CF grammar + a specific FA observer

Starting from a **universal type-0 grammar** $G_u=(N_u, T_u, P_u)$

N_u, T_u, P_u

For any type-0 grammar $G=(N, T_u, S, P)$ there is a string $w(G)$ such that $L(G_u, w(G))=L(G)$.

We can construct a CF grammar G'_u
that can first generate a string “ w ”
then it simulates the productions of P_u over “ w ”.

For any RE language $L(G)$ one can construct an observer O_G that checks:

- the string “ w ” generated is exactly $w(G)$
- the productions of P_u are correctly simulated.
- emits \perp if the first two conditions are not respected

Starting from a **universal type-0 grammar** $G_u=(N_u, T_u, P_u)$

N_u, T_u, P_u

For any type-0 grammar $G=(N, T_u, S, P)$ there is a string $w(G)$ such that $L(G_u, w(G))=L(G)$.

We can construct a CF grammar G'_u
that can first generate a string “w”
then it simulates the productions of P_u over “w”.

For any RE language $L(G)$ one can construct an observer O_G that checks:

- the string “w” generated is exactly $w(G)$
- the productions of P_u are correctly simulated.
- emits \perp if the first two conditions are not respected

The CF grammar G'_u is fixed (it is the universal one)
 O_G depends on the specific RE language $L(G)$.

Starting from a **universal type-0 grammar** $G_u = (N_u, T_u, P_u)$

N_u, T_u, P_u

For any type-0 grammar $G = (N, T_u, S, P)$ there is a string $w(G)$ such that $L(G_u, w(G)) = L(G)$.

We can construct a CF grammar G'_u
that can first generate a string “w”
then it simulates the productions of P_u over “w”.

For any RE language $L(G)$ one can construct an observer O_G that checks:

- the string “w” generated is exactly $w(G)$
- the productions of P_u are correctly simulated.
- emits \perp if the first two conditions are not respected

The CF grammar G'_u is fixed (it is the universal one)

O_G depends on the specific RE language $L(G)$.

Not Using \perp

Corollary

There exists a context-free grammar such that $\mathcal{L}^G(\text{FA})$ contains non recursive languages.

Restrictions on CF grammars

A CF grammar is nonterminal bounded if there exists a constant k such that all sentential forms generated by the grammar have at most k nonterminals.

Restrictions on CF grammars

A CF grammar is nonterminal bounded if there exists a constant k such that all sentential forms generated by the grammar have at most k nonterminals.

Theorem

For every G/O system $\Omega = (G, O)$ with G nonterminal bounded context-free, $L(G, O)$ is regular.

Restrictions on CF grammars

A CF grammar is nonterminal bounded if there exists a constant k such that all sentential forms generated by the grammar have at most k nonterminals.

Theorem

For every G/O system $\Omega = (G, O)$ with G nonterminal bounded context-free, $L(G, O)$ is regular.

The sentential forms and observations can be encoded in FA states.

Each state keeps track of nonterminals of the current sentential form and the current mappings for each “piece” of sentential forms of G .

$w = w_0 A_1 w_1 A_2 \dots A_l w_l \quad l \leq k$
we have the automaton state $[v_0 A_1 v_1 \dots A_l v_l]$

A_1, \dots, A_l nonterminal sentential forms.

v_0, v_1, \dots, v_l map each state of the observer to the state after reading

w_0, w_1, \dots, w_l

The state of the observer reading the entire sentential form can be deduced (put as label of the FA). The mappings can be update when a production is used.

Restrictions on CF grammars

Nonterminal bounded grammars are less powerful when observed.

A CF grammar is nonterminal bounded if there exists a constant k such that all sentential forms generated by the grammar have at most k nonterminals.

Theorem

For every G/O system $\Omega = (G, O)$ with G nonterminal bounded context-free, $L(G, O)$ is regular.

The sentential forms and observations can be encoded in FA states.

Each state keeps track of nonterminals of the current sentential form and the current mappings for each “piece” of sentential forms of G .

$w = w_0 A_1 w_1 A_2 \dots A_l w_l \quad l \leq k$
we have the automaton state $[v_0 A_1 v_1 \dots A_l v_l]$

A_1, \dots, A_n nonterminal sentential forms.

v_0, v_1, \dots, v_l map each state of the observer to the state after reading

w_0, w_1, \dots, w_l

The state of the observer reading the entire sentential form can be deduced (put as label of the FA). The mappings can be update when a production is used.

Corollary

For every G/O system $\Omega = (G, O)$ with G regular, $L(G, O)$ is regular.

A pushdown automaton corresponds to leftmost derivations in a type-0 grammar with productions of the form $pA \rightarrow aq\alpha$

Theorem

For every G/O system with G a type-0 grammar with only leftmost derivations and productions of the form $pA \rightarrow aq\alpha$, $L(G,O)$ is CF.

A pushdown automaton corresponds to leftmost derivations in a type-0 grammar with productions of the form $pA \rightarrow aq\alpha$

Theorem

For every G/O system with G a type-0 grammar with only leftmost derivations and productions of the form $pA \rightarrow aq\alpha$, $L(G,O)$ is CF.

There is a PDA accepting the observed language.

For a sentential form $wA\alpha$ the PDA M has read w from the tape and $A\alpha$ is on the stack.

Modify M to get M_O

“regular” information about the stack can be stored on the topmost symbol and update.

For a sentential form $wA\alpha$ M_O knows w (derived) and behavior of O on wA is regular (can be stored on the top stack).

M_O can deduce the observer’s output on $wA\alpha$ and read that symbol.

A pushdown automaton corresponds to leftmost derivations in a type-0 grammar with productions of the form $pA \rightarrow aq\alpha$

Theorem

For every G/O system with G a type-0 grammar with only leftmost derivations and productions of the form $pA \rightarrow aq\alpha$, $L(G,O)$ is CF.

There is a PDA accepting the observed language.

For a sentential form $wA\alpha$ the PDA M has read w from the tape and $A\alpha$ is on the stack.

Modify M to get M_O

“regular” information about the stack can be stored on the topmost symbol and update.

For a sentential form $wA\alpha$ M_O knows w (derived) and behavior of O on wA is regular (can be stored on the top stack).

M_O can deduce the observer’s output on $wA\alpha$ and read that symbol.

Pushdown automata are less computationally universal powerful than context-free grammars when observed.

Every CF language is a finite state transduction of the Dyck set D_2 .

Theorem

There is a leftmost context-free grammar universal m.o. for CF

Consider a CF grammar in Greibach for D_2 interleaved with $\{c,d\}^$.*

The observer can simulate the transducer and produces its output.

*The last symbol from $\{a,a',b,b'\}$ is the last one generated (leftmost derivation)
(the observer knows which step of the fst is observing).*

Symbols c,d are used to produce more than a symbol in sequence (like a fst).

Observing productions

The observer can only see the sequence of productions implemented by the grammar.

$$G = (N, T, S, P)$$

$$N = \{S, A, B, C\} \quad T = \{t, p\}$$

$$P = \{ S \rightarrow pS, S \rightarrow p, S \rightarrow A, \\ A \rightarrow AB, A \rightarrow C, B \rightarrow C, C \rightarrow t \}$$

$$O'(w) = \begin{cases} \lambda & \text{if } w = S \\ a & \text{if } w \in AB^* \\ b & \text{if } w \in C^+B^* \\ c & \text{if } w \in t^+B^* \\ \perp & \text{else} \end{cases}$$

$$S \Rightarrow A \Rightarrow AB \Rightarrow ABB \Rightarrow CBB \Rightarrow CCB \Rightarrow CCC \Rightarrow tCC \Rightarrow ttC \Rightarrow ttt$$

$$S \Rightarrow A \Rightarrow AB \Rightarrow ABB \Rightarrow CBB \Rightarrow \cancel{CBC} \Rightarrow CCC \Rightarrow CCt \Rightarrow tCt \Rightarrow ttt$$

same sequence of productions / same string as output

Observing productions

The observer can only see the sequence of productions implemented by the grammar.

$$G = (N, T, S, P)$$

$$N = \{S, A, B, C\} \quad T = \{t, p\}$$

$$P = \{ S \rightarrow pS, S \rightarrow p, S \rightarrow A, \\ A \rightarrow AB, A \rightarrow C, B \rightarrow C, C \rightarrow t \}$$

$$O'(w) = \begin{cases} \lambda & \text{if } w = S \\ a & \text{if } w \in AB^* \\ b & \text{if } w \in C^+B^* \\ c & \text{if } w \in t^+B^* \\ \perp & \text{else} \end{cases}$$

$$S \Rightarrow A \Rightarrow AB \Rightarrow ABB \Rightarrow CBB \Rightarrow CCB \Rightarrow CCC \Rightarrow tCC \Rightarrow ttC \Rightarrow ttt$$

$$S \Rightarrow A \Rightarrow AB \Rightarrow ABB \Rightarrow CBB \Rightarrow \cancel{CBC} \Rightarrow CCC \Rightarrow CCt \Rightarrow tCt \Rightarrow ttt$$

same sequence of productions / same string as output

Theorem

For each k , there exists a context-free grammar that is universal m.o. of productions for the family of languages accepted by partially blink k -counter automata (all semilinear and contained in CS).

Perspectives

“Everything” can be computed by finding the “right” observer, without changing the observed system.

Perspectives

“Everything” can be computed by finding the “right” observer, without changing the observed system.

How much does the (observed) evolution of a fixed system “differ” when watched by “different” observers?

Perspectives

“Everything” can be computed by finding the “right” observer, without changing the observed system.

How much does the (observed) evolution of a fixed system “differ” when watched by “different” observers?

Finding grammars whose observed evolutions always respect certain properties (independently of the used observer).

Perspectives

“Everything” can be computed by finding the “right” observer, without changing the observed system.

How much does the (observed) evolution of a fixed system “differ” when watched by “different” observers?

Finding grammars whose observed evolutions always respect certain properties (independently of the used observer).

Considering G/O systems with weaker observers (not “trashing”, not empty labels, etc..)

Perspectives

“Everything” can be computed by finding the “right” observer, without changing the observed system.

How much does the (observed) evolution of a fixed system “differ” when watched by “different” observers?

Finding grammars whose observed evolutions always respect certain properties (independently of the used observer).

Considering G/O systems with weaker observers (not “trashing”, not empty labels, etc..)

Inference of the (observed) system (systems biology)

*“The real voyage of discovery consists not in seeking new landscapes
but in having new eyes”*

Marcel Proust