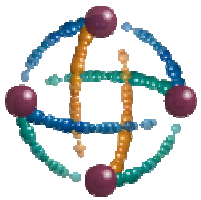
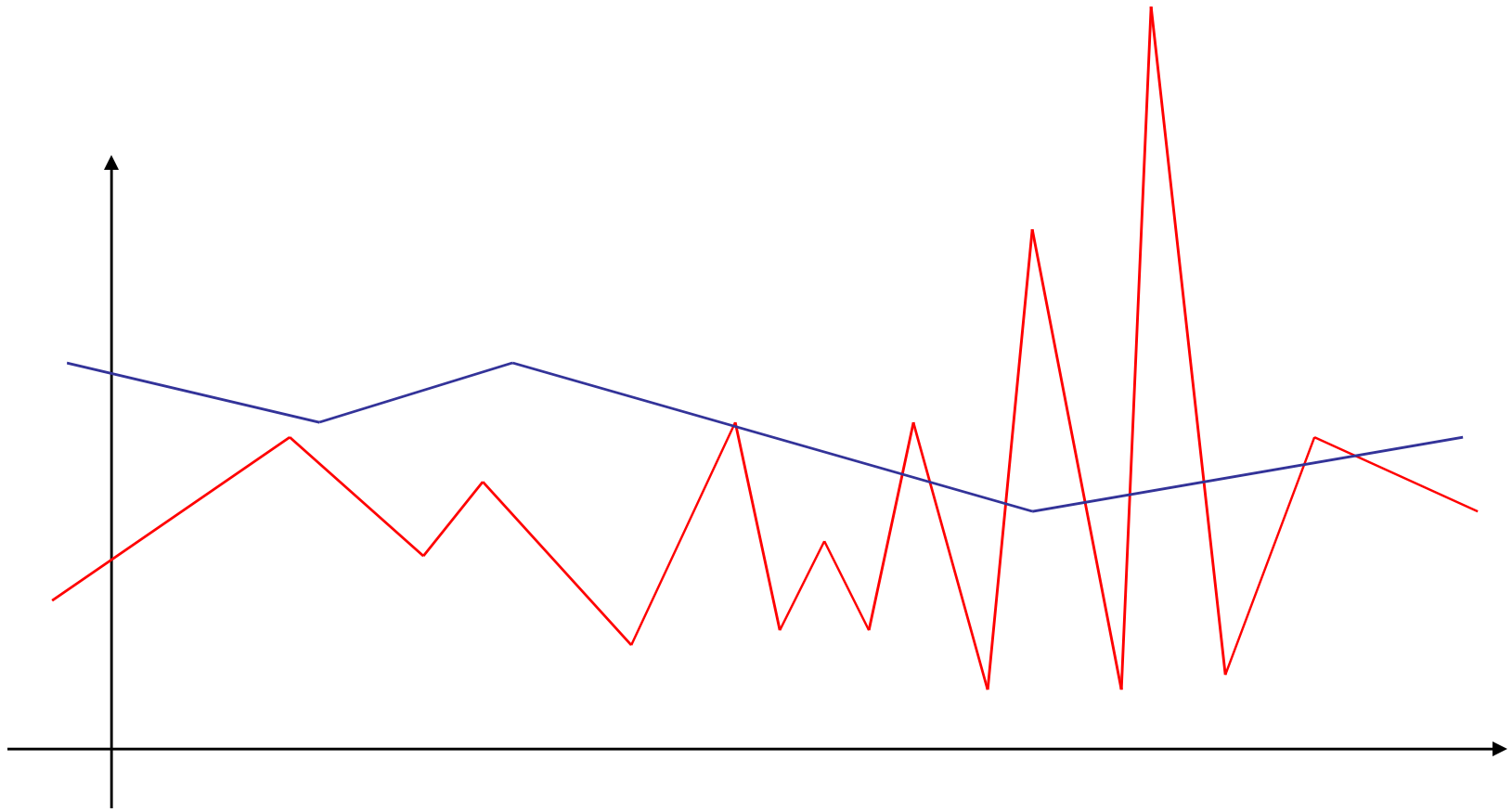
Computing by Observing

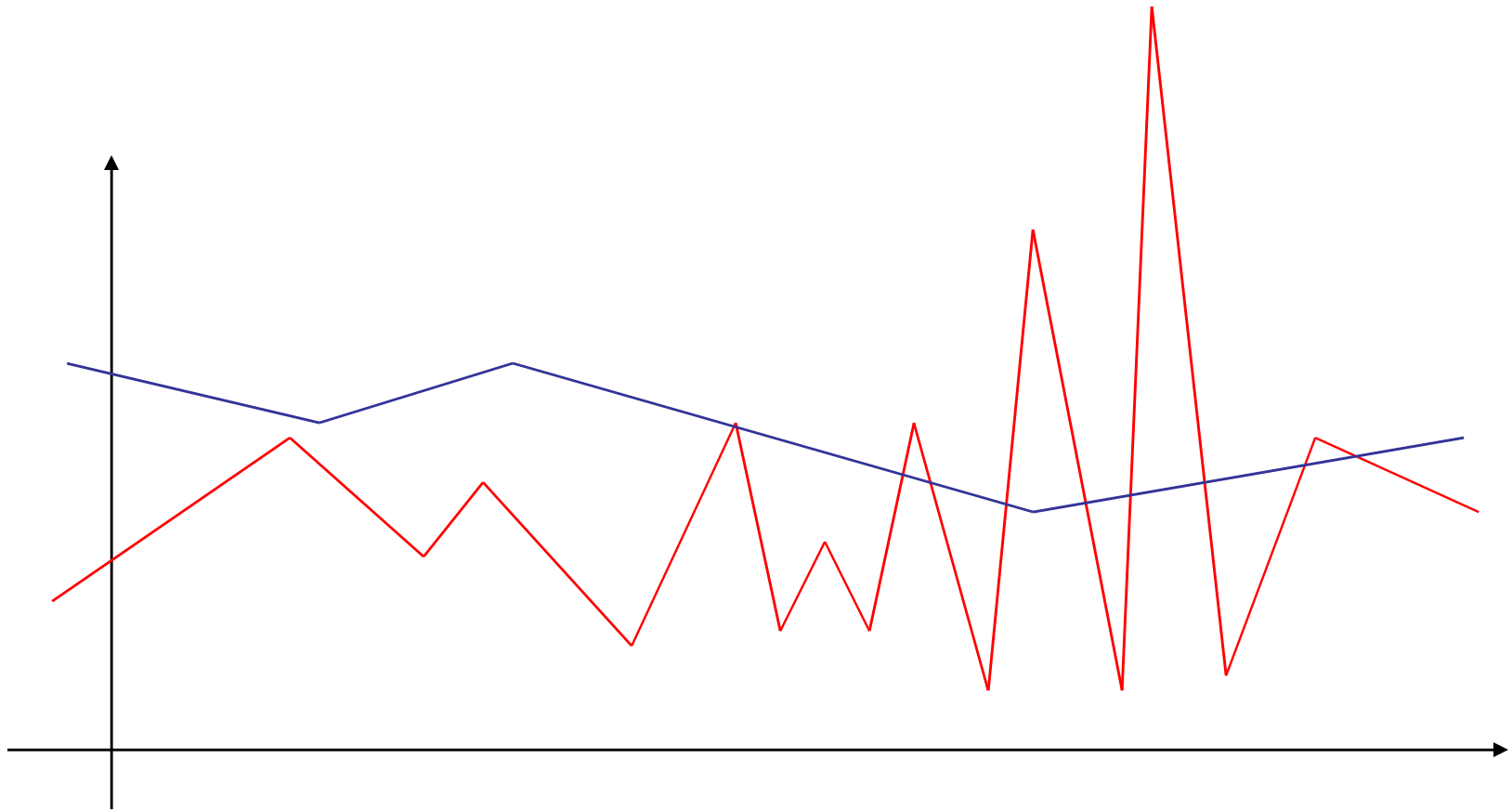
Matteo Cavaliere



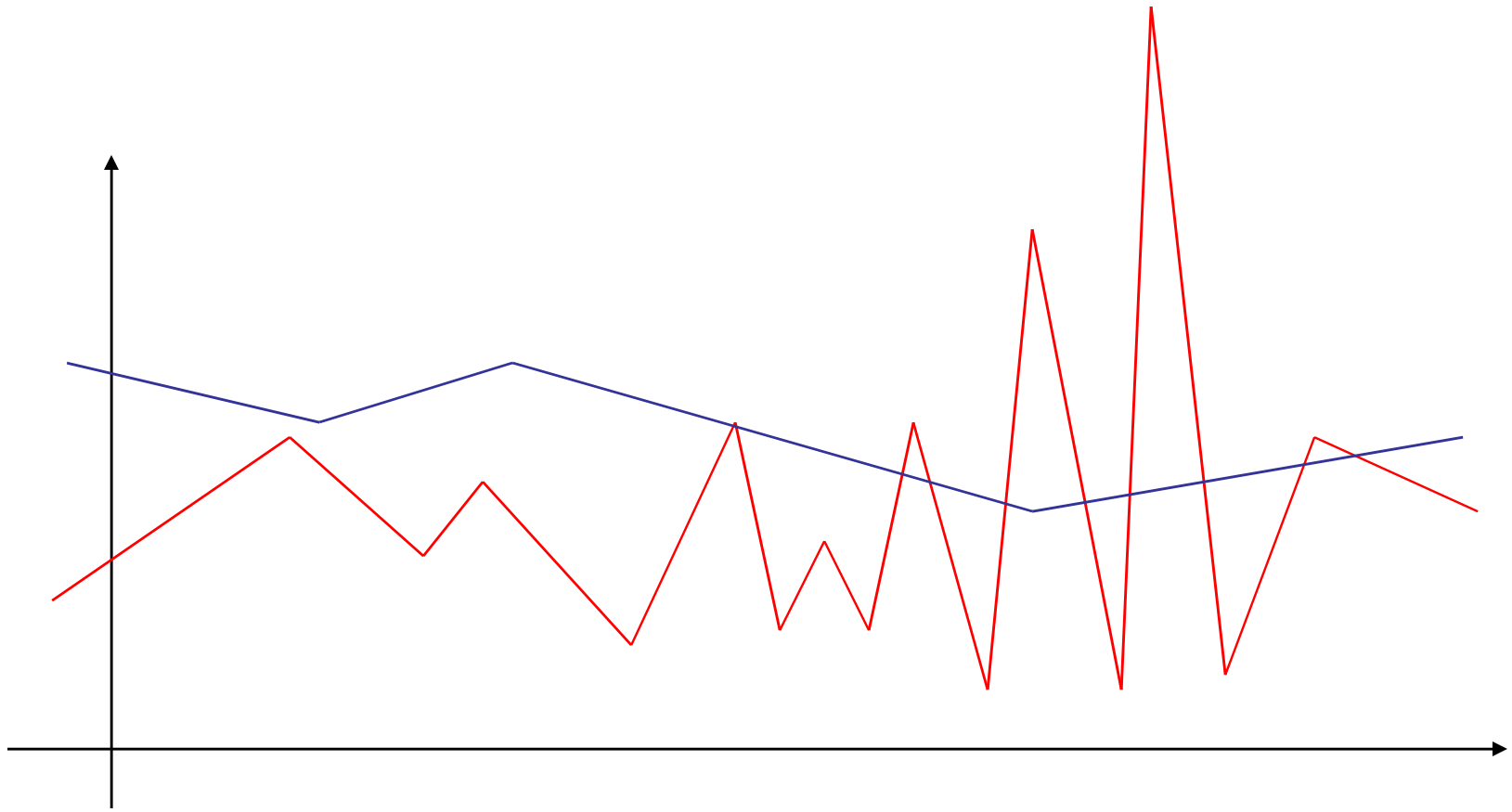
CoSBi, Trento, Italy



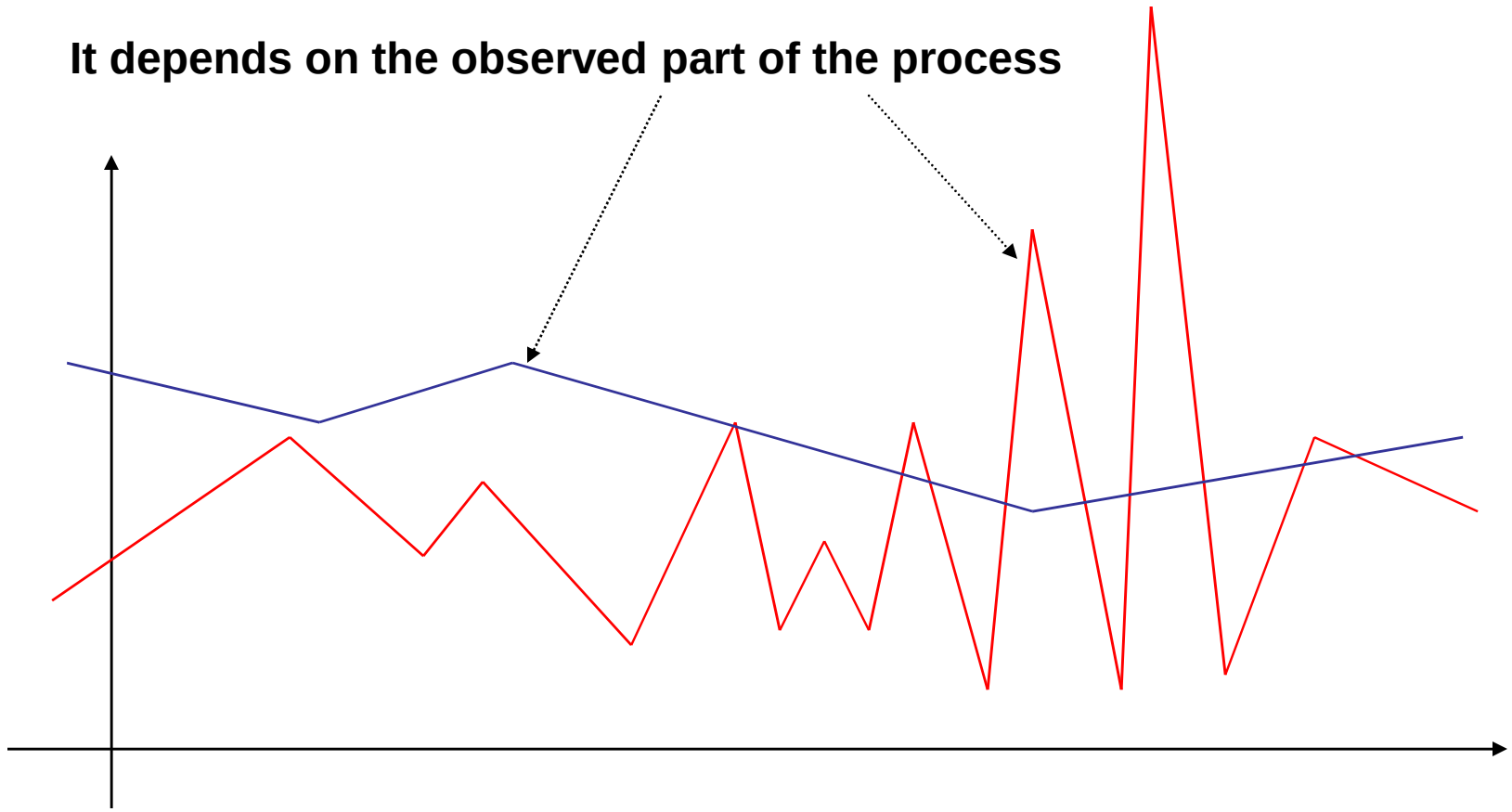
The **result** of an experiment is the observation (interpretation) of a process.



A momentary picture of the observed system is irrelevant, and rather **the entire trajectory** of the system is evaluated.



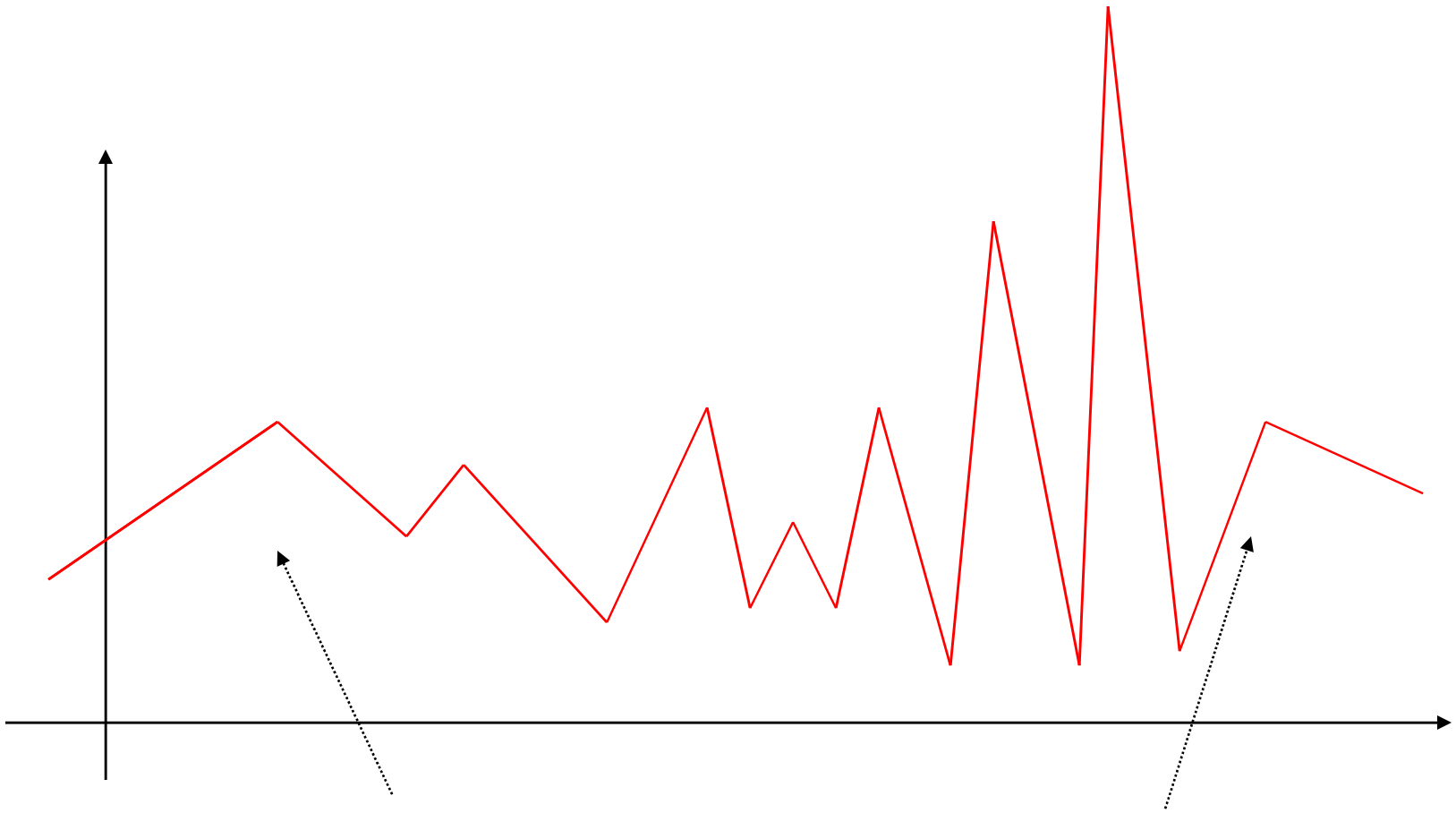
Can we evaluate the **complexity** of the observed trajectory?



Can we evaluate the complexity of the observed trajectory?

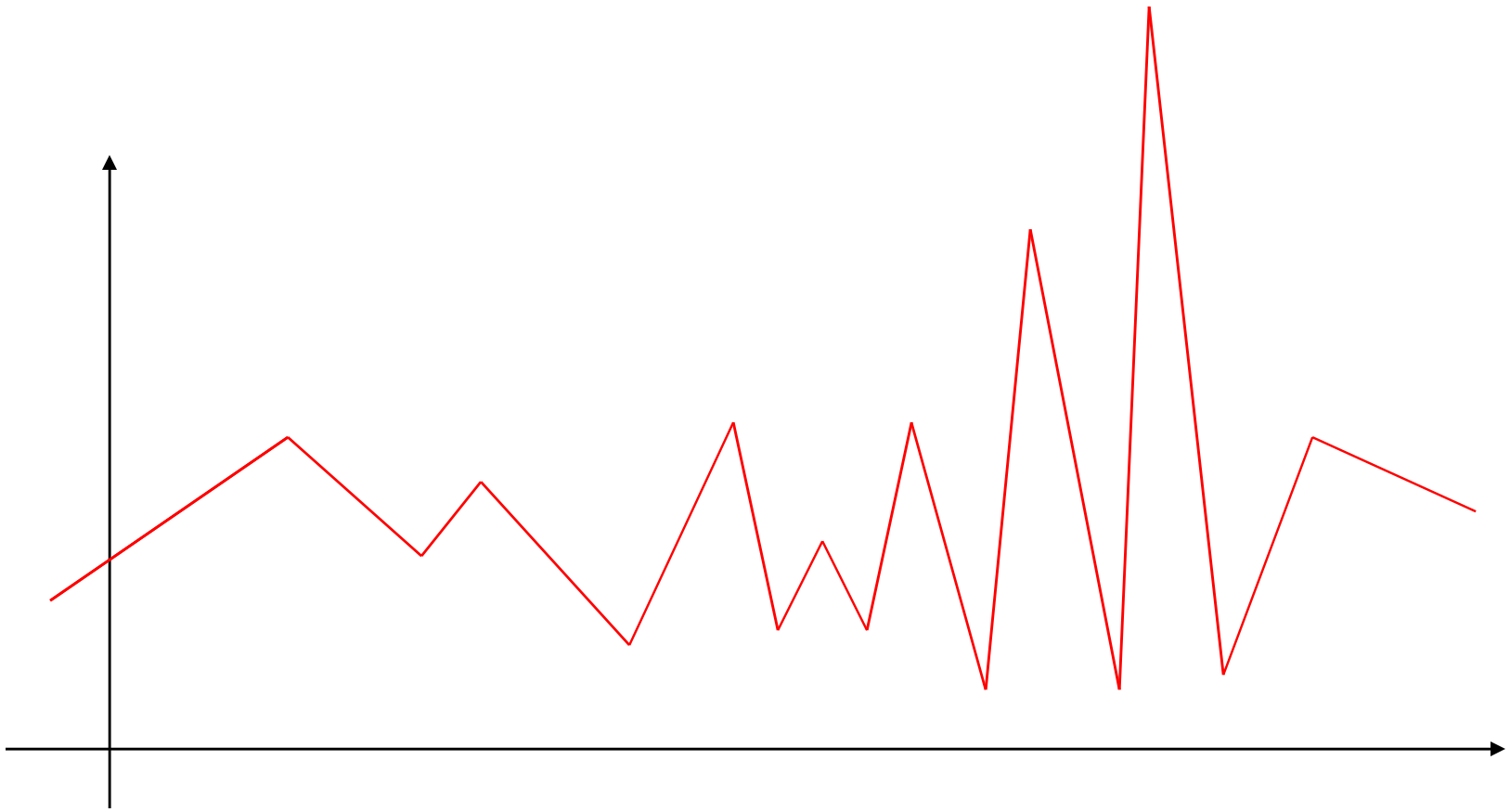
It depends on how much of the process is observed

Can we evaluate the complexity of the observed trajectory?



It depends on how much of the process is observed

Can we evaluate the complexity of the observed trajectory?



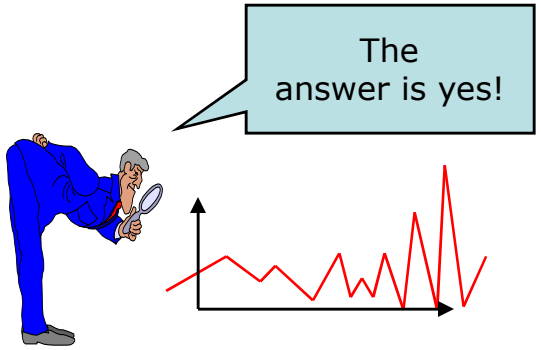
Can we use the trajectory of an experiment to **compute** whatever we want?

The **result** of an **experiment** is the observation (interpretation) of a process.

The **result** of an **experiment** is the observation (interpretation) of a process.

The **result** of a **computation** is

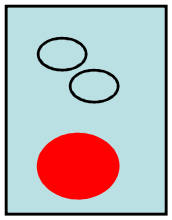
the observation (interpretation) of a process



Step 1

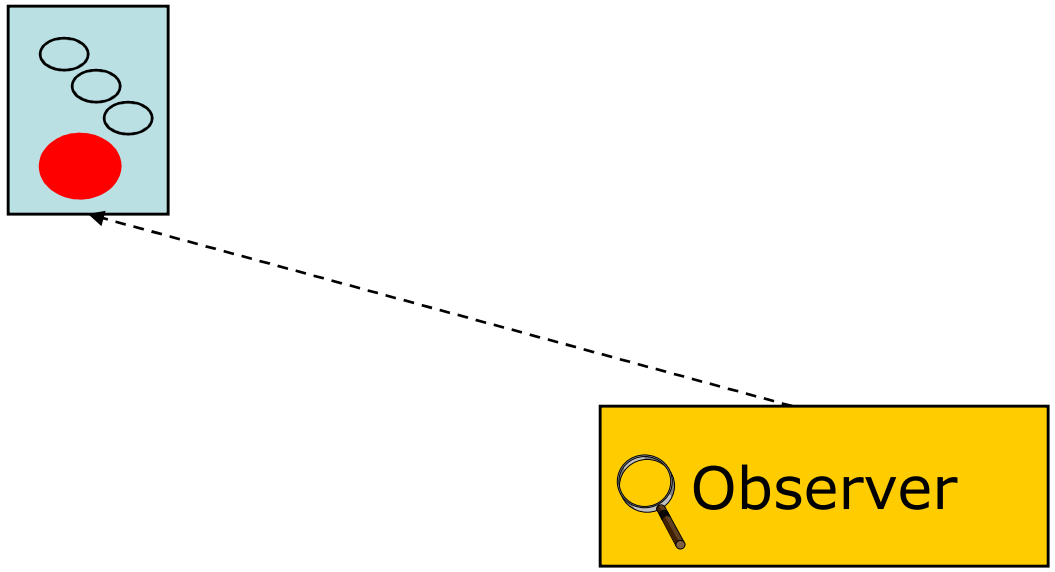
Observing the evolution of the system

state of the system



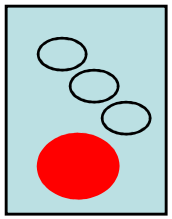
Step 1

Observing the evolution of the system



Step 1

Observing the evolution of the system



R

Only part of the system can be really seen.

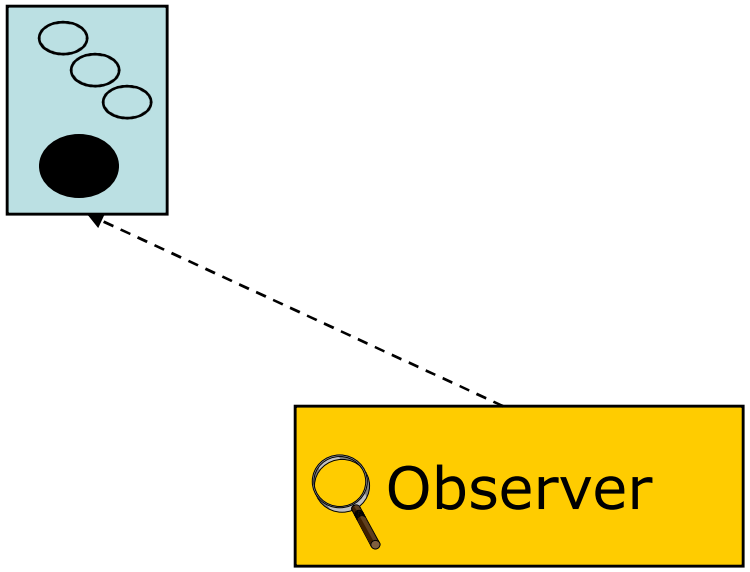
A symbol is associated to the observed state.

It is the amount of information we can extract from the system.

$\Sigma = \{R, B\}$

Step 2

Observing the evolution of the system



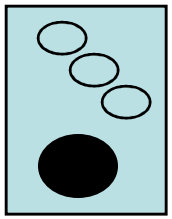
R

Observations of the system can be iterated.

$\Sigma = \{R, B\}$

Step 2

Observing the evolution of the system



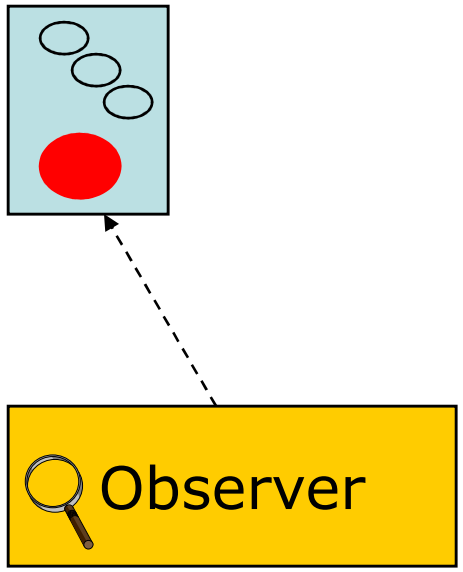
R B

Observations of the system can be iterated

$\Sigma = \{R, B\}$

Step 3

Observing the evolution of the system

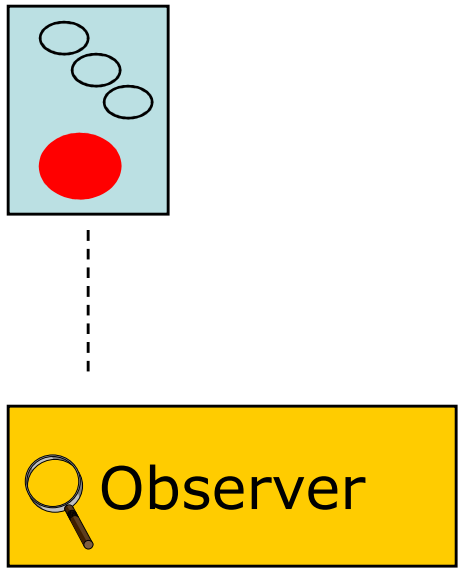


R B

$\Sigma = \{R, B\}$

Step 3

Observing the evolution of the system

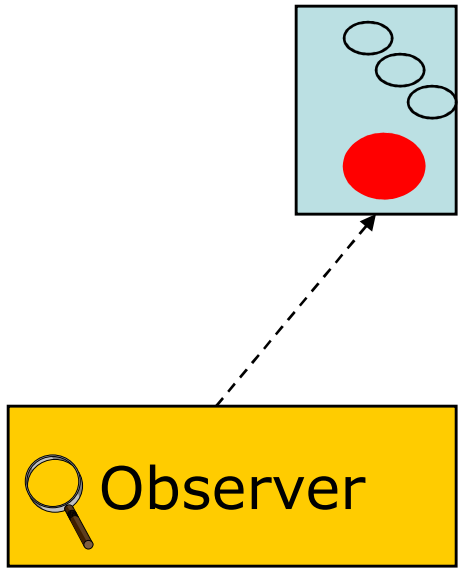


R B R

$\Sigma = \{R, B\}$

Step 4

Observing the evolution of the system

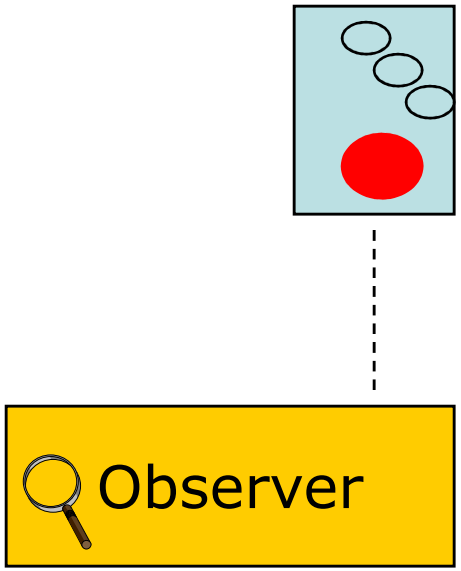


R B R

$\Sigma = \{R, B\}$

Step 4

Observing the evolution of the system

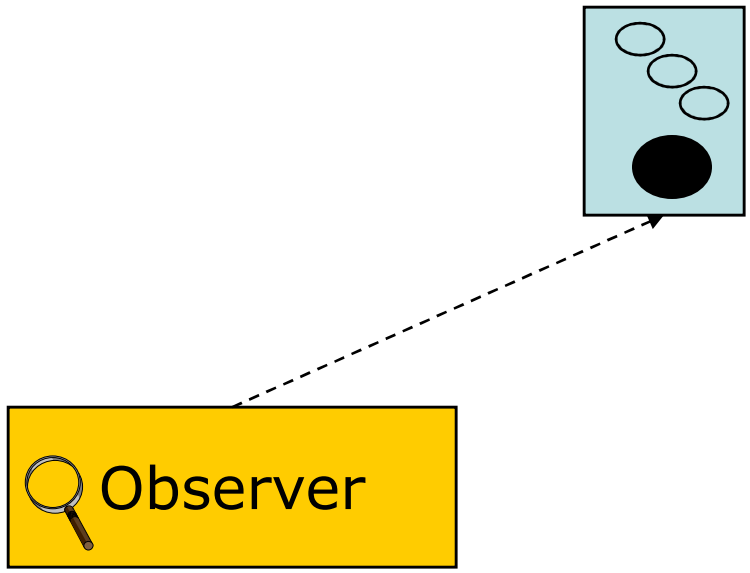


R B R R

$\Sigma = \{R, B\}$

Step 5

Observing the evolution of the system

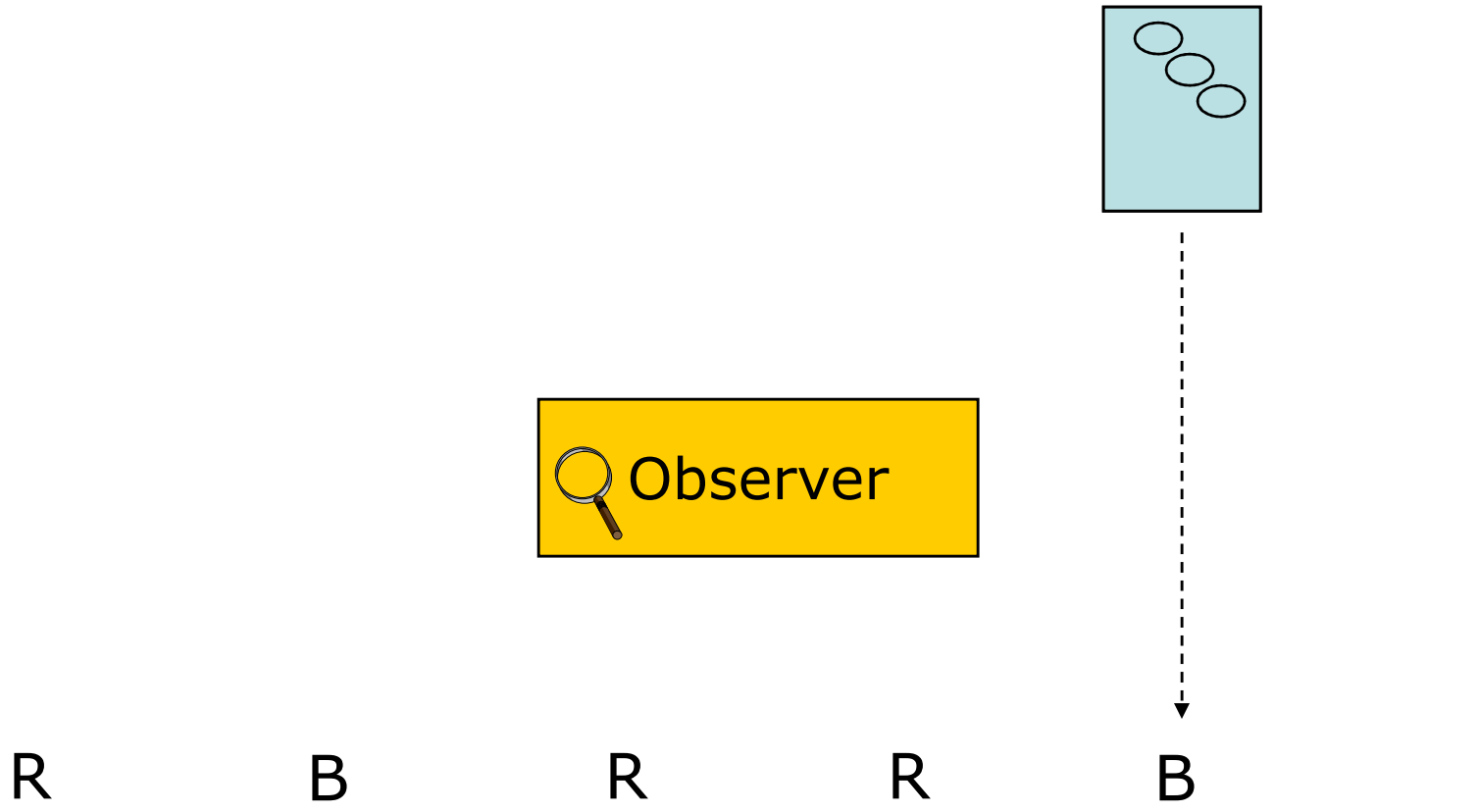


R B R R

$\Sigma = \{R, B\}$

Step 5

Observing the evolution of the system



$\Sigma = \{R, B\}$

R B R R B

$\Sigma = \{R, B\}$

Result of the computation = Observed trajectory

R B R R B

$\Sigma = \{R, B\}$

Result of the computation = Observed trajectory

R **B** **R** **R** **B**

*It is a **string** over* Σ

Σ = {R, B}

If the observed system is **non-deterministic**, then we get a **set of observed trajectories**, that is a language over Σ.

Result of the computation = Observed trajectory

R B R R B

*It is a **string** over Σ*

Σ = {R, B}

If the observed system is **non-deterministic**, then we get a **set of observed trajectories**, that is a language over Σ.

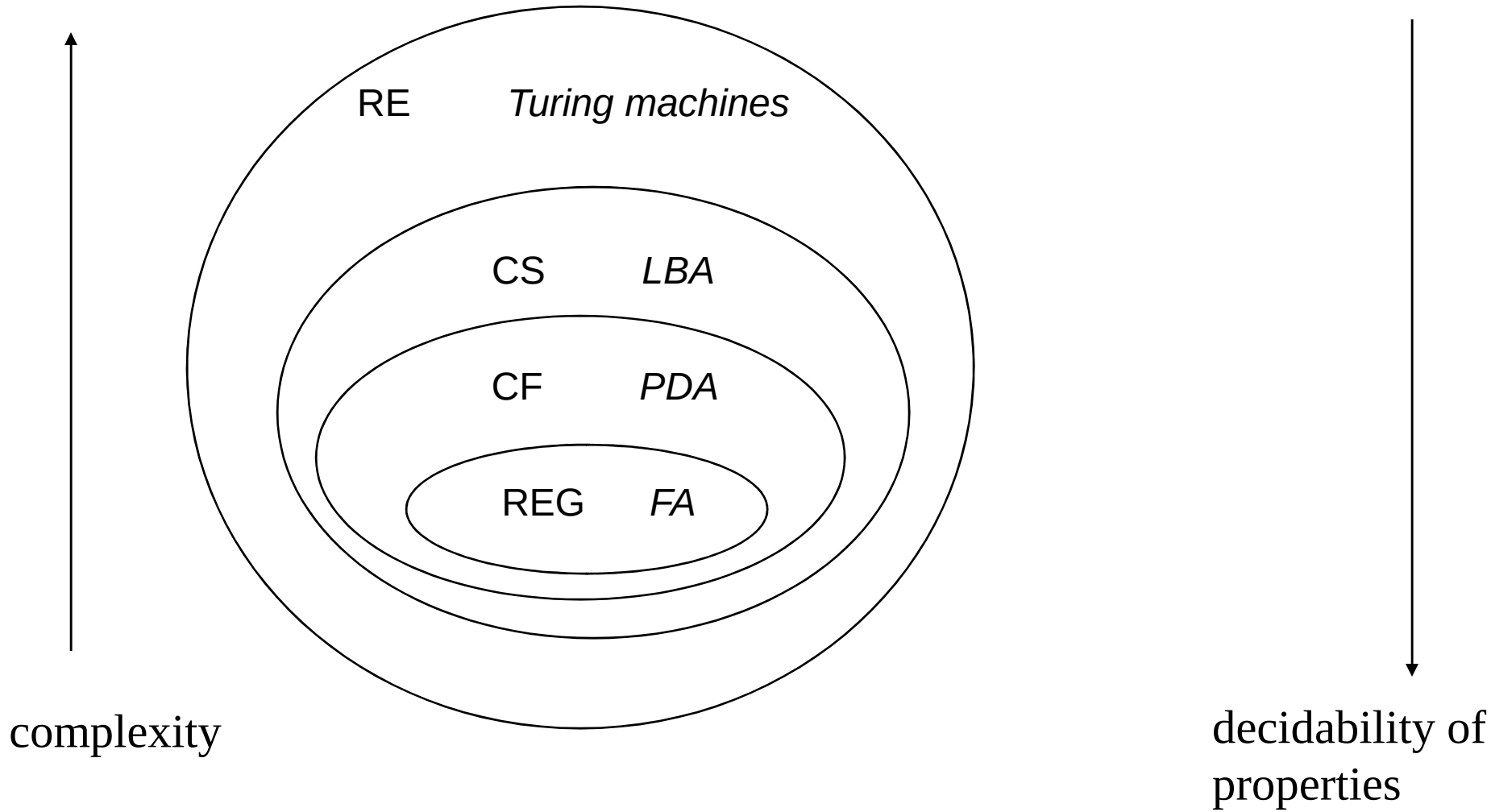
We can use formal language theory to study its complexity!

Result of the computation = Observed trajectory

R **B** **R** **R** **B**

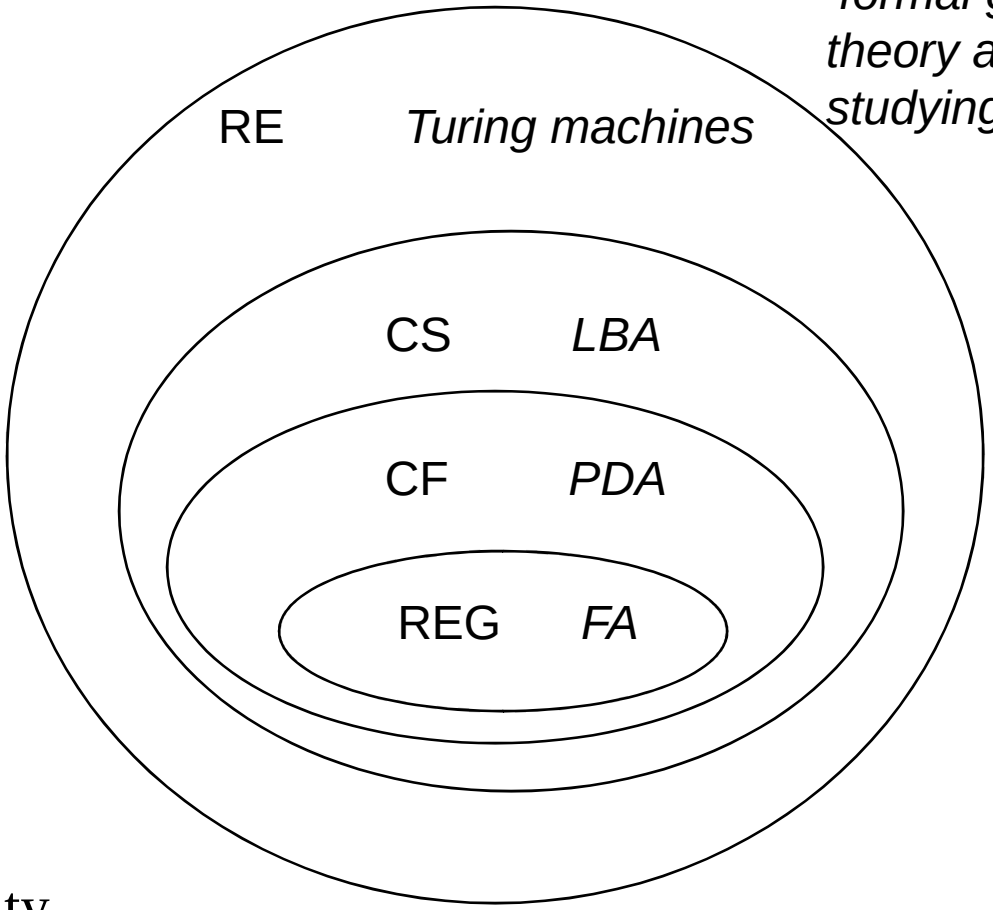
*It is a **string** over Σ*

Σ = {R, B}



Formal language theory can provide a formal tool to analyze complexity of sets of strings (languages)

Rewriting systems, formal grammars and automata theory are the formal tools for studying such hierarchy



$S \rightarrow pS$,
$S \rightarrow p$

$S \Rightarrow pS \Rightarrow ppS \Rightarrow ppp$

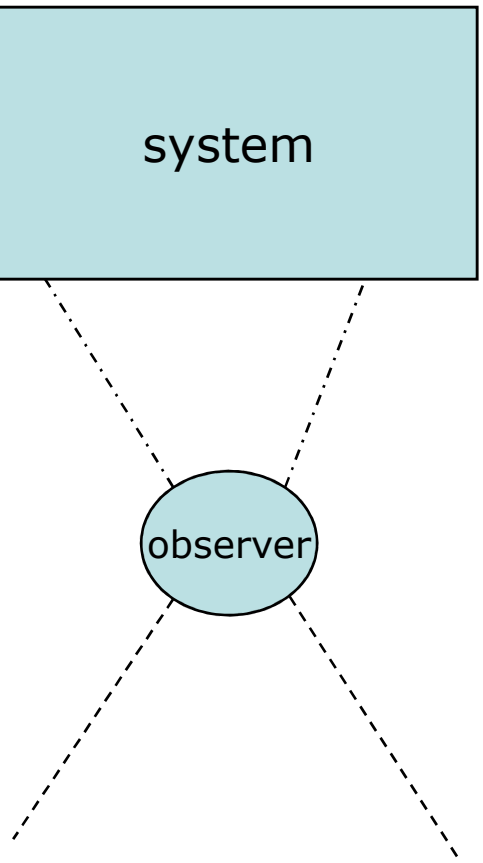
complexity

decidability of properties

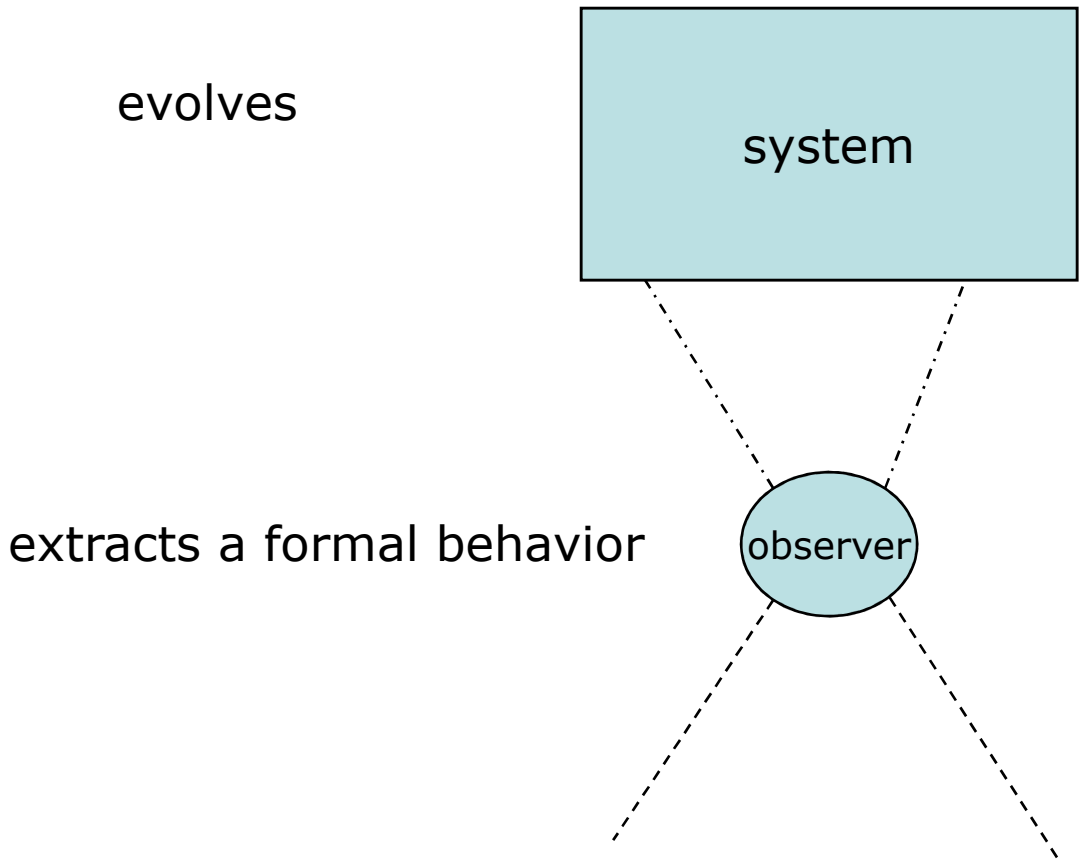
Formal language theory can provide a formal tool to analyze complexity of sets of strings (languages)

The framework

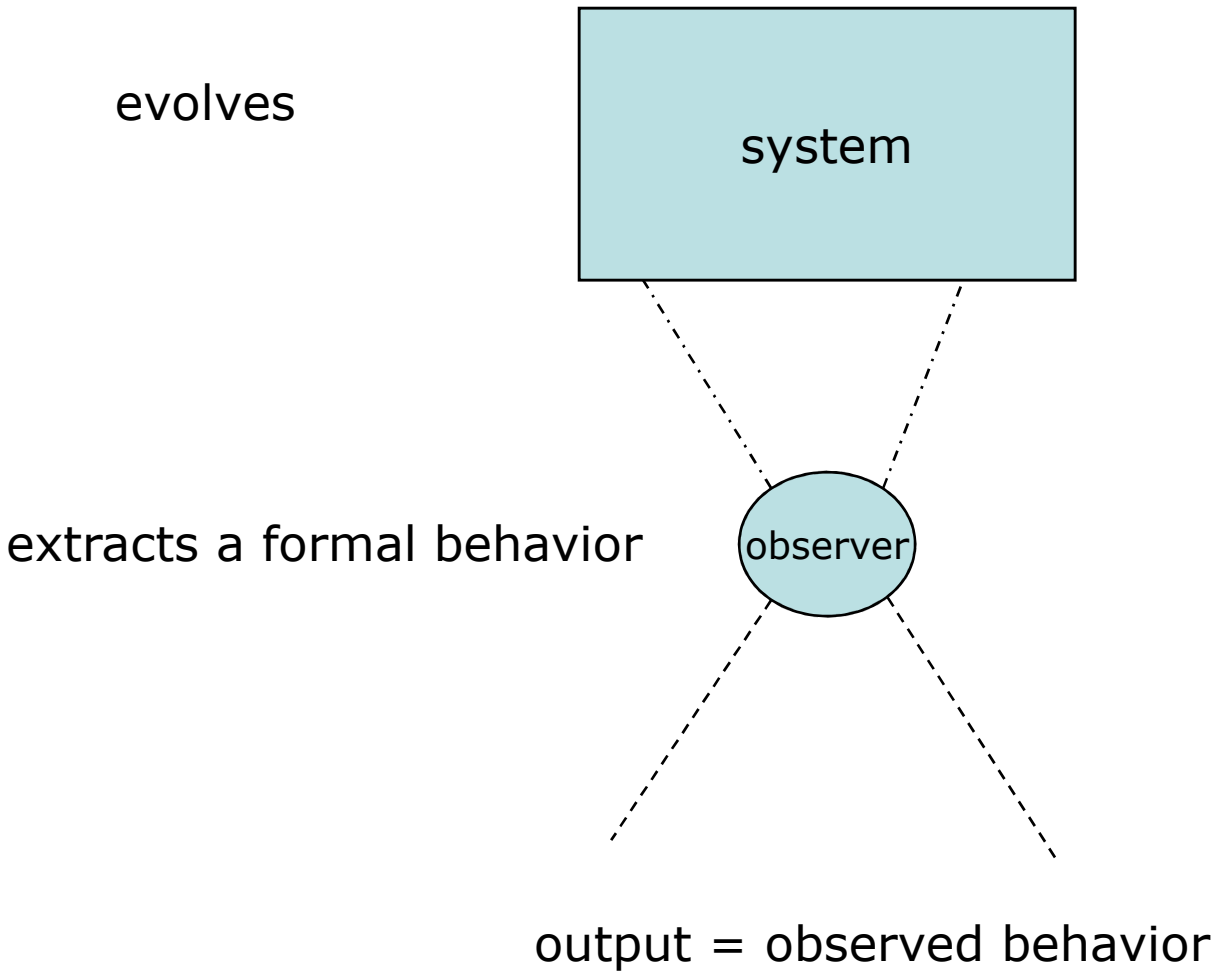
evolves



The framework



The framework



Natural Computing

Natural Computing

The observed system can be the formal model of a biological process.

Natural Computing

The observed system can be the formal model of a biological process.

Many in literature: *sticker systems, splicing systems, membrane systems, etc..*

Natural Computing

The observed system can be the formal model of a biological process.

Many in literature: *sticker systems, splicing systems, membrane systems, etc..*

Trade-off between the complexity of the obseved system and the complexity of the observer.

Natural Computing

The observed system can be the formal model of a biological process.

Many in literature: *sticker systems, splicing systems, membrane systems, etc..*

Trade-off between the complexity of the obseved system and the complexity of the observer.

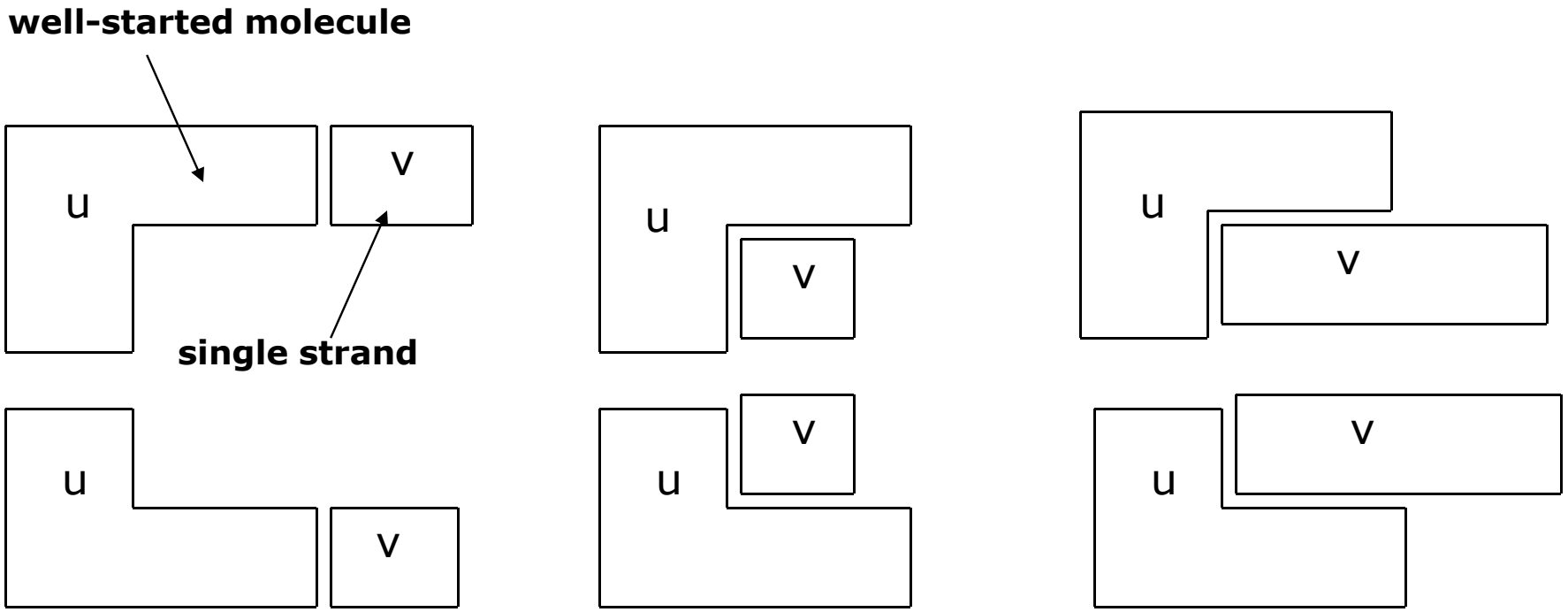
simple observer + simple observed system = ?

Sticker Systems

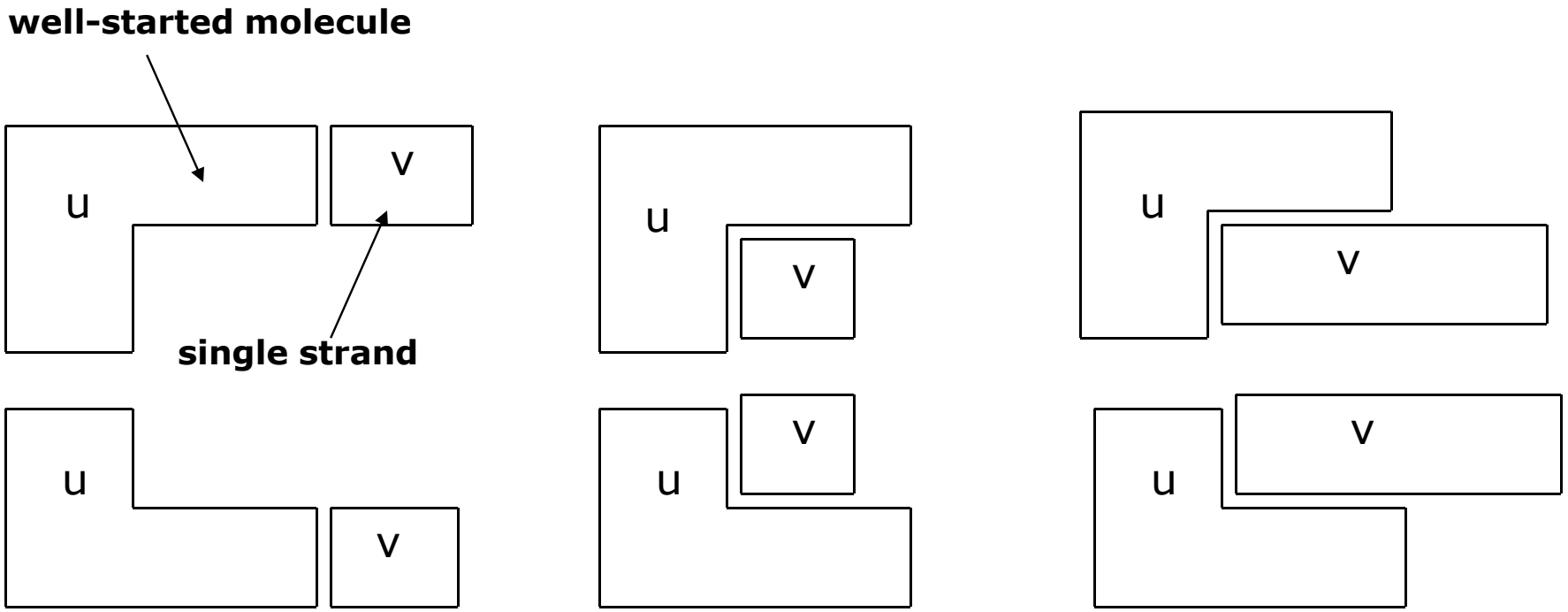
Sticker systems are formal model of the operation of annealing (and ligation) largely used in DNA computing area.

Basic operation of a sticker system is the **sticking operation** that construct double stranded sequence out of DNA dominoes.

Simple regular sticker system.



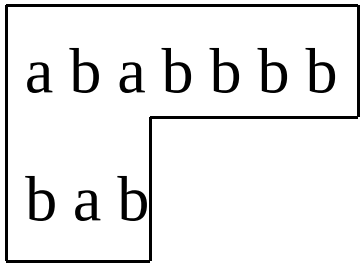
Simple regular sticker system.



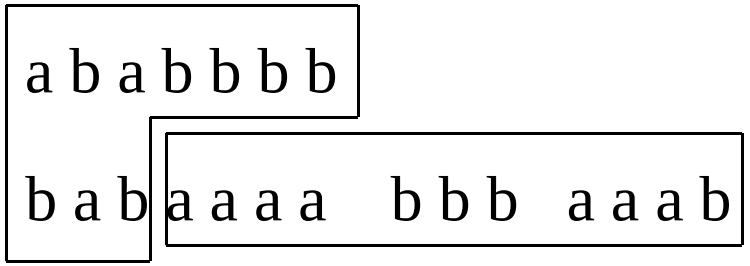
Sticker operation

new well started molecule =
well started molecule + a single strand

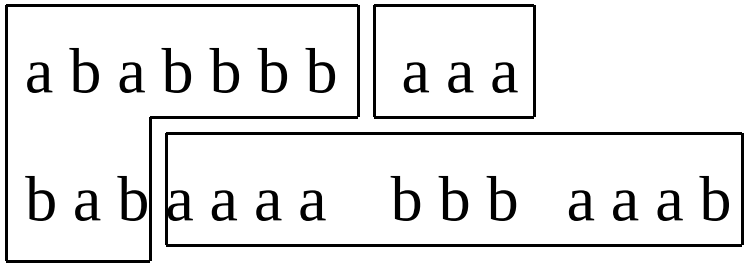
Complete computation starts from an initial well-started molecule and terminates when a complete double-strand molecule is derived.



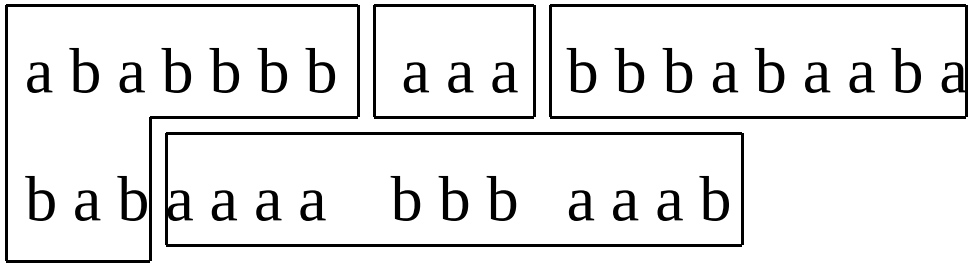
Complete computation starts from an initial well-started molecule and terminates when a complete double-strand molecule is derived.



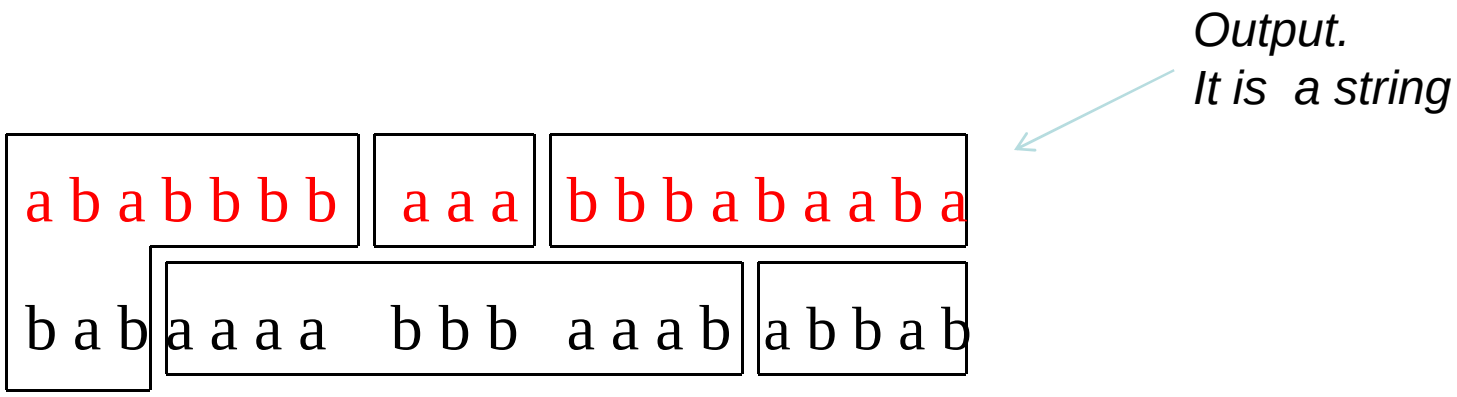
Complete computation starts from an initial well-started molecule and terminates when a complete double-strand molecule is derived.



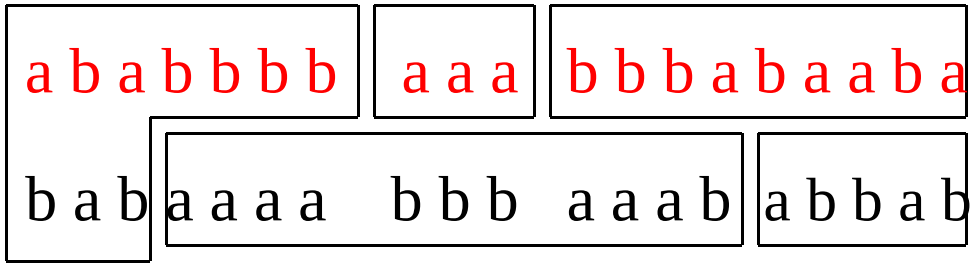
Complete computation starts from an initial well-started molecule and terminates when a complete double-strand molecule is derived.



Complete computation starts from an initial well-started molecule and terminates when a complete double-strand molecule is derived.

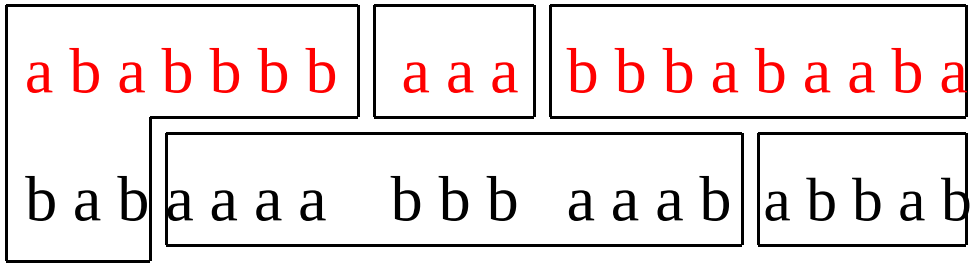


Complete computation starts from an initial well-started molecule and terminates when a complete double-strand molecule is derived.



non determinism: one gets a set of strings (language).

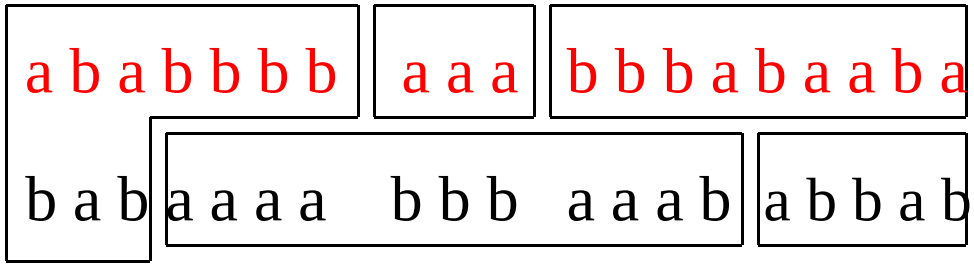
Complete computation starts from an initial well-started molecule and terminates when a complete double-strand molecule is derived.



non determinism: one gets a set of strings (language).

the family of languages generated by simple regular sticker systems is strictly included in the family of regular languages.

Complete computation starts from an initial well-started molecule and terminates when a complete double-strand molecule is derived.

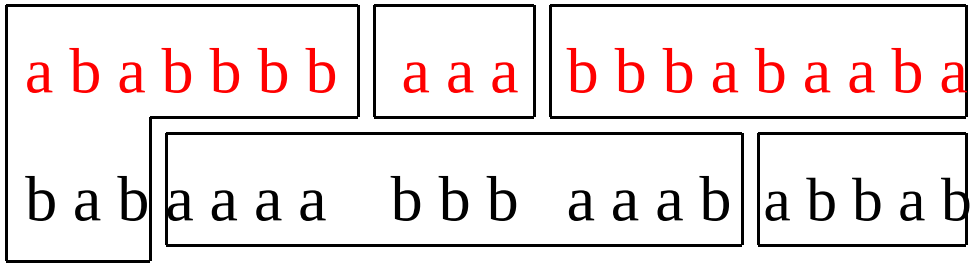


non determinism: one gets a set of strings (language).

The process is not really considered. Only the final obtained strand is considered as result.

the family of languages generated by simple regular sticker systems is strictly included in the family of regular languages.

Complete computation starts from an initial well-started molecule and terminates when a complete double-strand molecule is derived.



non determinism: one gets a set of strings (language).

The process is not really considered. Only the final obtained strand is considered as result. We can consider the process by adding an observer.

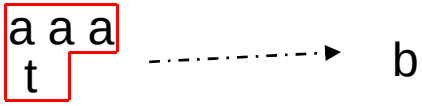
the family of languages generated by simple regular sticker systems is strictly included in the family of regular languages.

*The observer must be **simple**:e.g., can be implemented by a finite state automaton*

$$O(w) = \begin{cases} b, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^* \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^* \end{pmatrix}, \\ d, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^*\mathbf{c} \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^*\mathbf{g} \end{pmatrix}, \\ \lambda, & \text{otherwise.} \end{cases}$$

produced output

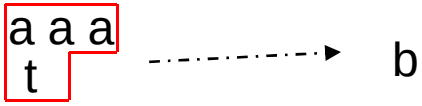
observed double strand



$$O(w) = \begin{cases} b, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^* \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^* \end{pmatrix}, \\ d, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^*\mathbf{c} \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^*\mathbf{g} \end{pmatrix}, \\ \lambda, & \text{otherwise.} \end{cases}$$

produced output

observed double strand

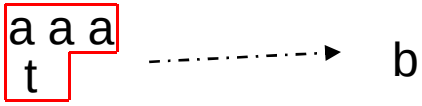


$$O(w) = \begin{cases} b, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^* \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^* \end{pmatrix}, \\ d, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^*\mathbf{c} \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^*\mathbf{g} \end{pmatrix}, \\ \lambda, & \text{otherwise.} \end{cases}$$

A strand is marked.

produced output

observed double strand



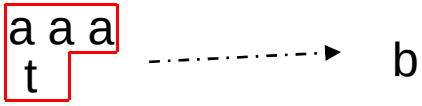
$$O(w) = \begin{cases} b, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^* \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^* \end{pmatrix}, \\ d, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^*\mathbf{c} \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^*\mathbf{g} \end{pmatrix}, \\ \lambda, & \text{otherwise.} \end{cases}$$

produced output

observed double strand

A strand is marked.

The observer can watch the marked strand and follows it during the entire process.



$$O(w) = \begin{cases} b, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^* \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^* \end{pmatrix}, \\ d, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^*\mathbf{c} \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^*\mathbf{g} \end{pmatrix}, \\ \lambda, & \text{otherwise.} \end{cases}$$

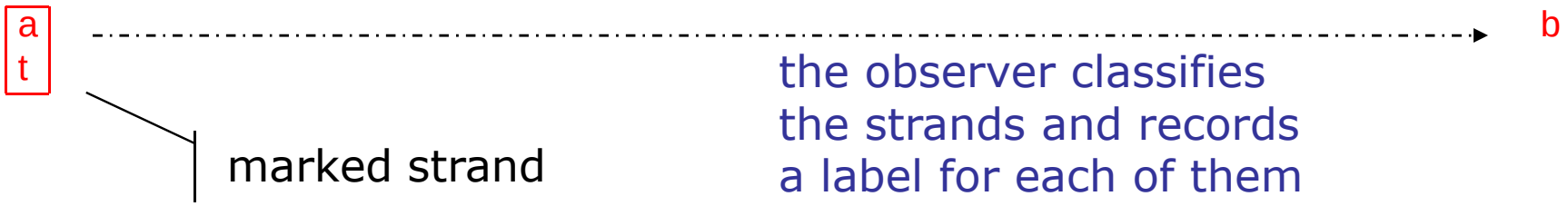
produced output

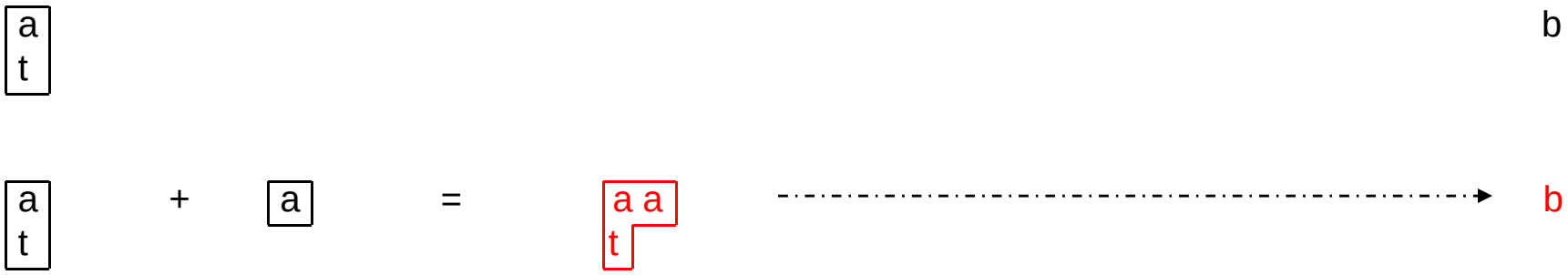
observed double strand

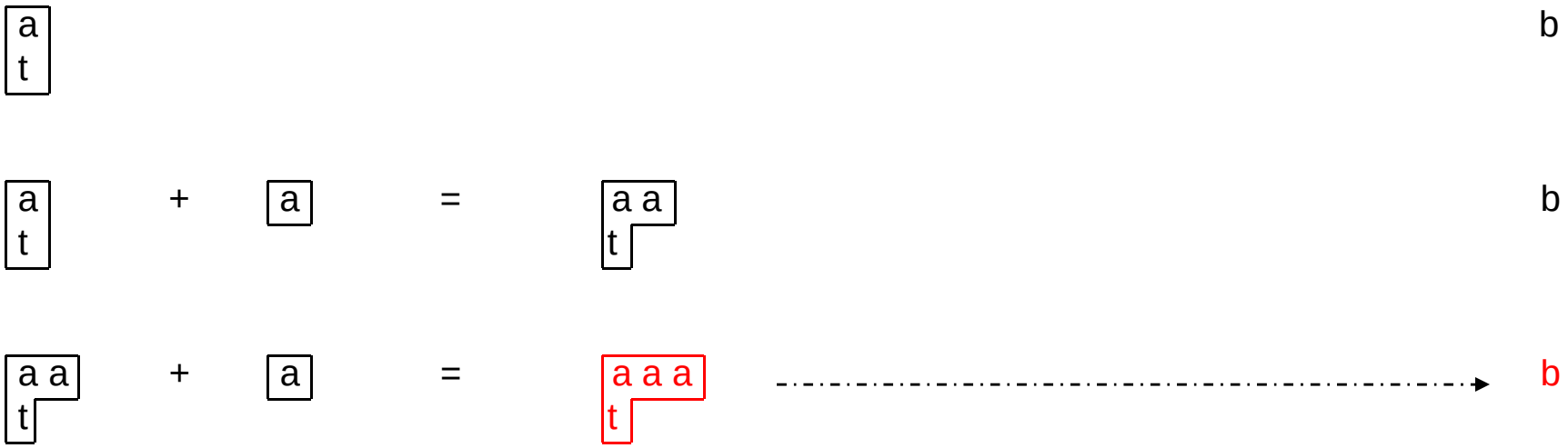
A strand is marked.

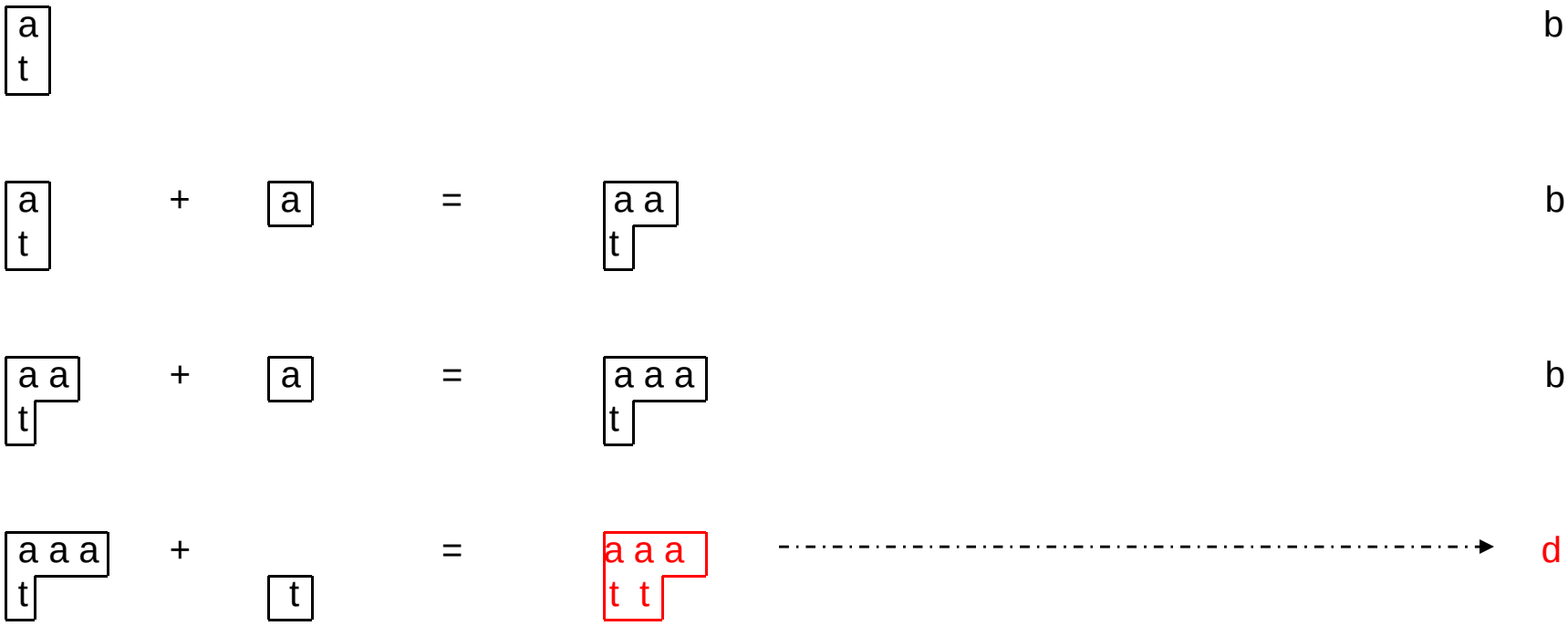
The observer can watch the marked strand and follows it during the entire process.

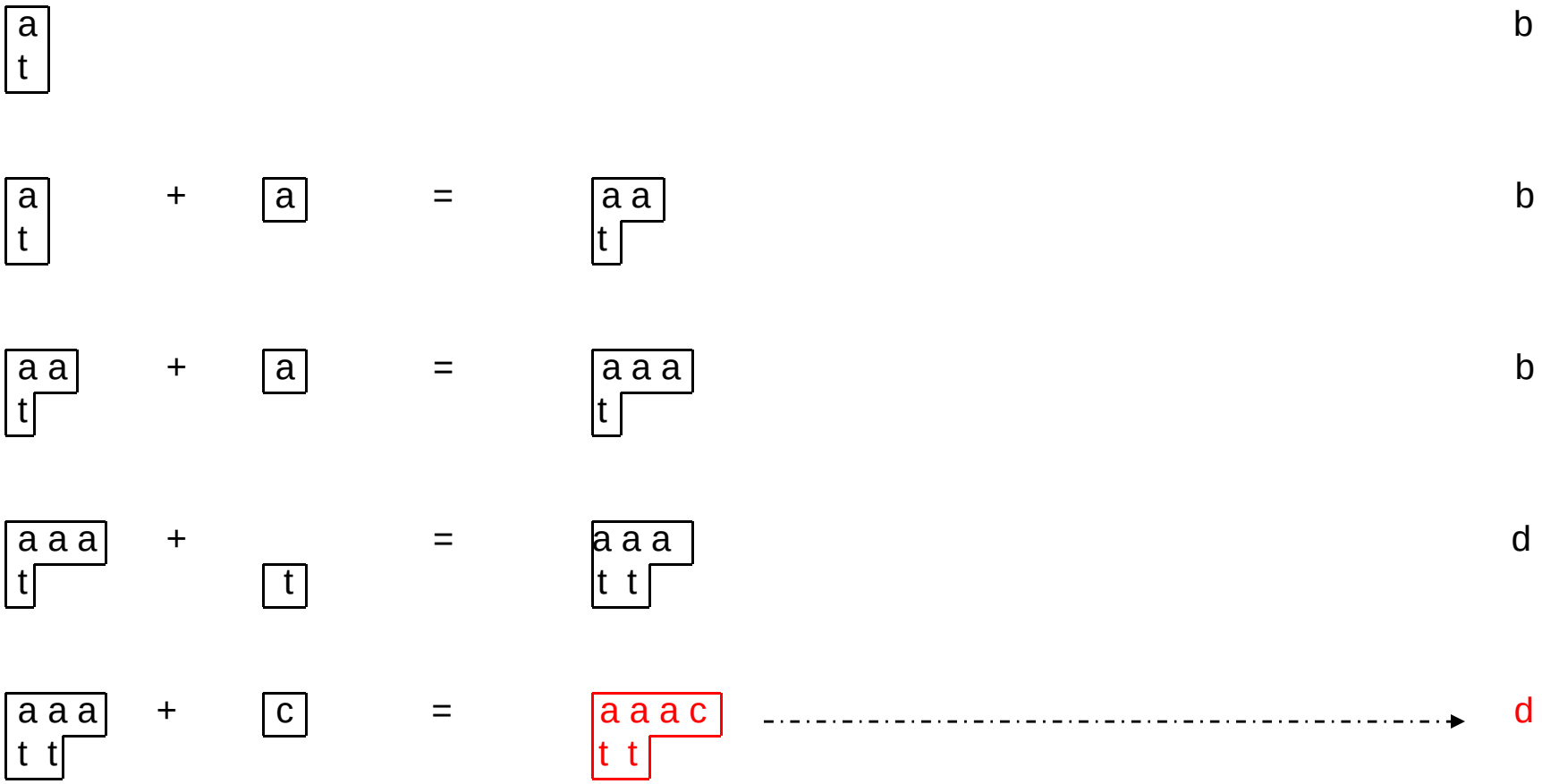
The result of a computation is the observed behaviour of the marked strand.

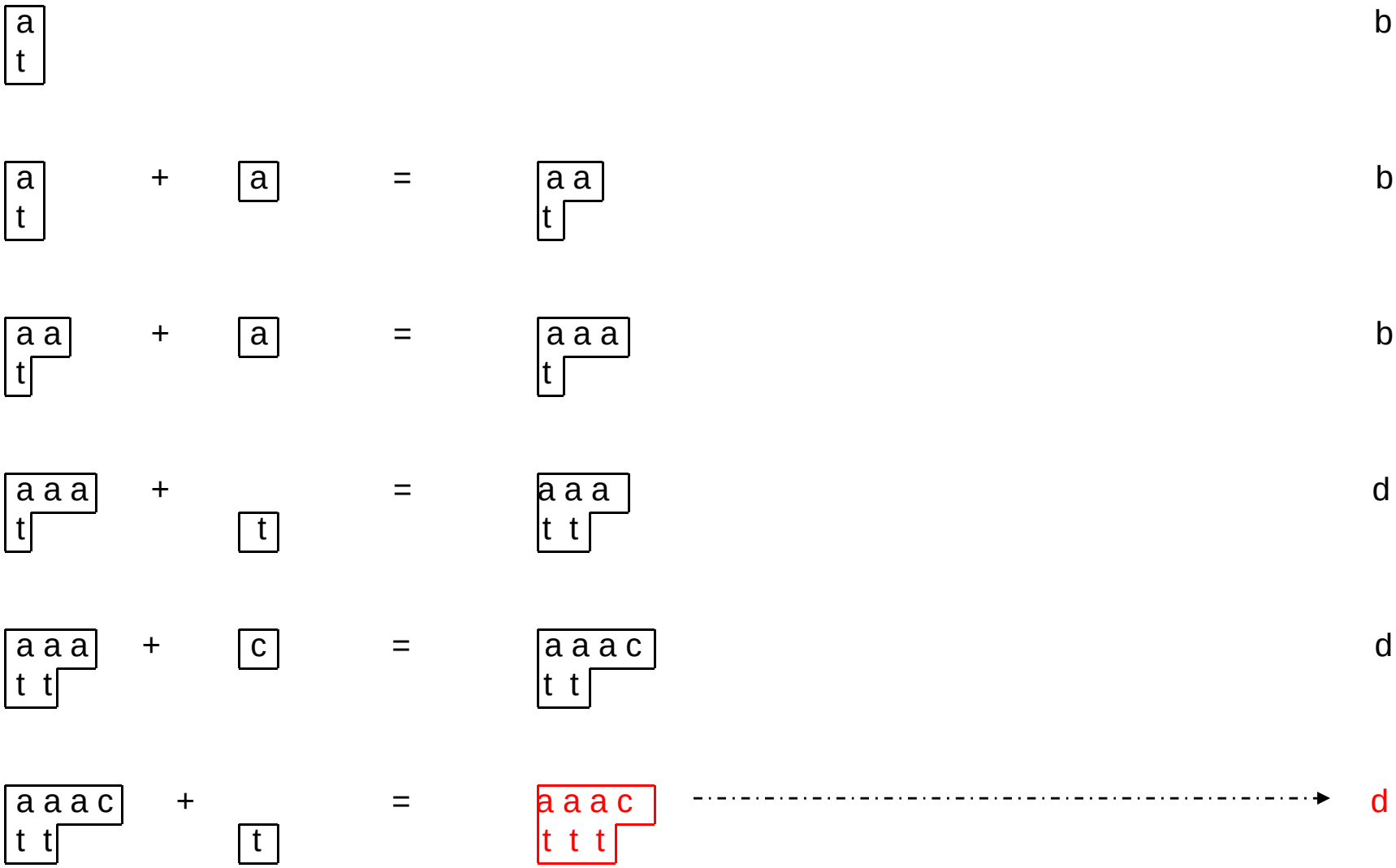

a
t
b
marked strand
the observer classifies
the strands and records
a label for each of them

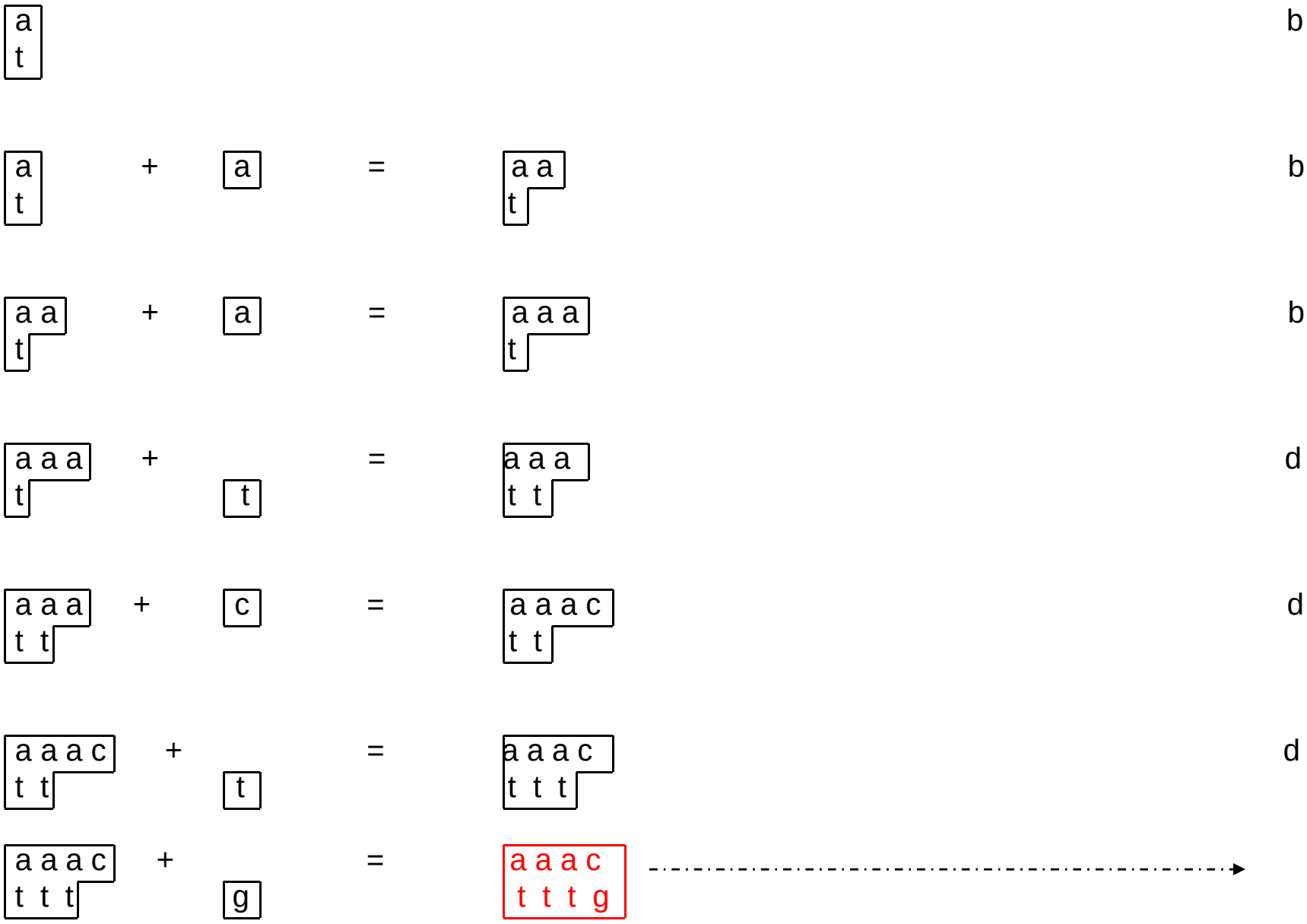

a
t
b
a
t
+
a
=
a a
t
b

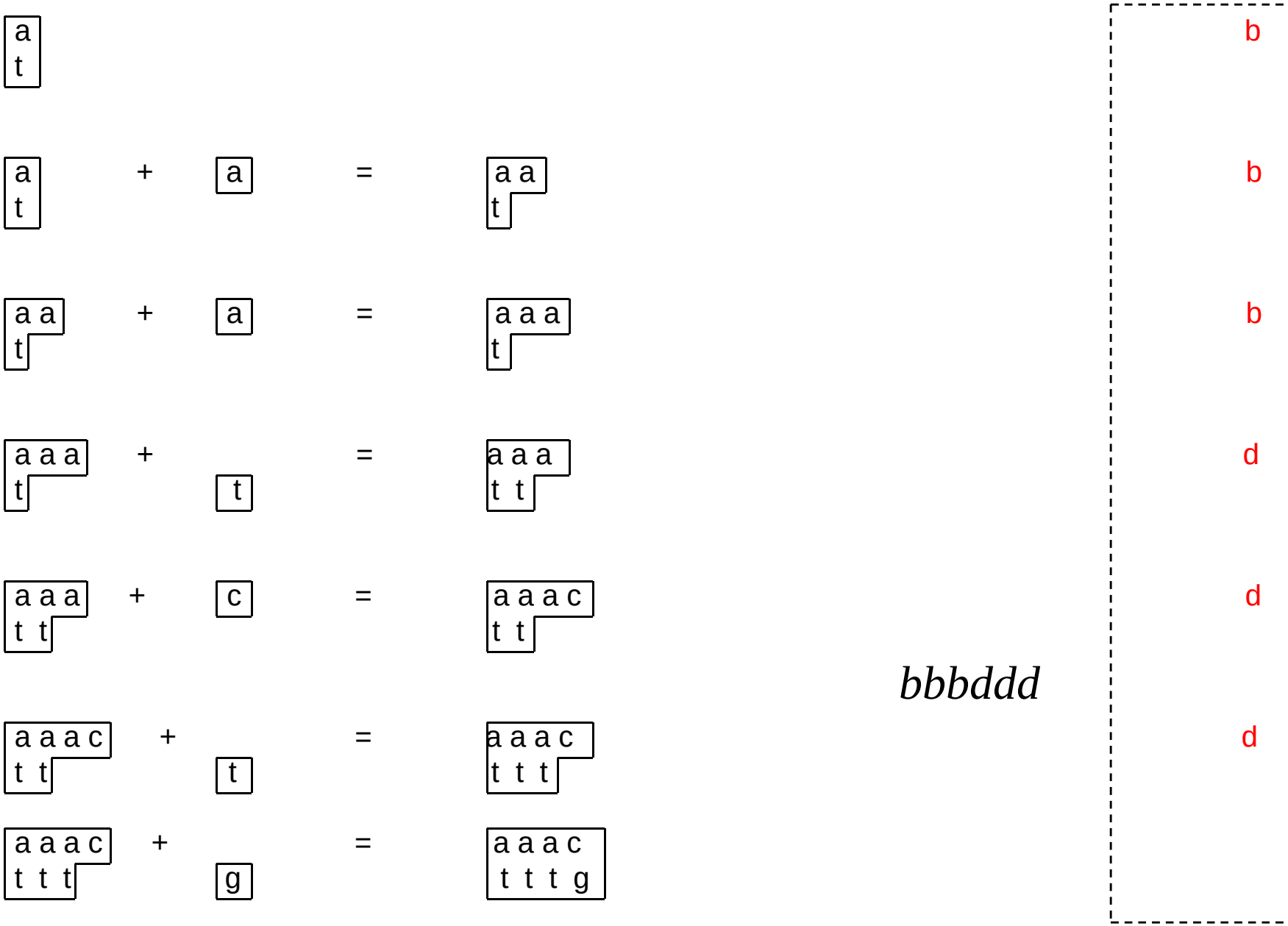

a
t
b
a
t
+
a
=
a a
t
b
a a
t
+
a
=
a a a
t
b

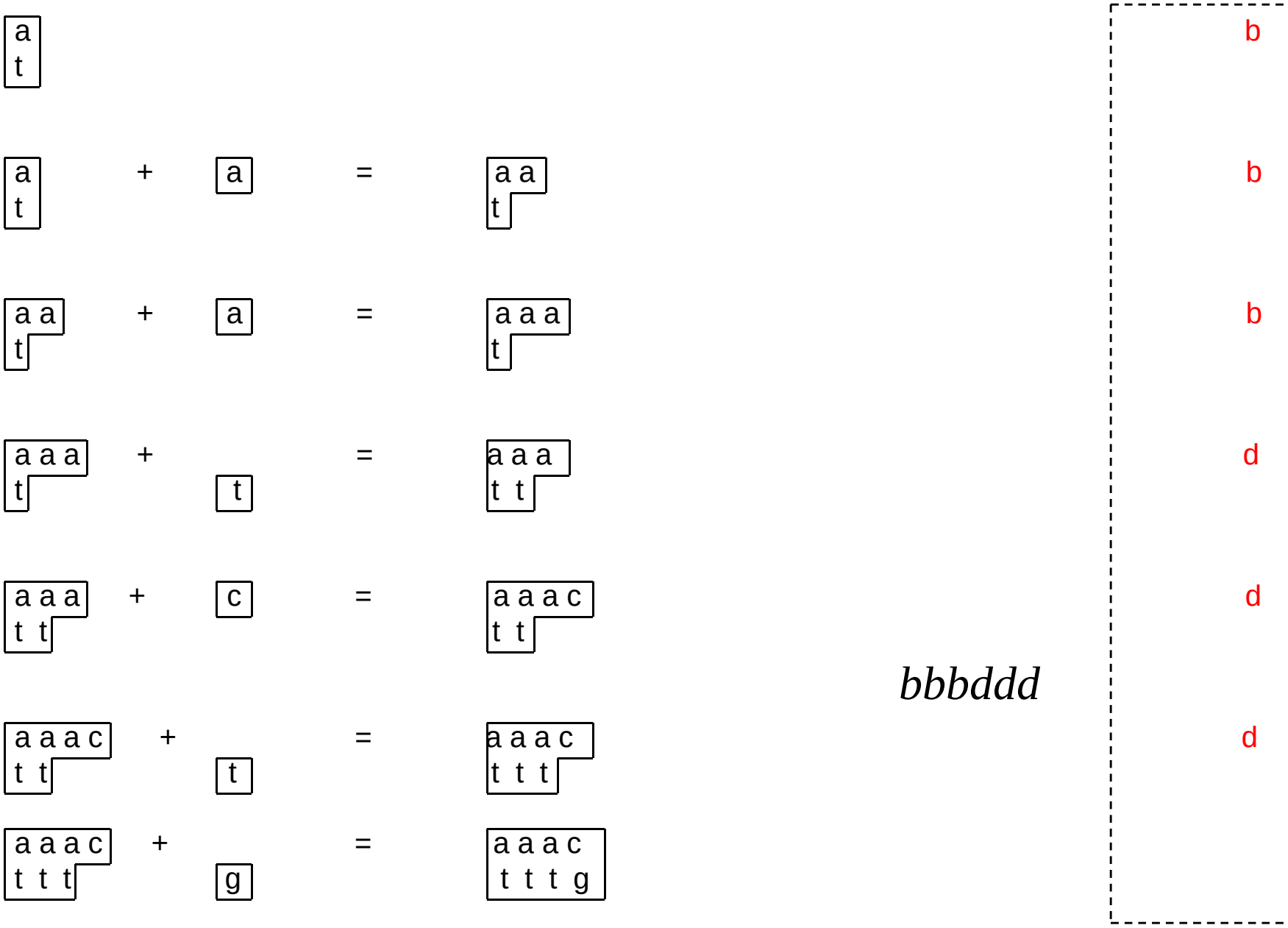

a
t
b
a
t
+
a
=
a a
t
b
a a
t
+
a
=
a a a
t
b
a a a
t
+
t
=
a a a
t t
d


a
t
b
a
t
+
a
=
a a
t
b
a a
t
+
a
=
a a a
t
b
a a a
t
+
t
=
a a a
t t
d
a a a
t t
+
c
=
a a a c
t t
d

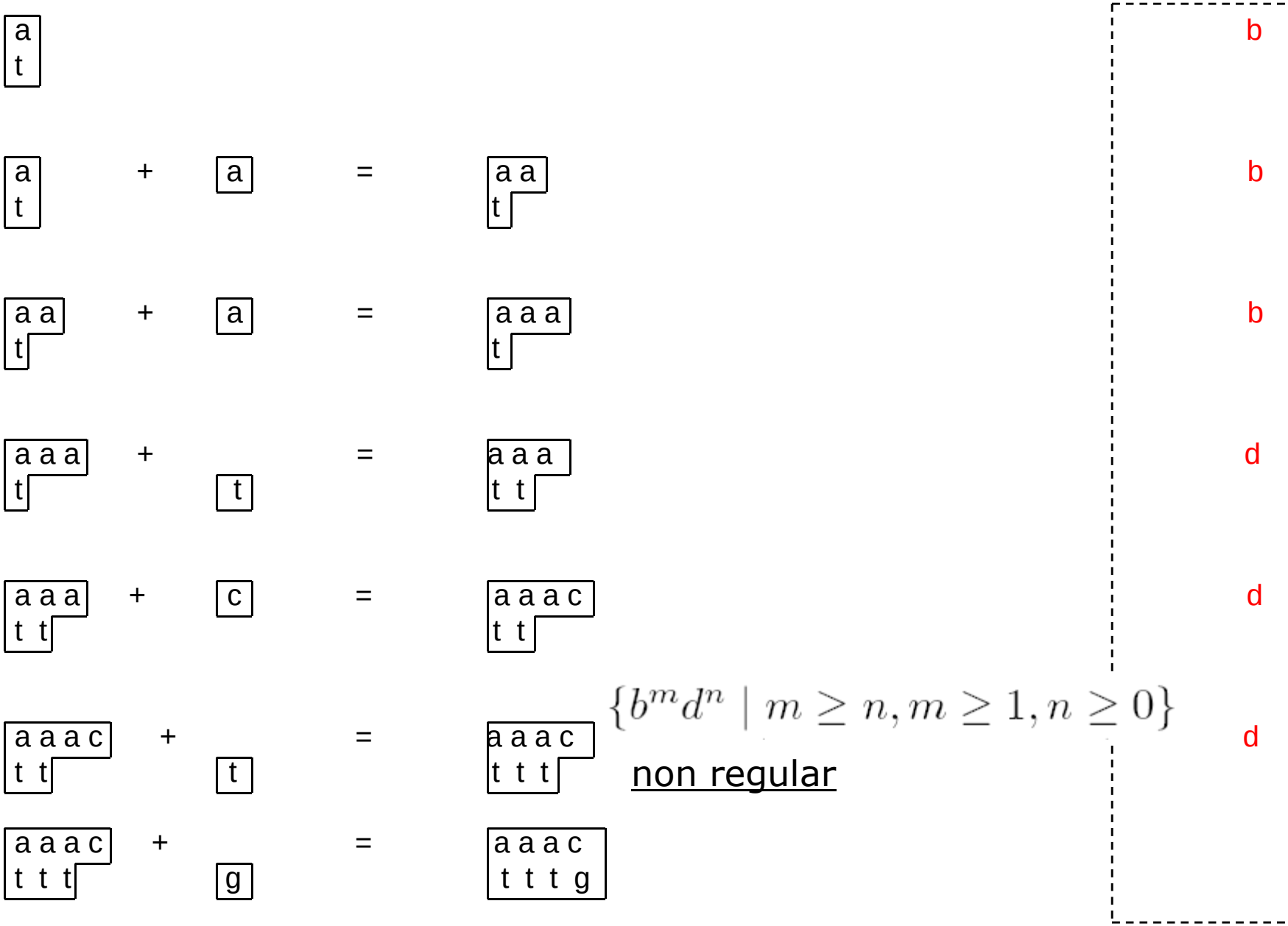

a
t
b
a
t
+
a
=
a a
t
b
a a
t
+
a
=
a a a
t
b
a a a
t
+
t
=
a a a
t t
d
a a a
t t
+
c
=
a a a c
t t
d
a a a c
t t
+
t
=
a a a c
t t t
d


a
t
b
a
t
+
a
=
a a
t
b
a a
t
+
a
=
a a a
t
b
a a a
t
+
t
=
a a a
t t
d
a a a
t t
+
c
=
a a a c
t t
d
a a a c
t t
+
t
=
a a a c
t t t
d
a a a c
t t t
+
g
=
a a a c
t t t g


a
t
a
t
+
a
=
a a
t
a a
t
+
a
=
a a a
t
a a a
t
+
t
=
a a a
t t
a a a
t t
+
c
=
a a a c
t t
a a a c
t t
+
t
=
a a a c
t t t
a a a c
t t t
+
g
=
a a a c
t t t g
bbbddd
b
b
b
d
d
d

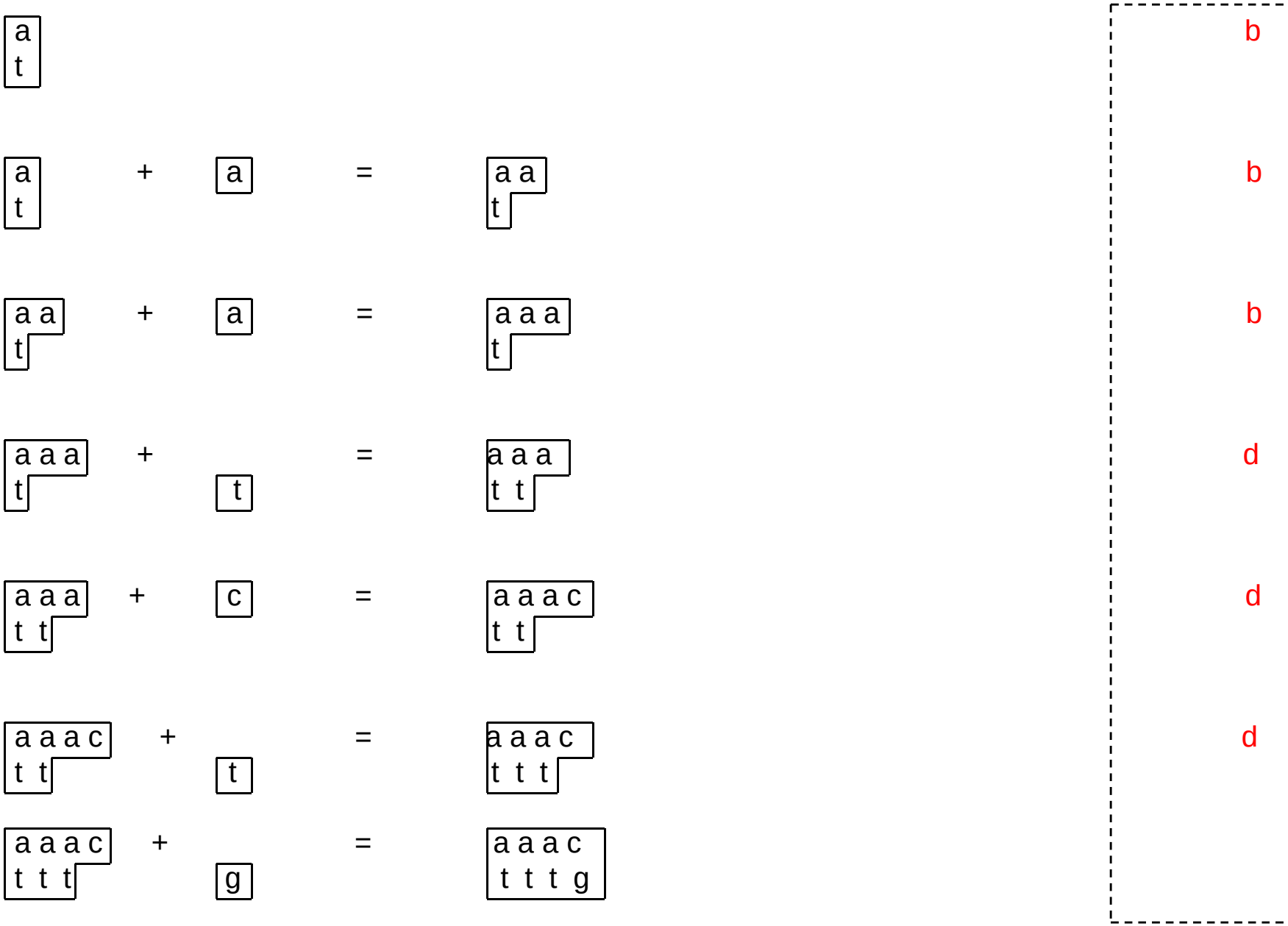


Considering all possible dynamics of the marked strand (non det.), one can get a language

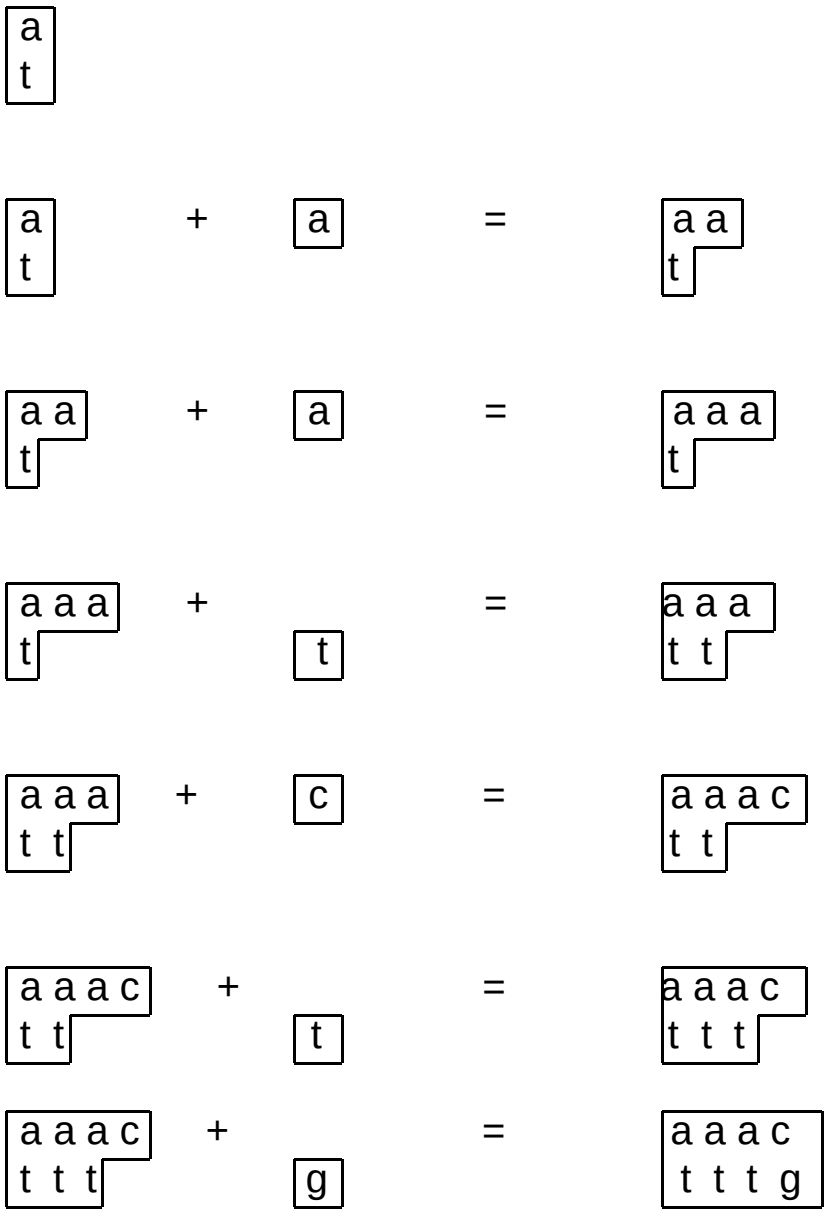


$$\{b^m d^n \mid m \geq n, m \geq 1, n \geq 0\}$$

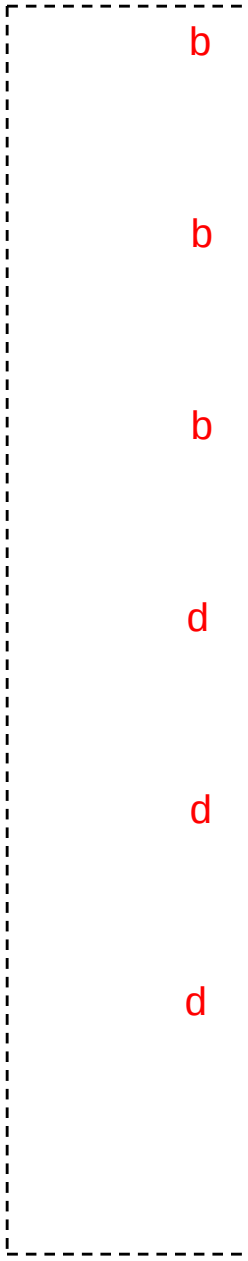
non regular



There exists a simple sticker system and an observer such that
the observed behaviour is <u>non recursive</u> *(there is no TM that can accept the language)*



Sticker systems used in the standard manner (input/output) are less than finite state automata



There exists a simple sticker system and an observer such that the observed behaviour is non recursive *(there is no TM that can accept the language)*

A General Observed System

A physical object is mapped to a symbol.

States of the observed system are represented by sequence of symbols (strings).

Rules that replace symbols with others are used to evolve the states.

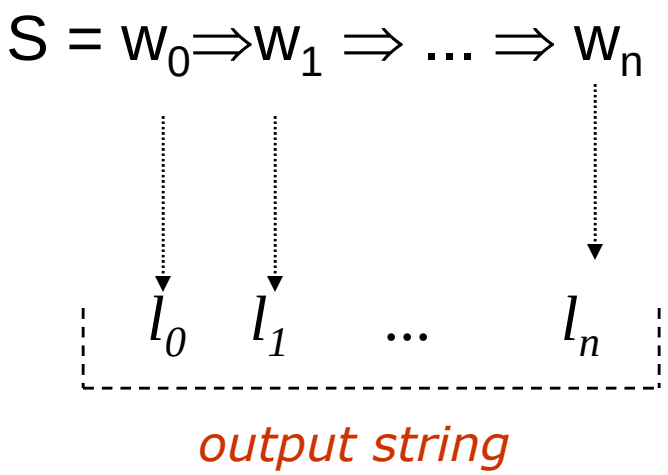
Observing Formal Grammars (a general approach)

The observed system is a formal grammar.

The observer is a finite state automaton.

Links with known classes of complexity, Chomsky hierarchy, algorithms, etc..

Observing Formal Grammars



grammar

+

observer

Observing Formal Grammars

G = (N, T, S, P)

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

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

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

Observing Formal Grammars

G = (N, T, S, P)

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

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

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

Context-free grammar. One symbol is replaced by a string

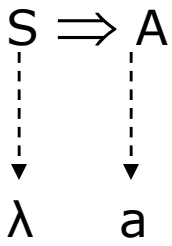
Observing Formal Grammars

G = (N, T, S, P)

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

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

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



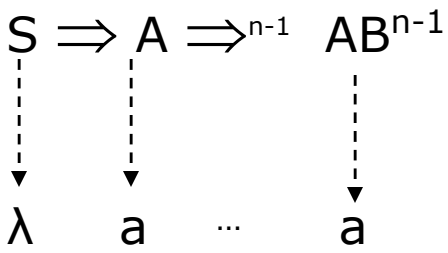
Observing Formal Grammars

G = (N, T, S, P)

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

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

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



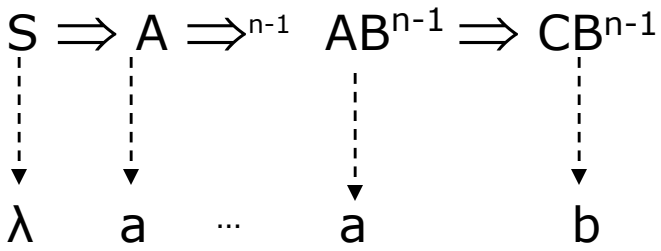
Observing Formal Grammars

G = (N, T, S, P)

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

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

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



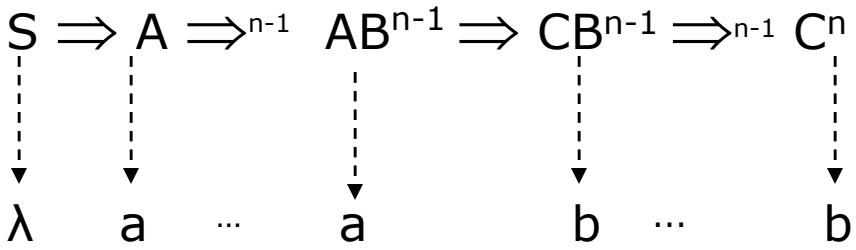
Observing Formal Grammars

G = (N, T, S, P)

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

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

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



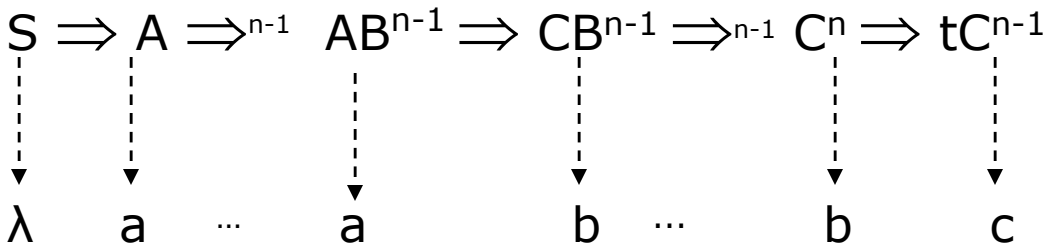
Observing Formal Grammars

G = (N, T, S, P)

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

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

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



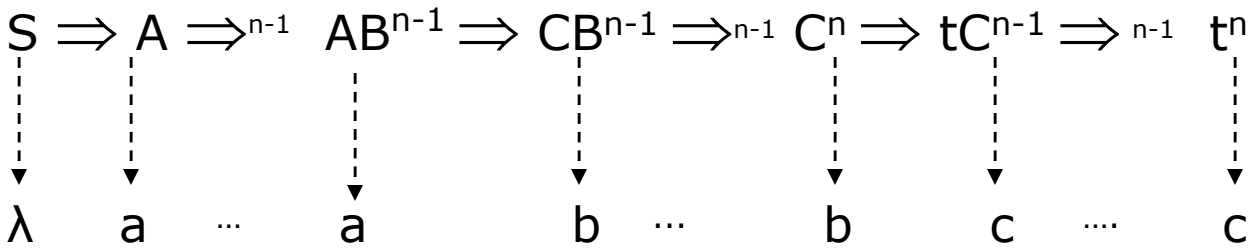
Observing Formal Grammars

G = (N, T, S, P)

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

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

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



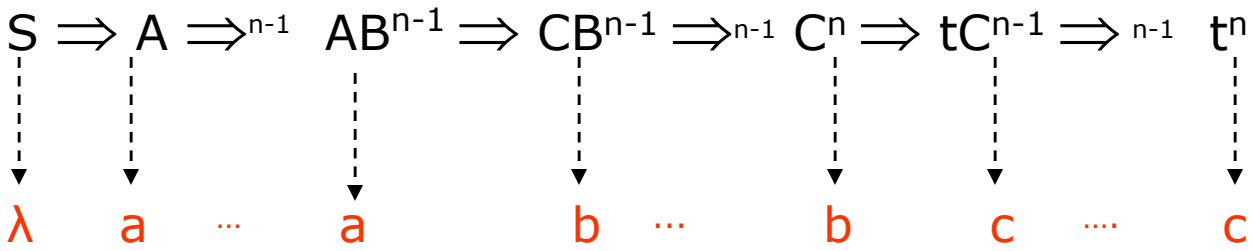
Observing Formal Grammars

G = (N, T, S, P)

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

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

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



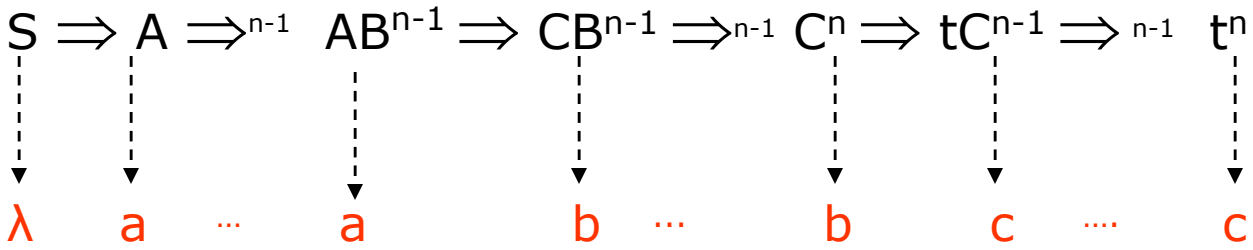
Observing Formal Grammars

G = (N, T, S, P)

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

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

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



observed behaviour of the grammar $\{a^n b^n c^n \mid n > 0\}$

non context-free language

Observing Formal Grammars

Always-writing observers must output a non-empty label at each step (each step of the observed process must be recorded)

Free observers can output an empty label (the observers can skip some of the steps of the observed process)

Observing Formal Grammars

Always-writing observers must output a non-empty label at each step (each step of the observed process must be recorded)

Free observers can output an empty label (the observers can skip some of the steps of the observed process)

Observed behaviours of:

context-free grammars

- correspond to recursively enumerable languages *with free observers.*
- less (or equal) to context-sensitive languages (but more than context-free languages) *with always -writing observers.*

Does the observer count?

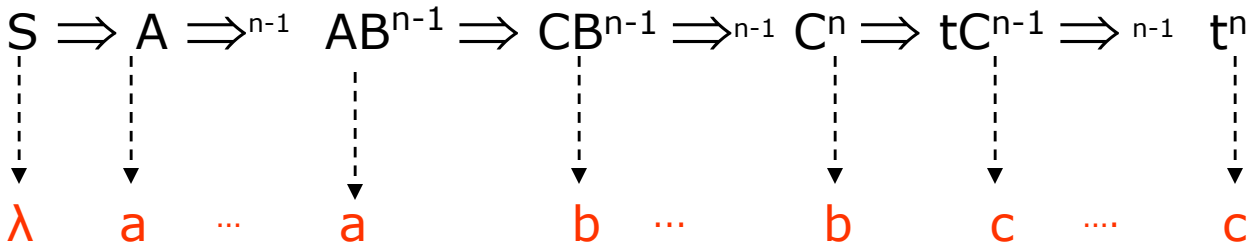
Does the observer count?

G = (N, T, S, P)

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

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

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



observed behaviour of the grammar $\{a^n b^n c^n \mid n > 0\}$

non context-free language

Does the observer count?

G = (N, T, S, P)

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

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

changing **only** the observer

Does the observer count?

G = (N, T, S, P)

N = {S, A, B, C} T = {t,p} O(w) = a if w ∈ {S, A, B, C, t, p}+

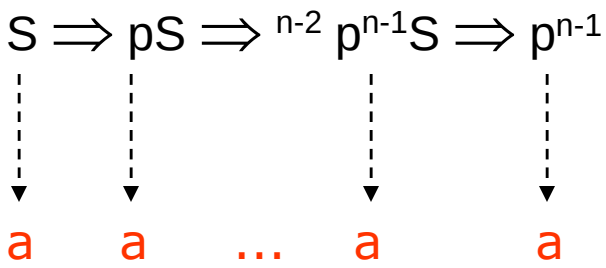
P= { $S \rightarrow pS, S \rightarrow p, S \rightarrow A,$
$A \rightarrow AB, A \rightarrow C, B \rightarrow C, C \rightarrow t$ }

Does the observer count?

G = (N, T, S, P)

N = {S, A, B, C} T = {t,p} O(w) = a if w ∈ {S, A, B, C, t, p}+

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



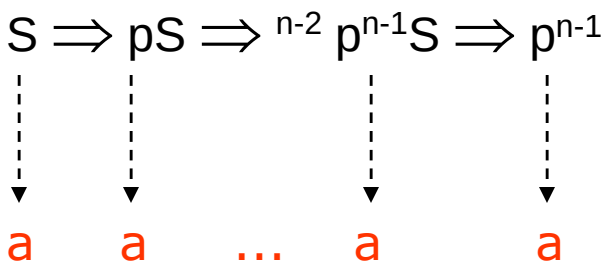
observed behaviour of the grammar $\{a^n \mid n > 0\}$
regular language

Does the observer count?

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

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

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



O'(w) produces a CS lang

O(w) produces a REG lang

observing the same grammar with the same dynamics

observed behaviour of the grammar $\{a^n \mid n > 0\}$
regular language

Computing by only observing

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

Computing by only observing

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

Everything, if the observed system is sufficiently “complex”!

Computing by only observing

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

Everything, if the observed system is sufficiently "complex"!

There is a given context-free grammar G such that:

Given an arbitrary recursively enumerable language L, one can find an observer O such that, the observed behaviors of G correspond exactly to L.

Computing by only observing

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

Everything, if the observed system is sufficiently "complex"!

There is a given context-free grammar G such that:

Given an arbitrary recursively enumerable language L, one can find an observer O such that, the observed behaviors of G correspond exactly to L.

If correctly observed, G can produce any possible behaviour.

Some comments

- Good news: Use of the result in natural computing.

We need *"only"* one fixed (functioning) biosystem, sufficiently complex. All computable functions can be implemented by finding the correct observer.

Some comments

- Good news: Use of the result in natural computing.

We need *"only"* one fixed (functioning) biosystem, sufficiently complex. All computable functions can be implemented by finding the correct observer.

- Bad news: The observer is crucial for understanding sufficiently complex systems.

Some comments

- Good news: Use of the result in natural computing.

We need *"only"* one fixed (functioning) biosystem, sufficiently complex. All computable functions can be implemented by finding the correct observer.

- Bad news: The observer is crucial for understanding sufficiently complex systems.

But what does it mean complex? Classical complexity does not work.

Some comments

- Good news: Use of the result in natural computing.

We need *"only"* one fixed (functioning) biosystem, sufficiently complex. All computable functions can be implemented by finding the correct observer.

- Bad news: The observer is crucial for understanding sufficiently complex systems.

But what does it mean complex? Classical complexity does not work.

Sticker systems (REG) + observers = non recursive languages.

Linear grammars (LIN>REG) + observers = regular languages.

Computing by Observing (to accept behaviours)

- “Computing by observing” can be used to implement accepting device.

Computing by Observing (to accept behaviours)

- "Computing by observing" can be used to implement accepting device.

- Observe the behaviour of a system. If it is the one expected accept it, otherwise reject it.

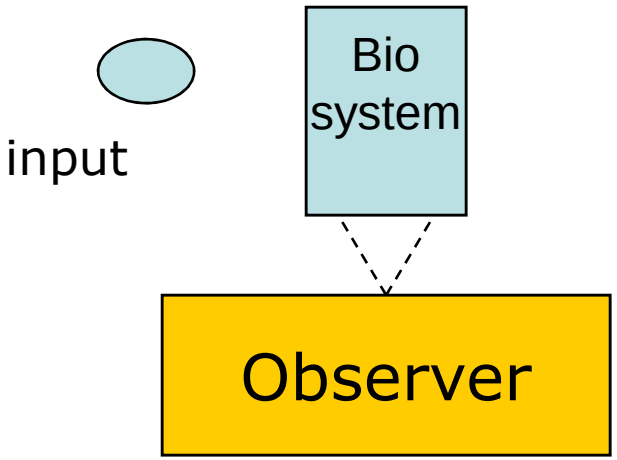
Computing by Observing (to accept behaviours)

- "Computing by observing" can be used to implement accepting device.

- Observe the behaviour of a system. If it is the one expected accept it, otherwise reject it.

- This is done all the time in experiments...

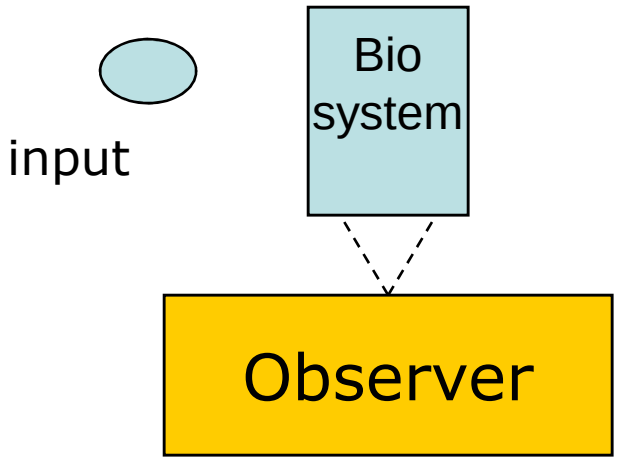
Consider the observed behaviour



Observed behaviour

R B R R B

Consider the observed behaviour

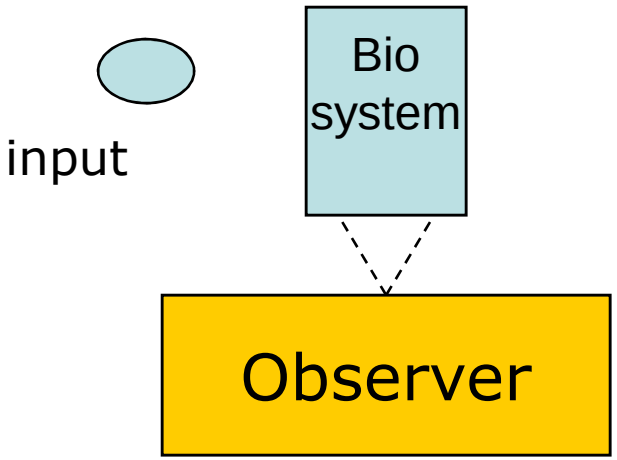


Observed behaviour

R B R R B

Is the observed behaviour the one *expected*?

Consider the observed behaviour



Observed behaviour

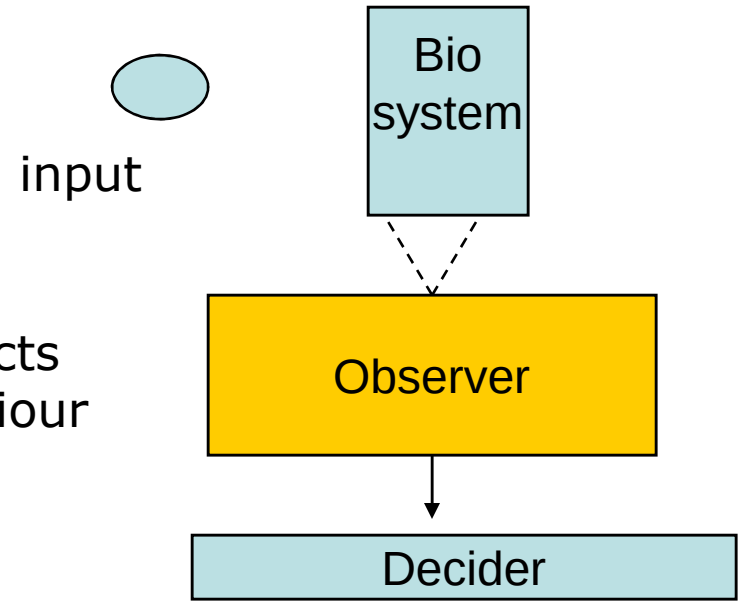
R B R R B

Is the observed behaviour the one *expected*?

Yes input accepted
No input rejected

General Architecture

Bio system receives and acts on an input

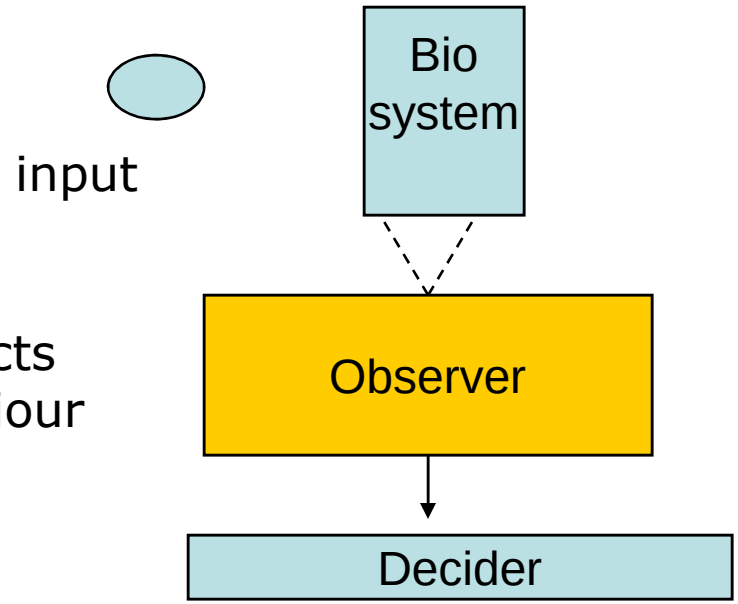


Observer extracts a formal behaviour

Decider checks if the observed behaviour is the one expected

General Architecture

Bio system receives and acts on an input



Observer extracts a formal behaviour

Decider checks if the observed behaviour is the one expected

Observer and decider are simple: FSA

Splicing Systems

Considering DNA recombination (splicing)

A splicing rule is of the form

$u_1 \# u_2 \$ u_3 \# u_4$ with u_1, u_2, u_3, u_4 strings

$x = x_1 u_1 u_2 x_2$

$y = y_1 u_3 u_4 y_2$

Splicing Systems

Considering DNA recombination (splicing)

A splicing rule is of the form

$u_1 \# u_2 \$ u_3 \# u_4$ with u_1, u_2, u_3, u_4 strings

$x = x_1\, u_1 \,|\, u_2\, x_2$

$y = y_1\, u_3 \,|\, u_4\, y_2$

Splicing Systems

Considering DNA recombination (splicing)

A splicing rule is of the form

$u_1 \# u_2 \$ u_3 \# u_4$ with u_1, u_2, u_3, u_4 strings

$x = x_1 u_1 | u_2 x_2$ $z = x_1 u_1 u_4 y_2$

$y = y_1 u_3 | u_4 y_2$

Splicing Systems

Considering DNA recombination (splicing)

A splicing rule is of the form

$u_1 \# u_2 \$ u_3 \# u_4$ with u_1, u_2, u_3, u_4 strings

$x = x_1 u_1 | u_2 x_2$ $z = x_1 u_1 u_4 y_2$

$y = y_1 u_3 | u_4 y_2$ $w = y_1 u_3 u_2 x_2$

Splicing Systems

Considering DNA recombination (splicing)

A splicing rule is of the form

$u_1 \# u_2 \$ u_3 \# u_4$ with u_1, u_2, u_3, u_4 strings

$x = x_1 u_1 | u_2 x_2$ $z = x_1 u_1 u_4 y_2$

$y = y_1 u_3 | u_4 y_2$ $w = y_1 u_3 u_2 x_2$

splicing systems can be seen as generative devices

axiom strings, splicing rules (iterative application).

Splicing Systems

Considering DNA recombination (splicing)

A splicing rule is of the form

$u_1 \# u_2 \$ u_3 \# u_4$ with u_1,u_2,u_3,u_4 strings

$x = x_1 u_1 | u_2 x_2$ $z = x_1 u_1 u_4 y_2$

$y = y_1 u_3 | u_4 y_2$ $w = y_1 u_3 u_2 x_2$

splicing systems can be seen as generative devices

axiom strings, splicing rules (iterative application).

Finite splicing can generate (at most) REG.

Observing a splicing system

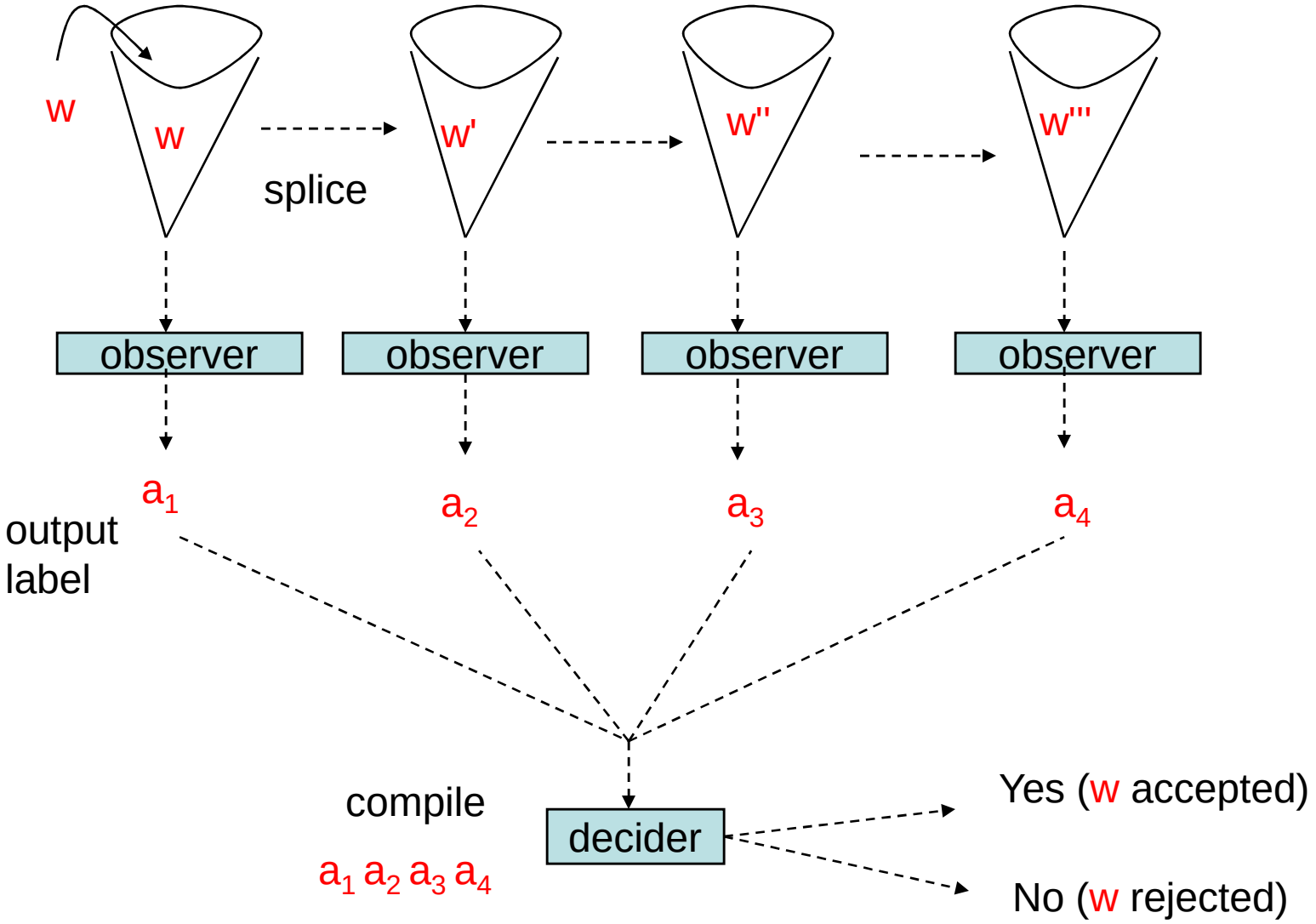
- Mark an input string.

- Decide if the observed behaviour of the marked string is the one expected.

Accept or reject the input string.

Splicing Recognizer

w marked DNA strand in input



Observing a splicing system...

- Mark a string.

- Decide if the observed behavior of the marked string is the one expected.

r_1 $\#bo_r \$ X_2 \# b' o_r$

r_2 $o_l a' \# Y_2 \$ o_l a \#$

r_3 $\#b' o_r \$ X_1 \# o_r$

r_4 $o_l \# Y_1 \$ o_l a' \#$

splicing rules

Observing a splicing system...

- Mark a string.

- Decide if the observed behavior of the marked string is the one expected.

r_1 $\#bo_r \$ X_2 \# b' o_r$

r_2 $o_l a' \# Y_2 \$ o_l a \#$

r_3 $\#b' o_r \$ X_1 \# o_r$

r_4 $o_l \# Y_1 \$ o_l a' \#$

splicing rules

$O(w) =$

l_0 if $w \in o_l (a^*b^*) o_r$

l_1 if $w \in o_l (a^*b^*b') o_r$

l_2 if $w \in o_l (a'a^*b^*b') o_r$

l_3 if $w \in o_l (a'a^*b^*) o_r$

‡ else

Observing a splicing system...

- Mark a string.

- Decide if the observed behavior of the marked string is the one expected.

r_1 $\#bo_r \$ X_2 \# b' o_r$

r_2 $o_l a' \# Y_2 \$ o_l a \#$

r_3 $\#b' o_r \$ X_1 \# o_r$

r_4 $o_l \# Y_1 \$ o_l a' \#$

splicing rules

$O(w) =$

l_0 if $w \in o_l (a^* b^*) o_r$

l_1 if $w \in o_l (a^* b^* b') o_r$

l_2 if $w \in o_l (a' a^* b^* b') o_r$

l_3 if $w \in o_l (a' a^* b^*) o_r$

‡ else

The decider accepts if only if the observed behaviour belongs to the regular language $l_0 (l_1 l_2 l_3 l_0)^*$

Step 0 Input marked string $w_0 = o_l\, aabb\, o_r$ $O(w_0) = l_0$

Step 0 Input marked string $w_0 = o_l\, aabb\, o_r$ $O(w_0) = l_0$

Step 1 apply splicing r_1 $w_1 = o_l\, aabb'\, o_r$ $O(w_1) = l_1$

Step 0 Input marked string $w_0 = o_l\, aabb\, o_r$ $O(w_0) = l_0$

Step 1 apply splicing r_1 $w_1 = o_l\, aabb'\, o_r$ $O(w_1) = l_1$

Step 2 apply splicing r_2 $w_2 = o_l\, a'abb'\, o_r$ $O(w_2) = l_2$

Step 0 Input marked string $w_0 = o_l\, aabb\, o_r$ $O(w_0) = l_0$

Step 1 apply splicing r_1 $w_1 = o_l\, aabb'\, o_r$ $O(w_1) = l_1$

Step 2 apply splicing r_2 $w_2 = o_l\, a'abb'\, o_r$ $O(w_2) = l_2$

Step 3 apply splicing r_3 $w_3 = o_l\, a'ab\, o_r$ $O(w_3) = l_3$

Step 0 Input marked string $w_0 = o_l\, aabb\, o_r$ $O(w_0) = l_0$

Step 1 apply splicing r_1 $w_1 = o_l\, aabb'\, o_r$ $O(w_1) = l_1$

Step 2 apply splicing r_2 $w_2 = o_l\, a'abb'\, o_r$ $O(w_2) = l_2$

Step 3 apply splicing r_3 $w_3 = o_l\, a'ab\, o_r$ $O(w_3) = l_3$

Step 4 apply splicing r_4 $w_4 = o_l\, ab\, o_r$ $O(w_4) = l_0$

Step 0 Input marked string $w_0 = o_l\, aabb\, o_r$ $O(w_0) = l_0$

Step 1 apply splicing r_1 $w_1 = o_l\, aabb'\, o_r$ $O(w_1) = l_1$

Step 2 apply splicing r_2 $w_2 = o_l\, a'abb'\, o_r$ $O(w_2) = l_2$

Step 3 apply splicing r_3 $w_3 = o_l\, a'ab\, o_r$ $O(w_3) = l_3$

Step 4 apply splicing r_4 $w_4 = o_l\, ab\, o_r$ $O(w_4) = l_0$

Step 5 apply splicing r_1 $w_5 = o_l\, ab'\, o_r$ $O(w_5) = l_1$

Step 0 Input marked string $w_0 = o_l\, aabb\, o_r$ $O(w_0) = l_0$

Step 1 apply splicing r_1 $w_1 = o_l\, aabb'\, o_r$ $O(w_1) = l_1$

Step 2 apply splicing r_2 $w_2 = o_l\, a'abb'\, o_r$ $O(w_2) = l_2$

Step 3 apply splicing r_3 $w_3 = o_l\, a'ab\, o_r$ $O(w_3) = l_3$

Step 4 apply splicing r_4 $w_4 = o_l\, ab\, o_r$ $O(w_4) = l_0$

Step 5 apply splicing r_1 $w_5 = o_l\, ab'\, o_r$ $O(w_5) = l_1$

Step 6 apply splicing r_2 $w_6 = o_l\, a'b'\, o_r$ $O(w_6) = l_2$

Step 0 Input marked string	$w_0 = o_l\, aabb\, o_r$	$O(w_0) = l_0$
Step 1 apply splicing r_1	$w_1 = o_l\, aabb'\, o_r$	$O(w_1) = l_1$
Step 2 apply splicing r_2	$w_2 = o_l\, a'abb'\, o_r$	$O(w_2) = l_2$
Step 3 apply splicing r_3	$w_3 = o_l\, a'ab\, o_r$	$O(w_3) = l_3$
Step 4 apply splicing r_4	$w_4 = o_l\, ab\, o_r$	$O(w_4) = l_0$
Step 5 apply splicing r_1	$w_5 = o_l\, ab'\, o_r$	$O(w_5) = l_1$
Step 6 apply splicing r_2	$w_6 = o_l\, a'b'\, o_r$	$O(w_6) = l_2$
Step 7 apply splicing r_3	$w_7 = o_l\, a'\, o_r$	$O(w_7) = l_3$

Step 0 Input marked string	$w_0 = o_l\, aabb\, o_r$	$O(w_0) = l_0$
Step 1 apply splicing r_1	$w_1 = o_l\, aabb'\, o_r$	$O(w_1) = l_1$
Step 2 apply splicing r_2	$w_2 = o_l\, a'abb'\, o_r$	$O(w_2) = l_2$
Step 3 apply splicing r_3	$w_3 = o_l\, a'ab\, o_r$	$O(w_3) = l_3$
Step 4 apply splicing r_4	$w_4 = o_l\, ab\, o_r$	$O(w_4) = l_0$
Step 5 apply splicing r_1	$w_5 = o_l\, ab'\, o_r$	$O(w_5) = l_1$
Step 6 apply splicing r_2	$w_6 = o_l\, a'b'\, o_r$	$O(w_6) = l_2$
Step 7 apply splicing r_3	$w_7 = o_l\, a'\, o_r$	$O(w_7) = l_3$
Step 8 apply splicing r_4	$w_8 = o_l\, o_r$	$O(w_8) = l_0$

Step 0 Input marked string	$w_0 = o_l\, aabb\, o_r$	$O(w_0) = l_0$
Step 1 apply splicing r_1	$w_1 = o_l\, aabb'\, o_r$	$O(w_1) = l_1$
Step 2 apply splicing r_2	$w_2 = o_l\, a'abb'\, o_r$	$O(w_2) = l_2$
Step 3 apply splicing r_3	$w_3 = o_l\, a'ab\, o_r$	$O(w_3) = l_3$
Step 4 apply splicing r_4	$w_4 = o_l\, ab\, o_r$	$O(w_4) = l_0$
Step 5 apply splicing r_1	$w_5 = o_l\, ab'\, o_r$	$O(w_5) = l_1$
Step 6 apply splicing r_2	$w_6 = o_l\, a'b'\, o_r$	$O(w_6) = l_2$
Step 7 apply splicing r_3	$w_7 = o_l\, a'\, o_r$	$O(w_7) = l_3$
Step 8 apply splicing r_4	$w_8 = o_l\, o_r$	$O(w_8) = l_0$

The observed evolution of the input marked w_0 string is

$l_0\, l_1\, l_2\, l_3\, l_0\, l_1\, l_2\, l_3\, l_0$

Step 0 Input marked string	$w_0 = o_l\, aabb\, o_r$	$O(w_0) = l_0$
Step 1 apply splicing r_1	$w_1 = o_l\, aabb'\, o_r$	$O(w_1) = l_1$
Step 2 apply splicing r_2	$w_2 = o_l\, a'abb'\, o_r$	$O(w_2) = l_2$
Step 3 apply splicing r_3	$w_3 = o_l\, a'ab\, o_r$	$O(w_3) = l_3$
Step 4 apply splicing r_4	$w_4 = o_l\, ab\, o_r$	$O(w_4) = l_0$
Step 5 apply splicing r_1	$w_5 = o_l\, ab'\, o_r$	$O(w_5) = l_1$
Step 6 apply splicing r_2	$w_6 = o_l\, a'b'\, o_r$	$O(w_6) = l_2$
Step 7 apply splicing r_3	$w_7 = o_l\, a'\, o_r$	$O(w_7) = l_3$
Step 8 apply splicing r_4	$w_8 = o_l\, o_r$	$O(w_8) = l_0$

The observed evolution of the input marked w_0 string is
$l_0\, l_1\, l_2\, l_3\, l_0\, l_1\, l_2\, l_3\, l_0$

The decider accepts if only if the observed behaviour belongs to the regular language $l_0\, (l_1\, l_2\, l_3\, l_0)^*$

Step 0 Input marked string	$w_0 = o_l\, aabb\, o_r$	$O(w_0) = l_0$
Step 1 apply splicing r_1	$w_1 = o_l\, aabb'\, o_r$	$O(w_1) = l_1$
Step 2 apply splicing r_2	$w_2 = o_l\, a'abb'\, o_r$	$O(w_2) = l_2$
Step 3 apply splicing r_3	$w_3 = o_l\, a'ab\, o_r$	$O(w_3) = l_3$
Step 4 apply splicing r_4	$w_4 = o_l\, ab\, o_r$	$O(w_4) = l_0$
Step 5 apply splicing r_1	$w_5 = o_l\, ab'\, o_r$	$O(w_5) = l_1$
Step 6 apply splicing r_2	$w_6 = o_l\, a'b'\, o_r$	$O(w_6) = l_2$
Step 7 apply splicing r_3	$w_7 = o_l\, a'\, o_r$	$O(w_7) = l_3$
Step 8 apply splicing r_4	$w_8 = o_l\, o_r$	$O(w_8) = l_0$

The observed evolution of the input marked w_0 string is
$l_0\, l_1\, l_2\, l_3\, l_0\, l_1\, l_2\, l_3\, l_0$ It is accepted by the decider D.

The decider accepts if only if the observed behaviour belongs to the regular language $l_0\, (l_1\, l_2\, l_3\, l_0)^*$

Step 0 Input marked string	$w_0 = o_l\, aabb\, o_r$	$O(w_0) = l_0$
Step 1 apply splicing r_1	$w_1 = o_l\, aabb'\, o_r$	$O(w_1) = l_1$
Step 2 apply splicing r_2	$w_2 = o_l\, a'abb'\, o_r$	$O(w_2) = l_2$
Step 3 apply splicing r_3	$w_3 = o_l\, a'ab\, o_r$	$O(w_3) = l_3$
Step 4 apply splicing r_4	$w_4 = o_l\, ab\, o_r$	$O(w_4) = l_0$
Step 5 apply splicing r_1	$w_5 = o_l\, ab'\, o_r$	$O(w_5) = l_1$
Step 6 apply splicing r_2	$w_6 = o_l\, a'b'\, o_r$	$O(w_6) = l_2$
Step 7 apply splicing r_3	$w_7 = o_l\, a'\, o_r$	$O(w_7) = l_3$
Step 8 apply splicing r_4	$w_8 = o_l\, o_r$	$O(w_8) = l_0$

The observed evolution of the input marked w_0 string is
$l_0\, l_1\, l_2\, l_3\, l_0\, l_1\, l_2\, l_3\, l_0$ It is accepted by the decider D.

So, the input marked string w_0 is accepted.

Step 0 Input marked string	$w_0 = o_l\, aabb\, o_r$	$O(w_0) = l_0$
Step 1 apply splicing r_1	$w_1 = o_l\, aabb'\, o_r$	$O(w_1) = l_1$
Step 2 apply splicing r_2	$w_2 = o_l\, a'abb'\, o_r$	$O(w_2) = l_2$
Step 3 apply splicing r_3	$w_3 = o_l\, a'ab\, o_r$	$O(w_3) = l_3$
Step 4 apply splicing r_4	$w_4 = o_l\, ab\, o_r$	$O(w_4) = l_0$
Step 5 apply splicing r_1	$w_5 = o_l\, ab'\, o_r$	$O(w_5) = l_1$
Step 6 apply splicing r_2	$w_6 = o_l\, a'b'\, o_r$	$O(w_6) = l_2$
Step 7 apply splicing r_3	$w_7 = o_l\, a'\, o_r$	$O(w_7) = l_3$
Step 8 apply splicing r_4	$w_8 = o_l\, o_r$	$O(w_8) = l_0$

The observed evolution of the input marked w_0 string is
$l_0\, l_1\, l_2\, l_3\, l_0\, l_1\, l_2\, l_3\, l_0$ It is accepted by the decider D.

So, the input marked string w_0 is accepted.

Collect all input strings that have a behaviour accepted by the decider D.

Step 0 Input marked string	$w_0 = o_l\, aabb\, o_r$	$O(w_0) = l_0$
Step 1 apply splicing r_1	$w_1 = o_l\, aabb'\, o_r$	$O(w_1) = l_1$
Step 2 apply splicing r_2	$w_2 = o_l\, a'abb'\, o_r$	$O(w_2) = l_2$
Step 3 apply splicing r_3	$w_3 = o_l\, a'ab\, o_r$	$O(w_3) = l_3$
Step 4 apply splicing r_4	$w_4 = o_l\, ab\, o_r$	$O(w_4) = l_0$
Step 5 apply splicing r_1	$w_5 = o_l\, ab'\, o_r$	$O(w_5) = l_1$
Step 6 apply splicing r_2	$w_6 = o_l\, a'b'\, o_r$	$O(w_6) = l_2$
Step 7 apply splicing r_3	$w_7 = o_l\, a'\, o_r$	$O(w_7) = l_3$
Step 8 apply splicing r_4	$w_8 = o_l\, o_r$	$O(w_8) = l_0$

The observed evolution of the input marked w_0 string is
$l_0\, l_1\, l_2\, l_3\, l_0\, l_1\, l_2\, l_3\, l_0$ It is accepted by the decider D.

So, the input marked string w_0 is accepted.

Collect all input strings that have a behaviour accepted by the decider D.

$L(\Omega) = \{o_l\, a^n b^n\, o_r \mid n \geq 0\}$ non- regular language!

Step 0 Input marked string	$w_0 = o_l\, aabb\, o_r$	$O(w_0) = l_0$
Step 1 apply splicing r_1	$w_1 = o_l\, aabb'\, o_r$	$O(w_1) = l_1$
Step 2 apply splicing r_2	$w_2 = o_l\, a'abb'\, o_r$	$O(w_2) = l_2$
Step 3 apply splicing r_3	$w_3 = o_l\, a'ab\, o_r$	$O(w_3) = l_3$
Step 4 apply splicing r_4	$w_4 = o_l\, ab\, o_r$	$O(w_4) = l_0$
Step 5 apply splicing r_1	$w_5 = o_l\, ab'\, o_r$	$O(w_5) = l_1$
Step 6 apply splicing r_2	$w_6 = o_l\, a'b'\, o_r$	$O(w_6) = l_2$
Step 7 apply splicing r_3	$w_7 = o_l\, a'\, o_r$	$O(w_7) = l_3$
Step 8 apply splicing r_4	$w_8 = o_l\, o_r$	$O(w_8) = l_0$

The observed evolution of the input marked w_0 string is
$l_0\, l_1\, l_2\, l_3\, l_0\, l_1\, l_2\, l_3\, l_0$ It is accepted by the decider D.

So, the input marked string w_0 is accepted.

Collect all input strings that have a behaviour accepted by the decider D.

$L(\Omega) = \{o_l\, a^n b^n\, o_r \mid n \geq 0\}$ non- regular language!

Splicing systems used in this way are computationally complete.

Restricted Observed Systems

- *rules of the system* $a \rightarrow b$, $b \rightarrow d$, $b \rightarrow b$, *etc..*

$abbabb \Rightarrow bbbbbb \Rightarrow bdbbdd \Rightarrow$ ....

input

Restricted Observed Systems

- *rules of the system* $a \rightarrow b$, $b \rightarrow d$, $b \rightarrow b$, *etc..*

$abbabb \Rightarrow bbbbbb \Rightarrow bdbbdd \Rightarrow$ ....

These systems (PA) can generate/accept only FIN.

Restricted Observed Systems

- *rules of the system* $a \rightarrow b$, $b \rightarrow d$, $b \rightarrow b$, *etc..*

$abbabb \Rightarrow bbbbbb \Rightarrow bdbbdd \Rightarrow$ ....

These systems (PA) can generate/accept only FIN.

Consider Ω = PA + FSA (observer)

Restricted Observed Systems

- *rules of the system* $a \rightarrow b$, $b \rightarrow d$, $b \rightarrow b$, *etc..*

$abbabb \Rightarrow bbbbbb \Rightarrow bdbbdd \Rightarrow$ ....

These systems (PA) can generate/accept only FIN.

L is in NSPACE-TIME(f,g) if there is a TM that accepts L using f(n) tape-cells in g(n) steps for a word of length n.

Consider Ω = PA + FSA (observer)

Restricted Observed Systems

- *rules of the system* $a \rightarrow b$, $b \rightarrow d$, $b \rightarrow b$, *etc..*

$abbabb \Rightarrow bbbbbb \Rightarrow bdbbdd \Rightarrow$ ....

These systems (PA) can generate/accept only FIN.

L is in NSPACE-TIME(f,g) if there is a TM that accepts L using f(n) tape-cells in g(n) steps for a word of length n.

Consider Ω = PA + FSA (observer)

Any language generated by Ω is in the class *NSPACE-TIME(id,*$\lambda n.n \cdot c^n$*), c size alphabet* .

Restricted Observed Systems

- *rules of the system* $a \rightarrow b$, $b \rightarrow d$, $b \rightarrow b$, *etc..*

$abbabb \Rightarrow bbbbbb \Rightarrow bdbbdd \Rightarrow$ ....

These systems (PA) can generate/accept only FIN.

L is in NSPACE-TIME(f,g) if there is a TM that accepts L using f(n) tape-cells in g(n) steps for a word of length n.

Consider Ω = PA + FSA (observer)

Any language generated by Ω is in the class *NSPACE-TIME(id,*$\lambda n.n \cdot c^n$*), c size alphabet* .

Any language in NSPACE (id) is generated by Ω

Restricted Observed Systems

- *rules of the system* $a \rightarrow b$, $b \rightarrow d$, $b \rightarrow b$, *etc..*

$abbabb \Rightarrow bbbbbb \Rightarrow bdbbdd \Rightarrow$ ....

These systems (PA) can generate/accept only FIN.

PA + FSA = NSPACE (id) = CS

Questions

- Making the observer more realistic (small number of states, subregular, probabilistic,..)
- Restrict the window of observation (hierarchy on the size of the window?)
- Implementation (software/hardware/lab)?
- Links with other related topics: process algebra and observability, observabilty in control theory.
- Systems with observer-independent behaviours.

“The result of a computation can be seen as already present in Nature: we only need to look (in an appropriate way) at it.”

G. Rozenberg, A. Salomaa,
Tech. Report, 96-28, Dept. of Computer Science,
Leiden University, 1996.

“The result of a computation can be seen as already present in Nature: we only need to look (in an appropriate way) at it.”

G. Rozenberg, A. Salomaa,
Tech. Report, 96-28, Dept. of Computer Science,
Leiden University, 1996.

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

M. Proust

References

M. Cavaliere, P. Leupold: Evolution and observation-a non-standard way to generate formal languages. Theoretical Computer Science, 2-3, 2004.

M. Cavaliere, P. Leupold: Observation of String-Rewriting Systems. Fundam. Inform. 4, 2006.

M. Cavaliere, N. Jonoska, P. Leupold, DNA Splicing: Computing by Observing, Natural Computing, 2007.

A. Alhazov, M. Cavaliere: Computing by Observing Bio-systems: The Case of Sticker Systems. DNA 2004.

M. Cavaliere, P. Leupold: Evolution and Observation: A New Way to Look at Membrane Systems. WMC 2003.

M. Cavaliere, P. Frisco, H.J. Hoogeboom: Computing by Only Observing. Developments in Language Theory 2006.

Thank you for your attention !