Computing by Observing Insertion

Alexander Krassovitskiy and Peter Leupold

Universitat Rovira i Virgili
Tarragona, Spain

LATA 2012

Levenshtein Distance

Insertion and deletion are operations of fundamental interest in information theory.

The Levenshtein Distance or Edit Distance measures the minimum number of

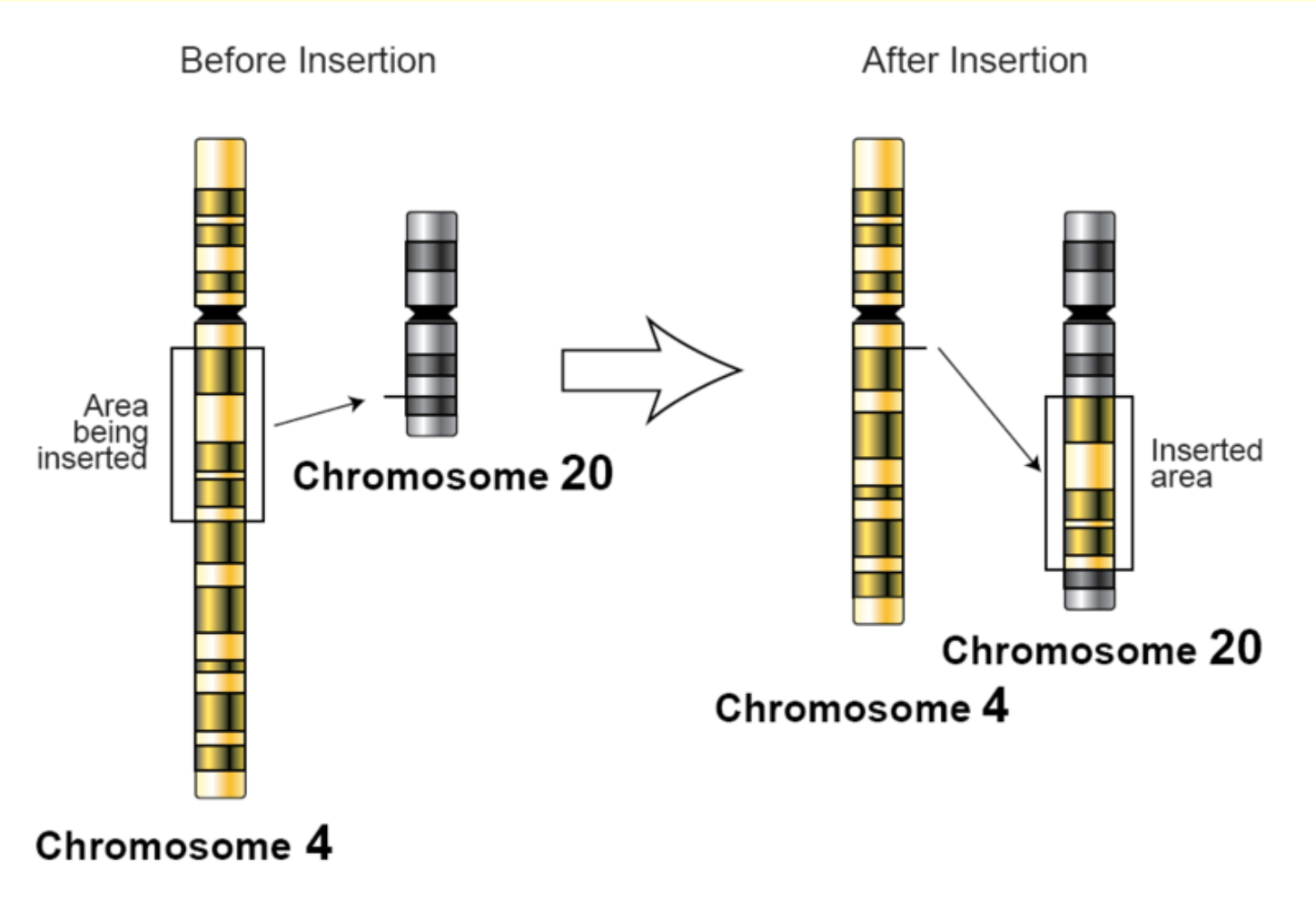
- insertions,
- deletions, and
- substitutions

that are necessary to convert one string into another.

This is motivated from signal transmission, where signals can appear through noise, can be lost, or can be altered.

DNA Computing

Insertion and deletion as operations on strings have received increased interest in the field of DNA Computing.



Picture from wikipedia.org

Insertion Systems

Definition

An insertion system is a triple (Σ, A, R), where

Σ is an alphabet,

the set of axioms A is a finite language over Σ, and

the set of insertion rules R is a finite set of triples of the form (u, α, v),

where u, α, and v are strings over Σ, $\alpha \neq \lambda$.

Insertion Rules

The three components of insertion rules are

- left context
- inserted string
- right context

A rule $[a, bb, ab]$ does the following:

`cabaababc`

Insertion Rules

The three components of insertion rules are

- left context
- inserted string
- right context

A rule $[a, bb, ab]$ does the following:

$$\texttt{caba}\cdot\texttt{ababc}$$

Insertion Rules

The three components of insertion rules are

- left context
- inserted string
- right context

A rule $[a, bb, ab]$ does the following:

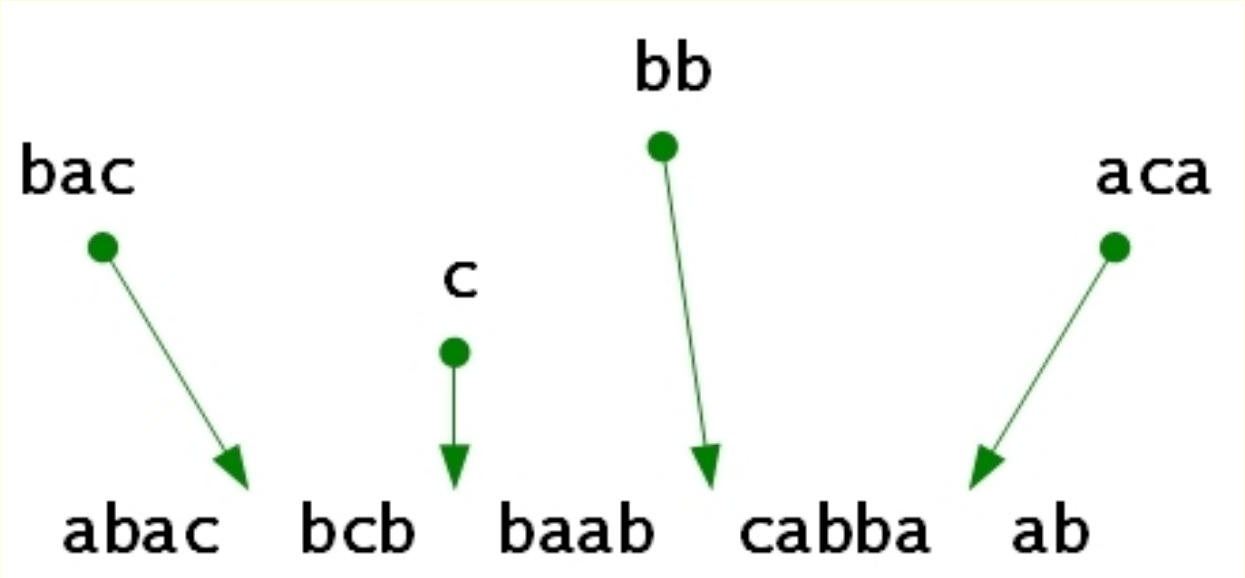
$$\texttt{caba}\texttt{bb}\texttt{ababc}$$

Insertion Systems

Every insertion makes the string longer.

So we do not have much computation time for reaching a given string,

and neither do we have much control over the place of insertions.



Insertion Systems

Pure insertion systems have a very limited computational power:

Theorem

Every language over the alphabet Σ generated by an insertion system

- *with insertion rules that have empty contexts and*
- *insert only one letter*

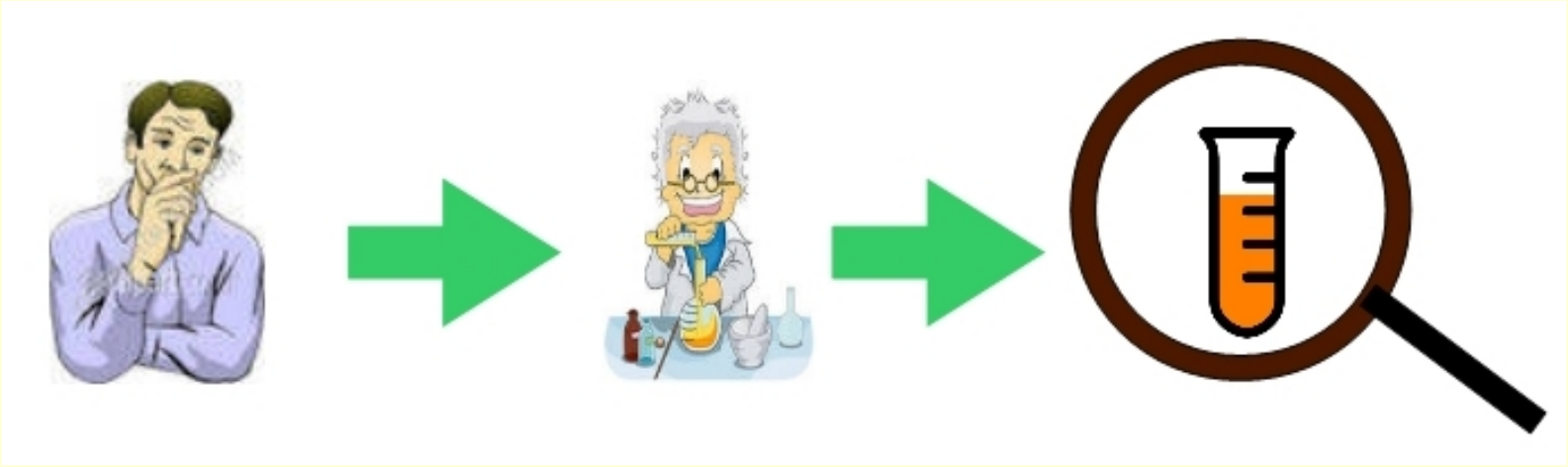
is a finite union of languages of the form

$$X^* x_1 X^* x_2 X^* \cdots X^* x_n X^*$$

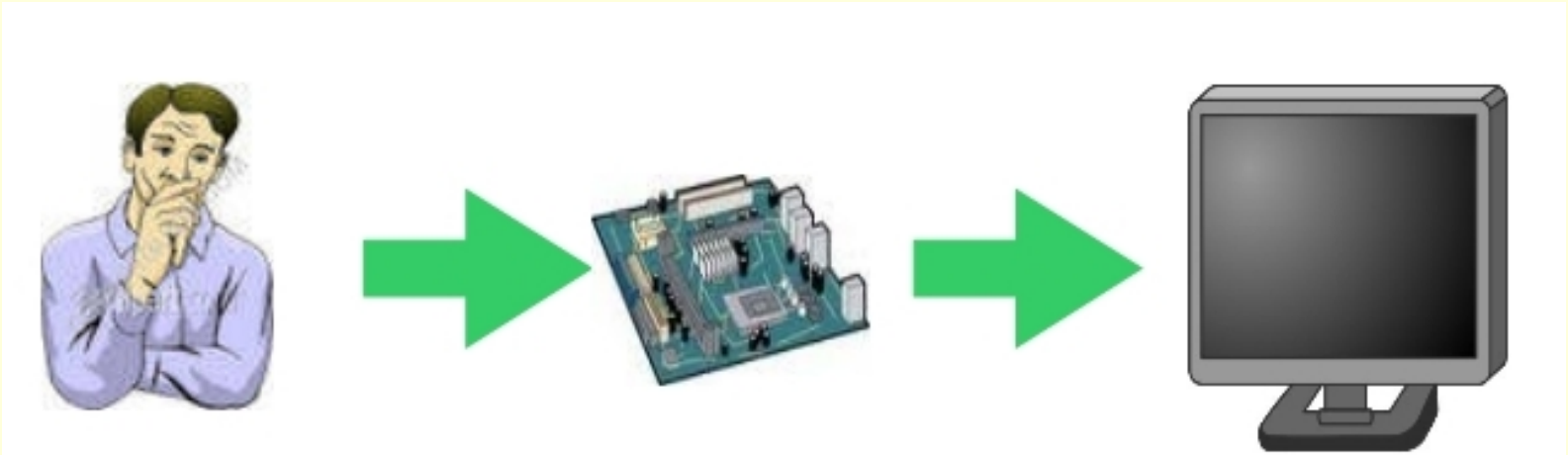
where for some $n \geq 0$ we have $x_1, x_2, \ldots, x_n \in \Sigma$ and $X \subseteq \Sigma$.

Standard DNA Computing

So insertion systems follow the standard approach in DNA Computing:



which is very similar to the standard computer science approach:



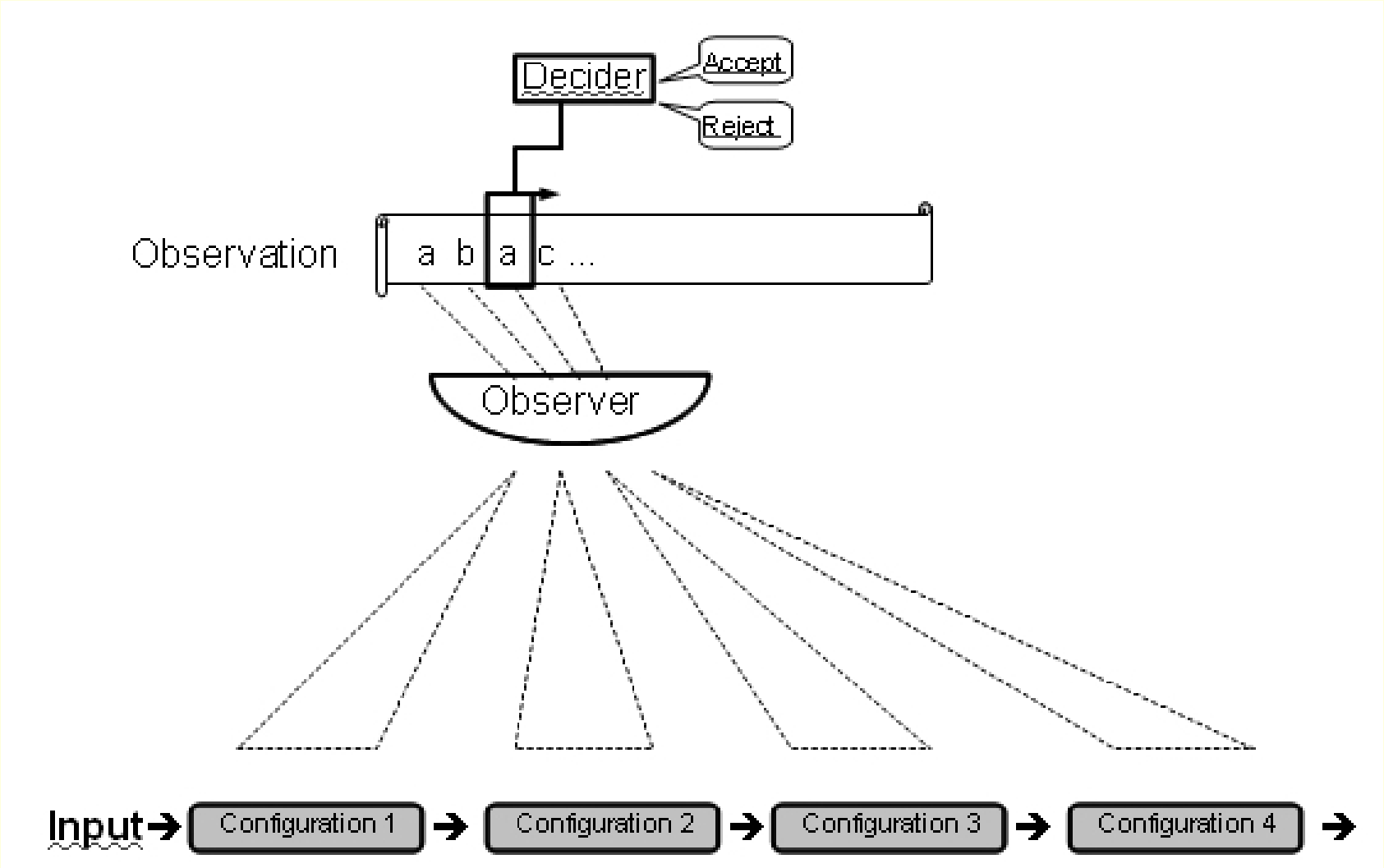
Computing by Observing

The Computing by Observing architecture is an alternative for DNA computing models that

- does not let the DNA itself produce the output
- but includes an observer that
  - sees the systems configuration in every step and
  - can output one letter per step.

Thus the observer resembles for example a chemist monitoring certain properties (temperature, pH, ...) during an experiment.

Computing by Observing



Insertion Observer Systems

Definition

An insertion observer system is a quintuple $\Omega = [\Delta, R, \mathcal{O}, D, \odot]$, where

- $\mathcal{O}$ is a monadic transducer,
- R is an insertion system over the input alphabet Σ of $\mathcal{O}$,
- Δ the system's input alphabet, a subset of Σ, and
- D is a regular language over the output alphabet of $\mathcal{O}$;

all words of D end in the termination symbol $\odot$.

Observers

The observer should

- read the configurations (strings),
- output a symbol and
- be simple.

So we use Monadic Transducers, which are basically finite deterministic automata with labels on their states.

Insertion Observer Systems – Example

1. Check, if the input word is from the language $a^+b^+c^+$.

2. Insert an A after the first letter a that is not followed by A.
3. Insert a B after the first letter b that is not followed by B.
4. Insert a C after the first letter c that is not followed by C.

5. Insert an $\overline{A}$ after the first letter A that is not followed by $\overline{A}$.
6. Insert a $\overline{B}$ after the first letter B that is not followed by $\overline{B}$.
7. Insert a $\overline{C}$ after the first letter C that is not followed by $\overline{C}$.

8. If there is some lower case letter left that is not followed by the corresponding upper case letter: go back to Step (2).
9. Accept the input word.

Insertion Observer Systems – Example

The algorithm accepts the language $\{a^n b^n c^n : n \geq 0\}$ and is implemented with the insertion rules

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

aabbcc

Observation: 1

Insertion Observer Systems – Example

The algorithm accepts the language $\{a^n b^n c^n : n \geq 0\}$ and is implemented with the insertion rules

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

aAabbcc

Observation: 2

Insertion Observer Systems – Example

The algorithm accepts the language $\{a^n b^n c^n : n \geq 0\}$ and is implemented with the insertion rules

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

aAabBbcc

Observation: 3

Insertion Observer Systems – Example

The algorithm accepts the language $\{a^n b^n c^n : n \geq 0\}$ and is implemented with the insertion rules

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

aAabBbcCc

Observation: 4

Insertion Observer Systems – Example

The algorithm accepts the language $\{a^n b^n c^n : n \geq 0\}$ and is implemented with the insertion rules

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

aA$\overline{\text{A}}$abBbcCc

Observation: 5

Insertion Observer Systems – Example

The algorithm accepts the language $\{a^n b^n c^n : n \geq 0\}$ and is implemented with the insertion rules

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

$$\texttt{aA}\overline{\texttt{A}}\texttt{abB}\overline{\texttt{B}}\texttt{bcCc}$$

Observation: 6

Insertion Observer Systems – Example

The algorithm accepts the language $\{a^n b^n c^n : n \geq 0\}$ and is implemented with the insertion rules

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

$$\texttt{aA}\overline{\texttt{A}}\texttt{abB}\overline{\texttt{B}}\texttt{bcC}\overline{\texttt{C}}\texttt{c}$$

Observation: 1

Insertion Observer Systems – Example

The algorithm accepts the language $\{a^n b^n c^n : n \geq 0\}$ and is implemented with the insertion rules

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

aA$\overline{\text{A}}$aAbB$\overline{\text{B}}$bBcC$\overline{\text{C}}$cC

Observation: 234

Insertion Observer Systems – Example

The algorithm accepts the language $\{a^n b^n c^n : n \geq 0\}$ and is implemented with the insertion rules

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

$$\mathtt{aA\overline{A}aA\overline{A}bB\overline{B}bB\overline{B}cC\overline{C}cC\overline{C}}$$

Observation: $56\odot$

Insertion Observer Systems – Example

The observer mapping is:

$$\mathcal{O}(w) = \begin{cases} 1 & \text{if } w \in (aA\overline{A})^* a^+ (bB\overline{B})^* b^+ (cC\overline{C})^* c^+, \\ 2 & \text{if } w \in (aA\overline{A})^* aAa^* (bB\overline{B})^* b^+ (cC\overline{C})^* c^+, \\ 3 & \text{if } w \in (aA\overline{A})^* aAa^* (bB\overline{B})^* bBb^* (cC\overline{C})^* c^+, \\ 4 & \text{if } w \in (aA\overline{A})^* aAa^* (bB\overline{B})^* bBb^* (cC\overline{C})^* cCc^*, \\ 5 & \text{if } w \in (aA\overline{A})^+ a^* (bB\overline{B})^* bBb^* (cC\overline{C})^* cCc^*, \\ 6 & \text{if } w \in (aA\overline{A})^+ a^* (bB\overline{B})^+ b^* (cC\overline{C})^* cCc^*, \\ \odot & \text{if } w \in (aA\overline{A})^+ (bB\overline{B})^+ (cC\overline{C})^+, \\ \bot & \text{else.} \end{cases}$$

The decider accepts the language $(123456)^+\odot$.

Insertion Observer Systems

The key techniques used in this example are:

- Using complementation instead of deletion.
- Use phases of marking for coordination.

In this way, we can simulate rewritability:

Instead of $a \to a' \to a''$ we use $a \to aA \to aA\overline{A}$.

And we can simulate context-sensitivity:

Instead of $ab \to AB$ we use $ab \to aAb \to aAbB \to a\bar{a}AbB \to a\bar{a}Ab\bar{b}B$.

Insertion Observer Systems

Using the techniques used in the example, we can actually simulate a Turing Machine.

Theorem

Every recursively enumerable language can be accepted by an insertion observer system

- *with insertion rules that have empty contexts and*
- *insert only one letter.*

Insertion Observer Systems

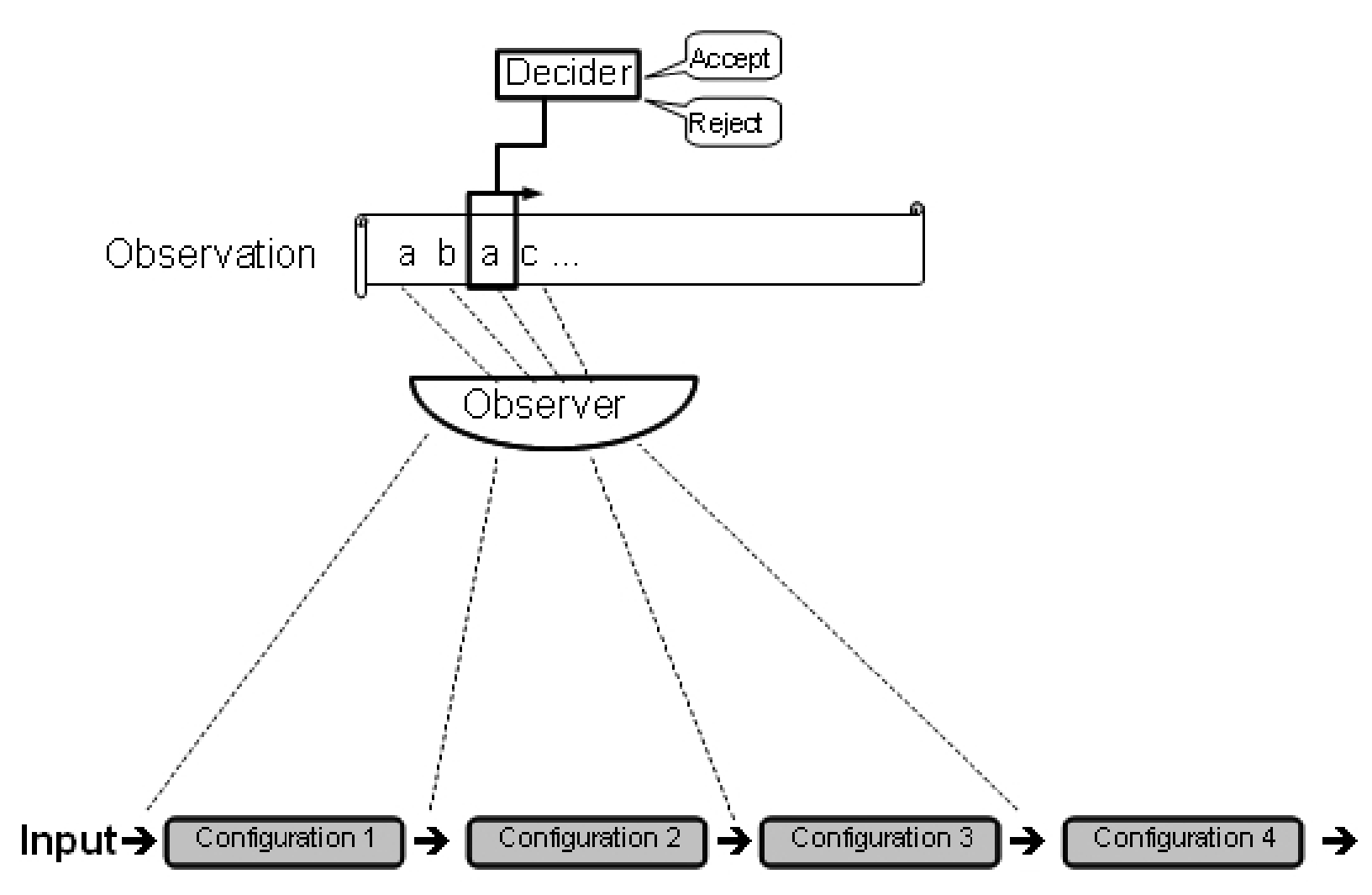
So with sub-regular insertion systems and regular observers we achieve computational completeness.

NICE: our architecture provides us with power far beyond the components' power.

NOT SO NICE: we obtain completeness very quickly.

Therefore we consider variants with less powerful modes of observation.

Observing Change



Observing Change – Example

We can accept $\{a^n b^n c^n : n \geq 0\}$ as the following three conditions characterize exactly the words from this language:

1. There is exactly one occurrence of each ab and bc.
2. The numbers of aa, bb and cc are equal.
3. Other blocks of two letters (ac, ba, ca, cb) do not occur.

We use insertion rules $(x, \bigwedge, y)$ for all pairs of letters x and y.

Observing Change

Restricting to the observation of change restricts computational power:

Theorem

The singleton language $\{a^2ba\}$ cannot be accepted by any insertion change-observing acceptor

- *with insertion rules that have left and right contexts of length at most one each and*
- *insert an arbitrary number of letters.*

The strings a^2ba and aba^2 generate exactly the same sequences of rule applications.

Observing Change

The lack of control over the place of insertion could be compensated by more context.

Conjecture

Insertion change-observing acceptors

- with insertion rules that have left and right contexts of length two each and
- insert strings of length two

can accept all recursively enumerable languages.