Accepting Networks of Splicing Processors With Filtered Connections

Juan Castellanos[1], Florin Manea[2], Luis Fernando de Mingo López[3], Victor Mitrana[2,4]

[1] Department of Artificial Intelligence, Polytechnical University of Madrid
E-mail: jcastellanos@fi.upm.es ,
[2] Faculty of Mathematics and Computer Science, University of Bucharest
E-mail: flmanea@gmail.com ,
[3]Dept. Organización y Estructura de la Información Escuela Universitaria de Informática, Universidad Politécnica de Madrid
E-mail: lfmingo@eui.upm.es ,
[4]Research Group in Mathematical Linguistics, Rovira i Virgili University
E-mail: mitrana@fmi.unibuc.ro

Outline

Accepting Hybrid Networks of Evolutionary Processors

Introduced in: *Hybrid Networks of Evolutionary Processors as Accepting Devices* (DNA 10) by M. Margenstern, V. Mitrana, M. J. Perez-Jimenez (related to earlier work by E. Csuhaj-Varju, A. Salomaa, V. Mitrana and J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere).

Accepting Hybrid Networks of Evolutionary Processors

Introduced in: *Hybrid Networks of Evolutionary Processors as Accepting Devices* (DNA 10) by M. Margenstern, V. Mitrana, M. J. Perez-Jimenez (related to earlier work by E. Csuhaj-Varju, A. Salomaa, V. Mitrana and J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere).

- **Network**: a graph with two distinguished nodes *In*, *Out*.
  Edges: bidirectional communication channels.
  Information: *strings*.
  A node contains (is associated with) a set of strings.

Accepting Hybrid Networks of Evolutionary Processors

Introduced in: *Hybrid Networks of Evolutionary Processors as Accepting Devices* (DNA 10) by M. Margenstern, V. Mitrana, M. J. Perez-Jimenez (related to earlier work by E. Csuhaj-Varju, A. Salomaa, V. Mitrana and J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere).

- **Network**: a graph with two distinguished nodes *In*, *Out*.
  Edges: bidirectional communication channels.
  Information: *strings*.
  A node contains (is associated with) a set of strings.
- All the sets of strings associated with the nodes of the network: **Configuration**.

Accepting Hybrid Networks of Evolutionary Processors

Introduced in: *Hybrid Networks of Evolutionary Processors as Accepting Devices* (DNA 10) by M. Margenstern, V. Mitrana, M. J. Perez-Jimenez (related to earlier work by E. Csuhaj-Varju, A. Salomaa, V. Mitrana and J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere).

- **Network**: a graph with two distinguished nodes *In*, *Out*.
  Edges: bidirectional communication channels.
  Information: *strings*.
  A node contains (is associated with) a set of strings.
- All the sets of strings associated with the nodes of the network: **Configuration**.
- Each node can process strings in a specialized way (substitution, deletion, insertion): **Evolutionary Processors**.

Accepting Hybrid Networks of Evolutionary Processors

Introduced in: *Hybrid Networks of Evolutionary Processors as Accepting Devices* (DNA 10) by M. Margenstern, V. Mitrana, M. J. Perez-Jimenez (related to earlier work by E. Csuhaj-Varju, A. Salomaa, V. Mitrana and J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere).

- **Network**: a graph with two distinguished nodes *In*, *Out*.
  Edges: bidirectional communication channels.
  Information: *strings*.
  A node contains (is associated with) a set of strings.
- All the sets of strings associated with the nodes of the network: **Configuration**.
- Each node can process strings in a specialized way (substitution, deletion, insertion): **Evolutionary Processors**.
- The way a node communicates with other nodes is restricted by **permitting/forbidding input/output filters**: look-up for the presence/absence of several symbols.

Computation in AHNEPs

- Initially: input string is present in *In*.

Computation in AHNEPs

- Initially: input string is present in *In*.
- Every node *processes* the strings that it contains.
  A node contains all the strings that can be obtained in this way from its previous configuration.

Computation in AHNEPs

- Initially: input string is present in *In*.
- Every node *processes* the strings that it contains.
  A node contains all the strings that can be obtained in this way from its previous configuration.
- Every node *communicates* copies of each string it contains to all its neighbors.
  A node contains the strings that were not allowed to exit the node plus the strings that were allowed to enter the node. Strings that leave a node but cannot enter any other node are lost.

Computation in AHNEPs

- Initially: input string is present in *In*.
- Every node *processes* the strings that it contains.
  A node contains all the strings that can be obtained in this way from its previous configuration.
- Every node *communicates* copies of each string it contains to all its neighbors.
  A node contains the strings that were not allowed to exit the node plus the strings that were allowed to enter the node. Strings that leave a node but cannot enter any other node are lost.
- Processing, Communication, Processing, Communication, ...

Computation in AHNEPs

- Initially: input string is present in *In*.
- Every node *processes* the strings that it contains.
  A node contains all the strings that can be obtained in this way from its previous configuration.
- Every node *communicates* copies of each string it contains to all its neighbors.
  A node contains the strings that were not allowed to exit the node plus the strings that were allowed to enter the node. Strings that leave a node but cannot enter any other node are lost.
- Processing, Communication, Processing, Communication, . . .
- This string is **Accepted**: after several steps a string enters *Out*. The computation halts when the network accepts, or when a the configurations remain identical after consecutive processing steps or consecutive communication steps.

Significant Results for AHNEPs

- Completeness and Universality
- Characterizations for: NP, P, $PSPACE$ and $Co-NP$ via AHNEP-Complexity Classes
- Problem Solving: NP Complete problems solved in *linear* AHNEP-time and with linearly bounded resources

Accepting Network of Splicing Processors

Introduced in: *Accepting networks of splicing processors* (CiE 2005) by F. Manea, V. Mitrana and C. Martin-Vide. Modify the Processing phase of the NEPs.

Accepting Network of Splicing Processors

Introduced in: *Accepting networks of splicing processors* (CiE 2005) by F. Manea, V. Mitrana and C. Martin-Vide. Modify the Processing phase of the NEPs.

- We replace all the operations of EP by *splicing*. The result: **Splicing Processors**

Accepting Network of Splicing Processors

Introduced in: *Accepting networks of splicing processors* (CiE 2005) by F. Manea, V. Mitrana and C. Martin-Vide. Modify the Processing phase of the NEPs.

- We replace all the operations of EP by *splicing*. The result: **Splicing Processors**
- Each node of an AHNEP did a different job: different operations, different rules.

Accepting Network of Splicing Processors

Introduced in: *Accepting networks of splicing processors* (CiE 2005) by F. Manea, V. Mitrana and C. Martin-Vide. Modify the Processing phase of the NEPs.

- We replace all the operations of EP by *splicing*. The result: **Splicing Processors**
- Each node of an AHNEP did a different job: different operations, different rules. Same for Splicing Processors: it can apply the splicing operation only on pairs of strings consisting in a string from its configuration and a string from a predefined set of auxiliary strings (specific for each node), according to a set of splicing rules (specific for each node).

Accepting Network of Splicing Processors

Introduced in: *Accepting networks of splicing processors* (CiE 2005) by F. Manea, V. Mitrana and C. Martin-Vide. Modify the Processing phase of the NEPs.

- We replace all the operations of EP by *splicing*. The result: **Splicing Processors**
- Each node of an AHNEP did a different job: different operations, different rules. Same for Splicing Processors: it can apply the splicing operation only on pairs of strings consisting in a string from its configuration and a string from a predefined set of auxiliary strings (specific for each node), according to a set of splicing rules (specific for each node).
- In *All NP-problems can be solved in polynomial time by accepting networks of splicing processors of constant size* (DNA 12) by F. Manea, C. Martin-Vide, V. Mitrana, we generalize the Networks of Splicing Processors by allowing the splicing operation to be applied to any pair of strings contained in a node.

Significant Results for ANSPs

- Completeness and Universality (small universal ANSPs)
- Characterizations for: *NP* and *PSPACE* via ANSP-Complexity Classes. All NP-problems can be solved in polynomial time by accepting networks of splicing processors of constant size and constant number of splicing rules and axioms.
- Problem Solving: NP Complete problems solved in *linear* ANSP-time and with linearly bounded resources

ANSPs with Filtered Connections

Modify the Communication phase of the NSPs.

ANSPs with Filtered Connections

Modify the Communication phase of the NSPs.

- We drop the nodes' filters. Instead we place filters on edges: each edge is viewed as a two-way channel such that input and output filters coincide.

ANSPs with Filtered Connections

Modify the Communication phase of the NSPs.

- We drop the nodes' filters. Instead we place filters on edges: each edge is viewed as a two-way channel such that input and output filters coincide.
- The possibility of controlling the computation in such networks seems diminished: for instance, there is no possibility to lose data during the communication steps.

ANSPs with Filtered Connections

Modify the Communication phase of the NSPs.

- We drop the nodes' filters. Instead we place filters on edges: each edge is viewed as a two-way channel such that input and output filters coincide.
- The possibility of controlling the computation in such networks seems diminished: for instance, there is no possibility to lose data during the communication steps.
- Results: Completeness, Characterizations for **NP** and **PSPACE**, Universality.

Definitions: Splicing

Definition

A splicing rule over the alphabet V: $\sigma = [x, y; u, v]$, with $x, y, u, v \in V^*$.
For a splicing rule σ over V as above and a pair of words (w, z) over the same alphabet we define **the action of σ on** (w, z) *by:*

$$\sigma(w,z) = \begin{cases} \{t \mid w = \alpha xy\beta, z = \gamma uv\delta \text{ for some words} \\ \alpha, \beta, \gamma, \delta \in V^* \text{ and } t = \alpha xv\delta \text{ or } t = \gamma uy\beta\} \\ \\ \{w\} \cup \{z\}, \text{ if the set above is empty.} \end{cases}$$

Definitions: Splicing

Definition

A splicing rule over the alphabet V: $\sigma = [x, y; u, v]$, *with* $x, y, u, v \in V^*$.
For a splicing rule σ *over* V *as above and a pair of words* (w, z) *over the same alphabet we define* **the action of** σ **on** (w, z) *by:*

$$\sigma(w,z) = \begin{cases} \{t \mid w = \alpha xy\beta, z = \gamma uv\delta \text{ for some words} \\ \alpha,\beta,\gamma,\delta \in V^* \text{ and } t = \alpha xv\delta \text{ or } t = \gamma uy\beta\} \\ \\ \{w\} \cup \{z\}, \text{ if the set above is empty.} \end{cases}$$

Extension to a language L by $\sigma(L) = \bigcup_{w,z \in L} \sigma(w,z)$

Extension to a pair of languages L_1, L_2 by $\sigma(L_1, L_2) = \bigcup_{w \in L_1, z \in L_2} \sigma(w,z)$.

For a finite set of splicing rules M: $M(L) = \bigcup_{\sigma \in M} \sigma(L)$, $M(L_1, L_2) = \bigcup_{s \in M} \sigma(L_1, L_2)$.

Definitions: Filters

Definition

For two disjoint subsets P and F of an alphabet V and a word x over V, we define the predicates

$$\varphi^{s}(x; P, F) \equiv P \subseteq alph(x) \wedge F \cap alph(x) = \emptyset$$
$$\varphi^{w}(x; P, F) \equiv alph(x) \cap P \neq \emptyset \wedge F \cap alph(x) = \emptyset.$$

Definitions: Filters

Definition

For two disjoint subsets P and F of an alphabet V and a word x over V, we define the predicates

$$\varphi^s(x; P, F) \equiv P \subseteq alph(x) \ \wedge \ F \cap alph(x) = \emptyset$$
$$\varphi^w(x; P, F) \equiv alph(x) \cap P \neq \emptyset \ \wedge \ F \cap alph(x) = \emptyset.$$

$\varphi^s(x; P, F)$ (s stands for strong): all permitting symbols (P) are and no forbidding symbol (F) is present in x.

Definitions: Filters

Definition

For two disjoint subsets P and F of an alphabet V and a word x over V, we define the predicates

$$\varphi^s(x; P, F) \equiv P \subseteq alph(x) \wedge F \cap alph(x) = \emptyset$$
$$\varphi^w(x; P, F) \equiv alph(x) \cap P \neq \emptyset \wedge F \cap alph(x) = \emptyset.$$

$\varphi^s(x; P, F)$ (s stands for strong): all permitting symbols (P) are and no forbidding symbol (F) is present in x.
$\varphi^w(x; P, F)$ (w stands for weak): at least one permitting symbol (P) appears in x but still no forbidding symbol (F) is present in x.

Definitions: Filters

Definition

For two disjoint subsets P and F of an alphabet V and a word x over V, we define the predicates

$$\varphi^s(x; P, F) \equiv P \subseteq alph(x) \ \wedge \ F \cap alph(x) = \emptyset$$
$$\varphi^w(x; P, F) \equiv alph(x) \cap P \neq \emptyset \ \wedge \ F \cap alph(x) = \emptyset.$$

$\varphi^s(x; P, F)$ (s stands for strong): all permitting symbols (P) are and no forbidding symbol (F) is present in x.
$\varphi^w(x; P, F)$ (w stands for weak): at least one permitting symbol (P) appears in x but still no forbidding symbol (F) is present in x.

For every language $L \subseteq V^*$ and $\beta \in \{s, w\}$:
$\varphi^\beta(L, P, F) = \{x \in L \mid \varphi^\beta(x; P, F)\}$.

Definitions: NSPs with Filtered Connections

Definition

A splicing processor over V is a pair (S, A), where S is a finite set of splicing rules over V and A is a finite set of words over V (auxiliary words)

Definitions: NSPs with Filtered Connections

Definition

A splicing processor over V is a pair (S, A), where S is a finite set of splicing rules over V and A is a finite set of words over V (auxiliary words)

Definition

An accepting network of splicing processors with filtered connections is a 9*-tuple* $\Gamma = (V, U, <, >, G, \mathcal{N}, \alpha, x_I, x_O)$*, where:*
– V and U are the input and network alphabet, respectively, $V \subseteq U$;
$<, > \in U \setminus V$ are two special symbols.
– $G = (X_G, E_G)$ is an undirected graph without loops, called the underlying graph of the network.
– $\mathcal{N} : E_G \longrightarrow 2^U \times 2^U$ is a mapping which associates with each edge $e \in E_G$ the disjoint sets $\mathcal{N}(e) = (P_e, F_e)$ (filters of the edge).
– $\alpha : E_G \longrightarrow \{s, w\}$ defines the filter type of an edge.
– $x_I, x_O \in X_G$ are the input and the output node of Γ, respectively.

Computations in ANSPs with Filtered Connections

Configuration of a ANSPFC Γ: a mapping $C : X_G \longrightarrow 2^{U^*}$ which associates with every node the set of words which are present in any node at a given moment.

Computations in ANSPs with Filtered Connections

Configuration of a ANSPFC Γ: a mapping $C : X_G \longrightarrow 2^{U^*}$ which associates with every node the set of words which are present in any node at a given moment.

For a word $z \in V^*$, the initial configuration of Γ on z is defined by $C_0^{(z)}(x_I) = \{< z >\}$ and $C_0^{(z)}(x) = \emptyset$, $\forall x \in X_G \setminus \{x_I\}$.

Computations in ANSPs with Filtered Connections

Configuration of a ANSPFC Γ: a mapping $C : X_G \longrightarrow 2^{U^*}$ which associates with every node the set of words which are present in any node at a given moment.

For a word $z \in V^*$, the initial configuration of Γ on z is defined by $C_0^{(z)}(x_I) = \{< z >\}$ and $C_0^{(z)}(x) = \emptyset$, $\forall x \in X_G \setminus \{x_I\}$.

A configuration can change by a splicing step $C \Longrightarrow C'$, where: $C'(x) = S_x(C(x) \cup A_x)$ for all $x \in X_G$.

Computations in ANSPs with Filtered Connections

Configuration of a ANSPFC Γ: a mapping $C : X_G \longrightarrow 2^{U^*}$ which associates with every node the set of words which are present in any node at a given moment.

For a word $z \in V^*$, the initial configuration of Γ on z is defined by $C_0^{(z)}(x_I) = \{< z >\}$ and $C_0^{(z)}(x) = \emptyset$, $\forall x \in X_G \setminus \{x_I\}$.

A configuration can change by a splicing step $C \Longrightarrow C'$, where:
$C'(x) = S_x(C(x) \cup A_x)$ for all $x \in X_G$.
If $C'(x) = S_x(C(x), A_x)$ for all $x \in X_G$, then all processors of Γ are called *restricted*.

Computations in ANSPs with Filtered Connections

Configuration of a ANSPFC Γ: a mapping $C : X_G \longrightarrow 2^{U^*}$ which associates with every node the set of words which are present in any node at a given moment.

For a word $z \in V^*$, the initial configuration of Γ on z is defined by $C_0^{(z)}(x_I) = \{< z >\}$ and $C_0^{(z)}(x) = \emptyset$, $\forall x \in X_G \setminus \{x_I\}$.

A configuration can change by a splicing step $C \Longrightarrow C'$, where:
$C'(x) = S_x(C(x) \cup A_x)$ for all $x \in X_G$.
If $C'(x) = S_x(C(x), A_x)$ for all $x \in X_G$, then all processors of Γ are called *restricted*.

A configuration can change by a communication step $C \Longrightarrow C'$, where:
$C'(x) = (C(x) \setminus (\bigcup_{\{x,y\} \in E_G} \varphi^{\alpha(\{x,y\})}(C(x), \mathcal{N}(\{x, y\})))) \cup$
$(\bigcup_{\{x,y\} \in E_G} \varphi^{\alpha(\{x,y\})}(C(y), \mathcal{N}(\{x, y\})))$, for all $x \in X_G$.

Computations in ANSPs with Filtered Connections

The computation of an ANSPFC Γ on the input word $z \in V^*$: a sequence of configurations $C_0^{(z)}, C_1^{(z)}, C_2^{(z)}, \ldots,$ with $C_0^{(z)}$ the initial configuration of Γ on z, $C_{2i}^{(z)} \Longrightarrow C_{2i+1}^{(z)}$ and $C_{2i+1}^{(z)} \vdash C_{2i+2}^{(z)}$, for all $i \geq 0$.

Computations in ANSPs with Filtered Connections

The computation of an ANSPFC Γ on the input word $z \in V^*$: a sequence of configurations $C_0^{(z)}, C_1^{(z)}, C_2^{(z)}, \ldots$, with $C_0^{(z)}$ the initial configuration of Γ on z, $C_{2i}^{(z)} \Longrightarrow C_{2i+1}^{(z)}$ and $C_{2i+1}^{(z)} \vdash C_{2i+2}^{(z)}$, for all $i \geq 0$.

A computation *halts* if one of the following two conditions holds:
(i) There exists a configuration in which the set of words existing in the output node x_O is non-empty. In this case, the computation is said to be an *accepting computation*.

(ii) There exist two identical configurations obtained either in consecutive evolutionary steps or in consecutive communication steps.

The Accepted Language

Definition

The language accepted by Γ is

$$L_a(\Gamma) = \{z \in V^* \mid \text{ the computation of } \Gamma \text{ on } z \text{ is an accepting one.}\}$$

The language accepted by Γ with restricted processors is

$$L_a^{(r)}(\Gamma) = \{z \in V^* \mid \text{ the computation of } \Gamma \text{ on } z \text{ is an accepting one.}\}$$

We say that an ANSPFC Γ (with restricted processors) decides the language $L \subseteq V^*$, and write $L(\Gamma) = L$ iff $L_a(\Gamma) = L$ ($L_a^{(r)}(\Gamma) = L$) and the computation of Γ on every $z \in V^*$ halts.

Complexity Classes

The *time complexity* of the finite computation $C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \ldots C_m^{(x)}$ of Γ on $x \in V^*$ is denoted by $Time_\Gamma(x)$ and equals m.

Complexity Classes

The *time complexity* of the finite computation $C_0^{(x)}$, $C_1^{(x)}$, $C_2^{(x)}$, $\ldots C_m^{(x)}$ of Γ on $x \in V^*$ is denoted by $Time_\Gamma(x)$ and equals m.
$Time_\Gamma(n) = \max\{Time_\Gamma(x) \mid x \in V^*, |x| = n\}$.
We say that Γ decides L in time $O(f(n))$ if $Time_\Gamma(n) \in O(f(n))$.

Complexity Classes

The *time complexity* of the finite computation $C_0^{(x)}$, $C_1^{(x)}$, $C_2^{(x)}$, $\ldots C_m^{(x)}$ of Γ on $x \in V^*$ is denoted by $Time_\Gamma(x)$ and equals m.
$Time_\Gamma(n) = \max\{Time_\Gamma(x) \mid x \in V^*, |x| = n\}$.
We say that Γ decides L in time $O(f(n))$ if $Time_\Gamma(n) \in O(f(n))$.

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define:

$$\begin{aligned}\mathbf{Time}_{ANSPFC_p}(f(n)) &= \{L \,|\, \text{there exists an ANSPFC } \Gamma, \text{ of size } p, \text{ deciding } L, \\ &\quad \text{and } n_0 \text{ such that } Time_\Gamma(n) \leq f(n) \forall n \geq n_0\}\end{aligned}$$

Complexity Classes

The *time complexity* of the finite computation $C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \ldots C_m^{(x)}$ of Γ on $x \in V^*$ is denoted by $Time_\Gamma(x)$ and equals m.
$Time_\Gamma(n) = \max\{Time_\Gamma(x) \mid x \in V^*, |x| = n\}$.
We say that Γ decides L in time $O(f(n))$ if $Time_\Gamma(n) \in O(f(n))$.

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define:

$$\begin{aligned}\mathbf{Time}_{ANSPFC_p}(f(n)) &= \{L \mid \text{there exists an ANSPFC } \Gamma, \text{ of size } p, \text{ deciding } L, \\ &\quad\ \text{and } n_0 \text{ such that } Time_\Gamma(n) \leq f(n) \forall n \geq n_0\}\end{aligned}$$

We write:
$\mathbf{PTime}_{ANSPFC_p} = \bigcup_{k \geq 0} \mathbf{Time}_{ANSPFC_p}(n^k)$ for all $p \geq 1$

$\mathbf{PTime}_{ANSPFC} = \bigcup_{p \geq 1} \mathbf{PTime}_{ANSPFC_p}$.

Complexity Classes

The *length complexity* of the finite computation $C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \ldots C_m^{(x)}$ of Γ on $x \in V^*$ is denoted by $Length_\Gamma(x)$ and equals is denoted by $Length_\Gamma(x)$ and equals $\max_{w \in C_i^{(x)}(z), i \in \{1,\ldots,m\}, z \in X_G} |w|$.
$Length_\Gamma(n) = \max\{Length_\Gamma(x) \mid x \in V^*, |x| = n\}$.

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define

$$\mathbf{Length}_{ANSPFC_p}(f(n)) = \{L \mid \text{there exists an ANSPFC } \Gamma, \text{ of size } p, \text{ deciding } L, \text{ and } n_0 \text{ such that } Length_\Gamma(n) \leq f(n) \forall n \geq n_0\}$$

We write

$$\mathbf{PLength}_{ANSPFC_p} = \bigcup_{k \geq 0} \mathbf{Length}_{ANSPFC_p}(n^k), \text{ for all } p \geq 1,$$

$$\mathbf{PLength}_{ANSPFC} = \bigcup_{p \geq 1} \mathbf{PLength}_{ANSPFC_p}.$$

The classes for ANSPFC with restricted processors: $\mathbf{PTime}_{ANSPFC_p^{(r)}}$ and $\mathbf{PLength}_{ANSPFC_p^{(r)}}$.

Completeness and Complexity

Theorem

For any language L, accepted (decided) by a Turing Machine M, there exists an ANSPFC Γ, of size 4, accepting (deciding) L. Moreover, Γ can be constructed such that:

1. *if $L \in NTIME(f(n))$ then $Time_\Gamma(n) \in \mathcal{O}(f(n))$.*
2. *if $L \in NSPACE(f(n))$ then $Length_\Gamma(n) \in \mathcal{O}(f(n))$.*

Completeness and Complexity

Theorem

For any language L, accepted (decided) by a Turing Machine M, there exists an ANSPFC Γ, of size 4, accepting (deciding) L. Moreover, Γ can be constructed such that:
1. if $L \in NTIME(f(n))$ then $Time_\Gamma(n) \in \mathcal{O}(f(n))$.
2. if $L \in NSPACE(f(n))$ then $Length_\Gamma(n) \in \mathcal{O}(f(n))$.

Proof: We construct an ANSPFC that simulates, in parallel, all the computations of the Turing machine on an input word.
1. Each move of the Turing machine M is simulated in a constant number splicing and communication steps. Therefore, Γ makes at most $\mathcal{O}(f(|w|))$ steps (both splicing and communication), where $f(|w|)$ is the number of steps made by M on the input w.
2. Also, if M uses $f(|w|)$ space on the input w, then $Length_\Gamma(n) \in \mathcal{O}(f(n))$.
3. The proof remains valid for ANSPFCs with restricted processors.

Completeness and Complexity

Turing machines can simulate ANSPFC as well (the Church-Turing Thesis supports this statement; formal proof is based on keeping track of all the configurations of the ANSPFC during the computation).

Completeness and Complexity

Turing machines can simulate ANSPFC as well (the Church-Turing Thesis supports this statement; formal proof is based on keeping track of all the configurations of the ANSPFC during the computation).

Theorem

For any ANSPFC Γ with restricted processors accepting the language L, there exists a Turing machine M accepting L. Moreover, M can be constructed such that:
1. M accepts in $\mathcal{O}((Time_{\Gamma}(n))^2)$ computational time for an input of length n, and
2. M accepts in $\mathcal{O}(Length_{\Gamma}(n))$ space for an input of length n.

Completeness and Complexity

Turing machines can simulate ANSPFC as well (the Church-Turing Thesis supports this statement; formal proof is based on keeping track of all the configurations of the ANSPFC during the computation).

Theorem

For any ANSPFC Γ with restricted processors accepting the language L, there exists a Turing machine M accepting L. Moreover, M can be constructed such that:
1. M accepts in $\mathcal{O}((Time_\Gamma(n))^2)$ computational time for an input of length n, and
2. M accepts in $\mathcal{O}(Length_\Gamma(n))$ space for an input of length n.

Proof: We construct a Turing machine that applies non-deterministically a series of splicing and communication steps to the input string, until the string cannot be processed anymore or the outcome is an acceptable string.

Completeness and Complexity

Turing machines can simulate ANSPFC as well (the Church-Turing Thesis supports this statement; formal proof is based on keeping track of all the configurations of the ANSPFC during the computation).

Theorem

For any ANSPFC Γ with restricted processors accepting the language L, there exists a Turing machine M accepting L. Moreover, M can be constructed such that:
1. M accepts in $\mathcal{O}((Time_\Gamma(n))^2)$ computational time for an input of length n, and
2. M accepts in $\mathcal{O}(Length_\Gamma(n))$ space for an input of length n.

Proof: We construct a Turing machine that applies non-deterministically a series of splicing and communication steps to the input string, until the string cannot be processed anymore or the outcome is an acceptable string.

Open problem: Does a similar result holds in the case of unrestricted ANSPFCs?

Computational Complexity

Theorem

1. $\mathbf{NP} = \mathbf{PTime}_{ANSPFC_4^{(r)}}$.
2. $\mathbf{PSPACE} = \mathbf{PLength}_{ANSPFC_4^{(r)}}$.

Universality

Theorem

There exists a deterministic Turing machine T_U, with the input alphabet A, satisfying the following conditions on any input $code(\Gamma)code(z)$, where Γ is an arbitrary unrestricted ANSPFC and z is a word over the input alphabet of Γ:
(i) T_U halts on the input $code(\Gamma)code(z)$ if and only if Γ halts on the input z.
(ii) $code(\Gamma)code(z)$ is accepted by T_U if and only if z is accepted by Γ.

Universality

Theorem

There exists a deterministic Turing machine T_U, with the input alphabet A, satisfying the following conditions on any input $code(\Gamma)code(z)$, where Γ is an arbitrary unrestricted ANSPFC and z is a word over the input alphabet of Γ:
(i) T_U halts on the input $code(\Gamma)code(z)$ if and only if Γ halts on the input z.
(ii) $code(\Gamma)code(z)$ is accepted by T_U if and only if z is accepted by Γ.

Theorem

There exists an ANSPFC of size 4, Γ_U, with the input alphabet A, satisfying the following conditions on any input $code(\Gamma)code(w)$, where Γ is an arbitrary ANSPFC and w is a word over the input alphabet of Γ:
(i) Γ_U halts on the input $code(\Gamma)code(w)$ if and only if Γ halts on the input w.
(ii) $code(\Gamma)code(w)$ is accepted by Γ_U if and only if w is accepted by Γ.

Further Work

- Smaller universal ANSPFCs? Conjecture: size 2, based on simulation of deterministic Turing machines by ANSPFCs .

Further Work

- Smaller universal ANSPFCs? Conjecture: size 2, based on simulation of deterministic Turing machines by ANSPFCs .
- ANSPFCs simulate efficiently Turing machines. Are there universal (restricted) ANSPFCs that simulate efficiently (restricted) ANSPFCs?

THANK YOU!