Computer Science Journal of Moldova, vol.**18**, no.1(52), 2010

A flexible navigation mechanism for complex data models

Oleg Burlaca

Abstract

The paper presents a way to build flexible navigation tools over a big dataset of well structured data models. The mechanism is underpinned by a search engine that is used to slice and dice the database. By applying a series of consecutive groupings, the result of a search query can be organized in a hierarchical structure and browsed using traditional user interface controls.

1 Introduction

Building successful user interfaces (UI) requires a good understanding of the end user needs. As the data model of an application evolves with the addition of new objects and relationships, it's hard to create a sustainable UI because of the user's ever changing requirements. To alleviate the issue we should devote more time to the initial development phase: the software architecture. Nevertheless, it's not always possible to encompass all aspects at the beginning. Moreover, the requirements may change at a later stage when the software is already delivered and launched.

The approach presented in [1] works well for websites due to simple data models that were used. As the number of entities in a model increase, there is a need to quickly seek a desired object basing on different criteria. It may happen that the user knows the chain of interrelated objects that are connected to the object he is looking for, and would like to easily spot it by "jumping" from one object to another.

©2010 by O. Burlaca

Any information system that deals with a vast amount of data should feature a search engine. Our idea is to leverage the search engine and build the navigation system on the fly. In the next section we introduce the data model we used for our application. Then we discuss the search engine implementation, and finally we present the idea of a ”navigation widget” that is powered by the search engine.

2 The Data Model

The Events Data Model elaborated by HURIDOCS [2] is depicted in Figure 1.

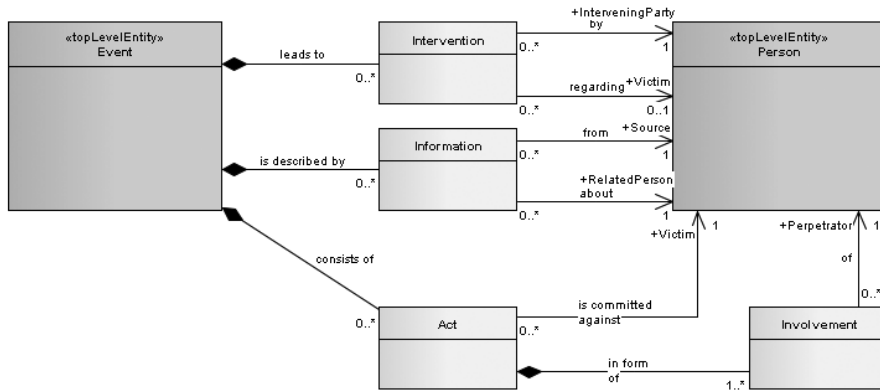

Figure 1. Events Data Model.

An Event is a complex thing, it is comprised of several entities: intervention, information and act. Acts are further comprised of victims, perpetrators. Trying to build an interface for editing an Event seems to be a tedious task and it might happen that due to the diverse number of sub-entities (interventions, acts) and complementary tasks (adding a new person or uploading a file), it might get cumbersome and a hard nut to crack for simple users. The proposed solution is to work with only one entity at a time and provide a suitable ”contextual skeleton”

for the user. In other words, we try to split a complex problem into smaller, but simpler ones, thus applying the divide and conquer design paradigm. (See [3] for an in depth discussion about the UI part).

By looking at the model in Figure 1, one can notice that if to exclude the person entity, the remaining structure has only 1 to Many relationships, i.e.: An event has many intervention, information and act entities. An act has many involvement entities. *It allows us to display an event as a tree.* In Figure 2 one can see the "Death Threats Against Monsignor Alvaro Ramazzini" event with all its acts, victims, perpetrators, information and other entities. We can think about this tree as a "contextual skeleton": the user will be able to edit a single entity from the tree, but he will always know to which parent entities it belongs (is it a victim or a perpetrator and to which act and event it belongs).

Figure 2. A hierarchical representation of an Event.

Let's assume that the data entry aspect is solved [3] and we have

a quite big database of such events (some real world databases have more than 20,000 events). We need a mechanism to easily browse this dataset, but we can't know beforehand how a user might want to browse the collection. In the next section we describe a search engine that will power our navigation mechanism: imagine that you'll be able to browse events in different ways: by date (an entity attribute), by type of act (a relationship), by method of violence, etc.

3 The Search Engine

There are different types of entities in the system, thus creating a search query will start by selecting what kind of entities to return (Events, Acts, Information). But it doesn't mean that it will be impossible to search for one type of entity and have attributes of another type in search results. For example, it will be possible to search for acts and display the Event.Title. Because of the 1 to Many nature of the event data model (an event has many acts, an act has many involvements, but an act CAN'T belong to many events), one can uniquely identify the parent entity. It means that if there are Involvements in search results, one can also display Act.TypeOfAct and Event.Title. Conversely, one can't search for Events and display Act.TypeOfAct in search resutls, because an event has Many acts.

In conclusion, *you need to search for the deepest type of entity that appears in search conditions if you need to display its attributes in search results.* It may sound a bit abstract for now, practical examples provided in the next section will clarify things.

3.1 Simple search

A simple search query consists of:

a) Type of entity to fetch results from (Events, Acts, Interventions etc.). We'll refer to it as *ResultEntity*.

b) A set of search conditions coupled by logical operators: AND, OR, AND NOT, OR NOT. A search condition may refer to a

type of entity (let's call it EntityX) that is a parent or child for ResultEntity. This rule is *transitive*. i.e. the relationship is not required to be direct, it's ok if EntityX is a parent of a parent of ResultEntity or vice versa: EntityX is a child of a child of ResultEntity.

c) A set of attributes from ResultEntity and all its *parent entities* to display in search results.

The search query illustrated in Figure 3 has 2 conditions, the first one refers to the Event entity and the second refers to the Act entity.

Figure 3. A simple search query.

Following the rule "you need to search for the *deepest type of entity* that appears in search conditions", the user needs to select the Act entity as ResultEntity if he needs to display the Act.TypeOfAct attribute in search results. If he selects Events as ResultEntity, the search will work, but one will not be able to display Act attributes.

You should be aware that search results depend on the ResultEntity, and sometimes even the correctness (validity) of a query changes. Let's see this in practice. Take the example from Figure 3. Suppose we have two events in the database and we need to display only the title of the event in search results:

```
Event1
     Title: "threat against somebody"
     Acts: no acts

Event2
     Title: "a threat to the party leadership"
     Act1
```

```
TypeOfAct: Violations of the right to life
InitialDate: 01/05/2008
```

If ResultEntity = Event, the results will be fetched from the event collection and one will get both events. If ResultEntity = Act, one will get results from the act collection. Because Event1 doesn't have any acts, it will not be listed at all in the EventAct relationship table, thus only Event2 will be in search results. It should be stressed once again: a search condition is evaluated/performed against the ResultEntity database table. It means that a search condition's entity MUST be linked to the ResultEntity.

Let's examine a situation when the validity of a search query depends on ResultEntity, Figure 4.

Event.EventTitle contains threat
AND **Act.InitialDate** is before 21/07/2009
AND **Incident.MethodOfViolence** is Wounding
AND **Information.DateOfSourceMaterial** is before 27/05/2009

Figure 4. A simple search with incompatible search conditions.

If ResultEntity = Event, everything is ok.

If ResultEntity = Act or Incident, then we get into trouble.

Let's assume ResultEntity = Act. Because an Act is not linked by a parent/child relationship to an Information entity (see the rule in point *(b)* above), we can't uniquely identify an Information entity for an Act and vice versa. Technically speaking, the results we get from the fourth search condition do not contain an Act column, so it is not possible to join these results with results from other search conditions (we don't have a common entity by which to link two results sets).

3.2 Complex search

Imagine that we encapsulate a simple search into a block, we'll call it a *search block*. By applying logical operators (AND, OR, AND NOT,

OR NOT) we can create a complex search query that consists of several blocks, see Figure 5.

Figure 5. A complex search query.

Because each block has its own ResultEntity, we need to use the least common denominator Entity before joining the result sets. The system can automatically infer it, but as one can see in Figure 6, sometimes the user may need to set it manually.

Figure 6. Normalization of search blocks.

In Figure 6 the ResultEntity is Incident and Involvement correspondingly. The least common denominator is the Act entity, and the system will determine it automatically. But what if we need to display only the title of found events? Since the overall ResultEntity of these two blocks is Act, the search results will contain act entities. Surely, we can display the event's title, but the same event will appear many times because there are many acts per event. Let's run the search query on a database that contains just one event:

```
Event1
```

```
Act1
   Incident1
      MethodOfViolence = Beating
      Victim = PersonX
   Involvement1
      DegreeOfInvolvement = Planned the act
      TypeOfPerpetrator = Police
      Perpetrator = PersonY

Act2
   Incident2
      MethodOfViolence = Wounding
      Victim = PersonZ
   Involvement2
      DegreeOfInvolvement = Directly carried out the act
      TypeOfPerpetrator = Paramilitary forces
```

The first block returns `Incident1` and `Incident2`, the second block returns `Involvement1` and `Involvement2`. Joining two results sets will give us `Act1` and `Act2`. So, if we need to display only event attributes in search results, we'll get `Event1` listed twice. In order to remove duplicates, the user is required to manually select the NormalizationEntity. In our case, it will be the Event entity.

Remark: since NormalizationEntity applies to a block, it means it applies to a simple search. As a result, the definition of a simple search query will be expanded to include the (d) point: NormalizationEntity.

A query that doesn't have a NormalizationEntity is invalid. In Figure 7 the first search condition returns Person entities while the second one returns `Events`. These are top level entities that miss the parent/child relationship.

3.3 Person roles in search conditions

The following example illustrates how to search for acts that have victims from one of the specified countries.

The same applies to other entities:

Figure 7. Invalid search query.

Figure 8. Using person roles in search conditions.

```
- Intervention.InterveningPary.[PersonAttribute]
- Intervention.Victim.[PersonAttribute]
- Involvement.Perpetrator.[PersonAttribute]
- etc
```

3.4 Query Infeasibility

The proposed search mechanism is not able to perform all kinds of searches you might have in mind, but we've tried to find a trade off between ease of use and power. As a last resort, one can write a bunch of SQL queries and PHP scripts to perform a complex and exotic search query. Here is an example of an unfeasible query: *Show me those events that have more than 3 acts and where one of the victims is also a perpetrator.*

4 The Navigation Mechanism

Having a powerful search engine capable of querying different types of entities at the same time and returning fields of different entities in the same row, we are able to introduce the concept of Navigation Widget.

What if we start grouping the flat list comprised of found rows by several columns? The output will be a hierarchy, where each level is

a "folder" that is actually an entity attribute. Once a complex search query is created, the user can specify a list of entity attributes used to group the results and save the query and attribute list as a *hierarchical result container*, such container we call *Navigation Widget*. The user can create an unlimited number of widgets, and the system will automatically display them as top level folders. Figure 9 illustrates three widgets: "By date", "By type of act", "[Event Widget]". The second folder "By type of act" is a two level navigation widget. Starting from the third level the system displays the event entities.

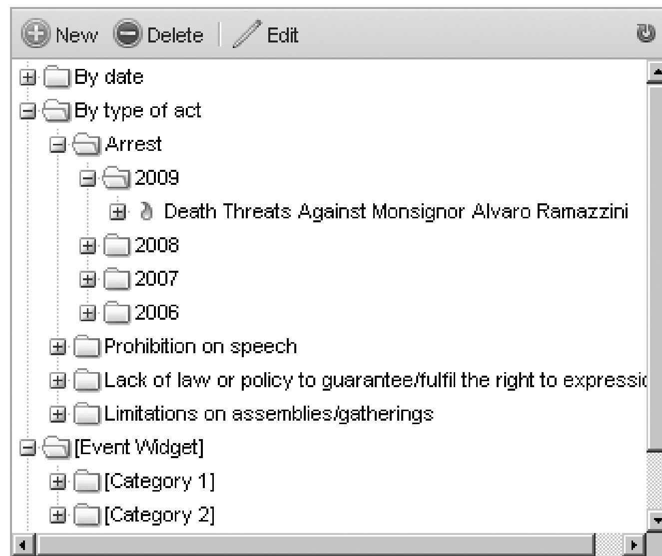

Figure 9. Navigation folders in event's tree.

A small Win32 application was developed that displays a collection of events along with attributes from child entities (see Figure 10). The application is available for download from http://openevsys.burlaca.com/. The user can drag&drop columns on the header and group search results, and he can group by as many columns as he wishes. The described approach depicts the way navigation folders are created. The

system may have two types of widgets:

- **global**: the administrator creates a widget that is automatically visible by all other users and they will not be able to modify/remove it;

- **local**: a user creates his own widgets that are not visible by others.

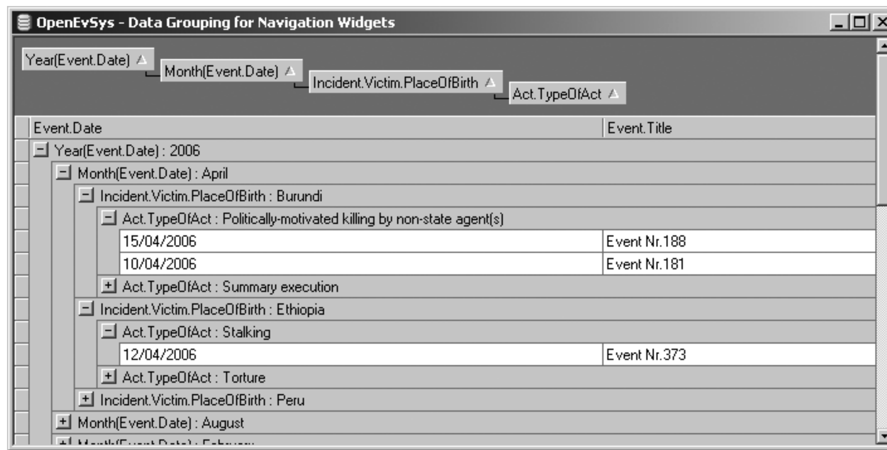

Figure 10. Grouping search results to create navigation folders.

5 Conclusions

Rapid Application Development tools became very flexible and powerful nowadays. Nevertheless, a developer has to rethink the User Interface when the underlying data model changes. The approach described in this paper implies considerable development efforts to implement the search engine, but the flexibility in managing an ever changing data model greatly reduces maintenance costs.

The approach is not universal in a sense that it can't be applied to an arbitrary data model, i.e. the search engine implementation may change.

For future work there are two directions: a) expand search engine capabilities to process different types of data models b) apply other paradigms in navigation mechanisms, e.g., the pivot tables concept as a complement to the hierarchical grouping.

References

[1] O. Burlaca. *Generic Interfaces for Managing Web Data.* Computer Science Journal of Moldova, vol. 13, nr. 1(37), 2005. pp. 70–83

[2] HURIDOCS - Human Rights Information and Documentation Systems, International. http://www.huridocs.org/

[3] OpenEvSys2 prototype. http://openevsys.burlaca.com/

Oleg Burlaca,
Received June 16, 2010

O. Burlaca
Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E–mail: *oburlaca@neonet.md*

Computer Science Journal of Moldova, vol.18, no.1(52), 2010

Three models for gene assembly in ciliates: a comparison

Miika Langille, Ion Petre, Vladimir Rogojin

Abstract

We survey in this paper the main differences among three variants of an intramolecular model for gene assembly: the general, the simple, and the elementary models. We present all of them in terms of sorting signed permutations and compare their behavior with respect to: (i) completeness, (ii) confluence (with the notion defined in three different setups), (iii) decidability, (iv) characterization of the sortable permutations in each model, (v) sequential complexity, and (vi) experimental validation.

Keywords: Ciliate, simple gene assembly, simple model, elementary model, confluence, completeness, characterization, sequential complexity, model validation, signed permutations, sorting.

1 Introduction

Gene assembly in ciliates has been subject of intense research in the last few years, both regarding the molecular details driving it, as well as the theoretical implications of some mathematical models proposed for it, see [7, 10, 25, 17, 16, 26, 1, 20].

Ciliates form an ancient and rich group of eukaryotes. There are about 8000 species of ciliates currently known. Two characteristics which are common for all ciliates distinguish them from other groups of unicellular eukaryotes. First, they all have *"cilia"*, organs used for motility and for feeding. Second, they all have two types of nuclei presented in each organism. Almost all RNA-transcriptions happen in

©2010 by M. Langille, I. Petre, V. Rogojin

macronuclei (somatic nuclei) during the life of a ciliate. The DNA-molecules in the *micronuclei* (germline nuclei) seem to remain silent until the sexual reproduction begins (see [24]).

The genetical information is stored in different ways on micro- and macronuclear molecules. The macronuclear genes are contiguous sequences of nucleotides. The micronuclear genes however, are split into coding blocks (called MDSs), shuffled and separated by noncoding blocks (called IESs). This shuffling and inversion of MDSs is especially visible in a species of ciliates called *stichotrichs*. Macronuclear molecules are known to be the shortest DNA in Nature, ranging in the Sterkiella nova organisms between 200bp and 3700bp with an average of 2200 bp in length (see [11, 5, 6, 23, 24, 27]). Macronuclear molecules consist mainly of coding sequences. On the other hand, coding sequences occupy as little as 2 – 5 % of the micronuclear molecules of the length about 10^7 bp (in Sterkiella nova, see [5, 23]).

At some point during sexual reproduction, ciliates destroy all macronuclei and develop new ones from the micronuclei. In the process they must excise non-coding sequences and assemble correctly all coding blocks of the micronuclear genes. This process is called gene assembly. For a brief introduction to the biology of ciliates, especially to the gene assembly process we refer to [7].

Two molecular models have been proposed for gene assembly in ciliates. The intermolecular model [17, 16] and the intramolecular model [10, 25] suggest splicing of gene fragments via short nucleotide sequences called pointers. Each pointer at the end of an MDS repeats at the beginning of the MDS which follows it in the assembled gene. Recent results [1, 20] suggest that some template molecules may assist the correct alignment of the recombining molecules. The intermolecular model suggests that two molecules may participate in the recombination, while the intramolecular model considers folding and recombination within a single molecule.

We focus in this paper on the intramolecular model (called in the sequel the general model) and on two of its variants: the simple model, introduced in [15] and the elementary model, introduced in [14].

The general model consists of three molecular operations, ld, hi,

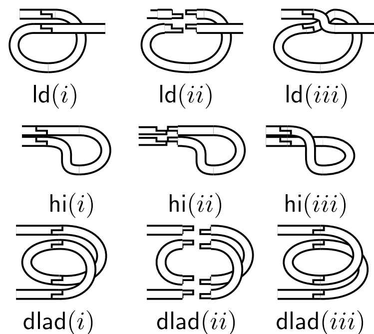

Figure 1. Illustration of the ld, hi, dlad molecular operation showing in each case: (i) the folding, (ii) the recombination, (iii) the result. Courtesy of Tero Harju.

dlad, see [10, 25]. The three operations are illustrated in Figure 1 where in each case we show the folding of the molecule on itself, the recombination that takes place and the subsequent result. A characteristic of this model is that all three operations operate on a single molecule that folds on itself in a specific way. One thus says that the model is *intramolecular*.

Note that the three intramolecular operations allow in their general formulation that the MDSs participating in an operation may be located anywhere along the molecule. Arguing on the principle of parsimony, a simplified model was introduced in [15], asking that all operations are applied 'locally'. This simple model consists of the same three molecular operations as the general model, requiring however that there is at most one coding block involved in each of the three operations. This idea was then further developed into two separate models, both using the terminology of *simple gene assembly*. In the first one, that we will refer to in here as the *elementary model*, introduced in [13, 14], the model was further restricted so that only *micronuclear*, but not *composite*, MDSs could be manipulated by the molecular operations. Consequently, once two or more micronuclear

MDSs are combined into a larger composite MDS, they can no longer be moved along the sequence. The second model, that we will refer to as the *simple model* [18], allowed that both micronuclear, as well as composite MDSs may be manipulated in each of the three molecular operations.

However minor the difference between the frameworks of the simple and the elementary models may seem, it does have a great impact on the characteristics of each model. We survey in this paper the main known results on the simple and elementary gene assembly, comparing them also with the corresponding properties of the general model with respect to: (i) completeness, (ii) confluence (with the notion defined in three different setups), (iii) decidability, (iv) characterization of the sortable permutations in each model, (v) sequential complexity, and (vi) experimental validation. For this, we introduce in this paper a permutation-based presentation of the general model. We discuss in particular the question of model validation and consider the assembly of all currently known ciliate gene patterns, see [4]. We also present several open problems in this area.

The results in this paper have been previously published in [8, 14, 18, 19] using non-uniform (and even conflicting) terminology and notation. In here we give the topic a uniform presentation, fix the terminology and discuss in some details differences among the three models of interest.

2 Mathematical preliminaries

For a finite alphabet $A = \{a_1, \ldots, a_n\}$, we denote by A^* the free monoid generated by A and call any element of A^* a *word*. For any $v \in A^*$, we denote $\mathrm{dom}(v) = \{a \in A \mid a \text{ occurs in } v\}$.

Let $\overline{A} = \{\overline{a}_1, \ldots, \overline{a}_n\}$, where $A \cap \overline{A} = \emptyset$. For $p, q \in A \cup \overline{A}$, we say that p, q have the same *signature* if either $p, q \in A$, or $p, q \in \overline{A}$ and we say that they have *different signatures* otherwise. For any $u \in (A \cup \overline{A})^*$, $u = x_1 \ldots x_k$, with $x_i \in A \cup \overline{A}$, for all $1 \leq i \leq k$, we denote $\|u\| = \|x_1\| \ldots \|x_k\|$, where $\|a\| = \|\overline{a}\| = a$, for all $a \in A$. We also denote $\overline{u} = \overline{x}_k \ldots \overline{x}_1$, where $\overline{\overline{a}} = a$, for all $a \in A$. We say, that u

is *uniformly signed*, if either $x_i \in A$ for all $1 \leq i \leq k$, or $x_i \in \overline{A}$ for all $1 \leq i \leq k$.

For strings u, v over Σ, we say that u is a *substring* of v, denoted by $u \leq v$, if $v = xuy$, for some strings x, y. We say that u is a *subsequence* of v, denoted by $u \leq_s v$, if $u = a_1 a_2 \ldots a_m$, $a_i \in \Sigma \cup \overline{\Sigma}$ and $v = v_0 a_1 v_1 a_2 v_2 \ldots a_m v_m$, for some strings $v_i, 0 \leq i \leq m$, over Σ.

A *permutation* π over A is a bijection $\pi : A \to A$. Fixing the order relation $(a_1, a_2, \ldots, a_m)$ over A, we often denote π as the word $\pi(a_1) \ldots \pi(a_m) \in A^*$. A *signed permutation* over A is a string $\psi \in (A \cup \overline{A})^*$, where $\|\psi\|$ is a permutation over A. We say that a signed permutation π is *(circularly) sorted* if it is of either of the following forms:

(i) $\pi = a_k a_{k+1} \ldots a_n a_1 \ldots a_{k-1}$, for some $k \geq 1$. In this case, we say that π is an *orthodox sorted permutation.*

(ii) $\pi = \overline{a_{k-1}} \ldots \overline{a_1}\, \overline{a_n} \ldots \overline{a_{k+1}}\, \overline{a_k}$, for some $k \geq 1$. In this case, we say that π is an *inverted sorted permutation.*

In both cases, if $k = 1$, then we say that π is a *linear sorted permutation*; otherwise, we say that it is *circular.*

A *sorted block* in the signed permutation π is a substring of π either of the form $a_i a_{i+1} \ldots a_j$, or of the form $\overline{a_j} \ldots \overline{a_{i+1}}\, \overline{a_i}$, $1 \leq i \leq j \leq n$, where $a_{i-1} a_i, \overline{a_i}\, \overline{a_{i-1}}, a_j a_{j+1}, \overline{a_{j+1}}\, \overline{a_j}$ are not substrings of π. By $S(\pi)$ we denote the total number of sorted blocks in π. Clearly, the permutation is cyclically sorted if we have $S(\pi) \leq 2$.

The notion of structure of a permutation will be useful in the paper. To define it, we first introduce the morphism $\xi_i : (A \cup \overline{A})^* \to (A \cup \overline{A})^*$, for any $1 \leq i \leq |A|$:

$$\xi_i(a_j) = \begin{cases} \lambda & \text{if } j = i; \\ a_j & \text{if } j < i; \\ a_{j-1} & \text{if } j > i; \end{cases}$$

where $a_j \in A \cup \overline{A}$.

Consider the mapping $\sigma_i : (A \cup \overline{A})^* \to (A \cup \overline{A})^*$, where for any string $u \in (A \cup \overline{A})^*$, $\sigma_i(u)$ is defined as follows:

(a) $\sigma_i(u) = u$, if $a_i a_{i+1} \not\leq u$, with $a_i, a_{i+1} \in A$, or $a_{i+1} a_i \not\leq u$, with $a_i, a_{i+1} \in \overline{A}$, and

(b) $\sigma_i(u) = \xi_i(u)$ otherwise.

Then, the structure of a string is the mapping $\sigma : (A \cup \overline{A})^* \to (A \cup \overline{A})^*$, such that $\sigma(u) = (\sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_{|A|-1} \circ \sigma_{|A|})(u)$. Note that the structure of a sorted permutation π is either $\sigma(\pi) = a_1$, or $\sigma(\pi) = a_2 a_1$, where $a_1, a_2 \in A$, or $\sigma(\pi) = a_1 a_2$, where $a_1, a_2 \in \overline{A}$.

Example 1. *Consider a sorted permutation* $\pi = 34512$. *We find its structure* $\sigma(\pi)$ *as follows:*

$\pi_5 = \sigma_5(\pi) = \pi$
$\pi_4 = \sigma_4(\pi_5) = \xi_4(\pi_5) = 3412$
$\pi_3 = \sigma_3(\pi_4) = \xi_3(\pi_4) = 312$
$\pi_2 = \sigma_2(\pi_3) = \pi_3$
$\pi_1 = \sigma_1(\pi_2) =$
$= \xi_1(\pi_2) = 21$
$\sigma(\pi) = \pi_1 = 21$

3 Gene assembly as a sorting of signed permutations

As discussed in [18, 13, 14], a natural formalization of the simple and elementary operations is through rewriting rules for signed permutations. A given gene is represented as a signed permutation by denoting the sequence and the orientation of its MDSs and assembling the gene is modeled through the sorting of the associated permutation. As shown in Definitions 1, 2, 3, the formalization of the molecular models in terms of sorting permutations is somewhat intricate: a high number of cases needs to be considered. For the general model, a more concise formalization can be done in terms of *signed double occurrence* (also called *legal*) strings, see [7]. The two main advantages of the legal string framework are: (i) it abstracts from denoting the sequence of gene blocks to denoting only the sequence of pointers (and in the process it ignores the two markers); (ii) it models gene assembly as a process of consecutive pointer removals, based on the observation that the assembled gene contains no pointers.

The simple model makes crucial use of the two markers. Consequently, this model can only be formalized through *extended* legal strings that denote the pointers, as well as the markers of the gene, as done in [3]. The resulting model is equivalent with the permutation-based model for simple operations but more concise for the same reasons (i)-(ii) discussed above.

In the case of the elementary model however, it is crucial that all pointers and markers are indicated throughout the gene assembly, rather than being removed as in the (extended) legal string framework. The main reason is that the elementary model distinguishes between the original (micronuclear) gene blocks and the larger (composite) blocks that are being formed throughout the process of assembly. It is an open problem whether a more concise formalizations may be introduced also for the elementary model.

In the following we consider a presentation based on signed permutations for all three models. This presentation in the case of the general model appears to be given here for the first time, although an equivalent presentation in terms of *MDS descriptors* was reported before, see [7]. As observed also in the case of simple and elementary operations, it is a characteristic of permutation-based models for gene assembly that the ld operation is not explicitly modeled. Instead, it is just assumed that two consecutive blocks are going to be spliced together in a bigger composite block at some arbitrary point, independently of the other operations applied to the permutation.

3.1 Modeling of the general operations

Consider a gene pattern formalized as a signed permutation over alphabet $\Pi_n = \{1, 2, \ldots, n\}$. We formalize the general operations over signed permutations as follows:

Definition 1. *i. For each* $1 \leq p < n$, hi_p *is defined as follows:*

$$\mathsf{hi}_p(xpy(\overline{p+1})z) = xp(p+1)\overline{y}z,$$
$$\mathsf{hi}_p(x\overline{p}y(p+1)z) = x\overline{y}p(p+1)z,$$
$$\mathsf{hi}_p(x(p+1)y\overline{p}z) = x\overline{y}(\overline{p+1})\overline{p}z,$$
$$\mathsf{hi}_p(x(\overline{p+1})ypz) = x(\overline{p+1})\overline{p}\,\overline{y}z,$$

where x, y, z *are signed strings over* Π_n. *We denote* $\mathsf{Hi} = \{\mathsf{hi}_i \mid 1 \leq i < n\}$.

ii. For each $1 \leq p, q < n$, *where* $|p - q| > 1$, $\mathsf{dlad}_{p,q}$ *is defined as follows:*

$$\mathsf{dlad}_{p,q}(xp''uq''vp'wq'z) = xwq'q''vp'p''uz,$$
$$\mathsf{dlad}_{p,q}(xp''uq'vp'wq''z) = xwvp'p''uq'q''z,$$
$$\mathsf{dlad}_{p,q}(xp'uq''vp''wq'z) = xp'p''wq'q''vuz,$$
$$\mathsf{dlad}_{p,q}(xp'uq'vp''wq''z) = xp'p''wvuq'q''z,$$

where $p' = p$, $p'' = p+1$, *or* $p' = (\overline{p+1})$, $p'' = \overline{p}$, *and* $q' = q$, $q'' = q+1$, *or* $q' = (\overline{q+1})$, $q'' = \overline{q}$, *and* x, u, v, w, z *are signed strings over* Π_n. *In all these cases, we also denote* $\mathsf{dlad}_{q,p} = \mathsf{dlad}_{p,q}$.

For each $1 < p < n$, *we define* $\mathsf{dlad}_{p-1,p}$ *and* $\mathsf{dlad}_{p,p-1}$ *as follows:*

$$\mathsf{dlad}_{p-1,p}(xp'''up''wp'z) = xwp'p''p'''uz,$$
$$\mathsf{dlad}_{p-1,p}(xp''vp'wp'''z) = xwvp'p''p'''z,$$
$$\mathsf{dlad}_{p-1,p}(xp'up'''vp''z) = xp'p''p'''vuz,$$

where $p' = p - 1$, $p'' = p$, $p''' = p + 1$, *or* $p''' = (\overline{p+1})$, $p'' = \overline{p}$, $p' = (\overline{p-1})$, x, u, v, w, z *are signed strings over* Π_n. *We denote* $\mathsf{Dlad} = \{\mathsf{dlad}_{i,j} \mid 1 \leq i, j < n, i \neq j\}$.

Example 2. *Consider the permutation* $\pi_1 = 2\overline{5}1\overline{4}376$. *We sort it by* hi *and* dlad *as follows:*

$\mathsf{hi}_5(2\overline{5}1\overline{4}376) = 2\overline{7}\,\overline{3}4\overline{1}56$ $\quad\quad$ $\mathsf{hi}_4(\overline{4}\,\overline{7}\,\overline{3}\,\overline{2}\,\overline{1}56) = 1237456$
$\mathsf{hi}_2(2\overline{7}\,\overline{3}4\overline{1}56) = 2374\overline{1}56$ $\quad\quad$ $\mathsf{dlad}_{3,6}(1237456) = 1234567$
$\mathsf{hi}_1(2374\overline{1}56) = \overline{4}\,\overline{7}\,\overline{3}\,\overline{2}\,\overline{1}56$

3.2 Modeling of the simple operations

Simple operations are a restriction of the general operations [9, 7]: they rearrange pieces of DNA containing at most one MDS, be that micronuclear, or composite.

Definition 2. *The molecular model of simple* hi *and simple* dlad *can be formalized as follows.*

i. For each $1 \leq p < n$, sh_p *is defined as follows:*

$$\begin{aligned}
&\mathsf{sh}_p(xp\ldots(p+i)(\overline{p+k})\ldots(\overline{p+i+1})y) =\\
&= xp\ldots(p+i)(p+i+1)\ldots(p+k)y,\\
&\mathsf{sh}_p(x(\overline{p+i})\ldots\overline{p}(p+i+1)\ldots(p+k)y) =\\
&= xp\ldots(p+i)(p+i+1)\ldots(p+k)y,\\
&\mathsf{sh}_p(x(p+i+1)\ldots(p+k)(\overline{p+i})\ldots\overline{p}y) =\\
&= x(\overline{p+k})\ldots(\overline{p+i+1})(\overline{p+i})\ldots\overline{p}y,\\
&\mathsf{sh}_p(x(\overline{p+k})\ldots(\overline{p+i+1})p\ldots(p+i)y) =\\
&= x(\overline{p+k})\ldots(\overline{p+i+1})(\overline{p+i})\ldots\overline{p}y,
\end{aligned}$$

where $k > i \geq 0$ *and* x, y *are signed strings over* Π_n*. We denote* $\mathsf{Sh} = \{\mathsf{sh}_i \mid 1 \leq i \leq n\}$.

ii. For each p, $2 \leq p \leq n-1$, sd_p *is defined as follows:*

$$\begin{aligned}
&\mathsf{sd}_p(x\,p\ldots(p+i)\,y\,(p-1)\,(p+i+1)\,z) =\\
&= xy(p-1)p\ldots(p+i)(p+i+1)z,\\
&\mathsf{sd}_p(x\,(p-1)(p+i+1)yp\ldots(p+i)z) =\\
&= x(p-1)p\ldots(p+i)(p+i+1)yz,\\
&\mathsf{sd}_{\overline{p}}(x(\overline{p+i+1})(\overline{p-1})y(\overline{p+i})\ldots\overline{p}z) =\\
&= x(\overline{p+i+1})(\overline{p+i})\ldots\overline{p}(\overline{p-1})yz,\\
&\mathsf{sd}_{\overline{p}}(x(\overline{p+i})\ldots\overline{p}y(\overline{p+i+1})(\overline{p-1})z) =\\
&= xy(\overline{p+i+1})(\overline{p+i})\ldots\overline{p}(\overline{p-1})z,
\end{aligned}$$

where $i \geq 0$ *and* x, y, z *are signed strings over* Π_n*. We denote* $\mathsf{Sd} = \{\mathsf{sd}_i, \mathsf{sd}_{\overline{i}} \mid 1 \leq i \leq n\}$.

Example 3. *Consider the following signed permutation* $\pi_1 = 5\overline{4}\,\overline{7}6\overline{3}\,\overline{1}\,\overline{2}$. *It can be sorted by the following composition of simple operations*

$$\mathsf{sh}_6(\pi) = 5\overline{4}\,\overline{7}\,\overline{6}\,\overline{3}\,\overline{1}\,\overline{2}, \qquad \mathsf{sh}_4 \circ \mathsf{sd}_{\overline{2}} \circ \mathsf{sh}_6(\pi) = \overline{5}\,\overline{4}\,\overline{7}\,\overline{6}\,\overline{3}\,\overline{2}\,\overline{1},$$
$$\mathsf{sd}_{\overline{2}} \circ \mathsf{sh}_6(\pi) = 5\overline{4}\,\overline{7}\,\overline{6}\,\overline{3}\,\overline{2}\,\overline{1}, \qquad \mathsf{sd}_{\overline{4}} \circ \mathsf{sh}_4 \circ \mathsf{sd}_{\overline{2}} \circ \mathsf{sh}_6(\pi) = \\ = \overline{7}\,\overline{6}\,\overline{5}\,\overline{4}\,\overline{3}\,\overline{2}\,\overline{1}.$$

3.3 Modeling of the elementary operations

The elementary model is a restriction of the simple model: elementary intramolecular operations rearrange only micronuclear MDSs. This leads to the following formalization for elementary operations.

Definition 3. *i. For each* $p \geq 1$, eh_p *is defined as follows:*

$$\mathsf{eh}_p(xp\overline{(p+1)}z) = xp(p+1)z,$$
$$\mathsf{eh}_p(x\overline{p}(p+1)z) = xp(p+1)z,$$
$$\mathsf{eh}_p(x(p+1)\overline{p}z) = x\overline{(p+1)}\overline{p}z,$$
$$\mathsf{eh}_p(x\overline{(p+1)}pz) = x\overline{(p+1)}\overline{p}z,$$

where x, z *are signed strings over* Π_n. *We denote* $\mathsf{Eh} = \{\mathsf{eh}_p \mid 1 \leq p \leq n\}$.

ii. For each p, $2 \leq p \leq n-1$, ed_p *is defined as follows:*

$$\mathsf{ed}_p(xpy(p-1)(p+1)z) = xy(p-1)p(p+1)z,$$
$$\mathsf{ed}_p(x(p-1)(p+1)ypz) = x(p-1)p(p+1)yz,$$
$$\mathsf{ed}_p(x\overline{p}y\overline{(p+1)}\,\overline{(p-1)}z) = xy\overline{(p+1)}\,\overline{p}\overline{(p-1)}z,$$
$$\mathsf{ed}_p(x\overline{(p+1)}\,\overline{(p-1)}y\overline{p}z) = x\overline{(p+1)}\,\overline{p}\,\overline{(p-1)}yz,$$

where x, y, z *are signed strings over* Π_n. *We denote* $\mathsf{Ed} = \{\mathsf{ed}_p \mid 1 < p < n\}$.

Note that $\mathsf{Eh} \subset \mathsf{Sh} \subset \mathsf{Hi}$ *and* $\mathsf{Ed} \subset \mathsf{Sd} \subset \mathsf{Dlad}$.

Example 4. *Assume the signed permutation* $\pi = 315\overline{2}46$. *It can be sorted by a composition of elementary operations as follows*

$$\mathsf{ed}_5(\pi) = 31\overline{2}456, \qquad \mathsf{ed}_3 \circ \mathsf{eh}_1 \circ \mathsf{ed}_5(\pi) = 123456.$$
$$\mathsf{eh}_1 \circ \mathsf{ed}_5(\pi) = 312456,$$

3.4 Sorting strategies: terminology

A composition of operations $\Phi = \phi_k \circ \phi_{k-1} \circ \ldots \phi_2 \circ \phi_1$, where all operations are from either $\mathsf{Hi} \cup \mathsf{Dlad}$, or $\mathsf{Sh} \cup \mathsf{Sd}$, or $\mathsf{Eh} \cup \mathsf{Ed}$ is called a *strategy.* A composition $\Phi = \phi_k \circ \phi_{k-1} \circ \ldots \phi_2 \circ \phi_1$ of operations is called a *sorting strategy* for π, if $\Phi(\pi)$ is a (circularly) sorted permutation. If $\phi \in (\mathsf{Hi} \cup \mathsf{Dlad})$ for all $1 \leq i \leq k$, we say that Φ is a *general sorting strategy.* If $\phi \in (\mathsf{Sh} \cup \mathsf{Sd})$ for all $1 \leq i \leq k$, we say that Φ is a *simple sorting strategy.* If $\phi \in (\mathsf{Eh} \cup \mathsf{Ed})$ for all $1 \leq i \leq k$, we say that Φ is an *elementary sorting strategy.* We say that an unsorted signed permutation π is *blocked* if no (simple, elementary) operation is applicable to it. We say that Φ is an *unsuccessful strategy* for π, if $\Phi(\pi)$ is blocked. If there are no sorting strategies for π, then we say that π is an *unsortable permutation.*

4 Comparison of the three models

In this section we compare the general, simple and elementary intramolecular models for gene assembly by different criteria:

- completeness: whether any gene pattern may be assembled or not;
- confluence, defined in three different ways:
    - (i) whether there are permutations having both successful and unsuccessful strategies,
    - (ii) whether different assembly strategies starting from the same gene pattern lead to assembled genes with the same structure,
    - (iii) if different assembly strategies starting from the same gene pattern lead to the same assembled gene;
- decidability of assembly: whether it is possible to decide effectively if a given gene pattern can be assembled or not;

- characterization of gene patterns that can be assembled (starting from certain characteristics of a given gene pattern we can conclude whether the gene pattern can be assembled);

- sequential complexity is constant: whether all assembly strategies apply the same number of intramolecular operations;

- model validation: whether it is consistent with biological data.

4.1 Completeness

It was shown in [9, 7] that the general model is complete, i.e., it assembles any gene pattern. The result was proved in terms of MDS-descriptors. To prove it for signed permutations, one may take two different approaches.

On one hand, one may observe that the set of signed permutations and that of MDS descriptors are in an one-to-one correspondence. Moreover, for a signed permutation π, if $\psi(\pi)$ is its corresponding MDS descriptor, then for any operation $f \in \mathsf{Hi} \cup \mathsf{Dlad}$, $\psi(f(\pi)) = f(\psi(\pi))$. The completeness result for signed permutations then follows easily from the corresponding result for MDS descriptors.

On the other hand, one may give a direct proof of the completeness, by essentially mimicking the proof in the case of MDS descriptors. The essential observation in this case is that for any $\phi \in \mathsf{Hi} \cup \mathsf{Dlad}$ and any signed permutation π, the number of sorted blocks of $\phi(\pi)$ is smaller than that of π (i.e., $S(\phi(\pi)) < S(\pi)$). One needs to observe then that a signed permutation π is sorted if and only if $S(\pi) \leq 2$ and π is uniformly signed.

Theorem 1. *All signed permutations are sortable over* $\mathsf{Hi} \cup \mathsf{Dlad}$.

Note however that the simple and the elementary models are not complete, as shown by the following example.

Example 5. *Consider the permutation $\pi = 321$. We cannot apply either* eh *or* sh *operations as all pointers have the same signature, and there is no applicable* ed *or* sd *operation either. On the other hand, π is successful in the general model:* $\mathsf{dlad}_{1,2}(\pi) = 123$.

4.2 Confluence

We consider the notion of *confluence* in three different setups, so as to reflect the success of different assembly strategies, the resulting gene structure, or the resulting gene pattern. These aspects are discussed below stressing the differences between the three models for gene assembly.

Consider first the most common notion of confluence, requiring that the result of all assemblies of a given input is the same. Equivalently, all strategies for a given signed permutation are confluent. It is easy to see that neither of the three models for gene assembly is confluent in this sense. For this, consider the permutation $\pi = 2413$. Then $\mathsf{dlad}_{2,1}(\pi) = \mathsf{sd}_2(\pi) = \mathsf{ed}_2(\pi) = 4123$, while $\mathsf{dlad}_{2,3}(\pi) = \mathsf{sd}_3(\pi) = \mathsf{ed}_3(\pi) = 2341$. Note that this observation does not contradict earlier invariant results of [8, 21], see also [7], where it was proved that the result of all assembly strategies of a given gene/string is always the same. The difference comes from considering a so-called *boundary* ld operation that would be applied as a last step in both strategies above to yield a circular string 1234 (that may also be denoted as 2341, 3412, or 4123, or even their inverses). In the permutation-based presentation, we have chosen to consider only standard linear permutations, rather than circular ones. The non-confluence result above is a direct consequence of this choice. We discuss more aspects of this matter in Section 5.

The example above shows that all three models are nondeterministic in the sense that different sorting strategies may lead to different results. A natural question is then whether a given signed permutation may have both successful, as well as unsuccessful strategies in any of the three models. Consider then the following notion of confluence. We say that the general (simple, elementary, resp.) model is confluent if there are no signed permutations having both successful and unsuccessful strategies.

It follows from Theorem 1 that the general model is indeed confluent in the sense above. As shown in [18], the simple model is also confluent. However, the elementary model is not confluent. To see it, consider the permutation $\pi = 24135$. Then $\mathsf{ed}_3(\pi) = 23415$ is a blocked

permutation, while $\mathsf{ed}_2 \circ \mathsf{ed}_4(\pi) = 12345$, a sorted permutation.

It was proved in [8, 21], see also [2], that for any gene pattern, either all general assembly strategies assemble it to a linear molecule, or all of them assemble it to a circular one. Consequently, even though if the assembly process is non-deterministic, the results of all possible assemblies of a given gene pattern have the same structure. I.e., the results of all *sorting* strategies applicable to a permutation have the same structure. As such, the same result holds also for all *sorting* strategies in the simple and in the elementary models. The question may however be asked also for the unsuccessful strategies. In this context, we say that a model for gene assembly is confluent if, for any signed permutation, all its sorting strategies lead to permutations having the same structure. Based on the considerations above, it follows easily that the general model is confluent in this sense, while the elementary model is not (since a permutation may have both successful and unsuccessful elementary strategies). Interestingly, it was proved in [18] that the simple model is in fact confluent in this sense.

Example 6. *Consider permutation $\pi = 623514$. There are only two simple strategies applicable to π: $\pi_1 = \mathsf{sd}_2(\pi) = 651234$ and $\pi_2 = \mathsf{sd}_4(\pi) = 623451$. These strategies are unsuccessful, and there are no other simple strategies applicable to π. Permutation π cannot be sorted by simple operations. Note however, that permutations π_1 and π_2 have the same structure $\sigma(\pi_1) = 321 = \sigma(\pi_2)$.*

The following table captures the behavior of the three models for gene assembly with respect to the three notions of confluence above. Interestingly, none of these notions distinguishes the simple and the general model. One property that does distinguish between the two is the completeness, valid only for the general model.

4.3 Deciding the sortability problem

For the simple and elementary models, which are not complete, deciding the sortability of a given signed permutation is an interesting problem. Based on the confluence results in the previous section, it

	Success	Same result	Same structure
General	confluent	not confluent	confluent
Simple	confluent	not confluent	confluent
Elementary	not confluent	not confluent	not confluent

Table 1. The results of considering confluence with regard to the three aspects are summarized here.

turns out that the problem is easy for the simple model: for any signed permutation, either all its sorting strategies are successful, or they are all unsuccessful. As such, to decide the sortability problem, it is enough to find an arbitrary strategy (e.g., using a straightforward procedure having quadratic time complexity) and answer 'yes'/'no', depending on whether or not that strategy is successful.

For the elementary model the problem of the eh-sortability of a signed permutation is easy.

Theorem 2 ([14]). *The signed permutation π is* eh*-sortable if and only if either*

(i) $\|\pi\| = k(k+1)\ldots n12\ldots(k-1)$ *and for some* $1 \leq i \leq k-1$, $k \leq j \leq n$ *we have* i, j *unsigned, or*

(ii) $\|\pi\| = (k-1)\ldots 21n\ldots(k+1)k$, *and for some* $1 \leq i \leq k-1$, $k \leq j \leq n$ *we have* i, j *signed.*

The problem of the ed-sortability turns out to be technically more involved, since a signed permutation may have both successful, and unsuccessful strategies. A complete characterization of the ed-sortable signed permutation has been given in [13, 14, 22]. The main notions used in the result are those of dependency graphs and forbidden elements. We only present here these notions for unsigned permutation; in the case of signed permutation, the setup is technically more complex, see [14]. Note also that an efficient decision procedure for the sortability problem is only known for unsigned permutation, see [22].

Dependency graphs in the elementary model

Dependency graphs suggest in which order elementary operations should be used to assemble a given gene pattern. Let π be an unsigned permutation with $\mathrm{dom}(\pi) = \{1, 2, \ldots, n\}$. We associate to it a *dependency graph* $\Gamma_\pi = (V_\pi, E_\pi)$, where $V_\pi = \mathrm{dom}(\pi)$, and

$$\begin{aligned} E_\pi = &\{(1,1),(n,n)\} \cup \{(i,i) | (i+1)(i-1) \leq_s \pi\} \cup \\ &\cup \{(j,i) | (i-1)j(i+1) \leq_s \pi\}. \end{aligned}$$

Intuitively, an edge (j, i) in Γ_π shows that in any sorting strategy for π, the operation ed_j should be used first, in order for ed_i to become applicable. If there is a loop (i, i) in Γ_π, then ed_i cannot be applied in any strategy applicable to π. We refer to [14] for a proof of these observations.

Example 7. *Consider the unsigned permutation* $\pi = 6\,2\,8\ \,4\,10\,7\,1\,3\,5\,9$. *Its associated dependency graph* $\Gamma_\pi = (V_\pi, E_\pi)$ *is shown in Figure 2.*

We have loops $(1, 1)$, $(5, 5)$, $(6, 6)$, $(10, 10)$ *in the dependency graph, and so, the operations* ed_1, ed_5, ed_6 *and* ed_{10} *cannot be applied in any strategy applicable to* G. *We have cycle* $8\,3\,8$ *in* Γ_π *and so, neither operation* ed_3, *nor operation* ed_8 *can be applied in any strategy applicable to* π. *The dependency graph* Γ_π *suggests the following order of operations to be applied in any sorting strategy of* π: ed_2 *should be applied before* ed_7, *and* ed_4 *should be applied before* ed_9. *Indeed, for instance, strategy* $\mathsf{ed}_9 \circ \mathsf{ed}_4 \circ \mathsf{ed}_7 \circ \mathsf{ed}_2(\pi)$ *sorts* π: $\mathsf{ed}_9 \circ \mathsf{ed}_4 \circ \mathsf{ed}_7 \circ \mathsf{ed}_2(\pi) = 6\,7\,8\,9\,10\,1\,2\,3\,4\,5$.

Forbidden elements and $\mathsf{ed}-$ sortability of unsigned permutations

For a signed permutation π, we say that $p \in \mathrm{dom}(\pi)$ is *forbidden* in π if and only if there exists no composition of eh and ed operations applicable to π with p in the domain of one of them. We denote U_π the set of all forbidden elements of π. It was proved in [14] that $p \in U(\pi)$ if and only if

(i) p is on a cycle of Γ_π or

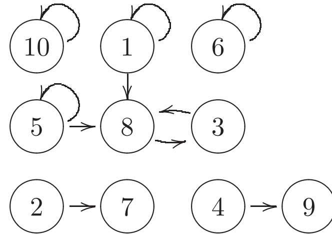

Figure 2. The dependency graph associated to $\pi = 6\,2\,8\,4\,10\,7\,1\,3\,5\,9$.

(ii) there is a path from q to p in Γ_π, for some q on a cycle of Γ_π or

(iii) there exists $r > 1$ such that there are paths from $r-1$ to p and from r to p in Γ_π.

The following result gives the ed-sortability of unsigned permutations.

Theorem 3 ([14]). *The unsigned permutation π is* ed*-sortable if and only if $\pi|_{U_\pi}$ is sorted.*

Finding an efficient method for the eh, ed-sortability of a signed permutation remains an open problem.

4.4 Characterization of sortable permutations

The following theorem characterizes ed-sortable unsigned permutations. A similar, albeit technically more involved, characterization exists also for signed permutations, see [14].

Theorem 4 ([14]). *Let π be a unsigned permutation. Then π is* Ed*-sortable if and only if there exists a partition $\{1, 2, \ldots, n\} = D \cup U$, such that the following conditions are satisfied:*

(i) $\pi|_U$ is sorted;

(ii) The subgraph induced by D in G_π is acyclic;

(iii) If $(p, q) \in G_\pi$ *with* $q \in D$*, then* $p \in D$*;*

(iv) For any $p \in D$*,* $(p-1)(p+1) \leq_s \pi$*;*

(v) For any $p \in D$*,* $(p-1), (p+1) \in U$*.*

For simple operations we do not have a characterization of sortable permutations for the moment. For general operations the question is moot since all signed permutations are sortable.

4.5 Sequential complexity

We focus now on the length of various sorting strategies of a given signed permutation, where the length is defined as the number of operations in the strategy. Consider first the general model and let $\pi_1 = 15\overline{2}\,\overline{4}36$. One can sort it by applying $\mathsf{dlad}_{1,5} \circ \mathsf{hi}_2$, or by applying $\mathsf{hi}_2 \circ \mathsf{hi}_3 \circ \mathsf{hi}_1$. These two sorting strategies are of different length, and use a different combination of operations.

Somewhat surprisingly, the situation is different in the simple model and by consequence, also in the elementary model. It was established in [19] (using a string-based formalism) that any two sorting strategies for a given signed permutation have the same assembly length.

Theorem 5 ([19])**.** *Let* π *be a signed permutation and* ϕ, ψ *be two simple sorting strategies for* π*. Then* ϕ *and* ψ *have the same sequential assembly length. Moreover, they have the same number of* sh *and the same number of* sd *operations.*

The differences between the general model and the two restricted models go beyond Theorem 5. E.g., when choosing operations in the simple model, we may always just choose the first available operation as the number of operations required in the end remains the same. If the operations were given different weights or costs, then the general model may have optimal and sub-optimal sorting strategies. We refer to [12] for a detailed discussion on various measures of complexity for gene assembly.

	General	Simple	Elementary
Completeness	complete	not complete	not complete
Confluence (Success)	confluent	confluent	not confluent
Confluence (Structure)	confluent	confluent	not confluent
Confluence (Result)	not confluent	not confluent	not confluent
Deciding Sortability	yes: trivial	yes: confluence	open for $\mathsf{eh} + \mathsf{ed}$
Characterizing sortable permutations	trivial	open	yes
Sequential Complexity constant	no	yes	yes
Model Validation	valid	valid	not valid

Table 2. Summary for general, simple and elementary intramolecular models

4.6 Model validation

A database of known sequences of micronuclear and macronuclear ciliate genes can be found in [4]. Based on the completeness result for the general model, it is clear that all the gene patterns have an assembly strategy in the general model. As it turns out however, the elementary model cannot account for the assembly of some of the gene patterns in [4].

Example 8. *Actin I gene in it Sterkiella nova is represented by the permutation* $\pi = 346579\bar{2}18$. *It is easy to check that there is no elementary sorting strategy applicable to* π. *However, we can sort* π *by applying the simple sorting strategy*

$$\mathsf{sh}_1 \circ \mathsf{sh}_2 \circ \mathsf{sd}_8 \circ \mathsf{sd}_5(\pi) = \bar{9}\,\bar{8}\,\bar{7}\,\bar{6}\,\bar{5}\,\bar{4}\,\bar{3}\,\bar{2}\,\bar{1}.$$

Below we will outline all the available scrambled gene patterns in [4], together with one simple sorting strategy. Genes that are not scrambled in their micronuclear form or the ones that have missing MDSs will not be included.

Actin I, Sterkiella n. : $\pi = 346579\bar{2}18$;

$$\mathsf{sh}_1 \circ \mathsf{sh}_2 \circ \mathsf{sd}_8 \circ \mathsf{sd}_6(\pi) = \overline{987654321}.$$

Actin I, Sterkiella h. : $\pi = 346579\,10\,\overline{2}18$;

$$\mathsf{sh}_1 \circ \mathsf{sh}_2 \circ \mathsf{sd}_8 \circ \mathsf{sd}_6(\pi) = \overline{10}\,\overline{987654321}.$$

Actin I, Stylonychia p. : $\pi = 346578\overline{2}1$;

$$\mathsf{sh}_1 \circ \mathsf{sh}_2 \circ \mathsf{sd}_6(\pi) = \overline{87654321}.$$

α **Telomere Binding Protein, Sterkiella n.** :

$$\pi = 1\,3\,5\,7\,9\,11\,2\,4\,6\,8\,10\,12\,13\,14;$$
$$\mathsf{sd}_{10} \circ \mathsf{sd}_8 \circ \mathsf{sd}_6 \circ \mathsf{sd}_4 \circ \mathsf{sd}_2(\pi) = $$
$$= 1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12\,13\,14.$$

DNA Polymerase α, Paraurostyla weissei:

$$\pi = \overline{44}\,\overline{42}\,\overline{40}\,\overline{38}\,\overline{36}\,\overline{34}\,\overline{32}\,\overline{30}\,\overline{28}\,\overline{26}\,\overline{24}\,\overline{22}\,\overline{20}\,\overline{18}\,\overline{16}\,\overline{14}\,\overline{12}\,\overline{10}\,\overline{8}$$
$$\overline{6}\,1\,2\,3\,4\,5\,7\,9\,11\,13\,15\,17\,19\,21\,23\,25\,27\,29\,31\,33\,35\,37\,39\,41$$
$$43\,45\,46\,47\,48$$

The signed permutation sorting strategy for this gene is just sh_1 repeated 40 times.

4.7 Summary

Table 2 summarizes properties of general, simple and elementary models considered in this paper.

5 Discussion and open problems

There has been significant interest in the last few years in the so-called simple operations for gene assembly, both for their biological appeal as a minimal, parsimonious model, but also for the properties of their mathematical formalization. The term simple has been used in connection with two different versions of the model. In this survey we review

these two models and fix the proper terminology. We also compare the mathematical properties of these two models with those of the general model.

For reasons detailed already in Section 3 we chose in this paper to follow a permutation-based presentation, rather than a string-based one. Indeed, a string-based presentation that would be more concise than the permutation-based one is still missing for the elementary model. Our choice of using permutations rather than strings has one direct consequence that we mentioned already in Section 4.2. Rather than eliminating all pointers as in the legal string and ending up with linear, or circular strings, we always end up with sorted linear permutations, where the term 'sorted' is extended to cover also permutations such as 3412. We call such a permutation circularly sorted, see Section 2. For this reason, a permutation such as 2413 may be sorted to two seemingly different results: either 2341, or 4123. Clearly, the two results correspond to the same circular string in the framework of legal strings. This ambiguity leads nevertheless to some open problems of independent interest. E.g., given a permutation that may be sorted circularly, enumerate efficiently all the circularly sorted permutations it can be sorted to. Similarly, the permutation $2\overline{13}$ may be sorted to either 231 or $\overline{213}$. One may also ask about the properties of those permutations that have sortings both to an unsigned permutation, as well as to a signed one. The properties of the three models may even be different in this respect.

There are two currently open problems related to the simple model: the linear decidability of the sortability problem and computing the number of sortable permutations of length n. It is however possible that these two problems are intertwined and an answer to one may at least partly solve the other.

Decidability. It was shown in [18] that it is possible to decide whether a permutation is sortable or unsortable in the simple model by applying available operations in an arbitrary order until the permutation is blocked or sorted. This gives us a quadratic method for deciding. Our first open problem is related to the optimality of this

method: is there a procedure to decide in linear time the sortability problem in the simple model?

For the elementary model, finding an efficient decision procedure for $\{\mathsf{eh}, \mathsf{ed}\}$-sortability problem is also open.

Sortable permutations of length n. As we pointed out also in this paper, not all permutations may be sorted using the simple operations. This differs from the general model which has been shown to be complete. Thus, an interesting problem is computing how many permutations of length n are sortable in the simple/elementary models. As a related problem, it should even be interesting to see whether the ratio of sortable signed permutations tends to 0 when n tends to infinity. Both problems are open also in the case of unsigned permutations.

6 Acknowledgments.

Ion Petre and Vladimir Rogojin are supported by Academy of Finland, project 108421. Vladimir Rogojin is on leave of absence from Institute of Mathematics and Computer Science of Academy of Sciences of Moldova, Chisinau MD-2028 Moldova. Vladimir Rogojin is supported by Science and Technology Center in Ukraine, project 4032.

References

[1] A. Angeleska, N. Jonoska, M. Saito, and L.F.Landweber. Rna-template guided dna assembly. *Journal of Theoretical Biology*, 248:706–720, 2007.

[2] R. Brijder, H. Hoogeboom, and G. Rozenberg. Reducibility of gene patterns in ciliates using the breakpoint graph. *Theoretical Computer Science*, 356:26–45, 2006.

[3] R. Brijder, M. Langille, and I. Petre. A string-based model for simple gene assembly. In E. Csuhaj-Varju and Z. Esik, editors,

FCT 2007, Proceedings, volume 4639 of *Lecture Notes in Computer Science*, pages 161–172. Springer-Verlag Berlin Heidleberg, 2007.

[4] A. Cavalcanti, T. Clarke, and L. Landweber. Mds_ies_db: a database of macronuclear and micronuclear genes in spirotrichous ciliates. *Nucleic Acids Research*, 33:396–398, 2005.

[5] W. Chang, P. Bryson, H. Liang, M. Shin, and L. Landweber. The evolutionary origin of a complex scrambled gene. In *Proceedings of the National Academy of Sciences of the US*, volume 102, pages 15149–15154, 2005.

[6] W. Chang, S. Kuo, and L. Landweber. A new scrambled gene in the ciliate *uroleptus*. *Gene*, 368:72–77, 2006.

[7] A. Ehrenfeucht, T. Harju, I. Petre, D. M. Prescott, and G. Rozenberg. *Computation in Living Cells: Gene Assembly in Ciliates*. Springer, 2003.

[8] A. Ehrenfeucht, I. Petre, D. M. Prescott, and G.Rozenberg. Circularity and other invariants of gene assembly in ciliates. In M. Ito, G. Paun, and S. Yu, editors, *Words, semigroups, and transductions*, pages 81–97. World Scientific, Singapore, 2001.

[9] A. Ehrenfeucht, I. Petre, D. M. Prescott, and G. Rozenberg. *Universal and simple operations for gene assembly in ciliates*, pages 329–342. Kluwer Academic, Dortrecht, 2001.

[10] A. Ehrenfeucht, D. M. Prescott, and G. Rozenberg. Computational aspects of gene (un)scrambling in ciliates. In L. F. Landweber and E. Winfree, editors, *Evolution as Computation*, pages 216–256. Springer, Berlin, Heidelberg, New York, 2001.

[11] W. Foissner and H. Berger. Identification and ontogenesis of the *nomen nudum* ypotrichs (protozoa: Ciliophora) *oxytricha nova* (*=sterkiella nova* sp. n.) and *o. trifallax* (*=s.histriomuscorum*). *Acta Protozool.*, 38:215–248, 1999.

[12] T. Harju, C. Li, I. Petre, and G. Rozenberg. Complexity measures for gene assembly. In K. Tuyls, editor, *Proceedings of the Knowledge Discovery and Emergent Complexity in Bioinformatics workshop*, volume 4366 of *Lecture Notes in Bioinformatics*, pages 42–60. Springer, 2007.

[13] T. Harju, I. Petre, V. Rogojin, and G. Rozenberg. Simple operations for gene assembly. In A. Carbone and N. A. Piercei, editors, *Proceedings of DNA-based computers 11*, volume 3892 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2006.

[14] T. Harju, I. Petre, V. Rogojin, and G. Rozenberg. Patterns of simple gene assembly in ciliates. *Discrete Applied Mathematics*, 2007. to appear.

[15] T. Harju, I. Petre, and G. Rozenberg. Modelling simple operations for gene assembly. In J.Chen, N.Jonoska, and G.Rozenberg, editors, *Nanotechnology: Science and Computation*, pages 361–376, 2006.

[16] L. F. Landweber and L. Kari. *Evolution as computation*, chapter Universal molecular computation in ciliates, pages 257–274. Natural computing series. Springer Verlag, Berlin, Heidelberg, New York, 1999.

[17] L. F. Landweber and L. Kari. The evolution of cellular computing: Nature's solution to a computational problem. In L. Kari, H. Rubin, and D. H. Wood, editors, *Proceedings of the 4th DIMACS Meeting on DNA-Based Computers*, volume 52, (1–3), pages 3–13. Elsevier, 1999.

[18] M. Langille and I. Petre. Simple gene assembly is deterministic. *Fundamenta Informaticae*, 73(1-2):179–190, 2006.

[19] M. Langille and I. Petre. Sequential vs. parallel complexity in simple gene assembly. *Theoretical Computer Science*, 395(1):24–30, 2008.

[20] M. Nowacki, V. Vijayan, Y. Zhou, K. Schotanus, T. Doak, and L. Landweber. Rna-mediated epigenetic programming of a genome-rearrangement pathway. *Nature*, 451:153–158, Jan. 2008. doi:10.1038/nature06452.

[21] I. Petre. Invariants of gene assembly in stichotrichous ciliates. *Information Technology*, 48(3):161–167, 2006.

[22] I. Petre and V. Rogojin. Decision problem for shuffled genes. *Information and Computation*, 2007. to appear.

[23] D. M. Prescott. The dna of ciliated protozoa. *Microbiology and Molecular Biology Reviews*, 58(2):233–267, 1994.

[24] D. M. Prescott. DNA manipulations in ciliates. In W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, editors, *Formal and Natural Computing*, volume 2300 of *Lecture Notes in Computer Science*, pages 394–417. Springer, 2002.

[25] D. M. Prescott, A. Ehrenfeucht, and G.Rozenberg. Molecular operations for dna processing in hypotrichous ciliates. *European Journal of Protistology*, 37:241–260, 2001.

[26] D. M. Prescott, A. Ehrenfeucht, and G.Rozenberg. Template-guided recombination for ies elimination and unscrambling of genes in stichotrichous ciliates. *Journal of Theoretical Biology*, 222:323–330, 2003.

[27] M. Swanton, J. Heumann, and D. Prescott. Gene-sized dna molecules of the macronuclei in three species of hypotrichs: size distribution and absence of nicks. *Chromosoma*, 77:217–227, 1980.

Miika Langille, Ion Petre, Vladimir Rogojin, Received March 22, 2010

Miika Langille,
Department of IT, Åbo Akademi University
ICT-building, Joukahaisenkatu 3-5 A, 5th floor
Turku 20520 Finland
E–mail: miika.langille@abo.fi

Ion Petre
Academy of Finland and
Turku Centre for Computer Science
Department of Computer Science, Åbo Akademi University
Turku 20520 Finland E–mail: ion.petre@abo.fi

Vladimir Rogojin
Turku Centre for Computer Science
Department of Computer Science, Åbo Akademi University
Turku 20520 Finland
E–mail: vrogojin@abo.fi

Computer Science Journal of Moldova, vol.18, no.1(52), 2010

All proper colorings of every colorable $BSTS(15)$

Jeremy Mathews, Brett Tolbert

Abstract

A Steiner System, denoted $S(t, k, v)$, is a vertex set X containing v vertices, and a collection of subsets of X of size k, called blocks, such that every t vertices from X are in exactly one of the blocks. A Steiner Triple System, or STS, is a special case of a Steiner System where $t = 2$, $k = 3$ and $v = 1$ or 3 $(mod 6)$ [7]. A Bi-Steiner Triple System, or $BSTS$, is a Steiner Triple System with the vertices colored in such a way that each block of vertices receives precisely two colors. Out of the 80 $BSTS(15)$s, only 23 are colorable [1]. In this paper, using a computer program that we wrote, we give a complete description of all proper colorings, all feasible partitions, chromatic polynomial and chromatic spectrum of every colorable $BSTS(15)$.

1 Introduction

A hypergraph is a generalized graph where an edge, called a hyperedge, can contain more than two vertices. A mixed hypergraph contains two kinds of hyperedges, C-edges and D-edges. A coloring of a mixed hypergraph is proper if every C-edge has at least two vertices mapped to the same color while every D-edge has at least two vertices mapped to different colors [5]. A Steiner Triple System, denoted by $STS(v)$ where v is the number of vertices, is a special case of a Steiner System in which its blocks are made up of exactly three vertices and no two blocks can share a pair of vertices [7]. Here we consider STSs on 15 vertices as bi-hypergraphs, which are mixed hypergraphs such that

©2010 by J. Mathews, B. Tolbert

the **C** family of C-edges and the **D** family of D-edges coincide, or equivalently $\mathbf{C} = \mathbf{D}$. We call these bi-hypergraphs Bi-Steiner Triple Systems of order 15 ($BSTS(15)$) and we consider every block of three vertices to be both a C-edge and a D-edge. Since each block of a $BSTS(15)$ contains exactly three vertices, two of those vertices must be mapped to the same color and the third vertex must be mapped to a different color to satisfy both the C-edge and D-edge requirements. Therefore, each block of a $BSTS(15)$ must be mapped to precisely two colors. The lower chromatic number of a mixed hypergraph is the minimum number of colors for which there exists a proper coloring, and it is denoted by χ [5]. The upper chromatic number of a mixed hypergraph is the maximum number of colors for which there exists a strict proper coloring, and it is denoted by $\bar{\chi}$ [5]. Of the 80 non-isomorphic $BSTS(15)$s, 23 contain $BSTS(7)$ as a subdesign and those 23 $BSTS(15)$s are colorable [1]. They are numbered in [1] as no. $1-22$ and no. 61. For these 23 $BSTS(15)$s, the upper and lower chromatic numbers are equal. They all have $\chi = \bar{\chi} = 4$ [2][3]. This means that all colorable $BSTS(15)$s are only colorable on exactly four colors. The chromatic spectrum of a mixed hypergraph is an integer vector, $R(H)$ whose components are $r_1, r_2, ..., r_k$, where r_i is the number of different feasible partitions into i color classes [5]. It is known that $BSTS(15)$s are only colorable on 4 colors, so $r_i = 0$, when $i \neq 4$. Only r_4 will have a value other than 0, so we can generalize and say the following:

$$R(BSTS(15)) = (0,0,0,r_{\bar{\chi}},0,0,0,0,0,0,0,0,0,0,0) \text{ [5]}$$

since $\chi = \bar{\chi} = 4$. Using a computer program that we wrote, we were able to find and display all proper colorings of every colorable $BSTS(15)$. Also we were able to display all feasible partitions and the permutations of colors of every $BSTS(15)$, which when multiplied together gives the number of all proper colorings. In this paper, we show all proper colorings, all feasible partitions, the chromatic polynomial, and the chromatic spectrum of all colorable $BSTS(15)$s.

2 Method

The chromatic polynomial of a mixed hypergraph is simply a polynomial in λ which gives the number of proper λ-colorings of a colorable mixed hypergraph, where λ is the number of available colors [5]. If we let $r_i(H)$ denote the number of feasible partitions of the mixed hypergraph into i color classes or sets, and we let $\lambda^{(i)}$ denote the falling factorial of λ, then we have the following equality:

$$P(H, \lambda) = \sum_{i=\chi(H)}^{\bar{\chi}(H)} r_i(H)\lambda^{(i)} \text{ [5].}$$

To construct the chromatic polynomial for each colorable $BSTS(15)$s, we need to alter the above equation to specify for $BSTS(15)$s. If we let a colorable $BSTS(15)$ be our H, the chromatic polynomial will give the number of proper λ-colorings of that colorable $BSTS(15)$, where $\lambda \geq 4$ since 4 colors are required for a proper λ-coloring of a colorable $BSTS(15)$. So we adjust the fundamental equality of mixed hypergraph coloring accordingly to accommodate the $BSTS(15)$s. First, [2, 3] showed that $\chi = \bar{\chi} = 4$ in all colorable $BSTS(15)$s; therefore, $\sum_{i=\chi(H)}^{\bar{\chi}(H)}$ is not needed. We simply say that $i = 4$ so now $r_{\chi}(H) = r_{\bar{\chi}}(H) = r_4(H)$. Therefore, these adjustments yield the following:

$$P(BSTS(15), \lambda) = r_4(BSTS(15))\lambda^{(i)}.$$

We also know that $BSTS(15)$s are colorable if and only if they contain $BSTS(7)$ as a subsystem or subdesign [4]. There are 21 feasible partitions in $BSTS(7)$ [6, 8] and there are four cases of $(7, 3, 1)$-subdesigns in various colorable $BSTS(15)$s. A $(7, 3, 1)$-subdesign is a subsystem on 7 vertices where each block contains 3 vertices and each vertex appears precisely once with every other vertex in a block. $BSTS(7)$ is the same as the finite projective plane of order 2, called the Fano Plane [7]. In all 23 cases of colorable $BSTS(15)$s, there exist(s):

1. 1 case in which a colorable $BSTS(15)$ has 15 (7,3,1)-subdesigns,
2. 1 case in which a colorable $BSTS(15)$ has 7 (7,3,1)-subdesigns,
3. 5 cases in which a colorable $BSTS(15)$ has 3 (7,3,1)-subdesigns, and
4. 16 cases in which a colorable $BSTS(15)$ has 1 (7,3,1)-subdesign.

[1]

This covers all colorable $BSTS(15)$s. Let the number of $(7,3,1)$-subdesigns in a $BSTS(15)$ be denoted by s. The number of feasible partitions in a particular colorable $BSTS(15)$ is equal to the number of feasible partitions in $BSTS(7)$ which is 21 using three colors, times the number of $(7,3,1)$-subdesigns in the $BSTS(15)$, or equivalently, $r_4 = 21s$. We will show this in the next section. Now we can write the fundamental equality of colorable $BSTS(15)$s as the following:

Proposition 1. *The number of proper λ-colorings of a colorable $BSTS(15)$ is a polynomial with the following form:*

$$P(BSTS(15), \lambda) = 21s(\lambda^{(4)}).$$

Now we will look at all colorable $BSTS(15)$s and arrive at each of their minimum number of proper colorings, their number of feasible partitions, and their chromatic spectrums.

3 Theorem and Proof

Theorem 1. *When $\lambda = 4$, the minimum number of proper colorings, the number of feasible partitions, and the minimum number of permutations of each partition of each colorable $BSTS(15)$ can be obtained by the following equality:*

$$P(BSTS(15), 4) = 21s(4!)$$

Proof. Let s denote the number of $(7,3,1)$-subdesigns in a particular colorable $BSTS(15)$. It is known that the number of feasible partitions of $BSTS(7)$ is 21 [6]. By an exhaustive search of all possible colorings using a program that we wrote and by applying the splitting-contraction algorithm [8] as defined in [5], the feasible partitions of $BSTS(7)$ are the following:

The blocks for $BSTS(7)$ are

`{1,2,4} {2,3,5} {3,4,6} {4,5,7} {5,6,1} {6,7,2} {7,1,3}`

The set of available colors is `{0,1,2}`

Vertices	1 2 3 4 5 6 7	Vertices	1 2 3 4 5 6 7
Partition 1	0 1 2 1 2 2 2	Partition 11	0 1 1 0 2 0 0
Partition 2	0 1 2 0 2 2 2	Partition 12	0 0 0 1 2 0 2
Partition 3	0 0 2 1 2 2 2	Partition 13	0 0 2 1 0 2 0
Partition 4	0 1 2 1 1 1 2	Partition 14	0 0 2 1 0 1 0
Partition 5	0 1 1 1 2 2 1	Partition 15	0 0 1 1 0 2 0
Partition 6	0 1 2 1 1 1 0	Partition 16	0 0 0 1 1 0 2
Partition 7	0 1 0 1 1 1 2	Partition 17	0 0 0 1 2 0 1
Partition 8	0 1 2 0 2 0 0	Partition 18	0 1 0 0 0 1 2
Partition 9	0 1 0 0 0 2 2	Partition 19	0 1 0 0 0 2 1
Partition 10	0 1 2 0 1 0 0	Partition 20	0 1 1 1 2 0 1
		Partition 21	0 1 1 1 0 2 1

Therefore, for any number of $(7,3,1)$-subdesigns in a colorable $BSTS(15)$, the number of the subdesigns times the number of feasible partitions of the subdesign will equal the number of feasible partitions of the colorable $BSTS(15)$. In the 4 cases of $(7,3,1)$-subdesigns mentioned earlier, the number of proper colorings, the number of feasible partitions, and the permutations of colors of each partition have been calculated by a computer program we wrote. The number of proper colorings, the number of feasible partitions, and the number of permutations of colors of each partition from the program for each of $(1-4)$ above are as follows:

1. 7560 proper colorings, 315 feasible partitions, and 24 permutations of colors of each partition;

2. 3528 proper colorings, 147 feasible partitions, and 24 permutations of colors of each partition;
3. 1512 proper colorings, 63 feasible partitions, and 24 permutations of colors of each partition; and
4. 504 proper colorings, 21 feasible partitions, and 24 permutations of colors of each partition.

This will also give us the chromatic spectrum. This is a simple proof by cases:

Case 1. $15(7,3,1)$-*subdesigns*

$$P(BSTS(15),4) = 21s(4!) = 21(15)(24) = 315(24) = 7560$$

This shows that the minimum number of proper colorings for $BSTS(15)$ no. 1 is 7560. It also gives the number of feasible partitions as $21(15) = 315$. The vertices of $BSTS(7)$ are mapped to this $BSTS(15)$, while keeping the same colorings and having the other eight vertices of this $BSTS(15)$ mapped to the fourth color. The vertices of $BSTS(7)$ are mapped to the vertices of $BSTS(15)$ no. 1 in the following way:

$BSTS(7)$:	1	2	3	4	5	6	7
$BSTS(15)$: first 21 partitions	1	2	4	3	6	7	5
$BSTS(15)$: second 21 partitions	1	2	8	3	10	11	9
$BSTS(15)$: third 21 partitions	1	2	12	3	14	15	13
$BSTS(15)$: fourth 21 partitions	1	4	8	5	12	13	9
$BSTS(15)$: fifth 21 partitions	1	4	10	5	14	15	11
$BSTS(15)$: sixth 21 partitions	1	6	10	7	12	13	11
$BSTS(15)$: seventh 21 partitions	1	6	8	7	14	15	9
$BSTS(15)$: eighth 21 partitions	2	4	8	6	12	14	10
$BSTS(15)$: ninth 21 partitions	2	4	9	6	13	15	11
$BSTS(15)$: tenth 21 partitions	2	5	9	7	12	14	11
$BSTS(15)$: eleventh 21 partitions	2	5	8	7	13	15	10

$BSTS(15)$: twelfth 21 partitions	3	4	9	7	13	14	10
$BSTS(15)$: thirteenth 21 partitions	3	4	8	7	12	15	11
$BSTS(15)$: fourteenth 21 partitions	3	5	8	6	13	14	11
$BSTS(15)$: fifteenth 21 partitions	3	5	9	6	12	15	10

Therefore, the above vertices in this $BSTS(15)$ are mapped to the same colors as in $BSTS(7)$ and the other eight vertices are mapped to the fourth color. This is minimum because $\lambda = 4$ is the minimum number of available colors needed to properly color any colorable $BSTS(15)$. This also shows that the number of feasible partitions is equal to the number of partitions in $BSTS(7)$ multiplied by the number of times the $(7, 3, 1)$-subdesign appears in the $BSTS(15)$, which is $21(15) = 315$ feasible partitions; therefore, the chromatic spectrum of this $BSTS(15)$ is:

$$R(BSTS(15), 4) = (0, 0, 0, 315, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0).$$

Case 2. $7(7, 3, 1)$-*subdesigns*

$$P(BSTS(15), 4) = 21s(4!) = 21(7)(24) = 147(24) = 3528$$

This shows that the minimum number of proper colorings for $BSTS(15)$ no. 2 is 3528. It also gives the number of feasible partitions as $21(7) = 147$. The vertices of $BSTS(7)$ are mapped to this $BSTS(15)$, while keeping the same colorings and having the other eight vertices of this $BSTS(15)$ mapped to the fourth color. The vertices of $BSTS(7)$ are mapped to the vertices of $BSTS(15)$ no. 2 in the following way:

$BSTS(7)$:	1	2	3	4	5	6	7
$BSTS(15)$: first 21 partitions	1	2	4	3	6	7	5
$BSTS(15)$: second 21 partitions	1	2	8	3	10	11	9
$BSTS(15)$: third 21 partitions	1	2	12	3	14	15	13
$BSTS(15)$: fourth 21 partitions	1	4	8	5	12	13	9
$BSTS(15)$: fifth 21 partitions	1	4	10	5	14	15	11
$BSTS(15)$: sixth 21 partitions	1	6	10	7	12	13	11
$BSTS(15)$: seventh 21 partitions	1	6	8	7	14	15	9

Therefore, the above vertices in this $BSTS(15)$ are mapped to the same colors as in $BSTS(7)$ and the other eight vertices are mapped to the fourth color. This is minimum because $\lambda = 4$ is the minimum number of available colors needed to properly color any colorable $BSTS(15)$. This also shows that the number of feasible partitions is equal to the number of partitions in $BSTS(7)$ multiplied by the number of times the $(7, 3, 1)$-subdesign appears in the $BSTS(15)$, which is $21(7) = 147$ feasible partitions; therefore, the chromatic spectrum of this $BSTS(15)$ is:

$$R(BSTS(15), 4) = (0, 0, 0, 147, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0).$$

Case 3. $3(7, 3, 1)$-*subdesigns*

$$P(BSTS(15), 4) = 21s(4!) = 21(3)(24) = 63(24) = 1512$$

This shows that the minimum number of proper colorings for $BSTS(15)$s no. $3 - 7$ is 1512. It also gives the number of feasible partitions as $21(3) = 63$. The vertices of $BSTS(7)$ are mapped to this $BSTS(15)$, while keeping the same colorings and having the other eight vertices of this $BSTS(15)$ mapped to the fourth color. The vertices of $BSTS(7)$ are mapped to the vertices of $BSTS(15)$ no. $3 - 7$ in the following way:

$BSTS(7)$:	1	2	3	4	5	6	7
$BSTS(15)s$: first 21 partitions	1	2	4	3	6	7	5
$BSTS(15)s$: second 21 partitions	1	2	8	3	10	11	9
$BSTS(15)s$: third 21 partitions	1	2	12	3	14	15	13

Therefore, the above vertices in these $BSTS(15)$s are mapped to the same colors as in $BSTS(7)$ and the other eight vertices are mapped to the fourth color. This is minimum because $\lambda = 4$ is the minimum number of available colors needed to properly color any colorable $BSTS(15)$. This also shows that the number of feasible partitions is

equal to the number of partitions in $BSTS(7)$ multiplied by the number of times the $(7,3,1)$-subdesign appears in the $BSTS(15)$, which is $21(3) = 63$ feasible partitions; therefore, the chromatic spectrum of these $BSTS(15)$s is:

$$R(BSTS(15),4) = (0,0,0,63,0,0,0,0,0,0,0,0,0,0,0).$$

Case 4. $1(7,3,1)$-*subdesign*

$$P(BSTS(15),4) = 21s(4!) = 21(1)(24) = 21(24) = 504$$

This shows that the minimum number of proper colorings for $BSTS(15)$s no. $8-22$ and no. 61 is 504. It also gives the number of feasible partitions as $21(1) = 21$. The vertices of $BSTS(7)$ are mapped to these $BSTS(15)$, while keeping the same colorings and having the other eight vertices of this $BSTS(15)$ mapped to the fourth color. The vertices of $BSTS(7)$ are mapped to the vertices of $BSTS(15)$ no. $8-22$ and no. 61 in the following way:

$BSTS(7)$:		1	2	3	4	5	6	7
$BSTS(15)s$:	21 partitions	1	2	4	3	6	7	5

Therefore, the above vertices in these $BSTS(15)$s are mapped to the same colors as in $BSTS(7)$ and the other eight vertices are mapped to the fourth color. This is minimum because $\lambda = 4$ is the minimum number of available colors needed to properly color any colorable $BSTS(15)$. This also shows that the number of feasible partitions is equal to the number of partitions in $BSTS(7)$ multiplied by the number of times the $(7,3,1)$-subdesign appears in the $BSTS(15)$, which is $21(1) = 21$ feasible partitions; therefore, the chromatic spectrum of these $BSTS(15)$s is:

$$R(BSTS(15),4) = (0,0,0,21,0,0,0,0,0,0,0,0,0,0,0).$$

Thus, all cases have been satisfied and all 23 colorable $BSTS(15)$s have been covered. Also, the $(2v+1)$ construction as described in [6] follows. $\square$

4 How the Program Works

This program was designed in C++ and contains several sub-programs and functions. We created files that were added to the source code of the program. We also created incidence matrices for each colorable $BSTS(15)$ as text files in the source code. We added display functions for all relevant data to check our results and to double check the computer results.

We started by creating headers that would find partitions and collect them not counting different permutations of colors. The main program calls the incidence matrix that is specified in a subprogram and displays it along with each block of vertices. The program then prompts the user to enter the number of colors that are to be used and then the number of colorings the user wishes to find (first 10 or first 200 for example, or the user can enter -1 for all colorings). If the user wants to find all proper colorings, the program runs an exhaustive search of all possible colorings from a string of all 0s to a string of all 3s. If a coloring is proper, then the coloring is displayed and counted; and if it is not a proper coloring, then that coloring is skipped. Also, if the coloring is proper, then that feasible partition is stored. After all proper colorings have been found and displayed and counted, the monitor prompts the user to press any key to see the feasible partitions displayed and counted and the number of colorings of each partition. All of the different permutations of colors of the partitions that were stored from the proper colorings are grouped together by the computer and only the first permutation of colors is displayed. For example, 011222233333333 would be displayed and 122333300000000 would not be displayed because it is a permutation of the same partition where vertex 1 is mapped to one color, vertices $2, 3$ are mapped to one color,

vertices $4-7$ are mapped to one color, and vertices $8-15$ are mapped to one color. When complete, the user can see the $BSTS(7)$ subsystem(s) and its coloring, and the expansion of colors to the remaining eight vertices. We were able to use this program to check the accuracy of our hypothesis and our results; and by displaying all of the relevant data on the monitor, we were able to check the accuracy of the computer results.

5 Concluding Remarks

This paper shows two things: 1. the minimal number of colorings over all feasible sets of colors, the number of feasible partitions, and the chromatic polynomial and chromatic spectrum for every colorable $BSTS(15)$; and 2. that computer science can be an invaluable part of research in mathematics. Of course, the findings on the number of proper colorings, the number of feasible partitions, and the number of permutations of colors of each partition can be generalized for any number of colors in a set of available colors by the equality in Proposition 1. It is true that all colorable $BSTS(15)$s are colorable only with 4 colors, but if you have 5 colors in the set of available colors, then you can choose to use some subset of colors from $1, 2, 3, 4, 5$ such as colors $1, 2, 3, 4$, or colors $1, 2, 3, 5$, or colors $2, 3, 4, 5$, etc. The generalization would simply be to change λ in the fundamental equality of colorable $BSTS(15)$s to whatever number of colors are available in the set of available colors. If we take the above example of 5 available colors on the $BSTS(15)$ with 15 $(7, 3, 1)$-subdesigns, we have:

Example 1. $P(BSTS(15), \lambda) = 21s(\lambda^{(4)}) = 21(15)(5^{(4)}) = 315(120) = 37,800$

This shows that the number of proper colorings with 5 colors available to use is $37,800$. This, of course, is not minimum. We still have $21(15) = 315$ feasible partitions and $(5^{(4)}) = 120$ permutations of colors of each partition. The chromatic spectrum would be:

$$R(BSTS(15)) = (0, 0, 0, 315, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$

since the number of feasible partitions would not change. There would simply be more permutations of colors of each partition.

We strongly believe that working with someone in a different field can help find answers and solutions to problems that have yet to be solved. We both benefited from working with one another on this project. When one may have strength in one area but very little development in another area, the other can bring balance in the areas needed. Also, the amount that you learn in the other's field is incredible. Working together also gives additional viewpoints and ideas that one may not think of on his or her own.

This paper leads obviously into the discussion of the 57 uncolorable $BSTS(15)$s and their induced and/or partial colorable $BSTS(15)$s. How many vertex deletions or edge deletions are needed to obtain an induced or partial colorable $BSTS(15)$? Is this result universal for all uncolorable $BSTS(15)$s? Much research is still needed in the area of uncolorable $BSTS(15)$s.

References

[1] C. J. Colbourn, A. Rosa. *Triple Systems.* (Oxford University Press, Inc.), New York, (1999).

[2] C. J. Colbourn, J. Dinitz, A. Rosa. *Bicoloring Steiner Triple Systems.* Electron. *Journal of Combinatorics* 6 (1999), R25.

[3] L. Milazzo, Zs. Tuza. *Upper chromatic number of Steiner Triple and Quadruple Systems. Discrete Mathematics* 174 (1997), 247-259.

[4] A. Rosa, V. I. Voloshin - Private Communication as cited in [6]

[5] V. I. Voloshin. *Introduction to Graph and Hypergraph Theory.* (Nova Science Publishers, Inc.), New York, (2009), 202-204.

[6] V. I. Voloshin. *Coloring mixed hypergraphs: theory, algorithms and applications. Fields Institute Monographs* (2002), 147-148.

[7] E. W. Weisstein, Steiner Triple System, *Mathworld - A Wolfram Web Source* http://mathworld.wolfram.com/SteinerTripleSystem.html, (2009).

[8] Splitting-Contraction Algorithm Applied to $BSTS(7)$ by J. Mathews.

Jeremy Mathews, Brett Tolbert,
Received January 13, 2010

Troy University, Troy, AL 36082
E–mail: *jmathews9106@gmail.com*
bretttolbert@gmail.co

Computer Science Journal of Moldova, vol.18, no.1(52), 2010

Convexity preserving interpolation by splines of arbitrary degree

Igor Verlan

Abstract

In the present paper an algorithm of C^2 interpolation of discrete set of data is given using splines of arbitrary degree, which preserves the convexity of given set of data.

Mathematics Subject Classification 2000: 65D05, 65D07, 41A05, 41A15.

Keywords and phrases: spline, interpolation, convexity preserving interpolation.

1 Introduction

It is well known that problems concerning nonnegativity, monotonicity, or convexity preserving interpolation have received considerable attention, because of their interest in computer aided design and in other practical applications [1]. Problem of construction of interpolating curve which preserves the convexity of the initial discrete set of data still remains in the focus of investigators [2]. In what follows an algorithm preserving the convexity of given set of data is presented.

2 Interpolating splines of arbitrary degree

Let us assume that the mesh $\Delta : a = x_0 < x_1 < ... < x_n = b$ is given on the interval $[a, b]$ and $f_i = f(x_i)$, $i = 0(1)n$, are the corresponding data points. Problem of construction of an interpolation function S, such that interpolation conditions $S(x_i) = f_i$, $i = 0(1)n$, are held and $S \in C^2[a, b]$, is considered.

©2010 by I. Verlan

Let us introduce splines as follows: on $[x_i, x_{i+1}]$

$$S(x) = f_i + (f_{i+1} - f_i)t + \\ + \frac{h_i^2 M_i (1-t)((1-t)^{\alpha_i} - 1)}{\alpha_i(\alpha_i + 1)} + \frac{h_i^2 M_{i+1} t (t^{\alpha_i} - 1)}{\alpha_i(\alpha_i + 1)}, \quad (1)$$

where the following notations are used:

$$t = (x - x_i)/h_i, h_i = x_{i+1} - x_i, S''(x_i) = M_i.$$

The α_i is a free parameter of the splines (1) and has to satisfy the condition $\alpha_i > 1$.

From (1) for the first derivative of spline we get:

$$S'(x) = \delta_i^{(1)} - \\ - \frac{h_i(M_i((\alpha_i + 1)(1-t)^{\alpha_i} - 1) - M_{i+1}((\alpha_i + 1)t^{\alpha_i} - 1))}{\alpha_i(\alpha_i + 1)}, (2)$$

where

$$\delta_i^{(1)} = (f_{i+1} - f_i)/h_i,$$

and for the second derivative, respectively:

$$S''(x) = M_i(1-t)^{\alpha_i - 1} + M_{i+1} t^{\alpha_i - 1}. \quad (3)$$

Obviously, at the knots of the mesh the second derivative is the continuous one.

For the first derivative at the knots of the mesh we have

$$S'(x_i-) = \delta_{i-1}^{(1)} + \frac{h_{i-1} M_{i-1}}{\alpha_{i-1}(\alpha_{i-1} + 1)} + \frac{h_{i-1} M_i}{\alpha_{i-1} + 1}$$

and

$$S'(x_i+) = \delta_i^{(1)} - \frac{h_i M_i}{\alpha_i + 1} - \frac{h_i M_{i+1}}{\alpha_i(\alpha_i + 1)}.$$

Requiring the continuity of the first derivative of the spline at the knots of the mesh we obtain the following system of linear algebraic equations:

$$c_i M_{i-1} + a_i M_i + b_i M_{i+1} = \delta_i^{(2)}, i = 1(1)n-1, \quad (4)$$

where

$$c_i = \frac{h_{i-1}}{\alpha_{i-1}(\alpha_{i-1}+1)},$$

$$a_i = \frac{h_{i-1}}{\alpha_{i-1}+1} + \frac{h_i}{\alpha_i+1},$$

$$b_i = \frac{h_i}{\alpha_i(\alpha_i+1)},$$

and

$$\delta_i^{(2)} = \delta_i^{(1)} - \delta_{i-1}^{(1)}.$$

The system (4), presented above, is the undetermined one. Since the system (4) provides only $n-1$ linear equations in $n+1$ parameters M_i, it follows that two additional linearly independent conditions are needed in order to have a determined system of equations.

In what follows we'll consider that the end conditions of the type $M_0 = f_0''$ and $M_n = f_n''$ are used as additional conditions.

In fact, it is easy to prove that the system of equations (4) has the diagonally dominant matrix of coefficients, therefore the solution of this system exists and it is the unique one for fixed parameters of the spline.

3 Convexity preserving algorithm

In this section it is considered that the initial set of data is the convex one, namely,

$$\delta_i^{(2)} \geq 0, i = 1(1)n-1.$$

From the formulae (3) it immediately follows, that in order to preserve the convexity of initial data the solution of the system (4) has to be the nonnegative one.

So, let's choose the value of free parameter as it follows:

$$\alpha_i \geq \max\left(\frac{2\delta_i^{(2)}}{\delta_{i+1}^{(2)}}, \frac{2\delta_{i+1}^{(2)}}{\delta_i^{(2)}}\right). \tag{5}$$

There is no problem to prove that in this case the solution of the system (4) is the nonnegative one.

This conclusion is based on the fact that for the coefficients of the matrix of linear algebraic equations (4) the following relations are valid:

$$\frac{c_i}{a_{i-1}} < \frac{1}{2}$$

and

$$\frac{b_i}{a_{i+1}} < \frac{1}{2}.$$

In this case we get that

$$\delta_i^{(2)} - \frac{c_i \delta_{i-1}^{(2)}}{a_{i-1}} - \frac{b_i \delta_{i+1}^{(2)}}{a_{i+1}} \geq 0.$$

As a result, taking into account [3] it can be concluded that the solution of the system (4) is the nonnegative one. As a result the spline, constructed using condition (5), preserves the convexity of the initial set of data.

4 Conclusions

In fact not only the problem of convexity preserving interpolation represents the interest, but also the problem of construction of interpolants which have the same number of inflection points as the initial set of data and preserve the convexity or concavity of data.

References

[1] B. I. Kvasov. *Methods of shape-preserving spline approximation*, World Scientific, Singapore, 2000.

[2] V. V. Bogdanov, Yu. S. Volkov, *Selection of parameters of generalized cubic splines with convexity preserving interpolation*, Sib. Zh. Vychisl. Mat., **9**:1 (2006), pp.5–22.

[3] I. I. Verlan. *Positive solutions of systems of linear algebraic equations with Jacobian matrices of coefficients.* Mat. Issled. No 104, Program. Obespech. Vychisl. Komlpeks. (1988), pp.52–59. (in Russian)

I.Verlan,

Received July 5, 2010

Department of Applied Mathematics, Moldova State University
60 Mateevich str. Chişinău,
MD2009, Moldova

Institute of Mathematics and Computer Science
5 Academiei str., Chişinău,
MD2028, Moldova
E–mail: *i_verlan@yahoo.com*

Computer Science Journal of Moldova, vol.18, no.1(52), 2010

Determining best-case and worst-case times of unknown paths in time workflow nets

Inga Camerzan

Abstract

In this paper we present a method aimed for determining best-case and worst-case times between two arbitrary states in a time workflow net. The method uses a discrete subset of the state space of the time workflow net and archives the results, which are integers.

1 Introduction

Time workflow nets (TWN) were developed to provide a suitable method to model, simulate, and analyze the behavior of time dependent systems, business processes. Best-case execution times (BCET) and worst-case execution times (WCET) are a necessary step in the development and validation process for hard real-time systems. Real-time systems need to satisfy stringent timing constraints, induced by the systems aims. We consider time workflow nets, those only which allow the modelling of time delay and deadlines for the execution of activities in the workflow process. This paper will mainly focus on modelling the control flow perspective. Thus the perspective workflow process definitions are formulated in order to specify which tasks need to be executed and in what order. If a worst-case input for the task were known, then there are reliable guarantees that processes always terminate. However the worst-case input is not known and it is hard to be determined.

In a workflow management system there is a delay beetween the moment when an activity becomes enabled and the moment when the

©2010 by I. Camerzan

activity is executed for a certain resource. For each transition t in the time workflow net there is a static interval $[a_t, b_t]$ associated to it. The times a_t, b_t are relative to the moment at which t was last enabled. Assuming that t was last enabled at the global time τ, then t may fire only during the interval $[a_t + \tau, b_t + \tau]$ and must fire the latest at the time $b_t + \tau$. This is a method of incorporating time into Petri Nets, introduced by Merlin [7] and studied in [1, 2, 8, 9, 10].

One of the most important problems, in the above mentioned nets, is to determine the deadlines for a sequence of system processes, i.e., to compare the longest duration of a transition sequence with a given limit. Such considerations are important for determining the best-case execution times and the worst-case execution times.

As a rule, the method used to estimate execution times bounds in practice consist in measuring the *end-to-end* execution time of the task for a subset of the possible executions, called test cases. This determines the minimal observed and the maximal observed execution times. Generally speaking, through these methods we are able to obtain an overestimation of the BCET and underestimation of the WCET, therefore we cannot consider them safe. Our aim is to propose an algorithm able to estimate the execution times in a safe way.

This paper is organized as follows: In Section 2 we introduce the basic notions and definitions for time workflow nets; In Section 3 we present the reachability graph of the time workflow nets, which can be used for computing the shortest and the longest path between two arbitrary states in the TWN; In the last section the algorithm for execution times estimation is proposed. The way it functions is illustrated by an example.

2 Basic notions and definitions

Definition 2.1 *A Petri net PN =(P, T, F, W) is a Workflow net iff:*

1. *PN has two additional places i and o, "start" place i, "destination" place o.*

2. *If we add a transition t^* to PN which connects o with i then the resulting Petri net is strongly connected.*

There are distinct methods of incorporating time in Petri nets: associating time delay to transition, associating time delay to places, associating time delay with arcs, associating time delays or time intervals to different types of objects of the net, associating stochastic time. Further we consider only Petri nets [5] which have deterministic time associated to transitions, in the form of time intervals, defined by Merlin [7] in 1972 and studied in [1, 2, 8, 9, 10].

We define a time workflow net in following way:

Definition 2.2 *A Time workflow net is a tuple Σ=(P, T, F, W, I) where PN=(P, T, F, W) is the workflow net (also called skeleton net), I: $T \rightarrow Q_0^+ \times Q_0^+$ is a time function which associates timed intervals with transitions and $I_1(t) \leq I_2(t)$, where $I(t) = (I_1(t), I_2(t))$, for each transition $t \in T$.*

A global clock is associated with the time workflow net, which begins to work as soon as the first token appears in the net. After time association, the workflow net will work in the following way: from the moment when a transition t is enabled, the tokens from the input locations are stored for $I_2(t) - I_1(t)$ time units, and after this time elapses the transition fires putting tokens in their output places. For transitions in conflict, the first transition that fires is the one which has the latest time interval smaller.

For the definition of a state and of a change of state of a net we will follow [3, 6]:

Definition 2.3 *Let Σ=(P, T, F, W, I) be a time workflow net and J: $T \rightarrow Q_0^+ \cup \{\sharp\}$. S=(m, J) is the state of the net Σ iff:*

1. *m is a marking in skeleton net,*

2. *if $t \in T$ and $t^- \leq m$, then $J(t) \leq I_2(t)$,*

3. *if $t \in T$ and $t^- \not\leq m$, then $J(t) = \sharp$,*

where $t^-(p) = W(p,t)$ *is arc weight from place* p *to transition* t.

We understand the notion of state in the following way. Let $S = (m, J)$ be a state. Each transition t in the net has a watch. The watch doesn't work ($J(t) = \sharp$) at the marking m if t is disabled at m. If t is enabled at m, then the watch of t shows the time $J(t)$ that has elapsed since t was last enabled.

Let $\Sigma = (P, T, F, W, I)$ be a time workflow net. The state $S_0 := (i, J_0)$ with i the initial marking of the time workflow net (the marking which has a single token in place i) and $J_0(t) = \begin{cases} 0, & \text{if} \quad t^- \leq m, \\ \sharp, & \text{if} \quad t^- \not\leq m \end{cases}$ is considered to be the initial state of the time workflow net. The states in a time workflow net can change due to transition firings or time elapsing.

Definition 2.4 *A transition* t *is enabled at the state S=(m, J), denoted by* $S \rightarrow$, *iff*

1. $t^- \leq m$;
2. $I_1(t) \leq J(t)$.

Thus, a transition is enabled in a time workflow net Σ, if t is enabled in the skeleton net (the timeless net) and the time specifications are satisfied, i.e t has been enabled for a sufficient amount of time. The resulting state is defined as follows:

Definition 2.5 *A transition* t *enabled at the state S=(m, J), will fire inducing state* $S' = (m', J')$, *denoted by* $S \rightarrow S'$ *defined thus:*

1. $m\prime = m + \triangle t$;
2. $$J'(t) = \begin{cases} \sharp, & t^- \not\leq m', \\ J(t), & t^- \leq m \wedge t^- \leq m\prime \wedge F_t \cap F'_t = 0, \\ 0, & \textit{otherwise}, \end{cases}$$

   where $\triangle t = W(t,p) - W(p,t)$.

The resulting state (m', J') has a marking m' which results by firing of transition t in the skeleton net and a time vector J'. The values of the vector for the not enabled transitions in the marking are undefined $\sharp$. If a transition was enabled in the old marking, and it is still enabled in the new marking, and it is not in conflict with the just fired transition, then it keeps the values of its local clock $J'(t) = J(t)$. Otherwise, if a transition has just become enabled in the new marking, then its local clock $J'(t) = 0$.

Definition 2.6 *Let $\Sigma = (P, T, F, W, I)$ be a time workflow net. The state $S = (m, J)$ changes into the state $S' = (m', J')$ by the time duration $\tau \in Q$, denoted by $S \xrightarrow{\tau} S'$ iff $m' = m$ and the time duration τ is possible i.e. for any $t \in T$ with $J(t) \neq \sharp$, we have $J(t) + \tau \leq I_2(t))$ and* $J'(t) = \begin{cases} J(t) + \tau, & \text{if} \quad t^- \leq m, \\ \sharp, & \text{if} \quad t^- \not\leq m. \end{cases}$

The sequence of transitions and time durations $\sigma = \tau_0, t_0, \tau_1, t_1, \ldots \tau_{n-1}, t_{n-1}$ is executable in the net Σ iff there exist the states $S_0, S'_0, S_1, S'_1, \ldots S'_{n-1}, S_n$ so that: $S_0 \xrightarrow{\tau_0} S'_0 \xrightarrow{t_0} S_1, \ldots S_{n-1} \xrightarrow{\tau_{n-1}} S'_{n-1} \xrightarrow{t_{n-1}} S_n$. That sequence shortly can also be noted by $S_0[\sigma\rangle S_n$. The transition-time sequence σ is called an execution sequence in the net Σ. State S' is reachable from the state S if there is a transition-time sequence σ so that $S[\sigma\rangle S'$. $RS(\Sigma, S_0)$ or $[S_0\rangle$ denotes the set of all reachable states of a net Σ and $R_\Sigma(S)$ denotes the set of all reachable states from the state S.

In Figure 1, the initial state is ((1,0,0,0,0,0)(0,$\sharp$, $\sharp$, $\sharp$)). Marking (1,0,0,0,0,0) is the initial marking of the skeleton net. The time vector has value 0 corresponding to transition t_1 enabled in the initial marking and the values $\sharp$ for the rest of the transitions which are disabled. Since t_1 is enabled at marking i and the time constraints are also satisfied, transition t_1 can fire in the initial state of the net Σ. The resulting state is $S_0 \xrightarrow{t_1} S_1 = ((0, 1, 1, 0, 0, 0)(\sharp, 0, 0, \sharp))$. The state S_1 has the marking $m_1 = (0, 1, 1, 0, 0, 0)$ and the time vector $J_1 = (\sharp, 0, 0, \sharp)$ with value 0 corresponding to the transitions t_2 and t_3, enabled in the marking m_1 in the skeleton net, and value $\sharp$ corresponding to the disabled transitions in the skeleton net. In the case of the time con-

straints, i.e. in the conditions $I_1(t_2) \geq J_1(t_2)$ and $I_1(t_3) \geq J_1(t_3)$ in state S_1 the transitions t_2 and t_3 cannot fire. So, state S_1 can change into another state in the net Σ only by time elapsing. For instance, the following state change is possible in the net Σ: $S_1 \stackrel{3.7}{\rightarrow} S_2$, where $S_2 = ((0,1,1,0,0,0)(\sharp, 3.7, 3.7, \sharp))$. We notice that the marking of the skeleton net remains unchanged, and results in a new time vector J_2 with updated time values for the transitions t_2 and t_3. The time duration 6.1 is not possible in state S_1 because the transition t_2, which, if enabled, must fire in 5 units of time. Now, both t_2 and t_3 can fire at state S_2. If t_2 fires, then we have $S_2 \stackrel{t_2}{\rightarrow} S_3 = ((0,0,1,1,0,0)(\sharp, \sharp, 3.7, \sharp))$. Now, the time duration 1.3 is possible at state S_3 and a new state appears: $S_3 \stackrel{1.3}{\rightarrow} S_4 = ((0,0,1,1,0,0)(\sharp, \sharp, 5, \sharp))$. Thus the state-transition-times sequence results in: $S_0 \stackrel{t_1}{\rightarrow} S_1 \stackrel{3.7}{\rightarrow} S_2 \stackrel{t_2}{\rightarrow} S_3 \stackrel{1.3}{\rightarrow} S_4$. The sequence $t_1, 3.7, t_2, 1.3$ is an execution sequence in the net Σ.

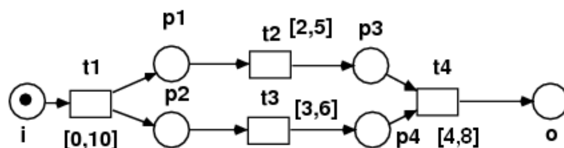

Figure 1. A time workflow net

3 Reachability graph of the time workflow net

The state space of a time workflow net is the set of all reachable states of the skeleton net, starting from i. Because of the markings, reachable in the net, this set is discrete, and it may be infinite. On the other hand, this set may be infinite because of the time of transitions. Thus, the set of all reachable states for a fix marking is infinite (and densely ordered) in general. Nevertheless, it is possible to pick up some "essential" states

only, so that quantitative and qualitative analysis is possible. In [10] is shown, that essential states are integer states.

Definition 3.1 *The graph $RG(\Sigma, i)$ is called the reachability graph of the TWN Σ from initial state i iff its vertices are the reachable integer-states and its edges are defined by the triples (S, t, S') and (S, τ, S'), where $S \xrightarrow{t} S'$ or $S \xrightarrow{\tau} S'$, respectively.*

The reachable integer-states are those states from the state space, which have clocks that are (of enabled transitions) integers only and can be reached from the initial state S_0 through any number of transition firings or time durations. The reachability graph of the time workflow net Σ is the transition relation $\rightarrow$ restricted to its reachable integer-states. This graph is finite iff the set of the reachable markings of the time workflow net is finite. This set is finite, if the set of reachable markings of the skeleton net is finite.

Using the parametric description of transition sequence [7] minimal and maximal length of time of the execution sequence can be evaluated. The maximal and minimal length of time is an integer and it can be reached by firing in integer-states. Thus, when the TWN is bounded, a sequence with maximal/minimal length of time can be found for given source-state and sink-state.

For time workflow net above the reachability graph consists of following reachable integer-states:

$m_0 = (1, 0, 0, 0, 0, 0)$, $J_0 = (0, \sharp, \sharp, \sharp)^T$, $J_0^* = (10, \sharp, \sharp, \sharp)^T$,

$m_1 = (0, 1, 1, 0, 0, 0)$, $J_1 = (\sharp, 0, 0, \sharp)^T$, $J_1^* = (\sharp, 2, 2, \sharp)^T$, $J_1^{**} = (\sharp, 5, 5, \sharp)^T$,

$m_2 = (0, 0, 1, 1, 0, 0)$, $J_2 = (\sharp, \sharp, 2, \sharp)^T$, $J_2^* = (\sharp, \sharp, 3, \sharp)^T$, $J_2^{**} = (\sharp, \sharp, 6, \sharp)^T$,

$m_3 = (0, 0, 0, 1, 1, 0)$, $J_3 = (\sharp, \sharp, \sharp, 0)^T$, $J_3^* = (\sharp, \sharp, \sharp, 4)^T$, $J_3^{**} = (\sharp, \sharp, \sharp, 8)^T$,

$m_4 = (0, 0, 0, 0, 0, 1)$, $J_4 = (\sharp, \sharp, \sharp, \sharp)^T$,

$S_8 = (m_0, J_0^*)$, $S_9 = (m_1, J_1^{**})$, $S_{10} = (m_2, J_2^{**})$, $S_{11} = (m_3, J_3^{**})$,

$$\underbrace{(m_0, J_0)}_{S_0} \xrightarrow{t_1} \underbrace{(m_1, J_1)}_{S_1} \xrightarrow{2} \underbrace{(m_1, J_1^*)}_{S_2} \xrightarrow{t_2} \underbrace{(m_2, J_2)}_{S_3} \xrightarrow{1} \underbrace{(m_2, J_2^*)}_{S_4} \xrightarrow{t_3}$$
$$\underbrace{(m_3, J_3)}_{S_5} \xrightarrow{4} \underbrace{(m_3, J_3^*)}_{S_6} \xrightarrow{t_4} \underbrace{(m_4, J_4)}_{S_7}$$

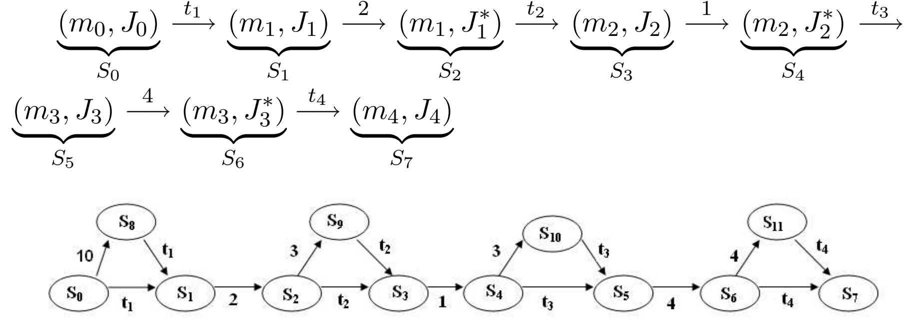

Figure 2. Reachability graph

4 Determining BCET and WCET

Methods from graph theory may be applied to determine best-case and worst-case execution times while constructing the reachability graph.

Determining the best-case execution leads directly to the well-known problem of the shortest path. Since reachability graph has nonnegative times only, all common shortest path algorithms are applicable, e.g. Dijkstra's algorithm or Bellman-Ford algorithm [4].

Determining worst-case execution is similar to the critical path problem, sometimes called longest path problem.

The problem can be formulated as follows:

For a given directed weighted graph $RG = (V, E)$, find the lengh l of a longest path from a source vertex v_s to a goal vertex v_d so that v_d is contained by most ones as a last vertex.

Actually we mean, that the length l is infinite, if there exists a cycle reachable starting on v_s before passing v_d and, otherwise, l is the sum of the weights of the longest path.

To determine worst-case execution, we propose the following algorithm A_1:

1. Remove from the graph RG all edges (v_d, v_j), i.e. all edges that are directed from v_d.

2. For each edge $(v_i, v_j) \in RG$ with the weight w_i assign a new weight $w_i^- = -w_i$. Edges, labeled by transitions names, obtain the weight 0.

3. Procedure Bellman-Ford (V, E, s) ** s is a source node
   for each $v \in V(RG)$ do

   $d[v] \longleftarrow \infty$

   $p[v] \longleftarrow NIL$ ** $p[v]$ is predecessor node of v

   $d[s] \longleftarrow 0$

   for $i \longleftarrow 1$ to $n-1$ ** $n = |V(RG)|$

   for each edge $(u, v) \in E(RG)$ do

   if $d[u] + w(u, v) < d[v]$ then

   $p[v] \longleftarrow u$

   $d[v] \longleftarrow d[u] + w(u, v)$

   for each edge $(u, v) \in E(RG)$ do

   if $d[u] + w(u, v) < d[v]$ then ** check for negative weight cycles

   return $FALSE$

   return $TRUE$

If algorithm returns false, then l is infinite. Otherwise, $l = -d(v_d)$. The complexity of this algorithm is dominated by the complexity of the Bellman-Ford algorithm, i.e. it is $O(|V| \cdot |E|)$. A correctness of A_1 is easy to be seen after the removal of all output edges from goal vertex. No path is possible, which contains goal vertex at another position than a final vertex. Obviously, the shortest path in the negative weighted graph corresponds to the longest path in the initial graph.

We computed a worst-case and a best-case execution times with the help of INA tool [11] for the example from the Figure 1. Our algorithm identified 11 states. The worst-case execution time of the service from source node i to target node o is 24 units of time. A maximal path is $i \Longrightarrow p_2 \Longrightarrow p_4 \Longrightarrow o$. The best-case execution time of service from source node i to target node o is 6 units of time. A minimal path is $i \Longrightarrow p_1 \Longrightarrow p_3 \Longrightarrow o$.

5 Conclusions

In this paper we presented a new approach aimed at determining best-case and worst-case times between two states in a TWN in polynomial time and demonstrated the application of our method for a certain time workflow net.

References

[1] B. Berthomieu, *Modeling and Verification of Time Dependent Systems Using Time Petri Nets*, In Advances of petri Nets 1984, vol 17 , No 3 of IEEE Trans. On Software Eng. 1991, 259–273.

[2] B. Berthomieu, *An Enumerative Approach for Analyzing Time*, In Proceedings IFIP 1983, R:E:A:Mason(ed), North-Holland, 1983, 41–47.

[3] I. Camerzan, *On soundness for time workflow nets*, Computer Sciece Journal of Moldova, volume 15, nr. 1(43), Chisinau, 2007, 74–87.

[4] T.H. Cormen, C.E. Leisserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, second edn. MIT Press, 2001.

[5] T. Jucan, F. Tiplea, *Petri Nets – theory and practice*, Academia Romana, Bucuresti, 1999. (in Romanian)

[6] T. Jucan, O. Prisecaru, I. Camerzan, *Time Interval Workflow Nets*, Scientific Annals of the "Al. I. Cuza" University, Computer Science Section, Tome XV, Iasi, 2005, 77–92.

[7] P. Merlin, *A study of the recoverability of computer system*, Ph. D. thesis, Dep. Computing Science, University California, Irvine, 1974.

[8] L. Popova-Zeugmann, *On Time Invariance in Time Petri Nets*, In Informatik-Bericht Nr. 36 der Institute pur Informatik der Humboldt-Univ. Zu Berlin, Oct. 1994

[9] L. Popova-Zeugmann, *On Liveness and Boundness in Time Petri Nets*

[10] L. Popova-Zeugmann, *On Parametrical Sequences in Time Petri Nets*, Proceedings of the CSP 97 Workshop, Warsaw(1997), 105–111.

[11] P.H. Starke, *INA – Integrated Net Analyzer*, Berlin, 1997.

Inga Camerzan,

Received March 4, 2010

State University of Tiraspol
E–mail: *caminga*2002@*yahoo.com*

Computer Science Journal of Moldova, vol.18, no.1(52), 2010

Determining best-case and worst-case times of unknown paths in time workflow nets

Inga Camerzan

Abstract

In this paper we present a method aimed for determining best-case and worst-case times between two arbitrary states in a time workflow net. The method uses a discrete subset of the state space of the time workflow net and archives the results, which are integers.

1 Introduction

Time workflow nets (TWN) were developed to provide a suitable method to model, simulate, and analyze the behavior of time dependent systems, business processes. Best-case execution times (BCET) and worst-case execution times (WCET) are a necessary step in the development and validation process for hard real-time systems. Real-time systems need to satisfy stringent timing constraints, induced by the systems aims. We consider time workflow nets, those only which allow the modelling of time delay and deadlines for the execution of activities in the workflow process. This paper will mainly focus on modelling the control flow perspective. Thus the perspective workflow process definitions are formulated in order to specify which tasks need to be executed and in what order. If a worst-case input for the task were known, then there are reliable guarantees that processes always terminate. However the worst-case input is not known and it is hard to be determined.

In a workflow management system there is a delay beetween the moment when an activity becomes enabled and the moment when the

©2010 by I. Camerzan

activity is executed for a certain resource. For each transition t in the time workflow net there is a static interval $[a_t, b_t]$ associated to it. The times a_t, b_t are relative to the moment at which t was last enabled. Assuming that t was last enabled at the global time τ, then t may fire only during the interval $[a_t + \tau, b_t + \tau]$ and must fire the latest at the time $b_t + \tau$. This is a method of incorporating time into Petri Nets, introduced by Merlin [7] and studied in [1, 2, 8, 9, 10].

One of the most important problems, in the above mentioned nets, is to determine the deadlines for a sequence of system processes, i.e., to compare the longest duration of a transition sequence with a given limit. Such considerations are important for determining the best-case execution times and the worst-case execution times.

As a rule, the method used to estimate execution times bounds in practice consist in measuring the *end-to-end* execution time of the task for a subset of the possible executions, called test cases. This determines the minimal observed and the maximal observed execution times. Generally speaking, through these methods we are able to obtain an overestimation of the BCET and underestimation of the WCET, therefore we cannot consider them safe. Our aim is to propose an algorithm able to estimate the execution times in a safe way.

This paper is organized as follows: In Section 2 we introduce the basic notions and definitions for time workflow nets; In Section 3 we present the reachability graph of the time workflow nets, which can be used for computing the shortest and the longest path between two arbitrary states in the TWN; In the last section the algorithm for execution times estimation is proposed. The way it functions is illustrated by an example.

2 Basic notions and definitions

Definition 2.1 *A Petri net PN =(P, T, F, W) is a Workflow net iff:*

1. *PN has two additional places i and o, "start" place i, "destination" place o.*

2. *If we add a transition t^* to PN which connects o with i then the resulting Petri net is strongly connected.*

There are distinct methods of incorporating time in Petri nets: associating time delay to transition, associating time delay to places, associating time delay with arcs, associating time delays or time intervals to different types of objects of the net, associating stochastic time. Further we consider only Petri nets [5] which have deterministic time associated to transitions, in the form of time intervals, defined by Merlin [7] in 1972 and studied in [1, 2, 8, 9, 10].

We define a time workflow net in following way:

Definition 2.2 *A Time workflow net is a tuple Σ=(P, T, F, W, I) where PN=(P, T, F, W) is the workflow net (also called skeleton net), I: T$\rightarrow$ $Q_0^+ \times Q_0^+$ is a time function which associates timed intervals with transitions and $I_1(t) \leq I_2(t)$, where I(t) = $(I_1(t), I_2(t))$, for each transition t$\in$T.*

A global clock is associated with the time workflow net, which begins to work as soon as the first token appears in the net. After time association, the workflow net will work in the following way: from the moment when a transition t is enabled, the tokens from the input locations are stored for $I_2(t) - I_1(t)$ time units, and after this time elapses the transition fires putting tokens in their output places. For transitions in conflict, the first transition that fires is the one which has the latest time interval smaller.

For the definition of a state and of a change of state of a net we will follow [3, 6]:

Definition 2.3 *Let Σ=(P, T, F, W, I) be a time workflow net and J: T$\rightarrow$ $Q_0^+ \cup \{\sharp\}$. S=(m, J) is the state of the net Σ iff:*

1. *m is a marking in skeleton net,*

2. *if $t \in T$ and $t^- \leq m$, then $J(t) \leq I_2(t)$,*

3. *if $t \in T$ and $t^- \not\leq m$, then $J(t) = \sharp$,*

where $t^{-}(p) = W(p,t)$ *is arc weight from place* p *to transition* t*.*

We understand the notion of state in the following way. Let $S = (m, J)$ be a state. Each transition t in the net has a watch. The watch doesn't work ($J(t) = \sharp$) at the marking m if t is disabled at m. If t is enabled at m, then the watch of t shows the time $J(t)$ that has elapsed since t was last enabled.

Let $\Sigma = (P, T, F, W, I)$ be a time workflow net. The state $S_0 := (i, J_0)$ with i the initial marking of the time workflow net (the marking which has a single token in place i) and $J_0(t)= \begin{cases} 0, & \text{if} \quad t^{-} \leq m, \\ \sharp, & \text{if} \quad t^{-} \not\leq m \end{cases}$ is considered to be the initial state of the time workflow net. The states in a time workflow net can change due to transition firings or time elapsing.

Definition 2.4 *A transition t is enabled at the state S=(m, J), denoted by* $S \rightarrow$*, iff*

1. $t^{-} \leq m$;
2. $I_1(t) \leq J(t)$.

Thus, a transition is enabled in a time workflow net Σ, if t is enabled in the skeleton net (the timeless net) and the time specifications are satisfied, i.e t has been enabled for a sufficient amount of time. The resulting state is defined as follows:

Definition 2.5 *A transition t enabled at the state S=(m, J), will fire inducing state* $S' = (m', J')$*, denoted by* $S \rightarrow S'$ *defined thus:*

1. $m\prime = m + \triangle t$;
2. $$J'(t) = \begin{cases} \sharp, & t^{-} \not\leq m', \\ J(t), & t^{-} \leq m \wedge t^{-} \leq m\prime \wedge F_t \cap F'_t = 0, \\ 0, & \textit{otherwise}, \end{cases}$$

   where $\triangle t = W(t,p) - W(p,t)$.

The resulting state (m', J') has a marking m' which results by firing of transition t in the skeleton net and a time vector J'. The values of the vector for the not enabled transitions in the marking are undefined $\sharp$. If a transition was enabled in the old marking, and it is still enabled in the new marking, and it is not in conflict with the just fired transition, then it keeps the values of its local clock $J'(t) = J(t)$. Otherwise, if a transition has just become enabled in the new marking, then its local clock $J'(t) = 0$.

Definition 2.6 *Let $\Sigma = (P, T, F, W, I)$ be a time workflow net. The state $S = (m, J)$ changes into the state $S' = (m', J')$ by the time duration $\tau \in Q$, denoted by $S \xrightarrow{\tau} S'$ iff $m' = m$ and the time duration τ is possible i.e. for any $t \in T$ with $J(t) \neq \sharp$, we have $J(t) + \tau \leq I_2(t))$ and*
$$J'(t) = \begin{cases} J(t) + \tau, & \text{if} \quad t^- \leq m, \\ \sharp, & \text{if} \quad t^- \not\leq m. \end{cases}$$

The sequence of transitions and time durations $\sigma = \tau_0, t_0, \tau_1, t_1, \ldots \tau_{n-1}, t_{n-1}$ is executable in the net Σ iff there exist the states $S_0, S'_0, S_1, S'_1, \ldots S'_{n-1}, S_n$ so that: $S_0 \xrightarrow{\tau_0} S'_0 \xrightarrow{t_0} S_1, \ldots S_{n-1} \xrightarrow{\tau_{n-1}} S'_{n-1} \xrightarrow{t_{n-1}} S_n$. That sequence shortly can also be noted by $S_0[\sigma\rangle S_n$. The transition-time sequence σ is called an execution sequence in the net Σ. State S' is reachable from the state S if there is a transition-time sequence σ so that $S[\sigma\rangle S'$. $RS(\Sigma, S_0)$ or $[S_0\rangle$ denotes the set of all reachable states of a net Σ and $R_\Sigma(S)$ denotes the set of all reachable states from the state S.

In Figure 1, the initial state is ((1,0,0,0,0,0)(0,$\sharp$, $\sharp$, $\sharp$)). Marking (1,0,0,0,0,0) is the initial marking of the skeleton net. The time vector has value 0 corresponding to transition t_1 enabled in the initial marking and the values $\sharp$ for the rest of the transitions which are disabled. Since t_1 is enabled at marking i and the time constraints are also satisfied, transition t_1 can fire in the initial state of the net Σ. The resulting state is $S_0 \xrightarrow{t_1} S_1 = ((0, 1, 1, 0, 0, 0)(\sharp, 0, 0, \sharp))$. The state S_1 has the marking $m_1 = (0, 1, 1, 0, 0, 0)$ and the time vector $J_1 = (\sharp, 0, 0, \sharp)$ with value 0 corresponding to the transitions t_2 and t_3, enabled in the marking m_1 in the skeleton net, and value $\sharp$ corresponding to the disabled transitions in the skeleton net. In the case of the time con-

straints, i.e. in the conditions $I_1(t_2) \geq J_1(t_2)$ and $I_1(t_3) \geq J_1(t_3)$ in state S_1 the transitions t_2 and t_3 cannot fire. So, state S_1 can change into another state in the net Σ only by time elapsing. For instance, the following state change is possible in the net Σ: $S_1 \stackrel{3.7}{\rightarrow} S_2$, where $S_2 = ((0,1,1,0,0,0)(\sharp, 3.7, 3.7, \sharp))$. We notice that the marking of the skeleton net remains unchanged, and results in a new time vector J_2 with updated time values for the transitions t_2 and t_3. The time duration 6.1 is not possible in state S_1 because the transition t_2, which, if enabled, must fire in 5 units of time. Now, both t_2 and t_3 can fire at state S_2. If t_2 fires, then we have $S_2 \stackrel{t_2}{\rightarrow} S_3 = ((0,0,1,1,0,0)(\sharp, \sharp, 3.7, \sharp))$. Now, the time duration 1.3 is possible at state S_3 and a new state appears: $S_3 \stackrel{1.3}{\rightarrow} S_4 = ((0,0,1,1,0,0)(\sharp, \sharp, 5, \sharp))$. Thus the state-transition-times sequence results in: $S_0 \stackrel{t_1}{\rightarrow} S_1 \stackrel{3.7}{\rightarrow} S_2 \stackrel{t_2}{\rightarrow} S_3 \stackrel{1.3}{\rightarrow} S_4$. The sequence $t_1, 3.7, t_2, 1.3$ is an execution sequence in the net Σ.

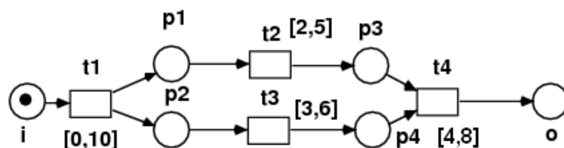

Figure 1. A time workflow net

3 Reachability graph of the time workflow net

The state space of a time workflow net is the set of all reachable states of the skeleton net, starting from i. Because of the markings, reachable in the net, this set is discrete, and it may be infinite. On the other hand, this set may be infinite because of the time of transitions. Thus, the set of all reachable states for a fix marking is infinite (and densely ordered) in general. Nevertheless, it is possible to pick up some "essential" states

only, so that quantitative and qualitative analysis is possible. In [10] is shown, that essential states are integer states.

Definition 3.1 *The graph $RG(\Sigma, i)$ is called the reachability graph of the TWN Σ from initial state i iff its vertices are the reachable integer-states and its edges are defined by the triples (S, t, S') and (S, τ, S'), where $S \xrightarrow{t} S'$ or $S \xrightarrow{\tau} S'$, respectively.*

The reachable integer-states are those states from the state space, which have clocks that are (of enabled transitions) integers only and can be reached from the initial state S_0 through any number of transition firings or time durations. The reachability graph of the time workflow net Σ is the transition relation $\rightarrow$ restricted to its reachable integer-states. This graph is finite iff the set of the reachable markings of the time workflow net is finite. This set is finite, if the set of reachable markings of the skeleton net is finite.

Using the parametric description of transition sequence [7] minimal and maximal length of time of the execution sequence can be evaluated. The maximal and minimal length of time is an integer and it can be reached by firing in integer-states. Thus, when the TWN is bounded, a sequence with maximal/minimal length of time can be found for given source-state and sink-state.

For time workflow net above the reachability graph consists of following reachable integer-states:

$m_0 = (1,0,0,0,0,0)$, $J_0 = (0,\sharp,\sharp,\sharp)^T$, $J_0^* = (10,\sharp,\sharp,\sharp)^T$,

$m_1 = (0,1,1,0,0,0)$, $J_1 = (\sharp,0,0,\sharp)^T$, $J_1^* = (\sharp,2,2,\sharp)^T$, $J_1^{**} = (\sharp,5,5,\sharp)^T$,

$m_2 = (0,0,1,1,0,0)$, $J_2 = (\sharp,\sharp,2,\sharp)^T$, $J_2^* = (\sharp,\sharp,3,\sharp)^T$, $J_2^{**} = (\sharp,\sharp,6,\sharp)^T$,

$m_3 = (0,0,0,1,1,0)$, $J_3 = (\sharp,\sharp,\sharp,0)^T$, $J_3^* = (\sharp,\sharp,\sharp,4)^T$, $J_3^{**} = (\sharp,\sharp,\sharp,8)^T$,

$m_4 = (0,0,0,0,0,1)$, $J_4 = (\sharp,\sharp,\sharp,\sharp)^T$,

$S_8 = (m_0, J_0^*)$, $S_9 = (m_1, J_1^{**})$, $S_{10} = (m_2, J_2^{**})$, $S_{11} = (m_3, J_3^{**})$,

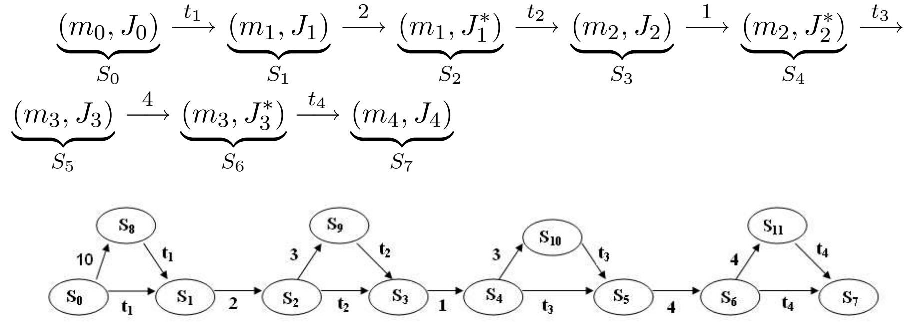

Figure 2. Reachability graph

4 Determining BCET and WCET

Methods from graph theory may be applied to determine best-case and worst-case execution times while constructing the reachability graph.

Determining the best-case execution leads directly to the well-known problem of the shortest path. Since reachability graph has nonnegative times only, all common shortest path algorithms are applicable, e.g. Dijkstra's algorithm or Bellman-Ford algorithm [4].

Determining worst-case execution is similar to the critical path problem, sometimes called longest path problem.

The problem can be formulated as follows:

For a given directed weighted graph $RG = (V, E)$, find the lengh l of a longest path from a source vertex v_s to a goal vertex v_d so that v_d is contained by most ones as a last vertex.

Actually we mean, that the length l is infinite, if there exists a cycle reachable starting on v_s before passing v_d and, otherwise, l is the sum of the weights of the longest path.

To determine worst-case execution, we propose the following algorithm A_1:

1. Remove from the graph RG all edges (v_d, v_j), i.e. all edges that are directed from v_d.

2. For each edge $(v_i, v_j) \in RG$ with the weight w_i assign a new weight $w_i^- = -w_i$. Edges, labeled by transitions names, obtain the weight 0.

3. Procedure Bellman-Ford (V, E, s) ** s is a source node
for each $v \in V(RG)$ do
    $d[v] \longleftarrow \infty$
    $p[v] \longleftarrow NIL$ ** $p[v]$ is predecessor node of v
$d[s] \longleftarrow 0$
for $i \longleftarrow 1$ to $n-1$ ** $n = |V(RG)|$
    for each edge $(u, v) \in E(RG)$ do
        if $d[u] + w(u, v) < d[v]$ then
            $p[v] \longleftarrow u$
            $d[v] \longleftarrow d[u] + w(u, v)$
for each edge $(u, v) \in E(RG)$ do
    if $d[u] + w(u, v) < d[v]$ then ** check for negative weight cycles
        return $FALSE$
return $TRUE$

If algorithm returns false, then l is infinite. Otherwise, $l = -d(v_d)$. The complexity of this algorithm is dominated by the complexity of the Bellman-Ford algorithm, i.e. it is $O(|V| \cdot |E|)$. A correctness of A_1 is easy to be seen after the removal of all output edges from goal vertex. No path is possible, which contains goal vertex at another position than a final vertex. Obviously, the shortest path in the negative weighted graph corresponds to the longest path in the initial graph.

We computed a worst-case and a best-case execution times with the help of INA tool [11] for the example from the Figure 1. Our algorithm identified 11 states. The worst-case execution time of the service from source node i to target node o is 24 units of time. A maximal path is $i \Longrightarrow p_2 \Longrightarrow p_4 \Longrightarrow o$. The best-case execution time of service from source node i to target node o is 6 units of time. A minimal path is $i \Longrightarrow p_1 \Longrightarrow p_3 \Longrightarrow o$.

5 Conclusions

In this paper we presented a new approach aimed at determining best-case and worst-case times between two states in a TWN in polynomial time and demonstrated the application of our method for a certain time workflow net.

References

[1] B. Berthomieu, *Modeling and Verification of Time Dependent Systems Using Time Petri Nets*, In Advances of petri Nets 1984, vol 17 , No 3 of IEEE Trans. On Software Eng. 1991, 259–273.

[2] B. Berthomieu, *An Enumerative Approach for Analyzing Time*, In Proceedings IFIP 1983, R:E:A:Mason(ed), North-Holland, 1983, 41–47.

[3] I. Camerzan, *On soundness for time workflow nets*, Computer Sciece Journal of Moldova, volume 15, nr. 1(43), Chisinau, 2007, 74–87.

[4] T.H. Cormen, C.E. Leisserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, second edn. MIT Press, 2001.

[5] T. Jucan, F. Tiplea, *Petri Nets – theory and practice*, Academia Romana, Bucuresti, 1999. (in Romanian)

[6] T. Jucan, O. Prisecaru, I. Camerzan, *Time Interval Workflow Nets*, Scientific Annals of the "Al. I. Cuza" University, Computer Science Section, Tome XV, Iasi, 2005, 77–92.

[7] P. Merlin, *A study of the recoverability of computer system*, Ph. D. thesis, Dep. Computing Science, University California, Irvine, 1974.

[8] L. Popova-Zeugmann, *On Time Invariance in Time Petri Nets*, In Informatik-Bericht Nr. 36 der Institute pur Informatik der Humboldt-Univ. Zu Berlin, Oct. 1994

[9] L. Popova-Zeugmann, *On Liveness and Boundness in Time Petri Nets*

[10] L. Popova-Zeugmann, *On Parametrical Sequences in Time Petri Nets*, Proceedings of the CSP 97 Workshop, Warsaw(1997), 105–111.

[11] P.H. Starke, *INA – Integrated Net Analyzer*, Berlin, 1997.

Inga Camerzan,

Received March 4, 2010

State University of Tiraspol
E–mail: *caminga2002@yahoo.com*

Computer Science Journal of Moldova, vol.18, no.1(52), 2010

Determination of inflexional group using P systems

Svetlana Cojocaru, Elena Boian

Abstract

The aim of this article is to describe the process of determining the inflectional group using P systems with replications. In this process firstly the sets of endings of the same length are constructed, to which inflectional models are put into correspondence. Based on these endings the inflectional model for arbitrary word is determined.

Introduction

Natural Language Processing (NLP) is one of the areas that requires high performance computing. In order to solve problems in this domain, it needs to operate with resources containing millions of entries, so there is a logical temptation to apply approaches based on parallelism. Such attempts have been undertaken since the '80s, and have an ample development in coming decades [1, 2].

An important direction in NLP is creation of computational linguistic resources. In [3] we presented the solution of one of the problems that contributes to resources enrichment: automatic inflection. The proposed solution was applied for the Romanian language. Thanks to this process, we received over 40 word forms for each verb, 24 new words for each adjective, etc. Our solution was based on the use of P systems with replication and applied for the case when inflectional model is known a priori. As a rule, there are classification dictionaries for high inflectional languages, where these models are established. In the case of the Romanian language we use the dictionary [4], containing

©2010 by S. Cojocaru, E. Boian

the inflectional models defined for about 30,000 words. In this paper it is shown how the inflection model for an arbitrary word can be determined. Knowing this information we are able to perform automatically the inflectional process. Analogously to [3, 5] for illustration we will use examples from the Romanian language, but the proposed method can be applied also to other natural languages with similar inflectional mechanisms.

From the beginning we must note that in general case an algorithmic solution to this problem is not possible. The first obstacle is the determining of part of speech: there are a lot of examples of homonyms which denote different parts of speech (e.g.: *abate* – masculine noun (engl. abbot) and verb (engl. to divert).

Let us restrict the formulation of the problem: is it possible to ascertain inflectional model knowing the part of speech? The answer is negative in this case too.

For confirmation there is a list of examples which prove that we can not determine the inflectional model without invoking the etymological or phonetic information.

This assertion can be illustrated by analysing feminine noun *masă*. Following the meaning of the furniture object we form the plural *mese* (engl. *tables*) using the inflectional model with vowel alternation "$a \rightarrow e$". But if the meaning is *mass* [6], plural *mase* will be produced without vowel alternation. The origin of this phenomenon is etymological: the first case is of the Latin origin *mensa*, and the second – the French word *masse*) [6].

But the problem can be tackled in other mode: we can establish certain criteria that allow us to conclude in the term of word structure analysis, if it is possible to determine the inflectional model or not, and in the case of "yes", to determine which is namely the respective model. Otherwise speaking, we try to formulate the criterion under which we can say that inflectional process is performed automatically and can indicate the appropriate inflectional model.

1 Problem of inflectional model determination for an arbitrary word

The specific character of the investigated area (natural language) is reflected in the fact that many of the objects and concepts it operates, cannot be the subject to strict formalisation. Therefore we will try to distinguish certain classes in which this formalisation is possible.

So let the word-lemma be known in its graphical representation (i.e., the data without phonetic, etymological notes, etc.). Also let the part of speech be known, and for nouns – the gender. We divide the words into three categories: irregular, absolutely regular and partially regular.

For all parts of speech the fact of belonging to the *irregular* class is determined by the fact of their belonging to a set of words known a priori. To simplify the statement we exclude from the examination the set of irregular words, their presence (or absence) does not affect the generality of the algorithm. We consider *absolutely regular* words, to which a single inflectional model corresponds and we note by A the set of their endings. We call *partially regular* those words, to which two or more inflectional models correspond. The set of their endings we denote by P. In the following we establish the criteria for belonging to these two classes (and corresponding inflectional models). The algorithm described in [7] determines these criteria in sequential mode. We propose to obtain these criteria in parallel mode using massive parallelism which is characteristic to membrane P systems [8].

Inflectional group determination will be made in two steps:

- building the sets of endings of the same length, to which the inflectional models are being put into correspondence;

- determination of the inflectional group in correspondence with the built sets of endings.

2 Construction of sets of endings

Let L be the set of all words of a language. We come from the assumption (valid for majority of natural languages) that there is a classifica-

tion dictionary $D \subseteq L$, so that to any $\omega \in D$ it puts into correspondence an inflectional model ν, where ν is a positive integer. We will present dictionary D as a union of words classified by parts of speech (and gender, for nouns), $D = \cup(C)_{i=1}^{5}$, where C is one of the sets of words, which belong to the open classes [3] (for Romanian these are the adjectives, verbs, nouns: masculine, feminine, neuter). For each C_i the dictionary D puts into correspondence the finite set of inflectional models $N_i = \{\nu_1, \ldots, \nu_{n_k}\}$, such that for $\forall\omega \in C_i$ there is at least a $\nu \in N_i$. We will separately operate with each of these classes.

Let C be one of these classes. The idea of algorithm to build the sets of endings is the following. For each word $\omega \in C$, to which the inflectional model $\nu_m \in N$ corresponds (N is the set of integers of inflectional models for words in C), there are built the endings with decreasing lengths from $|\omega|$ to 1. The pairs (γ_i, ν_m) are formed, where γ_i is a substring of length i of the word ω, $(1 \leq i \leq |\omega|)$. The pairs, constructed thus, are compared and filtered. The filtration process is carried out in the following way: out of each two elements (γ_i, ν_m), (η_i, ν_n), we keep only one, if $\gamma_i = \eta_i$ and $\nu_m = \nu_n$, where γ_i is a substring of length i of the word $|\omega|$, and η_i is a substring of length i of the word ψ (i.e. only noncoincident pairs are kept).

If for all the pairs in which $\gamma_i \neq \eta_i$ the equality $\nu_m = \nu_n$ takes place, then the pairs (γ_i, ν_m) and (η_i, ν_n) are elements of the set A of the endings corresponding to absolutely regular words.

If $\gamma_i = \eta_i$ and $\nu_m \neq \nu_n$, then the ending η_i indicates a substring of the word ψ partially regular from the set P, to which several inflectional models $\nu_m, \nu_n, \ldots$ correspond.

We denote by $L_{max} = max\{|\omega|\}$, $\omega \in C$, maximum of the length of words in C.

This algorithm can be realised using the following membrane system Π_1.

$$\Pi_1 = (O, \Sigma, \mu, R_0, R_i, A, P),$$

where $i = 1, ..., L_{max}$,

O is the alphabet of symbols, λ is an empty element, $\lambda \in O$.

$\Sigma \subseteq O$ – Romanian alphabet,

μ – membrane structure which is defined as:

$\mu = [_0\, [_A\,]_A\, [_{L_{max}}\, [_{L_{max-1}} \ldots [_1\,]_1 \ldots]_{L_{max-1}}\,]_{L_{max}}\, [_P\,]_P\,]_0,$

R_0: $\{(\omega, \nu_m) \to (\omega, \nu_m)^1 || \ldots || (\omega, \nu_m)^i || \ldots || (\omega, \nu_m)^{|\omega|}$, $1 \leq i \leq |\omega|$, $\omega \in C$, $|\omega| \leq L_{max}$,

$(\omega, \nu_m)^i \to ((\omega, \nu_m)^i, in_i)$, for $i = 1, \ldots, |\omega|$, $|\omega| \leq L_{max}\}$;

R_i: $\{(\omega, \nu_m)^i \to (\gamma_i, \nu_m)$, for $i = 1, \ldots, |\omega|$, $|\omega| \leq L_{max}$, where $\omega = \omega_l \gamma_i$, $|\gamma_i| = i$, $i = 1 \ldots |\omega|$,

$(\gamma_i, \nu_m) \to \lambda\, |_{\exists(\eta_i, \nu_n): \gamma_i = \eta_i \& \nu_m = \nu_n}$, $i = 1, \ldots, |\omega|$, $|\omega| \leq L_{max}$, $\psi = \psi_l \eta_i$, $|\eta_i| = i$, $i = 1 \ldots, |\psi|$, $|\psi| \leq L_{max}$, $\omega, \psi \in C$,

$(\gamma_i, \nu_m) \to (\gamma_i, \nu_m, \nu_n, \ldots,$
$\ldots, \nu_{n_k})\, |_{\exists(\eta_i^1, \nu_{n_1}), (\eta_i^2, \nu_{n_2}), \ldots, (\eta_i^{n_k}, \nu_{n_k}): \gamma_i = \eta_i^1 = \eta_i^2 = \ldots = \eta_i^{n_k} \& \nu_m \neq \nu_n}$,
$i = 1, \ldots, |\omega|$, $|\omega| \leq L_{max}$, $i = 1 \ldots, |\psi|$, $|\psi| \leq L_{max}$, for $\forall k = 1, \ldots, j$,

$(\gamma_i, \nu_m) \to ((\gamma_i, \nu_m), out_A)\, |_{\forall \eta_i: \eta_i \neq \gamma_i}$, $i = 1, \ldots, |\omega|$, $|\omega| \leq L_{max}$, $i = 1 \ldots, |\psi|$, $|\psi| \leq L_{max}$,

$(\gamma_i, \nu_m, \nu_{n_1}, \ldots, \nu_{n_k}) \to$
$((\gamma_i, \nu_m, \nu_{n_1}, \ldots, \nu_{n_k}), out_P)\, |_{\exists \eta_i^1, \eta_i^2, \ldots, \eta_i^{n_k}: \eta_i^1 \neq \gamma_i \& \eta_i^2 \neq \gamma_i \ldots \& \eta_i^{n_k} \neq \gamma_i}$,
$i = 1, \ldots, |\omega|$, $|\omega| \leq L_{max}$, $i = 1 \ldots, |\psi|$, $|\psi| \leq L_{max}$, for $\forall k = 1, \ldots, j\}$.

The membrane A contains objects of type (γ_i, ν_m).

The membrane P contains objects of type $(\gamma_i, \nu_m, \nu_{n_1}, \ldots, \nu_{n_k})$.

The rule R_0 indicates the replication of objects $(\omega, \nu_m)^i$ for $|\omega|$ times. Each object $(\omega, \nu_m)^i$ is transferred into the region bounded by the membrane i ($i = 1, \ldots |\omega|$).

The rule R_i indicates the following:

– truncation of the word ω keeping the ending γ_i of the length i,

– elimination of the pair (γ_i, ν_m) from the region i in case if there exists the duplicate pair. When such duplicate pair does not exist, the remained object is transferred to the membrane A,

– in case if the same ending of length i has in the capacity of pair different numbers of the inflectional models, then the object of the type $(\gamma_i, \nu_m, \nu_{n_1}, \ldots, \nu_{n_k})$ is formed, which is transferred into the membrane P.

Finally in the membrane 0 two resulting membranes A and P are obtained containing criteria for setting inflectional models for absolutely regular and partially regular words.

3 Determination of the inflectional group

We will determine the inflexional group for the word $\psi \in C$.

The idea of algorithm for the inflexional group determination is the following.

The substrings ξ_i $(1 \leq i \leq |\psi|)$ of the endings with decreasing length from $|\psi|$ to 1 of the word ψ are constructed. Initially we look for a completely regular model, comparing the ending ξ_i $(|\xi_i| = i)$ with the elements $(\gamma, \nu_m) \in A$ $(|\gamma_i| = i)$. If $\exists \gamma_i = \xi_i$, then ν_m is the inflectional model number. In case if we did not find an appropriate model in A, we look for it in P. If $\exists \gamma_i = \xi_i$ $(\gamma_i, \nu_{n_1}, \nu_{n_2}, \ldots, \nu_{n_k} \in P)$, the word ψ is partially regular and it has to inflect in correspondence with the inflexional models $\nu_{n_1}, \nu_{n_2}, \ldots, \nu_{n_k}$. In the case when $\xi_i \neq \gamma_i$ for $\forall \gamma_i$ from A and P the inflectional model can not be determined automatically and the intervention of user (the expert in linguistics) is needed.

This algorithm will be described by the following membrane system Π_2.

$$\Pi_2 = (O, \Sigma, \mu, A, P, R_0, R_i, 0),$$

where $i = 1, \ldots, L_{max}$,

O is the alphabet of symbols, including element "*false*" $\in \Sigma$,

$\Sigma \subseteq O$ – is the Romanian alphabet. The element "*false*" will signal about the case when the inflectional model is not found for the word to be inflected.

μ – membrane structure which is defined as the following:

$$\mu = [_0\ [_A]_A\ [_{L_{max}}\ [_{L_{max-1}}\ \dots\ [_1\ ]_1\ \dots\ ]_{L_{max-1}}\ ]_{L_{max}}\ [_P]_P\ ]_0,$$

A contains the set of pairs of type (γ_i, ν_m) own to absolutely regular words $(i = 1, \dots, |\omega|,\ |\omega| \le L_{max}, k = 1, \dots, |C|)$.

P contains the set of elements of type $(\gamma_i, \nu_m, \nu_{n_1}, \dots, \nu_{n_k})$ belonging to partially regular words $(i = 1, \dots, |\omega|,\ |\omega| \le L_{max}, k = 1, \dots, |C|)$.

R_0: $\{\ \psi \to (\psi, in_1)|| \dots ||(\psi, in_i)|| \dots ||(\psi, in_{|\psi|})$,

$\psi \to (\xi_1, in_1)|| \dots ||(\xi_i, in_i)|| \dots ||(\xi_{|\psi|}, in_{|\psi|})$, where $\psi = \omega_j \xi_i$, $|\xi_i| = i,\ i = 1, \dots, |\psi|\ \}$,

R_i: $\{\xi_i \to ((\psi, \nu_m), out_0)\ |_{\exists(\gamma_i, \nu_m) \in A : \xi_i = \gamma_i},\ i = 1, \dots |\psi|,\ i = 1, \dots, |\omega|$, $|\psi| \le L_{max},\ |\omega| \le L_{max}$,

$\xi_i \to ((\psi, \nu_m, \nu_{n_1}, \dots, \nu_{n_k}), out_0)\ |_{\exists(\gamma_i, \nu_m, \nu_n, \dots, \nu_{n_k}) \in P : \xi_i = \gamma_i}$, $i = 1, \dots |\psi|,\ i = 1, \dots, |\omega|,\ |\psi|,\ |\omega| \le L_{max}, k = 1, \dots, |C|$,

$\xi_i \to ((false), out_0)\ |_{\xi_i \ne \gamma_i}$, for $\forall(\gamma_i, \nu_m) \in A \vee (\gamma_i, \nu_{n_1}, \nu_{n_2}, \dots, \nu_{n_k}$ $\in P,\ i = 1, \dots |\psi|, k = 1, \dots, |C|\}$.

The rule R_0 indicates replication for $|\psi|$ times of the endings ξ_i $(\psi = \omega_j \xi_i, |\xi_i| = i)$ which are transferred then to membrane i $(1 \le i \le |\psi|)$.

The rule R_i forms objects of type $(\psi, \nu_m) \in A$ or $(\psi, \nu_m, \nu_{n_1}, \dots, \nu_{n_k})$ $\in P$ by which one or more inflectional models are assigned to the word ψ. The formed objects are transferred to the external membrane.

The appearance of the value "*false*" in the region 0 means that the number of the inflectional model is not found for the word $\psi \in C$. In this case the inflectional model can not be determined automatically.

4 Example of using membrane systems Π_1 and Π_2

Let D = { *(grup,1),(grup,2), (dulap,1), (cuvânt,2), (vânt,1), (tractor,3), (muzeu,41)*}.

Initially $A = \emptyset$, $P = \emptyset$ (see Fig.1).

We will take as C all the words from D, i.e.,

C = { *grup, dulap, cuvânt, vânt, tractor, muzeu*}

(in English: group, wardrobe, word, wind, tractor, museum).

$L_{max} = 7$; $N = \{1, 2, 3, 41\}$.

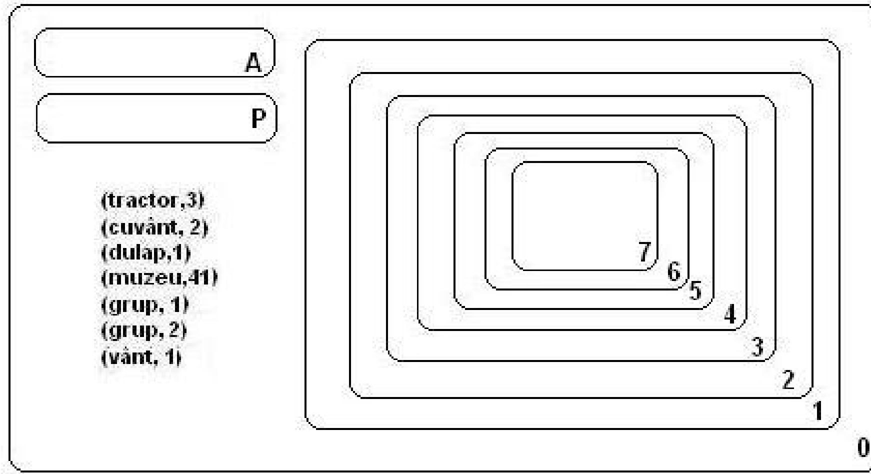

Figure 1. Initial state of membrane system Π_1.

The Figure 2 illustrates the process of building the sets of endings of the same length of words from C, to which the inflectional models N are being put into correspondence.

We obtained the sets A and P with the following components (see Fig.3):

A = { *(dulap,1), (ulap,1), (lap,1), (ap,1), (cuvânt,2), (uvânt,2), (tractor,3), (ractor,3), (actor,3), (ctor,3), (tor,3), (or,3), (r,3), (muzeu,41), (uzeu,41), (zeu,41), (eu,41), (u,41)* }.

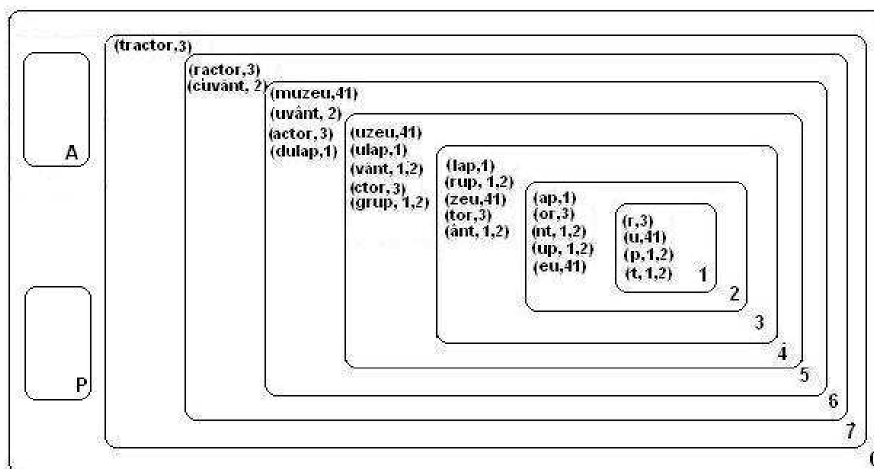

Figure 2. Building the sets of endings of the same length for words from C.

$P = \{$ *(grup,1,2), (rup,1,2), (up,1,2), (vânt,1,2), (ânt,1,2), (nt,1,2), (p,1,2), (t,1,2)* $\}$.

The Figure 3 illustrates the process of determining the inflectional group for the word *motor* (in English: engine) in relation to the built sets of endings.

We obtained that the word *motor* will be inflected using the inflectional model 3 (Fig. 4).

Conclusions

On the basis of the classification dictionary D for each part of speech C the sets A and P are constructed, which determine the inflexional models for absolutely and partially regular words. We mention the following.

1. In the general case we can renounce to build the respective sets for each part of speech apart, but in this way the number of partially

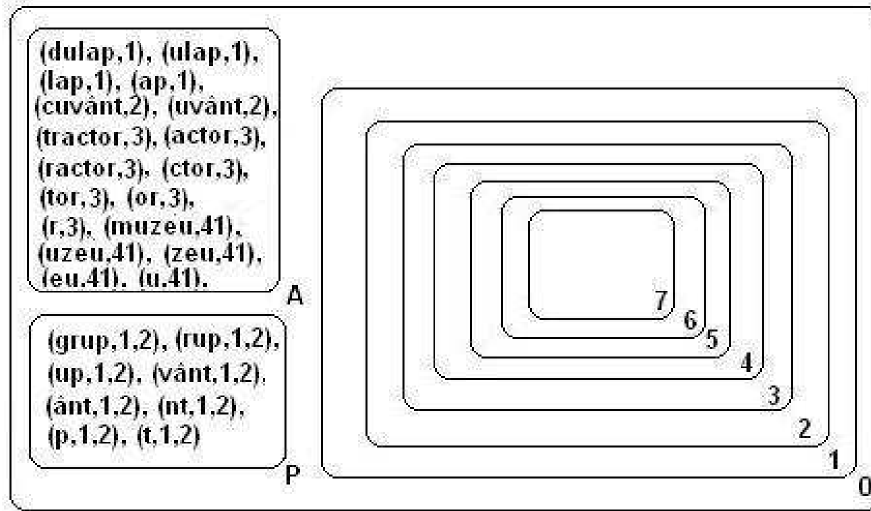

Figure 3. Obtaining the sets A and P.

regular words would be increased.

2. Both systems Π_1 and Π_2 could be reduced only to the membrane 0 with two membranes A and P contained in it. The inner membranes i were introduced in order to separate the comparing processes, which would allow us to simplify implementation of such mechanisms by a simulator.

3. The experiments, performed for a set of about 2000 base words, showed that in 97% of cases the inflectional model number can be determined using the systems described above. The 3%, for which the result was marked by "*false*", present in most cases the irregular words.

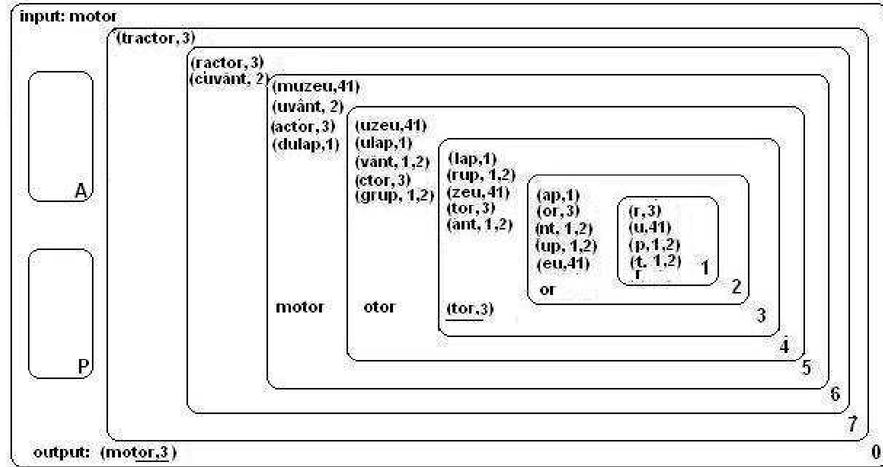

Figure 4. Example of using the membrane system Π_2

References

[1] Eiichiro Sumita, Kozo Oi, Osamu Furuse, Hitoshi Iida, Tetsuya Higuchi, Naotao Takahashi, Hiroaki Kitano. *Example-Based Machine Translation on Massively Parallel Processors.* Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France, 1993, vol.2, pp.1283–1289.

[2] Hiroaki Kitano. *Challenges of Massive Parallelism.* Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France, 1993, vol.1, pp.813–834.

[3] A.Alhazov, E. Boian, S. Cojocaru, Yi.Rogojin. *Modelling Inflexions in Romanian language by P Systems with String replications.* Computer Science Jourmal of Moldova, v.17, 2(50), 2009, pp.160–178.

[4] A.Lombard, C.Gâdei. *Dictionnaire morphologique de la langue roumaine.* Bucureşti, Editura Academiei, 1981, 232 p.

[5] S.Cojocaru. *Romanian lexicon: Tools, implementation, usage.* In Tufis, D., Andersen, P., eds.: Recent Advances in Romanian Language Technology. Volume I., Editura Academiei (1997), pp. 107–114 ISBN 973-027-00626-00.

[6] www.dexonline.ro

[7] S.Cojocaru. *The ascertainment of the inflexion models for Romanian.* Computer Science Journal of Moldova, vol.14, N1 (40), pp.103–112.

[8] Gh. Păun, *Membrane Computing. An Introduction*, Natural computing Series. ed. G. Rozenberg, Th. Back, A.E. Eiben, J.N. Kok, H.P. Spaink, Leiden Center for Natural Computing, Springer - Verlag Berlin Heidelberg New York, 2002, 420 p.

Svetlana Cojocaru, Elena Boian,

Received June 8, 2010

Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E–mail: {Svetlana.Cojocaru,lena}@math.md

Computer Science Journal of Moldova, vol.18, no.1(52), 2010

Developing a derivatives generator

Mircea Petic

Abstract

The article intends to highlight the particularities of the derivational morphology mechanisms that will help in lexical resources extension. Some computing approaches for derivational morphology are given for several languages, inclusively for Romanian. This paper deals with some preprocessing particularities, that are needed in the process of automatic generation. Then, generative mechanisms are presented in the form of derivational formal rules separately for prefixation and suffixation. The article ends with several approaches in automatic new generated words validation.

Key-words: derivatives, word generation, lexicon, word validation

1 Introduction

Romanian derivational morphology represents an important issue in Romanian lexical resources extension. To automate the process of derivation it is necessary: to establish rules that can be applied to stems in order to obtain new derivatives; to establish conditions in which these rules can be applied; if these restrictions do not guarantee the correctness of the generated words - to develop and to implement a validation mechanism.

In consideration of premises, this article pretends to highlight the particularities of the derivational morphology mechanisms that will help in lexical resources extension without any semantic information.

In order to understand the differences and similarities of the approaches used in our research, the article starts with a description

©2010 by M. Petic

of the known methods used in derivational morphology for Romanian and other languages. Then some preprocessing particularities, that are needed in the process of automatic generation, are described, such as the issues connected with derivatives analysis and the particular features of a lexicon for derivatives generations aims, followed by the short description of the vowel and/or consonant alternation. The generative mechanisms are presented in the form of derivational formal rules separately for prefixation and suffixation. The article ends with several approaches in automatic new generated words validation.

2 The derivational process automatization

This section will be a brief overview of automation methods of derivation for different languages. Note that the automation of the derivation process mechanisms can help to solve other problems, such as: generation of morphological families, generation of derivatives with predictable meanings, expanding dictionaries and lexicons, informational retrieval, machine translation, etc [1]. Below, different approaches of the derivational morphology will be described for the following languages: Russian, Italian, Serbian, Arabian, French and Romanian.

Studying the automatization of the derivational process on the examples from different languages, we came to a conclusion that the obvious elements for processing in derivational morphology are vocabulary, lexicon or dictionary. Though it is not the subject of the present compartment. Beside the list of words, there is also another important moment to be discussed. There are two approaches in finding the corresponding derivatives. The first approach provides, that the derivatives are simply described in the lexicons, and being needed they are extracted with the help of some restrictions. The second approach corresponds to generative mechanisms, that form derivatives using constraint rules.

So, the first approach is used in the description of the Italian derivational morphology where generation is fulfilled only with regard to derivatives included into descriptions of all derived words in terms of finite automata. Another example is the system designed for Arabic

morphology involving two kinds of hierarchies: one for morphological forms and the other – for set of rules. In this case all stems and the corresponding information are stored in the lexicon.

The second approach is found in the description of Russian, Serbian and French languages. **RUSLO** (RUsskoe SLOvoobrazovanie) **derivation system**, works with Russian words. It can analyse both present and not present (as the jargon and neologisms and/or slang) in the dictionary words. RUSLO solved the problem of generation and analysis of derivatives for Russian language through detailed generative mechanisms of derivation [2]. In the case of Serbian language derivatives were generated in a predictable way. The derivation is considered predictable if the word changes gender (profesor → profesorka), or enhances meaning, i.e. generates diminutives (profesorčić) and augments (profesorčina), forms relational adjectives (profesorski) and possessive adjectives (profesorov) as well. The last one in fact is not the case of Romanian language. This process was called **regular derivation** [3].

GeDeriF is a system for French, which automatically analyses unknown in dictionary words and overgenerates derivatives. The system uses derivational rules for suffixes *-able*, *-ité* and *-is (-er).* These generated derivatives had been checked in Encyclopedia Universalis and terminology review Le Banc de Mots, which was drawn from a variety of sources. Besides this, they made a program that automatically checked the search engine www.yahoo.fr for each of the generated terms. Nevertheless the average of the percentage of correct words is very low [4].

One of the first applications of automatic differentiation system for Romanian language was FAVR in the Mac environment ELU which aimed to complete coverage of the inflectional morphology. Then, prefixes and suffixes were described by means of lexical or grammatical paradigms. In this scope 20 grammar categories have been used. This morphologic description was tested on more than 15.000 lexical entries [5].

3 Preprocessing in derivational morphology

As it was emphasized above, the lexicon plays an important role in the process of automatization of the derivational morphology. That is why some particularities concerning lexicons are given below. Another important issues in derivational morphology are the derivatives recognition. In addition the problem of vowel and/or consonant alternations is described by presenting a short picture of the type of alternations with concrete examples.

3.1 Lexicon for derivatives generation

Lexicon represents one of the main elements in the process of new derivatives generation. In this case the lexicon is not simply a repository for input of words with syntactic and semantic information (or lemma level), but also prefixes and suffixes are described in it [6].

Another point of view supposes that lexicons should contain not only dictionaries of simple words and their inflections, but also the dictionary of compound words, and dictionary of the finite-state transducers used to recognise unregistered words in the dictionaries [3].

Although for our purposes the best solution is the Dictionary of derivatives [7] containing only the graphical representation and constituent morphemes without any information about their part of speech, though the vast majority are nouns, verbs and adjectives. Electronic version of the dictionary [7] was obtained after it was scanned, the original input OCRized and the corrections made. This electronic version of the dictionary [7] becomes important as it is difficult to establish criteria for validation of new generated derivatives. In addition, it allows detection of derivatives with the appropriate type morphemes (prefix, root and suffix) and is an important electronic resource for research derivational morphology. Basically, the entries in the dictionary [7] are being built based on an uncertain schedule. In this scheme it is not clear where the affixes and the root are. In order to exclude the uncertainty of the electronic version of the dictionary entries, a regular expression representing the structure of derivatives was developed:

```
derivative = (+ morpheme)*.morpheme(-morpheme)*
```

where `+morpheme` is a prefix, `.morpheme` is a root and `-morpheme` is a suffix. An example of an entry in the lexicon is:

```
antistatal=+anti.stat-al

reprogramabil=+re.programa-bil
```

3.2 Automatic derivatives recognition

The majority of the derivational rules are taking into account the consequence of letters referring to words endings or suffixes. Moreover, it is not a good thing to generate several times derivatives with the same prefix. That is why it is important to have a mechanism for derivatives recognition.

As a source for automatic derivatives recognition, a lexicon serves, containing not only graphic representation of the words, but also their part of speech. The lexicon consists of approximately 100000 of words bases, and words can have several entrances for different parts of speech. Besides the lexicon, lists of prefixes with their phonological forms and suffixes were used.

Since not all the words end (begin) with the same suffixes (prefixes), some algorithms were elaborated for enabling the automatic extraction of the derivatives from the lexicon. The elaborated algorithms took into account the fact that being $x, y \in \Sigma^+$, where Σ^+ is the set of all possible roots, and if $y = xv$ then v is the suffix of y and if $y = ux$ then u is the prefix of y. In this context both y and x must be valid words in Romanian language, and u and v are strings that can be affixes for Romanian language. The problem of consonant and/or vowel alternations was neglected in the case of the algorithm of derivatives extraction. This fact does not permit the exact detecting of all derivatives [8].

Being more precise, the following word formation scheme expresses the particularities of prefixation:

$$[prefix\,[stem]_x]_x$$

where x represents part of speech for stem and derivative. Note that in the process of prefixation the part of speech does not change. In the process of suffixation there are cases of part of speech changing, as it is presented in the following word formation scheme:

$$[[stem]_x\,suffix]_y$$

Taking into consideration the peculiarities of the Romanian affixes and derivatives the algorithm for automatic derivatives recognition was elaborated that lately was implemented in a program written in Java programming language. This program allows us to follow at every step of the algorithm the partial results listed in the corresponding textual files.

3.3 Classification of affixes attachment

We examine some classes of affixes attachment. The situation is that there are more derivatives without alternations, especially in the case of the prefix derivation. The lack of vowel and/or consonant alternations in the process of derivation is observed with the following most frequent prefixes: *ne-*, *re-*, *pre-*, *anti-*, *auto-*, *supra-*, and *de-* [8].

There are cases when affixes do not need vowel and/or consonant alternations in the process of derivation. Below we will present some of these cases. The attachment of the affixes to the words is done by means of:

– addition of a letter to the end of the root, for example, *şurub* → *înşurub**a**, bold* → *îmbold**i**, plin* → *împlin**i***;

– deleting of the final letter in the root, for example, *lîn**ă*** → *dezlîn**a**, purpur**ă*** → *împurpur**a**, puşc**ă*** → *împuşc**a***;

– changing in the prefix, for example, *şoca* → ***de(s)**şoca* → ***de**şoca, pat* → ***su(b)**pat* → ***su**pat*;

– avoiding of the double consonant, for example, *spinteca* → ***de(s)s**pinteca* → ***des**pinteca, braţ* → ***su(b)b**raţ* → ***sub**raţ*;

– changing of two final letters in the root, for example, *zeflem**ea*** → *zeflem**itor**, încăp**ea*** → *încăp**ătoare***,

– changing of the final letter in the root, for example, *alini**a*** → *alini**ere**, aşchi**a*** → *aşchi**etor**, cumpăr**a*** → *cumpăr**ător**, curăţ**i*** → *curăţ**ător**, delăs**a*** → *delăs**ător**, depun**e*** → *depun**ător**, faianţ**ă*** → *faianţ**ator**, fărîm**a*** → *fărîm**ător**, împing**e*** → *împing**ător**, transcri**e*** → *transcri**itor***, *c**a**n**ă*** → ***c**ă**n**eală***, *atr**a**g**e*** → *atr**ă**g**ătoare**, b**a**te* → ***b**ă**t**ătoare***;

– removing of the last vowel in the root, for example, *răşchi**a*** → *răşchi**tor**, acr**i*** → *acr**eală**, adun**a*** → *adun**ătoare***.

3.4 Problem of vowel and/or consonant alternation

The problem of derivation consists not only in the detection of the derivational rules for separate affixes, but also in the examination of the concrete consonant and/or vowel alternations for the affixes. It is important that not all affixes need vowel and/or consonant alternations in the process of derivation. The vowel and/or consonant alternations are a subject for research not only in derivational morphology but also in inflectional morphology. Though there are some similarities, for example, *ean* → *en* (moldov*ean* – moldov*en*i – moldov*en*esc), *o* → *u* (s*o*ră – s*u*rori – s*u*rioară), *oa* → *o* (pl*oa*ie – pl*o*i – pl*o*iţă), *t* → *ţ* (bărba*t* – bărbaţi – bărbăţie), etc. But the derivational alternations differ from those inflectional, for example: *at* →ăţ (argat – argăţesc), *ar* → *er*, (adevăr – adeveri), *g* → *s* (împunge – împunsătură), etc.

There are no cases with consonant and/or vowel alternations in the process of derivation with suffixes. It means that there are situations when the derivation is made up with minimum number of alternations and with maximum cases of changes in the root, for example: *a* →ă *a* →ă (balsam – îmbălsăma), *a* →ă *a* →ă *a* →ă (caimacam – căimăcămie) etc. Possible vowel and/or consonant alternations are so varied that it is difficult to describe them all in a chapter, but it is possible, at least, to classify them:

– removing of final vowel and changing of final consonant, for example, *descre**şte*** → *descre**scătoare**, închi**de*** → *închi**zătoare**, încre**de***

→ *încre**zătoare**, promi**te** → primi**ţătoare***;

– changing of the vowels in the root, for example, ***c**a**t**a**ra**m**ă*** → *înc**ă**t**ă**r**ă**m**a**, primăv**ar**ă → desprimăv**ăr**a, răd**ă**cin**ă** → dezrădăcin**a**, pl**a**to**şă** → împl**ă**to**şi***;

– changing in the root, for example, *rîde → rîzătoare, recunoaşte → recunoscătoare, roade → rozătoare, sta → stătătoare, şedea → şezătoare, vedea → văzătoare, şti → ştiutor.*

On purpose of precision which affixes have alternations in the process of derivation, the digital variant of the derivatives dictionary has been studied. Some of them are illustrated in the Table 1.

Taking into consideration all these observations, it is easier to understand the derivative structure, namely the prefixes, stem and the suffixes of the derivatives. It represents a starting point for the process of automatic derivatives generation.

4 Derivatives generation

Besides the problem of derivatives analysis there is a wish to have the possibility to generate new derivatives, taking into account the stem and affix peculiarities. In the process of linguistic resources completion by automatic derivation appear a natural tendency to use the most frequent affixes. In reality, the most productive affixes prove to be problematic because of their irregular behaviour. That is why for the research there have been chosen those affixes that have allowed to establish simpler behaviour rules, as not to appeal to too much exceptions [9]. That is why the examples of prefixation with *re-*, *ne-*, *in-/im-*, and suffixation with *-re*, *-bil*, *-tor*, *-toare*, *-esc/-ească*, *-iza* are described below.

4.1 Automatic prefixation

The rule of derivation with the prefix *re-* is the following, let ω be the infinitive of the verb, then the word of the form $\omega\prime = re\omega$ is also the infinitive of the verb, namely

Table 1. Vowel and/or consonant alternation

Alter. vow/cons	Root/ Stem	Context of alt. vow/cons	Pref/ Suf	Word Examples
a →ă	albastru arab cărare dalb	bas – băs rab – răb rar– răr dal – dăl	el ească uşă ior	albăstrel arăbească cărăruşă dălbior
a → ăr	gustare	tar – tăr	ică	gustărică
a → e	iarbă	iar – ier	ăluă	ierbăluă
a → ă a → ă	dandana	dan – dăn dan – dăn	 ie	dăndănaie
	balsam	bal – băl sam – săm	îm a	 îmbălsăma
a → ă a → ă a → ă	calafat	cal – căl laf – lăf fat – făt	ui	 călăfătui
a → ă at → ăţ	Banat	ban – băn nat – năţ	ean	 bănăţean
a → ă ţ –c	baniţă	ban – băn niţ – nic	ioară	 bănicioară
a → e a → ă	iatac iatac	iat - iet tac - tăc	el	 ietăcel
at → ieţ	băiat	ăiat – ăieţ	andru	băieţandru

$$[\omega]_{inf} \rightarrow \left[re\,[\omega]_{inf}\right]_{inf}.$$

In this case there are derivatives like (a) filma → (a) *re*filma, (a) genera → (a) *re*genera, etc. As in the previous case, there are no any vowel and/or consonant alternations in this case.

In this context it is observed that the root for the derivative with the prefix *re-* and suffix *-re* is the infinitive of the verb. So, let ω be the infinitive of a verb, then $\omega\prime = re\omega re$ is a noun, namely

$$[\omega]_{inf} \rightarrow \left[re\,[\omega]_{inf}\,re\right]_{Nn}.$$

The derivatives would be: (a) întîlni → *re*întîlni*re*, (a) verifica → *re*verifica*re*. There are no vowel and/or consonant alternations.

Another known affix, which will permit to generate many derivatives, is the prefix *ne-*. Thus, let ω be an adjective of the form $\omega\prime = \omega\beta$, where $\beta \in$ { *-tor, -bil, -os, -at, -it, -ut, -ind, -ind*}, then the derivatives of the form $\omega'' = ne\omega\beta$ are possible to generate and the resulted derivatives will be also adjectives.

$$[\omega\beta]_{Adj} \rightarrow \left[ne\,[\omega\beta]_{Adj}\right]_{Adj}.$$

In this case the obtained derivatives would be: conductor → *ne*conductor, nobil → *ne*nobil, invidios → *ne*invidios, iubit → *ne*iubit, născut → *ne*născut. In the process of derivation with the prefix *ne-* the vowel and/or consonant alternations are not observed. Though a question appears, what the endings β represent. If in some cases it is clear that they are forms of participle or gerund, then the strings *tor* and *bil* are lexical suffixes. So an interest appears to the process of derivation with these suffixes.

The derivatives with the prefixes $im-/in-$, as a rule, are adjectives, rarely nouns and verbs. The most numerous derivatives with prefix $in-$ $/im-$ are adjectives formed with the suffix bil, for example, *in*cura*bil*, *in*estima*bil*, etc. So, being the adjectives of the form $\omega\prime = \omega bil$, they form derivatives of the form $\omega'' = \omega bil$, where $\omega \in \{in-, im-\}$ [1].

Another well contoured group is that of adjectives derivated with the suffixes $-ent$ and $-ant$: inader*ent*, incoer*ent*, independ*ent*, etc. Similar, being the adjectives $\omega\prime = \omega\gamma$, they form derivatives $\omega'' = \beta\omega\gamma$, where $\beta \in \{in-, im-\}$ and $\gamma \in \{-ent, -ant\}$. In both cases the choice of the β depends on the first letter of the adjective ω, and namely in the case when the letter is b or p then $\beta = im-$, in other cases it is $in-$.

4.2 Automatic suffixation

For the suffix *-re* there is the following rule: let ω be the infinitive of a verb, then the word of the form $\omega\prime = \omega re$ is a noun, namely

$$[\omega]_{inf} \rightarrow \Big[[\omega]_{inf}\, re\Big]_{Nn}.$$

This formal model can generate derivatives such as: citi $\rightarrow$ citi*re*, mînca $\rightarrow$ mînca*re*, etc. In the process of derivation there are no vowel and/or consonant alternations.

That is because the suffixes *-tor* and *-bil* have been studied. Both of them have the same origin. Thus, let ω be the infinitive of the verb of the form $\omega\prime = \omega\beta$, where $\beta \in \{-a, i\}$, then it is possible to form the derivatives of the form $\omega'' = \omega\beta\gamma$, where $\gamma \in \{-tor, -bil\}$ is adjective/noun.

This examination includes the verbal lexical simple suffix *-iza*, which has neologic origin and nowadays is very productive and has very strong relation with the lexical suffixes *-ism* and *-ist*. Thus, let ω be an adjective/noun of the form $\omega\prime = \omega\beta\gamma$, where $\gamma \in \{-ism, -ist\}$, then it is possible to say about the derivatives the following:

1. if $\beta \in \{-an, -ian\}$, then the word of the form $\omega\prime\beta = \omega iza$ is a verb;

2. of $\beta \in \{-ean\}$, then the word of the form $\omega\prime e\boxed{a}n = \omega iza$ is a verb, where $\boxed{a}$ represents the cut out of the vowel a;

3. if $\beta = \mu ic$, where:

- $\mu \in \{-at, -et, -ot, -if\}$, then the word of the form $\omega\prime = \omega\mu iza$ is a verb;
- $\mu \notin \{-at, -et, -ot, -if\}$, then the word of the form $\omega\prime = \omega\beta iza$ is a verb;

4. if $\beta \in -ur$ă , then the word of the form $\omega\prime = \omega ur\boxed{ă}iza$ is a verb, where $\boxed{ă}$ represents the cut out of the vowel a.

Thus, the examples of such derivatives are: alcan → alcan*iza*, european → europen*iza*, dramatic → dramat*iza*, cosmetic → cosmet*iza*, patriotic → patriot*iza*, ştiinţific → ştiinţif*iza*, caricatură → caricatur*iza*, friptură → friptur*iza* [1].

The word gender changing can be achieved by switching to other corresponding suffixes, for example, $-tor$ → $-toare$, $-esc$ → *-ească*, etc. Thus, it was observed that the gender changing is made with the help of suffixation, not of prefixation one. The lexicon, mentioned above, consists of suffixed derivatives only with *-tor*, only with *-toare* and with *-tor* and *-toare* at the same time. There are 148 words (nouns and/or adjectives) of the form $\omega' = \omega tor$, which could change into the words of the form $\omega'' = \omega toare$. Similarly, there are 42 words (nouns and/or adjectives) of the form $\omega' = \omega toare$ which could change into the words of the form $\omega'' = \omega tor$. Nevertheless, these 190 words generated in an automatic way should be validated. First of all, words were checked on their presence in RRTLN. 122 from all generated words were present there. The remaining words were checked in the electronic documents of the Internet, and 49 of 68 derivatives have been validated. Thus 95% of generated words were valid.

The same situation is with the pair of the suffixes *-esc* and *-ească*. According to the same lexicon, it consists of 274 of derivatives with suffixes *-esc*, and 249 with the suffix *-ească*. Note, that 229 of the derivatives are suffixes both with *-esc* and *-ească*. It is natural to assume that the words (nouns and/or adjectives) of the form $\omega' = \omega esc$, could change into the words of the form $\omega'' = \omega$*ească*. Similarly, the words (nouns and/or adjectives) of the form $\omega' = \omega$*ească* could change into the words of the form $\omega'' = \omega esc$. Generating in an automatic way

those derivatives which lack in the case of gender and checking them in an automatic way in the electronic documents, it was established that with the help of RRTLN there were validated 43 words of all 65 generated words. Another 12 of 22 remaining derivatives were validated using a web application based on Google search engine opportunities. So, 84% of obtained words were validated.

5 Problem of derivatives validation

Automatic derivation represents an overgenerating mechanism. That is why validation of generated words is needed.

5.1 Models of validation

One of the methods of new word validation consists in manual verification of every new generated derivative as to correspond to semantic and morphologic rules. In the case of the proceeding is performed by a specialist in domain, the specific disadvantages of a manual work appear: considerable resources of time and the possibility to make mistakes. So, this method of validation becomes inefficient [1].

Another method of validation consists of the verification of the derivatives in the existent electronic documents.

5.2 Automatic validation

There are different types of electronic documents.

The first idea that appears – to validate words using existent corpora, that represent verified documents – seems to be the best solution. The condition for being the panacea in the new word validation is a representative corpus, with a big number of words from different domains. As there are no representative Romanian corpora, it is not possible to consider it a good idea.

On the other hand there are documents on Internet, that are not verified, that is why they are not credible. In order to make it more precise, the searching on the Internet should be made for the documents

typed only in Romanian language. Besides this, it is necessary that the following be assured: the possibility to exclude word segmentation; the part of speech of the derivatives [9].

This validation tool divides the generated derivatives in three categories. The first one contains words that are not found in Internet. The second consists of the derivatives that appear less than a frequency limit of n, in our case $n = 1000$. Derivatives that are more frequent that limit n, are registered in the third group. This classification pretends that the words, that are listed more than frequency limit of n, are surely valid. Those, that are from the second group, can be valid but should be verified by specialists in linguistics. The derivatives, that are not present, could not be valid.

The idea of classification pretends to be a mixt method of validation, because needs only the manual verification for the words from the second category.

6 Conclusions

Generation of derivatives is not a trivial problem, because the process does not have a regular mechanism. The solution to store all derivatives of a dictionary is a reasonable one, because these derivatives still will not cover the full diversity of language, being in continuous evolution. From the other hand, the approach to generate constraint derivatives according to constraint rules for derived groups is a mechanism of overgeneration, when the validation phase excludes many wrong formed words. Well defined rules will increase the level of the correct words generation.

References

[1] S. Cojocaru, E. Boian, M. Petic. *Stages in automatic derivational morphology processing*, KEPT2009, Knowledge Engineering, Principles and Techniques, Selected Papers, Cluj-Napoca, July 2 - 4, 2009, pp.97-104.

[2] N. Percova. *RUSLO: An Automatic System for Derivation in Russian - http : //lcl.srcc.msu.ru/library/pertsova_ruslo.pdf* - 22.04.09

[3] V. Duško, C. Krstev. *Derivational Morphology in a E-Dictionary of Serbian*, In Zygmunt Vetulani (ed.), Proceedings of the 2nd Language & Technology Conference, Poznan, Poland, 2005, pp. 139–143.

[4] F. Namer, G. Dall. *GeDeriF: Automatic Generation and Analysis of Morphologically Constructed Lexical Resources*, In LREC: 2nd International Conference on Language Resources & Evaluation,

[5] D. Tufiş, L. Diaconu, A. M. Barbu, C. Diaconu. *Romanian language morphology, a reversible and reusable linguistic resource*, Language and Technology, Publishing House of Romanian Academy, Bucureşti, 1996, pp. 59–65. (in Romanian)

[6] F. Carota. *Derivational Morphology of Italian: Principles of Formalization*, Literary and Linguistic Computing, Vol. 21, Suppl. Issue, 2006.

[7] S. Constantinescu. *The dictionary of derivated words*. Editura Herra, Bucureşti, 2008. (in Romanian)

[8] M. Petic. *Automatic derivational morphology contribution to Romanian lexical acquisition.* Special issue: Natural Language Processing and its Application. Research in Computing Science, Mexico, vol. 46, 2010, pp. 67–78.

[9] M. Petic. *Automatic extention of Romanian linguistic resources*, Romanian Workshop for Linguistic Tools and Resources Volume, Publishing House of the University "Al. I. Cuza", Iaşi, România, 2008, pp. 151–160. (in Romanian)

M. Petic, Received June 25, 2010

M. Petic
Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E–mail: *mirsha@math.md*