Volume 13 Number 2

ACTA CYBERNETICA

Szeged, 1997

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Contributions are accepted for review with the understanding that the same work has not been published elsewhere.

Manuscripts must be in English and should be sent in triplicate to any of the Editors. On the first page, the *title* of the paper, the *name(s)* and *affiliation(s)*, together with the *mailing* and *electronic address(es)* of the author(s) must appear. An *abstract* summarizing the results of the paper is also required. References should be listed in alphabetical order at the end of the paper in the form which can be seen in any article already published in the journal. Manuscripts are expected to be made with a great care. If typewritten, they should be typed double-spaced on one side of each sheet. Authors are encouraged to use any available dialect of TEX.

After acceptance, the authors will be asked to send the manuscript's source TEX file, if any, on a diskette to the Managing Editor. Having the TEX file of the paper can speed up the process of the publication considerably. Authors of accepted contributions may be asked to send the original drawings or computer outputs of figures appearing in the paper. In order to make a photographic reproduction possible, drawings of such figures should be on separate sheets, in India ink, and carefully lettered.

There are no page charges. Fifty reprints are supplied for each article published.

Publication information. Acta Cybernetica (ISSN 0324–721X) is published by the Department of Informatics of the József Attila University, Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. For 1997 Numbers 1–2 of Volume 13 are scheduled. Subscription prices are available upon request from the publisher. Issues are sent normally by surface mail except to overseas countries where air delivery is ensured. Claims for missing issues are accepted within six months of our publication date. Please address all requests for subscription information to: Department of Informatics, József Attila University, H-6701 Szeged, P.O.Box 652, Hungary. Tel.: (36)-(62)-311-184, Fax:(36)-(62)-312-292.

URL access. All these information and the contents of the last some issues are available in the Acta Cybernetica home page at http://www.inf.u-szeged.hu/local/acta.

Framework for Generating Object-Oriented Databases from Conceptual Specifications

György Kovács [*] Patrick van Bommel [†]

Abstract

When designing underlying databases of information systems, data are first modelled on conceptual level, then the obtained conceptual data models are transformed to database schemas. The focus of this paper is the transformation of conceptual models into database systems with object-oriented features. The transformation is captured within the framework of a two level architecture. Conceptual models are first mapped to abstract intermediate specifications, which are then transformed to database schemas in a given target environment. This enables us to treat different target systems, such as object-oriented and object-relational systems including the standards ODMG and SQL3, in a uniform way. To express intermediate representations of conceptual models we use F-logic, a logic-based abstract specification language for object-oriented systems. We focus on the first step of the overall transformation, i.e. the mapping of conceptual models into F-logic. Several transformation alternatives are discussed, and a corresponding graphical notation for specifying transformation alternatives is provided.

Keywords: database design, data transformation, conceptual data models, OO models

1 Introduction

It has been generally agreed on that conceptual data modelling is very important when building information systems. This means that data must be modelled first on conceptual level, and then the obtained conceptual model (conceptual schema) must be translated to the external and internal level, according to the three level architecture for information systems modelling ([15]). By doing so, the issues of correctness and efficiency are well-separated, which is quite desirable. In this paper we deal with the transformation of conceptual data models to the internal level.

[*]Department of Information Systems, Eötvös Loránd University, Múzeum krt. 6-8., H-1088 Budapest, Hungary, email: gykovacs@ullman.inf.elte.hu. Supported by the Hungarian Scientific Research Fund (OTKA W-015176) and Nuffic.

[†]Department of Information Systems, University of Nijmegen, Toernooiveld 1, NL-6525 ED Nijmegen, The Netherlands, email:pvb@cs.kun.nl

Internal (implementation-oriented) models are considered to be database models that are supported by database management systems (DBMS) running on computers. It is also assumed (required) that some database language is provided for such a model. The well-known relational model ([11], [36]) is the underlying database model of today's RDBMSs and the related database language is typically SQL (see e.g. [13]). However, the relational technology is not always appropriate for some real-life applications, e.g. multimedia or geographical applications, where usually highly structured, complex objects must be stored and manipulated. To overcome the limitations of the relational model, advanced database models have been developed, such as the nested relational model (see e.g. [30], [1], [5]) along with proposals for nested SQL extensions (e.g. [27], [28], [23]) as well as database models with object-oriented features. Database systems with object facilities serve as candidates for next generation database systems. Although a number of such models and query languages have been proposed (see e.g. [2], [32], [4], [31], [19]), there is no object-oriented database model and language yet, that has been commonly accepted. This fact is known and inspired people attempting to define the requirements for next generation DBMSs ([3], [34], [20], [12]). Basically, two main approaches, the pure object-oriented (OO) and the object-relational (OR), compete with each other and seem to co-exist in the future. The same is reflected in the standardisation efforts resulting in the ODMG-93 ([9]) de facto standard for truly object database systems and SQL3 ([24]) for object-relational systems, though some compatibility between them is also aimed.

For data modelling a number of semantic modelling techniques have been developed, such as (extended) ER ([10], [14]), NIAM ([26]) and PSM ([18], [16]). Semantic modelling has been used in practice for long and has proved to be a powerful technique. The mapping of resulting conceptual schemas into relational environments is well-defined (see e.g. [35], [26]) and this process is supported by many CASE-tools. Also, a transformation mechanism to nested relational schemas has been established ([8], [7]). Although there exist OO data modelling techniques, e.g. OMT ([29]), for designig object-oriented databases, the use of traditional semantic modelling will very likely not disappear from system design, but will remain as a powerful alternative, especially in data intensive domains. Indeed, in [33], where semantic modelling (using extended ER) and OO data modelling are compared, it is concluded that even when OO databases are designed the recommended strategy is: (1) creating an EER (or e.g. NIAM) schema; (2) map it to an OO schema; and (3) augment OO schema with behavioral constructs. As a consequence, a mechanism for transforming semantic models into modern database systems is needed.

In this paper we deal with the problem of how to transform conceptual data models into somehow object-oriented database environments. Although valuable previous work has been done on this topic (e.g. [25], [21], [6]), a general unifying mechanism is still missing, which inspired our work. In general, a conceptual data model (conceptual schema) consists of an information structure and a set of integrity constraints, both of which require translation. In addition to the results of existing proposals, advanced modelling constructs (set types, list types, generalisation) are to be considered as part of the structure translation process. Even for simple constructs additional tranformation alternatives can be recognized. Moreover, a more comprehensive treatment of constraints is necessary in general.

In our approach (similarly to that of [6]) the transformation is captured within the framework of a two level architecture. Conceptual schemas are first transformed to abstract intermediate specifications (design step). Then the obtained intermediate specifications are translated into the final implementation environment (implementation step). This means that in case of different target environments the same design step can be applied and only the implementation step will differ. That is, choosing a new target system requires only the implementation step to be adapted. Thus, as the main benefit of this approach, we gain general applicability. The common design decisions can be factored out in the first step. Here we focus on the design step, the second step is discussed only in very general terms.

For expressing intermediate representations F-logic ([19]), a logic-based abstract specification language for object-oriented systems, is used (cf. [6]). Conceptual models are defined in terms of PSM (Predicator Set Model, [18], [16]), a fully formalized extension of NIAM. However, our approach is easily applicable to other conceptual modelling techniques (e.g. ER) due to the usage of similar constructs.

In the present paper we set up the framework for a general transformation mechanism consisting of two steps as discussed above. It serves as a basis towards working out a comprehensive method for designing modern (OO,OR) databases based on conceptual (semantic) data modelling. We focus on structural aspects and outline a number of alternatives for the translation of information structures into OO systems. A corresponding graphical notation is introduced for illustration purposes. Since the mapping of structures is influenced by simple uniqueness (key) constraints, such constraints are also covered. However, the transformation of complex conceptual constraints in general is beyond the scope of the present paper, though it belongs to the whole picture and is seen as an essential part of the overall transformation, which has to be worked out.

The rest of the paper is organized as follows. In section 2 our approach is presented. In section 3 the conceptual data modelling technique PSM is summarized to an extent needed for the purpose of the paper. Section 4 gives an overview of F-logic, that is used for expressing intermediate specifications. Alternatives for the transformation of conceptual information structures into F-logic (the design step) are discussed in section 5. In section 6 the transformation of an example PSM schema into F-logic is worked out in detail. The implementation step is discussed in general terms in section 7, where also an example schema definition in ODMG-93 is provided. Section 8 contains the conclusions and topics for further work.

2 Approach

The expected end product of the transformation of a conceptual data model is something that can be run on a computer with a given target environment (DBMS) for creating a database. That is, at the end a sequence of statements in the database language of the assumed DBMS has to be generated to create the corresponding database schema. As it was already mentioned, in our approach the transformation is captured in the framework of a two level architecture, i.e. it is performed in two steps as shown in figure 1. Conceptual schemas are first transformed to abstract intermediate specifications (design step). Then the obtained intermediate specifications are translated into the final implementation environment (implementation step). The task of the second step is the generation of statements in a concrete database language. In [6] a similar approach is taken.

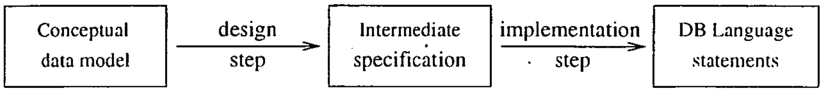


Figure 1: Two level architecture for transformation

There are several advantages of a two level architecture approach, e.g.:

- Provided that the intermediate specification language is general enough to cover all the final target models that are intended to be considered, the design step, which is the more complex and essential part of the whole transformation, becomes the same single task for different target environments and is independent of the choice of the actual target system. The different system specific details must be dealt with in the implementation step only.

- Provided that conceptual models and intermediate specifications have underlying precise formalism, the transformation from the conceptual to the intermediate level can be given algorithmically in a formal framework. This is very fundamental in order to have an automated transformation mechanism.

- Since a given conceptual model may have a number of correct representations on the internal level and possibly the best candidate should be chosen, optimization is important. In a two level transformation optimization can be incorporated at both levels.

We have already made it clear that to express data models on the conceptual level, we will use the Predicator Set Model ([18], [16]), an extension of NIAM ([26]). PSM is a fully formalized expressive modelling technique. It is briefly summarized in section 3. As potential final target environments, truly object-oriented as well as object-relational database systems are considered including the related standards ODMG and SQL3, respectively. Fixing an appropriate specification language for specifying intermediate models is a basic task. When doing so, the following requirements are fundamental to be taken into account:

- The chosen specification language should be implementation-oriented, i.e. it should be able to deal with (important) concepts of implementation environments. At the same time it should provide a high level of abstraction to make it possible for us not to deal with the irrelevant aspects in the design step.

- It must be general enough and support object-oriented concepts to cover object-relational and object-oriented target systems.
- It must be provided with sufficient support for constraint specifications. On the one hand, because the transformation of structures often imply integrity constraints in the target database to be specified. On the other hand, because the translation of conceptual integrity constraints typically (but not always) results in database constraints. Although the general treatment of constraints and their translation is outside the scope of our present paper, this is a very essential perspectival requirement.
- It must have formal syntax and semantics. This makes it possible to define our transformation in a formal framework.

To sum it up, we need an abstract and formal database model with object-oriented facilities, that also allows to specify integrity constraints. After investigating a number of proposals (e.g. [2], [32], [31], [19]) we have concluded that F-logic ([19]) is the one that fulfils our needs the best. Models of other proposals are not general enough and/or are not provided with precise formal syntax and semantics and/or do not deal with constraints at all. Consequently, we use F-logic as an abstract intermediate specification language. In section 4 an overview of F-logic is given based on [19], where it was presented. We note that F-logic has a pure object-oriented view. However, it can serve as an abstract intermediate specification language in case of object-relational database sytems as well. The picture of figure 1 can now be refined to show our approach more concretely, which is depicted in figure 2.

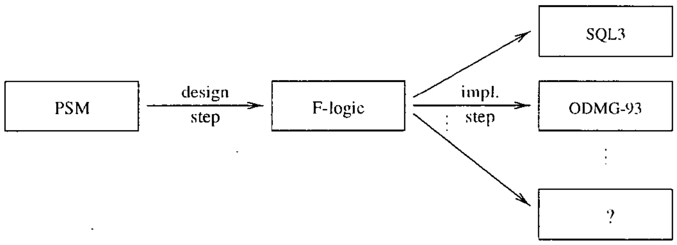


Figure 2: The concrete architecture

3 Conceptual data models

In this section we give a brief overview of the Predicator Set Model (PSM), that is used for expressing conceptual data models, without going into formal details (for details see [18] or [16]). A conceptual data model $\Sigma = \langle \mathcal{I}, \mathcal{C} \rangle$ consists of an information structure $\mathcal{I}$ and a set of integrity constraints $\mathcal{C}$. An information structure $\mathcal{I}$ is a structure consisting of the following basic components:

1. A set $\mathcal{P}$ of *predicators*. A predicator is intended to specify the role played by an object type in a fact type (see below).

2. A set $\mathcal{O}$ of *object types*. Object types are classified as follows:

   (a) Entity types ($\mathcal{E}$) and label types ($\mathcal{L}$). The difference is that labels can, in contrast with entities, be represented (reproduced) on a communication medium. $\mathcal{D}$ is a set of concrete domains (e.g. string, natno) associated with label types via the function $\mathsf{Dom} : \mathcal{L} \longrightarrow \mathcal{D}$.

   (b) Fact types ($\mathcal{F}$). The set $\mathcal{F}$ is a partition of the set of predicators $\mathcal{P}$. The fact type that corresponds with a predicator is obtained by the auxiliary function $\mathsf{Fact} : \mathcal{P} \longrightarrow \mathcal{F}$.

   (c) Power types ($\mathcal{G}$) and sequence types ($\mathcal{S}$). Power types are also called set types. The intention of sequence types is to model list structures.

3. A function $\mathsf{Base} : \mathcal{P} \longrightarrow \mathcal{O}$ specifying the object type associated to a predicator.

4. A function $\mathsf{Elt} : \mathcal{G} \cup \mathcal{S} \to \mathcal{O}$ specifying the element type of power types and sequence types.

5. A binary relation Spec on object types, capturing specialisation. $a \, \mathsf{Spec} \, b$ is interpreted as "a is a subtype (specialisation) of b", or "b is a supertype of a"". Specialisation of label types is prohibited, and only entity types can act as subtypes ($\mathsf{Spec} \subseteq \mathcal{E} \times \mathcal{O} \backslash \mathcal{L}$). Specialisation networks are acyclic.

6. A binary relation Gen on object types, capturing generalisation. $a \, \mathsf{Gen} \, b$ is interpreted as "a is a generalisation of b", or "b is a specifier of a". Generalisation of label types is prohibited, and only entity types can act as generalised object types ($\mathsf{Gen} \subseteq \mathcal{E} \times \mathcal{O} \backslash \mathcal{L}$). Generalisation networks are acyclic. Furthemore, to avoid conflicting situations, generalised object types cannot be subtypes. The difference between generalisation and specialisation lies in their population (see below).

The connection between (abstract) entity types and (concrete) label types is established by so-called *bridge types*. A fact type f is called a bridge type only if it has the form $f = \{p, q\}$ with $\mathsf{Base}(p) \in \mathcal{L}$ and $\mathsf{Base}(q) \notin \mathcal{L}$. A fact type is called an objectified fact type if it is the base object type of some predicator.

Figure 3 shows an example information structure. In this figure we have a number of entity types, e.g. *Person* and *Project*, represented by circles. Label types, also represented by circles, appear in parentheses, examples are *Date* and *Tel_nr*. By convention, if a label type is an identifier for an entity type, then the label type is represented within the same circle, see e.g. entity type *Person* and its identifying label type *P_id*. Fact types consist of predicators represented by boxes connected with circles for their base object types. For example, fact type *Employment* is a binary fact type consisting of two predicators, *employed_by* and *employs*.

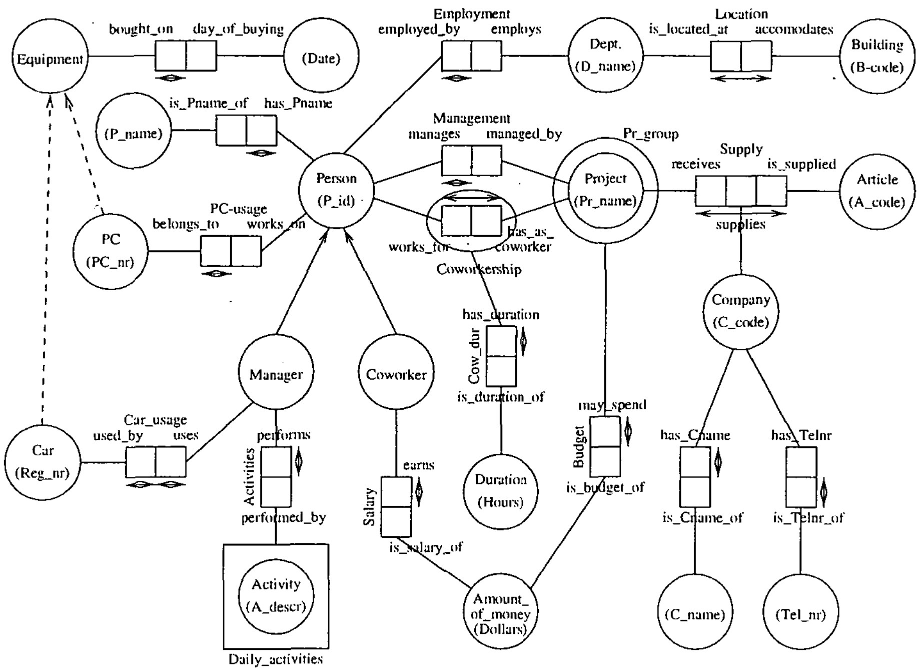


Figure 3: Information structure with uniqueness constraints

Figure 3 also contains representatives of advanced modelling constructs. We have power type *Pr_group* with element type *Project* and sequence type *Daily_activities* with element type *Activity*. Power types and sequence types are represented by circles and boxes, respectively, around their element type.. Entity types *Manager* and *Coworker* are specialised object types (subtypes) represented by solid arrows from subtypes to supertypes. Their common supertype is entity type *Person*. Entity type *Equipment* is a generalised object type with entity types *Car* and *PC* as its specifiers. Generalisations are represented by dashed arrows from specifiers to generalised types. As distinguished fact types, for bridge types many examples can be seen, e.g. $\{has\_Cname, is\_Cname\_of\}$ connecting entity type *Company* with label type *C_name*. As an example of objectified fact types we have fact type *Coworkership*.

Populations and constraints

An information structure is used as a frame for some part of the (real or fictive) world, the so-called Universe of Discourse (UoD). A *state* of the UoD corresponds with a so-called instantiation or population of the information structure, and vice versa. A population $\mathsf{Pop}_{\mathcal{I}}$ of an information structure $\mathcal{I}$ is a value assignment to the object types in $\mathcal{O}$, conforming to the structure as prescribed in $\mathcal{I}$. The population of a label type comes from the corresponding concrete domain (e.g. string, natural number), while the population of an entity type comes from an abstract domain containing unstructured values.

These domains are part of the *universe of instances* Ω, which is inductively defined as follows. Firstly, all possible atomic instances are contained in Ω. Secondly, Ω contains all possible composed instances such as (a) mappings from predicators to instances, intended for the population of fact types, (b) sets of instances, intended for the population of power types, (c) sequences of instances, intended for the population of sequence types. The basic difference between specialisation (subtyping) and generalisation is that a subtype gets the population (and identification) from its supertype using subtype defining rules, while the population of a generalised object type is the union of the population of its specifiers. For a formal treatment of populations we refer to [17].

Forbidden populations are excluded by so-called (static) integrity constraints. Uniqueness constraints require uniqueness of values in some set of predicators. Graphically such uniqness constraints are represented by double-headed arrows next to the predicators they belong to. For example, in figure 3 we have unique(*employed_by*) expressing that a person may belong to one department only. There is a variety of other constraints, such as total role, occurrence frequency, set, enumeration, power type, sequence type, and specialization constraints. These are not relevant for the purpose of the present paper.

4 F-logic

To express intermediate specifications we use F-logic, an abstract logic-based language for OO systems. In this section we shortly summarize the parts relevant for us. For a comprehensive description of F-logic we refer to [19], where it was presented. In [22] an overview of F-logic from the perspective of our transformation is given.

Basic elements in F-logic are *id-terms*, terms built from function symbols and variables as usual. They denote objects, classes and methods. Ground id-terms are variable-free id-terms playing the role of *logical* object identifiers (*oid*). By means of id-terms *F-molecules* can be constructed, and from F-molecules more complex formulas can be built. F-logic is provided with a model-theoretic semantics defined by means of semantic structures called *F-structures*. F-structures and the satisfaction of formulas are defined in such a way that the commonly known OO features are incorporated. Along with the description of its syntax, below we give an informal summary of the semantics of F-logic, for details see [19]. F-molecules are defined as follows (C, D, O, M, R, A_i-s, R_i-s, AT_i-s and RT_i-s below are id-terms, $k, l \geq 0$).

Is-a assertions of the form $C :: D$ and $O : C$ stating that class C is a subclass of class D and object O is a member of class C, respectively. Each class is subclass and superclass of itself. The subclass relation is transitive, and subclass hierarchies are acyclic. Objects belonging to a class also belong to any superclass of that class.

Structures (signature expressions, see below) are inherited from superclasses.

Object molecules of the form $O[$ a ';'-separated list of *method expressions* $]$. A method expression can be a *scalar data expression* $M \,@\, A_1, ..., A_k \rightarrow R$, a *set-valued data expression* $M \,@\, A_1, ..., A_k \twoheadrightarrow \{R_1, ..., R_l\}$, a *scalar signature expression* $M \,@\, AT_1, ..., AT_k \Rightarrow (RT_1, ..., RT_l)$, or a *set-valued signature expression* $M \,@\, AT_1, ..., AT_k \Rrightarrow (RT_1, ..., RT_l)$. Here O denotes an object or a class. M corresponds to a method. In data expressions it denotes method invocation, while in signature expressions it denotes the signature of some method. The syntactic context of M indicates that the corresponding method is a scalar function ($\rightarrow$, $\Rightarrow$) or a set-valued function ($\twoheadrightarrow$, $\Rrightarrow$). In scalar data expressions R represents the output of the method M when invoked on object O with arguments $A_1, ..., A_k$. In set-valued data expressions R_i-s represent elements of the resulting set. In signature expressions RT_i-s represent the *types* (classes) of the result (scalar case) or the types of the elements of the result (set-valued case) of the method M when invoked on an object of class O with arguments of types $AT_1, ..., AT_k$. The output (or the elements of the output, resp.) of the method must belong to *all* the RT_i classes. When only one result type is specified the parentheses may be omitted.

From F-molecules complex formulas (*F-formulae*) can be built by means of logical connectives $(\wedge, \vee, \neg)$ and quantifiers $(\forall, \exists)$ with their usual interpretation. The implication connective " $\longleftarrow$ " can also be used as usual, i.e. $\varphi \longleftarrow \psi$ is a shorthand for $\varphi \vee \neg\psi$.

F-logic databases (F-programs), well-typing

An F-logic *database*, also called an *F-program*, is basically an arbitrary set of F-formulae. Since this definition is too general, restrictions on the form of the allowed formulas are applied. An F-program **P** consists of a set of *rules*, statements of the form *head* $\longleftarrow$ *body*, where *head* is an F-molecule and *body* is a conjunction of literals (F-molecules or negated F-molecules). The semantics of F-programs is given by Herbrand interpretation, more concretely by *canonic Herbrand models* (H-models). An F-program can be structured in such a way that its schema (declaration) part and data (object base) part are shown separately.

Example 4.1

Let's suppose that we have a simple database about persons, employees and projects. Such a database can be defined by the following F-program:

Schema (declarations)

$employee :: person$

$person\,[name \Rightarrow string;$
$\quad age \Rightarrow integer\,]$

$employee\,[salary \Rightarrow integer;$
$\quad works\_for \Rrightarrow project\,]$

$project\,[title \Rightarrow string;$

$budget \Rightarrow integer;$
$is\_involved @ employee \Rightarrow boolean\,]$

Object base

$smith : person$

$jane : employee$

$natlang : project$

$dbopt : project$

$smith\,[name \rightarrow "John\ Smith";$
$\quad age \rightarrow 38\,]$

$jane\,[name \rightarrow "Eva\ Jane";$
$\quad age \rightarrow 22;$
$\quad salary \rightarrow 5100;$
$\quad works\_for \twoheadrightarrow \{dbopt, natlang\}\,]$

$natlang\,[title \rightarrow "Natural\ Language\ Processing";$
$\quad budget \rightarrow 50000\,]$

$dbopt\,[title \rightarrow "Database\ Optimization";$
$\quad budget \rightarrow 65000;$
$\quad is\_involved @ jane \rightarrow true\,]$

Note that class membership relations concerning simple objects, e.g. 38 : *integer, are omitted in the example.* □

In the schema part of example 4.1 we defined the classes *person*, *employee* and *project*. The is-a assertion states that *employee* is a subclass of *person*. The object molecules in the schema contain only signature expressions specifying argument and result types of methods and attributes. For example, *works_for* is a set-valued attribute in class *employee* returning a set of *project* objects for an employee. *is_involved* represents a scalar method with one argument of type *employee*. When it is applied to a project with a given employee it returns *true* or *false*.

The data part of example 4.1 describes the actual content of the database. Objects are identified by logical object identifiers. Is-a assertions represent class memberships. For instance, the example shows that Eva Jane identified by *jane* is an employee. For individual objects the values of attributes, or more generally the results of method invocations with some arguments, are given by data expressions , e.g. $name \rightarrow "Eva\ Jane"$ for employee *jane*, and $is\_involved @ smith \rightarrow true$ for project *dbopt*.

The connection between data expressions and signature expressions is not captured by the defintion of F-structures. This link is provided at a meta level by means of well-typing conditions. Informally, an F-program **P** is *well-typed* if canonic H-models of **P** obey the type restrictions given by signature expressions occuring in **P**. This means that (1) for any data expression on any object in **P** there exists a covering signature expression (i.e. the method can be invoked on the object with the given arguments), and (2) the result of such a data expression is of type prescribed by the covering signature expression. Note that the F-program of example 4.1 is well-typed.

Constraints

In database systems usually a set of integrity constraints is associated with a particular database to disallow invalid database states. F-logic itself does not define the concept of integrity (semantic) constraints. In [6], however, it has been extended for this purpose. We use the extension presented there. An *integrity constraint* in F-logic is an arbitrary F-formula. An *F-program with constraints* is a tuple $(\mathbf{P}, \mathbf{IC})$, where $\mathbf{P}$ is an F-program and $\mathbf{IC}$ is a set of integrity constraints (F-formulae). As a syntactic convention, integrity constraints are preceded by the symbol "!− ". The semantics of an F-program with constraints $(\mathbf{P}, \mathbf{IC})$ is defined by some canonic H-model of $\mathbf{P}$ that is also a model of $\mathbf{IC}$. In order to be a *well-typed F-program with constraints* $(\mathbf{P}, \mathbf{IC})$, $\mathbf{P}$ must be well-typed.

Example 4.2

Let's suppose that in the F-logic database of example 4.1 project titles must be unique. The corresponding F-program now is extended to become an F-program with constraints containing the single constraint as follows:

$$!-\ X_1 \doteq X_2 \longleftarrow X_1 : project\,[\,title \rightarrow Y\,] \wedge X_2 : project\,[\,title \rightarrow Y\,]$$

□

In the above example we used $X_1 : project\,[\,title \rightarrow Y\,]$ as a shorthand for $X_1 : project \wedge X_1 : [\,title \rightarrow Y\,]$ (with X_2 analogously). This kind of shorthand is often used in F-logic. The equality predicate $\doteq$ (defined in [19]) expresses that two objects are identical.

In section 6.2 some macros (shorthands) will be introduced for specifying key, mandatory and inverse constraints on F-logic level. Such constraints have to be generated during the transformation of information structures in several situations.

Lists and sets

For the transformation of PSM sequence types we need an F-logic construct that is usually known as *list*. F-logic itself is not provided with list data types, but lists can be modelled by defining a parametric family of classes, see [19] or [22]. We also need to translate PSM power types. Although the concept of set-valued attributes (methods) enable to manage sets, its applicability is limited. Defining attributes of nested sets (sets of sets) is not easy and natural. In [22] we defined a parametric family of classes for this purpose. From now on we assume that the aforementioned parametric classes are defined in each F-program, and $list(t)$ and $set(t)$, where t is a ground id-term denoting a class, can be used in F-programs whenever needed.

5 From PSM to F-logic: the design step

5.1 Framework

In this section we outline the essence of our approach to the transformation of conceptual models into F-logic (design step, see figure 2) by means of activity graphs with different decomposition levels. States denoted by ellipses are input and/or output of activities denoted by rectangles.

Basically, since the final goal is database schema generation, we are interested in schema (data structure + integrity constraints) transformation. However, the semantics of conceptual structures, and more particularly, the constraints are defined in terms of populations. This means that we have to deal with population transformation too. In fact, populations must be transformed anyway when our approach is integrated in an executable transformation mechanism concerning also operations and their transformation. Such an execution model is essential e.g. for optimization.

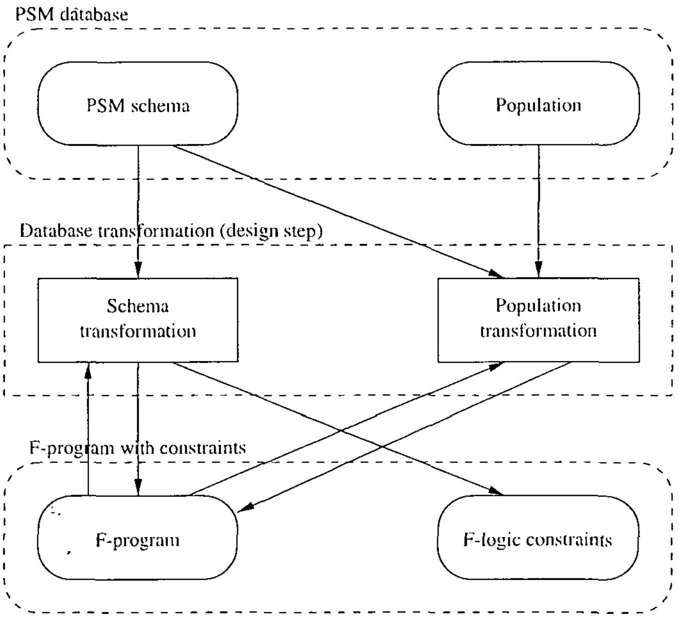

PSM database
PSM schema
Population
Database transformation (design step)
Schema transformation
Population transformation
F-program with constraints
F-program
F-logic constraints

Figure 4: The design step

As depicted in figure 4, a PSM schema $\Sigma = \langle \mathcal{I}, \mathcal{C} \rangle$ together with a population $\mathsf{Pop}_{\mathcal{I}}$ (PSM database) is translated to an F-program with constraints $(\mathbf{P}, \mathbf{IC})$. The activity graph of this figure already shows a first level decomposition to separate the schema from the population and their transformations. The population is represented in $\mathbf{P}$ (object base part), while the schema (structure + constraints) is represented in $\mathbf{P}$ (declaration part) as well as in $\mathbf{IC}$. For the transformation of populations and constraints (part of the schema) the result of the structure transformation is needed. That is the reason for the upward-directed arrows.

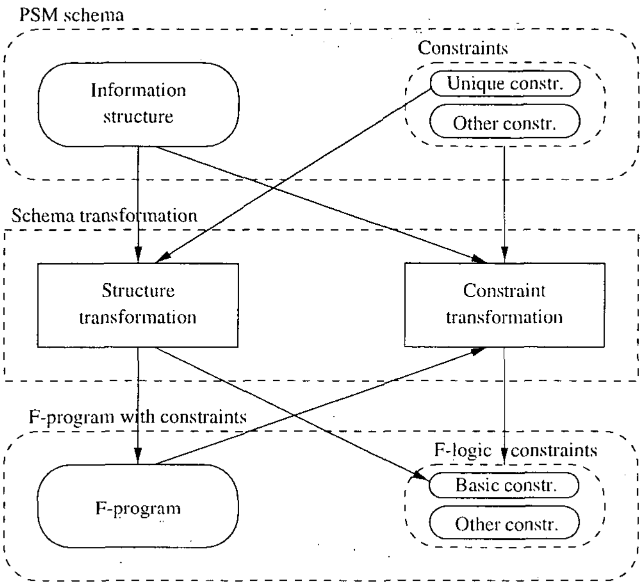


Figure 5: Decomposition of schema transformation

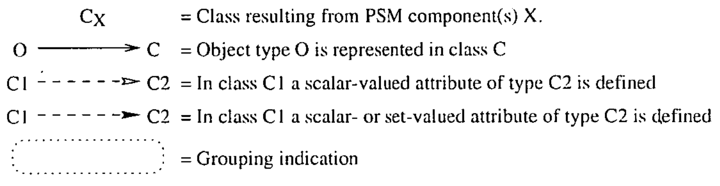
To illustrate the transformation of schemas in more detail, the corresponding part of the diagram of figure 4 is decomposed as depicted in figure 5. The information structure $\mathcal{I}$ becomes (the declaration) part of **P** on the one hand, and some basic constraints (e.g. inverse constraints) are also generated. Figure 5 also shows that the transformation of structures is influenced by uniqueness constraints. Except simple uniqueness constraints (over single predicators), the conceptual constraints in $\mathcal{C}$ are translated to F-logic constraints in **IC** in general. The transformation of constraints takes both the conceptual structure and its internal counterpart as its input. In the rest of section 5 the transformation of information structures is discussed.

5.2 Some preliminary issues

When transforming information structures, for all possible constructs of PSM we have to define their counterparts in F-logic. Basically, an information structure is mapped to F-logic class definitions, i.e. a set of object molecules with signature expressions only. For simplicity, we do not deal with assigning concrete names to the obtained F-logic components. Instead, an abstract notational convention is used, indicating the kind of an F-logic component and the components of the information structure it resulted from. For example, a class is denoted by C_X, where X contains the PSM component(s) to which the class corresponds.

In F-logic no difference is made between attributes and methods of classes. An attribute is considered to be a method without arguments. This enables the uniform treatment of (stored or derived) attributes and general methods. However, in object-oriented database systems a clear distinction between them is often made. This will be respected during our transformation, which is important also because, due to the elimination of certain kinds of redundancy during information analysis, a conceptual (PSM) schema represents data to be stored. From now on, by attributes we mean *stored* attributes. The distinction between (stored) attributes and general methods, however, is made only on syntactic (notational) level, thus preserving their uniform treatment in F-logic. Attributes and methods will be denoted in the form of $Attr_X$ and $Meth_X$, respectively,

For certain kinds of PSM components the translation is quite straightforward, but for others several alternatives are possible. The basic PSM components to be transformed are: object types, predicators, specialisation and generalisation hierarchies. The notion of object types in PSM is very general, it covers a number of more specific concepts, such as entity types, label types, fact types, set types, sequence types. Below we discuss the transformation of different PSM constructs separately. Some (simple) solutions have counterparts presented in [6] and [21], where the transformation is discussed in terms of ER and BRM schemas, respectively. Transformation alternatives presented in the rest of this section is discussed by means of abstract figures. Figure 6 shows how arrows in such figures are interpreted.



Cx = Class resulting from PSM component(s) X.
O → C = Object type O is represented in class C
C1 - - -▷ C2 = In class C1 a scalar-valued attribute of type C2 is defined
C1 - - -► C2 = In class C1 a scalar- or set-valued attribute of type C2 is defined
= Grouping indication

Figure 6: Graphical notation for specifying transformation alternatives

5.3 Entity types

By default, entity types are translated to F-logic classes. The corresponding class *definitions* are object molecules with signature expressions only. As an initial step, an empty object molecule is defined for each entity type. The structure of the class corresponding to an entity type depends on how the fact types in which the entity type is involved are transformed.

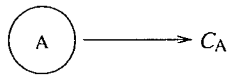


Figure 7: Entity types are mapped to classes

Sometimes an entity type simply models values of a label type connected with it (see e.g. entity type *Duration* in figure 3). In such cases the elimination of the counterpart class of the entity type is reasonable. Then the corresponding class is substituted with the concrete domain of the connected label type.

5.4 Label types and bridge types

A label type represents a set of simple values from a concrete domain. For simplicity, we assume that each conrete domain has a counterpart class built-in F-logic. Since the simple values represented by label types do not have independent existence and may only occur as part of more complex objects, for a label type L, in contrast with entity types, no separate F-logic class is created. It is mapped to the (assumed) built-in F-logic class that corresponds to its concrete domain $\mathsf{Dom}(L)$.

A bridge type is a special binary fact type connecting a non-label type with a label type. Assuming that the involved non-label type is mapped to a class, a bridge type is incorporated as an attribute in that class. If a uniqueness constraint is specified on the predicator with the non-label type as its base, then the attribute is scalar-valued. Otherwise, it is defined to be set-valued. Situations when our assumption does not hold will be mentioned and treated places where they may arise. The translation of bridge types and the related label types is illustrated in figure 8.

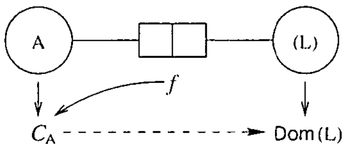


Figure 8: Mapping of bridge types and label types

5.5 Fact types and predicators

In this section we consider non-bridge fact types, when discussing the mapping of fact types into F-logic. The transformation of fact types is not straightforward, there are several possibilities how to transform them and their predicators. The translation of a fact type has a strong effect on the final F-logic counterparts of the object types involved in the fact type via predicators. Alternatives are discussed below.

5.5.1 Trivial mapping

The simplest solution is generating a separate F-logic (relation) class for each fact type. Then each predicator of a fact type results in a scalar-valued attribute in the corresponding class. Figure 9 illustrates this trivial translation.

This transformation is valid only if the base object type of each predicator in f has a corresponding F-logic class. However, this is not necessarily the case, e.g. if

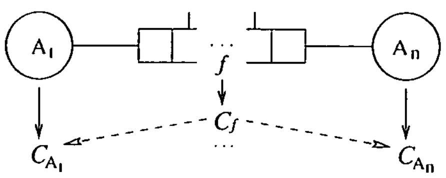


Figure 9: Illustration of trivial mapping

a base object type is a power type. Such cases will be discussed and treated later at the appropriate places.

Since the population of a fact type is a set of tuples, that are in turn *total* functions, it has to be ensured that each attribute of a fact (relationship) object carries some value. This can (has to) be forced by introducing appropriate constraints. On the other hand, fact objects are value-based and are identified by the participating objects. Therefore, a constraint has to be generated requiring that no two different fact objects may correspond to the same combination of participating objects (see also [6]).

Although the trivial mapping is sufficient to store all data, in order to improve query performance additional inverse attributes can be defined in any/all of the classes that correspond to the participating object types. Then such an inverse attribute stores references to relationship objects in which they participate (see also [21]). The type of inverse attributes is influenced by uniqueness constraints. Moreover, if inverse attributes are introduced, then also inverse constraints have to be generated to guarantee integrity.

Introducing inverse attributes with appropriate inverse constraints is a general option in the transformation. In principle, they can be generated in all alternatives that will be discussed for the transformation of fact types. In the sequel we will not explicitly mention the possibility again and again.

5.5.2 Incorporation of fact types

Fact types can be translated in such a way that they are incorporated in classes obtained for their object types, e.g. classes for entity types (see also [21]). In this case fact types are represented as reference attributes in such classes. More precisely, those attributes correspond to predicators constituting fact types.

Binary fact types

First we consider binary fact types as subjects of incorporation. As shown infigure 10, a binary fact type f can be incorporated in any/both of the two classes corresponding to its two base object types. The actual kind of reference attributes (scalar-valued or set-valued) is guided by uniqueness constraints. If f is incorporated in both classes, then an inverse constraint is needed as well.

Note that this way of transforming binary fact types looks very much like the

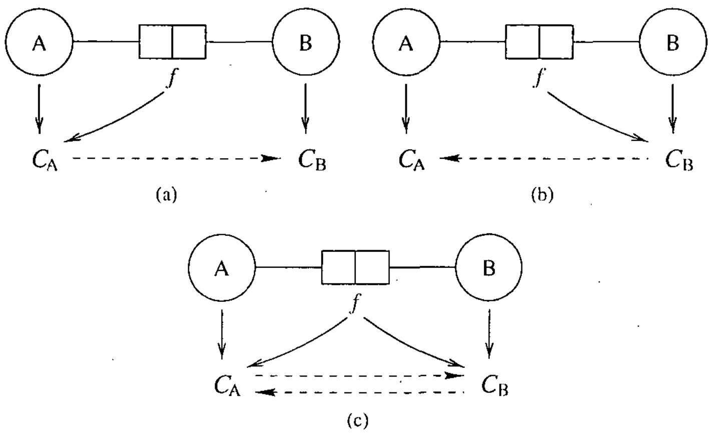


Figure 10: Incorporation of binary fact types

transformation of bridge types (see section 5.4), which is not surprising, since bridge types are always incorporated in the classes for the non-label types involved.

Relational view of incorporation

For binary fact types the incorporation mechanism can be combined with the trivial mapping as follows. A class is introduced for the fact type f similarly to the case of trivial mapping. At the same time, however, one of the two base object types is chosen, around which the related objects are arranged, similarly to the case of incorporation. Then the predicator with the "central" base object type becomes a scalar-valued attribute in the class corresponding to the fact type. The other predicator becomes either a set-valued or a scalar-valued attribute. This new alternative is depicted in figure 11.

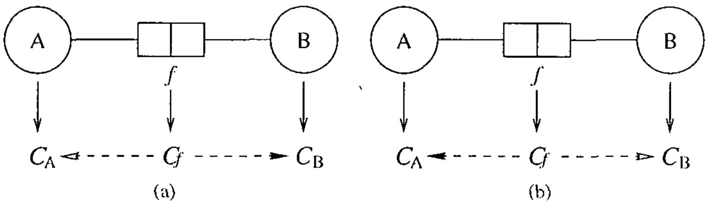


Figure 11: Relational incoporation of binary fact types

Note that if a uniqueness constraint is specified on the predicator with the "central" base object type, then the result is indentical to that of the trivial mapping. The combination of trivial mapping with incorporation can be viewed as a nest operation performed on the result of the trivial mapping. The basic constraints mentioned there are also needed here, but they have to be adapted according to the different structure.

We note that basically this alternative has relational nature. Since constructs from object-orientation are involved as well (set-valued attributes) it fits in the object-relational approach.

Fact types of higher degree

An n-ary fact type f ($n > 2$) can be incorporated in a class corresponding to a base object type of a predicator in f, for which a uniqueness constraint is specified (provided that each object type involved in f has a counterpart class, for now it is assumed). The situation is depicted in firgure 12.

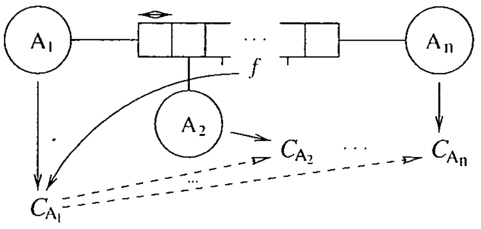


Figure 12: Incorporation of n-ary fact types

As opposed to the binary case, here the uniqueness constraint is required, because without that the concrete relationships between objects cannot be stored unambigously. The result of this transformation is that a class in which f is incorporated will have scalar-valued attributes for referencing related objects. The basic constraints discussed for trivial mapping have to be defined conforming this structure. If f is incorporated in more than one class, then inverse constraints are needed as well.

For the transformation of binary fact types we considered the combination of trivial mapping and incorporation. Since to incorporate fact types of higher degree uniqueness constraints over single predicators are required, we would get back exactly the trivial mapping when applying such a combination. Therefore, this alternative is not considered at all.

Quasi-incorporation

Although our latest note is valid, for n-ary fact types, where $n > 2$, a way to combine incorporation with the relational view can be recognized. The mechanism is slightly different from what we had for binary fact types. It is shown in figure 13. Fact type f is represented in a class for one of its base object types as well as in a subsidiary class denoted by $C_{f'}$. The class C_{A_1} in which f is incorporated has an attribute (scalar- or set-valued) for referencing objects of class $C_{f'}$. Those objects represent combinations of other objects that are related to a given C_{A_1} object via f. The attribute in C_{A_1} is scalar-valued if uniqueness constraint is on the predicator with base A_1, otherwise set-valued. As usual, structure conforming basic constraints are needed.

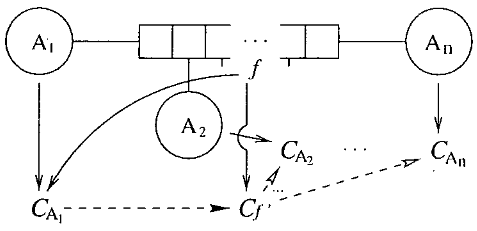


Figure 13: Illustration of quasi-incorporation

Note that, in contrast with pure incorporation of n-ary fact types, here no uniqueness constraint has been required. That is, this kind of transformation is more generally applicable. Note furthermore that quasi-incorporation of binary fact does not make sense, since it would produce a subsidiary class, as an unnecessary extra shell, with a single attribute.

5.5.3 Grouping

Since during information analysis fact types are determined such that they represent elementary recording types ([26]), most of them are binary or ternary in practice. That is, applying e.g. the trivial transformation would result in small but many object classes. Instead of transforming each fact type to a separate class, another alternative is to perform some grouping of fact types that are joinable via common object types before generating F-logic class definitions. Then for fact types in the same group a single class is created.

The grouping mechanism is implemented by means of a so-called *grouping profile*. A grouping *profile* is a set of groupings, where a *grouping* is a set of sets of predicators satisfying certain wellformedness conditions, e.g. conditions that require the joinability (predicators in a set have the same object type as their base) and connectivity of fact types to be grouped.

One particular grouping in the grouping profile implicitly specifies a set of fact types to be grouped together. The result of grouping is that one class is generated for the fact types in one group, and all those fact types are represented in that class. The structure of the class is obtained in such a way that one (scalar-vaued) attribute is defined for each set of predicators (corresponding to one object type) in the grouping and one (scalar-vaued) attribute is defined for each non-grouping predicator occuring in the grouped fact types.

The grouping meachnism is illustrated in figure 14. According to the figure, the grouping profile consists of a single (maximal) grouping. That grouping contains two sets of predicators (with common object types).

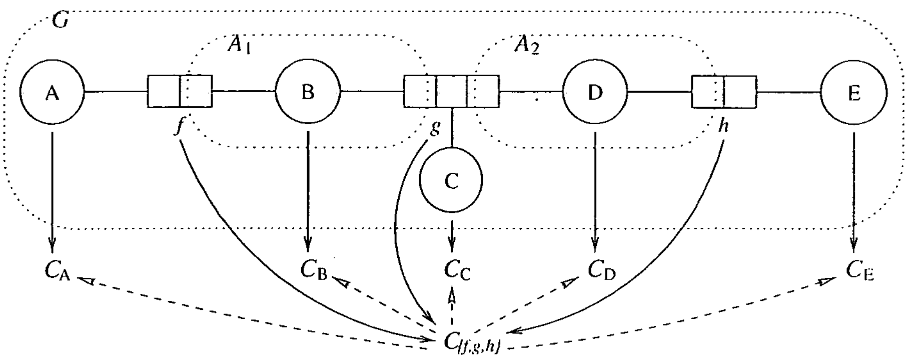


Figure 14: Illustration of the grouping mechanism

5.5.4 OR-like grouping

In section 5.5.2 we discussed the incorporation of fact types in classes corresponding to base object types. For binary fact types the relational variant, resulting from the combination of trivial mapping and incorporation, was also considered. In this section we show how this combined alternative can be further integrated with a restricted version of the grouping mechanism.

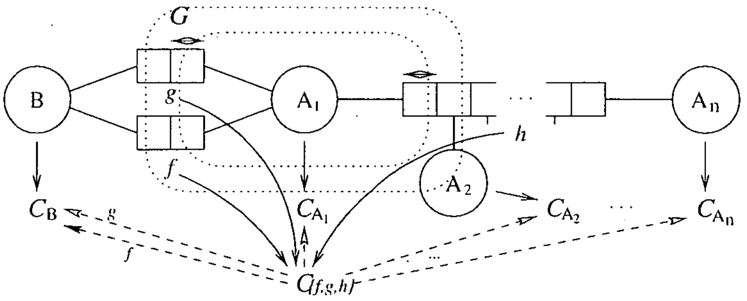


Figure 15: OR-like grouping

The situation is depicted in figure 15. The restrictions on the general grouping (having flat behavior) means that a particular grouping in the grouping profile may consist of only a single set of predicators with common base object type. (Note the difference with figure 14 with respect to the grouping indication.) Similarly to the relational incorporation for binary fact types (see figure 11), a central object type is chosen around which related data coming from several (grouped) fact types is arranged. This gives some OO characteristic to this transformation, while keeping also the relational view of grouping. From the alternatives discussed so far the general idea illustrated by figure 15 can be seen.

Grouping with quasi-incorporation

In figure 13 an alternative way to incorporate fact types of higher degree (called quasi-incorporation) with some relational characteristic was illustrated. There subsidiary classes have been introduced. That mechanism can be combined with (restricted) grouping analogously with the combination presented in the previous section, as shown in figure 16. Since only alternatives discussed earlier are combined in a way that has also been described, we do not explain this combination in more detail.

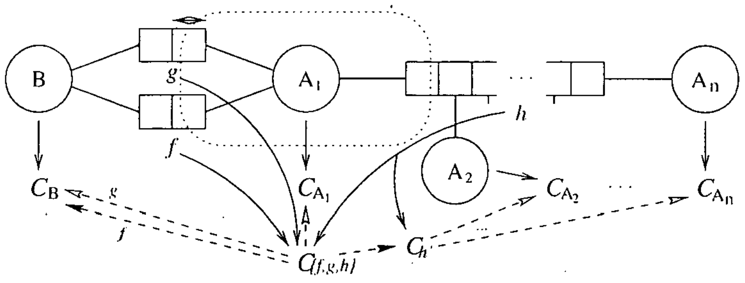


Figure 16: Grouping with quasi-incorporation

5.5.5 Objectification

As said in section 3, a fact type is objectified if it is the base object type of some predicator. The transformation of objectified fact types must be examined with some special care, since it has to be ensured that data attached to facts (and not e.g. to entities) are storable as well. Now we consider how the alternatives for the transformation of fact types are applicable to objectified fact types.

Objectification and trivial mapping

Obviously, the trivial mapping method can be applied to an objectified fact type as to any fact type without problems, which is the most natural solution. During the transformation of the rest of the information structure any of the alternatives can be chosen. Then the (objectified) fact type behaves as if it was an entity type.

Objectification and grouping

In principle, an objectified fact type can be considered for grouping, but not as it stands. In order to be able to transform the other fact types it is involved in, first it has to be unnested with respect to all those fact types. In many cases, however, the same final result can be obtained by applying different transformation alternatives. Therefore, we do not deal with providing an unnest operation for objectifications and do not consider objectified fact types to be grouped in any way.

Incorporation of objectified fact types

In section 5.5.2 the alternative of incorporating fact types in classes for their base object types was discussed. Binary fact types can always be incorporated, but fact types of higher degree can be considered for incorporation at presence of uniqueness constraints over single predicators. Incorporation of objectified fact types is a meaningful option in even more restrictive situations only. The rationale behind this is that beside representation of (objectified) facts, we also have to deal with the representability of facts attached to (objectified) facts.

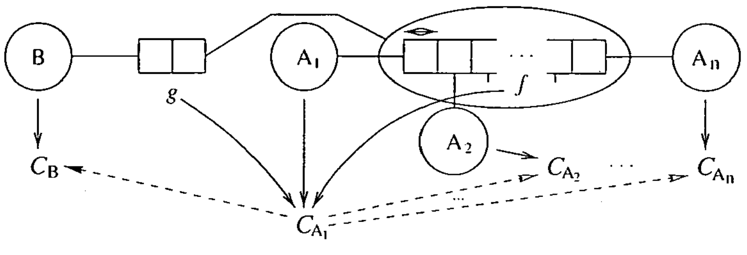


Figure 17: Incorporation of objectified fact type

The mechanism of incorporating objectified fact types is illustrated in figure 17. It shows that if an objectified fact type f is incorporated in the counterpart class of one of its base object types, then the fact types in which f participates are also incorporated in the same class. The unambigous representability is ensured by the required uniqueness constraint (also when f is binary).

Obviously, this transformation requires the fact types with base object type f to fulfill the condition for incorporation. If for any of those fact types this precondition is not satisfied, then f cannot be incorporated either. This leads to a situation that can be captured by recursive checking and evaluation.

At some earlier points of this paper we assumed the existence of classes for base object types when describing transformation alternatives for fact types (also bridge types). We also promised to highlight and treat situations when it's not the case. Now we reached such a situation, because the incorporation mechanism is applied to fact types with base f, but no class for f exists. The special treatment here is that we explicitly prescribed the class in which the incorporation has to be performed instead of the class for f. Note, however, that during the incorporation of f the existence of classes for its base object types is still an assumption (precondition).

The relational-like incorporation for binary fact types (see figure 11) is generally not applicaple to objectified binary fact types, only when the trivial mapping is obtained back due to the required uniqueness constraint. Otherwise the problem of unambigous representability arises.

For similar reason, quasi-incorporation of objectified fact types is only possible at the presence of appropriate uniqueness constraints. The illustrating figure is obtained as a combination of figures 13 and 17 and is presented in figure 18 without further explanation.

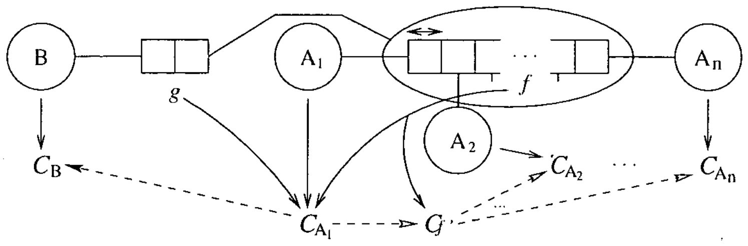


Figure 18: Quasi-incorporation of objectified fact type

5.6 Power types

Power types in PSM represent sets. An instance of a power type is a (non-empty) set of instances of its element type and is identified by its elements. Since in F-logic we have the concept of set-valued attribute, at first sight it might seem that power types can be represented simply by means of set-valued attributes.

However, set-valued attributes have limited applicability. Nested power types (power types of power types) cannot be expressed in F-logic in terms of set-valued attributes. For instance, in our example in figure 3 if budgets belong to groups of groups of projects, then that situation cannot be represented by set-valued attributes. Therefore, for the transformation of power types the parametric class *set*(C) (discussed in section 4), where the parameter C can be an arbitrary class, will be used.

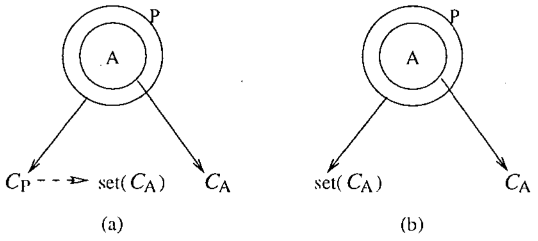


Figure 19: Alternatives for power types

A power type can be transformed such that it is simply represented as a *set*(...) type wherever it occurs. An alternative solution is that a separate class in defined for the power type with one attribute containing the set elements and possibly other attributes holding references to related objects. The choice depends on its context and/or the designer's preference. This consideration, however, suggests that the transformation of power types with or without defining counterpart classes can be both reasonable solutions, which is shown in figure 19. Our manner of transforming a power type P covers both alternatives in a uniform way. The procedure is as follows:

1. First, as an initial step, a class C_P is introduced with a single scalar-valued attribute $Attr_{elements}$ of type $set(C_{\mathsf{Elt}(P)})$ for containing the set elements.
2. During the transformation of the information structure consider P as if it was an entity type (remember that entity types are always mapped to classes).
3. When the whole information structure has been translated, C_P will or will not contain attributes other than $Attr_{elements}$. If C_P contains no attribute other than $Attr_{elements}$, then optionally the class C_P can be eliminated by substituting it with $set(C_{\mathsf{Elt}(P)})$ where it occurs.

As it can be seen, the translation of a power type requires its element type (not necessarily entity type, it can be e.g. a fact type) to be mapped to a class. To take this into account, it is required that if a fact type is the element type of a power type, then the fact type has to result in a class, i.e. it has to be transformed according to the trivial mapping.

Furthermore, note that although at the end power types do not necessarily result in classes, the assumption, made at several places before, that corresponding classes for participating object types exist when fact types are mapped (see section 5.5) is not violated, because the elimination of the counterpart class for a power type may be performed as a final step only.

5.7 Sequence types

The transformation of sequence types into F-logic is analogous to that of power types, see figure 20. The only differences are that (1) instead of the parametric class $set(C)$ the parametric class $list(C)$ is used, and (2) the implicit attribute is denoted by $Attr_{sequence}$ instead of $Attr_{elements}$. In other aspects the procedure is identical, therefore is not further detailed here. Moreover, similar considerations and notes are valid for the mapping of sequence types what we had for the transformation of power types.

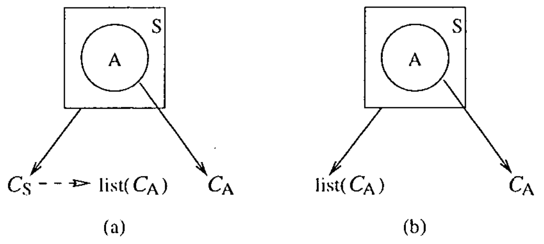


Figure 20: Alternatives for sequence types

5.8 Specialisation and generalisation

Specialisation and generalisation hierarchies of PSM models are translated to subclass hierarchies in F-logic. Both $A\,\mathsf{Spec}\,B$ and $B\,\mathsf{Gen}\,A$ result in $C_A :: C_B$. (C_A is subclass of C_B). As a consequence, participating object types have to be mapped to classes. Taking the restrictions on Spec and Gen (see section 3) into account this means that: (1) If B is a fact type, then it has to be translated according to trivial mapping. (2) If B is either a power type or a sequence type, then its corresponding class cannot be eliminated.

So far, specialisation and generalisation were treated in the same way. However, they are different concepts, and the difference has to be reflected on F-logic level as well. As mentioned in section 3, the difference lies in the way subtypes and generalised object types get their identification, and in the way their population is derived. A generalised object type inherits identification from its specifiers and its population is the union of the populations of its specifiers. Therefore, in an F-logic class hierarchy that corresponds to a generalisation hierarchy non-leaf level classes may have only object instances that belong to some leaf level class. This can be achieved by introducing appropriate constraints.

A subtype inherits identification from its supertype and its population is a subset of the population of its subtype and is derived by means of a subtype defining rule. When mapping a subtype relationship, the associated subtype defining rule has be translated too, resulting in a rule in the corresponding F-program. The body of the rule is the subtype defining rule translated into F-logic, while the head is an is-a assertion specifying class membership.

5.9 Methodization

In general, an attribute of a class carries direct reference(s) to related object(s), which means that by means of attributes (methods without arguments) only relationships between two objects can be captured. However, it would often be useful to have a device to enable that objects related to a given combination of other objects can be obtained. This can be achieved by defining general methods (method with arguments). A mechanism, called *methodization*, can be introduced for this purpose as a complementary device in order to provide efficient access (querying) of data stored in attributes of classes.

For example, provided that a fact type is mapped to a class, facts (relationships) become stored objects. The attributes of a relationship object contain references to objects that are in relationship. If now we want to know what objects of a given class are in relationship with some combination of objects of some classes, we have to perform an appropriate query. Clearly, it can be very useful to define access *methods* to make querying (traversing relationships) easier and more efficient.

The scale of choices for methods is quite wide. The signatures of methods are strongly influenced by uniqueness constraints. The behavior of methods can be defined in terms of F-logic rules. In [6] a similar technique is introduced, which is called *pivoting*.

Inverse attributes vs methodization

It was said earlier that inverse attributes can be optionally defined beside attributes needed for sufficient storage of data. It improves query performance, but makes update more complex. To ensure integrity, introducing inverse attributes must be done along with generating inverse constraints.

Alternatively, stored inverse attributes can be substituted with *methods* without arguments. In this case the inverse constraints needed for the inverse attributes are converted to rules in the F-program. Those rules define the behavior of the (inverse) methods.

6 Elaborated example

Until now we discussed possible ways to transform conceptual structures to OO database schemas. In this section we transform the PSM schema in figure 3 into F-logic. During this mapping object molecules defining only parts of classes are obtained often. Since, according to the semantics of F-logic, they can be unified to get equivalent more complex object molecules, at the end we will also present the unified result.

6.1 Declarations

Entity types are mapped to classes. As an initial step, for each entity type a an empty object molecule is generated, i.e. we obtain:

$$\begin{array}{lllll} C_{Equipment}\,[\,] & C_{Car}\,[\,] & C_{PC}\,[\,] & C_{Person}\,[\,] & C_{Manager}\,[\,] \\ C_{Coworker}\,[\,] & C_{Dept.}\,[\,] & C_{Building}\,[\,] & C_{Project}\,[\,] & C_{Article}\,[\,] \\ C_{Company}\,[\,] & C_{Activity}\,[\,] & C_{Duration}\,[\,] & C_{Amount\_of\_money}\,[\,] & \end{array}$$

Furthermore, in the initial step a class is created for each power type and sequence type with a single attribute for containing the set and list elements, respectively:

$$C_{Pr\_group}\,[\,Attr_{elements} \Rightarrow set(C_{Project})\,]$$
$$C_{Daily\_activities}\,[\,Attr_{sequence} \Rightarrow list(C_{Activity})\,]$$

Specialisation and generalisation relationship result in subclass relationships:

$$\begin{array}{ll} C_{Manager} :: C_{Person} & C_{Coworker} :: C_{Person} \\ C_{Car} :: C_{Equipment} & C_{PC} :: C_{Equipment} \end{array}$$

Bridge types are incorporated in classes obtained for the involved non-label type. Suppose that with the occuring label types the following concrete domains, that are assumed to be built-in classes in F-logic, are associated:

$$\mathsf{Dom}(Date) = Date$$
$$\mathsf{Dom}(Dollars) = \mathsf{Dom}(Hours) = Integer$$
$$\mathsf{Dom}(Reg\_nr) = \mathsf{Dom}(PC\_nr) = \mathsf{Dom}(P\_id) = \mathsf{Dom}(P\_name) = String$$
$$\mathsf{Dom}(Pr\_name) = \mathsf{Dom}(D\_name) = \mathsf{Dom}(B\_code) = \mathsf{Dom}(A\_code) = String$$
$$\mathsf{Dom}(C\_code) = \mathsf{Dom}(C\_name) = \mathsf{Dom}(Tel\_nr) = \mathsf{Dom}(A\_descr) = String$$

For identifying bridge types the predicator with non-label base object type X is referred to as has_X. Then the mapping of bridge types yields the following object molecules:

$$C_{Person}\,[\,Attr_{has\_P\_id} \Rightarrow String\,] \qquad C_{Person}\,[\,Attr_{has\_Pname} \Rightarrow String\,]$$
$$C_{Equipment}\,[\,Attr_{bought\_on} \Rightarrow Date\,] \qquad C_{Car}\,[\,Attr_{has\_Reg\_no} \Rightarrow String\,]$$
$$C_{PC}\,[\,Attr_{has\_PC\_nr} \Rightarrow String\,] \qquad C_{Project}\,[\,Attr_{has\_Pr\_name} \Rightarrow String\,]$$
$$C_{Dept.}\,[\,Attr_{has\_D\_name} \Rightarrow String\,] \qquad C_{Building}\,[\,Attr_{has\_B\_code} \Rightarrow String\,]$$
$$C_{Article}\,[\,Attr_{has\_A\_code} \Rightarrow String\,] \qquad C_{Activity}\,[\,Attr_{has\_A\_descr} \Rightarrow String\,]$$
$$C_{Duration}\,[\,Attr_{has\_Hour} \Rightarrow Integer\,] \qquad C_{Am.\_of\_m.}\,[\,Attr_{has\_Dollars} \Rightarrow Integer\,]$$
$$C_{Company}\,[\,Attr_{has\_C\_code} \Rightarrow String\,] \qquad C_{Company}\,[\,Attr_{has\_Cname} \Rightarrow String\,]$$
$$C_{Company}\,[\,Attr_{has\_Telnr} \Rightarrow\!\!\!\Rightarrow String\,]$$

Finally, we transform (non-bridge) fact types. In section 5.5 a number of alternatives were discussed. We transform the fact types of figure 3 such that we cover as many alternatives as reasonable according to the complexity of the input schema. In order to identify chosen alternatives unambigously we always refer to the corresponding figures. The specified uniqueness constraints, of course, are taken into account.

The objectified fact type *Coworkership* is mapped according to the trivial mapping (figure 9), which results in:

$$C_{Coworkership}\,[\,Attr_{works\_for} \Rightarrow C_{Person};\; Attr_{has\_as\_coworker} \Rightarrow C_{Project}\,]$$

The binary fact type *Cow_dur* is incorporated (figure 10) in the class obtained for the base object type of its predicator *has_as_duration* yielding:

$$C_{Coworkership}\,[\,Attr_{has\_as\_duration} \Rightarrow C_{Duration}\,]$$

Incorporation of binary fact types in one of the two classes corresponding to their base object types is applied to other fact types as well. Fact types *Salary*, *Activities* and *Budget* are incorporated in classes $C_{Coworker}$, $C_{Manager}$ and C_{Pr_group}, respectively. The following object molecules are generated:

$$C_{Coworker}\,[\,Attr_{earns} \Rightarrow C_{Amount\_of\_money}\,]$$
$$C_{Manager}\,[\,Attr_{performes} \Rightarrow C_{Daily\_activities}\,]$$
$$C_{Pr\_group}\,[\,Attr_{may\_spend} \Rightarrow C_{Amount\_of\_money}\,]$$

Fact type *Car_usage* is incorporated in both classes corresponding to the involved object types (figure 10, alternative (c)), namely in C_{Car} and $C_{Manager}$. Two object molecules are obtained:

$$C_{Car}\,[\,Attr_{used\_by} \Rightarrow C_{Manager}\,]$$
$$C_{Manager}\,[\,Attr_{uses} \Rightarrow C_{Car}\,]$$

In figure 11 a relational-like way of incorporating binary fact types, as the combination of trivial mapping and incorporation (alternatively seen as nesting on the result of trivial mapping), has been shown. Fact type *Location* is mapped according to this way of transformation. Object type *Dept.* is chosen as the central object type. This results in:

$$C_{Location}\,[\,Attr_{is\_located\_at} \Rightarrow C_{Dept.};\;\; Attr_{accomodates} \Rrightarrow C_{Building}\,]$$

To illustrate the alternative coming from the combination of incorporation (OO nature) with restricted grouping (relational nature), fact types *Employment* and *PC_usage* with common object type *Person* are grouped according to the OR-like grouping shown in figure 15. The following single class (object molecule) is generated:

$$C_{\{Employment,\, PC\_usage\}}\,[\,Attr_{\{employed\_by,\, works\_on\}} \Rightarrow C_{Person};$$
$$Attr_{employs} \Rightarrow C_{Dept.};\;\; Attr_{belongs\_to} \Rrightarrow C_{PC}\,]$$

The combination of OR-like grouping and quasi-incorporation has been depicted in Figure 16. To set an example for this mechanism, fact types *Management* and *Supply* are grouped together via object type *Project* such that for fact type *Supply* a subsidiary class $C_{Supply'}$ is introduced. Beside this subsidiary class, a relation class is also defined representing fact type *Management* and partially fact type *Supply*. The obtained object molecules are the following:

$$C_{Supply'}\,[\,Attr_{supplies} \Rightarrow C_{Company};\;\; Attr_{is\_supplied} \Rightarrow C_{Article}\,]$$

$$C_{\{Management,\, Supply\}}\,[\,Attr_{\{managed\_by,\, receives\}} \Rightarrow C_{Project};$$
$$Attr_{manages} \Rrightarrow C_{Person};\;\; Attr_{receives} \Rrightarrow C_{Supply'}\,]$$

By now, each fact type of figure 3 has been translated. We did not exploit every particular alternative discussed in section 5.5 coming from different kinds of combinations. However, all the basic building blocks used in some combination, such as trivial mapping, incorporation, quasi-incorporation, grouping, have been covered.

Class elimination

After completing the transformation of the PSM structure of figure 3, the elimination of classes for power types and sequence types can be considered (see sections 5.6 and 5.7). Since the class C_{Pr_group} contains an attribute other than $Attr_{elements}$, it cannot be eliminated. The elimination of class $C_{Daily\_activities}$, however, is reasonable. The class is substituted with $list(C_{Activity})$ and is removed.

As mentioned in section 5.3, classes for entity types with label type nature can also be eliminated. In our example such classes are $C_{Duration}$ and $C_{Amount\_of\_money}$ at least. They are, therefore, eliminated. Occurrences are replaced with the built-in classes for the concrete domains of their corresponding label types.

Unified declarations

Due to the fact that in some cases above different parts of class definitions were obtained at different points of the transformation, some classes are defined by means of more than one object molecule. The separate parts can now be put together as presented below. The class eliminations above are taken into account. Clearly, the input schema does not cover all the details of the application domain, which lead to simple (entity) classes in many cases. Those classes likely have additional attributes, which is also indicated below. During the above transformation we used denotations rather than names for classes and attributes. In order to make the example more readable we also give names to classes and attributes now.

$$Person\,[\,P\_id \Rightarrow String;\;\; name \Rightarrow String; \ldots\,]$$

$$Manager :: Person$$

$$Manager\,[\,daily\_activities \Rightarrow list(Activity);\;\; uses\_car \Rightarrow Car; \ldots\,]$$

$$Coworker :: Person$$

$$Coworker\,[\,salary \Rightarrow Integer; \ldots\,]$$

$$Equipment\,[\,bought\_on \Rightarrow Date; \ldots\,]$$

$$Car :: Equipment$$

$$Car\,[\,reg\_nr \Rightarrow String;\;\; used\_by \Rightarrow Manager; \ldots\,]$$

$$PC :: Equipment$$

$$PC\,[\,pc\_nr \Rightarrow String; \ldots\,]$$

$$Project\,[\,title \Rightarrow String; \ldots\,]$$

$$Department\,[\,name \Rightarrow String; \ldots\,]$$

$$Building\,[\,B\_code \Rightarrow String; \ldots\,]$$

$$Article\,[\,A\_code \Rightarrow String; \ldots\,]$$

$$Activity\,[\,description \Rightarrow String; \ldots\,]$$

$$Company\,[\,C\_code \Rightarrow String;\;\; name \Rightarrow String;\;\; tel\_nrs \Rightarrow\!\!\Rightarrow String; \ldots\,]$$

$$Pr\_group\,[\,projects \Rightarrow set(Project);\;\; budget \Rightarrow Integer; \ldots\,]$$

$$Coworkership\,[\,employee \Rightarrow Person;\;\; project \Rightarrow Project;\;\; duration \Rightarrow Integer\,]$$

$$\begin{aligned}
&Dept\_loc\,[\,dept \Rightarrow Department;\ \ building \Rightarrow\!\!\!\Rightarrow Building\,]\\
&Employment\,[\,employee \Rightarrow Person;\, dept \Rightarrow Department;\, works\_on \Rightarrow\!\!\!\Rightarrow PC\,]\\
&Supply\,[\,supplier \Rightarrow Company;\ \ article \Rightarrow Article\,]\\
&Project\_rel\,[\,project \Rightarrow Project;\ \ managers \Rightarrow\!\!\!\Rightarrow Person;\ \ receives \Rightarrow\!\!\!\Rightarrow Supply\,]
\end{aligned}$$

6.2 Constraints

The result of the transformation of the information structure in figure 3 was influenced by simple conceptual uniqueness constraints. On the other hand, in addition to class definitions, the structure transformation results in some kinds of basic constraints in F-logic, such as key, mandatory and inverse constraints. In this section we provide these constraints for our example.

Uniqueness constraints

F-logic uniqueness (key) constraints are obtained in two ways. Firstly, from the unique representation of facts (relationships) (see also [6]). Secondly, the simple conceptual uniqueness constraints in figure 3 are translated as well. For uniqueness constraints the macro "$!\!-\ \mathsf{Key}(C, \{A_1, ..., A_n\})$" is defined as follows:

$$\begin{aligned}
!\!-\ X \doteq Y \ \longleftarrow\quad & X : C\,[\,A_1 \rightarrow(\twoheadrightarrow)\ R_1; ...; A_n \rightarrow(\twoheadrightarrow)\ R_n\,]\ \wedge\\
& Y : C\,[\,A_1 \rightarrow(\twoheadrightarrow)\ R_1; ...; A_n \rightarrow(\twoheadrightarrow)\ R_n\,]
\end{aligned}$$

The notation $A_i \rightarrow(\twoheadrightarrow)\ R_i$ means that if A_i is scalar-valued, then $\rightarrow$ is used, otherwise $\twoheadrightarrow$. In our example the following F-logic uniqueness constraints are generated:

$$\begin{array}{ll}
!\!-\ \mathsf{Key}(Person, \{P\_id\}) & !\!-\ \mathsf{Key}(Manager, \{uses\_car\})\\
!\!-\ \mathsf{Key}(Car, \{reg\_nr\}) & !\!-\ \mathsf{Key}(Car, \{used\_by\})\\
!\!-\ \mathsf{Key}(PC, \{pc\_nr\}) & !\!-\ \mathsf{Key}(Project, \{title\})\\
!\!-\ \mathsf{Key}(Department, \{name\}) & !\!-\ \mathsf{Key}(Building, \{B\_code\})\\
!\!-\ \mathsf{Key}(Article, \{A\_code\}) & !\!-\ \mathsf{Key}(Activity, \{description\})\\
!\!-\ \mathsf{Key}(Company, \{C\_code\}) & !\!-\ \mathsf{Key}(Pr\_group, \{projects\})\\
!\!-\ \mathsf{Key}(Cow.ship, \{employee, project\}) & !\!-\ \mathsf{Key}(Dept\_loc, \{dept\})\\
!\!-\ \mathsf{Key}(Employment, \{employee\}) & !\!-\ \mathsf{Key}(Employment, \{works\_on\})\\
!\!-\ \mathsf{Key}(Supply, \{supplier, article\}) & !\!-\ \mathsf{Key}(Project\_rel, \{project\})
\end{array}$$

Mandatory constraints

Since the population of fact types consists of total functions, it has to be ensured that if a class corresponds to a fact type, then each attribute of a member of that class (fact object) carries some value. Again, a general macro

"$!-$ $\mathsf{Mandatory}(C, \{A_1, ..., A_n\})$" is introduced to serve as a shorthand for the following:

$$!-\ (\exists Y)X\,[\,A_1 \rightarrow(\twoheadrightarrow)\ Y\,] \longleftarrow X : C$$
$$\ldots$$
$$!-\ (\exists Y)X\,[\,A_n \rightarrow(\twoheadrightarrow)\ Y\,] \longleftarrow X : C$$

In the case of our example the following mandatory constraints are necessary:

$$!-\ \mathsf{Mandatory}(Coworkership, \{employee, project\})$$
$$!-\ \mathsf{Mandatory}(Dept\_loc, \{dept, building\})$$
$$!-\ \mathsf{Mandatory}(Supply, \{supplier, article\})$$

Inverse constraints

When two attributes are considered to be inverses of each other, inverse constraints have to be defined. According to the possible kinds of relationship types between two classes we introduce the following three macros:

$$\begin{aligned} !-\ \mathsf{Inverse1\text{-}1}(C_1, A_1, C_2, A_2) \overset{\text{def}}{=}\ & !-\ Y : C_2\,[\,A_2 \rightarrow X\,] \longleftarrow X : C_1\,[\,A_1 \rightarrow Y\,] \\ & !-\ Y : C_1\,[\,A_1 \rightarrow X\,] \longleftarrow X : C_2\,[\,A_2 \rightarrow Y\,] \end{aligned}$$

$$\begin{aligned} !-\ \mathsf{Inverse1\text{-}N}(C_1, A_1, C_2, A_2) \overset{\text{def}}{=}\ & !-\ Y : C_2\,[\,A_2 \rightarrow X\,] \longleftarrow X : C_1\,[\,A_1 \twoheadrightarrow Y\,] \\ & !-\ Y : C_1\,[\,A_1 \twoheadrightarrow X\,] \longleftarrow X : C_2\,[\,A_2 \rightarrow Y\,] \end{aligned}$$

$$\begin{aligned} !-\ \mathsf{InverseM\text{-}N}(C_1, A_1, C_2, A_2) \overset{\text{def}}{=}\ & !-\ Y : C_2\,[\,A_2 \twoheadrightarrow X\,] \longleftarrow X : C_1\,[\,A_1 \twoheadrightarrow Y\,] \\ & !-\ Y : C_1\,[\,A_1 \twoheadrightarrow X\,] \longleftarrow X : C_2\,[\,A_2 \twoheadrightarrow Y\,] \end{aligned}$$

In our example fact type *Car_usage* has been incorporated in both classes obtained for the two involved entity types, namely in classes *Manager* and *Car*, yielding one attribute in each being inverses of each other. Therefore, an inverse constraint is also required as follows:

$$!-\ \mathsf{Inverse1\text{-}1}(Manager, uses\_car, Car, used\_by)$$

Effects of specialisation and generalisation

The population of subtypes (specialisation) is defined by subtype defining rule. This mechanism has to be translated into F-logic. In our example we have $Manager\ \mathsf{Spec}\ Person$ and $Coworker\ \mathsf{Spec}\ Person$. Let $\Psi_{Manager}$ and $\Psi_{Coworker}$ denote the F-logic counterpart of the subtype defining rules for subtypes *Manager* and *Coworker*, respectively. Then we introduce the following two constraints:

$$!-\ X : Manager \longleftarrow X : Person \ \wedge\ \Psi_{Manager}(X)$$
$$!-\ X : Coworker \longleftarrow X : Person \ \wedge\ \Psi_{Coworker}(X)$$

The subtype defining rules are: A person is a manager if he/she plays the role *manages*. A person is a coworker if he/she plays the role *works_for*. It has effect on the final result of the translation of fact types *Management* and *Coworkership*. In class *Project_rel* the result type *Person* is replaced with *Manager*:

$$Project\_rel\,[\,project \Rightarrow Project;\ managers \Rightarrow\!\!\!> Manager;\ receives \Rightarrow\!\!\!> Supply\,]$$

In class *Coworkership* the result type *Person* is replaced with *Coworker*:

$$Cow.ship\,[\,employee \Rightarrow Coworker;\ project \Rightarrow Project;\ duration \Rightarrow Integer\,]$$

In case of generalisation it has to be ensured that every instance in the population of a generalised type belongs to the population of one of its specifiers. This can be expressed in terms of constraints on F-logic level. In our example we obtain:

$$!-\ (X : Car\ \vee\ X : PC)\ \longleftarrow\ X : Equipment$$

6.3 Inverse attributes and methods

As said in section 5.9, a wide range of introducing inverse attributes with inverse constraints as well as methods with the definition of their behavior is possible. To illustrate these general mechanisms, now we introduce an additional inverse attribute and a method. Class *Coworker* is augmented with attribute *involved_in* containing references to *Coworkership* objects in which a given person is involved. The following additional object molecule, that can be unified with the existing one for class *Coworker*, and inverse constraint are defined:

$$Coworker\,[\,involved\_in \Rightarrow\!\!\!> Coworkership\,]$$
$$!-\ \mathsf{InverseI\text{-}N}(Coworker, involved\_in, Coworkership, employee)$$

As an example of an access method, the method *suppliers* is introduced in class *Project* returning the companies that supply a given article for the project on which the method is invoked. The object molecule defining the signature of the method is the following:

$$Project\,[\,suppliers\ @\ Article \Rightarrow\!\!\!> Company\,]$$

The semantics of the method is given by means of the following F-logic rule:

$$\begin{aligned} Y\,[\,suppliers\ @\ W \rightarrow\!\!\!> V\,]\ \longleftarrow\quad & X : Project\_rel\,[\,project \rightarrow Y;\ receives \rightarrow\!\!\!> Z\,] \wedge \\ & Z : Supply\,[\,supplier \rightarrow V;\ article \rightarrow W\,] \end{aligned}$$

7 About the implementation step

The second step of the transformation (the implementation step, see figure 2) is the translation of intermediate models defined in terms of F-logic into a OO final target environment, e.g. SQL3 or ODMG-93. A sequence of DDL statements in the corresponding database language has to be generated from F-logic specifications. Unlike in the design step, in the implementation step separate translations have to be defined for different target environments, where all the system specific details must be dealt with. This can be implemented by means of translation tables, containing all the system specific information (e.g. supported data types, correspondece between F-logic "built-in" classes and system specific data types) needed for the generation of DDL statements.

Since in the present paper the main focus is on the design step, we do not further elaborate on the implementation step. However, in the section below an example is given in terms of ODMG-93.

7.1 Example in ODMG-93

Next the F-logic schema in section 6 obtained for the PSM schema of figure 3 is translated into ODMG-93. Beside class declarations (including the additional inverse attribute and method), uniqueness and inverse constraints are translated. Specifying other constraints is not supported directly in ODMG-93, therefore they are not considered here. For the syntax and semantics of ODMG-93 we refer to [9].

```
interface Person
(   extent Persons
    keys P_id   ) : persistent
{   attribute string P_id ;
    attribute string name ; ...  } ;

interface Manager : Person
(   extent Managers
    keys uses_car   ) : persistent
{   relationship List<Activity> daily_activities ;
    relationship Car used_car
            inverse Car:: used_by ; ...  } ;

interface Coworker : Person
(   extent Coworkers   ) : persistent
{   attribute integer salary ;
    relationship Set<Coworkership> involved_in
            inverse Coworkership:: employee ; ...  } ;

interface Activity
(   extent Activities
    keys description   ) : persistent
```

```
{   attribute string description ; ...  } ;

interface Project
(   extent Projects
    keys title   ) : persistent
{   attribute string title ;
    ...
    Set<Company> suppliers( in Article art ) ;   } ;

interface Article
(   extent Articles
    keys A_code   ) : persistent
{   attribute string A_code ; ...  } ;

interface Company
(   extent Companies
    keys C_code   ) : persistent
{   attribute string C_code ;
    attribute string name ;
    attribute Set<string> tel_nrs ; ...  } ;

interface Coworkership
(   extent Coworkerships
    keys (employee, project)   ) : persistent
{   attribute integer duration ;
    relationship Coworker employee
            inverse Coworker:: involved_in ;
    relationship Project project ;   } ;

interface Supply
(   extent Supplies
    keys (supplier, article)   ) : persistent
{   relationship Company supplier ;
    relationship Article article ;   } ;

interface Project_rel
(   extent Project_rels
        keys project  ) : persistent
    {   relationship Project project ;
        relationship Set<Manager> managers ;
        relationship Set<Supply> receives ;   } ;
```

8 Conclusions and further research

In this paper we dealt with the transformation of conceptual data models into database environments with object-oriented features, such as ODMG-93 and SQL3. In our approach this transformation is captured within the framework of a two level architecture. Conceptual models are first mapped to intermediate specifications (design step). Then the obtained intermediate specifications are translated into the database language of a given target database system (implementation step). For expressing conceptual models we used the object-role modelling technique PSM (Predicator Set Model), a formalized extension of NIAM. Intermediate specifications are expressed in terms of F-logic, a logic-based abstract specification language for object-oriented systems. The advantages of a two step tranformation mechanism have been discussed.

Here we focused on the first step of the overall transformation. A number of alternatives for the transformation of conceptual structures have been presented, resulting in a collection of design options. Such alternatives were discussed by means of illustrating figures. The mapping of information structures is often influenced by simple uniqueness constraints. Also, transforming structures often imply basic (e.g. key) integrity constraints in the target model to be generated. The treatment of these aspects has been incorporated. The transformation of a real life example conceptual schema into F-logic has been worked out in detail. Moreover, the obtained F-logic specification has been partially translated into ODMG-93, thus illustrating the applicability of the transformation process in practice.

For further research the most fundamental topic is the full formalization of the first (design) step of our transformation mechanism according to the formalisms of PSM and F-logic. Beside the mapping of information structures, the transformation of populations in a formal framework is also essential, since the semantics of an information structure is defined in terms of its possible populations. This is important in order to prove the correctness of the transformation formally. The general treatment of conceptual constraints, that are parts of conceptual schemas, and their translation are to be addressed. Furthermore, issues concerning the second (implementation) step of the overall transformation have to be worked out in more detail. Our present paper has set up the framework and provides the basis for a general automated transformation mechanism that covers all the aspects just desrcibed.

References

[1] S. Abiteboul, P.C. Fischer, and H.J. Schek. *Nested Relations and Complex Objects in Databases.* Springer-Verlag, Berlin, Germany, 1987.

[2] S. Abiteboul and P.C. Kanellakis. Object Identity as a Query Language Primitive. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 159–173, 1989.

[3] M. Atkinson, D. DeWitt, D. Maier, F. Bancilhon, K. Dittrich, and S.B. Zdonik. The object-oriented database system manifesto. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD-89)*, pages 40–57, Kyoto, Japan, 1989. Elsevier Science Publishers.

[4] C. Beeri. A formal approach to object-oriented databases. *Data & Knowledge Engineering*, 5:353–382, 1990.

[5] A. Benczúr, Cs. Hajas, and Gy. Kovács. Datalog Extension for Nested Relations. *Computers Math. Applic.*, 30(12):51–79, 1995.

[6] J. Biskup, R. Menzel, and T. Polle. Transforming an Entity-Relationship Schema into Object-Oriented Database Schemas. In *Proceedings of the International Workshop on Advances in Databases and Information Systems*, pages 67–78, Moscow, June 1995.

[7] P. van Bommel, Gy. Kovács, and A. Micsik. Transformation of database populations and operations from the conceptual to the internal level. *Information Systems*, 19(2):175–191, 1994.

[8] P. van Bommel and Th.P. van der Weide. Reducing the search space for conceptual schema transformation. *Data & Knowledge Engineering*, 8:269–292, 1992.

[9] R.G.G. Catell. *The Object Database Standard: ODMG-93.* Morgan Kaufmann, San Francisco, California, 1994.

[10] P.P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[11] E.F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[12] H. Darwen and C.J. Date. The Third Manifesto. *SIGMOD Record*, 24(1), March 1995.

[13] C.J. Date and H. Darwen. *A Guide to the SQL Standard.* Addison-Wesley, Reading, Massachusetts, 1992.

[14] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9(4):157–204, 1992.

[15] J.J. van Griethuysen, editor. *Concepts and Terminology for the Conceptual Schema and the Information Base.* Publ. nr. ISO/TC97/SC5-N695, 1982.

[16] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains.* PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[17] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, October 1993.

[18] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.

[19] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, 1995.

[20] W. Kim. Object-Oriented Database Systems: Promises, Reality, and Future. In *Proceedings of the 19th VLDB Conference*, pages 676–687, Dublin, Ireland, 1993.

[21] Y. Kornatzky and P. Shoval. Conceptual design of object-oriented database schemas using the binary-relationship model. *Data & Knowledge Engineering*, 14:265–288, 1994.

[22] Gy. Kovács and P. van Bommel. Overview of F-logic from Database Transformation Perspective. Technical Note CSI-N9607, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, 1996.

[23] Gy. Kovács, Cs. Hajas, and I. Quilio. Representations and Query Languages of Nested Relations. In L. Varga, editor, *Proceedings of the 4th Symposium on Programming Languages and Software Tools*, pages 360–373, Visegrád, Hungary, June 1995.

[24] J. Melton. (ISO/ANSI Working Draft) Database Language SQL/Foundation (SQL3). ISO DBL MCI-004 and ANSI X3H2-96-059, March 1996.

[25] J. Nachouki, M.P. Chastang, and H. Briand. From Entity-Relationship Diagram to an Object-Oriented Database. In *Proceedings of the 11th International Conference on the Entity-Relationship Approach*, pages 459–481, 1992.

[26] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach.* Prentice-Hall, Sydney, Australia, 1989.

[27] P. Pistor and F. Andersen. Designing a Generalized NF^2 Data Model with an SQL-type Language Interface. In *Proceedings of the 12th VLDB Conference*, pages 278–185, Kyoto, Japan, August 1986.

[28] M.A. Roth, H.F. Korth, and D.S. Batory. SQL/NF: a query language for $\neg$1NF relational databases. *Information Systems*, 12(1):99–114, 1987.

[29] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[30] H.J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[31] K.-D. Schewe and B. Thalheim. Fundamental Concepts of Object Oriented Databases. *Acta Cybernetica*, 11(1-2):49–83, 1993.

[32] G.M. Shaw and S.B. Zdonik. A Query Algebra for Object-Oriented Databases. In *Proceedings of the 6th International Conference on Data Engineering*, pages 154–162, 1990.

[33] P. Shoval and S. Shiran. Entity-relationship and object-oriented data modeling - an experimental comparision of design quality. *Data & Knowledge Engineering*, 21(3):297–315, 1997.

[34] M. Stonebraker, L.A. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, P. Bernstein, and D. Beech. Third-Generation Database System Manifesto. *SIGMOD Record*, 19(3):31–44, September 1990.

[35] T.J. Teorey, D. Yang, and J.P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222, 1986.

[36] J.D. Ullman. *Principles of Database and Knowledge-base Systems*, volume I. Computer Science Press, Rockville, Maryland, 1989.

Received, June 1997

CD Grammar Systems and Trajectories[*]

Alexandru Mateescu[†]

Abstract

In this paper we consider constraints, as a new research area for cooperating distributed (CD) grammar systems. Constraints are based on the notion of a trajectory. The flexible approach provides a framework to study some interesting properties of a CD grammar system like fairness or parallelization of languages. The use of teams in the derivations of words is also considered.

1 Introduction

The cooperating distributed (CD) grammar systems were introduced in [2] with motivations from Artificial Intelligence (the so-called blackboard model in problem solving, [22]). For more details see the monograph [3].

Such a system consists of several ordinary grammars working by turns on the same sentential form; at each moment, one component is active, the others are waiting. Natural variants are systems in which more components (a team) are active at the same time. Teams can be with prescribed number of elements, non-deterministically chosen. The notion was introduced in [8].

In this paper we consider constraints that involve the general strategy to switch from one component (team) to another component (team).

Usually, the operation is modelled by the shuffle operation or restrictions of this operation, such as literal shuffle, insertion, etc.

Syntactic constraints, we consider here, are based on the notion of a *trajectory*, introduced in [16]. Roughly speaking, a trajectory is a segment of a line in the plane, starting in the origin of axes and continuing parallel with the axis Ox or Oy. The line can change its direction only in points of non-negative integer coordinates.

A trajectory defines how to skip from a component (team) to another component (team) during the derivation operation.

Languages consisting of trajectories are a special case of picture languages introduced in [20].

[*]The work reported here has been supported by the Academy of Finland, Project 137358. Presented at the workshop *Grammar Systems: Recent Results and Perspectives, July 26-27, 1996, Budapest.*

[†]Department of Mathematics, University of Bucharest, Romania and Department of Mathematics, University of Turku, Finland

2 Basic definitions

The reader is referred to [25] for basic elements of formal language theory and to [3] for details about grammar systems.

For an alphabet Σ, we denote by Σ^* the free monoid generated by Σ under the operation of concatenation; λ is the empty string and $|x|$ is the length of $x \in \Sigma^*$. If $a \in \Sigma$ and $w \in \Sigma^*$, then $|w|_a$ denotes the number of occurrences of the symbol a in w.

The *anti-catenation* operation, denoted by "$^\circ$", is defined as: $u^\circ v = vu$, for any $u, v \in \Sigma^*$.

A *generalized sequential machine (GSM)* is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, q_0, F)$, where Q is a finite set of *states*, Σ is the *input alphabet*, Δ is the *output alphabet*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states*, and δ is the *transition function*, i.e., a function from $Q \times \Sigma$ to finite subsets of $Q \times \Delta^*$. Let $u = u_1 u_2 \dots u_n$ be a word from Σ^*, where $u_i \in \Sigma$, $1 \leq i \leq n$. The set of all output words of u by M, denoted $M(u)$, is:

$$M(u) = \{w \mid w = w_1 w_2 \dots w_n, \text{ where } w_i \in \delta(q_{i-1}, u_i), 1 \leq i \leq n,$$

$$\text{and } \delta(q_{n-1}, u_n) \in F\}.$$

If $L \subseteq \Sigma^*$ is a language, then:

$$M(L) = \bigcup_{u \in L} M(u).$$

For more informations about GSM, the reader is referred to [24].

A CD grammar system (of degree $n, n \geq 1$) is a construct

$$\Gamma = (N, \Sigma, P_1, P_2, \dots, P_n, S),$$

where N is a (nonterminal) alphabet, Σ is a (terminal) alphabet disjoint from N, $S \in N$ and P_i are finite sets of context-free rules over $N \cup \Sigma, 1 \leq i \leq n$.

For a given P_i, the direct derivation $\Longrightarrow_{P_i}$ is defined in the usual way; we denote by $\Longrightarrow_{P_i}^{=k}, \Longrightarrow_{P_i}^{\leq k}, \Longrightarrow_{P_i}^{\geq k}, \Longrightarrow_{P_i}^{*}, \Longrightarrow_{P_i}^{t}$ a derivation in P_i consisting of exactly k steps, at most k, at least k steps, $k \geq 1$, of any number of steps and as long as possible, respectively ($x \Longrightarrow_{P_i}^{t} y$ means that $x \Longrightarrow_{P_i}^{*} y$ and there is no z such that $y \Longrightarrow_{P_i} z$).

For $f \in \{*, t\} \cup \{\leq k, = k, \geq k \mid k \geq 1\}$ we denote by $L_f(\Gamma)$ the language generated by Γ in the f mode, that is

$$L_f(\Gamma) = \{x \in \Sigma^* \mid S \Longrightarrow_{P_{i_1}}^{f} x_1 \Longrightarrow_{P_{i_2}}^{f} \dots \Longrightarrow_{P_{i_m}}^{f} = x$$
$$1 \leq i_j \leq n, 1 \leq j \leq m, m \geq 1\}$$

and by $CD(f)$ the family of such languages. (Note that we do not distinguish here between systems with λ-free and with arbitrary components.)

Given a grammar system $\Gamma = (N, \Sigma, P_1, \ldots, P_n, w)$ with components $N, \Sigma, P_1, \ldots, P_n$ as above but with a string axiom $w \in (N \cup \Sigma)^*$ instead of a symbol $S \in N$, and given a natural number $s \geq 1$, a subset $Q = \{P_{i_1}, \ldots, P_{i_s}\}$ of $\{P_1, \ldots, P_n\}$ is called an s-team. For such an s-team Q and for $x, y \in (N \cup \Sigma)^*$, we write

$$x \Longrightarrow_Q y \quad \text{iff} \quad x = x_1 A_1 x_2 \ldots A_s x_{s+1}, y = x_1 y_1 x_2 \ldots y_s x_{s+1}, \\ x_j \in (N \cup \Sigma)^*, 1 \leq j \leq s+1, A_j \to y_j \in P_{i_j}, 1 \leq j \leq s.$$

Then the relations $\Longrightarrow_Q^{=k}, \Longrightarrow_Q^{\leq k}, \Longrightarrow_Q^{\geq k}, \Longrightarrow_Q^{*}, \Longrightarrow_Q^{t}, k \geq 1$, can be defined as above, with the clarification that in the t case the derivation is correct when no more rules of any of the team components can be used.

In the sequel we recall some operations from formal languages that simulate the parallel composition of words.

The *shuffle* operation, denoted by $\sqcup\!\sqcup$, is defined recursively by:

$$au \sqcup\!\sqcup bv = a(u \sqcup\!\sqcup bv) \cup b(au \sqcup\!\sqcup v),$$

and

$$u \sqcup\!\sqcup \lambda = \lambda \sqcup\!\sqcup u = \{u\},$$

where $u, v \in \Sigma^*$ and $a, b \in \Sigma$.
The *shuffle* of two languages L_1 and L_2 is:

$$L_1 \sqcup\!\sqcup L_2 = \bigcup_{u \in L_1, v \in L_2} u \sqcup\!\sqcup v.$$

The *literal shuffle*, denoted by $\sqcup\!\sqcup_l$, is defined as follows:

$$a_1 a_2 \ldots a_n \sqcup\!\sqcup_l b_1 b_2 \ldots b_m = \begin{cases} a_1 b_1 a_2 b_2 \ldots a_n b_n b_{n+1} \ldots b_m, & \text{if } n \leq m, \\ a_1 b_1 a_2 b_2 \ldots a_m b_m a_{m+1} \ldots a_n, & \text{if } m < n, \end{cases}$$

where $a_i, b_j \in \Sigma$.

$$(u \sqcup\!\sqcup_l \lambda) = (\lambda \sqcup\!\sqcup_l u) = \{u\},$$

where $u \in \Sigma^*$.

The *balanced literal shuffle*, denoted by $\sqcup\!\sqcup_{bl}$, is defined in the next way:

$$a_1 a_2 \ldots a_n \sqcup\!\sqcup_{bl} b_1 b_2 \ldots b_m = \begin{cases} a_1 b_1 a_2 b_2 \ldots a_n b_n, & \text{if } n = m, \\ \emptyset, & \text{if } n \neq m, \end{cases}$$

where $a_i, b_j \in \Sigma$.

The *insertion* operation, see [7], denoted by $\longleftarrow$, is defined as:

$$u \longleftarrow v = \{u'vu'' \mid u'u'' = u, u', u'' \in \Sigma^*\}.$$

All the above operations are extended in the usual way to operations with languages.

3 Trajectories and constraints

In this section we introduce the notion of the trajectory and that of the shuffle on trajectories, and study their basic properties which are necessary in the sequel. The shuffle of two words has a natural geometrical interpretation related to latticial points in the plane (points with nonnegative integer coordinates) and with a certain "walk" in the plane defined by each trajectory.

Definition 3.1 *Consider the alphabet $V = \{r, u\}$. We say that r and u are versors in the plane: r stands for the right direction, whereas u stands for the up direction. A trajectory is an element t, $t \in V^*$.*

□

Definition 3.2 *Let Σ be an alphabet and let t be a trajectory, $t = t_1t_2 \dots t_n$, where $t_i \in V, 1 \leq i \leq n$. Let α, β be two words over Σ, $\alpha = a_1a_2 \dots a_p, \beta = b_1b_2 \dots b_q$, where $a_i, b_j \in \Sigma, 1 \leq i \leq p$ and $1 \leq j \leq q$.*

The shuffle of α with β on the trajectory t, denoted $\alpha \sqcup\!\sqcup_t \beta$, is defined as follows: if $|\alpha| \neq |t|_r$ or $|\beta| \neq |t|_u$, then $\alpha \sqcup\!\sqcup_t \beta = \emptyset$, else

$\alpha \sqcup\!\sqcup_t \beta = c_1c_2 \dots c_{p+q}$, where, if $|t_1t_2 \dots t_{i-1}|_r = k_1$ and $|t_1t_2 \dots t_{i-1}|_u = k_2$, then

$$c_i = \begin{cases} a_{k_1+1}, & \text{if } t_i = r, \\ b_{k_2+1}, & \text{if } t_i = u. \end{cases}$$

□

If T is a set of trajectories, the *shuffle of α with β on the set T of trajectories*, denoted $\alpha \sqcup\!\sqcup_T \beta$, is:

$$\alpha \sqcup\!\sqcup_T \beta = \bigcup_{t \in T} \alpha \sqcup\!\sqcup_t \beta.$$

The above operation is extended to languages over Σ, if $L_1, L_2 \subseteq \Sigma^*$, then we define

$$L_1 \sqcup\!\sqcup_T L_2 = \bigcup_{\alpha \in L_1, \beta \in L_2} \alpha \sqcup\!\sqcup_T \beta.$$

Example 3.1 *Let α and β be the words $\alpha = a_1a_2a_3a_4a_5a_6a_7a_8$, $\beta = b_1b_2b_3b_4b_5$ and assume that $t = r^3u^2r^3ururu$. The shuffle of α with β on the trajectory t is:*

$$\alpha \sqcup\!\sqcup_t \beta = \{a_1a_2a_3b_1b_2a_4a_5a_6b_3a_7b_4a_8b_5\}.$$

The result has the following geometrical interpretation (see Figure 1): the trajectory t defines a line starting in the origin and continuing one unit to the right or up, depending of the definition of t. In our case, first there are three units right, then two units up, then three units right, etc. Assign α on the Ox axis and β on the Oy axis of the plane. Observe that the trajectory ends in the point with coordinates $(8, 5)$ (denoted by E in Figure 1) that is exactly the upper right corner of the rectangle defined by α and β, i.e., the rectangle $OAEB$ in Figure 1. Hence, the result of the shuffle of α with β on the trajectory t is nonempty. The result can be read following the line defined by the trajectory t: that is, when being in a lattice point of the trajectory, with the trajectory going right, one should pick up the corresponding letter from α, otherwise, if the trajectory is going up, then one should add to the result the corresponding letter from β. Hence, the trajectory t defines a line in the rectangle $OAEB$, on which one has "to walk" starting from the corner O, the origin, and ending in the corner E, the exit point. In each lattice point one has to follow one of the versors r or u, according to the definition of t.

Assume now that t' is another trajectory, say:

$$t' = ur^5u^3rur^2.$$

In Figure 1, the trajectory t' is depicted by a much bolder line than the trajectory t. Observe that:

$$\alpha \sqcup\!\sqcup_{t'} \beta = \{b_1a_1a_2a_3a_4a_5b_2b_3b_4a_6b_5a_7a_8\}.$$

Consider the set of trajectories, $T = \{t, t'\}$. The shuffle of α with β on the set T of trajectories is:

$$\alpha \sqcup\!\sqcup_T \beta = \{a_1a_2a_3b_1b_2a_4a_5a_6b_3a_7b_4a_8b_5, b_1a_1a_2a_3a_4a_5b_2b_3b_4a_6b_5a_7a_8\}.$$

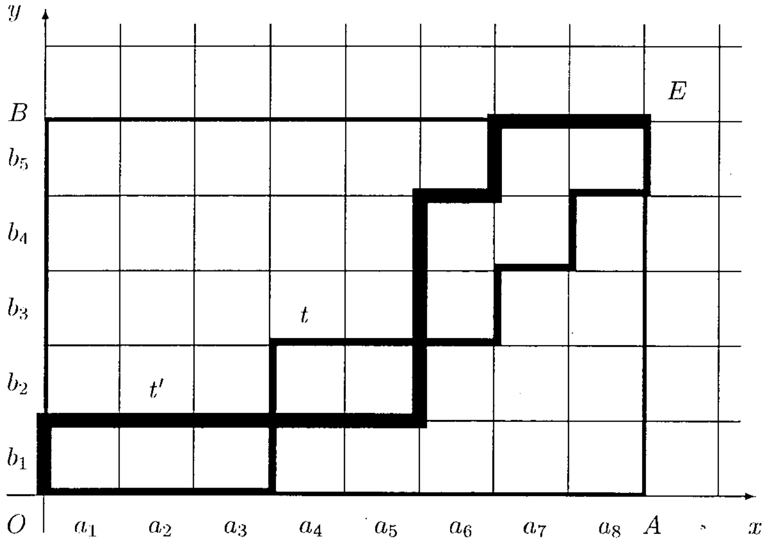


Figure 1

Remark 3.1 *One can easily observe that the following known operations for the parallel composition of words are particular cases of the operation of shuffle on trajectories.*

1. *Let T be the set $T = \{r, u\}^*$. Then for the shuffle operation $\sqcup\!\sqcup$, $\sqcup\!\sqcup_T = \sqcup\!\sqcup$.*

2. *Assume that $T = (ru)^*(r^* \cup u^*)$. Note that in this case $\sqcup\!\sqcup_T = \sqcup\!\sqcup_l$, the literal shuffle.*

3. *Consider $T = (ru)^*$. Then $\sqcup\!\sqcup_T = \sqcup\!\sqcup_{bl}$, where $\sqcup\!\sqcup_{bl}$ is the balanced literal shuffle.*

4. *Define $T = r^*u^*r^*$ and note that $\sqcup\!\sqcup_T = \longleftarrow$, where $\longleftarrow$ refers to the the insertion operation.*

5. *Assume that $T = r^*u^*$. It follows that $\sqcup\!\sqcup_T = \cdot$, where $\cdot$ is the catenation operation.*

6. *Consider $T = u^*r^*$ and observe that $\sqcup\!\sqcup_T = ^\circ$, where $^\circ$ denotes the anti-catenation operation.*

□

The following two theorems are representation results for the languages of the form $L_1 \sqcup\!\sqcup_T L_2$. We omit their rather straightforward proofs.

Theorem 3.1 *For all languages L_1 and L_2, $L_1, L_2 \subseteq \Sigma^*$, and for all sets T of trajectories, there exist a gsm M and two letter-to-letter morphisms g and h such that*

$$L_1 \sqcup\!\sqcup_T L_2 = M(h(L_1) \sqcup\!\sqcup g(L_2) \sqcup\!\sqcup T).$$

Our next theorem is a variant of Theorem 3.1.

Theorem 3.2 *For all languages L_1 and L_2, $L_1, L_2 \subseteq \Sigma^*$, and for all sets T of trajectories, there exist a morphism φ and two letter-to-letter morphisms g and h, $g : \Sigma \longrightarrow \Sigma_1^*$ and $h : \Sigma \longrightarrow \Sigma_2^*$ where Σ_1 and Σ_2 are two copies of Σ, and a regular language R, such that*

$$L_1 \sqcup\!\sqcup_T L_2 = \varphi((h(L_1) \sqcup\!\sqcup g(L_2) \sqcup\!\sqcup T) \cap R).$$

4 Constraints and CD grammar systems

Now we consider only CD grammar systems with two components. Moreover, we assume that the rules of each component have distinct labels. The case of CD grammar systems with more than two components can be easily obtained as a generalization.

Let $\Gamma = (N, \Sigma, S, P_1, P_2)$ be a CD grammar system with two components and let $T \subseteq \{r, u\}^*$ be a set of trajectories. *The constraint language generated by* Γ is the set of all words $w \in \Sigma^*$ such that w can be generated by Γ following a trajectory from T, i.e. the components P_1 and P_2 are used according to a trajectory $t \in T$. Whenever r does occur in t the component P_1 is used, otherwise, if u does occur in t, then the component P_2 is used.

Additionaly, one may consider constraint languages associated to each component. These languages are shuffled on the set T of trajectories.

Example 4.1 *Let* $\Gamma = (N, \Sigma, S, P_1, P_2)$ *be the following CD grammar system:* $N = \{S, X\}$, $\Sigma = \{a, b, c\}$,

$$P_1 = \{p_1 : S \longrightarrow aS,\ p_2 : X \longrightarrow cX,\ p_3 : X \longrightarrow \lambda\}$$

$$P_2 = \{q_1 : S \longrightarrow bS,\ q_2 : S \longrightarrow X\}.$$

The constraint language associated to the component P_1 *is* $L_1 = \{p_1^n p_2^n p_3 \mid n \geq 1\}$ *and the constraint language associated to the component* P_2 *is* $L_2 = \{q_1^n q_2 \mid n \geq 1\}$. *The set of trajectories is* $T = \{r^n u^{n+1} r^{n+1} \mid n \geq 1\}$. *The constraint language associated to the CD grammar system* Γ *is*

$$L_1 ⧢_T L_2 = \{p_1^n q_1^n q_2 p_2^n p_3 \mid n \geq 1\}.$$

One can easily verify that the language generated by the CD grammar system Γ *with the above constraints is:*

$$L(\Gamma) = \{a^n b^n c^n \mid n \geq 1\}.$$

□

Note that the language generated is non-context-free, but also the set T of trajectories is a non-context-free language. However we will see in the next section that this language can be generated also using only context-free constraints.

5 Regular and context-free trajectories

It is well known that the shuffle of two regular languages is a regular language. Moreover, given two finite automata A_1 and A_2 one can effectively find a finite automaton A such that $L(A) = L(A_1) ⧢ L(A_2)$.

The following theorem provides a characterization of those sets of trajectories T for which $L_1 ⧢_T L_2$ is a regular language, whenever L_1, L_2 are regular languages.

Theorem 5.1 *Let T be a set of trajectories, $T \subseteq \{r, u\}^*$. The following assertions are equivalent:*

(i) for all regular languages L_1, L_2, $L_1 ⊔⊔_T L_2$ is a regular language.

(ii) T is a regular language.

Proof. $(i) \Rightarrow (ii)$ Assume that $L_1 = r^*$ and $L_2 = u^*$ and note that $L_1 ⊔⊔_T L_2 = T$. It follows that T is a regular language.

$(ii) \Rightarrow (i)$ Assume that T is a regular language. Consider two regular languages L_1, L_2. Without loss of generality, we may assume that L_1 and L_2 are over the same alphabet Σ. Let $A_i = (Q_i, \Sigma, \delta_i, q_0^i, F_i)$ be a finite deterministic automaton such that $L(A_i) = L_i$, $i = 1, 2$. Also, let $A_T = (Q_T, \{r, u\}, \delta_T, q_0^T, F_T)$ be a finite deterministic automaton such that $L(A_T) = T$.

We define a finite nondeterministic automaton $A = (Q, \Sigma, \delta, Q_0, F)$ such that $L(A) = L_1 ⊔⊔_T L_2$. Informally, A, on an input $w \in \Sigma^*$, simulates nondeterministically A_1 or A_2 and from time to time changes the simulation from A_1 to A_2 or from A_2 to A_1. Each change determines a transition in A_T as follows: a change from A_1 to A_2 is interpreted as u and a change from A_2 to A_1 is interpreted as r. The input w is accepted by A iff A_1, A_2 and A_T accept.

Formally, $Q = Q_1 \times Q_T \times Q_2$, $Q_0 = \{(q_0^1, q_0^T, q_0^2)\}$, $F = F_1 \times F_T \times F_2$. The definition of δ is:

$$\delta((q_1, d, q_2), a) = \{(\delta_1(q_1, a), \delta_T(d, r), q_2), (q_1, \delta_T(d, u), \delta_2(q_2, a))\},$$

where, $q_1 \in Q_1, d \in Q_T, q_2 \in Q_2, a \in \Sigma$.

One can easily verify that $L(A) = L_1 ⊔⊔_T L_2$ and hence $L_1 ⊔⊔_T L_2$ is a regular language.

□

Next theorem gives a similar result as Theorem 5.1, but for context-free sets of trajectories.

Theorem 5.2 *Let T be a set of trajectories, $T \subseteq \{r, u\}^*$. The following assertions are equivalent:*

(i) for all regular languages L_1, L_2, $L_1 ⊔⊔_T L_2$ is a context-free language.

(ii) T is a context-free language.

Proof. $(i) \Rightarrow (ii)$ Assume that $L_1 = r^*$ and $L_2 = u^*$ and note that $L_1 ⊔⊔_T L_2 = T$. Therefore T is a context-free language.

$(ii) \Rightarrow (i)$ Assume that T is a context-free language. Consider two regular languages L_1, L_2. Without loss of generality, we may assume that L_1 and L_2 are over the same alphabet Σ. Let $A_i = (Q_i, \Sigma, \delta_i, q_0^i, F_i)$ be a finite deterministic automaton such that $L(A_i) = L_i$, $i = 1, 2$. Also, let $P_T = (Q_T, \Gamma_T, \{r, u\}, \delta_T, q_0^T, Z_T, F_T)$ be a pushdown automaton such that $L(P_T) = T$.

We define a pushdown automaton $P = (Q, \Gamma, \Sigma, \delta, Q_0, Z, F)$ such that $L(P) = L_1 \sqcup\!\sqcup_T L_2$. Informally, P, behaves as the automaton A from the proof of Theorem 5.1, except that on the second component of the states, P simulates the pushdown automaton P_T. That is, on an input $w \in \Sigma^*$, P simulates nondeterministically A_1 or A_2 and from time to time changes the simulation from A_1 to A_2 or from A_2 to A_1. Each change determines a transition in P_T as follows: a change from A_1 to A_2 is interpreted as u and a change from A_2 to A_1 is interpreted as r. The input w is accepted by P iff A_1, A_2 and P_T accept.

Formally, $Q = Q_1 \times Q_T \times Q_2$, $Q_0 = \{(q_0^1, q_0^T, q_0^2)\}$, $F = F_1 \times F_T \times F_2$, $\Gamma = \Gamma_T$, $Z = Z_T$. The definition of δ is:

$$\delta((q_1, d, q_2), a, X) = \cup_{(s,\alpha)\in\delta_T(d,r,X)}((\delta_1(q_1, a), s, q_2), \alpha)\cup$$

$$\cup_{(s',\alpha')\in\delta_T(d,u,X)}((q_1, s', \delta_2(q_2, a), \alpha')\}$$

where, $q_1 \in Q_1$, $d \in Q_T$, $q_2 \in Q_2$, $a \in \Sigma$, $X \in \Gamma$, $\alpha \in \Gamma^*$.

Additionally,

$$\delta((q_1, d, q_2), \lambda, X) = \cup_{(s,\alpha)\in\delta_T(d,\lambda,X)}((q_1, s, q_2), \alpha)$$

where, $q_1 \in Q_1$, $d \in Q_T$, $q_2 \in Q_2$, $X \in \Gamma$, $\alpha \in \Gamma^*$.

One can verify that $L(P) = L_1 \sqcup\!\sqcup_T L_2$ and hence $L_1 \sqcup\!\sqcup_T L_2$ is a context-free language. $\square$

Theorem 5.3 *Let T be a set of trajectories, $T \subseteq \{r, u\}^*$ such that T is a regular language.*

(i) If L_1 is a context-free language and if L_2 is a regular language, then $L_1 \sqcup\!\sqcup_T L_2$ is a context-free language.

(ii) If L_1 is a regular language and if L_2 is a context-free language, then $L_1 \sqcup\!\sqcup_T L_2$ is a context-free language.

Proof. The proof is similar with the proof of Theorem 5.2. For the case (i) the pushdown automaton is simulated on the first component of the states, whereas for the case (ii) the pushdown automaton is simulated on the third component of the states. $\square$

Alternative proofs for Theorems 5.1 - 5.3 can be obtained using Theorem 3.1 or Theorem 3.2.

From Theorems 5.1 - 5.3 we obtain the following corollary:

Corollary 5.1 *Let L_1, L_2 and T, $T \subseteq \{r, u\}^*$ be three languages.*

(i) if all three languages are regular languages, then $L_1 \sqcup\!\sqcup_T L_2$ is a regular language.

(ii) if two languages are regular languages and the third one is a context-free language, then $L_1 \sqcup\!\sqcup_T L_2$ is a context-free language.

6 Fairness

Fairness is a property of the parallel composition of processes that, roughly speaking, says that each action of a process is performed with not too much delay with respect to performing actions from another process. That is, the parallel composition is "fair" with both processes that are performed.

Definition 6.1 *Let $T \subseteq \{r, u\}^*$ be a set of trajectories and let n be an integer, $n \geq 1$. T has the n-fairness property iff for all $t \in T$ and for all t' such that $t = t't''$ for some $t'' \in \{r, u\}^*$, it follows that:*

$$\mid |t'|_r - |t'|_u \mid \leq n.$$

□

This means that all trajectories from T are contained in the region of the plane bounded by the line $y = x - n$ and the line $y = x + n$, see Figure 2, for $n = 4$.

Example 6.1 *The balanced literal shuffle ($\sqcup\!\sqcup_b$) has the n-fairness property for all n, $n \geq 1$.*

The following operations: shuffle ($\sqcup\!\sqcup$), catenation ($\cdot$), insertion ($\longleftarrow$) do not have the n-fairness property for any n, $n \geq 1$.

For instance, note that the catenation means shuffle on the set T of trajectories, where (see also Remark 3.1, 5.):

$$T = r^*u^* = \{r^iu^j \mid i, j \geq 0\}.$$

Therefore,

$$\{\mid |t'|_r - |t'|_u\mid \,\mid t't'' \in T \text{ for some } t''\} = \{\mid i - j \mid \,\mid i, j \geq 0\}.$$

Because the values $\mid i - j \mid$, where $i, j \geq 0$, cannot be bounded by any fixed constant n, $n \geq 1$, it follows that the catenation is not n-fair for any $n \geq 1$.

A similar argument is valid to prove that shuffle and insertion operations do not have the n-fairness property for any n, $n \geq 1$.

Definition 6.2 *Let n be a fixed number, $n \geq 1$. Define the language F_n by:*

$$F_n = \{t \in V^* \mid \mid |t'|_r - |t'|_u \mid \leq n, \text{ for all } t' \text{ such that } t = t't'', t'' \in V^*\}.$$

□

Remark 6.1 *Note that a set T of trajectories has the n-fairness property if and only if $T \subseteq F_n$.*

□

We omit the straightforward proof of the following statement.

Proposition 6.1 *For every n, $n \geq 1$, the language F_n is a regular language.*

Corollary 6.1 *Let T be a set of trajectories. If T is a context-free or a simple matrix language and n is a fixed number, $n \geq 1$, then it is decidable whether or not T has the n-fairness property.*

Proof. It is easy to observe that for the above families of languages the problem if a language from a family is contained in a regular language is a decidable problem. Hence, from Proposition 6.1, this corollary follows. □

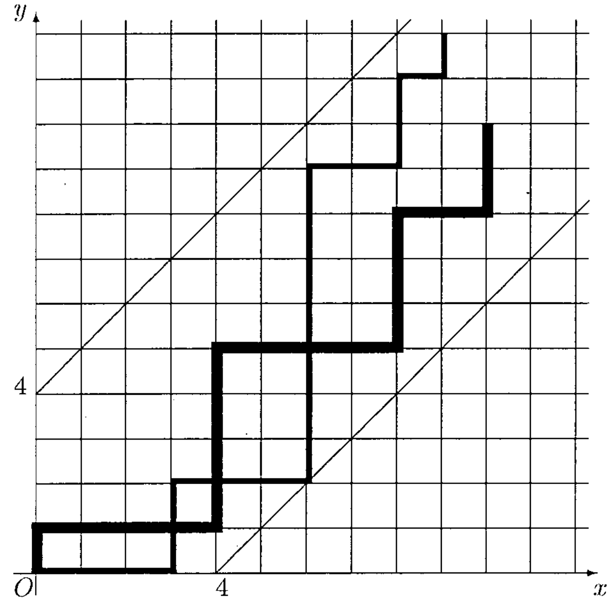


Figure 2

Comment. For a context-free language $T \subseteq V^*$ it is decidable whether or not there exists a non-negative integer n, such that T has the n-fairness property, see [19]. However, in general it is an open problem for what families $\mathcal{L}$ of languages it is decidable this problem.

Now we use the fairness concept in connection with CD grammar systems. Let Γ be a CD grammar system,

$$\Gamma = (N, \Sigma, P_1, P_2, \ldots, P_m, S).$$

Assume that the components of Γ are labelled, such that P_i has the label e_i, $1 \leq i \leq m$. Let E be the set of labels, $E = \{e_1, e_2, \ldots, e_m\}$.

In order to extend the notion of fairness for the general case of CD grammar systems with m components, $m \geq 2$, firstly we define the notion of a m-trajectory. A m-trajectory is an element $t \in E^*$, i.e., a word over the m letters alphabet $\{e_1, e_2, \ldots, e_m\}$.

Definition 6.3 *Let $T \subseteq E^*$ be a set of m-trajectories and let n be an integer, $n \geq 1$. T has the n-fairness property iff for all $t \in T$ and for all t' such that $t = t't''$ for some $t'' \in E^*$, it follows that:*

$$| \, |t'|_{e_i} - |t'|_{e_j} \, | \leq n,$$

for all $1 \leq i, j \leq m$.

A CD grammar Γ has the n-fairness property iff for all terminal derivations

$$S \Longrightarrow_{e_{i_1}} w_1 \Longrightarrow_{e_{i_2}} w_2 \Longrightarrow_{e_{i_3}} \cdots \Longrightarrow_{e_{i_k}} w_k$$

the corresponding trajectory $e_{i_1} e_{i_2} \ldots e_{i_k}$ has the n-fairness property.

A language L is n-fair, $n \geq 1$, iff there exists a CD grammar system Γ with the n-fairness property such that $L(\Gamma) = L$.

□

Theorem 6.1 *If a language L can be generated by a CD grammar system Γ such that Γ has the n-fairness property for some $n \geq 1$, then L can be generated by a CD grammar system Γ' in the $\leq k$ mode of derivation.*

The converse is not true.

Proof. Observe that for $k = n$ the CD grammar system Γ has the property that $L_{\leq k}(\Gamma) = L$. Hence, one can simply define the CD grammar system Γ' as being Γ.

The converse is not true since a CD grammar system Γ can generate terminal strings in the $\leq k$ mode, just by alternating two of its components and without using the other components. Thus, such a derivation is not n-fair for any n. □

Theorem 6.2 *There exists a non-context-free and semilinear language L such that:*

(i) L can be generated by a CD grammar system in the t mode of derivation.

(ii) L cannot be generated by any n-fair CD grammar system for any $n \geq 1$.

Proof. (*i*) Let L be the following non-context-free and semilinear language:

$$L = \{a^i b^i c^j \mid 1 \leq i \leq j\}.$$

Let Γ be the following CD grammar system:

$$\Gamma = (N, \Sigma, P_1, P_2, P_3, P_4, S),$$

where: $N = \{S, X, X', Y, Y', Y'', Z)$, $\Sigma = \{a, b, c\}$, and the components:

$$P_1 = \{S \longrightarrow XY, S \longrightarrow X'Y', S \longrightarrow X'Y''\},$$

$$P_2 = \{X \longrightarrow aX'b, Y \longrightarrow cY', Y \longrightarrow cY''\},$$

$$P_3 = \{X' \longrightarrow aXb, Y' \longrightarrow cY, Y'' \longrightarrow cZ\},$$

$$P_4 = \{X' \longrightarrow ab, Y'' \longrightarrow c, Y'' \longrightarrow cY''\}.$$

One can easily verify that $L_t(\Gamma) = L$.

(*ii*) In order to prove this statement, assume by contrary that L can be generated by a CD grammar system Γ that has the n-fairness property for some $n \geq 1$.

Clearly, Γ must have a component, say α, that increases the number of occurrences of the symbol a at least by one. Similarly, Γ should have a component, say γ, that increase the number of occurrences of the symbol c by s. Assume that s is the maximum of the number of c symbols that can be produced by a component when it is applied only once.

Since the CD grammar system Γ is n-fair for some fixed $n \geq 1$, it follows that after each n consecutive steps in a derivation the number of occurrences of the symbol a is increased with at least 1 and the number of occurrences of the symbol c with at most $(n-1)s$.

Therefore, if a terminal derivation has length p, where $p = nq + r$, such that $0 \leq r < n$, then the derived word has at least q occurrences of the symbol a and at most $q(n-1)s + rs$ occurrences of the symbol c.

Assume that this derivation produces the terminal word $w = a^i b^i c^j$. Note that $q < i$ and that $j < q(n-1)s + rs < q(n-1)s + ns$. Therefore $j < i(n-1)s + ns$. Note that n and s are fixed constants.

It follows that the CD grammar system Γ cannot generate words $a^i b^i c^j$ with $j \geq i(n-1)s + ns$. This contradicts our assumption that $L(\Gamma) = L$. $\square$

Comment. The above theorem is similar with another, well-known result from the theory of CD grammar-systems, see [3]. The derivation mode $= k$ gives also some idea of fairness. However, it is known that the language

$$L = \{a^{2^n} \mid n \geq 1\}$$

can be generated by a CD grammar system in the t mode, but L cannot be generated by any CD grammar system in the mode $= k$.

Theorem 6.2 provides an example of a language that can be generated by a CD grammar system in mode t, but it cannot be generated by any n-fair CD grammar system for any $n \geq 1$.

7 Parallelization of CD grammar systems

In the following we shall deal with parallelization of languages using shuffle on trajectories.

The *parallelization of a problem* consists in decomposing the problem in subproblems, such that each subproblem can be solved by a processor, i.e., the subproblems are solved in parallel and, finally, the partial results are collected and assembled in the answer of the initial problem by a processor. Solving problems in this way increases the time efficiency. It is known that not every problem can be parallelized. Also, no general methods are known for the parallelization of problems.

Here we formulate the problem in terms of languages and shuffle on trajectories, and present some examples.

The *parallelization of* a language L consists in finding languages L_1, L_2 and T, $T \subseteq V^*$, such that $L = L_1 ш_T L_2$ and moreover, the complexity of L_1, L_2 and T is in some sense smaller than the complexity of L. In the sequel the complexity of a language L refers to the Chomsky class of L, i.e., regular languages are less complex than context-free languages that are less complex than context-sensitive languages.

One can easily see that every language L, $L \subseteq \{a,b\}^*$ can be written as $L = a^* ш_T b^*$ for some set T of trajectories. However, this is not a parallelization of L since the complexity of T is the same with the complexity of L.

In view of Corollary 5.1 there are non-context-free languages L such that $L = L_1 ш_T L_2$ for some context-free languages L_1, L_2 and T. Moreover, one of those three languages can be even a regular language. Note that this is a parallelization of L.

As a first example we consider the non-context-free language $L \subseteq \{a,b,c\}^*$, $L = \{w \mid\mid w\mid_a = \mid w\mid_b = \mid w\mid_c\}$.

Consider the languages: $L_1 \subseteq \{a,b\}^*$, $L_1 = \{u \mid\mid u\mid_a = \mid u\mid_b\}$, $L_2 = c^*$ and $T = \{t \mid\mid t\mid_r = 2 \mid t\mid_u\}$.

One can easily verify that $L = L_1 ш_T L_2$. Moreover, note that L_1 and T are context-free languages, whereas L_2 is a regular language. Hence this is a parallelization of L. As a consequence of Corollary 5.1 one cannot expect a significant improvement of this result, for instance to have only one context-free language and two regular languages in the decomposition of L.

Now we consider the case of CD grammar systems. Next example shows how one can define context-free constraints to generate a non-context-free language.

Example 7.1 *Let* $\Gamma = (N, \Sigma, S, P_1, P_2)$ *be the following CD grammar system:* $N = \{S\}$, $\Sigma = \{a,b,c\}$,

$$P_1 = \{p_1 : S \longrightarrow aS,\ p_2 : S \longrightarrow bS\}$$

$$P_2 = \{q_1 : S \longrightarrow cS,\ q_2 : S \longrightarrow \lambda\}.$$

The constraint language associated to the component P_1 *is* $L_1 = \{p_1^n p_2^n \mid n \geq 1\}$ *and the constraint language associated to the component* P_2 *is* $L_2 = \{q_1^n q_2 \mid n \geq 1\}$. *The set of trajectories is* $T = \{r^{2n} u^{n+1} \mid n \geq 1\}$. *The constraint language associated to the CD grammar system* Γ *is*

$$L_1 ш_T L_2 = \{p_1^n p_2^n q_1^n q_2 \mid n \geq 1\}.$$

One can easily verify that the language generated by the CD grammar system Γ *with the above constraints is:*

$$L(\Gamma) = \{a^n b^n c^n \mid n \geq 1\}.$$

□

Each set T of trajectories from the above examples concerning CD grammar systems does not have the fairness property. However it is not known if the language $L = \{a^n b^n c^n \mid n \geq 1\}$ can be generated by a CD grammar system using constraints with the fairness property.

8 Conclusions

We considered the notion of trajectory in connection with CD grammar systems. The use of trajectories in the theory of CD grammar systems offers some new possibilities to investigate this area. The concept of fairness can be introduced at the level of the components of a CD grammar system, at the level of the productions of a CD grammar system or at the level of the teams used in the derivations of a CD grammar system. For the case of teams, one should put labels to all possible teams and consider as valid only those derivations that follow trajectories from a certain, fixed set of trajectories. Mixed fairness constraints are also possible. For a given CD grammar system, the valid derivations can be defined as being those derivations that satisfy a certain fairness constraint at the level of components and another fairness constraint at the level of productions, etc. Therefore, this framework offers a great flexibility in modelling the fairness phenomenon with respect to CD grammar systems.

Fairness is a natural property of a CD grammar system and it leads to new interesting properties. For instance, it is not known the generative power of the CD grammar systems that use constraints with the fairness property.

There are different natural variants of the fairness property. The fairness property can be considered also with respect to only a part (a fixed subset) of the components (or of productions or teams) of a CD grammar system.

The fairness property can be relaxed or modified in other, different ways. For instance one can consider the restriction that there exists a fixed $n \geq 1$ such that in any terminal derivation, in any n consecutive steps of it, each component does occur at least once, but it does not occur more than k times, where k is a fixed number.

The interrelations between the fairness property and the generative modes t, $= k$, $\leq k$ and $*$ are subjected for further research.

A more general approach, based on geometric considerations, can be considered. Assume that we fix two regions A and B in a many dimensional space (the number of dimensions is equal with the number of versors that encode the trajectories). The regions A and B are not necessarily disjoint. A derivation is considered valid iff the associated trajectory is contained in the region A but it avoids the region B. Note that this approach is an extensions of the notion of fairness depicted in Figure 2. There the region A is the band of the plane bounded by the lines $y = x + 4$ and $y = x - 4$ whereas the region B is empty or any region outside of A.

The idea behind this considerations is also the existence of non-critical sections (devices) described by the region A and of critical sections (devices) described by the region B.

Another important problem is the problem of parallelization of languages, i.e., to express a language as the shuffle of two (or more) other languages over a certain set (or sets) of trajectories. The possibility of decomposing a language as the parallel composition of other, less complex, languages is of theoretical but especially of practical interest. This problem leads to the possibility to perform the parsing operation or other operations, by a parallel machine.

It is an open problem to decide for a given language L (L defined using a CD grammar system) whether or not there exist two languages L_1 and L_2 and a set T of trajectories, such that $L = L_1 \sqcup\!\sqcup_T L_2$.

The problem of parallelization of languages opens new connections between CD grammar systems and the theory of parallel computation.

References

[1] A. Atanasiu and V. Mitrana, "The modular grammars", *Intern. J. Computer Math.*, 30 (1989), 101-122.

[2] E. Csuhaj-Varju and J. Dassow, "On cooperating distributed grammar systems", *J. Inform. Proc. Cybern. EIK*, 26 (1990), 49-63.

[3] E. Csuhaj-Varju, J. Dassow, J. Kelemen and Gh. Păun, *Grammar Systems*, Gordon and Breach, 1993.

[4] J. S. Golan, A. Mateescu and D. Vaida, "Semirings and Parallel Composition of Processes ", Journal of Automata, Languages and Combinatorics, 1 (1996) 3, 199 - 217.

[5] L. Guo, K. Salomaa and S. Yu, *Synchronization Expressions and Languages*, The University of Western Ontario London, Dept. of Comp. Sci., Technical Report 368, 1993.

[6] T. Harju, M. Lipponen and A. Mateescu, "Flatwords and Post Correspondence Problem", *Theoretical Computer Science*, TCS, 161 (1996) 93 - 108.

[7] L. Kari, *On insertion and deletion in formal languages*, PhD Thesis, University of Turku, Turku, Finland, 1991.

[8] L. Kari, A. Mateescu, G. Păun and A. Salomaa, "Teams in cooperating grammar systems", *J. Expt. Theor. Artif. Intell.*, 7(1995) 347-359.

[9] J. Kelemen and A. Kelemenova, "A subsumption arhitecture for generative symbol systems", *Cybernetics and Systems Research '92*, Proc. 11th European Meeting Cybern. Syst. Res. (R. Trappl, ed.), World Scientific, 1992, 1529-1536.

[10] A. Kelemenova and E. Csuhaj-Varju, "Languages of colonies", *2nd Intern. Coll. Words, Languages, Combinatorics*, Kyoto, 1992.

[11] H. C. M. Kleijn and G. Rozenberg, "A study in parallel rewriting systems", *Inform. Control*, 44 (1980), 134-163.

[12] M. Kudlek and A. Mateescu, "Distributed Catenation and Chomsky Hierarchy", FCT'95, Dresden, 1995, Lecture Notes in Computer Science, LNCS 965, Springer-Verlag, 1995, 313-322.

[13] M. Kudlek and A. Mateescu, "Rational and Algebraic Languages with Distributed Catenation", DLT'95, Magdeburg, 1995, in *Developments in Language Theory II*, eds. J. Dassow, G. Rozenberg and A. Salomaa, World Scientific, Singapore, 1996, 129-138.

[14] W. Kuich and A. Salomaa, *Semirings, Automata, Languages*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1986.

[15] A. Mateescu, "On (Left) Partial Shuffle", *Results and Trends in Theoretical Computer Science*, LNCS 812, Springer-Verlag, (1994) 264-278.

[16] A. Mateescu, G. Rozenberg and A. Salomaa, "Shuffle on Trajectories: Syntactic Constraints", TUCS Technical Report, 41, 1996.

[17] A. Mateescu and A. Salomaa, "Formal Languages: an Introduction and a Synopsis", Chapter 1, in *Handbook of Formal Languages*, eds. G. Rozenberg and A. Salomaa, Springer-Verlag, 1997, 1-40.

[18] A. Mateescu and A. Salomaa, "Parallel Composition of Words with Reentrant Symbols", TUCS Technical Report, 15, 1996.

[19] A. Mateescu, K. Salomaa and S. Yu, "Decidability of Fairness for Context-Free Languages", to appear in Proc. of DLT'97, Thessaloniki, July, 1997.

[20] H. A. Maurer, G. Rozenberg and E. Welzl, "Using String Languages to Describe Picture Languages", *Information and Control*, 3, 54, (1982) 155-185.

[21] R. Meersman and G. Rozenberg, "Cooperating grammar systems", *Proc. MFCS '78 Symp., LNCS 64*, Springer-Verlag, 1978, 364-376.

[22] P. H. Nii, "Blackboard systems", in *The Handbook of AI*, vol. 4 (A. Barr, P. R. Cohen, E. A. Feigenbaum, eds.), Addison-Wesley, 1989.

[23] Gh. Păun and L. Sântean, "Parallel communicating grammar systems: the regular case", *Ann. Univ. Buc., Series Matem.-Inform.* 38 (1989), 55-63.

[24] G. Rozenberg and A. Salomaa,(Eds.), *Handbook of Formal Languages*, Springer, 1997.

[25] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

On Hybrid Connectionist-Symbolic Models [*]

Petr Sosík [†‡]

Abstract

There are many similarities between grammar systems and artificial neural networks: parallelism, independently working elements (grammars/neurons), communication of the elements, absence of centralized control. On the other hand, there are crucial differences between symbolic and quantitative data processing.

We will try to give a brief overview of the methods of "building bridges" between symbolic and connectionist paradigm and to propose some recent results. After that, we will touch some essential problems, concerning incorporation of accepting/generating grammar systems and neural network models.

Keywords: *grammar system, artificial neural network, finite automaton, grammatical inference.*

1 Introduction

First let's recall some characteristics of the constructions we intend to deal with, emphasizing their aspects important for the following discussion. This is not to replace broad and exact descriptions in the basic literature referred to.

Moreover, we restrict our attention to rather theoretical aspects of interactions of neural nets and grammars/grammar systems in hybrid models. For a broader discussion of linking symbolic and subsymbolic systems we refer e.g. to [38].

Speaking about some types of distributed systems, we will use the term "agent" for some simple, but subsystem, interacting with the environment (including other agents) [14]. In terms of grammar systems the agent represents usually a grammar (Chomsky, Lindenmayer or other type [27]), in terms of neural network it represents one neuron or a small group of them.

Let's keep in mind that the artificial neural networks are (not only) biologically motivated, so that each model is looked at as a mathematical machine, possibly physically implementable. That's why this "implementing" point of view will be emphasized in the following paragraphs.

[*]Presented at workshop *Grammar Systems: Recent Results and Perspectives, July 26-27, 1996, Budapest.*

[†]Institute of Mathematics and Computer Science, Silesian University at Opava, Bezruv covo nám. 13, 74601 Opava, Czech Republic, email: Petr.Sosik@fpf.slu.cz

[‡]Research supported by the Grant Agency of Czech Republic, grant No. 201/95/0134.

1.1 Automata

A formal automaton is an accepting device, which is primarily intended to classify strings as *accepted* or *rejected.* Its input string (over a finite alphabet) is accessed *sequentially.* At each step the automaton reads one symbol of the string (and eventually performs some other actions), at the next step it can read the neighboring symbol only. Generally it needs a read/write memory of an unlimited size, but in the most cases of interest in this paper (languages within the context-sensitive class), the amount of memory actually needed can be limited to the length of the processed string. For basic description of formal automata and grammars we refer to [19], [39].

1.2 Grammars

A formal grammar is a form of syntactical description of some formal language. The most important component is a set of rewriting rules (over some finite alphabet). We can interpret it as a generating device, at each step applying some of its rules to a processed sentential form. Due to the fact, that such a device must be eventually able to produce any string of its (often infinite) language, it is mostly nondeterministic (which concern the selection of a rule and a position within the processed string where it is applied). An implementation of a grammar would require an unlimited size memory for reading/writing the string just generated. The memory must allow in principle random access, i.e. at each step any symbol(s) of the string generated can be processed. Some types of grammars can be transformed to a normal form suffering with sequential access. Moreover, the memory must be able to insert new symbols into a processed string.

1.3 Neural networks

An *artificial neural network* (ANN) is a finite set of interconnected autonomous agents – *neurons.* There exist also models of infinite size, but they are rather special [37]. Typically all neurons in the network compute the same function (1), where x_j are the inputs, the y_i is the output of i-th neuron in the network, θ_i is the *threshold function,* see Fig.1. Constants w_{ij} are called *weights* of the inputs. For a more detailed description we refer to [18], [34].

$$y_i = \theta_i(\sum_{j=1}^{N} w_{ij}x_j) \tag{1}$$

The input of ANN is some n-tuple and the output some m-tuple of real-valued signals. There is no relation between n and m. The neurons are often grouped to *layers,* each layer being connected by its inputs to its lower and by its outputs to its upper neighbor. Then the topmost layer is called the *output layer,* connecting its outputs to the outer environment, the most bottom one with its inputs incoming from the environmen is the *input layer* and the others are the *hidden* ones. Within

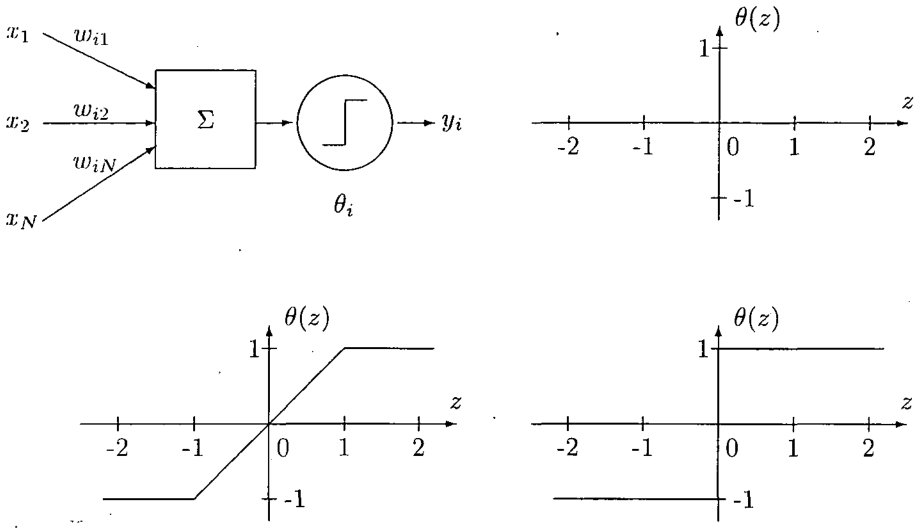


Figure 1: The basic model of neuron and the common types of threshold function.

one-layer networks, the same categorization can be applied to single neurons. There are also intra-layer or bidirectional connections in some models.

The topology of the network (with respect to the paths of passing signals) can be *feedforward* or *feedback*. In the later case, the existence of the feedback loops can lead to the complex dynamical behavior of the network.

There is a *pre-defined communication graph* of the neurons. It can evolve during a training process (some new connections can appear, some old ones can vanish, existing ones can change their strength), but generally these changes are much slower than the normal flow of information (signal levels) through the network.

Thus we distinguish between the *learning mode* of the network when the weights (and sometimes other parameters or even topology) change, and the *recall mode,* when only signal level changes. There is a lot of learning (training) algorithms, various heuristics, which we will not discuss here in general, although it is just the training algorithm, what mostly influences the network behavior.

From another point of view, it is possible to use ANNs with a certain set of threshold functions for approximating any real function with arbitrary small error [23], [30], but this approach is far from symbolic information processing we intend to deal with.

In the most of the known models all the neurons operate *in parallel* (except, for instance, Hopfield network with asynchronous mode) and there is no centralized control of the network activity; the neuron's behavior is influenced by its neighbours only. Neurons can operate in both continuous and discrete time, in the latter case often with a common clock.

1.4 Grammar systems

A grammar system usually consists of some set of (relatively autonomous) components (grammars, agents), performing some operations with a processed string. The notion, originally, was introduced for modelling syntactic properties of multi-agent systems (for details see [6] and [9]), but many variants of grammars (rewriting systems) with regulation and/or modularization can be considered as grammar systems, too (confer to [31], [8]). Here we only emphasize some special capabilities of grammar systems, different from those of single grammars.

- *Parallelism*: while grammars with parallel derivation are rather special, some of basic grammar system models are inherently parallel, e.g. Lindenmayer systems (although they are rather a kind of parallel grammar than a grammar system [27]), parallel communicating grammar systems (PCGS), ...
- In the case of these parallel grammar systems, a *memory* for storing the processed string must allow multiple agents to access different parts of the string at the same time.
- In some models (Lindenmayer systems [31], colonies with terminal mode of rewriting [21], ...) there is a virtually *unlimited potential* of rewriting rules (each rule must be simultaneously at arbitrary number of symbols in generated string).
- There are various forms of *communication* between agents; the simplest form is probably synchronization. A communication graph of agents can be predefined (e.g. colonies, eco-grammar systems [5]) or dynamically defined during the work of the system (e.g. PCGS).

Especially the requirements for the dynamic communication graph, the unlimited potential of rules, but also for the parallel memory capable of inserting symbols can be a source of implementational difficulties [24]. Among physically existing systems, most close to these requirements seem to be those with a great number of elements moving freely (immune system [29], splicing systems [12], ...).

Accepting grammar systems have also been defined [10], [11], relaxing some of these requirements (from the implementation point of view) of the generating systems: the degree of nondeterminism (however its definition could be problematic [10]) is often lower, there is no necessity of inserting new symbols into the processed string if we do not admit λ-rules.

2 A motivation for hybrid models

2.1 Goals and expectations

Both grammar and neural systems, representing symbolic and quantitative data manipulation, are inspired (partially) from the areas of AI and AL. Indeed in biological systems we can find connection of both approaches. For example: specific immunity could well be viewed as a competition between grammars, generating polysaccharide strings covering the antigen; and lymphocyte automata attempting to recognize them. And this competition causes adaptation of both of these systems [29].

Among the advantages of ANNs the most important are adaptability, generalizability, massive parallelism, noise tolerance and graceful degradation (robustness). Main disadvantages are difficult understanding and explanation of the results and time-consuming learning. The symbol processing systems have their strength in easy manipulation with symbolic and rule knowledge and wide base of theoretical results, their weaknesses dwell in sensitivity to noise and difficult, often almost impossible adaptation to a novel data. As the disadvantages of one type systems are balanced by the advantages of second type ones, the hybrid architectures seem to be a very promising way [4]. Another advantage of hybrid architectures can be their scalability. Finally, some authors find them crucial to understanding and constructing cognitive models [17].

2.2 Some basic ideas

The first step of a neural symbolic processing is to find a proper coding of symbols. In animal nerves all signals are of the same amplitude and stimuli intensity is expressed by their frequency [25]. Moreover, coding via the signal intensity leads to loss of robustness due to the necessity of distinguishing between close signal levels. Symbols are coded most often by a group of parallel stimuli, each with clearly distinguishable intensity levels (typically binary). Very often a "one-hot" coding is used (each symbol is assigned a separate input of the ANN).

With a piece of abstraction we can think about symbol-processing structures within the brain, with their abilities having been obtained by training. Training is mostly much more effective with a teacher: a neural network obtains some stimuli and after processing them gives output. Then the teacher produces a training signal, expressing the difference between desired output and output given by net. This signal is utilized by the to change its internal structure slightly, so that, after many such cycles, resulting changes converge to correct (desired) outputs. From this point of view it seems that the trainable neural network should be closer to an accepting symbolic system than a generating one, because it is easier to obtain a training signal in the accepting case. This is indeed the approach of [1], [2], [16], [36] and many others.

Now, how complex languages can biological neural nets "recognize"? Strictly speaking, all the languages we can ever recognize should be regular, because at some moment we are able to keep in mind only a limited amount of information. Really most of the languages we recognize (for example languages of syntactical description of visual images, which we know the brain uses [13]) are finite or without the need of recurrent application of the same rule(s) many times, which is typical for formal languages. This, on the one hand, could in an extreme case lead to an impression of the brain as a system of *finite state automata* (FSA). On the other hand, the effectiveness of neural processing of such languages clearly couldn't be by using FSA machinery. To give a rather expressive example, we know how long the transfer of one stimulus through the neuron and the following relaxation takes – something like "neural step" [3]. A simple reflex reaction, of a child who suddenly runs into the road, takes only about 40–60 such time steps [20], done of course massively parallel!

We often perform tasks that are clearly non-regular. Hardly anybody is capable of multiplying two arbitrary ten digit numbers in his mind, even after long training – this seems to be a rather "nonneural" task. The most primitive tool we can use is paper and pencil – external memory for symbol manipulation. Imagine, how we would recognize e.g. strings of language $\{0^n 1^n | n \in \mathcal{N}\}$. Moreover, the brain also contains some very specialized structures, created not by training of an individual, but by evolution [25].

To conclude, we use specialized tools for accepting non-regular language, that were not created by a simple training within our brains. This again leads us to the necessity of hybrid architectures, already expressed in the previous section.

2.3 Theoretical computational power of the neural nets

Now, let's briefly characterize ANN models with enough computational power to accept four basic classes of formal languages. The models are size-independent on the length of the processed string. In the later text a *recurrent neural network* (RNN) is referred to. It is one-layer fully interconnected ANN with N neurons and n inputs, with a dynamics characterized by the equation (2). In addition to (1), u_j are the inputs of the network with weights v_{ij} for i-th neuron, c_i is the threshold value. In the following four cases of equivalence the bottom-left threshold functions of the Fig.1 has The network operates in discrete time and the input string is presented sequentially, i.e. one symbol at each step. The precise definition of language acceptance can be found in [32], [33]. Here we only note, that the used RNN has two binary outputs, the "data" one with accepted/rejected signal and the "validation" one, stating when the data are valid.

$$y_i^{(t+1)} = \theta_i(\sum_{j=1}^{N} w_{ij} y_j^{(t)} + \sum_{j=1}^{n} v_{ij} u_j^{(t)} + c_i), \quad i = 1, 2, \ldots, N. \tag{2}$$

1. Acceptance of regular languages: RNN with discrete outputs of all neurons, i.e. output of each neuron is 0 or 1. A result (output of the network: accepted/rejected) is provided in the next step after the last input symbol has been presented. This well-known result can be found in [22].

2. Context-free languages: RNN with rational-valued outputs and weights of the neurons. As above, a result is provided immediately after the input string has been presented. The precision of computations needed is dependent on (and limited by) the length of the input string. This is rather trivial consequence of the proof of the main result in [32], the principle of which is given in section

3.1. It is clear, that the stack of a pushdown automaton could be implemented by the same way as the tape of a *Turing machine* (TM) in [32].

3. Context-sensitive languages: RNN with rational-valued outputs and weights of the neurons, providing a result after some delay after the input string has been presented. Again the precision of computations needed is by the length of the input string. For deriving this result from [32], it is enough to note the difference between TM and the linear-bounded automaton [39].

4. Recursively enumerable languages: recurrent ANN with rational-valued outputs and weights of the neurons, providing a result after a delay, which can grow over any limit. The necessary precision of computation is also unbounded. This is proven in [32].

3 An overview of hybrid architectures

Architectures coupling symbolic and "neural" methods contain mostly separate neural and symbolic modules. The input of neural net has usually a form of symbolic knowledge (a string or a set of them, properly coded), its output is again interpreted in terms of symbolic knowledge (rules, semantic knowledge...) [38]. Rarely we can meet direct incorporating of symbolic and neural computation principles in one module [22], [35],

3.1 Computational power studies

The aim of these studies is mostly to give a proof of computational power of particular type of ANN. The proofs are constructive, i.e. the result is an ANN modelling the behavior of the the original system (usually some formal automaton), without care of computational complexity and/or model size. Some models are even of infinite size, although the original system is finite [37]. As an example of this approach we present results in [32]. The authors proved that RNN restricted to rational-valued weights and signals has the universal computational power, by constructing a neural model of any TM. The principle of the model (highly simplified) is as follows: First, the tape alphabet is coded by a set of natural numbers less than some b, the empty symbol being coded as 0. Then any string $r_1 r_2 \dots r_k$ over the tape alphabet can be coded as

$$C(r_1 r_2 \dots r_k) = \sum_{i=1}^{k} \frac{r_i}{b^i} \in (0,1). \tag{3}$$

(In fact the coding is more complicated to avoid the necessity of distinguishing between two very close numbers.) Now let p be the position of the head on the tape of TM. Let output value of neuron α code the portion $r_{p+1} r_{p+2} \dots$ of the tape by the coding (3). Let code portion $r_1 r_2 \dots r_p$ of the tape *in reverse order*. Then we can extract the representation of the symbol read by the head as $\lfloor b o_\beta \rfloor$, where

o_β is the output value of neuron β. Movement head to the right is represented by setting

$$o_\beta := (o_\beta + \lfloor b o_\alpha \rfloor)/b, \qquad o_\alpha := b o_\alpha - \lfloor b o_\alpha \rfloor, \tag{4}$$

and analogously to the left. Here $\lfloor x \rfloor$ is the integer part of x. Rules of TM are then represented by groups of neurons realizing proper logic function. The whole model consists of a fixed number (some hundreds) of neurons. The neurons α, β play the role of wheels rewinding the TM tape, the head being placed between them. Thus the unbounded length of tape is replaced by the unbounded precision of rational number.

If we extend domain of computation to real numbers, we obtain even more powerful (from the computational complexity point of view) devices than TMs (the power equals to TM consulting sparse oracle) [33].

There are (from the implementational point of view) two major drawbacks of this approach: Firstly, there is the necessity of unbounded precise rational number computation. Secondly, the architecture is "hardwired" and practically involves no adaptation. Moreover, it would be very difficult to extend this approach towards parallel working grammar system due to sequential access to the processed string.

3.2 Topology preserving models

These models are topologically similar to the original system, including both number of elements (neurons, states, rules) and the connections between them. Typically each element of one system is modelled by one or a group of the model. The advantage of this approach is often the possibility of and giving a proof of functional equivalence of the systems in both directions, with respect to the computational complexity. Also direct manipulation with a symbolic knowledge within the ANN is then possible.

As an example the classical paper [22] can be presented, establishing equivalence between finite automata and RNNs with discrete-valued outputs. In [15], neural structures equivalent to the state transitions of FSA are used for inserting symbolic knowledge into a RNN. Also some of the models referred to in [38] have this property, what can lead to using a-priori training (see end of this section). In [35] (see Fig. 2) we deal with computational equivalence of eco-grammar systems and ANNs in both directions. Here parallel data processing is fully preserved, which allows to construct model of computational complexity very close to original system.

Again, the disadvantage of this approach is the fact that neural structures topologically similar to the original system are often heterogeneous, what can cause problems with learning, because the most of learning algorithms require a (computationally) homogeneous network for propagating a learning signal through it and making gradient-directed steps towards the solution.

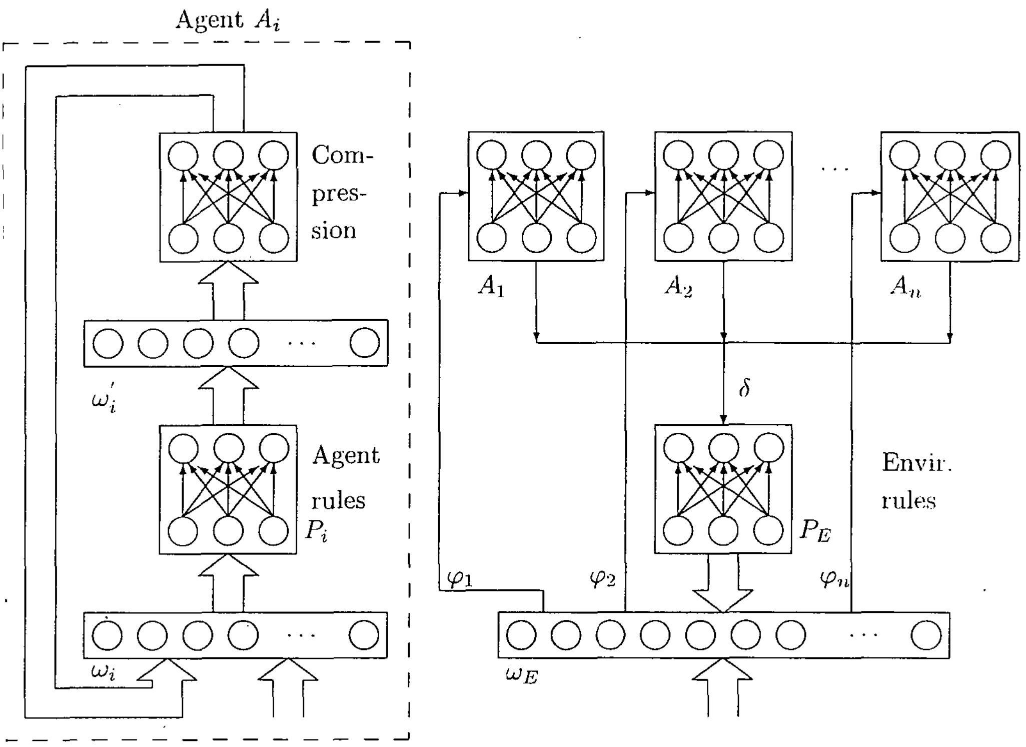


Figure 2: The neural model of an eco-grammar system.

3.3 Adaptive models

These models are clearly the most promising ones and are often applied to the grammatical inference problems, natural language understanding or as a part of knowledge and reasoning systems [38], [4].

A lot of recent works deal with a grammatical inference of some accepting device, as FSA [1], [15], or deterministic *pushdown automaton* (PDA) [36]. This approach seems to be most suitable for studying interactions of neural and symbolic parts of the systems, for instance methods for extracting FSA or PDA from trained RNN of various orders. It seems that if one step of the modelled accepting device is governed by n factors (for PDA $n = 3$: a state, a tape symbol, a stack symbol), n-th order neurons suit best [16], [36]. More complex languages (context-sensitive) accepting devices can be also inferred using RNNs, but in these cases extending "nonneural" inferring algorithms are used [2].

As an example we present the results in [1], using the extraction of unbiased FSA from the trained homogeneous first order RNN:

First, RNN (with one output unit giving value in $\langle 0, 1 \rangle$, which corresponds to rejecting/accepting of input string) is trained using positive and negative examples. Then a prefix tree of these strings is built and its nodes are identified with the states of an *unbiased FSA* (UFSA - its definition is given in [1]). Each state is assigned a set of corresponding hidden layer neuron output values, forming a cluster (initially consisting of a single point) in the state space of the hidden layer neuron outputs. Then, a hierarchical clustering in this space is performed. Whenever two clusters are merged, parallel merging of their associated states is performed in the UFSA representation. After each step the consistency of the UFSA with the training set is tested. Whenever an inconsistency occurs, the process is stopped. The advantage of this approach is that there is no estimation of the number of the UFSA states, minimal cluster distance and so on; only the metric for cluster distance must be defined.

A lot of similar results cited above lead us to the opinion that (from the adaptability point of view) there is no need for topological equivalence of the original system and the model. The most natural representation of one state of the UFSA in RNN is a cluster of neuron states, which has nothing to do with the net topology. Generally, even if we haven't any prior knowledge of the complexity of the grammar inferred, the RNN can create a model of this grammar in the space of its internal states. Then we apply some heuristic algorithm for extracting this grammar making it as simple as possible to be consistent with the set of training strings. The drawback of this approach is again the problematic interpretation of the results of ANN and its transformation to a symbolic knowledge.

4 Hybrid models with grammar systems

One possible way of constructing such models could be incorporating some symbol manipulating principles into the neural nets, what's in opposite to usual approaches, integrating neural nets as a part of rather complex symbolic knowledge manipulating architectures. Some problems of this intention are discussed here.

First, a communication between agents in distributed systems (including various kinds of systems from multicomputers to neural nets) could in general be characterized as *by request* (when the agents are passive until any external stimulus) or *by command* (when the agents activelly offers their free cappacity and/or results). The terminology is inspired by [7], [28], where PCGSs are studied. It seems, that the communication by request is closer to the nature of ANNs, when every agent (neuron), through the weights of its connection, can decide which other agents it gets information from.

As it was mentioned in section 1, dynamically defined and parallel communication between agents and parallel access of the agents to different parts of the processed string is typical for some types of grammar systems. This is contrasted to the nature of ANNs with fixed communication graphs. To introduce a "communication dynamics" into ANNs, one have to create highly interconnected structures (including all possible communication branches) equipped with a mechanism of dynamic activation of the connections needed. (This can be directed e.g. by an occurance of some patterns in the string(s) processed by the system) Due to the conclusions in section 2, pre-defined specialized structures allowing this dynamics need to be used as a part of NN. See [36] where a "neural PDA" is presented, equipped with external continuous stack. Unfortunately, the existence of such structures is often in contradiction with the request for homogenity of network dynamics, because many training algorithms require differentiability of the transfer function of the neurons and groups of neurons.

Another problem can be caused by the fact that the recurrent application of the same rule (group of rules) to the generated or accepted string is typical for grammars (and thus also agents of grammar systems). ANNs, in contrast, obviously reach a stable state in a few steps, due to their nature as an asymptotically stable dynamical system. In the models described in section 3.3 this problem has been overcome by the sequential reading of the input string, providing stimuli not allowing the system dynamics to stabilize. This solution leads nevertheless to the lack of parallelism.

Also a memory for storing the processed string, capable of replacing some substrings by ones of different length, is a very "nonneural" device. Both these problems seem to be best solved by adding another special structure integrating the one-step actions of agents into the processed string and re-loading the result into net inputs. During the phase of learning, such a network could be "unfolded in time", just as in the *Backpropagation algorithm* [18], [36] and similar ones, giving promising results.

5 Conclusion

Although there have been many successfully working hybrid neural-symbolic architectures referred to in the last two sections, it still seems to be possible to find a closer connection of grammar systems and ANNs, particularly exploiting massive parallelism and no need of centralized control in both system types. A highly interconnected RNN looks as a promising way of constructing trainable "neural" accepting symbol-processing system, eventually with a lot of autonomous modules, equipped with some special structures for symbol storing, transfer and some other tasks which are inherently "nonneural". It would be capable of parallel string processing and inter-agent communication.

There are many open problems, including the definition of training hybrid systems and proofs of their stability and convergence. Some of them have been partially solved already, as the training of ANNs is only now beginning to be deeper understood task [4]. Anyway, parallelism and decentralized processing are the main features of present computer architecture trends.

References

[1] R. Alquézar, A. Sanfeliu: A hybrid connectionist-symbolic approach to regular grammatical inference based on neural learning and hierarchical clustering. In: *Grammatical Inference and Applications,* proc. of ICGI-94, R. Carrasco, J. Oncina (eds.), Springer-Verlag 1994, pp.203–211.

[2] R. Alquézar, A. Sanfeliu, J. Cueva: Learning of context-sensitive language acceptors through regular inference and constraint induction. In: *Proc. ICGI'96,* L.Miclet and C.de la Higuera (eds.), Springer-Verlag, Lecture Notes in Artificial Intelligence 1147, pp.134-145, 1996.

[3] P. Århem: On artificial intelligence and neurophysiology: two necessary questions. In: J.L. Casti, A. Karlqvist (eds.): *Real Brains, Artificial Minds.* North Holland, Elsevier Science Publishing Co., Inc., N.Y. 1987.

[4] L.A. Bookman, R. Sun: Editorial: Integrating neural and symbolic processes. *Connection Science*, vol. 5, No. 3-4, p.203, 1993.

[5] E. Csuhaj-Varjú, A. Kelemenová, J. Kelemen, Gh. Păun: Eco-grammar systems: A gramatical framework for life-like interactions. Silesian University CSR-TR-95/2, Opava, 1995.

[6] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun: *Grammar Systems: a Grammatical Approach to Distribution and Cooperation.* Gordon and Breach Science Publishers, London, 1994.

[7] E. Csuhaj-Varjú, J. Kelemen, Gh. Păun: Grammar systems with WAVE-like communication. *Computers and Artificial Intelligence*, 15 (1996), 419-436.

[8] J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, Heidelberg, 1989.

[9] J. Dassow, Gh. Păun, G. Rozenberg: Grammar Systems. In: *Handbook of Formal Languages, Vol. 2,* G. Rozenberg and A. Salomaa (eds.), Springer, Heidelberg, 1997, 155-213.

[10] H. Fernau, H. Bordihn: Remarks on accepting parallel systems. *Intern. J. Computer Math.*, vol. 56, 51–67.

[11] H. Fernau, M. Holzer, H. Bordihn: Accepting multi-agent systems: the case of cooperating distributed grammar systems. *Computers and Artificial Intelligence*, vol. 15 (1996), No. 2–3, 123–139.

[12] R. Freund, L. Kari, Gh. Păun: DNA computing based on splicing: the existence of universal computers. Technische Universität Wien TR 185-2/FR-2/95, Wien 1995.

[13] K.S. Fu: *Syntactic Pattern Recognition and Applications.* Englewood Cliffs, NJ, Prentice-Hall, 1982.

[14] M.R. Genesereth, N. J. Nilsson: *Logical Foundations of Artificial Intelligence.* Morgan Kaufman, Los Altos, Cal., 1987.

[15] C. Giles, C. Omlin: Extraction, insertion and refinement of symbolic rules in dynamically-driven recurrent neural networks. *Connection Science*, vol. 5, No. 3-4, p.307, 1993.

[16] M. Goudreau, C. Giles, S. Chakradhar, D. Chen: First-order vs. second-order single layer recurrent neural networks. In: *IEEE Trans. on Neural Networks*, vol. 5, No. 3, p. 511, 1994.

[17] J. Hendler: On the need for hybrid systems. *Connection Science*, vol. 1, No. 1, p.0, 1989.

[18] J. Hertz et.al.:*Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[19] J.E. Hopcroft, J.D. Ullman: *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, 1979.

[20] J. Hořejš: A view on neural network paradigms development. *Neural Network World*, IDG Prague, 1991,1992.

[21] A. Kelemenová, E. Csuhaj-Varjú: Languages of colonies. *Theoretical Computer Science* 134, Elsevier, 1994, 119–130.

[22] S.C. Kleene: Representation of events in nerve nets and finite automata. In: C.E. Shannon, J. McCarthy (eds.): *Automata Studies*, 3–41, Princeton Univ. Press 1956.

[23] V. Kůrková: Universal approximation using feedforward neural Gaussian bar units. *Proceedings of ECAI'92*, Vienna, 1992, 193-197.

[24] M. Malita, Gh. Stefan: The eco-chip: A physical support for artificial life systems. In: [26], 260–275.

[25] J.G. Nicholls, A.R. Martin, B.G. Wallace: *From Neuron to Brain.* Third edition, Sinauer Ass., Sunderland, Ma, 1992.

[26] Gh. Păun (ed.): *Artificial Life: Grammatical Models.* The Black Sea University Press, Bucharest, 1995.

[27] Gh. Păun: Foreword. In: [26], v-ix.

[28] Gh. Păun, L. Sântean: Parallel communicating grammar systems: the regular case, *Ann. Univ. Buc., Math.-Informatics Series 38* (1989) 55–63.

[29] A.S. Perelson: Immune network theory. *Immunological Reviews,* Santa Fe Institute, No. 110 (1989), 5–36.

[30] T. Poggio, F. Girosi: Networks and the best approximation property. *Biological Cybernetics*, 63, 1990, 169-176.

[31] G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems.* Academic Press, New York 1980.

[32] H. Siegelmann, E. D. Sontag: On the computational power of neural nets. In: *Proc. Fifth ACM Workshop on Computational Learning Theory*, 440–449, Pittsburgh 1992.

[33] H.T. Siegelmann, E. D. Sontag: Analog computation via neural networks. *Theoretical Computer Science*, 1994.

[34] K.P. Simpson: *Artificial neural systems.* Pergamon Press, N.Y., 1990.

[35] P. Sosík: Eco-grammar systems and artificial neural networks. *Computers and Artificial Intelligence*, No. 2–3 (1996), 247–264.

[36] G. Sun, C. Giles, H. Chen, Y. Lee: The neural network pushdown automaton: model, stack and learning simulations. University of Maryland TR Nos. UMIACS-TR-93-77 & CS-TR-3118, 1995.

[37] D. Wolpert: A computationally universal field computer which is purely linear. *Los Alamos National Laboratory report LA-UR-91-2937.*

[38] A. Wilson, J. Hendler: Linking symbolic and subsymbolic computing. *Connection Science*, vol. 5, No. 3-4, p.395, 1993.

[39] D. Wood: *Theory of Computation.* John Wiley & Sons, 1987.

Various Communications in PC Grammar Systems [*]

György Vaszil [†‡]

Abstract

A slightly modified communication protocol called immediate communication is introduced for PC grammar systems and the generative power of these systems is shown to be equal to what we call homogeneous systems, systems with queries of a special form. To acquire this result we also introduce a generalization of returning systems, called systems with returning languages.

1 Introduction

Parallel communicating grammar systems (PC grammar systems) were introduced in [6] as a grammatical description of the so-called classroom model of problem solving. The agents of the classroom are generative grammars, which all operate on their own sentential form, these represent the subsolutions of the overall solution which is the language generated by the whole system. During their operation the agents may communicate, they may exchange their strings with each other. The language generated by the system is the language generated by the classroom leader which is one of the component grammars, usually called the master grammar of the system.

Parallel communicating grammar systems have been the subject of detailed study over the past few years. See [3], [4], [5] for results on their generative power, and [2] on their size parameters. A summary of their properties can be found in the monograph [1].

The power of PC grammar systems is measured by their generative capacity, which may depend on a number of factors. The type of the component grammars and the number of the components are obviously very important among these factors, but there are many others to be considered.

In their paper [6], Gh. Păun and L. Santean considered variants with a universal clock and two basic methods for communication. The presence of the universal clock means that all components use their rules synchronized in time, one derivation step is taken by the system with all components using one of their rewriting rules.

[*]Presented at the workshop *Grammar Systems: Recent Results and Perspectives, July 26-27, 1996, Budapest.*

[†]Research supported by the Hungarian Scientific Research Fund, OTKA no. T017105.

[‡]Computer and Automation Institute Hungarian Academy of Sciences Kende utca 13-17, 1111 Budapest, Hungary, email: `vaszil@luna.aszi.sztaki.hu`

Communication in this construction is realized with the aid of special nonterminals, the so-called query symbols. Each of these symbols points to one of the component grammars of the system, and when one of them appears in a sentential form, it has to be replaced with the current sentential form of the component it refers to.

This is communication by request, which has two basic variants. One is called returning communication: after a component sends its string to an other component, it must return to its start symbol (or axiom) and begin to generate a new string. The other is called non-returning communication: the component which sends its string keeps a copy for itself and continues to process it after communication.

In the following we keep the basic features of the original model. We will consider synchronized systems with communication by request, but propose a slight change in the communication protocol introducing *immediate communications*, and investigate the impact of this modification on the generative power. To do this, we also generalize the notion of a returning communication by introducing *systems with returning languages.*

The results we obtain will show that the languages generated with immediate communication can be generated with a very much simplified form of query rules using the original protocol. This simple form of queries is what we call *homogeneous.*

2 Preliminaries

The reader is assumed to be familiar with the basics of formal language theory; further details can be found in [7].

The set of all words over an alphabet V and the empty word are denoted by V^* and ϵ respectively, the family of regular, linear and context-free grammars by REG, LIN and CF, respectively. $\mid w \mid$ and $\mid w \mid_X$ denotes the length of a word w and the number of occurences of symbols from set X in w, respectively.

Now we recall the notion of parallel communicating grammar systems from [6], for more material see the monograph [1].

Definition 2.1 A *parallel communicating grammar system* with n components, where $n \geq 1$, (a PC grammar system, for short), is an $(n+3)$-tuple $\Gamma = (N, K, T, G_1, \ldots, G_n)$, where N is a nonterminal alphabet, T is a terminal alphabet and $K = \{Q_1, Q_2, \ldots, Q_n\}$ is an alphabet of *query symbols.* N, T, and K are pairwise disjoint sets, $G_i = (N \cup K, T, P_i, S_i)$, $1 \leq i \leq n$, called a *component* of Γ, is a usual Chomsky grammar with nonterminal alphabet $N \cup K$, terminal alphabet T, a set of rewriting rules P_i and an axiom or (a start symbol) S_i. G_1 is said to be the *master* (grammar) of Γ.

Definition 2.2 Let $\Gamma = (N, K, T, G_1, \ldots, G_n)$, $n \geq 1$, be a PC grammar system as above. An n-tuple $(x_1, \ldots, x_n)$, where $x_i \in (N \cup T \cup K)^*$, $1 \leq i \leq n$, is called a *configuration* of Γ. $(S_1, \ldots, S_n)$ is said to be the *initial configuration.*

PC grammar systems change their configurations by performing direct derivation steps.

Definition 2.3 Let $\Gamma = (N, K, T, G_1, \ldots, G_n)$, $n \geq 1$, be a PC grammar system and let $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ be two configurations of Γ. We say that $(x_1, \ldots, x_n)$ *directly derives* $(y_1, \ldots, y_n)$, denoted by $(x_1, \ldots, x_n) \Rightarrow (y_1, \ldots, y_n)$, if one of the next two cases hold:

1. There is no x_i which contains any query symbol, that is, $x_i \in (N \cup T)^*$ for $1 \leq i \leq n$. In this case $x_i \Rightarrow_{G_i} y_i$. For $x_i \in T^*$ we have $x_i = y_i$. The system is blocked, if there is an x_j with $\mid x_j \mid_N \neq 0$ and none of the rules of P_j can be applied to x_j.

2. There is some x_i, $1 \leq i \leq n$, which contains at least one occurrence of query symbols. Let x_i be of the form $x_i = z_1 Q_{i_1} z_2 Q_{i_2}, \ldots, z_t Q_{i_t} z_{t+1}$, where $z_j \in (N \cup T)^*$, $1 \leq j \leq t+1$ and $Q_{i_l} \in K$, $1 \leq l \leq t$. In this case $y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$, where x_{i_l}, $1 \leq l \leq t$ does not contain any query symbol. In *returning* systems $y_{i_l} = S_{i_l}$, $1 \leq l \leq t$, in *non-returning* systems $y_{i_l} = x_{i_l}$, $1 \leq l \leq t$. If some x_{i_l} contains at least one occurrence of query symbols, then $y_i = x_i$ and also $y_{i_l} = x_{i_l}$, $1 \leq l \leq t$.

If for all x_i with $\mid x_i \mid_K \neq 0, x_i = z_1 Q_{i_1} z_2 Q_{i_2}, \ldots, z_t Q_{i_t} z_{t+1}$ there is at least one Q_{i_j}, $1 \leq j \leq t$ that x_{i_j} also contains a query symbol, then the system is blocked due to a circular query.

For all i, $1 \leq i \leq n$, for which y_i is not specified above, $y_i = x_i$.

The first case is the description of a rewriting step: If no query symbols are present in any of the sentential forms, then each component grammar uses one of its rewriting rules except those which have already produced a terminal string. The derivation is blocked if a sentential form is not a terminal string, but no rule can be applied to it.

The second case describes a communication: If some query symbol, say Q_i, appears in a sentential form, then the rewriting stops and a communication step must be performed. The symbol Q_i must be replaced by the current sentential form of component G_i, say x_i, supposing that x_i does not contain any query symbol. If this sentential form also contains some query symbols, then first these symbols must be replaced with the requested sentential forms. If this condition cannot be fulfilled (a circular query appeared), then the derivation is blocked.

Let $\Rightarrow_{rew}$ and $\Rightarrow_{com}$ a denote a rewriting and a communication step respectively.

If the sentential form of a component was communicated to another, this component can continue its own work in two ways: In so-called *returning* systems, the component must return to its axiom and begin to generate a new string. In *non-returning* systems the components do not return to their axiom, but continue to process the current string.

A system is *centralized* if only the component G_1 is allowed to introduce query symbols, otherwise it is *non-centralized.*

By the word *query* we refer to a sentential form containing at least one query symbol. A query is satisfied by a *communication* replacing the query symbols with the requested sentential forms. This may be done in one or more *communication* *steps.* The phrase *communication step* is used to denote the process of satisfying the query symbols, which can be satisfied in "parallel". For example the returning communication prescribed by $(Q_2,\ Q_3,\ \alpha,\ Q_3)$ takes two communication steps to realise: first we get $(Q_2,\ \alpha,\ S_3,\ \alpha)$, and then $(\alpha,\ S_2,\ S_3,\ \alpha)$. The two consecutive steps together will be referred to as a *communication sequence.*

Let $\Rightarrow^+$ and $\Rightarrow^*$ denote the transitive, and the reflexive, transitive closure of $\Rightarrow$ respectively.

Definition 2.4 Let k be a natural number, $k \geq 1$ and let the k step derivations of a PC grammar system be denoted by $(S_1,\ldots,S_n) = (\alpha_1^0,\ldots,\alpha_n^0) \Rightarrow^k (\alpha_1^k,\ldots,\alpha_n^k)$ where $(\alpha_1^k,\ldots,\alpha_n^k)$ is the configuration reached by the system in k steps. The *language* generated by a PC grammar system Γ is

$$L(\Gamma) = \{\alpha_1^k \in T^* \mid (S_1,\ldots,S_n) \Rightarrow^k (\alpha_1^k,\ldots,\alpha_n^k), \alpha_1^j \notin T^*, 1 \leq j < k\}.$$

Thus, the generated language consists of the terminal strings first appearing as sentential forms of the master grammar, G_1.

Let the classes of returning and non-returning PC grammar systems with at most n components of type X, $X \in \{REG,\ LIN,\ CF\}$ and $n \geq 1$ and the corresponding language classes be denoted by PC_nX, NPC_nX and $\mathcal{L}(PC_nX)$, $\mathcal{L}(NPC_nX)$ for non-centralized systems and CPC_nX, $NCPC_nX$, $\mathcal{L}(CPC_nX)$, $\mathcal{L}(NCPC_nX)$ for centralized systems, respectively . When an arbitrary number of components is considered we use $*$ instead of n

3 PC grammar systems with immediate communications

In the communication protocol of [6] the query symbols occurring in one string can only be replaced in one communication step. If it is not possible, the system has to wait until all the query symbols of a sentential form can be replaced. For example the queries (Q_2Q_3, Q_3, a) are satisfied in the returning mode with the following two steps:

$(Q_2Q_3, Q_3, a) \Rightarrow_{com} (Q_2Q_3, a, S_3) \Rightarrow_{com} (aS_3, S_2, S_3)$. Observe that Q_3 of the query Q_2Q_3 did not get replaced in the first step.

In the *immediate communication mode* we allow the replacement of all query symbols that request sentential forms not containing other query symbols. The query above will be satisfied with:

$(Q_2Q_3, Q_3, a) \Rightarrow_{com} (Q_2a, a, S_3) \Rightarrow_{com} (aa, S_2, S_3)$.

Definition 3.1 Let $\Gamma = (N,\ K,\ T, G_1,\ldots,G_n)$, $n \geq 1$, be a usual PC grammar system and let $(x_1,\ldots,x_n)$ and $(y_1,\ldots,y_n)$ be two configurations of Γ. We say that $(x_1,\ldots,x_n)$ *directly derives* $(y_1,\ldots,y_n)$, *with immediate communications* if one of the next two cases holds:

1. There is no x_i which contains query symbols, $x_i \in (N \cup T)^*$ for $1 \leq i \leq n$. In this case the system performs a *rewriting step* denoted by $(x_1, \ldots, x_n) \Rightarrow (y_1, \ldots, y_n)$, where $x_i \Rightarrow y_i$ in G_i. For $x_i \in T^*$ we have $x_i = y_i$ and the system is blocked if there is an x_j with $| x_j |_N \neq 0$ and no rule of P_j can be applied to x_j.

2. There is some x_i, $1 \leq i \leq n$, which contains at least one occurrence of query symbols. In this case, the system performs an *immediate communication step* denoted by $(x_1, \ldots, x_n) \Rightarrow (y_1, \ldots, y_n)$, in the following way:

Let x_i be of the form $x_i = z_1 Q_{i_1} z_2 Q_{i_2}, \ldots, z_t Q_{i_t} z_{t+1}$, where $z_j \in (N \cup T)^*$, $1 \leq j \leq t+1$ and $Q_{i_l} \in K$, $1 \leq l \leq t$. Now $y_i = z_1 \delta_{i_1} z_2 \delta_{i_2} \ldots z_t \delta_{i_t} z_{t+1}$, where δ_{i_l}, $1 \leq l \leq t$ is x_{i_l} if x_{i_l} does not contain any query symbol, or δ_{i_l} is Q_{i_l} if x_{i_l} contains at least one query symbol. If $\delta_{i_l} = x_{i_l}$, then in *returning* systems $y_{i_l} = S_{i_l}$, in *non-returning* systems $y_{i_l} = x_{i_l}$, $1 \leq l \leq t$. If $\delta_{i_l} = Q_{i_l}$, then $y_{i_l} = x_{i_l}$, $1 \leq l \leq t$ in both type of systems. The derivation is blocked by a circular query if for all i with $| x_i |_K \neq 0$, $x_i = z_1 Q_{i_1} z_2 Q_{i_2}, \ldots, z_t Q_{i_t} z_{t+1}$ and $y_i = z_1 \delta_{i_1} z_2 \delta_{i_2} \ldots z_t \delta_{i_t} z_{t+1}$, there is a $\delta_{i_l} = Q_{i_l}$, for some l, $1 \leq l \leq t$.

The first case is the description of a usual rewriting step, the second case describes an immediate communication: if more query symbols, say Q_i, Q_j, appear in a sentential form and x_i, the current sentential form of component G_i, does not contain query symbols, then Q_i must be replaced by x_i, even if Q_j can not be replaced by x_j, the current sentential form of G_j in the same step, because it contains further queries. In short, strings without query symbols must be communicated immediately.

Let the class of PC grammar systems of type X with immediate communications and n components of type Y and the corresponding language classes be denoted by fX_nY and $\mathcal{L}(fX_nY)$ respectively, $X \in \{PC, NPC, CPC, CNPC\}$, $Y \in \{REG, LIN, CF\}$. If an arbitrary number of components is considered we put $*$ instead of n.

In a communication sequence with immediate communication, the strings requested by other components are always sent to their destination without any delay if they do not contain further queries. Using the usual communication protocol, it is possible that a sentential form is requested by two other components, but sent to only one of them. For example, if x_i is requested by $x_k = Q_i Q_j$ and $x_l = Q_i$, but $x_j = Q_m$ also contains a query symbol, then x_i can not be sent to x_k, until the query symbol of x_j, the other requested sentential form is replaced. This makes it possible in returning systems, that a query symbol is replaced by the axiom of the queried component instead of the string present at the appearance of the query. In the example above the result of the communication sequence is the following: $y_k = S_i x_m$, $y_l = x_i$, while using immediate communication it would be $y_k = x_i x_m$, $y_l = x_i$.

In a number of cases this difference can not influence the results of a communication sequence. For example, nonreturning systems do not return to their axiom during the communication sequence, centralized systems never request strings containing query symbols and regular or linear components have sentential forms containing at most one query symbol. In these cases the generative capacity of immediate communications coincides with the usual communication modes.

Observation 1

1. $\mathcal{L}(fNPC_nX) = \mathcal{L}(NPC_nX)$, $X \in \{REG, LIN, CF\}$
2. $\mathcal{L}(fPC_nX) = \mathcal{L}(PC_nX)$, $X \in \{REG, LIN\}$
3. $\mathcal{L}(fCPC_nX) = \mathcal{L}(CPC_nX)$, $X \in \{REG, LIN, CF\}$

In the next section we will investigate the generative power of the remaining case, the case of *non-centralized context-free returning* systems.

4 The power of returning systems with immediate communications

In this section we study the generative capacity of *context-free non-centralized returning systems with immeditae communications*, but first we introduce the notion of PC grammar systems with returning languages which will be of help in our investigations.

A PC grammar system with returning languages is a natural extension of a returning system. Each component has an associated language, the so-called returning language. After communication they are allowed to start a new derivation with any word of this language instead of starting with their axiom again.

Definition 4.1 A *PC grammar system with returning languages* is a $(2n+3)$-tuple $\Gamma = (N, K, T, R_1, \ldots, R_n, G_1, \ldots, G_n)$, where N, K, T and $G_1, \ldots, G_n$ are the same as usual, and $R_1, \ldots, R_n$ are non-empty sets of words over $(N \cup T)$, the so-called *returning languages*. $(R_i \subseteq (N \cup T)^*, R_i \neq \{\epsilon\}, R_i \neq \emptyset, 1 \leq i \leq n)$.

The system works like a usual returning system, but after communication components may start a new derivation with any word of their returning language.

Let the class of context-free PC grammar systems with returning languages of n components of type X, $X \in \{PC, fPC\}$ and the corresponding language classes be denoted by rX_nCF and $\mathcal{L}(rX_nCF)$, respectively.

With the aid of systems with returning languages we will be able to prove our theorem about the power of immediately communicating systems, which will turn out to be the same as that of usual returning systems with a certain form of queries, which we will call homogeneous queries.

Definition 4.2 Let us call a query *homogeneous*, if all query symbols contained in the corresponding sentential form request the same string, that is, the sentential form is of type $\alpha_1 Q_i \alpha_2 Q_i \ldots \alpha_{t-1} Q_i \alpha_t$, where $1 \leq i \leq n$, $2 \leq t$ and $\alpha_j \in (N \cup T)^*$, $1 \leq j \leq t$.

A *component with homogeneous queries* is a component grammar G_i, $1 \leq i \leq n$, which is allowed to introduce only homogeneous queries, it has no rule of the form $X \to \alpha Q_i \beta Q_j \gamma$, with $i \neq j$, $\alpha, \beta, \gamma \in (N \cup T \cup K)^*$.

A PC grammar system is called ***homogeneous***, if it has components with homogeneous queries only.

Let the class of homogeneous PC grammar systems of type X with n context-free components and the corresponding language classes be denoted by hX_nCF and $\mathcal{L}(hX_nCF)$ respectively, where $X \in \{PC, NPC\}$.

The following inclusion is obvious because communication sequences with homogeneous queries produce the same result in the usual and in the immediate communication modes.

Observation 2 $\mathcal{L}(hPC_nCF) \subseteq \mathcal{L}(fPC_nCF)$

Our aim is to prove also the converse inclusion. First we present a lemma about systems with returning languages.

Lemma 4.1 *Let Γ be a returning PC grammar system with immediate communications, having n context-free components and finite returning languages R_i consisting of only nonterminal symbols, $R_i \subseteq N$, $1 \le i \le n$.*

If these conditions hold, then there exists Γ', a returning system with immediate communications and $4n$ components which generates the same language as Γ.

Proof: Let $\Gamma = (N,\ K,\ T,\ G_1, R_1, \ldots, G_n, R_n) \in rfPC_nCF$ with nonterminal alphabet N, set of query symbols K, terminal alphabet T, n context-free components $G_1, \ldots, G_n$ and returning languages $R_1, \ldots, R_n$, $R_i \subseteq N$, $1 \le i \le n$. Now let $\Gamma' \in fPC_{4n}CF$ be the following:

$$\Gamma' = (N', K', T, G_1^1, .., G_n^1, G_1^2, .., G_n^2, G_1^a, .., G_n^a, G_1^t, .., G_n^t)$$

where

$$\begin{aligned}
N' &= \{S_i^1,\ S_i^2,\ S_i^a, S^{a'}, S_i^t, S_i^{t'}, S_i^{t''} \mid 1 \le i \le n\} \cup \\
&\quad\ \{X,\ [X] \mid X \in N\}, \\
P_i^1 &= \{S_i^1 \to Q_i^2,\ S_i^1 \to Q_i^a\} \cup \{X \to [X] \mid X \to \alpha \in P_i\}, \\
P_i^2 &= \{S_i^2 \to Q_i^1\} \cup \{[X] \to \alpha \mid X \to \alpha \in P_i, \alpha \in (N \cup T)^*\} \cup \\
&\quad\ \{[X] \to \alpha_1 Q_{j_1}^1 \alpha_2 \ldots Q_{j_t}^1 \alpha_{t+1} \mid X \to \alpha_1 Q_{j_1} \alpha_2 \ldots Q_{j_t} \alpha_{t+1} \in P_i, \\
&\quad\ \alpha_l \in (N \cup T)^*, 1 \le l \le t+1\}, \\
P_i^a &= \{S_i^a \to [S_i], S_i^a \to S_i^{a'}, S_i^{a'} \to [Y] \mid Y \in R_i\} \text{ and} \\
P_i^t &= \{S_i^t \to S_i^{t'}, S_i^{t'} \to S_i^{t''}, S_i^{t''} \to Q_i^a\} \cup \{[Y] \to [Y]', [Y]' \to Q_i^a \mid Y \in R_i\}
\end{aligned}$$

for $1 \le i \le n$.

The system has four n-tuples of component grammars, and the rules $X \to \alpha \in P_i$, $1 \le i \le n$ of Γ are broken into two parts $X \to [X]$ and $[X] \to \alpha$. $G_1^1, \ldots, G_n^1$ contain the first parts $X \to [X]$ and $G_1^2, \ldots, G_n^2$ the second parts $[X] \to \alpha$.

They work in the following way: all G_i^1 use the first part of some rules while G_i^2 introduce the queries Q_i^1. Now the sentential forms of G_i^1 replace the query symbols in G_i^2, where the application of the rules is finished using their second parts. Next the sentential forms are communicated to G_i^1 and the process starts all over again. The assistant components G_i^a and G_i^t are used to simulate the return of a component to some symbol of the returning language, $1 \leq i \leq n$.

First we show how the initial derivation step of Γ is simulated by Γ'. We start from a configuration

$$(S_1^1, \dots, S_n^1,\ S_1^2, \dots, S_n^2,\ S_1^a, \dots, S_n^a,\ S_1^t, \dots, S_n^t)$$

and get

$$(\delta_1^1, \dots, \delta_n^1,\ Q_1^1, \dots, Q_n^1,\ \delta_1^a, \dots, \delta_n^a,\ S_1^{t\prime}, \dots, S_n^{t\prime}),$$

where δ_i^1 is either Q_i^2 or Q_i^a. If some $\delta_j^1 = Q_j^2$ then the derivation is blocked by a circular query. δ_i^a is either $[S_i]$ or $S_i^{a\prime}$. If some $\delta_i^a = S_i^{a\prime}$ then the derivation is also blocked since δ_i^a is passed to G_i^2 and P_i^2 does not contain rules to rewrite $S_i^{a\prime}$. So we must have

$$(Q_1^a, \dots, Q_n^a,\ Q_1^1, \dots, Q_n^1,\ [S_1], \dots, [S_n],\ S_1^{t\prime}, \dots, S_n^{t\prime}).$$

After one communication step we get

$$(S_1^1, \dots, S_n^1,\ [S_1], \dots, [S_n],\ S_1^a, \dots, S_n^a,\ S_1^{t\prime}, \dots, S_n^{t\prime}),$$

and then

$$(\delta_1^1, \dots, \delta_n^1,\ \alpha_1^2, \dots, \alpha_n^2,\ \delta_1^a, \dots, \delta_n^a,\ S_1^{t\prime\prime}, \dots, S_n^{t\prime\prime}).$$

Here α_i^2 differ only in the indices of the query symbols from the strings produced by G_i of Γ, $1 \leq i \leq n$, through the first rewriting step. More precisely if $(S_1, ..., S_n) \Rightarrow_{rew} (\alpha_1, ..., \alpha_n)$, then $\alpha_i^2 = \alpha_i$ if $|\ \alpha_i\ |_K = 0$, $\alpha_i^2 = \alpha_{i_1} Q_{j_1}^1 \alpha_{i_2} ... Q_{j_t}^1 \alpha_{i_t}$ if $\alpha_i = \alpha_{i_1} Q_{j_1} \alpha_{i_2} ... Q_{j_t} \alpha_{i_t}$. The δ_i^1 are either Q_i^a or Q_i^2 and δ_i^a are either $S_i^{a\prime}$ or $[S_i]$, $1 \leq j \leq n$. If $\delta_j^1 = Q_j^a$ for some j, $1 \leq j \leq n$, then the system is going to be blocked after the next rewriting step, when G_j^t introduces Q_j^a, because P_j^t does not contain rules to rewrite $[S_j]$ or $S_j^{a\prime}$. If $\delta_i^1 = Q_i^2$ for all i, $1 \leq i \leq n$, then $\delta_i^a = S_i^{a\prime}$ for all i, $1 \leq i \leq n$, because $[S_i]$ can not be rewritten with the rules of P_i^a. So we must have

$$(Q_1^2, \dots, Q_n^2,\ \alpha_1^2, \dots, \alpha_n^2,\ S_1^{a\prime}, \dots, S_n^{a\prime},\ S_1^{t\prime\prime}, \dots, S_n^{t\prime\prime}),$$

and then after a number of communication steps we get

$$(\beta_1, \dots, \beta_n,\ S_1^2, \dots, S_n^2,\ S_1^{a\prime}, \dots, S_n^{a\prime},\ S_1^{t\prime\prime}, \dots, S_n^{t\prime\prime}),$$

where the $\beta_i = \gamma_i$ if $(\gamma_1, \dots, \gamma_n)$ are the sentential forms of Γ produced by the initial rewriting step and the possibly following communication sequence and $\gamma_i \notin R_i$. If $\gamma_j \in R_j$ for some j, $1 \leq j \leq n$, then $\beta_j = S_j^1$. If no α_i^2 contains query symbols (there are no communication steps following the initial rewriting step in Γ), then $\beta_i = \alpha_i^2$. If β_1 is terminal then the system can stop here, if it is not, then the simulation must go on. We start with

$$(\alpha_1, \ldots, \alpha_n,\ S_1^2, \ldots, S_n^2,\ S_1^{a'}, \ldots, S_n^{a'},\ S_1^{t''}, \ldots, S_n^{t''}),$$

where $\alpha_i \in (N \cup T)^* \cup \{S_i^1\}$ and get

$$(\delta_1^1, \ldots, \delta_n^1,\ Q_1^1 \ldots, Q_n^1,\ [Y_1], \ldots, [Y_n],\ Q_1^a, \ldots, Q_n^a),$$

with $\delta_i^1 = [\alpha_i]$ where $[\alpha_i]$ is α_i with one of its nonterminals X rewritten to $[X]$ or if $\alpha_i \in T^*$ then $[\alpha_i] = \alpha_i$. If some communication occured in the previous step and the j-th sentential form was sent to an other component, then $\alpha_j = S_j^1$ and δ_j^1 is either Q_j^a or Q_j^2. If $\delta_j^1 = Q_j^2$ for some j, $1 \leq j \leq n$, then the system is blocked by a circular query, so if $\alpha_j = S_j^1$ for some j, then we must have

$$([\alpha_1^1], .., Q_j^a, .., [\alpha_n],\ Q_1^1 \ldots, Q_n^1,\ [Y_1], \ldots, [Y_n],\ Q_1^a, \ldots, Q_n^a),$$

with $Y_i \in R_i$, $1 \leq i \leq n$. After a communication step we get

$$(S_1^1, \ldots, S_n^1,\ [\alpha_1], .., [Y_j], .., [\alpha_n],\ S_1^a, \ldots, S_n^a,\ [Y_1], \ldots, [Y_n]).$$

Now the system continues the derivation as if G_j^1 has have returned to Y_j instead of its start symbol. We get

$$(\delta_1^1, \ldots, \delta_n^1,\ \beta_1^2, \ldots, \beta_n^2,\ \delta_1^a, \ldots, \delta_n^a,\ [Y_1]', \ldots, [Y_n]'),$$

where δ_i^1 and δ_i^a are the same as above and β_i^2 differ only in the indices of the query symbols from the strings produced by G_i of Γ, $1 \leq i \leq n$, as described above. The δ_i^1 are either Q_i^a or Q_i^2 and δ_i^a are either $S_i^{a'}$ or $[S_i]$, $1 \leq j \leq n$. If $\delta_j^1 = Q_j^a$ for some j, $1 \leq j \leq n$, then previously described situation arises, the system is going to be blocked after the next rewriting step, when G_j^t introduces Q_j^a, because P_j^t does not contain rules to rewrite $[S_j]$ or $S_j^{a'}$. If $\delta_i^1 = Q_i^2$ for all i, $1 \leq i \leq n$, then $\delta_i^a = S_i^{a'}$ for all i, $1 \leq i \leq n$ again, because $[S_i]$ can not be rewritten with the rules of P_i^a. So we must have

$$(Q_1^2, \ldots, Q_n^2,\ \beta_1^2, \ldots, \beta_n^2,\ S_1^{a'}, \ldots, S_n^{a'},\ [Y_1]', \ldots, [Y_n]').$$

After the communication sequence we get

$$(\gamma_1, \ldots, \gamma_n,\ S_1^2, \ldots, S_n^2,\ S_1^{a'}, \ldots, S_n^{a'},\ [Y_1]', \ldots, [Y_n]'),$$

where γ_i are the results of the communication sequence prescribed by the β_i sentential forms with $\gamma_j = S_j^1$ if the sentential form of G_j^1 has been sent to an other component during the communication sequence. If $\gamma_1 \in T^*$ the system stops or else it can continue to simulate Γ in the same manner. □

Let the systems satisfying the conditions of the lemma and the corresponding language classes be denoted by $\bar{r}X_nCF$ and $\mathcal{L}(\bar{r}X_nCF)$, $X \in \{PC, fPC\}$, respectively. Note that this proof is based on the fact that using immediate communications each component sends its string only once during a communication sequence, in other words the strings a component has returned to after a communication step are never communicated in the same communication sequence. Since homogeneous systems also have this property and since the simulating system constructed according to the previous theorem is homogeneous if we simulate a homogeneous system, we have the following:

Corollary 4.2 $\mathcal{L}(\overline{r}hPC_*CF) = \mathcal{L}(hPC_*CF)$.

Before we proceed, we need some further observations about the nature of derivations in PC grammar systems. In the proof of our main theorem we would like to treat the communication sequences of a derivation as "units" together. This means that we will assume that terminal words of the master appear only as a result of a rewriting step or as a result of a whole communication sequence, so we need to prove that all languages of PC grammar systems can also be generated this way, where the details of communication sequences are "hidden".

Definition 4.3 Let Γ be a PC grammar system. The language generated by Γ with *hidden communications* is

$$L_h(\Gamma) = \{\omega \in T^* \mid (S_1, S_2, \ldots, S_n) \Rightarrow^* (\omega, \alpha_2, \ldots, \alpha_n)\},$$

where $\mid \alpha_i \mid_K = 0$, $2 \leq i \leq n$ or $\alpha_2, \ldots, \alpha_n$ contain a circular query. In other words, the generated language consists of terminal strings present as sentential forms of the first component either after a rewriting step which does not introduce queries, or at the end of a communication sequence, or in a final blocking configuration.

Let the class of languages generated with hidden communications by X type PC grammar systems with n context-free components be denoted by $\mathcal{L}_h(X_nCF)$, $X \in \{fPC, \overline{r}fPC\}$.

Lemma 4.3 *If L is a language generated by a context-free PC grammar system $\Gamma \in X_nCF$, $X \in \{PC, fPC\}$, $L = L(\Gamma)$, then L can also be generated by a system with returning languages $\Gamma' \in \overline{r}X_{2n+2}CF$ with hidden communications, $L = L_h(\Gamma')$.*

Proof: Let $\Gamma = (N, K, T, G_1, \ldots, G_n)$ with N, K, T and G_i, $1 \leq i \leq n$ as usual and let $\Gamma' = (N', K', T, G_0, R_0, G_1^1, R_1^1 \ldots, G_n^1, R_n^1, G_1^2, R_1^2 \ldots, G_n^2, R_n^2, G_a, R_a)$, where G_0 is the master grammar and

$$\begin{aligned}
N' &= \{X, [X] \mid X \in N\} \cup \{A_0, S_0, S_0', S_a, S_a', S_a'', S_i^1, S_i^2 \mid 1 \leq i \leq n\}, \\
R_0 &= \{A_0\}, \\
P_0 &= \{X \to X \mid X \in N\} \cup \{S_0 \to S_0', S_0' \to Q_1^2\} \cup \{A_0 \to Q_1^2\}, \\
R_i^1 &= \{A_i^1\}, \\
P_i^1 &= \{S_i^1 \to [S_i], A_i^1 \to Q_i^2\} \cup \{X \to [X] \mid X \in N\}, \\
R_i^2 &= \{S_i^2\}, \\
P_i^2 &= \{S_i^2 \to Q_i^1\} \cup \{[X] \to \alpha \mid X \to \alpha \in P_i, \mid \alpha \mid_K = 0\} \cup \\
&\quad \{[X] \to \alpha_1 Q_{i_1}^1 \alpha_2 \ldots \alpha_t Q_{i_t}^1 \alpha_{t+1} \mid X \to \alpha_1 Q_{i_1} \alpha_2 \ldots \alpha_t Q_{i_t} \alpha_{t+1} \in P_i, \\
&\quad \alpha_j \in (N \cup T)^*, 1 \leq j \leq t+1\}, \\
R_a &= \{S_a\}, \\
P_a &= \{S_a \to S_a', S_a' \to S_a'', S_a'' \to Q_0 S_a'\},
\end{aligned}$$

for $1 \leq i \leq n$.

This Γ' system starts with the initial configuration

$$(S_0, S_1^1, \ldots, S_n^1, S_1^2, \ldots, S_n^2, S_a).$$

After a rewriting step we get

$$(S_0', [S_1], \ldots, [S_n], Q_1^1, \ldots, Q_n^1, S_a')$$

and after a communication

$$(S_0', A_1^1, \ldots, A_n^1, [S_1], \ldots, [S_n], S_a').$$

Now a rewriting step follows producing

$$(Q_1^2, Q_1^2, \ldots, Q_n^2, \alpha_1', \ldots, \alpha_n', S_a''),$$

where $\alpha_i' = \alpha_i$ if $S_i \rightarrow \alpha_i \in P_i$ and $\mid \alpha_i \mid_K = 0$ or if $\mid \alpha_i \mid_K \neq 0$, $\alpha_i = \alpha_{i_1} Q_{j_1} \alpha_{i_2} ... Q_{j_t} \alpha_{i_t}$, then $\alpha_i' = \alpha_{i_1} Q_{j_1}^1 \alpha_{i_2} ... Q_{j_t}^1 \alpha_{i_t}$. After the communication we have

$$(\delta_0, \beta_1, \ldots, \beta_n, S_1^2, \ldots, S_n^2, S_a''),$$

where β_i are the results of the communication sequence prescribed by $\alpha_1', \ldots, \alpha_n'$ with $\beta_j = S_j^1$ if the j-th component has returned to its axiom and δ_0 is either β_1 or if $\beta_1 = S_1^1$ then δ_0 is the string which was sent by G_1^1 during the communication before it has returned to its axiom. If δ_0 is terminal Γ' stops here, otherwise its work continues. After a rewriting step we get

$$(\delta_0, [\beta_1], \ldots, [\beta_n], Q_1^1, \ldots, Q_n^1, Q_0 S_a'),$$

where $[\beta_i]$ is β_i with one of its nonterminals X in brackets $[X]$ ($[\beta_i] = [S_i]$ if $\beta_i = S_i^1$) or if it does not contain any nonterminals then $[\beta_i] = \beta_i$ and δ_0 is the same as above. Now we get

$$(A_0, A_1^1, \ldots, A_n^1, [\beta_1], \ldots, [\beta_n], \delta_0 S_a')$$

and then

$$(Q_1^2, Q_1^2, \ldots, Q_n^2, \gamma_1', \ldots, \gamma_n', \delta_0 S_a''),$$

where $\gamma_i' = \gamma_i$ if $\beta_i \Rightarrow_{G_i} \gamma_i$ with one rewriting step and $\mid \gamma_i \mid_K = 0$, or if $\mid \gamma_i \mid_K \neq 0$, $\gamma_i = \gamma_{i_1} Q_{j_1} \gamma_{i_2} ... Q_{j_t} \gamma_{i_t}$, then $\gamma_i' = \gamma_{i_1} Q_{j_1}^1 \gamma_{i_2} ... Q_{j_t}^1 \gamma_{i_t}$. After the communication sequence we get

$$(\delta_0, \delta_1, \ldots, \delta_n, S_1^2, \ldots, S_n^2, \delta S_a''),$$

where δ_i, $1 \leq i \leq n$ are the results of the communication sequence prescribed by $\gamma_1', \ldots, \gamma_n'$ with $\delta_j = S_j^1$ if the j-th component has returned to its axiom and δ_0 is either δ_1 or if $\delta_1 = S_1^1$ then δ_0 is the string which was sent by G_1^1 during the communication before it has returned to its axiom. If δ_0 is terminal Γ' stops here, otherwise its work continues in the same manner. □

Now we need to define a notion we will use in the proof of the next lemma.

Definition 4.4 Let $\Gamma = (N, K, T, G_1, \ldots, G_n)$ a context-free PC grammar system with $K = \{Q_1, \ldots, Q_n\}$ and let α be a query, $\alpha = \alpha_1 Q_{i_1} \alpha_2 Q_{i_2} \ldots \alpha_t Q_{i_t} \alpha_{t+1}$; $|\alpha_k|_K = 0,\ 1 \le k \le t+1,\ 1 \le i_j \le n,\ 1 \le j \le t.$

We define *the j-th portion* $1 \le j \le t+1$ of this query in the following way: If $j \le t-1$ then the j-th portion is $\alpha_j Q_{i_j}$. Moreover, if $j = t$, then it is $\alpha_t Q_t \alpha_{t+1}$.

Now we are ready to prove the following:

Lemma 4.4 $\mathcal{L}_h(\overline{r}fPC_*CF) \subseteq \mathcal{L}(\overline{r}hPC_*CF)$

Proof: Let $\Gamma = (N,\ K,\ T,\ R_1, \ldots, R_n,\ G_1, \ldots, G_n) \in \overline{r}fPC_nCF$ be a PC grammar system with immediate communications, nonterminal alphabet N, set of query symbols K, terminal alphabet T, returning languages $R_1, \ldots, R_n$ and n context-free components $G_1, \ldots, G_n$.

Now we construct $\Gamma' \in \overline{r}hPC_mCF$, which generates the same language as Γ. Here $m = (t+2)n + 2u + 3$, where t and u are the following: t is the number of possible rule combinations that we can try to apply to the sentential forms of Γ, u is the sum of u^k, $1 \le k \le t$, where u^k is the sum of u_i^k, $1 \le i \le n$ and u_i^k is the number of query symbol occurrences on the right-hand side of the i-th rule of the k-th rule combination. Formally $t = |P_1| \prod_{i=2}^{n}(|P_i| + 1)$. If we denote the rules of the k-th rule combination with $X_1^k \to \alpha_1^k, \ldots, X_n^k \to \alpha_n^k$, then $u = \sum_{k=1}^{t} u^k$, $u^k = \sum_{i=1}^{n} u_i^k$, $u_i^k = |\alpha_i^k|_K$.

Γ' simulates the application of each rule combination of Γ in a different n-tuple of simulating components with the aid of assistants assigned to each of the simulating n-tuples. First an integer k, $1 \le k \le t$ is selected and the application of the k-th rule combination is simulated in the k-th n-tuple and in the k-th set of assistant components in p steps, with rules using only homogeneous queries. The integer p must be twice the number of necessary communication steps, which is at most $p = 2n - 2$. The simulating system contains the following components:

$$\begin{aligned}\Gamma' \; = \; (\; & N', K', T, R_1, \ldots\ldots, R_b, \\ & G_1, .., G_n, G_1^1, .., G_n^1, G_1^2, .., G_n^2, \ldots, G_1^t, .., G_n^t, \\ & G_{11}^1, .., G_{1u_1^1}^1, G_{21}^1, .., G_{2u_2^1}^1, \ldots, G_{n1}^t, .., G_{nu_n^t}^t, \\ & G_{a_1}, G_{a_2}, G_1{}', .., G_n', G_{11}^{1'}, .., G_{nu_n^t}^{t'}, G_b \quad)\end{aligned}$$

where the n-tuples simulating the k-th rule combination are denoted by G_i^k, $1 \le i \le n$ with their assistant components $G_{i\,1}^k, \ldots, G_{i\,u_i^k}^k$. G_{a_1} and G_{a_2} are involved in selecting the number of the rule combination to be simulated, $G_1', \ldots, G_n'$ are needed to help in sending back the sentential forms to $G_1, \ldots, G_n$ after the simulation of a rule combination, $G_{11}^{1'}, .., G_{nu_n^t}^{t'}$ are used to force a restart of the components $G_{11}^1, .., G_{nu_n^t}^t$ by querying them when necessary and G_b makes sure the system blocks if it simulates a rule combination which produces a circular query.

Let $C \subseteq \{1, \ldots, t\}$ be the set of those integers which number rule combinations that introduce circular queries and let the start symbol of the component $G_{\alpha\gamma}^{\beta}$ be

$A^{\beta}_{\alpha\gamma}$.

$$\begin{aligned}
N' &= \{Z, B, F\} \cup \{(l)^j, (l), (S_i)^j \mid 1 \le l \le t, 1 \le j \le p+2, 1 \le i \le n\} \cup \\
&\quad \{S_{ij}^{k\,\prime}, S_{ij}^{k\,\prime m} \mid 1 \le k \le t, 1 \le m \le p+2, 1 \le i \le n, 1 \le j \le u_i^k\} \cup \\
&\quad \{S^{\beta}_{\alpha\gamma}, A^{\beta}_{\alpha\gamma} \mid G^{\beta}_{\alpha\gamma} \textit{ is a component of } \Gamma'\} \cup \{X, [X] \mid X \in N\} \text{ and} \\
R^{\beta}_{\alpha\gamma} &= \{S^{\beta}_{\alpha\gamma}\}, \text{ where } G^{\beta}_{\alpha\gamma} \text{ is a component of } \Gamma', \\
P_i &= \{A_i \to S_i\} \cup \{X \to [X] \mid X \to \alpha \in P_i\} \cup \{S_{a_1} \to [X] \mid X \in R_i\} \cup \\
&\quad \{S_i \to (S_i)^1, (S_i)^j \to (S_i)^{j+1}, (S_i)^{p+1} \to Q_i' \mid 1 \le j \le p\}, \\
P_i^j &= L1_i^j \cup \{S_i^j \to Q_{a_1}, A_i^j \to Q_{a_1}, S_{a_1} \to S_{a_1}, (j) \to Q_i\} \cup \\
&\quad \{(k) \to (k)^1, (k)^l \to (k)^{l+1}, (k)^{p+2} \to Q_{a_1} \mid 1 \le k \le t, k \ne j, \\
&\qquad 1 \le l \le p+1\} \\
&\text{for all } 1 \le i \le n,\ 1 \le j \le t \text{ and} \\
P_{ij}^k &= L2_{ij}^k \cup \{S_{ij}^k \to Q_{a_1}, A_{ij}^k \to Q_{a_1}, S_{a_1} \to S_{a_1}, (k) \to (k)^1\} \cup \\
&\quad \{(l) \to (l) \mid 1 \le l \le t, l \ne k\} \\
&\text{for all } 1 \le k \le t,\ 1 \le i \le n,\ 1 \le j \le u_i^k, \\
P_{a_1} &= \{A_{a_1} \to (k), S_{a_1} \to (k), S_{a_1} \to S_{a_1} \mid 1 \le k \le t\}, \\
P_{a_2} &= \{A_{a_2} \to Q_{a_1}, S'_{a_2} \to Q_{a_1}, S_{a_2} \to S'_{a_2}, (k) \to (k) \mid 1 \le k \le t\}, \\
P_i' &= \{X \to X \mid X \in (N \cup \{S_{a_1}\})\} \cup \{(k) \to Q_i^k \mid 1 \le k \le t\} \cup \\
&\quad \{A_i' \to S_i', S_i' \to S_i^{1\prime}, S_i^{l\prime} \to S_i^{l+1\prime} S_i^{p\prime} \to Q_{a_2} \mid 1 \le l \le p-1\}, \\
&\text{for } 1 \le i \le n, \\
P_{ij}^{k\,\prime} &= \{A_{ij}^{k\prime} \to S_{ij}^{k\prime 1}, S_{ij}^{k\prime} \to S_{ij}^{k\prime 1}, S_{ij}^{k\prime m} \to S_{ij}^{k\prime m+1}, S_{ij}^{k\prime p+2} \to Q_{ij}^k \\
&\qquad \mid 1 \le m \le p+1\} \cup \\
&\quad \{(l) \to S_{ij}^{k\prime 1}, S_{a_1} \to S_{ij}^{k\prime 1} \mid l \ne k, 1 \le l \le t\} \\
&\text{for all } 1 \le k \le t,\ 1 \le i \le n,\ 1 \le j \le u_i^k \text{ and} \\
P_b &= \{A_b \to S_b, S_b \to S_b^1, S_b^l \to S_b^{l+1}, S_b^p \to Q_{a_2} \mid 1 \le l \le p-1\} \cup \\
&\quad \{(j) \to B, B \to F \mid j \in C\} \cup \\
&\quad \{(j) \to (j)^1, (j)^1 \to S_b \mid j \notin C\}.
\end{aligned}$$

We construct the sets $L1_i^k$ and $L2_{ij}^k$ $1 \le i \le n$, $1 \le j \le u_i^k$ in the following way: Let us fix a k and observe the n rules of the k-th rule combination.

The right sides of the rules determine the communication sequence that would follow after rewriting with our certain rule combination.

We say that a sentential form *contains a query* at a certain point of the communication sequence if it contains query symbols which are not yet replaced at that point of the communication sequence.

If our k-th rule combination produces a circular query, we modify the rules. We replace those query symbols which participate in the circle with a new nonterminal Z and execute the following algorithm on this modified rule combination. (See the example at the end of this section.)

For each m, $1 \leq m \leq p/2$, we repeat the following steps. (Note that $p/2$ is the maximal number of communication steps in Γ.) If the j-th rule of our rule combination is the empty rule, then $L1_j^k$ is empty since $u_j^k = 0$, no assistant components G_{ji}^k are present, so we do not need to construct $L2_{ji}^k$. During the following algorithm we consider the j-th sentential form only if the j-th rule of the combination is not empty.

1.a. If the i-th sentential form does not contain a query at the beginning of the m-th communication step and it is not communicated in the m-th communication step then we put the rule $[X_i] \to \alpha_i[X_i]^1$ in $L1_i^k$ where $X_i \to \alpha_i$ is the i-th rule of the k-th rule combination if $m = 1$ and the rule $[X_i]^{2m-2} \to [X_i]^{2m-1}$ for all other m.

1.b. If the i-th sentential form does not contain a query and it is communicated in the m-th step, then we put $[X_i] \to \alpha_i$ in L_i^k if $m = 1$ and $[X_i]^{2m-2} \to \epsilon$ for all other m.

2. If the i-th sentential form contains a query which is not yet satisfied at the beginning of the m-th communication step, then we put $[X_i] \to [X_i]^1$ in $L1_i^k$ if $m = 1$ and $[X_i]^{2m-2} \to [X_i]^{2m-1}$ for all other m.

2.a. If the j-th query symbol of this query is replaced in the m-th communication step then we put $(k)^{2m-1} \to \alpha Q_l^k \beta (k)^{2m}$ in $L2_{ij}^k$, where $\alpha Q_l \beta$ is the j-th portion of the right side of the i-th rule of the k-th combination.

2.b. If the j-th query symbol was or will be replaced in a step different from the m-th, then we put $(k)^{2m-1} \to (k)^{2m}$ in $L2_{ij}^k$.

3. There must be queries that are completely satisfied during the m-th communication step. If the i-th sentential form contains a query which is satisfied completely during the m-th communication step, we put $[X_i]^{2m-1} \to Q_{i1}^k$ in $L1_i^k$ and we put $(k)^{2m} \to Q_{i(j+1)}^k$ in $L2_{ij}^k$ for all $1 \leq j \leq u_i^k - 1$ and $(k)^{2m} \to [X_i]^{2m}$ in $L2_{iu_i^k}^k$.

4. For all i we did not deal with in point **3**, we put $[X_i]^{2m-1} \to [X_i]^{2m}$ in $L1_i^k$. If the i-th sentential form contains a query which is not yet satisfied completely during the m-th communication step, we put $(k)^{2m} \to (k)^{2m+1}$ in all $L2_{ij}^k$, $1 \leq j \leq u_i^k$.

After repeating these steps for all $1 \leq m \leq p/2$, finally add $[X_i]^p \to \epsilon$ to $L1_i^k$, $1 \leq i \leq n$.

Now we turn to the proof of our lemma. First we concentrate on the overall architecture of the simulating system and show how it works. We will see how it provides p steps for simulating each rule combination with the rules of the sets $L1_i^k$ and $L2_{ij}^k$, $1 \leq k \leq t$, $1 \leq i \leq n$, $1 \leq j \leq u_i^k$. Γ' starts with the initial configuration

$$\begin{aligned}(&A_1, .., A_n, A_1^1, .., A_n^1, ..., A_1^t, .., A_n^t,\\ &A_{11}^1, .., A_{1u_1^1}^1, ..., A_{n1}^t, .., A_{nu_n^t}^t,\\ &A_{a_1}, A_{a_2}, A_1', .., A_n', A_{11}^{1'}, ..., A_{nu_n^t}^{t'}, A_b).\end{aligned}$$

After one rewriting step we get

$$(S_1, .., S_n, Q_{a_1}, .., Q_{a_1}, ..., Q_{a_1}, .., Q_{a_1},$$

$$Q_{a_1},..,Q_{a_1},...,Q_{a_1},..,Q_{a_1},$$
$$(k),Q_{a_1},\ S'_1,..,S'_n,\ S^{1'^1}_{11},...,S^{t'^1}_{nu^t_n},S_b),$$

where the component G_{a_1} introduced the nonterminal (k) $1 \leq k \leq n$. This selection of k means that the system will try to apply the k-th rule combination. Now a communication follows

$$(S_1,..,S_n,\ (k),..,(k),...,(k),..,(k),$$
$$(k),..,(k),...,(k),..,(k),$$
$$S_{a_1},(k),\ S'_1,..,S'_n,\ S^{1'^1}_{11},...,S^{t'^1}_{nu^t_n},S_b),$$

where k, $1 \leq k \leq t$ is the number of the rule combination to be applied to the start symbols. Next we get

$$(\delta_1,..,\delta_n,\ (k)^1,..,(k)^1\ ,...,\ Q_1,..,Q_n\ ,...,\ (k)^1,..,(k)^1,$$
$$(k),..,(k)\ ,...,\ (k)^1,..,(k)^1\ ,...,\ (k),..,(k),$$
$$\delta_{a_1},(k),\ S^{1'}_1,..,S^{1'}_n,\ S^{1'^2}_{11},...,S^{t'^2}_{nu^t_n},S^1_b),$$

where δ_i is either $(S_i)^1$ or $[S_i]$, $1 \leq i \leq n$ and δ_{a_1} is either S_{a_1} or (l), $1 \leq l \leq t$. If δ_i is $(S_i)^1$ or δ_{a_1} is (l) then the system will get blocked, since G^k_i do not have rules with $(S_i)^1$ and G_{a_1} does not have rules with (l) on the left side. So we must have

$$([S_1],..,[S_n],\ (k)^1,..,(k)^1\ ,...,\ Q_1,..,Q_n\ ,...,\ (k)^1,..,(k)^1,$$
$$(k),..,(k)\ ,...,\ (k)^1,..,(k)^1\ ,...,\ (k),..,(k),$$
$$S_{a_1},(k),\ S^{1'}_1,..,S^{1'}_n,\ S^{1'^2}_{11},...,S^{t'^2}_{nu^t_n},S^1_b).$$

The assistant grammars $G^k_1,...,G^k_n$ for the k-th rule combination introduced $Q_1,...,Q_n$, they will receive the sentential forms of G_i, $1 \leq i \leq n$ and G_{a_2} preserves the value of k for later use. After the communication we have

$$(S_1,..,S_n,\ (k)^1,..,(k)^1\ ,...,\ [S_1],..,[S_n]\ ,...,\ (k)^1,..,(k)^1,$$
$$(k),..,(k)\ ,...,\ (k)^1,..,(k)^1\ ,...,\ (k),..,(k),$$
$$S_{a_1},(k),\ S^{1'}_1,..,S^{1'}_n,\ S^{1'^2}_{11},...,S^{t'^2}_{nu^t_n},S^1_b).$$

If the k-th rule combination is not applicable to the start symbols, then the rules of P^k_i are not applicable to $[S_i]$, $1 \leq i \leq n$. In this case the system is blocked, so let us assume that the k-th rule combination is applicable.

In the next rewriting step the system starts to simulate the effect of the k-th rule combination in p rewriting steps. We are going to show that if the k-th rule combination is applicable to the current sentential forms, then the system provides time for the simulation, takes the resulting sentential forms back to the first n-tuple and starts the process all over again with an other rule combination. The details of the simulation of the rule combinations will be discussed later, for now we denote the sentential forms of the active simulating components G^k_i and their assistants G^k_{im}, $1 \leq m \leq u^k_i$ by α^j_i and β^j_l, $1 \leq i \leq n$, $1 \leq l \leq u^k$, $1 \leq j \leq p$.

We are only interested in the effect the active simulating components and their assistants can have on the rest of the system and this is the following: After communication they return to their axioms and then introduce the query symbol Q_{a_1} querying the "outside world", the component G_{a_1}.

If they receive S_{a_1} then they use the rule $S_{a_1} \to S_{a_1}$ and at the end of the p steps this nonterminal will be sent back to G_i, $1 \leq i \leq n$ with the other simulation result, where it behaves exactly as the original start symbol. We show that the system is blocked if they receive an other symbol. After one rewriting step we get

$$\begin{aligned}&(\delta_1,..,\delta_n,\ (k)^2,..,(k)^2\ ,...,\ \alpha_1^1,..,\alpha_n^1\ ,...,\ (k)^2,..,(k)^2,\\ &(k),..,(k)\ ,...,\ \beta_1^1,..,\beta_{u^k}^1\ ,...,\ (k),..,(k),\\ &\delta_{a_1},(k),\ S_1^{2'},..,S_n^{2'},\ S_{11}^{1'^3},...,S_{nu_n^t}^{t'^3},S_b^2),\end{aligned}$$

where δ_i is either $[S_i]$ or $(S_i)^1$, $1 \leq i \leq n$ and δ_{a_1} is either S_{a_1} or (l), $1 \leq l \leq t$. If δ_i is $[S_i]$ or δ_{a_1} is (l), then the system is blocked since P_i and P_{a_1} does not contain rules with $[S_i]$ or (l) on the left side, respectively and no other component (not even the active simulating components G_i^k and their assistants G_{ij}^k, $1 \leq i \leq n$, $1 \leq j \leq u_i^k$) could introduce queries requesting one of these δ_i or δ_{a_1} sentential forms. So we continue from

$$\begin{aligned}&((S_1)^1,..,(S_n)^1,\ (k)^2,..,(k)^2\ ,...,\ \alpha_1^1,..,\alpha_n^1\ ,...,\ (k)^2,..,(k)^2,\\ &(k),..,(k)\ ,...,\ \beta_1^1,..,\beta_{u^k}^1\ ,...,\ (k),..,(k),\\ &S_{a_1},(k),\ S_1^{2'},..,S_n^{2'},\ S_{11}^{1'^3},...,S_{nu_n^t}^{t'^3},S_b^2),\end{aligned}$$

and get

$$\begin{aligned}&((S_1)^2,..,(S_n)^2,\ (k)^3,..,(k)^3\ ,...,\ \alpha_1^2,..,\alpha_n^2\ ,...,\ (k)^3,..,(k)^3,\\ &(k),..,(k)\ ,...,\ \beta_1^2,..,\beta_{u^k}^2\ ,...,\ (k),..,(k),\\ &\delta_{a_1},(k),\ S_1^{3'},..,S_n^{3'},\ S_{11}^{1'^4},...,S_{nu_n^t}^{t'^4},S_b^3).\end{aligned}$$

where δ_{a_1} is the same as above. We claim that rewriting steps follow in this manner providing the time for the simulation of the rule combination:

$$\begin{aligned}&((S_1)^2,..,(S_n)^2,\ (k)^3,..,(k)^3\ ,...,\ \alpha_1^2,..,\alpha_n^2\ ,...,\ (k)^3,..,(k)^3,\\ &(k),..,(k)\ ,...,\ \beta_1^2,..,\beta_{u^k}^2\ ,...,\ (k),..,(k),\\ &\delta_{a_1},(k),\ S_1^{3'},..,S_n^{3'},\ S_{11}^{1'^4},...,S_{nu_n^t}^{t'^4},S_b^3) \Rightarrow ... \Rightarrow\end{aligned}$$

$$\begin{aligned}&((S_1)^{p-1},..,(S_n)^{p-1},\ (k)^p,..,(k)^p\ ,...,\ \alpha_1^{p-1},..,\alpha_n^{p-1}\ ,...,\ (k)^p,..,(k)^p,\\ &(k),..,(k)\ ,...,\ \beta_1^{p-1},..,\beta_{u^k}^{p-1}\ ,...,\ (k),..,(k),\\ &\delta_{a_1},(k),\ S_1^{p'},..,S_n^{p'},\ S_{11}^{1'^{p+1}},...,S_{nu_n^t}^{t'^{p+1}},S_b^p).\end{aligned}$$

To verify our claim we show that the active simulating components and their assistants can not interfere with the work of the other components. To do this we have to observe their rule sets.

If one of the simulating components G_i^k, $1 \leq i \leq n$ returns to its axiom during this series of rewriting steps, it introduces Q_{a_1} and receives δ_{a_1} from G_{a_1}. If δ_{a_1} is S_{a_1} then it uses the rule $S_{a_1} \to S_{a_1}$. If δ_{a_1} is (l), $l \neq k$, then it uses its rules $(l) \to (l)^1$ and $(l)^i \to (l)^{i+1}$, $1 \leq i \leq p+1$. If δ_{a_1} is (k), then it introduces Q_i in the next rewriting step and receive $(S_i)^m$, $1 \leq m \leq p-2$ from G_i. In this case the system is blocked since the simulating components do not have rules with $(S_i)^m$ on the left side.

Now let us look at the assistants of the simulating components G_{ij}^{k}, $1 \leq i \leq n$, $1 \leq j \leq u_i^k$. If one of them returns to its axiom it also introduces Q_{a_1} and receives δ_{a_1} from G_{a_1}. If δ_{a_1} is S_{a_1} or (l), $l \neq k$ then the same things happen as we explained above. If δ_{a_1} is (k) then it is rewritten to $(k)^1$ and then rules of $L2_{ij}^k$ must be used. These rules can only query the active simulating components or their assistants, so they do not interfere with the rest of the system.

From these considerations we see that the system is either blocked, or it reaches the following configuration:

$$\begin{aligned}&((S_1)^{p-1}, .., (S_n)^{p-1},\ (k)^p, .., (k)^p\ , ...,\ \alpha_1^{p-1}, .., \alpha_n^{p-1}\ , ...,\ (k)^p, .., (k)^p,\\ &(k), .., (k)\ , ...,\ \beta_1^{p-1}, .., \beta_{u^k}^{p-1}\ , ...,\ (k), .., (k),\\ &\delta_{a_1}, (k),\ S_1^{p'}, .., S_n^{p'},\ S_{11}^{1'p+1}, ..., S_{nu_n^t}^{t'p+1}, S_b^p),\end{aligned}$$

where α_i^{p-1} and β_j^{p-1}, $1 \leq i \leq n, 1 \leq j \leq u^k$ can be sentential forms of components that either returned to their axioms or not. If they did not, then we assume the sentential forms to be correct, if they did, then α_i^{p-1} and β_i^{p-1} can be either S_{a_1}, (l), $(l)^m$ or Q_i, $1 \leq l \leq t$, $1 \leq i \leq n$, $1 \leq m \leq p-1$. If any of them is Q_i then a communication step follows and the system is blocked. In the other cases rewriting is possible, so we get

$$\begin{aligned}&((S_1)^p, .., (S_n)^p,\ (k)^{p+1}, .., (k)^{p+1}\ , ...,\ \alpha_1^p, .., \alpha_n^p\ , ...,\ (k)^{p+1}; .., (k)^{p+1},\\ &(k), .., (k)\ , ...,\ \beta_1^p, .., \beta_{u^k}^p\ , ...,\ (k), .., (k),\\ &\delta_{a_1}, (k),\ Q_{a_2}, .., Q_{a_2},\ S_{11}^{1'p+2}, ..., S_{nu_n^t}^{t'p+2}, Q_{a_2}),\end{aligned}$$

and then after a communication

$$\begin{aligned}&((S_1)^p, .., (S_n)^p,\ (k)^{p+1}, .., (k)^{p+1}\ , ...,\ \alpha_1^p, .., \alpha_n^p\ , ...,\ (k)^{p+1}, .., (k)^{p+1},\\ &(k), .., (k)\ , ...,\ \beta_1^p, .., \beta_{u^k}^p\ , ...,\ (k), .., (k),\\ &\delta_{a_1}, S_{a_2},\ (k), .., (k),\ S_{11}^{1'p+2}, ..., S_{nu_n^t}^{t'p+2}, (k)),\end{aligned}$$

where α_i^p and β_j^p are correct by our assumption (if their component grammars never returned to their axioms), or α_i^p and β_i^p can be either S_{a_1}, (l), $(l)^m$ or Q_i, $1 \leq i \leq n$, $1 \leq m \leq p$. If any of them is Q_i then after the replacement of this symbol the system is blocked. In the other cases rewriting is possible again, so we get

$$\begin{aligned}&((S_1)^{p+1}, .., (S_n)^{p+1},\ (k)^{p+2}, .., (k)^{p+2}\ , ...,\ \alpha_1^{p+1}, .., \alpha_n^{p+1}\ , ...,\ (k)^{p+2}, .., (k)^{p+2},\\ &(k), .., (k)\ , ...,\ \beta_1^{p+1}, .., \beta_{u^k}^{p+1}\ , ...,\ (k), .., (k),\\ &\delta_{a_1}, S'_{a_2},\ Q_1^k, .., Q_n^k,\ Q_{11}^1, ..., Q_{nu_n^t}^t, \delta_b),\end{aligned}$$

and then after communication

$$\begin{aligned}&((S_1)^{p+1}, .., (S_n)^{p+1},\ (k)^{p+2}, .., (k)^{p+2}\ , ...,\ S_1^k, .., S_n^k\ , ...,\ (k)^{p+2}, .., (k)^{p+2},\\ &S_{11}^1, ..., S_{nu_n^t}^t,\\ &\delta_{a_1}, S'_{a_2},\ \alpha_1^{p+1}, .., \alpha_n^{p+1},\ (k), ..., \beta_1^{p+1}, .., \beta_n^{p+1}, ..., (k), \delta_b),\end{aligned}$$

where δ_b is either B if the system should block after the simulation of the k-th rule combination or $(k)^1$ if it should not. Now if any of the α_i^{p+1}, $1 \leq i \leq n$ whose component grammar has returned to its axiom is not S_{a_1} or some β_i^{p+1} was not S_{a_1} before the communication, then the system is blocked. Otherwise we get

$$(Q'_1,..,Q'_n,\ Q_{a_1},...,Q_{a_1},$$
$$Q_{a_1},...,Q_{a_1},$$
$$\delta_{a_1}, Q_{a_1},\ \alpha_1^{p+1},..,\alpha_n^{p+1},\ S_{11}^{1'1},...,S_{nu_n^t}^{t'1}, \delta'_b),$$

and then

$$(\alpha_1^{p+1},..,\alpha_n^{p+1},\ \delta_{a_1},...,\delta_{a_1},$$
$$\delta_{a_1},...,\delta_{a_1},$$
$$S_{a_1}, \delta_{a_1},\ S'_1,..,S'_n,\ S_{11}^{1'1},...,S_{nu_n^t}^{t'1}, \delta'_b),$$

where δ_{a_1} is either S_{a_1} or (l), $1 \leq l \leq t$. If it is S_{a_1} then the system is blocked, since G_{a_2} does not have a rule with S_{a_1} on the left. So we have

$$(\alpha_1^{p+1},..,\alpha_n^{p+1},\ (l),...,(l),$$
$$(l),...,(l),$$
$$S_{a_1},(l),\ S'_1,..,S'_n,\ S_{11}^{1'1},...,S_{nu_n^t}^{t'1}, \delta'_b),$$

where α_i^{p+1}, $1 \leq i \leq n$ is the result of the k-th rule combination with S_{a_1} instead of S_i if the i-th component has returned to its axiom after a communication and δ'_b is either F or S_b. If α_1 is terminal the system stops here, if it is not, then it can continue in the same manner with the simulation of the l-th rule combination if δ'_b is not F. δ'_b is F only if the k-th rule combination introduces a circular query in Γ in which case Γ' should be blocked. If $\alpha_j^{p+1} = S_{a_1}$ for some j, then the j-th component should return to an element of R_j. This is simulated by using a rule $S_{a_1} \to [X]$, $X \in R_j$ in the next step.

Now we show how the p step simulation of the rule combinations is done. We have two cases. If the rule combination to be simulated does not introduce a query, then no assistant components are present. At the beginning of the simulation we get

$$(...,[S_1],..,[S_n],...) \Rightarrow (...,\alpha_1[S_1]^1,..,\alpha_n[S_n]^1,...),$$

in G_i^k using the rules of L_i^k, $1 \leq i \leq n$, where α_i are the right sides of the rules of the k-th rule combination. Now p rewriting step follows, we get

$$(...,\alpha_1[S_1]^1,..,\alpha_n[S_n]^1,...) \Rightarrow ... \Rightarrow (...,\alpha_1[S_1]^p,..,\alpha_n[S_n]^p,...),$$

and in the next step

$$(...,\alpha_1,..,\alpha_n,...)$$

using the rules $[S_i]^p \to \epsilon$, $1 \leq i \leq n$. Here α_i is the result of the application of the i-th rule of the simulated rule combination, the system deals with it as we previously described.

If the k-th rule combination introduces queries, the situation is more complicated. At the beginning the sentential forms of the simulating n-tuple and the assistants are

$$(...,[S_1],..,[S_n],......,(k)^1,..,(k)^1,...).$$

The sentential forms of the components G_i^k and G_{ij}^k after the l-th rewriting step will be denoted by ${\alpha_i}^l$ and ${\alpha_{ij}}^l$, $1 \leq i \leq n$, $1 \leq j \leq u_i^k$, $1 \leq l \leq p+1$.

After the first rewriting step, the sentential forms of the simulating n-tuple and their assistants are the following:

If the i-th sentential form in Γ is communicated in the first step then the sentential form of G_i^k, ${\alpha_i}^1$ is ω_i, the right side of the i-th rule of the rule combination:

$$(..., \alpha_1^1, .., \omega_{i_1}, .., \alpha_n^1, ..., (k)^2, .., (k)^2, ...).$$

If the i-th sentential form in Γ does not contain a query and it is not communicated in the first step then ${\alpha_i}^1$ is $\omega_i[S_i]^1$, ω_i is as above:

$$(..., \alpha_1^1, .., \omega_{i_1}, .., \omega_{i_2}[S_{i_2}]^1, .., \alpha_n^1, ..., (k)^2, .., (k)^2, ...).$$

In these two cases $u_i^k = 0$, so there are no corresponding assistant components.

If the i-th sentential form of Γ contains a query and the j-th query symbol of this query is replaced in the first step, then ${\alpha_i}^1 = [S_i]^1$, and the sentential form of the assistant component corresponding to this query symbol, α_{ij}^1 is $\alpha_1 Q_l^k \alpha_2 (k)^2$, where $\alpha_1 Q_l \alpha_2$ is the j-th portion of righthand side of the i-th rule:

$$(..., \alpha_1^1, .., \omega_{i_1}, .., \omega_{i_2}[S_{i_2}]^1, .., [S_{i_3}]^1, .., \alpha_n^1, ...$$
$$..., (k)^2, .., \alpha_1 Q_l^k \alpha_2 (k)^2, .., (k)^2, ...).$$

If the j-th query symbol of the i-th sentential form is not replaced in the first step, then ${\alpha_{ij}}^1 = (k)^2$:

$$(..., \alpha_1^1, .., \omega_{i_1}, .., \omega_{i_2}[S_{i_2}]^1, .., [S_{i_3}]^1, .., \alpha_n^1, ...$$
$$..., (k)^2; .., \alpha_1 Q_l^k \alpha_2 (k)^2, .., (k)^2, .., (k)^2, ...).$$

Now a communication follows in Γ'. If the l-th sentential form replaces the j-th query symbol of the i-th sentential form in the first communication step of Γ, then in Γ' ${\alpha_l}^1$ becomes S_l^k, ${\alpha_i}^1$ remains $[S_i]^1$ and ${\alpha_{ij}}^1$ becomes $\alpha_1 {\alpha_l}^1 \alpha_2 (k)^2$:

$$(..., \alpha_1^1, .., S_{i_1}, .., \omega_{i_2}[S_{i_2}]^1, .., [S_{i_3}]^1, .., \alpha_n^1, ...$$
$$..., (k)^2, .., \alpha_1 \omega_{i_1} \alpha_2 (k)^2, .., (k)^2, .., (k)^2, ...).$$

Now a rewriting step follows in Γ'. If the i-th sentential form was communicated in the first step then $\alpha_i^2 = Q_{a_1}$.

If the i-th sentential form was not communicated in the first step and it does not contain a query, then $\alpha_i^2 = \omega_i[S_i]^2$.

If the i-th sentential form contains a query but it is not completely satisfied in the first step, then $\alpha_i^2 = [S_i]^2$. If the j-th sentential form of this query was replaced in the first step then ${\alpha_{ij}}^2 = \alpha_1 {\alpha_l}^1 \alpha_2 (k)^3$:

$$(..., \alpha_1^2, ..Q_{a_1}, .., \omega_{i_2}[S_{i_2}]^2, .., [S_{i_3}]^2, .., \alpha_n^2, ...$$
$$..., (k)^3, .., \alpha_1 \omega_{i_1} \alpha_2 (k)^3, .., (k)^3, .., (k)^3, ...).$$

If the i-th sentential form contains a query which is completely satisfied in the first step, then $\alpha_i^2 = {Q_{i1}}^k$ and $\alpha_{ij}^2 = \omega_j Q_{i(j+1)}^k$, $1 \leq j \leq u_i^k - 1$ and $\alpha_{iu_i^k}^2 = \omega_{u_i^k}[S_i]^2$ where ω_l, $1 \leq l \leq u_i^k$ is the satisfied l-th portion of the righthand side of the query of the i-th rule. In this case a communication step follows in which the results of the query are collected and passed back from the assistants to G_i^k:

$$(\ldots,\alpha_1^2,..,Q_{a_1},..,\omega_{i_2}[S_{i_2}]^2,..,Q_{i_4}^k,..,\alpha_n^2,\ldots$$
$$\ldots,(k)^3,..,\omega_1 Q_{i_4 2},\omega_2[S_{i_4}]^2,..,(k)^3,..,(k)^3,\ldots),$$

$$(\ldots,\alpha_1^2,..,\delta_{i_1},..,\omega_{i_2}[S_{i_2}]^2,..,\omega_1\omega_2[S_{i_4}]^2,..,\alpha_n^2,\ldots$$
$$\ldots,(k)^3,..,S_{i_4 1}^k,S_{i_4 2}^k,..,(k)^3,..,(k)^3,\ldots),$$

where δ_{i_1} is either S_{a_1}, (l) with $l\neq k$ or (k).

Now the simulation of the first communication step of Γ is complete, the system begins to simulate the second one in the same manner. A rewriting step follows. If the i-th sentential form in Γ is communicated in the second step then $[S_i]^2$ is erased from the sentential form of G_i^k. If the i-th sentential form in Γ does not contain a query after the first communication step and it is not communicated in the second step then either $[S_i]^2$ is changed to $[S_i]^3$ or if the i-th component has returned to its axiom after the first communication step of Γ then there are three possibilities. If Q_{a_1} was replaced by S_{a_1}, then it is not changed. If Q_{a_1} was replaced by (l), $l\neq k$, then it is rewritten to $(l)^1$. If Q_{a_1} was replaced by (k), then it is rewritten to Q_i and after this communication no further rewriting will be possible:

$$(\ldots,\alpha_1^3,..,\delta_{i_1}^1,..,\omega_{i_2},..,[S_{i_3}]^3,..,\omega_1\omega_2[S_{i_4}]^3,..,\alpha_n^3,\ldots\ldots\ldots),$$

where $\delta_{i_1}^1$ is either S_{a_1}, (l) with $l\neq k$ or Q_{i_1}. If $\delta_{i_1}^1=Q_{i_1}$ then the system is blocked after the communication.

Now let us look at the assistant components. If $u_i^k\neq 0$ (the i-th sentential form contained a query which was completely satisfied in the first step), then G_{ij}^k, $1\leq j\leq u_i^k$, the assistant components of G_i^k have also returned to their axiom and now have Q_{a_1} as their sentential form. If Q_{a_1} is replaced by S_{a_1} or by (l), $l\neq k$, then it will not be changed later. If Q_{a_1} is replaced by (k), then it will be rewritten to $(k)^1$ and the assistant will begin to repeat what it previously had done. This will not interfere with the rest of the simulation process, since the i-th sentential form was already communicated.

If the i-th sentential form of Γ contains a query and the j-th query symbol of this query is replaced in the second step, then $\alpha_i{}^3=[S_i]^3$, and the sentential form of the assistant component corresponding to this query symbol, $\alpha_{ij}{}^3$ is $\alpha_1 Q_l^k\alpha_2(k)^4$, where $\alpha_1 Q_l\alpha_2$ is the j-th portion of righthand side of the i-th rule. If the j-th query symbol of the i-th sentential form is not replaced in the second step, then $\alpha_{ij}{}^3=(k)^4$

$$(\ldots,\alpha_1^3,..,\delta_{i_1}^1,..,\omega_{i_2},..,[S_{i_3}]^3,..,\omega_1\omega_2[S_{i_4}]^3,..,\alpha_n^3,\ldots$$
$$\ldots,(k)^4,..,Q_{a_1},Q_{a_1},..,\alpha_1 Q_{i_2}^k\alpha_2(k)^4,..,(k)^4,\ldots).$$

Now a communication follows in Γ'. If the l-th sentential form replaces the j-th query symbol of the i-th sentential form in the second communication step of Γ, then in Γ' $\alpha_l{}^3$ becomes S_l^k, $\alpha_i{}^3$ remains $[S_i]^3$ and $\alpha_{ij}{}^3$ becomes $\alpha_1\alpha_l{}^3\alpha_2(k)^4$.

$$(\ldots,\alpha_1^3,..,\delta_{i_1}^1,..,S_{i_2}^k,..,[S_{i_3}]^3,..,\omega_1\omega_2[S_{i_4}]^3,..,\alpha_n^3,\ldots$$
$$\ldots,(k)^4,..,\delta_{i_4 1},\delta_{i_4 2},..,\alpha_1\omega_{i_2}\alpha_2(k)^4,..,(k)^4,\ldots).$$

Now a rewriting step follows in Γ'. If the i-th sentential form was communicated in the second step then $\alpha_i^4 = Q_{a_1}$. If the i-th sentential form was not communicated in the second step and it does not contain a query, then $[S_i]^3$ is changed to $[S_i]^4$ in α_i^4.

If the i-th sentential form contains a query but it is not completely satisfied in the second step, then $\alpha_i^4 = [S_i]^4$. If the j-th sentential form of this query was replaced in the second step then ${\alpha_{ij}}^4 = \alpha_1 {\alpha_l}^3 \alpha_2 (k)^5$.

If the i-th sentential form contains a query which is completely satisfied in the first step, then $\alpha_i^4 = {Q_{i1}}^k$ and $\alpha_{ij}^4 = \omega_j Q_{i(j+1)}^k$, $1 \leq j \leq u_i^k - 1$ and ${\alpha_{iu_i^k}}^4 = \omega_{u_i^k}[S_i]^4$ where ω_l, $1 \leq l \leq u_i^k$ is the satisfied l-th portion of the righthand side of the i-th rule:

$$\begin{aligned}&(\ldots, \alpha_1^4, .., \delta_{i_1}^2, .., Q_{a_1}, .., Q_{i_3 1}^k, .., \omega_1\omega_2[S_{i_4}]^4, .., \alpha_n^4, \ldots\\ &\quad \ldots, (k)^5, .., \delta_{i_4 1}^1, \delta_{i_4 2}^1, .., \alpha_1\omega_{i_2}\alpha_2[S_{i_3}]^4, .., (k)^5, \ldots).\end{aligned}$$

In this case a communication step follows in which the results of the query are collected and passed back from the assistants to G_i^k:

$$\begin{aligned}&(\ldots, \alpha_1^4, .., \delta_{i_1}^2, .., \delta_{i_2}, .., \alpha_1\omega_{i_2}\alpha_2[S_{i_3}]^4, .., \omega_1\omega_2[S_{i_4}]^4, ..\alpha_n^4, \ldots\\ &\quad \ldots, (k)^5, .., \delta_{i_4 1}^1, \delta_{i_4 2}^1, .., S_{i_2 1}^k, .., (k)^5, \ldots).\end{aligned}$$

Now the simulation of the second communication step of Γ is complete, the system begins to simulate the third one in the same manner, and so on.

If the simulation of all communication step is complete, then the system uses the rules $[S_i]^m \Rightarrow [S_i]^{m+1}$, $1 \leq i \leq n$, $1 \leq m \leq p-1$, and finally when G_i' get ready to receive the result, it erases $[S_i]^p$, $1 \leq i \leq n$.

It is clear that all our arguments about the simulation of the first rewriting step and the following communication sequence of Γ by Γ' also hold for all other rewriting steps and communication sequences, where all of the sentential forms contain at least one non-terminal.

Now let us consider the case when one or more of the sentential forms $\alpha_1, \ldots, \alpha_n$ of $G_1, \ldots, G_n$ is terminal and the system chose to simulate the application of a rule combination to these sentential forms.

If α_j is terminal for some j and the j-th rule of the chosen combination is empty, the simulation is correct. Now we show that the simulation is also correct, if α_j is terminal and the j-th rule of the chosen combination is not empty, but it is $X_j \to \omega_j$.

If $|\omega_j|_K = 0$, the j-th rule does not introduce queries, then the simulation would consist of rewriting $[X_j]$ to $\omega_j[X_j]^1$, $\omega_j[X_j]^2$ and so on, until the bracketed nonterminal $[X_j]^l$ is finally erased. Using these rules on $\alpha_j \in T^*$ has the same effect as if the chosen combination contained the empty rule instead of $X_j \to \omega_j$.

If $|\omega_j|_K \neq 0$, the j-th rule introduces queries, then the assistant components $G_{j1}^k, .., G_{ju_j^k}^k$ begin to collect the result of the query. The system will get blocked when they are ready to send the result to G_j^k, because G_{j1}^k can not rewrite the bracketed nonterminals $[X_j]^l$, $1 \leq l \leq p$. □

We demonstrate this construction on a simple example.

Example Consider the following PC grammar system $\Gamma \in \overline{r}fPC_4CF$ generating the language $\{aa\}$.

$\Gamma = (N, K, T, R_1, ..., R_4, G_1, ..., G_4)$, $N = \{S_i \mid 1 \leq i \leq 4\}$, $T = \{a, b\}$,

$P_1 = \{S_1 \to Q_3Q_2\}$, $P_2 = \{S_2 \to Q_3\}$, $P_3 = \{S_3 \to a\}$, $P_4 = \{S_4 \to b\}$.

Since we have only one rule in each rule set, our rule combinations contain the rule of P_1 and we are free to choose the empty rule instead of one or more rules of the other components. This gives us a total number of 8 combinations, of which only that one is applicable which contains the four rules of the four components. Let this one be the 8-th one and let us concentrate only on this combination.

Now $t = 8$, $u = 20$, $u_1^8 = 2$, $u_2^8 = 1$, $u_3^8 = 0$, $u_4^8 = 0$, the simulating system $\Gamma' \in \overline{r}hPC_{83}CF$ contains the following components:

$$\begin{aligned}\Gamma' = (\ & N', K', T, R_1, .., R_b,\\ & G_1, .., G_4,\ G_1^1, .., G_4^1, ..., G_1^8, .., G_4^8,\\ & G_{11}^1, \dots, G_{21}^7,\ G_{11}^8, G_{12}^8, G_{21}^8,\\ & G_{a_1}, G_{a_2},\ G_1', .., G_n',\ G_{11}^{1'}, ..., G_{21}^{8'},\ G_b\quad).\end{aligned}$$

The longest communication sequence of the original system contains 2 communications steps so the choise of $p = 4$ is appropriate. The rest of the system Γ' is:

$$\begin{aligned}N' &= \{S_i,\ [S_i] \mid 1 \leq i \leq 4\} \cup \{S_{\alpha\gamma}^{\beta}, A_{\alpha\gamma}^{\beta} \mid G_{\alpha\gamma}^{\beta} is\ a\ component\ of\ \Gamma'\} \cup\\ &\quad \{S_{ij}^{k\ \prime}, S_{ij}^{k\ \prime m} \mid 1 \leq k \leq 8,\ 1 \leq i \leq 4,\ 1 \leq j \leq u_i^k,\ 1 \leq m \leq 6\} \cup\\ &\quad \{(l), (l)^j, (S_i)^j \mid 1 \leq l \leq 8,\ 1 \leq j \leq 6,\ 1 \leq i \leq 4\} \cup\\ &\quad \{Z, B, F\},\\ R_{\alpha\gamma}^{\beta} &= \{S_{\alpha\gamma}^{\beta}\},\ \mathrm{G}_{\alpha\gamma}^{\beta} \text{ is a component of } \Gamma',\\ P_i &= \{A_i \to S_i\} \cup \{S_i \to [S_i]\} \cup \{S_{a_1} \to [S_i]\} \cup\\ &\quad \{S_i \to S_i^1, S_i^1 \to S_i^2, S_i^2 \to S_i^3, S_i^3 \to S_i^4; S_i^4 \to S_i^5, S_i^5 \to Q_i'\},\\ P_i^j &= L1_i^j \cup \{S_i^j \to Q_{a_1}, A_i^j \to Q_{a_1}, (j) \to Q_i; S_{a_1} \to S_{a_1}\} \cup\\ &\quad \{(k) \to (k)^1, (k)^1 \to (k)^2, (k)^2 \to (k)^3, (k)^3 \to (k)^4, (k)^4 \to (k)^5,\\ &\quad\ (k)^5 \to (k)^6, (k)^6 \to Q_{a_1} \mid 1 \leq k \leq 8, k \neq j\},\end{aligned}$$

for all $1 \leq i \leq 4,\ 1 \leq j \leq 8$ and

$$\begin{aligned}P_{ij}^k &= L2_{ij}^k \cup \{S_{ij}^k \to Q_{a_1}, A_{ij}^k \to Q_{a_1}, (k) \to (k)^1, S_{a_1} \to S_{a_1}\} \cup\\ &\quad \{(l) \to (l) \mid 1 \leq l \leq 8, l \neq k\},\end{aligned}$$

for all $1 \leq k \leq 8,\ 1 \leq i \leq 4,\ 1 \leq j \leq u_i^k$.

$$\begin{aligned}P_{a_1} &= \{A_{a_1} \to (k), S_{a_1} \to (k), S_{a_1} \to S_{a_1} \mid 1 \leq k \leq 8\},\\ P_{a_2} &= \{A_{a_2} \to Q_{a_1}, S_{a_2}' \to Q_{a_1}, S_{a_2} \to S_{a_2}', (k) \to (k) \mid 1 \leq k \leq 8\},\\ P_i' &= \{S_i \to S_i, S_{a_1} \to S_{a_1}\} \cup \{A_i' \to S_i', (k) \to Q_i^k \mid 1 \leq k \leq 8\} \cup\end{aligned}$$

$$
\begin{aligned}
& \{S_i' \to S_i^{1'}, S_i^{1'} \to S_i^{2'}, S_i^{2'} \to S_i^{3'}, S_i^{3'} \to S_i^{4'}, S_i^{4'} \to Q_{a_2}\}, \\
& \text{for } 1 \leq i \leq 4 \text{ and} \\
P_{ij}^{k'} \;=\; & \{A_{ij}^{k'} \to S_{ij}^{k'1}, S_{ij}^{k'} \to S_{ij}^{k'1}, S_{ij}^{k'm} \to S_{ij}^{k'm+1}, S_{ij}^{k'6} \to Q_{ij}^{k} \\
& \mid 1 \leq m \leq 5\} \cup \\
& \{(l) \to S_{ij}^{k'1}, S_{a_1} \to S_{ij}^{k'1} \mid l \neq k, 1 \leq l \leq 8\}, \\
& \text{for all } 1 \leq k \leq 8,\ 1 \leq i \leq 4,\ 1 \leq j \leq u_i^k, \\
P_b \;=\; & \{A_b \to S_b, S_b \to S_b^1, S_b^1 \to S_b^2, S_b^2 \to S_b^3, S_b^3 \to S_b^4, S_b^4 \to Q_{a_2}\} \cup \\
& \{(j) \to (j)^1, (j)^1 \to S_b \mid 1 \leq j \leq 8\}.
\end{aligned}
$$

Now if we construct the sets $L1_i^8$ and $L2_{ij}^8$ according to the algorithm given above, we get the following result:

$$
\begin{aligned}
& L_1^8 = \{[S_1] \to [S_1]^1, [S_1]^1 \to [S_1]^2, [S_1]^2 \to [S_1]^3, [S_1]^3 \to Q_{11}^8, [S_1]^4 \to \epsilon\}, \\
& L_2^8 = \{[S_2] \to [S_2]^1, [S_2]^1 \to Q_{21}^8, [S_2]^2 \to \epsilon\}, \\
& L_3^8 = \{[S_3] \to a, [S_3]^1 \to [S_3]^2, [S_3]^2 \to [S_3]^3, [S_3]^3 \to [S_3]^4, [S_3]^4 \to \epsilon\}, \\
& L_4^8 = \{[S_4] \to b[S_4]^1, [S_4]^1 \to [S_4]^2, [S_4]^2 \to [S_4]^3, [S_4]^3 \to [S_4]^4, [S_4]^4 \to \epsilon\}, \\
& L2_{11}^8 = \{(8)^1 \to Q_3^8(8)^2, (8)^2 \to (8)^3, (8)^3 \to (8)^4, (8)^4 \to Q_{12}^8\}, \\
& L2_{12}^8 = \{(8)^1 \to (8)^2, (8)^2 \to (8)^3(8)^3 \to Q_2^8(8)^4, (8)^4 \to [S_1]^4\}, \\
& L2_{21}^8 = \{(8)^1 \to Q_3^8(8)^2, (8)^2 \to [S_2]^2\}.
\end{aligned}
$$

□

By corollary 4.2, lemma 4.3 , lemma 4.4 and by observation 2 we have the following theorem:

Theorem 4.5 $\mathcal{L}(fPC_*CF) = \mathcal{L}(hPC_*CF)$

Proof: The inclusion $\mathcal{L}(hPC_*CF) \subseteq \mathcal{L}(fPC_*CF)$ holds by observation 2. To show the converse inclusion, we have $\mathcal{L}(fPC_*CF) \subseteq \mathcal{L}_h(\overline{r}fPC_*CF)$ by lemma 4.3, $\mathcal{L}_h(\overline{r}fPC_*CF) \subseteq \mathcal{L}(\overline{r}hPC_*CF)$ by lemma 4.4 and $\mathcal{L}(\overline{r}hPC_*CF) = \mathcal{L}(hPC_*CF)$ by corollary 4.2. □

5 Conclusion

In this paper we have introduced immediate communication in parallel communicating grammar systems. Since it differs only slightly from previously existing communications, the generative power of these systems do not change in most cases. To study the generative power of non-centralized, returning systems, we generalized the idea of "returning to the axiom after communication" and we have shown that the use of immediate communications in non-centralized returning PC grammar systems results in the same generative power as if we only used homogeneous queries with the usual communication protocol.

References

[1] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation,* Gordon and Breach, London, 1994.

[2] J. Kari, L. Santean, The impact of the number of cooperating grammars on the generative power, *Theor. Computer Sci.* 98 (1992), 249-263.

[3] V. Mihalache, On parallel communicating grammar systems with context-free components, in *Mathematical Linguistics and Related Topics* (Gh. Păun, ed.), The Publ. House of the Romanian Academy, Bucharest, 1995, 258-270.

[4] Gh. Păun, Parallel communicating grammar systems; the context-free case, *Found. Control Engineering,* 14, 1, (1989), 39-50.

[5] Gh. Păun, A. Salomaa, S. Vicolov, On the generative capacity of parallel communicating grammar systems, *Intern. J. Computer Math.* 45 (1992), 49-59.

[6] Gh. Păun, L. Santean, Parallel communicating grammar systems: the regular case, *Ann. Univ. Bucharest, Ser. Matem.-Inform.* 38, 2 (1989), 55-63.

[7] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

Acta Cybernetica **13** (1997) 197-207.

On Two-Step Methods for Stochastic Differential Equations

Rózsa Horváth Bokor [*][†]

Abstract

The paper introduces a new two-step method. Its order of strong convergence is proved. In the approximation of solutions of some stochastic differential equations, this multistep method converges faster in mean $E|X - Y_N|$ than the One-step Milstein scheme with order 1.0 or Two-step Milstein scheme with order 1.0.

Keywords: Stochastic differential equations, strong solutions, numerical schemes

1 Introduction

The problem considered in this article is that of approximating strong solutions of the following type of the Itô stochastic differential equation:

$$dX_t \quad = \quad a(t, X_t)dt + b(t, X_t)dW_t, \text{ for } 0 \le t \le T, \ X_t \in \mathbf{R}^d, \tag{1}$$

where

$$a = (a_1 \dots a_d)^\tau, b = (b_1 \dots b_d)^\tau, X_0 = X (\in \mathbf{R}^d).$$

The above system is driven by the one-dimensional Brownian motion. Details about this stochastic object and corresponding calculus can be found in Karatzas and Shreve [2].

We suppose that throughout this paper $E \| X_0 \|^2 < +\infty$ and X_0 is independent of $\mathcal{F}_t = \sigma\{W_s : 0 \le s \le t\}$, the σ-algebra generated by the underlying process. Also, suppose that coefficients $a(t,x)$ and $b(t,x)$ satisfy conditions which guarantee the existence of the unique, strong solution of the stochastic differential equation.

The approximations considered here are evaluated at points of regular partition of the interval $[0,T]$; these have the form $(0, \Delta, 2\Delta, \dots, N\Delta)$, where N is a natural number and $\Delta = \frac{T}{N}$. We denote $n\Delta$ by τ_n, for $n = 0, 1, \dots, N$.

[*]Department of Mathematics University of Zagreb, Bijemicka 30, 10000 Zagreb, Croatia, email: bokor@math.hr

[†]Work supported by the Central Research Found of the Hungarian Academy of Sciences (Grant No. T014548)

Here we shall use the abbreviation Y_n to denote the value of the approximation at time $n\Delta$ and the following operators

$$L^0 = \frac{\partial}{\partial t} + \sum_{k=1}^{d} a_k \frac{\partial}{\partial x_k} + \frac{1}{2} \sum_{k,l=1}^{d} b^k b^l \frac{\partial^2}{\partial x_k \partial x_l}, \tag{2}$$

$$L^1 = \sum_{k=1}^{d} b^k \frac{\partial}{\partial x_k}. \tag{3}$$

To classify different methods with respect to the rate of strong convergence as in [3] we say that a discrete time approximation Y^Δ converges with strong order $\gamma > 0$ if there exist constants $\Delta_0 \in (0, +\infty)$ and $K < +\infty$, not depending on Δ, such that we have a mean global error

$$Eps(T) = E\left|X_T - Y_N^\Delta\right| \leq K\Delta^\gamma \text{ for all } \Delta \subset (0, \Delta_0).$$

The widely used method of order 1.0 is the Milstein method, which has the form

$$Y_{n+1}^M = Y_n^M + a(\tau_n, Y_n^M)\Delta + b(\tau_n, Y_n^M)\Delta W_n + \frac{1}{2}L^1 b(\tau_n; Y_n^M)((\Delta W_n)^2 - \Delta), \tag{4}$$

with $Y_0^M = X_0$. The two-step Milstein strong scheme, for which the k-th component in the general multidimensional case $d = 1, 2, \ldots$ is given by

$$\begin{aligned} Y_{n+1}^{k,T} &= (1-\gamma_k)Y_n^{k,T} + \gamma_k Y_{n-1}^{k,T} + a^k(\tau_n, Y_n^T)\Delta + V_n^k \\ &+ \gamma_k \left[\left((1-\alpha_k)a^k(\tau_n, Y_n^T) + \alpha_k a^k(\tau_{n-1}, Y_{n-1}^T) \right)\Delta + V_{n-1}^k \right], \end{aligned} \tag{5}$$

with

$$\begin{aligned} V_n^k &= b^k(\tau_n, Y_n^T)\Delta W_n + \frac{1}{2}L^1 b^k(\tau_n, Y_n^T)((\Delta W_n)^2 - \Delta), \\ Y_0^T &= X_0,\ Y_1^T = Y_1^M, \end{aligned}$$

where $\Delta W_n = W_{\tau_{n+1}} - W_{\tau_n}$, $n = 0, 1, \ldots, N-1, k = 1, \ldots, d$, and $\alpha_k, \gamma_k \in [0, 1]$.

In the general multidimensional case with $d = 1, 2, \ldots$ the k-th component of the new multistep scheme takes the form

$$\begin{aligned} Y_{n+1}^k &= (1-\gamma_k)Y_n^k + \gamma_k Y_{n-1}^k + a^k(\tau_n, Y_n)\Delta + b^k(\tau_n, Y_n)\Delta W_n \\ &+ \frac{1}{2}L^1 b^k(\tau_n, Y_n)((\Delta W_n)^2 - \Delta) \\ &+ \gamma_k \left[\left((1-\alpha_k)a^k(\tau_n, Y_n) + \alpha_k a^k(\tau_{n-1}, Y_{n-1}) \right)\Delta \right. \\ &+ \frac{1}{2}\left(b^k(\tau_n, Y_n) + b^k(\tau_{n-1}, Y_{n-1}) \right)\Delta W_{n-1}, \end{aligned}$$

$$
\begin{aligned}
&- \left. \frac{1}{2} L^1 b^k(\tau_{n-1}, Y_{n-1})\Delta \right], \qquad (6)\\
Y_0 &= X_0,\ Y_1 = Y_1^M,
\end{aligned}
$$

where $\Delta W_n = W_{\tau_{n+1}} - W_{\tau_n}$, $\Delta = \tau_{n+1} - \tau_n$, $n = 0, 1, \ldots, N-1, k = 1, \ldots, d$ and $\alpha_k, \gamma_k \in [0,1]$.

During the last years several authors have proposed multistep methods for stochastic differential equations with respect to strong convergence criterious.

I refer here to the books of Kloeden and Platen [3], Boulean and Lépingle [1] and the paper of Lépingle and Ribémont [4].

2 The Main Results

Now we are able to state the corresponding convergence theorem for the multistep method (6):

Theorem 2.1 *Consider the Itô equation (1). Let*

$$
\frac{\partial a}{\partial t}, \frac{\partial a}{\partial x_i}, \frac{\partial^2 a}{\partial x_i \partial x_j}, \frac{\partial b}{\partial t}, \frac{\partial b}{\partial x_i}, \frac{\partial^2 b}{\partial t^2}, \frac{\partial^2 b}{\partial t \partial x_i},
$$

$$
\frac{\partial^2 b}{\partial x_i \partial t}, \frac{\partial^2 b}{\partial x_i \partial x_j}, \frac{\partial^3 b}{\partial x_i \partial x_j \partial x_k} \in C_b([0,T] \times \mathbf{R}^d, \mathbf{R}^d),
$$

be given for all $1 \leq i,j,k \leq d$, where $C_b([0,T] \times \mathbf{R}^d, \mathbf{R}^d)$ denotes the set of continuous and bounded functions from $[0,T] \times \mathbf{R}^d$ to $\mathbf{R}^d$, and functions $L^0 a, L^0 b, L^1 a, L^0 L^1 b, L^1 L^1 b$ fulfill the linear growth condition

$$
\| f(t,x) \| \leq K_1(1 + \| x \|),
$$

for every $t \in [0,T], x \in \mathbf{R}^d$, where K_1 is a positive constant. Under the assumptions the multistep method converges with strong order $\gamma = 1.0$, that is for all $n = 0, 1, \ldots, N$ and step size $\Delta = \frac{T}{N}$, $N = 2, 3 \ldots$

$$
E(\| X_{\tau_n} - Y_n \|) \leq K_2(1 + E \| X_0 \|)\Delta^{1.0},
$$

where K_2 does not depend on Δ.

Remarks 2.2 **(1)** *In computation, the boundedness assumption is no restriction since any number generated by the computer is bounded by the capacity of the computer.*

(2) $\| \cdot \|$ *is a norm in $\mathbf{R}^d$.*

(3) *We would prove the statement of the theorem for the scheme (6), where $\alpha_k = 0.0$. For $\alpha_k \in (0,1]$ we prove the statement of the theorem on the same way. For $\alpha_k = 0.0$ the scheme (5) equals (4) if $Y_0^T = Y_0^M$ and $Y_1^T = Y_1^M$.*

To prove Theorem (2.1), we recall the following lemmas:

Lemma 2.3 *For all natural number $N = 1, 2, \ldots$ and for all $k = 0, 1, \ldots, N$ are valid the next inequalities*

$$E(\| Y_k^M \|^2) \leq K_3(1 + E\|X_0\|^2),$$

$$E(\| Y_k^T \|^2) \leq K_3(1 + E\|X_0\|^2).$$

Lemma 2.4 *Under the assumptions of Theorem 2.1 the Milstein approximation Y_n^M converges with strong order 1.0 that is*

$$E \| X_T - Y_N^M \|^2 \leq K_5 \Delta^{2.0}(1 + E \| X_0 \|^2) + K_6 E \| X_0 - Y_0^M \|^2$$

where the constants K_5, K_6 do not depend on Δ.

Proof

Since the first-order partial derivatives of a and b are bounded, there exists a $K_7 < +\infty$ such that for all $x, y \in \mathbf{R}^d$, (see details in Newton [5])

$$\begin{aligned} \| a(t,x) - a(t,y) \| &\leq K_7 \| x - y \|, \\ \| b(t,x) - b(t,y) \| &\leq K_7 \| x - y \|, \\ \| L^1 b(t,x) - L^1 b(t,y) \| &\leq K_7 \| x - y \|, \\ \| a(t,x) \| + \| b(t,x) \| + \| L^1 b(t,x) \| &\leq K_7(1 + \| x \|). \end{aligned}$$

We introduce the Milstein approximation (4) in the form

$$\begin{aligned} Y_{n+1}^{k,M} &= (1-\gamma_k)Y_n^{k,M} + a^k(\tau_n, Y_n^M)\Delta + b^k(\tau_n, Y_n^M)\Delta W_n \\ &+ \frac{1}{2}L^1 b^k(\tau_n, Y_n)((\Delta W_n)^2 - \Delta) + \gamma_k Y_n^{k,M} \\ &= (1-\gamma_k)Y_n^{k,M} + a^k(\tau_n, Y_n^M)\Delta + b^k(\tau_n, Y_n^M)\Delta W_n \\ &+ \frac{1}{2}L^1 b^k(\tau_n, Y_n^M)((\Delta W_n)^2 - \Delta) + \gamma_k \Bigg(Y_{n-1}^{k,M} + a^k(\tau_{n-1}, Y_{n-1}^M)\Delta \\ &+ b^k(\tau_{n-1}, Y_{n-1}^M)\Delta W_{n-1} + \frac{1}{2}L^1 b^k(\tau_{n-1}, Y_{n-1}^M)((\Delta W_{n-1})^2 - \Delta) \Bigg). \end{aligned}$$

Taylor's expansion is used to give the term $b^k(\tau_{n-1}, Y_{n-1}^M)$ around (τ_n, Y_n^M) and

$$\begin{aligned} b^k(\tau_{n-1}, Y_{n-1}^M) &= b^k(\tau_n, Y_n^M) + \frac{\partial}{\partial t} b^k(\tau_n, Y_n^M)(\tau_{n-1} - \tau_n) \\ &+ \sum_{i=1}^{d} \frac{\partial b^k}{\partial x_i}(\tau_n, Y_n^M)(Y_{n-1}^{i,M} - Y_n^{i,M}) \end{aligned}$$

$$
\begin{aligned}
&+ \frac{1}{2}\frac{\partial^2 b^k}{\partial t^2}(\tau_n^*, Y_n^{*,M})(\tau_{n-1} - \tau_n)^2 \\
&+ \sum_{i=1}^{d} \frac{\partial^2 b^k}{\partial t \partial x_i}(\tau_n^*, Y_n^{*,M})(\tau_{n-1} - \tau_n)(Y_{n-1}^{i,M} - Y_n^{i,M}) \\
&+ \frac{1}{2}\sum_{i,j=1}^{d} \frac{\partial^2 b^k}{\partial x_i \partial x_j}(\tau_n^*, Y_n^{*,M})(Y_{n-1}^{i,M} - Y_n^{i,M})(Y_{n-1}^{j,M} - Y_n^{j,M}),
\end{aligned}
$$

and

$$
\begin{aligned}
\frac{\partial b^k}{\partial x_i}(\tau_n, Y_n^M) &= \frac{\partial b^k}{\partial x_i}(\tau_{n-1}, Y_{n-1}^M) + \frac{\partial^2 b^k}{\partial t \partial x_i}(\tau_{n-1}^{*,*}, Y_{n-1}^{*,*,M})(\tau_n - \tau_{n-1}) \\
&+ \sum_{j=1}^{d} \frac{\partial^2 b^k}{\partial x_j \partial x_i}(\tau_{n-1}^{*,*}, Y_{n-1}^{*,*,M})(Y_{n-1}^{j,M} - Y_n^{j,M}).
\end{aligned}
$$

Also; used the fact that

$$
\begin{aligned}
Y_{n-1}^{j,M} - Y_n^{j,M} &= -a^j(\tau_{n-1}, Y_{n-1}^M)\Delta - b^j(\tau_{n-1}, Y_{n-1}^M)\Delta W_{n-1} \\
&- \frac{1}{2}L^1 b^j(\tau_{n-1}, Y_{n-1}^M)((\Delta W_{n-1})^2 - \Delta).
\end{aligned}
$$

When these are substituted into the expression $Y_{n+1}^{k,M}$ and assumptions of the theorem are used we get

$$
\begin{aligned}
Y_{n+1}^k &- Y_{n+1}^{k,M} = (1-\gamma_k)(Y_n^k - Y_n^{k,M}) + (a^k(\tau_n, Y_n) - a^k(\tau_n, Y_n^M))\Delta \\
&+ (b^k(\tau_n, Y_n) - b^k(\tau_n, Y_n^M))\Delta W_n \\
&+ \frac{1}{2}\left(L^1 b^k(\tau_n, Y_n) - L^1 b^k(\tau_n, Y_n^M)\right)((\Delta W_n)^2 - \Delta) \\
&+ \gamma_k\left(Y_{n-1}^k - Y_{n-1}^{k,M} + (a^k(\tau_{n-1}, Y_{n-1}) - a^k(\tau_{n-1}, Y_{n-1}^M))\Delta\right. \\
&+ \frac{1}{2}\left[b^k(\tau_n, Y_n) - b^k(\tau_n, Y_n^M) + b^k(\tau_{n-1}, Y_{n-1}) - b^k(\tau_{n-1}, Y_{n-1}^M)\right]\Delta W_{n-1} \\
&- \left.\frac{1}{2}\left[L^1 b^k(\tau_{n-1}, Y_{n-1}) - L^1 b^k(\tau_{n-1}, Y_{n-1}^M)\right]\Delta\right) \\
&+ f_1(\tau_{n-1}, \tau_n, Y_{n-1}^M, Y_n^M)(\Delta \cdot \Delta W_{n-1}) \\
&+ f_2(\tau_{n-1}, \tau_n, Y_{n-1}^M, Y_n^M)((\Delta^2 \cdot \Delta W_{n-1}) \\
&+ f_3(\tau_{n-1}, \tau_n, Y_{n-1}^M, Y_n^M)(\Delta \cdot (\Delta W_{n-1})^2) \\
&+ f_4(\tau_{n-1}, \tau_n, Y_{n-1}^M, Y_n^M)(\Delta W_{n-1})^3 \\
&+ f_5(\tau_{n-1}, \tau_n, Y_{n-1}^M, Y_n^M)(\Delta \cdot (\Delta W_{n-1)}^3)) \\
&+ f_6(\tau_{n-1}, \tau_n, Y_{n-1}^M, Y_n^M)(\Delta W_{n-1})^5,
\end{aligned}
$$

where $\| f_i(\tau_{n-1}, \tau_n, Y_{n-1}^M, Y_n^M) \|^2 \le C_i(1 + \| Y_{n-1}^M \|^2), i = 1, 2, 3, 4, 5, 6.$

Squaring both sides of the equation, taking expectation and from Lemma (2.3) we get

$$\begin{aligned} E(\| Y_{n+1}^k - Y_{n+1}^{k,M} \|^2) &\le E(\| Y_n^k - Y_n^{k,M} \|^2)(K_8 + K_9\Delta + K_{10}\Delta^2) \\ &+ E(\| Y_{n-1}^k - Y_{n-1}^{k,M} \|^2)(K_{11} + K_{12}\Delta + K_{13}\Delta^2) + K_{14}\Delta^3, \end{aligned}$$

where $K_8, K_9, K_{10}, K_{11}, K_{12}, K_{13}$ and K_{14} do not depend on Δ.

Using for the starting routine Milstein approximation i.e. $Y_0^k = Y_0^{k,M}$ and $Y_1^k = Y_1^{k,M}$ we get that for all $n = 0, 1, \dots, N$

$$E(\| Y_n^k - Y_n^{k,M} \|^2) \le K_{15}\Delta^2,$$

where K_{15} does not depend on Δ.

From Lemma 2.4

$$E(\| X_{\tau_n} - Y_n^M \|^2) \le K_{16}(1 + E\| X_0 \|^2)\Delta^2,$$

where K_{16} does not depend on Δ (see in [3]), we apply these results to prove finally the strong order $\gamma = 1.0$ of the multistep method, as is claimed in Theorem 1.

3 Some Experiments

Let us consider the Milstein approximation (4), two-step order 1.0 strong scheme (5) and the approximation set out above (6). The three approximations set out above were each tested on the following examples.

Example 3.1

$$\begin{aligned} dX_t &= 1.5X_t dt + X_t dW_t \\ X_0 &= 1.0, \end{aligned} \tag{7}$$

where (W_t) is a Wiener process.
The solution of (7) is $X_t = X_0 \exp(t + W_t)$

Example 3.2

$$\begin{aligned} dX_t &= \left(\frac{\alpha X_t}{1+t} + X_0(1+t)^\alpha\right) dt + X_0(1+t)^\alpha dW_t \\ X_0 &= 1.0 \textit{ and } \alpha = 2.0 \end{aligned} \tag{8}$$

where (W_t) is a Wiener process.
The solution of (8) is $X_t = (1+t)^2\,(W_t + t + 1.0)$

In each case the mean-square error $E\| X_1 - Y_1 \|^2$ at the final time ($T = 1$) is estimated in the following way. A set of 20 blocks, each consisting of 100 outcomes ($1 \leq i \leq 20,\ 1 \leq j \leq 100$), were simulated and for each block the estimator

$$\varepsilon_i = \frac{1}{100} \sum_{j=1}^{100} \| X_1(\omega_{i,j}) - Y_N(\omega_{i,j}) \|^2$$

was found. Next the means and variances of these estimators were themselves estimated in the usual way:

$$\varepsilon = \frac{1}{20} \sum_{i=1}^{20} \varepsilon_i$$

and

$$\sigma^2 = \frac{1}{19} \sum_{i=1}^{20} (\varepsilon - \varepsilon_i)^2 .$$

According to the central limit theorem, the ε_i should be nearly Gaussian and so approximate 90 percent confidence limits for $E\| X_1 - Y_N \|^2$ can be found from the Gaussian distribution; these were calculated according to the formula $\varepsilon \pm 1.73\sqrt{\frac{\sigma^2}{20}}$.

The results of the simulations for Examples 3.1 and 3.2 are shown in Table 1 and 2. These results are gotten for $\alpha = 0$, $\gamma = 1.0$ in Example 3.1 and for $\alpha = 0$, $\gamma = 1.0$ and $\alpha = 0.5$, $\gamma = 1.0$ in Example 3.2. There is no sense to take γ near zero, because then the new term can be neglected, so the new scheme behaves as Milstein 1.0. The meaning of the headers in the tables is:

Δ - time step size of the strong approximation;

ε - absolute errors for different time step sizes;

L - half of the confidence interval lengths.

For example, we can see from the tables that in Example 3.2 for $\Delta = 2^{-8}$ and $\alpha = 0.0$ and $\gamma = 1.0$ the absolute error by Milstein method (4) is $3.42858 \cdot 10^{-2}$, by Two-step Milstein method (5) is $9.45832 \cdot 10^{-3}$, while by the new scheme (6) is $6.81161 \cdot 10^{-3}$. Also, the length of the confidence interval by the new scheme is smaller than by Milstein 1.0 and Two-step Milstein methods. This statement is also true for the Example 3.1.

Table 1: Example 3.1
Milstein method (4).

Δ	ε	L
1.00000E+00	2.27665E+00	1.47186E-01
5.00000E-01	1.97078E+00	2.40568E-01
2.50000E-01	1.20429E+00	8.45154E-02
1.25000E-01	7.37239E-01	5.64921E-02
6.25000E-02	3.82413E-01	3.99189E-02
3.12500E-02	2.39074E-01	6.31194E-02
1.56250E-02	1.10807E-01	1.27486E-02
7.81250E-03	5.60566E-02	8.09157E-03
3.90625E-03	2.53057E-02	3.36756E-03

Multistep method (6) for $\alpha = 0$ and $\gamma = 1.0$.

Δ	ε	L
1.00000E+00	2.51146E+00	1.98164E-01
5.00000E-01	1.41485E+00	9.57135E-02
2.50000E-01	6.39612E-01	5.46793E-02
1.25000E-01	3.21211E-01	2.94124E-02
6.25000E-02	1.50961E-01	8.22891E-03
3.12500E-02	7.51688E-02	5.73330E-03
1.56250E-02	3.92063E-02	2.09849E-03
7.81250E-03	2.00488E-02	1.25050E-03
3.90625E-03	9.94833E-03	6.94911E-04

Two-step Milstein (5) for $\alpha = 0$ and $\gamma = 1.0$.

Δ	ε	L
1.00000E+00	2.37813E+00	1.87704E-01
5.00000E-01	1.45746E+00	1.12863E-01
2.50000E-01	8.02364E-01	9.38468E-02
1.25000E-01	4.91936E-01	6.26155E-02
6.25000E-02	2.36993E-01	2.86351E-02
3.12500E-02	1.22735E-01	6.91430E-03
1.56250E-02	6.22639E-02	5.80727E-03
7.81250E-03	3.31988E-02	2.88916E-03
3.90625E-03	1.65349E-02	1.28400E-03

Table 2: Example 3.2
Milstein method (4).

Δ	ε	L
1.00000E+00	4.21558E+00	9.211294E-02
5.00000E-01	2.90298E+00	7.181054E-02
2.50000E-01	1.77082E+00	4.158990E-02
1.25000E-01	9.78134E-01	2.936154E-02
6.25000E-02	5.27383E-01	1.338104E-02
3.12500E-02	2.75086E-01	7.950747E-03
1.56250E-02	1.36424E-01	3.334465E-03
7.81250E-03	6.97031E-02	1.644745E-03
3.90625E-03	3.42858E-02	7.971471E-04

Multistep method (6) for $\alpha = 0$ and $\gamma = 1.0$.

Δ	ε	L
1.00000E+00	4.27766E+00	1.03425E-01
5.00000E-01	1.70013E+00	3.85968E-02
2.50000E-01	6.21525E-01	1.72348E-02
1.25000E-01	2.60004E-01	8.05579E-03
6.25000E-02	1.16169E-01	3.57810E-03
3.12500E-02	5.50517E-02	1.51257E-03
1.56250E-02	2.71983E-02	9.70674E-04
7.81250E-03	1.33966E-02	4.02296E-04
3.90625E-03	6.81160E-03	2.22766E-04

Multistep method (6) for $\alpha = 0.5$ and $\gamma = 1.0$.

Δ	ε	L
1.00000E+00	4.17855E+00	1.05099E-01
5.00000E-01	2.22505E+00	4.56814E-02
2.50000E-01	1.15922E+00	3.09267E-02
1.25000E-01	5.91574E-01	1.29769E-02
6.25000E-02	2.90397E-01	5.80337E-03
3.12500E-02	1.43653E-01	3.21847E-03
1.56250E-02	7.27217E-02	2.00281E-03
7.81250E-03	3.50626E-02	8.46181E-04
3.90625E-03	1.77133E-02	3.59233E-04

Two-step Milstein (5) for $\alpha = 0$ and $\gamma = 1.0$.

Δ	ε	L
1.00000E+00	4.24832E+00	9.85367E-02
5.00000E-01	1.77406E+00	5.03204E-02
2.50000E-01	7.62093E-01	1.71932E-02
1.25000E-01	3.37591E-01	1.07679E-02
6.25000E-02	1.60081E-01	5.27565E-03
3.12500E-02	7.77709E-02	2.43576E-03
1.56250E-02	3.73556E-02	1.02419E-03
7.81250E-03	1.96293E-02	5.93383E-04
3.90625E-03	9.45832E-03	2.41391E-04

Two-step Milstein (5) for $\alpha = 0.5$ and $\gamma = 1.0$.

Δ	ε	L
1.00000E+00	4.23623E+00	8.01164E-02
5.00000E-01	2.28984E+00	4.36308E-02
2.50000E-01	1.21665E+00	3.63160E-02
1.25000E-01	6.34940E-01	1.90075E-02
6.25000E-02	3.13706E-01	8.61972E-03
3.12500E-02	1.60810E-01	4.54545E-03
1.56250E-02	8.02790E-02	2.21567E-03
7.81250E-03	4.09332E-02	1.03573E-03
3.90625E-03	2.04743E-02	4.97450E-04

References

[1] Boulean, N. and Lépingle, D. Numerical Method for Stochastic Processes, *John Wiley & Sons*, 1994.

[2] Karatzas, I. and Shreve, S. E. Brownian Motion and Stochastic Calculus, *Springer-Verlag*, 1988.

[3] Kloden, P. E. and Platen, E. Numerical Solution of Stochastic Differential Equations, *Springer-Verlag*, 1992.

[4] Lépingle, D. and Ribémont, B. Un schéma multiplas d'approximation de l'equation de Langein, *Stochastic Processes and their Applications* 37 (1991), 61-69, North Holland

[5] Newton, N. J. Asymptotically efficient Runge-Kutta methods for a class of Itô and Straonocich equations, *SIAM J. Appl. Math.* 51 (1991), 542-567

[6] Rümelin, W. Numerical treatment of stochastic differential equations, *SIAM J. Numer. Anal.* 19 (1982), 604-613

[7] Talay, D. Résolution trajectorielle et analyse numérique des équations différentielles stochastiques, *Stochastics* 9 (1983), 275-306

Received March, 1997

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János
A kézirat a nyomdába érkezett: 1997: december
Terjedelem: 7,12 (B/5) ív