Volume 13 Number 3

# ACTA CYBERNETICA

Szeged, 1998

# 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.

Acta Cybernetica **13** (1998) 209-224.

# NP–completeness results concerning the transformation of logic programs into attribute grammars

Markus Lohrey *

**Abstract**

Attribute grammars and logic programs are two well investigated formalisms, which were related in [DM85] for the restricted class of simple logic programs. In this paper we define the more restricted class of very simple logic programs and we prove that the problem of deciding, whether a given logic program is (very) simple, is NP–complete.

## 1 Introduction

Attribute grammars were introduced in [Knu68] as a formalism for the specification of the semantics of programming languages. [DJ90] and [AM91] give an overview of current research trends. One of the most significant features of attribute grammars are their declarative programming style and the existence of efficient attribute evaluation methods. For the last point, see e.g. [Alb89].

Another declarative programming paradigm are definite program clauses. Their ability to specify computations was first recognized in [Kow74]. For a modern and exhaustive introduction to the wide area of logic programming see [Apt96].

In [DM85] a strong relationship between attribute grammars[1] and definite program clauses has been established. The investigation of this relationship is attractive, because one can try to apply a great number of techniques, which were originally developed for attribute grammars, to logic programs. For instance, dependency graphs can be used to study strictness, necessity of the occur–check or other run–time properties of logic programs. Moreover, attribute evaluation methods can be used for giving alternative operational semantics to logic programs.

It has been shown in [DM85] that every attribute grammar, whose function symbols are interpreted as term constructors, can be transformed into a semantically

*Grundlagen der Programmierung, Institut für Softwaretechnik I, Fakultät Informatik, Technische Universität Dresden, D-01062 Dresden, Germany.

[1]The kind of attribute grammars we use in this paper are called functional attribute grammars in [DM85].

equivalent logic program. However, the reverse construction, i.e., the transformation of a logic program into a semantically equivalent attribute grammar is not always possible. In [DM85] the restricted class of simple logic programs has been defined and a construction has been presented, which transforms a simple logic program into an equivalent attribute grammar. In general, the semantic domain of the resulting attribute grammar is not a free term algebra. In Section 3 we will define the even more restricted class of very simple logic programs. For a very simple logic program the construction in [DM85] produces an attribute grammar, whose semantic domain is a free term algebra. It should be noted that the class of simple logic programs is still Turing–complete whereas very simple logic programs are no longer Turing–complete (in fact, attribute grammars with a free term algebra as semantic domain are not Turing–complete).

For a given logic program it can be decided whether this logic program is (very) simple or not. Since many techniques, developed for attribute grammars, can be applied to simple logic programs and even more to very simple logic programs, see the note above and [DM85], [DM93] for more details, it would be useful to have efficient algorithms for these problems. In this paper we show that the problem to decide, whether a given logic program is (very) simple, is in fact NP–complete. Therefore it is unlikely to find efficient algorithms for these problems.

This paper is organized as follows. Section 2 gives some basic definitions concerning attribute grammars and logic programs. In Section 3 we define the class of simple and very simple logic programs and show how a simple logic program can be transformed into an attribute grammar. Finally in section 4 we prove the NP–completeness results mentioned above.

## 2 Preliminaries

First we recall some basic notions from universal algebra. Throughout this paper we assume that there is a countable infinite set $V$ of variables.

**Definition 2.1.** A *ranked alphabet* $\Gamma$ is a finite set of symbols together with a mapping $rank_\Gamma : \Gamma \to \mathbb{N} = \{0,1,2,\ldots\}$. $\Gamma^{(n)} = \{f \in \Gamma \mid rank_\Gamma(f) = n\}$ is the set of all symbols of rank $n$. The set of all $\Gamma$–*terms* with variables in $V$, denoted by $T_\Gamma(V)$, is the smallest set such that (i) $V \subseteq T_\Gamma(V)$ and (ii) $f \in \Gamma^{(n)}, t_1, \ldots, t_n \in T_\Gamma(V)$ implies $f(t_1, \ldots, t_n) \in T_\Gamma(V)$. For $t \in T_\Gamma(V)$, $V(t)$ denotes the set of all variables, which occur in $t$. If $x \in V(t)$, we say that $x$ *occurs in* $t$.

**Definition 2.2.** A *semantic domain* $D = (\Omega, \Gamma, \Pi, \varphi)$ consists of a set $\Omega$, two ranked alphabets $\Gamma$ and $\Pi$ of *function* and *predicate symbols*, respectively, such that $\Gamma \cap \Pi = \emptyset$, and an *interpretation function* $\varphi$, i.e., for every $n \geq 0$, $f \in \Gamma^{(n)}$, and $q \in \Pi^{(n)}$, $\varphi(f)$ is a partial function $\varphi(f) : \Omega^n \to \Omega$ and $\varphi(q)$ is a relation $\varphi(q) \subseteq \Omega^n$. An *assignment* is a function $val : V \to \Omega$. Every assignment $val$ can be lifted to a function $\widehat{val} : T_\Gamma(V) \to \Omega$ by (i) $\widehat{val}(x) = val(x)$ for $x \in V$ and (ii) $\widehat{val}(f(t_1, \ldots, t_n)) = \varphi(f)(\widehat{val}(t_1), \ldots, \widehat{val}(t_n))$ for $n \geq 0$, $f \in \Gamma^{(n)}$ and

$t_1, \ldots, t_n \in T_\Gamma(V)$ [2].

Note that we allow partial functions. For the rest of the paper we fix two ranked alphabets $\Gamma$ and $\Pi$ of function and predicate symbols, respectively, such that $\Gamma \cap \Pi = \emptyset$.

**Definition 2.3.** The *free* $\Gamma$*-term algebra (generated by* $V$*)*, denoted by $\mathcal{T}_\Gamma(V)$, is the semantic domain $(T_\Gamma(V), \Gamma, \emptyset, \varphi)$, where $\varphi(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for $n \geq 0$, $f \in \Gamma^{(n)}$, and $t_1, \ldots, t_n \in T_\Gamma(V)$. Thus, a free $\Gamma$-term algebra does not contain relations and every function symbol $f \in \Gamma$ is interpreted by itself. A *substitution* is an assignment $\theta : V \to T_\Gamma(V)$ such that $\{x \in V \mid \theta(x) \neq x\}$ is finite. If $\theta(x_i) = t_i$ for $1 \leq i \leq n$ and $\forall x \in V \backslash \{x_1, \ldots, x_n\} : \theta(x) = x$, we use the notation $t\{t_i/x_i \mid 1 \leq i \leq n\}$ rather than $\widehat{\theta}(t)$ for $t \in T_\Gamma(V)$.

Next we have to introduce the machinery of attribute grammars. For the simulation of logic programs it is not necessary to include terminals and start symbols in the grammatical part of attribute grammars. Therefore we just introduce the notion of an abstract context-free grammar (see [GTWW77]), which is equivalent to the notion of a many-sorted signature.

**Definition 2.4.** An *abstract context-free grammar*, or briefly *abstract cfg*, $G_0 = (N, L, P)$ consists of a finite set $N$ of *nonterminals*, a finite set $L$ of labels and a finite set $P \subset L \times N \times N^{*}$[3] of *productions* such that for every $(l, X, \alpha), (l', X', \alpha') \in P$, $l = l'$ implies $(l, X, \alpha) = (l', X', \alpha')$, i.e., different productions are labeled by different labels. The production $(l, X, \alpha)$ will be denoted by $l : X \to \alpha$.

Derivations of abstract cfgs will be represented by trees. Nodes of a tree are specified by means of the well-known Dewey notation, i.e., a tree node $x$ is a string $x = i_1.i_2 \ldots i_n$ with $i_j > 0$ for $1 \leq j \leq n$. Intuitively, this string indicates the path from the root of the tree to $x$. Thus, the root itself is denoted by the string $\epsilon$ of length 0. Furthermore, for technical reasons it is useful to define $x.0 = x$.

**Definition 2.5.** Let $G_0 = (N, L, P)$ be an abstract cfg. A *syntax tree* of $G_0$ is a finite tree $s$ whose nodes are labeled by nonterminals from $N$ such that the following condition holds:

For every node $x$ there is a production $(l : X_0 \to X_1 \ldots X_n) \in P$ with $n \geq 0$ such that $x$ is labeled by $X_0$, $x$ has exactly $n$ successor nodes $x.1, \ldots, x.n$, and $x.i$ is labeled by $X_i$ for $1 \leq i \leq n$. In this situation we say that the production $l : X_0 \to X_1 \ldots X_n$ is applied at the node $x$.

**Definition 2.6.** An attribute grammar, or briefly *ag*, $G = (G_0, D, B, R, C)$ consists of the following components:

- an abstract cfg $G_0 = (N, L, P)$

[2] We assume that $\varphi(f)(a_1, \ldots, a_n)$ is undefined if one of its arguments $a_i$ is undefined.

[3] As usual, $N^*$ denotes the set of all finite sequences of elements of $N$, including the empty sequence, which will be denoted by $\epsilon$.

- a semantic domain $D = (\Omega, \Gamma, \Pi, \varphi)$
- an *attribute description* $B = (Inh, Syn, inh, syn)$. $Inh$ and $Syn$ are finite disjoint sets of *inherited* and *synthesized attributes*, respectively. $inh$ and $syn$ are functions $inh : N \to 2^{Inh}$[4] and $syn : N \to 2^{Syn}$, respectively. $Att = Inh \cup Syn$ is the set of *attributes* of $G$.
- a set $R = \{R(p) \mid p \in P\}$ of finite sets $R(p)$ of *semantic rules*. To every production $p \in P$ of the form $l : X_0 \to X_1 \dots X_n$ with $n \geq 0$ we assign the set

  $$in(p) = \{\langle\gamma, i\rangle \mid (\gamma \in syn(X_0) \wedge i = 0) \vee (\gamma \in inh(X_i) \wedge 1 \leq i \leq n)\}$$

  of *inside attribute occurrences* of $p$ and the set

  $$out(p) = \{\langle\gamma, i\rangle \mid (\gamma \in inh(X_0) \wedge i = 0) \vee (\gamma \in syn(X_i) \wedge 1 \leq i \leq n)\}$$

  of *outside attribute occurrences* of $p$ . $att(p) = in(p) \cup out(p)$ is the set of *attribute occurrences* of $p$. For every $\langle\gamma, i\rangle \in in(p)$, the set $R(p)$ contains exactly one semantic rule of the form $\langle\gamma, i\rangle = t$ with $t \in T_\Gamma(out(p))$. Nothing else belongs to $R(p)$.
- a set $C = \{C(p) \mid p \in P\}$ of finite sets $C(p)$ of *semantic conditions*. For every $p \in P$, very element of $C(p)$ has the form $q(t_1, \dots, t_n)$ with $q \in \Pi^{(n)}$ and $t_i \in T_\Gamma(out(p))$ for $1 \leq i \leq n$.

$G$ is called *free* iff there is a ranked alphabet $\Gamma$ such that $D$ is the free $\Gamma$-term algebra $\mathcal{T}_\Gamma(V)$. In particular, a free ag does not contain semantic conditions.

The semantic of an ag is the set of all (correctly) decorated syntax trees of the underlying abstract cfg, defined as follows.

**Definition 2.7.** Let $G = (G_0, D, B, R, C)$ be an ag with an underlying abstract cfg $G_0 = (N, L, P)$ and a semantic domain $D = (\Omega, \Gamma, \Pi, \varphi)$. Furthermore let $s$ be a syntax tree of $G_0$ and $x$ be a node of $s$, which is labeled by $X \in N$. To $x$ we assign the set $inh(x) = \{\langle\gamma, x\rangle \mid \gamma \in inh(X)\}$ of *inherited attribute instances* of $x$ and the set $syn(x) = \{\langle\gamma, x\rangle \mid \gamma \in syn(X)\}$ of *synthesized attribute instances* of $x$. $inst(x) = inh(x) \cup syn(x)$ is the set of *attribute instances* of $x$. The set $inst(s)$ of all attribute instances of $s$ is $\bigcup\{inst(x) \mid x \text{ is a node of } s\}$. A *decoration* of $s$ is a function $val : inst(s) \to \Omega$. A decoration $val$ is called *valid* iff for every node $x$ of $s$ the following condition holds:

Assume that production $p$ is applied at node $x$. Let $\theta : out(p) \to inst(s)$ be the substitution, defined by $\theta(\langle\gamma', j\rangle) = \langle\gamma', x.j\rangle$ for every $\langle\gamma', j\rangle \in out(p)$ (recall that we defined $x.0 = x$), where we assume w.l.o.g. that $out(p)$ is contained in the set $V$ of variables. Then for every semantic rule $\langle\gamma, i\rangle = t$ in $R(p)$, the equation $val(\langle\gamma, x.i\rangle) = \widehat{val}(\widehat{\theta}(t))$ has to hold in $D$. Furthermore, for every semantic condition $q(t_1, \dots, t_n) \in C(p)$ we must have $(\widehat{val}(\widehat{\theta}(t_1)), \dots, \widehat{val}(\widehat{\theta}(t_n))) \in \varphi(q)$.

[4]For every set $A$, $2^A$ denotes the powerset of $A$.

A *decorated tree* of $G$ is a pair, consisting of a syntax tree $s$ of $G_0$ and a valid decoration of $s$.

In most investigations only the class of so called non–circular attribute grammars is considered, since non–circularity of an ag $G$ is sufficient for the construction of an attribute evaluator for $G$. For our purpose this restriction is not necessary. Next we give some basic definitions concerning logic programs.

**Definition 2.8.** The set of all $(\Gamma, \Pi)$–*atoms* with variables in $V$, denoted by $A_{\Gamma,\Pi}(V)$, is the set $\{q(t_1, \ldots, t_n) \mid q \in \Pi^{(n)} \wedge t_1, \ldots, t_n \in T_\Gamma(V)\}$. $A_{\Gamma,\Pi}(\emptyset)$ is the set of all *ground* $(\Gamma, \Pi)$–atoms. $V(q(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} V(t_i)$ is the set of all variables, which occur in the atom $q(t_1, \ldots, t_n)$. A *definite* $(\Gamma, \Pi)$–*clause*, or just *clause*, is a formula of the form $\forall x_1, \ldots, x_n(A_1 \wedge \ldots \wedge A_m \rightarrow A_0)$, where $m \geq 0$, $A_0, \ldots, A_m \in A_{\Gamma,\Pi}(V)$ and $\{x_1, \ldots, x_n\} = \bigcup_{i=0}^{m} V(A_i)$. In the sequel we will use the abbreviation $A_0 \leftarrow A_1, \ldots, A_m$ for this clause. A *definite logic program*, or briefly *dlp*, is a triple $H = (\Gamma, \Pi, U)$, where $U$ is a finite set of definite $(\Gamma, \Pi)$–clauses.

**Definition 2.9.** Let $\theta$ be a substitution. The atom $\theta(A) = q(\hat{\theta}(t_1), \ldots, \hat{\theta}(t_n))$ is called an *instance* of the atom $A = q(t_1, \ldots, t_n)$. The clause $\theta(A_0) \leftarrow \theta(A_1), \ldots, \theta(A_m)$ is called an instance of the clause $A_0 \leftarrow A_1, \ldots, A_m$.

There are several ways to assign a semantic to a dlp $H = (\Gamma, \Pi, U)$. Probably the way, which is familiar to most of the readers, is the least Herbrand model of $H$ (see [Apt96]). The least Herbrand model of $H$ consists of the set of all ground atoms that are logical consequences of $U$. This set can be generated by an operational method as well.

**Definition 2.10.** Let $H = (\Gamma, \Pi, U)$ be a dlp. A *proof tree* of $H$ is a finite tree $s$ whose nodes are labeled by atoms from $A_{\Gamma,\Pi}(V)$ such that the following condition holds:

For every node $x$ of $s$ there is an instance $A_0 \leftarrow A_1, \ldots, A_n$ $(n \geq 0)$ of a clause in $U$ such that $x$ is labeled by $A_0$, $x$ has exactly $n$ successor nodes $x.1, \ldots, x.n$, and $x.i$ is labeled by $A_i$ for $1 \leq i \leq n$.

As shown in [Cla78] the least Herbrand model of $H$ consists of all ground atoms that are the root of a proof tree of $H$.

## 3 Transformation of definite logic programs into attribute grammars

In the rest of the paper we fix a dlp $H = (\Gamma, \Pi, U)$.

The material in this section is taken from [DM85]. Our aim is to translate the dlp $H$ into a semantically equivalent ag $G$. What it means precisely that the dlp $H$ and the ag $G$ are semantically equivalent, will be defined at the end of this section. Intuitively, the predicate symbols in $\Pi$ will become the nonterminals of $G$. The clauses in $U$ will be translated into productions of $G$ by simply removing the argument positions of the predicate symbols in $\Pi$. To store the values of the argument positions of $q \in \Pi^{(n)}$, we assign $n$ attributes $q1, \dots, qn$ to $q$, one for each argument position $j \in \{1, \dots, n\}$ of $q$. Now the formalism of definite logic programs gives no hint of how to split this attribute set into inherited and synthesized attributes. Therefore we have to enrich $H$ with some additional information.

**Definition 3.1.** A *direction assignment*, or briefly *d-assignment*, for $H$ is a function $d$, mapping every pair $(q, j)$ with $q \in \Pi^{(n)}$ and $1 \leq j \leq n$ to an element of the set $\{\uparrow, \downarrow\}$.

$d(q,j) = \uparrow$ means that $qj$ becomes a synthesized attribute, whereas $d(q,j) = \downarrow$ means that $qj$ becomes an inherited attribute. In the rest of this section, let $d$ be a d–assignment for the dlp $H$. Since the value of an inside attribute occurrence must be calculated by use of the outside attribute occurrences, we have to impose a restriction on $d$.

**Definition 3.2.** Let

$$u = (q_0(s_{0,1}, \dots, s_{0,n_0}) \leftarrow q_1(s_{1,1}, \dots, s_{1,n_1}), \dots, q_m(s_{m,1}, \dots, s_{m,n_m})) \in U. \quad (1)$$

$POS(u) = \{\langle q_i j, i \rangle \mid 0 \leq i \leq m \wedge 1 \leq j \leq n_i\}$ is the set of *positions* of $u$ (this set will become the set of attribute occurrences of the translation of the clause $u$). For $0 \leq i \leq m$, $1 \leq j \leq n_i$, the term $s_{i,j}$ is denoted by $t(\langle q_i j, i \rangle, u)$. For $pos \in POS(u)$, we say that $t(pos, u)$ *appears at the position pos of u*. The set $POS(u)$ is partitioned into the sets $in_d(u)$ and $out_d(u)$ of *inside* and *outside positions under d of u*, respectively, defined as follows:

$$in_d(u) = \{\langle q_i j, i \rangle \mid (i = 0 \wedge d(q_0, j) = \uparrow) \vee (1 \leq i \leq m \wedge d(q_i, j) = \downarrow)\}$$

$$out_d(u) = \{\langle q_i j, i \rangle \mid (i = 0 \wedge d(q_0, j) = \downarrow) \vee (1 \leq i \leq m \wedge d(q_i, j) = \uparrow)\}$$

$d$ is called *safe for H* iff

$$\forall u \in U \, \forall pos \in in_d(u) \, \forall x \in V(t(pos, u)) \, \exists pos' \in out_d(u) : x \in V(t(pos', u)).$$

This condition says that for every clause $u \in U$, every variable that occurs in a term that appears at an inside position of $u$ occurs also in a term that appears at some outside position of $u$. $d$ is called *very safe for H* iff $d$ is safe for $H$ and the following additional condition holds:

$$\forall u \in U \, \forall pos, pos' \in out_d(u) : t(pos, u), t(pos', u) \in V \wedge (pos \neq pos' \Rightarrow t(pos, u) \neq$$
$$\neq t(pos', u))$$

This condition says that for every clause $u \in U$, the terms that appear at the outside positions of $u$ are different variables. $H$ is called *simple* iff a safe d–assignment for $H$ exists. $H$ is called *very simple* iff a very safe d–assignment for $H$ exists.

**Example 3.3.** Let $H = (\{nil, cons\}, \{app\}, \{u_1, u_2\})$ be the well-known append-dlp, where $u_1 = (app(nil, L, L) \leftarrow\ )$, $u_2 = (app(cons(X, L_1), L_2, cons(X, L_3)) \leftarrow app(L_1, L_2, L_3))$, and $X, L_1, L_2, L_3$ are variables. Let a d-assignment $d$ for $H$ be given by $d(app, 1) = d(app, 2) =\downarrow$, $d(app, 3) =\uparrow$. $d$ is a safe but not very safe d-assignment for $H$. In fact, $H$ is not very simple.

**Example 3.4.** Simple dlps are Turing-complete since every two-counter machine can be simulated by a simple dlp. A two-counter machine $M$ consists of a finite set $Q$ of states, an initial state $q_0 \in Q$, a set $F \subseteq Q$ of final states, and a finite set $R$ of statements. Every $r \in R$ has the following form, where $q$ denotes the current state, $x$ and $y$ are two registers whose values range over $\mathbb{N}$, and $i, j \in Q$.

- if $q = i$ then $x := x + 1$ and $q := j$ (analogously for the register $y$)
- if $q = i$ then $x := x - 1$ and $q := j$ (analogously for the register $y$)
- if $q = i$ and $x = 0$ then $q := j$ else $q := k$ (analogously for the register $y$)

A calculation of $M$ is defined in the obvious way. $M$ can be simulated by the dlp $H(M) = (\{0, s\}, \{P_i \mid i \in Q\}, \{P_i(x, y) \leftarrow\ \mid i \in F\} \cup \bigcup_{r \in R} U_r)$, where $U_r$ consists of the following clauses, depending on the form of the statement $r$.

- $P_i(x, y) \leftarrow P_j(s(x), y)$ (analogously for $y$)
- $P_i(s(x), y) \leftarrow P_j(x, y)$ (analogously for $y$)
- $P_i(0, y) \leftarrow P_j(0, y)$, $P_i(s(x), y) \leftarrow P_k(s(x), y)$ (analogously for $y$)

The d-assignment $d$ for $H(M)$, which is given by $d(P_i, 1) = d(P_i, 2) =\downarrow$ for every $i \in Q$ is a safe d-assignment for $H(M)$. On the other hand, if for instance there exist two statements of the form (if $q = k$ then $x := x + 1$ and $q := i$) and (if $q = i$ then $x := x - 1$ and $q := j$) then $H(M)$ is not very simple. In fact, it is not difficult to see that very simple dlps are not Turing-complete.

**Example 3.5.** Let $H = (\{0, s\}, \{plus\}, \{u_1, u_2\})$ be the well-known dlp for adding natural numbers, where $u_1 = (plus(0, x, x) \leftarrow\ )$ and $u_2 = (plus(s(x), y, s(z)) \leftarrow plus(x, y, z)$. Let a d-assignment $d$ for $H$ be given by $d(app, 1) = d(app, 3) =\uparrow$, $d(app, 2) =\downarrow$. $d$ is a very safe d-assignment for $H$.

Since for every given dlp there are only finitely many different d-assignments, it is decidable whether a dlp is simple or very simple, respectively. We call the corresponding computational problems SIMPLE and VERY-SIMPLE, respectively. A problem instance of (VERY-)SIMPLE consists of a dlp $H$. The question is whether $H$ is (very) simple or not.

In the rest of this section we will show that a (very) simple dlp $H$ together with a (very) safe d-assignment $d$ for $H$ can be transformed into a semantically equivalent (free) ag $G(H, d)$. This result is not necessary for understanding our main results in Section 4 but we present it for completeness.

If the d–assignment $d$ is very safe for $H$, then for every clause $u \in U$, the terms in $\{t(pos,u) \mid pos \in in_d(u)\}$ can be constructed from the terms in $\{t(pos,u) \mid pos \in out_d(u)\}$ (which are variables in this case) and the function symbols in $\Gamma$. If $d$ is only safe but not very safe, we need additional special selector functions $s_i\text{–}f$, defined as follows:

**Definition 3.6.** For every $f \in \Gamma^{(n)}$ and $i \in \{1,\ldots,n\}$, $s_i\text{–}f$ is a partial function on $T_\Gamma(V)$, defined by (i) $s_i\text{–}f(t) = t_i$ if $t = f(t_1,\ldots,t_n)$ and (ii) $s_i\text{–}f(t) =$ undefined otherwise. $s_i\text{–}f$ is called a *selector function*.

In order to refer to selector functions, for every $s_i\text{–}f$ we introduce a new function symbol $sel_i\text{–}f$. Of course we assume that $sel_i\text{–}f \notin \Gamma$. We introduce the new ranked alphabet $\Gamma' = \Gamma \cup \{sel_i\text{–}f \mid f \in \Gamma^{(n)}, n \geq 0, 1 \leq i \leq n\}$, where (i) $rank_{\Gamma'}(f) = rank_\Gamma(f)$ if $f \in \Gamma$ and (ii) $rank_{\Gamma'}(f) = 1$ otherwise.

For a term $t \in T_\Gamma(V)$ and a variable $x \in V$, we denote by $Sel(t,x)$ the set of all terms $s = sel_{i_1}\text{–}f_1(\ldots sel_{i_k}\text{–}f_k(x)\ldots)$ such that the term $s\{t/x\}$ evaluates the variable $x$ when every $sel_i\text{–}f$ is interpreted by $s_i\text{–}f$. In other words, $i_k.i_{k-1}.\ldots.i_1$ specifies a path from the root of $t$ to an occurrence of the variable $x$ in $t$. This is formalized by the following definition.

**Definition 3.7.** Let $t \in T_\Gamma(V)$ and $x \in V$. The set $Sel(t,x)$ is defined by (i) $Sel(x,x) = \{x\}$, (ii) $Sel(y,x) = \emptyset$ for $y \in V \backslash \{x\}$, and (iii) $Sel(f(t_1,\ldots,t_n),x) = \bigcup_{i=1}^{n} \{s\{sel_i\text{–}f(x)/x\} \mid s \in Sel(t_i,x)\}$ for $n \geq 0$, $f \in \Gamma^{(n)}$ and $t_1,\ldots,t_n \in T_\Gamma(V)$.

For instance, $Sel(g(f(x),g(x,y)),x) = \{sel_1\text{–}f(sel_1\text{–}g(x)), sel_2\text{–}g(sel_2\text{–}g(x))\}$.

In addition to the selector functions, semantic conditions are also necessary for the simulation of the dlp $H$, if $d$ is safe but not very safe. This is because of two reasons. Firstly, if there exist an $u \in U$ and a $pos \in out_d(u)$ such that the term $t(pos,u)$ is not just a variable, we have to express the fact that the value of the attribute occurrence $pos$ is an instance of $t(pos,u)$. This will be done with the help of a predicate symbol $instance_{t(pos,u)}$ of rank one. Secondly, if there exist $pos, pos' \in out_d(u)$ and $x \in V$ such that $x \in V(t(pos,u)) \cap V(t(pos',u))$, we must express the fact that the corresponding subterms of the values of the attribute occurrences $pos$ and $pos'$ are equal. Therefore we have to include the syntactic equality of terms into the semantic domain of the simulating ag. Note that both situations cannot occur if $d$ is very safe for $H$.

**Construction 3.8.** Let $d$ be a safe d–assignment for $H$. The ag $G(H,d) = (G_0, D, B, R, C)$ is defined as follows:

- $G_0 = (\Pi, U, P)$, where for every clause $u \in U$ of the form shown in (1) we put the production $p(u) = (u : q_0 \to q_1, \ldots, q_m)$ into $P$. Nothing else belongs to $P$.

- $D = (T_\Gamma(V), \Gamma', \{=\} \cup \{instance_t \mid \exists u \in U \exists pos \in out_d(u) : t = t(pos,u) \notin V\}, \varphi)$, where

  – $\varphi$ interprets every $f \in \Gamma$ as in the $\Gamma$–*term algebra* $T_\Gamma(V)$,

  - $\varphi(sel_i\text{–}f) = s_i\text{–}f$ for $f \in \Gamma^{(n)}$ and $1 \leq i \leq n$,
  - $\varphi(instance_{t_1})(t_2)$ iff $t_2$ is an instance of $t_1$ for $t_1, t_2 \in T_\Gamma(V)$,
  - $\varphi(=)$ is the syntactic equality on $T_\Gamma(V)$.

- $B = (Inh, Syn, inh, syn)$, where for every $q \in \Pi^{(n)}$, $inh(q) = \{qi \mid 1 \leq i \leq n, d(q,i) = \downarrow\}$ and $syn(q) = \{qi \mid 1 \leq i \leq n, d(q,i) = \uparrow\}$. $Inh = \bigcup_{q \in \Pi} inh(q)$ and $Syn = \bigcup_{q\in\Pi} syn(q)$.

- For every clause $u \in U$, $R(p(u)) = \{pos = t_{pos,u} \mid pos \in in_d(u)\}$, where for $pos \in in_d(u)$ the term $t_{pos,u} \in T_{\Gamma'}(V)$ is defined as follows:

  For every $x \in V(t(pos,u))$ let $pos_x \in out_d(u)$ such that $x \in V(t(pos_x,u))$. Since $d$ is safe, such a $pos_x$ must exist for every $x \in V(t(pos,u))$. Of course there may be several choices for $pos_x$. Therefore the construction is non-deterministic. Now let $t_x \in Sel(t(pos_x,u),x)$. Again there may be several choices for $t_x$. We set $t_{pos,u} = t(pos,u)\{t_x\{pos_x/x\}/x \mid x \in V(t(pos,u))\} \in T_{\Sigma'}(out_d(u))$.

- For every clause $u \in U$, $C(p(u)) = C_1(p(u)) \cup C_2(p(u))$, where
  - $C_1(p(u)) = \{instance_{t(pos,u)}(pos) \mid pos \in out_d(u), t(pos,u) \notin V\}$ and
  - $C_2(p(u)) = \{t\{pos/x\} = t'\{pos'/x\} \mid pos, pos' \in out_d(u), x \in V, t \in Sel(t(pos,u),x),$
    $t' \in Sel(t(pos',u),x)\}$. Of course, equations of the form $t = t$ can be omitted in $C_2(p(u))$.

**Example 3.9.** Let $H$ be the append program from Example 3.3 and let $d$ be the d–assignment for $H$ from the same example. The ag $G(H,d) = (G_0, D, B, R, C)$ has the following components.

- $G_0 = (\{app\}, \{u_1, u_2\}, \{u_1 : app \to app, u_2 : app \to\})$

- $B = (\{app1, app2\}, \{app3\}, inh, syn)$, where $inh(app) = \{app1, app2\}$ and $syn(app) = \{app3\}$.

- $R = \{R(u_1), R(u_2)\}$, where $R(u_1) = \{\langle app3, 0\rangle = \langle app2, 0\rangle\}$ and

$$R(u_2) = \{\langle app3,0\rangle = cons(s_1\text{–}cons(\langle app1,0\rangle), \langle app3,1\rangle),$$
$$\langle app1,1\rangle = s_2\text{–}cons(\langle app1,0\rangle), \langle app2,1\rangle = \langle app2,0\rangle\}.$$

- $C = \{C(u_1), C(u_2)\}$, where $C(u_1) = \{instance_{nil}(\langle app1, 0\rangle)\}$ and $C(u_2) = \{instance_{cons(X,L_1)}(\langle app1,1\rangle)\}$.

If $d$ is very safe, then $C(p(u)) = \emptyset$ for every $u \in U$. Moreover, the function symbols $sel_i\text{–}f$ do not appear in the semantic rules of $G(H,d)$. Therefore we can omit them in the semantic domain $D$ of $G(H,d)$. In this way $G(H,d)$ becomes a free ag.

Let $\Phi_H$ be the function, defined as follows. $\Phi_H$ maps a proof tree $s$ of the dlp $H$ to a pair, consisting of a syntax tree $s'$ of $G_0$ (the underlying abstract cfg of ag $G(H,d)$) and a decoration *val* of $s'$. $s'$ results from $s$ by replacing for every node $x$ of $s$ the label $q(t_1,\dots,t_n)$ of $x$ by the label $q$. *val* is defined by $val(\langle qi, x\rangle) = t_i$ for every node $x$ of $s$ with label $q(t_1,\dots,t_n)$ and $1 \leq i \leq n$.

**Theorem 3.10.** Let $H = (\Gamma, \Pi, U)$ be a simple dlp and let $d$ be a safe d–assignment for $H$. Furthermore, let $G(H,d)$ be the ag as constructed in Construction 3.8. Then $\Phi_H$ is a bijective mapping between the set of all proof trees of $H$ and the set of all decorated trees of $G(H,d)$.

The proof of this theorem can be found in [DM85].

## 4 Complexity of SIMPLE and VERY–SIMPLE

In this section we show that the computational problems SIMPLE and VERY–SIMPLE are NP–complete problems. The NP–completeness of SIMPLE will be proved by a reduction from SAT, which is the satisfaction problem for boolean expressions (see [Coo71] and [GJ79]), to SIMPLE. For the corresponding result for VERY–SIMPLE we will use a variant of SAT, called ONE–SAT, that will be introduced later.

**Theorem 4.1.** SIMPLE is NP–complete.

*Proof.* Since we can guess a d–assignment for the dlp $H$ and check in polynomial time, whether this d–assignment is safe for $H$, SIMPLE is in NP. To prove that SIMPLE is NP–hard, we will construct a polynomial time reduction from SAT to SIMPLE. Firstly, we recall the definition of SAT. A problem instance $S$ of SAT is a boolean expression, which can be assumed to be in conjunctive normal form, i.e.,

$$S = C_1 \wedge C_2 \wedge \dots \wedge C_n, \tag{2}$$

where $C_i$ is a nonempty disjunction of literals $A_{i,1}, \dots, A_{i,l_i}$ ($l_i > 0$) with

$$A_{i,j} = \begin{cases} \alpha_{i,j} & \text{or} \\ (\neg \alpha_{i,j}), & \end{cases} \tag{3}$$

where the $\alpha_{i,j}$ are boolean variables. To ease notation we will write the disjunction $C_i$ as a set[5], i.e.,

$$C_i = \{A_{i,1}, \dots, A_{i,l_i}\} \neq \emptyset. \tag{4}$$

Finally, let

$$\{\alpha_1, \dots, \alpha_m\} = \{\alpha_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq l_i\} \tag{5}$$

[5]Therefore in each disjunction a literal can only appear once. Of course this is not a real restriction.

be the set of all boolean variables that occur in $S$. Given such a problem instance $S$ the question is, whether there exists a truth assignment for $S$ which satisfies $S$, i.e., whether there exists a function

$$w : \{\alpha_1, \dots, \alpha_m\} \to \{true, false\} \tag{6}$$

such that

$$\forall i \in \{1, \dots, n\} \exists j \in \{1, \dots, m\} : (\alpha_j \in C_i \wedge w(\alpha_j) = true) \vee ((\neg\alpha_j) \in C_i \wedge w(\alpha_j) =$$
$$= false).$$

In [Coo71], this question was shown to be NP–complete.

In order to construct a polynomial time reduction from SAT to SIMPLE let $S$ be the problem instance of SAT given by (2) to (5). We construct a dlp

$$H(S) = (\{a\}, \{q\}, \{U(C_1), \dots, U(C_n)\}).$$

The only function symbol $a$ is of rank 0. The only predicate symbol $q$ is of rank $m$, which is the number of different boolean variables appearing in $S$, see (5). For every $1 \le i \le n$ the clause $U(C_i)$ is of the form

$$q(s_{i,1}, \dots, s_{i,m}) \leftarrow q(t_{i,1}, \dots, t_{i,m}).$$

We fix a variable $x \in V$. For $1 \le j \le m$ the terms $s_{i,j}$ and $t_{i,j}$ are defined by

$$s_{i,j} = \begin{cases} x & \text{if } \alpha_j \in C_i \\ a & \text{otherwise} \end{cases} \quad \text{and} \quad t_{i,j} = \begin{cases} x & \text{if } (\neg\alpha_j) \in C_i \\ a & \text{otherwise.} \end{cases} \tag{7}$$

Let $w$ be the truth assignment for $S$ given in (6). We define a corresponding d–assignment $d_w$ of $H(S)$ by

$$d_w(q, j) = \begin{cases} \uparrow & \text{if } w(\alpha_j) = false \\ \downarrow & \text{if } w(\alpha_j) = true \end{cases} \tag{8}$$

for $1 \le j \le m$. We will use the abbreviation $d_w(j)$ for $d_w(q, j)$.

**Example 4.2.** Let

$$S = (\alpha_1 \vee \alpha_2 \vee \alpha_3) \wedge (\neg\alpha_2 \vee \neg\alpha_3 \vee \alpha_4) \wedge (\alpha_1 \vee \alpha_3 \vee \neg\alpha_4).$$

Then $H(S)$ consists of the following clauses:

$$q(x, x, x, a) \leftarrow q(a, a, a, a)$$

$$q(a, a, a, x) \leftarrow q(a, x, x, a)$$

$$q(x, a, x, a) \leftarrow q(a, a, a, x)$$

Let $w$ be the following truth assignment:

$$w(\alpha_1) = w(\alpha_3) = w(\alpha_4) = false, w(\alpha_2) = true$$

In fact $w$ satisfies $S$. The corresponding d–assignment $d_w$ is

$$d_w(1) = d_w(3) = d_w(4) = \uparrow, d_w(2) = \downarrow .$$

$d_w$ is a safe d–assignment for $H(S)$. The following claim shows that this is not just a coincidence.

**Claim 4.3.** $w$ satisfies $S$ iff $d_w$ is a safe d–assignment for $H(S)$.

*Proof of the claim.* $w$ satisfies $S$ iff

$$\forall i \in \{1, \dots, n\} \exists j \in \{1, \dots, m\} : (\alpha_j \in C_i \wedge w(\alpha_j) = true) \vee ((\neg\alpha_j) \in C_i \wedge w(\alpha_j) = \\ = false).$$

Because of (7) and (8) this is equivalent to

$$\forall i \in \{1, \dots, n\} \exists j \in \{1, \dots, m\} : (s_{i,j} = x \wedge d_w(j) = \downarrow) \vee (t_{i,j} = x \wedge d_w(j) = \uparrow). \tag{9}$$

We claim that (9) is equivalent to the statement that $d_w$ is a safe d–assignment for $H(S)$.

Assume that (9) holds and consider a clause $U(C_i)$. (9) says that $x$ appears at an outside position $pos \in out_{d_w}(U(C_i))$ of $U(C_i)$. Since $x$ is the only variable that occurs in $U(C_i)$ (and therefore is the only variable $y$ that occurs in a term that appears at an inside position of $U(C_i)$), this shows that $d_w$ is a safe d–assignment for $H(S)$.

Now assume that $d_w$ is a safe d–assignment for $H(S)$ and consider a clause $U(C_i)$ of $H(S)$. We claim that $x$ appears at an outside position of $U(C_i)$. Since $C_i \neq \emptyset$, there exists a $j \in \{1, \dots, m\}$ such that either $s_{i,j} = x$ or $t_{i,j} = x$. Thus, $x$ appears at some position $pos$ of $U(C_i)$. If $pos$ is an outside position of $U(C_i)$, we are ready. Thus, assume that $pos$ is an inside position of $U(C_i)$. Since $d_w$ is a safe d–assignment for $H(S)$, there exists an outside position $pos' \in out_{d_w}(U(C_i))$ of $U(C_i)$ such that $x$ occurs in the term that appears at position $pos'$ of $U(C_i)$. Since $H(S)$ does not contain function symbols of rank greater than zero, this implies that $x$ appears at the outside position $pos'$ of $U(C_i)$, which proves (9). □

Since the function that maps a truth assignment $w$ for $S$ to the d–assignment $d_w$ of $H(S)$ is bijective, the claim shows that we have constructed a correct reduction from SAT to SIMPLE. Since this reduction can be done in polynomial time, the proof of the theorem is complete. □

The proof above shows that for the class of logic programs with only one constant symbol (and no other function symbol) and only one predicate symbol, SIMPLE is already NP–complete.

**Theorem 4.4.** VERY–SIMPLE is NP–complete.

*Proof* Of course, VERY–SIMPLE is also in NP. To prove that VERY–SIMPLE is NP-hard, we use a variant of SAT, called ONE–SAT (see [Sch78], where the problem is called ONE-IN-THREE 3SAT). Again a problem instance of ONE–SAT is a disjunctive normal form $S$. The question is whether there exists a truth assignment $w$ for $S$ such that in every disjunction $C_i$ *exactly one* literal $A_{i,j}$ becomes true under $w$. This problem is known to be NP–complete as well. Let $S$ be the disjunctive normal form $S$ given in (2) – (5) and let $w$ be the truth assignment for $S$ given in (6).

We will construct a dlp $H'(S)$ that is a mild variant of the dlp $H(S) = (\{a\}, \{q\}, \{U(C_1), \dots,$
$U(C_n)\})$ constructed in the proof of Theorem 4.1. It is easy to see that the dlp $H(S)$ and the d–assignment $d_w$ given in (8) satisfy the following fact:

> $w$ satisfies exactly one literal $A_{i,j}$ in every disjunction $C_i$ iff there is exactly one $pos \in out_{d_w}(U(C_i))$ for every clause $U(C_i)$ such that $x \in V(t(pos, U(C_i)))$.

Therefore, if $w$ satisfies exactly one literal $A_{i,j}$ in every disjunction $C_i$, the d–assignment $d_w$ is almost very safe for $H(S)$. The only problem that may arise is that there may exist a clause $U(C_i)$ and an outside position $pos \in out_{d_w}(U(C_i))$ such that $t(pos, U(C_i)) = a$, i.e., a term, which is not a variable, appears at an outside position of $U(C_i)$. We solve this problem by replacing every occurrence of the constant $a$ in $U(C_i)$ by a new variable (different from $x$ and all other new variables that will be introduced). Call the resulting clause $U''(C_i)$. Now it is possible that one of these new variables appears at an inside position $pos \in in_{d_w}(U''(C_i))$ of $U''(C_i)$ and thus has to appear also at an outside position of $U''(C_i)$. To compensate this, we collect all new variables of the clause $U''(C_i)$ (i.e., all variables different from $x$ that occur in $U''(C_i)$) in the argument positions of a new predicate symbol $p_i$ and attach the resulting atom to the body of $U''(C_i)$. The resulting clause will be called $U'(C_i)$. The d–assignments for the new predicate symbol $p_i$ are chosen in such a way that every new variable appears exactly once at an outside position $pos \in out_{d_w}(U'(C_i))$ of $U'(C_i)$ and once at an inside position $pos' \in in_{d_w}(U'(C_i))$ of $U'(C_i)$. A formal presentation of this idea follows.

As motivated, every clause $C_i$ is translated into a clause $U'(C_i)$ of the form

$$q(s'_{i,1}, \dots, s'_{i,m}) \leftarrow q(t'_{i,1}, \dots, t'_{i,m}), p_i(u_{i,1}, \dots, u_{i,m_i}).$$

The number $m_i$ will be specified later. Let $Y = \{y_1, y_2, \dots\} \subset V$ and $Z = \{z_1, z_2, \dots\} \subset V$ be two disjoint sets of new variables different from $x$. The terms

$s'_{i,j}$ and $t'_{i,j}$ are defined by

$$s'_{i,j} = \begin{cases} x & \text{if } \alpha_j \in C_i \\ y_j & \text{otherwise} \end{cases} \quad \text{and} \quad t'_{i,j} = \begin{cases} x & \text{if } (\neg\alpha_j) \in C_i \\ z_j & \text{otherwise} \end{cases}$$

(note the small difference to the terms $s_{i,j}$ and $t_{i,j}$ in the proof of Theorem 4.1). Fix an $i \in \{1, \ldots, n\}$. Let $K = \{j \mid 1 \leq j \leq m \wedge s'_{i,j} \in Y\} = \{k_1, \ldots, k_{\#K}\}$[6] and $L = \{j \mid 1 \leq j \leq m \wedge t'_{i,j} \in Z\} = \{l_1, \ldots, l_{\#L}\}$ be the set of all argument positions of $q$, where variables from $Y$ respectively $Z$ appear in $U'(C_i)$. W.l.o.g. we assume that $k_i < k_j$ and $l_i < l_j$ for every $i < j$. Define $m_i = \#K + \#L$. The terms $u_{i,j}$ are defined by

$$u_{i,j} = \begin{cases} y_{k_j} & \text{if } 1 \leq j \leq \#K \\ z_{l_{j-\#K}} & \text{if } \#K < j \leq m_i. \end{cases}$$

For a given truth assignment $w$ for $S$, the d–assignments $d_w(q,j)$ for the positions of the predicate symbol $q$ are defined as in (8). The d–assignments $d_w(p_i,j)$ for the argument positions of the new predicate symbol $p_i$ are defined by[7]

$$d_w(p_i,j) = \begin{cases} d_w(q,k_j) & \text{if } 1 \leq j \leq \#K \\ d_w(q,l_{j-\#K})^{-1} & \text{if } \#K < j \leq m_i. \end{cases}$$

**Example 4.5.** Let $S$ be the conjunctive normal form given in Example 4.2. $H'(S)$ consists of the following clauses:

$$\begin{aligned} q(x,x,x,y_4) &\leftarrow q(z_1,z_2,z_3,z_4), p_1(y_4,z_1,z_2,z_3,z_4) \\ q(y_1,y_2,y_3,x) &\leftarrow q(z_1,x,x,z_4), p_2(y_1,y_2,y_3,z_1,z_4) \\ q(x,y_2,x,y_4) &\leftarrow q(z_1,z_2,z_3,x), p_3(y_2,y_4,z_1,z_2,z_3) \end{aligned}$$

Let $w$ be the truth assignment given in Example 4.2. $w$ satisfies exactly one literal in each disjunction $C_i$. The corresponding d-assignment $d_w$ is

- $d_w(q,1) = d_w(q,3) = d_w(q,4) = \uparrow, d_w(q,2) = \downarrow$
- $d_w(p_1,1) = d_w(p_1,3) = \uparrow, d_w(p_1,2) = d_w(p_1,4) = d_w(p_1,5) = \downarrow$
- $d_w(p_2,1) = d_w(p_2,3) = \uparrow, d_w(p_2,2) = d_w(p_2,4) = d_w(p_2,5) = \downarrow$
- $d_w(p_3,2) = d_w(p_3,4) = \uparrow, d_w(p_3,1) = d_w(p_3,3) = d_w(p_3,5) = \downarrow$ .

This is a very safe d–assignment for $H'(S)$.

The following claim completes the proof. Its proof is very similar to the proof of the corresponding claim, made in the proof of Theorem 4.1, and it is therefore omitted.

[6] $\#A$ denotes the number of elements in the finite set $A$.
[7] We define $\uparrow^{-1} = \downarrow$ and $\downarrow^{-1} = \uparrow$.

**Claim 4.6.** $w$ satisfies exactly one literal $A_{i,j}$ in every disjunction $C_i$ of $S$ iff $d_w$ is a very safe d–assignment for $H'(S)$.

The proof above shows that for the class of logic programs with an empty set of function symbols, VERY–SIMPLE is already NP–complete.

## 5 Conclusion

In this paper, we have proved that the problem of deciding, whether a given logic program is simple, is NP–complete. A simple logic program $H$ can be transformed into an attribute grammar $G$. In general $G$ has a semantic domain, which is not a free term algebra, i.e., $G$ is not free. We have defined the more restricted class of very simple logic programs. Very simple logic programs can be transformed into equivalent free attribute grammars. We proved that the problem of deciding, whether a given logic program is very simple, is NP–complete as well.

## References

[Alb89] H. Alblas. Attribute evaluation methods. *Memoranda Informatica,* 89-20, University of Twente, Enschede, 1989.

[AM91] H. Alblas and B. Melichar. *International Conference SAGA, Prague,* volume 545 of *Lect. Notes Comput. Sci.* Springer–Verlag, Juni 1991.

[Apt96] K. Apt. *From Logic Programming to Prolog.* Prentice-Hall, January 1996.

[Cla78] K. L. Clark. Predicate logic as a computational formalism. Report Res. Mon. 79/59, Imperial College, London, 1978.

[Coo71] S.A. Cook. The complexity of theorem–proving procedures. In *Proc. 3rd IEEE Symp. on the Foundations of Computer Science*, pages 151–158, 1971.

[DJ90] P. Deransart and M. Jourdan. *Attribute Grammars and Their Applications, International Conference WAGA, Paris*, volume 461 of *Lect. Notes Comput. Sci.* Springer–Verlag, September 1990.

[DM85] P. Deransart and J. Maluszynski. Relating logic programs and attribute grammars. *J. Logic Programming*, 2:119–155, 1985.

[DM93] P. Deransart and J. Maluszynski. *A Grammatical View of Logic Programming.* MIT Press, Cambridge MA, 1993.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–completeness.* Freeman, San Francisco, 1979.

[GTWW77] J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24:68–95, 1977.

[Isa91] T. Isakowitz. Can we transform logic programs into attribute grammars. *Journal of Theoretical Informatics and Applications*, 15:499–543, 1991.

[Knu68] D.E. Knuth. Semantics of context-free languages. *Math. Systems Theory*, 2:127–145, 1968.

[Kow74] R. Kowalski. Predicate logic as a programming language. *Inform. Process. Letters*, 74:569–574, 1974.

[Sch78] T.J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th Ann. ACM Symp. on Theory of Computing*, Association for Computing Machinery, pages 216–226, New York, 1978.

*Received May, 1997*

# Economical Transformations of Phrase-Structure Grammars to Scattered Context Grammars

Alexander Meduna*

**Abstract**

This paper presents a transformation that converts any phrase-structure grammar, $H$, in Penttonen normal form to an equivalent scattered context grammar whose size differs from the size of $H$ quite insignificantly; specifically, $G$ has only five more nonterminals, four more context-dependent productions, and one more context-free production than $H$. An analogical result holds for Kuroda normal form, too.

## 1 Introduction

Transformations that convert grammars of one type to equivalent grammars of another type are central to the formal language theory. Initially, this theory designed these transformations regardless of the size of the output grammars. Because this size was usually enormously greater than the size of the input grammars, transformations of this kind were of no use in practice. Therefore, at present, the formal language theory modifies these transformations to produce the equivalent output grammars as small as possible (see [2], [3], [5], [6], [7], and Chapter 4 in [1], including references therein).

Following this line of the language theory, the present paper explains how to transform any phrase-structure grammar to an equivalent scattered context grammar whose size differs from the size of the input phrase-structure grammar quite insignificantly. More precisely, it converts any phrase-structure grammar, $H$, in Penttonen normal form to an equivalent scattered context grammar, $G$, so $G$ has only five more nonterminals, four more context-dependent productions, and one more context-free production than $H$. Then, this paper states an analogical result in terms of Kuroda normal form.

## 2 Definitions

This paper assumes that the reader is familiar with the language theory (see Chapter 0 in [1]).

*Computing Center at Technical University of Brno, Udolni 19, Brno 60200, Czech Republic

## Basic Notions

For a set, $Q$, *card*$(Q)$ denotes the cardinality of $Q$. Set

$$\mathbf{N} = \{1, 2, \ldots\} \text{ and } \mathbf{I} = \{0, 1, 2, \ldots\}.$$

Let $V$ be an alphabet. $V^*$ represents the free monoid generated by $V$ under the operation of concatenation. The unit of $V^*$ is denoted by $\varepsilon$. Set $V^+ = V^* - \{\varepsilon\}$; algebraically, $V^+$ is the free semigroup generated by $V$ under the operation of concatenaton.

For $w \in V^*, |w|$ denotes the lenght of $w$. Set

$subword(w) = \{x : x \in V^* \text{ and } x \text{ is a subword of } w\}$;
$prefix(w) = \{x : x \text{ is a prefix } w\}$;
$suffix(w) = \{x : x \text{ is a suffix } w\}$;
$alph(w) = subword(w) \cap V$.

For $a \in V$ and $w \in V^*$; *occur*$(a, w)$ denotes the number of occurrences of $a's$ in $w$.

## Grammars

A *phrase-structure grammar* is a quadruple

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

where $N$ and $T$ are alphabets such that $N \cap T = \emptyset$. Symbols in $N$ are referred to as *nonterminals* while symbols in $T$ are *terminals*. $N$ contains $S$ - the *start symbol* of $G$. $P$ is a finite set of productions of the form

$$x \longrightarrow y$$

where $x, y \in V^*$ so $alph(x) \cap N \neq \emptyset$. If $x \longrightarrow y \in P$ and $u_i \in (N \cup T)^*$ for $i = 1, 2$, then

$$u_1 x u_2 \Longrightarrow u_1 y u_2;$$

whenever this paper needs to specify the subword, $x$, rewritten during $u_1 x u_2 \Longrightarrow u_1 y u_2$, it underlines it as

$$u_1 \underline{x} u_2 \Longrightarrow u_1 y u_2.$$

Observe that $\Longrightarrow$ represents a relation on $(N \cup T)^*$. Let $\Longrightarrow^m$ denote the $m$-fold product of $\Longrightarrow$, where $m \in \mathbf{I}$. Furthermore, $\Longrightarrow^+$ and $\Longrightarrow^*$ denote the transitive closure of $\Longrightarrow$ and the transitive and reflexive closure of $\Longrightarrow$, respectively. The *language generated by* $G, L(G)$, is defined as

$$L(G) = \{w \in T^* : S \Longrightarrow^* w\}.$$

Let $G = (N, P, S, T)$ be a phrase-structure grammar. $G$ is in *Penttonen normal form* if $P$ has only these two kinds of productions

$$AB \longrightarrow AC \text{ where } A, B, C \in N, \text{ and}$$

$$A \longrightarrow x \text{ where } A \in N \text{ and } x \in NN \cup T \cup \{\varepsilon\}.$$

A *scattered context grammar* is a quadruple

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

where $N, T, S$ have the same meaning as in a phrase-structure grammar, and P is a finite set of productions of the form

$$(A_1, A_2, \ldots, A_n) \longrightarrow (x_1, x_2, \ldots, x_n),$$

where $n \in \mathbf{N}$, and for all $i = 1, 2, \ldots, n, A_i \in N$ and $x_i \in (N \cup T)^*$.

Let $G = (N, P, S, T)$ be a scattered context grammar, and let $v, w \in (N \cup T)^*$. If for some $n \in \mathbf{N}$

A. $(A_1, A_2, \ldots, A_n) \longrightarrow (x_1, x_2, \ldots, x_n) \in P$, and

B. $v = u_1A_1u_2A_2 \ldots u_nA_nu_{n+1}$ and $w = u_1x_1u_2x_2 \ldots u_nx_nu_{n+1}$ with $u_i \in (N \cup T)^*$ for $i = 1, 2, \ldots, n+1$,

then $v$ *directly derives* $w$ in $G$, symbolically written as

$$v \Longrightarrow w.$$

Express $v \Longrightarrow w$ as $u_1A_1u_2A_2 \ldots u_nA_nu_{n+1} \Longrightarrow u_1x_1u_2x_2 \ldots u_nx_nu_{n+1}$.

Whenever this paper needs to specify the nonterminals, $A_1$ through $A_n$, rewritten during this direct derivation, it underlines them as

$$u_1\underline{A_1}u_2\underline{A_2} \ldots u_n\underline{A_n}u_{n+1} \Longrightarrow u_1x_1u_2x_2 \ldots u_nx_nu_{n+1}.$$

For $m \in \mathbf{I}, \Longrightarrow^m$ denotes the $m$-fold product of $\Longrightarrow$ . Furthermore, $\Longrightarrow^+$ and $\Longrightarrow^*$ denote the transitive closure of $\Longrightarrow$ and the transitive and reflexive closure of $\Longrightarrow$, respectively. The *language generated by* $G, L(G)$, is defined as

$$L(G) = \{w \in T^* : S \Longrightarrow^* w\}.$$

Recall that phrase-structure grammars, phrase-structure grammars in Penttonen normal form, and scattered context grammars have the same generative power. Indeed, they all characterize the family of recursively enumerable languages (see [1], [4], and [7]).

## Context-Dependent and Context-Free Productions

Let $G$ be a grammar, and let $P$ be $G's$ set of production. In this paper, we separate $P$ into two disjoint subsets - the set of context-free productions, *ContextFree*$(P)$, and the set of context-dependent productions, *ContextDependent*$(P)$. A production, $p \in P$, belongs to *ContextFree*$(P)$ if and only if the left-hand side of $p$ consists of one nonterminal; otherwise, $p$ belongs to *ContextDependent*$(P)$.

Specifically, if $G = (N, P, S, T)$ is a phrase-structure grammar, then

$ContextFree(P) = \{A \longrightarrow x : A \longrightarrow x \in P$ and $A \in N\}$, and
$ContextDependent(P) = P - ContextFree(P)$.

If $G = (N, P, S, T)$ is a scattered context grammar, then

$ContextFree(P) = \{(A) \longrightarrow (x) : (A) \longrightarrow (x) \in P\}$, and
$ContextDependent(P) = P - ContextFree(P)$.

Equivalently, if $G = (N, P, S, T)$ is a scattered context grammar, then

$(A_1, \ldots, A_n) \longrightarrow (x_1, \ldots, x_n) \in ContextDependent(P)$ if and only if $n \geq 2$; otherwise, $(A_1, \ldots, A_n) \longrightarrow (x_1, \ldots, x_n) \in ContextFree(P)$.

## 3 Results

This section demonstrates that for every phrase-structure grammar, $H = (N', P', S', T)$, in Penttonen normal form, there exists an equivalent scattered context grammar, $G = (N, P, S, T)$, that satisfies

A. $L(G) = L(H)$;

B. $card\,(N) \leq card(N') + 5$;

C. $card(ContextDependent\ (P)) = card(ContextDependent(P')) + 4$;

D. $card(ContextFree(P)) = card(ContextFree(P')) + 1$.

**Theorem 1** *Let $H = (M, R, S, T)$ be a phrase-structure grammar in Penttonen normal form. Then, there exists a scattered context grammar, $G = (N, P, E, T)$, that satisfies*

*A.* $L(G) = L(H)$;

*B.* $card(M) = card(N) + 5$;

*C.* $card(ContextDependent(P)) = card(ContextDependent(R)) + 4$;

*D.* $card(ContextFree(P)) = card(ContextFree(R)) + 1$.

**Proof.:** Let

$$H = (M, R, S, T)$$

be a phrase-structure grammar in Penttonen normal form, where $M$ denotes $H's$ alphabet of nonterminals, $R$ denotes its set of productions, $S$ is its start symbol, and T denotes its alphabet of terminals. Without loss of generality, assume that

$$\{E, F, [, ], \$\} \cap M = \emptyset.$$

In the following, we describe how to construct a scattered context grammar, $G$, such that $L(G) = L(H)$ and $G$ satisfies the conditions of Theorem 1.
Define the scattered context grammar

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

where
$N = \{E, F, [, ], \$\} \cup M$
and

$$\begin{array}{rcl} P & = & \{E \longrightarrow F[]F[F]S[], \\ & & (F, [, ], F, F) \longrightarrow (F, \varepsilon, \varepsilon, F, F), \\ & & (F, F, \$, F) \longrightarrow (F, F, \varepsilon, F), \\ & & (F, [, ], F, F, [, ], [) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, F, [, ]F, F[), \\ & & (F, [, ], F, F, [, ]) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)\} \\ & \cup & \{(A, B) \longrightarrow (A[, ]C) : AB \longrightarrow AC \in R \text{ with } A, B, C \in M\} \\ & \cup & \{(A) \longrightarrow (x) : A \longrightarrow x \in R, A \in M, x \in MM\} \\ & \cup & \{(A) \longrightarrow (\$a) : A \longrightarrow x \in R, A \in M, a \in T \cup \{\varepsilon\}\} \end{array}$$

Observe that

$card(M) = card(N) + 5;$
$card(ContextDependent(P)) = card(ContextDependent(R)) + 4;$
$card(ContextFree(P)) = card(ContextFree\ (R)) + 1.$

The proof that $L(G) = L(H)$ is based on the following.
By productions in

$$\begin{array}{rl} & \{(A, B) \longrightarrow (A[, ]C) : AB \longrightarrow AC \in R \text{ with } A, B, C \in M\} \\ \cup & \{(A) \longrightarrow (x) : A \longrightarrow x \in R, A \in M, x \in MM\} \\ \cup & \{(A) \longrightarrow (\$a) : A \longrightarrow x \in R, A \in M, a \in T \cup \{\varepsilon\}\}, \end{array}$$

$G$ simulates $H's$ derivations. By productions in

$$\begin{array}{l} \{E \longrightarrow F[]F[F]S[], \\ (F, [, ], F, F) \longrightarrow (F, \varepsilon, \varepsilon, F, F), \\ (F, F, \$, F) \longrightarrow (F, F, \varepsilon, F), \\ (F, [, ], F, F, [, ], [) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, F, [, ]F, F[), \\ (F, [, ], F, F, [, ]) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)\}, \end{array}$$

$G$ verifies that the simulation was performed properly.
Next, this proof establishes several claims to demonstrate $L(G) = L(H)$ in a rigorous way.

**Claim 1** Let

$$E \Longrightarrow^+ u \Longrightarrow^+ t$$

in $G$, where $u \in (N \cup T)^*$ and $t \in T^*$. Then

$$u = wFx,$$

where $w \in T^*$ and $x \in ((N - \{E\}) \cup T)^*$ with $occur(F, v) = 2$.

**Proof. of Claim 1:** Consider any derivation,

$$E \Longrightarrow^+ u \Longrightarrow^+ t$$

in $G$, where $u \in (N \cup T)^*$ and $t \in T^*$. Examine $P$ to see that

$$occur(E, u) = 0 \text{ and } occur(F, u) = 3.$$

Express $u$ as

$$u = wFv,$$

where $w, v \in (N \cup T)^*$ and $occur(F, v) = 2$. Notice that

$$alph(w) \cap \{E, F\} = \emptyset.$$

By contradiction, prove that

$$alph(w) \cap (N - \{E, F\}) = \emptyset.$$

Assume that $alph(w) \cap (N - \{E, F\}) \neq \emptyset$. Consider

$(F, [, ], F, F) \longrightarrow (F, \varepsilon, \varepsilon, F, F)$,
$(F, F, \$, F) \longrightarrow (F, F, \varepsilon, F)$,
$(F, [, ], F, F, [, ], [) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, F, [, ]F, F[)$, and
$(F, [, ], F, F, [, ]) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)$.

As $occur(F, v) = 2$, none of these productions can rewrite any symbol in $w$. Because $alph(w) \cap (N - \{E, F\}) \neq \emptyset$, for every $y$ such that $wFv \Longrightarrow^+ y$,

$$alph(y) \cap (N - \{E, F\}) \neq \emptyset,$$

which contradicts $wFv \Longrightarrow^+ t$ with $t \in T^*$. Thus,

$$alph(w) \cap (N - \{E, F\}) = \emptyset.$$

Consequently,

$$u = wFv,$$

where $w \in T^*$ and $v \in ((N - \{E\}) \cup T)^*$ with $occur(F, v) = 2$.
Therefore, Claim 1 holds.

□

Define the morphism, $\alpha$, from $((N - \{E\}) \cup T)^*$ to $(\{[, ], F, \#\} \cup T)^*$ as

$\alpha(Y) = Y$ for all $Y \in (\{[, ], F\} \cup T)$, and
$\alpha(X) = \#$ for all $X \in (\{\$\} \cup M)$.

**Claim 2** Let

$$E \Longrightarrow^{+} wFvFxFy \Longrightarrow^{+} u$$

in $G$, where $w, u \in T^{*}$, and $v, x, y \in ((N - \{E, F\}) \cup T)^{*}$. Then,

$$v \in \{[, ]\}^{*}.$$

**Proof of Claim 2:** Let

$$E \Longrightarrow^{+} wFvFxFy \Longrightarrow^{+} u$$

in $G$, where $w, u \in T^{*}$, and $v, x, y \in ((N - \{E\}) \cup T)^{*}$. By contradiction we next prove that

$$\{\#\}^{+} \cap subword(\alpha(v)) = \emptyset.$$

Assume that

$$\{\#\}^{+} \cap subword(\alpha(v)) \neq \emptyset.$$

Examine

$(F, [, ], F, F) \longrightarrow (F, \varepsilon, \varepsilon, F, F)$,
$(F, F, \$, F) \longrightarrow (F, F, \varepsilon, F)$,
$(F, [, ], F, F, [, ], [) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, F, [, ]F, F[)$, and
$(F, [, ], F, F, [, ]) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)$.

The form of these productions and $\{\#\}^{+} \cap$ *subword*$(\alpha(v)) \neq \emptyset$ imply that every word that $G$ derives from $wFvFxFy$ contains $\#$. Specifically,

$$\{\#\}^{+} \cap subword(\alpha(u)) \neq \emptyset,$$

which contradicts $u \in T^{*}$. Hence,

$$\{\#\}^{+} \cap subword(\alpha(v)) = \emptyset.$$

Thus,

$$\$ \notin alph(v).$$

Consider

$$\{(A) \longrightarrow (\$a) : A \longrightarrow x \in R, A \in M, a \in T \cup \{\varepsilon\}\}.$$

Observe that this set includes all productions containing symbols from $T \cup \{\$\}$. Therefore, as $\$ \notin alph(v)$,

$$T \cap alph(v) = \emptyset.$$

Consequently,

$$\alpha(v) \in \{[, ]\}^{*}.$$

□

**Claim 3** Let

$$E \Longrightarrow^+ y$$

in $G$, where $y \in ((N - \{E\}) \cup T)^*$. Then,

$$][ \notin subword(y).$$

**Proof of Claim 3:** Let

$$E \Longrightarrow^+ y$$

in $G$, where $y \in ((N - \{E\}) \cup T)^*$. All productions containing [ or ] are included in this set

$$\begin{array}{ll} & \{E \longrightarrow F[]F[F]S[], \\ & (F, [,], F, F) \longrightarrow (F, \varepsilon, \varepsilon, F, F), \\ & (F, F, \$, F) \longrightarrow (F, F, \varepsilon, F), \\ & (F, [,], F, F, [,], [) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, F, [,]F, F[), \\ & (F, [,], F, F, [,]) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)\} \\ \cup & \{(A, B) \longrightarrow (A[,]C) : AB \longrightarrow AC \in R \text{ with } A, B, C \in M\}. \end{array}$$

By $E \longrightarrow F[]F[F]S[], G$ generates $F[]F[F]S[]$. Notice that

$$][ \notin subword(F[]F[F]S[]).$$

By using, the productions in

$$\{(A, B) \longrightarrow (A[,]C) : AB \longrightarrow AC \in R \text{ with } A, B, C \in M\},$$

$G$ can generate no word containing ][. Finally, consider the other four productions:

$(F, [,], F, F) \longrightarrow (F, \varepsilon, \varepsilon, F, F)$,
$(F, F, \$, F) \longrightarrow (F, F, \varepsilon, F)$,
$(F, [,], F, F, [,], [) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, F, [,]F, F[)$, and
$(F, [,], F, F, [,]) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)\}$.

If the current sentential form does not contain ][, then every word directly derived from this sentential form by using any of these productions does not contain ][ either. Thus,

$$][ \notin subword(y).$$

A formal version of this proof is left to the reader.

□

**Claim 4** Let

$$E \Longrightarrow^+ wFvFxFy \Longrightarrow^+ u$$

in $G$, where $w, u \in T^*; v \in \{[,]\}^*$; and $x, y \in ((N - \{E, F\}) \cup T)^*$. Then,

$$v \in \{[\}^*\{]\}^*.$$

**Proof of Claim 4:** This claim follows from Claims 2 and 3.

□

**Claim 5** Let

$$E \Longrightarrow^+ wFvFxFy \Longrightarrow^+ u$$

in $G$, where $w, u \in T^*; v \in \{[\}^*\{]\}^*$; and $x, y \in ((N - \{E, F\}) \cup T)^*$. Then,

$$v \in \{[^i]^i : i \in \mathbf{N}\}.$$

**Proof of Claim 5:** Consider

$$E \Longrightarrow^+ wFvFxFy \Longrightarrow^+ u$$

in $G$, where $w, u \in T^*; v \in \{[\}^*\{]\}^*$; and $x, y \in ((N - \{E, F\}) \cup T)^*$.
By contradiction, we next demonstrate

$$|v| \geq 1.$$

Assume that

$$|v| = 0.$$

Examine $P$ to see that at this point, $wFvFxFy$ derives no word over $T$, which contradicts $wFvFxFy \Longrightarrow^+ u$ with $u \in T^*$. Thus, $|v| \geq 1$.
Let $i, j \in \mathbf{I}$ such that $i < j$ and $|v| = i + j$. By contradiction, we now prove

$$[^i]^j \neq \alpha(v).$$

Assume that

$$[^i]^j = \alpha(v).$$

Examine $P$ to see that under this assumption,

$$] \in alph(u).$$

Thus,

$$u \notin T^*,$$

which contradicts $u \in T^*$. Hence,

$$[^i]^j \neq \alpha(v).$$

Analogously, prove $[^i]^j \neq \alpha(v)$, where $i, j \in \mathbf{I}$ so $iFj$ and $|v| = i + j$.
Thus, for any $i, j \in \mathbf{I}$ such that $i \neq j$,

$$[^j]^i \neq \alpha(v).$$

Therefore, Claim 5 holds.

□

**Claim 6** Let

$$E \Longrightarrow^{+} wFvFxFy \Longrightarrow^{+} u$$

in $G$, where $w, u \in T^*; v \in \{[\}^i\{]\}^i$ for some $i \in \mathbf{N}$; and $x, y \in ((N - \{E, F\}) \cup T)^*$. Then,

$$x \in (\{\$\} \cup T \cup M)^*.$$

**Proof of Claim 6:** In $G$, consider any derivation that has this form

$$E \Longrightarrow^{+} wFvFxFy \Longrightarrow^{+} u$$

in $G$, where $w, u \in T^*; v \in \{[\}^i\{]\}^i$ for some $i \in \mathbf{N}$; and $x, y \in ((N - \{E, F\}) \cup T)^*$. By contradiction, demonstrate

$$\{[, ]\} \cap alph(x) = \emptyset.$$

Assume that

$$\{[, ]\} \cap alph(x) \neq \emptyset.$$

Examine $P$; specifically,

$(F, [, ], F, F, [, ], [) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, F, [, ]F, F[)$, and
$(F, [, ], F, F, [, ]) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)$.

In this case,

$$\{[, ]\} \cap alph(u) \neq \emptyset,$$

which contradicts $u \in T^*$. Hence

$$\{[, ]\} \cap alph(x) = \emptyset.$$

Therefore, $x \in ((N - \{E, F, [, ]\}) \cup T)^*$, so

$$x \in (\{\$\} \cup T \cup M)^*.$$

**Claim 7** Let

$$E \Longrightarrow^{+} wFvFxFy \Longrightarrow^{+} u$$

in $G$, where $w, u \in T^*$, and $v \in \{[\}^i\{]\}^i$ for some $i \in \mathbf{N}, x \in (\{\$\} \cup T \cup M)^*, y \in ((N - \{E, F\}) \cup T)^*$. Then,

$$y \in (K \cup K\{[^i]^i : i \in \mathbf{N}\}K)^*\{[]\}$$

with

$$K = ((N - \{E, F, ], [\}) \cup T).$$

**Proof of Claim 7:** Examine $P$; specifically,

$(F, [, ], F, F) \longrightarrow (F, \varepsilon, \varepsilon, F, F)$,
$(F, F, \$, F) \longrightarrow (F, F, \varepsilon, F)$,
$(F, [, ], F, F, [, ], [) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, F, [, ]F, F[)$, and

$(F, [, ], F, F, [, ]) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)\}$.

Based on this examination, observe that this claim follows from Claims 5 and 6 (a formal verification of this observation is left to the reader).

$\square$

Define the morphism, $\beta$, from $((N - \{E\}) \cup T)^*$ to $(N - \{[, ], \$, E, F\}) \cup T)^*$ as

$\beta(Y) = \varepsilon$ for all $Y \in \{[, ], \$, F\}$, and
$\beta(X) = X$ for all $X \in (N - \{[, ], \$, F\}) \cup T$.

**Claim 8** Let

$$E \Longrightarrow^m w \Longrightarrow^* z$$

in $G$, where $m \in \{\mathbf{N}, z \in T^*, w \in (N \cup T)^*$. Then,

$$S \Longrightarrow^* \beta(w)$$

in $H$.

**Proof of Claim 8:** This claim is established by induction on $m = 0, \ldots$.

*Basis:*
Let $m = 1$. That is,

$$E \Longrightarrow F[]F[F]S[] \Longrightarrow^* v$$

in $G$. Notice that $\beta([]F[F]S[]) = S$. As

$$S \Longrightarrow^0 S$$

in $H$, the basis holds.

*Induction Hypothesis:*
Suppose that there exists $j \in \mathbf{N}$ such that Claim 8 holds for every $m \leq j$.

*Induction Step:*
Let

$$E \Longrightarrow^{j+1} w \Longrightarrow^* z$$

in $G$, where $z \in T^*$ and $w \in (N \cup T)^*$. Based on Claims 1 through 7, express this derivation as

$$S \Longrightarrow^j tFvFxFy \Longrightarrow w \Longrightarrow^+ z$$

in $G$, where $z, t \in T^*$, and $v \in \{[]\}^i\{[]\}^i$ for some $i \in \mathbf{N}, x \in (\{\$\} \cup T \cup M)^*$, $y \in (K \cup K\{([^i]^i : i \in \mathbf{N}\}K)^*\{[]\}$ with $K = ((N - \{E, F, ], []\}) \cup T)$. Let $p$ be the production that $G$ uses to make $tFvFxFy \Longrightarrow w$. By the induction hypothesis,

$$S \Longrightarrow^* \beta(tFvFxFy)$$

in $H$. Next, this proof considers all possible forms of $p$. Before this consideration notice that $p$ surely differs from $E \longrightarrow F[]F[F]S[]$ because $E$ does not appear in $tFvFxFy$.

1. Assume that $p$ has this form

$$(A, B) \longrightarrow (A[,]C),$$

where $A, B, C \in N$. Because $z, t \in T^*, v \in \{[)\}^i\{]\}^i$ for some $i \in \mathbf{N}, x \in (\{\$\} \cup T \cup M)^*$, and $y \in (K \cup K\{[^i]^i : i \in \mathbf{N}\}K)^*\{[]\}$ with $K = ((N - \{E, F, ], [\}) \cup T)$, the previous claims imply that

$$u = tFvFxFy'\underline{A}y''\underline{B}y''' \text{ and } w = tFvFxFy'A[y'']Cy''',$$

for some $y' \in \mathit{prefix}(y), y''' \in \mathit{suffix}(y)$, and

$$y'' \in (\{[^i]^i : i \in \mathbf{I}\}.$$

Thus,

$$\beta(y'') = \varepsilon.$$

As $(A, B) \longrightarrow (A[,]C) \in P$,

$$AB \longrightarrow AC \in R.$$

Notice that

$$\beta(tFvFxFy')\underline{A}\beta(y'')\underline{B}\beta(y''') \Longrightarrow \beta(tFvFxFy')A\beta(y'')C\beta(y''')$$

in $H$. Because $\beta(tFvFxFy')A\beta(y'')C\beta(y''') = \beta(w)$,

$$S \Longrightarrow^* \beta(w)$$

in $H$.

2. Assume that p has this form

$$(A) \longrightarrow (x).$$

$A \in M, x \in MM$. At this point,

$$u = tFvFxFy'\underline{A}y'' \text{ or } u = tFvFx'\underline{A}x''Fy.$$

Assume that $u = tFvFxFy'\underline{A}y''$. At this point,

$$w = tFvFxFy'xy''.$$

As $(A) \longrightarrow (x) \in P$,

$$A \longrightarrow x \in R.$$

Notice that $H$ makes

$$\beta(tFvFxFy')\underline{A}\beta(y'') \Longrightarrow \beta(tFvFxFy')x\beta(y'')$$

by using $A \longrightarrow x$. Because $\beta(tFvFxFy')x\beta(y'') = \beta(w)$,

$$S \Longrightarrow^* \beta(w)$$

in $H$. Analogically, prove that $S \Longrightarrow^* \beta(w)$ in the case when $u = tFvFx'\underline{A}x''Fy$.

3. Assume that $p$ has this form

$$(A) \longrightarrow (\$a),$$

where $A \in M$ and $a \in T \cup \{\varepsilon\}$. Observe that

either $x = x'Ax''$ so $u = tFvFx'\underline{A}x''Fy$
where $x' \in prefix(x)$ and $x'' \in suffix(x)$

or $y = y'\underline{A}y''$ so $u = tFvFxFy'\underline{A}y''$
where $y' \in prefix(y)$ and $y'' \in suffix(y)$.

Assume that $u = tFvFx'\underline{A}x''Fy$. At this point,

$$w = tFvFx'\$ax''Fy.$$

As $(A) \longrightarrow (\$a) \in P$,

$$A \longrightarrow a \in R.$$

Notice that H makes

$$\beta(tFvFx'\$)\underline{A}\beta(x''Fy) \Longrightarrow \beta(tFvFx'\$)a\beta(x''Fy).$$

Because $\beta(tFvFx'\$)a\beta(x''Fy) = \beta(w)$,

$$S \Longrightarrow^* \beta(w)$$

in $H$.
Analogously, establish $S \Longrightarrow^* \beta(w)$ under the assumption that $u = tFvFxFy'\underline{A}y''$.

4. Assume that $p$ is a production from

$\{(F, [, ], F, F) \longrightarrow (F, \varepsilon, \varepsilon, F, F),$
$(F, F, \$, F) \longrightarrow (F, F, \varepsilon, F),$
$(F, [, ], F, F, [, ], [) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, F, [, ]F, F[),$
$(F, [, ], F, F, [, ]) \longrightarrow (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)\}.$

Then,

$$\beta(tFvFxFy) = \beta(w)$$

By the induction hypothesis,

$$S \Longrightarrow^* \beta(w)$$

in $H$.
Thus, Claim 8 holds.

□

**Claim 9**

$$L(G) \subseteq L(H).$$

**Proof of Claim 9:** By Claim 6, if

$$E \Longrightarrow^{+} v$$

in $G$ with $v \in T^*$, then

$$S \Longrightarrow^{*} v$$

in $H$. Therefore, Claim 9 holds.

□

**Claim 10** Let

$$S \Longrightarrow^{j} u \Longrightarrow^{*} z$$

in $H$, where $j \in \mathbf{I}, z \in T^*$, and $u \in (M \cup T)^*$. Then,

$$E \Longrightarrow^{+} wFvFxFy$$

in $G$, where $w \in T^*, v \in \{[]\}^i\{]\}^i$ for some $i \in \mathbf{N}, x \in (\{\$\} \cup T)^*, y \in ((N - \{E, F\}) \cup T)^*$, so that

$$u \in \beta(wFvFxFy).$$

**Proof of Claim 10:** This claim is established by induction on $j = 0, \ldots$.

*Basis:*
Let $j = 0$; that is,

$$S \Longrightarrow^{0} S \Longrightarrow^{*} z$$

in $H$. Notice that $G$ makes

$$E \Longrightarrow F[]F[F]S[]$$

by using $E \longrightarrow F[]F[F]S[]$, and $S \in \beta(F[]F[F]S[])$. Thus, the basis holds.

*Induction Hypothesis:*
Suppose that

$$S \Longrightarrow^{i} u \Longrightarrow^{*} z$$

in $H$, where $z \in T^*$, $u \in (M \cup T)^*$, for all $i = 0, \ldots, j$, for some $j \in \mathbf{I}$.

*Induction Step:*
Let

$$S \Longrightarrow^{j+1} u \Longrightarrow^{*} z$$

in $H$, where $z \in T^*$ and $u \in (M \cup T)^*$. Express this derivation as

$$S \Longrightarrow^{j} u \Longrightarrow t \Longrightarrow^{*} z$$

in $H$, where $t \in (M \cup T)^*$ and $u \Longrightarrow t$ is made according to $p \in R$. By the induction hypothesis,

$$E \Longrightarrow^{+} wFvFxFy$$

in $G$, where $w \in T^*, v \in \{[]\}^i\{]\}^i$ for some $i \in \mathbf{N}, x \in (\{\$\} \cup T)^*, y \in ((N - \{E, F\}) \cup T)^*$, so $u \in \beta(wFvFxFy)$. Let $H$ make $u \Longrightarrow t$ by using a production, $p \in R$. Next, this proof considers all possible forms of $p$.

1. Assume that $p$ has this form

$$AB \longrightarrow AC.$$

Express $u \Longrightarrow t$ in $H$ as

$$u'\underline{AB}u'' \Longrightarrow u'ACu'',$$

where $u'ABu'' = u$ and $u'ACu'' = t$. As $w \in T^*, v \in \{[\}^i\{]\}^i$ for some $i \in \mathbf{N}, x \in (\{\$\} \cup T)^*, y \in ((N - \{E, F\}) \cup T)^*, u \in \beta(wFvFxFy)$, and

$$AB \in subword(y).$$

Assume that

$$y = y'A[^k]^k By'',$$

where $u' \in \beta(wFvFxFy'), k \in \mathbf{I}, u'' \in \beta(y'')$.
As $AB \longrightarrow AC \in R$,

$$(A, B) \longrightarrow (A[,]C) \in P.$$

Thus,

$$wFvFxFy'A[^k]^k By'' \Longrightarrow wFvFxFy'A[^{k+1}]^{k+1}Cy''$$

in $G$. Therefore,

$$E \Longrightarrow^+ wFvFxFy'A[^{k+1}]^{k+1}Cy''$$

in $G$ so

$$u \in \beta(wFvFxFy'A[^{k+1}]^{k+1}Cy'').$$

2. Assume that $p$ has this form

$$A \longrightarrow BC.$$

Because $w \in T^*, v \in \{[\}^i\{]\}^i$ for some $i \in \mathbf{N}, x \in (\{\$\} \cup T)^*, y \in ((N - \{E, F\}) \cup T)^*$, and $u \in \beta(wFvFxFy)$,

$$A \in subword(y).$$

Express $y$ as

$$y = y'Ay''.$$

As $A \longrightarrow BC \in R$,

$$(A) \longrightarrow (BC) \in P.$$

Thus,

$$wFvFxFy'Ay'' \Longrightarrow wFvFxFy'BCy''$$

in $G$. Therefore,

$$E \Longrightarrow^+ wFvFxFy'BCy''$$

in $G$ so

$$u \in \beta(wFvFxFy'BCy'').$$

3. Assume that $p$ has this form

$$A \longrightarrow a.$$

As $w \in T^*, v \in \{[\}^i\{]\}^i$ for some $i \in \mathbf{N}, a \in (\{\varepsilon\} \cup T), x \in (\{\$\} \cup T)^*, y \in ((N - \{E, F\}) \cup T)^*$, and $u \in \beta(wFvFxFy)$, we have

$$A \in subword(y).$$

Express $y$ as

$$y = y'Ay''.$$

As $A \longrightarrow a \in R$,

$$(A) \longrightarrow (\$a) \in P.$$

Thus,

$$wFvFxFy'Ay'' \Longrightarrow wFvFxFy'\$ay''$$

in $G$. Therefore,

$$E \Longrightarrow^+ wFvFxFy'\$ay''$$

in $G$ so

$$u \in \beta(wFvFxFy'\$ay'').$$

Therefore, Claim 10 holds.

□

**Claim 11**

$$L(H) \subseteq L(G).$$

**Proof of Claim 11:**
By Claim 10, if

$$S \Longrightarrow^* v$$

in $H$, where $v \in T^*$, then

$$S \Longrightarrow^* v$$

in $G$. Therefore, Claim 11 holds.

□

By Claims 9 and 11,

$$L(G) = L(H).$$

To summarize the proof,

A. $L(G) = L(H)$;

B. $card(M) = card(N) + 5$;

C. $card(ContextDependent(P)) = card(ContextDependent(R)) + 4$;

D. $card(ContextFree(P)) = card(ContextFree(R)) + 1$.

Thus, Theorem 1 holds.

□

In its conclusion, this paper points out that the previous theorem also holds for phrase-structure grammars in Kuroda normal form (see [2]). Recall that a phrase-structure grammar, $G = (N, P, S, T)$, is in *Kuroda normal form* if $P$ has only these two kinds of productions

$$AB \longrightarrow CD \text{ where } A, B, C, D \in N, \text{ and}$$

$$A \longrightarrow x \in R, \text{ where } A \in N \text{ and } x \in NN \cup T \cup \{\varepsilon\}$$

**Theorem 2** Let $H = (N', P', S', T)$ be a phrase-structure grammar in Kuroda normal form. Then, there exists a scattered context grammar, $G = (N, P, S, T)$, that satisfies

A. $L(G) = L(H)$;

B. $card(N) \leq card(N') + 5$;

C. $card(ContextDependent(P)) \leq card(ContextDependent(P')) + 4$;

D. $card(ContextFree(P)) \leq card(ContextFree(P')) + 1$.

**Proof.:** Prove this theorem by analogy with the proof of Theorem 1.

□

*Acknowledgement:* The author thanks the anonymous referee for useful comments concerning the first version of this paper.

## References

[1] Dassow, J. and Paun, G.: Regulated Rewriting in Formal Language Theory. Springer, New York, 1989.

[2] Kuroda, S. Y. : "Classes of Languages and Linear Bounded Automata," Inform. Control 7, 207 - 223, 1964.

[3] Meduna, A. : "Syntactic Complexity of Scattered Context Grammars " Acta Informatica 32, 285 - 298, 1995.

[4] Meduna, A. : "A Trivial Method of Characterizing the Family of Recursively Enumerable Languages by Scattered Context Grammars," EATCS Bulletin 56 (1995), 104 -106.

[5] Meduna, A. : "Four-Nonterminal Scattered Context Grammars Characterize the Family of Recursively Enumerable Languages," International Journal of Computer Mathematics 63, 67-83, 1997.

[6] Paun, Gh. : "Six Nonterminals are Enough for Generating each R.E. Language by a Matrix Grammar," International Journal of Computer Mathematics 15, 23-37, 1993.

[7] Penttonen, M. : "One-Sided and Two-Sided Context in Formal Grammars," Inform. Control 25, 371 - 392, 1974.

*Received March, 1997*

Acta Cybernetica 13 (1998) 243-256.

# Isomorphic representation of nondeterministic nilpotent automata

Ildikó Székely *

**Abstract**

In this paper, we deal with nondeterministic nilpotent automata and give a characterization of their isomorphic embedding with respect to the direct product.

## 1 Introduction

Investigation on homomorphic or isomorphic embeddings into products of automata is the starting point in the study of homomophic or isomorphic completeness of certain classes of automata with respect to different kinds of products. The problem of decomposition has a general approach in [4], where an abstract notion of composition and a general decomposition theorem are presented. Regarding the subdirect representation, there are several works dealing with this topic, see [3], [5], [2], [9], [8].

In this paper, we generalize the notion of nilpotency for nondeterministic automata and study the isomorphic representation of nondeterministic nilpotent automata under the direct product. As it turns out, this case is much more complicated than the deterministic one presented in [3].

## 2 Preliminaries

By a *nondeterministic automaton* we mean a system $\mathbf{A} = (X, A, \delta)$ , where $X$ and $A$ are nonempty finite sets, $X$ is the set of *input signs*, $A$ is the set of *states* and $\delta : A \times X \to \mathcal{P}(A)$ is the *transition function*. $\mathcal{P}(A)$ denotes here the power-set of $A$. For an input sign $x \in X$ and a state $a \in A$, $\delta(a, x)$ can be visualised as the set of all states in which the automaton goes when the current state is $a$ and the input sign is $x$. For $\delta(a, x)$, the notation $ax^{\mathbf{A}}$ is frequently used. Let us suppose that $a \in A$ and $p \in X^+$. The transition function can be extended as follows:

$$ap^{\mathbf{A}} = \begin{cases} \bigcup\limits_{b \in aq^{\mathbf{A}}} bx^{\mathbf{A}} & \text{if } p = qx \text{ where } x \in X \text{ and } q \in X^+, \\ ax^{\mathbf{A}} & \text{if } p = x \text{ and } x \in X. \end{cases}$$

*Department of Informatics, József Attila University, Árpád tér 2, H-6720 Szeged, Hungary

If $M \subseteq A, x \in X$, then we denote by $Mx^{\mathbf{A}}$ the set $\bigcup_{a \in M} ax^{\mathbf{A}}$ and if $p \in X^+$, then $Mp^{\mathbf{A}} = \bigcup_{a \in M} ap^{\mathbf{A}}$.

Let now $\mathbf{A} = (X, A, \delta)$ and $\mathbf{B} = (X, B, \delta_B)$ be two nondeterministic automata. It is said that $\mathbf{B}$ is a *subautomaton* of $\mathbf{A}$ if $B \subseteq A$ and $\delta_B$ is the restriction of $\delta$ to $B \times X$, i.e. $ax^{\mathbf{B}} = ax^{\mathbf{A}} \cap B$ is valid, for all $a \in A$ and $x \in X$. For two nondeterministic automata $\mathbf{A} = (X, A, \delta)$ and $\mathbf{B} = (X, B, \delta_B)$, a mapping $\mu : A \to B$ is a *homomorphism* if $\mu(ax^{\mathbf{A}}) = \mu(a)x^{\mathbf{B}}$ holds, for all $a \in A$ and $x \in X$. If the homomorphism $\mu : A \to B$ is an onto mapping, then we say that $\mathbf{B}$ is the homomorphic image of $\mathbf{A}$; moreover $\mu$ is called an *isomorphism*, if $\mu$ is a one-to-one mapping of $A$ onto $B$. In this case, we say that $\mathbf{A}$ and $\mathbf{B}$ are isomorphic.

Let $k$ be an arbitrary positive integer and $\mathbf{A}_r = (X, A_r, \delta_r), r = 1, \ldots, k$, be nondeterministic automata. By the *direct product* of these nondeterministic automata we mean the automaton $\mathbf{A} = (X, A, \delta)$ where $A = A_1 \times \cdots \times A_k$ and $\delta$ is defined as follows. For every $a = (a_1, \ldots, a_k) \in A$ and $x \in X$, $\delta(a, x) = \delta_1(a_1, x) \times \cdots \times \delta_k(a_k, x)$ or using the other notation: $ax^{\mathbf{A}} = (a_1, \ldots, a_k)x^{\mathbf{A}} = a_1x^{\mathbf{A}_1} \times \cdots \times a_kx^{\mathbf{A}_k}$. For the direct product of $\mathbf{A}_1, \ldots, \mathbf{A}_k$, we will use the notation $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k$. It is said that a nondeterministic automaton $\mathbf{A}$ *can be embedded isomorphically* into the direct product of $\mathbf{A}_1, \ldots, \mathbf{A}_k$ if $\mathbf{A}$ is isomorphic to a subautomaton $\mathbf{B}$ of $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k$.

Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic automaton and $\theta$ an equivalence relation on $A$. For every $a \in A$, let $\theta(a)$ denote the class of the partition belonging to $\theta$ and containing $a$. We can define a *factor-automaton* based on $\theta$ as follows. Let $\mathbf{A}/\theta = (X, A/\theta, \delta')$ be the nondeterministic automaton where the transition is defined by $\theta(a)x^{\mathbf{A}/\theta} = \{\theta(a') | a' \in bx^{\mathbf{A}} \text{ and } b \in \theta(a)\}$, for all $a \in A$ and $x \in X$. It is important to remark that the mapping $a \to \theta(a), a \in A$ is not a homomorphism of $\mathbf{A}$ onto $\mathbf{A}/\theta$ in general.

Let us introduce some special equivalence relations on $A$. If $a, b \in A$, then let $\theta_{a,b}$ be the equivalence relation defined in the following way: $u \ \theta_{a,b} \ v$ if and only if $u = v$ or $u, v \in \{a, b\}$, for all $u, v \in A$. It is obvious that the factor-set $A/\theta_{a,b} = \{\{u\} | u \in A \setminus \{a, b\}\} \cup \{\{a, b\}\}$. For the sake of simplicity we will denote by $u$ the classes of the form $\{u\}$ and by capital letters like $U$ or $V$ the classes of the form $\{a, b\}$ of the factor-set. We will use in the proofs of this paper the following important observation. If $\mathbf{A}$ has two states $c \neq d$ satisfying $cx^{\mathbf{A}} = dx^{\mathbf{A}}$, for all $x \in X$, then $\mathbf{A}/\theta_{c,d}$ is a homomorphic image of $\mathbf{A}$ under the homomorphism $a \to \theta_{c,d}(a), a \in A$.

Now, let $\theta_1, \ldots, \theta_k$ be arbitrary equivalence relations on $A$. Then, the mapping $\mu : A \to A/\theta_1 \times \cdots \times A/\theta_k$ given by $\mu(a) = (\theta_1(a), \ldots, \theta_k(a)), a \in A$, is called the *natural mapping*. It is easy to see that the natural mapping is a one-to-one mapping of $A$ into $A/\theta_1 \times \cdots \times A/\theta_k$ if $\bigcap_{r=1}^{k} \theta_r = \Delta_A$, where $\Delta_A$ is the equality relation on $A$. If we consider the factor-automata $\mathbf{A}_1, \ldots, \mathbf{A}_k$ based on arbitrary equivalence relations $\theta_1, \ldots, \theta_k$, respectively, then one can see that the natural mapping is not a homomorphism of $\mathbf{A}$ into $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k$, in general. In the constructive proofs given in this paper, we will have only natural mappings that are one-to-one mappings and are homomorphisms. By reducing the codomain of these mappings to $B = \mu(A)$ we obtain isomorphisms. Of course we will prove these features of the natural mapping in each case separately.

## 3 Nondeterministic nilpotent automata

Throughout the paper we restrict ourselves to the complete nondeterministic automata (see e.g. [1], [7]). Here, we recall its definition. A nondeterministic automaton $\mathbf{A} = (X, A, \delta)$ is called *complete* if $\delta(a, x) \neq \emptyset$ is valid, for all $a \in A$ and $x \in X$. In this paper, by nondeterministic automaton we mean a complete nondeterministic automaton. Moreover, we deal with a special class, the class of nondeterministic nilpotent automata. As a generalization of the traditional nilpotent automaton, it can be defined as follows.

A nondeterministic automaton $\mathbf{A} = (X, A, \delta)$ is called *nilpotent* if there exist a positive integer $n$ and a state $a_0$, such that $Ap^{\mathbf{A}} = \{a_0\}$ is valid, for all $p \in X^+$ with $|p| \geq n$ ($|p|$ denotes the length of the word $p$). The distinguished state $a_0$ is called the *absorbent state* of $\mathbf{A}$.

Now, let us define the following relation on $A$: $a \leq b$ if and only if $a = b$ or there is a $p \in X^+$ such that $b \in ap^{\mathbf{A}}$. It is easy to see that the introduced relation is a partial ordering on $A$ since $\mathbf{A}$ is nilpotent. If one of the relations $a \leq b$ or $b \leq a$ is valid, then $a$ and $b$ are called *comparable*. In the opposite case, we say that they are *incomparable* and it is denoted by $a \bowtie b$. Furthermore, the absorbent state $a_0$ is the greatest element in $(A, \leq)$ and if $A$ has at least two elements, then there exists a $b_0 \neq a_0 \in A$ such that $b_0$ is a maximal element in $(A \setminus \{a_0\}, \leq)$. From the maximality of $a_0$ and $b_0$, it follows that $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$ is valid, for all $x \in X$. Note that if there exists only one pair of states $a \neq b$ which satisfies $ax^{\mathbf{A}} = bx^{\mathbf{A}}$ for all $x \in X$, then these two states must be $a_0$ and $b_0$. Throughout the paper we will express this in a shorter way, using the sentence "$\mathbf{A}$ has exactly one pair of different states $a_0$ and $b_0$ for which $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$ holds, for all $x \in X$". In this case, $b_0$ is the greatest element in $(A \setminus \{a_0\}, \leq)$, because it is the only maximal element in this set.

The following statement is a consequence of Lemma 2 in [6] and it follows from the observation taken at the end of the previous section.

**Lemma 1** *Let $c$ and $d$ be two different states of a nondeterministic nilpotent automaton $\mathbf{A} = (X, A, \delta)$. If $cx^{\mathbf{A}} = dx^{\mathbf{A}}$ is valid, for all $x \in X$, then the factor-automaton $\mathbf{A}/\theta_{c,d}$ is also a nondeterministic nilpotent automaton.*

In particular, if $\{c, d\} = \{a_0, b_0\}$, then it is easy to see that the corresponding factor-automaton is a nondeterministic nilpotent automaton with the absorbent state $\theta_{a_0,b_0}(a_0) = \{a_0, b_0\}$.

Using the notion of incomparability, an other similar factor-automaton can be defined as follows. Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic nilpotent automaton with $|A| \geq 3$ and let $c \bowtie d \in A$ be incomparable states of $A$. Then, one can prove that $\mathbf{A}/\theta_{c,d}$ is a homomorphic image of $\mathbf{A}$ under the homomorphism $a \to \theta_{c,d}(a), a \in A$. This observation leads to the following statement:

**Lemma 2** *If a nondeterministic nilpotent automaton $\mathbf{A} = (X, A, \delta)$ with $|A| \geq 3$ has two incomparable states c and d, then the factor-automaton $\mathbf{A}/\theta_{c,d}$ is also a nondeterministic nilpotent automaton.*

The following property is very important with respect to the inner structure of the nondeterministic nilpotent automata. It shows that no "loops" or "circuits" may appear on the states of a nondeterministic nilpotent automaton except for the absorbent state.

**Lemma 3** *If $\mathbf{A} = (X, A, \delta)$ is a nondeterministic nilpotent automaton with the absorbent state $a_0$, then $a \notin ap^{\mathbf{A}}$ holds, for all $a \in A \setminus \{a_0\}$ and $p \in X^+$.*

We will also refer to the next Lemma, whose proof needs the following observation which is a direct consequece of the definitions. If $\mathbf{A}_1, \ldots, \mathbf{A}_k$ are nondeterministic nilpotent automata with the absorbent states $a_1^0, \ldots, a_k^0$, respectively, then their direct product is a nondeterministic nilpotent automaton with the absorbent state $(a_1^0, \ldots, a_k^0)$. Furthermore, every complete subautomaton of the direct product is nilpotent with the absorbent state $(a_1^0, \ldots, a_k^0)$. This also means that if a nondeterministic nilpotent automaton $\mathbf{A} = (X, A, \delta)$ with the absorbent state $a_0$ can be embedded into $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k$ under the isomorphism $\mu$, then $\mu(a_0) = (a_1^0, \ldots, a_k^0)$.

**Lemma 4** *Assume that $\mathbf{A} = (X, A, \delta)$ is a nondeterministic nilpotent automaton with the absorbent state $a_0$ and $\mathbf{A}$ has exactly one pair of different states $a_0, b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$. Let $\mathbf{A}_r = (X, A_r, \delta_r), r = 1, \ldots, k$, $(k \geq 2)$ be nondeterministic nilpotent automata with the absorbent states $a_1^0, \ldots, a_k^0$, respectively, and let $\mu : A \to B \subseteq A_1 \times \cdots \times A_k$ be an isomorphism of $\mathbf{A}$ into $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k$. If we denote the image of $b_0$ under $\mu$ by $(b_1^0, \ldots, b_k^0)$ and the set of indices $\{i \in \{1, \ldots, k\} | a_i^0 \neq b_i^0\}$ by $I$, then the components $a_i^0, b_i^0$ with $i \in I$ may appear in no other elements of $\mathbf{B}$ but $\mu(a_0)$ and $\mu(b_0)$.*

## 4 Isomorphic representation of nondeterministic nilpotent automata

**Theorem 1** *Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic nilpotent automaton with $|A| \geq 3$, the absorbent state $a_0$, and let $b_0$ be a maximal element in $(A \setminus \{a_0\}, \leq)$. If there exist $a_1, b_1 \in A$, $a_1 \neq b_1$, $\{a_1, b_1\} \neq \{a_0, b_0\}$ such that beside $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$, $a_1x^{\mathbf{A}} = b_1x^{\mathbf{A}}$ also holds for all $x \in X$, then $\mathbf{A}$ can be embedded isomorphically into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$.*

**Proof. Case 1.** $\{a_0, b_0\} \cap \{a_1, b_1\} = \emptyset$. Let $\mathbf{A}_1 = \mathbf{A}/\theta_{a_0,b_0}$ and $\mathbf{A}_2 = \mathbf{A}/\theta_{a_1,b_1}$. Then, $\mathbf{A}$ can be embedded isomorphically into $\mathbf{A}_1 \times \mathbf{A}_2$ under the natural mapping denoted by $\mu$. Let $B = \mu(A)$. Since $\theta_{a_0,b_0} \cap \theta_{a_1,b_1} = \Delta_A$, $\mu$ is a one-to-one mapping. To simplify the notation we will use $U_0$ for $\theta_{a_0,b_0}(a_0) = \{a_0, b_0\}$ and $U_1$ for $\theta_{a_1,b_1}(a_1) = \{a_1, b_1\}$. To prove that $\mu$ is an isomorphism, we have to show that $\mu(ax^{\mathbf{A}}) = \mu(a)x^{\mathbf{A}_1\times\mathbf{A}_2} \cap B$, for all $a \in A$ and $x \in X$. We can do this by evaluating $\mu(ax^{\mathbf{A}})$ and $\mu(a)x^{\mathbf{A}_1\times\mathbf{A}_2} \cap B$ in the following cases: $a \in A \setminus (U_0 \cup U_1)$ and $ax^{\mathbf{A}} \cap (U_0 \cup U_1) = \emptyset$; $a \in A \setminus (U_0 \cup U_1)$ and $ax^{\mathbf{A}} \cap U_0 = \emptyset$ and $ax^{\mathbf{A}} \cap U_1 \neq \emptyset$; $a \in A \setminus (U_0 \cup U_1)$ and $ax^{\mathbf{A}} \cap U_0 \neq \emptyset$ and $ax^{\mathbf{A}} \cap U_1 = \emptyset$; $a \in A \setminus (U_0 \cup U_1)$ and $ax^{\mathbf{A}} \cap U_0 \neq \emptyset$ and $ax^{\mathbf{A}} \cap U_1 \neq \emptyset$; $a \in U_1$ and $ax^{\mathbf{A}} \cap U_0 = \emptyset$ and $ax^{\mathbf{A}} \cap U_1 = \emptyset$; $a \in U_1$ and $ax^{\mathbf{A}} \cap U_0 = \emptyset$ and $ax^{\mathbf{A}} \cap U_1 \neq \emptyset$; $a \in U_1$ and $ax^{\mathbf{A}} \cap U_0 \neq \emptyset$ and $ax^{\mathbf{A}} \cap U_1 = \emptyset$; $a \in U_1$ and $ax^{\mathbf{A}} \cap U_0 \neq \emptyset$ and $ax^{\mathbf{A}} \cap U_1 \neq \emptyset$; $a \in U_0$.

**Case 2.** $\{a_0, b_0\} \cap \{a_1, b_1\} \neq \emptyset$. $\{a_0, b_0\} \neq \{a_1, b_1\}$, hence $\{a_0, b_0\} \cap \{a_1, b_1\}$ contains exactly one element. Let $\{b_1\} = \{a_0, b_0\} \cap \{a_1, b_1\}$. There are two possibilities: $b_1 = a_0$ or $b_1 = b_0$. In both cases, $a_1x^{\mathbf{A}} = b_0x^{\mathbf{A}} = a_0x^{\mathbf{A}} = \{a_0\}$, for all $x \in X$. Consequently, the factor-automata $\mathbf{A}_1 = \mathbf{A}/\theta_{a_0,b_0}$, $\mathbf{A}_2 = \mathbf{A}/\theta_{a_0,a_1}$ and $\mathbf{A}_3 = \mathbf{A}/\theta_{b_0,a_1}$ are nondeterministic nilpotent automata. We show that $\mathbf{A}$ can be embedded isomorphically into $\mathbf{A}_1 \times \mathbf{A}_2 \times \mathbf{A}_3$ under the natural mapping denoted by $\mu$. Let $B = \mu(A) \subseteq A_1 \times A_2 \times A_3$. The mapping $\mu$ is one-to-one since $\theta_{a_0,b_0} \cap \theta_{a_0,a_1} \cap \theta_{b_0,a_1} = \Delta_A$. It also satisfies $\mu(ax^{\mathbf{A}}) = \mu(a)x^{\mathbf{A}_1\times\mathbf{A}_2\times\mathbf{A}_3} \cap B$, for all $a \in A$ and $x \in X$. To prove this, we must evaluate once again $\mu(ax^{\mathbf{A}})$ and $\mu(a)x^{\mathbf{A}_1\times\mathbf{A}_2\times\mathbf{A}_3} \cap B$ in the following cases: $a \in A \setminus \{a_0, b_0, a_1\}$ and $ax^{\mathbf{A}} \cap \{a_0, b_0, a_1\} = \{a_0\}$ (or $\{b_0\}$, or $\{a_1\}$); $a \in A \setminus \{a_0, b_0, a_1\}$ and $ax^{\mathbf{A}} \cap \{a_0, b_0, a_1\} = \{a_0, b_0\}$ (or $\{a_0, a_1\}$, or $\{b_0, a_1\}$); $a \in A \setminus \{a_0, b_0, a_1\}$ and $ax^{\mathbf{A}} \cap \{a_0, b_0, a_1\} = \{a_0, b_0, a_1\}$; $a \in \{a_0, b_0, a_1\}$. □

**Corollary 1** *If a nondeterministic nilpotent automaton* $\mathbf{A} = (X, A, \delta)$ *cannot be embedded into a direct product of nondeterministic nilpotent automata having less states than* $|A|$, *then* $\mathbf{A}$ *has exactly one pair of different states* $a_0, b_0$ *for which* $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ *holds, for all* $x \in X$. *These states are* $a_0$, *the absorbent state and* $b_0$, *the greatest element in* $(A \setminus \{a_0\}, \leq)$.

It is worth noting that, unlike for deterministic automata (see [3]), the converse statement is not true.

**Lemma 5** *Let* $\mathbf{A} = (X, A, \delta)$ *be a nondeterministic nilpotent automaton* ($|A| \geq 3$) *that has exactly one pair of different states* $a_0, b_0$ *for which* $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ *holds, for all* $x \in X$. *If* $\mathbf{A}$ *can be embedded isomorphically into a direct product of nondeterministic nilpotent automata having fewer states than* $|A|$, *then there must exist a pair of incomparable states* $c \bowtie d \in A$.

**Proof.** Let $\mu : A \to B \subseteq A_1 \times \cdots \times A_k$ be an embedding isomorphism, where $k \geq 2$ is fixed and $|A_r| < |A|$, for all $r = 1, \ldots, k$. Since $|A_r| < |A|$, there exists a state $\bar{a}_r \in A_r$, such that $\bar{a}_r$ occurs in at least two different elements of $B$ on the $r$-th position, for all $r = 1, \ldots, k$. If we use the notation $(a_1^0, \ldots, a_k^0)$ for $\mu(a_0)$ and $(b_1^0, \ldots, b_k^0)$ for $\mu(b_0)$, then, because $a_0 \neq b_0$ and $\mu$ is an isomorphism, there has to be an index $i \in \{1, \ldots, k\}$ such that $a_i^0 \neq b_i^0$. Let $\mathbf{b}'$ and $\mathbf{b}''$ be two different elements of $B$ in which $\bar{a}_i$ occurs on the $i$-th position, and let $c, d \in A$ be the states for which $\mu(c) = \mathbf{b}'$ and $\mu(d) = \mathbf{b}''$. $\mu$ is an isomorphism, hence $c \neq d$. We will show now, that $c$ and $d$ are incomparable in $(A, \leq)$. Assume to the contrary that

$c \leq d$ (or $d \leq c$). Since $c \neq d$, there exists $p \in X^+$, such that $d \in cp^{\mathbf{A}}$. In the meantime, $\mu$ is an isomorphism, so we can conclude that $\mu(d) \in \mu(c)p^{\mathbf{B}}$, which implies $\bar{a}_i \in \bar{a}_i p^{\mathbf{A}_i}$. Thus, due to Lemma 3, the only possibility for $\bar{a}_i$ is $\bar{a}_i = a_i^0$. Since $b_0$ is the greatest element in $(A \setminus \{a_0\}, \leq)$, it is obvious that $d \leq b_0$, i.e. $d = b_0$ or there exists a $q \in X^+$ such that $b_0 \in dq^{\mathbf{A}}$. If $d = b_0$, then $b_i^0 = a_i^0$ follows immediately and contradicts the assumption $b_i^0 \neq a_i^0$. If $b_0 \in dq^{\mathbf{A}}$, then under the isomorphism $\mu$ we have $\mu(b^0) \in \mu(d)q^{\mathbf{B}}$, which for the $i$-th component means $b_i^0 \in a_i^0 q^{\mathbf{A}_i}$. This contradicts the nilpotency of $\mathbf{A}_i$. □

**Corollary 2** *If the partially ordered set $(A, \leq)$ of a nondeterministic nilpotent automaton $\mathbf{A} = (X, A, \delta)$ with $|A| \geq 3$ is a chain, then $\mathbf{A}$ cannot be embedded into a direct product of nondeterministic nilpotent automata having less states than $|A|$.*

**Lemma 6** *Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic nilpotent automaton ($|A| \geq 4$) that has exactly one pair of different states $a_0, b_0$ for which $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$ holds, for all $x \in X$. If there exist $c \bowtie d \in A$ such that for all $x \in X$, $cx^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ and $dx^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ jointly imply $cx^{\mathbf{A}} \cap \{a_0, b_0\} = dx^{\mathbf{A}} \cap \{a_0, b_0\}$, then $\mathbf{A}$ can be embedded isomorphically into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$.*

**Proof.** Since $c$ and $d$ are incomparable, $\{c, d\} \cap \{a_0, b_0\} = \emptyset$. Let $\mathbf{A}_1 = \mathbf{A}/\theta_{a_0,b_0}$ and $\mathbf{A}_2 = \mathbf{A}/\theta_{c,d}$. Since $c \bowtie d$ and $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$, for all $x \in X$, both $\mathbf{A}/\theta_{a_0,b_0}$ and $\mathbf{A}/\theta_{c,d}$ are nondeterministic nilpotent automata. Now, we will prove that $\mathbf{A}$ can be embedded isomorphically into $\mathbf{A}_1 \times \mathbf{A}_2$ under the natural mapping denoted by $\mu$. Since $\theta_{a_0,b_0} \cap \theta_{c,d} = \Delta_A$, $\mu$ is a one-to-one mapping of $A$ into $A/\theta_{a_0,b_0} \times A/\theta_{c,d}$. To prove that $\mu$ is an isomorphism, let $B = \mu(A)$. For the sake of simplity, let us denote by $U$ $\theta_{a_0,b_0}(a_0) = \{a_0, b_0\}$ and by $V$ $\theta_{c,d}(c) = \{c, d\}$. Then, we have to evaluate $\mu(ax^{\mathbf{A}})$ and $\mu(a)x^{\mathbf{A}_1 \times \mathbf{A}_2} \cap B$ in the following cases: $a \in A \setminus (U \cup V)$ and $ax^{\mathbf{A}} \cap (U \cup V) = \emptyset$; $a \in A \setminus (U \cup V)$ and $ax^{\mathbf{A}} \cap U = \emptyset$ and $ax^{\mathbf{A}} \cap V \neq \emptyset$; $a \in A \setminus (U \cup V)$ and $ax^{\mathbf{A}} \cap U \neq \emptyset$ and $ax^{\mathbf{A}} \cap V = \emptyset$; $a \in A \setminus (U \cup V)$ and $ax^{\mathbf{A}} \cap U \neq \emptyset$ and $ax^{\mathbf{A}} \cap V \neq \emptyset$; $a \in U$; $a = c$ (or $d$) and $cx^{\mathbf{A}} \cap U = \emptyset$ and $dx^{\mathbf{A}} \cap U = \emptyset$; $a = c$ (or $d$) and $cx^{\mathbf{A}} \cap U = \emptyset$ and $dx^{\mathbf{A}} \cap U \neq \emptyset$; $a = c$ (or $d$) and $cx^{\mathbf{A}} \cap U \neq \emptyset$ and $dx^{\mathbf{A}} \cap U = \emptyset$; $a = c$ (or $d$) and $cx^{\mathbf{A}} \cap U \neq \emptyset$ and $dx^{\mathbf{A}} \cap U \neq \emptyset$. In this latter case, we have to use the assumption of the lemma, namely, if $cx^{\mathbf{A}} \cap U \neq \emptyset$ and $dx^{\mathbf{A}} \cap U \neq \emptyset$, then $cx^{\mathbf{A}} \cap U = dx^{\mathbf{A}} \cap U$. □

**Corollary 3** *Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic nilpotent automaton with $|A| \geq 4$ that has exactly one pair of different states $a_0, b_0$ for which $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$ holds, for all $x \in X$. If $\mathbf{A}$ has exactly one pair of incomparable states $c \bowtie d$ and $\mathbf{A}$ cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$, then there has to be $\bar{x} \in X$ such that $c\bar{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$, $d\bar{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ and $c\bar{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq d\bar{x}^{\mathbf{A}} \cap \{a_0, b_0\}$ hold.*

**Lemma 7** *Assume that $\mathbf{A} = (X, A, \delta)$ is a nondeterministic nilpotent automaton ($|A| \geq 4$) that has exactly one pair of different states $a_0, b_0$ for which $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$ holds, for all $x \in X$, and that $\mathbf{A}$ can be embedded isomorphically into a direct* *product of nondeterministic nilpotent automata having fewer states than $|A|$. If for $c \bowtie d \in A$, there exists $\overline{x} \in X$ such that $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$, $d\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ and $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq d\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\}$ simultaneously hold, then there exists $e \bowtie f \in A$ such that $\{e, f\} \neq \{c, d\}$.*

**Proof.** We use the same notations as in the proof of Lemma 5. Let $\mu(c) = \mathbf{b}' = (b_1', \ldots, \overline{a}_i, \ldots, b_k')$ and $\mu(d) = \mathbf{b}'' = (b_1'', \ldots, \overline{a}_i, \ldots, b_k'')$. We must analyse the cases in which the given conditions $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$, $d\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ and $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq d\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\}$ hold. Let us assume that $c\overline{x}^{\mathbf{A}} = \{a_0\}$ and $d\overline{x}^{\mathbf{A}} = \{b_0\}$ (all other cases can be proved similarly to this one). This yields that $\{a_i^0, b_i^0\} \subseteq \overline{a}_i \overline{x}^{\mathbf{A}_i}$ and that there exists an index $j \neq i$, such that $b_j' \neq b_j''$ and $a_j^0 \neq b_j^0$. For this index $j$, the relations $a_j^0 \in b_j' \overline{x}^{\mathbf{A}_j}$, $b_j^0 \notin b_j' \overline{x}^{\mathbf{A}_j}$, $a_j^0 \notin b_j'' \overline{x}^{\mathbf{A}_j}$ and $b_j^0 \in b_j'' \overline{x}^{\mathbf{A}_j}$ also hold. Consider now $\overline{a}_j$ and two different elements of $B$ in which $\overline{a}_j$ occurs on the $j$-th position. Let $e$ and $f$ be those states of $A$ whose images under the isomorphism $\mu$ are these two elements. Like in the proof of Lemma 5, one can see that $e \bowtie f$. We still have to prove that $\{e, f\} \neq \{c, d\}$. For $\overline{a}_j$, there are three possibilities: $\overline{a}_j \notin \{b_j', b_j''\}$, $\overline{a}_j = b_j'$ or $\overline{a}_j = b_j''$. In the first case, we have $\{e, f\} \cap \{c, d\} = \emptyset$, in the second case, $\{e, f\} \cap \{c, d\} = \{c\}$ and in the third case, $\{e, f\} \cap \{c, d\} = \{d\}$. Consequently, $\{e, f\} \neq \{c, d\}$. □

**Corollary 4** *Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic nilpotent automaton with $|A| \geq 4$ that has exactly one pair of different states $a_0, b_0$ for which $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$ holds, for all $x \in X$. If $\mathbf{A}$ has exactly one pair of incomparable states $c \bowtie d$ and there exists $\overline{x} \in X$ such that $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$, $d\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ and $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq d\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\}$ are true, then $\mathbf{A}$ cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$.*

**Lemma 8** *Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic nilpotent automaton ($|A| \geq 4$) that has exactly one pair of different states $a_0, b_0$ for which $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$ holds, for all $x \in X$. If there are $c \bowtie d \in A$ and $e \bowtie f \in A$ such that $\{c, d\} \cap \{e, f\} = \emptyset$, then $\mathbf{A}$ can be embedded isomorphically into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$.*

**Proof.** Let $\mathbf{A}_1 = \mathbf{A}/\theta_{a_0,b_0}$, $\mathbf{A}_2 = \mathbf{A}/\theta_{c,d}$ and $\mathbf{A}_3 = \mathbf{A}/\theta_{e,f}$. Since $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$, for all $x \in X$, and $c \bowtie d$, $e \bowtie f$, the factor-automata $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$ are nondeterministic nilpotent automata. On the other hand, $\theta_{a_0,b_0} \cap \theta_{c,d} \cap \theta_{e,f} = \Delta_A$, and thus, the natural mapping denoted by $\mu$ is a one-to-one mapping. To prove that $\mu$ is an isomorphism, as in the constructive proofs given above, we must calculate $\mu(ax^{\mathbf{A}})$ and $\mu(a)x^{\mathbf{A}_1 \times \mathbf{A}_2 \times \mathbf{A}_3} \cap B$ in the corresponding cases, where $B = \mu(A)$. □

**Lemma 9** *Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic nilpotent automaton ($|A| \geq 5$) that has exactly one pair of different states $a_0, b_0$ for which $a_0 x^{\mathbf{A}} = b_0 x^{\mathbf{A}}$ holds, for all $x \in X$. If there are three pairwise incomparable states $c \bowtie d \in A$, $d \bowtie e \in A$ and $e \bowtie c \in A$, then $\mathbf{A}$ can be embedded isomorphically into the direct product of nondeterministic nilpotent automata having fewer states than $|A|$.*

**Proof.** We construct the following three factor-automata: $\mathbf{A}_1 = \mathbf{A}/\theta_{c,d}$, $\mathbf{A}_2 = \mathbf{A}/\theta_{d,e}$ and $\mathbf{A}_3 = \mathbf{A}/\theta_{e,c}$. Since $c \bowtie d$, $d \bowtie e$ and $e \bowtie c$, each of the automata $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$ is a nondeterministic nilpotent automaton. It can be proved that the natural mapping is an isomorphism in this case. The proof goes similarly to the cases mentioned above. □

Note that the condition that $\mathbf{A}$ has exactly one pair of different states $a_0, b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$, was not used in this proof, hence, it is not necessary.

**Corollary 5** *Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic nilpotent automaton with $|A| \geq 5$ that has exactly one pair of different states $a_0, b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$. If $\mathbf{A}$ has at least two incomparable states and $\mathbf{A}$ cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$, then there has to be $c \in A \setminus \{a_0, b_0\}$ such that for every $e \bowtie f \in A$, $c \in \{e, f\}$ must hold.*

**Lemma 10** *Let $\mathbf{A} = (X, A, \delta)$ be a nondeterministic nilpotent automaton with $|A| \geq 5$ that has exactly one pair of different states $a_0, b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$. If there are a natural number $k \geq 2$ and $c, e_1, \ldots, e_k \in A$ with $c \bowtie e_r, r = 1, \ldots, k$, such that*

$$\forall x \in X \left(\exists i \in \{1, \ldots, k\} : e_ix^{\mathbf{A}} \cap \{a_0, b_0\} \subseteq cx^{\mathbf{A}} \cap \{a_0, b_0\}\right) \text{ holds,}$$

*then $\mathbf{A}$ can be embedded isomorphically into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$.*

**Proof.** We construct the following $k+1$ automata: let $\mathbf{A}_0 = \mathbf{A}/\theta_{a_0,b_0}$ and for every $r = 1, \ldots, k$, let $\mathbf{A}_r = \mathbf{A}/\theta_{c,e_r}$.

Since $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ for all $x \in X$ and $c \bowtie e_r, r = 1, \ldots, r$, each of the automata $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_k$ is a nondeterministic nilpotent automaton. We prove now that the natural mapping is an isomorphism of $\mathbf{A}$ into $\mathbf{A}_0 \times \mathbf{A}_1 \times \cdots \times \mathbf{A}_k$. Since $\theta_{a_0,b_0} \cap \theta_{c,e_1} \cap \cdots \cap \theta_{c,e_k} = \Delta_A$, $\mu$ is a one-to-one mapping. To prove that $\mu$ is an isomorphism, we have to investigate more cases. For the sake of simplity, we shall use the notations: $U = \theta_{a_0,b_0}(a_0) = \{a_0, b_0\}$ and for $r = 1, \ldots, k$, $V_r = \theta_{c,e_r}(c) = \{c, e_r\}$. We will give the proof in detail for the following two cases.

Assume that $a = c$ and $cx^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$. Then,

$$\mu(cx^{\mathbf{A}}) = \mu(cx^{\mathbf{A}} \setminus \{a_0, b_0\}) \cup \mu(cx^{\mathbf{A}} \cap \{a_0, b_0\}) =$$

$$= \{(b, \ldots, b) | b \in cx^{\mathbf{A}} \setminus U\} \cup \{(U, b, \ldots, b) | b \in cx^{\mathbf{A}} \cap U\}.$$

On the other hand,

$$\mu(c)x^{\mathbf{A}_0 \times \mathbf{A}_1 \times \cdots \times \mathbf{A}_k} = (c, V_1, V_2, \ldots, V_k)x^{\mathbf{A}_0 \times \mathbf{A}_1 \times \cdots \times \mathbf{A}_k} =$$

$$= cx^{\mathbf{A}_0} \times V_1x^{\mathbf{A}_1} \times \cdots \times V_kx^{\mathbf{A}_k} =$$

$$= \left((cx^{\mathbf{A}} \setminus U) \cup \{U\}\right) \times (cx^{\mathbf{A}} \cup e_1x^{\mathbf{A}}) \times \cdots \times (cx^{\mathbf{A}} \cup e_kx^{\mathbf{A}}) =$$

$$= \left((cx^{\mathbf{A}} \setminus U) \times (cx^{\mathbf{A}} \cup e_1x^{\mathbf{A}}) \times \cdots \times (cx^{\mathbf{A}} \cup e_kx^{\mathbf{A}})\right) \cup$$
$$\cup \left(\{U\} \times (cx^{\mathbf{A}} \cup e_1x^{\mathbf{A}}) \times \cdots \times (cx^{\mathbf{A}} \cup e_kx^{\mathbf{A}})\right),$$

which implies

$$\mu(c)x^{\mathbf{A}_0\times\mathbf{A}_1\times\cdots\times\mathbf{A}_k} \cap B =$$
$$= \{(b,\ldots,b) | b \in cx^{\mathbf{A}} \setminus U\} \cup \left\{(U,b,\ldots,b) | b \in \left(\bigcap_{r=1}^{k}(cx^{\mathbf{A}} \cup e_rx^{\mathbf{A}})\right) \cap U\right\} =$$
$$= \{(b,\ldots,b) | b \in cx^{\mathbf{A}} \setminus U\} \cup \left\{(U,b,\ldots,b) | b \in (cx^{\mathbf{A}} \cap U) \cup \left(\bigcap_{r=1}^{k}(e_rx^{\mathbf{A}} \cap U)\right)\right\}.$$

Since there exists an index $i$ such that $e_ix^{\mathbf{A}} \cap \{a_0, b_0\} \subseteq cx^{\mathbf{A}} \cap \{a_0, b_0\}$, the inclusion $\bigcap_{r=1}^{k}(e_rx^{\mathbf{A}} \cap U) \subseteq cx^{\mathbf{A}} \cap U$ also holds and $\mu(cx^{\mathbf{A}}) = \mu(c)x^{\mathbf{A}_0\times\mathbf{A}_1\times\cdots\times\mathbf{A}_k} \cap B$.

Now, assume that $a = e_j$ for some $j \in \{1,\ldots,k\}$ and $e_jx^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$. Let $I = \{l | e_l \in e_jx^{\mathbf{A}}\}$ and $J = \{l | e_l \notin e_jx^{\mathbf{A}}\}$. Then,

$$\mu(e_jx^{\mathbf{A}}) = \mu\left(e_jx^{\mathbf{A}} \setminus (\{a_0,b_0\} \cup \{e_l | l \in I\})\right) \cup \mu\left(e_jx^{\mathbf{A}} \cap \{a_0,b_0\}\right) \cup \mu\left(\{e_l | l \in I\}\right) =$$
$$= \left\{(b,b,\ldots,b) | b \in e_jx^{\mathbf{A}} \setminus (\{a_0,b_0\} \cup \{e_l | l \in I\})\right\} \cup$$
$$\cup \left\{x^{\mathbf{A}}(U,b,\ldots,b) | b \in e_jx^{\mathbf{A}} \cap \{a_0,b_0\}\right\} \cup \bigcup_{l\in I}\{(e_l,e_l,\ldots,e_l,V_l,e_l,\ldots,e_l)\},$$

where $V_l$ occurs on the $(l+1)$-th position of the element $(e_l,e_l,\ldots,e_l,V_l,e_l,\ldots,e_l)$.

$$\mu(e_j)x^{\mathbf{A}_0\times\mathbf{A}_1\times\cdots\times\mathbf{A}_k} = (e_j,e_j,\ldots,e_j,V_j,e_j,\ldots,e_j)x^{\mathbf{A}_0\times\mathbf{A}_1\times\cdots\times\mathbf{A}_k} =$$
$$= e_jx^{\mathbf{A}_0} \times e_jx^{\mathbf{A}_1} \times \cdots \times e_jx^{\mathbf{A}_{j-1}} \times V_jx^{\mathbf{A}_j} \times e_jx^{\mathbf{A}_{j+1}} \times \cdots \times e_jx^{\mathbf{A}_k} =$$
$$= \left((e_jx^{\mathbf{A}} \setminus U) \cup \{U\}\right) \times M_1 \times \cdots \times M_{j-1} \times (cx^{\mathbf{A}} \cup e_jx^{\mathbf{A}}) \times M_{j+1} \times \cdots \times M_k,$$

where

$$M_l = \begin{cases} e_jx^{\mathbf{A}} & \text{if } l \in J, \\ (e_jx^{\mathbf{A}} \setminus V_l) \cup \{V_l\} & \text{if } l \in I, \end{cases}$$

for all $l \in \{1,\ldots,j-1,j+1,\ldots,k\}$. Therefore,

$$\mu(e_j)x^{\mathbf{A}_0\times\mathbf{A}_1\times\cdots\times\mathbf{A}_k} \cap B =$$
$$= \{(b,\ldots,b) | b \in e_jx^{\mathbf{A}} \setminus (U \cup \bigcup_{l\in I} V_l)\} \cup \left\{(U,b,\ldots,b) | b \in \bigcap_{l\in I}(e_jx^{\mathbf{A}} \setminus V_l) \cap \{a_0,b_0\}\right\} \cup$$
$$\cup \bigcup_{l\in I}\{(e_l,e_l,\ldots,e_l,V_l,e_l,\ldots,e_l)\}$$

and the proof of $\mu(cx^{\mathbf{A}}) = \mu(c)x^{\mathbf{A}_0\times\mathbf{A}_1\times\cdots\times\mathbf{A}_k} \cap B$ is complete in this case, too.

The proof of $\mu(ax^{\mathbf{A}}) = \mu(a)x^{\mathbf{A}_0\times\mathbf{A}_1\times\cdots\times\mathbf{A}_k} \cap B$ is similar in all other cases. □

By studying in detail all the cases of the proof of Lemma 10, we can state the following result.

**Corollary 6** *Let* $\mathbf{A} = (X, A, \delta)$ *be a nondeterministic nilpotent automaton with* $|A| \geq 5$ *that has exactly one pair of different states* $a_0, b_0$ *for which* $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ *holds, for all* $x \in X$. *If there exists a natural number* $k \geq 2$ *and there are* $c, e_1, \ldots, e_k \in A$ *with* $c \bowtie e_r, r = 1, \ldots, k$, *such that for all* $x \in X$,

$$cx^{\mathbf{A}} \cap \{a_0, b_0\} = \emptyset \text{ or } \exists i \in \{1, \ldots, k\} \text{ such that } e_ix^{\mathbf{A}} \cap \{a_0, b_0\} \subseteq cx^{\mathbf{A}} \cap \{a_0, b_0\}$$

*holds, then* $\mathbf{A}$ *can be embedded isomorphically into a direct product of nondeterministic nilpotent automata having fewer states than* $|A|$.

**Theorem 2** *Let* $\mathbf{A} = (X, A, \delta)$ *be a nondeterministic nilpotent automaton with* $|A| \geq 4$ *that has exactly one pair of different states* $a_0, b_0$ *for which* $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ *holds, for all* $x \in X$ . *If there exists* $c \in A$ *such that:*

(a) *for all* $e \bowtie f \in A$, $c \in \{e, f\}$, *and*

(b) *there exists* $\overline{x} \in X$ *such that* $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ *and for all* $e \in A$ *with* $c \bowtie e$, $e\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \not\subseteq c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\}$ *holds,*

*then* $\mathbf{A}$ *cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than* $|A|$.

**Proof.** Assume to the contrary that $\mathbf{A}$ can be embedded into the direct product of $k \geq 2$ nondeterministic nilpotent automata having fewer states than $|A|$. Let $A_1, \ldots, A_k$ be these nondeterministic nilpotent automata with $|A_r| < |A|$, $r = 1, \ldots, k$, and let $\mu : A \to B \subseteq A_1 \times \cdots \times A_k$ be the embedding isomorphism. Let $\mathbf{B}$ denote the subautomaton of $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k$ with state set $B$. Let $\mu(a_0) = (a_1^0, \ldots, a_k^0)$ and $\mu(b_0) = (b_1^0, \ldots, b_k^0)$. We consider the following set of indices: $I = \{r \in \{1, \ldots, k\}| \ a_r^0 \neq b_r^0\}$. Since $a_0 \neq b_0$ and $\mu$ is an isomorphism, $I \neq \emptyset$. Without loss of generality, we may assume that $I = \{1, \ldots, m\}$ where $m \leq k$ and $m$ is a fixed value.

For all $i \in I$, let us examine the state $\overline{a}_i$, i.e. the state which occurs in at least two different elements of $B$ on the $i$-th position. The existence of $\overline{a}_i$ follows from the fact that $|A_i| < |A|$. Due to Lemma 4 and the definition of $I$, $\overline{a}_i$ will appear neither in the element $(a_1^0, \ldots, a_k^0)$, nor in $(b_1^0, \ldots, b_k^0)$. According to this and to Lemma 3 that guarantees that in a nondeterministic nilpotent automaton no state may have "loops"or "circuits" except for the absorbent state, the ancestors with respect to $\mu$ of these elements are incomparable in $\mathbf{A}$. Consequently, for every $\overline{a}_i, i = 1, \ldots, m$, we have in $\mathbf{A}$ a pair of incomparable states, that don't have to be different for different $\overline{a}_i$-s. Due to (a) these states are of the form $c \bowtie e_1, \ldots, c \bowtie e_\nu$, where $\nu \leq m$ and $\nu$ is a fixed value. According to this, $\mu(c) = (\overline{a}_1, \ldots, \overline{a}_m, c_{m+1}, \ldots, c_k)$.

In the same time, $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \{a_0, b_0\}$ and also $e_i\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ holds for all $i \in I$, because otherwise there would exist $e \in A$ such that $e\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \subseteq c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\}$ would be true in a trivial way and condition (b) would not be satisfied. $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ stated by (b) and $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \{a_0, b_0\}$ implies $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} = \{a_0\}$ or $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} = \{b_0\}$.

Let us analyse the case $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} = \{a_0\}$. Because $\mu$ is an isomorphism and $\mathbf{B}$ is a subautomaton of $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k$, $a_i^0 \in \overline{a}_i\overline{x}^{\mathbf{A}_i}$, for all $i = 1, \ldots, m$, and $a_l^0 \in c_l\overline{x}^{\mathbf{A}_l}$, for all $l = m+1, \ldots, k$, must be true. Remember that $a_l^0 = b_l^0$, for all $l = m+1, \ldots, k$. This implies the existence of a $j \in I$ such that $b_j^0 \notin \overline{a}_j\overline{x}^{\mathbf{A}_j}$ because, otherwise $b_i^0 \in \overline{a}_i\overline{x}^{\mathbf{A}_i}$, for all $i = 1, \ldots, m$, and $b_l^0 \in c_l\overline{x}^{\mathbf{A}_l}$, for all $l = m+1, \ldots, k$ would hold, which would imply $b_0 \in c\overline{x}^{\mathbf{A}}$. Let us denote by $J$ the following set of indices $J = \{i \in I | \; b_j^0 \notin \overline{a}_j\overline{x}^{\mathbf{A}_j}\}$. Then, by the statements above, it is obvious that $J \neq \emptyset$. On the other hand, all $e_j$ with $j \in J$ must satisfy (b). This means that $e_j\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \not\subseteq \{a_0\}$, for all $j \in J$. We already observed that $e_i\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ for all $i \in I$, thus the single possibility for $e_j$ is $b_j^0 \in \overline{a}_j\overline{x}^{\mathbf{A}_j}$, for all $j \in J$, which contradicts the definition and the nonemptyness of the set $J$. The analysis of the case $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} = \{b_0\}$ is similar. □

**Theorem 3** *Assume that* $\mathbf{A} = (X, A, \delta)$ *is a nondeterministic nilpotent automaton* ($|A| \geq 5$) *that has exactly one pair of different states* $a_0, b_0$ *for which* $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ *holds, for all* $x \in X$. *If* $\mathbf{A}$ *cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than* $|A|$ *and* $A$ *has at least two pairs of incomparable states, then there exists* $c \in A \setminus \{a_0, b_0\}$ *such that the following statements are simultaneously true:*

(a) *for all* $e \bowtie f \in A$, $c \in \{e, f\}$, *and*

(b) *there exists* $\overline{x} \in X$ *such that* $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ *and for all* $e \in A$ *with* $c \bowtie e$, $e\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \not\subseteq c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\}$.

**Proof.** Let our starting assumption be that $\mathbf{A}$ has exactly one pair of different states $a_0, b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$, that $|A| \geq 5$, that $A$ has at least two pairs of incomparable states and that $\mathbf{A}$ cannot be embedded into a direct product of nondeterministic nilpotent automata. The proof of (a) is given by Corollary 5. To prove (b) assume to the contrary that for all $x \in X$, $cx^{\mathbf{A}} \cap \{a_0, b_0\} = \emptyset$ or there exists $e_x \in A$ with $c \bowtie e_x$ such that $e_xx^{\mathbf{A}} \cap \{a_0, b_0\} \subseteq cx^{\mathbf{A}} \cap \{a_0, b_0\}$.

Let $E$ be the set of all states of $A$ that are incomparable with $c$, i.e. $E = \{e_1, \ldots, e_k\}$. By (a) we know that there are no other pairs of incomparable states but $c \bowtie e_1, \ldots, c \bowtie e_k$, where $k \geq 2$. Thus, we can reformulate the converse of (b) as follows: there exist $k \geq 2$ and $e_1, \ldots, e_k \in A$ such that $c \bowtie e_r, r = 1, \ldots, k$, and for all $x \in X$, $cx^{\mathbf{A}} \cap \{a_0, b_0\} = \emptyset$ or there exists $i \in \{1, \ldots, k\}$ such that $e_ix^{\mathbf{A}} \cap \{a_0, b_0\} \subseteq cx^{\mathbf{A}} \cap \{a_0, b_0\}$ holds. Due to Corollary 6 $\mathbf{A}$ can be embedded isomorphically into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$ which contradicts the starting assumption. □

Now, we can prove our main result.

**Theorem 4** *A nondeterministic nilpotent automaton* $\mathbf{A} = (X, A, \delta)$ *with* $|A| \geq 2$ *cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than* $|A|$, *if and only if* $\mathbf{A}$ *has exactly one pair of different states* $a_0, b_0$ *for which* $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ *holds, for all* $x \in X$, *and one of the following statements is true:*

(1) *The partially ordered set* $(A, \leq)$ *is a chain,*

(2) $\mathbf{A}$ *has exactly one pair of incomparable states* $c \bowtie d$ *and there exists* $\overline{x} \in X$ *such that* $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$, $d\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ *and* $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq d\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\}$ *hold,*

(3) $\mathbf{A}$ *has at least two pairs of incomparable states and there exists* $c \in A$ *such that the following two statements are valid:*

   (a) *for all* $e \bowtie f \in A$, $c \in \{e, f\}$, *and*

   (b) *there exists* $\overline{x} \in X$ *such that* $c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset$ *and for all* $e \in A$ *with* $c \bowtie e$, $e\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\} \not\subseteq c\overline{x}^{\mathbf{A}} \cap \{a_0, b_0\}$ *holds.*

**Proof.** First, let us prove the necessity. Assume that $\mathbf{A}$ is a nondeterministic nilpotent automaton with $|A| \geq 2$ and that $\mathbf{A}$ cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$. The fact that there is exactly one pair of different states $a_0, b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$, is guaranteed by Corollary 1. Now, we will show that one of the statements (1),(2) or (3) holds. Examining $(A, \leq)$ we can distinguish the following three cases: there are no incomparable states at all, there is exactly one pair of incomparable states and there are at least two pairs of incomparable states.

If there are no incomparable states at all, then $(A, \leq)$ is a chain and (1) holds. If there is exactly one pair of incomparable states, then, by Corollary 3, (2) holds. If there are at least two pairs of incomparable states, then, by Theorem 3, (3) holds and the proof of the necessity is complete.

Now, let us prove the sufficiency. Assume that $\mathbf{A}$ is a nondeterministic nilpotent automaton with $|A| \geq 2$ that has exactly one pair of different states $a_0, b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$, and that also one of (1),(2) or (3) is true.

If (1) holds, then due to Corollary 2 $\mathbf{A}$ cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$. If (2) holds, then $\mathbf{A}$ cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$ due to Corollary 4. If (3) is true, then the fact that $\mathbf{A}$ cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$ follows from Theorem 2; and the proof of Theorem 4 is complete. □

The following statement presents the characterization of the isomorphic decomposability.

**Theorem 5** *A nondeterministic nilpotent automaton* $\mathbf{A} = (X, A, \delta)$ *with* $|A| > 2$ *can be embedded isomorphically into a direct product of nondeterministic nilpotent automata having fewer states than* $|A|$ *if and only if it fulfills one of the following conditions:*

(i) $|A| \geq 3$, $a_0$ *is the absorbent state,* $b_0$ *is a maximal element in* $(A \setminus \{a_0\})$ *and there exist* $a_1, b_1 \in A$, $a_1 \neq b_1$, $\{a_1, b_1\} \neq \{a_0, b_0\}$ *such that beside* $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$, $a_1x^{\mathbf{A}} = b_1x^{\mathbf{A}}$ *also holds, for all* $x \in X$.

(ii) **A** *(with $|A| \geq 4$) has exactly one pair of different states $a_0$, $b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$, and there exist $c \bowtie d \in A$ such that for all $x \in X$, the implication $(cx^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset \wedge dx^{\mathbf{A}} \cap \{a_0, b_0\} \neq \emptyset) \Rightarrow cx^{\mathbf{A}} \cap \{a_0, b_0\} = dx^{\mathbf{A}} \cap \{a_0, b_0\}$ holds.*

(iii) **A** *(with $|A| \geq 4$) has exactly one pair of different states $a_0$, $b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$, and there exist $c \bowtie d \in A$ and $e \bowtie f \in A$ such that $\{c, d\} \cap \{e, f\} = \emptyset$.*

(iv) **A** *(with $|A| \geq 5$) has exactly one pair of different states $a_0$, $b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$, and there are three pairwise incomparable states $c \bowtie d \in A$, $d \bowtie e \in A$ and $e \bowtie c \in A$.*

(v) **A** *(with $|A| \geq 5$) has exactly one pair of different states $a_0$, $b_0$ for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$, and there exist a natural number $k \geq 2$ and $c, e_1, \ldots, e_k \in A$ with $c \bowtie e_r, r = 1, \ldots, k$, such that, for all $x \in X$,*

$$cx^{\mathbf{A}} \cap \{a_0, b_0\} = \emptyset \text{ or } \exists i \in \{1, \ldots, k\} : e_ix^{\mathbf{A}} \cap \{a_0, b_0\} \subseteq cx^{\mathbf{A}} \cap \{a_0, b_0\}$$

*holds.*

**Proof.** We will prove the necessity by contradiction. One can see that in the class of nondeterministic nilpotent automata the following eqivalence holds:

$$\neg\big((\text{i}) \vee (\text{ii}) \vee (\text{iii}) \vee (\text{iv}) \vee (\text{v})\big) \Leftrightarrow \big(S \wedge ((1) \vee (2) \vee (3))\big),$$

where $S$ is the statement that there is exactly one pair of different states $a_0, b_0$ in **A** for which $a_0x^{\mathbf{A}} = b_0x^{\mathbf{A}}$ holds, for all $x \in X$ . It yields hence, that by Theorem 4, **A** cannot be embedded into a direct product of nondeterministic nilpotent automata having fewer states than $|A|$, which is a contradiction.

The sufficiency immediately follows from Theorem 1, Lemma 6, Lemma 8, Lemma 9 and Corollary 6, which imply (i), (ii), (iii), (iv) and (v), respectively. □

**Acknowledgement.** The author gratefully acknowledges the help and the support of her superviser, Dr. Balázs Imreh.

## References

[1] H.D. Burkhard, Zum Längeproblem homogener Experimente an determinierten und nichtdeterministischen Automaten, *EIK* **12** (1976), 301-306.

[2] Z. Ésik and B. Imreh, Subdirectly irreducible commutative automata, *Acta Cybernetica* **5** (1981), 251-260.

[3] B. Imreh, On finite nilpotent automata, *Acta Cybernetica* **5** (1981), 281-293.

[4] B. Imreh, On products of automata, *Papers on Automata Theory* **IV** (1982), 1-13.

[5] B. Imreh, On finite definite automata, *Acta Cybernetica* **7** (1984), 62-65.

[6] B. Imreh, Compositions of nondeterministic automata, Symposium on Semigroups, Formal Languages and Computer Systems, RIMS Kokyuroku **960** (1996), 44-53.

[7] P.H. Starke, *Abstrakte Automaten*, VEB Deutscher Verlag, Berlin, 1969

[8] G. H. Wenzel, Subdirect irreducibility and equational compactness in unary algebras, (A.f), *Arch. Math.* (Basel) **21** (1970), 256-263.

[9] M. Yoeli, Subdirectly irreducible unary algebras, *Amer. Math. Monthly* **74** (1967), 957-960.

*Received June, 1997*

Acta Cybernetica 13 (1998) 257-275.

# On the Information Content of Semi-Structured Databases

Mark Levene *

**Abstract**

In a semi-structured database there is no clear separation between the data and the schema, and the degree to which it is structured depends on the application. Semi-structured data is naturally modelled in terms of graphs which contain labels which give semantics to its underlying structure. Such databases subsume the modelling power of recent extensions of flat relational databases, to nested databases which allow the nesting (or encapsulation) of entities, and to object databases which, in addition, allow cyclic references between objects.

Due to the flexibility of data modelling in a semi-structured environment, in any given application there may be different ways in which to enter the data, but it is not always clear when the semantics are the same. In order to compare different approaches to modelling the data we investigate a measure of the *information content* of typical semi-structured databases in order to test whether such databases are *information-wise equivalent.* For the purpose of our investigation we use a graph-based data model, called the hypernode model, as our model for semi-structured data and formalise flat, nested and object databases as subclasses of hypernode databases.

We use formal language theory to define the context-free grammar induced by a hypernode database, and then formalise the *information content* of such a database as the language generated by this context-free grammar. Intuitively, the information content of a database provides us with a measure of how flexible the database is in modelling the information from different points of view. This enables us to prove the following results regarding the expressive power of databases: (1) in general, hypernode databases and thus semi-structured databases express the general class of context-free languages, (2) the class of flat databases expresses the class of finite languages whose words are of restricted length between one and four, (3) the class of nested databases expresses the class of finite languages, and (4) the class of object databases expresses the general class of regular languages.

We then define two hypernode databases to be *information-wise equivalent* if they generate the same context-free language. This allows us to prove

*Department of Computer Science University College London Gower Street London WC1E 6BT, U.K. email: mlevene@cs.ucl.ac.uk

the following results regarding the computational complexity of determining whether two databases are information-wise equivalent or inequivalent: (1) the problem of determining information-wise equivalence of hypernode databases and thus semi-structured databases is, in general, undecidable, (2) the problem of determining information-wise equivalence of flat databases can be solved in time polynomial in the size of the two databases, (3) the problem of determining information-wise inequivalence of nested databases is NP-complete, and (4) the problem of determining information-wise inequivalence of object databases is PSPACE-complete.

**Keywords** semi-structured databases, flat databases, nested databases, object databases, information content, information-wise equivalence

## 1 Introduction

In a traditional data model such as the relational model [Cod79] there is a clear separation between the schema and the data itself. Recently it has been recognised that there are applications where the data is *self-describing* in the sense that it does not come with a separate schema, and the structure of the data, when it exists, has to be inferred from the data itself. Such data is called *semi-structured*, the Web providing us with a rich source of semi-structured data to experiment with. Semi-structured data is also useful when integrating several databases, some of which may be structured. In such an integration process the data may come from several different sources and thus it may be difficult to constrain the integrated database to a single unifying schema. (For two recent surveys on semi-structured data, which provide more motivation and examples of semi-structured databases, see [Abi97, Bun97].)

Semi-structured data is naturally modelled in terms of graphs which contain labels that give semantics to the underlying structure [Abi97, Bun97]. Herein we use the *hypernode model* [PL94, LL95] as our data model for semi-structured data. The hypernode model is well-suited for this task as it is a graph-based data model that supports both complex objects of arbitrary structure and cyclic references between such objects. There have been several other previous proposals for graph-based data models [KV85, CM90, GPG90], all having the common thread of modelling objects as graphs (or subgraphs) which can reference each other. (See [BH90] for the graph-theoretic terminology.) Intuitively a *hypernode database* (or simply a database) is a collection of directed graphs, called hypernodes, each such hypernode modelling a unique object in the database which can reference other hypernodes.

Traditionally flat databases, as in the relational model, have been sufficient to model most applications, but recently, it has been proposed to extend the modelling power of flat databases to nested (or complex object) databases [AFS89] which allow the nesting (or encapsulation) of entities, and to object-oriented databases [Kim90] which, in addition, allow cyclic references between objects. (For the purpose of this paper we concentrate on the data modelling aspects of objects and ignore the wider issues of object-orientation in databases.) The hypernode model, as are other semi-structured models [Abi97, Bun97], is more general than the above extensions to the flat relational model, and in the sequel we will define suitable restrictions of hypernode databases that allow us to model flat, nested and object databases.

Due to the flexibility of data modelling in a semi-structured environment, in any given application there may be different ways in which to enter the data, but it is not always clear when the semantics are the same. In order to compare different approaches to modelling the data we would like to measure the *information content* of typical semi-structured databases in order to test whether such databases are *information-wise equivalent*. Moreover, for practical purposes it is essential to know what the computational cost of testing for such equivalence is, given that the database designer may have to choose one of the representations for the actual database. In particular, it would be useful to compare the expressive power of the above mentioned extensions to the basic flat data model in terms of the information content of the databases which are in the subclass of databases induced by each such extension.

We illustrate the modelling power of hypernode databases with a running example showing part of a hypernode database, where the labels of hypernodes represent unique identifiers of hypernodes in the database, providing the means by which hypernodes can reference each other. The hypernode shown in Table 1, which is labelled *EMPS*, models an entity set of employees, where each entity in *EMPS* is represented by an isolated node in the hypernode. Correspondingly, the hypernode shown in Table 2, which is labelled *ED2*, models the subset of employees in *EMPS* working in the Maths department. The hypernode shown in Table 3 models the information pertaining to *EMP1*, where each attribute and value of *EMP1* is represented by an arc in the hypernode. Similarly, the hypernode labelled *DEPTS*, shown in Table 4, models an entity set of departments and the hypernodes labelled *DEPT1* and *DEPT2*, shown in Tables 5 and 6, respectively, model the information pertaining to *DEPT1* and *DEPT2*. Note that in *DEPT2* the actual address of the department is missing and also that it has the additional attribute *faculty* which is missing from *DEPT1*. Finally, the hypernode labelled *WORKS*, shown in Table 7, models the relationship of an employee working in a department, where each employee and their department is represented by an arc in the hypernode. We observe that nesting (or alternatively encapsulation) is achieved by referencing another hypernode from within a hypernode; for example, *EMP1* and *EMP2* are nested in *DEPT1*, and *ED2* and *EMP3* are nested in *DEPT2*. Note the difference in modelling the set of employees working in a department from within *DEPT1* and *DEPT2*. In addition, we observe that cyclic references are achieved by two hypernodes referencing each other; for example a cyclic reference exists between *EMP1* and *DEPT1*, since *EMP1* references *DEPT1* and *DEPT1* references *EMP1*.

We use formal language theory [HU79] in order to reason about the information content of databases by showing that a hypernode database induces a context-free grammar and thus generates a context-free language. We then define two hypernode

| *EMPS* |
|---|
| *EMP1* |
| *EMP2* |
| *EMP3* |
| *EMP4* |

Table 1: The entity set EMPS

| *ED2* |
|---|
| *EMP3* |
| *EMP4* |

Table 2: A subset of EMPS

| EMP1 | | |
|---|---|---|
| (*attribute* | $\rightarrow$ | *value*) |
| *ename* | $\rightarrow$ | *john* |
| *dept* | $\rightarrow$ | *DEPT1* |
| *boss* | $\rightarrow$ | *EMP2* |

Table 3: The entity EMP1

| *DEPTS* |
|---|
| *DEPT1* |
| *DEPT2* |

Table 4: The entity set DEPTS

databases to be *information-wise equivalent* if they generate the same context-free language. In general, the problem of information-wise equivalence of hypernode databases and thus semi-structured databases is undecidable, since we show that the general class of hypernode databases expresses the general class of context-free languages. Therefore, we restrict our attention to three subclasses of hypernode databases: flat databases, nested databases and object databases, all of which are defined as suitable syntactic restrictions of hypernode databases.

(For an interesting example of the use of formal languages in database theory, see [Shm93] wherein it was shown that the problem of determining equivalence of Datalog queries is undecidable, by reducing the equivalence problem for context-free grammars to this problem; see also [Ull92] which looks into additional relationships between logic rules and formal languages.)

We prove the following results regarding the expressive power of different classes of databases:

1. The class of flat databases expresses the class of finite languages whose words are of length at most four.

2. The class of nested databases expresses the class of finite languages.

3. The class of object databases expresses the general class of regular languages.

We establish the following results regarding the computational complexity of determining whether two databases are information-wise equivalent or inequivalent:

1. The problem of determining information-wise equivalence of flat databases can be solved in time polynomial in the size of the two databases.

| DEPT1 | | |
|---|---|---|
| (*attribute* | $\rightarrow$ | *value*) |
| *dname* | $\rightarrow$ | *computing* |
| *emp* | $\rightarrow$ | $EMP1$ |
| *emp* | $\rightarrow$ | $EMP2$ |
| *head* | $\rightarrow$ | $EMP2$ |
| *address* | $\rightarrow$ | *london* |

Table 5: The entity DEPT1

| DEPT2 | | |
|---|---|---|
| (*attribute* | $\rightarrow$ | *value*) |
| *dname* | $\rightarrow$ | *maths* |
| *emp* | $\rightarrow$ | $ED2$ |
| *head* | $\rightarrow$ | $EMP3$ |
| *address* | | |
| *faculty* | $\rightarrow$ | *science* |

Table 6: The entity DEPT2

| $WORKS$ | | |
|---|---|---|
| $EMP1$ | $\rightarrow$ | $DEPT1$ |
| $EMP2$ | $\rightarrow$ | $DEPT1$ |
| $EMP3$ | $\rightarrow$ | $DEPT2$ |

Table 7: The relationship WORKS

2. The problem of determining information-wise inequivalence of nested databases is NP-complete.

3. The problem of determining information-wise inequivalence of object databases is PSPACE-complete.

It follows that in terms of information-content, object databases are strictly more expressive than nested databases and nested databases are strictly more expressive than flat databases. The interpretation that we place on the notion of being more expressive is that it affords us with more flexible means of modelling information. With respect to our running example, we have modelled the fact that $EMP1$ works in $DEPT1$ in three different ways, through the relationship $WORKS$, through the attribute *dept* in $EMP1$ and through the attribute *emp* in $DEPT1$. (We note that if we view primary keys in the relational model [Cod79] as object-identifiers, then relational databases can easily represent our notion of object databases.)

The problem of measuring the information capacity of database schemas was investigated in [Hul86] in the context of the relational data model, in [HY84, AH88] in the context of a complex objects data model and in [KV85] in the context of a graph-based data model, which supports cyclic references between objects. In [Hul86] several notions of equivalence are considered. The most restrictive measure is *query equivalence*, which informally holds between two database schemas when for any query on the first schema there is an equivalent query on the second schema and vice versa, and the least restrictive measure is *absolute equivalence*, which informally holds between two database schemas when there is a one-to-one correspondence between the number of objects that can be constructed by using sets of domain values over the attributes of both schemas. In the context of complex objects, a complete set of restructuring operations on database schemas that preserve absolute equivalence was exhibited in [HY84, AH88]. Moreover, it was shown in [KV85] that every database schema that has cyclic references is query equivalent (with respect to a well-defined query language which is given in [KV93]) to a database schema *without any* cyclic references; the concluding remark in [KV85] is: "But it is not clear that this measure is the ultimate one. We believe that the issue of cycles deserves further study".

The first difference in our work in comparison to the work mentioned above is that we concentrate on the information content of individual databases at the instance level rather than on the information content of database schemas. Thus we measure the information content of each database without regards to its schema. This difference is not so fundamental when the data has an underlying structure. As can be seen from the running example, a typical hypernode database may induce a database schema over which it is defined. On the other hand, since a hypernode database is only semi-structured it may not be possible to compare the information content of hypernode databases with reference to a fixed schema. The second difference is that we concentrate on the data modelling issues without reference to query equivalence, and as a result our definition of information-wise equivalence seems to be incomparable to the various definitions of equivalence referred to in the above work. This difference is fundamental since, for example, a flat database may be query equivalent to a nested database, in the sense that for every query defined on the flat database there is an equivalent query on the nested database and vice versa, but not information-wise equivalent to it according to our definition of information-wise equivalence. Intuitively, as demonstrated in the running example, nesting and/or cyclic referencing affords the database user with more flexible means of data modelling and therefore with several alternative ways to query the same information. Moreover, we also investigate the computational complexity of determining information-wise equivalence which was not dealt with in the work mentioned above.

The layout of the rest of the paper is as follows. In Section 2 we formalise the concept of a hypernode database, which comprises our model for semi-structured data. In Section 3 we introduce the necessary background material from formal language theory and formalise the notion of the information content of a database. In Section 4 we define flat, nested and object databases and prove our results concerning their expressive power. In Section 5 we prove our results concerning the computational complexity of determining whether two databases are information-wise equivalent. Finally, in Section 6 we give our concluding remarks.

## 2 Hypernode Databases

We first introduce the basic concepts pertaining to hypernode databases. We refer the reader to [PL94, LL95] for more detail on the hypernode model including a computationally complete query and update language operating on hypernode databases.

**Definition 2.1 (Hypernodes)** We assume two finite and disjoint sets of constants are available. Firstly we have the set of *labels* **L** whose elements are denoted by strings beginning with uppercase letters. Secondly we have the set of *atomic values* (or simply values) **A** whose elements are denoted by strings beginning with lowercase letters.

A *hypernode* is defined to be an equation of the form

$$H = (N, E),$$

where $H \in \mathbf{L}$ is termed the *defining label* of the hypernode and $(N, E)$ is a directed graph, termed *the graph of the hypernode*, such that $N \subseteq \mathbf{A} \cup \mathbf{L}$ is a set of nodes and $E \subseteq (N \times N)$ is a set of arcs. We denote the set of *isolated* nodes in $N$, i.e. the set of nodes which do not appear in any arc in E, by *isolated*($H$).

**Definition 2.2 (Hypernode databases)** A *hypernode database* (or simply a database), say HD, is a finite set of hypernodes satisfying the following two conditions:

**(H1)** No two (distinct) hypernodes in HD have the same defining label.

**(H2)** For any label, say $H$, in the node set of a graph of a hypernode in HD there exists a hypernode in HD whose defining label is $H$.

We denote the set of all labels that appear in the hypernodes in HD by LABELS(HD) and the set of all atomic values appearing in the hypernodes in HD by ATOMIC(HD). Moreover, we assume that a (possibly empty) set of distinguished labels in LABELS(HD) is associated with HD, which we denote by *root*(HD).

We note that condition H1 above corresponds to the *entity integrity* requirement of the relational data model [Cod79], since each hypernode can be viewed as representing a real-world entity. In object-oriented terminology labels are unique and serve as system-wide object-identifiers [Kim90], assuming that all of the hypernodes known to the system are stored in a single database. Similarly, condition H2 corresponds to the *referential integrity* requirement of the relational data model [Cod79], since it requires that only existing entities be referenced. The intuition behind the set of labels in *root*(HD) is that they represent the set of objects in the database through which all other objects in the database can be accessed.

The *Hypernode Accessibility Graph* (HAG) of a hypernode $H = (N, E)$ in a hypernode database HD (or simply the HAG of $H$, whenever HD is understood from context) is the directed graph telling us which hypernodes in HD are nested in the hypernode whose defining label is $H$, when considering nesting as a transitive relationship.

**Definition 2.3 (The accessibility graph of a hypernode)** The HAG of $H$, denoted by $(N_H, E_H)$, is the minimal directed graph which is constructed from hypernodes in HD as follows: $H \in N_H$, and if $H' \in N_H$ and $H' = (N', E') \in$ HD (such a hypernode must exist by condition H2), then $(\mathbf{L} \cap N') \subseteq N_H$ and $\forall n' \in (\mathbf{L} \cap N'), (H', n') \in E_H$.

A hypernode database HD is *acyclic* if for all $H \in root$(HD), the HAG of $H$ is acyclic, otherwise HD is *cyclic*.

We close this section with the definition of two operations on hypernode databases which, in the next section, are shown to preserve the information content of the database.

**Definition 2.4 (Renaming and duplication)** A hypernode database HD$'$ is the result of *renaming* some of the hypernodes in a hypernode database HD, if HD$'$ can be obtained from HD by renaming some of the labels in LABELS(HD) to distinct labels in $\mathbf{L}$ − LABELS(HD).

A hypernode database HD$'$ is the result of *duplicating* some of the hypernodes in HD, if HD$'$ is the union of HD and a hypernode database that is obtained by renaming some of the hypernodes in HD.

## 3 Information Content of Databases

We next present our notion of the information content of a database and briefly introduce the relevant definitions and results from the theory of context-free grammars [HU79].

**Definition 3.1 (Context-Free Grammar)** A *context-free grammar* (CFG) is a quadruple (V, T, P, $S$), where V and T are finite and disjoint sets of *variables* and *terminals*, P is a finite set of *productions* of the form $X \rightarrow \alpha$ such that $X$ is a variable and $\alpha$ is a finite and nonempty string of variables and terminals, and $S \in$ V is a distinguished variable called the *start symbol.*

A CFG (V, T, P, $S$) is said to be a *regular grammar* (RG), if each of the productions in P is either of the form $X \rightarrow \alpha Y$ or $X \rightarrow \alpha$, where $\alpha$ is a string of terminals (in the production $X \rightarrow \alpha Y$ $\alpha$ may be empty).

From now we will assume that G = (T, V, P, $S$) is a CFG and will refer to a finite string of variables and terminals as a *string* and to a finite string of terminals as a *word.* We define the *length* of a string $\alpha$ to be the number of symbols in $\alpha$.

**Definition 3.2 (Derivations)** If $\beta$ and $\gamma$ are strings and $X \rightarrow \alpha$ is a production in P, then $\beta X \gamma$ *directly derives* $\beta \alpha \gamma$ in G, written $\beta X \gamma \Rightarrow \beta \alpha \gamma$.

We say that a string $\alpha$ *derives* a string $\beta$ in G, written $\alpha \Rightarrow^* \beta$, if for some natural number $n \geq 0$

$$\alpha \Rightarrow \beta_1, \beta_1 \Rightarrow \beta_2, \ldots, \beta_n \Rightarrow \beta.$$

Thus $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$.

**Definition 3.3 (The language generated by a CFG)** The *context-free language* (or simply language) generated by G, denoted by $\mathcal{L}(G)$, is the set of all words that can be derived from the start symbol $S$ in G; a context-free language $\mathcal{L}(G)$ is said to be a *regular* language if G is an RG.

Two CFGs, $G_1$ and $G_2$, are *equivalent* written $G_1 \equiv G_2$ if $\mathcal{L}(G_1) = \mathcal{L}(G_2)$, otherwise they are *inequivalent.*

We note that according to Definition 3.1 the right-hand side of productions is always nonempty and thus *we only consider languages where the empty word is not a member of* $\mathcal{L}(G)$.

The next lemma allows us to simplify the productions in an RG.

**Lemma 3.1** Every RG, G, has an equivalent RG, G′, such that every production of G′ is either of the form $X \rightarrow Y$, $X \rightarrow aY$ or $X \rightarrow a$, where $X$ and $Y$ are variables and $a$ is a terminal.

*Proof.* The result easily follows by an induction on the length of $\alpha$ in productions of the form $X \rightarrow \alpha Y$. Suppose that the length of $\alpha$ is greater than one and $\alpha = \beta a$ for some terminal symbol $a$. Then, replace the production $X \rightarrow \beta a Y$ with the two productions $X \rightarrow \beta Z$ and $Z \rightarrow aY$, where $Z$ is a nonterminal not appearing in any other production in the resulting grammar. It is evident that the newly formed grammar is an RG and is equivalent to G. □

**Definition 3.4 (Chomsky Normal Form)** A CFG, G, is in *Chomsky Normal Form* (CNF) if all its productions are of the form $X \rightarrow YZ$ or $X \rightarrow a$, where $X, Y$ and $Z$ are variables and $a$ is a terminal.

The next theorem is a well-known result [HU79].

**Theorem 3.2** Every CFG, G, (such that $\mathcal{L}(G)$ does not contain the empty word) has an equivalent CFG, G′, which is in CNF and is equivalent to G.

Intuitively, the information content of a hypernode database HD is the context-free language that is generated by the CFG induced by G, and two hypernode databases are information-wise equivalent if they generate the same context-free language.

**Definition 3.5 (Information content of databases)** The CFG induced by a hypernode database HD, denoted by CFG(HD), is a quadruple (V, T, P, $S$), where $S \notin$ LABELS(HD), V = LABELS(HD) $\cup\{S\}$, T = ATOMIC(HD) and P is the smallest set of productions such that for every label $R \in root$(HD), $S \rightarrow R \in$ P, and for every hypernode $H = (N, E) \in$ HD, $H \rightarrow n \in$ P, if $n \in isolated(N)$ and $H \rightarrow n_1 n_2 \in$ P, if $(n_1, n_2) \in E$.

The language generated by HD, denoted by $\mathcal{L}$(HD), is the language generated by CFG(HD), i.e. $\mathcal{L}$(CFG(HD)).

The *information context* of a hypernode database HD is defined to be the language generated by HD. Two hypernode databases HD1 and HD2 are *information-wise equivalent* (or simply equivalent), denoted by HD1 $\equiv$ HD2, if CFG(HD1) $\equiv$ CFG(HD2), i.e. $\mathcal{L}$(HD1) = $\mathcal{L}$(HD2). Otherwise HD1 and HD2 are *information-wise inequivalent* (or simply inequivalent).

We note that the information content of a hypernode database may be the empty language, for example, if HD contains the single hypernode, $H = (\emptyset, \emptyset)$, or if *root*(HD) is empty.

The next proposition can be verified from Definitions 2.4 and 3.5.

**Proposition 3.3** Equivalence of hypernode databases is closed under renaming and duplication.

We next show that every CFG can be represented by a hypernode database.

**Definition 3.6 (The hypernode database representing a cfg)** The hypernode database *representing* a CFG, G = (V, T, P, $S$), denoted by DB(G), is constructed as follows. Firstly, by Theorem 3.2, G is converted into an equivalent CFG in CNF, which we also refer to as G. Secondly, we assume that V $\subseteq$ **L** and that T $\subseteq$ **A**, and say that the production X $\rightarrow$ YZ $\in$ P *induces* the hypernode X = ({Y, Z}, {(Y, Z)}) and the production X $\rightarrow a \in$ P *induces* the hypernode X = ($\{a\}$, $\emptyset$). Finally, DB(G) is the smallest set of hypernodes induced by the productions in P and where *root*(DB(G)) = $\{S\}$.

The next proposition is now immediate.

**Proposition 3.4** Two context-free grammars $G_1$ and $G_2$ are equivalent if and only if DB($G_1$) and DB($G_2$) are equivalent.

## 4 The Expressive Power of Database Classes

We are now ready to investigate the expressive power of various classes of hypernode databases in terms of the set of CFGs that they induce.

**Definition 4.1 (Expressive power of classes of hypernode databases)** A class of hypernode databases, **D**, is said to *express* a class of context-free languages, **C**, if for every hypernode database, HD $\in$ **D**, there is a CFG, G $\in$ **C**, such that $\mathcal{L}$(HD) $=$ $\mathcal{L}$(G), and for every CFG, G $\in$ **C**, there is a hypernode database, HD $\in$ **D**, such that $\mathcal{L}$(HD) $=$ $\mathcal{L}$(G).

A class, **D1**, of hypernode databases is *more expressive* than a class, **D2**, of hypernode databases, if the class of context-free languages that is expressed by **D1** is a proper superset of the class of context-free languages that is expressed by **D2**. Two classes of hypernode databases, **D1** and **D2**, are *equally expressive*, if both **D1** and **D2** express the same class of context-free languages.

The next lemma, which is an immediate consequence of Proposition 3.4 and Definition 3.6, establishes the expressive power of the general class of hypernode databases.

**Lemma 4.1** The general class of hypernode databases, and thus the general class of semi-structured databases, expresses the general class of context-free languages. □

For the rest of this section we investigate the expressiveness of various classes of hypernode databases, which correspond to flat, nested and object databases.

Our view of flat databases corresponds closely to Chen's binary entity-relationship model presented in [Che84], which in its essence captures the fundamental notions of the more general entity-relationship model [Che76, MM90, Teo94]. (For the purpose of this paper we do not address the concepts of specialisation and generalisation which are important notions in the entity-relationship model.)

**Definition 4.2 (Flat databases)** A *flat* database is a hypernode database HD such that the hypernodes $H = (N, E)$ in HD are restricted to be one of the following types:

1. An *entity set*, where $N \subseteq \mathbf{L}$ and $E = \emptyset$.
2. A *value set*, where $N \subseteq \mathbf{A}$ and $E = \emptyset$.
3. An *entity*, where $(N, E)$ is a bipartite graph [BH90], such that $N$ is partitioned into two nodes sets $N_1$ and $N_2$, with $N \subseteq \mathbf{A}$ and none of the nodes in $N_2$ are isolated, and there exists an entity set $H' = (M', \emptyset)$ in HD with $H \in M'$; the nodes in $N_1$ are called the *attributes* of the entity represented by $H$ and the nodes in $N_2$ are called the *values* of the entity represented by $H$.

4. A *relationship*, where $N = N_1 \cup N_2$, with $(N, E)$ having no isolated nodes, and such that there exist two entity sets $H_1 = (M_1, \emptyset)$ and $H_2 = (M_2, \emptyset)$ in HD such that $N_1 \subseteq M_1$ and $N_2 \subseteq M_2$.

Moreover, *root*(HD) contains a (possibly empty) subset of the set of defining labels of the entity sets, value sets and relationships in HD.

For example, the hypernodes shown in Tables 1, 2 and 4 represent entity sets, the hypernodes shown in Tables 8 and 9 represent entities, the hypernode shown in Table 7 represents a relationship, and the hypernode shown in Table 10 represents a value set.

| FLAT-EMP1 | | |
|---|---|---|
| (*attribute* | $\rightarrow$ | *value*) |
| *ename* | $\rightarrow$ | *john* |
| *dept* | $\rightarrow$ | *computing* |
| *boss* | $\rightarrow$ | *jack* |

Table 8: The entity FLAT-EMP1

| FLAT-DEPT1 | | |
|---|---|---|
| (*attribute* | $\rightarrow$ | *value*) |
| *dname* | $\rightarrow$ | *computing* |
| *emp* | $\rightarrow$ | *john* |
| *emp* | $\rightarrow$ | *jack* |
| *emp* | $\rightarrow$ | *jill* |
| *head* | $\rightarrow$ | *jack* |
| *address* | $\rightarrow$ | *london* |

Table 9: The entity FLAT-DEPT1

| $EMP - VALUE$ |
|---|
| *jack* |
| *jill* |
| *john* |

Table 10: The value set EMP-VALUE

It can easily be verified from Definition 4.2 that flat databases are acyclic and that such databases have no nesting of entities or relationships. Moreover, we observe that we represent attributes of entities by atomic values; see the hypernodes of the running example, shown in Tables 3, 5 and 6.

The next lemma thus follows from Definition 3.5 and 4.2.

**Lemma 4.2** The class of flat databases expresses the class of finite languages having nonempty words of length less than or equal to four.

We next define grouped databases which modify flat databases such that attribute values of entities are modelled by grouping them into value sets.

**Definition 4.3 (Grouped databases)** A *grouped* database HD is a variation of a flat database, where the definition of an entity is modified, as follows:

3. A hypernode $H = (N, E)$ is an *entity*, where $(N, E)$ is a bipartite graph, such that $N$ is partitioned into two nodes sets $N_1$ and $N_2$, with $N_1 \subseteq \mathbf{A}$, $N_2 \subseteq \mathbf{L}$ and none of the nodes in $N_2$ are isolated, there exists an entity set $H_1 = (M_1, \emptyset)$ in HD with $H \in M_1$, and for all $n \in N_2$, there exists a value set $H_2 = (M_2, \emptyset)$ such that $n = H_2$.

For example, the hypernode shown in Table 11 represents and entity in a grouped database.

| GROUPED-DEPT1 | | |
|---|---|---|
| (*attribute* | $\rightarrow$ | *value*) |
| *dname* | $\rightarrow$ | *computing* |
| *emp* | $\rightarrow$ | $EMP - VALUE$ |
| *head* | $\rightarrow$ | *jack* |
| *address* | $\rightarrow$ | *london* |

Table 11: The entity GROUPED-DEPT1

The next result follows from Definitions 4.2 and 4.3 on using Definition 4.1.

**Lemma 4.3** The classes of flat databases and grouped databases are equally expressive.

Our view of nested databases is to extend flat databases by allowing nesting of entities. In particular, we disallow the nesting of relationships, since otherwise by part (2) of Theorem 5.2, which is given in Section 5, equivalence of such databases would be intractable.

**Definition 4.4 (Nested databases)** A *nested* database HD is an extension of a flat database such that the definition of an entity is modified as follows, with the restriction that HD is acyclic:

3. A hypernode $H = (N, E)$ is an *entity*, where $(N, E)$ is a bipartite graph, such that $N$ is partitioned into two nodes sets $N_1$ and $N_2$, with $N_1 \subseteq \mathbf{A}$, $N_2 \subseteq \mathbf{A} \cup \mathbf{L}$ and none of the nodes in $N_2$ are isolated, there exists an entity set $H_1 = (M_1, \emptyset)$ in HD with $H \in M_1$, and for all $n \in N_2$, with $n \in \mathbf{L}$, there exists an entity set $H_2 = (M_2, \emptyset)$ such that either $n = H_2$ or $n \in M_2$.

For example, the hypdenodes shown in Tables 5 and 6 may represent entities in a nested databases. In this case the employee entities in the nested database may *not* reference either of the departments in order that HD be acyclic.

We observe that in nested databases we allow only the nesting of entity sets and entities. The next result follows from Definitions 3.5 and 4.4 on using Lemma 3.1, noting that any finite language can be generated by an RG.

**Lemma 4.4** The class of nested databases expresses the class of finite languages.

The next corollary follows from Lemmas 4.2 and 4.4.

**Corollary 4.5** The class of nested databases is more expressive than the class of flat databases.

Our view of object-oriented databases is to extend nested databases by allowing cycles as long as these do not involve relationships. This restriction is essential, since otherwise by part (1) of Theorem 5.2, which is given in Section 5, equivalence of such databases would be undecidable.

**Definition 4.5 (Object databases)** An *object* database is an extension of a nested database such that the database may be cyclic.

For example, the database shown in the running example in Section 1 is an object database.

We observe that as is the case of nested databases we disallow nesting of relationships in object databases.

**Lemma 4.6** The class of object databases expresses the general class of regular languages.

*Proof.* By Definitions 3.5 and 4.5 on using Lemma 3.1, it is easy to see that the class of object databases is at least as expressive as the general class of regular languages. It remains to show that relationships do not add expressive power to the class of object databases. By Definition 4.5 relationships are not nested and thus any derivation of a word which uses a production such as $R \rightarrow X_1 X_2$ must be of the form

$$S \Rightarrow R \Rightarrow X_1 X_2 \Rightarrow^* w,$$

where no other production of the form $R' \rightarrow X_1' X_2'$ is used in the derivation. Therefore, $w = w_1 w_2$, where for $i = 1$ and 2, $X_i \Rightarrow^* w_i$, implying that $w_i$ is a member of the language induced by the RG with start symbol $X_i$. The result now follows, since RGs are closed under concatenation [HU79]. □

The next corollary follows from Lemmas 4.4 and 4.6.

**Corollary 4.7** The class of object databases is more expressive than the class of nested databases.

## 5 The Complexity of Determining Equivalence of Databases

Herein we investigate the complexity of determining equivalence of hypernode databases for the classes of databases defined in Section 4. We assume that the reader is familiar with the notion of undecidability [HU79] and fundamental computational complexity classes NP (nondeterministic polynomial time), PSPACE (polynomial space) and NEXPTIME (nondeterministic exponential time) [GJ79]. (We define the *size* of a set $S$ to be the cardinality of a standard encoding of $S$.)

**Theorem 5.1** The following statements regarding the computational complexity of decision problems for CFGs are true:

(1) Equivalence of CFGs is undecidable [HU79, Theorem 8.12] (see also [HRS79]).

(2) Equivalence of CFGs which generate finite languages is NEXPTIME-hard [HRS79, Theorem 4.5].

(3) Inequivalence of RGs which generate finite languages is NP-complete [Hun73, Theorem 2.3].

(4) Inequivalence of RGs is PSPACE-complete [Hun73, Theorem 3.8].

The next theorem presents the results of this section.

**Theorem 5.2** The following statements regarding the computational complexity of decision problems for hypernode databases are true:

(1) Equivalence of hypernode databases is undecidable.

(2) Equivalence of acyclic hypernode databases is NEXPTIME-hard.

(3) Equivalence of flat databases can be tested in polynomial time in the size of the two databases.

(4) Inequivalence of nested databases is NP-complete.

(5) Inequivalence of object databases is PSPACE-complete.

*Proof.* (1) and (2) are immediate consequences of Proposition 3.4 and parts (1) and (2) of Theorem 5.1, noting that acyclic hypernode databases are finite.

(3) Let HD be a flat database. We show that the size of the language generated by HD is polynomial in the size of HD, implying the result. Let $m_1$ be the number of entities and value sets in HD, $m_2$ be the maximal number of arcs and isolated nodes in any entity or value set in HD, $m_3$ be the number of relationships in HD and $m_4$ be the maximal number of arcs in any relationship in HD. Now, let $m$ be the maximum of $m_i$, for $i = 1, 2, 3$ and 4. Thus the number of words in $\mathcal{L}(\text{HD})$ is bounded above by $3m^4$, since we need to count the number of words induced by

entity sets, value sets and relationships. The result now follows, since by Lemma 4.2, the length each word in $\mathcal{L}$(HD) is at most four.

(4) NP-hardness follows by Proposition 3.4 and part (3) of Theorem 5.1, on using Lemma 4.4. It remains to show that the equivalence problem for nested databases is in NP.

Given a nested database HD, the maximal length of words in $\mathcal{L}$(HD) is bounded above by twice the size of HD, since we disallow nesting of relationships. Now, let HD1 and HD2 be nested databases and nondeterministically guess a word, say $w$, whose length is less than or equal to the maximal length of words in either CFG(HD1) or CFG(HD2) and such that its atomic values are in ATOMIC(HD1) $\cup$ ATOMIC(HD2). The result now follows, since membership of a word $w$ in a CFG can be decided in polynomial time in the length of $w$ [HU79].

(5) PSPACE-hardness follows by Proposition 3.4 and part (4) of Theorem 5.1, on using Lemma 4.6. It remains to show that the inequivalence problem for object databases is in PSPACE.

Let HD be an object database. If there are no relationships in HD, the result follows from part (4) of Theorem 5.1, since it can easily be verified that CFG(HD) is an RG. Otherwise, suppose that due to a relationship whose defining label is $R$ we have the production $R \rightarrow X_1X_2$ in CFG(HD). Due to the fact that relationships cannot be nested, it follows that for $i = 1$ and 2, any derivation, $X_i \Rightarrow^* w$ of a word $w$, is induced by an RG whose start symbol is $X_i$. Thus a derivation $R \Rightarrow^* w$ of a word $w \in \mathcal{L}$(HD) can be viewed as the derivation of two words $w_1$ and $w_2$ such that $w_1w_2 = w$ and for $i = 1$ and 2, $w_i$ is a word in the RG induced by $X_i$. Moreover, $R$ can be chosen nondeterministically from the set of defining labels of relationships in HD. Thus the inequivalence of two object databases HD1 and HD2 reduces to the problem of finding a word $w = w_1w_2$, as above, which is a member of one of the languages $\mathcal{L}$(HD1) or $\mathcal{L}$(HD2), but is not a member of the other language. The result now follows by part (4) of Theorem 5.1, since both $w_1$ and $w_2$ can be derived by RGs in PSPACE. □

## 6 Concluding Remarks

We have investigated the information content of semi-structured databases and shown that the general class of databases expresses the general class of context-free languages, the class of object databases expresses the general class of regular languages, the class of nested databases expresses the class of finite languages, and the class of flat databases expresses the class of finite languages whose words are of length less than or equal to four. Moreover, we have shown that testing the equivalence of hypernode databases and thus semi-structured databases is, in general, undecidable, but for object databases it is PSPACE-complete, for nested databases it is NP-complete and for flat databases it is polynomial time in the size of the input. Our results support the view that relationships are *not* entities, since otherwise, if we allow relationships to be nested within entities, by parts (1) and (2) of Theorem 5.2 determining equivalence of nested databases would be intractable and determining equivalence of object databases would be undecidable.

The interpretation we place on the notion of being more expressive is that it affords us with more flexible means of modelling information. (We refer the reader back to the running example given in the introduction to verify this statement.) From the user's point of view this flexibility provides several alternative ways of viewing and querying the same information; for example, the fact that an employee works in a department can be modelled in three different ways. Moreover, this flexibility may be an advantage for the query optimiser, when there are several alternative routes to obtain an answer to a query. Although testing for equivalence of object and nested databases is, in general, intractable we can provide restructuring operations as in [HY84, AH88] in order to transform a database into an equivalent one having a different structure. The formulation of a complete set of restructuring operations that preserve information-wise equivalence for object and nested databases is an open problem.

We now briefly outline, through an example, an extension to measure the navigation capacity of hypernode databases, and thus semi-structured databases. Let HD1 be a hypernode database comprising the hypernodes with defining labels A and B shown in Tables 12 and 13, respectively, and let HD2 be a hypernode database comprising the hypernodes with defining labels C and D shown in Tables 14 and 15, respectively. It can easily be verified that both $\mathcal{L}(\text{HD1}) = \mathcal{L}(\text{HD2}) = \{a, b\}$, and thus HD1 and HD2 are information-wise equivalent. In this case the nesting of hypernodes does not increase the information-content of the database. Despite this equivalence, from a navigation point of view HD1 is less expressive than HD2, since in HD1 we cannot directly navigate from A to B or from B to A, while in HD2 it is possible to navigate either directly from C to D or directly from D to C. Thus information content on its own is insufficient to measure expressiveness of a database from the point of view of navigation. We suggest to utilise the hypernode accessibility graph (HAG) for this purpose (see Definition 2.3). In our example, it is evident that with respect to navigation HD2 is more expressive than HD1, since HD1 $\equiv$ HD2 and the HAGs of A and B are subgraphs of the HAGs of C and D, respectively, up to an appropriate renaming of labels.

| *A* |
|---|
| *a* |

Table 12: The hypernode labelled A

| *B* |
|---|
| *b* |

Table 13: The hypernode labelled B

| *C* |
|---|
| *a* |
| *D* |

Table 14: The hypernode labelled C

| *D* |
|---|
| *b* |
| *C* |

Table 15: The hypernode labelled D

Another open problem is to extend our formalism to deal with integrity constraints such as keys and cardinality constraints. Finally, we mention that an important application of our formalism is in software engineering process modelling [CKO92], as it was shown in [LSO97] that the graph-based approach of the hypernode model provides a suitable platform for such process modelling.

## References

[Abi97] S. Abiteboul. Querying semi-structured data. In *International Conference on Database Theory*, pages 1-18, Delphi, 1997. Invited talk.

[AFS89] S. Abiteboul, P.C. Fischer and H.-J. Schek, editors. *Nested Relations and Complex Objects in Databases*, volume 361 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1989.

[AH88] S. Abiteboul and R. Hull. Restructuring hierarchical database objects. *Theoretical Computer Science*, 62:3-38, 1988.

[BH90] F. Buckley and F. Harary. *Distance in Graphs.* Addison-Wesley, Redwood City, Ca., 1990.

[Bun97] P. Buneman. Semistructured data. *Proceedings of ACM Symposium on Principles of Database Systems*, Tucson, Az., 1997. Invited talk.

[Che76] P.P-S. Chen. The Entity-Relationship model - towards a unified view of data, *ACM Transactions on Database Systems*, 1:9-36, 1976.

[Che84] P.P-S. Chen. An algebra for a directional binary entity-relationship model. In *Proceedings of IEEE International Conference on Data Engineering*, pages 37-40, Los Angeles, 1984.

[CKO92] W. Curtis and M.I. Kellner and J. Over. Process Modelling. *Communications of the ACM*, 35:79-90, 1992.

[CM90] M.P. Consens and A.O. Mendelzon. Graphlog : A visual formalism for real life recursion. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 406-416, Nashville, Tn., 1990.

[Cod79] E.F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4:397-434, 1979.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, New York, 1979.

[GPG90] M. Gyssens and J. Paredaens and D. Van Gucht.A graph-oriented object database model. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 417-424, Nashville, Tn., 1990.

[HRS79] H.B. Hunt III, D.J. Rosenkrantz and T.G. Szymanski.On the equivalence, containment, and covering problems for regular and context-free languages. *Journal of Computer and System Sciences*, 12:222-268, 1976.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, Ma., 1979.

[Hul86] R. Hull. Relative information capacity of simple relational database schemata. *SIAM Journal on Computing*, 15:856-886, 1986.

[Hun73] H.B. Hunt III. On the time and tape complexity of languages I. *Proceedings of ACM Symposium on Theory of Computing*, pages 10-19, Austin, Tx., 1973.

[HY84] R. Hull and C.K. Yap. The format model: A theory of database organization. *Journal of the ACM*, 31:518-537, 1984.

[Kim90] W. Kim. *Introduction to Object-Oriented Databases*, MIT Press, Cambridge, Ma., 1990.

[KV85] G.M. Kuper and M.Y. Vardi. On the expressive power of the logical data model. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 180-187, Austin, Tx., 1985.

[KV93] G.M. Kuper and M.Y. Vardi. The logical data model. *ACM Transactions on Database Systems*, 18:379-413, 1993.

[LL95] M. Levene and G. Loizou. A graph-based data model and its ramifications. *IEEE Transactions on Knowledge and Data Engineering*, 7:809-823, 1995.

[LSO97] M. Levene, L. Scott and R. Offen. A framework for seamless conceptual data and process modelling. Research Report, Research Report No. 97/5, Joint Research Centre for Advanced Systems Engineering, CSIRO - Macquarie University, 1997.

[MM90] V.M. Markowitz and J.A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Transactions on Software Engineering*, 16:777-790, 1990.

[PL94] A. Poulovassilis and M. Levene. A nested-graph model for the representation and manipulation of complex objects. *ACM Transactions on Information Systems*, 12:35-68, 1994.

[Shm93] O. Shmueli. Equivalence of Datalog queries is undecidable. *Journal of Logic Programming*, 15:231-241, 1993.

[Teo94] T.J. Teorey. *Data Modelling & Design: The Fundamental Principles* Morgan-Kaufmann, San Francisco, Ca., 2nd edition, 1994.

[Ull92] J.D. Ullman. The interface between language theory and database theory. In J.D. Ullman, editor, *Theoretical Studies in Computer Science*, pages 133-151, Boston, Ma., 1992. Academic Press

*Received October, 1997*

Acta Cybernetica 13 (1998) 277-304.

# Limitations of Rule Triggering Systems for Integrity Maintenance in the Context of Transition Specifications

Klaus-Dieter Schewe [*] Bernhard Thalheim [†]

**Abstract**

Integrity Maintenance is considered one of the major application fields of rule triggering systems (RTSs). In the case of a given integrity constraint being violated by a database transition these systems trigger repairing actions. Then it is necessary to guarantee the termination of the RTS, its determinacy and the consistency of final states. Transition specifications provide some kind of dynamic semantics requiring certasin effects on database states to occur. In the context of transition specifications integrity maintenance has to cope with the additional problem of effect preservation.

Limitations of RTSs with respect to this extended problems are investigated. It will be shown that for any set of constraints there exist non-repairable transitions, which depend on the closure of the constraint set. This implies that integrity maintenance by RTSs is only possible, if the constraint implication problem is decidable. Even if unrepairable transitions are excluded, this does not prevent the RTS to produce undesired behaviour.

Analyzing the behaviour of RTSs leads to the definition of *critical paths* in associated rule hypergraphs and the requirement of such paths being absent. It will be shown that this requirement can be satisfied if the underlying set of constraints is *stratified*, but this notion turns out to be too strong to be also necessary. A sufficient and necessary condition for the absence of critical paths is obtained, if sets of constraints are required to be *locally stratified*.

**Keywords:** active databases, integrity maintenance, transition specifications

## 1 Introduction

Active databases (ADBs) aim at extending relational (or object oriented) DBMS by *rule triggering systems* (RTSs), i.e. by sets of rules which on a given *event* and in

[*]Technical University of Clausthal, Computer Science Institute, Erzstr. 1, 38678 Clausthal-Zellerfeld, Germany, e-mail: schewe@informatik.tu-clausthal.de

[†]Cottbus Technical University, Computer Science Institute, Karl-Marx-Str. 17, 03044 Cottbus, Germany, e-mail: thalheim@informatik.tu-cottbus.de

the case of a *condition* being satisfied trigger *actions* on the database (ECA-rules). Events can be external events, time conditions or internal events resulting from operations on the database. Conditions are usually given by boolean queries that have to be evaluated against the database. The action part consists of a sequence of basic operations to insert, delete or update tuples (or objects respectively) in the database.

The current research on ADBs (see e.g. [4]) is dominated by implementational aspects, whilst foundations of RTSs are seldom approached. The work in [2, 3, 5, 10, 11] and partly in [4] considers the problem to enforce database integrity by the use of RTSs. The results concern the generation of repairing ECA-rules and partly the analysis of the resulting RTS. This analysis concentrates on the *termination* of the rule system, the independence of the final database state from the chosen selection order of the rules (*determinacy*) and on *consistency*.

These properties are orthogonal to one another. Therefore, it is reasonable to investigate the consistency requirement alone and to neglect for the moment the other two requirements.

Besides the specification of static semantics by the use of integrity constraints there is an increasing interest in integrating dynamic aspects [1]. In general, not all transitions between consistent states – with respect to the given set of integrity constraints – will be allowed, e.g. allowing transitions from any consistent state to the empty database state are not very useful. As a consequence, given a specification of transitions, the consistency requirement for RTSs occurs to be too weak. In a sense to be made precise in the sequel the final state resulting from an execution of the RTS should be "close" to its inconsistent starting state.

Note that this formulation already assumes that there is a final state and that this state is unique, i.e. the termination and determinacy properties are already tacitly assumed. This formulation could be weakened for non-determinate RTSs requiring the "closeness" for one or all of the possible final states. It could also be weakened for non-terminating RTSs requiring the "closeness" only, if a final state exists. This underlines again the orthogonality of the three basic requirements for integrity maintenance.

In the presence of transition specifications an inconsistent starting state for the RTS always occurs as the result of a specified transition. Therefore, one reasonable additional requirement is to preserve the effect of the transition at hand. In this case not only the static semantics expressed by the integrity constraints is sacrificed, i.e. only consistent states will be reached, but also the dynamic semantics expressed by transitions. This means that the allowed state pairs will always imply the effects of prespecified transitions.

There may be other equally reasonable requirements how to handle transition specifications. E.g., we may want only to undo some or even all effects of a transition or only preserve as many effects as possible depending on some measure on effects.

In this paper we analyze limitations of the rule triggering approach for integrity maintenance under the additional requirement to preserve the effects of transitions. For a given set of constraints in implicational normal form we first investigate the existence of non-repairable transitions. These are determined by the closure of the constraint set. It turns out that the decidability of the constraint implication problem is necessary for integrity maintenance by RTSs.

Next we analyze, how to obtain RTSs that definitely repair constraint violations by a (repairable) transition without invalidating its intended effect. Given an RTS we first associate with it a *rule hypergraph* which corresponds to the possible sequences of triggered rules. Next we define *critical trigger paths* in these hypergraphs that correspond to the propagation of conditions. Indeed it can be shown that the existence of a single critical trigger path makes the RTS work incorrectly for at least one transition.

Finally, we analyze constraint sets in order to detect, whether it is possible to define an RTS of repairing actions such that the critical trigger paths in its associated hypergraph can only invalidate unrepairable transitions. For this we first introduce *stratified* constraint sets that satisfy this condition. Since the converse is not true, we finally weaken the concept to *locally stratified* constraint sets which gives a necessary and sufficient conditions for the RTS to work correctly.

## 2 Non-Repairable Transitions

In the following we consider the relational datamodel with *integrity constraints* given by formulae in *implicative normal form*

$$\mathcal{I} \equiv p_1(\vec{x}_1) \wedge \ldots \wedge p_n(\vec{x}_n) \Rightarrow q_1(\vec{y}_1) \vee \ldots \vee q_m(\vec{y}_m) \ , \tag{1}$$

with predicate symbols $p_i$, $q_j$, which correspond either to a relation of the schema or are comparison predicates $(=, \neq, \leq, <)$. Variables on the left hand side are assumed to be universally quantified, those occurring only on the right hand side are assumed to be existentially quantified (with all $\forall$-quantifiers preceding all $\exists$-predicates).

Moreover, we assume that there is at least one relation symbol on the left hand side of each such $\mathcal{I}$. Moreover, $\mathcal{I}$ should contain at least two relation symbols. The first restriction guarantees the empty database to be consistent, i.e. it satisfies all constraints $\mathcal{I}$, and the second one just states that there is no explicit constraint which requires a relation $p$ to be always empty. We may always write $\mathcal{I}$ in clausal form.

For the ECA-*rules* we use the notation ON $\langle$event$\rangle$ IF $\langle$condition$\rangle$ DO $\langle$action$\rangle$ with $\langle$event$\rangle$ corresponding to an internal event, i.e. an insert- or delete-operation. $\langle$condition$\rangle$ is a formula to be evaluated against the actual database state, written as a negation $\neg\mathcal{I}$ for a constraint $\mathcal{I}$ in implicative normal form (1). $\langle$action$\rangle$ is a sequence of basic insert- or delete-operations to be triggered, i.e. to be executed if the event occurred and the condition is satisfied.

In this paper, the assumed execution model for ECA-rules relies on a deferred modus, i.e. the system RTS of rules is started after finishing a transition. Furthermore, we do not assume any order of the rules. Instead of this, the execution model relies on demonic non-determinism, i.e. if the events of several rules $r_1, \ldots, r_n$ occur and their conditions evaluate to *true*, any of these $r_i$ may be executed unless it is undefined.

Given a single constraint $\mathcal{I}$ in implicative normal form (1) we already get minimum requirements for repairing rules. If a relation symbol $p$ occurs on the left hand side (right hand side) of (1), then each insert- (delete-)operation on $p$ may violate (1), hence give rise to event-parts. The corresponding condition-part is simply $\neg\mathcal{I}$. However, for the action-part there are still several alternatives.

We call a system of ECA-rules *complete* iff for all these cases of events and conditions there exists at least one repairing rule, i.e. whenever the rule is selectable in some database state, the execution of the action part will establish $\mathcal{I}$ as a postcondition. However, we exclude those rules, which simply invalidate the event. For transitions we simply consider sequences of insert- and delete-operations.

Let us first demonstrate the insufficiency of a naive RTS approach by a simple example. In "real" applications the situation of Example 1 will not occur in such an obvious way, but there are always implied and in general not detectable constraints leading to analogous problems as shown in [7].

EXAMPLE 1 Take two unary relations $p$ and $q$ and the constraints $\mathcal{I}_1 \equiv p(x) \Rightarrow q(x)$ and $\mathcal{I}_2 \equiv p(x) \wedge q(x) \Rightarrow \mathit{false}$. This implies $p$ to be always empty, hence insertions into $p$ should be abolished. Then we obtain the following repairing rules:

$$\begin{aligned}
R_1 &: \text{ON insert}_p(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO insert}_q(x)\\
R_2 &: \text{ON delete}_q(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO delete}_p(x)\\
R_3 &: \text{ON insert}_p(x) \text{ IF } \neg\mathcal{I}_2 \text{ DO delete}_q(x)\\
R_4 &: \text{ON insert}_q(x) \text{ IF } \neg\mathcal{I}_2 \text{ DO delete}_p(x)
\end{aligned}$$

If we try to execute a transition insert$_p(a)$ on a database state satisfying $q(a)$, then we successively trigger the rules $R_3$ and $R_2$ with the effect of only deleting $a$ in $q$. This contradicts the original intention of the transition. □

In order to analyze the unintended behaviour in Example 1 consider a set $\Sigma$ of constraints in implicational normal form. Let $\Sigma^*$ denote the (semantic) *closure*, i.e. $\Sigma^* = \{\mathcal{I} \mid \Sigma \models \mathcal{I}\}$. Now let $\mathcal{I} \in \Sigma^*$ be non-trivial, i.e. it does not hold in all database states. Write $\mathcal{I}$ in implicational normal form

$$\mathcal{I} \equiv p_1(\vec{x}_1) \wedge \ldots \wedge p_n(\vec{x}_n) \Rightarrow q_1(\vec{y}_1) \vee \ldots \vee q_m(\vec{y}_m)$$

and let $p_{i_1}, \ldots, p_{i_k}$ and $q_{j_1}, \ldots, p_{j_\ell}$ denote the relation symbols on the left and right hand sides of $\mathcal{I}$ respectively. We may define a transition $T$ by

$$\text{delete}_{q_{j_1}}(\vec{y}_{j_1}); \ldots; \text{delete}_{q_{j_\ell}}(\vec{y}_{j_\ell}); \text{insert}_{p_{i_1}}(\vec{x}_{i_1}); \ldots; \text{insert}_{p_{i_k}}(\vec{x}_{i_k}) \ .$$

If we start $T$ with values for the $\vec{x}_i$ and $\vec{y}_j$ such that the additional conditions on the left hand side of $\mathcal{I}$ are satisfied, whilst the additional conditions on the right hand side are not, $T$ will always reach a database state satisfying $\neg\mathcal{I}$. This effect of

$T$ is intentional and hence the only reasonable approach to integrity maintenance in this case is to disallow such transitions.

More formally, the *effect* of a transition $T$ in a state $\sigma$ is given by the strongest (with respect to $\Rightarrow$) formula $\mathbf{Eff}_\sigma(T) = \psi$ such that $\models_\sigma wp(T)(\psi)$ holds. Here $wp(T)(\psi)$ denotes the weakest precondition of $\psi$ under the transition $T$, i.e. starting $T$ in initial state $\sigma$ will reach a final state $\tau$ satisfying $\psi$.

Since we only consider sequences of insertions and deletions, $\mathbf{Eff}_\sigma(T)$ can always be written as a conjunction of literals, i.e. in negated implicational normal form, with the positive literals corresponding to insertions and the negative ones to deletions. In addition, we may consider the effect of a sequence $T; RTS$, where $T$ is a transition and $RTS$ a system of rules. We say that $RTS$ *invalidates the effect* of $T$ iff $\not\models \mathbf{Eff}_\sigma(T) \wedge \mathbf{Eff}_\sigma(T; RTS)$ holds for some state $\sigma$.

Then it is justified to call a transition $T$ *repairable* with respect to the constraint set $\Sigma$ iff $\neg\mathbf{Eff}_\sigma(T) \notin \Sigma^*$ holds for at least one state $\sigma$. Then a complete terminating system $RTS$ of ECA-rules always invalidates the effect of a non-repairable transition $T$. Hence the problem is to detect (and exclude) non-repairable transitions. In order to decide whether a given transition $T$ is repairable or not, we must be able to decide, whether $\neg\mathbf{Eff}_\sigma(T)$ is in the closure $\Sigma^*$. Hence the implication problem for constraints must be decidable.

**Proposition 1** Let $\Sigma$ be a set of constraints. The problem to decide, whether a transition $T$ is repairable with respect to $\Sigma$ is equivalent to the constraint implication problem for $\Sigma$, i.e. the problem to decide, whether a given constraint $\mathcal{I}$ is a member of $\Sigma^*$ or not. □

Proposition 1 defines the first limit on integrity maintenance by rule triggering systems. In the following sections we shall concentrate on repairable transitions.

Note that our treatment ignores the termination problem. Non-terminating transitions have to be excluded as well, but this problem is independent from the repairability problem, since non-termination of RTSs occurs as an orthogonal problem.

## 3 Critical Paths

Let us ask, whether we can always find a complete set of repair rules for all repairable transitions. For this we introduce the notions of *associated hypergraphs* and *critical trigger paths*.

**Definition 2** Let $\mathcal{S} = \{p_1, \dots, p_n\}$ be a relational database schema and $RTS = \{R_1, \dots, R_m\}$ a system of ECA-rules on $\mathcal{S}$. Then the *associated rule hypergraph* $(V, E)$ is constructed as follows:

- $V$ is the disjoint union of $\mathcal{S}$ and $RTS$. We then talk of $\mathcal{S}$-vertices and $RTS$-vertices respectively.

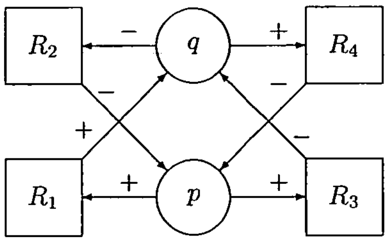

Figure 1: Associated Rule Hypergraph

- If $R \in RTS$ has event-part $Ev$ on $p \in \mathcal{S}$ and actions on $p_1, \dots, p_k$, then we have a hyperedge from $p$ to $\{R\}$ labelled by $+$ or $-$ depending on $Ev$ being an insert or delete, and a hyperedge from $\{R\}$ to $\{p_1, \dots, p_k\}$ analogously labelled by $k$ values $+$ or $-$. □

Figure 1 shows the associated rule hypergraph of Example 1 in which case we have a simple graph.

Definition 2 ignores the condition part of the rules. These come into play if we consider critical trigger paths in associated hypergraphs. These are defined in several steps starting from paths in the associated hypergraph which correspond to possible sequences of ECA-rules with respect only to their event- and action-parts. Secondly we attach formulae to the $\mathcal{S}$-vertices in the path in such a way that pre- and postconditions of the involved rules are expressed. Then we talk of *trigger paths*.

A maximal trigger path with contradicting initial and final condition will then be called *critical*. Then imagine a transition with an effect implied by the initial formula, i.e. that there is an initial state such that running the transition in this state results in a state which satisfies the initial condition of the trigger path. If we execute this transition followed by the rule triggering system along the critical trigger path will then turn the effect of the transition into its opposite. This means that the $RTS$ invalidates the effect of at least one transition.

**Definition 3** Let $G = (V, E)$ be the rule hypergraph associated with a system $RTS$ of rules. A *trigger path* in $G$ is a sequence $v_0, e_1, v'_1, e'_1, \dots, e'_\ell, v_\ell$ of vertices and hyperedges with the following conditions:

- $v_i \in \mathcal{S}$ holds for all $i = 0, \dots, \ell$,
- $v'_i \in RTS$ holds for all $i = 1, \dots, \ell$,
- $e_i$ is a hyperedge from $v_{i-1}$ to $v'_i$ and
- $e'_i$ is a hyperedge from $v'_i$ to $V_i$ with $v_i \in V_i$ and the same label as $e_{i+1}$.

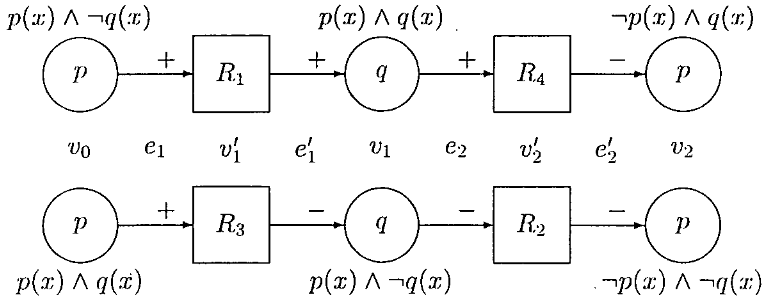

Figure 2: Critical Trigger Paths

We call $\ell$ the *length* of the trigger path.

In addition we associate with each vertex $v_i \in S$ $(i = 0, \dots, \ell)$ a formula $\varphi_i$ in negated implication normal form such that $\models \varphi_i \Rightarrow cond(v'_{i+1})$ holds for the condition part $cond(v'_{i+1})$ of rule $v'_{i+1} \in RTS$ and $\models \varphi_i \Rightarrow wp(A_{i+1})(\varphi_{i+1})$ holds for the action-part $A_{i+1}$ of rule $v'_{i+1}$ $(i = 0, \dots, \ell - 1)$. Furthermore, there is no $e_{\ell+1} \in E$ from $v_\ell$ to $v'_{\ell+1}$ with the same label as $e'_\ell$ such that $\models \varphi_\ell \Rightarrow cond(v'_{\ell+1})$ holds.

Then a trigger path is *critical* iff $\models \neg(\varphi_0 \wedge \varphi_\ell)$ holds. Such a critical trigger path is called *admissible* iff there is a consistent state $\sigma$ and a repairable transition $T$ such that $\mathbf{Eff}_\sigma(T) \Leftrightarrow \varphi_0$ holds. □

Critical trigger paths for the associated rule hypergraph in Figure 1 are sketched in Figure 2. Note that in this case both critical trigger paths are not admissible. If a critical trigger path is not admissible, then only a non-repairable transition can be invalidated by running the rules in the trigger path. Since we exclude non-repairable transitions, we only have to consider admissible trigger paths. After these remarks we are able to prove our first result.

**Proposition 4** Let $RTS$ be a complete set of rules associated with a set $\Sigma$ of constraints and let $G = (V, E)$ be the associated rule hypergraph. Then $G$ contains an admissible critical trigger path iff there exists a consistent database state $\sigma$ and a repairable transition $T$ such that executing $T$ in $\sigma$ and consecutively running $RTS$ invalidates the effect of $T$ without leaving the database unchanged.

*Proof.* Let us first assume that $G$ contains an admissible critical trigger path. Let $\varphi_0, \dots, \varphi_\ell$ denote the formulae associated with the $S$-vertices in this trigger path.

**Case 1.** Assume that $e_1$ is labelled by $+$. Then $\varphi_0$ contains at least one positive literal $p(\vec{x})$. Let $\sigma$ be a consistent state and $T$ a repairable transition such that $\mathbf{Eff}_\sigma(T)$ is given by $\varphi_0$. We may assume that $\models_\sigma \neg p(\vec{x})$ holds and that the final action in $T$ in an insertion into $p$. If we start $T$ in the initial state $\sigma$, then the resulting state satisfies $\varphi_0$.

$T$ followed by the $RTS$ may then result in a state $\tau$ satisfying $\varphi_\ell$. Hence the effect of $T; RTS$ in $\sigma$ is given by $\varphi_\ell$. Since $\models \neg(\varphi_0 \wedge \varphi_\ell)$ holds by the definition of critical trigger paths, this implies that $RTS$ invalidates the effect of $T$. Furthermore, $\tau$ is consistent with respect to all constraints in $\Sigma$, since $RTS$ is complete and there is no hyperedge $e_{\ell+1}$ from $v_\ell$ to some $v'_{\ell+1} \in RTS$ with the same label as $e'_\ell$ such that $\models \varphi_\ell \Rightarrow cond(v'_{\ell+1})$ holds.

It remains to show $\tau \neq \sigma$. If this does not hold, we get $\models_\sigma \varphi_\ell$ and consequently there exists some $\psi$ such that $\varphi_\ell \Leftrightarrow \neg p(\vec{x}) \wedge \psi$ and $\varphi_0 \Leftrightarrow p(\vec{x}) \wedge \psi$ hold. This implies $\ell > 1$, because otherwise the rule $v'_1$ would have the form ON $\text{insert}_p(\vec{x})$ IF $\neg\mathcal{I}$ DO $\text{delete}_p(\vec{x})$ , which we excluded.

If $\ell > 1$ holds, there is at least one other literal $q(\vec{y})$ (or $\neg q(\vec{y})$) in $\varphi_0$ such that $\text{delete}_q(\vec{y})$ (or $\text{insert}_q(\vec{y})$ respectively) occurs in the action-part of $v'_1$. Then we may consider the admissible critical trigger path $v_1, e_2, \dots, v_\ell$ of length $\ell - 1$ instead. Following the argumentation above, we may choose $\sigma$ and $T$ in such a way that $\models_\sigma \neg q(\vec{y})$ (or $\models_\sigma q(\vec{y})$ respectively) holds. This implies $\tau \neq \sigma$ as required.

**Case 2.** If $e_1$ is labelled by $-$, then $\varphi_0$ contains a literal $\neg p(\vec{x})$. Thus, we have to consider a transition $T$ containing $delete_p(\vec{x})$ as its final action and a consistent state $\sigma$ with $\models_\sigma p(\vec{x})$ and $\mathbf{Eff}_\sigma(T) \Leftrightarrow \varphi_0$. Then we may apply the same arguments as for case 1.

Conversely, assume that there is no admissible critical trigger path. Let $T$ be a repairable transition and $\sigma$ a database state which is consistent with respect to $\Sigma$. Now start $T$ in $\sigma$ and assume that the resulting state $\sigma'$ is not consistent. Then consider a trigger path of finite length such that $\models_{\sigma'} \varphi_0$ holds. The consecutive execution of the rules in this trigger path will result in a state $\tau$ satisfying $\varphi_\ell$. Thus, we have $\mathbf{Eff}_\sigma(T) \Leftrightarrow \varphi_0$ and $\mathbf{Eff}_\sigma(T; RTS) \Leftrightarrow \varphi_\ell$.

According to our assumption, the used trigger path cannot be critical, i.e. $\varphi_\ell \wedge \varphi_0$ is satisfiable. Hence $RTS$ does not invalidate the effect of $T$. □

## 4 Stratified Constraint Sets

According to the result in Proposition 4 we may ask for constraint sets that allow to define complete RTSs which exclude admissible critical trigger paths in their associated hypergraphs. Let us start with a simple example.

EXAMPLE 2 Take again two unary relations $p$ and $q$ and the constraints $\mathcal{I}_1 \equiv p(x) \Rightarrow q(x)$ and $\mathcal{I}_2 \equiv q(x) \Rightarrow p(x)$ which implies $p$ to be always equal to $q$. Then we obtain the following repairing rules:

$$\begin{aligned}
R_1 &: \text{ ON insert}_p(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO insert}_q(x) \\
R_2 &: \text{ ON delete}_q(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO delete}_p(x) \\
R_3 &: \text{ ON insert}_q(x) \text{ IF } \neg\mathcal{I}_2 \text{ DO insert}_p(x) \\
R_4 &: \text{ ON delete}_p(x) \text{ IF } \neg\mathcal{I}_2 \text{ DO delete}_q(x)
\end{aligned}$$

In this case there are no admissible critical paths in the associated rule hypergraph. We omit further details. □

Let us now investigate the reason for the absence of admissible critical trigger paths in Example 2. This leads us to the notion of a *stratified* set of constraints.

The motivation behind this is as follows: In Example 2 insertions (deletions) on a relation $p$ only trigger insertions (deletions) on $q$ and vice versa. This should be sufficient for not invalidating a once established effect. The corresponding constraints can therefore be grouped together.

**Definition 5** Let $\Sigma$ be a set of constraints in implicative normal form (1) on a schema $\mathcal{S}$. The $\Sigma$ is called *stratified* iff we have a partition $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ with pairwise disjoint constraint sets $\Sigma_i$ called *strata* such that the following conditions are satisfied:

(i) If $L$ is a literal on the left hand side (right hand side) of some constraint $\mathcal{I} \in \Sigma_i$, then all constraints $\mathcal{J} \in \Sigma$ containing a literal $L'$ on the right hand side (left hand side) such that $L$ and $L'$ are unifiable also lie in stratum $\Sigma_i$.

(ii) All constraints $\mathcal{I}$, $\mathcal{J}$ containing unifiable literals $L$ and $L'$ either on the left or the right hand side must lie in different strata $\Sigma_i$ and $\Sigma_j$. □

Now we can prove in general that stratified constraint sets always give rise to RTSs without admissible critical trigger paths in the associated rule hypergraph.

**Proposition 6** Let $\Sigma$ be a stratified constraint set on a schema $\mathcal{S}$. Then there exists a complete RTS such that for any repairable transition $T$ on $\mathcal{S}$ the RTS does not invalidate the effect of $T$.

*Proof.* Given a constraint $\mathcal{I}$ in implicative normal form (1), then each relation symbol $p_i$ on the left hand side gives rise to rules

$$\begin{aligned}
&\text{ON } \mathit{insert}_{p_i}(\vec{x}_i) \text{ IF } \neg\mathcal{I} \text{ DO } \mathit{insert}_{q_j}(\vec{y}_j) \quad , \\
&\text{ON } \mathit{insert}_{p_i}(\vec{x}_i) \text{ IF } \neg\mathcal{I} \text{ DO } \mathit{delete}_{p_j}(\vec{y}_j)
\end{aligned}$$

with relation symbols $q_j$ occurring on the right hand side and $p_j$ $(j \neq i)$ on the left hand side of $\mathcal{I}$. Similarly, each predicate symbol $q_j$ on the right hand side gives rise to rules

$$\begin{aligned}
&\text{ON } \mathit{delete}_{q_j}(\vec{y}_j) \text{ IF } \neg\mathcal{I} \text{ DO } \mathit{insert}_{q_i}(\vec{y}_j) \quad (i \neq j) \ , \\
&\text{ON } \mathit{delete}_{q_j}(\vec{y}_j) \text{ IF } \neg\mathcal{I} \text{ DO } \mathit{delete}_{p_i}(\vec{y}_j)
\end{aligned}$$

This defines a complete set $RTS$ of rules. Now assume there exists a critical trigger path $v_0, e_1, v_1', e_1', \dots, e_\ell', v_\ell$ in the associated rule hypergraph. Each $RTS$-vertex $v_i'$ corresponds to a constraint $\mathcal{I}_i \in \Sigma$. Since $e_i'$ and $e_{i+1}$ are equally labelled corresponding to the action- or event-part respectively, the construction of the rules above implies $\mathcal{I}_i$ and $\mathcal{I}_{i+1}$ to lie in the same stratum ($i = 0, \dots, \ell - 1$).

However, the condition $\models \neg(\varphi_0 \wedge \varphi_\ell)$ implies that $\varphi_0$ contains a literal $L$, $\varphi_\ell$ its negation, hence the construction of rules implies $\mathcal{I}_1$ and $\mathcal{I}_\ell$ to lie in different strata. Hence, there are only critical trigger paths of length $\ell = 1$.

According to our construction of $RTS$ this implies $\models \varphi_0 \Rightarrow \neg\mathcal{I}$ to hold for some $\mathcal{I} \in \Sigma$. Thus, $\neg\varphi_0 \in \Sigma^*$ holds. Due to the definition of admissible critical trigger paths and the definition of repairable transitions, we conclude that the trigger paths of length $\ell = 1$ cannot be admissible. Then the proposition follows from Proposition 4. □

Finally, we may ask for cases, where stratified constraint sets occur. Recall from [6] that a relational database schema $\mathcal{S}$ with constraint set $\Sigma$ is in *Entity-Relationship normal form* (ERNF) – and hence is equivalent to an ER-schema – iff

- all inclusion constraints in $\Sigma$ are key-based and non-redundant,
- there is no cycle of inclusion constraints in $\Sigma$,
- each relation schema $R \in \mathcal{S}$ is in BCNF with respect to the functional dependencies in $\Sigma^*$ and
- there are only inclusion and functional dependencies in $\Sigma^*$.

If a relational database schema $\mathcal{S}$ with constraint set $\Sigma$ is in ERNF, then it is easy to see that $\Sigma$ is stratified.

**Corollary 7** Let $\mathcal{S}$ be a database schema in ERNF with respect to the constraint set $\Sigma$. Then $\Sigma$ is stratified. □

Hence, following the design approach of Mannila and Räihä in [6] – if this is sufficient for the application – leads to schemata without any problems concerning consistency enforcement by RTSs.

EXAMPLE 3 Let us look at the following constraints

$$\begin{aligned} \mathcal{I}_1 &: \quad p(x,y) \Rightarrow q(x,z) \qquad \text{and} \\ \mathcal{I}_2 &: \quad q(x,z) \wedge q(y,z) \Rightarrow x = y \end{aligned}$$

Then this set of constraints corresponds to the Entity-Relationship diagram [9] in Figure 3. Obviously, the constraint set is stratified. □

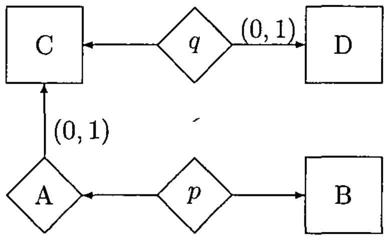

Figure 3: Entity-Relationship constraints

## 5 An Algorithm for Checking Stratification

Before we analyze the converse of Proposition 6 and present the weaker notion of locally stratified constraint sets, let us first concentrate on an algorithm for checking stratification and its complexity. For this we consider the set

$$BW \quad = \quad \{\top, \bot\} \cup (\mathbb{N} - \{0\}) \cup \{\{j_1, \dots, j_n\} \mid n \geq 1, j_k \in \mathbb{N} - \{0\}\} \quad .$$

In the algorithm we successively add labels from $BW$ to constraints. A label $i \in \mathbb{N}$ for a constraint $\mathcal{I}$ is used to indicate that $\mathcal{I}$ must lie in the stratum $\Sigma_i$. A label $\{j_1, \dots, j_n\}$ indicates that $\mathcal{I}$ must not lie in $\Sigma_{j_k}$ for $k = 1, \dots, n$. $\bot$ represents no information and $\top$ an inconsistent assignment of stratum numbers.

For a more convenient terminology we call an element of $BW$ *black*, if it is in $(\mathbb{N} - \{0\}) \cup \{\top\}$, otherwise *white*. Furthermore, we use a commutative, associative binary operation $\oplus$ on $BW$ defined by

$$\begin{aligned}
x \oplus \bot &= x \ , \\
x \oplus \top &= \top \ , \\
i \oplus j &= \begin{cases} i & \text{if } i = j \\ \top & \text{otherwise} \end{cases} \ , \\
\{j_1, \dots, j_n\} \oplus \{k_1, \dots, k_m\} &= \{j_1, \dots, j_n\} \cup \{k_1, \dots, k_m\} \quad \text{and} \\
i \oplus \{j_1, \dots, j_n\} &= \begin{cases} \top & \text{if } i = j_k \text{ for some } k \in \{1, \dots, n\} \\ i & \text{otherwise} \end{cases}
\end{aligned}$$

**Algorithm 8 ((Stratification Check))**

**Input:** a set $\Sigma = \{\mathcal{I}_1, \dots, \mathcal{I}_n\}$ of constraints in clausal form $\mathcal{I}_i = L_{i,1} \vee \dots \vee L_{i,n_i}$

**Output:** a boolean value $b$

**Method:**

```
VAR $gather$ : ARRAY 1...$n$ OF $BW$ ,
        $mb, mb' : BW$ ;
    BEGIN
        FOR $i = 1$ TO $n$ DO
            $gather(i) := \bot$
        ENDFOR ;
        $b := true$ ;
        $mb := 1$ ;
        WHILE $\Sigma \neq \emptyset$ DO
            CHOOSE $i_0 \in \{1, \dots, n\}$ WITH $\mathcal{I}_{i_0} \in \Sigma$ AND $gather(i_0)$ is maximal ;
            $\Sigma := \Sigma - \{\mathcal{I}_{i_0}\}$ ;
            IF $gather(i_0)$ is white
            THEN $gather(i_0) := mb$ ;
                $mb := mb + 1$
            ENDIF ;
            $mb' := gather(i_0)$ ;
            FOR $j = 1$ TO $n_{i_0}$ DO
                FOR ALL $\mathcal{I}_k \in \Sigma$ DO
                    FOR $\ell = 1$ TO $n_{i_k}$ DO
                        IF $L_{i_0,j}$ and $L_{k,\ell}$ are unifiable AND $gather(i_0) \neq \top$
                        THEN $gather(k) := gather(k) \oplus \{gather(i_0)\}$
                        ELSIF $L_{i_0,j}$ and $\sim L_{k,\ell}$ are unifiable
                        THEN $gather(k) := gather(k) \oplus gather(i_0)$
                        ENDIF
                    ENDFOR
                ENDFOR
            ENDFOR
        ENDDO ;
        FOR $i = 1$ TO $n$ DO
            IF $gather(i) = \top$
            THEN $b := false$
            ENDIF
        ENDFOR ;
        RETURN($b$)
    END                                                                    □
```

We have to check that the algorithm is correct. Then we analyze its time complexity. Before we do this let us first look at a simple example.

EXAMPLE 4 Consider the following constraints:

$$\begin{aligned}
\mathcal{I}_1 &= \neg p(x) \vee \neg q(x) \vee r(x) \vee s(x) \quad , \\
\mathcal{I}_2 &= \neg q(x) \vee r(x) \vee \neg t(x) \quad , \\
\mathcal{I}_3 &= p(x) \vee \neg r(x) \quad , \\
\mathcal{I}_4 &= \neg s(x) \vee t(x) \quad \text{and} \\
\mathcal{I}_5 &= q(x) \vee \neg t(x) \quad .
\end{aligned}$$

Table 1: Stratification Check

| | $L_{1,1}$ | $L_{1,2}$ | $L_{1,3}$ | $L_{1,4}$ | $L_{3,2}$ | $L_{3,1}$ | $L_{2,1}$ | $L_{2,2}$ | $L_{2,3}$ | $L_{4,1}$ | $L_{4,2}$ | $L_{5,2}$ | $L_{5,1}$ | *gather* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | | | | | | | | | | 1 |
| 3 | 1 | | 1 | | 1 | 1 | | | | | | | | 1 |
| 2 | | {1} | {1} | | 1 | | $\top$ | $\top$ | $\top$ | | | | | $\top$ |
| 4 | | | | 1 | | | | | $\top$ | $\top$ | $\top$ | | | $\top$ |
| 5 | | 1 | | | | | $\top$ | | | | $\top$ | $\top$ | $\top$ | $\top$ |
| | | $\mathcal{I}_1$ | | | $\mathcal{I}_3$ | | | $\mathcal{I}_2$ | | $\mathcal{I}_4$ | | $\mathcal{I}_5$ | | $b = false$ |

Then consider Table 1.

Each row corresponds to a constraint $\mathcal{I}_i$ and lists the values added to *gather*$(i)$ during the excution of the algorithm. The chosen order of the constraints in the algorithm is $\mathcal{I}_1$, $\mathcal{I}_3$, $\mathcal{I}_2$, $\mathcal{I}_4$, $\mathcal{I}_5$. Then $b$ will become *false* and hence $\Sigma$ is not stratifiable. □

Let us now address the correctness of Algorithm 8.

**Proposition 9** Let $\Sigma$ be a set of constraints. Then $\Sigma$ is stratifiable iff Algorithm 8 applied to the input $\Sigma$ computes the output $b = true$.

*Proof.* Let us first assume that $\Sigma$ is stratified. Let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ be a decomposition into strata and assume that the $\Sigma_i$ are taken minimal with the required properties. We use induction on $n$.

For $n = 1$ there are no unifiable literals $L$ and $L'$ in different constraints $\mathcal{I}, \mathcal{J} \in \Sigma$. Hence *gather*$(i)$ will become 1 for alle $i$ and we obtain $b = true$.

For $n > 1$ we may assume without loss of generality that some constraint in $\Sigma_1$ will be chosen first. Then, due to our minimality assumption, we get *gather*$(i) = 1$ for all $\mathcal{I}_i \in \Sigma_1$, whereas *gather*$(j)$ will be white for all $\mathcal{I}_j \notin \Sigma_1$. Thus, all constraints in $\Sigma_1$ will be chosen first.

Since *gather*$(j)$ was white for $\mathcal{I}_j \notin \Sigma_1$ and *gather*$(i) = 1$ for $\mathcal{I}_i \in \Sigma_1$ before chosing the first constraint in $\Sigma_2 \cup \ldots \cup \Sigma_n$, we may apply the induction hypothesis to $\Sigma_2 \cup \ldots \cup \Sigma_n$, which gives *gather*$(j) \neq \top$ for all $\mathcal{I}_j \notin \sigma_1$. This implies $b = true$ as claimed in the proposition.

Conversely, assume that the algorithm produces the result $b = true$. Then we must have *gather*$(i) \in \mathbb{N} - \{0\}$. Define $\Sigma_k = \{\mathcal{I}_i \in \Sigma \mid gather(i) = k\}$. Assume that the partition $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ does not satisfy the conditions for strata in Definition 5. Then there are two possible cases:

(i) There are literals $L$ and $L'$ in constraints $\mathcal{I}_i \in \Sigma_k$ and $\mathcal{I}_j \in \Sigma_\ell$ with $k \neq \ell$ such that $L$ and $\sim L'$ are unifiable. Suppose that $\mathcal{I}_i$ is chosen first by the algorithm. Then $k$ will be added to *gather*$(j)$, which gives *gather*$(j) = \top$ contradicting our assumption.

(ii) There are unifiable literals $L$ and $L'$ in constraints $\mathcal{I}_i, \mathcal{I}_j \in \Sigma_k$. If $\mathcal{I}_i$ is chosen first by the algorithm, $\{k\}$ will be added to *gather*$(j)$, which also gives *gather*$(j) = \top$ contradicting our assumption.

Thus $\Sigma_1 \cup \ldots \cup \Sigma_n$ is a partition into strata, which completes the proof. □

**Proposition 10** Let $\Sigma$ be a set of constraints in clausal form, $n = \#\Sigma$, $k$ the maximal arity of predicate symbols occurring in constraints $\mathcal{I} \in \Sigma$ and let $\ell$ be the maximum number of literals in these constraints. Then the time complexity of Algorithm 8 for checking, whether $\Sigma$ is stratified is in $O(k \cdot \ell^2 \cdot n^2)$.

*Proof.* The initialization and the final computation of $b$ can both be done in $O(n)$ steps.

In the inner FOR-loop the test for unifiability can be done in $O(k)$ steps, since there are no function symbols. All other operations have a complexity in $O(1)$. Hence the inner FOR-loop has a total complexity in $O(k)$. This loop is executed $\ell' \cdot \ell''$ times, where $\ell'$ is the number of literals in the chosen constraint $\mathcal{I}_{i_0}$ and $\ell''$ is the total number of literals in the remaining constraints. If $\mathcal{I}_{i_0}$ is the $i$'th literal chosen by the algorithm, this can be estimated by $\ell^2 \cdot (n - i)$.

Since each $\mathcal{I} \in \Sigma$ will be chosen by the algorithm, the outer WHILE-loop will be executed $n$ times. This gives the total complexity in

$$O(n) + O(\ell^2 \cdot \sum_{i=1}^{n}(n-i)) \cdot O(k) + O(n) \quad = \quad O(k \cdot \ell^2 \cdot n^2)$$

as claimed in the proposition. □

It is easy to see that $n \cdot \ell$ can be replaced by the total number $u = \sum_{i=1}^{n} n_i$ of literals in $\Sigma$ with $u < n \cdot \ell$. Thus, the time complexity of the stratification checking algorithm 8 is in $O(k \cdot u^2)$.

From Proposition 6 we know that active mechanisms can be effectively applied, if the constraint set is stratified. In particular, this holds for schemata in ERNF [6], which are equivalent to Entity-Relationship schemata. From Proposition 10 we know that a stratification check can be done efficiently.

## 6 Locally Stratified Constraint Sets

Unfortunately, the converse of Proposition 6 does not hold, as seen in the next example. The reason for this is that in the proof of Proposition 6 we considered all repairing rules for a given constraint, whereas the constraint set in Example 5 allows to select only a subset thus gaining the required result without loosing the completeness of the RTS.

EXAMPLE 5 Take three unary relations $p$ and $q$ and the constraints $\mathcal{I}_1 \equiv p(x) \wedge r(x) \Rightarrow q(x)$, $\mathcal{I}_2 \equiv q(x) \Rightarrow p(x)$ and $\mathcal{I}_3 \equiv p(x) \Rightarrow r(x)$. It is easy to see that this constraint set is not stratified.

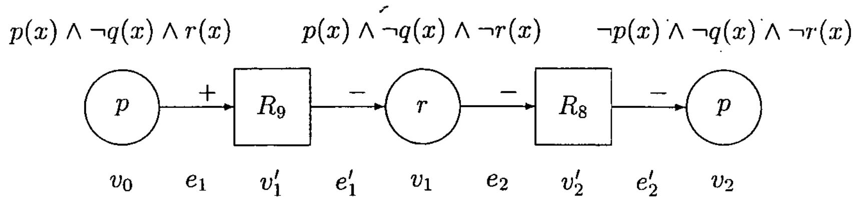
$p(x) \wedge \neg q(x) \wedge r(x)$
$p(x) \wedge \neg q(x) \wedge \neg r(x)$
$\neg p(x) \wedge \neg q(x) \wedge \neg r(x)$
$p$
$+$
$R_9$
$-$
$r$
$-$
$R_8$
$-$
$p$
$v_0$
$e_1$
$v'_1$
$e'_1$
$v_1$
$e_2$
$v'_2$
$e'_2$
$v_2$

Figure 4: An Admissible Critical Trigger Path

However, we may consider the following system of ECA-rules:

$$\begin{array}{lcl}
R_1 & : & \text{ON insert}_p(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO insert}_q(x) \\
R_2 & : & \text{ON delete}_q(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO delete}_p(x) \\
R_3 & : & \text{ON insert}_r(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO insert}_q(x) \\
R_4 & : & \text{ON delete}_q(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO delete}_r(x) \\
R_5 & : & \text{ON insert}_q(x) \text{ IF } \neg\mathcal{I}_2 \text{ DO insert}_p(x) \\
R_6 & : & \text{ON delete}_p(x) \text{ IF } \neg\mathcal{I}_2 \text{ DO delete}_q(x) \\
R_7 & : & \text{ON insert}_p(x) \text{ IF } \neg\mathcal{I}_3 \text{ DO insert}_r(x) \\
R_8 & : & \text{ON delete}_r(x) \text{ IF } \neg\mathcal{I}_3 \text{ DO delete}_p(x)
\end{array}$$

We dispense with showing that there are no admissible critical trigger paths in the associated rule hypergraph.

Note that the construction in the proof of Proposition 6 would result in two more rules corresponding to insertions:

$$\begin{array}{lcl}
R_9 & : & \text{ON insert}_p(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO delete}_r(x) \\
R_{10} & : & \text{ON insert}_r(x) \text{ IF } \neg\mathcal{I}_1 \text{ DO delete}_p(x)
\end{array}$$

These give rise to admissible critical trigger paths. The one shown in Figure 4 allows to invalidate the effect of the repairable transition insert$_p(x)$. □

The constraint set in Example 5 is not stratified, but nevertheless the associated RTS does not invalidate the effect of repairable transitions. This shows that a constraint set need not be stratified to allow a reasonable rule behaviour. Indeed, replacing $\mathcal{I}_1$ in the example by $\mathcal{I}'_1 \equiv p(x) \Rightarrow q(x)$ gives an equivalent constraint set, which is stratified. However, equivalence of constraint sets is undecidable in general. Therefore, we introduce the weaker notion of being *locally stratified*. In this case we shall construct RTSs which only contain a subset of the set of rules constructed in the proof of Proposition 6.

**Definition 11** Let $\Sigma$ be a set of constraints in implicative normal form on a schema $\mathcal{S}$.

A *labelled subsystem* consists of a subset $\Sigma' = \{\mathcal{I} \in \Sigma \mid \rho_L(\mathcal{I}) \text{ is defined }\}$ together with a set of clauses $\Sigma'' = \{\rho_L(\mathcal{I}) \mid \mathcal{I} \in \Sigma'\}$ and a literal $L$ (the *label*) such that each constraint $\mathcal{I} \in \Sigma'$ can be written as the disjunction $\rho_L(\mathcal{I}) \vee \mathcal{I}'$ with $\models \mathcal{I}' \Rightarrow L$.

Here $\rho_L(\mathcal{I})$ is defined iff the negation $\sim L$ does not occur in $\mathcal{I}$ (written as a clause). Then $\rho_L(\mathcal{I})$ results from $\mathcal{I}$ by omission of the literal $L$ if the result contains at least two literals. Otherwise $\rho_L(\mathcal{I})$ is simply $\mathcal{I}$. We call $\mathcal{I}'$ the *label part* and $\rho_L(\mathcal{I})$ the *label-free part* of the constraint $\mathcal{I}$. If $L$ is understood from the context, we drop the subscript and write $\rho$ instead of $\rho_L$.

A labelled subsystem $(\Sigma', \Sigma'', L)$ is called *stratified* iff the set $\Sigma''$ is stratified in the sense of Definition 5 or locally stratified as defined below.

The constraint set $\Sigma$ is called *locally stratified* iff $\Sigma = \Sigma'_1 \cup \ldots \cup \Sigma'_n$ with stratified labelled subsystems $(\Sigma'_i, \Sigma''_i, L_i)$ $(i = 1, \ldots, n)$ such that for each constraint $\mathcal{I} \in \Sigma'_i$ and each literal $L$ occurring in its label part with respect to $\Sigma_i$ there exists another $j$ with $\mathcal{I} \in \Sigma'_j$ and $L$ occurring in its label-free part of $\mathcal{I}$ with respect to $\Sigma_j$. □

EXAMPLE 6 For the constraint set $\Sigma$ in Example 5 we obtain the partition into $\Sigma'_1 = \{\mathcal{I}_1, \mathcal{I}_3\}$ and $\Sigma'_2 = \{\mathcal{I}_1, \mathcal{I}_2\}$.

For the first of these we have the label $L_1 \equiv \neg p(x)$ and the label-free parts defined by $\rho_{L_1}(\mathcal{I}_1) \equiv q(x) \vee \neg r(x)$ and $\rho_{L_1}(\mathcal{I}_3) \equiv \mathcal{I}_3$.

For $\Sigma'_2$ we get the label $L_2 \equiv \neg r(x)$ and the label-free parts $\rho_{L_2}(\mathcal{I}_1) \equiv \neg p(x) \vee q(x)$ and $\rho_{L_2}(\mathcal{I}_2) \equiv \mathcal{I}_2$.

This shows that the constraint set in Example 5 is indeed locally stratified. □

Note that each stratified constraint set $\Sigma$ is also locally stratified. In this case we define $depth(\Sigma) = 0$. If $\Sigma$ is locally stratified by a partition $\Sigma = \Sigma'_1 \cup \ldots \cup \Sigma'_n$, we define $depth(\Sigma) = \max_{i=1}^{n} depth(\Sigma''_i) + 1$. We call $depth(\Sigma)$ the *depth* of the locally stratified constraint set $\Sigma$.

Finally, we can strengthen Proposition 6 now dealing with locally stratified constraint sets. This condition turns out to be sufficient and also necessary for the absence of admissible critical trigger paths.

**Theorem 12** Let $\Sigma$ be a constraint set on a schema $\mathcal{S}$. Then $\Sigma$ is locally stratified iff there exists a complete RTS such that for any repairable transition $T$ the RTS does not invalidate the effect of $T$.

*Proof.* First assume that $\Sigma$ is locally stratified. Let the labelled subsystems in the partition be $(\Sigma'_i, \Sigma''_i, L_i)$ for $i = 1, \ldots, n$. We shall use induction on the depth of $\Sigma$. For $depth(\Sigma) = 0$ we are done by Proposition 6.

Let us now consider the case $depth(\Sigma) = 1$. As in the proof of Proposition 6 we construct an RTS for $\Sigma$. Since each $\Sigma''_i$ is stratified in the sense of Definition 5, we first construct a rule system $RTS'_i$ with respect to $\Sigma''_i$ as in the proof of Proposition 6. The condition parts in these rules have the form $\neg\rho_{L_i}(\mathcal{I})$ for $\mathcal{I} \in \Sigma'_i$. Then let $RTS_i$ result from $RTS'_i$ by changing all condition parts replacing $\neg\rho_{L_i}(\mathcal{I})$ by $\neg\mathcal{I}$. Finally, take $RTS = \bigcup_{i=1}^{n} RTS_i$.

Due to the last property in the definition of locally stratified constraint sets in Definition 11 we conclude that $RTS$ is complete.

Now consider a critical trigger path $v_0, e_1, v'_1, e'_1, v_1, \dots, e'_\ell, v_\ell$ in the rule hypergraph associated with $RTS$. Without loss of generality assume $v'_1 \in RTS_1$. According to Proposition 4 we have to show that this trigger path is not admissible.

We use induction on the length $\ell$ of this critical trigger path. For $\ell = 1$ we may use the same argument as in the proof of Proposition 4. Therefore, assume $\ell > 1$ and take a state $\sigma$ with $\models_\sigma \Sigma$ and a transition $T$ with $\models \mathbf{Eff}_\sigma(T) \Leftrightarrow \varphi_0$. Then we have to show that $T$ is not repairable.

Assume that $T$ is repairable. Then there exists a state $\tau$ with $\models_\tau \Sigma$ such that $\neg\mathbf{Eff}_\tau(T) \notin \Sigma^*$. We shall derive a contradiction from this.

For this regard the critical trigger path $v_1, e_2, v'_2, e'_2, v_2, \dots, e'_\ell, v_\ell$ of length $\ell - 1$. By induction it is not admissible. If $A_1$ is the action in the rule $v'_1$, we get $\models \mathbf{Eff}_\sigma(T; A_1) \Leftrightarrow \varphi_1$ and $T; A_1$ cannot be repairable. In particular, this implies $\neg\mathbf{Eff}_\tau(T; A_1) \in \Sigma^*$.

Since $A_1$ is a simple insertion or deletion, we get $\models \neg\mathbf{Eff}_\tau(T) \Leftrightarrow \varphi \wedge L$ and $\models \neg\mathbf{Eff}_\tau(T; A_1) \Leftrightarrow \varphi \wedge \sim\!L$ for some literal $L$ and its negation $\sim\!L$. From this we conclude $\varphi \in \Sigma^*$ and $\sim\!L \in \Sigma^*$.

Then there must exist a resolution refutation for $L$ from input $\Sigma$. Any literal $L'$ (except $L$) in this refutation must be selected at least once for building the resolvent. Therefore, due to our construction of $\rho_{L_1}(\mathcal{I})$ we may cancel all clauses $\mathcal{I} \in \Sigma$ containing the literal $\sim\!L_1$ and simultaneously the literal $L_1$ in all clauses. Thus, there must also exist a resolution refutation for $L$ from input $\Sigma''_1$.

On the other hand, each clause in $\Sigma''_1$ contains at least two literals. Therefore, any resolvent will also contain at least two literals unless we have some $\mathcal{I}_1 \in \Sigma''_1$ with literals $L_1$ and $L_2$ and another $\mathcal{I}_2 \in \Sigma''_1$ with literals $L'_1$ and $\sim\!L'_2$ such that $L_1$, $L'_1$ (and $L_2$, $L'_2$ respectively) are unifiable.

This property, however, means that $\Sigma''_1$ is not stratified contradicting our assumptions. Hence $T$ cannot be repairable and we are done.

Next let $depth(\Sigma) > 1$. We proceed analogously. By induction, since $\Sigma''_i$ is (locally) stratified, there exists a rule system $RTS'_i$ for $\Sigma''_i$ with the required property. The condition parts in these rules have the form $\neg\rho_{L_i}(\mathcal{I})$ for $\mathcal{I} \in \Sigma'_i$. Then let $RTS_i$ result from $RTS'_i$ by changing all condition parts from $\neg\rho_{L_i}(\mathcal{I})$ to $\neg\mathcal{I}$. Finally, take $RTS = \bigcup_{i=1}^{n} RTS_i$.

Again due to the last property in the definition of locally stratified constraint sets (cf. Definition 11) $RTS$ must be complete.

Now consider a critical trigger path $v_0, e_1, v'_1, e'_1, v_1, \dots, e'_\ell, v_\ell$ in the rule hypergraph associated with $RTS$. According to Proposition 4 we have to show that this trigger path is not admissible. Without loss of generality assume $v'_1 \in RTS_1$. Then take a maximal $k$ such that $v'_1, \dots, v'_k \in RTS_1$ holds. Then for $i = 0, \dots, k$ we may write $\varphi_i$ as a conjunction $\psi_i \wedge \mathcal{J}$ with $\models \psi_i \Rightarrow \neg\rho_{L_1}(\mathcal{I}_i)$ for some $\mathcal{I}_i \in \Sigma'_1$. Hence, if we replace $v'_i$ by the corresponding rule in $RTS'_1$, we obtain a critical trigger path for $RTS'_1$.

Now take a state $\sigma$ with $\models_\sigma \Sigma$ and a transition $T$ with $\models \mathbf{Eff}_\sigma(T) \Leftrightarrow \varphi_0$. We have to show that $T$ is not repairable. Assume the contrary. Then there exists a state $\tau$ with $\models_\tau \Sigma$ and $\neg\mathbf{Eff}_\tau \notin \Sigma^*$.

Assume $\models_\sigma \neg L_1$. Since $\models_\sigma \Sigma$ holds and each constraint $\mathcal{I} \in \Sigma_1'$ can be written as a disjunction $\mathcal{I}' \vee \rho_{L_1}(\mathcal{I})$ with $\models \mathcal{I}' \Rightarrow L_1$, we conclude $\models_\sigma \Sigma_1''$.

Since $v_0, e_1, v_1', e_1', v_1, \dots , e_k', v_k$ is a critical trigger path for $RTS_1'$ and $\models \mathbf{Eff}_\sigma \Leftrightarrow \varphi_0$ holds, we may apply the induction hypothesis to $\Sigma_1''$ with $depth(\Sigma_1'') < depth(\Sigma)$. Therefore, $T$ cannot be repairable, i.e. for any state $\zeta$ with $\models_\zeta \Sigma_1''$ we get $\neg\mathbf{Eff}_\zeta(T) \in (\Sigma_1'')^*$.

In particular, take $\zeta = \sigma$. Then $\neg\mathbf{Eff}_\sigma(T) \in (\Sigma_1'')^*$ implies $\models_\sigma \neg\rho_{L_1}(\mathcal{I})$ for some $\mathcal{I} \in \Sigma_1'$ and further $\not\models_\sigma \Sigma^*$ contradicting our assumption on $\sigma$. Thus, we must have $\models_\sigma L_1$.

Assume $\models_\tau \neg L_1$. Then we must have $\models_\tau \Sigma_1''$ and consequently $\neg\mathbf{Eff}_\tau(T) \in (\Sigma_1'')^*$. As above this implies $\models_\tau \neg\rho_{L_1}(\mathcal{I})$ for some $\mathcal{I} \in \Sigma_1'$ and hence $\not\models_\tau \Sigma^*$ contradicting our assumption on $\tau$. Hence, we must have $\models_\tau L_1$.

Now let $\mathcal{I}_1 \in \Sigma$ correspond to the rule $v_1'$. Without loss of generality we may assume $\models \varphi_0 \Rightarrow \neg L_1$. Otherwise, we must have $\models \neg\mathcal{I}_1'$ and $\rho_{L_1}(\mathcal{I}_1)$ must not contain $L_1$. This implies $L_1$ to occur in $\mathcal{J}$, in which case we may change it to $\neg L_1$ without affecting the trigger path being critical.

Since $\models_\sigma L_1$ holds, $T$ must involve an insertion (deletion) corresponding to a negative (positive) literal $L_1$. Hence, $\models \mathbf{Eff}_\tau(T) \Leftrightarrow \neg L_1 \wedge \neg\psi_\tau$ holds. Due to the independence of $\mathcal{J}$ from $\Sigma_1''$ we may choose $\tau$ in such a way that $\psi_\tau \in (\Sigma_1'')^*$ holds.

However, this implies $\models \neg\mathbf{Eff}_\tau(T) \Leftrightarrow L_1 \vee \psi_\tau \in \Sigma^*$ contradicting the non-repairability of $T$ with respect to $RTS_1'$. This completes the sufficiency proof.

Conversely, assume that we are given a complete RTS for $\Sigma$ which for any repairable transition $T$ does not invalidate its effect. According to Proposition 4 this implies that all critical trigger paths in the associated rule hypergraph are not admissible. From this we have to construct a partition of $\Sigma$ into stratified labelled subsystems.

First consider a single rule $R$ corresponding to a constraint $\mathcal{I} \in \Sigma$. In particular, $\mathcal{I}$ is the condition part of this rule. Since $RTS$ is complete, the event part of $R$ gives rise to a negative (positive) literal $L_{ev}$ in $\mathcal{I}$ for the case of an insertion (deletion). Similarly, an insertion (deletion) in the action part of $R$ gives rise to a positive (negative) literal $L_a$ in $\mathcal{I}$.

Let $\rho(\mathcal{I}) = L_{ev} \vee L_a$. If $\mathcal{I}$ contains $n \geq 1$ more literals $L_1, \dots , L_n$, let $\rho_i(\mathcal{I}) = \rho(\mathcal{I}) \vee L_1 \vee \dots \vee \underbrace{L_i}_{\text{omit}} \vee \dots \vee L_n$. Then define $\Sigma_i'(R) = \{\mathcal{J} \in \Sigma \mid \rho_{L_i}(\mathcal{J}) \text{ is defined}\}$ and $\Sigma_i''(R) = \{\rho_{L_i}(\mathcal{J}) \mid \mathcal{J} \in \Sigma_i'(R)\}$. (For $\mathcal{I} \Leftrightarrow \rho(\mathcal{I})$ let $L_1 = L_{ev}$ and $L_2 = L_a$ and define $\Sigma_i'(R)$ and $\Sigma_i''(R)$ analogously.)

Define $\Sigma(R) = \{(\Sigma_i'(R), \Sigma_i''(R), L_i) \mid \Sigma_i''(R) \text{ is locally stratified}\}$, if this satisfies the last condition of Definition 11. Otherwise let $\Sigma(R) = \emptyset$. Then the elements of $\Sigma(R)$ define stratified labelled subsystems of $\Sigma$.

In order to check the local stratification for $\Sigma_i''(R)$ first check, whether it is stratified. If not, define for each literal $L$ in $\rho_i(\mathcal{I})$ the sets $\Sigma_{i,L}'(R) = \{\mathcal{J} \in \Sigma_i''(R) \mid \rho_L(\mathcal{J}) \text{ is defined}\}$ and $\Sigma_{i,L}''(R) = \{\rho_L(\mathcal{J}) \mid \mathcal{J} \in \Sigma_{i,L}'(R)\}$. Consider

$\{(\Sigma'_{i,L}(R), \Sigma''_{i,L}(R), L) \mid \Sigma''_{i,L}(R)$ is locally stratified$\}$ and check the last condition of Definition 11.

Now take $LSS = \bigcup_{R \in RTS} \Sigma(R)$. If $\Sigma(R) \neq \emptyset$ holds for all $R \in RTS$, this satisfies the last condition of Definition 11 and we obtain a partition of $\Sigma$ into stratified labelled subsystems. Then $LSS$ is the required partition.

It remains to show $\Sigma(R) \neq \emptyset$ in the construction above. Assume $\Sigma(R) = \emptyset$. Then there exists a sequence $L_1, L_2, \dots, L_k$ of literals in $\mathcal{I}$ and a sequence $(\Sigma'_1, \Sigma''_1, L_1), \dots, (\Sigma'_k, \Sigma''_k, L_k)$ of non-stratified labelled subsystems such that $\Sigma'_{i+1} = \{\mathcal{J} \in \Sigma''_i \mid \rho_{L_{i+1}}(\mathcal{J})$ is defined$\}$ and $\Sigma''_k$ contains two clauses $\mathcal{I}^k_1$ and $\mathcal{I}^k_2$ with literals $L^1$, $L^{1'}$ and $L^2$, $L^{2'}$ respectively such that $L^1$, $L^2$ and $L^{1'}$, $L^{2'}$ are unifiable.

$\mathcal{I}^k_1$ and $\mathcal{I}^k_2$ correspond to rules with respect to $\Sigma''_k$ that define an admissible trigger path in the associated rule hypergraph. Since for $i = 1, 2$ $\mathcal{I}^k_1$ is $\neg\rho_{L_k}(\mathcal{I}^{k-1}_1)$, we may successively replace these rules by rules corresponding to $\Sigma''_{k-1}, \dots, \Sigma''_1, \Sigma$ and simultaneously replace the formulae $\varphi^k_i$ by $\varphi^{k-1}_i = \varphi^k_i \wedge \neg L_k, \dots, \varphi^0_i = \varphi^1_i \wedge \neg L_1$. The resulting trigger path is still critical and due to our construction it is also admissible with respect to $\Sigma$ contradicting our assumption. This completes the necessity proof. □

EXAMPLE 7 Let us extend Example 3 and add a third constraint

$$\mathcal{I}_3 \quad \equiv \quad p(x,z) \wedge q(y,z) \Rightarrow \mathit{false} \quad .$$

In terms of the Entity-Relationship diagram in Figure 3 $\mathcal{I}_3$ corresponds to an exclusion constraint B||D. It is easy to see that the new set $\{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3\}$ of constraints is not stratified.

In particular, any local stratification must contain a labelled subsystem with label $\neg q(x,z)$ with the reduced constraints $\mathcal{I}'_2 \equiv \neg q(y,z) \vee x = y$ and $\mathcal{I}'_3 \equiv \mathcal{I}_3$. However, $\neg q(x,z)$ cannot occur in the label-free part of some $\mathcal{I}'_2$, since this always defines the same labelled subsystem. Hence, the given constraint set is also not locally stratified. This shows that adding a single exclusion constraint to an Entity-relationship schema may already destroy a reasonable rule behaviour. □

## 7 Complexity of Local Stratification

Let us now look at the check, whether a given set of constraints $\Sigma$ is locally stratified. In the second part of the proof of Theorem 12 we have seen that this check can be done by direct construction of the desired partition into maximal stratified labelled subsystems. The first part of that proof then indicates how to construct the corresponding RTS. In [8] we gave an explicit algorithm which also produces for each constraint the set of "reduced" constraints used in the RTS construction. However, the time complexity of that algorithm was beyond any practicality, since we could proof the following result.

**Proposition 13** Let $\Sigma$ be a set of constraints in clausal form, $n = \#\Sigma$, $\ell$ the maximum number of literals in constraints $\mathcal{I} \in \Sigma$ and $k$ the maximal arity of predicate symbols occurring in these constraints. Then checking $\Sigma$ to be locally stratified can be done with a time complexity in $O(k \cdot \ell^2 \cdot n^{2n^2 \cdot \ell})$. □

We now want to show that this complexity result is not accidentally. For this we first show a technical lemma.

**Lemma 14** Let $\Sigma$ be a set of clauses containing only propositional atoms. Let $L$ be a literal, such that $\sim L$ does not occur in any of the clauses in $\Sigma$. Assume $\Sigma = \Sigma_1 \cup \Sigma_2$ such that $L$ does not occur in any of the clauses in $\Sigma_1$, but in all clauses of $\Sigma_2$. Moreover, $\Sigma_2$ contains only clauses with exactly two literals. If $\Sigma_1$ is locally stratified and $\Sigma_2$ is stratified, then $\Sigma$ is locally stratified.

*Proof.* First assume that $\Sigma_2$ contains a single clause $C = L \vee L'$. If $\Sigma_1$ is not stratified, there is a partition $\Sigma_1 = \Sigma'_{11} \cup \dots \cup \Sigma'_{1n}$ $(n > 2)$ with stratified labelled subsystems $(\Sigma'_{1i}, \Sigma''_{1i}, L_i)$. Then at most one $L_k$ can be $\sim L'$ and we may define

$$\Sigma'_i = \begin{cases} \Sigma'_{1i} & \text{if } L_i = \sim L' \\ \Sigma'_{1i} \cup \{C\} & \text{otherwise} \end{cases} .$$

By induction $(\Sigma'_i, \Sigma''_i, L_i)$ is a stratified labelled subsystem. Thus, $\Sigma = \Sigma'_1 \cup \dots \cup \Sigma'_n$ defines the required partition.

Now assume that $\Sigma_1$ is stratified. Let $\Sigma_1 = \Sigma_{11} \cup \dots \cup \Sigma_{1n}$ be a partition into pairwise disjoint strata. If $\Sigma_1$ contains just one clause $C'$ with $\sim L'$ and no clause with $L'$, we are done, since $C$ may be added to the stratum of $C'$. Analogously, $C$ may define its own stratum, if such a $C'$ does not exist at all. Therefore, we are reduced to the following two cases:

- There is more than one clause in $\Sigma_1$ containing $\sim L'$ (and hence none containing $L'$) and these clauses belong to different strata.
- There are exactly two clauses $C_1$ and $C_2$ containing $\sim L'$ or $L'$ respectively. Inparticular, $C_1$ and $C_2$ belong to the same stratum $\Sigma_{1i}$.

In both cases we choose the literals $L_1 = \sim L$ and $L_2 = L'$ to define labelled subsystems

$$(\Sigma_1, \Sigma_1, L_1) \quad \text{and} \quad (\underbrace{\{C\} \cup \Sigma_1 - \{C'' \mid C'' \text{ contains } \sim L'\}}_{\Sigma'_2}, \Sigma''_2, L_2)$$

where $\Sigma'_2$ (and hence also $\Sigma''_2$) are stratified by the previous remarks.

In the first case choose $C'$ containing $\sim L'$ and another literal $L''$ to define a labelled subsystem

$$(\underbrace{\Sigma'_1 \cup \{C\}}_{\Sigma'_3}, \Sigma''_3, L_3)$$

with $L_3 = \sim L''$, where $\Sigma_1'$ is a proper subset of $\Sigma_1$ not containing $C'$. By induction $\Sigma_3''$ must be locally stratified.

In the second case choose $C_2 = L' \vee C_2'$, a literal $L''$ in $C_2'$ and $L_3 = \sim L''$, which defines a labelled subsystem $(\Sigma_3', \Sigma_3'', L_3)$ as before with $\Sigma_3' = \Sigma_1' \cup \{C\}$ with a proper subset $\Sigma_1' \subsetneq \Sigma_1$ containing $C_1$, but not $C_2$. Thus, $\Sigma_3'$ and $\Sigma_3''$ are stratified.

In both cases we have obtained a partition $\Sigma = \Sigma_1 \cup \Sigma_2' \cup \Sigma_3'$ with stratified labelled subsystems $(\Sigma_1, \Sigma_1, L_1)$, $(\Sigma_2', \Sigma_2'', L_2)$ and $(\Sigma_3', \Sigma_3'', L_3)$. Since the additional condition for local stratification is easily verified, we conclude that $\Sigma$ is locally stratified.

For the general case we may assume that $\Sigma_0 = \Sigma_1 \cup (\Sigma_2 - \{C\})$ is locally stratified by successive application of the constructions in the first part of this proof. Then we observe that in the case of non-stratified $\Sigma_0$ we do not change labels, when we add $C$. However, it may happen that one of these labels now is $\sim L$. This label results (as label $L_1$) from adding $C'$ to some stratified constraint set. From the construction of this local stratification and the fact that $\Sigma_2$ is stratified we conclude that the other labels $L_2$ and $L_3$ are different from $\sim L$, which guarantees the local stratification condition to hold also in the general case.

For the case of $\Sigma_0$ being stratified the arguments are the same as before except for the case that $\Sigma_0$ contains exactly one clause $C'$ with $\sim L'$ and none with $L'$. Then the corresponding stratum may also contain clauses $C_i$ with literals $\sim L_i$ and $L_{i+1}$ $(i = 1, \dots, m)$, where $L_1$ occurs in $C'$ and $L_{m+1} = L$.

In particular, we have $C_m \in \Sigma_2$ and adding $C$ to this stratum is no longer possible. Since $\Sigma_2$ is stratified, we must have $m > 0$, but then the literals $L'$, $L_1$ and $\sim L$ define a local stratification with associated constraint sets $\Sigma_0 - \{C'\} \cup \{C\}$, $\Sigma_0 - \{C_m\} \cup \{C\}$ and $\Sigma_0$ respectively. □

We shall use Lemma 14 in the proof of NP-hardness to shrink propositional constraint sets. Another way to reduce the technical complexity of that proof is to drop the restriction on $\Sigma$ to contain only clauses with at least one negative literal. If $\Sigma$ is a set of propositional clauses containing neither the atom $q$ nor its negation, we add $\neg q$ to each clause to form the set $\Sigma^{\text{ext}}$ of clauses.

**Lemma 15** Let $\Sigma$ be a set of propositional clauses each with at least two literals. Then $\Sigma^{\text{ext}}$ is locally stratified iff $\Sigma$ is satisfiable and locally stratified.

*Proof.* First let $\Sigma$ be locally stratified and satisfiable. If $\Sigma$ is not stratified, we may choose the same labels to obtain a local stratification for $\Sigma^{\text{ext}}$.

Thus, assume $\Sigma$ to be stratified. Then $(\Sigma^{\text{ext}}, \Sigma, \neg q)$ is a stratified labelled subsystem. Since all clauses in all other labelled subsystems contain the literal $\neg q$, we have to isolate these clauses. Therefore, take a model for $\Sigma$ which is given by a set $\{L_1, \dots, L_n\}$ of literals occurring in $\Sigma$ which must be interpreted as *true*. Taking $\sim L_i$ as a label and the corresponding labelled subsystem $(\Sigma_i', \Sigma_i'', \sim L_i)$, we obtain a proper subset $\Sigma_i' \subsetneq \Sigma^{\text{ext}}$. For $\#\Sigma_i'' > 1$ we may proceed with the other literals $\sim L_j$. The last step results in unary sets $\{\neg q \vee L_k\}$ which are obviously stratified.

Conversely, given a local stratification for $\Sigma^{\text{ext}}$ we can remove $\neg q$ to obtain a local stratification for $\Sigma$. It remains to show that $\Sigma$ is satisfiable. If $\Sigma^{\text{ext}}$ is stratified, this is obvious, because a literal $L$ with $\sim L$ occurring in some clause in $\Sigma$ cannot occur in any clause of $\Sigma$.

If $\Sigma^{\text{ext}}$ is not stratified, there is at least one stratified labelled subsystem $(\Sigma', \Sigma'', L)$ such that $\neg q$ occurs in all clauses in $\Sigma''$, i.e. $\Sigma'' = \Sigma_0^{\text{ext}}$ and $\Sigma_0$ is satisfiable. This still holds if we put back the literal $L$ and extend our interprete $L$ as *false* to satisfy clauses in $\Sigma - \Sigma_0$. □

**Theorem 16** Let $\Sigma$ be a set of constraints. Then checking that $\Sigma$ is locally stratified is NP-hard.

*Proof.* We show that the *disjoint cover problem* (DCP) – which is known to be NP-complete – can be reduced in polynomial time to the local stratification problem. For this, let $(X, \mathcal{S})$ be an instance of DCP, i.e. $X$ is a finite set, say, $X = \{x_1, \dots, x_n\}$ and $\mathcal{S} = \{S_1, \dots, S_m\}$ is a subset of the power set $\mathfrak{P}(\mathfrak{X})$. The problem is to decide, whether a subset $\mathcal{S}' \subseteq \mathcal{S}$ exists such that $X$ is the disjoint union of the sets in $\mathcal{S}'$. Such a $\mathcal{S}'$ is called a *solution* for $(X, \mathcal{S})$.

Without loss of generality we may always assume that $X = \bigcup_{S_i \in \mathcal{S}} S_i$ holds. Moreover, we may allow $\mathcal{S}$ to be a multiset.

We now associate with $(X, \mathcal{S})$ a set of constraints $\Sigma$. For this let $p_{ij}$ be a propositional atom for all $x_i \in S_j$. For $S_i = \{x_{j_1}, \dots, x_{j_i}\} \in \mathcal{S}$ we define clauses $\neg p_{j_k i} \vee p_{j_\ell i}$ and $\neg p_{j_\ell i} \vee p_{j_k i}$ for $k, \ell \in \{1, \dots, i\}$, $k \neq \ell$. We refer to these clauses as *connection clauses* with respect to $S_i$. For $x_i \in S_j \cap S_k$ $(j \neq k)$ we define an *exclusion clause* $\neg p_{ij} \vee \neg p_{ik}$. Finally, for each $x_i$ we define a *cover clause* $p_{ij_1} \vee \dots \vee p_{ij_m}$ for the sets $S_{j_1}, \dots, S_{j_m} \in \mathcal{S}$ containing $x_i$ provided $m \geq 2$. $\Sigma$ contains all these connection, exclusion and cover clauses.

Then we have to show that $(X, \mathcal{S})$ has a solution iff $\Sigma$ is locally stratified and satisfiable. For this we introduce a partial order $\leq$ on DCP-instances letting $(X_1, \mathcal{S}_1) < (X_2, \mathcal{S}_2)$ iff

$$\sum_{S \in \mathcal{S}_1} | S | < \sum_{S \in \mathcal{S}_2} \quad \text{or} \quad \left( \sum_{S \in \mathcal{S}_1} = \sum_{S \in \mathcal{S}_2} \quad \text{and} \quad | \mathcal{S}_1 | > | \mathcal{S}_2 | \right)$$

holds.

First let $\mathcal{S}' = \{S_{i_1}, \dots, S_{i_k}\}$ be a solution for $(X, \mathcal{S})$. Then $\Sigma$ is obviously satisfiable. In order to use induction with respect to $\leq$ we consider the following two operations:

- Replace $S_j \in \mathcal{S}'$ by $S_j - \{x_\ell\}$ and add $S_{m+1} = \{x_\ell\}$ for some $x_\ell \in S_j$.
- Replace $S_j \notin \mathcal{S}'$ by $S_j - \{x_\ell\}$ for some $x_\ell \in S_j$.

In both cases we obtain a smaller DCP-instance which has a solution. By induction the corresponding constraint set $\Sigma_1'$ is locally stratified.

In the first case we remove all clauses with literals $p_{i,m+1}$ from $\Sigma_1'$. The resulting subset $\Sigma_1''$ is still locally stratified. Now build the labelled subsystem $(\Sigma', \Sigma'', L)$ with the label $L = \neg p_{\ell j}$.

The clauses in $\Sigma'$ (and hence in $\Sigma''$) do not contain $p_{\ell j}$, i.e. we omit the cover clause with respect to $x_\ell$ and connection clauses containing $p_{\ell j}$ with respect to $x_\ell \in S_j$. Clauses in $\Sigma''$ containing $\neg p_{\ell j}$ arise from the restriction to keep at least two literals, hence must also lie in $\Sigma'$. Therefore, we obtain $\Sigma'' = \Sigma_1 \cup \Sigma_2$, where $\Sigma_2$ is stratified and contains only clauses with two literals, one of them is $\neg p_{\ell j}$, whereas clauses in $\Sigma_1$ do not contain $\neg p_{\ell j}$.

Thus, the remaining connection clauses with respect to $x_\ell \in S_j$ and the exclusion clauses with respect to $x_\ell \in S_j$ occur in $\Sigma_2$. This implies $\Sigma_1 = \Sigma_1''$. From Lemma 14 we conclude that $\Sigma''$ is locally stratified.

In the second case we build the labelled subsystem $(\Sigma', \Sigma'', L)$ with the label $L = p_{\ell j}$. The clauses in $\Sigma'$ (and hence in $\Sigma''$) do not contain $\neg p_{\ell j}$, i.e. we omit exclusion clauses and connection clauses containing $\neg p_{\ell j}$ with respect to $x_\ell \in S_j$. Again, the clauses in $\Sigma''$ containing $p_{\ell j}$ only arise from the restriction to keep at least two literals. Hence, these clauses define a stratified subset $\Sigma_2$ of $\Sigma''$ (and of $\Sigma'$) containing only clauses with two literals.

The remaining clauses form a subset $\Sigma_1$ and clauses in $\Sigma_1$ do not contain $p_{\ell j}$, i.e. the remaining connection clauses with respect to $x_\ell \in S_j$ and the cover clause with respect to $x_\ell$ (if it contains just two literals) occur in $\Sigma_2$, which implies $\Sigma_1 = \Sigma_1'$. From Lemma 14 we conclude that $\Sigma''$ is locally stratified.

Since in the first case ($x_\ell \in S_j \in \mathcal{S}'$) only the cover clause with respect to $x_\ell$ and connection clauses containing $p_{\ell j}$ and in the second case ($x_\ell \in S_j \notin \mathcal{S}'$) only exclusion clauses with respect to $x_\ell \in S_j$ and connection clauses containing $\neg p_{\ell j}$ are omitted in $\Sigma'$, the additional condition for local stratification is easily verified, if all such choices are taken provided there are at least three such possibilities. The only critical case arises, if there are only three choices of the second kind, all with the same $x_\ell$. In this case we must have another $S_j = \{x_\ell\} \in \mathcal{S}'$ and we simply add the labelled subsystem $(\Sigma', \Sigma'', \neg p_{\ell j})$ to satisfy the additional local stratification condition.

If there are at most two choices, then either

- $\mathcal{S} = \mathcal{S}'$ and there is exactly one $S_j = \{x_k, x_\ell\}$ or
- $\mathcal{S}'$ contains only unary sets and these are exactly $S_j = \{x_j\} \notin \mathcal{S}'$ and $S_k = \{x_k\} \notin \mathcal{S}'$ or
- $\mathcal{S}'$ contains only unary sets and there is exactly one $S_j = \{x_k, x_\ell\} \notin \mathcal{S}'$.

In the first case $\Sigma$ contains only two connection clauses with respect to $S_j$ and hence is obviously stratified. In the second case $\Sigma$ contains only four clauses

$$\neg p_{kk} \vee \neg p_{kk'}, \ p_{kk} \vee p_{kk'}, \ \neg p_{jj} \vee \neg p_{jj'} \quad \text{and} \quad p_{jj} \vee p_{jj'}$$

for $S_{j'} = \{x_j\} \in \mathcal{S}'$ and $S_{k'} = \{x_k\} \in \mathcal{S}'$, hence $\Sigma$ is stratified.

In the third case we obtain six clauses

$$\neg p_{kj} \vee p_{\ell j}, \ \neg p_{\ell j} \vee p_{kj}, \ \neg p_{kj} \vee p_{kk'}, \ \neg p_{\ell j} \vee p_{\ell\ell'}, \ p_{kj} \vee p_{kk'} \quad \text{and} \quad p_{\ell j} \vee p_{\ell\ell'}$$

for $S_{k'} = \{x_k\} \in S'$ and $S_{\ell'} = \{x_\ell\} \in S'$. Using Lemma 14 it is easily verified that the labels $p_{kj}$, $p_{\ell j}$, $\neg p_{kj}$ and $\neg p_{\ell j}$ define a partition into stratified labelled subsystems.

For the converse let us first assume that $\Sigma$ is stratified, i.e. there cannot exist three clauses with literals $L$, $L$ and $\sim L$ respectively. In connection clauses we may have $L = p_{\ell j}$ (or $L = \neg p_{\ell j}$) and it follows that $\Sigma$ does not contain exclusion or cover clauses for $x_\ell \in S_j$. This implies $x_\ell \notin S_k$ for all $k \neq j$. If we have an exclusion clause for $x_\ell \in S_j$, say $\neg p_{\ell j} \vee \neg p_{\ell k}$, then we also have a cover clause $p_{\ell j} \vee p_{\ell k} \vee C'$ and vice versa, but there cannot be further exclusion clauses nor connection clauses for $x_\ell \in S_j$, i.e. $C' \equiv false$ and $S_j = \{x_\ell\}$.

To summarize, if $x_\ell$ occurs in more than one $S_j$, then $\# S_j = 1$ and there are just two such sets. Therefore, for a solution $S'$ we take all $S_j$ with $\# S_j \geq 2$ and select a singleton set $\{x_\ell\}$ for the remaining elements.

Next assume that $\Sigma$ is locally stratified, i.e. there is a local stratification with labels $L_1, \dots, L_n$ $(n \geq 3)$. Again, we proceed by induction on DCP-instances.

For $L_1 = \neg p_{\ell j}$ and the stratified labelled subsystem $(\Sigma_1', \Sigma_1'', L_1)$ the cover clause for $x_\ell$ and connection clauses for $x_\ell \in S_j$ containing $p_{\ell j}$ have been removed from $\Sigma$ to give $\Sigma_1'$, hence must occur in two other labelled subsystems such that for a label $\neg p_{ki}$ we must have $k \neg \ell$ and for a label $p_{ki}$ we must have $i \neq j$.

Analogously, for $L_1 = p_{\ell j}$ exclusion and connection clauses for $x_\ell \in S_j$, the latter ones containing $\neg p_{\ell j}$ have been removed omitted in $\Sigma_1'$ and must occur in two other labelled subsystems such that for another positive label $p_{ki}$ we must have $k \neg \ell$ and for a negative label $\neg p_{ki}$ we must have $i \neq j$. Hence, for the minimum number of three labels $L_1$, $L_2$ and $L_3$ we obtain the following four cases:

$$\begin{array}{rl}
L_1 = \neg p_{\ell j},\ L_2 = \neg p_{k_1 i_1},\ L_3 = \neg p_{k_2 i_2} & \text{with pairwise different } \ell, k_1, k_2 \quad , \\
L_1 = \neg p_{\ell j},\ L_2 = \neg p_{k_1 i_1},\ L_3 = p_{k_2 i_2} & \text{with } \ell \neq k_1 \text{ and } j \neq i_2 \neq i_1 \quad , \\
L_1 = \neg p_{\ell j},\ L_2 = p_{k_1 i_1},\ L_3 = p_{k_2 i_2} & \text{with } k_1 \neq k_2 \text{ and } i_1 \neq j \neq i_2 \quad \text{or} \\
L_1 = p_{\ell j},\ L_2 = p_{k_1 i_1},\ L_3 = p_{k_2 i_2} & \text{with pairwise different } \ell, k_1, k_2 \quad .
\end{array}$$

For a negative literal $L_i = \neg p_{\ell j}$ or a positive literal $L_i = p_{\ell j}$ it follows from Lemma 14 that replacing $S_j$ by $S_j - \{x_\ell\}$ and $\{x_\ell\}$ defines a locally stratified constraint set. Therefore, by induction in all four cases (with the restrictions for indices) we obtain solutions for smaller DCP-instances with

$$\begin{aligned}
\mathcal{S}_1 &= \{S_1, \dots, S_j - \{x_\ell\}, \dots, S_m, \{x_\ell\}\} \quad , \\
\mathcal{S}_2 &= \{S_1, \dots, S_{i_1} - \{x_{k_1}\}, \dots, S_m, \{x_{k_1}\}\} \quad \text{and} \\
\mathcal{S}_3 &= \{S_1, \dots, S_{i_2} - \{x_{k_2}\}, \dots, S_m, \{x_{k_2}\}\}
\end{aligned}$$

respectively. If any of these solutions contains both (or none) of the splitted components, e.g. $S_j - \{x_\ell\}$ and $\{x_\ell\}$, we also have a solution fot the original problem.

Therefore, assume that all solutions for $(X, \mathcal{S}_i)$ must contain exactly one of the splitted components denoted as $\overline{S_1}$, $\overline{S_2}$ and $\overline{S_3}$. Let $\mathcal{S}_i' = \{S_1^i, \dots, S_{n_i}^i, \overline{S_i}\}$ be a solution for $(X, \mathcal{S}_i)$. For $i \neq j$ we proceed in the following way:

Start with $\mathcal{T}_i = \mathcal{S}'_i - \mathcal{S}'_j$, $\mathcal{T}_j = \mathcal{S}'_j - \mathcal{S}'_i$ and $\mathcal{T} = \{\overline{S_j}\}$ and execute the following steps until there are no more changes:

- Remove all sets from $\mathcal{T}_i$ intersecting some set in $\mathcal{T}$ and let these define a new $\mathcal{T}$.
- Remove all sets from $\mathcal{T}_j$ intersecting some set in $\mathcal{T}$ and let these define a new $\mathcal{T}$.

Finally, if $\mathcal{T}_i$ (and then also $\mathcal{T}_j$) are non-empty, this means that we may replace $\mathcal{T}_j \subseteq \mathcal{S}'_j$ by $\mathcal{T}_i$ or $\mathcal{S}'_j - \mathcal{T}_j$ by $\mathcal{S}'_i - \mathcal{T}_i$. According to our assumption on solutions we always keep either $\overline{S_i}$ or $\overline{S_j}$. Consequently, the procedure above defines a chain

$$\overline{S_i} - S^j_{i_1} - S^i_{i_1} - S^j_{i_2} - S^i_{i_2} - \cdots - S^j_{i_k} - S^i_{i_k} - \overline{S_j} \quad ,$$

where neighbouring sets have a common element. This is still true, if we replace $\overline{S_i}$ by the original $S_j$. Taking together all three choices for $(i, j)$ we obtain an odd-length cycle

$$S_{i_1} - S_{i_2} - S_{i_3} - \cdots - S_{i_m} - S_{i_1}$$

with intersecting neighbouring sets $S_{i_j} \in \mathcal{S}$. Let $\Sigma'$ be the set of constraints corresponding to $\{S_{i_1}, \ldots, S_{i_m}\}$. Then $\Sigma'$ differs from a subset $\Sigma_0 \subseteq \Sigma$ only by the fact that cover clauses may have been shortened. Since omitted (positive) literals in these cover clauses do not occur in any other clauses in $\Sigma'$, this must be locally stratified iff $\Sigma_0$ is locally stratified. Therefore, the proof is completed, if we can show that cycles as above always define constraint sets that are not locally stratified or not satisfiable.

With each neighbouring pair $(S_{i_j}, S_{i_{j+1}})$ we may associate a witness $x \in S_{i_j} \cap S_{i_{j+1}}$. Then without loss of generality (just rename indices) we can always assume a cycle

$$S_1 \overset{x_1}{-} S_2 \overset{x_2}{-} S_3 - \cdots - S_m \overset{x_m}{-} S_{m+1} = S_1$$

and show that the following conditions can be achieved:

- $m$ is odd,
- the $x_i$ are pairwise different,
- the $S_i$ are pairwise different and
- the cover clause in $\Sigma'$ for $x_\ell$ has the form $p_{\ell\ell} \vee p_{\ell\ell+1} \vee C'_\ell$, where literals in $C'_\ell$ do not occur in any other clause in $\Sigma'$.

The last condition will allow us to assume without loss of generality that cover clauses in $\Sigma'$ only contain two literals.

In order to achieve such a cycle recall that our original cycle is composed of three subpaths (called *flanks*) corresponding to a solution of a smaller DCP-instance and each pair of flanks has a common set (called *corner*). If $\overline{S_i} \subsetneq S_j$ is such a corner, then the following cases may arise:

- The two nieghbours $S_i$ and $S_k$ coincide which allows to remove the corner $S_j$ and to identify $S_i$ with $S_k$.
- If $S_i$, $S_j$ and $S_k$ are pairwise different, we either obtain a simple cycle of length 3 or let the cycle unchanged.
- If one of the neighbours equals $S_j$, say $S_k = S_j$, then $S_k$ is not common in the solutions for the flank with $S_j$ and $S_k$, i.e. there must be some $S_{j'}$ in the same solution as $\overline{S_i}$ with $S_j \cap S_{j'} \neq \emptyset$. In this case we may replace the even number of edges between $S_j$ and $S_{j'}$ by a single edge. By the same argument the even number of edges between the opposite edge $S_\ell$ (in the same flank) and some $S_{\ell'}$ by a single edge.

In all these cases the cycle length remains odd.

If $x_i$ occurs twice, say between $Si_1$ and $S_{i_2}$ and between $Si_3$ and $S_{i_4}$ respectively, we may assume paths from $S_{i_1}$ to $S_{i_4}$ and from $S_{i_2}$ to $S_{i_3}$ of length $n_1$ and $n_2$ respectively. Then there are cycles with $S_{i_2}$, $S_{i_3}$ and $S_{i_1}$, $S_{i_4}$ connected by $x_i$ respectively and one of the corresponding lengths $n_1 + 1$ or $n_2 + 1$ must be odd. The only critical cases occur for $S_{i_2} = S_{i_4}$ or $S_{i_1} = S_{i_3}$, but these correspond to corners that have already been removed.

Finally, in order to achieve the condition on cover clauses consider $S_i \cap S_j \neq \emptyset$.

- If $S_i$ and $S_j$ belong to different flanks, but to the same solution, then we have $S_i = S_j$ and we may identify them and remove the even number of edges between them.
- If $S_i$ and $S_j$ belong to different flanks and different solutions, then for $S_i \neq S_j$ we may replace the odd number of edges between them by a single new edge, whereas for $S_i = S_j$ we may consider the odd number of edges between them as our new cycle.
- If $S_i$ and $S_j$ belong to the same flank, then the number of edges between them is even iff $S_i = S_j$, thus may be removed or replaced by a single new edge.

The conditions on our cycle now allows clauses to be arranged in such a way that we have

$$\Sigma' = \{\sim L_1 \vee L_2, \sim L_2 \vee L_3, \dots, \sim L_{p-1} \vee L_p, \sim L_p \vee L_1\}$$

for an even number $p$ with $L_{p/2+i} = \sim L_i$ for $i = 1, \dots, p/2$. Such a $\Sigma'$, however, is not satisfiable. □

## 8 Conclusion

In this article we investigated the limits of rule triggering systems (RTSs) for maintaining database integrity under the additional requirement to preserve the effects of transitions. The first result assures the existence of non-repairable transitions. In order to disallow such transitions the constraint implication problem must be decidable.

Secondly, we analyzed critical trigger paths in rule hypergraphs associated with RTSs. We could show that the existence of critical trigger paths leads to RTSs which may invalidate the effect of some transitions, even if these are repairable. Such a behaviour can only be excluded for *locally stratified* constraint sets. In this case the needed RTS can be computed effectively, but checking local stratification is NP-hard.

To summarize, both results limit the applicability of RTSs for integrity maintenance under the assumption that the intended effects of user-defined transitions should be preserved. Fortunately, there is a stronger condition on a constraint set to be *stratified*, which is only sufficient for reasonable rule behaviour, but not necessary. Stratified constraint sets occur, if we have a relational database schema in Entity-Relationship normal form, which means that it is equivalent to an ER-schema without exclusion constraints. Checking stratification is not only effective, but also efficient.

On the other hand, the RTS approach to integrity maintenance completely ignores user-defined transitions. Thus, a second conclusion from our studies is that these should be taken into consideration.

## References

[1] S. Abiteboul, V. Vianu: *Equivalence and Optimization of Relational Transactions*, Journal of the ACM, vol. 35(1), 1988, 70-120

[2] S. Ceri, J. Widom: *Deriving Production Rules for Constraint Maintenance*, Proc. 16th Conf. on VLDB, Brisbane (Australia), August 1990, 566-577

[3] S. Ceri, P. Fraternali, S. Paraboschi, L. Tanca: *Automatic Generation of Production Rules for Integrity Maintenance*. ACM ToDS, vol. 19(3), 1994, 367-422.

[4] S. Chakravarty, J. Widom (Eds.): *Research Issues in Data Engineering — Active Databases*, Proc., Houston, Februar 1994

[5] M. Gertz, U. W. Lipeck: *Deriving Integrity Maintaining Triggers from Transition Graphs*, in Proc. 9th ICDE, IEEE Computer Society Press, 1993, 22-29

[6] H. Mannila, K.-J. Räihä: *The Design of Relational Databases*, Addison-Wesley 1992

[7] K.-D. Schewe, B. Thalheim: *Consistency Enforcement in Active Databases*, in S. Chakravarty, J. Widom (Eds.): *Research Issues in Data Engineering — Active Databases*, Proc., Houston, Februar 1994

[8] K.-D. Schewe, B. Thalheim: *Active Consistency Enforcement for Repairable Database Transitions*, in S.Conrad, H.-J. Klein, K.-D. Schewe (Eds.): Integrity in Databases, Proc. 6th Int. Workskop on Foundations of Models and Languages for Data and Objects, Schloß Dagstuhl, 1996, 87-102, available via `http://wwwiti.cs.uni-magdeburg.de/~conrad/IDB96/Proceedings.html`

[9] B. Thalheim: *Foundations of entity-relationship modeling*, Annals of Mathematics and Artificial Intelligence, vol. 7, 1993, 197-256

[10] S. D. Urban, L. Delcambre: *Constraint Analysis: a Design Process for Specifying Operations on Objects*, IEEE Trans. on Knowledge and Data Engineering, vol. 2 (4), December 1990

[11] J. Widom, S. J. Finkelstein: *Set-oriented Production Rules in Relational Database Systems*, in Proc. SIGMOD 1990, 259-270

*Received, April, 1997*

# Non-Markovian Policies in Sequential Decision Problems

Csaba Szepesvári *†

**Abstract**

In this article we prove the validity of the Bellman Optimality Equation and related results for sequential decision problems with a general recursive structure. The characteristic feature of our approach is that also non-Markovian policies are taken into account. The theory is motivated by some experiments with a learning robot.

## 1 Introduction

The theory of sequential decision problems is an important mathematical tool for studying some problems of cybernetics, e.g. control of robots. Consider for example the robot shown in Figure 1. This robot, called Khepera[1], is equipped with eight infra-red sensors, six in the front and two at the back, the infra- red sensors measuring the proximity of objects in the range 0-5 cm. The robot has two wheels driven by two independent DC motors and a gripper that has two degrees of freedom and is equipped with a resistivity sensor and an object-presence sensor. The robot has a vision turret mounted on its top. The vision turret has an image sensor giving a linear image of the horizontal view of the environment with a resolution of 64 pixels and 256 levels of grey. The task of the robot was to find a ball in the arena, bring it to the stick and hit the stick by the ball so as to it jumps out of the gripper. Macro actions such as search, grasp, etc. were defined and the expected number of macro actions taken by the robot until the goal was reached was choosen as the performance measure. Some digital, dynamic filters provide the "state information" necessary for making decisions (for more details concerning these filters see [7]). The robot learnt on-line from the observations $(x_t, a_t, c_t)$, where $x_t \in \mathcal{X}$ is the actual output of the filters ($\mathcal{X}$ is a finite set, called the state space), $a_{t-1} \in \mathcal{A}$ is the previous (macro-)action taken by the robot ($\mathcal{A}$ is also a finite set, called the

*Research Group on Artificial Intelligence, "József Attila" University, Szeged, 6720 Aradi vértanúk tere 1., HUNGARY, e-mail: szepes@math.u-szeged.hu

†Work supported by the Central Research Fund of Hungarian Academy of Sciences (Grant No. T014548)

[1]Khepera is designed and built at Laboratory of Microcomputing, Swiss Federal Institute of Technology, Lausanne, Switzerland and is available commercially.

action set), and $c_t$ is the cost of transition $(x_{t-1}, a_{t-1}, x_t)$ which was 1 until the goal was reached. The task turns out to be well approximated as a Markovian Decision Problem (MDP), i.e. one may assume the existence of transition probabilities of form $p(x,a,y)$, where $p(x,a,y)$ gives the probability of going to state $y$ from state $x$ when action $a$ is used; and the existence of a cost-structure $c(x,a,y)$ s.t. $c_t = c(x_{t-1}, a_{t-1}, x_t)$. The objective is to minimize the total expected discounted cost, $E[\sum_{t=0}^{\infty} \gamma^t c_t]$, $0 < \gamma < 1$, by choosing an appropriate policy, a policy being any function that maps past observations to actions (sometimes to distributions over the action set). Because of the uniform cost structure and the absorbing goal state, the discounted cost criterion can be shown to be equivalent to the undiscounted one, i.e., to minimizing the expected number of steps until the goal is reached. The reason of considering the discounted total cost criterion is that the presence of the discount factor makes the theory of such MDPs quite appealing. In particular, it is well known that policies which, for any given state $x \in \mathcal{X}$, choose the action minimising

$$\sum_{y \in \mathcal{X}} p(x,a,y)(c(x,a,y) + \gamma v^*(y))$$

are optimal. Here $v^*$ is the so-called optimal cost function, defined by

$$v^*(x) = \inf_{\pi \in \Pi} v_\pi(x), \quad x \in \mathcal{X},$$

where $\Pi$ is the set of policies. More importantly, $v^*$ is known to satisfy the Bellman Optimality Equation

$$v^*(x) = \min_{a \in \mathcal{A}} \sum_{y \in \mathcal{X}} p(x,a,y)(c(x,a,y) + \gamma v^*(y)), \quad x \in \mathcal{X},$$

which is a non-linear equation for $v^*$. Fortunately, because of the presence of the discount factor, $\gamma$, $v^*$ can be found (approximately) in a number of ways. For example, introducing the *optimal cost operator*, $T : \mathbb{R}^{\mathcal{X}} \to \mathbb{R}^{\mathcal{X}}$, defined by

$$(Tv)(x) = \min_{a \in \mathcal{A}} \sum_{y \in \mathcal{X}} p(x,a,y)(c(x,a,y) + \gamma v(y)), \quad x \in \mathcal{X},$$

gives that $v^*$ is the fixed point of $T$, which can be shown to be a contraction in the sup-norm (in fact, $||Tv - Tu|| \leq \gamma ||v - u||$ holds) and so the Banach fixed-point theorem yields that $v_{n+1} = Tv_n$ converges to $v^*$ in the sup-norm for any choice of $v_0$. This algorithm, called the *value-iteration algorithm* (or dynamic programming algorithm), served as the basis of the most successful learning algorithm for the above robotic task. The idea of this learning algorithm is to estimate the transition probabilities $p(x,a,y)$ and the costs $c(x,a,y)$ by their respective maximum-likelihood estimates to obtain $p_t(x,a,y)$ and $c_t(x,a,y)$, respectively and then compute $v_t$, the $t^{\text{th}}$ approximation to the optimal cost function $v^*$, as the fixed point of the approximate optimal cost operator $T_t$ which is defined as $T$ but when $p$ and $c$ are replaced by their respective estimates. After a few hours of learning the performance of the

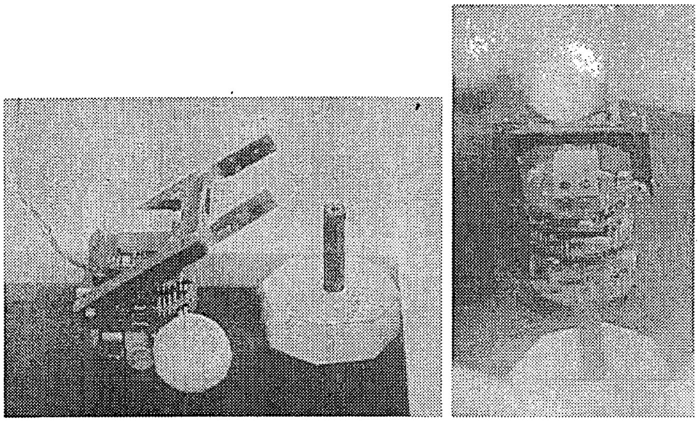

Figure 1: **The Khepera robot**
The figures show a Khepera robot. The description of the sensors and actuators of the robot can be found in the text.

robot using this learning strategy was comparable to that of a well designed control strategy (whose design took a couple of days).

It is important to observe that the *expected* total discounted cost criterion, although suitable in many cases, may yield undesirable behaviour in some cases. For example, in safety-critical applications (like controlling a Mars-rover) the average-case optimal policy may be too bold. Other criteria, such as the minimax criterion which concerns only the long-term worst-case outcomes of the decisions, take safety much better into account. There is a continuum of other criteria which are in between the expected and the minimax criteria. In this article we consider structural questions, such as the validity of the Bellman Optimal Equation, associated with sequential decision problems given by general decision criteria. However, the problem of learning optimal policies will not be considered here. Nevertheless, the theory investigated here is important as the analysis of such learning algorithms should be based on it. For further information on learning issues the reader is referred to the articles [5, 9] and [6]. The main contribution of this article to the "static-theory", which considers structural problems, is that here we do not restrict the analysis to Markovian policies as it is usual in the literature (see, e.g. [1, 10]), but we also consider general policies, which is important since learning policies are non-Markovian by nature.

## 2 Results

**Notation.** The set of natural numbers ($\{0, 1, 2, \ldots\}$), integers and reals will be denoted by $\mathbb{N}, \mathbb{Z}$ and $\mathbb{R}$, respectively. $\mathcal{R}(Z)$ will denote the set of extended real-valued functions over $Z$: $\mathcal{R}(Z) = [-\infty,\infty]^Z$, and $\mathcal{B}(Z)$ will denote the set of bounded real-valued functions over $Z$: $\mathcal{B}(Z) \subset \mathbb{R}^Z$, s.t. if $f \in \mathcal{B}(Z)$ then $\|f\| = \sup_{z\in Z} |f(z)| < \infty$. The relation $u \le v$ will be applied to functions in the usual way: $u \le v$ means that $u(x) \le v(x)$ for all $x$ in the domain of $u$ and $v$. Further, $u < v$ will denote that $u \le v$ and that there exists an element $x$ of the domain of $u$ and $v$ such that $u(x) < v(x)$. We employ the symbol $\le$ for operators in the same way, and say that $S_1 \le S_2$ $(S_1, S_2 : \mathcal{R}(Z) \to \mathcal{R}(Z))$ if $S_1 v \le S_2 v$ for all $v \in \mathcal{R}(Z)$. If $S : \mathcal{R}(Z) \to \mathcal{R}(Z)$ is an arbitrary operator then $S^k$ $(k = 1,2,3,\dots)$ will denote the composition of $S$ with itself $k$ times: $S^0 v = v$, $S^1 v = Sv$, $S^2 v = S(Sv)$, etc. In the following $t, s, n, i, j, k$ will denote natural numbers.

DEFINITION 2.1 *An operator $S : \mathcal{R}(Z_1) \to \mathcal{R}(Z_2)$ is said to be* Lipschitz *with index $0 < \gamma$ if $S$ maps $\mathcal{B}(Z_1)$ into $\mathcal{B}(Z_2)$ and if for all $f, g \in \mathcal{B}(Z_1)$, $\|Sf - Sg\| \le \gamma\|f - g\|$. $S$ is said to be a* contraction *with index $\gamma$ if $S$ is Lipschitz with index $\gamma < 1$.*

DEFINITION 2.2 *An operator $S : \mathcal{R}(Z_1) \to \mathcal{R}(Z_2)$ is said to be (weakly)* continuous *if for all pointwise continuous function sequence $\{f_n\} \subset \mathcal{R}(Z_1)$ with limit function $f$, also $\lim_{n\to\infty}(Sf_n)(z) = (Sf)(z)$, $\forall z \in Z_2$.*

It is well known that $S$ can be Lipschitz without being continuous and vice versa. continuous in the topology induced by pointwise convergence: Let $S : \mathcal{B}(\mathbb{N}) \to \mathcal{B}(\mathbb{N})$ be defined as $(Sf)(i) = \inf_{j\ge 1} f(j)$ if $i = 0$ and $(Sf)(i) = f(i)$, otherwise. Clearly, $S$ is Lipschitz with index 1. Let $f_n(i) = 1$, if $0 \le i \le n$ and $f_n(i) = 0$ otherwise. Now, if we let $f(i) = 1$, $i \in \mathbb{N}$ then $f_n \to f$ pointwise but not in the sup-norm, and $0 = \lim_{n\to\infty}(Sf_n)(0) \ne (Sf)(0) = 1$ showing that $S$ is not continuous in the sense of Definition 2.2.

## Sequential Decision Problems.

DEFINITION 2.3 *An* sequential decision problem *(SDP) is a quadruple $(\mathcal{X}, \mathcal{A}, \mathcal{Q}, \ell)$, where $\mathcal{X}$ is the state space of the process, $\mathcal{A}$ is the set of actions, $\mathcal{Q} : [-\infty,\infty]^{\mathcal{X}} \to [-\infty,\infty]^{\mathcal{X}\times\mathcal{A}}$ is the so-called* cost propagation operator *and $\ell \in B(\mathcal{X})$ is the so-called terminal cost function.*

In most of the results we will assume that $\mathcal{Q}$ is a contraction and is continuous in the sense of Definition 2.2.

The mapping $\mathcal{Q}$ makes it possible to define the cost of an action sequence in a recursive way: the cost of action $a$ in state $x$ is given by $(\mathcal{Q}f)(x,a)$ provided the decision process stops immediately after the choice of the first action and the terminal cost of stopping in state $y$ is given by $f(y)$.

The history of a decision process up to the $t^{\text{th}}$ stage is a sequence of state-action pairs: $(a_t, x_t, a_{t-1}, x_{t-1}, \dots, a_0, x_0)$. Set $H_t = (\mathcal{A} \times \mathcal{X})^t$, $t \ge 0$. For brevity, $h = ((a_t, x_t), \dots, (a_0, x_0))$ will be written as $h = a_t x_t \dots a_0 x_0$. Further, for any pair $h_1 = ((a_t, x_t), \dots, (a_0, x_0))$ and $h_2 = ((a'_s, x'_s), \dots, (a'_0, x'_0))$ we will denote by $h_1 h_2$ the concatenation of $h_1$ and $h_2$: $((a_t, x_t), \dots, (a_0, x_0), (a'_s, x'_s), \dots, (a'_0, x'_0))$. We admit the assumption that the ordering of the components of $h = a_t x_t \dots a_0 x_0$ corresponds to the time order, i.e., $(a_t, x_t)$ is the most recent element of the history.

DEFINITION 2.4 *A* policy *is an infinite sequence of mappings:* $\pi = (\pi_0, \pi_1, \ldots, \pi_t, \ldots)$, *where* $\pi_t : \mathcal{X} \times H_t \to \mathcal{A}$, $t \geq 0$. *If* $\pi_t$ *depends only on* $\mathcal{X}$ *then the policy is called* Markovian, *otherwise, it is called* non-Markovian. *If a policy is Markovian and* $\pi_t = \pi_0$ *for all* $t$ *then the policy is called* stationary. *Elements of* $\mathcal{A}^{\mathcal{X}}$ *are called* selectors *and every* $\pi \in \mathcal{A}^{\mathcal{X}}$ *is identified by the associated stationary policy* $(\pi, \pi, \pi, \ldots)$.

DEFINITION 2.5 *If* $\pi \in \mathcal{A}^{\mathcal{X}}$ *is an arbitrary selector let the corresponding* policy-evaluation operator $T_\pi : \mathcal{R}(\mathcal{X}) \to \mathcal{R}(\mathcal{X})$ *be defined as*

$$(T_\pi f)(x) = (\mathcal{Q}f)(x, \pi(x)).$$

In the literature the evaluation of Markov policies is defined with the help of the policy-evaluation operators:

DEFINITION 2.6 (BERTSEKAS, 1977) *The* evaluation function *of a finite-horizon Markov policy* $\pi = (\pi_0, \pi_1, \ldots, \pi_t)$ *is defined as* $v_\pi = T_{\pi_0} T_{\pi_1} \ldots T_{\pi_t} \ell$, *while the evaluation function of an infinite-horizon Markov policy* $\pi = (\pi_0, \pi_1, \ldots, \pi_t, \ldots)$ *is given by*

$$v_\pi = \lim_{t \to \infty} T_{\pi_0} T_{\pi_1} \ldots T_{\pi_t} \ell, \tag{1}$$

*assuming that the limit exists.*

If the policy is stationary ($\pi_t = \pi_0$ for all $t \geq 0$) the latter definition reduces to

$$v_\pi = \lim_{n \to \infty} T^n_{\pi_0} \ell. \tag{2}$$

*Note that if* $\mathcal{Q} : B(\mathcal{X}) \to B(\mathcal{X} \times \mathcal{A})$ *is a contraction then* $T_\pi$ *is a contraction with the same index* ($\pi \in \mathcal{A}^{\mathcal{X}}$) and so $v_\pi$ is well defined. The evaluation of arbitrary policies is more complicated and is the subject of the next section, but the following example may shed some light on the forthcoming definitions.

EXAMPLE 2.7 *Finite Markovian decision problems with the expected total cost criterion [2, 8].* $(\mathcal{X}, \mathcal{A}, p, c)$ is called a finite MDP if the following conditions hold:

1. $\mathcal{X}$ and $\mathcal{A}$ are finite sets;

2. $p : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathbf{R}$ and for each $a \in \mathcal{A}$, $p(\cdot, a, \cdot)$ is a transition probability matrix, i.e., for all $(x, a, y) \in \mathcal{X} \times \mathcal{A} \times \mathcal{X}$, $0 \leq p(x, a, y) \leq 1$; and for all $(x, a) \in \mathcal{X} \times \mathcal{A}$, $\sum_{y \in \mathcal{X}} p(x, a, y) = 1$;

3. $c : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathbf{R}$.

Now let $\pi$ be any policy. Then, for any $X$-valued random variable $\xi_0$, $\pi$ generates a probability measure $P = P_{\xi_0, \pi}$ over $(\mathcal{X} \times \mathcal{A})^{\mathbf{N}}$ which is uniquely defined by the finite-dimensional probabilities

$$P(x_0, a_0, x_1, a_1, \ldots, x_n, a_n) = p(\xi_0 = x_0)\delta(a_0, \pi_0(x_0))p(x_0, a_0, x_1) \ldots$$
$$\ldots p(x_{n-1}, a_{n-1}, x_n)\delta(a_n, \pi_n(x_n, a_{n-1}x_{n-1} \ldots a_0 x_0),$$

where $\delta : \mathcal{A}^2 \to \{0,1\}$ is defined by $\delta(a,b) = 1$ iff $a = b$. Clearly, one can construct a random sequence $(\xi_n, \alpha_n) \in \mathcal{X} \times \mathcal{A}$ (the controlled object) s.t. $P(\xi_{n+1}|\alpha_n, \xi_n, \ldots, \alpha_0, \xi_0) = p(\xi_n, \alpha_n, \xi_{n+1})$ and where $\alpha_n = \pi_n(\xi_n, \alpha_{n-1}\xi_{n-1}\ldots\alpha_0\xi_0)$. If $\xi_0$ is concentrated on $\{x\}$ for some $x \in X$ then $P_{\xi_0,\pi}$ is denoted by $P_{x,\pi}$.

Assume that $P(\xi_0 = x) > 0$ for all $x \in \mathcal{X}$. The evaluation of a policy $\pi$ in state $x$ is defined as

$$\hat{v}_\pi(x) \stackrel{\text{def}}{=} E_{\xi_0,\pi}\left[\sum_{t=0}^{\infty}\gamma^t c(\xi_t, \alpha_t, \xi_{t+1}) \,\middle|\, \xi_0 = x\right] = E_{x,\pi}\left[\sum_{t=0}^{\infty}\gamma^t c(\xi_t^{(x)}, \alpha_t^{(x)}, \xi_{t+1}^{(x)})\right]$$

where $0 < \gamma < 1$ is the discount factor and the expectation is taken w.r.t. $P_{\xi_0,\pi}$ (resp. $P_{x,\pi}$) and $\{(\xi_t, \alpha_t)\}$ (resp. $\{(\xi_t^{(x)}, \alpha_t^{(x)})\}$) is the controlled object corresponding to $\pi$ and the initial random state $\xi_0$ (resp. initial state $x$). The second equality in the above equation comes from the definitions and shows that $\hat{v}_\pi(x)$ is independent of the particular form of $\xi_0$. Now, if $\pi = (\pi_0, \pi_1, \ldots)$ is any policy then by the law of total probability

$$\hat{v}_\pi(x) = \sum_{y\in\mathcal{X}} P_{x,\pi}\left(\xi_1^{(x)} = y\right) E_{x,\pi}\left[\sum_{t=0}^{\infty}\gamma^t c(\xi_t^{(x)}, \alpha_t^{(x)}, \xi_{t+1}^{(x)}) \,\middle|\, \xi_1^{(x)} = y\right] =$$

$$= \sum_{y\in\mathcal{X}} p(x, \pi_0(x), y)\left(c(x, \pi_0(x), y) + \gamma E_{x,\pi}\left[\sum_{t=0}^{\infty}\gamma^t c(\xi_{t+1}^{(x)}, \alpha_{t+1}^{(x)}, \xi_{t+2}^{(x)}) \,\middle|\, \xi_1^{(x)} = y\right]\right) =$$

$$= \sum_{y\in\mathcal{X}} p(x, \pi_0(x), y)\left(c(x, \pi_0(x), y) + \gamma E_{y,\pi^x}\left[\sum_{t=0}^{\infty}\gamma^t c(\hat{\xi}_t^{(y)}, \hat{\alpha}_t^{(y)}, \hat{\xi}_{t+1}^{(y)})\right]\right) =$$

$$= \sum_{y\in\mathcal{X}} p(x, \pi_0(x), y)\Big(c(x, \pi_0(x), y) + \gamma\hat{v}_{\pi^x}(y)\Big) =$$

$$= \left(Q\hat{v}_{\pi^x}\right)(x, \pi_0(x)), \tag{3}$$

where $\pi^x$ denotes the policy executed after the first step, i.e., $\pi^x = (\pi_0^x, \pi_1^x, \ldots)$ with $\pi_t^x(x,h) = \pi_{t+1}(x, h\pi_0(x)x)$, $\{(\hat{\xi}_t^{(y)}, \hat{\alpha}_t^{(y)})\}$ is the corresponding controlled object given that the initial state is $y$, and $Q : \mathcal{R}(\mathcal{X}) \to \mathcal{R}(\mathcal{X} \times \mathcal{A})$ is defined by

$$(Qf)(x,a) = \sum_{y\in\mathcal{X}} p(x,a,y)(c(x,a,y) + \gamma f(y)).. \tag{4}$$

Equation 3 is called the Fundamental Equation [3] and will be proved to hold for general SDPs in the next section. Note that if $\pi$ is Markovian then $\pi^x = (\pi_1, \pi_2, \ldots)$ for any $x \in X$, and so Equation 3 yields that $v_{(\pi_0,\pi_1,\ldots)} = T_{\pi_0}\ldots T_{\pi_t} v_{(\pi_{t+1},\pi_{t+2},\ldots)}$. Therefore, for any given $\ell \in B(\mathcal{X})$, $\hat{v}_\pi = \lim_{t\to\infty} T_{\pi_0}\ldots T_{\pi_t}\ell = v_\pi$ since

$$\|T_{\pi_0}\ldots T_{\pi_t}\hat{v}_{(\pi_{t+1},\pi_{t+2},\ldots)} - T_{\pi_0}\ldots T_{\pi_t}\ell\| \le \gamma^{t+1}\|\hat{v}_{(\pi_{t+1},\pi_{t+2},\ldots)} - \ell\| \le \gamma^{t+1} C \stackrel{t\to\infty}{\longrightarrow} 0,$$

for some $C > 0$, where we exploited that for any selector $\pi$, $T_\pi$ is a contraction with index $\gamma$ and that $\sup_{\pi\in\Pi} \|v_\pi\| < \infty$.

Interesting "risk-sensitive" criteria may be obtained if $\mathcal{Q}$ is given by $(\mathcal{Q}f)(x,a) = (\sum_{y\in\mathcal{X}} p(x,a,y)(c(x,a,y) + \gamma f(y))^p)^{1/p}$, $1 \le p < \infty$, where $c$ and $f$ are assumed to be non-negative. This definition can be shown to give the minimax criterion when $p \to \infty$. The results derived below hold for these criteria as well.

**Objectives.** The objective of the decision maker is to choose a policy in such a way that the cost incurred during the usage of the policy is minimal. Of course, the smallest cost that can be achieved depends on the class of policies available for the decision maker.

DEFINITION 2.8 *The sets of general, Markov and stationary policies are denoted by $\Pi_g$, $\Pi_m$ and $\Pi_s$, respectively. Further, let*

$$v^{*\Delta}(x) = \inf_{\pi\in\Pi_\Delta} v_\pi(x),$$

*be the optimal cost function for the class $\Pi_\Delta$, where $\Delta$ is either $g$ or $m$ or $s$.*

For any $\varepsilon \ge 0$ and fixed $x \in \mathcal{X}$ the decision maker can assure a cost less than $v^{*\Delta}(x) + \varepsilon$ by the usage of an appropriate policy from $\Pi_\Delta$ but this policy will depend on $x$. Here we are interested in *uniformly* good policies:

DEFINITION 2.9 *Let $\Pi_\Delta(v) = \{\pi \in \Pi_\Delta \,|\, v_\pi \le v\}$, that is $\Pi_\Delta(v)$ contains the policies from $\Pi_\Delta$ whose cost is uniformly less than or equal to $v$. A policy is said to be* (uniformly) $\varepsilon$-optimal *if it is contained in $\Pi_g(v^{*g} + \varepsilon)$.*[2]

The objective of sequential decision problems is to give conditions under which $\Pi_\Delta(v^{*g} + \varepsilon)$ is guaranteed to be non-empty when $\varepsilon > 0$ or $\varepsilon = 0$.

DEFINITION 2.10 *Elements of $\Pi_g(v^{*g})$, $\Pi_m(v^{*g})$, and $\Pi_s(v^{*g})$ are called optimal, optimal Markovian and optimal stationary policies, respectively.*

**The Fundamental Equation.** Now we define the evaluation function associated to non-Markovian policies and derive the fundamental equation.

DEFINITION 2.11 *If $\pi = (\pi_0, \pi_1, \ldots, \pi_t, \ldots)$ is an arbitrary policy then $\pi^t$ denotes the* $t$-truncation *of $\pi$: $\pi^t = (\pi_0, \pi_1, \ldots, \pi_t)$. Further, let $\mathcal{P}_t$ and $\mathcal{P}$ denote the set of t-truncated (finite-horizon) policies and the set of (infinite horizon) policies, respectively. The s-truncation operator for t-truncated policies is defined similarly if $s \le t$.*

DEFINITION 2.12 *The* shift-operator, $S_{(x,a)} : \mathcal{P} \to \mathcal{P}$, *for any pair $(x,a) \in \mathcal{X} \times \mathcal{A}$ is defined in the following way:*

$$S_{(x,a)}\pi = (\pi'_0, \pi'_1, \ldots),$$

[2]If $v$ is a real valued function over $\mathcal{X}$ and $\varepsilon$ is real then $v + \varepsilon$ stands for the function $v(x) + \varepsilon$.

*where $\pi'_t$ is defined by*

$$\pi'_t(x,h) = \pi_{t+1}(x, hax)$$

*for all $t \geq 0$. We shall write $\pi^x$ for $S_{(x,\pi_0(x))}\pi$ and call $\pi^x$ the* derived policy. *For t-truncated policies $S_{(x,a)}$ is defined in the same way, just now $S_{(x,a)} : \mathcal{P}_t \to \mathcal{P}_{t-1}, t \geq 1$.*

The above definition means that $\pi^x \in \mathcal{P}_{t-1}$ holds for any $\pi \in \mathcal{P}_t$ and $x \in \mathcal{X}$. The following proposition follows from the definitions.

PROPOSITION 2.13 *$\pi^{t,x} = \pi^{x,t-1}$ and thus if $\pi \in \mathcal{P}_t$ then $\pi^{t,x} = \pi^{x,t-1} \in \mathcal{P}_{t-1}$, $t \geq 1$.*

Now we are in the position to give the definition of the evaluation of policies with finite horizon.

DEFINITION 2.14 *If $\pi \in \mathcal{P}_0$, i.e., $\pi = (\pi_0)$ then $v_\pi(x) = (Q\ell)(x, \pi_0(x))$, where $\ell \in \mathcal{R}(\mathcal{X})$ is the terminal cost function. Assume that the evaluation of policies in $\mathcal{P}_t$ is already defined. Let $\pi \in \mathcal{P}_{t+1}$. Then*

$$v_\pi(x) = (Qv_{\pi^x})(x, \pi_0(x)). \tag{5}$$

Since $\pi^x \in \mathcal{P}_t$, $v_{\pi^x}$ is already defined and thus (5) is well defined. One can interpret this definition as follows: $\pi^x$ is the policy that is applied after the first decision. The cost of the derived policy is $v_{\pi^x}$. This cost together with the cost of the first action (the first being $\pi_0(x)$ in state $x$) gives the total cost of the policy.

EXAMPLE 2.15 If $\pi$ is a $t$-horizon policy in an MDP $(\mathcal{X}, \mathcal{A}, p, c)$ (cf. Example 2.7) and we set

$$\hat{v}_\pi^{(t)}(x) = E\left[\sum_{n=0}^{t-1} \gamma^n c(\xi_n, \alpha_n, \xi_n) \,\middle|\, \xi_0 = x\right],$$

where $\{(\xi_n, \alpha_n)\}$ is the controlled object corresponding to $\pi$ and the random initial state $\xi_0$. The argument of Example 2.7 gives that $v_\pi^{(t)} = \hat{v}_\pi^{(t)}$, where $v_\pi^{(t)}$ is the evaluation of $\pi$ in the sense of Definition 2.14 in the SDP $(\mathcal{X}, \mathcal{A}, Q, \ell)$, with $Q$ given by (4) and where $\ell(x) = 0$ for all $x \in \mathcal{X}$.

The evaluation of an infinite horizon policy is defined as the limit of the evaluations of the finite horizon truncations of that policy:

DEFINITION 2.16 *Let $\pi \in \mathcal{P} = \mathcal{P}_\infty$. Then the total cost of $\pi$ for initial state $x$ is given by*

$$v_\pi(x) = \liminf_{t\to\infty} v_{\pi^t}(x), \qquad x \in \mathcal{X}.$$

EXAMPLE 2.17 Continuing the above example, if $\pi$ is an arbitrary policy then (by the boundedness of $c$)

$$\hat{v}_\pi(x) \overset{\text{def}}{=} E\left[\sum_{n=0}^{\infty} \gamma^n c(\xi_n, \alpha_n, \xi_n) \mid \xi_0 = x\right] = \lim_{t\to\infty} E\left[\sum_{n=0}^{t-1} \gamma^n c(\xi_n, \alpha_n, \xi_n) \mid \xi_0 = x\right],$$

and so $v_\pi = \hat{v}_\pi$.

DEFINITION 2.18 *$Q$ is said to be monotone if $Qv \le Qu$ whenever $u \le v$.*

*In what follows we will always assume that $Q$ is monotone.*

Equation 6 below, which in harmony with [3] we call the fundamental equation (FE), has already been derived for MDPs in Example 2.7. Here we show that it holds in general SDPs when $Q$ is continuous.

THEOREM 2.19 *If $Q$ is continuous then*

$$v_\pi(x) = (Qv_{\pi^x})(x, \pi_0(x)). \tag{6}$$

*Proof.* Let $v_t = v_{\pi^t}$ and let $\mu = \pi^{t+1}$. By definition $v_\mu(x) = (Qv_{\mu^x})(x, \mu_0(x))$. According to Proposition 2.13 $\mu^x = \pi^{t+1,x} = \pi^{x,t}$ and $\mu_0 = \pi_0$, therefore

$$v_{\pi^{t+1}}(x) = (Qv_{\pi^{x,t}})(x, \pi_0(x)). \tag{7}$$

Now, let $t$ tend to infinity and consider the lim inf of both sides of the above equation:

$$v_\pi(x) = \liminf_{t\to\infty}(Qv_{\pi^{x,t}})(x,\pi_0(x)) = \left(Q[\liminf_{t\to\infty} v_{\pi^{x,t}}]\right)(x,\pi_0(x)) = (Qv_{\pi^x})(x,\pi_0(x)),$$

where in the first equation we exploited the definition $v_\pi$, in the second equation we used that $Q$ is monotone and is continuous, and in the third equation the definition of $v_{\pi^x}$ was utilised. □

COROLLARY 2.20 *Assume that $Q$ is a contraction and is continuous. Then $v_{\pi^t}$ converges to $v_\pi$, i.e., in Definition 2.16* lim inf *can be replaced by* lim, *and for any Markovian policy $\pi$, the evaluation function associated to $\pi$ in the sense of Definition 2.16 coincides with the evaluation function in the sense of Definition 2.6. Moreover, if $\pi$ is stationary then $T_\pi^n v_0$ converges to $v_\pi$, where $v_0 \in B(\mathcal{X})$ is arbitrary, and $v_\pi = T_\pi v_\pi$.*

*Proof.* Recall that in Definition 2.6 the evaluation of a Markovian policy $\pi = (\pi_0, \pi_1, \ldots, \pi_t, \ldots)$ was defined as the limit $\hat{v}_\pi(x) = \lim_{t\to\infty}\left(T_{\pi_0}\cdots(T_{\pi_{t-1}}(T_{\pi_t}\ell))\ldots\right)$. Easily, $T_{\pi_0}\ldots T_{\pi_{t-1}}T_{\pi_t}\ell = v_{\pi^t}$, so the definition of Bertsekas coincides with that of ours. The rest of the statement follows from the Banach fixed-point theorem. □

## Uniformly Optimal Policies.

DEFINITION 2.21 *Policy $\pi$ is said to be uniformly $\varepsilon$-optimal if, for all $x \in \mathcal{X}$:*

$$v_\pi(x) \le \begin{cases} v^{*g}(x) + \varepsilon, & \text{if } v^{*g}(x) > -\infty; \\ -1/\varepsilon, & \text{otherwise.} \end{cases}$$

THEOREM 2.22 *If the FE is satisfied then for all $\varepsilon > 0$ there exists an $\varepsilon$-optimal policy.*

*Proof.* Fix an arbitrary $x \in \mathcal{X}$. By the definition of $v^{*g}(x)$ there exists a policy ${}_x\pi = ({}_x\pi_0, {}_x\pi_1, \ldots)$ for which $v_{{}_x\pi}(x) \leq v^{*g}(x) + \varepsilon$ when $v^{*g}(x) > -\infty$ and $v_{{}_x\pi}(x) \leq -1/\varepsilon$, otherwise. We define a policy which will be $\varepsilon$-optimal by taking the actions prescribed by ${}_x\pi$ when $x$ is the starting state of the decision process. The resulting policy, called the *combination* of the policies ${}_x\pi$, is given formally by $\pi_0(x) = {}_x\pi_0(x)$ and $\pi_t(y, hax) = {}_x\pi_t(y, hax)$. We claim that $v_\pi(x) = v_{{}_x\pi}(x)$ and thus $\pi$ is uniformly $\varepsilon$-optimal. Indeed, $\pi^x = ({}_x\pi)^x$ and $\pi_0(x) = {}_x\pi_0(x)$ and so $v_\pi(x) = (Qv_{{}_x\pi^x})(x, \pi_0(x)) = (Qv_{{}_x\pi^x})(x, {}_x\pi_0(x)) = v_{{}_x\pi}(x)$. □

**Finite Horizon Problems.**

DEFINITION 2.23 *The optimal cost function for n-horizon problems is defined by*

$$v_n^{*\Delta}(x) = \inf_{\pi \in \mathcal{P}_n^\Delta} v_\pi,$$

*where* $\mathcal{P}_n^\Delta = \{\, \pi^n \mid \pi \in \Pi_\Delta \,\}$, *and* $\Delta \in \{g, m, s\}$.

DEFINITION 2.24 *The* optimal cost operator $T : \mathcal{R}(\mathcal{X}) \to \mathcal{R}(\mathcal{X})$ *associated with the SDP* $(\mathcal{X}, \mathcal{A}, \mathcal{Q}, \ell)$ *is defined by*

$$(Tf)(x) = \inf_{a \in \mathcal{A}(x)} (\mathcal{Q}f)(x, a).$$

It is immediate from the definition and by the triangle inequality that if $\mathcal{Q}$ is a contraction with index $\gamma$ then the optimal cost operator is a contraction with the same index.

DEFINITION 2.25 $\mathcal{Q}$ *is called* upper semi-continuous *if for every (pointwise) convergent sequence of functions* $v_t \in \mathcal{R}(\mathcal{X})$ *for which* $v_t \geq \lim_{t\to\infty} v_t$ *we have*

$$\lim_{t\to\infty} \mathcal{Q}v_t = \mathcal{Q}(\lim_{t\to\infty} v_t)$$

THEOREM 2.26 (OPTIMALITY EQUATION FOR FINITE HORIZON PROBLEMS) *The optimal cost functions of the n-horizon problem satisfies*

$$v_n^{*g} = v_n^{*m} = T^n \ell \tag{8}$$

*provided that* $\mathcal{Q}$ *is USC and the FE is satisfied.*

*Proof.* We prove the proposition by induction. One immediately sees that the proposition holds for $n = 1$. Assume that we have already proven the proposition for $n$. Firstly, we prove that $T^{n+1}\ell \leq v_{n+1}^*$. Note that this inequality will follow from the FE and the monotonicity of $\mathcal{Q}$ alone: no continuity assumption is needed here.

Let $\pi \in \mathcal{P}_{n+1}$. We show that $T^{n+1}\ell \le v_\pi$. By the induction hypothesis $(T^{n+1}\ell)(x) = (Tv_n^{*g})(x)$. According to the FE, $v_\pi(x) = (Qv_{\pi^x})(x, \pi_0(x))$. Since $\pi^x \in \mathcal{P}_n$ so $v_{\pi^x} \ge v_n^{*g}$. Since $Q$ is monotone it follows that

$$(Tv_n^{*g})(x) = \inf_{a\in\mathcal{A}(x)} (Qv_n^{*g})(x,a) \le \inf_{a\in\mathcal{A}(x)} (Qv_{\pi^x})(x,a) \le (Qv_{\pi^x})(x,\pi_0(x) = v_\pi(x).$$

This holds for arbitrary $\pi \in \mathcal{P}_{n+1}$ and thus $Tv_n^{*g} \le v_{n+1}^{*g}$. Using the induction hypothesis we find that $T^{n+1}\ell \le v_{n+1}^{*g}$.

Now let us prove the reverse inequality, i.e., that $v_{n+1}^{*g} \le T^{n+1}\ell$ holds. Let us choose a sequence of Markovian policies $\pi_k \in \mathcal{P}_n$ such that $v_{\pi_k}$ converges to $v_n^{*m}$. Clearly, $v_{\pi_k} \ge v_n^{*m}$. Now let $\mu_j : \mathcal{X} \to \mathcal{A}$ be a sequence of mappings satisfying $\lim_{j\to\infty} T_{\mu_j} v_n^{*g} = Tv_n^{*g}$. Now consider the policies $\nu_{k,j} = \pi_k \bigoplus \mu_j \in \mathcal{P}_{n+1}$: the first $n$ actions of $\nu_{k,j}$ are the actions prescribed by $\pi_k$ while the last action is the action prescribed by $\mu_j$. It is clear that $v_{n+1}^{*g} \le v_{n+1}^{*m} \le v_{\nu_{k,j}} = T_{\mu_j} v_{\pi_k}$: the last equality follows from the FE. Taking the limit in $k$ we get that

$$v_{n+1}^{*m} \le \lim_{k\to\infty} T_{\mu_j} v_{\pi_k} = T_{\mu_j}(\lim_{k\to\infty} v_{\pi_k}) = T_{\mu_j} v_n^{*m}$$

holds owing to the choice of the policies $\pi_k$ and since $Q$ is USC. Now taking the limit in $j$ the induction hypothesis yields that $v_{n+1}^{*g} \le v_{n+1}^{*m} \le Tv_n^{*m} = T^{n+1}\ell$ which finally gives that $v_{n+1}^{*g} = v_{n+1}^{*m} = T^{n+1}\ell$, completing the proof. □

The following example shows that the conditions of the previous theorem are indeed essential.

EXAMPLE 2.27 [1] Let $\mathcal{X} = \{0\}$ and $\mathcal{A} = (0,1]$, $\ell(0) = 0$, and $(Qf)(0,a) = 1$, if $f(0) > 0$; and $(Qf)(0,a) = a$, otherwise. Note that $Q$ is not USC. It is easy to see that $0 = v_\infty(0) = (T^n\ell)(0) < v_n^{*g}(0) = 1 = v^{*g}(0)$ if $n \ge 2$.

**The Bellman Optimality Equation.** According to Theorem 2.26, if $v_n^{*g}$ converges to $v^{*g}$ then $v^{*g}$ can be computed as the limit of the function sequence $v_0 = \ell$, $v_{t+1} = Tv_t$ provided that $Q$ is USC and the FE holds. The convergence of $v_n^{*g}$ to $v^{*g}$ expressed in another way means that the inf and lim operations can be interchanged in the definition of $v^{*g}$:

$$v^{*g} = \inf_{\pi\in\mathcal{P}} \lim_{n\to\infty} v_{\pi^n} = \lim_{n\to\infty} \inf_{\pi\in\mathcal{P}} v_{\pi^n} = \lim_{n\to\infty} v_n^{*g}. \qquad (9)$$

THEOREM 2.28 *(i) Assume that $Q$ is continuous and set $v_\infty = \limsup_{n\to\infty} T^n\ell$. Then*

$$v_\infty \le v^{*g}. \qquad (10)$$

*(ii) If we further assume that $Q$ is a contraction then $\lim_{n\to\infty} v_n^{*g} = \lim_{n\to\infty} T^n v_0 = v^{*g} = v^{*m}$, where $v_0 \in B(\mathcal{X})$ is arbitrary, and*

$$Tv^{*g} = v^{*g}. \qquad (11)$$

*Proof.* (i) Note that by Theorems 2.19 & 2.26 $v_\infty = \limsup_{n\to\infty} v_n^{*g}$. Let $x \in \mathcal{X}$ and let $c$ be a number s.t. $c > v^{*g}(x)$. By the definition of $v^{*g}$ there exists a policy $\pi \in \mathcal{P}$ such that $v_\pi(x) < c$. Furthermore, since $v_\pi(x) = \lim_{n\to\infty} v_{\pi^n}(x)$ there exists a number $n_0$ such that from $n > n_0$ it follows that $v_{\pi^n}(x) < c$. Thus if $n > n_0$ then $v_n^{*g}(x) < c$ and consequently $\limsup_{n\to\infty} v_n^{*g}(x) < c$. Since $c$ and $x$ were arbitrary, we obtain the desired inequality.

(ii) By the Banach fixed-point theorem $v_\infty = \lim_{n\to\infty} T^n \ell = \lim_{n\to\infty} T^n v_0$ and $Tv_\infty = v_\infty$. It is sufficient to prove that $v^{*g} \le Tv^{*g}$ since then iterating this inequality will yield $v^{*g} \le T^n v^{*g} \to v_\infty, n \to \infty$, which together with Part (i) shows (11). Let $\pi_n$ be a sequence of $1/n$-uniformly optimal policies. Such policies exist by Theorem 2.22. Further, let $\mu_n$ be a selector such that $T_{\mu_n} v_{\pi_n} \le T v_{\pi_n} + 1/n$. Then $v^{*g} \le v_{\mu_n \oplus \pi_n} \le (Tv_{\pi_n}) + 1/n$, and taking the limit in $n$ yields the desired inequality. □

**Existence of Optimal Stationary Policies.**

DEFINITION 2.29 *A stationary policy $\phi$ is said to be* greedy *w.r.t. $v \in \mathcal{R}(\mathcal{X})$ if*

$$T_\phi v = Tv,$$

*i.e., if for each $x \in \mathcal{X}$, $(Qv)(x, \phi(x)) = (Tv)(x) = \inf_{a\in\mathcal{A}}(Qv)(x,a)$.*

Note that the finiteness of $\mathcal{A}$ assures the existence of greedy policies w.r.t. any function $v \in \mathcal{R}(\mathcal{X})$. If $\mathcal{A}$ is infinite special continuity assumptions are needed on $Q$ for the existence of greedy policies (see [1] for further information on this). The next theorem shows that greediness is a useful concept under the appropriate conditions since the knowledge of the optimal cost function can be sufficient to find optimal stationary policies.

THEOREM 2.30 *If $Q$ is a contraction and is continuous then optimal stationary policies are greedy w.r.t. $v^{*g}$; and vice versa.*

*Proof.* If $\phi$ is greedy w.r.t. $v^{*g}$ then $T_\phi v^{*g} = Tv^{*g} = v^{*g}$ and by induction we get that $T_\phi^n v^{*g} = v^{*g}$ holds for all $n$. Since by Corollary 2.20 the l.h.s. converges to $v_\phi$ as $n \to \infty$, we get that $v_\phi = v^{*g}$, i.e., $\phi$ is optimal. Now, if $\phi$ is an optimal stationary policy then $Tv^{*g} = v^{*g} = v_\phi = T_\phi v_\phi = T_\phi v^*$, showing the greediness of $\phi$. □

Theorems 2.28 & 2.30 are at the very core of the learning algorithms used in the robotic experiments. In particular, a theorem was proven in [5] which shows that in contractive models (i.e., when $Q$ is a contraction) value iteration can be combined with learning processes without effecting the convergence. In [9] and [6] examples are shown for asymptotically optimal learning policies which use the adaptive value iteration scheme.

**Final Remarks.** Similar statements hold for models when $(Q\ell)(\cdot,a) \geq \ell$ or when $(Q\ell)(\cdot,a) \leq \ell$ $(a \in \mathcal{A})$ in which cases we require $Q$ to be lower (resp. upper) semi-continuous on the set of functions $\{v \in \mathcal{R}(\mathcal{X}) \,|\, v \geq \ell\}$ (resp. $\{v \in \mathcal{R}(\mathcal{X}) \,|\, v \leq \ell\}$). In such cases the analysis should be based on the monotonicity of the various function sequences involved. However, problems related to the existence of stationary optimal policies become more complicated: in fact for models satisfying $(Q\ell)(\cdot,a) \geq \ell$ (these are called increasing models) value iteration does not necessarily converge to $v^{*g}$, but greedy policies w.r.t. $v^{*g}$ are optimal; whilst in models satisfying the opposite inequality, $(Q\ell)(\cdot,a) \leq \ell$ (these are called decreasing models) value iteration does always converge to $v^{*g}$ but greedy policies w.r.t. $v^{*g}$ are not necessarily optimal. It is also worth noting that Howard's policy improvement theorem [4] is valid in increasing or contractive models, and when iterated converges to optimum in contractive models [5] but does not necessarily converge to optimum in increasing ones. In certain contractive models one can estimate the speed of convergence of both the value and the policy iteration methods which turns out to be pseudo-polynomial [5].

**Acknowledgements.** This work was partially supported by OTKA Grants. F20132 and T014548 and by a grant provided by the Hungarian Educational Ministry under contract no. FKFP 1354/1997. I would like to express my sincere gratitude to András Krámli for the many discussions on the topics of this article.

## References

[1] D. P. Bertsekas. Monotone mappings with application in dynamic programming. *SIAM J. Control and Optimization*, 15(3):438–464, 1977.

[2] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1989.

[3] E. Dynkin and A. Yushkevich. *Controlled Markov Processes.* Springer-Verlag, Berlin, 1979.

[4] R. A. Howard. *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge, MA, 1960.

[5] M. L. Littman and Cs. Szepesvári. A Generalized Reinforcement Learning Model: Convergence and applications. In *Int. Conf. on Machine Learning*, pages 310–318, 1996.

[6] Cs. Szepesvári. Learning and exploitation do not conflict under minimax optimality. In M. Someren and G. Widmer, editors, *Machine Learning: ECML'97 (9th European Conf. on Machine Learning, Proceedings)*, volume 1224 of *Lecture Notes in Artificial Intelligence*, pages 242–249. Springer, Berlin, 1997.

[7] Zs. Kalmár, Cs. Szepesvári, and A. Lőrincz. Module based reinforcement learning for a real robot. In *Proc. of the 6th European Workshop on Learning Robots*, pages 22–32, 1997.

[8] S. Ross. *Applied Probability Models with Optimization Applications*. Holden Day, San Francisco, California, 1970.

[9] S. Singh, T. Jaakkola, M. L. Littman, and Cs. Szepesvári. On the convergence of single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 1997. submitted.

[10] S. Verdu and H. Poor. Abstract dynamic programming models under commutativity conditions. *SIAM J. Control and Optimization*, 25(4):990–1006, 1987.

*Received October, 1997*

# On Decision-Mappings Related to Process Network Synthesis Problem

Z. Blázsik [*] Cs. Holló[†] B. Imreh[†]

**Abstract**

Process network synthesis (PNS) has enormous practical impact and a structural model can be given for it on the basis of a combinatorial approach. An important tool of this approach is the notion of the decision-mapping. In the present work, the number of the consistent decision-mappings is counted and an upper bound is presented for the number of the feasible solutions of a PNS problem.

## Introduction

In a manufacturing system, materials of different properties are converted into desired products through various physical, chemical, and biological transformations. Devices in which these transformations are carried out are called operating units and a manufacturing system can be considered as a network of operating units, i.e., *process network*. Naturally, minimizing the cost of a process network is indeed essential. For this purpose, several papers have appeared for solving PNS problems by global optimization methods (see, e.g., [2] and [8]) and by combinatorial approach based on the feasible graphs of processes (cf. [3], [4], [5], [7], and [9]).

In this paper, using the combinatorial approach, the number of the consistent decision-mappings is counted, furthermore, an upper bound is given for the number of the feasible solutions of a PNS problem. The paper is organized as follows: Section 1 reviews the precise definition of the structural model of PNS problem and introduces some relevant basic concepts. In Section 2, the number of the consistent decision-mappings over a nonempty set is calculated. On the basis of the relationship between the maximal consistent decision-mappings and the feasible solutions, an upper bound is presented for the number of the feasible solutions in Section 3. Finally, Section 4 contains an example for illustrating this bound.

[*]Research Group on Artificial Intelligence of the Hungarian Academy of Sciences, Aradi vértanúk tere 1, H-6720 Szeged, Hungary

[†]Dept. of Informatics, József Attila University, Árpád tér 2, H-6720 Szeged, Hungary

## 1 Preliminaries

Let $M$ be a given finite set of objects which are materials capable of being converted or transformed by a process. Transformation between two subsets of $M$ occurs in an operating unit. It is necessary to link this operating unit to others through the elements of these two subsets of $M$. The resultant structure is a process graph (see [4] and [5]) defined as follows.

Let $M$ be a finite nonempty set, and also let $O \subseteq \wp'(M) \times \wp'(M)$ with $O \neq \emptyset$ and $M \cap O = \emptyset$, where $\wp'(M)$ denotes the set of all nonempty subsets of $M$. The elements of $O$ are called *operating units*: for operating unit $u = (\alpha, \beta) \in O$, $\alpha$ and $\beta$ are called the *input-set* and *output-set* of $u$, respectively. Pair $(M, O)$ is defined as *process graph* or P-*graph* in short. The set of vertices of this directed graph is $M \cup O$, and the set of arcs is $A = A_1 \cup A_2$ with $A_1 = \{(X, Y) : Y = (\alpha, \beta) \in O$ and $X \in \alpha\}$ and $A_2 = \{(Y, X) : Y = (\alpha, \beta) \in O$ and $X \in \beta\}$. If there exist vertices $X_1, X_2, ..., X_n$, such that $(X_1, X_2), (X_2, X_3), \ldots, (X_{n-1}, X_n)$ are arcs of process graph $(M, O)$, then $[X_1, X_n]$ is defined to be a *path* from vertex $X_1$ to vertex $X_n$. Let process graphs $(m, o)$ and $(M, O)$ be given; $(m, o)$ is defined to be a *subgraph* of $(M, O)$, if $m \subseteq M$ and $o \subseteq O$.

To define a structural model of PNS, the set of materials to be included in the model need be specified. In the sequal, each material is an element of $M^*$, an arbitrarily specified infinite set of the available materials. From the technical point of view, we suppose that $M^* \cap (\wp'(M^*) \times \wp'(M^*)) = \emptyset$. Now, a process design problem can be defined from a structural point of view in the following way. By a *structural model* of PNS, we mean the triplet, $\mathbf{M} = (P, R, O)$, where $P \subseteq M^*$ and $O \subseteq \wp'(M^*) \times \wp'(M^*)$ are finite nonempty sets representing the set of desired products and that of available operating units, respectively, and $R \subseteq M^*$ is a finite set representing the set of raw materials. Moreover, $P \cap R = \emptyset$, and $\alpha$, $\beta$ are finite sets for all operating units $u = (\alpha, \beta) \in O$.

Now, let $\mathbf{M} = (P, R, O)$ be a structural model of PNS; then, we can assign a P-graph to $\mathbf{M}$ as follows. Let $M'$ denote the set of materials belonging to the operating units from $O$ and $M$ denote set $M' \cup P \cup R$. It can be seen that $M$ and $O$ are nonempty finite sets and that $O \subseteq \wp'(M) \times \wp'(M)$ and $M \cap O = \emptyset$. Thus, $(M, O)$ is a P-graph representing the interconnections among the operating units in set $O$. Since $M \cap O = \emptyset$, the vertices which are the points in $(M, O)$ can be divided into the two disjoint sets, $M$ and $O$. The elements of $M$ are called *material points* and those of $O$, *unit points* of $(M, O)$. A subgraph of $(M, O)$ can be assigned to each feasible process of the PNS problem; such a subgraph represents the structure or network of the process under consideration. If additional constraints, e.g., the material balance, are disregarded, the subgraphs of $(M, O)$, which can be assigned to the feasible processes, have common combinatorial properties. Such properties, explored in [5], are given below.

Subgraph $(m, o)$ of $(M, O)$ is called a *feasible solution* of $\mathbf{M} = (P, R, O)$ if the following properties are satisfied:

(A1) $P \subseteq m$,

(A2) $\forall X \in m$, $X \in R \Leftrightarrow$ there exists no $(Y, X)$ arc in $(m, o)$,

(A3) $\forall Y_0 \in o$, $\exists$ path $[Y_o, Y_n]$ with $Y_n \in P$,

(A4) $\forall X \in m$, $\exists(\alpha, \beta) \in o$ such that $X \in \alpha \cup \beta$.

Let us denote the set of the feasible solutions of $\mathbf{M}$ by $S(\mathbf{M})$. It is easy to see that $S(\mathbf{M})$ is closed under the finite union. Consequently,

$$\cup\{(m, o) : (m, o) \in S(\mathbf{M})\}$$

is also a feasible solution provided that $S(\mathbf{M}) \neq \emptyset$; it is the greatest feasible solution with respect to the relation, subgraph ordering. This distinguished graph is called the *maximal structure* of $\mathbf{M}$.

Now, a simple class of PNS problems can be defined, a class of such PNS problems in which each operating unit has a positive fixed charge. We are to find a feasible process with the minimum cost; by the cost of a process, we mean the sum of the fixed charges of the operating units belonging to the process of interest. Each feasible process in this class of PNS problems is uniquely determined from the corresponding feasible solution and vice versa. Hence, the problem under consideration can be formalized as follows.

Let a structural model of PNS problem $\mathbf{M} = (P, R, O)$ be given; moreover, let $z$ be a positive real-valued function defined on $O$. The basic model is then

$$\text{(i)} \qquad \min\{\sum_{u \in o} z(u) : (m, o) \in S(\mathbf{M})\}.$$

It has been proved [1] that this PNS problem is NP-hard; therefore, the branch-and-bound technique may be a possible tool for its solution (see [7] and [9]).

## 2 Consistent decision-mappings

In the branch-and-bound procedures for solving PNS problems, the notion of the decision-mapping (see [6]) has been applied. Let $\mathbf{M} = (P, R, O)$ be a structural model of PNS. Then, P-graph $(M, O)$ of $\mathbf{M}$ determines a function $\Delta$ of $M \setminus R$ into $\wp'(O)$ as follows. For any material $X \in M \setminus R$, let

$$\Delta(X) = \{(\alpha, \beta) : (\alpha, \beta) \in O \ \& \ X \in \beta\}.$$

Let $m$ be a subset of $M \setminus R$; furthermore, let $\delta(X)$ be a subset of $\Delta(X)$ for each $X \in m$. Mapping $\delta$ from set $m$ into the set of subsets of $O$, $\delta[m] = \{(X, \delta(X)) : X \in m\}$, is called a *decision-mapping belonging* to $\mathbf{M}$; $\delta[m]$ is said to be *consistent* when $\delta(X) \cap \Delta(Y) \subseteq \delta(Y)$ is valid for all $X, Y \in m$, and the set of all consistent decision-mappings of $\mathbf{M}$ is denoted by $\Omega_{\mathbf{M}}$. In particular, if $\delta[m] \in \Omega_{\mathbf{M}}$ and $m = M \setminus R$, then sometimes we use the shorter notation $\delta$ instead of $\delta[M \setminus R]$.

A decision-mapping can be visualised as a sequence of decisions, each of which is concerned with a single material involved in the process being synthesized; it identifies the set of operating units to be considered for producing directly the material of interest. The meaning of the consistency can be presented as follows. Material $X$ is to be produced by operating units included in $\delta(X)$. Then, those operating units of $\delta(X)$ that also participate in the production of material $Y$, i.e., $\delta(X) \cap \Delta(Y)$, must be considered for the production of material $Y$, and thus, $\delta(Y) \supseteq \delta(X) \cap \Delta(Y)$.

We define function *op* on $\Omega_{\mathrm{M}}$ for selecting the set of those operating units that are decided to produce any of the materials in set $m$ based on consistent decision-mapping $\delta[m]$. Formally, for any $\delta[m] \in \Omega_{\mathrm{M}}$,

$$op(\delta[m]) = \cup\{\delta(X) : X \in m\}.$$

In what follows, we need the following functions. For any finite set of operating units $o$, let

$$mat^{in}(o) = \cup\{\alpha : (\alpha, \beta) \in o\}, \qquad mat^{out}(o) = \cup\{\beta : (\alpha, \beta) \in o\}.$$

Let $\delta_1[m_1]$ and $\delta_2[m_2]$ be arbitrary consistent decision-mappings. Then, $\delta_2[m_2]$ is called an *extension* of $\delta_1[m_1]$ if $m_1 \subseteq m_2$ and $\delta_1(X) = \delta_2(X)$ for all $X \in m_1$; this is denoted by $\delta_1[m_1] \leq \delta_2[m_2]$. Relation extension is reflexive, antisymmetric and transitive; hence, it is a partial ordering on $\Omega_{\mathrm{M}}$. Let us denote the set of all maximal elements of this partially ordered set by $\Omega'_{\mathrm{M}}$. Regarding the number of the consistent decision-mappings over a nonempty set $m$, the following statement is valid.

**Theorem.** *For every $\emptyset \neq m \subseteq M \setminus R$, the number of the consistent decision-mappings defined on $m$ is $2^{|\cup\{\Delta(X):X\in m\}|}$.*

*Proof.* We proceed by induction on $|m|$. If $|m| = 1$, then $X \to Q$ is a consistent decision-mapping for every subset $Q$ of $\Delta(X)$ where $X$ denotes the single element of $m$. Therefore, the required number is $2^{|\Delta(x)|}$.

Now, let $1 \leq i < |M \setminus R|$ be an arbitrary integer, and let us suppose that the statement is valid for all $m \subseteq M \setminus R$ with $|m| = i$. Let us consider an arbitrary subset $m'$ ($\subseteq M \setminus R$) consisting of $i+1$ elements. Without loss of generality, it can be assumed that $m' = \{X_1, \ldots, X_i, X_{i+1}\}$. Let $W = \Delta(X_{i+1}) \setminus (\cup\{\Delta(X_t) : t = 1, \ldots, i\})$. The following two cases are distinguished depending on $W$.

*Case* 1. $W = \emptyset$. From the definition of the consistent decision-mapping, the following observation can be directly obtained. For each consistent decision-mapping $\delta[m']$, the restriction of $\delta[m']$ to set $\{X_1, \ldots, X_i\}$ is .also consistent decision-mapping. On the other hand, if two consistent decision-mappings defined on the same set are different, then their extensions constitute two disjoint sets. In the light of these observations, it is enough to prove that consistent decision-mapping $\delta[\{X_1, \ldots, X_i\}]$ has one and only one extension to $\{X_1, \ldots, X_i, X_{i+1}\}$.

First, we construct an extension of $\delta[\{X_1, \ldots, X_i\}]$ to $\{X_1, \ldots, X_i, X_{i+1}\}$. The new decision-mapping is defined as follows. Let

$\delta'(X_{i+1}) = \{(\alpha, \beta) : (\alpha, \beta) \in \Delta(X_{i+1})\ \&\ (\alpha, \beta) \in \delta(X_j)$ for some $j \in \{1, \dots, i\}\}$,

and

$\delta'(X_t) = \delta(X_t)$ for all $t$, $t = 1, \dots, i$.

Regarding the consistency of $\delta'[\{X_1, \dots, X_i, X_{i+1}\}]$, we have to prove that

(1) $\delta'(X_t) \cap \Delta(X_{i+1}) \subseteq \delta'(X_{i+1})$,

and

(2) $\delta'(X_{i+1}) \cap \Delta(X_t) \subseteq \delta'(X_t)$

are valid for all $X_t \in \{X_1, \dots, X_i\}$. The validity of (1) follows from the definition of $\delta'$. For verifying (2), let $(\alpha, \beta) \in \delta'(X_{i+1}) \cap \Delta(X_t)$ for some $t \in \{1, \dots, i\}$. Since $(\alpha, \beta) \in \delta'(X_{i+1})$, there exists a $j \in \{1, \dots, i\}$ with $(\alpha, \beta) \in \delta(X_j) \cap \Delta(X_{i+1})$. Then, $(\alpha, \beta) \in \delta(X_j) \cap \Delta(X_t)$. On the other hand, $j, t \in \{1, \dots, i\}$ and the consistency of $\delta$ results in $\delta(X_j) \cap \Delta(X_t) \subseteq \delta(X_t) = \delta'(X_t)$. Consequently, $(\alpha, \beta) \in \delta'(X_t)$ yielding the validity of (2).

Now, let us suppose that decision-mapping $\delta^*[\{X_1, \dots, X_i, X_{i+1}\}]$ is an extension of $\delta[\{X_1, \dots, X_i\}]$. We show that $\delta'(X_t) = \delta^*(X_t)$ is valid for all $t$, $t = 1, \dots, i+1$. If $1 \leq t \leq i$, then the required equality obviously holds. Therefore, it is enough to prove that $\delta'(X_{i+1}) \subseteq \delta^*(X_{i+1})$ and $\delta'(X_{i+1}) \supseteq \delta^*(X_{i+1})$. To do so, let $(\alpha, \beta) \in \delta'(X_{i+1})$ be an arbitrary operating unit. By the definition of $\delta'$, $(\alpha, \beta) \in \delta(X_j) \cap \Delta(X_{i+1})$ for some $X_j \in \{X_1, \dots, X_i\}$. But $\delta(X_j) = \delta^*(X_j)$ and $\delta^*$ is consistent. Consequently,

$$(\alpha, \beta) \in \delta^*(X_j) \cap \Delta(X_{i+1}) \subseteq \delta^*(X_{i+1}).$$

Conversely, let $(\alpha, \beta) \in \delta^*(X_{i+1})$. Since $W = \emptyset$, there exists a $j \in \{1, \dots, i\}$ such that $(\alpha, \beta) \in \Delta(X_j)$, and thus, $(\alpha, \beta) \in \delta^*(X_{i+1}) \cap \Delta(X_j)$. Now, by the consistency of $\delta^*$, $\delta^*(X_{i+1}) \cap \Delta(X_j) \subseteq \delta^*(X_j) = \delta(X_j)$. Therefore, $(\alpha, \beta) \in \Delta(X_{i+1}) \cap \delta(X_j)$, but then, $(\alpha, \beta) \in \delta'(X_{i+1})$ from the definition of $\delta'$.

*Case* 2. $W \neq \emptyset$. Using the observations of Case 1 again, it is sufficient to prove that consistent decision-mapping $\delta[\{X_1, \dots, X_i\}]$ has $2^{|W|}$ extensions to $\{X_1, \dots, X_i, X_{i+1}\}$. For this purpose, let

$$T = \{(\alpha, \beta) : (\alpha, \beta) \in \Delta(X_{i+1})\ \&\ (\alpha, \beta) \in \delta(X_t) \text{ for some } t \in \{1, \dots, i\}\}.$$

From the definitions, $T \cap W = \emptyset$. Now, we show that decision-mapping $\delta'$ defined by

$$\delta'(X) = \begin{cases} \delta(X) & \text{if } X \in \{X_1, \dots, X_i\}, \\ T \cup Q & \text{if } X = X_{i+1}, \end{cases}$$

is consistent for every subset $Q$ of $W$. Since $\delta[\{X_1, \dots, X_i\}]$ is consistent, we have to prove that the following inclusions

(3) $\delta'(X_j) \cap \Delta(X_{i+1}) \subseteq \delta'(X_{i+1})$,

and

(4) $\quad \delta'(X_{i+1}) \cap \Delta(X_j) \subseteq \delta'(X_j),$

are valid for all $j$, $j = 1, \dots, i$.

To prove these inclusions, let $j \in \{1, \dots, i\}$ be an arbitrary index. First, let $(\alpha, \beta) \in \delta'(X_j) \cap \Delta(X_{i+1})$. Then, $(\alpha, \beta) \in T$, and thus, $(\alpha, \beta) \in \delta'(X_{i+1})$ resulting in (1). Now, let $(\alpha, \beta) \in \delta'(X_{i+1}) \cap \Delta(X_j)$. Then, $(\alpha, \beta) \in (T \cup Q) \cap \Delta(X_j) = T \cap \Delta(X_j)$. Inclusion $(\alpha, \beta) \in T$ implies that $(\alpha, \beta) \in \delta(X_t)$ for some $t \in \{1, \dots, i\}$. Consequently, $(\alpha, \beta) \in \delta(X_t) \cap \Delta(X_j)$. Since $\delta$ is consistent, $\delta(X_t) \cap \Delta(X_j) \subseteq \delta(X_j) = \delta'(X_j)$ which yields (4).

By the construction above, $2^{|W|}$ different extensions of $\delta[\{X_1, \dots, X_i\}]$ are presented. To complete the proof, it is shown that the decision-mapping under consideration has no further extensions to $\{X_1, \dots, X_i, X_{i+1}\}$. Indeed, let $\delta'[\{X_1, \dots, X_i, X_{i+1}\}]$ be an arbitrary extension of $\delta$ and $(\alpha, \beta) \in T$. Then, $(\alpha, \beta) \in \delta(X_t) \cap \Delta(X_{i+1}) = \delta'(X_t) \cap \Delta(X_{i+1})$ for some $t \in \{1, \dots, i\}$. Since $\delta'$ is consistent, $\delta'(X_t) \cap \Delta(X_{i+1}) \subseteq \delta'(X_{i+1})$, and thus, $(\alpha, \beta) \in \delta'(X_{i+1})$. Consequently, $T \subseteq \delta'(X_{i+1})$. On the other hand, $(\alpha, \beta) \in \delta'(X_{i+1}) \setminus T$ implies $(\alpha, \beta) \in W$. In the opposite case, $(\alpha, \beta) \in \Delta(X_t)$ for some $t \in \{1, \dots, i\}$, and then, $(\alpha, \beta) \in \delta'(X_t) = \delta(X_t)$ because of the consistency of $\delta'$ which is a contradiction. Then, $\delta'(X_{i+1}) \subseteq T \cup W$, and thus, $\delta'$ is equal to one of the given extensions of $\delta$.

Now, by the induction hypothesis, we obtain that the number of consistent decision-mappings defined on $\{X_1, \dots, X_{i+1}\}$ is

$$2^{|\cup\{\Delta(X_t):t=1,\dots,i\}|}2^{|W|} = 2^{|\cup\{\Delta(X_t):t=1,\dots,i+1\}|}$$

which completes the proof.

**Remark 1.** In particular, if $m = M \setminus R$, then from our Theorem it follows that the number of the maximal consistent decision-mappings is $2^{|O|}$. This shows that there is a strong relationship between the maximal consistent decision-mappings and the subsets of $O$. Indeed, it can be proved that mapping $\gamma$ defined by $\gamma(\delta) = op(\delta)$ is a one-to-one mapping of $\Omega'_{\mathrm{M}}$ onto $\wp(O)$ where $\wp(O)$ denotes the set of all subsets of $O$.

Regarding the relationship between the maximal decision-mappings and the feasible solutions, let us define mapping $\rho$ in the following way. For any $(m, o) \in S(\mathrm{M})$, let $\rho(m, o) = \delta$ where $\delta$ is defined by

$$\delta(X) = \{u : u = (\alpha, \beta) \in o \;\&\; X \in \beta\}$$

for all $X \in M \setminus R$. It can be easily proved that $\rho$ is a one-to-one mapping of $S(\mathrm{M})$ into $\Omega'_{\mathrm{M}}$. Therefore, $2^{|O|}$ is a trivial upper bound for $|S(\mathrm{M})|$. Taking into account property (A2), this bound can be improved as follows.

## 3 Bound calculation

Let $(m, o) \in S(\mathrm{M})$ be an arbitrary feasible solution and $\rho(m, o) = \delta$. Then, (A2) implies the following inclusion:

$$(ii) \qquad mat^{in}(op(\delta)) \subseteq mat^{out}(op(\delta)) \cup R.$$

Indeed, if $X \in mat^{in}(op(\delta))$, then there exists a $u = (\alpha, \beta) \in op(\delta)$ with $X \in \alpha$. By the definition of $\delta$, $u \in o$, and thus, $X \in m$ from the definition of the P-graph. Now, (A2) implies that $X \in mat^{out}(op(\delta)) \cup R$, i.e., inclusion (ii) must hold. Consequently, the number of the maximal consistent decision-mappings satisfying $(ii)$ is not less than $|S(\mathrm{M})|$.

Now, we are going to determine the number of the maximal consistent decision-mappings satisfying $(ii)$. It can be done by the Inclusion-Exclusion Formula. For this purpose, let us denote by $(M, O)$ the P-graph of PNS problem under consideration and let $O = \{u_1, \ldots, u_n\}$ and $M = \{X_1, \ldots, X_k\}$. Furthermore, let $O(X_j)$ denote the set, $\{u : u = (\alpha, \beta) \in O \ \& \ X_j \in \alpha\}$, for all $X_j \in M$. Let $j \in \{1, \ldots, k\}$ be an arbitrary index and let us define set $A_j$ by

$$A_j = \{\delta : \delta \in \Omega'_{\mathrm{M}} \ \& \ X_j \in mat^{in}(op(\delta)) \setminus (mat^{out}(op(\delta)) \cup R)\}.$$

Then, $(ii)$ is not satisfied by $\delta \in A_j$ and the reason is that $X_j \in mat^{in}(op(\delta))$ and $X_j \notin mat^{out}(op(\delta)) \cup R$. For every $\emptyset \neq I \subseteq \{1, \ldots, k\}$, let us define set $A_I$ by $A_I = \cap_{i \in I} A_i$, and in particular, let $A_\emptyset = \Omega'_{\mathrm{M}}$. Then, the required number is

$$|\Omega'_{\mathrm{M}} \setminus (A_1 \cup A_2 \cup \ldots \cup A_k)| = \Sigma_{I \subseteq \{1, \ldots, k\}} (-1)^{|I|} \cdot |A_I|.$$

Obviously, if $I = \{i_1, \ldots, i_l\}$, then

$$A_I = \{\delta : \delta \in \Omega'_{\mathrm{M}} \ \& \ \{X_{i_1}, \ldots, X_{i_l}\} \subseteq mat^{in}(op(\delta)) \setminus (mat^{out}(op(\delta)) \cup R)\}$$

**Remark 2.** It is worth noting that the bound presented above is independent of the set of the required products. It is valid under arbitrary $P \subseteq M \setminus R$.

Unfortunately, to count $|A_I|$ is a difficult problem. In general case, we have to cover $\{X_{i_1}, \ldots, X_{i_l}\}$ with such a system, $\alpha_{j_1}, \ldots, \alpha_{j_s}$ for which there are operating units $(\alpha_{j_t}, \beta_{j_t}) \in O$, $t = 1, \ldots, s$, with $\{X_{i_1}, \ldots, X_{i_l}\} \cap \beta_{j_t} = \emptyset$, $t = 1, \ldots, s$, and $|A_I|$ is equal to the number of the such covering systems. The determination of $|A_I|$ is easier if we restrict ourselves to special classes of PNS problems. An interesting special case is the class containing separator type operating units, i.e., $|\alpha| = 1$ is valid for all $u = (\alpha, \beta) \in O$. In what follows, we deal with this class.

Let us consider set $I = \{i_1, \ldots, i_l\}$ again. Let $O^*(X_{i_j}) = O(X_{i_j}) \setminus (\cup_{i \in I} \Delta\,(X_i))$. Then, $O^*(X_{i_j})$ is the set of operating units such that they do not produce any material from $\{X_t : t \in I\}$ and each of them has $X_{i_j}$ as input material. Now, it is easy to check that

$$|A_I| = \left( \prod_{t=1}^{l} \left( 2^{|O^*(X_{i_t})|} - 1 \right) \right) \cdot 2^{|O \setminus (\cup\{\Delta(X_i) : i \in I\}) \setminus (\cup\{O(X_i) : i \in I\})|}$$

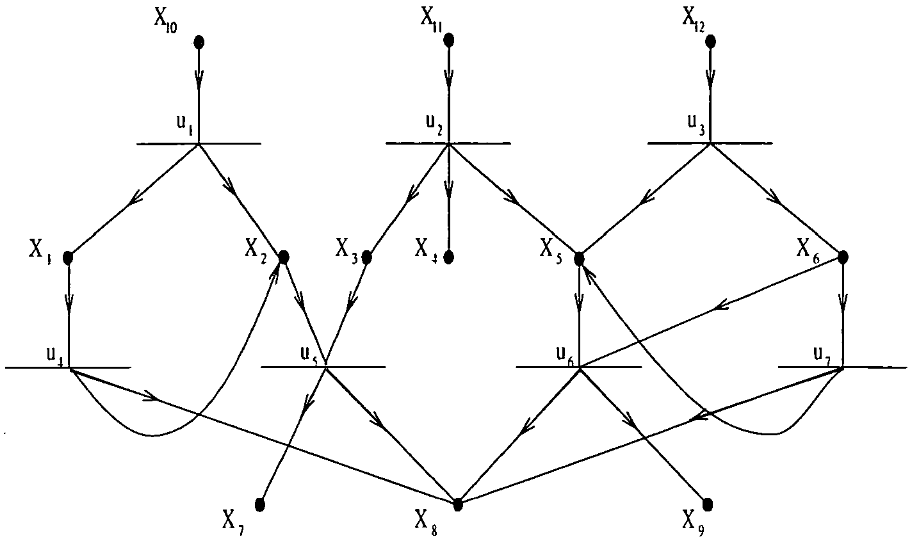

Figure 1: P-graph of the example.

## 4 Illustration

For illustrating the calculation of the bound in general case, let us consider the following example. Let $M = \{X_1, \ldots, X_{12}\}$, $O = \{u_1, \ldots, u_7\}$, $P = \{X_8\}$, and $R = \{X_{10}, X_{11}, X_{12}\}$. The input and output materials of the operating units are given in Table 1 and the corresponding P-graph is shown in Figure 1.

Table 1: Operating units

| unit | inputs | outputs |
|---|---|---|
| $u_1$ | $X_{10}$ | $X_1, X_2$ |
| $u_2$ | $X_{11}$ | $X_3, X_4, X_5$ |
| $u_3$ | $X_{12}$ | $X_5, X_6$ |
| $u_4$ | $X_1$ | $X_2, X_8$ |
| $u_5$ | $X_2, X_3$ | $X_7, X_8$ |
| $u_6$ | $X_5, X_6$ | $X_8, X_9$ |
| $u_7$ | $X_6$ | $X_5, X_8$ |

Using the relationship between the maximal consistent decision-mappings and the subsets of $O$ provided by Remark 1, set $A_1$ contains $\delta$ if and only if $op(\delta)$ satisfies the following properties: $u_1 \notin op(\delta)$ and $u_4 \in op(\delta)$. The number of such maximal consistent decision-mappings is $2^5$. Therefore, $|A_1| = 2^5$. In a similar way, we obtain that $|A_2| = 2^4$, $|A_3| = 2^5$, $|A_5| = 2^3$, $|A_6| = 3 \cdot 2^4$, and $|A_j| = 0$ for the remaining indices. Consequently,

$$\sum_{I \subseteq \{1,\dots,k\} \ \& \ |I|=1} |A_I| = 136.$$

Regarding the subsets of two elements, $A_{\{1,2\}}$ contains $\delta$ if and only if $u_1, u_4 \notin op(\delta)$ and $u_4, u_5 \in op(\delta)$, and thus, $A_{\{1,2\}} = \emptyset$. Similarly, $A_{\{1,3\}} = 2^3$ since $A_{\{1,3\}}$ contains $\delta$ if and only if $u_1, u_2 \notin op(\delta)$ and $u_4, u_5 \in op(\delta)$. Determining the corresponding values for the all subsets of two elements and summarizing, we obtain that

$$\sum_{I \subseteq \{1,\dots,k\} \ \& \ |I|=2} |A_I| = 60.$$

Continuing the procedure, we obtain 12 for the subsets of three elements. Finally, it can be seen that $|A_I| = 0$ if $|I| > 3$. Consequently, the required number is

$$2^7 - 136 + 60 - 12 = 40.$$

We note that $|\Omega'_{\mathrm{M}}| = 128$ and $|S(\mathrm{M})| = 19$ in this example.

## References

[1] Z. Blázsik and B. Imreh, A note on connection between PNS and set covering problems, *Acta Cybernetica* **12** (1996), 309-312.

[2] C. A. Floudas and I. E. Grossmann, Algorithmic Approaches to Process Synthesis: Logic and Global Optimization, AiChE Symposium Series No. 304, **91** (Eds: L. T. Biegler and M. F. Doherly), (1995), 198-221. .

[3] F. Friedler, L. T. Fan and B. Imreh, Process Network Synthesis: Problem Definition, *Networks*, to appear.

[4] F. Friedler, K. Tarján, Y. W. Huang and L. T. Fan, Combinatorial Structure of Process Network Synthesis, Sixth SIAM Conference on Discrete Mathematics, Vancouver, Canada, 1992.

[5] F. Friedler, K. Tarján, Y.W. Huang and L.T. Fan, Graph-Theoretic Approach to Process Synthesis: Axioms and Theorems, *Chem. Eng. Sci.*, **47**(8) (1992), 1973-1988.

[6] F. Friedler, J. B. Varga and L. T. Fan, Decision-mappings: a tool for consistent and complete decisions in process synthesis, *Chem. Eng. Sci.*, **50**(11) (1995), 1755-1768.

[7] F. Friedler, J. B. Varga, E. Fehér and L. T. Fan, Combinatorially Accerelated Branch-and -Bound Method for Solving the MIP Model of Process Network Synthesis, presented at the International Conference on State of the Art in Global Optimization: Computational Methods and Applications, Princeton University, Princeton, NJ, U.S.A., April 28-30, 1995; also to be published in Nonconvex Optimization and its Applications, Kluwer Academic Publishers, Norwell, MA, U.S.A. (in press).

[8] I. E. Grossmann, V. T. Voudouris and O. Ghattas, Mixed-Integer Linear Programming Reformulations for Some Nonlinear Discrete Design Optimization Problems, In: Recent Advances in Global Optimization (Eds: C. A. Floudas and P. M. Pardalos) Princeton University Press, New Jersey, 1992.

[9] B. Imreh, F. Friedler and L. T. Fan, An Algorithm for Improving the Bounding Procedure in Solving Process Network Synthesis by a Branch-and-Bound Method, Developments in Global Optimization (Eds: I. M. Bonze, T. Csendes, R. Horst, P. M. Pardalos), Kluwer Academic Publishers, 1997, 315-348.

*Received August, 1997*

# On the reformulation of some classes of PNS-problems as set covering problems

J. Fülöp [*] B. Imreh [†] F. Friedler [‡]

**Abstract**

Process network synthesis (PNS) has enormous practical impact; however, its solution is difficult in general. This experience has been recently reasoned by Blázsik and Imreh who pointed out that PNS-problems are NP-hard. They proved that a simple subclass of PNS-problems is equivalent to the class of set covering problems. In the present paper, it is shown that more general classes of PNS-problems can also be reformulated as set covering problems. This enables the sophisticated techniques developed for solving set covering problems also to be applied for solving some PNS-problems.

## 1 Introduction

The importance of process network synthesis (PNS) and the background of the combinatorial model studied here can be found in [5], [6], [7], [8], [9], and in the work [2] of this journal. Therefore, we shall confine ourselves only to the recall of the definitions. The combinatorial approach makes possible to show that the search of an optimal solution is difficult in general. This experience has been recently reasoned by Blázsik and Imreh [2] who pointed out that PNS-problems with weights are NP-hard. They proved that a simple subclass of PNS-problems with weights, to be discussed in Section 4, is equivalent to the class of set covering problems. Also in [2], it was raised as an open problem if there exist equivalent known optimization problems for more general classes of PNS.

In this paper, it is shown that the optimal solutions for a larger subclass of PNS-problems than the subclass presented in [2] as well as the optimal solutions of PNS-problems with nonnegative weights can be obtained by solving suitably constructed set covering problems. This enables the sophisticated techniques developed for solving set covering problems (see, e.g., [1, 4, 10] and the references therein) also to be applied for solving these special classes of PNS-problems with weights. To

[*]Laboratory of Operations Research and Decision Systems, Computer and Automation Institute, Hungarian Academy of Sciences, H-1518 Budapest, P.O.Box 63, Hungary

[†]Department of Informatics, József Attila University, Árpád tér 2, H-6720 Szeged, Hungary

[‡]Department of Computer Science, University of Veszprém, Egyetem u. 10, H-8200 Veszprém, Hungary

present our results, first we discuss the conjunctive normal form (CNF) proposed in [3] for describing the solution-structures of PNS-problems in Section 3. Some special classes of PNS-problems with weights, and the connection between the optimal solutions of these PNS and CNF-problems with weights are detailed in Section 4. The reformulation of a CNF with weights as a set covering problem is presented in Section 5.

## 2 Notions and notations

In the combinatorial approach, the structure of a process can be described by the process-graph (see [7] and [8]) defined as follows.

Let $M$ be a finite nonempty set, the set of the materials. Furthermore, let $\emptyset \neq O \subseteq \wp'(M) \times \wp'(M)$ with $M \bigcap O = \emptyset$ where $\wp'(M)$ denotes the set of all nonempty subsets of $M$. The elements of $O$ are called *operating units* and for an operating unit $(\alpha, \beta) \in O$, $\alpha$ and $\beta$ are called the *input-set* and *output-set* of the operating unit, respectively. Pair $(M, O)$ is defined to be a *process graph.* The set of vertices of this directed graph is $M \bigcup O$, and the set of arcs is $A = A_1 \cup A_2$ where $A_1 = \{(X, Y) : Y = (\alpha, \beta) \in O \text{ and } X \in \alpha\}$ and $A_2 = \{(Y, X) : Y = (\alpha, \beta) \in O \text{ and } X \in \beta\}$. If there exist vertices $X_1, X_2, ..., X_n$, such that $(X_1, X_2), (X_2, X_3), \ldots, (X_{n-1}, X_n)$ are arcs of process graph $(M, O)$, then the path determined by these arcs is denoted by $[X_1, X_n]$.

Let process graphs $(m, o)$ and $(M, O)$ be given. $(m, o)$ is defined to be a *subgraph* of $(M, O)$, if $m \subseteq M$ and $o \subseteq O$.

Now, we can define the structural model of PNS for studying the problem in structural point of view. For this reason, let $M^*$ be an arbitrarily fixed infinite set, the set of the available materials. By *structural model* of PNS, we mean a triplet $(P, R, O)$ where $P$, $R$, $O$ are finite sets, $\emptyset \neq P \subseteq M^*$ is the set of the *desired products*, $R \subseteq M^*$ is the set of the *raw materials*, and $O \subseteq \wp'(M^*) \times \wp'(M^*)$ is the set of the available operating units. It is assumed that $P \bigcap R = \emptyset$ and $M^* \bigcap O = \emptyset$.

Then, process graph $(M, O)$, where $M = \bigcup\{\alpha \bigcup \beta : (\alpha, \beta) \in O\}$, presents the interconnections among the operating units of $O$. Furthermore, every feasible process, producing the given set $P$ of products from the given set $R$ of raw materials using operating units from $O$, corresponds to a subgraph of $(M, O)$. Examining the corresponding subgraphs of $(M, O)$, therefore, we can determine an optimal process in principle. If we do not consider further constraints such as material balance, then the subgraphs of $(M, O)$ which can be assigned to a feasible process have common combinatorial properties. They are studied in [7] and their description is given by the following definition.

Subgraph $(m, o)$ of $(M, O)$ is called a *solution-structure* of $(P, R, O)$ if the following properties are satisfied:

$(S1)$ $P \subseteq m$,

$(S2)$ $\forall X \in m$, $X \in R \Leftrightarrow$ no $(Y, X)$ arc in the process graph $(m, o)$,

($S3$) $\forall Y_0 \in o$, $\exists$ path $[Y_0, Y_n]$ with $Y_n \in P$,

($S4$) $\forall X \in m$, $\exists(\alpha, \beta) \in o$ such that $X \in \alpha \bigcup \beta$.

Let us denote the set of solution-structures of $(P, R, O)$ by $S(P, R, O)$. In the sequel, we shall assume that $S(P, R, O) \neq \emptyset$. This can be checked in polynomial time by using the algorithm presented in [9] for generating the maximal structure of $(P, R, O)$.

Let the set of the operating units be given by $O = \{(\alpha_1, \beta_1), \ldots, (\alpha_l, \beta_l)\}$, and let $I = \{1, \ldots, l\}$. Then, for any subgraph $(m, o)$ of $(M, O)$, an $l$-vector of logical values $u_i$, $i \in I$, can be associated with such that $u_i$ is true if and only if $(\alpha_i, \beta_i) \in o$. It is easy to see that this is a one-to-one mapping between the subgraphs of $(M, O)$ fulfilling ($S4$) and the $l$-vectors of logical values. For logical $l$-vector $u$, subgraph $(m, o)$ associated with $u$ is determined by $m = \bigcup_{i \in T(u)} \alpha_i \cup \beta_i$ and $o = \{(\alpha_i, \beta_i) : i \in T(u)\}$ where $T(u) = \{i \in I : u_i \text{ is true}\}$.

## 3 CNF related to PNS

In [3], a logical expression given in CNF ($A1$)-($A4$) below was used to describe some structures of $(M, O)$.

($A1$) $\displaystyle\bigwedge_{X \in P} \bigvee_{\substack{i \in I \\ X \in \beta_i}} u_i,$

($A2$) $\displaystyle\bigwedge_{\substack{i \in I \\ R \cap \beta_i \neq \emptyset}} \neg u_i,$

($A3$) $\displaystyle\bigwedge_{\substack{i \in I \\ X \in \alpha_i \setminus R}} (\neg u_i \vee \bigvee_{\substack{h \in I \\ X \in \beta_h}} u_h),$

($A4$) $\displaystyle\bigwedge_{\substack{i \in I \\ P \cap \beta_i = \emptyset}} (\neg u_i \vee \bigvee_{\substack{h \in I \\ \beta_i \cap \alpha_h \neq \emptyset}} u_h).$

In this section, the relationship between ($S1$)-($S4$) and ($A1$)-($A4$) will be discussed.

**Proposition 1.** *For any solution-structure $(m, o)$, the logical vector, $u$, associated with $(m, o)$ fulfills ($A1$)-($A4$).*

*Proof.* Let $u$ be the logical vector associated with solution-structure $(m, o)$. From ($S1$)-($S2$) and $P \cap R = \emptyset$, we obtain that any $X \in P$ is in the output-set of an operating unit of $(m, o)$. This gives ($A1$). ($A2$) follows directly from ($S2$).

Concerning ($A3$), we have to show that if $u_i$ is true for some $i \in I$ and $X \in \alpha_i \setminus R$, then there exists an $h \in I$ such that $u_h$ is true and $X \in \beta_h$, i.e., $X$ is in the output-set of an operating unit of $(m, o)$. This follows however immediatly from ($S2$).

To prove ($A4$), it is sufficient to consider the case when $u_i$ is true and $P \cap \beta_i = \emptyset$. From ($S3$) we get that there exists a path in $(m, o)$ from $(\alpha_i, \beta_i)$ to an element of $P$.

Since $P \cap \beta_i = \emptyset$, the vertex second to $(\alpha_i, \beta_i)$ in the path is an $(\alpha_{h'}, \beta_{h'})$, $h' \in I$, such that $\beta_i \cap \alpha_{h'} \neq \emptyset$. This implies $(A4)$ immediately. □

**Proposition 2.** *For any logical vector u fulfilling* $(A1)$-$(A4)$, *the subgraph,* $(m, o)$, *associated with u satisfies* $(S1)$, $(S2)$, *and* $(S4)$.

*Proof.* $(A1)$ states that for every $X \in P$, there exists an $i \in I$ such that $u_i$ is true and $X \in \beta_i$. This gives $X \in m$, and thus, $(S1)$ holds.

To prove $(S2)$, consider an $X \in m \cap R$. From $(A2)$ we get that $u_i$ is false for every $i \in I$ with $X \in \beta_i$. The way of construction of $(m, o)$ from $u$ implies that there exists no $(Y, X)$ arc in $(m, o)$.

Conversely, consider an $X \in m \setminus R$. We show that there exists an arc $(Y, X)$ in $(m, o)$, i.e., $X$ is in the output set of an operating unit associated with a true component of $u$. Since $X \in m$, there exists an $i \in I$ such that $u_i$ is true and $X \in \alpha_i \cup \beta_i$. If $X \in \beta_i$, we are done. Otherwise, $X \in \alpha_i \setminus R$ and $(A3)$ implies that there exists an $h \in I$ such that $u_h$ is true and $X \in \beta_h$.

Finally, $(S4)$ follows from the way of construction of $(m, o)$ from $u$. □

It is worth noting that $(A1)$-$(A4)$ does not imply $(S3)$. Namely, considering a general process graph, $(M, O)$, there may exist an operating unit $Y_0 \in o$ in subgraph $(m, o)$ constructed from $u$ fulfilling $(A1)$-$(A4)$ such that there is no path from $Y_0$ to any element of $P$. However, for special PNS-problems, $(S3)$ is also implied by $(A1)$-$(A4)$, thus, $(S1)$-$(S4)$ and $(A1)$-$(A4)$ are equivalent.

**Proposition 3.** *If process graph* $(M, O)$ *does not contain circuit, then* $(S1)$-$(S4)$ *and* $(A1)$-$(A4)$ *are equivalent .*

*Proof.* By Propositions 1 and 2, it is sufficient to show that $(A4)$ implies $(S3)$ in this case. Consider a $Y_{i_0} = (\alpha_{i_0}, \beta_{i_0}) \in o$. If $P \cap \beta_{i_0} \neq \emptyset$, we can construct a path from $Y_{i_0}$ to an element of $P \cap \beta_{i_0}$. Otherwise, by $(A4)$, there exists another operating unit $Y_{i_1} = (\alpha_{i_1}, \beta_{i_1})$ such that $Y_{i_1} \in o$ and $\beta_{i_0} \cap \alpha_{i_1} \neq \emptyset$. We have now path $[Y_{i_0}, Y_{i_1}]$ in $(m, o)$, and we can repeat the investigation above now for $Y_{i_1}$ instead of $Y_{i_0}$.

In a general step, we have operating unit $Y_{i_k} = (\alpha_{i_k}, \beta_{i_k})$ and path $[Y_{i_0}, Y_{i_k}]$ in $(m, o)$. If $P \cap \beta_{i_k} \neq \emptyset$, we are ready. Otherwise, we can extend the path from $Y_{i_k}$. Since $(M, O)$ contains no circuit, every vertex of the path is different. However, $(M, O)$ is finite, thus, after constructing a finite number of arcs, the path has to terminate in an element of $P$. □

Assume that in process graph $(M, O)$ of a PNS- problem, with a suitable positive integer $k$, we have $M = M_1 \cup \ldots \cup M_{k+1}$ where the sets, $M_1, \ldots, M_{k+1}$, are pairwise disjoint nonempty sets. Furthermore, let $O = O_1 \cup \ldots \cup O_k$ with $O_i \subseteq \wp'(M_1 \cup \ldots \cup M_i) \times \wp'(M_{i+1})$, $i = 1, \ldots, k$. Let us call such a PNS-problem a PNS$_k$-*problem*. Then, it is easy to see that for any PNS$_k$-problem, there exists no circuit in its process graph, and consequently, we have the following corollary.

**Corollary 1.** $(S1)$-$(S4)$ *and* $(A1)$-$(A4)$ *are equivalent for PNS$_k$-problems.*

## 4 PNS-problems with weights

Let us consider PNS-problems in which each operating unit has a weight. We are to find a feasible process with the minimal weight where by weight of a process we mean the sum of the weights of the operating units belonging to the process under consideration. Every feasible process in such a class of PNS-problems is determined uniquely from the corresponding solution-structure and vice versa. Therefore, the above problem can be formalized in the following way.

Let a structural model of PNS-problem $(P, R, O)$ be given. Moreover, let $w$ be a real-valued function defined on $O$, the weight function. The basic model is then

$$\min \{\sum_{U \in o} w(U) : (m, o) \in S(P, R, O)\}. \tag{1}$$

We refer (1) as a PNS$_w$-*problem*; we denote the class of such problems by PNS$_w$. PNS$_k$-problems with weights are referred as PNS$_{wk}$-problems, their subclass is denoted by PNS$_{wk}$. These latter problems were introduced, and the connection between PNS$_{w1}$-problems and set covering problems was also discussed in [2].

The feasible set of the optimization problem (1) is the set of the subgraphs $(m, o)$ fulfilling $(S1)$-$(S4)$. According to the discussion of the relation between $(S1)$-$(S4)$ and $(A1)$-$(A4)$, another optimization problem based on the CNF $(A1)$-$(A4)$ can also be considered:

$$\min \{ \sum_{i \in T(u)} w_i : u \text{ fulfills } (A1) - (A4)\} \tag{2}$$

where $w_i = w((\alpha_i, \beta_i)), i \in I$. We refer (2) as a CNF$_w$-problem associated with PNS$_w$- problem (1), and denote the class of such problems by CNF$_w$.

By Propositions 1 and 2, CNF$_w$ can be considered as a relaxation of PNS$_w$. This gives rise to the following statements.

**Proposition 4.** *Both* (1) *and* (2) *have finite optimal value. The optimal value of* (1) *is greater than or equal to that of* (2). *Furthermore, if* $(S3)$ *holds in the subgraph* $(m^*, o^*)$ *associated with an optimal solution of* (2), *then* $(m^*, o^*)$ *is an optimal solution of* (1).

In the case of PNS$_k$-problems, the equivalence between $(S1)$-$(S4)$ and $(A1)$-$(A4)$ implies a similar equivalence between the relating problems of PNS$_{wk}$ and CNF$_{wk}$.

**Corollary 2.** *Consider problems* (1) *and* (2) *generated by a PNS$_k$-problem. Then, the subgraph,* $(m^*, o^*)$, *associated with an optimal solution of* (2) *is optimal to* (1), *and conversely, the l-vector of logical values associated with an optimal solution of* (1) *is an optimal solution to* (2).

The following statements relate to special subclasses of PNS$_w$.

**Proposition 5.** *If the weights, $w_i$, $i \in I$, are positive, then subgraph $(m^*, o^*)$ associated with an optimal solution of* (2) *is optimal to* (1), *and conversely, the* $I$*-vector of logical values associated with an optimal solution of* (1) *is an optimal solution to* (2).

*Proof.* Let $u^*$ be an optimal solution of (2), and let $(m^*, o^*)$ be the subgraph associated with $u^*$. By Proposition 4, it is sufficient to show that $(S3)$ holds for $(m^*, o^*)$. Let

$$\bar{o} = \{U \in o^* : \exists \text{ path in } (m^*, o^*) \text{ from } U \text{ to a } Y \in P\}, \tag{3}$$

$$\bar{m} = \sum_{\substack{(\alpha_i,\beta_i)\in \bar{o} \\ i \in I}} \alpha_i \cup \beta_i. \tag{4}$$

Clearly, $(\bar{m}, \bar{o})$ is a subgraph of $(m^*, o^*)$. If $\bar{o} = o^*$, we are done. Otherwise, we shall show below that $(\bar{m}, \bar{o})$ is a solution-structure of $(P, R, O)$. Then, the logical vector, $\bar{u}$, associated with $(\bar{m}, \bar{o})$ is feasible to (2). However, since $w(U) > 0$ for every $U \in o^* \setminus \bar{o}$, the objective function value of $\bar{u}$ is less than that of $u^*$, and this contradicts the optimality of $u^*$ in (2). Consequently, $o^* = \bar{o}$ must hold.

We show now that $(S1)$-$(S4)$ holds for $(\bar{m}, \bar{o})$. By Proposition 2, $(m^*, o^*)$ fulfills $(S1)$, $(S2)$, and $(S4)$. Thus, from (3)-(4), we get immediately that $(S1)$, $(S3)$, and $(S4)$ hold for $(\bar{m}, \bar{o})$.

To prove $(S2)$ for $(\bar{m}, \bar{o})$, consider an $X \in \bar{m} \cap R$. Since there exists no $(Y, X)$ arc in $(m^*, o^*)$, and $(\bar{m}, \bar{o})$ is a subgraph of $(m^*, o^*)$, there exists $(Y, X)$ arc neither in $(\bar{m}, \bar{o})$. Conversely, consider an $X \in \bar{m} \setminus R$. In $(m^*, o^*)$, there exists a $(Y, X)$ arc. In addition, since $X \in \alpha \cup \beta$ for an $(\alpha, \beta) \in \bar{o}$, there exists a path in $(\bar{m}, \bar{o})$ from $(\alpha, \beta)$ to an element of $P$, thus, also from $Y$ to the same element of $P$. Therefore, $Y \in \bar{o}$ and $(Y, X)$ is an arc in $(\bar{m}, \bar{o})$.

The second part of the statement can be easily proved by using Proposition 1 and the first part of the statement. □

**Proposition 6.** *If the weights, $w_i$, $i \in I$, are nonnegative, then subgraph $(\bar{m}, \bar{o})$ defined by* (3)-(4) *for $(m^*, o^*)$ associated with an optimal solution of* (2) *is optimal to* (1).

*Proof.* According to the proof of Proposition 5, $(\bar{m}, \bar{o})$ is feasible to (1). It may happen now that $o^* \setminus \bar{o} \neq \emptyset$ but from the nonnegativity of the weights and using the same reasoning as in the proof of Proposition 5, we obtain that $w(U) = 0$ for every $U \in o^* \setminus \bar{o}$. The objective function values of $(\bar{m}, \bar{o})$ and $(m^*, o^*)$ coincide in (1). Therefore, $(\bar{m}, \bar{o})$ is optimal to (1). □

The set $\bar{o}$ defined in (3) can easily be generated by using the classical labeling technique of graph theory [13]. A similar technique is used also in [9] for generating the maximal structure of a process graph. It can be shown that $(\bar{m}, \bar{o})$ is the union of all solution-structure subgraphs of $(m^*, o^*)$. See [9] for more details.

## 5 Reformulation of a CNF with weights as a set covering problem

By the results presented in the previous section, the optimal solution of some important classes of PNS$_w$-problems, such as problems with nonnegative weights and PNS$_{wk}$-problems, can be obtained by solving the appropriate CNF$_w$- problems of form (2). However, it has not been discussed yet how to solve (2). In this section, we show that (2) can be transcribed into the form of an equivalent set covering problem. This can also be considered as an extension of the results presented in [2] for PNS$_{w1}$-problems.

For every $u_i, i \in I$, we introduce two 0-1 variables, $z_i^+$ and $z_i^-$, such that $z_i^+ = 1$ if and only if $u_i$ is true, and $z_i^- = 1 - z_i^+$. Then, at the expense of doubling the number of variables and introducing some appropriate new constraints, (2) can be written into the equivalent form

$$\min \quad \sum_{i \in I} w_i z_i^+, \tag{5}$$

$$\sum_{\substack{i \in I \\ X \in \beta_i}} z_i^+ \geq 1 \quad \text{for all } X \in P, \tag{6}$$

$$z_i^- = 1 \qquad \text{for all } i \in I, R \cap \beta_i \neq \emptyset, \tag{7}$$

$$z_i^- + \sum_{\substack{h \in I \\ X \in \beta_h}} z_h^+ \geq 1 \qquad \text{for all } i \in I, X \in \alpha_i \setminus R, \tag{8}$$

$$z_i^- + \sum_{\substack{h \in I \\ \beta_i \cap \alpha_h \neq \emptyset}} z_h^+ \geq 1 \quad \text{for all } i \in I, P \cap \beta_i = \emptyset, \tag{9}$$

$$z_i^+ + z_i^- = 1 \qquad \text{for all } i \in I, \tag{10}$$

$$z_i^+, z_i^- \in \{0, 1\} \quad \text{for all } i \in I. \tag{11}$$

In (5)-(11), (5) and (6)-(9) are the direct transcription of the objective function in (2) and the constraints ($A1$)-($A4$), respectively. Constraints (10)-(11) describe the relation among $u_i$, $z_i^+$ and $z_i^-$. Since we have assumed that $S(P, R, O) \neq \emptyset$, problems (1), (2), hence (5)-(11), too, have feasible solution and finite optimal value.

Problem (5)-(11) is a set covering/partitioning problem for which efficient solution methods have been developed, see [4] and the references therein. Constraint (7) means to fix $z_i^- = 1$ and $z_i^+ = 0$ for all $i \in I$, $R \cap \beta_i \neq \emptyset$, and these can entail the possible fixation of further variables and the deletion of some constraints [1, 4, 10].

By using the well-known trick of converting set partitioning constraints into set covering ones (cf. e.g. [10]), we obtain the following statement.

**Proposition 7.** *Choose any $L > \sum_{i \in I} w_i$, and consider the set covering problem*

$$\min \quad \sum_{i \in I} \left[ (w_i + L) z_i^+ + L z_i^- \right], \tag{12}$$

$$\sum_{\substack{i\in I\\ X\in\beta_i}} z_i^+ \geq 1 \quad \text{for all } X \in P, \tag{13}$$

$$z_i^- \geq 1 \qquad \text{for all } i \in I, R\cap\beta_i \neq \emptyset, \tag{14}$$

$$z_i^- + \sum_{\substack{h\in I\\ X\in\beta_h}} z_h^+ \geq 1 \qquad \text{for all } i \in I, X \in \alpha_i \setminus R, \tag{15}$$

$$z_i^- + \sum_{\substack{h\in I\\ \beta_i\cap\alpha_h\neq\emptyset}} z_h^+ \geq 1 \quad \text{for all } i \in I, P\cap\beta_i = \emptyset, \tag{16}$$

$$z_i^+ + z_i^- \geq 1 \qquad \text{for all } i \in I, \tag{17}$$

$$z_i^+, z_i^- \in \{0,1\} \quad \text{for all } i \in I. \tag{18}$$

*Then, problems* (5)-(11) *and* (12)-(18) *have the same set of optimal solutions.*

*Proof.* It is easy to see that any feasible solution of (5)-(11) is feasible to (12)-(18) as well, and the difference of the two objective function values is the constant, $lL$. As a consequence, since the optimal value of (5)-(11) is less than $L$, the optimal value of (12)-(18) is less than $(l+1)L$.

Conversely, consider a feasible solution of (12)-(18) and assume that it is not feasible to (5)-(11). Then, its objective function value in (12) is greater than or equal to $(l+1)L$. Thus, any optimal solution of (12)-(18) is feasible to (5)-(11), and this implies the statement. □

In set covering problem (12)-(18), as well, constraint (14) entails the possible reduction of the problem size. For further size reduction techniques and for recent sophisticated methods for solving set covering problems, see [1, 4] and the references therein.

## References

[1] E. Balas and M. C. Carrera, A Dynamic Subgradient- Based Branch-and-Bound Procedure for Set Covering, *Operations Research*, **44**(6), 1996, 875-890.

[2] Z. Blázsik and B. Imreh, A Note on Connection between PNS and Set Covering Problems, *Acta Cybernetica*, **12**, 1996, 309-312.

[3] M. H. Brendel, F. Friedler and L.T. Fan, Combinatorial Foundation for Logical Formulation in Total Flowsheet Synthesis, *Computers chem. Engng.* (submitted).

[4] M. L. Fisher and P. Kedia, Optimal Solution of Set Covering/Partitioning Problems Using Dual Heuristics, *Management Science*, **36**, 1990, 674-688.

[5] C. A. Floudas and I. E. Grossmann, Algorithmic Approaches to Process Synthesis: Logic and Global Optimization, International Symposium on Foundations of Computer Aided Process Design, Snowmass Village, CO, U.S.A., July 10-15, 1994 (in press).

[6] F. Friedler, L. T. Fan and B. Imreh, Process Network Synthesis: Problem Definition, *Networks* (to appear).

[7] F. Friedler, K. Tarjan, Y. W. Huang and L. T. Fan, Graph-Theoretic Approach to Process Synthesis: Axioms and Theorems, *Chem. Eng. Sci.*, **47**(8), 1992, 1973-1988.

[8] F. Friedler, K. Tarjan, Y. W. Huang and L. T. Fan, Combinatorial Structure of Process Network Synthesis, Sixth SIAM Conference on Discrete Mathematics, Vancouver, Canada, 1992.

[9] F. Friedler, K. Tarjan, Y. W. Huang and L. T. Fan, Graph-Theoretic Approach to Process Synthesis: Polynomial Algorithm for Maximal Structure Generation, *Computers chem. Engng.*, **17**(9), 1993, 929-942.

[10] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*, Wiley, New York, 1972.

[11] I. E. Grossmann, V. T. Voudouris and O. Ghattas, Mixed-Integer Linear Programming Reformulations for Some Nonlinear Discrete Design Optimization Problems, In: *Recent Advances in Global Optimization* (Eds: C. A. Floudas and P. M. Pardalos) Princeton University Press, New Jersey, 1992.

[12] B. Imreh, F. Friedler and L. T. Fan, An Algorithm for Improving the Bounding Procedure in Solving Process Network Synthesis by a Branch-and- Bound Method, in: *Developments in Global Optimization* (Eds: I. M. Bonze, T. Csendes, R. Horst, P. M. Pardalos), Kluwer Academic Publishers, 1997, 315-348.

[13] E. Lawler, *Combinatorial Optimization*, Holt, Rinehart and Winston, New York, 1976.

*Received August, 1997*

CONTENTS

**Sponsored by SYSDATA Ltd.**

**ISSN 0324—721 X**

**Felelős szerkesztő és kiadó: Csirik János**
**A kézirat a nyomdába érkezett: 1998. július**
**Terjedelem: 7,12 (B/5) ív**