Volume 12 Number 1

ACTA CYBERNETICA

Szeged, 1995

Acta Cybernetica, Vol. 12, No. 1, Szeged, 1995

Generalized DOL trees*

Lila Kari† Grzegorz Rozenberg‡ Arto Salomaa §

Abstract

Infinite unlabeled trees having a finite number of different subtrees (also called infinite regular trees) arise in a natural way from a DOL system which also gives a natural labeling for the tree. A much more compact representation for the tree often results from a DOL system with fragmentation.

Keywords: formal languages, DOL system, fragmentation, tree labeling.

1 Introduction

One of the simplest, if not the simplest, models extensively investigated in the theory of computing is the *DOL system*. By definition, a *DOL system* is a triple $G = (\Sigma, h, w)$, where Σ is a finite alphabet, $h : \Sigma^* \longrightarrow \Sigma^*$ is a morphism, and $w \in \Sigma^*$ is a word (usually called the *axiom*). The DOL system G generates the sequence $S(G)$ of words $w_0, w_1, w_2, \ldots$, where

$$w_0 = w \text{ and } w_i = h^i(w) = h(w_{i-1}) \text{ for } i \geq 1.$$

Thus, $S(G)$ is obtained from the axiom by iterating the morphism. (Our exposition is largely self-contained. If need arises, [3] can be consulted. [1] and [4] are some of the recent papers concerning DOL systems.)

As an example, consider the DOL system with the alphabet $\Sigma = \{a, b\}$, axiom $w = a$ and the morphism h defined by the rules

$$a \longrightarrow b, \quad b \longrightarrow ab.$$

This is the well-known "Fibonacci system", where the lengths of the words in the generated sequence

$$a, b, ab, bab, abbab, babbabb, \ldots$$

form the sequence of Fibonacci numbers. The following tree, labeled by the letters of Σ, depicts the generation process:

*Research partially supported by the Academy of Finland, project 11281. All correspondence to Lila Kari.

†Department of Mathematics, University of Western Ontario, London, Ontario, N6A 5B7 Canada

‡Department of Computer Science, Leiden University, 2300 RA Leiden, The Netherlands

§Academy of Finland and Department of Mathematics, University of Turku, 20 500 Turku, Finland

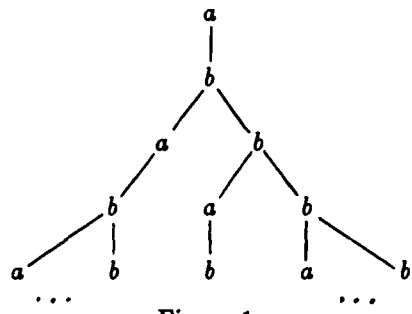

Figure 1.

For the DOL systems G considered in this paper, we assume that $|w| = 1$ (that is, the axiom is a letter) and that h is *nonerasing* (that is, we are dealing with *PDOL systems*). These assumptions guarantee that the sequence $S(G)$ can always be represented as an infinite tree labeled by letters of Σ, where all branches continue ad infinitum.

Remark. If we allow $|w| > 1$, we are dealing with forests instead of trees. An additional letter used only as the axiom brings us back to trees. Moreover, our main result remains valid for general DOL systems as well. Consequently, our assumptions do not exclude any interesting cases.

Infinite (labeled) trees obtained in the way described above are referred to as *DOL trees*. The formal definition of a DOL tree should be clear and is omitted here. It is also clear that if you begin with an infinite unlabeled tree that possesses only finitely many different subtrees (such trees are often referred to as *regular*), then you can label it with finitely many labels and view the result as a DOL tree. The labels constitute the alphabet of the corresponding DOL system.

The arity of each letter is the length of the right side of the rule for the letter.

Regular trees play a central role in the theory of automata, nonrecursive program schemes, etc. Such matters are of no direct concern to us in this paper. For the sake of later reference, we summarize the above discussion in the following lemma. Thus, an infinite unlabeled tree is *regular* if it possesses only finitely many different subtrees. The *unlabeled version* of a DOL tree is obtained from a DOL tree by removing the labels.

Lemma 1.1 *The unlabeled version of a DOL tree is regular. Conversely, every regular tree can be labeled to become a DOL tree.*

If we do not make the convention above (to the effect that all branches of the trees continue ad infinitum), then the DOL systems should containg also erasing rules.

As a further example, consider the tree

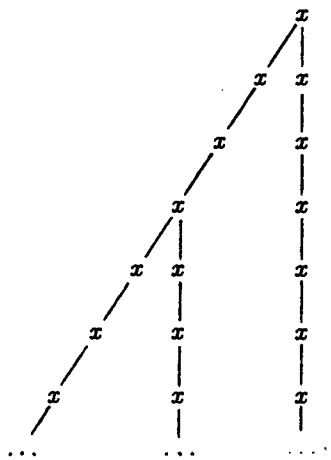

Figure 2.

Thus, a new branch is born at every third node of the stem. Clearly, the DOL system with the axiom a and the rules

$$a \longrightarrow bd, \quad b \longrightarrow c, \quad c \longrightarrow a, \quad d \longrightarrow d$$

provides the labeling.

Let us modify the example in such a way that the new branches are born at nodes whose distance from the root is a prime number. Then it is not possible to label the tree in such a way that it becomes a DOL tree. Indeed, infinitely many different subtrees arise.

2 Fragmentation

Consider the DOL system with the axiom a and rules

$$(1) \quad a \longrightarrow bc, b \longrightarrow bdc, c \longrightarrow bd^2c, d \longrightarrow bd^4c.$$

The beginning of the tree is as in Figure 3. We obviously need four labels for the simple reason that we have nodes of four different degrees: 2, 3, 4 and 6.

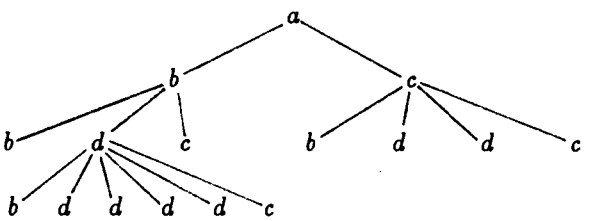

Figure 3.

However, we can represent the tree in the following much more compact way. The idea resembles DOL systems with *fragmentation*, introduced originally in [2].

Assume that the alphabet Σ contains a special letter $\#$, viewed as a marker or separator. Then we speak of *#-guarded subwords* of words y over Σ. They are the maximal parts of y separated by $\#$. For instance, if $y = ab\#a\#bab\#b$, then the $\#$- guarded subwords are ab, a, bab, b. Formally, a word x not containing $\#$ is a *#-guarded subword* of y iff $\#x\#$ is a subword of $\#y\#$.

Consider a *marked DOL system* $G_\# = (\Sigma, h, w)$, where the alphabet contains the marker $\#$, for which the rule is $\# \longrightarrow \#$. (Also now we assume that h is nonerasing and $|w| = 1$.) We now associate to $S(G_\#)$ a tree labeled by words over $(\Sigma - \{\#\})^*$. The labels of the tree will be the $\#$- guarded subwords of the words in $S(G_\#)$. In this process, several consecutive $\#$'s will be identified with one $\#$. Trees obtained in this fashion will be referred to as *generalized DOL trees*. Let us consider some examples.

If the marker $\#$ does not occur in the sequence, the generalized DOL tree has no branches. The generalized tree associated to the Fibonacci system is:

Figure 4.

Consider next the marked DOL system with the axiom a and rule

$$(2) \quad a \longrightarrow a^2\#a^3.$$

The generalized DOL tree begins now as follows:

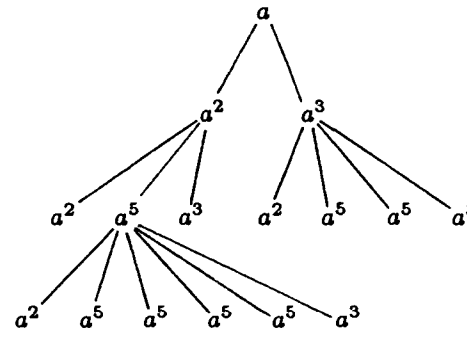

Figure 5.

We observe that the unlabeled tree is exactly the same as the one considered at the beginning of this section. Thus, in place of (1), we have obtained the much more compact representation (2)!

The term "generalised" can be justified as follows. An ordinary DOL system G can be transformed into a marked one $G_\#$ by separating all letters on the right sides of the rules with the marker $\#$. Since the axiom is a singleton letter, all labels of the resulting tree are singletons. Then the unlabeled versions of the DOL tree associated to G and the generalized DOL tree associated to $G_\#$ coincide. Consequently, the following result holds true.

Lemma 2.1 *The unlabeled version of every DOL tree equals the unlabeled version of a generalized DOL tree.*

Our main purpose is to prove the converse of Lemma 2.1. Thus, the unlabeled versions of DOL trees and generalised DOL trees coincide. However, in general, a marked DOL system provides a much more compact representation for the tree than a DOL system.

By Lemma 1.1, it suffices to prove that, given a marked DOL systems $G_\#$, there is a constant k such that all words appearing as labels in the generalized tree are of length less than k. Unfortunately, as such this claim is not true. Any DOL system generating an infinite language and not containing at all the marker $\#$ in its sequence, such as the Fibonacci system, provides a counterexample. Another counterexample is provided by the system with the axiom a and the rules

$$(3) \quad a \longrightarrow b\#ab, \quad b \longrightarrow b^2.$$

The generalized DOL tree is in this case

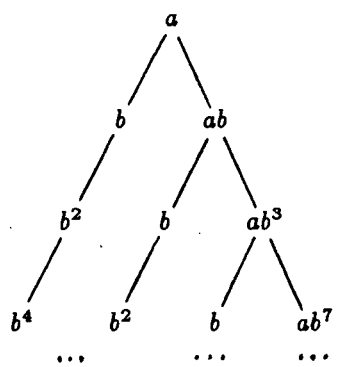

Figure 6.

However, in both cases our claim holds true. The above tree is generated by the DOL system with the axiom a and rules $a \longrightarrow ba$, $b \longrightarrow b$. The generalized tree of the Fibonacci system is generated by the DOL system with the axiom a and the rule $a \longrightarrow a$.

The tool for obtaining a constant k as described above is to eliminate the unbounded growth by transforming the given marked DOL system $G_\#$ into a marked system with the same (unlabeled version of the) generalized tree. We say that a letter b of $G_\#$ is *useful* if $b \Longrightarrow^* u\#v$, for some words u and v (that is, $h^i(b) = u\#v$, for some u, v, i). Otherwise, b is *useless*. Thus, the sequence starting from a useless letter does not contain the marker $\#$. Clearly, usefulness is a decidable property.

The *useful variant* $G'_{\#}$ of a marked DOL system $G_{\#}$ is constructed as follows. If all letters appearing in $S(G_{\#})$ are useful, then $G'_{\#} = G_{\#}$. If all letters are useless, then the axiom of $G'_{\#}$ is a and $a \longrightarrow a$ is the only rule. If every $\#$- guarded subword of the right-hand sides of the rules contains a useful letter, then to get $G'_{\#}$ we simply remove from $G_{\#}$ all useless letters and their occurrences in the rules. The case remains, where $S(G_{\#})$ contains useful letters but some $\#$-guarded subword of the right-hand side of some rule consists of useless letters. To get $G'_{\#}$, we also now first remove from $G_{\#}$ all useless letters and their occurrences in the rules. Then we add a new letter c with the rule $c \longrightarrow c$. Finally, all $\#$-guarded subwords that previously consisted of useless letters are replaced by c.

For instance, if $G_{\#}$ is defined by the rules (3), $G'_{\#}$ will be defined by the rules

$$a \longrightarrow c\#a, \quad c \longrightarrow c.$$

If $G_{\#}$ has the axiom a and the rules

$$a \longrightarrow d\#bcc\#d, \quad b \longrightarrow a^2d\#ab, \quad c \longrightarrow cd, \quad d \longrightarrow dcd,$$

then $G'_{\#}$ will be defined by the rules

$$a \longrightarrow c\#b\#c, \quad b \longrightarrow a^2\#ab, \quad c \longrightarrow c.$$

The following result is immediate by the construction of $G'_{\#}$.

Lemma 2.2 *If $G'_{\#}$ is the useful variant of a marked DOL system $G_{\#}$, then the unlabeled versions of the generalized DOL trees associated to $G_{\#}$ and $G'_{\#}$ coincide.*

3 The main result

We will establish in this section the converse of Lemma 2.1.

Theorem 3.1 *The unlabeled version of every generalized DOL tree equals the unlabeled version of a DOL tree. Moreover, given a marked DOL system producing a generalized tree, the corresponding DOL system can be effectively constructed.*

Thus, every tree possessing a compact representation (2) is a DOL tree, (as far as the unlabeled versions are concerned) and the corresponding DOL representation (1) can be effectively constructed. Let us discuss still a more sophisticated example.

A marked DOL system $G_{\#}$ has the axiom a and rules

$$a \longrightarrow a\#ab\#ab^2, \quad b \longrightarrow a.$$

Observe first that both a and b are useful and, thus, $G'_{\#} = G_{\#}$. Since the generalized tree is quite involved, we give it in parts, continuing the process as long as new labels are born:

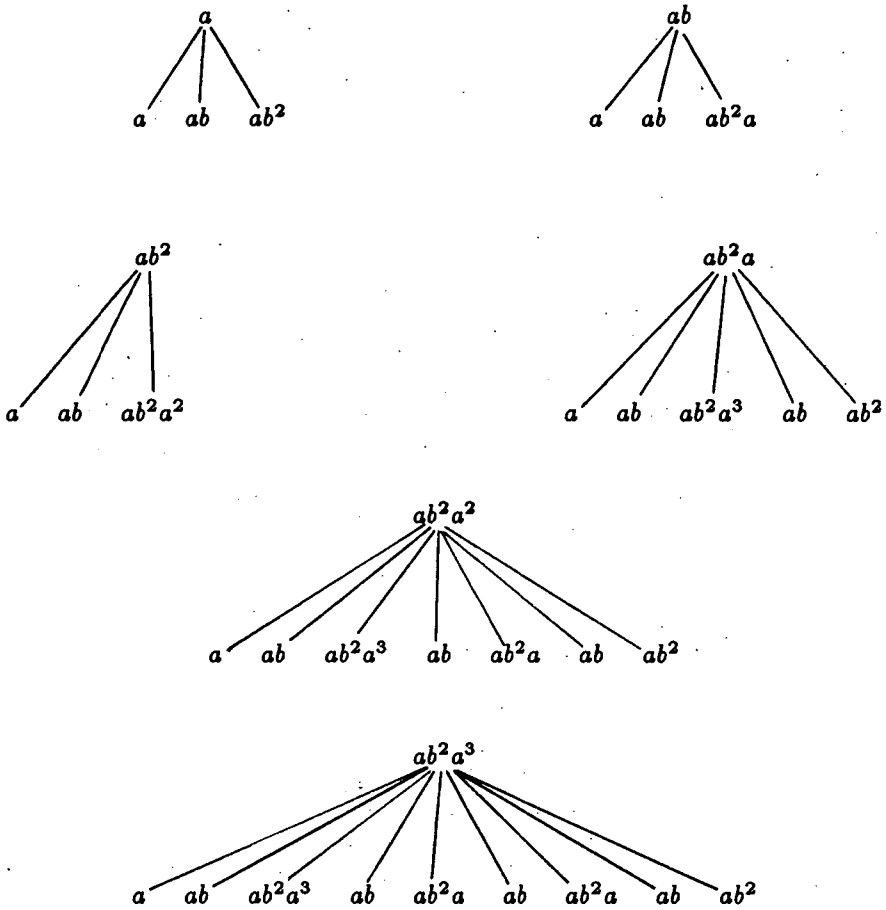

Figure 7.

Thus, if we denote the six labels appearing at the roots by a, b, c, d, e, f, we obtain the rules of the corresponding DOL system:

$$a \longrightarrow abc \qquad b \longrightarrow abd, \qquad c \longrightarrow abe,$$
$$d \longrightarrow abfbc, \quad e \longrightarrow abfbdbc, \quad d \longrightarrow abfbdbdbc.$$

We will now establish our Theorem. By Lemma 2.2 we may restrict the attention to useful variants. We have to show that a constant k can be effectively computed from the system such that all labels in the generalized tree are shorter than k. More specifically, we have to establish the following result.

Lemma 3.1 *Assume that $G_\#$ is a marked DOL system coinciding with its useful variant, $G_\# = G'_\#$. Then a constant k can be effectively computed such that the length of every label in the generalized DOL tree of $G_\#$ is at most k.*

Proof. The alphabet Σ contains at most one useless letter, c. Let Σ' be the subalphabet obtained by excluding $\#$ and c, and let r be the cardinality of Σ'. Thus, all

letters of Σ' are useful. Define the *rank* of a letter $a \in \Sigma'$ to be the smallest integer k such that $h^k(a)$ contains an occurrence of #. Clearly, the rank can be effectively computed and every letter is of rank $\leq r$.

Consider the lengths of #-guarded subwords of the words $h(a)$ when a ranges over letters of rank 1. Let m_1 be twice the maximal length. Define further

$$m_2 = \max\{|h(a)| \mid a \text{ is of rank } > 1\},$$

$$M = \max\{m_1, m_2\}.$$

We claim that we can choose

$$k = M^r + M^{r-1} + \ldots + M = (M^r - 1)M/(M-1).$$

Let v be a label in the generalized DOL tree. We have to estimate $|v|$ and show that $|v| \leq k$. Clearly, we may assume that v is not the label of the root. Hence, v is a #-guarded subword of $h(\bar{v})$, where $\bar{v}$ is in the sequence $S(G_\#)$. The situation can be depicted as follows, with $v = u_1u_3u_2$:

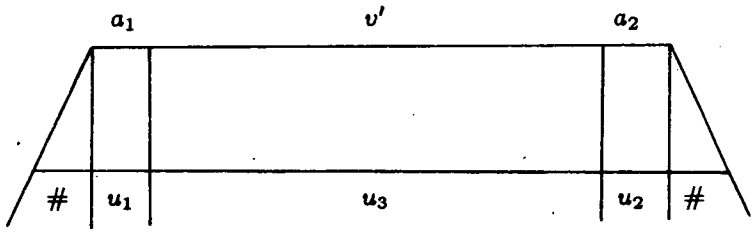

Figure 8.

Here every letter of v', if any, is of rank > 1, and a_1, a_2 are letters of rank 1. Thus, we look how the #-guarded subword v is created. a_1 and a_2 may also produce something else beyond the marker #. One of them (or both) may be missing if we are dealing with a prefix or suffix. We obtain the estimates

$$|u_1| + |u_2| \leq m_1 \text{ and } |u_3| \leq m_2|v'|$$

and, consequently,

$$|v| \leq M \cdot |v'| + M.$$

We now estimate similarly the length $|v'|$. (In fact, we obtain an upper bound for an eventually longer word that contains also a_1, a_2 and maybe still a prefix and suffix.) By considering the preceding word in the sequence, we get an analogous picture and the estimate

$$|v'| \leq M \cdot |v''| + M,$$

where the letters of v'', if any, are of rank > 2. Consequently,

$$|v| \leq M(M \cdot |v''| + M) + M = M^2 \cdot |v''| + M^2 + M.$$

Continuing in the same way, we obtain

$$|v| \leq M^r |v^{(r)}| + M^r + M^{r-1} + \ldots + M,$$

where every letter in $v^{(r)}$, if any, is of rank $> r$. But there are no letters of rank $> r$. Thus, $v^{(r)}$ is the empty word and, consequently,

$$|v| \leq M^r + M^{r-1} + \ldots + M = k.$$

This concludes the proof of Lemma 3.1 and also the proof of our Theorem. □

Consider the example discussed at the beginning of this section. We obtain

$$r = 2, \quad m_1 = 2 \cdot 3 = 6, \quad m_2 = 1, \quad M = 6, \quad k = 42,$$

whereas in the actual construction the maximal word length was 6. Indeed, our bound k can be improved. For instance, in the definition of m_1 it suffices to consider the sum of the lengths of the maximal #-guarded prefix and suffix, rather than twice the maximal word length. This improvement gives $m_1 = M = 4$, $k = 20$.

4 Conclusion

We have introduced a compact way of representing certain infinite trees. The method uses DOL systems with fragmentation and leads to trees whose nodes are labeled by words. Although the lengths of such words may grow beyond all bounds, the unlabeled versions of the trees are still regular and, thus, possess a DOL representation. However, the loss in compactness in the transition to the DOL representation can be enormous.

We do not investigate in this paper the complexity issues involved or for which classes of trees the new representation is especially suitable. We conclude with the following result along these lines. The result is easily established by extending the example of the preceding section, for values of $r > 2$, to contain the rules

$$a \longrightarrow a\#ab\#ab^2\#\ldots\#ab^r, \quad b \longrightarrow a.$$

Lemma 4.1 *For each $r > 2$, there is an infinite unlabeled tree T such that (i) T is the unlabeled version of the generalized DOL tree of a marked DOL system with 2 letters, and (ii) T is not the unlabeled version of the tree of any DOL system with $\leq r$ letters.*

References

[1] J.Dassow, G.Paun and A.Salomaa. On the union of OL languages. *Information Processing Letters* 47 (1993) 59-63.

[2] G.Rozenberg, K.Ruohonen and A.Salomaa. Developmental systems with fragmentation. *International Journal of Computer Mathematics* 5 (1976) 177- 191.

[3] G.Rozenberg and A.Salomaa. *The Mathematical Theory of L Systems.* Academic Press, New York (1980).

[4] A.Salomaa. Simple reductions between DOL language and sequence equivalence problems. *Discrete Applied Mathematics* 41 (1993) 271- 274.

Received January, 1995

Acta Cybernetica, Vol. 12, No. 1, Szeged, 1995

On isomorphic representation of nondeterministic tree automata*

B. Imreh †

Abstract

In this paper we deal with isomorphically complete systems of finite nondeterministic tree automata with respect to the general product and the cube-product. In both cases characterizations of isomorphically complete systems are presented which imply that the general product and the cube-product are equivalent regarding isomorphic completeness.

In the theory of finite automata it is a central problem to characterize such systems from which any automaton can be represented isomorphically or homomorphically under a given composition. Such systems are called isomorphically, respectively, homomorphically complete with respect to the composition considered. From the practical point of view, finite complete systems have great importance. The first composition admitting finite isomorphically complete systems was introduced by V. M. Glushkov in [7], who gave a characterization of the isomorphically complete systems. Later F. Gécseg [2] introduced a product hierarchy, the α_i-products, $i = 0, 1, \ldots$, and Z. Ésik [1] proved that, from the point of view of homomorphic completeness, Glushkov's composition is equivalent to the α_i-product for $i \geq 2$. Regarding isomorphic completeness, it turned out that there is no finite isomorphically complete system with respect to any of the α_i-products. A systematic account of the results on α_i-products including the ones mentioned above can be found in the monograph [3].

The first generalization of Glushkov's result to tree automata was given by M. Steinby in [10]. The generalization of the notion of finite automata to trees has a rigorius mathematical discussion in [6]. Another generalization of Gluskhov's result to nondeterministic automata is given in [4]. In this paper we extend this result to nondeterministic tree automata. Namely, we define the Glushkov-type product of nondeterministic tree automata and characterize the isomorphically complete systems with respect to this composition. Our characterization implies the existence of finite isomorphically complete systems of nondeterministic tree automata with respect to this product.

The cube-product, which is a simpler composition than Glushkov's one, was introduced in [8] where a characterization of the isomorphically complete systems

*This research has been supported by the Hungarian Foundation for Scientific Research, (OTKA), Grant 2035 and by the Hungarian Cultural and Educational Ministry, (MKM), Grant 434/94.

†Department of Informatics, A. József University Árpád tér 2, Szeged 6720, Hungary

with respect to this product was presented as well. From this characterization it follows that the Glushkov-type product and the cube-product are equivalent regarding isomorphic completeness.

The generalisation of the cube-product to tree automata and the characterisation of the isomorphically complete systems with respect to it is given in [9]. A similar generalisation and characterization for nondeterministic automata is presented in [5]. In both cases the characterisation of the isomorphically complete systems implies that the Glushkov-type composition and the cube-pruduct are equivalent regarding isomorphic completeness. Here we generalize the cube-product to nondeterministic tree automata and give a characterization of the isomorphically complete systems with respect to it. Our characterization shows that the cube-product and the Glushkov-type product are equivalent regarding isomorphic completeness for the class of nondeterministic tree automata, too.

To start the discussion, we introduce some notions and notations. By a set of *relational symbols* we mean a nonempty union $\Sigma = \Sigma_1 \bigcup \Sigma_2 \bigcup \ldots$, where Σ_m, $m = 1, 2, \ldots$, are pairwise disjoint sets of symbols. For any $m \geq 1$, the set Σ_m is called the set of *m-ary relational symbols*. It is said that the *rank* or *arity* of a symbol $\sigma \in \Sigma$ is m if $\sigma \in \Sigma_m$. Now let a set Σ of relational symbols and a set R of positive integers be given. R is called the *rank-type* of Σ if for any integer $m \geq 0$, $\Sigma_m \neq \emptyset$ if and only if $m \in R$. In the sequel we shall work under a fixed rank-type R.

Now let Σ be a set of relational symbols with rank-type R. By a *nondeterministic Σ-algebra* $\mathcal{A}$ we mean a pair consisting of a nonempty set A and a mapping that assigns to every relational symbol $\sigma \in \Sigma$ an m-ary relation $\sigma^A \subseteq A^m$, where the arity of σ is m. The set A is called the *set of elements of* $\mathcal{A}$ and σ^A is the *realization of* σ *in* $\mathcal{A}$. The mapping $\sigma \to \sigma^A$ will not be mentioned explicitly, we only write $\mathcal{A} = (A, \Sigma)$. For any $m \in R$, $\sigma \in \Sigma_m$, $(a_1, \ldots, a_{m-1}) \in A^{m-1}$, we denote by $(a_1, \ldots, a_{m-1})\sigma^A$ the set $\{a : a \in A \ \& \ \sigma^A(a_1, \ldots, a_{m-1}, a)\}$. If $(a_1, \ldots, a_{m-1})\sigma^A$ is a one-element set $\{a\}$, then we write $(a_1, \ldots, a_{m-1})\sigma^A = a$.

It is said that a nondeterministic Σ-algebra $\mathcal{A}$ is *finite* if A is finite, and it is of *finite type* if Σ is finite. By a *nondeterministic tree automaton* we mean a finite nondeterministic algebra of finite type. Finally, it is said that the *rank-type of a nondeterministic tree automaton* $\mathcal{A} = (A, \Sigma)$ is R if the rank-type of Σ is R.

Let $\mathcal{A} = (A, \Sigma)$ and $\mathcal{B} = (B, \Sigma)$ be nondeterministic tree automata with rank-type R. $\mathcal{B}$ is called a *subautomaton* of $\mathcal{A}$ if $B \subseteq A$ and, for all $m \in R$ and $\sigma \in \Sigma_m$, σ^B is the restriction of σ^A to B^m. A one-to-one mapping μ of A onto B is called an *isomorphism* of $\mathcal{A}$ onto $\mathcal{B}$ if $\sigma^A(a_1, \ldots, a_m)$ if and only if $\sigma^B(\mu(a_1), \ldots, \mu(a_m))$, for all $m \in R$, $(a_1, \ldots, a_m) \in A^m$, $\sigma \in \Sigma_m$. In this case it is said that $\mathcal{A}$ and $\mathcal{B}$ are *isomorphic*. It is easy to see that μ is an isomorphism of $\mathcal{A}$ onto $\mathcal{B}$ if and only if $(a_1, \ldots, a_{m-1})\sigma^A\mu = (\mu(a_1), \ldots, \mu(a_{m-1}))\sigma^B$ holds, for all $m \in R$, $\sigma \in \Sigma_m$, $(a_1, \ldots, a_{m-1}) \in A^{m-1}$.

Now let us denote by $\mathfrak{U}_R$ the class of all nondeterministic tree automata with rank-type R. A composition of nondeterministic tree automata from $\mathfrak{U}_R$ can be represented as a network in which each vertex denotes a nondeterministic tree automaton and the actual relation of a component automaton may depend only on those automata which have a direct connection to it.

In order to define this notion of composition let $\mathfrak{D}$ denote an arbitrary nonempty fixed set of finite directed graphs. We assume that the vertices of any graph in $\mathfrak{D}$ having n vertices are denoted by the numbers $1, \ldots, n$. Let $\mathcal{A} = (A, \Sigma) \in \mathfrak{U}_R$ and $\mathcal{A}_j = (A_j, \Sigma^{(j)}) \in \mathfrak{U}_R$, $j = 1, \ldots, n$. Furthermore, take a family Ψ of mappings

$$\Psi_{mj} : (A_1 \times \ldots \times A_n)^{m-1} \times \Sigma_m \to \Sigma_m^{(j)}, \quad m \in R, \quad 1 \leq j \leq n .$$

It is said that the nondeterministic tree automaton $\mathcal{A}$ is a $\mathfrak{D}$-*product* of the automata $\mathcal{A}_j$, $j = 1, \ldots, n$, with respect to Ψ if the following conditions are satisfied:

(i) $A = \prod_{j=1}^{n} A_j$

(ii) there exists a graph $D = (\{1, \ldots, n\}, E)$ in $\mathfrak{D}$ such that for any $m \in R$, $j \in \{1, \ldots, n\}$ and $((a_{11}, \ldots, a_{1n}), \ldots, (a_{m-11}, \ldots, a_{m-1n})) \in A^{m-1}$, the mapping Ψ_{mj} is independent of the elements a_{ts}, $t = 1, \ldots, m-1$, if $(s, j) \notin E$,

(iii) for any $m \in R$, $\sigma \in \Sigma_m$ and

$((a_{11}, \ldots, a_{1n}), \ldots, (a_{m-11}, \ldots, a_{m-1n})) \in A^{m-1}$,

$((a_{11}, \ldots, a_{1n}), \ldots, (a_{m-11}, \ldots, a_{m-1n}))\sigma^{\mathcal{A}} =$

$(a_{11}, \ldots, a_{m-11})\sigma_1^{\mathcal{A}_1} \times \ldots \times (a_{1n}, \ldots, a_{m-1n})\sigma_n^{\mathcal{A}_n}$

where

$$\sigma_j = \Psi_{mj}((a_{11}, \ldots, a_{1n}), \ldots, (a_{m-11}, \ldots, a_{m-1n}), \sigma), \qquad j = 1, \ldots, n.$$

We shall use the notation

$$\prod_{j=1}^{n} \mathcal{A}_j(\Sigma, \Psi, D)$$

for the product introduced above. In particular, if $\mathcal{A}_j$, $j = 1, \ldots, n$, are identical copies of some nondeterministic tree automaton $\mathcal{B}$, then we speak of a *general power* and we write $\mathcal{B}^n(\Sigma, \Psi, D)$ for $\prod_{j=1}^{n} \mathcal{A}_j(\Sigma, \Psi, D)$.

Let $\mathfrak{B}$ be a system of nondeterministic tree automata from $\mathfrak{U}_R$. It is said that $\mathfrak{B}$ is *isomorphically complete for* $\mathfrak{U}_R$ *with respect to the* $\mathfrak{D}$-*product* if any nondeterministic tree automaton from $\mathfrak{U}_R$ is isomorphic to a subautomaton of a $\mathfrak{D}$-product of nondeterministic tree automata from $\mathfrak{B}$.

In the sequel we shall need a special two-state nondeterministic tree automata. For every $m \in R$, let us assign a symbol to each m-ary relation on $\{0, 1\}$. Let $\bar{\Sigma}_m$ denote the set of these symbols and let $\bar{\Sigma} = \bigcup_{m \in R} \bar{\Sigma}_m$. Define the nondeterministic tree automaton $\mathcal{G} = (\{0, 1\}, \bar{\Sigma})$ such that, for every $m \in R$ and $\sigma \in \Sigma_m$, $\sigma^{\mathcal{G}}$ is the corresponding m-ary relation.

Now let $\mathfrak{D}$ be the set of all finite directed complete graphs having as vertices the sets $\{1, \ldots, n\}$, $n = 1, 2, \ldots$. Then the $\mathfrak{D}$-product is equal to the Glushkov-type product which is also called *general product*. We note that in this case the finite directed complete graphs are considered as possible networks. Since n determines the corresponding complete graph uniquely, we omit the graph component from the notation of the general product.

Regarding the general product, the following statement can be proved easily.

Lemma. *Let $\mathcal{A} = (A, \Sigma) \in \mathfrak{U}_R$, $\mathcal{A}_j = (A_j, \Sigma^{(j)}) \in \mathfrak{U}_R$, $j = 1, \ldots, n$, and $\mathcal{B}_{jt} = (B_{jt}, \Sigma^{(jt)}) \in \mathfrak{U}_R$, $t = 1, \ldots, i_j$, $j = 1, \ldots, n$. If $\mathcal{A}$ is isomorphic to a subautomaton of a general product $\prod_{j=1}^{n} \mathcal{A}_j(\Sigma, \Psi)$ and, for each $j \in \{1, \ldots, n\}$, $\mathcal{A}_j$ is isomorphic to a subautomaton of a general product $\prod_{t=1}^{i_j} \mathcal{B}_{jt}(\Sigma^{(j)}, \Psi^{(j)})$, then $\mathcal{A}$ is isomorphic to a subautomaton of a general product of the nondeterministic tree automata $\mathcal{B}_{jt}$, $t = 1, \ldots, i_j$, $j = 1, \ldots, n$.*

The following theorem provides necessary and sufficient conditions for a system of nondeterministic tree automata from $\mathfrak{U}_R$ to be isomorphically complete for $\mathfrak{U}_R$ with respect to the general product.

Theorem 1. *A system $\mathfrak{B}$ of nondeterministic tree automata from $\mathfrak{U}_R$ is isomorphically complete for $\mathfrak{U}_R$ with respect to the general product if and only if, for all $m \in R$ and $\mathbf{i} = (i_1, \ldots, i_m) \in \{0,1\}^m$, $\mathfrak{B}$ contains a nondeterministic tree automata $\mathcal{A}^{(\mathbf{i})} = (A^{(\mathbf{i})}, \Sigma^{(\mathbf{i})})$ satisfying the following conditions:*

(1) *$A^{(\mathbf{i})}$ has two different elements $a_0^{(\mathbf{i})}$ and $a_1^{(\mathbf{i})}$,*

(2) *there exists a $\bar{\sigma}_{\mathbf{i}} \in \Sigma_m^{(\mathbf{i})}$ with $(a_{i_1}^{(\mathbf{i})}, \ldots, a_{i_{m-1}}^{(\mathbf{i})})\bar{\sigma}_{\mathbf{i}}^{\mathcal{A}^{(\mathbf{i})}} \bigcap \{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} = \{a_{i_m}^{(\mathbf{i})}\}$,*

(3) *for all $u \in R$ and $\mathbf{s} = (s_1, \ldots, s_{u-1}) \in \{0,1\}^{u-1}$, there is a $\sigma_{\mathbf{i},\mathbf{s}} \in \Sigma_u^{(\mathbf{i})}$ for which $\{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} \subseteq (a_{s_1}^{(\mathbf{i})}, \ldots, a_{s_{u-1}}^{(\mathbf{i})})\sigma_{\mathbf{i},\mathbf{s}}^{\mathcal{A}^{(\mathbf{i})}}$ provided that $u \neq 1$, and there is a $\sigma_{\mathbf{i}}^* \in \Sigma_1^{(\mathbf{i})}$ with $\{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} \subseteq \sigma_{\mathbf{i}}^{*\mathcal{A}^{(\mathbf{i})}}$ if $1 \in R$ and $u = 1$.*

Proof. In order to prove the necessity, let us suppose that $\mathcal{B}$ is an isomorphically complete system of nondeterministic tree automata for $\mathfrak{U}_R$ with respect to the general product. Then there are $\mathcal{A}_j = (A_j, \Sigma^{(j)}) \in \mathfrak{U}_R$, $j = 1, \ldots, n$, such that $\mathcal{G}$ is isomorphic to a subautomaton $\mathcal{A} = (A, \Sigma)$ of a general product $\prod_{j=1}^{n} \mathcal{A}_j(\Sigma, \Psi)$. Let μ denote a suitable isomorphism and let

$$\mu(0) = (a_{01}, \ldots, a_{0n}) \text{ and } \mu(1) = (a_{11}, \ldots, a_{1n}).$$

Let us denote by K the set $\{k : 1 \leq k \leq n \ \& \ a_{0k} \neq a_{1k}\}$. Obviously, $K \neq \emptyset$. Now let $m \in R$ and $(i_1, \ldots, i_m) \in \{0,1\}^m$ be arbitrarily fixed elements. We distinguish two cases depending on m.

First let us suppose that $m \neq 1$. By the definition of $\mathcal{G}$, there is a $\bar{\sigma} \in \Sigma_m$ with $(i_1, \ldots, i_{m-1})\bar{\sigma}^{\mathcal{G}} = i_m$. Since μ is an isomorphism, this yields

$$(\mu(i_1), \ldots, \mu(i_{m-1}))\bar{\sigma}^{\mathcal{A}} = \mu(i_m).$$

Therefore, $a_{i_m k} \in (a_{i_1 k}, \ldots, a_{i_{m-1} k})\bar{\sigma}_k^{\mathcal{A}_k}$ holds, for all $k \in K$, where

$$\bar{\sigma}_k = \Psi_{mk}((a_{i_1 1}, \ldots, a_{i_1 n}), \ldots, (a_{i_{m-1} 1}, \ldots, a_{i_{m-1} n}), \bar{\sigma}).$$

But then there exists at least one index $l \in K$ such that

$$(a_{i_1 l}, \ldots, a_{i_{m-1} l})\bar{\sigma}_l^{\mathcal{A}_l} \bigcap \{a_{0l}, a_{1l}\} = \{a_{i_m l}\}.$$

Now let $1 \neq u \in R$ and $\mathbf{s} = (s_1, \ldots, s_{u-1}) \in \{0,1\}^{u-1}$ be arbitrary. By the definition of $\mathcal{G}$, there exists a $\sigma_{\mathbf{s}} \in \bar{\Sigma}_u$ with $(s_1, \ldots, s_{u-1})\sigma_{\mathbf{s}}^{\mathcal{G}} = \{0,1\}$. Since μ is an isomorphism, this implies

$$(\mu(s_1), \ldots, \mu(s_{u-1}))\sigma_{\mathbf{s}}^{\mathcal{A}} = \{\mu(0), \mu(1)\}.$$

Then $\{a_{0k}, a_{1k}\} \subseteq (a_{s_1 k}, \ldots, a_{s_{u-1}k})\sigma_{\mathbf{s},k}^{\mathcal{A}_k}$ holds, for all $k \in K$, where

$$\sigma_{\mathbf{s},k} = \Psi_{uk}((a_{s_1 1}, \ldots, a_{s_1 n}), \ldots, (a_{s_{u-1}1}, \ldots, a_{s_{u-1}n}), \sigma_{\mathbf{s}}).$$

Therefore, $\{a_{0l}, a_{1l}\} \subseteq (a_{s_1 l}, \ldots, a_{s_{u-1}l})\sigma_{\mathbf{s},l}^{\mathcal{A}_l}$. If $1 \in R$ and $u = 1$, then, by the definition of $\mathcal{G}$, there is a $\sigma^* \in \Sigma_1$ with $\sigma^* = \{0,1\}$. But then $\sigma^{*\mathcal{A}} = \{\mu(0), \mu(1)\}$, and so, $\{a_{0k}, a_{1k}\} \subseteq \sigma_k^{*\mathcal{A}_k}$, for all $k \in K$, where $\sigma_k^* = \Psi_{1k}(\sigma^*)$. Thus $\{a_{0l}, a_{1l}\} \subseteq \sigma_l^{*\mathcal{A}_l}$. This ends the proof of the necessity in the case $m \neq 1$.

Let us assume that $m = 1$. By the definition of $\mathcal{G}$, there is a $\bar{\sigma} \in \bar{\Sigma}_1$ with $\bar{\sigma}^{\mathcal{G}} = i_1$. But then $\bar{\sigma}^{\mathcal{A}} = \mu(i_1)$. Therefore, $a_{i_1 k} \in \bar{\sigma}_k^{\mathcal{A}_k}$ is valid, for all $k \in K$, where $\bar{\sigma}_k = \Psi_{1k}(\bar{\sigma})$. From this it follows that there exists at least one $l \in K$ such that

$$\bar{\sigma}_l^{\mathcal{A}_l} \bigcap \{a_{0l}, a_{1l}\} = \{a_{i_1 l}\}.$$

Now let $u \in R$ and $\mathbf{s} = (s_1, \ldots, s_{u-1}) \in \{0,1\}^{u-1}$ be fixed arbitrarily. In a similar way as above, it is easy to see that there is a $\sigma_{\mathbf{s},l} \in \Sigma_u^{(l)}$ such that $\{a_{0l}, a_{1l}\} \subseteq (a_{s_1 l}, \ldots, a_{s_{u-1}l})\sigma_{\mathbf{s},l}^{\mathcal{A}_l}$ if $u \neq 1$, and there is a $\sigma_l^* \in \Sigma_1^{(l)}$ with $\sigma_l^{*\mathcal{A}_l} = \{0,1\}$ if $u = 1$. This ends the proof of the necessity.

In order to prove the sufficiency, let us suppose that $\mathfrak{B}$ satisfies the conditions of Theorem 1. The isomorphic completeness of $\mathfrak{B}$ is proved in two steps.

First we show that $\mathcal{G}$ is isomorphic to a subautomaton of a general product of nondeterministic tree automata from $\mathfrak{B}$. For this reason let us denote by W the set $\bigcup_{m \in R}\{0,1\}^m$ and let $|W| = n$. Moreover, let γ denote a one-to-one mapping of the set $\{1, \ldots, n\}$ onto W. By our assumption on $\mathfrak{B}$, for any $j \in \{1, \ldots, n\}$, there exists an $\mathcal{A}^{(\gamma(j))} = (A^{(\gamma(j))}, \Sigma^{(\gamma(j))}) \in \mathfrak{B}$ satisfying conditions (1), (2) and (3) with $\mathbf{i} = \gamma(j)$. Form the general product $\prod_{j=1}^{n} \mathcal{A}^{(\gamma(j))}(\bar{\Sigma}, \Psi)$ in the following way.

Let $A = \{(a_0^{(\gamma(1))}, \ldots, a_0^{(\gamma(n))}), (a_1^{(\gamma(1))}, \ldots, a_1^{(\gamma(n))})\}$. Since $a_0^{(\gamma(j))} \neq a_1^{(\gamma(j))}$, $j = 1, \ldots, n$, we obtain that $|A| = 2$. Let us define the mapping μ of $\{0,1\}$ onto A by

$$\mu(0) = (a_0^{(\gamma(1))}, \ldots, a_0^{(\gamma(n))}) \text{ and } \mu(1) = (a_1^{(\gamma(1))}, \ldots, a_1^{(\gamma(n))}).$$

Now let $1 \neq m \in R$, $\sigma \in \bar{\Sigma}_m$, $(a_{i_t}^{(\gamma(1))}, \ldots, a_{i_t}^{(\gamma(n))}) \in A$, $t = 1, \ldots, m-1$, be arbitrarily fixed elements and let $\mathbf{i}^*$ denote the vector $(i_1, \ldots, i_{m-1})$. Then, for any $j \in \{1, \ldots, n\}$, let

$$\Psi_{mj}\left((a_{i_1}^{(\gamma(1))},\dots,a_{i_1}^{(\gamma(n))}),\dots,(a_{i_{m-1}}^{(\gamma(1))},\dots,a_{i_{m-1}}^{(\gamma(n))}),\sigma\right) =$$

$$= \begin{cases} \bar{\sigma}_{\gamma(j)} & \text{if } \mathbf{i}^*\sigma^{\mathcal{G}} = i_m \text{ and } \gamma(j) = (i_1,\dots,i_m), \\ \sigma_{\gamma(j),\mathbf{i}^*} & \text{if } \mathbf{i}^*\sigma^{\mathcal{G}} = i_m \text{ and } \gamma(j) \neq (i_1,\dots,i_m), \\ \sigma_{\gamma(j),\mathbf{i}^*} & \text{if } \mathbf{i}^*\sigma^{\mathcal{G}} = \{0,1\}, \\ \bar{\sigma}_{\gamma(j)} & \text{if } \mathbf{i}^*\sigma^{\mathcal{G}} = \emptyset \text{ and } (\gamma(j) = (i_1,\dots,i_{m-1},0) \\ & \text{or } \gamma(j) = (i_1,\dots,i_{m-1},1)), \\ \sigma_{\gamma(j),\mathbf{i}^*} & \text{if } \mathbf{i}^*\sigma^{\mathcal{G}} = \emptyset \text{ and } \gamma(j) \neq (i_1,\dots,i_{m-1},0) \\ & \text{and } \gamma(j) \neq (i_1,\dots,i_{m-1},1). \end{cases}$$

In all other cases when $1 \neq m \in R$, let the value of Ψ_{mj} be an arbitrarily fixed element of $\Sigma_m^{(\gamma(j))}$.

If $1 \in R$ and $m = 1$, then the mappings Ψ_{1j}, $j = 1,\dots,n$, are defined in the following way. For any $\sigma \in \bar{\Sigma}_1$, let

$$\Psi_{1j}(\sigma) = \begin{cases} \bar{\sigma}_{\gamma(j)} & \text{if } \sigma^{\mathcal{G}} = i_1 \text{ and } \gamma(j) = (i_1), \\ \sigma^*_{\gamma(j)} & \text{if } \sigma^{\mathcal{G}} = i_1 \text{ and } \gamma(j) \neq (i_1), \\ \bar{\sigma}_{\gamma(j)} & \text{if } \sigma^{\mathcal{G}} = \emptyset \text{ and } (\gamma(j) = (0) \text{ or } \gamma(j) = (1)), \\ \sigma^*_{\gamma(j)} & \text{otherwise.} \end{cases}$$

Now consider the subautomaton $\mathcal{A} = (A, \bar{\Sigma})$ of the general product $\prod_{j=1}^{n} \mathcal{A}^{(\gamma(j))}(\bar{\Sigma}, \Psi)$ which is determined by the set A. It is easy to show that μ is an isomorphism of $\mathcal{G}$ onto the subautomaton $\mathcal{A}$.

As a second step, we prove that an arbitrary nondeterministic tree automaton from $\mathfrak{U}_R$ is isomorphic to a subautomaton of a general power of $\mathcal{G}$. For this reason let $\mathcal{C} = (C, \Sigma) \in \mathfrak{U}_R$ be arbitrary with $C = \{c_1,\dots,c_r\}$. Let us take all the r-dimensional column vectors with components 0, 1, and order them in lexicographically increasing order. Let $\mathbf{Q}^{(r)}$ denote the matrix formed by these column vectors. Then $\mathbf{Q}^{(r)}$ is a matrix of type $r \times 2^r$ over $\{0,1\}$ and its row vectors are pairwise different. Moreover, let us observe that for any subset V of the set $\{1,\dots,r\}$, there exists exactly one index $k \in \{1,\dots,2^r\}$ such that for all $t \in \{1,\dots,r\}$, $t \in V$ if and only if $q_{tk}^{(r)} = 0$. Let us define the one-to-one mapping ν of $\{c_1,\dots,c_r\}$ onto the set of the row vectors of $\mathbf{Q}^{(r)}$ by $\nu(c_i) = (q_{i1}^{(r)},\dots,q_{i2^r}^{(r)})$, $i = 1,\dots,r$. Let $A = \{\nu(c_i) : i = 1,\dots,r\}$. Then $A \subseteq \{0,1\}^{2^r}$. Now we define the general power $\mathcal{G}^{2^m}(\Sigma, \Psi)$ in the following way.

Let $1 \neq m \in R$, $\sigma \in \Sigma_m$, $(q_{i_t1}^{(r)},\dots,q_{i_t2^r}^{(r)}) \in A$, $t = 1,\dots,m-1$, be arbitrary elements. In this case $\nu(c_{i_t}) = (q_{i_t1}^{(r)},\dots,q_{i_t2^r}^{(r)})$, $t = 1,\dots,m-1$. Let us suppose that $(c_{i_1},\dots,c_{i_{m-1}})\sigma^{\mathcal{C}} = \{c_{s_1},\dots,c_{s_l}\}$. Then $0 \leq l \leq r$. For each $j \in \{1,\dots,2^r\}$, let us denote by V_j the set $\{q_{s_1j}^{(r)},\dots,q_{s_lj}^{(r)}\}$. Obviously, $V_j \subseteq \{0,1\}$, $j = 1,\dots,2^r$. Thus, by the definition of $\mathcal{G}$, there exists a $\sigma_j \in \bar{\Sigma}_m$ with $(q_{i_1j}^{(r)},\dots,q_{i_{m-1}j}^{(r)})\sigma_j^{\mathcal{G}} = V_j$. Let us define the mappig Ψ_{mj} by

$$\Psi_{mj}((q_{i_11}^{(r)},\dots,q_{i_12^r}^{(r)}),\dots,(q_{i_{m-1}1}^{(r)},\dots,q_{i_{m-1}2^r}^{(r)}),\sigma) = \sigma_j.$$

In all other cases when $1 \neq m \in R$, let the value of Ψ_{mj} be an arbitrarily fixed symbol from Σ_m.

If $1 \in R$, $\sigma \in \Sigma_1$, $j \in \{1, \ldots, 2^r\}$, then the mappings Ψ_{1j}, $j = 1, \ldots, 2^r$, are defined as follows. Let us assume that $\sigma^C = \{c_{s_1}, \ldots, c_{s_l}\}$ and define the sets V_j, $j = 1, \ldots, 2^r$, in the same way as above. Again, by the definition of $\mathcal{G}$, there is a $\sigma_j^* \in \Sigma_1$ with $\sigma_j^{*\mathcal{G}} = V_j$. We put

$$\Psi_{1j}(\sigma) = \sigma_j^*.$$

Now let us consider the subautomaton $\mathcal{A} = (A, \Sigma)$ of the general power $\mathcal{G}^{2^r}(\Sigma, \Psi)$. Then it is easy to see that ν is an isomorphism of C onto $\mathcal{A}$. By our Lemma, the above isomorphic representations imply the sufficiency of the conditions which ends the proof of Theorem 1.

Remark. If $R = \{2\}$, then $\mathfrak{U}_{\{2\}}$ is the class of all nondeterministic automata. In this case our theorem gives a characterization of the isomorphically complete systems for the class of nondeterministic automata with respect to the general product. Therefore, Theorem 1 in [4] can be obtained as a corollary of our theorem.

In [8], n-dimensional hypercubes are used as possible networks. Now we define the product related to these networks for nondeterministic tree automata and characterize the isomorphically complete systems with respect to this product.

To introduce the formal definition of cube-product we need some preparation. Let $n \geq 2$ be an arbitrary integer and consider the n-dimensional hypercube. The set of vertices of this hypercube is $S_n = \{0, 1\}^n$. Define the mapping λ_n on this set as follows: for any vector $(s_1, \ldots, s_n) \in S_n$, let

$$\lambda_n(s_1, \ldots, s_n) = 1 + \sum_{t=1}^{n} s_t \cdot 2^{n-t}.$$

Then λ_n is a one-to-one mapping of S_n onto the set $\{1, \ldots, 2^n\}$.

Let us form the graph $D_n = (\{1, \ldots, 2^n\}, E_n)$, where for any $1 \leq i, j \leq 2^n$, $(i, j) \in E_n$ if and only if $\lambda_n^{-1}(i)$ is adjacent to $\lambda_n^{-1}(j)$. For any $j \in \{1, \ldots, 2^n\}$, let us denote by $J_j^{(n)}$ the set of all ancestors of j in D_n. Then $J_j^{(n)} \subseteq \{1, \ldots, 2^n\}$.

It is easy to see that for any $n \geq 2$ and integer $j \geq 1$,

$$|J_j^{(n)}| = n \text{ if } 1 \leq j \leq 2^n, \tag{4}$$

$$J_j^{(n+1)} = \begin{cases} J_j^{(n)} \cup \{j + 2^n\} & \text{if } 1 \leq j \leq 2^n, \\ \{l + 2^n : l \in J_{j-2^n}^{(n)}\} \cup \{j - 2^n\} & \text{if } 2^n < j \leq 2^{n+1}. \end{cases} \tag{5}$$

Now let $n \geq 2$ be an arbitrary integer and let $\mathcal{A} = (A, \Sigma) \in \mathfrak{U}_R$, $\mathcal{A}_j = (A_j, \Sigma^{(j)}) \in \mathfrak{U}_R$, $j = 1, \ldots, 2^n$. In addition, take a family Ψ of mappings

$$\Psi_{mj} : (A_1 \times, \ldots, \times A_{2^n})^{m-1} \times \Sigma_m \to \Sigma_m^{(j)}, \quad m \in R, \ 1 \leq j \leq 2^n.$$

It is said that the nondeterministic tree automaton $\mathcal{A} = (A, \Sigma)$ is a *cube-product* of $\mathcal{A}_j$, $j = 1, \ldots, 2^n$, with respect to Ψ if the following conditions are satisfied:

(a) $A = \prod_{j=1}^{2^n} A_j$

(b) for any $m \in R$, $\sigma \in \Sigma_m$ and $(a_{i1}, \ldots, a_{i2^n}) \in \prod_{j=1}^{2^n} A_j$, $i = 1, \ldots, m-1$, the mapping Ψ_{mj} is independent of the elements a_{ts}, $t = 1, \ldots, m-1$, if $s \notin J_j^{(n)}$,

(c) for any $m \in R$, $\sigma \in \Sigma_m$ and $((a_{11}, \ldots, a_{12^n}), \ldots, (a_{m-11}, \ldots, a_{m-12^n})) \in A^{m-1}$,

$$((a_{11}, \ldots, a_{12^n}), \ldots, (a_{m-11}, \ldots, a_{m-12^n}))\sigma^A =$$

$$(a_{11}, \ldots, a_{m-11})\sigma_1^{A_1} \times \ldots \times (a_{12^n}, \ldots, a_{m-12^n})\sigma_{2^n}^{A_{2^n}},$$

where

$$\sigma_j = \Psi_{mj}((a_{11}, \ldots, a_{12^n}), \ldots, (a_{m-11}, \ldots, a_{m-12^n}), \sigma), \qquad j = 1, \ldots, 2^n.$$

Since n determines the hypercube uniquely, we use the notation $\prod_{j=1}^{2^n} A_j(\Sigma, \Psi)$ for the cube-product just introduced.

Now we are ready to prove the following statement.

Theorem 2. *A system $\mathfrak{B}$ of nondeterministic tree automata from $\mathfrak{U}_R$ is isomorphically complete for $\mathfrak{U}_R$ with respect to the cube-product if and only if, for all $m \in R$ and $\mathbf{i} = (i_1, \ldots i_m) \in \{0,1\}^m$, $\mathfrak{B}$ contains a nondeterministic tree automata $\mathcal{A}^{(\mathbf{i})} = (A^{(\mathbf{i})}, \Sigma^{(\mathbf{i})})$ satisfying the following conditions:*

(6) *$A^{(\mathbf{i})}$ has two different elements $a_0^{(\mathbf{i})}$ and $a_1^{(\mathbf{i})}$,*

(7) *there exists a $\bar{\sigma}_{\mathbf{i}} \in \Sigma_m^{(\mathbf{i})}$ with $(a_{i_1}^{(\mathbf{i})}, \ldots, a_{i_{m-1}}^{(\mathbf{i})})\bar{\sigma}_{\mathbf{i}}^{A^{(\mathbf{i})}} \bigcap \{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} = \{a_{i_m}^{(\mathbf{i})}\}$,*

(8) *for all $u \in R$ and $\mathbf{s} = (s_1, \ldots, s_{u-1}) \in \{0,1\}^{u-1}$, there is a $\sigma_{\mathbf{i},\mathbf{s}} \in \Sigma_u^{(\mathbf{i})}$ for which $\{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} \subseteq (a_{s_1}^{(\mathbf{i})}, \ldots, a_{s_{u-1}}^{(\mathbf{i})})\sigma_{\mathbf{i},\mathbf{s}}^{A^{(\mathbf{i})}}$ provided that $u \neq 1$, and there is a $\sigma_{\mathbf{i}}^* \in \Sigma_1^{(\mathbf{i})}$ with $\{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} \subseteq \sigma_{\mathbf{i}}^{*A^{(\mathbf{i})}}$ if $1 \in R$ and $u = 1$.*

Proof. The necessity follows from the proof of Theorem 1. In order to prove the sufficiency, let us denote by W the set $\bigcup_{m \in R}\{0,1\}^m$ and let $W' = \{(i_1, \ldots, i_m) : (i_1, \ldots, i_m) \in W \ \& \ i_m = 0\}$. Let $|W'| = n$ and let γ denote a one-to-one mapping of the set $\{1, \ldots, n\}$ onto W'. Then, by our assumption on $\mathfrak{B}$, for any $p \in \{1, \ldots, n\}$, there exists a nondeterministic tree automaton $\mathcal{A}^{(\gamma(p))} = (A^{(\gamma(p))}, \Sigma^{(\gamma(p))}) \in \mathfrak{B}$ satisfying conditions (6), (7) and (8) with $\mathbf{i} = (i_1, \ldots, i_m) = \gamma(p)$, where $i_m = 0$. For the sake of simplicity, let us denote by 0, 1 the elements $a_0^{(\gamma(p))}$, $a_1^{(\gamma(p))}$, respectively, for all $p \in \{1, \ldots, n\}$.

Now consider the matrices $\mathbf{Q}^{(k)}$, $k = 2, 3, \ldots$, introduced in the proof of Theorem 1. In our argument we make use of some properties of these matrices. First let us observe that

$$(9) \qquad \mathbf{Q}^{(k+1)} = \begin{pmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{Q}^{(k)} & \mathbf{Q}^{(k)} \end{pmatrix}$$

where **0** and **1** denote the costant vectors of size 1×2^k with components 0 and 1, respectively. On the other hand, it can be seen (cf. [8] or [9]) that

(10) for any $k \geq 2$ and $1 \leq j \leq 2^k$, the $k+1$-tuples $(q_{tj}^{(k)}, q_{tj_1}^{(k)}, \ldots, q_{tj_k}^{(k)})$, $t = 1, \ldots, k$, are pairwise different where $\{j_1, \ldots, j_k\} = J_j^{(k)}$.

Now using (5), (9) and (10), it is easy to see that

(11) for any $k \geq 2$, $k > s \geq 1$, $1 \leq j \leq 2^k$, the k-tuples $(q_{tj_1}^{(k)}, \ldots, q_{tj_k}^{(k)})$, $t = s+1, \ldots, k$, are pairwise different where $\{j_1, \ldots, j_k\} = J_j^{(k)}$.

Now let $\mathcal{C} = (\{c_1, \ldots, c_r\}, \Sigma)$ be an arbitrary nondeterministic tree automaton from $\mathfrak{U}_R$. We prove that $\mathcal{C}$ is isomorphic to a subautomaton of a cube-product of nondeterministic tree automata from $\mathfrak{B}$.

For this purpose, let us denote by s the least positive integer with $n \leq 2^s$. Let $k = r + s$. Delete the first s rows of $\mathbf{Q}^{(k)}$. Then, by (9), the resulting matrix consists of 2^s copies of $\mathbf{Q}^{(r)}$ in its partitioned form. Let $\mathbf{Q}$ denote this matrix. For the sake of simplicity, let us denote by q_{tj}, $t = 1, \ldots, r$, $j = 1, \ldots, 2^k$, the elements of $\mathbf{Q}$. Then from (11) it follows that

(12) for any $1 \leq j \leq 2^k$, the k-tuples $(q_{tj_1}, \ldots, q_{tj_k})$, $t = 1, \ldots, r$, are pairwise different where $J_j^{(k)} = \{j_1, \ldots, j_k\}$.

Let us define the one-to-one mapping μ of $\{c_1, \ldots, c_r\}$ onto the set of the row vectors of $\mathbf{Q}$ by $\mu(c_i) = (q_{i1}, \ldots, q_{i2^k})$, $i = 1, \ldots, r$, and let $B = \{\mu(c_i) : i = 1, \ldots, r\}$.

Form the cube-product

$$\underbrace{A^{(\gamma(1))} \times \ldots \times A^{(\gamma(1))}}_{2^r \text{ times}} \times \ldots \times \underbrace{A^{(\gamma(n))} \times \ldots \times A^{(\gamma(n))}}_{2^r \text{ times}} \times$$

$$\times \underbrace{A^{(\gamma(1))} \times \ldots \times A^{(\gamma(1))}}_{2^k - n2^r \text{ times}}(\Sigma, \Psi)$$

in the following way. Observe that

$B \subseteq A =$

$$= \underbrace{A^{(\gamma(1))} \times \ldots \times A^{(\gamma(1))}}_{2^r \text{ times}} \times \ldots \times \underbrace{A^{(\gamma(n))} \times \ldots \times A^{(\gamma(n))}}_{2^r \text{ times}} \times \underbrace{A^{(\gamma(1))} \times \ldots \times A^{(\gamma(1))}}_{2^k - n2^r \text{ times}}.$$

Now let $1 \neq m \in R$, $\sigma \in \Sigma_m$, $(q_{i_t 1}, \ldots, q_{i_t 2^k}) \in B$, $t = 1, \ldots, m-1$, be arbitrary elements. Then $\mu(c_{i_t}) = (q_{i_t 1}, \ldots, q_{i_t 2^k})$, $t = 1, \ldots, m-1$. Let us assume that $(c_{i_1}, \ldots, c_{i_{m-1}})\sigma^C = \{c_{v_1}, \ldots, c_{v_l}\}$. Then $0 \leq l \leq r$. By the structure of $\mathbf{Q}$, there exists exactly one integer $d \in \{1, \ldots, 2^r\}$ such that for each $p \in \{1, \ldots, 2^s\}$, the following assertion is valid:

for all $t \in \{1, \ldots, r\}$, $q_{t,(p-1)2^r+d} = 0$ if and only if $t \in \{v_1, \ldots, v_l\}$.

On the other hand, let us observe that the column vectors of $\mathbf{Q}$ with indices $(p-1)2^r + d$, $p = 1, \ldots, 2^s$, are identical copies of some r-dimensional vector over $\{0,1\}$. Therefore, the vectors $(q_{i_1,(p-1)2^r+d}, \ldots, q_{i_{m-1},(p-1)2^r+d})$, $p = 1, \ldots, 2^s$, are the copies of an $(m-1)$-dimensional vector $(i'_1, \ldots, i'_{m-1})$ over $\{0,1\}$. Now let $\mathbf{i} = (i'_1, \ldots, i'_{m-1}, 0)$. Since $\mathbf{i} \in W'$, there exists one and only one $p_0 \in \{1, \ldots, n\}$ with $\gamma(p_0) = \mathbf{i}$. Let $j_0 = (p_0 - 1)2^r + d$. Then for each $j \in \{1, \ldots, 2^k\}$, the mapping Ψ_{mj} is defined by

$$\Psi_{mj}((q_{i_1 1}, \ldots, q_{i_1 2^k}), \ldots, (q_{i_{m-1} 1}, \ldots, q_{i_{m-1} 2^k}), \sigma) =$$

$$\begin{cases} \bar{\sigma}_{\mathbf{i}} & \text{if } j = j_0, \\ \sigma_{\gamma(p),(i'_1,\ldots,i'_{m-1})} & \text{if } j \neq j_0 \text{ and } (p-1)2^r < j \leq p2^r \\ & \text{for some } p \in \{1, \ldots, 2^s\}. \end{cases}$$

In all other cases when $1 \neq m \in R$, Ψ_{mj} can be defined arbitrarily in accordance with the definition of the cube-product.

Now let us suppose that $1 \in R$. Let $\sigma \in \Sigma_1$ be arbitrary and $\sigma^C = \{c_{v_1}, \ldots, c_{v_l}\}$. Then there exists again exactly one integer $d \in \{1, \ldots, 2^r\}$ such that the following statement holds for each $p \in \{1, \ldots, 2^s\}$:

for all $t \in \{1, \ldots, r\}$, $q_{t,(p-1)2^r+d} = 0$ if and only if $t \in \{v_1, \ldots, v_l\}$.

In this case $(0) \in W'$, and so, there is one and only one $p_0 \in \{1, \ldots, n\}$ with $\gamma(p_0) = (0)$. Let $j_0 = (p_0 - 1)2^r + d$. For each $j \in \{1, \ldots, 2^k\}$, let us define the mapping Ψ_{1j} as follows.

$$\Psi_{1j}(\sigma) = \begin{cases} \bar{\sigma}_{(0)} & \text{if } j = j_0, \\ \sigma^*_{\gamma(p)} & \text{if } j \neq j_0 \text{ and } (p-1)2^r < j \leq p2^r \text{ for some } p \in \{1, \ldots, 2^s\}. \end{cases}$$

By (12), the mappings Ψ_{mj}, $m \in R$, $1 \leq j \leq 2^k$, are well-defined. On the other hand, it is easy to see that the mapping μ is an isomorphism of C onto that subautomaton of the defined cube-product which is determined by the set B. Therefore, $\mathfrak{B}$ is isomorphically complete for $\mathfrak{U}_R$ with respect to the cube-product. This ends the proof of Theorem 2.

Remark. In the case $R = \{2\}$ we obtain a characterization of the isomorphically complete systems for the class of nondeterministic automata with respect to the cube-product. Therefore, the main result of [5] can be obtain as a corollary of Theorem 2.

Notice that the necessary and sufficient conditions stated by Theorem 1 and Theorem 2 are the same which gives us the following corollary.

Corollary. *A system of nondeterministic tree automata from $\mathfrak{U}_R$ is isomorphically complete for $\mathfrak{U}_R$ with respect to the general product if and only if it is isomorphically complete for $\mathfrak{U}_R$ with respect to the cube-product.*

References

[1] Z. Ésik, Homomorphically complete classes of automata with respect to the α_2-product, *Acta Sci. Math.*, **48** (1985), 135-141.

[2] F. Gécseg, Composition of automata, *Automata, Languages and Programming,2nd Colloquium*, Saarbrücken, 1974, Lecture Notes in Computer Science (Springer-Verlag, Berlin Heidelberg New York Tokyo) **14** (1974), 351-363.

[3] F. Gécseg, *Products of Automata*, Springer-Verlag, Berlin Heidelberg New York Tokyo (1986).

[4] F. Gécseg and B. Imreh, On completeness of nondeterministic automata, *Acta Math. Hungarica*, **68** (1995), 151-159.

[5] F. Gécseg and B. Imreh, On the cube-product of nondeterministic automata *Acta Sci. Math.*, to appear.

[6] F. Gécseg and M. Steinby, *Tree automata,* Akadémiai Kiadó, Budapest, 1984.

[7] V. M. Glushkov, Abstract theory of automata, *Uspekhi Mat. Nauk*, **16:5 101** (1961), 3-62 (in Russian).

[8] B. Imreh, On complete systems of automata, in: *Proc. of the 2nd International Colloquium on Words, Languages and Combinatorics*, Kyoto, 1992, World Scientific (Singapore-New Jersey-London-Hong Kong), 1994, 207-215.

[9] B. Imreh, On a special composition of tree automata, *Acta Cybernetica* **10** (1992), 237-242.

[10] M. Steinby, On the structure and realizations of tree automata, in *Second Coll. sur les Arbres an Algèbre et en Programmation* Lille, 1979, 235-248.

Received October 3, 1994

Acta Cybernetica, Vol. 12, No. 1, Szeged, 1995

Some Remarks on Directable Automata*

B. Imreh† M. Steinby‡

Abstract

A finite automaton is said to be directable if there exists a word, a directing word, which takes the automaton from every state to the same state. After some general remarks on directable automata and their directing words we present methods for testing the directability of an automaton and for finding the least congruence of an automaton which yields a directable quotient automaton. A well-known conjecture by J. černý claims that any n-state directable automaton has a directing word of length $\leq (n-1)^2$, but the best known upper bounds are of the order $o(n^3)$. However, for special classes of automata lower bounds can be given. We consider a generalized form of černý's conjecture proposed by J.-E. Pin for the classes of commutative, definite, reverse definite, generalized definite and nilpotent automata. We also establish the inclusion relationships between these classes within the class of directable automata.

1 Introduction

A finite automaton is *directable* if there is an input word, a *directing word*, which takes the automaton from every state to the same state. (Directable automata and directing words are also called *synchronizable automata* and *synchronizing words*, respectively.) In this paper we discuss a variety of questions concerning directable automata and their directing words. After the preliminaries and general remarks of Sections 2 and 3, we present in Section 4 a method for testing the directability of an automaton. The algorithm is based on the mergeability relation of states, and for computing effectively this relation the inverted transition table of the automaton is used. A congruence of an automaton is *directing* if the corresponding quotient automaton is directable. Such congruences were considered (under a different name) by Ito and Duske [ItD83] who noted that every automaton has a

*This work has been supported by the Finnish Academy of Sciences, the Turku University Foundation and the Hungarian National Foundation for Scientific Research, Grant 2035.

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

‡Department of Mathematics, University of Turku, SF-20500 Turku, Finland

minimal directing congruence. It gives the largest directable homomorphic image of the automaton. In Section 5 we describe an algorithm for computing the smallest directing congruence.

Černý [Čer64] conjectured that an n-state directable automaton must have a directing word of length $\leq (n-1)^2$. So far, this has been neither proved nor disproved, and the conjecture remains the main problem in the area. The best known upper bounds for the length of the shortest directing word are of the order $O(n^3)$ (cf. [Sta69,ČPR71,Pin78], for example). On the other hand, even better bounds than $(n-1)^2$ can be given for some special classes of automata [Pin79]. Recently, Rystsov [Rys94] proved that for commutative automata the exact bound is $n-1$. In Section 6 we give a short elementary proof of a generalized form of Rystsov's result. The generalization corresponds to an extension of Černý's conjecture proposed by Pin [Pin78]. An automaton is *r- directable*, for some $r \geq 1$, if it has an *r- directing word* which takes the automaton from every state to one of r states which depend on the word only. Pin's conjecture claims that if $1 \leq r \leq n$, then any n-state r-directable automaton has an *r-directing word of length* $\leq (n-r)^2$; for $r = 1$ this is Černý's conjecture. In Section 7 we consider the directability and the directing words of definite, reverse definite, generalized definite and nilpotent automata. In each case we can give exact bounds for the lengths of the minimal r-directing words. We also consider the inclusions and the intersections between these classes when restricted to directable automata. In particular, it is noted that every directable generalized definite automaton is definite.

2 Preliminaries

Although most of our notation is quite standard, some of it will be explained here along with some general notions we shall need. The cardinality of a set A is denoted by $|A|$. If $f : A \to B$ is a mapping, the value $f(a)$ of an element $a \in A$ is often denoted by af. Similarly, we may write Hf for $f(H) = \{af : a \in H\}$ when $H \subseteq A$. The composition of two mappings $f : A \to B$ and $g : B \to C$ is the mapping $fg : A \to C$, $a \mapsto (af)g$, and the product of two relations $\Theta \subseteq A \times B$ and $\rho \subseteq B \times C$ is the relation

$$\Theta\rho = \{(a,c) \in A \times C : (\exists b \in B)\ a\Theta b,\ b\rho c\}$$

from A to C; that $(a,b) \in \Theta$ holds is also expressed by writing $a\Theta b$. The set of equivalence relations on a set A is denoted by $\mathrm{Eq}(A)$. If $\Theta \in \mathrm{Eq}(A)$, the Θ-class of an element a of A is denoted by $a\Theta$, and the set of all such Θ-classes, the quotient set with respect to Θ, is denoted by A/Θ. For any set A, $\mathrm{Eq}(A)$ contains the diagonal relation $\Delta_A = \{(a,a) : a \in A\}$ and the universal relation $\nabla_A = A \times A$. These are the smallest and the greatest element, respectively, of the complete lattice $(\mathrm{Eq}(A), \subseteq)$ (cf. [BuS81], for example).

In this paper X is always a finite nonempty alphabet. The set of all (finite) words over X, also called *X- words*, is denoted by X^* and the empty word by ε.

For the length of a word w we use the notation $\lg(w)$. For any integer $k \geq 0$, X^k denotes the set of all X-words of length k, and also let

$$X^{<k} = \{w \in X^* : \lg(w) < k\},$$

$$X^{\leq k} = \{w \in X^* : \lg(w) \leq k\},$$

$$X^{\geq k} = \{w \in X^* : \lg(w) \geq k\}.$$

The prefix of length k and the suffix of length k of a word w are denoted by $\mathrm{pref}_k(w)$ and $\mathrm{suff}_k(w)$, respectively.

An *automaton*, or an *X-automaton* - to be more specific, is a system $\mathcal{A} = (A, X, \delta)$, where A is the finite nonempty set of *states*, X is the *input alphabet*, and $\delta : A \times X \to A$ is the *transition function*. The transition function is extended to $A \times X^*$ in the usual way. Each word $w \in X^*$ defines then a unary operation $w : A \to A$, $a \mapsto \delta(a, w)$, on the state set, and the state $\delta(a, w)$ into which the input word w takes $\mathcal{A}$ from state a is usually denoted by aw. This notation is extended also to subsets of A : if $H \subseteq A$, then $Hw = \{aw : a \in H\}$.

Subsets of X^* are called *X-languages*, or just *languages*. An *X-recognizer* is a system $\mathbf{A} = (A, X, \delta, a_0, F)$ which consists of an X-automaton (A, X, δ), an *initial state* a_0 $(\in A)$ and a set F $(\subseteq A)$ of *final states*. We say that $\mathbf{A}$ is *based* on the X-automaton (A, X, δ). The *language recognized* by $\mathbf{A}$ is $L(\mathbf{A}) = \{w \in X^* : a_0 w \in F\}$. An X-language is *recognizable*, or *regular*, if it is recognized by an X-automaton. The set of recognizable X-languages is denoted by $\mathrm{Rec}(X)$.

Next we define some basic algebraic notions for automata. These could be taken directly from general algebra by construing automata as unary algebras, but we use the usual definition of an automaton. Nevertheless, for in- depth treatments of these ideas one should consult texts on universal algebra such as [BuS81], for example. An X-automaton (B, X, η) is a *subautomaton* of the X-automaton $\mathcal{A} = (A, X, \delta)$ if $B \subseteq A$ and $\eta(b, x) = \delta(b, x)$ for all $b \in B$ and $x \in X$. An equivalence $\Theta \in \mathrm{Eq}(A)$ is a *congruence* of $\mathcal{A} = (A, X, \delta)$ if for all $a, b \in A$ and $x \in X$, $a\Theta b$ implies $ax\Theta bx$. The set of congruences of $\mathcal{A}$ is denoted by $\mathrm{Con}(\mathcal{A})$. It is well-known that $\mathrm{Con}(\mathcal{A})$ forms a sublattice of the lattice $(\mathrm{Eq}(A), \subseteq)$. Moreover, $\Delta_A, \nabla_A \in \mathrm{Con}(\mathcal{A})$. If $\Theta \in \mathrm{Con}(\mathcal{A})$, the *quotient automaton* $\mathcal{A}/\Theta = (A/\Theta, X, \delta_\Theta)$ is defined so that $\delta_\Theta(a\Theta, x) = \delta(a, x)\Theta$ for all $a\Theta \in A/\Theta$ and $x \in X$. A *morphism* of X-automata from $\mathcal{A} = (A, X, \delta)$ to $\mathcal{B} = (B, X, \eta)$ is a mapping $\varphi : A \to B$ such that for all $a \in A$ and $x \in X$, $\delta(a, x)\varphi = \eta(a\varphi, x)$. We write $\varphi : \mathcal{A} \to \mathcal{B}$ to indicate that $\varphi : A \to B$ is a morphism. An *epimorphism* is a surjective morphism. If there exists an epimorphism $\varphi : \mathcal{A} \to \mathcal{B}$, then $\mathcal{B}$ is an *image* of $\mathcal{A}$. The *direct product* of $\mathcal{A} = (A, X, \delta)$ and $\mathcal{B} = (B, X, \eta)$ is the X-automaton $\mathcal{A} \times \mathcal{B} = (A \times B, X, \gamma)$ in which $\gamma((a, b), x) = (\delta(a, x), \eta(b, x))$ for any $(a, b) \in A \times B$ and $x \in X$.

If $\mathbf{K}$ is a class of automata, then $\mathbf{K}(X)$ denotes the class of X-automata belonging to $\mathbf{K}$. A nonempty class of X-automata is called a *variety of X-automata* if it is closed under the operations of forming subautomata, images and (finite) direct products.

3 Directable automata and directing words

A word $w \in X^*$ is a *directing word* of an X- automaton $\mathcal{A} = (A, X, \delta)$ if it takes $\mathcal{A}$ from every state to the same state, i.e. if $|Aw| = 1$, and we call $\mathcal{A}$ *directable* if it has a directing word. The set of directing words of $\mathcal{A}$ is denoted by $\mathrm{DW}(\mathcal{A})$. The class of all directable automata is denoted by **Dir**.

It is obvious that every directing word of an X- automaton $\mathcal{A}$ is a directing word of every subautomaton of $\mathcal{A}$, too. If $\varphi : \mathcal{A} \to \mathcal{B}$ is an epimorphism from $\mathcal{A} = (A, X, \delta)$ onto $\mathcal{B} = (B, X, \eta)$, then $\mathrm{DW}(\mathcal{A}) \subseteq \mathrm{DW}(\mathcal{B})$. Indeed, let w be a directing word of $\mathcal{A}$. If $b, b' \in B$, then $b = a\varphi$ and $b' = a'\varphi$, for some $a, a' \in A$, and therefore

$$\eta(b, w) = \eta(a\varphi, w) = \delta(a, w)\varphi = \delta(a', w)\varphi = \ldots = \eta(b', w).$$

Similarly, $u \in \mathrm{DW}(\mathcal{A})$ and $v \in \mathrm{DW}(\mathcal{B})$ imply that $uv \in \mathrm{DW}(\mathcal{A} \times \mathcal{B})$. These observations lead to the following conclusion.

Remark 3.1. *For any alphabet X, $\mathbf{Dir}(X)$ is a variety of X-automata.*

If w is a directing word of an X-automaton $\mathcal{A}$, then so is uwv for any X-words u and v. This yields the next remark.

Remark 3.2. *For any X-automaton $\mathcal{A}$, $X^*\mathrm{DW}(\mathcal{A})X^* = \mathrm{DW}(\mathcal{A})$.*

With any X-automaton $\mathcal{A} = (A, X, \delta)$ we associate an X-automaton $\mathcal{A}_d = (B, X, \eta)$, where $B = \{Aw : w \in X^*\}$ and $\eta(Aw, x) = Awx$ for all $w \in X^*$ and $x \in X$. This $\mathcal{A}_d$ is the part of the usual subset automaton of $\mathcal{A}$ accessible from state A. For any $w \in X^*$, $\eta(A, w) = Aw$. Hence w is a directing word of $\mathcal{A}$ iff $\eta(A, w)$ is a singleton. This means that $\mathrm{DW}(\mathcal{A})$ is recognized by the X-recognizer (B, X, η, A, F), where $F = \{Aw : w \in X^*, |Aw| = 1\}$, and we can state the following conclusion.

Remark 3.3. *For any X-automaton $\mathcal{A}$, $\mathrm{DW}(\mathcal{A}) \in \mathrm{Rec}(X)$.*

If $\mathcal{A}$ is a directable automaton, let $\mathrm{d}(\mathcal{A}) = \min\{\lg(w) : w \in \mathrm{DW}(\mathcal{A})\}$, and for any $n \geq 1$, we define the number

$$\mathrm{d}(n) = \max\{\mathrm{d}(\mathcal{A}) : \mathcal{A} \in \mathbf{Dir}, |A| = n\}.$$

Černý's conjecture may now be formulated as the claim that $\mathrm{d}(n) = (n-1)^2$ for all $n \geq 1$. In [ČPR71] it was shown that the hypothesis holds for $n \leq 5$. For the general case only upper bounds of the order $O(n^3)$ are known (cf. [Sta69, ČPR71,Pin78], for example). On the other hand, there are examples showing that $\mathrm{d}(n) \geq (n-1)^2$ for all $n \geq 1$ (cf. [Čer64,Sta69]). We consider some modifications of the problem concerning Černý's conjecture. First of all, the question may be restricted to concern some subclass of **Dir**. If **K** is some class of automata, we set

$$\mathrm{d}_{\mathbf{K}}(n) = \max\{\mathrm{d}(\mathcal{A}) : \mathcal{A} \in \mathbf{K} \cap \mathbf{Dir}, |A| = n\}.$$

Pin [Pin79] has shown that $\mathrm{d}_{\mathbf{K}}(n) = (n-1)^2$ for all prime n when $\mathbf{K}$ is the class of automata in which some input letter defines a circular permutation. As we shall see, there are even classes $\mathbf{K}$ for which $\mathrm{d}_{\mathbf{K}}(n) < (n-1)^2$.

For any $r \geq 1$, we call $w \in X^*$ an *r- directing word* of an X-automaton $\mathcal{A} = (A, X, \delta)$, if $|Aw| \leq r$. Let $\mathrm{DW}(\mathcal{A}, r)$ denote the set of r-directing words of $\mathcal{A}$. If $|A| = n$, then

$$X^* = \mathrm{DW}(\mathcal{A}, n) \supseteq \mathrm{DW}(\mathcal{A}, n-1) \supseteq \ldots \supseteq \mathrm{DW}(\mathcal{A}, 1) = \mathrm{DW}(\mathcal{A}).$$

It is clear that Remarks 3.2 and 3.3 apply also to the languages $\mathrm{DW}(\mathcal{A}, r)$. We say that $\mathcal{A}$ is *r-directable* if $\mathrm{DW}(\mathcal{A}, r) \neq \emptyset$. For each $r \geq 1$, let $\mathbf{Dir}_r$ denote the class of r-directable automata. Clearly, $\mathbf{Dir} = \mathbf{Dir}_1 \subset \mathbf{Dir}_2 \subset \ldots$. Note that for $r \geq 2$ and any X, $\mathbf{Dir}_r(X)$ is not a variety of X-automata; it is not closed under direct products.

For any $r \geq 1$ and $\mathcal{A} \in \mathbf{Dir}_r$, let $\mathrm{d}(\mathcal{A}, r) = \min\{\lg(w) : w \in \mathrm{DW}(\mathcal{A}, r)\}$, and for $1 \leq r \leq n$, let

$$\mathrm{d}(n, r) = \max\{\mathrm{d}(\mathcal{A}, r) : \mathcal{A} \in \mathbf{Dir}_r, |A| = n\}.$$

In [Pin78] Pin put forward the following generalization of Černý's conjecture which we call *Pin's conjecture*: $\mathrm{d}(n, r) = (n-r)^2$ for all $1 \leq r \leq n$. For any class $\mathbf{K}$ of automata, we write

$$\mathrm{d}_{\mathbf{K}}(n, r) = \max\{\mathrm{d}(\mathcal{A}, r) : \mathcal{A} \in \mathbf{K} \cap \mathbf{Dir}_r, |A| = n\},$$

and one can again consider modifications of Pin's conjecture which apply to the numbers $d_{\mathbf{K}}(n, r)$ for various classes $\mathbf{K}$.

4 Testing for directability

Let $\mathcal{A} = (X, A, \delta)$ be an automaton, and suppose $|A| = n$ and $|X| = m$. To find out whether $\mathcal{A}$ is directable or not by constructing the state set $\{Aw : w \in X^*\}$ of $\mathcal{A}_d$ can be quite time-consuming: there may be almost 2^n sets to consider, and for each new set Aw one should form all sets Awx $(x \in X)$ and compare them with the previously found sets. If no essential improvements can be found, the worst case estimate for this method is at least of the order $O(m \cdot 2^n)$. Ito and Duske [ItD83] noted that the directability of $\mathcal{A}$ can be tested by applying an input word t which contains all words over X of length $\mathrm{d}(n)$ as subwords; obviously $\mathcal{A} \in \mathbf{Dir}$ iff $|At| = 1$. They show how one can construct such a word t, but the mere length $m^{\mathrm{d}(n)} + \mathrm{d}(n) - 1$ of the word renders the test unpractical even for small values of n and m. If we assume that Černý's conjecture holds, which is the best we can hope for, the length of the test word will be of the order $O(m^{(n-1)^2})$. We present here a simple $O(m \cdot n^2)$-algorithm for solving the directability problem.

For any $k \geq 0$, the relation $\mu_{\mathcal{A}}(k)$ of *k-mergeability* on the state set A of $\mathcal{A}$ is defined so that for $a, b \in A$, $(a, b) \in \mu_{\mathcal{A}}(k)$ iff $aw = bw$ for some $w \in X^{\leq k}$. Two

states a and b are *mergeable* if they are k- mergeable for some $k \geq 0$. We denote $\mu_{\mathcal{A}} = \bigcup_{k \geq 0} \mu_{\mathcal{A}}(k)$. It is well-known (cf. [Sta69]) that an automaton is directable iff all pairs of its states are mergeable. This and some other obvious facts about the relations $\mu_{\mathcal{A}}(k)$ and $\mu_{\mathcal{A}}$ are stated in the following proposition.

Proposition 4.1. *Let $\mathcal{A} = (A, X, \delta)$ be an n-state automaton.*

(a) *$\mathcal{A}$ is directable iff $\mu_{\mathcal{A}} = \nabla_A$.*

(b) *The relations $\mu_{\mathcal{A}}(k)$ are reflexive and symmetric ($k \geq 0$).*

(c) $\Delta_A = \mu_{\mathcal{A}}(0) \subseteq \mu_{\mathcal{A}}(1) \subseteq \ldots \subseteq \mu_{\mathcal{A}}$.

(d) The relations $\mu_{\mathcal{A}}(k)$ can be computed as follows:

1. $\mu_{\mathcal{A}}(0) = \Delta_A$;
2. $\mu_{\mathcal{A}}(k) = \mu_{\mathcal{A}}(k-1) \cup \{(a,b) : (\exists x \in X)\ (ax, bx) \in \mu_{\mathcal{A}}(k-1)\}$ *for* $k > 0$.

(e) *If $\mu_{\mathcal{A}}(k) = \mu_{\mathcal{A}}(k-1)$ for some $k > 0$, then $\mu_{\mathcal{A}}(k) = \mu_{\mathcal{A}}(k+1) = \ldots = \mu_{\mathcal{A}}$.*

(f) $\Delta_A = \mu_{\mathcal{A}}(0) \subset \mu_{\mathcal{A}}(1) \subset \ldots \subset \mu_{\mathcal{A}}(k) = \mu_{\mathcal{A}}(k+1) = \mu_{\mathcal{A}}$ *for some k, where* $0 \leq k \leq \binom{n}{2}$.

Proposition 4.1 suggests that the directability of $\mathcal{A}$ can be tested by computing successively $\mu_{\mathcal{A}}(0), \mu_{\mathcal{A}}(1), \mu_{\mathcal{A}}(2), \ldots$ until $\mu_{\mathcal{A}}(k) = \mu_{\mathcal{A}}(k-1)$. The most direct way of doing this leads to an $O(m \cdot n^4)$- algorithm, but by organizing the work better, we get an algorithm which operates in time $O(m \cdot n^2)$. A great part of the saving is achieved by using the inverse transition table of $\mathcal{A}$ instead of the transition table itself. Also, we do not form explicitly each $\mu_{\mathcal{A}}(k)$ although they appear in the sequence of relations that are computed.

The algorithm employs two auxiliary data structures, a Boolen $n \times n$-matrix $\mathbf{M}$ and a list *NewPair* of pairs of states. To simplify the notation, we assume that $A = \{1, 2, \ldots, n\}$. Then $\mathbf{M}[i,j] = 1$ means that the pair i, j ($\in A$) is known to be mergeable. Since it suffices to consider just the pairs (i,j), where $1 \leq i < j \leq n$, we actually need just the upper part of $\mathbf{M}$. A pair appears in *NewPair* when i and j have found to be mergeable, but this fact has not yet been used for finding further mergeable pairs. The *inverted transition table*

$$\mathbf{I} = (\mathbf{I}[a,x])_{a \in A, x \in X}$$

is defined so that $\mathbf{I}[a,x] = \{i \in A : ix = a\}$, for any $a \in A, x \in X$. The steps of the algorithm are as follows.

Step 1. (Initialize $\mathbf{M}$ and *NewPair*) $\mathbf{M}[i,j] := 0$ for all $1 \leq i < j \leq n$, and $NewPair := \varepsilon$ (the empty list).

Step 2. Form the inverted transition table $\mathbf{I}$.

Step 3. Find all pairs (a,x) ($\in A \times X$) for which $|\mathbf{I}[a,x]| > 1$. For every such (a,x) consider each pair $i, j \in \mathbf{I}[a,x]$ with $i < j$. If $\mathbf{M}[i,j] = 0$, let $\mathbf{M}[i,j] := 1$ and append (i,j) to *NewPair*.

Step 4. Until $NewPair = \varepsilon$ do the following. Delete the first pair from *NewPair*; suppose it is (a, b). From $\mathbf{I}$ find all pairs (i, j), $i < j$, such that for some $x \in X$, $i \in \mathbf{I}[a, x]$ and $j \in \mathbf{I}[b, x]$, or $i \in \mathbf{I}[b, x]$ and $j \in \mathbf{I}[a, x]$. If $\mathbf{M}[i, j] = 0$, let $\mathbf{M}[i, j] := 1$ and append (i, j) to *NewPair*.

Step 5. If $\mathbf{M}[i, j] = 1$ whenever $1 \leq i < j \leq n$, then $\mathcal{A}$ is directable, otherwise not.

Step 1 takes time $O(n^2)$. If $\mathcal{A}$ is given as a transition table, Step 2 can be carried out in time $O(m \cdot n)$. In Step 3 one has to consider for each of the m input symbols altogether $n(n-1)/2$ pairs (i, j), therefore the step takes time $O(m \cdot n^2)$. In Step 4 each pair (i, j) will be considered at most once for each $x \in X$, and this happens when the pair (a, b) for which $\{ix, jx\} = \{a, b\}$ is removed from *NewPair*. Hence, the time bound is $O(m \cdot n^2)$. Since Step 5 can be carried out in time $O(n^2)$, the time bound for the whole algorithm is $O(m \cdot n^2)$.

5 Directing conguences

We call a congruence ρ of an automaton $\mathcal{A} = (A, X, \delta)$ *directing* if the quotient automaton $\mathcal{A}/\rho$ is directable. The set of directing congruences of $\mathcal{A}$ is denoted by $\mathrm{Con}_d(\mathcal{A})$. The following observations are easily verified.

Lemma 5.1. *For any automaton $\mathcal{A}$, $\mathrm{Con}_d(\mathcal{A})$ is a filter of the congruence lattice $\mathrm{Con}(\mathcal{A})$, i.e. (1) $\mathrm{Con}_d(\mathcal{A}) \neq \emptyset$, (2) $\Theta \subseteq \rho$, $\Theta \in \mathrm{Con}_d(\mathcal{A})$ and $\rho \in \mathrm{Con}(\mathcal{A})$ imply $\rho \in \mathrm{Con}_d(\mathcal{A})$, and (3) $\Theta \cap \rho \in \mathrm{Con}_d(\mathcal{A})$ for all $\Theta, \rho \in \mathrm{Con}_d(\mathcal{A})$.*

Corollary 5.2. *Every automaton $\mathcal{A}$ has a unique minimal directing congruence, which we denote by $\rho_{\mathcal{A}}$, $\mathrm{Con}_d(\mathcal{A})$ is the principal filter $[\rho_{\mathcal{A}})$ of $\mathrm{Con}(\mathcal{A})$, and every directable image of $\mathcal{A}$ is an image of $\mathcal{A}/\rho_{\mathcal{A}}$.*

Let $\rho \in \mathrm{Eq}(A)$. We call two states a and b of the automaton $\mathcal{A} = (A, X, \delta)$ *ρ-mergeable* if $(aw, bw) \in \rho$ for some $w \in X^*$. The following obvious lemma shows that our directing congruences are the same as the 'cofinal congruences' of Ito and Duske [ItD83].

Lemma 5.3. *A congruence ρ of $\mathcal{A}$ is directing iff all pairs of states of $\mathcal{A}$ are ρ-mergeable.*

For computing the minimal directing congruence we present a sharper condition for a congruence to be directing. Since any two mergeable states are ρ-mergeable for every congruence ρ, it suffices to consider the nonmergeable pairs of states.

For any automaton $\mathcal{A} = (A, X, \delta)$, let $G_{\mathcal{A}} = (V, E)$ be the directed graph defined as follows. The vertex set $V = \{\{a, b\} : a, b \in A, (a, b) \notin \mu_{\mathcal{A}}\}$ consists of all unordered pairs of nonmergeable states of $\mathcal{A}$. The edge set is $E = \{(\{a, b\}, \{ax, bx\}) : \{a, b\} \in V, x \in X\}$. Note that $\{ax, bx\} \in V$ if $\{a, b\} \in V$ and $x \in X$. It is clear that a congruence which identifies all pairs in V is directing, but it actually suffices to consider a subset of V, the *trap* T of $G_{\mathcal{A}}$ which is the union of (the vertex sets of) all strongly connected components of $G_{\mathcal{A}}$ from which no edges lead outside the component (cf. [DDK85]).

Lemma 5.4. *A congruence ρ of an automaton $\mathcal{A} = (A, X, \delta)$ is directing iff $a\rho b$ for every pair $\{a, b\}$ which belongs to the trap T of $G_{\mathcal{A}}$.*

Proof. For any pair c, d $(\in A)$ of nonmergeable states there is a word w $(\in X^*)$ such that $\{cw, dw\} \in T$. Hence ρ is directing if it satisfies the condition of the lemma. Suppose now that $\rho \in \mathrm{Con}_d(\mathcal{A})$ and consider any pair $\{a, b\} \in T$. By Lemma 5.3 there is a word w such that $(aw, bw) \in \rho$. Since $\{aw, bw\}$ is in the same strongly connected component as $\{a, b\}, \{awu, bwu\} = \{a, b\}$ for some $u \in X^*$. This shows that also $(a, b) \in \rho$.

For any $a, b \in A$ $(a \neq b)$, let $\Theta(a, b)$ be the principal congruence generated by the pair (a, b) (cf. [BuS81]). The last part of the previous proof shows also that $\Theta(a, b) = \Theta(c, d)$ whenever $\{a, b\}$ and $\{c, d\}$ are in the same strongly connected component of $G_{\mathcal{A}}$. Although it will not be used here, we note that Lemma 5.4 yields the following description of the minimal directing congruence.

Corollary 5.5. *For any automaton $\mathcal{A} = (A, X, \delta)$,*

$$\rho_{\mathcal{A}} = \Theta(a_1, b_1) \vee \ldots \vee \Theta(a_k, b_k),$$

for any set $\{\{a_1, b_1\}, \ldots, \{a_k, b_k\}\}$ of representatives of the strongly connected components which form the trap of $G_{\mathcal{A}}$.

Since the reflexive closure $\tau_{\mathcal{A}} = \Delta_A \cup \{(a, b) : \{a, b\} \in T\}$ of the relation corresponding to the trap T of $G_{\mathcal{A}}$ is invariant with respect the state transitions of $\mathcal{A}$, then so is its transitive closure $\tau_{\mathcal{A}}^+$. Since $\tau_{\mathcal{A}}^+$ is the equivalence generated by the pairs in the trap, this means by Lemma 5.4 that $\tau_{\mathcal{A}}^+ = \rho_{\mathcal{A}}$. These observations lead to the following algorithm for finding the minimal directing congruence for a given automaton $\mathcal{A} = (A, X, \delta)$.

Step 1. Compute $\mu_{\mathcal{A}}$ using the method described in Section 4.

Step 2. Form the graph $G_{\mathcal{A}} = (V, E)$; the vertex set is obtained from $\mu_{\mathcal{A}}$.

Step 3. Compute the strongly connected components forming the trap T of $G_{\mathcal{A}}$ using the algorithm of [DDK85].

Step 4. Form the relation $\tau_{\mathcal{A}}$ and compute the transitive closure; $\tau_{\mathcal{A}}^+ = \rho_{\mathcal{A}}$.

We know that the computation of $\mu_{\mathcal{A}}$ takes time $O(m \cdot n^2)$. The vertex set V is then obtained in time $O(n^2)$, and computing the set E of edges can be done in time $O(m \cdot n^2)$. In [DDK85] Tarjan's algorithm [Tar72] for computing the strongly connected components of a directed graph is modified to yield the trap. The algorithm works in time $O(\nu + e)$, where ν is the number of vertices and e is the number of edges. In the present case $\nu \leq n(n-1)/2$ and $e \leq m \cdot n(n-1)/2$, for $n = |A|$ and $m = |X|$. Hence, also Step 3 can be carried out in time $O(m \cdot n^2)$. Step 4 takes time $O(n^3)$ if we use Warshall's algorithm (cf. [AHU83], for example) for computing the transitive closure. The total time used by algorithm is therefore bounded by $O(m \cdot n^2 + n^3)$.

6 Directable commutative automata

An automaton $\mathcal{A} = (A, X, \delta)$ is called *commutative* if $axy = ayx$ for all $a \in A$ and $x, y \in X$. Let **Com** denote the class of commutative automata. Rystsov [Rys94] has shown that $\mathrm{d}_{\mathbf{Com}}(n) = n - 1$ for every $n \geq 1$. We give a simple proof for a generalization of this fact. The generalization corresponds to Pin's conjecture.

Proposition 6.1. $\mathrm{d}_{\mathbf{Com}}(n, r) = n - r$ *whenever* $1 \leq r \leq n$.

Proof. Suppose $\mathcal{A} = (A, X, \delta)$ is commutative and r-directable, where $1 \leq r \leq n = |A|$. Let $w = x_1 \dots x_m$ $(x_i \in X)$ be an r-directing word of $\mathcal{A}$ of minimal length. The commutativity of $\mathcal{A}$ implies that $Auv = (Av)u \subseteq Au$ for all $u, v \in X^*$. Hence $A \supseteq Ax_1 \supseteq Ax_1x_2 \supseteq \dots \supseteq Aw$. All of these inclusions must be proper as $Ax_1 \dots x_{i-1} = Ax_1 \dots x_{i-1}x_i$, for some $1 \leq i \leq m$, would imply that $Ax_1 \dots x_{i-1}x_{i+1} \dots x_m = Aw$, contradicting the assumption that w is of minimal length. Therefore

$$n = |A| > |Ax_1| > \dots > |Ax_1 \dots x_{m-1}| > r,$$

and this implies that $m \leq n - r$. To see that equality is possible in all cases, it suffices to consider the automata $\mathcal{A}(n, X) = (\{1, \dots, n\}, X, \delta)$, where $n \geq 1$, X is any alphabet and $\delta(i, x) = \min\{i + 1, n\}$ for all $i \in \{1, \dots, n\}$ and $x \in X$.

7 Definiteness, nilpotency and directability

Let $k \geq 0$. An automaton $\mathcal{A} = (A, X, \delta)$ is *weakly k-definite* if $aw = bw$ for all $w \in X^k$ and all $a, b \in A$, and it is *definite* if it is weakly k-definite for some k. If $\mathcal{A}$ is definite and k is the smallest number for which it is weakly k-definite, then $\mathcal{A}$ is *k-definite* [Kle56,PRS63]. Let **Def** denote the class of all definite automata.

It is clear that every definite automaton is directable. Moreover, if an X-automaton $\mathcal{A}$ is weakly *k-definite*, then $\mathrm{DW}(\mathcal{A}) = P \cup X^{\geq k}$ for some $P \subseteq X^{<k}$. In [PRS63] it was shown that an n-state definite automaton is k- definite for some $k \leq n - 1$. This shows that $\mathrm{d}_{\mathbf{Def}}(n) \leq n - 1$ for every $n \geq 1$. That actually $\mathrm{d}_{\mathbf{Def}}(n) = n - 1$, is again witnessed by the automata $\mathcal{A}(n, X)$. This observation can be generalized to read as follows.

Proposition 7.1. $\mathrm{d}_{\mathbf{Def}}(n, r) = n - r$ *whenever* $1 \leq r \leq n$.

Proof. Let $\mathcal{A} = (A, X, \delta)$ be a given automaton. For every $i \geq 0$, we define on A a relation ρ_i so that for any $a, b \in A$,

$$a \rho_i b \text{ iff } (\forall w \in X^i)\ aw = bw.$$

It is easy to see (*cf.* [Ste69]) that these relations are congruences of $\mathcal{A}$, and that $\mathcal{A}$ is k-definite $(k \geq 0)$ iff

1. $\Delta_A = \rho_0 \subset \rho_1 \subset \dots \subset \rho_{k-1} \subset \rho_k = \nabla_A$.

Suppose now that $\mathcal{A}$ has n states and is k- definite. It is clear that if $0 \leq i \leq k$ and $w \in X^i$, then $aw\rho_{k-i}bw$ for all $a, b \in A$. On the other hand, by (1) the number of ρ_{k-i}-classes is at least $i+1$. Hence $|Aw| \leq n-i$ for every $w \in X^i$. Moreover, $|Aw| = 1$ whenever $\lg(w) \geq k$. This means that if $1 \leq r \leq n$ and $w \in X^{n-r}$, then $|Aw| \leq r$. Hence $\mathrm{d}_{\mathbf{Def}}(n,r) \leq n-r$. That the bound is exact, can be seen by considering again the automata $\mathcal{A}(n,X)$.

Definite automata correspond to *definite languages* [Kle56,PRS63]. Next we consider automata that correspond to *reverse definite languages* [Brz63,Gin66]. An automaton $\mathcal{A} = (A, X, \delta)$ is *weakly reverse k-definite* ($k \geq 0$) if $awx = aw$ for all $a \in A, w \in X^k$ and $x \in X$. *Reverse definite* and *reverse k-definite* automata are now defined in the natural way. Let **RDef** be the class of reverse definite automata. If $\mathcal{A} = (A, X, \delta)$ is weakly reverse k-definite, then for all $a \in A$ and $w \in X^{\geq k}$, aw is a 'dead state', *i.e.* $awx = aw$ for every $x \in X$. This means that $\mathcal{A}$ is directable exactly in case it has just one such dead state. Recall that an automaton $\mathcal{A} = (A, X, \delta)$ is *nilpotent* (cf. [GéP72]) if there is a state $a_0 \in A$, called the *absorbing state*, and a bound $k \geq 0$ such that $aw = a_0$ whenever $a \in A$ and $\lg(w) \geq k$. Let **Nil** denote the class of nilpotent automata.

Proposition 7.2. $\mathbf{RDef} \cap \mathbf{Dir} = \mathbf{Nil}$, *and* $\mathrm{d}_{\mathbf{RDef}}(n,r) = \mathrm{d}_{\mathbf{Nil}}(n,r) = n-r$ *for all* $1 \leq r \leq n$.

Proof. Any nilpotent automaton is clearly both reverse definite and directable, and the converse we noted already above. Hence $\mathbf{RDef} \cap \mathbf{Dir} = \mathbf{Nil}$ holds. Since $\mathbf{Nil} \subseteq \mathbf{Def}$, we get $\mathrm{d}_{\mathbf{Nil}}(n,r) \leq \mathrm{d}_{\mathbf{Def}}(n,r) = n-r$ for all $1 \leq r \leq n$. Once more, equality is seen to hold by considering the automata $\mathcal{A}(n,X)$ which are also nilpotent.

An X-language L is *generalized definite* [Gin66] if it has a representation

$$L = P \cup Q_1 X^* R_1 \cup \ldots \cup Q_m X^* R_m,$$

where $m \geq 0$ and the sets P, Q_i and R_i are finite. Let us call an automaton $\mathcal{A} = (A, X, \delta)$ *generalized definite* if there are integers $h, k \geq 0$ such that $asut = asvt$, for all $a \in A$, $s \in X^h$, $t \in X^k$ and $u, v \in X^*$. This definition is justified by the following facts.

Proposition 7.3. *Let* $\mathbf{A} = (A, X, \delta, a_0, F)$ *be an* X*-recognizer based on a given* X*-automaton* $\mathcal{A} = (A, X, \delta)$.

(a) *If* $L(\mathbf{A})$ *is a generalized definite language and* $\mathbf{A}$ *is its minimal recognizer, then the automaton* $\mathcal{A}$ *is generalized definite.*

(b) *If the automaton* $\mathcal{A}$ *is generalized definite, then the language* $L(\mathbf{A})$ *is also generalized definite.*

Proof. Suppose first that $L(\mathbf{A}) = P \cup Q_1 X^* R_1 \cup \ldots \cup Q_m X^* R_m$, where $m \geq 0$ and all of the sets P, Q_i and R_i are finite, and that $\mathbf{A}$ is a minimal recognizer of $L(\mathbf{A})$. We may then assume that $P \subseteq X^{<h+k}$, $Q_1, \ldots, Q_m \subseteq X^h$ and $R_1, \ldots, R_m \subseteq X^k$,

for some $h, k \geq 0$. Consider any $a \in A$, $s \in X^h$, $t \in X^k$ and $u, v \in X^*$. Since $\mathbf{A}$ is minimal, there is a word $r \in X^*$ such that $a = a_0 r$. For any $w \in X^*$,

$$\lg(rsutw), \lg(rsvtw) \geq h + k \text{ and}$$

$$\mathrm{pref}_h(rsutw) = \mathrm{pref}_h(rsvtw), \quad \mathrm{suff}_k(rsutw) = \mathrm{suff}_k(rsvtw),$$

and hence $rsutw \in L(\mathbf{A})$ iff $rsvtw \in L(\mathbf{A})$. This shows that $a_0 rsut = asut$ and $a_0 rsvt = asvt$ are equivalent states, and since $\mathbf{A}$ is minimal, $asut = asvt$ must hold. Hence $\mathcal{A}$ is generalized definite.

Assume now that $\mathcal{A}$ is generalized definite and let $h, k \geq 0$ be such that $asut = asvt$ whenever $a \in A$, $s \in X^h$, $r \in X^k$ and $u, v \in X^*$. Consider any words $u, v \in X^*$ such that $\lg(u), \lg(v) \geq h + k$, $\mathrm{pref}_h(u) = \mathrm{pref}_h(v)$ and $\mathrm{suff}_k(u) = \mathrm{suff}_k(v)$. We may then write $u = su't$ and $v = sv't$, where $s \in X^h$ and $t \in X^k$. Now

$$u \in L(\mathbf{A}) \Leftrightarrow a_0 su'v \in F \Leftrightarrow a_0 sv't \in F \Leftrightarrow v \in L(\mathbf{A}),$$

which shows that $L(\mathbf{A})$ is generalized definite.

Let **GDef** denote the class of generalized definite automata. Clearly, $\mathbf{Def} \subset \mathbf{GDef}$ and $\mathbf{RDef} \subset \mathbf{GDef}$, but it turns out that all directable generalized definite automata are definite.

Proposition 7.4. $\mathbf{GDef} \cap \mathbf{Dir} = \mathbf{Def}$, *and hence* $\mathrm{d}_{\mathbf{GDef}}(n, r) = n - r$ *for all* $1 \leq r \leq n$.

Proof. Let $\mathcal{A} = (A, X, \delta)$ be a directable generalized definite automaton, and let $h, k \geq 0$ be such that $asut = asvt$ whenever $a \in A$, $s \in X^h$, $t \in X^k$ and $u, v \in X^*$. Let u be a directing word of $\mathcal{A}$. If $w \in X^{h+k}$, we may write $w = st$ with $s \in X^h$ and $t \in X^k$. Then for any $a, b \in A$,

$$aw = ast = aset = asut = bsut = \ldots = bw.$$

Hence $\mathcal{A}$ is definite. The converse inclusion is obvious.

If we add to Propositions 7.2 and 7.4 the obvious fact $\mathbf{Def} \cap \mathbf{RDef} = \mathbf{Nil}$, we get the following complete description of the inclusion relationships between and the intersections of the classes **Def**, **RDef**, **GDef** and **Nil**.

Proposition 7.5.

1. $\mathbf{Nil} \subset \mathbf{Def} \subset \mathbf{Dir}, \mathbf{Def} \subset \mathbf{GDef}, \mathbf{Nil} \subset \mathbf{RDef} \subset \mathbf{GDef}$,
2. $\mathbf{GDef} \cap \mathbf{Dir} = \mathbf{Def}$, *and*
3. $\mathbf{Dir} \cap \mathbf{RDef} = \mathbf{Def} \cap \mathbf{RDef} = \mathbf{Nil}$.

The relations of Proposition 7.5 are summarized by the inclusion diagram of Figure 1.

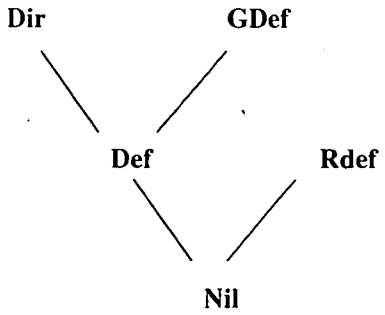

Figure 1

Finally, we note that within the intersection $\mathbf{Dir} \cap \mathbf{Com}$ all of the classes **Def**, **RDef**, **GDef** and **Nil** are equal. This follows from the next observation.

Remark 7.6. $\mathbf{Dir} \cap \mathbf{Com} \cap \mathbf{GDef} = \mathbf{Com} \cap \mathbf{Nil}$.

Proof. If $\mathcal{A} = (A, X, \delta) \in \mathbf{Dir} \cap \mathbf{Com} \cap \mathbf{GDef}$, then $\mathcal{A}$ is k-definite for some $k \geq 0$. Then for any $a, b \in A$ and $u, v \in X^{\geq k}$, $au = bvu = buv = bv$, which shows that $\mathcal{A}$ is nilpotent. The converse inclusion follows from Proposition 7.5.

References

[AHU83] A.V. Aho, J.E. Hopcroft & J.D. Ullman: *Data structures and algorithms.* - Addison-Wesley, Reading, Mass. 1983.

[Brz62] J.A. Brzozowski: Canonical regular expressions and minimal state graphs for definite events.- *Proc. Symp. Math. Theory of Automata*, Microwave Research Inst. Symp. Ser. **12** (Brooklyn 1963), Brooklyn, New York 1963, 529-561.

[BuS81] S. Burris & H.P. Sankappanavar: *A course in universal algebra.* - Springer-Verlag, New York 1981.

[Čer64] J. Černý: Poznámka k homogénym experimentom s konecinými automatami. - *Mat.-fyz. cas. SAV* **14** (1964), 208-215.

[ČPR71] J. Černý, A. Piricka & B. Rosenauerová: On directable automata - *Kybernetika (Praha)* **7** (1971), 289-297.

[DDK85] J. Demel, M. Demlova & V. Koubek: Fast algorithms constructing minimal subalgebras, congruences, and ideals in a finite algebra. - *Theoret. Comput. Sci.* **36** (1985), 203-216.

[GéP72] F. Gécseg & I. Peák: *Algebraic theory of automata.* - Akadémiai Kiadó, Budapest 1972.

[Gin66] A. Ginzburg: About some properties of definite, reverse-definite and related automata. *IEEE Trans. Electronic Computers* **EC-15** (1966), 806-810.

[ItD83] M. Ito & J. Duske: On cofinal and definite automata. - *Acta Cybern.* **6** (1983), 181-189.

[Kle56] S.C. Kleene: Representation of events in nerve nets and finite automata. - *Automata Sutides*, Princeton University Press, Princeton N.J. 1956, 3-41.

[PRS63] M. Perles, M.O. Rabin & E. Shamir: The theory of definite automata. - *IEEE Trans. Electronic Computers* **EC-12** (1963), 233-243.

[Pin78] J.-E. Pin: Sur les mots synchronisants dans un automata fini. - *Elekron. Inform.- Verarb. u. Kybernetik, EIK* **14** (1978), 297-303.

[Pin79] J.-E. Pin: Sur un cas particulier de la conjecture de Cerny. - *Automata, languages and programming,* ICALP'79 (Proc. Coll., Udine 1979), LNCS 62, Springer-Verlag, Berlin 1979, 345-352.

[Rys94] I. Rystsov: Exact linear bound for the length of reset words in commutative automata (to appear).

[Sta69] P.H. Starke: *Abstrakte Automaten.* - VEB Deutscher Verlag der Wissenschaften, Berlin 1969.

[Ste69] M. Steinby: On definite automata and related systems. - *Ann. Acad. Sci. Fenn.*, Ser. A, I Math. **444**, Helsinki 1969.

[Tar72] R. Tarjan: Depth first search and linear graph algorithms.- *SIAM J. Computing* **1** (1972), 146-160.

Received March 20, 1995

Acta Cybernetica, Vol. 12, No. 1, Szeged, 1995

The Optimistic and Cautious Semantics for Inconsistent Knowledge Bases

John Grant * V.S. Subrahmanian†

Abstract

We develop two alternative semantics, based on maximal consistent subsets, for knowledge bases that (possibly) contain inconsistencies. The optimistic (resp. cautious) semantics correspond to entailment in some (resp. all) maximal consistent subsets. We develop a Kripke-style model theory corresponding to these two semantics. We further extend these semantics to the case when knowledge bases contain both explicit and nonmonotonic negation. Notions of stratification and stability are defined and studied for both semantics.

1 Introduction

Databases and knowledge bases may be inconsistent for various reasons. For example, during the construction of an expert system, we may consult many different experts. Each expert may provide us with a group of facts and rules which are individually consistent. However, when we coalesce the facts and rules provided by these different experts, inconsistency may arise. Such an inconsistency may be due to various factors such as a disagreement between experts, an error made by an expert, or a misunderstanding between experts. In any case, we may be forced to reason in the presence of an inconsistency. Classical logic is not adequate in this situation because a single inconsistency makes all possible statements true, thereby trivializing the whole knowledge base.

In a previous companion paper [8], we developed the so-called over-determined (or OD) semantics for reasoning in inconsistent knowledge bases. However, OD-semantics is not based on classical model theory: it allows models to make a statement and its negation both simultaneously true. The semantics we develop in this paper takes the meaning of an inconsistent knowledge base to be the set of maximal consistent subsets of the knowledge base. The optimistic semantics deduces all statements deducible from at least one maximal consistent subset. In the scenario involving many experts, the optimistic semantics accepts all statements

*Department of Computer and Information Sciences, Towson State University, Towson, MD 21204. E-mail: grant@midget.towson.edu.

†Department of Computer Science, Institute for Advanced Computer Studies, and Institute for Systems Research, University of Maryland, College Park, MD 20742. E-mail: vs@cs.umd.edu.

that at least one expert can deduce and possibly additional statements deducible from the knowledge of the experts that do not involve any inconsistencies. The cautious semantics deduces those statements that are deducible from all maximal consistent subsets. In the scenario involving many experts, the cautious semantics accepts all statements that every expert can deduce and possibly additional statements deducible from the knowledge of the experts that do not involve any inconsistencies.

The organization of this paper is as follows: Section 2 contains the basic notation and definitions. It also includes a specific example that motivates the semantics for inconsistent knowledge bases. Section 3 provides the definitions and basic results for optimistic and cautious entailment. In Section 4, a Kripke semantics is developed for optimistic and cautious entailment and a fixpoint operator is presented for optimistic entailment. In Section 5, stratification and stability are extended to our framework, and their relationship is investigated. Section 6 contains a summary and a discussion of related work.

2 Motivation and Example

We assume that the facts and rules of a knowledge base are expressed as clauses of the form

$$L_0 \leftarrow L_1 \,\&\ldots\&\, L_n$$

where each L_i, $0 \leq i \leq n$, is a literal (positive or negative). Initially, only classical negation ($\neg$) is used, but in Section 5, non-monotonic negation (not) is added. A clause of the above form is called a *generally Horn clause*. Note that a generally Horn clause allows a negative literal in the head of a clause. A knowledge base is represented in the form of a *generally Horn program* (GHP, for short), which is a set, possibly infinite, of generally Horn clauses. A logical language $\mathcal{L}$ generated by a finite number of constant, function and predicate symbols (and infinitely many variable symbols) is implicit in our setup. Throughout the paper, we consider $\neg\neg A$ to be synonymous with A, i.e. double negations are deleted. We use the notation $grd(P)$ to denote the set of all ground instances of clauses in P; in fact, usually we will assume that a generally Horn program is already in ground form.
As usual, the symbol $\models$ denotes semantic consequence, i.e. $P \models L$ means that L is a semantic logical consequence of P with respect to the semantics of classical two-valued logic. In the next section, we will introduce two new notions of entailment: $\vdash_\exists$ for the optimistic semantics, and $\vdash_\forall$ for the cautious semantics.
Next, we present a motivating scenario that exemplifies situations involving inconsistencies that arise in law enforcement agencies and in the judicial process:
Bill hosted a dinner at his house on Jan. 26, 1995. The party was attended by Al, Carl, Dick, Ed and Tom. Tom had to leave during dinner because his daughter had a medical emergency. After dinner, Bill went to the kitchen to prepare coffee. As he did not return, Dick and Ed went to the kitchen where they found Bill strangled to death. At the time of the crime:

- (F1) Tom was in the emergency room of a hospital. His presence was recorded by a surveillance camera belonging to hospital security.
- (F2) Bill was in the kitchen.
- (F3) Dick and Ed said they were talking in the living room.
- (F4) Al said he was alone in the bathroom.

- (F5) Carl said he was alone in the bathroom.

Furthermore,

- (F6) Bill's house has only one bathroom.
- (F7) Al had been guilty of embezzling money from Bill's accounting firm.
- (F8) Carl was having an affair with Bill's wife. Bill was an intensely jealous man.

Let us examine the story more carefully. First of all, the story contains a glaring contradiction: Al and Carl's stories conflict. This suggests that one of them is lying and we may further suspect that the person who is lying is the murderer. For lack of additional information, we are unable to determine which of them is actually the criminal. In this situation, the police may well decide to forget about Dick and Ed and look more closely at Al and Carl. More fantastic scenarios are also possible: Al and Carl may have been in cahoots and killed Bill and then both lied so that a convincing case could not be made against either of them. Alternatively, it is possible that everybody (except Tom) is lying: Al may have been in Bill's study trying to steal documentary evidence of his embezzlement, while Carl may have been in Bill's bedroom trying to get back his tie which he had left behind during one of his previous soirees with Bill's wife. While this was going on, Dick and Ed (both in cahoots) may have teamed up and killed Bill.
Whether one chooses to believe the above scenarios or not, one must admit that each of them is *possible*, though some are perhaps more probable than others.
However, whatever version we choose to believe, we would all be agreed that the general floor plan of Bill's house should be the same in all versions of the story. Likewise, the fact that Bill was strangled is true in all versions. In other words, in all versions of the story, certain facts are true. One may accept all these facts as being "certain" or established. The different versions of the story would tell us who to believe and who not to believe – in other words, they identify the suspects. Presumably, Tom would not be a suspect.
A formal logical description of the scenario is given in the Appendix. The cause of the inconsistency in the above example is the set of sentences $Cause = \{13, 14, 15, 16, 17\}$. Maximal consistent subsets may be obtained by dropping any one of these clauses.

3 Optimistic and Cautious Entailment

Suppose P is a GHP. We say that the success set of P, denoted $SS(P)$ is the set $\{L \mid L$ is a ground literal such that $P \models L\}$. Note that as P is a first order theory, $\models$ denotes standard semantical consequence in first order logic.

Example 3.1 Suppose P is the GHP:

$$p \leftarrow$$

$$\neg q \leftarrow p$$

$$r \leftarrow q$$

Then $SS(P) = \{p, \neg q\}$. Neither r nor $\neg r$ is in $SS(P)$. In particular, non-monotonic inference rules such as negation as failure and/or the closed world assumption are not used here because negation is represented explicitly.

A set $Q \subseteq P$ (where P is a GHP) is said to be *maximal consistent* iff Q is consistent, and there is no consistent program Q' such that $Q \subset Q' \subseteq P$.

Theorem 1 *Every GHP P has at least one maximal consistent subset.*

Proof. P has at least one consistent subset, viz. the empty set of clauses. Let $CONS(P)$ be the set of all consistent subsets of P. We show below that every ascending chain of elements in $CONS(P)$ has an upper bound in $CONS(P)$. The result then follows from Zorn's Lemma.
Suppose $S_1 \subseteq S_2 \subseteq S_3 \subseteq \cdots$ is an ascending sequence of members of $CONS(P)$, i.e. each S_i is a consistent subset of $CONS(P)$. Then $S = \bigcup_{i=1}^{\infty} S_i$ is an upper bound for this ascending sequence. Moreover, S is consistent, i.e. $S \in CONS(P)$. To see this, suppose S is not consistent. Then, by the Compactness Theorem, there is a finite subset $S' \subseteq S$ such that S' is inconsistent. Let $S' = \{\gamma_1, \ldots, \gamma_n\}$ for some integer n. Hence, for each $1 \leq i \leq n$, there is an integer, denoted α_i such that $\gamma_i \in S_{\alpha(i)}$. Let $\alpha = max\{\alpha_1 \ldots, \alpha_n\}$. Then $S' \subseteq S_\alpha$. Hence, as S' is inconsistent, S_α is also inconsistent, thus contradicting our assumption that each S_j, $j \geq 1$, is in $CONS(P)$. □
Note that the above proof applies when P is any set of formulas, not just clauses. Furthermore, the proof applies even if P is an infinite set of formulas.

Example 3.2 Suppose P and Q are the two programs listed below:

P	Q
$p \leftarrow q$	$p \leftarrow r$
$\neg p \leftarrow \neg q$	$\neg p \leftarrow$

Here, $SS(P) = \emptyset$, while $SS(Q) = \{\neg p, \neg r\}$. Note that $Q \models \neg r$ because $(p \vee \neg r)$ and $\neg p$ yield $\neg r$ as a logical consequence.

Definition 3.1 Suppose P is a GHP, and F is a formula. We introduce[1] two new notions of entailment, denoted $\vdash_\exists, \vdash_\forall$ below:

1. $P \vdash_\exists F$ iff there is some maximal consistent subset $P' \subseteq P$ such that $P' \models F$.
2. $P \vdash_\forall F$ iff $P' \models F$ for every maximal consistent subset $P' \subseteq P$.

Example 3.3 Suppose P is the GHP below:
1 : $p \leftarrow q$
2 : $\neg p \leftarrow q$
3 : $q \leftarrow$
Clearly P is inconsistent. P has three maximal consistent subsets, viz. $P_1 = \{1,2\}, P_2 = \{2,3\}, P_3 = \{1,3\}$.
$SS(P_1) = \{\neg q\}$,
$SS(P_2) = \{q, \neg p\}$
$SS(P_3) = \{p, q\}$.

[1] We are grateful to Professor Newton da Costa for suggesting that $\vdash_\exists$ entailment may be a useful concept. The basic intuition behind $\vdash_\forall$ entailment is not entirely new. The idea of using maximal consistent subsets for hypothetical reasoning goes back to Rescher [12] whose work was later adapted to artificial intelligence by Ginsberg [5]. However, the technical properties of $\vdash_\exists$ and $\vdash_\forall$ entailment have not been studied carefully thus far and this is one of the things we do in this paper.

Thus, $P \vdash_\exists p$, $P \vdash_\exists \neg p$, $P \vdash_\exists q$ and $P \vdash_\exists \neg q$. There is no ground literal L such that $P \vdash_\forall L$.

Let us try to get some intuition. In $\vdash_\exists$ entailment, we adopt an *optimistic* approach. If P is inconsistent, then we say that L is true iff L is a consequence of *some* consistent subset of P. However, $\vdash_\forall$ is more *cautious*. It is not easily willing to admit that anything is true. For us to conclude L using $\vdash_\forall$ entailment, we must, intuitively, find all possible causes of inconsistency. If, after eliminating the cause of the inconsistency in all possible ways, it turns out that L is true in each scenario, only then do we consider L to be true. Intermediate concepts of inconsistency may also be devised such as the one in [9] where the concept of a "recoverable" literal is used.

Example 3.4 Consider the murder example of Section 2. Intuitively, a formula is $\vdash_\forall$ entailed iff it is true in all possible consistent scenarios. Thus, for example, the fact that Bill was alive when dinner was finished is $\vdash_\forall$ entailed by the evidence because it is true irrespective of whose version of the evidence we choose to believe. Likewise, the fact that Tom could not have been the murderer is clearly $\vdash_\forall$ entailed by the evidence.
On the other hand, for each person (except Tom) who had dinner with Bill that night, there is a scenario in which he could be the murderer. Thus, $\vdash_\exists$ entailment allows us to conclude, for example, that Carl is the murderer.
In effect, we can *use* $\vdash_\exists$ entailment in order to identify suspects, rather than to identify the murderer. $\vdash_\exists$ entailment tells us who we may safely ignore as a candidate murderer.

There is one feature of Example 3.3 that some readers may find curious. This concerns $\vdash_\exists$: Here, $P \vdash_\exists p$ and $P \vdash_\exists \neg p$, but $P \not\vdash_\exists (p \,\&\, \neg p)$. A brief discussion of this is in order. Even though p is true in some maximal consistent subset of P and likewise $\neg p$ is true in some maximal consistent subset of P, these two maximal subsets are different. In fact, there cannot be a *single* consistent subset of P in which *both* p and $\neg p$ are true. So even though P exhibits this kind of classical inconsistency with respect of $\vdash_\exists$ entailment, this inconsistency is not trivializing, i.e. the existence of such an inconsistency does not cause all formulas in our language to become $\vdash_\exists$ entailed by P.
Formally, some of these properties may be stated below:

Proposition 3.1 *Suppose P is a GHP. If P is consistent, then the following sentences are equivalent for all ground literals L:*

1. $P \models L$.
2. $P \vdash_\exists L$.
3. $P \vdash_\forall L$. □

Proposition 3.2 *Suppose P is a GHP (possibly inconsistent) and L, L_1, L_2 are ground literals. Then:*

1. *if $P \vdash_\exists (L_1 \,\&\, L_2)$, then $P \vdash_\exists L_1$ and $P \vdash_\exists L_2$.*
2. *In general, $P \vdash_\exists L_1$ and $P \vdash_\exists L_2$ do not imply that $P \vdash_\exists (L_1 \,\&\, L_2)$.*
3. *$P \vdash_\exists F$ for all tautologies F of classical logic.*

Proof. (1) Suppose $P \vdash_\exists (L_1 \,\&\, L_2)$. Then there is a maximal consistent subset $Q \subseteq P$ such that $Q \models L_1 \,\&\, L_2$. Hence, $Q \models L_1$. Thus, $P \vdash_\exists L_1$. Similarly for L_2.
(2) Immediate from Example 3.3.
(3) Suppose F is a tautology of classical logic. Then F is a logical consequence of the empty set, and hence F is a logical consequence of each consistent subset of P. □

The above proposition shows that the tautologies of classical logic hold with respect to $\vdash_\exists$- entailment. Similar properties hold for $\vdash_\forall$-entailment.

Theorem 2 *Suppose P is a GHP, and L, L_1, L_2 are ground literals. Then:*

1. *there is no ground literal L such that $P \vdash_\forall L$ and $P \vdash_\forall \neg L$.*
2. *$P \vdash_\forall (L_1 \,\&\, L_2)$ iff $P \vdash_\forall L_1$ and $P \vdash_\forall L_2$.*
3. *$P \vdash_\forall F$ for all tautologies F of classical logic.*

Proof. (1) Suppose $P \vdash_\forall L$ and $P \vdash_\forall \neg L$. Hence, for each maximal consistent subset Q of P, $Q \models L$ and $Q \models \neg L$, which contradicts our assumption that Q is maximal consistent. This means that there is no maximal consistent subset of P, which is impossible by Theorem 1.
(2) Suppose $P \vdash_\forall (L_1 \,\&\, L_2)$. Then $L_1 \,\&\, L_2$ is a logical consequence of every maximal consistent subset Q of P. Hence, each maximal consistent subset Q of P has L_1 and L_2 as a logical consequence, i.e. $Q \vdash_\forall L_1$ and $Q \vdash_\forall L_2$.
Suppose $P \vdash_\forall L_1$ and $P \vdash_\forall L_2$. Then L_1 and L_2 are both true in every maximal consistent subset Q of P, i.e. $P \vdash_\forall (L_1 \,\&\, L_2)$.
(3) The proof proceeds along the same lines as the proof of Proposition 3.2(3). □

Example 3.5 Let P be:

$$p \leftarrow q$$
$$\neg p \leftarrow q$$
$$r \leftarrow$$
$$q \leftarrow$$

In this case, $P \vdash_\forall r$. But $P \not\vdash_\forall q$ and $P \not\vdash_\forall p$ and $P \not\vdash_\forall \neg q$ and $P \not\vdash_\forall \neg p$.

Thus, unlike $\vdash_\exists$ which can cause both L and $\neg L$ (but never $(L \& \neg L)$) to be inferred from a program, $\vdash_\forall$ does not allow this. However, $\vdash_\forall$ allows very few conclusions to be drawn. The following result is an immediate consequence of the fact that $CONS(P)$ is always non-empty.

Proposition 3.3 *Suppose P is any GHP and L any ground literal. If $P \vdash_\forall L$, then $P \vdash_\exists L$.* □

We now demonstrate $\vdash_\exists$ and $\vdash_\forall$ entailment on a simple example.

Example 3.6 Consider the program P below:
1: $b \leftarrow a$
2: $\neg b \leftarrow a$
3: $a \leftarrow c$
4: $a \leftarrow$
5: $c \leftarrow$
There are four maximal consistent sets: $P_1 = \{1,2,3\}, P_2 = \{1,2,5\}, P_3 = \{1,3,4,5\}$ and $P_4 = \{2,3,4,5\}$. $SS(P_1) = \{\neg a, \neg c\}$, $SS(P_2) = \{\neg a, c\}$, $SS(P_3) = \{a, b, c\}$, $SS(P_4) = \{a, \neg b, c\}$. The literals that are $\vdash_\exists$-entailed by P are: $\{\neg a, \neg c, c, a, b, \neg b\}$. The set of literals $\vdash_\forall$-entailed by P is $\emptyset$.

In simple examples, such as Examples 3.3 and 3.6, for any two distinct maximal consistent subsets P_1, and P_2, there is usually a literal ℓ such that $\ell \in SS(P_1)$ and $\neg\ell \in SS(P_2)$. The following more complex example shows that this need not always be the case.

Example 3.7 Consider the set of clauses:

1: $\neg p \leftarrow$
2: $\neg q \leftarrow$
3: $p_3 \leftarrow$
4: $q_3 \leftarrow$
5: $q_1 \leftarrow \neg p_1 \,\&\, \neg p_2$
6: $q_1 \leftarrow p_1 \,\&\, \neg p_2$
7: $q_1 \leftarrow \neg p_1 \,\&\, p_2$
8: $q_2 \leftarrow p_1 \,\&\, \neg p_2$
9: $q_2 \leftarrow \neg p_1 \,\&\, p_2$
10: $q_2 \leftarrow \neg p_1 \,\&\, \neg p_2$
11: $p_1 \leftarrow \neg q_1 \,\&\, \neg q_2$
12: $p_1 \leftarrow q_1 \,\&\, \neg q_2$
13: $p_1 \leftarrow \neg q_1 \,\&\, q_2$
14: $p_2 \leftarrow q_1 \,\&\, \neg q_2$
15: $p_2 \leftarrow \neg q_1 \,\&\, q_2$
16: $p_2 \leftarrow \neg q_1 \,\&\, \neg q_2$
17: $p \leftarrow p_1 \,\&\, p_2 \,\&\, p_3$
18: $q \leftarrow q_1 \,\&\, q_2 \,\&\, q_3$

There are several maximal consistent sets here. Let $P_1 = \{1, \ldots, 16, 17\}$ and $P_2 = \{1, \ldots, 16, 18\}$. P_1 and P_2 are maximal consistent subsets of P. $SS(P_1) = \{\neg p, \neg q, p_3, q_3, q_1, q_2\}$, $SS(P_2) = \{\neg p, \neg q, p_3, q_3, p_1, p_2\}$. Note that there is no literal in $SS(P_1)$ whose negation is in $SS(P_2)$.

Before concluding this section, we briefly observe that the problem "Given as inputs, a GHP P, and a literal L, determining whether $P \vdash_\forall L$" is Π_2^p-complete and the analogous problem "Given as inputs, a GHP P, and a literal L, determining whether $P \vdash_\exists L$" is Σ_2^p-complete. The former is true because
$P \vdash_\forall L \Leftrightarrow$
$([(\forall Q \subseteq P)(Q \text{ is consistent } \&\ (\forall Q^*)(Q \subset Q^* \subseteq P \rightarrow Q^* \text{ is inconsistent})] \rightarrow Q \models L)$

This is a Π_2^p problem because it involves a universal quantification over an NP-complete problem (viz. checking the consistency of Q, and making a polynomial-number of inconsistency checks of Q^*). The Σ_2^p-result for optimistic entailment follows analogously, together with the observation that it involves an existential quantification over the same NP-complete problem.

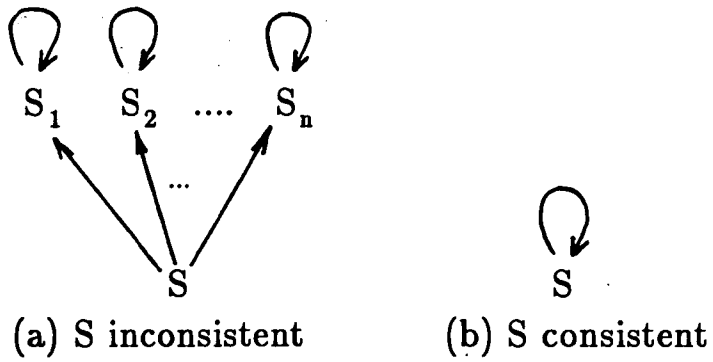

Figure 1: Graphical Representation of $PK(S)$

4 Kripke Semantics and a Fixpoint Operator

In this section, we develop a Kripke-style model theory for optimistic and cautious entailment. We also develop a fixpoint operator for the optimistic semantics. We assume that a GHP is a finite set of ground clauses. Given a GHP P, we use $\mathbf{D}(P)$ to denote the set of all ground disjunctions of literals (including the empty disjunction) expressible using the language of P.

Definition 4.1 An elementary structure of the language of P (*e-structure*, for short) is any subset of $\mathbf{D}(P)$.

Definition 4.2 An e-structure S of a GHP P is said to be a consistent structure (*c-structure*, for short) iff S has a model in the sense of classical logic.

Definition 4.3 Suppose P is a GHP and S is an e-structure of P. The *paraconsistent Kripke structure* (PK-structure, for short) *based on* S is a pair $(Int(S), Edge(S))$ defined as follows:

1. If S is a c-structure, then $Int(S) = \{S\}$ and $Edge(S) = \{(S, S)\}$.
2. If S is not a c-structure, then:
   (a) $Int(S) = \{S\} \cup \{S' \mid S' \subseteq S$ and S' is a c-structure and there is no $J \subseteq S$ such that J is a c-structure and $S' \subset J\}$.
   (b) $Edge(S) = \{(S, J) \mid J \in (Int(S) - \{S\})\} \cup \{(J, J) \mid J \in (Int(S) - \{S\})$.

Figure 1 shows a graphical representation of $PK(S)$. In Figure 1(a), $S_1, \ldots, S_n$ are maximal c-substructures of S.

Example 4.1 Suppose P is a GHP written in the language consisting of three propositional symbols p, q and r, and S is the e-structure $\{p, q, \neg q\}$, then $PK(S)$ is the pair $(Int(S), Edge(S))$ where:
$Int(S) = \{S, \{p, q\}, \{p, \neg q\}\}$
$Edge(S) =$ (the reflexive closure of $\{(S, \{p, q\}), (S, \{p, \neg q\})\}) - \{(S, S)\}$.

Suppose $\mathcal{L}$ is a first order language. We extend $\mathcal{L}$ to a modal language, denoted $\mathcal{L}_{\mathcal{M}}$ defined as follows:

1. Every wff of classical logic is a wff of $\mathcal{L}_{\mathcal{M}}$.
2. If F is a wff of $\mathcal{L}_{\mathcal{M}}$, then $\Diamond F$ and $\Box F$ are wffs of $\mathcal{L}_{\mathcal{M}}$. (Intuitively, $\Diamond F$ is to be read as "F is possible", while $\Box F$ is to be read as "F is necessary".)
3. If F and G are wffs of $\mathcal{L}_{\mathcal{M}}$, then $F \,\&\, G$, $F \vee G$, $\neg F$, $F \leftarrow G$, $F \leftrightarrow G$ and $(\forall x)F$ and $(\exists x)F$ are wffs of $\mathcal{L}_{\mathcal{M}}$.

$\mathcal{L}_{\mathcal{M}}$ is interpreted by a PK-structure based on an e-structure S defined as follows:

Definition 4.4 Suppose S is an e-structure, and let $E = (Int(S), Edge(S))$ be the PK-structure based on S. Let $w \in Int(S)$. Then we say that E, w *satisfies* F, denoted $E, w \mapsto F$ as follows:

1. If F is a wff of classical logic, then:
   (a) (F an atom) $E, w \mapsto F$ iff F is a logical consequence of w
   (b) ($F = \neg G$) $E, w \mapsto F$ iff G is not a logical consequence of w (here, we assume G is an atom)
   (c) ($F = G \,\&\, H$) $E, w \mapsto F$ iff $E, w \mapsto G$ and $E, w \mapsto H$
   (d) ($F = G \vee H$) $E, w \mapsto F$ iff $E, w \mapsto G$ or $E, w \mapsto H$
   (e) ($F = G \rightarrow H$) $E, w \mapsto F$ iff $E, w \mapsto H$ or $E, w \not\mapsto G$
   (f) Satisfaction of formulas whose leading connective is a quantifier is defined in the usual way.
2. Suppose $F \equiv \Diamond G$. Then $E, w \mapsto F$ iff for some w' such that $(w, w') \in Edge(S)$, $E, w' \mapsto G$.
3. Suppose $F \equiv \Box G$. Then $E, w \mapsto F$ iff for each w' such that $(w, w') \in Edge(S)$, $E, w' \mapsto G$.
4. Satisfaction of formulas whose leading connectives are conjuncts, disjuncts, implications, iff, and the quantifiers are defined in the usual way.

Using the notion of a PK-structure based on an e-structure S, we may define the model theoretic semantics of the logics corresponding to $\vdash_\exists$ and $\vdash_\forall$ entailment.

Definition 4.5 Suppose F is a formula of classical logic and S is an e-structure. Let E be the PK-structure determined by S. We say that

1. $S \models_\exists F$ iff $E, S \mapsto \Diamond F$.
2. $S \models_\forall F$ iff $E, S \mapsto \Box F$.

Suppose Δ is a set of formulas. $S \models_\exists \Delta$ iff $S \models_\exists \delta$ for all $\delta \in \Delta$.

In order to show the equivalence of $\vdash_\exists$ and $\models_\exists$, we need a definition.

Definition 4.6 Given a clause $C \equiv$

$$D \leftarrow L_1 \,\&\, \ldots \,\&\, L_n$$

the disjunctive form of C, denoted $disj(C)$, is the clause:

$$D \vee \neg L_1 \vee \cdots \vee \neg L_n.$$

The *disjunctive form*, $disj(P)$, of a GHP P is the set $\{disj(C) \mid C \in P\}$.

Proposition 4.1 Suppose P is a GHP and D is a ground disjunction. Then:

1. $P \vdash_\exists D$ iff $P \models_\exists D$
2. $P \vdash_\forall D$ iff $P \models_\forall D$.

Proof. We prove (1) above. the proof of (2) is similar.
Suppose $P \vdash_\exists D$. Then $disj(P) \vdash_\exists D$. Hence, there is a consistent subset $P' \subseteq disj(P)$ such that $P' \models D$. Suppose now that I is an e-structure such that $I \models_\exists P$. Clearly, $disj(P) \subseteq I$ and hence, $P' \subseteq I$. Extend P' to a maximal consistent subset of I. This maximal consistent subset of I must make D true.
Conversely, suppose $P \models_\exists D$. Consider the e-structure $disj(P)$. As $disj(P) \models_\exists P$, there is a maximal consistent subset $I^* \subseteq disj(P)$ such that D is true in I^*. Let $P' = I^*$. This completes the proof. □
Given a GHP P, we observe that P may entail a ground literal even though there is no clause in P having an instance containing that ground literal as the head. To see this observe that the program P below entails $\neg b$:

$a \leftarrow b$
$\neg a \leftarrow b$

There is no clause in P with $\neg b$ as the head. Now add the contrapositives to P.
$\neg b \leftarrow \neg a$ (contrapositive of first clause)
$\neg b \leftarrow a$ (contrapositive of second clause)
The expanded program is equivalent to the original program P. The addition of contrapositives now yields a clause with $\neg b$ in the head.
Consider now the program Q below:

$$p \leftarrow a$$

$$p \leftarrow b$$

$$a \leftarrow \neg b$$

$$b \leftarrow \neg a$$

We would like to define a fixed-point operator which yields p as a consequence of Q. Moreover, $(a \vee b)$ should also be a consequence of Q.
Based on the optimistic notion of entailment, we now develop a fixed point semantics for $\vdash_\exists$-entailment. We start by observing that given a clause C, there may be disjunctions, D, of literals that are logically entailed by the program P, but do not appear in the head of C. The implicational form of clause C, defined below, rewrites C in all possible disjunctive ways.

Definition 4.7 Suppose $C \equiv$

$$L \leftarrow L_1 \,\&\, \ldots \,\&\, L_n$$

is a clause. The *implicational form*, $IF(C)$, of C is the set of clauses $\{L'_1 \vee \cdots \vee L'_m \leftarrow Body \mid \{L'_1, \ldots, L'_m\} \cup \{\neg K \mid K \in Body\} = \{L, \neg L_1, \ldots, \neg L_n\}$ and $m > 0\}$. The *normal form*, $NF(P)$ of a generally Horn program P is then defined to be:

$$NF(P) = \bigcup_{C \in P} IF(C).$$

Note that $disj(C) \in IF(C)$ and $disj(P) \subseteq NF(P)$. Given a ground disjunction D and a GHP P, we use the notation $\mathbf{sub}(D, P)$ to denote the set $\{C \mid C$ is a clause in $NF(P)$ such that the head of C subsumes $D\}$. Thus, if D is not subsumed by the head of any clause in $NF(P)$, then $\mathbf{sub}(D, P) = \emptyset$.

Definition 4.8 Suppose P is a GHP and S is an e-structure. We define an operator that maps e-structures to e-structures. Let $TAUT$ denote the set of all tautologous clauses expressible in our language.
$V_P(S) = TAUT \cup \{D \mid \mathbf{sub}(D, P) \neq \emptyset$ and such that:

1. for all $1 \leq i \leq n$, there exists a disjunction E_i (possibly empty) of ground literals such that $PK(S), S \mapsto \Diamond(\bigvee_{i=1}^{k}(\bigwedge_{j=1}^{n_i}(L^i_j \vee E_i)))$ where $\mathbf{sub}(D, P) = \{C_1, \ldots, C_k\}$ for $k \geq 1$ and each C_i is of the form

$$C_i \equiv \; D'_i \leftarrow L^i_1 \,\&\, \cdots \,\&\, L^i_{n_i}$$

and

2. for all $1 \leq i \leq n$, the smallest factor of $(D'_i \vee E_i)$ subsumes D.

Remark 4.1 When a GHP is a disjunctive logic program in the sense of Minker and Rajasekar [11], (i.e. clause heads and clause bodies may contain no negated atoms), our operator is essentially the same as that of Rajasekar and Minker. The only difference is that in our case, the presence of subsumed clauses is explicit in $V_P(S)$, while in the case of Rajasekar and Minker, it is implicit.

To see how V_P works, consider the following example.

Example 4.2 Suppose P consists of the following five clauses:

1: $p \leftarrow q \,\&\, \neg q$
2: $r \leftarrow p$
3: $r \leftarrow \neg p$
4: $q \leftarrow$
5: $\neg q \leftarrow$

(Note here that $NF(P)$ contains more clauses, but these are not needed for this example.) Let S be the e-structure $\{q, \neg q\}$. Then the set of ground atoms in $V_P(S)$ is the set $\{q, \neg q, r\}$. Let us explain two things: (1) why $r \in V_P(S)$ and (2) why $p \notin V_P(S)$.
(1) Note that $\mathbf{sub}(r, P) = \{2, 3\}$. In particular, using the notation of Definition 4.8, we may assume the E_i's to be the empty clause. Observe that

$$PK(S), S \mapsto \Diamond(p \vee \neg p).$$

To see this note that in this case, $Int(S) = \{S, \{q\}, \{\neg q\}\}$. It is easy to see that $p \vee \neg p$ is true in both c-structures $\{q\}, \{\neg q\}$ that are accessible from world S.
(2) To see why p cannot be in $V_P(S)$, observe that the only clause with p is the head is clause (1). The antecedent of clause (1) is a flat contradiction which cannot be true in either $\{q\}$ or $\{\neg q\}$.

Intuitively, the operator V_P is supposed to capture the notion of $\models_\exists$ entailment.

Example 4.3 Consider the consistent GHP P below:

$$p \leftarrow a$$

$$\neg p \leftarrow a$$

$$b \leftarrow \neg a$$

$\neg a$ is a logical consequence of P, and hence b should be a logical consequence of P. Here, $NF(P)$ is the program:

1. $p \leftarrow a$
2. $\neg p \leftarrow a$
3. $b \leftarrow \neg a$
4. $\neg a \leftarrow \neg p$
5. $\neg a \leftarrow p$
6. $a \leftarrow \neg b$
7. $p \vee \neg a \leftarrow$
8. $\neg p \vee \neg a \leftarrow$
9. $b \vee a \leftarrow$

The least fixed-point of V_P is constructed as follows:
$V_P \uparrow 0 = \emptyset$
$V_P \uparrow 1$ contains $\neg a$, $b \vee a$ together with tautologies and subsumed clauses
$V_P \uparrow 2$ contains b, $V_P \uparrow 1$, together with tautologies and subsumed clauses

We end this section by proving the soundness and completeness of the computation captured by the fixed-point operator V_P.

Theorem 3 Suppose P is a GHP and D is any ground disjunction. Then $D \in V_P \uparrow \omega$ iff $P \vdash_\exists D$.

Proof. We first show that if $D \in V_P \uparrow \omega$ then $P \vdash_\exists D$.
Suppose $D \in V_P \uparrow \omega$. Then there is an integer $n < \omega$ such that $D \in V_P \uparrow n$. We proceed by induction on n.
Base Case. $(n = 0)$ Trivial.
Inductive Case. $(n = r + 1)$ Suppose $\mathbf{sub}(D, P) = \{C_1, \ldots, C_k\}$ where

$$C_i \equiv D'_i \leftarrow L^i_1 \,\&\, \cdots \,\&\, L^i_{n_i}.$$

Then, as $D \in V_P \uparrow (r + 1)$, it follows that

$$PK(V_P \uparrow r), V_P \uparrow r \mapsto \Diamond(\bigvee_{i=1}^{k}(\bigwedge_{j=1}^{n_i}(L_i \vee E_i)))$$

where the E_i's are ground clauses (possibly empty). Let $M_1, \ldots, M_s$ be the maximal c-structures that are subsets of $V_P \uparrow r$. From the above, we know that there is a $1 \leq j \leq s$ such that $M_j \mapsto (\bigvee_{i=1}^{k}(\bigwedge_{j=1}^{n_i}(L_i \vee E_i)))$, and hence it follows by the induction hypothesis that there is a maximal consistent subset P_j of P such that

$P_j \models (\bigvee_{i=1}^{k}(\bigwedge_{j=1}^{n_i}(L_i \vee E_i)))$. As P_j is maximal and consistent, it must also entail D. Therefore, $P \vdash_{\exists} D$.
To prove the converse, i.e. to show that if $P \vdash_{\exists} D$, then $D \in V_P \uparrow \omega$, we proceed as follows: As $P \vdash_{\exists} D$, there is a maximal consistent subset Q of P such that $Q \models D$. This is classical logic entailment. Transform Q into a disjunctive logic program (in the sense of Rajasekar and Minker [11]) as follows: if

$$A_1 \vee \cdots \vee A_n \vee \neg B_1 \vee \cdots \neg B_m \leftarrow D_1 \,\&\, \ldots \,\&\, D_r \,\&\, \neg E_1 \,\&\, \cdots \,\&\, \neg E_s$$

is in D, then replace it by the disjunctive clause:

$$A_1 \vee \cdots \vee A_n \vee E_1 \vee \cdots \vee E_s \leftarrow B_1 \,\&\, \cdots \,\&\, B_m \,\&\, D_1 \,\&\, \ldots \,\&\, D_r$$

The resulting program, called Q', is a disjunctive logic program in the sense of Rajasekar and Minker and hence it has the same logical consequences as D. As the V_P operator of ours is equivalent to that of Rajasekar's and Minker's for disjunctive logic programs, it follows by a result of theirs that $D \in lfp(V_P)$ and hence $D \in V_P \uparrow \omega$.
This completes the proof. □

5 Stratification and Stability

So far, we have assumed that negation (the symbol $\neg$) represents a "classical" form of negation, i.e. in order to conclude $\neg A$ for some ground atom A, one must explicitly establish the truth of $\neg A$ rather than reason from the lack of a proof of A. However, it is now widely accepted that requiring the explicit specification of negative information causes knowledge bases often to grow very large. However, as argued by Gelfond and Lifschitz [7] and Kowalski and Sadri [10], in many cases both classical and non-monotonic modes of negation are required. In this section, we extend the optimistic and cautious semantics to incorporate non-monotonic negation. Then we show how the concepts of stratification and stability can be extended to this framework.

Definition 5.1 If $L, L_1, \ldots, L_n, L'_1, \ldots, L'_m$ are literals, then

$$L \leftarrow L_1 \,\&\, \ldots \,\&\, L_n \,\&\, \mathbf{not}\, L'_1 \,\&\, \ldots \,\&\, \mathbf{not}\, L'_m$$

is called an *extended program clause*. An *extended GHP* (called EGHP, for short), is a finite set of extended program clauses.

Here, the symbol **not** denotes a non-monotonic mode of negation. As usual, we deal with the set of all ground instances of clauses in an extended GHP. Now, we extend the standard definitions of stability to deal with non-monotonic negation using the optimistic and cautious semantics.

Definition 5.2 Suppose P is an EGHP and X is a set of ground literals. The *transformation* of P w.r.t. X is the logic program $G(P, X)$ obtained as follows:

1. if C is a program clause in P of the form

$$L \leftarrow L_1 \,\&\, \ldots \,\&\, L_n \,\&\, \mathbf{not}\, H_1 \,\&\, \ldots \,\&\, \mathbf{not}\, H_m$$

such that for all $1 \leq i \leq m$, $H_i \notin X$, then

$$L \leftarrow L_1 \,\&\, \ldots \,\&\, L_n$$

is in $G(P, X)$.

2. Nothing else is in $G(P, X)$.

Definition 5.3 Given an EGHP P, we define two operators that map sets of ground literals to sets of ground literals as follows:
$\mathbf{A}_P(X) = \{\psi \mid \psi$ is a ground literal such that $G(P, X) \vdash_\forall \psi\}$.
$\mathbf{E}_P(X) = \{\psi \mid \psi$ is a ground literal such that $G(P, X) \vdash_\exists \psi\}$.

Definition 5.4 A set X of ground literals is

1. an **A**-answer set for EGHP P iff $\mathbf{A}_P(X) = X$
2. an **E**-answer set for EGHP P iff $\mathbf{E}_P(X) = X$

In general, EGHPs may have zero, one or many answer sets. The notion of an answer set is similar to the notion of a stable model; however, an answer set need not be a model of P.

Example 5.1 Consider the program:

$$a \leftarrow \mathbf{not}\, a \quad (1)$$
$$\neg a \leftarrow \mathbf{not}\, a \quad (2)$$

This program has an **A**-answer set $\emptyset$, but no **E**-answer set.

Example 5.2 Consider the program:

$$a \leftarrow \mathbf{not}\, b \quad (3)$$
$$\neg a \leftarrow \mathbf{not}\, b \quad (4)$$
$$b \leftarrow a \,\&\, \neg a \quad (5)$$

This program has an **A**-answer set $\emptyset$ and an **E**-answer set $\{a, \neg a\}$.

Example 5.3 Consider the program:

$$a \leftarrow \mathbf{not}\, a \,\&\, \mathbf{not}\, b \quad (6)$$
$$b \leftarrow \quad (7)$$
$$\neg b \leftarrow \quad (8)$$

This program has an **E**-answer set $\{b, \neg b\}$, but no **A**-answer set.

We may wonder under what conditions an EGHP has a unique **E**-answer set or **A**-answer set. We now study this problem and provide a sufficient, but not necessary condition to guarantee the existence of such answer sets. This is achieved by extending the concept of stratification to EGHPs.
For logic programs, stratification may be defined in terms of a level mapping of ground atoms. In our case, a level mapping is a function from the set of ground literals to the set of non-negative integers. The value of a literal L under level mapping ℓ is written as $\ell(L)$. The levels of a program are assumed to range from 0 to k for some integer k. The clauses of the program are placed in strata S_i, $0 \leq i \leq k$, by placing a clause whose head has level i into S_i. For the definitions below, we use the generic clause

$$L \leftarrow L_1 \,\&\, \ldots \,\&\, L_n \,\&\, \mathbf{not}\, L'_1 \,\&\, \ldots \,\&\, \mathbf{not}\, L'_m.$$

We start with an (intermediate) definition.

Definition 5.5 ([8]) An extended GHP P is said to be *OD-stratified* iff there is a level mapping ℓ such that for every clause $C \in grd(P)$ of the above form, $\ell(L_i) \leq \ell(L)$ and $\ell(L'_j) < \ell(L)$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$.

Basically, OD-stratification treats all literals equally and does not allow recursion through non-monotonic negation (**not**), but allows recursion through classical negation ($\neg$).

Definition 5.6 The *switched form* $SF(C)$ of a (generic) clause C is the set of clauses $\{\neg L_i \leftarrow \neg L \,\&\, L_1 \,\&\, \ldots \,\&\, L_{i-1} \,\&\, L_{i+1} \,\&\, \ldots \,\&\, L_n \,\&\, \mathbf{not}\, L'_1 \,\&\, \ldots \,\&\, \mathbf{not}\, L'_m\}$ obtained from C by switching (and negating) the literal in the head of C with a literal in the body not preceded by **not**. The *switched form* $SF(P)$, of an EGHP P is defined as $SF(P) = \bigcup_{C \in P} SF(C)$.

Definition 5.7 An EGHP P is called **E**-*stratified* iff $SF(P)$ is OD-stratified.

Definition 5.8 An EGHP P is called **A**-*stratified* iff P is **E**-stratified and for every ground atom A, $\ell(A) = \ell(\neg A)$.

Clearly, every **A**-stratified EGHP is also **E**-stratified; the latter also implies that the EGHP is OD-stratified. However, OD-stratification does not necessarily imply **E**-stratification, and **E**-stratification does not necessarily imply **A**-stratification.

Example 5.4 Let P be:

$$b \leftarrow a \,\&\, \mathbf{not}\, \neg a.$$

P is OD-stratified by $\ell(\neg a) = \ell(\neg b) = \ell(a) = 0$ and $\ell(b) = 1$. $SF(P)$ in this case is:

$$b \leftarrow a \,\&\, \mathbf{not}\, \neg a \tag{9}$$
$$\neg a \leftarrow \neg b \,\&\, \mathbf{not}\, \neg a \tag{10}$$

$SF(P)$ is not OD-stratified because that would require $\ell(\neg a) < \ell(\neg a)$. Hence, P is not **E**-stratified.

Example 5.5 Let P be:

$$\neg b \leftarrow a \,\&\, \mathbf{not}\, b.$$

Now $SF(P)$ is:

$$\neg b \leftarrow a \,\&\, \mathbf{not}\, b \tag{11}$$
$$\neg a \leftarrow b \,\&\, \mathbf{not}\, b \tag{12}$$

$SF(P)$ is OD-stratified by $\ell(a) = \ell(b) = 0$, $\ell(\neg a) = \ell(\neg b) = 1$. Hence, P is **E**-stratified. However, P is not **A**-stratified because any level mapping ℓ must have $\ell(b) < \ell(\neg b)$.

Now we show how stratification provides a sufficient condition for stability in our framework of non-monotonic negation within inconsistent knowledge bases.

Theorem 4 If P is a function-free **E**-stratified EGHP, then P has a unique **E**-answer set.

Proof. By the hypothesis, there is a level mapping ℓ for $SF(P)$ such that for every clause $C \in SF(P)$, $\ell(L_i) \leq \ell(L)$ and $\ell(L') < \ell(L)$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$, where C is written in the standard form (cf. Definition 5.1). Let $S_0, \ldots, S_n$ be the strata generated by this mapping and for $0 \leq i \leq n$, let $T_i = \{L \mid \ell(L) = i\}$. We construct an **E**-answer set M as follows:

$$M_0 = \{L \in T_0 \mid S_0 \vdash_\exists L\};$$
$$M_{i+1} = \{L \in T_{i+1} \mid G(S_i, \bigcup_{j=0}^{i} M_j) \vdash_\exists L\} \text{ for } 1 \leq i \leq n;$$
$$M = \bigcup_{i=0}^{n} M_i.$$

We need to show that M is an **E**-answer set. We start by observing that for every literal L and set of literals V, $G(P,V) \vdash_\exists L$ iff $SF(G(P,V)) \vdash_\exists L$. This is so because every clause in $SF(G(P,V))$ is logically equivalent to some clause in $G(P,V)$. Now, note that $SF(G(P,V)) = G(SF(P),V)$ because the non-monotonically negated literals are not modified by SF. Hence, $G(P,V) \vdash_\exists L$ iff $G(SF(P,V)) \vdash_\exists L$. To show that M is an **E**-answer set, we must obtain $M = \{L \mid G(P,M) \vdash_\exists L\}$, or by the previous discussion, $M = \{L \mid G(SF(P),M) \vdash_\exists L\}$. But the **E**-stratification of P implies that

$$G(SF(P),M) = \bigcup_{i=0}^{n} G(S_i, \bigcup_{j=0}^{i-1} M_j)$$

where $\bigcup_{j=0}^{i-1} M_j = \emptyset$, because for every clause in strata i, the non-monotonically negated clauses cannot be added to M at any level greater than or equal to i. The result follows from the construction of M and the fact that if $G(SF(P),M) \vdash_\exists L$, then there must be a clause in $SF(P)$ with L as the head.

We still need to show that M is the unique **E**-answer set for P. Suppose M' is any **E**-answer set for P. We show that $M = M'$ by showing that $M_i = M'_i$ for all $1 \leq i \leq n$ where $M'_i = M' \cap \{L \mid \ell(L) = i\}$.

Base Case. ($i = 0$) In this case, for every clause in $SF(P)$ in stratum S_0, there are no occurrences of **not**. Hence, $G(S_0, M_0) = G(S_0, M')$. Thus, $\mathbf{E}_P(M_0) = \mathbf{E}_P(M'_0)$. As M_0 and M'_0 must be **E**-answer sets for S_0, $M_0 = \mathbf{E}_P(M_0) = \mathbf{E}_P(M'_0) = M'_0$.

Inductive case. ($i > 0$) Assume that $M_j = M'_j$ for all $j < i$. By the **E**-stratification of P, $G(S_i, M_i) = G(S_i, M'_i)$ and then by reasoning similar to the base case, $M_i = \mathbf{E}_P(M_i) = \mathbf{E}_P(M'_i) = M'_i$. □

E-stratification is a sufficient, but not a necessary condition for an EGHP to have a unique **E**-answer set. In particular, the program of Example 5.3 is not **E**-stratified, but it has $\{b, \neg b\}$ as its unique **E**-answer set. The next example shows that **E**-stratification is not a sufficient condition for an EGHP to have an **A**-answer set.

Example 5.6 Consider the program:

$$a \leftarrow \mathbf{not}\, b \tag{13}$$
$$b \leftarrow \tag{14}$$
$$\neg b \leftarrow \mathbf{not}\, a \tag{15}$$

This program is **E**-stratified with $\ell(b) = \ell(\neg a) = 0$, $\ell(a) = 1$, $\ell(\neg b) = 2$. Here, $P = SF(P)$. However, there is no **A**-answer set. Note that this program is not **A**-stratified.

Theorem 5 If P is a function-free A-stratified EGHP, then P has a unique A-answer set.

Proof. The construction of the A-answer set is similar to the construction in Theorem 4 except for the substitution of $\vdash_\forall$ instead of $\vdash_\exists$. The key point in showing that M is an A-answer set is that for every ground atom A, since $\ell(A) = \ell(\neg A) = i$ for some level i, at that level either A is placed into M_i or $\neg A$ is placed into M_i or neither A nor $\neg A$ is placed into M_i. By the definition of A-stratification, it is impossible to add A at some level and $\neg A$ at another level. □

Example 5.1 shows that A-stratification is not a necessary condition for the existence of an A-answer set. The program of Example 5.1 is not A-stratified, but it has $\emptyset$ as its A-answer set.

6 Summary and Discussion

We have developed two semantics for inconsistent knowledge bases. Both semantics are based on the maximal consistent subsets of the inconsistent knowledge base. The cautious semantics accepts those statements which are true in all maximal consistent subsets, while the optimistic semantics accepts those statements which are true in at least one maximal consistent subset. We study various properties of these semantics and develop a Kripke-style model theory for the optimistic semantics. Finally, we extend our approach to include non-monotonic negation. Within this framework, we extend the concepts of stratification and stability from logic programming and show that stratification provides a sufficient condition for stability.

Reasoning with inconsistency in logic programs was first studied by Blair and Subrahmanian [2] whose work was subsequently expanded by Kifer and Lozinskii [9]. These works were grounded in multivalued logics. There are two significant differences between those approaches and that studied in this paper: first, when a database DB is consistent, the semantics of [2,9] may not always agree with the classical logic meaning of DB; both the optimistic and cautious approaches described here would agree with classical logic when DB is classically consistent. Second, the work described here includes support for non-monotonic negation via a stable model semantics. No support for non-monotonic negation was present in [2,9], though [9] discusses some ways non-monotonicity may occur. In particular, we present here, two kinds of stratification. Neither [2,9] did this.

A structure similar to maximal consistent subsets arises in the context of database updates [3,4]. Given a database DB and a new fact, f, to be inserted into the database, Fagin et. al. [3,4] define a *flock* to be the set of maximal consistent subsets of $DB \cup \{f\}$ that are supersets of $\{f\}$. In other words, priority is given to f over formulas in DB. This does not occur in our framework.

Finally, the work reported in this paper has been used as the formal theoretical basis for combining multiple knowledge bases [1].

Acknowledgements. This work has been supported by the Army Research Office under Grant Number DAAL-03-92-G-0225, by the Air Force Office of Scientific Research under Grant Number F49620-93-1-0065, and by the National Science Foundation under Grant Numbers IRI-9200898, IRI-9109755 and IRI-9357756. The work was also supported in part by ARPA/Rome Labs Contract F30602-93-C-0241 (ARPA Order Nr. A716).

References

[1] C. Baral, S. Kraus, J. Minker and V.S. Subrahmanian (1992) *Combining Knowledge Bases Consisting of First Order Theories*, Computational Intelligence, 8, 1, pps 45–71.

[2] H.A. Blair and V.S. Subrahmanian. (1989) *Paraconsistent Logic Programming*, Theoretical Computer Science, Vol. 68, pp 135–154. Preliminary version in: Lecture Notes in Computer Science, Vol. 287, Dec. 1987.

[3] R. Fagin, G. Kuper, J. Ullman, and M. Vardi. Updating Logical Databases. In *Advances in Computing Research*, volume 3, pages 1–18, 1986.

[4] R. Fagin, J.D. Ullman, and M.Y. Vardi. On the Semantics of Updates in Databases. In *ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 352–365, 1983.

[5] M. Ginsberg. (1986) *Counterfactuals*, Artificial Intelligence.

[6] M. Gelfond and V. Lifschitz. (1988) *The Stable Model Semantics for Logic Programming*, in Proc. of the 5th Intl. Conf./Symp. on Logic Programming, pp 1070–1080, MIT Press.

[7] M. Gelfond and V. Lifschitz. (1990) *Logic Programs with Classical Negation*, in: Proc. of the 7th Intl. Conf. on Logic Programming, pp 579–597, MIT Press.

[8] J. Grant and V.S. Subrahmanian. (1995) Reasoning In Inconsistent Knowledge Bases, IEEE Trans. on Knowledge and Data Engineering, volume 7, pp 177-189.

[9] M. Kifer and E.L. Lozinskii. (1989) *RI: A Logic for Reasoning with Inconsistency*, 4-th Symposium on Logic in Computer Science, Asilomar, CA, pp. 253-262.

[10] R. Kowalski and F. Sadri. (1990) *Logic Programs with Exceptions*, in: Proc. 7th Intl. Conf. on Logic Programming, pp 598–613.

[11] J. Minker and A. Rajasekar. (1990) *A Fixpoint Semantics for Non-Horn Logic Programs*, J. of Logic Programming.

[12] N. Rescher. (1964) *Hypothetical Reasoning*, North-Holland.

Appendix

Formalization of the Murder Example

The various facts relating to the murder mystery are described below:

1. $present(al)$
2. $present(carl)$
3. $present(ed)$
4. $present(dick)$
5. $in\_hospital(tom)$
6. $\neg present(X) \leftarrow in\_hospital(X)$

7. $suspect(X) \leftarrow present(X)$
8. $\neg suspect(X) \leftarrow \neg present(X)$
9. $suspect(X) \leftarrow embezzler(X)$
10. $suspect(X) \leftarrow having\_affair(X)$
11. $embezzler(al)$
12. $having\_affair(carl)$
13. $in\_bathroom(carl)$
14. $in\_bathroom(al)$
15. $\neg in\_bathroom(Y) \leftarrow in\_bathroom(X)\ \&\ X \neq Y$
16. $carl \neq al$
17. $al \neq carl$
18. $in\_living\_room(dick)$
19. $in\_living\_room(ed)$
20. $\neg in\_kitchen(X) \leftarrow in\_bathroom(X)$
21. $\neg in\_kitchen(X) \leftarrow in\_living\_room(X)$
22. $murderer(al) \leftarrow \neg murderer(carl)\&\neg murderer(ed)\&\neg murderer(dick)$
23. $murderer(carl) \leftarrow \neg murderer(al)\&\neg murderer(ed)\&\neg murderer(dick)$
24. $murderer(ed) \leftarrow \neg murderer(carl)\&\neg murderer(al)\&\neg murderer(dick)$
25. $murderer(dick) \leftarrow \neg murderer(carl)\&\neg murderer(ed)\&\neg murderer(al)$
26. $suspect(X) \leftarrow murderer(X)$
27. $\neg murderer(X) \leftarrow murderer(Y)\ \&\ Y \neq X.$
28. $\neg murderer(X) \leftarrow in\_bathroom(X)$
29. $\neg murderer(X) \leftarrow in\_living\_room(X)$
30. AXIOMS SAYING that Carl, Al, Ed, and Tom are not equal.

Received June, 1994

Acta Cybernetica, Vol. 12, No. 1, Szeged, 1995

Reconstruction of Unique Binary Matrices with Prescribed Elements*

A. Kuba †

Summary

The reconstruction of a binary matrix from its row and column sum vectors is considered when some elements of the matrix may be prescribed and the matrix is uniquely determined from these data. It is shown that the uniqueness of such a matrix is equivalent to the impossibility of selecting certain sequences from the matrix elements. The unique matrices are characterized by several properties. Among others it is proved that their rows and columns can be permutated such that the 1's are above and left to the (non-prescribed) 0's. Furthermore, an algorithm is given to decide if the given projections and prescribed elements determine a binary matrix uniquely, and, if the answer is yes, to reconstruct it.

1 Introduction

Let $A = (a_{ij})$ be a binary matrix of size $m \times n$. Let its *row sum vector* be denoted by $R(A) = R = (r_1, r_2, \ldots, r_m)$,

$$r_i = \sum_{j=1}^{n} a_{ij}, \qquad (i = 1, 2, \ldots, m),$$

and let its *column sum vector* be denoted by $S(A) = S = (s_1, s_2, \ldots, s_n)$,

$$s_j = \sum_{i=1}^{m} a_{ij}, \qquad (j = 1, 2, \ldots, n).$$

The vectors R and S are also called the *projections* of A. Denote the *class* of binary matrices with row sum vector R and column sum vector S by $\mathcal{A}(R, S)$.

The problem of reconstruction of binary matrices from their projections has an extensive literature (for surveys, see e.g. [14] and [4]). Gale [9] and Ryser [13] have proved existence conditions. A necessary and sufficient condition of uniqueness is, for example, in [15].

In this paper, a generalization of the mentioned reconstruction problem will be considered. Let P and Q be binary matrices with size $m \times n$. We say $Q \geq P$ or Q

*This work was supported by the OTKA grant 3195 and the NSF-MTA grant INT91-21281

†Department of Applied Informatics, József Attila University, H-6720 Szeged, Árpád tér 2., Hungary, Phone: +36-62-310011, Fax: +36-62-312292

covers P if $q_{ij} \geq p_{ij}$ for all positions $(i,j) \in \{1,2,\ldots,m\} \times \{1,2,\ldots,n\}$. The class $\mathcal{A}_P^Q(R,S)$ is then defined as

$$\mathcal{A}_P^Q(R,S) = \{A \mid A \in \mathcal{A}(R,S), \quad P \leq A \leq Q\}.$$

According to this definition, $\mathcal{A}_P^Q(R,S)$ can be regarded as the sub-class of $\mathcal{A}(R,S)$ having the prescribed value 1 in the positions where $p_{ij} = q_{ij} = 1$, and the prescribed value 0 where $p_{ij} = q_{ij} = 0$. It is clear that, if $P = O$ (zero matrix) and $Q = E(= (1)_{m\times n})$, then $\mathcal{A}_O^E(R,S) = \mathcal{A}(R,S)$.

Now, we show that this reconstruction problem can be simplified. It is clear that, if $A \in \mathcal{A}_P^Q(R,S)$ then $A \geq P$, so their *difference*, $A-P = (a_{ij}-p_{ij})_{n\times m}$, is a binary matrix with projections $R(A-P) = R(A) - R(P) = R - R(P)$ and $S(A-P) = S(A) - S(P) = S - S(P)$. Therefore, $A - P \in \mathcal{A}_O^{Q-P}(R - R(P), S - S(P))$. The reverse statement is also true in the sense that, if $B \in \mathcal{A}_O^Q(R,S)$ for some binary matrix Q, then $B + P \in \mathcal{A}_P^{Q+P}(R + R(P), S + S(P))$, where P is a binary matrix such that for all positions, if $p_{ij} = 1$, then $q_{ij} = 0$. This means that it is enough to study the class $\mathcal{A}_O^Q(R,S)$, or in short $\mathcal{A}^Q(R,S)$ or $\mathcal{A}^Q$.

It is interesting to note that the network flows [7] can also be used in the study of the class $\mathcal{A}^Q(R,S)$. To each class $\mathcal{A}^Q(R,S)$ there is a bipartite network with source s, sink t and nodes $\{R_1, R_2, \ldots, R_m\}$, $\{S_1, S_2, \ldots, S_n\}$ and arcs (s, R_i), (S_j, t) and (R_i, S_j) with capacity r_i, s_j and q_{ij}, respectively, $i = 1,2,\ldots,m$, $j = 1,2,\ldots,n$. Then each matrix $A \in \mathcal{A}^Q(R,S)$ corresponds to a flow in this network (see [6]). In this way, the results in this paper have a reformulation in network flows.

Considering the connected literature, Kellerer published a necessary and sufficient condition [11] for the existence of measurable functions with given "marginals" which is applicable also to the matrices in the class $\mathcal{A}^Q$. Recently W.Y.C. Chen has published theorems about integral matrices with given row and column sums satisfying a so-called main condition [6]. However, this main condition restricts the validity of the results only to a part of the prescribed binary matrices. As we shall see, there is unique binary matrix not satisfying Chen's main condition (e.g., the only binary matrix of the so-called normalized class corresponding to Fig. 5.1). There are papers dealing with special $\mathcal{A}^Q$ classes: Fulkerson gave a necessary and sufficient condition for the existence of (0,1)-matrices with zero trace [8] and Anstee published results on matrices having at most one prescribed position in their columns [1],[3] and having a triangular block of 0's [2].

Henceforth, consider the class $\mathcal{A}^Q(R,S)$ where $R = (r_1, r_2, \ldots, r_m)$ and $S = (s_1, s_2, \ldots s_n)$ are non-negative integer vectors and Q is a binary matrix of size $m \times n$. The position (i,j) is said to be *free* if the corresponding matrix element is not prescribed by Q, i.e. $q_{ij} = 1$.

In this paper, the aim is to generalize the uniqueness results of $\mathcal{A}$ to $\mathcal{A}^Q$ (and thus, to $\mathcal{A}_P^Q$). (The reconstruction problems of non-uniquely determined binary matrices is the subject of [10].) In Section 2 we reconsider the known results of uniqueness in certain classes $\mathcal{A}^Q(R,S)$, where Q has some special property. Then the general uniqueness problem is considered, when Q is an arbitrary binary matrix. Section 3 contains a definition of a switching chain, whose existence turns out to be a necessary and sufficient condition of the non-uniqueness of a binary matrix. Thus, a switching chain has the same role in the class $\mathcal{A}^Q$ as a switching component has in $\mathcal{A}$. In Section 4 a reconstruction algorithm is given to decide if the given projections and prescribed elements determine a binary matrix uniquely, and, if

the answer is yes, to reconstruct it. The unique matrices can be characterized in different ways. Some of these properties are discussed in Section 5. It is proved that the 1's of these matrices can be covered by certain rectagles, and that their rows and columns can be permuted so that the 1's are above and to the left of the (non-prescribed) 0's.

2 Uniqueness in special classes

In this section we reconsider the uniqueness results in different special classes proving that none of them is sufficient to characterize the uniqueness in the class $\mathcal{A}^Q$.

We say that $A \in \mathcal{A}^Q(R,S)$ is a *non-unique* (or *ambiguous*) binary matrix (in $\mathcal{A}^Q$) if there is a matrix $A' \in \mathcal{A}^Q(R,S)$ such that $A \neq A'$. In the other case, A is *unique* (or *unambiguous*). Accordingly, the *reconstruction data*, the projections (R,S) and the prescribed values Q together, is *non-unique* or *unique* if the number of elements of the class $\mathcal{A}^Q$ is greater than one or exactly one, respectively. If $\mathcal{A}^Q(R,S) = \emptyset$ then the reconstruction data is *inconsistent*.

There are results connected with the uniqueness in the class $\mathcal{A}(R,S)$, i.e. when $Q = E$: Consider the matrices

$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } A_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

An *interchange* is a transformation of the (free) elements of A that changes a minor of type A_1 into type A_2 or vice versa, and leaves all other elements of A unaltered. (The word minor is used here in the sense of submatrix.) We say that the four elements of the minor form a *switching component*.

Theorem 2.1 [13,15]. The binary matrix $A \in \mathcal{A}(R,S)$ is ambiguous (in $\mathcal{A}(R,S)$) if and only if it has a switching component.

In the more general class of $\mathcal{A}^Q(R,S)$, the extension of this result is not trivial. Consider, for example, the class $\mathcal{A}^Q((1,1,1),(1,1,1))$, where

$$Q = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix},$$

that is, the diagonal elements are prescribed. The matrices $A_3, A_4 \in \mathcal{A}^Q$ (see Fig. 2.1), but they have no switching components.

$$A_3 = \begin{pmatrix} \mathbf{x} & 0 & 1 \\ 1 & \mathbf{x} & 0 \\ 0 & 1 & \mathbf{x} \end{pmatrix} \qquad A_4 = \begin{pmatrix} \mathbf{x} & 1 & 0 \\ 0 & \mathbf{x} & 1 \\ 1 & 0 & \mathbf{x} \end{pmatrix}$$

Figure 2.1. Ambiguous matrices A_3 and A_4 having no switching components (x's denote the positions of the prescribed 0 elements).

The matrices A_3 and A_4 play a similar role in the classes of binary matrices having at most one prescribed element in each column as A_1 and A_2 do in $\mathcal{A}$ (classes having no prescribed element). Replacing a submatrix A_3 by A_4 or vice versa leaves the row and column sums unchanged. A *triangle interchange* is a replacement of any version of A_3 and A_4 obtained by applying the same row and column permutations to both A_3 and A_4 [1]. Anstee proved an analogous theorem [1 Corollary 3.2] in the case of prescribed 1's:

Theorem 2.2. Given a pair $A, B \in \mathcal{A}^Q(R,S)$, where Q has at most one 0 in each column, one can get from A to B by a series of interchanges and triangle interchanges without leaving $\mathcal{A}^Q(R,S)$.

However, if there is more than one prescribed element in the columns and rows, then the minors A_1, A_2, A_3 and A_4 are not enough to characterize uniqueness. For example, the matrices of Figure 2.2 are in the same class, but they have no such minors of free elements.

$$\begin{pmatrix} 0 & 1 & x & x \\ x & 0 & 1 & x \\ x & x & 0 & 1 \\ 1 & x & x & 0 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & x & x \\ x & 1 & 0 & x \\ x & x & 1 & 0 \\ 0 & x & x & 1 \end{pmatrix}$$

Figure 2.2. Ambiguous binary matrices having two prescribed elements in each row and column, and having no minors A_1, A_2, A_3, A_4, or any minors obtained from them by permuting rows and columns.

3 Switching chain

Our most important new concept is a generalization of the concept of a switching component. We say that the binary matrix $A \in \mathcal{A}^Q$ has a *switching chain* if there is a series of different free positions of A, $< (i_1,j_1), (i_1,j_2), (i_2,j_2), (i_2,j_3), \ldots, (i_p,j_p), (i_p,j_1) >$, such that

$$a_{i_1j_1} = a_{i_2j_2} = \ldots = a_{i_pj_p} =$$

$$= 1 - a_{i_1j_2} = 1 - a_{i_2j_3} = \ldots = 1 - a_{i_pj_1}$$

$(p \geq 2)$. It follows from the definition that if $< (i_1,j_1), (i_1,j_2), (i_2,j_2), (i_2,j_3), \ldots, (i_p,j_p), (i_p,j_1) >$ is a switching chain of A and $a_{i_1j_1} = a_{i_2j_2} = \ldots = a_{i_pj_p} = 1$, then $a_{i_1j_2} = a_{i_2j_3} = \ldots = a_{i_pj_1} = 0$. This statement remains true if we switch the 1's and 0's of the chain. As examples of switching chain see A_1, A_2, A_3, A_4 and the matrices of Figure 2.2. Each of them contains switching chains. (In fact a switching component is a switching chain with $p = 2$.)

An important property is that by switching the 1's and 0's of a switching chain in a matrix, another matrix is obtained that has the same projections. Therefore, the non-existence of a switching chain in a matrix is a necessary condition for uniqueness. In fact, it is also sufficient.

Theorem 3.1. The binary matrix $A \in \mathcal{A}^Q(R,S)$ is unique if and only if A has no switching chain.

Proof. One direction is obvious. For the other direction, let us suppose that there is another binary matrix $A' \in \mathcal{A}^Q(R,S)$ $(A' \neq A)$. Then, there is a position (i_1, j_1) such that

$$a_{i_1 j_1} = 1, \quad a'_{i_1 j_1} = 0$$

(or $a_{i_1 j_1} = 0$, $a'_{i_1 j_1} = 1$, in which case we can use a similar proof). Since $r_{i_1} = r'_{i_1}$, there is a column j_2 $(\neq j_1)$ such that

$$a_{i_1 j_2} = 0, \quad a'_{i_1 j_2} = 1.$$

Then, since $s_{j_2} = s'_{j_2}$, there is a row i_2 such that

$$a_{i_2 j_2} = 1, \quad a'_{i_2 j_2} = 0,$$

and so on. After a finite number of steps the sequence will terminate, i.e., it follows from

$$a_{i_p j_p} = 1, \quad a'_{i_p j_p} = 0$$

that there is a column among (the up-to-now all different) $j_1, j_2, \ldots, j_p$, say j_k, such that

$$a_{i_p j_k} = 0, \quad a'_{i_p j_k} = 1.$$

That is, $< (i_k, j_k), (i_k, j_{k+1}), (i_{k+1}, j_{k+1}), (i_{k+1}, j_{k+2}), \ldots, (i_p, j_p), (i_p, j_k) >$ is a switching chain in A.

Remark. The proof is almost the same in the case of switching components in class $\mathcal{A}$ (see [13] and [15]), but in $\mathcal{A}$ it is also shown that this switching chain can be used to find a switching component. In the class $\mathcal{A}^Q$, this is not necessarily true.

4 Reconstruction of unique matrices

Now, we give the characterization that can be used to decide the uniqueness and to reconstruct unique matrices efficiently. We say that a minor is *mixed* if each of its rows and columns contains both a free 1 and a free 0.

Theorem 4.1. The binary matrix A is unique if and only if it has no mixed minor.

Proof. If there is a switching chain in a binary matrix, then the rows and the columns of the switching chain determine a minor consisting of rows and columns each containing free 1's and 0's.

To prove the other direction, let us suppose that A has a mixed minor. Then let $a_{i_1 j_1} = 1$ be an element of the mixed minor. There is a column j_2 such that $a_{i_1 j_2} = 0$ is an element of the mixed minor. Then, there is a row i_2 and a column j_3 such that $a_{i_2 j_2} = 1$, $a_{i_2 j_3} = 0$ and they are in the minor. We have to continue the procedure until there is a row i_p and a column j_p such that $a_{i_p j_p} = 1$, $a_{i_p j_k} = 0$ (both in the minor), where $j_k \in \{j_1, j_2, \ldots, j_{p-1}\}$. Then $< (i_k, j_k), (i_k, j_{k+1}), (i_{k+1}, j_{k+1}), (i_{k+1}, j_{k+2}), \ldots, (i_p, j_p), (i_p, j_k) >$ is a switching chain.

From Theorem 4.1 it follows for each minor of a unique matrix that there is a row or column of the minor such that in that row or column either there are only 1's in the free positions, or there are only 0's in the free positions or there are no free positions at all. These rows/columns are called *primitive rows/columns* of the minor. A primitive row/column can be recognised from the number of the free positions and the projection values of the minor in the following way. A primitive row contains 0's in the free positions (if there is free position) if and only if the sum of that row/column of the minor is 0. A primitive row contains 1's in the free positions if and only if the sum of that row/column is equal to the number of the free positions in that row/column of the minor.

Similarly, we say that i is a *primitive row of* $\mathcal{A}^Q(R,S)$ if $0 = r_i$ or $r_i = \sum_{j=1}^{n} q_{ij}$ and that j is a *primitive column of* $\mathcal{A}^Q(R,S)$ if $0 = s_j$ or $s_j = \sum_{i=1}^{m} q_{ij}$.

If the class $\mathcal{A}^Q(R,S)$ has only one matrix, then it has a primitive row or column. By reducing R and S by the projection of a primitive row or column and setting Q to 0 in this row or column, the new class $\mathcal{A}^{Q'}(R',S')$, has also only one matrix having the same elements as the original one in the positions $q'_{ij} = 1$. Trivially, if $\mathcal{A}^Q(R,S)$ is non- unique or empty, the $\mathcal{A}^{Q'}(R',S')$ is also non-unique or empty, respectively.

From this property of the unique binary matrices a reconstruction algorithm follows:

Algorithm 4.1 (to decide the uniqueness of the reconstruction data and to reconstruct a unique matrix $A \in \mathcal{A}^Q(R,S)$ from given projections R and S, and prescribed positions of Q):

Step 1. Let $A := O$, $R' := R$, $S' := S$, $Q' := Q$.

Step 2. If $0 \le r'_i \le \sum_{j'=1}^{n} q'_{ij'}$ and $0 \le s'_j \le \sum_{i'=1}^{m} q'_{i'j}$ is not fulfilled for all i and j, then the reconstruction data is inconsistent; stop.

Step 3. If $Q' = O$, then output A; stop.

Step 4. If no row and no column of $\mathcal{A}^{Q'}(R',S')$ is primitive, then the reconstruction data is non-unique or inconsistent; stop.

Step 5. Select a primitive row or column of $\mathcal{A}^{Q'}(R',S')$. For every (i,j) in this row or column such that $q'_{ij} = 1$,
i. set a_{ij} equal to 0 or 1, appropriately;
ii. reduce r'_i and s'_j by a_{ij};
iii. set q'_{ij} to 0.
Go to Step 2.

Remarks.

a. It is supposed that m and n are positive integers and R and S are vectors of m and n non-negative integers, respectively.

b. During the iterations the number of 0-rows or the number of 0-columns of Q' is increases at least by one. Thus, the algorithm will terminate after at most $m + n - 1$ number of iterations, when all rows or columns of Q' contain only 0's (Step 3).

c. Step 2 is to test two conditions: The first is that vectors R' and S' contain only non-negative elements, and the second that the number of free positions in each row and column of the reduced class are enough to place r'_i and s'_j number of 1's, respectively. Both conditions are necessary for the existence.

d. Step 4 is to test if there is a primitive row or column in the class $\mathcal{A}^{Q'}(R', S')$. If no, then the matrix to be reconstructed has a mixed minor (see Theorem 4.1) consisting of the non-0-rows and non-0-columns of Q' (if the matrix exists at all).

e. It is not difficult to prove that the matrix A reconstructed by Algorithm 4.1 as an output in Step 3 is unique. It follows from the fact that primitive rows and columns do not contain any element of any switching chain.

f. Clearly, if a matrix A is constructed by Algorithm 4.1 then $A < Q$, because we assign 1's only into free positions (Step 5).

g. If the number of 1's in row i of A increases during the iterations, then r'_i decreases by the same number. This means that $r'_i + \sum_{j=1}^{n} a_{ij}$ remains constant in each iteration. In the first iteration this constant is

$$r'_i + \sum_{j=1}^{n} a_{ij} = r_i \tag{4.1}$$

(because $\sum_{j=1}^{n} a_{ij} = 0$ now). If we arrive Step 3 such that $Q' = O$ then $r'_i = 0$ and $s'_j = 0$ for each i and j (Step 2), and so again $R(A) = R$. Similarly, it can be shown that $S(A) = S$. That is, if a matrix A is constructed by Algorithm 4.1 then $A \in \mathcal{A}(R, S)$.

h. Algorithm 4.1 can be considered as a generalization of the assign and update algorithm [5] for reconstructing unique matrices without prescribed elements.

Therefore, Algorithm 4.1 is correct in the sense that it is terminated after a finite number of steps (Remarks b. and c.), the output matrix A is unique (Remark e.) and it is from the class $\mathcal{A}^{Q}(R, S)$ (Remarks f. and g.).

As an example see Figure 4.1.

2	x	.	.	.	.
2	.	x	.	.	x
4	.	.	.	x	.
1	.	.	.	.	.
2	.	x	.	.	x
	1	2	5	2	1

a.

1	x	.	1	.	.
1	.	x	1	.	x
0	1	1	1	x	1
0	.	.	1	.	.
1	.	x	1	.	x
	0	1	0	2	0

b.

1	x	.	1	.	0
1	0	x	1	.	x
0	1	1	1	x	1
0	0	0	1	0	0
1	0	x	1	.	x
	0	1	0	2	0

c.

0	x	1	1	.	0
0	0	x	1	1	x
0	1	1	1	x	1
0	0	0	1	0	0
0	0	x	1	1	x
	0	0	0	0	0

d.

0	x	1	1	0	0
0	0	x	1	1	x
0	1	1	1	x	1
0	0	0	1	0	0
0	0	x	1	1	x
	0	0	0	0	0

e.

Figure 4.1. Reconstruction of a unique binary matrix by Algorithm 4.1 showing matrix A and projections R' and S' during the iterations. The free elements of the minor to be reconstructed are denoted by '.'. The reconstructed elements of A are denoted by 0 and 1. The matrix Q' has a 1 at the positions where there is a '.'

a. Starting configuration.
b. Configuration after finding the primitive column 3 and primitive row 3.
c. Configuration after finding the primitive columns 1, 5 and primitive row 4.
d. Configuration after finding the primitive column 2 and primitive rows 2, 5.
e. Configuration after finding the primitive column 4.

5 Characterization of unique matrices

Knowing Theorems 3.1 and 4.1 the unique matrices can be characterized by having no switching chain or having no mixed minor. Another possible characterizations are based on the comparison of the prescribed and free 1 and 0 positions of the rows. Let us introduce the following notations in connection with a matrix $A \in \mathcal{A}^Q$:

$$A^{(1)} = \{(i,j) \mid a_{ij} = 1\}, \quad A^{(0)} = \{(i,j) \mid a_{ij} = 0, q_{ij} = 1\}$$

and

$$Q^{(0)} = \{(i,j) \mid q_{ij} = 0\}.$$

In words, $A^{(1)}$ and $A^{(0)}$ denotes the sets of the free 1 and 0 positions of the binary matrix A, respectively, and $Q^{(0)}$ denotes the set of prescribed positions. Furthermore, let $A_i^{(1)}$ and $A_i^{(0)}$ denote the set of column indices of the free 1's and free 0's of A in row i $(1 \leq i \leq m)$, respectively.

Theorem 5.1. The binary matrix $A \in \mathcal{A}^Q(R,S)$ is unique if and only if for any subset I of the rows there is a row $i \in I$ such that

$$A_i^{(0)} \cap A_{i'}^{(1)} = \emptyset \tag{5.1}$$

for each $i' \in I$.

Remarks.

a. In another words, Theorem 5.1 says that, exactly in the case of uniqueness, from any subset of rows we can select at least one row such that in the columns of the free 0's of this row there is no 1 in any other row. This means that the 1's and prescribed elements of the selected row "cover" all the 1's of the other rows in the subset. In this sense the selected row is a longest row of the subset.

b. Specially, if there is no prescribed element, i.e. $Q = E$, then (5.1) means that a row having the greatest r_i covers every other row.

Proof. Suppose that A has a switching chain $SC = < (i_1,j_1),(i_1,j_2),(i_2,j_2),(i_2,j_3),\ldots,(i_p,j_p),(i_p,j_1) >$ such that $a_{i_1j_1} = a_{i_2j_2} = \ldots = a_{i_pj_p} = 1$ and $a_{i_1j_2} = a_{i_2j_3} = \ldots = a_{i_pj_1} = 0$. Then let $I = \{i_1, i_2, \ldots, i_p\}$. If i_k is an arbitrary row of I $(1 \leq k \leq p)$, then i_{k+1} is another row of I such that $j_{k+1} \in A_{i_k}^{(0)} \cap A_{i_{k+1}}^{(1)}$ (if $k = p$ then instead of i_{k+1} let us select i_1). That is (5.1) is not fulfilled.

Suppose, now, that there is a subset of rows, I, such that for each row $i \in I$, there is a row $i' \in I$ such that $A_i^{(0)} \cap A_{i'}^{(1)} \neq \emptyset$. Let $i_1 \in I$ and i_2 another row index from I such that $j_2 \in A_{i_1}^{(0)} \cap A_{i_2}^{(1)}$ for some j_2, that is, $a_{i_1j_2} = 0$ and $a_{i_2j_2} = 1$. Applying the same condition to row i_2 we get a row i_3 from I and a column j_3 such that $a_{i_2j_3} = 0$ and $a_{i_3j_3} = 1$. And so on. After a finite number of steps the sequence will be ended, i.e. $a_{i_pj_k} = 0$ and $a_{i_kj_k} = 1$ for some $i_k \in \{i_1, i_2, \ldots, i_{p-1}\}$ and $j_k \in \{j_1, j_2, \ldots, j_{p-1}\}$. Then $< (i_k,j_k),(i_k,j_{k+1}),(i_{k+1},j_{k+1}),(i_{k+1},j_{k+2}),\ldots,(i_p,j_p),(i_p,j_k) >$ is a switching chain in A.

Now, we give another characterization of the unique matrices by proving that their 1's can be covered by special rectangles. The construction of these covering rectangles can be done by

Procedure 5.1 (to construct special covering rectangles of 1's): This is an inductive procedure to find a sequence of rectangles having increasing number of rows and decreasing number of columns step by step. Applying Theorem 5.1 to the whole set of rows we know that if A is unique, then we can select at least one row i such that in the columns of $A_i^{(0)}$ A has no 1 element. Let the set of such rows be denoted by $I_1^{(1)} (\neq \emptyset)$, and let

$$J_1^{(1)} = \bigcap_{i \in I_1^{(1)}} \overline{A_i^{(0)}}$$

(overline denotes the complement set). Clearly, $A^{(1)} \supseteq (I_1^{(1)} \times J_1^{(1)}) \setminus Q^{(0)}$. If $A^{(1)} = (I_1^{(1)} \times J_1^{(1)}) \setminus Q^{(0)}$ then we have a rectangle (in a general sense that $I_1^{(1)} \times J_1^{(1)}$ consists of not necessarily consecutive rows and columns) covering the 1's of A and the Procedure is terminated. If

$$A^{(1)} \supset \bigcup_{t=1}^{p} (I_t^{(1)} \times J_t^{(1)}) \setminus Q^{(0)}$$

for some $p > 1$ (the symbols $\supset$ and $\subset$ are used only for strict containment) then we can select at least one row i from $\overline{I_p^{(1)}}$ such that A has no 1 element in $\overline{I_p^{(1)}} \times A_i^{(0)}$. ($A_i^{(0)} \neq \emptyset$, because in this case $i \in I_p^{(1)}$.) Let the union of the set of these rows and $I_p^{(1)}$ be denoted by $I_{p+1}^{(1)}$. Clearly, $I_p^{(1)} \subset I_{p+1}^{(1)}$. Let

$$J_{p+1}^{(1)} = \bigcap_{i \in I_{p+1}^{(1)}} \overline{A_i^{(0)}}.$$

Then $J_p^{(1)} \supset J_{p+1}^{(1)}$ (because $A_i^{(0)} \neq \emptyset$ in the new rows of $I_{p+1}^{(1)}$) and $A^{(1)} \supseteq (I_{p+1}^{(1)} \times J_{p+1}^{(1)}) \setminus Q^{(0)}$. After a finite number of steps (if p is big enough), we reach the situation

$$A^{(1)} = \bigcup_{t=1}^{p} (I_t^{(1)} \times J_t^{(1)}) \setminus Q^{(0)},$$

that is the 1's of matrix A are covered by the union of rectangles $I_t^{(1)} \times J_t^{(1)}$ $(1 \leq t \leq p)$.

As an example of the application of Procedure 5.1 see Figure 5.1(a), where $(\{1\}\times\{1,2,3,4,5,6\})\cup(\{1,2,3\}\times\{1,2,3,4\})\cup(\{1,2,3,4\}\times\{1,3,4\})\cup(\{1,2,3,4,5,6\}\times\{1\})$ is the set of covering rectangles constructed by Procedure 5.1.

4	1	1	x	x	1	1		4	1	x	1	x	1	1
1	1	x	x	x	0	0		1	1	x	x	x	0	0
3	1	1	1	x	0	x		3	1	1	1	x	0	x
2	1	0	1	x	x	x		2	1	1	0	x	x	x
1	1	x	0	0	0	x		1	1	0	x	0	0	x
1	1	x	0	x	x	0		1	1	0	x	x	x	0
	6	2	2	0	1	1			6	2	2	0	1	1
			(a)								(b)			

Figure 5.1. (a) A unique binary matrix and its projections. (b) After changing columns 2 and 3 the matrix is ordered such that the 1's are to the left of the free 0's in each row, and the 1's are above the free 0's in each column.

Remark. Specially, if A has no 1 element (of course, in this case A is uniqe) then Procedure 5.1 gives $\{1,2,\ldots,m\}\times\emptyset$ as the only covering rectangle. In any other case the constructed rectangles are not degenerate.

Procedure 5.1 has proved a part of

Theorem 5.2. The binary matrix $A\in\mathcal{A}^Q(R,S)$ is unique if and only if there are subsets $I_1^{(1)}\subset I_2^{(1)}\subset\ldots\subset I_{p_1}^{(1)}$ of the row-indices $\{1,2,\ldots,m\}$ and subsets $J_1^{(1)}\supset J_2^{(1)}\supset\ldots\supset J_{p_1}^{(1)}$ of the column-indices $\{1,2,\ldots,n\}$ $(p_1\geq 1)$ such that

$$A^{(1)}=\bigcup_{t=1}^{p_1}(I_t^{(1)}\times J_t^{(1)})\setminus Q^{(0)}. \tag{5.2}$$

Proof. If A is unique then we can apply Procedure 5.1 to get the sequence of sets in (5.2).

To prove the other direction let us suppose that A is non-unique, but there are such covering rectangles. Then there is a switching chain $SC=<(i_1,j_1),(i_1,j_2),(i_2,j_2),(i_2,j_3),\ldots,(i_p,j_p),(i_p,j_1)>$ in A. Suppose that $a_{i_1j_1}=0$, $a_{i_1j_2}=1$, $a_{i_2j_2}=0$, $a_{i_2j_3}=1$ and so on. (Otherwise an

analogous proof can be used.) The first two 1-valued elements of SC can not be covered by the same rectangle, because in this case (i_2, j_2) would be covered. Thus, there are two rectangles, say $I_{k_1}^{(1)} \times J_{k_1}^{(1)}$ and $I_{k_2}^{(1)} \times J_{k_2}^{(1)}$ $(1 \leq k_1 < k_2 \leq p_1)$, such that $I_{k_1}^{(1)} \subset I_{k_2}^{(1)}$ (because $i_2 \in I_{k_2}^{(1)} \setminus I_{k_1}^{(1)}$) and $J_{k_1}^{(1)} \supset J_{k_2}^{(1)}$ (because $j_2 \in J_{k_1}^{(1)} \setminus J_{k_2}^{(1)}$). To cover (i_3, j_4) we have another rectangle $I_{k_3}^{(1)} \times J_{k_3}^{(1)}$ such that $I_{k_2}^{(1)} \subset I_{k_3}^{(1)}$ and $J_{k_2}^{(1)} \supset J_{k_3}^{(1)}$. And so on. Finally, to cover (i_p, j_1) we have the rectangle $I_{k_p}^{(1)} \times J_{k_p}^{(1)}$ $(k_{p-1} < k_p \leq p_1)$ such that $I_{k_{p-1}}^{(1)} \subset I_{k_p}^{(1)}$ and $J_{k_{p-1}}^{(1)} \supset J_{k_p}^{(1)}$. Furthermore, $I_{k_p}^{(1)} \subset I_{k_1}^{(1)}$ and $J_{k_p}^{(1)} \supset J_{k_1}^{(1)}$. But, here is the contradiction of $I_{k_1}^{(1)} \subset I_{k_2}^{(1)} \subset \ldots \subset I_{k_p}^{(1)} \subset I_{k_1}^{(1)}$ (and $J_{k_1}^{(1)} \supset J_{k_2}^{(1)} \supset \ldots \supset J_{k_p}^{(1)} \supset J_{k_1}^{(1)}$). That is, the uniqueness follows from (5.2).

The free 0 positions of the unique binary matrices can be characterized in a similar way: Consider a unique matrix $A \in \mathcal{A}^Q$. Then let us switch the free 1's and 0's in A. The new matrix is also unique (it has switching chain if and only if A has), and for its 1's, that is, for the free 0's of A, Theorems 5.1 and 5.2 can be applied. In this way we have analogous Theorems 5.3 and 5.4:

Theorem 5.3. The binary matrix $A \in \mathcal{A}^Q(R,S)$ is unique if and only if for any I subset of rows there is a row $i \in I$ such that

$$A_i^{(1)} \cap A_{i'}^{(0)} = \emptyset$$

for each $i' \in I$.

Theorem 5.4. The binary matrix $A \in \mathcal{A}^Q(R,S)$ is unique if and only if there are subsets $I_1^{(0)} \subset I_2^{(0)} \subset \ldots \subset I_{p_0}^{(0)}$ of the row-indices $\{1,2,\ldots,m\}$ and subsets $J_1^{(0)} \supset J_2^{(0)} \supset \ldots \supset J_{p_0}^{(0)}$ of the column-indices $\{1,2,\ldots,n\}$ $(p_0 \geq 1)$ such that

$$A^{(0)} = \bigcup_{t=1}^{p_0} (I_t^{(0)} \times J_t^{(0)}) \setminus Q^{(0)}.$$

For example, in the case of Figure 5.1(a)

$$(\{2,5,6\} \times \{2,3,4,5,6\}) \cup (\{2,4,5,6\} \times \{2,4,5,6\}) \cup (\{2,3,4,5,6\} \times \{4,5,6\})$$

is the set constructed by the Procedure 5.1 to cover the free 1's of the switched matrix (i.e. to cover the free 0's of the given matrix).

Remark. In the class $\mathcal{A}$ Theorems 5.2 and 5.4 give

$$A^{(1)} = \bigcup_{t=1}^{p_1} (I_t^{(1)} \times J_t^{(1)})$$

and

$$A^{(0)} = \bigcup_{t=1}^{p_0} (I_t^{(0)} \times J_t^{(0)}),$$

which is a special case of the structure results of [12].

Theorem 5.2 (and also 5.4) gives the possibility to "order" the rows and columns of the matrix such that the 1's are to the left of the free 0's in each row, and at the same time, the 1's are above the free 0's in each column of the ordered matrix. To get this matrix, we permute the rows and columns so that $I_t^{(1)}$ consists of the uppermost rows and $J_t^{(1)}$ consists of the leftmost columns for each $t \in \{1, 2, \ldots, p_1\}$. It is also true that if a matrix has this property then it has no switching chain. Thus, we have

Theorem 5.5. The binary matrix A is unique if and only if after eventual permutations the 1's are to the left of the free 0's in each row, and at the same time, the 1's are above the free 0's in each column.

For example, Fig. 5.1(b) shows the matrix ordered from the matrix Fig. 5.1(a).

Remark. In the class $\mathcal{A}$ (no prescribed elements) a unique matrix is easily transformed in such a form by ordering the rows and columns such that the projections are non-increasing vectors (see the normalized class in [14]).

Acknowledgements

The author would like to express his appreciation to Professor Herman (Philadelphia) and Professor Kölzow (Erlangen) for their comments and help.

References

[1] R.P. Anstee: Properties of a class of binary matrices covering a given matrix, *Can. J. Math.* **34**, 1982, 438-453.

[2] R.P. Anstee: Triangular (0,1)-matrices with prescribed row and column sums, *Discr. Math.* **40**, 1982, 1-10.

[3] R.P. Anstee: The network flow approach for matrices with given row and column sums, *Discr. Math.* **44**, 1983, 125-138.

[4] R.A. Brualdi: Matrices of zeros and ones with fixed row and column sum vectors, *Linear Algebra and Its Appl.* **33**, 1990, 159-231.

[5] S.K. Chang: The reconstruction of binary patterns from their projections, *Comm. ACM* **14**, 1971, 21-25.

[6] W.Y.C. Chen: Integral matrices with given row and column sums, *J. Comb. Theory*, Ser. A, **61**, 1992, 153-172.

[7] L.R. Ford, D.R. Fulkerson: Flows in Networks, *Princeton University Press*, Princeton, NJ. 1962.

[8] D.R. Fulkerson: Zero-one matrices with zero trace, *Pacific. J. Math.*, **10**, 1960, 831-836.

[9] D. Gale: A theorem on flows in networks, *Pacific J. Math.* **7**, 1957, 1073-1082.

[10] G.T.Herman, A. Kuba: On binary matrices with prescribed elements, *Technical Report MIPG-205, Department of Radiology, University of Pennsylvania, Philadelphia, PA*, 1993.

[11] H.G. Kellerer: Fuktionen auf Produkträumen mit vorgegebenen Marginal-Funktionen, *Math. Annalen*, **144**, 1964, 323-344.

[12] A. Kuba: Determination of the structure of the class $A(R,S)$ of binary matrices, *Acta Cybernetica* **9**, 1989, 121-132.

[13] H.J. Ryser: Combinatorial properties of matrices of zeros and ones, *Can. J. Math.* **9**, 1957, 371-377.

[14] H.J. Ryser: Combinatorial Mathematics, *The Math. Assoc. of Amer.*, 1963.

[15] Y.R. Wang: Characterization of binary patterns and their projections, *IEEE Trans. on Comp.* **C-24**, 1975, 1032-1035.

Received October, 1994

Acta Cybernetica, Vol. 12, No. 1, Szeged, 1995

Demonstration of a Problem-Solving Method*

Judit Nyéky-Gaizler † Márta Konczné-Nagy †

Ákos Fóthi † Éva Harangozó †

Abstract

A program for backtrack seeking is proved here by using deduction rules. The problem of whether a chessboard can be moved over by the knight stepping on every square once and only once, is studied, and is traced back to the theorem of backtrack seeking in two ways. A comparison is made between the programs obtained.

1 Introduction

The last forty years have seen a rapid development in programming. Initially the hardware developed more rapidly than the software technology. For a long time the effectiveness of the programs had been the most important factor in programming, *but the importance of the reliability* of the programs became underlined by the improving quality of hardware tools and by the demand for producing increasingly larger systems.

The first works of Floyd, Hoare, Dijkstra and others [2,1,11] on proving program correctness were published in the 70s; work in this field was continued by Gries, Mili, Jackson, Wirth, etc [7,8,9,10,13]. Parallel with the theoretical research the results were translated into practice.

To prove the correctness of existing programs is only one possibility. A better approach is: write correct programs. Several programming theorems are proven [3,8] etc. for solving classes of important problems. In the present paper a program for the general problem of backtrack seeking is proved by using deduction rules. The problem of whether a chessboard can be moved over by the knight stepping on every square once and only once, is studied, and is traced back to the theorem of backtrack seeking in two ways. A comparison is also made between the programs obtained.

The most important definitions and theorems that are necessary to understand the present paper are available in the literature [3,4,5,6].

*Supported by the Hungarian National Science Research Grant (OTKA), Grant Nr. 2045

†Dept. of General Computer Science, Eötvös Loránd University, Budapest, Hungary, 1088 Budapest, Múzeum krt. 6-8., E-mail: nyeky@ludens.elte.hu

2 Theorem of Backtrack Seeking

2.1 The problem

Let $\mathbf{U}_1, \ldots, \mathbf{U}_n$ be finite sets. Denote by α_i the number of elements of $\mathbf{U}_i$:

$$| \mathbf{U}_i | = \alpha_i \qquad \forall i \in [1, n].$$

Let $\mathbf{U}$ denote

$$\mathbf{U} = \overset{n}{\underset{i=1}{\times}} \mathbf{U}_i.$$

Let $\rho : \mathbf{U} \to \mathbf{L}$ be a logical function having the following properties: there exists a $\rho_i (i \in [0, n])$ sequence of logical functions, for which:

1. $\rho_0 = TRUE$
2. $\rho_{i+1}(u) \Rightarrow \rho_i(u) \qquad \forall i \in [1, n-1]$
3. $\forall j \in [1, i] : u_j = v_j \Rightarrow \rho_i(u) = \rho_i(v)$
   this means that ρ_i depends only on the first i component of u.
4. $\rho_n = \rho$

The problem is to decide whether there exists a $u \in \mathbf{U}$, for which $\rho(u)$ is true. If yes, let $u \in \mathbf{U}$ with the property ρ being given.

2.2 The specification of the problem

Let

$$\boldsymbol{N} = \overset{n}{\underset{i=1}{\times}} \mathbf{V}_i, \qquad \mathbf{V}_i = [0, \alpha_i - 1] \subset \mathbf{N}_0 \qquad \forall i \in [1, n].$$

In this case: $| \boldsymbol{N} | = | \mathbf{U} |$.

$\mathbf{U}_i$ can be ordered from 0 to $(\alpha_i - 1)$, $\forall i \in [1, n]$.

Denote by $u_{i_j} \in \mathbf{U}_i$ the jth element of $\mathbf{U}_i$.

Let ϕ denote a function, which is a bijection between $\boldsymbol{N}$ and $\mathbf{U}$: $\phi : \boldsymbol{N} \to \mathbf{U}$, and if $\nu \in \boldsymbol{N}$, then: $\phi(\nu) = (u_{1_{\nu_1}}, \ldots, u_{n_{\nu_n}})$.

We can consider the elements of $\boldsymbol{N}$ as numbers encoded in a mixed radix number system [12]. Therefore we have defined an ordering on $\boldsymbol{N}$, and can speak about the "follower" of an element.

Let us denote by $f(\nu)$ the value of $\nu \in \boldsymbol{N}$ in the decimal system, that is

$$f(\nu) = \sum_{i=1}^{n} (\nu_i * \prod_{j=i+1}^{n} \alpha_j).$$

If $\nu', \nu'' \in \boldsymbol{N}$, then we shall consider $\nu' < \nu''$ iff $f(\nu') < f(\nu'')$.

Denote by ϵ_0 the zero value of $\boldsymbol{N}$: $\epsilon_0 = (0, 0, \ldots, 0)$ and by ϵ_n the unit value of $\boldsymbol{N}$: $\epsilon_n = (0, 0, \ldots, 1)$. Moreover $\forall i \in [1, n-1], let \epsilon_i = (0, 0, \ldots, 1, \ldots, 0) \in \boldsymbol{N}$, such that:

$$f(\epsilon_i) = \prod_{j=i+1}^{n} \alpha_j.$$

With the help of these we can write the specification of the problem. Let the state space be $\boldsymbol{N} \times \mathbf{L}$, and its variables ν and l.

A: $\boldsymbol{N} \times \mathbf{L}$
$\nu \quad l$

The precondition of the problem is
$Q : TRUE$;
the postcondition of the problem is
$R : l = (\exists\, \nu' \in \boldsymbol{N} : \rho(\phi(\nu'))) \wedge l \Rightarrow (\rho(\phi(\nu)))$.

2.3 Solution of the problem

Disregarding the special features of ρ, the original problem can be solved by the theorem of the third variation of linear seeking [3] in the interval $[1, \mid \boldsymbol{N} \mid]$, for the property ρ, and with the stopping-condition $f(\nu) \geq \mid \boldsymbol{N} \mid -1$.

We can increment the values in the mixed radix number system by the unit value ϵ_n .

Let ϵ_0 be the initial value of ν, to avoid the problems coming from the use of negative values in the mixed radix number system; ϵ_0 being the first $\nu \in \boldsymbol{N}$ to be interpreted.

The result is given by

program {1}:
  $l, v, \nu := \rho(\phi(\nu)), \text{false}, \epsilon_0$
  while $\neg l \wedge \neg v$ **loop**
    $\nu := \nu \oplus \epsilon_n$
    $l := \rho(\phi(\nu))$
    $v := f(\nu) \geq \mid \boldsymbol{N} \mid -1$
  endloop
end

We can significantly increase the effectiveness of the algorithm by using the special features of ρ, namely, that if $\rho_i(\phi(\nu)) = \textbf{true}$ and $\rho_{i+1}(\phi(\nu)) = \textbf{false}$, then for every $\nu' \in \boldsymbol{N}$, which satisfies $\forall j \in [1, i+1] : \nu_j = \nu'_j$, then $\rho_{i+1}(\phi(\nu'))$ will also be **false** because of the third property of ρ.

So instead of $\nu \oplus \epsilon_n$ the next possible $\nu \in \boldsymbol{N}$ will be the value $\nu \oplus \epsilon_{i+1}$.

The counting algorithm will be more simple if we amplify ν with an overflow bit - denoted by c. If the overflow bit changes to 1, it means that we no longer have the possibility to change ν.

From these it follows that it is worth supplementing the assignment $l := \rho(\phi(\nu))$ by seeking the smallest index for which $\rho_i(\phi(\nu)) = \textbf{false}$.

Using the rules of deduction [3] the following program can be achieved.

As the invariant of the loop let us use: $P : (\forall \nu' : (0 \leq f(\nu') < f(\nu) + c * \mid \boldsymbol{N} \mid :$
$\neg\rho(\phi(\nu')) \wedge l = \rho(\phi(\nu)) \wedge \neg l \Rightarrow (\rho_{m-1}(\phi(\nu)) \wedge \neg\rho_m(\phi(\nu))) \wedge (\forall i \in [m+1, n] : \nu_i = 0))$

If $\pi = \neg l \wedge (c = 0)$ is considered as the condition of the loop, then $P \wedge \neg\pi \Rightarrow R$ is really completed.

Let the terminator function be $t = \mid \boldsymbol{N} \mid -f(\nu) - c * \mid \boldsymbol{N} \mid$. Evidently $t > 0$, while $P \wedge \pi$ is true. The function t will be decreased by increasing ν.

To perform the condition $\mathbf{Q} \Rightarrow \mathbf{P}$ we need adequate initial values for the variables before starting the loop.
The **Backtrack** program will be:

program {2}:
  $\nu, c, m := \epsilon_0, 0, 1$
  SEEK(ν, m, l) $\quad \leftarrow \mathbf{Q}'$
  while $\neg l \wedge (c = 0)$ **loop**
    SUM(ν, m, c)
    SEEK(ν, m, l)
  endloop
end

To verify $\mathbf{Q}' \Rightarrow \mathbf{P}$ let us define the **SEEK** program. Let the specification of the **SEEK**(ν, m, l) program be:

$$
\begin{array}{ll}
A_{\mathbf{SEEK}}: & \underset{\nu}{\mathbb{N}} \times \underset{m}{\mathbf{N}_0} \times \underset{l}{\mathbf{L}} \\
B_{\mathbf{SEEK}}: & \underset{\nu'}{\mathbb{N}} \times \underset{m'}{\mathbf{N}_0} \\
Q_{\mathbf{SEEK}}: & \nu = \nu' \wedge m = m' \wedge \rho_{m'-1}(\phi(\nu)) \\
R_{\mathbf{SEEK}}: & \nu = \nu' \wedge l = (\forall i \in [m', n] : \rho_i(\phi(\nu))) \wedge \\
 & \neg l \Rightarrow (\neg \rho_m(\phi(\nu)) \wedge \forall i \in [m', m) : \rho_i(\phi(\nu))) \wedge \rho_{m'-1}(\phi(\nu)))
\end{array}
$$

This problem can also be solved by the theorem of the third variation of linear seeking [3] bearing in mind that the following two statements are equivalent: (1) every element of a set has a certain property, (2) there is no a single element in the set without this property.

Thus, the **SEEK** program will be

program {3}:
  $l, m := \mathbf{true}, m - 1$
  while $l \wedge (m \neq n)$ **loop**
    $l := \rho_{m+1}(\phi(\nu))$
    $m := m + 1$
  endloop
end

Therefore in the main program there will be $\mathbf{Q}' = \mathbf{R}_{\mathbf{SEEK}} \wedge (\nu = \epsilon_0) \wedge (c = 0)$ and $\mathbf{Q}' \Rightarrow \mathbf{P}$ simply follows .

To prove the implication $\mathbf{P} \wedge \pi \Rightarrow wp(S_0, \mathbf{P})$ we need determine the **SUM** program as well. Let the specification of the **SUM**(ν, m, c) program be:

$$
\begin{array}{ll}
A_{\mathbf{SUM}}: & \underset{\nu}{\mathbb{N}} \times \underset{m}{\mathbf{N}_0} \times \underset{c}{E} \\
B_{\mathbf{SUM}}: & \underset{\nu'}{\mathbb{N}} \times \underset{m'}{\mathbf{N}_0} \\
Q_{\mathbf{SUM}}: & \nu = \nu' \wedge m = m' \\
R_{\mathbf{SUM}}: & (f(\nu) + c * \mid \mathbb{N} \mid = f(\nu') + \textstyle\prod_{i=m'+1}^{n} \alpha_i) \wedge m \in [0, m'] \wedge \\
 & (\forall i \in [m+1, m'] : \nu_i = 0) \wedge (c = 0 \Rightarrow \nu_m \neq 0).
\end{array}
$$

As the invariant of the loop let us use

$$P_{\mathbf{SUM}}\colon (f(\nu) + c * \prod_{i=m+1}^{n} \alpha_i = f(\nu') + \prod_{i=m'+1}^{n} \alpha_i) \wedge m \in [0, m'] \wedge (\forall i \in [m+1, m'] : \nu_i = 0) \wedge (c = 0 \Rightarrow \nu_m \neq 0).$$

Let us consider $\pi_{\mathbf{SUM}} = (m \neq 0) \wedge (c \neq 0)$ as the condition of the loop.

In this case $\mathbf{P_{SUM}} \wedge \neg\pi_{\mathbf{SUM}} \Rightarrow \mathbf{R_{SUM}}$.

Let the terminator function be: $t_{\mathbf{SUM}} = m + c$.

Evidently $t_{\mathbf{SUM}} > 0$ while $P_{\mathbf{SUM}} \wedge \pi_{\mathbf{SUM}}$ is true.

The function $t_{\mathbf{SUM}}$ will be decreased by increasing either m or c. Thus the program will be

program {4}:

$c := 1$ $\quad \leftarrow \mathbf{Q'_{SUM}}$

while $(m \neq 0) \wedge (c \neq 0)$ **loop**

if $\nu_m = \alpha_m - 1$ **then** $\nu_m := 0$

$m := m - 1$

else $c := 0$

$\nu_m := \nu_m + 1$

endif

endloop

end

In this case $Q'_{\mathbf{SUM}} : (\nu = \nu') \wedge (m = m') \wedge (c = 1)$. Therefore $\mathbf{Q'_{SUM}} \Rightarrow \mathbf{P_{SUM}}$.

To verify $\mathbf{P_{SUM}} \wedge \pi \Rightarrow \mathrm{wp}(\mathbf{S0_{SUM}}, \mathbf{P_{SUM}})$ we have to prove:

1. $\mathbf{P_{SUM}} \wedge \pi \wedge (\nu_m = \alpha_m - 1) \Rightarrow \mathrm{wp}((\nu_m := 0; m := m - 1), \mathbf{P_{SUM}})$
2. $\mathbf{P_{SUM}} \wedge \pi \wedge (\nu_m \neq \alpha_m - 1) \Rightarrow \mathrm{wp}((c := 0; \nu_m := \nu_m + 1), \mathbf{P_{SUM}})$

These are consequences of the definition of the function $f(\nu)$ using the weakest precondition of the assignment statement.

Having proved the SUM program for the verification of the main program we need $\mathbf{P} \wedge \pi \Rightarrow \mathrm{wp}((\mathbf{SUM}; \mathbf{SEEK}), \mathbf{P})$ and this follows from the above.

3 Solution of a demonstration problem

3.1 The problem

The 8×8 $(n \times n)$ chessboard is given. We have to decide whether it is possible for the knight to move over the whole chessboard stepping on each square once and only once. If it is possible, we should be able to give a "tour".

Two possible solutions of this problem will be given and compared below.

3.1.1 Specification of the first solution

Since we have to step on 64 (n^2) squares, we can use a vector of 64 (n^2) length for the storage of the knight's moves. The j^{th} component of the vector denotes the

position of the j^{th} step. Let us number each square of the table line by line from 0 to 63 $(0-(n^2-1))$:

$$A: \underset{\nu}{\boldsymbol{N}} \times \underset{l}{L} \qquad \boldsymbol{N} = \overset{64}{\underset{i=1}{\times}}[0,63]$$

The precondition of the problem is
$Q : TRUE$;
the postcondition of the problem is
$R : l = (\exists\, \nu' \in \boldsymbol{N} : \rho(\nu')) \wedge l \Rightarrow (\rho(\nu))$.

Let us denote by $LIN_i = \nu_i/8$;and by $COL_i = \nu_i - 8 * LIN_i$, (in general: $LIN_i = \nu_i/n$; and by $COL_i = \nu_i - n * LIN_i$), where the fraction bar denotes the division between integers.

Let $\rho : \boldsymbol{N} \rightarrow \mathbf{L}$ be a logical function to be defined as follows: Let $\rho_i (i \in [0, n^2])$ be a sequence of logical functions satisfying

1. $\rho_0 = \rho_1 = TRUE$
2. $\rho_i(\nu) = \rho_{i-1}(\nu) \wedge \mu_i(\nu) \wedge \gamma_i(\nu) \qquad \forall i \in [2,64]$
   $\mu_i(\nu) = \nu_i$ knight-move-distance from ν_{i-1} =
   $= (\mid LIN_i - LIN_{i-1} \mid = 2 \wedge \mid COL_i - COL_{i-1} \mid = 1) \vee$
   $(\mid LIN_i - LIN_{i-1} \mid = 1 \wedge \mid COL_i - COL_{i-1} \mid = 2)$.
   $\gamma_i(\nu) = \nu_i$ different from the squares over
   $= (\forall i : 1 \leq j < i : \nu_j \neq \nu_i)$

   In this case $\rho_i(\nu) \Rightarrow \rho_{i-1}(\nu)$, and obviously:

3. $\forall j \in [1,i] : \nu_j = \nu'_j \Rightarrow \rho_i(\nu) = \rho_i(\nu')$; that is, ρ_i depends on the first i component of $\boldsymbol{N}$ only.
4. $\rho_{64} = \rho$

3.1.2 The first solution of the problem

It can be seen that this specification is equivalent to the specification of the general **Backtrack** seeking algorithm, therefore the program for solving it can be used with the following way of correspondence:

$n = 64$
$\forall i \in [1,64] : \mathbf{U}_i = \{0,1,\ldots,63\}, \quad \alpha_i = 64$
(We shall use a 64 based number system instead of the general mixed radix number system.)
ϕ is the identical mapping.
The program is as follows

```
program {5}:
  ν, c, m := ε0, 0, 1
  SEEK(ν, m, l)
  while ¬l ∧ (c = 0) loop
    SUM(ν, m, c)
    SEEK(ν, m, l)
  endloop
end
```

The **SEEK** program is given by

program {6}:
  $l, m :=$ **true**, $m - 1$
  while $l \wedge (m \neq 63)$ **loop**
    $l := \rho_{m+1}(\nu)$
    $m := m + 1$
  endloop
end

The **SUM** program will be

program{7}:
  $c := 1$
  while $(m \neq 0) \wedge (c \neq 0)$ **loop**
    if $\nu_m = 63$ **then** $\nu_m := 0$
                $m := m - 1$
            else $c := 0$
                $\nu_m := \nu_m + 1$
    endif
  endloop
end

We now need the program for the assignment statement $l := \rho_{m+1}(\nu)$ only. As we have defined $\rho_{m+1}(\nu) = \rho_m(\nu) \wedge \mu_{m+1}(\nu) \wedge \gamma_{m+1}(\nu)$, consequently the precondition of this program is

$Q : ((l = \rho_m(\nu)) \wedge (\nu = \nu') \wedge l')$;

and the postcondition is

$R : (l = \rho_{m+1}(\nu) \wedge (\nu = \nu'))$.

In the state space A this is equivalent to

$R : (l_1 = \mu_{m+1}(\nu) \wedge l_2 = \gamma_{m+1}(\nu) \wedge l = (l' \wedge l_1 \wedge l_2) \wedge (\nu = \nu'))$.

The program realizing this condition is the sequence of states below

program {8}:
  $l_1 := \mu_{m+1}(\nu)$
  $l_2 := \gamma_{m+1}(\nu)$
  $l := l_1 \wedge l_2$
end

The solution of the assignment $l := \mu_{m+1}(\nu)$ will be:

$l_1 = (\mid LIN_{m+1} - LIN_m \mid = 2 \wedge \mid COL_{m+1} - COL_m \mid = 1) \vee$
  $(\mid LIN_{m+1} - LIN_m \mid = 1 \wedge \mid COL_{m+1} - COL_m \mid = 2)$.

The assignment $l_2 := \gamma_{m+1}(\nu)$ can be solved by the theorem of the third variation of linear seeking, with the considerations written under the **SEEK** program. The program $l_2 := \gamma_{m+1}(\nu)$ will be

program {9}:
  $l_2, i := \text{true}, 0$
  while $l_2 \wedge (i \neq m)$ **loop**
    $l_2 := \nu_{m+1} \neq \nu_{i+1}$
    $i := i + 1$
  endloop
end

This completes the first solution.

3.1.3 Specification of the second solution

The main idea of the second solution is to take advantage of the fact that we cannot step anywhere from a certain square of the chessboard. We can choose only from the eight possible moves of the knight. All the moves are represented by a vector showing the relative movement of the knight by two components, the first for the horizontal (lines) direction, the second for the vertical (columns) direction.

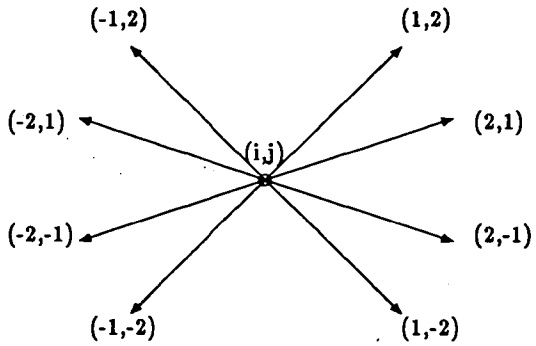

Figure 1: The knight moves

Let us consider these steps as the components of a constant vector called "knight-move-vector":

$$h = ((1,2),(2,1),(2,-1),(1,-2),(-1,-2),(-2,-1),(-2,1),(-1,2))$$

We use the **Backtrack** algorithm again with the following correspondence:

$\mathbf{n} = 64$

$\mathbf{U}_i = \{(i,j) \mid 0 \leq i,j \leq 7\}$, give an arrangement with the enumeration of the elements:

$\mathbf{U}_i = \{(0,0),(0,1),(0,2),\ldots,(7,0),(7,1),\ldots,(7,7),\}$

If we represent the chess-board by a matrix, the elements of $\mathbf{U}_i$ will be the values of the possible start positions.

$\alpha_1 = 64$

Let H denote the eight-element set obtained with the help of the knight-move-vector h, we define the arrangement on H with the enumeration in h.

$$\forall i \in [2,64]\ \mathbf{U}_i = H \qquad |\ \alpha_i\ | = 8 \qquad \mathbf{U} = \overset{64}{\underset{i=1}{\times}} \mathbf{U}_i$$

Using the sets $\mathbf{U}_i$ the actual knight move sequence on the chess-board can be given by the function:

pos: $\mathbf{U} \rightarrow \mathbf{V}$ $\quad \mathbf{V} = \overset{64}{\underset{i=1}{\times}} P$ $\quad \mathbf{P} = (LIN, COL)$ $\quad LIN, COL = \mathbf{N}_0$

$v_1 = pos(u)_1 = u_1$

$v_i = pos(u)_i = pos(u)_{i-1} \oplus u_i \qquad \forall i \in [2, 64]$

$\oplus$ denotes the addition component by component.

The correspondence between $\boldsymbol{N}$ and $\mathbf{U}$ is given by

$\boldsymbol{N} = [0, 63] \times (\overset{63}{\underset{i=1}{\times}} [0, 7])$

$\phi : \boldsymbol{N} \rightarrow U$

$u_1 = \phi(\nu)_1 = (\nu_1/8, \nu_1 - \nu_1/8 * 8)$ (the fraction bar denotes the division between integers)

$u_i = \phi(\nu)_i = h_{\nu_i} \qquad \forall i \in [2, 64]$.

The specification with the values above is

A: $\underset{\nu}{\boldsymbol{N}} \times \underset{l}{\mathbf{L}}$

$Q : TRUE$

$R : l = (\exists\ \nu' \in \boldsymbol{N} : \rho(\phi(\nu'))) \wedge l \Rightarrow (\rho(\phi(\nu)))$.

Let $\rho : \boldsymbol{N} \rightarrow \mathbf{L}$ be a logical function to be defined as follows: let $\rho_i (i \in [1, 64])$ be a sequence of logical functions satisfying

1. $\rho_1 = TRUE$
2. $\rho_i(\phi(\nu)) = \rho_{i-1}(\phi(\nu)) \wedge \mu_i(\phi(\nu)) \wedge \gamma_i(\phi(\nu)) \qquad \forall i \in [2..64]$

where $\mu_i(\phi(\nu)) =$ the i^{th} move does not move off the chess-board

$= pos(\phi(\nu))_i \in [0, 7] \times [0, 7]$

and $\gamma_i(\phi(\nu)) = pos(\phi(\nu))_i$ different from the squares over

$= (\forall j : 1 \leq j < i : pos(\phi(\nu))_j \neq pos(\phi(\nu))_i)$

In this case $\rho_i(\phi(\nu)) \Rightarrow \rho_{i-1}(\phi(\nu))$, and obviously:

3. $\forall j \in [1, i] : \nu_j = \nu'_j \Rightarrow \rho_i(\phi(\nu)) = \rho_i(\phi(\nu'))$; that is, ρ_i depends on the first i component of N only.
4. $\rho_{64} = \rho$.

3.1.4 The second solution of the problem

It can be observed, that the function pos given by a recursive formula in the program working out the assignment $l := \rho_{m+1}(\phi(\nu))$ can be substituted by a variable, and thus its evaluation will be significantly simpler. Let us amplify the state space with a component of type $\mathbf{V}$, denoted its variable by v.

The first m element of v shows then the sequence of actual positions of the knight.

The program obtained is

program{10}:

$\nu, c, m := \epsilon_0, 0, 1$

$v_1 := (0, 0)$ $\quad \leftarrow$ *startposition*

SEEK(ν, m, l)

while $\neg l \wedge (c = 0)$ **loop**

SUM(ν, m, c)

SEEK(ν, m, l)
endloop
end

Since the **SEEK** program differs from that written in the first solution in realizing the assignment $l := \rho_{m+1}(\phi(\nu))$ only, here we give just the difference.

The **SUM** program is changed in comparison with the first solution:

program{11}:
$c := 1$
while $(m \neq 0) \wedge (c \neq 0)$ **loop**
if $(m = 1 \wedge \nu_m = 63) \vee (m \neq 1 \wedge \nu_m = 7)$ **then** $\nu_m := 0$
$m := m - 1$
else $c := 0$
$\nu_m := \nu_m + 1$
endif
endloop
end

Let us give the program solving $l := \rho_{m+1}(\nu)$!

Since $\rho_{m+1}(\phi(n)) = \rho_m(\phi(\nu)) \wedge \mu_{m+1}(\phi(\nu)) \wedge \gamma_{m+1}(\phi(\nu))$; thus, the precondition of the program is

$Q : ((l' = \rho_m(\phi(\nu'))) \wedge (\nu = \nu') \wedge l')$;

the postcondition is

$R : (l = \rho_{m+1}(\phi(\nu)) \wedge (\nu = \nu'))$.

This is equivalent in the state space A with

$R : (l_1 = \mu_{m+1}(\phi(\nu)) \wedge l_2 = \gamma_{m+1}(\phi(\nu)) \wedge l = (l_1 \wedge l_2) \wedge (\nu = \nu'))$.

The program realizing this condition is the sequence of statements below.

program {12}:
$v_{m+1} := v_m \oplus h_{\nu_{m+1}}$
$l_1 := \mu_{m+1}(v_{m+1})$
$l_2 := \gamma_{m+1}(v)$
$l := l_1 \wedge l_2$
end

The solution of the assignment $l_1 := \mu_{m+1}(v_{m+1})$ will be

$$l_1 := (0 \leq (v_{m+1})_1 \leq 7) \wedge (0 \leq (v_{m+1})_2 \leq 7)$$

The assignment $l_2 := \gamma_{m+1}(v)$ can be solved by the theorem of the third variation of linear seeking, with the considerations written under the **SEEK**program.

Thus the program $l_2 := \gamma_{m+1}(v)$ is

program {13}:
$l_2, i := \mathbf{true}, 0$
while $l_2 \wedge (i \neq m)$ **loop**
$l_2 := v_{m+1} \neq v_{i+1}$
$i := i + 1$

endloop
end

This program is essentially the same as the corresponding one in the first solution. Thus the second solution is completed.

3.2 Comparison of the two solutions

If the two solutions are compared from the viewpoint of execution time, the second one is found to be essentially faster. The reason for this lies in the number of potential attempts at the first solution

$$| \boldsymbol{N} | = \prod_{i=1}^{n^2} n^2.$$

The corresponding value at the second solution is smaller by an order of magnitude:

$$| \boldsymbol{N} | = \prod_{i=1}^{n^2} n.$$

The above example indicates that one should never automatically trace the problems back to the various programming theorems, since the innovational way of thinking of the expert programmer is essential.

References

[1] Dahl, O.J., Dijkstra, E.W., Hoare, C.A.R.: Strukturált programozás (Structured programming), Műszaki Könyvkiadó, Budapest, 1978.

[2] Dijkstra, E.W.: A Discipline of Programming, Englewood Cliffs, 1976, Prentice-Hall Series in Automatic Computation.

[3] Fóthi Á.: Introduction into Programming (Bevezetés a programozáshoz), in Hungarian, manuscript, ELTE TTK, Budapest, 1983.

[4] Fóthi Á.: A Mathematical Approach to Programming, *Annales Uni. Sci. Budapest. de R. Eötvös Nom. Sectio Computatorica*, Tom. IX. (1988), 105-114.

[5] Fóthi Á., Horváth Z.: The Weakest Precondition and the Theorem of the Specification, in Proceedings of the Second Symposium on Programming Languages and Software Tools, Pirkkala, Finland, August 21-23, 1991, Eds.: Kai Koskimies and Kari-Jouko Räihä, Uni. of Tampere, Dep. of Comp. Sci. Report A-1991-5.

[6] Horváth Z.: Parallel asynchronous computation of the values of an associative function, *Acta Cybernetica*. 12 (1995) 83-94.

[7] Gries, D.: The Science of Programming, Springer Verlag, Berlin, 1981.

[8] Jackson, M.A.: Principles of Programming Design, Academic Press, New York, 1975.

[9] Mili, A.: A Relational Approach to the Design Deterministic Programs, *Acta Informatica*, 20 (1983) 315-328.

[10] Mili, A., Desharnais, J., Gagné, J.R.: Formal Models of Stepwise Refinement of Programs, *ACM Computing Surveys*, 18 (1986) 231-276.

[11] Mills, H.D.: The New Math of Computer Programming, *Comm. of the ACM* 18 (1975) 43-48.

[12] Szabo, M., S. Nicholas, Tanaka, I. Richard: Residue Arithmetic and its Applications to Computer Technology, McGraw-Hill Book Company, New Y. - San Francisco - Toronto, 1967.

[13] Wirth, N.: Systematic programming. An Introduction., Prentice-Hall Inc. 1973.

Received April, 1994

Acta Cybernetica, Vol. 12, No. 1, Szeged, 1995

Parallel asynchronous computation of the values of an associative function *

Zoltán Horváth †

Abstract

This paper shows an application of a formal approach to parallel program design. The basic model is related to temporal logics. We summarize the concepts of a relational model of parallelism in the introduction. The main part is devoted to the problem of synthesizing a solution for the problem of parallel asynchronous computation of the values of an associative function. The result is a programming theorem, which is wide applicable for different problems. The abstract program is easy to implement effectively on several architectures.

The applicability of results is investigated for parallel architectures such as for hypercubes and transputer networks.

1 Introduction

We summarize the basic concepts of a relational model of parallelism [11,13,12]. Our model is an extension of a powerful and well-developed relational model of programming, which formalizes the notion of state space, problem, sequential program, solution, weakest precondition, specification, programming theorem, etc. [8,9,16].

1.1 A relational model of parallel programs

We take the specification as the starting point for program design. We use a model of programming which supports the top-down refinement of specifications [19,8,10,9,2,11]. The proof of the correctness of the solution is developed parallel to the refinement of the specification of the problem. We formalize the main concepts of UNITY [2] in an alternative way. We use a relatively simple mathematical machinery [8,11]. The result is an expressive model, which is related to branching time temporal logics.

We give a brief survey of the main concepts and apply the methodology to solve the problem of parallel asynchronous computation of the values of an associative function in the main part.

*Supported by the Hungarian National Science Research Grant (OTKA), Grant Nr. 2045

†Dept. of General Computer Science, Eötvös Loránd University, Budapest, Hungary, 1088 Budapest, Múzeum krt. 6-8., E-mail: hz@ludens.elte.hu

1.1.1 Preliminary notions

In the following we use the terminology used also in [17,8,10,9,11]. Notations are defined often by the help of the special equality sign ::=.

The binary relation $R \subseteq A \times B$ is a *function*, if $\forall a \in A : |R(a)| = 1$. We define the domain of a relation R as $\mathcal{D}_R ::= \{a \in A | R(a) \neq \emptyset\}$. We use the notation $f : A \longmapsto B$ for functions.

The set of the logical values is denoted by $\mathcal{L}$, i.e., $\mathcal{L} ::= \{\uparrow, \downarrow\}$. A relation $f \subseteq A \times \mathcal{L}$ is called logical function, if it is a function. We use the words predicate and condition as synonyms for logical function. $\lceil f \rceil ::= \{a \in A | f(a) = \{\uparrow\}\}$ is called the truth-set of the logical function f. $[f]$ abbreviates the theorem $(\lceil f \rceil = A)$ [4]. The operations $\cup, \cap, A\backslash$ correspond to the function compositions $\wedge, \vee, \neg$. $\Rightarrow$ corresponds to $\subseteq$, $P \rightarrow Q$ is an abbreviation of $\neg P \vee Q$.

The set of the subsets of a set A is called the powerset of A and denoted by $\mathcal{P}(A)$.

Let $I \subset \mathcal{N}$. $\forall i \in I : A_i$ is a finite or numerable set. The set $A ::= \overset{\times}{_{i \in I}} A_i$ is called state space, the sets A_{i_j} are called *type value sets* . The projections $v_i : A \longmapsto A_i$ are called variables. A^* is the set of the finite sequences of the points of the state space and A^∞ the set of the infinite sequences. Let $A^{**} = A^* \cup A^\infty$.

We can imagine a statement (a sequential program) as a relation, which associates a sequence of points of the state space to some points of the state space, i.e., a statement is a subset of the direct product $A \times A^{**}$. The full formal definition of statement is given in [8].

The **effect relation** of a statement s is denoted by $p(s)$. The effect relation expresses the functionality of the statement. $p(s) \subseteq A \times A$, $\mathcal{D}_{p(s)} ::= \{a \in A \mid s(a) \subseteq A^*\}$, and
$\forall a \in \mathcal{D}_{p(s)} : p(s)(a) ::= \{b \in A \mid \exists \alpha \in s(a) : \tau(\alpha) = b\}$, where $\tau : A^* \rightarrow A$ is a function, which associates its last element to the sequence $\alpha = (\alpha_1, ..., \alpha_n)$, i.e., $\tau(\alpha) = \alpha_n$.

The logical function $wp(s, R)$ is called the **weakest precondition** of the postcondition R in respect of the statement s. We define $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$. The logical function $sp(s, Q)$ is called the strongest postcondition of Q in respect of s. $\lceil sp(s, Q) \rceil ::= p(s)(\lceil Q \rceil)$.

$A = A_1 \times ... \times A_n$, $F = (F_1, ..., F_n)$, where $F_i \subseteq A \times A_i$. Let $\lceil \pi_i \rceil ::= \mathcal{D}_{F_i}$. The relation $F_i|_{\lceil\uparrow\rceil}$ is the extension of F_i for the truth set of condition $\uparrow$ [6], i.e., $F_i|_{\lceil\uparrow\rceil}(a) ::= F_i(a)$, if $a \in \lceil \pi_i \rceil$ and $F_i|_{\lceil\uparrow\rceil}(a) ::= a_i$, otherwise. $F||_{\lceil\uparrow\rceil} ::= (F_1|_{\lceil\uparrow\rceil}, ..., F_n|_{\lceil\uparrow\rceil})$.

Let us use the notation $(\overset{\|}{_{i \in [1,n]}} (v_i :\in F_{j_i}(v_1, .., v_n), \text{ if } \pi_{j_i}))$ for the statement s_j, for which $((\mathcal{D}_{s_j} = A) \wedge (\forall a \in A : p(s_j)(a) = F||_{\lceil\uparrow\rceil}(a)))$. This kind of (simultaneous, nondeterministic) assignment is called *conditional*, if $\forall a \in A : |p(s_j)(a)| < \omega$.

Let us denote the set of n-ary relations over A by $R_n(A)$. A function $F : R_n(A) \longmapsto R_n(A)$ is monotone if $X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$. As it is well known every monotone function over a complete lattice has a minimal (least) and a maximal (greatest) fixpoint. The minimal fixpoint of the monotone function F is $\mu X : F(X) = \bigcap \{X | F(X) \subseteq X\}$, and the maximal fixpoint of F is $\eta X : F(X) = \bigcup \{X | X \subseteq F(X)\}$ [17].

1.1.2 The concepts of problem, parallel program and solution

The specification of a problem and its solution, the abstract program, is independent of architecture, scheduling and programming language. The abstract program is regarded as a relation generated by a set of deterministic (simultaneous) conditional assignments similar to the concept of abstract program in UNITY [2]. The conditions of the assignments encode the necessary synchronization restrictions explicitly. Some assignments are selected nondeterministically and executed in each step of the execution of the abstract program. Every statement is executed infinitely often, i.e., an unconditionally fair scheduling is postulated. The concept of fairness is used in the same sense as by Morris in [15] (Section 5.1), i.e., stricter than usually [2]. If more than one processor selects statements for execution, then the executions of different processors are fairly interleaved. A fixed point is said to be reached in a state, if none of the statements changes that state [2].

1.1.3 The specification of a problem

The problem is defined as a set of properties. Every property is a relation over the powerset of the sate space. Let $P, Q, R, U : A \longmapsto \mathcal{L}$ be logical functions. We define $\rhd, \mapsto, \hookrightarrow \in \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))$, and $\mathrm{FP}, \mathrm{INIT}, \mathrm{inv}, \mathrm{TERM} \subseteq \mathcal{P}(A)$.

We introduce the following infix notations:

$P \rhd Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \rhd$, $\quad P \mapsto Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \mapsto$,
$P \hookrightarrow Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \hookrightarrow$, $\quad \mathrm{FP} \Rightarrow R ::= \lceil R \rceil \in \mathrm{FP}$,
$Q \hookrightarrow \mathrm{FP} ::= \lceil Q \rceil \in \mathrm{TERM}$, $\quad Q \in \mathrm{INIT} ::= \lceil Q \rceil \in \mathrm{INIT}$,
$\mathrm{inv}\ P ::= \lceil P \rceil \in \mathrm{inv}$.

The $P \rhd Q, P \mapsto Q$, etc. formulas are called specification properties or shortly properties. The $\rhd, \mapsto, \hookrightarrow, \mathrm{inv}, \mathrm{TERM}$ relations define transition properties, the $\mathrm{FP}, \mathrm{INIT}$ relations define boundary properties. The transition relations $\rhd$ and inv express so called safety properties, while the relations $\mapsto, \hookrightarrow, \mathrm{TERM}$ express progress properties. The definition of a solution gives an interpretation for the introduced concepts.

Definition 1.1 *Let A be a state space and let B be a finite or numerable set. Two relations expressing boundary properties and four relations expressing transition properties are associated to every point of the set B. The relation $F \subseteq B \times (\overset{\times}{_{i\in[1..3]}} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A)) \, \overset{\times}{_{i\in[1..4]}} \mathcal{P}(\mathcal{P}(A)))$ is called a problem defined over the state space A. B is called the parameter space of the problem. The components of the elements of the direct products $\overset{\times}{_{i\in[1..3]}} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))$ and $\overset{\times}{_{i\in[1..4]}} \mathcal{P}(\mathcal{P}(A))$ are denoted by $\rhd_b, \mapsto_b, \hookrightarrow_b$ and by $\mathrm{INIT}_b, \mathrm{FP}_b, \mathrm{inv}_b, \mathrm{TERM}_b$ respectively.*

A program satisfies the safety property $P \rhd Q$, if and only if there is no direct transition from $P \wedge \neg Q$ to $\neg P \wedge \neg Q$ only through Q if any. A program satisfies the progress properties $P \mapsto Q$ or $P \hookrightarrow Q$ if the program starting from P inevitably reaches a state, in which Q holds. $P \mapsto Q$ defines further restriction for the direction of progress. The fixed point property $\mathrm{FP} \Rightarrow R$ defines necessary conditions for the case when the program is in one of its fixed point. The $Q \in \mathrm{INIT}$ property defines sufficient condition for the initial states of the program. $Q \hookrightarrow \mathrm{FP}$ expresses that the program starting from Q inevitably reaches one of its fixed points. P is said to be *stable* if and only if $P \rhd \downarrow$. If P holds initially and P is stable, then P is an **invariant**, denoted by inv P.

1.1.4 The definition of a parallel program

Let S be an ordered pair of a conditional assignment and a nonempty, finite set of conditional assignments, such that $S = (s_0, \{s_j \mid j \in J \wedge \mathcal{D}_{p(s_j)} = A \wedge \forall a \in A : (|s_j(a)| < \omega)\})$, $J = \{1..m\}$, $m \geq 1$.

The program $UPG(S)$ is a binary relation which associates equivalence classes of graphs generated by the effect relation of s_0 and by disjoint union of the effect relations of conditional assignments $\{s_1, ..., s_m\}$ to the points of the state space. The formal definition of a parallel program is given in [13]. The program $UPG(S)$ generated by the ordered pair $S = (s_0, \{s_1, ...s_m\})$ is denoted shortly by S. The conditional assignment s_0 is called the initialization in S and $s_j : j \in [1..m]$ is said to be an element of the program S.

1.1.5 The formal definition of a solution

The program S solves the problem F, if S satisfies all (subset of) the properties given in F. The justification of the following definitions and the proofs of the theorems is given in [11,13].

Definition 1.2 *Let S be an abstract program, $S = (s_0, \{s_1, ...s_m\})$. Let us denote the set of the indices of the deterministic assignments of abstract program S by J_d and the the set of the indices of the nondeterministic assignments by J_{nd}.*
$fixpoint_S ::= (\bigwedge_{j \in J_d, i \in [1..n]} (\pi_{j_i} \rightarrow a_i = F_{j_i}(a)) \wedge (\bigwedge_{j \in J_{nd}, i \in [1..n]} (\neg \pi_{j_i})))$.

Definition 1.3 *Let S be an abstract program. S satisfies (FP $\Rightarrow$ R) if $fixpoint_S \Rightarrow R$.*

Definition 1.4 *Let S be an abstract program, $S = (s_0, \{s_1, ...s_m\})$.*
$wp(S, R) ::= \forall s \in S : wp(s, R)$.
$wpa(S, R) ::= \exists s \in S : wp(s, R)$.

($wpa(S, R)$ is called the "angelic" weakest precondition [15]).

Definition 1.5 *The program S satisfies the property $P \rhd Q$ if and only if $(P \wedge \neg Q \Rightarrow wp(S, P \vee Q))$.*

Definition 1.6 *The program S satisfies the pair of properties $Q \in INIT$ and* inv P *if and only if $sp(s_0, Q) \Rightarrow P$ and P is stable.*

Definition 1.7
$G(P, Y, X) ::= P \vee (wpa(S, Y) \wedge wp(S, X \vee Y))$,
$F(P, Y) ::= \eta X : G(P, Y, X)$, *and*
$\rightsquigarrow P ::= \mu Y : F(P, Y)$.

Remark: Since G is monotone in P, Y, X, $\forall P, Y : \eta X : G(P, Y, X)$ exists, moreover $F(P, Y)$ is monotone in P, Y and $\rightsquigarrow P$ is monotone in P.

Definition 1.8 *(ensures) The program S satisfies the specification $(Q \mapsto P)$ if and only if $(Q \Rightarrow (P \vee (wpa(S, P) \wedge wp(S, Q \vee P))))$, i.e., $(Q \Rightarrow G(P, P, Q))$.*

Definition 1.9 *(leads-to, inevitable) The program S satisfies the specification $(Q \hookrightarrow P)$ if and only if $(Q \Rightarrow (\rightsquigarrow P))$.*

Theorem 1.1 *If $(\rightsquigarrow P)$ holds for $a \in A$, the scheduling is unconditionally fair and the program S is in the state a, then S inevitable reaches a state, for which P holds.*

We can prove the following theorems corresponding to the properties used in the definition of leads-to in UNITY [2]. The proof of progress properties is supported by the introduction of so called variant functions [6,2].

Theorem 1.2 *For an arbitrary program S,*
- *if $P \mapsto Q$ then $P \hookrightarrow Q$, and*
- *if $P \hookrightarrow Q$ and $Q \hookrightarrow R$, then $P \hookrightarrow R$.*
- *Let I be an arbitrary finite set. If $\forall i \in I : (P_i \hookrightarrow Q)$ then $(\exists i : P_i) \hookrightarrow Q$.*
- *Let W be a well-founded set in respect of the relation $<$.*
  If $\forall m \in W :: (P \wedge v = m) \hookrightarrow ((P \wedge v < m) \vee Q)$, then $P \hookrightarrow Q$.

Consequence 1.1 *If the program S satisfies the property: $(\neg\ fixpoint_S \wedge v = v') \mapsto ((\neg\ fixpoint_S \wedge v \leq v' - 1) \vee fixpoint_S)$, then S satisfies the property $(\uparrow\hookrightarrow fixpoint_S)$, i.e., $(\uparrow\hookrightarrow \mathrm{FP})$.*

A new specification is called a refinement of a previous one, if any solution for the new specification is a solution for the problem specified originally.

2 Computation of the values of an associative function

Let H be a set. Let $\circ : H \times H \longmapsto H$ denote an arbitrary associative binary operator over H.

$f : H^* \longmapsto H$ is a function describing the single or multiple application of the operator $\circ$. Since $\circ$ is associative, for any arbitrary sequence $x \in H^*$ of length at least three
$f(\ll x_1, ..., x_{|x|} \gg) = f(\ll f(\ll x_1, .., x_{|x|-1} \gg), x_{|x|} \gg) = f(\ll x_1, f(\ll x_2, .., x_{|x|} \gg) \gg)$. We write $f(\ll h_1, h_2 \gg)$ instead of the infix notation $(h_1 \circ h_2)$ in the following. We extend f for sequences of length one: $f(\ll h \gg) = h$.

Let a finite sequence $a \in H^*$ of the elements of H be given. The indices are associated to the elements of the sequence a in the reverse order, i.e., the last element is denoted by a_1. If the length of the sequence is n, then the first element is denoted by a_n. $a = \ll a_n, ..., a_1 \gg$, $(n \geq 1)$. Let us compute the value of the function $\mathcal{G} : [1..n] \longmapsto H$ for all $i \in [1..n]$, where $n \geq 1$ and

$$\mathcal{G}(i) = f(\ll a_i, ..., a_1 \gg).$$

To solve the problem we use a similar train of thought to those presented in the cases of parallel synchronous computation of the sum of binary numbers and of the asynchronous computation of the shortest path [2].

2.1 The formal specification of the problem

We specify that the program inevitably reaches a fixed point and the array g contains the values of f in any fixed point.

$A = G$, where $G = vector([1..n], H)$, $n \geq 1$; $g : G$

$$\uparrow\hookrightarrow \mathrm{FP} \tag{1}$$

$$\mathrm{FP} \Rightarrow (\forall i \in [1..n] : g(i) = f(\ll a_i, ..., a_1 \gg)) \tag{2}$$

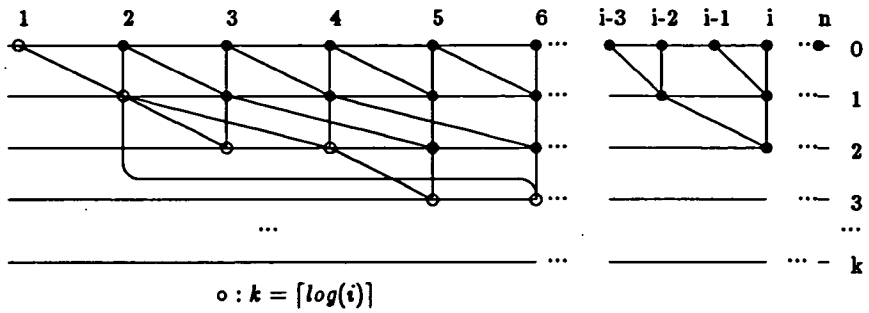

Figure 1: $gs(i,k) = h(i,k)$, if $k \leq \lceil k(i) \rceil$.

Let us observe that the computation of the values of $\mathcal{G}$ at place i is made easier with the knowledge of the value of f for subsequences $f(\ll a_u, ..., a_v \gg)$ indexed by the elements of an arbitrary $[u..v] \subseteq [i..1]$ interval. Moreover the result computed for a subsequence is useful in the computation of the value of f for any sequence which includes the subsequence.

From the above line of reasoning, we extend the state space and refine the specification of the problem. Let us introduce the auxiliary function h. Let $h(i,k)$ denote the value of f for the sequence of which the first element is a_i and its length is 2^k or the last element is a_1, if $i < 2^k$. The two-dimensional array gs is introduced to store the known values of h. This method is called the substitution of a function by a variable [7]. The connection between the variables gs, k, t and the function h is given by the invariants (4)-(6). The lines on the Figure 1 illustrate the connections among the elements of the matrix gs according to lemma 2.1 and to invariants (4)-(6).

$$
\begin{array}{lllll}
A' = & G\times & GS\times & K\times & T \\
 & g & gs & k & t
\end{array}
\qquad
\begin{array}{ll}
G & = vector([1..n], H), \\
GS & = vector([1..n, 0..(\lceil \log(n) \rceil)], H) \\
K & = vector([1..n], \mathcal{N}_0), \\
T & = vector([1..n], \mathcal{N}_0), \quad n \geq 1
\end{array}
$$

The precise definition of the partial function $h : [1..n] \times \mathcal{N}_0 \longrightarrow H$ is:

$$
h(i,k) ::= \begin{cases} f(\ll a_i, ..., a_1 \gg), & \text{if } i - 2^k + 1 \leq 1 \\ f(\ll a_i, ..., a_{(i-2^k+1)} \gg), & \text{if } i - 2^k + 1 \geq 1 \end{cases}
$$

Lemma 2.1
If $(i - 2^k \geq 1)$, *then* $f(\ll h(i,k), h(i-2^k,k) \gg) = h(i,k+1)$.

Proof: Since $i - 2^k \geq 1$, $h(i,k) = f(\ll a_i, ..., a_{(i-2^k+1)} \gg)$. If $(i-2^k)-2^k+1 \geq 1$, then $h(i-2^k,k) = f(\ll a_{(i-2^k)}, ..., a_{(i-2^k-2^k+1)} \gg)$. Since f is associative: $f(\ll h(i,k), h(i-2^k,k) \gg) = f(\ll a_i, ..., a_{(i-2^k+1)}, a_{(i-2^k)}, ..., a_{(i-2^k-2^k+1)} \gg) = h(i,k+1)$. If $(i-2^k) - 2^k + 1 < 1$, then $h(i-2^k,k) = f(\ll a_{(i-2^k)}, ..., a_1 \gg)$. Using the associativity of f: $f(\ll h(i,k), h(i-2^k,k) \gg) = f(\ll a_i, ..., a_{(i-2^k+1)}, a_{(i-2^k)}, ..., a_1 \gg) = h(i,k+1)$.

Let us choose the variant function $v : A \longmapsto \mathcal{N}_0$ in the following way:

$$v::=4 * n * n - \sum_{i=1}^{n} (k(i) + \chi(k(i) = \lceil \log(i) \rceil \wedge g(i) = gs(i, k(i))))$$

The variant function depends on the number of elements of the matrix gs which elements are different from the value of function h at the corresponding place and on the number of places where the value of the array g is different from the value of function $\mathcal{G}$.

Lemma 2.2 *The specification below is a refinement of the specification (1)-(2).*

$$\uparrow\!\!\hookrightarrow \mathrm{FP} \tag{3}$$

$$\mathrm{FP} \Rightarrow \forall i \in [1..n] : (k(i) = \lceil \log(i) \rceil) \wedge (g(i) = gs(i, \lceil \log(i) \rceil)) \tag{4}$$

$$\mathrm{inv}\ (\forall i \in [1..n] : k(i) \leq \lceil \log(i) \rceil \wedge \forall k : k \leq k(i) : gs(i, k) = h(i, k)) \tag{5}$$

$$\mathrm{inv}\ (\forall i \in [1..n] : t(i) = 2^{k(i)}) \tag{6}$$

Proof:
$k(i) = \lceil \log(i) \rceil$ and $g(i) = gs(i, \lceil \log(i) \rceil)$ in fixed point according to (4). Using (5) it follows that the equation $g(i) = gs(i, \lceil \log(i) \rceil) = h(i, \lceil \log(i) \rceil)$ holds in fixed point. Since $2^{\lceil \log(i) \rceil} \geq i$, after the application of the definition of h we get $h(i, \lceil \log(i) \rceil) = f(\ll a_i, ..., a_1 \gg)$, which is the same as property (2).

Remark 2.1 The property (1) is not refined. The proof of the correctness of any program in respect of (1)=(3) is based on Consequence 1.1. This means the choose of a variant function may be regarded as an implicit refinement step in respect of property (1). Since the property (6) defines restrictions over the new components of the state space only, we need not to use it in the proof of the refinement.

2.2 A solution

Theorem 2.1 *The abstract program below is a solution for the problem specified by (3)-(6), i.e., a solution for the problem of the computation of the values of an associative function.*

$$s_0 : \ \Box_{i=[1..n]}\, gs(i, 0), t(i), k(i) := f(\ll a_i \gg), 1, 0$$

$$S : \{ \ \Box_{i=[1..n]}\, gs(i, k(i) + 1), t(i), k(i) :=$$

$$\begin{cases} f(\ll gs(i, k(i))), & gs((i - t(i)), k(i)) \gg), 2 * t(i), k(i) + 1, \\ & \quad if\ (i - 2 * t(i) + 1 \geq 1) \wedge (k(i - t(i)) \geq k(i)) \\ f(\ll gs(i, k(i))), & gs(i - t(i), k(i - t(i))) \gg), \\ & \quad 2 * t(i), k(i) + 1, \\ & \quad if\ (i - t(i) \geq 1) \wedge (i - 2 * t(i) + 1 < 1) \\ & \quad \wedge (k(i - t(i)) = \lceil \log(i - t(i)) \rceil) \end{cases}$$

$$\Box_{i=[1..n]}\, g(i) := gs(i, k(i))\, if\ (k(i) = \lceil \log(i) \rceil)$$

$\}$

where $\Box_{i=[1..n]}$ *is used for the abbreviation of n statements. Each statement is instantiated from the general form by substituting the dummy variable i by a concrete value.*

Proof:

(3): Every statement of the program decreases the variant function by 1 or does not cause state transition. If the program is not in one of its fixed points, then there exists an $i \in [1..n]$ and a corresponding conditional assignment, which assignment increases the value of $k(i)$, or there exists an i for which $k(i) = \lceil \log(i) \rceil$ and the value of $g(i)$ is different from the value of $gs(i, (\lceil \log(i) \rceil))$.

(4): using the definition of the $fixpoint_S$:

$$\forall i \in [1..n] \quad (k(i) = \lceil \log(i) \rceil) \rightarrow g(i) = gs(i, k(i)) \ \wedge \tag{7}$$

$$((i - 2 * t(i) + 1 < 1) \vee (k(i - t(i)) < k(i))) \ \wedge \tag{8}$$

$$(i - t(i) < 1) \vee \ (i - 2 * t(i) + 1 \geq 1) \vee (k(i - t(i)) \neq \lceil \log(i - (t(i))) \rceil)) \tag{9}$$

We apply mathematical induction on i to prove: $\forall i \in [1..n] : (k(i) = \lceil \log(i) \rceil)$. Base case: $i = 1$. From (5) and $sp(s_0, \uparrow)$ it follows that $(k(1) = \lceil \log(1) \rceil)$. Inductive hypothesis: $\forall j < i : (k(j) = \lceil \log(j) \rceil)$. Since $t(i) \geq 1$, $(k(i - t(i)) \neq \lceil \log(i - (t(i))) \rceil)$ contradicts the hypothesis. This means (9) can be simplified to $(i - t(i) < 1) \vee (i - 2 * t(i) + 1 \geq 1)$. if $(i - 2 * t(i) + 1 \geq 1)$, then $k(i - t(i)) < k(i)$ else (8) does not hold. Using the inductive hypothesis and $t(i) \geq 1$ we get $k(i - t(i)) = \lceil \log(i - t(i)) \rceil$, i.e., $\lceil \log(i - t(i)) \rceil < k(i)$. The last statement contradicts the initial condition: $(i - 2 * t(i) + 1 \geq 1) \Rightarrow (i - t(i) - t(i) + 1 \geq 1) \Rightarrow \lceil \log(i - t(i)) \rceil \geq k(i)$. This means $(i - 2 * t(i) + 1 < 1)$.
$(i - 2 * t(i) + 1 < 1) \Rightarrow (i - t(i) < 1)$, otherwise (9) does not hold. $(i - t(i) < 1) \Rightarrow k(i) \geq \lceil \log(i) \rceil$. Using the invariant (5) we get: $k(i) = \lceil \log(i) \rceil$. Based on (7) : $g(i) = gs(i, k(i)) = gs(i, \lceil \log(i) \rceil)$.

(6): Since $sp(s_0, \uparrow)$ implies $t(i) = 1$ and $k(i) = 0$, the $t(i) = 2^{k(i)}$ equality holds initially. All the assignments change the value of $k(i)$ and $t(i)$ simultaneously.

(5): Since $h(i, 0) = f(\ll a(i) \gg)$, $sp(s_0, \uparrow) \Rightarrow gs(i, k(i)) = h(i, k(i))$. Since $k(i)$ is initially 0, $sp(s_0, \uparrow) \Rightarrow (k(i) \leq \lceil \log(i) \rceil)$.

After calculating the weakest preconditions of the assingments it is sufficient to show that

- $(i - 2 * t(i) + 1 \geq 1) \wedge (k(i - t(i)) \geq k(i))$ and $\forall k : k \leq k(i) : gs(i, k) = h(i, k)$ implies the equality for $k(i) + 1$, i.e., $f(\ll gs(i, k(i)), gs(i - t(i), k(i)) \gg) = h(i, k(i) + 1)$ and $k(i) + 1 \leq \lceil \log(i) \rceil$,
- $(i - t(i) \geq 1) \wedge (i - 2 * t(i) + 1 < 1) \wedge (k(i - t(i)) = \lceil \log(i) \rceil$ and $\forall k : k \leq k(i) : gs(i, k) = h(i, k)$ implies the equality for $k(i) + 1$, i.e., $f(\ll gs(i, k(i)), gs(i - t(i), (\lceil \log(i - t(i)) \rceil)) \gg) = h(i, k(i) + 1)$ and $k(i) + 1 \leq \lceil \log(i) \rceil$.

$(i - 2 * t(i) + 1 \geq 1) \wedge (t(i) \geq 1) \Rightarrow (i - t(i) \geq 1) \Rightarrow k \leq \log(i - 1) < \log(i) \leq \lceil \log(i) \rceil)$.
In the first case $k(i) \leq k(i)$ implies $gs(i, k(i)) = h(i, k(i))$ and $(k(i - t(i)) \geq k(i))$ implies $gs(i - t(i), k(i)) = h(i - t(i), k(i))$. In the second case $k(i) \leq k(i)$ implies $gs(i, k(i)) = h(i, k(i))$ and $k(i - t(i)) = \lceil \log(i - t(i)) \rceil$ implies $gs(i - t(i), (\lceil \log(i) \rceil)) = h(i - t(i), (\lceil \log(i) \rceil))$. In both of the cases the application of the Lemma 2.1 leads to the statement.

(end of proof.)

Let us suppose the abstract program is implemented on a parallel computer containing $O(n)$ processors. If the left side of an assignment refers to an array component indexed by i, then the assignment is mapped to the ith (logical) processor. Easy to see, that the program reaches one of its fixed point in at most $O\lceil\log(n)\rceil$ state transforming steps. The logical processors may work asynchronously.

2.3 Transformation of the program

The program corresponds neither to the rule of fine-grain atomicity [1](2.4) nor to the shared variable schema [2]. To ensure effective asynchronous computation we have to transform the program by introducing new variables and using the method of substitution of a function by a variable for the function log [7].

Let us use the auxiliary arrays $gst(i) = gs(i - t(i), k(i))$, $kt(i) = k(i - t(i))$, $gstk(i) = gs(i - t(i), kt(i))$, if the values are necessary and known by the ith logical processor and the value of $kt(i)$ is big enough to determine the next (i.e. the $(k(i) + 1)$th) value of the ith column of the matrix gs (10). Let us introduce the auxiliary boolean variables $ktf(i), gstf(i), gstkf(i)$ to administrate the usage of the auxiliary arrays. The ith component of the auxiliary arrays is local in respect of the ith processor.

Every assignment of the transformed program will refer to at most one nonlocal variable.

2.3.1 The refinement of the specification

We extend the specification (3)-(6) with the following invariants:

$$\text{inv } \forall i \in [1..n]: \quad (kt(i) \le k(i - t(i))) \wedge ktf(i) \rightarrow (kt(i) \ge k(i) \vee kt(i) = l(i - t(i)))) \tag{10}$$

$$\text{inv } \forall i \in [1..n]: \quad (gstf(i) \rightarrow ktf(i) \wedge (i - 2 * t(i) + 1 \ge 1) \wedge gst(i) = gs(i - t(i), k(i))) \tag{11}$$

$$\text{inv } \forall i \in [1..n]: \quad (gstkf(i) \rightarrow ktf(i) \wedge (i - t(i) \ge 1) \wedge (i - 2 * t(i) + 1 < 1) \wedge gsik(i) = gs(i - t(i), kt(i)) = gs(i - t(i), k(i - t(i)))) \tag{12}$$

$$\text{inv } \forall i \in [1..n]: \quad \lceil\log(i)\rceil = l(i) \tag{13}$$

2.3.2 The transformed program

$$s_0 : \quad \Box_{i=[1..n]}\, gs(i,0), t(i), k(i), l(i), ktf(i), gstkf(i), gstf(i), kt(i) := f(\ll a_i \gg), 1, 0, \lceil \log(i) \rceil, \downarrow, \downarrow, \downarrow, 0$$

$$S : \{ \quad \Box_{i=[1..n]}\, kt(i) := k(i - t(i)), \text{if } \neg ktf(i) \wedge (i - t(i)) \geq 1$$

$$\Box_{i=[1..n]}\, ktf(i) := \uparrow, \text{if } \neg ktf(i) \wedge (i - t(i)) \geq 1 \wedge (kt(i) \geq k(i) \vee kt(i) = l(i - t(i)))$$

$$\Box_{i=[1..n]}\, gst(i), gstf(i) := gs(i - t(i), k(i)), \uparrow, \text{if } ktf(i) \wedge (i - 2 * t(i) + 1 \geq 1) \wedge (kt(i) \geq k(i)) \wedge \neg gstf(i)$$

$$\Box_{i=[1..n]}\, gstk(i), gstkf(i) := gs(i - t(i), kt(i)), \uparrow, \text{if } ktf(i) \wedge (i - t(i) \geq 1) \wedge (i - 2 * t(i) + 1 < 1) \wedge (kt(i) = l(i - t(i))) \wedge \neg gstkf(i)$$

$$\Box_{i=[1..n]}\, gs(i, k(i) + 1), t(i), k(i), ktf(i), gstf(i), gstkf(i), kt(i) :=$$

$$\begin{cases} f(\ll gs(i, k(i))), & gst(i) \gg), 2 * t(i), k(i) + 1, \downarrow, \downarrow, \downarrow, 0 \\ & \text{if } gstf(i) \\ f(\ll gs(i, k(i))), & gstk(i) \gg), 2 * t(i), k(i) + 1, \downarrow, \downarrow, \downarrow, 0 \\ & \text{if } gstkf(i) \end{cases}$$

$$\Box_{i=[1..n]}\, g(i) := gs(i, k(i)), \quad \text{if } k(i) = l(i)$$

$\}$

Proof: The invariants (10)-(13) are easy to prove by the calculation of the weakest preconditions and $sp(s_0, \uparrow)$. Using the invariants (10)-(13) we can state that the assignments changing the variables mentioned in (3),(5)-(6) are equivalent of the original assignments. This means the specification properties (3),(5)-(6) remain valid for the transformed program too. To prove the fixpont property (4) it will be sufficient to show: if the transformed program reaches one of its fixed points then the original program is in one of its fixed points too and the conditions (7)-(9) hold.□

3 Discussion

The program is easy to implement on synchronous, asynchronous and on distributed architectures, such as for hypercubes [18] or T9000 transputer networks, where implementation of $O(\lceil \log(n) \rceil)$ communication channels is supported by the concepts of logical links.

A solution is developed in [14] for pipeline architectures.

The introduced relational model provides effective tools for the stepwise development of a parallel solution as illustrated by the chosen example. The theorem 2.1

may be called a programming theorem [6]. With its help we can solve a class of classical problems. For example parallel addition, comparison of ascending sequences [2], etc. are easy to formalise by the help of associative functions.

References

[1] Andrews, G.R.: *Concurrent Programming, Principles Practice*, Benjamin/Cummings, 1991.

[2] Chandy, K.M., Misra, J.: *Parallel program design: a foundation*, Addison-Wesley, 1988, (1989).

[3] Dijkstra, E.W.: *A Discipline of Programming*, Prentice-Hall, 1976.

[4] Dijksta, E.W., Scholten, C.S.: *Predicate Calculus and Program Semantics*, Springer-Verlag, 1989.

[5] Emerson, E.A., Srinivasan, J.: Branching Time Temporal Logic, in *Linear Time, Branching Time and Partial Order in Logics an Models for Concurrency, LNCS 354*. Springer-Verlag 1989, 123-172.

[6] Fóthi Á.: *Introduction into Programming (Bevezetés a programozáshoz)*, in Hungarian, ELTE TTK, Budapest, 1983.

[7] Fóthi Á.: verbal communications

[8] Fóthi Á.: A Mathematical Approach to Programming, *Annales Uni. Sci. Budapest. de R. Eötvös Nom. Sectio Computatorica*, Tom. IX. (1988), 105-114.

[9] Fóthi Á., Horváth Z.: The Weakest Precondition and the Theorem of the Specification, in *Proceedings of the Second Symposium on Programming Languages and Software Tools*, Pirkkala, Finland, August 21-23, 1991, Eds.: Kai Koskimies and Kari-Jouko Räihä, Uni. of Tampere, Dep. of Comp. Sci. Report A-1991-5, August, 1991, 39-47.

[10] Horváth Z.: Fundamental relation operations in the mathematical models of programming, *Annales Uni. Sci. Budapest. de R. Eötvös Nom. Sectio Computatorica*, Tom. X. (1990), 277-298.

[11] Horváth Z.: The Weakest Precondition and the the Specification of Parallel Programs, in *Proceedings of the Third Symposium on Programming Languages and Software Tools*, Kääriku, Estonia, August 21-23, 1993, 24-33.

[12] Horváth Z., Kozma L.: Parallel Programming Methodology, to appear in *Proceedings of the Workshop on Parallel Processing, Technology and Applications*, Technical University Budapest, February 10-11, 1994.

[13] Horváth Z.: The Formal Specification of a Problem Solved by a Parallel Program - a Relational Model, in Proceedings of the Fourth Symposium on Programming Languages and Software Tools, Visegrád, Hungary, June 9-10, 1995, 165-189.

[14] Loyens, L.D.J.C., van de Vorst, J.G.G.: Two Small Parallel Programming Exercises, *Science of Computer Programming* Vol. 15(1990), 159-169.

[15] Morris, J., M.: Temporal Predicate Transformers and Fair Termination, *Acta Informatica*, Vol. 26, 287-313, 1990.

[16] Nyéky-Gaizler J., Koncsné-Nagy M., Fóthi Á., Harangozó É.: Demonstration of a problem solving method, *Acta Cybernetica* 12 (1995) 71-82.

[17] Park, D.: On the semantics of fair parallelism, in *LNCS 86*, 504-526, Springer 1980.
[18] Quinn, M., J.: *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, Inc., 1987.
[19] Varga L.: *Programok analízise és szintézise*, Akadémiai Kiadó, Budapest, 1981.

Received, July, 1994

Acta Cybernetica, Vol. 12, No. 1, Szeged, 1995

Towards Computer Aided Development of Parallel Compilers Running on Transputer Architecture*

János Toczki †

Abstract

In this paper we state requirements for a software environment for computer aided development of parallel compilers executable on transputers. The structure of a compiler-compiler which generates parallel compilers from attribute grammar specifications is described. Problems of distributed attribute evaluation using dynamic load balancing are discussed.

Keywords: attribute grammars, compilers, transputers, parallel processing.

1 Introduction

Several types of parallel machines have become more and more popular recently. Various parallel algorithms have been developed to get more efficient softwares for several problems. An important application field is developing *parallel compilers.*

Attribute grammars are an efficient compiler specification method. Most of *compiler-compiler systems* are based on attribute grammars. A survey of sequential attribute evaluation methods can be found in [7] and [2]. A review of compiler-compiler systems based on attribute grammars is presented in [6].

Most of the positive experiences with developing *parallel semantic evaluators* are connected with non-distributed architectures with shared memory. Reviews of using parallel attribute evaluation and experiences developing parallel compilers can be found in [1], [4], [16] and [10]. A blueprint of a parallel compiler generator system is presented in [3].

On the other hand, there is no shared-memory available in *transputer* machines; each processor uses its own memory. Processors are connected through channels. Channels are used not only for synchronization, but also for sending data between processors. According to the practical experiences, the main problem with using transputers for parallel attribute evaluation is the large amount of *inter-processor communication* [17], [1]. It causes the inefficiency of these algorithms.

*This research is involved in the research project "Large Parallel Databases" financed by the European Communities, project number 93: 6638, and partly supported by the research found OTKA and the Ministery of Education and Culture, grant number F12852 and 434/94

†Department of Computer Science, József Attila University of Szeged

However, there are some *practical applications* efficiently implemented on transputers. For example, a database management system processing large databases implemented at Sheffield University on *IDIOMS* machine [12]. The input language of the system is the standard *SQL*/1. Users of *IDIOMS* requires some extension to *SQL*: it needs a precompiler which transforms queries to standard *SQL*. It is a natural demand that the precompiler should run on the same machine instead of the host computer.

In this paper we consider the problems of efficient evaluation of attribute grammars on distributed architectures from a practical point of view. Which evaluation methods can be used, which other facilities are needed to get a compiler-compiler or an environment for developing parallel compilers?

This paper is composed as follows. We summarise preliminaries and motivations in the next section. We repeat some of the basic definitions and notations, however we suppose that the reader is familiar with the following topics: *attribute grammars* and *evaluation strategies*, *compiler-compilers*, *transputer architectures*. The summary of requirements for a parallel compiler-compiler running on transputers is found in section 3. Our suggestions to meet these requirements and the preliminary design of a compiler-compiler system is given in section 4. A short summary of future research is found in the last section.

2 Motivations and Preliminaries

2.1 Motivations

Parallel machines are classified as *synchronous* and *asynchronous* machines. In synchronous machines all processors execute the same code at the same time. In asynchronous architectures processors may execute different code, synchronisation should be controlled directly by using semaphores and/or sending messages.

On the other hand, we can distinguish between *tightly coupled* and *loosely coupled* (distributed) architectures. In tightly coupled machines, all processors have access to the same shared memory, while in distributed machines each processor has its own memory. Processors can communicate by sending messages.

Transputers, which are asynchronous distributed machines have become more and more popular recently. Processors - nodes - of a transputer are usually configured along a more or less regular topology (a line, hypercube, polygon, etc.). Each processor has four channels for communication. Software connections can be configured in a flexible way via connecting channels. The number of nodes and the physical topology should be as irrelevant as possible from the point of view of transputer software. However the speed and efficiency of an algorithm usually strongly depends on the topology.

Usually the most expensive part of transputer software is the inter–processor communication. Consequently, in general, an "effective" algorithm should avoid large amounts of communication. There are some "typical" transputer applications. One of these areas is processing large distributed databases. If we connect a data storage device to each node and we distribute queries at an early stage, we get efficient data retrieving algorithms.

IDIOMS machine developed at NTSC, University of Sheffield is a special parallel machine for processing *large distributed databases* using parallel methods [12]. The user surface of the system is standard *SQL*/1 language.

Parsing user instructions, *evaluation* and distribution of queries is a usual formal language parsing and semantic evaluation problem. There are three possible solutions to perform this task.

- We can use a sequential compiler running on the host machine.
- We can use a sequential compiler running on a single transputer node and benefit only from the parallelism between the larger components of the software architecture.
- We can use a parallel compiler running on the transputer itself.

The advantage of the first two solutions is that the theory and practice of developing sequential compilers is a well-researched topic of computer science. On the other hand, the host machine is the only connection between the transputer and the outer world. The host machine will be very busy with transfering results of queries. It is a natural demand to use the inner part of the system instead.

Another advantage of the last method is that – using parallel algorithms – we can speed up the compilation process itself. For example Gross achieved 3 to 6 typical speed-up with a hand-written compiler running on 9 independent workstations benefiting only from parallel compilation of independent functions (blocks) [8]. It does not exclude running the compiler in parallel with some other components, as well.

2.2 Parallel Compilation

The first hand-written parallel compilers were developed in the early 1970s. Most of these compilers run on vector processors and based on parallel execution of certain phases of compilation. The first significant investigation into parallel semantic evaluation made by Schell [22]. Schell's method – in essence – is the same as Jordan's one reported in [10].

The most natural way to build a parallel compiler is to run different compilation phases as separate processes and form a pipeline. The maximal possible speed up is the number of phases (usually 3 or 4). However, it is a hard work to balance the different phases. Miller and LeBlanc compared sequential and pipeline versions of a Pascal compiler having 4 phases and they got 2.5 speed up as an average [20]. This result shows the limitations of pipelining.

Another possible way to construct a parallel compiler is to split the source program into smaller independent parts and compile these parts concurrently. Lipkie was the first who suggested the combination of pipelining with source fragmentation [19]. Vandevoorde [27] and Seshardi [23] used the same approach developing compilers running on different architectures. Seshardi investigated the concurrent processing of declarations as well.

These experiences shows the importance of pipelining as well as the necessity of concurrent semantic evaluation. In this paper we concentrate to the phase of semantic evaluation. Concerning the phase of lexical and syntactic parsing, pipelining with immediate fragmentation seems to be a proper solution. We also concentrate to more general methods which can be used in an automatic compiler development tool.

2.3 Automatic Compiler Construction

Compiler-compiler systems generate executable compilers from formal specifications. Most recent compiler writing systems are based on attribute grammars. Experiences with compiler construction proved feasibility and efficiency of attribute grammars for compiler specification. A survey of attribute grammar based compiler generation can be found in [6].

The compiler generator system *PROF-LP* [21] developed in Szeged has been used for generating various practical compilers, for example [25], [5]. We refer to our experiences at appropriate places in the next section. We mention that these experiences and suggestions for development in the case of sequential compilation are summarized in [26].

2.4 Attribute Grammars

Various compiler-compiler systems have been developed to generate compilers from formal specifications. Most of these systems are based on attribute grammars [15].

In this section we recall some basic notions of attribute grammar theory. We give only informal definitions instead of formal ones. We feel that it is enough to make clear our concepts. More complete definitions can be found in the literature for example [2], [7].

Let $G = (N, T, S, P)$ be a context-free grammar, where N is the set of nonterminal symbols, T is the set of terminal symbols, $S \in N$ is the start symbol, and P is the set of context-free productions.

We associate a set of *attributes* to each nonterminal symbol. There are two types of attributes. The values of *inherited* attributes are evaluated top-down (from the start symbol to the terminals) and the values of *synthesized* attributes are evaluated in the opposite direction.

Attribute values are determined by semantic functions associated to the synthesized attribute occurrences of the left-hand side nonterminal and to the inherited attribute occurrences of the right-hand side nonterminals of each production. The arguments of semantic functions are the other attribute occurrences of the same production.

An abstract syntax tree of the grammar G is said to be *decorated* if all its attribute instances have their own values. We say that a decoration of a syntax tree s is *correct* iff the values of all attribute instances of s satisfy corresponding semantic rules.

Attribute instances in a syntax tree s depend on each other. We say that an attribute instance $A(N)$ *depends on* an attribute instance $b(M)$, if the value of $b(M)$ is needed to evaluate the semantic rule computing the value of $a(N)$, i. e. if it occurs as a parameter of the semantic function.

In this paper we consider only *non-circular* attribute grammars. An attribute grammar is non-circular iff there cannot exist any circular dependencies among the attribute instances in any syntax tree. In this case all attribute instances can be evaluated in a definite order constrained by the dependencies of the given syntax tree.

2.5 Parallel Attribute Evaluation

Usually there are some *independent* attribute instances in a syntax tree. In the case of sequential evaluation, a linear order is constructed, evaluating independent attribute instances in a more or less *ad hoc* order. In general, it is possible to evaluate independent attribute instances in parallel.

Kuiper [16], [18] defined the concept of *distributor* as an algorithm to distribute attribute instances among evaluation processes. He defines two basic types of distribution:

- A *tree based* distributor allocates all attribute instances of a subtree of the syntax tree to the same evaluation process. The syntax tree is splitted at selected nodes. Selected nodes determined by the production applied at the

node – *production based distribution* – or by the left hand side nonterminal of that production – *nonterminal based distribution*. The distribution can be either nested or non-nested. In the case of *nested distribution* subtrees containing selected nodes are splitted again, while in the case of *non-nested distribution*, the syntax tree is splitted only at the selected nodes closest to the root.

A typical application of tree-based distribution is the fragmentation of a block-structured programming language. Disjoint blocks are usually independent to each other. We can allocate attribute instances of different blocks to different processes using a nested nonterminal based distributor.

- An *attribute based distributor* allocates all instances of an attribute to the same process. The distributor can not distinguish between different instances of an attribute. It means a strict limit on potential parallelism. If we combine it with a tree based distributor, we get a *combined distributor.*

A typical application of attribute based distribution is to allocate independent tables of a compiler to different processes. For example, symbol tables and label tables are usually independent.

- Jordan introduced third kind of distributors. A *dependency based distributor* allocates all attribute instances of a connected part of the dependency graph to the same process. The allocation is not predefined. An evaluation order containing parallel execution of new processes is generated from the dependency graph. In this sense this method is more "dynamic" than Kuiper's distributors.

Dependency based distributors are capable of handling more complicated situations, when neither tree based nor attribute based distributions are inefficient.

3 Parallel Compilation on Transputers

3.1 Assumptions

A transputer is a loosely-coupled parallel machine having no shared memory. Processors communicate via channels. Channels serve not only for synchronization but for data exchange, as well. Process loading and channel connections are flexible. The only physical bound is the amount of memory and the number of hardware connections (usually 4). Peripherals are handled by a host computer which is connected to the processor network via channels.

In this paper we concentrate on the *semantic* part of compilers. We suppose that the complete syntax tree is available on the host computer or on a transputer node. In the case of source fragmentation, before syntactic parsing different parts of the syntax tree may be produced by different processes running on different nodes. This situation can be handled by appropriate process level distribution, see in 4.1. The result of semantic evaluation, the decorated syntax tree, is sent back to the host. In the case of a semantic error, an error message is sent to the host and the evaluation process is stopped. In some applications, it would be better to pass the result to another application. It is only a technical point.

We suppose a *static evaluation method* driven by the dependency graphs of productions. Among others *OAG* [11] and *ASE* [9] are feasible strategies. These classes of attribute grammars are large enough in practical cases.

We suppose availability of a block structured high level algorithmic programming language - we have chosen parallel *C* - and availability of a flexible *CDL* (configuration description language) usual on transputers.

3.2 Distribution

The most important feature of transputers from the point of view of attribute distribution is that there is no shared memory available. Although it is possible to run more than a single process on the same processor, we should allocate them to as many different processors as possible to increase "real" parallelity of the compiler. *On the other hand*, attribute values have to be sent among processors. As inter–processor communication is the most expensive task on transputers, we should decrease the amount of sending data among processes allocated to different processors.

Some attributes – as symbol tables – are extremely large, while others are very small. Some attributes have the same or similar meaning. For example most of the tables of compilers are represented with a pair of a synthesized and an inherited attribute. Usually tables are stored in dynamic data structures, i.e. lists, trees, stacks and the attribute values are only pointers to these tables. It means that the basic operations "*send* the value of an attribute to a process" or "*compute* a semantic function" may have *quite different expenses.*

Only the *author* of a compiler knows the size of attributes, the complexity of semantic functions. The author has enough information on potential selected nonterminals - in the case of tree based distribution - and on "logically" independent attributes - in the case of attribute based distribution.

We can state now the following basic *requirements*:

- The user should choose between tree based and attribute based distribution. Probably he/she will choose a combined strategy.
- The user should declare selected and non-selected nonterminals in the case of tree based distribution and declare the set of attributes evaluated by the same process in the case of attribute based distribution.

On the other hand there are efficient *algorithms* to find independent attribute instances of an attribute grammar. For example, see Kuiper's algorithm [16]. An intelligent system can help the user's decision and *check* its correctness using these algorithms.

- The system should help and check the user's decision on distribution using dependency analysis.

3.3 Process Loading

Most of transputer operating systems include some *load balancing* mechanisms. It means that the system distributes processes among processors on the bases of their current status. On the other hand the user has the possibility to describe her/his own configuration using *CDL* (Configuration Description Language).

Automatic load balancing assumes a *farmer-workers* architecture, while the user can use (almost) any other architecture. It gives a large amount of flexibility. On the other hand, it is much easier to program an automatically balanced system.

Another important question that we should answer is: should we develop a general evaluation process which contains all the semantic functions and run it on all processors or should we develop several smaller processes? Execution time of

semantic functions and the number of attribute instances evaluated by a process may be quite different. Furthermore scheduling many small processes causes too much overhead time. It is more efficient to run a *general evaluator* on all processors and implement evaluation processes as tasks rather than real physical processes. In this approach a *task* means evaluation of all attribute instances allocated to a logical process. Each task has a set of *output attribute instances* – the attribute instances which are computed – and a set of *input attribute instances* – values of which are needed for the computation.

In this case, we cannot use the automatic load balancing mechanism of the operating system: load balancing means allocating tasks to processes, and not allocating physical processes to processors. The dynamic load balancing method described in [24] is applicable for any *decomposable problem*. Although attribute evaluation is not a decomposable problem, we can associate a home processor to each attribute instance. The value of an attribute instance is sent back to its home processor after evaluation. More detailed description of process loading can be found in the next section.

4 Developing Parallel Compilers

In this section we describe the structure of a software environment for developing parallel compilers based on the requirements stated in section 2. First we discuss the features of a metalanguage for specifying a parallel compiler and describe the general structure of the generated compiler. After that, we sketch the structure of the development environment including a generator tool. Finally we consider some technical questions.

4.1 Parallel Compiler Specification

The *specification* of a parallel compiler is an *attribute grammar* completed with evaluation instructions and with the implementation of semantic functions. We start from the metalanguage of *PROF-LP* [21]. This metalanguage has the following features.

- The set of synthesized and inherited *attributes* are declared. The domain of an attribute is given by a data type of the implementation language.
- The set of *nonterminals* with the list of their attributes is declared. The generated compiler is modular, a module is formed from a set of nonterminals.
- The set of *terminals* is declared. Some terminals, called tokens, may have input attributes. The lexical structure is described separately.
- *Productions* are listed together with semantic functions. A semantic function is given by an expression or by a subroutine written in the implementation language.
- The description is completed with one or more program *modules* written in the implementation language including attribute types, semantic functions and any other elements as constants, variables, subroutines. This makes it possible for the user to implement dynamic data structures and global program objects.

We mention here that an *augmented metalanguage* is defined in [26] containing such elements as regular right hand side productions (sometimes called as *extended cf grammar*), augmented semantic functions for such productions, global *table definitions*, structured dynamic *data type* declarations embedded in a *block structured modular* metalanguage.

- The metalanguage of *PROF-LP* augmented with modularity and block structure is applicable.
- We introduce four levels of modularity:

  Metalanguage level. A module is a usually large part of the attribute grammar described in one input file and processed at the same time. A module is formed from a set of nonterminals.

  Process level. A module is a – possibly different – part of the generated compiler implemented in one process. The user may develop some other processes containing the same elements as it is usual in *PROF-LP*. Configuration of these physical processes are up to the user.

  Tree level. A tree module is a connected part of the syntax tree determined by selected nonterminals. It is the basis of tree based distribution. Tree level modularity should be compatible with source fragmentation.

  Task level. A task is an elementary part of evaluation, target of automatic load balancing. A task is a set of attribute instances defined by the user.

  The first two levels are applicable only in large systems. These two levels are incomparable, either a metalanguage module may contain more processes or a process may be composed from more modules.
- Production descriptions are applicable in their original form.
- We do not consider the lexical description here.

Formal consistency of the specification can be checked in the same way as it is usual in the case of sequential compilers. Checking correctness of semantic functions against the requirements of the implementation language is left to the compiler.

4.2 General Structure of the Generated Parallel Compiler

The generated compiler consists of three parts: A static *kernel* contains basic routines, the *attribute evaluator* is generated from the specification, *user supplied parts* are copied into the system without any change.

- The *kernel* contains the following routines.

  Input-output and distribution. In this paper we do not deal with the syntactic parser part of the compiler, so we suppose that the syntax tree is *available*. As we use a dynamic load balancing method, the whole syntax tree should be sent to each processor first. The result – the values of synthesized attribute instances of the root symbol – are sent to the host.

  Task scheduler and load balancer. The dynamic *load balancer* given in [24] can be applied as follows. A task means evaluation of a set of attribute instances. Two attribute instances $N.a$ and $M.b$ are in the same set if and only if the following conditions hold:

- The nodes M and N are in the *same tree module*, that is, there are not selected nonterminals along the path between N and M in the syntax tree.
- Attributes a and b are in the *same attribute set* declared by the user.
- The attribute instances $N.a$ and $M.b$ are *dependent* on each other. As it is very hard to check this condition, we can use another conditions instead.
  * We can use Kuiper's algorithm [16] which decides whether any two instances of two attribute occurrences may be dependent.
  * We can use Jordan's dependency based dynamic distributor [10]. In its original form it is based on local dependencies of a single production. It is easy to extend it to check dependencies of a subtree (tree module).

  We suggest a *more simple* method instead. We can use Jordan's method to form elementary tasks. The problem is that only attribute instances evaluated in the same production are allocated to the same task. After that we can form the unions of these – small – tasks using tree based distribution.

The same universal evaluator algorithm is running on each processor. The load balancer distributes tasks among processors. The evaluation starts on a single processor with tasks belonging to the *root* of the parsing tree. When a task has become executable – that is, all its input attribute instances are *available* – the processor sends this task to the one of its neighboring nodes. The node is selected on the numbers of other tasks waiting for execution. Leaving a tree module means that virtually all tasks evaluating attribute instances of the module just entered are sent away.

Executing a semantic function may need an extremely long time, others may be divided into smaller parts. Rutins handling tasks – insert a new task to the waiting list, declaring input and output parameters, etc. – are available for the user.

Error handling routines. All error messages are sent to the host computer.

- The *evaluator* contains a branch for each task containing semantic functions evaluating the set of attribute instances belonging to this task. It may start other tasks, as well. An evaluator is generated from a process level module. The evaluator is called by the load balancer whenever a task is started.
- The routines containing user written semantic functions are simply copied into the system. They may send tasks for the load balancer for execution.

4.3 Compiler Development Environment

The compiler development environment contains the following modules.

Metalanguage parser: checking the formal correctness of the specification.

Dependency analyser: computing attribute dependencies and checking its properties against the requirements of the evaluation strategy.

Distribution analyser checking dependencies among tree modules, attribute sets and tasks. It can help the user choosing a proper distribution strategy.

Code generator: generating the evaluator.

Developer utilities: helping the user developing semantic functions.

Execution utilities: helping the user configuring and executing the generated system.

The development process can be run on the host compiler. We mention that some suggestions to develop parallel compiler-compilers can be found in [3]. As can be seen, the structure of the compiler-compiler is very similar to the structure of a sequential system.

4.4 Technical Issues

We have started the implementation of the parallel compiler development environment with developing a small *prototype* for semantic check of a simple block structured language. Using the prototype, we get a statement by statement *specification* of the generated system as well as the kernel of the compiler. We implement it in parallel C running on a network of 16 T8000 processors.

Meanwhile an *attribute grammar specification* of the metalanguage is under development. We will generate the metalanguage parser from this specification using the compiler generator *PROF-LP*. The whole system will run on IBM PC under DOS, the implementation language is Turbo Pascal. As the host computer of our transputer is a Unix machine we have to transfer the generated compiler to the host. It may cause some technical problems.

The implementation of the whole system needs a lot of time and manpower. Practical *experiences* will be available after the completion of the implementation.

5 Final Remarks

In this paper we considered the problem of developing *parallel compilers* running on *transputer* architecture. Our most important conclusion is that we should give a lot of *freedom* to the user during developing such compilers. Only the user has enough knowledge to make basic decisions on attribute distribution. However some steps of development can be done *automatically*. Moreover we can help the user's work with the results of some *test algorithms*.

We stated the most important requirements for a compiler-compiler for developing parallel compilers. The basic structure of the compiler and the compiler-compiler has been described.

The first version of the system is under implementation now. Moreover we should consider the following questions in the future.

- How our generated semantic analyser can be combined with *parsing*? The results of Klein and Koskimies [13], [14] also may help solving this problem.
- Which methods and *algorithms* can be used in parallel compilers? For example what kind of *symbol table handling* methods are suitable? Have these methods any consequence for the structure of the compiler?
- The basic motivation of our research was to contribute in developing softwares for *IDIOMS* machine. We should go on in this direction as well.
- It is also important to find other *application fields*, where a compiler running on transputer is suitable and efficient.

Finally we thanks to *Lajos Schrettner* for his valuable remarks and suggestions.

References

[1] Akker, R., H. Alblas, A. Nijholt, P. O. Luttighuis, K. Sikkel: An annotated bibliography on parallel parsing, updated version, Technical Report, Dept. of Computer Science, Univ. of Twente, 92-84, 1992.

[2] Alblas, H.: Attribute evaluation methods, in Proc. of SAGA, Prague, 1991. LNCS 545., pp 48-113.

[3] Alblas, H.: A blueprint for a parallel parser generator, Technical Report, Dept. of Computer Science, Univ. of Twente, 92-65, 1992.

[4] Alblas, H., R. Akker, P. O. Luttinghuis, K. Sikkel: A bibliography on parallel parsing, in ACM Sigplan Notices, Vol. 29, No. 1., 1994. pp 54-65

[5] Almási, J., T. Horváth, M. Medvey, J. Toczki: On the implementation of cellular software development system, in Proc. of PARCELLA 88, Berlin, 1988.

[6] Deransart, P., M. Jourdan, B. Lorho: Attribute grammars, systems and bibliography, LNCS 323., 1988.

[7] Engelfriet, J.: Attribute grammars: Attribute evaluation methods, in Methods and tools for compiler construction, Cambridge Univ. Press, 1984, pp. 103-138.

[8] Gross, T., A. Zobel, M. Zolg: Parallel Compilation for a Parallel Machine, in Proc. of SIGPLAN '89, SIGPLAN Notices 24, 7 (1989), pp 91-100.

[9] Jazayeri, M., K. G. Walter: Alternating semantic evaluator, In Proc. of ACM 1975 Annual Conf., 1975., pp. 230-234.

[10] Jourdan, M.: A survey of parallel attribute evaluation methods, in Proc of SAGA, Prague, 1991., LNCS 545., pp 234-254.

[11] Kastens, U.: Ordered attribute grammars, Acta Informatica 13, 1980, pp. 229-256.

[12] Kerridge, J. M.: The design of the IDIOMS parallel database machine, in Proc. of British National Conf. on Databases 9, 1991.

[13] Klein, K., K. Koskimeies: Parallel one pass compilation, in Proc of WAGA, Paris 1990, LNCS 461., pp 76-90.

[14] Klein, K., K. Koskimies: How to pipeline parsing with parallel semantic analysis, Structured Programming 13, 1992., pp 99-107.

[15] Knuth, D. E.: Semantics of context-free languages, Math. Systems Theory 2, 1968., pp. 127-145, correction Math. Systems Theory 5, 1971. pp 95-96.

[16] Kuiper, M. F.: Parallel attribute evaluation, Ph. D. Thesis, Fac. of Informatics, Univ. of Utrecht, 1989.

[17] Kuiper, M. F., A. Dijkstra: Attribute evaluation on a network of transputers, in Wexler (ed.): Developing transputer applications, Amsterdam, 1989., pp 142-149.

[18] Kuiper, M. F., D. Swierstra: Parallel Attribute Evaluation: Structure of Evaluators and Detection of Parallelism, in Proc. of WAGA 90, Paris, 1990., LNCS 461, pp. 61-75.

[19] Lipkie, D. E.: A compiler design for multiple independent processor computers. Ph.D. Th. Dept. of Computer Science, Univ. of Washington, Seattle, 1979.

[20] Miller, J. A., R. J. LeBlanc: Distributed compilation: a case study, in Proc. of the Third Int. Conf. on Distributed Computing Systems, (1982), pp. 548-553.

[21] PROF-LP User's Guide, Research Group on Theory of Automata, Szeged, 1987.

[22] Schell, R. M.: Methods for constructing parallel compilers for use in a multi-processor environment, Ph.D. Th., Rep. No. 958, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1979.

[23] Seshardi, V., D. B. Wortman: An investigation into Concurrent Semantic Analysis, Software, Practice and Experience Vol. 21. No. 12. (1991) pp. 1323-1348.

[24] Schrettner, L., J. Toczki: Dynamic Load Balancing for Decomposable Problems, Proc. of Workshop on Parallel Processing in Education, Impact Tempus JEP/Hungarian Transputer Users Group, Miskolc, 1993.

[25] Toczki, J., T. Gyimothy, G. Jahni: Implementation of a LOTOS precompiler, in Proc. of PD 88, Budapest, 1988.

[26] Toczki, J.: Attribute grammars and their applications, (in Hugarian), Dr. Univ. Thisis, Depts. of Informatics, József Attila Univ. of Szeged, 1991.

[27] Vandervoorde, M. T.: Parallel Compilation on a Tightly Coupled Multiprocessor, SRC Reports No. 26, Digital Systems Research Center, 1988.

Received October, 1994

Correction

I am very sorry to inform our readers that the footnote on page 121 of the paper

Mealy-automata in which the output-equivalence is a congruence

I. Babcsányi A. Nagy

appeared in Volume 11 Number 3 of Acta Cybernetica, was edited by mistake, and it actually belongs to another paper. The correct footnote to the above mentioned paper is "This work was supported by the Hungarian National Foundation for Scientific Research (OTKA) grant No. 7608." I apologize both the authors and the readers.

Zoltán Fülöp
Managing Editor

Correction

I am very sorry to inform our readers that the references on page 332 of the paper

Invariance Groups of Threshold Functions

E. K. Horváth

appeared in Volume 11 Number 4 of Acta Cybernetica, were misprinted. Hereby we provide the proper form of it.

References

[1] N. N. Aĭzenberg, A. A. Bovdi, È. I. Gergo, F. È. Geche, *Algebraic aspects of threshold logic,* Kibernetica (Kiev) no. 2 (1980), 26-30 (Russian); English transl. in Cybernetics **16** no. 2 (1980), 188-193.

[2] S. Yajima and T. Ibaraki, *A lower bound of the number of threshold functions,* Trans.IEEE, EC-14 no. **6** (1965), 926-929.

[3] Ching Lai Sheng, *Threshold logic* (H.G. Booker and N. DeClaris, eds.) Academic Press, London and New York, 1969.

I apologize both the author and the readers.

Zoltán Fülöp
Managing Editor

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* summarising 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 1995 Numbers 1–2 of Volume 12 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-6720 Szeged, P.O.Box 652, Hungary. Tel.: (36)-(62)-311-184, Fax:(36)-(62)-312-292.

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János
A kézirat a nyomdába érkezett: 1995. szeptember
Terjedelem: 7,12 (B/5) ív