TOWARDS A PROGRAMMING LANGUAGE BASED ON THE NOTION OF TWO-LEVEL GRAMMAR*

Jan MAŁUSZYŃSKI

Software Systems Research Center, Linköping University, 581 83 Linköping, Sweden, and
Institute of Computer Science, Polish Academy of Sciences, 00-901 Warsaw PKiN, Poland

Communicated by M. Sintzoff
Received July 1982
Revised December 1982

Abstract. This paper deals with the problem of computing relations from their abstract non-algorithmic specifications. The formalism under consideration is that of two-level grammars, introduced originally for defining languages. In this formalism the intuitions behind the formal definition can be directly expressed by the grammatical rules, which may have a form close to the statements of natural language. A notion of the relation specified by a two-level grammar is introduced and computability of such relations is discussed. For a class of two-level grammars, called the transparent grammars, an algorithm is outlined for computing the relations specified by the grammars of this class. The transparent grammars turn out to be a generalization of the formalism of Horn clauses, and the algorithm is based on unification. The language generated by a two-level grammar can be used as an additional tool for controlling computations. A transparent two-level grammar can be considered as a non-algorithmic program specifying an input/output relation. The computational algorithm defines an operational semantics of such programs.

1. Introduction

This paper deals with a non-algorithmic formalism for specifying relations and with the problem of computing relations defined in this formalism. Having solved it one can consider the formalism to be a non-algorithmic programming language; the specification of a relation in this formalism to be a program; and the relation specified to be the input/output relation of this program.

The formalism we deal with is that of two-level grammars introduced originally for defining languages [25]. However, a two-level grammar can also specify a relation. Examples of such definitions can be found in the Revised Report on ALGOL 68 [26], where among others a definition of the equivalence relation on the recursive types is given. The descriptive power of two-level grammars enables one to consider the formalism as a general model for computations [21]. At the same time, two-level grammars make it possible to express directly the intuitions behind the formal...
definition of a relation since the components of the grammatical rules can be designed in a form close to the statements of natural language. These properties make the formalism of two-level grammars a candidate for a non-algorithmic programming language. An early attempt of such an application is the system implemented by Chastelier and Colmerauer [3] but the class of two-level grammars which can be used in this system is very restricted. Another class of two-level grammars is considered by Sintzoff [22], who outlines a method for computing relations specified by the grammars of this class. In both approaches the relation specified by a given grammar is computed by parsing strings of the language generated by the grammar (though in [3] the notion of the relation specified by a two-level grammar does not appear explicitly). A possibility of using the formalism of two-level grammars for defining functions was pointed out by Kupka [10]. In this paper the strings of the language generated by the grammar specifying a function are used to represent the values of the function.

An important step in the development of concepts related to the notion of two-level grammar is the idea to superimpose a tree structure on the hypernotions of a two-level grammar. This idea appears in the notion of affix grammar introduced by Koster [8] and modified by Watt [23]. It is also present in the work of Sintzoff [22] and its explicit formulation by Simonet [20] resulted in a new class of grammars, called RW-grammars. The relation specified by an RW-grammar can be computed by a PROLOG program which can be obtained from the RW-grammar by its compilation.

This paper presents yet another approach to the problem of computing relations specified by two-level grammars: the hyperrules are neither transformed into reduction rules nor compiled into Horn clauses but they are used directly in the computational process. To make this possible a restriction on the structure of hyperrules is formulated, which is similar to one of the structuring conditions in [22]. It is also very close to the restrictions formulated by Watt [23] and to those of Simonet [20]. Two-level grammars fulfilling this restriction are called transparent two-level grammars. The class of transparent two-level grammars is very closely related to the class of extended affix grammars. It can also be considered as a generalization of the formalism of Horn clauses, and the method for computing relations specified by transparent two-level grammars described in this paper resembles the idea of the procedural interpretation of Horn clauses [9]. However, in contrast to a set of Horn clauses a two-level grammar defines not only a relation but also a language. In this respect, transparent two-level grammars are similar to the definite clause grammars [4, 16]. However, a DCG is usually considered as a 'syntactic sugar' for a set of Horn clauses [16, p. 244], while in our approach the language defined by a transparent grammar is handled in a different way and it may be used to control computation of the relation specified by the grammar.

The paper is organized as follows. Section 2 gives an informal introduction to the concept of two-level grammar and introduces formal definitions of the language and of the relation specified by a two-level grammar.
A two-level grammar is usually viewed as a context-free grammar with an infinite number of production rules, specified by a finite number of rule schemata, called the hyperrules. Since the notion of the relation specified by a two-level grammar is defined in terms of the derivations, for computing such relations it is necessary to find an effective method for constructing derivations. The conventional methods are not applicable since, in contrast to the usual case, one has to deal with an infinite set of the production rules. In Section 3 the idea of constructing derivations by direct application of the hyperrules is considered. It leads to an undecidable string matching problem called the grammatical unification problem. This problem is discussed in Section 4. It is shown that for a class of grammars called transparent two-level grammars the grammatical unification problem is decidable and can be reduced to the usual term unification problem [18]. Thus, using the hyperrules of a transparent two-level grammar and a usual term unification algorithm one can construct derivations of the grammar. On the other hand, it is shown that every recursively enumerable set can be generated by a transparent two-level grammar.

Section 5 deals with the relationship between Horn clauses and transparent two-level grammars. It is shown that a finite set of Horn clauses can be considered as a transparent two-level grammar.

Section 6 outlines an algorithm for computing relations specified by transparent two-level grammars. The algorithm is based on unification. The use of transparent two-level grammars for specifying input/output relations is discussed in Section 7.

2. Two-level grammars as a generalization of BNF

2.1. Informal introduction

The notion of two-level grammar is a natural generalization of the notion of context-free grammar. We explain the idea first informally. Consider the following set of context-free production rules specified in BNF:

\[ (\text{arithmetic expr}) ::= (\text{simple arithmetic expr}) | (\text{if clause})(\text{simple arithmetic expr}) \text{ else (arithmetic expr)} \]
\[ (\text{boolean expr}) ::= (\text{simple boolean expr}) | (\text{if clause})(\text{simple boolean expr}) \text{ else (boolean expr)} \]
\[ (\text{designational expr}) ::= (\text{simple designational expr}) | (\text{if clause})(\text{simple designational expr}) \text{ else (designational expr)} \]

One can abbreviate the above specification by introducing the parameter \( X \) ranging over the finite language \( L_X = \{\text{arithmetic, boolean, designational}\} \):

\[ (X \ expr) ::= (\text{simple } X \ expr)(\text{if clause})(\text{simple } X \ expr) \text{ else } (X \ expr) \]

Each of the original production rules can be obtained from the parameterized rule scheme by replacing consistently all occurrences of the parameter by one arbitrary...
element of its domain. This notation can be extended by introducing parameters whose domains are possibly infinite context-free languages specified by context-free grammars. A finite set of the parameterized rule schemata with a finite set of context-free production rules specifying the domains of the parameters defines an infinite set of the resulting production rules. This is the idea of the formalism of two-level grammars.

Thus, a two-level grammar can be considered to be a context-free grammar with a possibly infinite set of the production rules. As a consequence the basic notions of the formalism, like the notion of derivation and the notion of languages specified by a grammar, can be defined in the same way as in the case of context-free grammars.

Properties of two-levels grammars as a formalism for specifying languages have been discussed in many papers (e.g., [5, 21, 25] to mention only a few). On the other hand, it is known that two-level grammars can also be used to specify relations (see, e.g., [6, 10, 22, 26]). The example which follows should explain the intuition concerning such relations. It introduces also some notational conventions to be used in the sequel.

**Example 2.1.** Consider the following set of the parameterized BNF rules with parameters \(N1\) and \(N2\):

\[
\begin{align*}
(r_1) \quad \langle \text{bit string of length } N1 \text{ with value } N2 N2 \rangle &::=
\langle \text{bit string of length } N1 \text{ with value } N2:0 \\
(r_2) \quad \langle \text{bit string of length } N1 \text{ with value } N2 N2 \rangle &::=
\langle \text{bit string of length } N1 \text{ with value } N2:1 \\
\epsilon &::= \langle \text{bit string of length with value } \rangle
\end{align*}
\]

(where \(\epsilon\) denotes the empty string).

In this paper parameters are always denoted by capital letters with decimal suffixes and the convention is adopted that the parameters beginning with the same capital have identical domains (cf. the convention used in the Algorithm 68 Revised Report [26]). In that way the definitions of the domains can be abbreviated.

The domain of parameters \(N1\) and \(N2\) is defined as follows:

\[ N::=N1|\epsilon \]

Thus \(N\) consists of all possible sequences of the symbol \(i\), including the empty sequence. Such sequences are used in our example as the unary representations of the natural numbers.

Since the domains of the parameters are infinite the schemes \((r_1)\)–\((r_3)\) specify an infinite set of context-free production rules. An example of such a rule is

\[
\langle \text{bit string of length } iii \text{ with value } iii \rangle::=\langle \text{bit string of length } ii \text{ with value } ii \rangle\epsilon
\]

To define the language specified by a set of context-free production rules a start nonterminal should be chosen. Instead we choose an infinite set of the start non-
Towards a programming language based on two-level grammars

 terminals represented by the parameterized nonterminal

\[ h_0 = \text{(bit string of length } N1 \text{ with value } N2) \].

The language specified by the system consists of all terminal strings which can be derived from the start nonterminals. It is the regular language \{0|1\}^* of all 'bit strings'.

For specifying strings chosen as parameter values an explicit notation will be used: \([s_1/x_1, \ldots, s_n/x_n]\) means that the strings \(s_1, \ldots, s_n\) have been chosen as the values of the parameters \(x_1, \ldots, x_n\). Moreover, if \(h\) is a parameterized nonterminal and \(r\) a parameterized BNF rule, we denote by \(h[s_1/x_1, \ldots, s_n/x_n]\) and by \(r[s_1/x_1, \ldots, s_n/x_n]\) the nonterminal and the rule obtained by replacing the occurrences of each parameter \(x_i\) in \(r\) and in \(h\) by the string \(s_i\).

An example of a derivation of the two-level grammar specified above is the following sequence:

1. \(h_0[iii/N1, ii/N2] = \text{(bit string of length } iii \text{ with value } ii)\)
2. \(\langle \text{bit string of length } ii \text{ with value } i \rangle 0\)
3. \(\langle \text{bit string of length } i \text{ with value } \rangle 10\)
4. \(\langle \text{bit string of length with value} \rangle 010\)
5. \(010\)

The production rules used to construct it are the following instances of the original parameterized rules:

1. \(r_1[ii/N1, i/N2]\)
2. \(r_2[i/N1, f/N2]\)
3. \(r_1[f/N1, f/N2]\)
4. \(r_1[ ]\) (since \(r_1\) has no parameters).

Notice, that from certain parameter-free nonterminals no terminal string can be derived since no appropriate production rule can be obtained from the parameterized scheme. For example, the only string which can be derived from \(h_0[i/N1, ii/N2]\) is \(\langle \text{bit string of length with value } i \rangle 0\). Thus, the rules define a binary relation on \(L_N\) consisting of all such pairs \((x, y)\) that the start nonterminal \(h_0[x/N1, y/N2]\) derives a terminal string. It is the set of all pairs \((x, y)\) such that there exists a bit string of length \(|x|\) which is a binary representation of the natural number \(|y|\). In other words, the relation consists of all such pairs \((x, y)\) that \(2^{|x|} > |y|\). The intuition concerning this relation is expressed by the form of the start nonterminal scheme: all nonterminals which may be obtained by instantiating the scheme are of the form

\(\langle \text{bit string of length } i^k \text{ with value } i^m \rangle\)

where \(k \geq 0\) and \(m \geq 0\). It can be seen that any such a nonterminal derives at most one terminal string which is the binary representation of \(m\) augmented to the string of the length \(k\) by adding leading zeros.

The above example shows that a two-level grammar defines a linguistic relation whose arity is determined by the number of the occurrences of parameters in the start nonterminal.
2.2. Basic notions

We now introduce formal definitions of the notions discussed informally in Section 2.1. In order to relate our definitions to those known from the literature we adopt the following terminology:

- The auxiliary finite alphabet used to represent the nonterminals and to define domains of the parameters is called the orthovocabulary.
- The parameterized nonterminals are called the basic hypernotations.
- The parameterized BNF rules are called the basic hyperrules.
- We distinguish between the parameters occurring in the hyperrules, which are assumed to be indexed by (representations of) nonnegative integers according to the convention of Example 2.1, and those which are nonterminals of the context-free production rules specifying the domains. The former are called the grammatical variables, while the latter are called the metanotations of the grammar. The metanotation obtained by erasing the index of a grammatical variable is called the type of the variable.
- The start parameterized nonterminal of the grammar is called its start hypernotation. In contrast to the definitions of two-level grammars known from the literature we do not assume that the start hypernotation is parameter-free.

**Definition 2.2.** A two-level grammar $W$ is a 7-tuple $(X, M, T, H, R, Q, S)$ where

- $X$ is a finite orthovocabulary.
- $M$ is a finite set of metanotations.
- $T$ is a finite set of terminal symbols.
- $H$ is a finite set of basic hypernotations: it is a subset of the set $\{(h): h \in (M \times I) \cup X)*\}$, where 'C and 'r' are auxiliary symbols which do not belong to $X$ or $M$, and $I$ is the set of nonnegative integers.
- $R$ is a finite set of basic hyperrules: it is a subset of $H \times (H \cup T)^*$.
- $Q$ is a finite set of metarules: it is a subset of $M \times (M \cup X)^*$.
- $S \in H$ is the start hypernotation.

For the two-level grammar of Example 2.1:

- $X$ may be defined as the set of small letters: $a, b, \ldots, x, y, z$.
- $M = \{N\}$.
- $T = \{0, 1\}$.
- $H = \langle$ bit string of length $N1$ with value $N2 \cdot N2$, (bit string of length $N1$ with value $N2 \cdot N2$, (bit string of length $N1$ with value $N2 \cdot N2$), (bit string of length $N1$ with value $N2 \cdot N2$)\}$.
- $R$ consists of the hyperrules $(r_1), (r_2)$ and $(r_3)$.
- $Q$ is defined by the BNF rule $N::=N1|N0$ (i.e., $Q = \{(N, N1), (N, N0)\}$.
- $S = \langle$ bit string of length $N1$ with value $N2$\}$.

A two-level grammar is a finite specification of a possibly infinite set of context-free production rules as illustrated by Example 2.1. In order to formally define a production rule of a two-level grammar some auxiliary notions are introduced.
Towards a programming language based on two-level grammars

Let \( W = (X, M, T, H, R, 0, S) \) be a two-level grammar. The triple \((M, X, Q)\) will be called the *metagrammar* of \( W \). The metagrammar is a finite set of context-free production rules, in which \( M \) plays the role of the nonterminal alphabet while \( X \) is the terminal one.

By \( \Rightarrow_M \) we denote the binary relation of *direct derivability* in the metagrammar, which is a relation on \((M \cup X)^*\) and is defined as usual:

\[
x \Rightarrow_M y \text{ iff there exist strings } u \text{ and } v \text{ in } (M \cup X)^* \text{ and a metarule } (Z, w) \text{ such that } x = uZv \text{ and } y = uvw; \text{ e.g., for the grammar of Example 2.1, } N_2 \Rightarrow S_1.
\]

The reflexive and transitive closure of \( \Rightarrow_M \) is called the *derivability relation* in the metagrammar and is denoted \( \Rightarrow_M^* \); e.g., for the grammar of Example 2.1, \( N_i \Rightarrow^* \).

By a *grammatical variable* of \( W \) we mean any pair \((Z, n)\), where \( Z \) is a metanotion and \( n \) a nonnegative integer. According to the convention of Example 2.1 grammatical variables are represented as metanotions with decimal suffices, e.g., \( N_1 \) instead of \( (N, 1) \). The infinite set of the grammatical variables of \( W \) will be denoted by \( V \). If \( v = (Z, n) \) is a grammatical variable the metanotion \( Z \) is called the *type* of \( v \) and is denoted \( \text{Type}(v) \). We use the function \( \text{Type} \) to define a homomorphism \( \text{Form} \) on \((V \cup X \cup T \cup \{(,\})\)^* which 'removes the indices' of variables. It is defined as follows:

\[
\text{Form}(x) = \begin{cases} 
\text{Type}(x) & \text{if } x \in V, \\
 x & \text{otherwise},
\end{cases}
\]

e.g., for the grammar of Example 2.1,

\[
\text{Form}(\text{bit string of length } N_1 \text{ with value } N_2 N_2)) = (\text{bit string of length } N_1 \text{ with value } N N).
\]

Our definition of a production rule of a two-level grammar is based on a generalization of the usual notion of *consistent replacement*, called *hyperreplacement*.

By a *hyperreplacement* of \( W \) we mean any homomorphism \( \Theta \) on \((V \cup X \cup T \cup \{(,\})\)^* such that

1. for every \( x \in (X \cup T \cup \{(,\})\): \( \Theta(x) = x \).
2. the set \( D(\Theta) = \{v \in V: \Theta(v) \neq v\} \) is finite,
3. for every \( v \in D(\Theta): \text{Type}(v)_M \Rightarrow^* \text{Form}(\Theta(v)) \).

Notice that the composition of hyperreplacements is a hyperreplacement.

A hyperreplacement \( \Theta \) is called a *renaming* hyperreplacement iff the image of each element of \( D(\Theta) \) is a variable and it is different from the image of any other element of \( D(\Theta) \).

For representing hyperreplacements the notation of the Example 2.1 is used, e.g., a hyperreplacement \( \Theta \) such that \( D(\Theta) = \{N_1, N_2\} \) and \( \Theta(N_1) = N_3, \Theta(N_2) = N_1 \) is denoted by \([N_3/N_1, N_1/N_2]\).

A hyperreplacement \( \Theta_1 \) is said to be *more general* than a hyperreplacement \( \Theta_2 \) iff there exists a hyperreplacement \( \Theta_3 \) such that \( \Theta_2 = \Theta_1 \circ \Theta_3 \). For example, for the grammar of Section 2.1 the hyperreplacement \( \Theta_1 = [N_3i/N_1] \) is more general than
the hyperreplacement $\Theta_2 = [N4i/N1, N4i/N3, N5i/N2]$ since $\Theta_2 = \Theta_1 \cdot \Theta_3$ for $\Theta_3 = [N4i/N3, N5i/N2]$. This relation is transitive and reflexive but it is not antisymmetric, e.g.,

$$[N1/N2] = [N2/N1] \cdot [N1/N2] \quad \text{and} \quad [N2/N1] = [N1/N2] \cdot [N2/N1].$$

A hyperreplacement $\Theta$ is said to be a *most general* hyperreplacement in a class of hyperreplacements iff for every hyperreplacement $\Theta'$ in this class $\Theta$ is more general than $\Theta'$.

The images of basic hypernotions under hyperreplacements are called *hypernotions* of the two-level grammar, e.g., for the grammar of Example 2.1, *(bit string of length $N3i$ with value $i$)* is a hypernotion but it is not a basic hypernotion. Clearly, the image of a hypernotion under a hyperreplacement is also a hypernotion. The set of all hypernotions of a two-level grammar whose set of the basic hypernotions is $H$ will be denoted $H'$.

By the image of a pair $(x, y)$ of strings under a hyperreplacement $\Theta$ we mean the pair $(\Theta(x), \Theta(y))$. The images of basic hyperrules of a two-level grammar under hyperreplacements are called *hyperrules* of the grammar, e.g., for the grammar of Example 2.1,

*(bit string of length $N3i$ with value $N4i$)*

is a hyperrule but it is not a basic hyperrule. Clearly, the image of a hyperrule under a hyperreplacement is also a hyperrule.

The images of hypernotions under hyperreplacements are called *instances* of these hypernotions. By a *ground* instance of a hypernotion we mean any such instance in which no grammatical variable occurs. The same terminology will also be used for the images of hyperrules.

The (possibly infinite) set of all ground instances of the hypernotions of a two-level grammar $W$ is called the set of *nonterminals* of $W$ and is denoted by $N$, e.g., *(bit string of length $ii$ with value $i$)* is a nonterminal of the grammar of Example 2.1.

Ground instances of the start hypernotion are called *start nonterminals* of the grammar.

Ground instances of basic hyperrules of a two-level grammar are called *production rules* of the grammar. Note that ground instances of hyperrules are also production rules. Thus, in contrast to the usual definition, we are able to construct production rules through stepwise refinement of hyperrules.

We extend the notion of context-free derivation for the case of two-level grammars. For this we first define two auxiliary binary relations on $(N \cup T)^*$, namely $\Rightarrow$ (which, as usual, determines single steps of derivations) and $\Rightarrow^*$ (which determines single steps of left-most derivations). The relations are defined as follows:

$z \Rightarrow y$ iff there exist strings $u$ and $w$ in $(N \cup T)^*$ and a production rule $(Z, v)$ such that $z = uZw$ and $y = uvw$. 

Towards a programming language based on two-level grammars

- \( z \Rightarrow y \) if there exist strings \( u \) in \( T^* \) and \( w \) in \( (N \cup T)^* \) and a production rule \((Z, \rho)\) such that \( z = uZw \) and \( y = uw. \)

A derivation in a two-level grammar is any sequence \( z_0, z_1, \ldots, z_n \) of strings in \((N \cup T)^* \) such that \( n \geq 0 \) and \( z_i \Rightarrow z_{i+1} \) for \( 0 \leq i \leq n. \)

A left-most derivation in a two-level grammar is any sequence \( z_0, z_1, \ldots, z_n \) of strings in \((N \cup T)^* \) such that \( n \geq 0 \) and \( z_i \Rightarrow z_{i+1} \) for \( 0 \leq i \leq n. \)

The transitive and reflexive closure of \( \Rightarrow \) is denoted by \( \Rightarrow^* \). It is easy to see that \( z \Rightarrow^* y \) if there exists a derivation \( z_0, \ldots, z_n \) such that \( z_0 = z \) and \( z_n = y. \)

Definition 2.3. The language \( L(W) \) generated by a given two-level grammar \( W \) is the set of all terminal strings which can be derived from the start nonterminals of \( W \), i.e.,

\[
L(W) = \{ x \in T^* : \exists s \in A, s \Rightarrow^* x \}
\]

where \( A \) is the set of all start nonterminals of \( W. \)

It should be noted that \( z \in L(W) \) if there exists a left-most derivation whose first element is some start nonterminal of \( W \) and whose last element is \( z. \)

Let \( x_0, \ldots, x_m \) for some \( m \geq 0 \) be such strings in \( X^* \) that the start hypernotion \( S \) of the two-level grammar is of the form \( (x_0v_1 \cdots v_mx_m) \) where the \( v_i \) \( (i = 1, \ldots, m) \) are grammatical variables.

Definition 2.4. The relation \( Rel(W) \) determined by a two-level grammar \( W \) is the \( m \)-ary relation on \( X^* \) defined as follows:

\( (z_1, \ldots, z_m) \in Rel(W) \) if \( (x_0z_1 \cdots z_mx_m) \) is a start nonterminal of the grammar and there exists a string \( w \) in \( T^* \) such that \( (x_0z_1 \cdots z_mx_m) \Rightarrow^* w. \)

Thus, to find an element of \( Rel(W) \) it suffices to construct a left-most derivation of an element of \( L(W) \): any element of \( Rel(W) \) can be obtained in this way.

3. Hyperderivations

In this section we deal with the problem of constructing derivations of two-level grammars. A derivation whose first element is a start nonterminal of the grammar and whose last element is a terminal string is a demonstration that the string belongs to the language specified by the grammar. Moreover, according to Definition 2.4 it determines an element of the relation specified by the grammar. Thus, the ability to construct such derivations is the ability to compute elements of the language and elements of the relation specified by the grammar. However, the methods for constructing derivations of context-free grammars cannot be applied in this case since the set of the production rules of a two-level grammar is usually infinite. Therefore, in this section we consider the possibility of constructing derivations of a two-level grammar directly from the basic hyperrules of the grammar.
We first define two auxiliary binary relations $\rightarrow$ and $\Rightarrow$ on $(H' \cup T)^*$ as follows:

- $x \rightarrow y$ iff there exist strings $u$ and $w$ in $(H' \cup T)^*$ and a hyperrule $(h, z)$ such that $x = uhw$ and $y = uzw$.

- $x \Rightarrow y$ iff there exist strings $v$ in $T^*$ and $w$ in $(H' \cup T)^*$ and a hyperrule $(h, z)$ such that $x = hvw$ and $y = uzw$.

A hyperderivation in a two-level grammar $W$ is any sequence $x_0, x_1, \ldots, x_n$ of strings in $(H' \cup T)^*$ such that $n \geq 0$ and $x_i \rightarrow x_{i+1}$ for $i = 0, 1, \ldots, n - 1$.

A left-most hyperderivation in a two-level grammar $W$ is any sequence $x_0, x_1, \ldots, x_n$ of strings in $(H' \cup T)^*$ such that $n \geq 0$ and $x_i \Rightarrow x_{i+1}$ for $i = 0, 1, \ldots, n - 1$.

From these definitions we obtain at once the following.

**Proposition 3.1.** Every derivation in a two-level grammar $W$ is a hyperderivation in $W$.

If $x = (x_0, \ldots, x_n)$ is a hyperderivation and $\Theta$ is a hyperreplacement, then the sequence $(\Theta(x_0), \ldots, \Theta(x_n))$ is called an instance of $x$ and is denoted by $\Theta(x)$.

**Proposition 3.2.** An instance of a hyperderivation is a hyperderivation.

In particular, if $x = (x_0, \ldots, x_n)$ is a hyperderivation and $\Theta$ is a hyperreplacement such that for every variable $v$ occurring in an element of the hyperderivation $\Theta(v) \in X^*$, then $\Theta(x)$ is a derivation.

**Example 3.3.** The following sequence is an example of a hyperderivation of the grammar of Example 2.1:

1. (bit string of length $N0i$ with value $iii$)
2. (bit string of length $N0i$ with value $ii1$)
3. (bit string of length $N0$ with value $i01$)

The hyperrules which are used to transform (1) into (2) and (2) into (3) are the following instances of the basic hyperrules:

1. $r_1[N0i/N0, ii/N2]$.
2. $r_1[N0i/N0i, i/N2]$.

A left-most hyperderivation whose first element is an instance of the start hyperrule and whose last element is a terminal string will be called a standard hyperderivation. If we are able to construct such a hyperderivation, we get an element of $L(W)$ (which is the last element of the hyperderivation) and a subset of $Rel(W)$, since every ground instance of the first element of the hyperderivation determines an element of $Rel(W)$. Since every element of $L(W)$ and every element of $Rel(W)$ can be determined by some standard hyperderivation, we want to find a method for constructing such hyperderivations.

A hyperderivation $y_0, y_1, \ldots, y_n, y_{n+1}$ is called a direct extension of a hyperderivation $x_0, x_1, \ldots, x_n$ iff there exists a hyperreplacement $\Theta$ such that, for $i = 0, 1, \ldots, n$, $y_i = \Theta(x_i)$. 


Towards a programming language based on two-level grammars

A hyperderivation $x_0, \ldots, x_n$ has a direct extension iff there exists a basic hyperrule $(i, z)$, a hypernotion $h'$ occurring in $x_n$ and hyperreplacements $\Theta$ and $\Theta'$ such that $\Theta(h) = \Theta'(h')$. In that case $x_n = uh'w$ for some $u \in (H' \cup T)^*$ and the sequence $\Theta(x_0), \ldots, \Theta(x_n), \Theta(u)\Theta'(z)\Theta(w)$ is a direct extension of $x_0, \ldots, x_n$.

A hyperderivation $y$ is said to be an extension of a hyperderivation $x$ iff there exists a sequence of direct extensions beginning with $x$ and ending with $y$.

To construct a standard hyperderivation in a two-level grammar one can start with the one-element hyperderivation consisting of the start hypernotion of the grammar and try to construct a sequence of its direct left-most extensions until a hyperderivation is obtained whose last element is a terminal string. Since the set of basic hyperrules of the grammar is finite, the existence of a direct left-most extension of a given hyperderivation can be checked if the following problem can be solved: For given arbitrary hypernotions $h$ and $h'$, check whether there exist hyperreplacements $\Theta$ and $\Theta'$ such that $\Theta(h) = \Theta'(h')$. In the sequel we shall assume that the hypernotations $h$ and $h'$ have no common variables. If this is not the case, we may apply to one of them an appropriate renaming hyperreplacement. Under this assumption the problem reduces to searching for a one hyperreplacement $\Theta$ such that $\Theta(h) = \Theta(h')$.

**Definition 3.4.** The problem

"For given arbitrary hypernotions $h$ and $h'$ check whether there exists a hyperreplacement $\Theta$ such that $\Theta(h) = \Theta(h')"$

is called the grammatical unification problem. If such a $\Theta$ exists, it is called a unifier of $h$ and $h'$ and in this case $h$ and $h'$ are said to be unifiable.

The grammatical unification problem is a kind of string equation problem. However, in the classical string equation problem [19, p. 157] the domain of every variable is the set of all strings over some given alphabet while in the case of grammatical unification the domains of variables are context-free languages and the domains of different variables may be different.

**Example 3.5.** To construct a direct extension of the hyperderivation of Example 3.3 one has to find a basic hyperrule such that its left-hand side and the hypernotion \( \langle \text{bit string of length } N0 \text{ with value } i \rangle \) are unifiable. One can check that hyperrule $r_1$ is not appropriate but $r_2$ has the required property: Comparing the hypernotations \( \langle \text{bit string of length } N0 \text{ with value } i \rangle \) and \( \langle \text{bit string of length } N1i \text{ with value } N2 \rangle \), we get the conditions $\Theta(N0) = \Theta(N1i)$ and $\Theta(i) = \Theta(N2 N2 i)$. The hyperreplacement $[N1i/N0, r/N2]$ is a solution of the equations and results in the following direct extension of the hyperderivation of Example 2.1:

1. \( \langle \text{bit string of length } N1i \text{ with value } iiiii \rangle \)
2. \( \langle \text{bit string of length } N1i \text{ with value } ii \rangle \)
This hyperderivation can be further extended only by the hyperrules \( r_1 \) or \( r_3 \). Consider first the hyperrule \( r_1 \). As the variable \( NI \) occurs both in the hypernotion \( \langle \text{bit string of length } N1i \text{ with value } i \rangle \) and in the left-hand side of the hyperrule \( r_1 \) we remove it from the hyperderivation by application of the renaming hyperreplacement \([NI/0] \): 

\[
\begin{align*}
(1) & \langle \text{bit string of length } N0iii \text{ with value iiiii} \rangle \\
(2) & \langle \text{bit string of length } N0ii \text{ with value ii} \rangle 1 \\
(3) & \langle \text{bit string of length } N0i \text{ with value i} \rangle 01 \\
(4) & \langle \text{bit string of length } N0 \text{ with value} \rangle 101 
\end{align*}
\]

The hyperreplacement \([NIi/N0, \epsilon/N2] \) is a unifier for the hypernotion occurring in (4) and for the left-hand side of \( r_1 \). Using this unifier the hyperderivation can be extended as follows:

\[
\begin{align*}
(1) & \langle \text{bit string of length } N1iii \text{ with value iiiii} \rangle \\
(2) & \langle \text{bit string of length } N1ii \text{ with value ii} \rangle 1 \\
(3) & \langle \text{bit string of length } N1i \text{ with value i} \rangle 01 \\
(4) & \langle \text{bit string of length } N1 \text{ with value} \rangle 101 \\
(5) & \langle \text{bit string of length } N1 \text{ with value} \rangle 0101 
\end{align*}
\]

Using the hyperrule \( r_3 \) and the hyperreplacement \([\epsilon/N1] \) as a unifier we can construct the following direct extension of this hyperderivation:

\[
\begin{align*}
(1) & \langle \text{bit string of length } iii \text{ with value iiiii} \rangle \\
(2) & \langle \text{bit string of length } iii \text{ with value ii} \rangle 1 \\
(3) & \langle \text{bit string of length } ii \text{ with value i} \rangle 01 \\
(4) & \langle \text{bit string of length } i \text{ with value} \rangle 101 \\
(5) & \langle \text{bit string of length with value} \rangle 0101 \\
(6) & \quad 0101 
\end{align*}
\]

In that way we have constructed a standard hyperderivation which is also a derivation in the two-level grammar and we have obtained the element \((iii, iiiii)\) of the relation specified by the grammar.

4. The unification problem and transparent two-level grammars

This section deals with the unification problem, whose decidability is essential for constructing standard hyperderivations in two-level grammars. However, in the general case this problem is undecidable since its particular case can be reduced to a classical undecidable problem. Let \( G_1 = (N_1, T_1, P_1, S_1) \) and \( G_2 = (N_2, T_2, P_2, S_2) \) be arbitrary context-free grammars such that \( S_1 \neq S_2 \), and let \( W \) be a two-level grammar whose set of metarules is \( P_1 \cup P_2 \) and such that \( \langle S_1 \rangle \) and \( \langle S_2 \rangle \) are its
Towards a programming language based on two-level grammars

Clearly, a grammatical unifier for \((S_1)\) and \((S_2)\) exists iff the intersection of the context-free languages \(L(G_1)\) and \(L(G_2)\) is nonempty, what is known to be undecidable.

In the sequel a class of two-level grammars is defined, for which the grammatical unification problem is decidable.

**Definition 4.1.** A two-level grammar \(W = (X, M, T, H, R, Q, S)\) is called a transparent two-level grammar iff there exists a metanotion \(K\) such that

1. the context-free grammar \((M, X, Q, K)\) is unambiguous, and
2. for every \(w \in (V \cup X)^*\) such that \((w)\) is a basic hypernotion \(K \Rightarrow * Form(w)\). The metanotion \(K\) is called the main metanotion of the grammar.

The transparency condition is similar to the first of Sintzoff’s [22] structuring conditions. Since unambiguity of context-free grammars is undecidable, some sufficient condition for unambiguity of the metagrammar may be used. For example, one may require that the metagrammar is an LR(1) grammar.

A version of the transparency condition already appeared in the literature as a suggestion for ‘sugaring’ notation of extended affix grammars [23, p. 118]. The difference between both formulations mainly concerns the notation for variables. In our case, due to the ‘suffixing’ convention, the variables carry the information about their domains, while in the case of EAGs this information is expressed separately by so-called control of a given EAG. This shows a close relationship between both types of grammars: transforming a given EAG into a ‘sugared’ form and renaming its variables according to the suffixing convention one can construct a transparent two-level grammar generating the same language. Since every recursively enumerable set can be generated by an EAG, we obtain at once the following.

**Proposition 4.2.** Every recursively enumerable set can be generated by a transparent two-level grammar.

**Example 4.3.** We specify a transparent ‘version’ of the nontransparent grammar of Example 2.4:

**Hyperrules:**

\[
\begin{align*}
(1) & \quad (\text{bit string of length } N1i \text{ with value } N2) \triangleq \text{bit string of length } N1 \text{ with value } N3 \text{ if } N2 \text{ is doubled } N3 \\
(2) & \quad (\text{bit string of length } N1i \text{ with value } N2i) \triangleq \text{bit string of length } N1 \text{ with value } N3 \text{ if } N2 \text{ is doubled } N3 \\
(3) & \quad (\text{bit string of length with value}) \triangleq e \\
(4) & \quad (\text{is doubled}) \triangleq e \\
(5) & \quad (N1i \text{ is doubled } N2i) \triangleq (N1 \text{ is doubled } N2)
\end{align*}
\]
Metarules:

\[ K ::= \text{bit string of length } N \text{ with value } N | N \text{ is doubled } N \]
\[ N ::= \text{Nil} | E \]

Notice that the metanotion \( K \) does not occur in the basic hypernotions; it has been introduced only to fulfill the transparency condition.

For transparent two-level grammars the following theorem holds.

**Theorem 4.4.** For every pair of hypernotions of a transparent two-level grammar:

1. it is decidable, whether the hypernotions are unifiable, and
2. if they are unifiable, then there exists a most general unifier of these hypernotions

A proof of a version of this theorem can be found in [11], where a grammatical unification algorithm is given and proved to be correct. The rest of this section outlines another proof of the theorem, which can be found in [12]. A correspondence between the hypernotions of a transparent two-level grammar and a class of usual terms is established, which makes it possible to reduce the grammatical unification problem to usual unification. The reader is assumed to be familiar with the concept of a most general unifier of terms (e.g., from [18]).

Let \( G = (M, X, Q) \) be the metagrammar of a transparent grammar, let \( K \) be its main metanotion and let \( h_1 \) and \( h_2 \) be hypernotions. Thus, \( h_1 = (g_1) \) and \( h_2 = (g_2) \), for some \( g_1 \) and \( g_2 \) such that \( K_{\mathcal{M} \Rightarrow} \ast \text{Form}(g_1) \) and \( K_{\mathcal{M} \Rightarrow} \ast \text{Form}(g_2) \). Since \( G \) is unambiguous, the derivation trees of \( \text{Form}(g_1) \) and those of \( \text{Form}(g_2) \) are unique and describe the structure of the hypernotions. To be more precise, we shall represent the structure of \( g_1 \) and \( g_2 \) by terms. The terms will be constructed from functors, which are (names of) the production rules of \( Q \) and from the variables of \( V \).

We first define the arities of the functors.

For a production rule \( q \) of the form

\[ A ::= x_0 / A_1 \cdots A_n x_n \]

where \( n \geq 0 \), \( A_i \in M \) for \( i = 1, \ldots, n \) and \( x_j \in A^* \) for \( j = 0, \ldots, n \), the arity of \( q \) is defined to be \( n \).

A class \( C \) of the terms is defined as usual:

1. Each variable in \( V \) is a term.
2. If \( q \) is a functor of arity \( n \) and \( c_1, \ldots, c_n \) are terms, then \( q(c_1, \ldots, c_n) \) is a term.
3. Nothing else is a term.

The use of the production rules of a context-free grammar as functors bears some resemblance to the construction in [1] but the many-sorted approach is avoided in order to make usual term unification algorithms directly applicable to the defined terms. However, to relate hypernotions and terms some subclasses of the terms are distinguished, which are, as a matter of fact, the carriers of a many-sorted term algebra.
Towards a programming language based on two-level grammars

For $A \in M$ denote by $C_A$ the class of terms of type $A$ defined as follows:

1. Every variable $v$ such that $\text{Type}(v) = A$ is a term of the type $A$.
2. If $q$ is (a name of) a production rule in $Q$ of the form

$$A \equiv x_1A_1 \cdots A_nx_n$$

where $n \geq 0$, $A_i \in M$ for $i = 1, \ldots, n$ and $x_i \in X^*$ for $j = 0, \ldots, n$, and, for $i = 1, \ldots, n$, $c_i$ is a term of type $A_i$, then $q(c_1, \ldots, c_n)$ is a term of type $A$.

3. Nothing else is a term of type $A$.

It can be proved by considering Robinson's unification algorithm [18], that the image of arbitrary unifiable terms $c_1$ and $c_2$ of type $A$ under their most general unifier is a term of type $A$.

We now establish a correspondence between terms of type $K$ and hypernotions of the transparent grammar $W$. For this a function $\text{Rep}$ is defined, transforming terms into strings in $(V \cup X)^*$:

1. For each variable $v$: $\text{Rep}(v) = v$.
2. For a term $c = q(c_1, \ldots, c_n)$ such that $q$ is (a name of) a production rule of the form

$$A \equiv x_1A_1 \cdots A_nx_n$$

where $n \geq 0$, $A_i \in M$ for $i = 1, \ldots, n$ and $x_i \in X^*$ for $j = 0, \ldots, n$,

$$\text{Rep}(c) = x_n \text{Rep}(c_1) \cdots \text{Rep}(c_n) x_n$$

For each $A \in M$ denote by $\text{Rep}_A$ the restriction of the function $\text{Rep}$ to $C_A$. Since $G$ is unambiguous, $\text{Rep}_A$ is a one-one mapping onto the subset of $(V \cup X)^*$ consisting of all strings $x$ such that $A \Rightarrow^* \text{Form}(x)$. In particular, $\text{Rep}_K$ ranges over the set of the hypernotions with erased angle brackets. The inverse function to $\text{Rep}_A$ will be denoted $\text{Parse}_A$.

Theorem 4.4 directly follows from the following lemmas, whose proofs are omitted.

**Lemma 4.5.** If a hyperreplacement $\Theta$ is a grammatical unifier of hypernotions $\langle g_1 \rangle$ and $\langle g_2 \rangle$, then the function $\Theta': V \to C$ such that for each variable $v$: $\Theta'(v) = \text{Parse}_K(\Theta(v))$ is a unifier of the terms $\text{Parse}_K(g_1)$ and $\text{Parse}_K(g_2)$.

**Lemma 4.6.** If $\Theta$ is a most general unifier of terms $c_1$ and $c_2$ in $C_K$, then the string homomorphism $\Theta'$ on $(V \cup X \cup T \cup (,))^*$ such that:

1. For every $x \in (X \cup T \cup (,))$: $\Theta'(x) = x$, and
2. For every $v \in V$: $\Theta'(v) = \text{Rep}(\Theta(v))$

is a most general grammatical unifier of the hypernotions $\langle \text{Rep}(c_1) \rangle$ and $\langle \text{Rep}(c_2) \rangle$.

Thus, to check whether given hypernotions $\langle g_1 \rangle$ and $\langle g_2 \rangle$ are unifiable it suffices to 'parse' them, i.e., to construct the terms $\text{Parse}_K(g_1)$ and $\text{Parse}_K(g_2)$, and to apply to these terms a term unification algorithm. If the algorithm fails, the hypernotions
are not unifiable. Otherwise, the most general unifier of the terms constructed by the algorithm can be transformed into a most general grammatical unifier of the hypernotions.

5. Transparent two-level grammars and Horn clauses

In this section we discuss a relationship between transparent two-level grammars and Horn clauses. The terminology concerning Horn clauses is used as in the paper by Apt and van Emden [2].

Let \( Z \) be a finite definite sentence and let \( C \) be a negative clause consisting of a single atom of the form \( p(t_1, \ldots, t_n) \), where \( p \) is a predicate symbol of arity \( n, n \geq 0 \), and \( t_1, \ldots, t_n \) are terms. It is assumed that the predicate symbol \( p \) as well as the functors occurring in the terms also occur in the clauses of \( Z \). Consider the set \( M_{Z,C} \) of all variable-free instances of \( C \) which are in the least model of \( Z \) and denote by \( R_{Z,C} \) the \( n \)-ary relation on the Herbrand universe of \( Z \) consisting of all \( n \)-tuples \( (t_1, \ldots, t_n) \) such that \( p(t_1, \ldots, t_n) \) is in \( M_{Z,C} \). To relate Horn clauses and transparent two-level grammars we give a straightforward construction which for a given finite definite sentence \( Z \) and an atom \( C \) results in a transparent two-level grammar \( W_{Z,C} \) such that \( R_{Z,C} = Rel(W_{Z,C}) \). Roughly speaking, we give a 'grammatical interpretation' of \( Z \) and \( C \) considering the atoms occurring in the clauses to be the basic hypernotions, the atom \( C \) to be the start hypernotion and the clauses to be the basic hyperrules. To make it possible it is necessary to construct a metagrammar describing the structure of atoms and terms.

We now give a formal definition of the grammar \( W_{Z,C} \). First we construct the metagrammar. There are only two metanotions: \( A \) which is the main metanotion of \( W_{Z,C} \) and \( T \). The set of metarules is defined as follows:

1. If \( p \) is a predicate symbol of arity \( 0 \) occurring in the clauses of \( Z \), then
   \[
   A ::= p
   \]
is a metarule of \( W_{Z,C} \).

2. If \( p \) is a predicate symbol of arity \( n, n \geq 0 \), occurring in the clauses of \( Z \), then
   \[
   A ::= p(T, \ldots, T)
   \]
   (where the metanotion \( T \) occurs \( n \) times) is a metarule of \( W_{Z,C} \).

3. If \( c \) is a constant (nullary functor) occurring in the clauses of \( Z \), then
   \[
   T ::= c
   \]
is a metarule of \( W_{Z,C} \).

4. If \( f \) is an \( n \)-ary functor, \( n \geq 0 \), occurring in the clauses of \( Z \), then
   \[
   T ::= f(T, \ldots, T)
   \]
   (where the metanotion \( T \) occurs \( n \) times) is a metarule of \( W_{Z,C} \).
(5) There are no other metarules.

Observe that the set of all metanotion-free strings derivable from the metanotion $T$ is the Herbrand universe of $Z$ while the language generated by the metarules is the Herbrand base of $Z$. (The idea of using context-free grammars for describing sets of terms and substitutions already appeared in [17].)

Now we construct the basic hypernotions of $W_{ZC}$. For this we rename the variables occurring in the clauses of $Z$ and in $C$. According to the convention of Example 2.1 variables are to be represented by the metanotion $T$ with decimal suffixes. Denote by $Z'$ and $C'$ respectively the sentence and the atom obtained by renaming of variables necessary to meet this convention. Clearly, $R_{Z'C'} = R_{ZC}$. The basic hypernotions of $W_{ZC}$ are the atoms occurring in the clauses of $Z'$ and the atom $C'$, each of them enclosed in the angle brackets $\langle$ and $\rangle$ to meet the notational conventions of two-level grammars. The start hypernotion of $W_{ZC}$ is $\langle C' \rangle$.

The basic hyperrules of $W_{ZC}$ are constructed from the clauses of $Z'$: if $B \leftarrow B_1, \ldots, B_n$ is a clause of $Z'$ (where $B, B_1, \ldots, B_n$ are atoms), then $\langle B \rangle := \langle B_1 \rangle \cdots \langle B_n \rangle$ is a basic hyperrule and there are no other basic hyperrules. Since no terminal symbols appear in the hyperrules, the language generated by $W_{ZC}$ is either the singleton consisting of the empty string, or it is the empty set.

Example 5.1. To illustrate the construction we rewrite as a transparent two-level grammar the classical example of a clausal program for appending lists:

Clauses:

\[
\begin{align*}
(C_1) \quad \text{Append}(\text{nil}, X, X) & : \leftarrow \\
(C_2) \quad \text{Append}(l(X, U), V, l(X, W)) & : \leftarrow \text{Append}(U, V, W)
\end{align*}
\]

where $l$ and $\text{nil}$ are functors and $X, U, V, W$ are variables.

This can be rewritten in the two-level grammar notation as follows:

Hyperrules:

\[
\begin{align*}
(t_1) \quad \langle \text{append}(\text{nil}, T_1, T_1) \rangle & := \varepsilon \\
(t_2) \quad \langle \text{append}(l(T_1, T_2), T_3, l(T_1, T_4)) \rangle & := \langle \text{append}(T_2, T_3, T_4) \rangle
\end{align*}
\]

The metarules defining the structure of the atoms occurring in the clauses and the Herbrand universe are

\[
A := \text{append}(T_1, T_1) \quad \text{and} \quad T := \text{nil} | l(T_1, T_2).
\]

From the construction of $W_{ZC}$, we have the following results.

Proposition 5.2. If $\langle B_1 \rangle, \langle B_2 \rangle, \cdots \langle B_{2n} \rangle, \ldots, \langle B_k \rangle, \cdots \langle B_{kn} \rangle$, $\varepsilon$ is a derivation of $W_{ZC}$, where each $\langle B_i \rangle$ is a nonterminal of the grammar, then

\[
B_1, B_2, \cdots, B_{2n}, \ldots, B_k, \cdots B_{kn}, \varepsilon
\]

is an SLD-refutation of $Z \cup \{C\}$.\]
**Proposition 5.3.** If $B_{11}, B_{21} \cdots B_{2n}, \ldots, B_{k1} \cdots B_{k_n}, \square$ is an SLD-refutation of $Z \cup \{ \mathcal{C} \}$, where $B_{11}$ is a variable-free atom and each other $B_{ij}$ is an atom, then there exists a derivation of $W_{Z\mathcal{C}}$.

$$\langle B_{11}, \langle B'_{11}\rangle \cdots \langle B'_{2n}\rangle, \ldots, \langle B'_{k1}\rangle \cdots \langle B'_{k_n}\rangle, \tau$$

such that each $B'_{ij}$ is a variable-free instance of $B_{ij}$.

Since the relation $R_{Z\mathcal{C}}$ is defined in terms of the least model of $Z$ and SLD-resolution is complete, we obtain, by the above propositions, the following.

**Theorem 5.4**

$$R_{Z\mathcal{C}} = \text{Rel}(W_{Z\mathcal{C}}).$$

This theorem shows that transparent two-level grammars may be considered as a generalization of the formalism of Horn clauses in the following aspects:

- A notion of language is defined for transparent two-level grammars but not for Horn clauses. Horn clauses can, however, be used to define languages as described, e.g., in [16], where the formalism of definite clause grammars is discussed. But according to this paper 'a rule of a DCG is no more than syntactic sugar for a certain kind of definite clause' while a hyperrule of a transparent two-level grammar is a description of a set of context-free production rules.

- The syntax of Horn clause atoms is standard while the syntax of the hyperrules of a two-level grammar is defined by the metagrammar.

- The grammatical variables are typed in contrast to the variables occurring in Horn clauses. The typing mechanism of transparent two-level grammars is connected with the use of a metagrammar which can be viewed as a specification of a many-sorted term algebra. In contrast to the one-sorted term algebra defined by the function of a given set of Horn clauses, the types correspond to the nonterminals of the metagrammar which play the role of the sorts of the many-sorted term algebra.

Clearly, both formalisms have the same generative power, allowing one to specify every recursively enumerable set. However, from the pragmatic point of view the features of transparent two-level grammars pointed out above may have some advantages connected with the type discipline introduced by means of the user-defined metagrammar. Some examples illustrating this statement can be found in [13] and in [14].

6. Computing relations specified by transparent grammars

This section outlines a method for constructing relations specified by transparent two-level grammars. The method is based on grammatical unification: left-most extensions of the start hyperrule of the grammar are constructed in a systematic
way. The resulting standard hyperderivations determine subsets of the relation specified by the grammar. For presentation of the method some auxiliary concepts are introduced.

We define first a notion of the relation specified by an instance of the start hypernotion of the grammar. Let the start hypernotion $S$ of a transparent two-level grammar $W$ be of the form $(x_0, v_1, \ldots, v_n)$ where $x_0, \ldots, x_n$ are strings over the orthovocabulary $X$ and $v_1, \ldots, v_n$ are grammatical variables. Thus, $\text{Rel}(W)$ is an $n$-ary relation for some $n \geq 0$. Since the metagrammar is unambiguous and $\text{Form}(S)$ is derivable from its main metanotion, then for every instance $S'$ of $S$ there exists a unique $n$-tuple $(w_1, \ldots, w_n)$ of strings in $(V \cup X)^*$ such that $S' = (x_0, w_1, \ldots, w_n)$ and $\text{Type}(v_i) \Rightarrow^* \text{Form}(w_i)$ for $i = 1, \ldots, n$. We denote by $\text{Rel}(S')$ the $n$-ary relation on $X^*$ consisting of all $n$-tuples $(z_1, \ldots, z_n)$ such that $(x_0, z_1, \ldots, z_n)$ is a ground instance of $S'$. Notice that the relation $\text{Rel}(S')$, possibly infinite, can be represented by the finite tuple $(v_1, \ldots, v_n)$. If $S''$ is an instance of $S$ which can be obtained from $S'$ by a renaming hyperreplacement, then $\text{Rel}(S'') = \text{Rel}(S')$.

The relation determined by the first element of a standard hyperderivation $d$ is called the associated relation of $d$ and is denoted by $\text{Rel}(d)$. It follows from the definition of standard hyperderivation that its associated relation is a subset of the relation specified by the grammar. In the sequel a class of standard hyperderivations will be constructed such that the associated relations cover the relation specified by the grammar.

Let $d = (d_0, \ldots, d_n)$, $n \geq 0$, be a left-most hyperderivation such that $d_0$ is an instance of the start hypernotion of the grammar. For each $i = 0, \ldots, n - 1$ there exist $t_i \in T^*$, $w_i \in (H' \cup T)^*$ and a hyperrule $(h, z)$ such that $d_i = t_i h w_i$ and $d_{i+1} = t_i z w_i$. Any sequence $p_0, \ldots, p_{n-1}$ of basic hyperrules such that, for $i = 0, \ldots, n - 1$, the hyperrule $(h_i, z_i)$ is an instance of $p_i$ will be called a control sequence of the hyperderivation $d$. If $d$ is a one-element hyperderivation, its only control sequence is the empty sequence of basic hyperrules. A sequence of basic hyperrules is called a trace iff it is a control sequence of some standard hyperderivation. A trace $p$ is said to be a trace of a terminal string $x$ iff there exists a standard hyperderivation ending with $x$ whose control sequence is $p$.

For a sequence $p$ of basic hyperrules denote by $Hs(p)$ the set of all left-most hyperderivations $d$ such that

1. the first element of $d$ is an instance of the start hypernotion, and
2. $p$ is a control sequence of $d$.

**Proposition 6.1.** For arbitrary $p \in R^*$:

1. it is decidable whether $Hs(p)$ is empty or not, and
2. if $Hs(p)$ is not empty, then there exists a hyperderivation $d$ such that $Hs(p)$ consists of all instances of $d$.

**Proof.** Let $p = (p_0, \ldots, p_n)$, where $p_i \in R$ for $i = 0, \ldots, n$ and $n \geq 0$. We construct a sequence $d_0, \ldots, d_m$ of extensions of the start hypernotion, such that $m < n + 1$.
and if \( m < n + 1 \), then \( H_s(p) \) is empty. otherwise \( H_s(p) \) is the set of all instances of the hyperderivation \( d_{n+1} \).

The sequence is constructed as follows:

**Step 1.** \( d_n \) is the one-element sequence consisting of the start hypernotion of the grammar.

**Step 2.** For \( 0 \leq k \leq n \), if the last element of the hyperderivation \( d_k \) is a terminal string, then \( m = k \). Otherwise let \( d'_k \) be an instance of \( d_k \) obtained by a renaming hyperreplacement such that it has no common variables with the hyperrule \( p_k \). In this case the last element of \( d_k \) is of the form \( u_k h_k z_k \) for some \( h_k \in H' \), \( u_k \in T^* \) and \( z_k \in (H' \cup T)^* \). Let \( p_k = (g_k, w_k) \). If the hypernotions \( h_k \) and \( g_k \) are not unifiable, then \( m = k \). otherwise the hyperderivation \( d_{k+1} \) is defined to be the direct extension of \( d_k \) obtained by a most general unifier \( \Theta \) of \( h_k \) and \( g_k \) (i.e., it is the hyperderivation \( \Theta(d_k) \) with the attached last element \( \Theta(u_k)\Theta(w_k)\Theta(z_k) \)). It can be proved by induction that, for \( 0 \leq i \leq m \), the set \( H_s((p_0, \ldots, p_{i-1})) \) consists of all instances of the hyperderivation \( d_i \). If \( m < n + 1 \), then the hyperderivation \( d_m \) has no direct left-most extension, which could be obtained by the application of the hyperrule \( p_m \) and \( H_s(p) \) is empty.

The construction described above is nondeterministic: The resulting hyperderivation is determined up to renaming of the variables but the nondeterminism does not influence the relation associated with the hyperderivation. In the sequel it is assumed that some deterministic version of the construction is given which, for a given control sequence \( p \) of the grammar, produces a unique hyperderivation denoted \( hyp(p) \). Under this assumption the relation specified by a transparent two-level grammar \( W \) can be characterized as follows:

\[
Rel(W) = \bigcup \{ Rel(hyp(p)) : p \text{ is a trace of } W \}.
\]

The relations associated with the traces of \( W \) can be computed in a systematic way by the construction described above. Having computed the hyperderivation corresponding to some control sequence one can construct all its direct extensions. Those of them whose last elements are terminal strings determine subsets of the relation specified by the grammar. For the others the same construction should be applied recursively. The initial control sequence is empty and its corresponding hyperderivation has only one element: the start hypernotion \( S \). The computation may or may not terminate. The algorithm is described below in an Algol 68-like notation.

**Algorithm 6.2**

begin
  \textbf{relation} \( Z := 0; \)
  \textbf{proc} \texttt{dext} \( = \) \texttt{(hyperderivation \( d \), hyperrule \( r \)) \texttt{hyperderivation}}: \( \ldots \)
  \texttt{c the procedure attempts to construct a direct left-most extension of the}
hyperderivation \(d\) by application of the hyperrule \(r\), as described in the proof of Proposition 6.1. If it succeeds, the value returned is the extension of \(d\). Otherwise, if the left-hand side of \(r\) is not unifiable with the left-most hypernotion of the last element of \(d\), the value returned is the hyperderivation \(d\).

```
proc step = (hyperderivation d, hyperrule r) void:
begin
hyperderivation x = dext(d, r);
if x ≠ d
then
  if last(x) \(∈\) \(T^*\)
  then \(Z := Z \cup Rel(x)\)
  else for all \(p \in R\) do step(x, p) od
fi
fi
end;
for all \(p \in R\) do step((S), p) od
```

If the computational process terminates, the value assigned to \(Z\) is the relation specified by the two-level grammar; in this case the relation can be represented by a finite number of instances of the start hypernotion \(S\).

For implementation purposes the nondeterministic construct \(\text{for all } p\) should be replaced by some realistic control procedure. For example, one can use a sequential PROLOG-like control based on the textual ordering of the hyperrules in the specification of the grammar. In this case the control procedure should organize necessary backtracking.

### 7. Transparent grammars as programs

#### 7.1. The characteristic relation of a two-level grammar

In this section the language and the relation specified by a two-level grammar are handled jointly as one relation called the characteristic relation of the grammar. This notion is introduced with the intention to describe input/output relations of computational problems as characteristic relations of transparent two-level grammars. If such a specification of a computational problem is provided, the output values corresponding to given input values can be computed by an algorithm based on grammatical unification and the grammar may be considered to be a program for computing the output values from the input ones. Consequently, the class of transparent grammars becomes a programming language. The notion of characteristic relation provides a basis for the definition of its semantics while the algorithm is its interpreter. In the sequel we define the characteristic relation and we discuss some problems connected with the implementation of such a programming language.
Let \( W = (X, M, T, H, R, Q, S) \) be a transparent two-level grammar, and \( S = (x_0v_1 \cdots v_nx_n) \) where \( x_0, \ldots, x_n \) are strings over the orthovocabulary \( X \) and \( v_1, \ldots, v_n \) are grammatical variables (in this case \( \text{Rel}(W) \) is an \( n \)-ary relation). The relation \( C(W) \) is the \((n+1)\)-ary relation defined as follows:

\[
(z_1, \ldots, z_n, w) \in C(W) \text{ iff } (x_0z_1 \cdots z_nx_n) \text{ is a start nonterminal of the grammar,}
\]

\[
w \in T^* \text{ and } (x_0z_1 \cdots z_nx_n) \Rightarrow^* w.
\]

Thus, the elements of \( C(W) \) are constructed by attaching to every element \((z_1, \ldots, z_n)\) of the relation \( \text{Rel}(W) \) the nonterminal strings derivable from the start nonterminal \((x_0z_1 \cdots z_nx_n)\).

A standard hyperderivation \( d \) of \( W \) determines a subset of \( C(W) \), denoted \( C(d) \). This subset is defined as follows. Let \( d = (d_0, \ldots, d_k) \) for some \( k > 0 \). Thus, \( d_i = (x_0w_1 \cdots w_nx_n) \) where \( w_1, \ldots, w_n \) are strings in \((V \cup X)^*\) such that \( \text{Type}(r,x_0) \Rightarrow^* \text{Form}(w_i) \) for \( i = 1, \ldots, k \). The relation \( C(d) \) consists of all \((n+1)\)-tuples \((z_1, \ldots, z_{n+1})\) such that

1. \( z_{n+1} = d_k \), and
2. \( z_i \in X^* \) for \( i = 1, \ldots, n \), and the \( n \)-tuple \((z_1, \ldots, z_n)\) is an image of the \( n \)-tuple \((w_1, \ldots, w_n)\) under some hyperreplacement.

Assume that some \( m < n + 1 \) positions of the elements of \( C(W) \) have been distinguished as the input positions. We denote these positions by \( k_1, \ldots, k_m \), where \( 1 \leq k_i \leq n + 1 \) and \( k_i < k_{i+1} \) for \( i = 1, \ldots, m \). The problem of computing the output values for given input values can be formulated as follows:

For a given \( m \)-tuple \((w_1, \ldots, w_m)\) of strings, where, for \( i < m - 1 \), \( w_i \in X^* \), and \( w_m \in T^* \) if \( k_m = n + 1 \) and \( w_m \in X^* \) otherwise, find all \((n-m+1)\)-tuples \((z_1, \ldots, z_{n-1})\) of strings such that

\[
(z_1, \ldots, z_k, 1, w_1, z_{k+1}, 1, \ldots, z_{n-1}, w_{n-m-1}, z_{n-1}) \in C(W).
\]

**Example 7.1.** For the grammar of Example 4.3, \( n = 2 \) and the characteristic relation consists of triples \((z_1, z_2, z_3)\) such that

- \( z_1 \) is an arbitrary sequence of the symbols '0' and '1';
- \( z_2 \) is the sequence of \( c \) symbols '1', where \( c \) is the nonnegative integer, whose binary representation is \( z_1 \);
- \( z_3 \) is the sequence of \( l \) symbols \( i \), where \( l \) is the length of \( z_2 \).

Examples of the elements of the characteristic relation are the triples \((u, i, 01)\) and \((iii, iiiiiii, 111)\).

If we distinguish the \( k \)'th \((k = 1, 2, 3)\) position of the triples as the input position, the corresponding computational problem can be formulated as follows:

For \( k = 1 \): For a given nonnegative integer \( l \) construct all bit strings of length \( l \) and for each of them compute the nonnegative integer represented by this string in the binary notation. Notice that this set of pairs is finite.

For \( k = 2 \): For a given nonnegative integer \( c \) construct all its binary representations and with each of them associate the length of the representation. This set of pairs is infinite since the leading zeros are not suppressed.
For \( k = 3 \): For a given bit string \( s \) compute all pairs \((I, c)\) of nonnegative integers such that \( I \) is the length of \( s \) and \( s \) is a binary representation of \( c \). Clearly, this set is a singleton.

### 7.2. Computing the output values for given input values

For a transparent two-level grammar with distinguished inputs and outputs the output values corresponding to given input values can be obtained by computing a subset of the characteristic relation. In the sequel we outline an algorithm for computing such subsets. It is a version of Algorithm 6.2.

Denote by \( I \) the instance of the start hypernotion obtained by replacing the input variables \( v_k \) by given input values. Assume that \( k_m < n + 1 \) (this means that the terminal strings are not used as input values). In this case the subset of the characteristic relation corresponding to the input values can be computed by the following algorithm.

**Algorithm 7.2**

**begin**

relation \( C := \emptyset \);

 proc \( dext = (\text{hyperderivation } d, \text{hyperrule } r)\text{hyperderivation} \):
    \( c \), the procedure is defined as described in Section 6

 proc \( step = (\text{hyperderivation } d, \text{hyperrule } r)\text{ void} \):
    begin hyperderivation \( x = dext(d, r) \):
        if \( x > d \)
        then
            if \( \text{last}(x) \in T^* \)
            then \( C := C \cup C(x) \)
            else for all \( p \in R \) do \( step(x, p) \) od
        fi
    fi end;

    for all \( p \in R \) do \( step(l, p) \) od

**end**

The case when \( k_m = n + 1 \) (i.e., the case when one of the input values is a terminal string) can be handled in a similar way. In this case, for a given input terminal string \( w \), a hyperderivation \( x \) constructed by the procedure \( step \) contributes to the construction of the resulting relation only if its last element is \( w \). Thus, the condition "if \( \text{last}(x) \in T^* \)" in the body of the procedure should be replaced by the condition "if \( \text{last}(x) = w \)". In the next section another algorithm is proposed, which makes it possible to avoid constructing those standard hyperderivations whose last element is not \( w \).

Notice that the input positions and the output positions of the characteristic relation are determined by the instance \( I \) of the start hypernotion, and possibly by
supplying the string \(w\). Thus, the same transparent grammar can be used for computing different input/output relations as illustrated by Example 5.1. Such an interchanging of inputs and outputs is allowed also in Horn clause logic programs but in the other papers concerning relations specified by two level-grammars the inputs and the outputs are assumed to be fixed. The system of Chastellier and Colmerauer [3] works with pairs of two-level grammars and consists of two parts called the analyzer and the synthesizer. The analyzer uses as input data terminal strings of the language generated by the first grammar and computes for a given terminal string the corresponding elements of the relation specified by the grammar. The synthesizer uses these elements as input data and produces for them the corresponding strings of the language of the second grammar. In [22] it is assumed that input data are terminal strings for which the corresponding subsets of the relation specified by the grammar are computed and in [10] terminal strings are considered as output data for given ground instances of hypernotions, representing elements of the relation specified by the grammar.

7.3. Language-controlled computations

If the terminal strings are used as input values the problem of computing the output values is closely related to the parsing problem; for a given terminal string the corresponding subset of the characteristic relation is nonempty only if the string belongs to the language generated by the grammar. In this case it is possible to restrict a priori the set of the standard hyperderivations to be constructed during the computation. For this purpose we shall use an idea which resembles to certain extent Wegner’s approach to the parsing problem [24]. For a given two-level grammar \(W\) we construct a context-free grammar \(G_A\) with the same terminal alphabet and a mapping \(f\) from the basic hyperrules of \(W\) onto the production rules of \(G_A\). The grammar \(G_A\) is called the pattern grammar of \(W\). The function \(f\) relates standard hyperderivations of \(W\) and the left-most derivations of \(G_A\). The relation \(f^{-1}\) can be used to reconstruct standard hyperderivations of a given terminal string from its left-most derivation in the pattern grammar. The latter can be constructed by a context-free parsing algorithm.

To define formally the notion of pattern grammar we introduce some auxiliary notions. Let \(W = (X, M, T, H, R, Q, S)\) be a transparent two-level grammar. Denote by \(H_I\) the set of all left-hand sides of the basic hyperrules and by \(H_R\) the subset of \(H\) consisting of all hypernotions occurring in the right-hand sides of the basic hyperrules and of the start hypernotion \(S\).

Let \(\equiv\) be the binary relation on \(H\) defined as follows: \(h \equiv h'\) iff \(h\) and \(h'\) are unifiable and either \(h \in H_I\) and \(h' \in H_R\) or \(h \in H_R\) and \(h' \in H_I\). (Referring to the terminology of the ALCOR 68 Report we could call \(\equiv\) the cross-reference relation.)

The transitive closure of \(\equiv\) is an equivalence relation on \(H\): the equivalence class of \(h \in H\) will be denoted \([h]\).
The pattern grammar $G_w = (N_w, T_w, P_w, S_w)$ is defined as follows:

$N_w = \{[h]; h \in H\}$,

$T_w = T$,

$P_w = \{([h], t_0[h_1] \cdots [h_n]t_n); t_0, \ldots, t_n \in T^* (h, t_0h_1 \cdots h_nt_n) \in R\}$,

$S_w = [S]$.

Notice, that the production rules of $G_w$ are constructed from the basic hyperrules of $W$ by replacing the occurrences of the hypernotations by the corresponding equivalence classes. In that way for each $r \in R$ we obtain exactly one $p \in P_w$, which will be denoted $f(r)$. The mapping $f: R \rightarrow P_w$, need not be one-one, but, for each $p \in P_w$, $f^{-1}(p)$ is a finite set (since $R$ is finite). The function $f$ can be extended to a homomorphism transforming sequences of basic hyperrules of $W$ into sequences of production rules of $G_w$. We show that this homomorphism can be used to reconstruct hyperderivations of $W$ from derivations of its pattern grammar.

A sequence $t = (t_0, \ldots, t_n)$, where $n \geq 0$, of the production rules of a context-free grammar $G$ is called a control sequence of $G$ iff there exists a left-most derivation $(y_m, \ldots, y_1)$ in $G$ such that $y_n$ is the start nonterminal of $G$ and, for $i = 0, \ldots, n$, $y_{i+1}$ can be derived from $y_i$ by the production rule $t_i$. If $t$ is a control sequence, the derivation is unique and will be denoted $\text{Der}(t)$. Any control sequence $t$ of $G$ such that the last element of $\text{Der}(t)$ is a terminal string $z$ is called a trace of $z$ in $G$.

**Theorem 7.3.** For any $z \in T^*$ if $t$ is a trace of $z$ in a transparent two-level grammar $W$, then $f(t)$ is a trace of $z$ in the pattern grammar $G_w$.

**Proof.** Let $t$ be a control sequence of $W$, let $\text{hyp}(t) = (y_n, \ldots, y_1)$ and let $y_n = z_0h_1 \cdots h_mz_m$, where $z_i \in I^*$ for $i = 0, \ldots, m$ and $h_1, \ldots, h_m$ are hypernotations. We prove by induction on the length of $t$ that

1. $f(t)$ is a control sequence of the pattern grammar, and
2. There exist basic hypernotations $g_1, \ldots, g_m$ with the following properties:
   1. $z_0[g_1] \cdots [g_m]z_m$ is the last element of the derivation $\text{Der}(f(t))$, and
   2. $h_i$ is an instance of $g_i$ for $i = 1, \ldots, m$.

The theorem is a special case of the above statement for $m = 0$.

For $t = 0$, $t$ is the empty sequence of the basic hypernotations. Thus $f(t)$ is the empty sequence of the production rules of the pattern grammar and it is its control sequence. The corresponding derivation $\text{Der}(f(t)) = S_w = [S]$ and (2) holds since $\text{hyp}(t) = S$. In this case $m = 1$, $h_1 = S = g_1$.

Assume that (1) and (2) hold for each control sequence of length $n$ and consider a control sequence of length $n + 1$: $t = (r_0 \cdots r_n)$, where $r_i \in R$ for $i = 0, \ldots, n$. Let:

1. $r_k = (r_0, w_0h_1 \cdots h_kw_k)$, where $k \geq 0$, $w_i \in T^*$ for $i = 0, \ldots, k$ and $b, h_1, \ldots, h_k$ are basic hypernotations, and
2. $\text{hyp}(r_0 \cdots r_{n+1}) = (y_n, \ldots, y_1)$ and $y_n = z_0h_1 \cdots h_mz_m$, where $z_i \in T^*$ for $j = 0, \ldots, m$ and $h_1, \ldots, h_m$ are hypernotations.
We first prove (2). By the inductive hypothesis there exist basic hypernotions $q_1,\ldots,q_n$ such that $h_i$ is an instance of $q_j$ for $j = 1,\ldots,m$ and $z_0[q_1]\cdots[q_m]z_m$ is the last element of the derivation $Der((f(r_1,\ldots,f(r_n)))$). The last element of the hyperderivation $hyp(t)$ is of the form $z_0^t w'_i h'_i \cdots h'_k w_k z_1 h'_{k+1} \cdots h'_{k+m-1} z_m$ where $h'_i$ is an instance of $h_i$ for $i = 1,\ldots,k$ and it is an instance of $h_{i-k+1}$ for $i = k+1,\ldots,k+m-1$. Hence, for each $i = 1,\ldots,k+m-1$ there exists a basic hypernotion $g'_i$ such that $h'_i$ is an instance of $g'_i$. For $i = 1,\ldots,k$, $g'_i = h_i$, and for $i = k+1,\ldots,k+m-1$, we have $g'_i = g_{i-k+1}$.

To prove that $f(t)$ is a control sequence of $G_w$ it suffices to show that $z_0^t w'_i g'_i \cdots g'_k w_k z_1 g'_{k+1} \cdots g'_{k+m-1} z_m$ can be directly derived from $z_0(g_1)\cdots(g_m)z_m$ by the production rule $([b], w_i[b_1]\cdots[b_k]w_k)$. But $b$ and $g_1$ are unifiable since the instance $h_1$ of $g_1$ is unifiable with $b$—by the assumption that the hyperrule $r_n$ can be used to construct the direct extension of the hyperderivation $hyp(t_0\cdots t_{n-1})$. Thus, $[b] = [g_1]$ and (1) holds by the inductive hypothesis.

The above theorem can be used for parsing terminal strings in the transparent grammar $W$. For a given string $x \in T^*$ one can try to construct its left-most derivation in the pattern grammar. If such a derivation does not exist, $x$ does not belong to $I(W)$. Otherwise, the derivation determines a trace $t$ of $x$ in $G_w$ and one can check whether among the elements of the (finite) set $f^{-1}(t)$ there exists a trace $t'$ of $x$ in $W$. If this is the case, $hyp(t')$ is a standard hyperderivation whose last element is $x$. Thus $x \in I(W)$ and the first element of $hyp(t')$ (which is an instance of the start hypernotion) determines a set of output values corresponding to the input value $x$. Otherwise, if none of the elements of $f^{-1}(t)$ is a trace of $x$ in $W$, one has to check existence of another left-most derivation of $x$ in $G_w$. In the general case the number of such derivations may be infinite and the computation may not terminate. Notice that to compute all output values corresponding to $x$ one has to construct all traces of $x$ in $W$. Thus, even if the output values corresponding to some traces are found the computation may not terminate.

**Example 7.4.** We construct the pattern grammar of the two-level grammar of Example 4.3. Using the grammatical unification algorithm one can check that the equivalence classes of the basic hypernotions are the following:

$B = \{\langle\text{bit string of length } N0\text{ with value } N0\rangle,\$

$\langle\text{bit string of length } N1\text{ with value } N2\rangle,\$

$\langle\text{bit string of length } N1\text{ with value } N3\rangle,\$

$\langle\text{bit string of length with value}\rangle\}.$

$I = \{N2 \text{ is doubled } N3\}, \langle N1 \text{ is doubled } N2\rangle, \langle N1ii \text{ is doubled } N2\rangle, \langle N1i0 \text{ is doubled } N2\rangle, \langle N1i00 \text{ is doubled } N2\rangle.$

The resulting pattern grammar has the following production rules:

$$(1) \quad B ::= B \cdot 0 D.$$  
$$(2) \quad B ::= D.$$  
$$(3) \quad p ::= p.$$  
$$(4) \quad p ::= D.$$  
$$(5) \quad D ::= D.$$
Towards a programming language based on two-level grammars

The start nonterminal of the pattern grammar is $B$. The language specified by the grammar is the same as the language of the original two-level grammar and the function $f$ is a one-one correspondence between the basic hyperrules and the production rules of the pattern grammar.

For a given terminal string $x$ we want to use the pattern grammar to construct the traces of $x$ in the original two-level grammar. Let $x = 010$. The set of all traces of $x$ in the pattern grammar is the regular language

$$p_1p_2p_3p_4p_5p_6p_7p_8p_9.$$ Thus, by Theorem 7.3, if there exists a trace of $x$ in the two-level grammar it must belong to the language

$$r_1r_2r_3r_4r_5r_6r_7r_8r_9.$$ The prefix $r_1r_2r_3r_4$ gives the following hyperderivation determined up to renaming of the variables:

1. $\langle \text{bit string of length } 3 \text{ with value } N4 \rangle$
2. $\langle \text{bit string of length } 2 \text{ with value } N5i \rangle 0 \langle N4 \text{ is doubled } N5i \rangle$
3. $\langle \text{bit string of length } 1 \text{ with value } N6 \rangle 1 \langle N5 \text{ is doubled } N6 \rangle 0 \langle N4 \text{ is doubled } N5i \rangle$
4. $010 \langle N6 \text{ is doubled } N5 \text{ is doubled } N4 \rangle 0 \langle N4 \text{ is doubled } N5i \rangle$

To construct further extensions of the hyperderivation we have to find appropriate sequences of the hyperrules in the set $r_1^*r_2^*r_3^*r_4^*r_9$. The hypernotion $\langle N6 \text{ is doubled } \rangle$ is not unifiable with the left-hand side of $r_9$ but it matches $r_4$. As result we get the unifier $[r/N5, r/N1, r/N2]$. Continuing the check we discover that the only trace of 010 in the two-level grammar is $r_1r_2r_3r_4^*r_5r_9^*r_4$, which gives $\langle \text{bit strings of length } 1 \text{ with value } ii \rangle$ as the resulting instance of the start hypernotion; the output values for the input 010 are $(iii, ii)$.

It can be proved that the set of all traces of a terminal string in the pattern grammar is a regular language. It may be infinite only if the grammar is 'cyclic', i.e., if there exists a nonterminal symbol which can be derived from itself. In the example pattern grammar it was the nonterminal $D$. It is well known that any context-free grammar can be transformed into a non-cyclic one generating the same language (see, e.g., [7]). A regular expression characterizing the set of the traces of a given string in the original grammar can be reconstructed from the finite set of its traces in the transformed grammar. To obtain the latter some standard parsing technique might be used.

8. Conclusion

To handle the problem of computing the relation specified by a two-level grammar we extended the usual concept of derivation by allowing use of arbitrary instances
of the basic hyperrules, not only the ground ones. This idea leads to the problem of grammatical unification which turns out to be undecidable. This means that in the general case our approach to computing relations specified by two-level grammars cannot be taken. But this also means that in the general case it may be impossible to check existence of the 'cross-references' between the basic hyperrules of a two-level grammar, which are very important for human understanding of the grammar (for example reading of the ALGOL 68 Report without cross-references would be practically impossible). We believe that two-level grammars of practical importance are only those for which the grammatical unification problem is decidable.

Our solution to this problem is the transparency condition, which makes every hypernote of a transparent two-level grammar into a representation of a tree structure. In that case the unification problem is decidable and reduces to the usual term unification. This makes it possible to compute the relation specified by a transparent two-level grammar. On the other hand, this enables one to view transparent two-level grammars as a generalization of the Horn clause calculus based on the resolution principle. The main features of the extension are the following.

A free syntactic form of the hypernote

The hypernote may be defined by an arbitrary unambiguous metagrammar in contrast to the standard syntax of atoms of Horn clauses. This may contribute to the readability of the specification of a computational problem since the form of the hypernote may better express intuitions than the standard form of atoms. Moreover, the data representations used in the application area can be described by the metagrammar and used directly in the specification.

The typing mechanism

The metagrammar introduces a typing mechanism not present in the formalism of Horn clauses. This is due to the fact that it defines a many-sorted term algebra, as mentioned in Section 4, in contrast to the usual (one-sorted) term algebra defined by the functors of a set of Horn clauses.

The language

In contrast to a set of Horn clauses a transparent two-level grammar generates a language. It has a clear structure which can be described by the context-free pattern grammar. On the other hand, with each string of this language a subset of the relation specified by the grammar is associated, which can be considered as the 'meaning' of the string. Thus a transparent grammar can be viewed as a uniform definition of the syntax and of the semantics of the language. In particular, the language may be a conventional programming language and the relation specified
Towards a programming language based on two-level grammars

by the grammar may specify its semantics, as suggested by Hesse [6], who shows, that different types of semantic definitions can be expressed by two-level grammars. More recently it was suggested by Moss [15] to use definite clause grammars for describing the full syntax and semantics of programming languages. Such a definition can be compiled into Prolog and may be used as a prototyping tool for a new language. In view of our results this method can be seen as a restriction on the class of two-level grammars which makes it possible to apply the ideas of Hesse for prototyping. If the grammar describing the language is transparent, the relation associated with the string can be computed, though for certain strings the computation may never terminate. It might be interesting to find a condition for transparent grammars under which such computations terminate for every terminal string. This would be another decidability criterion for two-level grammars, different from those proposed by Deussen [5].

The transparency condition is a technical realization of the idea that the hypernotations of a two-level grammar should have a tree structure. This idea also appears in the definition of affix grammars [8] and in Watt’s thesis on extended affix grammars, but in both cases the aim of the restriction is to create a specialized tool for compiler writers, while our intention is to use two-level grammars as a universal programming language. The transparency condition resembles also the restriction of Simonet [20], who requires that the hypernotations are ramifications, i.e., explicit tree structures. However, the transparency condition is a restriction imposed on the class of two-level grammars, while an RW-grammar, strictly speaking, is not a two-level grammar, though it can be easily transformed into an equivalent two-level grammar. Simonet [20] uses the idea of tree-structured metanotions to compile RW-grammars into Horn clauses, while in our approach the resolution technique is applied directly to two-level grammars. Moreover, by dealing explicitly with the notion of the language generated by a transparent two-level grammar we are able to control computations by means of pattern derivations. In this way one can combine the resolution technique with the technique of context-free parsing. In certain cases this allows one to avoid backtracking which would be necessary after compilation of the grammar into a set of Horn clauses (an example can be found in [14]).

We have shown that transparent two-level grammars can be used for specifying and for computing relations. A programming language based on this formalism might be developed as an extension of Prolog, including the latter language as its proper subset. Some preliminary considerations concerning that matter can be found in [14]. It should be noted that both ‘levels’ of a transparent grammar are characterized by context-free grammars. The pattern grammar describes the structure of the strings of the language and the metagrammar describes the structure of the hypernotations. This enables one to use context-free parsing techniques both for finding the control constraints for computations and for compiling the basic hypernotations into the form required by the unification algorithm.
Acknowledgment

I am very grateful to Jorgen Fischer Nilsson of the Technical University of Denmark for many discussions during the preparation of the first draft of this paper. I gratefully acknowledge the comments and the suggestions which influenced my work, especially those received from Andrzej J. Blikle, Berthold Hoffmann, Gunter Riedewald, Michel Simonet, Michel Sintzoff and from the referees.

References

Towards a programming language based on two-level grammars


