Introduction

A pure logic program has a declarative reading and a procedural reading. This paper discusses the idea that these can be complemented by a grammatical reading, where the clauses are considered to be rewrite rules of a grammar. The objective is to show that this point of view facilitates transfer of expertise from logic programming to other research on programming languages and vice versa. Some examples of such transfer are discussed. On the other hand the grammatical view presented justifies some ad hoc extensions to pure logic programming and facilitates development of theoretical foundations for such extensions.

The logical notions of the declarative reading are to be related to their direct counterparts in the grammatical reading. As discussed in our early papers the grammar turns out to be a very special case of W-grammar (van Wijngaarden's two-level grammar). This opens for immediate extensions of the notion of logic program by introducing more general classes of W-grammars than these consisting of logic programs. One of them is the class of definite clause grammars. In our presentation the notion of DCG has a direct grammatical reading, in contrast to its usual understanding as a "syntactic sugar" for a special type of logic program. The construction of this program is now seen as a compilation of DCG.

It is well known that van Wijngaarden grammars are closely related to Knuth attribute grammars (AG's) which have been extensively studied and found many applications. Our previous work relating attribute grammars and logic programs uses as the unifying concept a common notion of decorated tree. This paper summarizes briefly this point of view and relates proof methods for logic programs with proof methods of attribute grammars; new proof methods for run-time properties and completeness of logics programs are presented.

1. Proof trees of definite clause programs.

This section relates the grammatical notion of derivation tree with the notion of proof tree of DCP. It shows that the semantics of logic programs can be expressed in the grammatical terms of proof trees.
One of the important concepts related to grammars is the notion of derivation tree. Let G be a context-free grammar and \( x \) a symbol of its alphabet. Then the notion of a derivation tree of G and \( x \) can be defined as follows.

**Definition 1.1.**

A *derivation tree* of G and \( z \) is any labeled ordered tree satisfying the following conditions:

1. Its root is labeled by \( z \),
2. If a node labeled \( x \) has \( n \) immediate descendants labeled \( x_1, \ldots, x_n \) (where the indices correspond to the ordering) then each \( x_i \) is either in the alphabet of G or \( n = 1 \) and \( x_1 \) is the empty string, and \( x \rightarrow x_1 \ldots x_n \) is a production rule of G.

A *complete derivation tree (a parse tree)* is a derivation tree such that none of its leaf nodes is labeled by a nonterminal of the grammar.

**Example 1.1**

Let G be the following collection of the production rules written in the BNF notation:

\[
\begin{align*}
<triple> & \rightarrow <aseq><bseq><cseq> \\
<aseq> & \rightarrow a \mid a <aseq> \\
<bseq> & \rightarrow b \mid b <bseq> \\
<cseq> & \rightarrow c \mid c <cseq>
\end{align*}
\]

A parse tree of G and \( <triple> \) is shown in Fig. 1.

A derivation tree can be seen as a result of pasting together "copies" of the production rules of the grammar. The parse trees of a grammar describe its language and give (possibly ambiguous) structure to every string in the language. Thus a grammar can be considered a specification of a class of parse trees rather than a specification of the language, which is a secondary concept defined in terms of parse trees.