A logic for one-pass, one-attributed grammars

by

A.J.J.M. Marcelis

90/7

July, 1990
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
All rights reserved
Editors: prof.dr.M.Rem
         prof.dr.K.M. van Hee
A logic for one-pass, one-attributed grammars

A.J.J.M. Marcelis

Department of Mathematics and Computing science
Eindhoven University of Technology
P.O.Box 513, 5600 MB Eindhoven
The Netherlands
e-mail: wsinmar@entws1.wie.tue.nl

April 1990

Abstract
A proof system for one-pass grammars is presented as an extension of a very general logic, with elements from typed λ-calculus and natural deduction. In the formulae of the logic, the emphasis is on contexts, which, at all times during a proof or derivation step, explicitly express the environment in which a step must take place. The proof method arrived at is compositional: to prove the correctness of a grammar (w.r.t. a specification), a proof per production rule suffices, where the contexts in which such a proof must be carried out ensure that local information is used only. The proof method is also reminiscent of the Hoare-style of proving programs.

Contents

1 Introduction ................................. 1
1.1 Motivation .................................. 1
1.2 Notational conventions .................... 2

2 Attribute grammars and correctness conditions ................................................. 2
2.1 One-attributed grammars ................. 2
2.2 Production trees ............................ 4
2.3 Translation mappings ..................... 5
2.4 Correctness condition ...................... 8

3 A logic for one-pass, one-attributed grammars ................................................. 9

4 Consistency of $TIS_B^+$ w.r.t. $TIS_B$ .......................... 10

5 Evaluation / Related work ................ 13

References ..................................... 14
1 Introduction

1.1 Motivation

In [C&D88] a particular method for proving the correctness of attribute grammars with respect to a specification has been presented. As the authors state, this method can be considered as an extension of the inductive assertions method [Flo67]. The inductive assertions method is one of the oldest forms of proving program correctness. It essentially amounts to labeling the nodes in a flowchart with assertions and showing that each branch respects these assertions. More recent methods for proving program correctness are usually based on Hoare's logic (see [Apt81] for a survey paper), which differs from the inductive assertions method in two essential aspects: first, it is a formal system with formulae and inference rules, and, second, proofs follow the syntactic structure of the program. Due to these properties, Hoare's logic lends itself better to the construction of correct programs than the inductive assertions method. The question arises whether a similar approach can be followed for attribute grammars, i.e., is it possible to design a logic for attribute grammars that can be considered as an analogon to Hoare's logic?

As a first step towards such a logic, this paper presents an inference system for one-pass, one-attributed grammars. The system is an extension of a typed inference system, as described in [Mar90].

More specifically, the line of reasoning pursued in this paper is the following.

In Section 2, we start off by introducing one-pass, one-attributed grammars. Here, "one-attributed" means that each nonterminal is supplied with exactly one inherited and one synthesized attribute domain (or type, as we shall call it), and "one-pass" denotes the well-known restriction on the evaluation rules, causing the attributes of each derivation tree \(t\) to be evaluatable in a single (left-to-right) pass over \(t\). The demand for "one-attributed"-ness is for notational convention only; it is not very restrictive, as tupling can always be used to simulate multi-attributes by one attribute.

Through one of its constituent components, an attribute grammar — according to our definition — has an associated typed inference system, denoted \(TIS_B\), and the well-formedness of the grammar is expressed via the derivability of certain formulae within \(TIS_B\). The latter turns out to be an important stepping-stone for the embedding of \(ag\) in formal logic.

Also in Section 2, we develop the notion of the correctness of an attribute grammar \(G'\) — with underlying context-free grammar \((N, T, P, Z)\) — w.r.t. a specification. To that end, we first introduce production trees (which are derivation trees labeled with context-free production rules), and we show how \(G'\) gives rise to a collection \(\{FA | A \in N\}\) of translation mappings, acting on the production trees. More specifically, \(FA\) maps production trees of type \(PT_A\) onto functions from \(itA\) to \(stA\) (where \(PT_A\) is the type of all production trees with root labeled by \(A\), and \(itA\) (\(stA\)) denotes the inherited (synthesized) attribute type of nonterminal \(A\), i.e., \(FA\) has signature \(PT_A \rightarrow itA \rightarrow stA\).

A specification for \(G'\) then consists of a pair of predicates \((Q, R)\) of signature \(Q : itZ \rightarrow bool\) and \(R : stZ \rightarrow bool\), and the correctness condition for \(G'\) is expressed as

\[
\forall d : PT_Z, \ i : itZ, (Q \cdot i \Rightarrow R \cdot i \cdot (FZ \cdot d \cdot i))
\]

i.e., for all complete production trees, and for all values \(i\) in the inherited domain of \(Z\) that satisfy \(Q \cdot i\), \(FZ\) applied to \(d\) and \(i\) satisfies \(R \cdot i \cdot (FZ \cdot d \cdot i)\).

If \(G'\) is an attribute grammar as in Section 2, with associated inference system \(TIS_B\), then in Section 3 we present a logic for \(G'\), denoted \(TIS_B^+\), as an extension of \(TIS_B\). More precisely, per production rule \(pr\) of \(G'\) there is an additional formula expressing 'pr is correct', and there is an additional formula expressing '\(G'\) is correct' (all in a context without notions concerning production trees or translation mappings). Furthermore, an additional inference rule expresses 'if all \(pr\) are correct, then \(G'\) is correct' (in ditto context).

In the following section, an interpretation \(I\) is provided from formulae in \(TIS_B^+\) onto formulae in \(TIS_B\). In particular, the interpretation of the formula '\(G'\) is correct' is the correctness condition
for \( G' \). We then show the consistency of \( TIS_{B}^{+} \) w.r.t. \( TIS_{B} \) under \( I \). This means that for each formula \( \phi \), derivable within \( TIS_{B}^{+} \), its interpretation \( I(\phi) \) is derivable within \( TIS_{B} \).

Thus we have achieved that, in order to prove the correctness condition for \( G' \) in \( TIS_{B} \), it suffices to infer ' \( G' \) is correct' within \( TIS_{B}^{+} \). The latter, in its turn, is accomplished by inferring 'pr is correct' for each production rule \( pr \); all without reference to production trees or translation mappings whatsoever.

Finally, in Section 5 we provide an evaluation of the method and formalism just described, and we state the relation with previous work (notably that of [C&D88]).

1.2 Notational conventions

All notations concerning typed inference systems are taken from [MargO]. In particular, it is useful to recall that the correctness condition displayed above is shorthand for

\[
\forall d : PTZ. (\forall i : itZ. (Q \cdot i \Rightarrow R \cdot i(PTZ \cdot d \cdot i)))
\]

and similar abbreviations apply to nested \( \lambda \)-abstractions and let-constructs. Also, \( \rightarrow \) (for denoting function types) associates to the right and \( \cdot \) (application) associates to the left.

For details concerning the "flag notation", which is used to express the proofs of Section 4, see Section 3 of [MargO].

If \( S \) is a set, then \( S^{*} \) denotes the set of all finite sequences over \( S \), and for a sequence \( \alpha \) and set \( S \), notation \( \alpha \upharpoonright S \) is used for the projection of \( \alpha \) onto \( S \).

2 Attribute grammars and correctness conditions

In Subsection 2.1 we define one-attributed grammars, and one-pass grammars as a special case of them. Such a grammar can be viewed as a context-free grammar (in the usual sense), extended with some restricted form of attribute structure.

Within the notational framework of typed inference systems, we then introduce some concepts related to (attribute) grammars, resulting in a notion of correctness for an attribute grammar w.r.t. a specification. More precisely, Subsection 2.2 defines production trees for a context-free grammar. An inference rule expressing structural induction over production trees is also given. Subsection 2.3 is concerned with the translation mappings induced by a one-pass, one-attributed grammar, and Subsection 2.4 states the correctness condition for such a grammar w.r.t. a specification.

2.1 One-attributed grammars

This subsection deals with one-attributed grammars, and one-pass grammars as a special case of them. The well-formedness of such grammars is expressed largely in terms of the derivability of formulae within a typed inference system. To that end, a typed boolean structure \( B \) (which forms the basis of such an inference system) appears in the definition of a one-attributed grammar. Another constituent is \( \Gamma \), a context over \( B \) (in the sense of definition 2.4 of [MargO]). \( \Gamma \) contains the "theory" of attribute-types involved. For instance, if, in a practical case, an attribute has as its domain the type "stack of integer", then \( \Gamma \) would contain the definition of this type, and an (axiomatic) definition of the operations on the type.

Definition 2.1 (one-attributed grammar)

A one-attributed grammar is a 8-tuple \( G' = (N, T, P', Z, it, st, B, \Gamma) \), where

- \( N \) is a finite set
- \( T \) is a finite set
- \( N \cap T = \emptyset \)
- \( Z \in N \)
\[ B = (C_t, V_t, C_e, V_e, L, TA) \] is a typed boolean structure

\[ \Gamma \] is a context over \( B \)

\[ it = \{ it_A \mid A \in N \} \]
\[ st = \{ st_A \mid A \in N \} \]

where, for all \( A \in N \), \( B \vdash_{TIS} \Gamma \vdash it_A : * \) and \( B \vdash_{TIS} \Gamma \vdash st_A : * \)

\( P' \) is a finite set of constructs, a typical element of which, \( p' \) say, reads

\[ A_0(i_0, s_0) \rightarrow w_0 A_1(i_1, s_1) w_1 \ldots w_{n-1} A_n(i_n, s_n) w_n \]
\[ s_0 = e_0, i_1 = e_1, \ldots, i_n = e_n \]

where

\[ n \geq 0 \]
\[ w_k \in T^* \], for all \( k : 0 \leq k \leq n \)
\[ \{ A_0, A_1, \ldots, A_n \} \subseteq N \]
\[ \{ i_0, s_0, \ldots, i_n, s_n \} \subseteq V_e \]
\( i_0, i_1, \ldots, i_n, s_n \) are pairwise different

\[ B \vdash_{TIS} \Gamma, i_0 : it_{A_0}, s_1 : st_{A_1}, \ldots, s_n : st_{A_n} \Rightarrow e_k : it_{A_k} \]

\[ B \vdash_{TIS} \Gamma, i_0 : it_{A_0}, s_1 : st_{A_1}, \ldots, s_n : st_{A_n} \Rightarrow e_0 : st_{A_0} \]

\( N \) and \( T \) are the sets of nonterminal and terminal symbols, respectively, and \( Z \) is the start symbol of the grammar.

For a nonterminal \( A \), \( it_A \) and \( st_A \) are the inherited and synthesized attribute types of \( A \), respectively.

\( P' \) is the set of attributed production rules. For a typical element \( p' \) of \( P' \), as specified above, \( i_k \) and \( s_k \) denote the inherited and synthesized attributes of nonterminal \( A_k \) (for \( 0 \leq k \leq n \)), \( s_0 = e_0, i_1 = e_1, \ldots, i_n = e_n \) are the evaluation rules (sometimes also referred to as semantic rules), and \( A_0 \rightarrow w_0 A_1 w_1 \ldots w_{n-1} A_n w_n \) is the underlying (context-free) production rule. If we let \( P \) denote the set of all underlying production rules, then \( (N, T, P, Z) \) is a context-free grammar, the underlying context-free grammar of \( G' \).

For attributed production rule \( p' \), exactly one evaluation rule occurs for each of the attributes \( s_0, i_1, \ldots, i_n \). Moreover, the restrictions on the type deduction for expressions \( \epsilon_k \) (\( 0 \leq k \leq n \)) imply that — as far as attribute variables are concerned — \( FEV(\epsilon_k) \subseteq \{ i_0, i_1, \ldots, s_n \} \). Thus, via the evaluation rules, attributes \( s_0, i_1, \ldots, i_n \) are expressed in terms of attributes \( i_0, s_1, \ldots, s_n \). These restrictions on the attributed production rules establish the usual normal form requirement for an attribute grammar (see [Boc76]).

This normal form supports the common view on inherited and synthesized attributes as carriers of downward ("input") and upward ("output") information, respectively, in any attributed derivation tree of the grammar. Namely, via the attributed production rules (that constitute such a tree) each inherited attribute in the tree is defined in terms of attributes in its "upper neighbourhood" and thus conveys information downward through the tree, while each synthesized attribute is defined in terms of its "lower neighbourhood" and as such provides for the upward flow of data. (See [Boc76] for details.) We shall re-encounter this nature of attributes when defining the translation mappings of an attribute grammar, later on in this section.

By limiting the occurrence of free variables in the expressions \( \epsilon_k \) (\( 1 \leq k \leq n \)) of definition 2.1 still further, a sub-class of the one-attributed grammars is obtained:
Definition 2.2 (one-pass condition)
A one-attributed grammar is called one-pass (left-to-right) if each element

\[ A_0(s_0, s_0) \rightarrow w_0 A_1(w_1, s_1) \ldots w_{n-1} A_n(w_n, s_n) w_n \]

of \( P' \) satisfies

\[ B \vdash_{TIS} \Gamma, i_0 : i_{A_0}, s_1 : s_{A_1}, \ldots, s_{k-1} : s_{A_{k-1}} \triangleright e_k : i_{A_k}, \text{ for all } k : 1 \leq k \leq n \]

(Hence, for all \( k : 1 \leq k \leq n \), \( FEV(e_k) \subseteq \{ i_0, s_1, \ldots, s_{k-1} \} \).

An attribute grammar \( G' \) satisfying the one-pass condition has the property that the attributes of a derivation tree \( t \), associated with \( G' \), can be evaluated in a single (left-to-right) pass over \( t \) (see e.g. [Eng84], [Boc76]).

### 2.2 Production trees

Throughout, derivation trees of a context-free grammar \( G = (N, T, P, Z) \) are assumed to be labeled with production rules (rather than grammar symbols), and will be called production trees for that reason. A production tree is fully determined by the label of its root and a sequence of direct subtrees; the latter in accordance with root label.

The following defines PT \( A \), for each nonterminal \( A \in N \), to be the type of the production trees issued from \( A \), i.e., the trees with root labeled with a production rule where left-hand side \( A \). \( PT_{A} \) is a sum type with an alternative for each such production rule (i.e., possible root label). The collection \( \{ PT_A \mid A \in N \} \) is defined with mutual recursion.

Definition 2.3 (production trees)
Let \( G = (N, T, P, Z) \) be a context-free grammar. The collection of types \( \{ PT_A \mid A \in N \} \) is defined with mutual recursion as

\[
\text{rec } ( \ldots , PT_A := \text{sum} ( \ldots , A_0 \rightarrow \alpha : \text{prod}(PT_{A_1}, \ldots , PT_{A_n}) , \ldots ) \\
\ldots )
\]

Herein, the rec-construct contains a clause per nonterminal. Above the clause is shown for nonterminal \( A_0 \). In its turn, the sum type defining \( PT_{A_0} \) contains an alternative per production rule with \( A_0 \) as its left-hand side. Above the alternative is displayed for rule \( A_0 \rightarrow \alpha \), with \( \alpha \in (N \cup T)^* \) such that \( \alpha \backslash N = A_1 \ldots A_n \).

An expression of type \( PT_A \) is called a production tree issued from \( A \). A complete production tree is a production tree issued from start symbol \( Z \).

Several matters should be noted.
In particular, a production tree issued from \( A_0 \) (i.e., an expression of type \( PT_{A_0} \)) is a construct \( \{ A_0 \rightarrow \alpha, \langle d_1, \ldots, d_n \rangle \} \), where \( A_0 \rightarrow \alpha \in P \) and \( \langle d_1, \ldots, d_n \rangle \) is a sequence of production trees, such that \( d_k \) is issued from \( A_k \), the \( k^{th} \) nonterminal in \( \alpha \), for \( 1 \leq k \leq n \).

The notation of concepts according to definition 2.3 leaves the relation to grammar \( G \) implicit; it will always be clear from the environment which grammar is meant. A similar remark applies to forthcoming definitions.

In addition to definition 2.3 it is possible to define the type, \( PT \) say, of the production trees of \( G \) (without further differentiation), namely

\[ PT = \text{sum}( \ldots , B_0 \rightarrow \beta : \text{prod}(PT_{B_1}, \ldots , PT_{B_m}) , \ldots ) \]

where the sum type contains an alternative per production rule in \( P \). Above the alternative for
rule $B_0 \rightarrow \beta$ is shown, with $\beta \vdash N = B_1 \ldots B_n$. Thus, the sum type defining $PT$ contains the collected alternatives of the definitions for $\{PT_A \mid A \in N\}$. However, for our purposes the use of the separate definitions for $PT_A$ will suffice, therefore the notion of $PT$ has not been included in definition 2.3.

Example 2.4 (on production trees)
Consider the context-free grammar $G = (N, T, P, Z)$, with

- $N = \{Z, Y\}$
- $T = \{z, y, x\}$
- $P = \{Z \rightarrow z, Z \rightarrow YYy, Y \rightarrow Zx\}$

The types of the production trees of $G$ are $PT_z$ and $PT_y$, defined with mutual recursion as follows

\[
\begin{align*}
\text{rec } (PT_z = & \sum (Z \rightarrow z : \text{prod}(\cdot), Z \rightarrow YYy : \text{prod}(PT_y, PT_y))) \\
, \text{PTy = } & \sum (Y \rightarrow Zz : \text{prod}(PT_z))
\end{align*}
\]

Hence, an expression of type $PT_z$ is of either of the forms $[Z \rightarrow z, ()]$ or $[Z \rightarrow YYy, (d_1, d_2)]$, where $d_1$ and $d_2$ are expressions of type $PT_y$. Likewise, an expression of type $PT_y$ is of the form $[Y \rightarrow Zz, (d)]$, with $d$ of type $PT_z$. An example of the latter is $[Y \rightarrow Z", ([Z \rightarrow z, ()])]$.

For an attribute grammar $G'$ with underlying context-free grammar $G$ we formulate an inference rule from $TIS_D$, expressing structural induction over the production trees of $G$. To that end, let $D$ be a context containing the appropriate recursive type definition, as in definition 2.3 above, and let $B \vdash_{TIS} D \Rightarrow R_A : PT_A \rightarrow \text{bool}$ (for all $A \in N$). Then the induction rule reads

\[
D \Rightarrow \forall A \rightarrow \alpha \in P, (d_1, \ldots, d_n) : \text{prod}(PT_{A_1}, \ldots, PT_{A_n}) \\
(R_{A^*\cdot d_1 \land \ldots \land R_{A^n}\cdot d_n) \Rightarrow R_A \cdot [A \rightarrow \alpha, (d_1, \ldots, d_n)])
\]

wherein $\alpha \in (N \cup T)^*$, with $\alpha \models N = A_1 \ldots A_n$.

2.3 Translation mappings
A one-pass, one-attributed grammar $G'$ gives rise to a collection $\{F_A \mid A \in N\}$ of translation mappings. Herein, $F_A$ has type $PT_A \rightarrow i_A \rightarrow st_A$, i.e., it maps production trees issued from $A$ onto functions from the inherited to the synthesized attribute domain of $A$. Stated differently, given a tree $d$ of type $PT_A$ and an expression ("value") $i$ of the inherited type $i_A$, $F_A \cdot d \cdot i$ yields a value of synthesized type $st_A$.

This way, an attribute grammar can be considered to realise, through $F_Z$, a translation from the language produced by the underlying context-free grammar — plus environment information, modelled by domain $i_Z$ — to some target language (represented by domain $st_Z$). More generally, each $F_A$ (A $\in N$) realises such a translation from the sublanguage produced by $A$ — plus $i_A$ — to $st_A$.

For $A \in N$, $F_A$ is defined to be a lambda-expression, the body of which consists of a case-construction, selecting among the possible forms of expressions of type $PT_A$.

Definition 2.5 (translation mappings)
Let $G' = (N, T, P', Z, i, st, B, \Gamma)$ be a one-pass, one-attributed grammar, with $(N, T, P, Z)$ as its underlying context-free grammar. The collection of expressions $\{F_A \mid A \in N\}$ is defined with mutual recursion as

\[
D \Rightarrow \forall A \rightarrow \alpha \in P, (d_1, \ldots, d_n) : \text{prod}(PT_{A_1}, \ldots, PT_{A_n}) \\
(R_{A^*\cdot d_1 \land \ldots \land R_{A^n}\cdot d_n) \Rightarrow R_A \cdot [A \rightarrow \alpha, (d_1, \ldots, d_n)])
\]
Herein, the rec-construct contains a clause per nonterminal. Above the clause is shown for nonterminal \( A_0 \). In its turn, the case-expression in the definition of \( F_{A_0} \) contains an alternative per kind of production tree of type \( PT_{A_0} \); there are as many of these kinds as there are production rules in \( P \) with left-hand side \( A_0 \) (such production rules act as root labels). Above the alternative is displayed for the kind of trees with \( A_0 \rightarrow \alpha \) as their root label, where \( \alpha \in \mathcal{N} = A_1, \ldots, A_n \). The direct subtrees \( d_1, \ldots, d_n \) are hence of types \( PT_{A_1}, \ldots, PT_{A_n} \), respectively. The above definition also expresses that \( H_{A_0} \) applied to a tree of the form \( [A_0 \rightarrow \alpha, \langle d_1, \ldots, d_n \rangle] \) yields \( H_{A_0 \rightarrow \alpha}(F_{A_0 \rightarrow \alpha}(d_1) \cdots (F_{A_0 \rightarrow \alpha}(d_n)) \). Herein, \( F_{A_1 \rightarrow \alpha} \) through \( F_{A_n \rightarrow \alpha} \) are the applications of the appropriate translation mappings to the direct subtrees of \( [A_0 \rightarrow \alpha, \langle d_1, \ldots, d_n \rangle] \), and \( H_{A_0 \rightarrow \alpha} \) is a higher-order function completely determined by the attributed production rule in \( P' \) that has \( A_0 \rightarrow \alpha \) as its underlying context-free rule (the structure of the \( H \)-functions will be dealt with hereafter).

\[ \square \]

**Example 2.6 (on translation mappings)**

Consider the one-pass, one-attributed grammar \( G' = (N, T', Z, \mathcal{I}, \mathcal{F}, \mathcal{G}, \mathcal{B}, \Gamma) \), of which only the following components will be specified:

- \( N = \{ Z, Y \} \)
- \( T = \{ z, y, x \} \)
- \( P' = \{ Z(i_0, s_0) \rightarrow z \}
\]

\[ s_0 = u_0 \]

\[ , Z(i_0, s_0) \rightarrow Y(i_1, s_1) Y(i_2, s_2) y \]

\[ s_0 = u_0 \times i_1 = v_1, i_2 = v_2 \]

\[ , Y(i_0, s_0) \rightarrow Z(i_1, s_1) x \]

\[ s_0 = u_0 \times i_1 = u_1 \]

Notice that grammar \( G \) of example 2.4 is the underlying context-free grammar of \( G' \).

The translation mappings \( F_Z \) and \( F_Y \) induced by \( G' \) are defined as mutually recursive expressions. Herein \( F_Z \) has type \( PT_Z \rightarrow \mathcal{I} T_Z \rightarrow \mathcal{I} T_Z \) and \( F_Y \) has type \( PT_Y \rightarrow \mathcal{I} T_Y \rightarrow \mathcal{I} T_Y \). The definition reads

\[
\text{rec (} F_Z = \varepsilon \lambda d : PT_Z . \text{ (case } d \text{ of) }
\]

\[
\begin{align*}
&\quad [Z \rightarrow z, \langle \rangle] \text{ then } H_{Z \rightarrow z},
\end{align*}
\]

\[
\begin{align*}
&\quad [Z \rightarrow YYy, \langle d_1, d_2 \rangle] \text{ then } H_{Z \rightarrow YYy}(F_Y \cdot d_1) \cdot (F_Y \cdot d_2)
\end{align*}
\]

\[
), F_Y = \varepsilon \lambda d : PT_Y . \text{ (case } d \text{ of) [ } Y \rightarrow Z \}, \langle d_1 \rangle \text{ then } H_{Y \rightarrow Z}(F_Z \cdot d_1)
\]

\]

wherein \( H_p \) (for \( p \in \{ Z \rightarrow z, Z \rightarrow YYy, Y \rightarrow Zx \} \)) is a higher-order function fully determined by the evaluation rules of the attributed production rule \( P' \) that has \( p \) as its underlying context-free rule.

\[ \square \]
Next we specify how an attributed production rule $p'$, with underlying context-free rule $p$, gives rise to a function $H_p$ as used in the definition of translation mappings.

As may be checked from the latter definition and the (required) type of $F_{A_0} \circ H_{A_0 \to \alpha}$, with $\alpha \mid N = A_1 \ldots A_n$, has type

$$(it_{A_1} \to st_{A_1}) \to \ldots \to (it_{A_n} \to st_{A_n}) \to (it_{A_0} \to st_{A_0})$$

We shall give the definition of $H_{A_0 \to \alpha}$ first, and go into its meaning (in connection with translation mappings) afterwards.

**Definition 2.7**

Let $G'$ be a one-pass, one-attributed grammar. An attributed production rule $p'$:

$$A_0(i_0, s_0) \to w_0 A_1(i_1, s_1)w_1 \ldots w_{n-1} A_n(i_n, s_n)w_n$$

$s_0 = e_0, i_1 = e_1, \ldots, i_n = e_n$ of $G'$ gives rise to $H_p$, where $p$ is the underlying context-free rule of $p'$, defined by

$$H_p = \varepsilon \lambda h_1 : (it_{A_1} \to st_{A_1}), \ldots, h_n : (it_{A_n} \to st_{A_n})$$

$$(\lambda i_0 : it_{A_0})$$

$$(\text{let } i_1 : it_{A_1} = e_1, \ldots, i_n : it_{A_n} = e_n)$$

$$(\text{let } s_1 : st_{A_1} = h_1 \cdot i_1, \ldots, s_n : st_{A_n} = h_n \cdot i_n)$$

$$(s_0 : st_{A_0} = e_0)$$

$$(\text{in } s_0)$$

The above presentation clearly shows how the evaluation rules $s_0 = e_0, i_1 = e_1, \ldots, i_n = e_n$ of $p'$ appear in the body of $H_p$ and, in fact, completely determine $H_p$.

A more concise denotation is obtained by substituting expressions $e_k$ for $i_k (1 \leq k \leq n)$ and $e_0$ for $s_0$, which yields

$$H_p = \varepsilon \lambda h_1 : (it_{A_1} \to st_{A_1}), \ldots, h_n : (it_{A_n} \to st_{A_n})$$

$$(\lambda i_0 : it_{A_0})$$

$$(\text{let } s_1 : st_{A_1} = h_1 \cdot e_1, \ldots, s_n : st_{A_n} = h_n \cdot e_n)$$

$$(s_0 : st_{A_0} = e_0)$$

Recall that the meaning ("value") of the let-construct in (*) equals that of $e_0$, with the proviso that variables $s_1, \ldots, s_n$ (that may occur free in $e_0$, cf. definition 2.1) are bound to $h_1 \cdot e_1, \ldots, h_n \cdot e_n$, respectively.

In fact, notice also that the order of bindings in the (nested) let-construct reflects the left-to-right nature of the evaluation rules of $p'$, in the following sense: according to definition 2.2, for each $k (1 \leq k \leq n)$ \( \text{FEV}(e_k) \subseteq \{i_0, s_1, \ldots, s_{k-1}\} \), of which $i_0$ is bound in the enclosing $\lambda$-abstraction, and $s_1, \ldots, s_{k-1}$ are bound "earlier" in the let-construct.

---

1 By the normal form requirement, a variable $i_k (1 \leq k \leq n)$ does not occur free in any of the expressions $e_i (1 \leq i \leq n)$, and neither does $s_0$. Therefore $i_k$ is used only in the subsequent formula $h_k \cdot i_k (s_0$ only in "... in $s_0$"), and (*) is a correct abbreviation.
Example 2.8 (on $H$-functions)

We give the functions $H_{z_{2}}$ and $H_{z_{yy}}$, determined by attributed production rules

$Z(i_{0}, s_{0}) \rightarrow z_{2}$

$Z(i_{0}, s_{0}) \rightarrow Y(i_{1}, s_{1}) Y(i_{2}, s_{2}) y$

of example 2.6 (and used in the translation mappings of that example).

$H_{z_{2}}$ is a function of type $it_{z} \rightarrow st_{z}$. Using the abbreviated notation (*), it is defined by

$H_{z_{2}}(i_{0}, s_{0}) = \lambda i_{0}: it_{z}. u_{0}$

$H_{z_{yy}}$ is a function of type $(ity \rightarrow st_{y}) \rightarrow (ity \rightarrow st_{y}) \rightarrow (it_{z} \rightarrow st_{z})$. Its definition reads

$H_{z_{yy}}(i_{0}, s_{0}) = \lambda h_{1} : (ity \rightarrow st_{y}), h_{2} : (ity \rightarrow st_{y})$

$(\lambda i_{0} : it_{z}) (\lambda s_{1} : st_{y} = h_{1} \cdot v_{1}, s_{2} : st_{y} = h_{2} \cdot v_{2} \text{ in } v_{0})$

Now consider the use of $H_{A_{0}} \cdot a$ in the definition of translation mapping $F_{A_{0}}$ ($A_{0} \rightarrow \alpha$ is the underlying context-free rule of $p'$ as in definition 2.7). According to definition 2.5, $F_{A_{0}}$ applied to $[ A_{0} \rightarrow \alpha, (d_{1}, \ldots, d_{n}) ]$ yields $H_{A_{0} \rightarrow \alpha}(F_{A_{1}} \cdot d_{1}), \ldots, (F_{A_{n}} \cdot d_{n})$. The latter reduces to (using (*)) for simplicity:

$\lambda i_{0} : it_{A_{0}}. (\text{let } s_{1} : st_{A_{1}} = F_{A_{1}} \cdot d_{1} \cdot e_{1},$

$\vdots$

$s_{n} : st_{A_{n}} = F_{A_{n}} \cdot d_{n} \cdot e_{n}$

$\text{in } e_{0})$

Thus, $H_{A_{0} \rightarrow \alpha}$ applied to $F_{A_{1}} \cdot d_{1}, \ldots, F_{A_{n}} \cdot d_{n}$ uses $e_{k}$ (associated with the inherited attribute of $A_{k}$ by the evaluation rules of $p'$) as argument ("input") for the application of translation mapping $F_{A_{k}}$ to the $k^{th}$ direct subtree $d_{k}$, and it incorporates the result of $F_{A_{k}} \cdot d_{k} \cdot e_{k}$ by binding it to $s_{k}$ (the synthesized attribute of $A_{k}$ in $p'$). The overall result is a function that, when applied to a value $i_{0}$ of type $it_{A_{0}}$, yields $e_{0}$ (with the appropriate bindings), which is associated with the synthesized attribute of $A_{0}$.

This way, $H_{A_{0} \rightarrow \alpha}$ reflects the nature of the inherited and synthesized attributes of $A_{0} \rightarrow \alpha$ (cf. the discussion following definition 2.1): the former act as input information for the application of translation mappings to (sub)trees, whereas the latter are identified with the results of these applications.

In fact, for this reason $H_{p}$ may well be conceived as the meaning of attributed production rule $p'$. Likewise, the collection of translation mappings $\{F_{A} \mid A \in N\}$ — determined by $\{H_{p} \mid p \in P\}$ according to definition 2.5 — may be considered as the meaning of attribute grammar $G'$. More precisely, the application of $F_{A}$ to a production tree $d$ of type $PT_{A}$ simulates attribute evaluation for $d$, considered as a function from $it_{A}$ to $st_{A}$. This way, an attribute grammar is identified with the translation mappings it induces; such a characterisation forms the basis for the relation between attribute grammars and functional programming, see for instance [Joh87].

2.4 Correctness condition

Finally we define the correctness condition for a grammar $G' = (N, T, P', Z, it, st, B, \Gamma)$ with respect to a specification. A specification for $G'$ is a pair $(Q, R)$ of predicates such that $B \vdash_{TIS} \Gamma \triangleright Q : it_{z} \rightarrow bool$ and $B \vdash_{TIS} \Gamma \triangleright R : it_{z} \rightarrow st_{z} \rightarrow bool$. 
Definition 2.9 (correctness condition)

Let \( G' = (N, T, P', Z, it, st, B, \Gamma) \) be a one-pass, one-attributed grammar. Let \( Q \) and \( R \) be predicates such that \( B \vdash_{TIS} \Gamma \triangleright Q : it_Z \rightarrow bool \) and \( B \vdash_{TIS} \Gamma \triangleright R : it_Z \rightarrow st_Z \rightarrow bool \), i.e., the pair \((Q, R)\) is a specification for \( G' \).

The correctness condition for \( G' \) w.r.t. \((Q, R)\) reads

\[
\forall d : PT_Z, i : it_Z \cdot (Q_i \Rightarrow R_i(F_Z \cdot d_i))
\]

wherein \( PT_Z \) is the type of the production trees issued from \( Z \) and \( F_Z \) is the translation mapping for \( Z \).

\( \Box \)

The correctness condition for \( G' \) expresses that for all complete production trees \( d \), and for all values \( i \) in the inherited domain of \( Z \) that satisfy \( Q_i \), \( F_Z \) applied to \( d \) and \( i \) satisfies \( R_i(F_Z \cdot d_i) \).

However, as the collection \( \{FA_I^A : A \in N\} \) is defined with mutual recursion over production trees, in order to prove the correctness condition for \( G' \) we have to prove similar conditions for incomplete production trees as well. This leads to the introduction of collections of predicates \( \{QA_I^A : A \in N\} \) and \( \{RA_I^A : A \in N\} \) of the appropriate types (with \( Q_Z = Q \) and \( R_Z = R \)) and a correctness condition for each nonterminal \( A \). The correctness condition for \( G' \) then equals the one for start symbol \( Z \).

Indeed, these additional correctness conditions will be encountered in connection with the logic to be developed next.

3 A logic for one-pass, one-attributed grammars

Let \( G' = (N, T, P', Z, it, st, B, \Gamma) \) be a one-pass, one-attributed grammar. We define a logic for \( G' \), denoted by \( TIS_B^\Gamma \), as an extension of \( TIS_B \). To save writing in this definition, assume that \( N = \{A_1, \ldots, A_n\} \), assume that \( QA_1, \ldots, QA_n \) and \( RA_1, \ldots, RA_n \) are such that

\[
B \vdash_{TIS} \Gamma \triangleright QA_k : it_{A_k} \rightarrow bool \quad (\text{for all } k : 1 \leq k \leq n)
\]

\[
B \vdash_{TIS} \Gamma \triangleright RA_k : it_{A_k} \rightarrow st_{A_k} \rightarrow bool \quad (\text{for all } k : 1 \leq k \leq n)
\]

and let \( \Gamma' \) denote the sequence

\[
q_{A_1} : it_{A_1} \rightarrow bool, \quad q_{A_1} = \varepsilon QA_1, \ldots, q_{A_n} : it_{A_n} \rightarrow bool, \quad q_{A_n} = \varepsilon QA_n,
\]

\[
r_{A_1} : it_{A_1} \rightarrow st_{A_1} \rightarrow bool, \quad r_{A_1} = \varepsilon RA_1, \ldots, r_{A_n} : it_{A_n} \rightarrow st_{A_n} \rightarrow bool, \quad r_{A_n} = \varepsilon RA_n
\]

i.e., \( \Gamma' \) serves to select a collection \( \{QA_k, RA_k : A_k \in N\} \) of predicates and to bind these predicates to the variables \( \{q_{A_k}, r_{A_k} : A_k \in N\} \). Notice that the predicates can be typed in a context only containing \( \Gamma \).

The logic \( TIS_B^\Gamma \) consists of inference formulae and rules as follows

Inference formulae are

1. the formulae of \( TIS_B \)
2. \( A \triangleright (pr \text{ correct}) \), where
   - \( pr \in P' \)
   - \( A \) is a context over \( B \)
3. \( A \triangleright (G', Q, R) \), where
   - \( B \vdash_{TIS} \Gamma \triangleright Q : it_Z \rightarrow bool \)
   - \( B \vdash_{TIS} \Gamma \triangleright R : it_Z \rightarrow st_Z \rightarrow bool \)
A is a context over B.

Inference rules are

1. the rules of \( TIS_B \)

2. For a production rule \( pr \in P' \) of the form

\[
A_0 \langle i_0, s_0 \rangle \rightarrow w_0 A_1 \langle i_1, s_1 \rangle w_1 \ldots w_{n-1} A_n \langle i_n, s_n \rangle w_n
\]

\( \sigma_0 = \epsilon_0 , \ i_1 = \epsilon_1 , \ldots , \ i_n = \epsilon_n \)

the rule

\[
\Gamma , \ \Gamma' , \ \ i_0 : \ \text{it}_{A_0} , \ \ s_0 : \ \text{st}_{A_0} , \ \ldots , \ i_n : \ \text{it}_{A_n} , \ \ s_n : \ \text{st}_{A_n} , \ \ s_0 = \epsilon \ e_0 , \ \ i_1 = \epsilon \ e_1 , \ \ldots , \ i_n = \epsilon \ e_n \\
\rightarrow \ \\
q_{A_0} \ i_0 ^ 0 \wedge \bigwedge _{j=1}^{n} (q_{A_j} \ i_j ^ j \wedge r_{A_j} \ i_j ^ j \ s_j ) \rightarrow q_{A_k} ^ k ^ 0 \text{ for all } k : 1 \leq k \leq n
\]

\[ \text{for all } k : 1 \leq k \leq n \]

\[
\Gamma , \ \Gamma' \ \\
\rightarrow \ (pr \ 	ext{correct})
\]

3. \( \Gamma , \ \Gamma' \ \\
\rightarrow \ (pr_1 \ 	ext{correct}) , \ldots , \ \Gamma , \ \Gamma' \ \\
\rightarrow \ (pr_m \ 	ext{correct})
\]

wherein \( P' = \{ pr_1 , \ldots , pr_m \} \), \( Q = Q_Z \), \( R = R_Z \)

Recall that \( \Gamma' \) selects a collection \( \{ Q_{A_k} , R_{A_k} \mid A_k \in N \} \) of predicates. Notice that in the last inference rule above the part \( \Gamma' \) is dropped from the context. This means that the information about the selected predicates is lost when applying this rule (except for \( Q_Z \) and \( R_Z \), concerning start symbol \( Z \), which are retained in the conclusion of the rule). On the other hand, if one is asked to derive \( (G', Q, R) \) — i.e., \( G' \) is correct w.r.t. \( Q \) and \( R \) — for some \( Q \) and \( R \), a reverse application of the last rule requires the selection ("invention") of predicates \( Q_{A_k} \) and \( R_{A_k} \) for each nonterminal \( A_k \), with the proviso that \( Q_Z = Q \) and \( R_Z = R \).

Derivability of an inference formula \( \varphi \) within \( TIS_B^+ \) is denoted by \( B \vdash TIS_B^+ \varphi \).

4 Consistency of \( TIS_B^+ \) w.r.t. \( TIS_B \)

We define the notion of consistency between two logics as follows

Definition 4.1 (Consistency)

Let \( L_1 = (F_1 , R_1) \) and \( L_2 = (F_2 , R_2) \) be two inference systems, with \( F_1 \supseteq F_2 \) and \( R_1 \supseteq R_2 \). Denote the derivability of a formula \( \varphi \) in \( L_1 \) and \( L_2 \) with \( \vdash_{L_1} \varphi \) and \( \vdash_{L_2} \varphi \), respectively. Let \( \mathcal{J} \) be an interpretation of the formulae in \( L_1 \) in terms of the formulae in \( L_2 \), i.e., \( \mathcal{J} \in F_1 \rightarrow F_2 \).

a. Rule \( \forall_{\varphi \in \mathcal{J} \in R_1} \) is consistent w.r.t. \( L_2 \) under \( \mathcal{J} \) if

\[ \vdash_{L_2} \mathcal{J}(\varphi_1) \text{ and } \ldots \text{ and } \vdash_{L_2} \mathcal{J}(\varphi_n) \text{ imply } \vdash_{L_2} \mathcal{J}(\varphi) . \]

b. \( L_1 \) is consistent w.r.t. \( L_2 \) under \( \mathcal{J} \) if for all \( \varphi \in \mathcal{F}_1 \)

\[ \vdash_{L_1} \varphi \text{ implies } \vdash_{L_2} \mathcal{J}(\varphi) \]

\[ \Box \]

Lemma 4.2

Let \( L_1 , L_2 \) and \( \mathcal{J} \) be as in the preceding definition. In order to prove that \( L_1 \) is consistent w.r.t. \( L_2 \) under \( \mathcal{J} \) it suffices to show that all rules of \( R_1 \) are consistent w.r.t. \( L_2 \) under \( \mathcal{J} \).

proof: straightforward, using the notion of derivability of formulae within an inference system. \( \Box \)
For $G' = (N, T, P', Z, it, st, B, \Gamma')$ we now provide an interpretation of the formulae in $TIS_B^+$ in terms of those in $TIS_B$, and we show the consistency of $TIS_B^+$ w.r.t. $TIS_B$ under this interpretation.

**Definition 4.3 (I, interpretation)**

Let $G', TIS_B, TIS_B^+$ and $\Gamma'$ be as in the preceding section. Let $\Delta$ be a piece of context containing

- the type definitions for $\{PT_A | A \in N\}$, the types of the production trees of $G$
- the definitions for $\{FA_A | A \in N\}$ and $\{H_{A \rightarrow \alpha} | A \rightarrow \alpha \in P\}$, concerning the translation mappings induced by $G'$

Finally, let $\Delta'$ be a piece of context containing a clause

$$\Phi_A : PT_A \rightarrow \text{bool}, \Phi_A = \lambda d : PT_A \cdot (\forall i : \text{it}_A . (q_A \cdot i = r_A \cdot i \cdot (F_A \cdot d \cdot i)))$$

for each $A \in N$. Notice that, this way, $\forall d : PT_A \cdot (\Phi_A \cdot d)$ expresses the correctness condition for $F_A$ w.r.t. $Q_A$ and $R_A$ (cf. subsection 2.4); where the latter two are bound to $q_A$ and $r_A$, respectively, by $\Gamma'$.

Then $I$ maps formulae of $TIS_B^+$ onto formulae of $TIS_B$ according to

1. $I( D \triangleright \varphi ) = D \triangleright \varphi$, for $D \triangleright \varphi$ an inference formula of $TIS_B$

2. $I( \Gamma, \Gamma' \triangleright (pr \text{ correct}) ) = \Gamma, \Gamma', \Delta, \Delta' \triangleright \forall (d_1, \ldots, d_n) : \text{prod}(PT_{A_1}, \ldots, PT_{A_n}) \cdot (\Phi_{A_0} \cdot d_1 \wedge \ldots \wedge \Phi_{A_n} \cdot d_n \Rightarrow \Phi_{A_0} \cdot [A_0 \rightarrow \alpha, (d_1, \ldots, d_n)])$

wherein $A_0 \rightarrow \alpha$, with $\alpha = w_0 A_1 w_1 \ldots w_{n-1} A_n w_n$, is the underlying production rule of $pr$.

3. $I( \Gamma' \triangleright (G', Q, R) ) = \Gamma, \Delta \triangleright \forall d : PT_Z, i : \text{it}_Z . (Q \cdot i \Rightarrow R \cdot i \cdot (F_Z \cdot d \cdot i))$

Notice that the interpretation of $\Gamma' \triangleright (G', Q, R)$ displays the correctness condition for $G'$ (subsection 2.4), in a context containing the appropriate definitions.

\[ \square \]

**Theorem 4.4**

Let $G', TIS_B, TIS_B^+, \Gamma', \Delta, \Delta'$ be as in definition 4.3, and let $I$ be as defined by the latter definition. Then $TIS_B^+$ is consistent w.r.t. $TIS_B$ under $I$.

**proof:**

Due to Lemma 4.2, this requires a proof per inference rule in $TIS_B^+$. These rules come in three kinds (cf. section 3). We provide a proof for each of the kinds.

**Rules of kind 1 (rules of $TIS_B$):** Obvious, as the rules of $TIS_B$ form a subset of those of $TIS_B^+$, and formulae appearing in these rules have identity interpretation.

**Rules of kind 2:** Consider attributed production rule $pr$ of the form

$$A_0(i_0, s_0) \rightarrow w_0 A_1(i_1, s_1) w_1 \ldots w_{n-1} A_n(i_n, s_n) w_n$$

$$s_0 = e_0, i_1 = e_1, \ldots, i_n = e_n$$

The premises of the corresponding inference rule are $(n+1)$ formulae in $TIS_B$ (hence, with identity interpretation). Assuming the derivability (in $TIS_B$) of these formulae, we provide a proof of $B \vdash_{TIS} I(\Gamma, \Gamma' \triangleright (pr \text{ correct}))$. Namely, starting from

11
repeated application of the rules for context extension and reordering of context terms, yields

1. \( \Gamma', \Gamma' \)

2. \( (d_1, \ldots, d_n) : \text{prod}(PT_{A_1}, \ldots, PT_{A_n}) \)

3. \( \Phi_{A_1} \cdot d_1 \land \ldots \land \Phi_{A_n} \cdot d_n \)  

4. \( t_0 : t_{A_0}, s_0 : s_{A_0}, \ldots, t_n : t_{A_n}, s_n : s_{A_n} \)

5. \( s_1 : s_{A_1}, s_1 = s_1 = e_1, i_1 \)

6. \( s_1 : s_{A_1}, s_1 = s_1 = e_1, i_1 \)

7. ...  

8. \( s_n : s_{A_n}, s_n = s_1 = e_n, \ldots, i_n \)

9. \( s_0 : s_{A_0}, s_0 = e_0 \)

10. \( q_{A_0} \cdot i_0 \land \bigwedge_{j=1}^{k-1} (q_{A_j} \cdot i_j \land t_{A_j} \cdot i_j \cdot s_j) \Rightarrow q_{A_k} \cdot i_k \), for all \( k : 1 \leq k \leq n \)

11. \( q_{A_0} \cdot i_0 \land \bigwedge_{j=1}^{n} (q_{A_j} \cdot i_j \land t_{A_j} \cdot i_j \cdot s_j) \Rightarrow r_{A_0} \cdot i_0 \cdot s_0 \)

and the proof continues as indicated:

12. \( q_{A_k} \cdot i_k \Rightarrow r_{A_k} \cdot i_k \cdot s_k \), for all \( k : 1 \leq k \leq n \) (el \( \forall, 3, 5-9 \))

13. \( q_{A_0} \cdot i_0 \) (4)

14. \( q_{A_1} \cdot i_1 \) (el \( \Rightarrow, 10, 13 \))

15. \( r_{A_1} \cdot i_1 \cdot s_1 \) (el \( \Rightarrow, 12, 14 \))

16. ...  

17. \( r_{A_n} \cdot i_n \cdot s_n \) (el \( \Rightarrow, 12, 16 \))

18. \( r_{A_0} \cdot i_0 \cdot s_0 \) (el \( \Rightarrow, 11, \ldots \))

19. \( r_{A_0} \cdot i_0 \cdot (\text{let } s_1 : s_{A_1} = s_1 : s_{A_1} = F_{A_1} \cdot d_1 \cdot i_1, \ldots, s_0 : s_{A_0} = s_0 \text{ in } s_0) \) (let-rule, repeatedly)

20. \( r_{A_0} \cdot i_0 \cdot (H_{A_0} \rightarrow \alpha \cdot (F_{A_1} \cdot d_1) \cdot \ldots \cdot (F_{A_n} \cdot d_n) \cdot i_0) \) (def. \( H_{A_0} \rightarrow \alpha \))

21. \( r_{A_0} \cdot i_0 \cdot (F_{A_0} \cdot [A_0 \rightarrow \alpha, (d_1, \ldots, d_n)] \cdot i_0) \) (def. \( F_{A_0} \))

22. \( \forall i_0 : t_{A_0}, (q_{A_0} \cdot i_0 \Rightarrow r_{A_0} \cdot i_0 \cdot (F_{A_0} \cdot [A_0 \rightarrow \alpha, (d_1, \ldots, d_n)] \cdot i_0)) \) (in \( \Rightarrow, \text{in } \forall, 4, 21 \))

23. \( \Phi_{A_0} \cdot [A_0 \rightarrow \alpha, (d_1, \ldots, d_n)] \) (def. \( \Phi \))

24. \( \forall (d_1, \ldots, d_n) : \text{prod}(PT_{A_1}, \ldots, PT_{A_n}) \cdot (\Phi_{A_1} \cdot d_1 \land \ldots \land \Phi_{A_n} \cdot d_n \Rightarrow \Phi_{A_0} \cdot [A_0 \rightarrow \alpha, (d_1, \ldots, d_n)]) \) (in \( \Rightarrow, \text{in } \forall, 2, 3, 23 \))
Line 24 (in its proper context, viz. line 1) displays $I(\Gamma, \Gamma', \triangleright (pr \ correct))$. Therefore rules of kind 2 are consistent w.r.t. $TIS_B$ under $I$.

Rules of kind 3:
Starting from $B \vdash_{TIS} I((pr_1 \ correct)), \ldots, B \vdash_{TIS} I((pr_m \ correct))$, using rules for reordering of contexts, and applying (in $\forall$), yields

1. $\begin{align*} & \Gamma, \Delta \end{align*}$
2. $\begin{align*} & \Gamma' \end{align*}$
3. $\begin{align*} & \Delta' \end{align*}$
4. $\begin{align*} & \forall A \rightarrow \alpha \in P, (d_1, \ldots, d_n) : \text{prod}(PT_{A_1}, \ldots, PT_{A_n}) \Rightarrow \Phi_{A_1 \cdot d_1} \land \ldots \land \Phi_{A_n \cdot d_n} \Rightarrow \Phi_A \cdot [A \rightarrow \alpha, (d_1, \ldots, d_n)] \end{align*}$

and, continuing:

5. $\begin{align*} & \forall A \in N, d : PT_A \cdot (\Phi_A \cdot d) \quad \text{(induction)} \end{align*}$
6. $\begin{align*} & \forall d : PT_Z, i : it_Z : (Q_Z \cdot i \Rightarrow R_Z \cdot i \cdot (F_Z \cdot d \cdot i)) \quad \text{(el $\forall, 5 \& \text{def. } \Phi_Z, q_Z, r_Z)} \end{align*}$
7. $\begin{align*} & \forall d : PT_Z, i : it_Z : (Q \cdot i \Rightarrow R \cdot i \cdot (F_Z \cdot d \cdot i)) \quad \text{(context reduction \& $Q_Z = Q, R_Z = R$)} \end{align*}$

Line 7 (in context 1) displays $I((G', Q, R))$. Hence rules of kind 3 are consistent.

With Lemma 4.2, the result now holds as claimed.

\( \Box \)

5 Evaluation / Related work

In this paper, we have presented a logic for one-pass, one attributed grammars, as an extension of a very general inference system based on \( \lambda \)-calculus and natural deduction. The proof method obtained is compositional. As a particular feature, contexts play a very important role in the inference formulae of the logic.

The merit of such a logic lies in the fact that it is a formal system with well-defined formulae and rules, and, thanks to the strong emphasis on contexts, the environment in which proofs and design steps must be carried out is defined very precisely. As an example of the latter phenomenon, the inference rules introduced in Section 3 express that the proof of the condition \( pr \ correct \) must take place in a context containing no information about production trees or translation mappings. Also, the last inference rule in the same section clearly exhibits the precise point where additional predicates \( Q_A \) and \( R_A \) must be selected for nonterminals \( A \), different from \( Z \).

As a consequence, we expect that an approach like the one presented in this paper will turn out to be well-suited for the derivation of attribute grammars from a specification.

For the scope of this paper we have restricted ourselves to one-pass grammars, but the method can be extended without too much difficulty to more general kinds of grammars; typically multi-pass or multi-sweep ones ([Fil83]). We feel that for the design of practical attribute grammars — the goal we ultimately strive for — attribute grammars with a more complicated attribute structure than the ones mentioned above are hardly likely to be of use.

As the most important work related to ours we mention that of Courcelle & Deransart [C&D88]. However, the latter paper is far more theoretically oriented, and, due to the application of the particular formalism, does not give too much hold where the “rules of the game” allowed are concerned.

13
The proof rule arrived at in our paper can be regarded as an instance of the annotations method in [C&D88], that is, with a suitable choice for the arcs in the graphs $D(p)$ — see section 4.3, p. 43 —, reflecting the "one-pass"-ness of the grammar.

In view of what was said in the preceding paragraph, it is in fact expected that in practical situations the choice of annotations (and the relations between them) will often be inspired by considerations of pass-orientedness of the attribute scheme under consideration.

Other related work is that of Katayama & Hoshino [K&H81], who follow a similar approach for the class of absolutely noncircular attribute grammars, but, again, in a less precisely defined framework.

Future work will be directed towards a further development of the method, for use in connection with more complicated types of grammars (typically multi-pass and multi-sweep ones). In addition, the use of the formalism as an aid in deriving correct attribute grammars will be investigated, especially in connection with code generation. This investigation may include the issue of transforming attribute grammars while preserving their correctness (w.r.t. a specification).

References


In this series appeared:

<table>
<thead>
<tr>
<th>No.</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>85/01</td>
<td>R.H. Mak</td>
<td>The formal specification and derivation of CMOS-circuits.</td>
</tr>
<tr>
<td>85/02</td>
<td>W.M.C.J. van Overveld</td>
<td>On arithmetic operations with M-out-of-N-codes.</td>
</tr>
<tr>
<td>85/03</td>
<td>W.J.M. Lemmens</td>
<td>Use of a computer for evaluation of flow films.</td>
</tr>
<tr>
<td>85/04</td>
<td>T. Verhoeff</td>
<td>Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate.</td>
</tr>
<tr>
<td>85/01</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems.</td>
</tr>
<tr>
<td>86/02</td>
<td>G.A. Bussing</td>
<td>ELISA, A language for formal specification of information systems.</td>
</tr>
<tr>
<td>86/03</td>
<td>Rob Hoogerwoord</td>
<td>Some reflections on the implementation of trace structures.</td>
</tr>
<tr>
<td>86/04</td>
<td>G.J. Houben</td>
<td>The partition of an information system in several systems.</td>
</tr>
<tr>
<td>86/05</td>
<td>R. Gerth</td>
<td>A framework for the conceptual modeling of discrete dynamic systems.</td>
</tr>
<tr>
<td>86/06</td>
<td>Tom Verhoeff</td>
<td>Nondeterminism and divergence created by concealment in CSP.</td>
</tr>
<tr>
<td>86/07</td>
<td>R. Gerth</td>
<td>On proving communication closedness of distributed layers.</td>
</tr>
<tr>
<td>86/08</td>
<td>R. Koymans</td>
<td>Compositional semantics for real-time distributed computing (Inf.&amp; Control 1987).</td>
</tr>
<tr>
<td>86/09</td>
<td>C. Huizing</td>
<td>Full abstraction of a real-time denotational semantics for an OCCAM-like language.</td>
</tr>
<tr>
<td>86/10</td>
<td>J. Hooman</td>
<td>A compositional proof theory for real-time distributed message passing.</td>
</tr>
<tr>
<td>86/11</td>
<td>W.P. de Roever</td>
<td>Questions to Robin Milner - A responder’s commentary (IFIP86).</td>
</tr>
<tr>
<td>86/12</td>
<td>A. Boucher</td>
<td>A timed failures model for extended communicating processes.</td>
</tr>
</tbody>
</table>
86/14 R. Koymans  Specifying passing systems requires extending temporal logic.

87/01 R. Gerth  On the existence of sound and complete axiomatizations of the monitor concept.

87/02 Simon J. Klaver  Federatieve Databases.
Chris F.M. Verberne

87/03 G.J. Houben  A formal approach to distributed information systems.
J. Paredaens

87/04 T. Verhoeff  Delay-insensitive codes - An overview.

87/05 R. Kuiper  Enforcing non-determinism via linear time temporal logic specification.

87/06 R. Koymans  Temporele logica specificatie van message passing en real-time systemen (in Dutch).

87/07 R. Koymans  Specifying message passing and real-time systems with real-time temporal logic.

87/08 H.M.J.L. Schols  The maximum number of states after projection.

87/09 J. Kalisvaart  Language extensions to study structures for raster graphics.
L.R.A. Kessener
W.J.M. Lemmens
M.L.P. van Lierop
F.J. Peters
H.M.M. van de Wetering

87/10 T. Verhoeff  Three families of maximally nondeterministic automata.

87/11 P. Lemmens  Eldorado ins and outs. Specifications of a database management toolkit according to the functional model.

87/12 K.M. van Hee and  OR and AI approaches to decision support systems.
A. Lapinski

87/13 J.C.S.P. van der Woude  Playing with patterns - searching for strings.

87/14 J. Hooman  A compositional proof system for an occam-like real-time language.

87/15 C. Huizing  A compositional semantics for statecharts.
R. Gerth
W.P. de Roever

87/16 H.M.M. ten Eikelder  Normal forms for a class of formulas.
J.C.F. Wilmont

87/17 K.M. van Hee  Modelling of discrete dynamic systems framework and examples.
G.-J. Houben
J.L.G. Dietz
<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>87/18</td>
<td>C.W.A.M. van Overveld</td>
<td>An integer algorithm for rendering curved surfaces.</td>
</tr>
<tr>
<td>87/19</td>
<td>A.J. Seebregts</td>
<td>Optimalisering van file allocatie in gedistribueerde database systemen.</td>
</tr>
<tr>
<td>87/20</td>
<td>G.J. Houben, J. Paredaens</td>
<td>The R²-Algebra: An extension of an algebra for nested relations.</td>
</tr>
<tr>
<td>87/21</td>
<td>R. Gerth, M. Codish, Y. Lichtenstein, E. Shapiro</td>
<td>Fully abstract denotational semantics for concurrent PROLOG.</td>
</tr>
<tr>
<td>88/01</td>
<td>T. Verhoeff</td>
<td>A Parallel Program That Generates the Möbius Sequence.</td>
</tr>
<tr>
<td>88/03</td>
<td>T. Verhoeff</td>
<td>Settling a Question about Pythagorean Triples.</td>
</tr>
<tr>
<td>88/04</td>
<td>G.J. Houben, J. Paredaens, D. Tahon</td>
<td>The Nested Relational Algebra: A Tool to Handle Structured Information.</td>
</tr>
<tr>
<td>88/06</td>
<td>H.M.J.L. Schols</td>
<td>Notes on Delay-Insensitive Communication.</td>
</tr>
<tr>
<td>88/09</td>
<td>A.T.M. Aerts, K.M. van Hee</td>
<td>A Tutorial for Data Modelling.</td>
</tr>
<tr>
<td>88/10</td>
<td>J.C. Ebergen</td>
<td>A Formal Approach to Designing Delay Insensitive Circuits.</td>
</tr>
<tr>
<td>88/12</td>
<td>A.E. Eiben</td>
<td>Abstract theory of planning.</td>
</tr>
<tr>
<td>88/13</td>
<td>A. Bijlsma</td>
<td>A unified approach to sequences, bags, and trees.</td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>88/14</td>
<td>H.M.M. ten Eikelder, R.H. Mak</td>
<td>Language theory of a lambda-calculus with recursive types.</td>
</tr>
<tr>
<td>88/15</td>
<td>R. Bos, C. Hemerik</td>
<td>An introduction to the category theoretic solution of recursive domain equations.</td>
</tr>
<tr>
<td>88/16</td>
<td>C. Hemerik, J.P. Katoen</td>
<td>Bottom-up tree acceptors.</td>
</tr>
<tr>
<td>88/17</td>
<td>K.M. van Hee, G.J. Houben, L.J. Somers, M. Voorhoeve</td>
<td>Executable specifications for discrete event systems.</td>
</tr>
<tr>
<td>88/18</td>
<td>K.M. van Hee, P.M.P. Rambags</td>
<td>Discrete event systems: concepts and basic results.</td>
</tr>
<tr>
<td>88/19</td>
<td>D.K. Hammer, K.M. van Hee</td>
<td>Fasering en documentatie in software engineering.</td>
</tr>
<tr>
<td>88/20</td>
<td>K.M. van Hee, L. Somers, M. Voorhoeve</td>
<td>EXSPECT, the functional part.</td>
</tr>
<tr>
<td>89/1</td>
<td>E.Zs. Lepoeter-Molnar</td>
<td>Reconstruction of a 3-D surface from its normal vectors.</td>
</tr>
<tr>
<td>89/2</td>
<td>R.H. Mak, P.Struik</td>
<td>A systolic design for dynamic programming.</td>
</tr>
<tr>
<td>89/3</td>
<td>H.M.M. Ten Eikelder, C. Hemerik</td>
<td>Some category theoretical properties related to a model for a polymorphic lambda-calculus.</td>
</tr>
<tr>
<td>89/4</td>
<td>J.Zwiers, W.P. de Roever</td>
<td>Compositional alysis and modularity in process specification and design: A trace-state based approach.</td>
</tr>
<tr>
<td>89/5</td>
<td>Wei Chen, T.Verhoeoff, J.T.Udding</td>
<td>Networks of Communicating Processes and their (De-)Composition.</td>
</tr>
<tr>
<td>89/6</td>
<td>T.Verhoeoff</td>
<td>Characterizations of Delay-Insensitive Communication Protocols.</td>
</tr>
<tr>
<td>89/7</td>
<td>P.Struik</td>
<td>A systematic design of a parallel program for Dirichlet convolution.</td>
</tr>
<tr>
<td>89/9</td>
<td>K.M. van Hee, P.M.P. Rambags</td>
<td>Discrete event systems: Dynamic versus static topology.</td>
</tr>
<tr>
<td>89/10</td>
<td>S.Ramesh</td>
<td>A new efficient implementation of CSP with output guards.</td>
</tr>
<tr>
<td>89/11</td>
<td>S.Ramesh</td>
<td>Algebraic specification and implementation of infinite processes.</td>
</tr>
<tr>
<td>Journal</td>
<td>Title</td>
<td>Authors</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>89/12</td>
<td>A concise formal framework for data modeling.</td>
<td>A.T.M. Aerts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K.M. van Hee</td>
</tr>
<tr>
<td>89/13</td>
<td>A program generator for simulated annealing problems.</td>
<td>A.T.M. Aerts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K.M. van Hee</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M.W.H. Hesen</td>
</tr>
<tr>
<td>89/14</td>
<td>ELDA, data manipulatie taal.</td>
<td>H.C. Haesen</td>
</tr>
<tr>
<td>89/15</td>
<td>Optimal segmentations.</td>
<td>J.S.C.P. van der Woude</td>
</tr>
<tr>
<td>89/16</td>
<td>Towards a framework for comparing data models.</td>
<td>A.T.M. Aerts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K.M. van Hee</td>
</tr>
<tr>
<td>89/17</td>
<td>A formal semantics for Z and the link between Z and the relational algebra.</td>
<td>M.J. van Diepen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K.M. van Hee</td>
</tr>
<tr>
<td>90/1</td>
<td>Formal methods and tools for the development of distributed and real time systems, pp. 17.</td>
<td>W.P. de Roever-H. Barringer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C.Courcoubetis-D. Gabbay</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R. Gerth-B. Jonsson-A. Pnueli</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M. Reed-J. Sifakis-J. Vytopil</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P. Wolper</td>
</tr>
<tr>
<td>90/2</td>
<td>Dynamic process creation in high-level Petri nets, pp. 19.</td>
<td>K.M. van Hee</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P.M.P. Rambags</td>
</tr>
<tr>
<td>90/3</td>
<td>Foundations of Compositional Program Refinement - safety properties - , p. 38.</td>
<td>R. Gerth</td>
</tr>
<tr>
<td>90/4</td>
<td>Decomposition of delay-insensitive circuits, p. 25.</td>
<td>A. Peeters</td>
</tr>
<tr>
<td>90/5</td>
<td>On the delay-sensitivity of gate networks, p. 23.</td>
<td>J.A. Brzozowski</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J.C. Ebergen</td>
</tr>
<tr>
<td>90/7</td>
<td>A logic for one-pass, one-attributed grammars, p. 14.</td>
<td>A.J.J.M. Marcelis</td>
</tr>
</tbody>
</table>