A logic for one-pass, one-attributed grammars

A logic for one-pass, one-attributed grammars

by

A.J.J.M. Marcelis

90/7

July, 1990

COMPUTING SCIENCE NOTES

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

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@eutws1.win.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 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 $\{F_A \mid A \in N\}$ of *translation mappings*, acting on the production trees. More specifically, F_A maps production trees of type PT_A onto functions from it_A to st_A (where PT_A is the type of all production trees with root labeled by $A \to \alpha$ (for some α), and it_A (st_A) denotes the inherited (synthesized) attribute type of nonterminal A), i.e., F_A has signature $PT_A \to it_A \to st_A$.

A *specification* for G' then consists of a pair of predicates (Q, R) of signature $Q : it_Z \to bool$ and $R : it_Z \to st_Z \to bool$, and the *correctness condition for* G' is expressed as

$$\forall d{:}PT_Z\,,\, i{:}it_Z\,.\,(Q{\cdot}i \;\Rightarrow\; R{\cdot}i{\cdot}(F_Z{\cdot}d{\cdot}i))$$

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

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 $\mathcal{I}$ 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* $\mathcal{I}$. This means that for each formula φ, derivable within TIS_B^+, its interpretation $\mathcal{I}(\varphi)$ 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 [Mar90]. In particular, it is useful to recall that the correctness condition displayed above is shorthand for

$$\forall\, d{:}PT_Z\,.\,(\forall\, i{:}it_Z\,.\,(\,Q{\cdot}i \;\Rightarrow\; R{\cdot}i{\cdot}(F_Z{\cdot}d{\cdot}i))\,)$$

and similar abbreviations apply to nested λ-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 [Mar90].

If S is a set, then S^* denotes the set of all finite sequences over S, and for a sequence α and set S, notation $\alpha{\restriction}S$ is used for the projection of α 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 Γ, a context over B (in the sense of definition 2.4 of [Mar90]). Γ 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 Γ 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
- Γ is a context over B
- $it = \{it_A \mid A \in N\}$
  $st = \{st_A \mid A \in N\}$
  , where, for all $A \in N$, $B \vdash_{TIS} \Gamma \rhd it_A : *$ and $B \vdash_{TIS} \Gamma \rhd st_A : *$
- P' is a finite set of constructs, a typical element of which, p' say, reads

  $A_0\langle i_0, s_0\rangle \rightarrow w_0\ A_1\langle i_1, s_1\rangle\ w_1 \ldots w_{n-1}\ A_n\langle i_n, s_n\rangle\ w_n$
  $s_0 = e_0\ ,\ i_1 = e_1\ ,\ \ldots\ ,\ i_n = e_n$

  where

  · $n \geq 0$
  · $w_k \in T^*$, for all $k : 0 \leq k \leq n$
  · $\{A_0, A_1, \ldots, A_n\} \subseteq N$
  · $\{i_0, s_0, \ldots, i_n, s_n\} \subseteq V_e$
  · $i_0, s_0, \ldots, i_n, s_n$ are pairwise different
  · $B \vdash_{TIS} \Gamma, i_0 : it_{A_0}, s_1 : st_{A_1}, \ldots, s_n : st_{A_n} \rhd e_k : it_{A_k}$, for all $k : 1 \leq k \leq n$
    $B \vdash_{TIS} \Gamma, i_0 : it_{A_0}, s_1 : st_{A_1}, \ldots, s_n : st_{A_n} \rhd e_0 : st_{A_0}$

□

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 e_k $(0 \leq k \leq n)$ imply that — as far as attribute variables are concerned — $FEV(e_k) \subseteq \{i_0, s_1, \ldots, s_n\}$. 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 e_k $(1 \leq k \leq n)$ of definition 2.1 still further, a sub-class of the one-atributed grammars is obtained:

Definition 2.2 (one-pass condition)
A one-attributed grammar is called *one-pass* (left-to-right) if each element

$$A_0\langle i_0, s_0\rangle \rightarrow w_0\ A_1\langle i_1, s_1\rangle\ w_1 \ldots w_{n-1}\ A_n\langle i_n, s_n\rangle\ w_n$$
$$s_0 = e_0\ ,\ i_1 = e_1\ ,\ \ldots\ ,\ i_n = e_n$$

of P' satisfies

$$B \vdash_{TIS} \Gamma\ ,\ i_0{:}it_{A_0}\ ,\ s_1{:}st_{A_1}\ ,\ \ldots\ ,\ s_{k-1}{:}st_{A_{k-1}}\ \rhd\ e_k : it_{A_k} \quad ,\ \text{for all } k : 1 \leq k \leq n$$

(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 — qua size and type — with that 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 by a production rule with 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

$$\begin{array}{l}\textbf{rec}\ (\ \ldots \\ \quad ,\ PT_{A_0} =_t \textbf{sum}(\ \ldots\ ,\ A_0 \rightarrow \alpha : \textbf{prod}(PT_{A_1}, \ldots, PT_{A_n})\ ,\ \ldots) \\ \quad ,\ \ldots \\ \quad)\end{array}$$

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 $\alpha{\restriction}N = A_1 \ldots A_n$.

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 α, 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

$$PT\ =_t\ \textbf{sum}(\ \ldots\ ,\ B_0 \rightarrow \beta : \textbf{prod}(PT_{B_1}, \ldots, PT_{B_m})\ ,\ \ldots)$$

where the sum type contains an alternative per production rule in P. Above the alternative for

rule $B_0 \rightarrow \beta$ is shown, with $\beta \restriction N = B_1 \ldots B_m$. 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

$$\begin{array}{l} \mathbf{rec}\ (\ PT_Z =_t \mathbf{sum}(\ Z \rightarrow z : \mathbf{prod}()\ ,\ Z \rightarrow YYy : \mathbf{prod}(PT_Y, PT_Y))\ ) \\ \quad ,\ PT_Y =_t \mathbf{sum}(\ Y \rightarrow Zx : \mathbf{prod}(PT_Z))\ ) \\ \quad) \end{array}$$

Hence, an expression of type PT_Z is of either of the forms $[\ Z \rightarrow z, \langle\rangle\ ]$ or $[\ Z \rightarrow YYy, \langle d1, d2\rangle\ ]$, where $d1$ and $d2$ are expressions of type PT_Y. Likewise, an expression of type PT_Y is of the form $[\ Y \rightarrow Zx, \langle d\rangle\ ]$, with d of type PT_Z. An example of the latter is $[\ Y \rightarrow Zx, \langle [\ Z \rightarrow z, \langle\rangle\ ] \rangle\ ]$.
□

For an attribute grammar G' with underlying context-free grammar G we formulate an inference rule from TIS_B, 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 $B \vdash_{TIS} D \rhd R_A : PT_A \rightarrow bool$ (for all $A \in N$). Then the induction rule reads

$$\frac{\begin{array}{l} D \ \rhd\ \forall A \rightarrow \alpha \in P\ ,\ \langle d_1, \ldots, d_n\rangle : \mathbf{prod}(PT_{A_1}, \ldots, PT_{A_n}) \\ \qquad\qquad .\ (\ R_{A_1} \cdot d_1 \wedge \ldots \wedge R_{A_n} \cdot d_n \ \Rightarrow\ R_A \cdot [\ A \rightarrow \alpha, \langle d_1, \ldots, d_n\rangle\ ]\ ) \end{array}}{D \ \rhd\ \forall A \in N\ ,\ d : PT_A\ .\ (\ R_A \cdot d\ )}$$

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

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 $PT_A \rightarrow it_A \rightarrow st_A$, 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 it_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 it_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 it_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 $G' = (N, T, P', Z, it, st, B, \Gamma)$ 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

$$
\begin{array}{l}
\textbf{rec}\ (\ \ldots \\
\quad ,\ F_{A_0} =_e \lambda\, d{:}PT_{A_0}\,.\,(\ \textbf{case}\ d\ \textbf{of} \\
\qquad\qquad\qquad\qquad \ldots \\
\qquad\qquad\qquad\qquad ,\ [\, A_0 \to \alpha, \langle d_1, \ldots, d_n \rangle\,]\ \textbf{then}\ H_{A_0 \to \alpha}{\cdot}(F_{A_1}{\cdot}d_1){\cdot}\ldots{\cdot}(F_{A_n}{\cdot}d_n) \\
\qquad\qquad\qquad\qquad ,\ \ldots \\
\qquad\qquad\qquad)\ \\
\quad ,\ \ldots \\
\quad)
\end{array}
$$

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 $A_0 \to \alpha$ as their root label, where $\alpha \restriction N = A_1, \ldots, A_n$. 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 F_{A_0} applied to a tree of the form $[\, A_0 \to \alpha, \langle d_1, \ldots, d_n \rangle\,]$ yields $H_{A_0 \to \alpha}{\cdot}(F_{A_1}{\cdot}d_1){\cdot}\ldots{\cdot}(F_{A_n}{\cdot}d_n)$. Herein, $F_{A_1}{\cdot}d_1$ through $F_{A_n}{\cdot}d_n$ are the applications of the appropriate translation mappings to the direct subtrees of $[\, A_0 \to \alpha, \langle d_1, \ldots, d_n \rangle\,]$, and $H_{A_0 \to \alpha}$ is a higher-order function completely determined by the attributed production rule in P' that has $A_0 \to \alpha$ as its underlying context-free rule (the structure of the H-functions will be dealt with hereafter).
□

Example 2.6 (on translation mappings)
Consider the one-pass, one-attributed grammar $G' = (N, T, P', Z, it, st, B, \Gamma)$, of which only the following components will be specified

- $N = \{\, Z\,,\, Y\,\}$
- $T = \{\, z\,,\, y\,,\, x\,\}$
- $P' = \{\ Z\langle i_0, s_0\rangle \to z$
  $\qquad s_0 = u_0$
  $\quad ,\ Z\langle i_0, s_0\rangle \to Y\langle i_1, s_1\rangle\ Y\langle i_2, s_2\rangle\ y$
  $\qquad s_0 = v_0\ ,\ i_1 = v_1\ ,\ i_2 = v_2$
  $\quad ,\ Y\langle i_0, s_0\rangle \to Z\langle i_1, s_1\rangle\ x$
  $\qquad s_0 = w_0\ ,\ i_1 = w_1$
  $\}$

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 $PT_Z \to it_Z \to st_Z$ and F_Y has type $PT_Y \to it_Y \to st_Y$. The definition reads

$$
\begin{array}{l}
\textbf{rec}\ (\ F_Z =_e \lambda\, d{:}PT_Z\,.\,(\ \textbf{case}\ d\ \textbf{of} \\
\qquad\qquad\qquad\qquad [\, Z \to z, \langle\,\rangle\,]\ \textbf{then}\ H_{Z \to z}, \\
\qquad\qquad\qquad\qquad [\, Z \to YYy, \langle d_1, d_2\rangle\,]\ \textbf{then}\ H_{Z \to YYy}{\cdot}(F_Y{\cdot}d_1){\cdot}(F_Y{\cdot}d_2) \\
\qquad\qquad\qquad)\ \\
\quad ,\ F_Y =_e \lambda\, d{:}PT_Y\,.\,(\ \textbf{case}\ d\ \textbf{of}\ [\, Y \to Zx, \langle d_1\rangle\,]\ \textbf{then}\ H_{Y \to Zx}{\cdot}(F_Z{\cdot}d_1)\,) \\
\quad)
\end{array}
$$

wherein H_p (for $p \in \{Z \to z\,,\, Z \to YYy\,,\, Y \to Zx\}$) 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.
□

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}, $H_{A_0 \to \alpha}$, with $\alpha \restriction N = A_1 \ldots A_n$, 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':

$$A_0\langle i_0, s_0\rangle \to w_0\ A_1\langle i_1, s_1\rangle\ w_1 \ldots w_{n-1}\ A_n\langle i_n, s_n\rangle\ w_n$$
$$s_0 = e_0\ ,\ i_1 = e_1\ , \ldots ,\ i_n = e_n$$

of G' gives rise to H_p, where p is the underlying context-free rule of p', defined by

$$\begin{array}{l} H_p =_e \lambda\, h_1 : (it_{A_1} \to st_{A_1})\ ,\ \ldots\ ,\ h_n : (it_{A_n} \to st_{A_n}) \\ \quad .(\lambda\ i_0 : it_{A_0} \\ \quad\quad .(\textbf{let}\ \ i_1 : it_{A_1} = e_1\ , \\ \quad\quad\quad .\ \ s_1 : st_{A_1} = h_1 \cdot i_1\ , \\ \quad\quad\quad\quad\quad \vdots \\ \quad\quad\quad\quad i_n : it_{A_n} = e_n\ , \\ \quad\quad\quad\quad s_n : st_{A_n} = h_n \cdot i_n\ , \\ \quad\quad\quad\quad s_0 : st_{A_0} = e_0 \\ \quad\quad\quad \textbf{in}\ s_0 \\ \quad\quad) \\ \quad) \end{array}$$

□

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[1]

$$\begin{array}{l} H_p =_e \lambda\, h_1 : (it_{A_1} \to st_{A_1})\ ,\ \ldots\ ,\ h_n : (it_{A_n} \to st_{A_n}) \\ \quad .(\lambda\ i_0 : it_{A_0} \\ \quad\quad .(\textbf{let}\ \ s_1 : st_{A_1} = h_1 \cdot e_1\ , \\ \quad\quad\quad\quad\quad \vdots \\ \quad\quad\quad\quad s_n : st_{A_n} = h_n \cdot e_n\ , \\ \quad\quad\quad \textbf{in}\ e_0 \\ \quad\quad) \\ \quad) \end{array} \qquad (*)$$

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 $(1 \leq k \leq n)$ $FEV(e_k) \subseteq \{i_0, s_1, \ldots, s_{k-1}\}$, of which i_0 is bound in the enclosing λ-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_l $(1 \leq l \leq n)$, 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 $H_{Z\to z}$ and $H_{Z\to YYy}$, determined by attributed production rules

$$\begin{array}{l} Z\langle i_0, s_0\rangle \to z \\ s_0 = u_0 \end{array} \qquad \text{and} \qquad \begin{array}{l} Z\langle i_0, s_0\rangle \to Y\langle i_1, s_1\rangle\ Y\langle i_2, s_2\rangle\ y \\ s_0 = v_0\ ,\ i_1 = v_1\ ,\ i_2 = v_2 \end{array}$$

of example 2.6 (and used in the translation mappings of that example).
$H_{Z\to z}$ is a function of type $it_Z \to st_Z$. Using the abbreviated notation $(*)$, it is defined by

$$H_{Z\to z} =_e \lambda\, i_0 : it_Z\, .\, u_0$$

$H_{Z\to YYy}$ is a function of type $(it_Y \to st_Y) \to (it_Y \to st_Y) \to (it_Z \to st_Z)$. Its definition reads

$$\begin{array}{l} H_{Z\to YYy} =_e \lambda\, h_1 : (it_Y \to st_Y)\ ,\ h_2 : (it_Y \to st_Y) \\ \qquad .\,(\lambda\, i_0 : it_Z \\ \qquad\quad .\,(\textbf{let}\ s_1 : st_Y = h_1{\cdot}v_1\ ,\ s_2 : st_Y = h_2{\cdot}v_2\ \textbf{in}\ v_0\,) \\ \qquad)\end{array}$$

□

Now consider the use of $H_{A_0\to\alpha}$ in the definition of translation mapping F_{A_0} ($A_0 \to \alpha$ is the underlying context-free rule of p' as in definition 2.7). According to definition 2.5, F_{A_0} applied to $[\, A_0 \to \alpha, \langle d_1, \ldots, d_n\rangle\,]$ yields $H_{A_0\to\alpha}{\cdot}(F_{A_1}{\cdot}d_1){\cdot}\ldots{\cdot}(F_{A_n}{\cdot}d_n)$. The latter reduces to (using $(*)$ for simplicity):

$$\begin{array}{l} \lambda\, i_0 : it_{A_0}\,.\,(\textbf{let}\ s_1 : st_{A_1} = F_{A_1}{\cdot}d_1{\cdot}e_1\ , \\ \qquad\qquad\qquad\qquad \vdots \\ \qquad\qquad\quad s_n : st_{A_n} = F_{A_n}{\cdot}d_n{\cdot}e_n \\ \qquad\quad \textbf{in}\ e_0 \\ \qquad)\end{array}$$

Thus, $H_{A_0\to\alpha}$ 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, $H_{A_0\to\alpha}$ reflects the nature of the inherited and synthesized attributes of $A_0 \to \alpha$ (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 $B \vdash_{TIS} \Gamma \rhd Q : it_Z \to bool$ and $B \vdash_{TIS} \Gamma \rhd R : it_Z \to st_Z \to bool$.

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 $B \vdash_{TIS} \Gamma \rhd Q : it_Z \rightarrow bool$ and $B \vdash_{TIS} \Gamma \rhd R : it_Z \rightarrow st_Z \rightarrow bool$, i.e., the pair (Q, R) is a *specification* for G'.
The *correctness condition for* G' *w.r.t.* (Q, R) reads

$$\forall d{:}PT_Z\,,\, i{:}it_Z\,.\,(Q{\cdot}i \;\Rightarrow\; R{\cdot}i{\cdot}(F_Z{\cdot}d{\cdot}i))$$

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

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 $Q{\cdot}i$, F_Z applied to d and i satisfies $R{\cdot}i{\cdot}(F_Z{\cdot}d{\cdot}i)$. However, as the collection $\{F_A \mid A \in N\}$ 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 $\{Q_A \mid A \in N\}$ and $\{R_A \mid A \in N\}$ of the appropriate types (with $Q_Z = Q$ and $R_Z = R$) and a correctness condition

$$\forall d{:}PT_A\,,\, i{:}it_A\,.\,(Q_A{\cdot}i \;\Rightarrow\; R_A{\cdot}i{\cdot}(F_A{\cdot}d{\cdot}i))$$

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^+, as an extension of TIS_B. To save writing in this definition, assume that $N = \{A_1, \ldots, A_n\}$, assume that $Q_{A_1}, \ldots, Q_{A_n}$ and $R_{A_1}, \ldots, R_{A_n}$ are such that

$$B \vdash_{TIS} \Gamma \rhd Q_{A_k} : it_{A_k} \rightarrow bool \quad \text{(for all } k : 1 \leq k \leq n)$$
$$B \vdash_{TIS} \Gamma \rhd R_{A_k} : it_{A_k} \rightarrow st_{A_k} \rightarrow bool \quad \text{(for all } k : 1 \leq k \leq n)$$

and let Γ' denote the sequence

$$q_{A_1} : it_{A_1} \rightarrow bool\,,\; q_{A_1} =_e Q_{A_1}\,,\; \ldots\,,\; q_{A_n} : it_{A_n} \rightarrow bool\,,\; q_{A_n} =_e Q_{A_n}\,,$$
$$r_{A_1} : it_{A_1} \rightarrow st_{A_1} \rightarrow bool\,,\; r_{A_1} =_e R_{A_1}\,,\; \ldots\,,\; r_{A_n} : it_{A_n} \rightarrow st_{A_n} \rightarrow bool\,,\; r_{A_n} =_e R_{A_n}$$

i.e., Γ' serves to select a collection $\{Q_{A_k}\,,\, R_{A_k} \mid A_k \in N\}$ of predicates and to bind these predicates to the variables $\{q_{A_k}\,,\, r_{A_k} \mid A_k \in N\}$. Notice that the predicates can be typed in a context only containing Γ.

The logic TIS_B^+ consists of inference formulae and rules as follows
Inference formulae are

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

· 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

$$A_0\langle i_0, s_0\rangle \rightarrow w_0\ A_1\langle i_1, s_1\rangle\ w_1 \ldots w_{n-1}\ A_n\langle i_n, s_n\rangle\ w_n$$
$$s_0 = e_0\ ,\ i_1 = e_1\ ,\ \ldots\ ,\ i_n = e_n$$

the rule

$$\frac{\begin{array}{l}\Gamma\ ,\ \Gamma'\ , \\ i_0 : it_{A_0}\ ,\ s_0 : st_{A_0}\ ,\ \ldots\ ,\ i_n : it_{A_n}\ ,\ s_n : st_{A_n}\ , \\ s_0 =_e e_0\ ,\ i_1 =_e e_1\ ,\ \ldots\ ,\ i_n =_e e_n \\ \rhd \\ q_{A_0}{\cdot}i_0 \wedge \bigwedge_{j=1}^{k-1}(q_{A_j}{\cdot}i_j \wedge r_{A_j}{\cdot}i_j{\cdot}s_j) \ \Rightarrow\ q_{A_k}{\cdot}i_k \quad \text{for all } k : 1 \leq k \leq n \\ q_{A_0}{\cdot}i_0 \wedge \bigwedge_{j=1}^{n}(q_{A_j}{\cdot}i_j \wedge r_{A_j}{\cdot}i_j{\cdot}s_j) \ \Rightarrow\ r_{A_0}{\cdot}i_0{\cdot}s_0\end{array}}{\Gamma\ ,\ \Gamma' \rhd (pr\ correct)}$$

3. $$\frac{\Gamma\ ,\ \Gamma' \rhd (pr_1\ correct)\ ,\ \ldots\ ,\ \Gamma\ ,\ \Gamma' \rhd (pr_m\ correct)}{\Gamma \rhd (G', Q, R)}$$

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

Recall that Γ' 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 Γ' 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 φ within TIS_B^+ is denoted by $B \vdash_{TIS+} \varphi$.

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 $L_1 = (\mathcal{F}_1\ ,\ \mathcal{R}_1)$ and $L_2 = (\mathcal{F}_2\ ,\ \mathcal{R}_2)$ be two inference systems, with $\mathcal{F}_1 \supseteq \mathcal{F}_2$ and $\mathcal{R}_1 \supseteq \mathcal{R}_2$. Denote the derivability of a formula φ 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., $\mathcal{J} \in \mathcal{F}_1 \rightarrow \mathcal{F}_2$.

a. Rule $\frac{\varphi_1 \ldots \varphi_n}{\varphi} \in \mathcal{R}_1$ *is consistent w.r.t.* L_2 *under* $\mathcal{J}$ if
$\vdash_{L_2} \mathcal{J}(\varphi_1)$ and ... and $\vdash_{L_2} \mathcal{J}(\varphi_n)$ imply $\vdash_{L_2} \mathcal{J}(\varphi)$.

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

□

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 $\mathcal{R}_1$ are consistent w.r.t. L_2 under $\mathcal{J}$.

proof: straightforward, using the notion of derivability of formulae within an inference system.
□

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 ($\mathcal{I}$, interpretation)
Let G', TIS_B, TIS_B^+ and Γ' be as in the preceding section. Let Δ be a piece of context containing

— the type definitions for $\{PT_A \mid A \in N\}$, the types of the production trees of G

— the definitions for $\{F_A \mid A \in N\}$ and $\{H_{A\to\alpha} \mid A \to \alpha \in P\}$, concerning the translation mappings induced by G'

Finally, let Δ' be a piece of context containing a clause

$$\Phi_A : PT_A \to bool\ ,\ \Phi_A =_e \lambda d{:}PT_A\,.\,(\forall i{:}it_A\,.\,(\,q_A{\cdot}i \Rightarrow r_A{\cdot}i{\cdot}(F_A{\cdot}d{\cdot}i))\,)$$

for each $A \in N$. Notice that, this way, $\forall d{:}PT_A\,.\,(\Phi_A{\cdot}d)$ 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 Γ'.

Then $\mathcal{I}$ maps formulae of TIS_B^+ onto formulae of TIS_B according to

1. $\mathcal{I}(\ D \rhd \varphi\ )\ =\ D \rhd \varphi$,for $D \rhd \varphi$ an inference formula of TIS_B

2. $\mathcal{I}(\ \Gamma\,,\,\Gamma'\ \rhd\ (pr\ correct)\ )\ =$

$$\Gamma\,,\,\Gamma'\,,\,\Delta\,,\,\Delta'$$
$$\rhd$$
$$\forall\,(d_1,\ldots,d_n) : \mathbf{prod}(PT_{A_1},\ldots,PT_{A_n})$$
$$.\,(\,\Phi_{A_1}{\cdot}d_1 \wedge \ldots \wedge \Phi_{A_n}{\cdot}d_n\ \Rightarrow\ \Phi_{A_0}{\cdot}[\,A_0 \to \alpha\,,\,(d_1,\ldots,d_n)\,]\,)$$

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

3. $\mathcal{I}(\ \Gamma\ \rhd\ (G', Q, R)\ )\ =\ \Gamma\,,\,\Delta\ \rhd\ \forall d{:}PT_Z\,,\,i{:}it_Z\,.\,(\,Q{\cdot}i \Rightarrow R{\cdot}i{\cdot}(F_Z{\cdot}d{\cdot}i)\,)$

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

Theorem 4.4
Let G', TIS_B, TIS_B^+, Γ', Δ, Δ' be as in definition 4.3, and let $\mathcal{I}$ be as defined by the latter definition. Then TIS_B^+ is consistent w.r.t. TIS_B under $\mathcal{I}$.

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

$$A_0\langle i_0, s_0\rangle \to w_0\ A_1\langle i_1, s_1\rangle\ w_1 \ldots w_{n-1}\ A_n\langle i_n, s_n\rangle\ w_n$$
$$s_0 = e_0\ ,\ i_1 = e_1\ ,\ \ldots\ ,\ i_n = e_n$$

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 $B \vdash_{TIS} \mathcal{I}(\Gamma, \Gamma' \rhd (pr\ correct))$. Namely, starting from

Γ , Γ'

$i_0 : it_{A_0}$, $s_0 : st_{A_0}$, ... , $i_n : it_{A_n}$, $s_n : st_{A_n}$,
$s_0 =_e e_0$, $i_1 =_e e_1$, ... , $i_n =_e e_n$

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

$q_{A_0} \cdot i_0 \ \wedge \ \bigwedge_{j=1}^{n} (q_{A_j} \cdot i_j \wedge r_{A_j} \cdot i_j \cdot s_j) \ \Rightarrow \ r_{A_0} \cdot i_0 \cdot s_0$

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

1. Γ , Γ' , Δ , Δ'
2. $\langle d_1, \ldots, d_n \rangle : \mathrm{prod}(PT_{A_1}, \ldots, PT_{A_n})$
3. $\Phi_{A_1} \cdot d_1 \wedge \ldots \wedge \Phi_{A_n} \cdot d_n$
4. $i_0 : it_{A_0}$, $q_{A_0} \cdot i_0$
5. $i_1 : it_{A_1}$, $i_1 =_e e_1$
6. $s_1 : st_{A_1}$, $s_1 =_e F_{A_1} \cdot d_1 \cdot i_1$

   ⋮
7. $i_n : it_{A_n}$, $i_n =_e e_n$
8. $s_n : st_{A_n}$, $s_n =_e F_{A_n} \cdot d_n \cdot i_n$
9. $s_0 : st_{A_0}$, $s_0 =_e e_0$
10. $q_{A_0} \cdot i_0 \ \wedge \ \bigwedge_{j=1}^{k-1} (q_{A_j} \cdot i_j \wedge r_{A_j} \cdot i_j \cdot s_j) \ \Rightarrow \ q_{A_k} \cdot i_k$, for all $k : 1 \leq k \leq n$
11. $q_{A_0} \cdot i_0 \ \wedge \ \bigwedge_{j=1}^{n} (q_{A_j} \cdot i_j \wedge r_{A_j} \cdot i_j \cdot s_j) \ \Rightarrow \ r_{A_0} \cdot i_0 \cdot s_0$

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 ∀,3,5-9)
13. $q_{A_0} \cdot i_0$ (4)
14. $q_{A_1} \cdot i_1$ (el ⇒,10,13)
15. $r_{A_1} \cdot i_1 \cdot s_1$ (el ⇒,12,14)

    ⋮
16. $q_{A_n} \cdot i_n$ (el ⇒,10,...)
17. $r_{A_n} \cdot i_n \cdot s_n$ (el ⇒,12,16)
18. $r_{A_0} \cdot i_0 \cdot s_0$ (el ⇒,11,...)
19. $r_{A_0} \cdot i_0 \cdot (\mathrm{let}\, i_1 : it_{A_1} = e_1, s_1 : st_{A_1} = F_{A_1} \cdot d_1 \cdot i_1, \ldots, s_0 : st_{A_0} = e_0 \ \mathrm{in}\ s_0)$ (let-rule, repeatedly)
20. $r_{A_0} \cdot i_0 \cdot (H_{A_0 \to \alpha} \cdot (F_{A_1} \cdot d_1) \cdot \ldots \cdot (F_{A_n} \cdot d_n) \cdot i_0)$ (def. $H_{A_0 \to \alpha}$)
21. $r_{A_0} \cdot i_0 \cdot (F_{A_0} \cdot [\, A_0 \to \alpha, \langle d_1, \ldots, d_n \rangle\,] \cdot i_0)$ (def. F_{A_0})
22. $\forall\, i_0 : it_{A_0} \,.\, (\, q_{A_0} \cdot i_0 \ \Rightarrow \ r_{A_0} \cdot i_0 \cdot (F_{A_0} \cdot [\, A_0 \to \alpha, \langle d_1, \ldots, d_n \rangle\,] \cdot i_0)\,)$ (in ⇒,in ∀,4,21)
23. $\Phi_{A_0} \cdot [\, A_0 \to \alpha, \langle d_1, \ldots, d_n \rangle\,]$ (def. Φ)
24. $\forall \langle d_1, \ldots, d_n \rangle : \mathrm{prod}(PT_{A_1}, \ldots, PT_{A_n})$
    $\,.\, (\, \Phi_{A_1} \cdot d_1 \wedge \ldots \wedge \Phi_{A_n} \cdot d_n \ \Rightarrow \ \Phi_{A_0} \cdot [\, A_0 \to \alpha, \langle d_1, \ldots, d_n \rangle\,]\,)$ (in ⇒,in ∀,2,3,23)

Line 24 (in its proper context, viz. line 1) displays $\mathcal{I}(\, \Gamma, \Gamma', \rhd (pr\ correct)\,)$. Therefore rules of kind 2 are consistent w.r.t. TIS_B under $\mathcal{I}$.

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

1. $\Gamma\ ,\ \Delta$
2. Γ'
3. Δ'
4. $\forall A \rightarrow \alpha \in P\ ,\ \langle d_1, \ldots, d_n \rangle : \mathrm{prod}(PT_{A_1}, \ldots, PT_{A_n})$
$$.\,(\,\Phi_{A_1}{\cdot}d_1\ \wedge\ \ldots\ \wedge\ \Phi_{A_n}{\cdot}d_n\ \Rightarrow\ \Phi_A{\cdot}[\,A \rightarrow \alpha, \langle d_1, \ldots, d_n \rangle]\,)$$

and, continuing:

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

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

With Lemma 4.2, the result now holds as claimed.
□

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 λ-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.

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

[Apt81] Apt, K.R.; *Ten Years of Hoare's Logic: A Survey — Part I*, ACM TOPLAS **3**, 4, pp. 431–483 (1981)

[Boc76] Bochmann, G.V.; *Semantic Evaluation from Left to Right*, Comm. ACM **19**, pp. 55–62 (1976)

[C&D88] Courcelle, B., and P. Deransart; *Proofs of partial correctness for attribute grammars with applications to recursive procedures and logic programming*, Information and Computation **78**, pp. 1–55 (1988)

[Eng84] Engelfriet, J.; *Attribute Grammars: Attribute Evaluation Methods*, in: Methods and Tools for Compiler Construction (B. Lorho, ed.), Cambridge U.P., pp. 103–138 (1984)

[Fil83] Filè, G.; *Theory of attribute grammars*, Dissertation, Twente University of Technology (1983)

[Flo67] Floyd, R.W.; *Assigning meanings to programs*, Proc. AMS Symp. Applied Mathematics, AMS, pp. 19–31 (1967)

[Joh87] Johnsson, T.; *Attribute Grammars as a Functional Programming Paradigm*, in: Proc. Functional Programming Languages and Computer Architecture, Portland (USA), LNCS 274, pp. 154–173 (1987)

[K&H81] Katayama, T., and Y. Hoshino; *Verification of Attribute Grammars*, Proc. ACM Symp. on POPL, ACM, pp. 177–186 (1981)

[Mar90] Marcelis, A.J.J.M.; *Typed Inference Systems: A Reference Document*, Computing Science Note 90/06, Eindhoven University of Technology, Dept. of Math. and Comp. Sci., The Netherlands (1990)

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS--circuits.
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes.
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films.
85/04	T. Verhoeff H.M.L.J.Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate.
86/01	R. Koymans	Specifying message passing and real-time systems.
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specification of information systems.
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures.
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several systems.
86/05	J.L.G. Dietz K.M. van Hee	A framework for the conceptual modeling of discrete dynamic systems.
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP.
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers.
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987).
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language.
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing.
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86).
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes.
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4).

86/14	R. Koymans	Specifying passing systems requires extending temporal logic.
87/01	R. Gerth	On the existence of sound and complete axiomati zations of the monitor concept.
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases.
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems.
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 L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
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 R. Gerth W.P. de Roever	A compositional semantics for statecharts.
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas.
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples.

87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces.
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen.
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations.
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG.
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence.
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems.
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples.
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to Handle Structured Information.
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems.
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication.
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way.
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification.
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling.
88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits.
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases.
88/12	A.E. Eiben	Abstract theory of planning.
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees.

88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types.
88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations.
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors.
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems.
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results.
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part.
89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.

89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.
90/1	W.P.de Roever-H.Barringer C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, pp. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.