An example of proving attribute grammars correct : the representation of arithmetical expressions by DAGs

Eindhoven University of Technology

Department of Mathematics and Computing Science

An example of proving attribute grammars correct: the representattion of arithmetical expressions by DAGs

by

A.J.J.M. Marcelis

Computing Science Note 91/16
Eindhoven, September 1991

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.

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs

A.J.J.M. Marcelis

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O.Box 513, 5600 MB Eindhoven
The Netherlands

Abstract

The proof system for one-pass attribute grammars is employed to prove the correctness of an example AG in a modular fashion. The AG concerns the translation of arithmetical expressions — as they are produced by the underlying CFG — into directed acyclic graphs (which are a suitable basis for the generation of efficient code for arithmetical expressions).

The entities needed to specify the problem formally are provided within the framework of typed inference systems and relevant properties of these entities are investigated. Such a formalisation is a prerequisite for the application of the proof system.

The emphasis is on the proof of the AG, rather than on its derivation. In particular, the exercise exemplifies how the remaining proof obligations for a non-trivial AG, yielded by the proof system, take the shape of a number of clear, logical formulae per production, the proofs of which can be conducted entirely by formal manipulation. The example also shows how such proof obligations can be further split up into elementary parts. This enhances a separation of concerns and results in a number of small proofs to be conducted, most of which are fairly simple and proceed on a "nothing else you can do" basis.

Preface: the relation to earlier work

The present paper embodies an application of the proof system for one-pass attribute grammars, as described — in the first instance — in volume 90/07 of this series of Computing Science Notes; see [Mar90].

Since the publication of the latter note, new insights have led to a slight modification of the theory, viz. the explicit inclusion of a reference to derivation trees in the specifications. As a result, the correctness condition for a one-pass AG w.r.t. Q and R — cf. section 2.4 of [Mar90] — now reads

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

where the types of Q and R are $PT_Z \to it_Z \to bool$ and $PT_Z \to it_Z \to st_Z \to bool$, respectively. This correctness condition then generalises in a straightforward way to nonterminals other than start symbol Z, as described in [Mar90]. Also the inference rule concerning a production pr of the form

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

changes slightly, so as to become (cf. [Mar90], p. 10)

$$\frac{\begin{array}{l} \Gamma\ ,\ \Gamma'\ ,\ d_1 : PT_{A_1}\ ,\ \ldots\ ,\ d_n : PT_{A_n}\ ,\ d_0 : PT_{A_0}\ ,\ d_0 =_e [A_0 \to \alpha, \langle d_1, \ldots, d_n\rangle] \\ i_0 : it_{A_0}\ ,\ s_0 : st_{A_0}\ ,\ \ldots\ ,\ i_n : it_{A_n}\ ,\ s_n : st_{A_n}\ , \\ s_0 =_e e_0\ ,\ i_1 =_e e_1\ ,\ \ldots\ ,\ i_n =_e e_n \\ \rhd \\ q_{A_0}{\cdot}d_0{\cdot}i_0 \wedge \bigwedge_{j=1}^{k-1}(q_{A_j}{\cdot}d_j{\cdot}i_j \wedge r_{A_j}{\cdot}d_j{\cdot}i_j{\cdot}s_j) \;\Rightarrow\; q_{A_k}{\cdot}d_k{\cdot}i_k \quad \text{for all } k : 1 \le k \le n \\ q_{A_0}{\cdot}d_0{\cdot}i_0 \wedge \bigwedge_{j=1}^{n}(q_{A_j}{\cdot}d_j{\cdot}i_j \wedge r_{A_j}{\cdot}d_j{\cdot}i_j{\cdot}s_j) \;\Rightarrow\; r_{A_0}{\cdot}d_0{\cdot}i_0{\cdot}s_0 \end{array}}{\Gamma\ ,\ \Gamma' \rhd (pr\ correct)}$$

It is the premiss of this rule that will be used in this paper to serve as a remaining proof obligation for production pr. The consistency proof for this inference rule runs completely analogous to the proof of theorem 4.4 in [Mar90].

The change in the theory described above is the result of still continuing research into the development of proof rules for AGs; as such it will be covered in future work. In the same vein the example dealt with in this paper is to be processed as a part of a more comprehensive treatise later on.

Eindhoven, August 1991 Jos Marcelis

Contents

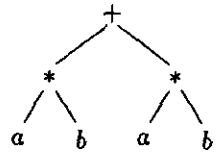
1 Introduction

The proof system for one-pass attribute grammars allows us to prove the correctness of such AGs in a modular fashion, viz. by showing a number of conditions local to the productions. In this paper the use of the proof system is demonstrated by showing the correctness of an AG that translates arithmetical expressions — consisting of constants, variables and operators — into directed acyclic graphs (DAGs).

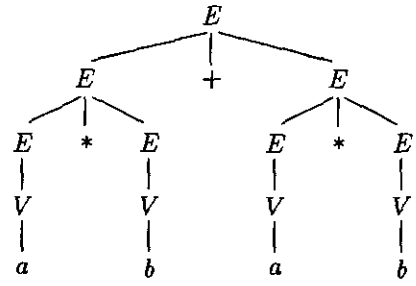
Such graph representations allow of the identification of the common sub-expressions of an expression, and are hence a suitable starting-point for a (back-end) code generator: they enable the code for evaluating common sub-expressions to be generated only once, instead of multiple times. The actual organisation of such a code generator is outside the scope of this note[1]; see e.g. [ASU86], [Hem85].

DAGs are usually related to an abstract form of parse trees (called *syntax trees* in the more general setting of [ASU86]), instead of the parse trees themselves, because the latter often contain disturbing irregularities, motivated by parsing considerations. Although this aspect only plays a minor role in our example grammar, in order to make the example more realistic we shall follow the common practice and view DAGs in relation with *expression trees*. (With regard to the usual production trees[2], expression trees will turn out to eliminate chain productions.)

In expression trees, an operator-labelled node is the father of the root nodes of the trees representing the operator's operands, instead of being a brother among the latter nodes (as in the usual parse trees). For example, an expression tree for an expression like $(a * b) + (a * b)$ may be depicted as



whereas its parse tree may look like



Then again, a DAG for the expression $(a * b) + (a * b)$ may be depicted

[1] The use of an intermediate language in the process of language translation is intended to separate the (back-end) code generation from specific features of the source language. It is therefore not appropriate to express this code generation by means of an AG on the structure of the source language.

[2] Within the scope of this informal introduction, the notions *production tree* and *parse tree* are identified. From section 2 onwards, we adopt a more rigorous view and only talk about production trees — in the formal meaning of the word, viz. as expressions of a certain recursively defined type.

This DAG bears to the expression tree above the relationship that the two occurrences of the sub-expression $(a * b)$ are identified. Taking it as a starting-point for a code generator, code for evaluating $(a * b)$ may be generated only once, instead of twice.

Based on a CFG producing arithmetical expressions, this paper now presents a one-pass AG that delivers a DAG for each complete production tree. Utilising the proof rules for one-pass AGs, it is then shown that the AG meets its specification, which means notably that the DAG delivered — for each production tree — indeed represents the expression tree corresponding to that production tree, and that it identifies common sub-expressions.

Of course, in order to express and reason about this problem in a precise way, the notions occurring in it must be defined formally. The framework of typed inference systems is employed to serve this goal; it is the topic of sections 3 and 4.

Rather than the development of the AG, the main purpose of the paper is to see the proof rules for one-pass AGs at work in connection with a fairly realistic example. In the same vein, the specific example is of minor importance, but only serves as a vehicle for this purpose.

In more detail, the paper is organised as follows. Section 2 introduces a simple CFG producing arithmetical expressions. Section 3 defines the expression trees corresponding to such expressions — or rather: to the production trees of the CFG. Next, in section 4 it is shown how such expression trees can be represented by acyclic pointer-like structures (DAGs). In section 5 we list a number of properties satisfied by the notions introduced in the preceding sections; these will be useful when proving (and deriving, for that matter) the AG to follow. In section 6 the CFG is turned into a one-pass AG, as described above. Section 7 contains an elaborate proof of the fact that the AG meets its specification. In section 8, then, we discuss a (correctness preserving) transformation of particular evaluation rules, aimed at improving the efficiency of attribute evaluation. This section is somewhat outside the original scope of the paper; it anticipates a future treatment of this example in a transformational context. Finally, section 9 contains some concluding remarks and addresses some questions about this example that have not been answered satisfactorily up to date.

2 Context-free grammar

We consider the following (part of a) CFG G, producing typical arithmetical expressions.

1. $Z \rightarrow E$
2. $E \rightarrow (E + E)$
3. $E \rightarrow (E * E)$
4. $E \rightarrow V$
5. $E \rightarrow C$

Z is the grammar's start symbol. Terminals $+$ and $*$ should be considered as representatives of a wider scale of binary operators. V and C are lexical nonterminals. Terminal productions

of V are the elements of a (finite) type VAR; if needed, a production tree issued from V is denoted $[V \to v, \langle\rangle]$, where v is a constant of type VAR. Likewise, C produces elements of a type CON and a production tree issued from C is denoted $[C \to c, \langle\rangle]$.

3 Expression trees

Expression trees (implicit: corresponding to CFG G above) are elements of type ET, as follows

$$\begin{array}{l}\mathbf{rec}\,(\,ET =_t \mathbf{sum}\,(\,plus : \mathbf{prod}(ET, ET)\ ,\\ \qquad\qquad\qquad\quad times : \mathbf{prod}(ET, ET)\ ,\\ \qquad\qquad\qquad\quad var : VAR\ ,\\ \qquad\qquad\qquad\quad con : CON\ )\\ \quad)\end{array}$$

The following function $\mathcal{E}$ associates expression trees with the production trees of G; thus $\mathcal{E} : PT \to ET$:

$$\begin{array}{l}\mathbf{rec}\,(\,\mathcal{E} =_e \lambda\, d{:}PT\\ \qquad .(\mathbf{case}\ d{:}PT\ \mathbf{of}\\ \qquad\quad [Z \to E, \langle d_1\rangle]\ \mathbf{then}\ \mathcal{E}{\cdot}d_1\ ,\\ \qquad\quad [E \to (E + E), \langle d_1, d_2\rangle]\ \mathbf{then}\ [plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2\rangle]\ ,\\ \qquad\quad [E \to (E * E), \langle d_1, d_2\rangle]\ \mathbf{then}\ [times, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2\rangle]\ ,\\ \qquad\quad [E \to V, \langle d_1\rangle]\ \mathbf{then}\ \mathcal{E}{\cdot}d_1\ ,\\ \qquad\quad [E \to C, \langle d_1\rangle]\ \mathbf{then}\ \mathcal{E}{\cdot}d_1\ ,\\ \qquad\quad [V \to v, \langle\rangle]\ \mathbf{then}\ [var, v]\ ,\\ \qquad\quad [C \to c, \langle\rangle]\ \mathbf{then}\ [con, c]\\ \qquad)\qquad)\end{array}$$

The last two alternatives in the case-expression actually represent the classes of production trees issued from V and C, respectively. Notice that $\mathcal{E}$ shortcuts chain productions appearing in a production tree.

Example: We consider the production tree for the expression $(a*b)+(a*b)$ and its image under $\mathcal{E}$, viz.

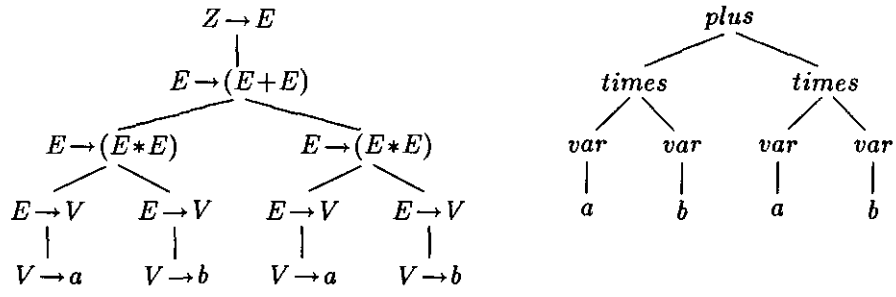
$$\begin{array}{l}[Z \to E\,,\langle\,[E \to (E + E)\,,\langle\,[E \to (E * E)\,,\langle\,[E \to V\,,\langle\,[V \to a\,,\langle\rangle]\,\rangle\,]\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ,\,[E \to V\,,\langle\,[V \to b\,,\langle\rangle]\,\rangle\,]\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \rangle\,]\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ,\,[E \to (E * E)\,,\langle\,[E \to V\,,\langle\,[V \to a\,,\langle\rangle]\,\rangle\,]\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ,\,[E \to V\,,\langle\,[V \to b\,,\langle\rangle]\,\rangle\,]\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \rangle\,]\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \rangle\\ ]\qquad\qquad \rangle\,]\end{array}$$

and

$$
\begin{array}{l}
[plus\,,\,\langle\,[times\,,\,\langle\,[var,a]\,,\,[var,b]\,\rangle\,] \\
\qquad\quad ,\,[times\,,\,\langle\,[var,a]\,,\,[var,b]\,\rangle\,] \\
\qquad\quad \rangle \\
]
\end{array}
$$

respectively.

As such expressions tend to get unwieldy very quickly, it is more appropriate to use a graphical representation for them, for instance



□

4 Representing expression trees by pointer-like structures

An expression tree (i.e., an object of type ET) can be represented conveniently by a pointer-like structure. This representation also allows of the identification of identical sub-trees. Because of the tree-like structure of the objects to be represented, no cycles are needed in the pointer representation. With this in mind, consider the definition of the type PN of the, so-called, *pointer nodes* and an acyclicity condition on a sequence of such nodes:

$$
\begin{array}{l}
PN =_t \mathbf{sum}\,(\; plus' : \mathbf{prod}(Nat, Nat)\;\;, \\
\qquad\qquad\quad times' : \mathbf{prod}(Nat, Nat)\;\;, \\
\qquad\qquad\quad var' : VAR\;\;, \\
\qquad\qquad\quad con' : CON\;\;)
\end{array}
$$

and the function $acyc : Seq(PN) \to bool$, with

$$
\begin{array}{l}
acyc =_e \lambda\, s : Seq(PN) \\
\qquad .\,(\bigwedge k \,|\, 1 \le k \le \#{\cdot}s \\
\qquad\qquad |\,\mathbf{case}\;\pi_k.s : PN\;\mathbf{of} \\
\qquad\qquad\qquad [plus'\,,\,\langle n1, n2\rangle]\;\mathbf{then}\;(1 \le n1 < k \;\wedge\; 1 \le n2 < k)\;\;, \\
\qquad\qquad\qquad [times'\,,\,\langle n1, n2\rangle]\;\mathbf{then}\;(1 \le n1 < k \;\wedge\; 1 \le n2 < k)\;\;, \\
\qquad\qquad\qquad [var'\,,\,v]\;\mathbf{then}\;true\;\;, \\
\qquad\qquad\qquad [con'\,,\,c]\;\mathbf{then}\;true \\
\qquad\qquad)
\end{array}
$$

If for a sequence $s : Seq(PN)$ of pointer nodes $acyc{\cdot}s$ holds, then we simply say that s is *acyclic*. Interpreting the numbers $n1$ and $n2$ in an element $[plus', \langle n1, n2\rangle]$ or $[times', \langle n1, n2\rangle]$ as references to positions in s, clearly the acyclicity condition expresses that only "backward" references should be made. In the sequel, only acyclic sequences of pointer nodes are considered, as these are the only ones of interest to us.

An acyclic sequence s can be employed to represent a collection of expression trees. More precisely, an expression tree is represented by a pair $\langle s, n\rangle$ of type $\mathbf{prod}(Seq(PN), Nat)$, according to the following representation function[3] $\mathcal{R} : Seq(PN) \rightarrow Nat \rightarrow ET$

For $s : Seq(PN)$ satisfying $acyc{\cdot}s$ and $n : Nat$ with $1 \leq n \leq \#{\cdot}s$

$$\begin{aligned}\mathcal{R}{\cdot}s{\cdot}n \;=_e\; & \mathbf{case}\ \pi_n.s : PN\ \mathbf{of}\\ & [plus', \langle n1, n2\rangle]\ \mathbf{then}\ [plus, \langle \mathcal{R}{\cdot}s{\cdot}n1, \mathcal{R}{\cdot}s{\cdot}n2\rangle]\ ,\\ & [times', \langle n1, n2\rangle]\ \mathbf{then}\ [times, \langle \mathcal{R}{\cdot}s{\cdot}n1, \mathcal{R}{\cdot}s{\cdot}n2\rangle]\ ,\\ & [var', v]\ \mathbf{then}\ [var, v]\ ,\\ & [con', c]\ \mathbf{then}\ [con, c]\end{aligned}$$

Example: Letting s denote the following sequence of pointer nodes

$$\langle [var', a], [var', b], [times', \langle 1, 2\rangle], [var', b], [var', a], [times', \langle 5, 4\rangle], [plus', \langle 3, 6\rangle]\rangle$$

s is acyclic, $\#{\cdot}s = 7$, and $\mathcal{R}{\cdot}s{\cdot}7$ is the expression tree corresponding to expression $(a{*}b){+}(a{*}b)$. □

An acyclic sequence s of pointer nodes thus represents $\#{\cdot}s$ expression trees. We take interest in those sequences that satisfy a uniqueness-property, viz. that no tree is represented more than once. This is expressed by the predicate $uniq : Seq(PN) \rightarrow bool$:

For $s : Seq(PN)$ satisfying $acyc{\cdot}s$

$$\begin{aligned}uniq{\cdot}s \;=_e\; & \forall n1, n2 : Nat\\ & .\ (1 \leq n1 \leq \#{\cdot}s \wedge 1 \leq n2 \leq \#{\cdot}s \wedge n1 \neq n2 \Rightarrow \mathcal{R}{\cdot}s{\cdot}n1 \neq \mathcal{R}{\cdot}s{\cdot}n2)\end{aligned}$$

Notice that $uniq{\cdot}s$ expresses that $\mathcal{R}{\cdot}s$ is an injection.

5 Some theory of attribute domains (and the like)

In view of the AG to follow, it is worthwhile to consider some properties of the entities introduced up till now[4]. In these properties, universal quantification over all free variables is implicit, on the understanding that

$$\begin{aligned}& d, d1, d2 : PT && e, e1, e2 : ET\\ & s, s1, s2 : Seq(PN) && n, n1, n2 : Nat\end{aligned}$$

[3] Actually, this is a recursive definition, which should be denoted $\mathbf{rec}\,(\mathcal{R} =_e \lambda \ldots)$.

[4] Here, and in the sequel, $\sqsubseteq_{PT}$ and $\sqsubseteq_{ET}$ denote the standard orderings on the elements of the free types PT and ET, respectively, as they come with the construction of the types. $\sqsubseteq_{PT}$ (resp. $\sqsubseteq_{ET}$) is the reflexive and transitive closure of $\sqsubset_{PT}$ (resp. $\sqsubset_{ET}$) "is a direct subtree of"; it is a well-founded partial order on PT (resp. ET), and as such the basis for structural induction.

Additional demands on such variables are given separately for each of the properties.

Full proofs are not provided. We only remark that $P1, P2$ and $P3$ are fundamental; their proofs require induction on $\sqsubseteq_{PT}, \sqsubseteq_{ET}$ and $\#{\cdot}s$, respectively, together with case analysis on the possible forms of (production- or expression-) trees. For the rest, $P4$ follows from $P2$ and the uniqueness of s, $P5$ follows from $P4$ and monotonicity of $\mathcal{E}$ — i.e., $P1$ —, and $P6$ is implied by $P5$. Finally, $P7$ is an easy consequence of the definition of acyclicity.

$P1$. $\mathcal{E}$ is monotonic, i.e.,

$$d1 \sqsubseteq_{PT} d2 \;\Rightarrow\; \mathcal{E}{\cdot}d1 \sqsubseteq_{ET} \mathcal{E}{\cdot}d2$$

$P2$. If $acyc{\cdot}s$ and $1 \le n \le \#{\cdot}s$

$$\mathcal{R}{\cdot}s{\cdot}n = e \;\Rightarrow\; \forall e' : ET \;.\; (e' \sqsubseteq_{ET} e \;\Rightarrow\; \exists n' : Nat \,.\, (1 \le n' \le n \wedge \mathcal{R}{\cdot}s{\cdot}n' = e'))$$

$P3$. If $acyc{\cdot}s$

$$uniq{\cdot}s \;\Leftrightarrow\; \forall n1, n2 : Nat \;.\; (1 \le n1 \le \#{\cdot}s \wedge 1 \le n2 \le \#{\cdot}s \wedge n1 \ne n2 \;\Rightarrow\; \pi_{n1}{\cdot}s \ne \pi_{n2}{\cdot}s)$$

$P4$. If $acyc{\cdot}s \wedge uniq{\cdot}s$, $1 \le n1 \le \#{\cdot}s$ and $1 \le n2 \le \#{\cdot}s$

$$\mathcal{R}{\cdot}s{\cdot}n1 = e1 \wedge \mathcal{R}{\cdot}s{\cdot}n2 = e2 \wedge e1 \sqsubseteq_{ET} e2 \;\Rightarrow\; n1 \le n2$$

$P5$. Under the same conditions as $P4$

$$\mathcal{R}{\cdot}s{\cdot}n1 = \mathcal{E}{\cdot}d1 \wedge \mathcal{R}{\cdot}s{\cdot}n2 = \mathcal{E}{\cdot}d2 \wedge d1 \sqsubseteq_{PT} d2 \;\Rightarrow\; n1 \le n2$$

$P6$. If $acyc{\cdot}s \wedge uniq{\cdot}s$

$$\{k \mid 1 \le k \le \#{\cdot}s \mid \mathcal{R}{\cdot}s{\cdot}k\} = \{d' \mid d' \sqsubseteq_{PT} d \mid \mathcal{E}{\cdot}d'\} \;\Rightarrow\; \mathcal{R}{\cdot}s{\cdot}(\#{\cdot}s) = \mathcal{E}{\cdot}d$$

$P7$. For the determination of $\mathcal{R}{\cdot}s{\cdot}n$ the elements in s beyond index n play no role:

$$1 \le n1 \le \#{\cdot}s1 \wedge acyc{\cdot}(s1 +\!\!+ s2) \;\Rightarrow\; \mathcal{R}{\cdot}s1{\cdot}n1 = \mathcal{R}{\cdot}(s1 +\!\!+ s2){\cdot}n1$$

6 Attribute grammar

The CFG given earlier is now extended to a one-pass AG. The goal of the attribute structure is to produce, for each complete production tree d, a pointer representation of the expression tree $\mathcal{E}{\cdot}d$ corresponding to d, in such a way that common sub-trees are identified. This is expressed by the following specification for start symbol Z:

$Z(+s : Seq(PN))$

$R_Z{\cdot}d{\cdot}s \;:\; acyc{\cdot}s \wedge uniq{\cdot}s \wedge \{k \mid 1 \le k \le \#{\cdot}s \mid \mathcal{R}{\cdot}s{\cdot}k\} = \{d' \mid d' \sqsubseteq_{PT} d \mid \mathcal{E}{\cdot}d'\}$

The first two conjuncts in $R_Z{\cdot}d{\cdot}s$ need no explanation. Concerning the third one, observe that, by property $P2$, a representation of $\mathcal{E}{\cdot}d$ cannot be provided without giving representations of all of d's sub-trees (in the sense of $\sqsubseteq_{PT}$) also. Hence the set of expression trees represented in s must at least contain $\{d' \mid d' \sqsubseteq_{PT} d \mid \mathcal{E}{\cdot}d'\}$. Now the third conjunct in $R_Z{\cdot}d{\cdot}s$ additionally expresses that — conversely — nothing more is represented in s, which seems in every respect a reasonable demand. On account of $P6$, then, the required expression $\mathcal{E}{\cdot}d$ can be found as $\mathcal{R}{\cdot}s{\cdot}(\#{\cdot}s)$.

Taking the specification for Z as a starting-point, a purely synthesized AG is obtained by providing nonterminal E with the same attribute and specification as Z. However, the synthesis of s_0 from s_1 and s_2 — in each of the productions $E \rightarrow (E + E)$ and $E \rightarrow (E * E)$ — would bring about operations of an unacceptable complexity, in order to establish the uniqueness of s_0.

A much more attractive solution is obtained by a bucket brigade-like attribute structure. Herein, a sequence s is passed around the nodes of the production tree in a depth-first fashion, adding representations of the sub-trees in question whenever appropriate. Upon detecting the existence of a sub-tree's representation, however, the sequence is left unaltered and an index in the sequence is produced, indicating where the representation of that sub-tree is to be found.

The following choice of a specification for E seems to be in support of this idea:

$$
\begin{array}{lll}
E\langle -si : Seq(PN), +ss : Seq(PN), +n : Nat\rangle & & \\
Q_E{\cdot}d{\cdot}si & : & acyc{\cdot}si \wedge uniq{\cdot}si \\
R_E{\cdot}d{\cdot}si{\cdot}ss{\cdot}n & : & acyc{\cdot}ss \wedge uniq{\cdot}ss \wedge 1 \leq n \leq \#{\cdot}ss \wedge \mathcal{R}{\cdot}ss{\cdot}n = \mathcal{E}{\cdot}d \wedge \\
 & & \{k \mid 1 \leq k \leq \#{\cdot}ss \mid \mathcal{R}{\cdot}ss{\cdot}k\} = \{k \mid 1 \leq k \leq \#{\cdot}si \mid \mathcal{R}{\cdot}si{\cdot}k\} \\
 & & \qquad \cup \{d' \mid d' \sqsubseteq_{PT} d \mid \mathcal{E}{\cdot}d'\}
\end{array}
$$

Note: This specification for E can be regarded as a typical bucket brigade-like generalisation of Z's specification; namely, the latter could also be written

$$
\begin{array}{lll}
Z\langle +s : Seq(PN), +n : Nat\rangle & & \\
R_Z{\cdot}d{\cdot}s{\cdot}n & : & acyc{\cdot}s \wedge uniq{\cdot}s \wedge 1 \leq n \leq \#{\cdot}s \wedge \mathcal{R}{\cdot}s{\cdot}n = \mathcal{E}{\cdot}d \wedge \\
 & & \{k \mid 1 \leq k \leq \#{\cdot}s \mid \mathcal{R}{\cdot}s{\cdot}k\} = \{d' \mid d' \sqsubseteq_{PT} d \mid \mathcal{E}{\cdot}d'\}
\end{array}
$$

As indicated before, however, the last conjunct implies $n = \#{\cdot}s$, on account of which the n-attribute can be left out.

□

However acceptable E's specification — as a generalisation of the one for Z — seems to be, unfortunately it turns out not to be sufficient. Imposing the obvious bucket brigade-like evaluation rules to either of the productions $E \rightarrow (E+E)$ or $E \rightarrow (E*E)$, the proof obligations cannot be met. Technically, this is caused by the fact that R_E is too weak; the remaining gap in the proof gives a good indication where to search for improvement. The phenomenon has an operational interpretation also; we shall discuss it on the basis of production $E \rightarrow (E + E)$:

$$
\begin{array}{l}
E\langle -si_0, +ss_0, +n_0\rangle \;\rightarrow\; (E\langle -si_1, +ss_1, +n_1\rangle + E\langle -si_2, +ss_2, +n_2\rangle) \\
si_1 = si_0 \;,\; si_2 = ss_1 \;,\; ss_0 = \ldots \;,\; n_0 = \ldots
\end{array}
$$

Evaluation rule $si_2 = ss_1$ shows that the result-sequence emanating from the left sub-tree is passed to the right sub-tree. Next the result ss_2 of the latter should be used to determine ss_0 and n_0, where n_1 and n_2 are to be employed as the indices in ss_2 at which $\mathcal{E}{\cdot}d_1$ and $\mathcal{E}{\cdot}d_2$ can be found. Now from R_2 we indeed have

$$\mathcal{R}{\cdot}ss_2{\cdot}n_2 = \mathcal{E}{\cdot}d_2$$

whereas R_1 expresses

$$\mathcal{R}{\cdot}ss_1{\cdot}n_1 = \mathcal{E}{\cdot}d_1$$

That is where the snag is: as ss_0 is composed on the basis of ss_2, we would like to be able to assert $\mathcal{R}{\cdot}ss_2{\cdot}n_1 = \mathcal{E}{\cdot}d_1$ instead. On account of property $P7$, this is allowed if ss_2 were an extension of ss_1. The latter statement, in itself, concerns attributes of different nonterminals and hence cannot be expressed in a specification. Fortunately we have the equation $si_2 = ss_1$, so the same effect can be obtained by asserting that ss_2 extends si_2. In general, we should add to R_E the conjunct

$$\exists sa : Seq(PN) \,.\, ss = si +\!\!+ sa$$

As far as the passing of result sequences is concerned, the evaluation rules to be added to the various productions directly follow the bucket brigade idea. In addition, property $P3$ is employed to prevent the addition of an element to such sequences from disturbing the uniqueness (this plays a role in productions 2 through 5). All in all, we arrive at the following

attribute grammar:

Specifications:

$Z\langle +s : Seq(PN)\rangle$

$R_Z{\cdot}d{\cdot}s \;:\; acyc{\cdot}s \;\wedge\; uniq{\cdot}s \;\wedge\; \{k \,|\, 1 \le k \le \#{\cdot}s \,|\, \mathcal{R}{\cdot}s{\cdot}k\} = \{d' \,|\, d' \sqsubseteq_{PT} d \,|\, \mathcal{E}{\cdot}d'\}$

$E\langle -si : Seq(PN), +ss : Seq(PN), +n : Nat\rangle$

$Q_E{\cdot}d{\cdot}si \;:\; acyc{\cdot}si \;\wedge\; uniq{\cdot}si$

$R_E{\cdot}d{\cdot}si{\cdot}ss{\cdot}n \;:\; acyc{\cdot}ss \;\wedge\; uniq{\cdot}ss \;\wedge\; 1 \le n \le \#{\cdot}ss \;\wedge\; \mathcal{R}{\cdot}ss{\cdot}n = \mathcal{E}{\cdot}d \;\wedge\; \exists sa : Seq(PN) \,.\, ss = si +\!\!+ sa \;\wedge\; \{k \,|\, 1 \le k \le \#{\cdot}ss \,|\, \mathcal{R}{\cdot}ss{\cdot}k\} = \{k \,|\, 1 \le k \le \#{\cdot}si \,|\, \mathcal{R}{\cdot}si{\cdot}k\} \cup \{d' \,|\, d' \sqsubseteq_{PT} d \,|\, \mathcal{E}{\cdot}d'\}$

$V\langle +v : VAR\rangle$

$R_V{\cdot}d{\cdot}v \;:\; v = yield{\cdot}d$

$C\langle +c : CON\rangle$

$R_C{\cdot}d{\cdot}c \;:\; c = yield{\cdot}d$

Production rules:

1. $Z\langle +s_0\rangle \;\rightarrow\; E\langle -si_1, +ss_1, +n_1\rangle$
   $si_1 = \langle\rangle \;,\; s_0 = ss_1$

2. $E\langle -si_0, +ss_0, +n_0\rangle \;\rightarrow\; (E\langle -si_1, +ss_1, +n_1\rangle + E\langle -si_2, +ss_2, +n_2\rangle)$

$si_1 = si_0$, $si_2 = ss_1$,
$([plus', \langle n_1, n_2\rangle] \in rng{\cdot}ss_2$
$\quad \rightarrow\ ss_0 = ss_2$, $n_0 = (\iota\, n \,|\, 1 \leq n \leq \#{\cdot}ss_2 \wedge \pi_n.ss_2 = [plus', \langle n_1, n_2\rangle] \,|\, n)$
$[]\, [plus', \langle n_1, n_2\rangle] \notin rng{\cdot}ss_2$
$\quad \rightarrow\ ss_0 = ss_2 +\!\!+ \langle [plus', \langle n_1, n_2\rangle]\rangle$, $n_0 = \#{\cdot}ss_2 + 1$
)

3. analogous to 2., replacing $plus'$ by $times'$

4. $E\langle -si_0, +ss_0, +n_0\rangle \rightarrow V\langle +v_1\rangle$
$([var', v_1] \in rng{\cdot}si_0 \rightarrow ss_0 = si_0$,
$\qquad\qquad n_0 = (\iota\, n \,|\, 1 \leq n \leq \#{\cdot}si_0 \wedge \pi_n.si_0 = [var', v_1] \,|\, n)$
$[]\, [var', v_1] \notin rng{\cdot}si_0 \rightarrow ss_0 = si_0 +\!\!+ \langle [var', v_1]\rangle$, $n_0 = \#{\cdot}si_0 + 1$
)

5. analogous to 4.

□

7 Correctness proofs

Using the proof rules for one-pass AGs, we now show that the AG above meets its specification.

The remaining proof obligations, resulting from our proof system, are organised per production. This section reflects that organisation by devoting one subsection to each production[5].

Then again, the correctness proof for an individual production p requires $\#p{+}1$ implications to be proven. Within each subsection, the proof parts corresponding to these implications are numbered consecutively, and a hierarchical numbering is employed to further subdivide each proof part. For example, if (2) denotes the proof part corresponding to the second implication, then (2.1) and (2.2) denote two sub-parts of this proof, etc. Such subdivisions are based on a couple of simple logical rules, e.g. that a conjunction of a number of terms can be proven by proving each conjunct separately.

In order to save writing, throughout this section we adopt the following

Notational convention: In connection with the attributes of nonterminal E, we use SS, SI and D to denote the following sets:

$$SS = \{k \,|\, 1 \leq k \leq \#{\cdot}ss \,|\, \mathcal{R}{\cdot}ss{\cdot}k\}$$
$$SI = \{k \,|\, 1 \leq k \leq \#{\cdot}si \,|\, \mathcal{R}{\cdot}si{\cdot}k\}$$
$$D = \{d' \,|\, d' \sqsubseteq_{PT} d \,|\, \mathcal{E}{\cdot}d'\}$$

and this notation carries over to the subscripted forms in the obvious manner, e.g. SS_0, SS_1.
□

7.1 Ad production 1

Proof obligation:

[5] Because of the resemblance between productions 2/3 and 4/5, only productions 1, 2 and 4 will be treated.

$$
\begin{array}{l}
\quad s_0, si_1, ss_1 : Seq(PN)\ ,\ n_1 : Nat\ ,\ d_1 : PT_E\ ,\ d_0 : PT_Z\ ,\ d_0 =_e [Z \to E, \langle d_1 \rangle]\ , \\
\quad si_1 = \langle\rangle\ ,\ s_0 = ss_1 \\
\rhd \\
\quad true \ \Rightarrow\ acyc{\cdot}si_1 \ \wedge\ uniq{\cdot}si_1 \qquad (1) \\
\wedge \\
\quad R_E{\cdot}d_1{\cdot}si_1{\cdot}ss_1{\cdot}n_1 \\
\quad \Rightarrow \\
\quad acyc{\cdot}s_0 \ \wedge\ uniq{\cdot}s_0 \ \wedge\ \{k \,|\, 1 \le k \le \#{\cdot}s_0 \,|\, \mathcal{R}{\cdot}s_0{\cdot}k\} = \{d' \,|\, d' \sqsubseteq_{PT} d_0 \,|\, \mathcal{E}{\cdot}d'\} \qquad (2)
\end{array}
$$

(1): proof of $acyc{\cdot}si_1 \wedge uniq{\cdot}si_1$: trivial

(2.1): proof of $acyc{\cdot}s_0 \wedge uniq{\cdot}s_0$: trivial

(2.2): proof of $\{k \,|\, 1 \le k \le \#{\cdot}s_0 \,|\, \mathcal{R}{\cdot}s_0{\cdot}k\} = \{d' \,|\, d' \sqsubseteq_{PT} d_0 \,|\, \mathcal{E}{\cdot}d'\}$:

$$
\begin{array}{ll}
 & \{k \,|\, 1 \le k \le \#{\cdot}s_0 \,|\, \mathcal{R}{\cdot}s_0{\cdot}k\} \\
= & \quad \{\!|\ s_0 = ss_1\ |\!\} \\
 & \{k \,|\, 1 \le k \le \#{\cdot}ss_1 \,|\, \mathcal{R}{\cdot}ss_1{\cdot}k\} \\
= & \quad \{\!|\ R_E{\cdot}d_1{\cdot}si_1{\cdot}ss_1{\cdot}n_1\ |\!\} \\
 & \{k \,|\, 1 \le k \le \#{\cdot}si_1 \,|\, \mathcal{R}{\cdot}si_1{\cdot}k\} \ \cup\ \{d' \,|\, d' \sqsubseteq_{PT} d_1 \,|\, \mathcal{E}{\cdot}d'\} \\
= & \quad \{\!|\ si_1 = \langle\rangle\ |\!\} \\
 & \{d' \,|\, d' \sqsubseteq_{PT} d_1 \,|\, \mathcal{E}{\cdot}d'\} \\
= & \quad \{\!|\ \text{def. } \mathcal{E}\ |\!\} \\
 & \{d' \,|\, d' \sqsubseteq_{PT} d_1 \,|\, \mathcal{E}{\cdot}d'\} \ \cup\ \{\mathcal{E}{\cdot}[Z \to E, \langle d_1 \rangle]\} \\
= & \quad \{\!|\ d_0 =_e [Z \to E, \langle d_1 \rangle]\ ,\ \text{def. } \sqsubseteq_{PT}\ |\!\} \\
 & \{d' \,|\, d' \sqsubseteq_{PT} d_0 \,|\, \mathcal{E}{\cdot}d'\}
\end{array}
$$

7.2 Ad production 2

Proof obligation:

$$
\begin{array}{l}
\quad si_0, ss_0, si_1, ss_1, si_2, ss_2 : Seq(PN)\ ,\ n_0, n_1, n_2 : Nat\ , \\
\quad d_0, d_1, d_2 : PT_E\ ,\ d_0 =_e [E \to (E + E), \langle d_1, d_2 \rangle]\ ,\ si_1 = si_0\ ,\ si_2 = ss_1\ , \\
\quad ([plus', \langle n_1, n_2 \rangle] \in rng{\cdot}ss_2 \ \to \ldots \ [\!]\ [plus', \langle n_1, n_2 \rangle] \notin rng{\cdot}ss_2 \ \to \ldots) \\
\rhd \\
\quad Q_E{\cdot}d_0{\cdot}si_0 \ \Rightarrow\ acyc{\cdot}si_1 \ \wedge\ uniq{\cdot}si_1 \qquad (1) \\
\wedge \\
\quad Q_E{\cdot}d_0{\cdot}si_0 \ \wedge\ R_E{\cdot}d_1{\cdot}si_1{\cdot}ss_1{\cdot}n_1 \ \Rightarrow\ acyc{\cdot}si_2 \ \wedge\ uniq{\cdot}si_2 \qquad (2) \\
\wedge \\
\quad Q_E{\cdot}d_0{\cdot}si_0 \ \wedge\ R_E{\cdot}d_1{\cdot}si_1{\cdot}ss_1{\cdot}n_1 \ \wedge\ R_E{\cdot}d_2{\cdot}si_2{\cdot}ss_2{\cdot}n_2 \\
\quad \Rightarrow \\
\quad acyc{\cdot}ss_0 \ \wedge\ uniq{\cdot}ss_0 \ \wedge\ 1 \le n_0 \le \#{\cdot}ss_0 \ \wedge\ \mathcal{R}{\cdot}ss_0{\cdot}n_0 = \mathcal{E}{\cdot}d_0 \ \wedge \qquad (3) \\
\quad SS_0 = SI_0 \cup D_0 \ \wedge\ \exists sa : Seq(PN)\,.\ ss_0 = si_0 +\!\!+ sa
\end{array}
$$

(1): proof of $acyc{\cdot}si_1 \wedge uniq{\cdot}si_1$: trivial

(2): proof of $acyc{\cdot}si_2 \wedge uniq{\cdot}si_2$: trivial

(3): the proof follows the case-analysis used to define synthesized attributes ss_0 and n_0. We consider the cases in turn

(3.1): first case, i.e., $[plus', \langle n_1, n_2\rangle] \in rng{\cdot}ss_2$

(3.1.1): $acyc{\cdot}ss_0$: immediately from $ss_0 = ss_2 \wedge acyc{\cdot}ss_2$

(3.1.2): $uniq{\cdot}ss_0$: immediately from $ss_0 = ss_2 \wedge uniq{\cdot}ss_2$

(3.1.3): $1 \leq n_0 \leq \#{\cdot}ss_0$: immediately from $ss_0 = ss_2 \wedge n_0 = (\iota\, n \mid \ldots \mid n)$

(3.1.4): proof of $\mathcal{R}{\cdot}ss_0{\cdot}n_0 = \mathcal{E}{\cdot}d_0$:

$$
\begin{array}{ll}
 & \mathcal{R}{\cdot}ss_0{\cdot}n_0 \\
= & \quad \{\!\!\{\, ss_0 = ss_2 \ ,\ n_0 = (\iota\, n \mid \ldots \mid n) \,\}\!\!\} \\
 & \mathcal{R}{\cdot}ss_2{\cdot}(\iota\, n \mid 1 \leq n \leq \#{\cdot}ss_2 \wedge \pi_n.ss_2 = [plus', \langle n_1, n_2\rangle] \mid n) \\
= & \quad \{\!\!\{ \text{def. } \mathcal{R} \ ,\ acyc{\cdot}ss_2 - \text{hence } 1 \leq n_1 < n \wedge 1 \leq n_2 < n \,\}\!\!\} \\
 & [plus, \langle \mathcal{R}{\cdot}ss_2{\cdot}n_1, \mathcal{R}{\cdot}ss_2{\cdot}n_2\rangle] \\
= & \quad \{\!\!\{ \exists sa : Seq(PN)\,.\, ss_2 = si_2 +\!\!+ sa - \text{say } sa_2 \,\}\!\!\} \\
 & [plus, \langle \mathcal{R}{\cdot}(si_2 +\!\!+ sa_2){\cdot}n_1, \mathcal{R}{\cdot}ss_2{\cdot}n_2\rangle] \\
= & \quad \{\!\!\{\, si_2 = ss_1 \,\}\!\!\} \\
 & [plus, \langle \mathcal{R}{\cdot}(ss_1 +\!\!+ sa_2){\cdot}n_1, \mathcal{R}{\cdot}ss_2{\cdot}n_2\rangle] \\
= & \quad \{\!\!\{\, 1 \leq n_1 \leq \#{\cdot}ss_1 \ ,\ acyc{\cdot}(ss_1 +\!\!+ sa_2) \ ,\ \text{prop. } P7 \,\}\!\!\} \\
 & [plus, \langle \mathcal{R}{\cdot}ss_1{\cdot}n_1, \mathcal{R}{\cdot}ss_2{\cdot}n_2\rangle] \\
= & \quad \{\!\!\{\, \mathcal{R}{\cdot}ss_1{\cdot}n_1 = \mathcal{E}{\cdot}d_1 \ ,\ \mathcal{R}{\cdot}ss_2{\cdot}n_2 = \mathcal{E}{\cdot}d_2 \,\}\!\!\} \\
 & [plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2\rangle] \\
= & \quad \{\!\!\{ \text{def. } \mathcal{E} \ ,\ d_0 =_e [E \rightarrow (E + E), \langle d_1, d_2\rangle] \,\}\!\!\} \\
 & \mathcal{E}{\cdot}d_0
\end{array}
$$

(3.1.5): proof of $SS_0 = SI_0 \cup D_0$:

$$
\begin{array}{ll}
 & SS_0 \\
= & \quad \{\!\!\{\, ss_0 = ss_2 \,\}\!\!\} \\
 & SS_2 \\
= & \quad \{\!\!\{\, [plus', \langle n_1, n_2\rangle] \in rng{\cdot}ss_2 \,\}\!\!\} \\
 & SS_2 \ \cup\ \{\mathcal{R}{\cdot}ss_2{\cdot}(\iota\, n \mid 1 \leq n \leq \#{\cdot}ss_2 \wedge \pi_n.ss_2 = [plus', \langle n_1, n_2\rangle] \mid n)\} \\
= & \quad \{\!\!\{ \text{ proof of (3.1.4)} \,\}\!\!\} \\
 & SS_2 \ \cup\ \{\mathcal{E}{\cdot}d_0\} \\
= & \quad \{\!\!\{\, SS_2 = SI_2 \cup D_2 \ ,\ \cup \text{ associative} \,\}\!\!\} \\
 & SI_2 \ \cup\ D_2 \ \cup\ \{\mathcal{E}{\cdot}d_0\} \\
= & \quad \{\!\!\{\, si_2 = ss_1 \,\}\!\!\} \\
 & SS_1 \ \cup\ D_2 \ \cup\ \{\mathcal{E}{\cdot}d_0\} \\
= & \quad \{\!\!\{\, SS_1 = SI_1 \cup D_1 \,\}\!\!\} \\
 & SI_1 \ \cup\ D_1 \ \cup\ D_2 \ \cup\ \{\mathcal{E}{\cdot}d_0\} \\
= & \quad \{\!\!\{\, si_1 = si_0 \ ,\ d_0 =_e [E \rightarrow (E + E), \langle d_1, d_2\rangle] \ ,\ \text{def. } \sqsubseteq_{PT} \,\}\!\!\} \\
 & SI_0 \ \cup\ D_0
\end{array}
$$

(3.1.6): proof of $\exists sa : Seq(PN) \,.\, ss_0 = si_0 +\!\!+ sa$:

ss_0

$=$ $\quad \langle\!\!\langle\, ss_0 = ss_2 \,\rangle\!\!\rangle$

ss_2

$=$ $\quad \langle\!\!\langle\, \exists sa : Seq(PN) \,.\, ss_2 = si_2 +\!\!+ sa$ – say $sa_2 \,\rangle\!\!\rangle$

$si_2 +\!\!+ sa_2$

$=$ $\quad \langle\!\!\langle\, si_2 = ss_1 \,\rangle\!\!\rangle$

$ss_1 +\!\!+ sa_2$

$=$ $\quad \langle\!\!\langle\, \exists sa : Seq(PN) \,.\, ss_1 = si_1 +\!\!+ sa$ – say sa_1 , $+\!\!+$ associative $\rangle\!\!\rangle$

$si_1 +\!\!+ sa_1 +\!\!+ sa_2$

$=$ $\quad \langle\!\!\langle\, si_1 = si_0 \,\rangle\!\!\rangle$

$si_0 +\!\!+ sa_1 +\!\!+ sa_2$

hence $\exists sa : Seq(PN) \,.\, ss_0 = si_0 +\!\!+ sa$

(3.2): second case, i.e., $[plus', \langle n_1, n_2 \rangle] \notin rng{\cdot}ss_2$

(3.2.1): proof of $acyc{\cdot}ss_0$:

$acyc{\cdot}ss_0$

$=$ $\quad \langle\!\!\langle\, ss_0 = ss_2 +\!\!+ \langle [plus', \langle n_1, n_2 \rangle] \rangle \,\rangle\!\!\rangle$

$acyc{\cdot}(ss_2 +\!\!+ \langle [plus', \langle n_1, n_2 \rangle] \rangle)$

$=$ $\quad \langle\!\!\langle$ def. $acyc$, $acyc{\cdot}ss_2 \,\rangle\!\!\rangle$

$1 \le n_1 \le \#{\cdot}ss_2 \;\wedge\; 1 \le n_2 \le \#{\cdot}ss_2$

$=$ $\quad \langle\!\!\langle\, \exists sa : Seq(PN) \,.\, ss_2 = si_2 +\!\!+ sa$ – say $sa_2 \,\rangle\!\!\rangle$

$1 \le n_1 \le \#{\cdot}(si_2 +\!\!+ sa_2) \;\wedge\; 1 \le n_2 \le \#{\cdot}ss_2$

$=$ $\quad \langle\!\!\langle\, si_2 = ss_1 \,\rangle\!\!\rangle$

$1 \le n_1 \le \#{\cdot}(ss_1 +\!\!+ sa_2) \;\wedge\; 1 \le n_2 \le \#{\cdot}ss_2$

$\Leftarrow$ $\quad \langle\!\!\langle\, \rangle\!\!\rangle$

$1 \le n_1 \le \#{\cdot}ss_1 \;\wedge\; 1 \le n_2 \le \#{\cdot}ss_2$

$=$ $\quad \langle\!\!\langle\, R_1 \;,\; R_2 \,\rangle\!\!\rangle$

$true$

(3.2.2): proof of $uniq{\cdot}ss_0$:

$uniq{\cdot}ss_0$

$=$ $\quad \langle\!\!\langle\, ss_0 = ss_2 +\!\!+ \langle [plus', \langle n_1, n_2 \rangle] \rangle \,\rangle\!\!\rangle$

$uniq{\cdot}(ss_2 +\!\!+ \langle [plus', \langle n_1, n_2 \rangle] \rangle)$

$=$ $\quad \langle\!\!\langle$ prop. $P3$, $uniq{\cdot}ss_2 \,\rangle\!\!\rangle$

$\forall n : Nat \,.\, (1 \le n \le \#{\cdot}ss_2 \;\Rightarrow\; \pi_n{\cdot}ss_2 \neq [plus', \langle n_1, n_2 \rangle])$

$=$ $\quad \langle\!\!\langle\, [plus', \langle n_1, n_2 \rangle] \notin rng{\cdot}ss_2 \,\rangle\!\!\rangle$

$true$

(3.2.3): $1 \le n_0 \le \#{\cdot}ss_0$: trivial

(3.2.4): proof of $\mathcal{R}{\cdot}ss_0{\cdot}n_0 = \mathcal{E}{\cdot}d_0$:

$\mathcal{R}{\cdot}ss_0{\cdot}n_0$

$=$ $\quad \langle\!\!\langle\, ss_0 = ss_2 +\!\!+ \langle [plus', \langle n_1, n_2 \rangle] \rangle \;,\; n_0 = \#{\cdot}ss_2 + 1 \,\rangle\!\!\rangle$

$$
\begin{array}{ll}
 & \mathcal{R}\cdot(ss_2 +\!\!+ \langle[plus', \langle n_1, n_2\rangle]\rangle)\cdot(\#\cdot ss_2 + 1) \\
= & \quad \{\!\!\{ \text{ def. } \mathcal{R} \text{ , } 1 \le n_1 \le \#\cdot ss_2 \text{ , } 1 \le n_2 \le \#\cdot ss_2 \text{ , prop. } P7 \}\!\!\} \\
 & [plus, \langle \mathcal{R}\cdot ss_2\cdot n_1, \mathcal{R}\cdot ss_2\cdot n_2\rangle] \\
= & \quad \{\!\!\{ \text{ proof of (3.1.4) } \}\!\!\} \\
 & \mathcal{E}\cdot d_0
\end{array}
$$

(3.2.5): proof of $SS_0 = SI_0 \cup D_0$:

$$
\begin{array}{ll}
 & SS_0 \\
= & \quad \{\!\!\{ ss_0 = ss_2 +\!\!+ \langle[plus', \langle n_1, n_2\rangle]\rangle \}\!\!\} \\
 & \{k \mid 1 \le k \le \#\cdot ss_2 + 1 \mid \mathcal{R}\cdot(ss_2 +\!\!+ \langle[plus', \langle n_1, n_2\rangle]\rangle)\cdot k\} \\
= & \quad \{\!\!\{ \text{ def. } SS \text{ , prop. } P7 \}\!\!\} \\
 & SS_2 \ \cup \ \{\mathcal{R}\cdot(ss_2 +\!\!+ \langle[plus', \langle n_1, n_2\rangle]\rangle)\cdot(\#\cdot ss_2 + 1)\} \\
= & \quad \{\!\!\{ \text{ proof of (3.2.4) } \}\!\!\} \\
 & SS_2 \ \cup \ \{\mathcal{E}\cdot d_0\} \\
= & \quad \{\!\!\{ \text{ proof of (3.1.5) } \}\!\!\} \\
 & SI_0 \ \cup \ D_0
\end{array}
$$

(3.2.6): proof of $\exists sa : Seq(PN) \,.\, ss_0 = si_0 +\!\!+ sa$:
completely analogous to the proof of (3.1.6); in fact, in the terminology of (3.1.6),

$$ss_0 = si_0 +\!\!+ sa_1 +\!\!+ sa_2 +\!\!+ \langle[plus', \langle n_1, n_2\rangle]\rangle$$

7.3 Ad production 4

Proof obligation:

$$
\begin{array}{l}
si_0, ss_0 : Seq(PN) \text{ , } n_0 : Nat \text{ , } v_1 : VAR \text{ , } d_1 : PT_V \text{ , } d_0 : PT_E \text{ , } \\
d_0 =_e [E \rightarrow V, \langle d_1\rangle] \text{ , } ([var', v_1] \in rng\cdot si_0 \ \rightarrow \ldots \ [\!] \ [var', v_1] \notin rng\cdot si_0 \ \rightarrow \ldots) \\
\rhd \\
Q_E\cdot d_0\cdot si_0 \ \wedge \ R_V\cdot d_1\cdot v_1 \\
\Rightarrow \\
acyc\cdot ss_0 \ \wedge \ uniq\cdot ss_0 \ \wedge \ 1 \le n_0 \le \#\cdot ss_0 \ \wedge \ \mathcal{R}\cdot ss_0\cdot n_0 = \mathcal{E}\cdot d_0 \ \wedge \qquad (1) \\
SS_0 = SI_0 \cup D_0 \ \wedge \ \exists sa : Seq(PN) \,.\, ss_0 = si_0 +\!\!+ sa
\end{array}
$$

(1): the proof follows the case-analysis used to define synthesized attributes ss_0 and n_0. We consider the cases in turn

(1.1): first case, i.e., $[var', v_1] \in rng\cdot si_0$

(1.1.1): $acyc\cdot ss_0$: immediately from $ss_0 = si_0 \ \wedge \ acyc\cdot si_0$

(1.1.2): $uniq\cdot ss_0$: immediately from $ss_0 = si_0 \ \wedge \ uniq\cdot si_0$

(1.1.3): $1 \le n_0 \le \#\cdot ss_0$: immediately from $ss_0 = si_0 \ \wedge \ n_0 = (\iota\, n \mid \ldots \mid n)$

(1.1.4): proof of $\mathcal{R}\cdot ss_0\cdot n_0 = \mathcal{E}\cdot d_0$:

$$
\begin{array}{ll}
 & \mathcal{R}{\cdot}ss_0{\cdot}n_0 \\
= & \qquad \langle\!\langle\, ss_0 = si_0\ ,\ n_0 = (\iota\, n \mid \ldots \mid n) \,\rangle\!\rangle \\
 & \mathcal{R}{\cdot}si_0{\cdot}(\iota\, n \mid 1 \leq n \leq \#{\cdot}si_0 \wedge \pi_n.si_0 = [var', v_1] \mid n) \\
= & \qquad \langle\!\langle\, \text{def. } \mathcal{R} \,\rangle\!\rangle \\
 & [var, v_1] \\
= & \qquad \langle\!\langle\, v_1 = yield{\cdot}d_1 \,\rangle\!\rangle \\
 & [var, yield{\cdot}d_1] \\
= & \qquad \langle\!\langle\, \text{def. } \mathcal{E} \,\rangle\!\rangle \\
 & \mathcal{E}{\cdot}[V \rightarrow yield{\cdot}d_1, \langle\rangle] \\
= & \qquad \langle\!\langle\, d_1 : PT_V\ ,\ \text{def. } yield \,\rangle\!\rangle \\
 & \mathcal{E}{\cdot}d_1 \\
= & \qquad \langle\!\langle\, \text{def. } \mathcal{E}\ ,\ d_0 =_e [E \rightarrow V, \langle d_1 \rangle] \,\rangle\!\rangle \\
 & \mathcal{E}{\cdot}d_0
\end{array}
$$

(1.1.5): proof of $SS_0 = SI_0 \cup D_0$:

$$
\begin{array}{ll}
 & SS_0 \\
= & \qquad \langle\!\langle\, ss_0 = si_0 \,\rangle\!\rangle \\
 & SI_0 \\
= & \qquad \langle\!\langle\, [var', v_1] \in rng{\cdot}si_0 \,\rangle\!\rangle \\
 & SI_0 \ \cup\ \{\mathcal{R}{\cdot}si_0{\cdot}(\iota\, n \mid 1 \leq n \leq \#{\cdot}si_0 \wedge \pi_n.si_0 = [var', v_1 \mid n)\} \\
= & \qquad \langle\!\langle\, \text{proof of (1.1.4)} \,\rangle\!\rangle \\
 & SI_0 \ \cup\ \{\mathcal{E}{\cdot}d_0\} \\
= & \qquad \langle\!\langle\, \text{def. } \mathcal{E}\ ,\ d_0 =_e [E \rightarrow V, \langle d_1 \rangle] \,\rangle\!\rangle \\
 & SI_0 \ \cup\ \{\mathcal{E}{\cdot}d_0\} \ \cup\ \{\mathcal{E}{\cdot}d_1\} \\
= & \qquad \langle\!\langle\, d_1 : PT_V\ ,\ \text{def. } \sqsubseteq_{PT}\ ,\ d_0 =_e [E \rightarrow V, \langle d_1 \rangle] \,\rangle\!\rangle \\
 & SI_0 \ \cup\ D_0
\end{array}
$$

(1.1.6): $\exists sa : Seq(PN)\,.\, ss_0 = si_0 +\!\!+ sa$: trivial

(1.2): second case, i.e., $[var', v_1] \notin rng{\cdot}si_0$

(1.2.1): $acyc{\cdot}ss_0$: immediately from $acyc{\cdot}si_0$, $ss_0 = si_0 +\!\!+ \langle [var', v_1] \rangle$ and def. of $acyc$

(1.2.2): proof of $uniq{\cdot}ss_0$:

$$
\begin{array}{ll}
 & uniq{\cdot}ss_0 \\
= & \qquad \langle\!\langle\, ss_0 = si_0 +\!\!+ \langle [var', v_1] \rangle \,\rangle\!\rangle \\
 & uniq{\cdot}(si_0 +\!\!+ \langle [var', v_1] \rangle) \\
= & \qquad \langle\!\langle\, \text{prop. } P3\ ,\ uniq{\cdot}si_0 \,\rangle\!\rangle \\
 & \forall n : Nat\,.\, (1 \leq n \leq \#{\cdot}si_0 \ \Rightarrow\ \pi_n.si_0 \neq [var', v_1]\,) \\
= & \qquad \langle\!\langle\, [var', v_1] \notin rng{\cdot}si_0 \,\rangle\!\rangle \\
 & true
\end{array}
$$

(1.2.3): $1 \leq n_0 \leq \#{\cdot}ss_0$: trivial

(1.2.4): proof of $\mathcal{R}{\cdot}ss_0{\cdot}n_0 = \mathcal{E}{\cdot}d_0$:

$$
\begin{array}{ll}
 & \mathcal{R}{\cdot}ss_0{\cdot}n_0 \\
= & \qquad \langle\!\langle\, ss_0 = si_0 +\!\!+ \langle [var', v_1] \rangle\ ,\ n_0 = \#{\cdot}si_0 + 1 \,\rangle\!\rangle
\end{array}
$$

$$
\begin{array}{ll}
 & \mathcal{R}\cdot(si_0 +\!\!+ \langle[var', v_1]\rangle)\cdot(\#\cdot si_0 + 1) \\
= & \quad \langle\!\!\langle \text{ def. } \mathcal{R} \rangle\!\!\rangle \\
 & [var, v_1] \\
= & \quad \langle\!\!\langle \text{ proof of (1.1.4) } \rangle\!\!\rangle \\
 & \mathcal{E}\cdot d_0
\end{array}
$$

(1.2.5): proof of $SS_0 = SI_0 \cup D_0$:

$$
\begin{array}{ll}
 & SS_0 \\
= & \quad \langle\!\!\langle\, ss_0 = si_0 +\!\!+ \langle[var', v_1]\rangle \,\rangle\!\!\rangle \\
 & SI_0 \;\cup\; \{\mathcal{R}\cdot(si_0 +\!\!+ \langle[var', v_1]\rangle)\cdot(\#\cdot si_0 + 1)\} \\
= & \quad \langle\!\!\langle \text{ proof of (1.2.4) } \rangle\!\!\rangle \\
 & SI_0 \;\cup\; \{\mathcal{E}\cdot d_0\} \\
= & \quad \langle\!\!\langle \text{ proof of (1.1.5) } \rangle\!\!\rangle \\
 & SI_0 \;\cup\; D_0
\end{array}
$$

(1.2.6): $\exists sa : Seq(PN)\,.\; ss_0 = si_0 +\!\!+ sa$: trivial

8 Improving the efficiency of attribute evaluation

Up till now we have adhered to the bucket brigade idea in a straightforward way. Notably with production $E \rightarrow (E + E)$ — and something similar holds for $E \rightarrow (E * E)$ — the idea has been to visit the left and right subtrees in strict succession, and to form the ultimate result — the pair ss_0, n_0 — on the basis of the outcome ss_2 of the right tree, via the evaluation rule

$$
\begin{array}{l}
(\,[plus', \langle n_1, n_2\rangle] \in rng\cdot ss_2 \\
\quad \rightarrow\; ss_0 = ss_2 \;\;,\;\; n_0 = (\iota\, n \mid 1 \le n \le \#\cdot ss_2 \;\wedge\; \pi_n.ss_2 = [plus', \langle n_1, n_2\rangle] \mid n) \\
[]\,[plus', \langle n_1, n_2\rangle] \notin rng\cdot ss_2 \\
\quad \rightarrow\; ss_0 = ss_2 +\!\!+ \langle[plus', \langle n_1, n_2\rangle]\rangle \;\;,\;\; n_0 = \#\cdot ss_2 + 1 \\
)
\end{array}
\qquad (*)
$$

This evaluation takes place under the assumption of

$$Q_E\cdot d_0\cdot si_0 \;\wedge\; R_E\cdot d_1\cdot si_1\cdot ss_1\cdot n_1 \;\wedge\; R_E\cdot d_2\cdot si_2\cdot ss_2\cdot n_2$$

while additionally $si_1 = si_0$ and $si_2 = ss_1$. On the basis of this knowledge it is possible to modify the selection criterion in $(*)$, thereby improving the efficiency of the evaluation of attributes ss_0 and n_0. We shall discuss two ways to do so; first, however, two more properties are stated.

The properties are biased towards their specific use with production 2 (see also item 1 of subsection 9.2); within them, universal quantification over free variables is implicit again, where variables are of the obvious types.

*P*8. If $acyc\cdot s \;\wedge\; uniq\cdot s$

$$[plus, \langle \mathcal{R}\cdot s\cdot n1, \mathcal{R}\cdot s\cdot n2\rangle] \in \{k \mid 1 \le k \le \#\cdot s \mid \mathcal{R}\cdot s\cdot k\} \quad \Leftrightarrow \quad [plus', \langle n1, n2\rangle] \in rng\cdot s$$

$P9$. If

$$acyc{\cdot}ss \;\wedge\; uniq{\cdot}ss \;\wedge\; 1 \le n \le \#{\cdot}ss \;\wedge\; \mathcal{R}{\cdot}ss{\cdot}n = \mathcal{E}{\cdot}d \;\wedge\; SS = SI \cup D \;\wedge\; ss = si +\!\!+ sa$$

(i.e., $R_E{\cdot}d{\cdot}si{\cdot}ss{\cdot}n$, with sa made explicit),
we have — letting SA denote $\{k \mid \#{\cdot}si < k \le \#{\cdot}ss \mid \mathcal{R}{\cdot}ss{\cdot}k\}$:

(i) $SI \cup SA = SS$ (immediately from $ss = si +\!\!+ sa$)
(ii) $SI \cap SA = \emptyset$ (from $uniq{\cdot}ss \;\wedge\; ss = si +\!\!+ sa$)
(iii) $SA \subseteq D$

proof of (iii):

$$\begin{array}{ll} & SA \\ = & \quad \{\!\!\{ \text{(i)} \wedge \text{(ii)}, \text{i.e, } SA = SS \backslash SI \}\!\!\} \\ & SS \backslash SI \\ = & \quad \{\!\!\{ SS = SI \cup D \}\!\!\} \\ & (SI \cup D) \backslash SI \\ \subseteq & \quad \{\!\!\{ \}\!\!\} \\ & D \end{array}$$

Now consider again the setting of evaluation rule $(*)$ in production 2. Recall that we have

$$Q_E{\cdot}d_0{\cdot}si_0 \;\wedge\; R_E{\cdot}d_1{\cdot}si_1{\cdot}ss_1{\cdot}n_1 \;\wedge\; R_E{\cdot}d_2{\cdot}si_2{\cdot}ss_2{\cdot}n_2 \;\wedge\; si_1 = si_0 \;\wedge\; si_2 = ss_1$$

We show that $[plus', \langle n_1, n_2\rangle] \in rng{\cdot}ss_2 \;\Leftrightarrow\; [plus', \langle n_1, n_2\rangle] \in rng{\cdot}si_0$:

$$\begin{array}{ll} & [plus', \langle n_1, n_2\rangle] \in rng{\cdot}ss_2 \\ = & \quad \{\!\!\{ \text{prop. } P8 \}\!\!\} \\ & [plus, \langle \mathcal{R}{\cdot}ss_2{\cdot}n_1, \mathcal{R}{\cdot}ss_2{\cdot}n_2\rangle] \in \{k \mid 1 \le k \le \#{\cdot}ss_2 \mid \mathcal{R}{\cdot}ss_2{\cdot}k\} \\ = & \quad \{\!\!\{ \mathcal{R}{\cdot}ss_2{\cdot}n_1 = \mathcal{E}{\cdot}d_1 \;,\; \mathcal{R}{\cdot}ss_2{\cdot}n_2 = \mathcal{E}{\cdot}d_2 \}\!\!\} \\ & [plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2\rangle] \in \{k \mid 1 \le k \le \#{\cdot}ss_2 \mid \mathcal{R}{\cdot}ss_2{\cdot}k\} \\ = & \quad \{\!\!\{ \exists sa : Seq(PN) \,.\, ss_2 = si_2 +\!\!+ sa \text{ – say } sa_2 \}\!\!\} \\ & [plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2\rangle] \in \{k \mid 1 \le k \le \#{\cdot}(si_2 +\!\!+ sa_2) \mid \mathcal{R}{\cdot}(si_2 +\!\!+ sa_2){\cdot}k\} \\ = & \quad \{\!\!\{ P9 \text{ – hence } \{k \mid \#{\cdot}si_2 < k \le \#{\cdot}(si_2 +\!\!+ sa_2) \mid \mathcal{R}{\cdot}(si_2 +\!\!+ sa_2){\cdot}k\} \subseteq D_2 \;, \\ & \qquad \mathcal{E}{\cdot}d_2 \sqsubset_{ET} [plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2\rangle] \text{ – hence } [plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2\rangle] \notin D_2 \;, \text{ prop. } P7 \}\!\!\} \\ & [plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2\rangle] \in \{k \mid 1 \le k \le \#{\cdot}si_2 \mid \mathcal{R}{\cdot}si_2{\cdot}k\} \\ = & \quad \{\!\!\{ si_2 = ss_1 \;,\; \text{repeat the two preceding steps for } d_1 \;,\; si_1 = si_0 \}\!\!\} \\ & [plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2\rangle] \in \{k \mid 1 \le k \le \#{\cdot}si_0 \mid \mathcal{R}{\cdot}si_0{\cdot}k\} \\ = & \quad \{\!\!\{ \mathcal{R}{\cdot}ss_2{\cdot}n_1 = \mathcal{E}{\cdot}d_1 \;,\; \mathcal{R}{\cdot}ss_2{\cdot}n_2 = \mathcal{E}{\cdot}d_2 \}\!\!\} \\ & [plus, \langle \mathcal{R}{\cdot}ss_2{\cdot}n_1, \mathcal{R}{\cdot}ss_2{\cdot}n_2\rangle] \in \{k \mid 1 \le k \le \#{\cdot}si_0 \mid \mathcal{R}{\cdot}si_0{\cdot}k\} \\ = & \quad \{\!\!\{ (\Rightarrow) : P2 \text{ – hence } \{\mathcal{R}{\cdot}ss_2{\cdot}n_1, \mathcal{R}{\cdot}ss_2{\cdot}n_2\} \subseteq \{k \mid 1 \le k \le \#{\cdot}si_0 \mid \mathcal{R}{\cdot}si_0{\cdot}k\} \;, \\ & \qquad \text{furthermore } ss_2 \text{ extends } si_0 \text{ and } uniq{\cdot}ss_2 \text{ (i.e., } \mathcal{R}{\cdot}ss_2 \text{ is an injection} \\ & \qquad \text{on } dom{\cdot}ss_2) \text{ – hence } 1 \le n_1 \le \#{\cdot}si_0 \wedge 1 \le n_2 \le \#{\cdot}si_0 \;, \text{ prop. } P7 \\ & \quad (\Leftarrow) : ss_2 \text{ extends } si_0 \;, \text{ prop. } P7 \}\!\!\} \\ & [plus, \langle \mathcal{R}{\cdot}si_0{\cdot}n_1, \mathcal{R}{\cdot}si_0{\cdot}n_2\rangle] \in \{k \mid 1 \le k \le \#{\cdot}si_0 \mid \mathcal{R}{\cdot}si_0{\cdot}k\} \\ = & \quad \{\!\!\{ \text{prop. } P8 \}\!\!\} \\ & [plus', \langle n_1, n_2\rangle] \in rng{\cdot}si_0 \end{array}$$

By this equivalence, the selection criterion in $(*)$ can be modified so as to obtain

$$
\begin{array}{l}
(\ [plus', \langle n_1, n_2 \rangle] \in rng{\cdot}si_0 \rightarrow \ \ldots \\
[\!]\ [plus', \langle n_1, n_2 \rangle] \notin rng{\cdot}si_0 \rightarrow \ \ldots \\
)
\end{array}
$$

However, the situation can be improved even further. Recall that ss_2 extends si_0. We show that if this extension is not empty, i.e., $\#{\cdot}ss_2 \neq \#{\cdot}si_0$, then $[plus', \langle n_1, n_2 \rangle] \notin rng{\cdot}ss_2$:

$$
\begin{array}{ll}
 & \#{\cdot}ss_2 \neq \#{\cdot}si_0 \\
= & \quad \{\!\!\{\ \{k \mid \#{\cdot}si_0 < k \leq \#{\cdot}ss_2 \mid \mathcal{R}{\cdot}ss_2{\cdot}k\} \subseteq D_1 \cup D_2\ \}\!\!\} \\
 & \exists d' : PT \,.\, (\ (d' \sqsubseteq_{PT} d_1 \ \vee \ d' \sqsubseteq_{PT} d_2) \ \wedge \ \mathcal{E}{\cdot}d' \in \{k \mid \#{\cdot}si_0 < k \leq \#{\cdot}ss_2 \mid \mathcal{R}{\cdot}ss_2{\cdot}k\}\ ) \\
\Rightarrow & \quad \{\!\!\{\ uniq{\cdot}ss_2\ ,\ \text{prop. } P5\ \}\!\!\} \\
 & \mathcal{E}{\cdot}d_0 \in SS_2 \ \ \Rightarrow \ \ \mathcal{E}{\cdot}d_0 \in \{k \mid \#{\cdot}si_0 < k \leq \#{\cdot}ss_2 \mid \mathcal{R}{\cdot}ss_2{\cdot}k\} \\
= & \quad \{\!\!\{\ \{k \mid \ldots \mid \mathcal{R}{\cdot}ss_2{\cdot}k\} \subseteq D_1 \cup D_2\ ,\ \mathcal{E}{\cdot}d_0 \notin D_1 \cup D_2\ \}\!\!\} \\
 & \mathcal{E}{\cdot}d_0 \notin SS_2 \\
= & \quad \{\!\!\{\ \text{def. } \mathcal{E}\ ,\ d_0 =_e [E \rightarrow (E + E), \langle d_1, d_2 \rangle]\ \}\!\!\} \\
 & [plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2 \rangle] \notin SS_2 \\
= & \quad \{\!\!\{\ \mathcal{R}{\cdot}ss_2{\cdot}n_1 = \mathcal{E}{\cdot}d_1\ ,\ \mathcal{R}{\cdot}ss_2{\cdot}n_2 = \mathcal{E}{\cdot}d_2\ \}\!\!\} \\
 & [plus, \langle \mathcal{R}{\cdot}ss_2{\cdot}n_1, \mathcal{R}{\cdot}ss_2{\cdot}n_2 \rangle] \notin SS_2 \\
\Rightarrow & \quad \{\!\!\{\ \text{prop. } P8 - \Leftarrow\text{-part only}\ \}\!\!\} \\
 & [plus', \langle n_1, n_2 \rangle] \notin rng{\cdot}ss_2
\end{array}
$$

On account of this, $(*)$ can be changed into

$$
\begin{array}{l}
(\ \#{\cdot}ss_2 \neq \#{\cdot}si_0 \rightarrow \ ss_0 = ss_2 +\!\!+ \langle [plus', \langle n_1, n_2 \rangle] \rangle \ \ , \ \ n_0 = \#{\cdot}ss_2 + 1 \\
[\!]\ \#{\cdot}ss_2 = \#{\cdot}si_0 \rightarrow (\ [plus', \langle n_1, n_2 \rangle] \in rng{\cdot}ss_2 \\
\qquad\qquad\qquad \rightarrow \ ss_0 = ss_2 \ \ , \ \ n_0 = (\iota\, n \mid \ldots \mid n) \\
\qquad\qquad [\!]\ [plus', \langle n_1, n_2 \rangle] \notin rng{\cdot}ss_2 \\
\qquad\qquad\qquad \rightarrow \ ss_0 = ss_2 +\!\!+ \langle [plus', \langle n_1, n_2 \rangle] \rangle \ \ , \ \ n_0 = \#{\cdot}ss_2 + 1 \\
\qquad\qquad) \\
)
\end{array}
$$

The outermost selection criterion is attractive because it does not involve inspection of the elements of a(ny) sequence. For that reason the latter solution is to be preferred to the earlier ones; in particular it will be profitable if common sub-expressions do not occur very frequently.

9 Conclusions, questions and remarks

This final section consists of two parts. In the first part we state the main conclusions to be drawn from the example under consideration, in the second we list a number of remaining questions and remarks. Some of the latter are specifically related to this example, while others have a more general scope; all of them are meant to be processed in future work.

9.1 Conclusions

Taking the problem of representing arithmetical expressions by DAGs as a vehicle, in this paper it is shown how the proof system for one-pass AGs can be employed to show the correctness of a non-trivial, fairly realistic AG in a modular fashion.

A prerequisite for being able to reason about such a problem in a precise way is the formalisation of the notions constituting the problem. The example presented here demonstrates that typed inference systems are an appropriate formalism to serve this purpose.

Then again, it is of vital importance to pinpoint the relevant properties of the attribute domains (and related notions), as they play a substantial role during the whole trajectory of developing and proving the AG. One could easily devote a separate paper to the study of such a theory. However, the theory *per se* is not the subject of this note, therefore it is condensed into properties $P1$ through $P9$ (sections 5 and 8).

By the nature of our proof system for one-pass AGs, the overall correctness proof for such an AG requires the proof of a number of implications per production. At all times, the context embodies the environment in which these proofs must be conducted and — provided the theory of the attribute domains has been developed satisfactorily — the latter can be done entirely by formal means. For a production p, $\#p + 1$ implications need to be proven; hence, the burden of proof increases with the number of right-hand side nonterminals of p.

The proof of an implication can in general be split up into a number of small parts. The current example demonstrates two occasions to do so. First, if attributes are defined with case-analysis, the cases may be dealt with separately. The proofs corresponding to such cases tend to share substantial parts and, hence, do not have to be elaborated fully. For an example of this phenomenon, observe in subsection 7.2 how the proof of (3.2.4) imports a large part of that of (3.1.4). Second, if the consequent (i.e., a Q- or R-term) of some implication to be proven consists of a number of conjuncts, the proof of this implication can be subdivided in equally many smaller proofs. Typically, such a subdivision yields a large number of rather isolated proofs; as far as they are not isolated, the proofs again tend to share considerable parts, e.g. notice how the proof of (3.1.5) in subsection 7.2 uses a large part of that of (3.1.4).

In general we can say that the subdivision of proof obligations enhances a separation of concerns and results in a number of small proofs to be conducted. Most of these proofs are fairly straightforward and can be omitted in a practical situation; this is true in particular for productions involving copy rules and/or multiple occurrences of the same nonterminal (these occurrences all have similar specifications). By an adequate numbering of the proofs — as in section 7 — the relations between them may be kept sufficiently clear; observe for instance in subsection 7.2 how for all x the proofs of $(3.1.x)$ relate to those of $(3.2.x)$. Then, sharing of proof parts will most likely occur between proofs with related numbers.

It can be observed from section 7 that the proofs for a production p are indeed local: apart from the general (axiomatic) properties of attribute domains — which are supposed to appear in Γ —, the only kinds of terms appealed to are

- specification parts appearing as antecedents of the implication in question
- equations (mainly evaluation rules) appearing in the context; even subject to a one-pass evaluation order.

The only exception to this rule occurs when p's right-hand side involves lexical nonterminals (see item 5 of subsection 9.2). This exception is not regarded disastrous to the modularity of

the proof system, however, as its consequences are not far-reaching; it is the price we have to pay for not wanting to deal extensively with lexical matters.

Another typical feature of the proofs, which is even enhanced with a bucket brigade-like attribute structure, is the strictly alternating appeal to specification parts and evaluation rules for the justification of proof steps. This is especially apparent in the proof of (3.1.6) in subsection 7.2.

9.2 Questions and remarks

1. (on the nature of the additional properties $P8$ and $P9$)

(i) Property $P8$ is biased towards the specific treatment of production $E \rightarrow (E + E)$ in section 8. In fact, it should be possible to formulate a more general property — $P8'$, say — instead, dealing with expression trees in general, not just those of the form $[plus, \ldots]$. Property $P8'$, then, would be an important key to the (transformational) *derivation* of the AG: it is the justification of the fact that a condition like

$$[plus, \langle \mathcal{E}{\cdot}d_1, \mathcal{E}{\cdot}d_2 \rangle] \in \{k \mid 1 \leq k \leq \#{\cdot}ss_2 \mid \mathcal{R}{\cdot}ss_2{\cdot}k\}$$

— which may act as a selection criterion for production 2 in an early version of the grammar — can be refined to

$$[plus', \langle n_1, n_2 \rangle] \in rng{\cdot}ss_2$$

(and something similar in productions 3, 4 and 5).

(ii) There is a difference in spirit between properties $P1$ through $P8$ (or rather: $P8'$) and $P9$. Whereas $P1$ through $P8$ can be considered as general domain properties, this is not true for $P9$. The latter is a derived property that is made explicit only to facilitate the reasoning about a certain local transformation. This is reflected by the fact that the premisses of $P9$ specifically concern a number of conditions that together constitute a specification for nonterminal E.

2. (concerning the transformations of section 8)
In section 8 it is shown how a particular evaluation rule in production 2 can be modified, while maintaining the local correctness of the production (and hence the overall correctness of the grammar).

The rule to be modified is the one for the evaluation of attributes ss_0 and n_0. The correctness of the transformation is shown under the assumption of

$$Q_E{\cdot}d_0{\cdot}si_0 \ \wedge \ R_E{\cdot}d_1{\cdot}si_1{\cdot}ss_1{\cdot}n_1 \ \wedge \ R_E{\cdot}d_2{\cdot}si_2{\cdot}ss_2{\cdot}n_2 \ \wedge \ si_1 = si_0 \ \wedge \ si_2 = ss_1$$

which are (qua topology of a one-pass evaluation) all relevant conditions that have been established upon performing the aforementioned evaluation. A close inspection of the proofs reveals that indeed all of these conditions are needed — the R_E-terms are notably used to instantiate $P9$. (Note: conditions $Q_E{\cdot}d_1{\cdot}si_1$ and $Q_E{\cdot}d_2{\cdot}si_2$ play no role. As appears from the proof obligation for production 2, they are implied by the other conditions.)

A couple of questions have not been answered yet. First, how do transformations like the ones in section 8 (that do not modify the attribute structure) relate to "Jonkers-like" AG-transformations, involving addition and removal of attributes? Second, can the optimised solutions be proven correct in isolation without making a detour as in section 8 (and does this bring about a change of specifications)?

3. (meaningfulness of evaluation expressions)
Up till now we have not considered the meaningfulness of the expressions occurring in the evaluation rules, notably

$$(\iota\, n \mid 1 \leq n \leq \#{\cdot}ss_2 \wedge \pi_n.ss_2 = [plus', \langle n_1, n_2\rangle] \mid n)$$

in production 2. The use of the ι-operator is allowed (meaningful) here on account of $uniq{\cdot}ss_2$ (from R_2), property $P3$ and the selection criterion $[plus', \langle n_1, n_2\rangle] \in rng{\cdot}ss_2$, which precedes the evaluation of the ι-expression.

In general, it is clear that such expressions have to be viewed in the right context, i.e., the topology of attribute evaluation has to be taken into account; in the case of the ι-expression, we have the validity of

$$Q_E{\cdot}d_0{\cdot}si_0 \;\wedge\; R_E{\cdot}d_1{\cdot}si_1{\cdot}ss_1{\cdot}n_1 \;\wedge\; R_E{\cdot}d_2{\cdot}si_2{\cdot}ss_2{\cdot}n_2$$

at our disposal. (Note: the equations $si_1 = si_0$ and $si_2 = ss_1$ are not needed for this purpose: an evaluation expression is concerned solely with outside attributes.)

Quite another point is the *well-formedness* of evaluation expressions, which has to do with the occurrence of certain attribute variables in evaluation expressions; this is compelled by the syntax of (one-pass) AGs. This allows us, for example, to write

$$[plus', \langle n_1, n_2\rangle] \in rng{\cdot}ss_2$$

in the selection criterion of production 2, as n_1, n_2 and ss_2 are well-defined at that point.

4. (on a pure bucket brigade solution)
The first proposal for a specification for E (section 6) seems to lead to a fairly straightforward bucket brigade solution. It turns out, however, that the proof obligations emerging from the "obvious" evaluation rules cannot be met. Hence, specification and implementation do not match. As explained in section 6, addition of

$$\exists sa : Seq(PN)\,.\; ss = si +\!\!+ sa \qquad (*)$$

to R_E fixes the deficiency. Operationally, this term prevents the determination of ss_2 and n_2 (from si_2) in a rule like

$$E\langle -si_0, +ss_0, +n_0\rangle \;\rightarrow\; (E\langle -si_1, +ss_1, +n_1\rangle + E\langle -si_2, +ss_2, +n_2\rangle) \qquad (**)$$

from shaking up si_2 in such a way that n_1 would no longer be reliable, i.e., satisfy $\mathcal{R}{\cdot}ss_2{\cdot}n_1 = \mathcal{E}{\cdot}d_1$.

As a consequence, the term $(*)$ disables some other implementations, e.g. one that requires all "minimal terms" $[var', v]$ and $[con', c]$ to be in the beginning of the result sequence. Indeed, under such an implementation n_1 would not be reliable. (Note: if a demand like that would

be "user required", a condition expressing the demand would have to be added to R_Z. This condition then generalises to R_E and there effectively prevents the term $(*)$ to occur, as the two are incompatible.)

It is still an open question how a more pure bucket brigade solution, i.e., one that does not exclude obvious implementations, would have to be specified.

At first it seems that the invalidation of n_1 in $(**)$ can be abolished by passing it to the right-most E-occurrence, alongside with ss_1. Via its specification, E would then be responsible for updating this value if necessary, and it would deliver *two* synthesized natural numbers, providing the indices in ss_2 where representations of both sub-trees can be found. However, all E-occurrences must be treated alike — notably: have the same specification —, so also the left E-occurrence would have to deliver two naturals; it is not clear at all what they would stand for. Continuing this reasoning, one may end up with a (variable sized) *sequence* of naturals to be passed around the tree in a depth-first fashion, alongside the result-sequence s.

5. (on lexical nonterminals)
The proof of production 4 (section 7.3) is on strained terms with the asserted modularity of our proof system, which can be expressed roughly as "proofs are local to the individual productions".

Consider notably the proof of (1.1.4), where a production tree $d_0 : PT_E$ of the form $[E \rightarrow V, \langle d_1 \rangle]$ is in force. As the proof proceeds, we are led to consider the internal structure of d_1, which — on account of the fact that $d_1 : PT_V$ and V is a lexical nonterminal — is of the form $[V \rightarrow yield{\cdot}d_1, \langle\rangle]$.

Now the latter property is actually the only thing about d_1 that plays a role in the proof, and as it is true for *all* production trees d_1 issued from V, no radical case-analysis emerges from the considering of d_1's structure.

This slight deviation from the modularity of the system is an immediate consequence of V's being a lexical nonterminal, as we shall explain now.

Recall that the choice to regard a nonterminal like V as a lexical nonterminal is inspired by the fact that we do not want to bother about the precise shape of the production trees issued from V. The only thing of interest about such a tree is its terminal production; therefore a lexical nonterminal is provided with a single synthesized attribute of the appropriate type and a specification expressing our intentions. For example

$$V\langle +v : VAR\rangle$$
$$R_V{\cdot}d{\cdot}v \quad : \quad v = yield{\cdot}d$$

On the other hand, however, the shape of the production trees of a lexical nonterminal cannot be neglected altogether, notably if an AG's problem specification is expressed in terms of a function on the production trees of the underlying CFG. In order for such a function — $\mathcal{E}$ in our example — to be well-defined (or: totally defined) there must be a clause expressing the function's result upon application to a tree issued from a lexical nonterminal. Again, this result will only involve the terminal production of such a tree, therefore it suffices to represent the tree in an abbreviated way, suggesting that a lexical nonterminal derives its terminal strings in one step; e.g. $[V \rightarrow v, \langle\rangle]$.

It is the discrepancy between these two views on lexical nonterminals — the attributed and the functional one — that causes the observed deviation from a purely modular proof

system. The deviation occurs when the two views clash, viz. with all productions involving lexical nonterminals at their right-hand sides.

A number of possibilities to abolish the discrepancy emerge; none of them seems acceptable, however:

- Considering lexical nonterminals as terminal symbols leads to an exhaustive spelling out of the "lexical types" (VAR and CON in our example). Instead, we want to express that such symbols derive a variety of terminal strings — all with a common structure —, without going into the details of this structure (type).
- Ignoring the production trees issued from lexical nonterminals when defining functions on production trees causes such functions not to be totally defined anymore. (Nb: in our current example, this option would lead to

  $$\mathcal{E}\cdot[E \rightarrow V, \langle d_1 \rangle] \;=\; [var, yield \cdot d_1]$$

  and the omission of the clause for $\mathcal{E}\cdot[V \rightarrow v, \langle\rangle]$.)
- Not providing lexical nonterminals with a default attribute (and specification) anymore has the disadvantage that each of them must be considered separately, which may lead to quite divergent specifications. Moreover, we lose the expression of the fact that the only thing of interest about a lexical nonterminal is the terminal string derived.

  In our current example, this option leads to V and C receiving the same attributes and specification as E. For productions $E \rightarrow V$ and $E \rightarrow C$, all evaluation rules would be copy rules.

Considering all aspects, it seems that the current solution — i.e., allowing a slight deviation from a purely modular proof structure when lexical nonterminals are involved — may be by all means the most acceptable one. It should however be recognised that the discrepancy exists.

References

[ASU86] Aho, A.V., R. Sethi, J.D. Ullman; *Compilers — Principles, Techniques and Tools*, Addison-Wesley (1986)

[Hem85] Hemerik, C.; *Notes on Compiler Construction 5: Common Subexpression Elimination*, internal note CH24/NCC5, Eindhoven University of Technology (1985)

[Mar90] Marcelis, A.J.J.M.; *A Logic for One-pass Attribute Grammars*, Computing Science Note 90/07, Eindhoven University of Technology, Dept. of Math. and Comp. Sci., The Netherlands (1990)

In this series appeared:

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 parallel 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, p. 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.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.
90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.

90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.

91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. .
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.