

# Strictness Analysis for Attribute Grammars

Mads Rosendahl  
DIKU, University of Copenhagen  
Universitetsparken 1, DK-2100 Copenhagen Ø  
Denmark

E-mail `rose@diku.dk`

1992

## Abstract

Attribute grammars may be seen as a (rather specialised) lazy or demand-driven programming language. The “programs” in this language take text or parse trees as input and return values of the synthesised attributes to the root as output. From this observation we establish a framework for abstract interpretation of attribute grammars. The framework is used to construct a strictness analysis for attribute grammars. Results of the analysis enable us to transform an attribute grammar such that attributes are evaluated during parsing, if possible. The analysis is proved correct by relating it to a fixpoint semantics for attribute grammars. An implementation of the analysis is discussed and some extensions to the analysis are mentioned.

## 1 Introduction

As pointed out by [11] there are broadly speaking two approaches to attribute evaluation: either one restricts the class of attribute grammars and makes it possible to use an efficient evaluation strategy or one allows any well-defined attribute grammar but reduces the efficiency of the evaluation method. A large part of the work in Attribute Grammars has been concentrated on defining classes or subclasses of attribute grammars with special evaluation properties, or constructing and analysing classification methods ([5]).

We follow a different line by analysing attribute grammars rather than classifying them. Hence, the aim is to locate those parts of an attribute grammar for which more efficient implementation techniques are available. This approach has mainly been pursued with a view to optimising the storage management in attribute grammars ([14] and [3]) but it has also been used for analysing the order of evaluation in an attribute grammar (see [9] and [21]).

In this paper we provide a semantic basis for this type of analysis based on abstract interpretation ([4] and [16]). The framework is based on a fixpoint semantics for attribute grammars in the style of [2] which is then changed to give non-standard interpretations for static analysis of attribute grammars. Results from this kind of interpretations may be used to answer certain questions about the runtime behaviour of attribute grammars.

Attribute Grammars as a language has many similarities with lazy or demand-driven functional languages. From a parse tree an evaluator computes values of the synthesised attributes of the root symbol. The order of evaluation of attribute values is determined by “demand” or dependencies rather than the order in which they occur in the grammar. This resembles the situation for lazy functional languages where expressions and arguments are only computed when needed for evaluating the result of function calls.

Lazy functional languages seem very well suited for static analysis; strictness analysis, one of the most successful abstract interpretations, is designed for such languages. Much attention has been given to extending the original definition ([16]), to higher-order functions, lazy data structures, polymorphic type systems, and to finding efficient implementations of the analysis. In this paper, however, strictness analysis in the first-order case suffices for the application to attribute grammars.

Strictness analysis may be used to find arguments to functions which may be evaluated prior to the call, so as to change call-by-need to call-by-value. By doing this one may save both time and space as one does not need to build suspensions for expressions which may be evaluated later. Strictness analysis for attribute grammars enable us to find attributes which may be evaluated during parsing. By this, one may eliminate the need for symbolic expressions for attributes or parts of the parse tree. In this way one is likely to save both time and space during evaluation.

## 2 Notation and terminology

Most of the notation in this paper is standard. The semantic framework uses fixpoint semantics based on the theory of complete partial orders and continuous functions. Grammars are as usual context-free grammars.

We will make a few restrictions on the specification of grammars and attribute grammars to simplify the semantic description. The two main restrictions imposed here are that all productions should have a right hand side of a single terminal symbol or of  $\mathbf{n}$  nonterminals for some fixed  $\mathbf{n}$ , and that each nonterminal has exactly two attributes: one synthesised and one inherited. Any actual implementation of an analysis should not of course impose such restrictions on attribute grammars. These restrictions can in practice be made without loss of generality. The extension to allow  $\mathbf{m}$  inherited and synthesised attributes ( $\mathbf{m} > 1$ ) is straightforward, as the evaluation of arguments in a fixpoint seman-

tics is done lazily. Furthermore we allow several common extensions to the original definition of attribute grammars ([15]) as expressions may contain conditionals ([7]) and the root symbol may have an inherited attribute. In this way the analysis should be applicable to a very large class of attribute grammars.

## 2.1 Definitions

A *context-free grammar* (or *grammar*, for short) is a four-tuple  $\mathbf{G} = (\mathbf{R}, \mathbf{P}, \mathbf{N}, \mathbf{T})$  of a root symbol  $\mathbf{R} \in \mathbf{N}$ , a finite set of productions  $\mathbf{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_k\}$ , a finite set of nonterminals  $\mathbf{N}$ , and a finite set of terminals  $\mathbf{T}$ . To each production  $\mathbf{p} \in \mathbf{P}$  there will be associated a left hand side symbol  $\text{lhs}(\mathbf{p}) \in \mathbf{N}$  and a right hand side of either  $\mathbf{n}$  nonterminals:  $\text{rhs}(\mathbf{p}) \in \mathbf{N}^{\mathbf{n}}$  or a terminal:  $\text{rhs}(\mathbf{p}) \in \mathbf{T}$ . We assume that  $\mathbf{N}^{\mathbf{n}} \cap \mathbf{T} = \emptyset$ . We may further assume (without loss of generality) that all nonterminals occurring in a production are distinct and distinct from the nonterminal on the left hand side.

The set of *parse trees*  $\mathcal{T}_{\mathbf{G}}$  for a grammar  $\mathbf{G}$  is the smallest set such that

$$\begin{aligned} \langle \mathbf{p}, \mathbf{q} \rangle \in \mathcal{T}_{\mathbf{G}} &\quad \Leftrightarrow \mathbf{q} \in \mathbf{T} \wedge \text{rhs}(\mathbf{p}) = \mathbf{q} \\ \langle \mathbf{p}, \mathbf{t}_1, \dots, \mathbf{t}_n \rangle \in \mathcal{T}_{\mathbf{G}} &\Leftrightarrow \mathbf{t}_j \in \mathcal{T}_{\mathbf{G}}, \mathbf{j} = 1, \dots, \mathbf{n} \wedge \\ &\quad \text{rhs}(\mathbf{p}) = \langle \mathbf{q}_1, \dots, \mathbf{q}_n \rangle \wedge \\ &\quad \text{lhs}(\mathbf{t}_j \downarrow 1) = \mathbf{q}_j, \mathbf{j} = 1, \dots, \mathbf{n} \end{aligned}$$

Notice that as parse trees we allow any tree built from productions in the grammar. We may from this define the subset of parse trees with the root symbol as the left hand side of the outermost production.

Let  $\mathbf{V}$  be a set of values which includes numbers and boolean values. Let  $\underline{\mathbf{c}}_j, \mathbf{j} \in \mathbb{N}$  be names for constants  $\mathbf{c}_j \in \mathbf{V}$  and  $\underline{\mathbf{a}}_j, \mathbf{j} \in \mathbb{N}$  be names for  $\mathbf{m}$ -ary partial functions over  $\mathbf{V}$ :  $\mathbf{a}_j : \mathbf{V}^{\mathbf{m}} \xrightarrow{\mathbf{p}} \mathbf{V}$ .

An *attribute expression* (or *expression*, for short) is a string in the language

$$\begin{aligned} \mathbf{e} &\rightarrow \underline{\mathbf{c}}_j \\ &| \mathbf{q}_j.\mathbf{s}, \quad \mathbf{j} = 1, \dots, \mathbf{n} \\ &| \mathbf{q}_0.\mathbf{i} \\ &| \underline{\mathbf{a}}_j(\mathbf{e}_1, \dots, \mathbf{e}_m) \\ &| \mathbf{if} \mathbf{e}_1 \mathbf{then} \mathbf{e}_2 \mathbf{else} \mathbf{e}_3 \end{aligned}$$

The meaning or semantics of expressions will be defined later.

An *attribute grammar* is a context-free grammar  $\mathbf{G}$  with a list of expressions associated with each production. For a production with  $\mathbf{n}$  nonterminals on the right hand side, there will be  $\mathbf{n} + 1$  expressions, and for productions with one terminal symbol on the right hand side there will be one expression.

## 2.2 Notation

An attribute grammar may be specified as a list of *attributed productions* of the form

$$\mathbf{p} : \mathbf{q}_0 ::= \mathbf{q}_1 \cdots \mathbf{q}_n \{ \mathbf{q}_1.i := \mathbf{e}_1; \dots; \mathbf{q}_n.i := \mathbf{e}_n; \mathbf{q}_0.s := \mathbf{e}_{n+1} \}$$

where  $\text{rhs}(\mathbf{p}) = \langle \mathbf{q}_1, \dots, \mathbf{q}_n \rangle$  and  $\text{lhs}(\mathbf{p}) = \mathbf{q}_0$  and  $[\mathbf{e}_1, \dots, \mathbf{e}_n, \mathbf{e}_{n+1}]$  is the list of expressions associated with production  $\mathbf{p}$ . For productions with a single terminal symbol on the right hand side the form is

$$\mathbf{p} : \mathbf{q}_0 ::= \mathbf{q}_1 \{ \mathbf{q}_0.s := \mathbf{e} \}$$

where  $\mathbf{e}$  is the expression associated with production  $\mathbf{p}$ .

The purpose of the  $n + 1$  expressions in the first form of attributed productions is that expressions  $\mathbf{e}_1$  to  $\mathbf{e}_n$  define the inherited attributes of the  $n$  nonterminals on the right hand side of the production, and that the expression  $\mathbf{e}_{n+1}$  defines the synthesised attribute of the nonterminal on the left hand side. In the notation for attributed productions this is indicated by “.*i*” for inherited attributes and “.*s*” for synthesised attributes. This is made precise by the fixpoint semantics below.

The attributed productions may be seen as an abstract syntax for some other kind of notation of attribute grammars. Whether the attribute grammar is defined in OLGA ([12]), Aladin ([13]), or as an extended attribute grammar ([23]) is not central to this work.

## 3 Fixpoint semantics

The meaning of an attributed production is a function which maps a parse tree and the value of its inherited attribute to the value of its synthesised attribute. The meaning of an attribute grammar is an environment of such maps. The semantics follows the usual pattern for recursion equation systems (see [8] and [22]).

The description differs from [2] in that the semantics is given directly as an interpretation of the attribute grammar rather than as rules to construct a recursion equation system which defines the meaning. In spirit it is similar to the recursive evaluation method for attribute grammars ([6] and [11]).

### 3.1 Semantic rules

In the semantics the domain of attribute values is  $\mathbf{D} = \mathbf{V}_\perp$ . An attributed production is given a meaning as a function of type  $\mathcal{T}_{\mathbf{G}} \rightarrow \mathbf{D} \rightarrow \mathbf{D}$  which maps a parse tree and its inherited attribute to the value of its synthesised. An attribute expression in a production may depend on the inherited attribute to the nonterminal on the left hand side and to the synthesised attributes to the

(at most)  $\mathbf{n}$  nonterminals on the right hand side. For this reason the semantic function for expressions uses an environment of  $\mathbf{n} + 1$  attribute values.

The semantics consists of three semantic functions, one for each of the syntactic categories in the language.

Syntactic categories

$$\begin{aligned} \mathbf{ag} &\in \mathbf{AG} && : \mathbf{ap}_1 \cdots \mathbf{ap}_k && \text{attribute grammar} \\ \mathbf{ap} &\in \mathbf{AP} && && \text{attributed production} \\ \mathbf{e} &\in \mathbf{Exp} && && \text{expressions} \end{aligned}$$

Semantic domains

$$\begin{aligned} \mathbf{v} &\in \mathbf{D} && : \mathbf{V}_\perp && \text{attribute values} \\ \mathbf{t} &\in \mathcal{T}_G && && \text{parse trees} \\ \rho &\in \mathbf{Env} && : (\mathcal{T}_G \rightarrow \mathbf{D} \rightarrow \mathbf{D})^k && \text{production environment} \end{aligned}$$

Semantic functions

$$\begin{aligned} \mathbf{M} &: \mathbf{AG} \rightarrow \mathbf{Env} \\ \mathbf{P} &: \mathbf{AP} \rightarrow \mathbf{Env} \rightarrow \mathcal{T}_G \rightarrow \mathbf{D} \rightarrow \mathbf{D} \\ \mathbf{E} &: \mathbf{Exp} \rightarrow \mathbf{D}^{\mathbf{n}+1} \rightarrow \mathbf{D} \end{aligned}$$

Semantic rules

$$\begin{aligned} \mathbf{M}[\mathbf{ap}_1 \dots \mathbf{ap}_n] &= \text{fix}(\lambda \rho. \langle \mathbf{P}[\mathbf{ap}_1] \rho, \dots, \mathbf{P}[\mathbf{ap}_n] \rho \rangle) \\ \mathbf{P}[\mathbf{p} : \mathbf{q}_0 ::= \mathbf{q}_1 \cdots \mathbf{q}_n \{ \mathbf{q}_1.i := \mathbf{e}_1; \dots; \mathbf{q}_n.i := \mathbf{e}_n; \mathbf{q}_0.s := \mathbf{e}_{n+1} \}] \rho \mathbf{t} \mathbf{v} &= \\ &\text{let } \langle \mathbf{p}, \mathbf{t}_1, \dots, \mathbf{t}_n \rangle = \mathbf{t} \text{ and} \\ &\quad \langle \mathbf{p}_{j_1}, \dots \rangle = \mathbf{t}_1 \text{ and } \cdots \text{ and } \langle \mathbf{p}_{j_n}, \dots \rangle = \mathbf{t}_n \\ &\text{in } \mathbf{E}[\mathbf{e}_{n+1}] (\text{fix} \lambda \xi. \langle (\rho \downarrow j_1) \mathbf{t}_1 \rangle (\mathbf{E}[\mathbf{e}_1] \xi), \dots, \langle (\rho \downarrow j_n) \mathbf{t}_n \rangle (\mathbf{E}[\mathbf{e}_n] \xi), \mathbf{v}) \\ \mathbf{P}[\mathbf{p} : \mathbf{q}_0 ::= \mathbf{q}_1 \{ \mathbf{q}_0.s := \mathbf{e} \}] \rho \mathbf{t} \mathbf{v} &= \mathbf{E}[\mathbf{e}] (\langle \perp, \dots, \perp, \mathbf{v} \rangle) \\ \mathbf{E}[\mathbf{q}_j.s] \xi &= \xi \downarrow j, j = 1, \dots, n \\ \mathbf{E}[\mathbf{q}_0.i] \xi &= \xi \downarrow (n+1) \\ \mathbf{E}[\mathbf{c}_j] \xi &= \mathbf{c}_j \\ \mathbf{E}[\mathbf{a}_j(\mathbf{e}_1, \dots, \mathbf{e}_m)] \xi &= \text{lift}(\mathbf{a}_j)(\mathbf{E}[\mathbf{e}_1] \xi, \dots, \mathbf{E}[\mathbf{e}_m] \xi) \\ \mathbf{E}[\text{if } \mathbf{e}_1 \text{ then } \mathbf{e}_2 \text{ else } \mathbf{e}_3] \xi &= \text{if } \mathbf{E}[\mathbf{e}_1] \xi = \text{true} \text{ then } \mathbf{E}[\mathbf{e}_2] \xi \text{ else} \\ &\quad \text{if } \mathbf{E}[\mathbf{e}_1] \xi = \text{false} \text{ then } \mathbf{E}[\mathbf{e}_3] \xi \text{ else } \perp \end{aligned}$$

where

$$\begin{aligned} \text{lift}(\mathbf{f}) &= \lambda \vec{x}. \text{if some } x_j = \perp \text{ then } \perp \text{ else} \\ &\quad \text{if } \mathbf{f}(\vec{x}) \text{ undefined then } \perp \text{ else } \mathbf{f}(\vec{x}) \end{aligned}$$

The semantic function  $\mathbf{P}[\mathbf{ap}]$  takes a production environment  $\rho \in \mathbf{Env}$ , a tree  $\mathbf{t} \in \mathcal{T}_{\mathbf{G}}$ , and an inherited value  $\mathbf{v} \in \mathbf{D}$  and returns the value of the synthesised attribute.

The semantic function  $\mathbf{E}[\mathbf{e}]$  evaluates an expression from an environment of values of the  $\mathbf{n} + 1$  attributes it may depend on: the  $\mathbf{n}$  synthesised attributes to nonterminals on the right hand side and the inherited to left hand side.

The innermost fixpoint iteration is not as unpleasant as it may look: it is over the domain  $\mathbf{D}^{\mathbf{n}+1}$  which with  $\mathbf{D}$  as a flat cpo will have a finite height of  $\mathbf{n} + 2$ . The fixpoint can be found using a lazily evaluated **letrec**-expression.

By allowing conditional expressions in attribute expressions one may have attribute grammars with circular dependency graphs but where no attributes in practice are circularly defined. If an attribute is circularly defined the semantics will give the value  $\perp$ . The bottom element  $\perp$  is used both for runtime errors and for circularly defined attributes.

## 4 Strictness analysis

Strictness analysis ([17]) was developed as a method to detect when call-by-value could be used instead of call-by-need in lazy functional languages. In this section we will recall the central definitions in strictness analysis and show how they may be applied to attribute grammars.

### 4.1 Strictness of functions

Let  $\mathbf{f}$  be a function of type  $\mathbf{D}^{\mathbf{n}} \rightarrow \mathbf{D}$ . If the function satisfies

$$\forall \vec{\mathbf{v}} \in \mathbf{D}^{\mathbf{n}}. \mathbf{f}(\mathbf{v}_1, \dots, \mathbf{v}_{\mathbf{j}-1}, \perp, \mathbf{v}_{\mathbf{j}+1}, \dots, \mathbf{v}_{\mathbf{n}}) = \perp$$

for some  $\mathbf{j}$  then we know that at calls of  $\mathbf{f}$ , the  $\mathbf{j}^{\text{th}}$  argument may be evaluated prior to calling  $\mathbf{f}$ . This is because nontermination of the argument would have resulted in nontermination of  $\mathbf{f}$  anyway and we may therefore use call-by-value instead of call-by-need in function calls. We then say that  $\mathbf{f}$  is *strict* in its  $\mathbf{j}^{\text{th}}$  argument.

Strictness, however, is not a decidable property but by analysis we may find a sufficient condition for strictness. In this analysis we represent values in  $\mathbf{D}$  in a two-point domain  $\mathcal{Q}$  of the values 0 and 1

$$\mathcal{Q} = \{0, 1\}, \quad 0 \sqsubseteq 1$$

where  $\perp$  in  $\mathbf{D}$  is represented by 0 and all other values are represented by 1.

$$\alpha_1 : \mathbf{D} \rightarrow \mathcal{Q} \quad \alpha_1(\mathbf{d}) = \text{if } \mathbf{d} = \perp \text{ then } 0 \text{ else } 1$$

We may now construct a function  $\mathbf{f}^{\#}$  with the property

$$\forall \vec{\mathbf{v}} \in \mathbf{D}^{\mathbf{n}}. \alpha_1(\mathbf{f}(\vec{\mathbf{v}})) \sqsubseteq \mathbf{f}^{\#}(\alpha_1(\mathbf{v}_1), \dots, \alpha_1(\mathbf{v}_{\mathbf{n}}))$$

This is always possible as the constant function 1 is a candidate but better alternatives may be constructed with a little bit of ingenuity. If this function has the property that  $\mathbf{f}^\sharp(1, \dots, 1, 0, 1, \dots, 1) = 0$  where all arguments are 1 except at the  $\mathbf{j}^{\text{th}}$  place then  $\mathbf{f}$  is strict in its  $\mathbf{j}^{\text{th}}$  argument. We may say that for  $\mathbf{f}^\sharp$  a value of 1 means *possibly* defined while a value of 0 means *definitely* undefined.

The function  $\mathbf{f}^\sharp$  gives an upper bound to the termination properties of  $\mathbf{f}$ . In a similar fashion we may give a lower bound with a function  $\mathbf{f}^\flat$  with the property

$$\forall \vec{\mathbf{v}} \in \mathbf{D}^n. \alpha_1(\mathbf{f}(\vec{\mathbf{v}})) \sqsupseteq \mathbf{f}^\flat(\alpha_1(\mathbf{v}_1), \dots, \alpha_1(\mathbf{v}_n))$$

For this function a value of 1 means *definitely* defined while a value of 0 means *possibly* undefined. The function  $\mathbf{f}^\flat$ , however, is not quite as important when analysing functional programs.

## 4.2 Abstract domains

Below we define an interpretation for attribute grammars which evaluates the strictness properties of attribute expressions. As for functional programs it is based on abstractions of the values in the usual fixpoint semantics. We will use three abstraction functions, one for each of the arguments to the semantic function  $\mathbf{P}$ .

Attribute values are abstracted as the values 0 or 1 in the two-point domain  $\mathcal{2}$  using the abstraction function  $\alpha_1$  defined above.

Parse trees may be abstracted as productions where a production denotes the set of parse trees with the given production in the root node. When necessary productions may be abstracted as nonterminals where a nonterminal denotes the set of production with the nonterminal on the right hand side. The abstraction function for parse trees is:

$$\alpha_2 : \mathcal{T}_{\mathbf{G}} \rightarrow \mathbf{P} \quad \alpha_2(\mathbf{t}) = \mathbf{t} \downarrow 1$$

Production environments of type  $(\mathcal{T}_{\mathbf{G}} \rightarrow \mathbf{D} \rightarrow \mathbf{D})^{\mathbf{k}}$  can be abstracted in two ways to abstract environments  $\mathbf{Env}' = (\mathbf{P} \rightarrow \mathcal{2} \rightarrow \mathcal{2})^{\mathbf{k}}$ . The first will give an upper bound to the strictness properties of an environment and the other will give a lower bound.

$$\begin{aligned} \alpha_3^\sharp : \mathbf{Env} \rightarrow \mathbf{Env}' & \quad \alpha_3^\sharp(\rho) = \langle \alpha_4^\sharp(\rho \downarrow 1), \dots, \alpha_4^\sharp(\rho \downarrow \mathbf{k}) \rangle \\ & \quad \alpha_4^\sharp(\mathbf{f}) = \lambda \mathbf{p}. \lambda \mathbf{v}. \sqcup \{ \alpha_1(\mathbf{f} \mathbf{t} \mathbf{d}) \mid \alpha_2(\mathbf{t}) = \mathbf{p}, \alpha_1(\mathbf{d}) = \mathbf{v} \} \\ \alpha_3^\flat : \mathbf{Env} \rightarrow \mathbf{Env}' & \quad \alpha_3^\flat(\rho) = \langle \alpha_4^\flat(\rho \downarrow 1), \dots, \alpha_4^\flat(\rho \downarrow \mathbf{k}) \rangle \\ & \quad \alpha_4^\flat(\mathbf{f}) = \lambda \mathbf{p}. \lambda \mathbf{v}. \sqcap \{ \alpha_1(\mathbf{f} \mathbf{t} \mathbf{d}) \mid \alpha_2(\mathbf{t}) = \mathbf{p}, \alpha_1(\mathbf{d}) = \mathbf{v} \} \end{aligned}$$

The abstraction functions  $\alpha_3^\flat$  and  $\alpha_3^\sharp$  are defined as the usual lifting of abstraction functions to function domains.

$$\forall \rho : \alpha_3^\flat(\rho) \sqsubseteq \alpha_3^\sharp(\rho)$$

We will also need an abstraction function for attribute value environments:

$$\alpha_5 : \mathbf{D}^{\mathbf{n}+1} \rightarrow \mathcal{Q}^{\mathbf{n}+1} \quad \alpha_5(\xi) = \langle \alpha_1(\xi \downarrow 1), \dots, \alpha_1(\xi \downarrow (\mathbf{n} + 1)) \rangle$$

Using these abstraction functions we may define two new semantic functions  $\mathbf{P}^\sharp$  and  $\mathbf{P}^b$  which satisfy the following properties:

$$\begin{aligned} \forall \rho, \mathbf{t}, \mathbf{v} : \mathbf{P}^\sharp \llbracket \mathbf{ap} \rrbracket \alpha_3^\sharp(\rho) \alpha_2(\mathbf{t}) \alpha_1(\mathbf{v}) &\supseteq \alpha_1(\mathbf{P} \llbracket \mathbf{ap} \rrbracket \rho \mathbf{t} \mathbf{v}) \\ \forall \rho, \mathbf{t}, \mathbf{v} : \mathbf{P}^b \llbracket \mathbf{ap} \rrbracket \alpha_3^b(\rho) \alpha_2(\mathbf{t}) \alpha_1(\mathbf{v}) &\sqsubseteq \alpha_1(\mathbf{P} \llbracket \mathbf{ap} \rrbracket \rho \mathbf{t} \mathbf{v}) \end{aligned}$$

It is now possible to define the various semantic functions and establish their relationship with the standard semantics.

### 4.3 Analysis

We will here only define the semantic function  $\mathbf{M}^b$  (with  $\mathbf{P}^b$  and  $\mathbf{E}^b$ ). The semantic function  $\mathbf{M}^\sharp$  follows a similar pattern.

$$\begin{aligned} \mathbf{M}^b \llbracket \mathbf{ap}_1 \dots \mathbf{ap}_n \rrbracket &= \\ &\text{fix}(\lambda \rho. \langle \mathbf{P}^b \llbracket \mathbf{ap}_1 \rrbracket \rho, \dots, \mathbf{P}^b \llbracket \mathbf{ap}_n \rrbracket \rho \rangle) \\ \mathbf{P}^b \llbracket \mathbf{p} : \mathbf{q}_0 ::= \mathbf{q}_1 \dots \mathbf{q}_n \{ \mathbf{q}_1.\mathbf{i} := \mathbf{e}_1; \dots; \mathbf{q}_n.\mathbf{i} := \mathbf{e}_n; \mathbf{q}_0.\mathbf{s} := \mathbf{e}_{\mathbf{n}+1} \} \rrbracket \rho \mathbf{p}' \mathbf{v} &= \\ &\mathbf{E}^b \llbracket \mathbf{e}_{\mathbf{n}+1} \rrbracket (\text{fix} \lambda \xi. \langle \sqcap \{ \rho \downarrow \ell(\mathbf{p}_\ell) (\mathbf{E}^b \llbracket \mathbf{e}_1 \rrbracket \xi) \mid \ell : \text{lhs}(\mathbf{p}_\ell) = \text{rhs}(\mathbf{p}) \downarrow 1 \}, \dots, \\ &\quad \sqcap \{ \rho \downarrow \ell(\mathbf{p}_\ell) (\mathbf{E}^b \llbracket \mathbf{e}_n \rrbracket \xi) \mid \ell : \text{lhs}(\mathbf{p}_\ell) = \text{rhs}(\mathbf{p}) \downarrow \mathbf{n} \}, \mathbf{v} \rangle) \\ \mathbf{P}^b \llbracket \mathbf{p} : \mathbf{q}_0 ::= \mathbf{q}_1 \{ \mathbf{q}_0.\mathbf{s} := \mathbf{e} \} \rrbracket \rho \mathbf{p}' \mathbf{v} &= \mathbf{E}^b \llbracket \mathbf{e} \rrbracket (\langle \perp, \dots, \perp, \mathbf{v} \rangle) \\ \mathbf{E}^b \llbracket \mathbf{q}_j.\mathbf{s} \rrbracket \xi &= \xi \downarrow \mathbf{j} \\ \mathbf{E}^b \llbracket \mathbf{q}_0.\mathbf{i} \rrbracket \xi &= \xi \downarrow (\mathbf{n} + 1) \\ \mathbf{E}^b \llbracket \underline{\mathbf{c}}_j \rrbracket \xi &= 1 \\ \mathbf{E}^b \llbracket \underline{\mathbf{a}}_j(\mathbf{e}_1, \dots, \mathbf{e}_m) \rrbracket \xi &= \min(\mathbf{E}^b \llbracket \mathbf{e}_1 \rrbracket \xi, \dots, \mathbf{E}^b \llbracket \mathbf{e}_m \rrbracket \xi) \\ \mathbf{E}^b \llbracket \text{if } \mathbf{e}_1 \text{ then } \mathbf{e}_2 \text{ else } \mathbf{e}_3 \rrbracket \xi &= \min(\mathbf{E}^b \llbracket \mathbf{e}_1 \rrbracket \xi, \mathbf{E}^b \llbracket \mathbf{e}_2 \rrbracket \xi, \mathbf{E}^b \llbracket \mathbf{e}_3 \rrbracket \xi) \end{aligned}$$

### 4.4 Correctness

As correctness proof for this analysis we will prove that for any attribute grammar  $\mathbf{ag}$  the following condition holds.

$$\mathbf{M}^b \llbracket \mathbf{ag} \rrbracket \sqsubseteq \alpha_3^b(\mathbf{M} \llbracket \mathbf{ag} \rrbracket)$$

There are two levels of fixpoint induction involved in the proof since the semantics contains two nested fixpoints.

Correctness for the inner level is obtained automatically as the interpretation of the basic operations can be induced using the abstraction function  $\alpha_1$ :

$$\forall \xi. \mathbf{E}^b \llbracket \mathbf{e} \rrbracket \alpha_5(\xi) \sqsubseteq \alpha_1(\mathbf{E} \llbracket \mathbf{e} \rrbracket \xi)$$

In the next level we will prove that:

$$\forall \rho, \mathbf{t}, \mathbf{v}. \mathbf{P}^b \llbracket \mathbf{ap} \rrbracket \alpha_3^b(\rho) \alpha_2(\mathbf{t}) \alpha_1(\mathbf{v}) \sqsubseteq \alpha_1(\mathbf{P} \llbracket \mathbf{ap} \rrbracket \rho \mathbf{t} \mathbf{v})$$

This requires fixpoint induction. The proof requires the following relationship.

$$\begin{aligned} \forall \rho, \xi, \mathbf{t}, \mathbf{j}. \sqcap \{ \alpha_3^b(((\rho) \downarrow \ell) \mathbf{p}_\ell) (\mathbf{E}^b \llbracket \mathbf{e}_j \rrbracket \alpha_5(\xi)) | \ell : \text{lhs}(\mathbf{p}_\ell) = \text{rhs}(\mathbf{p}) \downarrow \mathbf{j} \} \\ \sqsubseteq \alpha_1(((\rho) \downarrow \mathbf{i}_j) \mathbf{t}_j) (\mathbf{E} \llbracket \mathbf{e}_j \rrbracket \xi) \\ \text{where } \langle \mathbf{p}, \mathbf{t}_1, \dots, \mathbf{t}_n \rangle = \mathbf{t} \text{ and } \langle \mathbf{p}_{i_j}, \dots \rangle = \mathbf{t}_j \end{aligned}$$

For a start we notice that

$$\text{rhs}(\mathbf{p}) \downarrow \mathbf{j} = \text{lhs}(\mathbf{t}_j \downarrow 1) = \text{lhs}(\mathbf{p}_{i_j})$$

This means that

$$\begin{aligned} \sqcap \{ \alpha_3^b(((\rho) \downarrow \ell) \mathbf{p}_\ell) (\mathbf{E}^b \llbracket \mathbf{e}_j \rrbracket \alpha_5(\xi)) | \ell : \text{lhs}(\mathbf{p}_\ell) = \text{rhs}(\mathbf{p}) \downarrow \mathbf{j} \} \\ \sqsubseteq \alpha_3^b(((\rho) \downarrow \mathbf{i}_j) \mathbf{p}_{i_j}) (\mathbf{E}^b \llbracket \mathbf{e}_j \rrbracket \alpha_5(\xi)) \\ = \sqcap \{ \alpha_1(((\rho) \downarrow \mathbf{i}_j) \mathbf{t}' \mathbf{d}) | \alpha_2(\mathbf{t}') = \mathbf{p}_{i_j}, \alpha_1(\mathbf{d}) = \mathbf{E}^b \llbracket \mathbf{e}_j \rrbracket \alpha_5(\xi) \} \\ \sqsubseteq \alpha_1(((\rho) \downarrow \mathbf{i}_j) \mathbf{t}_j) (\mathbf{E} \llbracket \mathbf{e}_j \rrbracket \xi) \end{aligned}$$

Both levels of fixpoint induction follow now directly.

## 4.5 Using the analysis

Strictness analysis for attribute grammars can give two kinds of information. It can identify those productions where the inherited attribute is definitely needed to evaluate the synthesised attribute, and it can find the productions where the synthesised attribute can be evaluated without using the inherited attribute.

This may be expressed using the semantic functions as

$$(\mathbf{M}^\# \llbracket \mathbf{ag} \rrbracket \downarrow \mathbf{j}) \mathbf{p}_j 0 = 0$$

which states that the inherited attribute is needed to evaluate the synthesised in the  $\mathbf{j}^{\text{th}}$  production; and

$$(\mathbf{M}^b \llbracket \mathbf{ag} \rrbracket \downarrow \mathbf{j}) \mathbf{p}_j 0 = 1$$

which says that the inherited attribute is not used for evaluating the synthesised in the  $\mathbf{j}^{\text{th}}$  production.

An attributed production in a grammar may not satisfy either of these properties as the inherited attribute may be needed for some input but not used for other input. As an example consider

$$\begin{aligned} \mathbf{A} &\rightarrow \text{“a”} \mathbf{B} \text{“d”} && \{ \mathbf{A.s} := 3 + \mathbf{B.s} \\ & && \mathbf{B.i} := \mathbf{A.i} + 2 \} \\ \mathbf{B} &\rightarrow \text{“b”} && \{ \mathbf{B.s} := \mathbf{B.i} + 1 \} \\ \mathbf{B} &\rightarrow \text{“c”} && \{ \mathbf{B.s} := 0 \} \end{aligned}$$

where the inherited attribute to  $\mathbf{A}$  is needed for the string “abd” but not used for the string “acd”.

## 5 Evaluation order

From the strictness analysis we may derive other interpretations which analyse the order of evaluation in an attribute grammar. One example of this type of analysis is given below. It will enable us to determine when attributes can be evaluated during left-to-right parsing and it is obtained from the strictness analysis by ensuring that expressions defining inherited attributes may not use the values of the synthesised attribute to the right in a production.

$$\begin{aligned} \mathbf{M}^\triangleright[\mathbf{ap}_1 \dots \mathbf{ap}_n] &= \\ &\text{fix}(\lambda\rho. \langle \mathbf{P}^\triangleright[\mathbf{ap}_1]\rho, \dots, \mathbf{P}^\triangleright[\mathbf{ap}_k]\rho \rangle) \\ \\ \mathbf{P}^\triangleright[\mathbf{p} : \mathbf{q}_0 ::= \mathbf{q}_1 \dots \mathbf{q}_n \{ \mathbf{q}_1.i := \mathbf{e}_1; \dots; \mathbf{q}_n.i := \mathbf{e}_n; \mathbf{q}_0.s := \mathbf{e}_{n+1} \}] \rho \mathbf{p}' \mathbf{v} &= \\ &\mathbf{E}^\triangleright[\mathbf{e}_{n+1}] (\text{fix} \lambda\xi. \langle \prod \{ \rho \downarrow \ell(\mathbf{p}_\ell) (\mathbf{E}^\triangleright[\mathbf{e}_1] \text{strip}_1(\xi)) \mid \ell : \text{lhs}(\mathbf{p}_\ell) = \text{rhs}(\mathbf{p}) \downarrow 1 \}, \dots, \\ &\quad \prod \{ \rho \downarrow \ell(\mathbf{p}_\ell) (\mathbf{E}^\triangleright[\mathbf{e}_n] \text{strip}_n(\xi)) \mid \ell : \text{lhs}(\mathbf{p}_\ell) = \text{rhs}(\mathbf{p}) \downarrow n \}, \mathbf{v} \rangle) \\ \mathbf{P}^\triangleright[\mathbf{p} : \mathbf{q}_0 ::= \mathbf{q}_1 \{ \mathbf{q}_0.s := \mathbf{e} \}] \rho \mathbf{p}' \mathbf{v} &= \mathbf{E}^\triangleright[\mathbf{e}] (\langle \perp, \dots, \perp, \mathbf{v} \rangle) \\ \\ \mathbf{E}^\triangleright[\mathbf{e}] \xi &= \mathbf{E}^\flat[\mathbf{e}] \xi \end{aligned}$$

where

$$\text{strip}_j(\langle \mathbf{v}_1, \dots, \mathbf{v}_n, \mathbf{v}_{n+1} \rangle) = \langle \mathbf{v}_1, \dots, \mathbf{v}_{j-1}, \perp, \dots, \perp, \mathbf{v}_{n+1} \rangle$$

The soundness of this analysis follows directly from the strictness analysis due to the monotonicity of the semantic functions:

$$\mathbf{M}^\triangleright[\mathbf{ag}] \sqsubseteq \mathbf{M}^\flat[\mathbf{ag}]$$

## 5.1 Using the analysis

Using the result of this analysis we may identify the productions where the synthesised attribute may be evaluated during LR-parsing if the inherited attribute is evaluated.

In the semantics this may be expressed as the condition

$$(\mathbf{M}^{\triangleright}[\mathbf{ag}] \downarrow \mathbf{j})\mathbf{p}_j 1 = 1$$

which says that in the  $\mathbf{j}^{\text{th}}$  production the synthesised attribute may be evaluated if the inherited is defined.

## 5.2 Comparison

Our analysis may be seen as an extension to the concept of an L-attribute grammar ([1]). Whether it is possible to insert semantic actions into productions and preserve LR(1) (or LALR(1)) properties is not expressed by this analysis. That has been addressed by [18] where they find safe positions in productions for such actions to be inserted.

The analysis in [9] examines the structure of the LR-parse table and the attribute grammar and it can be used to find a list of attributes which can be evaluated (known) during LR-parsing. If this list contains all attributes one may conclude that it is an L-attribute grammar.

The life time analysis in [14] can be used to identify attributes which can be stored in a global variable or using a stack. The analysis is applicable to ordered attribute grammars and is based on transformations of the visit-sequences for the attribute grammar.

In both [9] and [14] the analysis will associate properties with each attribute in the grammar. Our analysis may give a finer grained information in that it examines the attributes in each production. It may be possible to evaluate an attribute during parsing in one production but not in other productions.

## 5.3 Example

A number of questions about the computational behaviour of an attribute grammar can be expressed using these interpretations.

In the attribute grammar below we will allow two synthesised attributes and two inherited attributes to the nonterminal  $\mathbf{Y}$ .

$$\begin{array}{ll}
 \mathbf{S} \rightarrow \mathbf{Y} & \{ \mathbf{Y.a} := 1 \\
 & \mathbf{Y.c} := \mathbf{f}_1(\mathbf{Y.b}) \\
 & \mathbf{S.x} := \mathbf{Y.d} \} \\
 \mathbf{Y}_1 \rightarrow \text{"x"} \mathbf{Y}_2 & \{ \mathbf{Y}_1.\mathbf{b} := \mathbf{f}_2(\mathbf{Y}_1.\mathbf{a}, \mathbf{Y}_2.\mathbf{b}) \\
 & \mathbf{Y}_2.\mathbf{a} := \mathbf{f}_3(\mathbf{Y}_1.\mathbf{a}) \\
 & \mathbf{Y}_2.\mathbf{c} := \mathbf{f}_4(\mathbf{Y}_1.\mathbf{c}, \mathbf{Y}_2.\mathbf{b}) \}
 \end{array}$$

$$\begin{array}{l}
\mathbf{Y} \rightarrow \text{“z”} \\
\mathbf{Y}_1.\mathbf{d} := \mathbf{f}_5(\mathbf{Y}_2.\mathbf{d}, \mathbf{Y}_1.\mathbf{c}) \\
\{ \mathbf{Y}.\mathbf{b} := \mathbf{f}_6(\mathbf{Y}.\mathbf{a}) \\
\mathbf{Y}.\mathbf{d} := 3 \}
\end{array}$$

In this example the attribute  $\mathbf{Y}_1.\mathbf{b}$  can be evaluated during LR-parsing in the second production and the attributes  $\mathbf{Y}.\mathbf{b}$  and  $\mathbf{Y}.\mathbf{d}$  can be evaluated during LR-parsing in the third production. This is under the assumption that the inherited attribute  $\mathbf{a}$  to  $\mathbf{Y}$  can be evaluated during parsing. Using the semantic function  $\mathbf{M}^\triangleright$  this may be expressed as

$$(\mathbf{M}^\triangleright[\cdot] \downarrow 2)(\mathbf{Y}_1 \rightarrow \text{“x”} \mathbf{Y}_2)(1, 0) = \langle 1, 0 \rangle$$

and

$$(\mathbf{M}^\triangleright[\cdot] \downarrow 3)(\mathbf{Y} \rightarrow \text{“z”})(1, 0) = \langle 1, 1 \rangle$$

## 5.4 Implementation

The strictness and evaluation order analyses have been implemented in a demonstrator system. The system uses a DAG-based strategy for attribute evaluation and the analyses can decrease the size of the DAG by allowing evaluation of some attribute values during parsing.

## 6 Further work

The analyses described here can be extended in several ways. It is not possible in this framework to give a satisfactory analysis of storage management in attribute grammars. Doing that using abstract interpretation is likely to require an instrumented standard semantics. Such extensions are frequently seen in analysis of functional and logic programming languages.

The treatment of inherited and synthesised attributes in this framework is somewhat asymmetrical. A better analysis of inherited attributes may be obtained if the semantics is based on a minimal function graph framework ([10]). This, however, is not easy as that technique was developed for an eager language and attribute evaluation uses laziness. The extension of the minimal function graph framework to a first-order lazy functional language is considered by [20].

Recent work in strictness analysis have examined extensions of the analysis to languages with higher-order functions and lazy data structures. Whether such results may be applicable to attribute grammars seems to be an open question. Attribute values are normally considered to be simple values in sets but the extension to allow higher-order functions and lazy data structures as attribute values is semantically well understood.

## 7 Conclusion

In this paper we have presented a framework for semantically based analysis of attribute grammars. The framework is used to define a strictness analysis for attribute grammars which is proved correct with respect to a standard fixpoint semantics. The analysis has been implemented as part of an attribute evaluation system.

The framework may be the basis for a number of analyses of attribute grammars. Attributes that may be evaluated during LR-parsing may be identified by an evaluation-order analysis for attributes.

Perhaps the most surprising result of this work is that Attribute Grammars as a language is well suited to semantic-based program analysis. It is a clean and relatively simple language with a tractable fixpoint semantics. Furthermore, “programs” in Attribute Grammars are typically run frequently enough for nearly any type of program optimisation to be profitable.

**Acknowledgements.** Thanks to Alan Mycroft, Gordon Gran, Helen Hansen, Troy Ferguson and the anonymous referees for comments on drafts of this paper. This work was partly supported by the Danish SNF Dart grant.

## References

- [1] A V Aho, R Sethi, and J D Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 1986.
- [2] L M Chirica and D F Martin. *An Order-Algebraic Definition of Knuthian Semantics*. Math. Systems Theory **13**, pp. 1–27, 1979.
- [3] H Christiansen. *Structure Sharing in Attribute Grammars*. In PLILP’88, Orléans (P Deransart, B Lorho, and J Maluszynski, eds.), pp. 180–200. Volume 348 of LNCS. Springer-Verlag, May, 1988.
- [4] P Cousot and R Cousot. *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. In 4th POPL, Los Angeles, CA, pp. 238–252, Jan., 1977.
- [5] P Deransart, M Jourdan, and B Lorho. *Attribute grammars. Definitions, systems, and bibliography*. Volume 323 of LNCS. Springer-Verlag, 1988.
- [6] J Engelfriet. *Attribute Grammars: Attribute Evaluation Methods*. In Methods and Tools for Compiler Construction (B Lorho, ed.), pp. 103–138. Cambridge Univ. Press, 1984.
- [7] J Gallier. *An Efficient Evaluator for Attribute Grammars with Conditionals*. Tech. Rep. MS-CIS-83-36. Philadelphia, PA, May, 1984.

- [8] M J C Gordon. *The Denotational Description of Programming Languages: An Introduction*. Springer-Verlag, 1979.
- [9] N D Jones and M Madsen. *Attribute-influenced LR parsing*. In *Semantics-Directed Compiler Generation* (N D Jones, ed.), pp. 393–407. Volume 94 of LNCS. Springer-Verlag, Jan., 1980.
- [10] N D Jones and A Mycroft. *Data Flow Analysis of Applicative Programs using Minimal Function Graphs*. In 13th POPL, St. Petersburg, Florida, pp. 296–306, Jan., 1986.
- [11] M Jourdan. *Recursive Evaluators for Attribute Grammars: An Implementation*. In *Methods and Tools for Compiler Construction* (B Lorho, ed.), pp. 139–163. Cambridge Univ. Press, 1984.
- [12] M Jourdan, C L Bellec, and D Parigot. *The OLGA Attribute Grammar Description Language*. In WAGA’90 (P Deransart and M Jourdan, eds.), pp. 222–237. Volume 461 of LNCS. Springer-Verlag, Oct., 1990.
- [13] U Kastens. *The GAG-System—A Tool for Compiler Construction*. In *Methods and Tools for Compiler Construction* (B Lorho, ed.), pp. 165–182. Cambridge Univ. Press, 1984.
- [14] U Kastens. *Lifetime Analysis for Attributes*. *Acta Inf.* **24**(6), pp. 633–652, Nov., 1987.
- [15] D E Knuth. *Semantics of Context-Free Languages*. *Math. Systems Theory* **2**(2), pp. 127–145, June, 1968. Correction *ibid* 5(1):95–96 Mar. 1971.
- [16] A Mycroft. *The Theory and Practice of Transforming Call-by-Need into Call-by-Value*. In *International Symposium on Programming’80, Paris, France* (B Robinet, ed.), pp. 269–281. Volume 83 of LNCS. Springer-Verlag, Apr., 1980.
- [17] A Mycroft. *Abstract Interpretation and Optimising Transformations for Applicative Programs*. Ph.D. Thesis. Univ. of Edinburgh, Dec., 1981.
- [18] P W Purdom and C A Brown. *Semantic routines and LR(k) parsers*. *Acta Inf.* **14**(4), pp. 299–316, Oct., 1980.
- [19] M Rosendahl. *Abstract Interpretation and Attribute Grammars*. Ph.D. Thesis. Cambridge Univ., 1992.
- [20] M Rosendahl and A Mycroft. *Minimal function graphs for call-by-need*. Unpublished. DIKU, Univ. of Copenhagen, Denmark, 1992.
- [21] M Sassa, H Ishizuka, and I Nakata. *ECLR-attributed Grammars: a Practical Class of LR-attributed Grammars*. *Inform. Process. Lett.* **24**(1), pp. 31–41, Jan., 1987.

- [22] D A Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, Newton, MA, 1986.
- [23] D A Watt and O L Madsen. *Extended Attribute Grammars*. *Comput. J.* **26**(2), pp. 142–153, 1983.