Linear Data Structures for Storage Allocation in Attribute Evaluators

F. J. Sluiman

Linear Data Structures for Storage Allocation in Attribute Evaluators

ISBN 90-365-1713-3

Copyright © 2000, 2001 by F. J. Sluiman, Amsterdam, Netherlands.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without written permission of the author, F. J. Sluiman, P.O. Box 6034, 9702 HA Groningen, Netherlands.

The author was sponsored by the Netherlands Computer Science Research Foundation (SION) with financial support from the Netherlands Organisation for Scientific Research (NWO).

Linear Data Structures for Storage Allocation in Attribute Evaluators

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. F. A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 1 maart 2002 te 16.45 uur

door

luitenant ter zee van de elektrotechnische dienst der tweede klasse
der Koninklijke Marine-reserve
ingenieur

Frederik Jan Sluiman

geboren op 24 februari 1965
te Groningen

Dit proefschrift is goedgekeurd door de promotoren

prof. dr. H. Brinksma

prof. dr. ir. A. Nijholt

en de assistent-promotor

dr. H. Alblas

SAMENVATTING

Praktische en theoretische resultaten zijn gevonden betreffende het gebruik van globale ruimte-allocatie voor de instanties van toegepaste voorkomens van een attribuut.

De praktische resultaten hebben voornamelijk betrekking op de noodzakelijke en voldoende voorwaarden om tijdens de constructie van de evaluator te kunnen beslissen of deze de instanties van een toegepast voorkomen kan toewijzen aan een aantal globale variabelen, stacks en queues. Het testen van deze voorwaarden neemt polynomiaal veel tijd voor een simple multi-visit evaluator en exponentieel veel tijd voor een absolutely non-circular evaluator.

De theoretische resultaten hebben betrekking op de gegevensstructuren die nodig zijn voor de globale ruimte-allocatie van de instanties van toegepaste voorkomens in simple multi-X evaluatoren, waarbij $X \in \{\text{pass, sweep, visit}\}$. Voor dit doel is de algemene klasse van basic linear data structures geïntroduceerd. Deze klasse van gegevensstructuren kan ook worden gebruikt om de theoretische mogelijkheden en beperkingen van ruimte-allocatietechnieken in andere gebieden dan attribuutgrammatica's te onderzoeken.

SUMMARY

Practical and theoretical results have been found concerning the use of global storage allocation for the instances of applied occurrences of an attribute.

The practical results focus on the necessary and sufficient conditions to decide at evaluator construction time whether an evaluator can allocate the instances of an applied occurrence to a number of global variables, stacks and queues. Checking these conditions takes polynomial time for a simple multi-visit evaluator and exponential time for an absolutely non-circular evaluator.

The theoretical results are concerned with the data structures that are required for the global storage allocation of the instances of applied occurrences in simple multi-X evaluators, where $X \in \{\text{pass, sweep, visit}\}$. For this purpose, the general class of basic linear data structures is introduced. This class of data structures can also be used to explore the theoretical possibilities and limitations of storage allocation techniques in domains other than attribute grammars.

dedicated to my parents

ABOUT THE AUTHOR

Frederik Jan Sluiman was born and raised in Groningen, Netherlands. He received a Master of Science degree (with distinction) in computer science from the University of Twente. After a research position of four years at the University of Twente he joined the Royal Netherlands Navy in the rank of Lieutenant to work on command and control systems for naval frigates using automatic object-oriented development techniques. Sluiman is currently Publishing Technology Manager at Elsevier Science where he is engaged in the development of Internet products. He enjoys a wide range of interests including navigation, shipbuilding and the history of the American Wild West. To keep fit, he enjoys a number of sports such as fencing, sailing and parachuting.

Preface

This booklet completes my investigations on storage allocation for simple multi-visit evaluators. These investigations started in 1989 after Rieks op den Akker made me enthusiastic for the topic of storage allocation in attribute evaluators. At that time variables and stacks were generally considered to be the only data structures suitable for global storage allocation in attribute evaluators. Since I suspected that other data structures like queues might be suitable as well, I started constructing attribute grammars that contained attributes whose instances could be allocated to queues during evaluation. When I succeeded in construction such attribute grammars I also noticed that it would make sense to refine storage allocation to the instances of an applied occurrence of an attribute.

The work of Engelfriet and De Jong was of great importance to the next step in my research. Their findings, which had just been published in a report preceding [10], were surrounded by rumors that they were erroneous. What may have caused these rumors was presumably that the report made use of a subtree lifetime set and a context lifetime set. This use of a context lifetime

set was unnecessary (and omitted in [10]) but not wrong. As such, I found no serious errors and walking in their footsteps I was able to formulate necessary and sufficient conditions to decide whether a simple multi-visit evaluator can allocate the instances of an applied occurrence to a global variable, stack or queue. These conditions were published in [33] and suffer a similar flaw as the earlier ones of Engelfriet and De Jong.

Meanwhile I got the impression that for simple multi-pass evaluators the possible data structures for $(a, a+1)$-allocating instances of applied occurrences were limited to global variables, stacks, and queues. A proof of this extraordinary fact, however, was not so easy to establish so I welcomed Rieks op den Akker's generous offer to assist me. Jointly we worked on the further formalization of basic linear data structures, a concept that I introduced to characterize the data structures appropriate for the allocation of instances of single-use applied occurrences. Engaged in this work (during which I had studied the revised report [10]) I also worked on conditions to decide whether a simple multi-visit evaluator can (a, b)-allocate the instances of a applied occurrence to a global stack or queue. The conditions that I found could be specialized to simple multi-sweep evaluators so as to make it possible to analyze what happens if a simple multi-pass evaluator cannot allocate the instances of a single-use applied occurrence to a global stack or queue. Thus as one result led to another I got all the evidence necessary to be able to complete the proof.

All the results found during these investigations were published in a report, which was unfortunately not well received. The main reason why this happened was the difficult presentation of this rather detailed material. Guided by the profound comments and suggestions of Joost Engelfriet, I have gone to great lengths explaining each new concept while using notations as

economically and consistently as possible. In addition, I also undertook new investigations regarding the relation between the evaluation strategy followed by an attribute evaluator and the possible global data structures to which the instances of applied occurrences can be $(a, a+1)$-allocated. The new results found were included in the revised work which is presented here.

This is essentially the genesis of this dissertation. I hope that my work serves those who are interested in its topic, either by supplying solutions or by opening the way for new investigations.

Audience

The material contained in this dissertation is intended for readers with a firm understanding of attribute grammars and evaluators. Since it not reasonable to expect that a reader gains this knowledge from an introductory chapter in this dissertation, I did not attempt to make this material accessible to a broader audience. These readers are advised to first study [2,6,8,9,19,23] before reading this work.

Conventions

The beginning of structures as theorems, lemmas, and definitions is marked by the name of the structure optionally followed by a number for referential purposes. Both the name of the structure and the optional number are printed in a clearly distinguishable type style (italic or boldface).

The square symbol $\square$ is used to mark the ending of these structures.

Acknowledgements

This manuscript has seen many years of evolution and refinement during which I was employed at different locations.

At the University of Twente, I wish to thank Rieks op den Akker, Henk Alblas, Robert Huis in 't Veld, Joost-Pieter Katoen, Harro Kremer, Albert Nijmeijer, and Vincent Zweije. Of these men, two deserve special note. Henk Alblas for offering me a research position. His continuous support during all my hardships is highly appreciated. Rieks op den Akker deserves special note for introducing me to the field of attribute grammars. His enthusiasm and supervision were second to none. I enjoyed working with him.

At the Royal Netherlands Navy, I thank Phons Bloemen for installing the TEXtelm^{e}xtel shell on my notebook and David Wilson for meticulously proofreading an early draft of this booklet.

At Elsevier Science, Chris Leonard brushed up the English and Karel de Vos provided me with the *WinEdt* shell by which the final copy of this dissertation was produced. To them I am grateful.

Finally, I wish to give my special thanks to Joost Engelfriet. His valuable comments and suggestions greatly improved this work.

Amsterdam, Netherlands — F. J. Sluiman
September 2000

PREFACE TO THE REVISION

The present booklet is a revision of the draft dissertation that I submitted to my supervisors. This revision was mainly necessitated by their wish for a more elaborate literature survey on $\mathcal{G}1$ global storage allocation techniques. During the revisions, I discovered conditions to decide whether an absolutely non-circular attribute evaluator can (a, b)-allocate the instances of an applied

occurrence to a global stack or queue. As these conditions completed the practical results of my work, I decided to add them in a postscript to this dissertation. The resulting postscript assumes a thorough understanding of absolutely non-circular attribute evaluators. Readers that are not familiar with these evaluators are referred to [3,5-7,29].

More acknowledgements

I would like to thank professors Ed Brinksma and Anton Nijholt for their kind willingness to act as my promotors and their patience in the completion of this revision.

Amsterdam, Netherlands — F. J. Sluiman

August 2001

Contents

CHAPTER ONE

Introduction 1

CHAPTER TWO

Attribute Grammars and Evaluators 13

CHAPTER THREE

Global Storage Allocation of Single-use Applied Occurrences . . . 25

CHAPTER FOUR

Global Storage Allocation of Multi-use Applied Occurrences . . . 67

CHAPTER FIVE

Implementation Aspects 93

CHAPTER SIX

Conclusions 105

POSTSCRIPT

Stacks and Queues for
Absolutely Non-circular Attribute Evaluators 111

References 129
Index of Symbols 135
Index of Definitions 139

CHAPTER ONE

Introduction

When Knuth proposed attribute grammars in [23] as a way of assigning meaning to derivation trees of context-free grammars, the computability of the attribute values in the derivation trees played a mayor role in the acceptance of the formalism. To this end, he presented an algorithm (containing an error corrected a few years later) to decide whether, for every derivation tree of an attribute grammar, there is a total order in which the attribute values can be computed.[1]

Attribute grammars were soon found to be a useful tool for describing compilers and translators (see also [1,6]) but the implemented attribute evaluators computing the attribute values were generally considered to be too inefficient. Especially the time consumed by these evaluators to determine the total order in which the attribute values are computed was felt to be something that could be avoided. Consequently, the main challenge to re-

[1] In this context it is interesting to note that four years after correction of the algorithm, the time complexity of the corrected algorithm was shown to be inherently exponential by Jazayeri, Ogden, and Rounds [14].

searchers was the development of attribute evaluators which determined this total order for all derivation trees at evaluator construction time.

Many such time-efficient attribute evaluators were developed: simple multi-pass evaluators making a fixed number of left-to-right passes by Bochmann [4], simple multi-pass evaluators making alternately left-to-right and right-to-left passes by Jazayeri and Walter [16], and tree-walk evaluators for absolutely non-circular attribute grammars by Kennedy and Warren [22] for example. To master the complexity, it was convenient to assume that the computed attribute values could be stored in the nodes of a derivation tree. This simple storage allocation technique, which had been applied in most of the attribute evaluators implemented at that time, was a logical step in the development of these time-efficient evaluators. However, shortly after the implementation of the first time-efficient attribute evaluators, storage space problems began to emerge. Therefore, researchers started to investigate more efficient storage allocation techniques.

Various storage allocation techniques were proposed to achieve more efficient use of storage space. The basic principle of all these techniques nonetheless being the same:

> *re-use of space occupied by attribute values which are only stored for the computation of dependent attribute instances.*

This re-use of space is obtained by the allocation of the instances of particular attributes to global variables and stacks (referred to as global storage allocation), or to variables assigned to certain regions in a derivation tree (referred to as local storage allocation).

Global storage allocation techniques found in the literature are divisible into three groups:

- $\mathcal{G}1$: Global storage allocation techniques in which allocation is decided at evaluator construction time by analyzing the evaluation strategy of a given time-efficient evaluator. These techniques, found in [10,12,13,15, 18,20,32], are independent of specific derivation trees and leave the evaluation strategy of the given time-efficient evaluator unchanged.
- $\mathcal{G}2$: Global storage allocation techniques in which allocation is decided at evaluator construction time for a given set of attributes. These techniques, found in [31,34], are independent of specific derivation trees and construct a time-efficient evaluator with an evaluation strategy such that the space occupied by attribute instances of the given set of attributes is re-used.
- $\mathcal{G}3$: Global storage allocation techniques in which allocation is decided at evaluator run time for a particular derivation tree. These techniques, found in [25,27,30], can be used with all kinds of attribute evaluators although it increases their time consumption.

Local storage allocation techniques are found in [28] only. The technique described there decides allocation at evaluator construction time by analyzing the attribute grammar. It does not depend on specific derivation trees and can be used with any type of attribute evaluator.

A completely different direction towards storage allocation of computed attribute values is taken by Pugh in [26]. He proposed function caching for the re-computation of attributes values in derivation trees which have undergone incremental changes. When the result of a semantic function call is computed, the call and the result (*i.e.*, an attribute value) are stored in the function cache so that the result can be retrieved the next time this call is made. This implementation not only avoids re-computation of previously

computed semantic function calls, but it might save storage space as well. The basic principle by which storage space is saved is stated as:

> *sharing of space occupied by attribute values which are computed by identical semantic function calls.*

Pugh's way of allocating attribute values to the function cache is therefore an advantageous storage allocation strategy for the (re-)computation of the attribute values in derivation trees that contain lots of isomorphic subtrees. As such, it was adapted by Vogt, Swierstra, and Kuiper [35] for use with simple multi-visit attribute evaluators of higher order attribute grammars. These grammars, introduced in [36] as an extension of attribute grammars, have derivation trees with lots of isomorphic subtrees so that efficient use of time and storage space is guaranteed.

However, in comparison with the approach of reusing storage space, sharing storage space does not generally lead to more efficient use of storage space. After all, arbitrary attribute grammars are not likely to have lots of derivation trees with isomorphic subtrees, but they are likely to have many attributes whose values are only stored for the computation of dependent attribute instances.

This dissertation explores a global storage allocation technique of the first group, $\mathcal{G}1$, for use with simple multi-visit evaluators. In order to place this work in the proper context, there follows a literature survey on $\mathcal{G}1$ allocation techniques. Thereafter the global storage allocation technique examined in this dissertation will be discussed in detail.

The first $\mathcal{G}1$ global storage allocation technique was introduced in 1978 by Saarinen [32]. In trying to reduce the run time space consumption of

absolutely non-circular attribute evaluators, he classified some attributes as temporary and allocated their instances to a single global stack. This classification, however, does not detect all possible attributes whose instances could be allocated to the global stack. Improving it requires a more thorough analysis at evaluator construction time, which is rather difficult (the order in which absolutely non-circular attribute evaluators compute attribute instances, determined at evaluator construction time, depend on the derivation trees to be evaluated).

A year later, Ganzinger [13] introduced another $\mathcal{G}1$ global storage allocation technique. He proposed to allocate the instances of particular attributes to global variables so that storage space is re-used and copy operations are eliminated (when the instances of attributes α and β are allocated to the same global variable, copy operations involved with semantic rules $(\alpha, p, j) = (\beta, p, k)$ can be eliminated as they only copy the value of the global variable to itself). Sufficient conditions were given to decide whether the instances of particular attributes can be allocated to global variables and it was shown that finding an allocation that optimizes the elimination of copy operations is NP-complete. His conditions can be applied with all types of time efficient evaluators for which the so-called minimal LAST sets can be computed. Ganzinger mentions that their constructability has not been fully investigated but that they are very easy to compute for simple multi-visit evaluators and some subclasses. Further details were not provided.

In 1981, Yazayeri and Pozefsky [15] applied the technique of Saarinen to simple multi-pass evaluators by classifying certain attributes as one-pass and allocating their instances to a single global stack. Additionally, they considered the allocation of the instances of multi-pass attributes. Amongst other techniques a $\mathcal{G}1$ allocation technique was proposed which assumes an

“attribute area” where storage space is allocated and freed by a simple multi-pass evaluator in the following manner:

— Just before visiting a node n, all multi-pass attribute instances of node n, that are computed during this visit, are allocated to the attribute area.

— After visiting a node n, all multi-pass attribute instances of node n, that are no longer needed for the computation of other instances, are freed from the attribute area.

This simple $\mathcal{G}1$ allocation technique has two disadvantages. The first is that the address of each instance within the attribute area must be stored in the derivation tree in order to be able to access it. This disadvantage was overcome by restricting the use of this technique to instances of multi-pass attributes with large storage space requirements where the space savings are worth the costs of storing addresses. The second and main disadvantage that was not overcome, however, is that the attribute area is not a data structure tailored to the order in which instances are allocated and freed. This causes the attribute area to become fragmented, which increases the storage space requirements of the attribute area and, ultimately, of the attribute evaluator.

Farrow and Yellin [12] compared in 1986 the efficiency of various storage optimizations applied by two compiler generator systems. These systems both used $\mathcal{G}1$ allocation techniques to generate simple multi-visit evaluators where the instances of certain attributes are allocated to global variables and stacks. The most important findings of the comparison with regards to $\mathcal{G}1$ allocation techniques were:

— A more thorough analysis on the order in which attribute instances are computed is needed at evaluator construction time to allocate more attribute instances to global variables and stacks. In particular, it was suggested to compute information to determine whether, during a visit to a node, an

instance of a given attribute can be computed or used for the computation of dependent attribute instances.

— Combining allocations to eliminate copy operations and minimize the number of global variables and stacks needed is generally not so effective. The time savings of eliminated copy operations is hardly noticeable and, since the storage overhead for using a global variable or a stack is quite small, the space savings of minimizing their number is also quite small. The only situation identified in which combining allocations is really beneficial is the one where copy operations are eliminated that avoid pushing a value on a stack that is already on its top. This situation prevents stacks from growing too large and is therefore recognized as an effective storage optimization, all be it one of lower importance for an attribute storage allocation strategy. The reason for this ranking [12, page 415] is that it is more effective to optimize the number of attribute instances allocated to global variables and stacks.

One year after the publication of the findings of Farrow and Yellin, the first steps towards a more thorough storage allocation analysis at evaluator construction time were made by Kastens in [20]. In this paper, context-free grammars were constructed from the visit-sequences of a simple multi-visit evaluator to describe the sequences in which it would allocate and free the instances of a set of attributes. Conditions on the language of such grammars were then used to decide whether a simple multi-visit evaluator can allocate the instances of a set of attributes to a global variable or a stack. Both the construction of a grammar for a set of attributes as well as the check whether its language satisfies the conditions can be done in polynomial time. The main disadvantage of the grammars of Kastens is that their language may contain more sentences than that which is induced by the simple multi-visit evaluator so that he decide sufficient conditions only.

Engelfriet and De Jong [10] took in 1990 the direction suggested by Farrow and Yellin with respect to the computation of information for storage allocation analysis in simple multi-visit evaluators. They presented a polynomial time algorithm to determine, for any node, all possible distinct visits in which an instance of a specified attribute can be computed and used (for the computation of a dependent attribute instance). With this, and some additional information, necessary and sufficient conditions could be given to decide whether a simple multi-visit evaluator can allocate the instances of an attribute to a global variable or stack. Checking these conditions takes polynomial time.

The last work that must be mentioned in this literature survey on $\mathcal{G}1$ allocation techniques is that of Julié and Parigot [18]. Their paper, published in 1990, also considers space optimization in simple multi-visit evaluators. It is of interest because it presents context-free grammars which allow more efficient computation of the information that Engelfriet and De Jong used for storage allocation analysis. However, aside from the provision of a detailed complexity analysis of the time savings, this work needs to be revised. The use of the grammar of contexts and visits G_{CV} is superfluous: the behaviour of a simple multi-visit evaluator in a subtree does not depend on its context (see also [10, Lemma 2]).

One purpose of the work now described is to develop a $\mathcal{G}1$ allocation technique for use with simple multi-visit evaluators that improves those above in two aspects: firstly, it must decide global storage allocation for the instances of an applied occurrence of an attribute (in a particular production), and secondly, it must consider queues in addition to stacks and global variables as a possible data structure for global storage allocation.

The reason for studying global storage allocation for the instances of an

applied occurrence of an attribute can be explained using Figure 1 which shows a part of an (attributed) derivation tree. The leftmost instance of the synthesized attribute s of the nonterminal X is an instance of a *defined occurrence* of some production p_2 (that is, an instance of an attribute whose value is defined by a semantic rule associated with p_2) whereas it is an instance of an *applied occurrence* of production $p_1 : Y \to XX$. Although the

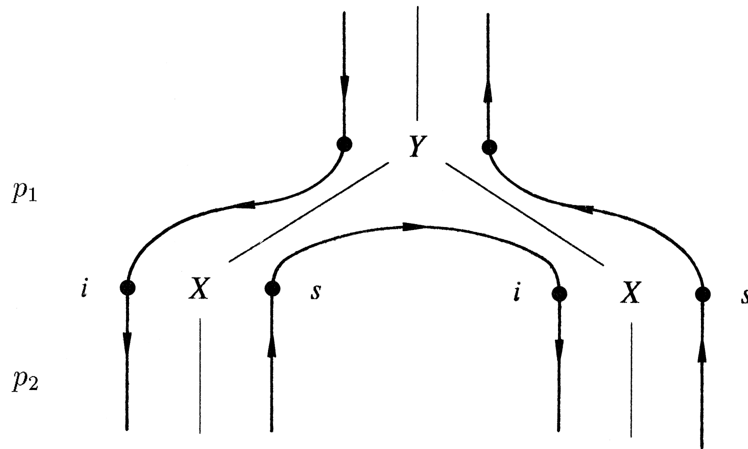

Figure 1. A part of a derivation tree.

instances of attribute s of X in this tree are instances of the same attribute, they are instances of distinct applied occurrences in p_1. In a given tree, the instances of a particular applied occurrence of an attribute form a subset of all instances of that attribute in that tree. Consequently, the instances of the applied occurrence can always be allocated to a data structure D if all the instances of the attribute can be allocated to D. Conversely, the fact that the instances of the attribute cannot be allocated to some data structure D does not necessarily imply that the instances of a particular applied occurrence

cannot be allocated to this data structure. Therefore, it is conceivable that less instances will have to be allocated to the nodes of the tree if storage allocation is used for the instances of an applied occurrence, and indeed, it will be shown that this can happen. The adaptation of a simple multi-visit evaluator to allocate the instances of an applied occurrence is straightforward and requires little extra space.

The decision to consider additionally queues for the global storage allocation of instances of applied occurrences raises two questions of interest:

(1) Are queues suitable as data structures for attribute evaluators?

(2) Are any other data structures appropriate for global storage allocation in attribute evaluators?

The answer to the first question will be shown to be affirmative. In fact, the results indicate that the use of a queue is as suitable as a stack in certain cases. The answer to the second question will also be shown to be affirmative. Firstly, by exhibition of another data structure suitable for storage allocation of an applied occurrence by a simple multi-sweep evaluator, and secondly, by theorems concerning this quest.

In this dissertation, a distinction will be drawn between single-use and multi-use applied occurrences. A single-use applied occurrence is an attribute occurrence on which exactly one defined occurrence depends. The term multi-use shall be used to stress the fact that an applied occurrence may not necessarily be a single-use applied occurrence.

Consideration is given initially to the simpler case of global storage allocation for single-use applied occurrences. First of all it will be shown that, for a given simple multi-sweep evaluator, it is decidable in polynomial time

whether it can allocate the instances of a single-use applied occurrence to a global variable, stack, or queue such that it is applicable for any derivation tree. Next, the data structures required for the global storage allocation of single-use applied occurrences in simple multi-X evaluators, where $X \in \{ \text{pass, sweep, visit} \}$, will be identified. In order to establish this result, the class of basic linear data structures is introduced. This class which is designed to be the most general class of data structures appropriate for the global storage allocation of single-use applied occurrences, can also be used to explore the theoretical possibilities and limitations of storage allocation in fields of research other than attribute grammars. Particular instances of this class of basic linear data structures are stacks and queues in which a stored value cannot be accessed more than once.

The results observed for the global storage allocation of single-use applied occurrences are carried over to multi-use applied occurrences using the idea of reallocation. After each use of an attribute instance of a multi-use applied occurrence, this instance can be reallocated (that is, removed from its current data structure and stored on another data structure). In this way, a conceptual distinction can be made between different single-use storage items; one for each data structure on which an instance of a multi-use applied occurrence is stored. Each single-use storage item, in turn, can be shown to be an instance of a single-use applied occurrence in a specifically constructed attribute grammar. This observation forms the crux of the generalization to multi-use applied occurrences. Using Engelfriet and De Jong's results [10], the decidability result is also generalized to simple multi-visit evaluators: it will be shown that independent of specific derivation trees it is decidable in polynomial time whether a given simple multi-visit evaluator can allocate the instances of a multi-use applied occurrence to a number of global variables,

stacks, and queues. To realize this generalization, some additional notations and notions are introduced.

The remainder of this dissertation is organized as follows. Chapter 2 presents definitions of the different types of evaluators which are considered. Chapter 3 discusses global storage allocation for single-use applied occurrences. This discussion leads to the introduction of basic linear data structures. Chapter 4 examines global storage allocation for multi-use applied occurrences, and generalizes all the results obtained in Chapter 3. Chapter 5 considers the implementation aspects of the allocation technique. An implementation is described and illustrated by means of an example. Finally, Chapter 6 summarizes the main results of the work and presents a number of conclusions. In addition, a number of problems are identified as a starting point for continued research.

CHAPTER TWO

Attribute Grammars and Evaluators

This chapter provides definitions of the basic concepts and notations that will be used throughout this dissertation. In addition, an example is presented to enable the reader to become familiar with the concepts and notations which are introduced. This example will be continued and expanded in the successive chapters.

The set of integers $\{ i \mid m \leq i \leq n \}$ is denoted by $[m, n]$. The empty string is denoted by the Greek symbol ϵ. The set of all strings, including ϵ, on some set V is denoted by V^*.

An attribute grammar as presented in [23] consists of an underlying context-free grammar augmented with attributes and semantic rules defining these attributes. The precise definition of attribute grammar can be stated in the following manner.

Definition 1. An *attribute grammar* G is a quadruple (G_u, A, V, R) consisting of:

(1) An underlying context-free grammar $G_u = (N, \Sigma, P, S)$ with nonterminals N, terminals Σ, production rules P, and start symbol S.

Production rule $p \in P$ has the form

$$p : X_0 \to w_0 X_1 w_1 \cdots w_{n-1} X_n w_n,$$

where n depends on p, $X_i \in N$ and $w_i \in \Sigma^*$ for every $i \in [0, n]$. Given that syntax is of secondary importance, p will always be abbreviated to $p : X_0 \to X_1 \cdots X_n$.

(2) For every nonterminal $X \in N$ a set of attributes $A(X) = I(X) \cup S(X)$, where $I(X)$ and $S(X)$ are disjoint sets of inherited and synthesized attributes, respectively.

(3) For every attribute α, a set of possible values denoted by $V(\alpha)$.

(4) For every production $p \in P$, a set of semantic rules $R(p)$.

To avoid confusion, attribute α will be called attribute occurrence (α, p, j) when $\alpha \in A(X_j)$ in production rule $p : X_0 \to X_1 \cdots X_n$ is meant. All the attribute occurrences of p are partitioned into two sets of defined and applied occurrences denoted by $DO(p)$ and $AO(p)$, respectively.

$$DO(p) = \{(s, p, 0) \mid s \in S(X_0)\} \cup \{(i, p, k) \mid i \in I(X_k),\ k \in [1, n]\}$$

$$AO(p) = \{(i, p, 0) \mid i \in I(X_0)\} \cup \{(s, p, k) \mid s \in S(X_k),\ k \in [1, n]\}$$

The semantic rules in $R(p)$ define all and only the attribute occurrences in $DO(p)$ as a function of certain attribute occurrences in $AO(p)$. Thus, a semantic rule defining $(\alpha, p, k) \in DO(p)$ has the form

$$(\alpha, p, k) = f((\alpha_1, p, k_1), \ldots, (\alpha_m, p, k_m)),$$

where f is a semantic function from $V(\alpha_1) \times \cdots \times V(\alpha_m)$ to $V(\alpha)$ and $(\alpha_i, p, k_i) \in AO(p)$ for all $i \in [1, m]$. It is said that (α, p, k) depends on $(\alpha_1, p, k_1), \ldots, (\alpha_1, p, k_1)$. □

Notation. Multiple occurrences of the same nonterminal, for example A, in a production are distinguished by subscripts: A_0 is written for the first occurrence of A, A_1 for the second occurrence of A, and so on. No confusion should arise using this notation: only when explicitly defined that X is a nonterminal of an attribute grammar G, each X_i in production $p : X_0 \to X_1 \cdots X_n \in P$ stands for an occurrence of X, and for an arbitrary nonterminal otherwise. □

Each attribute grammar G considered throughout the remainder of this work is, without loss of generality, assumed to be reduced in the sense that G_u is reduced and $I(S) = \varnothing$. The context-free grammar G_u on which attribute grammar G is based, is assumed to be reduced in the sense that for all $X \in V$ with $V = N \cup \Sigma$, there is a derivation $S \Rightarrow^* \alpha X \beta \Rightarrow^* \alpha\gamma\beta$, where $\gamma \in \Sigma^*$. (That is, G_u is reduced if all symbols in the vocabulary V are reachable from the start symbol S and produce at least one terminal string.)

For the sake of clarity, the dependencies between the attribute occurrences of a production p are often visualized by means of the dependency graph $DG(p)$ of p. This is a graph showing all the attribute occurrences of p as nodes and showing an arc from every $(\alpha, p, j) \in AO(p)$ to every $(\beta, p, k) \in DO(p)$ if and only if (β, p, k) depends on (α, p, j).

The notation and terminology used for a derivation tree t of an attribute grammar is defined by the following conventions in which n is assumed to be a node of t. If production $p : X_0 \to X_1 \cdots X_m$ is applied at node n, then n_i denotes the node with label X_i. (Thus n_i denotes the i-th nonterminal child

of node n if $i \in [1, m]$, and n itself if $i = 0$.) The subtree of n, denoted by $t_{(n)}$, is the tree consisting of node n and all its descendants in t. To avoid ambiguities, an attribute α of node n is referred to as attribute instance $\alpha(n)$, where n is labelled by X and $\alpha \in A(X)$. All the attribute instances of tree t depend on each other according to the dependencies of the applied productions. The root of a derivation tree is always assumed to be labelled by the start symbol S, and its leaves are assumed to be labelled by symbols from Σ.

An applied occurrence is said to be single-use if there is exactly one defined occurrence which depends on it. Similarly, an attribute instance is said to be single-use if there is exactly one instance which (directly) depends on it. To stress the fact that an applied occurrence or an attribute instance may not necessarily be single-use, the term multi-use will be used.

The evaluation of attribute instances in a derivation tree is carried out by a (tree-walking) attribute evaluator. This is a program which traverses the derivation tree in the following manner: starting at the root, it walks from node to node, evaluating at each node a number of its attribute instances until all attribute instances have been computed and the root is again reached.

A simple m-visit evaluator, $m > 0$, for an attribute grammar G is an attribute evaluator where the evaluation strategy is completely determined by a totally ordered partition $A_1(X), \ldots, A_{\phi(X)}(X)$ of $\phi(X) \in [0, m]$ parts over the attributes $A(X)$ of each $X \in N$. It traverses the derivation tree in such a way that, for each node n labelled by X and each $i \in [1, \phi(X)]$, all attribute instances of $A_i(X)$ at n are computed during the i-th visit to the subtree of n.

Definition 2. A *simple m-visit evaluator E for an attribute grammar G*

consists of components (1)-(4) where:

(1) For every $X \in N$, an integer $\phi(X) \in [0, m]$.

(2) For each $X \in N$, a partition $A_1(X), \ldots, A_{\phi(X)}(X)$ of $A(X)$.

(3) For every $p : X_0 \to X_1 \cdots X_n \in P$ and every $i \in [1, \phi(X_0)]$, a sequence

$$v_p(i) \in \{ \, visit_j(X_k) \mid j \in [1, \phi(X_k)], \, k \in [1, n] \, \}^*$$

such that sequence $visit_1(X_k) \cdots visit_{\phi(X_k)}(X_k)$ with $k \in [1, n]$ can be obtained from $v_p(1) \cdots v_p(\phi(X_0))$ by the deletion of all elements in $\{ \, visit_j(X_l) \mid j \in [1, \phi(X_l)], \, l \in [1, n], \, l \neq k \, \}$.

(4) For each $p \in P$, a set of evaluation rules $E(p)$ such that it has a rule

$$\textbf{set } (\alpha, p, k) \textbf{ to } f((\alpha_1, p, k_1), \ldots, (\alpha_m, p, k_m))$$

if and only if $(\alpha, p, k) = f((\alpha_1, p, k_1), \ldots, (\alpha_m, p, k_m)) \in R(p)$. □

The formal definition of a simple m-visit evaluator presented here is based on the definition given by Engelfriet and De Jong in [10]. With respect to the notation used here, the following should be noted.

(a) *Definition 2(1).* — As implied by the informal definition of a simple m-visit evaluator, integer $\phi(X)$ is assumed to be the total number of visits to any node x labelled by X.

(b) *Definition 2(2).* — No order is specified between the attributes in a given set $A_i(X)$ and, consequently, no order is forced on their computation. Nonetheless, a particular order is always assumed on their computation and, thus, on $A_i(X)$.

(c) *Definition 2(3).* — For every node x at which production p is applied, $visit_j(X_k)$ represents the j-th visit to node x_k so that $v_p(i)$ denotes the sequence of visits that must be made to the children of x during the i-th visit to x. Although X_k and X_l with $k \neq l$ possibly represent the same nonterminal, $visit_j(X_k)$ and $visit_j(X_l)$ are different visits of the evaluator by the notational convention to distinguish multiple occurrences of the same nonterminal in a production by subscripts.

(d) *Definition 2(4).* — Evaluation rules are distinct from semantic rules and specify how to compute the values of the attribute instances of the occurrences in $DO(p)$. The evaluation function f in an evaluation rule

$$\textbf{set } (\alpha, p, k) \textbf{ to } f((\alpha_1, p, k_1), \ldots, (\alpha_m, p, k_m))$$

is an implementation of the semantic function of the same name. This implementation evaluates the actual parameters $x_1, \ldots, x_k$ of function call $f(x_1, \ldots, x_k)$ in a sequential order. The attribute instances of applied occurrences $(\alpha_1, p, k_1), \ldots, (\alpha_m, p, k_m)$ necessary to compute an instance of (α, p, j) are thus used in sequential order.

A simple m-visit evaluator may visit any number of sons of a node n in any order, during one visit to n. A simple m-sweep evaluator is also a simple m-visit evaluator, but it operates a more restricted visit strategy. When this evaluator visits a node it must visit each of the sons of this node once, in a specific order which depends on the visit number and the production applied at the node.

Definition 3. A *simple m-sweep evaluator E of an attribute grammar G* is a simple m-visit evaluator of G such that:

(1) For all $X \in N$, $\phi(X) = m$.

(2) For every production $p : X_0 \to X_1 \cdots X_n \in P$ and every $i \in [1, \phi(X_0)]$, the sequence $v_p(i)$ has the form

$$visit_i(X_{p_1}) \cdot visit_i(X_{p_2}) \cdots visit_i(X_{p_n}),$$

where $\pi(p, i) = \langle p_1, \ldots, p_n \rangle$ is a permutation of $\langle 1, \ldots, n \rangle$. □

An even more restricted evaluator is obtained when the simple m-sweep evaluator is constrained to visit the sons of a node in a left-to-right (L) or a right-to-left (R) order depending on the sweep number. This evaluator is known as a simple m-pass evaluator.

Definition 4. A *simple m-pass evaluator E of an attribute grammar G* is a simple m-sweep evaluator of G such that for every $p : X_0 \to X_1 \cdots X_n \in P$ and every $i \in [1, \phi(X_0)]$, the sequence $v_p(i)$, has the form

$$visit_i(X_1) \cdot visit_i(X_2) \cdots visit_i(X_n),$$

if $d(i) = L$, and the form

$$visit_i(X_n) \cdot visit_i(X_{n-1}) \cdots visit_i(X_1),$$

if $d(i) = R$ where $d : [1, m] \to \{ L, R \}$ is a given mapping that specifies the direction of the passes. □

The term m-X is replaced by *multi*-X where $X \in \{$ pass, sweep, visit $\}$, if an evaluator is simple m-X for some $m > 0$. In addition to this convention, an attribute grammar will be called *simple multi*-X if there exists a simple multi-X evaluator for it. It will be clear from the previous definitions that if an attribute grammar is simple multi-pass, it is also simple multi-sweep, and

if an attribute grammar is simple multi-sweep, it is also simple multi-visit. The theory of these classes of evaluators and their corresponding classes of attribute grammars is developed in [2,4,8,9,19].

Remark. The pass direction in the simple multi-pass evaluator is the same for all trees during a particular pass, whereas in the simple multi-sweep evaluator the order depends on the production applied at the root node of the trees. An evaluator more restricted than the simple multi-sweep evaluator but more general than the simple multi-pass evaluator, is obtained if the pass direction depends on the pass number and on the production applied at the node (that is, $d(p,i)$ instead of $d(i)$). By definition the corresponding class of simple multi-waddle attribute grammars is properly included in the class of simple multi-sweep attribute grammars, and properly includes the class of simple multi-pass attribute grammars. Notice that each simple multi-sweep attribute grammar G with G_u in Chomsky normal form is also simple multi-waddle. It is shown in [9] that the problem of whether, for a fixed m, an attribute grammar is simple m-sweep is NP-complete whereas this problem is polynomial for simple m-pass. It is open whether this problem is still NP-complete for the class of simple m-waddle attribute grammars. □

The behaviour of the evaluators defined above can be expressed by a set of extended visit-sequences with one extended visit-sequence for each production. Let $I_{p,j}(X_i)$ be the set $\{ (\alpha, p, i) \mid \alpha \in A_j(X_i) \cap I(X_i) \}$ and $S_{p,j}(X_i)$ be the set $\{ (\alpha, p, i) \mid \alpha \in A_j(X_i) \cap S(X_i) \}$ for every production $p : X_0 \to X_1 \cdots X_n \in P$, every $i \in [0, n]$, and every $j \in [1, \phi(X_i)]$.

Definition 5. The *extended visit-sequence of production* $p : X_0 \to X_1 \cdots X_n$ is the sequence $V_p = V_p(1) \cdots V_p(\phi(X_0))$ where, for every $i \in [1, \phi(X_0)]$,

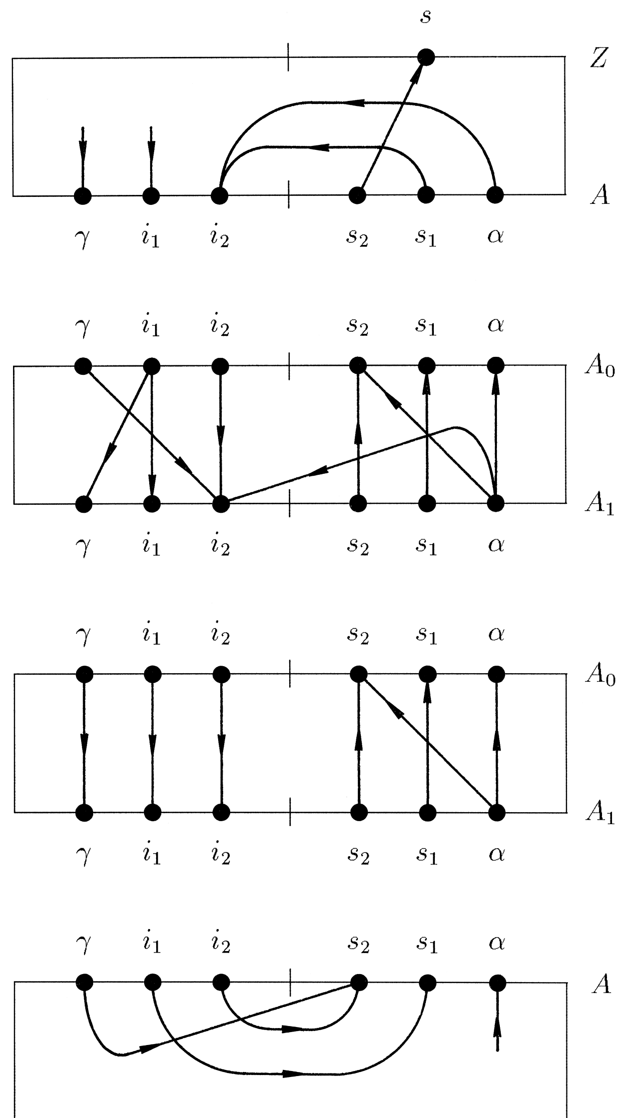

Figure 2. Dependency graphs $DG(p_1)$, $DG(p_2)$, $DG(p_3)$, and $DG(p_4)$.

sequence $V_p(i)$ equals

$$\langle I_{p,i}(X_0), I_{p,m_1}(X_{j_1}), visit_{m_1}(X_{j_1}), S_{p,m_1}(X_{j_1}), \ldots$$
$$\ldots, I_{p,m_r}(X_{j_r}), visit_{m_r}(X_{j_r}), S_{p,m_r}(X_{j_r}), S_{p,i}(X_0)\rangle,$$

if $v_p(i) = visit_{m_1}(X_{j_1}) visit_{m_2}(X_{j_2}) \cdots visit_{m_r}(X_{j_r})$. □

The following example illustrates this concept.

Example 1. Let G be a simple multi-visit attribute grammar with start symbol Z, nonterminals $N = \{ Z, A \}$, terminals $\Sigma = \{ a, b \}$, and productions

$$P = \{ p_1 : Z \to A,\ p_2 : A_0 \to aA_1,\ p_3 : A_0 \to bA_1,\ p_4 : A \to b \}.$$

The sets of attributes of each nonterminal are:

$$I(Z) = \varnothing \qquad I(A) = \{ i_1, i_2, \gamma \}$$
$$S(Z) = \{ s \} \qquad S(A) = \{ s_1, s_2, \alpha \}$$

Figure 2 shows the relevant aspects of sets $R(p_1)$, $R(p_2)$, $R(p_3)$, and $R(p_4)$ by means of their dependency graphs.

Attribute grammar G is evaluated by evaluator E whose behaviour is given by means of the extended visit sequences of Table 1. It should be clear from these sequences that E is a simple multi-pass evaluator. In the first pass all the instances of attributes i_1, s_1, γ, and α are evaluated, and in the second pass all the instances of i_2, s_2, and s are evaluated. The direction of the passes $d(1)$ and $d(2)$ are irrelevant in this case due to the linearity of the underlying context-free grammar G_u. □

When working with extended visit-sequences, the following notations will be used. The (unique) set in V_p that contains attribute occurrence (α, p, j)

Production Rule	Extended Visit-Sequence
p_1	$\langle \varnothing, \{(i_1, p_1, 1), (\gamma, p_1, 1)\}, visit_1(A), \{(s_1, p_1, 1), (\alpha, p_1, 1)\}, \varnothing \rangle$ $\langle \varnothing, \{(i_2, p_1, 1)\}, visit_2(A), \{(s_2, p_1, 1)\}, \{(s, p_1, 0)\}\rangle$
p_2	$\langle \{(i_1, p_2, 0), (\gamma, p_2, 0)\}, \{(i_1, p_2, 1), (\gamma, p_2, 1)\}, visit_1(A_1), \{(s_1, p_2, 1), (\alpha, p_2, 1)\}, \{(s_1, p_2, 0), (\alpha, p_2, 0)\}\rangle$ $\langle \{(i_2, p_2, 0)\}, \{(i_2, p_2, 1)\}, visit_2(A_1), \{(s_2, p_2, 1)\}, \{(s_2, p_2, 0\}\rangle$
p_3	$\langle \{(i_1, p_3, 0), (\gamma, p_3, 0)\}, \{(i_1, p_3, 1), (\gamma, p_3, 1)\}, visit_1(A_1), \{(s_1, p_3, 1), (\alpha, p_3, 1)\}, \{ (s_1, p_3, 0), (\alpha, p_3, 0)\}\rangle$ $\langle \{(i_2, p_3, 0)\}, \{(i_2, p_3, 1)\}, visit_2(A_1), \{(s_2, p_3, 1)\}, \{(s_2, p_3, 0)\}\rangle$
p_4	$\langle \{(i_1, p_4, 0), (\gamma, p_4, 0)\}, \{(s_1, p_4, 0), (\alpha, p_4, 0)\}\rangle$ $\langle \{(i_2, p_4, 0)\}, \{(s_2, p_4, 0)\}\rangle$

Table 1. Extended visit-sequences of simple multi-pass evaluator E.

is denoted by $set(\alpha, p, j)$. For elements a and b in V_p, the expression "$a < b$ in V_p" will be written if a occurs before b in V_p.

The evaluation performed by a simple multi-visit evaluator E given some derivation tree t can be partly described by means of an evaluation sequence. This sequence, denoted by $\rho(t)$, is a string in which all attribute instances of tree t occur exactly once, such that attribute instance α appears before instance β in $\rho(t)$ if and only if α is computed before β. Thus, $\rho(t)$ describes the order in which the attribute instances of t are computed, and in a partial manner, the way in which they are used: the order in which an evaluation function $f((\alpha_1, p, k_1), \ldots, (\alpha_m, p, k_m))$ of an evaluation rule of E uses the values of the instances of $(\alpha_1, p, k_1), \ldots, (\alpha_m, p, k_m)$ is missing in $\rho(t)$. Note that $\rho(t)$ can easily be obtained from the extended visit-sequences of the productions occurring in t using the implicitly assumed ordering of the sets $A_j(X_i)$.

CHAPTER THREE

Global Storage Allocation of Single-use Applied Occurrences

Typically, the values of computed attribute instances have to be stored during evaluation because the values of other attribute instances depend on them or because they have to be delivered as output. For this purpose, the evaluator needs a storage allocation strategy. In a naive allocation strategy, the values of computed attribute instances reside at their own nodes during the entire evaluation process. Unfortunately, such a simple strategy is too space consuming for practical use.

In this dissertation a storage allocation strategy for the evaluators of the previous chapter will be developed with the objectives of:

(1) reusing storage space occupied by attribute values which are no longer required, and

(2) leaving the evaluation strategy of the evaluator unchanged and its structure independent of specific derivation trees.

The technique investigated in this dissertation to accomplish these objectives is the allocation of particular attribute instances to some global data structure such that it is applicable to any derivation tree. Generally, global variables and stacks are considered as candidates for such a global storage allocation scheme (see for example [10,15,18,20]). However these are not the only conceivable global data structures appropriate for global storage allocation. In particular, the use of global queues for global storage allocation will also be considered.

This chapter presents a number of practical and theoretical results concerning the global storage allocation of all the attribute instances of a single-use applied occurrence. The practical results focus on the necessary and sufficient conditions to decide at evaluator construction time whether a simple multi-sweep evaluator can allocate the instances of a single-use applied occurrence to a global variable, stack, or queue.

The theoretical results extend the scope of the investigation by considering which data structures are required for the global storage allocation of the instances of single-use applied occurrences in simple multi-X evaluators, where $X \in \{\text{pass, sweep, visit}\}$. Let $\mathcal{D}$ be the set of data structures appropriate for the global storage allocation of the instances of single-use applied occurrences. This chapter sets out to establish the following theoretical results with respect to $\mathcal{D}$.

(1) *Simple multi-pass evaluators.* — For these evaluators, it will be shown that it is sufficient to consider stacks and queues for the global storage allocation of the instances of single-use applied occurrences.

(2) *Simple multi-sweep evaluators.* — With respect to these evaluators, it will be shown that it is sufficient to consider a proper subset of $\mathcal{D}$

for the global storage allocation of the instances of single-use applied occurrences. This subset, however, is not finite.

(3) *Simple multi-visit evaluators.* — For this class of evaluators, it will be shown that it is no longer sufficient to consider only a subset of $\mathcal{D}$ for the global storage allocation of the instances of single-use applied occurrences. The entire set $\mathcal{D}$ is needed.

In order to prove results (1)-(3) above, it is necessary to provide a definition of the very general class $\mathcal{D}$. Stacks and queues, in which a stored value cannot be accessed more than once, are particular instances of data structures belonging to this class.

All results presented in this chapter are carried over to multi-use applied occurrences in Chapter 4.

To provide a formal basis for this research, it is necessary to define precisely what it means to state that a group of attribute instances of a derivation tree, given some simple multi-visit evaluator performing the evaluation of this tree, can be allocated to a global variable, stack or queue. These definitions are not restricted to a group of single-use attribute instances in order to make them more general.

First the notion of the existence of an attribute instance in a tree t is introduced. Let E be a simple multi-visit evaluator computing the attribute instances of derivation tree t. (Note that at each moment in time at which E is executing an evaluation rule, it is either assigning a value to an instance of a defined occurrence or using the value of an instance of an applied occurrence for the computation of the evaluation function.) The existence $\xi(\alpha)$ of an attribute instance α in t is the sequence $\langle a_1, \ldots, a_n \rangle$ of natural numbers

derived from the evaluation performed by E, where $n \geq 1$ depends on α and $a_1 < a_2 < \cdots < a_n$ such that α is computed (*i.e.* assigned a value) at time a_1 and applied (*i.e.* used) at times $a_2, \ldots, a_n$. Thus, if α and β are distinct attribute instances in t with $\xi(\alpha) = \langle a_1, \ldots, a_n \rangle$ and $\xi(\beta) = \langle b_1, \ldots, b_m \rangle$ then $a_i \neq b_j$ (because of the sequential nature of E).

Readers familiar with references [15,20] or related work should not confuse the notion of existence with the notion of the lifetime of an attribute instance α which is defined as the pair (a_1, a_n) assuming that $\xi(\alpha) = \langle a_1, \ldots, a_n \rangle$ with $n \geq 2$. The concept of lifetime contains less information and is defined only for instances which are applied at least once.

It goes without saying that it is useless to store attribute instances which are never applied; that is, to store instances which do not have to be delivered as output and on which no other attribute instances depend. Therefore, it is assumed that $n \geq 2$ for every attribute instance under consideration with existence $\langle a_1, \ldots, a_n \rangle$.

Let A denote a group of attribute instances occurring in some derivation tree which is evaluated by a simple multi-visit evaluator.

Definition 6. A group A of attribute instances *can be allocated to a global variable* if for every $\alpha \in A$ with $\xi(\alpha) = \langle a_1, \ldots, a_n \rangle$ the following holds: there is no $\beta \in A$, with $\xi(\beta) = \langle b_1, \ldots, b_m \rangle$, such that $a_1 < b_1 < a_n$. □

A stack is a linear list in which insertion, access and deletion all take place at one end of the list.

Definition 7. A group A of attribute instances *can be allocated to a global stack* if for every $\alpha \in A$ with $\xi(\alpha) = \langle a_1, \ldots, a_n \rangle$ the following holds: there is no $\beta \in A$, with $\xi(\beta) = \langle b_1, \ldots, b_m \rangle$, such that $a_1 < b_1 < a_i < b_m$ for some $i \in [2, n]$. □

A queue is a linear list in which insertion takes place at one end and access and deletion at the other end of the list.

Definition 8. A group A of attribute instances *can be allocated to a global queue* if for every $\alpha \in A$ with $\xi(\alpha) = \langle a_1, \ldots, a_n \rangle$ the following holds: there is no $\beta \in A$, with $\xi(\beta) = \langle b_1, \ldots, b_m \rangle$, such that $a_1 < b_1$ and $b_2 < a_n$. □

The relation between stacks, queues, and global variables is expressed in the following lemma.

Lemma 1. *A group of attribute instances A can be allocated to a global stack as well as to a global queue if and only if A can be allocated to a global variable.*

Proof. Immediate from the definitions above and the assumption that all instances in A are used at least once. □

So far, the allocation of an arbitrary group of attribute instances of a particular derivation tree has been considered. However, to meet the objective of leaving the structure of the evaluator independent of specific derivation trees, the groups of attribute instances which are considered must be restricted to those that are statically characterizable. In the literature, global storage allocation is considered for the instances of a particular attribute. Here, however, global storage allocation for the instances of a particular applied occurrence is considered.

Why consider instances of applied occurrences? Before answering this question, note that it is not interesting to consider applied occurrences that are not used, because it is useless to store the instances of these applied occurrences. Therefore, each applied occurrence under consideration is assumed

to be used. Now returning to the question, the main reason for considering global storage allocation for the instances of an applied occurrence is that it is not only possible but also sensible. How this can be implemented so that no additional effort is needed in order to handle applied occurrences that are not used will be explained in Chapter 5. The reader is invited to consider Example 2 which illustrates the rationale behind this approach.

Why not consider attribute instances of a defined occurrence? To explain this, assume that an evaluator is visiting a node n of a tree t and is bound to visit node m after the computation of instance $\alpha(m)$ at m. (Note that the value of an instance of a node is always computed before the node is actually visited.) Then, since the production applied at node n is known, the evaluator knows of which defined occurrence $\alpha(m)$ is an instance. Thus it is possible to allocate the value of $\alpha(m)$ to the appropriate data structure for the defined occurrence of which $\alpha(m)$ is an instance. Now, assume that the evaluator actually visits node m and that it needs the value of $\alpha(m)$ to compute some attribute instance. Since only the production applied at node m is known, the evaluator no longer knows of which defined occurrence $\alpha(m)$ is an instance. Consequently, it does not know where the value of $\alpha(m)$ is allocated. Hence, the global storage allocation for the instances of a defined occurrence needs more information about the tree than is contained in its nodes and in the structure of the original evaluator.

Next, the necessary and sufficient conditions are presented to decide for a simple multi-sweep evaluator whether global storage allocation to a stack is possible for the attribute instances of a single-use applied occurrence.

Lemma 2. *Let E be a simple multi-sweep evaluator for an attribute grammar G and let $p : X_0 \rightarrow X_1 \cdots X_n \in P$ with $(\alpha, p, j) \in AO(p)$ single-use and*

(β, p, i) the (unique) occurrence in $DO(p)$ depending on (α, p, j). Assume $\alpha \in A_a(X_j)$ and $\beta \in A_b(X_i)$. The attribute instances of (α, p, j) cannot be allocated to a global stack during evaluation by E if and only if at least one of the following conditions is satisfied.

(1) *There is an integer $k \in [1, n]$ such that, for some γ and δ in V^*, $X_k \Rightarrow^* \gamma X_0 \delta$ and*

$$set(\alpha, p, j) < visit_a(X_k) < set(\beta, p, i) < visit_b(X_k) \text{ in } V_p.$$

(2) *There is an integer $k \in [1, n]$ such that, for some γ and δ in V^*, $X_k \Rightarrow^* \gamma X_0 \delta$ and*

$$visit_a(X_k) < set(\alpha, p, j) < visit_b(X_k) < set(\beta, p, i) \text{ in } V_p.$$

(3) *There exist a production $q : Y_0 \to Y_1 \cdots Y_m$ and distinct $k, l \in [1, m]$ such that, for some γ, δ, η and ω in V^*, $Y_k \Rightarrow^* \gamma X_0 \delta$, $Y_l \Rightarrow^* \eta X_0 \omega$, and*

$$visit_a(Y_k) < visit_a(Y_l) < visit_b(Y_k) < visit_b(Y_l) \text{ in } V_q.$$

Proof. Let t be a derivation tree with x and y denoting two of its nodes at which production p is applied. The instance of (α, p, j) at node x_j is denoted by γ and the instance of (α, p, j) at node y_j is denoted by δ, where $\xi(\gamma) = \langle g_0, g_1 \rangle$ and $\xi(\delta) = \langle d_0, d_1 \rangle$.

Firstly it will be shown that the instances of (α, p, j) cannot be allocated to a global stack during evaluation by E if one of the conditions (1)-(3) is satisfied. Each condition is treated separately (recall that G_u is reduced).

(a) *Condition (1) is satisfied.* — Then G has a derivation tree such as t (where node y is a descendant of node x_k or is node x_k itself) which is

evaluated by E is such a way that $g_0 < d_0 < g_1 < d_1$. This inequality directly shows that, by Definition 7, the instances of (α, p, j) cannot be allocated to a global stack during evaluation by E.

(b) *Condition (2) is satisfied.* — Then G has a tree such as t (where y is a descendant of x_k or is x_k itself) which is evaluated by E in such a way that $d_0 < g_0 < d_1 < g_1$ holds. This inequality again implies that the instances of (α, p, j) cannot be allocated to a global stack during evaluation by E.

(c) *Condition (3) is satisfied.* — Let z be a node of t at which production q is applied. Then G has a tree such as t (where x a descendant of node z_k or is z_k itself, and y a descendant of node z_l or is z_l itself) which is evaluated by E in such a way that $g_0 < d_0 < g_1 < d_1$ holds. Hence the instances of (α, p, j) cannot be allocated to a global stack during evaluation by E.

Secondly, it will be shown that at least one of the conditions (1)-(3) is satisfied if the instances of (α, p, j) cannot be allocated to a global stack during evaluation by E. Therefore, assume that the instances of (α, p, j) cannot be allocated to a global stack during evaluation by E. Then G must have a derivation tree such as t which is evaluated by E in such a way that inequality $g_0 < d_0 < g_1 < d_1$ holds (compare Definition 7). Three cases can be identified.

(a) *Node y is a descendant of node x_k or is x_k itself for some $k \in [1, n]$.* — It is clear that $set(\alpha, p, j) < visit_a(X_k) < set(\beta, p, i) < visit_b(X_k)$ in V_p. Hence condition (1) is satisfied.

(b) *Node x is a descendant of y_k or is y_k itself for some $k \in [1,n]$.*— It is clear that $visit_a(X_k) < set(\alpha,p,j) < visit_b(X_k) < set(\beta,p,i)$ in V_p, so that condition (2) holds.

(c) *Nodes x and y are incomparable.*— Let z be the root of the smallest tree of t containing both x and y, and let $q : Y_0 \to Y_1 Y_2 \cdots Y_m$ be the production applied at z. Assume x is a descendant of z_k or z_k itself and y a descendant of z_l or z_l itself, for some $k, l \in [1,m]$. (Note that $k \neq l$.) Then $visit_a(Y_k) < visit_a(Y_l) < visit_b(Y_k) < visit_b(Y_l)$ in V_q. This fact implies that condition (3) is satisfied. □

Lemma 3 states the necessary and sufficient conditions to decide for a simple multi-sweep evaluator whether global storage allocation to a queue is possible for the instances of a single-use applied occurrence. The proof is similar to that of Lemma 2.

Lemma 3. *Let E be a simple multi-sweep evaluator for an attribute grammar G and let $p : X_0 \to X_1 \cdots X_n \in P$ with $(\alpha,p,j) \in AO(p)$ single-use and (β,p,i) the (unique) occurrence in $DO(p)$ depending on (α,p,j). Assume $\alpha \in A_a(X_j)$ and $\beta \in A_b(X_i)$. The attribute instances of (α,p,j) cannot be allocated to a global queue during evaluation by E if and only if at least one of the following conditions is satisfied.*

(1) There is an integer $k \in [1,n]$ such that, for some γ and δ in V^*, $X_k \Rightarrow^* \gamma X_0 \delta$ and

$$set(\alpha,p,j) < visit_a(X_k), \text{ and } visit_b(X_k) < set(\beta,p,i) \text{ in } V_p.$$

(2) There is an integer $k \in [1,n]$ such that, for some γ and δ in V^*, $X_k \Rightarrow^* \gamma X_0 \delta$ and

$$visit_a(X_k) < set(\alpha, p, j), \text{ and } set(\beta, p, j) < visit_b(X_k) \text{ in } V_p.$$

(3) There exist a production $q : Y_0 \to Y_1 \cdots Y_m$ *and distinct* $k, l \in [1, m]$ *such that, for some* γ, δ, η *and* ω *in* V^*, $Y_k \Rightarrow^* \gamma X_0 \delta$, $Y_l \Rightarrow^* \eta X_0 \omega$, *and*

$$visit_a(Y_k) < visit_a(Y_l) \text{ and } visit_b(Y_l) < visit_b(Y_k) \text{ in } V_q. \qquad \square$$

The following theorem arises immediately from these lemmas.

Theorem 1. *Let* E *be a simple multi-sweep evaluator for an attribute grammar* G *and let* (α, p, j) *be a single-use applied occurrence of* G. *It is decidable in polynomial time whether the instances of* (α, p, j) *can be allocated to a global variable, stack or queue during evaluation by* E.

Proof. The theorem follows from Lemmas 1-3 using the fact that the conditions in Lemmas 2 and 3 can be checked in polynomial time. □

Applied Occurrence	Allocation
$(\gamma, p_2, 0)$	global queue
$(\gamma, p_3, 0)$	global variable
$(\gamma, p_4, 0)$	global variable

Table 2. Allocation of applied occurrences $(\gamma, p_2, 0)$, $(\gamma, p_3, 0)$, and $(\gamma, p_4, 0)$.

Example 2. This example continues Example 1. By means of Lemmas 1-3 the allocations of Table 2 can be found for the attribute instances of $(\gamma, p_2, 0)$, $(\gamma, p_3, 0)$, and $(\gamma, p_4, 0)$ during evaluation by E. □

Remark. In the literature, attributes are sometimes classified into temporary and nontemporary attributes. An attribute α is called temporary if every applied occurrence of α is temporary. An applied occurrence (α, p, j) is temporary if each defined occurrence (β, p, i) that depends on (α, p, j) occurs in the same sequence of V_p as (α, p, j). Lemma 2 shows that every temporary single-use applied occurrence can be allocated to a global stack. Thus, if such an occurrence can be allocated to a global queue, then it can be allocated to a global variable. On the other hand, for nontemporary single-use applied occurrences, Lemmas 2 and 3 show that queues can be handled as easily as stacks. □

To show which other data structures appropriate for the global storage allocation of the instances of single-use applied occurrences are needed in simple multi-X evaluators, with $X \in \{\text{pass, sweep, visit}\}$, the data structures that are relevant for this purpose must be characterized. In order to do this, it is useful to make the following observations.

(1) Because of the sequential nature of simple multi-visit evaluation, it is sufficient to consider only data structures in which the attribute values are stored chronologically.

(2) Because of the atomic nature of a single-use applied occurrence at the descriptional level, only two operations can be performed for all its instances, namely an input operation i to store an attribute value and an output operation o to fetch (*i.e.* access and delete) a stored value.

(3) Because of the objective to leave the structure of the evaluator independent of specific derivation trees, the global data structures used by the evaluator must also be independent of specific derivation trees.

From (1) and (2) it is inferred that only linear data structures that provide the basic operations i and o need to be considered. From (3) it is inferred that these basic operations can only depend on the order in which instances are stored and not on the stored values themselves. Particular examples of such data structures are stacks and queues where the possibility of accessing a stored value more than once is excluded.

The formal characterization of these data structures, called basic linear data structures, begins with their definition. Thereafter, an explanation of the definition will be given. The symbol $\cdot$ is used to denote concatenation of sequences, $\#_x(\alpha)$ to denote the number of occurrences of symbol x in sequence α, and $|\alpha|$ to denote the length of sequence α.

Definition 9. A *basic linear data structure* D over a set *Data* is a structure (Σ, c, p, i, o), where

(1) Σ is a set whose elements $\langle \delta, M \rangle$ represent the possible *content* of the data structure D, consisting of a *state* $\delta \in \{ i, o \}^*$ and a set of *time-stamped data* $M \subseteq \mathit{Data} \times \mathbb{N}^+$.

(2) c denotes the *initial content* of data structure D defined as $\langle \epsilon, \varnothing \rangle$.

(3) $p : \{ i, o \}^* \to \mathbb{N}^+$, is a *protocol*, satisfying,

$$p(w_1 \cdots w_n) = j$$

for some $j \in [1, \#_i(w_1 \cdots w_n)]$, such that for every $k \in [1, n]$, with $w_k = o$, $p(w_1 \cdots w_{k-1}) \neq j$ (if such a j exists).

(4) $i : \Sigma \times \mathit{Data} \to \Sigma$, is the *input operation*,

$$i(\langle \delta, M \rangle, \partial) = \langle \delta \cdot i, M \cup \{ (\partial, \#_i(\delta) + 1) \} \rangle.$$

(5) $o : \Sigma \to \Sigma \times Data$, is the *output operation*,

$$o(\langle \delta,\, M \rangle) = (\langle \delta \cdot o,\, M \setminus \{\, (\partial,\, p(\delta))\, \} \rangle,\, \partial). \qquad \square$$

Remark. There should be no confusion over whether i denotes the symbol i or the input operation i, and over whether o denotes the symbol o or the output operation o; state $\delta \in \{\, i,\, o \,\}^*$ wherein i and o denote the symbols i and o, respectively. $\square$

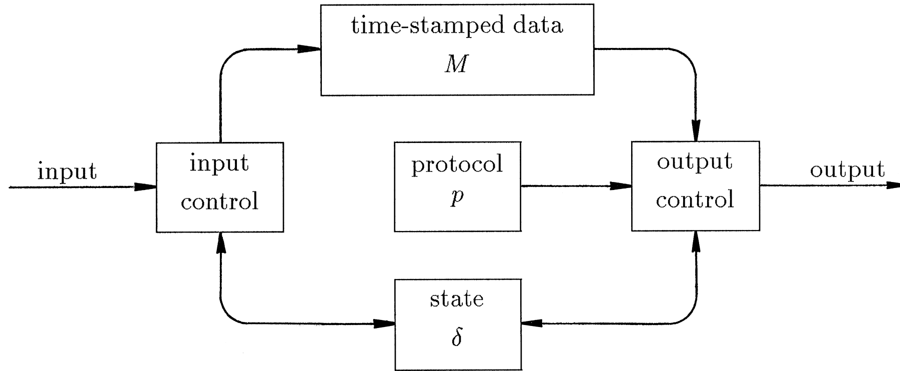

Figure 3. A model of a basic linear data structure.

How does a basic linear data structure work? A model of such a data structure is shown in Figure 3. The input control represents the program invoked by a call of input operation i with current content $\langle \delta,\, M \rangle \in \Sigma$ and some item $\partial \in Data$. It adds item ∂ with time stamp $\#_i(\delta) + 1$ to M and sets the state to $\delta \cdot i$. Initially, the content is c (*i.e.* $\langle \epsilon, \varnothing \rangle$). The output control represents the program invoked by a call of output operation o with current content $\langle \delta,\, M \rangle \in \Sigma$. If M is not empty, this program deletes the (by its time-stamp) unique element $(\partial,\, p(\delta))$ in M, sets the state to $\delta \cdot o$, and

delivers item ∂. If M is empty, this program aborts due to the simple fact that $p(\delta)$ is undefined in this case.

The input and output control are the same for all basic linear data structures; only the protocol changes from one basic linear data structure to another. So, for a given set *Data*, a basic linear data structure is completely determined by specifying the protocol p. The class of all these basic linear data structures will be denoted by $\mathcal{D}$. (Notice that not each structure in $\mathcal{D}$ can efficiently be implemented; it even may have a noncomputable protocol.)

Some additional concepts are needed to formalize what is meant by allocating a group of single-use instances A of some derivation tree t to a data structure $D \in \mathcal{D}$ during evaluation of a simple multi-visit evaluator E. (Note that the possible values of the instances in A form the set *Data* over which D is a data structure). Let $\alpha_1 \cdots \alpha_n$ be the evaluation sequence $\rho(t)$ induced by E and $\alpha_{i_1}, \ldots, \alpha_{i_{|A|}}$ all the instances of A in the order in which they occur in $\rho(t)$. The define-and-use sequence of A is the sequence ψ which is defined by the following construction.

set ψ to $\rho(t)$;
for $j = 1, \ldots, n$
do if there is a $k \in [1, |A|]$ such that $\alpha_j = \alpha_{i_k}$
  then replace α_j in ψ by @k
  else replace α_j in ψ by @
  fi;
  if α_j depends on instances in A
  then let $\alpha_{i_{j_1}}, \ldots, \alpha_{i_{j_m}}$ be all the instances in A which
      are successively used to compute α_j

```
            in replace @ in ψ by j_1 ··· j_m
            ni
      else  remove @ from ψ
      fi
   od;
```

Thus, the define-and-use sequence ψ of A is a sequence in which each number $i \in [1, |A|]$ occurs twice such that this sequence encodes the order in which the single-use instances of A are computed and used. The action sequence of A is obtained from ψ by replacing all first occurrences of numbers in ψ by i and replacing all second occurrences of numbers in ψ by o. The output sequence of A is obtained from ψ by deleting all first occurrences of numbers in ψ. Finally, $DU_E(\alpha, p, j)$ is the set containing the define-and-use sequences of the instances of a single-use applied occurrence (α, p, j) for any derivation tree evaluated by a simple multi-visit evaluator E.

Example 3. Consider Example 1 once again. It is easily seen that

$$DU_E(\gamma, p_2, 0) = \{\, \epsilon, 1 \cdot 1,\ 1 \cdot 2 \cdot 1 \cdot 2,\ 1 \cdot 2 \cdot 3 \cdot 1 \cdot 2 \cdot 3,\ \dots \},$$

$$DU_E(\gamma, p_3, 0) = \{\, \epsilon, 1 \cdot 1,\ 1 \cdot 1 \cdot 2 \cdot 2,\ 1 \cdot 1 \cdot 2 \cdot 2 \cdot 3 \cdot 3,\ \dots \}.$$

For each define-and-use sequence $\psi \in DU_E(\gamma, p_2, 0)$, an action sequence $i^n o^n$ and an output sequence $1 \cdot 2 \cdots n$ can be found where $|\psi| = 2n$. Similarly, for each $\psi \in DU_E(\gamma, p_3, 0)$, an action sequence $(io)^n$ and an output sequence $1 \cdot 2 \cdots n$ can be found where $|\psi| = 2n$. □

At this point, all the concepts have been introduced to enable presentation of the following definition. Let A be a group of single-use instances in some derivation tree which is evaluated by a simple multi-visit evaluator.

Definition 10. A group A of single-use attribute instances with action sequence $x_1, x_2, \ldots, x_{2|A|}$ and output sequence $y_1, \ldots, y_{|A|}$ *can be allocated to a basic linear data structure* D if its protocol satisfies

$$p(x_1 x_2 \cdots x_{k-1}) = y_{\#_o(x_1 x_2 \cdots x_k)}$$

for all $k \in [1, 2|A|]$ with $x_k = o$. $\square$

To illustrate the generality of the class of basic linear data structures $\mathcal{D}$, observe that the requirement of Definition 10 to allocate A does not conflict with the demand on protocols as stated in Definition 9(3). Hence, there is always some D in $\mathcal{D}$ (with *Data* the set of possible values of the instances in A) to which A can be allocated.

The behaviour of a basic linear data structure D is a useful concept which directly relates the sequence of actions performed on D with the order in which the stored values are fetched by D.

Definition 11. The *behaviour of a basic linear data structure* D, with protocol p, is the partial function $f : \{ i, o \}^* \to (\mathbb{N}^+)^*$ satisfying

$$f(x_1 \cdots x_n) = p(x_1 \cdots x_{i_1}) \cdots p(x_1 \cdots x_{i_m}),$$

for all possible action sequences $x_1 \cdots x_n$ such that $i_1 < i_2 < \cdots < i_m$ and, for each $j \in [1, m]$, $i_j \in [0, n-1]$ and $x_{i_j+1} = o$ where $m = \#_o(x_1 \cdots x_n)$. $\square$

The reader can check that any data structure D in $\mathcal{D}$, with behaviour f, has the following properties.

(1) D is deterministic (because f is a function).

(2) If an item $\partial \in Data$ has been stored in D, then it is possible to obtain it by a series of invocations of output operation o. That is, for all action

sequences σ such that $f(\sigma)$ is defined and for each $j \in [1, \#_i(\sigma)]$, there is a continuation sequence $\tau \in \{o\}^*$ such that j occurs in $f(\sigma \cdot \tau)$.

(3) The result of a sequence of actions applied on D is not influenced by how this sequence is continued. For all σ and τ, there is a θ, possibly ϵ, such that

$$f(\sigma \cdot \tau) = f(\sigma) \cdot \theta.$$

Notice that the equality $f(\sigma \cdot \tau) = f(\sigma) \cdot f(\tau)$ does not hold in general. The result of an action sequence may depend on its history.

As an immediate consequence of the previous definitions, the following lemma can be stated.

Lemma 4. *Let D be a basic linear data structure with behaviour f, and E a simple multi-visit evaluator for an attribute grammar G with single-use applied occurrence (α, p, j). The attribute instances of (α, p, j) can be allocated to D during evaluation by E if and only if*

$$f(\sigma) = \theta$$

for each $\phi \in DU_E(\alpha, p, j)$, where σ denotes the action sequence and θ the output sequence obtained from ϕ. □

Given previous conclusion based on the earlier discussion of the generality of the class of basic linear data structures $\mathcal{D}$, it follows from Lemma 4 that if a simple multi-visit evaluator E cannot allocate the instances of a single-use applied occurrence (α, p, j) to any basic linear data structure, then this is only caused by some incompatibility in at least two define-and-use sequences

of $DU_E(\alpha, p, j)$. This situation will be considered in detail in the proof of Theorem 2.

Now stacks and queues are defined as elements of the class $\mathcal{D}$.

Definition 12. Let D be a data structure in $\mathcal{D}$.

(1) D is a *basic stack* if it has a protocol p satisfying:

$$p(w_1 \cdots w_n) = \begin{cases} \#_i(w_1 \cdots w_n) & \text{if } w_n = i \\ p(reduce(w_1 \cdots w_n)) & \text{if } w_n \neq i \end{cases}$$

where $reduce(w)$ is the sequence obtained from w by deleting the last occurrence of a sequence of the form io in w.

(2) D is a *basic queue* if it has a protocol p satisfying:

$$p(w) = \#_o(w) + 1. \qquad \square$$

There are a lot of protocols besides the basic stack and the basic queue protocols. The alternating protocol, for instance, applies the queue protocol and the stack protocol successively, starting with application of the queue protocol on the first output demand. If $D \in \mathcal{D}$ with behaviour f has the alternating protocol, then, for instance,

$$f(iiiiiooooo) = 1 \cdot 5 \cdot 2 \cdot 4 \cdot 3$$

and

$$f(iiioooioioiioo) = 1 \cdot 3 \cdot 2 \cdot 4 \cdot 5 \cdot 7 \cdot 6.$$

The reader is invited to define the alternating protocol. Later, an attribute grammar with a single-use applied occurrence is presented that can be stored on the basic linear data structure with this unusual protocol.

The following two lemmas show that for single-use attribute instances the definitions of global stacks and queues are consistent with Definitions 12(1) and 12(2), respectively. Let A be a group of single-use instances in some derivation tree which is evaluated by a simple multi-visit evaluator.

Lemma 5. *A group A of single-use attribute instances can be allocated to a global stack if and only if it can be allocated to a basic stack.*

Proof. Let $\sigma = x_1 \cdots x_{2|A|}$ be the action-sequence of A. It has already been noted that there is always a data structure in $\mathcal{D}$, with *Data* the set of possible values in A, to which A can be allocated. Assume D is such a basic linear data structure with protocol p.

The lemma is proved if it is shown that:

(1) Protocol p cannot be the basic stack protocol if A cannot be allocated to a global stack.

(2) Protocol p, as far as it is used for the allocation of A, is the basic stack protocol if A can be allocated to a global stack.

In order to establish the proof of assertions (1) and (2), the following result is used (of which verification is left to the reader using Definition 7).

Claim. *Group A cannot be allocated to a global stack if and only if protocol p satisfies*

$$p(\sigma_1 i \sigma_2 i \sigma_3) = \#_i(\sigma_1 i) \quad \textit{and} \quad p(\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4) = \#_i(\sigma_1 i \sigma_2 i),$$

where, for some $m \in [4, 2|A|]$, $\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4 o = x_1 \cdots x_m$.

Proof of assertion (1). — Suppose A cannot be allocated to a global stack. Then by the claim, there must be a prefix $\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4 o$ of action sequence σ

for which p satisfies $p(\sigma_1 i \sigma_2 i \sigma_3) = \#_i(\sigma_1 i)$ and $p(\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4) = \#_i(\sigma_1 i \sigma_2 i)$. To obtain a contradiction, assume that p is the basic stack protocol. Consider the sequence $\tau_4 = \mathit{reduce}^*(\sigma_4)$, that is, the sequence obtained from σ_4 by applying *reduce* as many times as possible such that τ_4 either ends with an i or consists of zero or more symbols o. If τ_4 ends with an i, then the assumption that p is the basic stack protocol implies that

$$p(\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4) = p(\sigma_1 i \sigma_2 i \sigma_3 o \tau_4) > \#_i(\sigma_1 i \sigma_2 i).$$

This result contradicts the fact that $p(\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4) = \#_i(\sigma_1 i \sigma_2 i)$, so that $\tau_4 \in \{ o \}^*$. Similarly, it can be inferred that $\tau_3 = \mathit{reduce}^*(\sigma_3)$ must be in $\{ o \}^*$, for otherwise $p(\sigma_1 i \sigma_2 i \sigma_3) > \#_i(\sigma_i i)$ according to the assumption that p is the basic stack protocol. But, if $\tau_3, \tau_4 \in \{ o \}^*$, then $\mathit{reduce}^*(\sigma_3 o \sigma_4)$ consists of one or more symbols o. Thus $\#_i(\mathit{reduce}^*(\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4)) < \#_i(\sigma_1 i \sigma_2 i)$. From this fact, and the assumption that p is the basic stack protocol, it follows that

$$p(\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4)) = p(\mathit{reduce}^*(\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4)) < \#_i(\sigma_1 i \sigma_2 i).$$

However, this too contradicts the fact that $p(\sigma_1 i \sigma_2 i \sigma_3 o \sigma_4)) = \#_i(\sigma_1 i \sigma_2 i)$. Therefore it must be concluded that protocol p cannot be the basic stack protocol.

Proof of assertion (2).— Assume that A can be allocated to a global stack. Then it has to be shown that p, as far as it is used for allocation of A, satisfies the following two conditions (compare Definition 12(1)):

(a) If $x_l = i$ and $x_{l+1} = o$ holds for some $l \in [1, 2|A| - 1]$, then $p(x_1 \cdots x_l)$ must be $\#_i(x_1 \cdots x_l)$.

(b) If $x_l = o$ and $x_{l+1} = o$ holds for some $l \in [3, 2|A| - 1]$, then $p(x_1 \cdots x_l)$ must be $\#_i(\mathit{reduce}^*(x_1 \cdots x_l))$.

Demonstration that protocol p satisfies condition (a).— Assume p does not satisfy condition (a), so that there is an $l \in [1, 2|A|-1]$ such that $x_l = i$, $x_{l+1} = o$, and $p(x_1 \cdots x_l) < \#_i(x_1 \cdots x_l)$ holds. (Note that by definition $p(x_1 \cdots x_l) \leq \#_i(x_1 \cdots x_l)$.) Then there must be a sequence $x_{l+2} \cdots x_m$, for some $m \in [l+2, 2|A|-1]$, such that $p(x_1 \cdots x_{l-1} i o x_{l+2} \cdots x_m) = \#_i(x_1 \cdots x_l)$. This fact implies (see the claim above) that A cannot be allocated to a global stack, and so contradics the former assumption. Thus p, as far as it used for the allocation of A, satisfies condition (a).

Demonstration that protocol p satisfies condition (b).— Suppose p does not satisfy condition (b), so that there is an $l \in [3, 2|A|-1]$ for which $x_l = o$, $x_{l+1} = o$, and $p(x_1 \cdots x_l) \neq \#_i(\mathit{reduce}^*(x_1 \cdots x_l))$. Assume l to be the smallest integer in $[3, 2|A|-1]$ for which this property applies. Let $\sigma_1 i \sigma_2 o$ be $x_1 \cdots x_l$ such that $p(\sigma_1 i \sigma_2 o) = \#_i(\sigma_1 i)$. Then depending on $\mathit{reduce}^*(\sigma_2 o)$, three cases are distinguished.

(i) *Sequence $\mathit{reduce}^*(\sigma_2 o)$ ends with an i.*— By assuming that l is the smallest integer for which condition (b) does not hold and the fact that p satisfies condition (a), there is an $m \in [l+1, 2|A|-1]$ such that $p(\sigma_1 i \sigma_2 o o x_{l+2} \cdots x_m) = \#_i(\mathit{reduce}^*(\sigma_1 i \sigma_2 o))$. As $\mathit{reduce}^*(\sigma_2 o)$ ends with an i, $\#_i(\mathit{reduce}^*(\sigma_1 i \sigma_2 o)) \in [\#_i(\sigma_1 i) + 1, \#_i(\sigma_1 i \sigma_2 o)]$ so that by the claim A cannot be allocated to a global stack. This fact contradicts the former assumption.

(ii) *Sequence $\mathit{reduce}^*(\sigma_2 o)$ consists of symbols o only.*— This case is only possible if $\#_i(\sigma_2 o) < \#_o(\sigma_2 o)$. Thus, $\#_i(\mathit{reduce}^*(\sigma_1 i \sigma_2 o)) < \#_i(\sigma_1 i)$. Using the assumption that l is the smallest integer for which condition(b) does not hold and the fact that p satisfies condition (a), the existence of a prefix of $\sigma_2 o$, say τ, such that $p(\sigma_1 i \tau) = \#_i(\sigma_1 i)$ must

be inferred. Since $\tau \neq \sigma_2 o$, $p(\sigma_1 i \tau) = \#_i(\sigma_1 i)$ contradicts the assumption that $p(\sigma_1 i \sigma_2 o) = \#_i(\sigma_1 i)$. Hence $\mathit{reduce}^*(\sigma_2 o)$ cannot consist of symbols o only.

(iii) *Sequence* $\mathit{reduce}^*(\sigma_2 o) = \epsilon$. — Then $\mathit{reduce}^*(\sigma_1 i \sigma_2 o) = \sigma_1 i$. This fact implies that $p(\sigma_1 i \sigma_2 o) = \#_i(\mathit{reduce}^*(\sigma_1 i \sigma_2 o))$ holds, contradicting the assumption that $p(\sigma_1 i \sigma_2 o) \neq \#_i(\mathit{reduce}^*(\sigma_1 i \sigma_2 o))$. Thus $\mathit{reduce}^*(\sigma_2 o)$ cannot be ϵ.

So protocol p, as far as it used for the allocation of A, satisfies condition (b). This concludes the proof of this lemma. □

Lemma 6. *A group A of single-use attribute instances can be allocated to a global queue if and only if it can be allocated to a basic queue.*

Proof. This proof is similar to the proof of Lemma 5 and is thus omitted. The proof follows more directly since the queue protocol is simpler than the stack protocol. □

The theorem presented below shows that for simple multi-pass evaluators it is sufficient to consider stacks and queues for the global storage allocation of the instances of single-use applied occurrences.

Theorem 2. *Let E be a simple multi-pass evaluator for an attribute grammar G and let $p : X_0 \to X_1 \cdots X_n \in P$ with $(\alpha, p, j) \in AO(p)$ single-use. All the attribute instances of (α, p, j) can be allocated to a basic linear data structure in $\mathfrak{D}$ during evaluation by E if and only if E can allocate them to a global stack or a global queue.*

Proof. If all the attribute instances of (α, p, j) can be allocated to a global

stack or a global queue during evaluation by E, then by Lemmas 5 and 6, they can be allocated to a basic linear data structure as well.

A detailed proof of the converse statement is rather long, though not difficult. Suppose that the attribute instances of (α, p, j) cannot be allocated to a global stack or to a global queue. Then the following claim can be made.

Claim. *There are (at least) two define-and-use sequences in $DU_E(\alpha, p, j)$, say ϕ and ψ, such that $|\phi| = |\psi|$ and $\phi \neq \psi$.*

That this claim is sufficient to establish this part of the proof, can be seen from the following. Suppose that (α, p, j) can be allocated to the basic linear data structure D with behaviour f. From the assumption that (α, p, j) cannot be allocated to a global stack, it follows that (α, p, j) is nontemporary. Hence, the claimed define-and-use sequences ϕ and ψ have the form: $1 \cdot 2 \cdots k \cdot \phi'$ and $1 \cdot 2 \cdots k \cdot \psi'$, respectively, where $\phi' \neq \psi'$ and $k = \frac{1}{2}|\phi|$. (Note that the length of a define-and-use sequence in $DU_E(\alpha, p, j)$ is always even.) This implies that $f(i^k o^k) = \phi'$ and $f(i^k o^k) = \psi'$, which contradicts the fact that the behaviour f of D is a function.

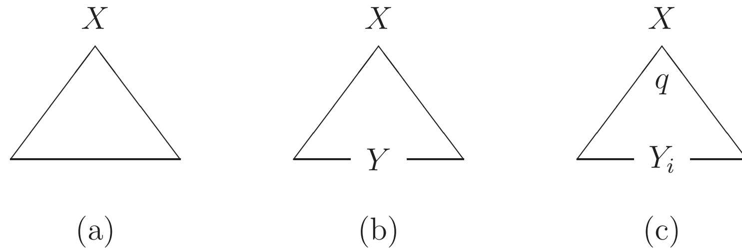

Figure 4. Pictorial denotations of context-free derivations.

To prove the claim, the following notation is useful. Figure 4(a) denotes that $X \Rightarrow^* w$ and Figure 4(b) denotes that $X \Rightarrow^* vYw$ where $X, Y \in N$ and

$v, w \in \Sigma^*$. Similarly, Figure 4(c) denotes that

$$X \Rightarrow w_0 Y_1 w_1 \cdots w_{i-1} Y_i w_i \cdots w_{m-1} Y_m w_m \Rightarrow^* v Y_i w,$$

with $w_0 Y_1 w_1 \cdots w_{i-1} \Rightarrow^* v$ and $w_i \cdots w_{m-1} Y_m w_m \Rightarrow^* w$ where $v, w \in \Sigma^*$, $i \in [1, m]$ and $q : X \to w_0 Y_1 w_1 \cdots w_{m-1} Y_m w_m \in P$.

Of course, every diagram in Figure 4 can be seen as a graphical representation of a derivation tree with root labeled X. In this conception, A_d denotes the group of attribute instances of (α, p, j) that occur in the tree represented by diagram d.

Proof of the Claim. The discussion will be separated into cases depending on the satisfied conditions of Lemmas 2 and 3. (Recall that (α, p, j) can neither be allocated to a global stack nor to a global queue.) The claim will be shown only for cases (a) and (b) below; the reader is invited to show the claim for the other cases using the same technique *mutatis mutandis*.

(a) Condition (3) of Lemma 2 as well as condition (1) of Lemma 3 is satisfied.

(b) Condition (1) of Lemma 2 as well as condition (1) of Lemma 3 is satisfied.

Note that the case in which condition (3) of both Lemmas 2 and 3 is satisfied simultaneously, can be ruled out. This can easily be seen using the fact that E follows a simple multi-pass strategy.

Proof for case (a).— Assume that production $q : Y_0 \to Y_1 \cdots Y_m$ satisfies condition (3) of Lemma 2 with respect to two right-hand side symbols, say Y_k and Y_l similar to this condition, and that right-hand side symbol X_h of production p satisfies condition (1) of Lemma 3. This assumption, together

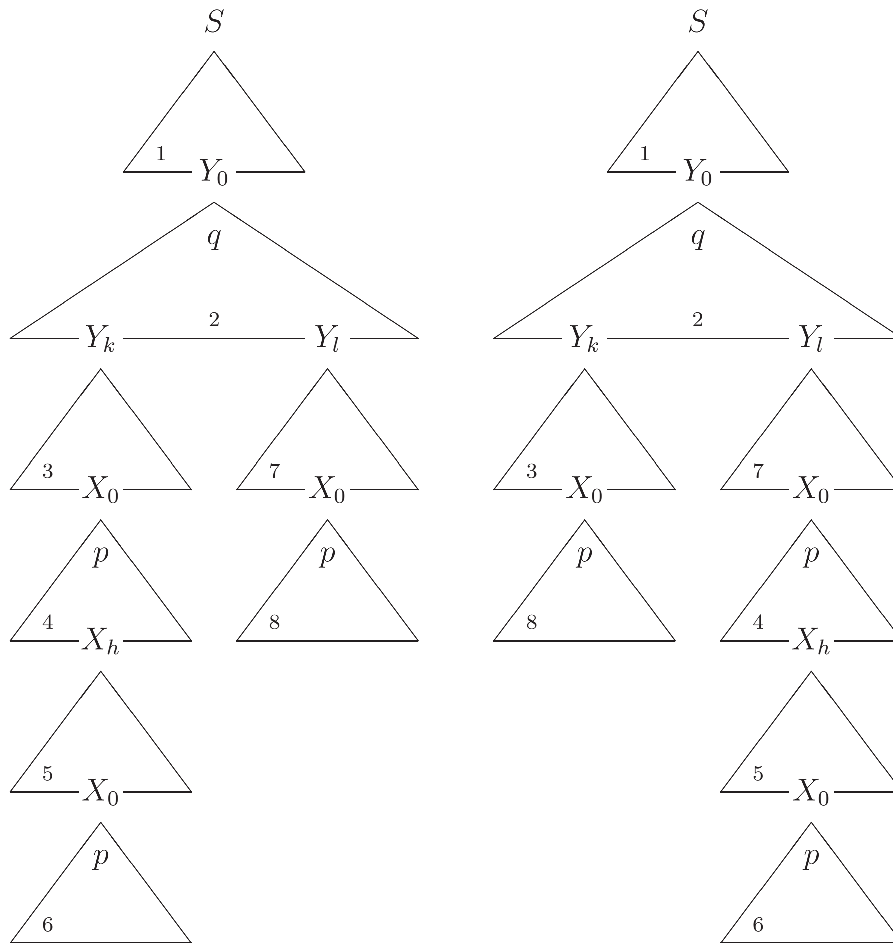

Figure 5. Derivation trees t_1 and t_2.

with the fact that G_u is reduced, implies the existence of the two derivation trees t_1 and t_2 shown in Figure 5. (Due to graphical limitations, Y_k is drawn to the left of Y_l although only $k \neq l$ is assumed.) Let $A_4 = \{ \beta \}$, $A_6 = \{ \gamma \}$, and $A_8 = \{ \delta \}$ for both trees t_1 and t_2. That is to say, the trees are constructed in such a way that these diagrams contain exactly one instance of the applied occurrence (α, p, j). It will be clear that this is possible (by taking the smallest such trees).

By construction, the inequalities

$$b_1 < d_1 < b_2 < d_2, \qquad b_1 < g_1 < g_2 < b_2, \qquad g_1 < d_1 < g_2 < d_2$$

hold for derivation tree t_1, and the inequalities

$$d_1 < b_1 < d_2 < b_2, \qquad b_1 < g_1 < g_2 < b_2, \qquad d_1 < g_1 < d_2 < g_2$$

hold for derivation tree t_2 where it is assumed that $\xi(\beta) = \langle b_1, b_2 \rangle$, $\xi(\gamma) = \langle g_1, g_2 \rangle$ and $\xi(\delta) = \langle d_1, d_2 \rangle$. From these inequalities, it is inferred that

$$b_1 < g_1 < d_1 < g_2 < b_2 < d_2 \tag{1}$$

holds for derivation tree t_1 and

$$d_1 < b_1 < g_1 < d_2 < g_2 < b_2 \tag{2}$$

holds for derivation tree t_2.

Now, let ϕ and ψ be the define-and-use sequences of (α, p, j) for derivation trees t_1 and t_2 respectively. Let $[\vartheta]_\omega$ denote the number in $\omega \in DU_E(\alpha, p, j)$ which encodes the order in which the attribute instance ϑ of (α, p, j) is computed and used in ω. Then, using the inequalities in (1), it follows that ϕ can be written as

$$\phi_1 \cdot [\beta]_\phi \cdot \phi_2 \cdot [\gamma]_\phi \cdot \phi_3 \cdot [\delta]_\phi \cdot \phi_4 \cdot \phi_5 \cdot [\gamma]_\phi \cdot \phi_6 \cdot [\beta]_\phi \cdot \phi_7 \cdot [\delta]_\phi \cdot \phi_8$$

where

$$\phi_1 \cdot [\beta]_\phi \cdot \phi_2 \cdot [\gamma]_\phi \cdot \phi_3 \cdot [\delta]_\phi \cdot \phi_4 = 1 \cdots n$$

$$|\, \phi_5 \cdot [\gamma]_\phi \cdot \phi_6 \cdot [\beta]_\phi \cdot \phi_7 \cdot [\delta]_\phi \cdot \phi_8 \,| = n$$

with $\phi_1, \ldots, \phi_8 \in [1, n]^*$ and $n = |A_1| + \cdots + |A_8|$.

Similarly, using the inequalities in (2) it follows that define-and-use sequence ψ can be written as

$$\psi_1 \cdot [\delta]_\psi \cdot \psi_2 \cdot [\beta]_\psi \cdot \psi_3 \cdot [\gamma]_\psi \cdot \psi_4 \cdot \psi_5 \cdot [\delta]_\psi \cdot \psi_6 \cdot [\gamma]_\psi \cdot \psi_7 \cdot [\beta]_\psi \cdot \psi_8$$

where

$$\psi_1 \cdot [\delta]_\psi \cdot \psi_2 \cdot [\beta]_\psi \cdot \psi_3 \cdot [\gamma]_\psi \cdot \psi_4 = 1 \cdots n$$

$$|\, \psi_5 \cdot [\delta]_\psi \cdot \psi_6 \cdot [\gamma]_\psi \cdot \psi_7 \cdot [\beta]_\psi \cdot \psi_8 \,| = n$$

with $\psi_1, \ldots, \psi_8 \in [1, n]^*$ and $n = |A_1| + \cdots + |A_8|$.

Thus, $|\phi| = |\psi|$ which was the first assertion of the claim. Since trees t_1 and t_2 are equal modulo the parts below diagrams 3 and 7 it follows that $\phi_1 = \psi_1$, so that $[\beta]_\phi = [\delta]_\psi$. This result, and the observation that (because E is simple multi-pass)

$$|\, \phi_5 \cdot [\gamma]_\phi \cdot \phi_6 \,| > |\, \psi_5 \,|$$

leads to the second assertion $\phi \neq \psi$ of the claim.

Proof for case (b). — Suppose that right-hand side symbol X_k of production p satisfies condition (1) of Lemma 2 and that right-hand side symbol X_h of p satisfies condition (1) of Lemma 3. Then, from the fact that G_u is reduced, the existence of the two derivation trees t_3 and t_4 in Figure 6 can be inferred. Let $A_2 = \{\, \beta \,\}$, $A_4 = \{\, \gamma \,\}$, and $A_6 = \{\, \delta \,\}$ for both derivation trees t_3 and t_4.

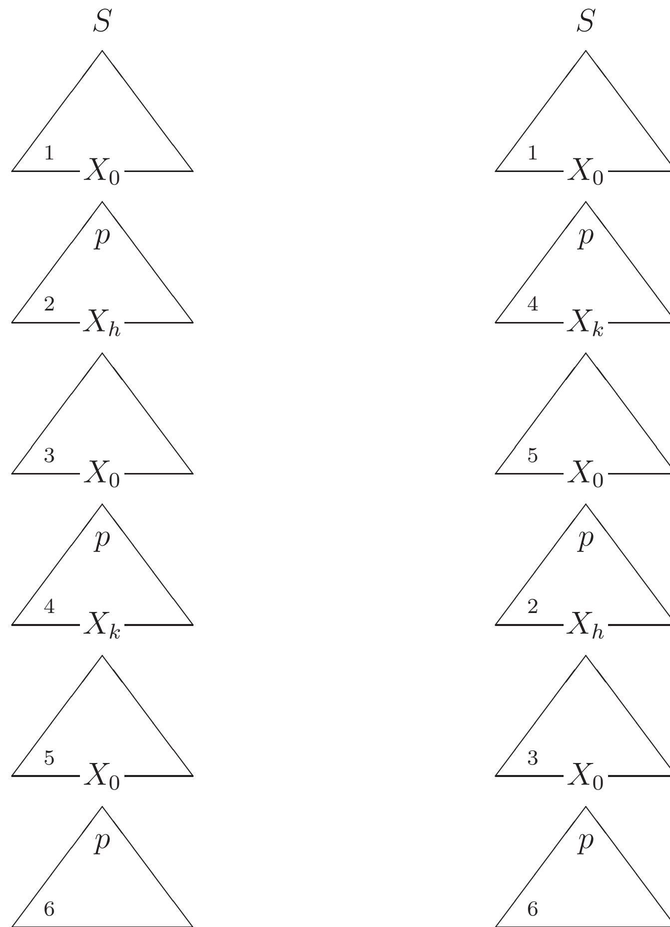

Figure 6. Derivation trees t_3 and t_4.

By construction, the inequalities

$$b_1 < d_1 < d_2 < b_2, \qquad b_1 < g_1 < g_2 < b_2, \qquad g_1 < d_1 < g_2 < d_2$$

hold for derivation tree t_3, and the inequalities

$$g_1 < d_1 < g_2 < d_2, \qquad g_1 < b_1 < g_2 < b_2, \qquad b_1 < d_1 < d_2 < b_2$$

hold for derivation tree t_4 where it is assumed that $\xi(\beta) = \langle b_1, b_2 \rangle$, $\xi(\gamma) = \langle g_1, g_2 \rangle$ and $\xi(\delta) = \langle d_1, d_2 \rangle$. These inequalities imply that

$$b_1 < g_1 < d_1 < g_2 < d_2 < b_2 \tag{3}$$

holds for derivation tree t_3, and

$$g_1 < b_1 < d_1 < g_2 < d_2 < b_2 \tag{4}$$

holds for derivation tree t_4.

Let ϕ and ψ denote the define-and-use sequences of (α, p, j) for trees t_3 and t_4 respectively. Then, using the inequalities in (3), it follows that ϕ can be written as

$$\phi_1 \cdot [\beta]_\phi \cdot \phi_2 \cdot [\gamma]_\phi \cdot \phi_3 \cdot [\delta]_\phi \cdot \phi_4 \cdot \phi_5 \cdot [\gamma]_\phi \cdot \phi_6 \cdot [\delta]_\phi \cdot \phi_7 \cdot [\beta]_\phi \cdot \phi_8$$

where

$$\phi_1 \cdot [\beta]_\phi \cdot \phi_2 \cdot [\gamma]_\phi \cdot \phi_3 \cdot [\delta]_\phi \cdot \phi_4 = 1 \cdots n$$

$$|\, \phi_5 \cdot [\gamma]_\phi \cdot \phi_6 \cdot [\delta]_\phi \cdot \phi_7 \cdot [\beta]_\phi \cdot \phi_8 \,| = n$$

with $\phi_1, \ldots, \phi_8 \in [1, n]^*$ and $n = |A_1| + \cdots + |A_6|$.

Similarly, using the inequalities in (4) it follows that define-and-use sequence ψ can be written as

$$\psi_1 \cdot [\gamma]_\psi \cdot \psi_2 \cdot [\beta]_\psi \cdot \psi_3 \cdot [\delta]_\psi \cdot \psi_4 \cdot \psi_5 \cdot [\gamma]_\psi \cdot \psi_6 \cdot [\delta]_\psi \cdot \psi_7 \cdot [\beta]_\psi \cdot \psi_8$$

where

$$\psi_1 \cdot [\gamma]_\psi \cdot \psi_2 \cdot [\beta]_\psi \cdot \psi_3 \cdot [\delta]_\psi \cdot \psi_4 = 1 \cdots n$$

$$|\, \psi_5 \cdot [\gamma]_\psi \cdot \psi_6 \cdot [\delta]_\psi \cdot \psi_7 \cdot [\beta]_\psi \cdot \psi_8 \,| = n$$

with $\psi_1, \ldots, \psi_8 \in [1, n]^*$ and $n = |A_1| + \cdots + |A_6|$.

Obviously, $|\phi| = |\psi|$, which is the first assertion of the claim. Again, from the fact that trees t_3 and t_4 are equal modulo the parts below diagram 1, $\phi_1 = \psi_1$ must hold so that $[\beta]_\phi = [\gamma]_\psi$. This result, and the observation that (because E is simple multi-pass)

$$|\, \phi_5 \cdot [\gamma]_\phi \cdot \phi_6 \cdot [\delta]_\phi \cdot \phi_7 \,| > |\, \psi_5 \,|$$

proves the second assertion $\phi \neq \psi$ of the claim. □

In general, Theorem 2 does not hold for simple multi-sweep attribute grammars. Analysis of the proof of this theorem shows that only in the case in which condition (3) of both Lemmas 2 and 3 are satisfied, are there basic linear data structures other than stacks and queues that can be used by a simple multi-sweep evaluator. This is shown in the next example.

Example 4. Consider the attribute grammar G which has nonterminals $N = \{ Z, A, B \}$, terminals $\Sigma = \varnothing$, start symbol Z and productions

$$P = \{ p_1 : Z \to B,\ p_2 : B_0 \to A_1 B_1 A_2,\ p_3 : B \to \epsilon,\ p_4 : A \to \epsilon \}.$$

The sets of attributes of each nonterminal are:

$$\begin{array}{lll} I(Z) = \varnothing & I(A) = \{ i \} & I(B) = \{ i \} \\ S(Z) = \{ s \} & S(A) = \{ s \} & S(B) = \{ s \} \end{array}$$

Figure 7 shows the relevant aspects of sets $R(p_1)$, $R(p_2)$, $R(p_3)$, and $R(p_4)$ by means of their dependency graphs.

Production Rule	Extended Visit-Sequence
p_1	$\langle \varnothing, \{ (i, p_1, 1) \}, visit_1(B), \varnothing, \varnothing \rangle$
	$\langle \varnothing, \varnothing, visit_2(B), \{ (s, p_1, 1) \}, \{ (s, p_1, 0) \} \rangle$
p_2	$\langle \{ (i, p_2, 0) \}, \{ (i, p_2, 1) \}, visit_1(A_1), \varnothing, \{ (i, p_2, 2) \}, visit_1(B_1), \varnothing, \{ (i, p_2, 3) \}, visit_1(A_2), \varnothing, \varnothing \rangle$
	$\langle \varnothing, \varnothing, visit_2(A_1), \{ (s, p_2, 1) \}, \varnothing, visit_2(A_2), \{ (s, p_2, 3) \}, \varnothing, visit_2(B_1), \{ (s, p_2, 2) \}, \{ (s, p_2, 0) \} \rangle$
p_3	$\langle \{ (i, p_3, 0) \}, \varnothing \rangle$
	$\langle \varnothing, \{ (s, p_3, 0) \} \rangle$
p_4	$\langle \{ (i, p_4, 0) \}, \varnothing \rangle$
	$\langle \varnothing, \{ (s, p_4, 0) \} \rangle$

Table 3. Extended visit-sequences of simple multi-sweep evaluator E.

Attribute grammar G is evaluated by simple multi-sweep evaluator E whose behaviour is given by the extended visit sequences of Table 3.

Note that for applied occurrence $(i, p_4, 0)$, production p_2 satisfies condition (3) of Lemma 2 with respect to symbols A_1 and A_2, and condition (3) of Lemma 3 with respect to symbols B_1 and A_2. Therefore applied occurrence $(i, p_4, 0)$ cannot be allocated to a global stack or a global queue.

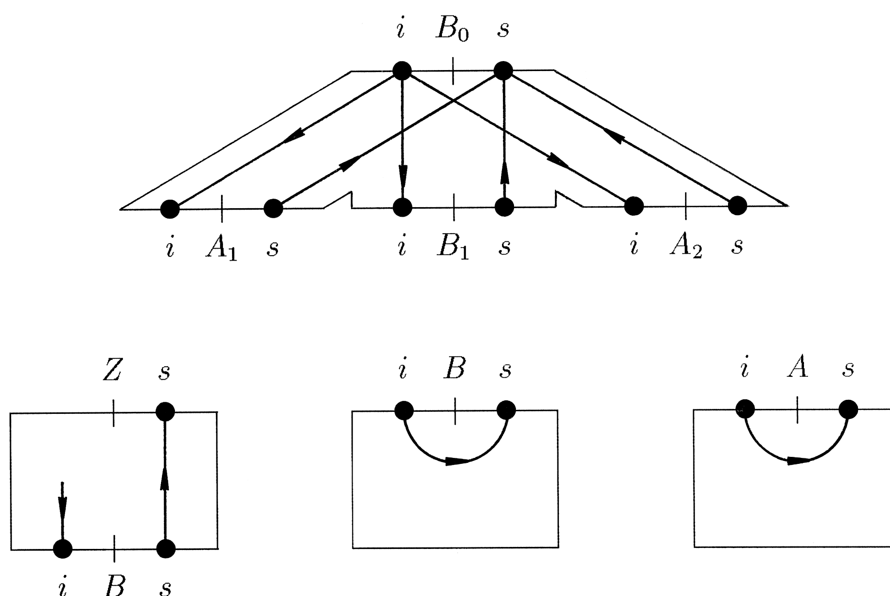

Figure 7. Dependency graphs $DG(p_2)$, $DG(p_1)$, $DG(p_3)$, and $DG(p_4)$.

Figure 8 shows a derivation tree of this attribute grammar and its evaluation graph. (The evaluation graph of a tree is the graph that represents the order in which the attribute instances of the tree are computed. It should not be confused with the dependency graph.) From this graph it can be seen that applied occurrence $(i, p_4, 0)$ can be allocated to a basic linear data structure with the alternating protocol (see page 42).

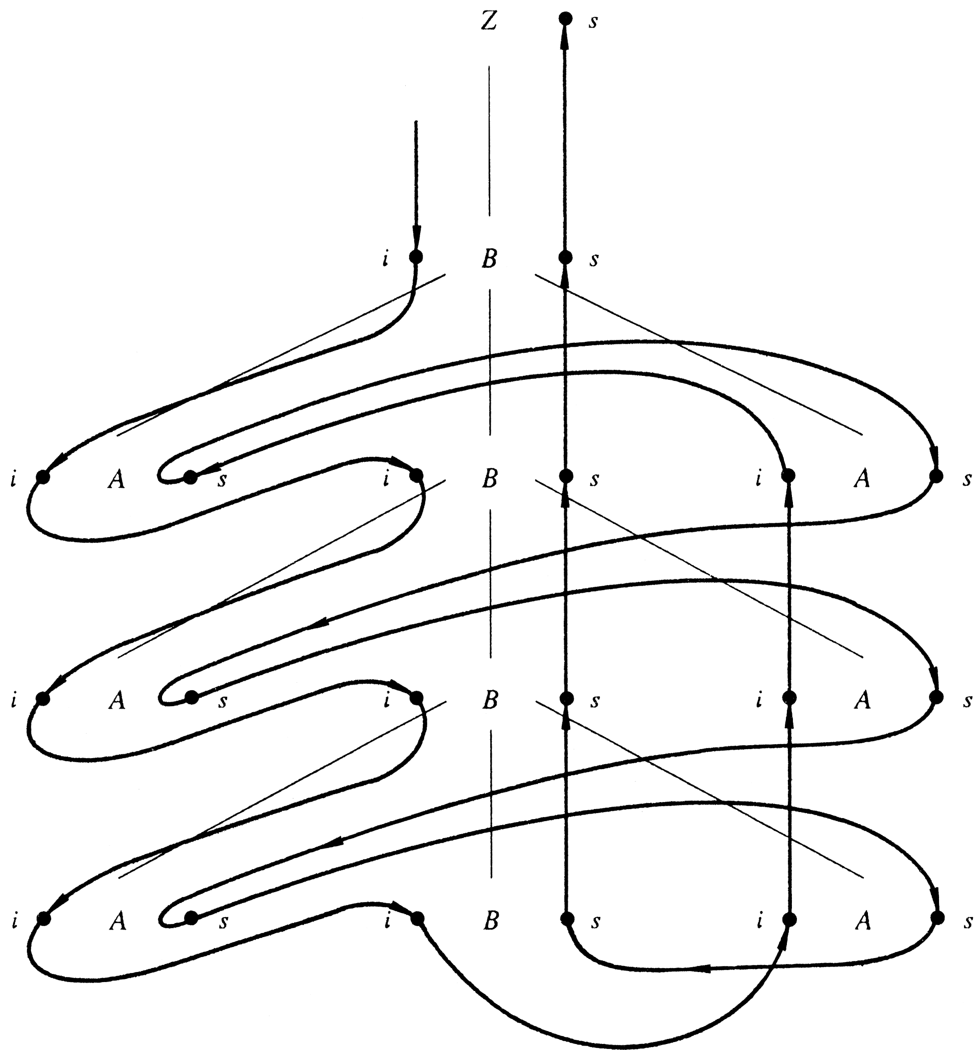

Figure 8. A derivation tree and its evaluation graph.

Of course, the evaluator given is certainly not the most efficient simple multi-sweep evaluator for this attribute grammar. However, the attribute grammar can be extended in a way that forces the specific order in which the instances of $(i, p_4, 0)$ are defined and used. $\square$

Remark. It is possible to transform the attribute grammar in Example 4 into a simple multi-waddle attribute grammar (see the remark on page 20), by splitting production p_2 into two productions with only two nonterminals in the right-hand side. Using the attribute grammar obtained by this transformation, it can be shown further that the result stated in Theorem 2 cannot be extended to the more restricted class of simple multi-waddle attribute grammars. $\square$

The following theorem asserts that for simple multi-sweep evaluators, it is sufficient to consider a proper subset of $\mathcal{D}$ for the global storage allocation of the instances of single-use applied occurrences. In contrast to simple multi-pass evaluators, however, this subset is not finite.

Theorem 3. *Let E be a simple multi-sweep evaluator for an attribute grammar G with single-use applied occurrence (α, p, j). Assume $\mathcal{S} \subset \mathcal{D}$ to be a set consisting of the basic stack and all other data structures in $\mathcal{D}$ such that*

$$D, D' \in \mathcal{S} \text{ if and only if } f(i^n o^n) \neq f'(i^n o^n) \text{ for some } n \in \mathbb{N}^+$$

where f and f' denote the behaviour of D and D', respectively. Apart from variants ($\mathcal{S}$ is not unique), $\mathcal{S}$ is the smallest possible subset of $\mathcal{D}$ with the property that all the attribute instances of (α, p, j) can be allocated to a data structure in $\mathcal{D}$ during evaluation by E if and only if E can allocate them to a data structure in $\mathcal{S}$.

Proof. First of all, it will be shown that $\mathcal{S}$ satisfies the claimed property that the attribute instances of (α, p, j) can be allocated to a $D \in \mathcal{D}$ during evaluation by E if and only if E can allocate them to a $D' \in \mathcal{S}$. Let (β, p, i) be the (unique) defined occurrence depending on (α, p, j). Two cases can be identified.

(1) *Occurrences* (α, p, j) *and* (β, p, i) *occur in the same subsequence of* V_p.— Then, by Lemma 2, E can always allocate the instances of (α, p, j) to a global stack and thus, by Lemma 5, to a basic stack. Since the basic stack is contained in both $\mathcal{D}$ and $\mathcal{S}$, it follows that $\mathcal{S}$ satisfies the claimed property.

(2) *Occurrences* (α, p, j) *and* (β, p, i) *occur in different subsequences of* V_p.— Then each define-and-use sequence $\psi \in DU_E(\alpha, p, j)$ has the form $1 \cdot 2 \cdots m \cdot \psi'$, where $|\psi'| = m$ for some $m \in \mathbb{N}^+$. Now there is always a $D \in \mathcal{S}$ for each $D' \in \mathcal{D}$ such that $f(i^n o^n) = f'(i^n o^n)$ for all $n \in \mathbb{N}^+$ where f and f' denote the behaviour of D and D' respectively. Thus, by Lemma 4 and the fact that each action sequence obtained from a sequence $\psi \in DU_E(\alpha, p, j)$ has the form $i^n o^n$ with $n = \frac{1}{2}|\psi|$, there is always a $D \in \mathcal{S}$ to which E can allocate the instances of (α, p, j) if E can allocate them to a $D' \in \mathcal{D}$. And since $\mathcal{S} \subset \mathcal{D}$, it is clear that $\mathcal{S}$ satisfies the claimed property.

The assertion that $\mathcal{S}$ (apart from variants) is the smallest subset of $\mathcal{D}$ which satisfies the claimed property, is proved by construction of an attribute grammar G' with a simple multi-sweep evaluator E' for any $D \in \mathcal{S}$ such that E' cannot allocate the instances of a single-use applied occurrence of G' to any other $D' \in \mathcal{S}$ than D.

Let $D \in \mathcal{S}$ with protocol p and let $n \in \mathbb{N}^+$.

(1) *Construction of attribute grammar G'.*— $G' = (G_u, A, V, R)$ has start symbol Z, nonterminals $N = \{Z, A, B\}$, and terminals $\Sigma = \{a\}$. The production rules in set P are, for all $k \in [1, n]$, given as

$$q_k : Z \rightarrow B_1 \cdots B_k \qquad q_{n+1} : B \rightarrow A \qquad q_{n+2} : A \rightarrow a$$

The sets of attributes for the nonterminals of attribute grammar G' are $A(Z) = A(B) = \varnothing$ and $A(A) = \{\, \alpha, \beta \,\}$, where $I(A) = \{\, \alpha \,\}$ and $S(A) = \{\, \beta \,\}$. The sets of semantic rules are, for all $k \in [1, n]$, given as

$$\begin{aligned} R(q_k) &= \varnothing \\ R(q_{n+1}) &= \{\, (\alpha, q_{n+1}, 1) = c \,\} \\ R(q_{n+2}) &= \{\, (\beta, q_{n+2}, 0) = (\alpha, q_{n+2}, 0) \,\} \end{aligned}$$

where $c \in V(\alpha)$ and $V(\beta) = V(\alpha)$. This concludes the description of attribute grammar G'.

(2) *Construction of simple multi-sweep evaluator E' for G'.*— Simple multi-sweep evaluator E' for attribute grammar G' has, for all $k \in [1, n]$, the following extended visit-sequences

$$\begin{aligned} V_{q_k} &= \langle \varnothing, \varnothing, visit_1(B_1), \varnothing, \ldots, \varnothing, visit_1(B_k), \varnothing, \varnothing \rangle \\ &\quad \langle \varnothing, \varnothing, visit_2(B_{p(i^k o^0)}), \varnothing, \ldots, \varnothing, visit_2(B_{p(i^k o^{k-1})}), \varnothing, \varnothing \rangle \\ V_{q_{n+1}} &= \langle \varnothing, \{\, (\alpha, q_{n+1}, 1) \,\}, visit_1(A), \varnothing, \varnothing \rangle \\ &\quad \langle \varnothing, \varnothing, visit_2(A), \{\, (\beta, q_{n+1}, 1) \,\}, \varnothing, \rangle \\ V_{q_{n+2}} &= \langle \{\, (\alpha, q_{n+2}, 0) \,\}, \varnothing \rangle \ \langle \varnothing, \{\, (\beta, q_{n+2}, 0) \,\} \rangle \end{aligned}$$

It should be clear that the set of define-and-use sequences of $(\alpha, q_{n+2}, 0)$ induced by these extended visit-sequences can be given as

$$DU_E(\alpha, q_{n+2}, 0) = \{\, 1 \cdots k \cdot p(i^k o^0) \cdots p(i^k o^{k-1}) \mid k \in [1, n] \,\}.$$

Thus, by Definition 11 and Lemma 4, E' is such that the instances of $(\alpha, q_{n+2}, 0)$ can always be allocated to data structure D. This ends the description of evaluator E'.

By construction, the behaviour required for a $D' \in \mathcal{S}$ such that E' can allocate the instances of $(\alpha, q_{n+2}, 0)$ to D' can be made as close to the behaviour of D as wished, provided n is chosen sufficiently large. So when n takes on arbitrarily large values, that is when $n \to \infty$, E' cannot allocate the instances of $(\alpha, q_{n+2}, 0)$ to any $D' \in \mathcal{S}$ other than D. (Note that n stays finite when n approaches infinity so that G' and E' are always properly defined.) This proves that $\mathcal{S}$ (apart from variants) is the smallest subset of $\mathcal{D}$ which satisfies the claimed property. □

Since they are more general than simple multi-sweep evaluators, the question arises whether for simple multi-visit evaluators it remains sufficient to consider a proper subset of $\mathcal{D}$ for the global storage allocation of the instances of single-use applied occurrences. The following theorem settles this question.

Theorem 4. *Let E be a simple multi-visit evaluator for an attribute grammar G with single-use applied occurrence (α, p, j). There is no subset $\mathcal{V} \subset \mathcal{D}$ which satisfies the following property: all the attribute instances of (α, p, j) can be allocated to a data structure in $\mathcal{D}$ during evaluation by E if and only if E can allocate them to a data structure in $\mathcal{V}$.*

Proof. The theorem is proved by construction of an attribute grammar G' with a simple multi-visit evaluator E' for any $D \in \mathcal{D}$ such that E' cannot allocate the instances of a single-use applied occurrence of G' to any other $D' \in \mathcal{D}$ than D.

Let $D \in \mathcal{D}$ with protocol p, and let $n \in \mathbb{N}^+$.

(1) *Construction of attribute grammar G'.*— Assume S is the set of all sequences $\sigma \in \{ i, o \}^+$ for which $p(\sigma)$ is defined and $\#_i(\sigma) \leq n$. (Note that $\#_i(\sigma) \geq 1$ for all $\sigma \in S$.) Attribute grammar $G' = (G_u, A, V, R)$ has start symbol Z, nonterminals $N = \{Z, A, B\}$, terminals $\Sigma = \{a\}$, and productions P, for all $\sigma \in S$, consisting of

$$q_\sigma : Z \to B_1 \cdots B_{\#_i(\sigma)} \qquad q_1 : B \to A \qquad q_2 : A \to a$$

The sets of attributes for the nonterminals are $A(Z) = A(B) = \varnothing$ and $A(A) = \{ \alpha, \beta \}$ where $I(A) = \{ \alpha \}$ and $S(A) = \{ \beta \}$. The sets of semantic rules are, for all $\sigma \in S$, given as

$$R(q_\sigma) = \varnothing$$

$$R(q_1) = \{ (\alpha, q_1, 1) = c \}$$

$$R(q_2) = \{ (\beta, q_2, 0) = (\alpha, q_2, 0) \}$$

where $c \in V(\alpha)$ and $V(\beta) = V(\alpha)$. This concludes the description of attribute grammar G'.

(2) *Construction of simple multi-visit evaluator E' for G'.*— Simple multi-visit evaluator E' for attribute grammar G' is, for all $\sigma \in S$, specified by the extended visit-sequences of Table 4. Assume $\sigma = w_1 \cdots w_m$ with $w_{2j} \in \{ o \}^+$ and $w_{2j-1} \in \{ i \}^+$ for all $j \in [1, \lfloor \frac{1}{2} m \rfloor]$.

Production Rule	Extended Visit-Sequence
q_σ	$\langle \varnothing, \varnothing, \mathit{visit}_1(B_1), \varnothing, \ldots, \varnothing, \mathit{visit}_1(B_{\#_i(w_1)}), \varnothing, \varnothing \rangle$
(σ ends with i)	$\langle \varnothing, \varnothing, \mathit{visit}_2(B_{p(w_1 \cdot o^0)}), \varnothing, \ldots, \varnothing, \mathit{visit}_2(B_{p(w_1 \cdot o^{\lvert w_2 \rvert - 1})}), \varnothing, \varnothing \rangle$
	$\langle \varnothing, \varnothing, \mathit{visit}_3(B_{\#_i(w_1 \cdot w_2)+1}), \varnothing, \ldots, \varnothing, \mathit{visit}_3(B_{\#_i(w_1 \cdots w_3)}), \varnothing, \varnothing \rangle$
	$\langle \varnothing, \varnothing, \mathit{visit}_4(B_{p(w_1 \cdots w_3 \cdot o^0)}), \varnothing, \ldots, \varnothing, \mathit{visit}_4(B_{p(w_1 \cdots w_3 \cdot o^{\lvert w_4 \rvert - 1})}), \varnothing, \varnothing \rangle$
	$\vdots \qquad \vdots$
	$\langle \varnothing, \varnothing, \mathit{visit}_m(B_{\#_i(w_1 \cdots w_{m-1})+1}), \varnothing, \ldots, \varnothing, \mathit{visit}_m(B_{\#_i(w_1 \cdots w_m)}), \varnothing, \varnothing \rangle$
q_σ	$\langle \varnothing, \varnothing, \mathit{visit}_1(B_1), \varnothing, \ldots, \varnothing, \mathit{visit}_1(B_{\#_i(w_1)}), \varnothing, \varnothing \rangle$
(σ ends with o)	$\langle \varnothing, \varnothing, \mathit{visit}_2(B_{p(w_1 \cdot o^0)}), \varnothing, \ldots, \varnothing, \mathit{visit}_2(B_{p(w_1 \cdot o^{\lvert w_2 \rvert - 1})}), \varnothing, \varnothing \rangle$
	$\langle \varnothing, \varnothing, \mathit{visit}_3(B_{\#_i(w_1 \cdot w_2)+1}), \varnothing, \ldots, \varnothing, \mathit{visit}_3(B_{\#_i(w_1 \cdots w_3)}), \varnothing, \varnothing \rangle$
	$\vdots \qquad \vdots$
	$\langle \varnothing, \varnothing, \mathit{visit}_m(B_{p(w_1 \cdots w_{m-1} \cdot o^0)}), \varnothing, \ldots, \varnothing, \mathit{visit}_m(B_{p(w_1 \cdots w_{m-1} \cdot o^{\lvert w_m \rvert - 1})}), \varnothing, \varnothing \rangle$
q_1	$\langle \varnothing, \{ (\alpha, q_1, 1) \}, \mathit{visit}_1(A), \varnothing, \varnothing \rangle$
	$\langle \varnothing, \varnothing, \mathit{visit}_2(A), \{ (\beta, q_1, 1) \}, \varnothing \rangle$
q_2	$\langle \{ (\alpha, q_2, 0) \}, \varnothing \rangle$
	$\langle \varnothing, \{ (\beta, q_2, 0) \} \rangle$

Table 4. Extended visit-sequences of simple multi-visit evaluator E'.

Although perhaps not immediately obvious, it is not difficult to see that each define-and-use sequence of $(\alpha, q_2, 0)$ has the structure

$$
\begin{aligned}
&1 \cdots \#_i(w_1)\cdot \\
&\quad p(w_1 \cdot o^0) \cdots p(w_1 \cdot o^{|w_2|-1})\cdot \\
&\qquad (\#_i(w_1 \cdot w_2) + 1) \cdots \#_i(w_1 \cdots w_3)\cdot \\
&\qquad\quad p(w_1 \cdots w_3 \cdot o^0) \cdots p(w_1 \cdots w_3 \cdot o^{|w_4|-1})\cdot \\
&\qquad\qquad\qquad\qquad \cdots \\
&\qquad\qquad\qquad (\#_i(w_1 \cdots w_{m-1}) + 1) \cdots \#_i(w_1 \cdots w_m)
\end{aligned}
$$

or

$$
\begin{aligned}
&1 \cdots \#_i(w_1)\cdot \\
&\quad p(w_1 \cdot o^0) \cdots p(w_1 \cdot o^{|w_2|-1})\cdot \\
&\qquad (\#_i(w_1 \cdot w_2) + 1) \cdots \#_i(w_1 \cdots w_3)\cdot \\
&\qquad\qquad\qquad \cdots \\
&\qquad\qquad p(w_1 \cdots w_{m-1} \cdot o^0) \cdots p(w_1 \cdots w_{m-1} \cdot o^{|w_m|-1})
\end{aligned}
$$

so that, by Definition 11 and Lemma 4, E' can always allocate the instances of $(\alpha, q_2, 0)$ to data structure D. This ends the description of evaluator E'.

As can be seen from the construction above, the behaviour required for a $D' \in \mathcal{D}$ such that E' can allocate the instances of $(\alpha, q_2, 0)$ to D', can be made as close to the behaviour of D as wished by taking n large enough. Thus when $n \to \infty$, evaluator E' (which together with attribute grammar G' is properly defined because n stays finite) cannot allocate the instances of $(\alpha, q_2, 0)$ to any $D' \in \mathcal{D}$ other than D. This concludes the proof of the theorem. $\square$

As a final comment, observe that for simple multi-visit evaluators it is necessary and sufficient to consider $\mathcal{D}$ for the global storage allocation of the instances of single-use applied occurrences: necessary by Theorem 4, and sufficient by the characterization of $\mathcal{D}$.

CHAPTER FOUR

Global Storage Allocation of Multi-use Applied Occurrences

Results were presented in Chapter 3 for the global storage allocation of the attribute instances of single-use applied occurrences in simple multi-X evaluators where $X \in \{\text{pass, sweep, visit}\}$. This chapter extends these results in two ways, firstly by generalizing to multi-use applied occurrences and secondly by generalizing to simple multi-visit evaluators.

Generalization to multi-use applied occurrences. — The generalization to multi-use applied occurrences is achieved using the idea of reallocation. Consider an attribute grammar with a simple multi-visit evaluator E. Let

$$\textbf{set } (\beta, p, i) \textbf{ to } f((\alpha, p, j))$$
$$\textbf{set } (\gamma, p, k) \textbf{ to } g((\delta, p, l), (\alpha, p, j))$$

be all the evaluation rules of E using the instances of multi-use applied occurrence (α, p, j). Suppose that each instance of (α, p, j) is used to compute an instance of (β, p, i) before it is used to compute an instance of (γ, p, k).

Let D_1 and D_2 denote two data structures of $\mathcal{D}$. Assume, as illustrated in Figure 9, that each instance of (α, p, j) can be allocated to D_1 after its computation and reallocated to D_2 (that is, removed from D_1 and stored in D_2) immediately after it is used to compute an instance of (β, p, i). Then each instance of (α, p, j) has a single-use storage item for D_1 and a single-use storage item for D_2. Each single-use storage item, in turn, can be shown (using the proof of Theorem 7) to be seen as an instance of a single-use applied occurrence. Thus, (α, p, j) can be thought of as being composed of a single-use applied occurrence whose instances are allocated to D_1, and a single-use applied occurrence whose instances are allocated to D_2. This observation forms the crux of the idea of reallocation, and *ipso facto* of the generalization to multi-use applied occurrences. All results presented in Chapter 3 will be generalized to multi-use applied occurrences.

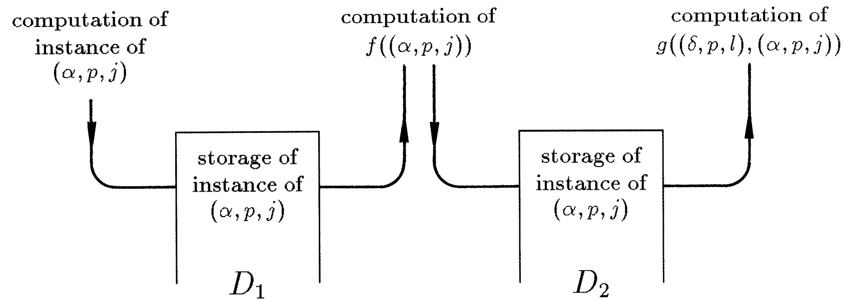

Figure 9. Schematic representation of the reallocation idea.

Generalization to simple multi-visit evaluators. — Theorem 1 will be generalized to multi-use applied occurrences as well as to simple multi-visit evaluators. To realize this generalization, some additional notations and concepts will be presented.

Firstly, the intuitive meaning of the concept (a, b)-allocated will be explained. Let E be a simple multi-visit evaluator for an attribute grammar with a multi-use applied occurrence (α, p, j). Let $(\alpha, p, j)_k$ for $k > 0$ denote the defined occurrence (β, p, i) whose instances are computed by E using the instances of (α, p, j) for the k-time. Let $(\alpha, p, j)_k$ for $k = 0$ denote applied occurrence (α, p, j) itself. By saying that the attribute instances of (α, p, j) are (a, b)-allocated to a data structure D during evaluation by E where $a < b$, a particular use of D by E for the storage of the instances of (α, p, j) is meant. Depending on integer a two cases are distinguished.

(1) *Integer* $a = 0$. — In this case D is used by E in the following manner. Each instance of (α, p, j), when computed, is stored in D; when the directly dependent instances of $(\alpha, p, j)_1, \ldots, (\alpha, p, j)_b$ are computed, the instance is accessed in D; just after the computation of the directly dependent instance of $(\alpha, p, j)_b$, it is removed from D.

(2) *Integer* $a > 0$. — In this case D is used by E in the following manner. Just after each instance of (α, p, j) is used to compute the directly dependent instance of $(\alpha, p, j)_a$, it is stored in D; when the directly dependent instances of $(\alpha, p, j)_{a+1}, \ldots, (\alpha, p, j)_b$ are computed, it is accessed in D; just after the computation of the directly dependent instance of $(\alpha, p, j)_b$, it is removed from D.

Note that data structure D can no longer be a basic linear data structure when $b > a + 1$.

Having explained the intuitive meaning of the concept (a, b)-allocated, it is now possible to precisely define what it means to state that the instances of an applied occurrence (α, p, j) can be (a, b)-allocated to a global

variable, stack, or queue during evaluation by a simple multi-visit evaluator. Definition 8, for example, can be stated as follows:

> A group A of attribute instances of (α, p, j) can be (a, b)-*allocated to a global queue* if for all $\gamma \in A$, with $\xi(\gamma) = \langle g_0, \ldots, g_n \rangle$, the following holds: there is no $\delta \in A$ with $\xi(\delta) = \langle d_0, \ldots, d_n \rangle$, such that $g_a < d_a$ and $d_{a+1} < g_b$.

Similarly, it is also possible to precisely define what it means to state that the instances of an applied occurrence can be (a, b)-allocated to a basic linear data structure during evaluation by a simple multi-visit evaluator. The reader should have no difficulty in transcribing Definitions 6, 7, and 10 as well as proving the analogue of Lemmas 1 and 4 for the (a, b)-allocation of the instances of a multi-use applied occurrence, where b should be taken as $a+1$ for the transcriptions of Definition 10 and Lemma 4. Note that the $(0, 1)$-allocation of a multi-use applied occurrence (α, p, j) is already considered in Chapter 3.

At this point it is straightforward to modify Lemmas 2 and 3 in such a way that it is decidable whether, given some simple multi-sweep evaluator, the instances of a multi-use applied occurrence can be $(a, a+1)$-allocated to a global stack or a global queue. However, instead of so doing, following on from Engelfriet and De Jong in [10] two lemmas shall be developed which are valid for the more general class of simple multi-visit evaluators. This development starts with the introduction of the concept of subtree information.

Let E be a simple multi-visit evaluator for an attribute grammar G with a multi-use applied occurrence (α, p, j). Denote by t a derivation tree of G and by n a node of t. Assume $(\beta, p, k) = (\alpha, p, j)_i$ and $(\gamma, p, l) = (\alpha, p, j)_{i+1}$.

Definition 13. The *subtree information* $SI_i(\alpha, p, j)$ for the $(i, i+1)$-allocation of (α, p, j) is a set of tuples $(X, (a, b))$ with $X \in N$, $a, b \in [1, \phi(X)]$, and $a \leq b$. A tuple $(X, (a, b)) \in SI_i(\alpha, p, j)$ when there is a subtree $t_{(n)}$, n labelled by X, that has a node m at which production p is applied. This subtree is evaluated by E (as part of t) in such a way that the instance of $(\alpha, p, j)_i$ at node m_k is computed during the a-th visit to n and the instance of $(\alpha, p, j)_{i+1}$ at node m_l is computed during the b-th visit to n. □

To express the constructability of set $SI_i(\alpha, p, j)$ the following result is presented using the extended visit sequences of evaluator E. For all elements a in V_p, let $a \in^+ V_p(n)$ denote that a occurs in sequence $V_p(n)$. For all sets $I_{p,n}(X_m), S_{p,n}(X_m) \in^+ V_p(n)$, let $(\delta, p, m) \in^* V_p(n)$ denote that occurrence $(\delta, p, m) \in I_{p,n}(X_m)$ or $(\delta, p, m) \in S_{p,n}(X_m)$.

Lemma 7. *$SI_i(\alpha, p, j) = SI_i{}'(\alpha, p, j)$, where $SI_i{}'(\alpha, p, j)$ is defined by steps (1)-(3) as follows.*

(1) Basis. — If $(\alpha, p, j)_i \in^* V_p(a)$ and $(\alpha, p, j)_{i+1} \in^* V_p(b)$, then tuple $(X_0, (a, b)) \in SI_i{}'(\alpha, p, j)$ where p is assumed to be $X_0 \to X_1 \cdots X_n$.

(2) Projection. — If $(X, (r, s)) \in SI_i{}'(\alpha, p, j)$ and there is a production $q : Y_0 \to Y_1 \cdots Y_m$ with nonterminal $Y_k = X$ for some $k \in [1, m]$ such that $visit_r(Y_k) \in^+ V_q(a)$ and $visit_s(Y_k) \in^+ V_q(b)$ holds, then tuple $(Y_0, (a, b)) \in SI_i{}'(\alpha, p, j)$.

(3) Closure. — Nothing is in $SI_i{}'(\alpha, p, j)$ except those tuples which are in $SI_i{}'(\alpha, p, j)$ by applying the basis step (1) and the projection step (2) on a finite number of occasions.

Proof. To prove $SI_i(\alpha, p, j) \subseteq SI_i{}'(\alpha, p, j)$, assume $(X, (a, b)) \in SI_i(\alpha, p, j)$. Then by Definition 13, there is a subtree $t_{(n)}$ with root n labelled by X and a

node m at which production p is applied, such that the instance of $(\alpha, p, j)_i$ at node m_k and the instance of $(\alpha, p, j)_{i+1}$ at node m_l are computed during the a-th and the b-th visit to node n respectively. Let λ be the length of the path from node n to node m in $t_{(n)}$. Now $(X, (a, b)) \in SI_i'(\alpha, p, j)$ follows by induction on path length λ.

To prove $SI_i'(\alpha, p, j) \subseteq SI_i(\alpha, p, j)$, assume $(X, (a, b)) \in SI_i'(\alpha, p, j)$. If tuple $(X, (a, b))$ is added to set $SI_i'(\alpha, p, j)$ by application of the basis step, then it is clear that tuple $(X, (a, b)) \in SI_i(\alpha, p, j)$ because G_u is reduced. Now assume that tuple $(X, (a, b))$ is added to $SI_i'(\alpha, p, j)$ by application of the projection step, and that all elements already in $SI_i'(\alpha, p, j)$ are also elements of $SI_i(\alpha, p, j)$. Then, again by the fact that G_u is reduced, it can be shown that tuple $(X, (a, b)) \in SI_i(\alpha, p, j)$. □

Example 5. Referring to Example 1, by the basis step and the projection step of Lemma 7, it is found that

$$SI_0(\alpha, p_2, 1) = \{\, (Z, (1, 1)),\ (A, (1, 1)) \,\} = SI_0(\alpha, p_3, 1),$$

$$SI_1(\alpha, p_2, 1) = \{\, (Z, (1, 2)),\ (A, (1, 2)) \,\} = SI_1(\alpha, p_3, 1),$$

$$SI_2(\alpha, p_2, 1) = \{\, (Z, (2, 2)),\ (A, (2, 2)) \,\},$$

and $$SI_0(\alpha, p_1, 1) = \{\, (Z, (1, 2)) \,\}.$$ □

Remark. In the case that E is simple multi-sweep (as in Example 1), it can be seen from Lemma 7 that tuple $(X, (a, b)) \in SI_i(\alpha, p, j)$ if and only if $(\alpha, p, j)_i \in^* V_p(a)$, $(\alpha, p, j)_{i+1} \in^* V_p(b)$ and $X \Rightarrow^* \gamma X_0 \delta$ for some $\gamma, \delta \in V^*$ where p is $X_0 \rightarrow X_1 \cdots X_n$. □

Now it will be shown how the subtree information $SI_i(\alpha, p, j)$ is used to decide whether the attribute instances of an applied occurrence (α, p, j) can

be $(i, i+1)$-allocated to a global stack or queue during evaluation by a simple multi-visit evaluator E.

Lemma 8. *Let E be a simple multi-visit evaluator for an attribute grammar G and let $p : X_0 \to X_1 \cdots X_n \in P$ with $(\alpha, p, j) \in AO(p)$ multi-use. The attribute instances of (α, p, j) cannot be $(i, i+1)$-allocated to a global stack during evaluation by E if and only if at least one of the following conditions is satisfied.*

(1) There exists a tuple $(X_k, (a, b)) \in SI_i(\alpha, p, j)$, for some $k \in [1, n]$, such that

$$set(\alpha, p, j)_i < visit_a(X_k) < set(\alpha, p, j)_{i+1} < visit_b(X_k) \text{ in } V_p.$$

(2) There exists a tuple $(X_k, (a, b)) \in SI_i(\alpha, p, j)$, for some $k \in [1, n]$, such that

$$visit_a(X_k) < set(\alpha, p, j)_i < visit_b(X_k) < set(\alpha, p, j)_{i+1} \text{ in } V_p.$$

(3) There exist a production $q : Y_0 \to Y_1 \cdots Y_m$, and tuples $(Y_k, (a, b))$, $(Y_l, (r, s)) \in SI_i(\alpha, p, j)$ for some distinct $k, l \in [1, m]$, such that

$$visit_a(Y_k) < visit_r(Y_l) < visit_b(Y_k) < visit_s(Y_l) \text{ in } V_q. \qquad \square$$

Lemma 9. *Let E be a simple multi-visit evaluator for an attribute grammar G and let $p : X_0 \to X_1 \cdots X_n \in P$ with $(\alpha, p, j) \in AO(p)$ multi-use. The attribute instances of (α, p, j) cannot be $(i, i+1)$-allocated to a global queue during evaluation by E if and only if at least one of the following conditions is satisfied.*

(1) There exists a tuple $(X_k, (a, b)) \in SI_i(\alpha, p, j)$, *for some* $k \in [1, n]$, *such that*

$$set(\alpha, p, j)_i < visit_a(X_k) \text{ and } visit_b(X_k) < set(\alpha, p, j)_{i+1} \text{ in } V_p.$$

(2) There exists a tuple $(X_k, (a, b)) \in SI_i(\alpha, p, j)$, *for some* $k \in [1, n]$, *such that*

$$visit_a(X_k) < set(\alpha, p, j)_i \text{ and } set(\alpha, p, j)_{i+1} < visit_b(X_k) \text{ in } V_p.$$

(3) There exist a production $q : Y_0 \rightarrow Y_1 \cdots Y_m$, *and tuples* $(Y_k, (a, b))$, $(Y_l, (r, s)) \in SI_i(\alpha, p, j)$ *for some distinct* $k, l \in [1, m]$, *such that*

$$visit_a(Y_k) < visit_r(Y_l) \text{ and } visit_s(Y_l) < visit_b(Y_k) \text{ in } V_q. \qquad \square$$

The proofs of these lemmas are omitted since they are in fact natural generalizations of Lemmas 2 and 3. The remarks made about the allocation of a temporary single-use applied occurrence also apply here for the $(i, i+1)$-allocation of a multi-use applied occurrence (α, p, j) where $(\alpha, p, 1)_i$ and $(\alpha, p, 1)_{i+1}$ occur within the same sequence of V_p. Again, Lemmas 8 and 9 show that it is rational to consider global queues as well as global stacks for the $(i, i+1)$-allocation of a multi-use applied occurrence (α, p, j) with $(\alpha, p, 1)_i$ and $(\alpha, p, 1)_{i+1}$ in distinct sequences of V_p.

Example 6. Consider Examples 1 and 5. With the help of Lemmas 8, 9, and the analogue of Lemma 1, the allocations of Table 5 can be found for the instances of $(\alpha, p_1, 1)$, $(\alpha, p_2, 1)$, and $(\alpha, p_3, 1)$ during evaluation by E. $\square$

Applied Occurrence	(0,1)-Allocation	(1,2)-Allocation	(2,3)-Allocation
$(\alpha, p_1, 1)$	global variable		
$(\alpha, p_2, 1)$	global variable	global stack	global stack
$(\alpha, p_3, 1)$	global variable	global queue	

Table 5. Allocation of applied occurrences $(\alpha, p_1, 1)$, $(\alpha, p_2, 1)$, and $(\alpha, p_3, 1)$.

From the explanation at the beginning of this chapter, it should be clear that the $(i, i+1)$-allocation for $i > 0$ of the instances of a multi-use applied occurrence (α, p, j) during evaluation of a simple multi-visit evaluator E is based on reallocation. That is, after the computation of an instance of $(\alpha, p, j)_i$, evaluator E reallocates that instance of (α, p, j) which is used for this computation from D_i to D_{i+1}, where D_i and D_{i+1} denote the data structures to which the instances of (α, p, j) are $(i-1, i)$- and $(i, i+1)$-allocated respectively. This leads to a simple modification of the evaluator of which the details are presented in Example 8.

To appreciate the importance of the reallocations, the following results are presented. In general terms they show that, irrespective of the class of simple multi-visit evaluators used, there is no need to consider different data structures for the global storage allocation of multi-use applied occurrences than for the global storage allocation of single-use applied occurrences when using reallocations.

Theorem 5. *Let E be a simple multi-pass evaluator for an attribute grammar G with multi-use applied occurrence (α, p, j). All the attribute instances of (α, p, j) can be $(i, i+1)$-allocated to a basic linear data structure in $\mathfrak{D}$ during evaluation by E if and only if E can $(i, i+1)$-allocate them to a global stack or to a global queue.* $\square$

Theorem 6. *Let E be a simple multi-sweep evaluator for an attribute grammar G with multi-use applied occurrence (α, p, j). Assume $\mathcal{S} \subset \mathcal{D}$ to be a set consisting of the basic stack and all other data structures in $\mathcal{D}$ such that*

$$D, D' \in \mathcal{S} \text{ if and only if } f(i^n o^n) \neq f'(i^n o^n) \text{ for some } n \in \mathbb{N}^+$$

where f and f' denote the behaviour of D and D' respectively. Apart from variants ($\mathcal{S}$ is not unique), $\mathcal{S}$ is the smallest possible subset of $\mathcal{D}$ with the property that all the attribute instances of (α, p, j) can be $(i, i+1)$-allocated to a data structure in $\mathcal{D}$ during evaluation by E if and only if E can $(i, i+1)$-allocate them to a data structure in $\mathcal{S}$. □

Theorem 7. *Let E be a simple multi-visit evaluator for an attribute grammar G with multi-use applied occurrence (α, p, j). There is no subset $\mathcal{V} \subset \mathcal{D}$ with the following property: all the attribute instances of (α, p, j) can be $(i, i+1)$-allocated to a data structure in $\mathcal{D}$ during evaluation by E if and only if E can $(i, i+1)$-allocate them to a data structure in $\mathcal{V}$.* □

The principal idea in the proofs of these theorems is the construction of an attribute grammar G' with an evaluator E' such that E' is simple multi-X if E is simple multi-X, and such that E' can allocate the instances of a single-use applied occurrence $(\alpha, p, j)'$ to a $D \in \mathcal{D}$ if and only if E can $(i, i+1)$-allocate the instances of (α, p, j) to D. Once that construction is established, the proofs of Theorems 5, 6, and 7 are greatly simplified. All that is left to show is that the data structures claimed necessary for the $(i, i+1)$-allocation of the instances of (α, p, j) are the same as the data structures necessary for the global storage allocation of the instances of $(\alpha, p, j)'$. This follows, depending on the theorem to be proved, from Theorem 2, 3, or 4. Thus, each proof consists of three parts: (1) the construction of attribute

grammar G', (2) the construction of evaluator E', and (3) the proof of the, by construction, remaining claim.

Proof of Theorem 5. The construction of attribute grammar G' is as follows. Attribute grammar G' has grammar G'_u entirely similar to G_u, except for sets N and P. Set N has a new nonterminal Y, and set P has a new production $q : Y \to \epsilon$ as well as a modified production p. The way in which production $p : X_0 \to X_1 \cdots X_n$ is modified depends on integer i.

(1) *Integer* $i = 0$. — In this case p is modified by the following construction, where it is assumed that $\alpha \in A_m(X_j)$ in E. Supposing that the direction of the m-th pass is the same for both E' as E, it modifies p in such a way that, for any node x at which p is applied, the first nonterminal child of x which is visited by E' after the m-th visit to node x_j is the one labelled by Y.

```
if j = 0
then if direction d(m) = L in E
        then replace X_1 in p by Y X_1
        else  replace X_n in p by X_n Y
        fi
else  if direction d(m) = L in E
        then replace X_j in p by X_j Y
        else  replace X_j in p by Y X_j
        fi
fi
```

(2) *Integer* $i > 0$. — In this case p is modified by the construction below, where it is assumed that $(\alpha, p, j)_i = (\beta, p, k)$ and $\beta \in A_m(X_k)$ in E. Supposing that the direction of the m-th pass is the same for both E' as E, it modifies p in such a way that, for any node x at which p is applied, the last nonterminal child of x which is visited by E' before the m-th visit to node x_k is the one labelled by Y.

```
if k = 0
then if direction d(m) = L in E
     then replace X_n in p by X_n Y
     else replace X_1 in p by Y X_1
     fi
else if direction d(m) = L in E
     then replace X_k in p by Y X_k
     else replace X_k in p by X_k Y
     fi
fi
```

Attribute grammar G' has the sets of attributes for the nonterminals of G extended with set $A(Y)$ given as $\{\alpha', \alpha''\}$ where $V(\alpha')$ and $V(\alpha'')$ are taken as $V(\alpha)$. The sets of semantic rules of G' are the same as those of G with two exceptions. One is, of course, set $R(q)$ defined as $\{(\alpha'', q, 0) = (\alpha', q, 0)\}$, and the second is set $R(p)$. Let $X_0 \to X'_1 \cdots X'_{n+1}$ be the modified p with $X'_l = Y$ for some $l \in [1, n+1]$. Set $R(p)$ has semantic rule $(\alpha', p, l) = (\alpha, p, j)$ added to it, and (α, p, j) replaced by (α'', p, l) in the semantic rule defining $(\alpha, p, j)_{i+1}$.

Evaluator E' for attribute grammar G' is obtained from E by an extension step and a modification step. Assume that $(\alpha, p, j)_i = (\beta, p, k)$ and $\beta \in A_m(X_k)$ in E. The extension step ensures that, for any node y labelled Y, the instances α' and α'' at y are computed during the m-th visit of E' to y. It consists of the extension of E with integer $\phi(Y)$, partition $A_1(Y), \ldots, A_{\phi(Y)}(Y)$ of $A(Y)$, sequence $v_q = v_q(1) \cdots v_q(\phi(Y))$, and set $E(q)$. These are defined by the following construction.

set $\phi(Y)$ to $\phi(X_0)$;
for each $u \in [1, \phi(Y)]$
do if $u = m$ **then** set $A_u(Y)$ to $\{\, \alpha', \alpha'' \,\}$
else set $A_u(Y)$ to $\varnothing$
fi;
set $v_q(u)$ to ϵ
od;
set $E(q)$ to $\{$ **set** $(\alpha'', q, 0)$ **to** $(\alpha', q, 0)\, \}$

The modification step places set $E(p)$ in conformity with set $R(p)$, and sequence v_p in conformity with mapping d specifying the direction of the passes of E. The modification of $E(p)$ follows at once from $R(p)$, and the modification of $v_p = v_p(1) \cdots v_p(\phi(X_0))$ by the construction below.

for each $i \in [1, \phi(X_0)]$
do if direction $d(i) = L$ in E
then change $v_p(i)$ into $visit_i(X_1') \cdots visit_i(X_{n+1}')$

```
        else  change v_p(i) into visit_i(X'_{n+1}) ··· visit_i(X'_1)
        fi
    od;
```

Now, by construction, E' is a simple multi-pass evaluator with the property that the order in which the instances of (α'', p, l) are computed and used is exactly the same as the order in which E stores and removes the instances of (α, p, j) on some $D \in \mathcal{D}$ when they are $(i, i+1)$-allocated to D. Therefore, it is easily seen that the instances of (α, p, j) can be $(i, i+1)$-allocated to a $D \in \mathcal{D}$ during evaluation by E if and only if the instances of (α'', p, l) can be allocated to D during evaluation by E'.

So the theorem is proved if it can be shown that, during evaluation by E', the instances of (α'', p, l) can be allocated to a data structure in $\mathcal{D}$ if and only if they can be allocated to a global stack or a global queue. And this follows from Theorem 2 because E' is simple multi-pass. □

Proof of Theorem 6. The construction of attribute grammar G' is entirely similar to that one of the previous proof, except for production p. Here p is modified to $X_0 \rightarrow X_1 \cdots X_n Y$.

Evaluator E' for attribute grammar G' is obtained from E is the same way as E' in the proof of Theorem 5 with one difference, namely the modification of sequence $v_p = v_p(1) \cdots v_p(\phi(X_0))$. Two cases are distinguished depending on the value of integer i.

(1) *Integer* $i = 0$. — In this case the modification of sequence v_p is specified by the following construction. It modifies sequence v_p in such a way that, for any node x at which p is applied, node x_{n+1} with label Y is

the first nonterminal child of x which is visited by E' after x_j has been visited.

```
for each u ∈ [1, φ(X_0)]
do if j = 0
     then change v_p(u) to visit_u(Y) · v_p(u)
     else  change visit_u(X_j) in v_p(u) to visit_u(X_j) · visit_u(Y)
     fi
od
```

(2) *Integer* $i > 0$. — In this case the modification of v_p is specified by the following construction where it is assumed that $(\alpha, p, j)_i = (\beta, p, k)$ in E. It modifies sequence v_p such that, for any node x at which p is applied, node x_{n+1} with label Y is the last nonterminal child of x which is visited by E' before x_k is visited.

```
for each u ∈ [1, φ(X_0)]
do if k = 0
     then change v_p(u) to v_p(u) · visit_u(Y)
     else  change visit_u(X_j) in v_p(u) to visit_u(Y) · visit_u(X_j)
     fi
od
```

By construction, E' is a simple multi-sweep evaluator with the property that the order in which the instances of $(\alpha'', p, n+1)$ are computed and used is exactly the same as the order in which E stores and removes the instances

of (α, p, j) on some $D \in \mathcal{D}$ when they are $(i, i+1)$-allocated to D. That is, the instances of $(\alpha'', p, n+1)$ can be allocated to a $D \in \mathcal{D}$ during evaluation by E' if and only if the instances of (α, p, j) can be $(i, i+1)$-allocated to D during evaluation by E.

Thus, to complete the proof, it needs only be shown that, during evaluation by E', the instances of $(\alpha'', p, n+1)$ can be allocated to a data structure in $\mathcal{D}$ if and only if they can be allocated to a data structure in $\mathcal{S}$. And that follows from Theorem 3 (because E' is simple multi-sweep). □

Proof of Theorem 7. Attribute grammar G' is entirely similar to G' as presented in the proof of Theorem 6.

Evaluator E' for attribute grammar G' is again obtained from E by an extension step and a modification step. The extension step extends E with integer $\phi(Y) = 1$, partition $A_1(Y) = A(Y)$, sequence $v_q = v_q(1) = \epsilon$, and set

$$E(q) = \{\, \textbf{set } (\alpha'', q, 0) \textbf{ to } (\alpha', q, 0) \,\}.$$

The modification step consists of the modification of set $E(p)$ as in the proof of Theorem 5, and of the modification of sequence v_p. The modification of sequence $v_p = v_p(1) \cdots v_p(\phi(X_0))$ depends on integer i.

(1) *Integer $i = 0$.* — In this case the modification of v_p is specified by the following construction, where it is assumed that $\alpha \in A_m(X_j)$ in E. It modifies v_p in such a way that, for any node x at which p is applied, node x_{n+1} with label Y is the first nonterminal child of x which is visited by E' after the m-th visit to x_j.

if $j = 0$

```
then change v_p(m) to visit_1(Y) · v_p(m)
else  for each u ∈ [1, φ(X_0)]
      do if visit_m(X_j) occurs in v_p(u)
           then replace visit_m(X_j) by visit_m(X_j) · visit_1(Y)
           fi
      od
fi
```

(2) *Integer* $i > 0$. — In this case the modification of v_p is specified by the following construction, where it assumed that $(\alpha, p, j)_i = (\beta, p, k)$ and $\beta \in A_m(X_k)$ in E. It modifies v_p such that, for any node x at which p is applied, node x_{n+1} with label Y is the last nonterminal child of x which is visited by evaluator E' before the m-th visit to x_k.

```
if k = 0
then change v_p(m) to v_p(m) · visit_1(Y)
else  for each u ∈ [1, φ(X_0)]
      do if visit_m(X_k) occurs in v_p(u)
           then replace visit_m(X_k) by visit_1(Y) · visit_m(X_k)
           fi
      od
fi
```

It should be clear that E' is a simple multi-visit evaluator with the property that the order in which the instances of $(\alpha'', p, n+1)$ are computed and used is exactly the same as the order in which E stores and removes the instances

of (α, p, j) on some $D \in \mathcal{D}$ when they are $(i, i+1)$-allocated to D. Thus the instances of (α, p, j) can be $(i, i+1)$-allocated to a data structure $D \in \mathcal{D}$ during evaluation by E if and only if the instances of $(\alpha'', p, n+1)$ can be allocated to D during evaluation by E'.

In other words, all that is needed to complete the proof is the demonstration that, during evaluation by E', there is no $\mathcal{V} \subset \mathcal{D}$ with the property that the instances of $(\alpha'', p, n+1)$ can be allocated to a data structure in $\mathcal{D}$ if and only if they can be allocated to a data structure in $\mathcal{V}$. This again follows from Theorem 4. $\square$

Additional evidence of the importance of reallocations is the fact that without reallocations there are linear data structures required with more than one protocol for the (a, b)-allocation of multi-use applied occurrences, where $b > a + 1$. As applied occurrence $(\alpha, p_3, 1)$ will show in Example 7, it is not sufficient to simply extend the basic linear data structures with an access operation $a : \Sigma \rightarrow \Sigma \times \mathit{Data}$ given as $a(\langle\delta, M\rangle) = (\langle\delta, M\rangle, \partial)$ which, when called with current contents $\langle\delta, M\rangle$, invokes a program that delivers item ∂ if $(\partial, p(\delta)) \in M$, and aborts if M is empty. To $(0, 2)$-allocate the instances of $(\alpha, p_3, 1)$ to a linear data structure, the access operation a requires the basic stack protocol and the output operation o requires the basic queue protocol.

The disadvantage of the reallocations is that they increase evaluation time. For this reason, it is important to reduce the number of reallocations for a multi-use applied occurrence. With this aim the following lemmas are presented (of which Lemma 10 generalizes Lemma 8).

Lemma 10. *Let E be a simple multi-visit evaluator for an attribute grammar G and let $p : X_0 \rightarrow X_1 \cdots X_n \in P$ with $(\alpha, p, j) \in AO(p)$ multi-use. The attribute instances of (α, p, j) can be $(a, b+1)$-allocated to a global stack*

during evaluation by E if and only if none of the following conditions is satisfied.

(1) There exists a tuple $(X_k, (d, e))$ in $\bigcup_{i=a}^{b} SI_i(\alpha, p, j)$ for some $k \in [1, n]$ such that for some $i \in [a, b]$,

$$set(\alpha, p, j)_i < visit_d(X_k) < set(\alpha, p, j)_{i+1} < visit_e(X_k) \text{ in } V_p.$$

(2) There exists a tuple $(X_k, (d, e))$ in $\bigcup_{i=a}^{b} SI_i(\alpha, p, j)$ for some $k \in [1, n]$ such that for some $i \in [a, b]$,

$$visit_d(X_k) < set(\alpha, p, j)_i < visit_e(X_k) < set(\alpha, p, j)_{i+1} \text{ in } V_p.$$

(3) There exist a production $q : Y_0 \to Y_1 \cdots Y_m$, and tuples $(Y_k, (d, e))$ and $(Y_l, (r, s))$ in $\bigcup_{i=a}^{b} SI_i(\alpha, p, j)$ for some distinct $k, l \in [1, m]$ such that

$$visit_d(Y_k) < visit_r(Y_l) < visit_e(Y_k) < visit_s(Y_l) \text{ in } V_q.$$

Proof. The part of the proof showing the necessity of the conditions requires careful consideration; a good reason to present it here. The other part of the proof, however, follows the same line of reasoning as the related part of Lemma 11.

To show the necessity of the conditions (1)-(3), let t be a derivation tree with x and y denoting two of its nodes at which p is applied. Denote by γ the instance of (α, p, j) at node x_j and by δ the instance of (α, p, j) at node y_j where $\xi(\gamma) = \langle g_0, \ldots, g_u \rangle$ and $\xi(\delta) = \langle d_0, \ldots, d_u \rangle$. Assume that the instances of (α, p, j) cannot be $(a, b+1)$-allocated to a global stack during evaluation by E. Then, by definition, attribute grammar G must have a derivation tree such as t where $g_a < d_a < g_i < d_{b+1}$ for some $i \in [a+1, b+1]$.

Hence there exists a $w \in [a,b]$ such that $g_a < d_w < g_i < d_{w+1}$, and so there exists a $v \in [a,b]$ such that $g_v < d_w < g_{v+1} < d_{w+1}$. Now three cases are distinguished.

(a) *Node y is a descendant of node x_k or x_k itself for some $k \in [1,n]$.* — Then there must be a tuple $(X_k,(d,e))$ in set $SI_w(\alpha,p,j)$ such that $\mathit{set}(\alpha,p,j)_v < \mathit{visit}_d(X_k) < \mathit{set}(\alpha,p,j)_{v+1} < \mathit{visit}_e(X_k)$ in V_p. This implies that condition (1) is satisfied.

(b) *Node x is a descendant of node y_k or y_k itself for some $k \in [1,n]$.* — Then there must be a tuple $(X_k,(d,e))$ in set $SI_v(\alpha,p,j)$ such that $\mathit{visit}_d(X_k) < \mathit{set}(\alpha,p,j)_w < \mathit{visit}_e(X_k) < \mathit{set}(\alpha,p,j)_{w+1}$ in V_p. This implies that condition (2) is satisfied.

(c) *Nodes x and y are incomparable.* — Let z be the root of the smallest tree containing both x and y, and let $q : Y_0 \to Y_1 \cdots Y_m$ be the production applied at z. Assume that x is a descendant of node z_l or z_l itself and y is a descendant of node z_k or z_k itself, for distinct $k,l \in [1,m]$. Then there are two tuples $(Y_k,(d,e)),(Y_l,(r,s)) \in \bigcup_{i=a}^{b} SI_i(\alpha,p,j)$ such that $\mathit{visit}_d(Y_k) < \mathit{visit}_r(Y_l) < \mathit{visit}_e(Y_k) < \mathit{visit}_s(Y_l)$ in V_q. That implies that condition (3) is satisfied. □

Lemma 11. *Let E be a simple multi-visit evaluator for an attribute grammar G and let $p : X_0 \to X_1 \cdots X_n \in P$ with $(\alpha,p,j) \in AO(p)$ multi-use. The attribute instances of (α,p,j) can be $(a,b+1)$-allocated to a global queue during evaluation by E if and only if E can (a,b)-allocate them to a global queue and none of the following conditions is satisfied.*

(1) There exists a tuple $(X_k,(d,e))$ in $SI_b(\alpha,p,j)$ for some $k \in [1,n]$ such that

$$visit_d(X_k) < set(\alpha, p, j)_{a+1} < visit_e(X_k) \ in \ V_p.$$

(2) There exists a tuple $(X_k, (d, e))$ *in* $SI_a(\alpha, p, j)$ *for some* $k \in [1, n]$ *such that*

$$set(\alpha, p, j)_b < visit_e(X_k) < set(\alpha, p, j)_{b+1} \ in \ V_p.$$

(3) There exist a production $q : Y_0 \to Y_1 \cdots Y_m$ *and tuples* $(Y_k, (d, e))$ *in* $SI_a(\alpha, p, j)$ *and* $(Y_l, (r, s))$ *in* $SI_b(\alpha, p, j)$ *for some distinct* $k, l \in [1, m]$ *such that*

$$visit_r(Y_l) < visit_e(Y_k) < visit_s(Y_l) \ in \ V_q.$$

Proof. Let t be a derivation tree with x and y denoting two of its nodes at which production p is applied. The instance of (α, p, j) at node x_j is denoted by γ and the instance of (α, p, j) at node y_j is denoted by δ where $\xi(\gamma) = \langle g_0, \ldots g_u \rangle$ and $\xi(\delta) = \langle d_0, \ldots d_u \rangle$.

Clearly the fact that during evaluation by E the instances of (α, p, j) can be $(a, b+1)$-allocated to a global queue implies that they can be (a, b)-allocated to a global queue. So essentially two things have to be proved, namely that:

(1) One of the conditions (1)-(3) is satisfied if during evaluation by E the instances of (α, p, j) can be (a, b)-allocated to a global queue but not $(a, b+1)$-allocated to a global queue.

(2) The instances of (α, p, j) cannot be $(a, b+1)$-allocated to a global queue during evaluation by E if one of the conditions (1)-(3) is satisfied.

Proof of assertion (1).— Suppose that during evaluation by E the instances of (α, p, j) can be (a, b)-allocated to a global queue but not $(a, b+1)$-

allocated to a global queue. Then, by definition, G must have a derivation tree such as t where $g_b < d_{a+1} < g_{b+1}$. Now three cases are distinguished.

(a) *Node y is a descendant of node x_k or x_k itself for some $k \in [1, n]$.*— Then it follows that there is a tuple $(X_k, (d, e)) \in SI_a(\alpha, p, j)$ such that $set(\alpha, p, j)_b < visit_e(X_k) < set(\alpha, p, j)_{b+1}$ in V_p. Hence condition (2) is satisfied.

(b) *Node x is a descendant of node y_k or y_k itself for some $k \in [1, n]$.*— In this case there must be a tuple $(X_k, (d, e)) \in SI_b(\alpha, p, j)$ such that $visit_d(X_k) < set(\alpha, p, j)_{a+1} < visit_e(X_k)$ in V_p. Hence condition (1) is satisfied.

(c) *Nodes x and y are incomparable.*— Let z be the root of the smallest tree containing both x and y, and let $q : Y_0 \to Y_1 \cdots Y_m$ be the production applied at z. Assume that x is a descendant of node z_l or z_l itself and y is a descendant of node z_k or z_k itself for $k, l \in [1, m]$. (Notice that $k \neq l$.) Then there is a tuple $(Y_k, (d, e)) \in SI_a(\alpha, p, j)$ and a tuple $(Y_l, (r, s)) \in SI_b(\alpha, p, j)$ such that condition (3) is satisfied.

Proof of assertion (2).— Assume that one of the conditions (1)-(3) is satisfied. Recalling that G_u is reduced, each condition will be considered separately.

(a) *Condition (1) is satisfied.*— Then G has a derivation tree such as t (where y is a descendant of node x_k or is x_k itself) which is evaluated by evaluator E in such a way that inequality $d_b < g_{a+1} < d_{b+1}$ holds. Now if $d_a < g_a$ then $d_a < g_a < g_{a+1} < d_{b+1}$, and if $g_a < d_a$ then $g_a < d_a < d_{a+1} \leq d_b < g_{a+1} < g_{b+1}$ and so $g_a < d_a < d_{a+1} < g_{b+1}$.

In either case it follows that, by definition, the instances of (α, p, j) cannot be $(a, b+1)$-allocated to a global queue during evaluation by E.

(b) *Condition (2) is satisfied.*— Then G has a tree such as t (where y is a descendant of node x_k or is x_k itself) which is evaluated by E in such a way that inequality $g_b < d_{a+1} < g_{b+1}$ holds. In the same way as in the previous case, it follows that the instances of (α, p, j) cannot be $(a, b+1)$-allocated to a global queue during evaluation by E.

(c) *Condition (3) is satisfied.*— Let z be a node of t at which production q is applied. Then G has a tree such as t (where x is a descendant of z_l or is z_l itself, and y is a descendant of node z_k or is z_k itself) which is evaluated by E in such a way that $g_b < d_{a+1} < g_{b+1}$ holds. Similar to the previous cases, it follows that the instances of (α, p, j) cannot be $(a, b+1)$-allocated to a global queue during evaluation by E. □

Remark. It is also possible to generalize the definition of subtree information to cope with the (a, b)-allocation instead of the $(a, a+1)$-allocation as shown in [10] and [33]. In so doing, lemmas similar to Lemmas 8 and 9 can be stated for the (a, b)-allocation. The reason for presenting Lemmas 10 and 11 in the way chosen here, is to show that it is sufficient to compute only the sets $SI_i(\alpha, p, j)$ of an applied occurrence (α, p, j) and still be able to decide all possible (a, b)-allocations of (α, p, j) to global stacks, queues, and variables. Moreover, this way avoids the need to introduce new definitions. □

Lemma 10 shows that a multi-use applied occurrence (α, p, j) can be (a, b)-allocated to a global stack if $(\alpha, p, j)_a$ and $(\alpha, p, j)_b$ occur within the same sequence of V_p. Thus, a temporary multi-use applied occurrence can

always be allocated to a stack without reallocations. For a nontemporary multi-use applied occurrence (α, p, j), the number of reallocations can be reduced to $n - m$ in the worst case, where (α, p, j) occurs in $V_p(m)$, and the occurrence which uses (α, p, j) for the last time occurs in $V_p(n)$.

Example 7. Consider Example 6. From Lemmas 10 and 11 it is inferred that during evaluation by E all the attribute instances of $(\alpha, p_2, 1)$ can be $(0, 2)$-allocated to a global stack. The reallocations for $(\alpha, p_3, 1)$ cannot be reduced, as $(\alpha, p_3, 1)$ cannot be $(0, 2)$-allocated to a global stack or a global queue. □

The following theorem captures a rather immediate but important consequence of the presented lemmas.

Theorem 8. *Let E be a simple multi-visit evaluator for an attribute grammar G and (α, p, j) a multi-use applied occurrence of G. It is decidable in polynomial time whether the instances of (α, p, j) can be (a, b)-allocated to a global variable, stack, or queue during evaluation by E.*

Proof. Observing that the maximum number of tuples in a set $SI_i(\alpha, p, j)$ is limited to $|N| \times M^2$, where $M = \max \{ \phi(X) \mid X \in N \}$, the proof is readily obtained by Lemmas 7, 9, 10, 11, and the analogue of Lemma 1. □

Remark. In [10], Engelfriet and De Jong present necessary and sufficient conditions to decide, for a given simple multi-visit evaluator E of an attribute grammar G and a given attribute α of G, whether the instances of α can be allocated to a global variable or a global stack. It should be clear that, during evaluation by E, the instances of any applied occurrence of α can be allocated to a global variable (or a global stack) if the instances of α can

be allocated to a global variable (or a global stack). The converse, however, does not hold in general.

Sometimes a less restricted notion of stack is used. It is shown in [20] that every temporary attribute α can be allocated to a stack during evaluation by some simple multi-visit evaluator E if E is allowed to read a fixed number of elements below the top. Since each applied occurrence of α can be allocated to a conventional stack during evaluation by E, there is no need for this kind of stack. □

CHAPTER FIVE

Implementation Aspects

This chapter explains how the global storage allocation of applied occurrences can be implemented in a simple multi-visit evaluator so that it satisfies the demands (1)-(3) listed below.

(1) The evaluation strategy is left unchanged.

(2) The structure remains independent of specific derivation trees.

(3) The applied occurrences that are not used are handled without additional effort.

It should be emphasized that the proposal described in this chapter is not the only possible implementation. The reader is challenged to find other, perhaps more efficient, implementations.

To begin, the notion of coupled attribute occurrences is introduced. Two attribute occurrences (α, p, j) and (α, q, k) are said to be coupled with respect to nonterminal X if $X = X_j = Y_k$, where

$$p : X_0 \to X_1 \cdots X_n \qquad \text{and} \qquad q : Y_0 \to Y_1 \cdots Y_m.$$

All the occurrences of attribute α which are coupled with respect to X can be partitioned into two disjoint sets of coupled defined occurrences and coupled applied occurrences denoted by $CDO(\alpha, X)$ and $CAO(\alpha, X)$, respectively. The set $CDO(\alpha, X)$ is defined as

$$\{ (\alpha, p, j) \mid p : X_0 \to X_1 \cdots X_n \in P,\ (\alpha, p, j) \in DO(p),\ X_j = X \},$$

and the set $CAO(\alpha, X)$ as

$$\{ (\alpha, p, j) \mid p : X_0 \to X_1 \cdots X_n \in P,\ (\alpha, p, j) \in AO(p),\ X_j = X \}.$$

If an applied occurrence $(\alpha, p, j) \in CAO(\alpha, X)$ is not coupled with another applied occurrence, that is $CAO(\alpha, X)$ is the singleton set $\{ (\alpha, p, j) \}$, then its allocation can be performed immediately. To see this, let E be a simple multi-visit evaluator and $\alpha(m)$ the instance of (α, p, j) connected to node m in some derivation tree. Suppose that E is visiting node n and bound to visit node m after the computation of instance $\alpha(m)$. Since the production applied at node n is known, E knows of which defined occurrence in $CDO(\alpha, X)$ attribute $\alpha(m)$ is an instance. Further it knows that $\alpha(m)$ can only be an instance of applied occurrence (α, p, j). Therefore E can immediately allocate the value of $\alpha(m)$ to the appropriate data structure for (α, p, j) once this instance is computed; that is, before actually visiting node m. It should be evident how this can be implemented for all the instances of (α, p, j), in such a way that demands (1)-(3) are satisfied.

If an applied occurrence $(\alpha, p, j) \in CAO(\alpha, X)$ is coupled with other applied occurrences, then the immediate allocation of the value of $\alpha(m)$ by evaluator E, as described above, is no longer possible. For this allocation to be made requires that E knows which production is applied at node m

before it visits this node. So here is a problem: evaluator E must postpone the allocation of the value of $\alpha(m)$ until it actually visits node m. One way to do this is by using an uncoupling variable (α, X) for all the occurrences of attribute α that are coupled with respect to X. This uncoupling variable is introduced in evaluator E in the following manner (recall Definition 2).

for every $(\alpha, q, k) \in CDO\,(\alpha, X)$
do replace **set** (α, q, k) **to** $f((\alpha_1, q, k_1), \ldots, (\alpha_m, q, k_m)) \in E(q)$
by **set** (α, X) **to** $f((\alpha_1, q, k_1), \ldots, (\alpha_m, q, k_m))$
od;

for every $(\alpha, p, j) \in CAO\,(\alpha, X)$
do add semantic rule **set** (α, p, j) **to** (α, X) to $E(p)$
od;

Instead of computing instance $\alpha(m)$ while visiting node n, E now computes the value of uncoupling variable (α, X). On visiting node m, E immediately allocates the value of (α, X) to the appropriate data structure for the applied occurrence in $CAO\,(\alpha, X)$ of which $\alpha(m)$ is an instance (that is applied occurrence (α, p, j) in this particular case). By now it should be clear how this can be implemented for all the instances of each applied occurrence in $CAO\,(\alpha, X)$. The reader is invited to verify that this approach using uncoupling variables satisfies demands (1)-(3).

Notice that the last reference to an uncoupling variable (α, X) is always made before the next uncoupling variable of α is computed (recalling that sequential evaluation is assumed). This property implies that all the uncou-

pling variables of attribute α, that is all the uncoupling variables collected in set

$$\{ (\alpha, X) \mid \alpha \in A(X),\ X \in N \},$$

can share the same memory. This is quite acceptable, particularly if cache memory is used for this heavily utilized part of the memory.

Example 8. This example demonstrates the proposed implementation in a program which implements a simple multi-pass evaluator. It also demonstrates the occurrence of queues for global storage allocation in attribute evaluators of attribute grammars with nonlinear underlying context-free grammars.

Let G be an attribute grammar with nonterminals $N = \{ S,\ A \}$, terminals $\Sigma = \varnothing$, start symbol S, and productions

$$P = \{ p_1 : S \rightarrow A,\ p_2 : A \rightarrow \epsilon,\ p_3 : A_0 \rightarrow A_1 A_2 \}.$$

The sets of attributes of each nonterminal are:

$$\begin{aligned} I(S) &= \varnothing & \qquad I(A) &= \{ i,\ \alpha \} \\ S(S) &= \{ s \} & \qquad S(A) &= \{ s_1,\ s_2 \} \end{aligned}$$

The semantic functions defining the defined occurrences

$$(\beta, p, k) \in \textstyle\bigcup_{p \in P} DO(p)$$

are named $f_{(\beta,p,k)}$. The dependencies among the attribute occurrences are given by the dependency graphs in Figure 10.

Attribute grammar G is evaluated by a simple multi-pass evaluator E whose behaviour is specified by the extended visit sequences of Table 6.

Production Rule	Extended Visit-Sequence
p_1	$\langle \varnothing, \{ (\alpha, p_1, 1) \}, visit_1(A), \{ (s_1, p_1, 1) \}, \varnothing \rangle$ $\langle \varnothing, \{ (i, p_1, 1) \}, visit_2(A), \{ (s_2, p_1, 1) \}, \{ (s, p_1, 0) \} \rangle$
p_2	$\langle \{ (\alpha, p_2, 0) \}, \{ (s_1, p_2, 0) \} \rangle \langle \{ (i, p_2, 0) \}, \{ (s_2, p_2, 0) \} \rangle$
p_3	$\langle \{ (\alpha, p_3, 0) \}, \{ (\alpha, p_3, 1) \}, visit_1(A_1), \{ (s_1, p_3, 1) \}, \{ (\alpha, p_3, 2) \}, visit_1(A_2), \{ (s_1, p_3, 2) \}, \{ (s_1, p_3, 0) \} \rangle$ $\langle \{ (i, p_3, 0) \}, \{ (i, p_3, 1) \}, visit_2(A_1), \{ (s_2, p_3, 1) \}, \{ (i, p_3, 2) \}, visit_2(A_2), \{ (s_2, p_3, 2) \}, \{ (s_2, p_3, 0) \} \rangle$

Table 6. Extended visit-sequences of simple multi-pass evaluator E.

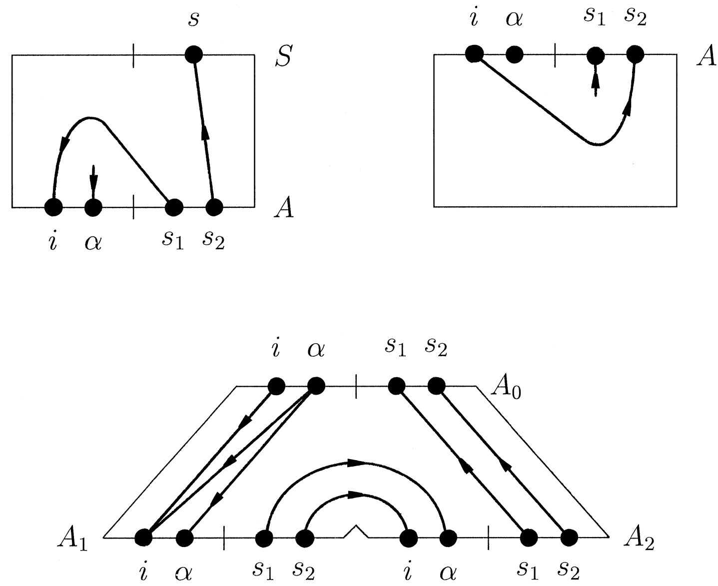

Figure 10. Dependency graphs $DG(p_1)$, $DG(p_2)$, and $DG(p_3)$.

The following program implements evaluator E. In this program, the values of all attribute instances are assumed to reside at their nodes in order to simplify the presentation.

procedure S-visit (n : node)
  case production applied at n **of**
    p_1 **:** set $\alpha(n_1)$ to $f_{(\alpha,p_1,1)}$;
      call A-visit$_1$ (n_1);
      set $i(n_1)$ to $f_{(i,p_1,1)}(s_1(n_1))$;
      call A-visit$_2$ (n_1);
      set $s(n)$ to $f_{(s,p_1,0)}(s_2(n_1))$
  esac;

```
procedure A-visit_1 (n : node)
   case production applied at n of
        p_2 : set s_1(n) to f_(s_1,p_2,0)
        p_3 : set α(n_1) to f_(α,p_3,1)(α(n));
              call A-visit_1 (n_1);
              set α(n_2) to f_(α,p_3,2)(s_1(n_1));
              call A-visit_1 (n_2);
              set s_1(n) to f_(s_1,p_3,0)(s_1(n_2))
   esac;

procedure A-visit_2 (n : node)
   case production applied at n of
        p_2 : set s_2(n) to f_(s_2,p_2,0)(i(n))
        p_3 : set i(n_1) to f_(i,p_3,1)(α(n), i(n));
              call A-visit_2 (n_1);
              set i(n_2) to f_(i,p_3,2)(s_2(n_1));
              call A-visit_2 (n_2);
              set s_2(n) to f_(s_2,p_3,0)(s_2(n_2))
   esac;
```

For this implementation of E, evaluation of a derivation tree t starts with execution of procedure call statement

call S-visit (r),

where r is the root of t.

In the following segments, (automatic) changes are presented which must be made in the program so that the instances of $(\alpha, p_3, 0)$ are $(0, 1)$-allocated

to a global variable v_1 and $(1, 2)$-allocated to a global queue q_1. (Verify that the reallocation for $(\alpha, p_3, 0)$ cannot be omitted by using stacks and queues which have the possibility to access a stored value more than once. According to Lemmas 10 and 11 the instances of $(\alpha, p_3, 0)$ cannot be $(0, 2)$-allocated to a global stack or queue.)

As explained, the coupled applied occurrences $(\alpha, p_2, 0)$ and $(\alpha, p_3, 0)$ need to be uncoupled. This is achieved by the introduction of the uncoupling variable (α, A) in the procedures S-visit and A-visit$_1$ below. A box is placed around each statement that has been changed.

procedure S-visit (n : node)
  case production applied at n **of**
    p_1 **:** [set (α, A) to $f_{(\alpha,p_1,1)}$;]
      call A-visit$_1$ (n_1);
      set $i(n_1)$ to $f_{(i,p_1,1)}(s_1(n_1))$;
      call A-visit$_2$ (n_1);
      set $s(n)$ to $f_{(s,p_1,0)}(s_2(n_1))$
  esac;

procedure A-visit$_1$ (n : node)
  case production applied at n **of**
    p_2 **:** [set $\alpha(n)$ to (α, A);]
      set $s_1(n)$ to $f_{(s_1,p_2,0)}$
    p_3 **:** [set $\alpha(n)$ to (α, A);]
      [set (α, A) to $f_{(\alpha,p_3,1)}(\alpha(n))$;]
      call A-visit$_1$ (n_1);
      [set (α, A) to $f_{(\alpha,p_3,2)}(s_1(n_1))$;]

call A-visit$_1$ (n_2);

set $s_1(n)$ to $f_{(s_1,p_3,0)}(s_1(n_2))$

esac;

Now that applied occurrence $(\alpha, p_3, 0)$ is uncoupled from $(\alpha, p_2, 0)$, the global storage allocation of applied occurrence $(\alpha, p_3, 0)$ can be implemented in the procedures. This is carried out in two stages. Firstly, the notation for queue operations is given, followed by the (automatic) changes that have to be made in the procedures using this notation.

Assuming that insertion takes place at the rear end of a queue and that access and deletion takes place at the front end of a queue, the notation for queue operations is given by:

(1) *append* (v, q). — Puts the value v at the rear end of queue q.

(2) *front* (q). — Returns the value at the front end of queue q.

(3) *decrease* (q). — Deletes the value at the front end of queue q.

Note that the operations *front* (q) and *decrease* (q) are not defined when there is no value at the front of queue q.

With the notation for queue operations established, the $(0, 1)$-allocation of the instances of $(\alpha, p_3, 0)$ to v_1 and $(1, 2)$-allocation to q_1 is obtained by altering procedures A-visit$_1$ and A-visit$_2$ as follows.

procedure A-visit$_1$ $(n$: node$)$

case production applied at n **of**

p_2 **:** set $s_1(n)$ to $f_{(s_1,p_2,0)}$

```
            p3 : [set v1 to (α, A);]
                 [set (α, A) to f(α,p3,1)(v1);]
                 [append (v1, q1);]
                 call A-visit1 (n1);
                 set (α, A) to f(α,p3,2)(s1(n1));
                 call A-visit1 (n2);
                 set s1(n) to f(s1,p3,0)(s1(n2))
      esac;


   procedure A-visit2 (n : node)
      case production applied at n of
            p2 : set s2(n) to f(s2,p2,0)(i(n))
            p3 : [set i(n1) to f(i,p3,1)(front (q1), i(n));]
                 [decrease (q1);]
                 call A-visit2 (n1);
                 set i(n2) to f(i,p3,2)(s2(n1));
                 call A-visit2 (n2);
                 set s2(n) to f(s2,p3,0)(s2(n2))
      esac;
```

Note that in the final segments the attribute instances of $(\alpha, p_2, 0)$ are not stored. These instances, on which no other attribute instances depend, are assumed not to be delivered as output.

Note further that the reallocation of the attribute instances of $(\alpha, p_3, 0)$ from variable v_1 to queue q_1 could have been avoided with a less strict notion of queue. All that it needs to $(0, 2)$-allocate the instances of $(\alpha, p_3, 0)$ to a

queue is the additional operation $rear(q)$ which returns the value at the rear end of the queue q. $\square$

These modifications conclude the explanation of how the global storage allocation of applied occurrences can be implemented in simple multi-visit evaluators. It should be clear that when the instances of an applied occurrence $(\alpha, p, j) \in CAO(\alpha, X)$ are $(0,1)$-allocated to a global stack, the coupled applied occurrences in $CAO(\alpha, X)$ can also be uncoupled using that stack instead of an uncoupling variable (α, X). Such an implementation can be beneficial since for each attribute instance of (α, p, j) it avoids allocating its value from uncoupling variable (α, X) to the global stack.

CHAPTER SIX

Conclusions

Global storage allocation has been examined for the instances of applied occurrences in simple multi-visit evaluators. The main results of this examination are summarized in (1)-(5) below.

(1) *Result of Theorem 5.* — For simple multi-pass evaluators, it is sufficient to consider global stacks and queues for the $(a, a+1)$-allocation of the instances of applied occurrences.

(2) *Result of Theorem 6.* — For simple multi-sweep evaluators, it is sufficient to consider the proper subset $\mathscr{S}$ of $\mathscr{D}$ for the $(a, a+1)$-allocation of the instances of applied occurrences.

(3) *Result of Theorem 7.* — For simple multi-visit evaluators, the full set of basic linear data structures $\mathscr{D}$ is needed for the $(a, a+1)$-allocation of the instances of applied occurrences.

(4) *Result of Theorem 8.* — For simple multi-visit evaluators, it is decidable in polynomial time whether the instances of an applied occurrence can be (a, b)-allocated to a global variable, stack, or queue.

(5) *Corollary of Lemma 10.* — For simple multi-visit evaluators, it is sufficient to consider global stacks for the (a, b)-allocation of the instances of an applied occurrence (α, p, j), where $(\alpha, p, j)_a$ and $(\alpha, p, j)_b$ occur within the same subsequence of V_p.

Since it is not possible to use more than one instance of a particular applied occurrence for the computation of an instance of a defined occurrence, results (1)-(5) do not depend on the assumption that evaluation functions f evaluate the actual parameters $x_1, \ldots, x_k$ of a function call $f(x_1, \ldots, x_k)$ in a sequential order. Hence, results (1)-(5) remain valid when evaluation functions are used which evaluate the actual parameters of a function call concurrently.

Figure 11 shows an inclusion diagram. This diagram has an ascending edge from x to y when the class of attribute evaluators x is a proper subset of the class of attribute evaluators y. The data structures needed for the $(a, a+1)$-allocation of the instances of an arbitrary applied occurrence are shown next to each class of attribute evaluators.

As indicated in Figure 11, the set of data structures needed for the $(a, a+1)$-allocation of the instances of an arbitrary applied occurrence increases in size with the generality of the class of attribute evaluators. (Note that for simple 1-pass, 1-sweep, and 1-visit evaluators it is sufficient to consider global stacks for the (a, b)-allocation of the instances of applied occurrences. This fact is an immediate consequence of result (5).)

The global storage allocation of the instances of applied occurrences has been demonstrated on representative attribute grammars. This technique has not yet been implemented so that real empirical data proving its usefulness on practical attribute grammars cannot be given. However, it is possible

to make some educated guesses using the empirical data of others (recall the definitions of temporary attributes and temporary applied occurrences on page 35).

Empirical data found in [12,15,18] shows that approximately 85% of the attributes in simple multi-visit evaluators are temporary. It also shows that approximately 60% of these temporary attributes, which can always be allocated to the global stacks of [20], can be allocated to global variables. (Figures for the allocation of nontemporary attributes are not available because [12,15,18] did not present such data.)

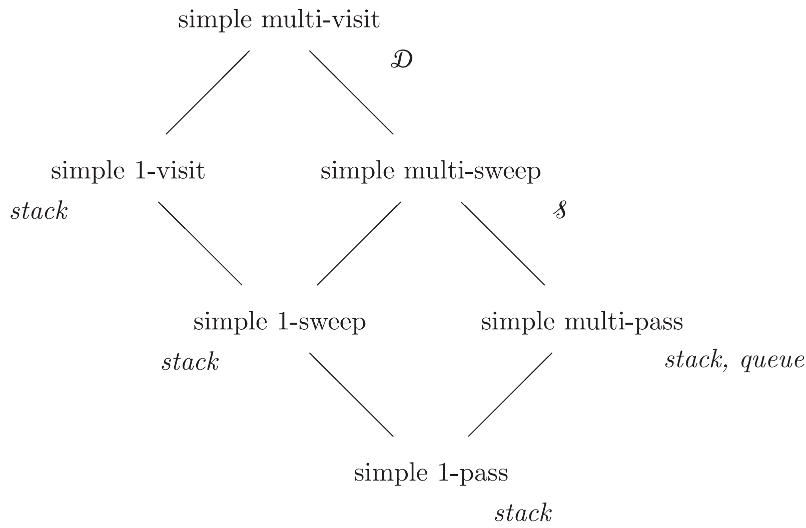

Figure 11. Inclusion diagram.

Now one of the virtues of attribute grammars is that they abstract from machine detail like evaluation order and storage allocation. As a result, there is no essential difference in the way in which the applied occurrences

of temporary and nontemporary attributes are treated while writing an attribute grammar. In other words, it is conceivable that 85% of the applied occurrences of nontemporary attributes are temporary, and that 60% of this quantity can be allocated to global variables (instead of global stacks). Similarly, it is conceivable that 60% of the (temporary) applied occurrences of temporary attributes that could not be allocated to global variables can be allocated to global variables.

Suppose for the sake of argument (empirical data is not available) that the applied occurrences are homogeneously distributed over the attributes. Then it can be calculated that 98% of the applied occurrences might be temporary, and that 92% of this quantity can be allocated to global variables (instead of global stacks).

I. CALCULATION OF THE PERCENTAGE OF TEMPORARY APPLIED OCCURRENCES WITH RESPECT TO THE TOTAL NUMBER OF APPLIED OCCURRENCES.

Contribution expected from temporary attributes:	100% of 85% = 85%
Contribution expected from nontemporary attributes:	85% of 15% $\approx$ 13%
Total:	98%

II. CALCULATION OF THE PERCENTAGE OF TEMPORARY APPLIED OCCURRENCES ALLOCATED TO GLOBAL VARIABLES WITH RESPECT TO THE TOTAL NUMBER OF TEMPORARY APPLIED OCCURRENCES.

Contribution expected from temporary attributes	
- *allocated to global variables:*	100% of 60% = 60%
- *allocated to global stacks:*	60% of 40% = 24%
Contribution expected from nontemporary attributes:	60% of 13% $\approx$ 8%
Total:	92%

These figures clearly indicate that the approach to globalize the instances of applied occurrences has significant advantages. Compared to those ap-

proaches that globalize the instances of attributes the percentage of temporary applied occurrences globalized in simple multi-visit evaluators might increase from 85 to 98, of which more than 90% (instead of 60%) might be allocated to global variables.

A word of caution is in place though. First of all, the empirical data found in [12,15] is based on attribute grammars that were used to define imperative programming languages, and the empirical data found in [18] is based on attribute grammars that were defined to bootstrap the FNC-2 attribute grammar system. It is likely that figures drop when attribute grammars are used for other purposes. Secondly, it is also possible that the figures are too optimistic because the specific assumptions made for this calculation do not hold (the assumption that the applied occurrences are homogeneously distributed over the attributes looks especially suspicious). Either way, in those cases the globalization of applied occurrences via global variables, stacks and queues with possible reallocations gains more importance.

Although some questions concerning the use of storage allocation in attribute evaluators have been answered in this work, other questions remain and some new questions have arisen. Two particularly interesting questions which have arisen are:

(1) Is there a proper subset $\mathcal{W}$ of $\mathcal{S}$ such that, with respect to the $(a, a+1)$-allocation of the instances of an applied occurrence, the use of data structures in $\mathcal{W}$ would be sufficient for simple multi-waddle evaluators?

(2) Is it decidable for a given simple multi-waddle evaluator whether the instances of an applied occurrence can be $(a, a+1)$-allocated to some data structure in $\mathcal{D}$ if it is not possible to $(a, a+1)$-allocate them to a global stack or queue?

The answer to question (2) is still unknown. Question (1), on the other hand, can be partially answered. By generating all combinations of derivation trees with four instances of a single-use applied occurrence together with all possible ways in which a simple multi-waddle evaluator can evaluate them, it has been found that the define-and-use sequences

$$1 \cdot 2 \cdot 3 \cdot 4 \cdot 2 \cdot 1 \cdot 4 \cdot 3, \qquad 1 \cdot 2 \cdot 3 \cdot 4 \cdot 2 \cdot 4 \cdot 1 \cdot 3,$$

$$1 \cdot 2 \cdot 3 \cdot 4 \cdot 3 \cdot 1 \cdot 4 \cdot 2, \quad \text{and} \quad 1 \cdot 2 \cdot 3 \cdot 4 \cdot 3 \cdot 4 \cdot 1 \cdot 2$$

never occur. In other words, there is such a proper subset $\mathcal{W}$ of $\mathcal{S}$, but it is not known how to characterize this set $\mathcal{W}$ precisely. (Note that a precise characterization of $\mathcal{W}$ could provide strong evidence concerning the question whether or not it is decidable in polynomial time that an attribute grammar is simple m-waddle for a fixed m. This idea is based on the observation that, for any $X \in \{\text{pass, sweep, visit}\}$, the complexity of deciding that an attribute grammar is simple m-X for a fixed m is polynomial if and only if for simple m-X evaluators a finite set of data structures is needed for the $(a, a+1)$-allocation of the instances of an applied occurrence. For further information, compare Figure 11 and reference [9].)

Lastly, another direction for future research must be mentioned. Using basic linear data structures, the theoretical possibilities and limitations of storage allocation in attribute evaluators have been explored. Results obtained during this investigation indicate that it would also be interesting to extend the scope of this work by making use of these data structures to explore the theoretical possibilities and limitations of storage allocation in domains other than attribute grammars.

POSTSCRIPT

Stacks and Queues for Absolutely Non-circular Attribute Evaluators

In the introduction of this dissertation it was stated that improving the results of Saarinen [32] required a more thorough analysis at evaluator construction time. This was considered to be rather difficult because the order in which absolutely non-circular attribute evaluators compute attribute instances, determined at evaluator construction time, depended on the derivation trees to be evaluated. However, advanced understanding of absolutely non-circular attribute evaluators shows that this dependency does not add that much complexity.

This postscript sets out to establish necessary and sufficient conditions to decide for an absolutely non-circular attribute evaluators whether the instances of an applied occurrence can be (a, b)-allocated to a global variable, stack, or queue. Definitions of new concepts and notations are kept to a minimum by adapting them from simple multi-visit evaluators as much as possible. Again a representative example is given to illustrate the concepts

and notations introduced. Implementation details, however, are omitted because they are similar to those for simple multi-visit evaluators.

Absolutely non-circular attribute evaluators

An absolutely non-circular attribute evaluator for an attribute grammar G is an attribute evaluator where the evaluation strategy is completely determined by a family $F(X)$ of totally ordered partitions over the attributes $A(X)$ of each $X \in N$. It is a program that selects for each node n labelled by X a partition $A_1(X), \ldots, A_m(X)$ of the family $F(X)$ so that, for each $i \in [1, m]$, all attribute instances of $A_i(X)$ at n are computed during the i-th visit to the subtree of n.

Let $\pi \in F(X)$ denote that π is a partition of $F(X)$, let $|\pi|$ denote the number of parts in partition π, and let $visit_i(X, \pi)$ denote the i-th visit to a node labelled $X \in N$ with selected partition $\pi \in F(X)$.

Definition 14. An *absolutely non-circular attribute evaluator E for an attribute grammar G* consists of components (1)-(3) where:

(1) For each $X \in N$, a family $F(X)$ of ordered partitions over $A(X)$. Each partition $\pi \in F(X)$ is necessary, that is, there must be at least one derivation tree in which π is selected for a node labelled by X. Between the attributes of each set $A_i(X)$ of a partition $A_1(X), \ldots, A_m(X)$ in $F(X)$ a particular computation order is assumed.

(2) For each $p : X_0 \to X_1 \cdots X_n \in P$, each $\pi \in F(X_0)$, and each $i \in [1, |\pi|]$, a sequence

$$v_{p,\pi}(i) \in \{\, visit_j(X_k, \tilde{\pi}) \mid j \in [1, |\tilde{\pi}|],\ \tilde{\pi} \in F(X_k),\ k \in [1, n] \,\}^*$$

describing, for every node x at which production p is applied and partition π is selected, the sequence of visits that must be made to the children of x during the i-th visit to x. The sequence $v_{p,\pi}(1) \cdots v_{p,\pi}(|\pi|)$ is such that sequence

$$visit_1(X_k, \tilde{\pi}) \cdots visit_{|\tilde{\pi}|}(X_k, \tilde{\pi})$$

with $k \in [1, n]$ and $\tilde{\pi} \in F(X_k)$ is obtained when all elements in the set $\{\, visit_j(X_l, \bar{\pi}) \mid j \in [1, |\bar{\pi}|],\ \bar{\pi} \in F(X_l),\ l \in [1, n],\ l \neq k \,\}$ are deleted.

(3) For each production $p \in P$, a set of evaluation rules $E(p)$ as per Definition 2(4). □

It will be clear from this definition that simple multi-visit evaluators are absolutely non-circular attribute evaluators where the family of totally ordered partitions over the attributes $A(X)$ of each $X \in N$ consists of a single partition. Similar to these evaluators, the behaviour of absolutely non-circular attribute evaluators can be expressed by a set of extended visit-sequences. This set of extended visit-sequences, however, has one extended visit-sequence for each production $p : X_0 \rightarrow X_1 \cdots X_n \in P$ and each partition $\pi \in F(X_0)$.

Let $I_{p,\pi,j}(X_i)$ be the set $\{\, (\alpha, p, i) \mid \alpha \in A_j(X_i) \cap I(X_i) \,\}$ and $S_{p,\pi,j}(X_i)$ be the set $\{\, (\alpha, p, i) \mid \alpha \in A_j(X_i) \cap S(X_i) \,\}$ for all $p \in P$, $\pi \in F(X_i)$, $j \in [1, m]$, and $i \in [0, n]$, where p is assumed to be $X_0 \rightarrow X_1 \cdots X_n$ and $\pi = A_1(X_i), \ldots, A_m(X_i)$.

Definition 15. The *extended visit-sequence of* $p : X_0 \rightarrow X_1 \cdots X_n \in P$ *and* $\pi \in F(X_0)$ is the sequence $V_{p,\pi} = V_{p,\pi}(1) \cdots V_{p,\pi}(|\pi|)$ where, for every $i \in [1, |\pi|]$, sequence $V_{p,\pi}(i)$ equals

$$\langle I_{p,\pi,i}(X_0), I_{p,\pi_{j_1},m_1}(X_{j_1}), visit_{m_1}(X_{j_1},\pi_{j_1}), S_{p,\pi_{j_1},m_1}(X_{j_1}), \ldots$$

$$\ldots, I_{p,\pi_{j_r},m_r}(X_{j_r}), visit_{m_r}(X_{j_r},\pi_{j_r}), S_{p,\pi_{j_r},m_r}(X_{j_r}), S_{p,\pi,i}(X_0)\rangle,$$

if $v_{p,\pi}(i) = visit_{m_1}(X_{j_1},\pi_{j_1}) visit_{m_2}(X_{j_2},\pi_{j_2}) \cdots visit_{m_r}(X_{j_r},\pi_{j_r})$. □

The notation $set(\alpha, p, j)$ is used to denote the (unique) set in $V_{p,\pi}$ that contains attribute occurrence (α, p, j). For elements a and b in $V_{p,\pi}$, the expression "$a < b$ in $V_{p,\pi}$" will be written if a occurs before b in $V_{p,\pi}$.

Example 9. Let G be an attribute grammar with start symbol S, nonterminals $N = \{ S, A \}$, terminals $\Sigma = \varnothing$, and productions $P = \{ p_1 : S \to A,$ $p_2 : S \to A,\ p_3 : A_0 \to A_1 A_2,\ p_4 : A \to \epsilon,\ p_5 : A \to \epsilon \}$. The sets of attributes of each nonterminal are:

$$\begin{array}{ll} I(S) = \varnothing & I(A) = \{ \delta, i, \} \\ S(S) = \{ s \} & S(A) = \{ s_1, s_2 \} \end{array}$$

Figure 12 shows the relevant aspects of sets $R(p_1)$, $R(p_2)$, $R(p_3)$, $R(p_4)$, and $R(p_5)$ by means of their dependency graphs.

Attribute grammar G is evaluated by an absolutely non-circular attribute evaluator E, with $F(S)$ consisting of partition $\pi_S = \{ s \}$ and $F(A)$ of partitions $\pi_A = \{ \delta, s_1 \}, \{ i, s_2 \}$ and $\tilde{\pi}_A = \{ \delta, i, s_1, s_2 \}$. The behaviour of E is given by means of the extended visit sequences of Table 7 (where the notational convention for multiple occurrences of a nonterminal applies so that, for instance, $visit_1(A_1, \pi_A)$ and $visit_1(A_2, \pi_A)$ are different visits).

From these extended visit-sequences it should be clear that E traverses the derivation tree in a preorder depth-first manner (visit root, then leftmost

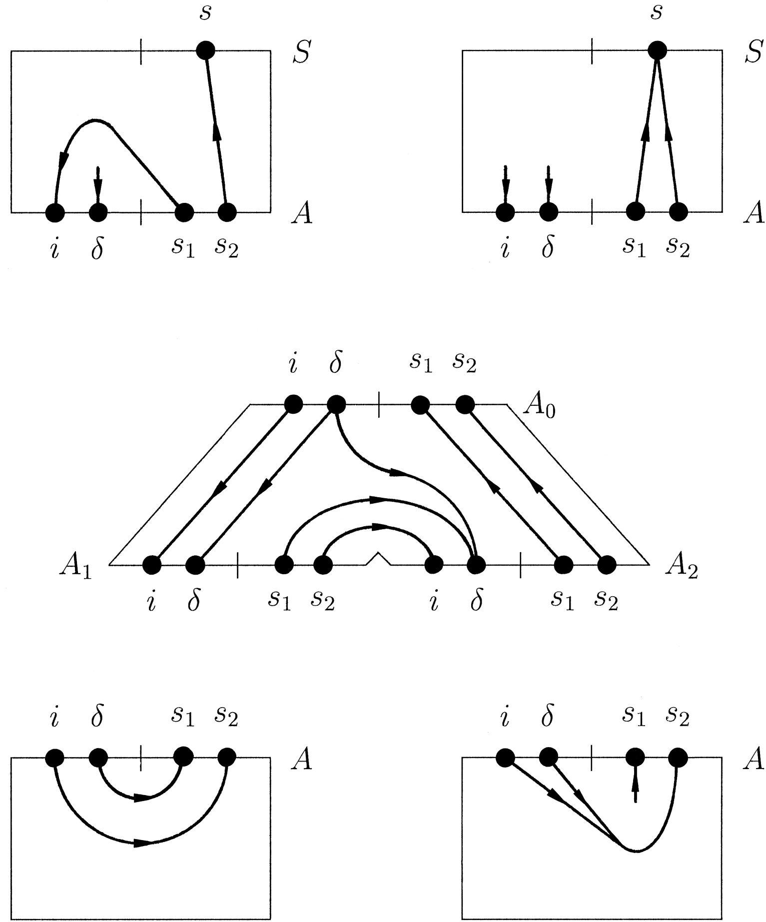

Figure 12. Dependency graphs $DG(p_1)$, $DG(p_2)$, $DG(p_3)$, $DG(p_4)$, and $DG(p_5)$.

subtree to rightmost subtree). When production p_1 is applied at the root, all the instances of attributes δ and s_1 are evaluated in the first traversal, and all the instances of i and s_2 in the second traversal. When production p_2 is applied at the root, all the instances of attributes δ, i, s_1, and s_2 are evaluated in one traversal. □

Other concepts and notations defined for simple multi-visit evaluators like evaluation sequence, existence of an attribute instance, and (a, b)-allocation can be simply generalized to absolutely non-circular attribute evaluators. Assuming that it does not give cause to confusion, these concepts and notations can be freely used for absolutely non-circular attribute evaluators without formally defining them.

Remark. The problem whether an attribute grammar can be evaluated by an absolutely non-circular attribute evaluators is decidable in polynomial time. The size of absolutely non-circular attribute evaluators, however, can be exponential in the size of the grammar. (See [3,5,6,7,11,17,21,22,29,32] for more information on absolutely non-circular attribute evaluators.)

Some implementations of absolutely non-circular attribute evaluators (for instance [17,21,32]) do not evaluate all attribute instances in the derivation tree. These implementations try to evaluate only those attribute instances that contribute to the computation of the attribute instances at the root. Aside from the fact that it is conceivable that other attribute instances have to be delivered as output than those at the root, it will be clear that such implementations are not covered by the notion of absolutely non-circular attribute evaluators as defined here. □

Global storage allocation of applied occurrences

Similar to simple multi-visit evaluators, the concept of subtree information is defined for absolutely non-circular attribute evaluators. Consider an absolutely non-circular attribute evaluator E for an attribute grammar G that has an applied occurrence (α, p, j). Let t be a derivation tree of G and let n be a node of t. Assume $(\beta, p, k) = (\alpha, p, j)_i$ and $(\gamma, p, l) = (\alpha, p, j)_{i+1}$.

Production Rule	Partition	Extended Visit-Sequence
p_1	π_S	$\langle \varnothing, \{(\delta, p_1, 1)\}, visit_1(A, \pi_A), \{(s_1, p_1, 1)\}, \{(i, p_1, 1)\}, visit_2(A, \pi_A), \{(s_2, p_1, 1)\}, \{(s, p_1, 0)\} \rangle$
p_2	π_S	$\langle \varnothing, \{(i, p_2, 1), (\delta, p_2, 1)\}, visit_1(A, \tilde{\pi}_A), \{(s_1, p_2, 1), (s_2, p_2, 1)\}, \{(s, p_2, 0)\} \rangle$
p_3	π_A	$\langle \{(\delta, p_3, 0)\}, \{(\delta, p_3, 1)\}, visit_1(A_1, \pi_A), \{(s_1, p_3, 1)\}, \{(\delta, p_3, 2)\}, visit_1(A_2, \pi_A), \{(s_1, p_3, 2)\}, \{(s_1, p_3, 0)\} \rangle$ $\langle \{(i, p_3, 0)\}, \{(i, p_3, 1)\}, visit_2(A_1, \pi_A), \{(s_2, p_3, 1)\}, \{(i, p_3, 2)\}, visit_2(A_2, \pi_A), \{(s_2, p_3, 2)\}, \{(s_2, p_3, 0)\} \rangle$
p_3	$\tilde{\pi}_A$	$\langle \{(i, p_3, 0), (\delta, p_3, 0)\}, \{(i, p_3, 1), (\delta, p_3, 1)\}, visit_1(A_1, \tilde{\pi}_A), \{(s_1, p_3, 1), (s_2, p_3, 1)\},$ $\{(i, p_3, 2), (\delta, p_3, 2)\}, visit_1(A_2, \tilde{\pi}_A), \{(s_1, p_3, 2), (s_2, p_3, 2)\}, \{(s_1, p_3, 0), (s_2, p_3, 0)\} \rangle$
p_4	π_A	$\langle \{(\delta, p_4, 0)\}, \{(s_1, p_4, 0)\} \rangle$ $\langle \{(i, p_4, 0)\}, \{(s_2, p_4, 0)\} \rangle$
p_4	$\tilde{\pi}_A$	$\langle \{(i, p_4, 0), (\delta, p_4, 0)\}, \{(s_1, p_4, 0), \{(s_2, p_4, 0)\} \rangle$
p_5	π_A	$\langle \{(\delta, p_5, 0)\}, \{(s_1, p_5, 0)\} \rangle$ $\langle \{(i, p_5, 0)\}, \{(s_2, p_5, 0)\} \rangle$
p_5	$\tilde{\pi}_A$	$\langle \{(i, p_5, 0), (\delta, p_5, 0)\}, \{(s_1, p_5, 0), (s_2, p_5, 0)\} \rangle$

Table 7. Extended visit-sequences of absolutely non-circular attribute evaluator E.

Definition 16. The *subtree information* $SI_i(\alpha, p, j)$ for the $(i, i+1)$-allocation of the instances of (α, p, j) is a set of tuples $(X, \pi, (a, b))$ with $X \in N$, $\pi \in F(X)$, $a, b \in [1, |\pi|]$, and $a \leq b$. A tuple $(X, \pi, (a, b)) \in SI_i(\alpha, p, j)$ when there is a subtree $t_{(n)}$, n labelled by X, that has a node m at which production p is applied. This subtree is evaluated by E (as part of t) in such a way that partition π is selected for node n while the instance of $(\alpha, p, j)_i$ at node m_k is computed during the a-th visit to n and the instance of $(\alpha, p, j)_{i+1}$ at node m_l is computed during the b-th visit to n. □

The constructability of set $SI_i(\alpha, p, j)$ is shown by the following lemma using the extended visit sequences of evaluator E. For all elements a in $V_{p,\pi}$, let $a \in^+ V_{p,\pi}(n)$ denote that a occurs in sequence $V_{p,\pi}(n)$. For all sets $I_{p,\pi,n}(X_m), S_{p,\pi,n}(X_m) \in^+ V_{p,\pi}(n)$, let $(\delta, p, m) \in^* V_{p,\pi}(n)$ denote that $(\delta, p, m) \in I_{p,\pi,n}(X_m)$ or $(\delta, p, m) \in S_{p,\pi,n}(X_m)$.

Lemma 12. *$SI_i(\alpha, p, j) = SI_i'(\alpha, p, j)$, where $SI_i'(\alpha, p, j)$ is defined by steps (1)-(3) as follows.*

(1) Basis. — If $(\alpha, p, j)_i \in^* V_{p,\pi}(a)$ and $(\alpha, p, j)_{i+1} \in^* V_{p,\pi}(b)$ for some $\pi \in F(X_0)$, then tuple $(X_0, \pi, (a, b)) \in SI_i'(\alpha, p, j)$ where p is assumed to be $X_0 \rightarrow X_1 \cdots X_n$.

(2) Projection. — If $(X, \tilde{\pi}, (r, s)) \in SI_i'(\alpha, p, j)$ and there is a production $q : Y_0 \rightarrow Y_1 \cdots Y_m$ with nonterminal $Y_k = X$ for some $k \in [1, m]$ such that $visit_r(Y_k, \tilde{\pi}) \in^+ V_{q,\pi}(a)$ and $visit_s(Y_k, \tilde{\pi}) \in^+ V_{q,\pi}(b)$ holds for a $\pi \in F(Y_0)$, then tuple $(Y_0, \pi, (a, b)) \in SI_i'(\alpha, p, j)$.

(3) Closure. — Nothing is in $SI_i'(\alpha, p, j)$ except those tuples which are in $SI_i'(\alpha, p, j)$ by applying the basis step (1) and the projection step (2) on a finite number of occasions.

Proof. Immediate by adjusting the proof of Lemma 7 and the assumption that, for all $X \in N$, each $\pi \in F(X)$ is necessary. □

Example 10. Referring to Example 9, by the basis step and the projection step of Lemma 12, it is found that

$$\begin{aligned} SI_0(\delta, p_3, 0) &= SI_1(\delta, p_3, 0) = SI_0(\delta, p_4, 0) \\ &= \{ (S, \pi_S, (1,1)),\ (A, \pi_A, (1,1)),\ (A, \tilde{\pi}_A, (1,1)) \} \\ SI_0(\delta, p_5, 0) &= \{ (S, \pi_S, (1,1)),\ (A, \pi_A, (1,2)),\ (A, \tilde{\pi}_A, (1,1)) \} \end{aligned}$$ □

The next lemma uses the subtree information to decide whether an absolutely non-circular attribute evaluator can (a, b)-allocate the attribute instances of an applied occurrence to a global stack.

Lemma 13. *Let E be an absolutely non-circular attribute evaluator for an attribute grammar G and let $p : X_0 \to X_1 \cdots X_n \in P$ with $(\alpha, p, j) \in AO(p)$. The attribute instances of (α, p, j) cannot be (a, b)-allocated to a global stack during evaluation by E if and only if at least one of the following conditions is satisfied.*

(1) There is a tuple $(X_k, \tilde{\pi}, (d, e)) \in \bigcup_{i=a}^{b-1} SI_i(\alpha, p, j)$ for some $k \in [1, n]$ such that for some $i \in [a, b-1]$ and $\pi \in F(X_0)$,

$$set(\alpha, p, j)_i < visit_d(X_k, \tilde{\pi}) < set(\alpha, p, j)_{i+1} < visit_e(X_k, \tilde{\pi}) \text{ in } V_{p,\pi}.$$

(2) There is a tuple $(X_k, \tilde{\pi}, (d, e)) \in \bigcup_{i=a}^{b-1} SI_i(\alpha, p, j)$ for some $k \in [1, n]$ such that for some $i \in [a, b-1]$ and $\pi \in F(X_0)$,

$$visit_d(X_k, \tilde{\pi}) < set(\alpha, p, j)_i < visit_e(X_k, \tilde{\pi}) < set(\alpha, p, j)_{i+1} \text{ in } V_{p,\pi}.$$

(3) *There is a production* $q : Y_0 \to Y_1 \cdots Y_m \in P$, *and tuples* $(Y_k, \tilde{\pi}, (d,e))$, $(Y_l, \bar{\pi}, (r,s)) \in \bigcup_{i=a}^{b-1} SI_i(\alpha, p, j)$ *for some distinct* $k, l \in [1, m]$ *such that for some* $\pi \in F(Y_0)$,

$$visit_d(Y_k, \tilde{\pi}) < visit_r(Y_l, \bar{\pi}) < visit_e(Y_k, \tilde{\pi}) < visit_s(Y_l, \bar{\pi}) \text{ in } V_{q,\pi}. \square$$

In order to prove this lemma it is necessary to know what it means that the instances of an applied occurrence (α, p, j) can be (a, b)-allocated to a global stack during evaluation by an absolutely non-circular attribute evaluator E. Although it requires a straightforward modification of Definition 7, it is formalized here to avoid confusion:

> A group A of attribute instances of (α, p, j) can be (a, b)-*allocated to a global stack during evaluation by* E if for all instances $\gamma \in A$, with $\xi(\gamma) = \langle g_1, \ldots, g_u \rangle$, the following holds: there is no instance $\delta \in A$, with $\xi(\delta) = \langle d_1, \ldots, d_u \rangle$, such that $g_a < d_a < g_i < d_b$ for some $i \in [a+1, b]$.

Proof of Lemma 13. Let t be a derivation tree with x and y denoting two of its nodes at which production p is applied. The instance of (α, p, j) at node x_j is denoted by γ and the instance of (α, p, j) at node y_j is denoted by δ, where $\xi(\gamma) = \langle g_0, \ldots g_u \rangle$ and $\xi(\delta) = \langle d_0, \ldots, d_u \rangle$.

Firstly it will be shown that the instances of (α, p, j) cannot be (a, b)-allocated to a global stack during evaluation by E if one of the conditions (1)-(3) is satisfied. Each condition is treated separately (recall that G_u is reduced and that, for all $X \in N$, each $\pi \in F(X)$ is necessary).

(a) *Condition (1) is satisfied.* — Then G has a derivation tree like t (where y is a descendant of x_k or is x_k itself) which is evaluated by E in such

a way that π is selected at x and that $g_v < d_w < g_{v+1} < d_{w+1}$ for some $v, w \in [a, b-1]$. Now if $g_a < d_a$ then $g_a < d_a \leq d_w < g_{v+1} < d_{w+1} \leq d_b$ so that $g_a < d_a < g_{v+1} < d_b$. If on the other hand $d_a < g_a$ then $d_a < g_a \leq g_v < d_w < g_{v+1} < g_b$ so that $d_a < g_a < d_w < g_b$. In both cases it is evident from the definition above that the instances of (α, p, j) cannot be (a, b)-allocated to a global stack during evaluation by E.

(b) *Condition (2) is satisfied.* — Then G has a derivation tree like t (where y is a descendant of x_k or is x_k itself) which is evaluated by E in such a way that π is selected at x and that $d_w < g_v < d_{w+1} < g_{v+1}$ for some $v, w \in [a, b-1]$. In the same way as in the previous case, it follows that the instances of (α, p, j) cannot be (a, b)-allocated to a global stack during evaluation by E.

(c) *Condition (3) is satisfied.* — Let z be a node of t at which production q is applied. Then G has a tree like t (where x is a descendant of node z_k or is z_k itself, and y is a descendant of node z_l or is z_l itself) which is evaluated by E in such a way that π is selected at z and that $g_v < d_w < g_{v+1} < d_{w+1}$ for some $v, w \in [a, b-1]$. Similar to the first case, it can be shown that the instances of (α, p, j) cannot be (a, b)-allocated to a global stack during evaluation by E.

Secondly, it will be shown that at least one of the conditions (1)-(3) is satisfied if the instances of (α, p, j) cannot be (a, b)-allocated to a global stack during evaluation by E. Therefore, assume that the instances of (α, p, j) cannot be (a, b)-allocated to a global stack during evaluation by E. Then G must have a derivation tree like t which is evaluated by E in such a way that

$g_a < d_a < g_i < d_b$ for some $i \in [a+1, b]$ (compare the definition on page 120). Hence there exists a $w \in [a, b-1]$ such that $g_a < d_w < g_i < d_{w+1}$, and so there exists a $v \in [a, b-1]$ such that $g_v < d_w < g_{v+1} < d_{w+1}$. Three cases are distinguished.

(a) *Node y is a descendant of node x_k or is x_k itself for some $k \in [1, n]$.*— Let $\pi \in F(X_0)$ be the partition selected by evaluator E for node x. Then there must be a tuple $(X_k, \tilde{\pi}, (d, e))$ in set $SI_w(\alpha, p, j)$ such that $set(\alpha, p, j)_v < visit_d(X_k, \tilde{\pi}) < set(\alpha, p, j)_{v+1} < visit_e(X_k, \tilde{\pi})$ in $V_{p,\pi}$. This implies that condition (1) is satisfied.

(b) *Node x is a descendant of y_k or is y_k itself for some $k \in [1, n]$.*— Let $\pi \in F(X_0)$ be the partition selected by evaluator E for node y. Then there must be a tuple $(X_k, \tilde{\pi}, (d, e))$ in set $SI_v(\alpha, p, j)$ such that $visit_d(X_k, \tilde{\pi}) < set(\alpha, p, j)_w < visit_e(X_k, \tilde{\pi}) < set(\alpha, p, j)_{w+1}$ in $V_{p,\pi}$. This implies that condition (2) is satisfied.

(c) *Nodes x and y are incomparable.*— Let z be the root of the smallest tree of t containing both x and y, let $q : Y_0 \to Y_1 Y_2 \cdots Y_m$ be the production applied at z, and let $\pi \in F(Y_0)$ be the selected partition at node z. Assume x is a descendant of z_k or z_k itself and y a descendant of z_l or z_l itself, for distinct $k, l \in [1, m]$. Then there are two tuples $(Y_k, \tilde{\pi}, (d, e))$ and $(Y_l, \bar{\pi}, (r, s))$ in set $\bigcup_{i=a}^{b-1} SI_i(\alpha, p, j)$ such that $visit_d(Y_k, \tilde{\pi}) < visit_r(Y_l, \bar{\pi}) < visit_e(Y_k, \tilde{\pi}) < visit_s(Y_l, \bar{\pi})$ in $V_{q,\pi}$. This implies that condition (3) is satisfied. □

The following two lemmas provide necessary and sufficient conditions to decide whether an absolutely non-circular attribute evaluator can (a, b)-allocate the attribute instances of an applied occurrence to a global queue.

The proof of Lemma 14 is similar to that of Lemma 13 and the proof of Lemma 15 is directly obtained by adjusting the proof of Lemma 11 using the fact that, for all $X \in N$, each $\pi \in F(X)$ is necessary.

Lemma 14. *Let E be an absolutely non-circular attribute evaluator for an attribute grammar G and let $p : X_0 \to X_1 \cdots X_n \in P$ with $(\alpha, p, j) \in AO(p)$. The attribute instances of (α, p, j) cannot be $(a, a+1)$-allocated to a global queue during evaluation by E if and only if at least one of the following conditions is satisfied.*

(1) There exists a tuple $(X_k, \tilde{\pi}, (d, e)) \in SI_a(\alpha, p, j)$ for some $k \in [1, n]$ such that for some $\pi \in F(X_0)$,

$$set(\alpha, p, j)_a < visit_d(X_k, \tilde{\pi}) \text{ and } visit_e(X_k, \tilde{\pi}) < set(\alpha, p, j)_{a+1} \text{ in } V_{p,\pi}.$$

(2) There exists a tuple $(X_k, \tilde{\pi}, (d, e)) \in SI_a(\alpha, p, j)$ for some $k \in [1, n]$ such that for some $\pi \in F(X_0)$,

$$visit_d(X_k, \tilde{\pi}) < set(\alpha, p, j)_a \text{ and } set(\alpha, p, j)_{a+1} < visit_e(X_k, \tilde{\pi}) \text{ in } V_{p,\pi}.$$

(3) There exist a production $q : Y_0 \to Y_1 \cdots Y_m$, and tuples $(Y_k, \tilde{\pi}, (d, e))$, $(Y_l, \bar{\pi}, (r, s)) \in SI_a(\alpha, p, j)$ for some distinct $k, l \in [1, m]$ such that for some $\pi \in F(Y_0)$,

$$visit_d(Y_k, \tilde{\pi}) < visit_r(Y_l, \bar{\pi}) \text{ and } visit_s(Y_l, \bar{\pi}) < visit_e(Y_k, \tilde{\pi}) \text{ in } V_{q,\pi}. \square$$

Lemma 15. *Let E be an absolutely non-circular attribute evaluator for an attribute grammar G and let $p : X_0 \to X_1 \cdots X_n \in P$ with $(\alpha, p, j) \in AO(p)$. The attribute instances of (α, p, j) can be (a, b)-allocated to a global queue during evaluation by E if and only if E can $(a, b-1)$-allocate them to a global queue and none of the following conditions is satisfied.*

(1) There exists a tuple $(X_k, \tilde{\pi}, (d, e))$ *in* $SI_{b-1}(\alpha, p, j)$ *for some* $k \in [1, n]$ *such that for some* $\pi \in F(X_0)$*,*

$$visit_d(X_k, \tilde{\pi}) < set(\alpha, p, j)_{a+1} < visit_e(X_k, \tilde{\pi}) \text{ in } V_{p,\pi}.$$

(2) There exists a tuple $(X_k, \tilde{\pi}, (d, e))$ *in* $SI_a(\alpha, p, j)$ *for some* $k \in [1, n]$ *such that for some* $\pi \in F(X_0)$*,*

$$set(\alpha, p, j)_{b-1} < visit_e(X_k, \tilde{\pi}) < set(\alpha, p, j)_b \text{ in } V_{p,\pi}.$$

(3) There exist a production $q : Y_0 \rightarrow Y_1 \cdots Y_m$ *and tuples* $(Y_k, \tilde{\pi}, (d, e))$ *in* $SI_a(\alpha, p, j)$ *and* $(Y_l, \bar{\pi}, (r, s))$ *in* $SI_{b-1}(\alpha, p, j)$ *for distinct* $k, l \in [1, m]$ *such that for some* $\pi \in F(Y_0)$*,*

$$visit_r(Y_l, \bar{\pi}) < visit_e(Y_k, \tilde{\pi}) < visit_s(Y_l, \bar{\pi}) \text{ in } V_{q,\pi}. \qquad \square$$

From Lemma 13 it can be inferred that an applied occurrence (α, p, j) of a production $p : X_0 \rightarrow X_1 \cdots X_k$ can be (a, b)-allocated to a global stack if, for every partition $\pi \in F(X_0)$, $(\alpha, p, j)_a$ and $(\alpha, p, j)_b$ occur within the same subsequence of $V_{p,\pi}$. This means that, for each sequence $V_{p,\pi}$, the number of reallocations of an applied occurrence (α, p, j) can be reduced to $n - m$ in the worst case, where (α, p, j) occurs in $V_{p,\pi}(m)$, and the occurrence which uses (α, p, j) for the last time occurs in $V_{p,\pi}(n)$.

The main result of this postscript is the following theorem.

Theorem 9. *Let* E *be an absolutely non-circular attribute evaluator for an attribute grammar* G *and* (α, p, j) *an applied occurrence of* G*. It is decidable in exponential time whether the instances of* (α, p, j) *can be* (a, b)*-allocated to a global variable, stack, or queue during evaluation by* E*.*

Proof. The number of unordered partitions of a set of $n \geq 0$ elements into $k \in [0, n]$ disjoint nonempty subsets is given by the Stirling number of the second kind $S(n, k)$; see also [24] for more information. Hence, for all $X \in N$, there are $k!\, S(n, k)$ ordered partitions of $A(X)$ into k disjoint nonempty subsets, where $n = |A(X)|$, so that there can be maximal $\sum_{k=0}^{n} k!\, S(n, k)$ ordered partitions in $F(X)$. This effectively means that there can be an exponential number of tuples in a set $SI_i(\alpha, p, j)$ since

$$SI_i(\alpha, p, j) \subseteq N \times \Pi \times [1, K] \times [1, K],$$

where $\Pi = \{\, \pi \,|\, \pi \in F(X),\ X \in N \,\}$ and $K = \max\{\, |\pi| \,|\, \pi \in F(X),\ X \in N \,\}$. Using this observation the proof is readily obtained by Lemmas 13-15 and the analogue of Lemma 1. □

Applied Occurrence	(0,1)-Allocation	(0,2)-Allocation
$(\delta, p_3, 0)$		global stack
$(\delta, p_4, 0)$	global variable	
$(\delta, p_5, 0)$	global queue	

Table 8. Allocation of applied occurrences $(\delta, p_3, 0)$, $(\delta, p_4, 0)$, and $(\delta, p_5, 0)$.

Example 11. Consider Examples 9 and 10. By means of Lemmas 13-15 and the analogue of Lemma 1 the allocations of Table 8 can be found for the instances of $(\delta, p_3, 0)$, $(\delta, p_4, 0)$, and $(\delta, p_5, 0)$ during evaluation by E. □

In what follows comparisons will be made with the storage allocation techniques of Saarinen [32] and Katayama [21]. So that their absolutely noncircular attribute evaluators need to compute all attribute instances in the

derivation tree it is assumed that the attribute grammars under consideration are such that all attribute instances in their derivation trees contribute to the computation of the attribute instances of the root. This assumption can be made without loss of generality (by introducing artificial dependencies) and has the advantage that the concept of absolutely non-circular attribute evaluator (notations included) as defined in this postscript can be used with the evaluators of Saarinen and Katayama.

Comparison 1. Considering the allocation of the applied occurrences of each visit sequence $V_{p,\pi}$ in isolation, Saarinen [32] allocated the temporary applied occurrences of each $V_{p,\pi}$ to a global stack (at least that was his intention so far as this allocation caused no problems with the other paths in the history graph of production p that might part or merge with the path described by $V_{p,\pi}$). An applied occurrence (α, p, j) is temporary in sequence $V_{p,\pi}$ when (α, p, j) and each defined occurrence (β, p, k) directly depending on (α, p, j) occur within the same subsequence of $V_{p,\pi}$.

Here allocation of an applied occurrence (α, p, j) is decided so that it is applicable for all visit sequences $V_{p,\pi}$. An applied occurrence (α, p, j) is therefore always allocated to a global stack when it is temporary for all $V_{p,\pi}$ (see also Lemma 13). Since the instances of (α, p, j) in a sequence $V_{p,\pi}$ are a subset of all instances of (α, p, j) it is conceivable that Saarinen is able to allocate (α, p, j) to a global stack for a sequence $V_{p,\pi}$ while this is not possible for all extended visit-sequences of p.

Specializing the definitions and lemmas in this postscript so that allocation is decided for the instances of an applied occurrence (α, p, j) in a particular visit sequence $V_{p,\pi}$ is straightforward, but its implementation is not. To create nodes where the instances of (α, p, j) can be stored that can-

not be allocated elsewhere, it must be known at derivation tree construction time which partitions the evaluator will select for the nodes. This is no problem for evaluators where the partitions must be selected and stored in the nodes in a separate stage (like Saarinen's evaluator) but it requires extra compilation time when this is not the case. □

Comparison 2. In [21], where absolutely non-circular attribute evaluators are implemented by mutually recursive procedures, Katayama utilized the standard procedure mechanism of programming languages to store the instances of the attributes in the stack frames of the procedure stack. Attributes α whose applied occurrences (α, p, j) are temporary for all extended visit-sequences of p are declared as local variables in the procedures $Q_{p,s}$ so that their instances are stored on the procedure stack without explicit stack manipulations. Although these instances can also be allocated to global stacks (see also Lemma 13), this solution may be more efficient since many machines offer hardware support for procedure mechanisms.

Katayama also suggests to store the instances of other attributes on the procedure stack, but does not provide much details (the procedure construction of [21, Section 6.1] does not apply for them). Aside from this omission, he also makes a mistake to claim [21, page 363] that the maximum length of the procedure stack is proportional to the height of the derivation tree. When every attribute instance is allocated to the procedure stack, which is what he proposes, the maximum length of the procedure stack can be exponential to the height of the derivation tree. Consider, for instance, the space requirements for applied occurrence $(\delta, p_5, 0)$ of Example 9. When production p_1 is applied at the root of a derivation tree t, the number of instances of $(\delta, p_5, 0)$ that needs to be stored between the first and the second traversal

of E to t can be exponential in the height of t. Hence, assuming that it is possible, allocating the instances of $(\delta, p_5, 0)$ to the procedure stack implies that the maximum length of the procedure stack is exponential in the height of the derivation tree. □

Conclusions

It is decidable in exponential time whether an absolutely non-circular attribute evaluator can (a, b)-allocate the instances of an applied occurrence to a global variable, stack, or queue. The exponential factor in the complexity of this decidability problem is the number of ordered partitions in the set Π as defined in the proof of Theorem 9, so that it remains to be seen whether Lemmas 13, 14, and 15 have practical significance.

The remark that was made with respect to evaluation functions in simple multi-visit evaluators (see page 106) also holds for absolutely non-circular evaluators. Hence, Lemmas 13, 14, and 15 can also be used when evaluation functions are considered which evaluate the actual parameters of a function call concurrently.

Finally, a word on the data structures that are needed for the $(a, a+1)$-allocation of the instances of applied occurrences in absolutely non-circular attribute evaluators. For these evaluators it is also necessary and sufficient to consider the set of basic linear data structures $\mathcal{D}$ for the $(a, a+1)$-allocation of the instances of applied occurrences. The necessity follows from Theorem 7 since simple multi-visit evaluators are a proper subset of absolutely non-circular evaluators, and the sufficiency follows from similar arguments as given for simple multi-visit evaluators on page 35.

References

1. Aho, A. V., Seti, R., and Ullman, J. D.: "Compilers: principles, techniques, and tools", Addison-Wesley (1986).

2. Alblas, H.: "A characterization of attribute evaluation in passes", Acta Informatica **16** (1981), 427-464.

3. Alblas, H.: "Attribute evaluation methods". In: Proceedings of the International Summer School SAGA, Lecture Notes in Computer Science **545** (1991), 48-113.

4. Bochmann, G. V.: "Semantic evaluation from left to right", Communications of the ACM **19** (1976), 55-62.

5. Courcelle, B. and Franchi-Zannettacci, P.: "Attribute grammars and recursive program schemes", Theoretical Computer Science **17** (1982), 163-191 and 235-257.

6. Deransart, P., Jourdan, and M., Lorho, B.: "Attribute grammars". Lecture Notes in Computer Science **323** (1988).

7. Engelfriet, J.: "Attribute Grammars: attribute evaluation methods". In: Methods and Tools for Compiler Construction, Cambridge University Press (1984), 103-138.

8. Engelfriet, J. and Filé, G.: "Simple multi-visit attribute grammars", Journal of Computer and System Sciences **24** (1982), 283-314.

9. Engelfriet, J. and Filé, G.: "Passes, sweeps and visits in attribute grammars", Journal of the ACM **36** (1989), 841-869.

10. Engelfriet, J. and De Jong, W.: "Attribute storage optimization by stacks", Acta Informatica **27** (1990), 567-581.

11. Farrow, R.: "Sub-protocol-evaluators for attribute grammars", ACM SIGPLAN Notices **19(6)** (1984), 70-80.

12. Farrow, R. and Yellin, D.: "A comparison of storage optimizations in automatically-generated attribute evaluators" , Acta Informatica **23** (1986), 393-427.

13. Ganzinger, H.: "On storage optimization for automatically generated compilers". In: 4th GI-Conference on Theoretical Computer Science, Lecture Notes in Computer Science **67** (1979), 132-141.

14. Jazayeri, M., Ogden, W. F., and Rounds, W. C.: "The intrinsically exponential complexity of the circularity problem for attribute grammars", Communications of the Association for Computing Machinery **18** (1975), 679-706.

15. Jazayeri, M. and Pozefsky, D.: "Space-efficient storage management in an attribute evaluator", ACM Transactions on Programming Languages and Systems **3** (1981), 388-404.

16. Jazayeri, M. and Walter, K. G.: "Alternating semantic evaluator", In: ACM 1975 Annual Conference (1975), 230-234.

17. Jourdan, M.: "Strongly non-circular attribute grammars and their recursive evaluation", ACM SIGPLAN Notices **19(6)** (1984), 81-93.

18. Julié, C. and Parigot, D.: "Space optimization in the FNC-2 attribute grammar system". In: Attribute grammars and their applications, Lecture Notes in Computer Science **461** (1990), 29-45.

19. Kastens, U.: "Ordered attribute grammars", Acta Informatica **13** (1980), 229-256.

20. Kastens, U.: "Lifetime analysis for attributes", Acta Informatica **24** (1987), 633-652.

21. Katayama, T.: "Translation of attribute grammars into procedures", ACM Transactions on Programming Languages and Systems **6** (1984), 345-369.

22. Kennedy, K. and Warren, S. K.: "Automatic generation of efficient evaluators for attribute grammars". In: 3rd ACM Symposium on Principles of Progrogramming Languages (1976), 32-49.

23. Knuth, D. E.: "Semantics of context-free languages", Mathematical Systems Theory **2** (1968), 127-145. Correction in: Mathematical Systems Theory **5** (1971), 95-96.

24. Knuth, D. E.: "The art of computer programming: fundamental algorithms", second edition, Addison-Wesley (1973).

25. Lorho, B.: "Semantic attributes processing in the system DELTA". In: Methods of algorithmic language implementation, Lecture Notes in Computer Science **47** (1977), 21-40.

26. Pugh, W. W.: "Incremental computation and the incremental evaluation of functional programs". Ph.D. thesis, Cornell University (1988).

27. Räihä, K.-J.: "Dynamic allocation of space for attribute instances in multi-pass evaluators of attribute grammars", ACM SIGPLAN Notices **14(8)** (1979), 26-38.

28. Räihä, K.-J. and Tarhio, J.: "A globalizing transformation for attribute grammars", ACM SIGPLAN Notices **21(7)** (1986), 74-84.

29. Riis Nielson, H.: "Computation sequences: a way to characterize classes of attribute grammars", Acta Informatica **19** (1983), 255–268.

30. Reps, T. and Demers, A: “Sublinear-space evaluation algorithms for attribute grammars”, ACM Transactions on Programming Languages and Systems **9** (1987), 408-440.

31. Sasaki, H. and Katayama, T.: “Global storage allocation in attribute evaluation”. In: Proceedings of the RIMS Symposia on Software Science and Engineering II, Lecture Notes in Computer Science **220** (1986), 181-211.

32. Saarinen, M.: “On constructing efficient evaluators for attribute grammars”. In: Proceedings of the 5-th ICALP Conference, Lecture Notes in Computer Science **62** (1978), 392-397.

33. Sluiman, F.J.: “Space optimization for simple multi-visit evaluators”. In: Proceedings of the Bilateral Workshop on Compiler Construction, University of Twente (1990), 121-135.

34. Sonnenschein, M.: “Global storage cells for attributes in an attribute grammar”, Acta Informatica **22** (1985), 397-420.

35. Vogt, H.H., Swierstra, S.D., and Kuiper, M.F.: “Efficient incremental evaluation of higher order attribute grammars”. In: Proceedings of the 3rd International Symposium on Programming Language Implementation and Logic Programming, Lecture Notes in Computer Science **528** (1991), 231-242.

36. Vogt, H.H., Swierstra, S.D., and Kuiper, M.F.: “Higher order attribute grammars”. In: Proceedings of the ACM SIGPLAN ’89 Conference on Programming Language Design and Implementation **24** (1989), 131-145.

Index of Symbols

The symbol index contains the most important symbols used. This index is alphabetically ordered, with the understanding that, for example, $\alpha \cdot \beta$ is alphabetized as "alpha beta", $[\vartheta]_\omega$ as "theta omega", (α, p, j) as "alpha p j", $A_1(X)$ as "a one x", and so on. Boldface numbers refer to the pages where a symbol is introduced.

A

$a < b$ in V_p — a occurs before b in sequence V_p **24**

$a < b$ in $V_{p,\pi}$ — a occurs before b in sequence $V_{p,\pi}$ **114**

A_d — attribute instances of (α, p, j) that occur in diagram d **48**

$A_1(X), \ldots, A_{\phi(X)}(X)$ — partition of set $A(X)$ **17**

$A(X)$ — set of attributes of nonterminal X **14**

$|\alpha|$ — length of sequence α **36**

$\alpha \cdot \beta$ — concatenation of sequence α and sequence β **36**

$\alpha(n)$ — attribute instance of attribute α at node n **16**

(α, p, j) — attribute occurrence **14**

$(\alpha, p, j)_k$ — for $k > 0$, the defined occurrence whose instances are computed using the instances of applied occurrence (α, p, j) for the k-time **69**

$(\alpha, p, j)_0$ — applied occurrence (α, p, j) itself **69**

$(\alpha, p, j) \in^* V_p(n)$ — (α, p, j) belongs to set $I_{p,n}(X_j)$ or $S_{p,n}(X_j)$ **71**
$(\alpha, p, j) \in^* V_{p,\pi}(n)$ — (α, p, j) belongs to set $I_{p,\pi,n}(X_j)$ or $S_{p,\pi,n}(X_j)$... **118**
(α, X) — uncouling variable........... **95**
$AO(p)$ — set of applied occurrences of production rule p **14**
append (v, q) — operation that puts value v at the rear end of queue q .. **101**

C

$c = \langle \epsilon, \varnothing \rangle$ — initial content............. **36**
$CAO(\alpha, X)$ — set of coupled applied occurrences of attribute α.......... **94**
$CDO(\alpha, X)$ — set of coupled defined occurrences of attribute α.......... **94**

D

D — basic linear data structure **36**
$\mathcal{D}$ — class of all basic linear data structures **38**
$d(i)$ — direction of the i-th pass...... **19**
decrease (q) — operation that deletes the value at the front end of queue q **101**
$\delta \in \{ i, o \}^*$ — state of basic linear data structure **36**
$DG(p)$ — dependency graph of production rule p........................... **15**
$DO(p)$ — set of defined occurrences of production rule p **14**
$DU_E(\alpha, p, j)$ — set of define-and-use sequences............................... **39**

E

E — (tree-walking) attribute evaluator **16**
$E(p)$ — set of evaluation rules of production rule p........................... **17**
ε — empty string.......................... **13**

F

$f : \{ i, o \}^* \to (\mathbb{N}^+)^*$ — behaviour of a basic linear data structure...... **40**
front (q) — operation that returns the value at the front end of queue q **101**
$F(X)$ — family of totally ordered partitions over the attributes $A(X)$ **112**

G

G — attribute grammar................. **13**
G_u — underlying context-free grammar of attribute grammar G......... **13**

I

$I_{p,j}(X_i)$ — set of attribute occurrences $\{ (\alpha, p, i) \mid \alpha \in A_j(X_i) \cap I(X_i) \}$ **20**
$I_{p,\pi,j}(X_i)$ — set of attribute occurrences **113**
$i : \Sigma \times \boldsymbol{Data} \to \Sigma$ — input operation **36**

$I(X)$ — set of inherited attributes of nonterminal X **14**

M

$M \subseteq Data \times \mathbb{N}^+$ — set of time-stamped data **36**

$[m, n]$ — set of integers $m, m+1, \ldots, n$ **13**

N

N — set of nonterminals **13**

$\mathbb{N}^+$ — set of integers greater than 0 . **36**

O

$o : \Sigma \to \Sigma \times Data$ — output operation **37**

P

P — set of production rules **13**

$|\pi|$ — number of parts in partition π **112**

$\pi \in F(X)$ — partition over the attributes $A(X)$ **112**

$p : \{ i, o \}^* \to \mathbb{N}^+$ — protocol **36**

$p : X_0 \to X_1 \cdots X_n$ — production rule **14**

$\phi(X)$ — total number of visits to any node x labelled X **17**

R

$R(p)$ — set of semantic rules of production rule p **14**

$\rho(t)$ — evaluation sequence of derivation tree t **24**

S

S — start symbol **13**

$S \Rightarrow^* \alpha X \beta$ — derivation **15**

$\mathcal{S} \subset \mathcal{D}$ — set of basic linear data structures sufficient for simple multi-sweep evaluators **58**

$S_{p,j}(X_i)$ — set of attribute occurrences $\{ (\alpha, p, i) \mid \alpha \in A_j(X_i) \cap S(X_i) \}$ **20**

$S_{p,\pi,j}(X_i)$ — set of attribute occurrences **113**

$S(X)$ — set of synthesized attributes of nonterminal X **14**

$set(\alpha, p, j)$ — unique set in an extended visit-sequence that contains (α, p, j) **24, 114**

$SI_i(\alpha, p, j)$ — subtree information **71, 118**

Σ — set of terminals **13**

T

t — derivation tree **15**

$[\vartheta]_\omega$ — number encoding the order in which attribute instance ϑ is computed and used in define-and-use sequence ω **50**

V

V^* — set of all strings on set V **13**

$V(\alpha)$ — set of possible values of attribute α **14**
$visit_d(X_k) \in^+ V_p(n)$ — $visit_d(X_k)$ occurs in sequence $V_p(n)$ **71**
$visit_d(X_k, \tilde{\pi}) \in^+ V_{p,\pi}(n)$ — $visit_d(X_k, \tilde{\pi})$ occurs in sequence $V_{p,\pi}(n)$ ... **118**
$visit_i(X)$ — the i-th visit to a node labelled X **18**
$visit_i(X, \pi)$ — the i-th visit to a node labelled X with selected partition π **112**
$V = N \cup \Sigma$ — vocabulary **15**
$v_p(i)$ — sequence of visits **17**
$v_{p,\pi}(i)$ — sequence of visits **112**
$V_{p,\pi} = V_{p,\pi}(1) \cdots V_{p,\pi}(|\pi|)$ — extended visit-sequence of production rule $p : X_0 \to X_1 \cdots X_n$ and partition $\pi \in F(X_0)$ **113**
$V_p = V_p(1) \cdots V_p(\phi(X_0))$ — extended visit-sequence of production rule $p : X_0 \to X_1 \cdots X_n$ **20**

X

$\#_x(\alpha)$ — number of occurrences of symbol x in sequence α **36**
$\xi(\alpha)$ — existence of attribute instance α **27**

Index of Definitions

The index of definitions contains the most important terms used in this dissertation. Boldface number(s) following a term refer to the page(s) where this term is defined.

A

(a, b)-allocation **69**
  to a global queue **70**
  to a global stack **120**
absolutely
  non-circular attribute evaluator ... **112**
action sequence **39**
allocation
  to a basic linear data structure **40**
  to a global queue **29**
  to a global stack **28**
  to a global variable **28**
alternating protocol **42**
applied occurrence **14**
attribute
  evaluator **16**
  grammar **13**
  instance **16**
  occurrence **14**

B

basic
  linear data structure **36**
  queue **42**
  stack **42**
behaviour
  of a basic linear data structure **40**

C

coupled

applied occurrences **94**

attribute occurrences **93**

defined occurrences **94**

D

define-and-use sequence **38**

defined occurrence **14**

dependency graph **15**

derivation tree **15**

E

evaluation

rule **17**

sequence **24**

existence of an attribute instance **27**

extended visit-sequence **20, 113**

F

family of ordered partitions **112**

G

global storage allocation **2**

I

input operation **36**

L

lifetime of an attribute instance **28**

local storage allocation **2**

M

multi-use

applied occurrence **16**

attribute instance **16**

N

nontemporary attribute **35**

O

output

operation **37**

sequence **39**

P

production rule **14**

protocol **36**

Q

queue **29**

R

reallocation **75**

reduced

attribute grammar **15**

underlying context-free grammar **15**

S

semantic rule **14**

simple multi-
  pass evaluator **19**
  sweep evaluator **18**, **19**
  visit evaluator **16**, **19**
  waddle evaluator **20**
single-use
  applied occurrence **16**
  attribute instance **16**
stack **28**
subtree **16**
  information **71, 118**

T

temporary
  applied occurrence **35, 126**
  attribute **35**

U

uncoupling variable **95**
underlying context-free grammar **13**

V

vocabulary **15**