PROSPECTRA
an ESPRIT Project

Grammar Flow Analysis

Ulrich R. Möncke

Universität des Saarlandes
6600 Saarbrücken
Bundesrepublik Deutschland

Deliverable Item S.1.3 - R - 2.2

1986 - 03 - 01 (revised 1987 - 01 - 13)

Distribution: public

ABSTRACT

This paper specifies the theoretical basis for the implementation of different generators of the OPTRAN system. Grammar Flow analysis transports the techniques of data flow analysis to the meta level of compiler construction. The analogon to the states in data flow analysis are the syntax trees together with some information, which is associated with trees by propagation functions. One example is the association of characteristic graphs, another example the association of sets of matching tree patterns.

public

Ulrich R. Möncke
Universität des Saarlandes

in the Project

PROgram development by
SPECification and
TRAnsformation

sponsored by the

Commission of the European Communities

under the

European
Strategic
Programme for
Research and development in
Information
Technology

Project Ref. No. 390

1. Introduction

Data flow analysis is a well known technique to collect information at compile time concerning possible states of program execution at run time [Cousot,Cousot79]. **Grammar Flow Analysis** (GFA) transports this theory to the generation time computation of compile time properties of syntax trees. The analogon to the states in data flow analysis are the syntax trees together with some information associated with tree nodes by propagation functions. This information may be of a very complex structure. One example is the association of characteristic graphs with tree nodes, another example the association of sets of matching tree patterns. As in data flow analysis we distinguish the universal GFA **scheme** or framework from its **application** in special problem areas (tasks). Characteristic graphs and tree pattern matching are such areas of application, optimal code selection (cf. chapter 5) may be another area.

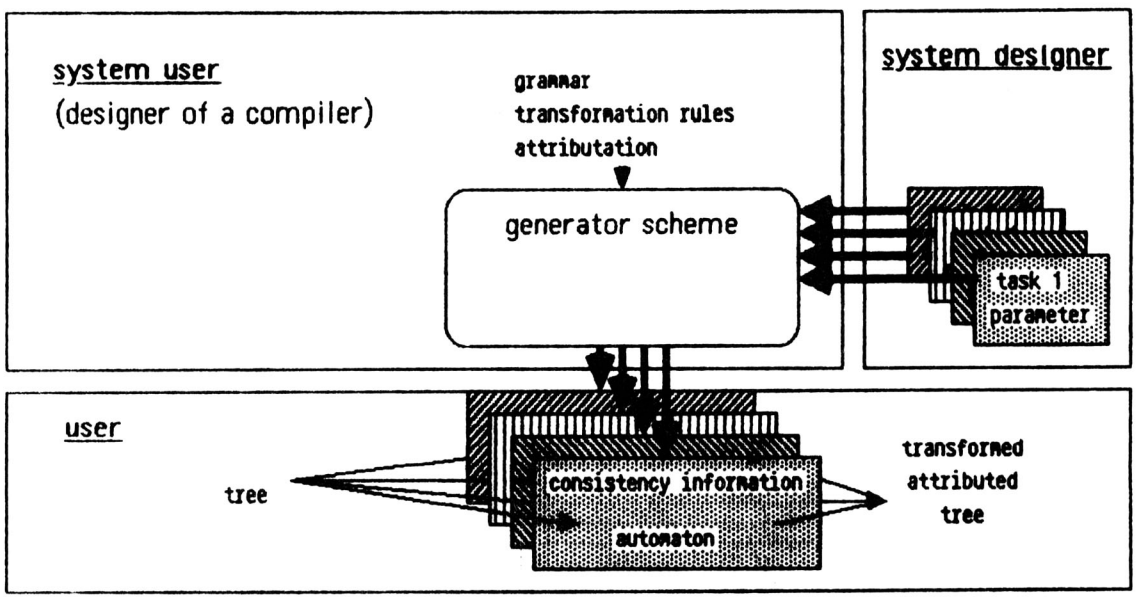

Figure 1

Depending on the nature of the functions, which assign information to tree nodes we furthermore distinguish two kinds of flow analysis schemes. In the **bottom-up scheme**, information is assigned to tree nodes bottom-up starting at the leaves, in the **top-down scheme** information is propagated from the root top down, thereby using the results of bottom-up propagation. Given a node which is not a leave or the root a tree can be split into a subtree and a upper tree fragment. Bottom up information characterizes sets of subtrees, top-down information characterizes sets of upper tree fragments. Parameterized for a given GFA problem, the

GFA scheme precomputes at generation time which information may be associated with subtrees rooted by nonterminal X, resp. precomputes, which information may be associated with an upper tree fragment at a node labeled X at runtime. In the latter case the answer may depend on a bottom-up context, which is the subtree associated with **X** as its root.

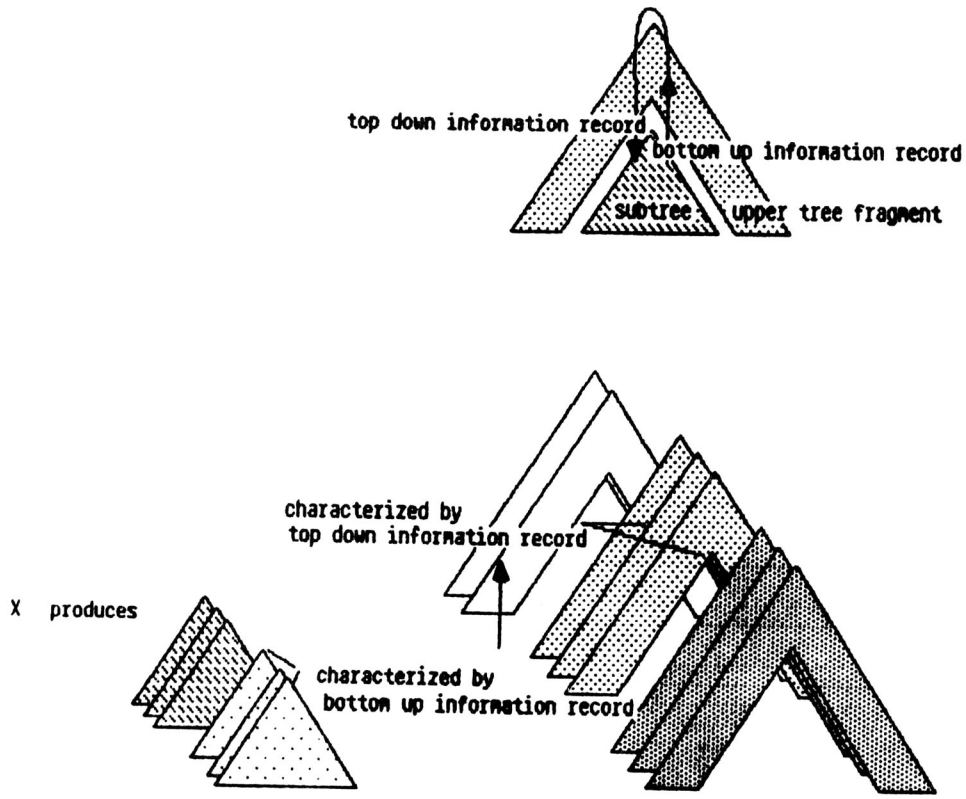

Figure 2

The motivation behind this theoretical framework "grammar flow analysis" stems from the implementation of a major **tree transformational system** including pattern matching and attribute reevaluation facilities. **Embedded** in a compiler – compiler, the tree transformations are executed at compile time. In such a system **precomputing** is done for two reasons:

First, at generation time, the system should try to give the user, who is going to design a transformational system, as much as possible information about the **(compile time) behaviour** of the system. In particular, the user has questions for **consistency**:

Are there trees with cyclic attribute dependencies ?

Are there transformation rules, which can never be applied ?

Are there sequences of transformations, which could block syntactically?

Second, both pattern matching and scheduling the attributes for (re)evaluation are time consuming tasks. Therefore, it is useful, to precompute **automata**, which would execute these tasks very fast. Grammar flow analysis is the mean to get both the consistency information and the efficient automata. Grammar flow analysis can be implemented as a **generic module** making the implementation of different and new applications easy. The generic module provides a lot of special features (cf. chapter 4), which can be used by a system designer with respect to special properties inherent to the application area. Use of such features will accelerate the generation process.

The system designer, i.e. the user of the generic module, must only take care of problem specific functions, i.e. s/he has to model the domain of **information**. Preparing the **grammar graph**, a representation of the underlying grammar, efficient computation of **fixed points**, bookkeeping and management of generated automata are part of the system.

2. Bottom up grammar flow analysis

The description of bottom-up flow analysis is split into two parts:

The first part describes the formalism for assignment of information to tree nodes at **compile time**. In the second part, the method of **precomputation** at **generation time** will be explained.

Before starting let us make a remark concerning our notation: We define objects with respect to individual derivation trees. These objects, which we call **information records** will represent assertions about trees or tree fragments. For a given problem we denote information records by the name of the problem and the approximation type "exact", e.g. "exact characteristic **graphs**". The exact information records are just those records, which we will obtain from derivation trees. Later on, in our grammar flow framework we will precompute these exact information records and, in addition, many approximations (lower and upper bounds) for them. We will denote the approximations by the problem name and the approximation type.

As a running example we will use the GFA problem *"characteristic graphs"*. Characteristic graphs describe relations between the set of inherited attributes I(X) and the set of synthesized attributes S(X) for a nonterminal X. An edge $i \rightarrow s$ in a characteristic graph means that there may be a dependency of an instance of s on an instance of i in some tree rooted X. The "exact characteristic graphs" only

have these edges if there in fact exists at least one tree that has global dependencies inducing these edges.

Another example is *tree pattern matching*. Without going into technical details (cf. [Moencke87]) we assume, that there are tree structured objects called **patterns**, each consisting of a label and a number of subpatterns, according to the rank of the label. Atomic patterns may be labels or parameters.
Trivially, a pattern pat matches a tree, if it is a parameter. A pattern pat matches a tree t rooted in X, if the label matches X and, for i=1..*rank*, each subpattern pat/i of pat matches t/i, the i.th subtree of t. Whether the label matches X or not may be specified by a function g, $g(pat,X) \in \{true, false\}$. We are looking for the set of matching patterns at the root of a given subtree produced by X.

2.1. Bottom up assignment of information records

A **bottom-up assignment system**

$$BU\text{-}ASYS = (G, \{B\uparrow(X)\}_{X \in NT}, \{f_p\}_{p \in P})$$

describes how so called **information records** may be attached to the nodes of syntax trees, produced by a given grammar.

For convenience, we assume that G contains productions of the form $p = (X_0 ::= s_0\ X_1\ s_1\ \ldots\ X_k\ s_k)$ and $p = (X ::= s)$, where X_0, $X_1,\ldots,X_k$ are nonterminals and s is a terminal symbol. Production p has **rank** k, the number of nonterminals on the right side of p, which in the latter case is 0.

The tree nodes may be labeled by pairs (X_0,p), where the leftside nonterminal p[0] of p is X_0. Depending on the context we refer to the production or to the nonterminal. We assume that the grammar is in normal form.

The assignment system contains the context free grammar G, a family of **basic sets** $\{B\uparrow(X)\}_{X \in NT}$, one attached to each nonterminal, and a family of **propagation functions** $\{f_p\}_{p \in P}$, one for each production.

A basic set $B\uparrow(X)$ contains the information records, which (at least in principle) can occur at the root of syntax trees produced by X.

In the *characteristic graph* example $B\uparrow(X)$ is the set of all graphs, which can be constructed with edges from inherited to synthesized attributes of X:
Let 2^M denote the powerset of set M.

$$B\uparrow(X) := 2^{Inh(X) \times Syn(X)}$$

In the *pattern matching* example $B\uparrow(X)$ is the whole set of patterns and subpatterns.

In general, there are elements in $B\uparrow(X)$, which never occur at the root of any syntax tree produced by X. Recognizing such elements requires flow analysis.

f_p denotes a propagation function:

$$f_p : B\uparrow(X_1) \times \cdots \times B\uparrow(X_k) \rightarrow B\uparrow(X_0)$$

where k is the rank of p. The propagation function for a production with rank 0 is a constant. Because this function is applied in an individual syntax tree, we sometimes call it **individual** propagation function. These functions may be not disturbed with the **flow propagation functions** defined in chapter 2.2.

f_p describes how information is associated with a node labeled (X_0,p) given information associated with the descendent nodes X_1.

In our *characteristic graph* example, the function f_p would be the composition of the following computation steps:

1) substitution of characteristic graphs $C_1,\ldots,C_k$ in the local dependency graph of the production, according to their positions.
2) building the transitive closure of the graph obtained in step 1
3) the restriction of this graph to the attributes of X_0.

In our *pattern matching* example, the function f_p would compute from the set of matching patterns and subpattern given for each position those patterns and subpatterns, which match tree t at its root, assumed the above mentioned sets match the subtrees t/i.

Applying the f_p's bottom-up we attach an information record to each subtree by a recursively defined function $f\uparrow$:

Let t/n be a subtree of derivation tree t at node n, n be a node in t :

Let p be the production, applied at node n: $p=(X_0::=X_1 \ldots X_k)$

Therefore, nodes n, n.1 ... n.k are labeled accordingly:

$(X_0,p)=t(n)$, $X_1=t(n.1),\ldots,X_k=t(n.k)$

$$f\uparrow(t/n):=f\uparrow_p(f\uparrow(t/n.1),\ldots,f\uparrow(t/n.k))$$

In principle, there is no need to precompute something: all the propagation could be done at transformation time in an interpretative way, applying the propagation functions bottom-up (cf. [Reps82] for propagation of *characteristic graphs*).

2.2. Simulation at generation time

Applying a propagation function and holding the information records themselves at the tree nodes may be expensive. In our *graph* example each step would cost in the worst case

$\max_{p \in P}((rank(p)+1)*\max_{X \in NT}|Inh(X)*Syn(X)|)^3$, because of the closure building.

Therefore, we would like to precompute the information records, which can actually occur in some tree, encode these sets and represent the propagation functions using a table. Technically, this table implements a propagation function, which is isomorphic to the original propagation function. Let 'codes(X)' denote the set of encoded information records for X, g denote the propagation of encodings, and h the mapping of encodings to information records. h is an injective function.

h_X: $codes(X) \rightarrow B\uparrow(X)$

g_p: $codes(X_1)\ x...x\ codes(X_k) \rightarrow codes(X_0)$

where: $h_{X_0} g_p(c_1,...,c_k) = f_p(h_{X_1}(c_1),...,h_{X_k}(c_k))$

Precomputation is, essentially, the simulation of the propagation process, which would be carried out in the individual tree. For each tree, produced by X, $f\uparrow(t)$ may be computed.

For each nonterminal X : $R(X) \in 2^{B\uparrow(X)}$;

$R(X) := \{f\uparrow(t) \mid t$ *is derivation tree for a sentence produced by* $X\}$

is the set of information records, which really occur in some individual tree, called the real set.

For each nonterminal a description of the sets of really occurring information records is to be computed. Therefore, GFA handles 'descriptions' of sets of information records. The descriptions may be **exact**, i.e. they describe the sets and only the sets of really occuring information records, or they may be **approximative**, i.e. they in general describe a superset of the set $R(X)$ of really occurring information records. As in DFA, approximating exact flow information may be precise enough for some purposes and cheaper to compute. Different kinds of approximations need different kinds of description domains, called **description spaces**. We denote the family of description spaces by $\{D\uparrow^a(X)\}_{X \in NT}$ where "a" denotes the approximation type. As $B\uparrow(X)$ is the domain of information records, so $D\uparrow^a(X)$ is the domain of descriptions.

The description spaces $D\uparrow^a(X)$ are structured, i.e in general they are **semilattices**. Finite description spaces have been always sufficient in the applications we have worked out so far with the methods of grammar flow analysis. Therefore we will restrict ourselves in the following to finite spaces.

The descriptions must be propagated over the **grammar graph**. This is the analogon of propagation of data flow information over the program graph. The grammar graph has a bipartite node set: the set of **nonterminal nodes** and the set of **production nodes**. Flow propagating functions are associated with production nodes of the grammar flow graph. In contrast to propagation functions in DFA

they may have more than one argument. Different edges entering the same nonterminal node transport descriptions stemming from the different alternative productions, i.e. productions with the same left side, for a nonterminal.

In general, the description space must be a join resp. meet semilattice with a additional bottom resp. top element. The semilattice operation ∇ handles the combination of descriptions, obtained from different pathes. A join resp. meet operation ∇_X^a is associated with each nonterminal node in the grammar graph describing how the flow of information from different alternative productions has to be combined. Of course, this depends on the approximation type.

F_P are the **flow propagating** functions, propagating descriptions:

$$F\uparrow_p^a : D\uparrow^a(X_1) \times \cdots \times D\uparrow^a(X_k) \to D\uparrow^a(X_0)$$

The definition of the flow propagation function depends on the type of approximation (cf. chapter 1.3).

The family of description spaces and the family of flow propagation functions together with the grammar define the **flow problem**:

$$FP := (G, \{D\uparrow^a(X)\}_{X \in NT}, \{F\uparrow_p{}^a\}_{p \in P}, \{\nabla_X^a\}_{X \in NT})$$

For convenience we avoid writing the subscript nonterminal and superscript type at ∇ if it is obvious from the context.

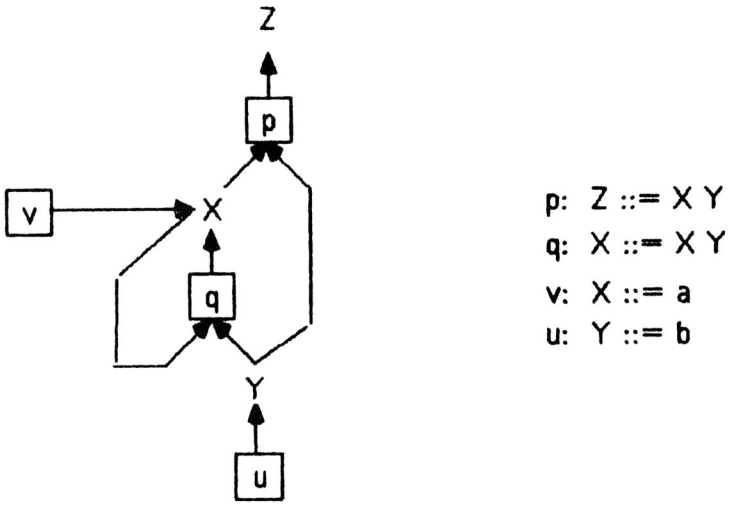

Figure 3

A solution of a flow problem consists of one description for each nonterminal. This family of descriptions is computed as a fixed point solution of the following equational system:

for all $X \in NT$:

$$L(X) \in D\uparrow^a(X);$$

$$L(X) = \nabla^a_{X \atop \{p \in P \mid X=p[0]\}} F\uparrow^a_p(L(X_1),\ldots,L(X_{rank(p)}))$$

2.3. Different types of simulation

2.3.1. Exact simulation

For the exact simulation, i.e. for precomputation of the sets of really occurring information records for each nonterminal, we define the description space as follows:

$$D\uparrow^e(X) := 2^{B\uparrow(X)}$$

Let "e" denote the approximation type "exact" precomputation. The descriptions are themselves the 'real' sets; there is no approximation. In this particular case the space of descriptions, belonging to a nonterminal, is the powerset of the basic set $B\uparrow(X)$. The **bottom** element in the description space is the **empty set** 0, the **compare operation** is set inclusion $\subseteq$, and the **join operation** is **set union** $\cup$.

The **flow propagation** function simply extends the individual propagation function f_p to sets:

$$F\uparrow^e_p(d_1,\ldots,d_k) := \{\, f_p(m_1,\ldots,m_k) \mid m_i \in d_i, \text{for } i=1..k,\ k=rank(p) \}$$

i.e. a flow propagation function applied to a tuple of descriptions forms the set of all results of the propagation function applied to all "combinations" of information records described by the tuple of descriptions.

The **solution** of the "exact" problem is the minimal **fixed point** solution of the following equations:
for all $X \in NT$:

$$L^e(X) = \bigcup_{X=p[0]} F\uparrow^e_p(L^e(X_1),\ldots,L^e(X_{rank(p)}))$$

Each information record $m \in L^e(X)$ characterizes trees, namely such trees t which could be produced by X **and** m is assigned to their roots by $f\uparrow(t)$. $L^e(X)$ induces a partition of the set of subtrees produced by X (cf. figure 4).
In our running example, the $L^e(X)$ are **sets** of ***characteristic graphs*** , in the second example **sets** of *patterns*.

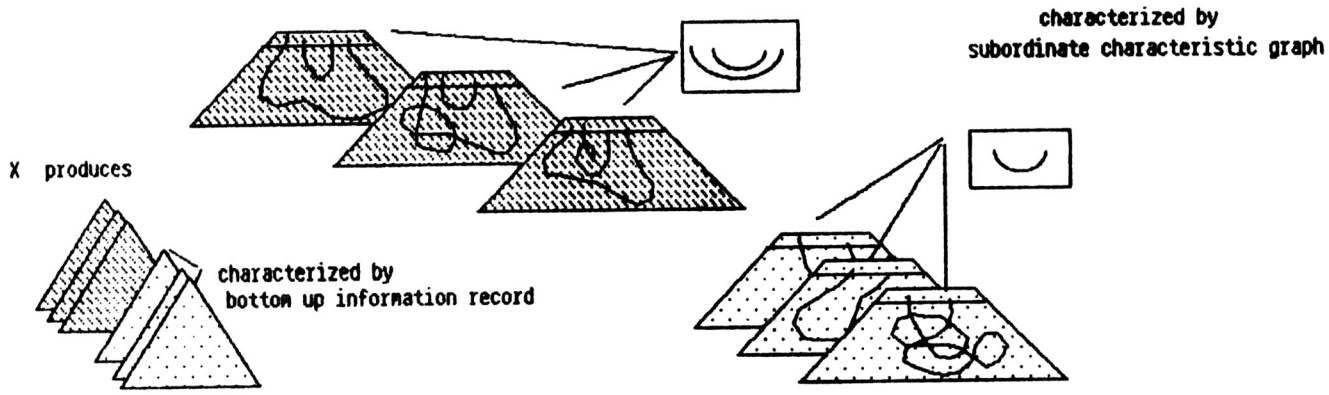

Figure 4

2.3.2. Generation of approximative descriptions

Approximation abstracts from the concrete set of individual information records. By the operation of **interpretation** each description can be mapped back to a set of information records. In general, the descriptions may be arbitrarily chosen assertions about sets of information records. In special cases, as considered in this paper, these descriptions are themselves elements from the basic sets or, at least, elements from the powerset of basic information record sets. In all cases of approximative flow analysis, we look for intervals which will include the really occurring information records. We will now discuss some descriptions and then consider the relationship between abstraction and interpretation.

2.3.2.1. Set of descriptions, abstraction and interpretation

Some elements in a set of information records might "dominate" others. A first try in reducing the size of descriptions keeps only maximal or minimal (incomparable) elements with respect to the order given in the basic sets. Another approximation takes upper/lower bounds of the information records in a set.

Approximation with maximal and minimal incomparable elements leads to 'realistic' results, because the information records, computed this way, are members of the real sets, i.e. they really occur in some trees. Computing upper/lower bounds on the other side may result in information records which might not occur in the real sets.

This leads to different types of descriptions, corresponding to different approximations. 'Exact descriptions' (cf. chapter 1.3.1), which contain just the really occuring information records are very space consuming. Descriptions, which only

contain incomparable information records may be much better. The best descriptions from the viewpoint of space usage are the upper resp. lower bound descriptions, which consist only of a single information record.

We give a summary of approximative description spaces: Let **a** denote the approximation type.

1) descriptions, which contain sets of incomparable elements
   1a) sets of maximal incomparable elements (**a = max**)
   1b) sets of minimal incomparable elements (**a = min**)
2) bounds (descriptions consisting of a single element)
   2a) upper bound (a = **ubd**)
   2b) lower bound (a = **lbd**)

According to the types of approximation, the **basic sets** must fulfill different requirements:

1) partially ordered by some relation $\leqq$
2) semilattice with operation and a special element:
2a) join operation, bottom element
2b) meet operation, top element

The special bottom/top element may be an artificially defined one.

In the *characteristic graph* example the partial ordering is induced by the inclusion of **sets of edges** and therefore join and meet operation are the corresponding set operations. Union of the edge sets corresponds to the superposition (merging) of graphs. Bottom element is the empty graph and top element the full graph, containing one edge from each inherited attribute to each synthesized attribute.

Analogously, the in the *pattern sets* constitute a semilattice.

The **description spaces** are the following:

Let $max\text{-}set\text{-}of(M) := \{m \in M \mid x \in M,\ x \geqq m \rightarrow x = m\}$

$min\text{-}set\text{-}of(M) := \{m \in M \mid x \in M,\ x \leqq m \rightarrow x = m\}$

1a) $D\uparrow^{max}(X) := \{max\text{-}set\text{-}of(M) \mid M \in 2^{B\uparrow(X)}\}$

1b) $D\uparrow^{min}(X) := \{min\text{-}set\text{-}of(M) \mid M \in 2^{B\uparrow(X)}\}$

2) $D\uparrow^{ubd}(X) := D\uparrow^{lbd} := B\uparrow(X)$

In these spaces the following compare relations are defined:

1a) $d_1 \leqq^{max} d_2$ iff for all $m_1 \in d_1$ there exists $m_2 \in d_2 : m_1 \leqq m_2$

1b) $d_1 \leqq^{min} d_2$ iff for all $m_2 \in d_2$ there exists $m_1 \in d_1 : m_1 \leqq m_2$

2a) $d_1 \leqq^{ubd} d_2$ iff $d_1 \leqq d_2$

2b) $d_1 \leqq^{lbd} d_2$ iff $d_1 \leqq d_2$

Remark: $\leqq$ is the relation in the basic sets.

The top resp. bottom elements are:

1a) $bottom^{max} = 0$

1b) $top^{min} = 0$

2a) $bottom^{ubd}$ is the bottom element in the basic set

2b) top^{lbd} is the top element in the basic set

Abstraction and interpretation play a crucial role in **grammar flow analysis** as in data flow analysis.

The **abstraction** maps sets of information records to a description: $\alpha\uparrow_X^a : 2^{B\uparrow(X)} \to D\uparrow^a(X)$

In the above mentioned four cases:

$\alpha\uparrow_X^a(M)$:=

1a) max-set-of(M)

1b) min-set-of(M)

2a) join(M)

2b) meet(M)

Interpretation maps descriptions to sets of information records:

$\gamma\uparrow_X^a : D\uparrow^a(X) \to 2^{B\uparrow(X)}$

$\gamma\uparrow_X^a(d)$:=

1a) $\bigcup_{m' \in d} \{m \leq m' \mid m \in B\uparrow(X)\}$

1b) $\bigcup_{m' \in d} \{m \geq m' \mid m \in B\uparrow(X)\}$

2a) $\{m \leq d \mid m \in B\uparrow(X)\}$

2b) $\{m \geq d \mid m \in B\uparrow(X)\}$

The image of an interpretation is therefore an union of intervals rsp. an interval. In the case of exact flow analysis, a = e, both abstraction and interpretation are the identity.

In general, the interpretation is **not** an inverse of the abstraction. A family $(\alpha\uparrow_X^a, \gamma\uparrow_X^a)_{X \in NT}$ of pairs of abstraction and interpretation is correct iff $\gamma\uparrow_X^a(\alpha\uparrow_X^a(M))) \supseteq M$ for all $M \in 2^{B\uparrow(X)}$, for all $X \in NT$, i.e. interpretation of a description includes the argument set of information records. Of course, in the exact flow analysis "=" holds.

2.3.2.2. Propagation of descriptions

Flow propagation must preserve correctness. We compare two mappings, the mapping in the space of descriptions to the mapping in the space of information records. Abstraction and interpretation describe the relationship between the

original individual propagation functions and the flow propagation functions to be applied in the flow analysis. The individual popagation function maps tuples of information records to information records, whereas the flow propagation function maps tuples of descriptions to descriptions. The relationship can be graphically shown by a kind of commutative diagram, wherein the vertical links symbolize applications of individual propagation functions resp. flow propagation, and the horizontal links symbolize the abstraction resp. interpretation.

$$
\begin{array}{ccc}
(d_1,\ldots,d_k) & \text{-interpretation}\rightarrow & \{\,(m_1,\ldots,m_k) \mid m_i \in \gamma\uparrow^a_{X_i}(d_i)\,,\ i=1..k\,\} \\
\Big| & & \Big| \\
\text{flow propagation} & & \text{multiple individual propagation} \\
\Big| & & \Big| \\
\Big| & & \{f_p(m_1,\ldots,m_k) \mid m_i \in \gamma\uparrow^a_{X_i}(d_i)\,,i=1..k\,\} \\
\Big| & & \Big| \\
d_0 & \text{-interpretation}\rightarrow & \gamma\uparrow^a_{X_0}(d_0)
\end{array}
$$

Propagating a tuple of descriptions by the **flow** propagation function $F\uparrow^a_p$ results in a **description**. The argument descriptions are abstractions of sets of information records. Let us now apply the propagation function f_p to all the **tuples of information records**, which are described by the tuple of descriptions. Let us compare the resulting set to the set of information records, which we would obtain from the result description by interpretation. Of course, the latter set may contain records, we never would compute from some tuple of records. This is just the effect of computing flow information **approximatively**.

On the other hand, we would expect each record, which we can compute from a single tuple, to be also in the interpretation set. There must be no "loss" of records in the computation of the flow propagation function:

$$\gamma\uparrow^a_X(F\uparrow^a_p(d_1,\ldots,d_k)) \supseteq F\uparrow^a_p(\gamma\uparrow^a_X(d_1),\ldots,\gamma\uparrow^a_X(d_k))$$

The lattice operation in the description space, which combines descriptions from different paths (i.e. productions) at the left side nonterminal, must preserve correctness, too:

$$\gamma\uparrow^a_X(d_1 \nabla^a_X d_2) \supseteq \gamma\uparrow^a_X(d_1) \cup \gamma\uparrow^a_X(d_2) \quad (*)$$

A general correctness preserving definition of flow propagation and combination is the following:

Flow propagation (cf. figure 5):

$$F\uparrow^a_p : D\uparrow^a(X_1) \times \cdots \times D\uparrow^a(X_k) \rightarrow D\uparrow^a(X_0)$$

$$F\uparrow^{a}_{p}(d_1,\ldots,d_k) := \alpha\uparrow^{a}_{X}(\ \{\ f_p(m_1,\ldots,m_k) \mid m_i \in \gamma\uparrow^{a}_{X_i}(d_i),\ \textit{for } i=1..k\ \}\ )$$
$$= \alpha\uparrow^{a}_{X} F\uparrow^{e}_{p}(d_1,\ldots,d_k)$$

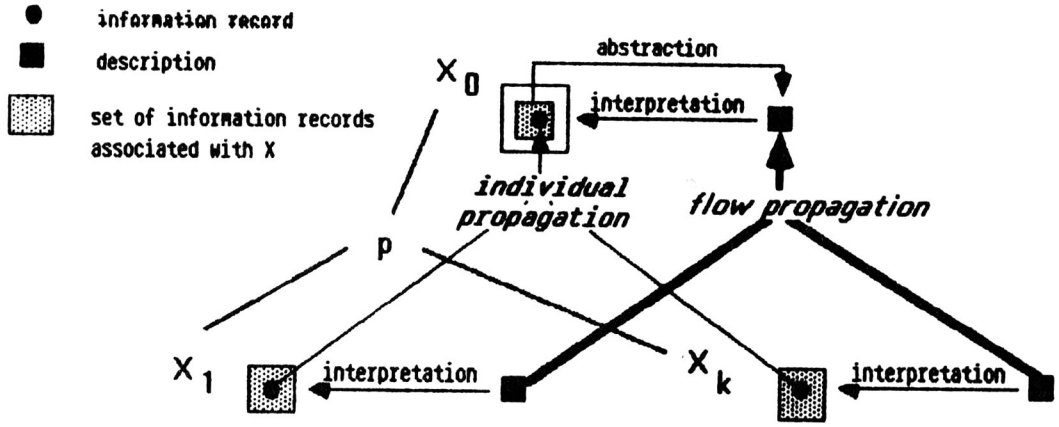

Figure 5

Flow combination (cf. figure 6):

$d_1 \nabla^{a}_{X} d_2 := \alpha\uparrow^{a}_{X}(\gamma\uparrow^{a}_{X}(d_1) \cup \gamma\uparrow^{a}_{X}(d_2))$

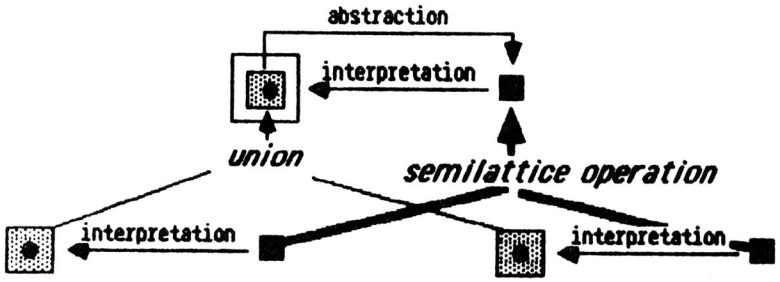

Figure 6

From these definitions and inclusion (*) follows that flow propagation and combination are correctness preserving.

The **special structure** of the description spaces allows to define the flow propagation function and the semilattice operation with respect to the abstraction in a simplified way:

Flow propagation:

$F\uparrow_p^a : D\uparrow^a(X_1) \times \cdots \times D\uparrow^a(X_k) \to D\uparrow^a(X_0)$

$F\uparrow_p^a(d_1,\ldots,d_k) := a\uparrow_X^a(\{ f_p(m_1,\ldots,m_k) \mid m_i \in d_i, \text{ for } i=1..k \})$

if $a \in \{max,min\}$

$F\uparrow_p^a(d_1,\ldots,d_k) := f_p(d_1,\ldots,d_k)$ if $a \in \{ubd,lbd\}$

Flow combination:

$d_1 \nabla_X^a d_2 := a\uparrow_X^a(d_1 \cup d_2)$, if $a \in \{max,min\}$

$d_1 \nabla_X^a d_2 := a\uparrow_X^a(\{d_1, d_2\})$, if $a \in \{lbd,ubd\}$

The result of the approximative analysis is the solution of the equational system which is described in the following:

for all $X \in NT$:

$$L^a(X) = \nabla_{\{p \in P \mid X=p[0]\}}^a F\uparrow_p^a(L^a(X_1),\ldots,L^a(X_{rank(p)}))$$

where $a \in \{max,min,lbd,ubd\}$

Different solutions of the previous systems (cf. figure 7) may be considered. The most precise solution both for the max – and the ubd – approximation is the **least** fixed point, the most precise one for min – and lbd – approximation is the **greatest** fixed point. For a = ubd (upper bound) we compute the **least upper bound** (lub), for a = lbd the **greatest lower bound** (glb).

In our *characteristic graph* example the elements obtained as the least fixed point to approximation **max** are called **covering** graphs in [Deransart,Jourdan,Lorho83] and were also proposed in [Raeihae,Saarinen82]. The least upper bounds of approximation ubd are called **input output graphs** and were introduced by [Kennedy,Warren79]. Of course, lower bounds for the set of characteristic graphs may be computed as greatest fixed point to approximation **min** rsp. **lbd** [Moencke,Wilhelm82].

Let $S^a \in [NT \to \bigcup_{X \in NT} D\uparrow^{ubd}(X)]$ be the least fixed point solution for the approximation a of a grammar flow problem, where $a \in \{ubd,max\}$. A solution S where $S(X) \geq^a S^a(X)$ for all X, is **acceptable**, iff

for all $X \in NT$:

$$S(X) \geq^a \nabla_{\{p \in P \mid X=p[0]\}}^a F\uparrow_p^a(S(X_1),\ldots,S(X_{rank(p)}))$$

An acceptable solution needs not to be a fixed point, but has to be consistent with the propagation function. Assumed a = ubd, an acceptable solution is a single information record for each nonterminal. The information record S(X) may be a very rough approximation compared to $S^{lub}(X)$. Of course, S(X) must approximate $join(\{ f_p(S(X_1),\ldots,S(X_{rank(p)})) \mid p[0]=X\})$

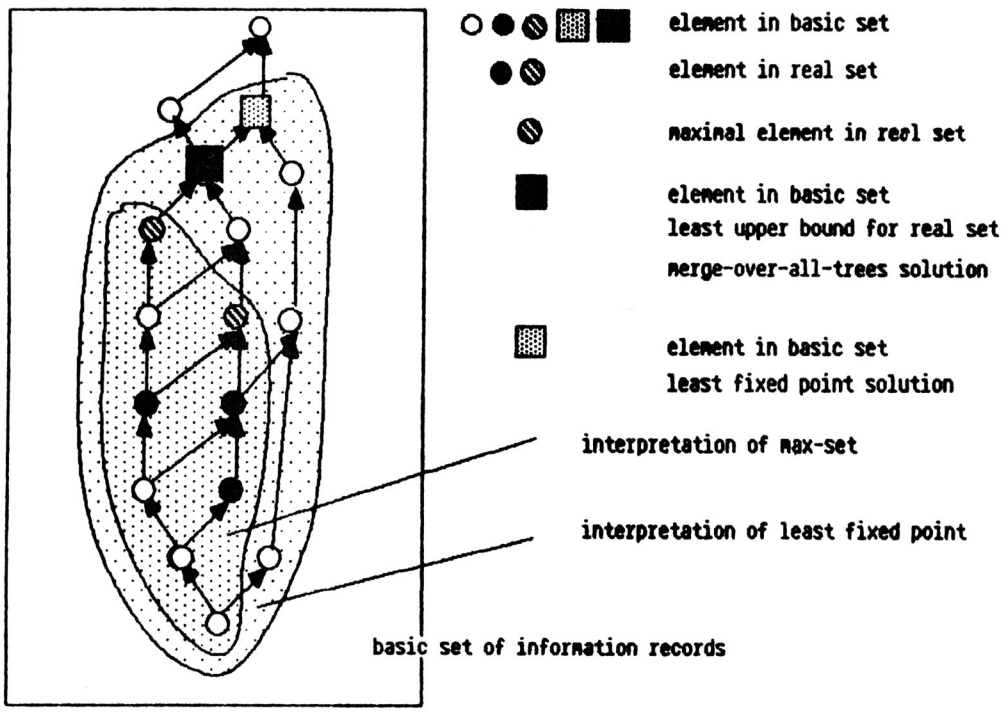

Figure 7

2.3.2.3. Construction of bottom-up assignments from precomputation

From a solution S of a grammar flow problem, a new bottom-up assignment system may be constructed. The knowledge, won by the precomputation, may be used to restrict the domains of information records, i.e. the basic sets, to such record sets, which are result of the fixed point computation, and to define the (individual) propagation function accordingly. On the one hand, this construction is used to get functions, which can be encoded (as described in chapter 2.2). The tables will be used in runtime. As mentioned before, the assignment of approximative information sometimes is sufficent. In the latter case the tables are smaller.

On the other hand, the constructed bottom-up assignment is used as a base for top-down flow analysis. In the case of approximative bottom-up analysis, only the information records contained in the solution S(X) serve as inputs to upper tree fragments (cf. figure 2).

The construction is based on assumptions on the internal structure of the bottom-up solution, i.e. the bottom-up descriptions.

We distinguish the cases a = e,max,ubd. The solution, from which an new bottom-up assignment system is derived needs not to be a fixed point.

$B\uparrow^S(X) := S^a(X)$ for a = e or a = max.

$B\uparrow^S(X) := \{S^{ubd}(X)\}$; for uniformity in case of the "ubd - approximation" we must widen the elements to sets.

The bottom-up propagation functions are constructed using the new basic sets.

a = e:

The propagation functions are restricted to the new domain.

$f_p : B\uparrow^S(X_1) \times \ldots \times B\uparrow^S(X_k) \rightarrow B\uparrow^S(X_0)$

$f_p^S(m_1,\ldots,m_k) := f_p(m_1,\ldots,m_k)$.

a = **max**:

$f_p^S(m_1,\ldots,m_k)$:= m, where $m \in S^{max}(X_0)$ and $m \geq f_p(m_1,\ldots,m_k)$. m is arbitrarily chosen but fixed (cf. figure 8). The reason for this somewhat artifical construction is, that , in general, image $\{f_p(m_1,\ldots,m_k) \mid m_i \in S^{max}(X_i),\ i=1..k\} \leq^{max} S^{max}(X_0)$.

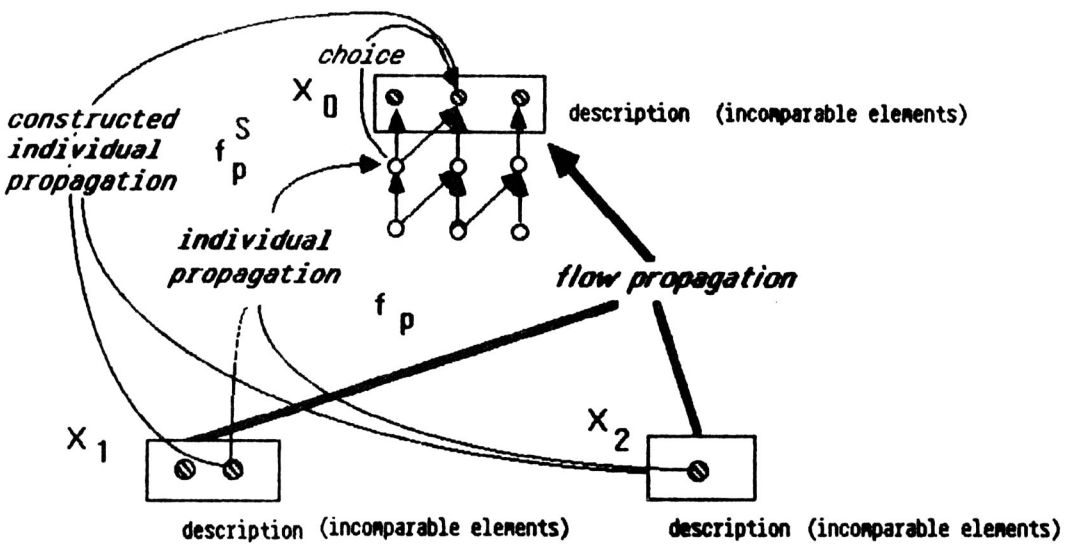

Figure 8

a = ubd:

$f_p^S(m_1,\ldots,m_k) := m$, where $m = S^{ubd}(X)$. This means, that as an approximation an assignment of information records to nodes labeled by X can be

found, which does not depend on subtrees. The constructed propagation function is really a constant. (cf. figure 9).

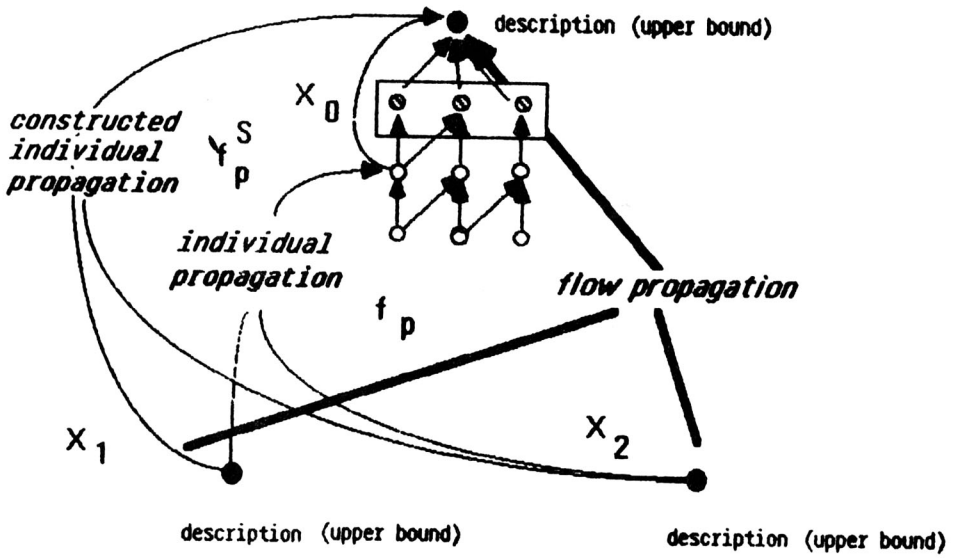

Figure 9

3. Top down grammar flow analysis

3.1. Top down assignment of information records

A top-down assignment system is build up on a bottom-up assignment system. The top-down system specifies how to propagate information top-down in a syntax tree, thereby assuming that bottom-up information has already been assigned to the nodes of the tree. Let S be a bu – assignment system BU-ASYS. S may be an original given system or may be constructed as described in the previous chapter. S is one component of a top-down assignment system.

$$TD\text{-}ASYS = (G,\ S,\ \{B{\downarrow}(X)\}_{X \in NT},\ \{f_{p,i}\}_{p \in P,\ 1 \leq i \leq rank(p)})$$

which consists of a context free grammar G, a bu – assignment system S, a family of basic sets of top-down information records $\{B{\downarrow}(X)\}_{X \in NT}$ and a family of propagation functions $\{f_{p,i}\}_{p \in P,\ 1 \leq i \leq rank(p)}$, where

$$f_{p,i}: B{\downarrow}(X_0)\ x\ B{\uparrow}(X_1)\ x\ \cdots\ x\ B{\uparrow}(X_k) \rightarrow B{\downarrow}(X_i)$$

Information records are propagated top-down to leaves by repeated application of the propagation function $f_{p,i}$.

Function $f\downarrow$ describes the assignment of information records to nodes of the concrete syntax tree:

Let n and n.i be nodes in tree t, $1 \leq i \leq k$, where k is the rank of the operator, which labels n.

$$f\downarrow(t,n.i) := f_{p,i}(\ f\downarrow(t,n), f\uparrow(t/n.1), \ldots, f\uparrow(t/n.k)\ )$$

Having computed an information for node n, we propagate this information downwards to all direct descendents. $f\downarrow(t,\varepsilon)$ may be defined as a special case:

$f\downarrow(t,\varepsilon) := f_{start}(f\uparrow(t))$, where $f\uparrow$ maps the bottom-up information record, which is obtained at the root of a tree produced by the axiom, to a top-down information record.

In the case of top-down assignment we must take care about the bottom-up context, i.e. the information records attached by the bottom-up propagation. Therefore, real sets of top-down information records cannot exist without their bottom-up context. Let R↑(X) be the real set of bottom-up information records for nonterminal X. We define the real sets with respect to bottom-up records and therefore obtain functions:

$$R\downarrow^{R\uparrow(X)} \in [R\uparrow(X) \rightarrow 2^{B\downarrow(X)}]$$

For all $r \in R\uparrow(X)$:

$$R\downarrow^{R\uparrow}(X)(s) := \{f\downarrow(t,n) \mid t \text{ produced by axiom } Z,\ n \in dom\ t,\ t/n \text{ produced by } X,\ \text{and } r = f\uparrow(t/n)\}$$

The real set for a fixed bottom-up information record r is the set of all records that can occur together with s at a node of a syntax tree which belongs to a sentence of the language. Formally seen, a top-down information record is a function from a bottom-up information record, which is input to an upper tree fragment and the upper tree fragment itself. $R\downarrow(X)$ combines the bottom-up record given as argument with all upper tree fragments. Note, that the term "real" is related to the underlying assignment systems. If the bu system is constructed as described in chapter 2.3.2.3, then $R\uparrow(X) = S(X)$ by construction.

3.2. Simulation at generation time with respect to bottom-up assignments

The first case to be discussed is again the exact simulation. Note that, at least in principle, there are two possibilities to use approximations: first, the underlying bottom-up solution may be itself a solution of the approximative problem, second, the top-down computation may be approximative by itself.

We will treat the case of exact simulation and of approximative analysis in an uniform way. Parameter "b" denotes the type of approximation, which is chosen for the top-down analysis independent from the underlying bottom-up solution. In the first step we shall explain the structure of the description spaces (with respect to bottom-up contexts), and in the second step we shall define the propagation functions.

The space of top-down descriptions for X is a set of functions with domain S(X). Let us define this set in two steps: In the first step, a local set of descriptions $D^b\text{-}local(X)$ is defined in the same way as in the case of bottom-up flow analysis. The basic sets are now the top-down basic sets $B{\downarrow}(X)$. The local description spaces do not depend on the bottom-up context.

In the second step, the top-down description space is defined with respect to the bottom-up context. Given an information record as an input to an arbitrarily chosen upper tree fragment a top-down information record would be returned.

$$D{\downarrow}^{S,b}(X) = [\ S(X) \rightarrow D\text{-}local^b(X)\ ].$$

Top-down descriptions are functions from the bu-information records, which are expected to occur at trees produced by X, to descriptions taken from the local top-down description space.

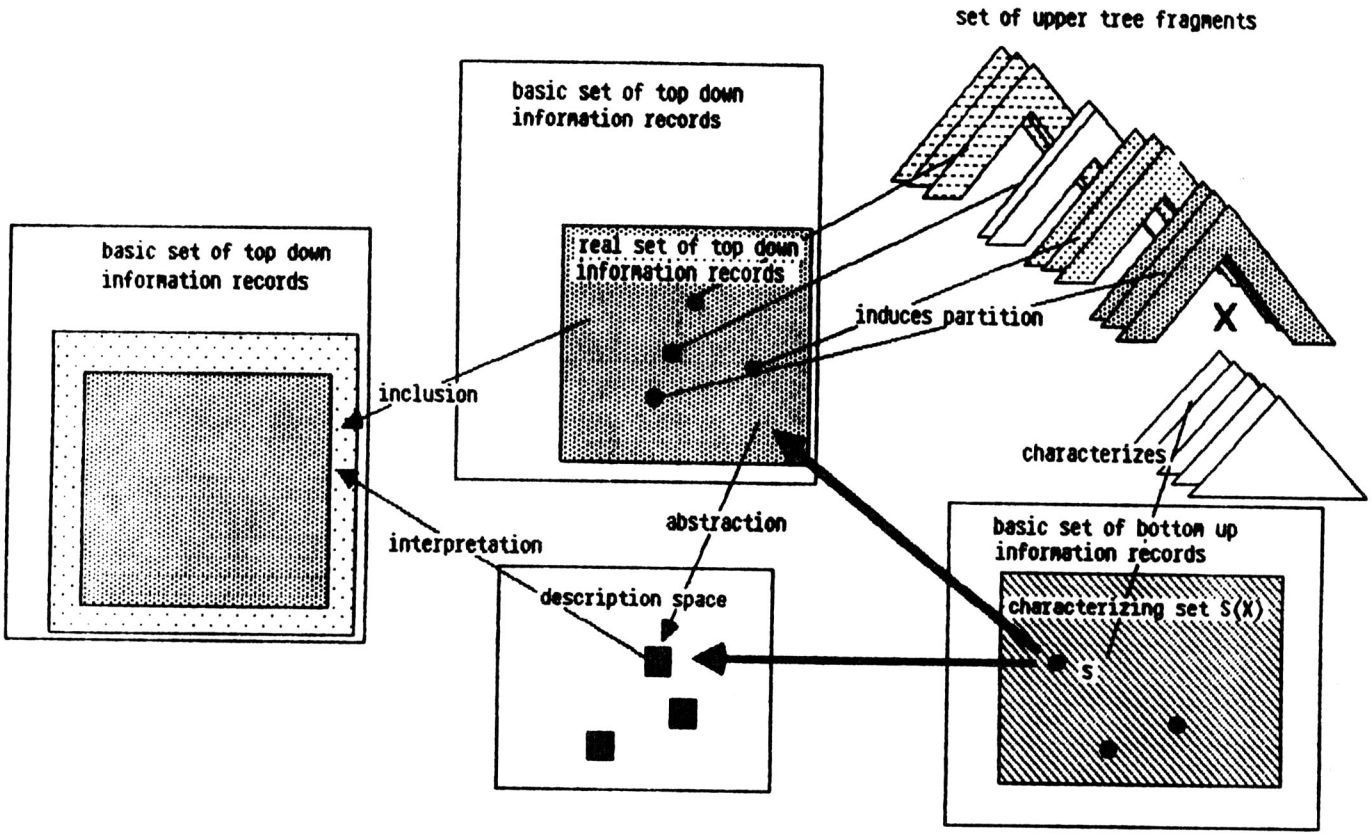

Figure 10

Relations *r-local* in the description space $D\text{-}local^b$ induce relations in the description space:

$r \subseteq D{\downarrow}^{S,b}(X) \times D{\downarrow}^{S,b}(X)$:

$d_1\ r\ d_2$ iff for all $s \in S(X)$: $d_1(s)\ r\text{-}local\ d_2(s)$

Top and **bottom** elements in space $D\text{-}local^b$ induce top and bottom elements in the top-down description space:

For all $s \in S(X)$: $top(X)(s) := top\text{-}local(X)$

For all $s \in S(X)$: $bottom(X)(s) := bottom\text{-}local(X)$

Semilattice operations:

Let $d_1, d_2 \in D{\downarrow}^{S,b}(X)$.

For all $s \in S(X)$:

$(d_1 \nabla_X^{S,b} d_2)\ (s) := d_1(s)\ \nabla_X^b\text{-}local\ d_2(s)$

Considering two sets of upper tree fragments produced at X and fixed bottom-up context s, for each set of upper fragments a description of top-down information records will be obtained. ∇_X^b*-local* combines those

(local) descriptions (with respect to s). $\nabla_X^{S,b}$ combines two S – indexed "arrays" of descriptions.

Interpretation and abstraction are defined locally, too, and are extended to functions. Of course, $\gamma\downarrow_X^{S,b}$ and $\alpha\downarrow_X^{S,b}$ are higher order functions.

$\gamma\downarrow^{S,b} : D\downarrow^{S,b}(X) \rightarrow R\downarrow^{S}(X)$
For all $s \in S(X)$:
$\gamma\downarrow_X^{S,b}(d)(s) := \gamma_X^b\text{-}local(d(s))$

$\alpha\downarrow_X^{S,b} : R\downarrow^{S}(X) \rightarrow D\downarrow^{S,b}(X)$
For all $s \in S(X)$:
$\alpha\downarrow_X^{S,b}(r)(s) := \alpha_X^b\text{-}local(r(s))$

The meaning of operations,functions and objects should become clear from the context.

Now let us illustrate the top-down case with a little example. We have choosen again the *characteristic graphs*. An **superior characteristic graph** represents a assertion about an upper tree fragment. It has edges from synthesized attributes to inherited attributes. The **propagation function** $f_{p,i}$ substitutes superior and subordinate graphs, with exception of the subordinate characteristic graph, which belongs to position i, to the local dependency graph, builds the transitive closure and restricts the result to the attributes at nonterminal X_i. The local description space for each nonterminal is the set of superior characteristic graphs over its synthesized and inherited attributes.

3.3. Approximative top-down propagation of descriptions

Let now $D\downarrow^{S,b}(X)$ denote the top-down description space, assuming S is a solution of the corresponding bottom-up problem. S defines the set of bottom-up contexts.

The flow propagation function maps a top-down description $d \in D\downarrow^{S,b}(X)$, occurring at position 0 of a production to a top-down description d', occuring at position i of the production. Hereby, the context must be taken into account.

$$F\downarrow_{p,i}^{S,b} : D\downarrow^{S,b}(X_0) \rightarrow D\downarrow^{S,b}(X_i)$$

Again, we define the flow propagation function with respect to the special structure of the description spaces by applying the abstraction function to the result of "exact" propagation.

$$F{\downarrow}^{S,b}_{p,I}(d_0) := \alpha{\downarrow}^{S,b}_{X_0}(F{\downarrow}^{S,e}_{p,I}(d_0))$$

The functions $F{\downarrow}^{S,b}_{p,I}$ are monotonic, the description spaces are finite and therefore the fixed point may be computed by iteration. The following formula describes the flow propagation in the case of simulating the exact top-down propagation.

Let $X=p[i]$, be the target position of propagation.

Let $X_j=p[j]$, for $j=1..i-1,i+1..k$.

for all $s \in S(X)$:

$F^{S,e}_{p,I}(d)(s):=$

$\{f_{p,I}(m_0,s_1,\ldots,s,\ldots,s_k) \mid m_0 \in d(f_p(s_1,\ldots,s_{I-1},s,s_{I+1},\ldots,s_k)),$

$s_j \in S(X_j),\ j=1..i-1,i+1..k\ \}$

The result is a function from $S(X_I)$ to the powerset of basic top-down sets. For each bottom-up context $s \in S(X_I)$, there is a set of top-down records. Assumed s being fixed, we have to construct the remaining arguments $s_j, j=1..i-1,i+1..k$ of the tuple, to combine them with s and to apply the individual top-down propagation function. The result is the set of top-down information records, which can occur in context of s. To get the "right" top-down information records from position 0, we have to compute the bottom-up context at this position for the chosen tuple of arguments at position 1..k.

The requirements for correctness are the same as described above: $\gamma{\downarrow}^{S,b}_{X}(F{\downarrow}^{S,b}_{p,I}(d)) \supseteq F{\downarrow}^{S,e}_{p,I}(\gamma{\downarrow}_{X_i}(d))$ and analogously for the meet/join operation.

Now, we have to solve the following system of equations:

$$L(X)= \nabla^{S,b}_{X} \quad F{\downarrow}^{S,b}_{p,I}(L(X_0))$$
$$\{(p,i) \mid X=p[i],\ 1 \le i \le rank(p)\}$$

where $p[0]=X_0$.

In the important special case of "homogeneous" top-down assignment systems (cf. [Moencke85]), the result of a top-down propagation $f_{p,I}$ does not depend on the bottom-up records at position i. In other words, we can substitute an arbitrarily chosen (sub)tree at node n.i in t, without altering any top-down information at n and ancestors of n in t. The top-down information record obtained at a tree node labeled by X depends only on the upper tree fragment. It does not depend on a bottom-up context $s \in S(X)$ (cf. figure 6). In this case, top-down flow analysis can be simplified: all the extensions, made with respect to the bottom-up context are now unnecessary.

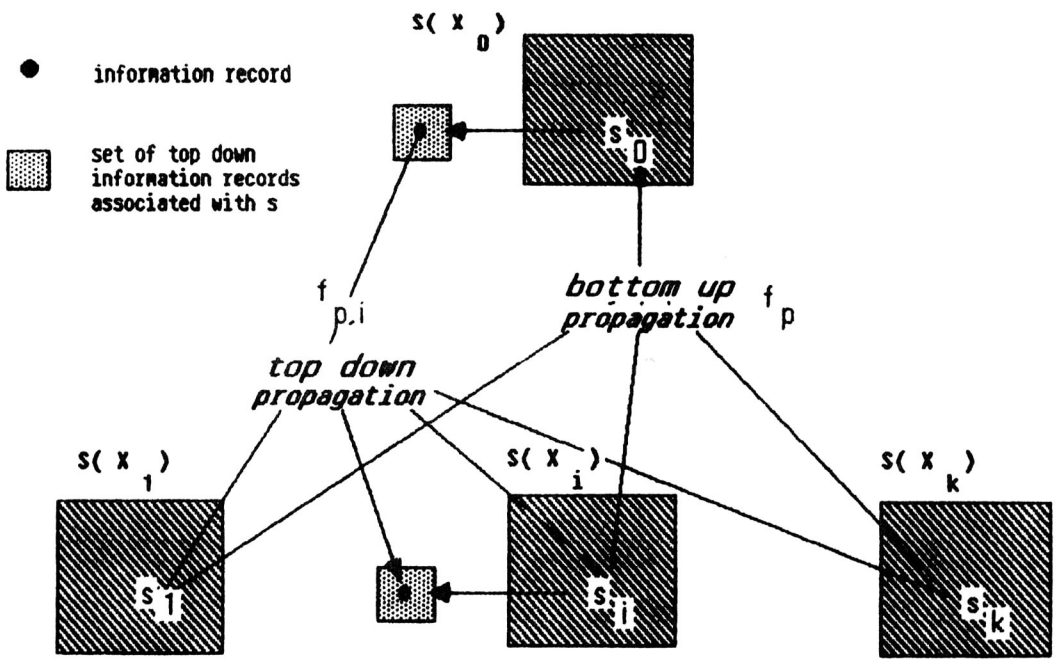

Figure 11

Therefore, top-down analysis handles sets of information records as in the bottom-up scheme, instead sets of functions.

The (top-down) *characteristic graph* assignment is homogenous, because the superior characteristic graph only mirrors attribute dependencies in the upper tree fragment of X and therefore does not depend on the characteristic graph for the subtree, which may be substituted in X.

Other problems, e.g. the problem of computing a top-down *ordered partition of attributes* , are not homogeneous. ("Ordered partitions" are important means to generate reattribute evaluators for arbitrary cycle - free attribute grammars, cf. [LMOW87]).

3.4. A hierarchy of approximations

At first, the bottom-up precomputation leads to a hierarchy of solutions by itself. Solutions are distinguished according to the degree of approximation. We restrict this chapter to the consideration of upper bounds.

As an analogon to the data flow analysis term **merge over all paths** solution we define the merge **over all trees** - solution:

$$S^{mot}(X) := \underset{m \in RT(X)}{join}\ m$$

Let $S^{a}(X)$ denote the least fixed point solutions. The following equalities resp. unequalities hold:

bu - 1) $S^{e}(X) = R(X)$ (exact simulation computes the real sets)

bu - 2) $S^{max}(X) = max\text{-}set\text{-}of(S^{e}(X))$

bu - 3) $S^{lub}(X) \geq (*) \underset{m \in S^{max}(X)}{join}\ m = \underset{m \in S^{e}(X)}{join}\ m = S^{mot}(X)$

Note, that in general the relation at (*) is strict. If the propagation functions f_p are distributive in each argument with respect to the join operation, then ' = ' holds at (*).

The functions for propagating *characteristic graphs* in general are not distributive. Intuitively, the effect of single edges in the argument graphs to the propagation could not be isolated without changing the result. Therefore, an input - output graph may contain edges, which represent no attribute dependency in any syntax tree. Such an edge could not be obtained by merging the graphs in the real set. (Definitions of IO - graphs, found in literature, sometimes neglect the difference between mot and lub - solution, cf. [Reps82]).

In contrast, the bottom up functions for pattern matching are distributive. Intuitively, the matching subpatterns may be propagated separately for each position.

The top-down hierarchy depends on the bottom-up approximation, on which the top-down problem is based. The idea is to construct a new bottom-up assignment system from the results of bottom-up grammar flow analysis as described in chapter 2.3.2.3.

Let $S^{a,b}(X)$ denote the result of the top-down precomputation, based on $S^{a}(X)$, the bottom-up solution. a denotes the bottom-up approximation, b the top-down approximation type. In the following, we assume that both the max and ubd - solution are computed as least fixpoints.

Now, let us compare $S^{e,e}(X)(s)$ with $S^{max,e}(X)(s)$ for $s \in S^{max}(X)$. This means, we change the bu - approximation (from "exact" to "maximal elements"). Surprisingly, examples can be constructed, where for some bu records s, $s \in S^{max}(X) \subseteq S^{e}(X)$, $S^{max,e}(X)(s)$ may contain information records, which are not in $S^{e,e}(X)(s)$. The reason is, that the approximating bu propagation functions f_p^S (see above) associate bu information records with tree contexts, in which the records would not really occur by exact propagation (cf. [Moencke85]). Instead of a concatenation of functions f_p a concatenation of f_p^S is applied to s. The td - functions $f_{p,i}$ take these records as arguments and compute also records, which we would never find in their

contexts otherwise.

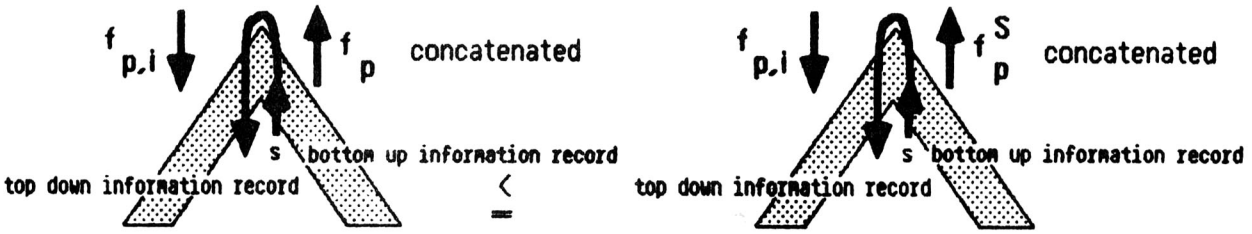

Figure 12

So, we conclude that (max,e) - approximation is less precise than (e,e) approximation not only in its bu - component (cf. equation bu - 2)), but also in the td - component.

Basing the top-down analysis on the bottom-up **lub** solution, we get $S^{lub,\bullet}(X)$. Note, that each X has one single information record as (bottom-up) least upper bound. Because of the monotony of both the bottom-up and the top-down propagation functions, for all $s \in S^{max}(X)$, and for all $m \in S^{max,\bullet}(X)(s)$, we find at least one $m' \in S^{lub,\bullet}(X)$, *where* $m' \geq m$. Of course, the same holds for $s \in S^{e}(X)$ and $m \in S^{e,\bullet}(X)(s)$. Figure 13 shows the "exact" top down analysis based on different bottom-up approximations. Note, that both the bottom-up record chosen as argument and the constructed (approximative) propagation function f_p^S determine the top-down approximation.

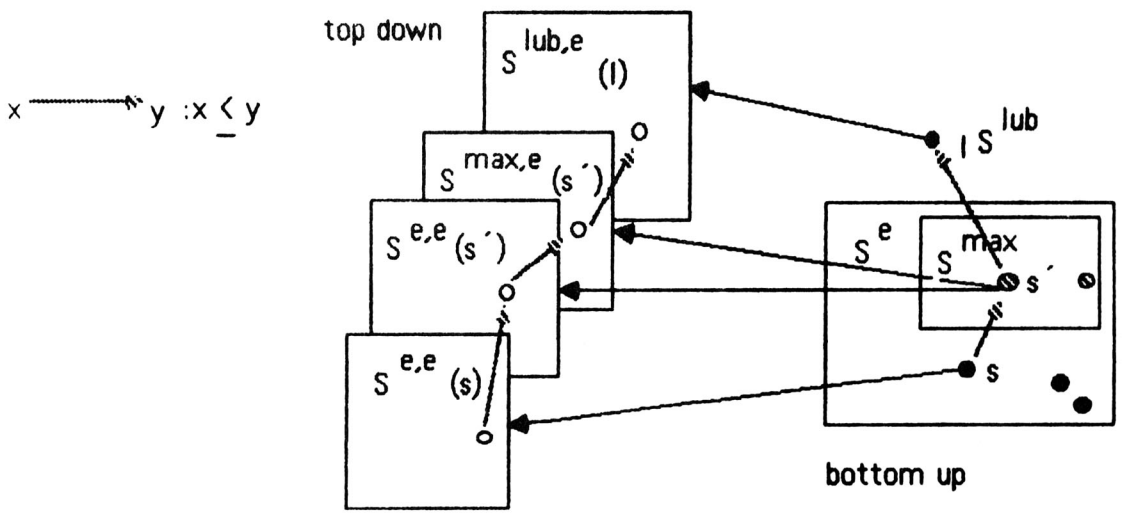

Figure 13

Furthermore we can change the td - approximation type **b** while leaving the bu - approximation type **unchanged**. This would result in a local hierarchy built upon the underlying bu - approximation

For $a \in \{e, max, lub\}$:

For all $s \in S^a(X)$:

td - 1) $S^{a,e}(X)(s) = R^S(X)(s)$

td - 2) $S^{a,e}(X)(s) = max\_set(S^{a,max}(X)(s))$ (cf. Figure 14: relation (2))

td - 3) $S^{a,lub}(X)(s) \geq \underset{m \in S^{a,max}(X)(s)}{join}$ (cf. Figure 14: relation (3))

In particular, the solution $S^{lub,lub}(X)$ may be useful and easy to compute.

In general, for comparison of two approximations (a,b) and (u,v), where (a,b) is more precise than (u,v) (cf. figure 11), holds:

For all $s \in S^a(X)$, for all $m \in S^{a,b}(X)(s)$ there is a $s' \in S^u(X)$, where ($s' \geq s$ and there is a $m' \in S^{u,v}(X)(s')$, where $m' \geq m$).

(cf. Figure 14 : relation $\geq$).

Figure 14 shows the hierarchy of approximations.

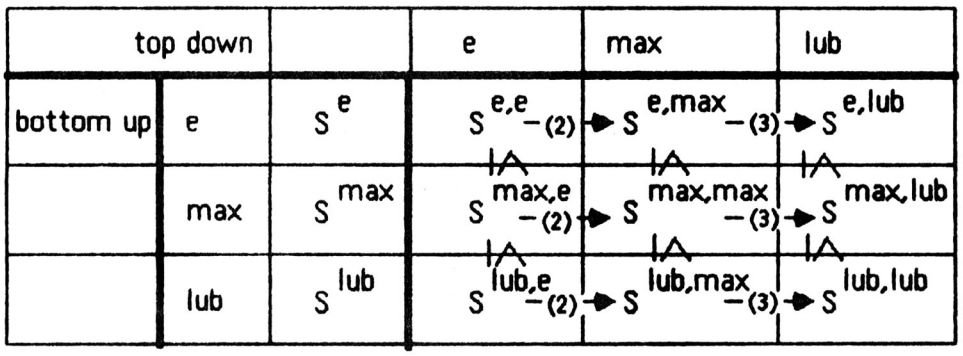

Figure 14

In the *characteristic graph* example the solution $S^{lub,lub}(X)$ gives the top-down counterparts of the input - output graphs.

4. Implementation

The solution of the equational systems is computed by iteration over the grammar graph. The bu - grammar graph, needed for implementation is slightly different to the grammar graph mentioned in chapter 2.2. It has the productions as nodes and there is an edge labeled by i from p_1 to p_2 iff $p_1[0]=X=p_2[i]$ for some i, $1 \leq i \leq rank(p)$. The td - grammar graph differs

from the bu - graph only because of the inverted edges. According to data flow techniques the graph was divided in strongly dependent components and the components are partially ordered. (This was also proposed in [Chebotar81] for the computation of subordinate characteristic graphs).

The information records, which have been computed in the iteration process, were encoded with respect to so called coding references. A coding reference is a set of nonterminals with identical basic sets. In the extreme case, records are encoded with respect to single nonterminals.

To each nonterminal a buffer is assigned, which collects the encodings for information records that are already computed for it. The buffers are collectors for information, obtained from a producing production node, and supply codes to production nodes acting as consumers.

In the bottom-up case p[i], $1 \leq i \leq rank(p)$, are suppliers to consumer p, and p[0] is the collector for p. In the top-down case the roles are interchanged: p[0] acts as the supplier to p, and p[i], $1 \leq i \leq rank(p)$ are the collectors. Note, that the bottom-up information at p[i] is fixed in the top-down computation process, and therefore we neglect it.

Each production has bookkeeping information about what it has already consumed from its supplying buffers. Technical, the bookkeeping is realized by pointers into the buffers.

The iteration follows the order of the components. Within each component iteration steps are carried out for each production in the component, until there is no new information in the buffers for any production node in this component. Each iteration step consists of a sequence of propagation steps.

One propagation step takes a tuple of encodings, decodes this to information records, applies the propagation function and encodes the result, by comparing it with information records already contained at the coding reference.

$$propagation\_step\ (m_1,\ldots,m_k) =$$
$$code\&entry_{reference\_of(X_0)}\ (f_p\ (decode_{reference\_of(X_1)}\ (m_1),\ldots,$$
$$decode_{reference\_of(X_k)}(m_k).$$

The bookkeeping is important, because it avoids double computations. Let us explain this in the bottom-up case:

One bottom-up iteration step (for a production) reads the buffers and enables the propagation steps for such argument tuples, where at least one of the argument components is new.

$$iteration\_step = \{propagation\_step(m_1,\ldots,m_k) \mid (m_1,\ldots,m_k) \in \bigcup_{j=1..rank(p)} (old\_codes(X_1) \times \ldots \times old\_codes(X_{j-1}) \times new\_codes(X_j) \times codes(X_{j+1}) \times \ldots \times codes(X_k)) \}$$

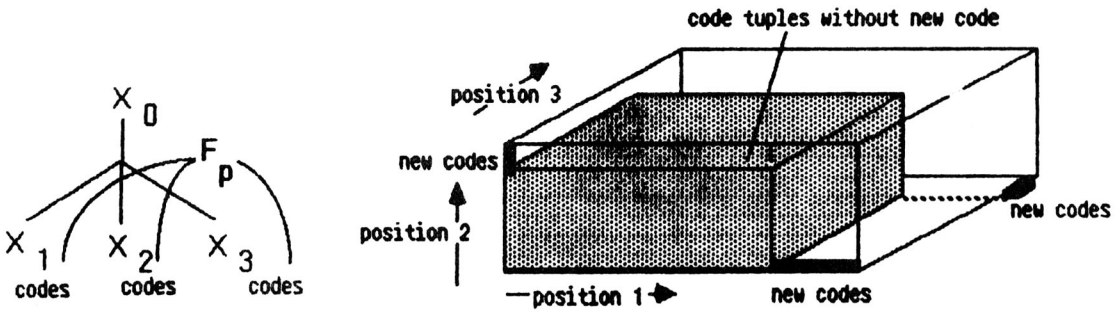

Figure 15

Of course, this does not change the complexity (here only considered for the exact simulation):

Let S be the solution of the bottom-up analysis.

$$C\uparrow(p) := \prod_{j=1..rank(p)} |S(X_j)|$$

propagation steps must be carried out at production p.

Top down propagation is a little bit more complicated, because of the functional dependency on bottom-up context.

Therefore, top-down buffers are associated with information records found in the bottom solutions.

Let S be the solution of bottom-up analysis. Let $S^{S,b}$ be the solution of the corresponding top-down analysis.

A top-down iteration step for production p computes $f_p(s_1,\ldots,s_k)$ for each argument tuple, where $s_i \in S(X_i)$, $i=1..k$. Note, that this computation may be done by table lookup in the previous computed bottom-up propagation table.

Let now $s_0 \in S(X_0)$ denote the result. Subsequently the buffer, associated with

s_0 is consulted:

If there is a new top-down record, say m_0, entered in the buffer since the last iteration step for p, these record is combined with the (actual) argument tuple and for all positions, i=1..k, $f_{p,i}(m_0,s_1,\ldots,s_k)$ is computed. The result is entered in the buffer associated with s_i at position i.

Of course, the top-down iteration step for p is enabled only if there is a buffer for some $s \in S(X_0)$, which has obtained a new top-down record since the last iteration step. Let s be such a bottom-up record. Considering all bottom-up argument tuples is not efficient, because only tuples which are mapped to s have to be combined with top-down records from the buffer associated with s. An argument tuple, which is not mapped to s, provides another context for the computation of top-down records. Top down records in the buffer at s are only valid in the context, which is mapped to s. Therefore, an improvement would be to restrict the top-down computation to (bottom-up) argument tuples, which are mapped to s. On the other hand, this would cost additional space for holding the inverse f_p^{-1}.

In the following, we assume, that the top-down records are independent on the bottom-up context, as is guaranteed in the "homogeneous" case.

Let $C{\downarrow}(X) := \max_{\{s_0 \in S(X)\}} |S^{s,b}(X)(s_0)|$

the worst case number of top-down information records, obtained as input at position 0 of production p, where $p[0]=X_0$.

$C{\downarrow}(p) := C{\uparrow}(p) * C{\downarrow}(X)$

is the number of coupled applications ($f_{p,i}$), i=1..k

There are two expensive operations, dealing with the information records:

1.) propagation operation

2.) compare operations (in 'code&entry').

In general there is a small number of records computed by a large number of propagation operations. Considering the output stream of records, generated by a sequence of propagation steps there are long subsequences of identical records in practice. So, (re)ordering the records by move to front can be very helpful for decreasing the number of compare operations.

5. Special features

For simplicity we restrict this section to the bottom-up flow problem.

5.1. Horizontal orientation

Horizontal orientation means, that the propagation step at production p is divided into substeps, one step for each child position of production p. This can only be done if the propagation function is separable into partial (transition) functions, as follows:

$f_p(m_1,\ldots,m_k) = f^k(\ldots(f^2(f^1(f^0(),m_1),m_2)\ldots)$ where

$f^0 : \rightarrow B\uparrow(X_0)$

$f^j : B\uparrow(X_0) \times B\uparrow(X_j) \rightarrow B\uparrow(X_0)$

Then, the intermediate results may be encoded, too, and if we find an intermediate result, which was already computed, the following transition steps could be avoided. In other words, we use the structure of the production local transition diagram, which is now seen as a directed acyclic graph rather than a tree. The scheme for one iteration step is very similar to the dynamic programming method, i.e. the basic idea is to tabulate the intermediate states:

1)

propagating a tuple of arguments

The single step, described now, computes $f_p(m_1,\ldots,m_{j-1},m_j,m_{j+1},\ldots,m_k)$, where only $m_j \in B\uparrow(X_j)$ is an information record, which was obtained in the buffer at X since the last iteration step. Due to the structure of the propagation function, transitions by $m_1,\ldots,m_{j-1}$ are already executed in previous iteration steps and the results stored in sets of intermediate states $I(0),I(1),\ldots,I(j)$.

Furthermore, for each element in $I(i),i=j..k$ transitions under m_j are computed.

Starting from each intermediate result found in $I(j-1)$ transitions under $m_j,\ldots,m_k$ must be computed.

Transitions under m_i, $i=j..k$ sometimes will reach an intermediate result, which was previouly computed (cf. figure 16). Of course, transitions under the the remaining chain $m_{i+1},m_{i+2},\ldots,m_k$ must not be computed repeatedly.

2)

horizontal propagation step for start position j

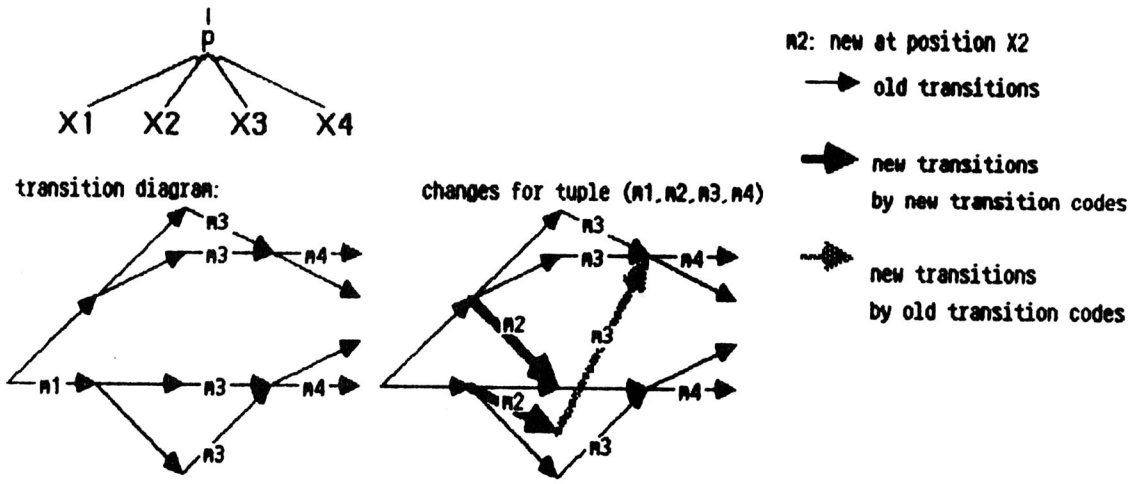

Figure 16

A list of sets of transition codes, *codes*(1),...,*codes*(k) contains the sets of information records, for which transitions have been executed in previous iteration steps.

New_codes(j) contains the information records, computed since the last iteration step for p at X_j. For all $m \in new\_codes(j)$ we have to start with the records in I(j - 1) and successively to carry out the transition sequence under all transition elements, beginning with transitions under new_codes(j) from I(j - 1) at position j and continuing with transitions under *codes*(j + 1),...,*codes*(k). In the transition step at position i, i = j..k, step we may find some intermediate results, which are already computed and available in I(i).

3)

horicontal step

The computation, described above in 2) is executed for each start position j from k = rank(p) down to 1. In this way, we find in step j intermediate results, which may be computed by step i, i>j, i.e. for a start position at the right side of the actual one.

Presuming that the cost of a partial propagation operation f^1 is less than a 1/k of the cost of f_p this method always is preferable.

5.2. Item set construction

Frequently, the information records themselves are sets, e.g. sets of *matching tree patterns.* or sets of edges in a *characteristic graph.* Abstracting from a individual task we call the elements of these sets 'items'. The precomputation gives sets of items, one set for each nonterminal X. The items all together characterize at least one tree, produced by the nonterminal X. The **greatest lower bound solution** contains items which characterize each tree, i.e. we compute the greatest common itemset. An item, found in glb will occur at the root of each subtree (i.e we conclude on the presence). The **least upper bound solution** contains at least such items which will characterize at least one tree, but there may be more items in this set. An item, which is missed in lub, will never occur at the root of some tree (i.e we conclude on the absence of items).

5.3. Product frameworks and functional dependencies

In general grammar flow analysis may be applied to different tasks, underlaying the same grammar. Tasks may be obtained from considering different problems or by dividing one problem into subproblems. Having computed separately the results by exact simulation for two different tasks (w.l.o.g. bottom-up assignments) we often ask for the simultaneous occurrence of information records from both tasks. Building the cartesian product $L_1(X)$ x $L_2(X)$ may be not sufficiently precise, because we could not conclude that for all pairs (m_1,m_2) from this product there is a tree produced by X, having the pair assigned to its root. The a posteriori building of cartesian products is only an approximative solution. Defining a product assignment from both tasks and computing the results for this product would give us the precise solution $(L_1 + L_2)(X) \subseteq L_1(X)$ x $L_2(X)$. A thorough investigation of the product solution may reveal inherent functional dependencies or the orthogonality of the tasks. In our example, we may be interested in the dependency of patterns and characteristic graphs: Given the patterns, matching t, will they determine the characteristic graph t? Of course, the products may be multidimensional. Then, analysis of functional dependencies often will show that the information records assigned for some tasks act as keys for the remaining tasks.

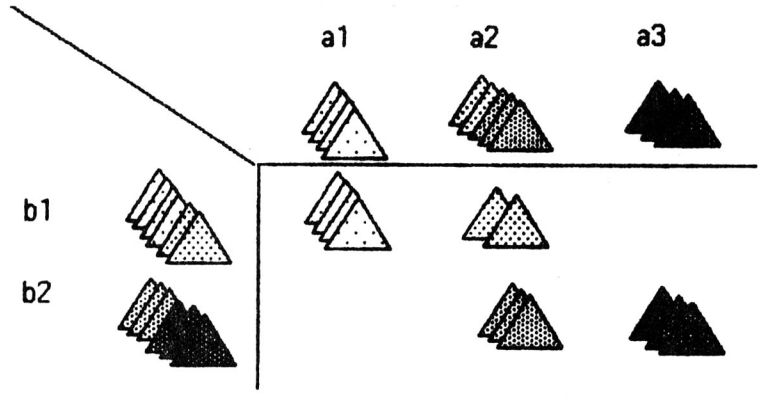

Figure 17

$\{(a_1,b_1), (a_2,b_1), (a_2,b_2), (a_3,b_2)\}$

5.4. Use of abstract syntax trees

This requires only slight modifications with respect to **chains** of nonterminals.

We say nonterminal X is in **substitution relation sub** to nonterminal Y if there is a production $X ::= Y$. Let sub^* be the symmetric reflexive and transitive closure of sub. We must be sure that nonterminals, which are in the same substitution class, posses the same basic sets. Then, the coding references may be the classes. The equational system must be modified with respect to the chains of nonterminals.

bottom-up system of equations:

for all $X \in NT$:

$$L^a(X) = \nabla^a_X \underset{X \to^* p[0]}{} F\uparrow^a_p(L^a(X_1),\ldots,L^a(X_{rank(p)}))$$

top-down system of equations:

for *all* $X \in NT$:

$$L^{S,b}(X) = \nabla^{S,b}_X \underset{\{(p,i)\,|\,p[i] \to^* X\ ,\ 1 \leq i \leq rank(p)\}}{} F\downarrow^{S,b}_{p,i}(L^{S,b}(X))$$

6. Applications

The scheme for grammar flow analysis was implemented and has been applied for a lot of different problems. All these different tasks had been implemented in a uniform way, parameterizing the flow analysis scheme with task specific domains and propagation functions. Despite the theoretical worst case time complexities of all these tasks, which are exponential or, in the case of covering, double exponential in the size of the specification, the

generation time measured so far is reasonable or was at least reduced to reasonable amount using the maximal/minimal incomparable approximative flow computation. Of course, computing bounds decreases the generation time essentially. Let us now give a short summary about these tasks.

6.1. Bottom up characteristic graphs

We have gained experience with different kinds of approximations for characteristic graphs. The observations, made by Raeihae and Saarinen, and the INRIA - group [Deransart,Jourdan,Lorho] could be restated. The number of characteristic graphs in general, and the number of maximal incomparable graphs is small. Nevertheless, as we have seen in student projects, the "personal style" of specifying attributed grammars plays an important role.

6.2. Bottom-up and top-down identity classes

Two attributes of a nonterminal are in an **identity** class iff their values are known to be identical in all instances of X. This information is used to estimate the effect of structural changes in syntax trees. It helps to construct dynamically storage classes of identical attribute instances. (cf. [Raeihae82]).

6.3. Top down ordered partitions

Ordered partitions (cf. [Farrow83]) divide the set of attributes associated with a nonterminal in w.l.o.g. n pairs of corresponding attribute groups. Each pair consists of a group of inherited attributes and a group of synthesized attributes, respectively. The i.th inherited group contains the attributes, whose instances are simultaneously available in the i.th **visit** of the subtree at the node marked by X, while the corresponding synthesized group contains the attributes, whose values are available to the upper tree fragment of the subtree *after* the i.th visit. Unfortunately, in general more than one ordered partition for each nonterminal is needed. The choice of the ordered partition at node n.i is then made dynamically, i.e. at attribute evaluation time one partition is selected from the precomputed set of ordered partitions. The choice depends on the context, which is represented in the ordered partition already computed for (the father) n and the tuple of bottom-up characteristic graphs associated with nodes $n.1,\ldots,n.k$. Practical experience has shown, that the solution $L^{lub,\theta}(X)$, which is based on the input - output

graphs, is sufficient for generation of attribute evaluators [LMOW87].

6.4. Pattern matching sets

Patterns characterize the input and output parts of transformation rules. A transformation rule is (structural) applicable, if its input part **matches** a subtree. The output part specifies the structural change of the tree, especially introducing new nodes, rearranging subtrees, deleting and duplicating subtrees. Flow analysis computes for each nonterminal the sets of patterns and subpatterns which together match at least one tree that can be derived from this nonterminal. Therefore, the precomputed sets of patterns induce a partition of trees in equivalence classes. An efficient pattern matching automaton is generated [Moencke87].

6.5. Sets of covering reduction alternatives

Covering reduction alternatives characterize how trees can be reduced completely by tree transformations, which are executed in a bottom-up strategy. There are a lot of possible applications: Code generation by tree reduction is only feasible, if **syntactic blocking** can be excluded, in other words, complete covering of the given syntax tree by tree transformation rules can be guaranteed. In principle, each transfer from one tree representation to another requires the complete covering, and so a generation time verification rsp. falsification is needed. Grammar flow analysis based on the pattern matching framework mentioned above can do this kind of consistency check [Moencke87].

Acknowledgements: I thank Reinhard Wilhelm, Reinhold Heckmann and Peter Lipps for numerous helpful comments on earlier drafts of this paper and the members of the OPTRAN working group P. Badt, J. Boerstler, M. Olk, S. Pistorius, H. Tittelbach, P. Raber, Th. Rauber, B. Weisgerber for implementation and gaining practical experience from examples.

References:

[Cousot,Cousot77]
Cousot P., Cousot R.
Systematic Design of Program Analysis Framework

6th ACM POPL, 1977

[Chebotar81]
Chebotar K.S.
Some Modifications of Knuth's Algorithm for Verifying Cyclicity of Attribute Grammars
Programming and Computer Software 7, 1 (pp 58 - 61), 1981

[Deransart,Jourdan,Lorho83]
Deransart P., Jourdan M., Lorho B.
Speeding up Circularity Tests for Attribute Grammars
Report RR - 211, INRIA, Roquencourt, 1983

[Farrow83]
Farrow R.
Covers of Attribute Grammars and Sub - Protocol Attribute Evaluators
Comp. Sci. Dept., Columbia University, New York, 1983

[Kennedy,Warren79]
Kennedy K., Warren S.K.
Automatic Generation of Efficient Evaluators for Attribute Grammars
3rd ACM POPL, Atlanta, 1979

[LMOW87]
Lipps P., Moencke U., Olk M., Wilhelm R.
Attribute reevaluation in OPTRAN
ESPRIT Prospectra Report S.1.3 - R - 4.1, Saarbruecken 1987 [Moencke87]
Moencke U.
Simulating Automata for Weigthed Tree Reductions
ESPRIT Prospectra Report S.1.6 - R - 5.0, Saarbruecken 1987

[Moencke85]
Moencke U.
Generierung von Systemen zur Transformation attributierter Operatorbaeume,
Komponenten des Systems und Mechanismen der Generierung,
Ph.D.Thesis, Saarbruecken, 1985

[Moencke,Wilhelm82]
Moencke U., Wilhelm R.
Iterative algorithms on grammar graphs,
in Proc. 8th Conference on Graphtheoretic Concepts
in Computer Science, ed. H. Goettler, pp. 177 - 194, Hanser - Verlag,
1982.

[Raihae81]
Raeihae K.J.
A Space Management Technique for Multi - Pass Attribute Evaluators
Dept. of Comp. Sc., University of Helsinki, Finland, 1981

[Raeihae,Saarinen82]
Raeihae K.J., Saarinen M.
Testing Attribute Grammars for Circularity
Acta Informatica 17 (pp.185 - 192) , 1982

[Reps82]
Reps T.
Generating Language based Environments
PhD thesis, Dept of Computer Sc., Cornell University, 1982