A grammar-based approach towards unifying hierarchical data models

Citation for published version (APA):

DOI:
10.1137/S0097539790188168

Document status and date:
Published: 01/01/1994

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 06. May. 2019
A GRAMMAR-BASED APPROACH TOWARDS UNIFYING HIERARCHICAL DATA MODELS*

MARC GYSSENS†, JAN PAREDAENS‡, AND DIRK VAN GUCHT§

Abstract. A simple model for representing the hierarchical structure of information is proposed. This model, called the grammatical model, is based on trees that are generated by grammars; the grammars describe the hierarchy of the information represented by the trees. Two methods for querying in this data model are given. The first, called the grammatical algebra, is based on a set of primitive grammar-oriented operators, the second, called the grammatical calculus, on local transformations on the trees. The semantics of both is formally defined. Decidability issues regarding the grammatical calculus are investigated. Finally, the two querying methods are proved to be equally expressive.

Key words. information base, grammars, trees, transformations, algebra, calculus

AMS subject classification. 68P15

1. Introduction. Until the mid-1980s much attention was paid to the relational database model (see, e.g., [17], [18], and [21]). We were intrigued by its simplicity, both for modeling and manipulating data. Recently, however, we became aware of its drawbacks when trying to model data applications beyond the traditional business-oriented applications, such as CAD-CAM, office automation, and text-oriented and multimedia databases. Therefore, a great number of data models have been proposed as a possible successor of the relational model.

Semantic data models, such as ER [7], FDM [19], SDM [12], Format [13], and IFO [2], provide a rich set of design tools for representing the complex interrelationships of data. These tools are typically variants of familiar aggregation, generalization, and set-formation constructs. Although query languages have been defined for some semantic data models, their main purpose is to provide database design tools that are more powerful than the modeling tools of the relational model. The logic-based models, such as Datalog [21], LDM [16], and LDL [4], zero in on the limited expressiveness of data manipulation languages of the relational model, i.e., they generalize the relational calculus to express queries that can be specified recursively. Finally, there are the relational extensions of the standard relational model, such as RM/T [8] and the nested relational model [9], [14], [20]. These models try to strike a balance between the elegance of the relational model and the expressiveness of semantic data models. In other words, they are not as rich as the semantic models in their modeling power, but they provide simple yet powerful extensions of the relational model.

Although there exist significant differences between all these models, they share the property that they recognize as the most fundamental characteristic of data its hierarchical structure. On the other hand, however, it is not quite clear whether they can effectively model all data applications which exhibit a hierarchical nature. A good example are textbases [11], which in addition to having a hierarchical structure, are constructed out of rules that follow a grammatical structure. Grammatical structures are also implicitly present in, e.g., VERSO [1], [5], a variation of the nested relational model, where at some level data are structured as regular expressions.

It is the intention of this paper to use a simple and a well-known model as a unifying skeleton to describe the hierarchy in an information base as well as the grammatical structure

---

*Received by the editors September 26, 1990; accepted for publication (in revised form) June 29, 1993.
†Department WNI, University of Limburg (LUC), B-3590 Diepenbeek, Belgium (gyssens@charlie.luc.ac.be).
‡Department of Math and Computer Science, University of Antwerp (UIA), B-2610 Antwerpen, Belgium.
§Computer Science Department, Indiana University, Bloomington, Indiana 47405-4101.
of texts. This model has been presented informally in [11]. It is called the grammatical model and is based on grammars introduced some 30 years ago to study the syntax of programming and formal languages (see, e.g., [10]). The grammatical model will benefit greatly from the clear understanding of the grammars and from their major importance for computer science.

It seems natural to formalize a hierarchical structure by a tree. Therefore, we represent information as a tree which can be generated by a grammar. Each leaf of the tree represents an object, and the internal nodes represent the relationship between the objects. The grammar specifies the scheme, i.e., the overall structure between these relationships. In this way, the information about two employees (one manager and one worker) will be represented by the tree in Fig. 1.

![Fig. 1. Tree representing information about employees.](image)

This tree is generated by a grammar with the following productions:

\[
\begin{align*}
P_1 : \langle \text{Emps} \rangle & \rightarrow \langle \text{Emp} \rangle \langle \text{Emps} \rangle \\
P_2 : \langle \text{Emp} \rangle & \rightarrow \langle \text{Name} \rangle \langle \text{Sal} \rangle \langle \text{Type} \rangle \\
P_3 : \langle \text{Type} \rangle & \rightarrow \langle \text{Man} \rangle \\
P_4 : \langle \text{Type} \rangle & \rightarrow \langle \text{Work} \rangle \\
P_5 : \langle \text{Man} \rangle & \rightarrow \langle \text{Dep} \rangle \\
P_6 : \langle \text{Work} \rangle & \rightarrow \langle \text{Level} \rangle \\
P_7 : \langle \text{Sal} \rangle & \rightarrow \$\langle \text{Am} \rangle
\end{align*}
\]

Notice that production \( P_1 \) specifies information as a list, which is used here to model a set. Production \( P_2 \) specifies an employee as the aggregation of his/her name, salary, and type. Productions \( P_3 \) and \( P_4 \) define the type of an employee as either manager or worker, which is an example of generalization, or, alternatively viewed, an example of specialization. This example illustrates that the grammatical model allows for the three basic constructs of most data models (set-formation, generalization, and aggregation) in a uniform way. Finally notice production \( P_7 \). The grammatical model allows, in a convenient way, for additional syntactic features; e.g., the terminal symbol “$” indicates that the salary is expressed in dollar amounts.

For hierarchically organized data, the tree structure of the grammatical model also has some advantages with respect to implementation. Contrary to the relational and the nested relational models, for example, the straightforward implementation of the grammatical model is feasible, yielding a physical representation that is fairly close to the conceptual representation.

Turning to the dynamical aspects of the grammatical model, we see that there are two obvious ways to express queries by transforming the trees. A first method consists of defining operators that locally transform the trees and the grammars. A second way consists of describing this transformation in a less procedural way, indicating the relationship between the given trees and the result trees. For reasons that are evident these methods are called the algebra
and the calculus, respectively. They are proved to be equivalent. Although the calculus is the more natural of both methods, its formal semantics need a detailed description to handle all possible problems of cyclicity and ambiguity.

The paper is organized as follows. In §2, we give the basic definitions of the grammatical model. In §3, we define an algebra to query and manipulate information bases defined over the grammatical model. Section 4 introduces an alternative query mechanism called the grammatical calculus. The grammatical calculus is based on pattern-matching. Section 5 briefly discusses some decidability issues regarding the grammatical calculus. In §6, we show that the grammatical algebra and calculus are equivalent with respect to expressive power. Finally, §7 proposes some directions for future research.

2. The grammatical model. Throughout this paper, we assume the reader is familiar with the basic terminology concerning trees (e.g., [3]) and formal languages (e.g., [10]). As stated in §1, we shall represent an information base as a tree, the structure of which is controlled by a formal grammar. We shall borrow the terms "scheme" and "instance" from the relational model and use the former to indicate the grammar and the latter to indicate the tree.

DEFINITION 2.1. An information base scheme is a grammar $G = (V, T, S, P)$ with

- $V$ being a finite set of attributes;
- $T$ being a finite set of constants;
- $S$ being a set of axioms, $S \subseteq V$;
- $P$ being a finite set of productions of the form $A \rightarrow s$ where
  - $A \in V$,
  - $s \in (V \cup T)^*$,
  - each attribute appears at most once in $s$.

Actual data will be represented in a tree the internal nodes of which are labeled by attributes. Note that we do not require the leaves to be labeled by constants; if a leaf is labeled by an attribute, this simply means there are no data known for that attribute.

DEFINITION 2.2. Let $(V, T, S, P)$ be an information base scheme. A data-tree (D-tree) over $G$ is a tree whose nodes are labeled with elements of $V \cup T$ in such a way that each internal node is labeled by an attribute. The set of all D-trees over $G$ is denoted $\mathcal{D}(G)$. The empty D-tree is denoted $\epsilon$.

In the upcoming sections, we shall frequently use some operations on D-trees, which will be denoted as in Definition 2.3.

DEFINITION 2.3. Let $(V, T, S, P)$ be an information base scheme, and let $D$ be a D-tree over $G$. Then

- if $n$ is a node of $D$, $\text{lbl}(n)$ denotes its label;
- $\text{rt}(D)$ denotes the root of $D$ and $\text{rt}(D)$ the label of the root of $D$;
- if $n$ is a node of $D$ with $n \neq \text{rt}(D)$, $\text{par}(n)$ denotes its parent and $\text{par}(n)$ the label of its parent;
- if $n$ is a node of $D$, $\text{chln}(n)$ is the sequence of all its children, $\text{chtrs}(n)$ is the sequence of the subtrees of $D$ whose roots are the children of $n$, and $\text{chln}(n)$ is the sequence of the labels of all the children of $n$.

Finally, a D-tree over an information base scheme that is also a derivation tree will be called an information base instance.

DEFINITION 2.4. Let $G = (V, T, S, P)$ be an information base scheme. A D-tree $D$ over $G$ is called an information base instance over $G$ if

- $\text{rt}(D) \in S$;
- for each internal node $n$ in $D$, the production $\text{lbl}(n) \rightarrow \text{chln}(n)$ is in $P$.

\footnote{The empty string is denoted $\varepsilon$.}
Example 2.5. Consider the information base scheme $\mathcal{G} = (V, T, S, P)$ with

- $V = \{\text{Fams, Fam, Father, Mother, Children, Child, String, Chr}\}$;
- $T = \{A, \ldots, Z, a, \ldots, z\}$;
- $S = \{\text{Families}\}$;
- $P = \{(\text{Fams}) \rightarrow (\text{Fam})(\text{Fams}) \quad (\text{Child}) \rightarrow (\text{String})$
  
  $(\text{Fam}) \rightarrow (\text{Father})(\text{Mother})(\text{Children}) \quad (\text{String}) \rightarrow (\text{Chr})(\text{String})$
  
  $(\text{Children}) \rightarrow (\text{Child})(\text{Children}) \quad (\text{Chr}) \rightarrow A$
  
  $(\text{Father}) \rightarrow (\text{String})$
  
  $(\text{Mother}) \rightarrow (\text{String}) \quad (\text{Chr}) \rightarrow z\}$

representing the structure on information base concerning families with their children. Note that this information base scheme also includes an “implementation” of strings. For example, the string *Ian* must be represented as the D-tree in Fig. 2.

![Fig. 2. Representation of a string.](image)

However, we might have considered strings as elements of some set, sufficiently large for our purposes, rather than as a sequence of characters. In the upcoming sections we shall not bother with this low-level representation, since this is not our main concern. From now on, we shall no longer write productions for attributes such as $(\text{String})$. With this in mind, the D-tree in Fig. 3 represents an information base instance over $\mathcal{G}$ showing two families, the former composed of *Ian* and *Mary* with children *Brian* and *Wendy* and the latter composed of *Nick* and *Brenda* with no children.

![Fig. 3. An instance of an information base about families.](image)

Grammar-based models turn out to be highly appropriate for representing text-dominated databases, as was observed by Gonnet and Tompa. The following example is inspired by [11].
Example 2.6. Consider an information base scheme with the following productions:

\[
\begin{align*}
(\text{Refs}) & \rightarrow (\text{Ref})(\text{Refs}) \\
(\text{Ref}) & \rightarrow (\text{Auths})(\text{Tit})(\text{Source})(\text{Year}) \\
(\text{Source}) & \rightarrow (\text{Book}) \\
(\text{Auths}) & \rightarrow (\text{Auth})(\text{Auths}) \\
(\text{Issue}) & \rightarrow (\text{Vol}) : (\text{Nr}).
\end{align*}
\]

The attributes not mentioned in the left-hand side of some production are supposed to take either a string or a number as a value. Part of some information base instance is represented in Fig. 4.

![Fig. 4. An instance of a bibliographic information base.](image)

We will also need the notion of isomorphic D-trees:

**Definition 2.7.** Let \( (V, T, S, P) \) be an information base scheme, and let \( D_1 \) and \( D_2 \) be D-trees over \( G \). \( D_1 \) and \( D_2 \) are said to be isomorphic, denoted \( D_1 \cong D_2 \), if there exists a mapping between the nodes of \( D_1 \) and \( D_2 \) that is one to one and onto, preserving the labels and the tree structure. Isomorphism is extended to finite sequences of D-trees in the canonical way.

3. An algebra for transforming information bases. In this section, we propose an algebraic language for the manipulation of grammatically defined information bases that not only allows us to formulate queries, but also to apply more general transformations. Each operator is defined both on scheme and on instance level. Here we implicitly assume that only one information base instance is considered at a time.

The algebra we propose consists of eight basic operators, defined below. At the same time, we also define some derived operators, both as an illustration and because we need them further on.

First, we define three types of substitutions, which do not alter the structure of an information base instance, but only change (attribute) labels.

The parent substitution \( \Sigma \pi [A \rightarrow s, B] \) substitutes by \( B \) all attributes \( A \) from which \( s \) is derived.

**Definition 3.1.** Let \( (V, T, S, P) \) be an information base scheme, and let \( D \) be an information base instance over \( G \). Let \( A \rightarrow s \in P \), or \( A \in V \) and \( s \in E \). Let \( B \) be an attribute (\( B \) does not have to be in \( V \)), and suppose that \( A \) and \( B \) never occur simultaneously in the right-hand side of a production of \( P \). The parent substitution is defined as follows:

- \( \Sigma \pi [A \rightarrow s, B](G) = G' = (V', T, S', P') \) where
  - \( V' = V \cup \{B\} \);
  - if \( A \in S \), then \( S' = S \cup \{B\} \), else \( S' = S \);
  - Let \( P'' = (P - \{A \rightarrow s\}) \cup \{B \rightarrow s\} \). Then

\[
P' = P'' \cup \{C \rightarrow s_1Bs_2 \mid C \in V', \ s_1s_2 \in (V \cup T)^*, \text{ and } C \rightarrow s_1As_2 \in P''\}.
\]
\[ \Sigma \pi[A \rightarrow s, B](D) \] is obtained by simultaneously relabeling by \( B \) each node \( n \) in \( D \) with \( \text{lbl}(n) = A \) and \( \text{chln}(n) = s \).

Note that the condition of \( A \) and \( B \) not occurring simultaneously in the right-hand side of a production of \( G \) prevents \( B \) from appearing more than once in the right-hand side of a production of \( G' \).

The child substitution \( \Sigma \chi[A \rightarrow s_1B_{s_2}, B, C](D) \) substitutes by \( C \) all attributes \( B \) in a string \( s_1B_{s_2} \) that is derived from \( A \).

**Definition 3.2.** Let \( G = (V, T, S, P) \) be an information base scheme, and let \( D \) be an information base instance over \( G \). Let \( B \in V \), and let \( A \rightarrow s_1B_{s_2} \in P \). Let \( C \) be an attribute (\( C \) does not have to be in \( V \)), and suppose that \( C \) does not occur in \( s_1s_2 \). The child substitution is defined as follows:

\[ \Sigma \chi[A \rightarrow s_1B_{s_2}, B, C](D) \] is obtained by simultaneously relabeling by \( C \) each internal node \( n \) in \( D \) with \( \text{lbl}(n) = B \), \( \text{par}(n) = A \) and \( \text{chln}(\text{par}(n)) = s_1B_{s_2} \).

As before, the condition on \( C \) prevents illegal substitutions.

Finally, we define child equality substitution. The child equality substitution \( \Sigma e[A \rightarrow s_1B_{s_2}, B \rightarrow s_3, C, D](D) \) substitutes by \( D \) all attributes \( B \) in a string \( s_1B_{s_2} \) that is derived from \( A \) and from which \( s_3 \) is derived, provided \( B \) has both a sibling and a child labeled \( C \) that define isomorphic subtrees.

**Definition 3.3.** Let \( G = (V, T, S, P) \) be an information base scheme, and let \( D \) be an information base instance over \( G \). Let \( A \rightarrow s_1B_{s_2}, B \rightarrow s_3 \in P \). Let \( C \in V \), and suppose that \( C \) occurs both in \( s_1s_2 \) and \( s_3 \). Let \( D \) be an attribute (\( D \) does not have to be in \( V \)), and suppose that \( D \) does not occur in \( s_1s_2 \). The child equality substitution is defined as follows:

\[ \Sigma e[A \rightarrow s_1B_{s_2}, B \rightarrow s_3, C, D](D) \] is obtained by simultaneously relabeling by \( D \) each such node \( n \) for which \( D_1 \cong D_2 \).

**Example 3.4.** Consider an information base scheme \( G = (V, T, S, P) \) with \( V = \{A, B, C, D\} \), \( T = \{a, b\} \), \( S = \{A\} \), and

\[ P = \{A \rightarrow BA, A \rightarrow B, B \rightarrow CD, C \rightarrow a, C \rightarrow b, D \rightarrow a, D \rightarrow b\}, \]

and let \( D \) be the information base instance over \( G \) shown in Fig. 5.

Then the parent substitution \( \Sigma \pi[A \rightarrow BA, E] \) yields the information base scheme \( G' = (V', T, S', P') \) with \( V' = \{A, B, C, D, E\} \), \( S' = \{A, E\} \), and

\[ P' = \{E \rightarrow BA, A \rightarrow B, B \rightarrow CD, C \rightarrow a, C \rightarrow b, D \rightarrow a, D \rightarrow b, E \rightarrow BE\} \]

and the information base instance \( D' \) in Fig. 6.

The child substitution \( \Sigma \chi[E \rightarrow BA, A, E] \) applied to the information base thus obtained yields the information base scheme \( G'' = (V'', T, S'', P'') \) with

\[ P'' = \{A \rightarrow B, B \rightarrow CD, C \rightarrow a, C \rightarrow b, D \rightarrow a, D \rightarrow b, E \rightarrow BE, E \rightarrow B\} \]
and the information base instance $D''$ in Fig. 7.

Finally, the child equality substitution $\Sigma e[E \rightarrow BE, E \rightarrow BE, B, D]$ applied to the last result yields the information base scheme $G'''' = (V', T', S', P''')$ with

$$P''' = \{A \rightarrow B, B \rightarrow CD, C \rightarrow a, C \rightarrow b, D \rightarrow a, D \rightarrow b, E \rightarrow BE, E \rightarrow B, E \rightarrow BD, D \rightarrow BE, D \rightarrow BD\}$$

and the information base instance $D''' = \Sigma \chi[E \rightarrow BA, A, E](D')$. 

**Fig. 5.** An information base instance $D$.

**Fig. 6.** The information base instance $D' = \Sigma \pi[A \rightarrow BA, E](D)$.

**Fig. 7.** The information base instance $D'' = \Sigma \pi[A \rightarrow BA, E](D')$. 
and the information base instance $D''$ in Fig. 8.

Next, we define two operators that allow the introduction of new nodes and the removal of existing ones.

![Diagram](image_url)

**Fig. 8.** The information base instance $D'' = \Sigma_\epsilon\{E \rightarrow BE, E \rightarrow BE, B, D\}(D').$

The node insertion $N_t[A \rightarrow s_1s_2s_3, s_1Bs_3]$ inserts in each derivation of $s_1s_2s_3$ from $A$ a node $B$ as a child of $A$ and the father of $s_2$.

**DEFINITION 3.5.** Let $G = (V, T, S, P)$ be an information base scheme, and let $D$ be an information base instance over $G$. Let $A \rightarrow s_1s_2s_3 \in P$, or $A \in V$ and $s_1s_2s_3 = \epsilon$. Let $B$ be an attribute not in $V$. The node insertion is defined as follows:

- $N_t[A \rightarrow s_1s_2s_3, s_1Bs_3](G) = G' = (V', T, S, P')$ where
  - $V' = V \cup \{B\};$
  - $P' = (P - \{A \rightarrow s_1s_2s_3\}) \cup \{A \rightarrow s_1Bs_3, B \rightarrow s_2\}.$

- Let $n$ be a node of $D$ with $\text{lbl}(n) = A$ and $\text{chln}(n) = s_1s_2s_3$. Then $N_t[A \rightarrow s_1s_2s_3, s_1Bs_3](D)$ is obtained by simultaneously inserting for each such node $n$ a node $n'$ for which $\text{lbl}(n') = B$, $\text{par}(n') = n$, and $\text{chln}(n')$ is the subinterval of $\text{chln}(n)$ corresponding to $s_2$.

The node deletion $N_d[A]$ deletes each subtree whose root is labeled by $A$.

**DEFINITION 3.6.** Let $G = (V, T, S, P)$ be an information base scheme, and let $D$ be an information base instance over $G$. Let $A \in V$. The node deletion is defined as follows:

- $N_d[A](G) = G' = (V', T, S', P')$ where
  - $V' = V - \{A\};$
  - $S' = S - \{A\};$
  - Let $P'' = \{B \rightarrow s \mid B \in V', s \in (V' \cup T)^*, \text{ and } B \rightarrow s \in P\}$. Then
    - $P' = P'' \cup \{B \rightarrow s_1s_2 \mid B \in V', s_1s_2 \in (V' \cup T)^*, \text{ and } B \rightarrow s_1s_2 \in P\}.$

- $N_d[A](D)$ is obtained by deleting each subtree $D'$ from $D$ with $\text{rt}(D') = A$.

**Example 3.7.** Consider an information base scheme $G = (V, T, S, P)$ with $V = \{A, B, C, D, E\}$, $T = \{a, b, c, d, e\}$, $S = \{A\}$, and

$$P = \{A \rightarrow BCD, B \rightarrow abEd, C \rightarrow c, D \rightarrow Bd, E \rightarrow e\},$$

and let $D$ be the information base instance over $G$ shown in Fig. 9.

Then the node insertion $N_t[B \rightarrow abEd, aFd](D)$ yields the information base scheme $G' = (V', T, S, P')$ with $V' = \{A, B, C, D, E, F\}$ and

$$P' = \{A \rightarrow BCD, B \rightarrow aFd, C \rightarrow c, D \rightarrow Bd, E \rightarrow e, F \rightarrow bE\}$$
The node deletion $N_6[F]$ applied to the information base thus obtained yields the information base scheme $G'' = (V'', T, S, P'')$ with $V'' = \{A, B, C, D, E\}$ and

$$P'' = \{A \rightarrow BCD, B \rightarrow ad, C \rightarrow c, D \rightarrow Bd, E \rightarrow e\}$$

and the information base instance $D''$ of Fig. 11.

Note that $N_3[A]$ applied to any of the above information bases would yield the empty information base.

We also need two operators that copy information from one place in an information base to another. They are defined recursively.

Essentially, the downward duplication $\Delta_6[A \rightarrow s_1 Bs_2, B \rightarrow s_3, C, D]$ copies the subtree with root $C$ in a string $s_1 Bs_2$ derived from $A$ as the rightmost sibling of a string $s_3$ which is
derived from $B$, and renames the root of that copy to $D$. What makes the definition below somewhat involved is that, for reasons of uniformity, we have to require that this duplication is propagated into these subtrees as well.

**Definition 3.8.** Let $\mathcal{G} = (V, T, S, P)$ be an information base scheme, and let $\mathbf{D}$ be an information base instance over $\mathcal{G}$. Let $A \rightarrow s_1Bs_2 \in P$, and let either $B \rightarrow s_3 \in P$ or $s_3 = \varepsilon$. Let $C \in V$, and suppose that $C$ occurs in $s_1s_2$. Let $D$ be an attribute ($D$ does not have to be in $V$), and suppose that $D$ does not occur in $s_3$. The downward duplication is recursively defined as follows:

- $\Delta \delta[A \rightarrow s_1Bs_2, B \rightarrow s_3, C, D](\mathcal{G}) = G' = (V', T, S, P')$ where
  - $V' = V \cup \{D\}$;
  - $P' = P \cup \{B \rightarrow s_3D\} \cup \{D \rightarrow s \mid s \in (V \cup T)^* \text{ and } C \rightarrow s \in P\}$.

- Let $n$ be an internal node in $\mathbf{D}$ with $\text{lbl}(n) = B$, $\text{par}(n) = A$, $\text{chln}(\text{par}(n)) = s_1Bs_2$, and $\text{chln}(n) = s_3$. Let $m$ be the node in $\text{chln}(n)$ with $\text{lbl}(m) = C$. Let $\mathbf{D}'$ be the subtree of $\mathbf{D}$ defined by $\text{rt}(\mathbf{D}') = m$, and let $\mathbf{D}'' = \Delta \delta[A \rightarrow s_1Bs_2, B \rightarrow s_3, C, D](\mathbf{D}')$. Then $\Delta \delta[A \rightarrow s_1Bs_2, B \rightarrow s_3, C, D](\mathcal{G})$ is obtained by simultaneously adding to $\text{chln}(n)$ for each such node $n$ a rightmost sibling $n'$ with $\text{lbl}(n') = D$. The subtree $\mathbf{D}'$ of $\Delta \delta[A \rightarrow s_1Bs_2, B \rightarrow s_3, C, D](\mathcal{G})$ defined by $\text{rt}(\mathbf{D}') = n'$ is then determined by $\text{chtrs}(n') \cong \text{chtrs}(\text{rt}(\mathbf{D}''))$.

Example 3.9. Consider an information base scheme $(V, T, S, P)$ with $V = \{A, B, C\}$, $T = \{a, b, c, d\}$, $S = \{A\}$, and $P = \{A \rightarrow BaC, B \rightarrow bd, C \rightarrow cA, C \rightarrow c\}$, and let $\mathbf{D}$ be the information base instance over $\mathcal{G}$ in Fig. 12.

Then the downward duplication $\Delta \delta[A \rightarrow BaC, B \rightarrow bd, C, D]$ yields the information base scheme $\mathcal{G}' = (V', T, S, P')$ with $V' = \{A, B, C, D\}$ and

$$P' = \{A \rightarrow BaC, B \rightarrow bdD, C \rightarrow cA, C \rightarrow c, D \rightarrow cA, D \rightarrow c\}$$

and the information base instance $\mathbf{D}'$ of Fig. 13.

The downward duplication copies information downward into the tree; the upward duplication is its upward counterpart.

**Definition 3.10.** Let $\mathcal{G} = (V, T, S, P)$ be an information base scheme, and let $\mathbf{D}$ be an information base instance over $\mathcal{G}$. Let $A \rightarrow s_1Bs_2, B \rightarrow s_3 \in P$. Let $C \in V$, and suppose that $C$ occurs in $s_3$. Let $D$ be an attribute ($D$ does not have to be in $V$), and suppose that $D$ does not occur in $s_3$. The upward duplication is recursively defined as follows:

- $\Delta \upsilon[A \rightarrow s_1Bs_2, B \rightarrow s_3, C, D](\mathcal{G}) = G' = (V', T, S, P')$ where
  - $V' = V \cup \{D\}$;
  - $P' = P \cup \{A \rightarrow s_1Bs_2D\} \cup \{D \rightarrow s \mid s \in (V \cup T)^* \text{ and } C \rightarrow s \in P\}$.

- Let $n$ be an internal node in $\mathbf{D}$ with $\text{lbl}(n) = B$, $\text{par}(n) = A$, $\text{chln}(\text{par}(n)) = s_1Bs_2$, and $\text{chln}(n) = s_3$. Let $m$ be the node in $\text{chln}(n)$ with $\text{lbl}(m) = C$. Let $\mathbf{D}'$ be the subtree of $\mathbf{D}$ defined by $\text{rt}(\mathbf{D}') = m$, and let $\mathbf{D}'' = \Delta \upsilon[A \rightarrow s_1Bs_2, B \rightarrow s_3, C, D](\mathbf{D}')$. Then $\Delta \upsilon[A \rightarrow
The information base instance $D' = \Delta\delta[A \rightarrow B a, B \rightarrow b d, C, D](D)$ is obtained by simultaneously adding to $\text{chln}(\text{par}(n))$ for each such node $n$ a rightmost sibling $n'$ with $\text{lbl}(n') = D$. The subtree $D'$ of $\Delta\nu[A \rightarrow s_1 B s_2, B \rightarrow s_3, C, D](D)$ defined by $\text{rt}(D') = n'$ is then determined by $\text{chtrs}(n') \cong \text{chtrs}(\text{rt}(D''))$.

Since upward duplication is very similar to downward duplication, we omit an example. Using both downward and upward duplication, it is possible to simulate sidewise duplication, defined below.

**Definition 3.11.** Let $(V, T, S, P)$ be an information base scheme, and let $D$ be an information base instance over $\mathcal{G}$. Let $A \rightarrow s C s_2 P$. Let $D$ be an attribute (that does not have to be in $V$), and suppose that $D$ does not occur in $s s_2$. The **sidewise duplication** is recursively defined as follows:

- $\Delta\sigma[A \rightarrow s_1 C s_2, C, D](\mathcal{G}) = (V', T, S, P')$ where
  - $V' = V \cup \{D\}$;
  - $P' = P \cup \{(A \rightarrow s_1 C s_2 D) \cup \{(D \rightarrow s | s \in (V \cup T)^* \text{ and } C \rightarrow s \in P)\}.$

- Let $n$ be a node in $D$ with $\text{lbl}(n) = A$ and $\text{chln}(n) = s_1 C s_2$. Let $m$ be the node in $\text{chln}(n)$ with $\text{lbl}(m) = C$. Let $D''$ be the subtree of $D$ defined by $\text{rt}(D'') = m$, and let $D''' = \Delta\sigma[A \rightarrow s_1 C s_2, C, D](D'')$. Then $\Delta\sigma[A \rightarrow s_1 C s_2, C, D](D)$ is obtained by simultaneously adding to $\text{chln}(n)$ for each such node $n$ a rightmost sibling $n'$ with $\text{lbl}(n') = D$. The subtree $D'$ of $\Delta\sigma[A \rightarrow s_1 C s_2, C, D](D)$ defined by $\text{rt}(D') = n'$ is then determined by $\text{chtrs}(n') \cong \text{chtrs}(\text{rt}(D''''))$.

**Theorem 3.12.** Sidewise duplication can be expressed in terms of node insertion, downward duplication, upward duplication, and node deletion.

**Proof.** Let $\mathcal{G} = (V, T, S, P)$ be the schema of some information base, and consider the sidewise duplication $\Delta\sigma[A \rightarrow s_1 C s_2, C, D]$ in which $A, C, D, s_1$ and $s_2$ are as in Definition 3.11. Let $E$ be an attribute not in $V$. Then $\Delta\sigma[A \rightarrow s_1 C s_2, C, D]$ can be performed by consecutively executing the following operations:

1. the node insertion $Nt[A \rightarrow s_1 C s_2, s_1 C s_2 E]$;
2. the downward duplication $\Delta\delta[A \rightarrow s_1 C s_2 E, E \rightarrow \varepsilon, C, C]$;
3. the upward duplication $\Delta\nu[A \rightarrow s_1 C s_2 E, E \rightarrow C, C, D]$;
4. the node deletion $N\Delta[E]$.

In step 1, the node $E$ is actually created so that downward duplication can always be applied, even if $s_1 s_2 = \varepsilon$. □

Note that the effects of downward duplication, upward duplication, and sidewise duplication can be easily undone by deleting $D$ (using the notations used in the respective definitions). This observation yields a natural “embedding” of the nodes of the original information base.
instance into the resulting instance. From the definitions of the above operations, the result below follows in a straightforward manner.

**Theorem 3.13.** Let \( D \) be an information base instance over some appropriate scheme, and let \( D' \) be the resulting instance after a downward duplication, an upward duplication, or a sidewise duplication. Let \( n_1 \) and \( n_2 \) be two nodes in \( D \), and let \( n'_1 \) and \( n'_2 \), respectively, be the corresponding nodes in \( D' \). Let \( D_1 \) and \( D_2 \) be the subtrees of \( D \) defined by \( rt(D_1) = n_1 \) and \( rt(D_2) = n_2 \) and let \( D'_1 \) and \( D'_2 \) be the subtrees of \( D' \) defined by \( rt(D'_1) = n'_1 \) and \( rt(D'_2) = n'_2 \). Then \( D_1 \cong D'_2 \) if and only if \( D_1 \cong D_2 \). Suppose furthermore that in \( D' \) the subtree \( D'_3 \) is a “duplicate” of \( D_2 \). Let \( D'_3 \) be any \( D \)-tree defined by \( rt(D'_3) = rt(D'_2) \) and \( chtrs(rt(D'_3)) = chtrs(rt(D'_2)) \). Then \( D'_2 \cong D'_3 \) whence \( D_1 \cong D_2 \).

Theorem 3.13 will turn out to be essential in the proof of our main theorem, given in §6.

Finally, we introduce a permutation. Basically, a permutation recursively rearranges the children derived by some production \( A \rightarrow s \).

**Definition 3.14.** Let \( (V, T, S, P) \) be an information base scheme, and let \( D \) be an information base instance over \( G \). Let \( A \rightarrow s \in P \), or \( A \in V \) and \( s \in S \), and let \( s_2 \in (V \cup T)^\ast \) contain the same attributes as \( s \). The permutation is recursively defined as follows:

- \( \Pi[A \rightarrow s_1, s_2](G) = (V, T, S, P') \) where \( P' = (P - \{ A \rightarrow s_1 \}) \cup \{ A \rightarrow s_2 \} \).

- Let \( n \) be a node in \( D \) with \( lbl(n) = A \) and \( chin(n) = s_1 \). \( \Pi[A \rightarrow s_1, s_2](D) \) is obtained by first simultaneously substituting new nodes for \( chin(n) \) such that \( chin(n) \) becomes \( s_2 \). Now let \( B \) be an attribute in \( s_2 \), and let \( m \) be the node in \( \Pi[A \rightarrow s_1, s_2](D) \) with \( lbl(m) = B \) and \( par(m) = n \). Then the subtree \( D' \) of \( \Pi[A \rightarrow s_1, s_2](D) \) defined by \( rt(D') = m \) is isomorphic to \( \Pi[A \rightarrow s_1, s_2](D'') \), where the \( D \)-tree \( D'' \) is the subtree of \( D \) defined by \( rt(D'') = B \) and \( par(rt(D'')) = n \).

Our notion of permutation is somewhat wider than what is usually understood by this term. A permutation does indeed permute attributes, but can also insert, delete, and rearrange constants.

**Example 3.15.** Consider an information base scheme \( G = (V, T, S, P) \) with \( V = \{ A, B \} \), \( T = \{ a, b, c \} \), \( S = \{ A \} \), and \( P = \{ A \rightarrow aB, B \rightarrow bBc \} \), and consider the information base instance over \( G \) shown in Fig. 14 left.

Then the permutation \( \Pi[B \rightarrow bBc, abB](G) \) yields the information base scheme \( G' = (V, T, S, P') \) with \( P' = \{ A \rightarrow aB, B \rightarrow abB \} \) and the information base instance shown in Fig. 14 right.

**Fig. 14. An example of a permutation.**
there is a sequence of instance-independent grammatical algebra operations that returns the same result at the instance level. In general, however, it is unavoidable that the scheme returned by the algebra sequence defines a larger language than the scheme returned by the original operator (although in Theorem 3.12, they are equal).

We already saw that sidewise duplication can be expressed in the grammatical algebra. Below, we give two more examples of derived operations that are often needed in practical applications.

First, we introduce node merging. The node merging $N\mu[A \rightarrow s_1B_s_2, B \rightarrow s_3]$ is obtained by pruning out each attribute $B$ in a string $s_1B_s_2$ which is derived from $A$ and from which $s_3$ is derived. In this way, $s_1_s_3s_2$ will be derived from $A$ instead.

**Definition 3.16.** Let $G = (V, T, S, P)$ be an information base scheme, and let $D$ be an information base instance over $G$. Let $A \rightarrow s_1B_s_2 \in P$, and let $B \rightarrow s_3 \in P$, or $B \in V$ and $s_3 = e$. Suppose that no attribute in $s_3$ appears in $s_1s_2$. The node merging is defined as follows:

1. Let $n$ be a node of $D$ with $\text{lbl}(n) = B$, $\text{par}(n) = A$, $\text{chin}(\text{par}(n)) = sBs_2$, and $\text{chin}(n) = s_3$. Then $N\mu[A \rightarrow s_1B_s_3, B \rightarrow s_3](D)$ is obtained by simultaneously substituting each such node $n$ in $\text{chin}(\text{par}(n))$ by $\text{chin}(n)$.

Clearly, on instance level, a node insertion $N\iota[A \rightarrow s_1s_2s_3, s_1B_s_2]$ can be undone by the node merging $N\mu[A \rightarrow s_1B_s_3, B \rightarrow s_2]$. We now show the following.

**Theorem 3.17.** Node merging can be expressed in the grammatical algebra.

**Proof.** Rather than giving a notationally cumbersome proof, we illustrate the general techniques that are needed on an example.

Consider an information base with scheme $G = (V, T, S, P)$ where $V = \{A, B, C\}$, $T = \{a, b, c\}$, $S = \{A\}$, and $P = \{A \rightarrow aBc, B \rightarrow bC, C \rightarrow c\}$. The node merging $N\mu[A \rightarrow aBc, B \rightarrow bC]$ can be expressed by consecutively performing the following operations:

1. the child substitution $\Sigma_{\chi}[A \rightarrow aBc, B, B']$;
2. the upward duplication $\Delta_{\nu}[A \rightarrow aB'c, B' \rightarrow bC, C, C]$;
3. the child substitution $\Sigma_{\chi}[A \rightarrow aB'cC, B', B'']$;
4. the node deletion $N\delta[B'']$;
5. the permutation $\Pi[A \rightarrow acC, abCc]$;
6. the child substitution $\Sigma_{\chi}[A \rightarrow aB'c, B', B]$.

We invite the reader to check our claim on a concrete instance.

In general, step 2 must be carried out for each attribute in $s_3$. In step 5, all the copied attributes must be arranged in the right order and all the constants in $s_3$ inserted in the right place.

Below, we give another example of a derived operator that will turn out to be very useful in the proof of our main result in §5.

**Definition 3.18.** Let $G = (V, T, S, P)$ be an information base scheme, and let $D$ be an information base instance over $G$. Let $A \rightarrow s_1C_1s_2C_2s_3 \in P$ with $C_1 \neq A$ and $C_2 \neq A$. Let $B$ be an attribute ($B$ does not have to be in $V$), and suppose that $A$ and $B$ never occur simultaneously in the right-hand side of a production of $P$. The parent equality substitution is defined as follows:

1. $\Sigma_{\sigma}[A \rightarrow s_1C_1s_2C_2s_3, C_1, C_2, B](G) = G' = (V', T, S', P')$ where $V' = V \cup \{B\}$;
2. if $A \in S$, then $S' = S \cup \{B\}$, else $S' = S$;
Let $P'' = P \cup \{B \rightarrow s_1s_2s_3s_5\}$. Then

$$P' = P'' \cup \{C \rightarrow s_3s_5 \mid C \in V', \ s_4s_5 \in (V \cup T)^*, \text{ and } C \rightarrow s_4s_5 \in P''\}.$$ 

• Let $n$ be a node in $D$ with $\text{lbl}(n) = A$ and $\text{chln}(n) = s_1s_2s_3s_5$. Let $m_1$ be the child of $n$ with label $C_1$ and $m_2$ be the child of $n$ with label $C_2$ and let $D_1$ and $D_2$ be the subtrees of $D$ with $\text{rt}(D_1) = m_1$ and $\text{rt}(D_2) = m_2$. Let $D_3$ be the $D$-tree defined by $\text{rt}(D_3) = m_1$ and $\text{chtrs}(D_3) = \text{chtrs}(D_2)$. Then $\Sigma_c[A \rightarrow s_1s_2s_2s_3, C_1, C_2, B](D)$ is obtained by simultaneously relabeling by $B$ each such node $n$ for which $D_1 \cong D_3$.

The reader may wonder why we have imposed the restriction $C_1 \neq A$ and $C_2 \neq A$. Indeed, without this restriction, parent equality substitution would still be well defined. In the upcoming sections, however, we only need the restricted parent equality substitution, and, although Theorem 3.19 below still holds for the unrestricted parent equality substitution, the proof would become very involved.

**Theorem 3.19.** Parent equality substitution can be expressed in the grammatical algebra.

**Proof.** Let $G = (V, T, S, P)$ be the scheme of some information base, and consider the parent equality substitution $\Sigma_c[A \rightarrow s_1s_2s_3s_5, C_1, C_2, B]$ (cf. Definition 3.18). Let $D$ and $E$ be attributes not in $V$. Then $\Sigma_c[A \rightarrow s_1s_2s_3s_5, C_1, C_2, B]$ can be performed by consecutively performing the following operations:

1. the node insertion $N_i[A \rightarrow s_1s_2s_3s_5, s_1s_2s_3s_5]$;
2. the child substitution $E_x[A \rightarrow s_1s_2s_3s_5, C_1, C_2]$;
3. the child equality substitution $E_e[A \rightarrow s_1s_2s_3s_5, D \rightarrow C_2, C_2, E]$;
4. the parent substitution $E_p[A \rightarrow s_1s_2s_3s_5, B]$;
5. the child substitution $E_x[B \rightarrow s_1s_2s_3s_5, C_2, C_1]$;
6. the child substitution $E_x[A \rightarrow s_1s_2s_3s_5, C_2, C_1]$;
7. the node merging $M_u[B \rightarrow s_1s_2s_3s_5, E \rightarrow C_2]$;
8. the node merging $M_u[A \rightarrow s_1s_2s_3s_5, D \rightarrow C_2]$.

We now return to the bibliographical Example 2.6 to illustrate on a more realistic information base how the grammatical algebra can be used to solve queries or to perform transformations.

**Example 3.20.** Reconsider the information base of Example 2.6. Suppose we want to extract only the information on journal titles (between double quotes) with the name of the journal and the volume. The instance of Fig. 4 would then be transformed into the instance of Fig. 15.

![Fig. 15. A transformation of the information base instance of Fig. 4.](image)

The transformation can be accomplished by consecutively performing the following operations:

1. $N_0[\langle \text{Authors} \rangle]$;
2. $N_0[\langle \text{Year} \rangle]$;
3. $\Delta v[\langle \text{Source} \rangle \rightarrow (\text{Journ})(\text{Issue}), (\text{Issue}) \rightarrow (\text{Vol}) : (\text{Nr}), (\text{Vol}), (\text{Volume})]$. 

4. Nδ[(Issue)];
5. Nμ[(Ref) → (Tit)(Source), (Source) → (Journ)(Volume)];
6. Π[(Ref) → (Tit)(Journ)(Volume), (Tit)"(Journ)(Volume)].

Observe that the instance obtained in the above example can be considered as a representation of a flat relational database model relation in the grammatical model. Below, we show how unary relational algebra operators can be performed. Note that, in order to simulate union, difference, and join, we need binary operators on information bases. This is beyond the scope of the present paper, however.

**Example 3.21.** Consider the following relational database relation \( R \):

\[
\begin{array}{ccc}
A & B & C \\
\hline
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
\vdots
\end{array}
\]

This relation can be represented as an information base with scheme \( G = (V, T, S, P) \) where \( V = \{U, R, A, B, C\} \), \( T = \{a, b, c\} \), \( S = \{U\} \), and

\[
P = \{U \rightarrow RU, R \rightarrow ABC, A \rightarrow a_1, \ldots, B \rightarrow b_1, \ldots, C \rightarrow c_1, \ldots\}
\]

and with instance the tree shown in Fig. 16.

![Fig. 16. A representation of a flat relation instance in the grammatical model.](image)

We now consider three typical examples of unary relational algebra operators:

- **The renaming** of \( A \) to \( A' \) can be expressed as the child substitution \( \Sigma_X[R \rightarrow ABC, A, A'] \).
- **The projection** of \( R \) onto \( AB \) can be expressed as the node deletion \( N\delta[C] \).
- **The selection** \( A = C \) can be expressed by consecutively performing the following operations:
  1. the parent equality substitution \( \Sigma\sigma[R \rightarrow ABC, A, C, R'] \);  
  2. the node deletion \( N\delta[R] \);  
  3. the parent substitution \( \Sigma\pi[R' \rightarrow ABC, R] \).

If desired, redundant \( U \)-nodes can be removed by repeatedly applying the node mergings \( N\mu[U \rightarrow U, U \rightarrow RU] \) and \( N\mu[U \rightarrow RU, U \rightarrow U] \).

Notice in the last example that the grammatical algebra does not have an iteration construct. The number of applications of the node mergings is therefore instance dependent (in this case, linear in the size of the instance).

**4. A calculus for transforming information bases.** Whereas in the previous section we defined a transformation language on information bases based on eight primitive operators
which we claim to be sufficiently powerful to express a large class of queries, it is also possible to define a more declarative transformation language inspired by the relational calculus.

An expression in the grammatical calculus we propose consists of a set of conditions and a transformation clause, both built from variables. Informally, applying a calculus expression to an information base means performing the required transformations for each “occurrence” of the variables satisfying the set of conditions. Since the variables in a calculus expression represent so-called rootless data trees, we first explain this notion.

**DEFINITION 4.1.** Let $\mathcal{G} = (V, T, S, P)$ be an information base scheme. A *rootless data tree* (R-tree) over $\mathcal{G}$ is a finite sequence of D-trees over $\mathcal{G}$. The set of all R-trees over $\mathcal{G}$ is denoted $\mathcal{R}(\mathcal{G})$.

We have to introduce the following notations concerning R-trees.

**DEFINITION 4.2.** Let $\mathcal{G} = (V, T, S, P)$ be an information base scheme, and let $R$ be an R-tree over $\mathcal{G}$. Then

- $R = (D_1, \ldots, D_n)$ denotes the sequence of D-trees of which $R$ consists;
- $\text{top}(R) = (\text{rt}(D_1), \ldots, \text{rt}(D_n))$ denotes the sequence of the roots of the D-trees of which $R$ consists;
- $\mathbf{F}$ denotes the empty R-tree.

Obviously, the isomorphism between D-trees defined in Definition 2.7 can be extended in a natural way to R-trees. The definition of R-trees also leads us to the following straightforward conclusions.

**PROPOSITION 4.3.** Let $\mathcal{G} = (V, T, S, P)$ be an information base scheme.

- Any interval$^2$ of an R-tree over $\mathcal{G}$ is also an R-tree over $\mathcal{G}$.
- Let $D$ be a D-tree over $\mathcal{G}$. Then $(D)$ is an R-tree.
- Let $D$ be a D-tree over $\mathcal{G}$. Then $\text{chltrs}(\text{rt}(D))$ is an R-tree.

It follows from Proposition 4.3 that R-trees can be contained in D-trees.

**DEFINITION 4.4.** Let $\mathcal{G} = (V, T, S, P)$ be an information base scheme. Let $D$ be a D-tree and $R$ be an R-tree over $\mathcal{G}$. $R$ is called a rootless subtree of $D$ if $R = (D_1, \ldots, D_n)$, $D_1, \ldots, D_n$ are subtrees of $D$ and $\text{top}(R)$ is a sequence of consecutive siblings of $D$. The common parent of these siblings is denoted $\text{par}(R)$. The set of all rootless subtrees of $D$ is denoted $\text{rst}(D)$. The R-tree $R$ is called a maximal rootless subtree of $D$ if, in addition, $\text{top}(R)$ is a maximal sequence of consecutive siblings of $D$, i.e., if $\text{top}(R) = \text{chln}(\text{par}(R))$.

Finally, we need two simple operations on R-trees.

**DEFINITION 4.5.** Let $\mathcal{G} = (V, T, S, P)$ be an information base scheme.

- Let $R_1 = (D_1, \ldots, D_m)$ and $R_2 = (D_{m+1}, \ldots, D_n)$ be R-trees over $\mathcal{G}$. Then the concatenation $R_1R_2$ is the R-tree defined by $R_1R_2 = (D_1, \ldots, D_n)$.
- Let $R$ be an R-tree over $\mathcal{G}$ and let $n$ be an arbitrary node. If $\text{lbl}(n)$ is an attribute or $R$ is empty, then the completion $nR$ is the D-tree defined by $\text{rt}(nR) = n$ and $\text{chltrs}(n) = R$.

As mentioned, variables in a calculus expression represent rootless subtrees of the information base instance under consideration. From these variables, terms are built using concatenation and completion. As a consequence of Proposition 4.3, these terms in turn represent R-trees. The set of conditions in a calculus expression consists of declarations of variables by terms and of equations between variables and terms; the substitution clause consists of a variable and the term by which that variable has to be substituted. Of course, the variables in the substitution clause have to occur in the set of conditions. Before formalizing the syntax of a grammatical calculus expression, we clarify the concept by an example.

**Example 4.6.** Reconsider the information base of Example 2.6 and the query of Example 3.20, described in the grammatical algebra. This query can also be solved by the following

---

$^2$By an interval of a sequence we mean a subsequence consisting of consecutive elements.
grammatical calculus expression:

\[
[p_2 \leftarrow "((\text{Tit})_p_4)(\text{Journ})_p_7)(\text{Volume})_p_9] |
\{
  p_1 := ((\text{Ref})_p_2)
  p_2 := ((\text{Auth})_p_3)(\text{Tit})_p_4)((\text{Source})_p_5)((\text{Year})_p_6)
  p_5 := ((\text{Journ})_p_7)((\text{Issue})_p_8)
  p_8 := ((\text{Vol})_p_9)((\text{Nr})_p_{10})].
\]

Another similar grammatical calculus expression that has the same effect on information bases over the scheme of Example 2.6 is the following:

\[
[p_2 \leftarrow "((\text{Tit})_p_4)(\text{Journ})_p_7)(\text{Volume})_p_9] |
\{
  p_1 := ((\text{Ref})_p_2)
  p_2 := ((\text{Auth})_p_3)(\text{Tit})_p_4)((\text{Source})_p_5)_p_6
  p_5 := ((\text{Journ})_p_7)((\text{Issue})_p_8)
  p_8 := ((\text{Vol})_p_9)((\text{Nr})_p_{10})].
\]

In both expressions, the four declarations in the right-hand side specify a “pattern” in the bibliographic information base; each time that pattern is found in the instance, it must be changed according to the substitution clause.

We now formally define the syntax of the grammatical calculus. Throughout this exposition, we assume that \( V \) is an infinitely enumerable set of variables.

**Definition 4.7.** Let \( (V, T, S, P) \) be an information base scheme.

- A basic term over has one of the following three types:
  - type 0: \( a \) \((a \in T)\);
  - type 1: \( p_i \) \((p_i \in V)\);
  - type 2: \( (A p_i) \) \((A \in V, p_i \in V)\).

- A term over \( G \) is a finite sequence of basic terms over \( G \) that contains at most one basic term of type 1 and in which each variable and each attribute appears at most once. The empty term is denoted \( \epsilon \). The set of all variables occurring in a term \( t \) is denoted \( \text{var}(t) \).

**Definition 4.8.** Let \( (V, T, S, P) \) be an information base scheme.

- A declaration over \( G \) has the form \( p_i := t \) with \( p_i \in V \) and \( t \) a term over \( G \) in which \( p_i \) does not occur.

- An equation over \( G \) has the form \( p_i = p_j \) with \( p_i, p_j \in V \).

**Definition 4.9.** Let \( (V, T, S, P) \) be an information base scheme. Let \( D = \{ p_i := t_i \mid i \in I \} \), \( I \) a set of indices, be a finite set of declarations over \( G \) in which no variable appears in the left-hand side of more than one declaration and in the right-hand side of more than one declaration. Let \( \text{var}(D) \) denote the set of all variables occurring in \( D \). Consider the associated directed graph \( G(D) \) with set of nodes \( \text{var}(D) \) and set of edges \( \{ p_j \rightarrow p_k \mid j \in I \text{ and } p_k \in \text{var}(t_j) \} \). \( D \) is called hierarchical if \( G(D) \) is a tree, and, furthermore, the root \( p_{\text{root}} \) has a declaration of the form \( p_{\text{root}} := (A p_i) \) for some \( A \in V \) and \( p_i \in V \).

For a hierarchical set of declarations it makes sense to define the following.

**Definition 4.10.** Let \( (V, T, S, P) \) be an information base scheme, and let \( D \) be a hierarchical set of declarations over \( G \).

- Let \( p_i \in \text{var}(D) \) be a variable which is not the root of \( G(D) \). Then \( p_i \) is said to be of type 1 (of type 2) if the unique basic term containing \( p_i \) in the right-hand side of a declaration of \( D \) is of type 1 (of type 2).

- Let \( p_i \in \text{var}(D) \) be an arbitrary variable. The depth \( d(p_i) \) of \( p_i \) is recursively defined as follows, using the hierarchy in \( G(D) \):
1. If \( \rho_i \) is the root of \( \mathcal{G}(\mathcal{D}) \), then \( d(\rho_i) = 0 \);
2. If \( \rho_i \) is of type 1 and \( \rho_j \) is the parent of \( \rho_i \), then \( d(\rho_i) = d(\rho_j) \);
3. If \( \rho_i \) is of type 2 and \( \rho_j \) is the parent of \( \rho_i \), then \( d(\rho_i) = d(\rho_j) + 1 \).

Observe that, given a hierarchical set of declarations \( \mathcal{D} \), the root of \( \mathcal{G}(\mathcal{D}) \) has no type.

**Example 4.11.** Consider the set of declarations in the right-hand side of the first expression in Example 4.6:

\[
\{ \rho_1 := ((\text{Ref})\rho_2) \\
   \rho_2 := ((\text{Auths})\rho_3)((\text{Tit})\rho_4)((\text{Source})\rho_5)(\text{Year}\rho_6) \\
   \rho_5 := ((\text{Journ})\rho_7)((\text{Issue})\rho_8) \\
   \rho_8 := ((\text{Vol})\rho_9) : ((\text{Nr})\rho_{10}) \}
\]

Obviously, this is a hierarchical set of declarations. Its associated tree is shown in Fig. 17.

**Fig. 17. The tree associated with the hierarchical sets of declarations in Example 4.11.**

All variables (except of course for \( \rho_1 \)) are of type 2. Furthermore, \( d(\rho_1) = 0 \), \( d(\rho_2) = 1 \), \( d(\rho_3) = d(\rho_4) = d(\rho_5) = d(\rho_6) = 2 \), \( d(\rho_7) = d(\rho_8) = 3 \), and \( d(\rho_9) = d(\rho_{10}) = 4 \). Now, consider the set of declarations in the right-hand side of the second expression in Example 4.6:

\[
\{ \rho_1 := ((\text{Ref})\rho_2) \\
   \rho_2 := ((\text{Auths})\rho_3)((\text{Tit})\rho_4)((\text{Source})\rho_5)\rho_6 \\
   \rho_5 := ((\text{Journ})\rho_7)((\text{Issue})\rho_8) \\
   \rho_8 := ((\text{Vol})\rho_9) : ((\text{Nr})\rho_{10}) \}
\]

This is a hierarchical set of declarations with the same associated tree as the previous one. However, \( \rho_6 \) is now of type 1 with \( d(\rho_6) = 1 \).

We now have all the ingredients to define an expression.

**Definition 4.12.** Let \( \mathcal{G} = (V, T, S, P) \) be an information base scheme. An expression over \( \mathcal{G} \) has the form \( [\rho_j \leftarrow u \mid \mathcal{D} \cup \mathcal{E}] \) with \( \rho_j \in \mathcal{V} \), \( u \) a term over \( \mathcal{G} \) with \( \rho_j \notin \text{var}(u) \), \( \mathcal{D} \) a hierarchical set of declarations over \( \mathcal{G} \), and \( \mathcal{E} \) a set of equations over \( \mathcal{G} \), satisfying the following conditions:

1. All variables in the expression occur in \( \mathcal{D} \);
2. No variable in \( u \) is an ancestor of \( \rho_j \) in \( \mathcal{G}(\mathcal{D}) \);
3. If, in addition, \( \rho_j \) is the root of \( \mathcal{G}(\mathcal{D}) \), then \( u = (B\rho_k) \) for some \( B \in V \) and \( \rho_k \in \mathcal{V} \), or \( u = \varepsilon \).

We invite the reader to check that the expression (without equations) in Example 4.6 satisfies Definition 4.12.

Before formally defining the semantics of a grammatical calculus expression, we show with examples how the grammatical algebra operators can be expressed in the calculus. As in
the previous section, we are not concerned with the resulting information base schemes (about which we have not yet said anything with regard to the calculus).

Example 4.13. Reconsider Examples 3.4, 3.7, 3.9, and 3.15.
- The parent substitution $\Sigma \pi[A \rightarrow BA, E]$ can be expressed by
  $$\rho_1 \leftarrow (E\rho_2) \mid \{\rho_1 := (A\rho_2), \rho_2 := (B\rho_3)(A\rho_4)\}.$$
- The child substitution $\Sigma \epsilon[E \rightarrow BA, A, E]$ can be expressed by
  $$\rho_2 \leftarrow (B\rho_3)(E\rho_4) \mid \{\rho_1 := (E\rho_2), \rho_2 := (B\rho_3)(A\rho_4)\}.$$
- The child equality substitution $\Sigma \epsilon[E \rightarrow BE, E \rightarrow BE, B, D]$ can be expressed by
  $$\rho_2 \leftarrow (B\rho_3)(D\rho_4) \mid \{\rho_1 := (E\rho_2), \rho_2 := (B\rho_3)(E\rho_4), \rho_4 := (B\rho_5)(E\rho_6), \rho_3 = \rho_3\}.$$
- The node insertion $\Sigma \iota[B \rightarrow abEd, aFd]$ can be expressed by
  $$\rho_2 \leftarrow a(F\rho_3)d \mid \{\rho_1 := (B\rho_2), \rho_2 := ap_3d, \rho_3 := b(E\rho_4)\}.$$
- The node deletion $\Sigma \delta[F]$ can be expressed by $\rho_1 \leftarrow \varepsilon \mid \rho_1 := (F\rho_2))$.
- The downward duplication $\Sigma \delta[A \rightarrow BaC, B \rightarrow bd, C, D]$ can be expressed by
  $$\rho_3 \leftarrow bd(D\rho_4) \mid \{\rho_1 := (A\rho_2), \rho_2 := (B\rho_3)(C\rho_4), \rho_3 := bd\}.$$
- The permutation $\Pi[B \rightarrow bBc, abB]$ can be expressed by
  $$\rho_2 \leftarrow ab(B\rho_3) \mid \{\rho_1 := (B\rho_2), \rho_2 := b(B\rho_3)c\}.$$

Describing the semantics of the grammatical calculus should consist of two parts: explaining what happens with schemes and explaining what happens with instances. Since, as observed earlier in this paper, it is unrealistic to compare information base operations at the scheme level with respect to expressiveness, we shall not elaborate on how calculus expressions work on information base schemes. The example below, however, should nevertheless convince the reader that calculus expressions can be applied to the schemes as well.

Example 4.14. Consider the calculus expression
  $$\rho_2 \leftarrow (D\rho_4) \mid \{\rho_1 := (B\rho_2), \rho_2 := (B\rho_3)(C\rho_4)\}.$$
and let $G = (V, T, S, P)$ be an information base scheme to which this calculus expression is applied. Let $G' = (V', T', S', P')$ denote the resulting scheme. Then $V' = V \cup \{D\}$ and $T' = T$ since there are no other attributes or constants in the substitution clause not occurring in one of the declarations. Furthermore, $S' = S$, since the attribute in the term defining the root variable is not altered. Finally, if $A \rightarrow BC \in P$, then $P' = P \cup \{A \rightarrow D\} \cup \{D \rightarrow s \mid C \rightarrow s \in P\}$, else $P' = P$.

Note that, for the sake of generality, the application of a calculus expression to an information base scheme can only result in adding new productions.

We now formally define the semantics of a grammatical calculus expression at the instance level. Although it is conceptually simple, as should be clear from the examples given thus far, the formalism itself is rather involved. This stems mainly from the fact that, when rearranging subtrees in applying a calculus expression, one must be able to describe how this rearrangement is “propagated” downward into these subtrees.

The evaluation of a calculus expression on a given information base instance can be described in two distinct stages.
First, the variables in an expression are “valuated” as rootless subtrees of the considered information base instance, satisfying the declarations and equations in that expression.

Then, the D-tree representing the information base instance is transformed according to these valuations and the transformation rule in the left-hand side of the expression.

The first stage is described in Definition 4.15, the second one in Definition 4.18.

**Definition 4.15.** Let \( G = (V, T, S, P) \) be an information base scheme, and let \( D \) be an information base instance over \( G \). Let \( E \) be a set of declarations and \( \mathcal{E} \) a set of equations over \( G \) such that all variables in \( \mathcal{E} \) occur in \( E \). Let \( f: \text{var}(D) \rightarrow \text{rst}(D) \) be a total mapping from variables in \( D \) to rootless subtrees of \( D \). \( f \) is called a valuation of \( D \) and \( \mathcal{E} \) in \( D \) if

1. for each declaration \( \rho_i := e \) in \( E \), \( f(\rho_i) = F \);
2. for each declaration \( \rho_i := t \) in \( D \) with \( t = t_1 \ldots t_k \), \( t_1 \ldots t_k \) being basic terms, \( f(\rho_i) = R_1 \ldots R_k \) with, for each \( j = 1, \ldots, k \),
   1. if \( t_j = a \) for some \( a \in T \), then \( R_j \) is a single-node rootless tree the node of which is labeled \( a \),
   2. if \( t_j = \rho_k \) for some \( \rho_k \in V \), then \( R_j = f(\rho_k) \),
   3. if \( t_j = (A \rho_k) \) for some \( A \in V \) and \( \rho_k \in V \), then there is a node \( n \) in \( D \) with \( \text{lbl}(n) = A \) such that \( R_j = (n \ f(\rho_k)) \),
3. for each equation \( \rho_i = \rho_j \) in \( E \), \( f(\rho_i) \cong f(\rho_j) \).

The set of all valuations of \( \mathcal{E} \) and \( \mathcal{E} \) in \( D \) is denoted \( \mathcal{T}(\mathcal{E}, \mathcal{E}, D) \).

**Example 4.16.** Consider the information base instance \( D \) of Fig. 18 (over some appropriate scheme) and let \( E \) be the following calculus expression:

\[
[\rho_5 \leftarrow (C\rho_7)\rho_8 \mid \{\rho_1 \leftarrow (A\rho_2), \rho_2 \leftarrow \rho_3c(\rho_4), \rho_3 \leftarrow (B\rho_5)\rho_6, \rho_4 \leftarrow \rho_7c(\rho_8), \\
\rho_5 \leftarrow (C\rho_9)(D\rho_{10}), \rho_7 \leftarrow (B\rho_{11})\rho_{12}, \rho_{10} \leftarrow \varepsilon, \rho_{11} \leftarrow (C\rho_{13})\rho_{14}, \rho_9 = \rho_{13}\}].
\]

There are three valuations of \( D \) and \( E \) in \( D \). For each such valuation \( f \), the node \( n \) satisfying \( \text{top}(f(\rho_1)) = (n) \) has been marked by a square in Fig. 18. We leave it to the reader to check that these markings completely determine the corresponding valuations.

---

**Fig. 18.** An information base instance \( D \).
mapped. Lemma 4.17 says even more: it suffices to specify a rootless subtree of which the valuation of one arbitrary variable is an interval. In particular, Lemma 4.17 ensures that the way in which \( f(\rho_5) \) was indicated in Example 4.16 is unambiguous. More generally, it excludes that images of the same variables under different valuations "overlap." This will allow us to define the result of a calculus expression by performing a transformation on the information base instance under consideration for each valuation of its hierarchical set of declarations and set of equations.

**Lemma 4.17.** Let \( G = (V, T, S, P) \) be an information base scheme, and let \( D \) be an information base instance over \( G \). Let \( D \) be a hierarchical set of declarations over \( G \), and let \( \mathcal{E} \) be a set of equations over \( G \) such that all variables in \( \mathcal{E} \) occur in \( D \). Let \( \rho_i \) be an arbitrary variable in \( D \), and let \( R \) be a rootless subtree of \( D \). There exists at most one valuation \( f \) of \( D \) and \( \mathcal{E} \) in \( D \) such that \( f(\rho_i) \) is an interval of \( R \).

**Proof.** Suppose there exists a valuation \( f \) such that \( f(\rho_i) \) is an interval of \( R \). We first show that for some variable \( \rho_i \in \text{var}(D) \), \( f(\rho_i) \) is unambiguously determined by this condition. If \( \rho_i \) is the root of \( G(D) \), we know there exists a declaration in \( D \) of the form \( \rho_i : = (A\rho_j) \). Since \( R \) contains at most one subtree with a root labeled \( A \), \( f(\rho_i) \) is unambiguously determined. Now suppose \( \rho_i \) is not the root of \( G(D) \). Then there exists a sequence of variables in \( D \), say \( \rho_{i_0}, \ldots, \rho_{i_k}, k \geq 1 \), that satisfy the following conditions:

1. \( \rho_{i_0} = \rho_i \);
2. For all \( k \) there is a declaration in \( D \) of the form \( \rho_{i_k} : = \rho_{i_{k-1}} \);
3. There is a declaration in \( D \) of the form \( \rho_{i_k} : = (A\rho_{i_{k+1}}) \).

Note that the second condition is voidlessly satisfied if \( k = 1 \). The last condition can always be satisfied because a similar condition holds for the root of \( G(D) \). Now let \( R' \) be the unique maximal rootless subtree of \( D \) containing \( R \). Then, by condition 2 above, \( f(\rho_i) \) being an interval of \( R \) implies that \( f(\rho_{i_0}), \ldots, f(\rho_{i_{k-1}}) \) are all intervals of \( R' \). Since a declaration of the form \( \rho_{i_k} : = (A\rho_{i_{k+1}}) \) is in \( D \), it now follows that \( f(\rho_{i_{k-1}}) \) equals \( R' \).

Up to now, we have shown there exists some variable \( \rho_i \in D \) for which \( f(\rho_i) \) is a fully determined rootless subtree of \( D \). Since it is easily shown that whenever \( f \) is unambiguously determined for a certain variable it is also unambiguously determined for both the parent and all children of that variable, a straightforward induction shows that \( f \) is unambiguously determined for all variables in \( D \).

We now define how an expression transforms D-trees. First, Definition 4.18 introduces so-called \( E \)-transformations. Theorem 4.19 then establishes the uniqueness of these \( E \)-transformations. Finally, Definition 4.20 points out how the unique \( E \)-transformation must be used to define the result of the calculus expression \( E \).

We start with the notion of \( E \)-transformation. When rearranging rootless subtrees of a given D-tree in applying a calculus expression, we must be able to describe how the rearrangement is “propagated” downward into these subtrees. Intuitively, the resulting tree will therefore have to be constructed “bottom-up.” Thus we cannot just define the effect of a calculus expression on D-trees alone; we need to define the effect on all rootless subtrees as well. The effect of a calculus expression on a rootless subtree is context-sensitive, however. Therefore, we introduce the notion of a \( E \)-transformation, which defines the effect of a calculus expression \( E \) on an R-tree \( R \) in the context of a D-tree \( D \) of which \( R \) is a rootless subtree.

**Definition 4.18.** Let \( G = (V, T, S, P) \) be an information base scheme and let \( E \equiv \{ \rho_j \leftarrow u \mid \mathcal{D} \cup \mathcal{E} \} \) be an expression over \( G \). A partial mapping \( g: \mathcal{R}(G) \times \mathcal{D}(G) \rightarrow \mathcal{R}(G)/_{\equiv} \) is an \( E \)-transformation of \( G \) if it satisfies the following conditions:

\[3\] For reasons of convenience, the conditions are formulated as if \( g(R, D) \) were an arbitrary representation of the class under consideration.
1. \( g(R, D) \) is defined if and only if \( R \in \text{rst}(D) \);
2. For all \( D \in \mathcal{D}(G) \), \( g(F, D) \cong F \);
3. For some \( D \in \mathcal{D}(G) \), let \((nR) \in \text{rst}(D)\) be a rootless subtree such that for no valuation \( f \in \mathcal{F}(D \cup E, D) \), \( f(p_j) = (nR) \). Then \( g((nR), D) \cong (ng(R, D)) \);
4. For some \( D \in \mathcal{D}(G) \), let \( R \in \text{rst}(D) \) be a rootless subtree such that for no valuation \( f \in \mathcal{F}(D \cup E, D), f(p_j) = (nR) \). Then if \( R = R_1 R_2, g(R, D) \cong g(R_1, D)g(R_2, D) \);
5. For some \( D \in \mathcal{D}(G) \), let \( R \in \text{rst}(D) \) be a rootless subtree such that for some valuation \( f \in \mathcal{F}(D \cup E, D), f(p_j) = R \). Let \( u = u_1 \ldots u_n \) with \( u_1, \ldots, u_n \) basic terms. Then \( g(R, D) \cong R_1 \ldots R_n \) with, for \( k = 1, \ldots, n \),
   1. if \( u_k = a \) for some \( a \in T \), then \( R_k \) is a one-node rootless tree labeled \( a \),
   2. if \( u_k = \rho_l \) for some \( \rho_l \in V \), then we distinguish two cases:
      1. if \( \rho_l \) is of type 1 in \( D \), and \( f(p_j) = (D_1, \ldots, D_m) \), then
         \[ R_k \cong g((D_1), D_1) \ldots g((D_m), D_m) \],
      2. if \( \rho_l \) is of type 2 in \( D \), and \( n = \text{par}(f(\rho_l)) \), then
         \[ R_k \cong (n'R') \text{ with } \text{lbl}(n') = A \text{ and } R' \cong g(f(\rho_l), n f(\rho_l)) \];
6. For some \( D \in \mathcal{D}(G) \), let \( R = R_1 R_2 R_3 \in \text{rst}(D) \) be a rootless subtree such that for some valuation \( f \in \mathcal{F}(D \cup E, D), f(p_j) = R_2 \). Then \( g(R, D) \cong g(R_1, D)g(R_2, D)g(R_3, D) \).

We now establish the uniqueness of \( E \)-transformations.

**Theorem 4.19.** Let \( (V, T, S, P) \) be an information base scheme and let \( E \equiv \langle \rho_j \leftarrow u | D \cup E \rangle \) be an expression over \( G \). Then there exists a unique \( E \)-transformation of \( G \).

*Proof.* The proof goes by double induction. For the empty \( D \)-tree \( E \), we know that \( g(F,E) = F \). We now assume that \( g \) is uniquely defined on all pairs \((R, D)\) with \( R \) being a rootless subtree of \( D \) and the depth of \( D \) at most, say \( p \) (outer induction hypothesis). Now let \( D \) be a \( D \)-tree over \( G \) with depth \( p + 1 \). We know that \( g(F, D) = F \). We now also assume that \( g \) is uniquely defined on all pairs \((R, D)\) with \( R \) being a rootless subtree of \( D \) and the depth\(^7\) of \( R \) at most, say \( q \) (inner induction hypothesis). Now let \( R \) be a rootless subtree of \( D \) with depth \( p + 1 \). We distinguish two cases.

**Case 1.** There is no valuation \( f \in \mathcal{F}(D \cup E, D) \) for which \( f(p_j) \) is an interval of \( R \). Let \( R = (D_1, \ldots, D_n) \) with, for \( i = 1, \ldots, n \), \( D_i = n R_i \) for some \( R_i \in \text{rst}(D) \) with the depth of \( R_i \) at most \( q \). Items 2, 3, and 4 of Definition 4.18 and the inner induction hypothesis guarantee that \( g(R, D) \) is uniquely defined.

**Case 2.** For some valuation \( f \in \mathcal{F}(D \cup E, D) \), \( f(p_j) \) is an interval of \( R \). By Lemma 4.17, we know this \( f \) is unique. Let \( R = R_1 R_2 R_3 \) with \( R_2 = f(p_j) \). By item 6 of Definition 4.18, \( g(R, D) \cong g(R_1, D)g(R_2, D)g(R_3, D) \). By the first case of this proof, \( g(R_1, D) \) and \( g(R_3, D) \) are uniquely determined. Let \( u = u_1 \ldots u_n \) with \( u_1, \ldots, u_n \) basic terms. By item 5 of

---

4Recall from Definition 4.2 that \( F \) is the empty \( R \)-tree.
5The reason for this distinction is the following. If a variable is part of a basic term of type 2, then we are interested in the entire tree this basic term represents, as opposed to merely the \( R \)-tree the variable represents.
6See footnote 5.
7The depth of \( F \) is 0; the depth of another \( R \)-tree is defined as the maximum of the depths of the \( D \)-trees of which it is composed.
Definition 4.18, we know that \( g(R_2, D) \cong R_{21} \ldots R_{2n} \) for some R-trees \( R_{21}, \ldots, R_{2n} \). It remains to show that, for \( k = 1, \ldots, n \), \( R_{2k} \) is uniquely determined. To do this we distinguish five subcases.

**Subcase 1.** \( u_k = a \) with \( a \in T \). By item 5.1 of Definition 4.18, \( R_{2k} \) is unambiguously determined.

**Subcase 2.** \( u_k = \rho_i \) with \( \rho_i \) a type 1 variable in \( D \). Let \( f(\rho_i) = (D_1, \ldots, D_m) \). By item 5.2.1 of Definition 4.18, we know that \( R_{2k} \cong g((D_1), D) \ldots g((D_m), D) \). Since, by Definition 4.12, the depth of \( \rho_i \) in \( G(D) \) is at least 1, it follows that, for all \( j = 1, \ldots, m \), the depth of \( D_j \) is at most \( p \). Hence the desired conclusion follows from the outer induction hypothesis.

**Subcase 3.** \( u_k = \rho_i \) with \( \rho_i \) a type 2 variable in \( D \). Let \( n = \text{par}(f(\rho_i)) \). By item 5.2.2 of Definition 4.18, we know that \( R_{2k} \cong g(f(\rho_i), n f(\rho_i)) \). If \( n \) is not the root of \( D \), then the depth of \( n f(\rho_i) \) is at most \( p \). Hence the desired conclusion follows from the outer induction hypothesis. However, if \( n = \text{rt}(D) \), then \( D = n f(\rho_i) \), whence the depth of \( f(\rho_i) \) is at most \( p \). Moreover, by Definition 4.12, it follows that \( f(\rho_i) = (D) = R_2 = R \), whence \( n = k = 1 \) and \( p = q \). So, \( f(\rho_i) \) has also depth at most \( q \). The uniqueness of \( R_{2k} = R_{21} \) now follows from the inner induction hypothesis.

**Subcase 4.** \( u_k = (A\rho_i) \) with \( A \in V \) and \( \rho_i \) a type 1 variable in \( D \). Let \( f(\rho_i) = (D_1, \ldots, D_m) \). By item 5.3.1 of Definition 4.18, we know that \( R_{2k} \cong (n' R') \) with \( \text{lbl}(n') = A \) and \( R' \cong g((D_1), D) \ldots g((D_m), D) \). The remainder of this case is now analogous to subcase 2.

**Subcase 5.** \( u_k = (A\rho_i) \) with \( A \in V \) and \( \rho_i \) a type 2 variable in \( D \). Let \( n = \text{par}(f(\rho_i)) \). By item 5.3.2 of Definition 4.18, we know that \( R_{2k} \cong (n' R') \) with \( \text{lbl}(n') = A \) and \( R' \cong g(f(\rho_i), n f(\rho_i)) \). The remainder of this case is now analogous to subcase 3.

Using the unique transformation defined above, we finally define the result of a calculus expression.

**Definition 4.20.** Let \( (V, T, S, P) \) be an information base scheme and let \( D \) be an information base instance over \( (V, T, S, P) \). Let \( E \) be an expression over \( (V, T, S, P) \). Let \( g \) be the unique \( E \)-transformation of \( (V, T, S, P) \). Then \( E(D) \) is (the class of) the \( D \)-tree \( D' \) for which \( g((D), D) \cong (D') \), if this \( D \)-tree is in turn an information base instance, and undefined otherwise.

We conclude this section with a final example.

**Example 4.21.** Reconsider the information base instance \( D \) and the calculus expression \( E \) of Example 4.16. Recall that in Fig. 18, for each valuation \( f \), the (unique) node in \( \text{top}(f(\rho_1)) \) is marked by a square and all nodes in \( \text{top}(f(\rho_5)) \) are marked by dots (\( \rho_1 \) is the root of the expression and \( \rho_5 \) is the left-hand side of the substitution clause). The reader is invited to check that the result \( E(D) \) of applying the calculus expression \( E \) to the information base instance \( D \) indeed equals the instance of Fig. 19.

**5. Properties of the grammatical calculus.** The definition of the grammatical calculus given in the previous section raises several decidability issues. Since a full treatment of these decidability issues would go beyond the scope of the present article, we shall deal with these here only briefly, in a fairly informal manner.

The key construct of this section is that of a condition tree of a calculus expression, or more precisely, of the set of hierarchical declarations and equations of a calculus expression. We first define this notion for the case in which no equations are present.

**Definition 5.1.** Let \( G = (V, T, S, P) \) be an information base scheme, and let \( D \) be a hierarchical set of declarations.

- The condition tree \( C(D) \) of \( D \) is constructed as follows. First, initialize \( C(D) \) to a tree consisting of one node labeled with the root of \( G(D) \). Then, as long as there is a node \( n \) in \( C(D) \) labeled with a variable \( \rho_i \) that is not a leaf in \( G(D) \), i.e., for which there exists some
hierarchical declaration \( \rho_i := t_1 \ldots t_m \) in \( \mathcal{D} \), substitute \( n \) by the rootless subtree \( \{C_1, \ldots, C_m\} \) where, for \( k = 1, \ldots, m \),

1. if \( t_k = a \) with \( a \in T \), \( C_k \) consists of one node which is labeled \( a \);
2. if \( t_k = \rho_l \) with \( \rho_l \) a type 1 variable in \( \mathcal{D} \), \( C_k \) consists of one node which is labeled \( \rho_l \);
3. if \( t_k = (A \rho_l) \) with \( A \in V \) and \( \rho_l \) a type 2 variable in \( \mathcal{D} \), \( C_k \) is a two-node tree the root of which is labeled \( A \) and the leaf of which is labeled \( \rho_l \).

Let \( E \leftrightarrow [\rho_j \leftarrow u \mid \mathcal{D}] \) be an expression over \( \mathcal{G} \) without equations. Then \( C(E) = C(\mathcal{D}) \).

Example 5.2. Let \( \mathcal{D} \) be the set of hierarchical declarations in Example 4.16. The condition tree \( C(\mathcal{D}) \) of \( \mathcal{D} \) is shown in Fig. 20.

Intuitively, the condition tree of an expression shows the pattern that must be present in an information base instance for the expression to have an action on that instance.

The notion of condition tree can be extended to the case in which both hierarchical declarations and equations are present. We shall informally explain how. Thereto, sup-
pose that in addition to the assumptions of Definition 5.1 we have a set of equations $E$ over $\mathcal{G}$.

If the equations only involve variables that are leaves in $\mathcal{C}(D)$ and hence also in $\mathcal{G}(D)$, then the condition tree $\mathcal{C}(D \cup E)$ of $D$ and $E$ is straightforwardly constructed from $\mathcal{C}(D)$ by equating labels according to the equations in $E$. For instance, if $E$ is the expression in Example 4.16, then $\mathcal{C}(E)$ is obtained from the condition tree in Example 5.2, Fig. 20, by equating $\rho_9$ and $\rho_{13}$.

In the perhaps more pathological case in which variables are equated that are not necessarily leaves of $\mathcal{C}(D)$, the construction is somewhat more involved. To illustrate this, let $D$ be the hierarchical set of declarations in the expression of Example 4.16 and consider the equation $\rho_5 = \rho_{11}$. The rootless subtrees corresponding to $\rho_5$ and $\rho_{11}$ in $\mathcal{C}(D)$ in Example 4.16 are shown in Fig. 21 left and right, respectively.

![Fig. 21. The rootless subtrees corresponding to $\rho_5$ and $\rho_{11}$.](image)

We can now try to expand in both rootless subtrees the nodes corresponding to variables in a minimal way such that the R-trees become isomorphic. In our example, this can be achieved by substituting $\rho_{14}$ by $D$ and equating $\rho_9$ and $\rho_{13}$. The R-tree thus obtained is actually the most general unifier of $\rho_5$ and $\rho_{11}$. Finally, the condition tree $\mathcal{C}(D \cup \{\rho_5 = \rho_{11}\})$ of $D$ and $\{\rho_5 = \rho_{11}\}$ is obtained by substituting $\rho_5$ and $\rho_{11}$ by their most general unifier.

Of course this most general unifier need not exist, e.g., because the two rootless subtrees involved are incompatible. Also, the unification process might result in an infinite tree. The latter case would occur if we tried to compute $\mathcal{C}(D \cup \{\rho_2 = \rho_4\})$. During the unification process we would find that $\rho_8$ must be equated to an R-tree strictly containing $\rho_8$ as a rootless subtree, whence the resulting tree would be infinite. Finally, it is possible that the resulting tree is not a *legal* D-tree in the sense that it contains sibling nodes labeled by the same attribute, whence the tree cannot be considered as an information base instance over some scheme.

Each time the construction of a condition tree requires an impossible unification process or a unification process resulting in an infinite tree, or does not result in a legal D-tree, we say that the condition tree is undefined. Obviously, this property is decidable. The undefinedness of a condition tree corresponds to the fact that any associated expression is not applicable to any information base instance.

Several decidability results regarding the grammatical calculus can be proved by using condition trees. The techniques employed in these proofs in essence come down to applying expressions to their own condition tree and are therefore reminiscent of similar techniques used in the relational model for conjunctive queries [6], [21].

The first decidability result is concerned with checking whether or not a calculus expression represents the identity.

**Lemma 5.3.** Let $\mathcal{G} = (V, T, S, P)$ be an information base scheme, and let $E \equiv [\rho_j \leftarrow u \mid D \cup E]$ be an expression over $\mathcal{G}$. Then $E$ represents the identity if and only if either the condition tree of $E$ is undefined or $E(\mathcal{C}(E)) \cong \mathcal{C}(E)$.

**Proof.** Obviously, if $E$ represents the identity and the condition tree of $E$ is defined, then $E(\mathcal{C}(E)) \cong \mathcal{C}(E)$, whence the “only if.” To see the “if,” we need to distinguish two cases. If

---

8If $E \equiv [\rho_j \leftarrow u \mid D \cup E]$ is an arbitrary expression over $\mathcal{G}$, then $\mathcal{C}(E) = \mathcal{C}(D \cup E)$. 

the condition tree of $E$ is undefined, then $E$ obviously represents the identity, since it is not applicable to any information base instance. Thus suppose the condition tree of $E$ does exist and satisfies $E(C(E)) \cong C(E)$. Let $f$ be the valuation\(^9\) of $D$ and $E$ in $C(E)$. If $f$ is extended to terms in the natural way, then $E(C(E)) \cong C(E)$ is equivalent to $f(p_j) \cong f(u)$. Using this latter condition, the proof can be completed by a straightforward double induction, as in the proof of Theorem 4.19.

Using Lemma 5.3, we can prove the following necessary condition for two expressions to be equivalent.

**Theorem 5.4.** Let $\mathcal{G} = (V, T, S, P)$ be an information base scheme. Two expressions over $\mathcal{G}$ are equivalent only if either they both represent the identity or their condition trees are isomorphic upon renaming of variables.

**Proof.** Let $E_1 \equiv [\rho_j \leftarrow u_1 \mid D_1 \cup E_1]$ and $E_2 \equiv [\rho_j \leftarrow u_2 \mid D_2 \cup E_2]$ be two equivalent expressions that do not represent the identity. By Lemma 5.3, the condition trees of $E_1$ and $E_2$ exist and satisfy $E_1(C(E_2)) \cong C(E_1)$ and $E_2(C(E_2)) \cong C(E_2)$. Hence, by the equivalence of $E_1$ and $E_2$, $E_2(C(E_2)) \cong C(E_2)$. In particular, this implies there must exist a valuation of $C_1$ and $\rho_1$ in $C(E_2)$ as well as a valuation of $C_2$ and $\rho_2$ in $C(E_1)$. Using this fact, the theorem is now easily shown. \(\square\)

Given Theorem 5.4, it is now tempting to conjecture that two grammatical calculus expressions that do not represent the identity are equivalent if and only if they yield the same result when applied to their common condition tree. Unfortunately, this condition is not sufficient because of the special way in which type 2 variables in a substitution term are handled in the calculus. Example 5.5 gives a counterexample.

**Example 5.5.** Let $\mathcal{D} \cup \mathcal{E}$ be the following set of hierarchical declarations and equations (over some appropriate scheme):

$$\{\rho_1 := (A\rho_2), \rho_2 := (B\rho_3)(A\rho_4)(C\rho_5), \rho_3 := (A\rho_6), \rho_6 := \rho_7b, \rho_4 = \rho_7\}.$$

The condition tree of $\mathcal{D}$ and $\mathcal{E}$ is shown in Fig. 22.

![Fig. 22. The condition tree $C(\mathcal{D} \cup \mathcal{E})$.](image)

Now consider the expressions

$$E_1 \equiv [\rho_5 \leftarrow \rho_4 \mid \mathcal{D} \cup \mathcal{E}];$$

$$E_2 \equiv [\rho_5 \leftarrow \rho_7 \mid \mathcal{D} \cup \mathcal{E}].$$

Notice that $\rho_4$ is of type 2 while $\rho_7$ is of type 1.

Obviously, $E_1(C(\mathcal{D} \cup \mathcal{E})) \cong E_2(C(\mathcal{D} \cup \mathcal{E}))$, because these resulting trees are both obtained by substituting $\rho_5$ by $\rho_4$ in the tree of Fig. 22. Nevertheless, $E_1$ and $E_2$ are not equivalent. To see this, we replace the nodes in Fig. 22 that are labeled $\rho_4$ with the rootless subtree obtained

---

\(^9\)By the definition of condition tree, there is a unique valuation $f \in F(\mathcal{D} \cup \mathcal{E}, C(E))$ determined by $f(\rho_{\text{root}}) = (C(E))$, $\rho_{\text{root}}$ being the root of $G(D)$. 
by chopping off the root from a copy of the condition tree. The result of this modification is shown in Figure 23.

The purpose of this modification was substituting the tree rooted in the parent of $\rho_4$ by its most general unifier with a copy of the entire condition tree.

Applying expression $E_2$ to the tree in Fig. 23 results in the straightforward substitution of $\rho_5$ by the rootless subtree identified by $\Delta$, as shown in Fig. 24 bottom.

In contrast, the application of $E_1$ results in the substitution of $\rho_5$ by $\Delta'$ where $\Delta'$ is obtained from $\Delta$ by applying $E_1$ to the tree rooted in the parent of $\rho_4$. This asymmetry is due to the fact that $\rho_4$ is of type 2, while $\rho_7$ is of type 1. The result of applying $E_1$ to the tree in Fig. 23 is shown in Fig. 24 top.

In general, given an expression and its condition tree, one has two choices for each type 2 variable in the substitution tree: one can either leave the corresponding node or rootless subtree in the condition tree unchanged, or one can transform it in the sense of Example 5.5. This procedure leads to a number of trees that is potentially exponential in the number of type 2 variables in the substitution term.\footnote{In most cases, however, the number of trees obtained will be significantly smaller because the transformation described in Example 5.5 requires a unification whose result may well be undefined.} To facilitate our further discussion, we shall...
call the set of legal D-trees thus obtained the set of representative instances of the given expression.

Intuitively, the set of representative instances of a calculus expression is constructed in such a way that for every possible valuation of the expression’s set of declarations and equations in a concrete information base instance, there is a representative instance whose transformation by the expression “models” the way in which the information base instance is transformed locally.

Therefore, we conjecture that equivalence of nonidentity expressions can be decided by considering all trees in their sets of representative instances and verifying whether both expressions yield the same results for all those trees.

Finally, the set of representative instances can also be used to decide whether or not the result of a calculus expression is always defined, independent of the information base instance to which the expression is applied. Now, the result of a calculus expression applied to a concrete information base instance can only be undefined if the resulting tree is no longer an instance, i.e., if this tree contains sibling nodes labeled with the same attribute in V. By what has been said above, it suffices to apply the calculus expression to all representative instances to verify whether or not undefinedness can occur. Thereto, one has to check whether or not

1. one of the resulting trees contains sibling nodes labeled by the same attribute;
2. in one of the resulting trees it is possible to substitute a variable by a sibling attribute.

The latter case occurs precisely when a variable in a resulting instance has a sibling attribute that is not a sibling to that variable in the original representative instance. Hence we have the following theorem.

THEOREM 5.6. It is decidable whether or not the result of a calculus expression over a given information base scheme is always defined.

6. The equivalence between algebra and calculus. In §3, we presented the grammatical algebra as a query language for transforming information bases. In §§4 and 5, we introduced and discussed grammatical calculus expressions. We can now consider the grammatical calculus as the language consisting of all finite sequences of calculus expressions. Note that, in contrast to the relational calculus, we cannot hope such a sequence will always be equivalent to a single expression, since in general there is no way to combine the various condition trees of the expressions in the sequence into one single condition tree that could be used to describe the net effect of the transformation. Since each grammatical algebra operation can be expressed by a single calculus expression, the grammatical calculus will nevertheless allow a more succinct representation of queries than the algebra.

In this section, we compare the expressive power of the grammatical algebra and calculus. Inspired by the classical result in the relational model, we were able to prove their equivalence. In view of the technical complexity of this proof, we use Example 4.16 as running example throughout the proof in order to improve its readability.

THEOREM 6.1. The grammatical algebra and grammatical calculus are equivalent with regard to expressive power.

Proof. In Example 4.13, applications of all algebra operators (except for upward duplication, which is analogous to downward duplication) are expressed in the calculus. It is straightforward to generalize the techniques used in these examples. Hence the algebra can be simulated in the calculus. The more involved part of the proof consists of showing that a calculus expression can be simulated in the algebra. In order to show this, we shall simulate in the algebra the various steps needed to evaluate a calculus expression of which, without loss of generality, we assume it does not represent the identity. (This assumption is needed to guarantee, by Lemma 5.3, the existence of the expression’s condition tree, which in turn is needed to validate some of the constructions made below.)
Therefore, let $G = (V, T, S, P)$ be an information base scheme, let $D$ be an instance over $G$, and let $[\rho_j \leftarrow u \mid D \cup E]$ be a calculus expression with $\text{var}(D) = \{\rho_1, \ldots, \rho_n\}$ and $\mathcal{D} = \{\rho_i := t_i \mid i \in I\}$, $I \subseteq \{1, \ldots, n\}$. Without loss of generality, we assume that $\rho_k$ being an ancestor of $\rho_i$ in $\mathcal{G}(D)$ implies $k < I$. (Observe that Example 4.16 satisfies this requirement.)

We further assume that $\rho_1$ is the root of $\mathcal{G}(D)$ and that the unique declaration for $\rho_1$ in $D$ has the form $\rho_1 = (A\rho_2)$.

We also number the equations starting from $n + 1$: $E = \{\epsilon_{n+1}, \ldots, \epsilon_{n+1}\}$. Since, obviously, a nontrivial equation involving $\rho_1$ can never be satisfied, we may assume, again without loss of generality, that $\rho_1$ is not contained in an equation of $E$.

Let $J$ be an arbitrary set of nonnegative integers. For each $B \in V$, we assume that $B^J$ denotes an attribute; similarly, for each $a \in T$, we assume that $a^J$ denotes a constant. We also assume that $N_1, i = 1, 2, 3, \ldots$, are attributes not in $V$. Finally, we also assume that $N/J$ denotes an attribute. Informally speaking, the superscripts of the labels will be used to remember which variables can be valuated into which rootless subtrees. The $N/J$ are auxiliary attributes which will be used for copying information in $D$ from one place to another in the tree.

The proof is basically a construction that consists of the following steps:

**Step 1. Initialization.** We index all node labels in $D$ with the empty set. This is done by using parent substitution (for the attribute nodes) and permutation (for the constant nodes).

**Step 2. Determination of all valuations of $D$ in $D'$.** We shall relabel by $A^{(1)}$ all nodes $n$ for which there exists a valuation $f$ of $D$ in $D$ with $\text{top}(f(\rho_1)) = (n)$. (Remember that $A$ is the attribute in the declaration for the root $\rho_1$ of $\mathcal{G}(D)$.) Therefore, we do the following steps:

**Substep 1. Transforming $D$.** From $D$ we construct $D'$ as follows. $D'$ contains $\rho_1$ as well as one declaration for each type 2 variable. The right-hand side of this declaration contains only type 2 variables and type variables that are leaves in $D$. These right-hand sides are obtained from the original right-hand sides in $D$ by subsequent substitutions. For example, if $D$ is the set of declarations in Example 4.16, then

$$
D' = \{\rho_1 := (A\rho_2), \rho_2 := (B\rho_3)\rho_6c(A\rho_4), \rho_4 := (B\rho_1)\rho_{12}c(A\rho_8), \rho_5 := (C\rho_9)(D\rho_{10}), \rho_{10} := \epsilon, \rho_{11} := (C\rho_{13})\rho_{14}\}.
$$

Note that $D'$ actually describes the structure of the condition tree $C(D)$ (see Definition 5.1).\(^{12}\) Obviously, the restriction to $\text{var}(D')$ of a valuation of $D$ in $D$ is a valuation of $D'$ in $D$; conversely, each valuation of $D'$ in $D$ can be extended to a valuation of $D$ in $D$.

**Substep 2. Indicating all rootless subtrees in $D$ to which type 2 leaf nodes in $\mathcal{G}(D')$ can be mapped.** We shall indicate these rootless subtrees by adding to the superscripts of the labels of their parent nodes the indices of the corresponding variables. Thereto, we perform, in any order, the following operation for each type 2 leaf node $\rho_i$ in $\mathcal{G}(D')$, until no further action is possible. If there is no declaration $\rho_i := \epsilon$ in $D'$, we do $\Sigma\pi[B \rightarrow s, B^{J\cup[i]})$ for each $B \rightarrow s$ with $i \notin J$ and either $B \rightarrow s$ a production in the current scheme or $B \rightarrow \epsilon$ an attribute in the current scheme and $s = \epsilon$; if $\rho_i := \epsilon$ is in $D'$, we only do $\Sigma\pi[B \rightarrow \epsilon, B^{J\cup[i]}$.

In our example, $\rho_{10}$ is the only type 2 leaf node in $\mathcal{G}(D)$ that has a declaration with an empty right-hand side. Consequently, an index 10 must be added to the $A$-labels of leaf nodes in the current instance: there are six such nodes. The other type 2 leaf nodes of $\mathcal{G}(D)$ are $\rho_8$, $\rho_9$ and $\rho_{13}$. Consequently, an index 8 must be added to all $A$-labels and indices 9 and 13 to all $C$-labels of nodes in the current instance. The result of these operations is shown in Fig. 25.

\(^{11}\)For the time being, we ignore the equations in $E$.

\(^{12}\)Since by assumption the condition tree $C(D)$ exists, it follows that the substitutions performed cannot yield illegal terms, i.e., terms containing two basic terms of type 2 with the same attribute.
Substep 3. Iteratively building up all valuations of $\mathcal{D}$ in $\mathbf{D}$. Let $\rho_1, \ldots, \rho_m$ be (in ascending order) all type 2 variables in $\mathcal{D}'$. Let, for $1 \leq p \leq m$, $\mathcal{D}_p'$ be the set of all declarations of $\mathcal{D}'$ involving only variables whose index is at least $i_p$. We will relabel the nodes in the current instance in such a way that, for each $p = 1, \ldots, m$,

the superscript of the label of a node $n$ contains the index $i_p$ if and only if there exists a valuation $f$ of $\mathcal{D}_p'$ in $\mathbf{D}$ with $\text{par}(f(\rho_{i_p})) = n$ (condition $(p)$).

Note that, by the construction in the previous step, the current instance already satisfies all conditions $(p)$ for which $\rho_{i_p}$ is a leaf node in $\mathcal{G}(\mathcal{D}')$. By a downward iterative procedure, we now enforce the conditions $(p)$ for which $\rho_{i_p}$ is not a leaf node in $\mathcal{G}(\mathcal{D}')$. Thereto, we perform the following operation for those $p := m$ down to 1 for which $\rho_{i_p}$ is an internal node of $\mathcal{G}(\mathcal{D}')$.

Let $\rho_{i_p} := t_1 \ldots t_l$ be the declaration for $\rho_{i_p}$ in $\mathcal{D}'$ with the $t_q, q = 1, \ldots, l$, basic terms. Let $\rho_k := \ldots (B \rho_{i_p}) \ldots$ be the declaration in $\mathcal{D}'$ containing $\rho_{i_p}$ in its right-hand side. Then, in any order and until no further action is possible, we do $\Sigma \pi [B \rightarrow s_1 \ldots s_l, B^{(i_p)}]$ for each production $B \rightarrow s_1 \ldots s_l$ in the current scheme in which $i_p \notin J$ and, for $q = 1, \ldots, l$, the $s_q$ have the following form:

$$s_q = \begin{cases} a^B & \text{if } t_q = a, a \in T; \\ C^K & \text{if } t_q = (C \rho_r), C \in V, \rho_r \in \text{var}(\mathcal{D}'), \text{ and } r \in K \end{cases}$$

($s_q$ is arbitrary if $t_q$ is of type 1).

The current instance for our example is now as shown in Fig. 26.

Observe that, by necessity, $i_1 = 2$. Hence the superscript of the label of a node $n$ contains the index 2 if and only if there exists a valuation $f$ of $\mathcal{D}' - \{\rho_1 \leftarrow (A \rho_2)\}$ in $\mathbf{D}$ with $\text{par}(f(\rho_1)) = n$.

Substep 4. Indicating all rootless subtrees of $\mathbf{D}$ to which $\rho_1$ can be mapped by a valuation of $\mathcal{D}$. We will add an index 1 to the superscript of the label of all nodes $n$ for which there exists a valuation $f$ of $\mathcal{D}'$ in $\mathbf{D}$ with $\text{top}(f(\rho_1)) = (n)$, or, equivalently, for which there exists a valuation $f$ of $\mathcal{D}'$ in $\mathbf{D}$ with $\text{top}(f(\rho_1)) = (n)$. Now, a valuation $f$ of $\mathcal{D}' - \{\rho_1 \leftarrow (A \rho_2)\}$ can be extended to a valuation of $\mathcal{D}$ in $\mathbf{D}$ if and only if $\text{par}(f(\rho_2))$ is labeled $A$. Therefore, in any order and until no further action is possible, we have to perform $\Sigma \pi [A^J \rightarrow s, A^{J \cup \{1\}}]$ for each $A^J \rightarrow s$ with $2 \in J$, $1 \notin J$, and either $A^J \rightarrow s$ a production in the current scheme or $A^J$ an attribute in the current scheme and $s = \varepsilon$. 
Substep 5. Cleaning up. Using parent substitution, we rename all node labels $A^J$ with $1 \in J$ to $A^{[1]}$ and all other node labels $B^J$ with $B \in V$ and $1 \not\in J$ to $B^{[0]}$.

All nodes $n$ labeled $A$ for which there exist a valuation $f$ of $\mathcal{D}$ in $\mathbf{D}$ with $\text{top}(f(n)) = (n)$ are now indexed by $\{1\}$; all other nodes are indexed by the empty set.

The current instance for our example is now as shown in Fig. 27. There are four valuations of $\mathcal{D}$ in $\mathbf{D}$.

Step 3. Evaluation of all type 2 variables under the valuations of $\mathcal{D}$ in $\mathbf{D}$. Once again, let $\rho_1, \ldots, \rho_m$ be the type 2 variables in $\mathcal{D}'$ (or, equivalently, in $\mathcal{D}$). We will relabel the nodes in the current instance in such a way that, for each $p = 1, \ldots, m$,

the superscript of the label of a node $n$ contains the index $i_p$ if and only if there exists a valuation $f$ of $\mathcal{D}'$ in $\mathbf{D}$ with $\text{par}(f(\rho_p)) = n$ (condition $(p')$).
Since we will need approximately the same procedure on several other occasions in the following parts of this proof, we will describe it in slightly more general terms than needed right now. Recalling that $\rho_1 = \rho_2$, we can easily satisfy condition (1') by doing, in any order, until no further action is possible, the parent substitution $\Sigma \tau[B^{[1]} \rightarrow s, B^{[1,2]}]$ for each $B^{[1]} \rightarrow s$ with either $B^{[1]} \rightarrow s$ a production in the current scheme or $B^{[1]}$ an attribute in the current scheme and $s = e$.\footnote{Note that, at this stage of the proof, the attribute $B$ will always equal $A$, the attribute in the declaration $P_1 \rightarrow (A^{[2]} B^{[1]})$.}

By an upward iterative procedure, we now enforce conditions (2')-(m').

Thereto, we perform the following operation for $p := 2$ up to $m$. Let $\rho_k := \ldots (B \rho_p) \ldots$ be the declaration in $D'$ containing $\rho_k$ in its right-hand side. Then, in any order and until no further action is possible, we do $\Sigma \chi[C^k \rightarrow s_1 B^j s_2, B^j, B^{[1,j]}]$ for each production $C^k \rightarrow s_1 B^j s_2$ in the current scheme with $C \in V, k \in K$, and $i_p \notin J$.

The current instance of our example is now as shown in Fig. 28.

**Fig. 28. Evaluation of all type 2 variables under the valuations of $D$ in $D$.**

**Step 4. Evaluation of all type 1 variables under the valuations of $D$ in $D$.** We will relabel the nodes in the current instance in such a way that for each type 1 variable $\rho_1$, the superscript of the label of a node $n$ contains the index $i$ if and only if there exists a valuation $f$ of $D$ in $D$ with $\text{top}(f(\rho_1))$ containing $n$. Therefore, we do the following steps:

**Substep 1. Transforming $D$.** From $D$ we construct $D''$ as follows. $D''$ contains one declaration for each type 1 variable. This type 1 variable is contained in the right-hand side of the declaration. The left-hand side is a type 2 variable. As for $D'$, the declarations of $D''$ are obtained by subsequent substitutions.\footnote{Again, these substitutions cannot yield illegal terms.} For example, in our example,

\[
D'' = \{\rho_2 := \rho_3 c(A \rho_4), \rho_2 := (B \rho_3) \rho_6 c(A \rho_4), \rho_4 := \rho_7 c(A \rho_8),
\rho_4 := (B \rho_11) \rho_12 c(A \rho_8), \rho_11 := (C \rho_13) \rho_14\}.
\]

Clearly, a valuation of $D$ in $D$ is also a valuation of $D' \cup D''$ in $D$; conversely, the extension of a valuation of $D'$ in $D$ to var($D$) is a valuation of $D$ in $D$ if and only if it is a valuation of $D' \cup D''$ in $D$. 
Substep 2. Relabeling. In order to enforce for each valuation of \( D \) in \( \mathbf{D} \) and for each type 1 variable \( \rho_i \) that each node in \( \text{top}(f(\rho_i)) \) contains the index \( i \) in the superscript of its label, we perform the following operations. For later use, they will be once again described in slightly more general terms than required right now. Let \( \rho_k := t_1 \ldots t_l \) be the unique declaration in \( \mathcal{D}' \) containing \( \rho_i \) in its right-hand side. Let in the above declaration \( t_q, 1 \leq q \leq l \), be the basic term with \( t_q = \rho_i \). Let \( \rho_r := \ldots (B \rho_k) \ldots \) be the unique declaration in \( \mathcal{D}' \) containing \( \rho_k \) in its right-hand side. Then, by step 3 of the construction in this proof, each node \( n \) occurring in \( f(\rho_i) \) for some valuation of \( D \) in \( \mathbf{D} \) is the child of a node labeled \( B^J \) with \( k \in J \).

We first relabel all nodes occurring in \( f(\rho_i) \) for some valuation of \( D \) in \( \mathbf{D} \) labeled by a constant. Thereto, we introduce the following notation. Let \( s \in \ldots \alpha_{w} \) be an arbitrary word over the current attributes and constants. (For \( v \in \ldots \alpha_{w} \), \( \alpha_{v} \) is an attribute or a constant.) Then we denote by \( \tilde{s} \) the word \( \tilde{s} = \ldots \alpha_{v} \cdot \tilde{s} \cdot \alpha_{v} \cdot \tilde{s} \cdot \alpha_{v} \cdot \tilde{s} \cdot \alpha_{v} \cdot \tilde{s} \cdot \alpha_{v} \cdot \tilde{s} \cdot \alpha_{v} \cdot \tilde{s} \cdot \alpha_{v} \cdot \tilde{s} \cdot \alpha_{w} \) with, for \( v \in \ldots \alpha_{w} \),

\[
\tilde{s} = \begin{cases} 
\alpha^K_{\nu(i)} & \text{if } \alpha_v = a^K, a \in T; \\
\alpha_v & \text{otherwise.}
\end{cases}
\]

The relabeling of the constant nodes is now achieved by performing, in any order and until no further action is possible, the permutation

\[
\Pi[B^J \to \ldots s_{q-1}s_q \ldots s_{l+l}^1s_{l+1}^1 s_{l+1}^1 \ldots s_{l+1}^1]
\]

for each production \( B^J \to \ldots s_{q-1}s_q \ldots s_{l+l} \) in the current instance with \( k \in J \), in which for \( r = 1, \ldots, q - 1, q + 1, \ldots, l \),

\[
s_r = \begin{cases} 
\alpha^K & \text{if } t_r = a, a \in T; \\
C^K & \text{if } t_r = (C \rho_k), C \in V,
\end{cases}
\]

for some set \( K \) of nonnegative integers, \( s_{l+1} \) is a (possibly empty) string of symbols \( N^K_a, \pi_1 \) and \( s_q \) is an arbitrary word in which the index \( i \) does not yet occur as a superscript. Finally, the attribute-labeled nodes occurring in \( f(\rho_i) \) for some valuation of \( D \) in \( \mathbf{D} \) are labeled by performing, in any order and until no further action is possible, the child substitution \( \Sigma_X[B^J \to \ldots s_{q-1}s_q \ldots s_{l+l}^1s_{l+1}^{1\ldots l+l}] \) for each production \( B^J \to \ldots s_{q-1}s_q \ldots s_{l+l} \) in the current instance with \( k \in J \) and \( i \not\in L \) and in which, for \( r = 1, \ldots, q - 1, q + 1, \ldots, l, l + 1, s_r \) is as above while \( s_{q-1} \) and \( s_{q+2} \) are arbitrary.

The current instance of our example is now as shown in Fig. 29.

Step 5. Duplication of rootless subtrees corresponding to variables occurring in equations of \( \mathcal{E} \). We will now take into account the set of equations \( \mathcal{E} = \{e_{n+1}, \ldots, e_{n+1} \} \). For each variable \( \rho_i \) occurring in an equation of \( \mathcal{E} \) and each valuation \( f \) of \( D \) in \( \mathbf{D} \), we will add to \( \text{chln}(n), n \) being the node with \( \text{top}(f(\rho_i)) = (n) \), a node \( m \) with \( \text{chtrs}(m) \) a “duplicate” of \( f(\rho_i) \). Therefore, we do the following steps, keeping in mind that the root \( \rho_1 \) of \( \mathcal{G}(D) \) does not occur in \( \mathcal{E} \):

Substep 1. Duplicating rootless subtrees corresponding to type 1 variables. For each type 1 variable \( \rho_i \) occurring in \( \mathcal{E} \), we perform the following procedure. Let \( \rho_k := \ldots \rho_i \ldots \) be the unique declaration in \( \mathcal{D}' \) containing \( \rho_i \) in its right-hand side and let \( \rho_r := \ldots (B \rho_k) \ldots \) be the unique declaration in \( \mathcal{D}' \) containing the type 2 variable \( \rho_k \) in its right-hand side.

In order to be able to apply sidewise duplication on rootless subtrees \( f(\rho_i), f \) a valuation of \( \mathcal{D} \) and \( \mathcal{E} \) in \( \mathbf{D} \), we first have to insert a node \( p \) satisfying \( \text{chln}(p) = \text{top}(f(\rho_i)) \). So we do, in any order and until no further action is possible, the node insertion \( N_{(J)}[B^J \to \ldots s_1s_2s_3, s_1 s_2] \) for each \( B^J \to \ldots s_1s_2s_3 \) with (i) \( k \in J \), (ii) either \( B^J \to s_1s_2s_3 \), a production in the current scheme or \( B^J \) an attribute in the current scheme and \( s_1s_2s_3 = \epsilon \), and (iii) \( s_{l+1} = \epsilon \).

\textsuperscript{15}Recall that \( N_a \not\in V \) are some of the auxiliary attributes introduced at the beginning of this proof. At this stage of the proof, however, the information base scheme does not yet contain auxiliary attributes, whence by necessity \( s_{l+1} = \epsilon \).
Substep 2. Duplicating rootless subtrees corresponding to type 2 variables. For each type 2 variable \( \rho_i \) occurring in \( \mathcal{E} \), we perform the following procedure. Let \( \rho_k = \ldots (B \rho_i) \ldots \) be the unique declaration in \( \mathcal{D}' \) containing \( \rho_i \) in its right-hand side.

First, we do, in any order and until no further action is possible, the node insertion \( \text{N}_i[B \rightarrow s_i] \) for each \( B_i \) with \( J, s_i \) not containing \( N_i \), and either \( B \rightarrow s \) a production in the current scheme or \( B \rightarrow s \) an attribute in the current scheme and \( s_i \), immediately followed by the sidewise duplication \( \text{A}_i[B \rightarrow N_i] \) and the node merging \( \text{N}_i[B \rightarrow N_i] \).

The current instance of our example is now as shown in Fig. 30.

Substep 3. Eliminating the undesired side effects of the duplication. For each variable \( \rho_i \) occurring in an equation of \( \mathcal{E} \) and for each valuation of \( \mathcal{D} \) in \( \mathcal{D} \), it is now our intention to propagate upward the “duplicate” of \( f(\rho_i) \), the parent of which is labeled \( N_i \), until that parent node becomes a child of the node \( n \) with \( \text{top}(f(\rho_i)) = (n) \). Unfortunately, we may run into trouble if the duplicate of \( f(\rho_i) \) contains a “copy of the duplicate” of \( g(\rho_k) \), \( g \) another valuation of \( \mathcal{D} \) in \( \mathcal{D} \) and \( \rho_k \) an arbitrary variable, without containing a corresponding “copy” of \( g(\rho_k) \). Such copies of duplicates will be called undesired copies of duplicates.

For example, in our example, “copies of duplicates” occur. For instance, if \( f \) is the valuation of \( \mathcal{D} \) in \( \mathcal{D} \) for which \( f(\rho_1) = (\mathcal{D}) \) and \( g \) is the valuation of \( \mathcal{D} \) in \( \mathcal{D} \) for which \( g(\rho_1) \) is the leftmost subtree of \( \mathcal{D} \) of which the root is labeled \( A \) (see Fig. 29), then the rightmost occurrence of \( \Delta_1 \) in Fig. 30 is the duplicate of \( f(\rho_1) \). Clearly, the leftmost occurrence of \( \Delta_1 \) in Fig. 30 contains \( g(\rho_9) \) (which is empty) as well as the duplicate of \( g(\rho_9) \). Hence the

---

Observe that if \( f(\rho_i) \) is the empty rootless subtree, then, by the first operation, one \( N_i[s_i] \)-labeled leaf node is inserted in an arbitrary place between the children of the corresponding \( B_i \)-node, not necessarily corresponding with the place of \( \rho_i \) in the right-hand side of the declaration \( \rho_i = \ldots \rho_i \ldots \) in \( \mathcal{D}' \). Fortunately, this nondeterminism gets immediately eliminated by the subsequent sidewise duplication and node merging.
duplicate of \( f(\rho_{13}) \) contains a copy of the duplicate of \( g(\rho_9) \). However, the duplicate of \( f(\rho_{13}) \) also contains a copy of \( g(\rho_1) \) (namely the duplicate of \( f(\rho_{13}) \) itself). Such copies of duplicates do not cause problems later; in fact, we will need them in order to generate the correct result. Undesired copies of duplicates do not occur in our example, and, as a consequence, the operations described below will not alter the instance in Fig. 30. An example in which undesired copies of duplicates do occur is given in the Appendix. There, it is also shown why these undesired copies of duplicates cause trouble.

We now remove undesired copies of duplicates as follows. First, we remove all indices except the index 1 from the superscripts of the node labels using the cleaning-up procedure in step 2.5 of this construction. Then, we reintroduce the other indices by repeating steps 3 and 4. The way the auxiliary attributes \( N^K_{u} \) are treated there prevents the indexing to be propagated downward through the parent nodes of duplicates.

Next, we will add an index 0 to the \( N^K_{u} \) corresponding to undesired copies of duplicates. They are recognized as follows. For each type 1 variable \( \rho_i \) occurring in \( \mathcal{E} \), we do the child substitution \( \Sigma_X[B^J \rightarrow s_1 N^K_u s_2, N^K_u, N^K_u \cup \{0\}] \) for all productions \( B^J \rightarrow s_1 N^K_u s_2 \) in the current

---

17 Of course, *desired* copies of duplicates can in turn contain *undesired* "copies of copies of duplicates," and so on, down to arbitrarily deep levels in the information base instance. While we choose not to complicate this already involved proof any further by explaining the problem only at the highest level where it can occur, the procedure described here is nevertheless general and removes undesired copies at any level in the information base instance.

18 Note that, even if step 5.3 is initiated with an instance in which the superscripts of auxiliary attributes are arbitrary finite sets of nonnegative integers, we always have at this point that either \( K = \{1\} \) or \( K = \emptyset \), so \( K \) never contains 0.
scheme for which the superscript of no symbol in $s$ contains the index $i$. Similarly, for each type 2 variable $\rho_i$ occurring in $E$, we do the child substitution $\Sigma X[B^J \rightarrow s_1 N^K_u s_2, N^K_u, N^K_{u+\emptyset}]$ for all productions $B^J \rightarrow s_1 N^K_u s_2$ in the current scheme with $i \not\in J$. Finally, undesired copies of duplicates are now removed by the node deletions $N^\delta[N^\emptyset_k]$ with $N^\emptyset_k$ in the current scheme, $N_u$ an auxiliary attribute, and $0 \in K$.

If for some type 1 or type 2 variables $\rho_i$ and $\rho_k$ and for some valuation $f$ of $D$ in $D$, $f(\rho_i)$ and $f(\rho_k)$ are isomorphic, then all their respective duplicates are also isomorphic, by Theorem 3.13. In general, some of these duplicates will have disappeared and some others will have been “trimmed” after step 5.3. However, the operations in step 5.3 preserve isomorphism.

Substep 4. Propagating duplicates upward in the instance. For each variable $\rho_i$ occurring in an equation of $E$ and for each valuation of $D$ in $D$, we will now move upward the “duplicate” of $f(\rho_i)$, the parent of which is labeled $N_1^\emptyset$, until that parent node becomes a child of the node $n$ with $\text{top}(f(\rho_i)) = (n)$. This is achieved by performing, in any order and until no further action is possible, the upward duplication $\Delta_U[C^K \rightarrow s_1 D^K s_2, D^K \rightarrow s_3 N^\emptyset_k s_4, N^\emptyset_k, N^\emptyset_{k+n}]$, immediately followed by the node deletion $N^\delta[N^\emptyset_k]$, and this for all productions $C^K \rightarrow s_1 D^K s_2$ and $D^K \rightarrow s_3 N^\emptyset_k s_4$ in the current scheme with $k \div n + 2 \leq d(\rho_k \mod n)$. As a consequence of this condition, duplicates of variables of depth 1 in $G(D)$ are not moved upward; indeed, they already are in their right position.

Finally, in order to be able to immediately recognize the variable to which an $N_1^\emptyset$-labeled node corresponds, we perform, in any order and until no further action is possible, the child substitution $E^C[A \rightarrow s, N, N, \text{chtrs}(m)]$ for each production $A \rightarrow s$ in the current scheme with $n > k$.

For each variable $\rho_i$ occurring in an equation of $E$ and for each valuation of $D$ in $D$ the $A$-labeled node $n$ with $\text{top}(f(\rho_i)) = (n)$ has a child $m$ labeled $N_1^\emptyset$ with $\text{chtrs}(m)$ a “duplicate” of $f(\rho_i)$. Because of Theorem 3.13, it suffices to compare these “duplicates” in order to decide the isomorphism of the original $f(\rho_i)$’s.

The current instance of our example is now as shown in Fig. 31.

Observe that the $N_1^\emptyset$-labeled nodes always occur to the right of each of their siblings labeled by another symbol.

Step 6. Determination of all valuations of $D$ and $E$ in $D$. We will relabel by $A^{[1]}$ all nodes $n$ for which there exists a valuation $f$ of $D$ and $E$ in $D$ with $\text{top}(f(\rho_1)) = (n)$. (The superscript of the label of all other nodes will be replaced by the empty set.) Therefore, we do the following steps:

Substep 1. Indicating the equations of $E$ satisfied by the valuations of $D$ in $D$. For each equation $e_k \equiv \rho_k_1 = \rho_k_2$ in $E$, $n + 1 \leq k \leq n + l$ and for each valuation $f$ of $D$ in $D$, we will add the index $k$ to the superscript of the label of the node $n$ with $\text{top}(f(\rho_1)) = (n)$ if and only if $f$ satisfies $e_k$, i.e., if $f(\rho_k_1)$ and $f(\rho_k_2)$ are isomorphic. This is achieved by performing, in any order and until no further action is possible, the parent equality substitution $\Sigma \sigma[A^J \rightarrow s, N^\emptyset_k, N^\emptyset_{k'} A^{\emptyset k}]}$ for all productions $A^J \rightarrow s$ in the current scheme with $N^\emptyset$ and $N^\emptyset_{k'}$ in $s$ and $k \not\in J$.

In our example, we only have one equation: $e_{11} \equiv \rho_9 = \rho_{13}$. Three of the four valuations of $D$ in $D$ satisfy this equation. Their corresponding $A$-nodes see an index 15 added to the superscript of their label. The current instance in our example is now shown in Fig. 32.

Substep 2. Relabeling and cleaning up. In any order and until no further action is possible, we perform $\Sigma \pi[A^J \rightarrow s, A^{[1]}]$ for each production $A^J \rightarrow s$ in the current scheme with $(n + 1, \ldots, n + l) \subseteq J$.

Using parent substitution, we now rename all attribute node labels $B^J$ with $B \in V$ and $1 \not\in J$ to $B^\emptyset$; using permutation, we rename all constant node labels $a^J$ with $a \in T$ by $a^\emptyset$. Finally, we remove all $N^\emptyset$, as well as the subtrees of which they are the root, using node deletion.
Fig. 31. Duplication of rootless subtrees corresponding to variables occurring in equations of $E$.

Fig. 32. The result of marking the equations of $E$ satisfied by the valuations of $D$ in $D$. 
Now, all nodes \( n \) for which there exists a valuation \( f \) of \( \mathcal{D} \cup \mathcal{E} \) in \( \mathbf{D} \) with \( \text{top}(f(P)) = (n) \) are labeled \( A^{(1)} \); all other nodes are labeled with an attribute or a constant indexed by the empty set.

Comparing the instances obtained at the end of step 2 and this step respectively, we have now eliminated all valuations of \( \mathcal{D} \) in \( \mathbf{D} \) that are not valuations of \( \mathcal{D} \) and \( \mathcal{E} \) in \( \mathbf{D} \). In our example, one of the valuations identified in Fig. 27, namely that for which the corresponding \( A \)-node did not receive an index 15 in Fig. 32, is eliminated.

**Step 7. Evaluation of all variables under the valuations of \( \mathcal{D} \) and \( \mathcal{E} \) in \( \mathbf{D} \).** We will relabel the nodes in the current instance in such a way that

- for each type 2 variable \( \rho_i \), the superscript of the label of a node \( n \) contains the index \( i \) if and only if there exists a valuation \( f \) of \( \mathcal{D} \) and \( \mathcal{E} \) in \( \mathbf{D} \) with \( \text{par}(f(P)) = n \);
- for each type 1 variable \( \rho_i \), the superscript of the label of a node \( n \) contains the index \( i \) if and only if there exists a valuation \( f \) of \( \mathcal{D} \) and \( \mathcal{E} \) in \( \mathbf{D} \) with \( \text{top}(f(P)) \) containing \( n \).

Thereeto, it suffices to repeat steps 3 and 4 of the construction of this proof, starting from the current instance. In our example, the resulting instance is then as shown in Fig. 33.

---

**Step 8. Duplication of rootless subtrees corresponding to variables in \( u \).** Now that all valuations of \( \mathcal{D} \) and \( \mathcal{E} \) in \( \mathbf{D} \) have been fully specified, we can start with the simulation of the actual transformation. Thereeto, for each variable \( \rho_i \) in the substitution term \( u \) and each valuation \( f \) of \( \mathcal{D} \) and \( \mathcal{E} \) in \( \mathbf{D} \), we will add to \( \text{chln}(n) \), \( n \) being the node with \( \text{top}(f(P)) = (n) \), a node \( m \) with \( \text{chtrs}(m) \) a “duplicate” of \( f(P) \). Therefore, we do the following steps, keeping in mind that \( \rho_i \) does not occur in \( u \):

**Substep 1.** Duplicating rootless subtrees corresponding to type 1 variables. Thereeto, we simply repeat step 5.1 of the construction in this proof.

**Substep 2.** Duplicating rootless subtrees corresponding to type 2 variables. In order to duplicate type 2 variables, we cannot simply repeat step 5.2. This is because of the different ways in which type 1 and type 2 variables are treated in Definition 4.18. Indeed, if \( \rho_i \) is a variable of type 1 in \( u \), \( f \) is a valuation of \( \mathcal{D} \) and \( \mathcal{E} \) in \( \mathbf{D} \) with \( f(P) = (D_1, \ldots, D_m) \), and \( g \) is an \( E \)-transformation, then we have to compute \( g((D_1), (D_1)) \ldots g((D_m), (D_m)) \). If, on the other hand, \( \rho_i \) is a type 2 variable, we have to compute \( g(f(P), p(f(P))) \), where \( p = \text{par}(f(P)) \).

Therefore, if \( \rho_i \) is a type 2 variable in \( u \) and \( f \) is a valuation of \( \mathcal{D} \) and \( \mathcal{E} \) in \( \mathbf{D} \), we will duplicate the completion of \( f(P) \) with its parent node, rather than \( f(P) \) itself. Thereeto, we
perform the following procedure for each type 2 variable $\rho_i$ in $u$. Let $\rho_k = \ldots (B \rho_i) \ldots$ be the unique declaration in $D'$ containing $\rho_i$ in its right-hand side.

First, we do, in any order and until no further action is possible, the node insertion $N_i[B \rightarrow s, N_i^J] \rightarrow (B \rho_i) \rightarrow (s, N_i^J)$ for each $B \rightarrow s$ with $i \in J$, $s$ not containing $N_i^J$, and either $B \rightarrow s$ a production in the current scheme or $B \rightarrow s$ an attribute in the current scheme and $s = \varepsilon$, immediately followed by the sidewise duplication $\Delta \sigma[B \rightarrow N_i^J, N_i^J, N_i^J \rightarrow s]$ and the node merging $N_i[B \rightarrow N_i^J, N_i^J, N_i^J \rightarrow s]$.

In comparison with step 5.2, we now also duplicated the superscript of the label of the parent node of $f(\rho_i)$.

**Substep 3. Eliminating the undesired side effects of the duplication.** We repeat step 5.3 of the construction in this proof.

**Substep 4. Propagating duplicates upward in the instance.** We repeat step 5.4 of the construction in this proof.

For each variable $\rho_i$ in the substitution term $u$ and for each valuation of $\mathcal{D}$ in $D$, the $A$-labeled node $n$ with $\text{top}(f(\rho_i)) = (n)$ has a child $m$ labeled $N_i^J$ for some $J \subseteq \{1, \ldots, n\}$ with $\text{chtrs}(m)$ a “duplicate” of $f(\rho_i)$.

The current instance of our example is now as shown in Fig. 34.

![Fig. 34. Duplication of rootless subtrees corresponding to variables in u.](image-url)

Observe that the $N_i^J$-labeled nodes always occur to the right of each of their siblings labeled by another symbol.

**Step 9. Downward propagation of duplicates to the place where they have to be inserted.** If $\rho_j$, the left-hand side of the substitution clause $\rho_j \leftarrow u$, equals $\rho_1$, the root of $\mathcal{G}(D)$, then no alterations are made.

Otherwise, for each variable $\rho_i$ in $u$ and for each valuation of $\mathcal{D}$ and $\mathcal{E}$ in $D$, we will move downward the “duplicate” of $f(\rho_i)$ until it becomes a “sibling” of $f(\rho_j)$. Therefore, we do the following steps:
Substep 1. Identifying the paths for the downward propagation of duplicates. We will identify these paths with the sequence of variables \( \rho_{q_1}, \ldots, \rho_{d(q_j)} \) where, for \( 1 \leq l \leq d(p_j) \), \( \rho_{q_l} \) is the unique ancestor of \( p_j \) in \( G(D) \) that is of type 2 and has depth \( l \).

Remembering that \( \rho_1 := (A \rho_2) \) is the declaration of the root of \( G(D) \), we always have \( \rho_{q_1} = \rho_2 \). In our example, we have \( \rho_1 = \rho_5 \), \( d(\rho_5) = 2 \), and \( \rho_{q_1} = \rho_2 \), \( \rho_{q_2} = \rho_5 \).

Substep 2. Propagating duplicates downward in the instance. For each variable \( \rho_i \) in \( u \) and for each valuation of \( D \) and \( E \) in \( D \), we will now move downward the “duplicate” of \( f(\rho_i) \) along the path identified in the previous step, until it becomes a “sibling” of \( f(p_j) \), i.e., until it has the same parent node as \( f(p_j) \), or, equivalently, until it has the same parent node as \( f(\rho_{d(q_j)}) \).

This is achieved by performing, in any order and until no further action is possible, the downward duplication \( \Delta \delta[B^k \rightarrow s_1 C^l s_2 N_i^j s_3, C^l \rightarrow s_4, N_i^j, N_{k+1}^j] \) immediately followed by the node deletion \( N \delta[N_i^j] \), for each production \( B^k \rightarrow s_1 C^l s_2 N_i^j s_3 \) in the current scheme with \( C \in V \), \( k \div n + 2 \leq d(p_j) \), and \( q_{k \div n + 1} \in L \) (whence \( q_{k \div n + 2} \in K \)), and for each \( C^l \rightarrow s_4 \) with either \( C \in V \) or \( s_4 \in E \).

Finally, in order to be able to immediately recognize the variable to which an \( N_i^j \)-labeled node corresponds, we perform, in any order and until no further action is possible, the child substitution \( \Sigma_X[B^k \rightarrow s_1 N_i^j s_2, N_{k \mod n}] \) for each production \( B^k \rightarrow s_1 N_i^j s_2 \) with \( q_{d(q_j)} \in K \) and \( k > n \).

The current instance of our example is now as shown in Fig. 35.

![Figure 35](image)

Observe that step 9.2 does not produce alterations if \( l = 1 \), i.e., if \( \rho_j \equiv \rho_2 \). Indeed, in that case, all duplicates already have the same parent node as \( f(p_j) \equiv f(p_2) \), namely the \( A \)-labeled node \( n \) with \( \text{top}(f(\rho_1)) = (n) \).
Step 10. The actual transformation in the case when in the substitution clause $\rho_j \leftarrow u$, $\rho_j \equiv \rho_1$. We distinguish two cases.

Case 1. The substitution clause is $\rho_1 \leftarrow \varepsilon$. Then, in any order and until no further action is possible, we do the node deletion $N\delta(A^j)$ for each attribute in the current scheme with $1 \in J$. Finally, we remove the superscripts from all labels using parent substitution (for the attribute nodes) and permutation (for the constant nodes).

The resulting instance is $[\rho_1 \leftarrow \varepsilon | D \cup E](D)$.

Case 2. The substitution clause has the form $\rho_1 \leftarrow (B\rho_i)$ with $B \in V$ and $\rho_i \in \text{var}(D)$.

First, we mark the subtrees to be deleted with the index $n + 1$ by doing, in any order and until no further action is possible, the child substitution

$$\Sigma\chi(C^J \rightarrow s_1D^ks_2, D^k, D^k \cup[n+1])$$

for each production $C^J \rightarrow s_1D^ks_2$ in the current scheme with $1 \in J$ (whence $C = A$ or $C = N_1$), $n + 1 \not\in K$, and $D \neq N_1$. Then, we actually remove these subtrees using the node deletions $N\delta(D^k)$ for all attributes $D$ in the current scheme with $n + 1 \in K$. Next, we remove the auxiliary attributes by applying, in any order and until no further action is possible, the node merging $N\mu[A^J \rightarrow N_i^K, N_i^K \rightarrow s]$ for all productions $A^J \rightarrow N_i^K$ and $N_i^K \rightarrow s$ in the current scheme with $1 \in J$. Finally, in any order and until no further action is possible, we perform the parent substitution $\Sigma\pi(A^J \rightarrow s, B)$ for each $A^J \rightarrow s$ with $1 \in J$ and either $A^J \rightarrow s$ a production in the current scheme or $A^J$ an attribute in the current scheme and $s = \varepsilon$.

Using parent substitution (for the attribute nodes) and permutation (for the constant nodes), we remove the index sets from all other labels.

The resulting instance is $[\rho_1 \leftarrow (B\rho_i) | D \cup E](D)$.

Step 11. The actual transformation in the case when in $\rho_j \leftarrow u$, $\rho_j$ is of type 1 in $D$. Let $k = q_d(\rho_j) = \rho_j$ and let $\rho_k := t_1 \ldots t_l$ be the unique declaration in $D''$ containing $\rho_i$ in its right-hand side. Let in the above declaration $t_q$, $1 \leq q \leq l$, be the basic term with $t_q = \rho_j$.

Let $\rho_r := \ldots (B\rho_k) \ldots$ be the unique declaration in $D''$ containing $\rho_k$ in its right-hand side.

Let $u = u_1 \ldots u_w$ with $u_1, \ldots, u_w$ basic terms. We then do the following steps:

Substep 1. Identifying and removing subtrees to be deleted. In any order and until no further action is possible, we do the node insertion $N[t[C^J \rightarrow s_1s_2s_3, s_1N_0^k s_3]]$ for each production $C^J \rightarrow s_1s_2s_3$ in the current scheme with $k \in J$, $s_2$ the substring of $s_1s_2s_3$ of all symbols containing the index $j$ in the superscript of their label, and $s_2 \neq \varepsilon$, immediately followed by the node deletion $N\delta[N_0^k]$.

Substep 2. Rearranging subtrees to be substituted. In any order and until no further action is possible, we perform the permutation $\Pi[C^J \rightarrow s_1s_2s_3, s_1s_4s_2]$ for each $C^J \rightarrow s_1s_2s_3$ with (i) $k \in J$, (ii) either $C^J \rightarrow s_1s_2s_3$ a production in the current scheme, or $C^J$ an attribute in the current scheme and $s_1s_2s_3 = \varepsilon$, and (iii)

- the superscript of no symbol in $s_1s_2s_3$ contains the index $j$;
- $s_1$ consists of $q - 1$ symbols;
- $s_2$ consists of $l - q$ symbols;
- $s_3$ only consists of symbols $N_i^K$ with $N_i$ an auxiliary attribute; and
- $s_4 = \alpha_1 \ldots \alpha_l$ with, for $v = 1, \ldots, l$,

$$\alpha_v = \begin{cases} \alpha & \text{if } u_v = \alpha; \\ N_i^K & \text{if } u_v = \rho_i \text{ and } N_i^K \text{ is in } s_3; \\ N_i^K & \text{if } u_v = (D\rho_i) \text{ and } N_i^K \text{ is in } s_3. \end{cases}$$

(Observable that $s_4$ is uniquely defined by the above conditions.)

---

19Observe that, in the above production, either $C = B$ ($B$ being the attribute in the declaration $\rho_r := \ldots (B\rho_k) \ldots$) or $C$ is an auxiliary attribute.
Substep 3. Cleaning up. For each indexed attribute $N_i^K$ in the current scheme with $\rho_i$ a variable that occurs in $u$ as a basic term of type 2, say $(D\rho_i)$, we do the parent substitution $\Sigma\pi[N_i^K \rightarrow s, D]$ for each production $N_i^K \rightarrow s$ in the current scheme. Then, for each indexed attribute $N_i^K$ in the current scheme with $\rho_i$ a variable that occurs in $u$ as a basic term of type 1, we do the node merging $N_i^K \rightarrow s$ for all productions $B_j \rightarrow s_1N_i^Ks_2$ and $N_i^K \rightarrow s_3$ in the current scheme with $k \in J$ ($B$ being the attribute in the declaration $\rho_r := \ldots (B\rho_k) \ldots$). Finally, we remove all remaining index sets from node labels using parent substitution (for the attribute nodes) and permutation (for the constant nodes).

The resulting instance is $[\rho_j \leftarrow u \mid D \cup \mathcal{E}]$.

Step 12. The actual transformation in the case that in $pj$ is of type 2 in $D$. If $\rho_j$ is of type 2 in $D$, then $q_{d(\alpha)} = j$. Let $\rho_r := \ldots (B\rho_j) \ldots$ be the unique declaration in $D'$ containing $\rho_j$ in its right-hand side. Let $u = u_1 \ldots u_w$ with $u_1, \ldots, u_w$ basic terms. We then do the following steps:

Substep 1. Identifying and removing subtrees to be deleted. In any order and until no further action is possible, we do the node insertion $N_i[C_j \rightarrow s_1s_2, N_0s_2]$ for each production $C_j \rightarrow s_1s_2$ in the current scheme with $j \in J$, $s_2$ the substring of $s_1s_2$ of all auxiliary attributes, and $s_1 \neq \varepsilon$, immediately followed by the node deletion $N_0[N_0s_2]$.

Substep 2. Rearranging subtrees to be substituted. In any order and until no further action is possible, we perform the permutation $\Pi[C_j \rightarrow s_1, s_2]$ for each $C_j \rightarrow s_1$ with $j \in J$, either $C_j \rightarrow s_1$ a production in the current scheme, or $C_j$ an attribute in the current scheme and $s_1 = \varepsilon$, and $s_2 = \alpha_1 \ldots \alpha_l$ with, for $v = 1, \ldots, l$,

$$\alpha_v = \begin{cases} a & \text{if } u_v = a; \\ N_i^K & \text{if } u_v = \rho_i \text{ and } N_i^K \text{ is in } s_1; \\ N_i^K & \text{if } u_v = (D\rho_i) \text{ and } N_i^K \text{ is in } s_1. \end{cases}$$

(Observe that $s_1$ only consists of auxiliary attributes and that $s_2$ is uniquely defined by the above conditions.)

Substep 3. Cleaning up. We repeat step 11.3 in the construction of this proof.

The resulting instance is $[\rho_j \leftarrow u \mid D \cup \mathcal{E}]$. Particularly, in our example, the operations above finally yield the information base instance shown in Fig. 19.

Finally note that all constructions in this proof were done at scheme level, i.e., they do not depend on the instance under consideration.

Hence the grammatical algebra and the grammatical calculus are equivalent. As Codd concluded for the relational model, this equivalence gives a naturalness to both languages. However, it still requires further investigation to find a precise language-independent characterization for the expressive power of the grammatical algebra and calculus.

7. Conclusions and future work. In this paper a simple model for representing the hierarchical structure in information is proposed. Two methods for querying in this data model are given and shown to be equivalent. The expressive power of these querying methods is not yet clear, however. In particular, it is not known how these methods are related to querying facilities in other data models, that can be simulated by the grammatical model. Furthermore, we are looking for a well-adapted interface that is integrated in a more general environment. It is remarkable that there seems to be no fundamental distinction between updating and querying in this model. Other aspects, such as transforming several given trees into one result tree, constraint checking, and implementation strategies, are under investigation.

Although it can be considered as an extension of the relational model, the grammatical model, because it is hierarchical in nature, is of course not suited for all database applications.

---

20Observe that, in the above production, either $C = B$ ($B$ being the attribute in the declaration $\rho_r := \ldots (B\rho_k) \ldots$) or $C$ is an auxiliary attribute.
In particular, the notion of “shared component” is difficult to express in a tree. It is therefore interesting to look for a characterization of the semantic expressiveness of the grammatical model. On the other hand, one could also look for “network-like” extensions of this model, using the theory of graph-grammars (e.g., [15]).

**Appendix.** As promised in step 5.3 of the construction in the proof of Theorem 6.1, we shall now exhibit an example in which duplication yields undesired side effects.

**Example.** Consider the information base instance \( D \) of Fig. 36 (over some appropriate scheme), let

\[ D = \{ \rho_1 := (A \rho_2), \; \rho_2 := (A \rho_3)(B \rho_4)(C \rho_5), \; \rho_3 := (A \rho_6)(B \rho_7)(C \rho_8) \}, \]

and let \( E \) be an arbitrary calculus expression involving \( D \) and the set of equations \( E = \{ \rho_6 = \rho_8 \} \).

If we apply the construction in the proof of Theorem 6.1 to the instance \( D \) up to step 4, i.e., until all valuations of \( D \) are determined and fully specified, we obtain the instance in Fig. 37.

Clearly, there are two valuations of \( D \) in \( D \). One of this, say \( f \), is determined by \( f(\rho_1) = (D) \). For the other one, say \( g \), \( g(\rho_1) \) is the leftmost component of \( \text{chtr}(\text{rt}(D)) \). Since \( \rho_6 \) and \( \rho_8 \) are the only variables occurring in an equation of \( E \), the only subtrees to be duplicated are \( f(\rho_6), \; f(\rho_8), \; g(\rho_6), \; \) and \( g(\rho_8) \). The result of applying steps 5.1 and 5.2 to the instance of Fig. 37 is shown in Fig. 38.

Now observe that the duplicate of \( f(\rho_6) \) contains a “copy” of the duplicate of \( g(\rho_6) \) as well as a “copy” of the duplicate of \( g(\rho_8) \). Clearly, the duplicate of \( f(\rho_6) \) does not contain a duplicate of \( g(\rho_1) \).
If we now would try to apply step 5.4 of the construction in the proof of Theorem 6.1 straight away, we would have to move all duplicates (and hence also all copies of duplicates) two levels upward. We leave it to the reader to verify that, for \( \rho_6 \), this would result in two nodes at the same level (more concrete, as children of the root) with the same attribute label \( N_{24}^6 \), which would imply that the result is undefined, and this is obviously not what we want. Clearly, a duplicate of \( g(\rho_6) \) does not belong at that level. Luckily, the relabeling procedure of step 5.3 reevaluates the valuations of \( \mathcal{D} \) in \( \mathcal{D} \) and prevents them from being propagated downward through nodes labeled by auxiliary attributes. The result of the relabeling is shown in Fig. 39.

The undesired node with label \( N_{6}^{9} \) (as well as the undesired node with label \( N_{8}^{9} \)) is now easily recognized from the fact that the label of its parent node no longer contains the index 6,
and hence, by step 5.3, the undesired copy of the duplicate of \( g(\rho_k) \) (as well as the undesired copy of the duplicate of \( g(\rho_k) \)) will be deleted.

**Acknowledgments.** The authors are indebted to Peter Peelman who produced the figures in this article and to Jan Van den Bussche whose critical comments on a previous version of this paper were very helpful.

**REFERENCES**


