Arbeitspapiere des SFB 340
Bericht Nr. 92, April 1997

THE EXPRESSIVITY OF TREE LANGUAGES FOR SYNTACTIC STRUCTURES

ADI PALM

ABSTRACT. In this article, we investigate the expressivity of a series of languages describing syntactic structures, i.e. labeled ordered trees. Such a tree language states a formalism to describe certain sets of trees. We apply them to represent syntactic principles recently used in some theories of syntax. An investigation of the expressivity establishes the relationship to appropriate types of formal grammars, i.e. some versions of the context-free grammars and the mildly context-sensitive grammars. The tree languages considered employ formalisms such as first-order logic, propositional dynamic logic, propositional modal logic, and rational trees. Each of these formalisms commonly describes structures that can be generated by a corresponding context-free grammar. As a basic result we can construct the set of trees specified by a certain principle and our approach can establish the explicit relationship between syntactic principles and the trees. Finally, we discuss some extensions of our approach to capture non-context-free properties of natural languages.

1. INTRODUCTION

During the last decades, several different formalisms in linguistics and computational linguistics have emerged, varying from context-free, or even regular, grammars to constraint-based and principle-based grammar formalisms. All are concerned with the same issue of describing and constraining syntactic structures of natural languages. They differ, however, in their manner to specify syntactic structures. The context-free grammars focus on the detailed construction of labeled trees. In detail, they employ a set of rules to describe the set of trees being constructed by these rules. On the other hand, a principle-based formalism emphasizes partial descriptions of trees by employing well-formedness conditions. Thus, a principle-based specification focuses on a particular property of a tree while stating less about the construction of a tree. In contrast, a rule-based description, e.g. a context-free grammar, provides detailed information concerning the explicit construction. The notion of principles, however,

ISSN 0947-6954/97

allows us to concentrate on particular aspects of the structure without constantly needing to keep the other properties in mind. Hence, we can utilize principles to formulate the universal properties of natural language.

This declarative notion of principles has served as the formal foundation of Chomsky's Theory of *Government and Binding* (G/B) (Chomsky 1981, 1986). The G/B-principles state certain structural relations between tree nodes having certain properties. In addition, we should mention some constraint-based grammar formalisms, e.g. *LFG* (Kaplan and Bresnan 1982), *GPSG* (Gazdar et al. 1985) and *HPSG* (Pollard and Sag 1987, 1994). They employ a similar notion of 'constraints' to specify syntactic structures. However, these constraints apply to context-free rules, cf. Shieber (1986, 1992), rather than to certain structural relations between tree nodes. Nevertheless, such constraints constitute a particular type of local principles.

Although the principle-based approaches offer a number of formal benefits they lack a detailed description of the trees specified. Since a principle emphasizes only a certain aspect of a tree, e.g. case assignment, binding, etc., it leaves open the properties of the nodes which are not regarded by this principle. Consequently, a principle-based specification of a set of trees provides only partial information on the shape of the trees. For lack of a complete description we do not exactly know the set of trees being defined by the principles. However, these sets of possible trees denote the expressivity of a principle-based approach. In essence, we formulate the expressivity of a formalism specifying sets of trees by means of formal grammars and their derivation trees, since a rule-based approach provides total descriptions of a set of trees rather than a partial description as provided by principles. In other words, the expressivity of a tree language states its strong generative power.

In this article we discuss a series of tree languages, i.e. languages to specify sets of trees. We establish a stepwise transformation from classical logic into rule-based formal grammars. We start by discussing the *Constraint Language for Attributed Trees* (CLAT($\mathfrak{L}$)) which is formally based on first-order logic. The node attributes used here are the simple labels of the finite label domain $\mathfrak{L}$. Beside the Boolean operators and the quantifiers, the language CLAT($\mathfrak{L}$) employs two binary structural relations. The *dominance* relation holds between a node and every other node below it while the *precedence* relation states that a node stands somewhere left of another one. In addition, each label $\mathfrak{l}$ of the finite label domain $\mathfrak{L}$ constitutes a unary label predicate. Those structural relations and the label predicates are sufficient to specify most of the structural relations for linguistic purposes, especially within the G/B framework.

To establish the expressivity of CLAT($\mathfrak{L}$) and, hence, of G/B-principles formulated in CLAT($\mathfrak{L}$), we deal with some intermediate formalisms. First we give up the notion of structural relations and replace them with paths. A path describes the route from one node called the source node to another one called the target node. The formal foundation of this path formalism is the *propositional dynamic logic*, cf. Harel (1984). Hence, $\mathsf{PDL}_{\mathsf{T}}(\mathcal{L})$ denotes the propositional dynamic logic for labeled ordered trees. After establishing the relationship between $\text{C}\scriptstyle\text{LAT}$$(\mathcal{L})$ and $\mathsf{PDL}_{\mathsf{T}}(\mathcal{L})$ we turn to a restricted version called the 'single-step-path' language $\mathsf{PML}_{\mathsf{T}}(\mathcal{L}, n)$ based on *propositional modal logic*. The crucial difference to $\mathsf{PDL}_{\mathsf{T}}(\mathcal{L})$ is the absence of long-distance paths, so a $\mathsf{PML}_{\mathsf{T}}(\mathcal{L}, n)$ formula only states local constraints on tree nodes. We compensate this reduction of expressivity by making use of the auxiliary label domain $\{0, 1\}^n$. In a particular representation, the local constraints expressed by $\mathsf{PML}_{\mathsf{T}}(\mathcal{L}, n)$ formulae specify types of tree nodes and possible structural relations among them. Such formulae correspond to disjunctive or regular rational trees which provide a graph-based definition of certain sets of trees. Moreover, these types of rational trees coincide with a slightly extended version of context-free grammars. Consequently, we achieve a transformation from CLAT$(\mathcal{L})$ principles into (extended) context-free grammars. In addition, we briefly discuss an extension of our approach that also deals with an infinite label domain and some certain kinds of context-sensitive grammar formalisms.

2. A FIRST-ORDER LANGUAGE FOR PRINCIPLES

The first tree language we consider is based on first-order logic (see e.g. Partee et al. 1990). This formalism serves to formulate principles such as the ones used within G/B. A principle is a logical formula constraining a tree structure with respect to some specific aspects. Therefore, a corresponding formalism dealing with G/B principles must be based on the structural properties of trees. The underlying notion of trees regards them as a particular kind of graph with certain properties. Some similar approaches to specify G/B-principles have been provided by Stabler (1992) and Rogers (1995). Stabler has utilized the horn clause fragment of first-order logic to specify principle-based logic programming. Like our formalism, Rogers' monadic second order language $\mathsf{L}^2_{\mathsf{K},\mathsf{P}}$ focuses on the description of trees rather than to establish a principle-based parsing method. In contrast to Rogers, we employ first-order logic which is properly weaker than (monadic) second-order logic. Nevertheless, first-order logic performs sufficient generative power to specify G/B principles.

Basically, we employ labeled ordered trees to represent syntactic structures. In essence, an ordered tree consists of a set of tree nodes $\mathcal{N}$ whose elements are ordered by two binary relations, the *dominance* and the *precedence* relation. The properties of these relations mainly characterize the typical shape of an ordered tree. Consider the example in Figure 1: A node, say n_i, *dominates* another

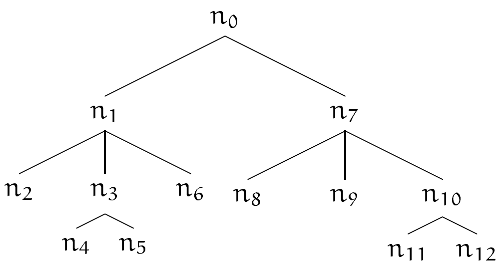


FIGURE 1. A tree

node, say n_j, if n_i stands above n_j. Every tree has a unique node, called the *root*, that dominates all other nodes and that is not dominated by any other one. If a node n_i immediately dominates the node n_j, we say n_i is the *parent* of n_j and n_j is a *child* of n_i. Two nodes n_i and n_j are *siblings* if they have the same parent. Similarly, a node n_i *precedes* a node n_j if n_i stands left of n_j. For a precedence relation between n_i and n_j it is not necessary that they are siblings, e.g. n_2 of the tree above precedes all other nodes except n_0 and n_1. Obviously, if n_i dominates n_j or vice versa then neither n_i precedes n_j nor n_j precedes n_i. If the nodes n_i and n_j are siblings and n_i immediately precedes n_j then n_i is the *right-hand sibling* of n_j and, conversely, n_j is the *left-hand sibling* of n_i. The *immediate neighbors* of a node are its children, its parent, its left-hand and right-hand sibling. For instance, the immediate neighbors of the node n_3 are the parent n_1, the left-hand sibling n_2, the right-hand sibling n_6 and the children n_4 and n_5.

In addition, a labeled ordered tree includes a labeling function α that associates every tree node with a certain label of the label domain $\mathcal{L}$. For a first-order formalism, it is sufficient only to employ the dominance and the precedence relation, since we can represent each of the other structural relations mentioned above by a corresponding first-order formula. Accordingly, we introduce CLAT($\mathcal{L}$), the *Constraint Language for Attribute Trees*. In detail, CLAT($\mathcal{L}$) employs only a single attribute of a tree node which is a label $l \in \mathcal{L}$:

Definition 1. CLAT($\mathcal{L}$) is a first-order language on a set of tree nodes $\mathcal{N}$ that includes the following predicate symbols:

$l \in \mathcal{L}$	denotes a monadic *label predicate*
$\delta \subset \mathcal{N} \times \mathcal{N}$	denotes the irreflexive *dominance relation*
$\pi \subset \mathcal{N} \times \mathcal{N}$	denotes the irreflexive *precedence relation*

A CLAT($\mathcal{L}$)-principle is a first-order formula using only these predicates. Beside the mentioned dominance and precedence relation CLAT($\mathcal{L}$) includes the node equivalence $\approx$ since we assume that the dominance and the precedence relation are *irreflexive*, i.e. a node does not dominate or precede itself. Moreover, the language CLAT($\mathcal{L}$) includes neither constant symbols for tree nodes nor function symbols. The only way to make statements about nodes is to use variables with the corresponding quantifiers. For instance, if we want to state that every node x with the label $l_1 \in \mathcal{L}$ must precede one of its siblings y with the label $l_2 \in \mathcal{L}$, we can write the following formula:

$$\forall x\,[l_1(x) \Rightarrow \exists y\,[\pi(x,y) \wedge l_2(y) \wedge \forall z\,[\delta(z,x) \Leftrightarrow \delta(z,y)]]]$$

To restrict the precedence relation to siblings, i.e. nodes with a common parent node, we use the additional condition $\forall z\,[\delta(z,x) \Leftrightarrow \delta(z,y)]$ which states that x and y must have the same nodes z dominating them. Since we often make use of this restricted version of the precedence relation, we define a special structural relation called the *sibling precedence*:

$$\pi_S(x,y) \quad := \quad \pi(x,y) \wedge \forall z\,[\delta(z,x) \Leftrightarrow \delta(z,y)]$$

The index $_S$ indicates the restriction to siblings. Note that the sibling precedence does not mean a real extension of CLAT($\mathcal{L}$). It is rather a short-cut of a particular CLAT($\mathcal{L}$) formula. In the same manner we may define some other frequently used 'auxiliary' structural relations as, for instance, the reflexive dominance relation δ^0, the immediate dominance relation δ^1 and the immediate sibling precedence relation π_S^1.

$$\begin{array}{rcl}
\delta^0(x,y) & := & \delta(x,y) \vee x \approx y \\
\delta^1(x,y) & := & \delta(x,y) \wedge \forall z\,[\delta(z,y) \Rightarrow (\delta(z,x) \vee (z \approx x))] \\
\pi_S^1(x,y) & := & \pi_S(x,y) \wedge \forall z\,[\pi_S(z,y) \Rightarrow (\pi_S(z,x) \vee (z \approx x))]
\end{array}$$

Although CLAT($\mathcal{L}$) has a well-defined syntax and semantics provided by the properties of first-order languages, we can never be sure that the models of CLAT($\mathcal{L}$) formulae are ordered trees. Therefore, we assume an axiomatization of ordered trees such that the models of these axioms exactly mean the set of labeled ordered trees. Such a first-order axiomatization has been already established by Partee et al. (1990), Backofen et al. (1995) and Palm (1997). In essence, these axioms must assert that the dominance and precedence relation are strict partial orders, i.e. irreflexive, asymmetric and transitive relations. Moreover, a tree must include a single root and every node except the root must be dominated by exactly one other node. The sibling precedence must state a linear order among the children of a node. In addition, we assume that the trees considered are only finite. Note that the finiteness cannot be expressed by means of first-order logic, but we can establish an axiom stating that every satisfiable formula corresponds to at least one finite tree, cf. Backofen et al. (1995).

Every satisfiable CLAT($\mathcal{L}$) formula that is compatible with the tree axioms specifies a non-empty set of (finite) trees. Therefore, the tree axioms correspond to the set of all ordered (finite) trees. Actually, we consider a tree as a CLAT($\mathcal{L}$) structure, i.e. a domain to interpret a CLAT($\mathcal{L}$) formula. In essence, CLAT($\mathcal{L}$) structures must provide an interpretation for the node variables, the label predicates, the dominance relation and the precedence relation. Accordingly, we define a labeled ordered tree as a CLAT($\mathcal{L}$) structure that satisfies the tree axioms. In addition, we claim that an ordered tree is finite.

Definition 2. A CLAT($\mathcal{L}$) structure $t = \langle \mathcal{N}^t, \delta^t, \pi^t, a^t \rangle$ denotes an $\mathcal{L}$-*labeled ordered tree* iff t satisfies the tree axioms and $\mathcal{N}^t$ is finite.

The set $\mathcal{N}^t$ represents the nodes of the tree t, $\delta^t \subseteq \mathcal{N}^t \times \mathcal{N}^t$ is the corresponding dominance relation and, analogously, $\pi^t \subseteq \mathcal{N}^t \times \mathcal{N}^t$ is the precedence relation. In addition, a tree includes the function $a^t : \mathcal{N}^t \to \mathcal{L}$ that associates each tree node with a label $l \in \mathcal{L}$. Note that every tree t provides a particular interpretation of the nodes, the structural relations and the label predicates occurring in a CLAT($\mathcal{L}$) formula. The superscript ^{t} shall indicate this.

By employing the language CLAT($\mathcal{L}$) we can formalize some structural notions and principles of G/B. The structural foundation of the trees described in G/B states the 'x-bar scheme', see e.g. Chomsky (1981). Basically, the x-bar scheme states that a leaf node uniquely projects its category upwards. The uniqueness asserts that the projection describes a line that does not meet any other projection line. In addition, the x-bar scheme assigns a 'level of projection' to each tree node. The level *head* denotes the underlying leaf of a projection line, the level *bar* denotes an intermediate node of the projection line and the level *maximal* denotes the unique ending point of a projection line. In general, we obtain structures of the shape exemplified in Figure 2. This tree describes the typical structure of an English main clause. For a category X, we mark the maximal projection by XP, an intermediate node by $\overline{X}$, and the head node by X_0. In contrast to the standard definition of the x-bar scheme, we propose a particular version that does not distinguish several levels of a projection. We assume only a predicate max indicating whether a node is the maximal element of projection. In essence, the x-bar scheme asserts the unique projection of category, i.e. every non-leaf node uniquely dominates the head of its projection. We employ the predicate max that indicates the end of a projection line to establish the unique projection. Actually, the non-maximal nodes mark the projection line. Hence, we achieve a unique projection line if every node has at most one non-maximal child or, alternatively, if every non-maximal node has only maximal

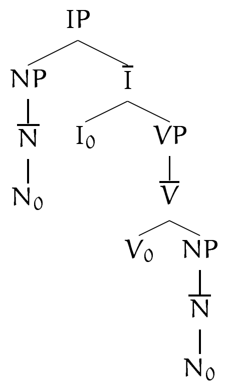


FIGURE 2. An x-bar tree

siblings. Moreover, the x-bar scheme demands that a projection line includes only elements of the same category. Finally, the root node must be maximal. According to these requirements, we employ some partial constraints specifying these properties. The monadic predicate $unique(x)$ asserts the uniqueness of the projection by stating that a non-maximal node has at most only maximal siblings. For the projection line, we employ the predicate $proj_k(x)$. It states that a maximal node x of the category cat_k dominates some leaf y such that y and the nodes z between x and y are not maximal and they have the category cat_k. The index $_k$ states that we actually require a version of $proj_k(x)$ for every category cat_k e.g. N, V, A, P, etc. Together with the predicate $unique(x)$ the predicate $proj_k$ asserts a unique line of projection. Finally, the predicate $maxroot$ states that the root must be maximal. Sometimes it is assumed that the x-bar scheme restricts the trees to binary branching. Therefore, we define the predicate $binary$ which states that at least the left-hand or the right-hand sibling is absent.

$$
\begin{aligned}
unique(x) &:= \neg max(x) \Rightarrow \neg\exists y\,[(\pi_S(y,x) \vee \pi_S(x,y)) \wedge \neg max(y)] \\
proj_k(x) &:= (max(x) \wedge cat_k(x)) \Rightarrow \\
&\quad\ \exists y[\delta(x,y) \wedge \neg\exists z\,[\delta(y,z)] \wedge cat_k(y) \wedge \neg max(y) \wedge \\
&\quad\ \forall z\,[(\delta(x,z) \wedge \delta(z,y)) \Rightarrow (cat_k(z) \wedge \neg max(z))]] \\
maxroot(x) &:= \neg\exists y\,[\delta(y,x)] \Rightarrow max(x) \\
binary(x) &:= (\neg\exists y\,[\pi_S(x,y)]) \vee (\neg\exists y\,[\pi_S(y,x)])
\end{aligned}
$$

Altogether, the x-bar scheme asserts that every node x must satisfy $unique(x)$, $proj_k(x)$, $maxroot(x)$ and $binary(x)$:

$$x\text{-}bar \;:=\; \forall x\,[unique(x) \wedge proj_k(x) \wedge maxroot(x) \wedge binary(x)]$$

Within the principles of G/B we often employ certain structural relations, the command relations and the government relation for instance. The *c-command* relation between two nodes x and y states that either x precedes y or y precedes x and every node z that dominates x must also dominate y. Similarly, the *m-command* considers pairs of nodes x and y where either x precedes y or y precedes x and every maximal node z that dominates x must also dominate y. In a simplified version, the *government* relation denotes a restricted version of the m–command between x and y where, additionally, every maximal node z dominating y also dominates x:

$$\begin{aligned}
c\text{-}comm(x,y) &:= (\pi(x,y) \vee \pi(y,x)) \wedge \forall z\,[\delta(z,x) \Rightarrow \delta(z,y)] \\
m\text{-}comm(x,y) &:= (\pi(x,y) \vee \pi(y,x)) \wedge \forall z\,[(max(z) \wedge \delta(z,x)) \Rightarrow \delta(z,y)] \\
govern(x,y) &:= m\text{-}comm(x,y) \wedge \forall z\,[(max(z) \wedge \delta(z,y)) \Rightarrow \delta(z,x)]
\end{aligned}$$

Likewise, we can establish all structural notions occurring within G/B-principles. However, CLAT($\mathcal{L}$) fails, in general, to capture the notion of *binding* since the finite label domain prevents us from using an arbitrary number of indices to represent references and co-indexation. We discuss this problem in Section 6. Nevertheless, in some cases we can infer the co-indexation from other (structural) relation, see e.g. Rogers (1995), Kracht (1995), so we need not specify it explicitly. Therefore, CLAT($\mathcal{L}$) captures most principles of the G/B-framework.

3. PROPOSITIONAL DYNAMIC LOGIC FOR TREES

The basic idea of the formalism presented here is to replace the notion of structural relations with paths. Simply put, a path describes the route from a tree node to another one within the tree. In contrast to the structural relations employed in CLAT($\mathcal{L}$), paths are directed. While we can use a structural relation in both directions, a path cannot 'look' back to the node where it has started from. Consequently, the notion of paths seems to be more restricted than the first-order formalism CLAT. However, we will be able to establish the contrast. The basic advantages of specifying principles in a path-based framework are the more compact representation and some built-in properties of trees.

Actually, a path corresponds to a sequence of tree nodes that are linked by certain structural relations. For instance, we consider the path from a node n_0 to a node n_k via the nodes $n_1, n_2, \ldots, n_{k-1}$. Moreover, we assume the structural relation r_i between the nodes n_{i-1} and n_i for $1 \leq i \leq k$. The result is a path description of the following shape:

$$n_0 \xrightarrow{r_1} n_1 \xrightarrow{r_2} n_2 \xrightarrow{r_3} \ldots \xrightarrow{r_k} n_k$$

In a more abstract view we ignore the intermediate nodes n_1 to n_{k-1} and focus only on the structural relations. Hence, we achieve a more simple description $n_0 \xrightarrow{r} n_k$ where the relation $r = r_1 \circ \ldots \circ r_k$ denotes the concatenation of the structural relations r_1 to r_k. Thus, r describes the path from n_0 to n_k. Consequently, we might regard a path corresponding to a structural relation r as a function r_{path} mapping the *source node* n_0 to the *target node* $n_k \approx r_{path}(n_0)$. However, the notion of function is too strong since it states the unique existence of the node n_k. Actually, it is quite sufficient that some node n_k exists which can be reached from n_0 by the structural relation r, i.e. $\exists n_k\, [r(n_0, n_k)]$. Therefore, we may consider a path represented by r as a kind of *restricted existential quantifier* where the structural relation r restricts the domain of quantification.

After these basic considerations, we turn to the kind of paths to employ within our approach. First of all, we assume that the relations r_1 to r_k used previously are only the dominance relation, the sibling precedence relation and their corresponding converse relations. Thus, we achieve the four path directions 'up', 'down', 'left', and 'right'. Nevertheless, the structural relations of CLAT($\mathcal{L}$) constitute unbounded, long-distance relations. In the path-based formalism considered now we employ simple *atomic* paths. In addition, we provide a method to establish more complex paths by composing simpler ones. Therefore, we introduce the atomic paths $\downarrow$, $\rightarrow$, $\uparrow$ and $\leftarrow$ that correspond to the immediate dominance relation, the immediate sibling precedence relation and their converse relations, respectively. Hence, the dominance relation, the sibling precedence relation and their complements correspond to the transitive closure of these atomic paths.

Another issue of paths we have ignored so far concerns the properties of the intermediate nodes. For instance, it should be possible to state that a path continues only if the currently considered node satisfies some condition φ. This kind of path is called a 'test'. It does not advance in any direction but the succeeding path is only entered if the node currently considered satisfies the condition specified. By employing a test, we can construct conditional paths. Moreover, it is sometimes required to select between some alternative paths. So we propose disjunctive paths, too. Altogether, our path-based formalism should include atomic paths, a test for conditional paths, the concatenation of paths, disjunctive paths and the transitive closure of paths.

The *propositional dynamic logic PDL*, cf. Harel (1984), provides the formal foundation we use for our path formalism. It agrees with the suggested properties of the path formalism. A similar approach has been proposed by Kracht

(1995). In general, *PDL* states a proper extension of *propositional modal logic PML* cf. Bull and Segerberg (1984), we describe in the next section. Formerly, *PDL* has been introduced to describe the properties of computer programs. For a detailed introduction of *PDL* we refer to Harel (1984). In our approach *PDL* serves to describe advancing paths instead of describing advancing computer programs. Hence, we speak of paths rather than of programs. We call this language $\mathrm{PDL_T}(\mathcal{L})$, where $\mathcal{L}$ denotes the same set of labels as in $\mathrm{CLAT}(\mathcal{L})$.

Definition 3. The language $\mathrm{PDL_T}(\mathcal{L})$ consists of a set of statements $\mathcal{S}$ and a set of paths $\mathcal{P}$. The following expressions are *statements*:

truth values	$\top, \bot \in \mathcal{S}$	
label statement	$l \in \mathcal{S}$	with $l \in \mathcal{L}$
path statement	$\langle p \rangle \varphi \in \mathcal{S}$	with $p \in \mathcal{P}$, $\varphi \in \mathcal{S}$
negation	$\neg\varphi \in \mathcal{S}$	with $\varphi \in \mathcal{S}$
conjunction	$\varphi \wedge \psi \in \mathcal{S}$	with $\varphi, \psi \in \mathcal{S}$

The following expressions are *paths*:

atomic paths	$\uparrow, \downarrow, \leftarrow, \rightarrow \in \mathcal{P}$	
concatenation	$p; q \in \mathcal{P}$	with $p, q \in \mathcal{P}$
disjunction	$p \cup q \in \mathcal{P}$	with $p, q \in \mathcal{P}$
transitive closure	$p^* \in \mathcal{P}$	with $p \in \mathcal{P}$
test	$\varphi? \in \mathcal{P}$	with $\varphi \in \mathcal{S}$

In contrast to the first-order formalism $\mathrm{CLAT}(\mathcal{L})$ the language $\mathrm{PDL_T}(\mathcal{L})$ neither employs variables nor predicates. Instead a $\mathrm{PDL_T}(\mathcal{L})$ expression only consists of statements and paths. Therefore, every label $l \in \mathcal{L}$ that formerly specified a label predicate in $\mathrm{CLAT}(\mathcal{L})$ now denotes a label statement l in $\mathrm{PDL_T}(\mathcal{L})$. The notion of paths used in $\mathrm{PDL_T}(\mathcal{L})$ includes the properties of paths discussed previously. The Kleene-star operator '*' generates the transitive closure while the positive closure '$^+$' corresponds to the path $p^+ = p; p^*$.

In contrast to first-order logic, *PDL* does not include quantifiers, variables or individuals. Note that the absence of individuals implies the absence of predicates and relations, too. The paths are used to replace the structural relations of $\mathrm{CLAT}(\mathcal{L})$ but also to remove the node variables and their quantifiers. Since a $\mathrm{PDL_T}(\mathcal{L})$ formula is a statement, we can interpret such statements only with respect to a certain node n of an ordered tree t.

Definition 4. We interpret every $\mathsf{PDL_T}(\mathcal{L})$ statement with respect to some node $n \in \mathcal{N}^t$ of an $\mathcal{L}$-tree $t = \langle \mathcal{N}^t, \delta^t, \pi^t, a^t \rangle$:

$$\begin{array}{lll} t, n \models \top & & \\ t, n \models l & \text{iff} & a^t(n) = l \\ t, n \models \langle p \rangle \varphi & \text{iff} & \exists n_p\, [R^t(p)(n, n_p) \wedge t, n_p \models \varphi] \\ t, n \models \neg\varphi & \text{iff} & t, n \not\models \varphi \\ t, n \models \varphi \wedge \psi & \text{iff} & (t, n \models \varphi) \wedge (t, n \models \psi) \end{array}$$

The interpretation of a program p is a binary relation $R^t(p) \subseteq \mathcal{N}^t \times \mathcal{N}^t$:

$$\begin{array}{rcl} R^t(\downarrow)(n_1, n_2) & = & R^t(\uparrow)(n_2, n_1) = {\delta^1}^t(n_1, n_2) \\ R^t(\rightarrow)(n_1, n_2) & = & R^t(\leftarrow)(n_2, n_1) = {\pi_S^1}^t(n_1, n_2) \\ R^t(p; q)(n_1, n_2) & = & \exists n \in \mathcal{N}^t\, [R^t(p)(n_1, n) \wedge R^t(q)(n, n_2)] \\ R^t(p \cup q)(n_1, n_2) & = & R^t(p)(n_1, n_2) \vee R^t(q)(n_1, n_2) \\ R^t(p^*)(n_1, n_2) & = & (R^t(p))^*(n_1, n_2) \\ R^t(\varphi?)(n_1, n_2) & = & (n_1 \approx n_2) \wedge t, n \models \varphi \end{array}$$

We use a labeled ordered tree t and a node $n \in \mathcal{N}^t$ to interpret a $\mathsf{PDL_T}(\mathcal{L})$ formula. The node n only satisfies a label statement l if n has the corresponding label l, i.e. $a^t(n) = l$. For interpreting a path statement $\langle p \rangle \varphi$ we employ the structural relation $R^t(p)$ corresponding to the path p. The path statement considered is only true if there is a target node n_p such that the relation $R^t(p)$ applies to n and n_p and, in addition, the target node n_p satisfies the target condition φ. Moreover, we interpret the Boolean operators like in first-order logic so we do not discuss them further.

The second part of the interpretation regards the transformation of paths into a corresponding binary relation. In detail, atomic paths are straightforwardly interpreted by the corresponding immediate structural relations. The concatenation and disjunction of paths agrees with the corresponding operations on binary relations. However, the Kleene-star operator and the test are more crucial. For a program p^* we employ the transitive closure of the relation $R^t(p)$. Finally, the test states a restricted version of the node equivalence which is only true if the node considered satisfies the test condition φ.

Since $\mathsf{PDL_T}(\mathcal{L})$ is unable to distinguish different tree nodes we interpret $\mathsf{PDL_T}(\mathcal{L})$ formulae with respect to a particular tree node n. Nevertheless, there is a notion of interpreting $\mathsf{PDL_T}(\mathcal{L})$ formulae that does not require to employ a node n. We consider rather all nodes of the trees. Consequently, we assume the following interpretation of $\mathsf{PDL_T}(\mathcal{L})$ formula ϕ with respect to a tree $t = \langle \mathcal{N}^t, \delta^t, \pi^t, a^t \rangle$:

$$t \models \phi \quad \text{iff} \quad \forall n \in \mathcal{N}^t\, [t, n \models \phi]$$

Obviously, the general method of interpreting $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ formulae agrees with the universally quantified principles used in G/B. We mainly employ $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ as a path-based, intermediate representation of principles on labeled ordered trees. When interpreting statements including a path, we have already seen the basic method to transform $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ formulae into the first-order formalism $\textsc{Clat}(\mathcal{L})$. For comparing expressions of both formalisms we assume the existence of a *standard translation* st_x that translates a $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ expression into $\textsc{Clat}(\mathcal{L})$. However, we will show later that this transformation does not apply in general but it works for most of the formulae considered here. The index x marks the free variable for which we interpret a $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ statement. We reach the following transformation from $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ into $\textsc{Clat}(\mathcal{L})$:

$$\mathsf{st}_x(\langle p\rangle\varphi) \equiv \exists y\left[\mathsf{R}^t(p)(x,y) \wedge \mathsf{st}_y(\varphi)\right]$$

Every path statement in $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ seems to correspond to an existential quantified $\textsc{Clat}(\mathcal{L})$ formula with a free variable x. However, there are $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ paths we cannot transform into a corresponding structural relation. For instance, consider the $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ formula $\langle(\uparrow;\uparrow)^*\rangle\varphi$. This formula states a path of an unbounded but even length. Unfortunately, we cannot establish a structural relation between nodes of an even distance in $\textsc{Clat}(\mathcal{L})$. Actually, the absence of a general transitive closure operator in first-order logic gives rise to this deficiency in expressivity. However, the transformation of a Kleene-star path p^* is straightforward if p already states a transitive-closed structural relation or its transitive closure states a structural relation in $\textsc{Clat}(\mathcal{L})$. This condition applies, for instance, to the atomic paths p_0 with $p_0 \in \{\uparrow,\downarrow,\leftarrow,\rightarrow\}$ and their transitive closures $p_0{}^*$. In general, we can handle the transitive closure of paths which are either atomic paths or, trivially, the transitive closure of atomic paths. Moreover, we can also handle the concatenation of atomic paths with a test. Consider for instance:

$$\begin{aligned}&\mathsf{st}_x\left(\langle(\psi?;\uparrow)^*\rangle\varphi\right) \equiv\\ &\qquad \mathsf{st}_x(\varphi) \vee \exists y[\delta(y,x) \wedge \mathsf{st}_y(\varphi) \wedge \forall z\,[(\delta(y,z) \wedge \delta(z,x)) \Rightarrow \mathsf{st}_z(\psi)]]\end{aligned}$$

The transitive closure of a test $\psi?$ concatenated with an atomic path describes a path with an intermediate condition $\psi?$. An intermediate condition constitutes a $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ statement ψ that must apply to all intermediate nodes of a path. Consequently, we employ the following universally quantified formula in $\textsc{Clat}(\mathcal{L})$ to ensure this:

$$\forall z\,[(\delta(y,z) \wedge \delta(z,x)) \Rightarrow \mathsf{st}_z(\psi)]$$

Similarly, we may handle the transitive closure of the paths $p_0;\psi?$ or $\psi;p_0;\psi'?$ with $p_0 \in \{\uparrow,\downarrow,\leftarrow,\rightarrow\}$. In general, we can transform the transitive closure of paths having the length 1, i.e. an atomic path which is optionally concatenated with a test. Due to the shape of trees some more complex paths correspond to an atomic path with a test. For instance, the formula $\langle(\leftarrow;\uparrow)^*\rangle\varphi$ is equal to $\langle(\langle\rightarrow\rangle\top?;\uparrow)^*\rangle\varphi$. Since siblings share a common parent node, we reach the same node by the paths $\uparrow$, $\rightarrow;\uparrow$, $\leftarrow;\uparrow$, etc. However, the corresponding $\mathrm{PDL}_T(\mathcal{L})$ statements are not equivalent since $\rightarrow;\uparrow$ states in contrast to $\uparrow$ the existence of a right-hand sibling. Therefore, only the path $\langle\rightarrow\rangle\top?;\uparrow$ agrees with $\rightarrow;\uparrow$.

After discussing the (partial) transformation from $\mathrm{PDL}_T(\mathcal{L})$ into CLAT($\mathcal{L}$), we turn to the converse direction. Here we employ an important result of Palm (1997) which establishes a normal form for CLAT($\mathcal{L}$) formulae. In essence, every CLAT($\mathcal{L}$) formula with a single free variable x can be transformed into a Boolean combination of label predicates and formulae having the following shape:

$$\exists y\,[r(x,y)\wedge\varphi_t\wedge\forall z\,[(r(x,z)\wedge r(z,y))\Rightarrow\varphi_i]]$$

where $r\in\{\delta,\delta^{-1},\pi_S,\pi_S^{-1}\}$ denotes a structural relation, φ_t denotes the target condition where at most the variable y is free, and φ_i denotes the intermediate condition where at most the variable z is free. Moreover, the conditions φ_t and φ_i consist of a Boolean combination of label predicates and of such formulae using the free variables y and z. We have already established that this kind of formulae corresponds to a $\mathrm{PDL}_T(\mathcal{L})$ expression having the following shape:

$$\langle\mathrm{path}(r);((\varphi_i')?;\mathrm{path}(r))^*\rangle\varphi_t'$$

where φ_t', and φ_i' denote the $\mathrm{PDL}_T(\mathcal{L})$ statements whose standard translation agrees with the CLAT($\mathcal{L}$) predicates φ_t, and φ_i, respectively. The function path maps a structural relation to the corresponding atomic path, i.e. $\mathrm{path}(\delta)=\downarrow$, $\mathrm{path}(\delta^{-1})=\uparrow$, $\mathrm{path}(\pi_S)=\rightarrow$ and $\mathrm{path}(\pi_S^{-1})=\leftarrow$. In addition, we obtain two special cases if the intermediate predicate φ_i states a constant value:

$$\varphi_i\equiv\top:\exists y\,[r(x,y)\wedge\varphi_t]\equiv\mathrm{st}_x\,(\langle\mathrm{path}(r);\mathrm{path}(r)^*\rangle\varphi_t')$$
$$\varphi_i\equiv\bot:\exists y\,[r(x,y)\wedge\varphi_t\wedge\neg\exists z\,[r(x,z)\wedge r(z,y)]]\equiv\mathrm{st}_x\,(\langle\mathrm{path}(r)\rangle\varphi_t')$$

If the intermediate condition holds in general we may omit it. But if it never applies, we achieve an immediate structural relation. Since we can transform every CLAT($\mathcal{L}$) formulae with a single free variable x into a Boolean combination of label constraints and formulae of this kind, we can also handle every CLAT($\mathcal{L}$) principle. On the other hand, CLAT($\mathcal{L}$) fails to represent every $\mathrm{PDL}_T(\mathcal{L})$ formula. Therefore, the first-order formalism CLAT($\mathcal{L}$) defines the proper sublanguage $\mathrm{PDL}_T^1(\mathcal{L})$ of $\mathrm{PDL}_T(\mathcal{L})$. In this sublanguage, we can only apply the Kleene-star to paths of the shape p_0, $\varphi?;p_0$ or $p_0;\varphi?$ where p_0 denotes an atomic path. Consequently, this result states that the expressivity of

$\textsc{Clat}(\mathcal{L})$ is in fact weaker than the one of $\mathsf{PDL}_T(\mathcal{L})$. However, from the perspective of G/B-principles or similar approaches the additional expressivity of $\mathsf{PDL}_T(\mathcal{L})$ is not required. Actually, such principles employ only paths of a fixed length or of an arbitrary length but never of an even or odd length or, more generally, a length that is the multiple of a certain natural number. Hence, using $\mathsf{PDL}^1_T(\mathcal{L})$ implies no restrictions that are of linguistic relevance.

We employ some linguistic examples to illustrate the relationship between $\textsc{Clat}(\mathcal{L})$ and $\mathsf{PDL}^1_T(\mathcal{L})$. In the $\textsc{Clat}(\mathcal{L})$ specification of the x-bar scheme we have made use of some auxiliary predicates. Hence, we analogously define some auxiliary $\mathsf{PDL}^1_T(\mathcal{L})$ statements. In essence, the statement $\mathtt{unique}'$ asserts the uniqueness of the projection, the statement $\mathtt{proj}'_k$ establishes the unique projection line assigned to a maximal node, the statement $\mathtt{maxroot}$ ensures a maximal root and, finally the statement $\mathtt{binary}'$ prevents more than binary branching:

$$\begin{aligned}
\mathtt{unique}' &:= \neg\mathtt{max} \Rightarrow \neg\langle \leftarrow \cup \rightarrow\rangle(\neg\mathtt{max}) \\
\mathtt{proj}'_k &:= (\mathtt{max} \wedge \mathtt{cat}_k) \Rightarrow \\
&\quad\ \langle\downarrow; (\neg\mathtt{max} \wedge \mathtt{cat}_k?; \downarrow)^*\rangle\, (\neg\langle\downarrow\rangle\top \wedge \neg\mathtt{max} \wedge \mathtt{cat}_k) \\
\mathtt{maxroot}' &:= \neg\langle\uparrow\rangle\top \Rightarrow \mathtt{max} \\
\mathtt{binary}' &:= \neg\langle\leftarrow\rangle\top \vee \neg\langle\rightarrow\rangle\top
\end{aligned}$$

When we formulate the statement $\mathtt{unique}'$, we already assume the binary branching. So we only consider the immediate siblings of a node. Thus, the $\mathsf{PDL}_T(\mathcal{L})$-version of the x-bar scheme constitutes the following formula:

$$\textit{x-bar'} \quad := \quad \mathtt{unique}' \wedge \mathtt{proj}'_k \wedge \mathtt{maxroot}' \wedge \mathtt{binary}'$$

Obviously, we can more or less straightforwardly transform the $\textsc{Clat}(\mathcal{L})$ principles specified previously into $\mathsf{PDL}_T(\mathcal{L})$. However, the general method is more complex, cf. Palm (1997), since it must deal with arbitrarily used quantifiers. The principles employed here strongly correspond to the normal form that separates the target condition and the intermediate condition for each structural relation. The basic contrast between $\textsc{Clat}(\mathcal{L})$ and $\mathsf{PDL}_T(\mathcal{L})$ means the different notion of structural relation and paths. A structural relation between two nodes only states an optional information since we do not need to specify it explicitly. For introducing another node, we must only employ a quantifier. In $\mathsf{PDL}_T(\mathcal{L})$ we can only relate nodes by specifying an explicit path from one node to another one. So $\mathsf{PDL}_T(\mathcal{L})$ provides a more explicit description of the tree structures specified. This clarity signifies an important step to establish the expressivity by means of formal grammars.

4. Propositional monadic logic for trees

After transforming structural relations into paths, we reduce the paths employed in such a way that only atomic paths occur. Consequently, we must provide an alternative formulation for each compositional path operator, i.e. the concatenation, the disjunction, the test and the Kleene star, to achieve atomic paths only. In detail, we utilize a particular version of the *propositional modal logic* (*PML*) (see e.g. Bull and Segerberg 1984) to represent these 'short-path constraints'. Actually, the propositional modal logic states a simplified version of *PDL* where only atomic programs occur in front of a statement. In general, PDL does not include operations on programs but rather every kind of program states a certain *modality* or *modal operator*. For our particular purposes, we additionally assume that a statement preceded by a path includes no further paths. So we can only formulate constraints on a node and its immediate neighbors, i.e. its parent, its children, its left-hand sibling and its right-hand sibling, but not any other node.

This particular version of *PML* is called the *propositional monadic logic for trees* ($\mathsf{PML}_T(\mathcal{L}, n)$) with the extended label domain $\mathcal{L} \times \{0, 1\}^n$. An important difference to $\mathsf{PDL}_T(\mathcal{L})$ means the additional component of auxiliary labels in $\{0, 1\}^n$. We must introduce this auxiliary label component to express the former long-distance paths by means of local constraints. The auxiliary label component $\{0, 1\}^n$ constitutes the domain of n-dimensional binary vectors, where the dimension n depends on the underlying $\mathsf{PDL}_T(\mathcal{L})$ formula. If we do not require any auxiliary labels, we set $n = 0$. Otherwise, the label of a tree node consists of two parts, i.e. a label $l \in \mathcal{L}$ of the former label domain and a binary vector $x \in \{0, 1\}^n$ of the auxiliary label domain. Each component of the binary vector serves to encode a former long-distance relation. Therefore, we normally have access to a certain component rather than to the whole binary vector. So we include particular statements in the language $\mathsf{PML}_T(\mathcal{L}, n)$ to have access to the value of a certain component. Besides this modification, the syntax of $\mathsf{PML}_T(\mathcal{L}, n)$ states a proper sublanguage of $\mathsf{PDL}_T(\mathcal{L})$ without complex paths.

Definition 5. $\mathcal{L}$ denotes a finite set of labels and $\{0, 1\}^n$ denotes the set of auxiliary labels, i.e. binary vectors with n components. The set $\mathcal{P}_0 = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ states the set of paths and the set $\mathcal{S} = \mathcal{S}(\mathcal{L}, n)$ states the set of $\mathsf{PML}_T(\mathcal{L}, n)$ *statements* which is defined as follows:

Boolean constants	$\top, \bot \in \mathcal{S}$	
label statement	$l \in \mathcal{S}$	for every label $l \in \mathcal{L}$
auxiliary statement	$\xi_i \in \mathcal{S}$	for every $1 \leq i \leq n$
path statement	$\langle p_0 \rangle \varphi \in \mathcal{S}$	with $p_0 \in \mathcal{P}_0$, $\varphi \in \mathcal{S}$ is *path-free*
Boolean combinations	$\neg\varphi, \varphi \wedge \psi \in \mathcal{S}$	with $\varphi, \psi \in \mathcal{S}$

A $\mathrm{PML_T}(\mathcal{L}, n)$ expression consists of statements and with at most a single preceding atomic path. The labels of $\mathrm{PML_T}(\mathcal{L}, n)$ constitute pairs of elements of $\mathcal{L}$ and of $\{0,1\}^n$. When using a tuple of labels, we achieve their independence, i.e. actually, we may combine each of the former labels of $\mathcal{L}$ with every binary vector of $\{0,1\}^n$. Analogously, every vector states an n-tuple of binary values where a component x_i serves to represent a certain (Boolean) path property. Hence, an auxiliary statement ξ_i is true if and only if the considered node n has any label (l, x) where the i^{th} component x_i of x is true.

Like for $\mathrm{PDL_T}(\mathcal{L})$, we only consider labeled ordered trees to interpret $\mathrm{PDL_T}(\mathcal{L})$ formulae. However, we may give up the interpretation of compositional paths. In general, every modality corresponds to a particular structural relation. In $\mathrm{PDL_T}(\mathcal{L})$ we employ the immediate structural relations, i.e. the parent for $\uparrow$, some child for $\downarrow$, the left-hand sibling for $\leftarrow$ and the right-hand sibling for $\rightarrow$. The interpretation of the remaining parts is straightforward.

Definition 6. We interpret every statement of $\mathrm{PML_T}(\mathcal{L}, n)$ with respect to some node $n_t \in \mathcal{N}^t$ of an $(\mathcal{L}^t \times \{0,1\}^n)$-labeled ordered tree $t = \langle \mathcal{N}^t, \delta^t, \pi^t, a^t \rangle$:

$$\begin{array}{lll}
t, n_t \models \top & & \\
t, n_t \models l & \text{iff} & a^t(n_t) = (l, x) \text{ for some } x \in \{0,1\}^n \\
t, n_t \models \xi_i & \text{iff} & a^t(n_t) = (l, x) \wedge x_i = 1 \text{ for some } l \in \mathcal{L} \\
t, n_t \models \langle p_0 \rangle \varphi & \text{iff} & \exists n_p \left[(R^t(p_0))(n_t, n_p) \wedge t, n_p \models \varphi \right] \\
t, n_t \models \neg \varphi & \text{iff} & t, n_t \not\models \varphi \\
t, n_t \models \varphi \wedge \psi & \text{iff} & (t, n_t \models \varphi) \wedge (t, n_t \models \psi)
\end{array}$$

The interpretation of a modality p is a binary relation $R^t(p_0) \subseteq \mathcal{N}^t \times \mathcal{N}^t$:

$$\begin{array}{lllll}
R^t(\downarrow)(n_1, n_2) & = & R^t(\uparrow)(n_2, n_1) & = & {\delta^1}^t(n_1, n_2) \\
R^t(\rightarrow)(n_1, n_2) & = & R^t(\leftarrow)(n_2, n_1) & = & {\pi_S^1}^t(n_1, n_2)
\end{array}$$

The only new part of this interpretation compared with $\mathrm{PDL_T}(\mathcal{L})$ concerns the handling of the label statements. We only consider the projections of the particular components of the label. A label statement l only considers the $\mathcal{L}$-component while ξ_i refers to the i^{th} component of the auxiliary part of the label.

The formalism $\mathrm{PML_T}(\mathcal{L}, n)$ only means a simple method to specify labeled ordered trees in the framework of propositional modal logic. For a more complex approach we refer to Blackburn and Meyer-Viol (1994). They employ some additional modal operators to capture long distance dependencies. Note that our approach mainly serves as an intermediate representation between $\mathrm{PDL_T}(\mathcal{L})$ and rule-based grammar formalisms. So we have not explicitly included long distance paths. Instead we express unbounded structural relations by means of the auxiliary label components.

Now we deal with a method to transform $\mathsf{PDL}_T(\mathcal{L})$ formulae into the corresponding $\mathsf{PML}_T(\mathcal{L}, n)$ formulae by employing the auxiliary label component. The absence of compositional paths in $\mathsf{PML}_T(\mathcal{L}, n)$ gives rise to a transformation that decomposes the paths and that establishes the auxiliary labels and the corresponding constraints. We organize this transformation by two steps. First we replace each compositional path with a simpler one that includes less path operators than the previous one. Moreover, the resulting simplified formula may involve an auxiliary label statement ξ_i. In the second step, we ensure a correct replacement by adding a new constraint that associates the replaced path operation with the corresponding auxiliary statement. The whole process encodes the former compositional paths by employing auxiliary statements. Actually, this method works already for the disjunction and the test:

$$\begin{aligned}\langle p \cup q \rangle \varphi &\equiv \langle p \rangle \varphi \vee \langle q \rangle \varphi \\ \langle \psi? \rangle \varphi &\equiv \psi \wedge \varphi\end{aligned}$$

By making use of these equations, we can simplify every path including a test or a disjunction. So the resulting formulae consist of less complex paths. However, this method does not apply to the concatenation and, consequently, it also fails for the Kleene-star. In general, we can simplify a concatenation by the following equation:

$$\langle p; q \rangle \varphi \equiv \langle p \rangle \langle q \rangle \varphi$$

Although we achieve two simpler paths, we have actually moved the second path inside the path statement. Hence, a previously path-free path statement becomes non-path-free. Nevertheless, we can employ this equation to construct a transformation that does not increase the number of paths inside the statement following the path. In essence, we replace the statement $\langle q \rangle \varphi$ following the first path $\langle p \rangle$ with an auxiliary statement ξ_i:

$$\langle p; q \rangle \varphi \equiv \langle p \rangle \xi_i$$

To ensure the correctness of this transformation we utilize the additional constraint $\xi_i \Leftrightarrow \langle q \rangle \varphi$. By making use of this simple transformation, we can decompose every concatenation of paths. The auxiliary statement ξ_i represents the former path property that there is a path q such that the target node satisfies φ. In other words, we have encoded the formula $\langle q \rangle \varphi$ by using the auxiliary statement ξ_i. Similarly, we can handle the Kleene-star. Consider the equation below:

$$\langle p^* \rangle \equiv \varphi \vee \langle p \rangle \langle p^* \rangle \varphi$$

At first, we use this equation for defining an adequate additional constraint for the auxiliary statement for Kleene-star expressions:

$$\xi_i \Leftrightarrow \varphi \vee \langle p \rangle \xi_i$$

After applying this constraint to a Kleene-star path, we achieve the equivalence of $\langle p^*\rangle\varphi$ and ξ_i, i.e. $\langle p^{\rangle}\varphi \equiv \xi_i$. By combing these transformations we obtain a general method to decompose non-atomic paths. Actually, the concatenation is the only basic operation whose transformation requires an auxiliary statement. Since the Kleene-star operator constitutes an arbitrary number of repeated concatenations, we consequently need an auxiliary statement for the Kleene-star, too.

PROPOSITION 7 (Path Decomposition).

$$\begin{array}{rcll} \langle p;q\rangle\varphi & \equiv & \langle p\rangle\xi_i & \text{where} \quad \xi_i \Leftrightarrow \langle q\rangle\varphi \\ \langle p\cup q\rangle\varphi & \equiv & \langle p\rangle\varphi \vee \langle q\rangle\varphi & \\ \langle\psi?\rangle\varphi & \equiv & \psi\wedge\varphi & \\ \langle p^*\rangle\varphi & \equiv & \xi_i & \text{where} \quad \xi_i \Leftrightarrow \varphi\vee\langle p\rangle\xi_i \end{array}$$

Note that each auxiliary statement ξ_i uniquely corresponds to a certain decomposition. Hence, the dimension of the auxiliary labels depends on the number of necessary decomposition operations for concatenations and of Kleene-star operators. After repeatedly applying these transformations to an arbitrary $\mathsf{PDL}_T(\mathcal{L})$ formula, we achieve an equivalent $\mathsf{PML}_T(\mathcal{L},n)$ expression that includes some auxiliary statements and the corresponding additional constraints. A special case means the path expressions of $\mathsf{PDL}_T^1(\mathcal{L})$ that arise after transforming $\textsc{Clat}(\mathcal{L})$ principles. Even a single auxiliary statement is sufficient to decompose such a path:

$$\langle p;(\psi?;p)^*\rangle\varphi \quad \equiv \quad \langle p\rangle\xi_i \quad \text{where} \quad \xi_i \Leftrightarrow \varphi\vee(\psi\wedge\langle p\rangle\xi_i)$$

A valid auxiliary statement ξ_i expresses an existing node in 'p-direction' that satisfies φ and all intermediate nodes satisfy ψ. Hence, ξ_i must only be true if the node considered satisfies the target condition φ or it satisfies the intermediate condition ψ and the next node in p-direction satisfies ξ_i, too. In the first case, ξ_i indicates the target nodes; otherwise, it denotes an intermediate path node. If a node fails ξ_i, it is neither an intermediate path node nor the target node.

We illustrate this particular property by applying the transformation to the x-bar scheme. The crucial part states the partial constraint $\mathtt{proj}'_k$ since it includes a Kleene-star. Previously, we have proposed the following $\mathsf{PDL}_T(\mathcal{L})$ based formulation:

$$\begin{array}{l} \mathtt{proj}'_k := (\mathrm{max}\wedge\mathrm{cat}_k) \Rightarrow \\ \qquad\qquad \langle\downarrow;((\neg\mathrm{max}\wedge\mathrm{cat}_k)?;\downarrow)^*\rangle(\neg\langle\downarrow\rangle\top\wedge\neg\mathrm{max}\wedge\mathrm{cat}_k) \end{array}$$

From the transformation above we obtain the following $\mathsf{PML}_T(\mathcal{L}, n)$ version of $\mathtt{proj}'_k$ and the auxiliary constraint $\mathtt{aux}'_k$:

$$\begin{aligned} \mathtt{proj}'_k &:= (\mathtt{max} \wedge \mathtt{cat}_k) \Rightarrow \langle\downarrow\rangle\xi_k \\ \mathtt{aux}'_k &:= \xi_k \Leftrightarrow (\neg\langle\downarrow\rangle\top \wedge \neg\mathtt{max} \wedge \mathtt{cat}_k) \vee (\neg\mathtt{max} \wedge \mathtt{cat}_k \wedge \langle\downarrow\rangle\xi_k) \end{aligned}$$

Since we require for every category $\mathtt{cat}_k$ an auxiliary statement ξ_k, we associate them with the same index k. The constraint $\mathtt{proj}'_k$ asserts that every maximal node must have a unique projection line to its corresponding head. After this transformation, it is sufficient that every maximal node with the category $\mathtt{cat}_k$ has a child satisfying ξ_k. Moreover, a node only satisfies ξ_k if it is the non-maximal projection child and it is either a leaf or its projection child satisfies ξ_k. In the first case, we normally say that the node considered is the 'head' of the projection otherwise it is a 'bar'-node. Consequently, it should be possible to replace the auxiliary statement ξ_k with the statement $\neg\mathtt{max} \wedge \mathtt{cat}_k$. However, we must ensure that every node satisfying $\neg\mathtt{max} \wedge \mathtt{cat}_k$ either is a leaf or it has a child that satisfies ξ_k, too. Actually, this transformation corresponds to the following splitting of the auxiliary constrain $\mathtt{aux}'_k$ leading to the constraints $\mathtt{aux}'_{k,1}$ and $\mathtt{aux}'_{k,2}$:

$$\begin{aligned} \mathtt{aux}'_{k,1} &:= \xi_k \Leftrightarrow (\neg\mathtt{max} \wedge \mathtt{cat}_k) \\ \mathtt{aux}'_{k,2} &:= (\neg\mathtt{max} \wedge \mathtt{cat}_k) \Rightarrow (\neg\langle\downarrow\rangle\top \vee \langle\downarrow\rangle\xi_k) \end{aligned}$$

The first constraint states a simplified definition of ξ_k while the second one asserts the projection of the category $\mathtt{cat}_k$. By applying the definition of $\mathtt{aux}'_{k,1}$ to all occurrences of ξ_k, we achieve another version of $\mathtt{proj}_k$ and of the corresponding auxiliary statement $\mathtt{aux}'_k$:

$$\begin{aligned} \mathtt{proj}'_k &:= (\mathtt{max} \wedge \mathtt{cat}_k) \Rightarrow \langle\downarrow\rangle(\neg\mathtt{max} \wedge \mathtt{cat}_k) \\ \mathtt{aux}'_k &:= (\neg\mathtt{max} \wedge \mathtt{cat}_k) \Rightarrow (\neg\langle\downarrow\rangle\top \vee \langle\downarrow\rangle(\neg\mathtt{max} \wedge \mathtt{cat}_k)) \end{aligned}$$

Thus, the long-distance occurring in the former formulation of $\mathtt{proj}'_k$ has an alternative formulation employing only atomic paths without any auxiliary statement. In detail, we have utilized the statement $\neg\mathtt{max} \wedge \mathtt{cat}_k$ to encode the long-distance relation. Note that, in general, we cannot replace every auxiliary statement with an alternative expression. However, the possibility of replacing the auxiliary statement in the x-bar scheme indicates that it actually states a local constraint.

So far we have discussed the transformation of $\mathsf{PDL}_T(\mathcal{L})$ formulae into $\mathsf{PML}_T(\mathcal{L}, n)$. Although we must employ additional auxiliary statements ξ_i, we can state that both, a $\mathsf{PDL}_T(\mathcal{L})$ formula and the resulting $\mathsf{PML}_T(\mathcal{L}, n)$ formula specify the same set of trees up to projection on the label domain $\mathcal{L}$. The number of necessary auxiliary statements depends on the complexity of the path occurring in the underlying $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ formula. In general, we assume that the dimension n of the auxiliary label domain $\{0,1\}^n$ is big enough to capture the $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ formulae considered. Unfortunately, Palm (1997) showed some special cases of paths, that cannot be directly transformed to $\mathsf{PML}_\mathsf{T}(\mathcal{L},n)$. However, we ignore these paths since they are not relevant for the paths discussed here.

In contrast to $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$, a $\mathsf{PML}_\mathsf{T}(\mathcal{L},n)$ formula only states local structural constraints since a $\mathsf{PML}_\mathsf{T}(\mathcal{L},n)$ formula can only restrict the label of a node and the labels of its immediate structural neighbors. In addition, it can state the absence of an immediate neighbor. This locality gives rise to a particular normal form of $\mathsf{PML}_\mathsf{T}(\mathcal{L},n)$ formulae where we can directly access to this information. Actually, such a formula describes a certain type of tree node. It denotes a *(node) type constraint* and the corresponding normal form means the *type normal form*. According to the interpretation of $\mathsf{PML}_\mathsf{T}(\mathcal{L},n)$ (and of $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$) a tree only satisfies a $\mathsf{PML}_\mathsf{T}(\mathcal{L},n)$ formula if the formula considered applies to every tree node. So a $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ formula in disjunctive normal form states a set of node constraints such that a tree satisfies this formula if every node satisfies at least one formula of this set. Consequently, every element of this set defines a certain type of tree node. Thus, the type normal form is based on the disjunctive normal form. We assume for every $\mathsf{PDL}_\mathsf{T}(\mathcal{L})$ formula and equivalent formula $\Phi_1 \vee \ldots \vee \Phi_k$ where every Φ_i consists of a conjunction of positive and negative label statements and paths statements.

Besides the disjunctive representation, the type normal form demands a structuring of the different path statements. In detail, each kind of atomic path should only occur one time in a type constraint Φ_i. We generally interpret modal operators by restricting an existential quantification by an immediate structural relation. Thus, a modal statement behaves like an existentially quantified formula. While we can combine a disjunction of existentially quantified formulae to a single formula with a single existential quantifier, this does not apply to a conjunction of existentially quantified formulae. However, we can employ some properties of ordered trees, where in three cases, i.e. the parent, the left- and right-hand sibling, the immediate neighbor is unique. Therefore, we may replace an existentially quantified formula with a combination of an existential quantifier and a universal quantifier according to the following transformation:

$$\exists y[\gamma(x,y) \wedge \varphi(y)] \quad \equiv \quad \exists y[\gamma(x,y)] \wedge \forall y[\gamma(x,y) \Rightarrow \varphi(y)]$$

where $\gamma(x,y)$ denotes the structural relation which corresponds to the underlying modal operator. Note that this equivalence only obtains if y is unique, i.e. there is a single y that satisfies $\gamma(x,y)$. In contrast to the existentially quantified formulae, we may combine a conjunction of universally quantified formulae to a formula with a single quantifier. Thus, we can also combine conjunctions of existentially quantified formulae if there is only a single y satisfying $\gamma(x, y)$.

$$\exists y[\gamma(x,y) \wedge \varphi_1(y)] \wedge \ldots \wedge \exists y[\gamma(x,y) \wedge \varphi_n(y)] \equiv \exists y[\gamma(x,y) \wedge \varphi_1(y) \wedge \ldots \wedge \varphi_n(y)]$$

Moreover, we can use this property to replace negative modalities. First we split each negative modal statement by the following transformation:

$$\neg\langle p_0\rangle\varphi \quad \equiv \quad (\neg\langle p_0\rangle\top) \vee (\langle p_0\rangle\neg\varphi)\text{, where } p_0 \in \{\uparrow, \leftarrow, \rightarrow\}$$

However, a negative path statement $\neg\langle p_0\rangle\top$ still remains. This formula states the absence of a certain immediate neighbor. To achieve a positive formulation we introduce the 'empty node statement' λ which applies only to absent or non-existing nodes, i.e. $\langle p_0\rangle\lambda \equiv \neg\langle p_0\rangle\top$. Hence, we must only make use of λ in connection with a modal operator. Otherwise, it is undefined. By employing the empty node statement λ we can transform every negative modal statement where the modal operator must be different from $\downarrow$:

$$\neg\langle p_0\rangle\varphi \quad \equiv \quad \langle p_0\rangle\lambda \vee \langle p_0\rangle(\neg\varphi)\text{, where } p_0 \in \{\uparrow, \leftarrow, \rightarrow\}$$

So far we can only handle modal statements that include the atomic paths $\uparrow$, $\leftarrow$ and $\rightarrow$. The method used does not apply to the $\downarrow$-operator since the number of children is not restricted. Nevertheless, we can capture the negation of the modal operator $\downarrow$, too. We simply introduce an additional modal operator $\Downarrow$ that represents the properties of all children of some node:

$$\langle\Downarrow\rangle\varphi \quad := \quad \neg\langle\downarrow\rangle(\neg\varphi)$$

As outlined earlier the path statement $\langle\downarrow\rangle\varphi$ means that at least one child satisfies the condition φ while $\langle\Downarrow\rangle\varphi$ constitutes that *all* children must satisfy φ. The following lemma summarize the mentioned transformations on modal statements:

LEMMA 8 (Properties of Tree Modalities).

$$\begin{aligned}
\neg\langle p_0\rangle\varphi &\equiv \neg\langle p_0\rangle\top \vee \langle p_0\rangle(\neg\varphi)\text{, where } p_0 \in \{\uparrow, \leftarrow, \rightarrow\} \\
\langle p_0\rangle\varphi \wedge \langle p_0\rangle\psi &\equiv \langle p_0\rangle(\varphi \wedge \psi)\text{, where } p_0 \in \{\uparrow, \leftarrow, \rightarrow, \Downarrow\} \\
\langle p_0\rangle\varphi \vee \langle p_0\rangle\psi &\equiv \langle p_0\rangle(\varphi \vee \psi)\text{, where } p_0 \in \{\uparrow, \leftarrow, \rightarrow, \downarrow\} \\
\neg\langle\downarrow\rangle\varphi &\equiv \langle\Downarrow\rangle(\neg\varphi) \\
\neg\langle\Downarrow\rangle\varphi &\equiv \langle\downarrow\rangle(\neg\varphi)
\end{aligned}$$

After repeatedly applying these transformations to a $\mathrm{PML}_T(\mathcal{L}, n)$ formula, we achieve type statements Φ which have the following shape:

$$\Phi_{node} \wedge \langle\uparrow\rangle\Phi_{par} \wedge \langle\leftarrow\rangle\Phi_{left} \wedge \langle\rightarrow\rangle\Phi_{right} \wedge \langle\Downarrow\rangle\Phi_{all} \wedge \langle\downarrow\rangle\Phi_{child_1} \wedge \ldots \wedge \langle\downarrow\rangle\Phi_{child_n}$$

Thus we have separated each type constraint Φ into 'partial constraints', i.e. the *node constraint* $\Phi_{\mathtt{node}}$ only includes restrictions on the node label; the *parent constraint* $\Phi_{\mathtt{par}}$, *left constraint* $\Phi_{\mathtt{left}}$ and the *right constraint* $\Phi_{\mathtt{right}}$ include restrictions on the labels of the corresponding immediate neighbor. For representing an absent immediate neighbor we employ the empty node statement λ. The partial constraint $\Phi_{\mathtt{all}}$ states the restrictions on the labels of all children, while the constraints $\Phi_{\mathtt{child}_1}, \ldots, \Phi_{\mathtt{child}_n}$ demand the existence of at least one child for every constraint. It is possible that one node meets more than one of those existential constraints. A $\mathsf{PML}_\mathsf{T}(\mathcal{L}, n)$ formula is in *type normal form* if it consists of a disjunction such type statements. To illustrate the transformation into type normal form, we continue the example of the x-bar scheme. After transforming it to $\mathsf{PML}_\mathsf{T}(\mathcal{L}, n)$, we have achieved the following version:

$$
\begin{aligned}
\textit{x-bar} \;\equiv\;\; & (\neg max \Rightarrow \neg\langle \leftarrow \cup \rightarrow\rangle(\neg max)) \wedge \\
& ((max \wedge cat_k) \Rightarrow \langle\downarrow\rangle(\neg max \wedge cat_k)) \wedge \\
& ((\neg max \wedge cat_k) \Rightarrow (\neg\langle\downarrow\rangle\top \vee \langle\downarrow\rangle(\neg max \wedge cat_k))) \wedge \\
& (\neg\langle\uparrow\rangle\top \Rightarrow max) \wedge \\
& (\neg\langle\leftarrow\rangle\top \vee \neg\langle\rightarrow\rangle\top)
\end{aligned}
$$

After transforming this formula into disjunctive normal form, we already achieve a disjunction of node types, i.e. $\Phi_1 \vee \ldots \vee \Phi_{13}$, which are defined as follows:

$$
\begin{aligned}
\Phi_1 &\equiv max \wedge cat_k \wedge \langle\downarrow\rangle(\neg max \wedge cat_k) \wedge \langle\rightarrow\rangle\lambda \\
\Phi_2 &\equiv max \wedge cat_k \wedge \langle\downarrow\rangle(\neg max \wedge cat_k) \wedge \langle\leftarrow\rangle\lambda \\
\Phi_3 &\equiv \neg max \wedge cat_k \wedge \langle\downarrow\rangle(\neg max \wedge cat_k) \wedge \langle\uparrow\rangle\top \wedge \langle\rightarrow\rangle\lambda \wedge \langle\leftarrow\rangle max \\
\Phi_4 &\equiv \neg max \wedge cat_k \wedge \langle\downarrow\rangle(\neg max \wedge cat_k) \wedge \langle\uparrow\rangle\top \wedge \langle\rightarrow\rangle\lambda \wedge \langle\leftarrow\rangle\lambda \\
\Phi_5 &\equiv \neg max \wedge cat_k \wedge \langle\downarrow\rangle(\neg max \wedge cat_k) \wedge \langle\uparrow\rangle\top \wedge \langle\rightarrow\rangle max \wedge \langle\leftarrow\rangle\lambda \\
\Phi_6 &\equiv \neg max \wedge cat_k \wedge \langle\downarrow\rangle\lambda \wedge \langle\uparrow\rangle\top \wedge \langle\rightarrow\rangle\lambda \wedge \langle\leftarrow\rangle max \\
\Phi_7 &\equiv \neg max \wedge cat_k \wedge \langle\downarrow\rangle\lambda \wedge \langle\uparrow\rangle\top \wedge \langle\rightarrow\rangle\lambda \wedge \langle\leftarrow\rangle\lambda \\
\Phi_8 &\equiv \neg max \wedge cat_k \wedge \langle\downarrow\rangle\lambda \wedge \langle\uparrow\rangle\top \wedge \wedge\langle\rightarrow\rangle max \wedge \langle\leftarrow\rangle\lambda \\
\Phi_9 &\equiv max \wedge \neg cat_k \wedge \langle\rightarrow\rangle\lambda \\
\Phi_{10} &\equiv max \wedge \neg cat_k \wedge \langle\leftarrow\rangle\lambda \\
\Phi_{11} &\equiv \neg max \wedge \neg cat_k \wedge \langle\uparrow\rangle\top \wedge \langle\rightarrow\rangle\lambda \wedge \langle\leftarrow\rangle max \\
\Phi_{12} &\equiv \neg max \wedge \neg cat_k \wedge \langle\uparrow\rangle\top \wedge \langle\rightarrow\rangle\lambda \wedge \langle\leftarrow\rangle\lambda \\
\Phi_{13} &\equiv \neg max \wedge \neg cat_k \wedge \langle\uparrow\rangle\top \wedge \wedge\langle\rightarrow\rangle max \wedge \langle\leftarrow\rangle\lambda
\end{aligned}
$$

The node types Φ_1 and Φ_2 specify the properties of maximal nodes of the category $\mathtt{cat}_k$ while the types Φ_3, Φ_4, Φ_5, Φ_6, Φ_7, and Φ_8 specify the non-maximal nodes of the category $\mathtt{cat}_k$. The remaining types concern the nodes of a category different from $\mathtt{cat}_k$. Thus, we would get $8 \times m$ node types for m different categories. In detail, a maximal node has at most one sibling and it has a non-maximal child of the same category. A non-maximal node has at most a maximal sibling and it is either a leaf or it has a non-maximal child of the same category. Hence, Φ_3, Φ_4, and Φ_5 denote the 'bar'-nodes while Φ_6, Φ_7 and Φ_8 denote the 'head'-nodes.

Through introducing $\mathsf{PML}_T(\mathcal{L}, n)$ we have replaced the long distance paths of $\mathsf{PDL}_T(\mathcal{L})$ by 'one-step' paths. Consequently, $\mathsf{PML}_T(\mathcal{L}, n)$ employs only local constraints. We have preserved the expressivity by making use of the auxiliary label domain $\{0, 1\}^n$ and the corresponding auxiliary statements ξ_i. Therefore, a $\mathsf{PDL}_T(\mathcal{L})$ formula and the corresponding $\mathsf{PML}_T(\mathcal{L}, n)$ formula describe the same set of trees up to the projection on the underlying finite label domain $\mathcal{L}$. Moreover, we have provided the type normal form of $\mathsf{PML}_T(\mathcal{L}, n)$ formulae that expresses the different type of nodes occurring in a tree satisfying the formula. We continue our approach by establishing the graph of possible structural relations between the node types. Based on this graph, we can compute a corresponding rule-based grammar generating the described set of trees.

5. Rational trees and context-free grammars

In the preceding sections we have discussed several approaches for declarative specifications of labeled ordered trees. We have defined a set of trees in a declarative manner by employing constraints or principles, i.e. particular logical formulae. However, we have said (more or less) nothing concerning the way to construct the set of trees described. One may consider this as a model theoretic perspective since the described set of trees constitutes the models of a given formula. In contrast, a rule-based formalism, e.g. context-free grammars, emphasizes the manner of constructing a tree. To link these different kinds of regarding a set of trees, we employ a graph-based representation called the rational trees. In detail, we utilize two particular versions of rational trees, namely, the disjunctive and the regular rational trees. A disjunctive (regular) rational trees unifies the properties of $\mathsf{PML}_T(\mathcal{L}, n)$ and of (extended) context-free grammars. On the one hand, we can establish that a disjunctive (or regular) rational tree describes a graph where the nodes and the edges correspond to the node types, i.e. $\mathsf{PML}_T(\mathcal{L}, n)$ formulae in type normal form, and to the structural relations between them, respectively. But on the other hand, each node of the graph and its children denotes a rule of an (extended) context-free grammar. Consequently, rational trees mean a suitable intermediate representation to establish the correspondence between $\mathsf{PDL}_{\mathsf{T}}(\mathcal{L})$ formulae and (extended) context-free grammars. In this paper we only deal with disjunctive rational trees and ordinary context-free grammars. However, this is only possible if the formulae considered constitute a bound degree of branching. Therefore, a more general approach dealing with arbitrary branching trees must employ regular rational trees and extended context-free grammars, cf. Palm (1997) for a detailed investigation.

In general, a rational tree states a tree that we can describe in a finite manner. The crucial condition indicating a rational tree is a finite set of subtrees. Hence, a rational tree consists of a finite number of different subtrees, where every subtree is a rational tree, too. However, this condition includes an infinite number of occurrences of the same subtree. Trivially, every finite tree satisfies this condition of rational trees. Nevertheless, a rational tree may denote an infinite tree, too. Consider the examples of infinite trees in Figure 3. Obviously,

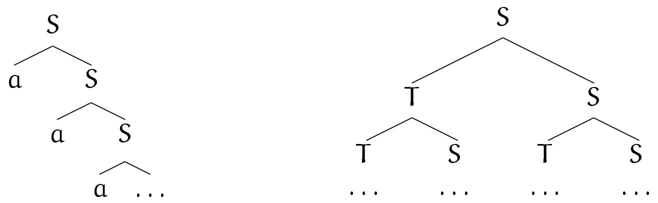


FIGURE 3. Infinite trees

both examples represent infinite but rational trees. The left-hand tree consists of two different kinds of subtrees. One of them includes only the leave a while the other subtree states the whole tree itself, i.e. the subtree with the root symbol S. In the right-hand tree we can also observe two different subtrees. However, in this example either of the possible subtrees is infinite. Actually, we can construct these infinite rational trees by the infinite repetition of the same substructure. In an alternative representation of rational trees, we employ cyclic edges to express these repetitions (cf. Figure 4). The cycles indicate the 'rational' character

FIGURE 4. Trees with cyclic edges

of these infinite trees. Similar to rational numbers, a rational tree is either finite or it includes an infinite repetition of some finite structures. Nevertheless, a rational tree represents a single tree but not a set of different trees. However, one might consider the cyclic links as a kind of Kleene-star operator. Normally, we employ the Kleene-star operator to represent an infinite set of finite strings.

We achieve a similar notion for rational trees by introducing an 'exit' to the cyclic edges. This exit means a particular alternative for going downward in the tree. Hence, we establish the notion of a *disjunctive rational tree*. In general, we assume two different types of nodes, the *disjunction nodes* and the 'ordinary' *branching nodes*. An ordinary rational tree considered so far only includes branching nodes, while a disjunctive rational tree includes both kinds of nodes. A disjunctive node only occurs immediately below a branching node and a branching node only occurs immediately below a disjunction node. Consequently, a disjunctive rational tree consists of alternating layers of branching and disjunction nodes. In contrast to the structural branches of a branching node, the branches of a disjunction node mark different alternative sequences of children of a node.

Definition 9. $R = \langle \mathcal{L}, N, N_d, N_S, d, r, a \rangle$ is a (labeled) disjunctive rational tree which is defined as follows:

- $\mathcal{L}$ denotes the non-empty, finite set of labels.
- N denotes the non-empty, finite set of (branching) nodes.
- N_d denotes the finite set of disjunction nodes.
- $N_S \subseteq N$ denotes the non-empty set of root nodes.
- $d \subset (N \times N_d) \cup (N_d \times N)$ denotes the downward relation.
- $r \subset (N \times N)$ denotes the rightward relation.
- $a : N \to \mathcal{L}$ denotes the label assignment.

where the following conditions must hold:

(i) $\forall n \in N : n \notin N_S \Rightarrow \exists n_d \in N_d : d(n_d, n)$
(ii) $\forall n_d \in N_d : \exists n_1, n_2 \in N : d(n_1, n_d) \wedge d(n_d, n_2)$
(iii) $\forall n_1, n_2 \in N : r(n_1, n_2) \Rightarrow \exists n_d \in N_d : d(n_d, n_1) \wedge d(n_d, n_2)$
(iv) $\forall n_1, n_2 \in N, n_d \in N_d :$
$(d(n_d, n_1) \wedge d(n_d, n_2)) \Rightarrow (r^*(n_1, n_2) \vee r^*(n_2, n_1))$

The conditions (i) and (ii) assert that the downward relation associates a branching node with its alternative disjunction nodes and, further, a disjunction node with the sequence of its branching nodes. So the downward relation resembles the immediate dominance relation but it is not unique, however. The condition (iii) ensures that siblings are properly dominated by some node and, in addition, the condition (iv) states a linear order by the transitive closure of the rightward relation. Actually, the conditions (i) to (vi) assert the structural properties of the disjunctive rational trees. The labels are only associated with the branching nodes. We indicate disjunction nodes by a small circle '$\circ$'. In addition, a root node receives an edge with an open end. Note that a disjunctive rational tree may include more than one root.

We can use disjunctive rational trees to represent an infinite set of labeled trees sharing some common properties. For instance, we consider the set of solely right-branching binary trees in Figure 5.

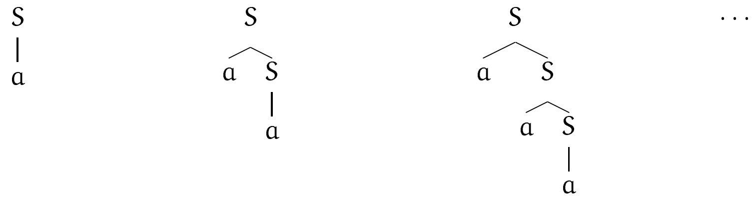


FIGURE 5. An infinite set of trees

Obviously, these trees resemble one of the previous sample rational trees. In essence, we must only modify the previous rational tree by adding an alternative 'exit' to the cyclic edge. Hence, the downward relation from the node S leads either once more to S or it leads to a single leaf a (Figure 6). If we 'walk' n

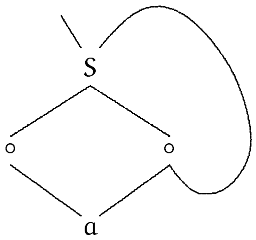


FIGURE 6. A disjunctive rational tree representing the set from Figure 5

times through the cycle, we obtain a tree describing the string a^{n+1}. So this disjunctive rational tree describes the language $\{a^{n+1}|n \geq 0\}$. Since a solely right-branching binary tree asserts a unique relation for its frontier string, i.e. the string of labels corresponding to the leaves, the disjunctive rational tree describes the intended set of labeled trees. However, we must establish the general relationship between a disjunctive rational tree and the set of trees it describes.

The basic idea is to 'unfold' the loops through the cyclic edges. Accordingly, an *unfolding* denotes a labeled tree that means an instance of a disjunctive rational tree. Therefore, we require the relationship between a disjunctive rational tree and its unfoldings. Obviously, every node must be associated with a branching node of the disjunctive rational tree. In addition, a node of an unfolding must share the structural properties with the corresponding branching node.

Definition 10. An $\mathcal{L}$-labeled ordered tree $t = \langle N^t, \delta^t, \pi^t, a^t \rangle$ is an *unfolding* of the disjunctive rational tree $R = \langle \mathcal{L}, N, N_d, N_S, d, r, a \rangle$ if there is a homomorphism $h : N^t \to N$.

We assume for the homomorphism h that it preserves the structural properties of the unfolding. In detail, the immediate dominance relation and the immediate sibling relation must agree with the downward relation and the rightward relation, respectively. Similarly, the homomorphism h must maintain the properties of the root, the leaves, the leftmost and the rightmost children. In addition, we assume the same label assignment.

Obviously, we can conclude from the definition above that the unfoldings structurally agree with the specifications of the disjunctive rational tree. Therefore, a disjunctive rational tree states an appropriate representation of sets of labeled trees. The crucial property is the correspondence of disjunctive rational trees with the $PML_T(\mathcal{L}, n)$ formulae in a certain representation. For instance, the previous sample tree corresponds to the following $PML_T(\mathcal{L}, n)$ formula in type normal form:

$$\begin{array}{l}(S \wedge \langle\uparrow\rangle\lambda \wedge \langle\leftarrow\rangle\lambda \wedge \langle\rightarrow\rangle\lambda \wedge \langle\downarrow\rangle a) \vee \\ (S \wedge \langle\uparrow\rangle S \wedge \langle\leftarrow\rangle a \wedge \langle\rightarrow\rangle\lambda \wedge \langle\downarrow\rangle a) \vee \\ (a \wedge \langle\uparrow\rangle S \wedge \langle\leftarrow\rangle\lambda \wedge \langle\downarrow\rangle\lambda)\end{array}$$

Actually, this formula describes three types of tree nodes we call ν_1, ν_2 and ν_3. Each of them uniquely corresponds to one of the alternatives above. Every labeled tree that satisfies the formula above only consists of nodes agreeing with these types. To establish a corresponding disjunctive rational tree we consider the possible structural relation between these types. The relation γ_d denotes a possible immediate dominance relation while γ_r denotes a possible immediate sibling precedence relation. We achieve the following relations among the types ν_1, ν_2 and ν_3:

$$\begin{array}{rcl}\gamma_d & = & \{(\lambda, \nu_1), (\nu_1, \nu_2), (\nu_1, \nu_3), (\nu_2, \nu_2), (\nu_2, \nu_3), (\nu_3, \lambda)\} \\ \gamma_r & = & \{(\lambda, \nu_1), (\nu_1, \lambda), (\lambda, \nu_3), (\nu_3, \nu_2), (\nu_2, \lambda), (\nu_3, \lambda), \}\end{array}$$

In addition, we use the empty node statement λ to express that a type forbids the existence of a particular immediate neighbor. Unfortunately, these relations are not sufficient to establish the corresponding disjunctive rational tree. In detail, we require the disjunction of the valid strings of children nodes as used in disjunctive rational trees. For instance, we only obtain from the relation γ_d and γ_r that a and S are possible children of S. However, there is no explicit statement providing the valid combinations of the possible children, i.e. which combinations of a and S are valid. Nevertheless, we can compute this information from those relations. First we consider all node types that can be a child of a given particular type. For instance, the possible children of the type v_2 are v_2 and v_3. The same applies to the type v_1. Accordingly, we obtain for the type v_2 a restricted relation $\gamma_r(v_2)$ only including the possible relation between children types v_2 and v_3:

$$\gamma_r(v_2) = \{(\lambda, v_3), (v_3, v_2), (v_2, \lambda), (v_3, \lambda), \}$$

Moreover, we achieve the same result for $\gamma_r(v_1)$. Based on this set we can construct the corresponding set of 'children type strings'. This set represents the set of valid children strings $\mathtt{children}(v_2) = \{v_3, v_3 v_2\}$.

In general Palm (1997) has shown for a similar formalism that the set of strings corresponding to the relation γ_r and its restricted versions have the generative power of regular expressions of (node-) types. Since this example uses only binary branching, the resulting set of strings is finite and it only includes strings of the maximal length two. Moreover, every string of possible children denotes a particular alternative. Therefore, we assume for each string a disjunction node in the corresponding disjunctive rational tree. So we have got sufficient information to construct a disjunctive rational tree that describes the same set of trees as the underlying $\mathrm{PML}_T(\mathcal{L}, n)$ formula. Every node type constitutes a branching node and every element of $\mathtt{children}(v)$ for some node type v constitutes a disjunction node. A node type v states a root node if $(\lambda, v) \in \gamma_d$ is true. The downward relation d associates a former node type with one of its children strings which is now represented by a disjunction node. On the other hand, the downward relation associates the disjunction node with the corresponding elements of the string. More simply, the rightward relation straightforwardly corresponds to the relation γ_r. Using this method we obtain the following disjunctive rational tree which is based on the node types v_1, v_2, and v_3. In detail we employ the label domain $\mathcal{L} = \{S, a\}$, the set of branching nodes $N = \{n_1, n_2, n_3\}$ corresponding to the types v_1, v_2, and v_3 where $N_S = \{n_1\}$ states the single root node. Moreover we need a set of two disjunction nodes $N_d = \{m_1, m_2\}$. The rightward relation r only includes a single pair (n_3, n_2). Finally the downward relation and the label assignment are defined as follows:

$$d = \{(n_1, m_1), (n_1, m_2), (n_2, m_1), (n_2, m_2), (m_1, n_3), (m_2, n_3), (m_2, n_2)\}$$
$$a = \{n_1 \to S, n_2 \to S, n_3 \to a\}$$

The branching nodes n_1, n_2, and n_3 correspond to the node types v_1, v_2, and $v3$, respectively. Since v_1 and v_2 share the same children string, we only require two disjunction nodes m_1 and m_2. They correspond to the strings 'v_3' and '$v_3 v_2$'. This definition leads to the graphical representation in Figure 7. Since

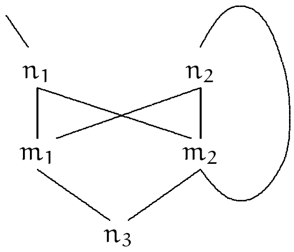


FIGURE 7

the nodes n_1 and n_2 share the same label and the same set of children strings, we can combine them to single one. Consequently, we obtain the same disjunctive rational tree as used previously.

Although we have only sketched this method to transform a $\mathsf{PML_T}(\mathcal{L}, n)$ formula into a corresponding disjunctive rational trees, it works in general. However, because of the generally arbitrary degree of branching, the disjunctive rational trees used so far are to weak to represent every $\mathsf{PML_T}(\mathcal{L}, n)$ formula. Palm (1997) has discussed a further extension of disjunctive rational trees called the *regular rational trees* that captures sets of arbitrarily branching trees. The crucial difference means the use of regular expressions of nodes to represent the set of children strings. Obviously, by dealing with regular expressions, we capture the generative power of the relation γ_r.

So far we have discussed the relationship between disjunctive rational trees and $\mathsf{PML_T}(\mathcal{L}, n)$ formulae. Now we turn to the other direction concerning the correspondence to the context-free grammars. Actually, in a disjunctive rational tree every branching node and its alternative set of children strings correspond to a context-free rule. In the previous example a node with the label S has either the sequence of children aS or a. So, we can form the corresponding context-free rules $S \rightarrow aS$ and $S \rightarrow a$. Since a never has any children it states a leaf, and hence, we may regard it as a terminal symbol. The symbol S constitutes the root and, consequently, the single non-terminal symbol. In contrast to the 'standard' definition of context-free grammars, a disjunctive rational tree neither demands a single root label nor it distinguishes terminal and non-terminal node labels. Consequently, we establish a modified notion of context-free grammars. In detail, a context-free grammar G is defined as a tuple $(\Sigma, \Sigma_T, P, \Sigma_S)$

where Σ denotes the set of (node-) *symbols*, $\Sigma_T \subseteq \Sigma$ denotes the non-empty set of *possible terminal symbols*, $P \subseteq (\Sigma \leftarrow \Sigma^+)$ denotes the set of *production rules*, and $\Sigma_S \subseteq \Sigma$ denotes the set of *start symbols*. Based on this modified notion we achieve a straightforward method to transform disjunctive rational trees into context-free grammars. However, this does not work in general. Consider the disjunctive rational tree R depicted in Figure 8. Actually, R describes the

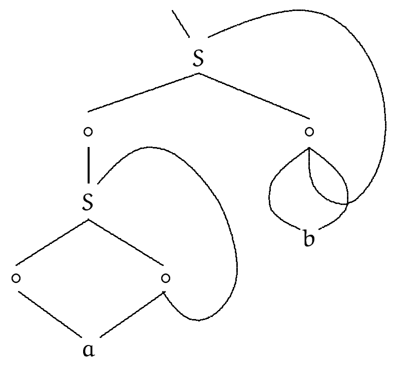


FIGURE 8. A disjunctive rational tree for $b^m a^n b^m$

language $L(R) = \{b^m a^n b^m | m \geq 0 \wedge n \geq 0\}$. If we 'blindly' transform this disjunctive rational tree, we achieve the following context-free grammar:

$$G = (\{S, a, b\}, \{a, b\}, \{S \to bSb, S \to S, S \to Sa, S \to a\}, \{S\})$$

However, the grammar G does not specify the same set of trees as the disjunctive rational tree R since $L(G)$ includes the word '$baba$' which is not a word of $L(R)$. The crucial property of R is the doubled occurrence of the label S used as branching nodes with different structural properties. Therefore, we require a *one-to-one* relation between branching nodes and their labels. However, we must extend the label domain once more to satisfy this condition. In the example above we distinguish the two occurrences of the former S-labels by the new 'indexed' labels S_1 and S_2:

$$G = (\{S_1, S_2, a, b\}, \{a, b\}, \{S_1 \to bS_1b, S_1 \to S_2, S_2 \to S_2a, S_2 \to a\}, \{S_1\})$$

Hence, we can transform a disjunctive rational tree into a corresponding context-free grammar provided that the labels and the branching nodes correspond one-to-one. In this case, we can equalize the labels and the branching nodes. We call this version a *normal disjunctive rational tree*. Thus, every normal disjunctive rational tree corresponds to a context-free grammar. Trivially, this relationship holds for the converse direction, too. In addition, we can normalize every arbitrary disjunctive rational tree by extending the label domain. In general, the correspondence to the described set of trees only holds for the $\mathcal{L}$-projection of the labels.

As mentioned earlier, the disjunctive rational trees only capture $\mathsf{PML_T}(\mathcal{L}, n)$ formulae including a bound branching constraints. However, we must utilize *regular* rational trees to represent arbitrary degrees of branching. The same branching restriction applies to the context-free grammars. Obviously every rule states a fixed degree of branching. A suitable extension of context-free grammars has been provided by Thatcher (1967). The most significant difference is that the right-hand side of a rule states a regular expression of terminal and non-terminal symbols. These extended context-free grammars correspond to the *regular rational trees* mentioned previously. Palm (1997) has established the relationship between $\mathsf{PML_T}(\mathcal{L}, n)$ formulae, regular rational trees and extended context-free grammars. Although the extended context-free grammars mean a proper extension of the context-free grammar, they share, however, the same weak generative capacity, i.e. the extended context-free languages coincide with the context-free languages.

So far we have sketched the general transformation of the first-order formalism CLAT($\mathcal{L}$) into the (extended) context-free grammar. Hence, we can transform an arbitrary CLAT($\mathcal{L}$) formula into a corresponding (extended) context-free grammar such that both formalisms specify the same set of finite trees up to the projection on the former label domain $\mathcal{L}$. We have preserved the expressivity of the formalisms visited by extending the domain of labels. In contrast to the transformation from monadic second-order logic into context-free grammars provided by Rabin (1969) and Doner (1970), our method deals with arbitrary branching trees and performs more transparent and direct alternations. Therefore, we can investigate the expressivity of single principles and the relationship among principles with respect to the described set of trees. Moreover, the resulting (extended) context-free grammars provide an efficient method to construct the set of trees that is described by a principle formulated in one of the presented tree languages.

6. EXTENSIONS

Our approach of transforming principles into rule-based grammars includes two connected restrictions. On the one hand, we can only describe sets of trees generated by (extended) context-free grammars. On the other hand, the finiteness of the label domain prevents us from using natural numbers or similar labels. Both restrictions constitute some inconsistencies with respect to particular properties of some natural languages. The most crucial phenomenon we can observe means the 'cross-serial dependencies' in Dutch, cf. Huybregts (1976), Bresnan et al. (1982), and in Swiss German, cf. Huybregts (1984), Shieber (1985). In both cases the generative power of context-free languages fails to capture the general relationship between nouns and verbs:

... *dat Jan*$_1$ *Marie*$_2$ *Piet*$_3$ *zag*$_1$ *helpen*$_2$ *zwemmen*$_3$
... *dass mer*$_1$ *d'chind*$_2$ *em Hans*$_3$ *s'hus lönd*$_1$ *hälfe*$_2$ *aastriche*$_3$

The cross-serial dependencies state an instance of the context-sensitive 'copy language' $L_{copy} = \{ww | w \in \Sigma^*\}$ for a given alphabet Σ. Another reason to extend our approach forces the restriction to a finite labeled domain $\mathcal{L}$. In G/B and most other grammar formalisms, we often employ 'indices' to indicate related parts of a sentence. Typically, noun phrases referring to the same entity receive the same index. We employ common indices to bind pronouns and anaphors as well as to link traces and their antecedents.

John$_i$ *thinks that Mary loves him*$_i$
who$_i$ *did we think that Mary loves t*$_i$

If the number of possible indices occurring in a sentence is bound then we can still employ a finite label domain. But if a linguistic theory gives rise to an arbitrary number of possible different indices within a single sentence, we require an infinite label domain $\mathcal{L} \times \mathcal{I}$ where the first component $\mathcal{L}$ denotes to former finite label domain while $\mathcal{I}$ represents the infinite domain of indices. Unfortunately, we require a modification of our approach to capture infinite label domains. Rogers (1995) has already shown that the free association of indices to noun phrases cannot be expressed in his formalism $L^2_{K,P}$. Provided bound branching CLAT($\mathcal{L}$) and $PDL_T(\mathcal{L})$ state subsets of $L^2_{K,P}$. Consequently, the same restriction applies to our approach, too.

The crucial issue of an infinite label domain is not the resulting infinite number of label predicates (in CLAT($\mathcal{L}$)) or label statements (in $PDL_T(\mathcal{L})$ and $PML_T(\mathcal{L}, n)$). But it is the relationship between the labels of two nodes. Although the syntax of CLAT($\mathcal{L}$) does not provide an explicit method to compare the labels of two nodes, we can express this relationship. For instance, we can define a predicate $\mathtt{equallabel}(x, y)$ that is true if the nodes x and y have the same label:

$$\mathtt{equallabel}(x, y) := \bigvee_{l \in \mathcal{L}} l(x) \wedge l(y)$$

Obviously, for an infinite label domain this predicate corresponds to an infinite formula. Therefore, a reasonable use of an infinite label domain should also include the binary node predicates concerning the relationship between the labels of the nodes considered.

For a more general approach we use *stacks* as the infinite label domain. In detail, we assume the infinite label domain $\mathcal{L} \times \mathcal{J}^*$ which combines the finite label domain $\mathcal{L}$ and the infinite domain of stacks $\mathcal{J}^*$ where $\mathcal{J}$ denotes the finite stack alphabet. Moreover we introduce some binary stack predicates to express that the stacks of two tree nodes x and y satisfy a particular relation. In essence, such a relation states a certain stack action σ such that $\mathrm{stack}(y) = \sigma(\mathrm{stack}(x))$ is true. The typical stack actions are the push_a operation that puts the symbol a on the top of the stack, the pop_a operation that removes the symbol a from the top of the stack, and the nop operation constituting 'no operation', i.e. the stacks of x and y must be equal.

$$\begin{array}{rcl} \mathrm{push}_a(x,y) & \equiv & \mathrm{stack}(y) = a \circ \mathrm{stack}(x) \\ \mathrm{pop}_a(x,y) & \equiv & \mathrm{stack}(x) = a \circ \mathrm{stack}(y) \\ \mathrm{nop}(x,y) & \equiv & \mathrm{stack}(x) = \mathrm{stack}(y) \end{array}$$

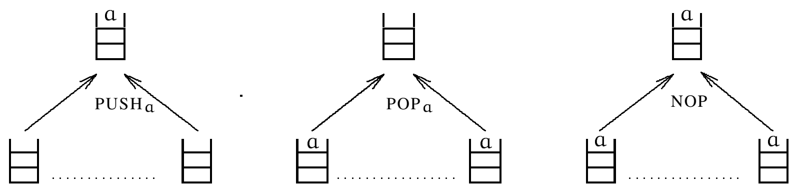
We call these relations the *stack predicates*. Obviously, push_a and pop_a constitute complementary relations since we can conclude from their definitions that $\mathrm{push}_a(x,y) \equiv \mathrm{pop}_a(y,x)$ holds for every stack symbol $a \in \mathcal{J}$. By employing the pop-operation only in connection with a certain stack symbol a, we prevent the popping of symbols from an empty stack.

Actually, the stack predicates $\mathrm{push}_a(x,y)$, $\mathrm{pop}_a(x,y)$ and $\mathrm{nop}(x,y)$ mean a new type of structural relation. Instead of certain structural properties, they state a relation between tree nodes with depending stacks. However, by arbitrarily using the stack predicates we can simulate the configurations of a Turing machine, cf. Palm (1997). Hence, such a formalism describes sets of trees corresponding to a type-0 language. Unfortunately, the resulting formalism is too strong for the linguistic purposes. But for describing the copy language we only require a weak extension of the context-free grammars. Like the context-free languages the copy language has the 'linear growth property'. In general we say a language grows linearly if there is a constant k such that for every word (sentence) of the language with the length n there is word of the length $n + k$. Obviously, for the copy language $k = 2$ obtains.

Therefore, we provide an extension that is in fact weaker than the one described above but that captures the copy language, too. Moreover, this formalism should be easy to integrate in the former approach. The basic idea is to encode the stack domain into a particular finite label domain. Then we can employ our transformation into the (extended) context-free grammars. Afterwards we must undo the previous encoding to get back to the intended set of trees. Although a stack constitutes an infinite domain, the number of possible stack predicates between two nodes is only finite. Based on this observation, we employ the stack predicates as labels instead of the stacks. However, this works only if we know the position of the node with the related stack. For instance, between two nodes x and y holds a stack predicate, say $push_a(x, y)$. We assume a label PUSH_a and a partial function f_{stack} mapping the node x to the node y. Then we can establish an equation between the unary label predicate and the binary stack relation:

$$\begin{aligned} \mathrm{PUSH}_a(x) &\equiv push_a(x, f_{stack}(x)) \\ \mathrm{POP}_a(x) &\equiv pop_a(x, f_{stack}(x)) \\ \mathrm{NOP}(x) &\equiv nop(x, f_{stack}(x)) \end{aligned}$$

By assuming a function f_{stack} that maps a node x to the node y with the depending stack, we can replace the binary stack predicate with a monadic one. Since the number of necessary monadic stack predicates is finite, we can encode the stacks in a finite label domain. Hence, we only represent the difference between two depending stacks rather than their absolute values. The basic restriction means the functional relation between nodes with depending stacks. In contrast, the method discussed previously permits arbitrary structural relations between nodes with depending stacks. So we talk of a *functional stack formalism* where the term 'functional' refers to the structural relation between the tree nodes with depending stacks. The partial function f_{stack} must be defined by means of the structural relations and the label predicates corresponding to the finite label domain $\mathcal{L}$. In addition, we require the monadic predicate $endstack(x)$ which indicates that $f_{stack}(x)$ is undefined or, in other words, the stack of x does not depend on any further stack. For instance, consider the following stacks of a node and its children:



Here we assume that f_{stack} yields the parent of a node, i.e. the stack of a node and its children depend on each other. Moreover, the stack actions between a node and each of its children are equal. However, this configuration agrees with the properties of *indexed grammars*, cf. Aho (1968) and Gazdar (1985) for some linguistic applications. Therefore, the trees described by this formalism corresponds to the trees generated by indexed grammars, cf. Palm (1997). Thus, this formalism captures the copy language since it constitutes an indexed language. In a similar manner, we can establish a correspondence between the *linear indexed grammars*, and a functional stack formalism where the function f_{stack} yields the unique child of a node with the depending stack. For a more detailed discussion we refer to Palm (1997).

So far, we have only considered how to encode the binary stack predicates. After encoding the stacks we now turn to a method to get back the set of trees described formerly. By means of encoding the stack we can transform a given principle into a corresponding (extended) context-free grammar. But this grammar also generates trees with invalid stack actions. For instance, the stack actions $\text{PUSH}_a(x)$ and $\text{POP}_b(y)$ with $y \approx f_{stack}(x)$ are invalid since after pushing the symbol a we cannot pop the symbol b. Actually, a sequence of stack actions $\sigma_0, \sigma_1, \ldots, \sigma_k$ must describe the configurations of a *push down automaton* see (see e.g. Partee et al. 1990) where the stack actions correspond to the nodes $x_0, x_1, \ldots, x_k$ with depending stacks $f_{stack}(x_{i-1}) \approx x_i$ for $1 \leq i \leq k$. Consequently, we can verify the validity of the stack actions by employing a push down automaton. Since such an automaton corresponds to a context-free grammar, we can establish the context-free grammar of valid stack actions. Based on this grammar of valid stack actions, we can reconstruct the actual stack values.

The functional stack formalism we have considered so far means a reasonable extension of our approach that is strong enough to capture the properties of natural language but that is also weaker than type-0 languages. Nevertheless, it is uncertain how to employ this extension to capture the previously mentioned problem of indexing co-referring noun phrases. Here we can use two basic methods. The first one equalizes the chains of co-indexed noun phrases and sequences of nodes with depending stacks. In contrast, the second method equalizes each index with a particular stack value. The crucial issue of both methods states the function f_{stack}. For instance, we typically move noun phrases to a specifier position of a functional category, i.e. IP or a CP. Hence, we could define f_{stack} as a function mapping a node to the next specifier node that must satisfy some additional constraints φ. We specify f_{stack} by means of a $\mathrm{PDL}_T(\mathcal{L})$ path

$$f_{stack} := \uparrow; (\neg(\langle\uparrow\rangle max \wedge \langle\leftarrow \cup \rightarrow\rangle(\neg max \wedge \varphi)))^* ; \leftarrow \cup \rightarrow; (\neg max \wedge \varphi)?$$

where we assume that the result of the function is the first node to be reached by the path. The above path 'searches' the next specifier of a node satisfying the condition φ. In detail, the formula $\langle\uparrow\rangle max \wedge \langle\leftarrow \cup \rightarrow\rangle(\neg max \wedge \varphi)$ indicates the non-maximal (left-hand or right-hand) sibling of a specifier, i.e. a maximal node with a maximal parent. We go upwards to the first node that satisfies this condition. Due to the binary branching, this nodes has a single sibling that means the searched specifier node. In the condition φ we can add some particular properties of the specifier considered. Hence, we can employ function f_{stack} to describe a 'chain' of specifiers nodes with depending stacks.

However, the suggested function f_{stack} means only a first step to describe nodes sharing the same index. The additional condition φ serves to include movement constraints. However, how we can employ this extension to represent binding and movement constraints of G/B is still an open question. Nevertheless, a functional stack formalism provides the expressivity of indexed grammars and other context-sensitive grammar formalism. Therefore, this extension should be strong enough to represent the indices as they are employed in G/B and related approaches.

7. Conclusion

The presented discussion on the expressivity of certain tree languages has provided a variety of results covering formal and linguistic issues. One basic result is the straightforward transformation of CLAT($\mathcal{L}$) principles into a corresponding (extended) context-free grammar. We have employed propositional dynamic logic ($\mathsf{PDL}_T(\mathcal{L})$) and propositional modal logic ($\mathsf{PML}_T(\mathcal{L}, n)$) as the main intermediate representation. While the transformation between the first-order formalism CLAT($\mathcal{L}$) and $\mathsf{PDL}_T(\mathcal{L})$ preserves the given finite label domain $\mathcal{L}$, both the transformations from $\mathsf{PDL}_T(\mathcal{L})$ into $\mathsf{PML}_T(\mathcal{L}, n)$ and from $\mathsf{PML}_T(\mathcal{L}, n)$ into (extended) context-free grammars require an extension of the label domain. In general, we can preserve the expressivity between non-equivalent formalisms describing trees by extending the label domain. Accordingly we never leave the realm of the (extended) context-free grammars as long as the label domain is finite.

Our approach of several tree languages provides a transparent method to obtain a rule-based grammar formalism from a declarative description. So we can follow the effects of certain principle on the set of trees described. We can verify the formulation of a principle by comparing the intended structures with the resulting trees. Starting with a given CLAT($\mathcal{L}$) principle, the transformation into $\mathsf{PDL}_T(\mathcal{L})$ results in an explicit, path-based description of the structural relations formerly occurred in the principle. Moreover, the correspondence to $\mathsf{PML}_T(\mathcal{L}, n)$ illustrates the relationship between local formulations and long-distance dependencies. For instance, we have seen that the x-bar scheme actually includes only local constraints although the underlying principle employs long-distance paths. The type normal form of $\mathsf{PML}_T(\mathcal{L}, n)$ formulae exhibits the different types of tree nodes corresponding to a principle. Finally, on the one hand the disjunctive rational trees clearly represent the valid structural relations between the types of nodes. But on the other hand, they actually state a certain manner of specifying a context-free grammar. Altogether we can trace a principle while we transform it step-by-step into a ruled-based approach. The latter provides an explicit and complete description of the set of trees specified.

In addition, the transparency of our approach combines two different notions of regarding a set of trees. The declarative view of the first order formalism CLAT($\mathcal{L}$) emphasizes the partial and modular aspects of describing a set of trees. Consequently, a CLAT($\mathcal{L}$) principle can focus on a certain aspect of trees without considering the properties of the whole structure. In contrast, a rule-based description means a complete specification of the considered part of the tree. For instance, a principle may only consider some child of node while a context-free rule must define all children of a node. However, in contrast to principles the rule-based grammar formalisms offer an efficient method to construct trees. Therefore, a transformation between these different kinds of formalism enables us to combine the benefits following from either of them.

Moreover, the extension to functional stack formalisms posits some interesting aspects on viewing binding and movement in G/B and related approaches. So far the finite label domain has kept us within the realm of (extended) context-free grammars. Albeit the relevance of context-sensitive languages to describe natural language is rather controversial, the problem of indexing co-referring noun phrases motivates an extension of our approach. The basic idea is to employ stacks as an additional infinite label domain. However, we must restrict the structural relation between nodes with depending stacks to keep the resulting approach in a reasonable range of complexity. Nevertheless, the functional stack formalism captures some well-known kinds of context-sensitive grammars, e.g. indexed grammars and linear indexed grammars, that claim to own sufficient generative power to describe natural language. Although we have made some suggestions concerning a method to represent movement and binding by a function stack formalism, we can only speculate on an effective formulation. Despite this, we obtained from the properties of functional stacks that the structural domain to co-index noun phrases must be rather restricted.

REFERENCES

Aho, A. (1968): Indexed grammars – an extension of context-free grammars. *Journal of the Association for Computing Machinery* 15:647–671

Backofen, R., J. Rogers, and K. Vijay-Shanker (1995): A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language and Information* 4:5–39

Blackburn, P. and W. Meyer-Viol (1994): Linguistics, logic and finite trees. *Bulletin of the IGPL* 2:2–29

Bresnan, J., R. Kaplan, S. Peters, and A. Zaenen (1982): Cross-serial dependencies in dutch. *Linguistic Incquiry* 13:613–635

Bull, R. and K. Segerberg (1984): Basic modal logic. In: Gabbay and Guenthner (1984), pp. 1–88

Chomsky, N. (1981): *Lectures on Government and Binding*. Dordrecht: Foris

Chomsky, N. (1986): *Knowledge of Language. Its Nature Origin and Use*. New York: Praeger

Doner, J. (1970): Tree acceptors and some of their applications. *Journal of Computer and System Science* 4:406–451

Gabbay, D. and F. Guenthner (eds.) (1984): *Handbook of Philososphical Logic vol. 2*. Dordrecht: Reidel

Gazdar, G. (1985): Applicability of indexed grammars to natural language. Technical report, Center for the Study of Language and Information, Stanford

Gazdar, G., E. Klein, G. Pullum, and I. Sag (1985): *Generalized Phrase Structure Grammar*. Oxford: Blackwell

Harel, D. (1984): Dynamic logic. In: Gabbay and Guenthner (1984), pp. 497–604

Huybregts, M. (1976): Overlapping dependencies in dutch. *Utrecht Working Papers in Linguistics* 1:24–65

Huybregts, M. (1984): The weak adequacy of context-free phrase structure grammar. In: G. de Haan, M. Trommelen, and W. Zonneveld (eds.), *Van periferie naar kern*, Dordrecht: Foris, pp. 81–99

Kaplan, R. and J. Bresnan (1982): Lexical Functional Grammar. In: J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, Cambridge, MA: MIT Press

Kracht, M. (1995): Syntactic codes and grammar refinement. *Journal of Logic, Language and Information* 4:41–60

Palm, A. (1997): *Tranforming Tree Constraints into Rules of Grammars*, vol. 173 of *DISKI*. St. Augustin: infix-Verlag

Partee, B., A. Meulen, and R. Wall (1990): *Mathematical Methods in Linguistics*. Dordrecht: Kluwer Academic Publishers

Pollard, C. and I. Sag (1987): *Information-based Syntax and Semantics*. Stanford, California: CSLI

Pollard, C. and I. Sag (1994): *Head-driven Phrase Structure Grammar*. Chicago: University of Chicago Press

Rabin, M. (1969): Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society* 141:1–35

Rogers, J. (1995): *Studies in the Logic of Trees with Application to Grammar Formalisms*. Ph.D. thesis, University of Delaware, Newark

Shieber, S. (1985): Evidence against the context-freeness of natural language. *Linguistic and Philosophy* 8:334–343

Shieber, S. (1986): *An Introduction to Unification-based Approaches to Grammar*. Stanford, California: CSLI

Shieber, S. (1992): *Constraint-based grammar formalisms*. Cambridge, MA: MIT Press

Stabler, E. (1992): *A logical approach to syntax*. Cambridge, MA: MIT Press

Thatcher, J. (1967): Characterizing derivation trees of context-free grammars. *Journal of Computer and System Science* 2:57–81

DEPARTMENT OF GENERAL LINGUISTICS, UNIVERSITY OF PASSAU, D-94030 PASSAU
E-mail address: palm@phil.uni-passau.de