Graph Transformations, Proofs, and Grammars

Bertrand Boisvert
IRIT, Université Toulouse 3
118, route de Narbonne, 31062, Toulouse, France
boisvert@irit.fr

Louis Féraud
IRIT, Université Toulouse 3
118, route de Narbonne, 31062, Toulouse, France
feraud@irit.fr

Sergei Soloviev
IRIT, Université Toulouse 3
118, route de Narbonne, 31062, Toulouse, France
soloviev@irit.fr

Abstract: Graph transformation systems or graph grammars are a generalization of formal grammars used in linguistics.One of principal differences is that the structures that are transformed by transformation rules are no more linear. In this paper we consider graph transformation systems in connection with proof theory. We develop the graph transformation framework, where the attributes are derivable judgments of some deductive system. The computation functions are represented by partial proofs. We consider the application of graph transformation systems to the description of Kleene-style permutation of rules, to the derivations with "distant links", to "parallel derivations" and "presupposed judgements". As a "feedback" to linguistics, we discuss applications to the linguistic derivations and their transformations, and some aspects of the notion of "presupposition".

Keywords: Graph Transformations, Proof Transformations, Formal Grammar, Presupposition

Introduction

Graph Transformation Systems (GTS) and Graph Grammars are a generalization of the notion of ordinary textual grammars. An important difference is that instead of linear sequences of symbols (words) more complex geometric structures (graphs) are transformed. It creates new problems that do not appear in case of textual grammars, e.g., how to deal with the edges that lost their source or target. In comparison with textual grammars, GTS provide much richer possibilities for direct scientific modelling (without complex encoding). They are largely used in computer science, for example in Model Driven Engineering, and in representation of graphical languages, such as UML. GTS-based approaches interact with many other themes of importance in theoretical and applied computer science and logics, *i.e.*, term rewriting and reduction systems (Barendregt, van Eekelen, Kennaway, Plasmeijer, & Sleep, 1987). Another domain of interest, not yet sufficiently explored, is proof theory. The tree-form representation of proofs is common. Other graph structures are sometimes considered (e.g., proof nets), but there is no unifying framework for these structures.

Concerning the formal theory of graph rewriting, a lot of significant results are due to H. Ehrig and his colleagues who have conceived an algebraic approach to graph rewriting by the means of category theory (Rozenberg, 1997). They worked mostly with the so called double pushout (DPO) approach in a graph category. Another approach using graph categories and simple pushout (SPo) was first suggested by Löwe (Rozenberg, 1997).

In most cases, graphs in GTS are attributed. An attributed graph is composed of a structural part (the nodes and edges of the graph), and of attributes that are "attached" to the nodes or

edges. A transformation of an attributed graph must both transform the structure and do some computations on attributes. One of the challenges of attributed GTS concerns the implementation of attribute computations. Most of the existing systems use a relatively weak formalism for attributes, the algebraic date types. For example, they do not admit functional expressions as attributes, and do not propose any inherent computational mechanism.

In the beginning of our studies of graph transformations we considered as attributes the λ-terms of simply typed λ-calculus with inductive types (Boisvert, Féraud, & Soloviev, 2011). Partly our inspiration came from linguistics: important advances in studies of natural languages were due to constructive type theory, see, *e.g.*, the books (Ranta, 1994; Pustejovsky, 1995). Type theory permitted to treat in a uniform way different natural languages. Probably most important to us in these studies was that type theory had shown its great unifying potential for interdisciplinary research. Its use seems appropriate when one has to work with heterogenous components even in case of mostly mathematical formalism, like the theory of GTS.

The formalism based on λ-calculus permitted already the use of functional attributes, not possible in case of algebraic data types. Due to recursion operators for inductive types, the natural computational mechanism was included. Its good properties were guaranteed by standard theorems concerning strong normalisation and confluence.

More recently we developed an even more general framework where attributes may be judgements of any chosen type theoretical or logical system (such as typing judgements $\Gamma \vdash t : A$ or logical sequents $\Gamma \vdash A$). Computation functions for attributes in this framework are represented by partial proofs. This approach provides a better unifying framework and permits also to treat constructively more complex graph structures (besides ordinary proof trees) associated with deductions in proof theory.

As a "feedback" to linguistics, graph transformations may be of interest in all situations where linguistic derivations and their transformations are considered.

In the next section we present briefly our graph transformation formalism, specifically designed for work with logical systems and proofs. Afterwards, we consider several examples of its application to Proof Theory, and outline several areas of linguistics where one may expect interesting applications. Short conclusion is included in the end.

1. Technical Definitions and Results

This section generalizes the approach to GTS developed in (Boisvert et al., 2011). Main novelty consists in considerable generalization of the definitions of attributes and attribute computations. The section includes two theorems, that provide a justification for graph transformation rules based on pushout construction. In difference from definitions, the proofs of these theorems need only a slight modification of proofs published in (Boisvert et al., 2011).

1.1. Category of Attributed Graphs Gr^T.

Let T be some deductive system defined by axioms and rules of inference in usual sense. We shall denote by J the judgements of this system[1]. Objects of Gr^T are attributed graphs. For any graph G the sets of its vertices and edges will be denoted respectively $V(G)$ and $E(G)$. The elements of the set $V(G) \cup E(G)$ will be called "elements of the graph". In this paper, we shall assume that there is a standard (lexicographic) ordering on $V(G) \cup E(G)$. The attributes are

[1]Some terminological rectification may be useful here. One may notice that in proof theory the terms "axiom" and "rule" apply usually to axiom scheme and rule scheme. The judgements of different theories may have different forms. For example, $A_1, ..., A_n \vdash A$ (in logics), $x_1 : A_1, ..., x_n : A_n \vdash t : A$ (in type theory). Often (but not always) the notion of judgement is defined independently of derivability. The difference is not relevant for our main definitions. That is, the definitions need only slight modifications to accomodate these differences.

judgements.There is exactly one attribute associated with each element of the graph[2]. The set of all attributes of graph G will be denoted Att(G).

The definitions concerning the notion of partial proof are common in literature. We give below a variant adapted to our case and refer to one of many possible sources (Bundy, 1988).

Definition 1.1. A partial proof is a tree with the following properties:

- Each node is labeled [3] with a judgement.

- Each node except the leaves is labeled also with the rule of inference which is applied to this judgement (backwards) to produce the node's children (and the children of course must be the premises of this rule).

- The final judgement (or the goal) is the judgement at the root of the tree.

- Among the leaves we distinguish "active" and "parameter" leaves[4]. Active leaves are fixed in arbitrary way among all leaves. Remaining leaves are considered as parameter ones.

There is some flexibility in this definition. A possible (but not necessary) restriction is that all leaves have only *derivable* judgements as labels. The judgements in partial proofs may be considered up to some equality (for exaple, based on conversion, like in λ-calculus).

The trees will be considered as equal if they are equal as elements of a corresponding inductive type, i.e., the subtrees are ordered from left to right similarly to the arguments of a type constructor. Partial proofs are equal if the trees are equal and the labels of equal nodes are equal.

We will consider the operation of *composition* of two partial proofs. Let an active leaf of a partial proof P_1 be equal to the final sequent (root) of another partial proof P_2. The composition of two trees via this leaf is obtained by replacement of the leaf (with its label) by another tree (with its labels). The result is obviously another partial proof.

Our definition of morphisms of Gr^T will use the notion of partial proof[5].

Definition 1.2. A morphism $f : G \to H$ of the category Gr^T has the following three-level structure.

- The "structural part" noted by f_{str} is a partial graph homomorphism (Rozenberg, 1997) from the stuctural part of G to the structural part of H.

- The "attribute dependency relation" f_{adr} is a relation between $V(G) \cup E(G)$ and $V(H) \cup E(H)$. For each $e \in V(H) \cup E(H)$ its preimage (*i.e.* the set of all its antecedents) is denoted by $[e]_{adr}$. The attributes of these antecedents are the attributes of G that can be used to derive the attribute of e.

- The "computational part" is represented by a partial proof for each element $e \in V(H) \cup E(H)$. The active leaves of the partial proof corresponding to e must be in one-to-one correspondence with the elements of $[e]_{adr}$, the judgements used as labels of these leaves must be equal to the attributes of corresponding elements of G (remember that we assume the elements of G and the leaves to be ordered), and the judgement at the root equal to the attribute of e.

Definition 1.3. Two morphisms $f, g : G \to H$ are equal iff:

- $f_{str} = g_{str}$;

- the relations $f_{adr} = g_{adr}$;

---

[2] In many systems, for example, λ-calculus with pairing, it can be used to represent the tuples of attributes. Otherwise, the records may be used but we shall not go into these details here. The existence of some trivial judgement, like $\vdash T$ in logic could be useful to represent the absence of non-trivial attributes.

[3] We may say also "attributes" here, but we do not consider these trees as *objects* of the category of attributed graphs.

[4] When we use partial proofs to represent computations with attributes, the distinction between axioms and other judgements (in difference of proof-search) is no more relevant.

[5] The idea to use trees with labels (judgements) in transformations of attributed graphs suggests to use of more complex "interleaving" structures (graphs in some way alternated with attributes). One may notice, though, that trees are a very special subclass of graphs. They can be easily represented as elements of an inductive type, in difference of graphs in general.

- and for each $e \in V(H) \cup E(H)$ the corresponding partial proofs have equal active leaves (taken in their order) and equal conclusions.

(One may notice that the equality of morphisms does not require the equality of corresponding partial proofs.)

Definition 1.4. Composition of morphisms is defined as follows:

- on the level of structure, composition of partial graph homomorphisms is used;
- on the level of attribute dependency relations composition of relations is used;
- On the level of partial proofs the composition of partial proofs (via active leaves) is used.

Definition 1.5. The identity morphisms are defined as follows:

- the identity graph homomorphism for the structural part;
- the identity relation for the attribute dependency relation;
- and the partial proofs have exactly one active leaf (= root), with the label J equal to the attribute of the corresponding element of the graph.

Theorem 1.1. *Attributed graphs and graph morphisms described above form a category.*

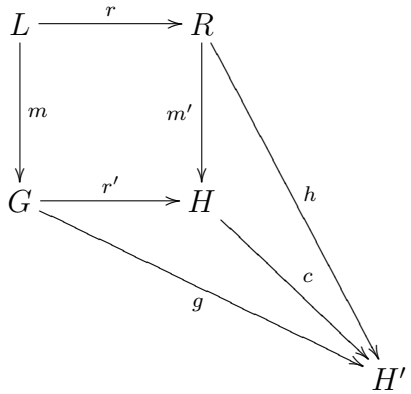
The proof consists in direct verification, based on definitions.

1.2. Graph Transformations in Gr^T.

A useful class of morphisms are the *embeddings* $G \to H$, represented by embeddings on the levels of structure and attribute dependency relation, where partial proofs are defined as above, but only for the elements of H that have non-empty set of antecedents.

As in the traditional SPo approach (Rozenberg, 1997), each morphism $r : L \to R$ of Gr^T defines a graph transformation rule. More precisely, to define an application of this rule, another morphism $m : L \to G$ being an embedding is necessary (m is usually called *match* and G is called *host graph*). To apply the rule to some subgraph of a larger graph we need to embed L as a subgraph of some graph G and to find the corresponding right side H.

A pushout of two morphisms $L \to R$ and $L \to G$ is obtained if we complement them to the commutative square which has, moreover, the universal property as shown below:



The existence of pushouts for the pairs $L \xrightarrow{r} R$ and $L \xrightarrow{m} G$ described above in the category of graphs is well known (Rozenberg, 1997). In the category Gr^T we had two more levels. We had to prove that, with these two additional levels, at least weak pushout does exist (the square in the diagram above is called weak pushout, if the property of uniqueness of c is absent).

Theorem 1.2. *Let $L \xrightarrow{r} R$ and $L \xrightarrow{m} G$ be an arbitrary morphism of Gr^T and an embedding respectively. Then there exists a weak pushout of r and m.*

The proof of this theorem is obtained by slight modification of the proof in (Boisvert et al., 2011). The weak pushout can be defined in a canonical way. This is enough to define rewriting systems based on productions of the form $L \stackrel{r}{\rightarrow} R$ in Gr^T as in traditional SPo approach (Rozenberg, 1997).

2. Graph Transformations and Proof Theory

We shall consider in this section several illustrations of interaction between the theory of graph transformations and proof theory.

2.1. Axiom and rule schemas.

Definition 2.1. We shall call a rule-schema a couple that consists of a category S with two objects L, R and a unique morphism $r : L \rightarrow R$ different from identity and a family of functors Φ from S to the category of attributed graphs and attributed graph morphisms. If $\phi \in \Phi$, we shall call the morphism $\phi(r) : \phi(L) \rightarrow \phi(R)$ an instance of the rule schema Φ. The morphism $r : L \rightarrow S$ will be called schematic morphism. By abuse of notation we shall often write $r : L \rightarrow R$ instead of S. We shall call an axiom schema (or initial graph schema) a couple that consists of a category S with one object I and identity morphism, and a family of functors Ψ to the category Gr^T. The graphs $\psi(I)$ will be called initial graphs.

The aim of this definition is to place the notion of GTS more firmly in the categorical context. Of course the main question from the technical point of view will be how the family of functors is defined. Often (but not always) it may be defined by substitution for metavariables, as in the schemas of logical rules.

We had two reasons to use the category S with almost trivial categorical structure in the definition: (a) the object I or the morphism $r : L \rightarrow R$ may have non-trivial syntactic structure (for example, being expressions with metavariables); and (b) this is a particular case of more general definition where S is a category with one *distinguished* object I (or one *distinguished* morphism $r : L \rightarrow R$), and the rest is used to represent constraints imposed on the instances, *e.g.*, similar to application conditions in ordinary graph rewriting (Rozenberg, 1997).

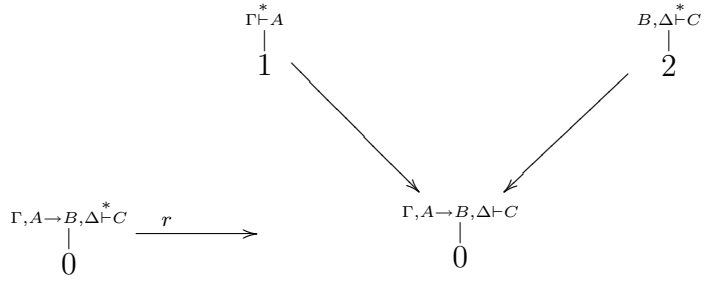
2.2. Generation of deduction trees and deduction graphs.

Using the notion of rule schema, it is not difficult to define a graph grammar that will generate the trees of deductions.

In the example below we consider genereation "backwards", as in proof search. Naturally, to model the deductions in certain system (*e.g.*, propositional calculus), a sligtly extended system will be necessary to work with attributes. For example, in addition to ordinary sequents $\Gamma \vdash A$ one may consider the calculus where each sequent $\Gamma \vdash A$ may be be marked as "open" $\Gamma \stackrel{*}{\vdash} A$. The calculus will include one additional rule schema $\dfrac{\Gamma \stackrel{*}{\vdash} A}{\Gamma \vdash A}$ to "close" the sequents. The graph grammar will include:

- an axiom schema $\stackrel{\Gamma \stackrel{*}{\vdash} A}{\cdot}$ with the functors Ψ defined by substitution of lists of formulas for Γ and single formula for A.

- The rule schemas for graph rewriting corresponding to all deduction rules, *e.g.*

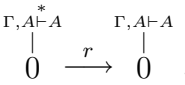


Here $0, 1, 2$ are nodes of the graph, r_{str} is an embedding ("schematism" concerns only attributes), r_{adr} links 0 with 0 (1 and 2 have no antecedents), partial proofs are

$$\frac{\Gamma, A \to B, \Delta \overset{*}{\vdash} C}{\Gamma, A \to B, \Delta \vdash C}$$

("closing" the sequent), and partial proofs without active leaves that attach $\Gamma \overset{*}{\vdash} A$ and $B, \Delta \overset{*}{\vdash} C$ to nodes $1, 2$.

- The rule schema to "close" the nodes with propositional axioms $\Gamma, A \vdash A$:



.

- In all schemas the families of functors to Gr^T will be defined by substitution to metavariables in attributes.

Now, the trees that do not contain open attributes will be exactly the trees representing complete deductions. By adding to this grammar some extra rules (for example, to create links between distant nodes with attributes of particular form) we may derive more complex graphical structures. In certain situations, like consideration of non-strictly positive operators in Type Theory, it is possible to admit even some controlled form of circularity.

2.3. Transformations of derivations.

We shall consider as an example permutation of rules (Kleene, 1952). Let us consider two rules in propositional calculus:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}(\to -right)$$

and

$$\frac{\Gamma_1 \vdash C \quad D, \Gamma_2 \vdash E}{\Gamma_1, C \to D, \Gamma_2 \vdash E}(\to -left).$$

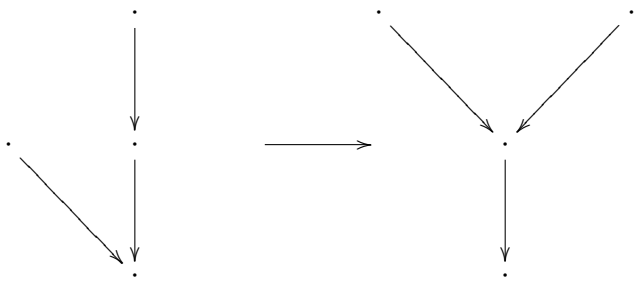
Let us consider first the schema where $(\to -right)$ is applied first. It is to notice that we have to consider the premise of $(\to -right)$ more "finely structured" than in case when each rule schema is taken separately:

$$\frac{\dfrac{}{\Gamma_1 \vdash C} \quad \dfrac{D, \Gamma_2, A \vdash B}{\Gamma_1, D, \Gamma_2 \vdash A \to B}}{\Gamma_1, C \to D, \Gamma_2 \vdash A \to B}.$$

Permutation of two inferences gives:

$$\frac{\dfrac{\Gamma_1 \vdash C \quad D, \Gamma_2, A \vdash B}{\Gamma_1, C \to D, \Gamma_2, A \vdash B}}{\Gamma_1, C \to D, \Gamma_2 \vdash A \to B}.$$

On the level of graph structure (with sequents as attributes) this may be seen as a transformation



2.4. Presupposed judgements.

When we consider formal systems, most significant examples of presupposed judgements appear in Type Theory. Consider, for example, the judgement $\Gamma \vdash t_1 = t_2 : K$. Here as presupposed judgements one may consider $\Gamma \vdash t_1 : K,\ \Gamma \vdash t_2 : K,\ \Gamma \vdash \mathbf{valid}$. If the "main" judgement is derivable, they must be derivable as well. The equality of t_1 and t_2 in type K presupposes that both are of type K, that the context Γ is valid *etc.*, (Soloviev & Luo, 2002).

Notice, that the deduction of a presupposed judgement is not necessarily a sub-deduction of the "main" judgement. Thus, the graph structure that would be natural in treatment of presupposed judgements and their derivations, would rather consist of deduction graphs (maybe, trees, maybe richer structures) of several judgements developed in "parallel", with "horizontal" links indicating correspondences between auxilliary judgements.

3. Feedback to linguistics

Below we outline a few areas where, to our opinion, the approach based on GTS may be useful.

(i) "The language L determines a set of derivations (computations)." (Chomsky, 1995). Chomsky continues his analysis in terms of sets of computations (or derivations), such as convergent, admissible *etc.* Initially these derivations were directly related to *parsing* of the sentences of a language, later the picture became more complicated, admitting many levels (surface structure, deep structure *etc.*). We think that (even if one remains within general paradigm of Chomsky's approach) the inclusion of graph representations and transformations of linguistic derivations (not only textual formal grammars and trees) would provide a richer picture and better understanding of linguistic phenomena. Within the limits of a short paper, we refer to the examples of the previous section as an analogy.

(ii) The approach to linguistics based on Type Theory initiated by Montague (Partee, 1976) and actively developed later (see (Pustejovsky, 1995), (Asher & Pustejovsky, 2006), (Asher, 2008)) is another area where the use of GTS, in this case with type theoretical judgements as attributes, would be interesting. Type Theory provided a method of analysis of the structure and meaning of sentences in natural language based on typing judgements and their derivations. Types corresponded to conceptual categories (with complex types constructed from a set of basic building blocs), λ-terms could represent complete or incomplete sentences, *e.g.* the verb $build : noun \to noun \to sent$, *etc.* (Pustejovsky, 1995). Usually only tree-form derivations were considered. The idea of derivations repesented by graphs more general than trees and their transformations was not yet explored, due to the absence of necessary formalism. Our present work is partly motivated by applications in this domain.

(iii) To give the reader some idea of concrete applications of type theory to linguistics, let us mention a series of works on applications of so called coercive subtyping (Luo & Callaghan, 1999; Luo, 2010, 2011). In the first of these papers, the authors consider semantic aspects of

so called mathematical vernacular, *i.e.* a subset of natural language. Later they extend their approach to the problems that are common in the study of unrestricted natural language. As writes Luo in (Luo, 2010): "modern type theories, together with the theory of coercive subtyping, may offer a powerful language in which interesting lexical phenomena such as logical polysemy and copredication can be properly interpreted".The basic idea of coercive subtyping is, according to Luo, that subtyping is considered as an abbreviation mechanism. A is a (proper) subtype of B ($A < B$) if there is a (unique) coercion c from A to B, and in this case an object a of type A can be used in any context that expects an object of type B (Luo, 2010). Common nouns in type theoretical approach are usually considered as types, and, for instance, the type of men may be considered as a subtype of human beings (corresponding coercion must be declared). In itself, it would be not very interesting, but within the formalism of type theory it may give much more. For instance, subtyping relations propagate through the function types, contravariantly in the first argument: if $A' \leq A$ and $B \leq B'$, then $A \to B \leq A' \leq B'$. For example,

$$[|human|] \to ([|book|] \to Prop) \leq [|man|] \to ([|heavybook|] \to Prop).$$

We may consider some verb, *e.g.*, the verb $[|read|]$ of type $[|human|] \to [|book|] \to Prop$. Using subtyping, the terms representing the sentence like "John reads a heavy book" can be considered as well typed, without any extra operations or conventions. Using coercive subtyping, more complex problems can be treated, for example the problem of disambiguation. Pustejovsky treated it using so called "dot-types". As he wrote "Dot objects have a property that I will refer to as inherent polysemy. This is the ability to appear in selectional contexts that are contradictory in type specification." (Pustejovsky, 1995). A typical example is that "book" can be considered at the same time as a physical and informational object. The formalism of coercive subtyping permits to consider the type $Phy \cdot Info$ as a subtype of each of its components Phy and $Info$, and dot-types can be easily "embedded" into formalism of type theory with coercive subtyping (Luo, 2010).

In the study of coercive subtyping the derivations in dependent type systems and their transformations, for example, the extraction of proofs of presupposed judgements (Soloviev & Luo, 2002), play a very important role. To our opinion, the GTS considered above would be a very useful tool in the study of coercions, including their linguistic applications.

(iv) One may elaborate on linguistic aspects of presupposition. This notion is well known to linguists. Its definition uses the notion of entailment. *E.g.,* in (Karttunen, 1974) we read: "Surface sentence A pragmatically presupposes a logical form L, if and only if it is the case that A can be felicitously uttered only in contexts which entail L." But what exactly does it mean? Does entailment include "co-derivability", like in type theory, or only entailment on the level of sentence itself? If we reformulate the notion of presupposed judgement from type theory to linguistic setting, the meaning would be similar to "if A is derived correctly then B also should be possible to derive correctly". The difference is that in one case (entailment) the relationship is between sentences, and in another between their derivations. GTS are more useful in the study of such relationships than ordinary logical methods.

4. Conclusion

In brief, our main conclusion would be that the passage from textual grammars considered in linguistics to graph grammars and GTS "shifts the center of gravity" to the transformation aspects as complemetary to generation. Indeed, graph structures (and, moreover, graphs with attributes) are much more complex objects than syntactic trees considered in linguistic derivations. These graph structures may reflect syntactic, semantic and pragmatic aspects in linguistics and proof theory. They often represent a sort of "external input" that may be successfully treated using

graph transformation rules. We think that this direction was not yet studied systematically, and that would be interesting to explore using graph transformations framework described above.

Acknowledgments

This research has been partly supported by the Climt project, ANR-11-BS02-016.

References

Asher, N. (2008). A Type Driven Theory of Predication with Complex Types. *Fundamenta Informaticae*, *84*(2), 151–183.

Asher, N., & Pustejovsky, J. (2006). A Type Composition Logic for Generative Lexicon. *J. of Cognitive Science*, *7*, 1-38.

Barendregt, H., van Eekelen, M., Kennaway, J., Plasmeijer, M., & Sleep, M. (1987). Term graph rewriting. In *PARLE Parallel Architectures and Languages Europe, LNCS 249* (p. 164-177). Berlin/Heidelberg: Springer-Verlag.

Boisvert, B., Féraud, L., & Soloviev, S. (2011). Typed lambda-terms in categorical attributed graph rewriting. In *2nd workshop on algebraic methods in model-based software engineering (AMMSE), TOOLS 1-day satellite event* (Vol. 56, p. 33-47). EPTCS.

Bundy, A. (1988). The use of explicit plans to guide inductive proofs. In E. Luck & R. Overbeek (Eds.), *Proceedings of the 9th International Conference on Automated Deduction (CADE), LNCS 310* (p. 111-120). Argonne: Springer-Verlag.

Chomsky, N. (1995). *The Minimalist Program*. Cambridge, MA: M.I.T. Press.

Karttunen, L. (1974). Presupposition and linguistic context. *Theoretical Linguistics*, *1*, 181-194.

Kleene, S. C. (1952). Permutability of inferences in Gentzen's calculi LK and LJ. *Mem. Amer. Math. Soc*, *10*, 1-26.

Luo, Z. (2010). Type-theoretical semantics with coercive subtyping. *Semantics and Linguistic Theory*, *20*, 38-56.

Luo, Z. (2011). Contextual analysis of word meanings in type-theoretical semantics. In J.-P. Pogodalla S. Prost (Ed.), *Logical Aspects of Computational Linguistics (LACL'2011), LNAI 6736* (p. 159-174). Berlin: Springer.

Luo, Z., & Callaghan, P. (1999). Mathematical vernacular and conceptual well-formedness in mathematical language. In F. Lamarche, A. Lecomte, & G. Perrier (Eds.), *Logical Aspects of Computational Linguistics, Proceedings of the Second International Conference (LACL'97), LNAI/LNCS 1582* (p. 231-250). Berlin: Springer-Verlag.

Partee, B. (1976). *Montague grammar*. New York, NY: Academic Press.

Pustejovsky, J. (1995). *The Generative Lexicon*. Cambridge, MA: M.I.T. Press.

Ranta, A. (1994). *Type-theoretical grammar*. Oxford, UK: Oxford Science Publications.

Rozenberg, G. (1997). *Handbook of graph grammars and computing by graph transformations* (Vol. 1). World Scientific.

Soloviev, S., & Luo, Z. (2002). Coercion completion and concervativity in coercive subtyping. *Annals of Pure and Applied Logic*, *113*(1-3), 297-322.