Internship Report, Magistere Informatique et Telecommunication ENS Cachan and University of Rennes 1.
Cluster of Excellence MULTIMODAL COMPUTING AND INTERACTION
Research group EFFICIENT ALGORITHMS IN COMPUTATIONAL LINGUISTICS
Supervised by Alexander Koller, Head of the research group
September 10 2010

Scope Ambiguity: Getting closer to weakest readings using Rewrite Sytems Completion

Antoine Venant

Abstract

We apply Knuth Bendix completion to the rewrite system used in Koller and Thater's method [KT10] for computing weakest-readings of sentences with scope-ambiguity. To do this we design an adaption of this procedure which doesn't break the soundness of the rewrite systems w.r.t logical entailment and logical equivalence, but makes them approximates better these two relations. We presents important optimizations relative to dominance graphs allowing a working implementation. We expose results obtained after implementation and shows that there is a trade-off between the gain in precision with a deeper completion and the increase of computation time.

Saarbrücken
Internship dates: Mai 25 - August 12 2010

.

Rapport de stage, Magistere Informatique et Telecommunication ENS Cachan et Université de Rennes 1.
Cluster d'Excellence MULTIMODAL COMPUTING AND INTERACTION
Research group EFFICIENT ALGORITHMS IN COMPUTATIONAL LINGUISTICS
stage supervisé par: Alexander Koller, Head of the resarch group
10 septembre 2010

Completion d'un système de réécriture pour un calcul plus precis des sens les plus faibles de phrases ambigues

Antoine Venant

Abstract

Nous appliquons la procédure de complétion de Knuth-Bendix au système de réécriture employé par Koller et Thater [KT10], au sein de leur méthode de calcul des sens les plus faibles des phrases rendues ambigues par l'imbrication de quantifieurs. Pour ce faire, nous avons mis au point une adaptation de la procédure à notre cas qui respecte la compatibilité des systèmes de réécriture de termes avec la conséquence et l'équivalence logique, tout en leur permettant d'approximer mieux ces deux relations. Nous présentons des optimisations de cette procédure basées sur des considérations relatives aux graphes de dominance et permettant une implémentation exploitable. Nous détaillons les résultats obtenus après implétation et montrons qu'on doit négocier un compromis entre le gain de précision apporté par une complétion poussée et l'augmentation conséquente du temps de calcul

Saarbrücken
dates de stage: Mai 25 - August 12 2010

Contents

1 Introduction **5**

2 Preliminaries **5**
2.1 Terms 6
2.2 Underspecification 8
2.3 Term Rewrite System 8

3 Completing rewrite systems **11**
3.1 Aim of completion 11
3.2 Adapting the basic completion procedure 12

4 Optimizations relative to dominance graphs **15**
4.1 Filtering rewrite by existence of a common accepting dominance graph 15
4.2 Fully relative completion 16

5 Implementation results **17**

6 Conclusion **18**

7 Bibliography **18**

8 Annex **19**

1 Introduction

Computing a representation of the meaning of a given natural-language sentence that can then be used in some application, like testing if a sentence is a consequence of another, is one of the most important goal that computational linguistics attempts to achieve. For instance, representing the meaning of a sentence into some logic, like predicate logic, would allow to model textual entailment at a logical level.

However this approach is limited by the ambiguity of some sentences, which makes it difficult to choose just one formula (or reading) among the whole set of readings that a sentence can have (which is in the worst case exponential in the length of the sentence, and is likely to count over million of readings for some ambiguous sentences, like those of the Rondane Tree-bank). And even if large-scale grammars offers today to derive underspecified representations of a sentence (*i.e* compact representation of the set of readings it can have) it remains difficult to chose some particular readings. Ideally, the chosen meaning should be the one intended by the speaker, with respect to the context in which the sentence occurs. But there is today no sufficient way to find this particular meaning *i.e.* to disambiguate the sentence in its context. Another approach consists of trying to compute the *weakest readings* of the sentence, *i.e* those one that only retain informations shared by all other readings. Doing this keeps only safe information, and can be used for the particular purpose of deciding textual entailment, because if a logical formula is a consequence of all weakest readings of a sentence, it is then a consequence of all readings of the sentence.

To solve the problem of computing these weakest readings, Koller and Thater [KT10] have proposed a method that allows to compute these weakest readings as a regular tree language, without comparing all readings of the considered sentence w.r.t logical entailment. This algorithm is indeed based on a approximation of logical entailment by a *sound* rewrite system which normal-forms computed relatively to a regular tree language are considered as weakest readings. This is an approximation to the extent that all weakest readings are actually relative normal forms of this system, but extra non-weakest-readings normal-forms can also occur. Moreover, this system is combined with a redundancy-elimination one, which aims at reducing the set of computed readings using logical equivalence: over two logically equivalent readings, only one should be kept. Both systems are instances of the same general algorithm, so redundancy elimination is also based on approximating a relation over logical formulas using a rewrite system, but this time it approximates logical equivalence. In both cases, systems are correct but not complete, and the set of readings still can be refined. One can think that completing the rewrite systems using a completion procedure such as the Knuth Bendix[KB70] one would reduce the sets of normal forms and thus the computed readings. But this would also add many rules of higher depth, and then could slow the computation. The aim of the internship was to try to complete this two rewrite-system in some way, and then to give a concrete vision of the trade-off that we expect between reducing the numbers of readings and increasing the cost of the computation.

In the first section we will briefly introduce some needed basic notions around terms and term rewrite systems, precise the kind of rewrite system that we are dealing with and its relation to the computed readings. The second section exposes the adaptation of Knuth-Bendix completion we design for our purpose, and explains what can be expected of it. Then we will focus on two way of using dominance graph to optimize completion and keep things manageable. The last section presents our concrete implementation and results and shows that our work confirm the expecting trade off and can be used to concretely quantify it.

2 Preliminaries

Let's first see an example of ambiguity. Look at the following sentence:

Example 1. .Every students reads a book.

Two quantifiers are present in this sentence, the first one is **every**, the second one is **a**.The sentence is semantically ambiguous because we don't know if the students are reading the same book or not. In term of logic, it depends on the way the two quantifiers are interleaved: looking at the formula $\forall_x(student_x \implies \exists_y book_y \wedge reads_{x,y})$ the books can be different or not. Looking at $\exists_y book_y \wedge \forall_x(student_x \implies reads_{x,y})$ they are reading the same book. But the first formula is a logical consequence of the second one, and is therefore less informative. Indeed the first one is a *weakest-readings* of this sentence but not the second one.

First of all, let's briefly describe the kind of meaning representations we aim at producing for a sentence: These are first-order like formulas of a logic adapted to natural language sentence representation. Below are the few things that it is good to know about it to fix ideas:

Used Logic The logic used for representation of meaning is not strictly-speaking first-order predicate logic, but generalized-quantifier logic, following [BC81]. This is justified by the lack of expressivity of the universal and existential quantifiers, which are known to be insufficient to express some quantified proposition of natural-language like "most of the q". This logic contains an extended set of quantifiers adapted to natural-language parsing. *most of the*, *the*, *two* and even proper nouns are considered as quantifiers. Two assumptions are made:

1. Quantifiers of arity two are sufficient to express all sentences of natural language.
2. For each sentence, there exists a context that allows to know the exact semantic of vague quantifiers like *most of the q*.

Indeed we don't really care here about interpreting those extended quantifiers, neither Koller and Thater's method nor this work assume anything on these interpretations. All constructions remain correct whatever the semantic of vague quantifiers is.

2.1 Terms

We start by giving a formal definition of *Terms*; which are the kind of objects constituting the semantic representations, as (abreviated[1]) first-order predicate logic formulas.

Definition 1 (Ranked Alphabet)**.** A ranked alphabet is a couple $\Sigma = (R, arity)$, where R is a set of symbol and $arity$ is a function that assigns an integer at each symbol in R, $arity : R \mapsto \mathbb{N}$. $arity(r)$ is called the *arity* of r.

Symbols of arity 0 are called *constants*.

Definition 2 (Terms over a ranked alphabet)**.** Let χ be a denombrable set of variable symbols. The set $T(\Sigma, \chi)$ of terms over a ranked alphabet $\Sigma = (R, arity)$ is inductively defined by:

- $\Sigma \cup \chi \in T(\Sigma, \chi)$.
- for all $r \in R$, for all $(T_1, \ldots, T_{arity(r)}) \in T(\Sigma, \chi)^{arity(r)}$, $r(T_1, \ldots, T_{arity(r)}) \in T(\Sigma, \chi)$.

$Vars(t)$ denote the set of all variable symbols of χ occurring in t. these are used to define substitutions:

Definition 3 (Substitution)**.** A substitution σ is a mapping from χ to $T(\Sigma, \chi)$ where only a finite set of variables are not mapped to themselves. A substitution can be seen as a mapping from $T(\Sigma, \chi)$ to $T(\Sigma, \chi)$ applying it recursively to a term: $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$. The set of all substitutions over $T(\Sigma, \chi)$ is called $Sub(T(\Sigma, \chi))$.

[1] for instance $\forall_x(P, Q)$ stands for $\forall x,\ P \implies Q$

If for some term C $Vars(C) = \{x\}$, C is a *context* over Σ. Given a term t $C[t]$ denote $\sigma(C)$ where $\sigma = x \leftarrow t$ is the substitution that maps x to t and leaves all other variables unchanged. $\mathcal{C}(\Sigma)$ denote the set of all contexts over Σ.

In the following, we will also need the definition of a labeled tree (following the definition of the *TATA* book[CDG+07]):

Definition 4 (Labeled tree over a ranked alphabet)**.** A (ranked) labeled tree t over the ranked alphabet $\Sigma \cup \chi$, where $\Sigma = (R, arity)$ is a ranked alphabet and χ is a set of variable symbols, is a couple $(Pos(t), F)$ verifying the following:

1. $Pos(t)$ is a non-empty prefix-closed subset of $\mathbb{N}^*$
2. F is a function from $Pos(t)$ into R.
3. $$\begin{array}{lll} \forall p \in Pos(t), \text{ if } F(t) \in R \wedge arity(F(t)) \geq 1 & \text{then} & \{j \mid p \cdot j \in Pos(t)\} = [\![1, arity(F(t))]\!] \\ & \text{else} & \{j \mid p \cdot j \in Pos(t)\} = \emptyset \end{array}$$

.

(A^* denote the set of strings over A and $\cdot$ the strings concatenation).
Each index in $Pos(t)$ is called a position.

Trees can be also be seen as finite connected oriented acyclic graph (and conversely), seeing $Pos(t)$ as the set of vertices, with an edge from p to p' *iff* $\exists i \in \mathbb{N}$, $p' = p.i$.

By $t|_p = (Pos(t|_p), F|_p)$ we denote the subtree at position p of a tree $(Pos(t), F)$, defined as follow:

$$\begin{array}{rcl} Pos(t|_p) & = & \{q \in \mathbb{N}^* \mid p \cdot q \in Pos(t)\} \\ F|_p(q) & = & F(p \cdot q) \end{array}$$

$t[u]_p$ is the tree obtained by replacing the subtree at position p in t with u.

Every Term $t \in T(\Sigma, \chi)$ can be seen as a tree using the following inductive transformation:

- if $t = x$ where x is a variable or a constant, t can be seen as the tree $(\{\epsilon\}, f : \epsilon \mapsto x)$
- if $t = f(t_1, \ldots, t_n)$ where f is a symbol of R of arity n, and the terms $t_1, \ldots t_n$ are respectively represented by the trees $(Pos(t_1), F_1), \ldots (Pos(t_n), F_n)$, t can be seen as the tree $((\{\epsilon\} \cup \{1 \cdot Pos(t_1), \ldots, n \cdot Pos(t_n)\}), F)$ where F is defined as follows:

$$\begin{array}{rcl} F(\epsilon) & = & f \\ F(1 \cdot p) & = & F_1(p) \\ & \vdots & \\ F(n \cdot p) & = & F_n(p) \end{array}$$

In the following we will assimile a term and its representation as tree. We will for instance refer to $F(p)$ as "the label at position p in the term t".

2.2 Underspecification

One way to face the problem of ambiguous sentences is to compute and work with an *underspecified* representation of the meaning, instead of deriving each single reading directly from the syntactic analysis. Single readings can be enumerated from this *underspecification*: they are not stored explicitly, but in a compact way, and a computation is needed for the enumeration. There exists several underspecification formalisms but we will restrict ourselves here to the one of *dominance graphs*[ADK+03] which is the one used by Koller and Thater as input for their algorithm.

Before giving the formal definition, we can draw out the underlying general idea: a dominance graph is used to impose restrictions on the way to plug together the different logical quantifiers that appear in a given sentence. More specifically, it imposes *dominance* constraints on these quantifiers. If for instance the second child of A is precised to domine B, then a meaning for the sentence can only be constructed by plugging quantifiers in a such way that B is a descendant of the second child of A. The formal definition of a dominance graph and its configurations is given below and figure 2.2 presents a dominance graph representing the two possibles readings of the sentence of example 1.

Definition 5 (Dominance Graph)**.** A dominance graph labeled over $\Sigma = (R, arity)$ is a tuple $(V, E, D, L, <)$ where V is a set of nodes, E and V are two sets of edge of $V \times V$, respectively called tree- and dominance-edge, L is a labeling function $L : V \mapsto R$, $<$ a strict linear order over V and the following conditions hold:

1. (V, E) is a graph which connected-components are all trees of hight 0 or 1. The roots of these trees are called *roots*. Every node in V which is not a root is called a *hole*
2. $\forall (h, r) \in D\, h \in holes$ and $r \in roots$.
3. every hole has at least one outgoing dominance edge in D.
4. L is a partial node-labeling function mapping each root to a symbol in R.

One can extract a Ranked signature W_g from a dominance graph by considering each root r in V as a tree-constructor of arity the number of holes h such that $(r, h) \in E$ and ordering this holes w.r.t $<$. A dominance graph describes a set of trees called configurations of the graph, in the following way:

Let's first define an *unlabeled* configuration of a dominance graph.

Definition 6 (Unlabeled configuration)**.** A tree t over W_g is a configuration of the dominance graph $d = (V, E, D, L, <)$ iff each symbol of W_g occurs exactly once in t and each dominance edge of d is realized in t, which means that if $(h_i, r') \in D$ where the hole h_i is the i^{th}(w.r.t $<$) child of a root r in d, then there is a path from the i^{st} child of the node r to r' in t.

Definition 7 (Labeled configuration)**.** A labeled configuration of a dominance graph $d = (V, E, D, L, <)$ is a tree t over Σ such that there exists an unlabeled configuration u of d verifying $L(u) = t$.

The main idea of computing weakest-readings of a dominance graph following [KT10] is to approximate logical entailment using a term rewrite system. This is the reason of the next paragraph in which we will detail what a term rewrite-system is and how it can be used to approximate entailment and equivalence.

2.3 Term Rewrite System

The ground reason for the ambiguity we are trying to deal with here, is the many different possibilities of plugging the different quantifiers together that a dominance graph can offer. Sometime

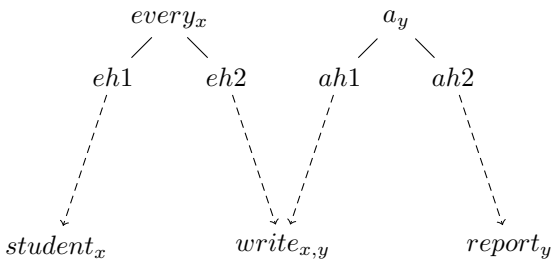

Figure 1: a dominance graph

it is easy to see that a labeled configuration of a graph is a logical consequence of another, because the permutations of two quantifiers under some logical polarity makes the global formula satisfied by more models. To understand this, look at the following example:

Example 2. $\forall_x(P_x, \exists_y(Q_y, R_{x,y}))$ which in our logical formalism stands for $\forall x\, P(x) \Rightarrow \exists y\, Q(y) \wedge R(x,y)$ in predicate logic. Permuting the existential quantifier with the universal one would make the formula satisfied by more models: $\exists_y(Q_y, \forall_x(P_x, R_{x,y}))$ ($\exists y\, Q(y) \wedge \forall x\, P(x) \Rightarrow R(x,y)$ in predicate logic).

Indeed the transformation $\forall_x(P_x, \exists_y(Q_y, R_{x,y})) \rightarrow \exists_y(Q_y, \forall_x(P_x, R_{x,y}))$ can be applied at any position of a formula which is assigned a positive logical polarity: a position p in a formula has a positive polarity if making the sub-formula at position p weaker[2] makes the global formula weaker. Conversely, a position p is under negative polarity if making the sub-formula at position p weaker makes the global formula stronger. If we restrict to predicate logic, all position are either in positive or negative polarity, and each position can be assigned the corresponding polarity by starting with a positive polarity at the root of the formula and then running top-down over the formula reverting the polarity each time a negation is encountered.

With our logical formalism, certain quantifiers have *non-monotonous* children positions, which means that even if they are in positive or negative polarity, their children haven't any polarity. This is addressed by adding an annotation *neutral* to the set $\{+, -\}$ which is assigned to these non-monotonous positions.

Thus, the idea for computing weakest readings is to group all this weakening transformations and the corresponding annotations into an annotated Term Rewrite System. Let's formally define that kind of object:

Definition 8 (Annotated Term Rewrite System)**.** Let A be a finite set of annotations. An *annotated rewrite rule* is an identity $[a]l \rightarrow r$ where a is an annotation of A, l and r are Terms of $T(\Sigma, \chi)$ such that $Vars(l) \subseteq Vars(r)$. An annotated Term Rewrite System is a set of annotated rewrite rules.

A Term Rewrite System induces a rewrite relation $\rightarrow_R \subseteq T(\Sigma, \chi) \times T(\Sigma, \chi)$:

Before defining this, we first need to be able to compute annotations for every nodes of a tree. To do this, we use an *Annotator*:

Definition 9 (Annotator)**.** an Annotator is a function that given an annotation and a ranked symbol of Σ and an annotation returns the annotation assigned to each children of this symbol.

Given a starting annotation, assigned to the root of all terms, such a function allows to assign an annotation to each node of a term doing a top-down traversal of it. In the following we assume a starting annotation and the annotator to be given, and refer to them as, respectively, a_0 and Ann.

We also surcharge the symbol Ann and denote by $Ann(a_0, t, p)$ the annotation assigned by Ann to the position p of t after annotating t running Ann top-down along it with a_0 as starting annotation.

[2]satisfied by more models, ϕ weaker than ψ iff $M \vdash \psi$ implies $M \vdash \phi$

Definition 10 (Rewrite relation)**.** let R be a Term Rewrite System, $t, t' \in T(\Sigma, \chi)$.
The rewrite relation $\rightarrow$ is defined as follow:

$$t \rightarrow_R t' \text{ iff}$$
$$\exists([a]l \rightarrow r) \in R,\ \exists C \in \mathcal{C}(\Sigma), \exists \sigma \in Sub(T(\Sigma;\chi)) \text{ s.t}$$
$$Ann(a_0, t, p) = a \wedge t = C[\sigma(l)] \wedge t' = C[\sigma(r)].$$

$\rightarrow_R^*$ and $\leftrightarrow_R$ respectively denote the reflexive-transitive and the symmetric closure of this relation. We write $[a]t \leftarrow t'$ when we want a to be taken as starting annotation in t instead of a_0.

A rewrite system R is *sound* w.r.t a binary relation $\sim$ over $T(\Sigma, \chi)$ iff $t \rightarrow_R u$ implies $(t \sim u)$. We denote the logical entailment by $\succ$: $f \succ g$ iff (f entails $g \mid g$ is weaker than f.

Now, assume we have a sound (with respect to $\succ$, the logical entailment) rewrite system W containing all weakening rules like the one of example 2. We want to use this rewrite system to compute weakest readings of a dominance graph d, approximating logical entailment over its configuration by $\rightarrow_W$. According to this definition of weakest-readings, the set of terms we aim at computing is the set containing every configurations of d such that there exists no other smaller, according to $\rightarrow_R$, configuration, *i.e* the set of configurations of d that can't be rewritten into another configuration of d, and these are what we call *relative normal forms*:

Definition 11 (Normal Forms and Relative Normal Forms)**.** A term t is a *normal form* of a rewrite system R is there is no other term u such that $t \rightarrow_R u$. Given a dominance graph d, a term t is a *relative normal form* of R w.r.t d if

$$\begin{cases} t \in Conf(d) \\ \neg\exists u \in Conf(d)\ t \rightarrow_R u \end{cases}$$

To reduce redundancy using logical equivalence an additional "equivalence" rewrite system E is added to W. This one is sound with respect to logical equivalence and its relative normal forms are the selected representant of equivalence classes that are not deleted (but there can be more than one representant for a single class, since the rewrite-system approximation is not complete).

We also define confluence of a term rewrite system because this notion will be of great use in the following:

Two terms t and t' are *joinable* w.r.t R (We write $t \downarrow_R t'$) iff

$$\exists z \in T(\Sigma, \chi)\ \begin{matrix} t \rightarrow_R^* z \\ t' \rightarrow_R^* z \end{matrix}$$

Definition 12 (Confluence)**.** A term rewrite system R is confluent iff

$$\forall t, u_1, u_2 \in T(\Sigma, \chi),\ \begin{matrix} t \rightarrow_R^* u_1 \\ t \rightarrow_R^* u_2 \end{matrix} \implies u_1 \downarrow_R u_2$$

[KT10] proposes a general method for computing relative normal forms over a regular tree-language which rely on intersecting tree-automata. Since dominance-graph can be efficiently converted into tree-automata([KRT08]), computing weakest-readings is a sub case of this general purpose. the detail of this method are not needed to understand the work presented here, because we focus on the completion itself, and the method is leaved unchanged after performing completion.

A last things to know, is that our rewrite system is a *permutation rewrite system*, which means that it only changes position of nodes, without changing labels, or introducing or removing nodes. Example 2 illustrates this.

Now that we know about term rewriting, relative-normal forms, and how they are related to the set of weakest readings we aim at computing, we can interest ourselves to the completion of the two(equivalence and rewriting) rewrite systems. Before trying to complete the rewrite systems in any way, one should try to fix in what could be expected from completion and how it could be achieved. Hence, the next section presents the expectations we find out to be reasonable, and the completion procedures we design to try to achieve it.

3 Completing rewrite systems

In the following, for every binary relation $\bowtie$ and dominance graph d, $\bowtie|_d$ denote the restriction of R to the configurations of d, *i.e.* $x \bowtie|_d y$ iff $x \in Conf(d) \wedge y \in Conf(d) \wedge x \bowtie y$. Every binary relation over a set A is also seen as a part of $A \times A$.

3.1 Aim of completion

The Knuth-Bendix completion procedure initially aims at building a convergent term rewrite system that allows to decide the word problem for a given set of identities $S = \{(l \to r)\} \subseteq T(\Sigma, chi)^2$. The reflexive, transitive and symmetric closure of the set of the relation $\to_S$ defines an equivalence relation $\leftrightarrow_E^*$ and the completion procedure attempts to build a convergent rewrite system R such that $t \sim u$ iff $t \downarrow u$. Although this does not particularly seems to fit our use of the weakening rewrite system, it is perfectly adapted to the equivalence rewrite system which we can think in term of a set of identities (since logical equivalence is symmetric). Thus, if we could apply successfully the Knuth-Bendix completion procedure to it, we would end up with a convergent term rewrite system reducing each term into a single normal form-representant of its equivalence class. This would probably reduce the set of computed normal forms in a fair manner. However the impact of completing the rewrite relation $\to_E$ on its restriction $\to_E|_d$ to the configurations of a dominance graph d is not clear.

Can we also complete the rewrite system in some way? This seems more difficult. Knuth Bendix completion procedure assumes an underlying equivalence relation $\sim$ containing $\to_R$. if it is the case and a term t can be rewritten into two different terms u_1 and u_2 with R, then $t \sim u_1$ and $t \sim u_2$, hence $u_1 \sim u_2$ and a rule $u_1 \to u_2$ (for instance) can safely be added to R without breaking its inclusion in $\sim$. The weakening rewrite system has no underlying equivalence relation, but an underlying strict order $\succ$. And if

$$\begin{array}{lll} t & \to_W & u_1 \\ t & \to_W & u_2 \end{array}$$

we have $t \succ u_1$, $t \succ u_2$. But since neither $u_1 \succ u_2$ nor $u_2 \succ u_1$ can be deduced (without any supplement of information) from these hypotheses, adding a rule rewriting one of the u_i into the other may break the soundness of the system and prevent us from computing weakest-readings. Despite this, it turns out that the weakening-rewrite system can be completed in some way: when a term can either be rewritten into an equivalent one u, using E, or a weaker one v, using W, we can deduce $u \succ v$ and thus add a rule $v \to_W u$.

We are now able to define the "best hope" we can have in completing the rewrite systems. Formally speaking, without completion, the computed readings of a dominance graph d are the normal forms of the rewrite relation $(\to_W|_d \cup \to_E|_d)$. Using completion we can hope to compute an equivalence rewrite-system E_c such that $t \leftrightarrow_E u$ iff $t \downarrow_E u$. We also would like to add all possible weakening rules of the kind we presented before *i.e* we would like to fully use the information that $\succ$ is the strict part of $\succeq$, and then that every chain like $t \sim t_1 \cdots \sim t_n \succ t'$ implies $t \succ t'$. The rewrite relation $\to_W$ approximates $\succ$, $\leftrightarrow_E$ approximates logical entailment. Thus the relation $\sqsubseteq = (\to_W \cup \leftrightarrow_E)^*$ approximates $\succeq$ (is logically equivalent or entails). This

relation is built to takes into into account every entailment that can be deduced from $\rightarrow_W$ and $\leftrightarrow_E$. The strict part of this relation w.r.t $\leftrightarrow_E$, *i.e* $\sqsubset = \sqsubseteq \setminus \leftrightarrow_E$ should be a better approximation of $\succ$ than $\rightarrow_W$. It would be nice to compute a completed rewrite-system which normal forms are the minimal elements of $\sqsubset$.

Given such a rewrite system, C and $\rightarrow_C$ its rewrite relation, it is sadly not necessary that given a dominance graph d, the normal-forms of $\rightarrow_C|_d$ are the minimal elements of $\sqsubset|_d$. For the same reason and as already mentioned above, even if $\rightarrow_{E_c}$ is confluent, a term t can have more than one relative normal forms w.r.t to E_c. We will precise this in the following after briefly recalling the basic completion procedure and expose its adapatation to our case.

3.2 Adapting the basic completion procedure

The basic completion procedure(as presented in [BN99]), as well as more sofisticated completion procedures[3], relies on the notion of *critical pair*, which are at the source of non-confluent derivations. The notion of critical pair is legitimated by the following remark: if $t \rightarrow_R t_1$ using a rule $l_1 \rightarrow r_1$ and $t \rightarrow_R t_2$ using $l_2 \rightarrow r_2$, then either $t_1 \downarrow t_2$ or l_1 and l_2 have to *overlap*, which means that l_2 occurs at non-variable position of l_1 (or symetricaly l_1 in l_2). More details and a proof of this can be found in [BN99]. We extend here this notion for annotated rewrite systems. It is easy to see that the assertion above still holds with this definition of critical pair, the only change is that we impose the annotation of the second rule to be coherent with the annotation of the first one, which is a necessary condition for both rules to remain applicable when they overlap:

Definition 13 (Critical pair)**.** An annotated couple of terms of $T(\Sigma, \chi)$, $[a](t_1, t_2)$ is a critical pair of a rewrite system R iff

$$\exists([a_1]l_1 \rightarrow r_1) \in R, \exists([a_2]l_2 \rightarrow r_2) \in R, \sigma \in Sub(T(\Sigma,\chi)), \exists p \in Pos(l_1) \text{ s.t}$$
$$\sigma(l_1)|_p = l_2 \wedge Ann(a_1, l_1, p) = a_2 \wedge t_1 = \sigma(r_1) \wedge t_2 = l_1[r_2]_p.$$

We denote by $Crit(R)$ the set of all critical pairs of R.

The following lemma is the key interest of critical pairs:

Lemma 1 (Critical Pair lemma)**.** An [annotated] rewrite system R is confluent iff $\forall [a](t_1, t_2)$, $[a]t_1 \downarrow_R [a]t_2$.

The basic completion procedure, assumes as input a given order $>$, which is used to orient new generated rules. Given an arbitrary order over quantifiers (like string comparison), we oriented rules by recursively top-down comparing quantifiers symbols.

Basic completion procedure with annotations

Require: $>$ a well founded order over $T(\Sigma, \chi)$, a set of annotated identities $S \subseteq A \times T(\Sigma, \chi) \times T(\Sigma, \chi)$.
Ensure: $\forall([a]l \rightarrow r) \in S, l > r$.
  $R_0 = R$
  $i = 0$
  while $Crit(R_i) \neq \emptyset$ **do**
    $R_{i+1} = R_i$
    for $[a](t_1, t_2) \in Crit(R_i)$ **do**
      Reduce t_1 and t_2 to some R_{i+1} normal-form: $((\hat{t}_1, \hat{t}_2))$.
      if $\hat{t}_1 \neq \hat{t}_2$ **then**
        if $\hat{t}_1 > \hat{t}_2$ **then**
          $R_{i+1} = R_i \cup \{[a]\hat{t}_1 \rightarrow \hat{t}_2\}$

[3] Like Huet's completion precedure, allowing for rules simplification

else if $\hat{t}_1 < \hat{t}_2$ **then**
$R_{i+1} = R_i \cup \{[a]\hat{t}_2 \to \hat{t}_1\}$
else
FAIL
end if
end if
end for
$i = i + 1$
end while
Output R_i

There are three possible way for this procedure to behave:

- It can successfully terminates and outputs a confluent rewrite system R where all critical pairs are joinable.
- It can fail because encountering an orientable pair of term using $>$
- It can run forever generating infinitely many new critical pairs.

Anyway, according to our preliminary analysis in paragraph 3.1 we should be able to apply this procedure to the equivalence rewrite system.

We previously examinated the possibility of completing the weakening rewrite-system when a term can either be rewritten into a logically equivalent term or a weaker one. There are two possible cases for this: either this two rewrites are an instance of a critical pair outstanding from the overlap between a weakening and a equivalence rule (or conversely), or it is always possible to apply the two rules separately one after the other in any order with the same result. In the latter case, there is no need to add a new rule. We propose the following adapted completion procedure to complete the weakening system. $Crit(R_1, R_2)$ denote the sets of critical pair obtained by unifying a rule of R_2 with the subtree at some position of a rule of R_1.

Weakening rewrite system completion

Require: A rewrite system W sound w.r.t a strict order $>$. A rewrite system E.
$W_0 = W$
$i = 0$
while $Crit(W, E) \cup Crit(E, W) \neq \emptyset$ **do**
$W_{i+1} = W_i$
for $[a](u_1, u_2) \in Crit(W, E) \cup Crit(E, W)$ **do**
if $(u_1, u_2) \in Crit(W, E)$ **then**
$(t_1, t_2) = (u_1, u_2)$
else if $(u_1, u_2) \in Crit(E, W)$ **then**
$(t_1, t_2) = (u_2, u_1)$
end if
Reduce t_1 to some W_{i+1} normal form $\hat{t}_1$, reduce t_2 to some W_{i+1} normal form $\hat{t}_2$.
if $\hat{t}_1 \neq \hat{t}_2$ **then**
Reduce t_2 to some E normal form $\tilde{t}_2$.
$W_{i+1} = W_{i+1} \cup \{[a]\tilde{t}_2 \to \hat{t}_1\}$
end if
end for
$i = i + 1$
end while
Output W_i

If the input equivalence system E is confluent, then, if this procedure terminates, the computed rewrite system W_C is such that the normal forms of $W_C \cup E$ are the normal forms of E which are minimal elements of the $\sqsubset$, the strict part of $(\to_W \cup \leftrightarrow_E)^*$.

Proof: Notice first that, at any completion step i of the procedure above, $\rightarrow_{W_i} \subseteq \sqsubset$. This is trivially true for $\rightarrow_W$ and remains true each time a rule $[a]\tilde{t}_2 \rightarrow \hat{t}_1$ is added during completion: $t_2 \rightarrow_E^* \tilde{t}_2$ thus $\tilde{t}_2 \leftrightarrow t_2 \rightarrow_W^* \hat{t}_1$ and it follows $\tilde{t}_2 \sqsubset \hat{t}_1$. This will help us at proving that a term t which is a minimal element of $\sqsubset$ and a normal form of E, is a normal form of $W_c \cup E$: Assume t a minimal element of $\sqsubset$ but not a normal form of $W_c \cup E$ then either $t \rightarrow_{W_c} t'$ and thus $t \sqsubset t'$ which is a contradiction, or $t \rightarrow_E t'$ and thus t would not be a normal form of E.

Conversely assume that t is a normal form of $\rightarrow_{W_c} \cup E$. Assume that $t \sqsubset t'$ by definition of $\sqsubset$ t' is obtained from t after a chain of derivation $t\,\mathrm{rel}_1\,t_1\,\mathrm{rel}_2\,t_2 \ldots \mathrm{rel}_n\,t_n$ where $\forall i \in [\![1; n]\!]$, $\mathrm{rel}_n \in \{\rightarrow_{W_c}, \leftrightarrow_E\}$ and there is at least one i such that $r_i = \rightarrow_{W_c}$. Moreover, $\mathrm{rel}_1 = \leftrightarrow_E$ or t would not be a W_c normal form. Then consider the longest sub-chain of $\leftrightarrow_E$ starting with rel_1 before encountering a $\rightarrow_{W_c}$: $t \leftrightarrow_E t_1 \ldots t_{k-1} \leftrightarrow_E t_k \rightarrow_{W_c} u$. Using the confluence of E we have $t \downarrow_E t_k$, and since t is a E normal form, $t_k \rightarrow_E^* t$. Hence either (t_k, u) is an instance of a critical pair and a weakening rule with left-hand side a subtree of t should have been added and t could be reduced. Otherwise t_{k-1} can also be weakened using the same rule as t_k and the same reasoning can be applied to the strictly smaller chain $t \leftrightarrow_E t_1 \ldots t_{k-1} \rightarrow_{W_c} u'$.

Now that we have detailed the completion procedures, we can more easily illustrate the problem that can occur when restricting the rewrite relation of a completed rewrite system to the configurations of a dominance graph d. Look at the following derivation scheme:

$$\begin{array}{lcl} t \in Conf(d) & \rightarrow & t_1 \in Conf(d) \cdots \rightarrow \hat{t}_1 \notin Cond(d) \\ t \in Conf(d) & \rightarrow & t_1' \in Conf(d) \rightarrow \cdots \rightarrow \hat{t}' \end{array}$$

Where (t_1, t_1') is an instance of a critical pair, d a dominance graph and $\hat{t}_1$, $\hat{t}_2$ are normal forms. If a derivation like the one above happens, the basic completion will had a rule with $\hat{t}_1$ as left-hand side. Even if the completion procedure terminates and output a confluent rewrite system, the restriction to the configurations of d of this rewrite system will not be confluent, because t will not be rewritten into $\hat{t}_1$ but in another relative normal-form which is not a non-relative normal form and the added rule will never be applied.

We can even give a concrete example of this situation in the context of our permutation rewrite system:

Example 3. Consider tree symbols of arity two $A, B, C.$ with the following rules (applicable whatever the annotation):

$$\begin{array}{lcll} A(P, B(Q, R)) & \rightarrow & B(A(P, Q), R) & (1) \\ B(P, C(Q, R)) & \rightarrow & C(B(P, Q), R) & (2) \\ B(P, C(Q, R)) & \rightarrow & C(Q, B(P, R)) & (3) \end{array}$$

(1) and (3) overlap in $t = A(P, B(Q, C(R, S)))$ which yield the following critical pair: $[B(A(P, Q), C(R, S)), A(C(R, B(Q, S)))]$. The completion procedure could then choose to rewrite the first member of this pair using (2): $B(A(P, Q), C(R, S)) \rightarrow C(B(A(P, Q), R), S)$. But this last term, can't be a configuration of any dominance graph that has an instance of t and the corresponding instance of the critical pair among its configurations: A dominance graph has at least one outgoing dominance edge for every hole. If the graph has an instance of the second member of the pair ($\sigma(A(C(R, B(Q, S))))$ for some $\sigma \in Sub(T(\Sigma, \chi))$) among its configuration, the second hole of the root B has to domine one node of $\sigma(S)$. Hence $\sigma(C(B(A(P, Q), R), S))$ cannot be a configuration of the same dominance graph, because it doesn't realize this dominance edge.

Computing the critical pairs of the two rewrite systems points out another problem: there are around 30000 of them for the equivalence rewrite system and 50000 equivalence-weakening or weakening-equivalence pairs. Normalizing them with a bottom-up or top-down strategy to eliminate those which yield the same normal forms only decrease the number of critical pairs up

to the half of the original ones. This number should be compared with the size of the original rewrite system; around 600 rules, much less. For the equivalence rewrite system, this means that running only one step of completion procedure without any further modification would add around 15000 rules. This is not manageable. Moreover explicitly computing all normal-forms for the left and right hand side of each critical pair and testing the two obtained set for intersection shows that only 2.6% of the equivalence-only critical-pairs are actually unjoinable (around 12% and 13% for the weakening-equivalence, which is still very small regarding the total numbers of pairs). However, this approach becomes too slow when the depth of rules increases a little, for instance after one step of completion.

We have pointed out two issues: the first one concern the possible loss of confluence occurring when restricting the rewrite relations of the completed rewrite-systems to the configurations of a dominance graph, the second the dramatically grow of the size of the rewrite system when using a "naive" normalization strategy. The following presents optimizations of the completion that partially address these two issues.

4 Optimizations relative to dominance graphs

4.1 Filtering rewrite by existence of a common accepting dominance graph

Going back to example 3 we can discuss the legitimity of applying rule (2) to rewrite $B(A(P,Q),C(R,S))$ when rule (3) could have been applied to the same term leading to $u = C(R,B(A(P,Q),S))$ and it would not have been true that for all dominance graph d and substitution sigma, instantiating the overlap-term t, its two critical rewrites and u with sigma gives at least a term that is not in $Conf(d)$.

Indeed it turns out that this property can quite easily be checked on a list of linear terms ranging over the same variables, with the same sets of labels, and such that every label occurs only once in it. The latter assumption is a bit annoying in our case, but even after any number of completion steps, our systems remains a "permutation" system to the extent that applying a rule only change the positions of the nodes of the tree. Thus assigning a unique name to each node of the tree we can split a label that occurs at two different position into two different labels and assume that every label of a term occurs only once in it, and that it remains true for any of its rewrite. (for instance $\exists(P,\exists(Q,R))$ can be specialized as $\exists_{\#1}(P,\exists_{\#2}(Q,R))$ and applying a equivalence rule it becomes $\exists_{\#2}(Q,\exists_{\#1}(P,R))$).

for a given dominance graph d and set S of such terms Let $Vars$ denote the common set of variables ($Vars = Vars(t) \forall t \in S$), L the common set of labels and $Common(S,d)$ be the following property:

$$\exists C \in \mathcal{C}(\Sigma),\ \exists \sigma \in Sub(T(\Sigma, Vars),\ \forall t \in S,\ C[\sigma(t)] \in Conf(d).$$

and let $ExistsCommon(S) = \exists d :$ dominance graph , $Common(d;S)$.

$\forall f \in L$ s.t , $\forall i \in [\![1; arity(f)]\!]$ we define $domVars(f,i) = \bigcap_{t \in S} Vars(t_{pos(f,t)\cdot i})$ where $pos(f,t)$ is the position at which the label f occurs in t. We claim that $ExistsCommon(S)$ iff $\forall f \in L$, $\forall i \in [\![1; arity(f)]\!]$, $domVars(f,i) \neq \emptyset$.

A proof is proposed in annex.

Computing $DomVars(f,i)$ for all labels f is easy, and we can use this proposition to check $ExistsCommon(S)$. We can modify both completion procedures as follows without breaking their validity because all critical pairs are still joinable after completion. When a critical pair (t_1,t_2) occurs outstanding from a term t in which two rules overlap, we can first, check $ExistsCommon(\{t,t_1,t_2\})$ and leave this critical pair in case of negative answer, because no dominance graph can allow this three terms to be instantiated together and so to lead to an unconfluent rewrite. Then in case of positive answer, instead of normalizing t_1 and t_2 w.r.t some term rewrite system, we use the following:

Require: S, t
  while $t \to t' \wedge ExistsCommon(S \cup \{t; t'\})$ **do**
    $t = t'$
    $S = S \cup \{t\}$
  end while
  Return (t, S)

First with $S = t, t_1, t_2$ to rewrite t_1 into some $\hat{t}_1$ with a set S', then with $S = S'$ to rewrite t_2 into some $\hat{t}_2$.

Experimentally, this works very well: first because it basically rejects some useless critical pairs, then because it seems to offers a very good compromise between normalizing with a naive but quick strategy, and computing the set of all normal-forms: the rewrite of the second term is "directed" by the restriction built by rewriting of the first one, and in much cases it seems to orient rewrite of the second term in the good direction to find out that the two sides of the critical pair are joinable without explicitly computing and comparing all normal forms.

Moreover, we get with this only chain of rewrites such that there exists a common dominance graph verifying that all elements of the chain are within its configurations. It is not sufficient to reach relative confluence but makes loss of confluence due to restriction to a dominance graph like the one of example 3 more unlikely to arise.

Since the beginning we deal with completion procedures that depend of a particular given dominance graph. The advantage of this is to allow to complete a rewrite system once, and then only use it. But we can still refine the way to rewrite left-hand sides and right-hand sides of critical pairs, using specific informations given by a dominance graph.

4.2 Fully relative completion

Given a dominance graph d we can adapt the idea presented in the previous subsection in a way than strengthen it using the information of actually knowing the dominance graph. Indeed, knowing d it is feasible for a Set of term S to check whether there exists a context C and a substitution σ such that for all t in S, $C[\sigma(t)] \in Conf(d)$[4] let $SameContextConfigurations(S, d)$ denote this property.

Thus we do exactly the same as before but this time filtering critical pairs, as well as rewrites by the existence of a common context and substitution keeping things into configurations of a dominance graph:

Require: S, t, d
  while $t \to t' \wedge SameContextConfigurations(S \cup \{t; t'\}, d)$ **do**
    $t = t'$
    $S = S \cup \{t\}$
  end while
  Return (t, S)

First with $S = t, t_1, t_2$ to rewrite t_1 into some $\hat{t}_1$ with a set S', then with $S = S'$ to rewrite t_2 into some $\hat{t}_2$.

Relative completion often drastically reduces the numbers of critical pairs, and then allow to really focuses completion on the subset of the rules that applies to configurations of a dominance graph. However, this is still insufficient to guarantee that restricting the completed rewrite systems to the set of configurations of a dominance graph even if it still makes us closer from a relatively confluent system. However, doing completion for every graph is an expensive counterpart.

Using these optimizations we were able to implement reasonably fast and efficient completion

[4] using a tree-automaton recognizing the set of all configurations of d and checking for a common states assigned to leaves and to the root of all $t \in S$ and allowing a full run over every t. We do note detail this because introducing tree-automata here would takes too much place and would not be very relevant regarding all other parts of this report

procedures. It remains to check in what proportion completion reduces the number of rules and increase computation time. The next section exposes implementations results obtained at the end of the internship.

5 Implementation results

All results presented here were obtained using the relative completion procedures(equivalence and weakening) of above, because implementation of global completion was not put to an end at the end of the internship (principally because of the need of assigning node names to each labels and respecting these assignations after permutating as we saw in subsection 4.1).

We have performed two experiments over the same corpus of dominance graph than the one used in [KT10]. First two general remarks:

- Relative equivalence completion terminates over all dominance graph of the corpus
- This is not the case for the weakening completion: Even if it terminates for the most of the graphs, it still needs to be bounded for graphs that have a very big number of readings (like around a billion).

The first experiment was completion with a bound of no more than 1000 rules added in each completion procedure. We computed an average number of 3.5 readings in a total time of 2719665 ms. Reducing the corpus without completion takes only around 90000 ms but computes an average number of readings of 9.9 readings. Since it is useless to complete a rewrite system that already have only one reading before completion and there are a lot of graph of the corpus that are found to have only one readings in an almost negligible time of computation (without completion), it is also interesting to see the results for graph with a number of readings greater than 2: without completion the average number of readings is 27.6 with completion, it is 8.5 with completion (and in a time of 2055286 ms which represensant 75% of total computation time for only 33% of the data).

The second experiment consisted in doing just one step of relative completion in each procedure, we found an average of 4.6 readings with a total computation time of 312659 ms which is much faster and still reduced a lot the number of readings.

We can draw two conclusions out of these results: first, both experiments, especially the first one, confirm our expectation that completion actually reduces the number of readings. It is also very expensive in term of computation time. Then, the difference between the results of the two experiment confirm that there is a trade off between precision and computation time: the second experiment do a very superficial and therefore much faster completion but still reduces a lot the number of readings. So one should try to adapt the depth of completion with its need in term of precision.

It turns also out that the time lost at completion itself is not the only one making things slower. Increasing the number and the depth of rewrite rules is also very expensive: over this two experiments, another interesting thing to see was how increasing the depth of the rules slows the method itself, without taking the time spent at completing into account.

Considering only the time spent at computing weakest readings with Koller and Thater's method, it takes 8109 ms on the whole corpus, with an average of 6.51 ms without completion. With the rules added in the first experiment, it takes 54167 ms for all tests, and the average becomes $43,5$ ms, about 6.7 times longer. With the second experiment, testing on the whole corpus takes 25421 ms, with an average of $20,4$ ms, about 3 times longer. This shows that adding a lot of rules of high depth, makes the method itself rather slower. Once again, the further completion is done, the bigger depth the rewrite-rules have, and the slower the algorithm is, but the smaller the set of readings is.

6 Conclusion

The work presented here precises in what extent completion can be useful to compute weakest readings of an ambiguous sentence and defines a reasonable hope we can have in it which consists in getting rewrite systems that better approximates logical entailment and logical equivalence. We have designed two optimized adaptations of the basic completion procedure to our case: a global one, that completes the rewrite system without taking a particular dominance graph into account and thus allows to complete only once the rewrite system and use the output for any dominance graph, and a relative one, that does completion for any different graph but add rules in a more accurate way and is more likely to terminate. Both completion always add rules in a fair manner (*i.e* never breaking soundness), and tend to achieve the previously defined expectation. We have fully implemented the second one, and obtained results that allows us to answer the initial question: using completion actually reduces the number of found readings but at the cost of a considerably increase in term of computation time, even when not taking the time spent at completion into account. This means that even global (non-relative) completion should increase a lot the time needed to reduce a dominance graph.

However, a lot of open question and future works remain: first, it would be very interesting to implement the non-relative completion that works independently of any dominance graph, to see if it would actually save the completion time or if the globally completed rewrite system is too big and slows Koller and Thater's method too much. Then although our two experiments highlight the trade off between the number of readings and computation time, it does not clearly quantify it, but our implementation could be used to perform more experiments and try to achieve this. Moreover, we have shown that Koller and Thater's method becomes very slow when the depth of rules increases. Since Koller and Thater's method was designed for rules of depth two, we could try find out if the method could be modified for a better support of high-depth rules. Another open question concern the difference between normal forms and relative normal forms. As we noticed, what happens when computing relative normal forms of the completed rewrite systems is not clear. Even if the rewrite system is confluent, a term can have more than one normal forms. This yield two possible investigations: we could first try to find a way of completing that preserve a relative confluence. On the other hand, it could be interesting to see if another kind of relative normal that allows to step out of the configurations of a dominance graph could be computed: given a dominance graph D, we could try to consider terms obtained rewriting a term $t \in Conf(D)$ into another one $t' \notin Conf(D)$, and then rewrite t' into a normal forms in $Conf(D)$ as relative normal forms.

7 Bibliography

References

[ADK+03] E. Althaus, D. Duchiera, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. An efficient graph algorithm for dominance constraints. *Journal of Algorithms*, 48:194–219, 2003.

[BC81] J. Barwise and R. Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4:159–219, 1981.

[BN99] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, 1999.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree automata techniques and applications*. http://www.grappa.univ-lille3.fr/tata, 2007.

[KB70] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, page 263–297. Pergamon Press, Oxford, 1970.

[KRT08] A. Koller, M. Regneri, and S. Thater. Regular tree grammars as a formalism for scope underspecication. In *Proceedings of ACL-08: HLT*, 2008.

[KT10] Alexander Koller and Stephan Thater. Computing weakest readings. In *Proceedings of the 48th ACL*, 2010.

8 Annex

Proof of $ExistsCommon(S)$ iff $\forall f \in L$, $\forall i \in [\![1; arity(f)]\!]$, $domVars(f,i) \neq \emptyset$**:**

$\Longrightarrow$ Assume

$$ExistsCommon(S) \wedge \exists f \in L,\ \exists i \in [\![1; arity(f)]\!],\ domVars(f,i) = \emptyset$$

. Let d be the dominance graph that satisfies $Common(d,S)$ and n=$arity(f)$ and $t \in S$. We denote by L_d the subset of the roots of d corresponding to the symbols of t (they are the same whatever the choice of t among S), and let f_d be the node of L_d which is assigned the label f. There is a subterm $f(t_1, \ldots, t_n)$ at some position in t. If t_i is variable x, because each root in d has at least one outgoing dominance edge from each hole, and $C[\sigma(t)] \in Conf(d)$, there is an outgoing edge from the i^{th} hole of f_d and it has to be incident to some node of $\sigma(x)$. But since $domVars(f,i) = \emptyset$ there must be at least one tree $t' \in S$ s.t $x \notin Vars(t'|_{pos(f,t')\cdot i})$. Hence f is not a parent of any node of σx in $C[\sigma(t')]$ and the previous dominance edge is not realized, which is contradictory. If t_i is not a variable, then either there is a variable x and a node of $\sigma(x)$ such that there is a dominance edge from the i^{th} hole of f_d to this node in d and the same reasoning can be applied, or there is a dominance edge from the i^{th} hole of f_d to some other node of L_d corresponding to another label of L. Let g_d be this node and g the corresponding label. Since in every tree u of S, f is a parent of g, $Vars(u|_{pos(g,t)}) \subseteq Vars(u|_{pos(f,t)})$ thus, $\forall i \in [\![1; arity(g)]\!]$, $domVars(g,i) = \emptyset$, and we can starts again at the beginning replacing f with g. Since the numbers of label in L is finite, we end up in one of the other cases at some point.

$\Longleftarrow$ Consider a dominance graph d containing $|Vars|$ roots labeled with arbitry constant symbols (not necessary different) of Σ, and assign one of this node (c_x) to each variable symbol x of $Vars$. assume that d also contain a root f_d for each symbol f in L, with $arity(f)$ holes. For every symbol f in L, for every i in $[\![1 \ldots arity(f)]\!]$ we can chose a variable x in $DomVars(f,i)$ because it is not empty, and add a dominance edge from the i^{th} hole of f to c_x. Using a context C reduced to a single variable, and a the subsitution σ that assign to each variable x of $Vars$ the constant-label of c_x, $C[\sigma(t)]$ is in $Conf(d)$ for all $t \in S$.