Tree Descriptions, Constraints and Incrementality

Denys Duchier (duchier@ps.uni-sb.de)
Programming Systems Lab, Universität des Saarlandes

Claire Gardent (claire@coli.uni-sb.de)
Computational Linguistics, Universität des Saarlandes

Abstract. In (Duchier and Gardent, 1999), we presented a constraint-based method for enumerating the models satisfying a given tree descriptions and described its application to the underspecified semantic representation of discourse advocated in (Gardent and Webber, 1998). In this paper, we indicate how the approach may be further extended to support discourse level *incremental* processing.

Keywords: incremental processing, underspecified representations, tree descriptions, dominance constraints, constraint programming, discourse semantics

1. Introduction

In (Duchier and Gardent, 1999), we presented a constraint-based approach for solving tree descriptions and described its application to the underspecified semantic representation of discourse advocated in (Gardent and Webber, 1998). As later work showed, the strength of the proposal is that it provides a general logical framework and a processing method which can be tailored depending on the application. For instance, (Duchier and Thater, 1999) shows that it can be customised to description-based syntactic parsing while (Egg et al., 1998) adapts it to deal with underspecified semantic representation at the sentential level.

In this paper, we indicate how the approach may be further extended to support incremental discourse processing.

We first give an informal explanation of how descriptions can be exploited to incrementally process discourse. Thus Section 2 motivates the use of tree descriptions; Section 3 sketches an architecture for incremental processing which rests on the notion of Solved Forms; Section 4 gives an intuitive introduction to this notion; And Section 5 shows the architecture at work by going through some example analyses.

We then show how the constraint-based approach to descriptions presented in (Duchier and Gardent, 1999) can be extended to permit incremental processing: Section 6 introduces the logical framework used to talk about trees and section 7 presents a constraint-based method for computing the partial structures built during incremental processing.

© 2000 *Kluwer Academic Publishers. Printed in the Netherlands.*

Our formal presentation follows the recent work of Duchier and Niehren (2000).

2. Description-Based Incremental Processing

It is well known from the work of M.P.Marcus et al. (1983) and later psycholinguistic work (Pritchett, 1992; Gorrell, 1995; Sturt and Crocker, 1996) that the use of descriptions of trees rather than trees nicely supports incremental processing. The crucial observation is that the use of dominance rather than strict dominance permits (i) a monotone treatment of attachment ambiguity and (ii) a distinction to be made between "simple" local ambiguity and "garden-path" local ambiguity (i.e. ambiguity that leads to conscious reanalysis of the syntactic structure built so far).

Gardent and Webber (1998) further extend the use of descriptions to discourse, showing their benefit for incremental discourse processing. In particular they argue that discourse semantics exhibits the same type of local ambiguities as sentential syntax (simple and garden-path) and that therefore the same benefits accrue from the use of descriptions in incremental near-deterministic discourse processing as in incremental syntax (additionally, they argue that the use of descriptions permits a deterministic treatment of global ambiguity).

The question therefore arises of how an incremental processor can be defined which produces the appropriate descriptions. In the psycholinguistic literature (Gorrell, 1995; Sturt and Crocker, 1996), the approach taken is to define update operations on descriptions which ensure that the incremented description (i) is tree shaped and (ii) preserves word order (the sequential order of the leaves in the tree match the order of the words in the input).

We propose an alternative approach to description-based processing which rests on a logical perspective. In this approach, descriptions are viewed as formulae of a tree logic and trees as models satisfying these formulae. Moreover, *solved forms* can be derived from descriptions by means of a normalization process. A solved form is a notion closely related to that of D-tree (Rambow et al., 1995) and is guaranteed to be satisfiable.

Within this perspective, incremental processing consists in (i) conjoining the description built so far with the description associated with the incoming unit and (ii) computing the solved forms satisfying this conjunction.

We now informally describe the workings of an incremental discourse processor based on this idea first by sketching an architecture for discourse level, incremental processing (section 3 and 4) and second by illustrating its operation by means of examples (section 5).

3. An Architecture for Discourse Processing

Following (Webber and Joshi, 1998; Hitzeman et al., 1995), we view discourse parsing as not essentially different from sentence parsing. In both cases, a grammar is used which describes the syntax and the compositional semantics of natural language. The parser then uses this grammar to build the appropriate descriptions.

Naturally, the grammar must extend to discourse. We assume a grammar in the spirit of Webber's Lexicalised Tree Adjoining Grammar (LTAG) for discourse (Cristea and Webber, 1997; Webber and Joshi, 1998; Webber et al., 1999) where discourse connectives are treated either as functors or as modifiers and clauses as arguments of these functors and modifiers.

To support incremental processing, we further assume that Webber's LTAG for discourse is modified in two ways. First, the structures associated by the grammar with the discourse units are descriptions of trees rather than trees. Second, the syntax/semantic interface is made precise by using a synchronous LTAG (Shieber and Schabes, 1990) i.e. two LTAGs, one for the syntax and one for the semantics, which are interfaced via a synchronisation relation.

In short, the grammar framework we are assuming is a discourse variant of Kallmeyer's Synchronous Local Tree Description Grammar (Kallmeyer, 1998; Kallmeyer, 1999). We assume a synchronous grammar to provide a TAG-like discourse grammar with a well-defined syntax/semantic interface, and we require that the objects defined by the grammars be tree descriptions rather than trees in order to support incremental processing both at the syntactic and at the semantic level.

Given such a grammar, an incremental discourse parser could then function as follows. As each new discourse unit (i.e. clause or discourse connective) is processed:

1. The syntactic and semantic descriptions of the new unit, together with any additional constraints from the syntax/semantics interface, are conjoined to the description accumulated sofar.

2. The resulting description is then subjected to a normalization process that produces the corresponding *solved forms.*

3. If there are no solved forms, the description is not satisfiable and the parser must backtrack. Otherwise, by appeal to a preference criterion, it non-deterministically picks one and proceeds with it.

4. Tree Descriptions, Dtrees and D-Solved Forms

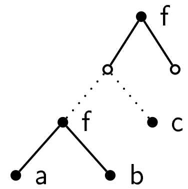
While our formalism (Section 6) is generally more expressive than D-trees, the latter have the advantage of familiarity and can be more intuitively presented by means of graphical illustrations. For this reason, we now describe a variant of D-trees called *D-solved forms* that is appropriate for introducing our formalism and processing architecture. This variant is used throughout Section 5 to illustrate incremental discourse processing.

Rambow et al. (1995) define a D-tree as a tree with domination edges (d-edges) and immediate domination edges (i-edges). We depart slightly from their definition and distinguish *open* and *closed* nodes:

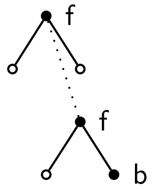
- a closed node has only i-edges. Its arity is fixed. We do not allow a non-monotonic operation such as sister-adjunction.

- an open node has only d-edges. Its arity is unknown. We do allow more than 1 d-edge.

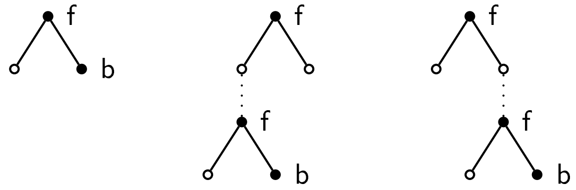
We draw a d-edge as a solid line, an i-edge as a dotted line with the dominated node lower than the dominating one, a closed node as black circle, and an open node as a hollow circle. A node may be labelled with a constant from a given signature: in this case the label is displayed next to the node.



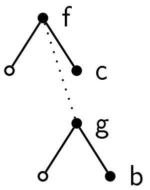
At each step of incremental processing, the current description is augmented with new material and this new material is related to earlier one e.g. by d-edges. For example, we might thus obtain a description of the form:



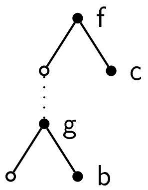
This is *not* a D-solved form since there is a closed node with an outgoing d-edge. We can obtain a D-solved form either by identifying the two end-points of the d-edge, or by propagating the d-edge downward to one of the daughters. Thus, in this case, there are three possible D-solved forms:



Often, constraints of labeling or precedence can make the process of obtaining a D-solved form deterministic. Consider:



Identification of the nodes labeled f and g is not possible since $\mathsf{f} \neq \mathsf{g}$. Also the node labeled c is closed, has no daughters, and cannot be identified with the node labeled g. Therefore only one D-solved form remains:



This result can be derived purely through deterministic inference. In Section 6.4, we will make precise both the system of inference rules and the formal definition of a D-solved form.

5. Incremental Processing Illustrated

We now describe an idealised analysis of examples involving simple and garden-path ambiguity. The analysis is idealised in that it assumes – rather than uses – the incremental discourse processor sketched in section 3. In other words, the input descriptions are given by reasoning about the syntax/semantic interface of the input discourse rather than by the parsing process.

Given this simplifying assumption, we show that the solved forms computed from the input descriptions either support determinism (in the case of simple ambiguity) or force backtracking (in the garden-path cases).

5.1. Attachment ambiguity

When processing incrementally, it is sometimes unclear how far below an already existing node, the incoming structure should be attached. Such ambiguity is known in the literature as *attachment ambiguity.* It is illustrated by the following examples.

(1) On the one hand (a) Jon is content.
On the other hand (b) Mary isn't.

(2) On the one hand (a) Jon is content if (b) he can read a novel.
On the other hand (c) He is too poor to buy books.

(3) On the one hand (a) Jon is content if (b) he can read a novel or if (c) he can go to the movies
On the other hand (d) He is too poor to do either.

Figure 1 gives the syntactic and semantic structures associated with examples (1) and (2). The gray arrows indicate the relations of synchronization between syntax and semantics. As these structures show, (a) might attach arbitrarily low in the syntactic as well as in the semantic structure.

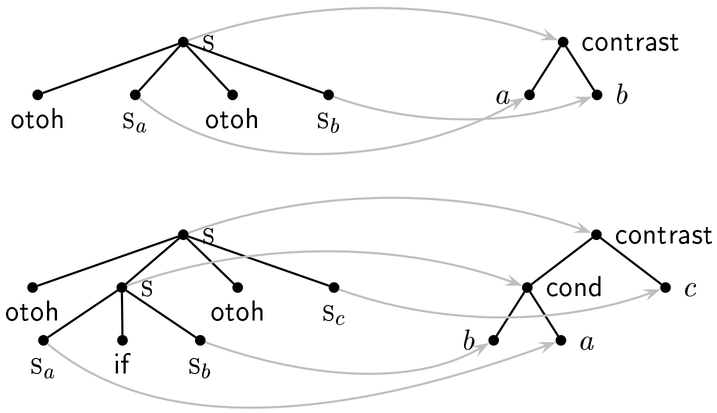


Figure 1. Attachment Ambiguity

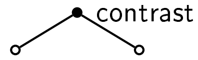
Attachment ambiguities raise two issues. First, a representation must be found which is compatible with the theoretically infinite set of possible continuations. Second, since such ambiguities do not lead the hearer down the garden path, the chosen representation must only commit to those aspects of syntax/meaning which cannot be defeated by later information.

Now we know from Marcus' work and from related work in psycholinguistics (Pritchett, 1992; Gorrell, 1995; Sturt and Crocker, 1996) that tree descriptions provide the right amount of underspecification to solve both these issues: by using dominance rather than strict dominance, a finite representation of the syntactic tree can be obtained which is compatible with every possible completion of the sentence.

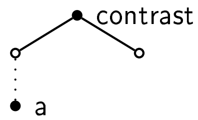
The question we are addressing is this: How can the appropriate tree descriptions be built incrementally from the input discourse?

We illustrate this process by going through the semantic derivation of example (2) and showing how, given some standard assumptions about the syntax/semantic interface, the appropriate solved forms can be computed from the conjunction of the description built so far with the description of the incoming basic discourse unit.

For the purpose of this paper, we take basic discourse units to be either discourse connectives or clauses. The first basic discourse unit in example (2) is *on the one hand*, a discourse connective which at the semantic level, denotes a relation of contrast between two eventualities. This is captured by associating with it the following semantic representation:



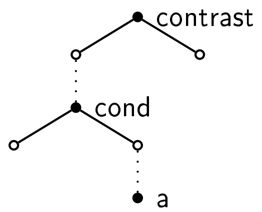
Next the (a) clause *Jon is content* is processed. Syntactically, S_a must be part of the first argument of the connective *on the one hand/on the other hand* since (i) S_a is right-adjacent with *on the one hand* and (ii) *on the other hand* has not yet been processed. By compositionality, the semantic representation a of S_a must therefore be part of the first semantic argument of the contrast relation. Hence, the solved form for *on the one hand, Jon is content* is:



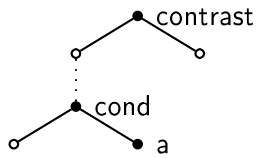
Intuitively, this solved form indicates that at this stage in processing, the interpretation available to the hearer/reader is that there is a relation of contrast holding between the eventuality denoted by S_a and some other eventuality.

Now consider how processing continues in a case where the (a) clause turns out to attach lower in the tree e.g. in the case of example (2).

Next *if* is encountered which is associated with a semantic representation similar to that of *on the one hand/on the other hand* but where the relation labelling the root node is cond (for "condition") rather than contrast. By the same reasoning as for s_a, the semantics of *if* must be part of the first semantic argument of *on the one hand/on the other hand.* Furthermore, since infix *if* requires a left-hand argument and s_a is left-adjacent to *if*, s_a must be part of this left-hand syntactic argument and consequently, its semantics a must be part of the consequent of the conditional. Given this, the solved form for *on the one hand, Jon is content if* will be:



Moreover, since all the material to the left of *if* has been processed, the consequent argument of the conditional can be closed:



Thus the parser processes attachment ambiguity deterministically by monotonically adding information to the current description and each time computing the corresponding solved form.

5.2. Preference criterion and garden path sentences

From a psycholinguistic perspective, two types of ambiguities are generally distinguished: those that lead to processing difficulties (conscious re-analysis) and those that do not (unconscious re-analysis).

In the preceding section, we saw how solved forms support a deterministic treatment of discourse-level ambiguities which intuitively do not seem to involve conscious re-analysis namely, attachment ambiguities. We now show that not all discourse level ambiguities can be processed deterministically within our framework and thereby predict, as for sentential syntax, that discourse level ambiguities can be of two types: those that can be processed deterministically within the description framework and those that cannot.

The examples we consider are the following:

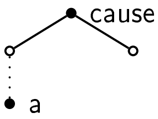
(4) Because (a) Jon is easily upset, whenever (b) he flies, (c) he gets very nervous.

(5) Because (a) Jon is easily upset, whenever (b) he flies, (c) he goes to Paris for example, (d) he should practice yoga.

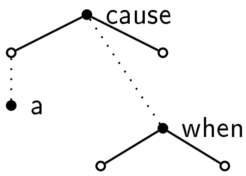
(6) Because (a) Jon is easily upset, whenever (b) he flies, (c) he gets very nervous for example, (d) he should practice yoga.

Intuitively, there is a stark contrast in processing ease between (4) and (6): whereas (4) is easy to process, (6) is much more difficult and seems to involve a garden path effect. The situation is less clear in (5) though there seems to be a slight increase in processing difficulty relative to (4). In what follows, we show that the description based framework sketched here predicts these differences and thereby offers a basis for experimental testing. Whereas (4) can be processed deterministically and (5) implies a very limited backtrack, (6) involves extensive backtracking.

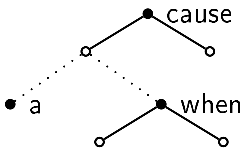
We first go through the derivation for (4). By a reasoning similar to that for example (2) above, after processing the (a) clause the solved form is:



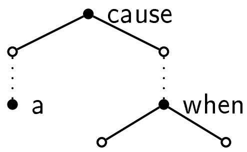
Next *whenever* is processed extending the description with a binary tree representing the when relation. Since the connective *because* requires two right-hand arguments and *whenever* is the second basic discourse item occurring to its right, the proposition expressed by *whenever* and its arguments must be within the scope of *because*. Hence the description associated with *Because (a) Jon is easily upset, whenever* is:



This description has two solved forms:

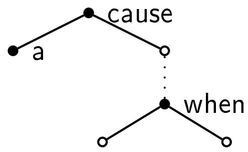


(7)

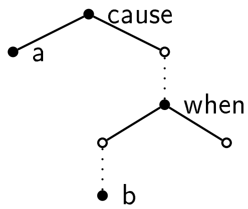


(8)

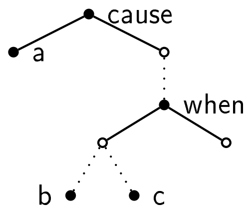
Argument Filling Principle. In order to preserve determinism, some preference criterion must be determined which permits choosing between the two forms. We use a criterion (henceforth called the *Argument Filling Principle*) similar to Gorrell's (1995) Incremental Licensing principle or to Sturt and Crocker's (1996) preference for substitution over adjunction: we prefer normal forms which provide material for an earlier argument that was so far empty. Thus here, we prefer (8) because it provides material for the second argument of the cause relation whilst (7) leaves it empty. If we (standardly) assume that the arguments of a discourse relation are given by adjacent material, the fact that *whenever* is committed to being part of the second argument of *because* means that the latter's first argument is now closed: it cannot be extended by material occuring later in the discourse. Thus the solved form now is:



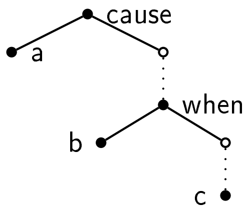
Next the (b) clause is processed which given the syntax and semantics of *whenever* can only be part of its first syntactic and semantic argument:



Again since *whenever* takes two arguments and the (c) clause is the second basic discourse item to its right, (c) must be within the syntactic and semantic scope of *whenever*. Given the resulting constraints, we again have two solved forms:

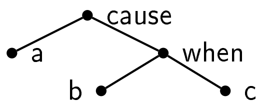


(9)



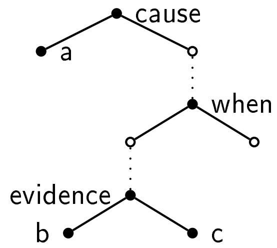
(10)

By the Argument Filling Principle mentioned above, (10) is preferred because it fills the second valency of *whenever* instead of leaving it empty. As a result, the reading obtained for (4) is:

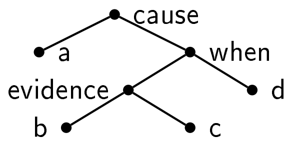


(11)

This shows that example (4) can be processed deterministically. Now consider how the derivation would proceed given example (5). In this case, the next discourse item is the connective *for example* which takes two arguments to its left. This cannot be satisfied in (10), therefore we must backtrack: (9) is the chronologically closest alternative and allows b and c as arguments for the semantic relation of evidence. Since *for example* occurs to the right of the corresponding clauses, no further material can be added to these arguments and so they can be closed. The resulting solved form is:

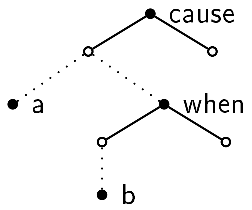


Finally, the (d) clause is processed which permits filling the open valency of *whenever*. The following semantic representation is therefore assigned to (5).

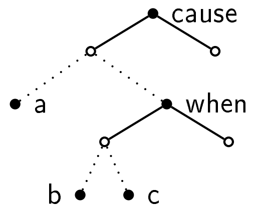


Thus for examples such as (5), the approach predicts a limited backtracking. Intuitively at least, this matches the fact that example (5) is relatively easy to process: the garden path effect induced by "for instance" is very mild.

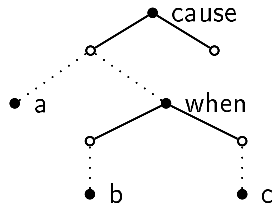
Now consider again example (6), in which the garden path effect is much stronger. In this case, backtracking to the solved form in (9) is not sufficient because it would involve an evidence relation to be posited between (6b) and (6c) and this is ruled out by pragmatics: *he gets very nervous* cannot be taken as giving evidence for *he flies*. Therefore we must backtrack further and start from the next alternative namely (7). By the same argument as before, b must be in the first argument of when.



Further, c must be below when since the latter doesn't have all its arguments yet. Again we obtain two normal forms:

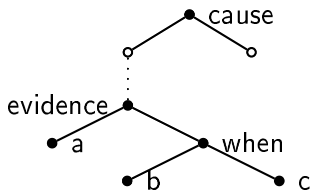


(12)

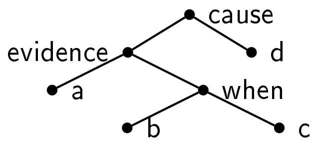


(13)

Following the *argument filling principle*, we prefer (13). Now evidence needs to find two arguments using material on the left: the only possibilities are a and the subtree rooted in when. Moreover the tree below evidence can be closed since it is formed only from earlier material.



Finally d fills the second argument of cause and so we obtain the following semantic representation for (6).



6. Dominance Constraints

In this section, we introduce a logical framework for tree descriptions. In 6.1 we introduce a language of dominance constraints for writing tree descriptions. In 6.2 we give its semantics by interpretation over finite tree structures. In 6.3 we intuitively motivate the notion of solved form, and in 6.4 we formally define it in terms of saturation with respect to a system of inference rules.

6.1. Language

In (Duchier and Gardent, 1999), we followed a classical presentation of dominance constraints in which a tree description is given by a conjunction of dominance literals $x \lhd^* y$ and labeling literals $x{:}f(x_1, \ldots, x_n)$ where variables denote nodes in the tree. $x \lhd^* y$ expresses that the node denoted by x is equal to or a proper ancestor of the node denoted by y and $x{:}f(x_1, \ldots, x_n)$ expresses that the node denoted by x must be formed from the n-ary constructor f and the sequence of daughter nodes denoted by x_1 through x_n.

However, the constraint treatment we proposed turned out to be more general and the added expressivity was noticed by Duchier and Niehren (2000) and formalized under the name of *dominance constraints with set operators*. It is this revised formulation which we adopt now. The abstract syntax of dominance constraints with set operators is given by:

$$\phi \quad ::= \quad x\, R\, y \;\mid\; x{:}f(x_1, \ldots, x_n) \;\mid\; \phi \wedge \phi'$$

where x, y, x_i range over an infinite set of node variables, f ranges over a finite signature Σ, and R ranges over arbitrary subsets of the relation symbols $\{=, \lhd^+, \rhd^+, \perp\}$. The symbol $\lhd^+$ denotes proper dominance and $\perp$ represents disjointness. In a dominance literal $x\, R\, y$, R is called a set operator and is given a disjunctive interpretation: one of the relations in R must hold between the nodes denoted by x and y. For example $x\,\{=, \perp\}\, y$ is satisfied either if the nodes denoted by x and y are equal, or if they lie in disjoint subtrees.

In all tree structures we have $\neg(x\, R y) \equiv x \neg R y$ and $x\,(R_1 \cup R_2)\, y \equiv x\; R_1\; y \;\;\vee\;\; x\; R_2\; y$. Thus set operators introduce a controlled form of negation and disjunction without admitting full propositional connectives.

The formal account can be straightforwardly extended to permit relations of precedence, see e.g. (Duchier and Thater, 1999). In this case the set of relation symbols is $\{=, \lhd^+, \rhd^+, \prec, \succ\}$.

6.2. Semantics

The semantics of dominance constraints with set operators are given by interpretation over finite tree structures. We identify a node in a tree with the path that leads to it starting from the root. A path π is a word (i.e. a sequence) of positive integers. We write ϵ for the empty path and $\pi_1\pi_2$ for the concatenation of π_1 and π_2. We say that π' is a proper prefix of π, and write $\pi' \lhd^+ \pi$, if there exists $\pi'' \neq \epsilon$ such that $\pi = \pi'\pi''$. We say that π_1 is disjoint from π_2, and write $\pi_1 \perp \pi_2$, when there exist paths π, π'_1, π'_2 and integers $i \neq j$ such that $\pi_1 = \pi i \pi'_1$ and $\pi_2 = \pi j \pi'_2$. A tree-domain is a non-empty prefix-closed set of paths. A finite tree τ is a triple (D_τ, L_τ, A_τ) of a finite tree-domain D_τ, a labeling function $L_\tau : D_\tau \to \Sigma$, and an arity function $A_\tau : D_\tau \to \mathbb{N}$, and such that for all $\pi \in D_\tau$, $\pi i \in D_\tau$ iff $1 \leq i \leq A_\tau(\pi)$.

We write V_ϕ for the set of variables occurring in ϕ. A model of ϕ is a pair (τ, α) of a finite tree τ and a variable assignment $\alpha : V_\phi \to D_\tau$ mapping each variable of ϕ to a node in τ. We write $(\tau, \alpha) \models \phi$ for the relation of satisfaction and define it as follows:

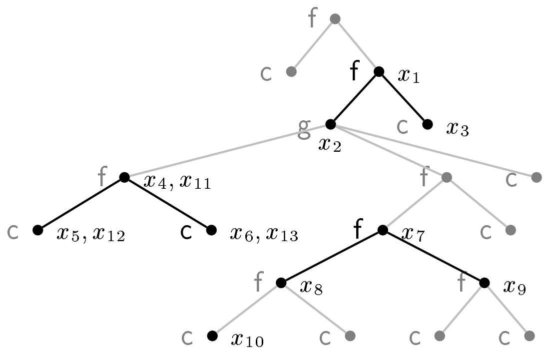
$$
\begin{array}{lll}
(\tau,\alpha) \models \phi \wedge \phi' & \text{if} & (\tau,\alpha) \models \phi \text{ and } (\tau,\alpha) \models \phi' \\
(\tau,\alpha) \models x\, R\, y & \text{if} & \alpha(x)\ r\ \alpha(y) \quad \text{for some } r \in R \\
(\tau,\alpha) \models x{:}f(x_1,\ldots,x_n) & \text{if} & L_\tau(\alpha(x)) = f \quad \wedge \\
 & & A_\tau(\alpha(x)) = n \quad \wedge \\
 & & \alpha(x)i = \alpha(x_i) \quad \text{for } 1 \leq i \leq n
\end{array}
$$

Koller et al. (1998) have shown that the satisfiability of propositional logic formulae can be reduced to the satisfiability of dominance constraints over a signature containing a binary constructor cons and two constants, true and false, thus establishing an NP-hardness result.

6.3. Models and Solved Forms

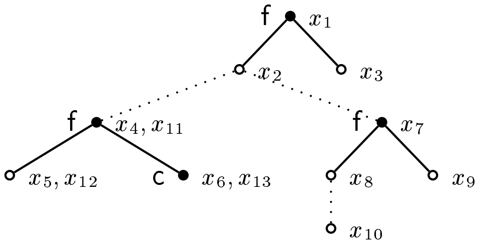
A practical solver cannot simply attempt to enumerate the models of a description: if τ is a model of ϕ, then any tree τ' which contains τ is also a model of ϕ. Thus, whenever a description is satisfiable, it has infinitely many solutions.

Consider the example below. On the right, is a description ϕ and on the left is a possible tree model where each variable is listed next to the node which it denotes.



$$\begin{aligned} & x_1 : \mathsf{f}(x_2, x_f) \\ \wedge\ & x_1 \lhd^* x_7 \\ \wedge\ & x_1 \lhd^* x_{10} \\ \wedge\ & x_2 \lhd^* x_4 \\ \wedge\ & x_2 \lhd^* x_{11} \\ \wedge\ & x_4 \lhd^* x_4 \\ \wedge\ & x_4 \lhd^* x_6 \\ \wedge\ & x_7 : \mathsf{f}(x_8, x_9) \\ \wedge\ & x_{11} : \mathsf{f}(x_{12}, x_{13}) \\ \wedge\ & x_{13} : \mathsf{c} \end{aligned}$$

There is much in this tree which is not required to model ϕ. All the superfluous information is shown in gray. If we remove the gray parts, we are left with a much simpler *tree shape:*



Such a shape is what we introduced in Section 4 under the name of D-solved forms. It may be helpful to draw an analogy between D-solved forms and most-general unifiers. Firstly, just like a most general unifier instantiates two terms only as far as necessary to make them equal, a D-solved form explicitates only as much of the shape of the tree as is necessary to model the description. Secondly, if there is a unifier for an equation $t_1 = t_2$ between first-order terms, then there exist ground solutions. Similarly, if ϕ has a D-solved form, then there exist finite trees which satisfy it.

6.4. Solved Forms and Inferential Saturation

Hopefully, the intuition underlying the notion of D-solved form has become clear and we now define it formally. In this, we follow Duchier and Niehren (2000) who describe an abstract solver based on inferential saturation according to the propagation and distribution rules of Figure 2.

The process of inferential saturation is defined as follows: a propagation rule has the form $\phi_1 \longrightarrow \phi_2$ and is said to apply to ϕ whenever $\phi_1 \subseteq \phi$ and $\phi_2 \setminus \phi \neq \emptyset$. In this case, we proceed with $\phi \wedge \phi_2$. A distribution rule has the form $\phi_1 \longrightarrow \phi_2' \vee \phi_2''$ and is said to apply when

Propagation Rules

$$
\begin{array}{rcll}
x \emptyset y & \longrightarrow & \mathsf{false} & \\
 & \longrightarrow & x \lhd^* x & \\
x \lhd^* y \ \wedge \ y \lhd^* z & \longrightarrow & x \lhd^* z & \\
x{:}f(x_1,\ldots,x_n) \ \wedge \ y{:}f(y_1,\ldots,y_n) \ \wedge \ x = y & \longrightarrow & x_i = y_i & \\
x{:}f(\ldots) \ \wedge \ y{:}g(\ldots) & \longrightarrow & x \neg{=} y & \text{if } f \neq g \\
x{:}f(\ldots,x_i,\ldots,x_j,\ldots) & \longrightarrow & x_i \perp x_j & \text{if } i \neq j \\
x{:}f(\ldots,y,\ldots) & \longrightarrow & x \lhd^+ y & \\
x\,R_1\,y \ \wedge \ x\,R_2\,y & \longrightarrow & x\,(R_1 \cap R_2)\,y & \\
x\,R\,y & \longrightarrow & x\,R'\,y & \text{if } R \subseteq R' \\
x\,R\,y & \longrightarrow & y\,R^{-1}\,x & \\
x \perp y \ \wedge \ y \lhd^* z & \longrightarrow & x \perp z & \\
x \lhd^* z \ \wedge \ y \lhd^* z & \longrightarrow & x \neg{\perp}\, y & \\
x \lhd^* y \ \wedge \ x{:}f(x_1,\ldots,x_n) \ \wedge \ \wedge_{i=1}^n x_i \neg{\lhd^*}\, y & \longrightarrow & x = y &
\end{array}
$$

Distribution Rules

$$
\begin{array}{rcl}
x \lhd^* y \ \wedge \ x{:}f(x_1,\ldots,x_n) & \longrightarrow & x_i \lhd^* y \ \vee \ x_i \neg{\lhd^*}\, y \\
x \neg{\perp}\, y & \longrightarrow & x \lhd^* y \ \vee \ x \neg{\lhd^*}\, y
\end{array}
$$

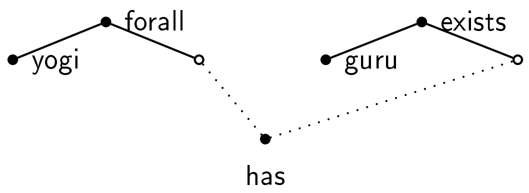
Figure 2. Rule System of Duchier & Niehren

$\phi_1 \subseteq \phi$, $\phi_2' \setminus \phi \neq \emptyset$ and $\phi_2'' \setminus \phi \neq \emptyset$. In this case, we non-deterministically proceed with either $\phi \wedge \phi_2'$ or $\phi \wedge \phi_2''$.

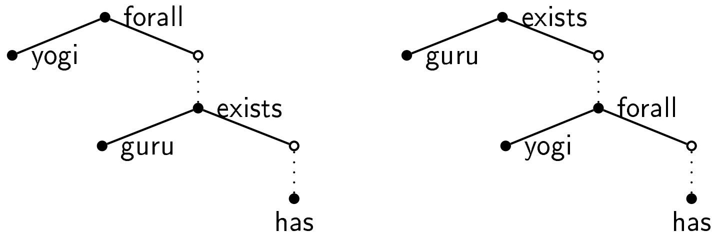
A D-solved form of ϕ is a saturation of ϕ that does not contain false. In other words, it is a consistent saturation of ϕ. Duchier and Niehren (2000) proved that a D-solved form is satisfiable and that the saturation-based solver is sound and complete.

The first alternative of the first distribution rule implements the propagation of a d-edge downward to a daughter as we described in Section 4. Repeated application of its second alternative together with the last propagation rule implements identification.

The second distribution rule takes care of descriptions which are not yet tree shaped. A classical example is quantifier scope ambiguity. Consider the sentence *"every yogi has a guru"* whose underspecified semantic representation in the spirit of (Egg et al., 2000) albeit much simplified is:



In conjunction with the next-to-last propagation rule, the 2nd distribution rule derives the two D-solved forms below which correspond to the two possible scoping arrangements:



A D-solved form is defined as a consistent saturation of a description. The graphical representation which we have been using is essentially a convenient summary of the D-solved form, where redundant constraints have been omitted, in particular all literals $x \lhd^* y$ which can be deduced by transitivity (3rd propagation rule).

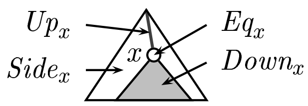
While the graphical representation is intuitive and helpful for illustration, it is not as expressive as the formalism and some literals present in a D-solved form cannot always be faithfully represented. For example literals of strict dominance $x \lhd^+ y$ and of disjointness $x \perp y$.

7. Constraint-Based Incremental Parsing

The NP-hardness result of Koller et al. (1998) is not a show-stopper, but it requires that a practical solver devise very effective means to address the combinatorial complexity of the task, for example by drastically reducing the number of choices that need be considered.

An approach based on constraint propagation has proven particularly successul. Efficient constraint programming solvers can be derived by transformation of a dominance constraint into a constraint satisfaction problem on finite sets (Duchier and Gardent, 1999; Duchier and Niehren, 2000).

7.1. Encoding into finite sets constraints



The idea of the encoding is based on the following observation: when viewed from a specific node x, the nodes of a solution tree are partitioned into 4 regions: the node interpreting x, all nodes above, all nodes below, and all nodes to the side. Therefore the variables which these nodes interpret are similarly partitioned into 4 sets and the idea is to introduce set variables to represent them, then express and solve the problem in terms of these variables.

We are going to present an encoding which transforms a description ϕ into a constraint satisfaction problem (CSP) $[\![\phi]\!]$ expressed solely in terms of variables ranging over finite domains of integers and variables ranging over finite sets of integers. We write V_ϕ for the set of node variables in ϕ. In $[\![\phi]\!]$, every $x \in V_\phi$ must be encoded by a distinct integer; similarly, every $f \in \Sigma$. However, in the interest of legibility, we will leave all such trival encodings implicit. Our encoding consists of three parts:

Representation. for each node variable x, $\mathsf{A}_1(x)$ expresses the local invariants for the CSP variables introduced for x.

Well-formedness constraints. for each pair of node variables x, y, $\mathsf{A}_2(x, y)$ expresses the well-formedness constraints that must be satisfied for a solution to be *tree-shaped.*

Problem-specific constraints. $\mathsf{A}_3[\![\phi]\!]$ forms the problem-specific constraints that restrict admissibility to only those tree shapes that actually satisfy ϕ.

7.2. Representation

We restrict ourselves to trees with a maximum arity (i.e. branching factor) MAX. For the purpose of this paper, MAX can be the maximum arity used in the input description ϕ. For each variable $x \in V_\phi$, we introduce 7+MAX set variables written Eq_x, Up_x, $Down_x$, $Side_x$, $Equp_x$, $Eqdown_x$, $Parent_x$, $Down_x^i$ for $1 \leq i \leq \textsc{max}$, and one integer variable $Label_x$. First, we state that x is indeed one of the variables interpreted by the node which it denotes:

$$x \in Eq_x \tag{14}$$

Eq_x, Up_x, $Down_x$, $Side_x$ encode the set of variables that are respectively equal, above, below and to the side (i.e. disjoint) of x. Thus we have:

$$V_\phi = Eq_x \uplus Up_x \uplus Down_x \uplus Side_x$$

As described in (Duchier and Niehren, 2000), we must improve propagation by introducing $Eqdown_x$ and $Equp_x$ as intermediate results:

$$Eqdown_x = Eq_x \uplus Down_x \tag{15}$$
$$Equp_x = Eq_x \uplus Up_x \tag{16}$$
$$V_\phi = Eqdown_x \uplus Up_x \uplus Side_x \tag{17}$$
$$V_\phi = Equp_x \uplus Down_x \uplus Side_x \tag{18}$$

$Down_x^i$ encodes the sets of variables in the subtree rooted at x's ith daughter (empty if there is no such daughter):

$$Down_x = \uplus\{Down_x^i \mid 1 \leq i \leq \text{MAX}\} \tag{19}$$

The contribution $\mathsf{A}_1(x)$ to the encoding is defined by:

$$\mathsf{A}_1(x) \;=\; (14) \wedge (15) \wedge (16) \wedge (17) \wedge (18)$$

7.3. Wellformedness

Posing $\mathcal{R} = \{=, \lhd^+, \rhd^+, \perp\}$, the relationship R_{xy} that obtains in a solution tree between the nodes denoted by x and y must be one in $\mathcal{R}$. For each $r \in \mathcal{R}$, we can formulate corresponding constraints $\mathsf{D}[\![x\, r\, y]\!]$ on the variables of the CSP that must be satisfied in this case. Similarly for the negation $\mathsf{D}[\![x \neg r\, y]\!]$.

$$\begin{aligned}
\mathsf{D}[\![x = y]\!] \;&=\; Eq_x{=}Eq_y \wedge Up_x{=}Up_y \wedge Down_x{=}Down_y \wedge Side_x{=}Side_y \\
&\quad \wedge Eqdown_x{=}Eqdown_y \wedge Equp_x{=}Equp_y \\
&\quad \wedge Parent_x{=}Parent_y \wedge Label_x{=}Label_y \wedge_i Down_x^i{=}Down_y^i \\
\mathsf{D}[\![x \neg{=}\, y]\!] \;&=\; Eq_x \parallel Eq_y \\
\mathsf{D}[\![x \lhd^+ y]\!] \;&=\; Eqdown_y \subseteq Down_x \wedge Equp_x \subseteq Up_y \wedge Side_x \subseteq Side_y \\
\mathsf{D}[\![x \neg{\lhd^+}\, y]\!] \;&=\; Eq_x \parallel Up_y \wedge Down_x \parallel Eq_y \\
\mathsf{D}[\![x \perp y]\!] \;&=\; Eqdown_x \subseteq Side_y \wedge Eqdown_y \subseteq Side_x \\
\mathsf{D}[\![x \neg{\perp}\, y]\!] \;&=\; Eq_x \parallel Side_y \wedge Side_x \parallel Eq_y
\end{aligned}$$

With these, we can formulate a quadratic number of *wellformedness* constraints. For each $r \in \mathcal{R}$ and $x, y \in V_\phi$:

$$\mathsf{D}[\![x\, r\, y]\!] \wedge R_{xy} = r \quad \textbf{or} \quad R_{xy} \neq r \wedge \mathsf{D}[\![x \neg r\, y]\!] \tag{20}$$

$$R_{xy} \in \mathcal{R} \tag{21}$$

The contribution $\mathsf{A}_2(x, y)$ to the encoding is defined by:

$$\mathsf{A}_2(x, y) \;=\; \wedge\{(20) \mid r \in \mathcal{R}\} \wedge (21)$$

Disjunctive Propagators. The construct (C_1 **or** C_2) used in (20) is called a disjunctive propagator. It has the declarative semantics of disjunction but its operational semantics are those of a constraint rather than a choice point: when C_i becomes inconsistent with the constraints derived so far, then (C_1 **or** C_2) *commits* to (i.e. infers) the other alternative C_j, for $\{i,j\} = \{1,2\}$. A formal statement of its semantics can be found in (Duchier and Niehren, 2000).

7.4. Problem specific constraints

The last part of the encoding forms the problem-specific constraints that further limit the admissibility of well-formed solutions and only accepts those which actually satisfy ϕ.

$$
\begin{array}{rcl}
\mathsf{A}_3[\![\phi \wedge \phi']\!] & = & \mathsf{A}_3[\![\phi]\!] \wedge \mathsf{A}_3[\![\phi']\!] \\
\mathsf{A}_3[\![x \, R \, y]\!] & = & R_{xy} \in R \\
\mathsf{A}_3[\![x{:}f(x_1, \ldots, x_n)]\!] & = & \mathit{Label}_x = f \\
& & \wedge_{i=n+1}^{i=\text{MAX}} \mathit{Down}_x^i = \emptyset \\
& & \wedge_{i=1}^{i=n} (\mathit{Parent}_{x_i} = \mathit{Eq}_x \wedge \\
& & \qquad \mathit{Down}_x^i = \mathit{Eqdown}_{x_i} \wedge \\
& & \qquad \mathit{Up}_{x_i} = \mathit{Equp}_x)
\end{array}
$$

7.5. Stating and solving the CSP

We can now formulate the full encoding by conjoining the contributions defined above.

$$
[\![\phi]\!] \;=\; \bigwedge_{x \in V_\phi} \mathsf{A}_1(x) \quad \bigwedge_{x,y \in V_\phi} \mathsf{A}_2(x,y) \quad \wedge \, \mathsf{A}_3[\![\phi]\!]
$$

To solve $[\![\phi]\!]$ is to find assignments to the CSP variables so that $[\![\phi]\!]$ is satisfied. This is realized by alternating steps of propagation and distribution.

Constraint propagation performs deterministic inference that shrinks the set of possible values that may be assigned to each variable. This set of values is called the domain of the variable. When only one value remains in its domain, we say that the variable is determined, i.e. its assignment has been decided.

When, after propagation, there are still undetermined variables, a step of distribution must be performed: one non-determined variable is selected, its domain is split in two non-empty parts and one of them is non-deterministically chosen as its new domain.

In reality, we do not need to find complete assignments to all variables, rather we need only restrict their domains enough to reach a solved form. Duchier and Niehren (2000) prove that it is sufficient to adapt the distribution rules given in Figure 2 as follows. For each $x{:}f(x_1, \ldots, x_n) \in \phi$ and $y \in V_\phi$:

$$
R_{xy} \in \{=, \lhd^+\} \quad \longrightarrow \quad R_{x_i y} \in \{=, \lhd^+\} \;\vee\; R_{x_i y} \notin \{=, \lhd^+\}
$$

and for all $x, y \in V_\phi$:

$$R_{xy} \neq \bot \quad \longrightarrow \quad R_{xy} \in \{=, \lhd^+\} \ \vee \ R_{xy} \notin \{=, \lhd^+\}$$

In either case, a distribution step splits the domain of a R_{xy} into a subset of $\{=, \lhd^+\}$ and a subset of its complement.

7.6. Incremental processing

We now indicate how the encoding and constraint-based method described above can be adapted to support the incremental processing of descriptions. In this view, processing consists of alternating steps of information acquisition and processing. The description ϕ is not given entirely up front; instead it is incrementally acquired.

Surprisingly enough, this has no impact on our encoding: we need simply accommodate the fact that both ϕ and therefore V_ϕ are only incrementally revealed. This is easily achieved (i) by representing, in the CSP, V_ϕ as a set variable which is merely constrained to contain the node variables which have been revealed so far, (ii) by incrementally producing additional constraints for the CSP as more conjuncts of ϕ and more node variables become available. We omit the details of this procedure, as they should be fairly obvious when looking at the encoding.

At each step we derive all corresponding solved forms. An incremental near-deterministic solver can be obtained with the addition of a preference criterion, such as the *Argument Filling Principle* of Section 5, that allows us to chose one solved form before proceeding to the next step.

8. Conclusion

In this paper, we proposed a new application for the constraint-based treatment of descriptions presented in (Duchier and Gardent, 1999) namely, incremental discourse parsing. Specifically, we have argued that, given the appropriate parsing architecture, this constraint-based approach could be tailored to produce the partial structures built during the incremental interpretation of discourse.

Two important questions remain open. In section 3, we suggest using a discourse variant of Kallmeyer's Synchronous Local Tree Description Grammars to produce the descriptions from which the partial structures built during incremental processing are computed. This implies that the descriptions the constraint solver works with are synchronous descriptions. It is a matter for further research how the contraint-based treatment of descriptions presented in section 7 can be extended to deal with synchronisation.

A second open issue concerns the cognitive plausibility of our model. As illustrated in section 5, this model predicts different levels of processing difficulty thereby providing a basis for experimental validation. It would be interesting to test whether the predictions made by the model are empirically correct.

References

Cristea, D. and B. Webber: 1997, 'Expectations in Incremental Discourse Processing'. In: *Proceedings of ACL*.

Duchier, D. and C. Gardent: 1999, 'A Constraint-Based Treatment of Descriptions'. In: H. Bunt and E. Thijsse (eds.): *Third International Workshop on Computational Semantics (IWCS-3)*. Tilburg, NL, pp. 71–85.

Duchier, D. and J. Niehren: 2000, 'Dominance Constraints with Set Operators'. In: *Proceedings of the First International Conference on Computational Logic (CL2000)*. Springer.

Duchier, D. and S. Thater: 1999, 'Parsing with Tree Descriptions: a constraint-based approach'. In: *Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP'99)*. Las Cruces, New Mexico, pp. 17–32.

Egg, M., A. Koller, and J. Niehren: 2000, 'The Constraint Language for Lambda Structures'. Technical report, Universität des Saarlandes, Programming Systems Lab. Submitted. Available at `http://www.ps.uni-sb.de/Papers/abstracts/clls2000.html`.

Egg, M., J. Niehren, P. Ruhrberg, and F. Xu: 1998, 'Constraints over lambda-structures in semantic underspecification'. In: *Proceedings of ACL/COLING '98*. Montreal, Canada.

Gardent, C. and B. Webber: 1998, 'Describing Discourse Semantics'. In: *Proceedings of the 4th TAG+ Workshop*. University of Pennsylvania, Philadelphia.

Gorrell, P.: 1995, *Syntax and Parsing*. Cambridge University Press.

Hitzeman, J., M. Moens, and C. grover: 1995, 'Algorithms for analysing the temporal structure of discourse'. In: *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*. Dublin, pp. 253–260.

Kallmeyer, L.: 1998, 'Tree Description Grammars and Underspecified Representations'. Ph.D. thesis, Neuphilologishen Facultät, Universität Tübingen.

Kallmeyer, L.: 1999, 'Synchronous Local TDGs and Scope Ambiguities'. In: G. Bouma, E. Hinrichs, G. Kruijff, and R. Oehrle (eds.): *Constraints and Resources in Natural Language Syntax and Semantics*. CSLI, pp. 245–262.

Koller, A., J. Niehren, and R. Treinen: 1998, 'Dominance Constraints: Algorithms and Complexity'. In: *Third International Conference on Logical Aspects of Computational Linguistics*. Grenoble, France. Extended abstract.

M.P.Marcus, D. Hindle, and M.M.Fleck: 1983, 'Talking about Talking about Trees'. In: *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*. Cambridge, MA.

Pritchett, B.: 1992, *Grammatical Competence and Parsing Performance*. Chicago: University of Chicago Press.

Rambow, O., K. Vijay-Shanker, and D. Weir: 1995, 'D-Tree Grammars'. In: *Proceedings of ACL'95*. MIT, Cambridge, pp. 151–158.

Shieber, S. and Y. Schabes: 1990, 'Synchronous Tree Adjoining Grammars'. In: *Proceedings of the 13th International Conference on Computational Linguistics*. Helsinki.

Sturt, P. and M. Crocker: 1996, 'Monotonic syntactic processing: a cross linguistic study of attachment and reanalysis'. *Language and Cognitive Processes* **11**(5), 449–494.

Webber, B. and A. Joshi: 1998, 'Anchoring a lexicalised tree adjoining grammar for discourse'. In: *COLING/ACL Workshop on Discourse Relations and Discourse Markers*. Montreal, pp. 86–92.

Webber, B., A. Knott, M. Stone, and A. Joshi: 1999, 'Discourse relations: a structural and presuppositional account using lexicalised TAG'. In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics*. Maryland.