Context-Free Representational Underspecification for NLG

Anja Belz

November 2004

Contents

1 Introduction and Research Context

The purpose of the COGENT Project[1] is to look at issues in generic (wide-coverage and reusable) surface generation. Central to generic generation is the issue of nondeterminism, i.e. multiple outputs for the same input, and how to control it. Nondeterminism arises from three main sources in natural language generation[2]: (i) *wide syntactic and lexical coverage*: the wider the coverage of grammar and lexicon, the more word strings can be generated from the same semantic representation; (ii) *underdetermined inputs*: the less specific the semantic or conceptual representation, the more word strings correspond to it; and (iii) *unconstrained mapping from inputs to realisations*: the fewer constraints (e.g. rule application conditions, intermediate selection processes, probabilities) there are, the more realisations can be generated from an input. Wide coverage and (even extensively) underdetermined semantics can both make an NLG system more generic, because they help make a system more portable and reusable. However, at present no comprehensive methodology for *controlling the nondeterminism*, for deciding between alternatives, exists. It is one of the core aims of COGENT to develop such a methodology.

There are three basic strategies for reducing nondeterminism: (i) increasing the specificity of the inputs, (ii) using specialised (domain-specific) grammars and lexicons, and (iii) constraining the generation process more. Most approaches to surface generation that aim to be generic either require highly specified inputs (in particular wide-coverage realisers such as FUF/SURGE and KPML) with no methods for informed choice among alternative realisations, *or* they permit less specific inputs and have a methodology for selecting among alternative realisations, applied after the generation process is complete (in particular the relatively recent statistical generators, e.g. Nitrogen and its successors). Some approaches to NLG use heuristics to prune the generation space on the fly, e.g. [Varges and Mellish, 2001].

In all approaches, the *degree of specificity of the inputs is fixed*. This is a problem because, on the one hand, a high level of specificity may force the module creating the inputs to make default or random choices. On the other hand, lowering the required level of specificity means increasing the number of alternative realisations that can be generated from any input (e.g. in Nitrogen, typically trillions of alternatives have to be considered, according to [Langkilde, 2000]).

The basis for the approach to generic surface generation we are developing in COGENT is to enable both (i) flexibility in the degree of specificity of the inputs *and* (ii) control over the generation process, in the simplest case in the form of selection among alternative realisations.

The key to control over the specificity of surface generator inputs is to encode the inputs in a representation formalism that is highly underspecifiable. The basic idea is simple: the deep generator creates underspecified semantic representations (USRs)[3], and the first thing that the surface generator does is expand these to the corresponding set of fully specified semantic representations (SSRs). Lower specificity translates into higher underspecification. The more underspecified the USRs, the larger the set of realisations that can be generated from them (given the same grammar and lexicon).

An important consequence is that the deep generator can choose how specific it wants to be, and can avoid making default or random choices altogether. The inputs can be specific where information is available to make decisions, and where there is not, decisions can be left to a later stage in the generation process, when either enough information becomes available to make the remaining decisions (e.g. from context), or dedicated selection modules can make an informed choice (e.g. based on domain likelihood). This in turn makes the surface generator more easily reusable, because a wider range of different types of deep generation modules will be able to produce inputs from which it can generate.

Figure 1 is a diagram of the architecture of the COGENT generation system we are currently building as a research platform. The system has three stages: expansion from underspecified to fully specified representations, surface generation and selection.

We are using an existing surface realiser, LKB/Lingo developed by Copestake et al. [Copestake et al., 1999], and this choice has partly determined the architecture of the system, in that it has a two-stage surface generation process.

We are considering various selection mechanisms, including statistical language models and sets of stylistic filters. Control over such mechanisms (by the user or application environment) will take the form of a set of parameters that can be set for different domains, applications and generation contexts. E.g. a language model would be trained on domain corpora, while style filters can be configured and weighted in different ways for different domains.

In future work, we will look in detail at control over the generation process and selection among alternative realisations. In the current project phase, we are focussing on control over input specificity, and the purpose of this report is to present *Context-Free Representational Underspecification* (CRU).

CRU is a very general framework for defining underspecifiable representation formalisms, encoded as expansion

[1] A joint EPSRC project at Brighton and Sussex Universities, August 2003–January 2007, EPSRC grants GR/S24480/01 (Brighton) and GR/S24497/01 (Sussex).

[2] Assuming a fixed mapping from semantic representations to realisations.

[3] Using the term "semantic representation" in a loose sense: a meaning representation that may be specific with respect to lexical and syntactic properties.

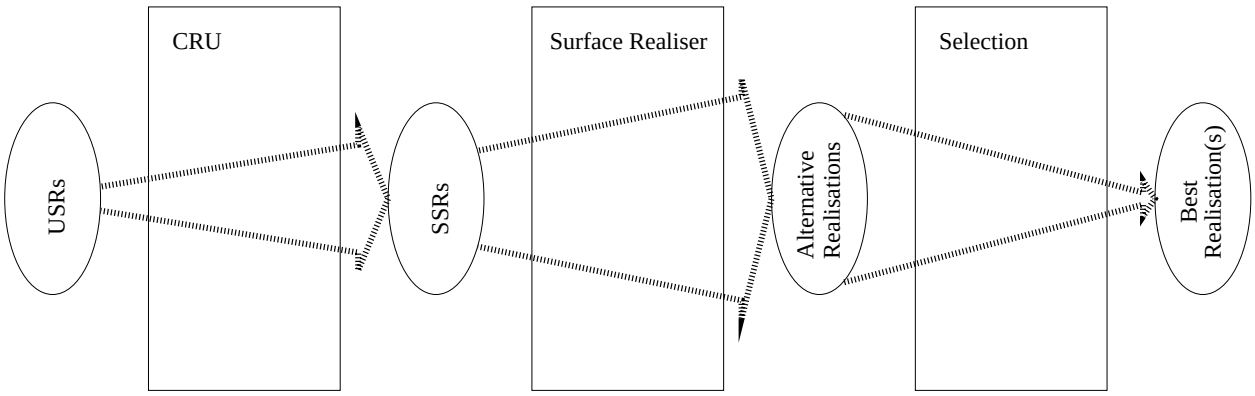


Figure 1: Architecture of COGENT generation platform.

rule grammars. In the context of COGENT, the idea is that a CRU grammar, defined on top of some given semantic representation formalism (SRF), defines an *underspecifiable* semantic representation formalism (USRF). The CRU grammar defines both (i) the different ways in which semantic representations (SRs) may be underspecified, and (ii) the mapping from underspecified to fully specified SRs. The CRU grammar encodes the underspecifiable meaning representation language for the outputs created by deep generation module. If these outputs are underspecified, then the expansion rules of the CRU grammars are used to expand them to all possible fully specified meaning representations which are then realised syntactically and lexically.

Underspecification in meaning representation has a long history, going back at least to the early 1970s. However, research has focussed predominantly, if not exclusively, on using it to represent ambiguity between several possible analyses of the same word string. With Context-free Representational Underspecification (CRU), this report proposes the first systematic use of underspecification for NLG.

This report describes CRU in detail and discusses its motivation and research context in some depth. It is organised as follows. Section 2 provides an overview of underspecification research and the three types of underspecification technique that have predominantly been used. Section 3 talks about the tree-shaped generation spaces defined by NLG systems, in which the nodes correspond to choice points where decisions between alternatives are made, and in which individual generation processes are a set of paths from root to leaf nodes. This discussion provides the basis for outlining an approach to underspecification for NLG where choice points are made representationally explicit in the form of underspecified representations. The main part of this report is Section 4 which describes CRU in detail, in terms of formal definitions and illustrating examples. Section 5 picks up some points from Section 4 and investigates them in more detail. Directions for future research are discussed in Section 6.

2 Underspecification in Semantic Representation

2.1 Overview of underspecification research

Research on developing underspecification techniques has — probably exclusively — focussed on NLU, in the context of which it has been motivated in a number of different ways (summaries are provided in [König and Reyle, 1999, Pinkal, 1999, Bunt, 2003]): use of underspecified representations can increase processing efficiency by making it possible to postpone disambiguation decisions, or to avoid the disambiguation bottleneck altogether where disambiguation is too fine-grained, irrelevant or simply impossible in the current context. Underspecification has been advocated as a way to make processing robust, and so as a deep alternative (or complement) to flat heuristic and statistical language understanding [Pinkal, 1999, p. 40]. The theoretical-linguistic argument has been made that underspecified representations simply are the most appropriate representations of context-independent analyses of ambiguous word strings [Bunt and Muskens, 1999, Ahn et al., 1994, Pulman, 2000, p. 21].

Another motivation comes from psycholinguistics: it has become fairly standard to assume that people do not disambiguate exhaustively when processing ambiguous sentences, and some evidence has been provided for this. E.g. [Traxler et al., 1998] note that people read the following ambiguous sentence (1) faster than the syntactically identical, but unambiguous sentence (2):

(1) The son of the driver that had the mustache was pretty cool.
(2) The car of the driver that had the mustache was pretty cool.

One possible explanation is that "the syntactic representation in the ambiguous case remains underspecified" [Ferreira et al., 2002, p. 20].

The existing underspecification literature looks at phenomena of ambiguity, vagueness and incompleteness in the word string, where underspecified representations are used to encode analyses of ambiguous, vague or incomplete word strings. [Bunt, 2003, p. 42] has an overview of phenomena that have been accounted for by underspecification, including lexical ambiguity, structural semantic ambiguity (quantifier scope and distributivity, nominal compounding), syntactic ambiguity, discourse-level ambiguity (ellipsis, short answers, anaphora, cataphora), and incomplete information (unknown words, partially recognised speech).

While theoretical-linguistic motivations are cited, the driving force behind the development of underspecification techniques has been practical and application-oriented considerations. Probably the two earliest examples of underspecification are of quantifier scope in the LUNAR system [Woods et al., 1972, Woods, 1978], and of quantifier scope and word senses in PHLIQA [Bronnenberg et al., 1979]. Many other systems have used underspecification since, e.g. the TENDUM system [Bunt et al., 1984] had meta-constants for count/mass readings as well as quantifier storage. The Core Language Engine used QLF which achieves underspecification — e.g. of pronouns and quantifier scope — uniformly by meta-variables and indices [Alshawi, 1990]. [Pulman, 2000, pp. 26-29] developed a syntactically and semantically improved version of QLF. ULF, a formalism somewhat related to CLE-QLF (but using HPSG and Type Theory), was developed for the PLUS system [Geurts and Rentier, 1991, Kievit, 1994], and also used in DenK, a generic multimodal user interface [Kievit et al., 2001]. The dialogue systems SPICOS I and II [Niedermair, 1987, Niedermair, 1992] used quantifier storage, while MRS [Copestake et al., 1999] and UDRT [Reyle, 1993] were both used in Verbmobil to structurally underspecify scope.

2.2 Underspecification techniques

There are two basic categories of underspecification techniques: (i) structural underspecification: incompletely specifying the way in which smaller expressions combine to form larger expressions; and (ii) ambiguous terms, where one term 'stands for' a set of terms.

The one standard technique for all kinds of structural underspecification is what has variously been called **holes, handles**, and **labels** (Section 2.2.1 below). Two different types of ambiguous terms have been used: **meta-constants**, the main mechanism for underspecifying atomic expressions (Section 2.2.2), and **meta-variables** which are mostly used for cases where the variants themselves depend on context (Section 2.2.3). There is not a complete separation between the ambiguity phenomena that have been accounted for with each of these mechanisms, but between them, it has been claimed [Bunt, 2003, p. 42], they cover pretty much all the word-string level ambiguities in NL.

2.2.1 Structural underspecification: Holes, handles and labels

This mechanism has been used widely in underspecification research where it first appears in exactly this form in Bos's 'Hole Semantics' [Bos, 1995], although Cooper Storage [Cooper, 1983] and UDRT [Reyle, 1993] anticipate much of the mechanism. Various terms have been used to describe the mechanism, but here I will use *pointers* and *labels* as follows. Instead of a syntactically nested representation such as (1) below, (semantically) embedded expressions are written in a syntactically flat representation such as (2), where pointers and labels with the same index match.

(1) $node_0(leaf_1, node_2(leaf_3, leaf_4))$

(2) $label_0{:}node_0(pointer_1, pointer_2), label_1{:}leaf_1, label_2{:}node_2(pointer_3, pointer_4), label_3{:}leaf_3, label_4{:}leaf_4$

Underspecification can then be achieved by replacing indices (which amounts to cutting connections between pointers and labels), and constraining the ways in which pointers and labels can be linked once more using some constraint language (which amounts to subexpressions being (re)embedded). In Hole Semantics, the connections are achieved by variable instantiation (labels are constants, and pointers are variables ranging over labels), and in MRS they are achieved by co-indexation.

A representation formalism capable of underspecifying structure can be seen as defining two languages and the mapping betweeen them: the language of fully specified expressions L (where every pointer is connected with some label), the language of underspecified expressions U, and the mapping from U to L. In principle, and given a suitably expressive constraint language, any subset of L can be represented by a single expression in U. However, the tendency has been to attempt to restrict in linguistically meaningful ways the set of subsets of L that can be represented by a single expression. In particular, for underspecification of quantifier scope, researchers have developed methods for restricting the subsets (sets of readings) that can be represented in the formalism to the plausible readings, examples including Hobbs and Shieber's algorithm [Hobbs and Shieber, 1987] and MRS [Copestake et al., 1999].

Some formalisms end up being too restrictive, e.g. Bunt et al.'s TENDUM system and MRS with QEQ-constraints cannot do partial scope specification, as required e.g. where word order partially disambiguates a sentence with quantifiers (reducing the number of possible readings). MRS with QEQ-constraints as used in the Lingo parser runs into particular problems with the interaction between quantifier and adverbial scope. As an example consider the following two sentences and two readings (in a highly simplified pseudo-semantic representation):

1a. *some dogs only bark*	2a. some(dogs,only(bark))
1b. *only some dogs bark*	2b. only(some(dogs,bark))
	2c. some(only(dogs),bark)

(2c) is a valid reading for both (1a) and (1b), but (2a) is valid only for (1a), and (2b) is valid only for (1b). However, only the underspecified representation that permits just readings (2a) and (2b) is possible in LKB-Lingo MRS, and is the analysis assigned to both (1a) and (1b).

2.2.2 Meta-constants

Meta-constants are a simple generic mechanism that has been widely used in underspecification, where a single atomic term (the meta-constant) is mapped to a set of atomic terms (the object constants). Some form of lookup table holds information about which meta-constants correspond to which object constants. One of the oldest examples is from [Bronnenberg et al., 1979] who used meta-constants for homonymous and polysemous words: e.g. the meta-constant `AMERICAN` which was domain-specifically expanded to individual senses such as 'manufactured in America' and 'located in America'.

Meta-constants have also been used to account for other types of variation, e.g. quantifier distributivity.

2.2.3 Meta-variables

What the above two underspecification techniques have in common is that they are used only for cases where the complete (or perhaps, maximal) set of possible resolutions or readings can be stated independently of the context (although selection of a subset of readings may be context-dependent). There is, however, a whole range of ambiguity phenomena where the set of possible readings, while possibly inferrable by some general function from the context, cannot be stated generally and independently of the context. Examples include unresolved anaphoric and deictic references. Such phenomena are what meta-variables in underspecification are intended for.

The term 'meta-variable' has come to refer to a "placeholding device" [Kempson, 2003, p. 304], which is, at a later stage of analysis, linked by some mechanism to one interpretation as determined by a function over the context. For example, in CLE-QLF, pronouns are represented by underspecifiable terms that have (in addition to two generic arguments) an argument of type 'restriction' which may specify e.g. gender, and an argument of type 'meta-variable' which is, in the resolved forms, instantiated to the contextually preferred referent [Pulman, 2000, p. 522].

2.3 Underspecification for NLU and NLG

The field of semantic underspecification has tended to concern itself exclusively with phenomena of ambiguity in the word string: word sense, syntactic and semantic scope, reference, etc. Such ambiguity arises during analysis of the word string when there are decisions about the correct analysis that cannot be made on the basis of the available information. The idea in using underspecification is then to have a representation of the analysis that is as ambiguous as the word string. For the construction of such representations no knowledge about the full set of variants, or alternative readings, is required. Furthermore, such representations have a use even if they are not, or cannot, be mapped to their corresponding variants at all.

In NLG too, underspecification is potentially useful for situations where decisions cannot be made. However, while in NLU decisions are between different meanings of the same word string, in NLG, decisions are between different word strings expressing the same meaning. For example, in NLU, it would be useful to have an expression such as `BANK(X)` that is underspecified between the lexical items *bank* in the river bank sense, and *bank* in the financial institution sense, but in NLG, such an expression would be literally useless. This is because the NLG task is to map from a given known meaning (if the meaning is known so is the sense of *bank*) to a word string that expresses the meaning. What NLG does have a use for are expressions that are underspecified between 'different ways of saying the same thing', e.g. *automobile*, *car*, *motor*, *banger* for the concept CAR, but it does not make sense in NLG to be able to not distinguish between homonyms (such as the two senses of 'bank'), to underspecify, in other words, between 'identical ways of saying different things'.

In both NLG and NLU, underspecified expressions provide a way of postponing or avoiding decisions. Ideally, such expressions can be specific where information is available to make decisions, and where there is not, decisions can be left to a later stage in the generation or analysis process, when either enough information becomes available to

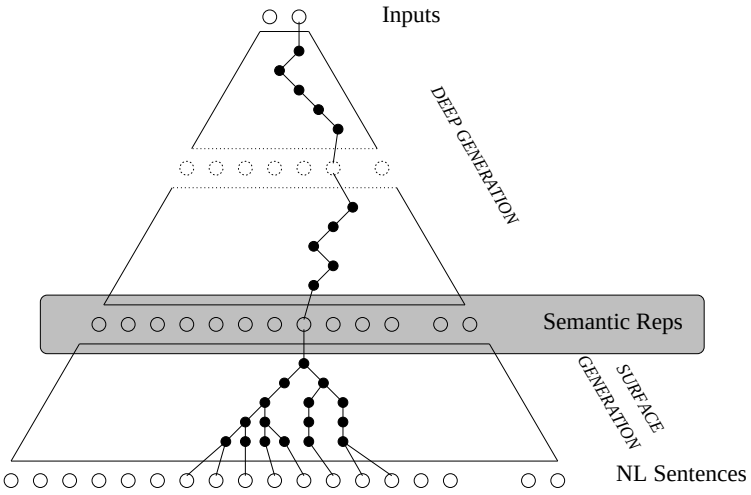


Figure 2: Generation space as decision tree.

make the remaining decisions (e.g. from context), or dedicated selection modules can make an informed choice (e.g. based on domain likelihood).

3 Underspecifiable Representation for NLG: Towards a Formalism

This section starts by taking a high-level look at the entire generation process (Section 3.1). *Generation spaces* are described in terms of decision trees where the nodes correspond to choice points at which decisions between alternatives are made, and where individual generation processes can be seen as a set of paths from root to leaf nodes.

The next section (3.2) discusses the *representational spaces* defined by underspecifiable and other representation languages. A type of tree-shaped representation space is described where the leaf nodes are the fully specified representations and the other nodes are underspecified precisely between the leaf nodes they dominate. Such a representational space can be used to explicitly model the decision tree underlying the generation process.

Section 3.3 argues that this is precisely the way that underspecification should be used in NLG: modelling the decision tree underlying the generation space as a whole. The section finishes with the outline of a proposal to implement an underspecifiable semantic representation formalism (USRF) for NLG as a generic framework with a nongeneric component that have much the same respective roles as parser and grammar in NLU.

3.1 Generation space and intermediate representations

While there is a lot of disagreement in NLG about appropriate architectures, interfaces and the representations that are constructed, there is broad agreement that the NLG process passes through three phases, roughly corresponding to three levels of representation: (1) conceptual, (2) semantic, and (3) syntactic and lexical (even [Reiter, 1994] and RAGS [Cahill et al., 2001] agree in these general terms). NLG systems do not necessarily have three separate modules corresponding to the three phases (in the sense that there are well-defined interfaces with associated representation languages), but most have at least one distinct level of representation that is intermediate between conceptual representation and NL strings (a semantic level, in the broad sense). I will refer to the phase before the semantic interface as *deep generation* (determining what to say), and that following it as *surface generation* (determining how to say it).

Figure 2 is an illustration of a generic three-phase generation space (for NLG systems with two or three distinct modules). Seen as a whole, the space consists of all possible individual generation processes in a system, i.e. all the different ways you can get from an input to an output. The unfilled circles are intermediate representations passed from one module to the next, and the filled circles are choice points at which decisions between alternatives are made. Type of decision and corresponding set of alternatives depends on the stage of generation, e.g. at an early stage a decision may be the selection of a particular dialogue act, later it may be whether to pronominalise or not. Internally, a module will tend to maintain data structures that are updated at each choice point, depending on the decision made. There will tend to be several ways of getting to the same (intermediate) representation, so the representations and decisions represented by the circles are not unique.

The idea is that any generation space can be equivalently represented as a decision tree. From the input to the NL output a chain of decisions are made that determine the paths that are taken through the generation space, and ultimately the output set of NL strings. In an actual NLG system, decisions can be implicit or explicit, based on properties of intermediate data structures or on generation context.

As an example consider the following intermediate, conceptual-level representation from [Reiter and Dale, 1997, p. +9]:

```
+-                                                -+
| message-id:msg02                                 |
| relation: DEPARTURE                              |
|            +-                                -+  |
|            | departing-entity: CALEDONIAN-EXPRESS |  |
| arguments: | departure-location: ABERDEEN         |  |
|            | departure-time: 1000                 |  |
|            +-                                -+  |
+-                                                -+
```

Suppose in a given NLG system, the set of realisations that can be generated from this representation includes *The Caledonian Express departs from Aberdeen at 10am, The train departs from Aberdeen at 10am, It leaves Aberdeen at 10am*, and *It leaves here in 5 minutes*. At some point between content representation and realisation, a large number of decisions have to be made. Some of these are conventionally dealt with by a referring expressions module, some by a process of lexicalisation, some by the grammar. An example of an atomic decision (which would be a single decision node in the graph in Figure 2) is that between the verbs 'depart from' and 'leave'. Depending on the system, the decision may be implicit in the conceptual-level representation, so that only word strings containing, say, 'leaves' are accessible from it, or there may be a point in the generation process where the two alternatives are explicitly considered and a decision is made based on some information external to the representation. A third option is that both paths are pursued and the decision is delegated to an external procedure so that if and when made it will cut off some paths.

If it is possible to pursue multiple paths, then the system has nondeterminism. The tree of paths shown in Figure 2 is a typical scenario in a generation system: in the conceptual and semantic space, decisions tend to (explicitly or implicitly) lead down a single path, but the grammar and/or lexicon has some degree of nondeterminism. Nondeterminism is handled in a variety of ways, most commonly by some form of post-selection, which selects the first n, a random set of n, or in the more recent statistical NLG, the best n realisations.

Many NLG systems use a separate module at least for surface generation, and if a surface generator is to be in any way generic (reusable across different applications and domains), it is necessarily separate from the rest of the system. Crucially, a separate surface generator means that in every generation process, representations have to reach the same degree of specificity at the same stage. This is often not a good idea, e.g. in generation in machine translation: if a piece of information required by the target language is not provided locally by the source language, then instead of forcing a local decision, it would make more sense to postpone the decision between the possible alternatives until either it can be made on the basis of additional, nonlocal information, or dedicated selection modules can make an informed choice[4].

3.2 Underspecifiable representation spaces

Figure 3 shows an example of part of the generation space of Figure 2 — the semantic representation level (in the shaded box) and some of the decision tree above it — in more detail. The word strings below the semantic representation nodes are meant to indicate approximate the meaning of the semantic representations. Lexicalisation might at a later stage map "12am" to *noon*, *noontime*, *midday* among others, and the grammar might permit e.g. *at 12am it departs* as well as *it departs at 12am*.

Ordinarily in NLG, some semantic representation language is used that encodes the nodes in the shaded box, but no representation language is used for the nodes above it. If such a language were used, its representations would have to encode the same information that the position of a node in the decision tree encodes: they would have to be specific with respect to the decisions already taken, and contain information about all possible future decisions. Or, looked at another way, they would have to denote precisely the set of realisations that can be generated from that point, and thus the entire subtree rooted at that point.

This is precisely what underspecified representation is about. E.g. when quantifier scope is underspecified in NLU, the underspecified representation encodes the set of possible scopings. Similarly, an underspecifiable representation formalism can be created that encodes decision trees such as the one in Figure 3: the nodes and branches

[4]Example: when translating pronouns into a language which distinguishes gender in non-person nouns, the decision between the target language genders cannot always be made at the sentence level.

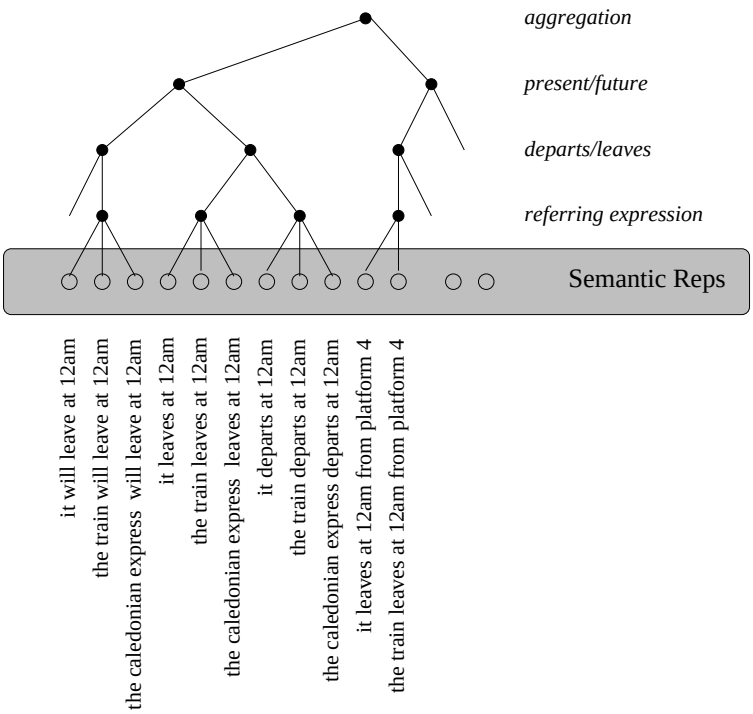


Figure 3: Underspecifiable representation space.

then correspond to the underspecifiable representation space, while each individual node corresponds to an explicit semantic representation that is underspecified between the (fully specified) representations corresponding to the leaves it dominates.

With appropriately defined representations, decision space and representational space can overlap or even be identical: it is in principle possible to explicitly represent the generation space underlying any generation system as an underspecifiable representation space.

3.3 Surface generator input language as modelling a generation space decision tree

It was sketched in the preceding sections in general terms how a USRF can be used to model part or even all of the generation space. The way we want to use it in the COGENT project is somewhat more focussed than that: the aim is to use such a USRF to create an underspecifiable and therefore flexible input language for a generic surface generator. In terms of Figure 2, we want to create an underspecifiable representation space that encodes the semantic representation level (shaded box) and part of the generation tree above it, so that the deep generator can create any of these underspecified or fully specified representations to pass on to the surface generator.

Such a representation space can be seen as consisting of three distinct components: (i) the **object language** R of fully specified representations, (ii) some encoding of the different ways in which expressions in R can be underspecified, defining a **meta-language** M^R of underspecified representations, and (iii) **expansion mechanism(s)**: some way of deriving the subset of R that corresponds to an arbitrary element of M^R, i.e. some function $f(m) = R', m \in M^R, R' \subseteq R$.

Every expansion mechanism can be combined with a range of different R. For every R many different M^R can be defined. This makes it natural to base the design of the USRF formalism on a distinction between generic and nongeneric components: to define a framework which has generic expansion mechanisms but but allows different R to be defined and for every R, allows different M^R to be 'plugged in'.

The following section introduces a framework based on context-free grammars that allows R and M^R to be varied freely, while keeping the expansion function fixed.

4 CRU: Representational Underspecification in a Context-Free Framework

This section describes CRU (Context-free Representational Underspecification), a context-free formalism for underspecifying a wide range of different types of representation languages, including languages for structured semantic

representation. The context-free setting has a range of advantages including that (i) any number of different underspecification spaces can be defined simply by writing new sets of context-free expansion rules, and (ii) generic underspecification techniques can be defined in terms of computationally inexpensive context-free parsing, generation and operations on derivations.

As outlined in the preceding section, the space of all possible decisions that a generation system can make in the course of increasingly specifying the word string can be modelled by a decision tree. The basic idea in using the CRU formalism for NLG is to model all or part of this decision space with what is essentially a context-free grammar, such that the decision tree is encoded by an individual CRU grammar, and decision nodes and the branches rooted at them are encoded by nonterminals and sets of production rules with the same nonterminal in the left-hand side.

This approach provides a way of writing down for individual linguistic points of variation the set of possible variants in the form of a set of expansion rules, e.g. for lexical variation the set of paraphrases realising the same concept. Because CRU is context-free, writing down such a set of rules means specifying that the decision between the alternatives, the process by which the expression in the LHS becomes fully specified, is independent from the context. This means taking a very systematic view of the decision space underlying NLG or NLU: it must be defined entirely in terms of a tree of decisions that depend only on decisions already taken.

The formal description of CRU starts in Section 4.1 with the three basic context-free definitions (4.1.1), followed by the definitions of the basic CRU formalism (4.1.2), a simple worked example (4.1.3), and a summary of the advantages of using a context-free formalism (4.1.5).

Section 4.2 describes how (non)terminals with argument structure are used in CRU, and explains how this enables structured languages to be represented and underspecified.

It is argued that CRU is well suited for underspecification of semantic representations, and Section 4.3 gives one possible set of notational conventions and argument types for defining a USRF for NLG in CRU. This is illustrated by defining and discussing a simple example CRU-grammar for generating train departure time expressions.

4.1 Basic CRU Formalism

4.1.1 Basic context-free definitions

The following are the three standard definitions for context-free grammars, given here in their entirety because definitions and explanations in the remainder of this section will refer to their details. The definitions are essentially the same as in [Hopcroft and Ullman, 1979, p. 80–81].

Definition 1 Context-Free Grammar (CFG)

A context-free grammar (CFG) is a 4-tuple (W, N, S, R), where W is a set of terminal symbols, N is a set of nonterminal symbols, $S \in N$ is the start symbol, and R is a set of production rules, where each rule is of the form $n \rightarrow \alpha$, $n \in N$, $\alpha \in (W \cup N)^*$. W and N are disjoint.

Definition 2 Derivation

Given a CFG $G = (W, N, S, R)$, $\alpha A \gamma$ **directly derives** $\alpha\beta\gamma$ in grammar G, denoted $\alpha A \gamma \underset{G}{\Longrightarrow} \alpha\beta\gamma$, if

1. $A \rightarrow \beta$ is a production of R, and
2. α and γ are any strings in $(W \cup N)^*$.

α_1 **derives** α_m in grammar G, denoted $\alpha_1 \overset{*}{\underset{G}{\Longrightarrow}} \alpha_m$, if

1. $\alpha_1, \alpha_2, ..., \alpha_m$ are strings in $(W \cup N)^*$, $m \geq 1$, and
2. $\alpha_1 \underset{G}{\Longrightarrow} \alpha_2, \quad \alpha_2 \underset{G}{\Longrightarrow} \alpha_3, \quad ... \ , \quad \alpha_{m-1} \underset{G}{\Longrightarrow} \alpha_m$.

Definition 3 Language of a CFG

The **language** of a CFG $G = (W, N, S, R)$, denoted $L(G)$, is $\{w \big| w \in W^* \text{ and } S \overset{*}{\underset{G}{\Longrightarrow}} w\}$.

Note that under these definitions $\alpha \overset{*}{\underset{G}{\Longrightarrow}} \alpha, \alpha \in (W \cup N)^*$ is possible, because a nonterminal may directly or indirectly derive itself.

Apart from the above definition, the remainder of this report will refer to the standard CFG concepts of **derivation tree** (aka parse trees) and **sentential form**. A derivation tree is constructed from a derivation by creating internal nodes from the LHS from every rule application used in the derivation, and creating child nodes for every (non)terminal in the corresponding RHS. Every derivation tree corresponds to one or more derivations, but to only one left-most (right-most) derivation, in which it is always the left-most (right-most) nonterminal that is expanded first.

A sentential form under $G = (W, N, S, R)$ is any string $\alpha \in (W \cup N)^*$ that can be derived from S.

4.1.2 CRU terminology

CRU uses context-free technology and applies it in a way that differs from existing uses in some respects. For example, in computer program compilation, CFGs are used to define the set of syntactically correct programs. Here, the CFG is used only to test for membership, and is written only with the aim of correctly defining $L(G)$. In NLP, the most common existing use of CFGs is as phrase structure grammars (PSGs), and here the notion of correct derivation tree (parse tree) is crucial. PSGs are written with the aim of correctly defining two things: (i) $L(G)$, the set of strings licensed by G, and (ii) the sets of derivation trees that G assigns to the strings in $L(G)$.

In neither of these uses of CFGs are grammars written with the aim of correctly defining the set of sentential forms, but this is fundamental in CRU. In CRU, grammars are a strict subset of CFGs, and a CRU grammar G is taken to define a second language, $U(G)$, which consists of all and only the sentential forms under G that derive a string in $L(G)$, but are not themselves in $L(G)$:

Definition 4 Basic CRU-grammar

The set of all **basic CRU-grammars** is the set of all CFGs $G = (W, N, S, R)$, such that R does not contain productions of the form $A \rightarrow \epsilon$, where ϵ is the empty string.

Definition 5 Underspecified language of a CRU-grammar

The **underspecified language** of a CRU-grammar G, denoted $U(G)$, is the set of all sentential forms α under G, such that $\alpha \underset{G}{\overset{*}{\Rightarrow}} \beta, \alpha \neq \beta, \beta \in L(G)$.

The elements of $U(G)$ are called the **underspecified expressions** under G.

The elements of $L(G)$ are called the **fully specified expressions** under G.

The **underspecifies** relation, is defined between the elements of $U(G)$ and $L(G)$.

Definition 6 Underspecifies

Given a CRU-grammar $G = (W, N, S, R)$, and two strings $\alpha, \beta \in (W \cup N)^+$, α **underspecifies** β, denoted $\alpha \underset{G}{\overset{<}{\Rightarrow}} \beta$, if and only if

1. α and β are sentential forms under G,
2. $\alpha \neq \beta$, and
3. $\beta \underset{G}{\overset{*}{\Rightarrow}} \gamma$, for some $\gamma \in L(G)$.

This says that for $\alpha \underset{G}{\overset{<}{\Rightarrow}} \beta$ to be true, α must be derivable from the top symbol S, and β must be in $L(G)$ or derive a string in $L(G)$. Note that in contrast to the derives relation, the underspecifies relation is not reflexive, and requires some fully specified expression to be derivable from β.

For convenience, two further terms are defined. A CRU-grammar assigns to every fully specified string under G, that is to say, every element β of $L(G)$, a set of expressions that underspecify it:

Definition 7 Underspecification set

Given a CRU-grammar G, the **underspecification set**, or **U-set**, of a string β in $L(G)$, denoted $U(\beta)$, is $\{\alpha \mid \alpha \underset{G}{\overset{<}{\Rightarrow}} \beta\}$.

Conversely, every underspecified representation α has a set of consistent fully specified representations which is a subset of $L(G)$:

Definition 8 Expansion set

Given a CRU-grammar G, the **expansion set**, or **E-set**, of an underspecified expression α under G, denoted $L(\alpha)$, is $\{\beta \mid \alpha \underset{G}{\overset{<}{\Rightarrow}} \beta,\ \beta \in W^+\}$.

Note that the above definitions do not care about the process by which derivations are constructed, or how many ways there are of constructing the same derivation, since they only refer to strings.

4.1.3 Simple example

This section presents a very simple example of a CRU grammar, its set of fully specified strings, and a derivation tree for one of the strings together with its corresponding underspecification set. The grammar encodes a small generation space for the example from [Reiter and Dale, 1997, p. 65] that was used in Section 3.2. For convenience, the example had the following conceptual-level message representation:

```
+-                                                -+
| message-id:msg02                                 |
| relation: DEPARTURE                              |
|           +-                                -+  |
|           | departing-entity: CALEDONIAN-EXPRESS |  |
| arguments: | departure-location: ABERDEEN        |  |
|           | departure-time: 1000                 |  |
|           +-                                -+  |
+-                                                -+
```

The following CRU-grammar defines a variety of realisations and underspecified realisations for this content representation[5]: $G = (W, N, S, R)$, with

W = { the, Caledonian, Express, train, it, leaves, departs, Aberdeen, from, at, 10am, 10, o'clock, in, the, morning },
N = { DEPARTING_EVENT, TRAIN, DEPART, LOCATION, TIME },
S = DEPARTING_EVENT,

and the following rule set R:

CRU-grammar 1

```
DEPARTING_EVENT --> TRAIN DEPART
TRAIN --> the Caledonian Express
TRAIN --> the train
TRAIN --> it
DEPART --> leaves LOCATION TIME
DEPART --> departs LOCATION TIME
DEPART --> departs from LOCATION TIME
LOCATION --> Aberdeen
TIME --> at 10am
TIME --> at 10 o'clock in the morning
```

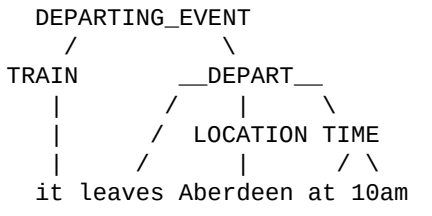
W and N will be omitted in subsequent examples, because, given the convention of representing terminals in lower, nonterminals in upper case, they can be derived from the rule set.

CRU-grammar 1 has a set of fully specified strings $L(G)$ that consists of the following 18 sentences.

```
the Caledonian Express leaves Aberdeen at 10am
the Caledonian Express leaves Aberdeen at 10 o'clock in the morning
the Caledonian Express departs Aberdeen at 10am
the Caledonian Express departs Aberdeen at 10 o'clock in the morning
the Caledonian Express departs from Aberdeen at 10am
the Caledonian Express departs from Aberdeen at 10 o'clock in the morning
the train leaves Aberdeen at 10am
the train leaves Aberdeen at 10 o'clock in the morning
the train departs Aberdeen at 10am
the train departs Aberdeen at 10 o'clock in the morning
the train departs from Aberdeen at 10am
the train departs from Aberdeen at 10 o'clock in the morning
it leaves Aberdeen at 10am
it leaves Aberdeen at 10 o'clock in the morning
it departs Aberdeen at 10am
it departs Aberdeen at 10 o'clock in the morning
it departs from Aberdeen at 10am
it departs from Aberdeen at 10 o'clock in the morning
```

As an example of a derivation tree, the one for the sentence *it leaves Aberdeen at 10am* is shown below. Note that each string has its own set of derivation trees under a CFG.

[5] The notational convention adopted represents nonterminals in upper case, terminals in lower case.



As a visualisation, each horizontal 'cut' through this tree (that does not contain both parent nodes and their child nodes) is an underspecified expression for *it leaves Aberdeen at 10am*. The U-set (underspecification set) for this sentence under CRU-grammar 1 is the following (where line numbers, e.g. `01:` and expansion set sizes, e.g. `(18)` are included for ease of reference only):

```
01: DEPARTING_EVENT (18)
02: TRAIN DEPART (18)
03: TRAIN leaves LOCATION TIME (6)
04: TRAIN leaves LOCATION at 10am (3)
05: TRAIN leaves Aberdeen TIME (6)
06: TRAIN leaves Aberdeen at 10am (3)
07: it DEPART (6)
08: it leaves LOCATION TIME (2)
09: it leaves LOCATION at 10am (1)
10: it leaves Aberdeen TIME (2)
```

CRU-grammar 1 thus allows 10 different ways of underspecifying *it leaves Aberdeen at 10am*. Each picks out a different subset of $L(G)$, the language of fully specified representations generated by the grammar. The start symbol `DEPARTING_EVENT` expands to all strings in $L(G)$, whereas e.g. `it leaves Aberdeen TIME` has an expansion set of only two strings: *it leaves Aberdeen at 10am* and *it leaves Aberdeen at 10 o'clock in the morning*.

The *underspecifies* relation is defined (Definition 6) not only for pairs of one underspecified and one fully specified string, but also for pairs of underspecified strings. So, string `01`, for example, underspecifies all other strings in the above U-set, while `02` underspecifies `03,04,05,06,07,08,09` and `10`, and `08` underspecifies `09` and `10`, etc.

The size of $U(G)$, the underspecified language of the above grammar is much larger than $L(G)$, containing altogether 59 strings. The set of all complete derivations $S \underset{G}{\overset{*}{\Rightarrow}} \alpha, \alpha \in L(G)$ is larger still: there are 144 different derivations, meaning 144 different orders in which decisions can be taken. Or, in terms of the tree representation used in Figures 2 and 3, there are 144 different ways of getting from the root node to a leaf. The following is part of the tree representation (with the tree on its side and the root on the top left) of the generation space encoded by Grammar 1 (the complete tree is included in this report as Appendix B). Fully specified strings (leaf nodes) are preceded by '<<<':

```
DEPARTING_EVENT
-- TRAIN DEPART
   -- the Caledonian Express DEPART
      -- the Caledonian Express leaves LOCATION TIME
         -- the Caledonian Express leaves Aberdeen TIME
            <<< the Caledonian Express leaves Aberdeen at 10am
            <<< the Caledonian Express leaves Aberdeen at 10 o'clock in the morning
         -- the Caledonian Express leaves LOCATION at 10am
            <<< the Caledonian Express leaves Aberdeen at 10am
         -- the Caledonian Express leaves LOCATION at 10 o'clock in the morning
            <<< the Caledonian Express leaves Aberdeen at 10 o'clock in the morning
      -- the Caledonian Express departs LOCATION TIME
         -- the Caledonian Express departs Aberdeen TIME
            <<< the Caledonian Express departs Aberdeen at 10am
            <<< the Caledonian Express departs Aberdeen at 10 o'clock in the morning
         -- the Caledonian Express departs LOCATION at 10am
            <<< the Caledonian Express departs Aberdeen at 10am
         -- the Caledonian Express departs LOCATION at 10 o'clock in the morning
            <<< the Caledonian Express departs Aberdeen at 10 o'clock in the morning
...
```

4.1.4 CRU-grammars are not generators

It may be worth emphasising at this point that a CRU-grammar is not a generator, but simply defines a representational space. In NLG, a CRU-grammar is intended to be interfaced with a deep generator which makes all the decisions

about content that it is able to make. The job of CRU is simply to expand the representations that are output by the deep generator to all permissible expansions that have the degree of specificity requried by the surface realiser, and that are in this sense fully specified. The deep generator makes decisions between alternatives, CRU does not. Where the deep generator needs to have knowledge to base its decisions on, CRU can be dumb.

4.1.5 Advantages of going context-free

There are a number of considerable advantages resulting from the context-freeness of the CRU-formalism:

- CRU is computationally less expensive than existing approaches to underspecification.
- There is not just one underspecified expression for each fully specified expression (e.g. in MRS with QEQ-constraints, there is only one underspecified MRS expression for every fully specified MRS expression) — there can be arbitrarily many underspecified expressions for each fully specified one.
- A CRU-grammar defines the *underspecifies* relation not only for pairs of one underspecified and one fully specified string, but also between the underspecified strings themselves, imposing a hierarchical ordering of increasing specificity.
- It is straightforward to introduce probabilities attached to the rules, e.g. in order to generate the n most likely expansions instead of all; this is useful because it allows domain-specific preferences to be modelled, and it provides a basis for making decisions if that basis is otherwise absent.
- It opens the possibility of automatically constructing or optimising underspecification formalisms directly by a machine learning technique for CFGs, e.g. [Belz, 2002].

4.2 CRU with structured relations

The simple representations that were used in Example 1 are not very useful as semantic representations because they are completely unstructured. This section describes an extension to CRU that makes it possible to have the basic argument structure that is conventionally part and parcel of semantic representation formalisms. Furthermore, the examples of underspecification in previous sections were all of the meta-constant variety (see Section 2.2.2). Below, different types of structural underspecification are considered.

4.2.1 Formal extension to basic CRU

In addition to the atomic symbols of basic CRU, (non)terminals can now have one or more arguments. These CRU (non)terminals are simple terms defined as follows.

Definition 9 Term

Given an alphabet of relation names F, an alphabet of variable names V, a set C of constants, and a function σ that takes elements of V into subsets of C, if $f \in F$ is an n-ary relation with $n \geq 0$, and $b_1, ..., b_n$ are variables and constants in $V \cup C$, then $f(b_1, ..., b_n)$ is a **term**.

Derivation with terms that may have variables for arguments requires the following standard definition of term substitution and unification.

Definition 10 Substitution and unification for CRU-grammars with atomic arguments

A **substitution** θ is a set $\{v_1 \leftarrow c_1, ..., v_m \leftarrow c_m\}$, where the v_i are distinct variables and the c_i are constants.

If t is a term, then $t\theta$ is the term obtained by replacing every occurrence of v in t with c, for all $v \leftarrow c \in \theta$.

Let $T = \{t_1, ..., t_n\}$ be a finite set of terms. A substitution θ is called a **unifying substitution** or simply a **unifier** for T if $t_1\theta = ... = t_n\theta$.

The basic CFG definitions in Section 4.1.1 need to be adapted. Definition 1 is unchanged except that every terminal and nonterminal is now a CRU-term as defined above:

Definition 11 CRU-grammar

A CRU-grammar is a 4-tuple (W, N, S, R), where W is a set of CRU-terms called terminals, N is a set of CRU-terms called nonterminals, $S \in N$ is the start term, and R is a set of production rules, where each rule is of the form $n \rightarrow \alpha$, $n \in N$, $\alpha \in (W \cup N)^+$. W and N are disjoint.

The definition of derivation is adapted as follows. One string can now be derived from another if a matching rule can be found in the grammar and there is a unifying substitution for the variables in the strings and the rule:

Definition 12 Derivation for CRU-grammars with atomic arguments

Given a CRU-grammar $G = (W, N, S, R)$, an alphabet F of relation names, an alphabet V of variables, a set C of constants, a function $\sigma(v) = C^v, v \in V, C^v \subseteq C$, and a substitution $\theta = \{v_1 \leftarrow c_1, ..., v_s \leftarrow c_s\}, s \geq 0, v_i \in V, c_i \in \sigma(v_i)$,
$\alpha\ B_1(b_1^1, b_1^2, ..., b_1^m)\ \gamma$ **directly derives** $\alpha\ B_2(b_2^1, b_2^2, ..., b_2^p)\ B_3(b_3^1, b_3^2, ..., b_3^q)\ ...\ B_n(b_n^1, b_n^2, ..., b_n^r)\ \gamma$, $b_i^j \in (V \cup C)$ in grammar G and under substitution θ, denoted
$\alpha\ B_1(b_1^1, b_1^2, ..., b_1^m)\ \gamma \underset{G,\theta}{\Rightarrow} \alpha\ B_2(b_2^1, b_2^2, ..., b_2^p)\ B_3(b_3^1, b_3^2, ..., b_3^q)\ ...\ B_n(b_n^1, b_n^2, ..., b_n^r)\ \gamma$,
if

1. α and γ are any strings in $(W \cup N)^*$,
2. $B_1(a_1^1, a_1^2, ..., a_1^m) \rightarrow B_2(a_2^1, a_2^2, ..., a_2^p)\ B_3(a_3^1, a_3^2, ..., a_3^q)\ ...\ B_n(a_n^1, a_n^2, ..., a_n^r)$, $a_i^j \in (V \cup C)$ is a production of R, and
3. for all $i, 1 \leq i \leq n$, θ is a unifier for $\{B_i(b_i^1, b_i^2, ..., b_i^j), B_i(a_i^1, a_i^2, ..., a_i^j)\}$.

α_1 **derives** α_m in grammar G and under substitution θ, denoted $\alpha_1 \underset{G,\theta}{\overset{*}{\Rightarrow}} \alpha_m$, if

1. $\alpha_1, \alpha_2, ..., \alpha_m$ are strings in $(W \cup N)^*$, $m \geq 1$, and
2. $\alpha_1 \underset{G,\theta}{\Rightarrow} \alpha_2,\ \ \alpha_2 \underset{G,\theta}{\Rightarrow} \alpha_3,\ \ ...\ ,\ \ \alpha_{m-1} \underset{G,\theta}{\Rightarrow} \alpha_m$.

Since all the other definitions in the section on basic CRU were defined in terms of the definition of derivation, they do not need to be redefined.

Note that the CRU definitions in this section subsume the basic CRU definitions presented earlier, basic CRU-grammars being a special case of CRU-grammars.

4.2.2 Representation of structured relations

Consider a simple example of a structured semantic representation like *exists(x, train(x), departs(x))*. It is clear that atomic arguments like x can directly be encoded under the extension described in the last section, but what about arguments like *train(x)*, i.e. arguments that are themselves structured, vital for the representation of scopal structure?

It is not possible in a context-free framework to have nonterminals generally embeddable within nonterminals, and I wanted to avoid the greater cost of using full (nonatomic) term unification. The alternative is to write syntactically recursive structures as syntactically flat representations that retain their recursive structural meaning (they are syntactic variants of each other), as has become common in semantic representation, especially in the underspecification literature, e.g. in MRS and UDRT (see also Section 2.2.1).

CRU with atomic arguments allows this to be encoded directly. An expression like *exists(x, train(x), departs(x))* can be encoded as the three CRU terms `exists(H0,X,H1,H2)`, `train(H1,X)`, and `departs(H2,X)`, where the first argument is always interpreted as the label of the relation and any other arguments can be pointers to embedded relations — if they match the label of another relation, that relation is interpreted as embedded in the first.

There is one complication: In context-free parsing, the order of constituents matters. That is, a parse only succeeds if the grammar explicitly allows the order that the constituents are presented in.

In syntactically flat representation of recursive structure the order of constituents is not meaningful. The expression `a2(Ha) a1(Ha) b(Hb) ASS_AND(H0,Hb,Hc,Ha) c(Hc)` should mean the same no matter what order the five components are presented in. As implied above, this matters only when parsing expressions, but because a CRU expression has to be derivable from the start symbol, it is relevant for generating both E-Sets and U-Sets.

The solution I am adopting is to impose a default ordering on constituents that are part of a flat representation of recursive structure ('structural constituents' for short): the order of structural constituents must correspond to a depth-first left-to-right traversal of the tree structure represented by the constituents. E.g. the example above has to be written `ASS_AND(H0,Hb,Hc,Ha) b(Hb) c(Hc) a1(Ha) a2(Ha)` to observe this default order. Constituents other than structural constituents, and sets of structural constituents that attach to the same pointer, must observe the order stipulated by the grammar.

This makes it possible to have order-independence for structural constituents (e.g. ordering them before parsing an expression), while allowing the CRU-grammar writer to fix the order of other constituents.

This section is only a description of how representation of structured relations in CRU can be done in principle. More formalised notational conventions for semantic representation with CRU are described in Section 4.3.1 below.

Notational convention for writing expressions with recursive structure: When writing a fully specified expression with recursive structure generated by a CRU grammar G, I will tend to use the syntactically recursive form because it is much more readable. This will be highlighted by use of italics, in contrast to the literal expressions generated by CRU grammars which are in typewriter font.

4.2.3 Underspecification of structured relations

Logical relations provide good examples of relations where structural underspecification is useful, enabling e.g. sets of logically equivalent expressions to be represented by a single, underspecified expression. For example, commutativity of logical *and* can be accounted for directly by underspecifying the order of its two arguments. Consider the following CRU-grammar fragment (again, nonterminals upper, terminals lower case):

```
AND(H0,H1,H2) --> and(H0,H1,H2)
AND(H0,H1,H2) --> and(H0,H2,H1)
```

These rules allow expressions like *AND(blue(x),large(x))* to be written which is underspecified between exactly the two fully specified expressions *and(blue(x),large(x))*, and *and(large(x),blue(x))*.

Note that this commutative definition is entirely general — the argument swapping works regardless of the degree of nesting and regardless of what the arguments are (potentially an infinite set of different possible ones). This is not something that would be ordinarily possible in a context-free setting, and is only made possible by the fact that the arguments are the equivalent of pointer variables that can become connected to any relation or subexpression.

Equivalences can also be defined for other kinds of logical equivalence (ASS_AND = associativity of *and*):

```
IMPLICATION(H,Ha,Hb) --> implies(H,Ha,Hb)
IMPLICATION(H,Ha,Hb) --> OR(H,H1,Hb) not(H1,Ha)

DEMORGAN1(H,Ha,Hb) --> OR(H,Ha,Hb)
DEMORGAN1(H,Ha,Hb) --> not(H,H1) AND(H1,H2,H3) not(H2,Ha) not(H3,Hb)

DEMORGAN2(H,Ha,Hb) --> AND(H,Ha,Hb)
DEMORGAN2(H,Ha,Hb) --> not(H,H1) OR(H1,H2,H3) not(H2,Ha) not(H3,Hb)

ASS_AND(H,Ha,Hb,Hc) --> AND(H,H1,Ha) AND(H1,Hb,Hc)
ASS_AND(H,Ha,Hb,Hc) --> AND(H,H1,Hb) AND(H1,Ha,Hc)
ASS_AND(H,Ha,Hb,Hc) --> AND(H,H1,Hc) AND(H1,Ha,Hb)

OR(H,H1,H2) --> or(H,H1,H2)
OR(H,H1,H2) --> or(H,H2,H1)
```

This is akin to defining a normal form representation and an associated mapping from expressions in the normal form to some other class of expressions[6], only here the mapping is given a separate, explicit representation. The pointer arguments take care of any embedded logic expressions.

Using both the associative and commutative rules for logical *and* above, expressions can be written that completely underspecify the order and bracketting of the *and*-ed constituents. Given the above ASS_AND and AND rules, e.g. *ASS_AND(blue(x),large(x),round(x))* underspecifies between all of the following:

and(and(large(x),round(x)),blue(x))
and(blue(x),and(large(x),round(x)))
and(and(round(x),large(x)),blue(x))
and(blue(x),and(round(x),large(x)))
and(and(blue(x),round(x)),large(x))
and(large(x),and(blue(x),round(x)))
and(and(round(x),blue(x)),large(x))
and(large(x),and(round(x),blue(x)))
and(and(blue(x),large(x)),round(x))
and(round(x),and(blue(x),large(x)))
and(and(large(x),blue(x)),round(x))
and(round(x),and(large(x),blue(x)))

Note that rules can be written in CRU that pick out *any* subset of orderings and bracketings.

The above definition of the associative *and* rules allows complete structural underspecification, but of course it is possible to allow only a subset of embeddings, e.g. just enough to get all the orderings of the adjectives (and no more). The changed rule set, and the set of expansions for *ASS_AND(blue(x),large(x),round(x))* are as follows:

```
ASS_AND(H,Ha,Hb,Hc) --> and(H,H1,Ha) AND(H1,Hb,Hc)
ASS_AND(H,Ha,Hb,Hc) --> and(H,H1,Hb) AND(H1,Ha,Hc)
ASS_AND(H,Ha,Hb,Hc) --> and(H,H1,Hc) AND(H1,Ha,Hb)
```

[6] E.g. defining a conversion procedure from any CFG to Greibach normal form.

and(and(large(x),round(x)),blue(x))
and(and(round(x),large(x)),blue(x))
and(and(blue(x),round(x)),large(x))
and(and(round(x),blue(x)),large(x))
and(and(blue(x),large(x)),round(x))
and(and(large(x),blue(x)),round(x))

4.3 Using CRU for underspecification of MRS-like semantic representations

CRU is intended as an entirely general framework for representational underspecification, and while pseudo-semantic examples were used in previous sections, the description has been general up to this point. From here on, the focus is going to be on using CRU for underspecification of semantic representations for NLG. This section sets down some conventions for writing nested semantic relations in CRU (Section 4.3.1), and gives a simple example of a CRU grammar generating semantic representations for train departure times (Section 4.3.2). These conventions are intended for semantic representation formalisms that are based on generalised quantifier logic, like MRS.

4.3.1 Conventions for representing semantic relations in CRU

As previously mentioned, a syntactically flat representation of embedded semantic relations is achieved in CRU in a standard way that has become fairly standard over recent years — with the use of labels and pointer arguments, atomic arguments that 'point to' the embedded constituent. E.g. *exists(y, cake(y), forall(x, child(x), eat(e,x,y)))* is written as `exists(H0,Y,H1,H2) cake(H1,Y) forall(H2,X,H3,H4) child(H3,X) eat(H4,E,X,Y)`. The arguments starting with `H` are there only to represent the structure of the expression. The first argument is always interpreted as the label of a relation, and any other H-type arguments are pointer arguments. For this to work systematically and unambiguously, and at the same time result in human-readable expressions, a number of conventions need to be observed. These, along with some further conventions for readability of CRU-grammars are described in this section[7].

Basic notational conventions: In relation names, upper-case letters are used for nonterminals, lower-case for terminals. In arguments, initial upper-case letters are used for variables, lower-case for values.

Argument types:

1. Labels and pointers (type H): variables $H, H_0, H_1, H_2, ...$ ranging over the values $h_1, h_2, ...$
2. Object variables (type X): $X_1, X_2, ...$ ranging over $x_1, x_2, ...$
3. Event variables (type E): $E_1, E_2, ...$ ranging over $e_1, e_2, ...$
4. Constant variable types: any string, e.g. $Name$, ranging over finite set of strings, e.g. $john, mary$.

(Non)terminals: Every nonterminal and terminal in a CRU-grammar for semantic representation is of the following form:

$$Relation(Label, Arg_1, ..., Arg_n, H_1, ..., H_m), n, m \leq 0$$

Here, $Relation$ is a string representing the name of the relation, the first argument $Label$ is obligatory and represents the label of the relation, $Arg_1, ..., Arg_n$ are arguments of type X, E or any constant type, and $Label, H_1, ..., H_m$ are arguments of type H.

H-type arguments: While CRU allows the same syntactically flat representation of recursive structures as some other underspecification formalisms, it does not achieve underspecification by tree constraints (using unconnected labels and pointers in combination with sets of constraints in accordance with which pairs of labels and pointers may be connected). In fully specified CRU-expressions, all labels except the topmost label must be connected to exactly one pointer, and all pointers must be connected to at least one label.

As is the case in MRS, each argument position is allowed to be associated with any number of embedded structures, that is, each pointer arguments may be identical to any number of label arguments. E.g.:

```
EXISTS(H0,Y,H4,H3) greedy(H1,X) child(H1,X)
  FORALL(H3,X,H1,H5) big(H4,Y) cake(H4,Y) eat(H5,E,X,Y)
```

[7]Note, however, that these are just one possible set of conventions, and that such conventions will depend on the semantic representation formalism that is being used.

In this expression two nonterminals attach to each `H1` and `H4`. In syntactically recursive representation, CRU expressions with one-to-many pointer-label connections, sets of constituents that are connected to the same pointer argument are enclosed in square brackets to highlight this fact. E.g. the syntactically recursive representation of the above CRU expression is *EXISTS(y,[big(y),cake(y)],FORALL(x,[greedy(x),child(x)],eat(e,x,y))*.

The reason for allowing this is again greater generality in writing CRU-grammar rules: it allows rules to be written that affect a set of relations as one, regardless of their number and type. E.g. in the above expression, it allows all relations in the restriction or the body of a quantifier to be referred to as a set.

4.3.2 Example CRU grammar defining a small generation space

Using the conventions described in previous sections, a CRU grammar can be written that generates structured semantic representations (`def`, `udef` refers to the (in)definite distinction):

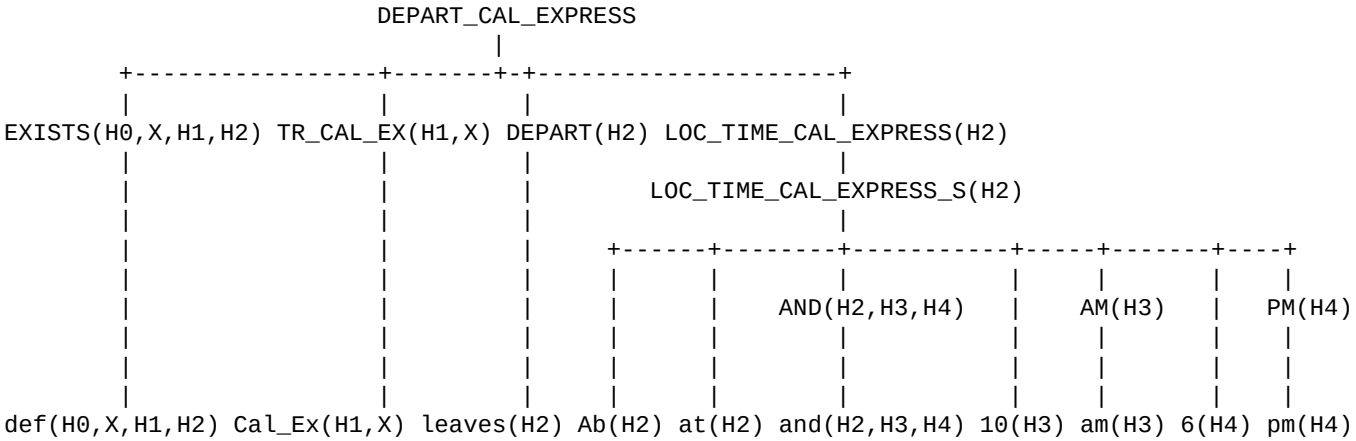
CRU-grammar 2

```
DEPART_CAL_EXPRESS --> EXISTS(H0,X,H1,H2)
                       TRAIN_CAL_EXPRESS(H1,X)
                       DEPART(H2) LOC_TIME_CAL_EXPRESS(H2)
EXISTS(H,X,Ha,Hb) --> def(H,X,Ha,Hb)
EXISTS(H,X,Ha,Hb) --> udef(H,X,Ha,Hb)
TRAIN_CAL_EXPRESS(H,X) --> Caledonian_Express(H,X)
TRAIN_CAL_EXPRESS(H,X) --> train(H,X)
TRAIN_CAL_EXPRESS(H,X) --> it(H,X)
DEPART(H) --> leaves(H)
LOC_TIME_CAL_EXPRESS(H) --> LOC_TIME_CAL_EXPRESS_N(H)
LOC_TIME_CAL_EXPRESS(H) --> LOC_TIME_CAL_EXPRESS_S(H)
AND(H,Ha,Hb) --> and(H,Ha,Hb)
AND(H,Ha,Hb) --> and(H,Hb,Ha)
LOC_TIME_CAL_EXPRESS_N(H) --> London(H) at(H) AND(H,H1,H2) 5(H1) AM(H1) 1(H2) PM(H2)
LOC_TIME_CAL_EXPRESS_S(H) --> Aberdeen(H) at(H) AND(H,H1,H2) 10(H1) AM(H1) 6(H2) PM(H2)
AM(H) --> am(H)
AM(H) --> o'clock(H) in(H) the(H) morning(H)
PM(H) --> pm(H)
PM(H) --> o'clock(H) in(H) the(H) evening(H)
```

This grammar generates 96 strings, the first 20 of which are shown below (converted into syntactically recursive format):

def(x_2, caledonian_Express(x_2), [leaves, london, at, and([five,am],[one,pm])])
def(x_2, caledonian_Express(x_2), [leaves, london, at, and([five,am],[one,oclock,in,the,evening])])
def(x_2, caledonian_Express(x_2), [leaves, london, at, and([five,oclock,in,the,morning],[one,pm])])
def(x_2, caledonian_Express(x_2), [leaves, london, at, and([five,oclock,in,the,morning],[one,oclock,in,the,evening])])
def(x_2, caledonian_Express(x_2), [leaves, london, at, and([one,pm],[five,am])])
def(x_2, caledonian_Express(x_2), [leaves, london, at, and([one,oclock,in,the,evening],[five,am])])
def(x_2, caledonian_Express(x_2), [leaves, london, at, and([one,pm],[five,oclock,in,the,morning])])
def(x_2, caledonian_Express(x_2), [leaves, london, at, and([one,oclock,in,the,evening],[five,oclock,in,the,morning])])
def(x_2, caledonian_Express(x_2), [leaves, aberdeen, at, and([ten,am],[six,pm])])
def(x_2, caledonian_Express(x_2), [leaves, aberdeen, at, and([ten,am],[six,oclock,in,the,evening])])
def(x_2, caledonian_Express(x_2), [leaves, aberdeen, at, and([ten,oclock,in,the,morning],[six,pm])])
def(x_2, caledonian_Express(x_2), [leaves, aberdeen, at, and([ten,oclock,in,the,morning],[six,oclock,in,the,evening])])
def(x_2, caledonian_Express(x_2), [leaves, aberdeen, at, and([six,pm],[ten,am])])
def(x_2, caledonian_Express(x_2), [leaves, aberdeen, at, and([six,oclock,in,the,evening],[ten,am])])
def(x_2, caledonian_Express(x_2), [leaves, aberdeen, at, and([six,pm],[ten,oclock,in,the,morning])])
def(x_2, caledonian_Express(x_2), [leaves, aberdeen, at, and([six,oclock,in,the,evening],[ten,oclock,in,the,morning])])
def(x_2, train(x_2), [leaves, london, at, and([five,am],[one,pm])])
...

The (single) derivation tree for *def(x, Caledonian_Express(x),leaves Aberdeen at and(10am, 6pm))* looks like this (some of the nonterminal/terminal names are abbreviated for space reasons):



def(x, Caledonian_Express(x),leaves Aberdeen at and(10am, 6pm)) has an underspecification set that contains 80 strings (i.e. the string can be underspecified in 80 different ways), including the following 6 (for the full listing see Appendix A):

```
DEPART_CAL_EXPRESS
EXISTS(H0,X,H1,H2) TR_CAL_EX(H1,X) DEPART(H2) LOC_TIME_CAL_EXPRESS(H2)
EXISTS(H0,X,H1,H2) TR_CAL_EX(H1,X) leaves(H2) Ab(H2) at(H3) AND(H2,H3,H4)
  10(H3) am(H3) 6(H4) PM(H4)
EXISTS(H0,X,H1,H2) Cal_Ex(H1,X) DEPART(H2) Ab(H2) at(H3) and(H2,H3,H4)
  10 am(H3) 6(H4) PM(H4)
EXISTS(H0,X,H1,H2) Cal_Ex(H1,X) leaves(H2) Ab(H2) at(H3) and(H2,H3,H4)
  10(H3) AM(H3) 6(H4) pm(H4)
def(H0,X,H1,H2) TR_CAL_EX(H1,X) leaves(H2) Ab(H2) at(H3) AND(H2,H3,H4)
  10(H3) AM(H3) 6(H4) pm(H4)
...
```

4.4 Generating E-sets

The basic idea in using CRU for NL generation is that a CRU-grammar defines a representational space of underspecifiable surface generator inputs, which enables the deep generator, instead of having to generate only fully specified representations (the shaded box in Figure 2), to generate any underspecified representation licensed by the grammar (the nodes above the shaded box in Figure 2). The subsequent generation process proceeds either from the single fully specified representation that was created by the deep generator, or from the set of fully specified representations corresponding to the single *under*specified expression created by the deep generator.

The context-free setting allows the language $L(G)$ generated by a CRU-grammar $G = (W, N, S, R)$, and therefore any E-Set $L(\alpha), \alpha \in U(G)$ to be infinite. That is as it should be for an underspecification formalism. A natural language licenses an infinite number of word strings, so as underspecification increases, at some point, the underspecified forms must expand to an infinite number of fully specified strings. For the sake of descriptive adequacy, therefore, it must be possible for E-Sets $L(\alpha)$ to be infinite.

However, in an NLG system, $L(\alpha)$ will eventually have to be generated, so it makes no sense to allow underspecified representations α for which $L(\alpha)$ is infinite. In a practical context, it is not useful to allow underspecification to the extent where there are an infinite number of ways of realising the representation.

There are two ways of dealing with this, one that is invisible to the grammar writer and one that is not. The former is based on the fact that if there is an infinite $L(\alpha)$, then R must license derivations of the form $A \overset{*}{\underset{G}{\Rightarrow}} \beta A \gamma$. So, one solution is to include a check at compile time that identifies all nonterminals A for which $A \overset{*}{\underset{G}{\Rightarrow}} \beta A \gamma$, and then to disallow such nonterminals from being used in underspecified representations created by the NLG system.

The second possibility is to introduce a formal distinction between nonterminals that can be used in underspecified expressions (NTs) and nonterminals that cannot (RNTs). The idea is then that the former would correspond to decision nodes in the bottom part of the generation tree (Figure 2), while the latter would correspond to those in the top. For example, a top-most rule set could be added to CRU-grammar 1 that licences any number of conjunctions of train departure statements:

```
TOP --> DEPARTING_EVENT
TOP --> DEPARTING_EVENT and TOP
```

This would require the CRU-grammar definitions to be changed to make the set of RNTs explicit. Both solutions mentioned here have promise and will be looked at in more detail.

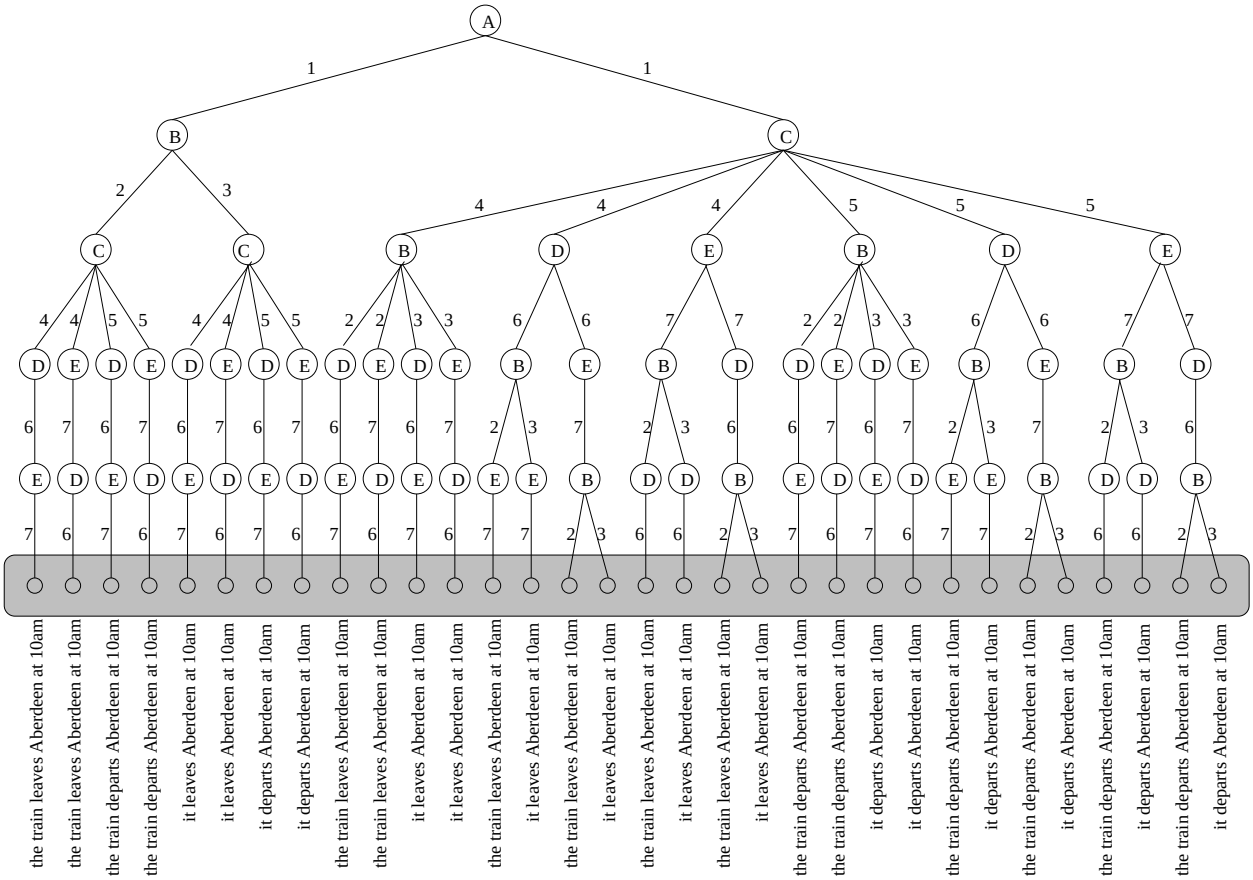


Figure 4: Underspecifiable representation space defined by CRU grammar 3 .

5 Discussion and Comments

This section picks up a few points from previous sections. Section 5.1 returns to the initial claim that generation spaces can be modelled by decision trees, that the nodes in such decision trees can be made representationally explicit in the form of underspecified representations, and that this space of underspecified representations can be defined by a CRU-grammar. To demonstrate how CRU achieves this, it shows a diagram of the generation space encoded by a simple CRU-grammar.

Section 5.2 considers whether meta-variables should and can be incorporated into CRU. Section 5.3 informally compares CRU with tree constraint formalisms.

5.1 CRU and the generation space

As discussed previously, CRU permits the decision trees in terms of which generation spaces can be construed to be made representationally explicit in the form of underspecification spaces. Figure 4 shows a representation of the generation space defined by the following grammar (which is a simplified version of CRU-grammar 1)[8]. The tree diagram is analogous to that in Figure 3.

CRU-grammar 3

```
A 1 DEPARTING_EVENT --> TRAIN DEPART
B 2 TRAIN --> the train
B 3 TRAIN --> it
C 4 DEPART --> leaves LOCATION TIME
C 5 DEPART --> departs LOCATION TIME
D 6 LOCATION --> Aberdeen
E 7 TIME --> at 10am
```

[8]To keep the representational space of the grammar small enough to be representable as a diagram on a printed page, it had to be a very simple grammar generating a language of unstructured strings.

Each decision node in the tree corresponds to a set of rules that expand the same nonterminal in the grammar (indexed A–E above), while each branch corresponds to a single rule (numbered 1–7). The grammar generates four different fully specified strings, and 24 different underspecified strings. However, it licenses 32 different generation processes (32 different complete CFG derivations, or $S \stackrel{*}{\underset{G}{\Rightarrow}} \alpha, \alpha \in L(G)$), as it allows the order in which decisions are taken (nonterminals expanded) to be fairly free. Not completely free of course: decision A has to be taken before any others, and C has to be taken before D and E.

The idea underlying CRU is that each nonterminal together with its set of expansion rules encodes a minimal dimension of variation. The LHS can be considered a name for the variation (decision to be taken) and the RHSs are the corresponding variants. From the point of view of underspecification: the expression on the LHS is underspecified between all its RHSs.

The generation space must therefore be defined in terms of a tree of decisions that depend only on other decisions already taken. If two or more decisions are entirely interdependent they should be encoded as a single decision node, or rather, as a subtree (of the decision tree) rooted at that node. If there are large sets of entirely interdependent decisions then the CRU grammar becomes inelegant and unwieldy (though not impossible).

5.2 CRU and meta-variables

There are a whole range of ambiguity phenomena (from the point of view of NLU) where the set of possible readings, while possibly inferrable by some general function from the context, cannot be stated generally and independently of the context. Examples are unresolved anaphoric and deictic references. Meta-variables (see Section 2.2.3) are an important mechanism for NLU because such ambiguity cannot be accounted for by meta-constants or structural underspecification. In NLG, the usefulness of meta-variables is less obvious, because the set of variants is (usually) known. An exception is interlingua-based MT, where generation may have to be from an incomplete source language analysis.

It would be straightforward to represent meta-variables themselves in CRU, e.g. by introducing a new argument type for them. However, in order to expand underspecified forms to fully specified forms, what is also needed is the function that maps meta-variables (and context representations) into sets of variants. The issue of how exactly to do this in conjunction with CRU has not so far been looked at, as it is unlikely to be of concern for the kind of generation the COGENT project is planning to look at.

5.3 Structural underspecification in CRU and in tree constraint formalisms

The main underspecification formalisms capable of structural underspecification all use some tree constraint language, e.g. Hole Semantics, MRS, the Constraint Language for Lambda Structures (CLLS), and Context Unification (CU). The fundamental idea is to view expressions as trees, and to use partially specified tree descriptions to represent sets of expressions.

Partial tree descriptions can usually be construed as 2-tuples $< B, C >$ where B is a set of 'tree building blocks' and C a set of constraints on how the building blocks may be put together to form trees. The language $\mathbf{B}$ of building blocks specifies what constitutes a building block, in some cases simply a node (CLLS, MRS), in others, trees (CU). The constraint language $\mathbf{C}$ is usually some small set of binary constraint relations (e.g. dominance or identity) over elements of $\mathbf{B}$. Building blocks are usually labelled, and constraints over building blocks are represented as constraint relations between labels.

If a tree T can be constructed from the elements of B in such way that all relations in C are true, then T is said to satisfy $< B, C >$. The problem of determining whether some T exists that satisfies $< B, C >$ is the *satisfiability problem* of tree constraint languages. The problem of determining the set of all T that satisfy $< B, C >$ is the *enumeration problem* of tree constraint languages.

[Koller et al., 1998] showed that the satisfiability problem of the language of dominance and labelling constraints (with nodes as building-blocks) is NP-complete. And that is only a subproblem of the general satisfiability problem of tree constraint languages, which in turn is only a subproblem of the enumeration problem.

Koller et al. have, however,identified a fragment of the general tree constraint formalism for which a polynomial solution exists for the satisfiability problem, and for which solutions can be listed in polynomial time [Koller et al., 2000]. They have also demonstrated that this fragment can be used to make enumeration of solutions efficient for subsets of Hole Semantics expressions [Koller et al., 2003] and MRS expressions [Fuchs et al., 2004], but have not shown to what extent the expressive power of the two formalisms can be maintained.

It is not straightforward to compare tree constraint formalisms with the way CRU achieves structural underspecification. Underspecification formalisms that use tree constraints provide a very powerful, but very expensive formalism in which any tree underspecification can be achieved. Their effect can be seen as defining the set of all possible scopings and on top of that defining constraints on the basis of which certain elements of the set can be eliminated. This

means in the worst case all members of the complete set have to be considered explicitly before the subset of those that violate the constraints can be determined.

In contrast, CRU forces you to write down explicitly for each case what the possible scopings are, so there is a direct mapping from underspecified representation to all possible scopings, and no impossible scopings ever have to be considered. This, together with the upper limit on structural underspecification, results in much lower computational cost.

The price is that while the CRU formalism is general enough to allow any kind of structural underspecification, a particular CRU grammar must (implicitly) define a depth of embedding for underspecifiable embedded structures. Furthermore, CRU is less representationally efficient than tree constraint languages.

The above is only an superficial comparison of the different ways in which CRU and tree constraint languages achieve structural underspecification, and a more in-depth investigation will follow, in particular, to what extent tree-constraint languages have representational power necessary for NL underspecification that CRU does not have.

6 Summary and current work

This report introduced Context-free Representational Underspecification, a general framework for defining underspecifiable representation formalisms. It described how CRU can be used to make the decision space underlying the natural language generation process representationally explicit, and this constitutes the first approach to systematic use of underspecification for NLG. It also described plans for using CRU in the COGENT Project, where we intend to use it to make the interface to surface generation flexible in the sense that it allows interface representations of a range of different degrees of specificity. A flexible interface makes the surface generator more readily reusable because a greater number of deep generation modules will be capable of generating suitable inputs.

In terms of representational underspecification, CRU has several innovative properties: it (i) allows an arbitrary number of degrees of underspecification, (ii) specifies a hierarchical ordering of underspecified and fully specified representations in terms of their degree of underspecification, (iii) offers the possibility to assign probability distributions to mappings from underspecified to fully specified representations (PCFGs), and (iv) is computationally more efficient than the main existing underspecification formalisms.

A prototype implementation of CRU and several small-scale CRU-grammars for generating patient information leaflets and weather forecast summaries are currently being created. A software package implementing the technology described in this report will be released in the future.

Appendix A: Example of a U-Set (CRU-grammar 2)

The full underspecification set for the string *def(x, Caledonian_Express(x),leaves Aberdeen at and(10am,6pm))* under CRU-grammar 2 (Section 4.2):

```
DEPART_CAL_EXPRESS
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:LOC_TIME_CAL_EXPRESS
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:LOC_TIME_CAL_EXPRESS_S
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:LOC_TIME_CAL_EXPRESS
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:LOC_TIME_CAL_EXPRESS_S
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:LOC_TIME_CAL_EXPRESS
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:LOC_TIME_CAL_EXPRESS_S
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:LOC_TIME_CAL_EXPRESS
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:LOC_TIME_CAL_EXPRESS_S
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:LOC_TIME_CAL_EXPRESS
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:LOC_TIME_CAL_EXPRESS_S
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:PM
```

```
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:EXISTS(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:LOC_TIME_CAL_EXPRESS
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:LOC_TIME_CAL_EXPRESS_S
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:def(X,H1,H2) H1:TR_CAL_EXPRESS(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:LOC_TIME_CAL_EXPRESS
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:LOC_TIME_CAL_EXPRESS_S
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:DEPART H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:LOC_TIME_CAL_EXPRESS
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:LOC_TIME_CAL_EXPRESS_S
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:AND(H3,H4) H3:10 H3:am H4:6 H4:pm
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:PM
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:AM H4:6 H4:pm
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:PM
H0:def(X,H1,H2) H1:Caledonian_Express(X) H2:leaves H2:Aber H3:at H2:and(H3,H4) H3:10 H3:am H4:6 H4:pm
```

Appendix B: Generation space encoded by CRU-grammar 1

```
DEPARTING_EVENT
-- TRAIN DEPART
   -- the Caledonian Express DEPART
      -- the Caledonian Express leaves LOCATION TIME
         -- the Caledonian Express leaves Aberdeen TIME
            <<< the Caledonian Express leaves Aberdeen at 10am
            <<< the Caledonian Express leaves Aberdeen at 10 o’clock in the morning
         -- the Caledonian Express leaves LOCATION at 10am
            <<< the Caledonian Express leaves Aberdeen at 10am
         -- the Caledonian Express leaves LOCATION at 10 o’clock in the morning
            <<< the Caledonian Express leaves Aberdeen at 10 o’clock in the morning
      -- the Caledonian Express departs LOCATION TIME
         -- the Caledonian Express departs Aberdeen TIME
            <<< the Caledonian Express departs Aberdeen at 10am
            <<< the Caledonian Express departs Aberdeen at 10 o’clock in the morning
         -- the Caledonian Express departs LOCATION at 10am
            <<< the Caledonian Express departs Aberdeen at 10am
         -- the Caledonian Express departs LOCATION at 10 o’clock in the morning
            <<< the Caledonian Express departs Aberdeen at 10 o’clock in the morning
      -- the Caledonian Express departs from LOCATION TIME
         -- the Caledonian Express departs from Aberdeen TIME
            <<< the Caledonian Express departs from Aberdeen at 10am
            <<< the Caledonian Express departs from Aberdeen at 10 o’clock in the morning
         -- the Caledonian Express departs from LOCATION at 10am
            <<< the Caledonian Express departs from Aberdeen at 10am
         -- the Caledonian Express departs from LOCATION at 10 o’clock in the morning
            <<< the Caledonian Express departs from Aberdeen at 10 o’clock in the morning
   -- the train DEPART
      -- the train leaves LOCATION TIME
         -- the train leaves Aberdeen TIME
            <<< the train leaves Aberdeen at 10am
            <<< the train leaves Aberdeen at 10 o’clock in the morning
         -- the train leaves LOCATION at 10am
            <<< the train leaves Aberdeen at 10am
         -- the train leaves LOCATION at 10 o’clock in the morning
            <<< the train leaves Aberdeen at 10 o’clock in the morning
     -- the train departs LOCATION TIME
         -- the train departs Aberdeen TIME
            <<< the train departs Aberdeen at 10am
            <<< the train departs Aberdeen at 10 o’clock in the morning
         -- the train departs LOCATION at 10am
            <<< the train departs Aberdeen at 10am
         -- the train departs LOCATION at 10 o’clock in the morning
            <<< the train departs Aberdeen at 10 o’clock in the morning
      -- the train departs from LOCATION TIME
         -- the train departs from Aberdeen TIME
            <<< the train departs from Aberdeen at 10am
            <<< the train departs from Aberdeen at 10 o’clock in the morning
         -- the train departs from LOCATION at 10am
            <<< the train departs from Aberdeen at 10am
         -- the train departs from LOCATION at 10 o’clock in the morning
            <<< the train departs from Aberdeen at 10 o’clock in the morning
   -- it DEPART
      -- it leaves LOCATION TIME
         -- it leaves Aberdeen TIME
            <<< it leaves Aberdeen at 10am
            <<< it leaves Aberdeen at 10 o’clock in the morning
         -- it leaves LOCATION at 10am
            <<< it leaves Aberdeen at 10am
         -- it leaves LOCATION at 10 o’clock in the morning
            <<< it leaves Aberdeen at 10 o’clock in the morning
      -- it departs LOCATION TIME
         -- it departs Aberdeen TIME
            <<< it departs Aberdeen at 10am
            <<< it departs Aberdeen at 10 o’clock in the morning
         -- it departs LOCATION at 10am
            <<< it departs Aberdeen at 10am
         -- it departs LOCATION at 10 o’clock in the morning
            <<< it departs Aberdeen at 10 o’clock in the morning
      -- it departs from LOCATION TIME
         -- it departs from Aberdeen TIME
            <<< it departs from Aberdeen at 10am
            <<< it departs from Aberdeen at 10 o’clock in the morning
```

```
      -- it departs from LOCATION at 10am
         <<< it departs from Aberdeen at 10am
      -- it departs from LOCATION at 10 o'clock in the morning
         <<< it departs from Aberdeen at 10 o'clock in the morning
-- TRAIN leaves LOCATION TIME
   -- the Caledonian Express leaves LOCATION TIME
      -- the Caledonian Express leaves Aberdeen TIME
         <<< the Caledonian Express leaves Aberdeen at 10am
         <<< the Caledonian Express leaves Aberdeen at 10 o'clock in the morning
      -- the Caledonian Express leaves LOCATION at 10am
         <<< the Caledonian Express leaves Aberdeen at 10am
      -- the Caledonian Express leaves LOCATION at 10 o'clock in the morning
         <<< the Caledonian Express leaves Aberdeen at 10 o'clock in the morning
   -- the train leaves LOCATION TIME
      -- the train leaves Aberdeen TIME
         <<< the train leaves Aberdeen at 10am
         <<< the train leaves Aberdeen at 10 o'clock in the morning
      -- the train leaves LOCATION at 10am
         <<< the train leaves Aberdeen at 10am
      -- the train leaves LOCATION at 10 o'clock in the morning
         <<< the train leaves Aberdeen at 10 o'clock in the morning
   -- it leaves LOCATION TIME
      -- it leaves Aberdeen TIME
         <<< it leaves Aberdeen at 10am
         <<< it leaves Aberdeen at 10 o'clock in the morning
      -- it leaves LOCATION at 10am
         <<< it leaves Aberdeen at 10am
      -- it leaves LOCATION at 10 o'clock in the morning
         <<< it leaves Aberdeen at 10 o'clock in the morning
   -- TRAIN leaves Aberdeen TIME
      -- the Caledonian Express leaves Aberdeen TIME
         <<< the Caledonian Express leaves Aberdeen at 10am
         <<< the Caledonian Express leaves Aberdeen at 10 o'clock in the morning
      -- the train leaves Aberdeen TIME
         <<< the train leaves Aberdeen at 10am
         <<< the train leaves Aberdeen at 10 o'clock in the morning
      -- it leaves Aberdeen TIME
         <<< it leaves Aberdeen at 10am
         <<< it leaves Aberdeen at 10 o'clock in the morning
      -- TRAIN leaves Aberdeen at 10am
         <<< the Caledonian Express leaves Aberdeen at 10am
         <<< the train leaves Aberdeen at 10am
         <<< it leaves Aberdeen at 10am
     -- TRAIN leaves Aberdeen at 10 o'clock in the morning
         <<< the Caledonian Express leaves Aberdeen at 10 o'clock in the morning
         <<< the train leaves Aberdeen at 10 o'clock in the morning
         <<< it leaves Aberdeen at 10 o'clock in the morning
   -- TRAIN leaves LOCATION at 10am
      -- the Caledonian Express leaves LOCATION at 10am
         <<< the Caledonian Express leaves Aberdeen at 10am
      -- the train leaves LOCATION at 10am
         <<< the train leaves Aberdeen at 10am
      -- it leaves LOCATION at 10am
         <<< it leaves Aberdeen at 10am
      -- TRAIN leaves Aberdeen at 10am
         <<< the Caledonian Express leaves Aberdeen at 10am
         <<< the train leaves Aberdeen at 10am
         <<< it leaves Aberdeen at 10am
   -- TRAIN leaves LOCATION at 10 o'clock in the morning
      -- the Caledonian Express leaves LOCATION at 10 o'clock in the morning
         <<< the Caledonian Express leaves Aberdeen at 10 o'clock in the morning
      -- the train leaves LOCATION at 10 o'clock in the morning
         <<< the train leaves Aberdeen at 10 o'clock in the morning
      -- it leaves LOCATION at 10 o'clock in the morning
         <<< it leaves Aberdeen at 10 o'clock in the morning
      -- TRAIN leaves Aberdeen at 10 o'clock in the morning
         <<< the Caledonian Express leaves Aberdeen at 10 o'clock in the morning
         <<< the train leaves Aberdeen at 10 o'clock in the morning
         <<< it leaves Aberdeen at 10 o'clock in the morning
-- TRAIN departs LOCATION TIME
   -- the Caledonian Express departs LOCATION TIME
      -- the Caledonian Express departs Aberdeen TIME
         <<< the Caledonian Express departs Aberdeen at 10am
         <<< the Caledonian Express departs Aberdeen at 10 o'clock in the morning
      -- the Caledonian Express departs LOCATION at 10am
         <<< the Caledonian Express departs Aberdeen at 10am
      -- the Caledonian Express departs LOCATION at 10 o'clock in the morning
         <<< the Caledonian Express departs Aberdeen at 10 o'clock in the morning
   -- the train departs LOCATION TIME
      -- the train departs Aberdeen TIME
         <<< the train departs Aberdeen at 10am
         <<< the train departs from Aberdeen at 10 o'clock in the morning
      -- the train departs LOCATION at 10am
         <<< the train departs Aberdeen at 10am
      -- the train departs LOCATION at 10 o'clock in the morning
         <<< the train departs from Aberdeen at 10 o'clock in the morning
   -- it departs LOCATION TIME
      -- it departs Aberdeen TIME
         <<< it departs Aberdeen at 10am
         <<< it departs Aberdeen at 10 o'clock in the morning
      -- it departs LOCATION at 10am
         <<< it departs Aberdeen at 10am
      -- it departs LOCATION at 10 o'clock in the morning
         <<< it departs Aberdeen at 10 o'clock in the morning
   -- TRAIN departs Aberdeen TIME
      -- the Caledonian Express departs Aberdeen TIME
         <<< the Caledonian Express departs Aberdeen at 10am
         <<< the Caledonian Express departs Aberdeen at 10 o'clock in the morning
      -- the train departs Aberdeen TIME
         <<< the train departs Aberdeen at 10am
         <<< the train departs Aberdeen at 10 o'clock in the morning
      -- it departs Aberdeen TIME
         <<< it departs Aberdeen at 10am
         <<< it departs Aberdeen at 10 o'clock in the morning
      -- TRAIN departs Aberdeen at 10am
         <<< the Caledonian Express departs Aberdeen at 10am
         <<< the train departs Aberdeen at 10am
         <<< it departs Aberdeen at 10am
      -- TRAIN departs Aberdeen at 10 o'clock in the morning
         <<< the Caledonian Express departs Aberdeen at 10 o'clock in the morning
         <<< the train departs Aberdeen at 10 o'clock in the morning
         <<< it departs Aberdeen at 10 o'clock in the morning
   -- TRAIN departs LOCATION at 10am
      -- the Caledonian Express departs LOCATION at 10am
         <<< the Caledonian Express departs Aberdeen at 10am
      -- the train departs LOCATION at 10am
         <<< the train departs Aberdeen at 10am
```

```
      -- it departs LOCATION at 10am
         <<< it departs Aberdeen at 10am
      -- TRAIN departs Aberdeen at 10am
         <<< the Caledonian Express departs Aberdeen at 10am
         <<< the train departs Aberdeen at 10am
         <<< it departs Aberdeen at 10am
   -- TRAIN departs LOCATION at 10 o'clock in the morning
      -- the Caledonian Express departs LOCATION at 10 o'clock in the morning
         <<< the Caledonian Express departs Aberdeen at 10 o'clock in the morning
      -- the train departs LOCATION at 10 o'clock in the morning
         <<< the train departs Aberdeen at 10 o'clock in the morning
      -- it departs LOCATION at 10 o'clock in the morning
         <<< it departs Aberdeen at 10 o'clock in the morning
      -- TRAIN departs Aberdeen at 10 o'clock in the morning
         <<< the Caledonian Express departs Aberdeen at 10 o'clock in the morning
         <<< the train departs Aberdeen at 10 o'clock in the morning
         <<< it departs Aberdeen at 10 o'clock in the morning
-- TRAIN departs from LOCATION TIME
   -- the Caledonian Express departs from LOCATION TIME
      -- the Caledonian Express departs from Aberdeen TIME
         <<< the Caledonian Express departs from Aberdeen at 10am
         <<< the Caledonian Express departs from Aberdeen at 10 o'clock in the morning
      -- the Caledonian Express departs from LOCATION at 10am
         <<< the Caledonian Express departs from Aberdeen at 10am
      -- the Caledonian Express departs from LOCATION at 10 o'clock in the morning
         <<< the Caledonian Express departs from Aberdeen at 10 o'clock in the morning
   -- the train departs from LOCATION TIME
      -- the train departs from Aberdeen TIME
         <<< the train departs from Aberdeen at 10am
         <<< the train departs from Aberdeen at 10 o'clock in the morning
      -- the train departs from LOCATION at 10am
         <<< the train departs from Aberdeen at 10am
      -- the train departs from LOCATION at 10 o'clock in the morning
         <<< the train departs from Aberdeen at 10 o'clock in the morning
   -- it departs from LOCATION TIME
      -- it departs from Aberdeen TIME
         <<< it departs from Aberdeen at 10am
         <<< it departs from Aberdeen at 10 o'clock in the morning
      -- it departs from LOCATION at 10am
         <<< it departs from Aberdeen at 10am
      -- it departs from LOCATION at 10 o'clock in the morning
         <<< it departs from Aberdeen at 10 o'clock in the morning
   -- TRAIN departs from Aberdeen TIME
      -- the Caledonian Express departs from Aberdeen TIME
         <<< the Caledonian Express departs from Aberdeen at 10am
         <<< the Caledonian Express departs from Aberdeen at 10 o'clock in the morning
      -- the train departs from Aberdeen TIME
         <<< the train departs from Aberdeen at 10am
         <<< the train departs from Aberdeen at 10 o'clock in the morning
      -- it departs from Aberdeen TIME
         <<< it departs from Aberdeen at 10am
         <<< it departs from Aberdeen at 10 o'clock in the morning
      -- TRAIN departs from Aberdeen at 10am
         <<< the Caledonian Express departs from Aberdeen at 10am
         <<< the train departs from Aberdeen at 10am
         <<< it departs from Aberdeen at 10am
      -- TRAIN departs from Aberdeen at 10 o'clock in the morning
         <<< the Caledonian Express departs from Aberdeen at 10 o'clock in the morning
         <<< the train departs from Aberdeen at 10 o'clock in the morning
         <<< it departs from Aberdeen at 10 o'clock in the morning
   -- TRAIN departs from LOCATION at 10am
      -- the Caledonian Express departs from LOCATION at 10am
         <<< the Caledonian Express departs from Aberdeen at 10am
      -- the train departs from LOCATION at 10am
         <<< the train departs from Aberdeen at 10am
      -- it departs from LOCATION at 10am
         <<< it departs from Aberdeen at 10am
      -- TRAIN departs from Aberdeen at 10am
         <<< the Caledonian Express departs from Aberdeen at 10am
         <<< the train departs from Aberdeen at 10am
         <<< it departs from Aberdeen at 10am
   -- TRAIN departs from LOCATION at 10 o'clock in the morning
      -- the Caledonian Express departs from LOCATION at 10 o'clock in the morning
         <<< the Caledonian Express departs from Aberdeen at 10 o'clock in the morning
      -- the train departs from LOCATION at 10 o'clock in the morning
         <<< the train departs from Aberdeen at 10 o'clock in the morning
      -- it departs from LOCATION at 10 o'clock in the morning
         <<< it departs from Aberdeen at 10 o'clock in the morning
      -- TRAIN departs from Aberdeen at 10 o'clock in the morning
         <<< the Caledonian Express departs from Aberdeen at 10 o'clock in the morning
         <<< the train departs from Aberdeen at 10 o'clock in the morning
         <<< it departs from Aberdeen at 10 o'clock in the morning
```

References

[Ahn et al., 1994] Ahn, R., Kievit, L., Rentier, G., and Verlinden, M. (1994). Two levels of semantic representation in denk. In oine Andernach, Moll, M., and Nijholt, A., editors, *CLIN V: papers from the fifth CLIN meeting*, Enschede. Parlevink Research Group, Universiteit Twente.

[Alshawi, 1990] Alshawi, H. (1990). Resolving quasi logical forms. *Computational Linguistics*, 16:133–144.

[Belz, 2002] Belz, A. (2002). PCFG learning by nonterminal partition search. In Adriaans, P., Fernau, H., and van Zaanen, M., editors, *Grammatical Inference: Algorithms and Applications. Proceedings of the 6th International Colloquium on Grammatical Inference (ICGI 2002)*, pages 14–27. Berlin: Springer.

[Bos, 1995] Bos, J. (1995). Predicate logic unplugged. In Dekker, P. and Stokhof, M., editors, *Proceedings of the 10th Amsterdam Colloquium*.

[Bronnenberg et al., 1979] Bronnenberg, W., Bunt, H., Landsbergen, J., Scha, R., w. Schoenmakers, and van Utteren, E. (1979). The question answering system phliqa2. In Bolc, L., editor, *Natural language question answering systems*, pages 217–305. McMillan.

[Bunt, 2003] Bunt, H. (2003). Underspecification in semantic representations: Which technique for what purpose? In *Proceedings of the 5th Workshop on Computational Semantics (IWCS-5)*, pages 37–54.

[Bunt and Muskens, 1999] Bunt, H. and Muskens, R. (1999). Computational semantics. In Bunt, H. and Musken, R. A., editors, *Computing Meaning*, number 73 in Studies in Linguistics and Philosophy, pages 1–32. Kluwer, Dordrecht.

[Bunt et al., 1984] Bunt, H. C., Beun, R. J., Dols, F., van der Linden, J., and thoe Schwartzenberg, G. O. (1984). The TENDUM dialogue system and its theoretical basis. IPO Annual Progress Report 19, IPO.

[Cahill et al., 2001] Cahill, L., Carroll, J., Evans, R., Paiva, D., Power, R., Scott, D., and van Deemter, K. (2001). From rags to riches: Exploiting the potential of a flexible generation architecture. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics and the 10th Conference of the European Chapter of the Association for Computational Linguistics (ACL-EACL '01)*, pages 98–105.

[Cooper, 1983] Cooper, R. (1983). *Quantification and Syntactic Theory*. Reidel, Dordrecht.

[Copestake et al., 1999] Copestake, A., Flickinger, D., and Sag, I. (1999). Minimal recursion semantics: An introduction. Draft, available online at http://www-csli.stanford.edu/ aac/papers/newmrs.ps.

[Ferreira et al., 2002] Ferreira, F., Bailey, K., and Ferraro, V. (2002). Good enough representations in language comprehension. *Current Directions in Psychological Science*, 11:11–15.

[Fuchs et al., 2004] Fuchs, R., Koller, A., Niehren, J., and Thater, S. (2004). Minimal recursion semantics as dominance constraints: Translation, evaluation, and analysis. In *Proceedings of the 42nd ACL*, Barcelona.

[Geurts and Rentier, 1991] Geurts, B. and Rentier, G. (1991). Quasi-logical form in PLUS. Esprit project p5254 (plus) internal report, ITK, Tilburg University.

[Hobbs and Shieber, 1987] Hobbs, J. R. and Shieber, S. M. (1987). An algorithm for generating quantifier scopings. *Computational Linguistics*, 13(1–2):47–63.

[Hopcroft and Ullman, 1979] Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

[Kempson, 2003] Kempson, R. (2003). Nonrestrictive relatives and growth of logical form. In *Proceedings of the West Coast Conference on Formal Linguistics 22*, pages 301–314.

[Kievit, 1994] Kievit, L. (1994). Proto-ULF: DenK research report 94/02. Technical report, ITK, Tilburg University.

[Kievit et al., 2001] Kievit, L., Piwek, P., Beun, R.-J., and Bunt, H. (2001). Multimodal cooperative resolution of referential expressions in the DenK system. *Lecture Notes in Computer Science*, 2155:197–214.

[Koller et al., 2000] Koller, A., Mehlhorn, K., and Niehren, J. (2000). A polynomial-time fragment of dominance constraints. In *Proceedings of the 38th ACL*, Hong Kong.

[Koller et al., 2003] Koller, A., Niehren, J., and Thater, S. (2003). Bridging the gap between underspecification formalisms: Hole semantics as dominance constraints. In *Proceedings of the 11th EACL*, Budapest.

[Koller et al., 1998] Koller, A., Niehren, J., and Treinen, R. (1998). Dominance constraints: Algorithms and complexity. In *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics (LACL '98)*, Grenoble, France. Also published in Springer LNCS.

[König and Reyle, 1999] König, E. and Reyle, U. (1999). A general reasoning scheme for underspecified representations. In Ohlbach, H. J. and Reyle, U., editors, *Logic, Language and Reasoning: Essays in Honour of Dov Gabbay*. Kluwer. Written 1996.

[Langkilde, 2000] Langkilde, I. (2000). Forest-based statistical sentence generation. In *Proceedings of the 6th Applied Natural Language Processing Conference and the 1st Meeting of the North American Chapter of the Association of Computational Linguistics (ANLP-NAACL '00)*, pages 170–177.

[Niedermair, 1987] Niedermair, G. T. (1987). Syntactic analysis in speech understanding. In *Proceedings of EUROSPEECH 1987*.

[Niedermair, 1992] Niedermair, G. T. (1992). Syntax, semantik und dialog in spicos ii. In Mangold, H., editor, *Sprachlicher Mensch-Maschine-Kommunikation*, pages 79–90. Oldenbourg Verlag, Munich.

[Pinkal, 1999] Pinkal, M. (1999). On semantic underspecification. In *Computing Meaning*, pages 33–55. Kluwer, Dordrecht.

[Pulman, 2000] Pulman, S. G. (2000). Bidirectional contextual resolution. *Computational Linguistics*, 26(4).

[Reiter, 1994] Reiter, E. (1994). Has a consensus NL generation architecture appeared and is it psycholinguistically plausible? In *Proceedings of INLG 1994*, pages 163–170.

[Reiter and Dale, 1997] Reiter, E. and Dale, R. (1997). Building applied natural language generation systems. *Natural Langauge Engineering*, 3(1):57–87.

[Reyle, 1993] Reyle, U. (1993). Dealing with ambiguities by underspecification: Construction, representation and deduction'. *Journal of Semantics*, 10:123–179.

[Traxler et al., 1998] Traxler, M., Pickering, M., and Clifton, C. J. (1998). Adjunct attachment is not a form of ambiguity resolution. *Journal of Memory and Language*, 39:558–592.

[Varges and Mellish, 2001] Varges, S. and Mellish, C. (2001). Instance-based natural language generation. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL '01)*, pages 1–8.

[Woods, 1978] Woods, W. (1978). Semantics and quantification in natural language question answering. In Yovits, M., editor, *Advances in Computers*, pages 2–64. Academic Press, New York.

[Woods et al., 1972] Woods, W. A., Kaplan, R. M., and Nash-Webber, B. (1972). The lunar sciences natural language information system: Final report. Technical Report BBN Report No. 2378, Bolt, Beranek and Newman, Inc., Cambridge, Massachusetts.