Incremental Generation of Self-corrections Using Underspecification

In: Mariët Theune, Anton Nijholt, and Hendri Hondorp, eds. (2002) Computational Linguistics in the Netherlands 2001: Selected Papers from the Twelfth CLIN Meeting, pp 118–132. Amsterdam, New York: Rodopi.

Incremental generation of self-corrections using underspecification

Markus Guhe, Frank Schilder

Department of Informatics, University of Hamburg

Abstract

The focus of the present paper is on the interface between the first two components of a language production model, i.e. conceptualiser and formulator. We propose a method of how the incremental input for the formulator can be represented within an underspecification formalism. The proposed method can be used in dynamically changing environments, e.g. an airport supervision and information system, to produce descriptions of ongoing events. Because such a highly dynamic setting requires a real time production, an *incremental* conceptualiser that produces increments piece by piece is used in order to speed up the production process. Moreover, changes in the environment have to be recognised and quickly reported to the user. For this reason an *underspecified* representation for the increments is crucial. Underspecification allows information to be added monotonically as soon as a change in the environment has been noticed, e.g. 'CK-314 is on time … uh … is delayed.'

1 Introduction

Interactive dialogue systems have to cope with numerous challenges. They can only be considered to adequately simulate a human-human-like dialogue if they meet a high standard regarding robustness and naturalness of natural language understanding and production. We are concerned here with the production side of this enterprise and propose a method of how already generated information can be dynamically *corrected*. Current dialogue systems lack the following vital requirements for building a robust and natural system: (1) the system needs to react quickly and appropriately to changes in the environment, (2) the output should be generated similarly to what a user would expect from a human communication partner.

As part of fulfilling these requirements, we employ incrementality as a processing principle for our model of language production and propose underspecification as the underlying representation formalism. More specifically, we build on earlier work on the incremental conceptualiser INC (Guhe and Habel 2001) and employ the underspecification formalism CLLS (*Constraint Language for Lambda Structures*) (Egg, Koller and Niehren 2001) in order to produce self-corrections.

Incrementality is advantageous for systems in a real time processing environment, because smaller pieces (*increments*) can already be processed by subsequent components before the entire planning process is finished (Kempen and Hoenkamp 1987, Levelt 1989, De Smedt 1990). An underspecification formalism is well suited for incremental language generation, because an underspecified representation can easily be extended without overwriting earlier made planning decisions: subsequently added increments can be inserted where the underspecified representation left room.

From a dialogue management perspective, generating self-corrections is an appropriate way to draw the user's attention to detected mistakes or to changes in the content the system plans to convey. Moreover, self-corrections enhance the naturalness of the output of a language production system. Not only does a self-correction sound more natural, a system without this particular feature may produce contradictory output, e.g. (1). This contradiction is absent in (2).

(1) I have two seats available. I have one seat available.

(2) I have two seats … uh no … one seat available.

The method for generating self-corrections we are proposing here is partially implemented. However, note that our current goal is not the implementation of a fully functioning dialogue system. Instead, we want to show how self-corrections can be conceptualised and how the output of the conceptualisation process can be represented in an underspecified format. More precisely, we present how INC (Guhe and Habel 2001), incrementally generates so-called *preverbal messages* (Levelt 1989) that can contain self-corrections. Preverbal messages are semantic structures that are suited to be encoded linguistically by a subsequent incremental formulator, e.g. (De Smedt 1990). Such formulators are capable of incrementally producing a linguistic surface structure. However, the input – the preverbal message – is added by hand. We want to close this gap and provide a method of how to *automatically* produce a preverbal message in an incremental way.

INC is a cognitively motivated model of the conceptualiser. Systems that take into account the cognitive foundations of human communications can help to improve the naturalness of output (De Smedt, Horacek and Zock 1996, Reiter 1994). Right now, the implementation of INC is capable of incrementally generating a sequence of preverbal messages, in which a each preverbal message is represented by a complex concept, cf. Guhe and Habel (2001). Based on this and on the considerations of Guhe (under review) the topic of current and future work is to generate the preverbal messages themselves in an incremental fashion, as well, so that it becomes possible to connect INC to a subsequent incremental formulator. – In other words: up to now the preverbal messages are the increments, now we propose how the increments a preverbal message consists of can be generated. –

In Guhe and Schilder (2002) we focus on the issue of how the internal conceptual representations of INC can be combined with the underspecification formalism CLLS (Egg et al. 2001), and how it can be used to incrementally generate VP-ellipses. This is the case in which a preverbal message is dynamically *extended*. In the present paper we focus on the issue of how already generated information can be dynamically *corrected*. Our approach provides a method to extend the preverbal message monotonically in both cases. The generation of self-corrections is a more complex task, because the preverbal message is not only monotonically extended, but extended in a way that ensures that the resulting preverbal message is valid, i.e. is equivalent to a structure that would have been generated without the self-correction.

We are interested in a special kind of self-correction, those that arise because of a *conceptual change*. We understand conceptual changes as a change in the

input data during the incremental generation of the preverbal message. In contrast to conceptual changes we call errors that are caused by a malfunctioning of the system *performance errors*.

One important point of our approach and of the dynamic domain chosen is that there are certain effects that are due to the (partly) parallel processing of increments. That is, the sequence in which computations are performed and the point in time when a computation takes place has effects on the generated output, cf. also Guhe (under review). For example, different outputs are possible depending on when the change was discovered by the conceptualiser:

(3) CK-314 … uh … is delayed.

(4) CK-314 is on time … uh … is delayed.

(5) CK-314 is on time … uh … CK-314 is delayed.

Example (3) is an covert correction, because the conceptualiser noticed the change before the predicate *is on time* was verbalised, i.e. left the formulator. (4) presents a correction that is produced if the formulator was occupied generating the utterance while the correction arrived from the conceptualiser. (5) is generated when the formulator has already finished producing the utterance and is waiting for the increments for the next one. Thus, which utterance is generated depends on the state the formulator is in when the the change in the environment is spotted and the correction is generated. Since conceptualiser and formulator work in parallel and independently from each other, a flexible format for the preverbal message is needed. In this paper, we will show how an underspecified representation of the preverbal message can fulfil all the conditions that are necessary so that the formulator produces (3)–(5) depending on the point in time of the correction.

The structure of the paper is as follows. First we describe the semantic underspecification formalism CLLS. After that we discuss different causes for self-corrections and introduce conceptual changes. Then we present the model INC in some detail and show how it generates self-corrections. After that we give a detailed account of how self-corrections can be dynamically inserted into the ongoing message generation process without the need for a restructuring of the preverbal message generated up to that point, and how the underspecification of CLLS can be exploited for this purpose. From the means provided by CLLS we use the *parallelism constraint* that has mainly been used for the description of VP ellipses.

2 Constraint Language for Lambda Structures (CLLS)

CLLS is a formal framework for the partial description of lambda structures. Lambda structures are represented as ordinary trees amended by the two partial functions lam for binding variables of the λ-term and ante for modelling anaphoric expressions. The lambda term $\mathrm{Mary}(\lambda x.\mathrm{sleep}(x))$, for instance, can be represented as the tree structure indicated by figure 1.[1]

[1]Note that NPs including PNs are type-raised. Hence, the term Mary in $\mathrm{Mary}(\lambda x.\mathrm{sleep}(x))$ is a function from sets of entities to truth values.

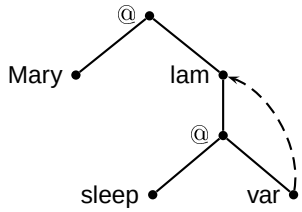

Figure 1: The CLLS representation for $\mathrm{Mary}\lambda x.\mathrm{sleep}(x)$

The tree possesses further labels and lines: the label @ indicates application, a dashed line between two nodes labelled by lam and var ensure the correct binding between variables and the λ-abstractor. Additionally, variables denoting tree nodes (e.g. $X_0, X_1, \ldots$) are added to the lambda tree structure in order to allow for underspecification, the specification of anaphoric references etc. Formally, a CLLS formula is described as a conjunction of atomic literals. In order to satisfy such a formula a lambda tree structure such as the one in figure 1 and variable assignments have to be found such that every literal is satisfied.

Several different constraints are defined in CLLS. Crucial for the definition of underspecification is the dominance relation holding between tree nodes: $X \lhd {}^*Y$ is satisfied iff X denotes an ancestor of Y in the lambda structure. The constraint graph indicates this relation via the dotted lines, e.g. the relation between the nodes labelled by Y_1 and Y_2 in figure 2. Note that the dominance relation is reflexive and transitive. Hence, nodes connected via a dotted line can either be the same node, or an infinite number of further tree structures can be inserted between these two nodes. Another constraint ensures that the binding between variables and lambda operators is given: $\lambda(X) = Y$ is satisfied iff the denotation of X maps to the denotation of Y. Within the constraint graph the mapping is indicated by the dashed line pointing from the variable to the lambda operator (lam).

Most important for the present paper, however, is the definition of the *parallelism constraint*. The parallelism constraint defines a parallel structure between tree segments. Segments in a lambda structure are defined as X/Y where X denotes the root of the segment and Y a hole such that $X \lhd {}^*Y$. The segment covers all nodes that are dominated by the root X with the exception of the node Y and all nodes dominated by Y. In other words, a segment is a sub-tree starting with the node X with the exception of a further subtree which has Y as root node. For instance, in figure 2 the segment X_1/X_2 has the root node X_1 including all nodes dominated by it apart from node X_2.The actual parallelism constraint $X_1/X_2 \sim Y_1/Y_2$ is satisfied iff the segment X_1/X_2 of the lambda structure is parallel to a segment Y_1/Y_2. The segments are described by brackets in the constraint graphs (see figure 2). Formally, the parallelism between two segments is captured via a correspondence function which is defined as a bijective mapping between the two segments, see Erk (2000) for further details.

The parallelism constraint has proven to be especially useful for the description of VP ellipses, as in the following example sentence, cf. figure 2:

(6) Mary sleeps and John does, too.

The parallelism constraint $X_1/X_2 \sim Y_1/Y_2$ is reflected in the graph via two brackets denoting the two parallel segments (i.e. X_1/X_2 and Y_1/Y_2). The brackets precisely determine the part of the source sentence ('Mary sleeps') that has to be copied into the target sentence ('John does, too') as well as the part that has to be kept separate (Mary and John). A lambda tree structure that satisfies the constraint graph in figure 2 is given in figure 3.

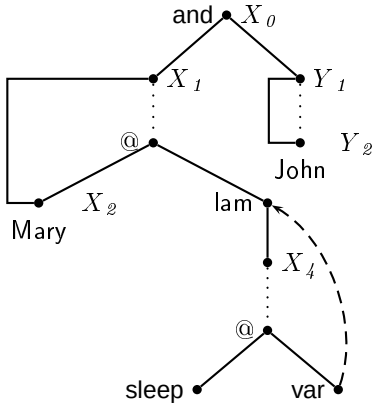

Figure 2: The parallelism constraint for the elliptical sentence in (6)

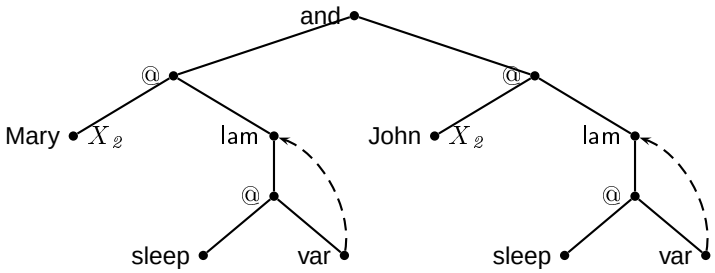

Figure 3: A lambda structure that satisfies the constraint graph in figure 2

A CLLS representation can not only be given as constraint graph but also as a linear constraint that is a conjunction of constraint literals. A CLLS constraint

φ is defined as follows: $\varphi ::= X : f(X_1, \ldots, X_n) \mid X \lhd {}^*Y \mid \lambda(X) = Y \mid \mathrm{ante}(X) = Y \mid A \sim B \mid \varphi \wedge \varphi'$.

3 Generation of self-corrections

For the purpose of this paper, we distinguish two types of causes that can lead to a self-correction: *performance errors* and *conceptual changes*. Performance errors occur either within the formulator, e.g. a wrong lexical access such as in example (7), or within the conceptualiser, e.g. misconceptions such as in (8).

(7) I went to the toast office … uh no … post office.

(8) Whales are fish … uh no … mammals.

The means to detect performance errors is self-monitoring, i.e. the generated utterances are fed back into the system, analysed by the language comprehension component, and compared to what the system had planned (Levelt 1989). If the system generates 'toast office' instead of 'post office' this can be detected by comparing the generated 'toast office' to the planned 'post office', noting that it differs and computing an appropriate correction that results in something like: 'toast office … uh no … post office'.

While performance errors can be attributed to errors in the functioning of the system, e.g. to a faulty lexical access or a flawed inference rule, this is not possible in the case of conceptual changes. The reason is that the cause lies outside the system: the internal representation of the external states of affairs may change during utterance production. For example, (2) may be generated when one seat is reserved by another ticket reservation system during utterance generation. Therefore, it is impossible to solve the problem of conceptual changes for a system, be it as perfect as it may. Put differently, conceptual changes are triggered by perceived input data, not by self-monitoring.[2]

Obviously, systems that do not obtain any further information after utterance planning commences do not get this problem. However, systems in fast changing environments have to deal constantly with new input data, updates, and changes. In order to keep the user up-to-date such systems must have the ability to immediately react to changes, which includes the generation of self-corrections; otherwise a system may even generate contradictory output as we saw in (1).

The causes for self-corrections are usually assumed in the relevant literature, e.g. Levelt (1983), are the performance errors. However, since conceptual changes make self-corrections necessary even in a perfect system, we concentrate on this cause type. We have no doubt, though, that our proposal can offer a solution for dealing with performance errors, as well.

In general, the following steps are necessary to perform a self-correction. At first, a performance error or a conceptual change is detected (*error detection*).

[2] Conceptual *errors* may not only be detected by self-monitoring but also by further inferences the conceptualiser performs while generating the utterance. We do not discuss this possibility here. However, in this case the only difference to conceptual changes is how the need for a self-correction arises, not the way it is treated afterwards.

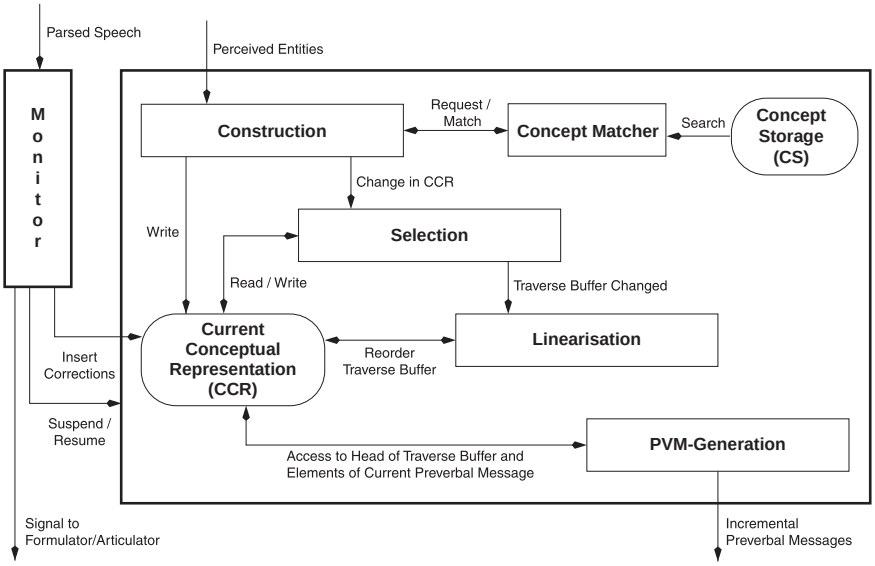

Figure 4: The incremental conceptualiser INC

The part of the utterance that has to be corrected is marked as the *reparandum*, e.g. 'fish' is the reparandum in (8). As soon as an error is detected the current generation is immediately interrupted (*interruption*). Then, a correction term, e.g. 'uh' or 'no', is uttered (*correction term generation*). Finally, the content to be corrected, i.e. the information difference, is given to the formulator (*correction*).

4 Self-corrections with INC

In the first part of this section we present INC, the *incremental conceptualiser*, and describe those components that are involved in the generation of an *incremental preverbal message*. After that the generation of a self-correction caused by a conceptual change is discussed in more detail.

4.1 The incremental conceptualiser INC

The input of INC (cf. figure 4) consists of *perceived entities* and of *parsed speech*. Its output are *preverbal messages*, which is Levelt's (1989) term for semantic representations. One should be more precise at this point, however: they are not simply preverbal messages but *incremental* preverbal messages, because they are not generated as a whole but in a piecemeal fashion, i.e. increment by increment, cf. Guhe (under review).

Besides the main part in which the message generation proper takes place – this is the right, bigger box of figure 4 – INC also contains a *monitor*. It performs func-

tions laid out by Levelt (1989) in order to detect performance errors: it compares the parsed speech it obtains from the language comprehension component with the planned utterance(s). When it detects a (significant) deviation it temporarily suspends operation of the message generation part, sends a signal to the rest of the generation system (Hartsuiker and Kolk 2001), and initiates a self-correction. Since the monitor is necessary only for the detection and correction of performance errors it plays no role in the correction of conceptual changes. For this reason, we will not use it in the following.

The message generation part consists of the four main processes *construction*, *selection*, *linearisation*, and PVM-*generation*. These four processes operate on the *current conceptual representation* (CCR), the internal representation of external states of affairs. The CCR is a hierarchically structured network representation of concepts. It is built up by the construction process with the help of the *concept matcher*, which is the interface to the *concept storage* (CS). The CS contains rules on how to construct complex concepts from simpler concepts. Initially these simpler concepts are perceived entities; later on, the constructed (more complex) concepts can be combined to even more complex ones. That is, the construction algorithm works recursively until (1) no more complex concepts can be constructed, or (2) a newly perceived entity arrives and must be handled by construction and concept matcher.

The selection process chooses the concepts to be verbalised from the CCR, linearisation brings these concepts into an appropriate order,[3] and PVM-generation incrementally produces preverbal messages for them. The latter three processes operate mainly on a sub-structure of the CCR, the *traverse buffer*[4]. The traverse buffer is an array of fixed length and selection appends the selected nodes to this buffer. PVM-generation takes out the first element of the traverse buffer (called *head of traverse buffer*) after a latency, decides upon how this concept is to be verbalised, and hands it on to the formulator. During the latency linearisation can reorder the concepts in the buffer and selection can replace concepts with other concepts. The *current preverbal message* is finished when PVM-generation takes the next head of traverse buffer so as to start a new preverbal message. Until a new preverbal message is begun the current preverbal message can be extended.

The concepts in the traverse buffer are complex concepts, cf. Guhe (under review). PVM-generation hands on the head of traverse buffer as first increment of a new preverbal message to the formulator. However, not all information of the complex concept is needed for the preverbal message. Therefore, PVM-generation has to decide on an appropriate description of the concept. In example (4) there may be more detailed information available about flight CK-314, but PVM-generation decides on the appropriate information for the verbalisation with respect to the

[3] Note that this is a linearisation of utterances, not of phrases. Linearisation of phrases takes place in the formulator, cf. De Smedt (1990).

[4] The name traverse buffer stems from the fact that we regard all nodes of the CCR that were verbalised and all nodes that are selected for verbalisation but are not yet verbalised as a temporally ordered path through the network. This path is called the *traverse*. The traverse buffer is that part of the traverse that contains the nodes selected for verbalisation but that are not yet verbalised, i.e. the nodes that can still be changed.

current situation. In our example only the plane's flight number is chosen, while other information about the plane (plane type, number of seats, etc.) is omitted.

However, this is only the first increment, not a complete preverbal message. Since the CCR and therefore also the head of traverse buffer are nodes in a network representation, this concept is linked to other concepts. PVM-generation follows these links (in some cases recursively) until a complete preverbal message is generated. As soon as another concept becomes available it is handed on to the formulator. Since the way increments of a preverbal message are computed by INC depends on the structure of the CCR, the details are left out here. They can be found in Guhe (under review).

As soon as an increment is sent to the formulator it is inaccessible to the conceptualiser. The only way to change information afterwards is to generate corrections. In particular, selection and linearisation have no further access to the head of traverse buffer after this, because it is no longer part of the traverse buffer. For instance, if selection were now to decide to verbalise a different concept, a self-correction is the only option to convey the information. Note that this would be a correction of a whole preverbal message, not an 'intra-preverbal message' correction such as we are concerned with here.

Note that due to the incrementality of the whole system the preverbal message is also generated incrementally, see Guhe (under review). We therefore adopt the term *incremental preverbal message*.

4.2 Generating self-corrections for conceptual changes with INC

INC detects conceptual changes not with the help of the monitor, because they are not performance errors. Note that this kind of correction can be generated completely without the help of the monitor, because (1) the error is not detected by monitoring parsed speech and (2) all information about how far the generation of the incremental preverbal message has proceeded is available in the conceptualiser.

A self-correction in the case of a conceptual change is generated as follows:

1. Each time new information arrives the construction process integrates it into the CCR, i.e. it changes the CCR.

2. When PVM-generation notices a significant change in one of the concepts in the current preverbal message, a self-correction is initiated.

3. An interruption signal is sent to the formulator.[5]

4. The difference between planned and actual utterance content is computed and a correction increment is generated, i.e. inserted into the incremental preverbal message according to the parallelism constraint. The correction increment contains information about:

[5]This signal is not part of the preverbal message, because it is contains no semantic content but simply tells the formulator that a correction will follow. The signal is sent, because it may withhold the formulator from generating wrong output, e.g. in (4) the signal may have the effect that 'time' is not generated. This depends on how far the formulator has progressed.

(a) which concept to change

(b) what information is to be deleted by the formulator

(c) what information is to be added by the formulator

When a correction increment is generated, it is the duty of the formulator to compute exactly how the correction is to be treated: this depends on how much of the utterance the formulator has already generated. This division of labour is consistent with models of the formulator, e.g. De Smedt (1990).

PVM-generation performs this four-step computation each time a concept belonging to the current preverbal message changes. However, when more than one concept is changed at a time the computation and the correction to be performed get more complicated, and INC may decide to break off the preverbal message and start anew.

5 Underspecification for the incremental generation of self-corrections

In the following we focus on the representation of the incremental preverbal message and present a formalisation of the generated increments in CLLS.

5.1 Incremental preverbal messages in CLLS

INC forges a stream of semantic increments out of concepts from the CCR. We show now how an underspecified semantic representation in CLLS is incrementally generated by INC. Consider as an example the case that INC decides to generate an utterance about a plane's departure time, e.g. 'CK-314 is on time'. The preverbal message for this utterance holds the proposition: $on\_time(CK\text{-}314)$.

INC feeds the actual proposition $on\_time(CK\text{-}314)$ to the formulator with the following increments: $[I_2 : [$*noun semantics*$], I_1 : [$*verb semantics*$]]$. The first increment of the preverbal message is the *verb semantics*, because a proposition about the plane's current *status* is to be uttered. In case something about *CK-314* is to be uttered, the *noun semantics* would be the first increment. Obtaining the first increment from INC, the formulator starts generating an utterance. The *verb semantics* is encoded as a CLLS constraint:

$$[X_0 : @(X_1, X_2), X_2 : \mathsf{lam}(X_3), X_4 : @(X_5, X_6), X5 : \mathsf{on\_time},$$
$$X_6 : \mathsf{var}, \lambda(X_2) = X_6, X_3 \lhd^* X_4]$$

Here, the CLLS constraints are written as a list of constraint literals as defined earlier. Adding further increments means that the lists are concatenated. Hence, at any time we have a clause of constraint literals representing a concatenation of literals. The subsequent *noun semantics* is represented as follows:

$$[X_1 \lhd^* X_7, X_7 : \mathsf{CK{-}314}]$$

The graph representation is given in figure 5, where we reordered the increments so that they do not reflect the temporal order in which they were generated but instead follow the standard way of semantic representations. Note that the

CLLS constraint contains more underspecification than the semantic representation produced by the corresponding analysis: the *noun semantics* contains an additional dominance constraint. The current incremental preverbal message therefore contains *two* places where further information can be added. If the planning process stopped after these two increments, the sentence 'CK-314 is on time.' would be generated.

However, it is still possible that the conceptualiser decides to add further information about the plane or its departure status. For example, INC could add the information that the plane will depart from gate C-21. The following two increments express this: $I_3 : [X_3 \lhd {}^*X_8,\ X_8 : \mathsf{and}(X_9, X_{10}),\ X_9 \lhd {}^*X_4,\ X_{10} \lhd {}^*X_{11}],\ I_4 : [X_{11} : @(X_{12}, X_{13}),\ X_{12} : \mathsf{depart\_C21},\ X_{13} : \mathsf{var},\ \lambda(X_2) = X_{13}]$. (The predicate depart_C21 is clearly a simplification.) These two increments can be found in figure 6.

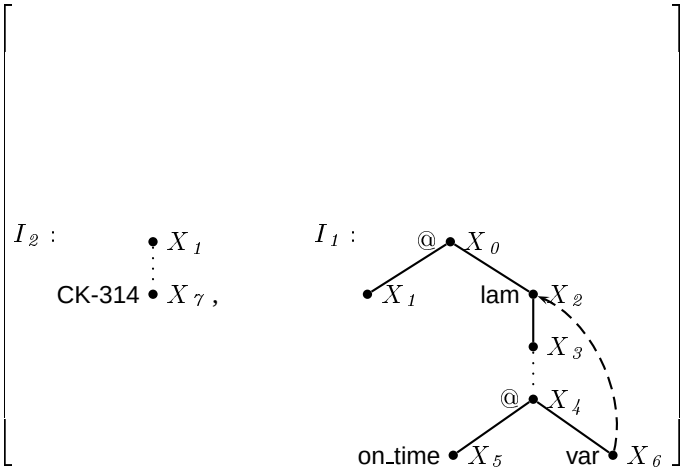

Figure 5: The increments for 'CK-314 is on time.'

5.2 Incremental generation of a self-correction

Levelt (1983) observes that utterances containing a self-correction obey a well-formedness rule. This rule states that a repair $\langle\ \alpha\ \gamma\ \rangle$ is well-formed if and only if there is a string β such that the string $\langle\ \alpha\ \beta$ and$^*\ \gamma\ \rangle$ is well-formed. The string β is a completion of the constituent directly dominating the last element of α.[6]The well-formedness rule ensures that the repair utterance can be described as a coordination. The following example sentences exemplify this rule.

(9) CK-314 departs from gate C-21 … uh … from gate C-22.
$\langle$CK-314 departs from gate C-21$\rangle_\alpha$ or $\langle$from gate C-22$\rangle_\gamma$.

[6]The connective and* is omitted if γ's first element is also a sentence connective.

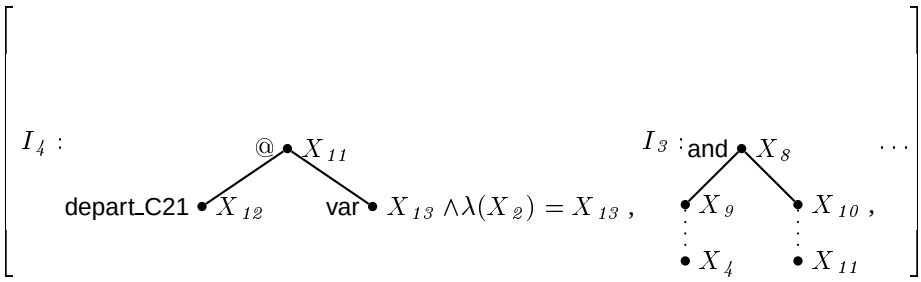

Figure 6: Two increments for '... and departs from gate C-21'

(10) CK-314 departs from ... uh ... from gate C-22.
$\langle$CK-314 departs from$\rangle_\alpha$ $\langle$gate C-21$\rangle_\beta$ or $\langle$from gate C-22$\rangle_\gamma$.

(11) * Flight CK-314 departs 20 minutes later than scheduled from gate C-21 ... uh ... later than scheduled from gate C-22.

This rule is mainly syntactic, even if the choice for the connector may be guided by semantic considerations. The connector *and*, for instance, would sound odd in (9) or (10). Note also that a string β may be added in order to achieve a correctly formed coordination.

Since we deal with the *generation* of self-corrections, the well-formedness rule has to be transformed into a rule appropriate for this purpose. A rule for generating self-corrections has to contain restrictions on how a correction can be built into an already generated incremental preverbal message, i.e. a sequence of CLLS increments. The parallelism constraint in particular will play an important role for the generation of self-corrections.

After INC recognises the need for a self-correction the change is cast into a correction construction in CLLS. Note that the lambda structure for a correction is structurally the same as for a coordination, only that instead of a logical connector such as and or or a function correction is used. After INC has generated the correction, this information, which is simply another increment, is added to the current incremental preverbal message. In a further step the alternation is added as another increment.

These two increments contain the main information needed for the generation of a self-correction. Let I_3' be the correction increment. It specifies the parallelism between the reparandum and the alternation. Node X_9 dominates X_4 (the *reparandum*) and node X_{10} dominates X_{11} (the *correction*), cf. figure 7. In addition, INC also delivers the parallelism constraint for the correction. The correction is explicitly marked by $X_9/X_5 \sim X_{10}/X_{12}$. Node X_5 is labelled on_time, whereas node X_{12} is labelled delayed. Taking all this together, example (4) can be generated with the following increments ($\top$ indicates the root node of the tree):

- I_1' : [[$\top \lhd {}^*X_0, X_0 : @(X_1, X_2), X_2 : \mathsf{lam}(X_3), X_4 : @(X_5, X_6),$

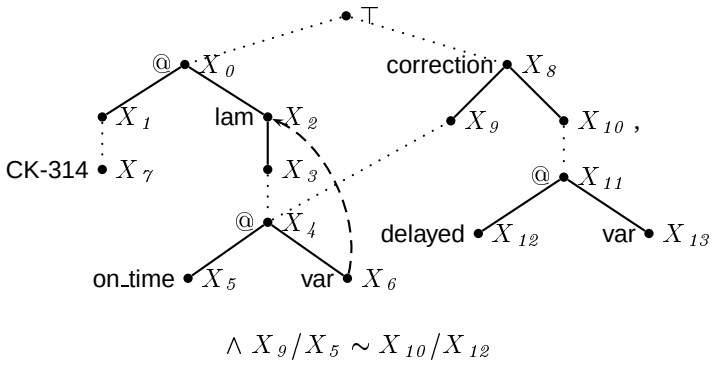

Figure 7: The underspecified preverbal message of the self-correction in (7)

$X5 : \mathsf{on\_time}, X_6 : \mathsf{var}, \lambda(X_2) = X_6, X_3 \lhd {}^*X_4]$
$(= \lambda X.on\_time(X))$

- $I_2{}' : [X_1 \lhd {}^*X_7, X_7 : \mathsf{CK\text{-}314}]$ (= CK-314)
- $I_3{}' : [\top \lhd {}^*X_8,\ X_8 : \mathsf{correction}(X_9, X_{10}),\ X_9 \lhd {}^*X_4,\ X_{10} \lhd {}^*X_{11}], X_9/X_5 \sim X_{10}/X_{12}$ $(= correction(\_,\_))$
- $I_4{}' : [X_{11} : @(X_{12}, X_{13}),\ X_{12} : \mathsf{delayed},\ X_{13} : \mathsf{var}]$ $(= delayed(X))$

Two observations regarding these increments can be made. First, increment $I_4{}'$ contains no binding constraint as it is the case for I_4 (i.e. $\lambda(X_2) = X_{13}$). Instead, a parallelism constraint $X_3/X_5 \sim X_{10}/X_{12}$ is added. The parallelism constraint is based on the computation that INC carried out when triggering the correction and specifying the concepts to be changed. It also ensures that a binding constraint is introduced later because of the bijective mapping of the two segments X_9/X_5 and X_{10}/X_{12}.

Second, the correction node X_8 is subordinated under the root node $\top$. A coordination such as in 'CK-314 is on time or delayed' would have a more restrictive constraint for I_3 (i.e. $X_3 \lhd {}^*X_8$). Why does the correction increment contain a different constraint? Remember that the generation proceeds incrementally in different components at the same time. At the time the correction is produced by INC the increments representing the previously generated concepts have already been fed to the formulator. INC does not have any information about the current state of the formulator. Therefore, it is equally possible that the entire sentence has already been uttered or that no syllable has been generated at all. Hence, the placement of the correction node in the preverbal message has to be *underspecified*.

Figure 7 shows the entire constraint graph for the four increments $I_1{}'$ to $I_4{}'$. This graph indicates that the current incremental preverbal message is underspecified with respect to the lambda structure that can satisfy the given constraints. Two lambda structures are conceivable:

1. CK-314$(\lambda x.correction(on_time(x), delayed(x)))$
2. $correction($CK-314$(\lambda x.on_time(x)),$ CK-314$(\lambda x.delayed(x)))$

The two lambda structures are equivalent, because both can be beta-reduced to the same term: $correction(on\_time(\mathrm{CK}-314), delayed(\mathrm{CK}-314))$. However, each lambda structure possesses a different linguistic surface structure:

(12) CK-314 is on time … uh … is delayed.

(13) CK-314 is on time … uh … CK-314 is delayed.

The two lambda structures reflect two self-corrections that are conceivable for the given example according to Levelt's well-formedness rule. Since INC does not have access to the processing state of the formulator, an underspecified CLLS constraint has to be generated. Depending on how much progress has already been made by the formulator the self-correction in (12) or (13) is generated. If both corrections are still possible, because only 'CK-314' has been produced, the formulator prefers low attachment of the correction node. Low attachment results in generating (12).

Two further linguistic surface structures are conceivable:

(14) CK-314 … uh … is delayed.

(15) CK-314 is on time … uh … delayed.

In (14) the conceptual change is noticed before a statement about CK-314 is uttered. Such a covert correction is also possible with the underspecified semantic representation. Here, the formulator is supplied with the correction node before the VP 'is on time' is generated. Consequently, this part of the semantic representation is not generated, and only a hesitation ('uh') is produced. The variation (15) from (12) can be generated when the conceptual change has been noticed right after the copula 'is' was uttered. The formulator then decides whether the copula has to be repeated or not.

6 Conclusions

We presented an account of how underspecified representations can be employed in an incrementally working system for the generation of self-corrections. We used the *incremental conceptualiser* INC to generate underspecified incremental preverbal messages. Their increments are represented as underspecified lambda structures. The underspecified incremental preverbal messages can be extended by further increments at precisely defined locations in the structure. This modification of the preverbal messages is in particular useful for the generation of corrections.

Our model and the underlying formalisation is an improvement over current approaches to language generation. First of all, an incremental system can speed up the production of utterances. Second, the underspecified representation of increments is flexible and can be monotonically extended. Third, the generation of self-corrections is an important feature to improve the acceptability of dialogue

systems, because changes in the environment can be conveyed to the user quickly and appropriately, even avoiding cases in which contradictory output would otherwise be generated. This also increases the naturalness of the output.

While the current work focuses on the conceptualisation process carried out by INC, future work will include the question of how an incremental formulator can produce natural language utterances from an underspecified representation, as defined by the present paper.

References

De Smedt, K.(1990), *Incremental Sentence Generation: A Computer Model of Grammatical Encoding*, PhD thesis, Katholieke Universiteit te Nijmegen. NICI Technical Report 90-01.

De Smedt, K., Horacek, H. and Zock, M.(1996), Some problems with current architectures in natural language generation, *in* G. Adorni and M. Zock (eds), *Trends in Natural Language Generation: An Artificial Intelligence Perspective*, Springer, New York, pp. 17–46.

Egg, M., Koller, A. and Niehren, J.(2001), The constraint language for lambda structures, *Journal of Logic, Language, and Information* **10**, 1–29.

Erk, K.(2000), Die Verarbeitung von Parallelismus-Constraints, *Informatik 2000 – 30. Jahrestagung der Gesellschaft für Informatik*, Springer, Berlin.

Guhe, M.(under review), Incremental preverbal messages.

Guhe, M. and Habel, C.(2001), The influence of resource parameters on incremental conceptualization, *in* E. M. Altmann, A. Cleeremans, C. D. Schunn and W. D. Gray (eds), *Proceedings of the 2001 Fourth International Conference on Cognitive Modeling: July 26–28, 2001, George Mason University, Fairfax, VA*, Lawrence Erlbaum, Mahwah, NJ, pp. 103–108.

Guhe, M. and Schilder, F.(2002), Underspecification for incremental generation, *Proceedings of KONVENS 2002, 6. Konferenz zur Verarbeitung natürlicher Sprache*, Saarbrücken.

Hartsuiker, R. J. and Kolk, H. H.(2001), Error monitoring in speech production: A computational test of the perceptual loop theory, *Cognitive Psychology* **42**, 113–157.

Kempen, G. and Hoenkamp, E.(1987), An incremental procedural grammar for sentence formulation, *Cognitive Science* **11**(2), 201–258.

Levelt, W. J.(1983), Monitoring and self-repair in speech, *Cognition* **14**, 41–104.

Levelt, W. J.(1989), *Speaking: From Intention to Articulation*, MIT Press, Cambridge, MA.

Reiter, E.(1994), Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible?, *Proceedings of the Seventh International Workshop on Natural Language Generation (INLGW-1994)*, Kennebunkport, Maine, USA, pp. 163–170.