Boosting LUCK

Improving the language understanding capabilities of Kaitito

Master's Thesis for
Human Media Interaction

by

M.P.J. Penning

submitted November 2007

Supervisors

dr. A. Knott
P. Vlugter

prof. dr. ir. A. Nijholt
dr. ir. H.J.A. op den Akker
dr. ir. B.W. van Schooten

Abstract

Kaitito is a natural language system that can be used to have conversations with one or more artificial characters. All natural language systems like Kaitito have limited abilities for understanding user input. The best way to improve Kaitito's language understanding capabilities, seems to be using reformulations (paraphrases) of the user input on the level of semantic representations. This conclusion is based on the specifications of the Kaitito system as well as literature from systems related to Kaitito, like chatbots and question-answer systems.

The Minimal Recursion Semantics (MRS) representations used in the Kaitito system can be paraphrased using transfer rules, also called munge rules. Deriving the transfer rules by hand is quite tricky and time consuming and moreover, it does not support Kaitito's rationale of allowing the system to be authored by non-computer-linguists such as for instance English teachers. Therefore, deriving the transfer rules was automated as much as possible. Although a lot more difficult to implement than deriving the rules by hand, this automated version of deriving the transfer rules will make the system usable for non-computer-linguists.

Using transfer rules involves issues like coverage and precision; rules can be too restrictive, or fire on occasions they should not. The main problem with automating the transfer rule derivation process is matching two MRSs, which can result in multiple, ambiguous possible matches. Even though using paraphrases of MRSs and the semi-automatic derivation of the transfer rules have these problems inherent to them, the (preliminary) results in the domain of people introducing each other look promising.

Preface

Completing a graduation project is a journey, starting with defining the project topic and ending with a report and a presentation. In this case, it also was a journey literally across the world from the Netherlands to New Zealand and back. Such a journey is impossible to make without support from family and friends back home, nor without new friends abroad. So, thank you all! Thanks to you I've been able to learn a lot more than just about Kaitito and the project, because I've been introduced to a new country, new cultures and (let's not forget) new styles of dancing. I want to thank Margaret Gibbs in particular, for having me in her home for five months, (re)teaching me to knit, showing me parts of New Zealand and making me part of her family there.

But the thesis lying in front of you would not have been there without my supervisors. Ali, you guided me from Cluedo to LUCK and it was a lot of fun to have you there every once in a while at the guy's weekly drinks on Fridays. I'm a bit sorry for Peter, because I bugged him with the stupidest of questions concerning CMUCL and Kaitito. But I like to think that in the end I kind of got the hang of it. Anton, I thank you for providing me with the opportunity to go to New Zealand in the first place. Boris and Rieks, you both taught me subjects that made me enthusiastic for computer linguistics and I'm happy I chose my graduation project in that direction. Ali, Anton, Boris and Rieks, you of course all helped me with the evaluation and with writing my thesis, which helped improve my work and for which I am grateful.

Manon Penning, Enschede

Contents

List of Figures

List of Tables

1 Introduction

Over the years, a lot of research has been conducted in the area of conversational systems. Here, conversational systems are considered to be everything from Question Answering (QA) systems used in a real world setting through pedagogical systems like Computer Aided Language Learning (CALL) systems and tutorial dialogue systems to chatbots. Platforms for comparison and evaluation of these systems have been set up in order to facilitate collaboration and progress, such as for instance the Loebner Contest for chatbots (see http://www.loebner.net/Prizef/loebner-prize.html) and the well-known Text REtreival Conference (TREC) for QA systems (see Voorhees and Harman (2005)). From these evaluations, it can be concluded that the limited input understanding capabilities of conversational systems are still a major obstacle in their practical use (Pan & Shaw, 2007; Purver, Ratiu, & Cavedon, 2006).

Efforts have been made to circumvent this problem, for instance, most pedagogical systems and adventure games use multiple choice questions or other menu-based ways to query users (Rosé, Gaydos, Hall, Roque, & VanLehn, 2003; Shaw, Johnson, & Ganeshan, 1999; Zacharski, 2002). Still, there are systems like Thespian (a tactical language training system for soldiers sent on missions in Arabic countries, see Si, Marsella, and Pynadath (2005)) with which users can interact through spoken or written dialogue and games like Electronic Arts' Majestic which gives players the opportunity to interact with characters through the AOL Instant Messaging system (Zacharski, 2002).

In the domain of pedagogical systems, robustness is critical, especially in CALL systems (Angelova et al., 2004; Core & Moore, 2004). The main reason for this, is that these systems are designed to help people acquire a language and getting hands-on experience in using it in a natural way. Most QA systems, tutorials and games are restricted to a certain knowledge domain that fits with their application. On the other hand, the automatic processing of natural language in CALL systems faces the problem of handling more of an unrestricted domain as learners advance. Moreover, an actual conversation entails utterances that can never be responded to using simple direct matching between the words in the utterance and words in possible responses, stored in some sort of knowledge base. This is because the answers only partially match their associated questions, as illustrated below. There are several types of partially matching answer. Sometimes the match between question and answer is determined by particular linguistic constructions in the question (e.g. *How are you doing?* asks for the current mental and/or physical state of the addressee as opposed to a literal mapping onto possible answer sentences with a structure like *I am doing great*, while something like *Fine* or *Good* is more probable. Another possibility is that the partial match relates to underlying inferences (e.g. *Where are your folks from? – My parents are from the Netherlands*). Furthermore, sometimes there is an explicit (often marked) departure from the question (e.g.

Where are your parents from? – Well, my mother is from the Netherlands).

This project is aimed at designing and implementing a method for extending the language understanding capabilities of systems like Kaitito, with the focus on questions with indirect answers. The next section (Section 1.1) elaborates upon the specifics of the Kaitito system. Section 1.2 then formulates the problem the rest of this thesis will focus on. The different levels of processing and representations that can be used to implement a solution for this problem, will be described in Section 1.3. Finally in Section 1.4, the lay-out of the remainder of this thesis is presented.

1.1 Kaitito

Kaitito is a bilingual, natural language processing system for both English and Māori, which is intended to serve as a platform for various types of computational linguistics research (Vlugter, Knott, & Weatherall, 2004; Knott, Bayard, Jager, & Wright, 2002). Amongst other things, it can be used as a CALL system allowing a user to practice dialogues or use it as a simple sentence translator. At this point all functionality of Kaitito is text-based.

The next section roughly describes the general architecture of the Kaitito system. This description is quite shallow, but touches on the things that are important for this project. In Section 1.1.2, the parts of the Kaitito system that are of particular interest to its function as a dialogue system and its language understanding capabilities are elaborated on.

1.1.1 Kaitito's general architecture

The architecture of Kaitito mainly follows the regular design of NLP systems, but has the special features that it incorporates bidirectionality and multilinguality (see Knott et al. (2002) for more details). The bidirectionality entails using the same resources for interpreting and generating texts, the multilinguality inspired the choice of trying to use a single bilingual grammar for both English and Māori.

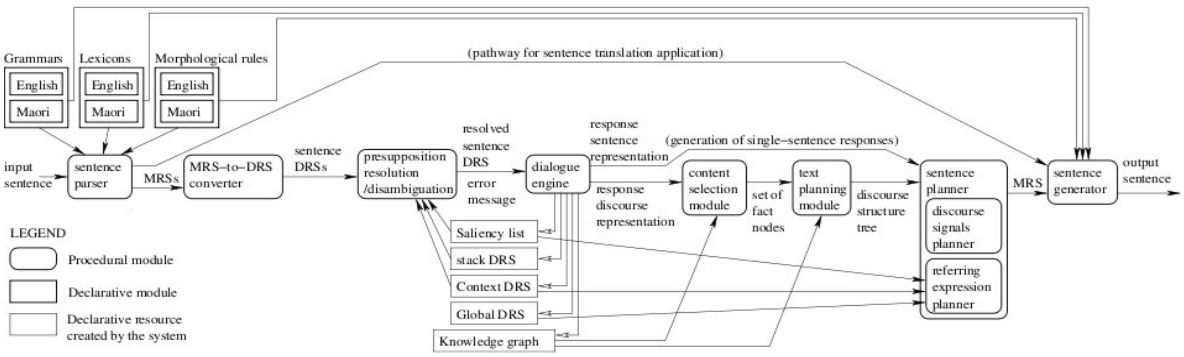

Figure 1.1: Kaitito architecture (Slabbers, 2005, p. 5).

For parsing (and generating) sentences in Kaitito, the Linguistic Knowledge Building (LKB) system by Copestake, Carroll, Malouf, Oepen, et al. (2000) is used. The parser generates representations of the sentences in Minimal Recursion Semantics (MRS, see Section 3.2.1 and Copestake, Flickinger, Pollard, and Sag (2005)), which are either sent directly to

the generator (when the system is purely used to translate sentences) or turned into Discourse Representation Structures (DRSs, see Kamp and Reyle (1993)).

Depending on the context and whether the sentence contains things like anaphora that have to be resolved, the DRS representation of the sentence undergoes a series of other transformations before the sentence planner turns it back into an MRS which is fed to the sentence generator. A graphical overview of the architecture that contains a bit more detail than the above description is depicted in Figure 1.1. This overview does not contain all modules that are present or will be present shortly. The modules that are of significance to this project will be described in the next section.

1.1.2 Kaitito as a dialogue system

Given the fact that this project focuses on improving the language understanding capabilities of Kaitito and the time limitations, it was decided to solely focus on the parts of Kaitito that enable it to act as a dialogue system for practicing English. Kaitito has no database with elaborate built-in world knowledge, but a knowledge base ('knowledge graph' in Figure 1.1) containing all sorts of facts put there by a domain expert. The facts in the knowledge base comprise of information the system can use to start a new topic, break a silence (etc.) or answers to questions the system might get from users. The facts can be general knowledge for all artificial characters, or specific knowledge of a certain character. As mentioned, the knowledge base is supposed to be filled and authored by a domain expert, an English teacher in this case. The idea behind this, is that a domain expert knows exactly what knowledge the characters need to have a conversation about a certain topic. Thus hopefully, the knowledge base can be kept relatively light, because it is not filled with irrelevant (world) knowledge. When the knowledge base is as small as possible, this would make the system more usable as a real-time application. It can be presumed that an English teacher has some linguistic knowledge, but this knowledge will definitely not be excessive. This is an important restriction for methods possibly used to improve Kaitito's language understanding capabilities.

For the purpose of functioning as a dialogue system in a CALL context, a couple of extra modules (some of which have not been fully developed yet and thus not appear in the schematics of the system in Figure 1.1), have been designed in addition to the basic system:

- **Anaphora resolution**
  Solves the anaphora, specifically those relating to dialogues. In dialogues and even more so in multilogues, anaphora used in relation to person denomination (using *you* or *she* when talking to or about a certain computer character for instance) are very important. A module that is quite good at resolving those anaphora is available within the Kaitito system.

- **Diagnosis of and response to student errors**
  Benificial both for the teaching experience of the students (Ham, 2005), and the system itself. Students can be provided with on topic and immediate feedback, whereas the system does not have to try to deeply parse as many wrong sentences that would only result in errors anyway. This feature allows for an important assumption to be made: even though the system is handling input that is prone to errors, it can be assumed that with a working version of this error correction module in place, the input the system actually gets to parse will be free of errors.

- **Generation of (teaching) initiatives**
  Enables the system to keep pushing the dialogue in a certain direction and to keep the conversation flowing (Slabbers, 2005). With the slightly unnatural action of hitting the return key without inputting anything, it allows the student to escape from having to be the next person to say something before the system will respond. This functionality does not really directly relate to the issues in this project.

- **Authoring the knowledge base in natural language**
  Allows domain experts like language teachers with limited linguistic knowledge to design and revise dialogue cases (Knott & Vlugter, 2007). As mentioned, Kaitito does not come with an enormous built-in knowledge base containing a lot of world knowledge. This module allows easy access to the knowledge base and thus it does not need to be filled in advance by linguists.

The Kaitito system is only able to respond with a suitable reaction (the option of choosing a random next utterance set aside), if the question looks a lot like a fact in the knowledge base. For instance, consider the character Ben that was given the information *Your name is Ben* and *You are fine*. Some small example dialogues using this information can be found in Figure 1.2 and Figure 1.3.

```
- Hi.
> What is your name?
- I don't know what is my name.
```

```
- Hi.
> Your name is what?
- My name is Ben.
```

Figure 1.2: Short example dialogues in Kaitito (1/2).

Even though the difference in questions posed to Ben in Figure 1.2 seems to be purely syntactical, somehow the semantic representations of the questions differ in a way that makes the system unable to find the answer in the first case.

```
- Hi.
> Hi.
- My name is Ben.
> How are you, Ben?
- I'm fine.
```

```
- Hi.
> Hi.
- My name is Ben.
> How do you do?
- I'm sorry, I don't know how do I
do.
```

Figure 1.3: Short example dialogues in Kaitito (2/2).

For a human with some understanding of English, it is immediately clear that *How do you do?* in the left-hand dialogue of Figure 1.3 means roughly the same as *How are you?* in the right-hand dialogue and that thus the answer *I'm fine.* would be appropriate to give.

From Figure 1.2 and 1.3 it becomes clear that Kaitito needs some work done on its language understanding capabilities. The next section will describe the goal of this project that flows from this observation a bit more formally.

1.2 Project goal

Kaitito suffers from the same kind of problems other conversational systems suffer from, including the lack of sufficient input understanding capabilities. Even sentences that language learners learn quite early on, can pose a problem for the current Kaitito system. Consider for instance the previously mentioned example of the question *How are you doing?* and a question like *What time is it?* or the more polite *Could you tell me the time please?*. These are questions a language learner will acquire pretty early on, but that are rather hard to answer when trying to find an entirely matching answer.

Of course the ultimate solution is to have a system with general reasoning capabilities and world knowledge. But that approach has serious disadvantages, because it is computationally very heavy, if even possible to make. This makes it unsuitable to use in a real-time system like Kaitito. Other solutions, like controlling the user input to make sure the system only gets questions that have completely matching answers, seem less feasible. After all, users should be allowed to ask questions that have partially matching answers when these are part of their curriculum. Handling questions with partially matching answers also rules out using some sort of system with feedback and clarification questions. As illustrated before, even though questions and answers clearly seem to be related to humans, the system might not see this. The same holds when trying to come up with reasonable clarification questions, which is rather impossible without a good grasp on the intentions behind the original question. As a possible alternative, using pre-defined sets of rules to paraphrase the questions into something Kaitito is more likely to be able to answer comes to mind. The feasibility of this solution is to be investigated, but considering the discussion above, the general problem can thus be stated as:

> How can Kaitito get better at finding answers in its knowledge base that only partially match their corresponding questions, but are relevant to them nonetheless?

1.3 Levels of processing and representations

In the previous section, a couple of different levels of processing and representation were very briefly mentioned, which will be explained here in some more detail. Looking at sentence interpretation, usually four different levels are distinguished:

- **Surface**, concerning the words or perhaps rather the words seen as specific groups of letters
- **Syntactic**, concerning word order and other grammar related issues
- **Semantic**, concerning the actual meaning of a sentence and the dependency relationships between words, as for example semantic graphs (Pan & Shaw, 2004; Mollá, 2006)
- **Inference**, concerning logical reasoning to deduce facts from the meaning of the sentence

Automatic translation might seem like an entirely different track of natural language processing than producing a response to a user utterance in a dialogue system. However, when addressing the issue of partially matching questions and answers (or more general utterances

and reactions), one could consider this as a problem of translating or paraphrasing the original utterance into an utterance that can directly be answered, an idea also brought forward by Iida, Inui, Iwakura, Fujita, and Takahashi (2001). The famous Vauquois triangle (see Figure 1.4 for a slightly adapted version, adopting terms relevant to the current topic) is often used as an illustration for the different paths that can be chosen to transform a sentence from a source language into the target language. The point of this triangle is, that it is hard to travel along the sides of the triangle and easier to cross through the triangle a little way up than down at the bottom.

The ultimate goal for researchers is to find the interlingua; an artificial language that can be used as a sort of international language, a formal representation into which the source language can be translated and from which the target language can directly be generated. Although some authors actually claim that to have found an interlingua (Stymne & Ahrenberg, 2006) for natural language translation, Copestake, Flickinger, Malouf, Riehemann, and Sag (1995) note that an interlingua would require an enormous amount of reasoning. An easy example to prove this is the fact that some sentences cannot be translated from one sentence in another while completely preserving meaning, which would require an interlingua to somehow model the communicative intentions of the speaker. Copestake (1995) therefore proposes a modified Vauquios triangle, depicted in Figure 1.5.

Imagining that going through the triangle is harder than moving upwards and downwards along the sides, using the level of underspecified semantic representations as a point to get across seems very reasonable.

1.4 Overview of this thesis

In the next chapter, relevant research which covers terrains that have some overlap with the question of how to deal with answers partially matching the question will be discussed. Chapter 3 will describe the design of rules to be incorporated in the Kaitito system and Chapter 4 their implementation. The performance of the system is tested and these tests with their results are presented in Chapter 5. Finally, Chapter 6 describes the conclusions and offers some issues for discussion.

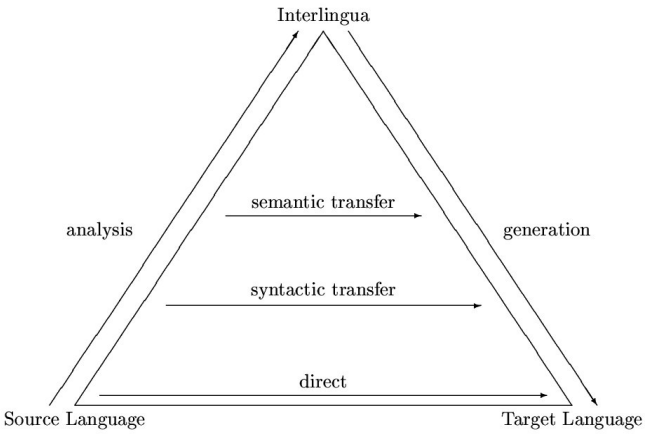

Figure 1.4: Vauquois triangle (Copestake, 1995, p. 4).

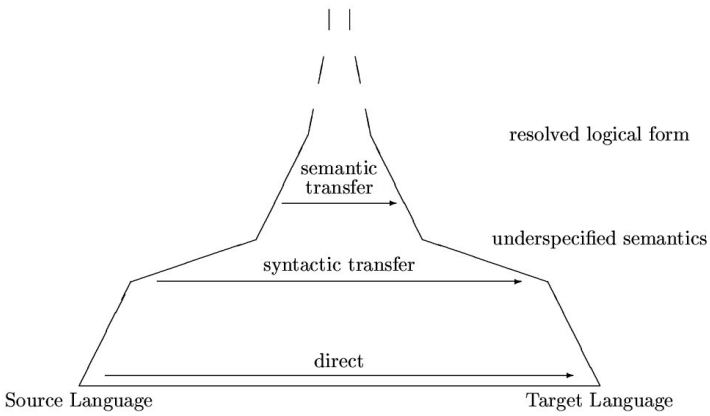

Figure 1.5: Modified Vauquois 'triangle' (Copestake, 1995, p. 5).

2 Paraphrasing and response techniques

With the limited input understanding capabilities of conversational systems being a critical problem in conversational systems, many attempts have been made to somehow improve these capabilities. The methods devised for improving the input understanding capabilities vary with the type of application they are designed for as the next sections dealing with different fields of related research will show.

Since the focus of this project is on the Kaitito system, possible solutions must fit within the Kaitito framework and rationale. Kaitito is a practical application, which for instance means that it simply has not got the possibility to engage in complex analysis and reasoning in order to come up with an appropriate utterance. At the same time, Kaitito has to deal with input that is in some ways more complex than for example the input that QA systems developed for TREC have to deal with. For one thing, as all CALL systems, it is more likely to encounter out-of-vocabulary words or extra-grammatical utterances (Core & Moore, 2004). As described in Section 1.1.2, within Kaitito incoming (English) sentences are always parsed using the ERG grammar before being processed into an MRS, a semantic representation. This MRS is then used to find a suitable response, which is also in the form of an MRS, that needs to be generated back to a natural language sentence. Using these steps, Kaitito is unable to parse grammatically incorrect sentences. Soon there will be a feedback module that will try to correct wrong sentences, so on all levels higher than surface level, within Kaitito it can be assumed that sentences are correct.

The following sections describe a range of methods from basic, surface-based methods through paraphrasing all the way to deep parsing and analysis. Basically, all these methods either try to match an input utterance with an appropriate output utterance or match the input utterance with another utterance which is then used to find the appropriate output utterance (a question is transformed into another question). These two mapping methods combined with the levels presented in Section 1.3, strictly make eight spaces for transformations to take place as shown by the big black dots in Figure 2.1. In reality of course, these spaces are on a continuum.

In the following sections, the content of the rectangles in Figure 2.1 so to speak, will be elaborated upon. First, the surface-based pattern matching approach will be described, looking at various systems in which it is used. Secondly, in Section 2.2, the range of methods based on paraphrasing are looked into. Section 2.3 describes the way in which external resources such as the Web and WordNet are used in both surface-based and paraphrase approaches and Section 2.4 describes some methods involving deeper parsing and a better understanding of the semantics of utterances. Finally, in Section 2.5 it is decided which approach is most suitable for Kaitito.

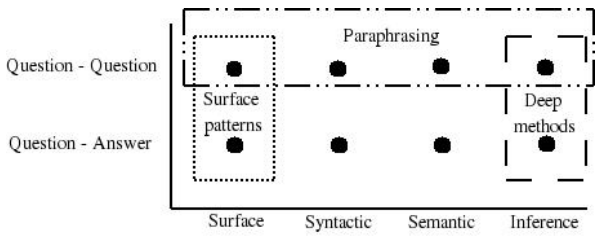

Figure 2.1: Transformation spaces.

2.1 Surface-based pattern matching

As an alternative for parsing sentences using syntactic rules to try to distinguish the question target and answer target, systems based on simple pattern matching have been developed (Hu, Wenyin, Chen, Chen, & Li, 2006). There are different kinds of simple, surface-based pattern matching methods. The ones mostly used are based on simple word or phrase patterns, but sometimes other features are incorporated as well. In the following two sections (2.1.1 and 2.1.2) these types of patterns will be explored, after which the acquisition of patterns and the problems encountered when using surface-based pattern matching are dealt with in Section 2.1.3 and Section 2.1.4 respectively.

2.1.1 Word and phrase patterns

Games with user interaction usually use canned text and so does EA's Majestic, but it also reacts to certain words and phrases the user is likely to use. Zacharski (2002) notes that people have lower expectations of agents for gaming than they do for agents designed to book an airline reservation, which is supposed to be the reason Majestic and other games get away with this type of implementation but practical applications would not. Still, a very similar approach was used in the famous early chatbot ELIZA, which managed to fool some people in believing she was human (Weizenbaum, 1966). More modern variants of ELIZA like A.L.I.C.E. and Jabberwacky use thousands or even millions of patterns (Alfonsi, 2006; Fryer & Carpenter, 2006; Wallace, 2000).

Also in QA systems this approach has proved to be very effective; the winner of TREC-2001, the InsightSoft-M system by Soubbotin and Soubbotin (2001), used only patterns based on word and phrase matching. For example for questions like *When did Jimi Hendrix die?* patterns were constructed as potential answer expression, like (Soubbotin & Soubbotin, 2001, p.2):

```
capitalized word; parenthesis; four digits; dash; four digits; parenthesis
```

Of course, this is only one of many variations as utterances like *Jimi Hendrix died in 1970* or *Jimi Hendrix died on September 18, 1970* are quite often used instead of *Jimi Hendrix (1942-1970)* and these are not covered by the aforementioned pattern.

A major advantage of these types of systems, is that they (depending on the level of detail in the patterns used) can handle ungrammatical sentences, sentence fragments and many misspellings (Angelova et al., 2004; Gockley et al., 2004; Jijkoun, Mur, & Rijke, 2004). A disadvantage is, that just matching words and phrases does not seem to be enough to

cover a sufficient portion of the possible variations in language. Therefore, several researchers have tried to adapt these simple patterns somewhat to incorporate a number of features, as described in the next paragraph.

2.1.2 Patterns involving other characteristics

Most state-of-the-art QA systems use slightly more complicated patterns involving other text features than just the words and phrases themselves. Amongst these features are:

- Starting positions of words or phrases (Pan, Shen, Zhou, & Houck, 2005)
- Length of a word or phrase (Pan et al., 2005)
- Part-of-speech tag a word or phrase has (Pan et al., 2005; Ravichandran & Hovy, 2002)
- Synonyms and hyponym relations or glosses, extracted from sources such as WordNet or even from the source texts themselves (Hearst, 1992; C. Lin, 2002; Marton, Tellex, Fernandes, & Katz, 2005, a.o.)
- Possible syntactic attachments to other words or phrases (Pan et al., 2005)
- Statistics describing co-occurrences of certain words or n-grams of words (Jijkoun et al., 2004)

Some researchers would argue that using these features would make it something else than simply surface-based pattern matching and instead go into the realm of linguistic-based systems. After careful examination of both types of systems however, Hu et al. (2006) conclude that there is no essential difference between the two approaches. All that the 'original' surface-based patterns are, according to them, is instantial versions of the syntax rules and looking at it the other way around, syntax rules are generalized language patterns. This makes the instantiation degrees the only difference. For similar reasons, Jijkoun et al. (2004) use the term lexio-syntactic patterns rather than surface-based patterns.

It is interesting to note that on some occasions systems that do use more complicated types of surface-patterns, seem to be unable to beat simpler systems as the runner up of TREC-2001, the LCC system (Harabagiu et al., 2001), was in beating the InsightSoft-M system.

2.1.3 Acquisition of surface-based patterns

At first the surface-based patterns were simply based on all different formulations the developers could think of, like in the early chatbot ELIZA (Weizenbaum, 1966) and in the first QA systems using surface patterns, as the one by Soubbotin and Soubbotin (2001). Apart from this being a very tedious, time consuming endeavor in a domain where many formulations are possible (Ravichandran, Ittycheriah, & Roukos, 2003), humans tend to overlook many perfectly suitable reformulations (D. Lin & Pantel, 2001). In order to (partly) overcome these problems, (semi-)automatic learning of surface patterns has been implemented into several systems.

The chatbot Jabberwacky learns its conversational patterns on its own from interactions with previous users. Jabberwacky does this by simply storing answers and reactions that users give and rate their likelihood to be used by the number of times different users react

in a similar way. Using this method, Jabberwacky has learned about 8 million conversational patterns (Alfonsi, 2006; Fryer & Carpenter, 2006). A similar approach is used in the tutoring system AutoTutor (Graesser et al., 2000), which tries to simulate dialog moves from human tutors based on previous dialogues between students and human tutors.

In their effort to make a translation system that automatically acquires translation rules, Seneff, Wang, Peabody, and Zue (2004) use example translations of sentences, try to generalize them and apply these generalizations to sentences slightly different from the original example. Greenwood and Saggion (2004) have developed an acquisition algorithm for surface based patterns used for answering factoid questions, which to a large extent does the same thing. Their algorithm is applied to twenty example questions, of which the question and answer term are used to find instances of answer sentences using Google. After setting appropriate sentence boundaries and replacing a number of types of words (apart from the question and answer term themselves, also things like dates and locations are generalized to their concepts rather than the instantiations that naturally occur in the retrieved answer sentences), they sequentially test the patterns thus collected for their degree of appropriateness, considering the number of occurrences of their particular construction in another set of twenty question-answer pairs. IBM's statistical QA system also assigns weights to instances of patterns using their frequencies (Ittycheriah & Roukos, 2002), as does the system by Shen, Kruijff, and Klakow (2005).

Ravichandran et al. (2003) warn that the approach to train weights can suffer from noise induction when used on an unsupervised collection, but when used on a supervised collection the the problem of having very little training data as compared to the number of features becomes an issue. Another thing that requires attention is how most systems try to generalize the patterns found. Because overgeneralizing can lead to inappropriate answers, it is important to pinpoint the suitable level of instantiation for the patterns and generalize them as far as accordingly appropriate (Hu et al., 2006).

2.1.4 Problems with (surface-based) patterns

Even though research on surface-based patterns has come a long way and is fairly successful, there seem to be a number of problems inherent to the surface-based pattern matching approach that cannot be easily dealt with. Among these are:

- Surface-based patterns cannot easily be generalized for answering more difficult questions to which the answers do not appear directly in texts, like answers to exploratory questions as *What can you tell me about the war in Iraq?* (Galea, 2003; Leveling, 2006; Nyberg & Frederking, 2003)

- Most pattern-based systems take years to engineer and are very difficult to duplicate and reuse, contrary to some other systems like the statistical ones (Ravichandran et al., 2003)

- The set of patterns is never complete, which implies recall problems (Soubbotin & Soubbotin, 2001; Jijkoun et al., 2004)

- Surface-based patterns strongly depend on word ordering and distance in the text and are often too specific to the question type, as for instance *Who killed Kurt Cobain?* could look for a named entity very close to words like *killed* or *murdered*, but would find neither a match that is close to the verb nor a named entity in *Kurt Cobain was killed by*

a gunshot wound to the head, which he supposedly inflicted upon himself (Ravichandran & Hovy, 2002; Soubbotin & Soubbotin, 2002; Shen et al., 2005)

- Surface-based patterns do not represent meaning, thus they sometimes yield imprecise or unreliable information, which implies precision problems (Hermjakob, Echihabi, & Marcu, 2002; Hovy, Hermjakob, & Ravichandran, 2002; Jijkoun et al., 2004)

Galea (2003) suggests to store previously answered questions to remedy part of these problems, an approach quite similar to the ones used in Jabberwacky and AutoTutor (see above). These previously answered questions might not match the patterns precisely and thus they can bridge gaps between new questions that are too far away from the existing patterns, but are close enough to previously answered questions. For instance, suppose that the sentence *The yellow-eyed penguins in Dunedin attract many tourists* is in the knowledge base and that the new question is *What is worth seeing in Dunedin?*. This question can never be answered directly unless some sophisticated reasoning system is implemented. However, when the question *What are the major tourist attractions worth seeing in Dunedin?* has been successfully answered before and was added to the indexed questions, the (lexical) link with the new question is a lot more direct.

2.2 Paraphrasing

Apart from the need for anaphora resolution and synonymy, the wide range of possible syntactic surface forms makes finding correct answers quite difficult (Brill, Lin, Banko, Dumais, & Ng, 2001). These different syntactic forms, with different word orderings or indirect answers are exactly the type of things where the above mentioned simple pattern matching approaches seem to fail. In particular when going from factoid questions into the more complicated questions like questions of type "what" and "why", there seem to be a lot of variations in answer formulation (Anaya & Kosseim, 2003).

In this section, paraphrasing is considered as a very broad term. Even though it does not conform to the standard definition of paraphrase (express the same message in different words), here its meaning incorporates paraphrasing at levels beyond the surface level. For instance, it includes 'semantic paraphrasing' which does not concern words anymore, but some sort of semantic representation. Note that by the definition of semantics (meaning) the notion of paraphrasing semantics might seem ridiculous, because things meaning the same thing have the same semantics and thus their semantic representation should not need to be paraphrased. In practice though, the semantic representations have not yet evolved to a point where two things that in fact have the same meaning also get the same semantic representation, so paraphrasing semantic representations can be necessary.

Section 2.2.1 describes the techniques used in paraphrasing and Section 2.2.2 describes the way rules for paraphrasing are aqcuired. The problems that are encountered with paraphrases are dealt with in Section 2.2.3.

2.2.1 Techniques in paraphrasing

Here, paraphrasing will be used to transform one question with a partially matching answer, into a question that completely matches the answer. For doing this, just like with the patterns, in paraphrasing there are simpler types of reformulations at word level. These involve both

simple word permutation and lexical variations like synonyms (Anaya & Kosseim, 2003). Even though paraphrases can happen on all language levels, these rather simple, surface-based reformulations are mostly used (Anaya & Kosseim, 2003; Rinaldi, Dowdall, Kaljurand, Hess, & Mollá;, 2003). Among the general techniques currently used in paraphrasing are:

- Synonyms for important words; *length* instead of *extent* (Echihabi & Marcu, 2003; Hermjakob et al., 2002; Rinaldi et al., 2003)
- Alternate spelling for important words; *New Zealand* or *Newzealand* (Echihabi & Marcu, 2003; Hermjakob et al., 2002)
- Simple query formulation, which preserves quoted terms and smallest noun phrases; *What is the most well-known bird in New Zealand?* becomes *the most well-known bird* AND *New Zealand* (Echihabi & Marcu, 2003; Hermjakob et al., 2002; Ravichandran & Hovy, 2002)
- Simple verb shifting, which generates a rewrite for every possible position of the main verb and leaves out the question word; *Who is the prime-minister of New Zealand?* becomes *the is prime-minister of New Zealand*, *the prime-minister is of New Zealand* and *the prime-minister of is New Zealand*, but also the more probable *the prime-minister of New Zealand is* (Brill et al., 2001)
- Declarative versions of the question; *What is a kiwi?* becomes *a kiwi is* or *a kiwi is defined as* etc., which is especially helpful when searching for definitions in a text that contains descriptive information like an encyclopedia
- Morphological variants; *When did the first settlers come to New Zealand?* has reformulations involving *came* instead of *come* (Echihabi & Marcu, 2003; Hermjakob et al., 2002; Rinaldi et al., 2003)
- Comparatives resolved to their superlatives or other forms; *is better than all others* can be written as *is the best* or vice versa (Rinaldi et al., 2003)
- Expand with potentially occurring units, prepositions and/or discourse cues; *How far is it from Holland to New Zealand* becomes a reformulation that includes things like *miles* or *km*, but also *When did Jimi Hendrix die?* might be turned into *Jimi Hendrix died on* (Echihabi & Marcu, 2003; Hermjakob et al., 2002)
- Subordinate clauses resolved to separate sentences linked by anaphoric pronouns; *The tree healed its wounds by growing new bark* becomes *The tree healed its wounds. It grew new bark.* (Rinaldi et al., 2003), which allows the 'main' part of the original sentence to be used for a query, apart from the rest of the original sentence
- Prepositional phrase attachment or detachment; *a guitar from Jimi Hendrix* can be written as *Jimi Hendrix' guitar* or vice versa (Rinaldi et al., 2003)
- Pattern-matching-based munging of semantic representations; paraphrasing at the level of semantic representations where abstractions of these representations serve as a pattern (Copestake et al., 2005; Copestake, last update 2006)

- Inference of facts; *The souvenir costs $10* or *The price of the souvenir is $10* (Rinaldi et al., 2003)

Note that the first two types can be dealt with using some sort of dictionary that contains synonyms for words and can function as a spellchecker thus allowing for slightly misspelled words. Moving further down the list, more effective parsing and mapping of the underlying structures is needed. The inference of facts even requires a rather elaborate world model and a decent reasoning capability, or the formulation of patterns for many frequently used inferences in a certain domain (Hermjakob et al., 2002).

2.2.2 Acquisition of paraphrases

Whereas it may not be difficult to manually devise rules that account for the most popular ways of paraphrasing a question, variations in the sentences containing the answer are much less predictable. Still, there is a way to derive paraphrases even though a lot of them might be unpredictable for humans, because the rules can automatically be learned based on a representative corpus of questions and answers (Mollá & Zaanen, 2005).

Just as in surface-based pattern acquisition, ways to paraphrase sentences are generalized in order to cover a broader range of utterances, which leads to the development of what one might call paraphrase patterns (Mollá & Zaanen, 2005; Echihabi et al., 2006). The newer versions of systems that use these paraphrase patterns (like Echihabi et al. (2006)) allow for multiple variables, where the older ones usually allowed for just one (like Ravichandran and Hovy (2002) and D. Lin and Pantel (2001)). With these more general paraphrase patterns however, it does become more important to check them by hand in order to filter out misreformulations (Echihabi et al., 2006).

2.2.3 Problems

The main problem with paraphrases seems to be the chance of forming misreformulations. Part of this cannot be easily remedied, because it would involve extensive reasoning and incorporation of world knowledge. On the other hand, something simple like a cut-off on the edit distance between two paraphrases might prevent too obscure instances of the paraphrase patterns to be rejected. Also, some sort of weight could be attached to the edit distance between paraphrases, thus estimating the reliability of the paraphrase. There are some systems that use Minimum Edit Distance calculation algorithms for this, but another popular approach is that of using a probabilistic Noisy Channel system (Echihabi & Marcu, 2003; Shen et al., 2005). Those systems use statistics on previously used paraphrases and the number of times the paraphrases were helpful, to sort out the less likely candidates. Within Kaitito, not only can the misreformulations lead to sentences that mean something else than the user intended, but they can also lead to system errors when the misreformulations are ungrammatical. Using paraphrases on a higher level like the semantic level, is somewhat less prone to these kinds of errors. Since the sentence has already been processed and (too use the semantic level as an example) interpreted to some extent, using paraphrases that alter the meaning of a sentence becomes less likely. As for the ungrammatical sentences, the structure of a sentence that yet needs to be parsed by the ERG grammar is much more error prone than the loosely connected bag of relations in MRSs in semantic level.

2.3 Using external resources

In both surface-based pattern matching and paraphrasing, external resources (in particular dictionaries like WordNet and the World Wide Web) are used in order to overcome some of the eminent problems in those techniques. The following sections describe in more detail how and to what end these external resources are used. First WordNet and other dictionaries are looked into (Section 2.3.1) and after that the World Wide Web (Section 2.3.2).

2.3.1 WordNet and other dictionaries

As already mentioned, in both pattern matching and paraphrasing, synonyms are used in order to broaden the coverage of systems. Several researchers have shown that incorporating synonyms improves the performance of their system (Core & Moore, 2004; Jijkoun et al., 2004; C. Lin, 2002). Moreover, WordNet and collections like Wikipedia can provide direct answers to definition questions, or at least help determine the correct definition sentences in the target text (Echihabi et al., 2006; Fahmi & Bouma, 2006).

Core and Moore (2004) use the WordNet taxonomies to search for possible meanings of unknown words, which is an entirely different approach to using WordNet. Their algorithm uses the fact that which synonym is appropriate, depends on various features such as the part-of-speech a word is in case a word can be used as different types. If a word has successfully been tagged with a part-of-speech tag, this can be used to distinguish between the possible known synonyms of a word as found in the WordNet taxonomy that might be suitable and the ones that probably are not. This method has proved to sometimes be very effective in assisting the system identify the correct answers even without exactly understanding what the unknown word means.

A problem with using dictionaries like WordNet is that they sometimes lead to incorporation of wrong alternatives when context is considered, which only hinders precision (Galea, 2003). However using it for word sense disambiguation and the application as used by Core and Moore (2004) does seem to be very useful. The World Wide Web, as described in the next section, has some advantage over dictionaries, since a context usually is provided by the pages on the web (Duclaye & Collin, 2002).

2.3.2 World Wide Web

When using patterns or simple reformulations, the rarity of information restatement in a single document makes it harder to come up with a correct answer (Grois, 2005). Using a resource such as the web in order to find answers to factoid or definition questions about general world knowledge, provides a lot of redundancy and a lot of alternate formulations of the same answer (Brill, Dumais, & Banko, 2002; C. Lin, 2002; Plamondon & Lapalme, 2002). But using the web to increase recall directly is not the only way to apply it. It can also be used to extract terms that co-occur frequently with the target answer when trying to construct patterns (Greenwood & Saggion, 2004; Yang & Chua, 2002), or just to increase the certainty with which an answer is selected as the right one (Clarke, Cormack, Lynam, Li, & McLearn, 2001).

There are some downsides to using the web as a resource as well. For one thing, not all information that resides on the web can be taken for granted. Resources must be cross-validated in order to be able to refrain from retreating into more or less supervised parts of

the web like online encyclopedias. When using the web as a resource for reformulations, these also need to be checked as to avoid acquiring invalid patterns (Duclaye & Collin, 2002).

2.4 Other (deeper) methods

Shallow methods like surface-based pattern matching and the simpler of the paraphrasing methods might be computationally light, but they seem to be unable to handle more demanding types of utterances. Several authors question the practical usefulness for these shallow approaches in more complicated domains, which seem to require more sophisticated question decomposition, reasoning and answer synthesis (Nyberg & Frederking, 2003; Mollá & Zaanen, 2005). Also, just having shallow understanding of utterances is not good enough when trying to estimate the current level of a learner's knowledge, which is relevant for tutorial and CALL systems (Rosé et al., 2003).

The next section, Section 2.4.1, deals with using dependency relations and topics to come up with a suitable response. Section 2.4.2 describes using inferences and logic. In Section 2.4.3 the problems with using deeper methods are discussed.

2.4.1 Dependencies and topics

AutoTutor uses a technique could be called a surface-based approach, but the intelligence gathered is used to derive a rather deep level of understanding. It tries to grasp the intentions of the speaker by classifying utterances according to a taxonomy, using surfacy methods in the form of an adapted version of Latent Semantic Analysis (Graesser et al., 2000). This LSA approach is based upon Harris' Distributional Hypothesis, which states that words tend to be semantically similar if they appear in the same texts, which can be transformed to: texts are similar if they contain similar words (Angelova et al., 2004). Usually semantic, conceptual and pragmatic criteria are used instead of lexical and syntactical information, but AutoTutor tries to use more surface information (Olney et al., 2003). Using this idea, it can reveal links between words and phrases without requiring too sophisticated knowledge and with a large enough corpus to train on, it can be made reasonably robust to noisy input. On the other hand, this type of classification is quite coarse-grained which makes it blind to subtle details and it is somewhat limited in its classification of utterances as it cannot invent new classes on its own (Angelova et al., 2004). Accomplishing this would require a full artificial intelligence to be developed (Jurafsky & Martin, 2000). Still, similar approaches are used in QA systems to try and determine the type of expected answer in order to thus narrow down the possible answer set (Hovy et al., 2002; Echihabi et al., 2006).

Deeper methods are also used to try and reach the same sort of topic identification or coherency. Using dependency relations, several researchers try to find links between questions and answers (Bouma, 2006; Shen et al., 2005, a.o.). This allows them to find relations between entities that might be far apart on surface level and allows to find things like definitions, causes and symptoms (Bouma, 2006; Jijkoun et al., 2004).

2.4.2 Inferences and logic

The question-answer pair *Where is Devi from? Devi is from Australia* is rather straight forward, but the questions like *Is Devi Australian?* would pose a problem from Kaitito. The system does not know that being from Australia means that you are Australian. Let alone

that Kaitito could deduct that Devi would not be Australian if she came from Iran. Several attempts have been made to develop a system with (basic) inferential capability. For instance D. Lin and Pantel (2001) have their system learn inference rules from text. Based on the same principle AutoTutor uses (i.e. Harris' Distributional Hypothesis), their system finds links in dependency trees that tend to link the same set of words, which should be an indication of similar meaning. This system discovers many inference rules easily missed by humans. For instance for the sentence *Douglas Adams is the author of HHGTTG* the researchers did not come up with the pattern *Douglas Adams notes in HHGTTG that...*, but the system did. Obviously, *Douglas Adams notes in HHGTTG that...* does indeed imply that Douglas Adams has written (part of) HHGTTG. The system used in AutoTutor seems to do reasonably well for simple inferences, but as Olney et al. (2003) note, there are some important issues that need to be addressed in future work like recognizing polarity in inference relationships.

Moldovan et al. (2002) and (Rinaldi et al., 2003) actually try to prove that something is an answer to a question. Unfortunately, for now this approach only works for relatively easy question-answer pairs.

2.4.3 Problems

A limitation of these types of approaches is that, for now, they seem to be unable to cover more complicated question-answer pairs and questions with indirect answers. Theoretically, that is a typical limitation for shallower approaches, but the amount and level of reasoning required for answering these questions is beyond the reach of current deeper, inference-based systems. Deeper methods would not help with questions like *What kind of music do you like?* (which can be answered with music genres, but also with names of bands and musicians), *What do/does X look like?* (which would entail a description of body parts, so that the system would need to know which are the appropriate body parts to describe and what kind of properties they can have) and the somewhat more open-ended *What is/am/are X like?* (which can be answered by a description of personality and common traits or it could be something like *You are like a baboon*). Also, deep analysis is costly in terms of computational complexity, which makes it less suitable for real-time systems or scaling up to cover large domains (Leveling, 2006). Furthermore, because these types of approaches require more in depth parsing, parse errors are a big issue (Jijkoun et al., 2004).

The integration of deep analysis and shallow, surface-based methods is a solution that at least remedies some of the problems in part. A number of systems use deep analysis, but somehow use shallow methods as a fallback. This can be done either by relying on surface methods when deeper analysis fails, or by involving all different ways of finding an answer and awarding confidence scores (Boytcheva, Vitanova, Strupchanska, Yankova, & Angelova, 2004; Gockley et al., 2004; Leveling, 2006; Purver et al., 2006). Noteworthy is the fact that tests with the system by Gockley et al. (2004) show that in their system that uses the fallback method, actually almost all of their input ended up being handled by the surface-based method.

2.5 Conclusions

All of the approaches considered in this chapter have their advantages and disadvantages. There are some issues that are most important for making a decision about the right approach to improve the language understanding capabilities of systems like Kaitito:

- Certain characteristics and generalizations of a sentence simply cannot be grasped using only surface-based methods, or require an enormous amount of surface-based rules which perhaps might be replaced by a single deeper rule, which overall is 'cheaper'. Consider for instance a bunch of sentences that in fact all have the same meaning and require the same answer that is an indirect answer to the question. On surface level for each of these sentences a rule could be derived to map the question onto the answer, but (under ideal circumstances) semantically all questions would have the same semantic representation and only one rule needs to be derived to be able to map the questions onto the indirect answer. The next chapter will look into this more closely.

- The state-of-the-art deep methods are not advanced enough yet to handle the level of inferences needed for a dialogue system.

- When dealing with questions that are more complicated, devising patterns to match the questions to possible answers is expected to be quite hard. Automatic learning could be considered, but this also opens up the door for faulty rules to be derived. This is especially so because there is no elaborate corpus and even though the range of utterances a new language learner will use is quite limited, even within this set there are subtle differences the automatic learning might not pick up on (consider the difference between *How do you do?* and *How did you do?* or *How did you do that?*).

When only considering the Kaitito system, some of the advantages and disadvantages become less important:

- Surface-based approaches using shallow parsing have less trouble with processing both syntactically incorrect and incomplete answers as deeper methods do. In this respect, shallow parsing has no advantage over deeper parsing when using it with the Kaitito system, because Kaitito has a module that handles errors in the user input (see Section 1.1.2).

- Shallow parsing is computationally lighter than deeper parsing. Within the Kaitito system, deeper parsing up to semantic level will be executed regardless at what level the rules are defined, so the rules might just as well be defined at the semantic level.

The Kaitito system needs input that can be mapped quite literally onto the knowledge in the knowledge base (see Section 1.1.2), with the knowledge put in there by domain experts. Since the idea is that students will practice natural dialogues with the system, there is little need for a variety of answers to the same question (other than the slight alterations to amongst others politeness, which the dialogue system will be capable of making). Because the input needs to match so perfectly to the limited available knowledge, it makes sense to rewrite the non-matching input to input that will match. As for the level at which this should be done, the semantic level seems to be most appropriate. This is illustrated in Figure 2.2, using adapted versions of the Vauqois triangle (Figure 1.4) for both paraphrasing the input and finding a suitable response in the knowledge base. At the semantic level, Kaitito uses MRS representations of both input and possible answers to determine what its output is going to be. MRSs are thus always generated for each input sentence (as long as it is a sentence that can be parsed), and using these would not increase the load of the system to much compared to using surface level patterns. Choosing the semantic level, means that transformations are on the level of MRSs. Transformations on the MRS level are executed using transfer rules,

which will be described in the next chapter. An additional advantage of using these transfer rules on MRSs compared to for instance regular expressions on surface level is, that deriving transfer rules will be easier to automate than deriving complex regular expressions.

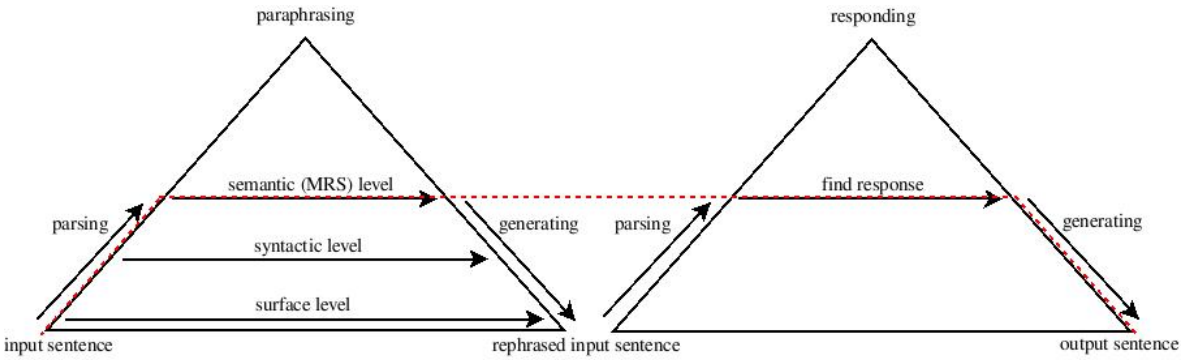

Figure 2.2: Paraphrasing MRSs and finding a response should follow the red dotted line.

3

Designs to improve Kaitito's language understanding

Understanding natural language input is very complex, as stated previously. Without very sophisticated reasoning capabilities and other deep parsing techniques, natural language input can never be fully comprehended, if even with them. For instance, humans sometimes misinterpret each other because the intention behind a message can be different from the way the receiver interprets it.

Since Kaitito is a system intended for language learners who are at levels that are not too advanced, this issue is not very important here. What is important however, is which parts of the language Kaitito should be able to handle. The next section (Section 3.1) therefore deals with gathering information on English language learning curricula. As shown in the previous chapter, a solution can be devised anywhere on the full range of levels in language understanding, but paraphrasing at the level of semantic representations seems to be the most appropriate approach for Kaitito. Section 3.2 will describe the design for using this approach within Kaitito. Concluding this chapter, Section 3.3 will briefly deal with issues like handling large amount of paraphrasing rules and keeping consistency among them.

3.1 English language learning curricula

Even though English is spoken all over the world, structured information on English (as a second language) language teaching curricula is quite scarce. Both the Malaysian and the Canadian ministries of education have some information on their English as a second language curricula on-line (see http://myschoolnet.ppk.kpm.my/sp_hsp/b_ing.htm and http://www.education.gov.ab.ca/k_12/curriculum/bySubject/ESL/ respectively) and there are a number of free on-line resources[1] for learning English (as a second language) that seem to be used quite frequently, based upon the listings of reactions to the sites and their fora.

Gathering and organizing all example sentences from the different available sources, led to a small corpus of sentences. The corpus contains little under 350 lines, 70% of which are questions, usually with their corresponding answer on the same line. There was a big overlap in the sentences used within the themes. This is a good indication that the scope of sentences that language learners master at these beginner levels, will be limited to those sentences. In

[1] http://www.usingenglish.com/
http://www.learnenglish.de/
http://www.learn-english-online.org/
http://esl.about.com/od/beginningvocabulary/a/basic_conv.htm

the interest of keeping the Te Kaitito system as light as possible, the calculation whether to include rare cases is relatively more beneficial than leaving them out, is an important one.

The following sections (Section 3.1.1 and 3.1.2) take a closer look at the question-answer pairs that can be found in the English language learning curricula. Section 3.1.3 fences off the theme used for the rest of this project.

3.1.1 Questions with (partially) matching answers

Within the gathered corpus, there are a lot of questions that can be answered literally, as long as the appropriate information was authored in a sensible way[2]. For instance, the question *How do you take your coffee?* can be literally answered when the system knows *I take my coffee with milk and sugar*, which could result in an answer like *With milk and sugar please.* Note that this answer is shortened (*with milk and sugar*) instead of the full version *I take it with milk and sugar*) and that some level of politeness is added through *please.* These are examples of adaptations that can be added after internal generation of the answer to make the dialogue system behave more naturally, but the knowledge in the system that is behind this answer is the thing that is important for matching questions to answers or rather matching them to the knowledge that makes up the answer.

Also literally answerable is *How are you?* by *I am fine*, but questions like *How are you doing?*, *How do you do?* and *How's life?* are not, even though these should lead to the same answer. When further inquiring about a person, questions like *What do you do?* (what is you occupation/profession?) and *What are you doing?* can be encountered. These two questions look a lot like two of the *how are you* variants when you swap the word *how* with *what.* In fact their respective answers are probable to look pretty similar as well (compare *I am a teacher* and *I am reading* to *I am fine*). Still, it is pretty awkward for *What do you do?* and *What are you doing* to both be answered by *I am a teacher.*

As previously mentioned, the Kaitito system can only answer the questions that have a completely matching answer. Answering questions with only partially matching answers is exactly what needs to be addressed by paraphrasing on semantic level, which will be dealt with in Section 3.2.

3.1.2 Questions with answers of a non-matching type

When literally answering a question like *Could you tell me the time please?* with *Yes*, people may consider this answer humorous or get annoyed because it really did not tell them what they wanted to know. The more acceptable or usual way of answering this question would be something like *Yes, it's two o'clock.* This answer strictly consists of two answers; the first corresponds to the literal meaning of the question and matches the type of the question (a yes/no question in this case), the second corresponds to the intention behind the question.

This kind of yes/no question regularly occurs in constructions that have a certain level of politeness due to using the modal verbs *can* and *will.* These kind of constructions seem to be very frequent in the restaurant theme (*Would you like something to drink?* and *Can I bring you anything else?*). In the same theme however, also more precise questions like *Would you like a cup of coffee?*, which actually are nothing but yes/no questions, are quite frequent. But

[2]Authoring is considered to be done in a sensible way if the reactions authored correspond to the most probable ones.

even without the modal verbs *can* and *will* similar sentences can be produced (*Do you have the time?* and *Have you got the time?*).

Currently, the Kaitito system cannot handle these kind of answers, because they are a combination of two answer sentences and Kaitito can only handle single sentences in both input and output. Some time in the future this should be solved by splitting multiple sentence input into a stack of single sentences, without losing the coherence between the sentences that exists from their co-occurrence, which is a task that comes with many complications (such as which sentence should be reacted to or, on the other end, when is a multiple answer sentence required and when is it not).

3.1.3 Focus on introductions

One thing that becomes obvious when looking at the curricula as a whole, is that the learning tends to be centered around certain themes. The most often occurring themes include *getting to know each other*, *eating in a restaurant*, *staying in a hotel* and *asking and telling the time*. Within these themes, various elements of the language such as vocabulary and grammar are dealt with. This kind of theme-based instruction is in fact handy for a system like Kaitito, because it limits the domain for each conversation somewhat. Also, constructions that are typical for certain themes can be integrated per theme, thus not increasing the load of the system when another theme (in which these constructions are not, or very scarcely, used) is enacted. Examples of this are things like *Could I get you anything else?* asked by a waiter to a customer, which has some politeness-construction (with *could I*), is very open ended (*anything else*) and obviously will not have an answer that will literally match the question in terms of words occurring in the question that will also occur in the candidate answer sentence. Instead, the answer will probably be something like *Another glass of orange juice please.*

As mentioned in the previous section, Kaitito cannot handle questions and answers that require multiple sentences. Another limitation of the system as currently implemented, is that it lacks any notion of time. The system is unable to reason about actions as being executed sequentially, which limits its ability to cope with the themes in which order is important, like the *eating in a restaurant* and *staying in a hotel* themes. Also *telling the time* is something that should be hard-coded into the system, which is undesirable. Most of the issues from the *getting to know each other* theme however, can be addressed within the current limits of the system. Since this theme still incorporates a rather rich range of different problems, this project will from now on be focused on this theme.

3.2 Derivation of transfer rules

In the previous chapter it was concluded that the semantic level (the level of Minimal Recursion Semantics in Kaitito, see Section 1.1.1) seems to be the right level to paraphrase questions so that they match better with available answers. The way paraphrases on MRS level can be specified using so-called 'transfer rules', will be explained in this section. Before the composition of the actual transfer rules can be explained however, some background knowledge concerning MRSs is required as provided in Section 3.2.1. Section 3.2.2 deals with the elements and syntax of transfer rules and Section 3.2.3 describes the steps of the manual derivation algorithm, whereas Section 3.2.4 deals with issues regarding automating the derivation process. Finally, Section 3.2.5 addresses coverage and precision of transfer rules.

3.2.1 Short introduction to MRSs

In order to understand how transfer rules should be defined, it is necessary to understand the basics of MRSs. As mentioned before, MRSs are a way to describe semantic structures. They have a couple of interesting properties such as the fact that they have a flat structure and can be used to represent underspecified semantic structures because every element is given a handle. Here, the underspecification is related to the scope of the quantifiers. An example of this is the sentence *Every living person has some parents*. This sentence can be read as it was probably intended in this case: "every living person has parents, but not necessarily the same ones", but it can also be read as: "every living person has exactly the same parents". The elements (which are more commonly know as 'elementary predications', 'EPs' for short, and are also called 'relations', which is a bit confusing considering the specification of EPs as written below, see Copestake et al. (2005) and Slabbers (2005)) are at the level of individual lexemes, entries in the lexicon. Each of them consists of:

- A handle, the label of the EP which is used to link it to other parts of the MRS
- A relation, with a name that is called the 'predicate'
- A list of zero or more arguments of the relation with their values, which either have a static value (the name of a named entity) or consist of a 'type' and an 'ID' and are called 'referents', with the following possible types:

  'e-vars' refer to events

  'h-vars' refer to handles that are used to present the links between EPs

  'i-vars' refer to an event or index

  'u-vars' refer to semantic arguments

  'x-vars' refer to entities (persons, items)

  thus referents look like `h1`, which is a referent of type 'handle' (h) with ID 1.

Furthermore, all handles and values of referents contain within them IDs that link them together in the right fashion. In the most simple MRS format, EPs are thus written:

$$\texttt{handle:relation(arg}_1\texttt{,...,arg}_n\texttt{, scarg}_1\texttt{,...,scarg}_n\texttt{)}$$

The things that the referents refer to can contain a bit of extra information, most common are temporal information for events and person information (both number and singular/plural) for entities[3]. An MRS structure consists not only of a bag of these EPs, but it also has a top-handle to which (through links with other EPs) ultimately each EP is linked to and a bag of handle constraints that restrict the MRS to one possible reading when the semantics are underspecified. So, an MRS structure is a tuple $\langle$ `T, L, C` $\rangle$, where `T` is the top handle, `L` is the bag of EPs and `C` is the bag of handle constraints. To use the sentence *Every living person has some parents* again to illustrate, the most basic MRS representation (called 'indexed MRS output' in Kaitito) would look something like this:

```
⟨{h1:every(x2,h3,hA),h3:living(x2), h3:person(x2), h7:parents(x8),
     h5:some(x8,h7,hB), h4:have(x2,x8)}, {(hA:h5), (hB:h4)}⟩
```

[3]For a detailed description, please refer to Copestake et al. (2005).

Here, `h1` is the top handle, the bag of EPs is the first part between curly brackets and the second part between curly brackets is the bag of handle constraints (HCONs). This bag of HCONs basically limits the MRS representation to a single reading of an MRS where there are multiple options, by linking under-scoped referents (`hA` and `hB`).

The thing is, that the exact appearance of MRSs is highly influenced by the grammar(s) used to build them and the level of specification wanted. From now on, all examples given will be in a format that can be used with the Kaitito system (for a more elaborate explanation of MRSs in general, please refer to Copestake et al. (2005)). Within Kaitito, there are three levels of specification for MRSs. The most basic of these is the 'indexed MRS output', which was shown earlier and takes only a few lines. A bit more elaborate is the 'simple MRS output' which also shows the arguments that referents belong to. Finally, there is the 'full MRS output' that also shows all extra information of referents (things like person and tense) and easily is over five pages long. Appendix A contains examples of all three of the aforementioned levels of MRS output.

3.2.2 Defining transfer rules

Transforming one MRS into another is also called '(MRS) munging' or 'transfer', hence the names 'munge rules' and 'transfer rules'. Transfer rules consist of a number of different parts (Nygaard, Lønning, Nordgård, & Oepen, 2006; Oepen et al., 2004):

- Name of the transfer rule
  - ID of the transfer rule (should be unique)
  - Parent of the transfer rule (is optional, specifies another rule from which the current rule should inherit all properties)
- Input slot specifying what input needs to be munged
- Output slot specifying to what the input must be transformed
- Context slots specifying when the munging should or should not be applied

Apart from the transfer rule name (ID and parent), things that are put into slots mentioned above are (part of) an MRS. The specification of input and output is mandatory, but the specification of context is optional and can be left out. At the moment, there are two slots for specifying context:

- the condition slot for things that have to be present in the input for the transfer rule to be applied, but that need not to be changed (otherwise they would be put into the input slot of the transfer rule)
- the filter slot for things that should not be present in the input for the transfer rule to be applied

There is reason to believe that the functionality of the context slots will increase quite a bit shortly, allowing for regular expressions and more partial MRSs to be used[4].

An example of a very simple transfer rule is depicted in Figure 3.1, illustrating the rule that makes sure that water is changed into wine, without any restriction to the circumstances or the presence of something or someone.

[4]This information was published on an intranet part of the website http://wiki.delph-in.net, which was open to everyone for a short period of time.

```
water_into_wine := monotonic_mtr &
[ INPUT [ HOOK.LTOP #h1 & handle,
          HOOK.INDEX #e2 & event,
          RELS <! [ LBL #h3 & handle,
                    PRED ``_water_n_1_rel'',
                    ARG0 #x5 & ref-ind ] !> ],
  OUTPUT [ HOOK.LTOP #h1,
           HOOK.INDEX #e2,
           RELS <! [ LBL #h3,
                     PRED ``_wine_n_1_rel'',
                     ARG0 #x5 ] !> ] ].
```

Figure 3.1: Basic noun transfer rule for changing water into wine.

Here, `water_into_wine` is the ID of the transfer rule, `monotonic_mtr` is the parent of the transfer rule. Both input and output are still MRSs. They have something at the top of the MRS (pointed to by `#h1`) with type 'handle' (specified by `& handle`), an index with type 'event' and finally one EP with arguments (`_water_n_1_rel` and `_wine_n_1_rel` respectively). Note that the representation of the MRSs in input and output strongly resemble a simple MRS output, with only strictly necessary extra specifications added. The condition and filter (context slots) have no content in this example and are thus left out.

In general within the transfer rules, input, output and the optional condition and filter are all still (parts of) MRSs. When writing a transfer rule, these (parts of) MRSs are specified using a syntax just slightly different from the syntax used for the simple MRS output. Some extra specifications are included, explicitly stating the type of the referents (`handle`, `event` and `ref-ind` for instance) and when defining an entirely new referent, its full specification (including extra information like gender and person) needs to be added. In a transfer rule, the (former) referents are mere pointers to a certain value and could be given any arbitrary string value, as long as they match up when needed. The `#` are added to the original referents signal this distinction. Note that using the simple MRS output as basic format for the transfer rules, allows for easy underspecification of rules so that they can be used to generalize over things like gender. After all, this extra information is only available in the full MRS output, except when explicitly included in the transfer rule. The context, split in condition and filter, is also directly derived from MRSs, restricting the application of a transfer rule to occurrences where the input matches and for instance the whole sentence is in present tense or contains a specific word like *farmer*. In Section 3.2.5 the importance of specifying rules at the right the level of precision of rules will be dealt with in more detail.

3.2.3 Algorithm for deriving transfer rules

To munge one sentence into the other (or rather transform one MRS representation into the other), rules have to be derived based upon the two original MRSs. The basic set of operations that have to be performed to derive a transfer rule by hand can be described as follows:

1. Put the input and output sentence into the input and output of the transfer rule respectively using the correct syntax and give the rule an ID

2. Remove all EPs and HCONs that literally overlap between input and output (except for EPs needed to keep remaining HCONs lined-up)

3. Give handles and referents new IDs so that only IDs that should match do so

4. Add context to the condition/filter slot of the transfer rule, if applicable

5. Add type information to (at least the first occurrence of) all referents in the transfer rule

6. Add the full specification of the extra information belonging to referents if necessary

ad. 2 Since the transfer rule concerns the things that need to change in one MRS to make it into another MRS, the things that do not change between the MRSs are irrelevant. The could simply be left alone, but that clogs the rules and also makes them slower to use since more EPs and HCONs need to be checked. However, remaining, non-overlapping HCONs might contain a referent that only occurs in an overlapping EP. In that case, the EP in question should be put in the condition slot of the transfer rule.

ad. 3 Assigning correct IDs to all handles and referents, really is quite complicated. It has to be made sure that the IDs are correctly co-indexed within the input part of the transfer rule, within the output part of the transfer rule as well as between input and output. This entails mapping the input and output part of the transfer rule onto each other, to see which referents should have the same ID. A simplified, but elaborate example can be found directly below. Note that this example does not in any way address the issues of underspecification and extra information that needs to be added, but is merely aimed at illustrating the links between two MRSs and within MRSs themselves.

In many ways the process of mapping two MRSs and giving the referents new IDs can be compared to aligning two town maps and accompanying building indexes with the same grid, but different labels for the rows and columns. In Figure 3.2 two maps are shown that clearly describe the same area[5]. Some of the buildings present on the first map are not on the second map and vise versa. Looking at the buildings in turn (this is similar to the matching of MRSs, using indexes instead of IDs) gives:

AI has index `R2` on the first map, `D2` on the second map (linking `R2` – `D2`)

AW does not exist on the first map, has index `E4` on the second map

CS does not exist on the first map, has index `D4` on the second map

CT has index `R3` on the first map, `D4` on the second map (linking (`R3` – `D4`)

G has index `R2` on the first map, does not exist on the second map

[5]These maps are loosely based on the Dunedin Campus Map (that can be found on http://www.otago.ac.nz/about/pdfs/dunedin_campus.pdf), however, the maps are far from realistic.

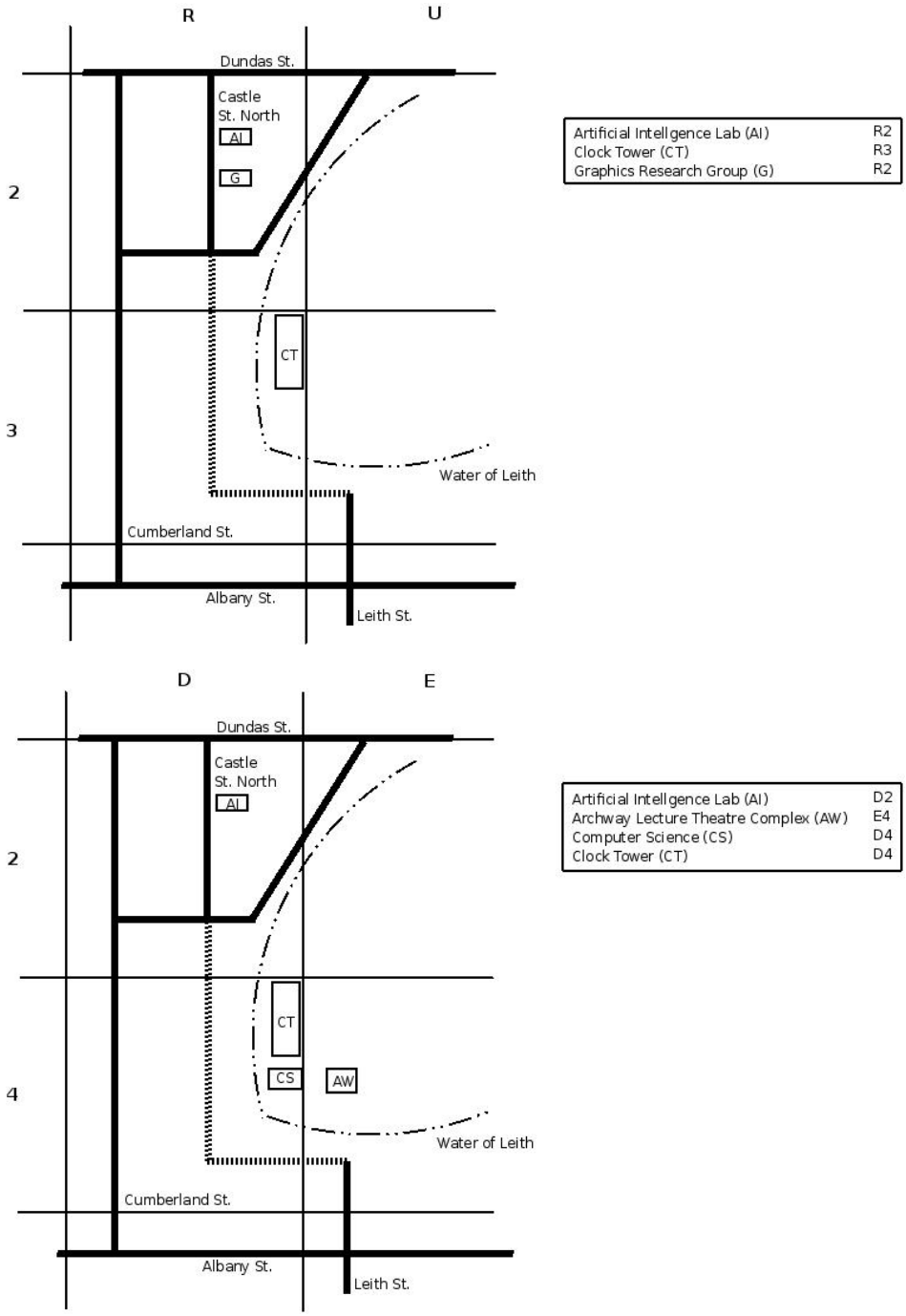

Figure 3.2: Illustrating problem with IDs (1/2), maps to be changed.

Although not totally arbitrary, the current labels for the maps are not very easy to use when the map needs to be expanded. Suppose it can safely be assumed that expansion will only be possible to the right and downwards. Now it makes sense to start numbering from the left upper corner outwards to the right and down. Using the alphabet from `A` to `Z` and numbers

starting at `1` and increasing with one at a time, an easily expandable grid can be created. Again looking at the buildings in turn, but now giving them new indexes:

AI this is the first building, so it gets new index `A1` and the index `R2` on the first map and `D2` on the second map are replaced.

AW has index `E4` on the second map, which is not bound to anything on the first map nor on the new map yet, neither vertically nor horizontally, so it gets index `B2`

CS has index `D4` on the second map, so is bound to the new map vertically through the **AI** building and horizontally through the **AW** building, giving new index `A2`

CT has index `D4` on the second map just like the **CS** building, so gets the same index as **CS** (`A2`)

G has index `R2` on the first map just like the **AI** building, so gets new index `A1`, that **AI** already has

The new maps with their indexes now look like Figure 3.3. Note that for the IDs in MRSs, there is no horizontal and vertical: it is just a singular index instead of a compound like with the maps. Other than that, the ideas behind matching and giving the new IDs (or indexes for the maps) is pretty much the same.

3.2.4 Semi-automatic derivation of transfer rules

Even though deriving transfer rules can never be fully automated, since things like selecting the correct parse of a sentence always have to be done by hand, but it can partly be automated. There are two reasons for wanting to automate the derivation:

- It will probably be more time efficient in the end than deriving all rules by hand
- It can allow for non-linguists to add and edit the rule database, especially when the automatic derivation makes use of the authoring module

The first reason speaks for itself, because deriving a single rule by hand can take up to a few hours, while automatically deriving a rule takes a few minutes at the most. The second reason is more important when thinking about the rationale behind managing Kaitito's knowledge base: people with domain expertise (English teachers) are going to be the ones editing the knowledge base. So, it would make sense that these experts are the best people to make a list of alternative questions that should lead to the same answer. With the automatic derivation, these experts just need to give the system an example input sentence (like *How do you do?*) and the sentence which it should be paraphrased into, the output sentence (like *How are you?*). The authoring module can then be used to disambiguate different semantic parses of the input and output sentences if necessary (see Section 1.1.2). Other, new ways to further improve the usability of the system for non-linguists are being developed (see for instance Knott and Vlugter (2007)) and might improve the usability of the automatic derivation of transfer rules even more.

So far, no known attempts have been undertaken to automate derivation of transfer rules, apart from where researchers have used a template-based approach to make simple rules for synonyms and hypernyms. Of course, the algorithm presented here is very capable to derive

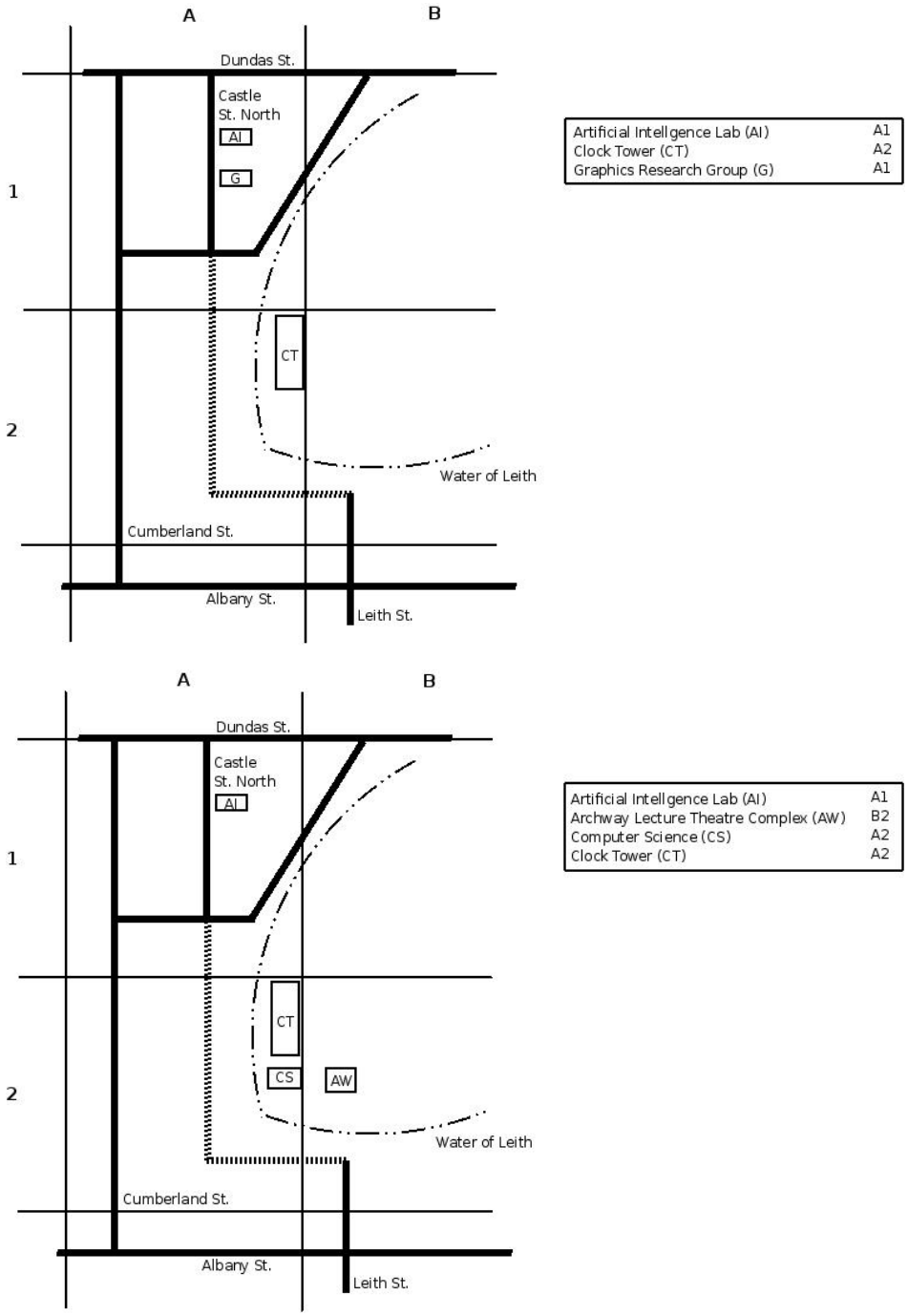

Figure 3.3: Illustrating problem with IDs (1/2), changed maps.

these simple rules as well, especially when a connection to a WordNet interface is made to automatically find the synonyms and hypernyms. A problem with this is though, that a single word can have different meanings in different contexts. Therefore, you cannot simply use all synonyms and hypernyms (Flickinger, Lønning, Dyvik, Oepen, & Bond, 2005; Nygaard et al.,

2006). When using the algorithm presented here, it is possible to use the extra information present in the form of the rest of the sentence, in the context part of a transfer rule to see whether a certain context is present. That way, the already existing `add-synonym` function in the Kaitito system could be improved.

Since the automated derivation process handles MRSs as represented internally in the system, the basis of the transfer rules is no longer formed by the simple MRS output, but by the full MRS output (refer to Appendix A for examples that clearly demonstrate the difference in level of specification). This means that instead of adding extra information to make the rule complete, information needs to be left out explicitly to allow for generalizing over things like person. Thus, the algorithm from Section 3.2.3 needs a slight adaptation to the second step of the original algorithm, so that it incorporates step 5 and 6. The new version is as follows:

1. Put the input and output sentence into the input and output of the transfer rule respectively (using the correct syntax if the rule is to be printed and not just used internally) and give the rule an ID

2. Strip the transfer rule of all things that unnecessarily restrict the transfer rule:
   - Remove all EPs and HCONs that literally overlap between input and output (except for EPs needed to keep remaining HCONs lined-up)
   - Remove all non-essential extra information from the variable specifications

3. Give handles and referents new IDs so that only IDs that should match do so

4. Add context to the condition/filter slot of the transfer rule, if applicable

For an example of the execution of this algorithm, see Appendix C. There, a transfer rule is derived for munging *How do you do?* to *How are you?*, with the second person as context condition.

3.2.5 Coverage and precision

Simply stripping the rule based on the heuristics of the algorithm(see the previous two sections) can make rules too general. However, not stripping the rules makes them far too restricting, as the rule derived to transfer *How do you do?* into *How are you?*, of which a schematic version is depicted in Figure 3.4.

Preferably, this rule would also be applicable to *How does he do?* and *How do they do?*, etc. Therefore, the rule form Figure 3.4 should be underspecified to match all persons, resulting in the rule depicted in Figure 3.5.

$$
\begin{bmatrix}
\text{INPUT} & \begin{bmatrix}
\text{HOOK.LTOP } \#h1 \text{ \& handle} \\
\text{HOOK.INDEX } \#e2 \text{ \& event} \\
\text{RELS} & \left(\begin{array}{l}
\begin{bmatrix}
\text{LBL } \#h6 \text{ \& handle} \\
\text{PRED unspec\_manner\_rel} \\
\text{ARG0 } \#e2 \text{ \& event} \\
\text{ARG1 } \#u8 \text{ \& semarg} \\
\text{ARG2 } \#x7 \text{ \& ref-ind}
\end{bmatrix} \\
\begin{bmatrix}
\text{LBL } \#h13 \text{ \& handle} \\
\text{PRED pron\_rel} \\
\text{ARG0 } \#x14 \text{ \& ref-ind} \begin{bmatrix} \text{PNG png} \begin{bmatrix} \text{PN 2per} \\ \text{GEN real\_gender} \end{bmatrix} \\ \text{PRONTYPE std\_pron} \end{bmatrix}
\end{bmatrix} \\
\begin{bmatrix}
\text{LBL } \#h15 \text{ \& handle} \\
\text{PRED pronoun\_q\_rel} \\
\text{ARG0 } \#x14 \\
\text{RSTR } \#h16 \text{ \& handle} \\
\text{BODY } \#h17 \text{ \& handle}
\end{bmatrix} \\
\begin{bmatrix}
\text{LBL } \#h6 \text{ \& handle} \\
\text{PRED ``\_do\_v\_1\_rel''} \\
\text{ARG0 } \#e2 \\
\text{ARG1 } \#x14
\end{bmatrix}
\end{array}\right) \\
\text{HCONS} & \left(\text{qeq} \begin{bmatrix} \text{HARG } \#h16 \\ \text{LARG } \#h13 \end{bmatrix}\right)
\end{bmatrix} \\
\text{OUTPUT} & \begin{bmatrix}
\text{HOOK.LTOP } \#h1 \\
\text{HOOK.INDEX } \#e2 \\
\text{RELS} & \left(\begin{array}{l}
\begin{bmatrix}
\text{LBL } \#h6 \\
\text{PRED unspec\_manner\_rel} \\
\text{ARG0 } \#e2 \\
\text{ARG1 } \#x8 \text{ \& ref-ind} \begin{bmatrix} \text{PNG png} \begin{bmatrix} \text{PN 2per} \\ \text{GEN real\_gender} \end{bmatrix} \\ \text{PRONTYPE std\_pron} \end{bmatrix} \\
\text{ARG2 } \#x7
\end{bmatrix} \\
\begin{bmatrix}
\text{LBL } \#h13 \\
\text{PRED pron\_rel} \\
\text{ARG0 } \#x8
\end{bmatrix} \\
\begin{bmatrix}
\text{LBL } \#h14 \\
\text{PRED pronoun\_q\_rel} \\
\text{ARG0 } \#x8 \\
\text{RSTR } \#h15 \\
\text{BODY } \#h16
\end{bmatrix}
\end{array}\right) \\
\text{HCONS} & \left(\text{qeq} \begin{bmatrix} \text{HARG } \#h15 \\ \text{LARG } \#h13 \end{bmatrix}\right)
\end{bmatrix}
\end{bmatrix}
$$

Figure 3.4: Specific transfer rule (how do you do → how are you).

$$
\begin{bmatrix}
\text{INPUT} & \begin{bmatrix}
\text{HOOK.LTOP \#h1 \& handle} \\
\text{HOOK.INDEX \#e2 \& event} \\
\text{RELS} & \left(\begin{array}{l}
\begin{bmatrix}\text{LBL \#h6 \& handle} \\ \text{PRED unspec\_manner\_rel} \\ \text{ARG0 \#e2 \& event} \\ \text{ARG1 \#u8 \& semarg} \\ \text{ARG2 \#x7 \& ref-ind}\end{bmatrix} \\
\begin{bmatrix}\text{LBL \#h13 \& handle} \\ \text{PRED pron\_rel} \\ \text{ARG0 \#x14 \& ref-ind}\end{bmatrix} \\
\begin{bmatrix}\text{LBL \#h15 \& handle} \\ \text{PRED pronoun\_q\_rel} \\ \text{ARG0 \#x14} \\ \text{RSTR \#h16 \& handle} \\ \text{BODY \#h17 \& handle}\end{bmatrix} \\
\begin{bmatrix}\text{LBL \#h6 \& handle} \\ \text{PRED “\_do\_v\_1\_rel”} \\ \text{ARG0 \#e2} \\ \text{ARG1 \#x14}\end{bmatrix}
\end{array}\right) \\
\text{HCONS} & \left(qeq \begin{bmatrix}\text{HARG \#h16} \\ \text{LARG \#h13}\end{bmatrix}\right)
\end{bmatrix} \\
\text{OUTPUT} & \begin{bmatrix}
\text{HOOK.LTOP \#h1} \\
\text{HOOK.INDEX \#e2} \\
\text{RELS} & \left(\begin{array}{l}
\begin{bmatrix}\text{LBL \#h6} \\ \text{PRED unspec\_manner\_rel} \\ \text{ARG0 \#e2} \\ \text{ARG1 \#x14} \\ \text{ARG2 \#x7}\end{bmatrix} \\
\begin{bmatrix}\text{LBL \#h13} \\ \text{PRED pron\_rel} \\ \text{ARG0 \#x14}\end{bmatrix} \\
\begin{bmatrix}\text{LBL \#h14} \\ \text{PRED pronoun\_q\_rel} \\ \text{ARG0 \#x14} \\ \text{RSTR \#h16} \\ \text{BODY \#h17}\end{bmatrix}
\end{array}\right) \\
\text{HCONS} & \left(qeq \begin{bmatrix}\text{HARG \#h16} \\ \text{LARG \#h13}\end{bmatrix}\right)
\end{bmatrix}
\end{bmatrix}
$$

Figure 3.5: Semi-specific transfer rule (how [to do] *pronoun* do → how [to be] *pronoun*).

Unfortunately, this rule still does not cover questions like *How does Peter do?*, because the rule only fires when a pronoun is present. Underspecifying the rule even more, it can be reduced to require just a form of the request 'how' (denoted in the semantic representation by the EP `unspec_manner_rel`) and the EP representing the (non-auxiliary) verb 'do' (the `_do_v_1_rel`) with the appropriate linking arguments. Thus, the rule in Figure 3.6 can be defined. At first sight, it looks like the rule in Figure 3.6 is going to be far too general. But in

$$\begin{bmatrix} \text{INPUT} & \begin{bmatrix} \text{HOOK.LTOP \#h1 \& handle} \\ \text{HOOK.INDEX \#e2 \& event} \\ \text{RELS} & \left(\begin{array}{l} \begin{bmatrix} \text{LBL \#h6 \& handle} \\ \text{PRED unspec\_manner\_rel} \\ \text{ARG0 \#e2 \& event} \\ \text{ARG1 \#u8 \& semarg} \\ \text{ARG2 \#x7 \& ref-ind} \end{bmatrix} \\ \begin{bmatrix} \text{LBL \#h6 \& handle} \\ \text{PRED "\_do\_v\_1\_rel"} \\ \text{ARG0 \#e2} \\ \text{ARG1 \#x14 \& ref-ind} \end{bmatrix} \end{array} \right) \end{bmatrix} \\ \text{OUTPUT} & \begin{bmatrix} \text{HOOK.LTOP \#h1} \\ \text{HOOK.INDEX \#e2} \\ \text{RELS} & \left(\begin{bmatrix} \text{LBL \#h6} \\ \text{PRED unspec\_manner\_rel} \\ \text{ARG0 \#e2} \\ \text{ARG1 \#x14} \\ \text{ARG2 \#x7} \end{bmatrix} \right) \end{bmatrix} \end{bmatrix}$$

Figure 3.6: General transfer rule (how [to do] *person_des* do → how [to be] *person_des*).

fact, thanks to certain links between the EPs that are present due to the MRS architecture, sentences like *How do you do that?* are not transformed into *How is you that?* or something similar but are left alone as they should be. Sometimes sentences with a prepositional phrase should be transformed, so this might pose a problem in those cases, solvable by deriving separate transfer rules.

Now, consider the sentences *The farmer sees rain* and *The farmer is happy.* Needless to say, not *everybody* is happy when it rains so the rule derived for this pair of sentences should be made rather specific to the farmer. Furthermore, consider the sentence *How did you do?*, which would be transformed to *How were you?* then applying the rule from Figure 3.6. *How did you do?* has a meaning that is more explicit in *How did you do on your exam?*. Clearly, here the munging should not take place. So the variable that can contain extra information about tense, should be augmented to contain that extra information. In this case, that would be the event referent who's first occurrence is in the index, so the new rule should look like Figure 3.7.

$$
\begin{bmatrix}
\text{INPUT} & \begin{bmatrix}
\text{HOOK.LTOP \#h1 \& handle} \\
\text{HOOK.INDEX \#e2 \& event} \begin{bmatrix} \text{TENSE PRES} \\ \text{MOOD INDICATIVE} \\ \text{PROG -} \\ \text{PERF -} \end{bmatrix} \\
\text{RELS} \left(\begin{bmatrix} \text{LBL \#h6 \& handle} \\ \text{PRED unspec\_manner\_rel} \\ \text{ARG0 \#e2} \\ \text{ARG1 \#u8 \& semarg} \\ \text{ARG2 \#x7 \& ref-ind} \end{bmatrix} \begin{bmatrix} \text{LBL \#h6 \& handle} \\ \text{PRED “\_do\_v\_1\_rel”} \\ \text{ARG0 \#e2t} \\ \text{ARG1 \#x14 \& ref-ind} \end{bmatrix} \right)
\end{bmatrix} \\
\text{OUTPUT} & \begin{bmatrix}
\text{HOOK.LTOP \#h1 \& handle} \\
\text{RELS} \left(\begin{bmatrix} \text{LBL \#h6} \\ \text{PRED unspec\_manner\_rel} \\ \text{ARG0 \#e2} \\ \text{ARG1 \#x14} \\ \text{ARG2 \#x7} \end{bmatrix} \right)
\end{bmatrix}
\end{bmatrix}
$$

Figure 3.7: Time restricted transfer rule (how [to do]$_{pres}$ *person_des* do → how [to be]$_{pres}$ *person_des*).

3.3 Dealing with large amounts of transfer rules

The list of transfer rules becomes rather large rather quickly. Dealing with this number of transfer rules, gives rise to two issues:

- How should overlapping rules be handled?
- How can using the rules be sped up?

3.3.1 Handling overlapping rules

The Kaitito system simply uses the first rule it finds in the list of rules that can be applied to the given input sentence. So, if there is more than one rule that would fire given a certain input sentence, all but the first one in the list will never be used. There are two options for this to occur:

- One transfer rule input and context can be subset of the input and context of another transfer rule. In this case the more specific rule can simply be placed earlier in the list than the less specific rule. It probably is possible to come up with examples where actually the less specific rule should be used, but the more specific one would be used when this ordering of rules is in place, because that one can also be applied to the sentence.

- Two transfer rules can have an identical input and context. In this case, the only thing that can be done to differentiate at this point, is adding a EP or HCON to the context of either of the transfer rules. This extra EP or HCON has to come from the list of EPs and HCONs intersecting between the two sentences used to derive the transfer rule. At this point there are two problems:

  - Which of the overlapping rules should be extended?
  - Which of the available EPs and HCONs should be used to extend the rule?

  Addressing the first issue, there is something to be said for choosing either of the rules. Probably the most important criterion is tied together with the second issue, choosing a EP or HCON to differentiate between the rules. Some EPs and HCONs are not appropriate to be used for the differentiating, because they would be especially restricting, like EPs representing pronouns for instance. So, also when choosing which of the rules should be adapted, which EPs and HCONs are available and should be used is an important issue. As mentioned, there are couple of EPs that are not suitable to be used, but choosing between the remaining ones should be done by either a human expert or a very advanced algorithm. Since the idea is that non-linguists should be able to author and maintain the rules, the only suitable option is an advanced algorithm. This is a quite complicated matter and considering overlapping rules are unlikely to occur within this project, resolving this issue is left for further research. It should be noted that adding extra constraints does not always help. Suppose the sentence *Are you married?* needs to be munged to either *Do you have a wife?* or *Do you have a husband?*. The two rules that are derived to accomplish this, completely overlap in input and condition and there is no way to differentiate between them by adding an extra EP or HCON. The only thing that can be done, is take into account the gender of the addressee somehow (although same-sex marriages are also possible these days). On the other hand, one might argue that the choice to have *Are you married* as input rather than output or target is not a logical one.

Note that overlapping rules do not signal an error in the derivation process, but they might point to inconsistencies in the logic behind the sentence pairs chosen to derive rules for. A very simple example of this are the rules derived for *The farmer sees rain* → *The farmer is happy* and *The farmer sees rain* → *The farmer is sad*. Although theoretically something can make a person happy and sad at the same time, it is more likely that something is wrong with the reasons for wanting either of the rules.

3.3.2 Speeding up usage

Checking parses of sentences against the input and context of all rules will become quite slow when the number of rules becomes large. Considering this is something that only becomes a concern when hundreds or even thousands of rules are used simultaneously, the actual implementation is left for future work, but some thoughts on this are:

- Transfer rules could be put in some sort of map with key words that should be a the sentence, before considering checking the complete input and context of the transfer rules, because just checking for a word is computationally less heavy.

- Transfer rules could be divided into the theme categories they belong to, thus lowering the number of transfer rules that needs to be loaded during a dialogue as long as the rules loaded shift according to the topic changed in the dialogue.

4

Implementing semi-automatic derivation of transfer rules

As concluded in Chapter 2, the best way to improve the language understanding capabilities of Kaitito seems to be to use transfer rules. Further more, Chapter 3 shows that automating the derivation of these transfer rules seems profitable. This chapter describes the process of implementing the automatic derivation and the problems encountered during this implementation.

Section 4.1 deals with the matching of the input and output MRS used to make up the transfer rules. Section 4.2 describes the implementation of the actual derivation process.

4.1 Matching MRSs

Even though the existing code already contained a set of functions that matched MRSs, a new set of matching functions has been defined. The reason behind this is that the original matching functions could only be used to determine whether one MRS is part of another MRS or a complete match with the other MRS. In order to derive the transfer rules, a lot of additional information is necessary which is now gathered by the new set of matching functions.

The next section describes the basic structure that is used to represent the matches and explains the basics of matching. After that, the more complicated matching of EP lists and HCONs are explained in Sections 4.1.2 and 4.1.3 respectively. Finally, Section 4.1.4 discusses the issues encountered during the implementation of the matching.

4.1.1 Match structure and basics of matching

The new set of matching functions classifies the parts of an MRS into the different components of a match struct, working upwards from argument values. So using a bottom-up approach, a nested match struct is made in which referents, extra information on things like entities and events, EPs, and HCONs[1] will all appear, see Appendix B for an example.

As mentioned, matching results in a nested structure with all elements from both MRSs in it and the components of such a match struct are:

Matching a complete match, containing pairs of matched items

[1] Albeit the HCONs and the referents in them appear in a structure of their own, separate from the structure with the EPs and extra information

Not-matching not a complete match and not left-only or right-only, can be a partial match

Left-only something only occurring in the input MRS

Right-only something only occurring in the output MRS

Bindings containing the pairs of IDs that have been successfully linked together in this match

For an example of the full structures that result from matching two example sentences, see Appendix B

Making the matching this elaborate also allows for things to partially match, which is very convenient because otherwise a lot of bindings would unjustly be left out. When matching *how do you do* onto *how are you*, while both MRSs amongst other EPs have an `unspec_manner_rel`, these EPs will be considered `not-matching`. When looking at the EPs (`h6:unspec_manner_rel(e2 u8 x7)` and `h6:unspec_manner_rel(e2 x8 x7)`) the reason for this is obvious, since the two EPs have a different type of referent as value of their second argument. Note that the IDs of the referents line-up pretty neatly, but that is just sheer coincidence. However, in this case the first and third argument values should indeed be bound to each other. Because the matching works bottom-up, even though the EPs themselves will be put into the `not-matching` component, a level down the first and third argument values will show up `matching` and contribute `(2 . 2)` and `(7 . 7)` to the bindings. These bindings are useful for issuing new IDs later on. The other main part of the resulting matching structure that is useful, is the list of EPs that are qualified `matching`. This is the intersection of EPs that will be removed from both input and output MRS of the transfer rule. The next section deals with matching the EPs present in the MRSs.

4.1.2 Matching EP lists

Matching EP lists starts at the level of referents and the extra information that might be linked to the entities and events they refer to. When two referents from the MRSs are considered a match, their IDs are added to a bindings list. This list is used for further matching to ensure that no referent is matched more than once and to match up HCONs. Further on this list of bindings is also used in the derivation process, mainly to rename referents.

The next level up is the level of matching within single EPs that have arguments with the referents as their values. Matching EPs begins with matching the names of the predicates and if successful all arguments are compared afterward.

Finally, the entire lists of EPs from both MRSs are matched to get a match in chunks at EP level.

4.1.3 Matching HCONs

When the EP lists have been matched, the HCONs are matched using the existing bindings. This is a rather straight forward match, since HCONs consist of two referents and a link between them. So if the link between the referents is of the same type for two HCONs, the equality of the referents is tested next taking into account the existing referent bindings.

Matching HCONs can result in new bindings, just as matching EPs can. It is possible for the first referent of two HCONs to be considered the same, while the second referent is not. When the HCONs contain `qeq` relations (bluntly put 'equals' relations, at the moment

the only type of relation in use) it is obvious that the second referents should be added as a binding. For example, given two HCONs `h1 qeq h3` and `h2 qeq h3` and the only existing binding `(1 . 2)`, because of the equality relations the binding `(3 . 3)` should be added to the match.

4.1.4 Issues with matching

With the current implementations, the matching functions might come up with more than one possible match in case an EP (or set of linked EPs) occurs more than once. For instance when matching the sentence *the girl smiled* onto the somewhat artificial sentence *the girl and the girl smiled*, the matching will not know to which of the two occurrences of *the girl* in the second sentence the single occurrence of *the girl* from the first sentence should be mapped. Therefore, matching these two sentences will return two matches. This way, there can be sentences with a dozen matches. However, the derive functions will always use the first match and discard the other possibilities. The first reason behind this is that it is thought to be too far fetched for this kind of sentences to occur when using the derivation process as intended for this project. Secondly, in most cases even a human would not be able to single out one of the matches as the best one based on any kind of logic (consider which occurrence of *the girl* should be matched in the example above). Thirdly, there might be situations in which it will make a significant difference for the resulting transfer rule and its application, but usually (like with the example of the girls) it does not influence the resulting transfer rule in a bad way anyway. Therefore, this issue is currently put aside, since it does not really seem to pose a problem.

Sometimes the matching functions fail to match referents that should actually be matched. This applies mainly to the ones that are values of the `RSTR` and `BODY` arguments. Some of these arguments will become obsolete; they will not be used anymore in the newer version of the underlying LKB system (see Section 1.1.1), but others will continue to exist and pose a problem. These types of arguments usually do not occur often in a single MRS and moreover, they seem to quite often occur in different EPs in different MRSs. Since the matching of EPs and the underlying levels with the referents starts with matching the predicate names from EPs, if the arguments appear in different EPs there is no way the referents will be bound. Since because of the rarity of occurrence of these arguments there is no alternative route either, these usually end up not matched. No practical solution for this issue that would fix more than it would break has been found yet.

During matching some errors were encountered that were caused by somewhat strange quirks of and inconsistencies in the underlying system and grammar. An example of these errors are the `greatest-common-subtype` errors that complained about certain types not existing. The reason behind this was that when creating MRSs the representation of certain features is altered using a mapping between two sets of representations. For example for the person information, `png.pn: 1sg` is converted to `per: 1` and `num: sg`. When matching and looking for existing types, the latter representation was not recognized, because the first representation was expected. There are couple of options to remedy this, like turning off the mapping at the point where the MRS are created, change the matching code to match the affected features differently or change the function that returns the `greatest-common-subtype` errors to also match the second representation. Since the third option seemed simplest, this option was chosen. Unfortunately this resulted in other errors, because this way somewhere else in the programming the MRS were considered incorrect. So eventually, even though it is

not the nicest of solutions, part of the mapping had to be taken out completely by simply stopping certain mapping files from being loaded. Needless to say, a couple of quirks like this in the underlying system and grammar take quite some time to pinpoint and remedy.

4.2 Deriving transfer rules

Since in essence transfer rules are not much more than a (partial) input and output MRS, for the first step of the algorithm from Section 3.2.4 nothing needs to be done except when printing the rules (see Section 4.2.4). The approach to the subsequent steps is described below in Section 4.2.1 through to 4.2.3. After that, printing and using the derived rules is described in Section 4.2.4 and finally in Section 4.2.5 issues that arose during the implementation of the derivation process are discussed.

4.2.1 Stripping the transfer rule

Based on the match that has been made between the two MRSs, it is quite easy to remove the parts from both MRS that completely overlap. All matching EPs and HCONs can always be removed from both MRSs, because when parts of these are needed to make the rule more specific, their place is in the condition part of the rule and not in the input and/or output part of the rule. There are cases where HCONs that do not overlap contain handles that only appear in the EPs intersecting between both MRSs. In those cases, the intersecting EP in question is added to the input condition.

Apart from entire EPs and HCONs, also the extra information on variables like person and gender are removed, unless they are absolutely necessary. Being right-only is the one reason for the extra information on variables to be kept, because this means that this is information newly introduced to the munged MRS and thus needs to be fully specified. At first this was done using printing the transfer rule as a filter, since from the print certain features were left-out already. Due to issues with reading in the printed rules (for one thing the transfer rule ID was not read, although that was fixed easily enough), this proved to be an inappropriate approach. Carefully selecting the extra information that could be deleted form MRSs without making them invalid, the MRSs in the internal representation are now stripped of the extra information themselves.

4.2.2 Issuing new IDs

When issuing new IDs, first the entire input MRS is dealt with. Referents occur as handles from EPs, but also in the index of the MRS and as values of arguments of EPs. The old ID of referents that have been given a new ID get added to a map of "already seen" referents. This map is used to correctly give other referents either another, entirely new ID or an ID as listed in the map if applicable. With the "already seen" map completely filled after issuing new IDs for all the EPs, issuing new IDs for the HCONs is very straightforward. After all, the MRS would not be valid when the HCONs contain referents that occur nowhere else in the MRS.

The output MRS has its own map of "already seen" referents that works the same way as with the input MRS. When issuing new IDs for the output MRS, the bindings list has to be taken into account as well. When a binding between a referent from the input and output MRS exists the new ID has to be the same for both referents, so the ID is looked up

through the bindings list and through the "already seen" map from the input MRS. Issuing new IDs for the HCONs of course works exactly the same way as with the input MRS, using the "already seen" map from the output MRS.

4.2.3 Adding context

Since the usefulness of having the functionality of the context of transfer rules operational is presumed to be limited within this project, the implementation of this part is limited too. Only the specification of the `INPUT-CONDITION` will be considered, the `INPUT-FILTER` can always be implemented in a similar way later and besides, in a newer version of the system the context might be specified using more advanced methods (see Section 3.2.2), which will probably deprecate the `INPUT-FILTER` altogether anyway.

To allow the condition to be specified by a non-linguist, deriving the context for now is done by presenting all words from the input sentence to the author. The author is then asked to select one of those words to make the rule more specific. This is a somewhat crude method and it does not use the full power of the context by far.

Next, the system will derive the actual condition that has to have the form of an MRS, with all the handle and referent IDs correctly lined-up with those in the input and output MRS. There is no way to directly link a single word to a certain EP (or a couple of relations) of an MRS created for a full sentence, so deriving the actual condition calls for a more elaborate approach. First, an MRS is created for the selected word. Since this MRS might be more elaborate than it should be, due to the nature of the grammar that always considers its input to be a full sentence, first the condition MRS has to be checked against the original version of the input MRS to cross out all relations that should not be present. Because the original input MRS cannot be easily stored in a way that would allow it to be retrieved again for later use long after the initial rule was written, the original input sentence (and for that matter the output sentence too) is stored in the transfer rule ID, together with the chosen parse of this sentence. In Section 3.2.2 it was explained that every transfer rule gets a unique ID and supposing that there is no use for two or more different rules, derived from the same two sentences, using these sentences in the ID ensures this uniqueness. The transfer rule IDs are thus of the following format, where $\#_1$ and $\#_2$ denote the numbers of the chosen parses of the sentences and the underscores are added because the ID is not allowed to contain spaces:

$$\texttt{first\_sentence\_}\#_1\sim\texttt{second\_sentence\_}\#_2$$

Furthermore, the condition MRS needs to be matched with the current input MRS, since the condition only needs to contain things that are not already present in the current input MRS. Also, there might already be a condition MRS. The relations in this existing condition MRS will have to be merged with the ones that are newly added.

Finally, the IDs of the handles and referents in the condition MRS need to be lined-up with the ones already present in the input and output MRS and possibly even with an existing condition MRS. To this end one more match is needed, matching the original input MRS and the current MRS. Using the bindings between the condition MRS and the original input MRS, those between the original input MRS and the current input MRS and those between the possibly already existing condition MRS and the new condition MRS, the IDs of the handles and referents can be changed.

4.2.4 Printing and using transfer rules

Since a transfer rule basically consists of two MRSs that need to be printed, for printing the transfer rules a set of functions could be used that greatly resembles existing functions for printing MRSs. A new printing class was made with an additional attribute and a global variable for already seen referents to allow for things to only be printed the first time the occur. Only the first time every referent is printed, it has to be accompanied by a specification of its type and leaving all but the first out really cleans up the rules. More important is of course that when some new referent is introduced in the output MRS a full specification of its values and if applicable extra information need to be printed as well. The set of printing functions was rewritten and the printing methods adapted so that the transfer rules would be printed using the correct syntax.

The rules can be printed to a file in the format humans would have to write them and later on be read using functions that would add an internal representation of the rule to the `*ordered-mrs-rule-list*`. Instead of printing and reading the rules, they can also directly be loaded into the `*ordered-mrs-rule-list*` and used to munge MRSs encountered.

4.2.5 Issues with deriving transfer rules

Since deriving a transfer rule is rather straightforward when the matching has already been done correctly, giving the correct bindings and overlapping relations and HCONs, there is not much worth mentioning here.

However, there are some things implemented earlier that were adapted for the derivation process to run smoothly:

- The matching functions needed to be altered slightly to allow for matching with the current input MRS in the context derivation. Because the current input MRS is stripped of some extra information (like gender, see Section 4.2.1), this current input MRS will never match correctly with the original input MRS that still contains this extra information. Therefore, an optional flag was put in place to signal whether the matching would need to ignore the extra information or not.

- The system needs to know what new IDs to give to the IDs that are not bound through bindings when changing the IDs of handles and referents in the condition MRS. For issuing new IDs, a counter is used and somehow the value of this counter needs to be stored to allow for it to be used here. The counter is now stored in the transfer rule ID, which now looks like this, with $\#_3$ as the ID counter:

$$\texttt{first\_sentence\_}\#_1\sim\texttt{second\_sentence\_}\#_2\texttt{\_}\#_3$$

5

Evaluating the appropriateness of derived transfer rules

In Section 1.2 the following goal was set for this project:

> How can Kaitito get better at finding answers in its knowledge base that only partially match their corresponding questions, but are relevant to them nonetheless?

A solution has been designed and implemented: deriving transfer rules to paraphrase the input on the semantic level, so that partially matching answers should become more likely to be found.

The main thing about these transfer rules that needs to be evaluated, is whether the new part of Kaitito is able to derive rules that effectively rephrase input that should be rephrased, but leave alone sentences that should not. Besides this evaluation from a linguistic point of view, the usability should also be considered. After all, it will be the domain experts that fill the knowledge base and derive the rules to paraphrase the input. The general usability issue for the new part of Kaitito can be split into two smaller issues:

- Can the rules (easily enough) be derived by the domain experts?
- Can the collection of rules be maintained by the domain experts?

For both usability issues, adding context to the rules is a major underlying matter. The current level at which the context can be defined is rather unrefined, but this should change in the near future. For now however, this means that even though the evaluation should include these issues, they should not be considered too much of a crucial component for the success or failure of this project. Unfortunately, defining the context of course also influences how well the rules themselves will function, although this is expected not to pose too big a problem for the domain currently used. Furthermore, the usability is highly dependent on the interface the domain experts use. At this point, the natural language interface is not available yet and an elaborate user test is thus out of the question. Therefore, it is decided to let this part of the evaluation be and concentrate on the technical backbone and the options this offers to facilitate the usability.

Concentrating on the technical backbone means testing:

- whether the rules necessary for the introduction domain can be derived using the automated algorithm
- whether the derived rules do what they are supposed to do

The next section describes the actual test design in more detail. Section 5.2 describes the results from testing.

5.1 Test design

The main objective is to test whether rules sufficient for at least the domain of introductions can be derived. The following two sections will deal with the methods used to execute this test and the corpus used to perform the test respectively.

5.1.1 Method of evaluation

Using a corpus (see the next section) of input sentences and sentences that are used as a target (the output sentence for the rule), a wide range of sentences within the domain of introductions will be tested. First each transfer rule needs to be created using an appropriate input and target sentence, all derived transfer rules will be stored. After that, two different tests have to be executed:

- Sentences in some way similar to the original input sentence will be given to the system which either munges the sentence, or leave it unchanged. The system will only use one transfer rule at a time for this test to purposefully avoid possible conflicts between different transfer rules. Besides counting the number of input sentences that are munged correctly, the number of false positives and false negatives, the exact causes behind the wrongly munged input sentences need to be discovered as well as the severeness of the wrong munge.
- The full list of derived rules will be checked for overlap which might result in conflicts when all rules would have been used at once. Where a conflict might arise, it will be checked whether the conflict is solvable by changing either or both rules slightly and how easily this can be done. This should pick up on rules that, even though designed for another type of sentences, affect sentences that should be left alone.

Note that using this design, everything about the test is conducted by the same person that designed and implemented the automatic derivation, although a few other people have been involved in selecting the test sentences. This is just a pilot test, but from the earliest test on, tests should be conducted as thoroughly as possible. The intense involvement of the designer gives rise to some validity issues, but there is no easy way to escape those in this case. On the other hand, the involvement of the designer in testing the system can have its advantages too. The designer is very into the material and thus has an advantage when trying to find loopholes in the system. The system can be evaluated more properly when all components are up and running later on.

5.1.2 Corpus for evaluation

In order to test the derived rules, first the rules need to be derived. Choosing which rules are to be derived is done based on the corpus gathered for the design phase of this project (see Section 3.1) and a bit of common sense. Based on the frequencies of answers, the target input sentence is selected and all possible, often occurring, deviating input sentences are used as input to derive a transfer rule with the chosen target. As a base, at least for the sentences where it is appropriate, the sentences will be put in second person since this is the form most often used in dialogues. Also in principle the derived rules will not contain any context specification at all, unless absolutely necessary. The list of sentences used to derived rules can be found in Table 5.1.

The second part of the corpus consists of sentences that strongly resemble the input sentences used to derive the transfer rules, but differ slightly in a number of ways:

- person (you/he/she/we, etc.)
- subject type (pronoun/named entity/proverb and noun)
- tense (present, past, past perfect, etc.)
- question word (what/which, etc.)
- statement instead of question
- additional information in the sentence (like prepositional phrases: *at the meeting*)

These variations are based on the rationale behind the second step of the derivation algorithm (stripping the MRSs), where all extra information that contains things like person and tense are removed (see Section 3.2.4). The slightly differing sentences, ranging from 5 (an exception for *How is life?*) to commonly about 20 to 40 sentences, have mostly been made up and should or should not be munged just like the original input sentence. All sentences have been tagged by hand, deciding whether they indeed should be munged according to a certain rule or not.

Table 5.1: Evaluation corpus, input for transfer rules.

Input	**Output**
How are things?	How are you?
How is life?	How are you?
How do you do?	How are you?
How are you doing?	How are you?
How are you feeling?	How are you?
How do you feel?	How are you?
Do you know English?	Can you speak English?
What languages do you know?	What languages can you speak?
Are you English?	Are you from England?
Where do you come from?	Where were you born?
Do you come from London?	Were you born in London?
When were you born?	How old are you?
Do you have (any) hobbies?	What do you enjoy?
What do you do?	What are you?
Tell me (something) about yourself?	What do you enjoy?
Tell me (something) about your family?	Do you have children?
Are you married?	Do you have a wife/husband?

There are some conditions that apply to sentences in both parts of the corpus, due to the nature of Kaitito:

- All sentences need to be self contained, apart from simple anaphora denoting persons or things.

- All sentences need to be correct English.
- All sentences need to be parsable by Kaitito and the system must be able to generate back from these parses.

These conditions are things that can all be guaranteed when using Kaitito as a dialogue system because of the special modules the system will have (see Section 1.1.2). Only the last condition is a bit tricky when viewed in light of the future use of the system. Even though sentences might be correct English, Kaitito could not be able to parse them due to for instance lack of vocabulary and. A second problem is, that Kaitito sometimes is unable to generate a sentence back from a parse it made itself. The occasions this poses a problem are rare, but they do occur.

5.2 Test results

This section briefly describes the results from testing the system using the sentences and test corpus put together in Section 5.1.2. Section 5.2.1 describes the results from the first part of the evaluation: deriving and testing single transfer rules. Section 5.2.2 deals with the second part of the evaluation: overlapping and conflict between rules.

5.2.1 Deriving and using individual rules

A brief summary of the test results can be found in Table 5.2. The trial sentences that did not perform satisfactorily in some way, are discussed in more detail below.

As already predicted in Section 3.2.5, more specific transfer rules are needed to accommodate sentences that include prepositional phrases (as for both *How are things?* → *How are you?* and *How is life?* → *How are you?*). As long as authors know about this, it should not pose a problem at all, because it is easy to remedy. For instance, in *How are things with John?* and *How were things at the meeting?* the prepositional phrases add some extra intra-sentence relations, which make the MRS representations of the sentences munged using the rule derived from *How are things?* → *How are you?* invalid. Strictly speaking, the system does not see these MRSs as invalid, but it can not generate the actual sentences from them. By deriving new rules like *How are things with John?* → *How is John?* and putting these rules in the rule list before the more general rule for *How are things?* → *How are you?*, this problem can be circumvented. Note that the sentence *How are things with John?* can also have a meaning more like *How is your relationship with John.* Obviously, for that meaning the proposed rule does not give the correct sentence. This is an issue that could never be addressed unless the dialogue system has a very sophisticated topic analyzer, able to distinguish the user's intentions based on context and even in that case it would be very hard. On the other hand, this is a mistake a human could also make and when occurring in a conversation between two people, the question would be reformulated to clearly transcribe the intention behind it. However, this does not necessarily mean the same will happen when a user is in a dialogue with an artificial character.

Table 5.2: Evaluation results.

Trial sentences	**Result**
How are things? → How are you?	OK, but needs additional, specific rules for input containing prepositional phrases.
How is life? → How are you?	OK, but needs additional, specific rules for input containing prepositional phrases.
How do you do? → How are you?	OK.
How are you doing? → How are you?	OK, but overgenerating (system can deal with this).
How are you feeling? → How are you?	OK.
How do you feel? → How are you?	OK.
Do you know English? → Can you speak English?	OK.
What languages do you know? → What languages can you speak?	OK.
Are you English? → Are you form England?	OK.
Where do you come from? → Where were you born?	OK.
Do you come from London? → Were you born in London?	OK, but prone to fire on occasions it should not.
When were you born? → How old are you?	OK.
Do you have (any) hobbies → What do you enjoy?	FAULTY, rule is derived but sentences do not generate.
What do you do? → What are you?	OK, but prone to fire on occasions it should not.
Tell me (something) about yourself? → What do you enjoy?	FAULTY, rule is derived but sentences do not generate.
Tell me (something) about your family? → Do you have children?	FAULTY, rule is derived but sentences do not generate.
Are you married → Do you have a wife/husband?	FAULTY, rule is derived but sentences do not generate.

The rule derived for *How are you doing?* → *How are you?* overgenerates and thus also gives the sentences *You are how?* and *Are you how?* as well as *How are you?*. Unfortunately, this is a quirk in the grammar used and nothing can be done about that at this point. However, all three sentences are (sort of) correct and the dialogue system will pick the best one, so this does not pose a problem when used within the dialogue system (but it might for other applications).

The rule for derived for *Do you come from London?* → *Were you born in London?* will also fire for a sentence like *Do you come from home?*, giving *Were you born in home?*. This rule should thus be made specific for names of places, countries and world parts, but that is quite hard to do. The rule could be made specific for named entities, but even then things can go wrong (*Do you come from John?*, short for *Do you come from John's house?*).[1] The rule derived for *What do you do?* → *What are you?* has a similar problem. This rule also fires on sentences like *What do you do for a living?* and *What do you do to make it taste like that?*. From munging *What do you do for a living?* the system does not manage to generate, but after munging *What do you do to make it taste like that?*, the system actually returns *What are you to make it taste like that?*, *To make it taste like that, what are you?* and *In order to make it taste like that, what are you?*. Clearly, these sentences are incorrect and both that and the rule firing but not generating from the munged sentences will make the system fail to find correct answers to questions.
An at least partial solution to these problems might be to use the munging as a back-up plan for finding correct responses. That way, the dialogue system would first try to find a direct match between the user input and the knowledge base and only if this fails, the system would see if it could munge the user input and try to find a match between the munged sentence (if any) and the knowledge base. A variation to this is, that the system tries to find a response for both the user input itself and a munged version of the user input, and afterward chooses the best response.

The few rules that completely fail, all get derived but somehow the system is unable to generate from sentences munged using these rules. This means there is a mistake somewhere in the derived rule[2] and so far, the cause (or causes) has not been found. The principal for this project decided that for now, this is something that needs not to be looked into further.

5.2.2 Overlap and conflict between rules

In the tested corpus, there originally is no overlap between rules. After introducing the extra rules that need to be derived because of the prepositional phrases, some overlap is present. As mentioned, those extra, more specific rules should simply be put higher up on the list of rules than the less specific ones. There was no sentence for which the condition slot of a transfer rule needed to be used. Moreover, apart from situations in which HCONs needed relations from the intersection to be put in the condition slot, the condition slot was not used at all. None of the test rules needed the rule to be made specific for person or a specific word.

[1] Note that in this kind of sentences normally the auxiliary verb 'have' would be used (*Have you (just) come from home?*, *Where have you come from?*). But people not so familiar with the English language might use these sentences anyhow and the grammar allows for them too.

[2] Not being able to generate from a munged sentence, is not always caused by a mistake in the rule. In this case however, both the input and output sentence used to derive the rule can be parsed and the system is able to generate back from these parses. When testing the derived rule with the exact same sentence that was used as input sentence to derive the rule, there is thus no other explanation for the munged sentence being faulty than that the rule is faulty.

Using tests involving the transfer rule IDs that contain the original input sentences or the input MRSs of the transfer rules, it can be determined whether a newly derived rule is in a more specific version of an existing rule. That way the rules can be ordered correctly automatically. Earlier on in the design this idea was also brought forward, but it has not been implemented yet.

6 Conclusions and future work

During the execution of this project, a lot of choices have been made, based upon previous related research and the specifications of the existing Kaitito system. Most of these are open for discussion and some have given rise to other decisions yet to be made.

The more important, fundamental issues are dealt with in this chapter. The next section (Section 6.1) describes issues that have been dealt with already, Section 6.2 describes those that need to be dealt with in the future.

6.1 Conclusions and discussion

The Kaitito system can only deal properly with input that can be mapped quite literally onto the knowledge it has (see Section 1.1.2). Given the specifications of the system and the various options, it makes sense to rewrite the non-matching user input to input that will match. This rewriting or paraphrasing should be done at the semantic level, using the Minimal Recursion Semantics (MRS) representations of sentences. Paraphrasing at the MRS level involves transfer rules (also called munge rules), that are a quite tricky to derive by hand and time consuming too. Furthermore, to support the Kaitito rationale, it is necessary for non-linguists to be able to author the rules too. Thus the rules need to be derived automatically and when functioning properly enough, this can be used together with the authoring module (see Section 1.1.2). That would allow non-linguists to, for instance, differentiate between possible parses without having to look at the parse trees.

The main problem inherent in using transfer rules to improve Kaitito's input understanding capabilities is that some rules tend to fire on occasions they should not. Sometimes this can be solved by making the rules more specific (for instance to concern only named entities), but the level of detail needed for this is not always available using the current way transfer rules need to be defined. So, there are times when just making the rules more specific is not an option. There are a couple of approaches to this problem:

- Use a very sophisticated topic analyzer able to grasp the user's intentions with a certain utterance based on the utterance context, so that it can be decided whether to deploy the appropriate transfer rule or not.

- Use transfer rules as a back-up plan that only starts working when trying to find a response to user input using the input itself directly fails.

- Count on the user to reformulate the utterance as a user would when in conversation with another human.

A similar problem would have always existed, regardless whether the paraphrasing was used on another linguistic level (surface, syntax or inference) or any other type of rule-based system was used. Therefore, even though it is a problem, it is not a reason to think another approach should have been taken.

Another, in some ways related problem, is that it is quite hard to make the derivation process, even though mainly automated, suitable for use by non-linguists. The fact that sometimes surprising things cause for the need to derive separate rules, might not be very intuitive. For instance, this is usually the case with prepositional phrases; the rule derived for *How were things?* to *How are you?* cannot be applied to *How are things with John?*. There is not much more that can be done about this and compared to other approaches, semi-automatic derivation of transfer rules actually is quite a reasonable one.

Finally, there is also the problem where the same input utterance might need to be rewritten to different sentences (a 1:n mapping) depending on the context. For instance *Are you married?* might be rewritten to *Do you have a wife?*, but that is quite awkward when asked to a female character (except when the woman in question is a known lesbian). This is solvable using the topic analyzer and taking into account the addressee before deciding which transfer rule to deploy. However, this decreases the usability of authoring the rules, because the rules need to be marked to belong to different groups that are only used under certain circumstances. For the simple dialogues, this does not pose a problem yet, but when learners get more advanced and the user utterances less predictable, it will.

Apart from issues inherent in the current design, there is also some sort of implementation error still present (although it is not certain whether the error comes from the system code, or the newly developed transfer rule deriving code). There are some rules that get derived and get applied to input, but the resulting representations cannot be used to generate a natural language sentence back from. However, the principal for this project decided that for now, this is something that needs not to be looked into further. The main reason behind this is that the rules that were derived based upon the gathered corpus all work, and only some of the extra made-up rules seem to fail.

6.2 Recommendations and future work

Apart from the issues already mentioned above that need some tweaking, there are some other things that could use some work done:

- An algorithm to automatically adapt derived transfer rules when there is overlap between an existing rule and a newly derived rule should be developed. When the input specification and condition are a subset of the input specification and condition of another rule, the more specific rule should simply be placed higher up in the transfer rule list. When two rules completely overlap concerning the input specification and condition, there should be some sort of algorithm that decides that a certain relation occurring in the intersection between input and output of one rule (but not in the intersection between input and output of the other rule) should be added to the input condition. Of course some relations are not suitable to be used for that, because they would restrict the rule while it should not (in most cases this would include relations denoting pronouns for instance).

- A way to correctly handle multiple matches between two MRSs needs to be designed and implemented. So far there have not been many examples of MRS pairs that resulted in multiple matches, but it will occur more often with more complicated sentences. Simply choosing the first match and use that one, is a solution that is unsuitable in the end. The system might try to derive a rule using the first match and if that rule works for the input and output sentence used to derive the rule, adopt that rule as a functioning one. In case the rule does not work for the input and output sentences used to derive the rule, the second match should be used to derive a rule, that rule should be tested, and so on.

- An approach to handling vast amount of transfer rules should be designed and implemented (see Section 3.3 for a preliminary discussion of this). A logical choice would be to do something with topic dependence, so categorizing rules for topics and addressee and use a topic analyzer to decide which set of rules may be applicable.

- Test the automatic derivation in combination with the authoring module (see Section 1.1.2), using possible users (English teachers) as test subjects. This is something that can only be done by the time the system is developed far enough to facilitate for user trials with the authoring module up and running properly.

- Test how difficult it will be for English teachers to use the logic behind deriving rules and to cope with the fact that for instance for propositional phrases, extra rules are needed.

Bibliography

Alfonsi, B. (2006). "Sassy" chatbot wins with wit. *IEEE Intelligent systems*, *21*(1), 6–7.

Anaya, G. B., & Kosseim, L. (2003, June). Generation of natural responses through syntactic patterns. In *Actes de la 10ème conférence annuelle sur le traitement automatique des langues naturelles (TALN'2003).* Batz-sur-Mer, France.

Angelova, G., Strupchanska, A., Kalaydjiev, O., Yankova, M., Boytcheva, S., Vitanova, I., et al. (2004). Towards deeper understanding and personalisation in ecall. In *Proceedings of "eLearning for Computational Linguistics and Computational Linguistics for eLearning", workshop at COLING 2004* (pp. 45–52).

Bouma, G. (2006, April). Linguistic knowledge and question answering. In *Proceedings of the workshop kraq'06: Knowledge and reasoning for language processing* (pp. 2–3). Association for Computational Linguistics.

Boytcheva, S., Vitanova, I., Strupchanska, A., Yankova, M., & Angelova, G. (2004). Towards the assessment of free learner's utterances in call. In *Proceedings of instil/icall2004 - nlp and speech technologies in advanced language learning systems* (pp. 187–190).

Brill, E., Dumais, S., & Banko, M. (2002). An analysis of the AskMSR question-answering system. In *EMNLP'02: Proceedings of the ACL-02 conference on Empirical Methods in Natural Language Processing* (pp. 257–264). Morristown, NJ, USA: Association for Computational Linguistics.

Brill, E., Lin, J. J., Banko, M., Dumais, S. T., & Ng, A. Y. (2001). Data-intensive question answering. In *Trec.*

Clarke, C., Cormack, G., Lynam, T., Li, C., & McLearn, G. (2001). Web reinforced question answering (multitext experiments for trec 2001). In Voorhees & Buckland (Eds.), *Text REtrieval Conference.* Gaithesburg, MD: NIST.

Copestake, A. (1995). *Semantic transfer in Verbmobil* (Tech. Rep. No. Verbmobil Report 93). IMS, Universität Stuttgart, Germany and CSLI, Stanford University.

Copestake, A. (last update 2006). *Robust minimal recursion semantics.* Retrieved on 29-03-2007, from: http://www.cl.cam.ac.uk/~aac10/papers/rmrsdraft.pdf.

Copestake, A., Carroll, J., Malouf, R., Oepen, S., et al. (2000). *The (new) LKB system.* CSLI, Stanford University.

Copestake, A., Flickinger, D., Malouf, R., Riehemann, S., & Sag, I. (1995). Translation using minimal recursion semantics. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation.* Leuven, Belgium.

Copestake, A., Flickinger, D., Pollard, C., & Sag, I. A. (2005). Minimal Recursion Semantics: An introduction. *Journal of Research on Language and Computation*, *3*(4), 281 – 332.

Core, M. G., & Moore, J. D. (2004, May 2 - May 7). Robustness versus fidelity in natural

language understanding. In R. Porzel (Ed.), *HLT-NAACL 2004 Workshop: 2nd Workshop on Scalable Natural Language Understanding* (pp. 1–8). Boston, Massachusetts, USA: Association for Computational Linguistics.

Duclaye, Y. F., & Collin, O. (2002). Using the web as linguistic resource for learning reformulations automatically. In *Proceedings of the third international conference on Language Resources and Evaluation (LREC'02)* (pp. 390–396).

Echihabi, A., Hermjakob, U., Hovy, E., Marcu, D., Melz, E., & Ravichandran, D. (2006). How to select an answer string? In *Advances in open domain question answering* (pp. 383–406). Netherlands: Springer.

Echihabi, A., & Marcu, D. (2003). A noisy-channel approach to question answering. In *Acl '03: Proceedings of the 41st annual meeting on association for computational linguistics* (pp. 16–23). Morristown, NJ, USA: Association for Computational Linguistics.

Fahmi, I., & Bouma, G. (2006). Learning to identify definitions using syntactic features. In R. Basili & A. Moschitti (Eds.), *Proceedings of the EACL workshop on Learning Structured Information in Natural Language Applications.*

Flickinger, D., Lønning, J. T., Dyvik, H., Oepen, S., & Bond, F. (2005). SEM-I rational MT. Enriching deep grammars with a semantic interface for scalable machine translation. In *Proceedings of the 10th Machine Translation Summit* (pp. 165–172). Phuket, Thailand.

Fryer, L., & Carpenter, R. (2006). Emerging technologies: Bots as language learning tools. *Language Learning & Technology*, *10*(3), 8–14.

Galea, A. (2003). Open-domain surface-based question answering system. In G. J. Pace & J. Cordina (Eds.), *Proceedings of CSAW'03.* Malta: Department of Computer Science & A.I., University of Malta.

Gockley, R., Simmons, R., Wang, J., Busquets, D., DiSalvo, C., Caffrey, K., et al. (2004). Grace and george: Social robots at aaai. In F. Crabbe, W. D. Smart, & S. Tejada (Eds.), *Aaai mobile robot competition and exhibition: Papers from the 2004 aaai workshop* (pp. 15–20). Menlo Park, CA: AAAI Press.

Graesser, A., Wiemer-Hastings, P., Wiemer-Hastings, K., Harter, D., Person, N., & the Tutoring Research Group. (2000). Using Latent Semantic Analysis to evaluate the contributions of students in AutoTutor. *Interactive Learning Environments*, *8*(2), 129–147.

Greenwood, M. A., & Saggion, H. (2004, April). A pattern based approach to factoid, list and definition question answering. In *RIAO 2004: conference proceedings* (p. 232-243). Paris, France: Le Centre de Hautes Études Internationales d'Informatique Documentaire – C.I.D.

Grois, E. (2005, June). Learning strategies for open-domain natural language question answering. In *Proceedings of the ACL Student Research Workshop* (pp. 85–90). Ann Arbor, Michigan: Association for Computational Linguistics.

Ham, E. van der. (2005, June). *Diagnosing and responding to student errors in a dialogue-based computer-aided language-learning system* (Tech. Rep. No. OUCS-2005-06). Department of Computer Science, University of Otago.

Harabagiu, S. M., Moldovan, D. I., Pasca, M., Surdeanu, M., Mihalcea, R., Girju, R., et al. (2001). Answering complex, list and context questions with lcc's question-answering server. In *Trec.*

Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *Proceeding of the 14th Conference on Computational Linguistics* (pp. 539–545). Morristown, NJ, USA: Association for Computational Linguistics.

Hermjakob, U., Echihabi, A., & Marcu, D. (2002). Natural language based reformulation

resource and web exploitation for question answering. In E. Vorhees & L. P. Buckland (Eds.), *Proceedings of the eleventh text retrieval conference (TREC 2002)* (pp. 670–677). Gaithersburg, MD.: NIST.

Hovy, E. H., Hermjakob, U., & Ravichandran, D. (2002). A question/answer typology with surface text patterns. In *Proceedings of the darpa human language technology conference (hlt).* San Diego, CA.

Hu, D., Wenyin, L., Chen, E., Chen, X., & Li, X. (2006). Siipu*s: A semantic pattern learning algorithm. In *Second international conference on semantics, knowledge, and grid (SKG'06)* (p. 52). Los Alamitos, CA, USA: IEEE Computer Society.

Iida, R., Inui, K., Iwakura, T., Fujita, A., & Takahashi, T. (2001). Kura: A lexico-structural paraphrasing engine. In *Proceedings of the NLPRS2001 Workshop on Automatic Paraphrasing: Theories and Applications* (p. 763-764).

Ittycheriah, A., & Roukos, S. (2002). IBM's statistical question answering system – TREC-11. In Voorhees & Buckland (Eds.), *Proceedings of the trec-11 conference.* Gaithesburg, MD: NIST.

Jijkoun, V., Mur, J., & Rijke, M. D. (2004). Information extraction for question answering: Improving recall through syntactic patterns. In *COLING-2004* (pp. 1284–1290).

Jurafsky, D., & Martin, J. H. (2000). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.* Upper Saddle River, NJ: Prentice Hall.

Kamp, H., & Reyle, U. (1993). *From discourse to logic.* Dordrecht, The Netherlands: Kluwer Academic Publishers.

Knott, A., Bayard, I., Jager, S. de, & Wright, N. (2002). An architecture for bilingual and bidirectional NLP. In S. Geldof & D. Mollá-Aliod (Eds.), *Proceedings of ANLO2002* (pp. 63–70).

Knott, A., & Vlugter, P. (2007). *Dialogue-based authoring strategies for dialogue agents.* (Unpublished manuscript)

Leveling, J. (2006). The role of information retrieval in the question answering system irsaw. In *Proceedings of the lwa 2006 (learning, knowledge, and adaptability), workshop information retrieval* (p. 119-125). Hildesheim, Germany: Universität Hildesheim.

Lin, C. (2002). The effectiveness of dictionary and web-based answer reranking. In *Proceedings of the 19th international conference on computational linguistics* (pp. 1–7). Morristown, NJ, USA: Association for Computational Linguistics.

Lin, D., & Pantel, P. (2001). Discovery of inference rules for question-answering. *Natural Language Engineering*, *7*(4), 343–360.

Marton, G. A., Tellex, S., Fernandes, A., & Katz, B. (2005, September). Hypernyms as answer types. In *Proceedings of the 1st annual CSAIL Student Workshop (CSW05).*

Moldovan, D., Harabagiu, S., Girju, R., Morarescu, P., Lacatusu, F., Novischi, A., et al. (2002). Lcc tools for question answering. In E. M. Voorhees & Buckland (Eds.), *Proceedings of the 11th Text REtrieval Conference (TREC-2002)* (pp. 144–155). Gaithersburg, MD: NIST.

Mollá, D. (2006). Learning of graph-based question answering rules. In *Proceedings of the hlt/naacl 2006 workshop on graph algorithms for natural language processing* (pp. 37–44).

Mollá, D., & Zaanen, M. van. (2005). Learning of graph rules for question answering. In *Proceedings of ALTW 2005.*

Nyberg, E., & Frederking, R. (2003, May 27 - June 1). JAVELIN: A flexible, planner-

based architecture for question answering. In B. Frederking & B. Younger (Eds.), *HLT-NAACL 2003: Demo session* (pp. 19–20). Edmonton, Alberta, Canada: Association for Computational Linguistics.

Nygaard, L., Lønning, J. T., Nordgård, T., & Oepen, S. (2006). Using a bi-lingual dictionary in lexical transfer. In *Proceedings of the 11th conference of the European Association for Machine Translation.* Oslo, Norway.

Oepen, S., Dyvik, H., Lønning, J. T., Velldal, E., Beermann, D., Carroll, J., et al. (2004, October). Som å kapp-ete med trollet? Towards MRS-based Norwegian–English Machine Translation. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation.* Baltimore, MD.

Olney, A., Louwerse, M., Matthews, E., Marineau, J., Hite-Mitchell, H., & Graesser, A. (2003). Utterance classification in autotutor. In *Proceedings of the hlt-naacl 03 workshop on building educational applications using natural language processing* (pp. 1–8). Morristown, NJ, USA: Association for Computational Linguistics.

Pan, S., & Shaw, J. C. (2004). Segue: A hybrid case-based surface natural language generator. In *Proceedings of the third international conference of natural language generation (inlg)* (pp. 130–140).

Pan, S., & Shaw, J. C. (2007). Natural language query recommendation in conversation systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1701–1706).

Pan, S., Shen, S., Zhou, M. X., & Houck, K. (2005). Two-way adaptation for robust input interpretation in practical multimodal conversation systems. In *Iui '05: Proceedings of the 10th international conference on intelligent user interfaces* (pp. 35–42). New York, NY, USA: ACM Press.

Plamondon, L., & Lapalme, G. (2002). The quantum question answering system at trec-11. In E. Voorhees & D. Harman (Eds.), *Proceedings of the eleventh text retrieval conference (TREC 2002)* (pp. 670–677).

Purver, M., Ratiu, F., & Cavedon, L. (2006, September). Robust interpretation in dialogue by combining confidence scores with contextual features. In *Proceedings of the 9th International Conference on Spoken Language Processing (Interspeech/ICSLP)* (pp. 1–4). Pittsburgh, PA.

Ravichandran, D., & Hovy, E. (2002). Learning surface text patterns for a question answering system. In *ACL '02: proceedings of the 40th annual meeting on association for computational linguistics* (pp. 41–47). Morristown, NJ, USA: Association for Computational Linguistics.

Ravichandran, D., Ittycheriah, A., & Roukos, S. (2003). Automatic derivation of surface text patterns for a maximum entropy based question answering system. In *NAACL '03: proceedings of the 2003 conference of the north american chapter of the association for computational linguistics on human language technology* (pp. 85–87). Morristown, NJ, USA: Association for Computational Linguistics.

Rinaldi, F., Dowdall, J., Kaljurand, K., Hess, M., & Mollá;, D. (2003). Exploiting paraphrases in a question answering system. In *Proceedings of the second international workshop on paraphrasing* (pp. 25–32). Morristown, NJ, USA: Association for Computational Linguistics.

Rosé, C. P., Gaydos, A., Hall, B. S., Roque, A., & VanLehn, K. (2003). Overcoming the knowledge engineering bottleneck for understanding student language input. In *Proceedings of the 11 International Conference on Artificial Intelligence in Education (AIED'03).*

Seneff, S., Wang, C., Peabody, M., & Zue, V. (2004). Second language acquisition through human computer dialogue. In *Proceedings of the 4th International Symposium on Chinese Spoken Language Processing (ISCSLP-2004).*

Shaw, E., Johnson, W. L., & Ganeshan, R. (1999). Pedagogical agents on the web. In *Agents '99: Proceedings of the third annual Conference on Autonomous Agents* (pp. 283–290). New York, NY, USA: ACM Press.

Shen, D., Kruijff, G.-J. M., & Klakow, D. (2005, October). Exploring syntactic relation patterns for question answering. In R. Dale, K.-F. Wong, J. Su, & O. Y. Kwong (Eds.), *Natural Language Processing IJCNLP 2005: Second International Joint Conference.* Springer-Verlag.

Si, M., Marsella, S. C., & Pynadath, D. V. (2005). Thespian: using multi-agent fitting to craft interactive drama. In *AAMAS'05: Proceedings of the fourth international joint conference on Autonomous Agents and MultiAgent Systems* (pp. 21–28). New York, NY, USA: ACM Press.

Slabbers, N. (2005). *A system for generating teaching initiatives in a computer-aided language learning dialogue* (Tech. Rep. No. OUCS-2005-02). Department of Computer Science, University of Otago.

Soubbotin, M. M., & Soubbotin, S. M. (2001). Patterns of potential answer expressions as clues to the right answers. In *Trec-10 notebook papers.* Gaithesburg, MD: NIST.

Soubbotin, M. M., & Soubbotin, S. M. (2002). Use of patterns for detection of answer strings: A systematic approach. In Voorhees & Buckland (Eds.), *Proceedings of the trec-11 conference.* Gaithesburg, MD: NIST.

Stymne, S., & Ahrenberg, L. (2006). A bilingual grammar for translation of english-swedish verb frame divergences. In *Proceedings of the 11th annual conference of the European Association for Machine Translation (EAMT)* (pp. 9–18).

Vlugter, P., Knott, A., & Weatherall, V. (2004). A human-machine dialogue system for call. In *InSTIL/ICALL 2004 Symposium on Computer Assisted Learning* (p. 050).

Voorhees, E. M., & Harman, D. K. (Eds.). (2005). *TREC: Experiment and evaluation in information retrieval.* Cambridge, MA: MIT Press.

Wallace, R. S. (2000). *The anatomy of a.l.i.c.e.* Retrieved on 27-03-2007, from: http://www.alicebot.org/documents/Anatomy.doc. A.L.I.C.E. AI Foundation.

Weizenbaum, J. (1966). Eliza - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45.

Yang, H., & Chua, T.-S. (2002, November). The integration of lexical knowledge and external resources for question answering. In *Proceedings of the eleventh text retrieval conference (TREC'2002)* (pp. 155–161).

Zacharski, R. (2002). *Conversational agents for language learning.* Retrieved on 16-03-2007, from http://www.zacharski.org/papers/IAAI-2002.pdf.

A
MRSs for Kaitito

The following are all MRS representations of *every living person has some parents* as produced by the LKB system set up for usage with Kaitito using the ERG grammar for English. The MRS representations vary in level of specificity.

A.1 Indexed MRS output

The first MRS representation looks most like the simple example from Section 3.2.1 (which looked like ⟨{`h1:every(x,h3,hA), h3:living(x), h3:person(x), h7:parents(y), h5:some(y,h7,hB), h4:have(x,y)`}, {`(hA:h5), (hB:h4)`}⟩).

```
<h1,e2:PRES:INDICATIVE:-:-,
{h1:prop-or-ques_m(e2, h4, u3, u5),
h6:_every_q(x7, h9, h8),
h10:_live_v_1(e11, x7, i12),
h13:prpstn_m(e11, h14, u16, u15),
h13:_person_n_1(x7),
h17:_have_v_1(e2, x7, x18),
h19:_some_q(x18, h20, h21),
h22:_parent_n_of(x18, i23)},
{h4 qeq h17,
h9 qeq h13,
h14 qeq h10,
h20 qeq h22}>
```

Figure A.1: Indexed MRS output of Kaitito

A.2 Simple MRS output

The second MRS representation is a bit more complicated as it is incorporated in a typed feature structure design.

```
[ LTOP: h1
  INDEX: e2 [ e TENSE: PRES MOOD: INDICATIVE PROG: - PERF: - ]
  RELS: <
         [ prop-or-ques_m_rel
           LBL: h1
           ARG0: e2
           MARG: h4
           PSV: u3
           TPC: u5 ]
         [ _every_q_rel
           LBL: h6
           ARG0: x7
           RSTR: h9
           BODY: h8 ]
         [ "_live_v_1_rel"
           LBL: h10
           ARG0: e11
           ARG1: x7
           ARG2: i12 ]
         [ prpstn_m_rel
           LBL: h13
           ARG0: e11
           MARG: h14
           PSV: u16
           TPC: u15 ]
         [ "_person_n_1_rel"
           LBL: h13
           ARG0: x7 ]
         [ "_have_v_1_rel"
           LBL: h17
           ARG0: e2
           ARG1: x7
           ARG2: x18 ]
         [ _some_q_rel
           LBL: h19
           ARG0: x18
           RSTR: h20
           BODY: h21 ]
         [ "_parent_n_of_rel"
           LBL: h22
           ARG0: x18
           ARG1: i23 ] >
  HCONS: < h4 qeq h17 h9 qeq h13 h14 qeq h10 h20 qeq h22 > ]
```

Figure A.2: Simple MRS output of Kaitito

A.3 Full MRS output

The Kaitito system has an even more detailed and complicated version of MRSs, which is a direct print of the internal representation.

```
#S(PSOA
  :TOP-H #S(VAR :TYPE "h" :EXTRA NIL :ID 1)
  :LISZT (#S(CHAR-REL
         :PRED LKB::PROP-OR-QUES_M_REL
         :FLIST (#S(FVPAIR
             :FEATURE LKB::ARG0
             :VALUE #S(VAR
                   :TYPE "e"
                   :EXTRA (#S(EXTRAPAIR
                       :FEATURE LKB::TENSE
                       :VALUE LKB::PRES)
                     #S(EXTRAPAIR
                       :FEATURE LKB::MOOD
                       :VALUE LKB::INDICATIVE)
                     #S(EXTRAPAIR
                       :FEATURE PROG
                       :VALUE -)
                     #S(EXTRAPAIR
                       :FEATURE LKB::PERF
                       :VALUE -))
                   :ID 2))
           #S(FVPAIR
             :FEATURE LKB::MARG
             :VALUE #S(VAR :TYPE "h" :EXTRA NIL :ID 4))
           #S(FVPAIR
             :FEATURE LKB::PSV
             :VALUE #S(VAR :TYPE "u" :EXTRA NIL :ID 3))
           #S(FVPAIR
             :FEATURE LKB::TPC
             :VALUE #S(VAR :TYPE "u" :EXTRA NIL :ID 5)))
         :STR NIL
         :HANDEL #S(VAR :TYPE "h" :EXTRA NIL :ID 1)
         :ANCHOR NIL
         :PARAMETER-STRINGS NIL
         :EXTRA NIL
         :LINK NIL
         :CFROM 0
         :CTO 36)
       #S(CHAR-REL
         :PRED LKB::_EVERY_Q_REL
         :FLIST (#S(FVPAIR
             :FEATURE LKB::ARG0
```

```
      :VALUE #S(VAR
            :TYPE "x"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::PERS
                :VALUE LKB::|3|)
              #S(EXTRAPAIR
                :FEATURE LKB::NUM
                :VALUE LKB::SG)
              #S(EXTRAPAIR
                :FEATURE LKB::DIV
                :VALUE -))
            :ID 7))
    #S(FVPAIR
      :FEATURE LKB::RSTR
      :VALUE #S(VAR :TYPE "h" :EXTRA NIL :ID 9))
    #S(FVPAIR
      :FEATURE LKB::BODY
      :VALUE #S(VAR :TYPE "h" :EXTRA NIL :ID 8)))
  :STR NIL
  :HANDEL #S(VAR :TYPE "h" :EXTRA NIL :ID 6)
  :ANCHOR NIL
  :PARAMETER-STRINGS NIL
  :EXTRA NIL
  :LINK NIL
  :CFROM 0
  :CTO 5)
#S(CHAR-REL
  :PRED "_live_v_1_rel"
  :FLIST (#S(FVPAIR
      :FEATURE LKB::ARG0
      :VALUE #S(VAR
            :TYPE "e"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::TENSE
                :VALUE LKB::UNTENSED)
              #S(EXTRAPAIR
                :FEATURE LKB::MOOD
                :VALUE LKB::INDICATIVE)
              #S(EXTRAPAIR
                :FEATURE PROG
                :VALUE +)
              #S(EXTRAPAIR
                :FEATURE LKB::PERF
                :VALUE -))
            :ID 11))
    #S(FVPAIR
      :FEATURE LKB::ARG1
```

```
      :VALUE #S(VAR
            :TYPE "x"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::PERS
                :VALUE LKB::|3|)
              #S(EXTRAPAIR
                :FEATURE LKB::NUM
                :VALUE LKB::SG)
              #S(EXTRAPAIR
                :FEATURE LKB::DIV
                :VALUE -))
            :ID 7))
    #S(FVPAIR
      :FEATURE LKB::ARG2
      :VALUE #S(VAR :TYPE "i" :EXTRA NIL :ID 12)))
  :STR NIL
  :HANDEL #S(VAR :TYPE "h" :EXTRA NIL :ID 10)
  :ANCHOR NIL
  :PARAMETER-STRINGS NIL
  :EXTRA NIL
  :LINK NIL
  :CFROM 6
  :CTO 12)
#S(CHAR-REL
  :PRED LKB::PRPSTN_M_REL
  :FLIST (#S(FVPAIR
      :FEATURE LKB::ARG0
      :VALUE #S(VAR
            :TYPE "e"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::TENSE
                :VALUE LKB::UNTENSED)
              #S(EXTRAPAIR
                :FEATURE LKB::MOOD
                :VALUE LKB::INDICATIVE)
              #S(EXTRAPAIR
                :FEATURE PROG
                :VALUE +)
              #S(EXTRAPAIR
                :FEATURE LKB::PERF
                :VALUE -))
            :ID 11))
    #S(FVPAIR
      :FEATURE LKB::MARG
      :VALUE #S(VAR :TYPE "h" :EXTRA NIL :ID 14))
    #S(FVPAIR
      :FEATURE LKB::PSV
```

```
      :VALUE #S(VAR :TYPE "u" :EXTRA NIL :ID 16))
    #S(FVPAIR
      :FEATURE LKB::TPC
      :VALUE #S(VAR :TYPE "u" :EXTRA NIL :ID 15)))
  :STR NIL
  :HANDEL #S(VAR :TYPE "h" :EXTRA NIL :ID 13)
  :ANCHOR NIL
  :PARAMETER-STRINGS NIL
  :EXTRA NIL
  :LINK NIL
  :CFROM 6
  :CTO 12)
#S(CHAR-REL
  :PRED "_person_n_1_rel"
  :FLIST (#S(FVPAIR
      :FEATURE LKB::ARG0
      :VALUE #S(VAR
            :TYPE "x"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::PERS
                :VALUE LKB::|3|)
              #S(EXTRAPAIR
                :FEATURE LKB::NUM
                :VALUE LKB::SG)
              #S(EXTRAPAIR
                :FEATURE LKB::DIV
                :VALUE -))
            :ID 7)))
  :STR NIL
  :HANDEL #S(VAR :TYPE "h" :EXTRA NIL :ID 13)
  :ANCHOR NIL
  :PARAMETER-STRINGS NIL
  :EXTRA NIL
  :LINK NIL
  :CFROM 13
  :CTO 19)
#S(CHAR-REL
  :PRED "_have_v_1_rel"
  :FLIST (#S(FVPAIR
      :FEATURE LKB::ARG0
      :VALUE #S(VAR
            :TYPE "e"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::TENSE
                :VALUE LKB::PRES)
              #S(EXTRAPAIR
                :FEATURE LKB::MOOD
```

```
                :VALUE LKB::INDICATIVE)
              #S(EXTRAPAIR
                :FEATURE PROG
                :VALUE -)
              #S(EXTRAPAIR
                :FEATURE LKB::PERF
                :VALUE -))
            :ID 2))
    #S(FVPAIR
      :FEATURE LKB::ARG1
      :VALUE #S(VAR
            :TYPE "x"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::PERS
                :VALUE LKB::|3|)
              #S(EXTRAPAIR
                :FEATURE LKB::NUM
                :VALUE LKB::SG)
              #S(EXTRAPAIR
                :FEATURE LKB::DIV
                :VALUE -))
            :ID 7))
    #S(FVPAIR
      :FEATURE LKB::ARG2
      :VALUE #S(VAR
            :TYPE "x"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::PERS
                :VALUE LKB::|3|)
              #S(EXTRAPAIR
                :FEATURE LKB::NUM
                :VALUE LKB::PL)
              #S(EXTRAPAIR
                :FEATURE LKB::DIV
                :VALUE +))
            :ID 18)))
  :STR NIL
  :HANDEL #S(VAR :TYPE "h" :EXTRA NIL :ID 17)
  :ANCHOR NIL
  :PARAMETER-STRINGS NIL
  :EXTRA NIL
  :LINK NIL
  :CFROM 20
  :CTO 23)
#S(CHAR-REL
  :PRED LKB::_SOME_Q_REL
  :FLIST (#S(FVPAIR
```

```
      :FEATURE LKB::ARG0
      :VALUE #S(VAR
            :TYPE "x"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::PERS
                :VALUE LKB::|3|)
              #S(EXTRAPAIR
                :FEATURE LKB::NUM
                :VALUE LKB::PL)
              #S(EXTRAPAIR
                :FEATURE LKB::DIV
                :VALUE +))
            :ID 18))
    #S(FVPAIR
      :FEATURE LKB::RSTR
      :VALUE #S(VAR :TYPE "h" :EXTRA NIL :ID 20))
    #S(FVPAIR
      :FEATURE LKB::BODY
      :VALUE #S(VAR :TYPE "h" :EXTRA NIL :ID 21)))
  :STR NIL
  :HANDEL #S(VAR :TYPE "h" :EXTRA NIL :ID 19)
  :ANCHOR NIL
  :PARAMETER-STRINGS NIL
  :EXTRA NIL
  :LINK NIL
  :CFROM 24
  :CTO 28)
#S(CHAR-REL
  :PRED "_parent_n_of_rel"
  :FLIST (#S(FVPAIR
      :FEATURE LKB::ARG0
      :VALUE #S(VAR
            :TYPE "x"
            :EXTRA (#S(EXTRAPAIR
                :FEATURE LKB::PERS
                :VALUE LKB::|3|)
              #S(EXTRAPAIR
                :FEATURE LKB::NUM
                :VALUE LKB::PL)
              #S(EXTRAPAIR
                :FEATURE LKB::DIV
                :VALUE +))
            :ID 18))
    #S(FVPAIR
      :FEATURE LKB::ARG1
      :VALUE #S(VAR :TYPE "i" :EXTRA NIL :ID 23)))
  :STR NIL
```

```
        :HANDEL #S(VAR :TYPE "h" :EXTRA NIL :ID 22)
        :ANCHOR NIL
        :PARAMETER-STRINGS NIL
        :EXTRA NIL
        :LINK NIL
        :CFROM 29
        :CTO 36))
:H-CONS (#S(HCONS
         :RELATION "qeq"
         :SCARG #S(VAR :TYPE "h" :EXTRA NIL :ID 4)
         :OUTSCPD #S(VAR :TYPE "h" :EXTRA NIL :ID 17))
       #S(HCONS
         :RELATION "qeq"
         :SCARG #S(VAR :TYPE "h" :EXTRA NIL :ID 9)
         :OUTSCPD #S(VAR :TYPE "h" :EXTRA NIL :ID 13))
       #S(HCONS
         :RELATION "qeq"
         :SCARG #S(VAR :TYPE "h" :EXTRA NIL :ID 14)
         :OUTSCPD #S(VAR :TYPE "h" :EXTRA NIL :ID 10))
       #S(HCONS
         :RELATION "qeq"
         :SCARG #S(VAR :TYPE "h" :EXTRA NIL :ID 20)
         :OUTSCPD #S(VAR :TYPE "h" :EXTRA NIL :ID 22)))
:A-CONS NIL
:VCS NIL
:INDEX #S(VAR
       :TYPE "e"
       :EXTRA (#S(EXTRAPAIR :FEATURE LKB::TENSE :VALUE LKB::PRES)
         #S(EXTRAPAIR
           :FEATURE LKB::MOOD
           :VALUE LKB::INDICATIVE)
         #S(EXTRAPAIR :FEATURE PROG :VALUE -)
         #S(EXTRAPAIR :FEATURE LKB::PERF :VALUE -))
       :ID 2))
```

Figure A.3: Full MRS output of Kaitito

B
Match example

Example result from matching *how do you do* and *how are you*.

Relations list match

```
(#S(MATCH
      :MATCHING ((h15:pronoun_q_rel(x14 h16 h17) h14:pronoun_q_rel(x8 h15 h16)
      #S(MATCH
           :MATCHING ((#S(FVPAIR :FEATURE LKB::BODY :VALUE h17)
               #S(FVPAIR :FEATURE LKB::BODY :VALUE h16)
               #S(MATCH
                :MATCHING NIL
                :NOT-MATCHING NIL
                :LEFT-ONLY NIL
                :RIGHT-ONLY NIL
                :BINDINGS ((17 . 16))))
              (#S(FVPAIR :FEATURE LKB::RSTR :VALUE h16)
               #S(FVPAIR :FEATURE LKB::RSTR :VALUE h15)
               #S(MATCH
                :MATCHING NIL
                :NOT-MATCHING NIL
                :LEFT-ONLY NIL
                :RIGHT-ONLY NIL
                :BINDINGS ((16 . 15))))
              (#S(FVPAIR :FEATURE LKB::ARG0 :VALUE x14)
               #S(FVPAIR :FEATURE LKB::ARG0 :VALUE x8)
               #S(MATCH
                :MATCHING ((#S(EXTRAPAIR
                      :FEATURE LKB::PRONTYPE
                      :VALUE LKB::STD_PRON)
                     . #S(EXTRAPAIR
                        :FEATURE LKB::PRONTYPE
                        :VALUE LKB::STD_PRON))
                    (#S(EXTRAPAIR
                      :FEATURE LKB::PERS
                      :VALUE LKB::|2|)
```

```
              . #S(EXTRAPAIR
                 :FEATURE LKB::PERS
                 :VALUE LKB::|2|)))
           :NOT-MATCHING NIL
           :LEFT-ONLY NIL
           :RIGHT-ONLY NIL
           :BINDINGS ((14 . 8)))))
      :NOT-MATCHING NIL
      :LEFT-ONLY NIL
      :RIGHT-ONLY NIL
      :BINDINGS ((15 . 14) (14 . 8) (16 . 15) (17 . 16))))
   (h13:pron_rel(x14) h13:pron_rel(x8)
#S(MATCH
      :MATCHING ((#S(FVPAIR :FEATURE LKB::ARG0 :VALUE x14)
          #S(FVPAIR :FEATURE LKB::ARG0 :VALUE x8)
          #S(MATCH
           :MATCHING ((#S(EXTRAPAIR
                :FEATURE LKB::PRONTYPE
                :VALUE LKB::STD_PRON)
              . #S(EXTRAPAIR
                 :FEATURE LKB::PRONTYPE
                 :VALUE LKB::STD_PRON))
             (#S(EXTRAPAIR
                :FEATURE LKB::PERS
                :VALUE LKB::|2|)
              . #S(EXTRAPAIR
                 :FEATURE LKB::PERS
                 :VALUE LKB::|2|)))
           :NOT-MATCHING NIL
           :LEFT-ONLY NIL
           :RIGHT-ONLY NIL
           :BINDINGS ((14 . 8)))))
      :NOT-MATCHING NIL
      :LEFT-ONLY NIL
      :RIGHT-ONLY NIL
      :BINDINGS ((13 . 13) (14 . 8))))
   (h12:manner_rel(x7) h12:manner_rel(x7)
#S(MATCH
      :MATCHING ((#S(FVPAIR :FEATURE LKB::ARG0 :VALUE x7)
          #S(FVPAIR :FEATURE LKB::ARG0 :VALUE x7)
          #S(MATCH
           :MATCHING NIL
           :NOT-MATCHING NIL
           :LEFT-ONLY NIL
           :RIGHT-ONLY NIL
           :BINDINGS ((7 . 7)))))
      :NOT-MATCHING NIL
```

```
     :LEFT-ONLY NIL
     :RIGHT-ONLY NIL
     :BINDINGS ((12 . 12) (7 . 7))))
   (h9:which_q_rel(x7 h10 h11) h9:which_q_rel(x7 h10 h11)
#S(MATCH
     :MATCHING ((#S(FVPAIR :FEATURE LKB::BODY :VALUE h11)
         #S(FVPAIR :FEATURE LKB::BODY :VALUE h11)
         #S(MATCH
          :MATCHING NIL
          :NOT-MATCHING NIL
          :LEFT-ONLY NIL
          :RIGHT-ONLY NIL
          :BINDINGS ((11 . 11))))
        (#S(FVPAIR :FEATURE LKB::RSTR :VALUE h10)
         #S(FVPAIR :FEATURE LKB::RSTR :VALUE h10)
         #S(MATCH
          :MATCHING NIL
          :NOT-MATCHING NIL
          :LEFT-ONLY NIL
          :RIGHT-ONLY NIL
          :BINDINGS ((10 . 10))))
        (#S(FVPAIR :FEATURE LKB::ARG0 :VALUE x7)
         #S(FVPAIR :FEATURE LKB::ARG0 :VALUE x7)
         #S(MATCH
          :MATCHING NIL
          :NOT-MATCHING NIL
          :LEFT-ONLY NIL
          :RIGHT-ONLY NIL
          :BINDINGS ((7 . 7)))))
     :NOT-MATCHING NIL
     :LEFT-ONLY NIL
     :RIGHT-ONLY NIL
     :BINDINGS ((9 . 9) (7 . 7) (10 . 10) (11 . 11))))
   (h3:prpstn_m_rel(e2 h5 u4 e2) h3:prpstn_m_rel(e2 h5 u4 e2)
#S(MATCH
     :MATCHING ((#S(FVPAIR :FEATURE LKB::TPC :VALUE e2)
         #S(FVPAIR :FEATURE LKB::TPC :VALUE e2)
         #S(MATCH
          :MATCHING ((#S(EXTRAPAIR
               :FEATURE LKB::PERF
               :VALUE -)
              . #S(EXTRAPAIR
                 :FEATURE LKB::PERF
                 :VALUE -))
             (#S(EXTRAPAIR
               :FEATURE PROG
               :VALUE -)
```

```
      . #S(EXTRAPAIR
          :FEATURE PROG
          :VALUE -))
     (#S(EXTRAPAIR
       :FEATURE LKB::MOOD
       :VALUE LKB::INDICATIVE)
      . #S(EXTRAPAIR
          :FEATURE LKB::MOOD
          :VALUE LKB::INDICATIVE))
     (#S(EXTRAPAIR
       :FEATURE LKB::TENSE
       :VALUE LKB::PRES)
      . #S(EXTRAPAIR
          :FEATURE LKB::TENSE
          :VALUE LKB::PRES)))
  :NOT-MATCHING NIL
  :LEFT-ONLY NIL
  :RIGHT-ONLY NIL
  :BINDINGS ((2 . 2))))
(#S(FVPAIR :FEATURE LKB::PSV :VALUE u4)
 #S(FVPAIR :FEATURE LKB::PSV :VALUE u4)
 #S(MATCH
  :MATCHING NIL
  :NOT-MATCHING NIL
  :LEFT-ONLY NIL
  :RIGHT-ONLY NIL
  :BINDINGS ((4 . 4))))
(#S(FVPAIR :FEATURE LKB::MARG :VALUE h5)
 #S(FVPAIR :FEATURE LKB::MARG :VALUE h5)
 #S(MATCH
  :MATCHING NIL
  :NOT-MATCHING NIL
  :LEFT-ONLY NIL
  :RIGHT-ONLY NIL
  :BINDINGS ((5 . 5))))
(#S(FVPAIR :FEATURE LKB::ARG0 :VALUE e2)
 #S(FVPAIR :FEATURE LKB::ARG0 :VALUE e2)
 #S(MATCH
  :MATCHING ((#S(EXTRAPAIR
       :FEATURE LKB::PERF
       :VALUE -)
      . #S(EXTRAPAIR
          :FEATURE LKB::PERF
          :VALUE -))
     (#S(EXTRAPAIR
       :FEATURE PROG
       :VALUE -)
```

```
                . #S(EXTRAPAIR
                    :FEATURE PROG
                    :VALUE -))
               (#S(EXTRAPAIR
                  :FEATURE LKB::MOOD
                  :VALUE LKB::INDICATIVE)
                . #S(EXTRAPAIR
                    :FEATURE LKB::MOOD
                    :VALUE LKB::INDICATIVE))
               (#S(EXTRAPAIR
                  :FEATURE LKB::TENSE
                  :VALUE LKB::PRES)
                . #S(EXTRAPAIR
                    :FEATURE LKB::TENSE
                    :VALUE LKB::PRES)))
            :NOT-MATCHING NIL
            :LEFT-ONLY NIL
            :RIGHT-ONLY NIL
            :BINDINGS ((2 . 2)))))
      :NOT-MATCHING NIL
      :LEFT-ONLY NIL
      :RIGHT-ONLY NIL
      :BINDINGS ((3 . 3) (2 . 2) (5 . 5) (4 . 4) (2 . 2))))
   (h1:basic_int_m_rel(e2 h3 u4 e2)
h1:basic_int_m_rel(e2 h3 u4 e2)
#S(MATCH
      :MATCHING ((#S(FVPAIR :FEATURE LKB::TPC :VALUE e2)
          #S(FVPAIR :FEATURE LKB::TPC :VALUE e2)
          #S(MATCH
           :MATCHING ((#S(EXTRAPAIR
                 :FEATURE LKB::PERF
                 :VALUE -)
                . #S(EXTRAPAIR
                    :FEATURE LKB::PERF
                    :VALUE -))
               (#S(EXTRAPAIR
                 :FEATURE PROG
                 :VALUE -)
                . #S(EXTRAPAIR
                    :FEATURE PROG
                    :VALUE -))
               (#S(EXTRAPAIR
                 :FEATURE LKB::MOOD
                 :VALUE LKB::INDICATIVE)
                . #S(EXTRAPAIR
                    :FEATURE LKB::MOOD
                    :VALUE LKB::INDICATIVE))
```

```
      (#S(EXTRAPAIR
        :FEATURE LKB::TENSE
        :VALUE LKB::PRES)
       . #S(EXTRAPAIR
          :FEATURE LKB::TENSE
          :VALUE LKB::PRES)))
  :NOT-MATCHING NIL
  :LEFT-ONLY NIL
  :RIGHT-ONLY NIL
  :BINDINGS ((2 . 2))))
(#S(FVPAIR :FEATURE LKB::PSV :VALUE u4)
 #S(FVPAIR :FEATURE LKB::PSV :VALUE u4)
 #S(MATCH
  :MATCHING NIL
  :NOT-MATCHING NIL
  :LEFT-ONLY NIL
  :RIGHT-ONLY NIL
  :BINDINGS ((4 . 4))))
(#S(FVPAIR :FEATURE LKB::MARG :VALUE h3)
 #S(FVPAIR :FEATURE LKB::MARG :VALUE h3)
 #S(MATCH
  :MATCHING NIL
  :NOT-MATCHING NIL
  :LEFT-ONLY NIL
  :RIGHT-ONLY NIL
  :BINDINGS ((3 . 3))))
(#S(FVPAIR :FEATURE LKB::ARG0 :VALUE e2)
 #S(FVPAIR :FEATURE LKB::ARG0 :VALUE e2)
 #S(MATCH
  :MATCHING ((#S(EXTRAPAIR
        :FEATURE LKB::PERF
        :VALUE -)
       . #S(EXTRAPAIR
          :FEATURE LKB::PERF
          :VALUE -))
      (#S(EXTRAPAIR
        :FEATURE PROG
        :VALUE -)
       . #S(EXTRAPAIR
          :FEATURE PROG
          :VALUE -))
      (#S(EXTRAPAIR
        :FEATURE LKB::MOOD
        :VALUE LKB::INDICATIVE)
       . #S(EXTRAPAIR
          :FEATURE LKB::MOOD
          :VALUE LKB::INDICATIVE))
```

```
                  (#S(EXTRAPAIR
                    :FEATURE LKB::TENSE
                    :VALUE LKB::PRES)
                   . #S(EXTRAPAIR
                      :FEATURE LKB::TENSE
                      :VALUE LKB::PRES)))
               :NOT-MATCHING NIL
               :LEFT-ONLY NIL
               :RIGHT-ONLY NIL
               :BINDINGS ((2 . 2)))))
          :NOT-MATCHING NIL
          :LEFT-ONLY NIL
          :RIGHT-ONLY NIL
          :BINDINGS ((1 . 1) (2 . 2) (3 . 3) (4 . 4) (2 . 2)))))
:NOT-MATCHING ((h6:unspec_manner_rel(e2 u8 x7)
        h6:unspec_manner_rel(e2 x8 x7)
        #S(MATCH
         :MATCHING ((#S(FVPAIR :FEATURE LKB::ARG2 :VALUE x7)
             #S(FVPAIR :FEATURE LKB::ARG2 :VALUE x7)
             #S(MATCH
              :MATCHING NIL
              :NOT-MATCHING NIL
              :LEFT-ONLY NIL
              :RIGHT-ONLY NIL
              :BINDINGS ((7 . 7))))
            (#S(FVPAIR :FEATURE LKB::ARG0 :VALUE e2)
             #S(FVPAIR :FEATURE LKB::ARG0 :VALUE e2)
             #S(MATCH
              :MATCHING ((#S(EXTRAPAIR
                    :FEATURE LKB::PERF
                    :VALUE -)
                   . #S(EXTRAPAIR
                      :FEATURE LKB::PERF
                      :VALUE -))
                  (#S(EXTRAPAIR
                    :FEATURE PROG
                    :VALUE -)
                   . #S(EXTRAPAIR
                      :FEATURE PROG
                      :VALUE -))
                  (#S(EXTRAPAIR
                    :FEATURE LKB::MOOD
                    :VALUE LKB::INDICATIVE)
                   . #S(EXTRAPAIR
                      :FEATURE LKB::MOOD
                      :VALUE LKB::INDICATIVE))
                  (#S(EXTRAPAIR
```

```
                    :FEATURE LKB::TENSE
                    :VALUE LKB::PRES)
                   . #S(EXTRAPAIR
                      :FEATURE LKB::TENSE
                      :VALUE LKB::PRES)))
               :NOT-MATCHING NIL
               :LEFT-ONLY NIL
               :RIGHT-ONLY NIL
               :BINDINGS ((2 . 2)))))
          :NOT-MATCHING ((#S(FVPAIR
               :FEATURE LKB::ARG1
               :VALUE u8)
              #S(FVPAIR
               :FEATURE LKB::ARG1
               :VALUE x8)))
          :LEFT-ONLY NIL
          :RIGHT-ONLY NIL
          :BINDINGS ((6 . 6) (2 . 2) (7 . 7)))))
  :LEFT-ONLY (h6:"_do_v_1_rel"(e2 x14))
  :RIGHT-ONLY NIL
  :BINDINGS ((13 . 13) (11 . 11) (10 . 10) (9 . 9) (4 . 4) (5 . 5) (3 . 3)
         (1 . 1) (6 . 6) (2 . 2) (12 . 12) (7 . 7) (15 . 14) (14 . 8)
         (16 . 15) (17 . 16))))
```

HCONS match

```
#S(MATCH
     :MATCHING ((#S(HCONS :RELATION ``qeq'' :SCARG h16 :OUTSCPD h13)
         . #S(HCONS :RELATION ``qeq'' :SCARG h15 :OUTSCPD h13))
        (#S(HCONS :RELATION ``qeq'' :SCARG h10 :OUTSCPD h12)
         . #S(HCONS :RELATION ``qeq'' :SCARG h10 :OUTSCPD h12))
        (#S(HCONS :RELATION ``qeq'' :SCARG h5 :OUTSCPD h6)
         . #S(HCONS :RELATION ``qeq'' :SCARG h5 :OUTSCPD h6)))
     :NOT-MATCHING NIL
     :LEFT-ONLY NIL
     :RIGHT-ONLY NIL
     :BINDINGS ((17 . 16) (16 . 15) (14 . 8) (15 . 14) (7 . 7) (12 . 12)
        (2 . 2) (6 . 6) (1 . 1) (3 . 3) (5 . 5) (4 . 4) (9 . 9)
        (10 . 10) (11 . 11) (13 . 13)))
```

C
Deriving a transfer rule example

The algorithm for deriving transfer rules semi-automatically that was presented in Section 3.2.4, will be explained below in more detail. This explanation uses the same two example sentences as Appendix B, which shows exactly how the match that underlies the derivation process is made. The transfer rule will thus be based upon munging *How do you do?* into *How are you?*.

The algorithm is as follows (when using the derived rules internally only, thus not regarding printing the rule):

1. Put the input and output sentence into the input and output of the transfer rule respectively and give the rule an ID
2. Strip the transfer rule of all things that unnecessarily restrict the transfer rule
3. Give handles and referents new IDs so that only IDs that should match do so
4. Add context to the condition/filter slot of the transfer rule, if applicable

These steps will be illustrated in the sections below.

C.1 Putting everything in the right spot

As explained in Chapter 4, the ID of the transfer rule before reissuing the handle and referent IDs consists of the original sentences that were used to derive the transfer rule and the chosen parse of each sentence. The `INPUT-SPEC` and `OUTPUT-SPEC` are the complete input and output MRSs, although in the printed version they are the indexed representations (see Appendix A) of those MRSs, without the HCONS. The print of the internal representation thus looks like Figure C.1.

```
MRS-MUNGE-RULE
     :ID ‘‘how_do_you_do?_0~how_are_you?_0’’
     :INPUT-SPEC h1:e2:{h1:int_m(e2, h3, u4, e2) h3:prpstn_m(e2, h5, u4,
e2) h6:unspec_manner(e2, u8, x7) h9:which_q(x7, h10, h11) h12:manner(x7)
h13:pron(x14) h15:pronoun_q(x14, h16, h17) h6:_do_v_1(e2, x14)}
     :INPUT-CONDITION NIL
     :OUTPUT-SPEC h1:e2:{h1:int_m(e2, h3, u4, e2) h3:prpstn_m(e2, h5, u4,
e2) h6:unspec_manner(e2, x8, x7) h9:which_q(x7, h10, h11) h12:manner(x7)
h13:pron(x8) h14:pronoun_q(x8, h15, h16)}
```

Figure C.1: The internal representation of the transfer rule with full MRSs.

C.2 Stripping the transfer rule

As explained in Chapter 4, the stripping entails removing the overlapping relations and HCONS, as long as they can be removed without making the resulting MRS invalid. For finding out which are the overlapping relations and HCONS, the two sentences need to be matched. Although the elaborate outcome of this match can be found in Appendix B, the summary of this match is illustrated by Figure C.2. In Figure C.2, the lines connect the re-

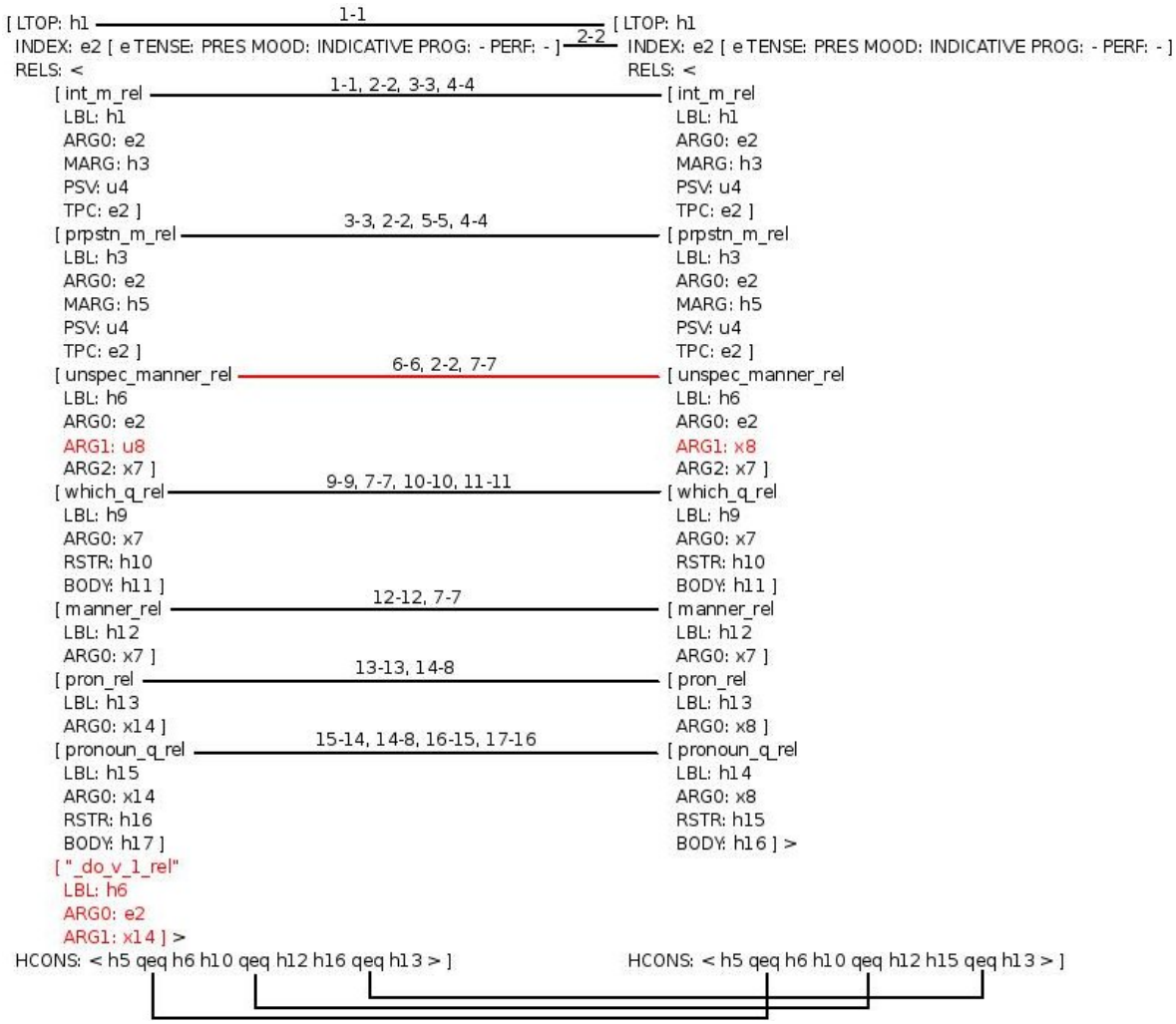

Figure C.2: Summary of matching the sentences *How do you do?* and *How are you.*

lations and HCONS that were matched, the numbers above these lines denote the derived bindings and the color red is used to indicate things that could not be matched. The overlapping relations thus are `int_m`, `prpstn_m`, `which_q`, `manner`, `pron` and `pronoun_q` and all HCONS can be matched.

The print of the internal representation now looks like Figure C.3. The thing that cannot be seen in this representation of the MRSs, is that also some extra information that is found deep inside the MRS (containing information about for instance person, compare the MRSs from Figure A.2 and A.3 in Appendix A for a better idea of what exactly this entails) is

```
MRS-MUNGE-RULE
     :ID ``how_do_you_do?_0~how_are_you?_0''
     :INPUT-SPEC h1:e2:{h6:unspec_manner(e2, u8, x7) h6:_do_v_1(e2, x14)}
     :INPUT-CONDITION NIL
     :OUTPUT-SPEC h1:e2:{h6:unspec_manner(e2, x8, x7)}
```

Figure C.3: The internal representation of transfer rule after stripping it.

removed unless it is present in the `OUTPUT-SPEC` MRS only.

C.3 Issuing new IDs

Based upon the bindings that are found matching the two sentences, the remaining handles and referents are given new IDs. The bindings resulting from the match are:
`((13 . 13) (11 . 11) (10 . 10) (9 . 9) (4 . 4) (5 . 5) (3 . 3) (1 . 1) (6 . 6) (2 . 2) (12 . 12) (7 . 7) (15 . 14) (14 . 8) (16 . 15) (17 . 16))`.

Starting with the first handle in the first MRS, the MRS in the `INPUT-SPEC`, all handles and referents are given a new ID (counting from 100 on), unless it is an handle or referent with an ID that was previously encountered.

```
INPUT-SPEC h1:e2:{h6:unspec_manner(e2, u8, x7) h6:_do_v_1(e2,x14)}
```

becomes

```
INPUT-SPEC h100:e101:{h102:unspec_manner_rel(e101 u103 x104)
                      h102:_do_v_1_rel(e101 x105)}
```

The accompanying bind-list containing the already seen IDs is[1]:

```
((1 . 100) (2 . 101) (6 . 102) (8 . 103) (7 . 104) (14 . 105))
```

Next, the handles and referents of the MRS in the `OUTPUT-SPEC` can be changed, using both the bindings from matching the two sentences and the bind-list from changing the handle and referent IDs from the MRS in the `INPUT-SPEC`. Starting with the first handle, `h1`, through the binding `(1 . 1)` and the bind-list-item `(1 . 100)`, this handle can be given the new ID `100`, and so on. For IDs that cannot be found and reissued this way, again the counter (at the beginning set to 105 where the first MRS left off) used to issue a new ID.

```
OUTPUT-SPEC h1:e2:{h6:unspec_manner(e2, x8, x7) }
```

becomes

```
OUTPUT-SPEC h100:e101:{h102:unspec_manner_rel(e101 x106 x104)}
```

The new version of the print of the internal representation now looks like Figure C.4. Note that `_106` has been added to the ID of the transfer rule, to be able to use the counter later on.

[1] In the Chapter 4 it is explained that actually all new IDs themselves are also put in the bind-list linked to themselves, otherwise when encountering a previously changed handle of referent the ID is changed again and again. This is however strictly an implementation issue that is due to the fact that the handles and referents are pointers and is thus left out here for simplicity.

```
MRS-MUNGE-RULE
     :ID ``how_do_you_do?_0~how_are_you?_0_106''
     :INPUT-SPEC h100:e:101{h102:unspec_manner(e101, u103, x104)
h6:_do_v_1(e101, 105)}
     :INPUT-CONDITION NIL
     :OUTPUT-SPEC h100:e101:{h102:unspec_manner(e101, x106, x104)}
```

Figure C.4: The internal representation of transfer rule after reissuing IDs.

C.4 Adding context

For the sake of illustration, let's say that the rule just derived should be made specific for the second person. When the rule author uses the menu to choose a word, in this case the word *you* would have been chosen. This word has the MRS `h1:e2:{h1:prop-or-ques_m(e2, h3, u5, u4) h6:unknown(e2, x7) h8:pron(x7) h9:pronoun_q(x7, h10, h11)}`. Matching this with the original MRS put into the `INPUT-SPEC`, only the `pron` and `pronoun_q` relation are kept. For reissuing the handle and referent IDs for this MRS, two matches are needed:

- The match between the full MRS of *you* and the original `INPUT-SPEC` MRS
- The match between the original `INPUT-SPEC` MRS and the current `INPUT-SPEC` MRS

In a similar vein as changing the IDs on the `OUTPUT-SPEC` MRS, but with an extra step of lining up bindings in between, the handles and referents of this `INPUT-CONDITION` MRS can be given new IDs, using a counter for newly issued IDs that starts at 106. This results in the final version of the rule, depicted in Figure C.5.

```
MRS-MUNGE-RULE
     :ID ``how_do_you_do?_0~how_are_you?_0_110''
     :INPUT-SPEC h100:e:101{h102:unspec_manner(e101, u103, x104)
h6:_do_v_1(e101, 105)}
     :INPUT-CONDITION h100:e101{h106:pron(x105) h107:pronoun_q(x105, h108,
h109)}
     :OUTPUT-SPEC h100:e101:{h102:unspec_manner(e101, x106, x104)}
```

Figure C.5: The internal representation of transfer rule after adding a condition.