NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET
FAKULTET FOR FYSIKK, INFORMATIKK OG MATEMATIKK

HOVEDOPPGAVE

Kandidatens navn:	Jon S Bratseth
Oppgavens tittel (engelsk):	**Bustuc - A Natural Language Bus Traffic Information System**
Oppgavens tekst:	NTNU has started a three year pilot project TAGORE (Talebaserte Grensesnitt og Resonnerende Systemer) for a speech based user interfaces for transport information, e.g. bus departures and arrivals. Central in this development will be a text based NL interface for the same purpose. Building upon TUC's grammar system, a running sentence based system shall be built through a careful modelling of the semantics and the database.

Oppgaven gitt:	20. September 1996
Besvarelsen leveres innen:	17. Februar 1997
Besvarelsen levert:	17. Februar 1997
Utført ved:	Institutt for datateknikk og informasjonsvitenskap
Veileder:	Tore Amble

Trondheim, 17. Februar 1997

Faglærer

Abstract

The Bustuc project is an attempt to make an application that can answer single bus traffic questions stated as complete natural language sentences. Since the application is accessible for real users via World Wide Web and the arriving questions are logged, it has been possible to show that the current application answers between 70% and 80% of real questions correctly. The Bustuc application uses The Understanding Computer to do lexical, syntactic and semantic analysis of the input sentence. There is both an English and a Norwegian version of the system. The application consists of a pragmatic rule base, a bus route data base, a domain reasoning module and an answer generation rule base. The rule bases are written in a powerful production system called Pragma which has been developed as a part of this project.

Acknowledgment

I want to thank my supervisor, Tore Amble for dedicating much time to improving TUC and testing the Bustuc application, and for answering my numerous email even in the evening.

I would also like to thank my girlfriend, Hilde for showing patience when I have worked long nights.

Jon S Bratseth, 17. February 1997

Contents

CHAPTER 1 Introduction

This chapter will present an introduction to the Bustuc project and this report.

1.1 The Bustuc project

The Bustuc project was given by Tore Amble at the Institute for Computer Science (IDT) at the Norwegian University of Science and Technology (NTNU).

The goal of the project has been to make a working application capable of answering single questions stated in natural language text about bus traffic in Trondheim. The application will be used to serve the public via World Wide Web. This task should be solved by building on the Bustuc prototype made autumn 1996 (Bratseth 96) and by using The Understanding Computer (TUC), a general natural language system that is being developed at IDT by Tore Amble. Versions of the Bustuc application have been available on the World Wide Web during the entire project.

The project is a part of a larger project, Speech-based Interfaces and Reasoning Systems (TAGORE). The aim of TAGORE is to produce a prototype of a speech-based system that can answer queries about public traffic spoken by casual users over telephone.

The motivation for the Bustuc and TAGORE projects is:

- The ability to communicate smoothly and naturally with computers using natural language will be extremely useful for humans in many situations. This goal is still far ahead, but the obvious starting point is to achieve good performance on this task for a very limited domain, like in this project.
- A working natural language system capable of giving helpful answers to bus traffic questions asked by casual users is something that will be of great value. Every week, Trondheim Trafikkselskap receives about 300 phone requests for route information,

and the public transport traffic service in Oslo, Oslo og Akershus Trafikkservice receives as much as 25900[1] calls.

1.2 Thesis

For an application to be successful in the domain of public traffic information, the main task is to understand a wide variety of different formulations. This is because the users are casual and must be assumed to have no computer experience, so they can neither know or guess a particular subset of natural language that the system understands. Also, the problems the users can have in the domain are quite simple, but the language they uses is rich and informal, since this is a domain common in daily speech.

We claim that it is possible to make a system that succeeds at this task by using TUC and by building on earlier work with the Bustuc system. The success of the system should be possible to show by running statistics on the use of the application on World Wide Web.

1.3 This report

This document will report the current status of the Bustuc application as well as explaining how TUC and Bustuc works. Improvements to TUC and future work on the Bustuc project will also be discussed.

The next chapter gives an introduction to natural language processing and a review of TUC. Chapter 3 will discuss the requirements of the Bustuc applications and present an overview of the Bustuc system. The fourth chapter explains the Bustuc system in detail, and the fifth chapter discusses future work on Bustuc. The last chapter reports the status on the Bustuc application and gives a conclusion.

1. That is not a typo. Apparently the number of requests increases rapidly with the complexity in the route net.

CHAPTER 2

Natural language processing and TUC

This chapter gives an introduction to natural language processing and explains the TUC system. The chapter concludes by discussing current limitations of TUC.

2.1 Natural language processing

Natural language processing (NLP) is the use of computers to understand human natural languages, where “understand” means the ability to recognize and use information expressed in the language (Covington 94).

The structure of a human language is usually divided into five levels:

- Phonology - sounds of the language.
- Morphology - the formation of words.

  Word forms can be classified into three categories:

  Inflection - various forms of a single word, such as “run” and “runs”.

  Derivation - various derivations of the same word, usually in different categories, like “smart” and “smartness”.

  Compounding - new words composed of other word, for example

  “dogcatcher”.
- Syntax - sentence structure.
- Semantics - the literal meaning of a sentences.
- Pragmatics - the use and interpretation of language in context.

In this report, the semantic meaning of a sentence will be equivalent to the TQL expression that TUC produces from a sentence. Thus we can say that the morphological, syntactic and semantic level of analysis are handled by TUC, while the Bustuc application handles the pragmatic level of analysis.

Natural language consist of extremely many words which each have many meanings and which combine in generally undecomposeable ways to form sentences. This makes natural language processing hard, but the main reason for the hardness of the task is that natural language is not a representation language but a communication language - correct interpretation of a natural language utterance is only possible if the receiver has a great amount of knowledge of the world, the concrete situation in question and the human mind.

Still, practical applications of NLP are emerging in at least the following areas:

- Command language systems using a natural language for making commands which some receiving system will execute. The receiving system can be for instance a computer operating system or a car.

  TUC has been used for an application of this type called Unix Help - a natural language system translating english sentences to Unix shell commands (Wøien 95).
- Database query systems that translate natural language queries into formal database queries.
- Natural language expert systems, which communicate with the system in natural language and which are able to reason over their own data.

  Bustuc belongs to this category.
- Automatic translation between natural languages.
- Knowledge extraction from natural language text.

Examples of prototypes or commercial systems exists in all these categories.

The success of a natural language processing system lies in limiting the domain of the system, and by hiding the system's limited general knowledge of the world and the human mind by hardcoding the relevant knowledge directly into the system's behavior.

2.2 Overview and history of TUC

The Understanding Computer (TUC) is a general, adaptable natural language processing system. It is general because it is meant to be useful for all kinds of natural language applications, and adaptable because it separates the semantic data base from the rest of the system, so that semantics for a new domain can be built without looking inside TUC.

TUC tries to achieve generality and adaptability by translating the input natural language sentence to a quasi-logic formula which is meant to represent all the information in the input sentence independent of context (or pragmatics).

TUC is a successor of the HSQL (Help system for SQL) project (Amble 89), which was a Scandinavian research project aiming at providing both natural language and graphical access to databases with SQL as an intermediate language.

The natural language part of HSQL was built upon CHAT-80 - a natural language system created at the university of Edinburgh by Warren and Pereira.

After the HSQL project, an internal research project called TUC was initiated at IDT to carry over the results from the HSQL project. Since CHAT-80 had some inherent weaknesses, it was decided to build a new system from scratch - the TUC system (Amble 94).

In 1995, a Unix Help system, using natural language as an interface to Unix, was made at IDT with TUC as the basis for implementation (Wøien 1995).

2.3 Languages supported by TUC

TUC currently exists in two versions - one for norwegian language and one for english. The norwegian version was built on the english one by using english as an internal language, and by translating from norwegian to english word by word during the morphological analysis. The norwegian version also uses a grammar that has some differences from the english one, but the differences between the grammars of the two languages are not too many.

A small problem with this approach is of course that some words have a different set of meanings in norwegian than english, especially prepositions. I.e., norwegian “av” sometimes means “from” and sometimes “of”. Because of this, “buss til sentrum av byen” (“bus to the centre of town”) can end up being understood by TUC as “bus to the centre from town”, something completely different. We solve this problem by using several internal versions of these english words (like from1, from2), one for each kind of meaning, so that we can translate the norwegian words directly to the correct version of the internal english word.

2.4 How TUC works

TUC translates a natural language sentence (a string of characters) to a logical expression in a language called Tuc Query Logic (TQL). The TQL expression represents the semantic meaning of the input sentence, as understood by TUC. TUC can also do some simple reasoning over the TQL expression to output answers to some questions.

TUC analyses sentences in a 5-step process as show in the figure on the next page.

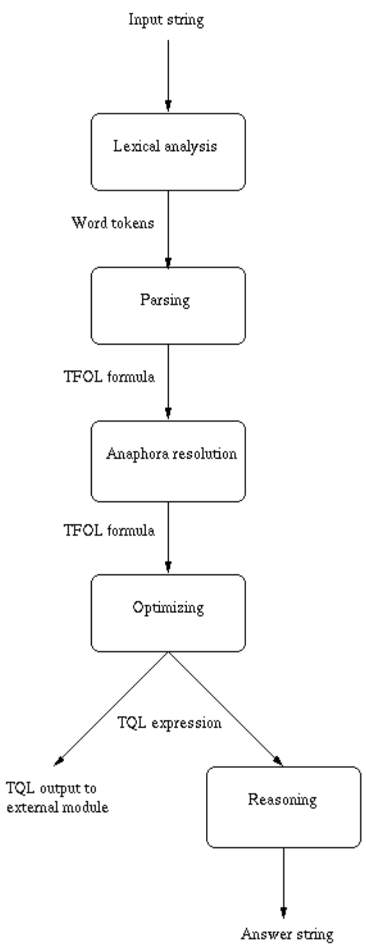

FIGURE 1 The architecture of TUC

- **Lexical analysis**

  The lexical analyzer looks up each word in the input sentence in it's dictionary. The dictionary consists of words kept in a separate dictionary file and the words in the semantic knowledge base. If a word is not found in the dictionary, the lexical analyzer tries to find it in a case specific data base that contains words mentioned in earlier sentences. The lexical analyzer also performs spelling correction of names by checking if the input word is close to a name in the dictionary, and if so using that name instead of the unrecognized word.

  If all characters in the input string are mapped to a word in this process, the set of words are output as tokens in their inflective root forms (i.e. "stopped" will be output as "stop") together with their possible word classes.

- **Syntactic and semantic analysis**

The list of tokens produced by the lexical analyzer is then parsed using a differential attribute grammar. The parser builds a TUC first order logic (TFOL) formula which represents the semantics of the sentence by using the semantic knowledge base. The parser uses a greedy heuristic by trying the longest possible phrases of the input sentence in decreasing order of probability. It will output the first TFOL representation of the sentence it finds that is syntactically and semantically satisfying.

The parser is not robust - if not all of the sentence is understood, the parser fails.

- **Anaphora resolution**

  Anaphors are words such as *he, it, did* and *then* that refers to objects or situations that have been mentioned or by implied by some other part of a discourse. In this phase such words are replaced with the internal object they represent.

- **Optimizing**

  The optimizing phase simplifies the TFOL formula into a TQL formula by skolemizing and removing unnecessary constituents from the TFOL formula.

- **Reasoning**

  TUC can do two types of reasoning over the TQL expression.

  It can do some common-sense reasoning to answer simple questions of the kind that can be answered by listing one or more individuals in the system, a count of individuals, or a yes or no.

  It can also do theorem proving with the Inger theorem prover (Amble 95).

In addition to being used by TUC's reasoning module, the TQL expression produced by TUC from an input sentence can be used by an external reasoning module, as is done in the Bustuc application.

2.5 The TQL language

TQL is an acronym for Tuc Query Language. TUC produces a TQL expression for every natural language sentence it recognizes. The TQL expression is used to express the semantics of the input sentence to an external application like Bustuc. Because of this important role, the TQL language will here be explained in more detail.

A TQL expression is a skolemized, simplified situation calculus formula.

- Skolemization is the process of removing all existential quantifiers in a formula.
- The simplifying process removes all atomic formulas in the expression which are always true or which are implied by other conditions.
- Situation calculus is a form of logic where the atomic formulas belong to a situation represented by a situation variable added as a last argument to the atomic formula. Situation variables provides a mean to bind simple formulas together to describe more complicated aspects of a situation than can be described by a single atomic formula. A situation in situation calculus can be both a certain time interval and place, and an abstract situation representing for instance a belief a person have which might not be true.

Example:

When does bus 5 leave today?
is represented in TQL as

```
which(A)::(5 isa bus,A isa time,leave/5/B,
           event/real/B,srel/intime/time/A/B,
           srel/today/time/nil/B)
```

This TQL expression can be paraphrased as follows:
Which A's are such that

5 is a bus
A is a time
5 leaves in situation B
situation B is an event in the real world
situation B is in time A
situation B is today

The single situation here (B) binds the individual statements together so that they together represent all important information in the input natural language sentence.

A TQL sentence consists of a *marker* which determines the sentence type, and a *body* that expresses the actual meaning of the sentence. The marker classifies the sentence into one following classes:

- A "which" question asking for individuals satisfying the body.
- A "test" question asking for the truth of the body
- A "how many" question asking for the number of individuals satisfying the body.
- An "explain" question asking for an explanation of the contents of the body.
- A "do" command, telling TUC to execute the body.
- A "new" sentence, representing new information that should be stored in TUC's case specific data base.

The body of the TQL sentence consist of a number of atomic (and in a few cases compounded) expressions of which the following are the most important:

- Individual isa Class
  Says that Individual is of class Class.
- Verb/Name/Situation
  Says that Name does Verb in Situation.
  This represents an intransitive verb phrase.
- Verb/Agent/Patient/Situation
  Says that Agent does Verb to Patient in Situation.
  This represents a transitive verb phrase.
- srel/Modifier/Class/Individual/Situation
  Says that Individual of Class is modified by Modifier in Situation.
  This represents a verb modifier phrase.
- nrel/Modifier/Class1/Class2/Individual1/Individual2
  Says that Individual1 of Class1 is modified by Modifier and Individual2 of Class2.
  This represents a noun modifier phrase.
- adj/Adjective/Individual/_
  Says that Individual has the property Adjective.

2.5.1 The role of TQL

TQL is meant to be a sufficient interface between TUC and an application. TQL must therefore be independent of context, because TUC shall be usable for different applications without changing the TQL representation, and since TUC has no knowledge of the pragmatics and context of a certain application area.

Of course, making TUC independent of pragmatics can be achieved both by knowing everything about pragmatics (and thus always choose the right pragmatic interpretation) or by knowing nothing about pragmatics and making no pragmatic judgements. In TUC the latter approach is used for obvious reasons.

To make TQL context independent, the semantics of a natural language sentence as represented in a TQL expression must contain all (or at least all likely) possible interpretations, so that the right interpretation of the sentence can be left to the application.

To achieve this goal the TQL must be limited to expressing the *structure* of the given input sentence, more precisely the type and literal content of each phrase, the type of words and the interdependecies of the phrases.

Since the pragmatic meaning of the input sentence must be determined by the application, an application using TUC will still be dealing with natural language, but TUC does the job of filtering out irrelevant details of the input sentences and structuring the sentence.

2.5.2 Separation of semantics and pragmatics

The approach of separating the structural semantic analysis completely from the pragmatics has the obvious advantage of making it possible to create new natural language applications with TUC without working directly with natural language and writing morphological analyzers, grammars and parsers again for each application.

The problem with the approach is that it requires a difficult trade-off between the need to have TUC filter away as much irrelevant details of natural language as possible on the one side and the need to keep all possible interpretations for any domain on the other side.

If TQL represents enough detail of the input sentence (for instance, by keeping past tenses) all interpretations of the sentence is clearly kept, but suppressing detail of the language is what makes TUC usable in the first place.

It is not clear if this trade-off can be tuned so that TUC can satisfy the needs of any practical application, but the advantage of having a natural language system that is application-independent is great enough to take this approach anyway.

2.6 Semantic knowledge in TUC

TUC has a semantic knowledge base that is used during the parsing to build up the TQL expression representing a sentence. When using TUC for a new domain the semantic knowledge base is ideally the only thing that will have to be changed in TUC.

The main constructs of the semantic knowledge base are:

- **A tree of all nouns**

  Every noun is defined to be a kind of ("ako") exactly one other noun (except "thing" which is the root of the tree).

  For instance, the declarations from "bus" to the root are

  ```
  bus             ako    vehicle.
  vehicle         ako    object.
  object          ako    thing.
  ```

  If there is a declaration in the semantic knowledge base for a noun, then this declaration is also valid for all subtypes (ako's) of this noun and all instances (isa's) of the noun. For example, all buses have a speed, since it is declared that all vehicles have a speed.

- **Nouns that "have" noun**

  For instance, the declarations for a bus is

  ```
  bus             has_a  number.
  bus             has_a  departure.
  bus             has_a  arrival.
  bus             has_a  endstation.
  ```

- **Intransitive verb phrases**

  These definitions can produce Verb/Name/Situation formulas in TQL.

  For instance, that a vehicle can pass is defined by
  `iv_templ(pass,vehicle).`

- **Transitive verb phrases**

  These definitions Verb/Agent/Patient/Situation formulas in TQL.

  For instance, that a vehicle can leave is defined by
  `tv_templ(leave,vehicle,place).`

- **Verb complements**

  These definitions are used in *srel* formulas in TQL.

  For instance, that a vehicle can "pass by a place" is defined as
  `v_compl(pass,vehicle,by,place).`

- **Noun complements**

  These definitions are used in *nrel* formulas in TQL.

  For instance, that a bus can be "to byen" as in "The bus to byen passes nardo." is defined by
  `n_compl(to,bus,place).`

- **Adjectives**

  These definitions can produce *adj* formulas in TQL.

  For instance, that any thing can be fast is defined by
  `adj_templ(fast,thing).`

There are also several predicates that deals with less common language constructs which will not be reviewed in detail.

The semantic knowledge base contains knowledge about what it is possible (or likely) to say, regardless of the truth of the sentence. For instance, if we want the sentence “Can Tuc think?” to be parsed by Tuc, we must declare that “programs think” is a legal intransitive verb phrase, even if we think that this is false.

2.7 Current limitations of TUC

The current version of TUC has several important limitations, of which some are of importance for this project and some will be important in the future of TUC.

The most important limitations are discussed here.

2.7.1 TUC’s parser

TUC only produces a TQL expression from a sentence if it recognizes all (spell- corrected) words in the sentence and finds a semantically valid parse of all of the sentence. That is, the parsing is strict.

The historical reasons for this are:

- TUC was inspired from work with the HSQL system, where the target users would use the system enough to get to know the subset of natural language that the system recognized.
- The view taken when initiating the TUC project was that in order to focus on the problems of understanding, communication and reasoning, the linguistics had to be kept as simple as possible.
- Writing a fairly good strict parser is simpler than writing a robust one.

The language TUC should understand was seen as natural readable logic (NRL), a language with a simple and well-defined syntax and semantics, but readable as english.

While this was a useful approach in TUC’s early stages, the situation is now different:

- Projects like Bustuc have target users that are casual and (possibly) computer ignorant, they can not be expected to know or guess any particular subset of natural language.
- The TABOR project will ultimately end up in a system that answers queries over telephone using TUC. This requires a parser that can handle incomplete and erroneous sentences since speech recognition can never be made perfect, and since incomplete and repaired sentences are very common in speech.
- The TABOR system will also handle dialogue. In a dialogue, sentences are often elliptic or otherwise incomplete.
- TUC has reached a level of sophistication where an improved parser is the single feature that will contribute most to improving the performance of the system, as will be shown in chapter 6.

A good robust parser should not be restricted to only giving output for sentences that do not have a complete parse, it should also try to guess missing but likely words and phrases and repair words not fitting with the rest of the sentence. The input should be treated more like evidence to what the user might want to say than as a complete and correct representation of some query.

A practical domain-independent robust parser is still beyond the state of the art, but the task can best be addressed with a probabilistic parser. Robust probabilistic parsers are starting to emerge, but while unification-based grammars dominates in unprobabilistic parsing, there is still no agreement in the statistical parsing community as to which grammar formalism to use, see Eisner 1996. The choice of a usable probabilistic parsing algoritm and grammar is a large project of it's own and will not be elaborated upon further here.

If it is decided that changing to a probabilistic parser is too complicated, substantial improvements can also be made by using an unprobabilistic but robust parser. For instance, the TRAINS project (Allen 95) uses a straightforward bottom-up chart parser, and accomplish robustness by using *monitors* - routines that are invoked when certain constituents are added to the chart to extract key information about the constituents. Since chart parsers are so accepted and standard, this could be a good approach also for TUC. Since everything that a chart parser recognizes is added to the chart at some point, it should also be possible to achieve acceptable robustness by doing an intelligent choice of monitors.

2.7.2 TUC's semantic knowledge base

TUC's knowledge base arranges nouns in a strict tree where each node is a noun that is "a kind of" it's parent in the tree. Declarations for a noun always hold also for the successors of the noun in the tree. This representation is simplistic enough to yield results quickly, and since the semantic knowledge base is meant to be changed for a new application, words with different usage in different contexts can be handled by hardcoding the implifications of the known context of the application into the semantic base.

There are still some reasons for changing this scheme with a more flexible one at some point in the future:

- If context is hardcoded into the knowledge base for a new application, the knowledge base can not be expected to accumulate more language knowledge over the time, since old definitions must be deleted to cope with the new context. In a system where many meanings of the same words can live together in the knowledge base, porting to a new domain will always be a process of *adding* to the knowledge base, which will cause TUC's knowledge of language to increase monotonously for each new application it is used for.
- Some applications will require different uses of the same word. For instance a physics teaching system will need to handle many uses of words like "time".
- If TUC shall be able to do more reasoning over and preprocessing of sentences on its own, it seems unlikely that the current scheme will be powerful enough. It is clear that much knowledge about uses of words are unrepresentable when using single inheritance, since almost any words can be said to be "a kind of" many other words, not just one, and rules for word usage are not applicable in all cases.

- TUC can not use it's current semantic knowledge to understand metaphors, since metaphors use phrases in place of other phrases, which TUC will report as meaningless unless all possible metaphor usage are coded into the knowledge base.

  ```
  E: his wife is a rose.
  --- Meaningless at *
  his wife is a rose * .
  ```

  Since a wife is a woman, not a flower.
- There are always exceptions to semantic rules. There is no way to code exceptions into the current model.
- From a more pragmatic viewpoint, it is useful to be able to say that a noun has the properties of multiple other nouns, not just the ones that are above it on the same leaf in the tree of nouns. That would make maintaining and refining the semantic knowledge base a simpler task, since there would be less duplicated declarations.

A semantic knowledge base scheme completely solving all the problems above is beyond the current state of the art. I will anyway suggest some features a semantic knowledge base with such capabilities should have.

- The semantic knowledge base should be probabilistic rather than binary. The most probable semantic interpretation of a sentence given the circumstances should be selected, instead of the first parse. Also, if no probable parse is found the semantic base should be able to suggest repairs to the input sentence that will produce a probable parse.
- New applications should never need to erase correct definitions from another application.
- Semantic knowledge should be kept in a unrestricted semantic net, where each usage of each word is a node, and where the edges between nodes encodes direct relations between the words. The links should have a type attribute which determines the relation between the words. the type can be “a kind of”, “is a”, “uses”, “is a part of” etc., but ideally the possible types of nodes should be restricted only in that their definition should be contained as semantic knowledge in the base. (Of course this will not hold for some initial types for bootstrapping.) Links should have a number describing their probability (or strongness).
- The knowledge base should have a representation of context so that context can both be supplied dynamically from an application and updated by the natural language input as it is parsed.

  This context could be a set of nodes from the semantic net above, with a number attached to each context node to indicate the nodes current strength as a context indicator.

  For instance, in the Bustuc application, the context would give a high score to the node for a bus as a vehicle, a passenger, time as indicating moments on the day, time as indicating a measure of length and so on.
- The nodes that has a nonzero score in the current context should activate their neighbors in the semantic net proportional to the strength of the link and the strength of the node in the current context. Activation should propagate in the network until the strength reaches below a predefined level.

  This is essentially a numerical extension to traditional (Quillian 67) spreading activation.

- The current activation level of the node should influent the total probability calculated for a specific parse.
- Metaphor comprehension and elaboration should be built into the system. In recent years metaphor has been recognized as a deep cognitive phenomenon (Way 91, Veale 94). The semantic network sketched above seems to fit well with networks suggested as adaptive models for metaphor like the Sapper network (Veale 95). The Sapper framework views metaphor comprehension as building bridges in a semantic network which subsequently alter the strengths of edges between nodes. A Sapper network is not only capable of interpreting metaphors, but it can also elaborate upon them over time to create and understand new, related metaphors.

Building a system with all this characteristics is a major task, but the reward is a system that will be much more powerful and flexible than the current one. Since the semantic knowledge base of TUC already is a semantic network (but with a very stringent structure), changes can be done gradually, and all probabilities can be set to one or to one of a few values until a system that find values automatically from a text corpus can be employed to provide better values.

2.7.3 Meaningless sentences

Not all sentences that contain meaningless parts are meaningless as a whole. Consider

Is it meaningless to say that Norway jumps quietly?

The correct answer here is “yes.”. But TUC will report the sentence as meaningless unless that a country can jump etc. is included in the knowledge base (which is something we do not want to do).

Also, many questions are reported meaningless when the correct answer is “no”. An example is

Are buses alive?

An improvement would have been to be less strict with semantics for questions than for statements. For a question, TUC should try to find any semantically invalid but otherwise satisfying parse when it is not possible to find a semantically valid parse.

2.7.4 Morphology in Norwegian

In contrast to English, Norwegian allows unrestricted compounding of words. For instance, bus departure is “bussavgang” in Norwegian. Because of this, it is inconvenient to list all compounded words that is even likely to appear in a domain.

To solve this problem, TUC’s morphological analysis should be extended to handle word formation. The root meaning of the word, which is almost always the last part of the word should be extracted, and the other parts of the word should be taken as adjectives to the root word. In this way, “busdeparture” would be represented by

```
A isa departure,adj/bus/A/B
```

similar to how the equivalent phrase would be represented in English.

CHAPTER 3 Bustuc - An overview

This chapter will present an overview over the Bustuc system and the problem domain it addresses. The chapter concludes with a brief review of some related work.

3.1 The problem domain

The Bustuc application answers questions about bus departures in Trondheim stated as complete natural language sentences in norwegian or english.

The types of questions that can be answered are restricted in two ways:

- TUC limits the types of questions that can be answered by refusing to parse formulations that does not fit with it's grammar and semantics.
- Bustuc limits the types of questions by having logic and data to handle only a limited set of concepts.

The limitations that TUC imposes are that sentences must be complete[1] and that the sentence must be syntactically and semantically correct according to TUC's grammar and semantics.

The types of questions that Bustuc can answer is somewhat harder to define. Since the goal has been to make a useful application, the focus has been on answering the types of questions that are actually asked by real users instead of making some logical definition of the type of questions that should be answered.

1. There is one exception to the completeness requirement. Because of their frequent use, single noun phrases are parsed as if they were preceded by "what is".

The information about which questions are actually asked comes from two sources:

- During the summer of 1996, 217 real telephone dialogues between operators ~~informers~~ at the public traffic information centre in Trondheim (Trafikanten) and customers was recorded and transcribed. The transcriptions can be found in Gudmestad 96.
- A web page with an interface to a recent version of Bustuc has been available since the spring of 1996. Nearly 5000 questions from real users has been logged. (See appendix A and B for examples.)

Not surprisingly the real questions showed that most user simply wanted to know how or when they could get somewhere by bus. The questions produced to retrieve this information mostly involved combinations of the following concepts:

- Buses.

  Which buses goes from Nardo to Byen?

  What bus do I take from Lade?
- Bus departures

  When does bus 5 leave from Byen?

  How often does 66 pass Jakobsli?
- Times

  ...between 1500 and 1700 on friday?

  ...around 18 in the evening?

  ...in the next fifty minutes?
- Orders and sets

  ...the five next buses...

  ...the second last bus...
- Stations

  Where can I change from bus 5 to bus 66?

  What is the endstation for bus 5?
- Prices

  how much does it cost to go from nardo to blakli?

The transcribed spoken discourses showed a great variety and flexibility of language - incomplete, ungrammatical, elliptic, sentences with interruptions and parallel threads. Since this project is limited to dealing with complete sentences and no dialogue, we will leave that fact here after noting that a system that can communicate with casual users without requiring the user to adapt to the system needs much more research, even for a domain as simple as this. On the other hand, the operators answering the telephone call did not have any computer help at all, and was not able to find optimal solutions to the users questions. They also forgot elements of the query, misunderstood the user unnecessarily and delayed, so the performance a spoken dialogue system in this domain must compete with is far from perfect.

In summary, the domain and the questions from the logs has the following interesting characteristics:

- The domain is small and the ~~are~~ problems simple.

  Omitting the communication bit, the problems of the user are fairly easy to solve.
- The language in the queries is rich and informal.

  The users are casual, and the domain is so common that the rich variety of synonyms, matters of speech, metonyms and abbreviation that characterizes daily speech has been developed.
- Many users thinks of the domain as even simpler than it actually is. This gives rise to underspecified questions that are hard to handle without dialogue.

  Consider the question “How often does a bus leave from Lerkendal?”.

  This question is problematic by two reasons:

  - The user obviously think that buses leaves with some fixed interval that can be communicated simply and be of value for the user. In reality buses passes many stations at almost any interval.

  - The user is most likely interested only in buses going in one direction, but which? The buses that passes Lerkendal is far from both ends of their route, so there is no way to deduce the direction the user is really interested in.
- Many users use imprecise expressions that must be artificially defined precisely before they are answered.

  For instance, what does “in the evening” or “around 1500” mean, precisely?

The problem we are faced has more to do with communication than with anything else, since the difficult part is to understand correctly all the different ways to formulate a question, not to solve the actual problems once understood.

We find that the system - given it’s limits of single, complete questions - should meet the following requirements:

- Most questions likely to appear about the domain must be understood by the system.
- The system must rephrase all it has understood about the question clearly in the answer to avoid misunderstandings and to express the choices in interpretation it has done when people uses underspecified or vague questions. This will also alert the user if the system has failed to understand the intended meaning of the question.
- The answers must be correct and optimal.
- When a question is not successfully parsed, the system must indicate the nature of the error - lexical, syntactic, semantic or factual.
- The system must inform the user when the question implies that the user has false presuppositions, like nonexisting buses or relevance of weather.
- When making answers, the system must only use language that is acceptable in return.
- The system must be able to make answer in different languages (English and Norwegian), without needing unnecessary multiple code for different languages.
- The system should, as far as possible, inform users about it’s limitations when questions outside the domain arrives.
- The response time must be acceptable. The system must not delay long enough to annoy the user and it must be faster to use this system than a conventional paper-based bus timetable.

Making a system meeting this requirements available on the web would be both a useful service and an interesting experiment.

3.2 Architecture of bustuc

The Bustuc system takes input from TUC in form of TQL expressions and produces a natural language answer in Norwegian or English from the TQL expression. Bustuc is, like TUC implemented in Prolog.

The process from TQL input to answer text is divided into four phases:

- First, the TQL expression is translated to a Buslog program that represents the question in a definite and unambiguous form. Thus, this phase makes up the pragmatic phase of natural language processing.
- Then, the problem defined in the Buslog program is solved. The Buslog predicates that makes up the Buslog program will reason over the bus route data when they are interpreted by Prolog, and instantiate their free variables to the answer.
- When the problem is solved, a natural language answer must be generated. In this phase an answer program (Busans) is produced from the instantiated Buslog program. The Busans program can output an answer text that represents the instantiated Buslog program.
- Last, the Busans program is executed by Prolog yielding an answer text in the active language (English or Norwegian).

The steps and the knowledge bases that are central to the various steps are shown in the figure on the next page.

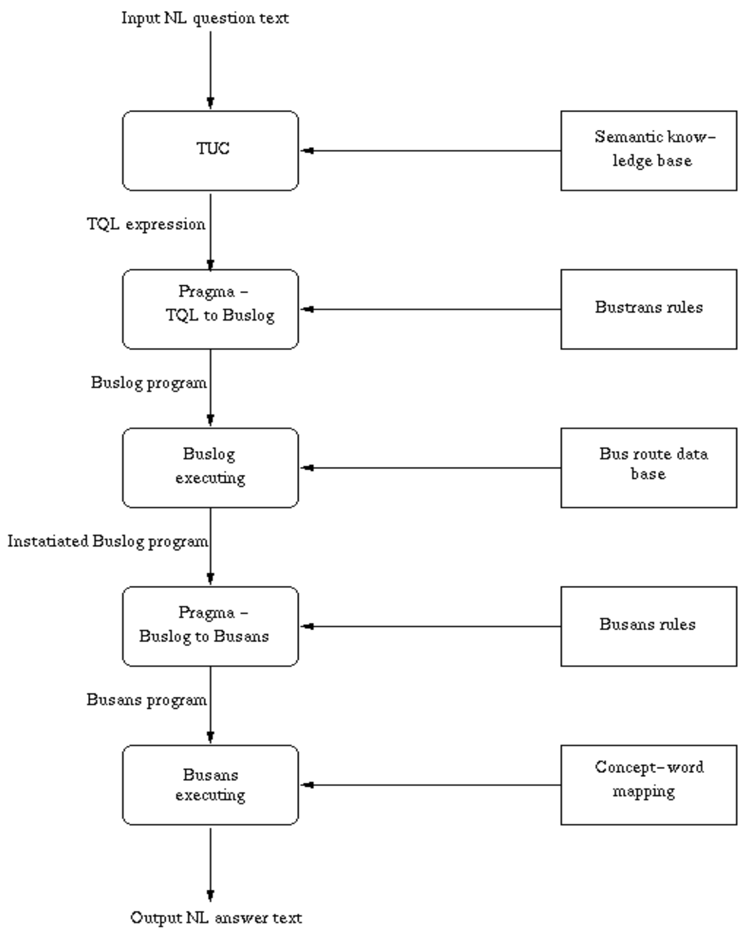

FIGURE 2 The architecture of the Bustuc application

Note that the answer is produced solely on the basis of the Buslog program. When making the answer one does not have access to the actual question as represented by TQL. This choice simplifies the process of generating answers since the number of solution types that needs an answer is much smaller that the number of actual problem formulations represented in TQL. This also makes it easier to ensure that answers are clear in all situations.

To do the two translations, from TQL to Buslog and from Buslog to Busans, a powerful production system called Pragma developed specially for the project is used. A Pragma production rule can condition on and produce changes in both the source and the destination, and it can also declare arbitrary Prolog code that must succeed for the rule to trigger. This gives a flexible but very high-level system where one can have rules both for dealing with the destination only for adding things to the destination based on the source, or a combination of this.

The knowledge bases that is shown in the figure above are explained below.

The semantic knowledge base describes which phrases that are semantically valid, as described in chapter 2. The semantic declarations for Bustuc is concerned mainly with which verbs and complements that fits with buses, and how one can express movement of people. There are an estimated 500 declarations concerning the Bustuc domain.

The Bustrans rule base is the pragmatic knowledge base of Bustuc. The rule base consists of Pragma rules that mainly describes which predicates that shall be added to or altered in the Buslog program given which TQL sub-expressions. Many rules are of course needed for each atomic pragmatic meaning, since different formulations with the same pragmatic meaning is represented with different TQL code. There are about 260 Bustrans rules.

The bus route data base is a relational data base implemented in Prolog that consists of two main parts. One part is derived automatically from Trafikanten's own data base. This part contains all bus routes and all departures for each bus route. The other part of the data base is entered manually and contains information about which stations that lies in short walking distance from each other, and relations between places and stations. The automatically generated bus route consists of about 37600 declarations, and the manually entered part of about 400.

The Busans rules are Pragma rules that describes the subphrases of answers that must be produced from different parts of the Buslog program, and how phrases must be modified to fit together in nice sentences. Some of the rules are language-dependent to accommodate differences in formulations for different languages. There are about 70 Busans rules.

The concept-to-word mappings maps from an internal representation of words, wordforms or phrases that are produced in the Busans program to actual words in the active language. The mapping from the internal representation to words in the active language thus happens at the latest possible stage.

3.3 An example

To give an idea of how the system works in practise, an example query is presented from input text to answer text. For clarity, many details has been removed in this example.

Input natural language text:

when does buses leave from nardo after 1600?

This sentence is translated to the following TQL expression by TUC:

```
which(A)::(1600 isa time,nardo isa neighbourhood,
           B isa bus,A isa time,
```

```
leave/B/C,event/real/C,srel/intime/time/A/C,
srel/from/place/nardo/C,
srel/after/time/1600/C)
```

This expression is then used to produce a Buslog program by applying the Bustrans rules:

One Bustrans rule recognizes
`B isa bus,leave/B/C,srel/from/place/nardo/C`
which says that a bus leave from nardo in situation C.
The rule adds a *departure* predicate to the Buslog program that collects all bus departures from Nardo into a list in one of it's arguments, and a *passevent* predicate that represents the fact that the list of departures is in situation C (independent of whether the departures was produced by a *departure* predicate or not).
We now have this Buslog program:

```
departure(Place,Departures),
passevent(Departures,C)
```

Another Bustrans rule recognizes
`srel/after/time/1600/C`
and the facts that a *passevent* predicate for the situation C is already present in the Buslog program.
The TQL fragment means that the events happening in C happens after 1600, and since we have a list of departures that happens in situation C we must constrain this list to be after 1600.
The rule adds a filtering predicate that filters out all departures from the list in *passevent* that happens before 1600.
Now we have this Buslog program:

```
departure(Place,Departures),
keepafter(1600,Departures,NewDepartures),
passevent(NewDepartures,C)
```

After this, no more translation rules triggers, so the resulting Buslog program is interpreted to instantiates it's free variables to lists of departures compiled from the data in the bus route data base.

In the third phase the Busans rules are employed to produce an answer program from the instantiated Buslog program.

A Busans rule recognizes the *passevent* predicate and adds code that can the first, next (if applicable) and last departure in the list (when the list has more than 10 elements, the whole list is not written out).

Another Busans rule recognizes that the departures in *passevent* are limited to be after 1600 by *keepafter* and adds code to output "after 1600" in the right place in the output.

Last, a rule that adds spaces between each word triggers.

We now have a Busans program that can output an answer text. When this program is run, it finds concept-to word mappings, that maps the internal representation of the busans program to words in the current language, which is english.

An example of a concept-to-word mapping that is employed here is
`cwc(thenext,['the next','neste']).`

which maps the concept *thenext* to an english or norwegian phrase.

The Busans program produces this output (when run at 19:33). Notice the “after 1600” added by the second busans rule trigged.

The first bus after 1600, number 52a passes by Nardosenteret at 1602.
The next bus after now, number 9 passes by Nardosenteret at 1934.
The last bus, number 5c passes by Nardosenteret at 2422.

3.4 Related work

In this section will name some systems and projects that addresses problems related to those of Bustuc.

3.4.1 The Philips Research System for Continuous-Speech Recognition

The Philips Research System for Continuous-Speech Recognition is a commercial system that can answer questions about train departures in Germany on german over telephone. This system has little flexibility in the input it accepts and the dialogue is completely system-controlled. It starts by asking where the person wants to go *from*, then where the user wants to go *to* and last the *date* the user wants to travel. The only deviation allowed from this is supplying both startpoint and destination in one sentence.

3.4.2 The circuit fix-it shop

Smith and Hipp describes a dialogue system that can assist in finding errors on electric circuits by performing a spoken dialogue in English with the user (Smith 94). The focus in this system is mainly on dialogue modeling with the variable initiative dialogue as a central theme. The system views a dialogue as analog to proving a theorem, where a missing axiom in the proof corresponds to a missing piece of information the user must be queried for in the dialogue.

3.4.3 Pegasus

Pegasus is a project at MIT which provides a spoken natural language interface to flight information and reservation. The system builds on a existing menu-driven system and conducts dialogue with the user to clarify the needs of the user.

3.4.4 TRAINS

The TRAINS project is an attempt to build a system that can interact and collaborate with humans in problem solving (Allen 95) by communicating via speech and graphics. The current prototype, ToyTRAINS is a system that solves route planning problems in a dialogue with the user. The project addresses many of the aspects that will also be of importance when making a dialogue bus route transport system, namely robust behavior when speech understanding is poor, effective acknowledgment strategies, accumulating context, and clarification and correcting subdialogues. The route planning capabilities of the system is deliberately weak to encourage interaction with the system.

CHAPTER 4 Bustuc in detail

In this chapter the various components of the Bustuc application will be explained in some detail.

4.1 Declarations in the semantic knowledge base

The declarations in the knowledge base that was added for the Bustuc application are mainly about which verbs and compliments that can combine with bus (or any vehicle), how one can express moving of people and how one can express time and place compliments.

Bus is declared as

```
bus             ako    vehicle.
    bus             has_a  number.
    bus             has_a  departure.
    bus             has_a  arrival.
    bus             has_a  station.
vehicle         ako    object.
    vehicle         has_a  frequency.
    vehicle         has_a  speed.
    vehicle         has_a  driver.
```

The transitive and intransitive verbs that are declared together with buses or vehicles are *go*, *run*, *stop*, *reach*, *serve*, *take*, *be*, *arrive*, *come*, *depart*, *get*, *leave*, *meet*, *pass*, *let* and *use* (the transitive verbs with appropriate patients, of course).

The verbs declared for persons that have to do with the domain are the same as those for buses and in addition *wait* and *travel*.

Agents, which both people and TUC are subtypes of can be combined with many verbs, but those most relevant for this domain is *ask*, *tell*, *do*, *know*, *write*, *listen*, *help*, *understand* and *think*.

Together with numerous complements, this seem to be a nearly sufficient set of actions for this domain, since there seldom arrives new questions to the web application that uses unknown word, as shown in chapter 6.

To illustrate the number of compliments needed, we will list all the compliment declarations needed for one single verb phrase; intransitive *go* with *bus* as patient.

```
iv_templ(go,bus).
    v_compl(go,bus,from,place).
    v_compl(go,bus,between,time).
    v_compl(go,bus,between,place).
    v_compl(go,bus,to,place).
    v_compl(go,bus,by,place).
    v_compl(go,bus,through,place).
    v_compl(go,bus,in,direction).
    v_compl(go,bus,nil,direction).
    v_compl(go,bus,towards,place).
    v_compl(go,bus,for,place).
    v_compl(go,bus,past,place).
    v_compl(go,bus,into,place).
v_compl(go,bus,out_of,place).
    v_compl(go,bus,with,frequency).
    v_compl(go,bus,with,speed).
    v_compl(go,bus,I,J):- stanprep(I,J).
```

where *nil* means *no preposition*, and *stanrep* is true for a list of about 25 of the most common combinations of a preposition and time or place. In all, there are defined about 40 compliments for this phrase alone.

Places and stations are defined as:

```
place          ako    thing.
       place          has_a   name.
station        ako    place.
endstation     ako    station.
```

In addition, there are noun compliments to deal with modifications of buses and places, adverbs for orders, adjectives for weather, speed, color, definitions of days and times of the day etc.

In all there are nearly 2500 declarations in the semantic knowledge base, of which an estimated 500 are for the bus domain.

4.2 The bus route data base

The bus route data base contains the data about buses, bus departures, places and stations. The data base is a relational data base in 3. normal form implemented in Prolog.

The data on bus departures, bus names and stations are generated automatically from the data base of Trafikanten, while the data on places and correspondence between stations are entered manually.

Busdat consist of the relations listed below.

4.2.1 route(Key,BusNumber,BusName,BusNote)

Says that there exists a bus with key Key, number BusNumber, name BusName and note BusNote. The name is usually either equal to the busnumber or it consist of the number plus a letter. The buses in Trondheim are identified by a number and a letter, but different routes can have the same number and letter. In addition, a note are sometimes associated with a route.

Example:

```
route(bus_5_8,5,'5c',0).
```

4.2.2 departure(Key,Time,Day)

Says that the bus with Key leaves from its (unique) start station at Time at Day.

Example:

```
departureday(bus_4_6,2020,sunday).
```

4.2.3 passes(Key,Station,Delay)

Says that the bus with Key leaves from Station Delay minutes after the departure.

Example:

```
passes(bus_4_6,torvet,30).
```

4.2.4 station(Station)

Says that Station is a station.

4.2.5 isat(Station,Place)

Says that Place is at Station. This relation maps the places users mention to stations. The same place can map to many stations as long as it only maps to one station at a single bus route. In this way, Bustuc can find the correct station from the context of a given route.

For instance does the two declarations

```
isat(universitetet_lade,ntnu).
isat(universitetet_dragvoll,ntnu).
```

enable this:

When does the next 4 leave from NTNU?

The next bus, 4a passes by Universitetet Lade at 2224.
When does the next 66 leave from NTNU?
The next bus, 66 passes by Universitetet Dragvoll at 2255.

4.2.6 corresponds(Station1,Station2)

Says that Station1 and Station2 are so close to each other that it is advisable to change between buses that passes any of the stations.

This relation is reflexive and symmetric.

4.2.7 foreign(Place)

Says that Place is not in the reach of the buses of Trondheim Trafikkselskap. This enables Bustuc to inform the user that the place she is asking for is not reachable with TT-buses instead of simply answering with that place is unknown.

4.3 Domain reasoning - Buslog

Buslog, or Bus logic, is the system that reasons over the data in the bus route data base and comes up with solutions that can combined and used to generate answers to the users.

Buslog compiles and filters lists of departures or stations, finds departure frequencies, the best route between two places and reasons with days and times.

Each of these main groups of predicates will here be explained.

4.3.1 Lists of departures

The predicate *departure* is used to find a list of departures from a given place and day, and optionally for only a specified list of buses. All the departures are ordered after time of day.

In addition there are many predicates that does filtering on these lists. It is possible to keep only buses *before*, *after*, *at* or *between* times, only buses with a specified number or only buses that comes from or goes to a specified place. This filters can be combined to fit almost any constraint the user might give on the type of departures from a station.

For example, all departures with bus 5 from nardo at monday in direction blakli between 1500 and 1700 is retrieved with:

```
departure(5,nardo,monday,Departures1),
keepto(blakli,Departures1,Departures2),
keepbetween(1500,1700,Departures2,Departures3)
```

(The resulting list of departures is Departures3.)

The departure lists can also be used by predicates that extracts all the bus numbers from them or the intervals between departures.

4.3.2 Lists of stations

The predicate *findstations* are used to find all stations that a bus passes. This list can be filtered to contain only all stations that lies between two places on the route, or only the stations that *corresponds* with the stations in another list.

This is used for instance to solve the query “where can I change from 5 to 66?”, which produces

```
findstations(5,Day,Stations5),
findstations(66,Day,Stations66),
corrstats(Stations5,Stations66,BothStations)
```

(The answer is the stationlist BothStations.)

There is also predicates to find the stations at a place (all in a *isat* relations including the place), and all stations near a place (all that is at the place or that is at stations that corresponds to each others).

4.3.3 Departure intervals

The intervals between bus departures in a bus departure list is found by the *frequency* predicate. This is used in a query like “How often does a bus leave lerkendalsveien in the evening?” which produces

```
departure(_,lerkendalsveien,Day,Departures1),
keepbetween(1700,2200,Departures1,departures2),
frequency(Departures2,Interval1,Interval2,_)
```

(“the evening” is understood as between 1700 and 2200).

Since there is often many different departure intervals from a place, it is not clear which ones to present for the user. This is solved by selecting two intervals that might will seem to be among the most important ones by a user. The two intervals selected are those that has the highest numerical significance as defined as

Significance = Interval + Number * 3

where Interval is the number of minutes between each departure of a single frequency and Number is the number of departures in the list that has this Interval. Also departures with Interval larger than 3 minutes are preferred.

This formula seems to represent humans perception of significance of frequencies adequate by balancing the weight on the time span the frequency lasts (higher Interval means larger time span) with the number of departures for the frequency. However, this informal approach is not acceptable unless the user is notified that this is not definite answers. The user is notified of this by using “normally” in the answers, as in

> A bus normally passes by Lerkendalsveien every 30 minutes or every 14 minutes between 1700 and 2000.

which is the answer to the query above.

4.3.4 The best route between two places

When the user wants to know how to get from one place to another, two departure lists must be coupled together so that the best paths between the two places can be found. This is done by the *coupled* predicate.

For instance does “How can i go from jakobsli to blakli?” produce the Buslog program

```
departure(_,jakobsli,Day,Departures)
departure(_,blakli,Day,Arrivals)
coupled(Departures,Arrivals,...)
```

coupled covers these cases:

- If there are departures in both lists that belongs to the same route, these departures will be retrieved. This means that when possible, the user will be presented with solutions that includes no bus changes.
- If there are no departures in both list that belongs to the same route, coupled will find the optimal solution with *one* bus transfer. The optimal solution can be two different things depending on the type of question:

  - If one of the departure lists has been filtered to only the departures before a time or if the user has asked explicitly for the last possible option, the optimal path is the one that has an as late departure as possible from the departure list.

  - Unless the condition above is true, the optimal path is the one with the earliest possible arrival time.

  If there are more than one path that is optimal with respect to one of the two criteria above, the one with the shortest travel time is selected.

When these three cases are covered, all the following questions can be handled correctly by *corresponds*:

> *when can I go from nardo to blakli the next time?*
> The next bus, 5e passes by Nardosenteret at 2005 and arrives at Blakli at 2014.
>
> w*hen can i go from nardo to jakobsli the next time?*
> 5 passes by Nardosenteret at 2025 and 66 passes by Idrettsplassen Jakobsli at 2100.
> You can change from 5 to 66 by leaving bus 5 at Lerchendal Gård at 2029 and entering bus 66 at Lerchendal Gård at 2047.
>
> *when can i go from nardo to jakobsli the last time?*
> 5 passes by Nardosenteret at 2325 and 66 passes by Idrettsplassen Jakobsli at 2400.
> You can change from 5 to 66 by leaving bus 5 at Lerchendal Gård at 2329 and entering bus 66 at Lerchendal Gård at 2347.

Note also that since the departure lists are found by *departure*, any of the departure filtering predicates can be used in conjunction with *coupled* as can the frequency predicate.

Since a bus passes a station in Trondheim about a half million times during a week, efficiency becomes an important issue in defining the part of *coupled* that finds the best route with one transfer. The algoritm works optimal (given that no more data is stored) by doing the following:

> If we are optimizing for late departure, choose the last existing departure first, and then try to find a matching arrival, starting with the arrival as soon after the departure as possible and working forward in time. If no matching arrival is found for this departure, the next to best departure is tried, and so on.
>
> If we are optimizing for early arrival, choose the first existing arrival first, and try to find a matching departure, starting with the departure as soon before the arrival as possible and working backwards in time. If no matching departure is found for this arrival, the next to best arrival is tried, and so on.

This algoritm will usually find a solution by trying to match only a few departures and arrivals, without missing any.

4.3.5 Days, times and prices

Buslog's time logic includes telling which days that exists, which days that are successors (to answer questions about tomorrow and yesterday etc.) finding the current day or time from the system clock and subtracting and adding minutes to times.

Buslog can also return the price of a trip between two places.

4.4 The Pragma production system

In this section, the Pragma production system that is used to produce a Buslog program from the TQL expression and a Busans program from the Buslog program is explained.

4.4.1 Introduction to Pragma

A production system is an automated reasoning system that use implications as it's primary representation, and which interprets the consequent of each implication as an action recommendation (Russel 95).

Pragma is a production system that is used to translate from a *source* to a *destination*, where the source and destinations are conjunctions of terms. Pragma matches rules from the top and down in the rules base, so it needs no explicit conflict resolution. The main philosophy of Pragma is to make it easy to declare common operations, and possible to declare uncommon ones. In a practical application like Bustuc, flexibility is needed to accommodate for the many cases in a real application where it is not suitable simply to add something to the destination depending on occurrences in the source. Pragma rules can condition on both the source and the destination, they can add and remove from both the source and destination and they can declare arbitrary Prolog code that must succeed after the source and destination is matched for the rule to trigger.

This flexibility has several advantages in Bustuc (and systems with similar needs) of which most has to do with reducing the number of rules:

- The fact that Pragma allows conditioning on destination terms in addition to source terms can be exploited to make the rule base shorter by using the information that already has been found by a rule to trigger new rules. The rule base thus takes advantage of what it already has discovered during the translation instead of being equally ignorant during the entire operation.

Consider the case where 50 different source occurrences leads to that A is added to a destination, and 50 other source occurrences leads to that B should be added to the destination provided that one of the first occurrences is also present.

With the ability to match for an A in the source, this is solved with 50+50 rules (50 for adding A, and 50 for adding B given A in the destination).

Without this ability, there would be 50*50+50 rules (50*50 to match for each combination of A and B sources that must produce B, and 50 to produce A alone).

- The fact that Pragma allows removal (and replacement) of already produced rules can be exploited in ways similar to the above.

  Consider when 50 source occurrences produces A, but that two occurrences should produce C instead of two A's. By using the technique described above, this would take 100 rules (50 for A and 50 for A given A), but by removing the two A's, one can do with 50+1 rules (50 for A, and one for replacing the with B given two A's).
- Replacing terms in the source can reduce the number of rules when source terms that occur in many rules comes in several variants. All the term variations can be replaced with a standardized one in the source so that one only need to write rules that depends on the standardized version.
- The ability to evaluate arbitrary Prolog code before the actions of a rule is executed gives unlimited flexibility, but the most important use is to condition rules on the content of data bases.

4.4.2 The syntax and semantics of Pragma rules

The syntax of Pragma rules are

```
is  S
id  D
ip  P
```

where

is reads "in source",
id reads "in destination"
ip reads "if this Prolog code succeeds"[1]

and S and D are conjunctions of *Pragma commands*, grouped by parentheses as in Prolog, and P are arbitrary Prolog code.

In the following, A and B are arbitrary grouped conjunctions of whatever the source and destination consist of.

Pragma *source* commands are

```
A
```

1. Unfortunately, "if" was already used by TUC

means *if A is present in source and previously not seen by any rule*

`present A`
means *if A is present in source (regardless of if it has been seen before)*

`not A`
means *if A is not present in source*

`replace A with B`
means *replace first occurrence of A with B in source*

`remove A`
means *remove first occurrence of A in source*

`exactly A`
means *if A is exactly like the source (not just a part of it)*

Pragma *destination* commands are

`A`
means *if A is present in destination*

`add A`
means *add A to the end of destination*

`remove A`
means *remove A from destination*

`replace A with B`
means *replace first occurrence of A with B in destination*

`to A append B`
means *append B immediately after first occurrence of A in destination*

`replacelast A with B`
means *replace last occurrence of A with B in destination*

`not A`
means *if A i not present in destination*

`no A`
means *if there are an A in the destination, remove it, but succeed anyhow*

`exactly A`
means *if A is exactly like the destination (not just a part of it)*

Pragma will try to unify parts of the source and destination with the expression in the rules to make them fit, but it will never backtrack to undo a rule.

Notice also that the order of the conjunctions in the rules does not need to correspond with the order in the source or destination, so that the rule

```
is  hi,you
id  add hello
```

```
if  [].
```

will produce a *hello* in the destination also if the source is *you, hi*.

Pragma remembers which terms in the source that has already been used to match a rule, so a rule will trigger only once on a given content in the source. However, it is possible to match the same part of the source indefinitely many times by using *present*.

4.5 Bustrans - from TQL to Buslog

This section will present the Busrans rule base.

4.5.1 Bustrans

Bustrans is the rule base that produces a Buslog program from the TQL expression when fed to the Pragma interpreter. The Bustrans rule base is a reasoning system that determines the pragmatic meaning of a semantic representation of a natural language query. The Buslog program that is produced can be viewed as both a pragmatic representation of the input question and as a program that will retrieve the information that the user needs when it is run.

4.5.2 The rules

The about 260 Bustrans rules can be divided into the following categories:

- Rules that determines the day of discourse.

  An *atday* predicate is inserted in the Buslog program to tell which day we are talking about. For instance does this rule trigger if the users says something like “on monday”.

  ```
  is  srel/WeekDay/time/nil/_
  id  add atday(WeekDay)
  ip  isday(WeekDay).
  ```

  There are also rules to deal with “yesterday”, “tomorrow”, “on weekends”, “on weekdays” and that no day is mentioned.
- Rules that discovers irrelevant and incorrect assumptions in the user.

  These rules discovers frequent incorrect assumptions and adds expressions to the Buslog program that will lead to that a sentence that informs the user of this. The rules will not affect other rules, so the user will still get an answer to his questions if it is otherwise meaningful. Such assumptions includes existence of nonexistent buses and relevancy of weather for bus departures.
- Rules that handles trip price questions.

  These rules handles questions about the price of a trip, the cost of travelling, the price of a ticket etc.
- Rules that handles explicit questions about time and date.

  For instance if the user asks “what time is it”, the following rule triggers:
  ```
  is  exactly (which(Time),Time isa time,event/real/_)
  ```

```
id add (timenow(T),timeis(T))
ip [].
```

Notice that this rule only triggers if the TQL expression consists of exactly these conjunctions. This is necessary because this code can appear in larger TQL expressions where time plays a part, without the question being about the time of the day.

There are also rules to handle other questions about time and the date.

- Rules that handles questions about stations.

  Rules handling station questions can be divided into tree subcategories:

  - Rules for handling questions about which stations a bus pass, as in “where can I go off 5?”, “which stations is at route 5” and “which stations does bus 5 pass?” combined with such as “between Nardo and Blakli?”.

  - Rules for handling questions about transfers, as in “where can one change from 5 to 66?” and “does 4 and 9 meet at Buran?”.

  - Rules for handling questions about places and station as in “which stations is at NTNU?” and “is Munkegata near downtown?”

- Rules that handles questions about bus departures or trips.

  Such questions can be formulated in very many ways, such as “when does bus 5 leave from Nardo”, “can you travel from Nardo to Byåsen by bus?”, “is there a bus from byen to NTNU?”, “Can i get to munkvoll from byen?” and so on.

  These phrases are recognized by rules in this category which adds one or more *departure* predicates are added to Buslog.

  If more than one station is mentioned, a rule will detect this after each place phrase has been recognized separately, and generate a *coupled* predicate.

- Rules that handles phrases which says something about trip direction.

  Trip direction is combined with departures in questions like “towards Blakli” and “in direction Byen”. These are handled by rules that adds direction filtering predicates to the *departure* predicates already in Buslog.

- Rules that handles phrases which says something about the time of trips.

  Time is combined with departures as in “after 12”, “at night”, “in the next 3 hours”, “around 1530” etc. These are handled by rules that adds time filtering predicates to the *departure* predicates already in Buslog.

- Rules that handles phrases that says something about the order and number of trips.

  Order and number of trips are expressed in phrases like “three next”, “the last”, “the second first” etc. These phrases are handled by rules that adds information to the *departure* predicate that tell which departures in the final departure list that should be used.

- Rules that handles questions about departure intervals (or frequencies)

  Departure frequencies are handled by adding a predicate that finds the two most significant intervals of departures in a *departure* or *coupled* predicate. In this way, all constraints that is understood about departures is also understood about frequencies.

- Rules that handles questions about buses

  Questions about buses are handled in the same way as those for frequencies, a list of departures is used and the list of buses are extracted from the resulting departure list.

- Rules that handles questions about time length of bus trips.

Length of bus trips is handled by rules that draws the length information out of a *coupled* predicate if it exists.

4.5.3 Questions handled by TUC

If there are no fitting rules in Bustrans for the input TQL expression, the TQL expression is handed to TUC's reasoning module, so TUC can try to handle it instead.

Even the Bustuc system is in principle restricted to handle questions in the bus domain, a few other questions should be handled, either because they are so common or because it would be regarded as impolite not to answer them. TUC is told some facts at start-up that it is used to answer such questions. Some examples are

you look like a computer.
you can tell me about bus departures.
you understand english.
tore and jon made you.
you are nice.

("you" here refers to TUC, of course.).

In addition to answering questions about these facts, TUC is used to find the count of buses and to tell if something is a station or not.

4.5.4 Discussion

To keep the number of rules low, the Bustrans system aims at understanding small parts of the question separately and combine this knowledge into an understanding of the complete sentence, mostly by employing the techniques explained in the Pragma section.

In most cases sentences parts combine regularly into larger sentences, but in some cases the pragmatic meaning of the complete sentence is not a regular combination of it's part. Such cases are easily handled by making larger, special rules to handle the cases. The Pragma system makes no restrictions on what parts and how much of the TQL expressions are handled by each rule, so small rules that handle small common sentence parts combine fine with special rules to handle complete sentences.

4.6 Busans - From Buslog to answer program

The Busans rules produce an answer program from a instantiated (executed) Buslog program when interpreted by the Pragma interpreter.

The answer program consists of a set of concepts, atoms and lists. When the answer program is executed, the concepts will be output as strings on the current language, atoms will be output with their namestring or as themselves, and lists will be output as atoms separated by "," and "and". The Busans rules produces words in an internal form ("concepts") instead of plain text to keep the rules language independent. In some cases, the sentence structure is different in norwegian than in english, so there are still a few rules that are language dependent.

The rules makes output that expresses all aspects of the Buslog program, to enable the user to discover when the Buslog program did not represent her intentions with the question. Some of the answer rules make complete sentences of output, while others modify these sentences. There is a natural correspondence between Buslog predicates that finds list of solutions and answer rules that make complete sentences, and Buslog predicates that filters lists of solutions and answer rules that modify output sentences.

Since some predicates make complete sentences, there is little need for explicit grammatical knowledge in the output rules. The little that is needed, is encoded as normal rules. Low-level formatting, like capitalizing the first word in a sentence and putting spaces between concepts is also handled by the rules.

In all, there are about 70 Busans rules. Some examples that shows how the rules are used for different tasks are shown below.

This rule outputs sentences like “Bus 5 normally passes by Nardo every 10 minutes or every 7 minutes.”:

```
is  passevent(Deps,Bus,Place,_,_,_),
    frequency(Deps,MinF,MaxF,_)
id  add (bcp(bus),bwr(Bus),bcp(normally),
         bcp(passes),bwr(Place),
         bcp(every),bwr(MinF),bcp(minutes),
         bcp(or),
         bcp(every),bwr(MaxF),bcp(minutes),
         period)
ip  [].
```

This rule adds a phrase like “after 1700” at the appropriate place to the sentence made by the previous rule if the departure set is filtered to be after some time.

```
is  keepafter(Time,_,_)
id  to     (bcp(or),bcp(every),bwr(_),bcp(minutes))
    append (bcp(after),bwr(Time))
ip  [].
```

This rule removes plural form where there only one station (instead of a list) follows.

```
is  []
id  replace (bcp(thestations),bwr(Station))
    with    (bcp(thestation),bwr(Station))
ip  atomic(Station).
```

This rule bigcaps the first letter in a sentence.

```
is  []
id  replace (Sentenceend,bcp(A))
    with    (Sentenceend,bcpbc(A))
ip  sentenceend(Sentenceend).
```

4.7 A detailed example

In this section, an example query will be presented in detail, from input to output text. This will illustrate how the different phases and the rule bases works to produce an answer to a input sentence.

Input question:

how often does 5 leave from nardo in direction blakli this evening?

This sentence produces the following TQL expression:

```
which(A)::(5 isa bus,blakli isa direction,
           nardo isa neighbourhood,B isa evening,
           leave/5/C,event/real/C,
           srel/with/frequency/A/C,
           srel/from/place/nardo/C,
           srel/in/direction/blakli/C,
           srel/in/time/B/C)
```

The translation to Buslog program goes as follows:

First, a rule triggers to insert the current (default) day into the Buslog program when no day is mentioned (no other rules has inserted a *atday* predicate):

```
is  []
id  not atday(_),
    add (today(Day),atday(Day))
ip  [].
```

The Buslog program is now

```
today(Day),
atday(Day)
```

Then a rule triggers that recognizes that the question talks about bus 5 leaving (a kind of travelling) from a place, and adds a departure predicate, and a *passevent* predicate that says that this list is the current actual list of departures containing the answer.

```
is  present Bus isa bus,present TRAVEL/Bus/C,
    srel/from/place/Place/C
id  atday(Day),
    add (departure(Bus,Place,Day,Depset),
         passevent(Depset,Bus,Place,[],Day,C))
ip  statorplace(Place),busorfree(Bus),
    dmeq(travel,TRAVEL).
```

(dmeq(A,B) means that the word B is equivalent to A in this domain.)

The Buslog program is now

```
today(Day),
atday(Day),
departure(5,nardo,Day,Departures),
```

```
passevent(Departures,5,nardo,[],Day,Situation)
```

Then a rule triggers that recognizes that the situation of the departure is in direction byen. Notice that the *passevent* predicate is replaced to reflect that the actual set of departures have changed:

```
is  Place isa direction,srel/in/direction/Place/C
id  replace passevent(Depset,Bus,OnPlace,Opts,Day,C),
    with    (keepto(Place,Depset,NewDepset),
             passevent(NewDepset,Bus,OnPlace,Opts,Day,C))
ip  statorplace(Place).
```

The Buslog program is now

```
today(Day),
atday(Day),
departure(5,nardo,Day,Departures),
keepto(blakli,Departures,NewDepartures),
passevent(NewDepartures,5,nardo,[time],Day,Situation))
```

Last, a rule recognizes that the event is in the evening (taken to be between 1700 and 2200) triggers:

```
is  A isa evening,srel/AT/time/A/_
id  replace passevent(Deps,Bus,Place,_,Day,B)
    with    (keepbetween(1700,2200,Deps,NewDeps),
             passevent(NewDeps,Bus,Place,[time],Day,B))
ip  dmeq(at,AT).
```

The Buslog program is now

```
today(Day)
atday(Day)
departure(5,nardo,Day,Departures1)
keepto(byen,Departures1,Departures2)
keepbetween(1700,2200,Departures2,Departures3)
passevent(Departures3,5,nardo,[time],Day,Situation)
```

Finally, a rule that recognizes that we are asking for departure frequencies, not departure times, triggers.

```
is  present which(A),
    present srel/with/frequency/A/B
id  not frequency(_,_,_,B),
    passevent(Deps,_,_,_,_,B),
    add frequency(Deps,_,_,B)
ip  [].
```

The final Buslog program is

```
today(Day)
atday(Day)
departure(5,nardo,Day,B)
keepto(blakli,Departures1,Departures2)
keepbetween(1700,2200,Departures2,Departures3)
```

```
passevent(Departures3,5,nardo,[time],Day,Situation)
frequency(Departures3,Interval1,Interval2,Situation)
```

This program is now evaluated by Prolog and all the free variables are bound to the right values derived from the data base.

Next, the Busans rule base is used to generate an answer program from the instantiated Buslog program.

First a rule that recognized that this program was about frequencies, and produces a complete sentence is trigged:

```
is  passevent(Deps,Bus,Place,_,_,_),
    frequency(Deps,Interval1,Interval2,_)
id  add  (bcp(bus),bwr(Bus),bcp(normally),
          bcp(passes),bwr(Place),
          bcp(every),bwr(Interval1),bcp(minutes),bcp(or),
          bcp(every),bwr(Interval2),bcp(minutes),period)
ip  [].
```

This rule basically produces *Bus 5 normally passes by Nardo every 10 minutes or every 15 minutes* except that spaces and capitals are missing.

Then a rule that recognizes that the departures was between 1700 and 2200 modifies this program to include that fact:

```
is  keepbetween(FromTime,ToTime,_,_)
id  to     (bcp(or),bcp(every),bwr(_),bcp(minutes))
    append (bcp(between),bwr(FromTime),
            bcp(and),bwr(ToTime))
ip [].
```

This rule turns the sentence into *Bus 5 normally passes by Nardo every 10 minutes or every 15 minutes between 1700 and 2200.*

Then another rule recognizes that the departures are in direction byen, and modifies the sentence accordingly:

```
is  keepto(Place,_,_),present passevent(_,Bus,_,_,_,_)
id  to     (bcp(bus),bwr(Bus),bcp(normally),
            bcp(passes),bwr(_))
    append (bcp(direction),bwr(Place))
ip  [].
```

The output sentence then becomes *Bus 5 normally passes by Nardo in direction Blakli every 10 minutes or every 15 minutes between 1700 and 2200.*

At last, the following rules trigger repeatedly to produce respectively spaces between words and bigcaps at the start of sentences.

```
is  []
id  replace (A,B) with (A,space,B)
ip  [] :-
```

```
            (A=bcp(_);A=bwr(_)),
            (B=bcp(_);B=bwr(_)).

is  []
id  replace (Sentenceend,bcp(A))
    with    (Sentenceend,bcpbc(A))
ip  sentenceend(Sentenceend).
```

The resulting output program is

```
bcpbc(bus)
space
bwr(5)
space
bcp(normally)
space
bcp(passes)
space
bwr(nardo)
space
bcp(direction)
space
bwr(blakli)
space
bcp(every)
space
bwr(13)
space
bcp(minutes)
space
bcp(between)
space
bwr(1700)
space
bcp(and)
space
bwr(2200)
period
```

This program will use the concept to word mappings to produce

> Bus 5 normally passes by Nardo in direction Blakli every 10 minutes or every 15 minutes between 1700 and 2200.

if the language is english, and

> Buss 5 passerer vanligvis Nardo i retning Blakli hvert 10. minutt eller hvert 15. minutt mellom 1700 og 2200.

if the language is norwegian.

CHAPTER 5 Future work

This chapter will outline some areas for interesting future work relating to Bustuc. The use of Bustuc in a dialogue system is also discussed.

5.1 Using TUC's grammar for making answers

A natural extension is to employ TUC's knowledge of language to generate output text. When the answer generation grows more complicated, more grammatical knowledge will be needed to make good answer. Using TUC for this has the following advantages:

- One is ensured that the output of the applications can be parsed by TUC, without taking special care of this.
- Multiple languages will be of no concern in the application.
- No grammatical knowledge and dictionary knowledge will need to exist dually in both the application output system and TUC.
- Any new grammatical knowledge in TUC will automatically be applicable for the application.

To make TUC generate answers, one need a language independent interface between the application and TUC. One such interface already exists - the TQL language.

Thus, a natural way of doing this will be to make TUC able to paraphrase TQL sentences in natural language, and then make the application generate a TQL expression representing the answer instead of a answer program.

Making Bustuc generate TQL instead of a Busans program would not be very hard. It would consist mainly of changing the actual expressions produced by every rule in the Busans rule base. Making TUC able to paraphrase TQL expressions should also be feasible since TUC's grammar is declarative, and should be able to generate sentences as easily as it parses them.

5.2 Making a multilingual system

Merging the Norwegian and English versions into one system would be useful. The multilingual version must accept both languages (Norwegian and English) as input and answer in the language the user provided input in. This would be advantageous both from a developing perspective and from a user perspective:

- Developing two versions simultaneously generates a significant amount of extra work, since it means doing testing and compiling for two systems instead of one. Much of this extra work could be avoided by having one version that is tested for two languages at the same time.
- Users does not need to think of choosing a version that accepts their language, they can simply write in the language they chooses (Norwegian or English).

A multilingual version would not require that the two languages grammars are merged into one grammar with grammatical rules for both languages. The actual language could be determined in the lexical analysis - if no Norwegian words are found in the lexical analysis, the English grammar is used. If Norwegian words are found, the Norwegian grammar is used. This would potentially cause problems if people are mixing languages in a question, but understanding mixed language questions is not a requirement of a multilingual system.

5.3 Generalizing Bustuc logic

Some of the knowledge of the Bustuc system is applicable in many other areas than bus traffic information. For instance, reasoning with time intervals is relevant in solving most task. The logic of Bustuc that falls into this category could be generalized and placed into a *common sense module*. This module could contain knowledge valid to all or most applications to avoid having to write down the logic all over again for each new application.

Technically, this includes extracting a section of the Bustrans rules into another rule base without damaging the co-operation of the rules, generalizing the Buslog logic in question and making the generalized knowledge work together with the special knowledge left in Buslog.

5.4 Metareasoning

When using a natural language application, many people will feel need to get answers to metaquestions - questions regarding the domain and the communication taking place. Some examples are

What do you know?
Can I ask you questions?
Will you answer questions about bus transfers?

In the current version, TUC can not answer such questions unless the exact answer to the question is supplied to TUC explicitly. Supplying explicit questions is unsatisfactory both because it will only cover certain, common questions and because it does not seem to be a very intelligent way to answer the question. After all - we want TUC to understand.

To answer such questions intelligently, we need to build a metareasoning module for TUC. This module should have a model of TUC itself, a model of the world including the users, and a model of the communication process. Based on these models, it should be possible to make TUC able to deduce reasonable and user-friendly answers to such metaquestions.

5.5 Towards a dialogue system

The ultimate goal of the TABOR project is to make a dialogue system capable of conduction dialogues about bus traffic over telephone. A dialogue system must in addition to doing what Bustuc does, have a dialogue manager that can reason over dialogues and generate appropriate clarification questions in addition to answering the users questions.

5.5.1 Using Buslog in a dialogue system

In a dialogue system there is still need for a pragmatic interpretation of every sentence. The Buslog program that Bustuc makes for every input sentence can be used as a pragmatic representation of the sentence in a dialogue system. The main task of the dialogue manager would then be to reason over a list of Buslog programs, where each element in the list represents a input sentence in the dialogue. Building on Bustuc in this way has two advantages over starting from scratch with a new pragmatic representation:

- The dialogue system will be at least as good as Bustuc from the start.A dialogue system must also be capable of answering complete single-sentence questions and this ability will be present for free when Buslog is used as representation.
- The Buslog representation can be interpreted at any point by Prolog to retrieve solutions, and it will be possible to generate answers from it.

The Bustrans rules will have to be changed somewhat because in many cases where the best response is to ask for clarification from the user, the Bustrans system chooses to make a reasonable guess of what the user meant instead, since the option of asking for clarification is not available. This will mostly be a straightforward task of just removing the rules that makes the guess, and removing the check for enough information that is done in the normal rules. For instance, if someone asks for frequencies of a bus without supplying a place, the place will default to the centre of town. This should be changed to just leaving the place variable unbound, so that the dialogue controller can discover that it is missing and decide to ask the user for it.

5.5.2 Ellipsis

A dialogue system requires understanding of ellipsis. An ellipse is a sentence that omits phrases used earlier in a discourse. Ellipsis is used by humans to avoid communicating information that can be inferred faster by the human brain than spoken.

An example is the following dialogue:

When does the first bus leave from Nardo in the morning?
The first bus, 5 passes by Nardosenteret at 553.
And the next bus?

Ellipsis resolution should ideally be TUC's responsibility since it should be resolvable by using semantic and syntactic information from the current and recent sentences. TUC will need to keep a stack of recent sentences, both the users sentences and the applications responses. This stack should contain both parse trees and semantic representations of earlier sentences, since some ellipse types can be resolved on the syntactic level, and others mostly on the semantic level. The ellipsis resolution should be integrated with the anaphora resolution phase of TUC.

Ellipsis resolution in TUC would be a general, application-independent solution to the problem. This is not likely to be achieved in the near future, since there are not yet any general agreed-upon theory of ellipsis. To get working results, the ellipsis resolution could be left to the dialogue manager of an application, which knows the pragmatic interpretation of sentences, and which therefore has a much easier task. Of course, this approach requires that TUC is able to parse and construct TQL expressions for incomplete sentences.

An incremental approach could also be used. The ellipsis types that there are good theories for could be handled by TUC while other types could be handled by the application. As more types of ellipsis get a good theoretical foundation they could be handled by TUC, eventually causing ellipsis resolution in the application to be unnecessary. Ellipsis that are beginning to be well understood and which therefore could be handled by TUC includes *gapping*, and *verb phrase-ellipsis* (Kehler 94). Gapping is characterized by the elision of all but two constituents in a target sentences following a source sentence, i.e.

Tuc became confused, and Jon angry.

VP-ellipsis is characterized by a bare auxiliary indicating the elision of a verb phrase from the source in the target sentence, i.e.

Tuc became confused, and Jon too.

5.5.3 Incomplete sentences

Regardless of ellipsis resolvment, TUC must be able to parse incomplete sentences when it shall be used for dialogue systems. For instance will most users respond with simple noun phrases when answering questions, as in

When does 36 leave?
from where?
Dragvoll.

TUC does already handle single noun phrases by parsing them as if they were proceeded by “what is”. This yields TQL code which does not properly represent the meaning of the sentence, so it should ideally be changed to something more appropriate.

Other incomplete sentences like single noun complements should also be parsed.

5.5.4 The dialogue manager

The dialogue manager has the responsibility of producing answers and questions in response to the users input, so that the user gets answers to her questions as fast and smoothly as possible. This goal requires that the dialogue manager can handle

- Multiple threads

  In human dialogues, there are often multiple threads (or focuses) that the dialogue swaps between. The dialogue controller should handle multiple threads and subthreads and handle changes between them appropriately.
- Interruption of threads

  The system should handle interruption of threads at any point. This happens when the user changes his mind, gives up or is satisfied with the information he has received.
- Variable initiative

  Some users wishes to take the initiative in the dialogue, while other will expect the system to control the dialogue. The dialogue controller should be able to have the initiative in the dialogue, but it should also handle users controlling the dialogue. In many cases users will take the initiative or give it away as they feel like. The dialogue manager should be flexible and responsive enough to take initiative when expected to and give up the control when the user takes it by supplying more or other information than expected.
- Validating and correcting input efficiently

  The system should present answers in such a way that the user can discover when the system has misunderstood her intent or input. The user should be able to correct such mistakes at any point.

In this domain, the problems are simple, but language is informal and rich. The main communication issue is resolving underspecified and vague questions. In other words, the basic mode of operation will be clarification dialogue, dialogue to determine what information that is missing to retrieve sensible answers from the data base and making the user to supply this information. This task should be manageable since it will consist mainly of determining which required variables in the Buslog program that is unbound and which questions that must be output to retrieve this information from the user.

CHAPTER 6 Results and conclusion

This chapter will presents the results of the project, and give some conclusions.

6.1 Results

The results of this project will be discussed in this section.

6.1.1 Pragma

The pragma production system is a general, flexible, expressive and practical production system which has value independent of the Bustuc project. It can be used for other applications together with TUC and for any other problem which has as main task to produce a set of conjunctions from another set of conjunctions, but which needs greater flexibility than a conventional production system.

6.1.2 TUC

TUC has proven good enough to be used successfully in a real world application. The requirements of the Bustuc system is quite different than those for a Unix natural language command system, which was the last application TUC was used with. This suggests that the adaptability goal of TUC is well taken care of.

TUC together with Pragma can be used as a practical tool to develop useful natural language applications rapidly at a high, declarative level. The steps that must be taken to develop the natural language end of a new application using TUC and Pragma are:

- Declare the semantics of the domain building on TUC's existing semantic knowledge base.

- Declare Pragma rules to build a program that makes the appropriate reasoning from TQL expressions.
- Declare Pragma rules that produce a response from the instantiated reasoning program.

The work required at the natural language side for a new application using TUC and Pragma is greatly reduced compared to starting from scratch since using TUC and Pragma requires only high-level declarations, and since TUC provides for many of the things applications have in common.

6.1.3 The web application

The Bustuc system has been publicly available at World Wide Web at

```
http://www.idt.ntnu.no/~tagore/bustuc/
http://www.idt.ntnu.no/~tagore/busstuc/
```

(English and Norwegian versions) throughout the entire project. All the questions that has been asked and their answers has been saved in a log file. This has enabled testing of the system and improvement in areas where there was a proven need. The questions asked shows a great variety in language but only a moderately small set of problems to be solved. See appendix X for some example questions and answers.

At the end of this project, all the questions from the log was run on the latest version of Bustuc to test the applications ability to answer real questions. The test results are presented at the next page. The test was conducted as a low-priority process on a SPARCstation 10 student server.

Norwegian version

```
Total number of questions:      2304
Percentage answered correctly:  78.3 %
Answer processing time mean:    3512 milliseconds
Answer processing time median:  1880 milliseconds

Error types (in percentage of all questions):
  Thrash (meaningless) input:   0.2 %
  Wrong language (English):     0.2 %
  Unknown (names of) places:    2.0 %
  Unknown words:                2.0 %
  Misspelled words:             2.9 %
  Incomplete sentences:         4.0 %
  Missing grammatics:           4.3 %
  Missing semantics:            1.1 %
  Missing pragmatics:           2.1 %
  Obviously outside the domain: 1.8 %
  Multiple sentences:           1.1 %
```

English version

```
Total number of questions:      2534
Percentage answered correctly:  72.7 %
Answer processing time mean:    2935 milliseconds
Answer processing time median:   330 milliseconds

Error types (in percentage of all questions):
  Thrash (meaningless)input:    2.2 %
  Wrong language (Norwegian):   2.4 %
  Unknown (names of) places:    2.2 %
  Unknown words:                1.4 %
  Misspelled words:             3.4 %
  Incomplete sentences:         4.9 %
  Missing grammatics:           4.4 %
  Missing semantics:            1.5 %
  Missing pragmatics:           1.5 %
  Obviously outside the domain: 3.1 %
  Multiple sentences:           0.5 %
```

The main observation here is that the Bustuc system stands the test of the real world. The web page gives no hints about the type of language to be used in the questions, except that the system understands *questions* as *complete sentences*. However, it must be noted that he questions from this test has been driving some of the improvements of the system, so the results are a bit biased. The percentage of new questions that is answered correctly seems to be a few points lower. On the other hand, many of the errors is clearly from users that is more interested in testing the limits of the system than in finding out about bus departures.

Some other observations can also be made from the results:

- The response time is acceptable, even when the tests was run at low priority on a machine with other users. The median is lower than the mean because in a few cases TUC uses several minutes to decide it can not parse a sentence.
- Even a perfect system will not have a success rate at 100%, at least not when available on World Wide Web. For instance, in the English version 7.7% of the questions are thrash input, obviously outside the domain or in the wrong language (in Norwegian).
- Allowing incomplete sentences and spell correcting all words (not only names) will increase the success rate by about 7 or 8%.
- Missing domain knowledge accounts for only about two percents of the errors, which means that it is quite possible to make the system close to perfect in this respect.
- Roughly speaking, half of the errors can be handled by improving TUC, a quarter of the errors by improving the Bustuc application and the last quarter of the errors can not be handled by a limited domain system.

Some examples of working questions and the answers Bustuc gives is presented in appendix A. A discussion of the error categories used above and some example questions from each category is presented in appendix B.

6.2 Conclusion

As the results from the last section show, we have succeeded in making an application that handles most real questions in the bus traffic domain. The system still makes errors, but only about a quarter of them (about 6 to 7% of all questions) can be solved by improving the Bustuc application.

By achieving this, we have also proved that TUC can be used for real word applications. We also believe that the Pragma system will be a useful tool together with TUC when new applications is made.

References

Allen, J F ; Ferguson, G ; Miller, B ; Ringger, E (1995). *Spoken Dialogue and Interactive Planning*. Department of Computer Science, Univerity of Rochester

Amble, T ; Knudsen E ; Lehtola A ; Ljungberg J ; Ravnholt O (1989). *Naturligt språk & grafik - nye vagar in i databaser.*

Amble, T (1994). *Domain Modelling and Automated Natural Language Interfaces*. Knowledge Systems Group, Norwegian University of Science and Technology

Amble, T (1995). *Automatic Solving of Problems stated in Natural Language*. Knowledge Systems Group, Norwe gian University of Science and Technology

Bratseth, J S (1996). *Bustuc - A natural language interface to public traffic knowledge.* Knowledge Systems Group, Norwegian University of Science and Technology

Covington, M A (1994). *Natural Language Processing For Prolog Programmers*, Prentice-Hall

Eisner, J M (1996). *Three New Probabillistic Models for Dependency Parsing: An Exploration*. CIS Department, University of Pennsylvania

Gudmestad, J (1996). *Dialogue Structures*. Knowledge Systems Group, Norwegian University of Science and Technology

Kehler, A (1994). *Common topics and coherent situations: Interpreting ellipsis in the context of discourse inference*. Harvard University, Aiken Computation Laboratory

Russel S ; Norvig P (1995). *Artificial Intelligence - A modern approach*. Prentice Hall

Quillian, M R (1967). *Word concepts: A theory and simulation of some basic semantic capabilities*. Behavioral Science 12

Smith, R W ; Hipp, R D (1994). *Spoken Natural Language Dialog Systems - A practical approach*. Oxford University Press

Veale T ; Keane M (1995). *Epistemological Pitfalls in Metaphor Comprehension: A comparison of Three Models and a New Theory of Metaphor.* Presented at The International Cognitive Linguistics Conference 1995

Way, E C (1991). *Knowledge representation and Metaphor.* Kluwer Academic Publishers

Wøien, K & E (1995). *Unix Help, A natural language interface to Unix*. Knowledge Systems Group, Norwegian University of Science and Technology

APPENDIX A Examples

This appendix presents some examples on questions and the answers Bustuc produces to show the variety of language accepted by Bustuc and the answers it returns. Some of the answers will of course depend on when the question is asked. Most of the examples are taken from the Web log.

A.1 English version

Can you tell me when a bus leaves from Munkvoll in direction sentrum today?

> Yes. The first bus, number 5c passes by Munkvoll at 616 in direction Sentrum.
> The next bus after now, number 5e passes by Munkvoll at 2016 in direction Sentrum.
> The last bus, number 5c passes by Munkvoll at 2346 in direction Sentrum.

How can i get from Nardo to Jakbsli by bus tonight?

> 5 passes by Nardosenteret at 1928 and 66 passes by Idrettsplassen Jakobsli at 2000.
> You can change from 5 to 66 by leaving bus 5 at Studentersamfundet at 1938 and entering bus 66 at Studentersamfundet at 1943.

When does the six next buses pass by Professor Brochs Gate towards Lerkendal?

> Bus 5c passes by Prof.Brochs gate at 2030 and 2110 in direction Lerkendal.
> Bus 60 passes by Prof.Brochs gate at 2055 in direction Lerkendal.
> Bus 66 passes by Prof.Brochs gate at 2045 in direction Lerkendal.
> Bus 7 passes by Prof.Brochs gate at 2055 in direction Lerkendal.

Can one take the bus from E C Dahls to Nedre Møllenberg between 12 and 13 on Sunday?

Yes. At Sunday, bus 4 passes by Vinmonopolet at 1242 and arrives at Rosenborg gate at 1246.

Bus from Byen to Granåsen around 2100?

Bus 24c passes by Hovedterminalen at 2115 and arrives at Granåsen at 2127.
Bus 9 passes by Hovedterminalen at 2045 and arrives at Granåsen at 2106.
Bus 9e passes by Hovedterminalen at 2045 and arrives at Granåsen at 2105.

How often does bus number 5 go from Blakli to byen tonight?

Bus 5 normally goes from Blakli to Byen every 30 minutes after 1900.

Which buses passes by NTH and Steinan Studentby?

52 and 5e passes by Nth and Vestlia.

Where does 5 come from when it passes by Tors vei at 2000?

When bus 5 passes by Tors veg at 2000, it comes from Blakli.

What day is it?

It is Monday 16. of 2. 1997.

How often does a bus go from Gløshaugen to Moholt Studentby?

A bus normally goes from Nth to Moholt Studentby every 60 minutes or every 30 minutes.

Which buses stops at Skistua on Sunday mornings before 1100?

At Sunday, bus 10 passes by Skistua at 1005 and 1035.

What bus should I take to get from Fiolsvingen to Vestlia?

The first bus, number 52 passes by Fiolsvingen at 609 and arrives at Vestlia at 611.
The next bus after now, number 5e passes by Fiolsvingen at 2042 and arrives at V estlia at 2047.
The last bus, number 5e passes by Fiolsvingen at 2403 and arrives at Vestlia at 2406.

When does bus 5 go?

You must specify a place in such questions.

How long does it take to go by bus from Risvollan to Sjetnemarka?

9 passes by Blakli at 1459 and 49 passes by Hagen at 1530.
You can change from 9 to 49 by leaving bus 9 at Okstad at 1519 and entering bus 49 at Okstad at 1523.
That means going from Blakli to Hagen takes 31 minutes.

Where can i change from bus 41 to bus 8?

Both 41 and 8 passes by the stations Studentersamfundet, Prinsen Kinosenter, Torvet and Hovedterminalen.

What is the price of a bus ticket?

The price of a trip is 16 or 8 NOK.

What are the endstations for bus 8?

The end stations for 8 is Hovedterminalen, Dragvoll and Stavset.

Are there any departures for bus 36 from midtbyen to Charlottenlund this afternoon?

Yes. The first bus after 1200, number 36 passes by Hovedterminalen at 1210 and arrives at Konglevegen at 1231.
The next bus after now, number 36 passes by Hovedterminalen at 1310 and arrives at Konglevegen at 1331.
The last bus, number 36 passes by Hovedterminalen at 2410 and arrives at Konglevegen at 2429.

When do I have to take the bus from Trondheim, when I want to arrive at Dragvoll around 0900?

The first bus after 845, number 66 passes by Hovedterminalen at 845 and arrives at Universitetet Dragvoll at 900.
The last bus before 915, number 9 passes by Hovedterminalen at 915 and arrives a t Dragvoll at 935.

When is the third first arrival at Byåsen from Trondheim tomorrow?

At Tuesday, the third bus, 8 passes by Hovedterminalen at 625 and arrives at Byåsen at 652.

Can i travel by bus from Ila to Sverresborg folkemuseum the next 50 minutes?

Yes. Bus 8 passes by Ila at 1333 and arrives at Sv.B.Folkemuseum at 1340.
Bus 9 passes by Ila at 1319 and 1349 and arrives at Sv.B.Folkemuseum 6 minutes later.

Is Munkegata near Dronningensgate?

Yes. Munkegata is near Dronningensgate.

Can you show me the buses for Vernes the next hour?

I have only routes for buses in Trondheim.

When does the 3 last buses leave from Studentersamfundet to Valgrindveien tonight?

Bus 7 passes by Studentersamfundet at 2318, 2323 and 2353 and arrives at Valgrindvegen 3 minutes later.

Is it possible to go from Risvollan to Jakobsli between 2300 and 2350?

Yes. 5 passes by Blakli at 2320 and 66 passes by Idrettsplassen Jakobsli at 2400.
You can change from 5 to 66 by leaving bus 5 at Lerchendal Gård at 2329 and ente ring bus 66 at Lerchendal Gård at 2347.

A.2 Norwegian version

Når går neste buss fra Lade til Byen fra NTNU?

Buss 4 passerer Universitetet Lade klokken 1416 på vei fra Lade til Byen.

Hvilke stasjoner passerer 63 mellom samfundet og festningen?

63 passerer stasjonene Jonsvannsveien, Gudes gate og Vollabakken mellom Samfundet og Festningen.

Kan man reise med buss fra Selsbakk til Torvet innen klokka 9?

Ja. Første buss, nummer 5a passerer Selsbakkflata klokken 603 og kommer til Torvet klokken 626.
Siste buss før 900, nummer 6 passerer Selsbakk klokken 849 og kommer til Torvet klokken 900.

Kan jeg komme meg fra Romulslia til kinosenteret til klokka 19?

Ja. 43 passerer Romulslia klokken 1801 og 49 passerer Prinsen Kinosenter klokken 1818.
Du kan bytte fra 43 til 49 ved å gå av buss 43 på Selsbakk klokken 1802 og å gå på buss 49 på Selsbakk klokken 1809.

Hvor mye er klokka?

Klokka er 1424.

Når går det en rød buss fra Buenget til Kongens Gate omkring 700 i dag?

Fargen til en buss er irrelevant.
Buss 3 passerer Buenget klokken 650 og 710 og kommer til Hospitalskirka 20 minutter senere.
Buss 5a passerer Buenget klokken 700 og kommer til Hospitalskirka klokken 720.

Kan jeg ta bussen fra Tyholt til Byåsen etter 2300 på søndager?

Ja. På søndag passerer 60 Tyholt klokken 2312 og 5 passerer Munkvoll klokken 23 45.
Du kan bytte fra 60 til 5 ved å gå av buss 60 på Munkegata (M3) klokken 2331 og å gå på buss 5 på Hovedterminalen klokken 2335.

Når går siste bussen fra NTH til Moholt?

Siste buss, 8a passerer Nth klokken 2416 og kommer til Moholt klokken 2420.

Hvor går buss 69?

Det er ingen buss som kalles 69.

Når kjører første bussen fra Sentralstasjonen til Dragvoll?

Første buss, 9 passerer Trondheim Sentralstasjon klokken 617 og kommer til Dragvoll klokken 635.

Hvordan kommer jeg fra kaia til Sjørdalen i 16 tida?

Jeg har bare ruter for busser i Trondheim.
Første buss, nummer 46 passerer Pirterminalen klokken 615.
Neste buss etter nå, nummer 46 passerer Pirterminalen klokken 2307.
Siste buss, nummer 46 passerer Pirterminalen klokken 2407.

Kan jeg ta overgang fra buss 36 til buss 8 på Moholt Studentby?

Ja. Både 36 og 8 passerer stasjonen Moholt Studentby.

Når går buss forbi Nardosenteret mot byen, etter kl 23?

Buss 5 passerer Nardosenteret klokken 2325 og 2353 i retning Byen.

Hvilke busser går det fra Jakobsli?

24c, 24, 36 og 66 passerer Idrettsplassen Jakobsli.

Når går bussen fra sentrum til Heimdal etter kl 1800 og før kl 2000 ?

Første buss etter 1800, nummer 4 passerer Hovedterminalen klokken 1802 og kommer til Heimdal stasjon klokken 1825.
Siste buss før 2000, nummer 48 passerer Hovedterminalen klokken 1942 og kommer til Heimdal stasjon klokken 1959.

Går det en buss til Tyholt fra Buran nær klokken 0800?

Ja. Buss 20 passerer Buran klokken 809 og 814 og kommer til Tyholt 13 minutter senere

Hvordan kan jeg reise fra Rotvoll til Skansen?

Buss 6 passerer Rotvoll klokken 1824, 1924, 2024, 2124, 2224 og 2324 og kommer til Skansen 19 minutter senere.

Når går de fem første bussene fra sentrum til Ugla søndag morgen?

På søndag passerer buss 8 Hovedterminalen klokken 1015 og 1045 og kommer til Ugla 13 minutter senere.
Buss 9 passerer Hovedterminalen klokken 945 og 1045 og kommer til Ugla 15 minutter senere.
Buss 9e passerer Hovedterminalen klokken 945 og kommer til Ugla klokken 959.

Hvor ofte går det en buss mot Reppe fra byen på hverdager?

En buss passerer vanligvis Byen i retning Reppe hvert 60. minutt eller hvert 52. minutt.

Når går første buss forbi Gløshaugen til Steintrøveien etter klokken 1500 ?

Den andre bussen, 8a passerer Nth klokken 1501 og kommer til Steintrøvegen klokken 1510.

Når gikk nest første buss fra Dalen hageby til Byen i går?

På søndag passerer den andre bussen, 6 Dalen Hageby klokken 928 og kommer til Byen klokken 945.

Hvilke bussholdeplasser passerer buss nr. 9e mellom Steinberget og Nyborg?

9e passerer stasjonene Fagerliveien, Schiøtz vei, Hammersborg, Sv.B.Folkemuseum og Odenseveien mellom Steinberget og Nyborg.

Hvor kjører buss nr. 9E fra når den passerer nyborg ca. kl. 21?

Når buss 9e passerer Nyborg mellom 2045 og 2115, kommer den fra Hovedterminalen eller Stavset.

Når ankommer det busser til Grensen fra byen i de neste tre timene?

Første buss etter 1546, nummer 52 passerer Hovedterminalen klokken 1550 og kommer til Vollabakken klokken 1553.
Siste buss før 1846, nummer 8a passerer Hovedterminalen klokken 1840 og kommer til Vollabakken klokken 1844.

Hvilke busser passerer Kroppan Bro og Heimdal?

48, 4a, 4 og 90 passerer Kroppan bru og Heimdal.

Kan du liste de 6 neste avgangene fra Ola Setroms Vei til Byen?

Ja. Buss 8 passerer Ola Setroms veg klokken 1601, 1616, 1631 og 1646 og kommer til Byen 18 minutter senere.
Buss 9 passerer Ola Setroms veg klokken 1619 og kommer til Byen klokken 1638.
Buss 98 passerer Ola Setroms veg klokken 1629 og kommer til Byen klokken 1651.

Må jeg skifte buss mellom Hallset og jakobsli?

Ja. 5c passerer Nordre Hallset klokken 2344 og 36 passerer Idrettsplassen Jakobs li klokken 2434.
Du kan bytte fra 5c til 36 ved å gå av buss 5c på Hovedterminalen klokken 2405 o g å gå på buss 36 på Hovedterminalen klokken 2410.

Når går første buss etter 1630 forbi Ladeveien til Munkegata?

Første buss etter 1630, 4 passerer Ladeveien klokken 1647 og kommer til Munkegata klokken 1702.

Når kommer siste bussen som går fra Trollahaugen til holdeplassen biologen?

Siste buss, 75 passerer Trolla klokken 2243 og kommer til Biologen klokken 2304.

Når går linje 66 fra Professor Brochs Gate til Peder Kroghs Vei mellom 1500 og 1600?

Buss 66 passerer Prof.Brochs gate klokken 1500, 1515, 1530, 1545 og 1600 og kommer til Peder Kroghs veg 3 minutter senere.

Hvor kommer 66 som passerer dragvoll klokken 2355 fra?

Når buss 66 passerer Dragvoll, kommer den fra Hovedterminalen.

Buss fra Rotvoll til Væretrøa mellom 12.00 og 15.00?

Buss 6 passerer Rotvoll klokken 1224, 1249, 1319, 1349, 1419 og 1449 og kommer t il Væretrøa 13 minutter senere.

APPENDIX B

Errors and error categories

This appendix will present some examples of sentences that are not understood by Bustuc, and discuss the categorization of errors that was used in the results in chapter 6. All the examples in this appendix is taken from the Web log.

The classification of errors into one of the eleven categories used in chapter 6 is not always straightforward. Some sentences fail of several reasons that belongs in different categories, and some failure reasons can belong in several categories. We will here say a bit more about which sentences we have placed in each category.

The percents given are of all questions in the logs and are repeated from chapter 6 for convenience.

- **Thrash (meaningless) input**

  The input that is placed in this category is input for which there does not exist any good answers (at least not when the input is taken as single questions).

  Examples are:

  What?
  w lksw,.aZL
  PLING.
  ji-dl lwdj jdiqw

  0.2% of the Norwegian and 2.2% of the English questions was placed in this category.

- **Wrong language**

  All sentences that was in the wrong language (of English and Norwegian) was placed in this category.

These errors could be eliminated by merging the English and Norwegian versions into one as discussed in section 5.2

0.2% of the Norwegian and 2.4% of the English questions was placed in this category.

- **Unknown (names of) places**

  The questions that was placed in this category are those that would have been answered correctly by Bustuc if the places used was known.

  Most of the errors is due to that Bustuc does not know all the names used on places in Trondheim. Coding in most of the names used is a large and tedious but quite simple job. Some of the errors are misspellings that are too far from the correct word to be recognized by TUC. Those can be solved by letting TUC do spell correcting on word distances of more than 1. Finally, some of the errors are due to nonexisting places.

  Examples of each of those subtypes are:

  Does bus 66 stop at Leif Tronstadsvei ?
  Which busses pass johnaaesveg?
  When does the next bus go to Gokk?

  2.0% of the Norwegian and 2.2% of the English questions was placed in this category.

- **Unknown words**

  In this category we have placed all questions that are not obviously (to the user) outside the domain, but that contains words that are unknown to Bustuc.

  These problems can of course only be addressed by declaring more words in the dictionary and semantic knowledge base.

  Examples are:
  is a bus a trailer?
  I should like to go to Dragvoll just after breakfast?
  Hvor lang er den lengste bussen?
  Er det kommet nye ruter for 44 fra Trondheim?

  2.0% of the Norwegian and 1.4% of the English questions was placed in this category.

- **Misspelled words**

  In this category, words that would have been understood if they were spelled correctly are placed.

  This errors can be completely removed by letting TUC spell correct all words instead of just names.

  Examples are

  Wich bus passes Samfundet to Blaklihøgda between four and five?
  How can i get from Nardo to Risvollan bu bus?
  Gåe det en buss fra nth ca. kl 1500 ?
  N}r g}r bussen fra Valentinlyst til Sentrum?

2.9% of the Norwegian and 3.4% of the English questions was placed in this category.

- **Incomplete sentences**

  The questions in this category are all the incomplete sentences not understood by TUC that are not thrash input.

  many of these can be addressed by making the parser robust.

  When does the first bus from Væretrøa to Lerkendal?
  From Nardo to Blaki at 1430?
  Første buss Halseth til Sentrum?
  Sentrum til Moholt 0900 til 1000?

  4.0% of the Norwegian and 4.9% of the English questions was placed in this category.

- **Missing grammatics**

  The questions that are placed in this category are those that TUC is unable to parse even it know all the words and the sentences are complete.

  Some of these questions can be addressed by making the parser more robust, and some by making the grammar better.

  Examples are:

  When does the first bus leave Heimdal after 1600 for arrive to Klæbu?
  Where goes a bus from ?
  Når går neste 44 buss fra trondheim?
  Hva koster det å kjøre buss ?

  4.3% of the Norwegian and 4.4% of the English questions was placed in this category.

- **Missing semantics**

  In this category all questions that fails because of missing definitions in the semantic knowledge base are placed.

  These can of course be addressed by doing more work on the semantics for the application.

  Examples are:

  Hvilken buss må jeg ta fra NTH for å være på Dragvoll før kl 13 ?
  Is there a bus connection from Dragvoll to Rotvoll
  When does the first bus between Dragvoll and Sentrum leave on sunday?
  Is there a bus stop called sentrum?

  1.1% of the Norwegian and 1.5% of the English questions was placed in this category.

- **Missing pragmatics**

  This are the questions that the Bustrans rule base either interprets incorrectly even TUC makes correct TQL for them or does not interpret at all.

These can be addressed by refining the Bustrans rule base and making more logic for the domain.

Examples are:

Når går neste buss 5 fra Nardo etter kl 1300?
Hvor mange busser går fra Lade i dag?
Which stations are between Festningen and Grensen?

2.1% of the Norwegian and 1.5% of the English questions was placed in this category.

- **Obviously outside the domain**

  In this category we place all questions that everybody must understand is outside the domain.

  These can of course not be solved generally without having a general artificial intelligence.

  Examples are:

  What is the meaning of life, universe and everything?
  What is your interface between cgi and Prolog, Id like to use it myself ?
  Døde Napoleon den 7. i 1911?
  Kan du sette på potetene når du kommer hjem?
  Hva heter den søte blondinen som pleier å kjøre rute 52 fra Steinan om morgenen?

  1.8% of the Norwegian and 3.1% of the English questions was placed in this category.

- **Multiple sentences**

  In this category all questions containing multiple sentences are placed.

  These can only be addressed by extending TUC to understand discourse, including ellipsis.

  Examples are:
  Hva heter bussholderplassen ved MAX senteret (like ved lerkendal).... og hvilke busser går dit?
  Hvordan kan jeg komme fra NTH til Rosendal slik at jeg er på Rosendal før klokken 21, og tilbake igjen fra Rosendal etter klokken 23?
  I want to go from Nardo to Heimdal. When does the next bus leave?

  1.8% of the Norwegian and 3.1% of the English questions was placed in this category.