Tore Bruland

Building World Event Representations From Linguistic Representations

Thesis for the degree of Philosophiae Doctor

Trondheim, November 2013

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Imformation Science

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

ISBN 978-82-471-4291-2 (printed ver.)
ISBN 978-82-471-4292-9 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2013:97

Printed by NTNU-trykk

For Kari Marie Bruland (1937-2007)

Abstract

The goal of the research presented here is to build a natural language processing system for our future natural language applications. We believe that real applications will move our research in Computational Linguistics and Artificial Intelligence forward, and we prefer applications that are attractive to a large number of users. The engine in this system is the wide-coverage grammar NorSource, and the research topic is to build a prototype of the natural language processing system.

The first part of our research topic is to build a pipeline for our grammar. We present two versions of the pipeline. The first pipeline has unmapped predicates that contain object and event references at the discourse level. Each object and event has an unique identifier in the discourse, and the pipeline performs a simple pronoun resolution. The second pipeline is a pipeline with predicates mapped to a selected domain, and the discourse contains object and event references at the world domain level. The world references are the result of an interpretation with a logic model or a selection of a previous stored situation. The domain ontology predicates are mapped from the underspecified predicates. Both pipelines have a demonstrator and the specified world pipeline's domain is the classical Box World from Artificial Intelligence.

The second part of our research topic is to fill the gap between underspecified predicates and domain specific predicates. The meaning representation produced by NorSource is Minimal Recursion Semantics (MRS), and this representation has underspecified quantifier scope and word senses. We have algorithms for solving the underspecified quantifier scope, but we don't have algorithms for mapping underspecified predicates to domain specific predicates. The starting point is Vendler's Aktionsart types. The types have a structure and Moen and Steedman showed that a verb argument can coerce the verb from one Aktionsart type to another. Some verbs have culminating states that are not a part of the surface structure of the sentence. Some verbs have additional structure like sub events and causal relations between sub events. Structure of a verb and coercion of Aktionsart types are outside the scope of an MRS, so we want to incorporate some of these notions into our mapping between underspecified and specified predicates. We use a domain ontology and a mapping algorithm. The ontology contains a collection of concepts with a "is-a" hierarchy, "has" relations, and "use" relation. The ontology also contains templates that use the hierarchy and the relations in order to implement constraints on a predicate and its arguments. The ontology contains complex domain objects that generate possible structures, time points, roles and states. We use the Change Location domain to demonstrate how the mapping algorithm works.

We have created a natural language processing system prototype and we have filled the gap between underspecified predicates and domain specific predicates. We can transform our MRSs into expressions in First-Order Logic and reason with them. The tools from the DELPH-IN consortium creates "deep" grammars that offers the meaning representation MRS, and this means that our work can be used by other grammars and languages.

Acknowledgments

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfillment of the requirements for the degree *Philosophiae doctor* (PhD). This work has been completed at the Department of Computer and Information Science at Norwegian University of Science and Technology, Trondheim.

I will thank my supervisors for their effort, Associate professor Tore Amble for endless discussion about Natural Language Understanding in the public transport domain, and to Professor Jon Atle Gulla for administering the last leg of my journey. A special thanks to Professor Lars Hellan, the creator of NorSource grammar, for making NorSource a part of my work. I will also thank Professor Lars Hellan and Professor Agnar Aamodt for their comments on this text.

To my fellow PhD students, near and far, for discussions and company: Rune Sætre, Trond Schjelderup Hegdahl, Martin Thorsen Ranang, and a special thanks to Anders Kofod-Petersen for his LaTeX PhD *ready to go* framework adjusted to the B5 size and pdf format.

Tore Bruland
Trondheim, July 2013.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Part I

Introduction

Chapter 1

Introduction

The goal of the research presented here is to build a natural language processing (NLP) system for our future natural language applications. We believe that real applications will move our research in Computational Linguistics and Artificial Intelligence forward, and that they will be a good measure for the coverage of our grammar and the quality of our reasoning abilities. We prefer applications that are attractive to a large number of users, because attractive applications generate a large amount of data that can be very useful for our NLP system. In order to build applications like dialogue systems or question-answering systems, we need to create a natural language processing system that can connect the linguistic representations to a domain ontology and then reason with the domain ontology elements. The engine in this system is a wide-coverage grammar, and we need a pipeline to connect our linguistic information, domain specific information and reasoning information. The NLP system will contain tools for semi-automatic and perhaps automatic acquisition of information for use in Computational Linguistics and Artificial Intelligence. The acquisition process will extract relevant information from texts.

Our wide-coverage grammar is NorSource[1] [8; 49; 50; 53; 9; 51]. The goal of the NorSource grammar is to be complete in an underspecified way for written Norwegian. NorSource is a Head-Driven Phrase Structure Grammar (HPSG) [84; 86], developed and maintained with the Linguistic Knowledge Builder (LKB) tool [27], and based on the HPSG Grammar Matrix, which is a starter kit for developing HPSG grammars [10; 11]. The consortium sustaining this architecture is the DELPH-IN Consortium[2].

Our goal is to build a pipeline for our natural language processing system.

[1] `http://typecraft.org/tc2wiki/Norwegian_HPSG_grammar_NorSource`
[2] `http://www.delph-in.net`

The first step in our pipeline is the parsing step, where we use NorSource, but we need to build the remaining steps of the pipeline. The meaning representation produced by NorSource is Minimal Recursion Semantics (MRS) [29], and this representation has underspecified quantifier scope and word senses[3]. We have algorithms for solving the underspecified quantifier scope, but we don't have algorithms for the underspecified senses.

For example, in the sentence "the king of pop met the king of Norway", the PPs can be expressed in an underspecified way with the relation *king_of(*arg_1*,*arg_2*)*, but if we want a more detailed description closer to the world we want to use two senses of "king": the relations *male_monarch(*x_1*,*x_2*)* and *best_in_field(*x_3*,*x_4*)*, where x_1 and x_3 are persons, x_2 is a country, and x_4 is a field.

We need to fill the gap between the underspecified representation and the description with more world knowledge. The Linguistic Representation is the underspecified meaning representation MRS, and the World Event Representation is a collection of concepts describing a selected part of the world. This is also reflected in the title of this work ("Building World Event Representations From Linguistic Representations"). The selected parts of the world are the domain of "Change Location" (movement of objects in the world) and the domain "Box World" (a small world from Artificial Intelligence where boxes can be moved around).

1.1 My Own Background of Research

My involvement in the field leading up to the present project can be divided into three phases.

1.1.1 The Brage Project

The Brage project, financed by The Research Council of Norway (Norsk forskningsråd), was a collaboration between the Norwegian University of Science and Technology (Department of Electronics and Telecommunications, Department of Computer and Information Science, and Department of Language and Communication Studies), Telenor Research and Development, and Sintef Information and Communication Technology. The research topic was Speech Technology, and the main goal of the project was to improve the speech recognition technology for Norwegian. Brage produced a number of publications[4],

[3] all senses that are relevant in a linguistic way are not underspecified

[4] `http://www.iet.ntnu.no/projects/brage/archive.php?file=Publikasjoner.html`

an improved speech recognizer, a Wizard-of-Oz experiment, and a demonstrator [55]. My role in the project was to create a dialogue manager for Norwegian spontaneous speech. The development of the spontaneous speech grammar was assigned to a fellow PhD student. The Buster dialogue system [104], based on BusTUC [19; 5], was used to limit the scope of the speech recognizer. Buster answers questions about bus departures in Trondheim and questions about persons from the telephone book at NTNU.

My first research topic was therefore dialogue management of spontaneous Norwegian speech. However, there do not exist many resources for this purpose for Norwegian, and we had no grammar for spontaneous speech. Unfortunately, it takes a long time to develop a usable grammar; a fair estimate is from 5 to 10 years. The wizard-of-Oz data was analyzed by the student in charge of the grammar [20]. Together, we left the research topic of spontaneous speech. My research status so far was therefore a literature study in the field of dialogue systems and dialogue management [70; 69; 97; 41; 2].

1.1.2 A Dialogue Manager Using BusTUC

My research focus moved from spontaneous speech to written text. I decided to start with the dialogue system Buster that is built over the BusTuc system. The Buster system is already a dialogue system for bus questions, so my domain focus was set to sentences about movement in general.

The BusTUC system[5] answers questions about bus departures in the city of Trondheim. The system performs a deep semantic analysis and contains references to a bus domain ontology. The system is a commercial system and it has been in production since 1998. In 2007, BusTUC had 800.000 requests from its web service and 100.000 requests from its SMS-service. Trondheim is a city with about 175.000 people.

BusTUC's parse strategy is to use the first valid occurrence of the input sentence. This is obtained by ordering the grammar rules with the longest sentences first and by placing a few strategic cuts. This way of operating is no problem within the two domains (time table questions and telephone book questions), but we need to choose among ambiguities if we want more coverage for the grammar.

My last attempt with this research topic was to move the BusTUC grammar to a chart parser. The BusTUC grammar has been developed from the principles of ExtraPosition Grammars [83], and combined with ideas from Combinatory Categorial Grammars [95]. The grammar is implemented with Prolog's

[5] `https://www.atb.no/?lang=en_GB`

Definite Clause Grammar framework. The transformation to a chart grammar failed due too many technical obstacles. I couldn't spend more time preparing for parsing of text.

My research status so far was therefore an Event Calculus approach and an analysis of a sample of input sentences from BusTUC. Lambalgen and Hamm [62] created an Event Calculus program for the Aktionsart type accomplishment. I used the same idea and I created "The Extended Event Nucleus Model" based on Moens' and Steedman's Event nucleus model [72]. The details are found in the Appendix. BusTUC received 2.517.047 sentences between 31 October 2010 and 1 March 2013. The sentences were analyzed and the goal was to establish a typology of the sentences BusTUC receives.

1.1.3 Use the Wide-Coverage Grammar NorSource

The focus moved from BusTUC to the wide-coverage grammar NorSource, where I participated in connection with a grammar tutoring application[6] development project [53]. The focus also moved from dialogue management to mapping from an underspecified representation to a description of the world. The grammar needed some adaption before it could be used in a pipeline. Such as creating workable and valid MRSs, compiling the grammar for faster performance, creating a parse ranker, and fine tuning of the grammar. The latter in cooperation with the NorSource grammar maintainers. The software created in the grammar tutoring application is also used in our pipeline. I have also participated in a project for creating a multilingual Database of Verb Valence [52]. My contribution was to create the database and a web application demo.[7]

A brief summary of the results from my current research phase:

- The publication "Pre-Processing MRSes" [21] in The 10th International Conference on Computational Semantics (IWSC 2013).
- A pipeline for underspecified MRSs with Pronoun Resolution is developed. A batch demonstrator is available.
- A pipeline for specified (world domain) MRSs is developed. A demonstrator with the Box World domain is available on the web.
- A mapping algorithm from underspecified predicates to domain specific predicates is developed.
- A domain ontology for the Change Location domain is developed.

[6] `http://regdili.idi.ntnu.no:8080/studentAce/parse`
[7] `http://regdili.idi.ntnu.no:8080/multilanguage_valence_demo/multivalence`

A brief summary of the results from our previous research phase:

- A number of sentences from BusTUC is generalized and a table of the most frequent types is presented
- A model of the Aktonsart type accomplishment called the Expanded Nucleus Model is developed. The model has sub events for the process and sub states for the state.

1.2 Research Method, Questions and Thesis Outline

Computer Science has strong bonds to Mathematics and Electrical Engineering, and there is a new bond between the physical sciences and computer science in massive high-speed computations [33]. A computer scientist should master several paradigms [103]:

- the *empirical tradition* where we collect a large amount of data about a phenomena, analyze the data, develop models (theory), and then classify the data
- the *mathematical tradition* where we study algorithms and information structures
- the *engineering tradition* where we manage the creation of complex software systems

In this thesis we use the mathematical and the engineering tradition. At this point we don't have empirical results, but in our future research we will be able to compare different algorithms in our NLP system.

Our research topic consists of two parts. Part one is to map the gap between underspecified MRS predicates and elements of a domain ontology. Part two is building a pipeline for our grammar and our mapping process.

In part one, we know that the world has more details than our linguistic representation. The linguistic representation has elements from the morphological rules, the syntax and the lexical semantics. We start with the Aktionsart types. Some of these types change with verb arguments and some verbs have an inner structure with different arguments.

In part two, we want the grammar to perform well with as few resources as possible, and the grammar should have an interface with other software components. This means we leave the development tool (LKB) and compile our grammar into an environment with fast algorithms. We have two options,

PET[8] tool [24] and ACE[9]. In order to collect performance measures for the grammar, the competence and performance laboratory or [incr tsdb()] [80; 63] is used. The tool is created by CSLI Stanford and the German Research Center for Artificial Intelligence (DFKI), Saarbrücken. After the parsing of a sentence, we want to read the most relevant MRSs into data structures and process them. The process has two main tasks: the underspecified quantifier scoping and the predicate gap. The first task is solved by the Swiss Army Knife of Underspesification (Utool) [60]. The second task is the other part of our research focus. We also want to see if we can convert an MRS into a First-Order Logic expression, and reason with the expression with existing First-Order Logic tools [12].

To summarize our research topic, our purpose is to build a natural language processing system where the first component is the wide-coverage[10] grammar NorSource. NorSource produces MRSs with underspecified scope and underspecified senses. We want to connect the undespecified senses from an MRS to a domain ontology from a selected domain, and then reason with the ontology elements. The reasoning process can be a question-answering system or a dialogue system. We split the research topic into two parts. Part one is to map the underspecified predicates in the MRS to a domain ontology. In part one, we have the following research questions:

1 What are suitable formal representations of linguistic events and world events?

2 How can linguistic events be transformed into world events for later reasoning and analysis?

In part two, we place the mapping procedure in a pipeline where we reason with the domain ontology elements.

[8] `http://pet.opendfki.de/`
[9] `http://moin.delph-in.net/AceTop`
[10] open domain

In the remaining part of Part I, we define the term Ontology and Knowledge representation, then we present a chapter for State-Of-the-Art.

Part II contains the design of the NLP system. In this part, we present a chapter on Minimal Recursion Semantics, a chapter with algorithms for two pipeline versions, and a chapter with an algorithm for mapping underspecified predicates to world domain predicates.

Part III contains a description of the results. First, two pipeline implementations are presented, one with underspesified predicates and one from the BoxWorld domain. Second, we present the design of the Change Location domain ontology, and we show how the mapping algorithm works together with the domain ontology.

Part IV contains a chapter with a section for discussion, research questions, conclusion and future work.

Chapter 2

Definitions

We define the terms Ontology and Knowledge Representation. The definition of Ontology is taken from Computer Science. The authors of the definition of Knowledge Representation point out that knowledge representation is more than just a data structure.

2.1 Ontology

The term ontology can at least be found in Philosophy and in Computer Science (Information Science). We use the term from Computer Science and Tom Gruber's ontology definition [48] is:

> A body of formally represented knowledge is based on a *conceptualization*: the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold them [46]. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. ... An *ontology* is an explicit specification of a conceptualization.

Sowa's definition [94, Page 492] is:

> The subject of *ontology* is the study of the *categories* of things that exists or may exist in some domain. The product of such a study, called *an ontology*, is a catalog of the types of things that are assumed to exist in a domain of interest $\mathscr{D}$ from the perspective of a person who uses a language $\mathscr{L}$ for the purpose of talking about $\mathscr{D}$. The types in the ontology represents the *predicates*, *word senses*, or *concepts and relations types* of the language $\mathscr{L}$ when used to discuss topics in the domain $\mathscr{D}$.

2.2 Knowledge Representation

Davis, Shrobe and Szolovits defined Knowledge Representation (KR) through five basic principles [31].

A knowledge representation is a surrogate. All our definitions are only a surrogate for things in the real world. We need a model of the world with focus on the important parts. Such a model is often an expert's point of view. The part of the world that is to be modeled can sometimes be crowded by details and sometimes be unknown. Either way, the relevant parts in the real world are modeled.

A knowledge representation is a set of ontological commitments. We want to find the relevant terms that are used in our subset of the world. Our model needs precise definitions of things that exist in the world and how these things are connected; things like objects, relations, features, events, etc..

A knowledge representation is the fragmentary theory of intelligent reasoning. The theory is expressed by three parts:

- the representations fundamental conception of intelligent reasoning. Different reasoning techniques for intelligent reasoning exist. These are mathematic logic (formal calculation), psychology (human behavior), biology (stimulus-response), probability theory (casual networks), and economics (rational agents). Each technique has different strengths in different areas of reasoning, but no technique covers all areas.

- The set of inferences that the representation sanctions. We need to know which inferences we can do.

- The set of inferences that the representation recommends. We need a guidance to find the most intelligent inference.

A knowledge representation is a medium for efficient computing. The algorithm that uses the shortest amount of time and the minimum set of resources is the most suitable algorithm to solve the problem. The algorithm is a result of knowing how to reason as an expert in the domain.

A knowledge representation is a medium of human expression. The representation is the result of the collaboration between the knowledge engineers that implements the representation and the domain experts. Both are users of the representation.

2.3 Summary

If we use the definition of Knowledge Representation in our research, we have a representation used by the designers and the users of the linguistic application. The MRS predicates are linked to a domain ontology, and the domain contains reasoning rules. If we want to create a gold standard for the linguistic application, we need a representation for every definition and decision in our application. The inference in the application can be expressed with a chain of explanations. This way the maintainers of the grammar, the maintainers of the knowledge representation, and the users of the application can communicate about the selected domain. We have seen golden standards for words connected to a domain, but not for the entire application.

Chapter 3

State-Of-The-Art

We have an example from Artificial Intelligence and we searched for meaning representation from linguistic frameworks.

3.1 Watson

Watson [37] is a computer system that was developed at IBM research and it was designed to compete against humans in the American TV quiz show Jeopardy. Jeopardy is rendered to be an extraordinary demanding task. Watson is based on the DeepQa architecture and it was developed over three years with about 20 researchers. The motivation for the system was the need for computer systems that deeply analyze the breadth of relevant content to more precisely answer and justify answers to user's natural language questions. It is an AI challenge; a synthesis of information retrieval, natural language processing, knowledge representation and reasoning, machine learning and computer-human interfaces. Jeopardy is a competition where confidence, precision and answering speed are critical important (about 3 seconds). The aim of the project was to reach the level of human champions. This means that the system must answer 70% of the questions with 80% precision in 3 seconds or less. The challenge was expected to advance the research in QA technologies (parsing, question classification, question decomposition, automatic source acquisition and evaluation, entity and relation detection, logic form generation, KR and reasoning). With a large number of components in the system, the researchers needed to compute a confidence for each component. A method that was useful was Factoid questions – questions whose answers are based on factual information about one or more individual entities. Some complex clues contain multiple facts (decompose and find the common answer). The

researchers analyzed 20000 questions and extracted the Lexical answer type (LAT). They created a general-purpose, reusable NLP and KR and reasoning technology that can exploit as-is natural language resources as-is structured knowledge rather than to construct task-specific knowledge resources.

Before they started the work on Watson, the QA system Piquant was developed. This was a four person team that worked for six years. The system was placed top three and top five in the Text Retrieval Conference (TREC). Piquant allows use of web and the contenders prepare for the question before the event. However, the system performed only with 13% precision in Jeopardy. The team needed a better approach. A solution was drafted together with Carnegie Mellon University [38]. Together, the Open Advancement of Question answering initiative was started. The goal was to replicate and reuse research results and to rapidly advance the state-of-the-art in QA. DeepQA is a massively parallel probabilistic evidence-based architecture that use 100 different techniques for analyzing natural language, identifying sources, finding and generating hypothesis, finding and scoring evidence, and merging and ranking hypothesis.

At the end of 2008 the performance was about 70% precision at 70% attempted over 12,000 questions, but it took over two hours to answer one question. The system was moved to a massively parallel high-performance platform. In fall of 2010, Watson answered more than 85% of the questions in 5 seconds or less. 70% of the questions was attempted.

3.2 Step 2008

The aim of the shared task Step 2008 [14] was primarily to compare semantic representations. The representations where generated by state-of-the-art NLP systems parsing text. All participants had the same goal: computing semantic representations for text. Each participant of Step 2008 parsed the same text, and each group presented a paper describing their NLP system's performance on this text. A brief description of the systems components and resources follows:

- the BLUE system by Clark and Harrison [25]
- the Boxer system by Bos [15]
- the GETARUNS system by Delmonte [32]
- the LXGram system by Branco and Costa [17]

- the OntoSem system by Nierenburg et al. [78]
- the TextCap system by Callaway [23]
- the Trips system by Allen et al. [3]

BLUE, Boeing Language Understanding Engine, uses the broad coverage and domain general parser SAPIR. SAPIR's meaning representation is called Logical Form (LF). LF is transformed into first-order syntax with a module called the logic form generator. Subsequent processing modules are word sense disambiguation, semantic role labeling, co-reference resolution, metonymy resolution, and structural transformation. BLUE can use one of the two Ontologies: WordNet [36] and University of Texas at Austin's Component Library (CLib) [6]. WordNet was used in Step 2008.

BOXER is an open domain software component for semantic analysis of text. The text is parsed with the C&C tools and it generates syntax trees. The Boxer system has created lambda-DRSs for almost all categories from the C&C parser. The Boxer system creates the meaning representation Discourse Representation Structure (DRS). The DRS can be transformed into First-Order Logic. The logic can be processed with off-the-shelf provers and model builders. The system uses the roles from VerbNet [57].

GETARUNS is a closed domain system. The meaning representation is called Situation Semantics. The system is divided into three parts: the lower module, the middle module and the higher module. The lower module uses the LFG theoretical framework for parsing. The middle module is used for semantic interpretation and discourse model construction. The higher model is used for reasoning and generation. The system uses COMLEX [47], WordNet, EuroWordNet [101] and CoreLex[22].

LXGram is a general purpose HPSG grammar developed with the LKB tool. The meaning representation is Minimal Recursion Semantics. The parser is designed for Portuguese and the text was translated into Portuguese before parsing. The system stores the parsing results in the [incr tsdb()] tool. The preferred analysis was selected by members of the group.

OntoSem, Ontological Semantics theory, use a meaning representation called Text Meaning Representation (TMR). The representation uses elements from the OntoSem ontology (multiple inheritance hierarchical collection of frames that contains richly interconnected descriptions of types of OBJECTs, EVENTs and PROPERTies). The system uses the Stanford parser (the process of incorporating the parser was almost complete). The TMR can be in a basic mode or in a extended mode. The semantic analysis (word-sense disambiguation, semantic dependency determination, aspect, time) produces basic

TMRs. The discourse analysis (speaker attitude, reference resolution, semantic ellipsis, discourse relations, metonymy, etc) produces extended TMRs. The semantic analysis was used in Step 2008. The system uses the OntoSem lexicon (written in a Lexical-Functional Grammar formalism using a Lisp-compatible format), Onomasticon (proper names), fact repository and micro theories. A gold standard was created with the tool DEKADE (Development, Evaluation, Knowledge Acquisition and Demonstration Environment).

TextCap produces a meaning representation called semantic triplets (Concept, Relation, Concept). The system uses the Charniak parser that outputs syntactic parses. The system applies syntax-based discourse rules to the syntactic parses. Then grammatical roles and syntactic features are added. A simple anaphora resolution algorithm is used and senses are grounded in WordNet.

TRIPS is a broad coverage system that creates a meaning representation called Logic Form. The system use statistical methods to guide the deep parser. The chart is filled with data form a part-of-speech tagger and named entity recognizer. The Trips grammar is a lexicalized context-free grammar augmented with feature structures and feature unification (X-bar theory). The system uses the Trips ontology. The system uses VerbNet, the Extended version of EuroWordNet and Comlex. The group created a gold standard for the text with a graphical editor.

3.3 Other Related Work

Frank et al. [42] describe a question answering system. They use the Heart of Gold (HoG) NLP architecture[1] [90; 89]. Their domains are the Nobel prize domain and the Language Technology World information portal LT World.[2] They use Robust Minimal Recursion Semantics (RMRS) [28], a version of MRS, and they transform their RMRSs into frames[3] using rewriting rules. The questions are answered from domain-specific concepts and properties stored in a relational database.

Other examples of world event models for the Change Location domain are: the Motion frame in FrameNet,[4] the VerbNet class run-51.3.2, and the "Travel by train" plan, from [1, page 481].

Schäfer et. al. [88] are extracting factual relations from the ACL Anthol-

[1] A middleware architecture for language processing components. Located at `http://heartofgold.dfki.de/`

[2] `http://www.lt-world.org`

[3] theory of Frame Semantics, as pursued in the FrameNet project

[4] `https://framenet.icsi.berkeley.edu/fndrupal/home`

ogy[5]. They are using the HoG NLP architecture with the tools: the generic named entity recognizer SProUT [35], part-of-speech tagger TNT [18], the ERG grammar and OntoNERdIE [87] (a tool that can map ontologies to resources for named entity recognition and resources for information extraction). They are using the ontology LT World web portal.

Schlangen describes a dialogue system that uses fragments [91]. They explain how they implement fragments in ERG for their solution.

3.4 Summary

WATSON is a system with a large number of theories in a massively parallel probabilistic evidence-based architecture. It is difficult for a PhD student to compare his research to the work of a experienced research group with large resources. However, the PhD student can seek a collaboration with the group.

In Step 2008, we have a description of several linguistic frameworks, but not a comparison of the representations. The systems use different parsing technologies and representation formats and this makes it difficult to compare the representations and reach a consensus on what linguistic theory to use and how to implement it. Bos [16] advocates to use the NLP systems as black-box units as a solution to this problem. If we want to create a linguistic application for a selected domain we also experience that there is no consensus on describing and reasoning in a domain of the world either. This makes it difficult to create linguistic applications independent of a linguistic framework.

We did not find any system that used MRS to reason within a domain.[6] The LOGON project used MRS in machine translation, but they did not solve the MRS or connect it to a domain.

[5]The ACL Anthology is a collection of scientific articles from international conferences, workshops and journals on computational linguistics and language technology

[6]Schlangen uses MRSs, but he does not describe how

Part II

Design

Chapter 4

Minimal Recursion Semantics

We start with a brief introduction to Minimal Recursion Semantics (MRS) [29] and then we explain how we use an MRS.

4.1 A Brief Introduction

The DELPH-IN[1] collaboration is a consortium that jointly works with deep linguistic processing of natural language based on the grammar formalism HPSG and the meaning representation MRS. MRS is a meaning representation that can be used for parsing (text to MRS) and generating (MRS to text). An MRS underspecifies the quantifier scopes and the word senses. The MRS contains elements that can be put together in different ways to form a logic expression. An MRS can be the source of many logic expressions, one for each scope. Instead of returning all the possible logic expressions, the compiling process is delayed or as we say the quantifier scope is underspecified. All the word senses that are relevant in a linguistic way are not underspecified, the rest are delayed or underspecified. We name an MRS with underspecified scope an unresolved MRS, and each logic expression an MRS can form for a resolved MRS. The parsing process gives one or more unresolved MRSs. An unresolved MRS can be the source of logical form equality ambiguities, while a resolved cannot.

The elements of an MRS are defined by the structure mrs(T,R,C), where T is the top handle, R is a bag of elementary predictions (EP), and C is a bag of constraints. An EP can be written as:

$handle{:}relation(arg_1 \ldots arg_n,\ sc\_arg_1 \ldots sc\_arg_m)$

[1] `http://www.delph-in.net/`

The arguments (*arg*) are a list of zero or more ordinary variable arguments of the relation. The arguments (*sc_arg*) are a list of zero or more scope arguments of the relation. A handle that is placed before the relation is called a label (root), and a handle that is in the list of the scope arguments is called a hole. A hole is a placeholder for a label. In this way, all the EPs in the MRS form a tree.

A variable for an event starts with the character *e*; a variable for an object starts with the character *x*. A variable can be augmented with extra information with the format: variable, type and value. The variable *e* can have the type TENSE with the value PRES, and the variable *x* can have the type NUM with the value SING.

Every dog chases some white cat (4.1)

The MRS from (4.1) has the following structure:

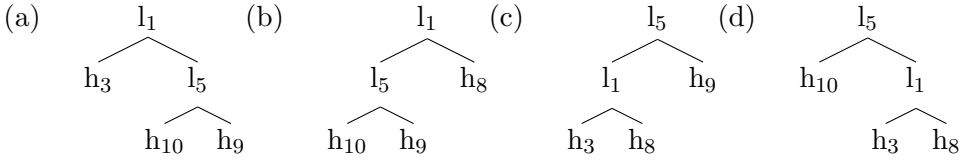
T h_0

C $h_{10} =_q h_7$

R { h_1:**every**(x_1,h_3,h_8), h_3:**dog**(x_1), h_7:**white**(x_2), h_7:**cat**(x_2), h_5:**some**(x_2,h_{10},h_9), h_4:**chase**(e_1,x_1,x_2) }

The top handle is the highest node in the tree (h_0 in our example). In the EP h_1:every(x_1,h_3,h_8), h_1 is a label and both h_3 and h_8 are holes. If there is more than one EP with the same label, they form an EP conjunction. The label h_7 ("white" and "cat") thus creates an EP conjunction. We say that the quantifier *every* outscopes the quantifier *some* when the label h_5 (some) is placed in one of the holes of label h_1 (every). In other words, the EP with *every* is higher in the tree than the EP with *some*. The handle constraint used is called "equality modulo quantifier", qeq constraint or just "=q". These constraints always relate a hole to a label. The hole is either directly filled-in by the label or it is floated-in (another EP is between the hole and the label). The only constraint in our example states that the hole in h_{10} (some) must be directly connected or filled-in by label h_7. An unresolved MRS has holes that are not filled with an existing label (unplugged holes).

The resolved MRS has additional constraints, definition 4 [29, page 293]. It must not introduce free variables (all variables must be bound by a quantifier). The resolved MRS must be constructible as a tree and the immediate outscoped constraint and the qeq constraints must be true. In our example, the EPs can be compiled together in four ways:



Since the hole h_3 immediately outscopes the label h_3 (dog) in tree (b), the tree is pruned away. The (c) tree is pruned away because it introduces a free x-type variable in h_4 (constraint causes $h_7 = h_8$). The (a) and (d) trees remain. The possible assignments between the labels and holes are (hole 3 in l_1 is bound):

- $h_0 = l_1$, $h_8 = l_5$, and $h_9 = l_4$
- $h_0 = l_5$, $h_9 = l_1$, and $h_8 = l_4$

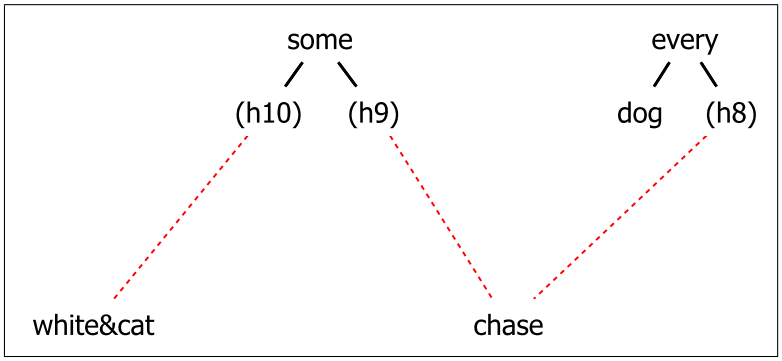


Figure 4.1: Unresolved MRS in Utool

The current MRS is read and displayed by the Swiss Army Knife of Underspesification (Utool) [60], and the unresolved MRS is shown in Figure 4.1. The resolved MRSs are shown in Figure 4.2 and Figure 4.3. The same resolved MRSs can be displayed as First-Order Logic expressions:

- $\exists(y)[white(y) \wedge cat(y) \wedge \forall(x)[dog(x) \rightarrow chase(x, y)]]$
- $\forall(x)[dog(x) \rightarrow \exists(y)[white(y) \wedge cat(y) \wedge chase(x, y)]]$

Both expressions can be expressed in plain text. The first:

> There exist a white cat and every dog chases that cat

The second:

> For every dog there exist a white cat which this dog chases

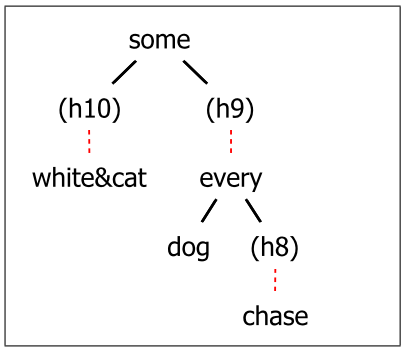


Figure 4.2: First Resolved MRS

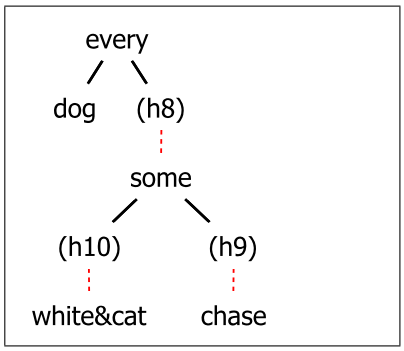


Figure 4.3: Second Resolved MRS

4.2 Resolved MRSs

We need an algorithm to transform an MRS from the unresolved state to the resolved state. The Linguistic Knowledge Builder (LKB) tool [27] contains one such algorithm and Utool contains another. The main differences between the two are that Utool implements stricter constraints and can perform redundancy elimination for logically equal readings. A resolved MRS has its holes filled with another EP's handle and the EPs form a tree according to the definition 4 by Copestake [29]. The algorithm's goal is to compile a valid resolved MRS in the shortest amount of time. A brute force approach will start to collapse when the number of EPs passes somewhere around eight. Niehren and Thater's approach was to translate the MRS into a normal dominance constraint [77]. The dominance constraints have existing algorithms that solve the problem[2] efficiently [4; 13]. After an MRS is translated, only a subset of the possible graphs are preferred. One solution to this problem is to add extra graph constraints. Koller classifies graphs as hypernormally connected [59] if his set of constraints are true. Even when a resolved MRS is found, it can be the source of a large number of logically equal ambiguities. The problem can be illustrated by an example from the Rondane Treebank [81], parsed by the English Resource Grammar [40]. About 2.4 trillion readings were reported from (4.2).

> Myrdal is the mountain terminus of the Flåm rail line which makes its way down the lovely Flåm valley to its sea-level terminus at Flåm. (Rondane 650). (4.2)

[2] transform an unresolved MRS to one or more resolved MRSs

Koller solved the logically equal ambiguity problem by transforming the dominance graph into a chart [58]. Koller and Thater presented an algorithm that removes logically equal resolved MRSs from this chart [59]. The algorithm reduced the ambiguities in (4.2) from 2.4 trillion readings to 1. The transformation algorithm from unresolved MRSs to resolved MRSs and the algorithm for elimination of logically equal readings are implemented in Utool.

4.3 Types of EPs

An MRS can contain different types of EPs. Some EPs denote an individual, others denote an event. These EPs can be classified with a part-of-speech type. In some MRSs, we can find extra EPs. For example "that car" can result in: h_{10}:_commsg_deict_rel(x_9) and h_{10}:_bil_n_rel(x_9). The relation "_commsg_deict_rel" can be used in our pragmatic processing. An other example is the predicate "first_position_prominent" that indicates that an adverb is placed first in the sentence. Other predicates can be part of a larger structure.

The boy, the girl and the dog smiled (4.3)

The sentence (4.3) is parsed with the English Resource Grammar (ERG) and

key	pos	pred	sense	EP
x_{10}	n	boy	1	h_{11}:_boy_n_1_rel(x_{10})
x_6	x	implicit_conj		h_{16}:implicit_conj_rel(x_6,x_{10},x_{14})
x_{20}	n	girl	1	h_{21}:_girl_n_1_rel(x_{20})
x_{14}	x	and_c		h_{22}:_and_c_rel(x_{14},x_{20},x_{23})
x_{23}	n	dog	1	h_{27}:_dog_n_1_rel(x_{23})
e_2	v	smile	1	h_{28}:_smile_v_1_rel(e_2,x_6,p_{29})

Table 4.1: The EPs of Example 2

it gives the EPs in Table 4.1 and the logical form is displayed in Figure 4.4. The "dog" and the "girl" are paired together in the predicate "_and_c_rel". This predicate and the "boy" are paired together in the predicate "implicit_conj_rel". The "implicit_conj_rel" structure (predicate) has the key x_6 and this key is used as a verb argument in e_2.

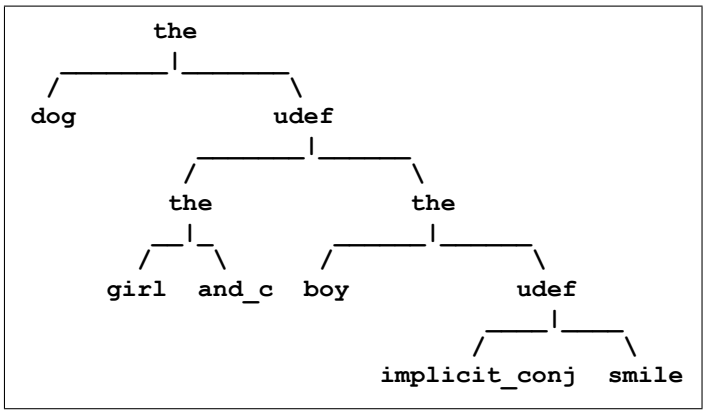


Figure 4.4: The Noun List

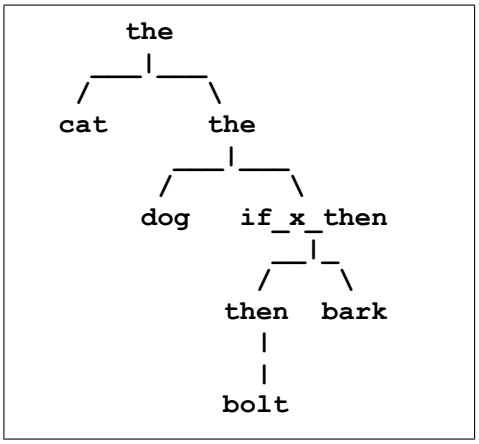


Figure 4.5: The IF Structure

If the dog barks then the cat bolts (4.4)

The sentence (4.4) is also parsed with ERG and it gives the EPs in Table 4.2

key	pos	pred	sense	EP
e_5	x	if_x_then		h_3:_if_x_then_rel(e_5,h_4,h_6)
	q	the		h_7:_the_q_rel(x_{10},h_9,h_8)
x_{10}	n	dog	1	h_{11}:_dog_n_1_rel(x_{10})
e_{13}	v	bark	1	h_{12}:_bark_v_1_rel(e_{13},x_{10})
e_{15}	a	then	1	h_{14}:_then_a_1_rel(e_{15},h_{16})
	q	the		h_{17}:_the_q_rel(x_{20},h_{19},h_{18})
x_{20}	n	cat	1	h_{21}:_cat_n_1_rel(x_{20})
e_2	v	bolt	1	h_{22}:_bolt_v_1_rel(e_2,x_{20},p_{23})

Table 4.2: The EPs of Example 3

and the logical form as displayed in Figure 4.5. The event e_5 is a structure holding two other events, one for the "if" condition node and one for the *then* conclusion node.

4.4 Quantifiers

There are a large number of quantifiers that can appear in an MRS compared to First-Order Predicate Logic. A small collection from NorSource:

- _pronoun_q_rel
- _def_q_rel
- _udef_q_rel
- _some_q_rel
- _univ_q_rel

We can think of quantifiers as sets and we can use the First-Order Predicate Logic quantifiers $\exists$ (one member) and $\forall$ (every member). The quantifiers used in an MRS can be defined as one of these types. For example, the quantifier "_def_q_rel(x_1)" can be defined as "_exist_q_rel(x_1)" and an additional feature on the variable x_1 can be created "feature(x_1,quantifier,_def_q_rel)". In this way, the MRS can be converted into an expression in First-Order Logic and

we can at the same time preserve the information given by the quantifier. Quantifiers like "_some_q_rel" can be solved with an extra cardinality function on the set following the theory of Generalized Quantifiers [44, page 230, table 7.1].

4.5 Interpretation

A resolved MRS with First-Order Logic quantifiers is an expression that can be true or false in a model of the world. An example of an interpretation in First-Order Predicate Logic is given in Appendix B.3. In natural language processing, statements can have an interpretation, but questions and commands do not, they only have a subset of the interpretation often called presupposition, and finding the presuppositions for a question or a command is not straight forward. When we have found our presuppositions, we have to find the members of our quantifier sets. For example, in the ACME Computer Company, 2 boys and 4 girls work in the first floor, and 6 boys and 8 girls work in second floor. If we say: "All the girls that work on the second floor wear pink overalls", the $\forall$ set must be filled with the 8 girls from the second floor and not the 12 girls from the Company. Another complicating factor is the model of the world. In natural language processing, we don't know the entire model. We actually use natural language to collect information. A possible approach to this problem is to classify predicates as "known" or "unknown", and we can only interpret predicates that are marked as "known". Anyway, this is outside the scope of our research focus.

4.6 Summary

An MRS contains predicates that we can connect to a domain, and an MRS can be solved with Utool. We found several types of predicates: predicates with part-of-speech, reasoning predicates like "_if_x_then_rel" from ERG, and predicates introduced by the linguist like "first_position_prominent" from NorSource. The predicates used in an MRS must be defined in order to be used. If we transform the quantifiers to the quantifiers in First-Order Predicate Logic, we can use the notion of truth from First-Order Predicate Logic. The alternative is to define an interpretation for every quantifier. However, the challenge is to fill the sets of the quantifiers with members from the selected domain. With an expression in First-Order Predicate Logic, we can use Theorem Provers and Model Finders as described by Bos and Blackburn [12].

Chapter 5

Pipelines

We called an MRS that underspecifies the quantifier scope for *unresolved*. An unresolved MRS is the source of one or more *resolved* MRSs, one for each set of quantifier scopes. We call an MRS that underspecifies the word senses for *unmapped* and when the word senses are specified for *mapped*. We can also distinguish three levels for the object and event references in an MRS: the *local level*, the *discourse level*, and the *world level*.

The *local level* contains MRSs with local object and event references. The MRSs that come from the parser have local object and event references. This level is used when we look at an MRS in isolation.

The *discourse level* contains MRSs with discourse object and event references. This level is used when we look at a series of MRSs. The numbering of objects and events start at 1 and continues to the end of the discourse. The objects are in the interval $[x_{1d}, x_{2d}, \ldots, x_{nd}]$ and the events are in the interval $[e_{1d}, e_{2d}, \ldots, e_{md}]$. Each object and event has a unique identifier in the discourse, and anaphora resolution is possible at this level.

The *world level* contains world references. The objects are in the interval $[x_{1w}, x_{2w}, \ldots, x_{nw}]$ and the events are in the interval $[e_{1w}, e_{2w}, \ldots, e_{mw}]$. Each object and event has a unique identifier in our model of the world. For example, if we read a document about the assassination of Kennedy, we want a unique identifier for the event and a unique identifier for the person. We can store the situation and reuse it later on if we are looking for similar situations. If we don't have a world model we can use the series from the underspecified discourse level. This is the case when we are collecting and storing information about a domain. The assigning of world references is a manual process.

The underspecified scopes and word senses together with the three levels of object and event references can be used together in different combinations.

The MRSs that come from the parser is unresolved, unmapped and the object and event references are at the local level. If we work with a discourse, we want to use object and event references at the discourse level, and we can use Utool to find one or more resolved MRSs. If we work with a selected domain and a discourse, we find the word senses and we use object and event references at the discourse level. For each resolved MRS we find the object and event references at the world level.

A linguistic application at the specified world level can be divided into a series of steps. The steps for a dialogue system can be: read a sentence from the speaker, parse the sentence, select an MRS, validate the MRS, calculate discourse references, execute anaphora resolution, map predicates, find world references, infer dialogue act, analyze dialogue act, execute the act, analyze the result, and send message back to speaker.

5.1 Initial Steps

Parsing a sentence with an HPSG grammar that is developed in the LKB framework, can produce a series of meaning representations. Each member in this series can be classified as wanted or unwanted. A wanted MRS is the MRS the linguist intended to create, while an unwanted MRS is an MRS that is the result of too few restrictions in some part of the grammar. Unfortunately, these types of MRSs come in a random order and they are not formally marked as wanted or unwanted. The wanted MRSs should further be placed at a point in the range: most probable to least probable. One solution to this classification problem is to create a parse ranker for our grammar. Our parse ranker is created with the LOGON tree tools ([tsdb++], Velldal's training and experimenting software [98], and Malulf's statistical software TADM[1] [67]). Before we can create the ranker, we need to collect a series of sentences that is typical for the domains we use in our pipeline. We store these in a [tsdb++] profile and then we parse and store the result in the same profile. Then we annotate the data. The grammar rules, syntax tree, and the parts of the MRS are annotated as wanted or unwanted. This culminates in the most wanted MRSs and we save these choices in our profile. This profile and other annotated profiles can be described as our treebank. Next, we run an experiment in order to find the best parameters for the training of our Maximum Entropy model. The experiment starts with a range of parameters and the process measures the effect of each combination. Finally, we select the best combination and train our Maximum Entropy model.

[1] Toolkit for Advanced Discriminative Modeling

We also create an equivalence file for the quantifiers (Utool). Utool uses this file to eliminate logically equivalent readings.

5.2 The Pipeline with Unmapped Predicates

Our goal at this point is to produce unmapped MRSs with object and event references at the discourse level. We assume that the grammar is loaded together with the Maximum Entropy model. In **line 1 to 3** in Algorithm 1, shown below, the equivalence file is loaded, the discourse and the pronoun model are initiated. In **line 4**, a sentence is read from the input. In **line 5**, we continue the loop until the input sentence is empty. The sentence is parsed in **line 6** and the result is placed in the table MrsTab. In **line 7** the top ranked MRS is placed in the variable theMrs. In **line 8**, Utool's solvable function is used

Algorithm 1 Unmapped Discourse Pipeline

```
1:  ef ← NorSource.equivalence_file                    ▷ Utool data
2:  ds ← Ø                                             ▷ discourse
3:  pm ← Ø                                             ▷ pronoun model
4:  sentence ← NorSource.readSentence
5:  while ( sentence.isNotEmpty ) do
6:      MrsTab[] ← NorSource.parse(sentence)
7:      theMrs ← MrsTab[1]                             ▷ top ranked MRS
8:      if ( Utool.isSolvable(theMrs) ) then
9:          plugTab[] ← Utool.solve(theMrs, ef)        ▷ data to solve MRS
10:         theMrs.add( plugTab )                      ▷ update MRS
11:         vMrs ← NorSource.validate(theMrs)          ▷ validate MRS
12:         if ( vMrs.accepted() ) then
13:             vMrs.discource_references()            ▷ change references
14:             vMrs ← pm.processMrs( vMrs )           ▷ solve and collect
15:             ds.add(vMrs)                           ▷ discourse element
16:         end if
17:     end if
18:     sentence ← NorSource.readSentence
19: end while
20: NorSource.process(ds)                              ▷ discourse
```

to check if the MRS is well-formed according to the MRS definitions[2] [29]. If the condition doesn't hold the algorithm jumps to line 18, else the algorithm

[2]Utool is stricter than the LKB software, see [43].

continues at line 9. The plugTab is filled (**line 9**) with plugging data from Utool that takes the MRS and the equivalence file as arguments. The plugging data are assignments between holes and labels in the MRS; there are a set of plugging data for each relevant scope. We don't solve the MRS at this point; we postpone that until we have a world model. The plugging data are stored in the MRS in **line 10**. In **line 11** we validate the MRS. Here we want to search our MRS for properties that can lead to problems. Our validating procedure contains a set of functions. The functions have names and they are true or false for an argument. We create variables or list of variables for each function that is positive. The functions are: *empty index*, *empty feature*, *empty reference*, *key conjunction*, and *argument EP conjunction*. An *empty index* exist when the index value refers to a variable that is not an EP's arg_0. An *empty feature* exist when a feature value refers to a variable that is not found in the EP's arguments. An *empty reference* exist when an argument refers to a variable that is not an EP's arg_0 and the variable is not in the set of feature values. A *key conjunction* exist when more than one EP in an MRS have the same arg_0 and they are not quantifiers. An *argument EP conjunction* exists when an argument contains a label that is an EP conjunction. In Table 5.1, the

EPs	Features
h_3:pred$_1$($arg_0(e_1)$,$arg_1(h_9)$)	e_1,feature$_1$,value$_1$
h_9:pred$_2$($arg_0(u_1)$,$arg_1(x_1)$,$arg_2(u_{10})$)	u_{12},feature$_2$,value$_2$
h_9:pred$_3$($arg_0(u_1)$,$arg_1(x_1)$,$arg_2(x_2)$,$arg_3(u_{15})$)	u_{15},feature$_3$,value$_3$
h_2:pred$_4$($arg_0(x_1)$)	u_{16},feature$_4$,value$_4$
h_4:pred$_5$($arg_0(x_2)$)	

Table 5.1: Validating EPs and Features

variable u_{12} is an *empty feature*. The variable u_{10} is an *empty reference*. The arg$_0$ of pred$_2$ and pred$_3$ form a *key conjunction*. The argument h_9 in arg$_1$ of pred$_1$ is an *argument EP conjunction*.

If the MRS isn't accepted the algorithm continues at line 18, else we continue at line 13. In **line 13**, we replace all local object and event references with the series from the discourse. Now, the objects and events have a unique reference. The "processMrs" function in **line 14** performs pronoun resolutions and it collect objects. The "processMrs" function takes an MRS as an argument. We perform a resolution when we find a pronoun. The algorithm is very simple: we search for candidates from the last element in the candidate table and a candidate is found when the pronoun and the candidate unify on the object properties gender, number, and person. We collect all objects that are

not pronouns and we select their predicate name, reference, CARG, together with the object properties gender, number, and person. The selected data are inserted last in the candidate table. In **line 15**, the MRS is added to the discourse. In **line 18**, a sentence is read from the input. In **line 20**, all the sentences are added to the discourse and we can process our discourse.

Algorithm 2 Mapped Discourse Pipeline

```
 1: ef ← NorSource.equivalence_file                          ▷ Utool data
 2: ds ← Ø                                                   ▷ discourse
 3: dm ← Ø                                                   ▷ pronoun model
 4: on ← NorSource.ontology                                  ▷ domain ontology
 5: mo ← NorSource.worldModel                                ▷ logic model
 6: dk ← NorSource.domainKnowledge                           ▷ detailed domain
 7: sentence ← NorSource.readSentence
 8: while ( sentence.isNotEmpty ) do
 9:     MrsTab[] ← NorSource.parse(sentence)
10:     theMrs ← MrsTab[1]                                   ▷ top ranked MRS
11:     if ( Utool.isSolvable(theMrs) ) then
12:         plugTab[] ← Utool.solve(theMrs, ef)              ▷ data to solve MRS
13:         theMrs.add( plugTab )                            ▷ update MRS
14:         vMrs ← NorSource.validate(theMrs)                ▷ validate MRS
15:         if ( vMrs.accepted() ) then
16:             vMrs.discource_references()                  ▷ change references
17:             vMrs ← pm.processMrs( vMrs )                 ▷ solve and collect
18:             mapMrs[] ← NorSource.map(vMrs, on)
19:             worldMrs[] ← NorSource.interpretation(mapMrs, mo)
20:             actMrs ← NorSource.dialogueManagement(worldMrs,dk)
21:             answer ← NorSource.execute(actMrs,dk,ds,mo)
22:             NorSource.writeAnswer(answer)
23:         end if
24:     end if
25:     sentence ← NorSource.readSentence
26: end while
```

5.3 The Pipeline with Mapped Predicates

In this algorithm we want to produce resolved and mapped MRSs with object and event references at the world domain level. We assume that the grammar is loaded together with the Maximum Entropy model. In **line 1 to 6** in

Algorithm 2, the equivalence file is loaded, the discourse and the pronoun model are initiated, the domain ontology is loaded, the world model is loaded, and the detailed domain knowledge is loaded. **Line 7 to 17** are the same as described in Algorithm 1. In **line 18**, we map the underspecified predicates to domain ontology predicates and the result is placed in the table mapMrs. The algorithm is presented in Chapter 6. In **line 19**, we transform the unsolved MRS to one or more solved MRS with the plugging data, then we interpret the MRS. Statements are completely interpreted, but commands and questions are interpreted by their presuppositions. The MRSs that are true First-Order Logic expressions are placed in the worldMRS table with world predicates and world references. In **line 20**, the simple dialogue manager finds the most probable dialogue act and places it in the variable actMrs. The dialogue manager uses the worldMrs table together with the detailed domain knowledge. The dialogue act is executed in **line 21**. The function uses the dialogue act, the detailed domain knowledge, the discourse and the world model. The world model can be changed in line 21. The result of the dialogue act is sent back to the user.

5.4 Summary

We solve the problem with wanted and unwanted MRSs with a parse ranker. We must build our ranker from statements that is typical from our selected domain. We check the quality of the MRS with Utool and our validating procedure. Utool checks if the MRS can be put together as logic expression (one set of pluggings for each scope), and our validating procedure checks if the MRS is in a format we can handle. If we transform the object and event references to a sequence of unique references, we can create a discourse. We have presented two algorithms for processing an MRS; one for the underspecified discourse level, and one the specified world level.

We want to implement a more suitable anaphora resolution algorithm in our pipeline, for example Hobbs's naive approach [71, chaper 4.5]. At this point we receive a syntax tree and an MRS from the parser, but the nodes of the syntax tree is not connected to the predicates (EPs) of the MRS. Words are showed as tokens in the syntax tree and words are showed as predicates in the MRS. Anaphora resolution within an MRS is already implemented in NorSource, so we have to adjust an extern algorithm to work on the resolutions that are outside the scope of NorSource.

In the pipeline with unmapped predicates we delayed the solving of an MRS, because we did not have a model. However, we can find a minimal model with a Model finder as shown by Blackburn and Bos [12]. This way we

can check if our discourse is satisfiable.

Chapter 6

Mapping From Linguistic Representations To World Events

Vendler [99; 100] grouped verbs into classes based on their temporal properties, according to duration and presence of a terminal point; For instance a verb with a terminal point is called telic (the verb culminates). Vendler's classes are also known by other terms such as: eventualities, situations, lexical aspect or Aktionsart[1]. We prefer the term Aktionsart. The defined verb classes are:

- state
- point
- process
- achievement
- accomplishment

Comrie [26] added the semelfactive or the point to the list. The *state* class describes an object with a property or an object in a state. The example in (6.1) states that Mary is in a pregnant state. The state has a time point for the beginning and one for the end. The t_n is within the duration of the state.

Mary is pregnant (6.1)

[1] German for 'kind of action'

The *point* class describes an event that happens at a singular time point (no duration). An example is shown in (6.2).

John blinked (6.2)

The verb class *process*, shown in (6.3), describes an event with duration.

Fred is watching the news (6.3)

The *achievement* class is an event that happens at a time point, and it culminates in a state. An example is (6.4). The culminated state is that Mary has a book.

Mary bought a book about Artificial Intelligence (6.4)

The *accomplishment*[2] class describes a process that culminates in a state. After the drinking process in (6.5), the orange juice is consumed and the glass is empty.

John drank a glass of orange juice (6.5)

Some basic regularities about the Aktionsart types and prepositions can be stated. For instance in English, if an event is modified with the preposition *for*, then the event is not telic (6.6). If an event is modified with the preposition *in*, then the event is telic (6.7).

John drank wine for an hour (6.6)

John drank a glass of orange juice in an hour (6.7)

The verb's connection to a class is not fixed, because a verb argument can move an event from one class into another. This phenomenon is called aspectual composition or coercion. Moens and Steedman [72; 73] introduced a state diagram for coercion, and a system of rules for carrying out such aspectual composition is called aspect calculus.

Some verbs that are telic can culminate in a state that is not that obvious. In (6.8), the poem is translated from one thing into another, but the state is hidden in knowledge about the world.

John translated the poem (6.8)

[2] An *achievement* is also called a culmination and an *accomplishment* is called a culminated process [72]

Sometimes it is not the case that a simple set of rules determines the Aktionsart type and the state. In (6.9), the type is state. In (6.10) the type is achievement, and in (6.11), the type is accomplishment.

The soup was cool (6.9)

The soup cooled (6.10)

John cooled the soup (6.11)

An event can be described as a structure. One such structure is the previously presented Aktionsart types. According to Levin and Hovav, the components of an event structure are the root (initial verb class), aspectual notions and causal notions [65]. A verb argument can change the root. Verbs and their arguments can be divided into a number of classes with different features [64].

An event can be split into a chain of sub-events where the sub-events have different arguments. In (6.12), the first sub-event is kick(John,Ball) and the second sub-event is move(Ball, Over_fence). The first sub-event causes the second sub-event to happen. John is not involved in the second sub-event.

John kicked the ball over the fence (6.12)

So far, we have illustrated that the Aktionsart types and some verbs have structure and that a culminating state can be located in the context and not in the sentence. We now present features of the domain "Change Location" , where we model the verbs and the verb arguments for objects that are moving from one location to another. In this domain we have concepts like vehicle, location, path, sub-path, driver, cargo, traveler, directions, etc.. These concepts are part of the language describing events in the domain.

mannen kjørte bil til Trondheim
the man drove a car to Trondheim (6.13)

The event in (6.13) can be described by event=change location, driver=man, vehicle=car, path.end=Trondheim.

mannen kjørte E6 til Trondheim
the man drove E6 to Trondheim (6.14)

The event in (6.14) can be described by event=change location, driver=man, vehicle=unknown, path.name=E6[3], path.end=Trondheim.

mannen kjørte til Trondheim
the man drove to Trondheim (6.15)

The event in (6.15) can be described by event=change location, driver=man, vehicle=unknown, path.end=Trondheim.

veien går til Trondheim
the road goes to Trondheim (6.16)

The event in (6.16) can be described by event=path description, path.name=road, path.end=Trondheim.

We want to incorporate structure and the domain language in our mapping algorithm. The concepts and the relations in this language is our domain ontology. Elements from the ontology are used to create restrictions between a verb and its arguments. Our claim is that the predicates in the MRS to some extent reflect the syntax tree. Our starting point is the principle of compositionality, and a definition from Partee [82] is:

The meaning of an expression is a function of the meanings of its parts and of the way they are syntactically combined.

There are more than one way to combine two constituents into a third. Cruse [30, chap. 4] has analyzed the problem and refers to the *additive mode* and the *interactive mode*. The additive mode is when the meanings are simply added together. The interactive mode is when the meaning of at least one constituent is radically modified. The interactive mode can be divided into *endocentric* and *exocentric*. The endocentric mode is when the resulting meaning is of the same type as one of the constituents. The exocentric mode is when the resulting meaning is of a different basic type from either of the constituents. Examples of endocentric combinations are:

- *Boolean combination.* An example is 'round table' where the class of round is intersected with the class of tables.

[3]The UN organ United Nations Economic Commission of Europe (UNECE) has named the road from Trelleborg in Sweden to Kirkenes in Norway as E6 (3120 km)

- *Relative descriptors.* An example is 'a small elephant' and the intersection here is more complex. The smallness is measured with an elephant scale.

- *Negational descriptor.* An example is 'a fake Ming vase'. The head is negated, but the referent is not given. The expression gives an indication of where to find the referent.

An example of an exocentric combination from [30, page 67] is the preposition *in* which denotes a relation, and a noun phrase such as *the box*, which denotes a thing, producing a prepositional phrase *in the box*, which denotes a location.

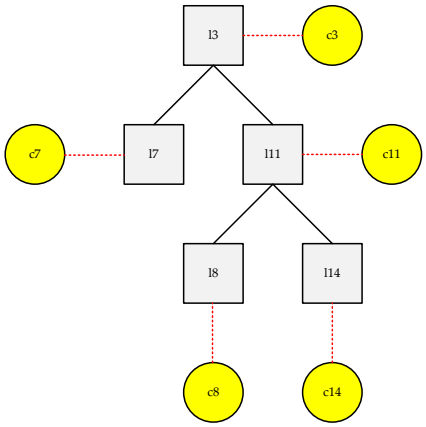


Figure 6.1: Tree Semantics

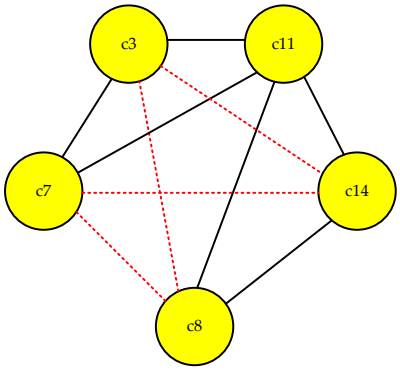


Figure 6.2: Node Relations

The predicates in an MRS can be constructed to form a tree. A simple example is an intransitive verb with a modifying preposition phrase. The verb is the root of the tree, the verb argument is a leaf, and the preposition phrase is a sub tree. This tree can be used to compile deeper semantics together. We use a simple depth-first search to compile the meaning of a node and its constituents. In Figure 6.1, the search starts at the top node and continue down to node *l7*; up to the root and down to node *l8*; up to node *l11* and down to node *l14*; then up to node *l11* and the root. At each node, a semantic representation is compiled with a template. The template applies constraints to the nodes and it defines a returning type for the parent node. The boxes represent the EPs and the circles represent the elements from the ontology. The question is: what happens when we compile the meaning of two words together?

We claim that the meaning can create a higher meaning structure where the words' meaning representations are parts, but not necessarily prominent units. Two nodes can be arguments of a third (verb), or the nodes are connected with a relation (preposition phrase). When all nodes are compiled together as shown in Figure 6.1, possible tests are performed between nodes. This is expressed as solid lines in Figure 6.2. A dotted line represents a possible relation that is not yet checked. These relations must be checked with a reasoning mechanism after all nodes are compiled together.

In the next three sections, we prepare the MRS, create a predicate tree and present the mapping algorithm.

6.1 Prepare The MRS

Before we create the predicate tree, we prepare our MRS. The predicates from the EPs are split into the fields predicate, part-of-speech and sense. The argu-

Row	EP
1a	h_{15}:_property_rel(arg$_0$(e_2),arg$_1$(h_{16}))
1b	h_{16}:_svart_a_rel(arg$_0$(u_{17}),arg$_1$(x_4))
2a	h_3:_buss_n_rel(x_4)
2b	h_3:_fra_p_rel(u_{10},x_4,x_8,u_9)
2c	h_{11}:named_rel(x_8,dragvoll)
3a	h_8:_kjøre_v_rel(e_2,x_4)
3b	h_8:_fra_p_rel(u_9,x_4,x_{11},u_{10})
3c	h_8:_til_p_rel(u_{16},x_4,x_{17},u_{18})
4a	h_8:_gå_v_rel(e_2,x_4)
4b	h_8:_ut_adv_rel(u_{10},x_4,u_{11},u_9)
4c	h_8:_av_p_rel(u_{10},x_4,x_{12})

Table 6.1: Parts of MRS, Pseudo Code

ments are divided into keys, predicate arguments and holes. The quantifiers are removed and the predicates called *named* are replaced by their CARG values[4].

[4]When we create our Named Entity Recognition program, we will attach a type to every name and use that type instead of the CARG

Algorithm 3 Create Predicate Tree Table Algorithm

```
1: procedure CREATE-PREDICATE-TREE(theMrs)
2:     pTree ← NorSource.replaceLabelWithKey(theMrs)
3:     pTree ← NorSource.fillModTab(pTree)
4:     return pTree
5: end procedure
```

The algorithm for creating the Predicate Tree Table is described in Algorithm 3. At **line 2** in the algorithm, the table is filled with the elements from the prepared MRS. In **line 3**, if an EP has an argument containing a label, the label of the argument is replaced with the key from the predicate that the labels refer to. An example is from the sentence "bussen fra Dragvoll er svart"/"the bus from Dragvoll is black". Here the predicate "_property_rel" has the argument $\text{ARG}_1(h_{16})$ (shown in row 1a in Table 6.1). The label of the argument is replaced by the key of EP with label h_{16} (shown in row 1a in Table 6.2). In **line 4**, the modTab is filled with modifying EPs like PPs, adverbs, and adjectives. The sentence "bussen fra Dragvoll er svart"/"the bus from Dragvoll is black" has an MRS where a noun is modified by a PP (shown in row 2a,2b and 2c in Table 6.1). In NorSource, the preposition for a movement is expressed with the subject of the verb, and the difference between a modification of a noun and a modification of a verb is the label of the EPs. A modification of a noun is expressed by an EP conjunction with the noun and the preposition, while a modification of a verb is expressed with an EP conjunction between the verb and the preposition. In our example, the noun's

Row	label	key	predicate	pos	sense	argTab	modTab
1a	h_{15}	e_2	property	v		u_{17}	
1b	h_{16}	u_{17}	svart	a		x_4	
2a	h_3	x_4	buss	n			u_{10}
2b	h_3	u_{10}	fra	p		x_4,x_8	
2c	h_{11}	x_8	dragvoll	na			
3a	h_8	e_2	kjøre	v		x_4	u_9,u_{16}
3b	h_8	u_9	fra	p		x_4,x_{11}	
3c	h_8	u_{16}	til	p		x_4,x_{17}	
4a	h_8	e_2	gå	v		x_4	u_{10}
4b	h_8	u_{10}	ut_av	adv_p		x_4,x_{12}	

Table 6.2: Parts of Predicate Tree Table, Pseudo Code

modTab is filled with the key from the PP (shown in row 2a in Table 6.2). The

sentence “bilen kjører fra Trondheim til Bergen”/“the car drives from Trondheim to Bergen” has an MRS where a VP is modified by two PPs. The EPs are shown in row 3a, 3b and 3c in Table 6.1. The verb’s modTab is filled with the keys from the two PPs (shown in row 3a in Table 6.2). A movement with a direction is expressed with a verb, preposition and an adverb. An example is “Frank gikk ut av bilen”/“Frank went out of the car”. The example has an MRS that contains the EPs in row 4a,4b and 4c in Table 6.1. Since the preposition and the adverb have the same key, the EPs are merged together into a adv_p type. The merged EP is modifying the verb (shown in row 4a in Table 6.2).

6.2 The Predicate Tree

After the Predicate Tree Table is created, the root of the tree must be found, and sometimes an MRS can contain more than one tree that share a common node. An example is “Gutten gikk til min bror”/“the boy went to my brother”. The Predicate Tree Table for a selected MRS is shown in Ta-

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	gutt	n			
h_8	e_2	gå	v		x_4	u_{11}
h_8	u_{11}	til	p		x_4,x_{10}	
h_{12}	x_{13}	pron	pr			
h_{17}	x_{10}	bror	n			
h_{17}	e_{18}	poss_c	v		x_{13},x_{10}	

Table 6.3: Search for Roots Example

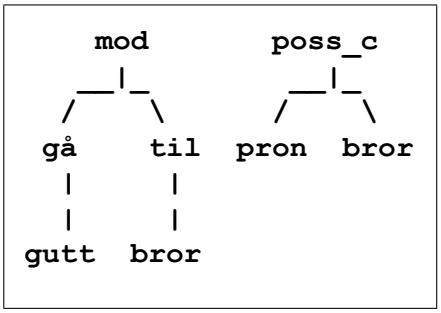


Figure 6.3: Search for Roots Example

ble 6.3 and the corresponding trees are shown in Figure 6.3. The rows in the Predicate Tree Table are used to create edges. A predicate without arguments and modifications does not create an edge. A predicate with arguments creates an edge between its key and each of its arguments[5]. A predicate with modifications creates an edge between the key and each modification. Our

Algorithm 4 Get Predicate Tree Algorithm

```
1: procedure GET-PREDICATE-TREES(topTab, ptt)
2:     treeTab[] ← Ø
3:     for ( i=1 to topTab.length ) do
4:         treeTab[i] ← GET-TREE-NODE( topTab[i], ptt )
5:     end for
6:     return treeTab
7: end procedure
```

example creates the following edges: edge(e_2,x_4), edge(e_2,u_{11}), edge(u_{11},x_{10}), edge(e_{18},x_{13}), and edge(e_{18},x_{10}). If we use these edges and search for roots we get the following paths: (e_2,x_4), (e_2,u_{11}), (e_2,u_{11},x_{10}), (e_{18},x_{10}), and (e_{18},x_{13}). The unique roots in the paths are e_2 and e_{18}. With a Predicate Tree Table and

Algorithm 5 Get Tree Node Algorithm

```
1: procedure GET-TREE-NODE( key, ptt)
2:     ep ← ptt.get( key )
3:     node.pred ← ep.pred
4:     if ( ep.argSize > 0 ) then
5:         for ( i=1 to ep.argSize ) do
6:             node.arg[i] ← GET-TREE-NODE( ep.arg[i], ptt )
7:         end for
8:     end if
9:     if ( ep.modSize > 0 ) then
10:        for ( i=1 to ep.modSize ) do
11:            node.mod[i] ← GET-TREE-NODE( ep.mod[i], ptt )
12:        end for
13:    end if
14:    return node
15: end procedure
```

a list of roots, we can create one or more trees as shown in Figure 6.3. The

[5]the first argument of a PP is referring to the subject in NorSource and no edge is created

algorithm for for creating a tree is shown in Algorithm 4 and Algorithm 5. The GET-PREDICATE-TREE algorithm is a simplified version of the MAP-PREDICATE-TO-ONTOLOGY algorithm that is presented in the next section. For each root we call the GET-TREE-NODE algorithm and places the result in the "treeTab" variable. The variable is returned when all roots are processed. In Algorithm 5, we receive a key and a Predicate Tree Table as arguments. We find the EP with the key in **line 2**. The algorithm fills the structure "node" with arguments and modifications. If the EP has arguments we execute a recursive call[6] for each argument (**line 4** to **line 8**). The algorithm also executes a recursive call for each modification in the EP (**line 9** to **line 13**). The algorithm returns the "node" variable. For our Predicate Tree Table in Table 6.3 and the roots e_2 and e_{18}, we get the trees: "mod(gå(gutt),til(bror))" and "poss_c(pron,bror)", shown as Prolog terms.

6.3 The Mapping Algorithm

The basic idea is to search the predicate tree depth first and to find the corresponding domain ontology elements for each EP predicate in the tree, like the trees in Figure 6.4. The predicate tree is to the left and the tree with domain ontology elements is to the right. The mapping process uses a table with top

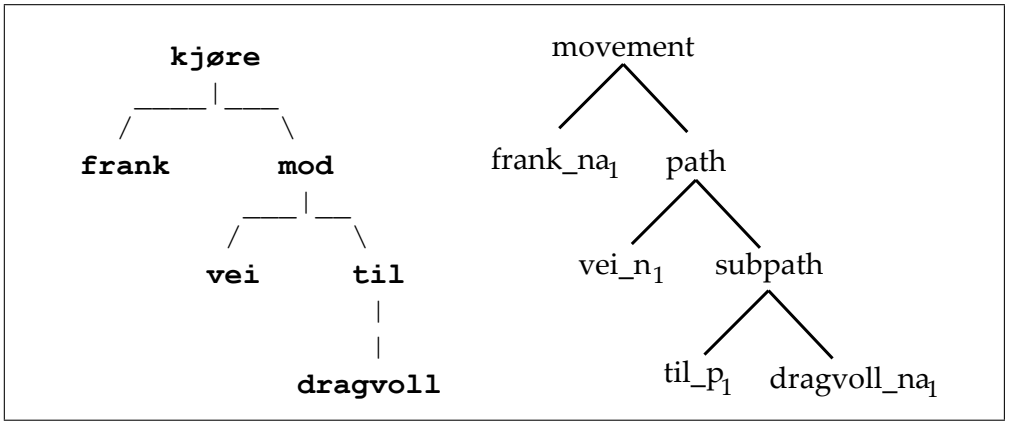


Figure 6.4: Predicate Tree And Domain Element Tree

nodes and a Predicate Tree Table. The algorithm is shown in Algorithm 6.

[6] A recursive call is a programming technique where a procedure calls itself. The recursive calling stops when we reach some stop criteria. A leaf node is such a creteria in our case.

The algorithm returns all the valid mappings. All the top nodes found in the MRS will be mapped (**line 3**). In **line 4** a variable for failed mappings is initiated and in **line 5** a closed list is initiated. The algorithm calls the MAP-EP procedure for each top node (**line 7**). The MAP-EP algorithm, Algorithm 7

Algorithm 6 Map Underspecified Predicates to Domain Ontology Algorithm

```
 1: procedure MAP-PREDICATE-TO-ONTOLOGY(topLst[], mrs)
 2:     result ← Ø
 3:     for ( i=1 to topLst.length ) do
 4:         fa ← Ø                                   ▷ failed mappings
 5:         cl ← Ø                                   ▷ prevent loop
 6:         mapping.top ← topLst[i]
 7:         mapping.item ← MAP-EP(mrs, cl, 1, topLst[i], fa)
 8:         mapping.failed ← fa
 9:         result.add(mapping)
10:     end for
11:     return result
12: end procedure
```

shown below, takes an MRS, a closed list, a level counter, a key, and a failed list as parameters.

Algorithm 7 Map EP Algorithm

```
 1: procedure MAP-EP(mrs, cl, level, key, fa)
 2:     if ( cl.contains(key) ) then
 3:         return with error "loop in tree at node =" + key
 4:     end if
 5:     cl.add(key)
 6:     matrix ← Ø
 7:     result ← Ø
 8:     ep ← mrs.find(key)                          ▷ Ep with ARG0=key
 9:     level = level+1
10:     sTab[] ← Util.ontology(key, ep.pred, pos)
11:     if ( ep.pos='v' and ep.size=2 ) then
12:         arg1Tab[] = MAP-EP(mrs, cl, level, ep.arg[1], fa)   ▷ recursive call
13:         arg2Tab[] = MAP-EP(mrs, cl, level, ep.arg[2], fa)   ▷ recursive call
14:         for ( i=1 to sTab.length ) do
15:             for ( j=1 to arg1Tab.length ) do
```

```
16:                 for ( k=1 to arg2Tab.length ) do
17:                     subMatrix ← CHECK_TV(sTab[i], arg1Tab[j], arg2Tab[k])
18:                     matrix.add(subMatrix)
19:                 end for
20:             end for
21:         end for
22:         if ( matrix.length=0 ) then
23:             fa.add(arg1Tab, arg2Tab, sTab)              ▷ no semantics for key
24:         end if
25:     else
26:         if ( ep.pos='n' and ep.size=0 ) then
27:             for ( i=1 to sTab.length ) do
28:                 tmp ← Parse0(ep.key, sTab[i].sense, 'n'))
29:                 matrix.add(tmp) ;
30:             end for
31:         end if
32:     end if
33:     if ( matrix.length > 0 ) then
34:         headLst ← matrix
35:         mIdx ← 1
36:         while ( ep.modSize >= mIdx ) do
37:             modMatrix ← ∅
38:             arg[] = MAP-EP(mrs, cl, level, ep.modTab[mIdx], fa)
39:             for ( k=1 to headLst.length ) do
40:                 for ( j=1 to arg.length ) do
41:                     theSubMatrix = CHECK-MOD(headLst[k],arg[j])
42:                     modMatrix.add(theSubMatrix)
43:                 end for
44:             end for
45:             if ( modMatrix.length=0 ) then
46:                 fa.add( findEp(ep.modTab[i]) )          ▷ no semantics for key
47:             else
48:                 headLst ← modMatrix
49:             end if
50:             mIdx ← mIdx + 1
51:         end while
52:         result.add(headLst)
53:     else
54:         result.add(matrix)
```

```
55:     end if
56:     return result
57: end procedure
```

In **line 8**, the MRS is searched for an EP with ARG_0 = key. In **line 10**, the predicate of the EP and the part-of-speech of the EP is used to search the domain ontology for concepts. The key variable is used as a reference back to the EP. The result is stored in sTab. In order to make the algorithm more readable we only show the code for transitive verbs and nouns. In **line 12** and **line 13**, every concept is found for the two verb arguments with recursive calls to the MAP-EP procedure. Between **line 14** and **line 21**, we call the algorithm CHECK-TV for the permutations of the concepts from the predicate and the two arguments. In **line 28**, we create a variable for a noun (leaf node). In **line 33**, if we found any mappings then we process the possible modifications. In **line 38**, we find all the mappings for ep.modTab[mIdx] and place the result in the arg table. Between **line 39** and **line 44**, we call the procedure CHECK-MOD for all the permutations between headLst and arg. If we found any mappings then we move these to the variable headLst (**line 48**). The variable headLst contains the latest mappings and the next modification is checked against the latest mappings. After the processing of the modifications, the variable headLst is copied to the variable result. If the EP doesn't have any modification, the matrix is stored in the variable result. The procedure returns the variable result.

The CHECK-TV procedure is shown in Algorithm 8. The procedure takes a predikate, arg_1, and arg_2 as arguments. Between **line 3** and **line 13**, the Prolog term pStr is built up with the types and senses from the arguments. The term is executed in Prolog and the result is stored in the table pTab (**line 14**). The pTab table contains all the templates found in the domain ontology. Between **line 22** and **line 31**, the algorithm tests if the checks from the templates are valid in the domain ontology. If all the checks are valid then the template is valid, and if the return function is "new" we call the class factory with the arguments. The class factory creates our complex domain object. We only show the return function "new" in order to make the algorithm more readable.

Algorithm 8 Check-Tv Algorithm

```
1: procedure CHECK-TV(pred, arg1, arg2)
2:     resultTab[] ← Ø
3:     pStr.name ← "tv_tmpl"
4:     pStr.id ← "Id"
5:     pStr.pred-type ← pred.type
6:     pStr.pred-sense ← pred.sense
7:     pStr.arg1-type ← arg1.type
8:     pStr.arg1-sense ← arg1.sense
9:     pStr.arg2-type ← arg2.type
10:    pStr.arg2-sense ← arg2.sense
11:    pStr.lst ← "Lst"
12:    pStr.rf ← "Rf"
13:    pStr.rc ← "Rc"
14:    pTab[] ← Prolog.query(pStr)                    ▷ Call Prolog
15:    for ( i = 1 to pTab.length ) do
16:        checkTab ← pTab[i].Lst
17:        templateId ← pTab[i].Id
18:        returFunc ← pTab[i].Rf
19:        returClass ← pTab[i].Rc
20:        theChecksTab[] ← Ø
21:        checkOK ← true
22:        for ( j = 1 to checkTab.length ) do
23:            cStr.name ← "check"
24:            cStr.check ← checkTab[j]
25:            queryCheck ← Prolog.query(cStr)        ▷ Call Prolog
26:            if ( queryCheck.hasSolution ) then
27:                theChecksTab.add(checkTab[j])
28:            else
29:                checkOK ← false
30:            end if
31:        end for
32:        if ( checkOK ) then
33:            if ( returFunc = "new" ) then
34:                theClass ← ClassFactory.create(returClass,arg1,arg2)
35:                theClass.add(theChecksTab,templateId)
36:                resultTab.add(theClass)
37:            end if
38:        end if
39:    end for
40:    return resultTab
41: end procedure
```

6.4 Templates And Return Functions

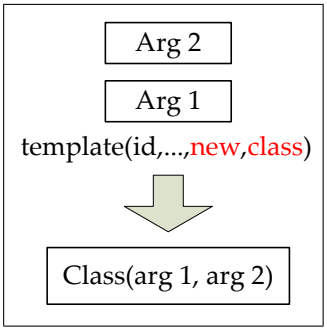


Figure 6.5: Function New

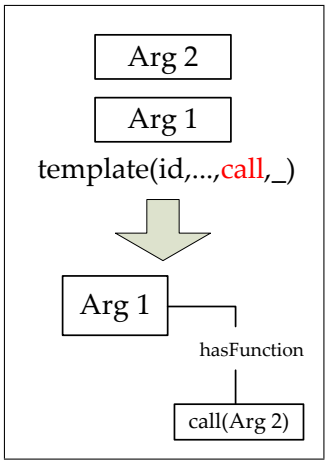


Figure 6.6: Function Call

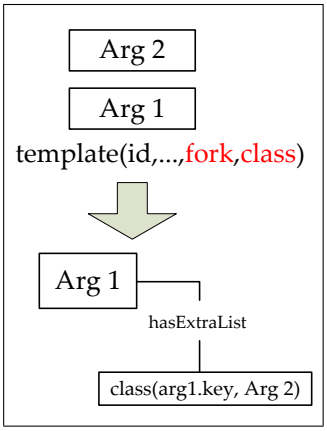


Figure 6.7: Function Fork

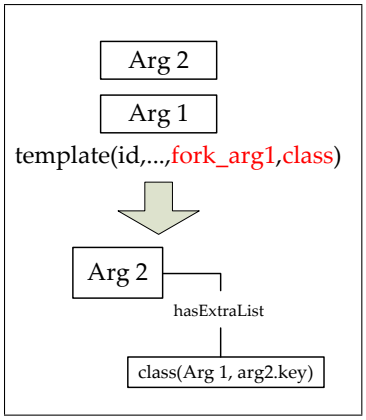


Figure 6.8: Function Fork Arg1

We have seen the function "new" used in the CHECK-TV algorithm. Before we introduce more functions, we present the format for a template:

tmpl(id, $node_1$, $node_2$, check list, return function, return class)

A template has one node for the predicate and one for each argument. The id identifies the template. The checklist contains ontology checks that must be

true in order for the template to be valid. The return function and the return class denote the action for the nodes. A template can contain the return functions: *new*, *fork*, *fork_arg*$_1$, and *call*. The return function and the return class determine how two nodes are compiled together. With the function *new*, a new class is created with the arg_1 and arg_2 node as arguments (Figure 6.5). The *fork* function returns the arg_1 node and creates a new class with a reference to the arg_1 node and the arg_2 node (Figure 6.7). The new class is added to a list called ExtraList in the arg_1 node. An example is *the black bus*; the bus node is returned and a feature class with a reference to bus and the color is created and added to the list. The *fork_arg*$_1$ function is like the *fork* function. The difference is that arg_1 and arg_2 have switched places (Figure 6.8). The *call* function denotes that arg_2 is consumed by arg_1. The arg_1 node has a function with the name *call*, and this function is called with arg_2 as argument. For example if the first class is Movement and the second is Subpath, the Movement class's function *call* is called with the argument Subpath.

6.5 Summary

Algorithm 6 maps underspecified predicates to domain specific predicates and it can be used in line 18 in Algorithm 2. With our approach we can implement a structure for an event and for a verb argument. For example "depart" is a sub-event of "move", and if a car moves to a location, the ongoing state of movement is followed by an arrival event. A car can have a number of location connected to it. We are now ready to model some parts of the Change Location domain.

Part III

Result

In the next chapter we presents two versions of the pipeline. One for the undespecified discourse level and one for the specified world level. Then we present a chapter that contains a detailed model of the Change Location domain. The domain shows how the domain ontology and the mapping algorithm works together. The chapter contains a number of examples of how the mapping works and how the complex domain elements can be used to generate roles and states.

Chapter 7

The Pipelines

7.1 The Pipeline with Unmapped Predicates

The unmapped pipeline is an implementation of Algorithm 1 with object and event references at the discourse level. The pipeline reads well-formed sentences in Norwegian. The sentences are parsed with the NorSource grammar together with a Maximum Entropy ranker. The highest ranked MRS is selected. The pipeline performs a simple pronoun resolution. The resolution algorithm works as follows: for each selected MRS we collect candidates for a pronoun and we store them in a table, and when we find a pronoun we search the table from the back for a candidate that has the same *type*, *gender*, and *number* as the pronoun. We process a small discourse to show how the pipeline works.

$$\frac{\text{mannen smiler}}{\text{the man smiles}} \tag{7.1}$$

The first sentence from our discourse is (7.1). The selected MRS is shown in Table 7.1 and the features for the MRS are shown in Table 7.2. The object x_{1d} is stored as an pronoun candidate.

MRS
h_3:_mann_n_rel(x_{1d})
h_4:_def_q_rel(x_{1d},h_5,h_6)
h_7:_smile_v-intr_rel(e_{1d},x_{1d})

Table 7.1: MRS for (7.1)

Key	Feature	Value
e_{1d}	mood	indicative
e_{1d}	tense	pres
e_{1d}	sf	prop
x_{1d}	gen	m
x_{1d}	num	sing
x_{1d}	pers	thirdpers

Table 7.2: Features for (7.1)

$$\frac{\text{Gro liker ham}}{\text{Gro likes him}} \tag{7.2}$$

The second well-formed sentence from our discourse is (7.2). The selected MRS is shown in Table 7.3 and the features for the MRS are shown in Table 7.4. We can see that the object and event references are unique. Here, the object x_{2d} is a pronoun and we search our candidate table and we found the discourse referent x_{1d}. The object x_{3d} is stored as an pronoun candidate.

MRS
h_{10}:han_pron_rel(x_{2d})
h_{11}:_pronoun_q_rel(x_{2d},h_{12},h_{13})
h_3:named_rel(x_{3d},Gro)
h_5:_def_q_rel(x_{3d},h_6,h_7)
h_8:_like_v-tr_rel(e_{2d},x_{3d},x_{2d})
pronoun_ref=x_{2d}, map_ref=x_{1d}

Table 7.3: MRS for (7.2)

Key	Feature	Value
e_{2d}	mood	indicative
e_{2d}	tense	pres
e_{2d}	sf	prop
x_{3d}	gen	f
x_{3d}	num	sing
x_{3d}	pers	thirdpers
x_{2d}	gen	m
x_{2d}	num	sing
x_{2d}	pers	thirdpers

Table 7.4: Features for (7.2)

han elsker henne
he loves her

(7.3)

The third sentence from our discourse is (7.3). The selected MRS is shown in

MRS
h_{10}:_pronoun_q_rel(x_{4d},h_{11},h_{12})
h_3:han_pron_rel(x_{5d})
h_4:_pronoun_q_rel(x_{5d},h_5,h_6)
h_7:_elske_v-tr_rel(e_{3d},x_{5d},x_{4d})
h_9:hun_pron_rel(x_{4d})
pronoun_ref=x_{5d}, map_ref=x_{1d}
pronoun_ref=x_{4d}, map_ref=x_{3d}

Table 7.5: MRS for (7.3)

Key	Feature	Value
e_{3d}	mood	indicative
e_{3d}	tense	pres
e_{3d}	sf	prop
x_{5d}	gen	m
x_{5d}	num	sing
x_{5d}	pers	thirdpers
x_{4d}	gen	f
x_{4d}	num	sing
x_{4d}	pers	thirdpers

Table 7.6: Features for (7.3)

Table 7.5 and the features for the MRS are shown in Table 7.6. Here, we found two pronouns. The first pronoun x_{5d} is connected to the discourse referent x_{1d}, and the second pronoun x_{4d} is connected to the discourse referent x_{3d}.

After we have processed the discourse, the pronoun resolution algorithm stored two pronoun candidates, which are shown in Table 7.7.

seqNr	predicate	ref	carg	num	gen	pers
1	_mann_n_rel	x_{1d}	null	sing	m	thirdpers
2	named_rel	x_{3d}	Gro	sing	f	thirdpers

Table 7.7: Pronoun Candidates

7.2 The Pipeline with Mapped Predicates

This is an implementation of Algorithm 2. The domain is the classic Box World from Artificial Intelligence. The simple First-Order Logic model is shown as an image, and the initial model is shown in Figure 7.1 on page 64. The idea is that we can command an invisible robot to move the boxes around, and that the effects of the commands are reflected in a new image. The simple dialogue system is implemented as a web application at:

`http://regdili.idi.ntnu.no:8080/boxworldweb/boxworlddemo`.

The application starts with the initial model where a red, blue, green and yellow box are laying on a table.

$$\frac{\text{flytt den gule boksen på den røde boksen}}{\text{move the yellow box to the red box}} \tag{7.4}$$

The first command in our discourse is (7.4), and it gives the Predicate Tree Table in Table 7.8.

label	key	predicate	pos	sense	argTab	modTab
h_{10}	x_{1d}	boks-1	n			u_{2d}
h_{10}	u_{2d}	gul	a		x_{1d}	
h_{18}	x_{2d}	boks-1	n			u_{4d}
h_{18}	u_{4d}	rød	a		x_{2d}	
h_3	e_{1d}	flytte_trScpr	v		x_{3d},x_{1d},u_{5d}	
h_5	u_{5d}	på_dirtel	p		x_{2d}	
h_6	x_{3d}	addressee-rel	n			

Table 7.8: Box World Predicate Tree, Example 1

According to step 18, in Algorithm 2, the MRS is mapped to domain elements. For our MRS we get the following elements:

Flytt
event(e_{1d}:flytte_trScpr_v$_1$)
subject(x_{3d}:addressee_rel_n$_1$)
object(x_{1d}:boks_n$_1$)
path
end-point(x_{2d}:boks_n$_1$)
Noun_feature
color(u_{2d}:gul_a$_1$)
noun(x_{1d}:boks_n$_1$)
Noun_feature
color(u_{4d}:rød_a$_1$)
noun(x_{2d}:boks_n$_1$)

In step 19, the MRS is interpreted in a world model. Our model is shown in Figure 7.1, and since the MRS is a command, we can only find the presuppositions to the command and interpret them in the world model. We find the predicates in the MRS that are leaf nodes (they have no arguments): x_{1d}, x_{2d}, and x_{3d} (the two boxes and the addressee). The leaf nodes are also modified as

key	modified by	world reference
x_{1d}	$[u_{2d}]$	$[x_{2w}]$
x_{2d}	$[u_{4d}]$	$[x_{3w}]$
x_{3d}	[]	$[x_{5w}]$

Table 7.9: Reference Mapping, Example 1

shown in Table 7.9. We have stored the model with predicates from an MRS and changed the references to the series from the world . If we interpret the sentence "flytt boksen/(move the box)", we find four boxes in our model, and since the sentence has a quantifier that takes singular objects, we can return an error message to the user stating that the definite quantifier is inappropriate. If we interpret the sentence "flytt de gule boksene/(move the yellow boxes)", we have another mismatch between the model and the quantifier.

In step 20, in Algorithm 2, we find the dialogue act. The dialogue acts in our Box World application are simple; they are just the commands with valid arguments. We use the complex domain element to export the MRS to Prolog format, and we use Prolog to reason with the dialogue act. In our example the act is:

folEp(e_{1d}, flytte_trScpr_v$_1$, arg$_1$(x_{3d}), arg$_2$(x_{1d}), from(nil), to(x_{2d})).

In step 21 in our algorithm, we execute the dialogue act. First, we check if the dialogue manager accepts the command and its arguments. Second, we also check if the move is legal. For example, only boxes that are free[1] can be moved. If the dialogue act pass these checks, we execute the dialogue act. The result is loaded into the logic model and we generate a new image that

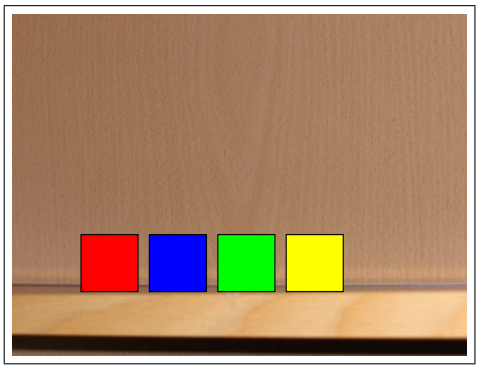


Figure 7.1: Step init

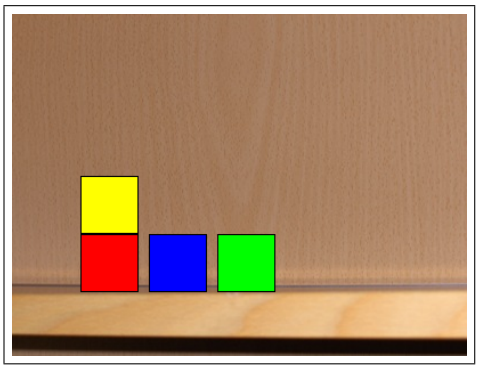


Figure 7.2: Step 1

shows the current logic model, see Figure 7.2. Each user of the application has their own state of the logic model. The model is selected before we execute the dialogue act and the model is stored in the database with the updated result.

[1] a free box has no box on top of itself

Legg den grønne boksen opp på den gule boksen
put the green box on the yellow box (7.5)

The second command in our discourse is (7.5) and the Predicate Tree Table is shown in Table 7.10. The complex elements mapped from the Predicate Tree

label	key	predicate	pos	sense	argTab	modTab
h_{19}	x_{5d}	boks-1	n			u_{7d}
h_{19}	u_{7d}	gul	a		x_{5d}	
h_3	e_{2d}	legge_tr	v		x_{6d},x_{4d}	u_{8d}
h_5	x_{6d}	addressee-rel	n			
h_9	x_{4d}	boks-1	n			u_{12d}
h_9	u_{12d}	grønn	a		x_{4d}	
h_3	u_{8d}	opp_dirtel_på_dirtel	adv_p		x_{4d},x_{5d}	

Table 7.10: Box World Predicate Tree, Example 2

Table are:

Legg
event(e_{2d}:legge_tr_v$_1$)
subject(x_{6d}:addressee_rel_n$_1$)
object(x_{4d}:boks_n$_1$)
path
end-point(x_{5d}:boks_n$_1$)
Noun_feature
color(u_{12d}:grønn_a$_1$)
noun(x_{4d}:boks_n$_1$)
Noun_feature
color(u_{7d}:gul_a$_1$)
noun(x_{5d}:boks_n$_1$)

The discourse variables for the leaf nodes are interpreted in the logic model and the result is shown in Table 7.11. The dialogue act for our MRS is:

key	modified by	world reference
x_{5d}	$[u_{7d}]$	$[x_{2w}]$
x_{6d}	[]	$[x_{5w}]$
x_{4d}	$[u_{12d}]$	$[x_{4w}]$

Table 7.11: Reference Mapping, Example 2

folEp(e_{2d}, legge_tr_v1, $\text{arg}_1(x_{6d})$, $\text{arg}_2(x_{4d})$, $\text{to}(x_{5d})$).

The result from the dialogue act is shown in Figure 7.4.

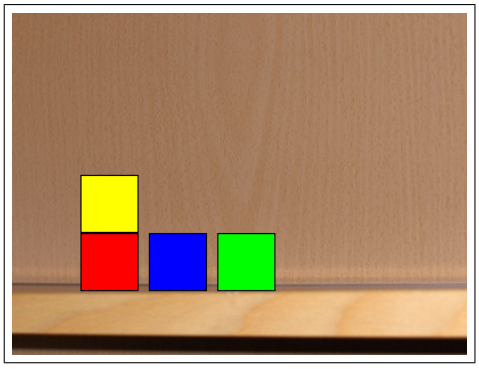


Figure 7.3: Step 1

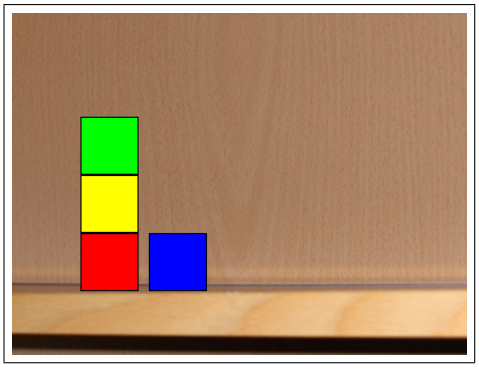


Figure 7.4: Step 2

flytt boksene ned på bordet
move the boxes down on the table (7.6)

The next command in our discourse is (7.6) and the MRS generates the Predicate Tree Table shown in Table 7.12. The complex element mapped from the

label	key	predicate	pos	sense	argTab	modTab
h_{10}	x_{7d}	boks-1	n			
h_{16}	x_{8d}	bord-1	n			
h_3	e_{3d}	flytte_trScpr	v		x_{9d},x_{7d},u_{13d}	
h_6	x_{9d}	addressee-rel	n			
h_5	u_{13d}	ned_dirtel_på_dirtel	adv_p		x_{7d},x_{8d}	

Table 7.12: Box World Predicate Tree, Example 3

Predicate Tree Table is:

Flytt
event(e_{3d}:flytte_trScpr_v_1)
subject(x_{9d}:addressee_rel_n_1)
object(x_{7d}:boks_n_1)
path
end-point(x_{8d}:bord_n_1)

Since we have a quantifier with a set that is plural, the set is filled with all the boxes in our world model. The interpretation of the presuppositions are shown in Table 7.13.

key	modified by	world reference
x_{7d}	[]	[x_{1w}, x_{2w}, x_{3w}, x_{4w}]
x_{8d}	[]	[x_{6w}]
x_{9d}	[]	[x_{5w}]

Table 7.13: Reference Mapping, Example 3

The dialogue act generated is:

folEp(e_{3d}, flytte_trScpr_v_1, arg$_1$(x_{9d}), arg$_2$(x_{7d}), from(nil), to(x_{8d})).

The logic model after the dialogue act was executed is shown in Figure 7.6.

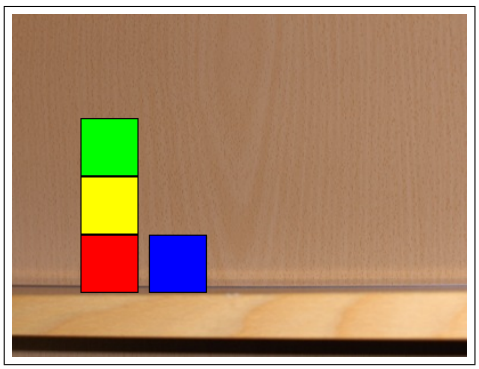


Figure 7.5: Step 2

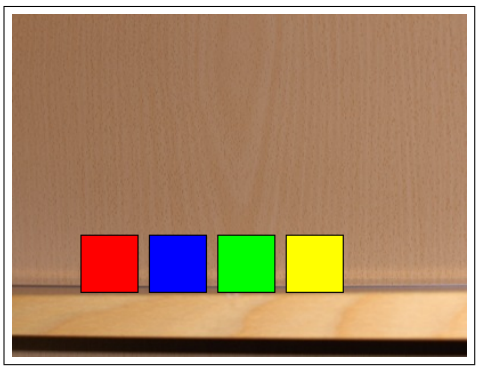


Figure 7.6: Step 3

flytt den røde boksen
move the red box (7.7)

The next command in our discourse is (7.7) and the Predicate Tree Table is

label	key	predicate	pos	sense	argTab	modTab
h_3	e_{4d}	flytte_intr	v		x_{11d},x_{10d}	
h_5	x_{11d}	addressee-rel	n			
h_9	x_{10d}	boks-1	n			u_{15d}
h_9	u_{15d}	rød	a		x_{10d}	

Table 7.14: Box World Predicate Tree, Example 3

shown in Table 7.14.

The Predicate Tree Table is mapped to the following complex elements:

Flytt
event(e_{4d}:flytte_tr_v$_1$)
subject(x_{11d}:addressee_rel_n$_1$)
object(x_{10d}:boks_n$_1$)
Noun_feature
color(u_{15d}:rød_a$_1$)
noun(x_{10d}:boks_n$_1$)

The interpretation of the presuppositions are shown in Table 7.15. The di-

key	modified by	world reference
x_{11d}	[]	[x_{5w}]
x_{10d}	[u_{15d}]	[x_{3w}]

Table 7.15: Reference Mapping, Example 4

alogue act for the complex element is:

folEp(e_{4d}, flytte_tr_v$_1$, arg$_1$(x_{11d}), arg$_2$(x_{10d}), from(nil), to(nil)).

The logic model is updated, Figure 7.8, after the dialogue act is executed. As we can see, the dialogue act is executed, but the move operation is not

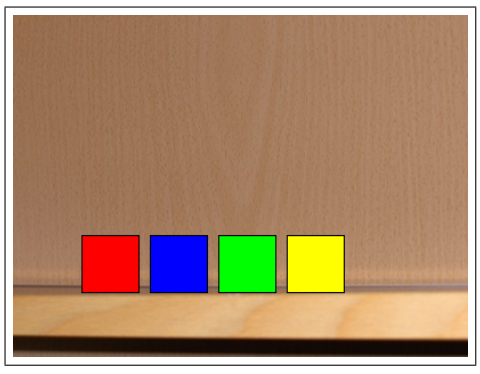


Figure 7.7: Step 3

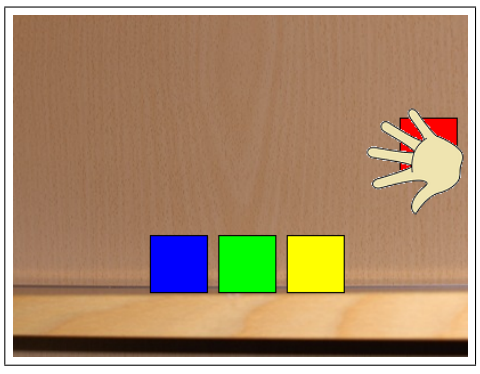


Figure 7.8: Step 4

finished. The robot shows his hand and he is keeping the box until a location is provided.

The Box World application returns errors when the sentence from the user is not according to the interpretation of the presuppositions or the dialogue manager's accepted commands. For example, "mannen smiler / the man smiles", results in an error from the dialogue manager. The manager accepts only commands.[2] When try the sentence "flytt den gråe boksen opp på den gule boksen / move the grey box on the yellow box", the interpretation process of the presuppositions is not able to find the color grey on any box in our model. Another example is "flytt bordet / move the table". The dialogue manager is not allowed to move the table, and we get the error message: command is rejected.

7.3 Summary

We presented the pipeline with unmapped predicates, where we had unmapped predicates and references from the discourse level. We also presented a simple algorithm for pronoun resolution. This pipeline can be used on text if we have coverage in the lexicon and a ranker that places the most relevant MRS on top.

We can map an MRS to a complex element in a domain, where we can generate roles and express event structure. We can use the connected MRS to infer a reaction with our linguistic application. This is shown with the pipeline for mapped predicates (we can ask the robot to move boxes around). Here, we used the "Box World" domain together with a model. We interpreted the robot's commands in the model, and we checked if the robot was allowed to execute the commands. The model changed after the command was executed.

[2]This will change in the future. At this stage the application is only a prototype

Chapter 8

Mapping Predicates To A Domain Ontology

In Section 6, we prepared an MRS and we created a predicate tree, then we used Algorithm 6 with the predicate tree. Now, we proceed with examples from the Change Location domain and with a Change Location domain ontology. We have analyzed over 2 million sentences that was collected from BusTUC, and we use types that where discovered there in our design. The analysis of the BusTUC sentences is placed in Appendix A.2.

8.1 The Change Location Language

The domain ontology contains a set of concepts, a type hierarchy, a set of *has* and *use* relations, the templates, and the complex domain types. Parts of the ontology are shown in Figure D.1 on page 156, Figure D.2 on page 157. Figure D.3 on page 158, Figure D.4 on page 159, and Figure D.5 on page 160. The complex domain elements from the Change Location domain are: Movement, SubPath and Path.

Movement
- event: ParseElem
- subject: ParseElem
- object: ParseElem
- path: Path
- cargo: ParseElem
- vehicle: ParseElem
- companion: ParseElem

ParseElem
- type: String
- key: Integer
- sense: String
- ep-ref: String

SubPath
- type: String
- key: Integer
- sense: String
- ep-ref: String
- argument: ParseElem
- time-point: String

Path
- object: ParseElem
- fra: SubPath
- til: SubPath
- onPathList: ⟨ SubPath ⟩
- direction: ParseElem

We present a number of examples from the Change Location domain, and they show how the mapping algorithm works together with the templates and the complex domain elements. The examples also show how we can generate time points, roles and states with the complex domain elements. Sometimes a word can be treated as a symbol, but not always. We show that objects can have structure and that part of one structure is connected to a part of another structure.

A summary of the roles we use are shown in Table 8.1. The roles *work* and *cargo* are used together with example sentence "NP1 går med NP2" ("NP1 goes with NP2"). The arguments positions (subject and PP.NP) are not enough to decide the roles. The verb *go* expresses movement in the example. The domain knowledge created for a verb argument is: the thing's ability to move on its own and the thing's ability to be moved by others. The volume and weight are also important; for example, an adult can carry an infant. The medium for the movement is also important; a ferry use the water and an airplane use the air. How people talk is another factor that comes into the consideration. Some of the examples were odd in Norwegian.

subject	pp.np	cargo,cargo	cargo,work	work,cargo	work,work
person	person		x	x	x
person	vehicle		x	x	
person	item		x		
vehicle	person		x	x	
vehicle	vehicle		x	x	x
vehicle	item			x	
item	person		x		
item	vehicle		x		
item	item	x			

Table 8.1: Roles, Summary

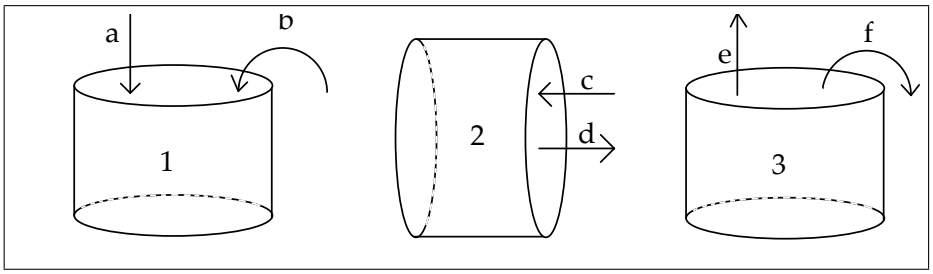


Figure 8.1: Containers with Orientation

We look into some examples where the combinations of adverbs, prepositions and objects can indicate the orientation of a container. The orientation and the containers are shown in Figure 8.1. The orientation of the container depends on world knowledge and the relation within the world. Some odd examples were found with the idiosyncratic preposition *på*. The grammar is the obvious place for defining idiosyncratic use of language. Our framework can sort out the domain relevant information in such cases and extend it into the domain.

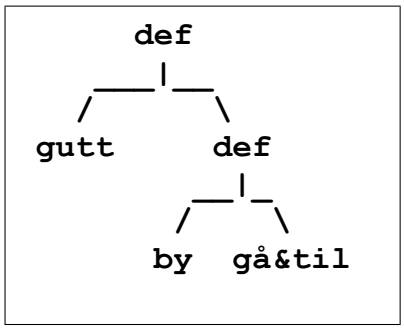


Figure 8.2: FOL

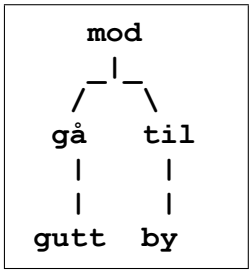


Figure 8.3: Predicate Tree

8.2 The Change Location Examples

These examples to follow illustrate the basic use of the Path and the Movement class. The Path contains sub events with an internal time variable. The first basic sentence is used to show the details of the inner workings of the mapping algorithm. The next sentences focus on valid templates and senses.

The sentence in (8.1) is parsed and it returns one reading. The MRS details are listed in Listing E.1 on page 163 and a simplified version of the MRS is presented in Table 8.2.

Gutten går til byen
The boy goes to the town (8.1)

The MRS is valid and Utool returns the resolved MRS as the prolog pred-

key	pos	pred	sense	EP
x_4	n	gutt		h_3:_gutt_n_rel(x_4)
	q	def		h_5:_def_q_rel(x_4,h_7,h_6)
e_2	v	gå		h_8:_gå_v_rel(e_2,x_4)
u_{11}	p	til		h_8:_til_p_rel(u_{11},x_4,x_{10},u_9)
x_{10}	n	by		h_{12}:_by_n_rel(x_{10})
	q	def		h_{13}:_def_q_rel(x_{10},h_{15},h_{14})

Table 8.2: Domain Example 1, MRS

icate: *def(gutt, def(by, gå & til))*, and this predicate is shown in Figure 8.2. The predicates in the MRS are prepared and the result (the Predicate Tree Table) is presented in Table 8.3. The variables e_2 and u_{11} denote events. The variables x_4 and x_{10} denote individuals. The predicate table is used to create

the predicate tree, see Figure 8.3. The Predicate Tree Table and the root nodes from the predicate tree are input to Algorithm 6. The algorithm starts with

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	gutt	n			
h_8	e_2	gå	v		x_4	u_{11}
h_8	u_{11}	til	p		x_4,x_{10}	
h_{12}	x_{10}	by	n			

Table 8.3: Predicate Tree Table, Example 1

the event (e_2), the main event (top handle) in the predicate tree. The key is used to look up its entry in the predicate tree table. The entry is a modified predicate node. The semantics for the predicate *gå* returns two senses: v(1, gå_v$_1$, e$_2$), v(2, gå_v$_2$, e$_2$). The first sense is for animates that move along a path (John goes to Oslo), and the second sense is for static objects that denote a path (The road goes to Oslo). The completion step starts with the completion of the nodes argument. The key x_4 is a leaf node and it returns one sense n(4, gutt_n$_1$, x_4). The predicate step for key=e_2 can be finished. The algorithm searches for templates with the arguments arg$_1$(v,gå_v$_1$) and arg$_2$(n,gutt_n$_1$). Three templates are found (Table 8.4). The template with

key	nodes	check list	return	class
itv$_1$	arg1(v,gå_v$_1$)	isa(gå_v$_1$,mannerType_v$_1$)	new	movement
	arg2(n,gutt_n$_1$)	has(gutt_n$_1$,moveable)		
itv$_2$	arg1(v,gå_v$_1$)	isa(gå_v$_1$,underSpecSubj_v$_1$)	new	movement
	arg2(n,gutt_n$_1$)	has(gutt_n$_1$,moveable)		
itv$_6$	arg1(v,gå_v$_1$)	isa(gå_v$_1$,beg_end_v$_1$)	new	movement
	arg2(n,gutt_n$_1$)	has(gutt_n$_1$,moveable)		

Table 8.4: Example 1, template set 1

key *itv*$_2$ has valid checks. The others do not, because *gå_v*$_1$ is neither a hyponym of the senses *mannerType_v*$_1$ nor *beg_end_v*$_1$. The verb *gå_v*$_2$ is not valid in any of the templates, so a Movement class is created with the arguments *gå_v*$_1$ and n(4, gutt_n$_1$, x_4). The next step for key x_4 is the modification step. There is a modification with the key u_{11}. The table entry is a predicate node. The ontology returns the sense: p(6, *til_p*$_1$,u_{11}). The argument with key=x_{10} returns the sense n(7, *by_n*$_1$, x_{10}). A template search is performed with the search arguments arg$_1$(p, *til_p*$_1$) and arg$_2$(n, *by_n*$_1$). The search returns two templates (Table 8.5). Template *pp*$_{13}$ has an invalid check list. The template

key	nodes	check list	return	class
pp_2	arg1(p,til_p_1) arg2(n,by_n_1)	has(by_n_1,location)	new	subpath
pp_{13}	arg1(p,til_p_1) arg2(n,by_n_1)	isa(til_p_1,direction_creation_r_1) isa(by_n_1,path_n_1)	new	subdirection

Table 8.5: Example 1, template set 2

pp_2 is valid, and a SubPath class is created with the arguments p(6, *til_*p_1,u_{11}) and n(7, *by_*n_1, x_{10}). The modification step for key e_2 searches for templates with the arguments arg$_1$(movement,gå_v$_1$) and arg$_2$(subpath,til_p$_1$). One template is found (Table 8.6). The template is valid and the return function is *call*.

key	nodes	check list	return	class
mod_1	arg1(movement,gå_v$_1$) arg2(subpath,til_p$_1$)	isa(til_p$_1$,subpath_p$_1$) isa(gå_v$_1$,movement_v$_1$)	call	

Table 8.6: Example 1, template set 3

This means that the Movement class is called with argument number two. All nodes are now completed and the following complex element is returned:

movement
- event(e_2:gå_v$_1$)
- subject(x_4:gutt_n$_1$)
- path
  - end-point(x_{10}:by_n$_1$,t_2)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(gutt_n$_1$,movable)
  - isa(til_p$_1$,subpath_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
- templates
  - [itv$_2$,mod$_1$,pp$_2$]

Only variables with assigned values are displayed. The checks are the elements of the check lists for all valid templates for this element. There are three *is-a* relations, two for the verb and one for the preposition.

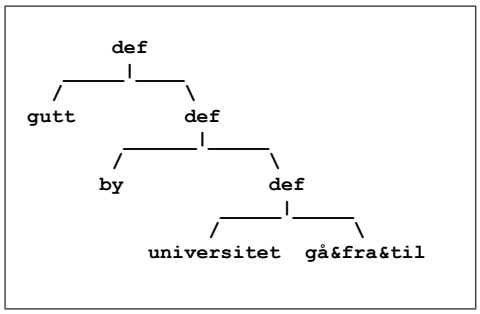


Figure 8.4: FOL

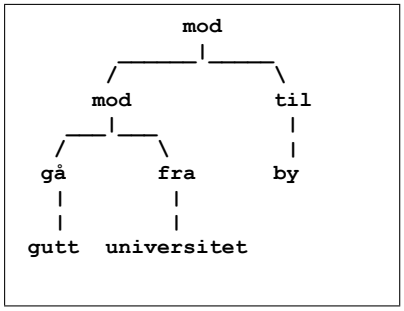


Figure 8.5: Predicate Tree 2

The second example is (8.2). The sentence is like the first, but with more information about the beginning of the change location event. The NorSource grammar returns four readings. The readings are caused by the usual preposition ambiguity and two variants of the preposition *til* (to). The first variant is the sense that denote possession (*_possessed_by_rel*) and the second is underspecified. The reading with the underspecified preposition is ranked at the top.

Gutten går fra universitetet til byen	(8.2)
The boy goes from the university to the town	

The parsed sentence's selected MRS is shown in Table 8.7. The Predicate

key	pos	pred	sense	EP
x_4	n	gutt		h_3:_gutt_n_rel(x_4)
	q	def		h_5:_def_q_rel(x_4,h_7,h_6)
e_2	v	gå		h_8:_gå_v_rel(e_2,x_4)
u_{11}	p	fra		h_8:_fra_p_rel(u_{11},x_4,x_9,u_{10})
x_9	n	universitet		h_{12}:_universitet_n_rel(x_9)
	q	def		h_{13}:_def_q_rel(x_9,h_{15},h_{14})
u_{18}	p	til		h_8:_til_p_rel(u_{18},x_4,x_{17},u_{16})
x_{17}	n	by		h_{19}:_by_n_rel(x_{17})
	q	def		h_{20}:_def_q_rel(x_{17},h_{22},h_{21})

Table 8.7: Domain Example 2, MRS

Tree Table is shown in Table 8.8. One way to express a PP modification in NorSource is to use the subject from the verb and place the EP for the PP with the same handle as the verb. In Table 8.8 the event e_2 is modified by

u_{11} and u_{18}. This information is used when the predicate table is created. The current sentence has the same valid templates as the previous sentence,

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	gutt	n			
h_8	e_2	gå	v		x_4	u_{11},u_{18}
h_8	u_{11}	fra	p		x_4,x_9	
h_{12}	x_9	universitet	n			
h_8	u_{18}	til	p		x_4,x_{17}	
h_{19}	x_{17}	by	n			

Table 8.8: Predicate Tree Table, Example 2

plus templates for the extra PP: one template for the preposition and one for the modification. The templates are shown in Table 8.9. The preposition template returns a SubPath class, while the modification template indicate that Movement class is called with the the SubPath class as argument. When

key	nodes	check list	return	class
pp_1	arg_1(p,fra_p_1) arg2(n,universitet_n_1)	has(universitet_n_1,location)	new	subpath
mod_1	arg1(movement,gå_v_1) arg_2(subpath,fra_p_1)	isa(fra_p_1,subpath_p_1) isa(gå_v_1,movement_v_1)	call	

Table 8.9: Example 2, template set 1

all the nodes in the predicate tree are completed, time points are added to the sub events. The first time point is added to the first sub event. The second is the ongoing activity denoted by the change location verb. The last time point is added to the last sub event.

movement
- event(e_2:gå_v_1)
- subject(x_4:gutt_n_1)
- path
  - begin-point(x_9:universitet_n_1,t_1)
  - end-point(x_{17}:by_n_1,t_3)

checks
- isa(gå_v_1,underSpecSubj_v_1)
- has(gutt_n_1,movable)
- isa(fra_p_1,subpath_p_1)
- isa(gå_v_1,movement_v_1)
- isa(til_p_1,subpath_p_1)

templates
- [itv_2,mod_1,pp_1,pp_2]

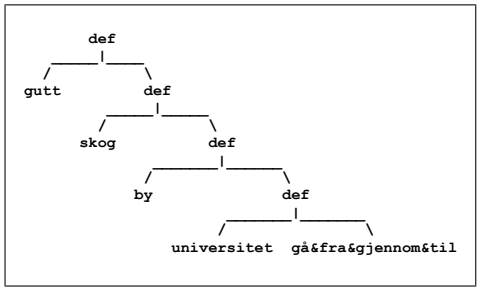


Figure 8.6: FOL

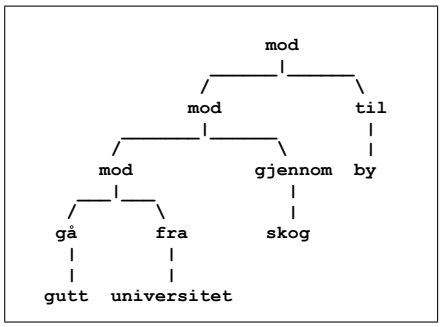


Figure 8.7: Predicate Tree

The next sentence (8.3) is like the previous sentence, but with extra information about the ongoing activity. The logical form from Utool is shown in Figure 8.6 and the predicate tree is shown in Figure 8.7.

Gutten går fra universitetet gjennom skogen til byen	
The boy goes from the university through the woods to the town	(8.3)

The event, the prepositions *fra* and the preposition *til* are as described in the

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	gutt	n			
h_8	e_2	gå	v		x_4	u_9,u_{16},u_{25}
h_8	u_9	fra	p		x_4,x_{11}	
h_{12}	x_{11}	universitet	n			
h_8	u_{16}	gjennom	p		x_4,x_{17}	
h_{19}	x_{17}	skog	n			
h_8	u_{25}	til	p		x_4,x_{24}	
h_{26}	x_{24}	by	n			

Table 8.10: Example 3, predicate

previous examples. The preposition *gjennom* (through) expects the PP.NP to have the *penetrable* feature. The template for this preposition is shown in Table 8.11. The feature is set on towns, woods, hedges, etc. The PP.NP is an area with objects. It is possible to move between these objects. The moving object's height is compared to the objects in the area (not present in the template checks). An ant can for example move through a lawn. The *gjennom* preposition creates a SubPath class and the Movement class receives the Sub-

key	nodes	check list	return	class
pp_6	arg1(p,gjennom_p$_1$) arg2(n,skog_n$_1$)	has(skog_n$_1$,penetrable)	new	subpath

Table 8.11: Example 3, template set 1

Path through the call function. The SubPath preposition describes the path the ongoing movement is following, and it is placed in the Path feature inside the Movement class. One complex element is returned.

movement
- event(e_2:gå_v$_1$)
- subject(x_4:gutt_n$_1$)
- path
  - begin-point(x_{11}:universitet_n$_1$,t_1)
  - subpath
    - Rel(u_{16}:gjennom_p$_1$)
    - Point(x_{17}:skog_n$_1$)
    - Time(t_2)
  - end-point(x_{24}:by_n$_1$,t_3)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(gutt_n$_1$,movable)
  - isa(fra_p$_1$,subpath_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - isa(gjennom_p$_1$,subpath_p$_1$)
  - isa(til_p$_1$,subpath_p$_1$)
- templates
  - [itv$_2$,mod$_1$,pp$_1$,pp$_6$,pp$_2$]

The first timepoint is at the beginning of the path, the second is at the element in the sub path list, and the third is at the end point of the path.

The next sentence is the same as (8.3), but with template mod_1 removed from the ontology. With this change, the final complex element is:

movement
    event(e_2:gå_v$_1$)
    subject(x_4:gutt_n$_1$)
    checks
        isa(gå_v$_1$,underSpecSubj_v$_1$)
        has(gutt_n$_1$,moveable)
    templates
        [itv$_2$]
failed
    p(u$_9$:fra)
    p(u$_{16}$:gjennom)
    p(u$_{25}$:til)

The EPs that are not according to the Change Location domain or others domains are listed in the *failed* list. This is an easy way to show incomplete parts of the ontology, instead of rejecting the completion and giving an error message. In "My uncle went to London", the fact that the speaker has an uncle can be part of the family relation domain and not the Change Location domain. These elements are used together with the return function "fork" and the family relation ends up in the *extra* list

Veien går til byen
The road goes to the town (8.4)

The example (8.4) is parsed and the MRS in Table 8.12 is selected. The MRS is prepared and the Predicate Tree Table is created (Table 8.13). The previ-

key	pos	pred	sense	EP
x_4	n	vei		h_3:_vei_n_rel(x_4)
	q	def		h_5:_def_q_rel(x_4,h_7,h_6)
e_2	v	gå		h_8:_gå_v_rel(e_2,x_4)
u_{11}	p	til		h_8:_til_p_rel(u_{11},x_4,x_{10},u_9)
x_{10}	n	by		h_{12}:_by_n_rel(x_{10})
	q	def		h_{13}:_def_q_rel(x_{10},h_{15},h_{14})

Table 8.12: Domain Example 4, MRS

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	vei	n			
h_8	e_2	gå	v		x_4	u_{11}
h_8	u_{11}	til	p		x_4,x_{10}	
h_{12}	x_{10}	by	n			

Table 8.13: Predicate Tree Table, Example 4

key	nodes	check list	return	class
itv_5	arg_1(v,gå_v_2) arg_2(n,vei_n_1)	isa(vei_n_1, path_n_1)	new	pathdescription
mod_6	arg_1(pathdescription, vei_n_1) arg_2(subpath,til_p_1)		call	
pp_2	arg_1(p,til_p_1) arg_2(n,by_n_1)	has(by_n_1, location)	new	subpath

Table 8.14: Templates for Example 4

ous examples showed an object in motion along a path. The templates used are shown in Table 8.14. According to the predicate table and the template table, the e_2 event is transformed into the class PathDescription. The pp is

transformed into a SubPath class and it is consumed by the PathDescription class. The current example shows a description of the path itself and there is not any movement involved. One complex element is returned.

PathDescription
  event(e2:gå_v$_2$)
  path
    object(x_4:vei_n_1)
    end-point(x_{10}:by_n$_1$)
  checks
    has(by_n$_1$,location)
  templates
    [itv$_5$, mod$_6$, pp$_2$]

Frank går oppover veien
Frank goes up the road (8.5)

The example (8.5) is parsed and an MRS is selected (Table 8.15). The MRS is prepared and the Predicate Tree Table is created (Table 8.16). The example uses the templates in Table 8.17. The preposition denotes a direction and SubDir class is created for the PP. The SubDir class is consumed by the

key	pos	pred	sense	EP
x_4	na	named		h_3:named_rel(x_4,frank)
	q	def		h_5:_def_q_rel(x_4,h_7,h_6)
e_2	v	gå		h_8:_gå_v_rel(e_2,x_4)
u_9	p	oppover		h_8:_oppover_p_rel(u_9,x_4,x_{10})
x_{10}	n	vei		h_{11}:_vei_n_rel(x_{10})
	q	def		h_{12}:_def_q_rel(x_{10},h_{14},h_{13})

Table 8.15: Domain Example 5, MRS

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	frank	na			
h_8	e_2	gå	v		x_4	u_9
h_8	u_9	oppover	p		x_4,x_{10}	
h_{11}	x_{10}	vei	n			

Table 8.16: Predicate Tree Table, Example 5

key	nodes	check list	return	class
pp_{13}	arg_1(p,oppover_p_1) arg_2(n,vei_n_1)	isa(oppover_p_1,dir_creation_r_1) isa(vei_n_1,path_n_1)	new	subdir
mod_{11}	arg_1(movement,_) arg_2(subdir,_)		call	
itv_2	arg_1(v,gå_v_1) arg_2(n,frank_na_1)	isa(gå_v_1,underSpecSubj_v_1) has(frank_na_1,movable)	new	movement

Table 8.17: Domain Example 5, templates

Movement class. The Path class has a path object and a path direction. One complex element is returned.

movement
- event(e_2:gå_v_1)
- subject(x_4:frank_na_1)
- path
  - direction(u_9:oppover_p_1)
  - object(x_{10}:vei_n_1)
- checks
  - isa(gå_v_1,underSpecSubj_v_1)
  - has(frank_na_1,movable)
  - isa(oppover_p_1,direction_creation_r_1)
  - isa(vei_n_1,path_n_1)
- templates
  - [itv_2,mod_{11},pp_{13}]

So far, we have used the mapping algorithm together with examples from the Change Location domain. The Movement element has the inner event structure: "departure", "ongoing movement" and "arrival". "Ongoing movement" is expressed with the Path element that has a beginning, a middle and an end. We have also seen that Movement calculates time points for its inner structure. Now, we proceed our Change Location design with the generation of roles.

8.3 The Roles

The next topic in the ontology design is about roles. Instead of general roles that fit an unspecified range of events, we will design roles with more domain

focus. Only a small part of the domain is designed. Our focus is on the preposition *med* and on how the verb arguments are related to the change location event. According to Kunnskapsforlaget's dictionary [54] the preposition *med* has 17 senses and a number of idioms. Three of these senses are used in our design.

1. The NP in the PP is transported by the subject. *The man goes with the suitcase. Mannen går med kofferten.*

2. The NP in the PP transports the subject. *The man goes by the train. Mannen går med toget.*

3. The NP in the PP is traveling together with the subject. *The man travels with the boy. Mannen går med gutten.*

A verb in the change location domain can be one of the following types (incomplete list):

- *mannerType_v*$_1$, indicates that the subject is doing the work of moving
- *beg_end_v*$_1$, an event that denotes the beginning or end of a path. An example is: *The train arrived*
- *path_v*$_1$, an object between locations. *The road to Trondheim*
- *underSpecSubj_v*$_1$, an event where it is unclear which verb argument that is doing the work

We focus on the subject and the noun in the preposition phrase and we try to classify which one is doing the work in the movement. The options are: *applied*, *work*, and *cargo*. The *applied* role reflects that an external force caused the movement. The *work* role reflects that the noun is moving by its own propulsion. The *cargo* role reflects that the noun is transported. The ball has the *applied* role in the sentence: "John kicked the ball over the fence". The sentence has two sub-events: one for the kicking event and one for the movement-over-the-fence event. If the verb type is *mannerType_v*$_1$, then the subject has the *work* role. In the remaining text of this section we will focus on the roles *work* and *cargo* together with the *underSpecSubj_v*$_1$ verb *gå* (go). A noun is classified according to the following list:

- *person*, the subject is a hyponym to *person_n*$_1$
- *vehicle*, the subject is a hyponym to *transportmiddel_n*$_1$ (vehicle)

- *item*, a movable object without propulsion

The example sentence is: "NP1 går med NP2" ("NP1 goes with NP2"). We will look at all the combinations where NP1 and NP2 are selected from the types: *person*, *vehicle*, and *item*.

Mannen går med gutten
The man goes with the boy

(8.6)

The first sentence is (8.6) and the MRS is shown in Table 8.18. Both NPs are of the type *person*. The MRS's Predicate Tree Table is shown in Table 8.19. The templates used for the verb and the PPs are shown in Table 8.20. The template itv_2 is used for the verb *gå* and the template mod_2 is used for the preposition phrase. The preposition *med* has three senses in this example.

key	pos	pred	sense	EP
x_4	n	mann		h_3:_mann_n_rel(x_4)
	q	def		h_5:_def_q_rel(x_4,h_6,h_7)
e_2	v	gå		h_8:_gå_v_rel(e_2,x_4)
u_9	p	med		h_8:_med_p_rel(u_9,e_2,x_{10})
x_{10}	n	gutt		h_{11}:_gutt_n_rel(x_{10})
	q	def		h_{12}:_def_q_rel(x_{10},h_{13},h_{14})

Table 8.18: Movement Role Example 1, MRS

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	mann	n			
h_8	e_2	gå	v		x_4	u_9
h_8	u_9	med	p		e_2,x_{10}	
h_{11}	x_{10}	gutt	n			

Table 8.19: Predicate Tree Table, Role Example 1

The first sense implies that the NP in the PP is transported by the subject. The constraint in the template is that the NP must have the *movable* feature. The second sense implies that the NP in the PP transports the subject. The constraint is that the NP must have the *propulsion* feature. The third sense implies that the NP and the subject are moving together. The NP must have the *movable* feature. Sense one and three are similar, but they have different constraints on the subject. The sec_1 template applies constraints on the PP

key	arguments	check list	return	class
itv_2	arg_1(v,gå_v_1) arg_2(n,mann_n_1)	isa(gå_v_1,underSpecSubj_v_1) has(mann_n_1,moveable)	new	movement
pp_7	arg_1(p,med_p_1) arg_2(n,gutt_n_1)	has(gutt_n_1,movable)	new	pp
pp_8	arg_1(p,med_p_2) arg_2(n,gutt_n_1)	has(gutt_n_1,propulsion)	new	pp
pp_9	arg_1(p,med_p_3) arg_2(n,gutt_n_1)	has(gutt_n_1,movable)	new	pp
mod_2	arg_1(movement, gå_v_1) arg_2(pp,med_p_3)	isa(med_p_3,work_role_p_1) isa(gå_v_1,movement_v_1)	call	
sec_1	arg_1(movement, subject,Arg_1) arg_2(prep,med_p_1)	has(Arg_1,propulsion)		
sec_2	arg_1(movement, subject,Arg_1) arg_2(prep,med_p_2)	has(Arg_1,movable)		

Table 8.20: Movement Role Example 1, template set

with med_p_1 and the subject. The sec_2 template does the same for med_p_2. The preposition med_p_3 have a constraint implemented in code of the complex element Movement: the subject and the NP's type must be equal and in the set: *person*, *vehicle*, and *item*. After the completion of the predicate tree, the Movement class is returned. The roles are inferred with the Movement's function *generateRoles()*.

After the above selection, three complex elements are returned for (8.6). The first element is with the predicate med_p$_1$. This is not the most valid element, but with a context where the boy is an infant, the element is valid.

movement
- event(e_2:gå_v$_1$)
- subject(x_4:mann_n$_1$)
- med_p$_1$(x_{10}:gutt_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(mann_n$_1$,movable)
  - isa(med_p$_1$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(mann_n$_1$,propulsion)
  - has(gutt_n$_1$,movable)
- templates
  - [itv$_2$,mod$_2$,pp$_7$,sec$_1$]
- roles
  - cargo(gutt_n$_1$)
  - work(mann_n$_1$)

The second element is with the predicate med_p$_2$ (see left column below). It is less valid than the first element. Since we use general constraints for the roles, the system overgenerates valid senses. One way to prevent this effect is to bring in a discourse and more detailed world and domain knowledge. If we switched NP and subject (see right column below), then the complex element becomes more valid. The third element is the most valid one.

movement
- event(e_2:gå_v$_1$)
- subject(x_4:mann_n$_1$)
- med_p$_2$(x_{10}:gutt_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(mann_n$_1$,movable)
  - isa(med_p$_2$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(mann_n$_1$,movable)
  - has(gutt_n$_1$,propulsion)
- templates
  - [itv$_2$,mod$_2$,pp$_8$,sec$_2$]
- roles
  - work(gutt_n$_1$)
  - cargo(mann_n$_1$)

movement
- event(e_2:gå_v$_1$)
- subject(x_4:mann_n$_1$)
- med_p$_3$(x_{10}:gutt_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(mann_n$_1$,movable)
  - isa(med_p$_3$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(gutt_n$_1$,movable)
- templates
  - [itv$_2$,mod$_2$,pp$_9$]
- roles
  - work(gutt_n$_1$)
  - work(mann_n$_1$)

Gutten går med pakken
The boy goes with the package (8.7)

The second example (8.7) has a *person* class as subject and an *item* class as PP.NP. The verb has the valid templates itv_2, mod_2, pp_7, and sec_1, as described in the previous example. The package is transported by the boy. The following complex element is returned: movement

event(e_2:gå_v$_1$)
subject(x_4:gutt_n$_1$)
med_p$_1$(x_{10}:pakke_n$_1$)
checks
isa(gå_v$_1$,underSpecSubj_v$_1$)
has(gutt_n$_1$,movable)
isa(med_p$_1$,work_role_p$_1$)
isa(gå_v$_1$,movement_v$_1$)
has(gutt_n$_1$,propulsion)
has(pakke_n$_1$,movable)
templates
[itv$_2$,mod$_2$,pp$_7$,sec$_1$]
roles
cargo(pakke_n$_1$)
work(gutt_n$_1$)

Mannen går med flyet
The man goes by airplane (8.8)

The next example (8.8) has a *person* as subject and a *vehicle* as PP.NP. The templates are described in the first example. The pp_7 and pp_8 template are valid for the preposition phrases. The roles are inferred as described in the previous example. Two complex elements are returned:

movement
- event(e_2:gå_v$_1$)
- subject(x_4:mann_n$_1$)
- med_p$_1$(x_{10}:fly_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(mann_n$_1$,movable)
  - isa(med_p$_1$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(mann_n$_1$,propulsion)
  - has(fly_n$_1$,movable)
- templates
  - [itv$_2$,mod$_2$,pp$_7$,sec$_1$]
- roles
  - cargo(fly_n$_1$)
  - work(mann_n$_1$)

movement
- event(e_2:gå_v$_1$)
- subject(x_4:mann_n$_1$)
- med_p$_2$(x_{10}:fly_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(mann_n$_1$,movable)
  - isa(med_p$_2$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(mann_n$_1$,movable)
  - has(fly_n$_1$,propulsion)
- templates
  - [itv$_2$,mod$_2$,pp$_8$,sec$_2$]
- roles
  - work(fly_n$_1$)
  - cargo(mann_n$_1$)

The first element is valid if the man can carry the airplane, and that is true only in a context where the airplane is a toy. The second element is the most valid: the airplane moves with the man inside.

Pakken går med flyet	(8.9)
The package goes with the airplane	

The (8.9) example has an *item* as subject and a *vehicle* as PP.NP. The valid template for the preposition is pp_8. The following complex object is returned:

movement
- event(e_2:gå_v_1)
- subject(x_4:pakke_n_1)
- med_p_2(x_{10}:fly_n_1)
- checks
  - isa(gå_v_1,underSpecSubj_v_1)
  - has(pakke_n_1,movable)
  - isa(med_p_2,work_role_p_1)
  - isa(gå_v_1,movement_v_1)
  - has(fly_n_1,propulsion)
- templates
  - [itv$_2$,mod$_2$,pp$_8$,sec$_2$]
- roles
  - work(fly_n_1)
  - cargo(pakke_n_1)

In this complex element, the airplane transports the package.

Flyet går med pakken
The airplane goes with the package (8.10)

The (8.10) example has a *vehicle* as subject and an *item* as PP.NP. The valid template for the preposition is pp_7. The following complex element is returned:

movement
- event(e_2:gå_v_1)
- subject(x_4:fly_n_1)
- med_p_1(x_{10}:pakke_n_1)
- checks
  - isa(gå_v_1,underSpecSubj_v_1)
  - has(fly_n_1,movable)
  - isa(med_p_1,work_role_p_1)
  - isa(gå_v_1,movement_v_1)
  - has(fly_n_1,propulsion)
  - has(pakke_n_1,movable)
- templates
  - [itv_2,mod_2,pp_7,sec_1]
- roles
  - cargo(pakke_n_1)
  - work(fly_n_1)

The airplane is transporting the package.

Flyet går med gutten
The airplane goes with the boy (8.11)

The (8.11) example has a *vehicle* as subject and a *person* as PP.NP. The sentence has pp_7 and pp_8 as valid templates for the preposition. Two complex elements are returned:

movement
- event(e_2:gå_v$_1$)
- subject(x_4:fly_n$_1$)
- med_p$_1$(x_{10}:gutt_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(fly_n$_1$,movable)
  - isa(med_p$_1$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(fly_n$_1$,propulsion)
  - has(gutt_n$_1$,movable)
- templates
  - [itv$_2$,mod$_2$,pp$_7$,sec$_1$]
- roles
  - cargo(gutt_n$_1$)
  - work(fly_n$_1$)

movement
- event(e_2:gå_v$_1$)
- subject(x_4:fly_n$_1$)
- med_p$_2$(x_{10}:gutt_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(fly_n$_1$,movable)
  - isa(med_p$_2$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(fly_n$_1$,movable)
  - has(gutt_n$_1$,propulsion)
- templates
  - [itv$_2$,mod$_2$,pp$_8$,sec$_2$]
- roles
  - work(gutt_n$_1$)
  - cargo(fly_n$_1$)

The first element is the most valid: the boy is transported by the airplane. The second element requires a context with a toy airplane, but the meaning is odd in Norwegian.

Bilen går med fergen
The car goes by ferry (8.12)

The (8.12) example has a *vehicle* as subject and a *vehicle* as PP.NP. The sentence results in the valid templates: pp_7, pp_8, and pp_9. Three complex elements are returned:

movement
- event(e_2:gå_v$_1$)
- subject(x_4:bil_n$_1$)
- med_p$_1$(x_{10}:ferge_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(bil_n$_1$,movable)
  - isa(med_p$_1$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(bil_n$_1$,propulsion)
  - has(ferge_n$_1$,movable)
- templates
  - [itv$_2$,mod$_2$,pp$_7$,sec$_1$]
- roles
  - cargo(ferge_n$_1$)
  - work(bil_n$_1$)

movement
- event(e_2:gå_v$_1$)
- subject(x_4:bil_n$_1$)
- med_p$_2$(x_{10}:ferge_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(bil_n$_1$,movable)
  - isa(med_p$_2$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(bil_n$_1$,movable)
  - has(ferge_n$_1$,propulsion)
- templates
  - [itv$_2$,mod$_2$,pp$_8$,sec$_2$]
- roles
  - work(ferge_n$_1$)
  - cargo(bil_n$_1$)

movement
- event(e_2:gå_v$_1$)
- subject(x_4:bil_n$_1$)
- med_p$_3$(x_{10}:ferge_n$_1$)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(bil_n$_1$,movable)
  - isa(med_p$_3$,work_role_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)
  - has(ferge_n$_1$,movable)
- templates
  - [itv$_2$,mod$_2$,pp$_9$]
- roles
  - work(ferge_n$_1$)
  - work(bil_n$_1$)

The first element implies that the car is transporting the ferry. If the ferry is small and the ferry fits in the car, the element is valid. The second element is the most valid. The car is transported by the ferry. The third element is less valid with the two nouns, because the ferry goes on water and the car on land. If we replace ferry with lorry the element becomes more valid.

$$\frac{\text{Pakken går med mannen}}{\text{The package goes with the man}} \tag{8.13}$$

The (8.13) example has an *item* as subject and a *person* as PP.NP. The sentence has the valid template pp_8, and one complex element is returned.

movement
- event(e_2:gå_v_1)
- subject(x_4:pakke_n_1)
- med_p_2(x_{10}:mann_n_1)
- checks
  - isa(gå_v_1,underSpecSubj_v_1)
  - has(pakke_n_1,movable)
  - isa(med_p_2,work_role_p_1)
  - isa(gå_v_1,movement_v_1)
  - has(pakke_n_1,movable)
  - has(mann_n_1,propulsion)
- templates
  - [itv$_2$,mod$_2$,pp$_8$,sec$_2$]
- roles
  - work(mann_n_1)
  - cargo(pakke_n_1)

The man is transporting the package.

Pakken går med brevet
The package goes with the letter (8.14)

The last example is (8.14). The example has an *item* as subject and an *item* as PP.NP. It has the valid template pp_9, and one complex element is returned.

movement
event(e_2:gå_v_1)
subject(x_4:pakke_n_1)
med_p_3(x_{10}:brev_n_1)
checks
isa(gå_v_1,underSpecSubj_v_1)
has(pakke_n_1,movable)
isa(med_p_3,work_role_p_1)
isa(gå_v_1,movement_v_1)
has(brev_n_1,movable)
templates
[itv$_2$,mod$_2$,pp$_9$]
roles
cargo(brev_n_1)
cargo(pakke_n_1)

The element implies a context where the package and the letter are transported together.

8.4 Adverbs, Prepositions and Object Structures

So far, we have used a simple symbol for each sense of a word, but sometimes a more detailed structure is required in order to express meaning. In the examples, "the cat sits in the car", "the cat sits on the car", and "the cat sits under the car", there are three different locations related to the car. We have a container in the car, an area on top of the car, and a space under the car. The goal of the examples is to show that some senses of the words have an internal structure and parts of this structure can be connected to other senses or parts of senses. Our focus are still on the Change Location domain. An object can contain a location and this location together with an adverb and a preposition can form a path with a direction. The adverbs have a direction that is horizontal or vertical. The prepositions have a beginning of a path or an end of a path. The nouns in this example have the object types:

- *container*, a container is typically inside another object (x has container) or the container is the object (x is-a container)
- *surface*, an area that can be a part of an object or independent
- *line*, a long and thin object; can be determined with two locations

We consider some examples.

Frank går ut av båten
Frank goes out of the boat (8.15)

We use the MRS in Table 8.21 for (8.15). The adverb is placed in the same label as the verb and the preposition. The first verb argument (x_4) is equal to the first adverb argument and the first preposition argument. The preposition and the adverb have the same key. All these characteristics are used to find the adverb phrases in our examples. The Predicate Tree Table in Table 8.22

key	pos	pred	sense	EP
x_4	na	named		h_3:named_rel(x_4,frank)
	q	def		h_5:_def_q_rel(x_4,h_6,h_7)
e_2	v	gå		h_8:_gå_v_rel(e_2,x_4)
u_{10}	adv	ut		h_8:_ut_adv_rel(u_{10},x_4,u_{11},u_9)
u_{10}	p	av		h_8:_av_p_rel(u_{10},x_4,x_{12})
x_{12}	n	båt		h_{13}:_båt_n_rel(x_{12})
	q	def		h_{14}:_def_q_rel(x_{12},h_{16},h_{15})

Table 8.21: Direction Example 1, MRS

is created from the Table 8.21. Before the Change Location event happens, Frank is inside the boat. The boat has a container where there is a location

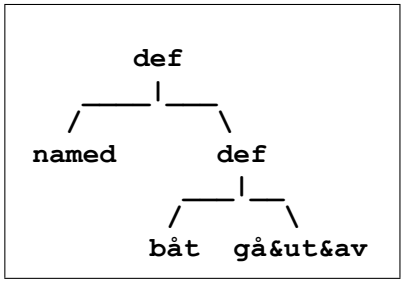


Figure 8.8: FOL

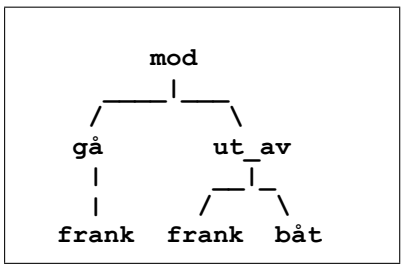


Figure 8.9: Predicate Tree

inside. The orientation of the entrance of the container is either horizontal or vertical. A large boat (ship) has the entrance on the side, and a small, open boat has the entrance at the top. To confuse the matter even more, a long ship (a Viking ship) is a ship and an open boat. The templates used for the

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	frank	na			
h_8	e_2	gå	v		x_4	u_{10}
h_{13}	x_{12}	båt	n			
h_8	u_{10}	ut_av	adv_p		x_4,x_{12}	

Table 8.22: Directional Example 1, Predicate

key	nodes	check list	return	class
itv_2	arg_1(v,gå_v_1)	isa(gå_v_1,underSpecSubj_v_1)	new	movement
	arg_2(n,frank_n_1)	has(frank_n_1,movable)		
mod_1	arg_1(movement,gå_v_1)	isa(fra_p_1,subpath_p_1)	call	
	arg_2(subpath,fra_p_1)	isa(gå_v_1,movement_v_1)		

Table 8.23: Directional Example 1, template set

Movement class is listed in Table 8.23. The tree for the logical form of the MRS is shown in Figure 8.8. The predicate tree is shown in Figure 8.9. The adverb phrase with its deeper structures has its own templates and prolog code:

```
aPart(ut_r1,horizontal).
aPart(ut_r1,vertical).
pPart(av_p1,beg).
oPart(bil_n1,container).
oPart(bil_n1,surface).
pTemplate(ut_r1,av_p1,container).

partCombo(A,P,O,Dir,Subpath,Obj):-
  aPart(A,Dir),
  pPart(P,Subpath),
  oPart(O,Obj),
  pTemplate(A,P,Obj).
```

Since the adverb and the preposition have the same key, we compile them together to one predicate. In our current example, the adverb *ut* and the

preposition *av* becomes the *ut_av* predicate (see predicate with key = u_{10} in Table 8.22). A search in the ontology is done with the adverb *ut*, the preposition *av*, and the noun *båt*. The result is: ut_r_1, av_p_1, and (båt_n_1, båt_n_2). The program searches for templates with the three arguments. The partCombo predicate is used to find templates, and the parameters for the search are:

- ut_r_1, av_p_1, båt_n_1 (List 1)
- ut_r_1, av_p_1, båt_n_2 (List 2)

List 2 is empty, but List 1 contains the elements: *AdvCombo(horizontal, beg, container)*, and *AdvCombo(vertical, beg, container)*. The first element has an adverb with a horizontal orientation, a preposition with the beginning of path feature, and the object has a container. The returned information is used to create a Subpath class and an Objectpart class. The Subpath class is either beginning, ending, or ongoing. The Objectpart class is added to the extra list. List 1 causes two complex elements to be returned.

movement
- event(e_2:gå_v_1)
- subject(x_4:frank_na$_1$)
- path
  - begin-point(id_1:lokasjon_n_1,t_1)
- checks
  - isa(gå_v_1,underSpecSubj_v_1)
  - has(frank_na$_1$,moveable)
  - isa(fra_p_1,subpath_p_1)
  - isa(gå_v_1,movement_v_1)

objectpart
- objectType:container
- object:x_{12}
- orientation:horizontal
- location:id_1

location(id_1:lokasjon_n_1)
n(x_{12}:båt_n_1)
p(k_2:av_p_1)
adv(k_1:ut_r_1)

movement
- event(e_2:gå_v_1)
- subject(x_4:frank_na$_1$)
- path
  - begin-point(id_3:lokasjon_n_1,t_1)
- checks
  - isa(gå_v_1,underSpecSubj_v_1)
  - has(frank_na$_1$,moveable)
  - isa(fra_p_1,subpath_p_1)
  - isa(gå_v_1,movement_v_1)

objectpart
- objectType:container
- object:x_{12}
- orientation:vertical
- location:id_3

location(id_3:lokasjon_n_1)
n(x_{12}:båt_n_1)
p(k_2:av_p_1)
adv(k_1:ut_r_1)

The difference between the complex elements is the orientation in the Objectpart class. The location in the Objectpart class is used in the Movement class.

Frank gikk opp i båten
Frank went into the boat (8.16)

(8.16) is parsed and the second of the two readings is shown in Table 8.24. The predicate table (Table 8.25) is created. Frank is standing beside the boat

key	pos	pred	sense	EP
x_4	na	named		h_3:named_rel(x_4,frank)
	q	def		h_5:_def_q_rel(x_4,h_6,h_7)
e_2	v	gå		h_8:_gå_v_rel(e_2,x_4)
u_{10}	adv	opp		h_8:_opp_adv_rel(u_{10},x_4,u_{11},u_9)
u_{10}	p	i		h_8:_i_p_rel(u_{10},x_4,x_{12})
x_{12}	n	båt		h_{13}:_båt_n_rel(x_{12})
	q	def		h_{14}:_def_q_rel(x_{12},h_{16},h_{15})

Table 8.24: Direction Example 2, MRS

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	frank	na			
h_8	e_2	gå	v		x_4	u_{10}
h_{13}	x_{12}	båt	n			
h_8	u_{10}	opp_i	adv_p		x_4,x_{12}	

Table 8.25: Direction Example 2, Predicate

before the event. The ontology contains the following Prolog predicates:

- pTemplate(opp_r$_1$,i_p$_1$,container).
- oPart(båt_n$_1$,container).
- pPart(i_p$_1$,end).
- aPart(opp_r$_1$,vertical).

From the predicate tree the predicate *opp_i* and *båt* are used as parameters to search the ontology for relevant senses. The search found these senses:

- opp_r$_1$, i_p$_1$, båt_n$_1$ (List 1)
- opp_r$_1$, i_p$_1$, båt_n$_2$ (List 2)

The partCombo predicate is used to find templates. List 2 is empty, but List 1 returns the element *AdvCombo(vertical, end, container)*. A SubPath class is created with the location from the container and the end point of a path. An extra object is also created for the boat. The predicate tree returns one complex element. movement
    event(e_2:gå_v$_1$)
    subject(x_4:frank_na$_1$)
    path
        end-point(id_1:lokasjon_n$_1$,t_2)
    checks
        isa(gå_v$_1$,underSpecSubj_v$_1$)
        has(frank_na$_1$,moveable)
        isa(til_p$_1$,subpath_p$_1$)
        isa(gå_v$_1$,movement_v$_1$)
objectpart
    objectType:container
    object:x_{12}
    orientation:vertical
    location:id_1
location(id_1:lokasjon_n$_1$)
n(x_{12}:båt_n$_1$)
p(k_2:i_p$_1$)
adv(k_1:opp_r$_1$) The path ends inside the container of the boat and the orientation of the movement is vertical.

Frank gikk inn i bilen
Frank went into the car (8.17)

(8.17) is parsed and an MRS is selected. The following predicate table is created: Table 8.26. Frank is outside the car before the event happens. The

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	frank	na			
h_8	e_2	gå	v		x_4	u_9
h_{11}	x_{10}	bil	n			
h_8	u_9	inn_i	adv_p		x_4,x_{10}	

Table 8.26: Direction Example 3, Predicate

templates for the adverb phrase are:

- aPart(inn_r$_1$,horizontal).
- pPart(i_p$_1$,end).
- oPart(bil_n$_1$,container).
- oPart(bil_n$_1$,surface).
- pTemplate(inn_r$_1$,i_p$_1$,container).

The template search algorithm has the arguments inn_r$_1$, i_p$_1$, bil_n$_1$, and it return the element *AdvCombo(horizontal, end, container)*. One complex element is returned. Frank is moving along a horizontal path that ends in the location id_1 inside the container of the car.

movement
event(e_2:gå_v$_1$)
subject(x_4:frank_na$_1$)
path
end-point(id_1:lokasjon_n$_1$,t_2)
checks
isa(gå_v$_1$,underSpecSubj_v$_1$)
has(frank_na$_1$,moveable)
isa(til_p$_1$,subpath_p$_1$)
isa(gå_v$_1$,movement_v$_1$)

objectpart
objectType:container
object:x_{10}
orientation:horizontal
location:id_1
location(id_1:lokasjon_n$_1$)
n(x_{10}:bil_n$_1$)
p(k_2:i_p$_1$)
adv(k_1:inn_r$_1$)

Frank gikk ut av bilen
Frank went out of the car (8.18)

(8.18) is parsed and an MRS is selected. The predicate table (Table 8.27) is created. Frank is inside the car before the event happens. The words in the

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	frank	na			
h_8	e_2	gå	v		x_4	u_{10}
h_{13}	x_{12}	bil	n			
h_8	u_{10}	ut_av	adv_p		x_4,x_{12}	

Table 8.27: Direction Example 4, Predicate

adverb phrase node have the following senses: ut_r$_1$, av_p$_1$, and bil_n$_1$. These senses have the following templates in the ontology:

- aPart(ut_r$_1$,horizontal).
- aPart(ut_r$_1$,vertical).
- pPart(av_p$_1$,beg).
- oPart(bil_n$_1$,container).
- oPart(bil_n$_1$,surface).
- pTemplate(ut_r$_1$,av_p$_1$,container).

The predicate tree returns two complex elements. One element with a horizontal movement and one element with a vertical.

movement
- event(e_2:gå_v$_1$)
- subject(x_4:frank_na$_1$)
- path
  - begin-point(id_1:lokasjon_n$_1$,t_1)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(frank_na$_1$,moveable)
  - isa(fra_p$_1$,subpath_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)

objectpart
- objectType:container
- object:x_{12}
- orientation:horizontal
- location:id_1

location(id_1:lokasjon_n$_1$)
n(x_{12}:bil_n$_1$)
p(k_2:av_p$_1$)
adv(k_1:ut_r$_1$)

movement
- event(e_2:gå_v$_1$)
- subject(x_4:frank_na$_1$)
- path
  - begin-point(id_3:lokasjon_n$_1$,t_1)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(frank_na$_1$,moveable)
  - isa(fra_p$_1$,subpath_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)

objectpart
- objectType:container
- object:x_{12}
- orientation:vertical
- location:id_3

location(id_3:lokasjon_n$_1$)
n(x_{12}:bil_n$_1$)
p(k_2:av_p$_1$)
adv(k_1:ut_r$_1$)

Frank started his movement inside the car. The most valid element is the one with horizontal direction.

Frank gikk inn på kjøkkenet
Frank went into the kitchen (8.19)

The grammar returns four readings and the third is selected. The predicate table is shown in Table 8.28. Frank is outside the kitchen before the event happens. The senses from the ontology for the words in the adverb phrase are

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	frank	na			
h_8	e_2	gå	v		x_4	u_9
h_{11}	x_{10}	kjøkken	n			
h_8	u_9	inn_på	adv_p		x_4,x_{10}	

Table 8.28: Direction Example 5, Predicate

inn_r$_1$, på_p$_1$, and kjøkken_n$_1$. These senses have the following templates for the adverb phrase:

- aPart(inn_r$_1$,horizontal).
- pPart(på_p$_1$,end).
- oPart(kjøkken_n$_1$,surface).
- pTemplate(inn_r$_1$,på_p$_1$,surface).

One complex element is returned.

movement
- event(e_2:gå_v$_1$)
- subject(x_4:frank_na$_1$)
- path
  - end-point(id_1:lokasjon_n$_1$,t_2)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(frank_na$_1$,moveable)
  - isa(til_p$_1$,subpath_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)

objectpart
- objectType:surface
- object:x_{10}
- orientation:horizontal
- location:id_1

location(id_1:lokasjon_n$_1$)
n(x_{10}:kjøkken_n$_1$)
p(k_2:på_p$_1$)
adv(k_1:inn_r$_1$)

Frank gikk ut på kjøkkenet
Frank went into the kitchen (8.20)

Four readings are returned from the parsing of (8.20). Number three is selected. The predicate table is shown in Table 8.30. The words in the adverb phrase are

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	frank	na			
h_8	e_2	gå	v		x_4	u_{10}
h_{13}	x_{12}	kjøkken	n			
h_8	u_{10}	ut_på	adv_p		x_4,x_{12}	

Table 8.29: Direction Example 6, Predicate

used to find the following senses in the ontology: ut_r$_1$, på_p$_1$, and kjøkken_n$_1$. The following templates are found for the senses:

- aPart(ut_r$_1$,horizontal).
- aPart(ut_r$_1$,vertical).
- pPart(på_p$_1$,end).
- oPart(kjøkken_n$_1$,surface).
- pTemplate(ut_r1,på_p$_1$,surface).

The predicate tree returns two complex elements.

movement
event(e_2:gå_v$_1$)
subject(x_4:frank_na$_1$)
path
end-point(id_1:lokasjon_n$_1$,t_2)
checks
isa(gå_v$_1$,underSpecSubj_v$_1$)
has(frank_na$_1$,moveable)
isa(til_p$_1$,subpath_p$_1$)
isa(gå_v$_1$,movement_v$_1$)

objectpart
objectType:surface
object:x_{12}
orientation:horizontal
location:id_1
location(id_1:lokasjon_n$_1$)
n(x_{12}:kjøkken_n$_1$)
p(k_2:på_p$_1$)
adv(k_1:ut_r$_1$)

movement
event(e_2:gå_v$_1$)
subject(x_4:frank_na$_1$)
path
end-point(id_3:lokasjon_n$_1$,t_2)
checks
isa(gå_v$_1$,underSpecSubj_v$_1$)
has(frank_na$_1$,moveable)
isa(til_p$_1$,subpath_p$_1$)
isa(gå_v$_1$,movement_v$_1$)

objectpart
objectType:surface
object:x_{12}
orientation:vertical
location:id_3
location(id_3:lokasjon_n$_1$)
n(x_{12}:kjøkken_n$_1$)
p(k_2:på_p$_1$)
adv(k_1:ut_r$_1$)

Frank gikk ut fra kjøkkenet
Frank went out of the kitchen (8.21)

(8.21) is parsed and an MRS is selected. The predicate table is shown in Table 8.30. Frank is in the kitchen before the event. The senses of the words

label	key	predicate	pos	sense	argTab	modTab
h_3	x_4	frank	na			
h_8	e_2	gå	v		x_4	u_{10}
h_{13}	x_{12}	kjøkken	n			
h_8	u_{10}	ut_fra	adv_p		x_4,x_{12}	

Table 8.30: Direction Example 7, Predicate

in the adverb phrase are: ut_r$_1$, fra_p$_1$, fra_p$_2$, and kjøkken_n$_1$. The search for templates uses these senses as arguments and returns the following information:

- ut_r1, fra_p1, kjøkken_n1, List 1

- ut_r1, fra_p2, kjøkken_n1, List 2

Two complex elements are returned. One element for a vertical movement and one for a horizontal.

movement
- event(e_2:gå_v$_1$)
- subject(x_4:frank_na$_1$)
- path
  - begin-point(id_1:lokasjon_n$_1$,t_1)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(frank_na$_1$,movable)
  - isa(fra_p$_1$,subpath_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)

objectpart
- objectType:container
- object:x_{12}
- orientation:horizontal
- location:id_1

location(id_1:lokasjon_n$_1$)
n(x_{12}:kjøkken_n$_1$)
p(k_2:fra_p$_1$)
adv(k_1:ut_r$_1$)

movement
- event(e_2:gå_v$_1$)
- subject(x_4:frank_na$_1$)
- path
  - begin-point(id_3:lokasjon_n$_1$,t_1)
- checks
  - isa(gå_v$_1$,underSpecSubj_v$_1$)
  - has(frank_na$_1$,movable)
  - isa(fra_p$_1$,subpath_p$_1$)
  - isa(gå_v$_1$,movement_v$_1$)

objectpart
- objectType:container
- object:x_{12}
- orientation:vertical
- location:id_3

location(id_3:lokasjon_n$_1$)
n(x_{12}:kjøkken_n$_1$)
p(k_2:fra_p$_1$)
adv(k_1:ut_r$_1$)

8.5 Summary

We have presented a detailed example for the mapping algorithm and a design of the Change Location domain. We selected an MRS and then we prepared the MRS and we created a predicate tree. We used the predicate tree with our mapping algorithm together with a design from the Change Location domain. The domain ontology contains a set of concepts, a type hierarchy, a set of *has* and *use* relations, the templates, and the complex domain types. Parts of the ontology are shown in Figure D.1, Figure D.2. Figure D.3, Figure D.4, and Figure D.5. The complex domain elements from the Change Location domain are: Movement, SubPath and Path. The Change Location elements expresses some event structure from statements about objects moving from one place to another. The complex elements are used to generate time points for our event

structure and to generate roles for our verb arguments. We also presented objects with structures. The objects were described with examples that had adverbs and prepositions.

Part IV

Conclusion

Chapter 9

Conclusion

9.1 Discussion

Our approach to select the top of the list of MRSs from our ranker is a simple strategy. This approach works with few domains and a good parse ranker. The alternative is to select the n-best from the list or select the MRSs that have a probability in the same range. Then we have to compute the most suitable MRS. If we scale up our system with more domains, previous stored situations and a dialogue state from a dialogue system, we create a lot of ambiguities that we need to choose from. We use the stratified pipeline architecture according to Nirenburg and Raskin [79]. This architecture is modularized and each module computes its input and deliver its output to the next module. There is no other communication between the modules. The problem with this architecture is to have a precise definition of the knowledge needed for each module and not apply too few or too much constraints. Nirenburg and Raskin [79] advocates a flat architecture that is more like a Constraint-Satisfaction architecture. The flat architecture can communicate through a blackboard architecture.[1] We cannot change the internals of the HPSG grammar parser, but we could apply this architecture to our pipeline (future research).

An MRS can be a source to a number of scope ambiguities. Some of these can be eliminated by Utool if we resolve an MRS with a file for elimination of logically equal readings. If we have a mapping of the quantifiers in the grammar to the two First-Order Logic quantifiers and some generalized quantifiers, the creation of the file for elimination of logically equal readings is easier. However, the remaining scope ambiguities can be solved with the architecture discussed above.

[1] Each module have access to the same data structure

The question-answering system WATSON [37] uses about 100 different techniques for analyzing natural language. This approach is also used by Nirenburg and Raskin [79] with their microtheories. Our approach is just one technique, so we have to look into using more overlapping techniques in our future research.

Our analysis of over two million BusTUC sentences from our previous research focus is in Appendix A.2. A large number of these sentences contain fragments with PPs and NPs. Fragments were recently implemented in NorSource, so we haven't tried to parse the BusTUC sentences. Before we try, we have to implement a preprocessing functionality that recognize time expressions, addresses, companies, bus stops, etc., because all the relevant BusTUC sentences contains at least one of these elements. As we see from the tables in Appendix A.2, the most frequent sentences are not very complex. We therefore claim that NorSource is able to parse the sentences if we add the preprocessing functionality, and if we want to reason with the BusTUC sentences we can transform an MRS into a FOREL (BusTUC's meaning representation) expression and reason with the BusTUC system.

One way to document our domain ontology is to create an OWL model, and then we could have performed a consistency check on the domain ontology and the model could have been accessed by other research groups through a known format, but we had already implemented a Prolog implementation when we used WordNet in our previous research phase. We will consider using OWL in our future research.

9.2 The Research Questions Revisited

The first research question from Section 1.2 on page 7 is "What are suitable formal representations of linguistic events and world events?"

> The Linguistic Representation's interface is the MRS. Unfortunately, MRSs come in the varieties of wanted or unwanted. A wanted MRS is an MRS that the linguist intended to create, but an unwanted MRS is an MRS that is the result of too few restrictions in parts of the grammar. So, it is important that we have a way of avoiding the unwanted MRSs, and a solution to this problem is to use a parse ranker.
>
> With a wanted MRS, our starting point for Linguistic Representation types is the Aktionsart types with verb arguments. This information is not coded in the MRS, so we need to add this information together with coercion information. Another feature of the

Aktionsart types is that they can have structure, for example sub-events. The event Movement from the Change Location domain can have the sub-events Arrival and Departure. This information is not coded in the MRS. Objects can introduce context (structure). For example, in "the cat ran into the car", the cat is located inside the container (location) that is a part of the car. Other locations connected to the car are: under the car, on top of the car, behind and in front of the car, etc. These factors are also not part of the MRS. Together these types of information call for a representation on the domain side that has structure. This is our main motivation for creating complex domain elements. In these complex domain elements we can implement algorithms for creating structure, time points, roles and states.

An MRS has different types of EPs: normal part-of-speech types, quantifiers, special predicates from the grammar[2], larger structures as lists, and predicates designed for reasoning purposes[3].

The domain ontology contains a collection of concepts for each part-of-speech type. It also contains a "is-a" hierarchy with the relations "has" and "use". The other parts of the domain ontology are templates and complex domain elements. The part-of-speech predicates are mapped to one or more senses in the domain ontology. Predicates outside the part-of-speech type must also be recognized and their function must be documented.

The second research question is: "How can linguistic events be transformed into world events for later reasoning and analysis?"

We use the template approach, where a template is a structure with constraints on the senses used for the arguments. The constrains must be true before we accept the set of senses. The constraints use the concepts in the domain ontology and the "is-a" hierarchy together with the relations "has" and "use".

We also use complex domain elements that instantiate a predicate and its arguments. As mentioned in the previous question, we can implement algorithms for creating roles and states in the complex domain elements.

Before we map underspecified predicates to specified domain predicates, we prepare the MRS. The predicates in the MRS's EPs

[2]Such as "_commsg_deict_rel", "first_position_prominent"
[3]Such as "if" and "then"

are split into predicate, sense, and part-of-speech. The prepared MRS is then transformed into a Predicate Tree Table; a structure that holds information about the connected graphs that the predicates can form. This tree is different from the scoped tree of the MRS. The Predicate Tree Table is searched for the top nodes of the graphs. The top nodes and the Predicate Tree Table are arguments in the mapping algorithm. The mapping algorithm is described in Algorithm 6.

The mapping algorithm is used in our Box World application together with Algorithm 2 which shows how we can reason with the complex domain elements. The Box World application is a simple dialogue system that uses a logic model of the Box World domain. An MRS can be transformed into a First-Order Logic formula which can be interpreted with the logic model. Dialogue acts are inferred from the complex domain elements and they are executed by the dialogue manager.

9.3 Contribution

Two pipelines have been presented, one at the underspecified discourse level and one at the specified world level.

The pipeline at the underspecified discourse level contains MRSs with discourse object and event references and underspecified predicates. This level is used when we look at a series of MRSs representing a discourse. The numbering of objects and events start at 1 and continue to the end of the discourse. Each object and event has a unique identifier in the discourse, and a simple pronoun resolution is performed.

The pipeline at the specified world level contains world references and domain ontology predicates. This means that the predicates are mapped from underspecified predicates to domain ontology predicates. The world references are obtained by an interpretation with a world model, if we have one, or by finding a similar previously stored situation. Each object and event has a unique identifier in our model of the world. For example, if we read a document about the assassination of Kennedy, we want to have unique identifiers for the event and the person. We can store the situation and reuse it later if we are looking for similar situations. If we don't have a world model we can use the series from the underspecified discourse level. This is the case when we are collecting and storing information about a domain. The assigning of world references is a manual process.

A mapping algorithm from underspecified predicates to domain specific predicates has been developed. The algorithm is used in the pipeline at the specified world level.

A domain ontology for the Change Location domain is developed. The ontology is used together with the mapping algorithm.

The tools from the DELPH-IN consortium creates "deep" grammars that offers the meaning representation MRS. This means that our work can be used by other grammars and languages.

9.4 Future Research

We want to continue our work on the natural language processing system. The grammar we are using needs improvement with longer and medium complex sentences (ten to twenty words). This is possible if we select a large amount of text from a source that writes correct Norwegian about a relevant topic. There are several options here. Some texts can be accessed from the web. Some of them are available from Really Simple Syndication (RSS) at the URL `http://www.nrk.no/rss/`, and other texts are Norwegian Wikipedia articles. Another option is text that is not online; text from books, newspapers etc.

Either way, we need to find relevant tools in the research field to process Norwegian Wikipedia files, and adopt them to our needs. A relevant candidate for mapping our concepts to is Norsk ordvev[4].

With an improved grammar that can parse semi-complex sentences (about 10-13 words), we can create a parse ranker that list the most relevant MRSs on top. This can be used to create underspecified discourses that can be used for information retrieval purposes. This means that the tool for extracting names and compound words must be able to work at least at a semi-automatic level. We hope to find a possible future solution to complex sentences, and we assume that the solution requires more software and hardware resources. We do not expect the task of compound words to be completely solved. The tool for creating a Gold standard and storing situations depends on a well functional grammar. At first, the Gold standard will be manually annotated, but we expect that this can be done semi-automatically. The tool for extracting semi-structured data from the MRSs also depends on a well functional grammar. This process is automatic. The linguistic applications depends on the previous parts and is expected to be a workbench for trying out different theories. We

[4] `http://www.nb.no/Tilbud/Forske/Spraakbanken/Tilgjengelege-ressursar/Leksikalske-ressursar`

expect the system to be functional in a few domains, but we do not expect to solve all problems with question-answering.

We need to augment out mapping algorithm to take stored situations into consideration. We also want to use more domains. The process first and select afterwards approach is not an option. A preprocessing step where we remove possible senses and domains is possible way to reduce the ambiguity problem. Another option is to keep the Predicate Tree Table and use a similarity measure to find domain elements and previous stored situations. The found elements and situations can be ranked with different dimensions, like "close as an answer to a question", "in current domain", "in domain that dialogue system accepts", etc. Then the dialogue state and the dimensions can indicate the most relevant elements.

Part V

Appendix

Appendix A

Results From Previous Phase

We have two contributions from the previous research phase. A number of sentences from BusTUC was generalized and we present a couple of tables with the most frequently used types. A model of the Aktonsart type accomplishment called the Expanded Nucleus Model was developed. The model has sub events for the process and sub states for the state.

A.1 The Expanded Nucleus Model

The Expanded Nucleus Model is based on the Aktionsart type accomplishment. The process has sub events for a beginning, a normal ongoing activity, a process break, and an end. Each sub event has a time point. The verb "depart" is

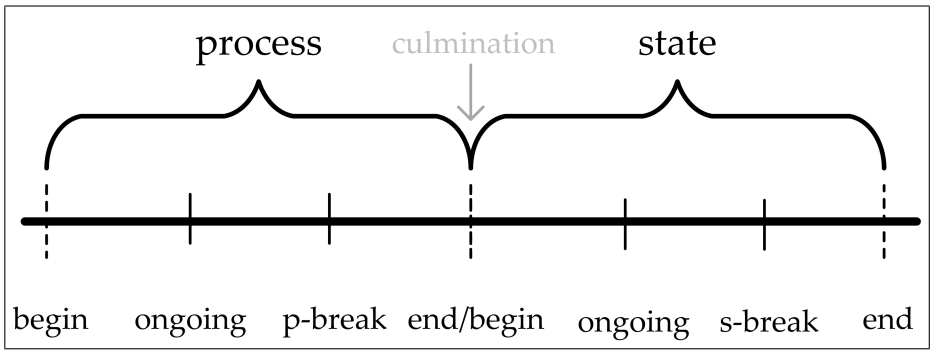


Figure A.1: Expanded Nucleus Model

an example of a begin sub event from the Change Location domain. The verb "arrive" is an example of an end sub event . The sub event for normal

ongoing activity represent many events. An example is the sentence "John walks". Here the process is ongoing and normal, but as times goes by, John is at several unmentioned locations. The p-break sub event is an abnormal termination of the process. The state has sub states for a beginning, a normal ongoing state, a state break, and an end. Each sub state has a time point. The s-break sub state is an abnormal termination of the state.

John almost reached the top (A.1)

(A.1) is a p-break example. The process of moving to the top is stopped and the culmination "atLocation(john,top)" is not activated. (A.2) is an s-break example. Here the jailing state is terminated after two years instead of four.

John was jailed for four years, but he escaped after two (A.2)

The Expanded Nucleus Model contains a structure that can be used as a guide when we design events in a domain.

A.2 Analysis Of Sentences From BusTUC

BusTUC received 2.517.047 sentences between 31 October 2010 and 1 March 2013 from its web application and SMS service. We wanted to analyze the data in order to look at the semantic structure of the incoming sentences. We parsed the sentences with BusTUC and we stored the meaning representations. BusTUC's meaning representation is called First Order Reified Event Logic (FOREL). Our strategy was to make the meaning representation more general. If we can remove the details from the representations, we can look for patterns in the remaining general representations. FOREL has the format dialogue act:expression list.

Når går bussen fra Romolslia til Dalen?
When does the bus go from Romolslia to Dalen? (A.3)

The question in (A.3) produces the following FOREL:

```
which(A): (
    romolslia isa station,
    dalen isa neighbourhood,
    B isa bus,
    A isa time,
    do/go/B/e1,
```

```
    event/real/e1,
    mod_vp/in/time/A/e1,
    mod_np/from/vehicle/place/B/romolslia,
    mod_np/to/vehicle/place/B/dalen
)
```

We use the "isa"-relation to generalize the expressions. The detailed part of the relation is replaced with the general part. For example, the FOREL elements

```
romolslia isa station,
B isa bus,
mod_np/from/vehicle/place/B/romolslia,
```

are transformed into "mod_np/from/vehicle/place/bus/station". After the replacements of the four "isa"-relations we have the following expression:

```
which(time): (
    do/go/bus/e1,
    event/real/e1,
    mod_vp/in/time/time/e1,
    mod_np/from/vehicle/place/bus/station,
    mod_np/to/vehicle/place/bus/neighbourhood
)
```

The 2.517.047 sentences were transformed into 38.657 generalized sentences. The groups with the most members are presented in Table A.2, Table A.3, Table A.4, and Table A.5. The generalized FOREL expressions were stored in a database. The numbers were selected with a GROUP BY clause and the FOREL expression were used to find the first occurrence of the sentence text. This text was augmented with generalized types like <neighbourhood>, <station>, <street>, and <clock>. (A.3) is augmented to:

"Når går bussen fra <station> til <neighbourhood>?"

6.14 % (154.568) of the sentences did not parse. The main part of these are misspelled words, unknown abbreviations, or sentences completely out of context. The forth row in Table A.2 is "Når går bussen...", which is the test sentence (fragment) from web application for the bus oracle. The answer to this "question" is: "Du må oppgi et sted i slike spørsmål" ("You must state a place in these kind of questions"). 191 groups out of total 38.657 contain more than a thousand sentences. The 191 groups have 71,83 % (1807915) of the total members. That leaves 38466 groups with less that 1000 members. These

have 28,17 % of the total members. We created a statistic for the number of FOREL elements, Table A.1.

Number of elements	Count
1	157435
2	110476
3	866900
4	385261
5	506963
6	306177
7	46936
8	14723
9	113476
10	8943
11	679
12	91
13	23
14	3
16	2

Table A.1: Number of FOREL elements

Count	Sentence Pattern
160633	<station> til <station>
143073	<station> <station>
107315	når går bussen fra <station> til <station>?
103128	Når går bussen ...
96318	<station> <neighbourhood>
93011	Når går bussen fra <station> til <neighbourhood>?
87032	<station> til <neighbourhood>
51473	fra <neighbourhood> til <station>
49785	<neighbourhood> til <station>
46589	når går neste buss fra <station> til <neighbourhood>?
44703	når går neste buss fra <station> til <station>?
35928	når går bussen fra <neighbourhood> til <station>?
33886	<station>
31130	når må jeg ta bussen fra <station> for å være på <station> <time>?
28037	neste buss til <station> fra <station>?
25897	når går buss <route> from <station>
25056	når går bussen fra <station> til <station> etter <time>?
22643	fra <neighbourhood> til <neighbourhood>
20760	<neighbourhood> til <neighbourhood>
20507	neste buss fra <station> til <neighbourhood>?
19136	når går neste buss fra <neighbourhood> til <station>
19033	når går bussen fra <neighbourhood> til <neighbourhood>
18598	fra <station> til <station> etter kl <clock>
17087	fra <station> til <station> kl <clock>
16615	når går bussen fra <station> til <neighbourhood> etter kl <clock>?
15478	når må jeg ta bussen fra <station> for å være i <neighbourhood> <clock>?
15256	Buss fra <station> til <station>
14765	<route>
14135	<station> til <station> <clock>
14014	fra <station> til <station> før <clock>
13982	<neighbourhood>
13979	<station> to <street>
13568	<street> til <station>
12989	<station> til <station> etter kl <clock>
11697	<station> <street>
11346	fra <station> til <neighbourhood> etter kl <clock>
11309	når går bussen fra <station> til <street>
10717	når har neste buss fra <neighbourhood> til <station>?
10466	fra <street> til <station>
10369	<station> til <station> senest <clock>?
10280	når går neste buss fra <station>

Table A.2: Sentence Patterns For The Highest Count, Portion I

Count	Sentence Pattern
9707	når går bussen fra <station> til <station> <clock>
9415	når går neste buss fra <neighbourhood> til <neighbourhood>?
9116	når går bussen fra <street> til <station>.
9031	<vehicle> fra <station> til <neighbourhood>.
8850	Fra <station>
8745	<station> <station> <clock>.
8640	når går bussen fra <neighbourhood> til <station> etter kl. <clock>.
8139	<station> <neighbourhood> klokken <clock>
7927	<station> til <neighbourhood> etter <clock>
7380	<station> te <neighbourhood> kl. <clock>
7279	<street>.
6754	fra <neighbourhood> til <station> etter <clock>.
6445	når går bussen fra <station> til <station> før <time>.
6422	fra <station> til <neighbourhood> før <clock>.
6371	<route> fra <station>.
6080	<vehicle> fra <neighbourhood> til <station>.
5886	neste fra <station> til <station>.
5859	<station> - <neighbourhood> <clock>
5606	når går bussen til <station>.
5590	når må jeg ta bussen fra <neighbourhood> for å være på <station> <clock>?
5352	buss fra <station>.
5218	<route> fra <station> til <station>.
4958	neste buss fra <neighbourhood> til <neighbourhood>
4901	når går <route> fra <station> til <station>?
4796	når går <vehicle> fra <neighbourhood>.
4719	når går buss fra <street> til <neighbourhood>?
4718	<station> til <neighbourhood> før <clock>
4700	når går bussen fra <station> til <neighbourhood> <clock>?
4608	<street> til <neighbourhood>.
4598	når har neste buss fra <station> til <station> etter <clock>
4442	<company>.
4391	<vehicle>.
4274	fra <neighbourhood> til <station>.
4271	når går <route> fra <station>?
4241	når må jeg ta bussen fra <station> for å være på <neighbourhood> kl <clock>?
4191	<neighbourhood> til <station> etter <clock>.
4101	<street> <neighbourhood>.
4048	<route> fra <station> på vei mot <neighbourhood>.

Table A.3: Sentence Patterns For The Highest Count, Portion II

Count	Sentence Pattern
3963	<neighbourhood> til <station> kl <clock>.
3936	fra <neighbourhood> til <station> <clock>.
3931	Når går <vehicle> fra <station> etter klokka <time>?
3786	Når går neste buss til <station> fra <street>?
3776	når går det buss fra <neighbourhood> til <neighbourhood> etter <clock>?
3592	buss fra <station> til <station> kl <clock>.
3560	når går neste buss fra <station> til <street>?
3375	neste fra <station> til <neighbourhood>.
3279	<neighbourhood> til <street>.
3261	når må jeg ta bussen fra <station> for å være ved <station> <time>?
3256	til <station>.
3235	<vehicle> til <station>.
3174	Når går bussen fra <neighbourhood> til <street>.
3170	når må jeg ta buss nr 5 fra <station> for å være i <station> kl <time>.
3134	neste buss fra <station>.
3078	fra <neighbourhood> til <station> ankomst før <clock>.
3051	når går <vehicle> fra <station> til <neighbourhood> før <time>.
3034	når går <route> fra <station> til <neighbourhood>?
3017	buss fra <station> til <station> etter <clock>.
2897	når går bussen til <neighbourhood>.
2782	når går neste buss til <station>.
2769	hvilke busser går til <station>.
2754	når går siste buss ifra <station> til <station>?
2632	Når må jeg ta buss fra <station> for å være på <station> før <clock>.
2614	<vehicle> f <neighbourhood> til <neighbourhood>.
2561	neste buss fra <street> til <station>.
2435	neste buss fra <station> til <street>.
2392	neste buss fra <station> til <neighbourhood> etter kl <clock>.
2368	<route> fra <neighbourhood> til <station>.
2327	Når må jeg ta buss fra <station> for å være på <station> til <clock>?
2252	når går siste buss fra <neighbourhood> til <station>?
2250	når går bussen fra <station> til <street> etter <clock>?
2167	når går bussen fra <street> til <station> etter klokken <clock>?
2158	fra <neighbourhood> til <neighbourhood> etter <clock>.
2068	Når går neste buss etter <clock> fra <station> til <neighbourhood>.
2027	når går neste buss fra <station> til <station> etter kl.
2014	når går <vehicle> fra <neighbourhood> til <station> <clock>.

Table A.4: Sentence Patterns For The Highest Count, Portion III

Count	Sentence Pattern
2013	når går siste buss fra <station> til <neighbourhood>?
2012	Neste buss fra <station> til <station> for å være der til <clock>.
2009	når går neste buss fra <street> til <neighbourhood>?
1982	<neighbourhood> til <station> ankomst før <clock>.
1928	buss fra <station> til <neighbourhood> etter klokka <clock>.
1859	<item>
1817	fra <station> til <station> til kl. <clock>.
1802	når går neste <route> fra <station> til <station>.
1800	når må jeg ta bussen fra <station> for å være i <street> kl <clock>?
1766	buss fra <station> som er på <station> før klokken <clock>?
1762	<route> f <neighbourhood>
1745	neste <neighbourhood> <station>.
1744	fra <neighbourhood>.
1715	når går <vehicle> fra <station> til <station> i <day> etter <clock>.
1696	når går det buss fra <station> og til <station> ca <clock>.
1689	<route>.
1685	når går <vehicle> fra <station> for å være på <station> <clock>?
1663	Når går neste <route> fra <station> til <neighbourhood>?
1649	når må jeg ta buss fra <neighbourhood> for å være i <neighbourhood> <clock>?
1646	når går <vehicle> fra <street>.
1644	fra <station> til <street> kl <clock>.
1631	<station> til <station> til <clock>.
1607	buss til <neighbourhood> fra <station> klokken <clock>.
1594	<vehicle> til <neighbourhood>.
1573	når går bussen fra <station> til <station> i <day> <clock>.
1566	<neighbourhood> til <neighbourhood> etter <clock>.
1518	til <neighbourhood>.
1489	neste buss tl <station> fra <neighbourhood> etter <clock>.
1479	når går <route> fra <neighbourhood> til <station>?
1475	fra <neighbourhood> til <neighbourhood> kl <clock>.

Table A.5: Sentence Patterns For The Highest Count, Portion IV

Appendix B

Reasoning with First-Order Logic

First-Order Logic (FOL) is a representation language for commonsense knowledge and natural language semantics. Most of the expressiveness in natural language can be handled by First-Ordered Logic. Intensional Logic is an extension of First-Ordered Logic and it can be used for natural language [44].

Formula	→	AtomicFormula
	|	(Formula Connective Formula)
	|	Quantifier Variable,... Formula
	|	¬Formula
	|	(Formula)
AtomicFormula	→	Predicate(Term,...)
Term	→	Function(Term,...)
	|	Constant
	|	Variable
Connective	→	$\wedge$ | $\vee$ | $\Rightarrow$ | $\Leftrightarrow$
Quantifier	→	$\forall$ | $\exists$
Constant	→	A | B | ...
Variable	→	x | y | ...
Predicate	→	Person | Loves | ...
Function	→	MotherTo | LocationOf | ...

Table B.1: FOL syntax

B.1 Syntax

A context-free grammar that denotes the syntax is shown in Table B.1. The syntax, in Backus-Naur form from [56], is adapted from [85]. The grammar produces formulas that are well-formed. Only well-formed formulas are formulas. A term is represents an object by representing it or pointing to it. A constant refers to a particular object. A function is returns an object. The variable is a mechanism to refer to objects (with predicates and quantifiers).

B.2 Example

The example is from [45, Example 2, Page 92]. The example shows a number of points and the connections between them. The graph is one possible state of the world and is shown in Figure B.1. The example is used to explain the

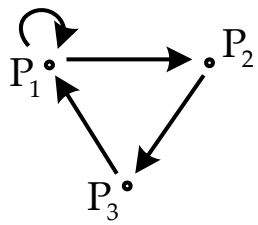


Figure B.1: Points

notion of truth (the next sections) for formulas with the quantifiers: $\forall$ and $\exists$. The objects are the points: P_1, P_2, and P_3. There are three constants: a_1, a_2, and a_3. There are two variables: x and y. There is one 2-ary predicate R. The example contains three formulas. One atomic formula and two formulas with two quantifiers each. The formulas are shown in Table B.2.

$R(a_1, a_2)$	there is a relation between point one and point two
$\forall x \exists y R(x, y)$	every point has an arrow pointing away from it
$\exists x \forall y R(x, y)$	there is a point from which arrows go to all other points

Table B.2: FOL formulas

(i)	$V_{M,g}A(t_1, ..., t_n) = 1$ iff $< [\![t_1]\!]_{M,g}, ..., [\![t_n]\!]_{M,g} > \in I(A)$
(ii)	$V_{M,g}(\neg\phi) = 1$ iff $V_{M,g}(\phi) = 0$
(iii)	$V_{M,g}(\phi \wedge \psi) = 1$ iff $V_{M,g}(\phi) = 1$ and $V_{M,g}(\psi) = 1$
(iv)	$V_{M,g}(\phi \vee \psi) = 1$ iff $V_{M,g}(\phi) = 1$ or $V_{M,g}(\psi) = 1$
(v)	$V_{M,g}(\phi \Rightarrow \psi) = 1$ iff $V_{M,g}(\phi) = 0$ or $V_{M,g}(\psi) = 1$
(vi)	$V_{M,g}(\phi \Leftrightarrow \psi) = 1$ iff $V_{M,g}(\phi) = V_{M,g}(\psi)$
(vii)	$V_{M,g}(\forall x\phi) = 1$ iff for all $d \in D, V_{M,g[x/d]}(\phi) = 1$
(viii)	$V_{M,g}(\exists x\phi) = 1$ iff there is at least one $d \in D$, $V_{M,g[x/d]}(\phi) = 1$

Figure B.2: The Valuation functions from [45]

x	$V_{M,g}(\exists yR(x,y))$		
P_1	$V_{M,g}(\exists yR(P_1,y))$	$V_{M,g}(R(P_1,P_1)) = \langle P_1,P_1\rangle \in I(R) = 1$	
		$V_{M,g}(R(P_1,P_2)) = \langle P_1,P_2\rangle \in I(R) = 1$	$\leftarrow$
		$V_{M,g}(R(P_1,P_3)) = \langle P_1,P_3\rangle \in I(R) = 0$	
P_2	$V_{M,g}(\exists yR(P_2,y))$	$V_{M,g}(R(P_2,P_1)) = \langle P_2,P_1\rangle \in I(R) = 0$	
		$V_{M,g}(R(P_2,P_2)) = \langle P_2,P_2\rangle \in I(R) = 0$	
		$V_{M,g}(R(P_2,P_3)) = \langle P_2,P_3\rangle \in I(R) = 1$	$\leftarrow$
P_3	$V_{M,g}(\exists yR(P_3,y))$	$V_{M,g}(R(P_3,P_1)) = \langle P_3,P_1\rangle \in I(R) = 1$	$\leftarrow$
		$V_{M,g}(R(P_3,P_2)) = \langle P_3,P_2\rangle \in I(R) = 0$	
		$V_{M,g}(R(P_3,P_3)) = \langle P_3,P_3\rangle \in I(R) = 0$	

Figure B.3: Valuation of Formula 2

B.3 Interpretation and Model

A model (M) is a description of a state in the world, and it is dependent on a interpretation (I). An interpretation is the non-empty set D_I of objects, the collection of constants on D_I, the collection of functions on D_I, the collection of relations on D_I. The interpretation for the example above, consist of the objects and the collections of constants, functions and relations. The objects are: $D_I = \{P_1, P_2, P_3\}$. The constants are: $I(a_1) = P_1, I(a_2) = P_2, I(a_3) = P_3$, The relations are: $I(R) = \{\langle P_1,P_1\rangle, \langle P_1,P_2\rangle, \langle P_2,P_3\rangle, \langle P_3,P_1\rangle\}$.

B.4 Semantics

A formula has the truth values: *true* (1), or *false* (0). The truth value can be found with a valuation function for a formula with an interpretation. A

function g[1] is called assignments and it contains all the variables in the domain with all its objects. $[\![t]\!]_{M,g}$ is the interpretation of a term t in a model M under assignment g. $[\![t]\!]_{M,g} = I(t)$ if t is a constant and $[\![t]\!]_{M,g} = g(t)$ if t is a variable. The valuation function V, shown in Table B.2, is described in [45, Page 97]. The truth value for $R(a_1, a_2)$ is found by a valuation of the formula:

$$V_{M,g}(R(a_1, a_2)) = 1 \text{ iff } \langle [\![a_1]\!]_{M,g}, [\![a_2]\!]_{M,g} \rangle = \langle P_1, P_2 \rangle \in I(R) = 1$$

and the formula is *true*. The truth value for $\forall x \exists y R(x, y)$ is found by a valuation of the formula, see Table B.3. The table column x shows the valuation of the $\forall x$ and next column shows the valuation of $\exists y R(x, y)$ where x is constant. The formula is *true* if for each value of x there exist at least one formula that is *true*. The table shows (the arrows) that this is the case, so the formula is *true*.

The truth value for $\exists x \forall y R(x, y)$ is found by a valuation of the formula, see Table B.3. The table column x shows the valuation of the $\exists x$ and next column

x	$V_{M,g}(\forall y R(x, y))$	
P_1	$V_{M,g}(\forall y R(P_1, y))$	$V_{M,g}(R(P_1, P_1)) = \langle P_1, P_1 \rangle \in I(R) = 1$
		$V_{M,g}(R(P_1, P_2)) = \langle P_1, P_2 \rangle \in I(R) = 1$
		$V_{M,g}(R(P_1, P_3)) = \langle P_1, P_3 \rangle \in I(R) = 0$
P_2	$V_{M,g}(\forall y R(P_2, y))$	$V_{M,g}(R(P_2, P_1)) = \langle P_2, P_1 \rangle \in I(R) = 0$
		$V_{M,g}(R(P_2, P_2)) = \langle P_2, P_2 \rangle \in I(R) = 0$
		$V_{M,g}(R(P_2, P_3)) = \langle P_2, P_3 \rangle \in I(R) = 1$
P_3	$V_{M,g}(\forall y R(P_3, y))$	$V_{M,g}(R(P_3, P_1)) = \langle P_3, P_1 \rangle \in I(R) = 1$
		$V_{M,g}(R(P_3, P_2)) = \langle P_3, P_2 \rangle \in I(R) = 0$
		$V_{M,g}(R(P_3, P_3)) = \langle P_3, P_3 \rangle \in I(R) = 0$

Table B.3: Valuation of Formula 3

shows the valuation of $\forall y R(x, y)$ where x is constant. The formula is *true* if for at least one x among all the formulas is *true*. The table show that this is not the case, so the formula is *false*.

B.5 Logic Topics

A knowledge base (KB) is a number of formulas thus: $f_1 \wedge ... \wedge f_n$. A KB is **satisfiable** if there exists at least one model for it.

[1] A tool to handle free variables.

An example is three boxes where Box_0 is on Box_1, and Box_1 is on Box_2, and Box_2 is on the floor. The boxes are shown in Figure B.4. The Mace4 program[2] is used to find models. Each box can only be on one box at the time and the model from this example is shown in Figure B.5.

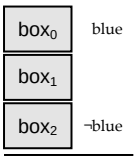


Figure B.4: Three Boxes

Box:	0	1	2
	1	1	1

On:	0	1	2
0	0	1	0
1	0	0	1
2	0	0	0

Figure B.5: Three Boxes, part 1

Blue:	0	1	2
	1	0	0

Box:	0	1	2
	1	1	1

On:	0	1	2
0	0	1	0
1	0	0	1
2	0	0	0

Figure B.6: Three Boxes Example, part 2a

Blue:	0	1	2
	1	1	0

Box:	0	1	2
	1	1	1

On:	0	1	2
0	0	1	0
1	0	0	1
2	0	0	0

Figure B.7: Three Boxes Example, part 2b

Now we introduce the predicate *Blue* that shows if the color of a box is blue or not. We know that Box_0 is blue, the colour of Box_1 is unknown, and that Box_2 is not blue. There are now two models, see Table B.6 and Table B.7.

Another example is from Table B.2. The KB here is not satisfiable, because of formula $\exists x \forall y R(x, y)$ that has the truth value *false*.

The next topic is reasoning and we want to know if a formula follows logically from another. This is written as $KB \models p$. This is called **entailment**. Entailment is connected to the notion of truth and satisfiable. A **proof** is used to find if a formula p entails KB and is written as $KB \underset{i}{\vdash} p$. This means that p is derived from KB by the inference algorithm i. The connection between

[2]Mace4 (model finder) and Prover9 (theorem prover) can be downloaded from the `http://www.cs.unm.edu/~mccune/mace4/`

proof and entailment is:

$$\text{if } KB \underset{i}{\vdash} p \text{ then } KB \vDash p$$

A method for implementing i (resolution) is demonstrated in [85, Chap 7.]. In this example the Prover9 theorem prover is used. Given the box example from above, we want to determine if $Blue(1)$ logically follows from KB. Since $KB \underset{Prover9}{\nvdash} Blue(1)$, then $KB \nvDash Blue(1)$. We try if $\neg Blue(1)$ logically follows from KB. Since $KB \underset{Prover9}{\nvdash} \neg Blue(1)$, then $KB \nvDash \neg Blue(1)$. This can be seen in Table B.6 and Table B.7. Given the formula: $\exists x \exists y(Blue(x) \wedge On(x,y) \wedge \neg Blue(y))$ (there exist a blue box that is on a box that is not blue). Does this formula logically follows from KB? The answer is yes, since $KB \underset{prover9}{\vdash} (\exists x \exists y(Blue(x) \wedge On(x,y) \wedge \neg Blue(y)))$, then $KB \vDash (\exists x \exists y(Blue(x) \wedge On(x,y) \wedge \neg Blue(y)))$.

Appendix C

Event Calculus

Event Calculus was used in our previous research process. In retrospect, we can say that the event in Event Calculus is too simple for our natural language purpose. We found that an event can have structure like sub-events, causal relations and that the structure can be coerced from one type into another. Our complex domain elements can handle these issues together with allowing algorithms that generated roles, states and time points. Mueller uses the Event Calculus reasoner described below to understand storytelling [74].

C.1 A Brief Introduction To Event Calculus

In Artificial Intelligence, commonsense reasoning is a research topic that has been concerned with the question: what changes and what remains unchanged when an event happens in the real world? It is called "the frame problem" [68] and the core of the problem is to represent all the things that stay the same and all the things that change when an event happens. In fact, only a small part of the world changes when an event happens. A thorough introduction to the Frame problem is given by Murray P. Shanahan [93]. One solution to the frame problem was to use a successor axiom that gives the changes after the axioms are executed. This approach is used in Situation Calculus [68] (successor state). The successor state can be implemented with the planning system STRIPS [39]. However, this creates a problem when more than one events happen in parallel and when the events are somehow connected. Murray P. Shanahan [93] solved the Frame Problem with Event Calculus [61] using circumscription and discrete time points. Another solution to the Frame problem was Thielscher's Fluent Calculus [96].

Eric Mueller [75] introduced the Event Calculus (EC) syntax and the com-

puter system for reasoning with EC that is used in this text. The reasoning types in his framework are deduction, abduction, postdiction, and model finding.

- Deduction, or temporal projection, detects the result after performing a series of events. An example with one event: *John lights the candle.* After this event happens the following states hold: *The candle is burning* and *the candle is emitting light.*

- Abduction and planning is the determination of a series of events that are possible given an initial state and a final state. We have the initial state: *John is in Trondheim* and the final state is: *John is in Bergen.* One possible series of events is: *John goes to the airport in Trondheim. John goes on a plane. The plane goes to Bergen Airport. John takes the bus from Bergen Airport to the city centre.*

- Postdiction is when an event leads to a state and we reason with the state prior to the event. An example is: *If we are told that Lisa picked up a newspaper and she was holding the newspaper, we may reason that Lisa was not previously holding the newspaper.*

- Model finding is the process of finding models from the given states and events. An example from Logic is in Appendix B.5.

An Event Calculus program contains the types: event, fluent, timepoint, and object. The core of Event Calculus is a collection of axioms. An event is something that happens at a discrete time point and events can take arguments such as subjects and objects. A fluent is an outcome of the event and can take arguments as well. In commonsense reasoning, things stay the same unless they are affected by an event. The term "Commonsense Law of Inertia" was initiated by John McCarthy and Lifschitz [66], according to Mueller [76, Chapter 5]. The "Commonsense Law of Inertia" is a fundamental principle in Event Calculus. The principle states that only an event causes a fluent to change, and this is expressed with the axioms *Initiates* and *Terminates*. The principle is either on or off for a fluent, and in an Event Calculus program, including the axioms, this is expressed with the predicate *ReleasedAt*. A fluent is released from the "Commonsense Law of Inertia" when the fluent is involved in continuous change. Otherwise the fluent is **not** released from the "Commonsense Law of Inertia". Fluents can also change when they are expressed directly.[1]

[1] Often used in *Initially*, the starting state of an Event Calculus program.

HoldsAt(f,t) states that fluent f is true at timepoint t
$\neg$*HoldsAt(f,t)* indicates that fluent f is false at timepoint t

Happens(e,t) states that event e happens at timepoint t

Initiates(e,f,t) states that when event e happens the fluent f becomes true and t is not released from the "Commonsense Law of Inertia"

Terminates(e,f,t) states that when event e happens the fluent f is set false and f is not released from the "Commonsense Law of Inertia"

Figure C.1: Domain Dependent axioms

An Event Calculus program is divided into the parts: initially, initiates & terminates, event history, triggers, rules, and preconditions. In the initially part, fluents can be set before the program starts. If the fluents are not set, then they can be true or false, which leads to more models. The initiates & terminates part contains the definitions of events causing fluents to become true or false. The event history part lists the events that are happening at a given timepoint. The trigger part and precondition part are explained later.

$$PersistBetween(t_1, f, t_2) \stackrel{def}{\equiv} \neg\exists(t)[ReleasedAt(f, t) \wedge t_1 < t \leq t_2]$$

$$Clipped(t_1, f, t_2) \stackrel{def}{\equiv} \exists(e, t)[Happens(e, t) \wedge t_1 \leq t < t_2 \wedge Terminates(e, f, t)]$$

$$Declipped(t_1, f, t_2) \stackrel{def}{\equiv} \exists(e, t)[Happens(e, t) \wedge t_1 \leq t < t_2 \wedge Initiates(e, f, t)]$$

$$HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge PersistBetween(t_1, f, t_2) \wedge \neg Clipped(t_1, f, t_2) \Rightarrow HoldsAt(f, t_2)$$

$$\neg HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge PersistBetween(t_1, f, t_2) \wedge \neg Declipped(t_1, f, t_2) \Rightarrow \neg HoldsAt(f, t_2)$$

Figure C.2: Dependent and independent axioms

The EC program can contain rules such as: a person can be at only one place at the time.

The Event Calculus axioms can be divided into domain dependent and

name	formula
Fluent precondition 1	$\gamma \rightarrow \mathit{Initiates}(e,f,t)$
Fluent precondition 2	$\gamma \rightarrow \mathit{Terminates}(e,f,t)$
Event precondition	$\mathit{Happens}(e,t) \rightarrow \gamma$
State constraints	if γ_1 and γ_2 are conditions then $\gamma_1 \rightarrow \gamma_2$ is a state constraint
Trigger axiom	$\gamma \rightarrow \mathit{Happens}(e,t)$
Trigger fluents 1	$\gamma \rightarrow \mathit{Happens}(e,t)$ $\mathit{Initiates}(e,f,t)$
Trigger fluents 2	$\gamma \rightarrow \mathit{Happens}(e,t)$ $\mathit{Terminates}(e,f,t)$

Table C.1: Expressing context in Event Calculus

domain independent axioms. The domain dependent axioms are shown in Figure C.1. The *Initiates* and *Terminates* axioms are bringing the fluent back into the "Commonsense Law of Inertia". The domain independent axioms are the building blocks for the domain dependent axioms. For example, the axiom *HoldsAt(f,t)* contains the domain independent axioms *PersistBetween*, *ReleasedAt*, *Clipped*, and *Declipped*. The connections between the axioms are shown in Figure C.2. Further details about the axioms are described in [75, Section 2.3].

Event Calculus has the ability to express context with: *fluent preconditions*, *event preconditions*, *state constraints*, *trigger axioms*, and *trigger fluents*. The formulas are given in Table C.1. Fluent precondition 1 is a positive fluent precondition: γ is a condition, e is an event, f is a fluent, and t is a timepoint. The axiom *Initiates* is performed if the condition is true. Fluent precondition 2 is a negative fluent precondition and γ, e, f, and t are the same as in Fluent precondition 1. The axiom *Terminates* is performed if the condition is true. An event precondition is a requirement that must be satisfied before an event can happen. The state constraint expresses conditions that must hold between states. An example is to express that a person can only drive one car at a time. The trigger axiom is using a condition to make an event occur. An example is an alarm clock. If *AlarmTime* and *AlarmOn* are true, the *StartsBeeping* event occurs. The trigger fluents are represented with two formulas. Trigger fluent 1 states that if condition γ is true when event e happens at t, then f will be true after t. Trigger fluent 2 states that if condition γ is true when event e happens at t, then f will be false after t.

Commonsense reasoning can imply reasoning with:

- **Indirect effects**. An indirect effect is when one object changes because another object changed. An example is the location of a book when it is carried by a person. The location of the book depends on the location of the person.

- **Delayed effects**. A delayed effect is when a process starts and after a while some state occurs or the process ends in a state. An example is when somebody drops an object, the object falls and the object hits the ground. The delayed effect is the moment the object hits the ground.

- **Continuous change**. The continuous change can be explained with a falling object. The fluent for the object can be *Height(o,h)*. The fluent can be set to true when a drop event happens. When the object hits the ground, an event trigger fires. The event can be *HitGround(o)*. The *Height* fluent is released from the "Commonsense Law of Inertia" when the object is falling. The fluent is no longer determined by an event, but by the function $H - \frac{1}{2}G(t - t_1)^2$, where H is the starting height and G is the acceleration due to gravity (9.8 m/s^2). The *Height* fluent is only released during the continuous change.

	formula
1	*Initiates(Drop(a,o),Falling(o),t)*
2	*Releases(Drop(a,o),Height(o,h),t)*
3	*HoldsAt(Height(o,h),*t_1*)* $\wedge$
	Happens(Drop(a,o),t_1*)* $\wedge 0 > t_2 \wedge$
	$\neg$ *StoppedIn(*t_1*,Falling(o),*$t_1 + t_2$*)* $\rightarrow$
	HoldsAt(Height(o,h-$\frac{1}{2}G(t_2)^2), t_1 + t_2$*)*
4	*Terminates(HitGround(o),Falling(o),t)*
5	*HoldsAt(Falling(o),t)* $\wedge$
	HoldsAt(Height(o,0),t) $\rightarrow$
	Happens(HitGround(o),t)
6	*HoldsAt(Height(o,h),t)* $\rightarrow$
	Initiates(HitGround(o),Height(o,h),t)
7	*HoldsAt(Height(o,h),*t_1*)* $\rightarrow$
	Trajectory(Falling(o),t_1*,Height(*$o, h - \frac{1}{2}G(t_2)^2), t_2$*)*

Table C.2: Falling Object with Events

In order to express some of the notions described above the following predicates are introduced:

- *Release(e,f,t)* indicates that when event e happens, the fluent f is released from the "Commonsense Law of Inertia" after timepoint t.

- *ReleasedAt(f,t)* indicates that fluent f is released from the "Commonsense Law of Inertia" at timepoint t. $\neg$*ReleasedAt(f,t)* indicates that fluent f is *not* released from the "Commonsense Law of Inertia" at timepoint t.

- *Trajectory*(f_1, t_1, f_2, t_2) indicates that if fluent f_1 is initiated by an event at timepoint t_1 and if $t_2 > 0$, then the fluent f_2 is set to true at timepoint $t_1 + t_2$.

- *AntiTrajectory*(f_1, t_1, f_2, t_2) indicates that if fluent f_1 is terminated by an event that happens at timepoint t_1, and $t_2 > 0$, then the fluent f_2 is set to true at timepoint $t_1 + t_2$.

The falling object example is from [76, chap. 7]. The formulas are given in Table C.2. Formula number 3 can be replaced by formula number 7 in the table. The trajectory predicate shows that the object hits the ground and is at zero height at t_2. Formula number 6 turns on[2] the "Commonsense law of inertia" for the fluent Height(o,h).

C.2 Similarities And Differences

In Event Calculus, an event causes a fluent to change. Some Aktionsart types culminate in a state that is mentioned in the sentence. The terms fluent and state are equivalent. When they occur is different. A culminating state is something that starts after an event happens. This is shown in Moens' and Steedman's nucleus model. Event Calculus can also express a culminating state, but in addition, EC can have a fluent terminated when an event happens and a fluent initiated after an event starts.

John runs from the fence to the tree (C.1)

In (C.1) and expressed with Aktionsart types, the event is coerced into an accomplishment that culminates in the state *atLocation(john,tree)*. The same sentence expressed in Event Calculus can terminate the previous fluent *atLocation(john,fence)* when the process starts in addition to initiating the fluent *isMoving(john)*. When the process ends, the fluent *isMoving(john)* is terminated and the fluent *atLocation(john,tree)* is initiated. The similarities between

[2] Initiates and Terminates do this.

the Aktionsart types and Event Calculus are that both can express the culminating state, and the difference is that EC can express more fluents than the Aktionsart types. Both can express a distinct culminating state. However, a fuzzy culminating state is problematic in both cases. One explanation can be that a state or a fluent is expressed in a logical form. A logical form does not express fuzziness very well. A fuzzy example is in (C.2).

John cooled the soup in two minutes (C.2)

This is an *accomplishment* that lasts for two minutes and it culminates in a state. The temperatures from the beginning and at the end of the process are from some temperature scale, but their values are not known. The best we can do is to guess. The notion of a cool soup can vary among different people.

Symbol	**Meaning**	**Symbol**	**Meaning**
`+`	addition	`!=`	not equal to
`-`	subtraction, negation	`|`	disjunction (OR, $\vee$)
`*`	multiplication	`&`	conjunction (AND, $\wedge$)
`/`	division	`!`	logical negation
`%`	modulus	`->`	implication
`<`	less than	`<->`	bi-implication
`<=`	less than or equal to	`{ }`	existential quantification ($\exists$)
`=`	equal to	`[ ]`	universal quantification ($\forall$)
`>=`	greater than or equal to	`( )`	grouping
`>`	greater than	`,`	separator

Table C.3: Syntax for the Event Calculus Reasoner

C.3 The Event Calculus Reasoner

The Discrete Event Calculus reasoner (DEC) [76, Chapter 13] [75] is used to show the syntax and is used to run the Event Calculus code. DEC[3] converts the Event Calculus expressions into a satisfiability problem (SAT) using the DIMACS format [34] and the result is used by a model finder and then DEC decodes the model finder's result. In our implementation, DEC uses a combination of the Relsat Sat solver [7] and the Walksat solver [92]. If the Relsat solver does not find any models, then the Walksat solver computes a near-miss model.

[3]The reasoner can be downloaded from: `http://decreasoner.sourceforge.net/`

The syntax of the reasoner is according to [75], quote: "Sentences are expressed in the language of many-sorted first-order predicate calculus with sub-sort orders [102]". This means that:

1. Sorts can be sub-sorts of other sorts.
2. Every variable, constant, and function symbol has an associated sort.
3. Every argument position of every function and predicate symbol has an associated sort.
4. For a term to fill an argument position of a function or predicate symbol, the sort associated with the term must be a sub-sort of the sort associated with the argument position.

The syntax for the reasoner is shown in Table C.3.

C.4 Event Calculus, Source Code

In the program, Mary is of type *person*. The statement:

[person,time] Initiates(BeginProcess(person), Processing(person), time)

expressed in natural language is: *For all persons and for all time points: the event BeginProcess(person) at time results in the state Processing(person) at time + 1*

C.4.1 Our Aktionsart Program

```
1  ; *** declarations ***
2  load foundations/Root.e
3  load foundations/EC.e
4
5  sort person
6  person Mary
7
8  fluent State(person)
9  fluent Processing(person)
10
11 event BeginProcess(person)
12 event EndProcess(person)
13 event BreakProcess(person)
14 event EndState(person)
15 event BreakState(person)
16
17 ; *** Init & Term ***
```

```
18 [person,time]
19   Initiates(BeginProcess(person),
20             Processing(person), time).
21 [person,time]
22   Terminates(EndProcess(person),
23              Processing(person), time).
24 [person,time]
25   Initiates(EndProcess(person),
26             State(person), time).
27 [person,time]
28   Terminates(BreakProcess(person),
29              Processing(person), time).
30 [person,time]
31   Terminates(EndState(person),
32              State(person), time).
33 [person,time]
34   Terminates(BreakState(person),
35              State(person), time).
36
37 ; *** Event history ***
38 Happens( BeginProcess(Mary), 0).
39 ;Happens( BreakProcess(Mary), 1).
40 ;Happens( BreakState(Mary), 5).
41
42 ; *** Event triggers ***
43 HoldsAt(Processing(Mary),3) ->
44   Happens(EndProcess(Mary),3).
45
46 HoldsAt(State(Mary),6) ->
47   Happens(EndState(Mary),6).
48
49 ; *** Preconditions ***
50 [person, time]
51   Happens( BeginProcess(person), time) ->
52   !HoldsAt(Processing(person),time).
53 [person, time]
54   Happens( EndProcess(person), time) ->
55   HoldsAt(Processing(person),time).
56
57 ; *** Initially ***
58 !HoldsAt( State(Mary), 0).
59 !HoldsAt( Processing(Mary), 0).
60
61 ; *** Additional stuff ***
62 completion Happens
63 range time 0 7
64 range offset 1 1
```

Listing C.1: Our Aktionsart Program

C.4.2 Lambalgen and Hamm's Building Program

```
1 ; *** declarations ***
2 load foundations/Root.e
3 load foundations/EC.e
4
```

```
5  sort agent
6  sort prog: integer
7
8  agent Hamm
9
10 fluent Build(agent)
11 fluent House(prog)
12
13 event Start(agent)
14 event Finish(agent)
15
16 ; *** Init & Term **
17 [agent,time]
18   Initiates(Start(agent),Build(agent),time).
19 [agent,prog,time]
20   Releases(Start(agent),House(prog),time).
21 [agent,time]
22   Terminates(Finish(agent),Build(agent),time).
23 [agent,prog,time]
24   HoldsAt(House(prog),time) ->
25   Initiates(Finish(agent),House(prog),time).
26
27 ; *** Event history ***
28 Happens(Start(Hamm),0).
29
30 ; *** Event triggers ***
31 [agent,time]
32   HoldsAt(Build(agent),time) &
33   HoldsAt(House(3),time) ->
34   Happens(Finish(agent),time).
35
36 ; *** Rules ***
37 ; House can not be at different prog at the same time
38 [prog1,prog2,time]
39   HoldsAt(House(prog1),time) &
40   HoldsAt(House(prog2),time) ->
41   prog1=prog2.
42 ; algorithm for House fluent
43 [agent,prog1,prog2,offset,time]
44   HoldsAt(House(prog1),time) &
45   prog2 = (prog1 + offset) ->
46   Trajectory(Build(agent),time,House(prog2),offset).
47
48 ; *** Initially ***
49 !HoldsAt(Build(Hamm),0).
50 HoldsAt(House(0),0).
51
52 ; *** Additional stuff ***
53 completion Happens
54 range time 0 5
55 range prog 0 3
56 range offset 1 3
57 ; End of file.
```

Listing C.2: Lambalgen and Hamm's Building Program

C.5 Event Calculus, Program Trace

C.5.1 Trace, Lambalgen and Hamm's Building Program

```
1  Copyright (c) 2005 IBM Corporation and others.
2  All rights reserved. This program and the accompanying materials
3  are made available under the terms of the Common Public License v1.0
4  which accompanies this distribution, and is available at
5  http://www.eclipse.org/legal/cpl-v10.html
6
7  Contributors:
8  IBM - Initial implementation
9
10 Discrete Event Calculus Reasoner 1.0
11 loading EC_bruland/build.e
12 loading foundations/Root.e
13 loading foundations/EC.e
14 72 variables and 347 clauses
15 relsat solver
16 the cmd:solvers\relsat -#10 ./tmp/solver.input_3540_6 > ./tmp/solver.
       output_3540_7
17 1 model
18 ---
19 model 1:
20 0
21 House(0).
22 Happens(Start(Hamm), 0).
23 1
24 -House(0).
25 +Build(Hamm).
26 +House(1).
27 2
28 -House(1).
29 +House(2).
30 3
31 -House(2).
32 +House(3).
33 Happens(Finish(Hamm), 3).
34 4
35 -Build(Hamm).
36 5
37 P
38 ReleasedAt(House(0), 1).
39 ReleasedAt(House(0), 2).
40 ReleasedAt(House(0), 3).
41 ReleasedAt(House(0), 4).
42 ReleasedAt(House(0), 5).
43 ReleasedAt(House(1), 1).
44 ReleasedAt(House(1), 2).
45 ReleasedAt(House(1), 3).
46 ReleasedAt(House(1), 4).
47 ReleasedAt(House(1), 5).
48 ReleasedAt(House(2), 1).
49 ReleasedAt(House(2), 2).
50 ReleasedAt(House(2), 3).
```

```
51 ReleasedAt(House(2), 4).
52 ReleasedAt(House(2), 5).
53 ReleasedAt(House(3), 1).
54 ReleasedAt(House(3), 2).
55 ReleasedAt(House(3), 3).
56 !Happens(Finish(Hamm), 0).
57 !Happens(Finish(Hamm), 1).
58 !Happens(Finish(Hamm), 2).
59 !Happens(Finish(Hamm), 4).
60 !Happens(Finish(Hamm), 5).
61 !Happens(Start(Hamm), 1).
62 !Happens(Start(Hamm), 2).
63 !Happens(Start(Hamm), 3).
64 !Happens(Start(Hamm), 4).
65 !Happens(Start(Hamm), 5).
66 !ReleasedAt(Build(Hamm), 0).
67 !ReleasedAt(Build(Hamm), 1).
68 !ReleasedAt(Build(Hamm), 2).
69 !ReleasedAt(Build(Hamm), 3).
70 !ReleasedAt(Build(Hamm), 4).
71 !ReleasedAt(Build(Hamm), 5).
72 !ReleasedAt(House(0), 0).
73 !ReleasedAt(House(1), 0).
74 !ReleasedAt(House(2), 0).
75 !ReleasedAt(House(3), 0).
76 !ReleasedAt(House(3), 4).
77 !ReleasedAt(House(3), 5).
78 EC: 7 predicates, 0 functions, 0 fluents, 0 events, 0 axioms
79 Root: 0 predicates, 0 functions, 0 fluents, 0 events, 0 axioms
80 build: 0 predicates, 0 functions, 2 fluents, 2 events, 10 axioms
81 encoding 0.4s
82 solution 0.0s
83 total 0.6s
```

Listing C.3: Trace, Lambalgen and Hamm's Building Program

C.5.2 Normal Trace, Our Aktionsart Program

```
1  Copyright (c) 2005 IBM Corporation and others.
2  All rights reserved. This program and the accompanying materials
3  are made available under the terms of the Common Public License v1.0
4  which accompanies this distribution, and is available at
5  http://www.eclipse.org/legal/cpl-v10.html
6
7  Contributors:
8  IBM - Initial implementation
9
10 Discrete Event Calculus Reasoner 1.0
11 loading EC_bruland/aspect.e
12 loading foundations/Root.e
13 loading foundations/EC.e
14 72 variables and 200 clauses
15 relsat solver
16 the cmd:solvers\relsat -#10 ./tmp/solver.input_3584_6 > ./tmp/solver.
       output_3584_7
17 1 model
```

```
18 ---
19 model 1:
20 0
21 Happens(BeginProcess(Mary), 0).
22 1
23 +Processing(Mary).
24 2
25 3
26 Happens(EndProcess(Mary), 3).
27 4
28 -Processing(Mary).
29 +Culmination(Mary).
30 5
31 6
32 Happens(EndState(Mary), 6).
33 7
34 -Culmination(Mary).
35 P
36 !Happens(BeginProcess(Mary), 1).
37 !Happens(BeginProcess(Mary), 2).
38 !Happens(BeginProcess(Mary), 3).
39 !Happens(BeginProcess(Mary), 4).
40 !Happens(BeginProcess(Mary), 5).
41 !Happens(BeginProcess(Mary), 6).
42 !Happens(BeginProcess(Mary), 7).
43 !Happens(BreakProcess(Mary), 0).
44 !Happens(BreakProcess(Mary), 1).
45 !Happens(BreakProcess(Mary), 2).
46 !Happens(BreakProcess(Mary), 3).
47 !Happens(BreakProcess(Mary), 4).
48 !Happens(BreakProcess(Mary), 5).
49 !Happens(BreakProcess(Mary), 6).
50 !Happens(BreakProcess(Mary), 7).
51 !Happens(BreakState(Mary), 0).
52 !Happens(BreakState(Mary), 1).
53 !Happens(BreakState(Mary), 2).
54 !Happens(BreakState(Mary), 3).
55 !Happens(BreakState(Mary), 4).
56 !Happens(BreakState(Mary), 5).
57 !Happens(BreakState(Mary), 6).
58 !Happens(BreakState(Mary), 7).
59 !Happens(EndProcess(Mary), 0).
60 !Happens(EndProcess(Mary), 1).
61 !Happens(EndProcess(Mary), 2).
62 !Happens(EndProcess(Mary), 4).
63 !Happens(EndProcess(Mary), 5).
64 !Happens(EndProcess(Mary), 6).
65 !Happens(EndProcess(Mary), 7).
66 !Happens(EndState(Mary), 0).
67 !Happens(EndState(Mary), 1).
68 !Happens(EndState(Mary), 2).
69 !Happens(EndState(Mary), 3).
70 !Happens(EndState(Mary), 4).
71 !Happens(EndState(Mary), 5).
72 !Happens(EndState(Mary), 7).
73 !ReleasedAt(Culmination(Mary), 0).
```

```
!ReleasedAt(Culmination(Mary), 1).
!ReleasedAt(Culmination(Mary), 2).
!ReleasedAt(Culmination(Mary), 3).
!ReleasedAt(Culmination(Mary), 4).
!ReleasedAt(Culmination(Mary), 5).
!ReleasedAt(Culmination(Mary), 6).
!ReleasedAt(Culmination(Mary), 7).
!ReleasedAt(Processing(Mary), 0).
!ReleasedAt(Processing(Mary), 1).
!ReleasedAt(Processing(Mary), 2).
!ReleasedAt(Processing(Mary), 3).
!ReleasedAt(Processing(Mary), 4).
!ReleasedAt(Processing(Mary), 5).
!ReleasedAt(Processing(Mary), 6).
!ReleasedAt(Processing(Mary), 7).
EC: 7 predicates, 0 functions, 0 fluents, 0 events, 0 axioms
Root: 0 predicates, 0 functions, 0 fluents, 0 events, 0 axioms
aspect: 0 predicates, 0 functions, 2 fluents, 5 events, 13 axioms
encoding 0.1s
solution 0.0s
total 0.3s
```

Listing C.4: Normal Trace, Our Aktionsart Program

C.5.3 P-break Trace, Our Aktionsart Program

```
Copyright (c) 2005 IBM Corporation and others.
All rights reserved. This program and the accompanying materials
are made available under the terms of the Common Public License v1.0
which accompanies this distribution, and is available at
http://www.eclipse.org/legal/cpl-v10.html

Contributors:
IBM - Initial implementation

Discrete Event Calculus Reasoner 1.0
loading EC_bruland/aspect.e
loading foundations/Root.e
loading foundations/EC.e
72 variables and 200 clauses
relsat solver
the cmd:solvers\relsat -#10 ./tmp/solver.input_3840_6 > ./tmp/solver.
    output_3840_7
1 model
---
model 1:
0
Happens(BeginProcess(Mary), 0).
1
+Processing(Mary).
Happens(BreakProcess(Mary), 1).
2
-Processing(Mary).
3
4
5
```

```
6
7
P
!Happens(BeginProcess(Mary), 1).
!Happens(BeginProcess(Mary), 2).
!Happens(BeginProcess(Mary), 3).
!Happens(BeginProcess(Mary), 4).
!Happens(BeginProcess(Mary), 5).
!Happens(BeginProcess(Mary), 6).
!Happens(BeginProcess(Mary), 7).
!Happens(BreakProcess(Mary), 0).
!Happens(BreakProcess(Mary), 2).
!Happens(BreakProcess(Mary), 3).
!Happens(BreakProcess(Mary), 4).
!Happens(BreakProcess(Mary), 5).
!Happens(BreakProcess(Mary), 6).
!Happens(BreakProcess(Mary), 7).
!Happens(BreakState(Mary), 0).
!Happens(BreakState(Mary), 1).
!Happens(BreakState(Mary), 2).
!Happens(BreakState(Mary), 3).
!Happens(BreakState(Mary), 4).
!Happens(BreakState(Mary), 5).
!Happens(BreakState(Mary), 6).
!Happens(BreakState(Mary), 7).
!Happens(EndProcess(Mary), 0).
!Happens(EndProcess(Mary), 1).
!Happens(EndProcess(Mary), 2).
!Happens(EndProcess(Mary), 3).
!Happens(EndProcess(Mary), 4).
!Happens(EndProcess(Mary), 5).
!Happens(EndProcess(Mary), 6).
!Happens(EndProcess(Mary), 7).
!Happens(EndState(Mary), 0).
!Happens(EndState(Mary), 1).
!Happens(EndState(Mary), 2).
!Happens(EndState(Mary), 3).
!Happens(EndState(Mary), 4).
!Happens(EndState(Mary), 5).
!Happens(EndState(Mary), 6).
!Happens(EndState(Mary), 7).
!ReleasedAt(Culmination(Mary), 0).
!ReleasedAt(Culmination(Mary), 1).
!ReleasedAt(Culmination(Mary), 2).
!ReleasedAt(Culmination(Mary), 3).
!ReleasedAt(Culmination(Mary), 4).
!ReleasedAt(Culmination(Mary), 5).
!ReleasedAt(Culmination(Mary), 6).
!ReleasedAt(Culmination(Mary), 7).
!ReleasedAt(Processing(Mary), 0).
!ReleasedAt(Processing(Mary), 1).
!ReleasedAt(Processing(Mary), 2).
!ReleasedAt(Processing(Mary), 3).
!ReleasedAt(Processing(Mary), 4).
!ReleasedAt(Processing(Mary), 5).
!ReleasedAt(Processing(Mary), 6).
```

```
!ReleasedAt(Processing(Mary), 7).
EC: 7 predicates, 0 functions, 0 fluents, 0 events, 0 axioms
Root: 0 predicates, 0 functions, 0 fluents, 0 events, 0 axioms
aspect: 0 predicates, 0 functions, 2 fluents, 5 events, 14 axioms
encoding 0.1s
solution 0.0s
total 0.4s
```

Listing C.5: P-break Trace, Our Aktionsart Program

C.5.4 S-break Trace, Our Aktionsart Program

```
Copyright (c) 2005 IBM Corporation and others.
All rights reserved. This program and the accompanying materials
are made available under the terms of the Common Public License v1.0
which accompanies this distribution, and is available at
http://www.eclipse.org/legal/cpl-v10.html

Contributors:
IBM - Initial implementation

Discrete Event Calculus Reasoner 1.0
loading EC_bruland/aspect.e
loading foundations/Root.e
loading foundations/EC.e
72 variables and 200 clauses
relsat solver
the cmd:solvers\relsat -#10 ./tmp/solver.input_3948_6 > ./tmp/solver.
    output_3948_7
1 model
---
model 1:
0
Happens(BeginProcess(Mary), 0).
1
+Processing(Mary).
2
3
Happens(EndProcess(Mary), 3).
4
-Processing(Mary).
+State(Mary).
5
Happens(BreakState(Mary), 5).
6
-State(Mary).
7
P
!Happens(BeginProcess(Mary), 1).
!Happens(BeginProcess(Mary), 2).
!Happens(BeginProcess(Mary), 3).
!Happens(BeginProcess(Mary), 4).
!Happens(BeginProcess(Mary), 5).
!Happens(BeginProcess(Mary), 6).
!Happens(BeginProcess(Mary), 7).
!Happens(BreakProcess(Mary), 0).
```

```
!Happens(BreakProcess(Mary), 1).
!Happens(BreakProcess(Mary), 2).
!Happens(BreakProcess(Mary), 3).
!Happens(BreakProcess(Mary), 4).
!Happens(BreakProcess(Mary), 5).
!Happens(BreakProcess(Mary), 6).
!Happens(BreakProcess(Mary), 7).
!Happens(BreakState(Mary), 0).
!Happens(BreakState(Mary), 1).
!Happens(BreakState(Mary), 2).
!Happens(BreakState(Mary), 3).
!Happens(BreakState(Mary), 4).
!Happens(BreakState(Mary), 6).
!Happens(BreakState(Mary), 7).
!Happens(EndProcess(Mary), 0).
!Happens(EndProcess(Mary), 1).
!Happens(EndProcess(Mary), 2).
!Happens(EndProcess(Mary), 4).
!Happens(EndProcess(Mary), 5).
!Happens(EndProcess(Mary), 6).
!Happens(EndProcess(Mary), 7).
!Happens(EndState(Mary), 0).
!Happens(EndState(Mary), 1).
!Happens(EndState(Mary), 2).
!Happens(EndState(Mary), 3).
!Happens(EndState(Mary), 4).
!Happens(EndState(Mary), 5).
!Happens(EndState(Mary), 6).
!Happens(EndState(Mary), 7).
!ReleasedAt(State(Mary), 0).
!ReleasedAt(State(Mary), 1).
!ReleasedAt(State(Mary), 2).
!ReleasedAt(State(Mary), 3).
!ReleasedAt(State(Mary), 4).
!ReleasedAt(State(Mary), 5).
!ReleasedAt(State(Mary), 6).
!ReleasedAt(State(Mary), 7).
!ReleasedAt(Processing(Mary), 0).
!ReleasedAt(Processing(Mary), 1).
!ReleasedAt(Processing(Mary), 2).
!ReleasedAt(Processing(Mary), 3).
!ReleasedAt(Processing(Mary), 4).
!ReleasedAt(Processing(Mary), 5).
!ReleasedAt(Processing(Mary), 6).
!ReleasedAt(Processing(Mary), 7).
EC: 7 predicates, 0 functions, 0 fluents, 0 events, 0 axioms
Root: 0 predicates, 0 functions, 0 fluents, 0 events, 0 axioms
aspect: 0 predicates, 0 functions, 2 fluents, 5 events, 14 axioms
encoding 0.1s
solution 0.0s
total 0.4s
```

Listing C.6: S-break Trace, Our Aktionsart Program

Appendix D

Change Location Domain Ontology

The domain ontology contains a type hierarchy, a set of *has* and *use* relations, the templates, and the complex types (objet-oriented types). The complex domain elements from the change location domain are: Movement, SubPath and Path.

Movement
- event: ParseElem
- subject: ParseElem
- object: ParseElem
- path: Path
- cargo: Parse1
- vehicle: Parse1
- companion: Parse1

ParseElem
- type: String
- key: Integer
- sense: String
- ep-ref: String

SubPath
- type: String
- key: Integer
- sense: String
- ep-ref: String
- argument: ParseElem
- time-point: String

Path
- object: ParseElem
- fra: SubPath
- til: SubPath
- onPathList: ⟨ SubPath ⟩

Parse1
- arg: ParseElem

Parse2
- arg1: ParseElem
- arg2: ParseElem

We list template definitions used in our mapping algorithm. The templates are coded in Prolog:

```
%
% adverb
%
adv_tmpl(adv_1,
   adv, at_which_time_r1, movement,_,
   [],
   fork, question ).
%
% adjective
%
adj_tmpl(adj_1,
   a,Head, n, _Arg,
   [isa(Head,farge_a1)],
   new, adj_feature ).
%
% intransitive verb
%
itv_tmpl(itv_1,
   v, Head, n, Arg1,
   [isa(Head,mannerType_v1),has(Arg1,movable)],
   new, movement ).
itv_tmpl(itv_2,
   v, Head, n, Arg1,
   [isa(Head,underSpecSubj_v1),has(Arg1,movable)],
   new, movement ).
itv_tmpl(itv_3,
   v, er_property_v1, adj_feature, _,
   [],
   new, feature ).
itv_tmpl(itv_4,
   v, gå_v2, path,_Arg,
   [],
   new, pathdescription ).
itv_tmpl(itv_5,
   v, gå_v2, n,Arg,
   [isa(Arg,path_n1)],
   new, pathdescription ).
itv_tmpl(itv_6,
   v, Head, n,Arg1,
   [isa(Head,beg_end_v1),has(Arg1,movable)],
   new, movement ).
%
% transitive verb
%
tv_tmpl(tv_1,
   v,time_sit_at_v1, movement,_Arg1, n,Arg2,
   [isa(Arg2,tidspunkt_n1)],
   fork_arg1, time ).
tv_tmpl(tv_2,
   v,kjøre_v1, n,_Arg1, path,_Arg2,
   [],
   new, movement ).
tv_tmpl(tv_3,
   v,kjøre_v1, n,_Arg1, n,Arg2,
   [isa(Arg2,transportmiddel_n1)],
   new, movement ).
```

```
57  tv_tmpl(tv_4,
58     v,gå_v1, n,Arg1, path,_Arg2,
59     [has(Arg1,movable)],
60     new, movement ).
61  tv_tmpl(tv_5,
62     v,gå_v1, n,Arg1, direction,_Arg2,
63     [has(Arg1,movable)],
64     new, movement ).
65  tv_tmpl(tv_6,
66     v,gå_v1, n,Arg1, n,Arg2,
67    [has(Arg1,movable),isa(Arg2,path_n1)],
68    new, movement ).
69  %
70  % PP
71  %%
72  pp_tmpl(pp_1,
73     p,fra_p1, n,Arg1,
74     [has(Arg1,location)],
75     new, subpath).
76  pp_tmpl(pp_2,
77     p,til_p1, n,Arg1,
78     [has(Arg1,location)],
79     new, subpath).
80  pp_tmpl(pp_3,
81     p,via_p1, n,Arg1,
82     [has(Arg1,location)],
83     new, subpath).
84  pp_tmpl(pp_4,
85     p,langs_p1, n,Arg1,
86     [has(Arg1,longThin)],
87     new, subpath).
88  pp_tmpl(pp_5,
89     p,over_p2, n,Arg1,
90     [has(Arg1,longThin)],
91     new, subpath).
92  pp_tmpl(pp_6,
93     p,gjennom_p1, n,Arg1,
94     [has(Arg1,penetrable)],
95     new, subpath).
96  pp_tmpl(pp_7,
97     p,med_p1, n,Arg1,
98     [has(Arg1,movable)],
99     new, pp).
100 pp_tmpl(pp_8,
101    p,med_p2, n,Arg1,
102    [has(Arg1,propulsion)],
103    new, pp).
104 pp_tmpl(pp_9,
105    p,med_p3, n,Arg1,
106    [has(Arg1,movable)],
107    new, pp).
108 pp_tmpl(pp_10,
109    p,på_p1, n,Arg1,
110    [has(Arg1,location)],
111    new, location).
112 pp_tmpl(pp_11,
```

```
113     p,over_p2, n,Arg1,
114     [isa(Arg1,plass_n1)],
115     new, subpath).
116  pp_tmpl(pp_12,
117     p,langs_p1, path,Arg1,
118     [],
119     new, subpath).
120  pp_tmpl(pp_13,
121     p,Head, n,Arg1,
122     [isa(Head,direction_creation_r1),isa(Arg1,path_n1)],
123     new, subdirection).
124  pp_tmpl(pp_14,
125     p,Head, path,_Arg1,
126     [isa(Head,direction_creation_r1)],
127     new, subdirection).
128  pp_tmpl(pp_15,
129     p,forbi_p1, n,Arg1,
130     [has(Arg1,location)],
131     new, subpath).
132  %
133  % modification
134  %
135  mod_tmpl(mod_1,
136     movement,Arg1,subpath,Prep,
137     [isa(Prep,subpath_p1),isa(Arg1,movement_v1)],
138     call,_).
139  mod_tmpl(mod_2,
140     movement,Arg1,pp,Prep,
141     [isa(Prep,work_role_p1),isa(Arg1,movement_v1)],
142     call,_).
143  mod_tmpl(mod_5,
144     n,Arg1,subpath,_Prep,
145     [isa(Arg1,path_n1)],
146     new,path).
147  mod_tmpl(mod_6,
148     pathdescription,_Arg1,subpath,_Prep,
149     [],call,_).
150  mod_tmpl(mod_7,
151     n,Arg1,subpath,Prep,
152     [isa(Prep,fra_p1),has(Arg1,movable)],
153     fork,movement).
154  mod_tmpl(mod_8,
155     adv,Arg1,subpath,Prep,
156     [isa(Prep,til_p1),isa(Arg1,path_creation_r1)],
157     new,path).
158  mod_tmpl(mod_9,
159     movement,_Arg1,path,_Arg2,
160     [],
161     call,_).
162  mod_tmpl(mod_10,
163     n,Arg1,path,Arg2,
164     [isa(Arg1,path_n1)],
165     new,path).
166  mod_tmpl(mod_11,
167     movement,Arg1,subdirection,Prep,
168     [],
```

```
169     call,_).
170  mod_tmpl(mod_12,
171     adv,Arg1,subpath,Prep,
172     [isa(Arg1,direction_creation_r1)],
173     new,direction).
174  mod_tmpl(mod_13,
175     movement,_Arg1,direction,_Arg2,
176     [],
177     call,_).
178  mod_tmpl(mod_14,
179     movement,_Arg1,subpathdeep,_Arg2,
180     [],
181     call,_).
182  %
183  second_event_two_tmpl(sec_1,
184     movement, subject,Arg1, prep,med_p1,
185     [has(Arg1,propulsion)]).
186  second_event_two_tmpl(sec_2,
187     movement, subject,Arg1, prep,med_p2,
188     [has(Arg1,movable)]).
```

Listing D.1: Templates

We divide our "is-a" hierarchy into smaller parts so that the hierarchy parts can be displayed on a page. The hierarchy is divided into four parts and we have one part for "has" and "use" relations.

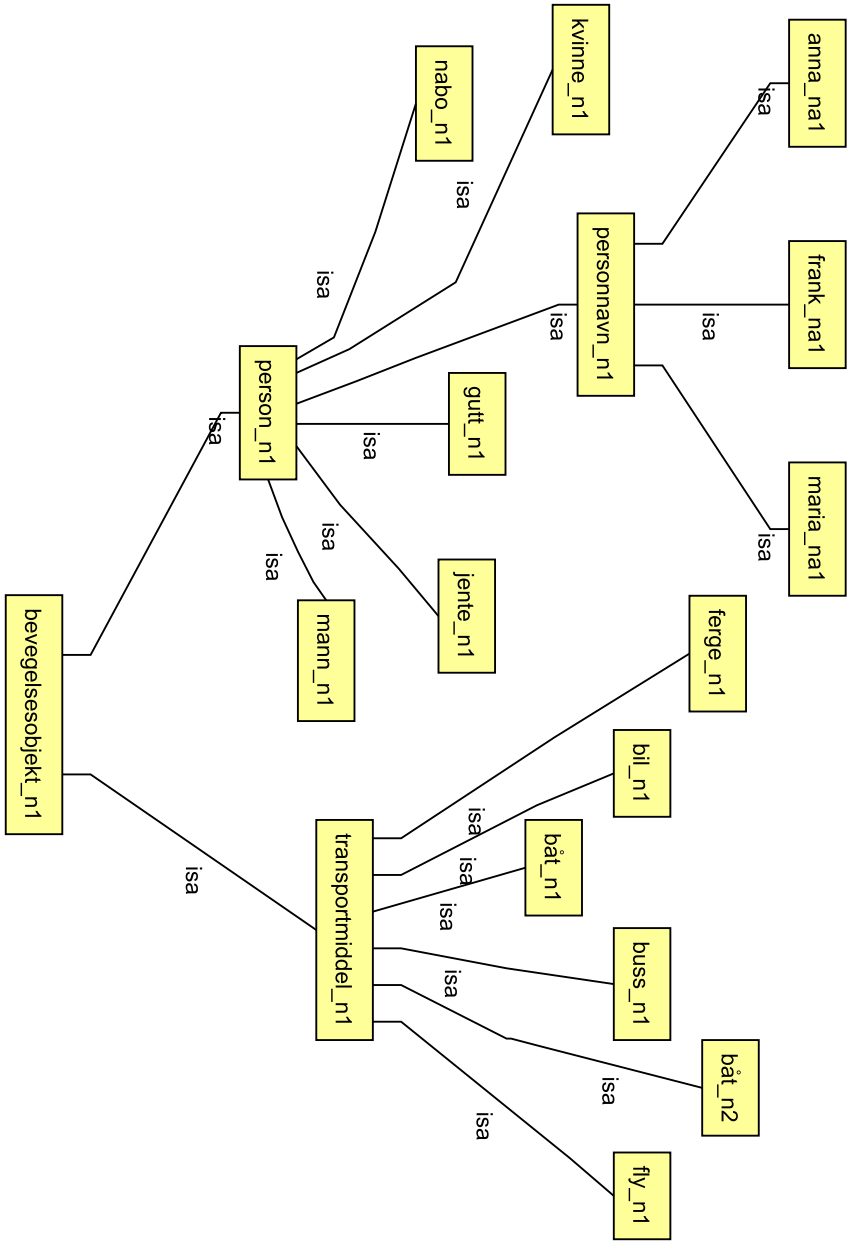


Figure D.1: Type Hierarchy, Part A

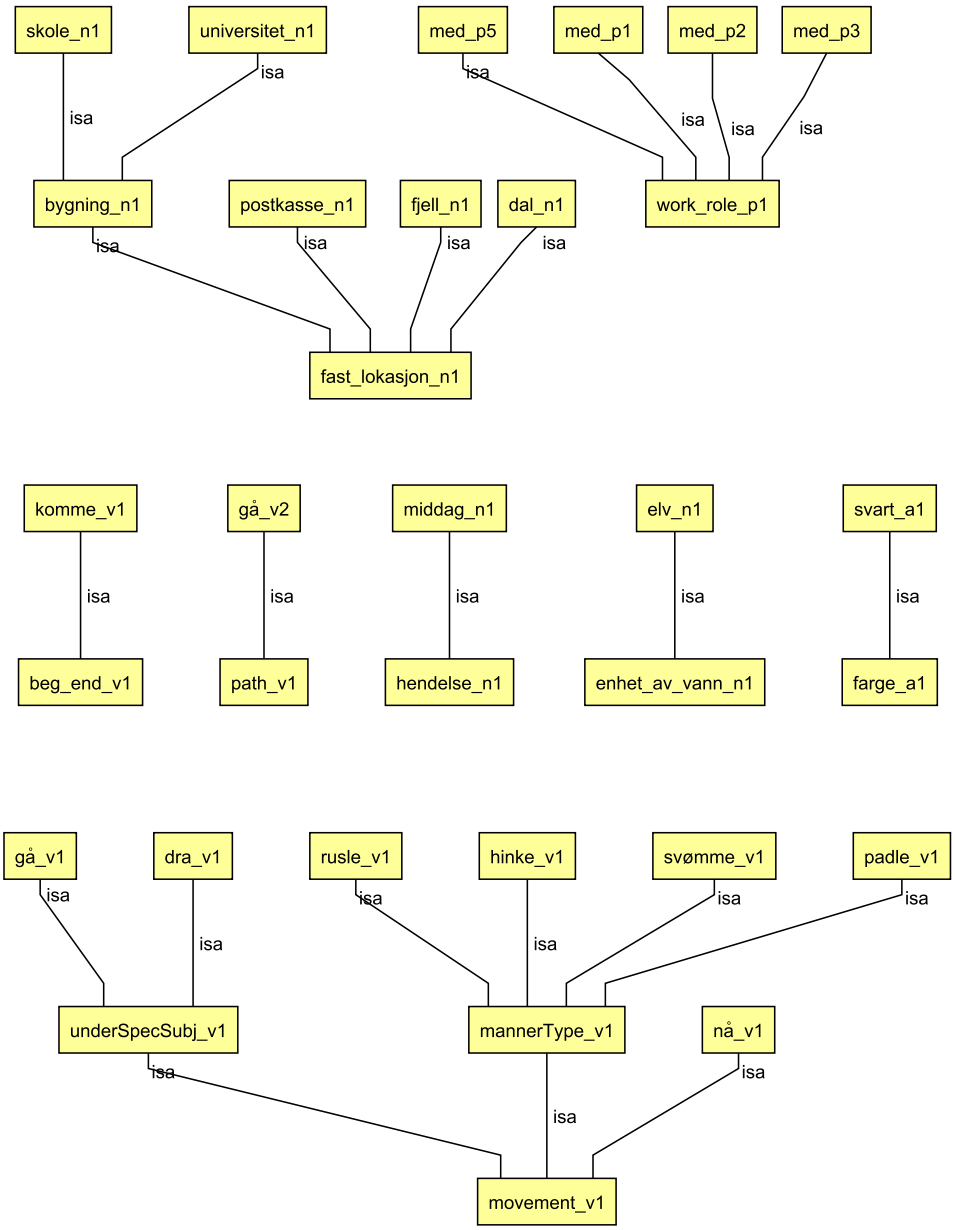


Figure D.2: Type Hierarchy, Part B

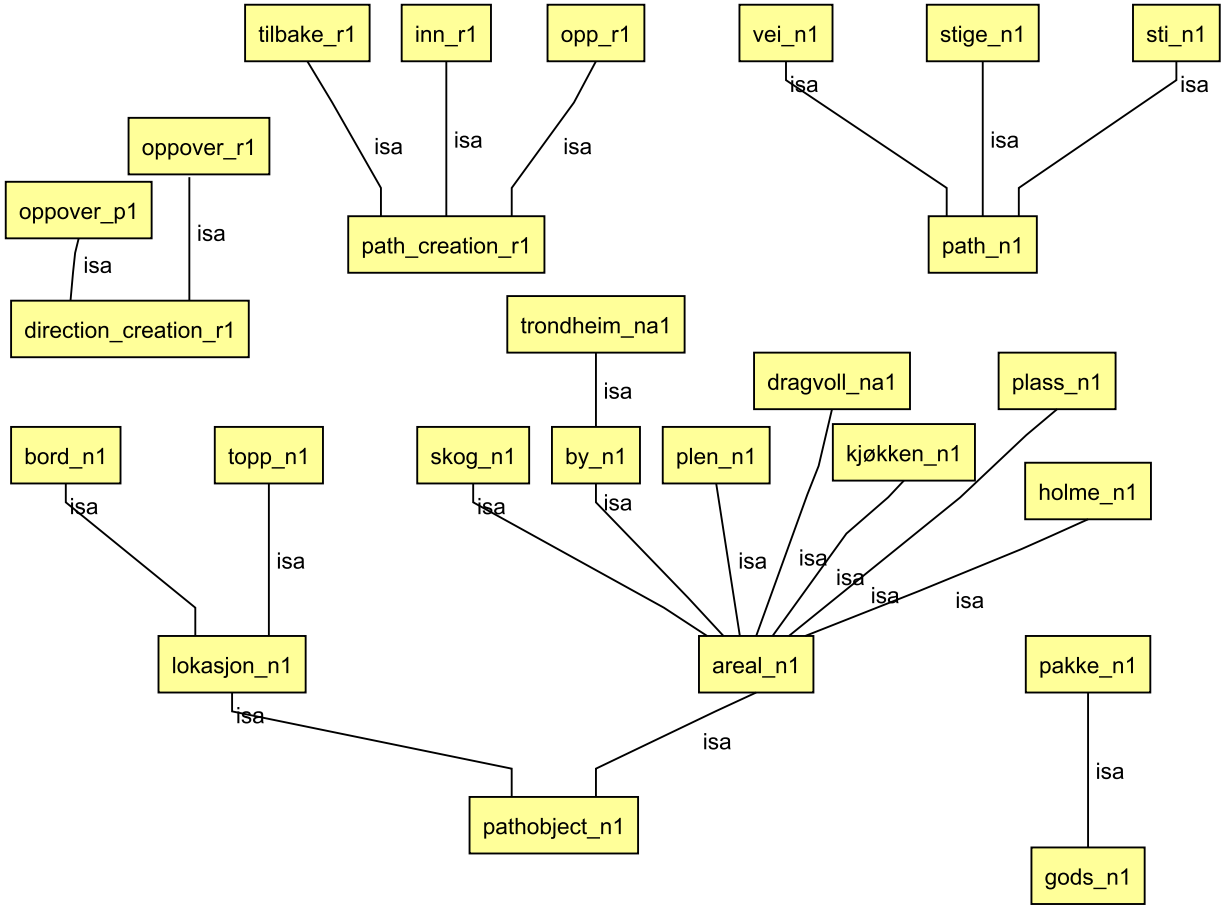


Figure D.3: Type Hierarchy, Part C

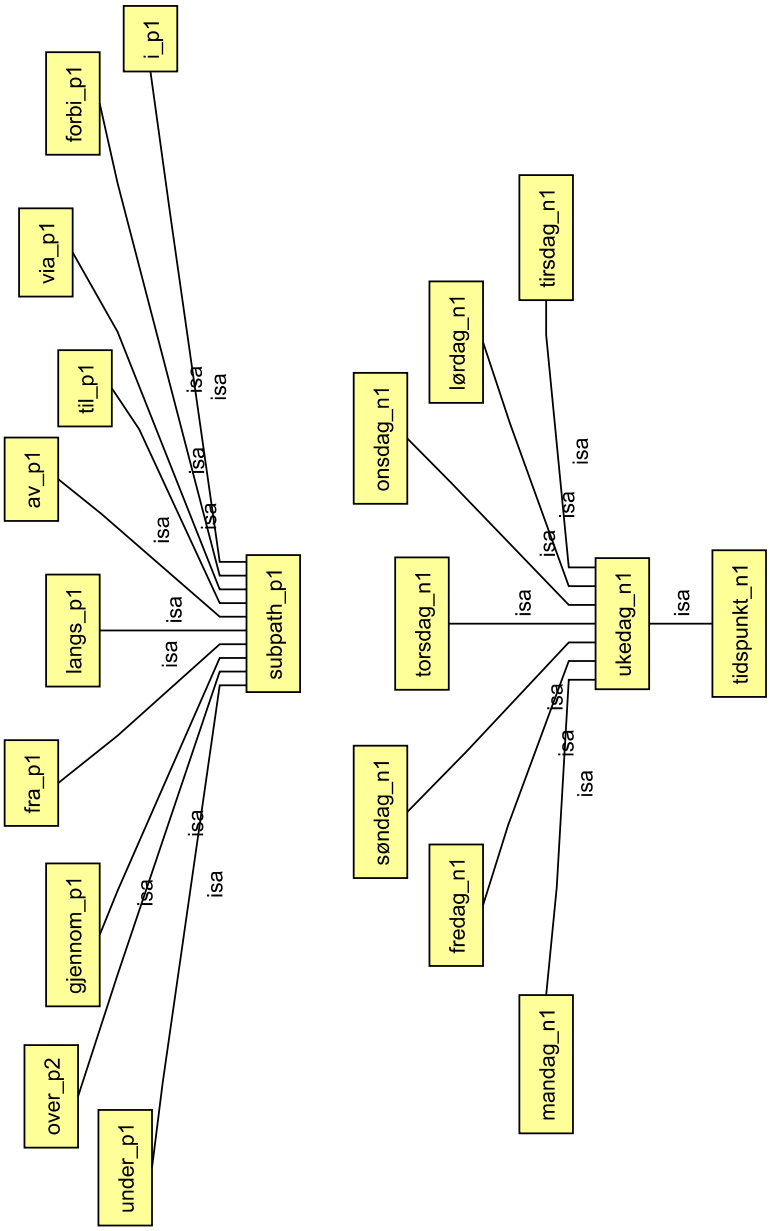


Figure D.4: Type Hierarchy, Part D

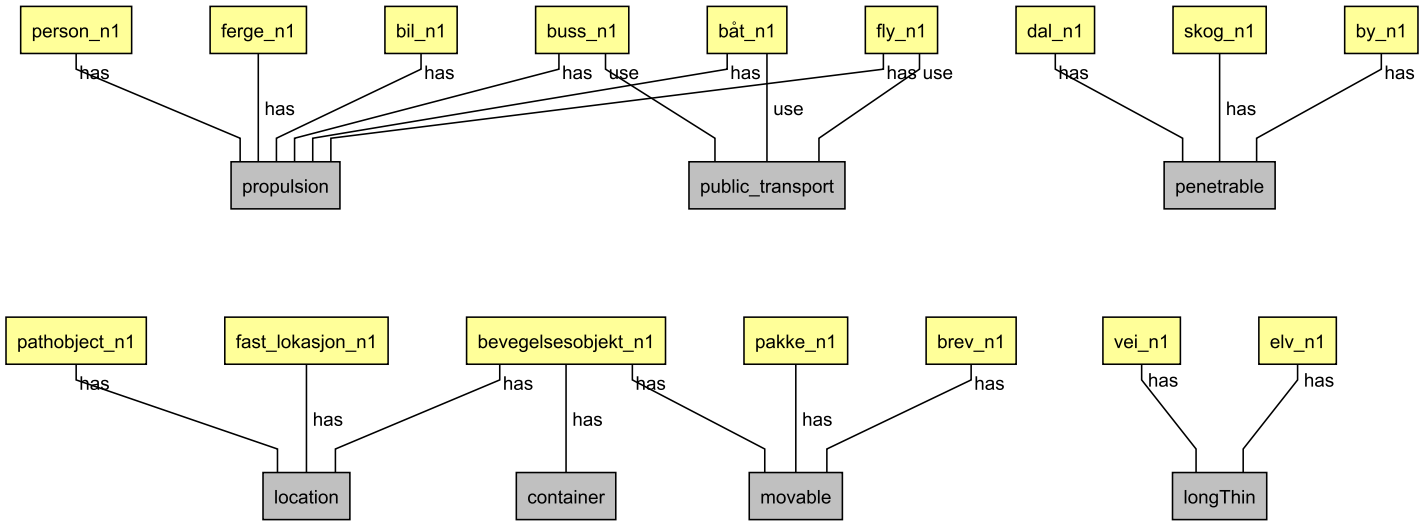


Figure D.5: Has and Use relations

Appendix E

Parser Details

E.1 Prolog Code for Templates

We list the Prolog code used for finding templates and for checking the validity of the "is-a", "has" and "use" relations. The relations are coded in the templates, and the templates are used in the mapping algorithm.

```
check(isa(A,B)):-
   synset(SynId1,_,_,_,A),
   synset(SynId2,_,_,_,B),
   isa_crawl_up(SynId1,SynId2).
check(has(A,B)):-
   synset(SynId1,_,_,_,A),
   has_crawl_up(SynId1,B).
check(use(A,B)):-
   synset(SynId1,_,_,_,A),
   use_crawl_up(SynId1,B).

check(neg(Check)):-
  \+check(Check).

check(and(A,B)):-
  check(A),
  check(B).

%%% crawl_up ontologi
isa_crawl_up(Denne,Kilde):-
  Denne=Kilde,
```

```
  !
;
  isa_rel(Denne,Kilde),
  !
;
  isa_rel(Denne,Foreldre),
  isa_crawl_up(Foreldre,Kilde).

%%% crawl_up ontologi looking for has info
has_crawl_up(Denne,Kilde):-
  has_rel(Denne,Kilde),
  !
;
  isa_rel(Denne,Foreldre),
  has_crawl_up(Foreldre,Kilde).

%%% crawl_up ontologi looking for use info
use_crawl_up(Denne,Kilde):-
  use_rel(Denne,Kilde),
  !
;
  isa_rel(Denne,Foreldre),
  use_crawl_up(Foreldre,Kilde).
find_template(Key,Head,Arg,Result):-
  ( itv_tmpl(Key,A,Head,B,Arg,C,D,E),
    Result=itv_tmpl(Key,A,Head,B,Arg,C,D,E)
  )
  ;
  ( pp_tmpl(Key,A,Head,B,Arg,C,D,E),
    Result=pp_tmpl(Key,A,Head,B,Arg,C,D,E)
  )
  ;
  ( mod_tmpl(Key,A,Head,B,Arg,C,D,E),
    Result=mod_tmpl(Key,A,Head,B,Arg,C,D,E)
  )
  ;
  ( adj_tmpl(Key,A,Head,B,Arg,C,D,E),
    Result=adj_tmpl(Key,A,Head,B,Arg,C,D,E)
  )
```

```
  ;
  ( adv_tmpl(Key,A,Head,B,Arg,C,D,E),
    Result=adv_tmpl(Key,A,Head,B,Arg,C,D,E)
  ).

find_template(Key,Head,Arg1,Arg2,Result):-
  tv_tmpl(Key,A,Head,B,Arg1,Arg2,C,D,E),
  Result=itv_tmpl(Key,A,Head,B,Arg1,Arg2,C,D,E).
```

E.2 MRS files

We use an example in Chapter 8 that refers to this detailed MRS.

E.2.1 Basic Example 1

```
1  Reading=1
2  Statement=Gutten går til byen
3  ltop=h1
4  Index=2
5  Ep
6    h3:_gutt_n_rel([ARG0(x4)], [])
7    h5:_def_q_rel([ARG0(x4)], [RSTR(h7), BODY(h6)])
8    h8:_gå_v_rel([ARG0(e2), ARG1(x4)], [])
9    h8:_til_p_rel([ARG0(u11), ARG1(x4), ARG2(x10), IARG(u9)], [])
10   h12:_by_n_rel([ARG0(x10)], [])
11   h13:_def_q_rel([ARG0(x10)], [RSTR(h15), BODY(h14)])
12 Var
13   x4, BOUNDED, +
14   x4, PNG.NG.GEN, M
15   x4, PNG.NG.NUM, SING
16   x4, PNG.PERS, THIRDPERS
17   x4, ROLE, MILEAGE-OBJ
18   x4, WH, -
19   e2, SF, PROP
20   e2, E.ASPECT, SEMSORT
21   e2, E.MOOD, INDICATIVE
22   e2, E.TENSE, PRES
23   e2, SORT, VERB-ACT-SPECIFICATION
24   u11, SORT, VERB-ACT-SPECIFICATION
25   x10, BOUNDED, +
26   x10, PNG.NG.GEN, M
27   x10, PNG.NG.NUM, SING
28   x10, PNG.PERS, THIRDPERS
29   x10, ROLE, ENDPNT
30   x10, WH, -
31 Hcons
32   h7, qeq, h3
33   h15, qeq, h12
```

Listing E.1: Basic Example 1

Bibliography

[1] James Allen. *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, 2. edition, 1995.

[2] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. Towards conversational human-computer interaction. *AI Magazine*, 22(4):27–37, 2001.

[3] J.F. Allen, M. Swift, and W. de Beaumont. Deep semantic analysis of text. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, pages 343–354. Association for Computational Linguistics, 2008.

[4] Ernst Althaus, Denys Duchier, Alexander Koller, Kurt Mehlhorn, Joachim Niehren, and Sven Thiel. An efficient graph algoritm for dominance constraints. *Journal of Algorithms*, 48:194–219, 2003.

[5] Tore Amble. BusTUC - a natural language bus route oracle. In *Proceedings of the 6th Applied Natural Language Processing Conference*, pages 1–6, Seattle, Washington, USA, April 29–May 4 2000. Association for Computational Linguistics.

[6] K. Barker, B. Porter, and P. Clark. A library of generic concepts for composing knowledge bases. In *Proceedings of the 1st international conference on Knowledge capture*, pages 14–21. ACM, 2001.

[7] R.J. Bayardo and R.C. Schrag. Using csp look-back techniques to solve real-world sat instances. In *Proceedings of the National Conference on Artificial Intelligence*, pages 203–208. JOHN WILEY & SONS LTD, 1997.

[8] D. Beermann and L. Hellan. A treatment of directionals in two implemented hpsg grammars. In Stefan Müller, editor, *Proceedings of the HPSG04 Conference*. CSLI Publications, 2004.

[9] Dorothee Beermann, Jon Atle Gulla, Lars Hellan, and Atle Prange. Trailfinder: a case study in extracting spatial information using deep language processing. In Ton van der Wouden, Michaela Poss, Hilke Reckman, and Crit Cremers, editors, *Computational Linguistics in the Netherlands Selected papers from the fifteenth CLIN meeting, 2004*, pages 121–131, Leiden, Netherlands, 2004.

[10] Emily M. Bender, Scott Drellishak, Antske Fokkens, Laurie Poulson, and Safiyyah Saleem. Grammar customization. *Research on Language & Computation*, 8(1):23–72, 2010.

[11] Emily M. Bender, Dan Flickinger, and Stephan Oepen. The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In John Carroll, Nelleke Oostdijk, and Richard Sutcliffe, editors, *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan, 2002.

[12] Patrick Blackburn and Johan Bos. *Representation and Inference for Natural Language*. CSLI Publications, 2005.

[13] M. Bodirsky, D. Duchier, J. Niehren, and S. Miele. A new algorithm for normal dominance constraints. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 59–67. Society for Industrial and Applied Mathematics, 2004.

[14] J. Bos. Introduction to the shared task on comparing semantic representations. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, pages 257–261. Association for Computational Linguistics, 2008.

[15] J. Bos. Wide-coverage semantic analysis with boxer. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*, volume 1, pages 277–286, 2008.

[16] Johan Bos. Let's not argue about semantics. In *Proceedings of the 6th Language Resources and Evaluation Conference (LREC 2008)*, 2008.

[17] A. Branco and F. Costa. Lxgram in the shared task "comparing semantic representations" of step 2008. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*, volume 1, pages 299–314, 2008.

[18] T. Brants. Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231. Association for Computational Linguistics, 2000.

[19] Jon S. Bratseth. BusTUC - a natural language bus traffic information system. Master Thesis, Norwegian University of Science and Technology, 1997.

[20] Heidi Brøseth. *A neo-constructional approach to computer-oriented talk*. Norwegian University of Science and Technology, Faculty of Arts, Department of Language and Communication Studies, 2007.

[21] Tore Bruland. Pre-Processing MRSes. In *In Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013), Potsdam, Germany*, 2013.

[22] P. Buitelaar. *CoreLex: systematic polysemy and underspecification*. PhD thesis, Brandeis University, 1998.

[23] C.B. Callaway. The textcap semantic interpreter. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*, volume 1, pages 327–342, 2008.

[24] Ulrich Callmeier. *Efficient Parsing with Large-Scale Unification Grammars*. Master thesis, Universit¨at des Saarlandes, 2001.

[25] P. Clark and P. Harrison. Boeing's nlp system and the challenges of semantic representation. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*, volume 1, pages 263–276, 2008.

[26] B. Comrie. *Aspect: An Introduction to the Study of Verbal Aspect and Related Problems*. Cambridge University Press, 1976.

[27] Ann Copestake. *Implementing Typed Feature Structure Grammars*. CSLI, 2002.

[28] Ann Copestake. *Robust Minimal Recursion Semantics*, 2004. `http://www.cl.cam.ac.uk/~aac10/papers/rmrsdraft.pdf`.

[29] Ann Copestake, Daniel Flickinger, Carl Pollard, and Ivan A. Sag. Minimal Recursion Semantics. An introduction. *Journal of Research on Language and Computation*, 3(4):281 – 332, 2005.

[30] Alan Cruse. *Meaning in Language*. Oxford University Press, 2. edition, 2004.

[31] Randall Davis, Howard Shrobe, and Peter Szolovits. What is knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

[32] R. Delmonte. Semantic and pragmatic computing with getaruns. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*, volume 1, pages 287–298, 2008.

[33] Peter J Denning. Computer science: The discipline. In A. Ralston and D. Hemmendinger, editors, *Encyclopaedia of Computer Science*. George Mason University, 2005.

[34] DIMACS. Satisfiability suggested format. Technical report, Center for Discrete Mathematics and Theoretical Computer Science, 1993.

[35] W. Drozdzynski, H.U. Krieger, J. Piskorski, U. Schäfer, and F. Xu. Shallow processing with unification and typed feature structures - foundations and applications. *Künstliche Intelligenz*, 18(1):17–23, 2004.

[36] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. the MIT Press, 1998.

[37] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building Watson: An overview of the DeepQA project. *AI magazine*, 31(3):59–79, 2010.

[38] David Ferrucci, Eric Nyberg, James Allan, Ken Barker, Eric Brown, Jennifer Chu-Carroll, Arthur Ciccolo, Pablo Duboue, James Fan, David Gondek, et al. Towards the open advancement of question answering systems. *IBM, Armonk, NY, IBM Res. Rep.*, 2009.

[39] Fikes and Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2 no.3,4 - January*, pages 189–208, 1971.

[40] D. Flickinger. On building a more effcient grammar by exploiting types. *Natural Language Engineering*, 6(01):15–28, 2000.

[41] Annika Flycht-Eriksson and Arne Jønsson. Dialogue and domain knowledge management in dialogue systems. In *Proceedings of 1st SiGdial Workshop on discourse and Dialogue*, Hong Kong, 2000.

[42] A. Frank, H.U. Krieger, F. Xu, H. Uszkoreit, B. Crysmann, B. Jörg, and U. Schäfer. Question answering from structured knowledge sources. *Journal of Applied Logic*, 5(1):20–48, 2007.

[43] Ruth Fuchss, Alexander Koller, Joachim Niehren, and Stepfan Thater. Minimal recursion semantics as dominance contraints: Translation, evaluation and analysis. In *Proceedings of the 42nd ACL*. Association for Computational Linguistics, 2006.

[44] L.T.F Gamut. *Intensional Logic and Logical Grammar*. Logic, Language and Meaning. The Univerity of Chicago Press, 1995.

[45] L.T.F Gamut. *Introduction to Logic*. Logic, Language and Meaning. The Univerity of Chicago Press, 1995.

[46] M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence*. Springer, 1987.

[47] R. Grishman, C. Macleod, and A. Meyers. Comlex syntax: Building a computational lexicon. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 268–272. Association for Computational Linguistics, 1994.

[48] T.R. Gruber et al. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[49] L. Hellan and P. Haugereid. Norsource: An exercise in matrix grammar-building design. In *Proc. the Workshop on Ideas and Strategies for Multilingual Grammar Development, ESSLLI 2003*, pages 41–48, 2003.

[50] Lars Hellan and Dorothee Beermann. Classification of prepositional senses for deep grammar applications. In Valia Kordoni and Aline Villavicencio, editors, *Proceedings of the 2nd ACL-Sigsem Workshop on The Linguistic Dimensions of Prepositions and their Use in Computational Linguistics Formalisms and Applications*, Colchester, United Kingdom, 2005.

[51] Lars Hellan and Dorothee Beermann. Semantics of spatial prepositions in the grammar norsource. Ms, NTNU, NO-7491 Trondheim, Norway, 2009.

[52] Lars Hellan and Tore Bruland. Constructing a multilingual database of verb valence. In *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013), Oslo, Norway*. NEALT Proceedings Series 16, 2013.

[53] Lars Hellan, Tore Bruland, Elias Aamot, and Mads H. Sandøy. A Grammar Sparrer for Norwegian. In *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013), Oslo, Norway*. NEALT Proceedings Series 16, 2013.

[54] Petter Henriksen and Vibecke Haslerud. *Norsk-Engelsk Stor Ordbok*. Kunnskapsforlaget, 2006.

[55] Magne Hallstein Johnsen, Tore Amble, and Erik Harborg. A Norwegian Spoken Dialogue System for Bus Travel Information. *Teletronikk*, 99(2), 2003.

[56] Daniel Jurafsky and James H. Martin. *SPEECH and LANGUAGE PROCESSING: An introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, 2. edition, 2008.

[57] K. Kipper, A. Korhonen, N. Ryant, and M. Palmer. A large-scale classification of english verbs. *Language Resources and Evaluation*, 42(1):21–40, 2008.

[58] A. Koller and S. Thater. The evolution of dominance constraint solvers. In *Proceedings of the Workshop on Software*, pages 65–76. Association for Computational Linguistics, 2005.

[59] A. Koller and S. Thater. An improved redundancy elimination algorithm for underspecified representations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 409–416. Association for Computational Linguistics, 2006.

[60] Alexander Koller and Stefan Thater. *Utool: The Swiss Army Knife of Underspesification*. Saarland University, Saarbrücken, 2006. `http://www.coli.uni-saarland.de/projects/chorus/utool/`.

[61] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[62] Michiel Lambalgen and Fritz Hamm. *The Proper Treatment of Events*. Blckwell Publishing, 2005.

[63] S. Lehmann, S. Oepen, S. Regnier-Prost, K. Netter, V. Lux, J. Klein, K. Falkedal, F. Fouvry, D. Estival, E. Dauphin, et al. Tsnlp: Test suites for natural language processing. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 711–716. Association for Computational Linguistics, 1996.

[64] Beth Levin. *English Verb Classes and Alternations*. The University of Chicago Press, 1993.

[65] Beth Levin and Malka Rappaport Hovav. *Argument Realization*. Cambridge University Press, 2005.

[66] V. Lifschitz. Formal theories of action (preliminary report). In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 966–972, 1987.

[67] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the sixth conference on Natural Language Learning (CoNLL-2002)*, pages 49–55, 2002.

[68] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[69] Michael F. McTear. Spoken dialogue technology: enabling the conversational user interface. *ACM Computer Surveys*, 34(1):90–169, 2002.

[70] Michael F. McTear. *Spoken Dialogue Technology: Towards the Conversational User Interface*. Springer-Verlag, London, 2004.

[71] Ruslan Mitkov. *Anaphora Resolution*. Longman, 2002.

[72] Marc Moens and Mark Steedman. Temporal ontology and temporal reference. *Computational Linguistics*, 14(2), 1988.

[73] Marc Moens and Mark Steedman. Temporal ontology and temporal reference. In Inderjeet Mani, James Pustejovsky, and Robert Gaizauskas, editors, *The Language of Time: a reader*, chapter 27, pages 545–558. Oxford University Press, 2005.

[74] Erik T. Mueller. Story understanding through multi-representation model construction. In *Proceedings of the HLT-NAACL 2003 workshop*

on Text meaning, pages 46–53, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[75] Erik T. Mueller. Discrete event calculus reasoner documentation. Software documentation, IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA, 2005.

[76] Erik T. Mueller. *Commonsense Reasoning*. Morgan Kaufman Publishing, 2006.

[77] Joachim Niehren and Stefan Thater. Bridging the gap between underspesification formalisms: Minimal recursion semantics as dominance constraints. In *41st Meeting of the Association of Computational Lingustics*, pages 367–374, 2003.

[78] S. Nirenburg, S. Beale, and M. McShane. Baseline evaluation of wsd and semantic dependency in ontosem. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*, volume 1, pages 315–326, 2008.

[79] S. Nirenburg and V. Raskin. *Ontological semantics*. MIT Press Cambridge, MA, 2004.

[80] S. Oepen and D.P. Flickinger. Towards systematic grammar profiling. test suite technology ten years after. *Journal of Computer Speech and Language*, 12(4):411–436, 1998.

[81] S. Oepen, K. Toutanova, S. Shieber, C. Manning, D. Flickinger, and T. Brants. The lingo redwoods treebank: Motivation and preliminary applications. In *Proceedings of the 19th international conference on Computational linguistics-Volume 2*, pages 1–5. Association for Computational Linguistics, 2002.

[82] Barbara H. Partee. Compositionality. In Barbara H. Partee, editor, *Compositionality in Formal Semantics*, pages 153–181. Blackwell Publishing, 2004.

[83] Fernando Pereira. Extraposition grammars. *Computational Linguistics*, 7(4):243–256, 1981.

[84] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, Chicago, Illinois, 1994.

[85] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2. edition, 2003.

[86] Ivan A Sag, Thomas Wasow, and Emily M Bender. *Syntactic Theory: a formal introduction*. CSLI Publications, 2. edition, 2003.

[87] U. Schäfer. Ontonerdie-mapping and linking ontologies to named entity recognition and information extraction resources. In *Proceedings of the 5th international conference on language resources and evaluation (LREC), Genova*, 2006.

[88] U. Schäfer, H. Uszkoreit, C. Federmann, T. Marek, and Y. Zhang. Extracting and querying relations in scientific papers on language technology. *Proc. of LREC-2008, Marrakesh, Morocco*, 2008.

[89] Ulrich Schäfer. Middleware for creating and combining multi-dimensional nlp markup. In *Proceedings of the EACL-2006 Workshop on Multi-dimensional Markup in Natural Language Processing*, Trento, Italy, 4 2006.

[90] Ulrich Schäfer. *Integrating Deep and Shallow Natural Language Processing Components – Representations and Hybrid Architectures*. PhD thesis, Faculty of Mathematics and Computer Science, Saarland University, Saarbrücken, Germany, 2007. Doctoral Dissertation; also available as Vol. 22 of the Saarbrücken Dissertations in Computational Linguistics and Language Technology series (http://www.dfki.de/lt/diss), ISBN 978-3-933218-21-6.

[91] David Schlangen and Alex Lascarides. The interpretation of non-sentential utterances in dialogue. In *Proceedings of the 4th SIGdial workshop on Discourse and Dialogue*, 2003.

[92] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability: Second DIMACS implementation challenge*, 26:521–532, 1993.

[93] Murray P. Shanahan. *Solving the Frame Problem*. The MIT Press, 1997.

[94] John F. Sowa. *Knowledge Representation: logical, philosophical, and computational foundations*. Brooks/Cole, 2000.

[95] Mark Steedman. *The Syntactic Process*. The MIT Press, 2000.

[96] M Thielscher. From situating calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111:277–299, 1996.

[97] David R. Traum and Staffan Larsson. The information state approach to dialogue management. In Jan van Kuppevelt and Ronnie W. Smith, editors, *Current and New Directions in Discourse and Dialogue*, chapter 15, pages 325–354. Kluwer Academic Publishers, 2003.

[98] Erik Velldal. *Empirical Realization Ranking*. PhD thesis, The University of Oslo, 2008.

[99] Zeno Vendler. *Linguistics in Philosophy*. Cornell Univerity Press, 1967.

[100] Zeno Vendler. Verbs and times. In Inderjeet Mani, James Pustejovsky, and Robert Gaizauskas, editors, *The Language of Time: a reader*, chapter 1, pages 21–32. Oxford University Press, 2005.

[101] P. Vossen. *EuroWordNet: a multilingual database with lexical semantic networks*. Kluwer Academic, 1998.

[102] Christoph Walther. *A Many-Sorted Calculus Based on Resolution and Paramodulation*. Pitman, London, 1987.

[103] Peter Wegner. Research paradigms in computer science. In *Proceedings of the 2nd international conference on Software engineering*, pages 322–330. IEEE Computer Society Press, 1976.

[104] Øystein Fledsberg and Kim Bjerkevoll. Buster - Robust Dialogue Management. MSc thesis, Norwegian University of Science and Technology (NTNU), November 1999.