KIT – REPORT 117

A Proof Theory for Preferential Default Description Logics

Sven Suska
Humboldt–Universität zu Berlin
svs@cs.tu-berlin.de

Berlin, July 1994

Abstract

An extension of Description Logics by weighted default rules is presented, using the semantics of preferential models. Such Preferential Default Description Logics adhere to the principle of exception minimization and are therefore suited for certain tasks in Natural Language Processing.

A sound and complete proof theory is given, based on establishing a correspondence between models and sets of default applications ('default spaces'). A Prolog algorithm computing maximal default spaces is described and formally verified.

As a consequence, the default extension is decidable if the underlying Description Logic is. However, its complexity is exponential, some ideas are proposed to cope with this problem. A tractable subclass is characterized, for which a polynomial algorithm is sketched.

Preface

This report is a slightly adapted version of my graduation thesis. At this point I would like to thank all those who contributed in some respect.

I am very grateful to my supervisor Joachim Quantz for supporting me in an open and gentle style, and to Hans-Dieter Burkhard for a lot of encouragement and many ideas.

I thank Rüdiger Klein, Andreas Hoppe and Ingo Jentsch for various helpful comments.

Thanks also go to the KIT-VM11 team, who were always ready to answer the many technical questions, especially to Guido Dunker for helping me with the FLEX system.

Contents

Chapter 1

Introduction

All propositions are approximations to an elusive truth
H.G.Wells

Description Logics have become quite popular for Knowledge Representation in a wide range of Artificial Intelligence areas, mainly due to their exact formal syntax, semantics and proof theory. Also, efficient inference strategies have by now been developed, resulting in many practicable representation and reasoning systems.

However, a decisive drawback of logic is the rigid monotonicity of its consequence relation, not adequate for most real-world situations and forcing modelings to become complicated and unintuitive. Only a few systems acknowledge this problem by clearly defining a mechanism for default reasoning. Many others just provide some informal loopholes for defeasible descriptions, without formal semantics and with only rudimentary algorithmic coverage.

Preferential Default Description Logics (PDDL) are a recent development to incorporate default reasoning into Description Logics on the sound basis of a model-theoretic semantics. Various kinds of PDDL have been devised as a result of different ways to deal with conflicting defaults. (cf. [Quantz, Ryan 93])

Preferential Default Description Logics based on weighted defaults have been found to be especially suited for a task in Natural Language Processing, the task of disambiguation. Ambiguity is an intrinsic feature of natural language; the meaning of small entities (words, phrases) on their own is rather vague, only when contextual information is taken into account, a clear meaning can be established.

In more formal terms all the information from every phrase constituent and from the context can be viewed as (soft) constraints on the eventual interpretation. Many of those ‘constraints’ are not universally true, but mere default rules, expressing what is to be expected normally. Since PDDL are based on the principle of exception minimization, they are ideal for disambiguation by preferring those interpretations which are closest to normality, or, in other words, are least exceptional.

The weighted PDDL have been thoroughly studied from a theoretical point of view in [Quantz, Ryan 93]. They turn out to have very strong logical properties (such as Rational Monotonicity).

The aim of this paper is to take a second step and describe a way to perform syntactic deduction in the weighted PDDL. At this early stage, only marginal attention can be given to efficiency. The emphasis was rather placed on finding a proof theory and an algorithm that are sound and complete. Later, practical —possibly incomplete— algorithms may be based upon and compared with it. Also, some insight into the complexity of the task can be gained.

The paper is organized in four main chapters. In Chapter 2 I will briefly review some relevant aspects of the background of PDDL, before introducing the preferential default extension examined here. Chapter 3 contains the proof theory, which is based on default spaces. An algorithm to compute the maximal default spaces is given in Chapter 4 and its soundness and correctness is proved. A few complexity issues of this approach are touched in Chapter 5. Afterwards, a polynomial algorithm for a subclass of the problem is outlined, and several practically motivated ideas are discussed.

Chapter 2

Foundations

PDDL are a result of merging default reasoning with logic-based knowledge representation – both areas will be briefly reviewed in the first two sections of this chapter, the third section introduces the weighted preferential entailment relation itself, including examples and a list of formal properties. Much of the presented material can also be found in [Quantz, Ryan 93], where more references are given.

2.1 Description Logics

2.1.1 Origins of Description Logics

In his Knowledge Representation Hypothesis, Brian Smith claims that any AI system must contain a component that (a) *we (humans) recognize to represent the knowledge of that system* and (b) *plays a causal and essential role in engendering the behavior of the system* [Smith 85].

In the light of the success of Neural Network models one may doubt its truth, but the field of Knowledge Representation (KR) has long been of considerable importance in various areas of Artificial Intelligence, such as expert systems and natural language processing.

KR started in the late 1960's, when Quillian proposed his *Semantic Networks* [Quillian 68]. Semantic memory here was still assumed to be general memory, suited for all kinds of activity including natural language understanding as well as visual perception. It was regarded as a model of the human cognitive structures, and thus was open for verification in psychological experiments.

This paradigm views knowledge as organized by labeled *links* between *nodes* (concepts, objects). There can be all kinds of links to represent relations among the nodes, but one form is always used, the IsA-link, which expresses that one concept is a subconcept of another. While Semantic Nets have the advantage of high flexibility, they lack a clearly defined interpretation and resist formal treatment.

A second paradigm was introduced by Minsky's Frames [Minsky 75]. Again, this approach was motivated by psychological and linguistic models. Frames, as well as the similar idea of Scripts, put emphasis on the grouping of knowledge into nested chunks, or *slots*. Inheritance between subframes corresponds to the IsA-link of Semantic Nets.

Frames are a more formal notion than Semantic Nets, but still, their semantics was not defined in exact terms. For both formalisms, there was no entailment relation, that is, whether one Semantic Net/Frame contains more information than another was an unanswerable question. All this gave rise to criticism from the side of *logic* in the late 70's.

In 1980 Hayes [Hayes 80] managed to map Frame definitions into first order logic (FOL). He thus showed that Frames are not more expressive than FOL. But this is only one side of the coin — equally important for AI is the question of structural adequacy, i.e. perhaps Frames allow for a representation that is more succinct and easier to manipulate. So, Hayes' result can in fact be seen as a justification for these representation formalisms, in the same way as programming languages are important *because* they can be compiled into machine code, (which also implies that they are not more expressive).

Brachman went further along the logic line of KR (e.g. [Brachman 79]), paying attention to the structural needs for knowledge representations. With KL-ONE [Brachman, Schmolze 85], he created a model for exact knowledge representation languages. A variety of similar systems (KRYPTON, KR-L, KRS) were developed in the early 1980's. The terms 'KL-ONE-alike systems', 'term subsumption systems', 'concept logics', 'terminological logics', and 'description logics' are all used for them.

As the capabilities and limitations of Description Logics (DL) emerged more clearly, a second generation of DL-systems developed towards the end of the 80's. CLASSIC, BACK and LOOM are the most well-known examples, being still in use today. To ensure reasonable answer times, LOOM only provides incomplete reasoning algorithms for its quite expressive language, whereas the language of CLASSIC is very restricted, but complete inference is performed. BACK takes a stance somewhere in the middle of this trade-off. The newly developed KRIS system tries to overcome these deficiencies by offering complete treatment of an expressive language using powerful tableaux algorithms, but long computation times are often inevitable.

Recently, DLs have been extended to incorporate special categories of reasoning like epistemic operators, part-whole-relations, nonmonotonic inference or temporal relations (see for example [Fischer 92]). Also, some systems provide a facility to manage alternative worlds simultaneously in the knowledge base. This is convenient as a basis for implementing nonmonotonic reasoning, such a system was used for implementing the algorithm proposed in this thesis.

2.1.2 A simple DL

In the following, I will define a small Description Logic to illustrate what has been said about DL and also to use it as a basis for the definition of the nonmonotonic extension later.

All DLs distinguish between concrete *objects* and abstract *terms*. Terms split up into *concepts* (unary predicates) and *roles* (binary predicates). Formulae in DL are traditionally divided into *ABox* and *TBox* formulae. The ABox is taken to contain all the assertions of concepts to objects, they are also called *descriptions*. The TBox consists of the *terminology*, that is all *definitions* of one term name as abbreviation for another term, and *rules* of subsumption of terms.

In the following definition of the DL syntax, letters c, c_1, c_2 are used for arbitrary concepts, r for roles and t, t_1, t_2 for terms. c_n, r_n and $o \in \mathcal{O}$ denote concept-names, role-names and object-names respectively. (Let $\mathcal{O}$ be the set of all object-names, this will also be needed later.) γ will denote a DL-formula.

$$
\begin{array}{lcll}
t & \rightarrow & c, r & \\
c & \rightarrow & \top & \text{(top concept, `anything')} \\
 & & \bot & \text{(bottom concept, `nothing')} \\
 & & c_n & \\
 & & \neg c & \text{(negation)} \\
 & & c_1 \sqcap c_2 & \text{(conjunction)} \\
 & & c_1 \sqcup c_2 & \text{(disjunction)} \\
 & & r : o & \text{(fills construct)} \\
 & & \forall r : c & \text{(value restriction)} \\
 & & \exists r : c & \text{(existential restriction)} \\
r & \rightarrow & r_n & \\
 & & r_1.r_2 & \text{(role composition)} \\
\gamma & \rightarrow & t_1 \sqsubseteq t_2 & \text{(subsumption)} \\
 & & o :: c & \text{(description)}
\end{array}
$$

For denoting the terminology, the symbol $\rightarrow$ is used as an alias for subsumption and $\doteq$ as an abbreviation to express definitions:

$$
\begin{array}{lcll}
c_n \doteq c & \overset{\text{def}}{=} & c_n \sqsubseteq c \text{ and } c \sqsubseteq c_n & \text{(definition)} \\
c_1 \rightarrow c_2 & \overset{\text{def}}{=} & c_1 \sqsubseteq c_2 & \text{(rule)}
\end{array}
$$

The following model theoretic semantics is taken from [Quantz, Ryan 93]. A model M of a set of DL-formulae Γ is a pair $\langle D, [\![\cdot]\!]^{\mathcal{I}} \rangle$. D is a finite set, called the domain, and $[\![\cdot]\!]^{\mathcal{I}}$ is an interpretation function mapping concepts into subsets of D, roles into subsets of $D \times D$, and object names injectively (Unique Name Assumption) into D, in accordance with the following equations (I will use $r(d)$ to denote $\{e : \langle d, e\rangle \in r\}$):

$$\begin{array}{rcl}
[\![\top]\!]^{\mathcal{I}} & = & D \\
[\![\bot]\!]^{\mathcal{I}} & = & \emptyset \\
[\![\neg \mathrm{c}]\!]^{\mathcal{I}} & = & D \setminus [\![\mathrm{c}]\!]^{\mathcal{I}} \\
[\![\mathrm{c}_1 \sqcap c_2]\!]^{\mathcal{I}} & = & [\![\mathrm{c}_1]\!]^{\mathcal{I}} \cap [\![\mathrm{c}_2]\!]^{\mathcal{I}} \\
[\![\mathrm{c}_1 \sqcup \mathrm{c}_2]\!]^{\mathcal{I}} & = & [\![\mathrm{c}_1]\!]^{\mathcal{I}} \cup [\![\mathrm{c}_2]\!]^{\mathcal{I}} \\
[\![\forall \mathrm{r} : \mathrm{c}]\!]^{\mathcal{I}} & = & \{d \in D : [\![\mathrm{r}]\!]^{\mathcal{I}}(d) \subseteq [\![\mathrm{c}]\!]^{\mathcal{I}}\} \\
[\![\exists \mathrm{r} : \mathrm{c}]\!]^{\mathcal{I}} & = & \{d \in D : [\![\mathrm{r}]\!]^{\mathcal{I}}(d) \cap [\![\mathrm{c}]\!]^{\mathcal{I}} \neq \emptyset\} \\
[\![\mathrm{r} : \mathrm{o}]\!]^{\mathcal{I}} & = & \{d \in D : [\![\mathrm{o}]\!]^{\mathcal{I}} \in [\![\mathrm{r}]\!]^{\mathcal{I}}(d)\} \\
[\![\mathrm{r}_1 . \mathrm{r}_2]\!]^{\mathcal{I}} & = & \{\langle d, e\rangle \in D \times D : \text{there exists an} f \in D \\
 & & \quad \text{with } \langle d, f\rangle \in [\![\mathrm{r}_1]\!]^{\mathcal{I}} \text{ and } \langle f, e\rangle \in [\![\mathrm{r}_2]\!]^{\mathcal{I}}\}
\end{array}$$

Satisfaction of formulae is then defined as follows:

$$\begin{array}{rcl}
M \models t_1 \sqsubseteq t_2 & \text{iff} & [\![t_1]\!]^{\mathcal{I}} \subseteq [\![t_2]\!]^{\mathcal{I}} \\
M \models \mathrm{o} :: \mathrm{c} & \text{iff} & [\![\mathrm{o}]\!]^{\mathcal{I}} \in [\![\mathrm{c}]\!]^{\mathcal{I}}
\end{array}$$

We call M a model *of* a formula γ iff $M \models \gamma$. M is a model of a set of formulae Γ iff it is a model of every formula in Γ. Then the entailment relation can be straightforwardly defined by stating that a formula γ is *entailed* by a set of formulae Γ (written $\Gamma \models \gamma$) iff every model of Γ is also a model of γ.

This logic is obviously subsumed by first order logic. A translation is given in [Quantz, Ryan 93].

A Small Example Modeling

An application of DL, i.e. a set of DL-formulae, is called a *domain model.* The following could be seen as a very simple example.

Suppose, we are talking about widows and canaries. Canaries are vertebrates, so are humans, but humans are never canaries. Widows are humans. They all can be old or lively. Canaries only like widows. Then we get the following terminology or TBox:

$$\begin{array}{rcl}
\Gamma_T = \{\mathrm{vertebrate} & \rightarrow & \top, \\
\mathrm{canary} & \rightarrow & \mathrm{vertebrate} \sqcap \forall\ \mathrm{likes} : \mathrm{widow}, \\
\mathrm{human} & \rightarrow & \mathrm{vertebrate} \sqcap \neg \mathrm{canary}, \\
\mathrm{widow} & \rightarrow & \mathrm{human}, \\
\mathrm{old} & \rightarrow & \top, \\
\mathrm{lively} & \rightarrow & \top\}
\end{array}$$

Now suppose that we only know two objects, called Emma and Tweety. Emma is a lively widow who likes Tweety. Tweety is a lively canary.

$$\Gamma_A = \{\mathrm{Emma} \quad :: \quad \mathrm{widow} \sqcap \mathrm{lively} \sqcap \mathrm{likes} : \mathrm{Tweety},$$

$$\begin{aligned} \text{Tweety} &:: \text{canary}, \\ \text{Tweety} &:: \text{lively}\} \end{aligned}$$

Then a DL system can draw the following inferences:

$$\begin{aligned} \Gamma_T \cup \Gamma_A &\models \text{Emma} :: \text{human}, \\ \Gamma_T \cup \Gamma_A &\models \text{widow} \sqcap \text{canary} \sqsubseteq \bot, \\ \Gamma_T \cup \Gamma_A &\models \text{Tweety} :: \neg\text{likes} : \text{Tweety} \end{aligned}$$

There are basically two main reasoning components in a DL system: the *classifier* computes subsumption between terms, i.e. whether $\Gamma \models t_1 \sqsubseteq t_2$; the *recognizer* computes whether an object ois an instance of a concept, i.e. whether $\Gamma \models$ o :: c. A typical DL system provides a wider range of queries; although they are all reducible to the above two, their specialized nature sometimes reduces complexity significantly. The terminological reasoning, i.e. evaluating the concept hierarchy, is normally the most important and complicated part, it is also needed for recognizing a single object. That is why for theoretical considerations it is usually sufficient to examine subsumption.

2.2 Nonmonotonic Reasoning

2.2.1 Motivation

Nonmonotonic Reasoning (NMR) is only defined in negative terms, by the absence of the property of monotonicity. This is quite unusual, and one may ask why this property has attracted so much attention. Monotonic reasoning, i.e. that new premises do not invalidate old conclusions, has been adopted by traditional logic for the sake of its simplicity and its nice theoretical properties. Although a sound basis for mathematics, it is quite inadequate for the majority of real world tasks.

All the early KR formalisms had nonmonotonic inferences integrated in a natural way, e.g. for Frames there is the notion of defeasible inheritance; but after logic was incorporated into knowledge representation formalisms, much effort is used to eliminate monotonicity.

In more detail, we can distinguish at least three practical reasons to prefer nonmonotonic reasoning: open worlds, succinct representation and temporal change.

Natural human speech takes place in an open world, it is simply impossible to state every fact about a situation. So, if we are talking about fish, we usually assume that it swims, but are ready to withdraw that conclusion if we learn that the fish is dead, in which case it can only float in the water. This can be relevant in machine translation, for example in German both verbs have the common equivalent of "schwimmen" — additional information is needed for deciding which which English verb to use.

A machine translation lexicon could be viewed as an open world, since new words can be added, but is usually rather stable. Still, we would like to have a general rule that states that the past tense is normally formed by suffixing '-ed'. This is not universally true of course, but we can explicitly specify when a verb is an exception. So, the rule is applied *unless* the verb is known to be irregular. In the case of a lexicon it would in principle be possible to mention all exceptions in a strict rule. But this would lead to a quite clumsy representation.

If the domain contains only a handful of objects, this is not a problem. It is the *complexity* of the task, that generates the need for adequate representations.

Clearly, for temporal reasoning nonmonotonicity is vital. We cannot repeat the whole specification for every new situation; somehow it has to be coded implicitly that everything stays where it is *unless* its change is explicitly known.

There is also a different type of motivation for nonmonotonicity. If AI claims to model human cognitive processes, it must be sensitive to the question whether human thinking can be monotonic. This is answered negatively by *prototype theory* from Cognitive Science, stating that our cognitive categories are formed not by strict definition but rather by associating features of varying relevance.

Another important aspect is the significance of the *context*, or *background*. Barwise's notion of *conditional constraints* [Barwise 89] is based on the experience that nothing ever applies to all situations — even if we are not aware of them, there are always some implicit conditions in the background. Strict theories like mathematics try to cut off those context references, but AI probably cannot afford to do the same.

2.2.2 Different Approaches to NMR

Rules expressing "if a, then normally b", or "if a then b, unless not b is known" are usually called *defaults*. Here we will use a curled arrow to denote them, e.g.

$$\text{bird} \leadsto \text{flies}.$$

Nonmonotonic inference systems mostly operate on a set Δ of special formulae (e.g. defaults) and strict knowledge Γ. Δ can be empty, then the approach is called *homogeneous*; if there are two kinds of formulae, it is called *heterogeneous*.

In classical logic, the operator $\mathrm{Th}(\Gamma)$ is used to denote the set of all formulae derivable from a set of formulae Γ. We will use $\mathrm{NmTh}_\Delta(\Gamma)$ for what we can derive nonmonotonically from Γ and Δ, when we compare the different NMR systems.

Usually, adding default information to a set of formulae allows to conclude more information: $\mathrm{Th}(\Gamma) \subseteq \mathrm{NmTh}_\Delta(\Gamma)$. This property is called *supraclassicality* in [Makinson 93]. [1]

By their very nature, defaults may be contradictory. This raises the important problem of conflicts between defaults. For example, if sailors are normally smokers, chaplains are normally non-smokers, what can we say about a boat chaplain?

The term *extension* of Γ under Δ is introduced to tackle this problem. It is the set of formulae deducible in *one* of the consistent cases. If defaults are in conflict, we get multiple extensions. Let $ext_\Delta(\Gamma)$ denote the set of all extensions of Γ under Δ. There are several general strategies to arrive at a unique entailment operator $\mathrm{NmTh}_\Delta(\Gamma)$.

A straightforward way is the *skeptic* strategy which takes the intersection of the extensions: $\mathrm{NmTh}_\Delta(\Gamma) = \cap ext_\Delta(\Gamma)$. This is sometimes called *disjunctive skepticism*. Note that if two different extensions contain γ_1 and γ_2 respectively, they also contain $\gamma_1 \vee \gamma_2$ (since they are deductively closed), so the disjunction is contained in the intersection. The *choice* strategy selects one particular extension for $\mathrm{NmTh}_\Delta(\Gamma)$. To do that, priorities between defaults are often used. The *credulous* strategy, taking the union $\cup ext_\Delta(\Gamma)$, is of little significance, as it is rarely consistent.

Reiter's Default Logic

In addition to premise and conclusion, the syntax of Reiter's defaults includes a set of justifications, written as follows:

$$\frac{P : J_1, \ldots, J_n}{C}$$

This is meant to express that if P holds and none of the negations of the justifications is provable, then also C holds. bird $\rightsquigarrow$ flies here can be written as a so-called *normal* default: $\frac{\text{bird:flies}}{\text{flies}}$

Given a set of formulae W and a set of defaults D, the extensions of W are then defined as the fixed point of the following operator $\Gamma_{D,W}(\cdot)$, mapping sets of formulae to sets of formulae.

$$\begin{aligned} \Gamma_{D,W}(S) &= \mathrm{Th}(W \cup S \cup \{C : \tfrac{P:J_1,\ldots,J_n}{C} \in D, \\ &\qquad P \in \mathrm{Th}(S), \neg J_1, \ldots \neg J_n \notin S\}) \\ \mathrm{ext}_D(W) &= \{S : S = \Gamma_{D,W}(S)\} \end{aligned}$$

[1]Simply omitting the default information in the comparison is of course a little unfair. Obviously, if defaults are interpreted as strict implications, then $\mathrm{NmTh}_\Delta(\Gamma) \subseteq \mathrm{Th}(\Gamma \cup \Delta_{strict})$, and actually $\Gamma \cup \Delta_{strict}$ is normally inconsistent.

Modal Approaches

Modal approaches do not distinguish between strict formulae and defaults, they introduce a modal operator extending the whole language, thus they are homogeneous. As an example I will give Moore's Autoepistemic Logic (AEL). A default can here be represented by the formula

$$\text{bird} \wedge \neg \mathsf{L} \neg \text{flies} \quad \rightarrow \text{flies}$$

where L is the modal operator, meaning “it is believed (or assumed) that”. Again, the semantics is defined through fixed points (A, S are sets of AEL formulae):

$$\begin{aligned} \text{NM}_A(S) &= \text{Th}(A \cup \{\mathsf{L}p : p \in S\} \cup \{\neg \mathsf{L}p : p \notin S\}) \\ \text{ext}(A) &= \{S : S = \text{NM}_A(S)\} \end{aligned}$$

Konolige [Konolige 88] showed that when this definition is restricted to the subclass of ‘strongly grounded extensions’, AEL is equivalent to Reiter's Default Logic (see also [Brewka 91]).

Preferred Subtheories

Again, no special syntax for defaults is defined, the language of logic is not extended at all.

Instead, a (usually inconsistent) set of premises D is examined for its maximal consistent subsets. Since there may be many such subtheories, a preference relation is often defined among them, provability is then defined as provability in the preferred subtheories.

Defaults would here become simple implications: bird $\rightarrow$ flies.

Strict knowledge can be represented in a separate set A of formulae. Poole [Poole 88] also uses an additional set of constraints K that are not premises but must be fulfilled by the subtheories. The extensions can simply be described as follows

$$\begin{aligned} \text{ext}_{D,K}(A) \; = \{\text{Th}(A \cup D') : \; & D' \subset D,\; D' \cup A \cup K \text{ is consistent and} \\ & \text{there is no } D'' \text{ such that} \\ & D' \subset D'' \subset D \text{ and } D' \cup A \cup K \text{ is consistent}\} \end{aligned}$$

Preferential Models

This approach was first proposed by Shoham [Shoham 87], it uses a preference ordering on models rather than on theories. Classically, a formula is entailed by a set of premises if it is valid in all models of that set. This definition is modified by defining a formula to be *preferentially entailed* if it is valid in all the preferred models of the premises.

$$\text{NmTh}(\Gamma) \; = \{\gamma : \forall M \quad \text{with } M \models \Gamma : \quad M \text{ is preferred} \Rightarrow M \models \gamma\}$$

Defaults may be represented in many ways, the preference ordering can then be defined as a function of them. For example by preferring models with fewer exceptional defaults or more default applications. Also, McCarthy's Circumscription can be formalized in this framework. (By preferring a model iff the circumscribed predicate has a smaller interpretation.)

The extensions in this approach are the sets of formulae $\mathrm{Th}(M)$ valid in preferred models M:

$$\mathrm{ext}(\Gamma) \quad = \quad \{Th(M) : M \text{ is preferred}\}$$

Then the above definition of the consequence relation implements the skeptic strategy $\mathrm{NmTh}(\Gamma) = \cap\mathrm{ext}(\Gamma)$.

2.3 The Weighted PDDL

In the following, the approach of preferential models will be used to define a Weighted Preferential Default Description Logic (WPDDL). Actually, it will be a class of such logics, since many different DLs can be extended by that nonmonotonic apparatus, provided they contain the concept-forming connectives $\sqcap, \sqcup$ and $\neg$. (That is why I will often refer to WPDDL in the plural.)

2.3.1 Definition of the Entailment Relation $\approx_\Sigma$

The syntactic format of defaults will be an adaptation of the notation used in the previous section:

Definition 1 *A* weighted default *δ has the form $\mathrm{c}_1 \leadsto_w \mathrm{c}_2$, where c_1 and c_2 are* DL *concepts, and $w \in \mathbf{N}$ is a positive natural number. c_1 is called the premise of δ (written δ_p), c_2 its conclusion (written δ_c), and w its weight (written $w(\delta)$).*

Quite different from Reiter's logic, defaults are not applied like chaining rules here. Rather, their content is used to generate a preference ordering on models.

The idea is to prefer models where most defaults are valid, or in other words, models with as few exceptions as possible. Exceptions to a default δ are objects which are instances of the interpretation of δ_p but not of the interpretation of δ_c in the model.

Definition 2 *Let δ be a weighted default, $M = \langle \mathcal{D}, [\![\cdot]\!]^{\mathcal{I}} \rangle$ a model. The* **exceptions** *to δ in M are defined as*

$$E_M(\delta) \quad \stackrel{\text{def}}{=} \quad \{\mathrm{o} \in \mathcal{O} : [\![\mathrm{o}]\!]^{\mathcal{I}} \in [\![\delta_p]\!]^{\mathcal{I}} \wedge [\![\mathrm{o}]\!]^{\mathcal{I}} \notin [\![\delta_c]\!]^{\mathcal{I}}\}$$

Only objects obtained as interpretation of object names ($o \in \mathcal{O}$) are considered here, there may be more objects in the domain $\mathcal{D}$, which can be relevant for existential quantifications or value restrictions. Since there are no 'anonymous' objects in many applications, this definition suits practical needs as well as theoretical ease. If the domain $\mathcal{D}$ were taken as a basis for exceptions, we would encounter the problem of *lifted defaults*, which is addressed in [Quantz, Royer 92].

Now every model can be assigned the sum of the weights of exceptions as its negative score for the construction of the preference relation between models. For all the following definitions, Δ will be assumed to be a finite set of weighted defaults, Γ a set of DL-formulae, and $\mathcal{O}$ a finite set of object names.

Definition 3 *Let M be a model. Then the* **negative score** *of M wrt Δ and $\mathcal{O}$ is defined as*

$$\mathbf{score}^{-}(M) \stackrel{\text{def}}{=} \sum_{\delta \in \Delta} (|E_M(\delta)| \cdot w(\delta))$$

Finiteness of Δ and $\mathcal{O}$ ensure that this score^{-} always exists.

Models with lower negative scores will be preferred by the partial ordering:

Definition 4 Σ-preference[2]
Let M, N be two models. Then N is called **preferred** *to M*

$$M \sqsubset_{\Sigma} N \quad \textit{iff} \quad \textit{score}^{-}(M) > \textit{score}^{-}(N)$$

This is 'almost' a total ordering, all models are comparable unless their score is equal. This property of $\sqsubset_{\Sigma}$ is called *ranked* in [Makinson 93].

In the following two definitions, the preferential model technique (mentioned in the previous section) is utilized to define the nonmonotonic entailment operator.

Definition 5 *Let M be a model of Γ. M is a* $\sqsubset_{\Sigma}$**-maximal** *model of Γ iff*
1. M is a model of Γ and
2. there is no model N of Γ with $M \sqsubset_{\Sigma} N$.

Definition 6 *Let Γ be a set of* DL *formulae, Δ a finite set of weighted defaults, $\sqsubset_{\Sigma}$ the corresponding ordering on* DL *models, and γ a* DL*-description, i.e. a formula of the form* o :: c. *Then γ is* Σ**-entailed** *by Γ and Δ*

$\Gamma; \Delta \mathrel{|\!\!\approx_{\Sigma}} \gamma$ *iff all $\sqsubset_{\Sigma}$-maximal models of Γ are models of γ.*

[2]In accordance with [Quantz, Ryan 93], the symbol Σ is merely used for distinguishing this numerical approach from other kinds of PDDL.

In addition to the entailment of object descriptions, I will also define entailment on the concept-level. As the PDDL approach is heterogeneous, a relation between concepts can be expressed in two ways — as a (strict) subsumption or as a default. Though an interpretation can also be found for $\Gamma; \Delta \approx_{\Sigma} \mathrm{c}_1 \sqsubseteq \mathrm{c}_2$ (deriving strict subsumption from default knowledge), I will define only what it means to derive a *default subsumption* $\mathrm{c}_1 \leadsto \mathrm{c}_2$, (i.e. a 'weightless default'). This enables us to apply the theoretical properties from [Kraus et al. 90], but is less important for practical use.

Definition 7 *Let Γ be a set of* DL *formulae, Δ a finite set of weighted defaults, and $\mathrm{c}_1 \leadsto \mathrm{c}_2$ a default subsumption.*

Then $\Gamma; \Delta \approx_{\Sigma} \mathrm{c}_1 \leadsto \mathrm{c}_2$ iff for any new object-name o_n not occurring in Γ or Δ we have $\Gamma \cup \{\mathrm{o}_n :: \mathrm{c}_1\}; \Delta \approx_{\Sigma} \mathrm{o}_n :: \mathrm{c}_2$.

2.3.2 Some Properties of $\approx_{\Sigma}$ Illustrated

Now I am going to demonstrate the effects of this semantics in a few examples. Let us begin with a single object o, a single default $\delta_1 = \mathrm{c}_1 \leadsto_{17} \mathrm{c}_2$ and Γ just consisting of a mini ABox $\{\mathrm{o} :: \neg\mathrm{c}_2\}$.

To test what is preferentially entailed, the definition demands to check all $\sqsubset_{\Sigma}$-maximal models. Literally, there is an infinite number of these, but for this case it only matters whether o is an instance of respectively c_1 and c_2. Models interpreting o as an instance of c_2 are no models of Γ, so there are only 2 kinds of models[3] left for consideration.

In the following tables, each line represents such a relevant class of models. An 'o' below a concept name means that in this line the object name o is interpreted as belonging to the respective concept, a '$-$' says that it is not. Similarly, an o below $E(\delta_1)$ means that $\{o\}$ is the set of exceptions to δ_1.

Example 2.1 *(Contraposition)*

$\mathcal{O} = \{\mathrm{o}\}$

$\Gamma = \{\mathrm{o} :: \neg\mathrm{c}_2\}$

$\delta_1 = \mathrm{c}_1 \leadsto_{17} \mathrm{c}_2$

	c_1	c_2	$E(\delta_1)$	*score*$^-$
$+M_1$	$-$	$-$	$-$	0
M_2	o	$-$	o	17

$M_2 \sqsubset_{\Sigma} M_1 \quad \Gamma; \Delta \approx_{\Sigma} \mathrm{o} :: \neg\mathrm{c}_1$

Clearly, M_1 is preferred to M_2 since it has one exception less. (This is marked by a + sign)

This example already exhibits one feature of the preferential model semantics, not present in many other approaches to default reasoning: A default $\mathrm{c}_p \leadsto_w \mathrm{c}_c$ not only expresses "if c_p, then normally c_c", but also "if $\neg\mathrm{c}_c$, then normally $\neg\mathrm{c}_p$". This is quite clear, since an instance of $\neg\mathrm{c}_c$ but

[3] *Canonical models* could be introduced to represent such classes of models.

not of $\neg c_p$ (i.e. of c_p) is an exception to the second statement as well as to the first. On the other hand, if defaults were interpreted in the form of forward chaining trigger rules (as in Reiter's Default Logic), nothing could be concluded in this example, since δ_1 would not be applicable at o.

The following example shows how conflicts are resolved by the weights.

Example 2.2 *Conflict Resolution*

$\mathcal{O} = \{o\}$
$\Gamma = \{o :: c_1 \sqcap c_2\}$
$\delta_1 = c_1 \rightsquigarrow_5 c_3$
$\delta_2 = c_3 \rightsquigarrow_{20} c_4$
$\delta_3 = c_2 \rightsquigarrow_{10} \neg c_4$

	c_1	c_2	c_3	c_4	$E(\delta_1)$	$E(\delta_2)$	$E(\delta_3)$	*score*$^-$
$+M_1$	o	o	–	–	o	–	–	5
M_2	o	o	o	–	–	o	–	20
M_3	o	o	–	o	o	–	o	15
M_4	o	o	o	o	–	–	o	10

$M_2 \sqsubset_\Sigma M_3 \sqsubset_\Sigma M_4 \sqsubset_\Sigma M_1 \quad \Gamma; \Delta \mathrel{|\!\!\approx}_\Sigma o :: \neg c_4$

If $\delta_1, \delta_2, \delta_3$ were seen as strict implications, c_4 could be concluded from δ_1, δ_2 and $\neg c_4$ from δ_3, this is a conflict. It can be resolved by giving up any of the three defaults — there are three (classes of) models (M_1, M_2, M_4) with just one exception. Since δ_1 has the smallest weight, M_1 is the $\sqsubset_\Sigma$-maximal model. (Even though δ_1 is the first default in the chain $c_1 \rightsquigarrow c_3 \rightsquigarrow c_4$.)

Of course, weights of conflicting defaults may add up to the same score, resulting in several $\sqsubset_\Sigma$-maximal models. Then disjunctive skepticism applies, illustrated in the following example:

Example 2.3 *Disjunctive Skepticism*

$\mathcal{O} = \{o\}$
$\Gamma = \{o :: c_1 \sqcap c_2\}$
$\delta_1 = c_1 \rightsquigarrow_{17} c_3 \sqcap c_4$
$\delta_2 = c_2 \rightsquigarrow_{17} c_3 \sqcap \neg c_4$

	c_1	c_2	c_3	c_4	$E(\delta_1)$	$E(\delta_2)$	*score*$^-$
M_1	o	o	–	–	o	o	34
$+M_2$	o	o	o	–	o	–	17
M_3	o	o	–	o	o	o	34
$+M_4$	o	o	o	o	–	o	17

$\Gamma; \Delta \mathrel{|\!\!\approx}_\Sigma o :: c_3$

Another feature of PDDL is the *disjunctive application* of defaults. Consider the following:

Example 2.4 *Disjunctive Default Application*

$\mathcal{O} = \{o\}$
$\Gamma = \{o :: c_1 \sqcup c_2\}$
$\delta_1 = c_1 \rightsquigarrow_5 c_3$
$\delta_2 = c_2 \rightsquigarrow_{10} c_3$

	c_1	c_2	c_3	$E(\delta_1)$	$E(\delta_2)$	*score*$^-$
M_1	o	–	–	o	–	5
M_2	–	o	–	–	o	10
M_3	o	o	–	o	o	15
$+M_5$	o	–	o	–	–	0
$+M_6$	–	o	o	–	–	0
$+M_7$	o	o	o	–	–	0

$\Gamma; \Delta \mathrel{|\!\!\approx}_\Sigma o :: c_3$

Property	Preferential Entailment	Default Subsumption
Reflexivity	$\Gamma, \gamma; \Delta \mid\!\approx_{\Sigma} \gamma$	$\Gamma; \Delta \mid\!\approx_{\Sigma} \mathrm{c} \leadsto \mathrm{c}$
Left Logical Equivalence	$\frac{\Gamma,\alpha \models \beta \quad \Gamma,\beta \models \alpha \quad \Gamma,\alpha;\Delta \mid\!\approx_{\Sigma}\gamma}{\Gamma,\beta;\Delta \mid\!\approx_{\Sigma}\gamma}$	$\frac{\Gamma \models \mathrm{c}_1 \doteq \mathrm{c}_2 \quad \Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \leadsto \mathrm{c}_3}{\Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_2 \leadsto \mathrm{c}_3}$
RightWeakening	$\frac{\Gamma,\alpha \models \beta \quad \Gamma;\Delta \mid\!\approx_{\Sigma}\alpha}{\Gamma;\Delta \mid\!\approx_{\Sigma}\beta}$	$\frac{\Gamma \models \mathrm{c}_1 \rightarrow \mathrm{c}_2 \quad \Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_3 \leadsto \mathrm{c}_1}{\Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_3 \leadsto \mathrm{c}_2}$
Cut	$\frac{\Gamma,\alpha;\Delta \mid\!\approx_{\Sigma}\gamma \quad \Gamma;\Delta \mid\!\approx_{\Sigma}\alpha}{\Gamma;\Delta \mid\!\approx_{\Sigma}\gamma}$	$\frac{\Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \sqcap \mathrm{c}_2 \leadsto \mathrm{c}_3 \quad \Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \leadsto \mathrm{c}_2}{\Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \leadsto \mathrm{c}_3}$
Cautious Monotonicity	$\frac{\Gamma;\Delta \mid\!\approx_{\Sigma}\alpha \quad \Gamma;\Delta \mid\!\approx_{\Sigma}\gamma}{\Gamma,\alpha;\Delta \mid\!\approx_{\Sigma}\gamma}$	$\frac{\Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \leadsto \mathrm{c}_2 \quad \Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \leadsto \mathrm{c}_3}{\Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \sqcap \mathrm{c}_2 \leadsto \mathrm{c}_3}$
Or		$\frac{\Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \leadsto \mathrm{c}_3 \quad \Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_2 \leadsto \mathrm{c}_3}{\Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \sqcup \mathrm{c}_2 \leadsto \mathrm{c}_3}$
Rational Monotonicity		$\frac{\Gamma;\Delta \mid\!\not\approx_{\Sigma}\mathrm{c}_1 \sqcap \mathrm{c}_2 \leadsto \mathrm{c}_3 \quad \Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \leadsto \mathrm{c}_3}{\Gamma;\Delta \mid\!\approx_{\Sigma}\mathrm{c}_1 \leadsto \neg \mathrm{c}_2}$

Figure 2.1: Formal properties of the weighted PDDL as proven in [Quantz, Ryan 93], for both preferential entailment and default subsumption.

From the viewpoint of chaining, neither δ_1 nor δ_2 alone can be applied at o, c_3 could be concluded only by 'case analysis' of the disjunction. In the PDDL this is done implicitly (due to several differing $\sqsubset_{\Sigma}$-maximal models).

These features can be summarized by saying that the weighted defaults behave like *material implications.*

Kraus, Lehman and Magidor [Kraus et al. 90], as well as Makinson [Makinson 93] compare different approaches to nonmonotonic reasoning using formal properties of the entailment relation. Within their framework, the weighted PDDL turns out to be very well-behaved, it satisfies all the properties of system **P** from [Kraus et al. 90] and Rational Monotonicity [Makinson 93], given in Figure 2.1. This implies Cumulativity (Cautious Monotonicity, i.e. that if derivable formula are added to the premises all conclusions remain valid), a property often required for nonmonotonic logics. 'Or' and 'Rational Monotonicity' are non-Horn conditions only seldomly satisfied by other approaches.

Chapter 3

Proof Theory

To actually check an entailment $\Gamma; \Delta \mathrel{|\!\!\approx_\Sigma} \gamma$ in practice, the semantic definition of $\mathrel{|\!\!\approx_\Sigma}$ is not very useful, because it would mean to test whether γ is valid in all $\sqsubset_\Sigma$-maximal models of Γ, which are unlimited in number.

The aim of this chapter is to present a way of characterizing $\mathrel{|\!\!\approx_\Sigma}$ syntactically, i.e. in terms of DL-formulae and DL-defaults. Intuitively, the task is to find out, which defaults are actually applied and which are defeated by others or by the strict knowledge, and, in cases of stalemate, which defaults are applied in combination with others. In [Quantz, Royer 92] default spaces are introduced to handle the problem of conflicting defaults. I will use default spaces in a different manner – rather than looking for conflicts, the aim is to maximize sets of defaults that can go together in 'maximal default spaces'. A very close relationship between models and default spaces will be established that way. (This is in contrast to [Quantz, Royer 92] where not even a complete characterization of the preferential entailment could be achieved.)

In the same way as the preference semantics is defined on the basis of the classic DL semantics, the proof theory for $\mathrel{|\!\!\approx_\Sigma}$ will rely on deduction in the underlying DL. This reduction implies that the default logic is decidable if the underlying strict logic is.

3.1 Basic Definitions

In the following, let Γ be a finite set of DL-formulae, Δ a finite set of DL-defaults, and $\mathcal{O}$ a finite set of object names.

Definition 8 *A* **default space** *S (over $\mathcal{O}$, Δ) is any set of* **atoms** $\langle o, \delta \rangle$ *with $o \in \mathcal{O}, \delta \in \Delta$.*

Its **score** *is defined as* $$\mathbf{score}(S) \overset{\text{def}}{=} \sum_{\delta \in \Delta} |\{o : \langle o, \delta \rangle \in S\}| w(\delta)$$

So far, default spaces are merely sets of pairs, equipped with a score. Note that the score function is monotonous, the ordering generated by the score does not conflict with set inclusion, i.e. for all S, S' if $S \subset S'$ then $\mathrm{score}(S) < \mathrm{score}(S')$, because all the weights are positive.

The score of a model is defined through its exceptions. As we want to relate default spaces to models, the following obvious statement will be useful to express exceptions through object descriptions.

Lemma 1 *Let M be a model of Γ, $o \in \mathcal{O}, \delta \in \Delta$. Then*

$$\begin{array}{rll} o \in E_M(\delta) & \textit{iff} & M \models o :: \delta_p \sqcap \neg\delta_c \\ & \textit{iff} & M \not\models o :: \neg\delta_p \sqcup \delta_c \end{array}$$

Proof: We just have to apply some set theory to the definitions of the semantics. Let $\langle \mathcal{D}, [\![\cdot]\!]^{\mathcal{I}} \rangle = M$.

$$\begin{array}{rll} o \in E_M(\delta) & \text{iff} & [\![o]\!]^{\mathcal{I}} \in [\![\delta_p]\!]^{\mathcal{I}} \text{ and } [\![o]\!]^{\mathcal{I}} \notin [\![\delta_c]\!]^{\mathcal{I}} \\ & \text{iff} & [\![o]\!]^{\mathcal{I}} \in [\![\delta_p]\!]^{\mathcal{I}} \cap (\mathcal{D} \setminus [\![\delta_c]\!]^{\mathcal{I}}) \\ & \text{iff} & [\![o]\!]^{\mathcal{I}} \in [\![\delta_p \sqcap \neg\delta_c]\!]^{\mathcal{I}} \\ & \text{iff} & M \models o :: \delta_p \sqcap \neg\delta_c \end{array}$$

$$\begin{array}{rll} \text{and also} & \text{iff} & [\![o]\!]^{\mathcal{I}} \notin \mathcal{D} \setminus [\![\delta_p]\!]^{\mathcal{I}} \text{ and } [\![o]\!]^{\mathcal{I}} \notin [\![\delta_c]\!]^{\mathcal{I}} \\ & \text{iff} & [\![o]\!]^{\mathcal{I}} \notin ([\![\neg\delta_p]\!]^{\mathcal{I}} \cup [\![\delta_c]\!]^{\mathcal{I}}) \\ & \text{iff} & [\![o]\!]^{\mathcal{I}} \notin [\![\neg\delta_p \sqcup \delta_c]\!]^{\mathcal{I}} \\ & \text{iff} & M \not\models o :: \neg\delta_p \sqcup \delta_c \end{array}$$

■

It became clear in the previous section, that with given Γ and Δ, weighted defaults are not either 'applied' or 'defeated', due to the disjunctive skepticism of the semantics. However, when we consider a certain model, we can say when a default is applied to an object, namely when this object is no exception to that default. With the above Lemma in mind we define

Definition 9 *Let $\delta \in \Delta$ be a default.*

The **conceptual content**[1] *of δ is* $\phi(\delta) \overset{\text{def}}{=} \neg\delta_p \sqcup \delta_c$

Let S be a default space. Then $\mathbf{\Phi}(S) \overset{\text{def}}{=} \{o :: \phi(\delta) : \langle o, \delta \rangle \in S\}$

We will call $\Gamma \cup \Phi(S)$ the **application** *of S to Γ.*

What we are interested in are the default spaces with maximal score, provided that their application is consistent.

[1]Using this notion, every default δ is equivalent to $\top \rightsquigarrow_{w(\delta)} \phi(\delta)$ (since $E_M(\mathrm{c}_p \rightsquigarrow_w \mathrm{c}_c) = E_M(\top \rightsquigarrow_w \neg\mathrm{c}_p \sqcup \mathrm{c}_c)$) This can be regarded as a kind of 'normal form' for defaults.

Definition 10 *Let S be a default space.*
S is Γ-**consistent** *(or* **consistent** *wrt Γ) iff $\Gamma \cup \Phi(S)$ is consistent.*
S is Γ-**maximal** *(or* **maximal***) iff*

1. *S is Γ-consistent*
2. *for all Γ-consistent default spaces S': $score(S) \geq score(S')$*

3.2 Associating Models with Default Spaces

For a given model, each atom can be classified either as exception or non-exception. Using this, we can assign default spaces of those atoms to models. We will show later, that such a default space captures relevant features of its model.

Definition 11 *Let M be a model of Γ.*
The **supporting default space of** *M is*

$$\mathbf{S}_M \overset{\text{def}}{=} \{\langle o, \delta\rangle : M \models o :: \phi(\delta)\}$$

And the **exception space** *is*

$$\mathbf{X_M} \overset{\text{def}}{=} \{\langle o, \delta\rangle : M \not\models o :: \phi(\delta)\}$$

Then the scores are directly related

Lemma 2 *Let M be a model of Γ. Then*

$$(1) \qquad score(X_M) + score(S_M) = score(\mathcal{O} \times \Delta) = \sum_{\delta \in \Delta} |\mathcal{O}| w(\delta)$$

$$\text{and} \quad (2) \qquad score^{-}(M) = score(X_M)$$

Proof: (1) is obvious since S_M and X_M are complementary with respect to the set $\mathcal{O} \times \Delta$ of all atoms.

(2) follows directly from Lemma 1. ■

So far we were able to assign a unique 'supporting default space' to every model. Different models may have the same supporting default space, that is why we cannot straightforwardly reverse this construction to obtain a model from a default space.

But the following lemma goes a step in this direction. It states, that all models of the application of a default space contain this default space as a subset in their supporting default space. Even more, if a default space is maximal, then having the same supporting default space and being a model of its application become equivalent.

Lemma 3 *Let M be a model of Γ, S a default space.*

Then

$$M \models \Gamma \cup \Phi(S) \quad \textit{iff} \quad S \subseteq S_M$$

If S is Γ-maximal, then

$$M \models \Gamma \cup \Phi(S) \quad \textit{iff} \quad S = S_M$$

Proof: We begin with the first equivalence:

$$\begin{array}{llll} M \models \Gamma \cup \Phi(S) & \text{iff} & M \models \Gamma \text{ and } \quad \forall \langle o, \delta \rangle \in S: & M \models o :: \phi(\delta) \\ & \text{iff} & \forall \langle o, \delta \rangle \in S: & \langle o, \delta \rangle \in \mathrm{S}_M \\ & \text{iff} & S \subseteq \mathrm{S}_M & \end{array}$$

The second equivalence is almost identical, except that for the only-if part it must be shown that the set-inclusion cannot be strict. But since all the weights are positive, $S \subset \mathrm{S}_M$ would mean score(S) $<$ score(S_M) contradicting maximality of S. ■

The following lemma states that maximality carries over from models to default spaces. It may be noted that the converse also holds, i.e. that every maximal default space has a maximal model. This is indirectly implied by the previous lemma.

Lemma 4 *Let M be any model. Then*

$$M \textit{ is a } \sqsubset_\Sigma\textit{-maximal model of } \Gamma \quad \textit{iff} \quad S_M \textit{ is a } \Gamma\textit{-maximal default space.}$$

Proof: (if) We have to prove that M is a model of Γ and that score$^-(M) \leq$ score$^-(N)$ for all models N of Γ.

$M \models \Gamma$ follows from $M \models \Gamma \cup \Phi(S)$ (Lemma 3).

Let us consider any model N of Γ. Then, since S_M is a maximal default space, score(S_M) $\geq$ score(S_N), and with Lemma 2, score$^-(M) \leq$ score$^-(N)$.

(only-if) We have to prove, that score(S_M) $\geq$ score(S), for all Γ-consistent default spaces S.

Γ-consistency of S means that there exists a model $N \models \Gamma \cup \Phi(S)$. From Lemma 3 we know that $S \subseteq \mathrm{S}_N$. This means score(S) $\leq$ score(S_N) (because the weights are positive). On the other hand we have score(S_N) $\leq$ score(S_M) from $\sqsubset_\Sigma$-maximality of M and Lemma 2. So we get score(S) $\leq$ score(S_M). ■

3.3 Characterization of Entailment

The entailment operator $\approx_\Sigma$ was defined both for object descriptions and default subsumptions, first we will state how to derive descriptions.

Proposition 1 *Let* Γ *be a finite set of* DL*-formulae,* Δ *a finite set of* DL*-defaults, and* $\mathcal{O}$ *a finite set of object names.*
Let $\mathrm{o} \in \mathcal{O}$ *be an object,* c *a concept.*

$$\Gamma; \Delta \mathrel{|\!\!\approx}_{\Sigma} \mathrm{o} :: \mathrm{c} \quad \textit{iff} \qquad \Gamma \cup \Phi(S) \models \mathrm{o} :: \mathrm{c}$$

Proof: **(if)** To prove that $\Gamma; \Delta \mathrel{|\!\!\approx}_{\Sigma} \mathrm{o} :: \mathrm{c}$, consider an arbitrary $\sqsubset_{\Sigma}$-maximal model M of Γ. The proof will be done when we get $M \models \mathrm{o} :: \mathrm{c}$.

Due to Lemma 4 we know that S_M is a Γ-maximal default space. Then we can conclude from Lemma 3 that $M \models \Gamma \cup \Phi(S)$. But since we know that $\Gamma \cup \Phi(\mathrm{S}_M) \models \mathrm{o} :: \mathrm{c}$ (premise), M must also be a model of $\mathrm{o} :: \mathrm{c}$.

(only-if) Consider a maximal default space S. We have to prove, that any model of $\Gamma \cup \Phi(S)$ is also a model of $\mathrm{o} :: \mathrm{c}$.

Let M be such a model. Since S is a maximal default space, we know that $S = \mathrm{S}_M$ (Lemma 3). From Lemma 4 it follows that M is a $\sqsubset_{\Sigma}$-maximal model of Γ. From the premise $\Gamma; \Delta \mathrel{|\!\!\approx}_{\Sigma} \mathrm{o} :: \mathrm{c}$ we thus know that $M \models \mathrm{o} :: \mathrm{c}$. ■

Corollary 2 *Let* Γ *be a finite set of* DL*-formulae,* Δ *a finite set of* DL*-defaults, and* $\mathcal{O}$ *a finite set of object names.*
Let o *be an new object, occurring neither in* Γ *nor in* Δ*, let* c_1*,* c_2 *be concepts.*

$\Gamma; \Delta \mathrel{|\!\!\approx}_{\Sigma} \mathrm{c}_1 \rightsquigarrow \mathrm{c}_2$
iff
for all $\Gamma \cup \{\mathrm{o} :: \mathrm{c}_1\}$*-maximal default spaces* S *over* $\mathcal{O} \cup \{\mathrm{o}\}$*,* Δ
$$\Gamma \cup \{\mathrm{o} :: \mathrm{c}_1\} \cup \Phi(S) \models \mathrm{o} :: \mathrm{c}_2$$

Proof: This follows immediately from the previous proposition when Definition 7 (entailment of default subsumption) is applied. ■

This result is a sound and complete characterization of the preferential entailment operator $\mathrel{|\!\!\approx}_{\Sigma}$. It means that it is sufficient to know all the maximal default spaces to perform deduction in our PDDL. Their number is finite, this ensures decidability of entailment, if the underlying Description Logic is itself decidable.

However, in the worst case there may be $\binom{n}{\lfloor n/2 \rfloor}$ different maximal default spaces ($n = |\mathcal{O}| \cdot |\Delta|$), for instance when all defaults have the same weight and all sets of cardinality $\geq n/2$ are inconsistent, all others consistent. (For proving the upper bound, consider that if a default space of cardinality k is maximal, all its 2^k subsets and 2^{n-k} supersets cannot be maximal.)

It is worth noting, that the characterization of the PDDL in terms of default spaces corresponds exactly to the preferred subtheories as in [Brewka 91] which is a generalization of Poole's approach. If $\Phi(\mathcal{O} \times \Delta)$ is taken as the set of assumptions, a subtheory can be defined as preferred over another, iff the score of its corresponding default space is higher.

3.4 Merging Maximal Default Spaces

In natural language disambiguation, as well as in many other tasks, a unique maximal default space S normally is to be expected. Then the application $\Gamma \cup \Phi(S)$ of this default space to the strict knowledge is an ordinary DL representation of the *whole* knowledge (defeasible and strict), and thus straightforward DL deduction derives the preferentially entailed consequences. It would be desirable, if such a *single* representation could also be found for *multiple* default spaces.

From the perspective of consequence relations ($\mathrm{NmTh}_{.}(\cdot)$, as introduced in Chapter 2), default spaces are viewed as generating sets of extensions of Γ. Then Proposition 1 expresses that the set of consequences $\mathrm{NmTh}_{\Delta}(\Gamma)$ is the intersection of the sets of formulae derivable in each of the Γ-maximal default spaces over Δ. In normal logic, the intersection $\mathrm{Th}(A) \cup \mathrm{Th}(B)$ of consequences of two sets A, B of formulae can be generated by the disjunction of two conjunctions of all the formulae of each set $(a_1 \wedge a_2 \wedge \ldots) \vee (b_1 \wedge b_2 \wedge \ldots)$. (This is usually rewritten to a 'long' conjunction of disjunctions.) But in DL there are no logical connectives on the description level. This is not a problem for conjunction, as it is implicit between formulae. Disjunction of object descriptions, however, can only be expressed if all the described objects are identical — by taking the disjunction of the describing concepts.

Following the terminology of [Quantz, Royer 92] we will now define defaults as being *active* or *cancelled*, to formalize the intuition that usually defaults are either valid or defeated (for a given object), i.e. that the object is (or is not) an exception in *all* the $\sqsubset_\Sigma$-maximal models. This can also be expressed by saying that the corresponding atom is (or is not) contained in all Γ-maximal default spaces.

If there exist several different maximal default spaces, an atom contained in on of them but not in another must be replaced by (at least) one other atom, otherwise maximality can not hold. In a sense, those atoms neutralize, they are applied disjunctively. The term *neutralization* is used in [Quantz, Royer 92] in this case to refer to sets which contain just enough atoms such that in every $\sqsubset_\Sigma$-maximal model at least one of them is non-exceptional. Then, their disjunction can be regarded as valid. But, as has been remarked, this is usually not expressible in DL.

Definition 12 *Let $o \in \mathcal{O}$ be an object, $\delta \in \Delta$ a default. Then we define*

δ is **active** *at o*	*iff*	*$\langle o, \delta \rangle$ is contained in all the Γ-maximal default spaces*
δ is **cancelled** *at o*	*iff*	*$\langle o, \delta \rangle$ is contained in none of the Γ-maximal default spaces.*

These terms will also be used for atoms here, i.e. $\langle o, \delta \rangle$ is active/cancelled iff δ is active/cancelled at o.

Definition 13 *Let $\breve{S} = \{\langle o_1, \delta_1\rangle, \ldots, \langle o_m, \delta_m\rangle\}, m \geq 2$ be a set of atoms.*

$\breve{S}$ *is* **valid**	*iff*	*i.e. every Γ-maximal default space S contains an atom $\langle o_i, \delta_i\rangle$ from $\breve{S}$.*
$\breve{S}$ *is a* **neutralization**	*iff*	*$\breve{S}$ is a valid default space and and no proper subset of $\breve{S}$ is valid*

Obviously, if an atom is not cancelled, it must be contained in some valid default space. (Take for example the union of all maximal default spaces.) By successively removing atoms, we can minimize every valid set, to obtain either a neutralization or a single active atom. To summarize, every atom $\langle o, \delta\rangle$ is either active, cancelled or a member of some neutralization.

Now we will prove, that neutralizations are applied disjunctively. We can only do this for neutralizations containing the same object in each of its atoms.

Lemma 5 *Let $\breve{S} = \{\langle \mathrm{o}, \delta_1\rangle, \ldots, \langle \mathrm{o}, \delta_m\rangle\}$ be a default space. Then*

$$\breve{S} \textit{ is valid} \quad \textit{iff} \quad \Gamma; \Delta \mathrel{|\!\!\approx}_\Sigma \mathrm{o} :: \phi(\delta_1) \sqcup \ldots \sqcup \phi(\delta_m)$$

Proof: $\breve{S}$ is valid iff

$\Leftrightarrow \quad \forall S$ being Γ-maxi*mal* $\quad \breve{S} \cap S \neq \emptyset$

The following statement restricts quantification to those maximal default spaces, that can be generated as the supporting default spaces of a model, so this statement clearly follows. However, it is even *equivalent*, since every maximal default space S can be generated by a model: From Γ-consistency we can conclude that a model M of $\Gamma \cup \Phi(S)$ exists and then with Lemma 3 and Γ-maximality that $S = \mathrm{S}_M$

$\Leftrightarrow \quad \forall M$ with S_M being Γ-maxi*mal* $\quad \breve{S} \cap S \neq \emptyset$

$\Leftrightarrow \quad \forall M$ with S_M being Γ-maxi*mal* $\quad \exists i \leq m : \langle \mathrm{o}, \delta_i\rangle \in S$

With Lemma 4 and Definition 11

$\Leftrightarrow \forall M$ being a $\sqsubset_\Sigma$-maximal model of $\Gamma \quad \exists i \leq m : \ M \models o :: \phi(\delta_i)$

By applying the definition of disjunction

$\Leftrightarrow \forall M$ being a $\sqsubset_\Sigma$-maximal model of $\Gamma \quad M \models \mathrm{o} :: \phi(\delta_1) \sqcup \ldots \sqcup \phi(\delta_m)$

$\Leftrightarrow \quad \Gamma; \Delta \mathrel{|\!\!\approx}_\Sigma \mathrm{o} :: \phi(\delta_1) \sqcup \ldots \sqcup \phi(\delta_m)$

■

The following corollaries follow immediately. The first is trivial anyway, the second is useful if a neutralization is local to a fixed object.

Corollary 3 *Let Γ be a finite set of* DL*-formulae, Δ a finite set of* DL*-defaults, and $\mathcal{O}$ a finite set of object names.*
Let $o \in \mathcal{O}$ be an object, δ a default.

$$\delta \textit{ is active at } o \quad \textit{iff} \quad \Gamma; \Delta \mid\!\approx_\Sigma o :: \phi(\delta)$$

Corollary 4 *Let Γ be a finite set of* DL*-formulae, Δ a finite set of* DL*-defaults, and $\mathcal{O}$ a finite set of object names.*
Let $o \in \mathcal{O}$ be an object, $\delta_1, \ldots, \delta_m \in \Delta$ be defaults.
Then $\check{S} = \{\langle \mathrm{o}, \delta_1 \rangle, \ldots, \langle \mathrm{o}, \delta_m \rangle\}$ is a neutralization iff

$$\begin{array}{l} \Gamma; \Delta \mid\!\approx_\Sigma \mathrm{o} :: \phi(\delta_1) \sqcup \ldots \sqcup \phi(\delta_m) \quad \textit{and} \\ \Gamma; \Delta \mid\!\not\approx_\Sigma \mathrm{o} :: \phi(\delta_1) \sqcup \ldots \sqcup \phi(\delta_{i-1}) \sqcup \phi(\delta_{i+1}) \sqcup \ldots \sqcup \phi(\delta_m) \quad \forall i \leq m \end{array}$$

From a theoretical point of view one could expect most atoms to be interrelated with many others, and thus forming huge 'neutralizations'. But it can be assumed for many practical applications (cf. Chapter 5) that usually only a few defaults form neutralizations. Active defaults, although only a marginal case in theory, are the usual case to expect in practice.

In cases, where there are not many neutralizations, and all of them are local, the alternative applications of default spaces can be expressed by the (single) set of descriptions containing $\mathrm{o} :: \phi(\delta)$ for all active atoms and $\mathrm{o} :: \phi(\delta_1) \sqcup \ldots \sqcup \phi(\delta_m)$ for all neutralizations.

Chapter 4

A Complete Algorithm

Having shown that the preferential entailment is fully described by the maximal default spaces, this chapter discusses a basic algorithm to build them. The algorithm will rely on the DL system for all logical operations, this implies that we will regard default spaces just as sets of unstructured atoms.

Before the full algorithm is presented, a naive approach will be given, as a means of introducing the computational setting and the used notation. The correctness of the second algorithm will be formally proven, and some optimizations will be suggested. Its complexity will be evaluated in the following chapter.

4.1 Introduction

Because all subsets of a consistent default space are consistent too, we can incrementally enlarge consistent sets by adding the given atoms as long as the set remains consistent, starting from the empty default space. This fits very well to the machinery of DL systems: Facts are told sequentially to the system, if a new fact is inconsistent with the knowledge stored already, the system will reject it. Accordingly, we will first input the strict knowledge, and then, for each object, DL-descriptions from the conceptual content of each default.

Usually, there are several default spaces to be considered, but we do not want to initialize the knowledge base every time we try to build a new maximal default space. However, some systems are capable of handling alternative worlds, or *situations* simultaneously. We will base our algorithm on such a system.

4.1.1 Notational Conventions

Given a set of strict formulae Γ, a set of DL-defaults Δ and a set of object names $\mathcal{O}$, the task is to compute the set

$$\begin{array}{rl}\mathsf{MaxDS} \overset{\text{def}}{=} \{S \subseteq \mathcal{O} \times \Delta : & S \text{ is } \Gamma\text{-consistent and score}(S) \geq \text{score}(S') \\ & \text{for all consistent default spaces} S'\}\end{array}$$

We will call $\mathsf{AU} \overset{\text{def}}{=} \mathcal{O} \times \Delta$ the *atom universe*. For a set $\mathcal{S} \in 2^{\mathsf{AU}}$ of default spaces let $\max_\Sigma(\mathcal{S})$ be the set of all those with highest score:

$$\max\nolimits_\Sigma(\mathcal{S}) \quad \overset{\text{def}}{=} \quad \{S_{Max} \in \mathcal{S} : \forall S \in \mathcal{S} \quad \text{score}(S_{Max}) \geq \text{score}(S)\}$$

Then

$$\mathsf{MaxDS} \quad = \quad \max\nolimits_\Sigma(\{S \subseteq \mathsf{AU} : S \text{ is consistent}\})$$

A Prolog notation will be used to specify the algorithms. The arguments will be mostly lists and special data structures introduced below. Whenever the interpretation is clear, we will also use set notation. Following standard Prolog conventions, predicate arguments may be preceded by signs, indicating whether the variable is an input (+) or an output argument (–).

Data Structures

To bring out the essence of the algorithm more clearly, some parameters of predicates will be grouped together in small data structures. The exact technical details may be found in the appendix, but are unimportant for this chapter.

Default Spaces. The variables Space, NewSpace... contain all the current information about a default space in a so-called space-structure. This is basically a list of the atoms which are already in that default space (and the value of the score) and a reference to the DL-situation representing the application of that default space. It may be implemented like space(DL_Ref, S, score(S)), but we will not need to refer to this when specifying an algorithm.

For modifying default spaces there is a predicate add_atom(+Atom, +Space, –NewSpace), which tries to add the content of Atom to the knowledge in the situation of Space. If this succeeds, NewSpace will be the updated description (of $\mathsf{Space} \cup \{\mathsf{Atom}\}$), else the predicate fails.

Maxima. A current set of the 'best' default spaces encountered so far is maintained throughout the course of the computation. The variables Max, OldMax, NewMax,..., being so-called max-structures, are used to refer to them, basically they are lists of lists of atoms together with the common score of these lists, (for example: $\mathsf{max}([\mathsf{Space}_1, \ldots, \mathsf{Space}_k], \mathsf{Score})$). A max-structure is called *valid*, if all the default spaces in it have the same score. For valid max-structures we can extend the definition of the score-function to return the common score.

They are usually modified by the predicate update_max(+Space, +OldMax, –NewMax), which results in

$$\text{NewMax} = \begin{cases} \{\text{Space}\} & \text{if score(Space)} > \text{score(OldMax)} \\ \{\text{Space}\} \cup \text{OldMax} & \text{if score(Space)} = \text{score(OldMax)} \\ \text{OldMax} & \text{if score(Space)} < \text{score(OldMax)} \end{cases}$$

4.1.2 A Simple Algorithm

All maximal default spaces can be obtained by generating all the consistent default spaces and picking those with maximal score. The following predicate does this by a simple backtracking strategy, i.e. depth-first search in the tree of all subsets of the atom universe AU. Note that Prolog-backtracking is not used here, it would have meant to use the (inefficient) Prolog database to store the current set of maximal spaces, or a 'bagof'-kind predicate. So the predicate max_spaces, as all others of this chapter, is determinate. The top-level call is max_spaces(AU, EmptySpace, EmptyMax, ResultMax).

```
max_spaces([], Space, OldMax, NewMax):–
        update_max(Space, OldMax, NewMax).
max_spaces([Atom|Atoms], Space, OldMax, ResultMax):–
        (add_atom(Atom, Space, NewSpace) →
                max_spaces(Atoms, NewSpace, OldMax, Max);
        Max = OldMax),
        max_spaces(Atoms, Space, Max, ResultMax).
```

A call to max_spaces(FreeAtoms, Space, OldMax, NewMax) searches all those superspaces of Space which contain only atoms of FreeAtoms in addition to the atoms of Space. If the score of the best of these spaces is at least that of OldMax, NewMax will be updated accordingly, otherwise it will keep the value of OldMax. In formal terms

$$\text{NewMax} = \max_{\Sigma}(\text{OldMax} \cup \{S : \text{Space} \subseteq S \subseteq \text{Space} \cup \text{FreeAtoms}, \; S \text{ is consistent}\})$$

The predicate works in a very simple way: If FreeAtoms is empty, Space is a leaf of the search tree. It must be consistent, since it has been consistent all the way up. Thus, the set of best default spaces gets updated. If FreeAtoms contains at least one Atom, the predicate calls itself again two times – with this atom added to Space and without it. But the first case can be skipped if adding the Atom would lead to an inconsistent NewSpace; this way a whole subtree of the search tree is pruned. This case distinction corresponds to the following split in the searched set:

$$\begin{aligned}\mathsf{NewMax} = \max\nolimits_{\Sigma}(&\mathsf{OldMax} \cup \\ &\{S : \mathsf{Space} \subseteq S \subseteq \mathsf{Atoms} \wedge S \text{ is consistent} \wedge \mathsf{Atom} \in S\} \cup \\ &\{S : \mathsf{Space} \subseteq S \subseteq \mathsf{Atoms} \wedge S \text{ is consistent} \wedge \mathsf{Atom} \notin S\})\end{aligned}$$

There is one major source of inefficiency of this algorithm. Often, an atom can be added independently of the others, meaning that Space $\cup$ {Atom} $\cup$ Atoms is consistent whenever Space $\cup$ Atoms is. Then it makes no sense to consider the second case (without Atom), since the score is always lower than the first, so half the time of this predicate call is superfluous. The following algorithm improves on this.

4.2 The hurrymax Algorithm

Obviously, every proper superset of a maximal default space must be inconsistent. (Otherwise maximality of the score would be violated.) Default spaces with this property, i.e. being maximal as consistent sets, are called *maximally consistent* (abbrev. *maxcons*) default spaces. The algorithm presented now avoids many of the futile searches of the previous algorithm by considering only those of the consistent default spaces which cannot be enlarged (consistently).

The basic idea is to first construct a maximally consistent set by sequentially trying to include the given atoms, rejecting only those which violate consistency. Then any other maximally consistent set must contain at least one of the rejected atoms, otherwise it would be a superset of the first set, which is maximally consistent already.

So, we can take any of these atoms as a starting point for a new maximally consistent default space (if this atom is not already inconsistent with the strict knowledge), and then add the other atoms as described above, rejecting an atom iff it makes the set inconsistent. Again, for any resulting set, alternatives can be constructed by starting with one of its rejected atoms.

But we want to ensure, that no default space is built twice. To achieve this, we keep the originally rejected atom in the default space in addition to the new rejected atom which is chosen as the start for an alternative maximally consistent default space of the second generation.

This ensures that the search terminates, since one more atom is added for every generation of rejected atoms. However, starting with two different atoms we may still end up at the same maximally consistent default space. This can be avoided if we keep a record of atoms which should not be considered for adding to a default space anymore. Thus, we can 'forget' an atom after all sets containing this atom have been explored, before we turn to the next atom of the rejected ones. If a rejected atom cannot begin a new alternative because it alone already causes inconsistency, we can 'forget' it immediately.

The data structure already used in the previous section for incremental construction of the default spaces will be adapted to handle the problems above. It now contains a second set of atoms, those which are not permitted into this default space. A Prolog representation could look like this space(DL_Ref,In,InScore,Out,OutScore). When Space is a variable, bound to such a data structure, Space_{In} will be used to denote In(the atoms in the described default space) and Space_{Out} for the set of atoms Out (which must not be included in the default space anymore).

We now need a counterpart to add_atom, called exclude_atom(+Atom, +Space, -NewSpace). It always succeeds, adding Atom to the list of forgotten atoms of Space(Space_{Out}), while keeping Space_{In} and the knowledge base unchanged.

Knowing which atoms are definitely excluded, allows to apply an optimization: If enough atoms have been forgotten such that the score of the remaining atoms is lower than the current maximal score, we can stop searching this path immediately.

The algorithm consists of three main predicates: max_spaces, next_maxcons and alternatives. The main recursion is performed by alternatives, but max_spaces is needed for the top level call, which is max_spaces(+AU, +EmptySpace, +EmptyMax, -ResultMax).

The following predicate basically consists of the call to next_maxcons to find any maximally consistent space extending FixSpace, and the call to alternatives to search the rest.

```
max_spaces(FreeAtoms, FixSpace, OldMax, ResultMax):-
    get_score(OldMax, MaxScore),
    next_maxcons(FreeAtoms, MaxScore, FixSpace, NewSpace, Completed),
    (Completed = complete →
        update_max(NewSpace, OldMax, Max);
    Max = OldMax),
    get_new_rejected(NewSpace, FixSpace, Rejected),
    alternatives(Rejected, FreeAtoms, FixSpace, Max, ResultMax).
```

As the name suggests, get_score is just an abbreviation for retrieving the score of a max-structure, a simple unification, perhaps implemented like this:

```
        get_score(max( _ ,Score),Score).
```

It is put here only as a means of optimization, to avoid that next_maxcons has to carry around a whole max-structure just for score comparison.

The parameter Completed is due to the score optimization, if the construction of the maxcons default space was terminated because of too many excluded atoms, the set of maxima need not be updated.

The set of rejected atoms is determined by get_new_rejected. Since NewSpace also contains the old Out-atoms of FixSpace, these have to be subtracted[1].

The following predicate mainly runs through all atoms in the first parameter, adding them to the OldSpace (on third position) if this can be done consistently, otherwise excluding the atom (putting it into the set NewSpace_{Out}). The first clause guarantees the score optimization.

```
next_maxcons(_, MaxScore, Space, Space, incomplete):–
    score_exceeded(Space, MaxScore), !.
next_maxcons([], _, Space, Space, complete):–!.
next_maxcons([Atom|Atoms], MaxScore, OldSpace, NewSpace, Completed):–
    (add_atom(Atom, OldSpace, Space),
    !;
    exclude_atom(Atom, OldSpace, Space)),
    next_maxcons(Atoms, MaxScore, Space, NewSpace, Completed).
```

The same pruning mechanism is used also for alternatives: The predicate immediately succeeds (returning the old set of maximal spaces), if the excluded atoms have narrowed down FixSpace so much that no better default spaces than OldMax can be found (in that branch of the search space).

In the normal case, one of the formerly rejected atoms is tried to be added to the current FixSpace. If this succeeds, the search continues with the extended FixSpace. Afterwards, in both cases, the Atom is prevented from further use and the search continues among the remaining atoms.

```
alternatives([], _, _, Max, Max):–!.
alternatives(_, _, FixSpace, OldMax, OldMax):–
    get_score(Max, MaxScore),
    score_exceeded(FixSpace, MaxScore), !.
```

[1]An implementation could look like this:

```
get_new_rejected(space( _ , _ , _ , NewOut, _ ), space( _ , _ , _ , OldOut, _ ), Rejected):–
    sets:subtract(OldOut, NewOut, Rejected).
```

```
alternatives([Atom|RejRest], FreeAtoms, FixSpace, OldMax, ResultMax):–
        delete(Atom, FreeAtoms, RestFree),
        (add_atom(Atom, FixSpace, PlusSpace) →
                max_spaces(RestFree, PlusSpace, OldMax, Max);
        Max = OldMax),
        exclude_atom(Atom, FixSpace, MinusSpace),
        alternatives(RejRest, RestFree, MinusSpace, Max, ResultMax).
```

An Example

Consider an example of the five atoms **a**(4), **b**(2), **c**(4), **d**(2) and **e**(1) — their weights are given in parenthesis — with the following maximal default spaces: $\{\mathbf{a},\mathbf{b}\},\{\mathbf{a},\mathbf{c}\},\{\mathbf{a},\mathbf{e}\},\{\mathbf{b},\mathbf{c},\mathbf{d}\},\{\mathbf{c},\mathbf{d},\mathbf{e}\}$.

The predicate max_spaces starts by calling next_maxcons. This is symbolized in the Figure 4.1 by square brackets enclosing the FreeAtoms. They are in the upper half, if they could be included in the resulting NewSpace, or in the lower if they had to be excluded. They are shown in the sequence in which the predicate uses them.

The subsequent call to alternatives is represented by a sequence of ' add ⓧ' underneath each other, where x is the atom starting a new default space. It is a sequence, since alternatives calls itself recursively. If max_spaces is called from there, a new column is opened to the right.

For each line, the value of FixSpace is given in the table, as well as the default space returned by next_maxcons. To show the score optimization, the number after the Out-set indicates the maximal score of default spaces that can still be constructed with atoms that are not yet in the Out-set. An asterix shows the points where this is noticed by the predicate alternatives. The last four lines are actually not executed by the algorithm, they are just given as an illustration.

Let us assume that the atoms are passed to the algorithm in alphabetical order. In the beginning max_spaces calls next_maxcons, trying to add as many atoms as possible to the empty set. It succeeds with **a** and **b**, but each of the other atoms is not consistent with $\{\mathbf{a},\mathbf{b}\}$. This is the first maximally consistent default space found.

Then the three rejected atoms are tried as starting atoms of a default space, **c** being the first one. Only **a** is consistent with it, so $\{\mathbf{a},\mathbf{c}\}$ is obtained. With **c** kept in the default space (FixSpace), alternatives are computed on this sub-level, beginning by exploring all the default spaces containing **b**. (Then **d** and **e** follow.) But it turns out that neither **a** nor **e** can still be added consistently, these 'dead ends' are marked by a '†' in Figure 4.1.

When the search space is exhausted for an atom (like **b**) that was added to the FixSpace, the second part of alternatives is executed. This 'forgetting' of the atom is symbolized by a slash across the atom ($\cancel{\mathbf{b}}$).

a	**b**	**c**	**d**	**e**	FixSpace			
4	2	4	2	1	*In*	*Out*	NewSpace	
$\rightarrow \left[{}^{\mathbf{a\,b}}{}_{\mathbf{c\,d\,e}}\right]$					$\{\}$	$\{\}/13$	$\{\mathbf{a},\mathbf{b}\}$	6
add ⓒ $\rightarrow [{}^{\mathbf{a}}{}_{\mathbf{b\,d\,e}}]$					$\{\mathbf{c}\}$	$\{\}/13$	$\{\mathbf{a},\mathbf{c}\}$	8
		add ⓑ $\rightarrow \left[{}_{\mathbf{a}}{}^{\mathbf{d}}{}_{\mathbf{e}}\right]$			$\{\mathbf{b},\mathbf{c}\}$	$\{\}/13$	$\{\mathbf{b},\mathbf{c},\mathbf{d}\}$	8
			add ⓐ †		$\{\mathbf{a},\mathbf{b},\mathbf{c}\}$	$\{\}/13$	–inconsistent–	
			$\cancel{\mathbf{a}}$ add ⓔ †		$\{\mathbf{b},\mathbf{c},\mathbf{e}\}$	$\{\mathbf{a}\}/9$	–inconsistent–	
		$\cancel{\mathbf{b}}$ add ⓓ $\rightarrow [{}_{\mathbf{a}}{}^{\mathbf{e}}]$			$\{\mathbf{c},\mathbf{d}\}$	$\{\mathbf{b}\}/11$	$\{\mathbf{c},\mathbf{d},\mathbf{e}\}$	7
			add ⓐ †		$\{\mathbf{a},\mathbf{c},\mathbf{d}\}$	$\{\mathbf{b}\}/11$	–inconsistent–	
		$\cancel{\mathbf{b}}$ $\cancel{\mathbf{d}}$ add ⓔ †			$\{\mathbf{c},\mathbf{e}\}$	$\{\mathbf{b},\mathbf{d}\}/9$	–inconsistent–	
$\cancel{\mathbf{c}}$ add ⓓ $\rightarrow \left[{}_{\mathbf{a}}{}^{\mathbf{b}}{}_{\mathbf{e}}\right]$					$\{\mathbf{d}\}$	$\{\mathbf{c}\}/9$	$\{\mathbf{b},\mathbf{d}\}$	[4]
		add ⓐ †			$\{\mathbf{a},\mathbf{d}\}$	$\{\mathbf{c}\}/9$	–inconsistent–	
		$\cancel{\mathbf{a}}$* add ⓔ $\rightarrow [{}_{\mathbf{b}}]$			$\{\mathbf{d},\mathbf{e}\}$	$\{\mathbf{a},\mathbf{c}\}/5$	$\{\mathbf{d},\mathbf{e}\}$	[3]
			add ⓑ †		$\{\mathbf{b},\mathbf{d},\mathbf{e}\}$	$\{\mathbf{a},\mathbf{c}\}/5$	–inconsistent–	
$\cancel{\mathbf{c}}$ $\cancel{\mathbf{d}}$* add ⓔ $\rightarrow [{}^{\mathbf{a}}{}_{\mathbf{b}}]$					$\{\mathbf{e}\}$	$\{\mathbf{c},\mathbf{d}\}/7$	$\{\mathbf{a},\mathbf{e}\}$	5
		add ⓑ †			$\{\mathbf{b},\mathbf{e}\}$	$\{\mathbf{c},\mathbf{d}\}/7$	–inconsistent–	

Figure 4.1: example

Two default spaces are found, that are not maximally consistent, their score is given in brackets. This can be avoided by further refining the algorithm, as suggested at the end of this chapter.

4.3 Verification

In this section it will be formally proven, that the algorithm is sound and complete, i.e. that it outputs *all* and *not more than* the maximal default spaces. We are able to show both parts simultaneously, i.e. that the algorithm produces *exactly* the set of all maximal default spaces.

Proposition 5 *Let Γ be a set of* DL*-formulae, Δ a set of* DL*-defaults, and $\mathcal{O}$ a set of object names.*

Let AU *be a list of all atoms in $\mathcal{O} \times \Delta$, let* EmptySpace *be the* space*-structure composed of a* DL*-situation representing Γ and empty sets* $\mathsf{EmptySpace}_{In}$ *and* $\mathsf{EmptySpace}_{Out}$*. Let* EmptyMax *be the empty* max*-structure and let* ResultMax *be an uninstantiated variable.*

Then

$\mathsf{max\_spaces}(\mathsf{AU}, \mathsf{EmptySpace}, \mathsf{EmptyMax}, \mathsf{ResultMax})$

succeeds with $\mathsf{ResultMax}$ *instantiated to a* max*-structure of all the* Γ*-maximal default spaces over* $\mathcal{O}$ *and* Δ*:*

$$\mathsf{ResultMax} = max_{\Sigma}\Big(\{S \subseteq \mathsf{AU} : S \textit{ is consistent}\}\Big).$$

To prove this, we need a number of assumptions about the auxiliary predicates:

1. If the space-structure Space is a representation of a consistent default space, and Atom represents an atom, then

   $\mathsf{add\_atom}(+\mathsf{Atom}, +\mathsf{Space}, -\mathsf{NewSpace})$ succeeds iff $\mathsf{Space} \cup \{\mathsf{Atom}\}$ is consistent.

   Then $\mathsf{NewSpace}$ is instantiated to a space-structure representing this set, where $\mathsf{NewSpace}_{Out}{=}\mathsf{Space}_{Out}$ remains unchanged.

2. $\mathsf{exclude\_atom}(+\mathsf{Atom}, +\mathsf{Space}, -\mathsf{NewSpace})$ always succeeds, instantiating $\mathsf{NewSpace}$ with the same as Space, except that $\mathsf{NewSpace}_{Out} = \mathsf{Space}_{Out} \cup \{\mathsf{Atom}\}$.

3. $\mathsf{update\_max}(+\mathsf{Space}, +\mathsf{OldMax}, -\mathsf{NewMax})$ iff
   $$\mathsf{NewMax} = \max_{\Sigma}(\mathsf{OldMax} \cup \{\mathsf{Space}_{In}\}).$$

4. $\mathsf{get\_score}(+\mathsf{Max}, -\mathsf{Score})$ iff $\mathsf{Score} = \mathrm{score}(\mathsf{Max})$.

5. $\mathsf{get\_new\_rejected}(+\mathsf{NewSpace}, +\mathsf{OldSpace}, -\mathsf{Rejected})$ iff
   $$\mathsf{Rejected} = \mathsf{NewSpace}_{Out} \setminus \mathsf{OldSpace}_{Out}.$$

6. $\mathsf{score\_exceeded}(+\mathsf{Space}, +\mathsf{MaxScore})$ iff $\mathsf{MaxScore} > \mathrm{score}(\mathsf{Space})$.

We can interpret the space-structures as describing families of possible default spaces, namely those which contain all atoms of Space_{In}, but none of Space_{Out}:

Definition 14

$$\textit{Let} \quad range(\mathsf{Space}) \overset{\text{def}}{=} \{S : \mathsf{Space}_{In} \subseteq S \subseteq \mathsf{AU} \setminus \mathsf{Space}_{Out} \textit{ and } S \textit{ is consistent}\}$$

For the special case of $\mathsf{Space}_{In} \cup \mathsf{Space}_{Out} = \mathsf{AU}$ (Space is complete), clearly $range(\mathsf{Space}) = \{\mathsf{Space}\}$.

We say that Space is a **consistent** space-structure, iff Space_{In} and Space_{Out} are disjoint subsets of AU and the application Space is consistent, and the internal DL representation does correspond to $\Gamma \cup \Phi(\mathsf{Space})$. We will see that only consistent space-structures occur in the course of the computation.

4.3.1 The Predicate next_maxcons

The following lemma basically expresses that in a call to next_maxcons(+Free, +MaxScore, +Space, −NewSpace, −Completed), NewSpace is obtained by consistently adding atoms of Free to Space.

Lemma 6 *If*

(p1) Space *is a consistent* space-*structure, and*
(p2) $\mathsf{Free} = \mathsf{AU} \setminus \mathsf{Space}_{In} \setminus \mathsf{Space}_{Out}$

then next_maxcons(+Free, +MaxScore, +Space, −NewSpace, −Completed) *succeeds and*

(c1) NewSpace *is a consistent* space-*structure, and*
(c2) *either* Completed=complete *and*
$\mathit{score}(\mathsf{NewSpace}) \geq \mathsf{MaxScore}$ *and*
$\mathsf{NewSpace}_{In} \cup \mathsf{NewSpace}_{Out} = \mathsf{AU}$
or Completed=incomplete *and*
$\mathit{score}(\mathsf{AU} \setminus \mathsf{NewSpace}_{Out}) < \mathsf{MaxScore}$

Proof: We will use induction on the cardinality of Free.

base case:$|\mathsf{Free}| = 0$.

Then either the first clause of next_maxcons applies, which gives us (c1) immediately from (p1) and trivially the second case of (c2). Or the second clause applies, again giving us consistency of NewSpace=Space from (p1). Since Free is empty, it follows from (p2) that $\mathsf{NewSpace}_{In} \cup \mathsf{NewSpace}_{Out} = \mathsf{AU}$. Since the first clause did not succeed, it follows from Assumption 6 that score(NewSpace) $\geq$ MaxScore — and thus we have shown (p2).

step case:$|\mathsf{Free}| > 0$.

Here only the first and the third clause match. If the first clause succeeds we obtain (c1) and (c2) immediately, just as in the base case.

The third clause selects an Atom from Free, Premise (p2) ensures it is neither in Space_{In} nor in Space_{Out}. So, if add_atom succeeds, Assumption 1 ensures that NewSpace is consistent. If not, Atom is added to Space_{Out}, still resulting in a consistent space-structure NewSpace. It then follows that the list Atoms, i.e. all the remaining atoms of Free equals $\mathsf{AU} \setminus \mathsf{NewSpace}_{In} \setminus \mathsf{NewSpace}_{Out}$. Atoms contains one atom less than Free, so we can use the induction hypothesis on the call next_maxcons(Atoms, MaxScore, Space, NewSpace, Completed). Both statements (c1) and (c2) follow directly from it. ■

Note that it is not needed for the verification of the algorithm that next_maxcons returns only maximally consistent default spaces. In fact, it may happen that a non-maxcons default space is produced, namely when Space_{Out} already contains atoms which could still be consistently added to Space_{In}.

4.3.2 The Predicates max_spaces and alternatives

Lemma 7 *If*

1. FixSpace *is a consistent* space-*structure,*
2. $\mathsf{FreeAtoms} = \mathsf{AU} \setminus \mathsf{FixSpace}_{In} \setminus \mathsf{FixSpace}_{Out}$, *a list of atoms,*
3. OldMax *is a valid* max-*structure, and*
4. Rejected *is a sublist of* FreeAtoms

then
(A) the call
alternatives*(*+Rejected*,* +FreeAtoms*,* +FixSpace*,* +OldMax*,* –ResultMax*)*
succeeds,

$$\begin{aligned} \textit{with } \mathsf{ResultMax} &= \mathit{max}_{\Sigma}(\mathsf{OldMax} \cup \mathit{AltRange}) \\ \textit{where } \mathit{AltRange} &= \{S : S \in \mathit{range}(\mathsf{FixSpace}) \wedge S \cap \mathsf{Rejected} \neq \emptyset\} \end{aligned}$$

(B) the call
max_spaces*(*+FreeAtoms*,* +FixSpace*,* +OldMax*,* –ResultMax*) succeeds,*
with $\mathsf{ResultMax} = \mathit{max}_{\Sigma}(\mathsf{OldMax} \cup \mathit{range}(\mathsf{FixSpace}))$

Proof: We use induction on the number of atoms in FreeAtoms again, but we will do it for both predicates simultaneously, corresponding to the way they call each other.

1. Base Case

$|\mathsf{FreeAtoms}| = 0$.
First we consider a call to alternatives(Rejected, FreeAtoms, FixSpace, OldMax, ResultMax). In this case, always the first clause of alternatives unifies, yielding ResultMax=OldMax. Premise 4 entails that Rejected is empty, this results in *AltRange* being empty too. So it just remains to be shown that $\mathsf{ResultMax}=\max_{\Sigma}(\mathsf{OldMax})$, which follows from Premise 3.

Now let us examine max_spaces(FreeAtoms, FixSpace, OldMax, ResultMax). The parameter FreeAtoms is passed to next_maxcons; since it is empty, one of the first two clauses of next_maxcons must succeed. In either case, the returned NewSpace is equal to FixSpace. This means that Rejected is empty (due to Assumption 5). From Assumption 3 we know, that $\mathsf{Max} = \max_{\Sigma}(\mathsf{OldMax} \cup \{\mathsf{FixSpace}\})$.
Since FreeAtoms=Rejected=[], we can apply what has been said above for the call to alternatives, and conclude that ResultMax=Max. From Premise 2 we can infer $\mathit{range}(\mathsf{FixSpace}) = \{\mathsf{FixSpace}\}$, so we get
$\mathsf{ResultMax} = \max_{\Sigma}(\mathsf{OldMax} \cup \mathit{range}(()\mathsf{FixSpace}))$.

2. Step Case

Let $n \in \mathbf{N}, n > 0$.
Let us assume that the Lemma is valid when $|\mathsf{FreeAtoms}| < n$. We will prove it for any FreeAtoms, with $|\mathsf{FreeAtoms}| = n$.

Part (A): alternatives.
Again, we will first consider alternatives(Rejected, FreeAtoms, FixSpace, OldMax, ResultMax). All three clauses may unify here. If it is the first clause, i.e. Rejected=[], the conclusion follows just as easily as in the base case. If the second clause succeeds, score(AU $\setminus$ FixSpace) $<$ score(OldMax), but then all default spaces in *range*(FixSpace) have a lower score than OldMax, so $\max_\Sigma(\mathsf{OldMax} \cup \mathit{AltRange}) = \max_\Sigma(\mathsf{OldMax}) = \mathsf{OldMax}$.
Else, the third clause unifies, selecting one Atom from the list Rejected, while assigning the rest to RejRest. Atom is also a member of FreeAtoms (Premise 4), and RestFree will be the remaining atoms of FreeAtoms.

As the clause has two main parts, the idea of the proof is to show that both subsearches together exhaust the whole search space.

If add_atom(Atom, FixSpace, PlusSpace) succeeds, Assumption 1 ensures that all Premises of the lemma are fulfilled for the call to max_spaces(RestFree, PlusSpace, OldMax, Max). Since $|\mathsf{RestFree}| = n-1$, the induction hypothesis gives us $\mathsf{Max} = \max_\Sigma(\mathsf{OldMax} \cup \mathit{range}(\mathsf{PlusSpace}))$.
If add_atom does not succeed, FixSpace $\cup$ {Atom} is inconsistent, i.e. $\mathit{range}(\mathsf{FixSpace} \cup \{\mathsf{Atom}\})$is empty. But then
$\max_\Sigma(\mathsf{OldMax} \cup \mathit{range}(\mathsf{FixSpace} \cup \{\mathsf{Atom}\})) = \max_\Sigma(\mathsf{OldMax}) = \mathsf{OldMax}$
which is what the algorithm assigns to Max in this case.
So, in both cases
$\mathsf{Max} = \max_\Sigma(\mathsf{OldMax} \cup \mathit{range}(\mathsf{FixSpace} \cup \{\mathsf{Atom}\}))$.

Let us now consider the second part.
MinusSpace, the result of exclude_atom(Atom, FixSpace, MinusSpace), must be a consistent space-structure. RestFree contains exactly those atoms that are neither in $\mathsf{MinusSpace}_{In}$ nor in $\mathsf{MinusSpace}_{Out}$. We can thus apply the induction hypothesis to the call to alternatives(RejRest, RestFree, MinusSpace, Max, ResultMax). We obtain:

$$\begin{aligned} \mathsf{ResultMax} &= \max_\Sigma(\mathsf{Max} \cup \mathit{ARange}) \\ \text{with } \mathit{ARange} = &\ \{S : S \in \mathit{range}(\mathsf{MinusSpace}) \wedge S \cap \mathsf{RejRest} \neq \emptyset\} \end{aligned}$$

Since $\mathsf{MinusSpace}_{Out} = \mathsf{FixSpace}_{Out} \cup \{\mathsf{Atom}\}$

$$\begin{aligned} \mathit{ARange} = &\ \{S : S \in \mathit{range}(\mathsf{FixSpace}) \wedge \mathsf{Atom} \notin S \wedge S \cap \mathsf{RejRest} \neq \emptyset\} \\ \mathit{ARange} = &\ \{S : S \in \mathit{range}(\mathsf{FixSpace}) \wedge \mathsf{Atom} \notin S \wedge S \cap \mathsf{Rejected} \neq \emptyset\} \end{aligned}$$

Now we turn to the other set of the above union and expand Max:

$$\begin{aligned}\mathsf{ResultMax} &= \max_\Sigma(\max_\Sigma(\mathsf{OldMax} \cup \mathit{range}(\mathsf{FixSpace} \cup \{\mathsf{Atom}\})) \\ &\qquad \cup \mathit{ARange}) \\ &= \max_\Sigma(\mathsf{OldMax} \cup \mathit{range}(\mathsf{FixSpace} \cup \{\mathsf{Atom}\}) \cup \mathit{ARange})\end{aligned}$$

Also

$$\begin{aligned}\mathit{range}(\mathsf{FixSpace} \cup \{\mathsf{Atom}\}) = &\ \{S : S \in \mathit{range}(\mathsf{FixSpace}) \wedge \mathsf{Atom} \in S\} \\ = &\ \{S : S \in \mathit{range}(\mathsf{FixSpace}) \wedge \mathsf{Atom} \in S \wedge S \cap \mathsf{Rejected} \neq \emptyset\}\end{aligned}$$

The union of both sets is then

$$\mathit{AltRange} = \ \{S : S \in \mathit{range}(\mathsf{FixSpace}) \wedge S \cap \mathsf{Rejected} \neq \emptyset\}$$

Part (B): max_space.

Now let us examine max_spaces(FreeAtoms, FixSpace, OldMax, ResultMax). Here the search is split between next_maxcons and alternatives into two sets called *FirstRange* and *AltRange*

Preconditions 1...3 ensure that Lemma 6 can be applied for the call to next_maxcons. Using this, we will first show that the updated Max can be characterized by $\max_\Sigma(\mathsf{OldMax} \cup \mathit{FirstRange})$, where

$$\mathit{FirstRange} = \ \{S \in \mathit{range}(\mathsf{FixSpace}) : \mathsf{Rejected} \cap S = \emptyset\}$$

Using the definition of $\mathit{range}(\cdot)$, we can write this as

$$\begin{aligned}\mathit{FirstRange} = &\ \{S \subseteq \mathsf{AU} : S \text{ is consistent} \wedge \mathsf{FixSpace}_{In} \subseteq S \ \wedge \\ &\qquad S \cap \mathsf{FixSpace}_{Out} = \emptyset \ \wedge\ \mathsf{Rejected} \cap S = \emptyset\} \\ = &\ \{S \subseteq \mathsf{AU} : S \text{ is consistent} \wedge \mathsf{FixSpace}_{In} \subseteq S \ \wedge \\ &\qquad S \cap \mathsf{NewSpace}_{Out} = \emptyset\}\end{aligned}$$

If Completed=incomplete, we know that this set does not contain default spaces with better score than OldMax. In the other case, NewSpace is consistent, and $\mathsf{NewSpace}_{In}$ contains all the atoms that are not in $\mathsf{NewSpace}_{Out}$. As a result, all default spaces in *FirstRange* must be subsets of $\mathsf{NewSpace}_{In}$ and their score cannot be higher. This can be summarized by

$$\max_\Sigma(\mathsf{OldMax} \cup \mathit{FirstRange}) = \max_\Sigma(\mathsf{OldMax} \cup \{\mathsf{NewSpace}\})$$

And this is what is assigned to Max by update_max.

As we get $\mathsf{Rejected} \subseteq \mathsf{FreeAtoms}$ from Assumption 5 and as the other three premises are obviously fulfilled, we can use the result of the first part of this proof to describe the result of the call to alternatives(Rejected, FreeAtoms, FixSpace, Max, ResultMax).

$$\begin{aligned}\mathsf{ResultMax} = &\ \max_\Sigma(\mathsf{Max} \cup \mathit{AltRange}) \\ \text{where } \mathit{AltRange} = &\ \{S : S \in \mathit{range}(\mathsf{FixSpace}) \wedge S \cap \mathsf{Rejected} \neq \emptyset\}\end{aligned}$$

Obviously, *AltRange* and *FirstRange* are complementary:

$$\begin{aligned}\mathsf{ResultMax} = &\ \max_\Sigma(\max_\Sigma(\mathsf{OldMax} \cup \mathit{FirstRange}) \cup \mathit{AltRange}) \\ = &\ \max_\Sigma(\mathsf{OldMax} \cup \mathit{FirstRange} \cup \mathit{AltRange}) \\ = &\ \max_\Sigma(\mathsf{OldMax} \cup \{S : S \in \mathit{range}(\mathsf{FixSpace})\}) \\ = &\ \max_\Sigma(\mathsf{OldMax} \cup \mathit{range}(\mathsf{FixSpace}))\end{aligned}$$

This is what had to be proven. ■

Now Proposition 5 immediately follows from Lemma 7, since

$$\begin{aligned}&\max_\Sigma(\mathsf{EmptyMax} \cup \mathit{range}(\mathsf{EmptySpace}))\\ &\quad= \max_\Sigma(\emptyset \cup \{S \in \mathsf{AU} : S \text{ is consistent}\}\\ &\quad= \mathsf{MaxDS}\end{aligned}$$

4.4 Further Optimizations

The aim of this chapter was to present a simple algorithm that is sound and complete in producing the maximal default spaces, and thus showing a way of reducing deduction in $\approx_\Sigma$ to deduction and complexity check in the underlying DL.

Avoiding Non-maxcons Default Spaces

As has been pointed out already, the presented algorithm still constructs some default spaces that are not maximally consistent and are thus superfluous in the search. This happens because not all the information from the encountered default spaces is used: Only the atoms rejected by the (currently) last maxcons default space are considered, but actually, any new default space must contain an atom from *every* exception set, if it is to be maximally consistent.

This membership check could be implemented using a data structure instead of the Rejected sets, which contains a list of all sets of atoms rejected previously and how far these have already been explored. The call to add_atom would then be replaced by a more general procedure start_space finding a set of atoms that both is consistent with the knowledge base and has a nonempty intersection with each of the exception sets, adding the atoms to the KB and updating the Rejected-structure.

Of course, for no input this algorithm needs more add_atom-operations than the original one, so it should always be given preference. But there may also be cases, where the first version is sufficient.

Reducing the Input

There may be several atoms $\langle o, \delta \rangle$ whose content is already deducible from the strict knowledge ($\Gamma \models o :: \neg\delta_p \sqcup \delta_c$), and others, already inconsistent with it ($\Gamma \cup \{o :: \neg\delta_p \sqcup \delta_c\} \models \bot$). Clearly, they do not need to be considered for searching default spaces and can be eliminated from the input. After all the strict knowledge has been fed into the DL-system, we can simply run through all atoms and ask the system whether $o :: \neg\delta_p \sqcup \delta_c$ can be concluded.

To check for inconsistent atoms, we simply ask whether $o :: \delta_p \sqcap \neg\delta_c$ can be concluded, instead of using add_atom (what the algorithm would do).

Of course it has a considerable impact on an exponential algorithm, if the input consists of disjoint subsets which may be treated separately.

Another point is the ordering of the atoms, this determines the first maximally consistent default space generated, which in turn can affect the course of the computation enormously. These and other optimization issues are addressed in the next chapter.

Chapter 5

Efficient Computation

Now I am going to address some aspects concerning practical applications of the weighted PDDL. The most problematic issue is to keep the nonmonotonic deduction time reasonably small. This time depends on several factors: the speed of the DL-system, the input, and the algorithm. The first two factors are largely unknown to us, but the algorithm must be adapted to these if the application is to be usable.

For theoretical considerations the relationship between input and execution time is almost always much too complicated to be captured directly by theoretical considerations. That is why usually only the worst case of a class of inputs (e.g. inputs with the same length) is analyzed to obtain an upper bound for time complexity. Average complexity would be a much better measure, but it is by far not obvious how to obtain the average. In principle, almost every input is possible, in practice most inputs occur next to never.

Assumptions or intuitions about usual application scenarios are often expressed as heuristics. These are guiding principles to speed up the search for normal cases, but do not improve (or even worsen) the time complexity in the worst case.

I will base many heuristic considerations on a natural language application for the WPDDL, as proposed in [Quantz, Schmitz 94]. The usual input here would be a sentence, or some other form of utterance, introducing new objects for the words and phrases contained in it. These objects are described as instances of some concepts of the terminology, which contain the initial information about the objects (perhaps only a phonological representation and their position in the sentence). The number of defaults is large, but only a fraction of them is actually significant for any given sentence, the others are active (in the sense of Chapter 3), with their premise being unfulfilled.

5.1 Complexity Results

In this section we will take a closer look at the *asymptotic complexity* of a special class of algorithms (containing hurrymax), to give a rough idea of what can and what cannot be achieved by algorithms transferring all the logical work to the DL-system.

Here only time-complexity will be examined, i.e. how the running time increases with input length. Most considerations are focussed on classifying algorithms whether their running time is bounded by a polynomial in the input length. A problem class is called *tractable*, if there is an algorithm solving it in polynomial time.

The class of algorithms considered here is severely restricted by the access to the input.
An algorithm may only

1. query whether an arbitrary subset[1] of the given atoms AU is consistent with the given knowledge Γ, or
2. retrieve the weight of an atom of AU.

Under this perspective, the task is viewed as an optimization problem, namely the maximization of the score w.r.t. the constraint imposed by consistency.

The complexity evaluations here are orientated more on this class of algorithms than on needs of practical applications. That is why I will regard the number of atoms in AU as input size, because Γ is directly accessed only by the DL-system.

Let $\text{TIME}_{\mathsf{algo}}(AU,\Gamma)$ denote the running time of algorithm algo on $\langle AU,\Gamma\rangle$ in terms of a fixed set of elementary operations. It does not matter which set is used precisely, as it only influences time by a constant factor.

The time the DL-system needs to answer a consistency query is not constant, usually increasing with the size of the KB. We will call $\text{Q-TIME}(AU,\Gamma)$ the maximal time needed to test a subspace of AU for consistency with Γ. The number of queries an algorithm poses to the DL-system is one of the main characteristics of its complexity, it will be called $\text{QUERIES}_{\mathsf{algo}}(AU,\Gamma)$.

We are only interested in the worst case, that is the highest value of $\text{TIME}(AU,\Gamma)$ and $\text{QUERIES}(AU,\Gamma)$ for given input length $|AU|$. It is usual practice to consider only asymptotic relations, conveniently expressed by the $O(\cdot)$ operator. Let f, g be two functions (on natural numbers), then

$$g(n) = O(f(n)) \quad \text{iff} \quad \exists c \in \mathbf{N}\ \forall^{\infty} n \ \ g(n) \leq c \cdot f(n)$$

(g is of the order of f)

[1]Note, that the hurrymax algorithm uses an even more restricted class of queries: add_atom(A,S,_) succeeds iff $S \cup \{A\}$ is consistent.

Here the '=' symbol does not, of course, express equality. The notation should rather be taken to mean that g is in the *set of all* functions asymptotically bounded by f (more precisely: bounded by a multiple of f for almost all arguments n.)

The Complexity of the hurrymax Algorithm

Since the hurrymax algorithm performs only a limited (by a constant) number of steps between two successive queries, its complexity can be bounded by

$$\textsc{time}_{\mathsf{hurrymax}}(AU,\Gamma) \quad = \quad O(\textsc{queries}_{\mathsf{hurrymax}}(AU,\Gamma)\cdot\textsc{q-time}(AU,\Gamma))$$

Thus knowing $\textsc{queries}(AU,\Gamma)$ allows us to reduce time-complexity to that of the DL-system. However, as shown in [Donini et al. 91], Description Logics (of some minimal expressiveness) are intractable as soon as the disjunctive connective is admitted. [2]

We know already that no algorithm can do better than exponential in the worst case, it still has to be shown that the hurrymax algorithm does not do worse.

Proposition 6

$$\textsc{queries}_{\mathsf{hurrymax}}(AU,\Gamma) \quad \leq \quad |AU|\cdot 2^{|AU|}$$

Proof: Let $n = |AU|$. The initial call first invokes next_maxcons, which uses n queries, and then executes alternatives. It remains to be shown that no more than $n \cdot 2^n - n$ queries are needed by this predicate.

The proof will be by induction on the cardinality of Free. Let queries(k,n) be the number of consistency queries used by alternatives(Rejected, Free, Space, OldMax, _) where $k = |\mathsf{Free}|$. We have to show queries$(k,n) \leq n\cdot(2^k-1)$.

(base case) $k = 0$: When Free is empty, the first clause fires, succeeding immediately, with queries$(0,n) = 0 = n\cdot(2^0-1)$

(step case) $k > 0$: If the first or second clause succeed, queries(k,n)=0 which is lower than $n\cdot(2^k-1)$

If the third clause succeeds, one query will be made immediately. Depending on the result, max_spaces will be called or not. Let us consider the (worst) case that it is called. Then at most $n-1$ queries are made by next_maxcons (at least one atom has already been included by alternatives), and alternatives is called again, this time with one atom deleted from Free. So, from the induction hypothesis we know that queries$(k-1,n) \leq n\cdot(2^{k-1}-1)$.

[2] Note that already the satisfiability test for conjunctive normal forms is NP-complete.

Afterwards, or if max_spaces has not been called at all, alternatives is called recursively, also with one atom less in Free. Thus the induction hypothesis can be applied again, yielding:

$$\begin{aligned} \text{queries}(k,n) &\leq 1 + (n - 1 + \text{queries}(k-1,n)) + \text{queries}(k-1,n) \\ &\leq n + 2 \cdot (2^{k-1} - 1)n = (2^k - 1)n \end{aligned}$$

This concludes the proof, since we know that initially Free=AU, i.e. k=n. ■

The lower exponential bound for the complexity of hurrymax was obtained by the trivial argument that if the number of resulting maximal default spaces is exponential, it must take exponential time to output them. But this says nothing about the usual case, where there are only a few maximal default spaces. And also the number of queries does not (at least theoretically) have to be exponential just because the output is.[3] However, polynomial number of maximal default spaces can not be a sufficient condition to ensure polynomial behavior of hurrymax, since there still can be exponentially many maximally consistent default spaces, which are all considered by the algorithm.

A Simple Lower Bound

As the following proposition shows, even if the number of maximally consistent default spaces is known to be polynomial, no algorithm can ensure computing the maximal default spaces in polynomial time.

Let p_1, p_2 be polynomials, then let

$$\begin{aligned} &\mathit{PolyMaxcons}_{p_1,p_2} \overset{\text{def}}{=} \\ &\Big\{ \langle AU, \Gamma \rangle : \quad (1.) \text{ the number of maxcons subspaces of } AU \\ &\qquad\qquad\qquad\quad \text{is bounded by } p_1(|AU|), \\ &\qquad\qquad\qquad (2.)\ AU \text{ has exactly one } \Gamma\text{-maximal default space}, \\ &\qquad\qquad\qquad (3.)\ |\Gamma| \leq p_2(|AU|), \text{ and the lengths of formulae} \\ &\qquad\qquad\qquad\quad \text{are bounded by } p_2(|AU|) \Big\} \end{aligned}$$

Proposition 7 *Let* algo *be an algorithm, p a polynomial.*
Then there are polynomials p_1, p_2, such that for almost all $n \in \mathbf{N}$ there is an $\langle AU, \Gamma\rangle \in \mathit{PolyMaxcons}_{p_1,p_2}$ with $|AU| = n$ and

$$\text{QUERIES}_{\mathsf{algo}}(AU, \Gamma) > p(n)$$

[3] Actually, in special cases the algorithm presented in the following section is able to find an exponential number of default spaces using only a polynomial number of consistency checks.

Proof: Suppose the converse were true, i.e. there were an algorithm algo, a polynomial p such that for all polynomials p_1, p_2 there are infinitely many $n \in \mathbf{N}$ such that

$$\forall \langle AU, \Gamma\rangle \in \mathit{PolyMaxcons}_{p_1,p_2}, |AU| = n : \ \text{QUERIES}_{\mathsf{algo}}(AU,\Gamma) \leq p(n) \quad (*)$$

For deriving a contradiction it will suffice to take $p_1 = p$ and $p_2(n) = O(n)$.

Let n be a natural number, sufficiently large such that $\binom{n}{\lfloor n/2 \rfloor} \geq p(n) + 2$ and (*) holds. This is possible, since $\binom{n}{\lfloor n/2 \rfloor}$ grows exponentially and will thus exceed $p(n) + 2$ at some point. Beyond that point there still must be infinitely many n satisfying (*) from the assumption.

The basic idea of the proof is to show that there are two inputs with different maximal default spaces but which are indistinguishable by algo.

As the algorithm has no access to the logic structure, we can simulate algo on a 'semi-specified' input, where everything is left open except the weights, which are all given the same value w. Each consistency query will be answered in the following way:

$$\text{S is consistent?} \begin{array}{ll} \text{– yes,} & \text{if } |S| < n/2 \\ \text{– no,} & \text{if } |S| \geq n/2 \end{array}$$

After the algorithm has finished, we have two sets of queried default spaces, one with 'yes'-answers $\{\mathrm{Con}_1, \ldots, \mathrm{Con}_k\}$, and one of inconsistent default spaces $\{\mathrm{Inc}_1, \ldots, \mathrm{Inc}_l\}$. Since the large spaces are inconsistent, and the small ones consistent, one can easily imagine that there exists an input which produces exactly these answers to the queries. Whether this input also satisfies the polynomial restrictions of $\mathit{PolyMaxcons}_{p_1,p_2}$ is not obvious; in the following, a way of constructing such an input will be given.

Construction of Input Formulae.
Let us just take one single default $\delta = \top \rightsquigarrow_w \mathrm{c}$ and n object names:

$$\Delta_0 = \{\top \rightsquigarrow_w \mathrm{c}\} \quad \mathcal{O}_0 = \{\mathrm{o}_1, \ldots, \mathrm{o}_n\}$$

The basic idea of the construction is to have a concept name c_j for every consistent set Con_j such that in every model the object o_1 is interpreted as an instance of c_j iff only objects belonging to the default space Con_j are non-exceptions to the default δ (in that model). The choice of o_1 is arbitrary, any object would do — it is only important that the 'local' notion of exception to a default is made global.

For expressing these relationships by DL-formulae, we introduce further concept names $\mathrm{c}_{\mathrm{o}_i}$ for every object name o_i. They will mean "o_i is no exception to δ", or more precisely, that for every model M it will be true, that (the special object) o_1 is an instance of $\mathrm{c}_{\mathrm{o}_i}$ iff $\mathrm{o}_i \notin E_M(\delta)$.

How can the local (to o_i) condition $\mathrm{o}_i \notin E_M(\delta)$ be transferred to always the same object, i.e. transferred to the global condition $[\![\mathrm{o}_1]\!]^{\mathcal{I}} \in [\![\mathrm{c}_{\mathrm{o}_i}]\!]^{\mathcal{I}}$?

Only by a binary relation, a role. We introduce a role name r such that o_1 is a filler of r at every object o_i. This can be expressed by the TBox subsumption $\top \sqsubseteq r : o_1$.

All this taken together, we define the following set of strict knowledge:

$$\begin{array}{llcl}
\Gamma_0 := \{ & c_1 & \sqsubseteq & \sqcap_{\langle o_i,\delta\rangle \notin \mathrm{Con}_1} \neg c_{o_i} \\
& & \vdots & \\
& c_k & \sqsubseteq & \sqcap_{\langle o_i,\delta\rangle \notin \mathrm{Con}_k} \neg c_{o_i} \\
& \top & \sqsubseteq & r : o_1 \\
& o_1 & :: & (\neg c \sqcup \forall r : c_{o_1}) \sqcap (c_1 \sqcup \ldots \sqcup c_k) \\
& o_2 & :: & (\neg c \sqcup \forall r : c_{o_2}) \\
& & \vdots & \\
& o_n & :: & (\neg c \sqcup \forall r : c_{o_n}) \quad \}
\end{array}$$

The first k rules express that c_j is only valid if none of the concepts c_{o_i} (corresponding to objects o_i not occurring in Con_j) are valid. Of course, to obtain actual DL-formulae, they must be rewritten as (long) conjunctions. The next rule states that it is universally valid that o_1 is a filler of r, as mentioned above. The description of o_1 states, that one of the above concepts $c_1, \ldots, c_k$ must be valid for o_1. Furthermore, and this is stated also for all other objects o_i:[4] Concept c_{o_i} must be valid for the role fillers of r (i.e. at least o_1) if c is valid, i.e. o_i is no exception to $\top \rightsquigarrow_w c$. (This implication can only be written as a disjunction in our syntax.)

From this it can be seen, that the constructed input $\langle AU, \Gamma_0\rangle$, with $AU = \mathcal{O}_0 \times \Delta_0$, would lead to positive answers to consistency checks of exactly those default spaces that are subsets of $\mathrm{Con}_1, \ldots, \mathrm{Con}_k$.

However it is still possible, that this input is not in *PolyMaxcons* since it may have more than one maximal default space. (Namely if two sets Con_i and Con_j have the same cardinality.) And anyway, to get a contradiction in the proof, we need *two* different inputs in $\mathit{PolyMaxcons}_{p_1,p_2}$ leading to the same query answers, but having different (and unique) maximal default spaces.

We will see, that two sets of cardinality $\lfloor n/2 \rfloor$ can be taken to function as maximal default spaces by slightly supplementing the input construction above. They will be maximal, since all other consistent sets contain fewer atoms. (And the score only depends on the cardinality since the weights do not differ. Also, they can be chosen in accordance with the answers to the queries, i.e. none of the queried inconsistent default spaces will be contained in them, as we will now see:

[4]In fact, for o_1 the role is just used for notational uniformity, it would of course be sufficient to write $o_1 :: (\neg c \sqcup c_{o_1}) \sqcap (c_1 \sqcup \ldots \sqcup c_k)$

There are exactly $\binom{n}{\lfloor n/2 \rfloor}$ subsets of AU of cardinality $\lfloor \frac{n}{2} \rfloor$. Since there are at most l tested sets of cardinality $\geq \frac{n}{2}$ and since $l \leq p(n)$ and because n was chosen large enough such that $\binom{n}{\lfloor n/2 \rfloor} \geq p(n) + 2$, there must be two sets M_1, M_2 with $\frac{n}{2}$ atoms that have not been tested by the algorithm.

We can then construct for each of M_1, M_2 an input $\langle AU, \Gamma_i \rangle$ $(i = 1, 2)$ in the way shown above, satisfying the following conditions:

1. the set of maxcons default spaces is a subset of $\{\mathrm{Con}_1, \ldots, \mathrm{Con}_k, M_1, M_2\}$ and thus its size is bounded by polynomial p
2. both inputs have exactly one maximal default space, either M_1 or M_2
3. the number of formulae and formula length is linear in n (thus also polynomial in n)
4. for both inputs the algorithm algo uses exactly the same queries, obtaining the same answers (namely $\mathrm{Con}_1, \ldots, \mathrm{Con}_k$ and $\mathrm{Inc}_1, \ldots, \mathrm{Inc}_l$).

So, both inputs are in $\mathit{PolyMaxcons}_{p_1,p_2}$, and the algorithm should thus output their respective maximal default spaces. But as it received the same answers to the same queries for both of them and has no additional information except the weights (which are all equal), it cannot possibly do so. ■

A Class of Easy Inputs

Taking up the idea of *conflicts* between defaults from [Quantz, Royer 92], we can obtain a quite general characterization of a tractable class of inputs.

Definition 15 *A default space S is called* **clash** *or* **minimal conflict** *iff*
1. S is inconsistent and
2. all subsets of S are consistent.

It is often known from the modeling, that the number of clashing atoms is not higher than a small number k (sometimes even $k = 2$). Then only a polynomial number of consistency checks has to be performed to find the maximal default spaces, even if there are exponentially many.

Proposition 8 *Let $k \in \mathbf{N}$. If all clashes in AU have at most k members, then there exists an algorithm computing the maximal default spaces of AU with less than $O(|AU|^k)$ queries.*

If additionally
1. the cardinality of the set

$$\mathit{Common} \;=\; \{\langle o, \delta \rangle : \langle o, \delta \rangle \text{ is a member of 2 different clashes}\}$$

is bounded by $|\text{Common}| = O(\log(|AU|))$*,*
2. if the number of maximal default spaces is polynomially bounded, and
3. the consistency queries are answered in polynomial time,
then the algorithm finishes in time polynomial in $|AU|$*.*

Proof: The number of possible clashes is bound by the number of subsets of AU containing not more than k atoms. Checking inconsistencies costs at most $k+1$ queries, so the maximum number of queries needed is bound by

$$\begin{aligned} \text{QUERIES}(AU, \Gamma) &\leq (k+1)\sum_{i\leq k}\binom{|AU|}{i} \\ &\leq (k+1)|AU|^k = O(|AU|^k) \end{aligned}$$

To know all clashes is sufficient for building all maximal default spaces, for example by enumerating all possible default spaces and eliminating the ones containing a clash.

The second part of the proof relys on the fact that this exponential enumeration procedure can be simplified, if the set of atoms common to two or more clashes is small. This is because instead of generating all subsets of AU, it is sufficient to generate all subsets of *Common* (using $O(2^{|Common|})$ operations), and then add the remaining atoms such that the score of this default space is maximized, but ensuring that no clash becomes a subset. Since all these remaining atoms are contained in none or only a single clash, this maximization can be done in linear time ($O(|AU|)$).

The total number of operations needed for that is

$$\begin{aligned} & O(2^{|Common|} \cdot |AU|) \\ = \; & O(2^{O(\log|AU|)} \cdot |AU|) \\ = \; & O(|AU|^{O(1)}) \end{aligned}$$

(Since $O(1)$ denotes constants, $O(n^{O(1)})$ is used for polynomials in n.)

From the first part we know, that the clashes can be found with a polynomial number of queries, together with the premises this gives us

$$\text{TIME}(AU, \Gamma) = O(|AU|^{O(1)})$$

■

5.2 A Polynomial Algorithm

As became clear from the proof of Proposition 8, the algorithm has two stages: First finding all clashes and then using this information to build the maximal default spaces. For the second stage the logical content of the input does not need to be accessed anymore, only the weights are used. Although the (normally long) answer time of DL queries plays no role in this part, its complexity can be exponential if too many clashes are interrelated.

I will now sketch a possible algorithm, calling it smallclash. It can still be optimized in many respects.

Finding the Clashes

The clashes can be computed in layers of increasing cardinality. This means, first to test all atoms whether they are consistent with the KB, then to test all unordered pairs of the consistent atoms, then all three-element sets where all the two-element subsets are consistent, and so on until cardinality k is reached.

This method has the lowest worst-case complexity, but is only efficient if the number of clashes is high. In practical cases, the opposite, i.e. large consistent sets, is to be expected. So various optimization heuristics should be considered here.

Assembling the Default Spaces

After all clashes have been found, the first step is to group them into clusters. A cluster is a minimal set of clashes such that all its members are disjoint with every clash not in the set. (In graph-theoretical terms, when the clashes are regarded as nodes being adjacent iff they have a common atom, the clusters are the maximal connected subgraphs.)

Splitting up the set of clashes can significantly reduce complexity (if it results in more than one cluster), because the following algorithm is exponential.

Now each cluster can be treated separately, resulting in a set of maximal subspaces of all atoms of the cluster. All of these are to be combined at the end.

Let us consider any cluster. First the set Common of atoms contained in more than one of its clashes is computed. (It will be empty iff the cluster contains just one clash.) Then the maximal default subspaces are searched by

1. enumerating all subsets of Common
2. extending each subset to a set with maximal score but containing no clash completely

The predicate check_all generates subsets of Common in the same way as the simple algorithm of Chapter 4, and performs the maximization each time a new subset (candidate for starting a maximal default space) is generated.

```
check_all([Atom|Free], Subset, Cluster, Common, OldCluMax, CluMax):–
    check_all(Free, Subset, Cluster, Common, OldCluMax, NewCluMax),
    check_all(Free, [Atom|Subset], Cluster, Common, NewCluMax, CluMax).
check_all([], Candidate, Cluster, Common, OldCluMax, CluMax):–
    maximize_candidate(Cluster, Candidate, Common, NewMax), !,
    append_to_all([Candidate], NewMax, NewCluMax),
    update_max(NewCluMax, OldCluMax, Max).
check_all([], _ , _ , _ , OldCluMax, OldCluMax).
```

The maximize_candidate predicate determines the maximal supersets of Candidate by maximizing each clash of the Cluster separately. It fails iff a call to maximize_clash fails, that is, iff the Candidate already contains a whole clash.

```
maximize_candidate([], _ , _ , []).
maximize_candidate([Clash, Clashes], Candidate, Common, Max):–
    maximize_candidate(Clashes, Candidate, Common, OldMax),
    maximize_clash(Clash, Candidate, Common, Results),
    append_to_all(Results, OldMax, Max).
```

The actual work is done by the maximize_clash predicate. First the set Free of atoms contained in no other but the examined Clash is determined. Whether the remaining (Bound) atoms of Clash are in the current default space is already controlled by the enumeration of the candidates; so if all atoms of Bound are contained in the current Candidate, an atom of Free must be omitted in the resulting default space. If not all Bound atoms are in Candidate, the conflict is already resolved, so all Free atoms can be safely returned for adding to the default space.

```
maximize_clash(Clash, Candidate, Common, Results):–
    Free = Clash $\setminus$ Common,
    Bound = Common $\cap$ Clash,
    ((Bound $\subseteq$ Candidate) $\rightarrow$
        delete_min_atom(Free, Results)
    ; Results = [Free]).
```

The predicate delete_min_atom looks for all atoms of Free with minimal score, and builds a list of subsets of Free with each of them deleted. Iff Free is empty the predicate will fail.

5.3 Heuristic Ideas

Compared to other methods of default reasoning, the complexity of PDDL is considerably higher, stemming from 2 major factors:

- defaults are treated as material implications, that is why complex reasoning with disjunctions and negations becomes necessary.
- even though usually only a few default spaces are maximal, in most cases there are many maximally consistent default spaces, so, a high number of consistency checks must be performed to find the maximal ones.

A third point concerns a problem facing most nonmonotonic logics: multiple extensions. Here this means several maximal default spaces. As mentioned in Chapter 3, alternative maximal default spaces can in some cases be merged into a single set of DL-formulae. This does not necessarily improve deduction time — if many disjunctions occur in this set (from many neutralizations), it may be preferable to do separate deduction in every maximal default space.

The algorithm of the preceding section only works for a restricted class of inputs. Another way to speed up computation is to design algorithms that can be used on all inputs, but do not (generally) produce the desired output, but only a sufficiently close approximation.

To do this, one must first have a notion of approximation, or similarity of solutions. Especially in logic, this can be a tricky question — for example adding only a single formula can enormously change set of consequences, or even lead to inconsistency. Small difference in the score does not necessarily mean that the default spaces have most atoms in common, since weights of completely different atoms may add up to the same value. Such an *unstable* behavior is not, however, frequently expected in practice, this should be mirrored by the modelings. In Natural Language Processing, this would correspond to a sentence (or even a paragraph) having two totally different readings, but such 'large ambiguities' do not occur often.

When the semantics is relaxed, new issues open up. Sometimes ambiguities are genuine, perhaps two readings are intended. But the corresponding (maximal) default spaces may not have exactly the same score. Thus, one interpretation is lost when only the strictly maximal default space is taken. Instead, a limit could be set for the score difference between maximal default spaces and additionally admitted *quasi-maximal* default spaces.

It may well be, that allowing approximate solutions can lead to both quicker heuristic algorithms and a more natural semantics, at least for NLP applications.

Using Defaults as Chaining Rules

By also accepting quasi-maximal default spaces, only the second source of complexity, the number of sets to be tested, can be reduced. In addition to that, the disjunctive reasoning can be avoided by treating defaults as forward-chaining trigger rules rather than as material implications.

This changes the semantics, obviously. It becomes more like Reiter's Default Logic, or Moore's AEL. Either approaches have already been tried to utilize for DL(cf. [Baader, Hollunder 92] and [Donini et al. 92]). I will now say a bit more about how the efficient deduction process of Reiter's logic could be applied without losing too much of the characteristics of the weighted defaults.

There, a set of 'extensions' of the strict knowledge is built incrementally, starting with the strict knowledge itself, and adding the conclusion of every default, if the premise but not the negated conclusion is already derivable from the current extension. Actually, these are not extensions in the sense of Chapter 2, only their generating sets.

Since in DL a concept is not a formula itself, we adapt this procedure by not only running through defaults $c_p \rightsquigarrow_w c_c$ but also through all objects o_i and checking whether $o_i :: c_p$ (and not $o_i :: \neg c_c$) is derivable. If this is the case, $o_i :: c_c$ is added to the extension. If $o_i :: c_p$ follows, but also $o_i :: \neg c_c$, then o_i is an exception to the default $c_p \rightsquigarrow_w c_c$, so its weight is recorded.

After no default can be applied anymore, the sums of the recorded weights of each extension represent negative scores. They correspond to the negative scores of models, so the extension(s) scoring minimal are the preferred ones.

Then we can deduce DL-descriptions, if we can derive them from all the preferred extensions, just as we did with maximal default spaces.

This procedure can still be improved, as we shall see in the following example. Let

$$\begin{aligned}\Delta &= \{ \; c_1 \rightsquigarrow_{100} c_2, \\ & \qquad c_2 \rightsquigarrow_{300} c_3, \\ & \qquad c_1 \rightsquigarrow_{5} \neg c_3 \; \} \\ \text{and } \Gamma &= \{ \; o :: c_1 \; \}\end{aligned}$$

In the first step, the initial extension $E_0 = \{o :: c_1\}$ would be enlarged by $o :: c_2$ and $o :: \neg c_3$ (since both first and last default are applicable), yielding a single extension E_1 where no more defaults are applicable. Though it has a negative score of 300, it would be the preferred one for lack of alternatives.

But rather we would like to see the low-scoring third default being defeated. This can be achieved by suspending the application of a default as long as higher-weighted defaults can be applied.

Still, the resulting procedure behaves very different from the original PDDL semantics. Only if neither disjunctive default application nor contraposition play a role in the application, both approaches yield the same results.

Contraposition could, of course, straightforwardly be incorporated into the chaining system, simply by also doing the chaining backward, i.e. by adding the negated premise $\mathrm{o}_i :: \neg\mathrm{c}_p$ if the negated conclusion $\mathrm{o}_i :: \neg\mathrm{c}_c$ but not the premise $\mathrm{o}_i :: \mathrm{c}_p$ is derivable from the current extension. However, the negated concepts would increase complexity of the inference task again, since negated conjunctions (assumed to occur frequently) result in disjunctions, which raise complexity.

How to incorporate the disjunctive PDDL features into the chaining approach is not apparent, to say the least. So, sacrificing logical properties seems inevitable in order to reduce complexity.

Applying defaults as trigger rules does not always improve complexity. Obviously, if all defaults have the form $\top \rightsquigarrow \mathrm{c}_c$, both semantics are equivalent, in this case all objects are instances of each premise. As already remarked in Chapter 3, every weighted default $\mathrm{c}_p \rightsquigarrow_w \mathrm{c}_c$ can be replaced by $\top \rightsquigarrow_w \neg\mathrm{c}_p \sqcup \mathrm{c}_c$ without changing what is entailed. So if the modeling is all done that way, nothing will be gained by default chaining. But it is to be expected, that in a real domain modeling the first, intuitive notation is used. Actually, this can be seen as an advantage, as it gives the designer of the representation the choice over the way each default is to be handled.

Precomputing Default Interactions

The following proposals are again motivated by the natural language scenario, but probably are also of general interest.

Since the set of defaults remains fixed during the analysis of the various input sentences, it will probably be worthwhile to find out, once and for all, the possible relations between them, mainly which sets of defaults usually clash, or which are completely independent from each other.

In languages where there are cases, such as in German, every noun occurring in a sentence must be of exactly one case. There can be rules for cases on all levels, from morphology to semantics. Defaults expressing these rules are prone to clash. Here, the tendency to clash can easily be recognized from the logical structure: the conclusions of these defaults (like 'case x') are already inconsistent with one another.

Splitting Up the Set of Defaults

There are two main cases where the defaults can be split up into disjoint subsets which can be treated completely separately. That means, maximal default spaces can be computed independently, and then by choosing one maximal default space for every subset and taking their union, a maximal default space for AU can be obtained. All such combinations yield all maximal default spaces.

The first case is when the domain has itself areas, which are completely independent from each other.

The second case emerges, when there are large weight differences between sets of defaults. It is possible that the set of defaults has a layered structure, $\Delta = \Delta_1 \cup \ldots \cup \Delta_l$, such that the weights of atoms containing only defaults from the lower layers $\Delta_1, \ldots, \Delta_i$ cannot possibly add up to a value higher than the smallest weight of a default from Δ_{i+1}. If this is the case, the defaults of a high layer behave like strict knowledge to a lower layer default, they can be called *constraint-like*.

It should be pointed out that if two defaults δ_1, δ_2 can form a non-local conflict, that is, if there exist two atoms $\langle o_1, \delta_1 \rangle$, $\langle o_2, \delta_2 \rangle$ contained in the same clash, but with differing objects $o_1 \neq o_2$, then no weight difference is large enough to put these defaults into different layers, since this may be compensated by a sufficiently large number of objects being possible exceptions to the smaller-weighted default but not to the other.

5.4 Putting it together

From what has been said so far, its semantics may seem too difficult to implement the weighted PDDL in a practicable system. But at the moment this cannot really be decided by purely theoretical considerations, the actual tasks to be represented may have a sufficiently simple structure that even an incomplete algorithm always works correctly or a complete algorithm works fast enough.

Though restricting the clash size to a constant probably means demanding too much of practical applications, large minimally inconsistent sets are expected to be very infrequent. If larger clashes exist, the smallclash algorithm would not find all clashes and thus regard more default spaces as maximally consistent than actually are, this might lead to the output of an inconsistent default space.

But it is important to note, that if the output of smallclash only consists of consistent default spaces, then these are also the maximal default spaces, regardless whether the input clash size is bounded.

It is no problem to check consistency of each output set (unless their number is large). Then, if an inconsistency is detected, the algorithm can proceed in two ways: it can either call a complete, but slow algorithm (hurrymax for example), or, to save time, backtrack into the construction of the maximal default spaces to obtain a quasi-maximal and consistent default space.

Since most atoms are usually consistent with most others, much time can be saved by the smallclash algorithm, when a large consistent set is known — it cannot contain any clashes.

Consequently, a realistic procedure should begin by calling an algorithm like hurrymax, but only until it has found one or two maximally consistent default spaces. Then smallclash can be called with this information.

In this procedure it becomes even more important for the hurrymax algorithm, that good candidates for maximal default spaces are found early (among the many maxcons sets).

Obviously, this depends on the ordering of the atoms. A way of ordering would be to include those atoms first, that can be directly applied in the manner of chaining rules, described in the previous section. Instead of making it a separate algorithm, the approach of adding conclusions (or negated premises) could be used as a heuristic to start maximally consistent default spaces — an atom would be considered being in the space, if it was applied in the extension, and out, if it was defeated. The remaining atoms would then tried to be added in the usual manner. In this way, a large part of the search space can be explored, before a complete algorithm is called to do the rest. Of course, the other optimization techniques mentioned would be applied before starting the process above.

Due to their enormous complexity, the performance of deduction systems depends on a variety of factors. Some issues could be clarified here using theoretical methods, but still only covering a fraction of the relevant aspects. Whether the proposed ideas really work cannot be judged before they are put into practice.

Chapter 6

Conclusions

In this paper Preferential Default Description Logics based on weighted defaults were introduced, as a way of extending Description Logics by default reasoning on the basis of preferential models. Since they strictly obey the principle of exception minimization, they are suited for modeling knowledge in Natural Language Analysis.

The task was to develop a proof theory for this class of logics. This could be achieved by the use of default spaces, corresponding to subtheories of the 'application' of all defaults to all objects. The benefits of this proof theory are almost exclusively of theoretical nature. The question of decidability is answered in the most favorable way, namely that the PDDL is decidable, if and only if the underlying DL is decidable.

On the practical side, tractability is a big problem — in fact, no sound and complete algorithm can achieve a subexponential worst-case complexity. If the nice semantics is intended to have any practical value, heuristics must be developed, based on the specific applications. At the time of writing the paper, a real modeling using the Weighted PDDL was not yet available. It may be expected that an implementation of the logic is needed before such a modeling can be completed, so consequently, both aspects can only be pursued in parallel.

In this light, the proposed algorithm can be seen as a starting point for small experimental modelings.

A subclass of modelings, characterized by small clash size, was proved to be of polynomial complexity, provided that the DL is tractable. However, it must be kept in mind, that if disjunction is added to the DL-syntax, tractability is usually lost.

For inputs outside of this class, it may even be best to give up some of the semantic features of the logic. It was suggested to interpret defaults as chaining rules rather than as material implications, but substantial research is still needed here.

A very promising idea has been put forward by Pinkas [Pinkas 91]. He introduces so-called *penalty logic*, which is similar to WPDDL but has real-numbered weights. Then he shows a way to transform a set of such logic formulae into a symmetric neural network, the energy function corresponding to our score-function. This allows to use classical neural network models, like Hopfield Nets or Boltzman Machines, as a 'connectionist inference engine'.

If one aims at retaining the full semantics, the complexity of this method remains exponential, of course. However, the advantage of the approach is the ability to trade correctness of the answer with computation time. Another feature of neural nets could also prove to be valuable: Tuning the weights is a non-trivial task for the application modeler. In a network they could (at least partly) be learned.

Appendix A

Experimental Implementation

A.1 FLEX Syntax

The following is by no means intended as a formal description or a manual for the system, just as an informal introduction.

FLEX is a DL-system based on BACK with flexible inference strategies and a capability to handle alternative ABoxes or *situations*. The used version was a research implementation in Prolog. FLEX contains all the syntactic features of the simple DL introduced in Chapter 2, using a slightly different notation:

`anything`	for	$\top$
`nothing`	for	$\bot$
c_1 `and` c_2	for	$c_1 \sqcap c_2$
c_1 `or` c_2	for	$c_1 \sqcup c_2$
`not`(c)	for	$\neg c$
`all`(r, c)	for	$\forall r : c$
`exists`(r, c)	for	$\exists r : c$
`rtop`	for	the universal role

Entering a formula into the knowledge base is called a *tell*. Two kinds of tells can be distinguished, one for terminology and one for descriptions. Any new concept name or role name must first be introduced on the left-hand side of a TBox tell. Furthermore, a second attempt to use this name on the left-hand side of a tell is regarded as double definition and rejected by the system. The only TBox tells needed here are rules:

c_n `:<`c for $c_n \sqsubseteq c$

An ABox description must always be told in some situation, the syntax is `o::c in` *sit*

Note that a tell will fail, if the description is inconsistent with the knowledge already stored in *sit*.

Descriptions in different situations are treated independently, they only have the terminology in common. However, there is a mechanism to link situations:

`extend_sit`(sit_1, sit_2)

expresses that every descriptions told in situation sit_1 is also valid in sit_2. If sit_2 is a free variable, FLEX binds it to a newly generated situation name. In all other cases situations must be instantiated to atomic names.

For knowledge retrieval there are several kinds of queries, we will only need ABox queries, which have the form:

`o ?: c in` *sit*

All three variables must be instantiated, then the query succeeds iff o :: c is derivable in *sit*.[1]

The integration of defaults into the FLEX system is intended, the following syntax has been proposed for weighted defaults

$prem \ \sim w \sim > \ conc$ for $prem \rightsquigarrow_w conc$

Two Prolog operators ($\sim$ and $\sim>$) were introduced to handle this format.

A.2 Main Predicates

The following program basically consists of the three predicates descriped in Chapter 4 (hurrymax-algorithm). In addition, the defaults here are ordered according to their weights, to give the first maximally consistent default space a good chance of being already a maximal one. As another means of optimization, all atoms are first tested whether their content contradicts the strict knowledge or whether they would add no new information to the knowledge base; in both cases they are not considered as 'applicable', they can be omitted in the search.

The main predicate `maximal_DS` is called with a FLEX representation of the strict knowledge, a list of defaults and a list of objects, after constructing the initial space-structure from the applicable atoms, the main predicate of the hurrymax-algorithm is called.

Differing slightly from Chapter 4, max-structures are just represented as a term formed by a slash

$MaxList/Score$

$MaxList$ being a list of lists of atoms, and $Score$ the (common) score of these lists.

[1]This is not actually true since FLEX inference algorithms are not complete, but can be assumed for our purposes.

The space-structures are here represented as

$$\mathsf{space}(sit, In/\mathrm{score}(In), Out/RestScore),$$

where $RestScore = \mathrm{score}(\mathsf{AU} \setminus Out)$

```
%%%  File:         ~svs/back/maxspace.pl
%%%  Date:         8/June/94
%%%  Author:       Sven Suska
%%%  System:       Quintus Prolog v3.1

:-module(maxspace,[maximal_DS/4]).

:- use_module(my_inout,
         [write_space/1,
         flex_tell_atom/2,
         flex_ask_atom/2,
         flex_ask_atom_false/2,
         flex_extend_sit/2]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% main predicate.
%% first sorts the given defaults,
%% then from all pairs removes the obvious, easy atoms
%% before searching the max default spaces

maximal_DS( FlexSit, Objects, Defaults, MaxDSs):-
         sort_by_weight(Defaults,SoDefaults),
         applicable(FlexSit, Objects, SoDefaults, AppAtoms, InAtoms, OutAtoms),
         sum_weights(AppAtoms, AppSum),
         sum_weights(InAtoms, InSum),
         RestSum is AppSum + InSum,
         StartSpace = space( FlexSit, InAtoms/InSum, OutAtoms/RestSum),
         max_spaces( AppAtoms, StartSpace, []/0, MaxDSs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%                     hurrymax                 %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% extend a given OldSpace consistently until no atom remains
%% or too many have been excluded that MaxScore cannot be reached

next_space( _, MaxScore, OldSpace, OldSpace, incomplete):-
         score_exceeded(OldSpace,MaxScore),!.
next_space( [],_ , Space, Space, complete).
next_space( [Atom|Atoms], MaxScore, OldSpace, NewSpace, Completed):-
         ( add_atom( Atom, OldSpace, Space),!
         ; exclude_atom( Atom, OldSpace, Space)
         ),
         next_space( Atoms, MaxScore, Space, NewSpace, Completed).
```

```
%% search all default spaces obtainable by adding
%% atoms of FreeAtoms to FixSpace for maximal spaces

max_spaces( FreeAtoms, FixSpace, OldMax, ResultMax):-
         get_score( OldMax, NegScore),
         next_space( FreeAtoms, NegScore, FixSpace, NewSpace, Completed),
         ( Completed = complete ->
             update_max( NewSpace, OldMax, NewMax)
         ;
         NewMax=OldMax),
         get_new_rejected( NewSpace, FixSpace, Ex),
         alternatives( Ex, FreeAtoms, FixSpace, NewMax, ResultMax).

%% same as max_spaces, only requiring one of the
%% Rejected atoms to be in the max.ds. to be found

alternatives( [], _, _, Max, Max).
alternatives( _, _, FixSpace, Max, Max):-
         get_score( Max, MaxScore),
         score_exceeded( FixSpace, MaxScore),!.
alternatives( [Atom|RejRest], FreeAtoms, FixSpace, OldMax, ResultMax):-
         delete( Atom, FreeAtoms, RestFree),
         ( add_atom( Atom, FixSpace, PlusSpace) ->
             max_spaces( RestFree, PlusSpace, OldMax, NewMax)
         ; NewMax = OldMax),

         exclude_atom( Atom, FixSpace, MinusSpace),
         alternatives( RejRest, RestFree, MinusSpace, NewMax, ResultMax).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Auxiliary Predicates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% include an atom in a space structure, if it can
%% be successfully added to the FLEX ABox of this space structure

add_atom( Atom, space(OldSit, OldIn/OldInSc, Out),
                                  space(NewSit, NewIn/NewInSc, Out) ):-
         flex_extend_sit(OldSit,NewSit),
         atom( _, default( _,_, Weight) ) = Atom,
         flex_tell_atom( Atom, NewSit ),
         !,
         NewInSc is OldInSc + Weight,
         append( OldIn, [Atom], NewIn).

%% add an atom to the out-list of a space structure
exclude_atom( Atom, space(Sit, In, OldOut/OldRestSc),
                                  space(Sit, In, NewOut/NewRestSc) ):-
         atom( _, default( _,_, Weight) ) = Atom,
         NewRestSc is OldRestSc - Weight,
         append( OldOut, [Atom], NewOut).


%%  update_max( Space, OldMax, NewMax)
%%  update OldMax if score of Space is higher or equal
%%  `InScore' and `RestScore' must be equal (space is complete)

update_max(        space(_, _/Score, _/Score),
                   MaxList/MaxScore ,
                   MaxList/MaxScore):-MaxScore > Score,!.

update_max(        space(_, In/Score, _/Score),
                   OldMaxList/Score,
                   [In|OldMaxList]/Score):-     !.

update_max(        space(_, In/Score, _/Score),
                   _/OldMaxScore,
                   [In]/Score):-      OldMaxScore < Score.


%%  elementary operdations, library(sets) is used

get_score( _/Score, Score ).

score_exceeded( space(_,_, _/RestScore), MaxScore ):-
         RestScore < MaxScore.

get_new_rejected( space(_, _, NewOut/_), space(_, _, OldOut/_), Rejected):-
         sets:subtract( NewOut, OldOut, Rejected).

delete(Element, Set, Rest):-
         sets:select(Element, Set, Rest).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% testing each pair of objects and defaults for
%% being derivable from knowledge of Sit         (in)
%%         contradictory to knowledge of Sit     (out)
%%        or else applicable (app)

applicable(_, _, [], [], [],[]).
applicable(Sit, Objs, [Df1|Defaults], AppAtoms, InAtoms, OutAtoms):-
         applicable(Sit, Objs, Defaults, AppOld, InOld, OutOld ),
         applicable_default(Sit, Objs, Df1, NewApp, NewYes, NewContra),
         append(AppOld, NewApp,  AppAtoms),
         append(InOld,  NewYes,  InAtoms),
         append(OutOld, NewContra,  OutAtoms).
```

```
applicable_default(_, [], _, [], [],[]).
applicable_default(Sit, [Obj1|Objs], Df, App, In, [Atom|Out]):-
         Atom=atom(Obj1, Df),
         flex_ask_atom_false( Atom, Sit),
         !,
         applicable_default(Sit, Objs, Df, App, In, Out).
applicable_default(Sit, [Obj1|Objs], Df, App, [Atom|In], Out):-
         Atom=atom(Obj1, Df),
         flex_ask_atom( Atom, Sit),
         !,
         applicable_default(Sit, Objs, Df, App, In, Out).
applicable_default(Sit, [Obj1|Objs], Df, [Atom|App], In, Out):-
         Atom=atom(Obj1, Df),
         applicable_default(Sit, Objs, Df, App, In, Out).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  straightforfart summing and sorting,
%%  using built-in predicate `keysort'

sort_by_weight(Unsorted,Sorted):-
         add_key(Unsorted, KeyUn),
         keysort(KeyUn, KeySor),
         add_key(Sorted, KeySor).

add_key([], []).
add_key([default(P,C,W)|Defaults], [W-default(P,C,W) | KeyList]):-
         add_key( Defaults, KeyList).


sum_weights( [], 0).
sum_weights([Atom|DSRest], Result):-
         atom(_, default(_,_, Weight) ) = Atom,
         sum_weights( DSRest, OldScore ),
         Result is OldScore +  Weight.
```

A.3 Interface Predicates

The following file contains predicates for three tasks

1. predicates to read and store input knowledge from a file
2. interface predicates for FLEX tells and queries on the level of atoms
3. output procedures for writing default spaces

The program is started by the goal

run($FileName$)

where the file specified by $FileName$ must contain a sequence of FLEX tells and defaults.

```
%%%  File: ~/svs/back/my_inout.pl
%%%  Date: 10/June/94
%%%  Author: Sven Suska
%%%  System:       Quintus Prolog v3.1, FLEX v0.1.18

:- module(my_inout,
         [
         go/0,    run/1,
         flex_ask_atom/2,
         flex_ask_atom_false/2,
         flex_tell_atom/2,
         flex_extend_sit/2,
         write_space/1]).

:- use_module(flex, '/home/kit/kitvm11/flexi/flex.pl', all).

:- style_check(all).
:- use_module('maxspace.pl',[maximal_DS/4]).

%% syntax of defaults:     Prem  ~W~>  Con
:- op( 700, xfx , ~>).
:- op( 701, xfx, ~).

go:-
         run('~svs/back/example.flx').

%% read, compute, output:
run(FileName):-
         backinit,
         read_kb(FileName,Defaults,Objects),
         format("Objects: ~w~n",[Objects]),
         maximal_DS(sit, Objects, Defaults, M/Score),
         format("~nMaxScore=~d ~nResult:~n",Score),
         write_ds(M).

read_term(T)           :-
         read(T),
         ( T=end_of_file->
             format('Finished Reading Data~n',[])
         ;true).

%% if execution was terminated during read, a file may be left hanging
:-dynamic hanging_file/1.
```

```
read_kb(FN,Defaults,Objects):-
          abolish(data_df,1),
          abolish(obj_intro,1),
          (hanging_file(FN1) ->
              write(closing_hanging_file(FN1)),nl,
              see(FN1),seen,
              retract(hanging_file(FN1))
          ; true),
          see(FN),
          assert(hanging_file(FN)),
          readkb,
          seen,
          retract(hanging_file(FN)),
          collect_dfs(Defaults),
          collect_objs(Objects),
                    %% store in database to enable
                    %% formatted output at any time:
          abolish(mem_defaults,1),
          assert(mem_defaults(Defaults)),
          abolish(mem_objects,1),
          assert(mem_objects(Objects)).

readkb:-
          repeat,
            read_term(T),
            store_data(T),
            T=end_of_file,
          !.

%% if it is a default, it is stored and printed
%% if not, it is told to FLEX, printed automatically
%% all described objects are remembered
store_data(end_of_file):-!.
store_data(Pre~Weight~>Con):-
          !,
          assert(data_df(default(Pre,Con,Weight))),
          write_default(default(Pre,Con,Weight)),nl.
store_data(X):-
          backtell(X).
store_data(Role :< rtop):-
          assert_active_role(1,Role).  % enables concept propagation
store_data( Obj :: _ ):-
          assert(ob_intro(Obj)).


%% make a list out of the asserted objects/defaults
collect_dfs([Df|DefaultList]):-
          retract(data_df(Df)),
          collect_dfs(DefaultList).
collect_dfs([]).
```

```
collect_objs([Obj|Objects]):-
          ob_intro(Obj),!,
          retractall(ob_intro(Obj)),
          collect_objs(Objects).
collect_objs([]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  adapting FLEX access for atoms
%%

flex_ask_atom(atom(Obj,Default), Sit):-
          default(Pre,Con,_) = Default,
          Obj ?: not(Pre) or Con in Sit .

flex_ask_atom_false(atom(Obj,Default), Sit):-
          default(Pre,Con,_) = Default,
          Obj ?: Pre and not(Con) in Sit .

flex_tell_atom(atom(Obj, Default), Sit):-
          default(Pre,Con,_) = Default,
          Tell = (Obj :: not(Pre) or Con in Sit),
          Tell.

flex_extend_sit(Sit1,Sit2):-
          extend_sit(Sit1, Sit2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%   Output Procedures
%%

write_default(default( Pre, Con, Weight )):-
          format('~t~w~t~17+~~~w~~>~t~w~t ~22+ ', [Pre, Weight, Con]).

%% write a space-structure (useful for testing)
write_space(space(FlexSit,In/UpSc,Out/LowSc)):-
          format('situation: ~w, Up=~w, Low=~w ~n',[FlexSit,UpSc,LowSc]),
          write_ds([In]),
          write_ds([Out]).

%% the following predicates show default spaces in table form
%% they use the information in mem_defaults/2 and mem_objects/1
write_ds([]).
write_ds([Atoms|DS]):-
          mem_defaults(Defaults),
          format("~`*t  Default Space: ~`*t ~70|~n", []),
          out_ds_lines(Defaults,Atoms, Remaining),
          (  \+ Remaining = []  ->
              format("remaining ~w ~n", atoms(Remaining))
```

```
          ; nl
          ),
          write_ds(DS).

out_ds_lines([], Remainder, Remainder).
out_ds_lines([Df| DfRest], Atoms, ResultingRemainder):-
          out_ds_lines( DfRest, Atoms, Remainder),
          write_default(Df),format("~t ~44|:      ",[]),
          out_obj_line(Df, Remainder, ResultingRemainder).


out_obj_line(Df, Atoms, Remain):-
          mem_objects(Objects),
          out_objs(Df, Objects, Atoms, Remain),
          nl.

out_check_atoms(_, _, [], Remainder, Remainder, ' ').
out_check_atoms(Df, Obj, [atom(Obj,Df)| AtomRest], OldRemain, NewRemain, Obj):-
          !,
          out_check_atoms(Df, Obj, AtomRest, OldRemain, NewRemain, Res),
          ( \+ Res=' ' -> write('internal error: atom twice in list'), nl
    ;      true).
out_check_atoms(Df, Obj, [ Atom | AtomRest ], OldRemain, NewRemain, Res):-
          out_check_atoms(Df, Obj, AtomRest, [Atom|OldRemain], NewRemain, Res).

out_objs(_, [], Remain, Remain).
out_objs(Df, [Obj|ObjRest], Atoms, Remainder):-
          out_check_atoms(Df, Obj, Atoms, [], RemainAtoms, Result),
          format("~t~w ~t~5+",Result),
          out_objs(Df, ObjRest, RemainAtoms, Remainder).
```

A.4 Test Data

The following example has 6 atoms, but since the last default is not applicable at `o1`, only 5 of them are considered by the algorithm. They correspond exactly to the example of Figure 4.1.

```
% TBox
         c1 :< anything.
         c2 :< not(c1).
         c3 :< anything.
         c5 :< anything.
         r :< rtop.
         c4 :< not(c3) or all(r,(c5)).
% c4 cannot be defined earlier,
% because terminology must have a unique definition

% ABox
         o1 ::  c1 in sit.
         o2 ::  ( c2 and r:o1 ) in sit.

% Defaults
         c5 or not(c1)       ~4~>      c2 and c4.
         anything  ~2~>      c3 and c5.
         c2                  ~1~>      c3 and not(c5).
```

The following default spaces are found:

```
MaxScore=9
Result:
**************************  Default Space: *************************
 c5 or (not c1)  ~4~>    c2 and c4          :           o2
    anything     ~2~>    c3 and c5          :       o1   o2
       c2        ~1~> c3 and (not c5)       :       o1

*************************  Default Space: *************************
 c5 or (not c1)  ~4~>    c2 and c4          :       o1   o2
    anything     ~2~>    c3 and c5          :
       c2        ~1~> c3 and (not c5)       :       o1


yes
| ?-
```

Bibliography

[Baader, Hollunder 92] F. Baader, B. Hollunder, "Embedding Defaults into Terminological Knowledge Representation Formalisms", in [KR 92], 306–317.

[Barwise 89] J. Barwise, *The Situation in Logic*, Stanford: CLSI Lecture Notes 17, 1989.

[Brachman 79] R.J. Brachman, "On the Epistemological Status of Semantic Networks", in [Findler 79], 3–50.

[Brachman, Schmolze 85] R.J. Brachman, J.G. Schmolze, "An Overview of the KL-ONE Knowledge Representation System" *Cognitive Science* **9**, 171–216, 1985.

[Brewka 91] G. Brewka, *Nonmonotonic Reasoning: Logical Foundations of Commonsense* Cambridge: Cambridge University Press, 1991.

[Donini et al. 91] F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt, "Tractable Concept Languages" *IJCAI-91*, 458–463.

[Donini et al. 92] F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, W. Nutt, "Adding Epistemic Operators to Concept Languages", in [KR 92], 342–353.

[Findler 79] N.V. Findler (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, New York: Academic Press, 1979.

[Fischer 92] M. Fischer, *The Integration of Temporal Operators into a Terminological Representation System*, KIT-Report 99, Technische Universität Berlin, 1992.

[Gabbay et al. 93] D. Gabbay, C. Hogger, J. Robinson (eds), *Handbook of Logic in Artificial Intelligence and Logic Programming*, Oxford: Oxford University Press, 1993

[Goldszmith, Pearl 91] J. Goldsmith, J. Pearl, *System-Z^+: A formalism for reasoning with variable-strength defaults*, in *AAAI-91*, 399-404.

[Hayes 80] P.J. Hayes, "The Logic of Frames", in [Metzing 80], 46–61.

[Konolige 88] K. Konolige, *On the Relatioon Between Default and Autoepistemic Logic*, *Artificial Intelligence* **35**,343-382, 1988.

[Kraus et al. 90] S. Kraus, D. Lehman, M. Magidor, “Nonmonotonic Reasoning, Preferential Models and Cumulative Logics”, *Artificial Intelligence* **44**, 167–207, 1990.

[Makinson 93] D. Makinson, “General Patterns in Nonmonotonic Reasoning”, in [Gabbay et al. 93].

[Metzing 80] D. Metzing (Ed.), *Frame Conceptions and Text Understanding*, Berlin: de Gruyter, 1980.

[Minsky 75] M. Minsky, “A Framework for Representing Knowledge”, in P.H. Winston (Ed.): *The Psychology of Computer Vision*, 211–277, New York: McGraw-Hill, 1975.

[KR 92] B. Nebel, C. Rich, W. Swartout, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR92)*, San Mateo: Morgan Kaufmann, 1992.

[Pinkas 91] *Propositional Non-Monotonic Reasoning and Inconsistency in Symmetric Neural Networks IJCAI-91*, 525-530.

[Poole 88] D. Poole, “A logical framework for default reasoning”, *Artificial Intelligence* **36**, 27-47, 1988.

[Quantz, Royer 92] J.J. Quantz, V. Royer, “A Preference Semantics for Defaults in Terminological Logics”, in [KR 92],294–305.

[Quantz, Ryan 93] J.J. Quantz, M. Ryan, *Preferential Default Description Logics*, KIT-Report 110, Technischer Universität Berlin, 1993.

[Quantz, Schmitz 94] J.J. Quantz, B. Schmitz, “Knowledge-Based Disambiguation for Machine Translation”, *Minds and Machines* **4**, 39–57, 1994.

[Quantz 94] J.J. Quantz, “An HPSG Parser Based On Description Logics”, *to appear in COLING’94*.

[Quillian 68] M.R, Quillian, “Semantic Memory” in M. Minsky (Ed.), *Semantic Information Processing*, Cambridge (Mass): MIT Press, 1968.

[Reiter 80] R. Reiter, “A Logic for Default Reasoning”, *Artificial Intelligence* **13**, 1980.

[Shoham 87] Y. Shoham, “A semantical approach to non-monotonic logics” *IJCAI-87*.

[Smith 85] B.C. Smith, “Prologue to ‘Reflection and Semantics in a Procedural Language’ ” in R. J. Brachman, H. J. Levesque (eds), *Readings in Knowledge Representation*, Morgan Kaufman, 1985.

[Wells 29] H.G. Wells, *First and Last Things*, London: Watts & Co, 1929.