Modelling the logical mind

Modelling the logical mind

Using the cognitive architecture ACT-R to model human symbolic reasoning in the description logic $\mathcal{ALE}$

Jelle Tjeerd Fokkens

UNIVERSITY OF GOTHENBURG

Thesis submitted for the Degree of Licentiate in Logic
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg

The publication is also available in fulltext at:
`https://hdl.handle.net/2077/74797`

Typeset in Adobe Garamond Pro, Arial, Garamond-Math, and
Linux Libertine Mono O using XeLaTeX and KOMA-Script

Cover:
Reminding of a coastwards train ride after a long day full of new experiences.

Abstract

Title: Modelling the logical mind –
Using the cognitive architecture ACT-R to model
human symbolic reasoning in the description logic $\mathcal{ALE}$
Author: Jelle Tjeerd Fokkens
Language: English (with a summary in Swedish)
Department: Philosophy, Linguistics and Theory of Science
Keywords: Cognitive Modelling, Description Logic

The problem of optimising automated explanations for entailments in knowledge bases is tackled by modelling deductive reasoning processes using the cognitive architecture ACT-R. This results in the model SHARP which simulates the algorithm for deciding inconsistency of an ABox in the description logic $\mathcal{ALE}$ as executed by a human. More precisely, SHARP enables predicting the inference time of this task, which is assumed to reflect cognitive load of a human agent. With the inference time, two complexity measures on ABoxes are defined that should correlate with cognitive load by design.

Acknowledgements

Most theses are works of team effort and this thesis is no exception: it bears my name, but I had help and support from many wonderful people. That is why I would like to thank Fredrik Engström for his excellent supervision and mentoring, the many inspiring conversations and wonderful support, both academic and beyond. Thanks also to Graham Leigh, my secondary supervisor, for the many stimulating discussions and the very valuable contributions to this work, without which it would not be of the same quality. I would like to thank Rasmus Blanck for his Latex wizardry. I want to thank Bahareh Afshari and Martin Kaså for being wonderful and inspiring teachers. Many thanks also to Mattias Granberg Olsson and Dominik Wehr, my fellow PhD students who are always very supportive, clever and constructive, and many thanks to Giacomo Barlucchi, my office mate, who preceded me in achieving the licentiate degree and who helped me greatly with logic, life and a bit of the Italian language. I would like to thank all my other colleagues as well for creating a supporting, creative and high-standard working atmosphere. Many thanks to Johanna Wolff for her incredibly precise proof reading, which greatly improved this text. Lastly, I would like to thank my parents, who might not understand this text, but who supported me greatly in making it.

Contents

Chapter 1 Introduction 1
Proofs as explanations 1
Proofs and cognition 2
Artificial intelligence and explainability 3
Goals 4

Chapter 2 Knowledge Representation 5
Description Logics 6
The description logic $\mathcal{AL}$ and its relatives 7
Relations to first-order logic 12
Reasoning tasks 14
Reasoning algorithms 16
Justifications 18
Finding all justifications 19
Finding one justification 20
Improving justifications 20
Cognitive aspects 22

Chapter 3 Logic and cognition 25
Mental Logic 26
Mental Models 28
Comparison 30
Other Schools 30
Interpretations 31
Cognitive Architectures 32
ACT-R 33
Knowledge representation 34
The Goal and Imaginal modules 34
The Declarative module 35
The Procedural module 35
The Motor module 35
The Vision module 35

Base-level learning 36

Chapter 4 $\mathcal{ALE}$ ABox inconsistency 41
The $\mathcal{ALE}$ ABox inconsistency problem 41
NP-solvability 42
NP-hardness 46
A tableau algorithm for $\mathcal{ALE}$ ABox inconsistency 48

Chapter 5 The model SHARP 55
Obstacles in designing SHARP 55
The issue of parsing 55
The issue of discarding 56
The issue of the universal restrictions 56
The issue of finding new elements 57
The issue of the buffers 57
The issue of the production rules 58
SHARP's design 58
A note on nondeterminism 58
SHARP's chunk types 58
Overview 59
Module 1: find a clash 59
Module 2: find next complex formula 60
Module 3: infer from conjunction 61
Module 4: infer from existential restriction 62
Module 5: infer from universal restriction 63
Justification 64
Analysis of performance 65
Exponential scaling 65
Polynomial scaling 65
Linear scaling 68
Future extensions 68

Chapter 6 Predicting inference times with SHARP 71
Predicted effects 71
Statistical significance 72

Name dependence 73
Order insensitivity 75
Insensitivity to negations 77
Nesting sensitivity 78
Time scaling with input size 81
Spreading 83
∃ before ∀ 84
The role of decay 85
Complexity measures 87
Naive measures 88
Measures based on formal proof 88
Least-time 88
Average-time 89
Linear Combination 89
Average Inference Step time (AIS) 90
SHARP simplified 91
Comparing different measures 92

Chapter 7 Conclusion 97

References 103

Sammanfattning på svenska 111

List of figures

Figure 1 Exponential scaling of simulation time with input size in case of AND-branching. 66
Figure 2 The less complex scenarios show polynomial scaling of the simulation time with the input size. 67
Figure 3 In some easy cases, there is a linear dependence of simulation time on the size of the input. 69
Figure 4 Changing element names does not affect inference times. 73
Figure 5 Changing concept names does not affect inference times. 74
Figure 6 Changing role names does affect inference times non-trivially. 76
Figure 7 The order of presenting the formulas does not affect the inference time. 77
Figure 8 Conjunction order does not affect inference time. 78
Figure 9 Whether atomic concepts are negated or not does not affect inference times. 79
Figure 10 Nesting affects the inference times. 80
Figure 11 The inference time roughly scales exponentially with the input size in the case of AND-branching, showing a similar scaling as the simulation time in Figure 1. 82
Figure 12 The nesting of clashing concepts affects the spread of inference times. 83
Figure 13 Making inferences on existential restrictions before universal restrictions is faster. 84
Figure 14 The number of switches only slightly affects inference time, but it affects the likelihood of the runs quite profoundly (not displayed in graph) 86

List of tables

Table 1 The entailment-complexity measures 95
Table 2 The proof-complexity measures 96

1 Introduction

Many a teacher knows that an explanation may satisfy one student while it confuses another. This sometimes raises practical problems in the classroom, and on a more philosophical level sparks interesting debates centered around the question: what counts as an explanation?

1.1 Proofs as explanations

With regards to this question, mathematical proofs form interesting objects of study; they have since the fifth century been seen as explanations for why theorems hold (Harari, 2008) in addition to them being seen as mere witnesses of the theorems' truth. There are at least two different ways in which proofs can explain (Mancosu, 2018): the reductive/local way (as advocated in (Steiner, 1978)) and the unificative/holistic way (as advocated in (Kitcher, 1989)). A *reductively* explanatory proof is one in which often a certain construction is detailed that proves the theorem, whereas a *unificatively* explanatory proof proves the theorem more abstractly by showing that the theorem is a special case of a more general idea.

To contrast these two notions, (Colyvan and McQueen, 2018) gives two different proofs of the Free Group Theorem such that the two ways of proving can be compared. Because of their different nature, however, it is difficult to say which of the two proofs is the most explanatory. Moreover, proofs may generalise in different directions (Wagon, 1987), making their explanatory values difficult to compare among each other. In (Paseau, 2010), however, the author critiques the unification as explanation stance and highlights the virtues of the reductive case by giving and comparing five different proofs of the compactness theorem: Henkin's proof, a topological proof, a proof based on completeness, a combinatorial proof and a proof based on ultraproducts. One may doubt whether the above perspectives are the most fruitful tools to judge a proof's explanatory value. Indeed, in (Paseau, 2010)'s conclusion the author mentions that the explanatory value of a proof depends on its assumptions and on whether the person reading the proof is familiar with those assumptions, making the explanatory value of a proof thus a notion relative to the *understander*. It can

therefore be understood that multiple different proofs are often desirable:

> Any good theorem should have several proofs, the more the better. For two reasons: usually, different proofs have different strengths and weaknesses, and they generalize in different directions – they are not just repetitions of each other. (Raussen and Skau, 2004)

Though in the above quote it is not explicit which strengths and weaknesses are meant, explainability may well be considered one of them. The above mentioned psychological aspects of explanation in proofs are discussed relatively little in the literature. We consider these aspects to be very important, however, as it is easy to imagine a scenario in which two understanders, one highly familiar with ultraproducts and the other not, judge the explanatory value of the ultraproduct-based proof of the compactness theorem rather differently. The understander who is more acquainted with ultraproducts may experience the proof as relatively easy, whereas the understander who is less familiar with ultraproducts would experience the proof as relatively hard. From this scenario, we conclude that the cause of the difference in understandability is a psychological one: the proof posed a lower *cognitive load* on one understander than on the other.

1.2 Proofs and cognition

In (Alrabbaa et al., 2022) some psychological aspects of proofs are studied. They showed different presentations of a proof to users to see which presentation is optimal (as well as how this correlated with people's experience in logic and there scores on an IQ-like test). They concluded that shorter proofs are generally considered easiest, because the understander is then not distracted by trivial inferences. Also tree-representations of the proof are sometimes preferred over proofs written in text. These preferences were subjective and estimated by asking the participants to rate their experienced comprehensibility on a scale from 1 to 5. The paper also measured the comprehensibility of proofs more objectively by asking the subjects questions such as "Which of the following would be a correct replacement for the deduction 'XYZ' in the proof?" after giving a set of possible alternatives. This method to objectively determining proof comprehensibility turned out to be ineffective because the questions were too difficult: all scores were very low and therefore unable to be distinguished statistically.

This last point indicates that perhaps a different method is needed to measure the comprehensibility of a proof. In an attempt to construct such a method, we suggest a measure on logical entailments based on cognitive load alone, as

opposed to performance on related tasks. A proof is then most explanatory if it can be understood with the least cognitive effort by the understander. There are various ways of measuring the cognitive load of a certain task, so the above idea does not give a unique measure, but it captures the fundamental perspective of this thesis.

1.3 Artificial intelligence and explainability

In the context of AI, explanation is also becoming increasingly relevant as a reaction to the recent increase in *opaqueness* of AI systems. AI systems that exhibit machine learning can be (and become more) opaque because the system's behaviour is not explicitly programmed and can in some cases change over time based on its input. It sometimes happens that the algorithm recognises certain patterns that the programmer is not aware of (Ribeiro et al., 2016). The output of such systems is then hard to predict and even harder to understand, sometimes up to the point where the programmer is unable to give an account of the system's output. This can be especially harmful if the input data is biased and is used to make decisions that impact humans directly; e.g. ethnically biased police data.

Not only machine learning systems, but also non-learning intelligent systems may exhibit opaqueness. In the context of *knowledge bases* for example, it can be unclear how a certain inference is made. Knowledge bases are data bases that contain, besides *factual data*, a *terminology* (also called *ontology*) which contains domain knowledge in the form of hierarchically structured concepts. It is used to make inferences on the relevant concepts that a certain element satisfies. This query answering procedure is fully *symbolic* (based on logic or other formal methods (Ilkou and Koutraki, 2020) as opposed to *subsymbolic* (based on statistics or numerical calculations).

Contrary to common misconception, these systems satisy the garbage-in-garbage-out rule, because unwanted consequences may appear if input data was poorly stated. This makes it, even for these systems, sometimes hard to understand which of the axioms in the data base has given rise to the conclusion and in case of an unwanted conclusion debugging is difficult.

Therefore, these types of systems demand explainability, a trend that seemed to have been around since the early nineties (Swartout et al., 1991), but is still, and perhaps more, relevant today (Whittaker et al., 2018a). The latter source illustrates the need for explanation by mentioning a variety of examples: lethal accidents involving autonomous vehicles, a voice recognition system unrightfully cancelling visas and unsafe automatised cancer treatment recommendations.

This trend has culminated in the concept of *algorithmic accountability reporting*, where AI decisions are accounted for by certain mechanisms, although some authors opine that this may be too much to ask for, especially in the light of humans often being unable to account for their choices (Zerilli et al., 2019). Moreover, when it comes to building *trust* in AI systems, explainability is considered an essential ingredient (Lockey et al., 2021). An important development is the Regulation (EU) 2016/679 (EU, 2016) better known as GDPR, which states the right to explanation for certain automated decision processes. Another advantage of explainability is that it facilitates debugging and optimisation (Kulesza et al., 2015), on which, interestingly enough, not much research seems to have been done (Petrillo et al., 2016). This last thread of research is another motivation of this thesis.

1.4 Goals

By understanding explanations in the way stated before, we aim to:

optimise automatic explanations for entailments in knowledge bases.

The first step in this direction is modelling the cognitive process of deductive reasoning. The scope is limited to the *description logic* $\mathcal{ALE}$, with the associated reasoning task of finding out whether a given ABox is inconsistent, i.e. containing an element satisfying clashing concepts. The *cognitive architecture* ACT-R (Adaptive Control of Thought - Rational) is used for the modelling. ACT-R works largely symbolically, making it a priori well-suitable for modelling deductive reasoning; it also has *subsymbolic parameters* allowing more flexible and fine-grained modelling. The model thus constructed is called SHARP and can be seen as the main contribution of this thesis. The predictions using SHARP are used to define two *complexity measures* on the logical task of $\mathcal{ALE}$ ABox inconsistency.

Later work will focus on validating the predictions made with the model and the associated complexity measures, as well as certain computational optimisations and functional extensions, after which optimised explanations of logical entailments can be made.

2 Knowledge Representation

It is unclear when the research on artificial intelligence started, but Alan Newell's report on his General Problem Solver GPS-I (Newell et al., 1959) is considered to be one of the first publications in the field as it is currently understood. The report describes a computer program based on logic that would be able to perform general problem solving (propositional logical deductions, solving algebraic equations etc.). The program consisted of problem-specific knowledge encoded in Horn clauses (logical formulas of a specific form) as well as deduction rules to transform certain expressions into others and was the first system that had a clear separation between the two. The generality of this problem solver was rather limited, as only simple-to-formalise problems could be solved; moreover, the program was unable to tackle the vast search space effectively. GPS-I developed into the Soar cognitive architecture (which will be discussed in later sections), but it also moved in another direction: expert systems (Hayes-Roth and Waterman, 1983).

An example of an expert system is MYCIN (Shortliffe and Buchanan, 1975). This expert system is designed to reason about medical diagnoses related to bacterial infections and the corresponding antibiotic treatments and allows for 'inexact' reasoning in case of probabilistic data. It computed certainty factors for the relevant propositions and asked the user for more information on the patient under consideration. The system later spawned research on Bayesian networks; it was never commercially used itself, because the process of entering the data was too cumbersome.

The problem of how to most effectively represent knowledge remained and was adressed in (Minsky, 1974). Minsky proposed a frame language with the aim of designing systems that could reason with 'common-sense thought' as opposed to reason only about specific facts. In 1977 one of the first such frame-based knowledge representation system was published: KL-ONE. It had logical machinery to reason about classes and relations and it could make assertions about individuals (Brachman and Schmolze, 1985). The distinction between knowledge of concepts (often called *ontology* or *terminology*) and knowledge of individuals proved very powerful and is still relevant today. KL-ONE formed the basis of research on description logics when in the paper (Brachman and

Levesque, 1984) the trade-off was discussed between expressiveness on the one hand and tractability of reasoning on the other. The term *Description Logics* only became popular later after research focus has shifted from constructors with which to form concepts to the logical properties of knowledge representation systems (Baader et al., 2003).

The transition from a database to a knowledge base is a non-trivial one regarding the closed world versus open world assumption, CWA and OWA respectively. Most databases usually use the CWA, meaning that if something is not in the data base, it is assumed to be false. Logic, on the other hand, uses the OWA so that if something does not follow from a set of axioms, it cannot generally be assumed to be false. Extending a data base to a knowledge base with conceptual knowledge and logical reasoning, therefore, means that the usual CWA is dropped. For that reason, a knowledge base admits many more models than a data base. For example, if an element a is not assumed to satisfy the concept C, then in most data bases under the CWA it satisfies the concept $\neg C$, i.e. the complement of the concept C. Knowledge bases, using the OWA, remain indifferent as to a satisfying C or $\neg C$, meaning that they admit at least two models, one in which a satisfies C and one in which it satisfies the complement $\neg C$. A discussion of this can be found in (Baader et al., 2003, p.68).

Knowledge representation systems are still being developed today and form a rich area of research (Sowa, 2000). They have many applications, including SNOMED CT, a systematic medical terminology that has been being developed for over 50 years (Cornet and de Keizer, 2008) (although its practical usefulness is unclear). Another direction of development is extending the World Wide Web to the Semantic Web, with the aim of making the internet machine-readable, and therefore requiring the knowledge of the web to be structured (Horrocks et al., 2023). A language developed for this purpose is OWL, which is widely used to construct ontologies (Wang et al., 2006). OWL officially stands for Web Ontology Language; the reason for permuting the acronym's letters is unknown (Herman, 2010). Constructing ontologies is a difficult task, because the causes of bugs are difficult to trace. Many editor tools are available for constructing ontologies, e.g. Protégé. We return to facilitating of ontology debugging at the end of this chapter after a discussion on description logics.

2.1 Description Logics

Description logics lie at the foundation of ontologies and form a big family of different logics where each one has specific properties tailored to the desired

purpose. A good introduction is (Baader et al., 2017) and more detailed info can be found in (Baader et al., 2003). Much of the following is derived from the latter source.

As mentioned above, description logics are used to represent and reason about knowledge. In principle, first-order logic could be used for this purpose, but there are several reasons why this is not done. The first reason is a historical one, as description logic originated outside of logic and connections with logic were discovered later. The second one is that it is easier to read, which will become clearer later. The third reason is that computational properties such as the complexity of certain reasoning tasks are more closely tied to the syntax of the language, making it easier to see why such properties are ensured. Lastly, some description logics go beyond first-order logic in that they can express transitive closure of roles.

2.1.1 The description logic $\mathcal{AL}$ and its relatives

The description logic that is considered most fundamental to the family is $\mathcal{AL}$, which stands for Attributive Language. This section is devoted to the description logic $\mathcal{AL}$ and its closest relatives, but two other important branches of the family are the $\mathcal{EL}$ (Existential Language) family and the $\mathcal{FL}$ (Frame based Language) family, which are both weaker than the first.

Definition 2.1.1. The triplet $\mathbf{S} = (\mathbf{A}_C, \mathbf{A}_R, \mathbf{A}_E)$, where $\mathbf{A}_C$ is a set of atomic concept symbols, $\mathbf{A}_R$ is a set of atomic role symbols and $\mathbf{A}_E$ is a set of element symbols, is called a *signature*.

The signature contains all the primitive concept, role and element symbols in the language. If $A \in \mathbf{A}_C$, we can write $A \in \mathbf{S}$ if it is clear from the context that A is a concept symbol.

Definition 2.1.2. Let $\mathbf{S}$ be a signature. A *concept description* in $\mathcal{AL}$ is made according to the following syntax in Backus-Naur Form (BNF), where $A \in \mathbf{S}$.

$$C ::= A \mid \top \mid \bot \mid \neg A \mid C \sqcap C \mid \forall r.C \mid \exists r.\top.$$

The expressions respectively stand for: atomic concepts, the universal concept, the bottom concept, complement of atomic concept, concept intersection, universal restriction and limited existential restriction.

Complementation (negation) is only applied to atomic concepts and existential restriction is limited in the sense that it only uses the $\top$ concept to restrict to,

instead of an arbitrary concept C. The logic $\mathcal{EL}$ allows only concept intersection and existential restriction. The logic $\mathcal{FL}$ allows, apart from the above, also role restrictions, but does not allow negations. Much of the following is stated as generally as possible, making it apply to $\mathcal{EL}$ and $\mathcal{FL}$ as well as $\mathcal{AL}$; if it only applies to $\mathcal{AL}$ this will be mentioned.

Definition 2.1.3. Let **S** be a signature and $\mathcal{L}$ some description logic. A *TBox* $\mathcal{T}$ in signature **S** for the logic $\mathcal{L}$ is a set of formulas ϕ, where each ϕ has the form:

$$C \sqsubseteq D \text{ or } C \equiv D, \tag{2.1}$$

where C and D are concept descriptions in the signature **S** and the logic $\mathcal{L}$.

Definition 2.1.4. Let **S** be a signature and $\mathcal{L}$ some description logic. An *ABox* $\mathcal{A}$ in signature **S** for the logic $\mathcal{L}$ is a set of formulas ϕ, where each ϕ has the form:

$$a : C \text{ or } (a, b) : r, \tag{2.2}$$

where C is any concept and r and role, both of which are in the signature **S** and the logic $\mathcal{L}$.

The T in TBox stands for *terminological*, as TBoxes contain all the terminological knowledge of the system, i.e. the knowledge of how concepts relate to other concepts. Information about specific individuals is encoded in ABoxes, where A stands for *assertional*.

Formulas of the second form in the above definition are in some texts excluded from ABoxes. Such formulas are then included in RBoxes, where R stands for *relational*. This distinction is not used in this text and the above definition applies in the following.

Definition 2.1.5. Let **S** be a signature and $\mathcal{L}$ some description logic. A *knowledge base* $\mathcal{K}$ in signature **S** for the logic $\mathcal{L}$ is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox, both of which are in signature **S** for the logic $\mathcal{L}$.

Semantically, we have the following.

Definition 2.1.6. (Baader et al., 2003, p.48)**.** Let **S** be a signature. An *interpretation* in signature **S** and logic $\mathcal{AL}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set and $\mathcal{I}$ is an interpretation function mapping concept names $A \in \mathbf{S}_C$ into $\wp(\Delta^{\mathcal{I}})$, the powerset of the domain, mapping role names $r \in \mathbf{S}_R$ into $\wp(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$, and mapping element names $a \in \mathbf{S}_E$ into $\Delta^{\mathcal{I}}$, such that the following conditions are

satisfied:

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall r.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in r^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists r.\top)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in r^{\mathcal{I}}\}
\end{aligned}$$

With the interpretations defined we can define the satisfaction conditions for a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$.

For TBox axioms, we have $\mathcal{I} \models C \sqsubseteq D$ if and only if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. We say that $C \sqsubseteq D$ is true for interpretation $\mathcal{I}$. Moreover, $\mathcal{I} \models C \equiv D$ if and only if $C^{\mathcal{I}} = D^{\mathcal{I}}$; where we say that $C \equiv D$ is true for interpretation $\mathcal{I}$. For a TBox $\mathcal{T}$, we have $\mathcal{I} \models \mathcal{T}$ if and only if for all formulas $\phi \in \mathcal{T}$, $\mathcal{I} \models \phi$.

For ABox axioms we have $\mathcal{I} \models a : C$ if and only if $a^{\mathcal{I}} \in C^{\mathcal{I}}$. We say that $a : C$ is true for interpretation $\mathcal{I}$. Moreover, $\mathcal{I} \models (a,b) : r$ if and only if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$, where we say that $(a,b) : r$ is true for interpretation $\mathcal{I}$. We call $b^{\mathcal{I}}$ the r-successor of $a^{\mathcal{I}}$ and $a^{\mathcal{I}}$ the r-predecessor of $b^{\mathcal{I}}$. For an ABox $\mathcal{A}$, we have that $\mathcal{I} \models \mathcal{A}$ if and only if for all formulas $\phi \in \mathcal{A}$, $\mathcal{I} \models \phi$.

For a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we have $\mathcal{I} \models \mathcal{K}$ if and only if $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$. We then call $\mathcal{I}$ a *model* for $\mathcal{K}$. For any ABox or TBox sentence ϕ, we have $\models \phi$ if and only if $\mathcal{I} \models \phi$ for all interpretations $\mathcal{I}$.

The logic $\mathcal{AL}$ can be expanded with various concept constructors, some of which are described below. Most of the following constructors are associated with a script letter, the naming conventions of which will be explained later.

Union

The logic $\mathcal{AL}$ can be extended with *unions* of concepts. Concept descriptions then allow for formulas of the form $C \sqcup D$, where $C, D \in \mathbf{A}_C$. Expressions like this are then interpreted by an interpretation $\mathcal{I}$ as:

$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}.$$

This construction is indicated with the letter $\mathcal{U}$.

Existential restriction

Full existential restriction (also: quantification) allows concept descriptions of the form $\exists r.C$. Note that C can be any concept description, so it does not need to be $\top$ as in the logic $\mathcal{AL}$. For an interpetation $\mathcal{I}$, we have:

$$(\exists r.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in r^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}.$$

So, compared to limited existential restriction, elements that are claimed to exist can be claimed to satisfy any concept description. This concept construction is indicated by $\mathcal{E}$.

Number restrictions

There are two different kinds of number restrictions: $\leqslant nr$ and $\geqslant nr$, for at-most and at-least restriction respectively, n being a natural number. For an interpretation $\mathcal{I}$, the concepts are interpreted as:

$$(\leqslant nr)^{\mathcal{I}} = \left\{a \in \Delta^{\mathcal{I}} \;\middle|\; |\{b \mid (a,b) \in r^{\mathcal{I}}\}| \leq n\right\},$$

and

$$(\geqslant nr)^{\mathcal{I}} = \left\{a \in \Delta^{\mathcal{I}} \;\middle|\; |\{b \mid (a,b) \in r^{\mathcal{I}}\}| \geq n\right\}.$$

Conjunctions of the two concepts allow for number restrictions that express that there are exactly that many r-successors. Logics that allow concept constructors like these are labelled by $\mathcal{N}$.

Negation

As mentioned above, complements in $\mathcal{AL}$ can only be applied to atomic concepts and not to compound concepts. We can therefore extend the logic by allowing the negation to be applied to any concept: $\neg C$. An interpretation $\mathcal{I}$ needs to satisfy:

$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}.$$

Logics with this concept constructor are labelled by $\mathcal{C}$.

Boolean role constructors

Boolean operations that apply to concepts can also be applied to roles. For roles r, s, their intersection is for example straightforwardly written as $r \sqcap s$. It is interpreted as:

$$(r \sqcap s)^{\mathcal{I}} = r^{\mathcal{I}} \cap s^{\mathcal{I}}.$$

Likewise we can have role union $(r \sqcup s)$, interpreted as:

$$(r \sqcup s)^{\mathcal{I}} = r^{\mathcal{I}} \cup s^{\mathcal{I}}.$$

And we also have role complement $\neg r$:

$$(\neg r)^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus r^{\mathcal{I}}.$$

There seems to be no notational concensus on which of the above constructors are present in the logic; sometimes the corresponding symbol is written in superscript, e.g. $\mathcal{ALCN}^{\sqcap}$, in other cases it is explicitly mentioned.

Role composition

Roles may be composed with each other. For two roles r and s, we write $r \circ s$, which is intepreted as:

$$(r \circ s)^{\mathcal{I}} = \left\{(a, c) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \;\middle|\; \exists b.(a, b) \in r^{\mathcal{I}} \wedge (b, c) \in s^{\mathcal{I}}\right\}.$$

There again seems to be no standardised notation to indicate the presence of role composition in a description logic.

Transitive closure of roles

Elements may be 'connected' by arbitrarily long chains of role compositions. This is called the transitive closure and is notated as r^{+}. It has the formal interpretation:

$$(r^{+})^{\mathcal{I}} = \bigcup_{i \geq 1} (r^{\mathcal{I}})^{i}.$$

Here, $r^{1} = r$ and $r^{i+1} = r \circ r^{i}$. Usually it seems that transitive closure of roles is indicated by subscript 'trans', e.g. $\mathcal{ALC}_{trans}$.

Role inverses

One may be interested in inverse roles, i.e. a b related to an a if a relates to b. This is written as r^-. Semantically we have:

$$(r^-)^{\mathcal{I}} = \left\{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \;\middle|\; (a,b) \in r^{\mathcal{I}}\right\}.$$

If a description logic admits role inverses, this seems usually explicitly mentioned.

The family

We get different logics by taking different combinations of the concept constructors discussed above. These logics are named by the convention:

$$\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{C}], \tag{2.3}$$

where square brackets indicate optional inclusion and each letter indicates the presence of the corresponding concept constructor. The logic $\mathcal{ALC}$ with transitive roles is abbreviated as $\mathcal{S}$.

The above logics are not all distinct in expressivity, because for logics with negation and disjunction: $\models C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$, so some concept constructors can be used to define others. In fact, it is possible to use only the symbols $\mathcal{U}$, $\mathcal{E}$ and $\mathcal{N}$ to distinguish all the semantically distinct logics that can be made with the above constructors. $\mathcal{UE}$ is replaced by $\mathcal{C}$, because of such semantic equivalence (Baader et al., 2003, pp. 49 - 50).

2.1.2 Relations to first-order logic

Most formulas of the description logics above can be translated into first-order logic (FOL); the exception being the transitive closure of roles because of the compactness theorem: finiteness cannot be expressed in FOL. Concepts and roles are translated as unary and binary predicates in FOL. In doing so, every complex concept C can be translated into a formula $\phi_C(x)$ with one free variable x which ranges over the domain $\Delta^{\mathcal{I}}$ such that a satisfies $\phi_C(x)$ if and only if $a \in C^{\mathcal{I}}$. The variables that appear in the translated formula are not present in the description logic formulas, so the translation is parametrised by those variables.

Definition 2.1.7. The *translations* τ_x, σ_{xy} and Φ from description logic (DL) concepts and roles to first-order logic (FOL) formulas are defined recursively as follows:

$$
\begin{aligned}
\tau_x(A) &= A(x) \\
\sigma_{xy}(r) &= R(x,y) \\
\tau_x(C \sqcap D) &= \tau_x(C) \wedge \tau_x(D) \\
\tau_x(C \sqcup D) &= \tau_x(C) \vee \tau_x(D) \\
\tau_x(\neg C) &= \neg \tau_x(C) \\
\tau_x(\exists r.\, C) &= \exists \bar{x}.\, \sigma_{x\bar{x}}(r) \wedge \tau_{\bar{x}}(C) \\
\tau_x(\forall r.\, C) &= \forall \bar{x}.\, \sigma_{x\bar{x}}(r) \rightarrow \tau_{\bar{x}}(C) \\
\tau_x(\geqslant nr) &= \exists y_1 ... y_n.\, \sigma_{xy_1}(r) \wedge ... \wedge \sigma_{xy_n}(r) \wedge \bigwedge_{i<j} y_i \neq y_j \\
\tau_x(\leqslant nr) &= \forall y_1 ... y_{n+1}.\, \sigma_{xy_1}(r) \wedge ... \wedge \sigma_{xy_{n+1}}(r) \rightarrow \bigvee_{i<j} y_i = y_j \\
\sigma_{xy}(r \sqcap s) &= \sigma_{xy}(r) \wedge \sigma_{xy}(s) \\
\sigma_{xy}(r \sqcup s) &= \sigma_{xy}(r) \vee \sigma_{xy}(s) \\
\sigma_{xy}(\neg r) &= \neg \sigma_{xy}(r) \\
\sigma_{xy}(r \circ s) &= \exists z.\, \sigma_{xz}(r) \wedge \sigma_{zy}(s) \\
\sigma_{xy}(r^-) &= \sigma_{yx}(r),
\end{aligned}
$$

where A is a unique unary predicate symbol in FOL corresponding to the atomic concept A in the description logic and R is a unique binary predicate symbol in FOL corresponding to the primitive role symbol r in the description logic. Furthermore, $\bar{x} = y$ and $\bar{y} = x$. Using the above translations, we can define the *translation* Φ from concept/role inclusions/equivalences and assertional statements in description logic to first-order logic sentences:

$$
\begin{aligned}
\Phi(C \sqsubseteq D) &= \forall x.\, \tau_x(C) \rightarrow \tau_x(D) \\
\Phi(C \equiv D) &= \forall x.\, \tau_x(C) \leftrightarrow \tau_x(D) \\
\Phi(R \sqsubseteq S) &= \forall xy.\, \sigma_{xy}(R) \rightarrow \sigma_{xy}(S) \\
\Phi(R \equiv S) &= \forall xy.\, \sigma_{xy}(R) \leftrightarrow \sigma_{xy}(S) \\
\Phi(a : C) &= \tau_x(C)[a/x] \\
\Phi((a,\ b) : r) &= \sigma_{xy}(r)[a/x,\ b/y],
\end{aligned}
$$

where for every element name c in DL, there is a unique corresponding constant c in FOL. The last two lines are cases of substitution for the free variables in the

respective FOL formulas.

With the above translation, most description logic formulas can be translated into the two-variable fragment $\mathcal{L}^2$ of FOL with only unary and binary predicates, the exception being role composition and the number restrictions. Note that the existential and universal restrictions make use of the function $\bar{\cdot}$, the iterated application of which alternates between the two variables x and y. This ensures that the translation step involving an existential or universal formula does not create an expression with more than two variables (though expressions exceeding this limit may be created if role composition or number restrictions are present).

Translations can also be made to modal logics. For example $\mathcal{ALC}$ is equivalent to the multi-modal logic $\mathbf{K}_m$ (Schild, 1991) and the description logic $\mathcal{ALC}_{reg}$ (with role union, role composition and transitive closure on roles) is equivalent to Propositional Dynamic Logic (PDL) (de Giacomo and Lenzerini, 1994).

2.1.3 Reasoning tasks

Before we can discuss the various reasoning algorithms that are available for the different description logics, a few notions need to be explicated.

Definition 2.1.8. (Baader et al., 2003, p.51)**.** A *definition* is a TBox axiom of the form $A \equiv D$ where A is an atomic concept.

In the following we will focus on TBoxes with only definitions and ignore cases in which the left-hand side is a compound concept.

A *base symbol* of a TBox $\mathcal{T}$ is an atomic concept symbol that occurs only in the right-hand side of a definition in $\mathcal{T}$. The rest of the atomic concept symbols, i.e. the ones that occur in some left hand side of a definition are called the *name symbols*. A *base interpretation* of $\mathcal{T}$ is an interpretation $\mathcal{I}$ that interprets only the base symbols in $\mathcal{T}$.

For the following, we focus exclusively on definitions, as concept inclusions can be replaced by them while preserving the satisfaction relation. For example, a concept inclusion $A \sqsubseteq C$ in which A_C does not occur can be replaced by $A \equiv A_C \sqcap C$. Any interpretation $\mathcal{I}$ that satisfies $A \sqsubseteq C$ can be extended to an interpretation $\mathcal{I}'$ that satisfies $A \equiv A_C \sqcap C$ by setting $A_C^{\mathcal{I}'} = A^{\mathcal{I}}$ (Baader et al., 2017, p.23).

Definition 2.1.9. (Baader et al., 2003, p.52)**.** For atomic concepts C and D occuring in a TBox $\mathcal{T}$, we say that C *directly uses* D if the latter appears on the

right-hand side of a definition of the former. Now *uses* is the transitive closure of the relation *directly uses*.

Definition 2.1.10. (Baader et al., 2003, p.52)**.** A TBox $\mathcal{T}$ contains a *cycle* if and only if an atomic concept C in $\mathcal{T}$ uses itself. $\mathcal{T}$ is then said to be *cyclic*. If $\mathcal{T}$ does not contain any cycle, it is *acyclic*.

In the following we only focus on acyclic TBoxes. An acyclic TBox $\mathcal{T}$ has the property of being *definitorial*, meaning that every base interpretation $\mathcal{I}$ has a unique extension $\mathcal{J}$ that is a model of $\mathcal{T}$, i.e. the interpretation of the name symbols is determined by the interpretation of the base symbols.

The following definition describes a process with which the information from the TBox can be incorporated in the ABox.

Definition 2.1.11. (Baader et al., 2017, p.25)**.** Let a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be given, such that $\mathcal{T}$ is acyclic and of the form $\mathcal{T} = \{A_i \equiv C_i \mid 1 \leq i \leq m\}$, where A_i is atomic. Let $\mathcal{A}_0 = \mathcal{A}$ and let $\mathcal{A}_{j+1}$ be constructed by:

1. find some formula $a : D \in \mathcal{A}_j$ that has some A_i occuring in D for some $1 \leq i \leq m$;
2. replace all occurences of A_i in D with C_i, the definition of A_i.

Let k be the number of steps after which no more such replacements are possible. $\mathcal{A}_k$ is the *result of unfolding $\mathcal{T}$ into $\mathcal{A}$*.

It can be proved that the result of unfolding an acyclic TBox exists (Baader et al., 2017). Moreover, for a given knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a model $\mathcal{M}$ of the result $\mathcal{A}_k$ of unfolding the TBox $\mathcal{T}$ has a unique extension to a model $\mathcal{M}'$ of $\mathcal{K}$. This follows from the facts that $\mathcal{T}$ is definitorial and $\mathcal{A}_k$ is base interpretation (the concept names that are not base concepts have been removed by the unfolding procedure).

The algorithm from definition 2.1.11 is eager and its duration may scale exponentially with the size of the knowledge base (Baader et al., 2017, p.27). There exists also a lazy algorithm which limits the times of unfolding to only a selection of cases. For now it suffices to understand that acyclic knowledge bases can in principle be reduced to just ABoxes so that knowledge base reasoning tasks can be reduced to ABox reasoning tasks.

Reduction

An ABox is inconsistent if it entails a *clash*, i.e. two expressions of the form $a : C$ and $a : \neg C$. Many reasoning tasks that one may be interested in can be reduced to

the task of checking inconsistency. For example, checking whether $\mathcal{K} \models C \sqsubseteq D$ is equivalent to checking whether $\mathcal{K} \cup \{a : C \sqcap \neg D\}$ is inconsistent for a new in $\mathcal{K}$. Checking whether $\mathcal{K} \models C \equiv D$, amounts to checking the inconsistency of both $\mathcal{K} \cup \{a : C \sqcap \neg D\}$ and $\mathcal{K} \cup \{a : D \sqcap \neg C\}$, with a new in $\mathcal{K}$. Checking whether C and D are disjoint in $\mathcal{K}$ is equivalent to checking whether $\mathcal{K} \cup \{a : C \sqcap D\}$, with a new in $\mathcal{K}$, is inconsistent. Checking whether $\mathcal{K} \models a : C$ is equivalent to checking whether $\mathcal{K} \cup \{a : \neg C\}$ is inconsistent. Checking whether a concept C is satisfiable in $\mathcal{K}$ is equivalent to checking whether $\mathcal{K} \cup \{a : C\}$ with a new a is not inconsistent. A more elaborate reasoning task is finding all elements that satisfy a given concept C; this can also be reduced to checking ABox inconsistency by iterating over all elements a_i appearing in $\mathcal{K}$ and checking whether $\mathcal{K} \models a_i : C$, a task previously reduced to ABox inconsistency.

It should be noted that not all of the above reductions are available for all description logics, as some reductions require the description logic to have full negation.

2.1.4 Reasoning algorithms

For logics without full negation, the subsumption task (finding out whether $C \sqsubseteq D$ holds for concepts C and D) is solved by so-called *structural subsumption* algorithms, because no reduction to ABox inconsistency can be made. For description logics with full negation and disjunction, these structural subsumption algorithms are incomplete. Hence, for those logics, *tableau-based* algorithms are used (Baader et al., 2003, p.74).

Structural subsumption

The structural subsumption algorithms are only applicable to weak logics. This section illustrates the workings of these algorithms by showing one for the description logic $\mathcal{FL}_0$. This logic is a sublogic of $\mathcal{AL}$ and has only concept conjunction $C \sqcap D$ and universal restriction $\forall r.C$. ABoxes in this logic are always consistent, because it is not possible to express a contradiction in this language. The algorithm calculates the subsumption relation by first transforming the concepts into a normal form and then checking the normal forms against each other. In this logic a concept description is in normal form if and only if it has the form:

$$A_1 \sqcap \ldots \sqcap A_m \sqcap \forall r_1.C_1 \sqcap \ldots \sqcap \forall r_n.C_n,$$

where A_i are atomic concepts and C_j are concepts in normal form. All concepts can be easily transformed into normal form by utilising the properties of $\sqcap$ and

the equivalence between $\forall r.(C \sqcap D)$ and $(\forall r.C) \sqcap (\forall r.D)$.

The subsumption is then tested for, by making use of the following theorem.

Theorem 2.1.1. (Baader et al., 2003, p.76)**.** Let

$$\begin{aligned} C &= A_1 \sqcap ... \sqcap A_m \sqcap \forall r_1.C_1 \sqcap ... \sqcap \forall r_n.C_n, \text{ and} \\ D &= B_1 \sqcap ... \sqcap B_k \sqcap \forall s_1.D_1 \sqcap ... \sqcap \forall s_l.D_l, \end{aligned} \tag{2.4}$$

be two concepts in normal form. Then $C \sqsubseteq D$ if and only if the following two conditions hold:

- for all $1 \leq i \leq k$, there exists a $1 \leq j \leq m$ such that $B_i = A_j$.
- for all $1 \leq i \leq l$, there exists a $1 \leq j \leq n$ such that $S_i = R_j$ and $C_j \sqsubseteq D_i$.

This property can then be used to check the subsumption relation between two concept descriptions in polynomial time (Baader et al., 2003, p.77).

The algorithm can be extended to logics stronger than $\mathcal{FL}_0$ which for example allow for the bottom concept ($\bot$), atomic negation ($\neg A$) and number restrictions ($\leqslant nr$ and $\geqslant nr$). For many stronger logics, however – e.g. logics including full negation ($\neg C$) or concept union ($C \sqcup D$) – this type of algorithm is incomplete and tableau based methods are necessary.

Tableau algorithms

For logics with full negation tableau-based algorithms are used. They operate in three steps, an illustration of which is given in this section (Baader et al., 2003, p.78). As a first step, when the algorithm checks whether $(\exists r.A) \sqcap (\exists r.B)$ is subsumed by $\exists r.(A \sqcap B)$, it checks whether the concept

$$C = (\exists r.A) \sqcap (\exists r.B) \sqcap \neg\exists r.(A \sqcap B)$$

is unsatisfiable. Note that it has the negation of the concept which is supposed to subsume the other. This is an instance of the equivalence between $C \sqsubseteq D$ and unsatisfiability of $C \sqcap \neg D$, which is only possible if the logic has full negation available.

In the second step, the concept is transformed to the normal form

$$C_0 = (\exists r.A) \sqcap (\exists r.B) \sqcap \forall r.(\neg A \sqcup \neg B),$$

where negations occur only in front of atomic concepts. Here we use a disjunction to make use of DeMorgan's laws, the logic should of course allow for this.

The last step is an *ABox inconsistency algorithm*, where an attempt is made to construct an interpretation for C_0. One specific such algorithm will be discussed in more detail later. Generally, the algorithm starts by generating an individual that is supposed to satisfy the concept, for example: $a : C_0$. From this, certain deductions are made, in this example: $a \in (\exists r.A)^{\mathcal{I}}$, $a \in (\exists r.B)^{\mathcal{I}}$, and $a \in (\forall r.(\neg A \sqcup \neg B))^{\mathcal{I}}$.

These facts in turn give rise to new deductions, namely the first fact claims the existence of an element (for example b) such that: $(a, b) \in r^{\mathcal{I}}$ and $b \in A^{\mathcal{I}}$. Analogously, there must be an element (for example c) such that both $(a, c) \in r^{\mathcal{I}}$ and $c \in B^{\mathcal{I}}$. The algorithm treats b and c as distinct. From the third fact we can deduce that $b \in (\neg A \sqcap \neg B)^{\mathcal{I}}$ so that $b \in (\neg A)^{\mathcal{I}}$ or $b \in (\neg B)^{\mathcal{I}}$. The first possibility would clash with the earlier derived fact that $b \in A^{\mathcal{I}}$, but the second possibility is viable. Similarly, we deduce that $c \in (\neg A)^{\mathcal{I}}$. Now the algorithm is done and an explicit model has been made for C_0. This model violates the subsumption we checked and we conclude that the subsumption does not hold.

Similar algorithms can be constructed for more complex logics, but the basic idea is the same: a reduction is made to the ABox inconsistency algorithm by inferring simpler facts from the given axioms. In chapter 4 we will return to the ABox inconsistency algorithm for the description logic $\mathcal{ALE}$.

2.2 Justifications

It was mentioned before that it is often difficult to find the cause of a certain unwanted conclusion ϕ (a clash or a false statement) from a knowledge base, especially if the latter is very large. A famous example is SNOMED CT, a medical ontology, in which a bug caused the concept 'amputation of the upper limb' to be subsumed by the concept 'amputation of the finger' (Baader and Suntisrivaraporn, 2008).

Axiom pinpointing is a method to find *justifications*, also known as *MinAs* (minimal axiom sets) that may help find the cause of the unwanted conclusion. A good introduction to axiom pinpointing can be found in (Peñaloza, 2019) and an application to the medical terminology DICE can be found in (Schlobach and Cornet, 2003). A justification is defined as a subset $\mathcal{O} \subseteq \mathcal{K}$ of the knowledge base such that both $\mathcal{O} \models \phi$, where ϕ is the unwanted conclusion and for all its proper subsets $\mathcal{O}' \subsetneq \mathcal{O}$, we have $\mathcal{O}' \not\models \phi$. This subontology $\mathcal{O}$ is generally not unique; in fact, the number of justifications can grow exponentially with the number of axioms. It is argued that justifications are better than proofs in explaning why a certain conclusion follows, as their structures are simpler and no

proof system needs to be understood by the user; besides that, they are practical to find automatically (Horridge, 2011, p.221), making them a popular form of explanation. Depending on the user's goals one may be interested in finding all, several or some of the justifications.

2.2.1 Finding all justifications

To find all justifications to a given conclusion, one method has been developed based on a tableaux algorithm, an example of which we saw earlier. The technique is called *tracing* and its idea is to label each axiom in the knowledge base by an individual propositional variable after which derived formulas are labelled using these propositional variables.

More precisely, a formula ψ, derived from ϕ_1 and ϕ_2 that have labels p_1 and p_2 respectively, is labelled with the conjunction $p_1 \wedge p_2$, except if ψ already has a label l; it then gets labelled by the disjunction $l \vee (p_1 \wedge p_2)$. If a formula ψ is derived from ϕ_1 and ϕ_2 that have labels $P \vee Q_1 \vee ... \vee Q_n$ and $P \wedge R$ respectively, it gets the label $(P \wedge R) \vee (Q_1 \wedge R) \vee ... \vee (Q_n \wedge R)$, so that the disjuncts are 'distributed' over the conjuncts, forming a disjunction. In such a way, each clash gets associated with a propositional formula that expresses all combinations of axioms that it follows from (Baader et al., 2007). We illustrate the method with an example.

Let $\mathcal{K} = \{b : \neg C,\ b : (C \sqcap D),\ (a, b) : r,\ a : \forall r.C\}$. And let's label the axioms by p_1, p_2, p_3, p_4 from left to right. The formula $b : C$ can be derived from p_2 and gets labelled as such. The clash $b : \neg C$ and $b : C$ is labelled with the conjunction of both labels, namely $p_1 \wedge p_2$. The formula $b : C$ can also be derived from p_3 and p_4, which means that it gets the label $p_3 \wedge p_4$. The clash's final label becomes the disjunction of the two $(p_1 \wedge p_2) \vee (p_1 \wedge p_3 \wedge p_4)$, where each of the disjuncts represents a justification.

Another approach was developed based on automata. It has the advantage that termination is always guaranteed, although the practicality is limited due to the fact that its best-case complexity is similar to the worst-case complexity. Both methods, however, suffer from complexity issues that makes practical applications difficult. Even for expressively weak description logics, the number of justifications grows exponentially with the size of the knowledge base. Examples are the TBoxes $\mathcal{T}_n = \{B_{i-1} \sqsubseteq P_i \sqcap Q_i,\ P_i \sqsubseteq B_i,\ Q_i \sqsubseteq B_i \mid 1 \leq i \leq n\}$, which grow linearly with n, but the conclusion $B_0 \sqsubseteq B_n$ has 2^n many justifications as for each i only one of the axioms $P_i \sqsubseteq B_i$ and $Q_i \sqsubseteq B_i$ is needed for the proof. Therefore, there is a demand for finding only one justification instead of all of them.

2.2.2 Finding one justification

In the so-called *black box* approach for finding one justification, one axiom is deleted from the knowledge base, after which it is checked if the given conclusion is still entailed by it. If the conclusion still follows, the axiom is permanently removed, but otherwise it is put back. This is procedure iterates over all the axioms and what is left is a justification. In the above example, removing the first axiom ($b : \neg C$) from $\mathcal{K}$ makes the clash ($b : \neg C$ and $b : C$) no longer derivable, so it is put back in. Then, removing the second axiom ($b : C \sqcap D$) makes the clash still derivable, so it is permanently removed. Then removing either one of the last two axioms makes it impossible to derive the clash again, so they cannot be removed. The justification thus found is: $\mathcal{O} = \{b : \neg C,\ (a, b) : r,\ a : \forall r.C\}$.

Glass box approaches are again based on the tableaux algorithms extended with the tracing technique discussed above. Because of the fact that only one justification needs to be found, such algorithms are much simpler and do not suffer from the issues like before; among other things, termination is now guaranteed. One can for example label the axioms and label every derived formula with the labels of the formulas it is derived from. The clash is then labelled with a list of the axioms used to derive it. In our running example, we could infer $b : C$ from the second axiom $b : (C \sqcap D)$ and we readily have a clash with the first axiom. The conjunction $p_1 \wedge p_2$ now represents a justification.

A *grey box* approach is a combination of both approaches above, in that the glass box algorithm finds an approximate justification with possibly a few superfluous axioms, after which the black-box axiom minimises the axiom set to form a proper justification.

2.2.3 Improving justifications

As opposed to justifications – which lack any internal structure – there are proofs, which have a tree-like structure that shows the interrelations between the axioms and the relevant derived formulas. The approaches of e.g. (Alrabbaa et al., 2020b) and (McGuinness, 1996) stand in the latter tradition of generating automated explanations. Justifications, however, seem the dominant form of automated explanations because they are easy to compute, and there is no need for users to understand a proof system. Recently, in-between approaches are emerging, where some structure is added to a justification without turning it into full proofs, although there are also other techniques to improve justifications themselves. A cognitively adequate measure on proof complexity can be useful in creating these optimised versions of justifications and proofs.

The paper (Peñaloza, 2019) mentions several extensions of the above techniques. One extension takes into account the granularity of the axioms: instead of considering axioms as indivisible units, the *superfluous parts* that are not relevant to the derivation of interest are ignored. For example the set $\{a : A \sqcap B \sqcap C,\ A \equiv D\}$ is a justification for $\{a : D\}$, but the part $\sqcap B \sqcap C$ in the first axiom is not relevant for this deduction. Moreover, the axiom $A \equiv D$ is superfluous in the sense that only $A \sqsubseteq D$ is needed to derive the conclusion. These superfluous parts are logically irrelevant to derive the conclusion, but they could distract the user. Ignoring them seems beneficial from a debugging/understanding point of view, so that $\{a : A,\ A \sqsubseteq D\}$ might be a more effective justification to present to the user.

Apart from superfluity in axioms, justifications can result in *masking*. This effect takes place when a justification allows for multiple different reasons for the conclusion.

For example, the set $\mathcal{J} = \{A \sqsubseteq B \sqcap \mathbf{C},\ A \sqsubseteq \mathbf{C} \sqcap D,\ E \equiv B \sqcap C \sqcap D\}$ is a justification for the conclusion $A \sqsubseteq E$, but the concept in bold indicates that there are two different reasons for the conclusion, even though they are both necessary to derive the conclusion; (Horridge, 2011) defines this as *internal masking*. There is also *external masking*, where some axiom that is not necessary for deriving the conclusion – but still relevant in some sense – is being excluded from the justification.

For example, the ontology $\mathcal{O} = \{A \sqsubseteq \exists r.B,\ A \sqsubseteq \forall r.B,\ B \sqsubseteq C,\ D \equiv \exists r.C\}$ entails the statement $A \sqsubseteq D$. It has one justification: $\mathcal{J} = \{A \sqsubseteq \exists r.B,\ B \sqsubseteq C,\ D \equiv \exists r.C\}$, where the second axiom is left out. Changing the ontology $\mathcal{O}$ to $\mathcal{O}'$, where the first axiom is replaced by $A \sqsubseteq \exists r.\top$, i.e. the full existential qualification is replaced by a limited one, changes the (only) justification to $\mathcal{O}'$ itself, as now the second axiom is necessary to derive the conclusion. Both internal and external masking likely make understanding and repairing ontologies more difficult.

Another issue is how to specify a *repair*. Given an ontology $\mathcal{O}$ and a conclusion ϕ, a repair is a set of axioms $\mathcal{R}$ that intersects all justifications $\mathcal{J}_i$ for ϕ, such that $\mathcal{O}' = \mathcal{O} \setminus \mathcal{R}$ does not entail ϕ. Ideally, these repairs are 'minimal' so that the repaired ontology $\mathcal{O}'$ deviates as little as possible from $\mathcal{O}$. This minimality condition can be explicated in at least three different ways: justification minimality, cardinality minimality and semantic minimality (Horridge, 2011).

To develop techniques to deal with the above issues, the concepts of *precise* and *laconic* justifications are defined in (Horridge, 2011). Loosely speaking, laconic justifications are justifications that do not have superfluidity and precise

justifications are justifications designed to produce repairs.

Another way that is suggested to improve justifications is by introducing lemmas. Instead of ignoring irrelevant parts of certain axioms, multiple axioms get merged into simple expressions entailed by them. Informally, a set of lemmas Λ for a justification $\mathcal{J}$ for conclusion ϕ is a set of formulas, each entailed by $\mathcal{J}$, such that it can be used to replace a set $\mathcal{S} \subseteq \mathcal{J}$, i.e. $\mathcal{J}' = \mathcal{J} \setminus \mathcal{S} \cup \Lambda$ (called a *lemmatisation* of $\mathcal{J}$) is a justification for ϕ that is easier to understand than the original $\mathcal{J}$, according to some complexity measure (Horridge, 2011), (Horridge et al., 2009) and (Horridge et al., 2010). Formally defining lemmata is non-trivial, because certain boundary conditions need to be met. Without those boundary conditions the lemmatisations might bear only little relation to the original justification. For example, the justification $\mathcal{J} = \{A \sqsubseteq \exists r.B,\ B \sqsubseteq E \sqcap \exists s.C,\ B \sqsubseteq D \sqcap \forall s.\neg C\}$ for $A \subseteq \bot$ could yield the lemmatisation $\mathcal{J}' = \{A \subseteq E,\ A \subseteq \neg E\}$, because the concept A is unsatisfiable. But this justification might be not so useful in understanding why $A \subseteq \bot$ follows (Horridge et al., 2009).

This lemmatisation procedure forms the basis of *justification oriented proofs* (Horridge et al., 2010), where a proof can be build up from justifications by repeated lemmatisation. A related approach is finding *optimal interpolations* (Schlobach, 2004), where the simplest concept D is found to explain a given subsumption $C \sqsubseteq E$, such that both $C \sqsubseteq D$ and $D \sqsubseteq E$.

It is argued that these types of proofs strike a good balance between justifications on the one hand (which are easy to interpret, but from which the connection with the conclusion might be obscure) and proofs on the other hand (for which a proof system needs to be understood, but which allows for a more transparent presentation of the connection between the axioms and the conclusion).

2.2.4 Cognitive aspects

Making justifications more understandable by lemmatisation should incorporate some cognitive aspects of deductive reasoning. The measure that (Horridge et al., 2009) and (Horridge, 2011) use to compare the complexity of justifications is a certain weighted score of several syntactic and semantic aspects of the justification. It is somewhat ad hoc because it seems largely inspired by the researchers' intuition and not based on theories of human cognition, but the measure does agree quite well with the data. In (Horridge et al., 2010), however, it is pointed out that the proof-generating framework is agnostic with regards to the (cognitive) complexity measure, meaning that any complexity measure (possibly more

cognitively accurate) can be used. In (Horridge et al., 2013) several experiments are performed to test the cognitive complexity of reasoning about description logic subsumptions.

Empirical work in this direction is summarised in (Horridge, 2011). Some conclusions are:

1. there are naturally occurring justifications that are very difficult to understand, even for people who have experience with ontologies,
2. certain proof steps are not obvious and difficult to understand, e.g. trivial satisfaction of universal restrictions,
3. other patterns of reasoning are easy, e.g. chains of concept inclusions: from $A \sqsubseteq B$, $B \sqsubseteq C$, $C \sqsubseteq D$, $D \sqsubseteq E$, conclude $A \sqsubseteq E$,
4. superfluity non-trivially affects understanding, but needs to be investigated further.

A different focus is given in (Alrabbaa et al., 2022), which investigates different ways of presenting a given proof (as opposed to finding a different proof or transforming a proof to a new one).

1. short proofs are experienced as easier to understand than long proofs,
2. the way the proof is presented (proof tree vs. text) does not seem to affect objective understandability, though in some cases it affects subjective understandability,
3. experience in logic seems to affect proof understandability non-trivially and more research is needed.

In the literature, connections to cognitive theory seem rather unsystematic. A more systematic connection to cognitive psychological theory is made in (Warren, 2017), where three theoretical paradigms are used to explain various patterns occurring in description logic reasoning. These different theories, however, have their inherent problems and their usefulness as theoretical underpinnings for applied research seems limited.

We can conclude that there is a strong desire for proofs or justifications for description logic entailments that are optimised for ease of understanding for humans. Some empirical work has been done in this respect and some interesting conclusions have been established, but a coherent cognitive theoretical foundation seems lacking. Later in this thesis, the idea of cognitive architectures is proposed as a possibly more fruitful direction to apply cognitive theory to facilitating understanding of justifications and proofs.

3 Logic and cognition

(Marr, 1982, p.25), identifies at least three levels at which human reasoning can be studied:

1. the input-output level,
2. the level of the algorithm which causes this input-output behaviour,
3. the level of the neural implementation of this algorithm.

The first level is studied by performing experiments and leaves little room for disagreement or discussion. The second and third levels, however, are less easily studied and, not surprisingly, cause more disagreement and speculation. That these last two levels are interesting can be seen by observing the following experiment, taken from (Stenning and van Lambalgen, 2008) in which adult subjects are presented with the premises:

> If Julie has an essay, she studies late in the library.
> Julie does not study late in the library.

and they are asked what, if anything, can be derived from this.

Logic tells us that the statement

> Julie does not have an essay.

can be concluded by a rule that is referred to as *modus tollens*. Surprisingly, though, half of the subjects claim that nothing logically follows from the premises.

One might say that humans are just generally bad at deductive reasoning, but the picture becomes more interesting when the same subjects are asked the same question about the premises:

> If Julie has an essay, she studies late in the library.
> Julie has an essay.

In this case 95% of the subjects correctly conclude (by applying a rule called *modus ponens*):

> Julie studies late in the library.

This shows that humans are not always bad at deductive reasoning. This asymmetry in performance, which can be consistently experimentally confirmed, begs

the need to study deductive reasoning on levels deeper than the input-output level.

There are different 'schools' that study deductive reasoning at this algorithmic level with the aim of explanaining the phenomena at the input-output level. The two most important ones are the Mental Logic school and the Mental Models school.

3.1 Mental Logic

The *Mental Logic* or *Mental Rules* school is proposed and defended in (Rips, 1994). This school explains the asymmetry in deductive reasoning performance by claiming that deductions are performed by mentally applying rules of inference and that different inference rules have different probabilities to 'fire', i.e. to be applied during a reasoning process. Some deductions rely on applications of modus ponens and are easy because this rule has a high associated probability of firing. Other deductions rely on modus tollens and are more difficult because this rule either has a lower associated probability to fire, or needs to be itself derived in terms of other rules.

Because deductive reasoning bears many similarities to language processing, the school also makes claims about the level of neuro-anatomical implementations:

> ... deductive reasoning is a rule governed syntactic process where internal representations preserve structural properties of linguistic strings in which the premisses are stated. This linguistic hypothesis predicts that the neuro-anatomical mechanisms of language (syntactic) processes underwrite human reasoning processes... (Goel et al., 2000, p.504)

In (Rips, 1994) PSYCOP is set forth, which can be seen as a method to create models for human deductive reasoning. For propositional reasoning, for example, Rips includes 24 inference rules in the model, which, besides the regular natural deduction style rules, also involve more complex rules like DeMorgan rules. The rules come in pairs where one is a forward and the other a backward version of the deduction rule; the reason for this is that a subject can reason forwards by combining certain known statements and deriving others, or it can reason backwards by determining goals on what to derive next based on a putative conclusion. The set of rules incorporated in the system does not have to be minimal: some rules may be derivable from others. Each rule is assumed to have an associated probability of firing, i.e. the probability that the human actually uses the rule to produce conclusions in their mind.

To determine the firing probability of the inference rules, students (with no familiarity with formal logic) where given arguments like:

> If the light goes on or the piston expands, then the wheel turns.
> If the light goes on or the wheel turns, then the wheel turns.

and were asked whether this is a piece of valid reasoning. Some students correctly judged the argument valid, while others claimed it was invalid; this yielded an error rate for each argument. Using the probabilities of each inference rule, this and similar error rates can be estimated as follows.

> ...the model will prove [the argument above] using a combination of IF introduction, OR elimination and Disjunctive Modus Ponens. If these rules are available with probabilities p_1, p_2, and p_3, respectively, then (assuming independence) the probability of a correct 'necessarily true' response might be [...]:
>
> $$P = p_1 p_2 p_3 + 0.5 p_g (1 - p_1 p_2 p_3)$$

Here p_g is the probability that the subject makes a guess. The above expression is not completely correct, because the same argument could also be proved with other inference rules, namely IF elimination and OR introduction. If these rules fire with probabilities p_4 and p_5 respectively, the expression becomes:

$$P = p_1 p_2 p_3 + (1 - p_3) p_1 p_2 p_4 p_5 + 0.5 p_g (1 - p_1 p_2 p_3 - (1 - p_3) p_1 p_2 p_4 p_5),$$

where the first term is interpreted as the probability of finding a proof by using the three first rules, the second term is the probability of finding the proof using the other two rules and the third term is the probability of a correct guess. A limitation to the experiment is that no superfluous inference steps can be performed: all inference steps are necessary to reach the conclusion. It might be interesting to test PSYCOP in situations where some inference steps can be made which do not contribute to the conclusion and see how this affects the model's predictions.

The probabilities p_i for each inference rule are found by fitting expressions like the one above to the experimentally found error rates. The fit is a non-linear one without a clear linearization transformation available and is therefore probably executed numerically; not much info about the fit can be found in the source. The estimated error rates of PSYCOP for propositional logic correlate well with the empirical data, with a correlation of 93%. For PSYCOP applied to first-order logic, the correlation between the estimated and empirical values is 83%.

The resulting probabilities agree with intuitions, apart from a few exceptions: the NOT introduction rule has a rather low probability, OR Elimination and IF introduction have a relatively high probability. These might not be expected from the apparent level of difficulty of the rules.

Models thus made depend on which rules of the logic are used and with many rules, the number of parameters to fit the model to the data is large. This number is reduced somewhat by equating the probabilities of forward and backward inference rules, as well as by estimating the guess probability by taking it to be twice the error rate of the rate of 'necessarily true' guesses for the invalid arguments. This results in 10 parameters. These parameters are further restricted by the logical structure of the arguments, but it is a priori difficult to quantify how much this restriction prevents overfitting. A method to quantify the amount of overfitting would be to use a test set of data that is not in the training data, i.e. the data used in making the fit. The model, with the parameter settings it learned from the fit, is then used to predict the values in the test set, after which it can be determined how good this fit is. This procedure seems unfortunately not carried out in (Rips, 1994).

3.2 Mental Models

Another school of thought, advocated by Philip Johnson-Laird, is called the *mental models* school. This school claims that instead of applying syntactic rules to certain expression, the premises are used to construct a model in the mind. A conclusion is then a statement that 'fits' the model thus constructed. The school argues that many deviations from logic in human deductive reasoning seem caused by semantic effects; abstract rules, being devoid of any semantic content, seem unable to explain these effects. One example is (Johnson-Laird, 2010):

> All of the Frenchmen in the restaurant are gourmets.
> Some of the gourmets in the restaurant are wine-drinkers.
> What, if anything, follows?

From which most people infer that some of the Frenchmen in the restaurant are wine-drinkers. This, in fact, does not follow from the premises.

People tend to be more careful drawing conclusions when they are presented:

> All of the Frenchmen in the restaurant are gourmets.
> Some of the gourmets in the restaurant are Italians.
> What, if anything, follows?

which has the exact same logical structure as the argument above. Now, however, only a small number of people concludes that some of the Frenchmen are Italians.

Because the logical structure of the argument is exactly the same, the difference is caused by the semantic content of the statements. Effects like these cannot be explained by the Mental Logic school of human deductive reasoning, so it is argued. The Mental Models school argues that, based on people's background experiences, it is easy to make a model in which Frenchmen are wine-drinkers, but it is much harder to imagine Frenchmen who are also Italians, making the second inference less likely to be accepted.

Another prediction that the Mental Models school makes is that the more models are necessary in an inference, the more difficult the inference becomes.

For example, if human subjects are presented:

> Raphael is in Tacoma or else Julia is in Atlanta, bot not both.
> Julia is in Atlanta, or Paul is in Philadalphia, but not both.
> What follows?

it is generally hard to deduce that either Raphael is in Tacoma and Paul is in Philadelphia, or else Julia is in Atlanta. This is explained because the premises allow for two mental models each, which is considered a lot. It becomes even more difficult when the exclusive disjunctions from this example are replaced by inclusive disjunctions. The premises then allow three different models each, and the premises together allow for five models, thereby putting an even higher cognitive load on the subject's working memory. The percentage of accurate conclusions then drops from 20% to less then 5% for people from the general public (for students both percentages are higher: 75% and 30% respectively).

Although the Mental Models school gives excellent explanations for certain effects, its quantitative predictions are sometimes easy to refute. For example, the paper (Newstead et al., 1999a) reports on experiments that suggest people do not consider more than one model. In Bonatti (1994) further discrepancies are discussed between the mental model theory's predictions and empirical data.

Moreover, the mental model theory is claimed to be formally ill-defined. Hodges (1993) claims that the formal notation is unclear and that the mental models bear little relation to an actual deduction in case of modus ponens. Cohen (1993) states that the mental models theory fails to find easy counter-examples. Garnham (1993) mentions that there are syllogisms the number of associated models of which are not accounted for by the mental models theory.

3.3 Comparison

The structure of both theories seems to make the mental models theory better at *explaining* the experimental data on human deduction, but worse at *predicting* and for the mental logic theory the contrary holds. (O'Brien et al., 1994) seems to clearly favour the mental logic theory over the mental models, but the differences between the two schools might not be so great or might be at least vague, as some authors argued.

(Andrews, 1993) makes a comparison between the mental models theory and the tableau proof method and argues that the two can be conceptually difficult to distinguish. In (Braine, 1993) it is argued that mental models theory implicitly incorporates some mental logic theory and (Bundy, 1993) argues that the difference between the two might be a gradual one instead of a difference in kind. The examples in the latter are taken from the field of automated theorem proving and claim that resolution can be seen both as a rule-based proof method and as a model-based mechanism and that a theorem prover called SUMS displayed a gradual transition from a model- to a rule-based system.

It can be concluded from the above that both theories are sometimes difficult to distinguish, that they are in some aspects ill-defined and that it is difficult to derive falsifiable predictions from the theories.

3.4 Other Schools

There are also other schools that explain human deductive reasoning on the algorithmic level of abstraction. For example, there is the school referred to as 'Darwinian Algorithms' (Stenning and van Lambalgen, 2008, p.120). The school stands in the tradition of evolutionary psychology which claims that human intelligence consists of different modules that are designed by the process of natural selection. This school's predictions regarding the famous Wason selection task (Wason, 1968) are worth mentioning.

In the Wason selection task, people are presented four cards showing an 'A', a 'B', a '2' and a '3' respectively together with the rule 'If a card has an A on one side, then it has a 3 on the other side' (Cox and Griggs, 1982) It is also mentioned that every card has a number on one side and a letter on the other. The subjects were then asked to turn those and only those cards that are necessary to check if the given rule holds for the given cards. The correct answer is that one should check the card with the 'A' and the card with the '2'. Many people succeed in turning the card with the 'A', but fail to turn the card with the '2' and turn the

card with the '3' instead. One explanation of the Darwinian Algorithms school is that human performance on the Wason selection task is so low because it does not resemble typical situation.

The task can, however, be modified to one which is more realistic. The rule becomes 'If a person is drinking beer, then the person must be over 19' and the cards become people satisfying 'drinking a beer', 'drinking a coke', '16 years of age', and '22 years of age' respectively. In this version of the Wason selection task the performance is dramatically increased. The Darwinian Algorithms school explains this by regarding the given rule as a social contract, for which humans are evolved to perform well by natural selection, because social contracts were of essential importance in prehistoric tribal communities.

Another paradigm is the relational complexity is developed in (Andrews and Halford, 2002). The perspective the authors take is from cognitive development in children. As children develop they become capable of reasoning about increasingly complex relations between objects. The 'taller than' relation is a binary one which children can reason with from a young age. Later, more complex relations like '2 and 3 equals 5' become accessible to reason with. A connection with working memory is that the latter bounds the number of objects within a relation that a person can reason with. This relational complexity framework can be used to estimate the complexity of syllogistic reasoning problems. In (Zielinski et al., 2010), however, it is shown that the relational complexity paradigm does not make more accurate predictions about experimental data on human performance on syllogisms than the mental model theory does. In this article it is also mentioned that '[r]elational complexity theory is essentially a theory of mental models [...], but it conceptualises models in a different way than have previous mental model theories of reasoning.'(Zielinski et al., 2010, p.418), suggesting that also the conceptual difference between these two schools is small.

3.5 Interpretations

The Wason selection task and its variations illustrate how human performance on deductive reasoning is affected by non-logical content. Problems like this are perhaps deeper than one would expect. Not only may non-logical content improve or decrease performance on a deductive reasoning task, but also the meanings of the logical constants themselves are affected, an effect that is called *modulation*. A very thorough discussion of this can be found in chapter three of (Stenning and van Lambalgen, 2008). By means of interviews the researcher investigated how the implication connective of the rule in the Wason selection

task was understood in different scenarios and for different people. The classical interpretation of the conditional was often not used. Besides it being interpreted as the deontic implication (as above in the drinking age example) it was interpreted as having existential import and it was even interpreted as a conjunction.

Content-effects are considered outside of the scope of this thesis. In the following, therefore, there is a focus on *symbolic* reasoning, i.e. reasoning with syntactic expressions that follows abstract rules of inference.

3.6 Cognitive Architectures

Besides the various schools of deductive reasoning, a relevant area of research is the rise of cognitive architectures, which are integrative frameworks for modelling general human cognitive behaviour. To start, it is not completely clear what a cognitive architecture is (Kotseruba and Tsotsos, 2020). Many theories and softwares exist that can be considered cognitive architectures and among them they vary greatly in their research goals, structures, operations and applications, making them difficult to compare. Most cognitive architectures are inspired by Alan Newell's idea of an integrated theory of human cognitive behaviour (Newell, 1992). In an attempt to study the different cognitive architectures more coherently, Sun stated a number of desiderata for cognitive architectures in (Sun, 2004). The paper states that

> a cognitive architecture is the overall, essential structure and process of a domain-generic computational cognitive model, used for a broad, multiple-level, multiple-domain analysis of cognition and behavior.

This should broadly capture the intended intuitive idea. Even so, there are of course boundary cases that some researchers do and some do not consider to be cognitive architectures. For example Laird explicitly excludes GOMS and BDI from his list of cognitive architectures, although they are included by a survey by Samsonovich (Kotseruba and Tsotsos, 2020).

One of the first cognitive architectures is called ACT-R and is currently still being developed. It is also one of the most popular cognitive architectures in the literature (others being CLARION, Soar, EPIC and LIDA) and many models for various purposes are made with it (ACT, 2023a). It is this framework that is used for the rest of this thesis.

3.7 ACT-R

ACT-R started in the late seventies with the aim of being an intergrated theory of cognitive behaviour. The intention was to create not only a framework for models, but at the same time an explanative theory of human cognitive behaviour, for which both psychological and neurological research were used in building ACT-R. Good introductions can be found in (Anderson, 2007), (Whitehill, 2013) and (Anderson and Byrne, 2004).

The basic idea of the framework is that the brain has regions which are highly specialised in certain tasks and can perform these tasks in massively parallel fashion. In the theory of ACT-R, these regions are represented by *modules*, each of which has a *buffer* connecting the module to the *procedural memory*. The latter corresponds to the *basal ganglia* structure of the human brain. The buffers act as communication bottlenecks and can contain only a limited amount of information.

ACT-R has been applied in widely varying areas of cognition, such as:

- the *Stroop task*, where a certain response delay can be measured to incongruent stimuli, compared to congruent stimuli (Juvina and Taatgen, 2009),
- syntactic parsing of natural language (Brasoveanu and Dotlačil, 2020),
- complex cognitive tasks such as solving algebraic equations (Anderson, 2005),
- explaining brain region activity during certain cognitive tasks (Anderson et al., 2008),
- intelligent cognitive tutors that optimise retention of learning material (Lewis et al., 1987)

The third application above is relevant for this thesis, as later sections discuss an ACT-R model that simulates symbolic reasoning. The two tasks seem similar in the sense that both processes are instances of symbol manipulation.

Creating a model within this framework requires a few steps. Firstly, the *procedural knowledge* (knowledge that cannot be put into words, i.e. skills) that the model is assumed to have, needs to be codified into production rules for the procedural memory. Secondly, the *declarative knowledge* (knowledge that can be put into words, i.e facts) that the model is assumed to know, needs to be codified into chunks that are stored in ACT-R's declarative module. Lastly, the subsymbolic parameters are given specific values to make the behaviour of the model correspond to empirical data. What follows is not a complete overview of

the theory of ACT-R, but rather a selection of the topics that are most relevant to understand the rest of the thesis. Most of the following is based on the ACT-R tutorial (ACT, 2023b).

3.7.1 Knowledge representation

Knowledge in ACT-R is encoded in *chunks* and *productions*. A chunk is a collection of attribute-value pairs which represents factual knowledge that a person is expected to know when solving a particular problem. The attributes are called *slots*, each of which can contain at most one value. Each chunk also has a name making it easier to refer to it, but this has no other function from a modelling perspective. An example of a chunk is:

```
Fact 3+4
   addend1 three
   addend2 four
   sum seven
```

The production rules encode procedural knowledge. They are condition-action rules and are stored in the procedural module. An example is:

```
Rule Add
IF the goal is to add two digits d1 and d2
   and d1 + d2 = d3
THEN create a goal to write d3
```

3.7.2 The Goal and Imaginal modules

The *goal* and *imaginal* modules are simple modules that put a chunk in their associated buffer upon request. The purpose for the goal module is to keep track of the control information of the task at hand; it is used to plan the next step of the problem solving process. With brain scanning techniques such as fMRI measurements, this module can be associated to the brain region called the Anterior Cingulate Cortex (Anderson et al., 2008). The imaginal module has two associated buffers: the *imaginal* and the *imaginal-action* buffer. They are designed to store context-relevant information. Contrary to the goal buffer, processes in these buffers take time. The time is given by the imaginal-delay parameter, which can be set, but defaults at 0.2s; it may also have a randomised component. This module can be associated to the Posterior Parietal region (Anderson et al., 2008).

3.7.3 The Declarative module

The *declarative* module contains all the factual information of the system, though not all this information might be relevant for solving the problem. It can be related to the Lateral Inferior Prefrontal region in the brain (Anderson et al., 2008). The declarative memory (as the module is also called) contains all chunks the modeller placed there at the beginning of a simulation, as well as all chunks created during a simulation (in any buffer). It has one buffer, named the *retrieval* buffer which can store one chunk. The module responds to a retrieval request by searching through all chunks stored in the declarative memory for a match and places the matching chunk in the buffer. By default this process takes 50 milliseconds. In case no chunk is found, the buffer assumes the error state. The declarative module also allows for retrieving chunks which partially match the retireval request, but in the following we ignore and disable this mechanism.

3.7.4 The Procedural module

The *procedural* module does not have a buffer. It contains all the model's production rules and is continuously monitoring all the buffers' contents for matching the conditions of any production rule. In case of a match, the rule is *fired*, which means that the action of the rule is performed. Only one production can fire at a time and this takes 50 milliseconds. It can be related to the Caudate Nucleus in the brain (Anderson et al., 2008).

3.7.5 The Motor module

The *motor* module controls the movement of the hands and fingers, more specifically, finger presses on a keyboard. The module is designed to simulate typing behaviour of someone who can type 40 words per minute without looking, which is considered average. Any action in this module goes in stages where firstly the buffer makes a request to the module, then the module prepares the movement, executes it and returns to the initial state. During this last stage, the module is still busy, meaning that no new movement request can be made.

3.7.6 The Vision module

Te *vision* module will only be quickly mentioned, as it is not used in the following. The module has two buffers that store the location of visual stimuli and the information of that visual stimulus respectively. The module takes into account the distance between two points that the eyes consecutively focus on, where the

angle determines the duration of the refocussing process.

3.7.7 Base-level learning

Instead of the default retrieval time of 50 milliseconds, a more realistic approximation takes into account learning and forgetting effects. This process, in sum, is called *base-level learning*. Each chunk has a certain *activation*, a number that is determined by: the number of *presentations*, the time passed since each presentation, the *decay* parameter, which is usually set to 0.5 and a random component. A presentation of a chunk is the moment when the chunk is cleared from a buffer, or when it is stored in the declarative memory. With these ingredients, the formula to calculate a chunk's activation is:

$$A = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right) + \epsilon$$

The noise ϵ is a random number drawn from a logistic distribution with mean 0 and standard deviation $\sigma^2 = \frac{\pi^2}{3}s^2$, where s is the noise value which can be set by the modeller. In ACT-R there is a distinction between the permanent and the instantaneous noise, but in the following we disregard the former. To decrease the computational complexity of a model, optimized learning can be used as an easy to compute approximation to base-level learning. The exact time of each chunks' presentation is then ignored and only the number of presentations matters to calculate the chunk's activation with the formula:

$$A = \ln\left(\frac{n}{1-d}\right) - d \cdot \ln(L) + \epsilon,$$

Activations also depend on contexts (chunks which are similar), but in the following we disregard this mechanism.

Probability of recall

The activation allows ACT-R to calculate the retrieval probability of a chunk. This probability is zero if a chunk's activation drops below the retrieval threshold τ; the chunk is then not accessible anymore unless new presentations of the chunk occur (which increases its activation). The probability of recall is given by the formula:

$$P = \frac{1}{1 + \exp\left(\frac{\tau - A}{s}\right)}$$

The formula shows that higher activation entails a higher probability of recall. Likewise, higher values of τ lower the probability of recall. The parameter s models the sensitivity with which the probability changes from 0 to 1: large values of s make the change gradual while small values of s make the change more abrupt. We also see that if the decay parameter d has a large value, all chunks' activation drops very rapidly and the chunks can very quickly not be derived anymore.

Retrieval latency

The ACT-R theory assumes that memories that are more recently or more frequently stored in memory can be retrieved more quickly. Again this retrieval latency is based on the chunks' activation and is calculated by the formula:

$$T_i = Fe^{-A_i}.$$

Here, F is the latency factor, which can be used to fit data; its default value is 1. Higher activations allow chunks to be retrieved more quickly. Note that this formula only applies when a chunk is actually retrieved; in case the activation drops below the retrieval threshold the chunk is not retrieved but this still takes time. The time this takes, is given by:

$$T_i = Fe^{-\tau},$$

where again τ is the retrieval threshold.

Utilities

ACT-R's production rules have associated utilities that model the probability of selecting a rule in case more than one could fire. Each rule's utility can be set individually, after which it (in this case for production rule i) can be updated by the formula:

$$U_i(n) = U_i(n-1) + \alpha\Big(R_i(n) - U_i(n-1)\Big),$$

where $U_i(n-1)$ is the previous value of the rule's utility, α is the learning rate, typically set to 0.2 and $R_i(n)$ is a reward that the rule receives for its n-th application. The reward is interpreted as the time interval that the model is expected to save by applying the rule and the time between applying the rule and receiving the reward is subtracted from the reward. For example, if rule i fires for the n-th time at time t_0 and at time t_1 the model receives a reward R, then $R_i(n) = R - (t_1 - t_0)$.

The utilities also have a random noise component, similarly calculated as for the activations by a logistic distribution with mean 0 and standard deviation $\sigma^2 = \frac{\pi^2}{3}s^2$.

With the utilities thus defined, the probability of firing production rule i could be calculated as:

$$P(i) = \frac{\exp\left(\frac{U_i}{\sqrt{2}s}\right)}{\Sigma_j \exp\left(\frac{U_i}{\sqrt{2}s}\right)}$$

This calculation is not performed by the system during simulations; only the production rule with the highest utility fires.

Compilation

Production rules also allow for learning at the symbolic level. Two mechanisms in this respect can be distinguished. Firstly, there is *proceduralization*, where a specific instance of a production rule is memorised. Secondly, there is production rule *composition* where two rules combine to make one rule with the same effect as the two separate rules.

As an example of proceduralization, if we have the production rule:

```
Rule i
IF
   goal add
   buffer1 x
   buffer2 5
THEN
   write x + 5.
```

and in buffer1 the chunk:

```
Chunk j
   number 6.
```

is stored, then a new rule, namely:

```
Rule i
IF
```

```
      goal add
      buffer1 6
      buffer2 5
   THEN
      write 11
```

is stored in the procedural memory. By having one rule that directly gives the model the desired answer, the model is able to perform the task directly (and hence more quickly) the next time.

As an example of production rule composition, if the two rules:

```
   Rule i
   IF
      goal1 add
      buffer1 x
      buffer2 5
   THEN
      goal2 add
      buffer1 x + 5
```

and

```
   Rule j
   IF
      goal2 add
      buffer1 y
      buffer2 2
   THEN
      goal3 add
      write y + 2
```

fire after one another, ACT-R will (if production composition is enabled) store the rule:

```
   Rule j
   IF
      goal1 add
      buffer1 x
```

```
    buffer2 5
THEN
    goal3 add
    write x + 7
```

in its procedural memory. This rule is a composition of the previous two rules: it has the conditions of the first and the actions of the second. Possible actions of the first rule that are not negated by the second rule will also be put in the newly created rule, although the above is not an example of that. By having one rule instead of two, the model is again able to perform the task more quickly.

4 $\mathcal{ALE}$ ABox inconsistency

This chapter discusses first the complexity of the $\mathcal{ALE}$ ABox inconsistency problem. Then an abstract tableau style algorithm of this problem is described and its complexity is determined. In the next chapter, this algorithm's implementation (SHARP) into the cognitive architecture ACT-R is described.

4.1 The $\mathcal{ALE}$ ABox inconsistency problem

To appreciate the effectiveness of the tableau algorithm and its implementation in the following two sections, it is important to first determine the complexity of the $\mathcal{ALE}$ ABox inconsistency problem. The complexity of this problem was exactly determined in 1991, with (Schmidt-Schauß and Smolka, 1991) proving that the problem is solvable in **NP** by looking at so-called traces, then later in (Donini et al., 1991) it was proved that the problem is actually **NP**-complete by reducing it to the set-traversal problem. In (Donini et al., 1994) an overview is given of similar results for logics closely related to $\mathcal{ALE}$.

First, we will present the proof that the problem is in **NP** taken from (Baader et al., 2017) and then we present the proof that it is **NP**-hard based on (Donini et al., 1991), establishing that it is **NP**-complete. To describe what follows effectively, we need the following definitions, most of them taken from (Baader et al., 2017).

Definition 4.1.1. (Baader et al., 2017, p.58)**.** Given an $\mathcal{ALE}$ concept C, its *size* $\mathsf{size}(C)$ and set of *subconcepts* $\mathsf{sub}(C)$ are defined by recursion on the structure of C:

- If C is a concept name, then $\mathsf{size}(C) = 1$ and $\mathsf{sub}(C) = \{C\}$,
- If $C = C_1 \sqcap C_2$, then $\mathsf{size}(C) = 1 + \mathsf{size}(C_1) + \mathsf{size}(C_2)$ and $\mathsf{sub}(C) = \{C\} \cup \mathsf{sub}(C_1) \cup \mathsf{sub}(C_2)$,
- If $C = \neg D$ or $C = \exists r.D$ or $C = \forall r.D$, then $\mathsf{size}(C) = 1 + \mathsf{size}(D)$ and $\mathsf{sub}(C) = \{C\} \cup \mathsf{sub}(D)$.

These definitions extend to an ABox $\mathcal{A}$ as follows: $\mathsf{sub}(\mathcal{A}) = \bigcup_{a:C \in \mathcal{A}} \mathsf{sub}(C)$ and $\mathsf{size}(\mathcal{A}) = \sum_{a:C \in \mathcal{A}} \mathsf{size}(C) + R_{\mathcal{A}}$. $R_{\mathcal{A}}$ is the number of role formulas (i.e. formulas of the form $(a, b) : r$) in $\mathcal{A}$. In this text RBox formulas are included in ABoxes.

From the above we can readily deduce the following lemmas.

Lemma 4.1.1. (Baader et al., 2017, p.58)**.** For any $\mathcal{ALE}$ concept C

$$|\mathsf{sub}(C)| \leq \mathsf{size}(C).$$

Lemma 4.1.2. (Baader et al., 2017, p.58)**.** For each $\mathcal{ALE}$ ABox $\mathcal{A}$,

$$|\mathsf{sub}(\mathcal{A})| \leq \mathsf{size}(\mathcal{A}).$$

Other useful definitions are:

Definition 4.1.2. (Baader et al., 2017, p.58)**.** Given an $\mathcal{ALE}$ ABox $\mathcal{A}$ and an element name a. The set of concepts of a is defined by:

$$\mathsf{cpt}_{\mathcal{A}}(a) = \{C \mid a : C \in \mathcal{A}\}.$$

Which gives rise to:

Lemma 4.1.3. (Baader et al., 2017, p.58)**.** For any $\mathcal{ALE}$ ABox $\mathcal{A}$ and element name a:

$$|\mathsf{cpt}_{\mathcal{A}}(a)| \leq \mathsf{size}(\mathcal{A}).$$

Definition 4.1.3. (Baader et al., 2017, p.77)**.** For any $\mathcal{ALE}$-formula $(a, b) : r$ in an ABox $\mathcal{A}$, we say that b is an *r-successor* of a in $\mathcal{A}$ and a is an *r-predecessor* of b in $\mathcal{A}$. A *successor* is an r-successor in $\mathcal{A}$ for some r. Likewise, a *predecessor* is an r-predecessor in $\mathcal{A}$ for some r.

Definition 4.1.4. A finite sequence of individual names $a_0, \ldots, a_n$ is a *chain* in $\mathcal{A}$ if every $a_i \in \mathcal{A}$ with $1 \leq i \leq n$ is a successor of $a_{i-1} \in \mathcal{A}$. The number $n + 1$ is the *length* of the chain.

4.1.1 NP-solvability

The following is based on (Schmidt-Schauß and Smolka, 1991) and (Baader et al., 2017). To decide if a given $\mathcal{ALE}$ ABox is inconsistent, it is enough to look at *traces*. These can be computed and checked in polynomial time by the following *trace-completion rules*:

$\sqcap$-trace-rule: if $a : C \sqcap D \in \mathcal{A}$ and
$\{a : C,\ a : D\} \notin \mathcal{A}$,
then $\mathcal{A} \to \mathcal{A} \cup \{a : C,\ a : D\}$.
$\exists$-trace-rule: if $a : \exists r.C \in \mathcal{A}$ and
there is no b such that $\{(a, b) : r'\} \subseteq \mathcal{A}$ for any r',
then $\mathcal{A} \to \mathcal{A} \cup \{(a, d) : r,\ d : C\}$, where d is new in $\mathcal{A}$.
$\forall$-trace-rule: if $\{a : \forall r.C,\ (a, b) : r\} \subseteq \mathcal{A}$ and
$b : C \notin \mathcal{A}$,
then $\mathcal{A} \to \mathcal{A} \cup \{b : C\}$.

The *trace-completion* rules for $\mathcal{ALE}$ ABox inconsistency. After: (Schmidt-Schauß and Smolka, 1991). Note that in the condition for the $\exists$-trace-rule a does not have any successor.

Applying a trace-completion rule to an ABox $\mathcal{A}_i$ yields a new ABox $\mathcal{A}_{i+1}$. Repeatedly applying trace-completion rules to an ABox $\mathcal{A}$ creates a *completion sequence* $(\mathcal{A}_i)$, where every $\mathcal{A}_{i+1}$ is created by applying some trace-completion rule to $\mathcal{A}_i$. Each ABox in a completion sequence $(\mathcal{A}_i)$ is said to *correspond* to $\mathcal{A}_0$. We call an ABox a *trace* when no trace-completion rules can be applied to it. A completion sequence that contains a trace is called a *perfect sequence*.

In the next section, we introduce the *syntax expansion rules*, which are very similar to the trace-completion rules, except that the condition in the $\exists$-rule (which is the analog of the $\exists$-trace-rule) is weaker.

We now establish that a completion sequence can always be extended to a perfect sequence so that it contains a trace; moreover, this can be done in polynomial time.

Lemma 4.1.4. (Schmidt-Schauß and Smolka, 1991)**.** Let $\mathcal{A}$ be an $\mathcal{ALE}$ ABox and let $(\mathcal{A}_i)$ be a corresponding perfect sequence. The length of $(\mathcal{A}_i)$ is polynomially bounded by $\mathsf{size}(\mathcal{A})$ and the last element of the completion sequence is a trace.

Proof. We will show that the length of the sequence $(\mathcal{A}_i)$ is polynomially bounded by $m = \mathsf{size}(\mathcal{A})$. First we note that the role formulas in $\mathcal{A}$ define a labelled directed graph. The vertices of this graph are formed by the elements and the edges are defined by the role formulas with the role name labelling the edge. By applying the trace-rules, we construct a model that consists of trees, the roots of which form the labelled directed graph. The $\exists$-trace-rule is the only rule that

can add new elements to the model, and it adds precisely one element for every individual name it is applied to, therefore every tree consists of one branch only.

The depth of these branches is bounded by m, for let x be an individual that is introduced by the $\exists$-trace-rule with a as its predecessor. Now $\mathsf{sub}(\mathsf{cpt}_{\mathcal{A}_i}(x)) \subsetneq \mathsf{sub}(\mathsf{cpt}_{\mathcal{A}_i}(a))$ for any i, so the length of the branches is linearly bounded by m.

None of the expansion rules remove a formula from the ABox, but they do add a formula $a : C$ for some element name a and concept expression $C \in \mathsf{sub}(\mathcal{A})$. We know by Lemma 4.1.3 that $\mathsf{sub}(\mathcal{A}) \leq m$, so for any individual x, we have $\mathsf{cpt}_{\mathcal{A}_i}(x) \leq m$ for any i.

So with the number of individual names bounded by m, and the number of elements in each branch bounded by m and the number of concepts that each element satisfies bounded by m, we see that the number of trace-rule applications is bounded by m^3.

When no more applications of the trace-rules can be made, the ABox under consideration is the last element of the completion sequence and is a trace by definition. □

Lemma 4.1.5. (Schmidt-Schauß and Smolka, 1991)**.** Let $\mathcal{A}$ be an $\mathcal{ALE}$ ABox. If there is a trace $\mathcal{A}'$ corresponding to $\mathcal{A}$ that contains a clash, $\mathcal{A}$ is inconsistent.

Proof. The trace-completion rules preserve consistency, for let $\mathcal{I}$ be an interpretation of ABox $\mathcal{A}_i$, now:

- The $\sqcap$-trace-rule. If $a : C \sqcap D \in \mathcal{A}_i$, then $a^{\mathcal{I}} \in (C \sqcap D)^{\mathcal{I}}$. So we have both $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $a^{\mathcal{I}} \in D^{\mathcal{I}}$. So $\mathcal{I}$ is also a model of $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{a : C,\ a : D\}$.
- The $\exists$-trace-rule. If $a : \exists r.C \in \mathcal{A}_i$, then $a^{\mathcal{I}} \in (\exists r.C)^{\mathcal{I}}$. By definition, there exists a $b \in \Delta^{\mathcal{I}}$ with both $(a^{\mathcal{I}}, b) \in r^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}$. Hence, for a new individual name x, $\mathcal{I}$ is also a model of $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{(a, x) : r,\ x : C\}$, by choosing $x^{\mathcal{I}} = b$.
- The $\forall$-trace-rule. If $\{a : \forall r.C,\ (a, b) : r\} \subseteq \mathcal{A}_i$, then $a^{\mathcal{I}} \in (\forall r.C)^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ and $b^{\mathcal{I}} \in C^{\mathcal{I}}$. This means that $\mathcal{I}$ is also a model of $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{b : C\}$.

Hence, consistency is preserved, so if a trace $\mathcal{A}'$ corresponding to $\mathcal{A}$ contains a clash, $\mathcal{A}$ is inconsistent. □

Lemma 4.1.6. (Schmidt-Schauß and Smolka, 1991)**.** Let $\mathcal{A}$ be an $\mathcal{ALE}$ ABox. If $\mathcal{A}$ is inconsistent, there is a corresponding trace $\mathcal{A}'$ which has a clash.

Proof. From a clash-free trace $\mathcal{A}'$, we can now construct a model $\mathcal{I}$ for $\mathcal{A}$. We

first define:

$$\begin{aligned}
\Delta^{\mathcal{I}} &= \{a \mid a : C \in \mathcal{A}' \text{ or } (a,b) : r \in \mathcal{A}' \text{ or } (b,a) : r \in \mathcal{A}'\},\\
a^{\mathcal{I}} &= a \text{ for each individual name } a \text{ occurring in } \mathcal{A}',\\
A^{\mathcal{I}} &= \{a \mid a : A \in \mathcal{A}'\} \text{ for each concept name } A \text{ occurring in } \mathcal{A}',\\
r^{\mathcal{I}} &= \{(a,b) \mid (a,b) : r \in \mathcal{A}'\} \text{ for each role name } r \text{ occurring in } \mathcal{A}'.
\end{aligned}$$

By construction, $\mathcal{I}$ is an interpretation. We now need to prove the following statement:

$$\text{if } a : C \in \mathcal{A}', \text{ then } a^{\mathcal{I}} \in C^{\mathcal{I}}.$$

The proof proceeds by induction on the structure of concepts. As for the induction basis: if C is a concept name and $a : C \in \mathcal{A}'$, then $a^{\mathcal{I}} \in C^{\mathcal{I}}$ by definition of $\mathcal{I}$.

The induction step consists of the following cases:

- $C = \neg D$. Because $\mathcal{A}'$ has no clash, $a : \neg D \in \mathcal{A}'$ implies that $a : D \notin \mathcal{A}'$, where D is atomic. So $a^{\mathcal{I}} \notin D^{\mathcal{I}}$ and hence $a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \setminus D^{\mathcal{I}} = C^{\mathcal{I}}$.
- $C = D \sqcap E$. For every formula $a : D \sqcap E \in \mathcal{A}'$, because $\mathcal{A}'$ is a trace, we must have both $a : D \in \mathcal{A}'$ and $a : E \in \mathcal{A}'$. So, by definition of $\mathcal{I}$, we have both $a^{\mathcal{I}} \in D^{\mathcal{I}}$ and $a^{\mathcal{I}} \in E^{\mathcal{I}}$. Hence we conclude $a^{\mathcal{I}} \in D^{\mathcal{I}} \cap E^{\mathcal{I}} = (D \sqcap E)^{\mathcal{I}}$, as desired.
- $C = \forall r.D$. Let $a : \forall r.D \in \mathcal{A}'$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. By the latter and how $\mathcal{I}$ is defined, we know that $(a,b) : r \in \mathcal{A}'$. We must have $b : D \in \mathcal{A}'$, because $\mathcal{A}'$ is a trace. D is of lower complexity than $\forall r.D$, so we may use the induction hypothesis to deduce that $b^{\mathcal{I}} \in D^{\mathcal{I}}$. This holds for all b such that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$, so $a^{\mathcal{I}} \in (\forall r.D)^{\mathcal{I}}$.
- $C = \exists r.D$. Let $a : \exists r.D \in \mathcal{A}'$. We have both $(a,b) : r \in \mathcal{A}'$ and $b : D \in \mathcal{A}'$ for some individual name b, because $\mathcal{A}'$ is a trace. The first expression guarantees that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ by construction of $\mathcal{I}$. For the other, we can use the induction hypothesis (D has lower complexity than $\exists r.D$) to conclude that $b^{\mathcal{I}} \in D^{\mathcal{I}}$. Hence, $a^{\mathcal{I}} \in (\exists r.D)^{\mathcal{I}}$.

This proves that all concept assertions are satisfied by the interpretation, so $\mathcal{I}$ is a model for $\mathcal{A}'$. So $\mathcal{I}$ is also a model of $\mathcal{A}$, because $\mathcal{A} \subseteq \mathcal{A}'$. □

Theorem 4.1.7. The problem of ABox inconsistency is in **NP**, (Schmidt-Schauß and Smolka, 1991).

Proof. To determine the inconsistency of an $\mathcal{ALE}$ ABox $\mathcal{A}$, we only need to look at the traces, because $\mathcal{A}$ is inconsistent if and only if there is a trace $\mathcal{A}'$ corresponding to $\mathcal{A}$ which has a clash. The traces individually are checked in polynomial time relating to the size of the ABox $\mathsf{size}(\mathcal{A})$, by using the trace-rules. If the $\exists$-trace-rule can be applied to a formula in $\{a : \exists r_1.C_1, \ldots, a : \exists r_n.C_n\}$, it selects one nondeterministically and in doing so, disallows the $\exists$-trace-rule to be applied to any of the other formulas. Applying the $\exists$-trace-rule to a different formula therefore creates a different trace. In short, the nondeterministic selection yields a unique trace, one such trace is enough to prove inconsistency and every trace is computed in polynomial time, therefore the problem of ABox inconsistency is in **NP**. □

4.1.2 NP-hardness

In this section we make a reduction from the *set traversal problem* (which is proved to be **NP**-complete) to the problem of $\mathcal{ALE}$ ABox inconsistency, thereby proving that $\mathcal{ALE}$ ABox inconsistency is **NP**-hard. The following proof is taken from (Donini et al., 1991).

A *positive clause* is a finite set of positive integers. Let $\mathcal{M} = \{M_1, \ldots, M_m\}$ be a set of positive clauses. A *traversal* of $\mathcal{M}$ is a finite set N of positive integers such that for all $i \in \{1, \ldots, m\}$, $N \cap M_i$ is a singleton. The *set traversal problem* is deciding whether a given finite set of positive clauses has a traversal. This problem is known to be **NP**-complete (Donini et al., 1991).

Now we define a polynomial reduction from the set traversal problem to $\mathcal{ALE}$ ABox inconsistency. The idea is to construct a concept $C_{\mathcal{M}}$ from each set of positive clauses $\mathcal{M}$ such that $\mathcal{M}$ has a traversal if and only if $C_{\mathcal{M}}$ is inconsistent. Let $n = \max(\bigcup \mathcal{M})$ and let $m = |\mathcal{M}|$ be the number of positive clauses in $\mathcal{M}$.

We first define $C^j_{2m+1} = \top$ and $D_{2m+1} = \bot$ and the strings $C^j_k = Q^j_k r. \cdots Q^j_{2m} r.C^j_{2m+1}$ and $D^j_1 = (\forall r.)^{2m} T$, where the last expression is a concatenation of $2m$ many universal restrictions. The Qs are defined as:

$$Q^j_l = \begin{cases} \exists & \text{if } j \in M_l \text{ or } j \in M_{l-m} \\ \forall & \text{if } j \notin M_l \text{ or } j \notin M_{l-m} \end{cases}$$

For example, if $\mathcal{M} = \{\{1, 3, 5\}, \{2, 4\}, \{4, 5\}\}$, then $C^1_1 = \exists r.\forall r.\forall r.\exists r.\forall r.\forall r.\top$.

Note that, in a sense, the index l passes over the clauses in $\mathcal{M}$ twice; this construction is later used to prove that the intersection of the traversal set with any of the clauses has only one element. $\mathcal{M}$ is translated to the concept: $C_{\mathcal{M}} = C^1_1 \sqcap \ldots \sqcap C^n_1 \sqcap D_1$.

Following (Donini et al., 1991), we define a concept C to be *active* in a trace T if and only if it is of the form $\exists r.D$ and there are variables y, z such that T contains the formulas $y : C$, $(y, z) : r$ and $z : D$. In other words, a concept $\exists r.D$ is active if the $\exists$-trace-rule has been applied to a formula of the form $y : \exists r.D$.

Lemma 4.1.8. (Donini et al., 1991)**.** Let T be a trace of $\{x : C_{\mathcal{M}}\}$.

1. Suppose C_k^j is active in T. Then for all $l \in \{1, \dots, k\}$ the concept C_l^j is active in T if it is of the form $\exists r.C_{l+1}^j$.
2. If T contains a clash, then for every $l \in \{1, \dots, 2m\}$ there exists exactly one j such that C_l^j is active in T.

Proof. We are proving 1. by induction on k. For $k = 1$ the statement trivially holds. Suppose the statement holds for a certain k, we want to prove that the statement holds for $k + 1$. We assume that $y_{k+1} : C_{k+1}^j \in T$. If $C_k^j = \exists r.C_{k+1}^j$, then the expression $y_{k+1} : C_{k+1}^j$ was introduced by the $\exists$-trace-rule. This means that both $y_k : C_k^j \in T$ and $(y_k,\ y_{k+1}) : r \in T$ for some variable y_k. Using the induction hypothesis, we know that $y_l : C_l^j \in T$ and $(y_l,\ y_{l+1}) : r \in T$ for all $l \in \{1,\ \dots,\ k-1\}$ as well. So we conclude that $y_l : C_l^j \in T$ and $(y_l,\ y_{l+1}) : r \in T$ for all $l \in \{1,\ \dots,\ k\}$.

For 2. we first note, that because the $\exists$-trace-rule applies to only one expression of the form $y : \exists r.C$, there can be at most one j such that C_l^j is active in T for a given l. We thus need to show that there is at least one such j. Assume on the contrary that for some $l \in \{1, \dots, 2m\}$ there is no j with C_l^j active in T. Then for every $k \in \{l + 1, \dots, 2m + 1\}$ there is no constraint in T of the form $y : C_k^i$ or $y : D_k$. Therefore the only possible clash $y : D_{2m+1} \notin T$, contradicting our assumption. □

Theorem 4.1.9. (Donini et al., 1991)**.** A set $\mathcal{M}$ of positive clauses has a traversal if and only if $C_{\mathcal{M}}$ is inconsistent.

Proof. Suppose $\mathcal{M} = \{M_1, \dots, M_m\}$ and $C_{\mathcal{M}} = C_1^1 \sqcap \dots \sqcap C_1^n \sqcap D_1$ is its translation.

For the 'only if' direction, let N be a traversal of $\mathcal{M}$. We will show that the ABox $\{x_1 : C_{\mathcal{M}}\}$ has a corresponding trace T that contains a clash by constructing a completion sequence with a slightly modified $\exists$-trace-rule.

Since N is a traversal, there is a function $f_N : \{1, \dots, 2m\} \to \{1, \dots, n\},\ l \mapsto k$ such that $C_l^k = \exists r.C_{l+1}^k$. Note that for all $j \in N \setminus \{k\}$, $C_l^j = \forall r.C_{l+1}^j$. The modified $\exists$-trace-rule is now:

- if $x_i : \exists r.C_l^{f_N(l)} \in \mathcal{A}$ and there is no b such that $\{(a, b) : r'\} \subseteq \mathcal{A}$ for some r', then $\mathcal{A} \to \mathcal{A} \cup \left\{(x_i, x_{i+1}) : r,\ x_{i+1} : C_l^{f_N(l)}\right\}$, where x_{i+1} is new in $\mathcal{A}$.

This modified rule is a restriction on the regular ∃-trace-rule in that it selects the formula $a : \exists r.C_l^{f_N(l)} \in \mathcal{A}$ by using the traversal, as opposed to selecting one nondeterministically.

Let T be the trace resulting from $\{x_1 : C_{\mathcal{M}}\}$ after applying the modified trace rules. Now $x_{2m+1} : D_{2m+1} \in T$ and because $D_{2m+1} = \bot$, T contains a clash.

For the converse, if $C_{\mathcal{M}}$ is inconsistent, then there exists a trace T corresponding to $\{x : C_{\mathcal{M}}\}$ that contains a clash and for every $l \in 1, \ldots, 2m$ there exists precisely one j such that C_l^j is active in T. We now construct a traversal N from the trace T, by defining: $N = \left\{ j \mid C_{m+l}^j \text{ is active in } T \text{ for some } l \in \{1, \ldots, m\} \right\}$.

To prove N is a traversal, let M_l be a positive clause in $\mathcal{M}$. For every $l \in \{1, \ldots, m\}$ there exists a j such that C_{m+l}^j is active in T, so $j \in N$. Moreover, $C_{m+l}^j = \exists r.C_{m+l+1}^j$ because the concept is active. That means that $j \in M_l$, because that is how $C_{\mathcal{M}}$ is defined. So $j \in N \cap M_l$. Now we need to prove uniqueness, so let $i, j \in N \cap M_l$. By definition of N there are h, k such that C_{m+h}^i and C_{m+k}^j are active in T. Furthermore, $C_l^i = \exists r.C_{l+1}^i$ and $C_l^j = \exists r.C_{l+1}^j$, because $i, j \in M_l$. By 1. in the previous lemma, we know that both C_l^i and C_l^j are active in T, because C_{m+h}^i and C_{m+k}^j are active in T. So by 2. of the previous lemma: $i = j$. □

Theorem 4.1.10. (Donini et al., 1991)**.** $\mathcal{ALE}$ ABox inconsistency is **NP**-complete.

Proof. We saw earlier that there is an **NP**-algorithm for the problem.

The previous theorem stated that a set of positive clauses $\mathcal{M}$ can be translated into an $\mathcal{ALE}$ concept $C_{\mathcal{M}}$ in polynomial time such that $\mathcal{M}$ has a traversal if and only if $C_{\mathcal{M}}$ is inconsistent. This polynomial time reduction from the set traversal problem to the problem of $\mathcal{ALE}$ ABox inconsistency and the fact that the set traversal problem is proved to be **NP**-complete, proves that $\mathcal{ALE}$ ABox inconsistency is **NP**-hard. Therefore, $\mathcal{ALE}$ ABox inconsistency is **NP**-complete. □

4.2 A tableau algorithm for $\mathcal{ALE}$ ABox inconsistency

The following tableau algorithm is based on (Baader et al., 2017, pp. 71 - 82), with the difference that the ⊔-rule is absent, because $\mathcal{ALE}$ lacks the concept unions from $\mathcal{ALC}$.

$\sqcap$-rule: if $a : C \sqcap D \in \mathcal{A}$ and
$\{a : C,\ a : D\} \notin \mathcal{A}$,
then $\mathcal{A} \to \mathcal{A} \cup \{a : C,\ a : D\}$.
$\exists$-rule: if $a : \exists r.C \in \mathcal{A}$ and
there is no b such that $\{(a, b) : r,\ b : C\} \subseteq \mathcal{A}$,
then $\mathcal{A} \to \mathcal{A} \cup \{(a, d) : r,\ d : C\}$, where d is new in $\mathcal{A}$.
$\forall$-rule: if $\{a : \forall r.C,\ (a, b) : r\} \subseteq \mathcal{A}$ and
$b : C \notin \mathcal{A}$,
then $\mathcal{A} \to \mathcal{A} \cup \{b : C\}$.

The syntax expansion rules for $\mathcal{ALE}$ ABox inconsistency. After: (Baader et al., 2017). Note that in the condition for the $\exists$-trace-rule, a does not have an r-successor (although it might have an r'-successor for $r' \neq r$).

Above we see the *syntax expansion rules* for $\mathcal{ALE}$ ABox inconsistency. They are very similar to the trace-completion rules from the previous section, only the $\exists$-rule differs from the $\exists$-trace-rule in that it has a weaker condition on the element name a: the $\exists$-rule requires that it has no r-successor, whereas the $\exists$-trace-rule requires that it has no successor at all, i.e. not for any r'. This means that the algorithm using the syntax expansion rules computes the union of all traces instead of just one trace.

The algorithm works by applying these rules to an ABox $\mathcal{A}$ until it is *complete*. Completeness here means that either no expansion rule can be applied, or that $\mathcal{A}$ contains a *clash*, i.e. for some individual name a and concept name C, $\{a : C,\ a : \neg C\} \subseteq \mathcal{A}$. Note that we allow role formulas in our ABox, contrary to the convention that role formulas form an RBox instead.

To describe the algorithm, we define a function $\mathsf{exp}(\mathcal{A},\ R,\ \alpha)$ which takes an ABox, an expansion rule and either one or two formulas and returns the ABox which is the result of applying the rule to the formula(s). An example:

$$\mathsf{exp}(\{b : \neg D,\ a : \forall r.D,\ (a, b) : r\},\ \forall\text{-rule},\ (a : \forall r.D,\ (a, b) : r)) = \{b : \neg D,\ a : \forall r.D,\ (a, b) : r,\ b : D\}$$

Algorithm inconsistent()
Input: an $\mathcal{ALE}$ ABox $\mathcal{A}$.
repeat
  if $\mathcal{A}$ contains a clash **then**
    return "inconsistent"
  end if
  select a rule R that is applicable to $\mathcal{A}$, and a (pair of) formula(s) $\alpha \in \mathcal{A}$ to which R is applicable
  set $\mathcal{A} \rightarrow \mathsf{exp}(\mathcal{A}, R, \alpha)$
until $\mathcal{A}$ is complete
if $\mathcal{A}$ contains a clash **then**
  return "inconsistent"
else
  return $\mathcal{A}$
end if

The tableau algorithm inconsistent for $\mathcal{ALE}$ ABox inconsistency. After: (Baader et al., 2017, p.75).

inconsistent() is a decision procedure for $\mathcal{ALE}$ ABox inconsistency, which we shall prove below. It returns either a complete ABox (in case it contains no clash) or "inconsistent" (in case it finds a clash at some point in the process). Note that the algorithm does not specify which rule to select; any applicable rule might be selected. Also to be noted is that it looks once more for a clash when $\mathcal{A}$ is complete, as the last expansion rule application might have created one.

The following lemmas are analogous to (Baader et al., 2017) and to the previous section.

Lemma 4.2.1. (Baader et al., 2017, p.78)**.** For each nonempty $\mathcal{ALE}$ ABox $\mathcal{A}$, $\mathsf{inconsistent}(\mathcal{A})$ terminates.

Proof. Let $m = \mathsf{size}(\mathcal{A})$ and let $\mathcal{A}'$ be an ABox obtained by applying some expansion rules some number of times to $\mathcal{A}$. We have the following three properties:

1. The expansion rules never remove a formula from $\mathcal{A}$ but add a new formula $a:C$ for some element name a and concept $C \in \mathsf{sub}(\mathcal{A})$. We know by Lemma 4.1.3 that the cardinality of $\mathsf{sub}(\mathcal{A})$ is bounded by $\mathsf{size}(\mathcal{A}) = m$. So for any individual name a we have $|\mathsf{cpt}_{\mathcal{A}'}(a)| \leq m$.

2. For a given individual name a and formula $a : \exists r.C$, the $\exists$-rule adds one and only one new element to $\mathcal{A}'$. The number of formulas that are existential restrictions is bounded by m, so there are at most m successors in $\mathcal{A}'$ for a given individual name a.
3. If b is a new individual name which is added by an application of an $\exists$-rule and has a as its predecessor, we have $\mathsf{sub}(\mathsf{cpt}_{\mathcal{A}'}(b)) \subsetneq \mathsf{sub}(\mathsf{cpt}_{\mathcal{A}'}(a))$. Moreover, the number of formulas of the form $(a, b) : r \in \mathcal{A}$ is bounded by m. So from both cases we see that the length of a chain in $\mathcal{A}$ is bounded by $2m$.

The above properties guarantee the following. The element names form a tree under the successor relation which has branching bounded by m and depth bounded by $2m$. This tree therefore has at most m^{2m} elements. For each element, the number of concept assertions to add is bounded by m, so the algorithm terminates after at most $m^{m^{2m}}$ steps. □

Lemma 4.2.2. (Baader et al., 2017, p.79)**.** If $\mathcal{A}$ is inconsistent, then $\mathsf{inconsistent}(\mathcal{A})$ returns "inconsistent".

Proof. Proof by contraposition, so let's assume $\mathsf{inconsistent}(\mathcal{A})$ returns $\mathcal{A}'$. Then $\mathcal{A}'$ is a complete ABox that does not contain a clash. We can now easily construct a model for $\mathcal{A}'$ and hence for $\mathcal{A}$. We define:

$$\begin{aligned}
\Delta^{\mathcal{I}} &= \{a \mid a : C \in \mathcal{A}' \text{ or } (a, b) : r \in \mathcal{A}' \text{ or } (b, a) : r \in \mathcal{A}'\},\\
a^{\mathcal{I}} &= a \text{ for each individual name } a \text{ occurring in } \mathcal{A}',\\
A^{\mathcal{I}} &= \{a \mid a : A \in \mathcal{A}'\} \text{ for each concept name } A \in \mathcal{A},\\
r^{\mathcal{I}} &= \{(a, b) \mid (a, b) : r \in \mathcal{A}'\} \text{ for role } r \text{ occurring in } \mathcal{A}'.
\end{aligned}$$

We can easily see that $\mathcal{I}$ is an interpretation. We now need to prove the following statement:

$$\text{if } a : C \in \mathcal{A}', \text{ then } a^{\mathcal{I}} \in C^{\mathcal{I}}.$$

The proof proceeds by induction on the structure of concepts. As for the induction basis: if C is a concept name and $a : C \in \mathcal{A}'$, then $a^{\mathcal{I}} \in C^{\mathcal{I}}$ by definition of $\mathcal{I}$.

The induction step consists of the following cases:

- $C = \neg D$. Because $\mathcal{A}'$ has no clash, $a : \neg D \in \mathcal{A}'$ implies that D is atomic and $a : D \notin \mathcal{A}'$. So $a^{\mathcal{I}} \notin D^{\mathcal{I}}$ and hence $a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \setminus D^{\mathcal{I}} = C^{\mathcal{I}}$.

- $C = D \sqcap E$. For every formula $a : D \sqcap E \in \mathcal{A}'$, by completeness of $\mathcal{A}'$, we must have both $a : D \in \mathcal{A}'$ and $a : E \in \mathcal{A}'$. So, by definition of $\mathcal{I}$, we have both $a^{\mathcal{I}} \in D^{\mathcal{I}}$ and $a^{\mathcal{I}} \in E^{\mathcal{I}}$. Hence we conclude $a^{\mathcal{I}} \in D^{\mathcal{I}} \cap E^{\mathcal{I}} = (D \sqcap E)^{\mathcal{I}}$, as desired.
- $C = \forall r.D$. Let $a : \forall r.D \in \mathcal{A}'$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. By the latter and how $\mathcal{I}$ is defined, we know that $(a, b) : r \in \mathcal{A}'$. By completeness of $\mathcal{A}'$, we must have $b : D \in \mathcal{A}'$ too. D is of lower complexity than $\forall r.D$, so we may use the induction hypothesis to deduce that $b^{\mathcal{I}} \in D^{\mathcal{I}}$. This holds for all b such that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$, so $a^{\mathcal{I}} \in (\forall r.D)^{\mathcal{I}}$.
- $C = \exists r.D$. Let $a : \exists r.D \in \mathcal{A}'$. By the completeness of $\mathcal{A}'$, we have both $(a, b) : r \in \mathcal{A}'$ and $b : D \in \mathcal{A}'$ for some individual name b. The first formula of these guarantees that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ by construction of $\mathcal{I}$. For the other formula, we can use the induction hypothesis (D has lower complexity than $\exists r.D$) to conclude that $b^{\mathcal{I}} \in D^{\mathcal{I}}$. Hence, $a^{\mathcal{I}} \in (\exists r.D)^{\mathcal{I}}$.

This proves that all concept assertions are satisfied by the interpretation, so $\mathcal{I}$ is a model for $\mathcal{A}'$. This means that $\mathcal{A}'$ is consistent and thus that $\mathcal{A}$ is consistent too, proving the theorem. □

Lemma 4.2.3. (Baader et al., 2017, p.80)**.** If $\mathsf{inconsistent}(\mathcal{A})$ returns "inconsistent", then $\mathcal{A}$ is inconsistent.

Proof. Proof by contraposition. Let's assume $\mathcal{A}$ to be consistent and let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of $\mathcal{A}$. If $\mathcal{A}$ is already complete, there is nothing to check, as $\mathsf{expand}(\mathcal{A}) = \mathcal{A}$ and $\mathsf{inconsistent}$ returns "consistent". For incomplete $\mathcal{A}$, expand applies expansion rules until completeness. These expansion rules preserve consistency.

- The $\sqcap$-rule. If $a : C \sqcap D \in \mathcal{A}$, then $a^{\mathcal{I}} \in (C \sqcap D)^{\mathcal{I}}$. So by definition we have both $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $a^{\mathcal{I}} \in D^{\mathcal{I}}$. So $\mathcal{I}$ is also a model of $\mathcal{A} \cup \{a : C,\ a : D\}$. Therefore $\mathcal{A}$ remains consistent after applying the rule.
- The $\exists$-rule. If $a : \exists r.C \in \mathcal{A}$, then $a^{\mathcal{I}} \in (\exists r.C)^{\mathcal{A}}$. By definition, there exists a $b \in \Delta^{\mathcal{I}}$ with both $(a^{\mathcal{I}}, b) \in r^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}$. Hence, for a new individual name x, $\mathcal{I}$ is also a model of $\mathcal{A} \cup \{(a, x) : r,\ x : C\}$, by choosing $x^{\mathcal{I}} = b$. This means that $\mathcal{A}$ is still consistent after applying the rule.
- The $\forall$-rule. If $\{a : \forall r.C,\ (a, b) : r\} \subseteq \mathcal{A}$, then $a^{\mathcal{I}} \in (\forall r.C)^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ and $b^{\mathcal{I}} \in C^{\mathcal{I}}$. This means that $\mathcal{I}$ is also a model of $\mathcal{A} \cup \{b : C\}$, so $\mathcal{A}$ is still consistent after applying the rule.

□

Theorem 4.2.4. (Baader et al., 2017, p.81)**.** The tableau algorithm formed by expand and inconsistent is a decision procedure for the inconsistency of nonempty $\mathcal{ALE}$ ABoxes.

Proof. From the above lemmas, it follows that the algorithm terminates and outputs "inconsistent" if and only if the input is inconsistent. □

This algorithm is not the most efficient. It solves the $\mathcal{ALE}$ ABox inconsistency problem by using exponentially much space compared to the input, because it may store a union of the set of all traces in its memory and there can be exponentially many traces. This is caused by AND-branching (Baader et al., 2003, p.83), and an example can be defined by:

$$C_1 = \exists r.A \sqcap \exists r.B$$
$$C_{n+1} = (\exists r.A \sqcap \exists r.B) \sqcap \forall r.C_n$$

The size of the ABox $\mathcal{A} = \{a_i : C_i \mid i < n\}$ grows linearly with n, but the fully expanded ABox $\mathcal{A}'$ models a binary tree of depth n and therefore grows exponentially with n. The tableau algorithm has the advantage of being easy to define and implement, whereas an **NP**-complete algorithm is more difficult to define and implement because it needs to keep track of which traces it has already looked at. In the next chapter we implement this tableau-based $\mathcal{ALE}$ ABox inconsistency algorithm in ACT-R.

5 The model SHARP

A human deciding the inconsistency of an $\mathcal{ALE}$ ABox can be assumed to mentally execute the inconsistent() algorithm, or something very similar. This assumption motivates simulating the inconsistent() algorithm in ACT-R. The result is SHARP, which stands for **S**imulating **H**uman **A**Box **R**easoning **P**erformance. SHARP is a piece of original work and is the main contribution of this thesis.

SHARP has a more extensive output than the algorithm because of its simulation purpose. After taking $\mathcal{ALE}$ ABox $\mathcal{A}$ as input, it outputs 'C' when $\mathcal{A}$ is consistent and 'I' when $\mathcal{A}$ is inconsistent, but it also outputs a *run* $(\phi_0, ..., \phi_n)$ (i.e. the order of formulas that were inspected) and the *inference time* IT (i.e. the time it thinks the thought process will take). Note that the latter is generally different from the simulation time (i.e. the time a computer needs to complete the simulation).

In designing SHARP several obstacles had to be overcome. These obstacles, together with how we avoided them, are discussed first, because they motivated many design choices. After this, a more detailed picture of SHARP's structure is given.

5.1 Obstacles in designing SHARP

It is important to mention that due to certain technicalities that will be discussed below, SHARP deviates from inconsistent() in some respects. Note moreover that we abstracted away from visual processes (i.e. we are not using the visual module in ACT-R) such that only the logical processes are modelled.

5.1.1 The issue of parsing

ACT-R uses chunks as its data structure. Values in a certain chunk's slots are strings and are treated as indivisible units; hence, they cannot be parsed by ACT-R during simulation. For example, if a string in a chunk's slot represents the description logic formula $a : A \sqcap B$, the strings representing the formulas $a : A$ and $a : B$, which are derivable from it in $\mathcal{ALE}$, cannot be constructed by an ACT-R model. Each formula therefore needs to be represented by a chunk that has specific slots dedicated to the formulas that can be inferred from it, as well as

other relevant strings. So the chunk representing the formula $a : A \sqcap B$ has one slot containing the string $a : A$ and another containing the string $a : B$.

Before the simulation, the declarative memory is loaded with chunks that represent each formula in the given ABox, as well as chunks representing each formula that can possibly be derived. Performing an inference step, now, amounts roughly to:

1. retrieving the desired formula chunk from the declarative memory,
2. *labelling* it by changing the contents of its 'derived'-slot,
3. storing it back in the declarative memory.

In our example, when inspecting the $a : A \sqcap B$-chunk, SHARP uses the string $a : A$ that is stored in one of its slots to retrieve the (unlabelled) chunk representing $a : A$ which has been stored in its declarative memory from the beginning of the simulation. It then creates a labelled copy of the $a : A$-chunk and stores it in the declarative memory; SHARP can, from then on, make further inferences using this labelled chunk.

5.1.2 The issue of discarding

We want to prevent SHARP from making the same inference twice. For example, after having inferred the formulas $a : A$ and $a : B$ from the formula $a : A \sqcap B$, we do not want SHARP to select $a : A \sqcap B$ again to make an inference from. That is, we want some method to *discard* the formula $a : A \sqcap B$ thereby preventing it from being retrieved from the declarative memory.

In ACT-R, however, the declarative memory stores all chunks that it has used earlier and a retrieval request may retrieve any chunk satisfying the request's criteria. One solution is to inspect the formulas in a fixed order, but we wanted the flexibility of having no fixed order of inspection. The solution to the problem of discarding is therefore that SHARP keeps a list of formulas on which it has already made an inference. Using this list, SHARP formulates retrieval requests that exclude the possibility of retrieving a formula that we would discard. The list of 'used' formulas is, however, of finite length due to the way ACT-R is designed: a given production rule can modify only a fixed number of slots in each buffer.

5.1.3 The issue of the universal restrictions

Universal restrictions like $a : \forall r.A$ should never be discarded, as formulas such as $(a, x) : r$ might be derived at any point in time and when this happens, SHARP needs the formula $a : \forall r.A$ again to be able to infer $x : A$. Therefore, the discard-

ing mechanism from the previous paragraph does not work for universal restrictions. Instead, the chunks representing universal restrictions like $a : \forall r.A$ have designated slots forming its *role-list* that may contain role formulas like $(a, x) : r$. After inferring $x : A$ from the above two formulas, SHARP creates a copy of the $a : \forall r.A$-chunk with the string $(a, x) : r$ in its role-list. Then it stores the chunk in the declarative memory. In selecting the next role-formula, SHARP then uses the chunk's role-list to retrieve a role formula that is not in its role-list.

One potential problem with the above is that if the formula $a : \forall r.A$ is selected to make an inference from, we want to retrieve the chunk representing it that has the most recently updated role-list. This is not something we can directly specify in a retrieval request, so we use so-called count-order-chunks. The chunks in the goal buffer contain a designated slot for counting. Each time when SHARP inspects a universal restriction, it is labelled with the current count. After inspecting all universal restrictions, the count in the goal buffer is increased by one, using the count-order-chunks. From all the chunks in the declarative memory that represent a certain universal restriction $a : \forall r.A$, only the ones with the most recent count are retrieved, making sure that the corresponding role-list is up to date. Note that it is necessary to inspect all universal restrictions after another; otherwise, one chunk's most recent count may be different from another chunk's most recent count.

5.1.4 The issue of finding new elements

When SHARP infers $(a, x) : r$ and $x : A$ from an existential restriction $a : \exists r.A$, it is important that this x is a new element, i.e. not appearing in any formula that has already been derived. SHARP therefore needs to keep a used-list. An alternative would be to designate slots for the same purpose in the goal chunks, but it is good practice to keep the number of slots as low as possible. Each time after making an inference like the above, the element x is stored in the used-list. If, at a later point, SHARP makes an inference from a $a : \exists r.B$ (with $B \neq A$), the new element is selected to be not in the list, so that x cannot be chosen.

5.1.5 The issue of the buffers

ACT-R models have only a limited number of buffers. SHARP uses the following five: goal, imaginal, imaginal-action, retrieval and manual. Using more buffers is for the purposes of SHARP not recommended. So at any point in time, SHARP only has direct access to up to 5 chunks. When for a certain inference, two new chunks need to be retrieved, these chunks are retrieved in succession (as

only the retrieval buffer is capable of retrieving chunks from declarative memory, and it can only store one chunk). That is, after the first one appears in the retrieval buffer, it is moved to another buffer to clear the retrieval buffer for the second chunk.

5.1.6 The issue of the production rules

ACT-R's production rules are chunk-type unspecific. SHARP has two chunk types for formulas: one for the universal restriction formulas and one for the other formulas. This is due to the facts that chunks representing universal restrictions need to keep a role-list and they cannot be discarded, as discussed above. The different functions that these two types of chunk fulfil, demand production rules that are chunk-type specific; this was done by adding an extra slot for the chunk's type. This chunk-type dependency is the reason that SHARP's production rules sometimes come in two versions that each fulfil the same function from a logical perspective and only differ in the chunk types.

5.2 SHARP's design

5.2.1 A note on nondeterminism

SHARP often makes random selections among several possibilities, i.e. when deciding which formula to inspect next. These selections are *nondeterministic* in the sense that they affect SHARP's output run and inference time. They do not, however, affect the judgement: a consistent ABox will always be judged as such.

5.2.2 SHARP's chunk types

SHARP uses the following chunk types.

- goal: chunks of this type keep track of which sub-task SHARP needs to perform. They store important values to retrieve the most recent chunks.
- proposition: chunks of this type represent formulas that are not universal restrictions. They have slots for the relevant sub-formulas and labels.
- uproposition: the counterpart of the above. Additionally there are slots for the role-list.
- clash-list: the list that temporarily stores atomic formulas to derive a clash.
- store-list: the list of used formulas that are never to be inspected again. Contains only non-universal formulas.

- universal-list: this chunk temporarily stores the universal restrictions that do not need to be inspected for the moment.

- count-order: these chunks have two slots. One containing a number and the other the successor of that number.

- used-list: this chunk stores all roles that have been derived after inferring from an existential restriction.

5.2.3 Overview

During initialising SHARP, the chunks necessary in the simulation run (chunks representing each formula, count-orders and the starting goal chunk) are stored into the declarative memory. For this, an estimation is made of the number of distinct elements that are needed for the deduction. Chunks created during the simulation are stored in the declarative memory, except in case the buffer containing the chunk is cleared.

After initialisation, SHARP first tries to find a *clash* (a contradiction among atomic formulas), then it runs Module2 which selects a complex formula to make a derivation from. This complex formula can be one of three kinds: a conjunction, an existential restriction and a universal restriction, after which the $\sqcap$-rule, $\exists$-rule or $\forall$-rule fires. Depending on which one is selected, Module3, Module4 or Module5 will start running. The process stops if either a clash is found (in which case the ABox is inconsistent), or no clash is found and no new formula can be derived (in which case the ABox is consistent). The more detailed view of what happens inside the different modules can be seen in the frames below.

5.2.4 Module 1: find a clash

SHARP uses Module1 to find a clash among the atomic concept assignments. If at some point no clash is found, either the ABox is consistent, or the ABox is inconsistent and some other formula still needs to be derived in order to give rise to a clash. The variable $\text{Derive}_{\text{new}}$ handles these different situations. $\text{Derive}_{\text{new}}$ = **yes** denotes ignorance of whether a new formula can be derived. $\text{Derive}_{\text{new}}$ = **no** denotes certainty that nothing can be derived further (which, in the absence of a clash, means the ABox is consistent).

```
find atomic formula (nondeterministic)
store atomic formula in clash-list
repeat
    find atomic formula clashing with first formula in clash-list
    if found then
        output I and stop
    else
        find atomic formula not in clash-list (nondeterministic)
        stack this formula to clash-list
    end if
until no atomic formula found
if Derive_new = yes then
    go to Module2
else
    output C and stop
end if
```

Module1, find clash. nondeterminism of selections is indicated. $\text{Derive}_{\text{new}}$ indicates whether a new formula can be derived; it is set in Module2.

5.2.5 Module 2: find next complex formula

SHARP selects a new complex formula to be inspected and goes to the corresponding module where the relevant inference step is made. There are two different production rules taking care of this selection, because universal restrictions have a different chunk type compared to the other formulas. If one of those production rules fails to retrieve a chunk, the other one fires. Note that only those universal restrictions are retrieved with the current count in the goal state. If the two rules both fail to retrieve a formula after one another, the variable $\text{Derive}_{\text{new}}$ is set to **no** to indicate that if Module1 does not find a clash, the ABox is consistent.

```
find complex formula (nondeterministic)
if conjunction found then
    go to Module3
else if existential restriction found then
    go to Module4
else if universal restriction found (with prvious count) then
    go to Module5
else if no non-universal formula found then
    find universal restriction (nondeterministic)
    if universal restriction found (with previous count) then
        go to Module5
    else
        set Derive_new = no
        go to Module1
    end if
else if no universal restriction found then
    find non-universal formula (nondeterministic)
    if conjunction found then
        go to Module3
    else if existential restriction found then
        go to Module4
    else
        set Derive_new = no
        go to Module1
    end if
end if
```

Module2, find next complex formula. There are different production rules for finding a non-universal formula and finding a universal restriction, because of their different chunk types. When a retrieval request for the one fails, it tries the other. Nondeterministic choices are indicated.

5.2.6 Module 3: infer from conjunction

In Module3, the formula under consideration is first stored in a store-list, thereby disqualifying it for further inference. Then SHARP, in random order, labels the two conjuncts as derived. From then on the conjuncts qualify for further in-

ference (if they not also happen to be used earlier in the simulation run). In case one of the conjuncts is an atomic formula, SHARP runs Module1 to find a clash. If both conjuncts are atomic, SHARP randomly selects one to find a clash with, so only in this case is the choice nondeterministic. If both conjuncts are non-atomic, it runs Module2.

store conjunction in store-list
retrieve a conjunct and label it as derived (nondeterministic)
retrieve the other conjunct and label it as derived
if atomic formula derived **then**
    select atomic formula and put in clash-list (possibly nondeterministic)
    go to Module1
else
    go to Module2
end if

Module3, derive from conjunction. The choice of which conjunct to derive first is a nondeterministic one.

5.2.7 Module 4: infer from existential restriction

Module4 first stores the existential restriction in a store-list. It then selects a corresponding and unused role formula and labels it as derived. It retrieves the corresponding (complex) formula that represents the (complex) concept of the found related element and labels it as derived too. Module1 is invoked if the found concept is atomic. Else, it runs Module2.

```
store formula in store-list
retrieve used-list of role formulas that are already used
retrieve role formula not in the used-list (nondeterministic)
label role formula as derived
retrieve corresponding (complex) formula
label formula as derived
if atomic formula found then
    put formula in clash-list
    go to Module1
else
    go to Module2
end if
```

Module4, derive from an existential restriction.

5.2.8 Module 5: infer from universal restriction

Module5 makes inferences on the universal restriction formulas and does so exhaustively. That it does so exhaustively is undesired from a modelling perspective, but it is necessary to solve the problem that ACT-R is not quite suitable to keep track of an unlimited number of chunks.

```
repeat
    repeat
        retrieve role formula not in universal-chunk (nondeterministic)
        store role formula in universal-chunk
        retrieve corresponding (complex) concept
        label formula as derived
    until no role formula found
    label universal restriction with current count
    put universal restriction in universal-list
    find universal restriction not in universal-list
until no universal restriction found
set count → count + 1
go to Module2
```

Module5, derive from a universal restriction.

First, a corresponding role formula is retrieved. In case of multiple role for-

mulas corresponding, this selection is nondeterministic. Then the corresponding (complex) concept formula is retrieved and labelled. This process is repeated until no more relevant role formulas can be found. Then the universal chunk is labelled with the current count. After that, another universal formula is retrieved and the same procedure is applied. This is repeated until no universal formula can be retrieved. count is increased by one to indicate that, for the time being, it is useless to select a universal restriction to make an inference from. Lastly, Module2 is run.

5.2.9 Justification

Three properties of ACT-R prevent a complete implementation of the inconsistent() algorithm in SHARP. The first is that the list-chunk types in SHARP are fixed, meaning in particular that their length is limited. This causes problems when the ABox under consideration is too complex. In that case, the lists fail to store the necessary formulas and SHARP keeps selecting formulas that should be disqualified from retrieval (this can happen when: selecting atomic formulas using the clash-list in Module1, selecting a complex formula using the store-list in Module2, selecting a universal restriction using the universal-list in Module5 and selecting a role formula using the role-list in Module4). In these cases SHARP does not terminate. A future improvement on SHARP may be to make the lists' lengths scale with the complexity of the input, but this would require code over and above ACT-R and seems difficult to implement.

The second property preventing a full implementation of the $\mathcal{ALE}$ ABox inconsistency algorithm is ACT-R's subsymbolic parameters. More specifically: the retrieval threshold, the decay parameter, the base-level activation, noise level and latency factor. These parameters, which are explained in Section 3.7.7, affect the retrieval process. Most relevant for our implementation is the possibility of a retrieval failure. This happens when a chunk's activation drops below the retrieval threshold. SHARP might then fail to find a clash, even if the ABox under consideration is inconsistent.

Besides the two problems above, the other point where SHARP differs from the abstract $\mathcal{ALE}$ ABox inconsistency algorithm is in its selection of the expansion rule to apply. Due to the issue related to the universal restrictions reported earlier, SHARP's selection can at some point during a simulation run be confined to just universal restrictions. This deviation does not, however, have an influence on SHARP's final judgement on the input ABox.

5.3 Analysis of performance

As established in the previous section, SHARP needs exponentially much memory space with increasing input, because the inconsistent() algorithm does too.

5.3.1 Exponential scaling

To verify that there is at least exponential scaling with the input, simulations were made on the five ABoxes below. They are designed to show AND-branching, meaning that the ABox's models are trees with at least binary branching, so that the size of these structures grows exponentially with the size of the ABox.

- $\mathcal{A}_0 = \{a : \exists r.A \sqcap \exists r.B\}$ with $\mathsf{size}(\mathcal{A}_0) = 5$,
- $\mathcal{A}_1 = \{b : \exists r.(\exists r.A \sqcap \exists r.B) \sqcap \exists r.B\}$ with $\mathsf{size}(\mathcal{A}_1) = 9$,
- $\mathcal{A}_2 = \{c : (\exists r.A \sqcap \exists r.B) \sqcap \forall r.(\exists r.A \sqcap \exists r.B)\}$ with $\mathsf{size}(\mathcal{A}_2) = 12$,
- $\mathcal{A}_3 = \{d : (\exists r.A \sqcap \exists r.B) \sqcap \forall r.(\exists r.(\exists r.A \sqcap \exists r.B) \sqcap \exists r.B)\}$ with $\mathsf{size}(\mathcal{A}_3) = 16$,
- $\mathcal{A}_4 = \{e : ((\exists r.A \sqcap \exists r.B) \sqcap \forall r.((\exists r.A \sqcap \exists r.B) \sqcap \forall r.(\exists r.A \sqcap \exists r.B)))\}$ with $\mathsf{size}(\mathcal{A}_4) = 19$.

The exponential scaling of the size of the models with the size of the input requires an exponential scaling in the number of reasoning steps. This results in an exponential increase of run time, which can be recognised in the graph of Figure 1. Each data-point corresponds to an average of 10 simulations. The data is fit to an exponential function of the form:

$$y = a \cdot e^{b \cdot x},$$

where a, b are the parameters and e is the exponential constant and has $r^2 = 0.9991$, indicating that the increase is indeed exponential.

SHARP is expected to be unable to handle ABoxes that have a bigger size than $\mathcal{A}_4$ above, as some list types may have insufficiently many slots. It is difficult to determine the size limit beyond which SHARP fails, because different ABoxes with the same size may overflow different list-types.

5.3.2 Polynomial scaling

The above situation is the worst possible scenario. In scenarios without AND-branching the run time scales polynomially or even linearly with the input size. To verify the polynomial scaling, we ran SHARP on the following ABoxes:

- $\mathcal{A}_0 = \{a : A\}$ with $\mathsf{size}(\mathcal{A}_0) = 1$,
- $\mathcal{A}_1 = \{a : A,\ (a, b) : r,\ a : \forall r.B\}$ with $\mathsf{size}(\mathcal{A}_1) = 4$,

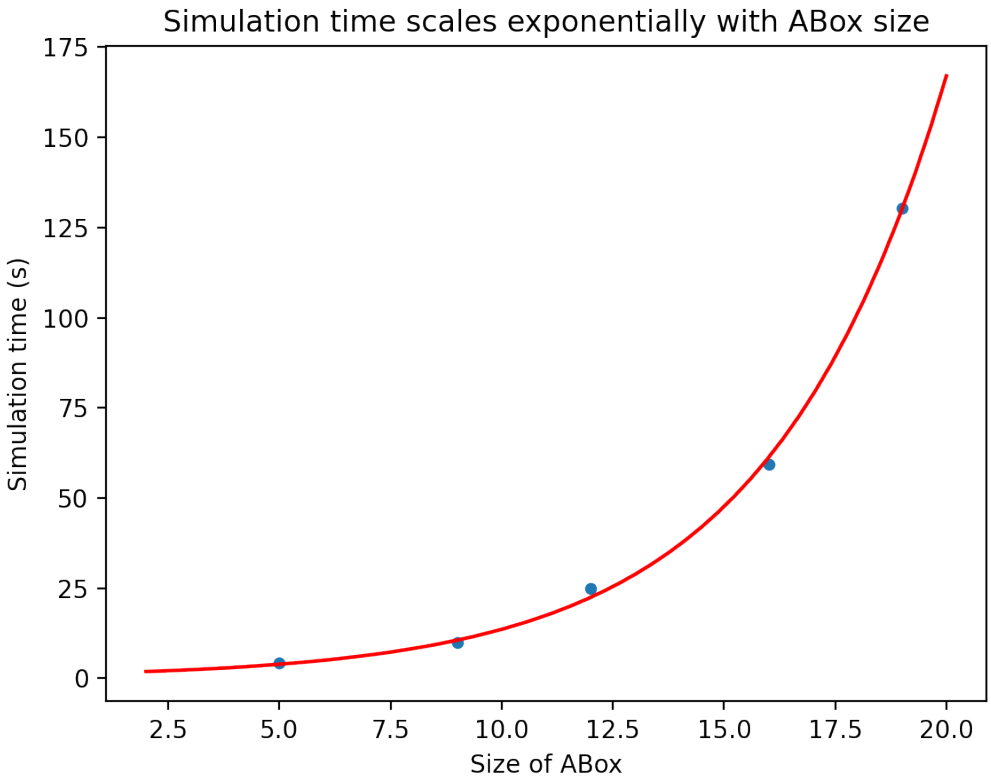

Figure 1: Exponential scaling of simulation time with input size in case of AND-branching.

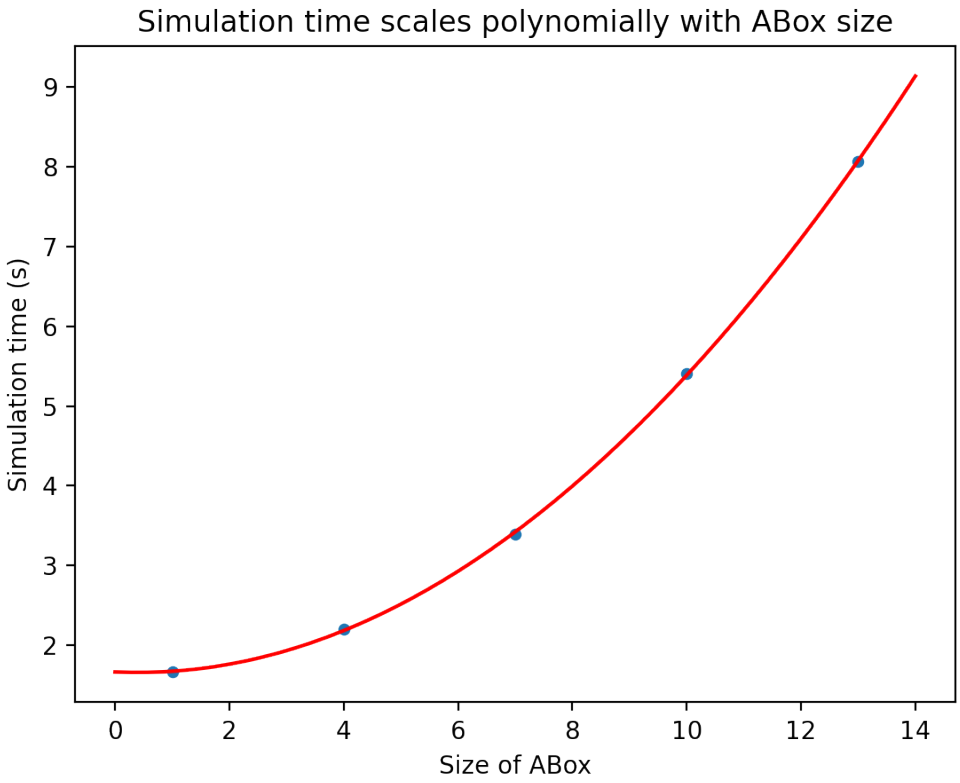

Figure 2: The less complex scenarios show polynomial scaling of the simulation time with the input size.

- $\mathcal{A}_2 = \{a:A,\ (a,b):r,\ (a,c):r,\ a:\forall r.B,\ a:\forall r.C\}$ with $\mathsf{size}(\mathcal{A}_2) = 7$,
- $\mathcal{A}_3 = \{a:A,\ (a,b):r,\ (a,c):r,\ (a,d):r,\ a:\forall r.B,\ a:\forall r.C,\ a:\forall r.D\}$ with $\mathsf{size}(\mathcal{A}_3) = 10$,
- $\mathcal{A}_4 = \{a:A,\ (a,b):r,\ (a,c):r,\ (a,d):r,\ (a,e):r,\ a:\forall r.B,\ a:\forall r.C,\ a:\forall r.D,\ a:\forall r.E\}$ with $\mathsf{size}(\mathcal{A}_4) = 13$.

For these ABoxes, the number of $\forall$-rule applications in the inconsistent() algorithm grows quadratically with the size of the ABoxes. This results in a quadratic scaling of run time. The time average of 10 simulations was taken for each ABox and the resulting data was fit to the polynomial formula:

$$y = a + b \cdot x + c \cdot x^2,$$

with a, b and c being the parameters. This resulted in Figure 5.3.2.

For the fit, we can report $r^2 = 0.9999$, confirming the quadratic dependence on input size.

5.3.3 Linear scaling

In some simple cases, the scaling is linear; to demonstrate this, we fed the ABoxes below to SHARP, where the number of steps needed in the inconsistent() scales linearly with the size of the ABox.

- $\mathcal{A}_0 = \{a:A\}$ with $\mathsf{size}(\mathcal{A}_0) = 1$,
- $\mathcal{A}_1 = \{a:A,\ a:B\}$ with $\mathsf{size}(\mathcal{A}_1) = 2$,
- $\mathcal{A}_2 = \{a:A,\ a:B,\ a:C\}$ with $\mathsf{size}(\mathcal{A}_2) = 3$,
- $\mathcal{A}_3 = \{a:A,\ a:B,\ a:C,\ a:D\}$ with $\mathsf{size}(\mathcal{A}_3) = 4$,
- $\mathcal{A}_4 = \{a:A,\ a:B,\ a:C,\ a:D,\ a:E\}$ with $\mathsf{size}(\mathcal{A}_4) = 5$.

SHARP ran 10 simulations on each ABox and the average simulation time was taken as the data, on which a linear regression was made, using the formula:

$$y = a + b \cdot x,$$

with a and b being the parameters. This resulted in Figure 3, for which we can report $r^2 = 0.982$, showing that the linear dependence on input size is a good approximation to the run time, although slight deviations from the linear pattern can be seen.

5.3.4 Future extensions

In the future, SHARP could be extended to simulate reasoning in description logics stronger than $\mathcal{ALE}$. In particular, the following non-exhaustive list of suggestions contains constructors that can likely be incorporated into SHARP.

- Role intersection, $r \sqcap s$. New role chunks need to be added with slots that contain their sub-roles. SHARP can then reason with the sub-roles separately.
- Role negation, $\neg r$. A chunk type representing a list with all the role formulas corresponding to r needs to be added. SHARP can then retrieve all role formulas not in that list.
- Role composition, $r \circ s$. The role formulas need to be extended to contain the sub-formulas. SHARP can then derive those and reason about them separately.
- Transitive closure of roles, r^+. A similar construction as in Module5 needs to be made, as formulas containing transitive closures of roles can never be discarded. SHARP is then able to re-use these formulas when needed.

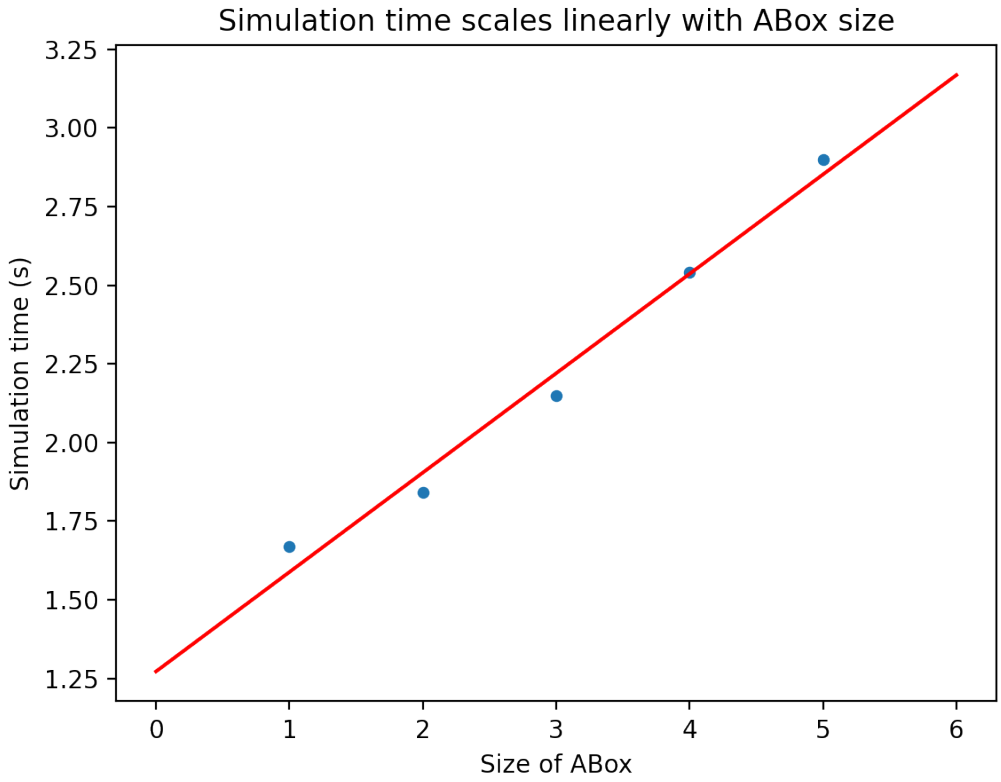

Figure 3: In some easy cases, there is a linear dependence of simulation time on the size of the input.

- Role inverses, r^-. New slots need to be added to store the inverse pair. SHARP can then retrieve that pair from declarative memory and mark it as derived.
- Number restrictions, $\geqslant nr$ for a natural number n. SHARP needs a chunk type to list the number of r- successors. It can then add as many as needed, similar to what happens in Module4. Note that n is now assumed to be fixed.
- TBox formulas, $A \sqsubseteq B$ or $A \equiv B$. A construction as in Module5 needs to be made, because these formulas cannot be discarded. SHARP may then infer formulas such as $a : B$ from $a : A$ and $A \sqsubseteq B$.

The above suggestions are necessarily a bit vague as they are not implemented yet. Complex concept negations and concept unions are not in the list, as they can probably not be implemented in SHARP: backtracking is needed and this seems impossible in ACT-R, because it is impossible to erase chunks from the declarative memory.

Moreover, SHARP could be extended to allow *deep inference* (Tubella and Straßburger, 2019) and (Strannegård et al., 2013). Deep inference refers to inference rules that apply to connectives other than the main connective. This extension would most likely be quite involved, as the chunks representing the formulas must have slots for all their subformulas.

Another way in which SHARP can be improved is by tweaking its subsymbolic parameters to make its output fit empirical data. The empirical data needed for this step is planned to be collected in the near future, by measuring human performance on the list of ABoxes in Section 6.2.8.

Based on the proposed methodology in (Dimov et al., 2020) multiple models need to be made in order to compare their performance on data after fitting. This will improve the performance of the final model and makes it easier to argue for the modelling choices made.

In the literature there are indications that some inference steps are much harder than others, e.g. trivial satisfaction of universal quantification (Horridge, 2011, p.245). This distinction is not directly made in SHARP, but it can be added by associating rewards to the production rules. These rewards can in turn be tweaked to make the model's output fit the data thereby increasing its accuracy.

SHARP does not make use of the visual module, but this functionality could be added to it. Adding this feature will make predictions perhaps more accurate, as the order of formulas will likely affect inference time. Design choices in the visual module could also be supported by eye-tracking measurements.

6 Predicting inference times with SHARP

The inference times that SHARP predicts are dependent on the subsymbolic parameters in ACT-R, the values of which we do not know exactly. This means that SHARP's predicted inference times might deviate from experimental data, because the parameters take the wrong settings. It is therefore important to understand the distinction between SHARP's quantitative predictions, i.e. the exact inference times predicted, and the qualitative predictions, i.e. the more general effects that are displayed in the simulated data. These qualitative effects seem robust with respect to minor changes in the used ACT-R parameter values, so they can be more easily empirically falsified. The quantitative predictions are empirically weaker, because these are dependent on the exact parameter values, which are not fixed, but can be adjusted post hoc for fitting to data.

6.1 Predicted effects

The graphs in this chapter serve to illustrate the qualitative effects found. We set these subsymbolic parameters to the indicated values:

- utility noise = 0.2
- retrieval threshold = −0.05
- decay = 0.005 , default value: 0.5.
- instantaneous noise = 0.005, typical value: 0.25.

These values are rather different from the default values. They were chosen such as to make SHARP's output match the inference times we deemed realistic: no formal criterion was used for this estimation, in particular there was no experimental data to fit the parameter values to. We enabled base-level learning and production compilation, but disenabled partial matching and utility learning.

In the following, unless otherwise indicated, 300 simulations were run on each ABox. The data is presented in terms of kernel density estimations. These estimations are non-parametric smoothings of the data, aimed to approximate the underlying probability density function; they can be regarded as the continuous analogues of histograms. They are made as follows, if $(x_0, ..., x_n)$ is a certain sample from a distribution we are interested in estimating, we calculate and plot the function:

$$f_b(x) = \frac{1}{nb} \sum_{i=0}^{n} K\left(\frac{x - x_i}{b}\right),$$

where K is the so-called *kernel*. In our case, we used a gaussian kernel, where K is given by:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

$f_b(x)$ can be seen as the resulting function by summing over the kernels that are evaluated at each data-point. Increasing b increases the width of these kernels, so that the influence of a data-point reaches further. In the limit of $b \to 0$, the kernels become dirac delta distributions and the data is unsmoothed.

The Scikit Learn Python library was used to create the plots in this section; the bandwidth of its KernelDensity function was set to 0.05 (which yielded visually good results, no formal criterion was used); otherwise default settings were used.

6.1.1 Statistical significance

According to (Troitzsch, 2014) and (White et al., 2013) statistical significance of sample differences is considered less relevant than effect size when simulated data is concerned. To illustrate this, let us imagine two populations that are different in some way. In principle, if we make sample sizes large enough, samples can always be drawn from these populations that are statistically significantly different. When it comes to empirical data, sample sizes this large are usually unattainable because of budget constraints, deadlines, and other limitations.

In simulation experiments, on the other hand, there is much more control over the sample size. Therefore, population differences, if they exist, can *always* be made visible by making the sample sizes large enough, i.e. by running the simulation long enough. This makes the situation of two samples being statistically significantly different less interesting.

So, rather than demonstrating *that* two samples are different, we will look at *how large* this difference actually is and whether this is interesting. We therefore estimate the *relative difference of the means* of two samples (t_i) and (s_i), expressed as a percentage. This seems reasonable for our purposes, as we are interested in the factor by which the inference time of one process exceeds the inference time of another. The absolute differences are also reported and serve as illustrations. The relative difference of the means is defined by:

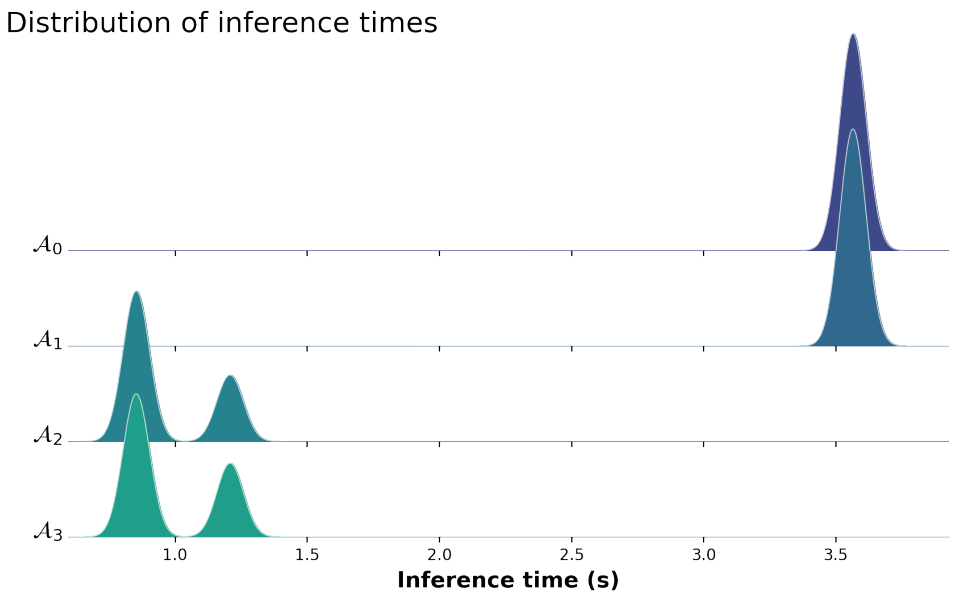

Figure 4: Changing element names does not affect inference times.

$$\Delta((t_i),(s_i)) = \frac{\mathsf{mean}(\{t_i\}) - \mathsf{mean}(\{s_i\})}{\mathsf{mean}(\{t_i\})}.$$

We use $\mathsf{IT}_{\mathcal{A}_i}$ to denote the sample of simulated inference times corresponding to $\mathcal{A}_i$. The absolute differences reported below serve as illustrations.

6.1.2 Name dependence

$\mathcal{ALE}$ has element names, concept names and role names. We are interested in how renaming those things in a given ABox affects the predicted inference time, in particular when the new name already appears elsewhere in the ABox. To investigate the dependence on renaming element names, we ran simulations on the following ABoxes:

- $\mathcal{A}_0 = \{a:A,\ b:\neg A,\ a:B\}$
- $\mathcal{A}_1 = \{a:A,\ b:\neg A,\ b:B\}$
- $\mathcal{A}_2 = \{a:A,\ a:\neg A,\ a:B\}$
- $\mathcal{A}_3 = \{a:A,\ a:\neg A,\ b:B\}$

Where the ABoxes are identical except for certain element names. $\mathcal{A}_0$ and $\mathcal{A}_1$ are consistent and the other two are inconsistent.

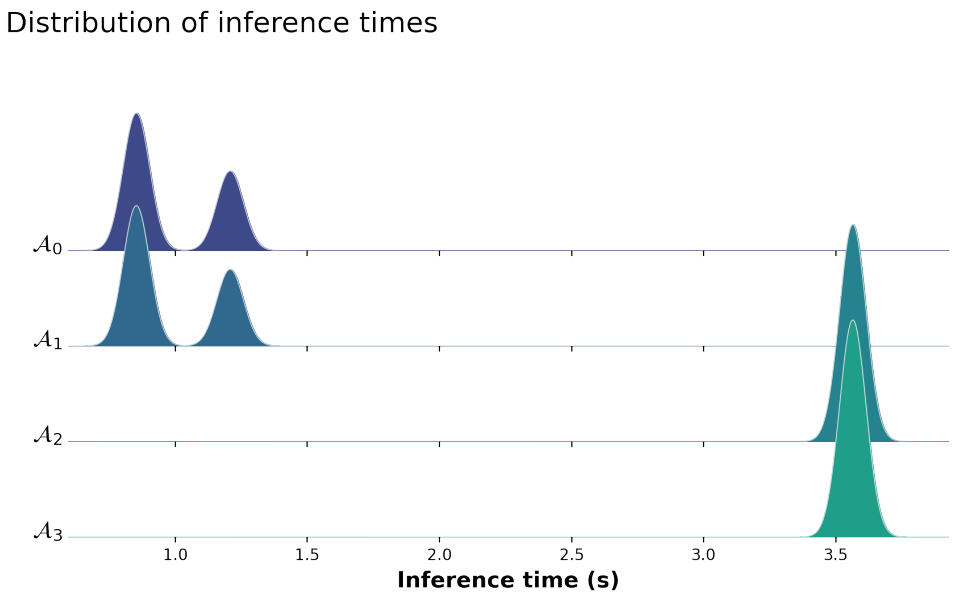

Figure 5: Changing concept names does not affect inference times.

The graph in Figure 4 shows that the inference times for $\mathcal{A}_0$ and $\mathcal{A}_1$ are very similar, as are the inference times for the other two ABoxes.

More numerically, we get $\Delta(\mathsf{IT}_{\mathcal{A}_0}, \mathsf{IT}_{\mathcal{A}_1}) = 0.0\%$ with the absolute difference being 0 seconds. And $\Delta(\mathsf{IT}_{\mathcal{A}_2}, \mathsf{IT}_{\mathcal{A}_3}) = 1.2\%$ with the absolute difference being 0.012 seconds.

This illustrates that when changing an element name in a given ABox without changing its consistency, no interesting change in inference time is predicted. This is not surprising from SHARP's design, as it makes no distinction between element names.

A similar pattern is seen with concept names. We used the following ABoxes:

- $\mathcal{A}_0 = \{a:A,\ b:A,\ a:\neg A\}$
- $\mathcal{A}_1 = \{a:A,\ b:B,\ a:\neg A\}$
- $\mathcal{A}_2 = \{a:A,\ b:A,\ a:\neg B\}$
- $\mathcal{A}_3 = \{a:A,\ b:B,\ a:\neg B\}$

Where again, these ABoxes are identical except for some concept names. The first two ABoxes are inconsistent and the last two are consistent.

In Figure 5 no difference is visible between the two inconsistent ABoxes; neither is there any substantial difference between the two consistent ABoxes. This is numerically confirmed by the relative difference in the mean inference

times: $\Delta(\mathsf{IT}_{\mathcal{A}_0}, \mathsf{IT}_{\mathcal{A}_1}) = -0.5\%$ with an absolute difference of -0.005 seconds. And $\Delta(\mathsf{IT}_{\mathcal{A}_2}, \mathsf{IT}_{\mathcal{A}_3}) = 0.0\%$ with an absolute difference of 0.0 seconds.

This shows that renaming concepts, if it does not change the consistency of the ABox, does not change inference time. Again this is to be expected on the basis of SHARP's design.

Things change, however, when it comes to role names. To investigate the dependence of renaming roles, we used the following ABoxes:

- $\mathcal{A}_0 = \{a : \exists r.A,\ a : \forall r.\neg A,\ a : \exists r.B\}$
- $\mathcal{A}_1 = \{a : \exists r.A,\ a : \forall r.\neg A,\ a : \exists s.B\}$
- $\mathcal{A}_2 = \{a : \exists r.A,\ a : \forall s.\neg A,\ a : \exists r.B\}$
- $\mathcal{A}_3 = \{a : \exists r.A,\ a : \forall s.\neg A,\ a : \exists s.B\}$
- $\mathcal{A}_4 = \{a : \exists r.A,\ a : \forall r.\neg A,\ a : \forall r.B\}$
- $\mathcal{A}_5 = \{a : \exists r.A,\ a : \forall r.\neg A,\ a : \forall s.B\}$
- $\mathcal{A}_6 = \{a : \exists r.A,\ a : \forall s.\neg A,\ a : \forall r.B\}$
- $\mathcal{A}_7 = \{a : \exists r.A,\ a : \forall s.\neg A,\ a : \forall s.B\}$

The first four ABoxes are identical to each other, except for some role names. The same holds for the last four ABoxes. The last four differ from the first only in that their last formula is a universal instead of an existential restriction; this is to investigate whether the effect of changing role names is dependent on the type of restriction they occur under (universal or existential).

The results are plotted in Figure 6. More quantitatively, we have:

- $\Delta(\mathsf{IT}_{\mathcal{A}_0}, \mathsf{IT}_{\mathcal{A}_1}) = -4.7\%$, with an absolute difference of -0.31 seconds.
- $\Delta(\mathsf{IT}_{\mathcal{A}_2}, \mathsf{IT}_{\mathcal{A}_3}) = -8.2\%$, with an absolute difference of -0.48 seconds.
- $\Delta(\mathsf{IT}_{\mathcal{A}_4}, \mathsf{IT}_{\mathcal{A}_5}) = 13.9\%$, with an absolute difference of 0.91 seconds.
- $\Delta(\mathsf{IT}_{\mathcal{A}_6}, \mathsf{IT}_{\mathcal{A}_7}) = -12.7\%$, with an absolute difference of -0.74 seconds.

It makes a difference whether a role name is the same as another, or different from all the others, even when the consistency of the ABox is not affected by it. On average we see that the ABoxes with one universal restrictions have longer inference times than the ones with two universal restrictions. Moreover, the ABoxes with only one universal restriction show a wider spread in inference times.

6.1.3 Order insensitivity

The order in which the formulas of the ABox are presented does not affect the inference time. To demonstrate this effect, we ran simulations on the following ABoxes:

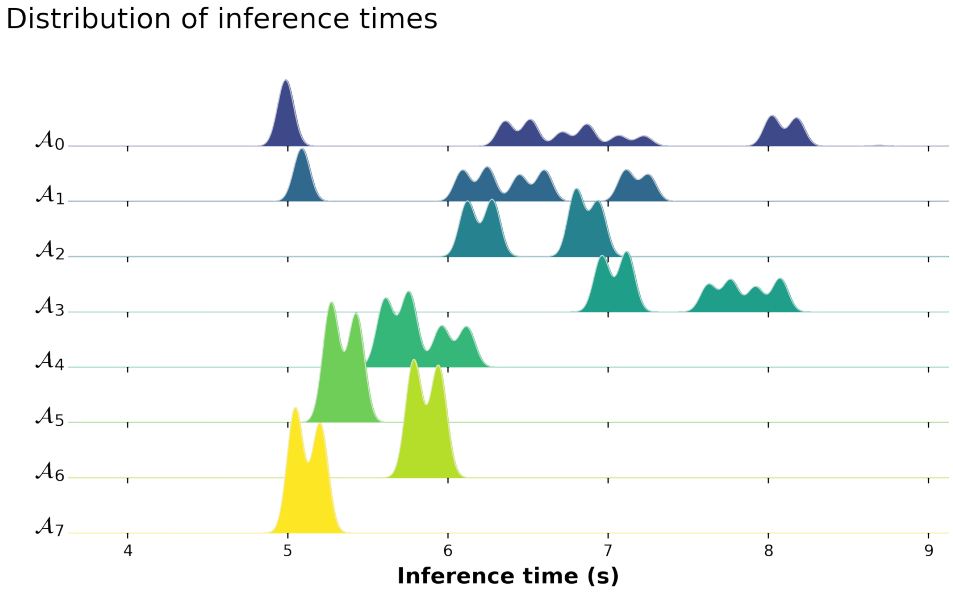

Figure 6: Changing role names does affect inference times non-trivially.

- $\mathcal{A}_0 = \{a : A \sqcap B,\ a : C\}$
- $\mathcal{A}_1 = \{a : C,\ a : A \sqcap B\}$
- $\mathcal{A}_2 = \{a : A \sqcap B,\ a : \neg A\}$
- $\mathcal{A}_3 = \{a : \neg A,\ a : A \sqcap B\}$

The first two ABoxes are consistent and differ only in the order of their formulas; the other two ABoxes are inconsistent and also have different formula orders. The insensitivity of order can be seen in Figure 7.

The first two ABoxes yield $\Delta(\mathsf{IT}_{\mathcal{A}_0}, \mathsf{IT}_{\mathcal{A}_1}) = 0.0\%$, with an absolute difference of 0.0 seconds. For the second two, we get: $\Delta(\mathsf{IT}_{\mathcal{A}_2}, \mathsf{IT}_{\mathcal{A}_3}) = -1.0\%$, with an absolute difference of -0.022 seconds. SHARP seems therefore insensitive to changing the order of formulas. From SHARP's design, this effect is not surprising, as all formulas in the ABox are fed to the model at once and the only distinction made in selecting the formula chunks is based on their activation.

Secondly, the order in which conjunctions are presented does not affect inference time. To demonstrate this, we fed SHARP the following ABoxes:

- $\mathcal{A}_0 = \{a : A \sqcap B,\ a : B\}$
- $\mathcal{A}_1 = \{a : B \sqcap A,\ a : B\}$
- $\mathcal{A}_2 = \{a : A \sqcap B,\ a : \neg B\}$
- $\mathcal{A}_3 = \{a : B \sqcap A,\ a : \neg B\}$

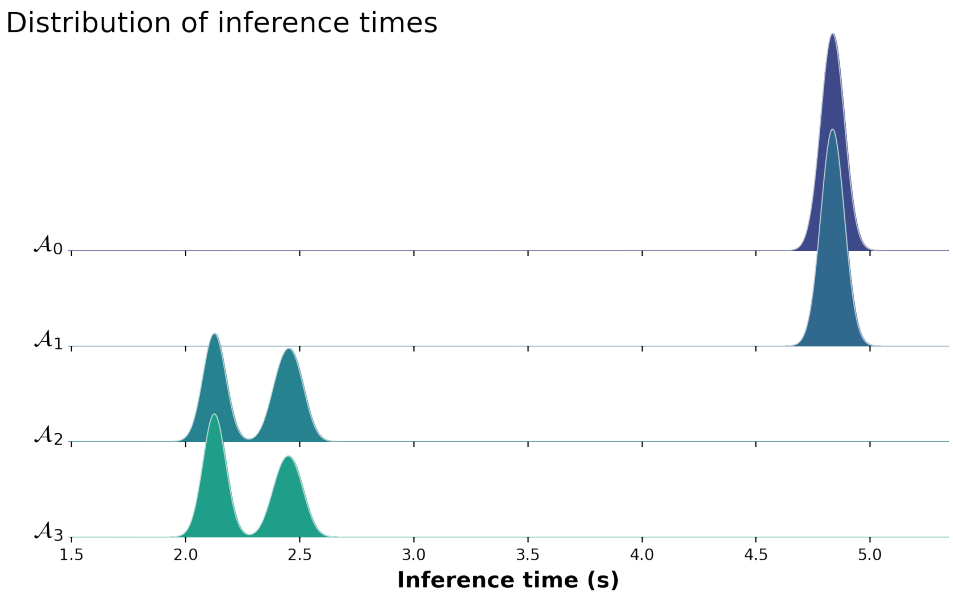

Figure 7: The order of presenting the formulas does not affect the inference time.

The first two are consistent and differ only in the order of the conjuncts in the first formula. The second two ABoxes are inconsistent and differ again only in the order of the conjuncts of the first formula.

Figure 8 shows no different inference time between the first two, and very similar difference in inference times between the second two ABoxes. More quantitatively speaking, we have $\Delta(\mathsf{IT}_{\mathcal{A}_0}, \mathsf{IT}_{\mathcal{A}_1}) = 0.0\%$, with an absolute difference of 0.002 seconds. For the inconsistent ABoxes, we have: $\Delta(\mathsf{IT}_{\mathcal{A}_2}, \mathsf{IT}_{\mathcal{A}_3}) = -0.6\%$, with an absolute difference of -0.015 seconds.

These small differences are not interesting, so we conclude that the order in which the conjuncts are given is irrelevant for predicting the inference time.

6.1.4 Insensitivity to negations

SHARP is insensitive to negations in the sense that a positive and negated atomic concepts are both processed equally easily. To show this effect, we observed SHARP's output on the following ABoxes:

- $\mathcal{A}_0 = \{a : A,\ a : B\}$
- $\mathcal{A}_1 = \{a : \neg A,\ a : \neg B\}$
- $\mathcal{A}_2 = \{a : A \sqcap B\}$
- $\mathcal{A}_3 = \{a : \neg A \sqcap \neg B\}$

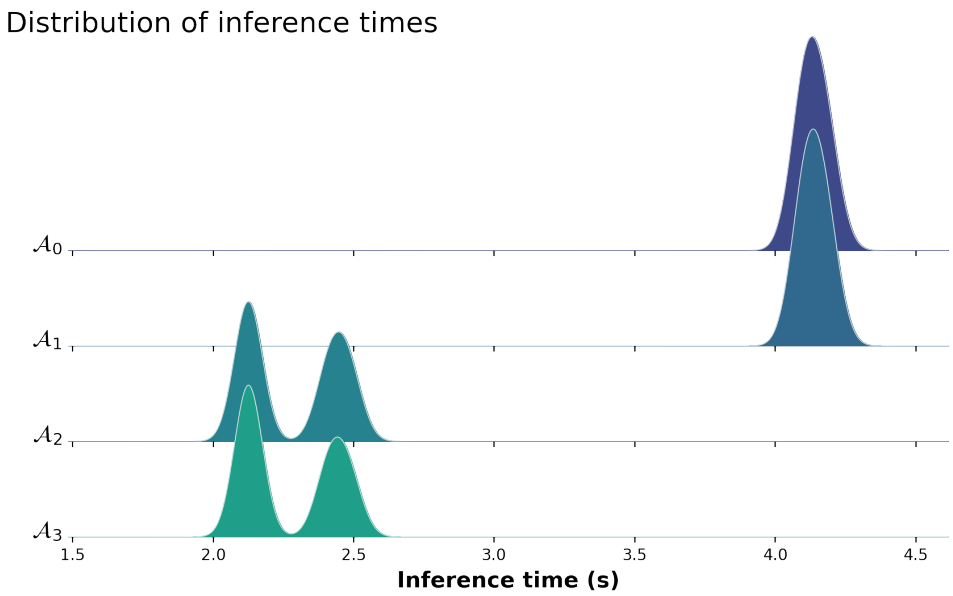

Figure 8: Conjunction order does not affect inference time.

- $\mathcal{A}_4 = \{a : \exists r.A\}$
- $\mathcal{A}_5 = \{a : \exists r.\neg A\}$

All the ABoxes above are consistent. $\mathcal{A}_0$ differs from $\mathcal{A}_1$ in that the concepts appears negated. The same holds for $\mathcal{A}_2$ and $\mathcal{A}_3$ and for $\mathcal{A}_4$ and $\mathcal{A}_5$ respectively.

Figure 9 indicates that it makes no difference to SHARP whether a concept appears negated or not. This is to be expected based on the inner workings of SHARP: retrieval requests when looking for a clash in Module1 make no distinction between positive and negative atomic concepts.

In quantitative terms, we have $\Delta(\mathsf{IT}_{\mathcal{A}_0}, \mathsf{IT}_{\mathcal{A}_1}) = 0.0\%$ with an absolute difference of 0.0 seconds. And $\Delta(\mathsf{IT}_{\mathcal{A}_2}, \mathsf{IT}_{\mathcal{A}_3}) = 0.0\%$ with an absolute difference of 0.001 seconds. And lastly $\Delta(\mathsf{IT}_{\mathcal{A}_4}, \mathsf{IT}_{\mathcal{A}_5}) = 0.0\%$ with an absolute difference of 0.0 seconds.

6.1.5 Nesting sensitivity

SHARP predicts that the inference time is highly dependent on how the primitive concepts are nested in conjunctions. For this subsection, we need the following useful definition.

Definition 6.1.1. Two ABoxes $\mathcal{A}$ and $\mathcal{B}$ are *concept-equivalent* if and only if

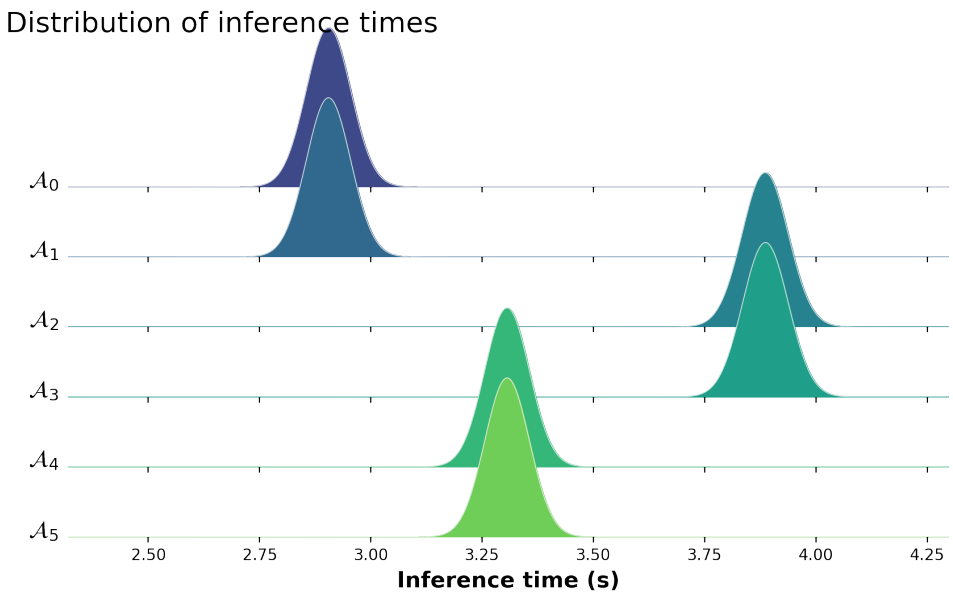

Figure 9: Whether atomic concepts are negated or not does not affect inference times.

there exists an ABox $\mathcal{C}$ that is a complete ABox corresponding to both $\mathcal{A}$ and $\mathcal{B}$.

The definition says that in concept-equivalent ABoxes, every element satisfies the same concepts in both ABoxes.

Deeply nested concepts can only be inferred after the more shallowly nested concepts have been inspected. We therefore expect that a deeply nested clash is found after a longer time than a shallowly nested clash. To demonstrate this effect, SHARP's outputs on the following ABoxes are compared.

- $\mathcal{A}_0 = \{a : ((A \sqcap \neg A) \sqcap (B \sqcap C))\}$
- $\mathcal{A}_1 = \{a : ((A \sqcap B) \sqcap (\neg A \sqcap C))\}$
- $\mathcal{A}_2 = \{a : (A \sqcap (\neg A \sqcap (B \sqcap C)))\}$
- $\mathcal{A}_3 = \{a : (A \sqcap (B \sqcap (\neg A \sqcap C)))\}$
- $\mathcal{A}_4 = \{a : (B \sqcap (A \sqcap (\neg A \sqcap C)))\}$
- $\mathcal{A}_5 = \{a : (B \sqcap (C \sqcap (\neg A \sqcap A)))\}$

These ABoxes are inconsistent and are all concept-equivalent. The first two differ from the rest in terms of the nesting structure of the conjunctions, they differ from each other in terms of how the clashing concepts are distributed in the conjunctions. The last four ABoxes also differ from each other in terms of how the clashing concepts are distributed in the conjunctions.

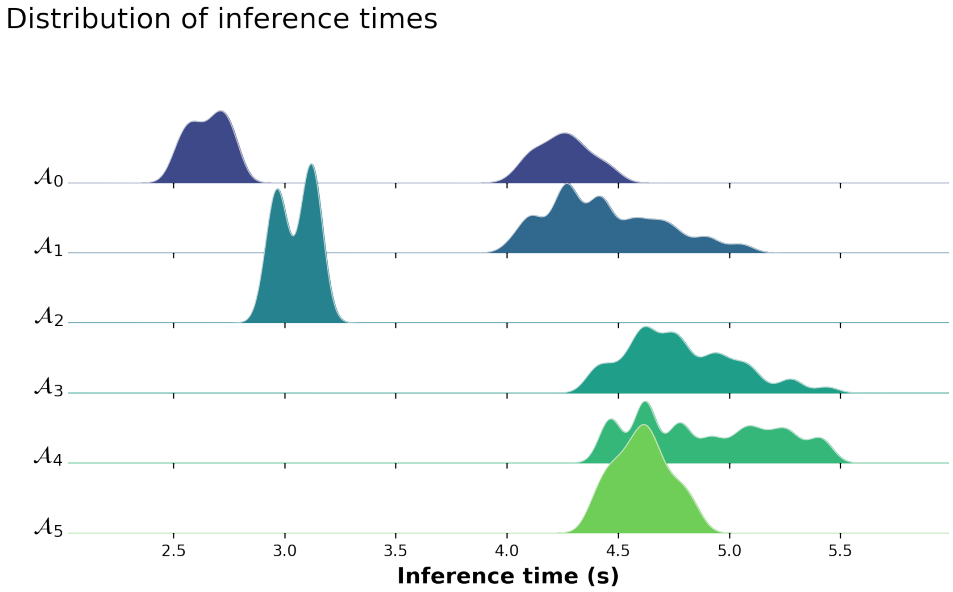

Figure 10: Nesting affects the inference times.

After feeding the ABoxes to SHARP, we get Figure 10.

The data shows varying inference times, despite concept-equivalence. Comparing $\mathcal{A}_0$ with $\mathcal{A}_1$ gives $\Delta(\mathsf{IT}_{\mathcal{A}_0}, \mathsf{IT}_{\mathcal{A}_1}) = 31.0\%$, the difference in absolute terms being 1.05 seconds. This difference can be explained by the fact that the clash in $\mathcal{A}_0$ can be derived after two inference steps, whereas the clash in $\mathcal{A}_1$ needs three inference steps. Moreover, $\mathcal{A}_0$ allows the clash to be derived after three steps, which creates the second peak.

Comparing $\mathcal{A}_1$ with $\mathcal{A}_3$, we get $\Delta(\mathsf{IT}_{\mathcal{A}_1}, \mathsf{IT}_{\mathcal{A}_3}) = 7.7\%$, with the absolute difference being 0.342 seconds. This difference is present because SHARP looks for a clash among primitive concepts – i.e. it starts Module1 – as soon as it derives any formula of the form $a : C$ with C a primitve concept name. In reasoning from $\mathcal{A}_3$, SHARP runs Module1 an extra time compared to when SHARP reasons from $\mathcal{A}_1$, making the inference time longer.

Comparing $\mathcal{A}_2$ with $\mathcal{A}_5$, we get $\Delta(\mathsf{IT}_{\mathcal{A}_2}, \mathsf{IT}_{\mathcal{A}_5}) = 51.1\%$, with the absolute difference being 1.56 seconds. This is again explained by the fact that SHARP has to make three inference steps when reasoning on $\mathcal{A}_5$, whereas it (precisely) needs two reasoning steps when reasoning from $\mathcal{A}_2$.

When we compare $\mathcal{A}_3$ with $\mathcal{A}_4$, we get $\Delta(\mathsf{IT}_{\mathcal{A}_3}, \mathsf{IT}_{\mathcal{A}_4}) = 2.2\%$, with the absolute difference being 0.108 seconds. This relatively small difference is explained by the fact that, in the case of $\mathcal{A}_3$, the chunk representing the formula

$a : A$ has been retrieved twice from declarative memory when the clash is derived. In the case of $\mathcal{A}_4$, however, it has been retrieved only once when the clash is derived. This means that the activation of the chunk representing $a : A$ is slightly higher in the case of $\mathcal{A}_3$, making it easier to retrieve the clash. For both ABoxes, the number of reasoning steps are the same. This shows that the chunks' activation has a nontrivial, albeit little, effect on inference time.

One might expect humans to be less susceptible to these nesting effects, as they are able to reason with *deep inference* rules (Tubella and Straßburger, 2019). These rules, however, can hardly be implemented in SHARP's production rules; indeed, such an implementation necessitates formulas represented by chunks with slots for all the formula's subformulas.

6.1.6 Time scaling with input size

As we saw in the previous chapter, AND-branching is what makes reasoning in $\mathcal{ALE}$ difficult. SHARP's run time was seen to scale exponentially with the input size, and it is therefore not surprising that the inference time also scales exponentially with input size in the case of AND-branching. To test this, we used the following ABoxes.

- $\mathcal{A} = \{a : \exists r.A \sqcap \exists r.B\}$ with $\mathsf{size}(A) = 5$,
- $\mathcal{B} = \{b : \exists r.(\exists r.A \sqcap \exists r.B) \sqcap \exists r.B\}$ with $\mathsf{size}(B) = 9$,
- $\mathcal{C} = \{c : (\exists r.A \sqcap \exists r.B) \sqcap \forall r.(\exists r.A \sqcap \exists r.B)\}$ with $\mathsf{size}(C) = 12$,
- $\mathcal{D} = \{d : (\exists r.A \sqcap \exists r.B) \sqcap \forall r.(\exists r.(\exists r.A \sqcap \exists r.B) \sqcap \exists r.B)\}$ with $\mathsf{size}(D) = 16$,
- $\mathcal{E} = \{e : ((\exists r.A \sqcap \exists r.B) \sqcap \forall r.((\exists r.A \sqcap \exists r.B) \sqcap \forall r.(\exists r.A \sqcap \exists r.B)))\}$ with $\mathsf{size}(E) = 19$.

The models corresponding to each ABox have a binary tree structure with 3, 5, 7, 11 and 15 nodes respectively. Twenty simulations were run on each of the ABoxes above. The data is fit to the formula:

$$y = a \cdot e^{b \cdot x},$$

with parameters a, b. The data and the fit are shown in Figure 11.

We reach $r^2 = 0.971$ (with parameter values $a = 1.678$, $b = 0.195$) indicating an exponential increase in inference time with increasing input size. ABoxes $\mathcal{B}$ and $\mathcal{D}$ have a slightly lower inference time than might be expected compared to the inference times of the other ABoxes. This happens because the relatively inefficient Module5 is invoked fewer times.

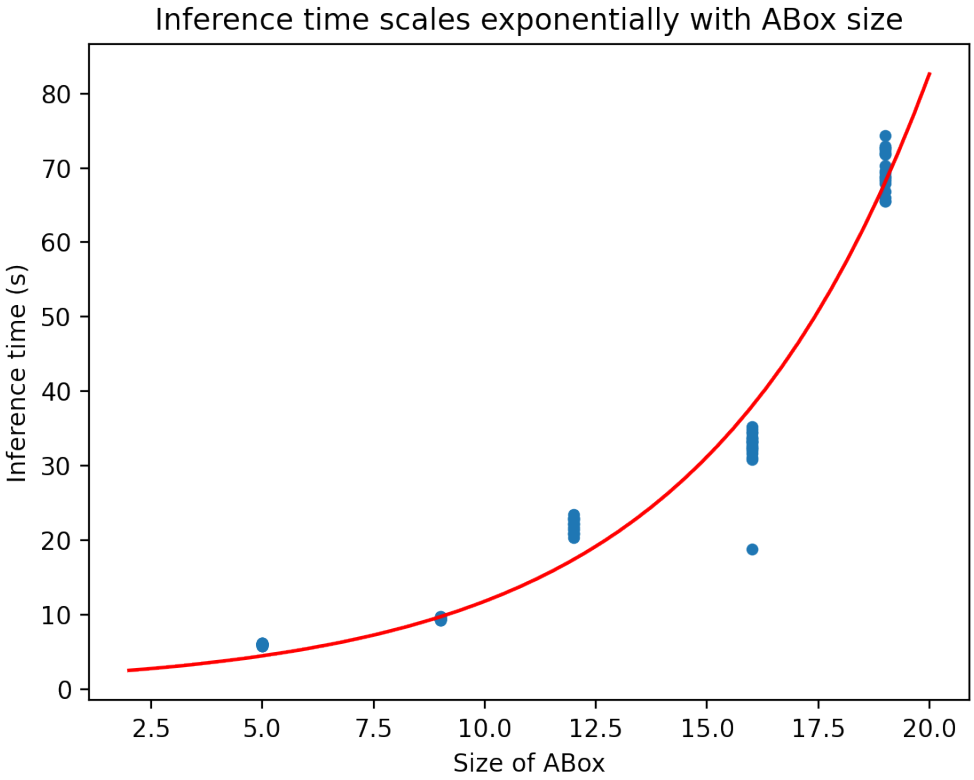

Figure 11: The inference time roughly scales exponentially with the input size in the case of AND-branching, showing a similar scaling as the simulation time in Figure 1.

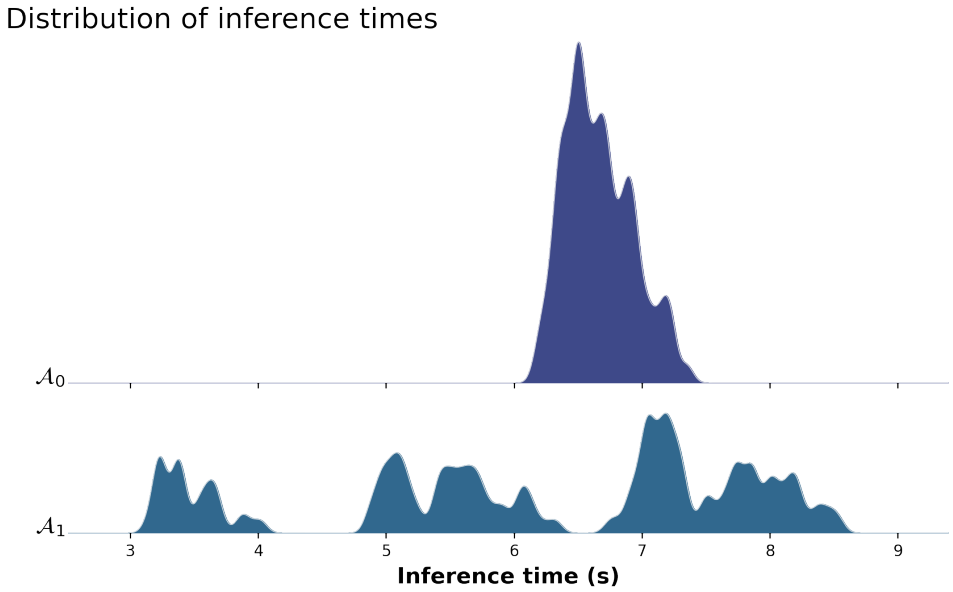

Figure 12: The nesting of clashing concepts affects the spread of inference times.

Moreover, we expect polynomial and linear time scaling of the inference time for the sets of ABoxes used to show these respective scalings for the simulation time in sections 5.3.2 and 5.3.3.

6.1.7 Spreading

As mentioned earlier, the run is the order of formulas that are created in inference steps by SHARP. Some ABoxes allow multiple different runs. For inconsistent ABoxes these runs generally correspond to different inference times: a clash can be found quickly, or more slowly if formulas that do not give rise to the clash are inspected in between. It can therefore be expected that inconsistent ABoxes that allow for multiple different runs have a greater spread in inference times than concept-equivalent ABoxes that allow fewer runs. This effect can be demonstrated by applying SHARP to the following ABoxes.

- $\mathcal{A}_0 = \{a : (A \sqcap (B \sqcap (C \sqcap (D \sqcap \neg A))))\}$,
- $\mathcal{A}_1 = \{a : (A \sqcap B),\ a : (B \sqcap C),\ a : (C \sqcap D),\ a : (D \sqcap \neg A)\}$.

The ABoxes are concept-equivalent, but the second one allows for multiple different runs, whereas the first one allows for only one run.

In Figure 12, the second ABox shows indeed more spread in inference time than the first. The standard deviation of the distribution corresponding to $\mathcal{A}_0$

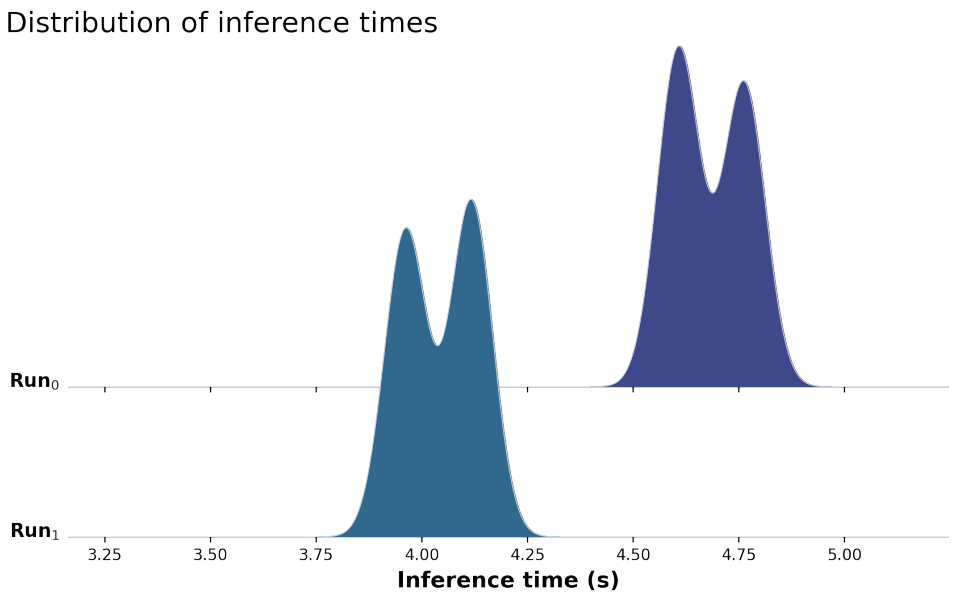

Figure 13: Making inferences on existential restrictions before universal restrictions is faster.

is 0.31 seconds, whereas the standard deviation of the $\mathcal{A}_1$ distribution is 1.60 seconds.

6.1.8 $\exists$ before $\forall$

SHARP does not conclude anything from a universal restriction $a : \forall r.C$ if there is no corresponding role formula $(a, b) : r$ in its memory. So if the role formula is concluded from an existential restriction $a : \exists r.D$, the order in which SHARP inspects $a : \forall r.C$ and $a : \exists r.D$ affects the inference time. Inspecting the existential restriction before the universal restriction therefore reduces the inference time.

We demonstrate this with the following ABox: $\mathcal{A} = \{a : \forall r.A,\ a : \exists r.\neg A\}$, for which the simulation results for its two runs are shown in Figure 13.

Run_0 is the run where the universal restriction is inspected first, then the existential restriction, and then the universal restriction again to derive the formula $x : A$, where x was introduced in the previous inference step.

Comparing Run_0 with Run_1, we get $\Delta(\mathsf{IT}_{\mathsf{Run}_0}, \mathsf{IT}_{\mathsf{Run}_1}) = 16.3\%$, with the absolute difference being 0.66 seconds.

It will be difficult to verify this effect experimentally, because the order of inspected formulas is likely not an easy thing to measure. This order might be influenced by the order in which the formulas are presented to the subject, but

this is mere speculation at this point, as based on SHARP, the order does not affect the reasoning process.

6.1.9 The role of decay

The more recently a formula is derived the less its chunk's activation has decayed. This means that recently derived formulas have chunks with a relatively high activation, meaning that they are more likely to be retrieved and are retrieved more quickly. We expect this mechanism to give different inference times for different runs. The decrease of a chunk's activation is determined by the decay parameter, the true value of which we don't know, so that the size of these effects is difficult to predict.

To demonstrate this effect, we ran SHARP on the following ABox:

$$\mathcal{A}_0 = \{a : (A \sqcap^{\alpha} (B \sqcap^{\beta} C)),\ a : (D \sqcap^{\gamma} (E \sqcap^{\delta} \neg C))\}$$

We have labelled the connectives to define the runs; the greek letters designate the connectives that the inference steps are made on.

For $\mathcal{A}_0$, we have the following runs:

- $\mathsf{Run}_0 = \alpha$ - β - γ - δ
- $\mathsf{Run}_1 = \alpha$ - γ - δ - β
- $\mathsf{Run}_2 = \alpha$ - γ - β - δ
- $\mathsf{Run}_3 = \gamma$ - δ - α - β
- $\mathsf{Run}_4 = \gamma$ - α - β - δ
- $\mathsf{Run}_5 = \gamma$ - α - δ - β

To compare the runs, we use the following definition.

Definition 6.1.2. Let ϕ or $\{\phi_0, \phi_1\}$ be the result of an inference step SHARP made, where ϕ and at least one of ϕ_0, ϕ_1 is a complex formula. If SHARP makes the next inference step on a formula ψ different from ϕ, ϕ_0 or ϕ_1, we say that SHARP made a *switch*.

This definition captures the situation of SHARP reasoning with a formula different from one it has just derived.

We ran 3994 simulations and categorised the data based on the number of switches. So we group together the data from Run_0 and Run_3 (with zero switches), as well as the data from Run_1 and Run_4 (with one switch) and also the data from Run_2 and Run_5 (with two switches).

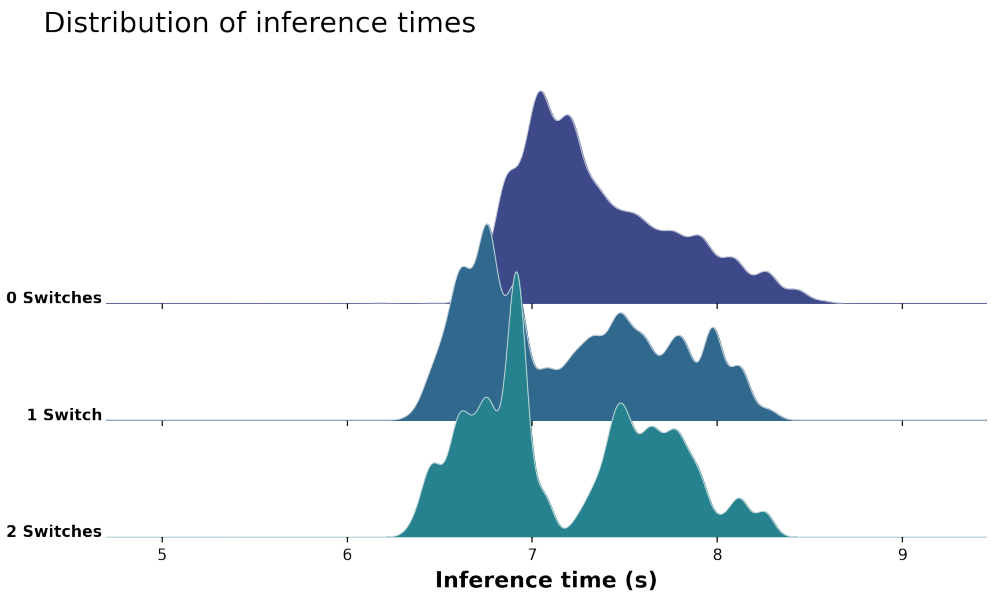

Figure 14: The number of switches only slightly affects inference time, but it affects the likelihood of the runs quite profoundly (not displayed in graph)

The groups are rather different in size: 3357 (84.1%) for zero, 529 (13.2%) for one, 108 (2.7%) for two switches.

So in short, the more switches, the less likely the run. This effect should be attributed to the decay, because only the decay causes differences in the activation of the chunks, which in turn determines the retrieval probability of the chunk.

This effect will be difficult to test experimentally, as the order of the formulas inspected can probably only be obtained from introspection, the accuracy of which is questionable.

Figure 14 shows that different runs also have moderately different inference times. We find: $\Delta(\mathsf{IT}_{\mathsf{0Switches}}, \mathsf{IT}_{\mathsf{1Switch}}) = 0.1\%$, with the absolute difference being 0.008 seconds, meaning groups 0 and 1 amount to the same inference times. And we get $\Delta(\mathsf{IT}_{\mathsf{0Switches}}, \mathsf{IT}_{\mathsf{2Switches}}) = 2.1\%$, with the absolute difference being 0.15 seconds.

The difference in time is not caused by the decay, but by the fact that Module1 searches through different numbers of formulas in the different runs. More specifically, in Run_0, after α Module1 only looks at one atom, namely: $a : A$. After β it looks at three atoms, after γ it looks at four, and after δ it looks at six. This totals to 14 atoms inspected in Module1. On the other hand, in Run_1, after α Module1 looks at one formula, then after γ it looks at two atoms, after δ it looks

at four, and after β it looks at six. This number totals only 13; the same holds for Run_2 and similarly for the other three runs. So because the atoms are derived later, Module1 has to look through fewer atoms in total, resulting in a shorter inference time.

Under some assumptions we can estimate the size of the effect the decay has on inference time. For a default Latency factor of $F = 1.0$ and the used decay value of $d = 0.005$, two chunks that are retrieved at times $t_1 = 1.0$ and $t_2 = 2.0$ respectively have, according to $A_i = \ln t_i^{-d}$, activations: $A_1 = 0.0$ and $A_2 = -0.00347$. This means that their retrieval times, according to $T_i = Fe^{-A_i}$ are: $T_1 = 1.0$ s and $T_2 = 0.997$ s, so that the absolute difference in their retrieval times is: $\Delta T = 0.003$, i.e. too small to be of interest to us. This difference is also smaller than the statistical accuracy with which we estimate inference times. This statistical accuracy can be estimated, under some further assumptions. If the data is distributed according to a gaussian, the standard error of the mean is given by $\sigma_{\text{SEM}} = \frac{\sigma}{\sqrt{n}}$, where n is the sample size. If we want σ_{SEM} to equal $\Delta T = 0.003$, assuming the standard deviation is typically 0.5 s, we need a sample size of $n = (\frac{0.5}{0.003})^2 \approx 27800$, which is bigger than our samples.

It should be noted that the size of this effect, to a first order approximation, scales linearly with the decay. For example, if we consider chunks that have had only one presentation, the retrieval time difference can be approximated linearly by the formula: $T_2 - T_1 = F(e^{A_2} - e^{A_1}) = F(t_2^{-d} - t_1^{-d}) \approx F\frac{d}{dt}(t^{-d})\big|_{t_2}\Delta t = -Fdt_2^{-d-1}\Delta t$. Considering that the default value of d is much higher than the value we used, it is then expected that the effect of inference time depending on run is more pronounced: in the above example, with a decay value of 0.5, we would get an absolute retrieval time difference of 0.29 s.

6.2 Complexity measures

The complexity of the ABox consistency problem can be analysed in two ways: by using some notion of proofs, or by using no such thing. We call the first kind *proof-complexity* measures and the second kind *entailment-complexity* measures.

A proof-complexity measure can be used to define a complexity measure on entailments by calculating some aggregate score (e.g. the mean) from (a subset of) all of the proof-complexity values of the entailment's proofs. Note that we don't call such a complexity measure on entailments an entailment-complexity measure, as it uses the notion of a proof in some way. Conversely, an entailment-complexity measure can be used to define a proof-complexity measure by looking at the entailment of the proof, abstracting away from the proof's deduction rule

applications.

Several different complexity measures exist in the literature (Alrabbaa et al., 2020a), although it is not immediately clear if they accurately measure the *cognitive* complexity; they are briefly discussed in the following. Then two complexity measures based on SHARP are defined. We hypothesize that SHARP, because it is designed to simulate human reasoning, will yield a complexity measure that accurately estimates the cognitive complexity of an ABox inconsistency problem.

6.2.1 Naive measures

These naive measures (Strannegård et al., 2013) are easy to compute and are primarily based on simple features of the ABoxes.

- $\mathsf{size}(\mathcal{A})$, the size of $\mathcal{A}$ as defined in Definition 4.1.1,
- $\#\exists\forall(\mathcal{A})$, the number of restrictions that appear in $\mathcal{A}$,
- $\#\neg(\mathcal{A})$, the number of negations that appear in $\mathcal{A}$,
- $\#\mathsf{Elm}(\mathcal{A})$, the number of elements in a model of $\mathcal{A}$,
- $\#\mathsf{Rel}(\mathcal{A})$, the number of relations in a model of $\mathcal{A}$.

These measures are all entailment-complexity measures, as no formal proofs are used in their definitions.

6.2.2 Measures based on formal proof

We found two proof-complexity measures in the literature (Strannegård et al. (2013), Section 6). They were originally defined for other logics, but can be easily adapted to $\mathcal{ALE}$):

- $\mathsf{Len}_{prf}(\mathcal{A})$, the proof length, i.e. the number of deduction steps in the shortest proof.
- $\mathsf{Size}_{prf}(\mathcal{A})$, the proof size, i.e. $\Sigma_i\mathsf{Size}(\phi_i)$, where i numbers each formula in the shortest proof and in case of multiple shortest proofs, $\mathsf{Size}_{prf}(\mathcal{A})$ is minimised.

The idea of these measures is that the human reasoner forms the proof in their mind, so that the cognitive complexity scales with the complexity of that proof, regardless of whether it is the length, or the size of that proof.

6.2.3 Least-time

For each run, the (arithmetic) mean inference time is computed; the least of these forms the *least-time complexity measure* $\mathrm{CM}^{\mathrm{SRP}}_{lst}$. This is a proof-complexity

measure, because it is based on a run. This measure is well-defined because the logistic distribution has a well-defined mean and the mean of a linear combination of random variables is well-defined; SHARP's predicted inference times are distributed according to a linear combination of logistic distributions. SHARP, being a simulation, can only be used to estimate the least-time complexity value, rather than compute it. By the law of large numbers, this measure can be approximated well: the estimate's precision scales with $\frac{1}{\sqrt{n}}$, where n is the number of simulations. The measure thus defined assumes that the inconsistency problem is not more difficult than the cognitively easiest way of solving it, where again cognitively easiest means that it takes least inference time. This measure is therefore suitable for modelling human reasoning in which clever heuristics are used.

6.2.4 Average-time

The *average-time complexity measure* $\mathrm{CM}^{\mathrm{SRP}}_{avg}$ estimates the complexity of an ABox inconsistency problem by SHARP's (arithmetic) mean inference time. As above, this complexity measure is well-defined. It is a proof-complexity measure, as it is dependent on runs. In case of multiple runs with each a different average inference time, each run contributes to the cognitive complexity value, weighted by the likelihood of the run. This complexity measure models human reasoning in which clever heuristics play a less prominent role.

6.2.5 Linear Combination

Another complexity measure in (Strannegård et al., 2013) is: $\mathrm{CM}_{\mathrm{lc}}$, a linear combination of $\mathsf{size}(\mathcal{A})$, $\#\mathsf{Elm}(\mathcal{A})$ and the product $\mathsf{size}(\mathcal{A}) \cdot \#\mathsf{Elm}(\mathcal{A})$, with coefficients fitted to data.

More formally, $\mathrm{CM}_{\mathrm{lc}} = \beta^0_{\mathrm{lc}} + \beta^1_{\mathrm{lc}} \cdot \mathsf{size}(\mathcal{A}) + \beta^2_{\mathrm{lc}} \cdot \#\mathsf{Elm}(\mathcal{A}) + \beta^3_{\mathrm{lc}} \cdot \mathsf{size}(\mathcal{A}) \cdot \#\mathsf{Elm}(\mathcal{A})$, with β^i_{lc} (for $i \in \{0, 1, 2, 3\}$) the coefficients to be estimated. We fitted these coefficients to the average-time complexity and got an adjusted R-squared of 0.575 with coefficient values $\beta^0_{\mathrm{lc}} = -0.955$, $\beta^1_{\mathrm{lc}} = 0.631$, $\beta^2_{\mathrm{lc}} = 0.650$, and $\beta^3_{\mathrm{lc}} = -0.0311$. An assumption is here that the average-time complexity correlates sufficiently with empirical data, so that it can be thought of as fitted to empirical data.

Note that $\mathrm{CM}_{\mathrm{lc}}$ is an entailment-complexity measure because no proofs are used in its definition; that the measure is here fitted to data from a proof-complexity measure is considered irrelevant, because the measure is supposed to be fitted to empirical data. The found coefficients are difficult to interpret when the average-

time complexity is taken as resembling how humans actually reason: somehow the base-level expected time is negative, which means we could positively gain time by solving ABox consistency problems with empty ABoxes; a very strange situation indeed.

6.2.6 Average Inference Step time (AIS)

This complexity measure is based on the idea that each deduction rule takes a certain average time to complete; it is a proof-complexity measure. The inference time for a certain ABox can then be approximated by summing the average times of the deduction rules used in the proof. More precisely, the $\sqcap$-rule, the $\exists$-rule and the $\forall$-rule are performed in average times $t_\sqcap$, $t_\exists$ and $t_\forall$ respectively. The formula:

$$T_{\text{AIS}} = t_0 + t_\sqcap N_\sqcap + t_\exists N_\exists + t_\forall N_\forall$$

Then gives the approximated inference time. Here, N_i is the (average) number of times the i-rule is used in the proof(s) for $i \in \{\sqcap, \exists, \forall\}$ and t_0 is interpreted as the time it takes to find a clash if no inference steps have to be performed.

If for example a certain proof is formed by one $\exists$-rule-application and one $\forall$-rule-application (i.e. $N_\sqcap = 0$, $N_\exists = 1$, $N_\forall = 1$), the time of this deduction is approximated by $t = t_0 + t_\exists + t_\forall$.

This complexity measure does not take into account that the time a certain deduction rule application takes might depend on the stage of solving the problem nor on the formula it is applied to.

The parameters t_0, $t_\sqcap$, $t_\exists$ and $t_\forall$ are approximated by a multilinear regression to the average-time complexity. In case an entailment allows multiple different runs, the (arithmetic) mean number of rule applications are used for the fit, without using weights for the likelihood of the runs.

The SKlearn library was used for calculating the statistics; its linear regression function minimises the residual sum of squares between the model and the data. Again we assume that the average-time complexity correlates well with empirical data, so that this linear combination would give similar results as when it is fitted to empirical data. The coefficients found are: $t_0 = 3.62$ s, $t_\sqcap = 1.59$ s, $t_\exists = 0.941$ s and $t_\forall = 0.270$ s.

To determine the quality of the fit, we compute the adjusted coefficient of determination (adjusted R-squared) as follows:

$$R^2_{adj} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1},$$

where N is the sample size and p is the degrees of freedom, i.e. the number of parameters in the fit. The adjusted R-squared is smaller than or equal to the R-squared and corrects for the number of parameters in the fit. Adding more parameters does not automatically give a better adjusted R-squared (even though it always will give a better R-squared). We can report an adjusted R-squared of: $R^2_{adj} = 0.770$, indicating that the AIS complexity measure is only able to replicate the average-time complexity to a moderate extent.

It seems unlikely that the ⊓-rule takes more time than the ∀-rule, but that it would be almost 6 times longer is truly hard to believe; so although this measure is easy to interpret in principle, the concrete values it yields are problematic explainability-wise.

This unexpected difference is caused by many runs invoking the ∀-rule more often than needed. Namely, SHARP can at any time execute the ∀-rule, even if there is no corresponding role formula so that it does not yield anything. Some runs therefore have many ∀-rule-applications with each one contributing little to the inference time.

6.2.7 SHARP simplified

It is desirable to have an accurate and simple-to-calculate complexity measure that closely resembles the least- or average-time complexity measures above. The least- and average-time complexity measures are not accurately estimated by the other measures, as can be seen in the table below. We therefore tried to find a linear combination of certain syntactic features that gives the highest adjusted R-squared value. This means we optimised for both accurate prediction and having few features.

This was done by calculating the adjusted R squared for the following list of features: number of conjunctions, number of existential restrictions, number of universal restrictions, number of total restrictions, number of clashes, number of elements in a model, size, the product of size and number of elements, number of roles, number of negations, proof lengths. The least predictive features were then removed from the fit one by one until removing further features decreased the adjusted R squared. The result is the following linear combination of the number of conjunctions, size, number of quantifiers and proof length (this last one makes the measure a proof-complexity one):

$$\mathrm{CM}^{\mathrm{SRP}}_{smp} = \beta_0 + \beta_1 \cdot N_{\sqcap} + \beta_2 \cdot \mathsf{size} + \beta_3 \cdot \#\exists\forall + \beta_4 \cdot \mathsf{Len}_{prf},$$

with the following coefficient values: $\beta_0 = 2.70\,\mathrm{s}$, $\beta_1 = 0.756$, $\beta_2 = 0.751$, $\beta_3 = -0.429$, $\beta_4 = 0.827$.

We can report an adjusted R-squared of 0.844, which is not surprisingly a substantial improvement in comparison with the linear combination and AIS. Even so, this measure leaves a significant gap between its values and the ones it is designed to estimate. Moreover, $\mathrm{CM}^{\mathrm{SRP}}_{smp}$ is difficult to interpret: the negative coefficient for the number of quantifiers makes explanation especially hard, but it is also difficult to explain why it is precisely these syntactic features that are relevant and how they combine to give the complexity value.

6.2.8 Comparing different measures

To compare the different complexity measures, it would not suffice to compare their values on one ABox only: a set of ABoxes is needed that somehow forms a representative sample of the description logic $\mathcal{ALE}$. To draw a representative sample from the population of ABoxes is difficult, as this population is infinite and has arbitrarily long formulas and ABoxes. We are not equally interested in every ABox, so some practical boundaries have to be drawn. Inspired by the technique of stratified sampling (Thompson, 2020), we attempt to create a sample representative of $\mathcal{ALE}$ in two steps. As a first step, we generated 30 quintuples $(X_1, X_2, X_3, X_4, X_5)$ of random natural numbers. The numbers were all distributed uniformly, but in different ranges, namely X_1 in range (3-20), X_2 in range (0-5), X_3 in range (0-5), X_4 in range (0-2) and X_5 in range (0-1). These random variables are then interpreted as follows:

- X_1: the size of an ABox
- X_2: the number of existential and universal restrictions occuring in an ABox
- X_3: the number of negations occuring in an ABox
- X_4: the degree of superfluousness, if 0 then every formula is needed to derive the clash, if 1 a third of the formulas is not needed to derive the clash, if 2, half of the formulas are not needed to derive the clash
- X_5: the number of clashes in the ABox

Step two is to construct an ABox that satisfies the constraints of the tuple, a construction which is still somewhat arbitrary. Moreover, in some cases no ABox could be found, e.g. there is no ABox satisfying $X_1 = 3$ and $X_3 = 5$; the created ABox was then chosen to stay as close to the constraints as possible. We

hope that this two step approach yielded a sample that can be considered in some sense representative for the logic $\mathcal{ALE}$. The result of the above process is, where a $*$ indicates inconsistency:

- $\mathcal{A}_0 = \{a : A,\ b : \neg A,\ a : \neg B\}$
- $\mathcal{A}_1 = \{a : \exists r.\forall s.\exists r.\forall t.\neg A\}$
- $\mathcal{A}_2^* = \{a : A,\ a : \neg A,\ b : \forall r.A\}$
- $\mathcal{A}_3 = \{a : (A \sqcap (\neg B \sqcap (C \sqcap D))),\ a : D\}$
- $\mathcal{A}_4 = \{a : \forall r.\exists s.\exists r.\neg A,\ b : \forall s.\neg B\}$
- $\mathcal{A}_5 = \{a : (A \sqcap \neg B),\ b : (\neg C \sqcap \neg D),\ a : \neg C\}$
- $\mathcal{A}_6 = \{a : \forall r.\exists s.(\neg A \sqcap \neg B),\ b : \neg A,\ c : \neg B\}$
- $\mathcal{A}_7^* = \{a : \exists r.A,\ a : \forall r.\neg A,\ b : \forall s.(B \sqcap A)\}$
- $\mathcal{A}_8^* = \{a : \exists r.\exists s.\neg A,\ \forall r.\forall s.A,\ \exists s.\neg B\}$
- $\mathcal{A}_9 = \{a : \neg A,\ a : \neg B,\ b : \neg C,\ b : \neg A,\ c : \neg A, c : B\}$
- $\mathcal{A}_{10}^* = \{b : \forall r.\forall s.\exists s.\neg A,\ (a, b) : r,\ b : \exists s.\forall s.A\}$
- $\mathcal{A}_{11}^* = \{a : \exists r.(A \sqcap (B \sqcap \neg A)),\ b : \forall r.\exists s.\exists t.(B \sqcap \neg C)\}$
- $\mathcal{A}_{12} = \{b : \neg D,\ (a, b) : r,\ a : \exists s.(\neg A \sqcap (\neg B \sqcap (\neg C \sqcap D)))\}$
- $\mathcal{A}_{13}^* = \{a : \forall r.\exists s.\neg B,\ b : \forall s.(\neg A \sqcap (\neg C \sqcap B)),\ (a, b) : r\}$
- $\mathcal{A}_{14}^* = \{a : \forall r \exists s.\neg B,\ (a, b) : r,\ b : \forall s.B,\ a : \neg A,\ b : B\}$
- $\mathcal{A}_{15}^* = \{a : \exists r.A,\ a : \forall r.\neg A,\ b : \exists s.B,\ b : \forall s.A,\ (b, a) : s\}$
- $\mathcal{A}_{16}^* = \{a : \exists r.\neg A,\ a : \forall r.(A \sqcap \neg B),\ b : \exists r.\neg B,\ a : \neg C,\ a : B\}$
- $\mathcal{A}_{17} = \{a : (\neg A \sqcap (B \sqcap \neg C)),\ b : (\neg B \sqcap (C \sqcap A)),\ b : (A \sqcap C),\ a : B\}$
- $\mathcal{A}_{18} = \{a : \forall r.\forall s.(A \sqcap \neg B),\ (a, b) : r,\ (b, c) : s,\ c : A,\ b : B\}$
- $\mathcal{A}_{19} = \{a : \forall r.\exists s.\forall t.\neg B,\ b : (\neg C \sqcap \exists r.(\neg B \sqcap A)),\ c : \exists s.\exists t.B\}$
- $\mathcal{A}_{20}^* = \{a : \forall r.\exists s.\exists t.(\neg A \sqcap B),\ (a, b) : r,\ b : \forall s.\forall t.(\neg B \sqcap \neg C)\}$
- $\mathcal{A}_{21}^* = \{a : (A \sqcap \neg B),\ b : (C \sqcap \neg A),\ a : (\neg B \sqcap (C \sqcap \neg A)),\ b : (C \sqcap (D \sqcap B))\}$
- $\mathcal{A}_{22}^* = \{a : \forall r.\exists r.\forall s.\neg A,\ (a, b) : r,\ b : \forall r \exists s.(\neg B \sqcap (\neg D \sqcap (\neg C \sqcap A)))\}$
- $\mathcal{A}_{23}^* = \{a : \forall r.\exists r.\exists s.\exists t.(\neg A \sqcap (\neg C \sqcap (\neg B \sqcap (\neg D \sqcap (\neg E \sqcap A))))),\ (a, b) : r\}$
- $\mathcal{A}_{24}^* = \{a : \forall r.(A \sqcap (B \sqcap \neg C)),\ (a, c) : r,\ c : (\neg B \sqcap \neg C),\ b : \neg A,\ b : \exists r.A\}$
- $\mathcal{A}_{25} = \{a : \forall r.(A \sqcap \neg B),\ b : \exists r.(A \sqcap (B \sqcap C)),\ c : (A \sqcap B),\ (b, c) : s,\ (c, d) : r\}$
- $\mathcal{A}_{26} = \{a : \exists r.(A \sqcap \neg B),\ b : \forall s.\exists t.\neg C,\ c : \forall r.(B \sqcap \neg A),\ b : (C \sqcap \neg D),\ (a, b) : s\}$
- $\mathcal{A}_{27} = \{a : \forall r.(B \sqcap \forall s.\neg A),\ (a, b) : r,\ b : \exists s.(A \sqcap B),\ b : \forall r.(\neg A \sqcap (B \sqcap \forall s.A))\}$
- $\mathcal{A}_{28}^* = \{a : \forall r.\forall s.(A \sqcap \neg B),\ (a, b) : r,\ (b, c) : s,\ c : B,\ b : \exists r.\exists r.(A \sqcap B),\ b : \forall s.B\}$

- $\mathcal{A}^{*}_{29} = \{a : \forall s.(\neg A \sqcap B),\ (b, a) : r,\ b : \forall r.\exists s.(A \sqcap B),\ c : (\neg B \sqcap C),\ a : (B \sqcap \neg C),\ (b, c) : s\}$

In the tables below the complexity measures can be compared; the first one contains the entailment-complexity measures and the second one the proof-complexity measures.

From the tables we can conclude that the simpler complexity measures on the ABox inconsistency problem seem to make rather coarse distinctions and seem not to correlate much with $\mathrm{CM}^{\mathrm{SRP}}_{avg}$ or $\mathrm{CM}^{\mathrm{SRP}}_{lst}$. Under the assumption that $\mathrm{CM}^{\mathrm{SRP}}_{avg}$ and $\mathrm{CM}^{\mathrm{SRP}}_{lst}$ correlate well with the cognitive complexity, the simpler complexity measures seem unsuitable for estimating the latter. If we, moreover, take into account the parameter values we get from the fits, then the explanatory value, even for AIS, is still more reduced as the results are absurd, however high the quality of the fit.

This means, firstly, that SHARP simulates processes in a way that is not easily reproduced by simple calculations. Secondly, the SHARP complexity measures stand out in their explanatory value compared to the other complexity measures as they both are easy to interpret.

ABox	size	#∃∀	#¬	#Elm	#Rel	CM_{lc}
$\mathcal{A}_0$	5	0	2	2	0	3.19
$\mathcal{A}_1$	6	4	1	1	0	3.29
$\mathcal{A}_2$	5	1	1	2	0	3.19
$\mathcal{A}_3$	9	0	1	1	0	5.09
$\mathcal{A}_4$	8	4	2	2	0	4.90
$\mathcal{A}_5$	11	0	4	2	0	6.60
$\mathcal{A}_6$	11	2	4	3	0	6.91
$\mathcal{A}_7$	9	3	1	3	1	5.83
$\mathcal{A}_8$	10	5	2	5	3	7.05
$\mathcal{A}_9$	11	0	5	3	0	6.91
$\mathcal{A}_{10}$	9	5	1	4	3	6.20
$\mathcal{A}_{11}$	14	4	2	3	1	8.52
$\mathcal{A}_{12}$	14	1	4	3	2	8.52
$\mathcal{A}_{13}$	13	3	3	3	2	7.98
$\mathcal{A}_{14}$	10	3	2	3	2	6.37
$\mathcal{A}_{15}$	10	4	1	4	3	6.71
$\mathcal{A}_{16}$	15	3	5	4	2	9.24
$\mathcal{A}_{17}$	17	0	3	2	0	10.0
$\mathcal{A}_{18}$	10	2	1	3	2	6.37
$\mathcal{A}_{19}$	16	6	3	6	3	10.1
$\mathcal{A}_{20}$	15	5	3	4	3	9.24
$\mathcal{A}_{21}$	20	0	4	2	0	11.7
$\mathcal{A}_{22}$	18	5	4	4	3	10.8
$\mathcal{A}_{23}$	21	4	5	5	4	12.3
$\mathcal{A}_{24}$	17	2	4	4	2	10.3
$\mathcal{A}_{25}$	16	2	1	5	3	9.90
$\mathcal{A}_{26}$	19	4	4	4	2	11.3
$\mathcal{A}_{27}$	19	5	2	3	2	11.2
$\mathcal{A}_{28}$	16	5	1	5	4	9.90
$\mathcal{A}_{29}$	20	3	3	4	3	11.8

Table 1: The entailment-complexity measures

ABox	Len_{prf}	Size_{prf}	$\mathrm{CM}^{\mathrm{SRP}}_{avg}$	$\mathrm{CM}^{\mathrm{SRP}}_{lst}$	T_{ps}	$\mathrm{CM}^{\mathrm{SRP}}_{smp}$
$\mathcal{A}_0$	0	0	3.56	3.56	3.62	4.21
$\mathcal{A}_1$	1	6	3.08	3.06	4.83	2.57
$\mathcal{A}_2$	0	3	0.853	0.853	3.62	3.78
$\mathcal{A}_3$	3	17	8.66	8.66	8.38	8.20
$\mathcal{A}_4$	0	0	2.62	2.61	4.16	2.49
$\mathcal{A}_5$	2	9	7.83	7.81	6.79	7.37
$\mathcal{A}_6$	0	0	3.59	3.59	3.89	4.10
$\mathcal{A}_7$	2	9	4.92	4.34	5.46	5.32
$\mathcal{A}_8$	4	17	7.80	5.89	7.25	7.62
$\mathcal{A}_9$	0	0	5.54	5.54	3.62	4.96
$\mathcal{A}_{10}$	5	23	8.20	7.44	7.42	7.70
$\mathcal{A}_{11}$	3	17	4.88	3.56	8.16	7.23
$\mathcal{A}_{12}$	4	32	10.2	10.2	9.32	10.1
$\mathcal{A}_{13}$	5	28	8.04	7.53	9.08	9.32
$\mathcal{A}_{14}$	4	14	7.78	7.76	5.64	6.98
$\mathcal{A}_{15}$	2	9	8.01	5.47	6.47	5.65
$\mathcal{A}_{16}$	3	16	8.31	6.00	7.45	7.66
$\mathcal{A}_{17}$	5	23	14.7	14.0	11.5	12.1
$\mathcal{A}_{18}$	3	20	7.82	7.81	6.28	7.34
$\mathcal{A}_{19}$	5	22	11.4	9.60	10.6	10.3
$\mathcal{A}_{20}$	7	46	9.11	7.95	8.73	10.1
$\mathcal{A}_{21}$	3	18	10.0	4.45	11.7	10.5
$\mathcal{A}_{22}$	8	62	12.5	10.8	12.9	12.4
$\mathcal{A}_{23}$	9	138	14.2	12.5	15.6	16.0
$\mathcal{A}_{24}$	4	26	9.71	6.66	9.70	10.4
$\mathcal{A}_{25}$	4	17	11.1	9.60	10.1	11.2
$\mathcal{A}_{26}$	3	13	9.89	8.10	9.22	7.98
$\mathcal{A}_{27}$	5	27	9.18	6.68	9.46	8.46
$\mathcal{A}_{28}$	3	20	6.56	4.64	8.89	7.97
$\mathcal{A}_{29}$	5	26	11.1	6.90	11.5	11.2

Table 2: The proof-complexity measures

7 Conclusion

Explanations are essential elements in many areas of human experience. It is difficult to define what makes a good explanation, but we proposed as a way forward the criterion of cognitive load: an explanation is best when it explains the fact by posing the lowest cognitive load on the user. With this idea in mind we looked at knowledge base debugging – a widely occurring activity in industry that is often deemed difficult, even for experienced users – and found much literature on how to optimise this debugging process, although connections with cognitive theory are often weak. This prompted the exploration of using cognitive architectures to model the reasoning process that takes place during debugging. It resulted in the ACT-R model SHARP which simulates symbolic human reasoning, more specifically: the $\mathcal{ALE}$ ABox inconsistency algorithm. With the model we can predict certain (qualitative) effects such as: renaming elements and concepts does not affect inference time while renaming roles does; certain formulas show a higher spread in inference times; reasoning from an existential restriction before a universal restriction is more time-efficient; and more. Quantitative effects are more difficult to predict in absence of empirical data for fine-tuning the model's subsymbolic parameters.

Based on this model we defined two different complexity measures on logical entailments. It is expected that these complexity measures estimate the cognitive complexity of the $\mathcal{ALE}$ ABox inconsistency task to a reasonable extent. It seemed difficult to reproduce the same results of the measures by simpler calculations, showing that SHARP's simulations are non-trivial.

The approach taken seems unique in that it forges a strong connection between the symbolic reasoning necessary for debugging of knowledge bases on the one hand and cognitive psychology on the other. Moreover, the use of a cognitive architecture for this purpose is unique and can be considered an improvement over using other theories of reasoning such as the mental models theory and the mental rules theory for two reasons: claims about the inner workings of the reasoning process are more explicit, and the predictions made with the model are more precise. In creating the model it was necessary to violate a basic recommendation well-known in the cognitive modelling community: that chunks should have a small number of slots. In fact, SHARP uses chunks with a number of slots too

great to be considered realistic. These large chunks were, however, considered necessary for modelling the task at hand. This could on the one hand indicate that logic is very difficult for humans, but on the other hand it could indicate that the cognitive architecture ACT-R is unable to model a least one basic cognitive mechanism.

SHARP is not claimed to be a perfect model; far from it, as there are many ways in which SHARP may be improved. One such way is that the subsymbolic parameters can (and should) be optimised to fit empirical data. Currently, in absence of empirical data, it is very difficult to judge whether simulation data output by SHARP is empirically accurate. In the future, experiments are planned to yield empirical data such that these subsymbolic parameters can be fine-tuned. A further possible improvement is that rewards could be used to make some reasoning steps easier than others; this would reflect better what we see in practice. Lastly, visual processes could be incorporated in the simulation. The ACT-R architecture allows for this, but in creating the current version of SHARP none such processes were considered. It would allow modelling how the order of presenting the formulas in the given ABox might affect inference time (SHARP currently and counterintuitively shows no such order effect).

The model can also be extended in certain ways. Stronger logics than $\mathcal{ALE}$ can be considered for this, although one has to keep in mind that not all description logics can be modelled, as discussed earlier. Among the extensions considered in Section 5.3.4 are number restrictions, various role constructors and TBox formulas. The latter might be the most desired extension, as much literature on description logic reasoning deals with TBoxes. It is expected that for this, a construction is necessary similar to the one in Module 5 for modelling reasoning with universal restrictions, as TBox formulas can never be discarded.

Another way in which SHARP may be extended is by using deep rules of inference. Such rules may lie closer to how humans reason and could therefore improve performance of the model.

As mentioned before, an experiment is planned to verify the predictions made with SHARP; a draft of its design can be found in the Appendix. The accuracy of SHARP will be determined after fitting its output to the experiment's data by fine-tuning the subsymbolic parameters.

The values of any complexity measure based on SHARP are computationally difficult to obtain, so approximation by a simpler model is desired. As linear regression does not seem to be able to approximate SHARP's output accurately – indeed, strong non-linear effects are expected – *symbolic regression* is proposed

as a more viable alternative as, it proved quite succesful in (Udrescu and Tegmark, 2020). In short, this type of regression does not make any assumption on the structure of the desired algebraic expression, only that it *is* an algebraic expression. The loss function to be minimised consists of an error term, which measures the discrepancy with the data, and a complexity term, which measures the complexity of the found algebraic expression. In this way, the trade-off between predictive accuracy and model parsimony is optimised.

Assuming the above will result in a complexity measure on description logic entailments, the measure can be used Horridge's algorithm for finding justification based proofs.

Appendix

The following experimental design is a draft and needs some improvements.

Introduction and Goals

The goal of this experiment is to verify the predictions made with the model SHARP. This will either give us trust in the model, or give specific pointers on how to improve it.

Participants

The sample consists of international participants (both women and men) recruited from the logic master program of the University of Gothenburg, aged between 20 and 45. Most participants have no experience with description logic nor with the description logic $\mathcal{ALE}$ in particular. We offered ... as a reward for participating in the experiment.

Material

To the participants we present a representative selection of ABoxes via a computer sceen using LimeSurvey. The selection of ABoxes should be able to identify all the effects that we predicted with the model SHARP in section 6.1. They are presented in symbolic form to reduce any semantic effects.

Procedure and Tasks

The experiment is conducted in English. Firstly, the participants get an introduction to the description logic $\mathcal{ALE}$. This introduction discusses the syntax and semantics, as well as the syntax expansion rules from section 4.2. Moreover, the notion of a clash is explained and some common pitfalls such as trivial satisfaction of universal restriction formulas. The participants are encouraged to be precise in their reasoning and to make as few mistakes as possible, while not taking too much time to think.

The participants are then asked some basic demographic information. They then get two practice cases to make sure what is expected of them and to reduce learning effects; they are encouraged to ask questions in case they have any. Par-

ticipants can at any moment in between two ABox presentations decide to take a break, to reduce fatigue effects. At the moment of presenting an ABox to the participant, a stopwatch is started and stopped at the moment the participant presses a key: 'i' for *inconsistent*, 'c' for *consistent* and 'n' for *not able to decide*. After each such decision is made, the participant is asked to score the experienced difficulty of the ABox inconsistency task on a Likert-scale. The participants are not allowed to use pen, paper or other tools.

Hypotheses

The hypotheses follow from the simulation results in section 6.1, namely:

- Inference time does not depend on concept or element names,
- Inference time depends on role names,
- The order in which formulas are presented does not affect inference time,
- Whether concepts are negated or not does not affect inference time,
- Inference time is dependent on how concepts are nested, as explained in section 10,
- Inference time scales linearly, polynomially or exponentially with the size of the ABox, depending on which formulas the ABox contains,
- Certain formulas show a spreading effect in inference time.

Apart from the above hypotheses, we are interested in the relationship between inference time and error rate; about this relationship we assume the null-hypothesis that there is no correlation. Two hypotheses made by SHARP are not tested: how inference time is affected by the order of inferring from universal and existential restrictions respectively, and that runs with more switches are less likely to occur. Perhaps a later experiment could be performed to test for these effects.

References

(2023a). Act-r publications and models. `http://act-r.psy.cmu.edu/publication/`. Accessed: 2022-12-15.

(2023b). Act-r software. `http://act-r.psy.cmu.edu/software/`. Accessed: 2022-12-15.

Adler, J. E. and Rips, L. (2008). *Reasoning: Studies of human inference and its foundations*. Cambridge University Press, Cambridge.

Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., and Kovtunova, A. (2020a). On the complexity of finding good proofs for description logic entailments. In *Proceedings of the 33rd International Workshop on Description Logics*, volume 33, pages 1–19. CEUR.

Alrabbaa, C., Baader, F., Borgwardt, S., Kovtunova, A., and Koopmann, P. (2020b). Finding small proofs for description logic entailments: Theory and practice. *LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, 73:32–67. arXiv:2004.08311.

Alrabbaa, C., Borgwardt, S., Hirsch, A., Knieriemen, N., Kovtunova, A., Rothermel, A. M., and Wiehr, F. (2022). In the head of the beholder: Comparing different proof representations.

Anderson, J. R. (2005). Human symbol manipulation within an integrated cognitive architecture. *Cognitive Science*, 29(3):313–341.

Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, Oxford.

Anderson, J. R. and Bower, G. H. (1973). *Human associative memory*. Winston and Sons, Washington.

Anderson, J. R. and Byrne, M. D. (2004). An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060.

Anderson, J. R., Fincham, J. M., Qin, Y., and Stocco, A. (2008). A central circuit of the mind. *Trends in Cognitive Sciences*, 12(4):136–143.

Andrews, A. D. (1993). Mental models and tableau logic. *Behavioural and Brain Sciences*, 16:334.

Andrews, G. and Halford, G. S. (2002). A cognitive complexity metric applied to cognitive development. *Cognitive Psychology*, 45:153–219.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. (2003). *Description Logic Handbook*. Cambridge University Press, Cambridge.

Baader, F., Horrocks, I., Lutz, C., and Sattler, U. (2017). *An introduction to Description Logic*. Cambridge University Press, Cambridge.

Baader, F., Peñaloza, R., and Suntisrivaraporn, B. (2007). Pinpointing in the description logic $\mathcal{EL}$+. *KI 2007*, pages 52–67.

Baader, F. and Peñaloza, R. (2007). Axiom pinpointing in general tableaux. In Olivetti, N., editor, *Automated Reasoning with Analytic Tableaux and Related Methods. TABLEAUX 2007*, volume 4548 of *Lecture Notes in Computer Science()*, pages 11–27, Berlin, Heidelberg. Springer Verlag.

Baader, F. and Peñaloza, R. R. (2010). Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 20.

Baader, F. and Suntisrivaraporn, B. (2008). Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proceedings of the 3rd Knowledge Representation in Medicine (KR-MED'08): Representing and Sharing Knowledge Using SNOMED*, volume 410 of *CEUR-WS*.

Barrouillet, P. and Lecas, J. F. (2000). Illusory inferences from a disjunction of conditionals: a new mental models account. *Cognition*, 76(2):167–173.

Bonatti, L. (1994). Propositional reasoning by model? *Psychological Review*, 101(4):725–733.

Brachman, R. J. and Levesque, H. J. (1984). The tractability of subsumption in frame-based description languages. In *AAAI-84 Proceedings*.

Brachman, R. J. and Schmolze, J. G. (1985). An overview of the kl-one knowledge representation system. *Cognitive Science*, 9:171–216.

Braine, M. D. S. (1993). Mental models cannot exclude mental logic and make little sense without it. *Behavioural and Brain Sciences*, 16:338.

Brasoveanu, A. and Dotlačil, J. (2020). *Computational Cognitive Modeling and Linguistic Theory*, volume 6. Springer Open, Cham, Switzerland.

Bundy, A. (1993). 'semanti procedure' is an oxymoron. *Behavioural and Brain Sciences*, 16:339.

Cohen, L. J. (1993). Some difficulties about deduction. *Behavioural and Brain Sciences*, 16:341.

Colyvan, M. J. (2012). *Introduction to the Philosophy of Mathematics*. Cambridge University Press, Cambridge.

Colyvan, M. J. and McQueen, K. (2018). Two flavours of mathematical explanation. In Reutlinger, A. and Saatsi, J., editors, *Explanation beyond Causation*, pages 231–249. Oxford University Press, Oxford.

Cornet, R. and de Keizer, N. (2008). Forty years of snomed: a literature review. *BMC Medical Informatics and Decision Making*, 8(1).

Cox, J. R. and Griggs, R. A. (1982). The effects of experience in wason's selection task. *Memory and Cognition*, 10(5):496–502.

de Giacomo, G. and Lenzerini, M. (1994). Boosting the correspondence between description logics and propositional dynamic logics. *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 205–212.

Diakopoulos, N. (2014). Algorithmic accountability: Journalistic investigation of computa-

tional power structure. *Digital Journalism*, 3(3):398–415.

Dimov, C., Khader, P. H., Marewski, J. N., and Pachur, T. (2020). How to model the neurocognitive dynamics of decision making: A methodological primer with act-r. *Behavior Research Methods*, 52:857–880.

Donini, F., Hollunder, B., Lenzerini, M., Speccamela, A. M., Nardi, D., and Nutt, W. (1991). The complexity of existential quantification in concept languages. Technical Report RR-91-02, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.

Donini, F., Lenzerini, M., Nardi, D., and Schaerf, A. (1994). Deduction in concept languages from subsumption to instance checking. *Journal of Logic and Computation*, 4(4):423–452.

Engström, F., Nizamani, A. R., and Strannegård, C. (2014). Generating comprehensible explanations in description logic. *27th International Workshop on Description Logics*.

EU (2016). Regulation 2016/679 - protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation).

Feilmayr, C. and Wöß, W. (2016). An analysis of ontologies and their success factors for application to business. *Data and Knowledge Engineering*, 101.

Garnham, A. (1993). Some difficulties about deduction. *Behavioural and Brain Sciences*, 16:350.

Goel, V. (2007). Anatomy of deductive reasoning. *Trends in cognitive sciences*, 1(10):435–441.

Goel, V., Buchel, C., Frith, C., and Dolan, R. J. (2000). Dissociation of mechanisms underlying syllogistic reasoning. *Neuroimage*, 12(5):504–514.

Gruber, T. R. (1993). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5-6).

Halford, G. S., Wilson, W. H., and Phillips, S. (2010). Relational knowledge: the foundation of higher cognition. *Trends in Cognitive Sciences*, 14(11):497–505.

Hanna, G., Jahnke, H. N., and Pulte, H. (2010). *Explanation and Proof in Mathematics. Philosophical and Educational Perspectives*. Springer, New York.

Harari, O. (2008). Proclus' account of explanatory demonstrations in mathematics and its context. *Mathematics and its Context*, 90(2):137–164.

Hayes-Roth, F. and Waterman, D. A. (1983). *Building expert Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Herman, I. (2010). Why owl and not wol?

Hodges, W. (1993). The logical content of theories of deduction. *Behavioural and Brain Sciences*, 16:353.

Horridge, M. (2011). *Justification Based Explanation in Ontologies*. PhD thesis, University of Manchester, Manchester.

Horridge, M., Bail, S., Parsia, B., and Sattler, U. (2013). Toward cognitive support for owl justifications. *Knowledge-Based Systems*, 53.

Horridge, M., Parsia, B., and Sattler, U. (2009). Lemmas for justifications in owl. In Grau, B., Horrocks, I., Motik, B., and Sattler, U., editors, *Proceedings of the 22nd International*

Workshop on Description Logics (DL 2009), volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Horridge, M., Parsia, B., and Sattler, U. (2010). Justification oriented proofs in owl. In *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference*, ISWC.

Horrocks, I., Ruttenberg, A., Hawke, S., and Herman, I. (2023). Owl working group.

Ilkou, E. and Koutraki, M. (2020). Symbolic vs sub-symbolic ai methods: Friends or enemies? In Conrad, S. and Tiddi, I., editors, *CIKMW2020: Proceeding of the CIKM 2020 Workshops*, volume 2699. CEUR.

Johnson-Laird, P. N. (2010). Mental models and human reasoning. *PNAS*, 107(43):18243–18250.

Juvina, I. and Taatgen, N. A. (2009). A repetition-suppression account of between-trial effects in a modified stroop paradigm. *Acta Psychologica*, 131:72–84.

Karplus, K. (1999). Algorithm description.

Kitcher, P. (1989). Explanatory unification and the causal structure of the world. In *Scientific Explanation*, volume XIII, pages 410–505. University of Minnesota Press, Minneapolis.

Kontopoulos, E., Bassiliades, N., and Antoniou, G. (2010). Visualizing semantic web proofs of defeasible logic in the dr-device system. *Knowledge-Based Systems*, 24(3):406–419.

Kotseruba, I. and Tsotsos, J. K. (2020). 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 53:17–94. https://doi.org/10.1007/s10462-018-9646-y.

Kulesza, T., Burnett, M., Wong, W.-K., and Stumpf, S. (2015). Principles of explanatory debugging to personalize interactive machine learning. *Proceedings of the 20th International Conference on Intelligent User Interfaces*, pages 126–137.

Lange, M. (2017). *Because Without Cause: Non-causal Explanations in Science and Mathematics*. Oxford University Press, Oxford.

Lewis, M. W., Milson, R., and Anderson, J. R. (1987). *Artificial Intelligence and Instruction*, chapter The teacher's apprentice: Designing an intelligent authoring system for high school mathematics. Addison-Wesley, Reading, MA.

Lockey, S., Gillespie, N., Holm, D., and Someh, I. A. (2021). A review of trust in artificial intelligence: Challenges, vulnerabilities and future directions. In *Proceedings of the 54th Hawaii International Conference on System Sciences*, HICSS.

Mancosu, P. (2018). Explanation in mathematics. *Stanford Encyclopedia of Philosophy*.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W.H. Freeman, San Francisco, CA.

McGuinness, D. L. (1996). *Explaining Reasoning in Description Logics*. PhD thesis, Rutgers University, New Jersey.

McGuinness, D. L. and van Harmelen, F. (2004). Owl web ontology language overview.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38.

Minsky, M. (1974). A framework for representing knowledge. *The Psychology of Computer*

Vision.

Müller, V. E., Bassiliades, N., and Antoniou, G. (2011). Ethics of artificial intelligence and robotics. *The Stanford Encyclopedia of Philosophy*.

Newell, A. (1992). Précis of unified theories of cognition. *The behavioural and brain sciences*, 15(3):425–437. doi:10.1017/S0140525X00069478.

Newell, A., Shaw, J. C., and Simon, H. A. (1959). report on a general problem solving program.

Newstead, S. E., Handley, S. J., and Buck, E. (1999a). Falsifying menatl models: Testing the predictions of theories of syllogistic reasoning. *Memory and Cognition*, 27(2):344–354.

Newstead, S. E., Handley, S. J., and Buck, E. (1999b). Falsifying mental models: testing the predictions of theories of syllogistic reasoning. *Memory and cognition*, 27(2):344–354.

Nunokawa, K. (2010). Proof, mathematical problem-solving, and explanation in mathematics teaching. In *Explanation and Proof in Mathematics*, pages 223–236. Springer, Boston.

O'Brien, D. P., Braine, M. D. S., and Yang, Y. (1994). Propositional reasoning by mental models? simple to refute in principle and in practice. *Psychological Review*, 101(4):711–724.

PARLIAMENT, T. E. and UNION, T. C. O. T. E. (2016). Regulation (eu) 2016/679 of the european parliament and of the council. Official Journal of the European Union.

Paseau, A. (2010). Proofs of the compactness theorem. *History and Philosophy of Logic*, 31:73–98.

Petrillo, F., Soh, Z., Khomh, F., Pimenta, M., Freitas, C., and Guéhéneuc, Y.-G. (2016). Towards understanding interactive debugging. *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*.

Peñaloza, R. (2019). Explaining axiom pinpointing. *Description Logic, Theory Combination, and All That*.

Peñaloza, R. and Sertkaya, B. (2017). Understanding the complexity of axiom pinpointing in lightweight description logics. *Artificial Intelligence*, 250.

Powers, D. M. W. (1983). Robot intelligence. *School of Electrical Engineering*. unable to find.

Powers, D. M. W. (1990). Bibliographies and literature reviews: Goals, issues and directions in machine learning of natural language and ontology. *ACM SIGART Bulletin*.

Raussen, M. and Skau, C. (2004). Interview with michael atiyah and isadore singer. *EMS Newsletter*, 53:24–30.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA. Association for Computing Machinery.

Rips, L. (1994). *Psychology of proof: deductive reasoning in human thinking*. MIT Press, Cambridge, Massachusetts.

Robinson, J. A. (2000). Proof = guarantee + explanation. In Hölldobler, S., editor, *Intellectics and Computational Logic*, pages 277–294. Kluwer, Dordrecht.

Schaerf, A. (1993). On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2:265–278.

Schild, K. (1991). A correspondence theory for terminological logics: Preliminary report. *Proceedings of the 12th International Conference on Artificial Intelligence*, pages 466–471.

Schlobach, S. (2004). Explaining subsumption by optimal interpolation. *European Workshop on Logics in Artificial Intelligence*.

Schlobach, S. and Cornet, R. (2003). Non-standard reasoning services for the debugging of description logic terminologies. *IJCAI*.

Schmidt-Schauß, M. and Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26.

Shortliffe, E. H. and Buchanan, B. G. (1975). A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23(3-4):351–379.

Sloutsky, V. M. and Goldvarg, Y. (2004). Mental representation of logical connectives. *The Quarterly Journal of Experimental Psychology A: Human Experimental Psychology*, 57A(4):636–665.

Sowa, J. F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove: Brooks / Cole.

Steiner, M. (1978). Mathematical explanation. In *Philosophical Studies*, volume 34, pages 135–151. Springer, Dordrecht.

Stenning, K. and van Lambalgen, M. (2008). *Human Reasoning and Cognitive Science*. MIT Press, Cambridge, Massachusetts.

Strannegård, C., Engström, F., Nizamani, A. R., and Rips, L. (2013). Reasoning about truth in first-order logic. *Journal of Logic, Language and Information*, 22:115–137.

Strannegård, C., Nizamani, A. R., Engström, F., and Häggström, O. (2014). Symbolic reasoning with bounded cognitive resources. *36th Annual Conference of the Cognitive Science Society*.

Sun, R. (2004). Desiderata for cognitive architectures. *Philosophical Psychology*, 17(3):341–373. https://doi.org/10.1080/0951508042000286721.

Swartout, W., Paris, C., and Moore, J. (1991). Explanations in knowledge systems: Design for explainable expert systems. *IEEE Expert: Intelligent Systems and Their Applications*, 6(4):58–64.

Thompson, S. (2020). Lesson 6: Stratified sampling.

Troitzsch, K. J. (2014). Simulation experiments and significance tests. *Artificial Economics and Self Organization*, 669.

Tubella, A. A. and Straßburger, L. (2019). Introduction to deep inference. `https://hal.inria.fr/hal-02390267`.

Udrescu, S.-M. and Tegmark, M. (2020). Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16).

unknown (1995). *Methods of Algorithm Description: Second Edition*. Board of Studies NSW, North Sydney.

Wagon, S. (1987). Fourteen proofs of a result about tiling a rectangle. *The American Mathematical Monthly*, 94:601–617.

Wang, T. D., Parsia, B., and Hendler, J. (2006). A survey of the web ontology landscape. In *The Semantic Web - ISWC 2006*, volume 101 of *ISWC 2006*.

Warren, P. (2017). *Human Reasoning and Description Logics: Applying Psychological Theory to Understand and Improve the Usability of Description Logics*. PhD thesis, The Open University.

Wason, P. (1968). Reasoning about a rule. *Quarterly Journal of Experimental Psychology*, 20(3):273–281.

White, J. W., Rassweiler, A., Samhouri, J. F., Stier, A. C., and White, C. (2013). Ecologists should not use statistical significance tests to interpret simulation model results. *Oikos*, 123(4):385–388.

Whitehill, J. (2013). Understanding act-r - an outsider's perspective.

Whittaker, M., Crawford, K., Dobbe, R., Fried, G., Kaziunas, E., Mathur, V., West, S. M., Richardson, R., Schultz, J., and Schwartz, O. (2018a). Ai now 2018 report.

Whittaker, M., Crawford, K., Dobbe, R., Fried, G., Kaziunas, E., Mathur, V., West, S. M., Richardson, R., Schultz, J., and Schwartz, O. (2018b). Ai now report 2018. Technical report, AI Now Institute, University of New York.

Yackel, E. (2001). Explanation, justification and argumentation in mathematics classrooms. In *Proceeding of the conference of the International Group of for the Psychology of Mathematics Education*, volume 1, pages 9–24. PME, Utrecht.

Zerilli, J., Knott, A., Maclaurin, J., and Gavaghan, C. (2019). Transparency in algorithmic and human decision-making: Is there a double standard? *Philosophy and Technology*, 32(4):661–683.

Zielinski, T. A., Goodwin, G. P., and Halford, G. S. (2010). Complexity of categorical syllogisms: An integration of two metrics. *EUROPEAN JOURNAL OF COGNITIVE PSYCHOLOGY*, 22(3):391–421.

Sammanfattning

Problemet att optimera automatiserade förklaringar för slutledningar i kunskapsbaser angrips genom att modellera deduktiva resonemangsprocesser med den kognitiva arkitekturen ACT-R. Detta resulterar i modellen SHARP som simulerar algoritmen för att avgöra inkonsistens av en ABox i beskrivningslogiken ALE så som den exekveras av en människa. Närmare bestämt kan SHARP förutsäga slutledningstiden för den här processen, vilket antas spegla den kognitiva belastningen hos en mänsklig agent. Med hjälp av slutledningstiden definieras två komplexitetsmått som bör korrelera med den kognitiva belastningen tack vare hur de är konstruerade.