Human Argument Structure Language

Jelmer van der Linde

August 30, 2018

Master Thesis
Artificial Intelligence
University of Groningen, The Netherlands

First Supervisor:
Prof. Dr. Bart Verheij (Artificial Intelligence, University of Groningen)
Second Supervisor:
Dr. Jennifer Spenader (Artificial Intelligence, University of Groningen)

Abstract

Computers reason, but humans argue. For computers to understand humans and human knowledge, they need to understand human argument. Likewise, they need to be understandable by humans.

There are multiple abstract models of argumentation, and argumentation software exists to help fill in such models and evaluate them. These models are often presented as graphs. Computers are becoming capable of interpreting argumentative text through such models, but the variety of language, subtle ways of expressing things, and the assumption of the presence of common knowledge when formulating text make interpreting a challenging task. Let alone that filling in abstract models (i.e. picking the right place for parts of each argument) can be a challenging practise on its own.

This work unites an abstract argument model with language, creating a tool that translates written argumentation into argument diagrams and vice versa.

The resulting program, HASL is a parser and grammar for argumentative text with accompanying interface. It allows you to enter a text and see how the claims inside the text support and attack each other. It also allowed you to reverse this process, and generate text.

The abstract argumentation model is connected to grammar rules which together create our language that is similar to English, which makes it intuitive to read and write. We construct and implement this grammar in two experiments: HASL/1 defines a grammar that allows for all possible elements in argument structures to occur. HASL/1 also adds support for anaphora and enthymemes (i.e. being able to fill in missing premises). HASL/2 makes use of a more abstract grammar that only uses discourse markers (i.e. keywords) and structure, allowing it to be used when the text cannot be understood completely. This grammar and the argument model are isomorphic, and can be used to formulate argumentative texts. We conclude with an evaluation of these two experiments applied to syllogisms, Toulmin arguments and Tort law.

HASL is a step towards understanding of argumentation by computers, allowing them to explain and discuss in the same language humans do.

Contents

1 Introduction **5**

2 Background **11**
2.1 Argument structure 11
2.2 Argument software 17
2.3 Argument mining 18

3 Argument structures **21**
3.1 Claims and relations 21
3.2 Support 22
3.3 Cooperative and independent support 22
3.4 Warranted support 24
3.5 Attack 26
3.6 Rules & reasons 29

4 HASL/1: A deep understanding **34**
4.1 Algorithm 34
4.2 Pros and cons 35
4.3 Conjunctions 38
4.4 Rules, warrants, generalizations 38
4.5 Undercutter 40
4.6 Collapsed arguments 41
4.7 Anaphora resolution 43
4.8 Enthymeme resolution 45
4.9 Implementation 47
4.10 Evaluation 50
4.11 Discussion 56

5 HASL/2: A flexible understanding **60**
5.1 Segmentation 61
5.2 Data structure 61
5.3 Parser 65
5.4 Grammar 66
5.5 Structured arguments and argument diagrams 69
5.6 Generating textual arguments 71
5.7 Implementation 71
5.8 Evaluation 72
5.9 Discussion 80

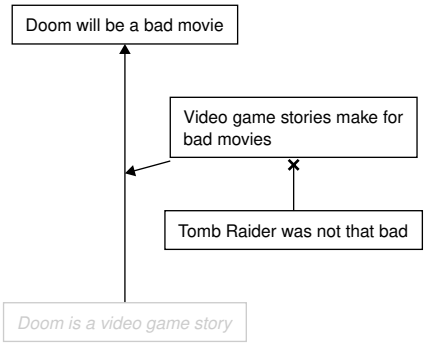
6 General Evaluation **84**
6.1 Pros 84
6.2 Cons 85
6.3 Rules 85

7 Discussion **91**
7.1 An argumentation theory perspective 91
7.2 An argument mining perspective 94
7.3 An argumentation tool perspective 98

8 Conclusion **102**

Appendices **104**

A Appendix **104**
A.1 Part-of-speech tags 104
A.2 HASL/1 Grammar rules 104
A.3 Discourse makers 107
A.4 Evaluation examples 108
A.5 Evaluation results 113



Doom will be a bad movie because video game stories make for bad movies, although Tomb Raider was not that bad.

Figure 1: What we aim for: the text is interpreted by our software and the argument structure is filled in.

1 Introduction

Problem Humans communicate and share their knowledge and understanding of the world by arguing with each other: hypotheses are shared, and defended or attacked by other knowledge. Assumptions are made and acknowledged or retracted based on arguments. Elementary but essential questions like 'why?' and 'how?' are answered with arguments. If computers could understand this process, they could take part, which would be a most natural way of communicating with computers.

Computers essentially reason in formal logic, where data, combined with a rule, leads to a conclusion. If the conclusion is wrong, either the data or the rule was wrong. Or the program contains a bug.

Humans reason differently. They start with the conclusion, an assumption, and then try to find reasons why this assumption should hold or not. These reasons can be generalized rules, assumptions themselves, that work for 90% of the time, but exceptions are allowed. Exceptions are just more specific generalized rules. The reasoning itself often still contains the same elements of formal logic, but the main difference is that any intermediate conclusion, any assumption, is asserted to be true unless questioned.

Current work As such, arguments need a different structure than just formal logic. This structure of arguments is well explored (van Eemeren et al., 2014). Elements essential to argumentation can be found in formal logics, such as Default Logic (Reiter, 1980) or Defeasible Logic (Pollock, 1987).

But most formalisms treat arguments as a graph structure with claims supporting and attacking each other (Thomas, 1981; Mann and Thompson, 1988; Freeman, 1991; Peldszus and Stede, 2013). Often these supporting and attacking relations are seen as just different types of the same relations, but that is not set in stone. E.g. Dung (1995) modelled only attacking relations between arguments, where arguments themselves are assertions, optionally supported by other claims, reasoning, evidence, etc.

Arguing is not just the creation of a graph, there are more 'rules' to it. Arguments to have a certain structure, elements that are expected to be present, either generally (Toulmin, 2003; Verheij, 2005) or in specific types of arguments (Walton et al., 2008). Likewise, there are expectations on how claims can be attacked, and how these attacks differ semantically (Pollock, 1987).

Given a structured argument in which all claims and their relations among each other (i.e. whether they support or attack each other), and a certain order of which claims or relations are stronger, more important, more impressive than others, we can use algorithms to evaluate which claim is accepted as true and which isn't (Reiter, 1980; Dung, 1995; Verheij, 2003b). Once we have computers that are able to follow along with an argument, we could ask it about its interpretation, and whether it thinks the conclusion is true given its knowledge of some of the initial claims in it. The processing power and memory capabilities of computers make them more efficient and less error prone for such reasoning tasks.

Making decisions based on arguments, especially when the decision process and conditions are not explicitly programmed, is one area where humans are superior to computers at this moment. For example, when a human is asked 'Is this a good film?' it can answer with a decision and often an explanation on how it came to that decision. You may not agree with it, but you may agree with parts of it or discover why you do not agree. Computers, for now, are limited to machine learning or statistical correlations for answering these vague questions, as there is not a strict definition or formula to define a good film.

But computers can learn, either from trying using approaches such as Machine Learning or statistical correlation, or by making use of the large amounts of knowledge in the form of argumentative text that humans have written. That way a computer cannot only learn what is a good film, but why it is a good film. And opposed to Machine Learning or statistical methods, you only have to let the computer encounter a single argument that says that humans find films made by Stanley Kubrick are good films because 'he is an acclaimed director', and that films that are adapted from video games are bad films, because 'that just never works'. And unlike with statistical approaches, the computer can now give an argued decision on why the next Tomb Raider film might not be a good film, without having seen tens of thousands of data points to train their classifier. But for this to work, computers need to be able to find, interpret and understand arguments.

Argument mining is the practise of extracting structured arguments from argumentative text, e.g. essays and dialogues, written by humans. Often this is done by finding sentences that look argumentative based on keywords (Marcu, 1997), and then constructing parts of the argument graph that form common argument structures (such as those described by Walton et al. (2008)'s schemes) using a set of rules based on keywords or machine learning (Mochales and Moens, 2011). However, training such algorithms is difficult, and the complexity of natural language makes it challenging to interpret arguments completely. To evade this complexity modern argument mining techniques attack the three stages of argument mining—identification, segmentation and prediction—as classification challenges: a text is argumentative or not, a new claim starts here and ends there, and this claim supports that claim or not. These can be attacked using machine learning algorithms, which use keywords and other linguistic features as input, but do not have a strict understanding

of the text. As a result, these approaches are forced to focus on finding hints of arguments in argumentative text instead of trying to gain a complete understanding, and, most importantly, exploring what such an understanding entails.

Our idea

> Very often natural language understanding entails the translation into another format that can be used by the machine to perform another task (Moens, 2018)

We approach the argumentative text as a 'natural' language that we transform into an argument structure using a parser and grammar. Specifically, we define a grammar for argumentative texts, and only text written in the language as defined by this grammar is to be interpreted. However, we want to keep this language as close to English as possible to make it easy to use and easy to interpret. We try two approaches: HASL/1 in which the grammar identifies every word and symbol in the text, and HASL/2 in which the grammar only describes the discourse markers (keywords such as 'because' and 'but') and symbols such as commas, and accepts the text between these without further interpretation.

Furthermore, we define an argument structure in terms of both grammar rules and a diagramming scheme. This argument structure combines the elements from Peldszus and Stede (2013), such as *independent* and *cooperative* support and attack, with the concepts from Toulmin's argument model (Toulmin, 2003), such as the concept of a *warrant* that warrants a reason supporting a conclusion. It should also be able to express Pollock's *undercutter*.

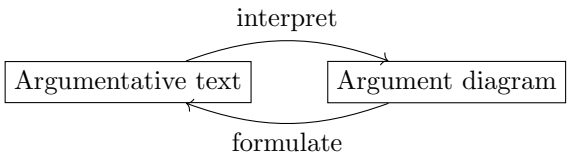


Figure 2: The process of interpreting and formulating arguments.

The grammar and the diagramming method will be two representations of the same underlying understanding of argumentation. The two representations are isomorphic: an argument that can be interpreted by the grammar can be drawn using the diagramming method, and a diagram can be formulated as an argumentative text. We can go from one to the other, and back, as drawn in Figure 2.

For the diagram representation to be as informative as the textual representation, we need to identify and resolve pronominal *anaphora* (pronouns such as 'he' and 'these') because while in an argumentative text it is easy to find out where these refer to, in an argument diagram the order of the sentences that preceded claims with anaphora is lost, and anaphora resolution becomes difficult. Hence we should remove these pronouns from the text used in claims in the argument diagrams.

In this thesis we focus on these two representations to explore how well language can be mapped on a structured representation of arguments, but we also explore usage of this structure: we use the structured form of the argument to identify and fill in *enthymemes*—arguments that have unstated premises or conclusions—and we use the argument diagrams to

provide an alternative method of communicating the argument that is expressed in the argumentative text.

Requirements To summarize, we define an argument structure, grammar, and implementation that allows us to accomplish the following requirements:

1. Express pros: claims supporting a claim.
2. Express cons: claims attacking a claim.
3. Express an argument consisting of multiple supporting and attacking claims.
4. Combine such expressions into a multiple sentences or a single, more complex sentence.
5. Express cooperative and independent support and attack.
6. Express warrants and undercutters.
7. Express Toulmin arguments.
8. Identify warrants.
9. Interpret rule-like claims.
10. Perform anaphora resolution.
11. Perform enthymeme resolution.
12. Formulate argumentative texts.

Structure of this thesis We first discuss our argument structure in section 3 using the diagram representation: each of the occurring patterns in this structure (i.e. particular types of support and attack relations) which we treat as our building blocks for argument structure. These can be combined to form any argument.

Next, for each of these elementary structures, we define one or more grammar rules. These dictate how the textual argument and diagram representation relate to each other.

We then implement these building blocks and their grammars into a larger grammar which allows us to create argumentative texts of multiple sentences. This language, the human argument structure language, or HASL for short, is explored in two variants, each implementation exploring different aspects of the computational understanding of textual arguments.

HASL/1 Our first experiment is HASL/1, which starts off with implementing the grammars for the building blocks: each sentence matches to one of these building blocks, and larger arguments can be constructed by writing multiple sentences, restating the claims that are relevant in each of them.

Our argument structure makes the distinction between claims that serve as direct support or attacking claims, and claims that serve as warrant. In HASL/1 we implement this as a distinction between general claims and specific claims. General claims are the more generalized rule-like claims such as ‘birds can fly’, as opposed to specific claims, which talk about a very specific and particular claim, e.g. ‘Tweety can fly’. We make this distinction by using part-of-speech tags. As such, HASL/1 is a grammar built on top of a part-of-speech tagger, which is a common approach in natural language processing.

Using the structure we extract, we implement a form of enthymeme resolution. Enthymemes, the occurrence of premises that are part of an argument but are not explicitly stated, occur often in natural language (Walton and Reed, 2005; Reed and Rowe, 2004). By filling in these missing premises in our argument graph we can make the right support or attack connections between claims, also when the targeted claim is not explicitly stated. We do this by treating building block around the attack and support relations as small syllogisms, and reconstruct the missing minor or major premise.

We also implement pronominal anaphora resolution. In HASL/1 claims that occur multiple times in argumentative text but that have the same meaning are merged into a single claim in our argument structure. In HASL/1 we determine whether two claims are the same by comparing their text. When this text contains references in the form of pronouns to other parts of the argument, for example if the premise is 'he has wings', we need to check whether 'he' refers to the same object. Otherwise claims that look the same but don't exactly mean the same are merged. Anaphora resolution helps us check this. It also helps with the understanding of arguments, as it makes the claims as drawn in the argument diagram more self-contained. One might say this is a necessary step since the information needed to resolve the anaphora as a human, namely the order of the sentences, is lost in the argument diagram.

HASL/2 Because HASL/1 is built upon the part-of-speech tags of the words in the sentence, its grammar needs to define every combination of these that can occur in the claims. This part of the grammar is not relevant for the argument structure, and inhibits us from formulating syntactically novel claims. In HASL/2 we explore a grammar that does not apply to the internal structure of the premises, but instead only looks at the discourse markers in the argumentative text: punctuation and keywords such as 'because' and 'except'.

This simplifies the grammar, focussing it on the argumentative structure instead of the structure of language in general. The remaining grammar is simple enough to make the implementation isomorph: HASL/2 is able to interpret textual arguments as argument diagrams, but can also formulate a textual argument when given an argument diagram. This makes HASL/2 useful to explore ambiguity in the language.

Furthermore, we found in HASL/1 that both the grammar rules that dictate how general claims are formulated, as well as the argument structure to represent them, to be incomplete: inside general claims there is a little rule structure on its own, dictating conditions and exceptions to their conclusions. For example, articles of law often have definitions, and general rules that describe when these definitions apply. Capturing this structure as well, we can understand argumentation both as it is applied (in an active discussion) and from where it originates (in the application of general rules), allowing for a deeper understanding that can help when evaluating an argument, or when reconstructing the missing claims in enthymemes.

Relevance HASL implements a form of argument mining, but one that is focussed on capturing all possible argumentative structures instead of capturing all possible formulations of such structures. This allows us to explore what a complete understanding of the argumentative text entails, such as the possibility to do enthymeme resolution. It also shows the

possibilities and boundaries of interpreting argumentative text using only the information available in the text itself and given by the structure of our argument model.

The implementations of HASL presented here can be used to gain a better understanding of complicated arguments, where there are many premises that interact with each other. Anaphora and enthymeme resolution help with further understanding of the argument.

The language of HASL itself can be used for a new type of interface to the reasoning process of computer programs, allowing for the explanation of the reasoning behind a conclusion, or even as an interactive interface to engage in a dialogue of arguments with the computer.

Availability Both implementations of HASL can be found and experimented with online:

HASL/1: `http://hasl1.ikhoefgeen.nl/`

HASL/2: `http://hasl2.ikhoefgeen.nl/`

2 Background

For a computational understanding of argumentative texts, two research topics need to be combined: what is the structure of an argument, and how can we extract structural arguments from text. The choices made in answering the first question define the best course of action for the second.

2.1 Argument structure

That there is structure in arguments is undisputed: statements can be supported by arguments. The amount of structure, and the strictness of it, is what the real discussion is about (van Eemeren et al., 2014). One simple but effective abstraction of the structure in argumentation is that of the syllogism, as described by Aristotle.

Syllogism Aristotle's syllogism is the prime example of structure in argumentation. Take the argument 'Socrates is mortal because Socrates is a man and men are mortal.' Or, transcribed in the classical form:

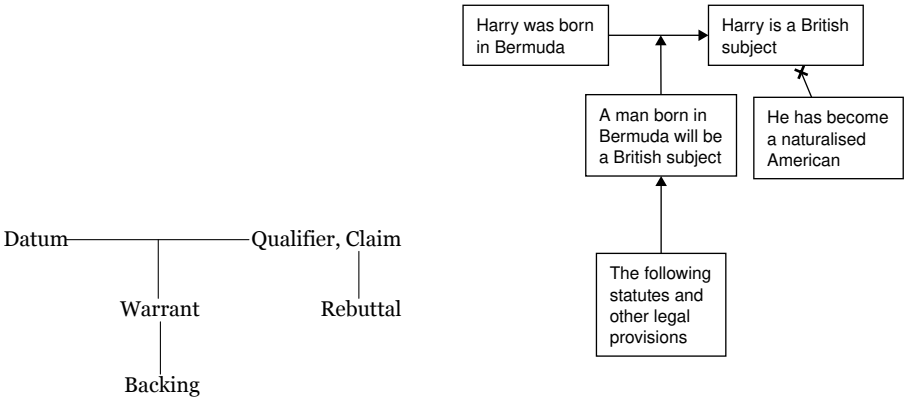
> Aristotle is a man (minor)
> Men all mortal (major)
> —
> Socrates is mortal (conclusion)

A syllogism always consists of a minor premise, which is what we expressed to be a singular claim and a major premise, which is a more general claim, and finally a conclusion. This conclusion describes the general phenomenon described in the major premise applied to the subject of the minor premise.

This structural description of an argument is not very refined, and as such it may be more interesting to search for a structure that can highlight more specific parts in an argument. More structure provides a better framework for working with arguments when, for example, evaluating them or when identifying which premises are explicitly stated and which are left implicit.

Toulmin Toulmin's argument model (Toulmin, 2003), shown in Figure 3, follows the form of the syllogism, but defines in more detail what the role of the premises can be in an argument, and how to deal with attacking claims. It also expands the number of elements, as it sees the need for more information than a syllogism can hold to describe an argument. The model, shown in Figure 3, consists of five elements: the *claim*, *datum* and *warrant*, which are respectively comparable to the conclusion, and the minor and major premise in a syllogism, and the *backing* and *rebuttal*.

The *warrant* serves the role of telling why jumping from the datum to the claim is justified. The warrant is universal, the datum and claim are specific to the scenario described in the argument. This matches our concept of a general rule, or a general claim. The warrant itself can be supported by a *backing*, which is a claim that is not specific to the argument at hand, and thus has barely anything to do with the *conclusion*, but it is often very specific to a field. For example, the warrant in the example shown in Figure 3b, that a man born in Bermuda will be a British subject, is supported by legal reasons, as opposed to for example statistical ('95% of all people born in Bermuda are a British subject') or empirical reasons ('Everyone I know born in Bermuda is a British subject').



(a) Elements of the model. (b) Example of an argument in this model.

Figure 3: Toulmin (2003)'s argument model.

Many general rules have exceptions, but these exceptions are only mentioned when applicable in a discussion. For example, the general statement that 'birds can fly' has many exceptions, such as the bird being a penguin. In a syllogism there is no clear place for such exceptions, and they are often treated either as a conditional statement in the major premise, or as major premise that reaches the opposite conclusion (i.e. 'penguins cannot fly') on its own. In the first approach, the distinction between a condition and an exception is lost, even though there is a clear distinction in argumentation: a condition needs to be met (e.g. there needs to be evidence to support this) before the conclusion of the major premise is assumed, while an exception is often not even mentioned, unless it is applicable. Then, both the exception itself and the supporting evidence that the exception is applicable are presented.

Toulmin allows for exceptions to enter the argument, or any attack to the conclusion, in the form of the *rebuttal*: a claim that can give a reason why the conclusion is not true in this particular scenario.

This makes clear that Toulmin's argument model assumes that the warrant, backing, etc. are not actively reasoned about. It is a model of the correct finalized argument. If the backing itself would be argued by someone, that itself would form a new argument with its own diagram. As such, this model is not mend to show opposing views in the same diagram, or highlight all considerations. It focusses on what elements are essential in a complete and concluded reasoning step inside a possibly larger argument.

Combining premises Toulmin's model does not take the composition of arguments into account; many small arguments often are part of a greater argument, for example a court case or a political debate.

An overview of such a large scale argument has been covered by argument maps, such as the ones developed by Wigmore (1913), shown in Figure 4. The squares and circles signify arguments supporting and attacking certain claims: whether a premise supports of attacks is a property of the premise itself. Hence the structure has to be a tree (or multiple trees) where arguments can only attack or support a single other argument

higher-up.

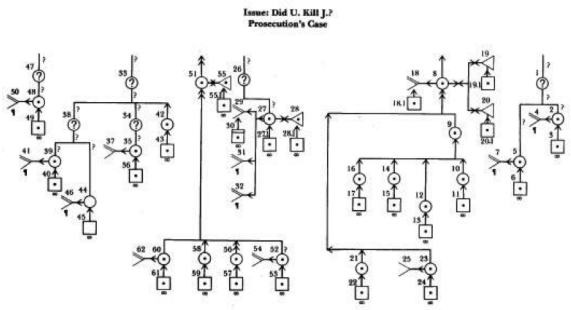


Figure 4: Example of a Wigmore chart, taken from Wigmore (1923).

Grewendorf (1980) moves the signalling of support or attack to the arrow between arguments, which now allows the schema to form a graph instead of a tree, where arguments can attack multiple other arguments while also supporting other arguments.

Thomas (1981) continued this use of argument graphs, but made a clear distinction in how two claims support a conclusion: either cooperatively, or independently. Arguments can support each other in different combinations. For example, some general rule-like claims assume that multiple conditions are met, e.g. 'it snowed outside when it rained and the temperature was below zero'. Only when both conditions can be supported, this general claim would strengthen the conclusion that 'it snowed'. In this case, the two claims *cooperatively support* the conclusion. On the other hand, often there are multiple but separate reasons supporting an argument, for example 'I see snow on the rooftops' and 'the man who just came from outside was covered in snow' are two separate reasons to assume that the conclusion 'it snowed' is true: These two claims *independently support* the conclusion.

These argument charts and graphs all do not have the clear distinct components from Toulmin's model. Freeman (1991) combines the descriptiveness of Toulmin's model with the freedom to compose arguments introduced with the argument graphs, and refines the meaning of the rebuttal, making a distinction between the different types of attacking arguments.

Peldszus and Stede (2013) in turn simplify the schema proposed by Freeman, letting instead the structure of the graph be more defining for the meaning of the components: a conclusion is a box that is supported, an attack is a box that is the source of an attack relation. Claims can support or attack other claims cooperatively or independent.

Their schema does not have a specific place for Toulmin's warrant or Pollock's undercutter. Verheij (2003b)'s DefLog can express these by not only allowing relations to target other claims, but also other relations. In this way a warrant is expressed as a support for the support relation between Toulmin's datum and conclusion (Verheij, 2005). Verheij expresses these relations graphically in his tool ArguMed 3 by drawing an arrow to the line of the supported or attacked relation (Figure 5).

In Verheij's model the relation originating from the warrant can also be supported, and that relation as well, etcetera. He does not define how such relations should be interpreted, only how such arguments can be

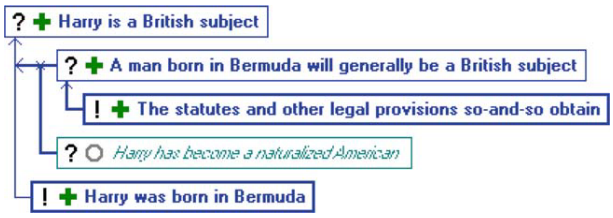


Figure 5: Toulmin's model drawn using DefLog's model. This representation includes evaluation information. Taken from Verheij (2005).

evaluated.

Argument evaluation

The simple and well defined structure of argument graphs forms a good starting point for the computational evaluation of arguments. As opposed to the syllogism, there is no need to comprehend the exact meaning of the text attached to the nodes in the argument graph in order to evaluate whether a conclusion is supported, under attack, or undecided. Of course, for any meaningful evaluation such understanding is necessary, but these tasks can be well separated.

HASL does not do evaluation, but the evaluation of and reasoning with arguments would be a logical extension of HASL: the purpose of understanding arguments in argumentative text is to reason with it in a later phase.

Default Logic A starting point for reasoning with the same semantics as arguments is a logic for default reasoning (Reiter, 1980). Default reasoning allows us to differentiate conditions from exceptions. For example, a condition for an animal to be able to fly might be that it needs to be a bird. An exception to this rule might be that the bird cannot be a penguin. If this would need to be expressed in first-order logic, we would need to define an implication that on the condition side would need to explicitly state all exceptions as predicates that may not be true. If we were to evaluate such an implication, we would need to find a truth value for each of these exceptions. I.e. we would need to test whether the bird is not a penguin, an ostrich, etc.

Default logic allows us to express the implications without mentioning all of the exceptions by making the implications a bit weaker: the implications gain an extra condition, one of consistency. I.e. a bird is able to fly unless that is inconsistent with what is known. This allows the usage of separate implications for exceptions, i.e. penguins cannot fly. If a bird is then a penguin, it cannot fly as the assumption that it can fly because it is a bird is inconsistent with the conclusion of the implication that says that penguins cannot fly.

To interpret this in the context of argumentation, exceptions do not have to be made explicit, unless they are the case, which is often the case in argumentation since there is a urgency to only express what is relevant due to the need for communication to be efficient.

This is the start for a formalism for the non-monotonicity of the logic behind arguments: the introduction of a new claim may not change the evaluation value of any of the previous claims directly, but it can introduce

a new path to reason along which would result in a new valuation for certain claims. This, essentially, is the act of arguing: by introducing new claims, the argument moves along.

But that does not cover the idea that arguing often starts with a conclusion, and then states the arguments in supports of that conclusion, after which the task to attack this reasoning to another party in the argument. The defeasible reasoning described by Pollock (1987) is closer to the argument structures previously described: Instead of allowing each logical statement to have exceptions, i.e. conditions which when met retract the conclusion of the statement, Pollock defines a second type of rules: prima facie reasons, which are true, unless they are not. This matches closely with general rules such as 'birds can fly', which are true unless more information tells us they are not in a particular scenario.

Verheij (2003b)'s DefLog builds on this idea of defeasible reasoning in its evaluation schema. Constants (i.e. claims) are assumed to be justified unless they are defeated, and become justified again when this 'suppression' it of their justification is lifted. This allows DefLog to express sets of assumptions (i.e. arguments with claims and attack and support relations between them) and then evaluate how these arguments can be interpreted, which claims can be assumed to be true at the same time.

A different view on defeat is offered by attack graphs (Dung, 1995). This idea matches the idea of an argument as a graph, except that an attack graph only consists of arguments that attack each other. Inside each argument the supporting arguments can still play a role in determining whether that particular node in the graph, that particular argument, is considered to be true or false. This view can be expanded to include monotonic logic and to model the different forms of attacking an argument described by Pollock and Freeman (Besnard and Hunter, 2014).

To conclude, for a structural representation of argumentation that can be used to describe and evaluate argumentation, we need defeasibile rules, and we need to express reasoning about such rules. Furthermore we need to accept that argumentation is a process, and as such the set of arguments that are accepted at one moment may change at any moment new information becomes available. The necessary elements for such argument structures are defined by Vreeswijk (1997) in order for them to support evaluation.

Argument schemes

The previous sections mostly regarded how ideal argumentation should be structured and interpreted, but aside from corpora of annotated arguments (Lawrence et al., 2012), arguments do not occur often in such ideal forms. Argumentation can be found in argumentative texts, which are much less structured. Hence the need for argument mining. However, for understanding the task of argument mining we first need to understand how arguments occur in text.

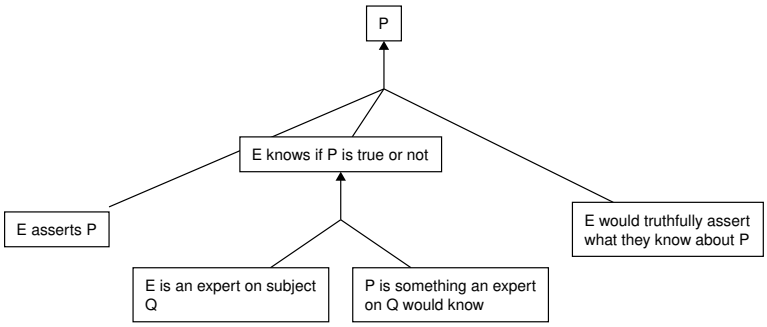
Walton et al. (2008) started with documenting common structures used in argumentation, marking their elements. An example of such a common structure in an argument would be the sentence:

> Attacking the Soviet Union lost the war for Germany claims Dr. Brown, and she is an expert on the Second World war.[1]

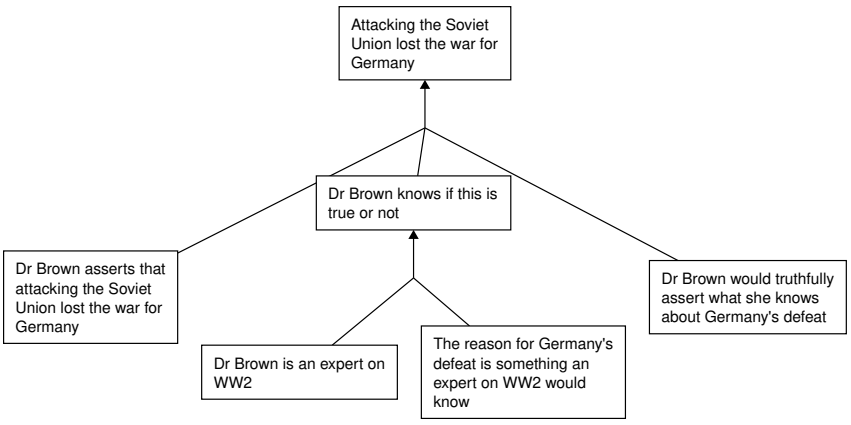
[1] Example obtained from `https://www.rationaleonline.com/map/96cb3e/argument-from-expert-opinion`

In this example, the claim of Dr. Brown about the reason why Germany lost is supported by the claim that she is an expert on the subject. Walton's schema for Argument from Expert Opinion, shown in Figure 6, describes this argument: it describes the premises and the conclusion that need to be present in a general format. All similar sort of arguments follow this schema. As such it could be interpreted as a model like Toulmin's model, except for very specific types of arguments.

Note that, just like with Toulmin's model, not all boxes in Figure 6a can directly be filled in based on the short sentence about Dr. Brown. Some of the slots in the schema are left empty as there is no information available in the sentence that would fill them. In this way, Walton's argument schemes also propose critical questions, such as 'Is Dr. Brown an expert on the subject?'. Walton, and many others, defined such schemes based on the arguments they encountered. Consequently, there is no complete set of all schemes.



(a) Walton's schema



(b) Example

Figure 6: Walton's schema for Argument from Expert Opinion, drawn according to the structure we will describe in section 3.

Enthymemes

Walton schemes also provides a base for detecting enthymemes: premises that are not explicitly mentioned, but assumed to be undisputed. Enthymemes are common (Reed and Rowe, 2004), but challenging for a complete understanding (e.g. when an unstated premise is supported or attacked) and the possibility of evaluation.

Walton and Reed (2005) discuss using forms of deductive or abductive inference and later argument schemes to identify and add the enthymemes to the argument diagram. This first approach may result in multiple competing interpretations, depending on what can be deduced from the present claims in the argument. Here the principle of charity is a common approach: 'When interpreting a text, make the best possible sense of it'. Walton and Reed interpret this as picking the interpretation that makes the argument stronger. However, they say, this might distort the original argument. Similarly, an argument may have a hidden premise that is false, and by making it explicit, an argument would become weaker.

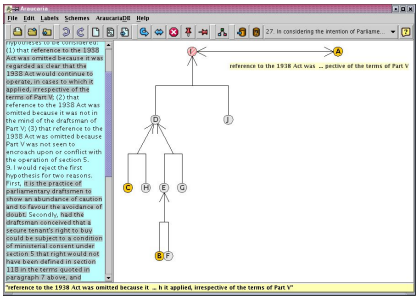
Using argument schemes can be a more general approach to enthymeme resolution. Walton and Reed argue that goal directed practical reasoning is the basis of many enthymemes. For example, arguments based on analogies often imply their conclusion or the claim that both types mentioned in the analogy are comparable. The argument scheme for analogies defines the existence of such claims, and could thus provide a useful basis for enthymeme resolution.

2.2 Argument software

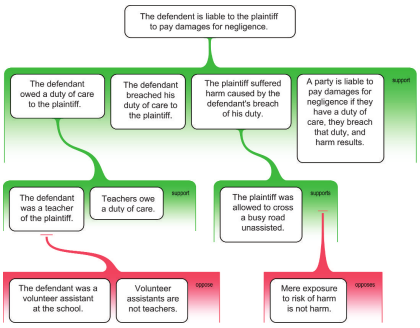
Walton's argument schemes describe particular types of naturally occurring arguments, but there is also a need to annotate the structure in specific textual arguments. Rhetorical Structure Theory (RST) is used to annotate the structure present in text. Originally developed for text generation, RST describes the relations between sentence segments (Mann and Thompson, 1988). In the use case of understanding arguments in text, these relations are often linked to Walton's argument schemes, which then serve as the framework and common ground between different arguments which do follow the same structure.

Reed and Rowe (2004)'s Araucaria is a tool for annotating these argumentative texts with argumentation schemes. While annotating a text, the argument can be visualised as a diagram for a better understanding. Arguments annotated in Araucaria can be exchanged in the Argument Markup Language (AML) or Argument Interchange Format (AIF), and these can then be shared among researchers using AIFdb Lawrence et al. (2012). The latter is a common source of material for machine learning approaches to argument mining, as well as for sharing arguments for educational purposes.

Rationale (van Gelder, 2007) is another argumentation software that focusses on helping people understand and construct correct arguments. It achieves this by creating a visual syntax, a consistent way of hinting the correct interpretation of the argument diagram using visual cues. It can also guide the evaluation of arguments, by going through it claim by claim with the user.



(a) Screenshot of Araucaria as shown in Reed and Rowe (2004).



(b) Screenshot of the visual syntax of Rationale as shown in (van Gelder, 2007).

2.3 Argument mining

RST and argumentation schemes offer ways to add structure to arguments in text. Adding this structure to text, either by hand or automatically, is the practise of discourse analysis. Approaches to discourse analysis usually aim at identifying various different types of discourse relations. However, we are only interested in a subset of such relations, namely those relevant for argumentation structure parsing. Argument Mining is this smaller task performed automatically.

Argument models in argument mining The argument models used in Argument mining are often specifically designed or adjusted for the data they work with. For example, much of the earlier research is done on corpora annotated using RST, and as such the relations are classified as 'attribution', 'background', 'condition', 'elaboration', etc. (Reitter, 2003; Herneault et al., 2010). More recent work uses more abstract models: Mochales and Moens (2011) just identifies premises that support arguments in favour of the conclusion in legal texts, and Lawrence and Reed (2015) combines support and attack relations with Walton's argumentation schemes to describe arguments that originate from AIFdb, a dataset largely constructed using tools specifically created for annotating arguments with Walton's schemes. Even more recent, Stab and Gurevych (2017) only identifies claims, conclusions (which they call major claims) and premises with support and attack relations between them to describe persuasive essays.

Argument mining approaches is generally split up in three subtasks, often implemented and evaluated independently (Lippi and Torroni, 2016).

Identification The parts of text that are argumentative — that may contain arguments – need to be identified: In many texts only certain sentences or paragraphs of the text are argumentative; the rest may be informative, for example providing context or anecdotes. In its essence this is a binary classification task. Some approaches combine this task with the next and directly detect and classify the argument components (Stab and Gurevych, 2017). In this case the classifier is also tasked with detecting the type of component (e.g. conclusion, minor or major premise, etcetera depending on the structural model used.)
The identification task is not always necessary. For example, Saint-Dizier

(2012) focusses on technical manuals, which are processed in total, i.e. their whole structure is interpreted and extracted.

Segmentation The identified parts of argumentative text need to be split into segments – the task of component boundary detection – where each segment may be a component in the final argument representation, i.e. a box in an argument diagram. Assuming that in a text about a court case the paragraphs containing the ruling and the reasoning behind it are identified, this step then splits those parts into components (which we refer to as claims, but often also named argumentative discourse units (Cohen, 1987) or elementary discourse units (Saint-Dizier, 2012)) that can then be identified as the conclusion, premises, etc. This, again, is a classification task. Knott and Dale (1994) argues for a data-driven search for cue phrases (i.e. discourse markers such as the words 'because', 'unless', but also words, word phrases and punctuation that are less explicit, or a complete analysis of the semantic structure) that indicate relations between text spans. Marcu (1997) uses regular expressions and procedures manually derived from a corpus analysis. Herneault et al. (2010); Mochales and Moens (2011); Saint-Dizier (2012); Lawrence and Reed (2015); Stab and Gurevych (2017) use models trained on lexicosyntactic features.

Prediction The relations between the identified components need to be identified. In this task the components are connected to each other to form the structure of the argument. This is the most challenging task as it requires insight in the meaning of a component to determine whether a premise component supports a certain claim component or whether it is related to another claim component. Some approaches also approach this as a binary classification task, testing each of the claim components in a certain defined order (Cohen, 1987). Others have a set of hand-coded grammar-like rules combined with constraint satisfaction to identify the structure between the components using discourse markers, certain word orders and other keywords occurring in the components (Mochales and Moens, 2011; Saint-Dizier, 2012) Another approach is to create a model that can determine the likelihood that two claims are related to each other and construct multiple possible argument structures to finally select the best overall structure, i.e. the one that is the most complete or has to best overall likelihood. This was done by Marcu (1997) by creating a model for relations based on the discourse markers occurring in both components. Later research uses many more types of features describing the words occurring in the component and the position of the component relative to other components (Herneault et al., 2010; Lawrence and Reed, 2015; Stab and Gurevych, 2017).

Often knowledge about the domain is used to help in this task. For example, for some texts, such as procedures or didactic text, it can be assumed that a claim is always followed by one or more premises supporting that claim. In juristic texts certain marker words may signal the relations between components.

Most approaches are based on text which has been annotated using RST and a predefined set of schemes. The rules used in the three subtasks are, either by hand or automatically, derived from these previously annotated texts to create a model which can apply the same annotation to new texts.

Trend towards more general approaches Modern argument mining approaches are leaning less on natural language processing tools like parsing and part-of-speech tagging, and are turning to unsupervised learning of the meaning (i.e. common knowledge) and relations (i.e. entailment and causal relations, and eventually argumentative relations) between segments in texts directly from the words and their embeddings (Moens, 2018).

3 Argument structures

The structure in arguments is made explicit by drawing it using boxes and arrows. These resulting diagrams, which we call argument diagrams, are used by HASL to represent the interpretation of the text. The structure also forms the basis on which the grammars of HASL are written.

This chapter explores that structure, describing how it should be interpreted and giving examples of what argumentative text could best be described by the structures. Our goal is to describe the most basic elements of argument structures which can be combined to form arguments. For each of the basic elements we will identify the general format of argumentative text which describes such an element.

3.1 Claims and relations

In our model, arguments consist of boxes, which represent claims, and arrows, which represent relations between those claims. A constellation of these form one large argument, where smaller parts of this constellation can be seen as local arguments that together contribute to the larger argument. For example, an argument about whether Jack should go to prison would encompass arguments about whether Jack committed a crime, and whether the crime is serious enough to require a prison sentence.

Claims Claims themselves are drawn as a box. Claims are statements such as 'Tweety can fly', 'Socrates is a man', 'Henry was born in Bermuda' but also 'People born in Bermuda are generally British subjects'. Claims are the atoms of our argument. The conclusion is a claim, the statements supporting a conclusion are also claims. In previous research these boxes are drawn with only a reference (i.e. a letter or number) to the actual text (Wigmore, 1913; Thomas, 1981; Peldszus and Stede, 2013). To make the argument diagram self-contained, we draw the actual text of the claim inside the claim box, as is done by Verheij (2005).

Relations Claims are then connected to each other by relations, drawn as arrows (Wigmore, 1913; Thomas, 1981; Peldszus and Stede, 2013; Verheij, 2005). To signal the position of the claim in the argument, these arrows either end with an arrowhead or a cross, indicating respectively a supporting or attacking relation (Thomas, 1981; Peldszus and Stede, 2013; Verheij, 2005). Such a relation marks what is indicated in the argumentative text, i.e. what one of the arguing parties expressed to be a good support for the supported claim. For an arrow from B to A this can be as strong as 'A has to logically follow from B' or as assumptive as 'I've seen many occurrences where A results from B'. Our model does not depict whether the reason is a good reason for the conclusion or is true or false—a property that Verheij (2005) is able to express—only what was expressed in the argument itself.

Layout Like Verheij (2005), but opposed to Peldszus and Stede (2013), we prefer to draw the main claim of the argument at the top. We draw the graph, if possible, as a tree below it. If a relation connects to another relation instead of a claim, we prefer to draw it on the left or ride side of it. The algorithm for these layouts can be found in the implementation of HASL, and is also used for all the diagrams shown in this article.

(a) Conclusion supported by a premise.

(b) In this example 'Socrates is human' serves the role of a premise and of a conclusion.

Figure 8: Examples of claims supporting claims. These visualisations combine the usage of a graph (Grewendorf, 1980) with the symbols for attack and support introduced by Verheij (2003b) and displaying the text of the argument itself, as done by Verheij (2005). This combination of elements and the layout of the graph is our own.

3.2 Support

Claims supporting other claims, i.e. that describe the reason why the concluding claim must be the case, are drawn as a simple arrow between the two boxes that represent these claims. Figure 8 shows examples these.

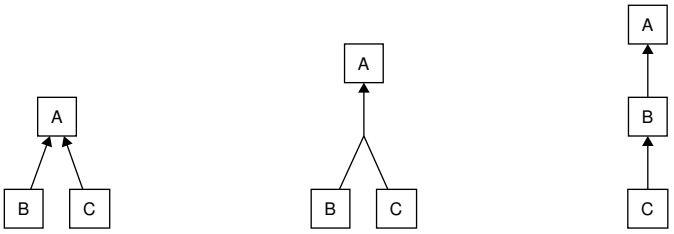
Often we call the box from which the arrow originates the *supporting claim*, and the box to which the arrow points the *supported claim* or *conclusion*. In the context of a syllogism the supporting claim can also be interpreted as the *premise*. In Toulmin (2003)'s model, it often matches the idea of a *datum*.

The relation between the claims, i.e. the support relation specifically in these examples, determines the role of the claims. In Figure 8b the claim that Socrates is human is supporting the claim that Socrates is mortal, and it is supported by the claim that his parents are human. As such the role of 'Socrates is human' is both that of premise and conclusion.

3.3 Cooperative and independent support

A conclusion can also be supported by multiple supporting claims, but how these supporting claims support the conclusion can be different. Either *independently*, by two separate claims or *cooperatively*, by two claims that together form a single supporting argument (Thomas, 1981; Peldszus and Stede, 2013).

Independent support When there are multiple but separate reasons to come to the same conclusion, the reasons are drawn with distinct support relations, as in Figure 9a. The idea is that if one of these reasons no longer holds, for example because it is successfully attacked, that this should have no impact on the other reasons. They are truly independent from each other, and each one of them is sufficient to on its own be a reason to conclude the conclusion.



A because *B* and because *C* *A* because *B* and *C* *A* because *B* because *C*

(a) Independent support (b) Cooperative support (c) Chained support

Figure 9: Combining support relations through independent, cooperative and chained support, as described by Grewendorf (1980). The way these relations are depicted is similar to Peldszus and Stede (2013)'s *linked*, *multiple* and *serial support* relations. The layout is our own.

Cooperative support Some combinations of supporting claims can only support the conclusion when they occur and remain unchallenged together. For example, Dutch tort law dictates that someone has to repair damages only when four conditions are met. These four conditions then all need to be present, and all remain unchallenged, to support the conclusion that the tortfeasor has to repair the damages. The multiple relations coming together to form a single line in Figure 9b symbolizes this.

Examples In Dutch tort law (Betlem, 1993) we can find examples of both independent and cooperative support.

(1) John has a duty to repair the damages because someone suffered damages, those damages were caused by an act of John, the act was unlawful and the act can be imputed to John.

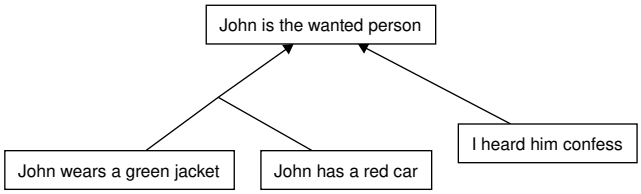
(2) The act can be imputed to John because this article of law describes it as such and because common opinion describes it as such.

In example 1 John has a duty to repair the damages because he meets all the conditions for this. If any of these conditions are not met, i.e. one of these claims is successfully attacked, the claim that John has a duty to repair damages is no longer supported.

In the second sentence, example 2, there are two reasons given for why the act can be imputed to John. Either one of them is sufficient, but in this example, for the sake of explaining, there are two given. Other examples could be eyewitness statements, where multiple witnesses claim to have seen something independently from each other. If the statement of one of these witnesses is challenged, the other statements remain unchallenged.

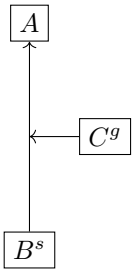
The combined argument from these two examples are drawn in Figure 10, in which the claims in example 1 are drawn as a connected single arrow, and those from example 2 as two separate arrows.

Chained support A third way of expressing multiple claims supporting a conclusion is through chained support, shown in Figure 9c. This is



John is the wanted person because he wears a green jacket and he has a red car and because I heard him confess.

Figure 10: Examples of cooperative and independent support in a single argument.



A because B^s and C^g.

Figure 11: Warranted support

essentially the combination of multiple arguments, as the conclusion of the second is the premise of the first.

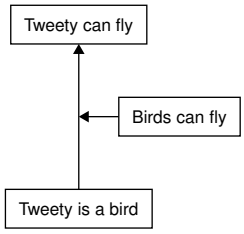
(3) Tweety can fly because Tweety has wings. Tweety has wings because Tweety is a bird.

(4) Tweety can fly because Tweety has wings because Tweety is a bird.

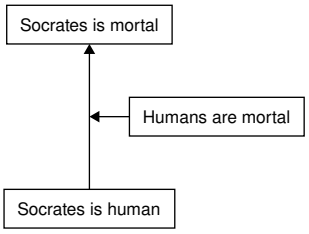
The resulting semantics when evaluating the argument do not different from those of cooperative support: if any of the supporting claims is attacked, the topmost conclusion itself is no longer supported. The intention of chained support is different: Given examples 3 or 4 the question 'Why can Tweety fly?' can now be answered with 'Tweety has wings', and 'Why does Tweety have wings?' with 'Tweety is a bird'. We can continue this unendingly.

3.4 Warranted support

Besides claims supporting conclusions, these relations between claims and conclusions can themselves also be supported. This should be interpreted as arguing on the argumentation itself: Supporting a support relation explains why that support relation should be taken serious, i.e. why the claim that Tweety is a bird is a valid reason to assume that Tweety can fly. A claim used in such a fashion is equal to the *warrant* in Toulmin's model, or to the *major premise* in a syllogism.



Tweety can fly because Tweety is a bird and birds can fly.



Socrates is mortal because he is human and humans are mortal.

Figure 12: Two examples of warranted support relations.

Warrants The warrant is a claim that supports the process of concluding the conclusion based on the presented premises. Both A^s and B^s in Figure 11 can be unchallenged claims, but the process of B^s supporting A^s itself may also need to be supported. This is the role of the warrant, the relation between C^g and the arrow pointing from B^s to A^s.

(5) Tweety can fly because Tweety is a bird and birds can fly.

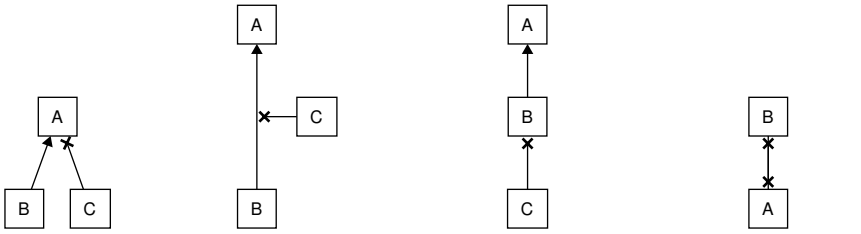
(6) Socrates is mortal because he is human and humans are mortal.

Generally, C^g in Figure 11 is a general rule, e.g. 'birds can fly' or 'humans are mortal' in examples 5 and 6. B^s and A^s are both specific to the situation and talk about individual instances, e.g. 'Tweety is a bird' and 'Tweety can fly'. We use the superscript g and s to indicate this. C^g is the general case that, when applied to this specific situation discussed in the relation it supports, further explains why concluding that 'Tweety can fly' makes sense given that 'Tweety is a bird'. This becomes more apparent in Figure 12, which shows the argument diagrams for examples 5 and 6.

General and specific claims The warrant should be interpreted as being a generalized rule, one without all the exceptions and conditions that are assumed to be irrelevant for the discussion at hand. As such, we will often refer to these claims as general claims.

We make a distinction between general claims and specific claims, for the reason that only general claims can serve as a warrant in an argument. A general claim is a statement that is more generally applicable than just the current argument, and which often is assumed to be undisputed among all participants of an argument. For example, 'birds can fly' is a general statement, as opposed to 'Tweety can fly'. However, 'Tweety can fly' today could also be a general statement in the context of the more specific argument whether Tweety can fly today. As a rule of thumb, a warrant is a claim which does not change meaning in an argument when you put a qualifying keyword such as 'generally' in front of it.

Note that the general claim is in itself still just a claim. It can be supported and attacked. For example, 'birds can fly' itself can be supported by statistical claims that show that indeed most birds can fly. And such a support relation in turn can again be warranted by a claim stating that statistics is generally a sufficient method to show that such a rule of thumb may be stated.



(a) C rebuts A (b) C undercuts A (c) C undermines A (d) A and B attack each other

Figure 13: Four different strategies of attack.

Syllogism This construction of general claim, specific claim and conclusion takes the form of a syllogism. The general claim is the major premise, and the specific claims supporting the conclusion form the minor premise. The major premise warrants concluding the conclusion given the minor premise.

Conditional claim Often the warrant is also a conditional claim, and as such we can interpret the warrant as formal logic: 'Birds can fly' can be interpreted as 'something can fly *if* something is a bird'. More generally: the warrant can be interpreted as a rule: 'X is a bird' is the condition for 'X can fly', where X serves as a variable for some sort of reference to the individual to which the warrant is applicable. Each of the conditions should match up with a claim in the minor premise. If there are multiple conditions, they should be combined using cooperative support.

3.5 Attack

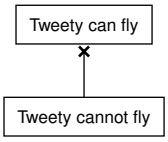
There are multiple types of attacks possible in an argument. We focus on claims themselves and on the context of a support relation, as these two elements essentially make up the building blocks of arguments.

As Figure 13 shows there are three strategies of attacking parts of the argument. One can attack any of the claims, or the relation between these two. In the context of this local argument, these three strategies are named rebutting, undercutting and undermining.

Counter-claims The example in Figure 14a is an example of such an attack which just claims the opposite. These two claims are clearly incompatible with each other, and as such exclude each other. If one of them is indeed true, the other has to be false (baring the context of time in this example). As such, we can draw the attack relation between these two claims as a mutual attack relation.

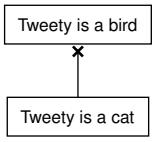
The second example, Figure 14b illustrates that the attack relation between two claims is not always as easy to perceive as in the first example. Here, only after knowing the meaning of being a bird or squirrel it is that one can conclude that being a bird excludes being a squirrel.

Counter-examples In the case of general claims, which are often generalized rules, these attacking claims can take the form of a counter-



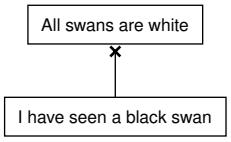
Tweety can fly but Tweety can not fly.

(a) Explicit attack



Tweety is a bird but Tweety is a cat.

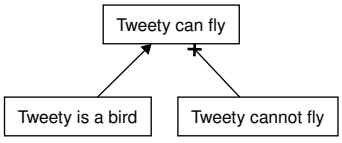
(b) Attack depending on common knowledge



All swans are black but I have seen a white swan.

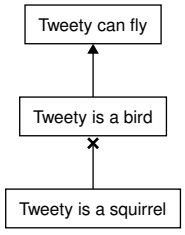
(c) Attack through counter-example

Figure 14: Three direct attacks on a claim. For each of these more common knowledge is needed to interpret correctly. I.e. 'cannot' versus 'can' can be interpreted using the explicit negation, but 'bird' versus 'cat' requires the common knowledge that birds cannot be cats. The last example requires us to understand both 'black' excludes 'white' and the meaning of 'all'.



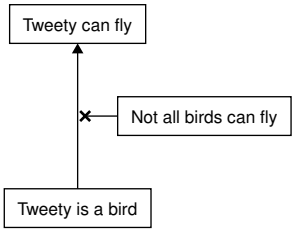
Tweety can fly because he is a bird but he can not fly.

(a) Rebutter



Tweety is a bird but Tweety is a squirrel.

(b) Underminer



Tweety can fly because he is a bird but not all birds can fly.

(c) Undercutter

Figure 15: Examples of different strategies of attack.

example, which is what happens in the example in Figure 14c. This can only be concluded after multiple inference steps, namely that 'All swans' implies that any swan seen by me must also be black, and that being black excludes being white. Again, these two claims mutually exclude each other.

Rebuttal Rebuttal in an argument is finding a way of directly attacking its conclusion, for example by claiming the opposite of the conclusion, or by showing that it is impossible or incorrect. The claim that Tweety cannot fly in the example in Figure 15a is a rebutter.

Undercutter The supporting relation between claim and conclusion may be warranted, but it might not be (or no longer be) relevant when evidence comes to the table that the situation discussed is an exception to the rule. This exception, named an *undercutting defeater* by Pollock

(1987), then takes the role of attacking this support relation. An example of such an undercutter indicating an exception:

In Figure 13b the general form of an undercutter is drawn. Note that the undercutter does not attack the warranting claim itself. In the example in Figure 15c it does not counter the claim that Tweety is a bird, nor that Tweety can fly. However, in the light of the undercutting claim that not all birds can fly, the claim that Tweety is a bird no longer lets us say anything about whether Tweety can fly.

Underminer Lastly, the supporting claim itself can be attacked, shown in the example in Figure 15b. Here, the support for the claim that Tweety can fly is questioned, but the conclusion itself is not. If the argument continues it could be that a new claim supporting the conclusion is presented, e.g. that Tweety is a flying squirrel, or the attack is continued in the form of the warrant that squirrels cannot fly and hence Tweety cannot fly, finally forming a rebuttal.

Cooperative and independent attacks Attacks can attack the same claim in the same fashion as supporting claims can, both cooperatively and independently.

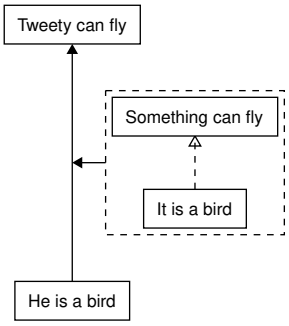
Attack in warranted support If we also warrant the support relation by introducing a general claim that supports the support relation itself, we have two more opportunities for attacks: the newly added arrow, and the general claim. The first we treat the same as undercutting, and the second as rebuttal of the general claim.

Attack versus mutual attack In our argument diagrams we do not draw them as mutual attack for the reason that in the argument itself, a claim is often presented as an attack on another claim. As such, the attack clearly has a direction after the intention of the participant of the argument that brought the claim to the argument. We would like to capture that direction and intention. In argumentative text this intention is presented in the word order: in 'A but B' it is B that attacks A.

This should not pose any problems, as it is up to the producer of the arguments to correctly state the intentions and ignoring those by filling in a mutual attack would do more harm. It is also not important for the correctness of the argument as our goal is to capture the intention of the argument as it is presented, not as it should be by all theories of correct argumentation.

For the evaluation of arguments it does not matter whether the attack relation is bidirectional or not: either the attacked claim holds, and as such the attacking claim cannot hold, or the attacking claim holds, and thus the attacked claim does not.

The rebuttal of an argument is essentially equal to attacking it by stating (and preferably supporting) the opposite of its conclusion. In that light, a rebuttal could also be drawn as a claim stating the opposite of the current conclusion, an attack relation between these two, and a support relation originating from your rebutting claim to the newly added counter-conclusion.



Tweety can fly because he is a bird and something can fly if it is a bird.

Figure 16: The warrant is analysed further, and its conditions are made explicit. The structure of the argument itself is expected to match the structure of the rule-like warrant.

3.6 Rules & reasons

Argumentation and rule-like text of general claims are closely related: the structure where claims are evidence for a conclusion works both for arguments as well as for rules. We do treat them differently though, as arguments describe the currently discussed situation whereas rules describe the general case. This is why in Figure 16 we draw them using the same arrows, but limit the condition and exception relations to the box of the general claim in the argument.

In this subsection we further explore this rule structure, which follows the same lines as the general structure for argumentation, including cooperative and independent conditions, and exceptions that are modelled as undercutters.

Defeasible rules The warrant in argumentation is often not complete: it only describes the general case, for example 'birds can fly', but does not explicitly mention all the exact conditions or all possible exceptions, i.e. birds that have wings that can carry their weight can fly, or that birds except penguins and ostriches and a few other exceptions can fly. In this manner, only the general rule, i.e. 'generally birds can fly', is expressed, and if there is an exception to this rule applicable in the current argument, it is expressed as a separate statement. Still, even a defeasible rule has conditions that have to be met for it to be in any way applicable.

Explicit conditions The following example demonstrates the transition between argumentative text as we have discussed it until now, and rules which are formulated as applied logical expressions, to show these conditions:

(7) Tweety can fly because Tweety is a bird and birds can fly.

(8) Tweety can fly because Tweety is a bird and something can fly if it is a bird.

(9) Tweety can fly because Tweety is a bird and Tweety can fly if Tweety is a bird.

The first example, 7, is the familiar example where 'birds can fly' is the general claim and 'Tweety is a bird' is the specific claim, and the general claim warrants the specific claim's support for the conclusion that Tweety can fly.
Example 8 and 9 are similar sentences, except that the general claim is being reformulated in a more rule-like formulate and applied to Tweety: 'Tweety can fly if Tweety is a bird'.

The general claim 'birds can fly' in example 7 hides the conditions that are explicit in example 9, i.e. something needs to be a bird if it is capable of flight according to this rule, it cannot be applied to dogs for example.

In our argument structure we expressed exceptions to general claims as rebutters if they directly attack the conclusion (i.e. claim the opposite, like 'Tweety cannot fly') or as undercutters if they cause the support to become irrelevant (i.e. the undercutting claim causes the support relation to longer be evidence for the conclusion, for example by expressing that the specific case being discussed is an exception to the general rule that is used as warrant).

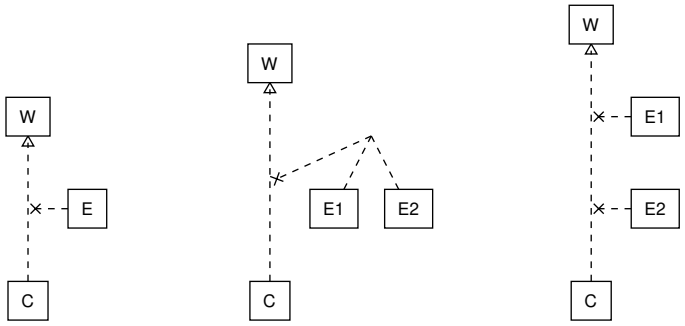
The conditions that are hidden in example 7 and made explicit in example 9 are not yet drawn in our argument diagram as these themselves are not part of the argumentation process: a general claim is called upon during argumentation, but not actively formed in that same argument, hence the individual conditions are not explicitly discussed.

Reasons for drawing conditions Still, we would like to be able to express these conditions in a more systematic way as they are relevant to arguing: when following along with an argument, one needs to check whether these conditions are met by the instance mentioned in the specific claims, and whether the conclusion of the general claim is then also the conclusion of the argument after being applied to the specific subject discussed. If not, then either the specific claims are incomplete or incorrect, or the general claim is not the right warrant for this argument.

For HASL there are two reasons to identify the conditions in rules explicitly: first it can help with enthymeme resolution because when given the conditions of the general claim and the conclusion, the specific claims can be filled in. Second, general claims might become rather complex. Take for example the argument shown in Figure 10: the warrant for concluding whether someone needs to repair the damages in a tort case states all kinds of conditions that the person and the scenario need to meet before being applicable. Additionally, tort law also explicitly states a number of exceptions, circumstances in which all the conditions might be met but in which the conclusion is not applicable. In such arguments, these exceptions might be explicitly stated when stating the complete warrant by for example citing an article from tort law, but the exception might not be used in the argument itself. We want to have a diagramming structure to express this construction of conditions and exceptions which occurs inside the general claim.

Structure inside general claims

Figure 17 shows the building blocks which can occur inside a general claim. These are modelled after the support and attack relations for arguments themselves.



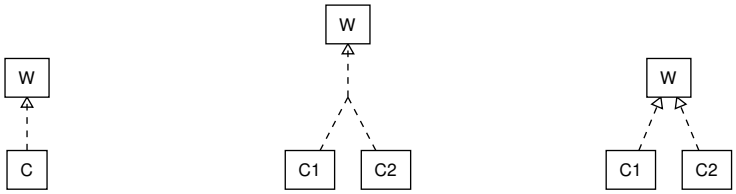
(a) General claim with one exception.

(b) General claim with one exception that has multiple exceptions. multiple conditions.

(c) General claim with

Figure 17: The structural elements inside a general claim.

conditions conditions as shown in Figure 18a, 18b and 18c are conditions that need to be met for the conclusion of the rule-like claim to be applicable. They can be cooperative or independent to each other, similar to how claims can cooperatively and independently support other claims (Figure 9).



(a) General claim with one criterion

(b) General claim with two related criteria

(c) General claim with two unrelated criteria

Figure 18: Multiple ways of connecting criteria to a general claim.

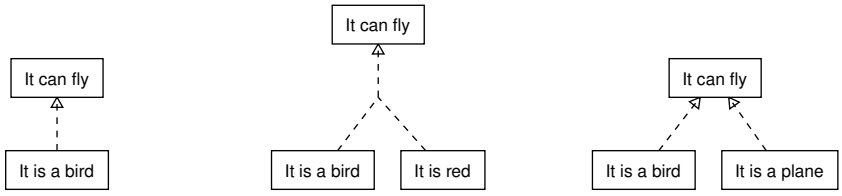
How these conditions are described in argumentative text varies greatly. As shown in the introduction of this section it could be as explicit as 'if something is a bird it can fly' or as casual as 'birds can fly'.

The process of extracting the conditions from formulations such as 'birds can fly' is not described in detail here as it in itself is a task as discourse analysis itself. Therefore the following grammars are but a subset of all possible ways to express conditions inside general claims.

Figure 19 shows examples of arguments that can be described using structures Figure 18.

Exceptions Each condition can have its own exception, similar to how undercutters can form exceptions to general claims (Figure 17a):

(10) If it is a bird it can fly except if it is a penguin.



If it is a bird, it can fly.

It can fly if it is a bird.

Birds can fly.

(a) Different ways of writing the same rule.

Red birds can fly.

(b) Conjunction of conditions.

Birds and planes can fly.

(c) Alternative conditions.

Figure 19: Multiple ways of connecting conditions to a general claim.

(11) Birds can fly except when they are penguins.

(12) Birds except penguins can fly.

There again can be linked (Figure 17b) and unrelated (Figure 17c) conditions for these exceptions. Examples of general claims with one condition which have one exception that has multiple conditions, drawn in Figure 20c:

(13) If it is a bird it can fly except if it has wings and they are clipped.

(14) Birds can fly except when they have clipped wings.

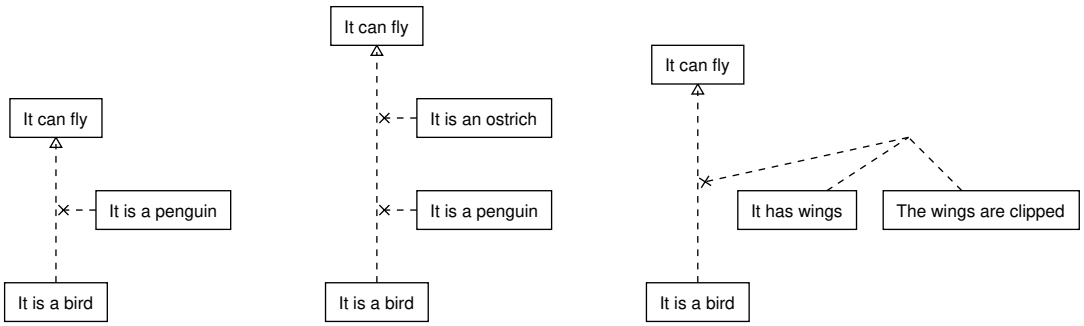
Examples of general claims with one condition that have multiple exceptions, drawn in Figure 20b:

(15) If it is a bird it can fly except if it is a penguin or if it is an ostrich.

(16) Birds can fly except when they are penguins or they are ostriches.

(17) Birds except penguins and ostriches can fly.

Combined example Using the abstract argumentation structure described in Figure 9 and Figure 17 we can combine arguments like shown in Figure 21, in which specific claims are supported by a general claim, and the general claim is interpreted as a rule-like claim with conditions and exceptions. The specific claim is indeed supported by a claim that matches the condition of the general claim, and none of the exceptions are present in the argument.



Birds except penguins can fly.

(a) Example of a general claim with one exception.

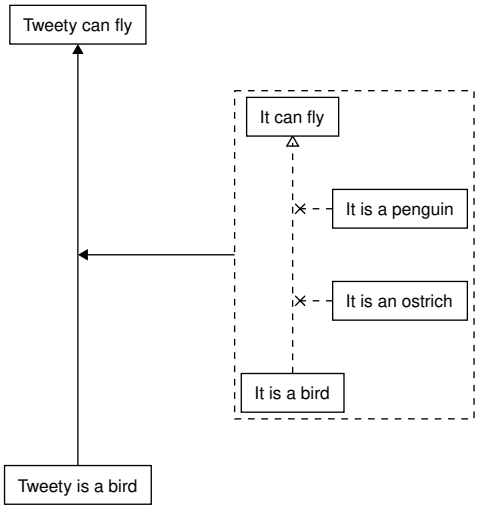
Birds except penguins and ostriches can fly.

(b) Example of a general claim with multiple exceptions.

Birds can fly except when they have clipped wings.

(c) Example of a general claim with one exception that has multiple conditions.

Figure 20: Examples of how we can have multiple exceptions or a single exception with multiple conditions.



Tweety can fly because Tweety is a bird and a bird can fly except when it is a penguin or when it is an ostrich.

Figure 21: Example of a general claim with one exception that has multiple conditions.

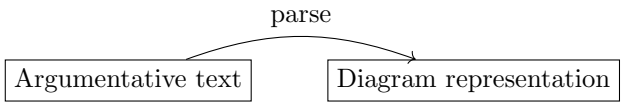


Figure 22: Transformation step in HASL/1.

4 HASL/1: A deep understanding

HASL is an implementation of the language which focusses on supporting all elements of the argument structure. It allows us to parse argumentative text into an argument diagram (Figure 22).

Combining sentences (or structures) in to larger arguments is done by repeating the premise, similar to repeating a variable in first-order logic, or where applicable by collapsing multiple arguments into a single sentence. It allows us to express both arguments for and against a claim, either in simple elementary sentences or in longer expressive sentences. These arguments can be supported by claims that are general rules. It also contains anaphora and enthymeme resolution to further support the human aspect of HASL: these two steps are tasks humans do without thinking, and for humans avoiding them would come at an additional cost. Therefore our language implements them as well. Anaphora resolution makes the claims in the argument diagram self-contained while allowing us to write argumentative text that does not contain needless repetition. Enthymeme resolution does this for whole claims, and as such has two functions: it helps us write argumentative text more concise, but it also shows the claims that we unintentionally left out.

4.1 Algorithm

This first iteration of HASL tests the idea of using a grammar that describes the structure of an argument in a sentence to interpret the textual argument. This idea is similar to parsing the syntactic structure in a sentence – a common practise in natural language processing – except the resulting structure does not describe which verb is the main verb in a sentence, it instead describes which claim is supported by which other claim.

Part-of-speech tagger Sentences are tokenized and to each of the tokens a part-of-speech tag (POS-tag) is assigned using spaCy (Honnibal and Montani, 2018). POS-tags indicate whether a word is a plural or singular, noun, verb or name, etc.

The POS-tagger in spaCy does this using a trained statistical model based on features of the words, such as its prefix, suffix, whether it is written all lower case or upper case, its shape, etc.

We refer to these tags in our grammar as this allows us to generalize positions in the grammar based on the form of these words instead of the actual words themselves. For example, we need to make the distinction between singular and plural and identify pronouns and proper names for anaphora resolution, and determine the type of words for it to match the correct position in a sentence.

Parser The tokens are then parsed into argument structures using an Earley parser and the grammar described in this section. The Earley

parser is capable of parsing both left-recursive and right-recursive rules, which makes writing the grammar easier. It is a chart parser, and performs an exhaustive search of all possible parses of the sentence given our grammar rules.

To explain the algorithm briefly: the chart consists of a set of states for each input position (i.e. token, word). The set of states for the next position are generated by three actions: *Predict* adds any rules that could help to advance or complete one of the rules in the current set of states. *Scan* advances any of the rules that accepts the current token at this position. *Complete* finds all rules that have no more tokens or completed rules to accept in this state, and combines these with the rules that are waiting for this rule. These are then advanced.

Our implementation is optimized by allowing it to ignore the structure below `verb-phrase` and its variants: they are treated equally, and if the matched texts of multiple parses is equal, only one remains on the chart. This reduces the number of ambiguous parses that the parser can yield, but only for cases in which it does not impact our argument diagrams.

Argument structure The partial and final argument structures are represented as a graph with nodes and edges, or claims and relations. Claims are essentially strings of text. Relations are like edges between these, with a few differences: First, a relation can have multiple claims as source to model Cooperative support relations between claims (Figure 9b). Second, a relation can target another relation. This is needed to encode the warranted support relation (Figure 11).

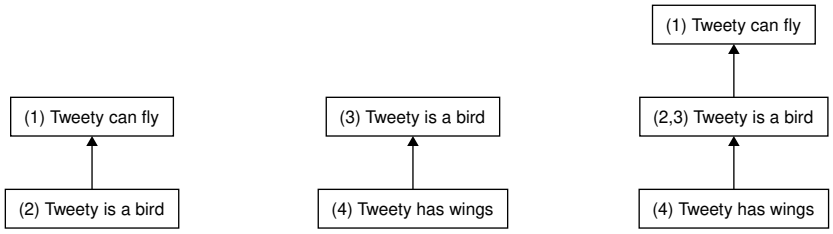
Merging arguments At the end of every grammar rule the matched structures are merged. For rules that combine a few words into for example a `noun` this merely involves combining the spans of text into a single span that encompasses both (i.e. 'the' + 'man' becomes 'the man') but when combining argument sentences the argument structures need to be merged. During this merge claims that were repeated in the text are deduplicated: they are merged into a single node in the argument graph using the following steps, listed below and depicted in Figure 23.

1. All claims from both arguments to be merged are added to the list as a pair consisting of the claims id and the claim itself.
2. Each claim that occurs multiple times in the list is then replaced with the first occurrence.
3. The list is used to replace all occurrences of the replaced claims with their replacement in the relations.
4. The resulting set of unique claims and up-to-date relations is used to create a new argument, the result which is yielded after finishing the grammar rule.

4.2 Pros and cons

Support relations between claims (Figure 24a) are the basis of argumentation: These indicate a reason to come to a certain conclusion.

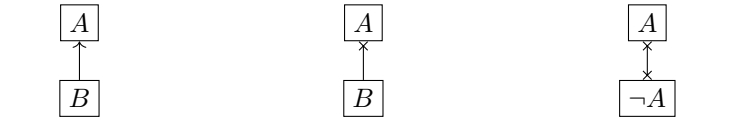
Attack relations (Figure 24b) indicate that a claim is, or has at some moment in the argumentative text been, disputed. The attack can be mutual: two claims exclude the possibility of each other, for example Socrates being mortal and Socrates being immortal. We draw these as two relations collapsed into one bidirectional one, shown in Figure 24c.



(a) First sentence (b) Second sentence (c) Merged argument

Tweety can fly because Tweety is a bird. Tweety is a bird because Tweety has wings.

Figure 23: Argumentative sentences are merged by the grammar to yield a single argument diagram. Here the repeated claim 'Tweety is a bird' is merged into a single claim, and the relations from and towards claim 3 is now updated to refer to claim 2.



A because B. A but B. A. $\neg A$.

(a) Support (b) Attack (c) Mutual attack

Figure 24: Pros and Cons

Grammar We start with describing the grammar for the basic support and attack relations as used in the arguments in Figures 8 and 14a in section 3.

⟨*argument*⟩ ::= ⟨*claim*⟩ '`because`' ⟨*specific-claim*⟩

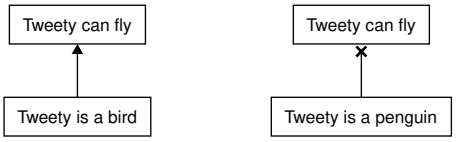
⟨*argument*⟩ ::= ⟨*claim*⟩ '`but`' ⟨*specific-claim*⟩

After parsing the rule in a text the argument structure is formed. If we use a and b to denote the text that matched `<claim>` and `<specific-claim>` we can describe the argument yielded by matching the first rule as follows: It consists of two claims, conclusion a and premise b, and a relation b supports a.

Attacks (Figure 25b) are parsed in a similar way, but with a different type of relation. Mutual attacks cannot yet be expressed as currently our grammar can only handle one single statement, while for mutual attacks we need two attacking statements, one for each direction.

Using these rules, we can already create simple arguments such as the ones shown in Figure 25.

General and specific claims We make the distinction between specific and general claims: specific claims are statements about someone or something particular, e.g. Tweety, Socrates, and 'the object' in our examples. General claims are claims of a more general or generative nature:



Tweety can fly because Tweety is a bird

(a) Example of Support

Tweety can fly but Tweety is a penguin

(b) Example of Attack

Figure 25: Examples of pros and cons in argumentative text and their interpretation in HASL/1.

they are more rule-like and express expectations, for example that birds can fly or that men are mortal.

General and specific claims can be related to the major and minor premises in syllogisms. However, unlike syllogisms the conclusion is also either a general or specific claim. We have to classify it as either as the conclusion of an argument can have the role of a minor or major premise in the next argument, e.g. in the case of chained support (Figure 9c).

Specific and general claims are recognized in HASL using the grammar rules described in subsection A.2. We use `<x>` with lower case x to refer to other grammar rules, `<X>` with upper case X to denote a token that matches one of the part-of-speech tags from Table 3 as identified by SPACY, and `'x'` to indicate that the token is just the string 'x'. Segments between square brackets are optional.

Both general and specific claims consist of a subject and a verb phrase. More complex subjects or verb phrases can often be parsed in multiple ways, but only yield one interpretation. Ambiguity in this structure is ignored as the internal structure of the subject or verb phrase is not stored in the argument structure. The parser treats general or specific claims with different a linguistic structure as a single claim. I.e. a sentence like 'He saw the man with the looking glass' only yields a single interpretation since the structural difference, i.e. whether he or the man had a looking glass, are ignored.

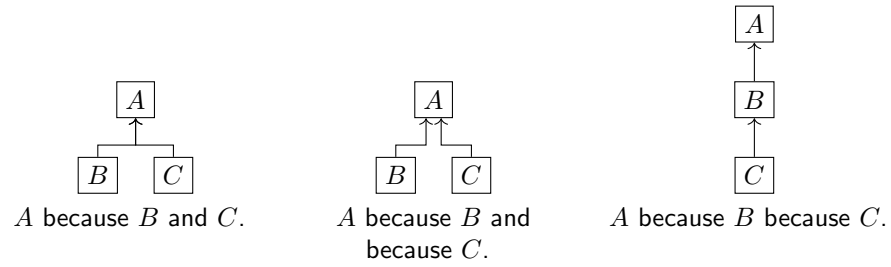
Sentences Finally, an argumentative text in HASL's language consists of one or more sentences which combine the argument structures into a single argument diagram. Effectively a sentence is an argument ending with a period, and `sentences` is the top-most element of the parse tree.

⟨*sentences*⟩ ::= ⟨*sentences*⟩ ⟨*sentence*⟩
| ⟨*sentence*⟩

⟨*sentence*⟩ ::= ⟨*argument*⟩ '.'

We can now also use multiple sentences to express mutual attacks (Figure 24c) by writing two separate attack statements:

(18) Socrates is mortal but Socrates is immortal. Socrates is immortal but Socrates is mortal.



(a) Cooperative support (b) Independent support (c) Chained support

Figure 26: Conjunctions

4.3 Conjunctions

Arguments can be supported or attacked by multiple claims in three different ways, as shown in Figure 26. First, multiple premises can cooperatively support the conclusion, implying that attacking one of the premises is sufficient to undermine the conclusion. Second, multiple claims can independently support the conclusion, where attacking one premise does not affect the support of the others on the conclusion. Lastly, the supporting premise itself can be supported, and in such a case that claim is both a conclusion and a premise at the same time.

The grammar for supporting and attacking claims is extended to allow a conclusion to be supported or attacked by multiple claims as shown in the diagrams in Figures 26c.

⟨*argument*⟩ ::= ⟨*claim*⟩ '`because`' ⟨*specific-claims*⟩

⟨*argument*⟩ ::= ⟨*claim*⟩ '`but`' ⟨*specific-claims*⟩

⟨*specific-claims*⟩ ::= ⟨*specific-claim-list*⟩ '`and`' ⟨*specific-claim*⟩
| ⟨*specific-claim*⟩

⟨*specific-claim-list*⟩ ::= ⟨*specific-claim-list*⟩ '`,`' ⟨*specific-claim*⟩
| ⟨*specific-claim*⟩

We can now express the argument structures with Cooperative support. Independent support (Figure 26b) and chained support (Figure 26c) can also already be expressed using the grammar thus far by repeating the claim that is the conclusion in multiple sentences in the argumentative text. This is demonstrated in Figure 27

4.4 Rules, warrants, generalizations

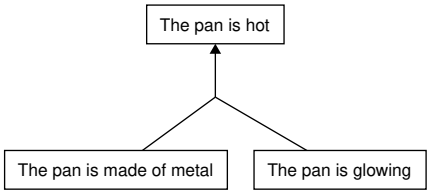
Warranted support and attack combines the usage of the general and specific claims. For clarity we add a little superscript s for claims that are specific claims, and g to claims that are general claims.

⟨*argument*⟩ ::= ⟨*claim*⟩ '`because`' ⟨*specific-claim-list*⟩ '`and`' ⟨*general-claim*⟩

⟨*argument*⟩ ::= ⟨*claim*⟩ '`because`' ⟨*general-claim*⟩ '`and`' ⟨*specific-claims*⟩

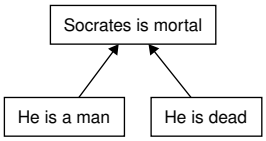
⟨*argument*⟩ ::= ⟨*claim*⟩ '`but`' ⟨*specific-claim-list*⟩ '`and`' ⟨*general-claim*⟩

⟨*argument*⟩ ::= ⟨*claim*⟩ '`but`' ⟨*general-claim*⟩ '`and`' ⟨*specific-claims*⟩



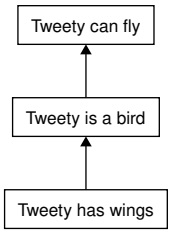
The pan is hot because the pan is made of metal and the pan is glowing.

(a) Cooperative support example



Socrates is mortal because he is a man. Socrates is mortal because he is dead.

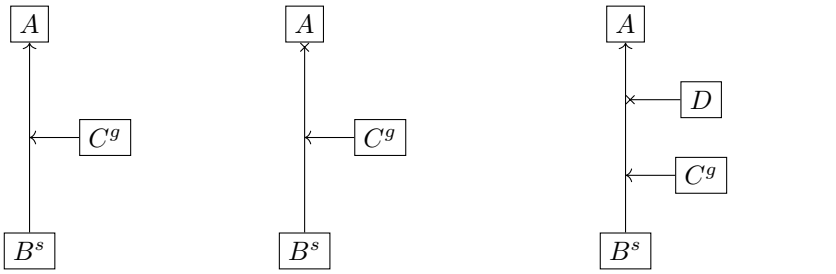
(b) Independent support example



Tweety can fly because Tweety is a bird. Tweety is a bird because Tweety has wings.

(c) Chained support example

Figure 27: Examples of conjunctions



A because B^s and C^g. A but B^s and C^g. A because B^s and C^g but D.

(a) Warranted support (b) Warranted attack (c) Undercutter

Figure 28: Support using rules and attack thereof.

Tweety can fly because Tweety is a bird and birds can fly.

(a) Warranted support example

Tweety can fly but Tweety is a penguin and penguins can not fly.

(b) Warranted attack example

Figure 29: Examples of warranted support and attack.

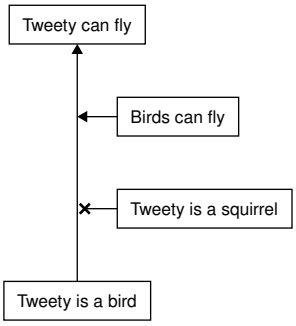
These rules add two relations to the argument: one linked relation from the specific claims that are the minor premises supporting or attacking the conclusion, and a relation from the general claim that serves as the major premise, which is drawn as a general claim supporting the previous created relation between the specific claims and conclusion. Demonstrations of the rules are shown in Figure 29.

Note that `specific-claim-list` and `specific-claims` can also consist of only a single claim. There is no `general-claim-list` or a way to add multiple general claims to a support relation as we only expect at most one general claim to serve as a major premise to be relevant per relation.

4.5 Undercutter

We need an extra rule to express the undercutter (Figure 28c), as we currently have no way to express an attack on a relation. We allow both general and specific claims as the attacking claim since there is no need to differentiate between the two here. The rule is demonstrated in Figure 30.

⟨*argument*⟩ ::= ⟨*claim*⟩ '`because`' ⟨*specific-claim-list*⟩ '`and`' ⟨*general-claim*⟩ '`but`' ⟨*claim*⟩



Tweety can fly because Tweety is a bird and birds can fly but Tweety is a squirrel.

Figure 30: Undercutter example

⟨*argument*⟩ ::= ⟨*claim*⟩ '`because`' ⟨*general-claim*⟩ '`and`' ⟨*specific-claims*⟩ '`but`' ⟨*claim*⟩

4.6 Collapsed arguments

The grammar thus far is unambiguous: There is only one way for each of the elements of the argument structure to be expressed. For simple examples it is very effective:

(19) Socrates is mortal because he is a man and men are mortal.

(20) The object is red because the object appears red but it is illuminated by a red light.

Other structures, such as the chaining or arguments (Figure 26c) and two arguments attacking each other (Figure 26a) are verbose to express:

(21) Tweety can fly because he is a bird. He is a bird because he has wings.

(22) Tweety is a bird but Tweety is not a bird. Tweety is not a bird but Tweety is a bird.

All argument constructions can be formulated with the grammar that has been presented, but some constructions, such as chained support (Figure 26c), are less intuitive to formulate because every sentence can consist of only one argumentative statement.

We extend the grammar to also allow argumentative statements to occur inside sentences. I.e. a conclusion from one statement can be the premise in the next.

In most cases we only allow these argumentative statements, which we call general and specific arguments, at the end of the sentence to make the behaviour of how sentences are interpreted more predictable. Since we need to make a distinction between general and specific claims for the warranted support and attack relations, we also need to inherit this distinction in the argumentative statements. Hence we call them `general-argument`

and `specific-argument` to reflect that their conclusion is respectively a general or specific claim, which then can serve as either the major or minor premise in another argument.

The grammar rules that express support and attack relations function the same way as the grammar rules that express a single claim. The conclusion of the argument will function as a premise for the next. At the sentence level it does not matter whether the conclusion an argument is a general or specific claim. We remove this distinction at that level using the following rule:

⟨*argument*⟩ ::= ⟨*general-argument*⟩ | ⟨*specific-argument*⟩

Support To limit the ambiguity of how supporting clauses can be interpreted, we only allow the repetition of supported conclusions functioning as a supporting premise to happen at the end of the sentence. Therefore `specific-arguments` only allows the last claim to be of the argued variety.

⟨*specific-argument*⟩ ::= ⟨*specific-claim*⟩ '`because`' ⟨*specific-arguments*⟩
  | ⟨*specific-claim*⟩

⟨*specific-arguments*⟩ ::= ⟨*specific-claim-list*⟩ '`and`' ⟨*specific-argument*⟩
  | ⟨*specific-argument*⟩

⟨*specific-claim-list*⟩ ::= ⟨*specific-claim-list*⟩ '`,`' ⟨*specific-claim*⟩
  | ⟨*specific-claim*⟩

The grammar for general arguments is adapted likewise.

⟨*general-argument*⟩ ::= ⟨*general-claim*⟩ '`because`' ⟨*specific-arguments*⟩
  | ⟨*general-claim*⟩

Using these rules, we can express chained support more friendly, as demonstrated in Figure 31c.

Multiple support In our grammar thus far we can construct independent support structures (Figure 26b) by writing multiple sentences and repeating the conclusion. We add the following syntax to allow us to leave out the repeated conclusion:

⟨*specific-argument*⟩ ::= ⟨*specific-claim*⟩ ⟨*reasons*⟩

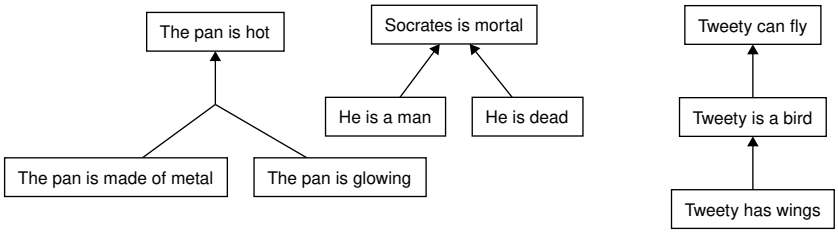
⟨*reasons*⟩ ::= ⟨*reason-list*⟩ '`and`' ⟨*reason*⟩

⟨*reason-list*⟩ ::= ⟨*reason-list*⟩ '`,`' ⟨*reason*⟩
  | ⟨*reason*⟩

⟨*reason*⟩ ::= '`because`' ⟨*specific-claims*⟩

Note that the difference between sentences with cooperative support (Figure 31a) and independent support (Figure 31b) is the repetition of 'because' in each of the separate reasons.

Attack The way attacks are expressed is a little different. To allow for mixing supporting and attacking relations to be expressed in the same sentence, we add variants of `<specific-argument>` and `<general-argument>` that are attacked. However, in text, 'but' often only occurs after 'because' and we have little expectation of what premise in the exposition so far it



The pan is hot because the pan is made of metal and the pan is glowing.

(a) Collapsed cooperative support example

Socrates is mortal because he is a man and because he is dead.

(b) Collapsed independent support example

Tweety can fly because Tweety is a bird because Tweety has wings.

(c) Collapsed chained support example

Figure 31: Conjunctions using collapsed sentences.

will be attacking. To stay true to this behaviour, we allow the attacked claim to be argued itself, unlike in the grammar for support relations. We also only allow the attack argument to occur as an argument on its own (i.e. a single sentence) to limit the number of possible ambiguous but unlikely interpretations. Forming chained attacks (like Figure 26c) however is allowed.

⟨*specific-argument*⟩ ::= ⟨*specific-argument*⟩ '`but`' ⟨*argument*⟩

⟨*general-argument*⟩ ::= ⟨*general-argument*⟩ '`but`' ⟨*argument*⟩

Mutual attack In our grammar we already parse claims with negations in them (the optional `RB` element in many of the `general-premise` and `specific-premise` rules), but we do not use this information yet to automatically create a mutual attack relation (Figure 13d) when these two attack each other.

To allow for this we alter the processing of the `specific-claim` rules that match negated claims to add the flag *negated=True* to the claim. We then also alter the `argument` and `specific-argument` rules that match an attack relation between two specific claims to create two relations (effectively a bidirectional relation) if the claims share the same subject, verb and object but one of them has the negated flag set and the other does not. This allows us to only write one sentence instead of both in example 18.

4.7 Anaphora resolution

Currently the pronouns that occur in the input are copied to the boxes in the argument diagram, but this makes the argument more difficult to understand as the order of the sentence, which we normally use to resolve these anaphora, is no longer present. Furthermore, the grammar to express multiple support or attack relations depend on recognizing the repetition of the conclusion, but if the first occurrence of the conclusion contains a name, and the second a pronoun which is an anaphora for that name, the exact text of both claims is different.

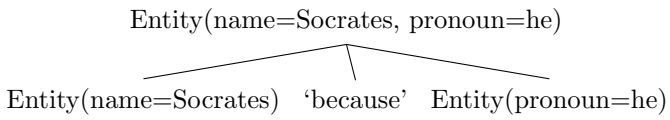


Figure 32: Anaphora resolution inside the parse tree. The two entities at the ends of the tree are merged into a single entity in all nodes higher up in the tree.

We are already able to identify the names and pronouns in our grammar using the part-of-speech tagged words. We alter the rules for claims to also incorporate information (number, and indirectly gender) on the entity that occurs in the subject of a claim for the purpose of connecting these entities later on.

The data structure that describes these entities allows them to be described using a name, noun and pronoun. If an entity has been mentioned using both a name and a pronoun in the sentence, the final argument structure will have a data structure that has both these slots filled in.

The name is assumed to be consistent per entity: someone is only referred to by a single name throughout an argument. In some cases people may be referred to initially using their full name, and later on by only their first or last name. Such cases are currently not covered by our system.

The noun is also assumed to be consistent per entity: again, we do not expect for example 'the king' to be referenced using a different noun. This does occur in for example news articles do this to make a text less repetitive, but that case is not covered by our system.

Lastly, the pronoun is assumed to be consistent as it depends on the gender of the entity, and we assume the gender does not change in a single argument. The pronoun effectively encodes the gender of the entity, without the need for a translation table from known pronouns to genders.

The anaphora resolution process itself is added to the merging of arguments, which occurs at almost every level in the parse tree since all rules built upon `general-argument` and `specific-argument` yield these partial argument structures in which multiple claims can occur. The resulting algorithm is similar to Hobbs' algorithm (Hobbs, 1978). Effectively, the entities occurring higher up in the tree are the merged variants of entities lower in the tree, as shown in Figure 32.

To determine whether an entity refers to another entity and therefore should be merged, the matching rules in Table 1 are used. On the left we have the initial entity. On the top we have the possibly referring entity (which occurred later in the text).

For example, while walking up the parse tree, if we first come across 'he', and then 'the king', and either we have seen previous occurrences of 'the king' being referred to using 'he', or it hasn't been referred to at all, we assume that 'the king' is connected to 'he', and we update our entity to remember that 'the king' is a 'he'. If we would first have stumbled upon 'the queen', to which has already been referred with 'she', then we would test whether these pronouns, 'he' and 'she', are identical, which they are not. Hence we would not link the two. Would we not yet have seen a reference to 'the queen', we would have connected it with 'he' as no previous pronoun or gender knowledge is available, and 'the queen' would be marked as a 'he' for the rest of the argument. We do not have

	name	noun	pronoun
pronoun			=
noun		=	yes
name	=		yes

Table 1: Pronoun resolution rule table. To determine whether two entities should be merged we look at the referring entity and move to the first column for which it has a value. The first test in that column that can be applied will determine the result. If none can be applied, the entities will not be merged. In case of a pronoun the entities are only merged if the first-occurring entity has a matching pronoun or no pronoun at all.

a database with nouns and genders, and try to identify the gender based on the proximity of the pronouns.

In the argument diagram the entities are printed using their most descriptive designation, so preferably their name, then their noun, and otherwise their pronoun. In case of anonymous entities created by the enthymeme resolution process the entities are displayed with the text 'something'.

The process to merge two arguments (i.e. combine their lists of claims and relations) is updated to the following process:

1. All claims from both arguments to be merged are added to the list as a pair consisting of the premise id and the premise itself.
2. All entities occurring in the claims are extracted from this list and added to a similar list, with pairs of their id and the entity itself.
3. This list with entities is sorted on the position where the entity first occurred in the input text.
4. For each entity in the list it is compared to all the following entities to check whether that entity could be referring to it.
5. If so, the entities are merged, the new merged entity is added to the table, and all occurrences of the two merged entities is replaced with the new entity.
6. All claims are updated to refer to the new entities using the entity replacement list that has just been constructed. These updated claims are also added to the list with id-claim pairs.
7. Each claim that occurs multiple times in the list is then replaced with the first occurrence of it in the list. The test to determine whether two claims are the same depend on the updated entities.
8. The list is used to replace all occurrences of the replaced claims with their replacement in the relations.
9. The resulting set of unique claims and up-to-date relations is used to create a new argument, the result which is yielded after finishing the grammar rule.

4.8 Enthymeme resolution

Parts of the argument are unstated: premises assumed to be known and accepted by all parties are not communicated (or they might not be communicated because they are weak and the speaker does not want to wake

sleeping dogs). However, even though these arguments are not part of the argumentative text, they can be supported and attacked by other claims that are explicit. Hence, they have to be drawn in the diagram in order for them to be able to supported or attacked. Therefore we need to reconstruct and draw them.

Enthymemes occur often, and visualising them also helps with the understanding of arguments. By doing enthymeme resolution in the same state as the duplication of claims we cannot only fill in the claims that are not stated explicitly, we can also connect the claims that are stated elsewhere in the argument in the right place: the initially missing claim is first reconstructed through the enthymeme resolution process in the place we expect it. If later, when partial arguments higher up in the parse tree are merged and the reconstructed claim is found to be explicitly stated elsewhere in the argument, the reconstructed and original claim are merged.

Rule-like interpretations of claims Enthymeme resolution constructs one of the three parts (minor premise, major premise, conclusion) of a syllogism in case the other two are present (Figure 33). In our grammar, this entails that it can fill in the missing claim and relation in the warranted support or attack construction. To do this we need to interpret the general claims in a different, more rule-like manner. To see this change, we have the following two examples of general claims:

(23) birds can fly

(24) men are mortal

(25) something can fly if it is a bird

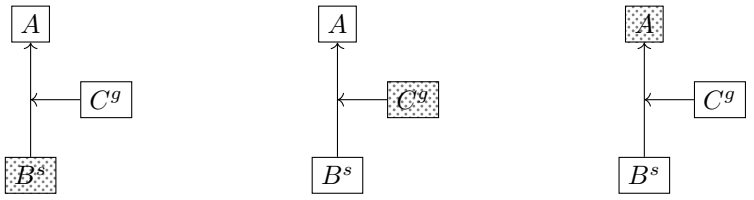
(26) something is mortal if it is a man

We alter the transformation rules of the `general-claim` rules in the grammar to store them as rules instead of spans of text. For example, examples 23 and 24 are how any general claim was observed and stored. For the general claim rules we change this to storing them as rules, very similar to examples 25 and 26, replacing the subject with an unspecified entity that occurs as the subject of conclusion of the new rule-like general claim and all its conditions. This entity functions much like a variable in first-order logic. The original subject is now reformed as a condition for the claim:

(27) something is a man $\rightarrow$ something is mortal

The data structure for general claims extends the original claim data structure by adding a slot for *conditions*. These conditions themselves are specific claims, but they are not explicitly drawn in the argument diagram. Instead, they are added to the text in the box of the general claim. We allow multiple conditions to, for example, encode the need for cooperative support by multiple claims that serve together as the minor premise.

We add grammar rules for arguments where one of the three elements of the syllogism is missing. In each of these the other two are used to construct the missing third element of the syllogism, and each of the rules will yield a (partial) argument that has all three elements present.



(a) Missing minor premise (b) Missing major premise (c) Missing conclusion

Figure 33: Enthymemes originating from syllogisms.

Missing minor premises The claims for the missing minor premises are reconstructed from the subject of the conclusion and the conditions of the general claim that serves as the major premise. For each of the conditions a specific claim is created with the subject of the specific claim that serves as the conclusion and these newly constructed specific claims are linked to the claim of the conclusion using a support relation. This relation is then supported by a support relation originating from the general claim that serves as the major premise.

⟨*general-argument*⟩ ::= ⟨*specific-claim*⟩ '`because`' ⟨*general-argument*⟩

Missing major premise The major premise can be reconstructed from the claims that serve as the minor premise by creating a condition for each of the specific claims and copying the verb and object from the conclusion to the new general claim that will serve as the major premise.

⟨*specific-argument*⟩ ::= ⟨*specific-claim*⟩ '`because`' ⟨*specific-arguments*⟩

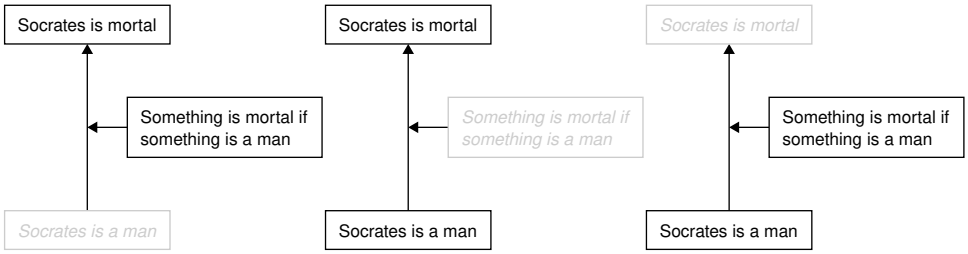
Missing conclusion The conclusion is reconstructed by taking the subject from one of the specific claims that serve as the minor premise and the verb and object from the general claim that is the major premise. We use the subject from the first specific claim as we assume they either all have the same subject or the one most relevant is mentioned first. The constructed general claim is then also supported by the specific claims, and this relation in turn is supported by a relation originating from the general claim.

⟨*specific-argument*⟩ ::= ⟨*specific-claim-list*⟩ '`and`' ⟨*general-argument*⟩

The enthymeme resolution using these three rules is demonstrated on a syllogism in Figure 34.

4.9 Implementation

HASL/1 is implemented in Python as a web service that returns JSON. Its interface consists of a web page on which the argument diagrams are drawn via JavaScript, and which allows you to enter argumentative texts, explore the interpretations through argument diagrams, and view additional information such as how the argument diagram is constructed through the parse trees, and which rules were applied to construct it.



Socrates is mortal because men are mortal.

(a) Missing minor premise

Socrates is mortal because he is a man.

(b) Missing major premise

Socrates is a man and men are mortal.

(c) Missing conclusion

Figure 34: Enthymemes originating from syllogisms.

User interface The user interface, shown in Figure 35, consists of a text field where sentences can be entered to be parsed. After one or multiple sentences are parsed, an interpretation window is shown with argument diagrams for every possible interpretation of the entered text. In this screenshot, only a single interpretation is available.

The argument diagrams (Figure 35a) are interactive. You can drag the claims around to disentangle them if necessary.

For each of the interpretations there is an option to show or hide the parse tree (Figure 35b), which shows the names of the grammar rules. When a node in this tree is hovered over with the mouse, it will show the file and line of the grammar rule it originates from.

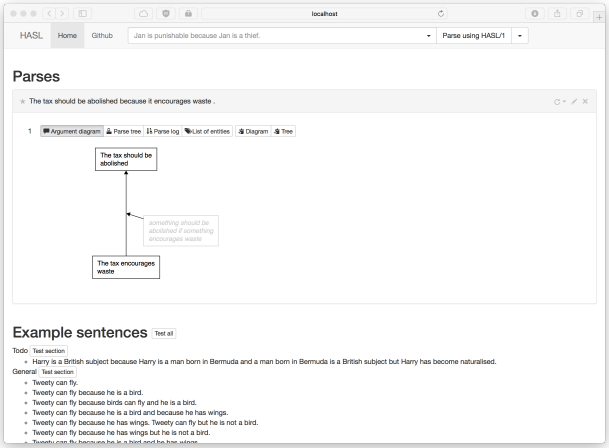
Furthermore, for each interpretation a parse log and a list of entities found and unified by the anaphora resolution can be displayed. There are also buttons to copy the argument diagram or the parse tree as LaTeX code to the clipboard.

The interpretation window itself shows the input text, and for each word the part-of-speech tag when hovered. There are buttons to retry the parsing of the input text using the same or a different grammar, to save the input text, or to close the window.

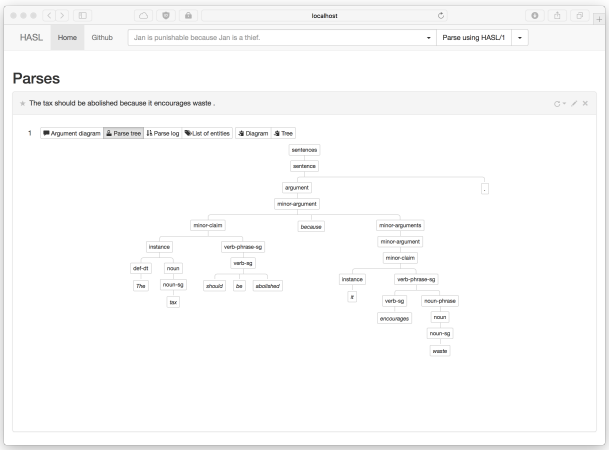
Each new parsing of an input text creates a new window. This makes it easy to compare the interpretations of different sentences side by side.

Switching grammars The interface has options to switch between multiple grammars through the drop-down menu on the right side of the parse or retry buttons, which allows you to switch between HASL/0 and HASL/1. HASL/0 is an earlier version of HASL/1 without the enthymeme resolution and ability to nest argumentative sentences. This option allows for easy experimentation with different versions of the grammar.

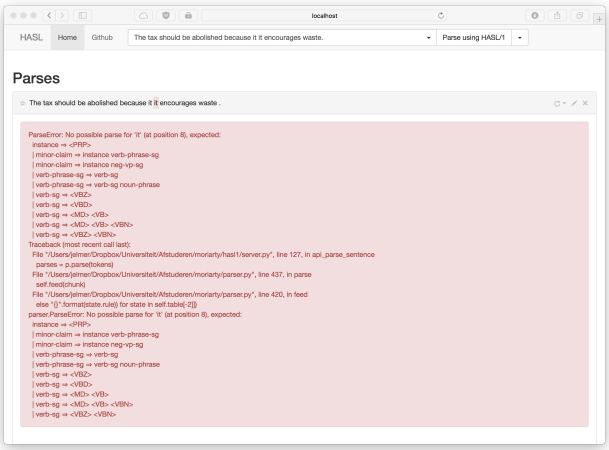
Incorrect input Argumentative texts that cannot be parsed by any of the grammar rules yield an error message telling us which word was the culprit. It also shows the rules it was trying to complete and a more detailed version of the error, for when the cause was not a parse problem but a logic exception in one of the transform functions of the grammar rules.



(a) The argument diagram for the entered text.

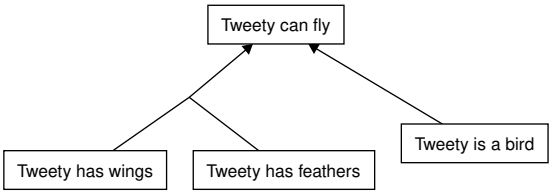


(b) The parse tree for the entered text.



(c) The error message when a text cannot be parsed.

Figure 35: Screenshots of the interface of HASL/1.



Tweety can fly because he has wings and he has feathers and because he is a bird.

Figure 36: Interpretation of an argument containing both cooperative and independent supports in HASL/1.

Evaluation sentences At the bottom of the web page (visible in Figure 35a) there is a long list examples, including the examples used for evaluation in this thesis. These are provided as shortcuts: clicking on them will cause them to be entered and parsed automatically. They can also be tested per section, which will add a small widget with the number of interpretations that were found to each of the sentences, or an exclamation mark if the sentence caused an error.

4.10 Evaluation

We experiment with the implemented grammar by trying out different kinds of arguments. We focus on the features we expect HASL/1 to cover, summarized in section 1. We do this by formulating test sentences (listed in subsection A.4) and interpreting these with the implementation of HASL/1. The complete results of this evaluation are listed in Table 5 in subsection A.4. Here we show our observations by drawing the diagrams for a selection of these sentences.

Pros

The examples used throughout this section all already demonstrate that simple claims supporting claims, such as example 42, are interpreted as expected.

42 Socrates is mortal because Socrates is a man.

44 Birds can fly because birds have wings.

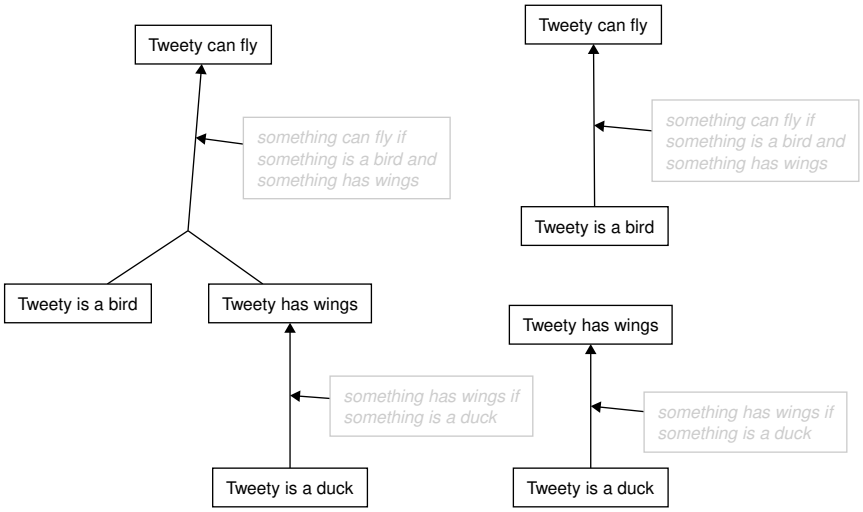
Because we differentiate between specific and general claims based on whether the subject of the claim refers to something specific or something more general, example 44 cannot be parsed by HASL/1. A simple example that combines these into a single sentence is shown in Figure 36.

Combinations of arguments, such as examples 53, 54 and 55 supporting each other are also interpreted correctly.

53 Tweety can fly because Tweety is a bird and because Tweety has wings.

54 Tweety can fly because Tweety is a bird and Tweety has wings.

55 Tweety can fly because Tweety is a bird because Tweety has wings.



(a) Interpretation of HASL/1.

(b) Intuitive interpretation.

Tweety can fly because Tweety is a bird and Tweety has wings because Tweety is a duck.

Figure 37: HASL/1's interpretation and the more intuitive interpretation of example 61.

61 Tweety can fly because Tweety is a bird and Tweety has wings because Tweety is a duck.

Example 61 is interpreted as shown in Figure 37a, which is correct according to the grammar, but its interpretation is not the most intuitive one. It would be more intuitive to interpret it as two separate arguments, as drawn in Figure 37b, incidentally joined through 'and' in a single sentence.

Cons

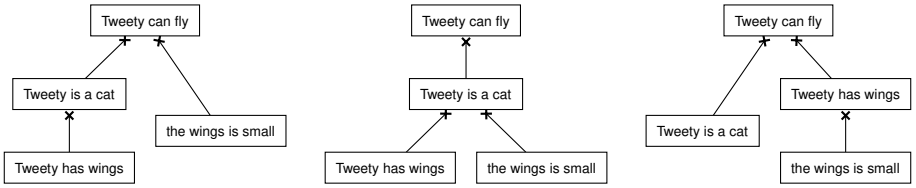
Simple attacks like example 46 are interpreted correctly. Mutual attacks such as example 50 can be formulated, but are not picked up automatically, even though HASL/1 is able to identify negated claims as such.

46 Socrates is mortal but Socrates is a god.

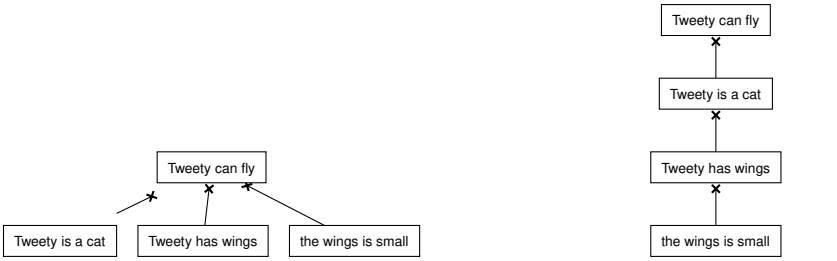
50 Tweety can fly but Tweety cannot fly.

64 Tweety can fly but Tweety is a cat but Tweety has wings but the wings are small.

Example 64 has five interpretations, shown in Figure 38. The grammar allows both the left and right side of 'but' to be an `argument` instead of just a `claim`, which causes the attacking claims to attack each of the previously mentioned claims.



(a) Interpretation 1 (b) Interpretation 2 (c) Interpretation 3



(d) Interpretation 4 (e) Interpretation 5

Tweety can fly but Tweety is a cat but Tweety has wings but the wings are small

Figure 38: Interpretations of example 64 by HASL/1.

Pros and cons

70 Tweety can fly because Tweety is a bird but Tweety is a penguin.

65 Birds can fly but Tweety can not fly because Tweety is a penguin.

In example 70 'Tweety is a penguin' is interpreted as a rebutter, an undercutter, and an underminer. Two of these are defensible interpretations, only Tweety is a penguin attacking Tweety is a bird is not.

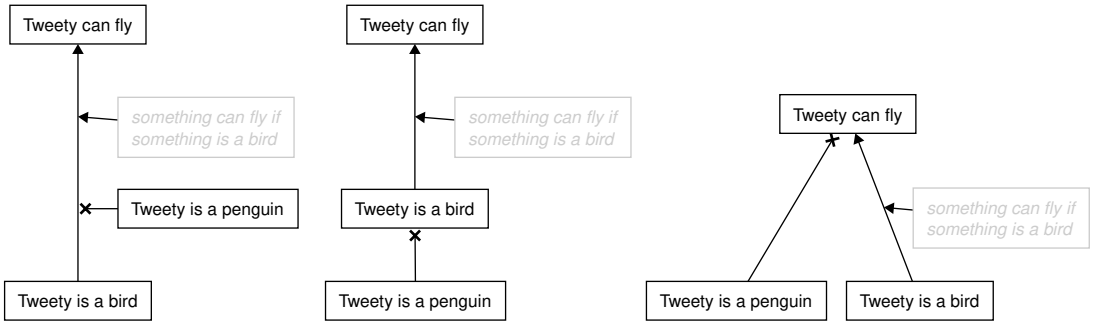
The interpretation of example 70 in Figure 39b shows that the grammar in the sentence allows for a nonsensical argument: our definition if penguin excludes this interpretation. If HASL would have had access to such world knowledge these interpretations could be excluded.

Example 65 (Figure 40) shows the attack of a general claim, which itself can be supported. Enthymeme resolution fills in the missing general claim for this support.

Undercutter

Example 76 (Figure 41) shows the classic example of undercutter by Pollock. Again, this argumentative sentence is interpreted by HASL/1 in all three ways. Only the first interpretation (Figure 41a) is the correct one. The correct interpretation cannot be determined solely based on grammar.

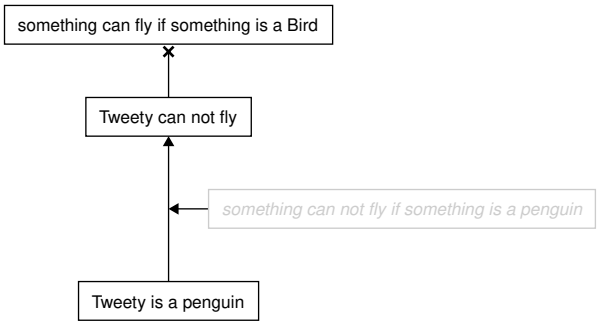
76 The object is red because the object appears red to me but it is illuminated by a red light.



(a) Undercutter (b) Underminer (c) Rebutter

Tweety can fly because Tweety is a bird but Tweety is a penguin.

Figure 39: Argument diagrams for example 70.



Birds can fly but Tweety can not fly because Tweety is a penguin.

Figure 40: Argument diagram for example 65.

Warranted support

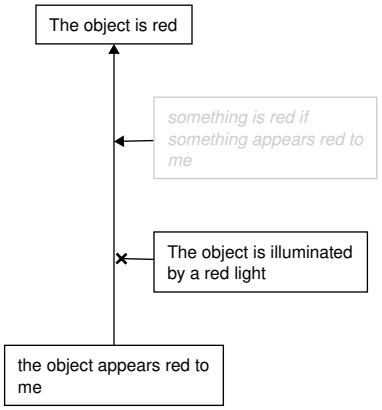
General and specific claims are interpreted correctly and placed accordingly in either a role supporting the conclusion, or supporting that support-relation that supports the conclusion.

79 Tweety can fly because he is a bird and birds can fly.

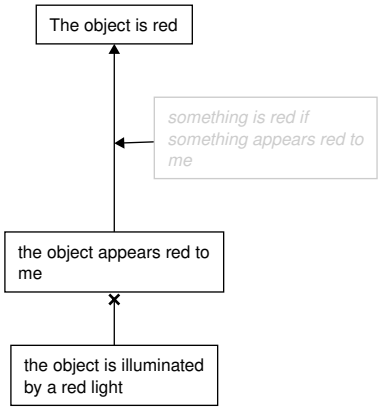
The distinction whether a claim is a general claim or a specific claim is based on the subject of the claim. In example 79 this would be 'Tweety', which is specific, 'birds', which is general, and 'he', which is specific again. Sentence 79 is interpreted correctly, as shown in Figure 42.

82 Tweety can fly because he is a bird and thieves are punishable.

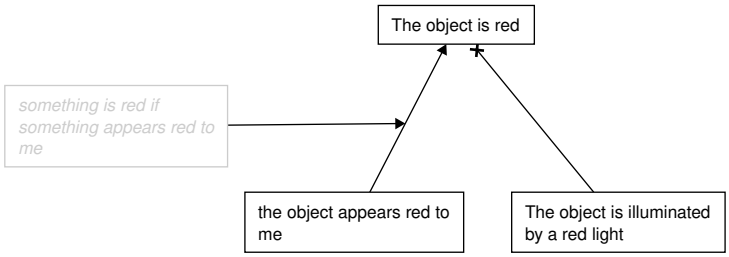
HASL/1 does not check whether the general claim that serves as warrant fits with the specific claim. For example, in example 82 a general claim that is not related to the argument at hand is mentioned, and it



(a) First interpretation



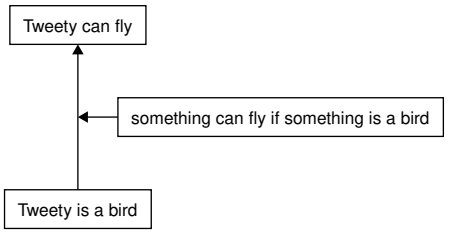
(b) Second interpretation



(c) Third interpretation

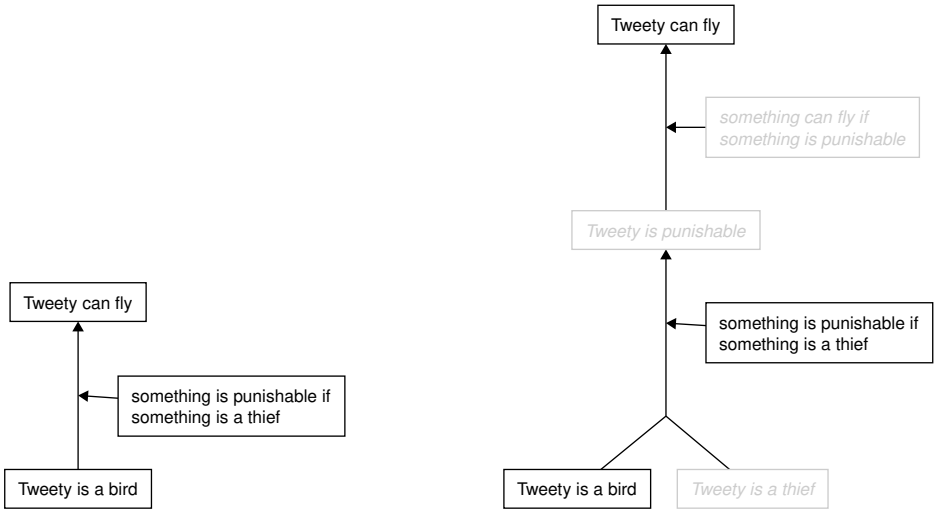
The object is red because the object appears red to me but it is illuminated by a red light.

Figure 41: Argument diagram for example 76.



Tweety can fly because he is a bird and birds can fly.

Figure 42: HASL/1's interpretation of example 79.



(a) First interpretation.

(b) Second interpretation with assumptions.

Tweety can fly because he is a bird and thieves are punishable.

Figure 43: HASL/1's interpretation of example 82.

is interpreted (Figure 43a) as the warrant for the support relation. The argument does not make much sense, but HASL/1 has interpreted what was written.

The enthymeme resolution causes a second interpretation of example 82. This interpretation (Figure 43b) adds the assumption that 'Tweety is a thief' based on our usage of the general claim here, and the intermediate conclusion that 'Tweety is punishable' for similar reasons. From this it also assumes that something can fly if something is punishable.

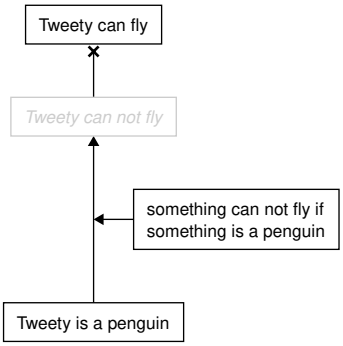
Anaphora resolution

116 The queen can rule because she is born in a royal family and because the king abdicated.

117 The queen can rule because the king abdicated and because she is born in a royal family.

118 The queen can rule because the king abdicated and she is born in a royal family.

Anaphora resolution (also shown in the other examples) works for most occurrences of pronouns that occur in the subject of claims, referring to other entities that also occur in the subject of claims. Examples 116 and 117 work as expected. In example 118 'she' is connected to 'the king' as there is no knowledge available on the gender of nouns or pronouns.



Tweety can fly but Tweety is a penguin and penguins cannot fly.

Figure 44: Enthymeme resolution applied to a warranted rebutter.

Enthymeme resolution

Syllogisms always consist of three elements, but not all of them have to be explicitly stated. Example 28 is the complete syllogism, and examples 119, 120 and 121 are the enthymemes. These are all interpreted correctly, and the missing claim is filled in.

(28) Socrates is mortal because he is a man and men are mortal.

119 Socrates is mortal because men are mortal.

120 Socrates is mortal because he is a man.

121 Socrates is a man and men are mortal.

124 Tweety can fly but Tweety is a penguin and penguins cannot fly.

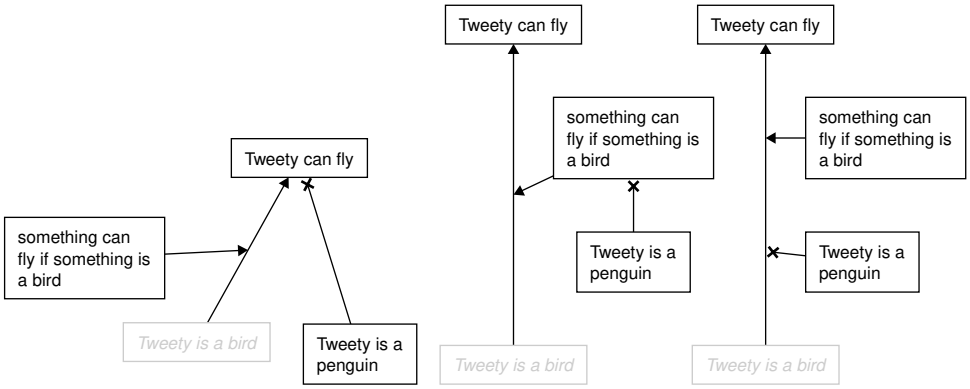
122 Tweety can fly because birds can fly but Tweety is a penguin.

Example 124 (shown in Figure 44) is interesting because the enthymeme is better hidden. Here, the intermediate conclusion that Tweety cannot fly is not explicitly mentioned. This intermediate conclusion is identified and filled in, and becomes the actual rebutter to the claim that Tweety can fly.

In example 122 (shown in Figure 45) 'Tweety is a bird' is assumed because of the general claim. 'Tweety is a penguin' attacks either the general claim 'birds can fly' (Figure 45b) or 'Tweety can fly' (Figure 45a), while it should undercut the relation between 'Tweety is a bird' and 'Tweety can fly' (Figure 45c).

4.11 Discussion

HASL/1 is able to parse simple and even more complex argumentative text, as long as it matches the specifically purposed grammar for English. For our examples this is sufficient, but for wider usage this becomes an obstacle. It also pollutes the focus of the grammar, which is argumentation, but now contains more rules on how to write a specific claim than on how to combine these.



(a) 'Tweety is a penguin' as a rebutter. (b) 'Tweety is a penguin' attacking the major premise. (c) Desired interpretation

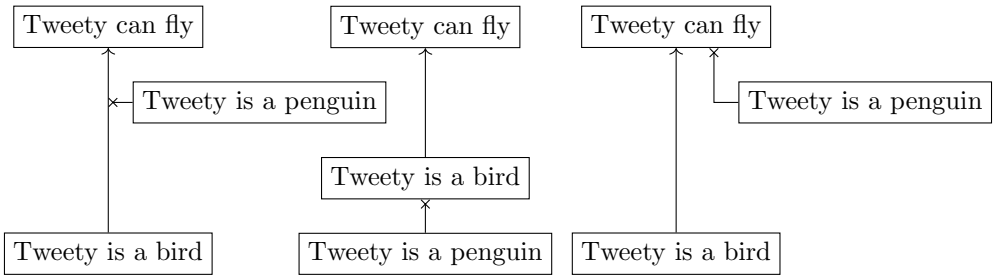
Tweety can fly because birds can fly but Tweety is a penguin.

Figure 45: Argument diagram for example 122.

Pros and cons Support relations are easy to express and behave in a very intuitive way. The distinction between cooperative, independent and chained support in textual form is very natural. To enforce the language to be at least somewhat understandable, and limit the ambiguity of sentences, we restrict this only to the last claim in a cooperative or independent support clause. For example, in 'A because B and C because D' B could not have been an argument on its own. This does not restrict the arguments that can be modelled as we can always still support or attack B in a second sentence. In the argument diagram B is drawn as a single box.

Attack relations are a different beast. In argumentative text they almost always are the last claim in a sentence due to their nature of attacking a chain of thought, closing an option. We model this in HASL/1 as well. Figure 46 shows an example sentence that contains both a support and an attack. We cannot determine which claim attacks which in such a construction, and all possible combinations are presented as argument diagrams. A sentence in the form of 'A because B but C' always yields three possible parses, as C is interpreted as a rebutter, undercutter and underminer. We cannot choose between these three based on syntactic features, and HASL/1 does not have a semantic understanding of the words in the argument.

Furthermore, attacks in sentences such as 'A but B because C' are odd. For example, take 'Tweety is a bird but Tweety cannot fly because Tweety is a penguin'. This would be interpreted as Tweety cannot fly being an argument against Tweety being a bird. Ideally, we would fill in that Tweety is a bird but Tweety cannot fly is really the argument that Tweety is a bird, birds generally can fly, but Tweety is an exception to this general case as Tweety cannot fly. Interpreting attacking arguments is hard, as there are often enthymemes involved.



(a) Interpretation as an undercutter.

(b) Interpretation as an underminer.

(c) Interpretation as an rebutter.

Tweety can fly because Tweety is a bird but Tweety is a penguin.

Figure 46: Three interpretations of an attack on a supported claim.

Rules We differentiate between general and specific claims by assuming that specific claims always have a very specific subject: a name, a noun preceded by the determiner 'the', or a pronoun such as 'he'. General claims have subjects such as plural nouns without determiner, or with the determiner 'a'. When such a general claim occurs in the list of claims of a cooperative support, the general claim is drawn as supporting the support relation itself. This generally works well, but limits the topics that can be discussed in HASL/1's language. For example, penguins are more specific birds, and as such, claims about penguins should be interpretable as specific claims in the context of a discussion involving birds as the general claim. But due to our strict separation on only allowing entities to be the subject of specific claims, this is not possible.

The general claims as they occur in the text do become part of the argument structure, and as such can be attacked and supported. This is both relevant for Toulmin's model (as a way to model the backing) as well as for the versatility of the argument language: we do not want to create a class of claims that can only be asserted, not discussed.

The general claims that are allowed are limited. For example, we can only express vague claims such as 'birds can fly', or at most 'red birds can fly'. For that we call these rule-like statements, we cannot express them as rules. There is no grammar for 'something can fly if it is a bird', and there is no structure for general claims that have more complex conditions. For our examples this is sufficient, but for arguments that follow stricter rules, a more expressive general claim would be interesting, especially so as this would allow for more interesting enthymeme resolution.

Enthymeme resolution Enthymeme resolution works for the simplest cases where the subject of an argument is consistent. Sentences such as 'birds can fly' are interpreted as rules: 'something can fly if something is a bird'. The subject and verb plus object of claims are then used to construct the missing minor premises major premise or conclusion. Grammar rules for arguments that only have a conclusion and a major premise, or no conclusion, are added. The grammar rule for a conclusion supported by minor premises is extended to predict the major premise. These predicted claims are drawn in a different shade, to indicate that they are not explicitly mentioned in the textual argument. Claims that were predicted during parsing the sentences, but do occur explicitly in a

following sentence, are merged in a single box, like any other claim that occurs multiple times in the textual argumentation.

Enthymeme resolution is very simple and can only resolve one step. For example, it is not implemented for attacking claims where often an attacking claim can be interpreted as a supporting claim for a conclusion opposite to the conclusion it attacks, which itself is then a rebuttal to the conclusion. Each of these steps/relations can be warranted, and these warrants should also be made explicit by predicting their contents. This would make the argument diagram much more verbose, but given the option to toggle these assumed claims and relations, it could help identify unstated and weak arguments, helping the critical evaluation of arguments.

By adding the missing elements of enthymemes of arguments to the argument diagram we do change in a sense the interpretation of the argument. Enthymemes may be intentional, and the resulting augmented argument may not be what the original author of the argument intended. Even though they are drawn in a different shade, the meaning of the argument or weight of the explicitly stated claims may become distorted, or it may become clear that the incomplete argument was, after being completed, a bad argument (Walton and Reed, 2005).

5 HASL/2: A flexible understanding

HASL/2 extends the core idea of HASL/1, but tries to overcome three of HASL/1's weaknesses: the underlying English grammar, the limited structure for general claims, and the ability to also use the grammar to formulate argumentative text.

The grammar in HASL/1 is a mix of English language constructions (i.e. how to write a claim) and argumentation constructions (i.e. how to write a support relation between two claims). In the end, we are only interested in the latter. The former is necessary, but also limiting as only claims that can be expressed in this grammar can be used in the construction of arguments. In HASl/2 we leave out the grammar for claims, opting instead for splitting sentences into claims and discourse markers purely on the presence of those discourse markers. This would no longer limit what can be expressed in claims, as HASL/2 will not try to understand those at a level that HASL/1 did.

Additionally HASL/2 has a better formalised structure for major claims: their conditions and exceptions are no longer ad-hoc encoded in the claim itself, but are an integral part of the argument diagram. This allows us to write more expressive general claims, and also understand the construction of these better by expressing them in a similar structure as those of arguments in the same argument diagram.

Finally, the grammar for argumentation itself could be used to also translate an argument diagram to argumentative text, but the implementation of HASL/1, which is build on a limited grammar for English, does not allow for this. In HASL/2 we add the ability to formulate argumentative texts given an argument diagram by applying we use to parse argumentative text in reverse on the argument diagram. This would allow us to use HASL/2 as a tool to explore the understanding of argumentative text, e.g. try to draw an argument diagram and see how this is expressed in text, and then in turn see how these texts can also be interpreted as some constructions can be formulated in different ways, and similarly some argumentative texts can be interpreted in different ways.

(29) Tweety can fly because if something is a bird it can fly.

(30) Tweety can fly because birds can fly.

For HASL/1, examples 29 and 30 are one and the same argument only formulated differently, but only the latter can be parsed. HASL/2 is able to parse both sentences and create the same argument diagram for each of them, or take that argument diagram and generate at least these two sentences.

The process for parsing text in HASL/2 is similar to that of HASL/1, but simpler. Like in HASL/1, text is first segmented into tokens, and these tokens are then parsed into a structured argument. Different from HASL/1, these tokens are not single words and punctuation, but spans of text: claims are segmented into a single token. Therefore the grammar has to only consist of rules that are relevant to the argumentative structure of the text.

To facilitate these features, HASL/2 makes use of an intermediary representation of arguments: the structural representation. The structural representation can be transformed to the diagram representation used by HASL/1 and for our drawings, but also to the textual representation, i.e. argumentative texts. These three representations of the same argument are shown in Figure 48.

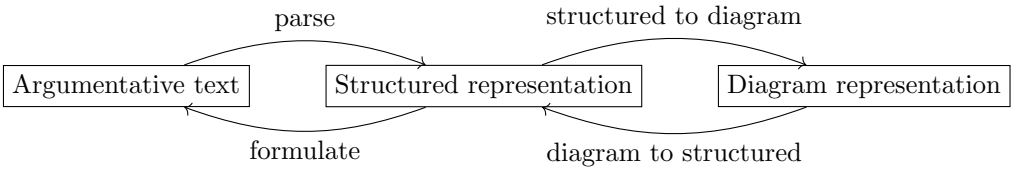


Figure 47: Transformation steps in HASL/2.

5.1 Segmentation

HASL/2 cuts the strings of text it receives as input into tokens by separating them using discourse markers (Marcu, 1997) to segment the text into stand-alone parts (argumentative discourse units, ADU's, or as we call them: claims).

These discourse markers are keywords such as 'but', 'because', etc. We also use conjunctives such as 'and', 'or', punctuation, and any other explicitly mentioned token in the grammar. A complete list of all discourse markers used by HASL/2's grammar can be found in Table 4. To illustrate, the following listing compares how the sentence 'Tweety can fly because birds can fly.' is split into respectively tokens or clauses by HASL/1 and HASL/2.

- * HASL/1: 'Tweety' 'can' 'fly' 'because' 'birds' 'can' 'fly' '.'
- * HASL/2: 'Tweety can fly' 'because' 'birds can fly' '.'

5.2 Data structure

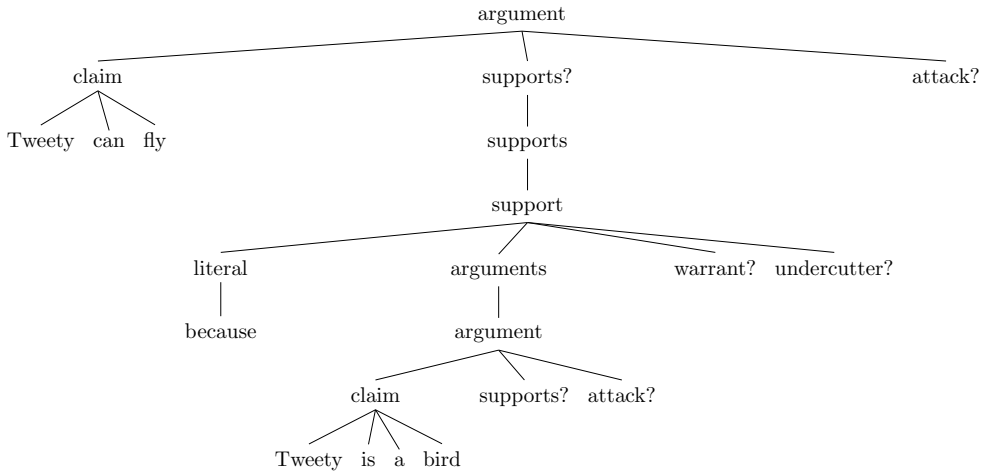
HASL/2 requires both the description of the structure as well as the interpretation parts of the grammar to be reversible. In HASL/1 the transformation rules that were applied as soon as a grammar rule was finalized and transforms the parsed text segments into a local argument structure are implemented as imperative Python code. This code cannot be run in the opposite direction (from local argument structure to parsed text segments).

Structural representation predicates We construct a limited data structure that implements the functionality we need for this transformation that is bidirectional. This data structure contains several constructs that allows us to map the data from one representation onto the other. It consists of three predicates:

slot The slot primitive maps elements from the textual grammar onto the nested structure of the interpretation part of the grammar and vice versa. The textual grammar is represented as a list (one element for each terminal or non-terminal) and via `slot(`n`)` the nth element of that list can be accessed. When reversed (i.e. going from interpretation to a sentence) this maps the structure at that position in the structural representation to be reversed into a textual representation by the grammar for the nth non-terminal.

tlist The `tlist`[2] creates a list that consists of a head and a tail, either element can be either a value or one of the predicates like

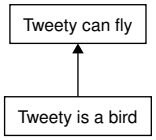
[2]Called tlist as the name 'list' was already taken by a native structure in Python.



(a) Parse tree

```
Argument(
  claim = Claim(Tweety can fly)
  supports = [
    Support(
      datums = [
        Argument(
          claim = Claim(Tweety is a bird)
          supports = []
          attack = None
            )
      ]
      warrant = None
      undercutter = None
    )
  ]
  attack = None
)
```

(b) Structural representation



(c) Diagram representation

Tweety can fly because Tweety is a bird.

Figure 48: Three representations of the same sentence.

`slot(n)`. `tlist` is used to construct lists in the structural representation, for example when multiple supporting arguments support the same claim.

template Using `template` allows us to structure the parts of the structural representation into data structures described below, and to identify which data structure it has. E.g. a supporting claim is mapped via `template` to a *Support* structure as described below. When reversing the structural representation, the `template` primitive will only be reversible if the structure is of the same type. Using `slot` and `tlist` we can link values to the properties of the structure (e.g. link the list of supporting claims to the *datums* property of the *Support* data structure.)

Using these rules we can create grammar rules such as the following two examples, that can be used in both directions:

⟨*argument*⟩ ::= ⟨*claim*⟩ [⟨*supports*⟩] [⟨*attack*⟩]

This grammar rule, including transformation function, is implemented using the predicated described earlier. In our Python code, it is expressed in the following way:

```
rule('argument',
['claim', 'supports?', 'attack?'],
template(Argument, claim=slot(0), supports=slot(1), attack=slot(2))),
```

In this example, the first line contains the name of the rule, 'argument'. The second line expresses which rules it consists of, i.e. a claim and optionally a list of supports and an attack. The last line shows the transformation function. If this grammar rule is matched, and each of the rules on its right hand side can be filled in (where the optional rules are allowed to be filled in with empty matches) then this transformation function is used to construct the result of this grammar rule.

The `template` predicate constructs a new *Argument*, which *claim* is then filled in by `slot(0)` with the result of the first element on the right hand side of the rule, i.e. `<claim>`. The same is done for *supports* and *attack*. *supports* is a list, and since it is optional it may be an empty list.

The following example shows how these lists are constructed using the simple example of `<sentences>` since the actual rule for `<supports>` also deals with the discourse markers between them, i.e. the comma's and 'and'.

⟨*sentences*⟩ ::= ⟨*sentence*⟩ ⟨*sentences*⟩

The rule is translated into our Python format of our implementation as follows:

```
rule('sentences',
['sentence', 'sentences'],
tlist(head=slot(0), tail=slot(1))),
```

In the above example the recursive case of `<sentences>` rule is defined, which consists of a single sentence followed by itself. The single `<sentence>` rule, when matched, yields an *Argument* instance, which is then connected to `slot(0)` as it is the element at index 0. The `<sentences>` rule yields a `tlist`, which is connected to `slot(1)`. As a result, this rule extends that `tlist` with the *Argument* as head. This way a list of tokens that matches sentences is transformed into a list of Arguments.

We extend the data structure with more named elements so that they can be matched with the `template` primitive described above when reversing an argumentative structure.

- Claim
  - **Text**: the text of a claim that is drawn in the box or forms a clause in the textual argument.
- Argument
  - **Claim**: the *Claim* that is the conclusion of this branch in the argument.
  - **Supports**: a list of *Support* items that each are an argument for supporting the conclusion (i.e. as in independent support).
  - **Attack**: an optional *Attack* structure attacking the conclusion of this argument.
- Attack
  - **Arguments**: a list of one or more arguments that in a cooperative way attack the conclusion.
- Support
  - **Datums**: a list of *Claim*s that together support the conclusion of the argument that this Support structure is a part of.
  - **Warrant**: an optional *Argument* that warrants the relation, telling us why it is allowed to interpret the datums as support.
  - **Undercutter**: an optional *Argument* that undercuts this support. It does not attack any of the individual claims, but instead argues that the supporting clause is not (or no longer) relevant.
- Warrant
  - **Claim**: a *Claim* that is the conclusion of this warrant, e.g. 'Something can fly'.
  - **Conditions**: a list of *WarrantCondition*s in disjunctive normal form that indicate when the claim of this warrant can be applied to a certain entity. E.g. conditions could be 'if something is a bird' and 'if something has wings', and generally if one of the claims in the *Datums* attribute of the *Support* structure matches either of those for the subject of the *Claim* of the *Argument* which is supported by that *Support* structure, it should apply.
- WarrantCondition
  - **Claims**: a list of *Claim*s in conjunctive normal form (i.e. cooperative support) that indicate when this condition is true, e.g. 'something is a bird and something is not a penguin'. These are the conditions that need to be explicitly stated as or being assumed as being the case.
  - **Exceptions**: a list of *WarrantException*s in disjunctive normal form (i.e. independent support) indicating under which circumstances the warrant cannot be the case according to this condition. E.g. 'something is a penguin' could be a exception. By stating it as an exception instead of a condition it is assumed to not be the case, unless explicitly stated in the argument.
- WarrantException

- **Claims**: a list of *Claim*s in conjunctive normal form. Most often it is just one claim, e.g. 'something is a penguin', but it could be that an exception is very specific, and due to the rule-like nature of the argument diagram structure used by HASL/2 we specify all clauses of the exceptions (and conditions) explicitly.

The *Warrant* structure mirrors the *Claim*, and the *Support* mirrors the *WarrantCondition*. Only in the latter there is no attribute for a warrant and no room for multiple exceptions: in the case of an supporting argument in a direct claim, one undercutter is enough to stop a supporting argument from being relevant. In the context of a warrant, which is more like the mentioning of a rule, one can state multiple exceptions to the application of that rule. Stating these happens only to inform on their existence, it does not apply them to this particular case. For example, in a sentence like 'Tweety can fly because he is a bird except he is a penguin.' there is no need to also state that his wings are too short, just being a penguin is enough to attack the claim that Tweety can fly. In a warrant, the sentence is more rule-like and not directly applied to a specific context: 'birds can fly unless they are a penguin or their wings are too short'. It is relevant to state multiple exceptions in this scenario as there may be (we don't know yet) a datum that matches one of these conditions. Additionally, in HASL/2 we use the warrant to describe rule-like text such as a body of law. In such texts, there is no ongoing argument, there is only a structure of when certain claims can be made (e.g. when an act is tortious) and under what exceptional circumstances they can't be stated.

5.3 Parser

HASL/2 parses sentences using depth-first search as this algorithm is easier to adapt to use the grammar rules in both directions and is sufficient for demonstration purposes.

We do not identify part-of-speech tags, which we did for HASL/1, as the smallest elements are now spans of text, and we do not analyse these spans in depth. Only discourse markers are identified.

Note that we still explore the full search space. When there are multiple possible rules to match (e.g. multiple rules with the same name) all of these are tried.

Because we now use depth-first search, all grammar rules are only allowed to be right-recursive. Left-recursive rules would cause an infinite search space.

Interpreting Each time a token is matched to a literal in the grammar, it is consumed. How the token is processed depends on the literal it matches against. For example, literals that match discourse markers just check whether they match, but literals that match the spans of text between discourse markers (i.e. claims) capture and transform these to a span of text that will eventually be displayed as a box in the argument diagram.

When all elements of a rule are complete, a rule consumes these elements as well. This is where the structure of arguments is created. For example, the grammar rule that matches an `<argument>`, which is constructed of a claim with one or more supports, creates the *Argument* data structure at this stage.

Formulating The reverse grammar works the same way, except that first the top-most data structure is reversed using the rule. I.e. *Argument* is reversed into a list of its components based on the `<argument>` grammar rule.

5.4 Grammar

The grammar rules now consist of a name, the constituents which are names of terminals (discourse markers) and non-terminals (other rules), and a structural representation of the argumentative structure that matches the rule constructed from the primitives described earlier.

Each rule starts with a name, and is then followed by the constituents which refer to other grammar rules, or terminals such as 'because' in the second rule. Terminals, which are literals that also serve as the discourse markers, are surrounded by quotes, rules by guilletes.

Some of the rules use square brackets in the antecedent. These sections are optional, and can also match an empty list of tokens. In the parse trees and our implementation these are displayed as a rule name ending with a question mark.

Each rule has an associated transformation function, written using the `slot`, `tlist` and `template` predicates described earlier. In the first rule, when a text matches with the `<argument>` rule, it yields a structural representation in the form of an instance of *Argument*, which has a property named *claim* that holds the structural representation matched by the `<claim>` rule in the grammar, and a property *supports* that does the same for `<supports>`.

We make the distinction between argumentation and rules. In argumentation, a claim is proposed (e.g. Tweety can fly) and directly supported or attacked by other claims ('he is a bird' or 'he is a penguin'). In rules, we describe this same logic, but do not apply it directly to a claim that is discussed here and now., e.g. birds can fly unless they are penguins. More often, rules themselves are not the target of the discussion, but they are mainly discussed in whether they are applicable in the argument or not, or whether an explicit or implicit exception to the rule is the case.

We first define the grammar for argumentation. The grammar for rules (encoded in `<warrant>`) will be defined thereafter.

Arguments

⟨*sentences*⟩ ::= ⟨*sentence*⟩ ⟨*sentences*⟩
  | ⟨*sentence*⟩

⟨*sentence*⟩ ::= ⟨*argument*⟩ '.'

Arguments are sentences, always with a claim, and optionally with one or multiple reasons in favour of that claim. Optionally there can also be an argument against the claim.

⟨*argument*⟩ ::= ⟨*claim*⟩ [⟨*supports*⟩] [⟨*attack*⟩]

⟨*claim*⟩ ::= ⟨*text*⟩

We allow only a single attacking argument per sentence as this seems natural to do: a sentence seldom contains multiple different attacking arguments, and there is no natural way to write multiple different attacking arguments.

Pros

⟨*supports*⟩ ::= ⟨*support-list*⟩
  | ⟨*support*⟩

⟨*support-list*⟩ ::= ⟨*support*⟩ '`,`' ⟨*support-list*⟩
  | ⟨*support*⟩ '`and`' ⟨*support*⟩

⟨*support*⟩ ::= '`because`' ⟨*arguments*⟩ ['`and`' ⟨*warrant*⟩] [⟨*attack-marker*⟩ ⟨*argument*⟩]

The `<supports>` rule allows us to write 'because a, because b and because c' like constructions. The referencing of `<support-list>` and `<support>` from `<supports>` is different from the way HASL/1 writes these type of self-referencing constructions as the parser we use utilizes a depth-first search approach. As such, referencing the rule itself as the first constituent would allow it to recur infinitely without making progress, causing it to become stuck in an infinite loop.

Cons

⟨*attack*⟩ ::= ⟨*attack-marker*⟩ ⟨*arguments*⟩

⟨*arguments*⟩ ::= ⟨*argument-list*⟩
  | ⟨*argument*⟩

⟨*argument-list*⟩ ::= ⟨*argument*⟩ '`,`' ⟨*argument-list*⟩
  | ⟨*argument*⟩ '`and`' ⟨*argument*⟩

For the attacking argument we allow multiple arguments after the discourse marker marking these arguments as attacking. This way we can construct a combined attack from one or more claims, and each of these claims can themselves be supported or attacked.

⟨*attack-marker*⟩ ::= '`but`' | '`except`' ['`that`']

We separate the discourse markers that mark an attack using `<attack-marker>` to allow us to use different discourse markers. This list can be extended, but these ('but', 'except' and 'except that') are the most common and sufficient for our examples.

Rules We have not yet defined the `<warrant>` rule referenced by the `<support>` rule. The warrant can be interpreted as the major argument in a syllogism, or the warrant in Toulmin's argument model. The warrant is a general rule, or a generalization, e.g. 'birds can fly', or a rule with explicit conditions, e.g. 'an act is unlawful if it violates someone's rights, if it violates an written or unwritten law or if it violates a statutory duty'.

⟨*warrant*⟩ ::= ⟨*claim*⟩ [⟨*conditions*⟩]
  | ⟨*claim*⟩ ⟨*exceptions*⟩

A warrant can consist of a claim with one or more conditions where each condition is a separate reason for the claim part of the warrant to be applicable, e.g. something can fly if it is a plane or if it is a bird.

A warrant can also be just a claim (e.g. 'birds can fly') or a claim with only exceptions 'birds can fly except when they are penguins'.

⟨*conditions*⟩ ::= ⟨*condition-list*⟩ '`or`' ⟨*condition*⟩
  | ⟨*condition*⟩

⟨*condition-list*⟩ ::= ⟨*condition-list*⟩ '`,`' ⟨*condition*⟩
| ⟨*condition*⟩

⟨*condition-marker*⟩ ::= '`if`' | '`when`'

⟨*condition*⟩ ::= ⟨*condition-marker*⟩ ⟨*claims*⟩ [⟨*exceptions*⟩]

Conditions themselves can have multiple claims. This to write sentences that form a condition in a combined claim like structure, e.g. when all of the claims need to be the case for the condition to be applicable. Optionally each (combined) condition can have one or more exceptions: cases in which all of the claims in the condition are true but when the conclusion is not applicable.

Exceptions Exceptions can partially be interpreted as conditions that must not be true, with the difference that, like in default logic, the claims in conditions need to be explicitly the case while the exception is implicitly not the case, unless it is mentioned elsewhere.

(31) Something can fly when it has wings and it has an engine or when it is a bird.

(32) Something can fly when it has wings or when it is a bird unless it is a penguin or its wings have been clipped.

In contrast to the undercutter in a supporting argument, a warrant can have multiple exceptions. In the indirect case, there is a need to be able to mention all the exceptions to a rule since there no direct evaluation: the avenue in the argument doesn't end as soon as the exception is mentioned, as it is only a hypothetical case.

⟨*exceptions*⟩ ::= '`unless`' ⟨*unmarked-exceptions*⟩
| '`except`' ⟨*marked-exceptions*⟩

Exceptions come in two types, marked and unmarked. These are just to differentiate between writing 'unless a or b' and 'except if a or if b', where in the latter the 'if' (or 'when') is repeated with each exception to the rule. They are interpreted in the same way.

All in all, the examples 31 and 32 should be interpreted as shown in Figure 49b.

⟨*unmarked-exceptions*⟩ ::= ⟨*unmarked-exception-list*⟩ '`or`' ⟨*unmarked-exception*⟩
| ⟨*unmarked-exception*⟩

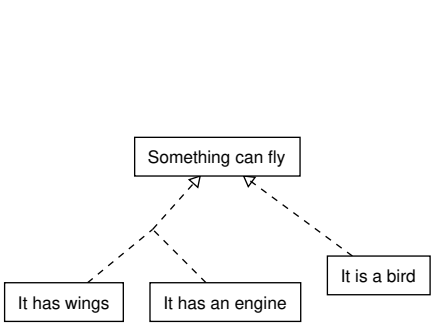
⟨*unmarked-exception-list*⟩ ::= ⟨*unmarked-exception-list*⟩ '`,`' ⟨*unmarked-exception*⟩
| ⟨*unmarked-exception*⟩

⟨*unmarked-exception*⟩ ::= ⟨*claims*⟩

⟨*marked-exceptions*⟩ ::= ⟨*marked-exception-list*⟩ '`or`' ⟨*marked-exception*⟩
| ⟨*marked-exception*⟩

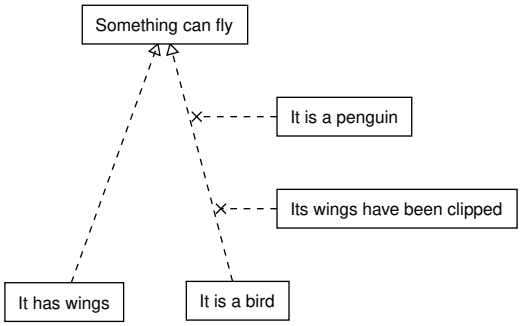
⟨*marked-exception-list*⟩ ::= ⟨*marked-exception-list*⟩ '`,`' ⟨*marked-exception*⟩
| ⟨*marked-exception*⟩

⟨*marked-exception*⟩ ::= ⟨*condition-marker*⟩ ⟨*claims*⟩



Something can fly when it has wings and it has an engine or when it is a bird.

(a) Interpretation of Example 31.



Something can fly when it has wings or when it is a bird unless it is a penguin or its wings have been clipped.

(b) Interpretation of Example 32.

Figure 49: Interpretations of rules.

5.5 Structured arguments and argument diagrams

Each argument has three representations, shown in Figure 50: The parsing of a sentence using the described grammar yields a parse tree. For each branch in the tree the transform function of the rule associated with that branch is applied (the transform function describes the mapping from the rule's antecedents to a data structure using the *slot*, *tlist* and *template* primitives).

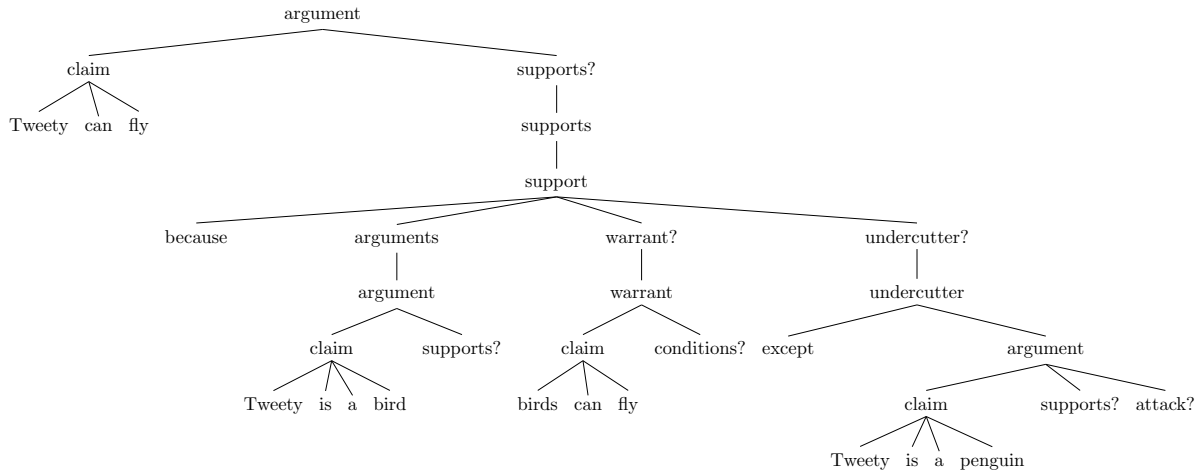
The resulting data structure, the structural representation, is a stricter representation of the argument. For example, it describes that a top-level argument consists of a single claim, which itself can only be text, with optionally multiple supporting arguments and an attacking argument. A supporting argument can have one or more sources, but optionally only a single warrant. It describes the rule-structure of the warrant, etc.

This opposed to the diagram representation of the argument, which can only describe boxes (claims) and arrows (relations), with the option for arrows to originate from multiple boxes as a way to encode combined support/attack relations.

Transitioning between the structural representation and the diagram representation can be done in both directions for arguments that fit both representations.

Structured to diagram For example, to construct an diagram representation from a structural representation of an argument, we can simply create relations for each of the supporting arguments of the argument, as well as create an attack relation if there is an attacking argument.

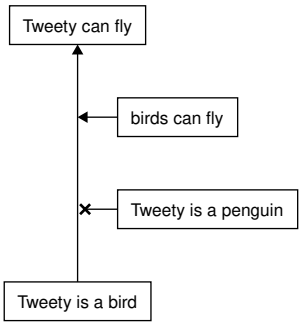
Diagram to structured To transition in the opposite direction, we first find the top-most claims in the argument diagram, and for each of these we identify the supporting and attacking arguments by following the relations in the diagram representation.



(a) Parse tree

```
Argument(
  claim=Claim(Tweety can fly)
  supports=[
    Support(
      datums=[
        Argument(
          claim=Claim(Tweety is a bird)
          supports=[]
        )
      ]
      warrant=Warrant(
        claim=Claim(birds can fly)
        conditions=[]
      )
      undercutter=Argument(
        claim=Claim(Tweety is a penguin)
        supports=[]
      )
    )
  ]
)
```

(b) Structural representation



(c) Diagram representation

Tweety can fly because Tweety is a bird and birds can fly except Tweety is a penguin.

Figure 50: Three representations of the same sentence.

If there are multiple attacking arguments for an argument, a choice needs to be made, as this cannot be described in the structured representation of an argument. We account for this difference by creating multiple structured representations (i.e. multiple sentences) with each of them one of the attacking arguments. Only the first sentence will have the supporting arguments as there is no way to represent the grouping (which attacking argument should be grouped near which supporting arguments) in the diagram representation.

Other structures in the diagram representation, such as relation which targets the relation between the warrant and the attack or support relation the warrant supports, cannot be represented in the structured representation. Even in the diagram representation it has no meaning, but that structure is allowing enough to encode it. These feature of an argument that cannot be represented in the structured representation are lost in the transition.

5.6 Generating textual arguments

The generation of text from an argument diagram also occurs in two stages (see Figure 47). First it converts the diagram representation to a structural representation. This conversion yields a list of *Argument* structures, one for each sentence. Then the parser applies the grammar rules in a depth-first search on this structural representation. For each rule, the transform function is applied in reverse. If this succeeds, e.g. if the rule describes an *Attack* structure and also finds such a structure at that position in the structural representation, the search continues according to the antecedents of the grammar rule. If it does not, this search avenue is skipped and the next rule (i.e. a different rule with the same name) is tried. As the parser does an exhaustive search, all search avenues are tried, hence if there are multiple solutions for a rule, each of them is once applied in one of the results of this stage. For example, as there are three ways to fill in the `<attack-marker>` rule, each attack yields three results, each with another attack marker.

The results of this reverse application of the parser are lists of tokens. These can be concatenated together to form sentences.

5.7 Implementation

HASL/2 is implemented using the same architecture as HASL/1, as a web service in Python that returns JSON.

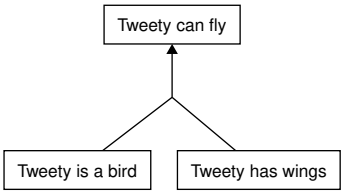
Interface The main interface of HASL/2, shown in Figure 52 has a panel for argumentative text on the left side and diagrams on the right side. The one can be translated to the other using the buttons on top. If multiple diagrams or formulations are created, they are each shown in a separate tab.

Both the text and argument diagram representations can be edited. Arguments can be changed, supporting or attacking arguments can be added or deleted through mouse and keyboard interaction. Clicking the *comprehend* or *formulate* buttons will update the other representation with the changes made.

Saving and loading diagrams The *save* and *load* diagram buttons allow you to copy and edit the diagram as text. This is the same text

```
a: Tweety can fly
b: Tweety is a bird
c: Tweety has wings
d: b c supports a
```

(a) Diagram code



(b) Diagram

Figure 51: Diagram representation used for saving, storing and editing diagrams.

representation as can be copied from HASL/1's interface and which is used for the argument diagrams in this thesis. This allows us to experiment with interpretations between HASL/1 and HASL/2, edit the diagrams in HASL/2, and print them.

This diagram representation (shown in Figure 51) is only used to make saving and editing the diagrams convenient, as it is easy to understand and change. It also allows for the styling of the diagrams through controlling the positions or width of the boxes directly, but this is optional. If the positions are not specified, the layout is calculated automatically.

5.8 Evaluation

We begin with evaluating HASL/2 using the same tests as HASL/1, listed in subsection A.4. The complete evaluation of HASL/2 on these examples is listed in Table 5. Here we highlight the observations from that evaluation.

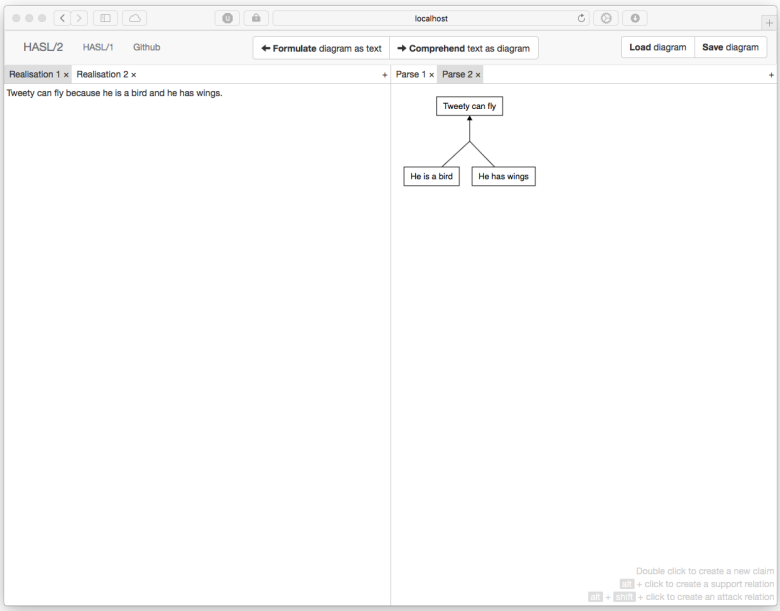
Pros

57 Tweety can fly because he has wings and he has feathers and because he is a bird.

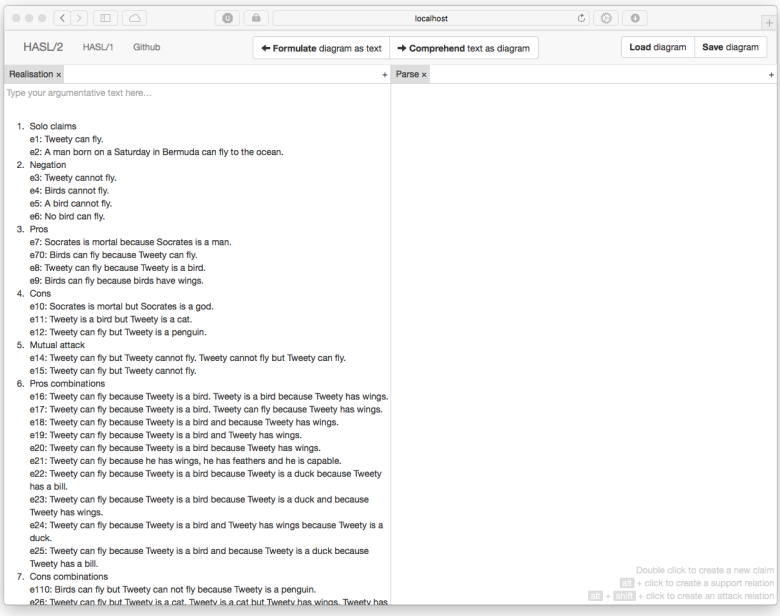
60 Tweety can fly because Tweety is a bird because Tweety is a duck and because Tweety has wings.

HASL/2 has a slightly different approach to capturing cooperative and independent support than HASL/1. Note that HASL/2 cannot determine the difference between a general and a specific claim, and does not make this distinction. As a result, like with HASL/1 and the attack relations, it does not make a choice and presents both interpretations. In Figure 53 and in Figure 58 this becomes visible. The only option in which this does not occur is when the last claim of a claim list is in the form of a claim that can only be interpreted as a general claim according to the `<warrant>` rule, i.e. if it contains explicit conditions.

HASL/2 does not limit the nesting in its supporting or attacking arguments. So while HASL/1 interpreted example 60 as Tweety having wings supporting Tweety being a bird, HASL/2 also interprets it as a claim supporting the conclusion 'Tweety can fly' (Figure 54).

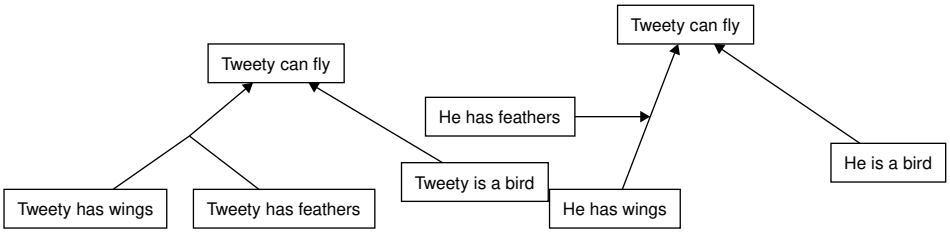


(a) HASL/2's interface after a text is interpreted into a diagram, and the diagram has been formulated as text.



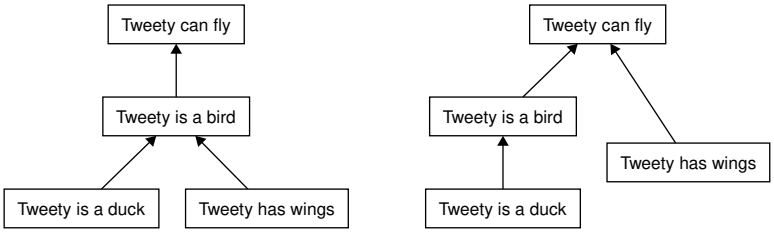
(b) HASL/2's interface when first opened. Tab panels on the left side show the evaluation sentences when empty. On the right side argument diagram can be drawn.

Figure 52: The interface of HASL/2.



Tweety can fly because he has wings and he has feathers and because he is a bird.

Figure 53: Interpretation of example 57, containing both cooperative and independent supports in HASL/2.



(a) First interpretation. (b) Second interpretation.

Tweety can fly because Tweety is a bird because Tweety is a duck and because Tweety has wings.

Figure 54: Interpretation of example 60, containing both cooperative and independent supports in HASL/2.

Cons

Attacking arguments are more difficult to place correctly in the diagram as was earlier observed in HASL/1.

70 Tweety can fly because Tweety is a bird but Tweety is a penguin.

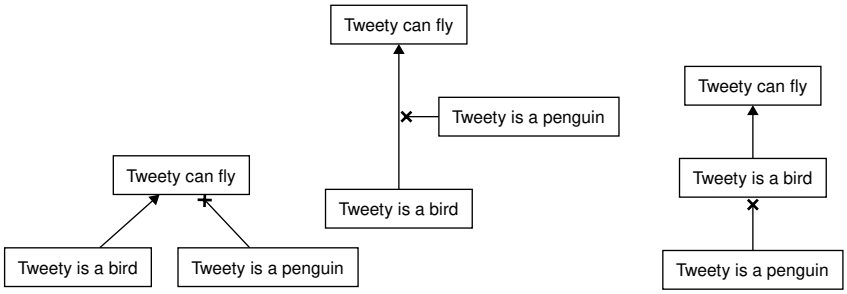
122 Tweety can fly because birds can fly but Tweety is a penguin.

65 Birds can fly but Tweety can not fly because Tweety is a penguin.

64 Tweety can fly but Tweety is a penguin but Tweety is a cat.

The distinction between attacking the main claim, the supporting claim, or the relation between those two cannot be determined from the grammar. This is also what we see when we parse example 70 with HASL/2. All three possible interpretations of the attack are shown in the three diagrams in Figure 55.

In example 122 (Figure 56) there is only one supporting claim, but it is a general claim. HASL/1 is able to interpret this as an enthymeme where this claim is the major premise, and the minor premise is missing.



(a) Being a penguin opposes being able to fly.

(b) Being a penguin attacks the inference because Tweety is a bird it is able to fly.

(c) Tweety being a penguin attacks the claim that Tweety is a bird.

Tweety can fly because birds can fly but Tweety is a penguin.

Figure 55: Interpretations of example 70 with an attacked argument in HASL/2.

HASL/2's grammar rules specifically expect there to be at least one minor premise in a support argument, hence even though the claim 'birds can fly' is a general rule, it is interpreted and only interpreted as a minor premise in Figure 56.

For example 65 HASL/2 has only one interpretation, shown in Figure 57a. This interpretation is as expected.

Example 64 shows a difference with HASL/1. In HASL/2, the attack can only attack the previous claim. There is only one parse for this sentence, shown in Figure 57b.

Warranted support

As already seen, HASL/2 does not make the distinction between specific and general claims. As a result, it cannot determine based on the words in the claim whether a supporting claim should be interpreted as a claim supporting the conclusion, or a claim supporting the relation, supporting the support of the conclusion.

79 Tweety can fly because birds can fly and he is a bird.

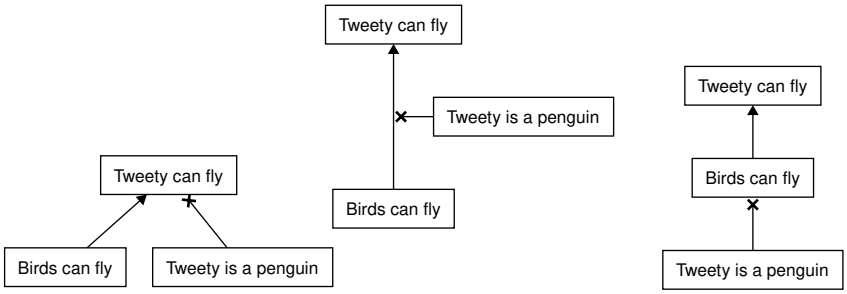
Sentence 79, shown in Figure 58, is unambiguous in its intentions. HASL/2 however cannot determine whether this is cooperative support, or warranted support.

Rules

Compared to HASL/1 the grammar for HASL/2 regarding rules has been expanded. We can now also formulate rules with explicit conditions, as they occur often in law, and with explicit exceptions.

102 Something can fly when it has wings and it has an engine or when it is a bird.

104 Something can fly if it has wings except when its wings are too short, when they are clipped or when they are broken.



(a) Being a penguin opposes being able to fly.

(b) Being a penguin attacks the inference because birds can fly Tweety should be able to fly.

(c) Tweety being a penguin attacks the general rule that birds can fly.

Tweety can fly because birds can fly but Tweety is a penguin.

Figure 56: Interpretations of example 122 in HASL/2.

Example 102, shown in Figure 59 shows how such a rule is interpreted. Example 104 (Figure 60a) shows how these rules can also contain exceptions. However, the grammar is ambiguous and cannot distinguish between reading multiple exceptions, or reading multiple conditions of which one is an exception (Figure 60b).

Arguments with rules

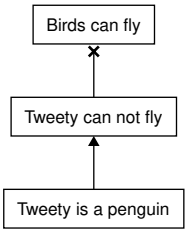
108 Socrates is mortal because he is a man and something is mortal if something is a man.

110 Socrates is mortal because he is a man and something is mortal if something is a man or if something is a mammal.

In example 108 the major premise from example 28 is changed to an explicit rule. This is interpreted as shown in Figure 61a. Example 110, interpreted in Figure 61b shows this extended further.

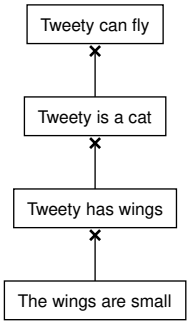
Tort A more real life example is based on the translation of Betlem (1993) of Dutch Tort law: Article 6:162 Definition of a 'tortious act' (unlawful act). The text in example 126 has been adapted from the quoted source to form self contained claims. Sentences like 'As a tortious act is regarded ... ' are simplified to have their structure be presented more clearly: 'an act is tortious if ...'.

126 A person must repair the damages if he committed an act against another person, the act is tortious, the act can be attributed to him and the other person has suffered the damage as a result thereof.
The act is tortious if there was a violation of someone else's right, if an act or omission is in violation of a duty imposed by law or if an act or omission is in violation of what according to unwritten law has to be regarded as proper social conduct unless there was a justification for this behaviour.
The act can be attributed to him if it results from his fault or if it



Birds can fly but Tweety can not fly because Tweety is a penguin.

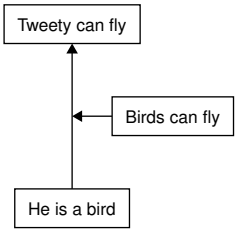
(a) Example 65.



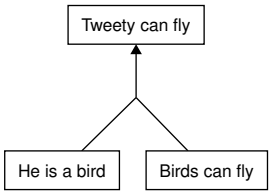
Tweety can fly but Tweety is a cat but Tweety has wings but the wings are small.

(b) Example 64.

Figure 57: Interpretation of attacking arguments in HASL/2.



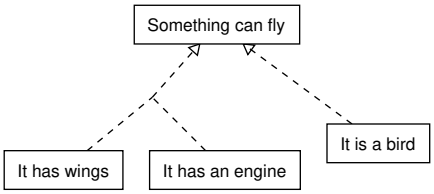
(a) Correct interpretation.



(b) Incorrect interpretation.

Tweety can fly because he is a bird and birds can fly.

Figure 58: HASL/2's interpretation of example 79.



Something can fly when it has wings and it has an engine or when it is a bird.

Figure 59: Interpretation of example 102, containing both cooperative and independent conditions in HASL/2.

> results from a cause for which he is accountable by virtue of law or generally accepted principles.

The interpretation of this text is shown Figure 62. The intended interpretation is shown in Figure 63. The two cannot be compared one to one since our interpretation (Figure 62) is constructed from taking segments of a text while the reference diagram (Figure 63) is constructed from manual interpretation of the original law text and contains more exceptions. If we disregard these differences, only the exception for justification is different. This is something we cannot easily express in HASL/2's grammar, except by repeating it for each of the situations.

Formulating argumentative text

HASL/2 can transform argument diagrams into text by applying the grammar rules in reverse.

Support and Attack Let us take the three argument diagrams in Figure 55 as basis for exploring HASL/2's formulation of arguments.

Figure 55a is formulated in three sentences. The are all almost the same, but differ in their usage of some discourse markers.

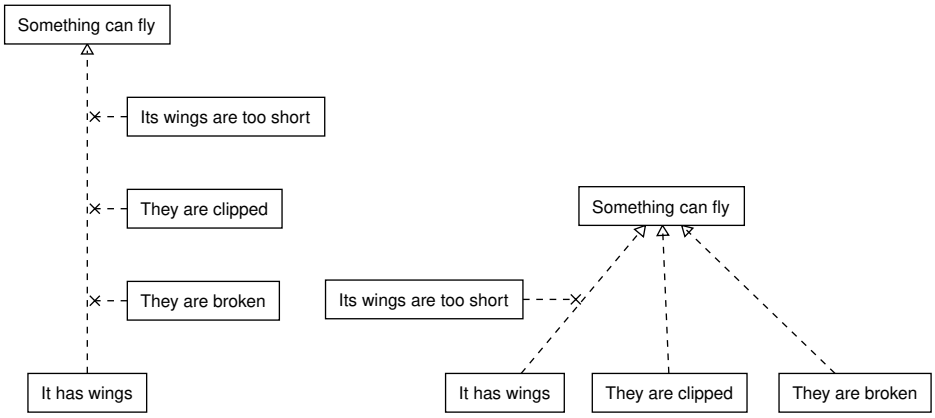
1. Tweety can fly because tweety is a bird but tweety is a penguin.
2. Tweety can fly because tweety is a bird except tweety is a penguin.
3. Tweety can fly because tweety is a bird except that tweety is a penguin.

Figure 55b has five formulations. The three listed above, as well as the following two:

1. Tweety can fly if tweety is a bird unless tweety is a penguin.
2. Tweety can fly if tweety is a bird except if tweety is a penguin.

Again, we see the different discourse markers, but we also see that for the initial three formulations HASL/2 interpreted the top level of the argument diagram as a specific claim, with a reason for it, and an undercutter. In the two new formulations, the argument diagram is interpreted as a general argument, a rule with a condition and an exception.

Figure 55c has four formulations. Again the three initially presented formulations, and the following:



(a) Correct interpretation.

(b) Incorrect interpretation.

Something can fly if it has wings except when its wings are too short, when they are clipped or when they are broken.

Figure 60: Interpretation of example 104, showing multiple exceptions in a rule in HASL/2.

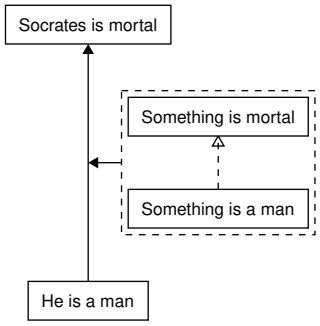
1. Tweety can fly if tweety is a bird.

This formulation is incomplete and incorrect. This is caused by HASL/2 interpreting the diagram as a general rule, but a condition to a general rule cannot be attacked in HASL/2's grammar. The claim 'tweety is a penguin' is lost.

Tort If we ask HASL/2 to formulate the diagram presented in Figure 62, it presents us with 193 different formulations, all equal except for slight variations (i.e. where 'but' has been replaced with 'except that'). The first formulation is written below.

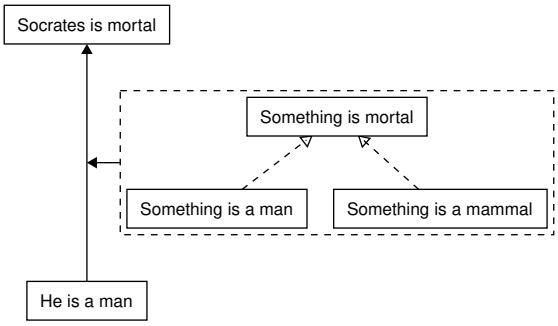
> A person must repair the damages if he committed an act against another person, the act is tortious, the act can be attributed to him and the other person has suffered the damage as a result thereof.
> The act is tortious if there was a violation of someone else's right, if an act or omission is in violation of a duty imposed by law or if an act or omission is in violation of what according to unwritten law has to be regarded as proper social conduct unless there was a justification for this behaviour.
> The act can be attributed to him if it results from his fault or if it results from a cause for which he is accountable by virtue of law or generally accepted principles.

If we offer the correct diagram from Figure 63 to HASL/2 to formulate, it constructs 3456 formulations, all again the same but with slight alterations in wording. The first of these formulations:



Socrates is mortal because he is a man and something is mortal if something is a man.

(a) Example 108.



Socrates is mortal because he is a man and something is mortal if something is a man or if something is a mammal.

(b) Example 110.

There is a duty to repair someone's damages if someone has suffered damages by someone else's act, the act committed was unlawful, the act can be imputed to the person that committed the act and the act caused the suffered damages unless the act is a violation of a statutory duty and the violated statutory duty does not have the purpose to prevent the damages.
The act committed was unlawful if the act is a violation of unwritten law against proper social conduct, if the act is a violation of a statutory duty unless there exists grounds of justification or if the act is a violation of someone's right unless there exists grounds of justification.
The act can be imputed to the person that committed the act if the act is imputable to someone because of common opinion, if the act is imputable to someone because of law or if the act is imputable to someone because of the person's fault.

5.9 Discussion

Splitting text on discourse markers has proven to be effective Marcu (1997), Reitter (2003).

By leaving out almost all grammar that deals with English sentences HASL/2's grammar focusses mainly on argumentation. As a result, the grammar is small and focussed. However, it remove the ability to make a distinction between general and specific claims based on the language they use, and HASL/2 always presents both interpretations.

Data structure The intermediate representation is necessary to match rules when applying the grammar in reverse, but is yet another constraint on the link between grammar and structure. It is also not always compatible with the structure, e.g. the structure of a general claim in HASL/2 does not allow for attack relations on conditions, but these can be drawn in the argument diagram. Maybe we can implement the grammar without this structural representation if we can use primitives that work less as predicates, and more like database search and mutation rules. Then an

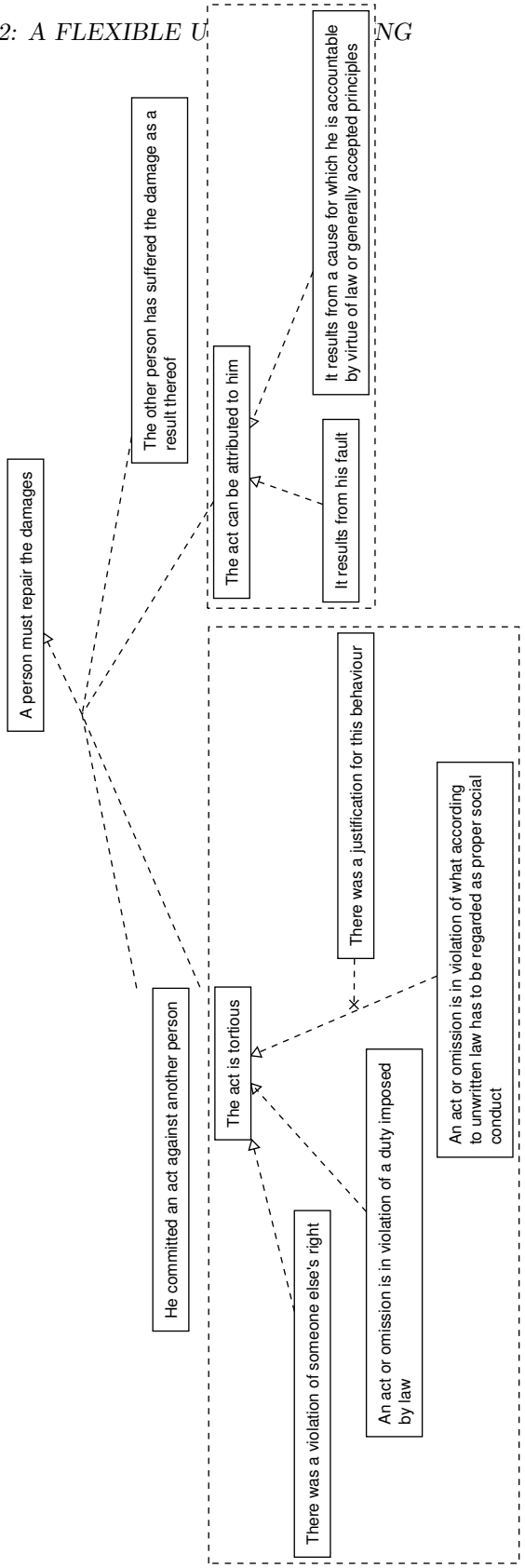


Figure 62: Interpretation of the tort law example 126 in HASL/2.

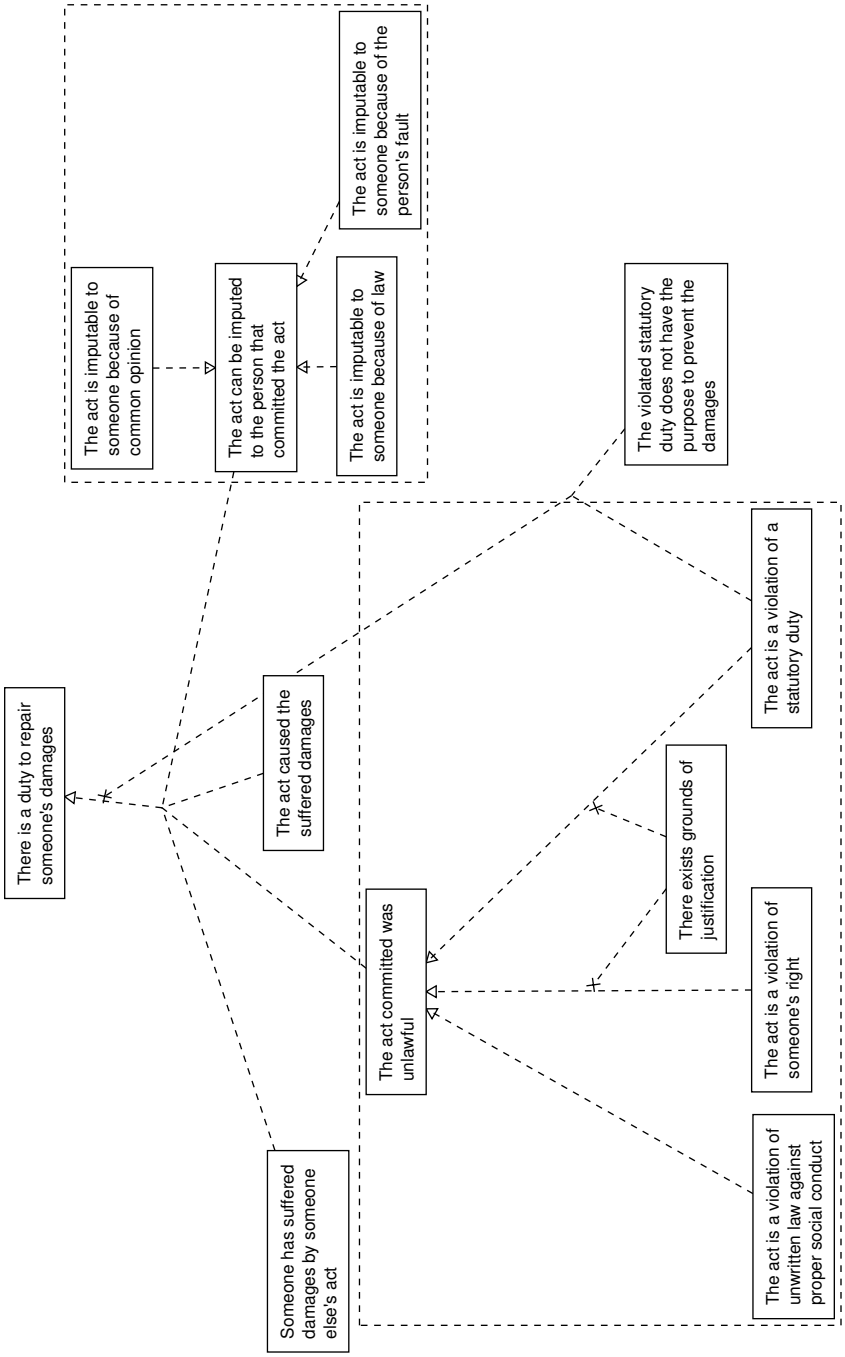


Figure 63: Argument in Dutch Tort law, adapted from Figure 2 from Verheij (2017).

argument can be approached as a database of claims and relations, and these transformation rules search, add and remove claims and relations from this database. A formulation would only be accepted if the database is empty, like a parse is only accepted when all tokens have been matched.

Pros HASL/2 allows for writing supporting arguments in both cooperative and independent support. These supports can also be warranted. However, the grammar cannot identify the difference between a specific and a general claim and as such assumes that if there is a general claim, it will be the last of the claims that occurs in the same structure as which is recognized as a cooperative supporting argument. These sentences will always yield multiple interpretations, unless the general claim uses grammatical constructions that cannot occur in a specific claim. A general claim in a position other than the last will not be recognized as such.

The grammar does not limit the usage of expanded arguments to the last supporting argument, as is done in HASL/1. As a result, one is free to use complicated language throughout and this does allow us to write more ambiguous sentences which one would otherwise not be writing. In practise, this poses more of a limitation when formulating arguments, as HASL/2 will use these rules to formulate hard to comprehend sentences. Ideally, we would use the same limitations we implemented in HASL/1 which describe which parts of the sentence may elaborate, and which need to stay simple.

Cons HASL/2 suffers the same fate as HASL/1 in that it cannot choose the type of attack relation. Additionally, HASL/2 has no enthymeme resolution, therefore if there is only one supporting claim for a conclusion, it is assumed to be a specific claim as in HASL/2 claims cannot be supported by only a general claim serving as major premise or warrant.

The attack relation in a sentence with multiple attacks is in HASL/2 more predictable than the one in HASL/1 because the attacking claim cannot be an argumented claim. For writing argumentative text with a specific argument diagram in mind, this is helpful, as one can still support or attack the attacking claim by writing separate sentences for those arguments. In HASL/1 a strict interpretation could only be achieved by writing separate sentences, or in some cases (e.g. undercutter) not at all.

Rules and exceptions HASL/2's grammar focusses more on argumentation than the grammar of HASL/1. As a result, the type of argumentative structures that can be expressed is extended to also cover the structure that can be found in the general claims.

The interpretation (Figure 62) is almost correct, except for the 'justification for this behaviour'. Here this is interpreted as an undercutter, but ideally it would have been an undercutter for all three relations supporting the 'the act is tortious' claim, or as an rebuttal of that claim. However, this is not possible with the current grammar in HASL/2.

Ideally we would need a combination of the anaphora resolution of HASL/1 with the extended grammar of HASL/2. This would allow for better anaphora resolution, better deducing which claims need to be added, and eventually evaluation.

6 General Evaluation

We have evaluated each of the HASL implementations on their own, to get an idea on whether they behave as expected, and whether the grammar they implement and design choices they are bound to form any limitations, and what these might be. This evaluation was done using the examples in subsection A.4 and the results are listed in Table 5.

Table 2 lists which requirements we set for HASL, and which are achieved by HASL/1 and HASL/2. This is further elaborated in this section.

Goal	HASL/1	HASL/2
Express pros: claims supporting a claim	✓	✓
Express cons: claims attacking a claim	✓	✓
Express an argument consisting of multiple supporting and attacking claims	✓	✓
Combine such expressions into a multiple sentences or a single, more complex sentence	✓	✓
Express cooperative and independent support and attack	✓	✓
Express warrants and undercutters	✓	✓
Express Toulmin arguments		
Identify warrants	✓	-
Interpret rule-like claims	✓	✓
Anaphora resolution	✓	-
Enthymeme resolution	✓	-
Formulate argumentative texts	-	✓

Table 2: Goals of HASL

6.1 Pros

Both HASL/1 and HASL/2 can express support structures, including cooperative and independent support, as well as chained support.

Independent and chained support can be expressed both by collapsing them into a single sentence in the form of 'A because B and because C' and 'A because B because C', as well as by writing separate sentences in the form of 'A because B. A because C.' and 'A because B. B because C.' This gives us the freedom to formulate a human readable text.

Supporting claims is a bit limited in HASL/1, as there are no grammar rules to allow general claims to be supported by general claims. HASL/2 does not have this limitation as it doesn't make the distinction between general claims and specific claims.

Syllogisms In HASL/1 and HASL/2 we make the distinction between supporting claims, and supporting relations. The latter serves as a way to model a warrant, a general statement that 'warrants' concluding the conclusion based on the claim. This often takes the form of a syllogism, where the warrant is a major premise and the claim the minor premise. We model this in HASL/1 with the general and specific claim.

79 Tweety can fly because he is a bird and birds can fly.

This split between general and specific claims based on whether the claim looks like a 'general' statement or a 'specific' statement works well in HASL/1 as it helps it determine whether a claim has the role of a major or minor premise. For example, example 79 makes use of this. In HASL/1 this sentence is interpreted correctly (Figure 42). In HASL/2, which cannot make the distinction between general and specific claims, this sentence has two interpretations (Figure 58).

Categorial syllogisms The categorical syllogism differs from the syllogism in that there are no claims that match HASL/1's definition of a singular claim: every claim is designating a group or a category.

44 Birds can fly because birds have wings.

84 Birds can fly because birds have wings and creatures with wings can fly.

Sentences such as example 44 and 84 cannot be interpreted by HASL/1. For HASL/2 they are no different than other syllogisms of this form, and are interpreted in a similar fashion. One of the goals of HASL, namely escaping the difficulties of language in argument mining by focussing on the structure, is not reachable by HASL/1 due to its dependency on a basic grammar for English.

6.2 Cons

All claims and relations can be attacked in HASL, and as such both claims made and the connections that are claimed to be there between claims can be countered. The exception to this are the relations between general claims and their conditions.

The approach of HASL for identifying which claim is attacked is limited because the attacking claim always comes at the end of the sentence. At that moment it can apply to any of the claims previously stated in the last sentence in the case of HASL/2, Figure 57b shows this, or even in any of the previous sentences in the case of HASL/1, visible in Figure 38.

While support relations can be combined into cooperative and independent relations, attack-relations cannot in HASL/1. Through repetition of the attacked claim independent attacks can be expressed, but cooperative attack is not possible. Supporting the attacking claim, or attacking the attacking claim (i.e. chained attack, which can be described as reinstatement) can be expressed. HASL/2 treats support relations and attack relations equally, and the same grammar rules apply for both.

6.3 Rules

In HASL/1 the support for rules as general claims is limited as it does not allow us to express explicitly state conditions for rules. In fact, the way general claims are written in the diagram (i.e. 'Something can fly if something is a bird') cannot be parsed by the grammar of HASL/1. This interpretation into this rule-like structure where a general claim becomes a claim with a condition was purely for purpose of enthymeme resolution, and for that purpose this level of detail is sufficient.

In HASL/2 the structure for rules as described in section 3.6 is implemented in the grammar. This allows us to capture complex rules, such as tort. To disambiguate this rule structure from the argument structure

we draw the condition and exception relations with dashed arrows. To indicate that the whole rule, including its conditions, is treated as a single general claim in the argumentation, the arrow that indicates its use as a warrant is drawn from a dashed box that includes the complete rule and its conditions. In this vision, the way HASL/1 interprets rules with exceptions (cf. as single claims) is correct.

Anaphora

Only HASL/1 implements anaphora resolution since HASL/2 does not parse claims and as such is not able to identify pronouns or entities they could refer to in claims.

The anaphora resolution in HASL/1 works well to create more descriptive argument diagrams. For most arguments it works well as it follows the relations in the argument in cooperative and independent support, which works well with the expected interpretation of pronouns. HASL/1 is not able to identify whether or not 'she' can refer to 'the king' as there is no background knowledge. It is also limited to the subject of claims, as entities and anaphora after the verb are just treated as text, and not as entities that can be linked. In the way anaphora resolution is implemented, it piggybacks on the infrastructure necessary to perform enthymeme resolution. HASL/2 does not implement any form of anaphora resolution. Although the claims in the diagrams are a bit less self-contained, the lack of anaphora resolution has no impact on its ability to identify the argument structure in text.

Enthymemes

Resolving enthymemes mainly helps with the understanding of arguments. Only HASL/1 is able to perform this step. For simple claims where only one part of the syllogism is missing (i.e. examples 119, 120 and 121) this works well except when the connection between the general and specific claim is not made, e.g. when 'thieves' and 'thief' not treated as the same thing in example 86.

Example 124, which was shown in Figure 44 is interesting in that way as the enthymeme resolution takes place deeper inside the argument. It shows how enthymeme resolution can complete the argument and make it easier to understand.

Formulating argumentative texts

In formulating HASL/2 uses the collapsing of sentences to formulate a single sentence for complex arguments, as is shown in section 5.8. The grammar for rules does not allow for this, thus complex rules such as example 126, the excerpt from Dutch Tort law, remain readable.

Toulmin

Our argument schema can emulate most elements of Toulmin's model. The datum, warrant and conclusion directly map to our warranted support structure. We do not have a specific element for the qualifier of an argument, and our attacks are specifically attacking a claim in the argument as where Toulmin gives attacking arguments their own place in the argument itself.

We formulated two of Toulmin's example arguments, Examples 75 and 33. Toulmin himself did not write them down in such a precise formulation, but for our purposes we need a purely textual representation. In the first example we intentionally left out the warrant to test whether this is picked up by HASL/1. In the second example we do state the warrant ('a Swede can be taken...'), and support it with a backing ('because the portion of...').

75 Harry is a British subject because Harry is a man born in Bermuda but Harry has become naturalized.

(33) Petersen will not be a Roman Catholic because he is a Swede and a Swede can be taken almost certainly not to be a Roman Catholic because the proportion of Roman Catholic Swedes is less than 2%.

Example 75 shows the same ambiguity in interpretation we saw in both HASL/1 and HASL/2: the are three ways to interpret the attack. In two of the three interpretations the warrant is identified.

In Figure 65 the interpretations of Example 33 are shown. We have drawn both the interpretations without and with enthymeme resolution to highlight the effect of adding unstated claims to the argument diagram. It shows that although qualifiers are not intentionally taken into account, their meaning is picked up and reflected in the argument by HASL/1: enthymeme resolution does show the difference between 'will not be' and 'can be taken almost certainly'.

Furthermore the backing is always correctly placed as supporting the warrant.

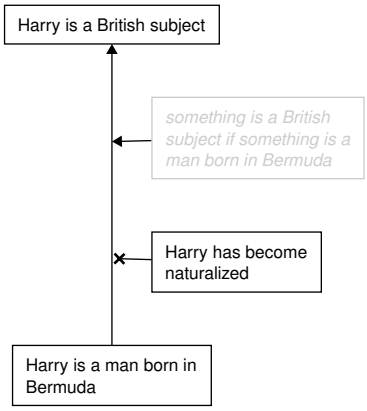
Tort

In the concept of HASL, Tort is very similar to categorial syllogisms, in that it is all general claims. In this there is a clear distinction between HASL/1 and HASL/2.

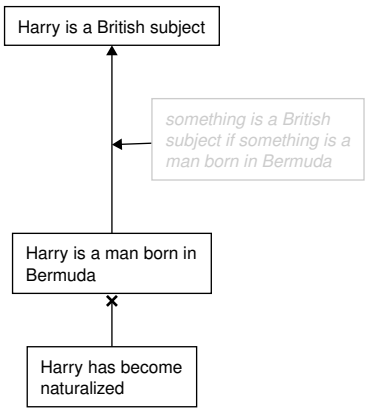
HASL/1 has only a very limited concept of general claims in terms of rules. The major premise of a syllogism is acceptable, but more complex rules are not. HASL/1 only really allows us to express relations using 'because'. This is perfect for the application of tort law, but not to describe tort law itself.

HASL/2 can handle general claims, or more specifically conditional claims that are expressed in terms of 'if' and 'when'. It is less a discussion of support and attack, and more a discussion of potential reasons: conditions and exceptions. These can mostly be expressed in the same argument diagram, as they are in HASL, but they have different grammar for the same structures.

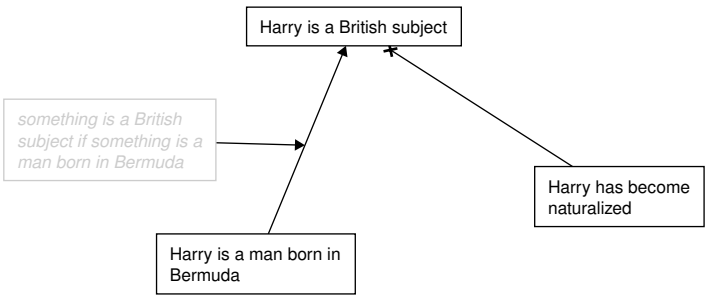
To conclude, HASL/1 is able to understand the application of tort law, but HASL/2 is able to comprehend the law itself. Finally, HASL/2 can comprehend the combination of the law itself, as its application, shown in Figure 66. Unfortunately there is no enthymeme resolution like in HASL/1 to fill in the assumptions made in such an argument.



(a) First interpretation: Being naturalized is an exception to the general rule.



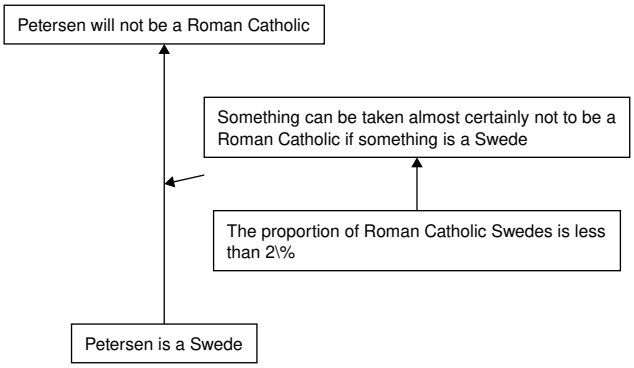
(b) Second interpretation: Being naturalized is an argument against being born in Bermuda.



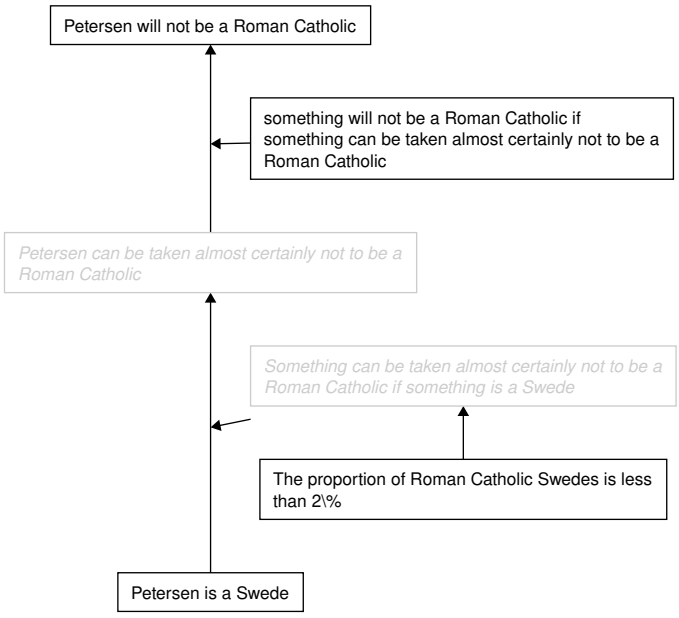
(c) Third interpretation: Being naturalized is an argument against being a British subject.

Harry is a British subject because Harry is a man born in Bermuda but Harry has become naturalized.

Figure 64: Argument diagram from HASL/1. Shaded boxes are reconstructed major premises.



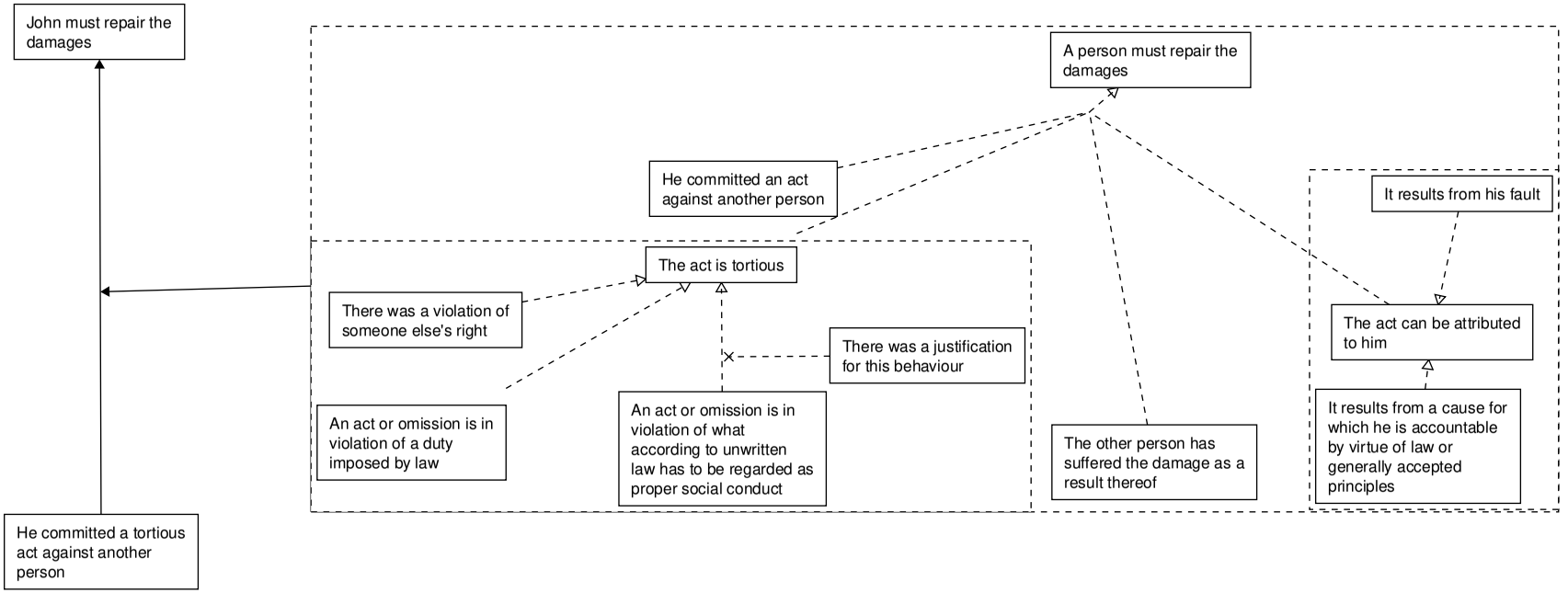
(a) Interpretation without enthymeme resolution.



(b) Interpretation with enthymeme resolution.

Petersen will not be a Roman Catholic because he is a Swede and a Swede can be taken almost certainly not to be a Roman Catholic because the proportion of Roman Catholic Swedes is less than 2%.

Figure 65: Interpretation of Example 33.



John must repair the damages because he committed a tortious act against another person and a person must repair the damages if he committed an act against another person, the act is tortious, the act can be attributed to him and the other person has suffered the damage as a result thereof.
The act is tortious if there was a violation of someone else's right, if an act or omission is in violation of a duty imposed by law or if an act or omission is in violation of what according to unwritten law has to be regarded as proper social conduct unless there was a justification for this behaviour.
The act can be attributed to him if it results from his fault or if it results from a cause for which he is accountable by virtue of law or generally accepted principles.

Figure 66: Interpretations of tort law and its application in HASL/2.

7 Discussion

The goal of HASL is to understand argumentative text, and to use this understanding to interpreted and formulate arguments. To this goal, we approached the argumentative text as a 'natural' language that we transform into an argument structure using a parser and grammar, while trying to keep this language as close to English as possible to make it accessible and intuitive. That language and its grammar rules are coupled to our argument structure. This is all implemented in HASL/1 and HASL/2, two tools that allow users to interpret and formulate such arguments through a simple interface. In this way, HASL incorporates aspects from argumentation theory, argument mining and argumentation tools.

7.1 An argumentation theory perspective

What makes up an argument in HASL is described in section 3. Essentially, every argument is a graph, with claims as nodes, connected by relations between them. By allowing claims to support claims while they themselves are supported by claims, we can express how arguments support or attack other arguments.

Support and attack Using claims and relations between them we can express claims supporting and attacking other claims. By composition we can use this to express a level of detail in attacks, such as undermining (i.e. attacking the supporting claim) and rebutting (i.e. attacking the supported claim) arguments.

Cooperative and independent We can express how claims support or attack other claims together by combining them in cooperative (i.e. a combined arrow, Figure 9b) or independent (i.e. independent arrows, Figure 9a) support and attack relations. This can be a way to express the difference between strong and weak arguments, as weak arguments are weak because they have to rely on each other in a cooperative support relation, while strong arguments can independently support a conclusion. We cannot express whether one relation attacking a claim is stronger than another relation supporting the same claim. In order to be able to achieve that level of descriptiveness we need to add a prioritization to our arguments as is proposed by Vreeswijk (1997), or to our relations in the form of weights, as is typical for Bayesian belief networks.

Warranted support and attack We describe relations between claims and other relations in subsection 3.4, to be able to express claims supporting and attacking the relations that are made between claims. This allows us to describe the argumentation of a discussion on the level of the discussion itself: one might be stating true and indisputable claims, but this should not lead to a supported conclusion when the reasoning on how those claims support the conclusion is not sound. The reasoning behind the claims should be part of the discussion as well, and in our model it is. To these claims that support a relation, and in this way support the reasoning behind that relation, we refer as warrants.

Rule-like warrants In HASL/1 we chose to enforce that warrants always are rule-like claims, i.e. they are general statements such as 'birds can fly' that are interpreted by HASL/1 as a general rule: Something

can fly if it is a bird. In this way, when we have a set of three claims in which the first is supported by the second, and the third supports this relation, we can interpret these as a conclusion, a minor premise and a major premise. The major premise is then always a rule-like statement. Using this format, HASL/1 is able to fill in enthymemes: if anyone of these three is missing, but the other two can be interpreted correctly, the third can be reconstructed.

HASL/2 partially let this idea go. In HASL/2 it is not enforced that the warrant is a general claim, i.e. rule-like sentence. But HASL/2 continued this assumption that a warrant often has the structure of a rule. It implements our model for such rules, described in subsection 3.6 and allows the structure of arguments and the structure of the rules used in those arguments to be expressed in the same diagram.

Toulmin's argument model The claims that support relations in our model can be interpreted the warrant in Toulmin's argument model (Toulmin, 2003). In Toulmin's model the warrant is a generalized rule, often grounded in some more detailed truth which can be expressed in the backing in Toulmin's model. These elements can be expressed in HASL's model as well. All of the elements of Toulmin's model can be expressed in our model: The datum and conclusion are expressed as a claim supporting another claim in our model. The warrant is expressed as a claim supporting the relation between the previous two. The backing is expressed as a claim supporting the claim that serves as warrant. Toulmin's idea of rebuttal can be expressed by a claim attacking any of the previous claims, or the relations. Only the qualifier does not have its own place in our argument model. Toulmin's qualifier is related to the warrant: the warrant decides what the qualifier for the conclusion should be. For example, if the warrant says that a Swede is almost certainly not a Roman Catholic, the conclusion is that Petersen, who is de Swede, is almost certainly not a Roman Catholic. We have modelled this argument in section 6.3, in Figure 65. The enthymeme resolution in HASL/1 will copy the qualifier from the conclusion. So while qualifiers are not explicitly part of our argument model, they are captured and taken into account.

Toulmin searches for a boundary between warrant and backing, arguing that the warrant is a general statement, and the backing is domain-specific. In this interpretation, 'birds can fly' can be interpreted as a backing based in the application of categories of the warrant 'something can fly if it is a bird'. In our argument model we not treat 'birds can fly' and 'something can fly if it is a bird' differently, both are modelled as a claim (although the second has an internal rule-like structure from subsection 3.6). The role of warrant is purely based on the position of a claim in the argument (i.e. a claim supporting a relation), as is the role of backing. In fact, since in our model the warrant is just another claim, the support relation of the backing supporting the warrant itself can also again be supported by a warrant. If we would continue to warrant our support relations, we would probably end up using the axioms of logic as warrant.

Toulmin's argument model has a place for attacking claims, but it does not specify what part of the argument is attacked. It is often drawn as a line to the conclusion, which would in the end be indeed the result: the conclusion should no longer be supported by the argument. Our model allows us to specify the attack on the conclusion explicitly, or we can attack any of the other claims or relations in the argument. As such, we

are more expressive in the meaning of an attack. The result would be the same: were we to evaluate the argument, the attacked claim would directly or indirectly cause the support for the conclusion to fall away.

A cumulative model A notable difference between our model and Toulmin's argument model is that our argument diagrams contains all the claims that have been encountered in an argumentative text. In our structure, arguments that have been attacked, and premises that have been retracted, are all still drawn in the argument diagram as long as they were part of the argumentative text we interpreted. In that sense the argument diagram also shows how an argument and possibly a conclusion has come to be, although there are no elements that indicate which claims have been accepted or rejected in the process, or which premises came before others. This allows us to express phenomena like reinstatement, where the claim that attacks another claim is itself attacked, thereby leaving the initially attacked claim undisputed. This occurs through multiple sentences, which at the end of the last sentence will all have ended up in the argument.

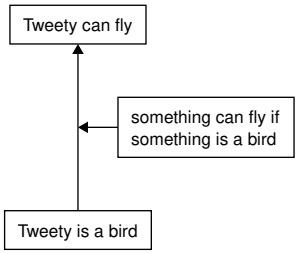
Verheij's DefLog The ability in our model to express claims supporting or attacking relations is similar to the model described in Verheij (2003b) DefLog. DefLog allows any relation to be supported or attacked, and our argument diagrams would allow this as well. In DefLog, these relations can be expressed via $c \rightsquigarrow (b \rightsquigarrow a)$. In our our interpretation, and in Toulmin's argument model, a would be the conclusion, b the datum, and c the warrant. Equally so, $c \rightsquigarrow \times(b \rightsquigarrow a)$ can be interpreted as an undercutter, attacking the relation between b and a. Our diagrams allow such relations to be drawn and interpreted from text as well.

However, if we take this one step further, we can express the attack of a relation between a warrant and the relation it supports, i.e. $d \rightsquigarrow \times(c \rightsquigarrow (b \rightsquigarrow a))$. Even though such relations can be drawn in our diagrams, we do not have a semantic interpretation for such attacks: we do not have grammar rules to write such expressions, nor do we have an idea on how diagrams with such relations should be interpreted. Would it stop the application of the warrant, therefore leaving the support relation from datum to conclusion without warranting? But in our argument diagrams many relations are without a warrant, as warrants are often left unstated. Equally so, if all support relations would have to be warranted, where does this requirement end? We chose not to include claims supporting and attacking our relation between warrant and the relation between datum and claim, but the idea of nodes and edges upon which our model is built allows for it. It might be worth reconsidering whether the relation between warrant and the supporting relation should be a common support relation, as we modelled it now, or whether the warrant should be a special part of the supporting relation itself, i.e. not drawn with another arrow.

Inductive and deductive reasoning Pollock (1987) mentions that non-deductive reasoning and deductive reasoning are two different but equally often occurring forms of reasoning and a reasonable epistemology must accommodate both. As an example, take the following two simple arguments, drawn in Figure 67:

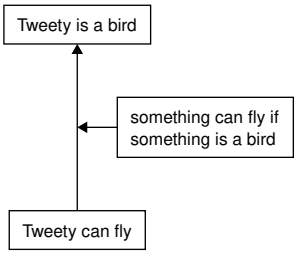
(34) Tweety can fly because he is a bird and birds can fly.

(35) Tweety is a bird because he can fly and birds can fly.



Tweety can fly because he is a bird and birds can fly.

(a) Deductive reasoning.



Tweety is a bird because he can fly and birds can fly.

(b) Inductive reasoning.

Figure 67: Two similar arguments showing different forms of reasoning.

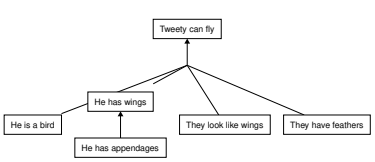
Both of these would be reasonable statements to utter. The first is deductive reasoning, we are just applying the logic stated in the general claim 'birds can fly'. The second is inductive reasoning, as we assume Tweety to be a bird as he has one of the properties of birds, i.e. being able to fly.

Both of these are interpreted in HASL in the way they are stated here, as shown in Figure 67. In the first, 'Tweety can fly' is the conclusion, in the second 'Tweety is a bird'. The warrant is the same for both, which is odd, since the reasoning in both arguments is different. Enthymeme resolution also proposes the warrant 'something is a bird if something can fly'. Our implementation at least is more focussed on inductive logic.

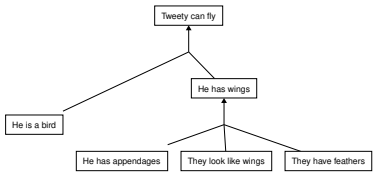
Since our model does not make the distinction between these two forms of reasoning, it results in different semantics for the argument diagram for the same type of relations. For example, in the second sentence we should interpret a claim 'planes can fly' as an undercutter, attacking the relation between datum and conclusion. I.e. if planes can also fly, then Tweety might just as likely be a plane instead of a bird. In the first sentence, such a claim would not make any sense.

7.2 An argument mining perspective

HASL as an argument mining system is an interesting comparison, as HASL is not an argument mining system. It does not try to parse real text, and only accepts text that is specifically written for its grammar. Text that contains segments that it cannot parse cause the whole input to be rejected. HASL/2 is more lenient, as the text between the keywords it searches for is accepted as is. Text that is ambiguous to its grammar, is parsed in all possible interpretations. This handling of ambiguity makes HASL not a good candidate for argument mining tasks, where ambiguity is not wanted and countered with additional disambiguation steps or writing parsing rules in such a way that ambiguity does not occur. However, we found that there is not always a preferred interpretation. For example, the preferred interpretation of example 70 (Figure 39) where 'Tweety is a penguin' is the attacking claim, depends on how you interpret this exception, either as undercutter ('birds can fly' is not a relevant claim as penguins are an exception to this rule) or a rebutter (because penguins cannot fly). The third option, Figure 39b, attacking 'Tweety is a bird',



Tweety can fly because he is a bird, he has wings because he has appendages, they look like wings, and they have feathers.



Tweety can fly because he is a bird **and** he has wings because he has appendages, they look like wings, and they have feathers.

Figure 68: The difference of a comma instead of 'and' in HASL/2.

can only be ruled out through the meaning of the words bird and penguin, and the common knowledge that penguins are a kind of bird, hence saying that Tweety is a penguin does not attack the claim that Tweety is a bird, instead it might serve as support for that claim.

A strict language In contrast to the goal of argument mining, the grammars used for HASL cause small differences in the text to have a large impact, as for example demonstrated in Figure 68. The simple difference of changing a comma into 'and' changes the interpretation of the argument. Humans would not mind the difference between a comma or 'and' in a sentence, and even if this would create ambiguity as the non-strict interpretation of discourse markers such as 'and' and comma's would create multiple ways to interpret the sentence, humans would be able to recover from this through trying to find the most sensible interpretation of the argument, for example by using common knowledge or, in this example, even the link created by the pronoun 'they'. Walton and Reed (2005) refer to this as the principle of 'Charity'.

Discourse markers Both HASL/1 and HASL/2 extracts the structure in the argument from the usage of discourse markers in the text. Their grammar uses these as points to connect the grammar rules to. In this way HASL uses very few discourse markers. Any additional markers require new grammar rules, and one of our goals with HASL was to create a minimal grammar to achieve the complete parsing of our argument model. The few discourse markers we use, i.e. 'because', 'but', 'if', 'when', 'and', 'or', 'unless', and punctuation symbols, are sufficient for our language. Knott (1996) gives a taxonomy of other discourse markers that serve similar roles. Many of these can be used as drop-in replacements for the discourse markers used in HASL's grammars, e.g. 'since', 'as', 'in that' can be used in addition to 'because'. Adding such variation to the grammar would for HASL/2 also imply that the number of possible formulations for arguments would greatly increase, i.e. it could use many different combinations of markers to say the same thing. For our purposes, the limited set of discourse markers used by HASL is sufficient.

The use of discourse markers to guide argument interpretation is common, and often a good starting point (Lawrence and Reed, 2015; Kang and Saint-Dizier, 2014). Lawrence and Reed (2015) state that when discourse markers are present, they are a good indicator, however, they are found with a low frequency. Often the argument structure is more im-

plicit in the text, and not highlighted by the keywords our grammar is built around.

Argument mining approaches rely therefore also on extra information, such as all sorts of linguistic properties of text to built a model that can identify relations in the absence of discourse markers, and common formats of arguments like Walton et al. (2008)'s argument schemes to try to fit arguments even when the identified relations are incomplete.

Parsing real texts HASL/1 parses the structure in claims as well as the argumentative structure in its input text. It identifies the subject and verb phrase of claims. This allows HASL/1 to differentiate between specific claims (i.e. claims about a specific entity) and general claims (i.e. more rule-like claims). This level of understanding allows it to do anaphora and enthymeme resolution.

The necessity of HASL/1 to be able to parse every word of its input comes at a high cost. Although we accept this for the language that connects claims to each other, it limits in the language we can use inside claims.

To make the language HASL is able to parse less limited, HASL/2 is only strict on the words and structure used to indicate the argument structure. The words between these, i.e. the text in the claims, is entirely free form. As a result it can parse complex text like the Dutch Tort law example 126, but also sentences in which the claims are replaced with single letters or random symbols. The grammar of HASL/2 focusses on the argument structure in text, and does not contain a large number of grammar rules that deals with real text, e.g. the parsing of noun or verb phrases. This level of understanding is sufficient to identify argument structures and to formulate argumentative text, but not to do anaphora or enthymeme resolution.

Compared to hand-written parsers While the goals of argument mining and HASL are different, the idea for us for HASL originates from argument mining, and there are a number of similarities both in the goals and in the approaches.

Mochales and Moens (2011) created a context-free grammar for argumentation structures in legal cases, where the terminals are discourse markers such as words that for example indicate that a conclusion or one or more premises will follow, references to articles (e.g. article numbers), the entity providing the argument (e.g. jury), and words or segments of text that do not fall into any of these categories. These categories are determined by trained classifiers. In this way they construct rules that describe conclusive sentences, their arguments, and eventually the argument structure (c.f. the parse tree) of a legal case. Using this method they achieve 60% accuracy on the ECHR corpus. HASL takes a similar approach to parsing its input, but uses the output of the parsing to further reduce the structure of the argument to that of our argument model: of claims and relations connected by relations.

Saint-Dizier (2012)'s TextCoop platform uses a set of rules for generating possible structures from the text it encounters. These rules are described as grammar rules with transformations, similar as we have done for HASL. Each rule constructs a representation by matching symbols and applying a transformation operation on the matched set, turning it into trees and predicates. Like in HASL, the matching of symbols can be defined by naming them, properties such as part-of-speech tags, by local

grammar rules, e.g. rules which can handle domain-specific or syntactic variation, or by taking advantage of the structure of the document it tries to parse, e.g. whether text occurs in a title section or a paragraph. However their grammar and parsing method is designed to yield only one parse. A preliminary evaluation of their approach showed that they are able to identify 88% of the conclusions and 91% of the supporting arguments in their corpus of 66 texts. Since HASL is not able to parse naturally occurring text, we are not able to perform such an analysis.

Compared to trained parsers Most argument mining approaches do not solely depend on discourse markers or certain grammatical constructions to identify claims and their roles in text, e.g. whether it functions as an example, conclusion, supporting argument, etc. A trained model that uses linguistic features describing the form of the words, and their context through the surrounding words, the position of the word, etc. is used to determine the role of words.

Large sets of annotated arguments are needed to train these models. These arguments are often annotated in a less detailed argument model than the one we use for HASL. For example, Mochales and Moens (2011) chose to define arguments as a set of at least two propositions with a relation between them, where the relations are as described by Walton et al.'s argument schemes, indicating e.g. an example or a commitment. The task of identifying argumentative text, segmenting those into separate clauses, and marking these as premise or conclusion is done using trained model.

Lawrence and Reed (2015) also use trained models to find segments of text that fit the slots in Walton's argument schemes. They then train their model to find which claim fits where in two of the schemes, Expert Opinion and Positive Consequences, and then let the best fitting scheme project the role of the claim (i.e. premise or conclusion) on it. In this approach, there is no need to write a specific grammar for argumentation. The structure of the argument schemes allows them to understand the structure of the argument. Lawrence and Reed are able to achieve an f-score of 0.83 for determining the types of connections between claims.

Such an approach is limited by the argumentation schemes that it has been trained on, as well as the connection to the language of a particular domain in which the model has been trained as this heavily influences the features it learns. With HASL we try to not be limited to a specific domain by not connecting our grammar to a limited set of argument schemes or discourse markers or other features of a specific domain.

The understanding of novel argument structures (i.e. arguments that do not fit a known argument scheme) is inherently challenging for trained approaches to argument mining. The argument models used in Argument mining have to be based on the argument models that are used for the annotations of the arguments in the corpora that are used for training. To illustrate, Stab and Gurevych (2017) introduces an annotation scheme for persuasive essays. The main difference with this scheme and common scheme of claims supporting and attacking each other is that their scheme features a distinction between claims and major claims, where the latter are the major topics of the essay that are supported by the non-major claims. This schema fits their domain better, but limits their approach to such a domain.

7.3 An argumentation tool perspective

The interface of HASL is designed as a tool in which you can write an argumentative text, and it will display its interpretation of it. This set-up was chosen because it focusses on presenting its understanding of the argument, both because it allows us to play with the language (you can try variants of the input sentence and compare the outcomes), as well as to introduce it as a tool for understanding dense argumentative language, where the understanding really depends on the correct interpretation of the text.

Diagrams The main way of communicating the interpretation of the argument it through diagrams. The diagrams presented in this paper are the same as used by HASL for presentation and as their internal structure of the argument.

The presentation of argumentative text side by side the interpretation is uncommon. As far as we know only ARAUCARIA (Reed and Rowe, 2004) lets users have the text and the argument diagram side by side, and allow them to construct the diagram by taking segments from the text. The commercial version of RATIONALE also allows users to have text next to their argument, but only for reference. Neither is able to parse the text automatically.

As tools ARAUCARIA and the modern version of RATIONALE are better developed. Both allow their users to store and retrieve arguments from a large shared database. RATIONALE also uses visual cues or keywords that indicate how to interpret the argument diagram.

Warrants ARAUCARIA allows relations between claims to be labelled with Walton's argument schemes. In HASL relations can be supported by general claims, or warrants. These form an alternative for such labels, allowing the relations to be 'labelled' by a supporting argument. Since these arguments are also present in the diagram they more prominently part of the argument. Our example of applied Dutch tort law (Figure 66) demonstrates this. Furthermore, since these general claims are itself part of the argument they can be supported (e.g. as a backing) or attacked like any other claim.

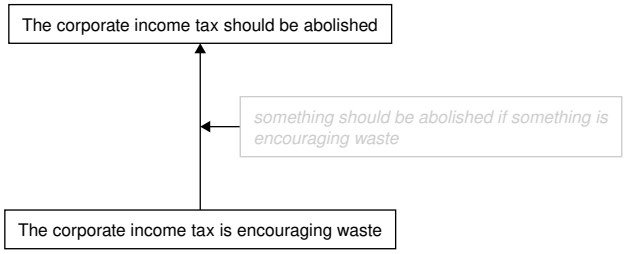
Rules In the case of HASL/2, the warrant can be a single claim, e.g. 'birds can fly', as well as a complex definition of tort law, like in Figure 66, including its conditions and exceptions. These can be mixed with an argumentation in which this law is applied, making the warrant not only serve as a completion of the argument, but also as a guide on how to argue for its applicability, or to discover flaws in the argument when exceptions are applicable, or a condition is not met.

Enthymeme resolution Because in HASL/1 the warrants are treated as the major premise in syllogisms, they can automatically be filled when missing. We have also implemented this of the minor premise and conclusion (i.e. the supporting and supported claim).

For enthymeme resolution to be possible HASL/1 has to interpret the warrant as a conditional rule, e.g. 'birds can fly' is interpreted as 'something can fly if it is a bird'. For general claims this works well, see section 4.10.

Walton and Reed (2005) argue that goal directed practical reasoning is the basis of many enthymemes. For example, arguments based on analogies often imply their conclusion or the claim that both types mentioned in the analogy are comparable. In HASL/1 the construction of missing claims does not apply any reasoning steps: It constructs the missing claim by combining the subject and the verb phrase of the two other claims that are present, as described in subsection 4.8. Arguably, for many arguments this is sufficient, e.g. an example Walton and Reed mentions is the following:

123 The corporate income tax should be abolished because it is encouraging waste.



The corporate income tax should be abolished because it is encouraging waste.

Figure 69: Example of enthymeme resolution mentioned by Walton and Reed (2005).

Figure 69 shows this being resolved as expected. However, they state that this argument might depend on two additional missing premisses, namely something along the line of 'waste is a bad practise', and 'a bad practise should be abolished'. Essentially, for such cases, one or more deductive reasoning steps are necessary, and in many cases additional knowledge (e.g. something being a bad practise) as well. This could be obtained from, for example, identifying which argumentation scheme is used in the argument.

In arguments where an extra deductive step is necessary, the results are unsatisfactory. For example, the sentence 'Tweety can fly because creatures with wings can fly and Tweety is a bird.' will not gain an assumed claim 'birds are creatures with wings'. Instead the claim 'Tweety is a creature with wings' will be constructed. The assumed claim is correct, but the argument is still an enthymeme, as the link between birds and creatures with wings is still not made explicit.

Walton and Reed (2005) discuss using forms of deductive or abductive inference and the use of argument schemes to identify and add the missing claims to the argument diagram. This may result in multiple competing interpretations, but this can be overcome by picking the interpretation that makes the argument strongest. The enthymeme resolution phase in HASL is limited, there are no argument schemes or common knowledge to reason with available to it. HASL interprets an argument only based on the linguistic features present in the text in which the argument occurs. This is one of HASL's design decisions, as our goal is to depict the

argument as it is presented, not as it should be. However, we do make assumptions on where to place a claim in a argument diagram, and we also fill in claims that we assume are necessary for the argument, but are not present in the text. As such, our interpretation of the argument needs to understand the argument and claims at a certain level. Currently this is only at a language level i.e. we can identify the subject of a claim, but if we would like to account for all inference steps in between (Walton and Reed, 2005; Mochales and Moens, 2011) we need a deeper understanding and inference.

While HASL/1 focusses more on the interpretation of arguments from argumentative text by identifying the general and specific claims and by filling in enthymemes, HASL/2 is able to formulate such texts.

Formulation of argumentative text The commercial version of RATIONALE has a feature that allows it to formulate an starting point for an essay based on the argument drawn by its users. It makes use of a template for the structure of essays, and fills this template with the arguments of the argument diagram that has been drawn by the user. These generated texts can then be used by for example students to write their own essay by elaborating more on the claims that they had already collected in the argument diagram.

HASL/2 is able to formulate argumentative text from the argument diagrams that are drawn or parsed in it. It does this by using the same grammar it uses to interpret these texts. This way, a dynamic way of constructing the arguments is possible, i.e. by first typing an argumentative text, letting HASL/2 interpret it, modify the argument diagram to show the argument as intended or expand on it, and finally letting HASL/2 formulate this modified diagram to text. The interface of HASL/2 (subsection 5.7) is set up for such a workflow.

For simple arguments the argumentative texts generated by HASL/2, shown in section 5.8, do not differ from the example sentences we have written for it. However, for more complex arguments the formulation process takes advantage of the unconstrained grammar, which allows arguments and sub-arguments to be collapsed into a single sentence. For HASL/2 to be a serious option to consider when trying to formulate argumentative texts, this process would need to improve, either by adding constraints to the grammar in regards to collapsed arguments similar to those in HASL/1, where only the last claim can be a collapsed argument. This would then need to be combined with a search process that finds a solution in which all claims and relations are formulated. This would require either some form of marking claims and relations as formulated or not while formulating the complete argument, or the formulation process would need to happen in steps, where each step would formulate as much as is allowed by the grammar, and pass all the claims and relations that have not yet been added to the text into the next iteration, which would formulate a new sentence. Reed (1997) explores this difference between the logical structure (claims and relations) and rhetorical structure (including typical argument structures, sub argument length, readability, convincingness) and approaches this as a planning task: planning the attention of the reader, optimising for the most convincing interpretation. His system, RHETORICA, creates such a plan.

Evaluating arguments One topic that is not touched upon by HASL is the evaluation of arguments. Its argument diagrams fit well with for ex-

ample Verheij (2003b)'s DefLog, which has a similar semantics in which claims can support and attack claims, and these relations can also be supported and attacked. DefLog contains semantics for the logical evaluation of arguments, and whether claims are accepted or rejected. The logic of DefLog is implemented ArguMed 3 (Verheij, 2003a), an argument assistance tool that allows its user to construct arguments (claims that support or attack relations, i.e. warrants and undercutters), and can then interactively evaluate which of these claims should be accepted or rejected based on how they support or attack each other. Since our argument model is similar to the argument model defined in DefLog and used in ArguMed 3, an extension on HASL that implements the evaluation semantics of DefLog on top of its argument diagrams would be feasible.

8 Conclusion

In this thesis we set out to explore what could be achieved with argument mining when language would not be an obstacle. We did this by first defining all the elements of our argument model, and then constructing a grammar which defines our limited language that can be used in argumentative text. We implemented this language in two experimental implementations, HASL/1 and HASL/2.

In HASL/1 we explored what a deeper understanding of arguments and its elements entails, by parsing every word in the argumentative text. This gave us the ability to add anaphora and enthymeme resolution.

In HASL/2 we opted to parse text using only the discourse markers we could detect, treating the claims as black boxes. While this would make anaphora and enthymeme resolution impossible, the simplification of the grammar allowed us to make the interpretation step reversible. HASL/2 is able to both interpret and formulate arguments using the same process.

Our argument model, a graph model, features besides the elements of the standard diagram, such as cooperative and independent support and attack relations, also the option to express support or attack of relations. These are used to model Toulmin's warrant and Pollock's undercutter. We make the distinction between specific and general claims, where the general claims can serve as generalized rules or warrants in our arguments. In HASL/2 we used an extended version of this model that also describes the structure inside general claims, which is able to express the conditions and exceptions for the conclusion of the generalized claim. This argument model works well for inductive reasoning, where it is used in enthymeme resolution, but for deductive logic the semantics do not work.

The language that we do allow for is described in a context free grammar. This grammar is constructed from all the elements of our argument model, which results in a set of grammar rules that can express each of the possible constructions in our argument model. However our grammar is ambiguous and it is not always possible to make the right choice on which argument structure is mend with a certain utterance. In this case we present all possible parses. Especially in the case of attack relations the grammar cannot differentiate between rebutter, undercutter and underminer. These relations can be expressed unambiguously by stating them on their own, however they cannot in the shorcuts we provide in our grammar, namely the collapsing of connected arguments into a single sentence. In such cases disambiguation would only be possible with access to common knowledge and a complete understanding of the content of the claims.

HASL is not an argument mining tool. The parser is hit or miss. If a text is parsed, it is often interpreted correctly. Even when the text is an enthymeme, and claims are missing, the interpretation can still be correct and complete due to enthymeme resolution. This is all done based on by discourse markers, which are the main guidance for our grammar. This makes HASL unfit for the parsing of natural text, as discourse markers are often left out in real argumentative text. HASL/2 performs slightly better upon this aspect compared to HASL/1 since HASL/2 treats the claims and the grammar inside them as a black box, but as a result the interpretation is even more reliant on the presence of the correct discourse markers.

HASL/2 focusses more on the structure of the argument than HASL/1, which also contains grammar rules to parse the text inside claims. Through

this HASL/1 is able to identify the structure in rule-like claims like 'birds can fly' and use this structure for its enthymeme resolution. While HASL/2 is not able to do this analysis, it supports an extended argument model in which these rule-like claims can be modelled as actual rules with conditions and exceptions. This extended model is not used for enthymeme resolution, and implementing the method used in HASL/1 in HASL/2 based would allow us to uncover many more enthymemes. However, the creation of these missing claims would require some form of interpretation of the text inside arguments as we would need to identify the subject and verb phrase. Instead of ignoring the structure inside claims, we could build the argument structure parsing upon a well developed parser for English.

As an argumentation tool HASL is promising. The argument model that can be viewed and edited inside HASL is more expressive than the diagramming methods often used in argumentation tools as it allows warrants to be expressed as part of the argument. By presenting all interpretations of ambiguous texts, HASL can be used to explore different interpretations of argumentative texts. By resolving enthymemes, claims that were intentionally or unintentionally unstated can be uncovered, helping with the understanding of the argument. Finally, HASL/2 is able to formulate an argumentative text from argument diagrams. This allows for a workflow where an argument is initially formulated by the user, parsed by HASL, the argument diagram is again corrected by the user, and the result is the formulated by HASL to a new argumentative text. This workflow may help people with both the understanding and formulation of arguments.

Appendices

A Appendix

A.1 Part-of-speech tags

Tag	Description	Example
NN	noun, singular	bird
NNS	noun, plural	birds
NNP	noun, proper singular	Socrates
NNPS	noun, proper plural	Swedes
PRP	pronoun	I, he
PRP$	possessive pronoun	his
MD	verb, modal auxiliary	can
VB	verb, infinitive	fly, walk
VBZ	verb, 3rd person singular present	is
VBP	verb, non-3rd person singular present	are
VBD	verb, past tense	became
VBN	verb, past particle	illuminated
VBG	verb, present participle	encouraging
DT	determiner	the, a, most
JJ	adjective	red
IN	preposition	in, on, by
RB	adverb	not
WDT	wh-determiner	that, which

Table 3: Part-of-speech tags used in HASL, part of the Penn Treebank tag set as used by spaCy (Honnibal and Montani, 2018).

A.2 HASL/1 Grammar rules

HASL/1's grammar consists for the most part of rules specifically for parsing English sentences

1. e.g. 'Tweety is a bird'
   ⟨*specific-claim*⟩ ::= ⟨*instance*⟩ ⟨*verb-phrase-sg*⟩
2. 'Socrates is not a man'
   ⟨*specific-claim*⟩ ::= ⟨*instance*⟩ ⟨*neg-verb-phrase-sg*⟩
3. 'they have wings' in 'birds can fly because they have wings'.
   ⟨*specific-claim*⟩ ::= ⟨*instance*⟩ ⟨*verb-phrase-pl*⟩
4. 'A man is mortal'
   ⟨*general-claim*⟩ ::= ⟨*prototype-sg*⟩ ⟨*verb-phrase-sg*⟩
5. 'A man is not mortal'
   ⟨*general-claim*⟩ ::= ⟨*prototype-sg*⟩ ⟨*neg-verb-phrase-sg*⟩
6. 'Birds can fly'
   ⟨*general-claim*⟩ ::= ⟨*prototype-pl*⟩ ⟨*verb-phrase-pl*⟩

7. 'Birds can not fly'
   ⟨*general-claim*⟩ ::= ⟨*prototype-pl*⟩ ⟨*neg-verb-phrase-pl*⟩

Effectively, specific claims begin with `instance`, general claims with `prototype`. All general and specific claim rules are supported by definitions for `instance`, and singular and plural versions of `prototype`. Furthermore, we need a definition for `verb-phrase`, both in singular and plural, and in negated form. These are defined in their own rules to make the general and specific claim rules more general.

Instances Instances are names or nouns with a definitive determiner, like 'Tweety' or 'the king', or 'I'.

⟨*instance*⟩ ::= ⟨*name*⟩
| ⟨*def-dt*⟩ [⟨*adjectives*⟩] ⟨*noun*⟩ [⟨*prep-phrase*⟩]
| ⟨*PRP$*⟩ ⟨*noun*⟩

Prototypes Prototypes are similar to instances, except that they start with an indefinite determiner (e.g. 'a bird') or have no determiner at all like 'birds'. We still allow a DT at the front to account for 'many birds' etc.

⟨*prototype-sg*⟩ ::= ⟨*indef-dt*⟩ [⟨*adjectives*⟩] ⟨*noun-sg*⟩ (⟨*vbn*⟩ | [⟨*prep-phrase*⟩])

⟨*prototype-pl*⟩ ::= [⟨*DT*⟩] [⟨*adjectives*⟩] ⟨*noun-pl*⟩ (⟨*vbn*⟩ | [⟨*prep-phrase*⟩])

We need a few more rules to help the construction of the previous, and to allow us to construct noun and verb phrases.

- Names like 'Tweety', 'Jean Claude', and 'Catholic Swedes'. Tweety is specifically defined since the POS tagger model we use sometimes tags it incorrectly.
  ⟨*name*⟩ ::= ⟨*NNP*⟩
  | ⟨*name*⟩ ⟨*NNP*⟩
  | ⟨*name*⟩ ⟨*NNPS*⟩
  | '`Tweety`'
- Nouns, both singular and plural
  ⟨*noun-sg*⟩ ::= [⟨*noun*⟩] ⟨*NN*⟩

  ⟨*noun-pl*⟩ ::= [⟨*noun*⟩] ⟨*NNS*⟩

  ⟨*noun*⟩ ::= ⟨*noun-sg*⟩ | ⟨*noun-pl*⟩
- Determiners, definite and indefinite
  ⟨*def-dt*⟩ ::= '`the`' | '`this`'

  ⟨*indef-dt*⟩ ::= '`a`' | '`an`'
- Adjectives, such as 'almost red' in 'the almost red apple'
  ⟨*adjectives*⟩ ::= ⟨*JJ*⟩ [⟨*adjectives*⟩]
- segments like 'born in America'
  ⟨*vbn*⟩ ::= ⟨*VBN*⟩ [⟨*prep-phrase*⟩]
- noun segments, ranging from 'Tweety' to 'the bald man born in America'.

⟨*noun-phrase*⟩ ::= ⟨*name*⟩
| ⟨*noun*⟩
| ⟨*instance*⟩
| ⟨*dt*⟩ [⟨*adjectives*⟩] (⟨*name*⟩ | ⟨*noun*⟩)
| ⟨*noun-phrase*⟩ ⟨*vbn*⟩
| ⟨*noun-phrase*⟩ ⟨*prep-phrase*⟩

- preposition phrases, 'in America', 'by a red light'
  ⟨*prep-phrase*⟩ ::= ⟨*IN*⟩ ⟨*noun-phrase*⟩
- segments that begin with 'that', i.e. in the noun phrase 'apples that appear red'. Two variants, to maintain the separation of singular and plural.
  ⟨*that-phrase-sg*⟩ ::= ⟨*WDT*⟩ ⟨*verb-phrase-sg*⟩

  ⟨*that-phrase-pl*⟩ ::= ⟨*WDT*⟩ ⟨*verb-phrase-pl*⟩
- Verbs such as is, has, appears, both for third person singular and plural, but also 'has eaten', 'have visited'.
  ⟨*verb-sg*⟩ ::= ⟨*VBZ*⟩ [⟨*VBN*⟩]

  ⟨*verb-pl*⟩ ::= ⟨*VBP*⟩ [⟨*VBN*⟩]
- Verbs such as 'became'
  ⟨*verb-sg*⟩ ::= ⟨*VBD*⟩

  ⟨*verb-pl*⟩ ::= ⟨*VBD*⟩
- combinations such as 'can fly' and 'should be abolished'
  ⟨*verb-sg*⟩ ::= ⟨*MD*⟩ ⟨*VB*⟩ [⟨*VBN*⟩]

  ⟨*verb-pl*⟩ ::= ⟨*MD*⟩ ⟨*VB*⟩ [⟨*VBN*⟩]
- Specifically for 'has become naturalized'
  ⟨*verb-sg*⟩ ::= ⟨*VBZ*⟩ ⟨*VBN*⟩ ⟨*VBN*⟩

  ⟨*verb-pl*⟩ ::= ⟨*VBP*⟩ ⟨*VBN*⟩ ⟨*VBN*⟩
- Negated forms, e.g. 'can not fly'
  ⟨*neg-verb-sg*⟩ ::= ⟨*MD*⟩ '`not`' ⟨*VB*⟩

  ⟨*neg-verb-pl*⟩ ::= ⟨*MD*⟩ '`not`' ⟨*VB*⟩
- Negated forms, e.g. 'has no', 'is not'
  ⟨*neg-verb-sg*⟩ ::= ⟨*VBZ*⟩ ('`no`' | '`not`')

  ⟨*neg-verb-pl*⟩ ::= ⟨*VBP*⟩ '`no`'
- Finally, these combined into phrases so they can have objects.
  ⟨*verb-phrase-sg*⟩ ::= ⟨*verb-sg*⟩ [⟨*noun-phrase*⟩] [⟨*prep-phrase*⟩]

  ⟨*verb-phrase-pl*⟩ ::= ⟨*verb-pl*⟩ [⟨*noun-phrase*⟩] [⟨*prep-phrase*⟩]
- Also for simpler verb phrases, e.g. 'appears red'
  ⟨*verb-phrase-sg*⟩ ::= ⟨*verb-sg*⟩ ⟨*adjectives*⟩ [⟨*prep-phrase*⟩]

  ⟨*verb-phrase-pl*⟩ ::= ⟨*verb-pl*⟩ ⟨*adjectives*⟩ [⟨*prep-phrase*⟩]
- Negated forms
  ⟨*neg-verb-phrase-sg*⟩ ::= ⟨*neg-verb-sg*⟩ [⟨*noun-phrase*⟩]

  ⟨*neg-verb-phrase-pl*⟩ ::= ⟨*neg-verb-pl*⟩ [⟨*noun-phrase*⟩]

A.3 Discourse makers

Token	Context
and	conjunctions
or	conjunctions
,	conjunctions
.	sentence
but	attack
except	attack, exception
that	attack
because	support
if	condition
when	condition
unless	exception

Table 4: Discourse markers used in HASL/2's grammar.

A.4 Evaluation examples

These are the examples used to evaluate HASL/1 and HASL/2. They are chosen to cover the requirements mentioned in the introduction. These examples are also available in the online versions of HASL.

Table 5 lists evaluation of HASL/1 and HASL/2 per example. For each example we noted the number of interpretations provided by HASL/1, and the number we judged to be correct. Any remarks on what went right or wrong are noted as well.

Solo claims

(36) Tweety can fly.

(37) A man born on a Saturday in Bermuda can fly to the ocean.

Negation

(38) Tweety cannot fly.

(39) Birds cannot fly.

(40) A bird cannot fly.

(41) No bird can fly.

Pros

(42) Socrates is mortal because Socrates is a man.

(43) Tweety can fly because Tweety is a bird.

(44) Birds can fly because birds have wings.

(45) Birds can fly because Tweety can fly.

Cons

(46) Socrates is mortal but Socrates is a god.

(47) Tweety is a bird but Tweety is a cat.

(48) Tweety can fly but Tweety is a penguin.

Mutual attack

(49) Tweety can fly but Tweety cannot fly. Tweety cannot fly but Tweety can fly.

(50) Tweety can fly but Tweety cannot fly.

Pros combinations

Explicitly stating which claim is the supported claim through multiple sentences.

(51) Tweety can fly because Tweety is a bird. Tweety is a bird because Tweety has wings.

(52) Tweety can fly because Tweety is a bird. Tweety can fly because Tweety has wings.

Combining supporting claims in independent, cooperative and chained support.

(53) Tweety can fly because Tweety is a bird and because Tweety has wings.

(54) Tweety can fly because Tweety is a bird and Tweety has wings.

(55) Tweety can fly because Tweety is a bird because Tweety has wings.

Nesting multiple supportive claims and relations in nested sentences.

(56) Tweety can fly because he has wings, he has feathers and he is capable.

(57) Tweety can fly because he has wings and he has feathers and because he is a bird.

(58) Tweety can fly because he is a bird and because he has wings and he has feathers.

(59) Tweety can fly because Tweety is a bird because Tweety is a duck because Tweety has a bill.

(60) Tweety can fly because Tweety is a bird because Tweety is a duck and because Tweety has wings.

(61) Tweety can fly because Tweety is a bird and Tweety has wings because Tweety is a duck.

(62) Tweety can fly because Tweety is a bird and because Tweety is a duck because Tweety has a bill.

Cons combinations

Attacking attacked claims using multiple sentences which state the explicit target, and in sentence 64 in a single sentence.

(63) Tweety can fly but Tweety is a cat. Tweety is a cat but Tweety has wings. Tweety has wings but the wings are small.

(64) Tweety can fly but Tweety is a cat but Tweety has wings but the wings are small.

(65) Birds can fly but Tweety can not fly because Tweety is a penguin.

Pros and cons combinations

(66) Tweety can fly because Tweety is a bird. Tweety is a bird but Tweety is a cat.

(67) Tweety can fly because Tweety is a bird but Tweety is a cat.

(68) Tweety can fly because Tweety is a bird. Tweety is a bird but Tweety is a penguin.

(69) Tweety can fly because Tweety is a bird. Tweety can fly but Tweety is a penguin.

(70) Tweety can fly because Tweety is a bird but Tweety is a penguin.

(71) Tweety can fly because Tweety is a bird but he can not fly.

(72) Tweety can fly because he is a bird and because he has wings.

(73) Tweety can fly because he has wings but he is not a bird.

(74) Tweety can fly because he is a bird and he has wings.

(75) Harry is a British subject because Harry is a man born in Bermuda but Harry has become naturalized.

Undercutters

(76) The object is red because the object appears red to me but it is illuminated by a red light.

(77) Jan is a prisoner but Jan is not a thief.

(78) Jan is a prisoner because thieves are prisoners but Jan is not a thief.

Warranted support

(79) Tweety can fly because he is a bird and birds can fly.

(80) Tweety can fly because birds can fly and he is a bird.

(81) Tweety can fly because birds can fly, he is a bird, he has wings and he has feathers.

An argument with a clearly incorrect warrant.

(82) Tweety can fly because he is a bird and thieves are punishable.

An argument with two separate claims supporting the conclusion that 'Tweety can fly' independently that are both warranted.

(83) Tweety can fly because birds can fly and he is a bird and because superheroes can fly and he is a superhero.

Then there are categorial syllogisms, where the minor premise is not a specific instance, but a subset of the major premise.

(84) Birds can fly because birds have wings and creatures with wings can fly.

Warranted attack

(85) Tweety can fly but he is a penguin and penguins can not fly.

(86) Jack is a thief because he is a liar and liars are thieves.

Warrants combinations

(87) Tweety can fly because he is a bird but not all birds can fly.

(88) Tweety can fly because he is a bird but birds cannot fly.

(89) Tweety can fly because she is a bird and birds can fly because they have wings.

Rule-like sentences

Sentences that can be interpreted as rules, i.e. in the form of 'something can fly if it is a bird'.

(90) A bird can fly.

(91) Birds can fly.

(92) Birds cannot fly.

(93) All birds can fly.

(94) Liars are thieves.

Rule-like sentences with exceptions

(95) Birds can fly except if they are penguins.

(96) Birds can fly except penguins.

(97) Birds except penguins can fly.

Rules

(98) If something is a bird it can fly.

(99) Something can fly if it is a bird.

(100) Something can fly if it is a bird and it has wings.

(101) Something can fly if it is a bird or if it has wings.

(102) Something can fly when it has wings and it has an engine or when it is a bird.

The same sentences can also be instantiated, i.e. 'something' is replaced with 'Tweety'. These are still rules, not arguments, as they do not argue that Tweety can fly.

(103) Tweety can fly if Tweety is a bird.

Rules with exceptions

(104) Something can fly if it has wings except when its wings are too short, when they are clipped or when they are broken.

(105) Something can fly if it has wings except when its wings are too short or when they are clipped or when they are broken.

(106) Something can fly when it has wings or when it is a bird unless it is a penguin or its wings have been clipped.

Arguments with rules

Applied rules are rules that are used as warrants to support a claim.

(107) Tweety can fly because if something is a bird it can fly.

(108) Socrates is mortal because he is a man and something is mortal if something is a man.

(109) Socrates is mortal because he is a man and something is mortal if something is a man or something is a mammal.

(110) Socrates is mortal because he is a man and something is mortal if something is a man or if something is a mammal.

(111) Socrates is mortal because he is a man and something is mortal if something is a man or if something is a mammal except when this person is Lazarus.

(112) Socrates is mortal because he is a man and something is mortal if something is a man except when this person is Lazarus or if something is a mammal.

Anaphora

(113) Tweety can fly because he has wings.

(114) The bird can fly because he has wings.

(115) The bird can fly because he has wings and they help him.

Changing the order of the pronoun and noun, and the difference between cooperative and independent relations.

(116) The queen can rule because she is born in a royal family and because the king abdicated.

(117) The queen can rule because the king abdicated and because she is born in a royal family.

(118) The queen can rule because the king abdicated and she is born in a royal family.

Enthymemes

The following three sentences all have one of the three parts of a syllogism missing.

(119) Socrates is mortal because men are mortal.

(120) Socrates is mortal because he is a man.

(121) Socrates is a man and men are mortal.

Enthymemes occur in combinations of claims.

(122) Tweety can fly because birds can fly but Tweety is a penguin.

Example of an enthymeme from Walton and Reed (2005).

(123) The corporate income tax should be abolished because it is encouraging waste.

In sentence 124 the attacking claim is a rebuttal on 'Tweety can fly', but it is not explicitly stated.

(124) Tweety can fly but Tweety is a penguin and penguins can not fly.

Tort

(125) John has a duty to repair the damages because Jack has suffered damages, those damages are caused by an act of John, the act was unlawful and the act can be imputed to John. The act can be imputed to John because this article of law describes it as such and because common opinion describes it as such.

(126) A person must repair the damages if he committed an act against another person, the act is tortious, the act can be attributed to him and the other person has suffered the damage as a result thereof.
The act is tortious if there was a violation of someone else's right, if an act or omission is in violation of a duty imposed by law or if an act or omission is in violation of what according to unwritten law has to be regarded as proper social conduct unless there was a justification for this behaviour.
The act can be attributed to him if it results from his fault or if it results from a cause for which he is accountable by virtue of law or generally accepted principles.

A.5 Evaluation results

Table 5 shows the results when we enter the sentences from subsection A.4 into HASL/1 and HASL/2. The first column refers back to which sentence it is. Each number uniquely identifies the example sentence throughout this whole document, both in the explaining sections as well as here in the evaluation. The *Interpretation* columns mention how many interpretations that version of HASL yields, and the *Correct* column lists how many of these were correct. If there are no interpretations due to an error, we use a dash in the *Interpretations* column. If there cannot be a correct interpretation due to a design decision, we use a dash in the *Correct* column. The *Remarks* column mentions the reasoning for such errors.

Sentence	HASL/1 Interpretations	HASL/1 Correct	HASL/2 Interpretations	HASL/2 Correct	Remarks
Solo claims					
36	1	1	1	1	
37	1	1	1	1	HASL/1: Interpreted as 'something can fly to the ocean if something is A man born on a Saturday in Bermuda'.
Negation					
38	1	1	1	1	HASL/2 can not mark these claims as negated.
39	1	1	1	1	
40	1	1	1	1	
41	-	-	1	1	HASL/1: 'No bird' cannot be interpreted.
Pros					
42	1	1	1	1	
43	1	1	1	1	
44	0	-	1	1	HASL/1: General claim supported by general claim which cannot be expressed in HASL/1.
45	1	1	1	1	
Cons					
46	1	1	1	1	
47	1	1	1	1	
48	1	1	1	1	
Mutual attack					
49	1	1	1	1	
50	1	0	1	0	Not automatically marked as mutual attack.
Pros combinations					
51	1	1	1	1	
52	1	1	1	1	
53	1	1	1	1	

54	1	1	2	1	HASL/2: Tweety has wings is both interpreted as a warrant and as a secondary cooperative claim.
55	1	1	1	1	
56	1	1	1	1	
57	1	1	2	1	HASL/2: 'He has feathers' is interpreted both as a warrant and as secondary cooperative claim.
58	1	1	2	1	HASL/2: Idem.
59	1	1	1	1	
60	1	1	2	2	HASL/1: No general claims are constructed/assumed for the part 'Tweety is a bird because Tweety is a duck and because Tweety has wings'. HASL/2: 'Tweety has wings' is connected to either 'Tweety is a bird' or 'Tweety can fly'.
61	1	0	1	0	Interpreted as a single argument concluding that Tweety can fly, not as two separate sentences.
62	-	-	1	1	HASL/1: Fails parsing on 'has' in 'Tweety has a bill'.
Cons combinations					
63	1	1	1	1	
64	5	1	1	1	HASL/1: The attacking claims attack each of the previously mentioned claims.
65	1	1	1	1	
Pros and cons combinations (incl. undercutters)					
66	1	1	1	1	
67	3	1	3	1	'Tweety is a cat' attacks both other claims as well as the relation between them (i.e. undercutter).
68	1	1	1	1	
69	1	1	1	1	
70	3	2	3	2	'Tweety is a penguin' is interpreted as a rebutter, an undercutter, and an underminer. Two of these are defensible interpretations, only Tweety is a penguin attacking Tweety is a bird is not.
71	3	1	3	1	'he is a penguin' (in which 'he' is replaced with 'Tweety') is interpreted as a rebutter, an undercutter, and an underminer. The rebutter interpretation is the most reasonable.
75	3	2	3	2	'Harry has become naturalized' is interpreted as rebutter, undercutter and underminer. The rebutter and undercutter interpretations are defensible.
Undercutters					
76	3	1	3	1	'The object is illuminated by a red light is as rebutter, undercutter and underminer. The undercutter representation is the only correct interpretation.'
77	1	1	1	1	

78	2	0	3	1	HASL/1: In one interpretation, the missing minor premise 'Jan is a thief' is added, but it is not attacked by the claim that 'Jan is not a thief'. In the other interpretation 'Jan is not a thief' is interpreted as an argument against 'Thieves are prisoners'.
Warranted support					
79	2	1	2	1	HASL/1: One interpretation constructs the general claim 'something can fly if something can fly'. We assume this is an implementation error. HASL/2: 'Birds can fly' is both interpreted as a warrant and as a secondary cooperative supporting claim.
80	1	1	2	1	HASL/2: 'He is a bird' is also interpreted as warrant.
81	-	-	1	1	HASL/1: Interpretation fails because the general claim and specific claims are in a single list. The next sentence is the 'correct' spelling.
82	2	2	2	1	HASL/1: One interpretation is the argument as presented, 'thieves are punishable' is warranting. We assume this is correct as this is what is stated. The second interpretation adds the assumption that Tweety is a thief and the intermediate conclusion that Tweety is punishable. From this it also assumes that something can fly if something is punishable. HASL/2: 'Thieves are punishable' is also interpreted as a secondary cooperative supporting claim.
83	-	-	4	1	HASL/1: Cannot be parsed as the grammar rule for multiple support only accepts `specific-claims`. HASL/2: The general claims are also interpreted as secondary cooperative supporting claims.
84	-	-	2	1	HASL/1: The grammar expects the minor premise/datum and conclusion to be a specific claim. HASL/2: The warrant cannot be identified as such, and is in one interpretation interpreted as a secondary cooperative supporting claim.
Warranted attack					
85	1	1	1	0	HASL/1: The claim 'Tweety cannot fly' is assumed as intermediate conclusion, and this claim is finally attacking 'Tweety can fly'. HASL/2: 'Penguins cannot fly' is only interpreted as secondary cooperative attacking claim.
86	2	1	2	1	HASL/1: In one interpretation 'thieves' is not correctly converted to the singular 'thief', presumably because it is interpreted like 'red' would be, causing the claim 'Jack is thieves' to be assumed, and the general claim 'something is a thief if something is thieves'. HASL/2: 'Liars are thieves' is also interpreted as secondary cooperative supporting claim.

Warrants combinations					
87	-	-	3	1	HASL/1: The grammar cannot parse 'not all birds'. HASL/2: 'Not all birds can fly' is also interpreted as rebutter and underminer.
88	-	-	3	1	HASL/1: The grammar does not expect a general claim as attacking claim. HASL/2: Same as 87.
89	1	1	1	1	'They have wings' is interpreted as a backing of 'birds can fly'.
Rule-like sentences					
90	1	1	1	1	HASL/2: These are not interpreted in a special way.
91	1	1	1	1	
92	1	1	1	1	
93	1	1	1	1	
94	1	0	1	1	HASL/1: Interpreted as 'Something is thieves if something is a liar'.
Rule-like sentences with exceptions					
95	-	-	-	-	HASL/1: The grammar cannot parse 'if'. HASL/2: The grammar only expects exceptions when there are explicit conditions.
96	1	0	1	0	HASL/1: The whole sentence is interpreted as a single claim. HASL/2: 'When they are broken' is interpreted as a condition for the conclusion. This is the correct parsing according to the grammar, but not the indented meaning of the sentence.
97	1	0	1	0	HASL/1: Idem. HASL/2: 'Birds' is interpreted as a single claim, attacked by 'Penguins can fly'.
Rules					
98	-	-	1	0	HASL/1: The grammar cannot parse 'if' or 'when'. HASL/2: Parsed as a single claim.
99	-	-	1	1	
100	-	-	1	1	
101	-	-	1	1	
102	-	-	1	1	
103	-	-	1	1	
Rules with exceptions					
104	-	-	2	1	HASL/1: The grammar cannot parse 'if' or 'when'. HASL/2: 'They are clipped' and 'They are broken' are either interpreted incorrectly as conditions, or as correctly exceptions to when 'it has wings'.
105	-	-	1	0	HASL/2: 'They are broken' is interpreted as a condition. The parse is grammatically correct, but not the intended interpretation.
106	-	-	2	1	HASL/2: The 'or' in 'it is a penguin or its wings have been clipped' is not in all cases interpreted as a boundary between two exceptions.

Argument with rules					
107	-	-	-	-	HASL/1: The grammar cannot parse 'if' or 'when'. HASL/2: The grammar cannot parse 'if something, it..'.
108	-	-	1	1	
109	-	-	1	0	HASL/2: 'Something is a man or something is a mammal' is not split into two conditions.
110	-	-	1	1	HASL/2: Correct interpretation of 109.
111	-	-	1	0	HASL/2: 'This person is Lazarus' is interpreted as an exception to 'Something is a mammal'.
111	-	-	2	1	HASL/2: 'Something is a mammal' is also interpreted as an exception.
112	-	-	2	1	HASL/2: 'Something is a mammal' is also interpreted as an exception.
Anaphora					
113	1	1	1	0	HASL/2 does not perform enthymeme resolution.
114	1	1	1	0	
115	1	0	1	0	HASL/1: 'they' is spotted as entity, but 'the wings' is not.
116	1	1	1	0	
117	1	1	1	0	
118	1	0	1	0	HASL/1: There is no method to identify the gender of words, and 'she' is interpreted as referring to 'the king'.
Enthymemes					
119	1	1	1	0	HASL/2 does not perform enthymeme resolution.
120	1	1	1	0	
121	1	1	1	0	
122	2	0	3	0	HASL/1: 'Tweety is a bird' is assumed and 'Tweety is a penguin' attacks either the claim 'birds can fly' or 'Tweety can fly', while it should undercut the relation between 'Tweety is a bird' and 'Tweety can fly'. HASL/2: 'Tweety is a penguin' undercuts, but 'birds can fly' is interpreted as directly supporting the claim 'Tweety can fly' instead of warranting such a support relation.
123	1	1	1	0	HASL/1: The general claim 'something should be abolished if something is encouraging waste' is assumed.
124	1	1	1	0	HASL/1: The intermediate conclusion that Tweety cannot fly is assumed.
Referential					
125	-	-	1	0	HASL/1: The grammar cannot parse this example. It fails on the first 'and', presumably due to one of the previous claims not being parsed correctly.
126	-	-	1	0	

Table 5: Evaluation of the evaluation sentences in HASL/1 and HASL/2.

References

P. Besnard and A. Hunter. Constructing argument graphs with deductive arguments: a tutorial. *Argument & Computation*, 5(1):5–30, 2014.

G. Betlem. *Civil liability for transfrontier pollution: Dutch environmental tort law in international cases in the light of Community law*. Graham & Trotman, 1993.

R. Cohen. Analyzing the structure of argumentative discourse. *Computational Linguistics*, 13(1-2):11–24, 1987.

P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

J.B. Freeman. *Dialectics and the Macrostructure of Arguments: A Theory of Argument Structure*. Foris Publications, Berlin/New York, 1991.

G. Grewendorf. Argumentation in der Sprachwissenschaft. *Zeitschrift für Literaturwissenschaft und Linguistik*, 10(38):129–151, 1980.

H. Herneault, H. Prendinger, D. duVerle, and M. Ishizuka. HILDA: A Discourse Parser Using Support Vector Machine Classification. *Dialogue & Discourse*, 1(3):1–33, 2010.

J.R. Hobbs. Resolving pronoun references. *Lingua*, 44(4):311–338, 1978.

M. Honnibal and I. Montani. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 2018.

J. Kang and P. Saint-Dizier. A Discourse Grammar for Processing Arguments in Context. In *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA 2014)*, pages 43–50, 2014.

A. Knott. *A Data-Driven Methodology for Motivating a Set of Coherence Relations*. PhD thesis, Edinburgh, 1996.

A. Knott and R. Dale. Using linguistic phenomena to motivate a set of coherence relations. *Discourse Processes*, 18(1):35–62, 1994.

J. Lawrence and C. Reed. Combining Argument Mining Techniques. In *Working Notes of the 2nd Argumentation Mining Workshop*, Denver, Colorado, US, 2015.

J. Lawrence, C. Reed, F. Bex, and M. Snaith. AIFdb: Infrastructure for the Argument Web. In *Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012)*, Vienna, 2012. IOS Press.

M. Lippi and P. Torroni. Argumentation mining: State of the art and emerging trends. *ACM Transactions on Internet Technology*, 16(2): 10:1–10:25, 2016.

W.C. Mann and S.A. Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse*, 8(3):243–281, 1988.

D. Marcu. The rhetorical parsing of natural language texts. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, ACL, pages 96–103, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics.

R. Mochales and M.-F. Moens. Argumentation mining. *Artificial Intelligence and Law*, 19(1):1–22, 2011.

M.-F. Moens. Argumentation mining: How can a machine acquire common sense and world knowledge? *Argument & Computation*, 9(1):1–14, 2018.

A. Peldszus and M. Stede. From Argument Diagrams to Argumentation Mining in Texts: A Survey. *International Journal of Cognitive Informatics and Natural Intelligence*, 7(1):1–31, 2013.

J.L. Pollock. Defeasible reasoning. *Cognitive Science*, 11(4):481–518, 1987.

C. Reed. Generating Arguments in Natural Language. 1997.

C. Reed and G. Rowe. Araucaria - Software for Argument Analysis, Diagramming and Representation. *International Journal on Artificial Intelligence Tools*, 13(4):961–979, 2004.

R. Reiter. A Logic for Default Reasoning. *Artificial intelligence*, 13(1-2): 81–132, 1980.

D. Reitter. Simple signals for complex rhetorics: On rhetorical analysis with rich-feature support vector models. *LDV-Forum, GLDV-Journal for Computational Linguistics and Language Technology*, (1/2):38–52, 2003.

P. Saint-Dizier. Processing natural language arguments with the <TextCoop> platform. *Argument & Computation*, 3(1):49–82, 2012.

C. Stab and I. Gurevych. Parsing Argumentation Structures in Persuasive Essays. *Computational Linguistics*, (43):619–659, 2017.

S. N. Thomas. *Practical reasoning in natural language*. Prentice Hall, Englewood Cliffs, NJ, 3 edition, 1981.

S.E. Toulmin. *The Uses of Argument*. Updated Edition. Cambridge University Press, Cambridge, England, 2 edition, 2003.

F. H. van Eemeren, B. Garssen, E. C. W. Krabbe, A. F. Snoeck Henkemans, B. Verheij, and J. H. M. Wagemans. *Handbook of Argumentation Theory*. Springer, Berlin, 2014.

T van Gelder. The rationale for Rationale™. *Law, Probability & Risk*, 6 (1-4):23–42, 2007.

B. Verheij. Artificial argument assistants for defeasible argumentation. *Artificial Intelligence*, 150(1-2):291–324, 2003a.

B. Verheij. DefLog: on the Logical Interpretation of Prima Facie Justified Assumptions. *Journal for Logic and Computation*, 13(3):319–346, 2003b.

B. Verheij. Evaluating Arguments Based on Toulmin's Scheme. *Argumentation*, 19(3):347–371, 2005.

B. Verheij. Formalizing arguments, rules and cases. In *Proceedings of the 16th International Conference on Artificial Intelligence and Law (ICAIL 2017)*, pages 199–208, New York, 2017. ACM Press.

G.A.W. Vreeswijk. Abstract argumentation systems. *Artificial intelligence*, 90(1-2):225–279, 1997.

D. Walton and C. Reed. Argumentation Schemes and Enthymemes. *Synthese*, 145(3):339–370, 2005.

D. Walton, C. Reed, and F. Macagno. *Argumentation schemes*. Cambridge University Press, Cambridge, GB, 2008.

J.H. Wigmore. *The Principles of Judicial Proof: As Given by Logic, Psychology, and General Experience, and Illustrated in Judicial Trials*. Number v. 1 in The Principles of Judicial Proof: As Given by Logic, Psychology, and General Experience, and Illustrated in Judicial Trials. Little, Brown, 1913.

J.H. Wigmore. *A treatise on the Anglo-American system of evidence in trials at common law*. Little, Brown and Company, 1923.