Bachelor's thesis

The influence of mental representations on human relational reasoning

Analyzing data structures as cognitive models

Phaina Koncebovski

Advisor: PD Dr. Marco Ragni
Lehrstuhl Grundlagen der künstlichen Intelligenz
Albert-Ludwigs-Universität Freiburg
Germany
Submitted August 18, 2016

DECLARATION

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work. I hereby also declare, that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date Signature

Abstract

Zusammenfassung

Contents

List of Terms

conclusion is a statement. In this thesis, conclusions are of the form "A left B" and denote that token A is left of B. Conclusions are checked for truth, i.e. in a model that has previously been built from premises, the validity of a conclusion is checked and returned. 5, 6, 10, 15, 18, 19

mental model is a model representation of e.g. spatial information that is constructed by a person - inside their mind, therefore a mental model. 5, 6, 11

premise is a statement. In this thesis, premises are of the form "A left B" and denote that token A is situated to the left of token B. 5–15, 18–20, 22, 24–27, 29, 30, 38, 39, 41

token is an object that a premise or conclusion refers to. E.g. in the premise "A left B", A and B are tokens. 6, 8, 11, 12, 14, 15, 18–25, 27–30, 38–40

working memory is the part of human memory between sensory information and long term memory. Information that is to be included in long term memory is selected here, so the working memory acts as a sort of buffer. According to Baddeley (2007) it is divided into three subcomponents, one of which is the visuospatial sketchpad, which is responsible for storing visual and spatial information. Working memory has limited capacity, Miller (1956)'s law states that this capacity is 7 $\pm$ 2, i.e. any individual can keep about 7 objects concurrently in their working memory. 5, 14, 18, 40

1 Introduction (or The Cave of Mysterious Monsters)

Imagine entering a mysterious cave. There are four doors, each with a combative animal behind it. You must choose a door to advance. Luckily, there's a clue:

- The polar bear is left of the python.
- The alligator is right of the python.
- The cuddly kitten is left of the python.

Figure 1: Mysterious doors

Obviously, fighting against pythons, alligators and polar bears is difficult (for different reasons). You may want to choose the door with the kitten behind it. Which one is that?

If you only had little time to decide and no particular love for logic and statistics, it's likely that you'd choose the leftmost door.
In fact, most people would do that. [1]

This thesis is not (only) about human irrationality, but about spatial or *relational reasoning*. Humans are born into and move through space all their life. The question of how objects are positioned in that space plays a crucial role in the ability of humans to stay alive and not constantly bump into walls.

[1] I guess. All people I personally asked were either confused by the contrived scenario or, perhaps due to their being computer science students, immediately convinced that this was Monty Hall and they had to "switch to the other door".

Giving directions ("Turn left at the intersection immediately after the bank."), information about where objects are located ("The box is on the table beneath the window in the room to your left.") or remembering positions ("The quote was in the blue book on a right page somewhere at the end of the book.") all involve relational reasoning: The ability to understand relative information as given in these sentences, interpret it correctly and - ideally - finding the object in question.
According to Baddeley (2007)'s model of working memory, relational reasoning has its own dedicated space in human working memory: the so-called *visuospatial sketchpad*. It holds visual and spatial information and allows mental manipulation thereof.

So, relational reasoning is:

- taking in information about space and objects in that space,
- taking in information about relationships between those objects and
- drawing conclusions from this information.

2 Model theory and Cognitive Basics

Since people are able to give relations such as "The python is left of the alligator", when given the information from the introduction, they are obviously able to derive relations that were not explicitly given in the premises (i.e. deduce information). One possible way to build inferences could be by logical rules, e.g. for the transitivity [2] of relations. If *A is left of B* and *B is left of C*, then *A is left of C*, by a simple logical rule.
However, this thesis is based on another way to understand human reasoning: *The general theory of mental models* by Johnson-Laird (1983). As summarized by Jahn, Knauff, and Johnson-Laird (2007):

> The theory of mental models, or the model theory for short, postulates that individuals use the meaning of assertions and general knowledge to construct models of the possibilities compatible with assertions.

So what is stored in working memory is not the information itself (e.g. as verbal representations of the premises), but a *constructed mental model* that represents the given information.

Goodwin and Johnson-Laird (2005) extend the general theory of mental models to describe the domain of relational reasoning by introducing the *model theory of relational reasoning*[3]. They state that

- models used for relational reasoning are structurally iconic, i.e. the relations holding between the represented entities are present in the modeled entities,

[2] A relation is transitive "if whenever an element a is related to an element b, and b is in turn related to an element c, then a is also related to c." (Wikipedia, 2016)

[3] This model theory is not to be confused with this model theory: `https://en.wikipedia.org/wiki/Model_theory`

- relational consequences are derived both from given relational information as well as general knowledge,
- individuals construct only one, typical model and
- "the difficulty of an inference depends on the process of integration of the information from separate premises, the number of entities that have to be integrated to form a model, and the depth of the relation".

There is considerable evidence for reasoners using the model theory for relational reasoning problems (Schaeken et al., 2007).

In the case of this thesis spatial information is given in the form of premises. One example for a set of premises can be found in the introduction (see section 1). Individuals construct models out of the premises. They use these models to draw conclusions and determine relationships between tokens mentioned in the premises.

Johnson-Laird and Byrne (1991) separate the relational reasoning process into three phases:

- The *comprehension phase*, named the *model construction phase* by Ragni and Knauff (2013) in which reasoners construct a model from given information and forget the premises,
- the *description phase*, named the *model inspection phase* by Ragni and Knauff (2013), in which reasoners inspect the mental model to derive relations that also hold but were not explicitly given in the premises and
- the *validation phase*, named the *model variation phase* by Ragni and Knauff (2013), in which reasoners try to find alternative models in which a given conclusion doesn't hold.

Even though the mental models are constructed individually, within one person, they seem to be quite predictably constructed and both stable

- within a population, i.e. most people construct the same internal mental model (Ragni and Knauff, 2013), (Knauff et al., 2013), (Rauh et al., 2005)
- as well as within an individual, i.e. one person constructs one mental model. (Goodwin and Johnson-Laird, 2005)

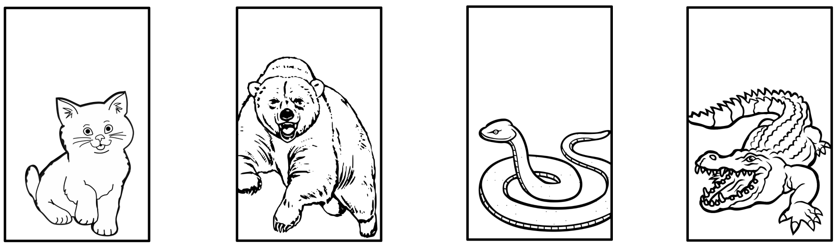
2.1 Inter- and Intrapersonal Stability of Mental Models

To explain these stabilities, let's turn to the thesis advisor's work. One key idea of Ragni and Knauff (2013) is the *preferred model theory*.

> A crucial assumption within our preferred model theory is that, in most situations, people only construct a single, simplified, and typical mental model and ignore all others. People are almost blind to alternative interpretations of the premises. We might only construct further models if the reasoning problem clearly requires us to consider alternatives. In other words, we do not think that the construction of an initial model is a stochastic process that produces one model this time and another the next time. In the preferred model theory, the construction of the initial model is, in principle, a deterministic process that always produces the same model for the same premises. We assume that this preferred model is the same for most people and that such preferred models bias people in a predictable way [...].
> (Ragni and Knauff, 2013)

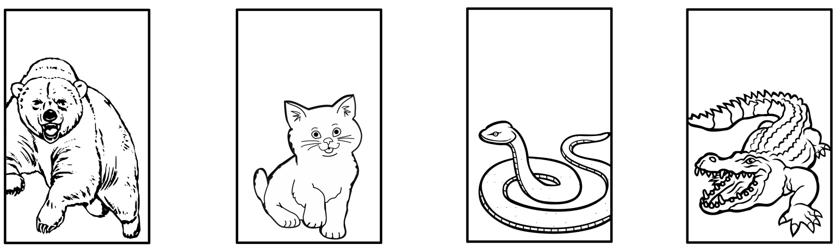
Given the example from the introduction (see section 1 for the premises) it is both possible that the kitten is to the far left and the polar bear one door further to the right,

Figure 2: Kitten, Polar bear, Python, Alligator



as well as them switching: The polar bear to the far left and the kitten one door further to the right.

Figure 3: Polar bear, Kitten, Python, Alligator



When receiving new information as to what else is to the left of the python, besides the polar bear, there's a choice of "where to put that information". One possibility is to leave the previously constructed model of "Polar bear, Python" intact and to add a kitten to the left of that. The other possibility is to move the polar bear further to the left so that there's space for the kitten to crawl in between polar bear and python. However, "the preferred model is favored over others because it is easier to construct in spatial working memory" (Ragni and Knauff, 2013). Moving items seems to be more work, since a moving operation requires the polar bear to be held in an extra slot, then adding the kitten in and moving the polar bear back. Ragni and Knauff (2013) argue that "preferred mental models of spatial descriptions are those constructed according to the principle that new objects are added to a model without disturbing the arrangement of those tokens already represented in the model." That is a variation to the description of the model construction phase by Johnson-Laird and Byrne (1991) - since only one model is constructed, the *preferred mental model*.

As (thereby) predicted, in an experiment by Ragni and Knauff (2013) 78% of tested individuals chose a model that was equivalently constructed to the first model for these premises, as shown in figure 2.

This is a quite remarkable interpersonal stability, also shown by Knauff et al. (2013) and Rauh et al. (2005).

A preferred mental model also seems to be stable within a person. If a person is presented with a different interpretation of given premises as the one that was constructed by the person, it seems to be very difficult to ascertain the truth of that model. Is it a correct interpretation of the given premises? In another experiment by Ragni and Knauff (2013) participants were much slower to identify correct interpretations and more likely to reject them erroneously if they differed from the preferred mental model. Only if explicit instructions such as "Generate all possible models" were given, it is probable that more than one model will be constructed by an individual. "This is a major departure from the standard model theory, which assumes that model validation always happens because people search for counterexamples to verify a putative conclusion." (Ragni and Knauff, 2013)

Ragni and Knauff (2013) argue that the farther removed an alternative model is from the individually constructed one it gets progressively more difficult to recognize as an also true interpretation of the premises.

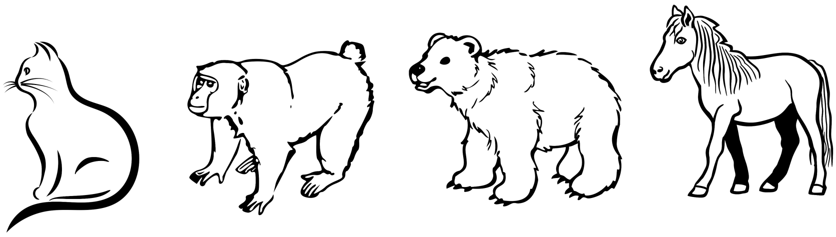
2.2 Alternative Mental Models

When presenting a set of premises such as:

- The monkey is left of the bear.
- The horse is right of the monkey.
- The cat is left of the horse.

The preferred mental model would be:

Figure 4: Cat, Monkey, Bear, Horse



Now, presenting people with alternative models, such as:

- Cat, Monkey, Horse, Bear;
- Monkey, Bear, Cat, Horse.

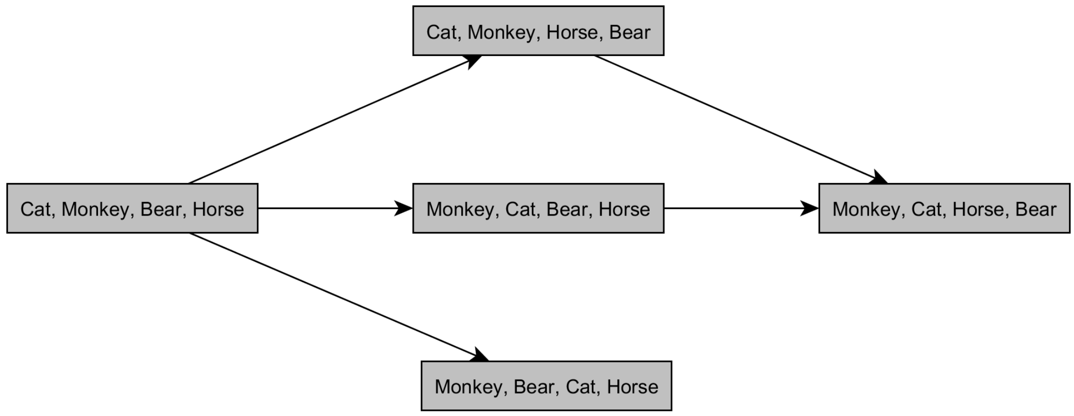
the first alternative model is much more easily recognized as valid (see experiments in Ragni and Knauff (2013) and Rauh et al. (2005)) than the second one. Ragni and Knauff (2013) argue that this is due to the distance between the preferred mental model and an alternative model. The smaller it is, the more likely an alternative model is recognized as valid. They write:

> In our theory and PRISM, this transformation distance is essential to explaining human reasoning difficulty: If an alternative model has a high transformation distance from the preferred model, it is difficult to construct and is therefore more likely to be neglected.
> (Ragni and Knauff, 2013)

To determine the distance between two models, a neighborhood graph is employed.

> In the following, we suggest using a neighborhood graph to determine the similarity between different models of a set of premises and, thus, the sequence in which models and alternative models are constructed. The idea is that the vertices in the neighborhood graph represent the models, and the edges connect the model with the fewest differences. Since some models are connected by fewer edges with other models, the similarity between models can be determined by the shortest path in this neighborhood graph. If a one-step transformation from one model to another model exists, then two models are called 1-nearest neighbors. In general, if two models can be connected by a minimal path of length k, then we call these two models k-neighbors.
> (Ragni and Knauff, 2013)

Figure 5: Neighborhood graph for ambiguous premises



The leftmost model is the preferred mental model. Following an arrow to the right increases the distance to the preferred mental model. A node is gained by looking for alternative positions of tokens whose position is not determinate. Experiments by Rauh et al. (2005) "support the assumption of a general revision process that takes the [preferred mental model] as input and locally transforms it to come to the next alternative."

3 Different Classes of Problems

A *problem* designates a set of premises and one or more conclusions, which may or may not follow from the premises. Problems can differ in some aspects, which will be described in the following.

3.1 Determinacy

A problem may be determinate or indeterminate. A determinate problem allows only one possible mental model to be constructed, e.g.:

- The cat is left of the dog.
- The horse is right of the dog.

An indeterminate problem allows several mental models to be constructed, e.g.:

- The cat is left of the dog.
- The dog is right of the horse.

Indeterminate problems may be solved differently. Ragni and Knauff (2013) differentiate between two strategies for inserting an ambiguous token: The *fff-strategy* (first free fit) or the *ff-strategy* (first fit). In an example such as the indeterminate one above, the

fff-strategy produces the preferred mental model, “Horse, Cat, Dog”. It would first produce “Cat, Dog” and then insert the horse to the left of this model, producing “Horse, Cat, Dog”.
The ff-strategy produces “Cat, Horse, Dog”, an alternative model. It would insert the horse immediately to the right of the cat, shifting the dog to the right.

3.2 Continuity

A problem may be continuous or discontinuous. A continuous problem gives tokens in order, e.g.:

- The cat is left of the dog.
- The dog is left of the horse.

Every end token of a premise is the start token of the next premise.

A discontinuous problem gives tokens out of order, e.g.:

- The cat is left of the dog.
- The horse is left of the dog.

3.3 Consistency

A problem may be consistent or inconsistent. A consistent problem gives a conclusion that follows logically out of the premises. An example would be:

- The cat is left of the dog.
- The dog is left of the horse.
- Conclusion: The cat is left of the horse.

An inconsistent problem gives a conclusion that doesn't follow logically out of the premises. An example would be:

- The cat is left of the dog.
- The horse is left of the dog.
- Conclusion: The cat is left of the horse.

3.4 Difficulty

As described in the section above, models may require different actions from the reasoner, such as inserting a new token into a previously constructed model or evaluating the truth of a conclusion. Obviously, problems differ from each other in the set of actions they require from the reasoner, and therefore, may also vary in their perceived difficulty and the reasoner's performance. Ragni and Knauff (2013) write: "The difficulty of an inference does not depend on the number of logically possible models but on the difficulty of mentally constructing preferred and alternative mental models of the circumstances the premises describe."

The question of how difficulty could be measured will be discussed in section 6.4.

4 Simulating Human Spatial Reasoning

To understand how individuals reason and what makes a problem difficult, relational reasoning problems are carried out on a computer. For a computational implementation, a data structure has to be chosen (how to represent tokens and their relations to each other) and strategies determined, such as how to proceed if an ambiguous premise is encountered.

5 Previous Implementations

Some different approaches to computationally modeling human relational reasoning will be introduced and discussed. I will limit this review to computational models that are based on the model theory by Goodwin and Johnson-Laird (2005).

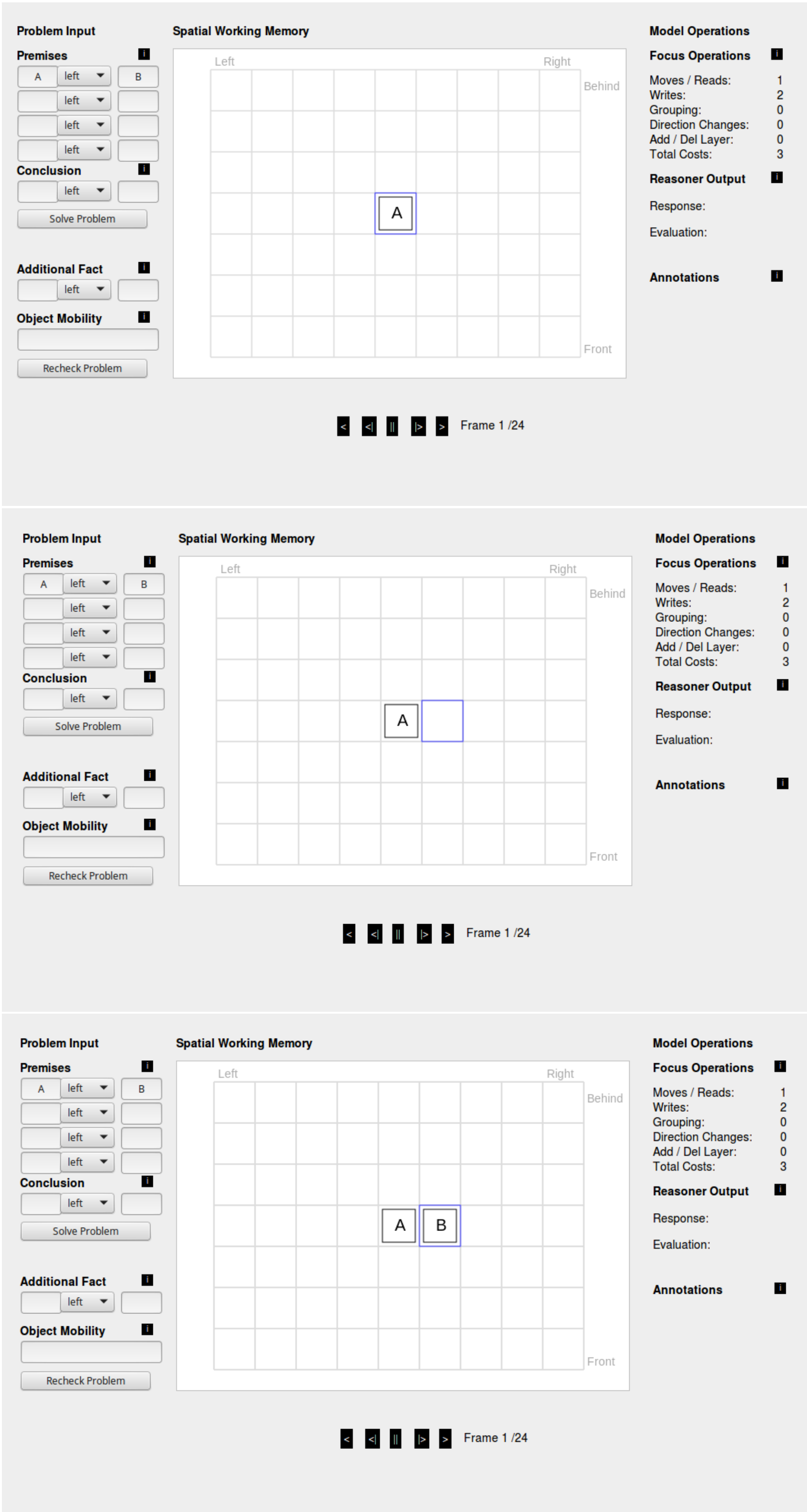
5.1 PRISM

Ragni and Knauff (2013) proposed a computational model to support the preferred model theory, "reflecting our main assumption that people usually construct just a single, simple, and typical model but fail to consider other models in which the premises hold". This computational model PRISM (PReferred Inferences in reasoning with Spatial mental Models) [4] works with a data structure that is reminiscent of the tape of a Turing machine in two dimensions [5], in which the cells of a seemingly endless matrix are filled with tokens according to the given spatial information. A head moves from cell to cell to fill in tokens. When the premise "A left B" is given, an A is written into an empty cell, the head moves one position to the right and a B is written into that cell. See figure 6 for a visual representation.

[4] `http://spatialmentalmodels.appspot.com/`

[5] `https://en.wikipedia.org/wiki/Turing_machine`

Figure 6: PRISM working on premise “A left B”



It has widely been shown and is a generally accepted fact that human working memory is limited in capacity (Baddeley, 2007), (Miller, 1956). It is very unlikely that a human reasoner is able to keep an infinite number of tokens and relations in their working memory, like PRISM is. This may be a drawback to this implementation and its utility in emulating human behavior.

PRISM collects information about the difficulty of a problem. This difficulty measure is based on the number of focus operations PRISM has to perform. A focus operation is the movement of the head from one cell to another. In the example "A left B", one focus operation is performed. In detail, the operations that PRISM counts and sums up to obtain a general difficulty measure are:

- Moves / Reads - Moving the head / Reading a token in a cell,
- Writes - Writing a token into a cell,
- Grouping - In the case of a merge of two models (in PRISM's case, this equates to two endless matrices that are merged into one), one of those models has to be grouped and inserted into the other, those are grouping operations,
- Direction Changes - If a model is constructed in one direction, e.g. in the example above, it is constructed into the right direction, and an action requires the focus head to move to the other direction, e.g. with the premise "D left A", a direction change would have occurred,
- Add / Del Layer - If two separate models are constructed, e.g. with "A left B", "C left D", two layers are created. If those layers are then merged, e.g. with "B left C", one layer will be deleted.

In some experiments [6], this difficulty measure correlated well with the number of test subjects who were able to correctly solve a problem.

PRISM deliberately chooses the fff-strategy. Ragni and Knauff (2013) state that by choosing the fff-strategy, PRISM creates the preferred mental model, which most people construct and is therefore better able to emulate human performance.

5.2 Krumnack's Linked Lists

Krumnack, Bucher, Nejasmic, Nebel, and Knauff (2011) proposed viewing relational reasoning as a form of verbal reasoning, i.e. they "assume[...] the cognitive processes in deductive reasoning to be based upon the same processes as language comprehension and generation". They specify basic attributes of the model they assume reasoners create:

[6] http://spatialmentalmodels.appspot.com/data

> There exists a starting point or first object.
> Each object is linked to the next object in the linear order. Only the last object is not linked to other objects.
> While this structure has an implicit direction, the interpretation of this direction depends on the context.
> (Krumnack et al., 2011)

Computationally, they implement this model with linked lists. A linked list has a starting node, which points to another node, which points to another node, and so on, until the final node, which doesn't point anywhere. The nodes contain a link to another node and a token.

When inserting a new token into an existing linked list, they propose minimizing the amount of new links that have to be created. When inserting a new token at the very beginning or very end of a linked list, only one new link will have to be created. This better predicts human behavior and matches the fff-strategy for inserts, as shown in an experiment in Krumnack et al. (2011).

When determining the truth of conclusions, it should be easier to check those models in which the tokens are named in the same order as they were named in the premises, because otherwise the queue has to be accessed twice: Once for finding the first token, then accessing the queue again and searching from the beginning for the second token mentioned in the conclusion. This was also shown in an experiment in Krumnack et al. (2011).

Just like with PRISM, this model is unlimited in the number of tokens it accepts, which is an unlikely assumption. It also limits the order in which objects can be accessed. That is, if there's a linked list with the content "Cat, Monkey, Bear, Horse", evaluating "Bear is left of Horse" is significantly easier than evaluating "Horse is right of Bear". The list is only traversable in one direction (left to right). It does seem to be more difficult to traverse it "in the other direction", but to effectively have to start scanning the list from the beginning (Cat) seems unlikely as well.

5.3 Bremen thingie

Will follow...

6 DaStruct

Each of these previous attempts to simulate human reasoning relied on one specific data structure. Some aspects of these data structures (e.g. the unlimited amount of tokens that can be remembered) may be unrealistic.
In this thesis, I will therefore analyze some competing data structures and methods and

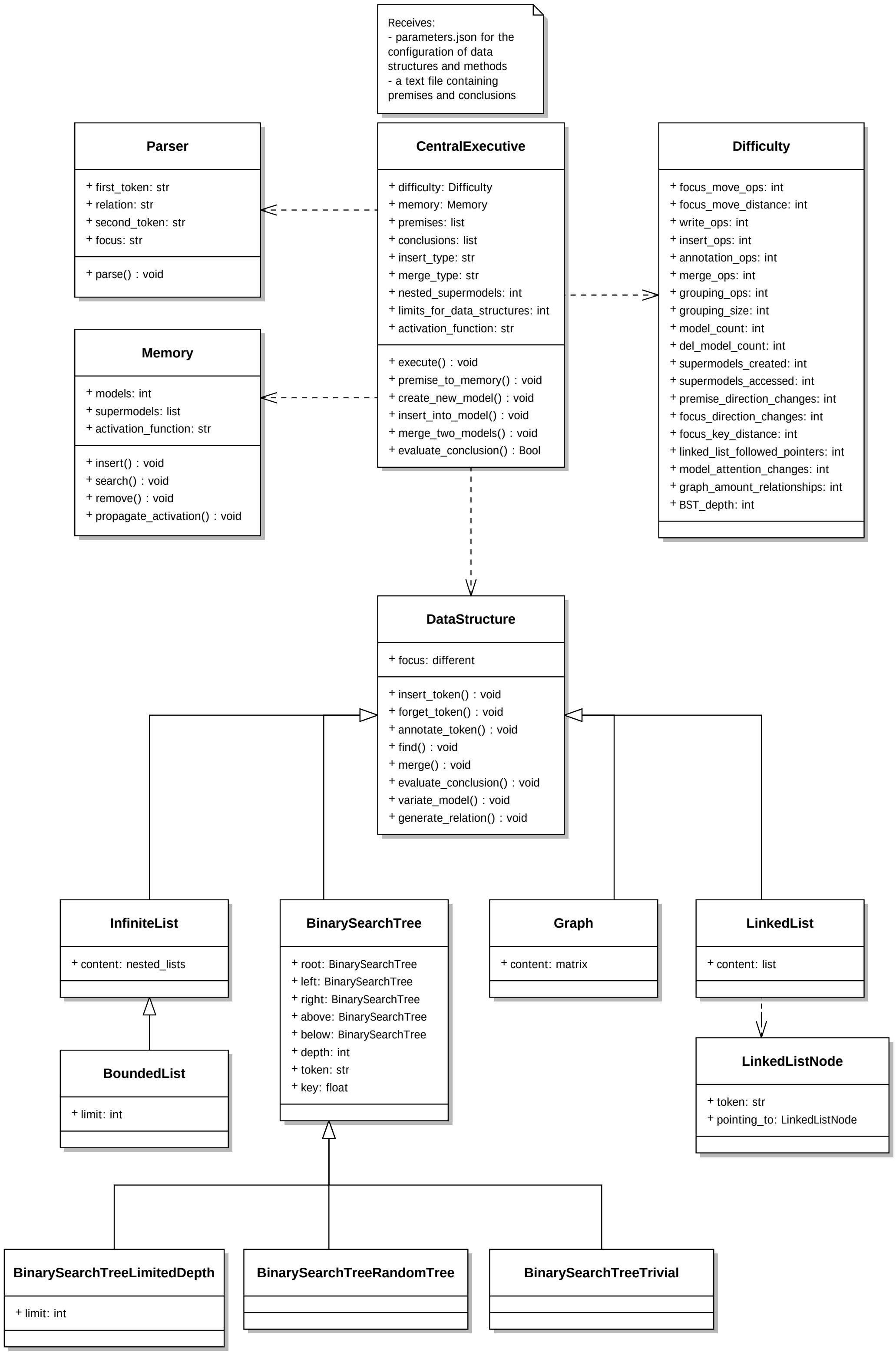
how they might relate to human relational reasoning. This computational implementation is called *DaStruct* (using DAta STRUCTures for cognitive modeling).

6.1 Implementation of DaStruct

DaStruct was implemented in the context of this bachelor's thesis, from May 2016 to August 2016. It is implemented in Python 2.7. It is object-oriented and highly modular, allowing for free combinations of its data structures and methods. It is available under a GPLv3 license and obtainable here: `https://bitbucket.org/anphisa/bachelor_thesis`. Its author is identical to the author of this bachelor's thesis.

A class diagram of DaStruct's general structure is shown in figure 7. It was simplified, e.g. not depicting helper functions and variables.

Figure 7: UML class diagram for DaStruct (simplified)



DaStruct works as follows:

6.1.1 Problem files

A problem is a text file in which each line describes a premise or a conclusion. It could look like this:

Listing 1: A sample problem file

```
P: A left B
P: B left C
C: A left C
```

A "P:" in front of a line denotes that it describes a premise, a "C:" denotes a conclusion whose truth value should be determined.
DaStruct saves these text files with a ".pf"-ending, short for problem file.
A premise or a conclusion will always contain two tokens and a relationship between them. The relationships may be "left", "right", "above" or "below".

6.1.2 Model construction phase

- The Central Executive is instantiated. [7]
- Every line of the problem file is parsed and categorized as either a premise or a conclusion.
- All premises are now iteratively added into models. For that, the type of the premise is determined:
  - A **type 1 premise** is the initial premise where both tokens have not previously been seen.
  - A **type 2 premise** is one where one token has previously been seen. The other token has to be inserted into the model of the known token.
  - A **type 3 premise** is one where both tokens have not previously been seen. A new model needs to be created for this premise. DaStruct doesn't differentiate between type 1 and type 3 premises.
  - A **type 4 premise** is one where both tokens are known. If they are part of different models, these models have to be merged.
- According to the premise type, a new model is created, a token inserted into an old one or two models are merged. In one execution of DaStruct, all models have the previously chosen same data structure type, the same insert and merge strategy.

[7] Named after one of the components of human working memory. In human working memory, the central executive directs attention and chooses which information to pass on to long term memory. In this implementation, it is the master class that controls all other classes.

- All models are held in memory.

6.1.3 Conclusion Validation or Relationship Generation Phase

- When all models were constructed, the conclusions from the problem file are validated *or* relationships between given tokens are returned.

6.1.4 Optional: Model Variation Phase

- In case a conclusion was evaluated as false, the model variation phase can be entered. Analogous to Ragni and Knauff (2013), this phase is only entered if a conclusion is falsified, reflecting the belief that humans will not question it if they evaluated a conclusion as true, but might search for alternative models if they evaluated a conclusion as false. In this phase, a neighborhood graph is constructed according to which indeterminate tokens were inserted into this model. See section 2.2 for an explanation of the neighborhood graph.

6.2 Data Structures

The data structures depicted in the class diagram (see image 7) inheriting from the general DataStructures class will be described and explained in the following. The problem file consisting of the following premises:

Listing 2: Exemplary problem file

```
P: A left B
P: B left C
P: D above A
```

will be used as an example problem to illustrate the differing representations in different data structures.

6.2.1 InfiniteList and BoundedList

InfiniteList is a data structure that is conceptually very close to PRISM's data structure. It is made up of lists containing single layers of tokens. A token is left of another token if it is further to the left in the list and vice versa.
A layer is a 2D-structure encoding left- and right-relationships. Layers above each other encode above- and below-relationships.

The exemplary problem file (see listing 2) would yield this InfiniteList representation:

Listing 3: InfiniteList representation

```
[['D', 'x', 'x'],
 ['A', 'B', 'C']]
```

The ’x’s denote empty space. It is trivial to verify a conclusion, e.g. “A left C”.

BoundedList is a data structure that is like InfiniteList, but contains a limit. The limit describes how many tokens may be saved in one BoundedList model at a time. E.g., given the problem file above and a BoundedList data structure with limit 2, the following three BoundedList models would represent the premises:

Listing 4: BoundedList representation with limit 2

```
[['A', 'B']]

[['B', 'C']]

[['D'],
 ['A']]
```

Having constructed the first model, [[’A’, ’B’]], the limit for tokens in this model is already filled. Therefore, when reading the premise “B left C”, it can not be inserted into this model anymore. A new model containing this information is created: [[’B’, ’C’]].
Obviously, this limits the possibilities of evaluating conclusions. The conclusion “A left C” can not be verified anymore, simply because there’s no model containing both the token A as well as the token C.

6.2.2 BinarySearchTree, BinarySearchTreeTrivial, BinarySearchTreeLimitedDepth and BinarySearchTreeRandomTree

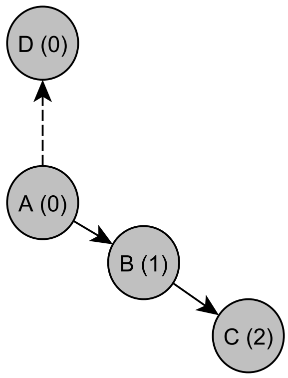
Binary search trees store information in a hierarchical data structure. A tree consists of nodes, each of which contains the information to be saved in that node and a key to index the node. Every node may have up to two child nodes. A tree has a root node which is traditionally depicted at the top of the tree, making the tree “grow downwards”. Every node may only have one parent node (or none if it is the root node). The key of a node must be bigger than its left child’s key and smaller than that of its right child’s key.

A node that is to be inserted into the tree compares its key with that of the root node: If its key is smaller, it tries an insert in the left child tree of the root tree, if its key is bigger, it tries an insert in the right child tree of the root tree. If that child tree doesn’t exist yet, the node will form this child tree. If it exists, the node will further compare its key until it finds the correct free position.

Traditionally, binary search trees only admit left and right child nodes (and thereby a two-dimensional structure to be represented), however in the problem files “above”- and “below”-relations can occur. For this case, a linking of binary search trees was implemented: The root nodes of different layers (layers describing “above”- and “below”-layers of tokens) are linked. All child nodes of the respective root nodes are informed of the existence of this new layer and thereby all nodes of a layer.

The exemplary problem file (see listing 2) would yield this BinarySearchTree representation:

Figure 8: BinarySearchTree representation



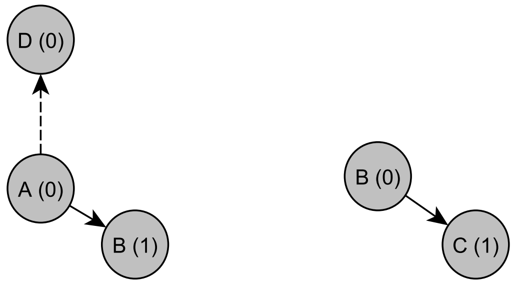
The nodes are drawn as circles, the keys indexing the nodes are in brackets after the tokens. The tokens A and D have the same key since they are in different trees (just linked with the "above"-relation). Tokens A and B have no left children, therefore no nodes to the left.
The data structure **BinarySearchTree** implements a binary search tree as described.

The **BinarySearchTreeTrivial** data structure differs only in the insert-procedure from the normal BinarySearchTree data structure. Trees support intelligent procedures such as the rotation of leaves (to free up space), which BinarySearchTreeTrivial doesn't support. [8]

The **BinarySearchTreeLimitedDepth** data structure has a depth limit, i.e. inserts node only to a given depth of the tree. The depth of a tree describes the distance of the furthest child node to the root node of a tree. E.g. with depth limit 2, the exemplary problem file (see listing 2) would yield this representation:

[8]This was rather a stage in development of the BinarySearchTree data structure than an individual idea. It worked well enough, which is why it was included in the tests.

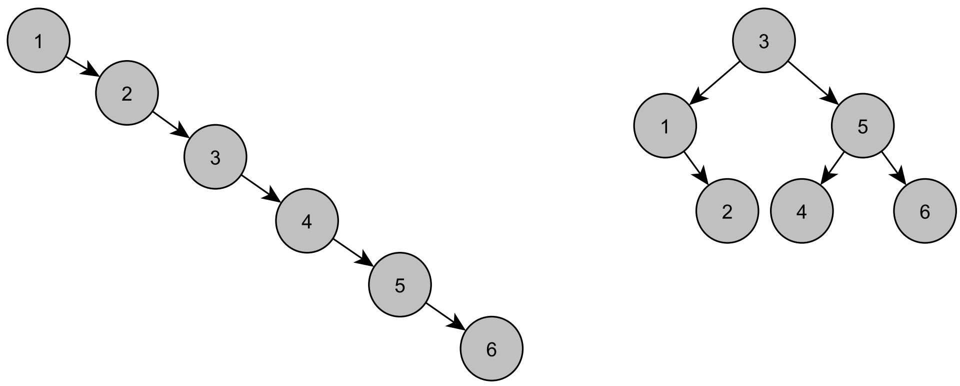
Figure 9: BinarySearchTreeLimitedDepth representation



The token D can still be inserted into the first model since it doesn't increase the depth of either tree to a depth larger than 2. However, the second premise "B left C" will yield its own tree (with own key numbering as well, so the token B has different keys in the two models). There is no alternative place to insert the token C into the first model: As token A's left child, it would be evaluated as being left of token B, which is incorrect.

The **BinarySearchTreeRandomTree** data structure builds a random binary search tree at each insert. Inserting tokens not by the order in which they "arrived" but randomly may even out the height of the tree. E.g. inserting keys 1, 2, 3, 4, 5, 6 in this order will yield a tree that is not very balanced. Inserting the same keys in a randomized order, e.g. as 3, 5, 1, 6, 2, 4 will yield a much more even tree.

Figure 10: Two binary search trees containing numbers from 1 to 6



To this end, the BinarySearchTreeRandomTree has an InfiniteList data structure at its base. Into this structure, it inserts all the tokens from the premises. For every layer in the InfiniteList, it assigns a key linearly: The leftmost object receives key 1, the next key 2, etc. Then, choosing the items from the list randomly, it inserts the objects into a BinarySearchTree structure.

In this case, it would construct the same structure as InfiniteList (see listing 3) and then assign key values. The assignation of key values is shown in table 1.

Table 1: Key assignment for random search tree

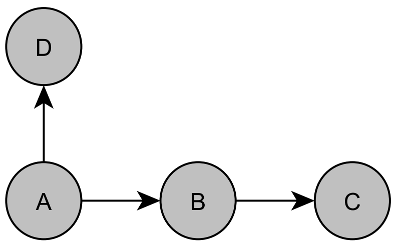
Token	Key value
D	1
A	1
B	2
C	3

After first constructing a binary search tree just for token D, it would construct a binary search tree and insert tokens A, B and C in a random order.

6.2.3 Graph

A graph stores information visually. A graph for the exemplary problem file (see listing 2) might look like this:

Figure 11: Graph representation



Other forms could be imagined, e.g. without arrows indicating the relationship between tokens, or a representation that isn't based on a grid-like structure. DaStruct doesn't allow for ambiguous relationship (as in a non grid-like structure).

A **Graph** model with the information from the exemplary problem file (see listing 2) would have this internal representation:

Listing 5: Graph representation

```
[[' ',  'A',  'B',  'C',  'D'],
 ['A',  'x',  'l',  'l',  'b'],
 ['B',  'r',  'x',  'l',  'x'],
 ['C',  'r',  'r',  'x',  'x'],
 ['D',  'a',  'x',  'x',  'x']]
```

This representation represents an adjacency matrix [9]. The first row and the first column contain the tokens which are contained in the model. The cells between them contain their relation. If the relation between B and C should be generated or evaluated, the row beginning with token B and the column beginning with token C will be chosen. The cell at their intersection describes the relation between B and C: 'l', short for left.
All other relations are similarly abbreviated. An 'x' denotes no recorded relationship.

6.2.4 LinkedList

The **LinkedList** data structure was implemented analogously to Krumnack et al. (2011)'s implementation of linked lists. LinkedLists are lists similar to the InfiniteList implementation, however they only support access in one direction (left to right). A linked list is a list of LinkedListNodes. A LinkedListNode structure contains a token and a pointer. If it is to the left of another node, its pointer will point to that node. If it is the rightmost node, its pointer won't point to another node.

The "outer list" that LinkedListNodes are contained in is necessary to enable three-dimensional indexing. Nodes that come first in the list are above nodes that come later in the list.
The outer list of LinkedListNodes for the exemplary problem file (see listing 2) would look like this:

Listing 6: LinkedList representation

```
['D', 'A']
```

Expanding the nodes, it would look like this:

Listing 7: Expanded LinkedList representation

```
['D', 'A' -> 'B' -> 'C']
```

The arrows denote a pointer from one node to another.

6.3 Data Methods and Additional Features

All data structures can freely be combined with different methods for treating the tokens and premises. Methods can describe different strategies for certain operations (such as inserts or merges), limit the size of a data structure or the memory strength of DaStruct.

[9] An adjacency matrix represents a graph by indicating which nodes are adjacent to each other, i.e. linked by an edge. In DaStruct, additionally the type of relationship between nodes (tokens) is specified.

6.3.1 Insert strategy

A type 2 premise, i.e. one where one token is known and present in at least one model and the other token is unknown, needs an insert procedure. In the exemplary problem file (see listing 2) the first premise “A left B” builds a model “A, B”, the second premise “B left C” now requires an insert of token C into the previously constructed model “A, B”. In this case the insert is deterministic and yields the model ”A, B, C”. However, for indeterminate inserts a strategy needs to be chosen. DaStruct supports the ff- and fff-strategy for inserts. For an explanation of these inserts, see section 3.1.

6.3.2 Merge strategy

A type 4 premise, i.e. one where both tokens are known, but part of different models, requires a merge of these models. DaStruct supports two merge strategies:

Integrate merge
The integrate merge integrates two models into one model. Both single models will be deleted and a new model containing the merged information will be inserted into memory.

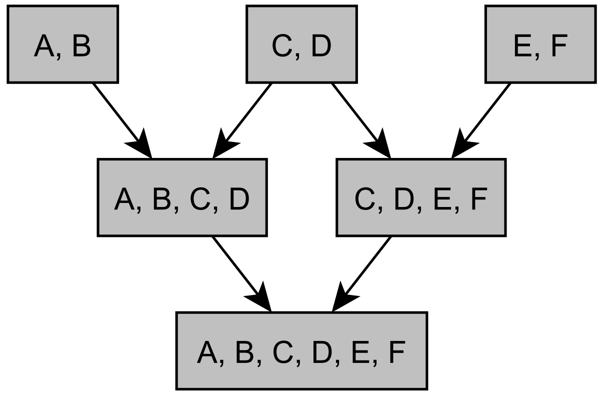
Supermodel merge
The supermodel merge will create supermodels instead of integrating models into one. This problem file:

Listing 8: Problem files for merges

```
P: A left B
P: C left D
P: E left F
P: B left C
P: D left E
```

would yield this hierarchy of supermodels:

Figure 12: Supermodel merge



The models on top are the models that represent the premises 1-3. Premises 4 and 5 specify the required merge operations. When performing the merges, the three models on top combine to the two supermodels on the second layer of the image. These two supermodels can again be combined to the supermodel of supermodels on the third layer of the image.
DaStruct has the option to limit the depth to which supermodels will be created. If that limit was set to 1 for the example above, the first layer of supermodels would still be constructed, the one below wouldn't. The premise "A left F" would then be evaluated as false. If the limit was set to 2, the second layer of supermodels would be constructed and the premise "A left F" would be evaluated as true.

6.3.3 Distance on the neighborhood graph

When premises call for indeterminate inserts, several models are possible representations of those premises. See section 2.2 for an explanation for how and when alternative mental models may be constructed. DaStruct has an optional limit for the distance on the neighborhood graph that an alternative model may have to the preferred mental model.

6.3.4 Activation function

In neural networks [10], activation functions serve as an analogue to the rate of action potential of a cell, i.e. the "likelihood of a neuron firing". In DaStruct, the activation function describes the process of forgetting information.
In 1885, Hermann Ebbinghaus started testing his own memory: He memorized sheets of nonsense syllables and tested his remaining knowledge at certain points in time

[10] Computer systems inspired by the structure of biological neural networks, e.g. constructed similarly to the human visual system.

(Ebbinghaus et al., 1913). The resulting *forgetting curve* describes how many of these syllables he could still remember at a given point in time. Extrapolating his data, the function

$$R = e^{(-t/S)} \tag{1}$$

is an approximation of his forgetting curve. R is the amount of information that was retained, t is the point in time after the learning of the information and S is a factor that describes the "memory strength". Increasing it will increase the amount of information that is remembered at a given point in time.

This curve is the basis for some activation functions that were used in experiments with DaStruct. The forgetting of tokens was modeled, not the forgetting of relations between tokens. Tokens are somewhat similar to Ebbinghaus' syllables since in the experiment they had names such as A, B or C and were therefore presumably not considerably more meaningful to participants than Ebbinghaus' random syllables. Ebbinghaus' curve describes the process of forgetting a set of syllables and therefore one chunk of information. For DaStruct, it was interesting to model the forgetting of several tokens that are not related to each other.

Yffelti (2016) proposes a function that works on the basis of Ebbinghaus' forgetting curve when several pieces of information are added into a memory system. In DaStruct's case, the time points of insertion of a token into memory were set as the premise number in which a token was first seen. It is assumed that every token has the same strength of insertion, so no token is inserted more strongly than other tokens. In that case, Yffelti (2016) propose the following formula based on the superposition principle[11]:

$$I(t) = \Delta I \cdot \sum_{i=1}^{n} e^{-\frac{t - i \cdot \Delta t}{S}} \cdot \theta(t - t_i), \tag{2}$$

where

$$\theta(t) = \begin{cases} 1, & t \geq 0, \\ 0, & t < 0 \end{cases} \tag{3}$$

is the Heaviside function [12].

For different memory strength factors S, this function is shown plotted in figures 13 and 14.

[11] https://en.wikipedia.org/wiki/Superposition_principle
[12] https://en.wikipedia.org/wiki/Heaviside_step_function

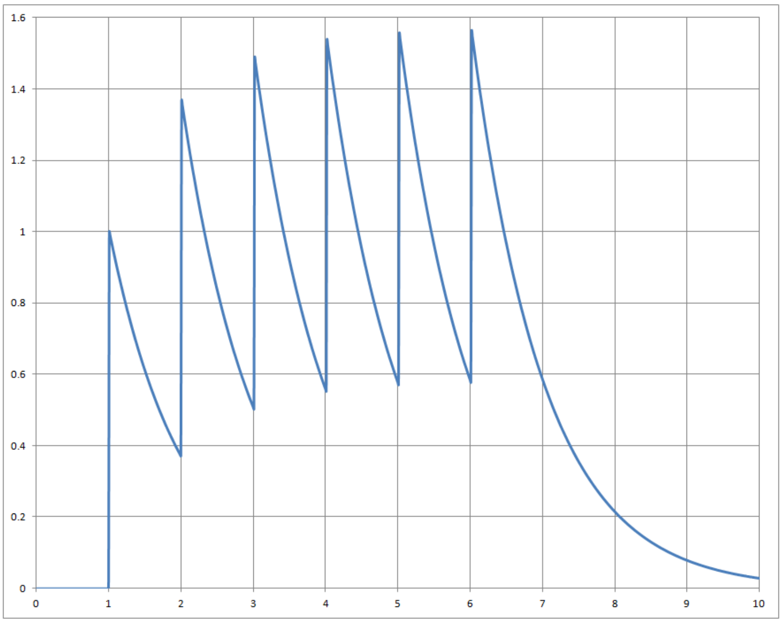


Figure 13: Equidistant information injections with equal intensities ($\Delta I_i = 1$) into a system with memory strength $S = 1$.

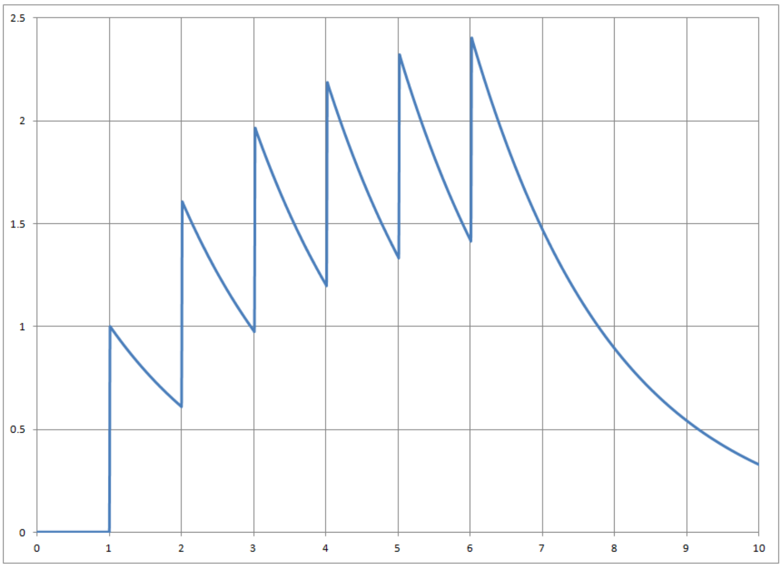


Figure 14: Equidistant information injections with equal intensities ($\Delta I_i = 1$) into a system with memory strength $S = 2$.

For DaStruct, I used the upper values of this function with memory strengths 2 and 3 to obtain a limit that denotes which tokens are forgotten again, assuming that tokens that were seen the longest ago are forgotten first. In practice, with this problem file:

Listing 9: Premise file

```
P: A left B
P: B left C
P: D above A
```

This is the table denoting the time step at which each token was first seen:

Time step	Token
1	A
1	B
2	C
3	D

Table 2: At which time step tokens were first seen

Tokens A and B were both seen in the first premise and thereby time step 1. The function given by Yffelti (2016) yields the following results for each time step:

Time step	Result of Yffelti (2016)'s function with memory strength $S = 2$
1	1
2	1.6065
3	1.9646

Table 3: At which time step tokens were first seen

In DaStruct, this result was used as a limit. That is, at time step 2 all tokens that were seen in a time step smaller than 1.6065 are forgotten. This would mean that tokens A and B are not accessible anymore. The premise "B left C" would now be categorized as a type 1/type 3 premise instead of a type 2 premise.

6.4 Difficulty Measures and Cumulated Difficulty Measures

While processing the premises and conclusion, DaStruct collects a variety of difficulty measures. PRISM collected some difficulty measures (see 5.1 for a description of these), DaStruct aims to obtain a higher resolution of measures.
The difficulty measures that were collected were chosen

- for the fact that they were easily implementable, e.g. if implementing them followed naturally out of their computational implementation,
- for the fact that a similar (or the same) difficulty measure was implemented in PRISM,
- for the fact that the author assumed that they would measure difficulty.

This exemplary problem set will help illustrate the difficulty measures collected by DaStruct:

Listing 10: Exemplary problem file

```
P: A left B
P: B left C
P: E right D
P: D right C
```

The explanations of difficulty measures are based on an unlimited data structure. The results of the measures will vary for limited data structures.

6.4.1 Collected Difficulty Measures

The following single difficulty measures were collected by DaStruct:

- **Focus move operations**: Measures whether the focus was moved at all. The focus is analogous to PRISM's head movements (see section 5.1 for those). The focus is always moved to the last token that was important. When considering the exemplary problem file (see listing 10), for the first premise one focus move operation would be performed. First, the token A would be inserted into the model and the focus would be on token A. Immediately thereafter, token B would be inserted and focus would be shifted to it. For premise 2, the focus would be shifted from token B to token C. In premise 3, the focus is shifted from token E to token D, for premise 4 no focus move operation is necessary since in both the models that are merged the focus lies on the tokens that describe the merge operation.
  Altogether, the exemplary problem file has 3 focus move operations.

- **Focus move distance**: Measures how far the focus was moved. In the exemplary problem file (see listing 10), every focus move operation only moves the focus for distance 1. If the distance between tokens is bigger, the focus move distance increases. E.g. given the premises "A left B", "B left C" and "D left A", the focus would have to move from token C to token A for the third premise. The move distance of that move operation would be 2.
  Altogether, the exemplary problem file has a focus move distance of 3.

- **Focus key distance**: Measures the key distance when performing a focus move operation in a BinarySearchTree. When the node in focus changes in a BinarySearchTree, this measure collects the key distance between them.
  Altogether, the exemplary problem file has a focus key distance of xxx, when utilizing the base data structure BinarySearchTree.

- **Write operations**: Measures the amount of write operations. When no merge operations are required, this equates to the number of tokens that are mentioned. If a merge operation is required and the merge strategy is set to integrate, one set of

tokens need to be "rewritten", i.e. integrated into another model. This increases the write operation measure to a number higher than the number of tokens mentioned. In the exemplary problem file (see listing 10), the model "D, E" needs to be inserted into the model "A, B, C" on the basis of the premise "D right C". 5 write operations were performed up to this point. If the merge strategy is set to integrate, the model "D, E" would need to be rewritten, thus creating the merged model "A, B, C, D, E".
Altogether, the exemplary problem file requires 7 write operations if the merge strategy is set to integrate and 5 if the merge strategy is set to supermodel.

- **Insert operations**: Measures the amount of insert operations. For a type 2 premise, a token needs to be inserted into an existing model.
Altogether, the exemplary problem file requires one insert operation. The first and third premise are type 1/type 3 premises and do not specify an insert. The fourth premise is a type 4 premise. Only the second premise is a type 2 premise and requires one insert operation.

- **Merge operations**: Measures the amount of merge operations. For a type 4 premise, two models need to be merged.
Altogether, the exemplary problem file requires one merge operation, as specified in the fourth premise.

- **Supermodels created**: Measures the amount of supermodels that were created. If a merge is performed, this increases the amount of supermodels. For the example in **??**, this would be 3.
Altogether, the exemplary problem file would, if executed with the supermodel merge strategy, require one supermodel to be created.

- **Supermodels accessed**: Measures the amount of supermodels that needed to be accessed. If a conclusion is to be evaluated, supermodels are checked going iteratively deeper into the hierarchy of supermodels. How many supermodels needed to be checked to verify a conclusion is what is measured here. If a conclusion is inconsistent, all supermodels are accessed.
Altogether, the exemplary problem file requires no supermodels to be accessed.

- **Grouping operations**: Measures the amount of times the content of a model is grouped for a merge operation. In preparation for a merge operation, the entire content of a model is grouped.
Altogether, the exemplary problem file requires one grouping operation.

- **Grouping size**: Measures the size (in amount of tokens) that were grouped for a merge. In the case of the exemplary problem file (see listing 10), the second model is merged into the first model. The second model is grouped. It contains two tokens.
Altogether, the exemplary problem file requires a grouping of size 2.

- **Premise direction changes**: Measures how often the relationship that is specified in the premises changes. In the exemplary problem file (see listing 10), the first two premises contain the relationship "left", while the second two premises contain the relationship "right". This change is measured.
  Altogether, the exemplary problem file has one premise direction change.

- **Focus direction change**: Measures how often the direction of the focus changed. Every time a focus move operation occurs, the focus moves into a direction. E.g. in the first premise of the exemplary problem file (see listing 10), the focus moves from token A to token B. Since token A is left of token B, the focus moves into the direction right. That direction stays the same for premise two. In premise three, no change of focus direction occurs, since "E right D" is the initial premise for a new model. Since premise four does not require any focus move operation, there's no focus direction change required.
  Altogether, the exemplary problem file has no focus direction changes.

- **Linked List followed a pointer**: Measures how often a pointer was followed in a LinkedList data structure.
  Altogether, the exemplary problem file requires xxx pointers to be followed.

- **Model attention changes**: Measures how often the attention changes from one model to another. In the exemplary problem file (see listing 10), the first two premises describe one model. The third premise instantiates a new model. The attention is shifted to this second model. The fourth premise describes a merge. The attention is shifted to the newly created merged model.
  Altogether, the exemplary problem file requires 3 model attention changes.

- **Annotation operations**: Measured how often ambiguous tokens were inserted. Every time a token is inserted whose position is described by an indeterminate premise, this ambiguity is noted inside the model. If a model variation phase is enabled (see section 2.2), these annotations enable the correct local transformation to the next node on the neighborhood graph.
  Altogether, the exemplary problem file requires no annotation operations.

- **Model count**: Measures how many single models were created. In the exemplary problem file (see listing 10), two models are instantiated, once in premise 1 and once in premise 3. They are both type 1/type 3 premises and therefore instantiate a new model.
  Altogether, the exemplary problem file has a model count of 2.

- **Deletion of Models Count**: Measures how often models are deleted. In the exemplary problem file (see listing 10), two models are instantiated, once in premise 1 and once in premise 3. They are merged by premise 4.
  Altogether, the exemplary problem file requires xxx deletion of models.

- **Amount of relationships in Graph**: Measures the amount of relationships recorded in a model of data structure type Graph.
  Altogether, the amount of relationships for the exemplary problem file when using data structure type Graph was xxx.
- **BinarySearchTree Depth**: Measures the maximum depth of BinarySearchTree models. For all models, this measures the BinarySearchTree with the highest depth.
  Altogether, the maximum depth for the exemplary problem file when using data structure type BinarySearchTree was xxx.

6.4.2 Summed Up Difficulty Measures

To simplify testing DaStruct, I propose the following sums of difficulty measures: "Sum of difficulty measures", "Sum of focus operations", "Sum of focus operations (focus distance not considered)", "Sum of focus operations (only focus distance considered)", "Sum of merge operations", "Sum of model operations"

- **Sum of difficulty measures**: The sum of all difficulty measures that were collected
- **Sum of focus operations**: The sum of all focus operations. These are:
  - Focus move operations
  - Focus move distance
  - Focus direction changes
  - Model in attention changes
- **Sum of focus operations (focus distance not considered)**: Sum of focus operations, but without Focus move distance
- **Sum of focus operations (only focus distance considered)**: Sum of focus operations, but without Focus move operations
- **Sum of merge operations**: The sum of all merge operations. These are:
  - Merge operations
  - Supermodels created
  - Supermodels accessed
  - Grouping operations
  - Grouping size
- **Sum of model operations**: The sum of all model operations. These are:
  - Supermodels created
  - Model in attention changes

– Model count
– Deletion of model count

7 Experiments with DaStruct

In order to test the utility of DaStruct, two experiments were designed comparing DaStruct's performance with the performance of human test participants.
The data that was used was collected in January 2012 in an online experiment. 40 test subjects participated in the experiment (mean age 34.97, standard deviation of age 10.28, 57.5% female). These 40 participants did the first part of the experiment: 16 problems were shown, each consisting of two premises and one conclusion. Of those 16 problems, 8 were consistent and 8 were inconsistent.
Of those 40 participants, 35 also completed the second part of the experiment (mean age 33.94, standard deviation age 9.73, 50% female). 48 problems were shown, each consisting of three premises and one conclusion. Of those 48 problems, 24 were consistent and 24 were inconsistent.
Participants could read the premises in a self-paced way. When they finished reading one premise and the next premise appeared, the last premise disappeared. Their reading time for the premises and answer time for the conclusion was measured.

7.1 Matching Percentages of Tests Subjects Able to Solve a Problem

In a first experiment, the following question was to be tested: Is it possible to predict mean human ability to solve a problem by the difficulty measures that DaStruct produces?
In other words: Do difficulty measures correlate with the percentage of test subjects who solved a problem correctly?

7.1.1 The Experiment

In a first analysis, the first set of problems which contained two premises and a conclusion and the second set of problems containing three premises and a conclusion were evaluated separately. In a second analysis, all problems were evaluated together. The following tables show how many problems could be solved correctly by a given data structure as well as the correlation coefficients [13] between difficulty measures generated by DaStruct and the percentage of test subjects able to correctly solve a problem. The correlation coefficient should be negative, since a higher difficulty measure should correlate with fewer participants solving the problem correctly and vice versa.
In those problems with only two premises and one conclusion, no merges were required. Therefore, merge operations and model operations were not considered for these problems.
There was no activation function specified, i.e. tokens were remembered indefinitely.

[13] Computed using Pearson product-moment correlation coefficient

In the following tables, the columns with the highest correlation factors were highlighted (and sometimes additionally cells from other columns with even higher correlation factors).
The rows in the tables are "Number of correctly solved problem instances", "Sum of difficulty measures", "Sum of focus operations", "Sum of focus operations (focus distance not considered)", "Sum of focus operations (only focus distance considered)", "Sum of merge operations", "Sum of model operations". To save space, in the tables these row names were shortened. See section **??** for a description of these difficulty measures.

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	16	16	8	16	16	15	4	4	16	16	16
Difficulty	-0.382	-0.382	-0.49	-0.382	-0.2	-0.22	-0.483	-0.483	-0.302	0.356	-0.192
Focus		-0.239		-0.239	-0.129	-0.21	-0.431	-0.431	-0.199	0.493	-0.09
(no δ)		-0.239		-0.239	-0.189	-0.23	-0.25	-0.25	-0.194	0.507	-0.129
(only δ)		-0.239		-0.239	-0.09	-0.2	-0.441	-0.441	-0.199	0.47	-0.085

Table 4: Comparison1 - Two premises and one conclusion, solved using ff-strategy

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	16	16	8	16	16	16	8	8	16	16	16
Difficulty	-0.239	-0.382	-0.49	-0.382	-0.15	-0.03	-0.2	-0.2	-0.167	0.356	-0.192
Focus		-0.239		-0.239	-0.129	-0.14	-0.204	-0.204	-0.199	0.493	-0.09
(no δ)		-0.239		-0.239	-0.189	-0.013	-0.121	-0.121	-0.194	0.507	-0.129
(only δ)		-0.239		-0.239	-0.09	-0.12	-0.173	-0.173	-0.199	0.47	-0.085

Table 5: Comparison1 - Two premises and one conclusion, solved using fff-strategy

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	48	48	25	25	45	48	20	18	45	48	48
Difficulty	-0.345	-0.307	-0.279	-0.229	0.146	-0.115	-0.021	0.087	0.136	0.013	0.005
Focus		-0.198	-0.315	-0.152	0.173	-0.069	0.087	0.124	0.176	0.274	-0.026
(no δ)		-0.279	-0.348	-0.186	0.009	-0.301	0.116	0.114	-0.283	-0.333	-0.02
(only δ)		-0.285	-0.348	-0.186	0.173	-0.087	0.09	0.112	0.188	0.282	-0.02
Merge		-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251
Model		-0.276	-0.379	-0.282	-0.271	-0.271	-0.185	-0.271	-0.271	-0.276	-0.276

Table 6: Comparison1 - Three premises and one conclusion, solved using ff-strategy and integrate-merge

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	48	48	25	25	48	48	14	27	48	48	48
Difficulty	-0.345	-0.307	-0.279	-0.279	0.148	-0.053	0.029	0.217	0.147	0.013	0.005
Focus		-0.198	-0.316	-0.316	0.162	-0.014	0.125	0.213	0.167	0.274	-0.026
(no δ)		-0.279	-0.348	-0.349	0.073	-0.243	0.095	0.078	-0.259	0.133	-0.333
(only δ)		-0.285	-0.348	-0.349	0.16	-0.03	0.128	0.22	0.179	0.282	-0.02
Merge		-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251
Model		-0.276	-0.379	-0.379	-0.271	-0.271	-0.196	-0.271	-0.271	-0.276	-0.276

Table 7: Comparison1 - Three premises and one conclusion, solved using fff-strategy and integrate-merge

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	48	41	25	25	38	41	16	11	38	41	41
Difficulty	-0.345	-0.278	-0.3	-0.194	-0.045	-0.06	0.155	0.274	-0.08	0.14	-0.122
Focus		-0.021	-0.379	0.044	-0.036	-0.027	0.16	0.294	-0.085	0.149	-0.006
(no δ)		-0.139	-0.425	-0.056	-0.005	0.05	0.152	0.346	-0.066	0.148	0.056
(only δ)		-0.144	-0.425	-0.056	-0.045	-0.035	0.162	0.294	-0.079	0.148	-0.038
Merge		-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251
Model		-0.282	-0.425	-0.291	-0.282	-0.282	-0.234	-0.282	-0.282	-0.282	-0.282

Table 8: Comparison1 - Three premises and one conclusion, solved using ff-strategy and supermodel-merge with 0 supermodels admitted

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	48	48	32	32	45	48	23	18	45	41	48
Difficulty	-0.345	-0.393	-0.347	-0.312	-0.074	-0.222	0.107	0.244	-0.11	0.102	-0.288
Focus		-0.198	-0.316	-0.152	-0.036	-0.192	0.159	0.294	-0.085	0.149	-0.187
(no δ)		-0.279	-0.348	-0.186	-0.005	-0.14	0.152	0.346	-0.066	0.148	-0.133
(only δ)		-0.285	-0.348	-0.186	-0.045	-0.194	0.162	0.295	-0.079	0.148	-0.21
Merge		-0.345	-0.345	-0.345	-0.251	-0.251	-0.345	-0.345	-0.251	-0.251	-0.251
Model		-0.276	-0.379	-0.282	-0.276	-0.276	-0.24	-0.276	-0.276	-0.276	-0.276

Table 9: Comparison1 - Three premises and one conclusion, solved using ff-strategy and supermodel-merge with 1 supermodel admitted

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	48	41	25	25	41	41	10	20	41	41	41
Difficulty	-0.345	-0.278	-0.3	-0.194	-0.052	-0.044	0.133	0.2	-0.075	0.14	-0.122
Focus		-0.021	-0.379	0.044	-0.029	0.002	0.155	0.193	-0.06	0.149	-0.006
(no δ)		-0.139	-0.425	-0.056	0.044	0.047	0.122	0.12	-0.004	0.148	0.056
(only δ)		-0.144	-0.425	-0.056	-0.038	-0.006	0.155	0.19	-0.054	0.148	-0.038
Merge		-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251	-0.251
Model		-0.282	-0.425	-0.291	-0.282	-0.282	-0.249	-0.282	-0.282	-0.282	-0.282

Table 10: Comparison1 - Three premises and one conclusion, solved using fff-strategy and supermodel-merge with 0 supermodels admitted

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	48	48	32	32	48	48	17	27	48	41	48
Difficulty	-0.345	-0.393	-0.347	-0.312	-0.076	-0.185	0.103	0.175	-0.102	0.102	-0.288
Focus		-0.198	-0.316	-0.152	-0.029	-0.099	0.155	0.193	-0.06	0.149	-0.187
(no δ)		-0.279	-0.348	-0.186	0.044	-0.156	0.122	0.12	-0.004	0.148	-0.133
(only δ)		-0.285	-0.348	-0.186	-0.038	-0.101	0.155	0.19	-0.054	0.148	-0.21
Merge		-0.345	-0.345	-0.345	-0.251	-0.251	-0.345	-0.345	-0.251	-0.251	-0.251
Model		-0.276	-0.379	-0.282	-0.276	-0.276	-0.251	-0.276	-0.276	-0.276	-0.276

Table 11: Comparison1 - Three premises and one conclusion, solved using fff-strategy and supermodel-merge with 1 supermodel admitted

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	64	64	33	41	61	64	24	22	61	64	64
Difficulty	-0.524	-0.517	-0.445	-0.409	0.026	-0.373	-0.06	-0.139	0.012	-0.413	-0.281
Focus		-0.444	-0.477	-0.329	0.125	-0.333	0.103	0.026	0.167	0.013	-0.279
(no δ)		-0.483	-0.522	-0.378	-0.09	-0.49	0.055	-0.05	-0.355	-0.116	-0.478
(only δ)		-0.49	-0.522	-0.378	0.157	-0.318	0.13	0.044	0.193	0.005	-0.26
Merge		-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356
Model		-0.375	-0.554	-0.45	-0.372	-0.372	-0.336	-0.372	-0.372	-0.375	-0.375

Table 12: Comparison1, solved using ff-strategy and integrate-merge

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	48	64	33	41	64	64	22	35	64	64	64
Difficulty	-0.524	-0.517	-0.445	-0.409	0.003	-0.389	0.044	0.027	-0.003	-0.413	-0.281
Focus		-0.444	-0.477	-0.329	0.093	-0.445	0.176	0.127	0.146	0.0134	-0.279
(no δ)		-0.483	-0.522	-0.378	-0.084	-0.464	0.086	-0.05	-0.331	-0.116	-0.478
(only δ)		-0.49	-0.522	-0.378	0.117	-0.434	0.203	0.16	0.174	0.005	-0.26
Merge		-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356
Model		-0.375	-0.554	-0.45	-0.372	-0.372	-0.344	-0.372	-0.372	-0.375	-0.375

Table 13: Comparison1, solved using fff-strategy and integrate-merge

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	64	57	33	41	54	57	20	15	54	57	57
Difficulty	-0.524	-0.521	-0.494	-0.423	-0.017	-0.065	0.239	0.229	-0.039	-0.037	-0.142
Focus		-0.269	-0.554	-0.157	-0.003	-0.032	0.254	0.266	-0.038	0.086	-0.006
(no δ)		-0.379	-0.588	-0.293	-0.026	0	0.258	0.326	-0.043	0.063	0.035
(only δ)		-0.374	-0.588	-0.293	0	-0.03	0.264	0.276	-0.03	0.084	-0.03
Merge		-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356
Model		-0.38	-0.588	-0.473	-0.38	-0.38	-0.384	-0.38	-0.38	-0.38	-0.38

Table 14: Comparison1, solved using ff-strategy and supermodel-merge with 0 supermodels admitted

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	64	64	40	48	61	64	27	22	61	57	64
Difficulty	-0.524	-0.56	-0.486	-0.46	-0.053	-0.312	0.194	0.192	-0.075	-0.114	-0.361
Focus		-0.444	-0.477	-0.33	-0.003	-0.28	0.254	0.266	-0.038	0.086	-0.245
(no δ)		-0.483	-0.522	-0.378	-0.026	-0.254	0.258	0.326	-0.043	0.063	-0.202
(only δ)		-0.49	-0.522	-0.378	0	-0.27	0.264	0.276	-0.03	0.084	-0.26
Merge		-0.425	-0.425	-0.425	-0.351	-0.351	-0.425	-0.425	-0.351	-0.356	-0.351
Model		-0.375	-0.554	-0.45	-0.375	-0.375	-0.38	-0.375	-0.375	-0.375	-0.375

Table 15: Comparison1, solved using ff-strategy and supermodel-merge with 1 supermodel admitted

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	64	57	33	41	57	57	18	28	57	57	57
Difficulty	-0.524	-0.521	-0.494	-0.423	-0.062	-0.087	0.224	0.191	-0.068	-0.037	-0.142
Focus		-0.27	-0.554	-0.157	-0.032	-0.068	0.258	0.21	-0.04	0.086	-0.006
(no δ)		-0.379	-0.588	-0.293	-0.005	0.003	0.24	0.157	-0.02	0.063	0.035
(only δ)		-0.374	-0.588	-0.293	-0.03	-0.067	0.264	0.218	-0.032	0.084	-0.03
Merge		-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356	-0.356
Model		-0.38	-0.588	-0.473	-0.38	-0.38	-0.395	-0.38	-0.38	-0.38	-0.38

Table 16: Comparison1, solved using fff-strategy and supermodel-merge with 0 supermodels admitted

	PRISM	InfiniteList	BoundedList2	BoundedList3	BinarySearchTree	BSTRandomTree	BSTLimitedDepth2	BSTLimitedDepth3	BSTTrivial	Graph	LinkedList
Solved instances	64	64	40	48	64	64	25	35	64	57	64
Difficulty	-0.524	-0.56	-0.486	-0.46	-0.094	-0.322	0.194	0.163	-0.1	-0.114	-0.361
Focus		-0.444	-0.477	-0.33	-0.032	-0.287	0.258	0.21	-0.04	0.086	-0.245
(no δ)		-0.483	-0.522	-0.378	-0.005	-0.252	0.24	0.157	-0.02	0.063	-0.202
(only δ)		-0.49	-0.522	-0.378	-0.03	-0.277	0.264	0.218	-0.032	0.084	-0.26
Merge		-0.425	-0.425	-0.425	-0.351	-0.351	-0.425	-0.425	-0.351	-0.356	-0.351
Model		-0.375	-0.554	-0.45	-0.375	-0.375	-0.39	-0.375	-0.375	-0.375	-0.375

Table 17: Comparison1, solved using fff-strategy and supermodel-merge with 1 supermodel admitted

7.1.2 Results and Discussion

Some interesting relationships surfaced and shall be discussed.

Two premises and one conclusion

Since there were no merge operations required in this set of problems, merge and model correlation were not measured. When looking at tables 4 and 5, the sum of difficulty measures of DaStruct correlated more highly than the sum of difficulty measures of PRISM.

In table 4, in which the ff-strategy was employed, the sum of difficulty of BinarySearchTree and derivatives correlated much higher than in table 5, in which the fff-strategy was employed. In fact, this trend holds through all the different combination of BinarySearchTrees: Those combinations utilizing a ff-strategy correlate higher than those utilizing a fff-strategy. This might be explained by the higher amount of operations required when utilizing the ff-strategy. When using a data structure of type BinarySearchTree, an insert with the ff-strategy will demand more operations since leaves may need to be rotated. With an insert using the fff-strategy, a leaf will only be "appended" to one end of the tree, which is computationally easier.

Here and in all following comparisons, LinkedList yields the same correlation factors whether employed together with the ff-strategy or the fff-strategy. This is due to DaStruct's implementation that doesn't differ in its counting of followed links between ff and fff. E.g., when inserting a new node, it identifies the parent of the located node, whether that parent will be needed in the insert procedure or not. Due to the generally low correlation factors of LinkedList, changing this implementation would probably not yield significantly higher correlation factors.

In this set of problems in which only insert operations were required, the data structure BinarySearchTreeLimitedDepth with depth 2 and 3 correlated highly when employed together with an ff-strategy.

The highest measured correlation was -0.49 as sum of difficulty measures for BoundedList with a limit of 2, in contrast to PRISM's correlation of -0.239.

Three premises and one conclusion

This set of problems was computationally more challenging, since there were more premises and therefore more operations required. There were problems that required merge operations and therefore more possible combinations of data structures and methods for this set, since merge operations can be performed with an integrate-merge or a supermodel-merge.

Using integrate merge

It is observable in tables 6 and 7 than BoundedList with limit 3 has a much lower correlation that BoundedList with limit 2. This is especially the case when combined with the ff-strategy and can easily be explained: When an insert is performed into a list of size 3, the inserted token can most probably still be inserted into that list (since a list is instantiated with at most 2 tokens for a type 1 premise). When the list limit is 2, the token can never be inserted into the list. Therefore the amount of operations (especially model operations) is higher for BoundedLists with limit 2 and the correlation grows. The difference between the ff- and fff-strategy in this case is as follows: When perform-

ing an ff-insert, the new token "lands closer" to the known token. The insert may still be contained in the list even if for an fff-insert, it may not be anymore.
In tables 6 and 7 a trend begins to manifest that holds for later comparisons: The correlation of model operations is higher than that of the sum of difficulties.
The highest correlation was -0.379 for model operations of BoundedList with limit 2 when using ff- or fff-strategy and integrate merge, in contrast to PRISM's correlation of -0.345.

Using supermodel merge
In tables 8, 9, 10 and 11 BoundedList with limit 2 crystallizes as the most highly correlating data structure across all measures.
Another trend manifests: Combining data structures with a supermodel merge, but admitting 0 supermodels (and thereby prohibiting the merge) shows higher correlations than admitting 1 supermodel (allowing the merge). Prohibiting the merge will increase the amount of computational operations performed, since all models which contain one or both tokens from the conclusion will be checked for whether they admit the conclusion or not. When the merge is allowed, usually the first model will contain one token and its supermodel will contain both tokens.
The highest correlation was -0.425 for model operations of BoundedList with limit 2 when using ff- or fff-strategy and supermodel merge with 0 supermodels admitted, in contrast to PRISM's correlation of -0.345.

The entire set of problems combined
In this test, problems consisting of two premises or three premises were not split up, but considered together.

Using integrate merge
In tables 12 and 13, BoundedList with limit 2 performed the highest, directly followed by InfiniteList, continuing the trend of BoundedList yielding the highest correlation factors.
The highest correlation was -0.554 for model operations of BoundedList with limit 2 when using ff- or fff-strategy and integrate merge, in contrast to PRISM's correlation of -0.524.

Using supermodel merge
In tables 14, 15, 16 and 17, across almost all measures, BoundedList with limit 2 has the highest correlation factors, topping in model operations. When using the sum of difficulty measures instead of model operations as a correlating figure, InfiniteList performs higher than BoundedList in tables 15 and 17, i.e. in those in which a merge was allowed with 1 supermodel.
The highest correlation was -0.588 for model operations of BoundedList with limit 2 when using ff- or fff-strategy and supermodel merge with 0 supermodels admitted, in contrast to PRISM's correlation of -0.524.

Summary of Observations

- BinarySearchTrees combined with the ff-strategy for integration instead of the fff-strategy correlated more highly
- Data structures combined with the supermodel-strategy for merging correlated more highly than data structures combined with the integrate-strategy
- Data structures combined with a supermodel-strategy for merging where 0 supermodels were admitted (factually no merge allowed) correlated more highly than those where 1 supermodel was admitted
- The data structures
  - Graph
  - BinarySearchTree
  - BinarySearchTreeRandom
  - BinarySearchTreeTrivial
  - LinkedList

  didn't correlate highly in any of the combinations tested
- The data structure BoundedList with limit 2 seems to correlate most highly of all data structures

Since the highest performing data structure (especially in the more complicated problem set) was BoundedList with limit 2, it follows that implementing a reasoner like PRISM, with a data structure at its base that is equivalent to InfiniteList, is not a bad move when trying to estimate the difficulty of a problem set for human participants. It should however not be unlimited in size.
Model operations correlated more highly than the sum of difficulty measures. Further research could be conducted on which parts of the model operations most influence could explain that observation.
It is worth noting that almost all combinations that increased the correlation to the percentage of test subjects able to solve a problem only made a problem more **difficult** to solve, e.g. by limiting the data structures unrealistically. Miller (1956)'s magical number was 7, not 2 - it is most likely that humans are able to keep more than two tokens concurrently in their working memory. Limiting a data structure so that one model can only contain two tokens increases the correlation and may better predict the difficulty for human test subjects, but is it realistic? Does it say anything about what data structures and methods best represent human cognitive structures?

7.2 Matching Individuals Solving Problem Sets

Therefore, the question that motivated the next experiment was: Is it possible to say for an individual test subject what data structure best represents their reasoning process? In other words: Is it possible to match an individual's performance in a problem set to the performance of a certain combination of data structure and data methods?

7.2.1 The Experiment

The individual responses of test participants (true/false) were compared with the response of a certain combination of data structures and methods for every problem.
For every problem that a human participant answered, a combination of data structures and methods was "voted up" to find those combinations that best matched an individual's reasoning process. The three highest rated combinations were returned for each participant. When evaluating the highest rated combinations, a relationship was usually pretty strong (a participant scoring votes in only one of all possible categories for data structures, e.g.). If that relationship wasn't that pronounced, a barrier or 66.6% was introduced: If one participant scored more than $\frac{2}{3}$ of their votes in one category, they were matched to it.
All data structures described in section 6.3 were utilized, as well as both insert strategies and both merge strategies as described in section 6.3. The distance on the neighborhood graph limit was set to 0. Four activation functions (see section 6.3.4) were used:

- Yffelti (2016)'s function with memory strength parameter S set to 2
- Yffelti (2016)'s function with memory strength parameter S set to 2
- $x - 10$, which meant that no token was forgotten
- A linear activation function which remembered all tokens from the last two seen premises and forgot all tokens before that.

Again, the problem set was split up into those problems consisting of two premises and one conclusion and those containing three premises and one conclusion.
Of the original 40 test participants, 18 were excluded: They either didn't solve all of the tasks or behaved otherwise suspiciously. Some began answering the tasks taking a certain time span which suddenly decreased significantly, possibly meaning that they started working on the tasks seriously and then lost their interest, but clicked through the rest of the study. 22 test subjects remained who finished both parts of the problem set and seemed reliable.

7.2.2 Results and Discussion

Two premises and one conclusion
There were 16 problems with two premises and one conclusion. For the test participants, 9 problems could be matched on average (mean = 9 problems, std = 1.23, highest

matched value = 12 problems, lowest matched value = 8). That is, the highest rated combination of data structure and data methods could explain between 8 and 12 problems per participant.
In these problems, only insert procedures and no merge procedures were required. Therefore, the preferred merge strategy was not determined.
It was difficult to reliably match a participant to a specific data structure. This is due to the high amount of data structures that received the highest vote: On average, 39.27 data structure received the highest vote per participant (mean = 39.27, std = 17.24). Only those combinations with the highest vote per participant were evaluated, since most participants didn't have a second or third tier of votes. Some visualizations of the gleaned data will be shown and discussed in the following.

Figure 15: Two premises and one conclusion - Matched data structures

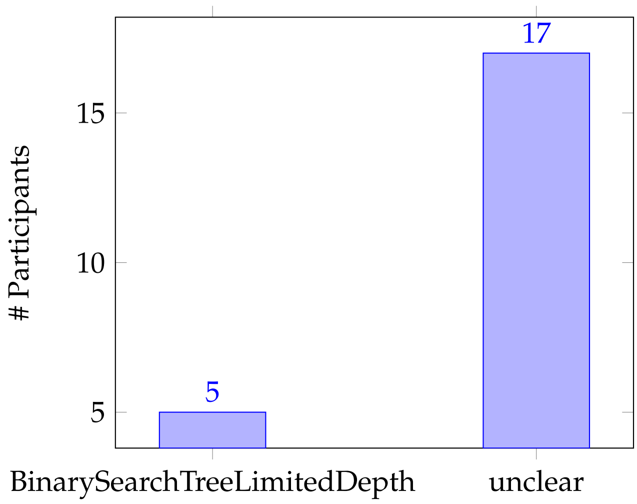


Figure 16: Two premises and one conclusion - Matched insert types

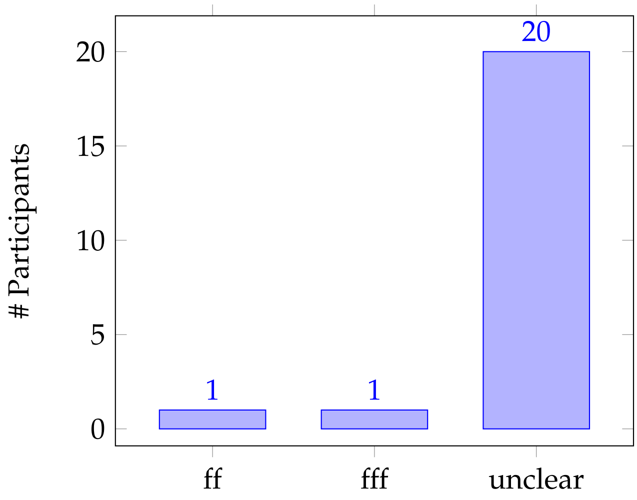


Figure 17: Two premises and one conclusion - Matched limits for data structures

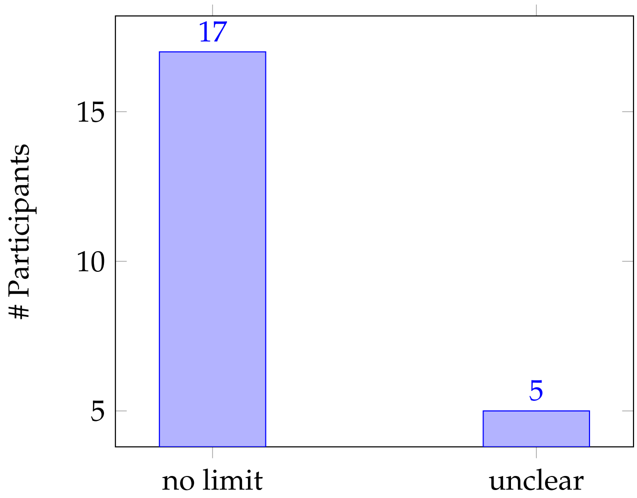


Figure 18: Two premises and one conclusion - Matched activation functions

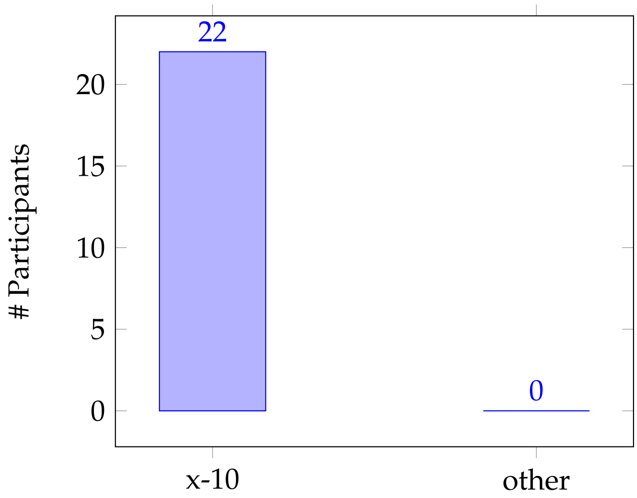


Figure 15 shows that even though most participants couldn't clearly be assigned to a data structure, those that could were matched to the BinarySearchTreeLimitedDepth structure. In figure 16, insert type matchings are shown. Only two participants could clearly be assigned to one insert type. That assignment held for both parts of the problem sets, i.e. they were assigned the same insert type in both those problems containing two premises and one conclusion as well as those containing three premises and one conclusion.
Figure 17 shows that those participants that were not to BinarySearchTreeLimitedDepth (see figure 15 were matched to unlimited data structures, i.e. all but BoundedList. This makes sense considering that this part of the problem set only contained three tokens at maximum, which is presumably an amount of tokens that participants can still hold completely in memory. Therefore, they don't match to limited data structures that might already cap the amount of tokens remembered. This also explains the solely matched activation function $x - 10$, as seen in figure 18. Tokens were simply not plentiful enough to see a forgetting effect in participants.

Three premises and one conclusion
There were 48 problems with three premises and one conclusion. For the test participants, 28.82 problems could be matched on average (mean = 28.82 problems, std = 1.92, highest matched value = 32 problems, lowest matched value = 26).
It was much easier to reliably match a participant to a specific data structure than for the first part of the problem set. The amount of data structures that received the highest vote per participant decreased steeply: On average, only 3.68 data structure received the highest vote per participant (mean = 3.68, std = 2.46). First, those combinations with the highest vote per participant were evaluated. Then, to see which trends were

stable, those combinations with the three highest votes per participants were evaluated together. Of course, the relationships in the analysis of the three highest matches were not as strongly anymore, however some held, which was interesting. Some visualizations of the gleaned data will be shown and discussed in the following.

Figure 19: Three premises and one conclusion - Highest match - Matched data structures

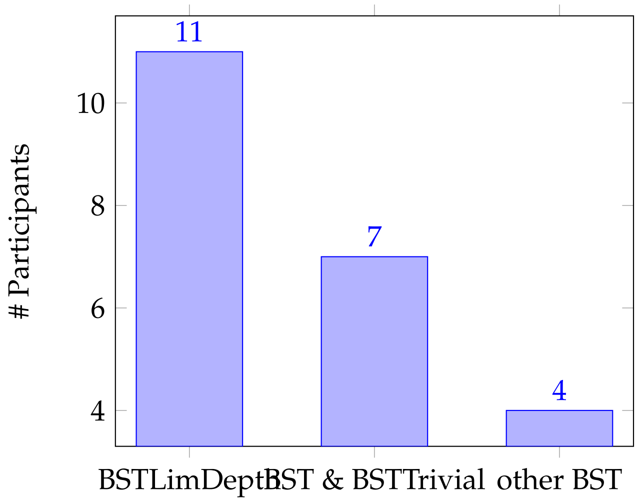


Figure 20: Three premises and one conclusion - Highest match - Matched insert types

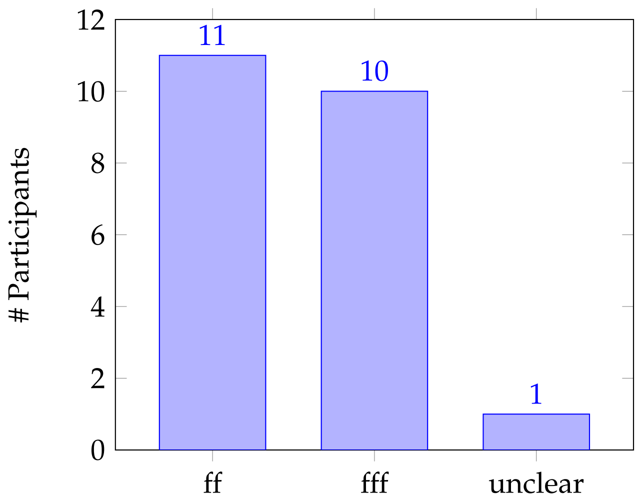


Figure 21: Three premises and one conclusion - Highest match - Matched limits for data structures

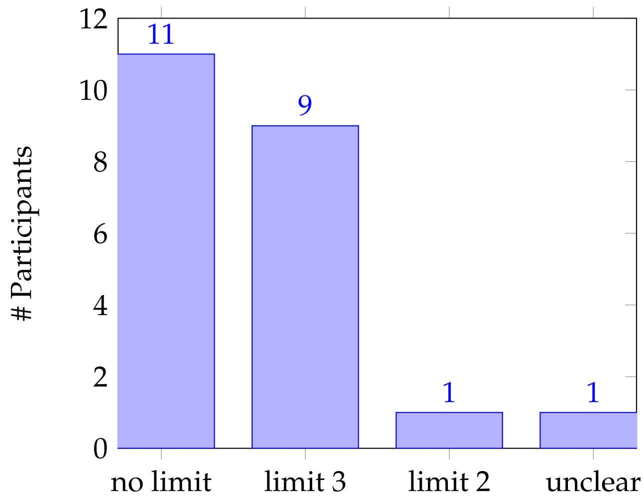


Figure 22: Three premises and one conclusion - Highest match - Matched activation functions

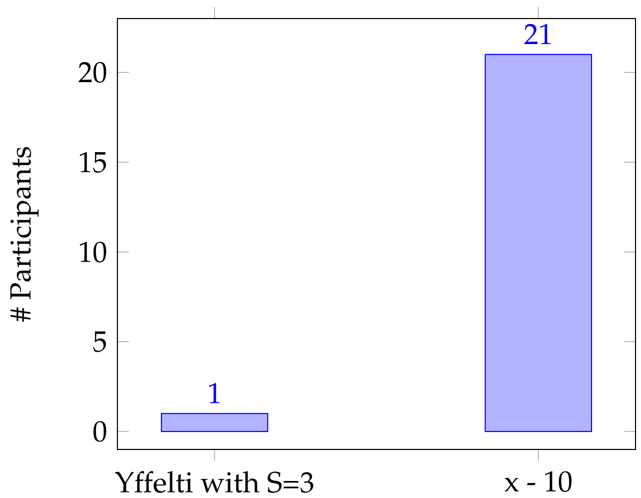


Figure 23: Three premises and one conclusion - Highest match - Matched merge type

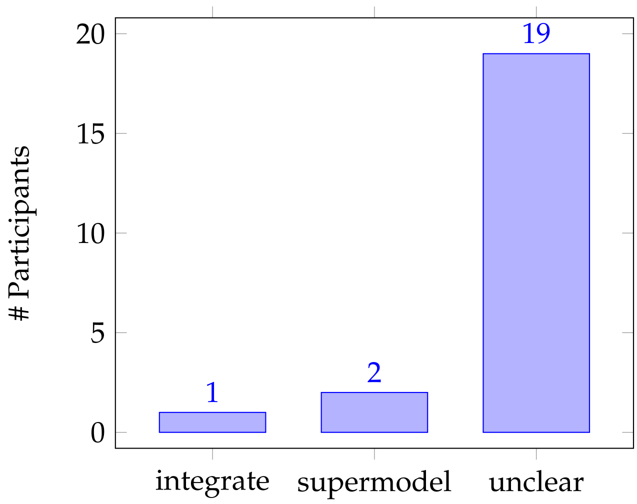


Figure 24: Three premises and one conclusion - Highest match - Merge occurrence

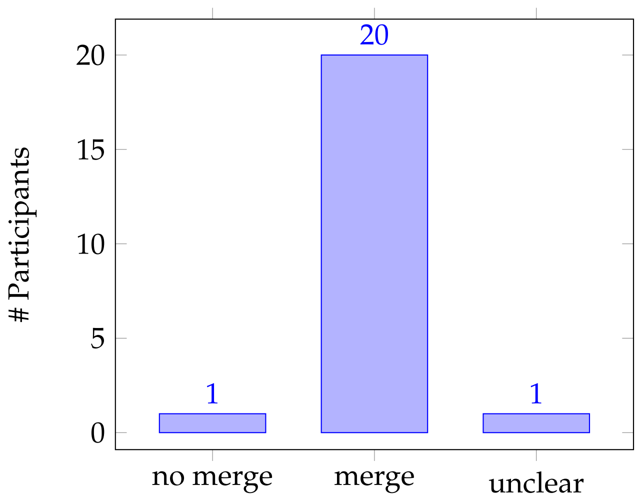


Figure 25: Three premises and one conclusion - Three highest matches - Matched data structures

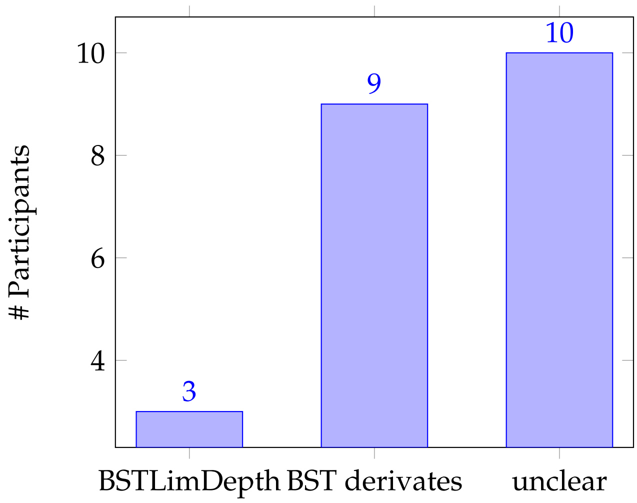


Figure 26: Three premises and one conclusion - Three highest matches - Matched insert types

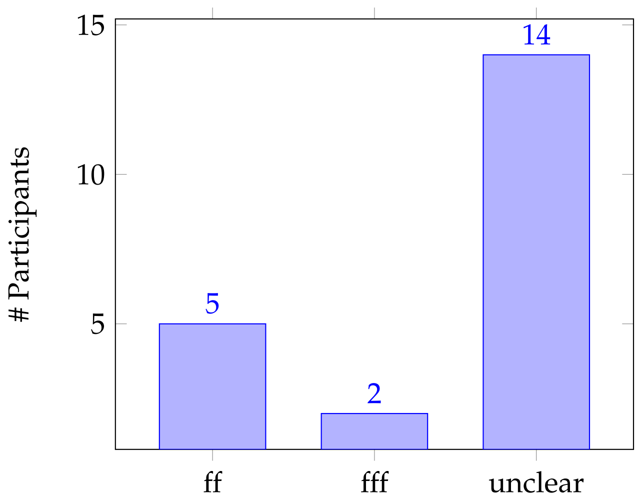


Figure 27: Three premises and one conclusion - Three highest matches - Matched limits for data structures

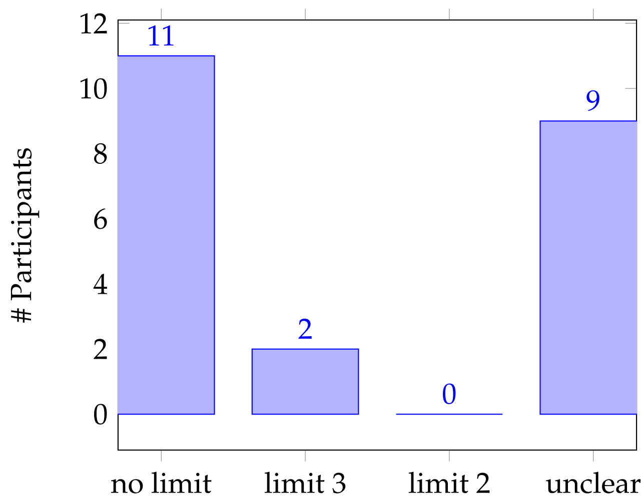


Figure 28: Three premises and one conclusion - Three highest matches - Matched activation functions

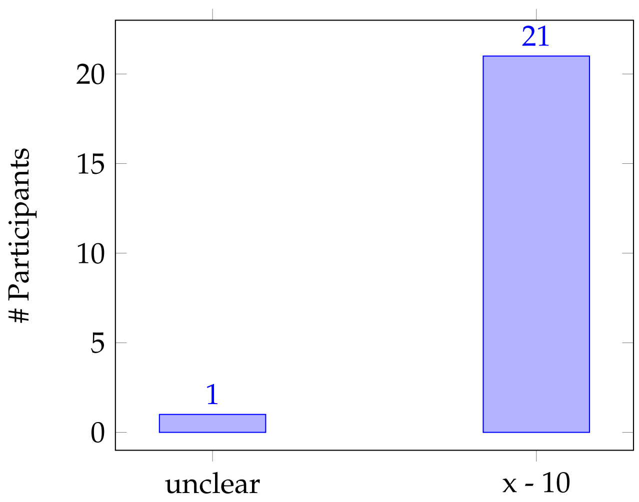


Figure 29: Three premises and one conclusion - Three highest matches - Matched merge type

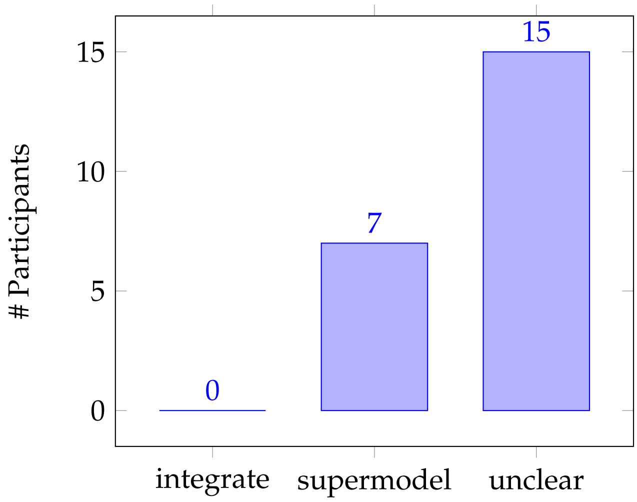


Figure 30: Three premises and one conclusion - Three highest matches - Merge occurrence

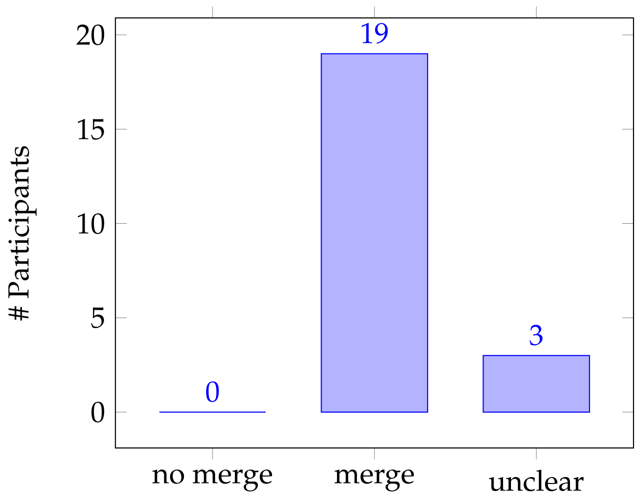


Figure 19 shows the data structures that participants were matched to with their highest match. BinarySearchTreeLimitedDepth is the data structure that was most commonly matched. Figure 21 shows the limits that were assigned to it. It is apparent that that limit stays on the higher side: When considering 3 premises in total, a depth of

3 may in most cases be enough to contain all named tokens. However, when tokens are named continuously, this limit comes to fruition (which is the case for 8 of the 48 problems).
All other matched data structure were also derivatives of BinarySearchTree. BinarySearchTree itself and BinarySearchTreeTrivial were considered together, since their differences didn't matter enough for this problem set to yield different results. Participants that couldn't clearly be matched to either BinarySearchTreeLimitedDepth or BinarySearchTree/BinarySearchTreeTrivial showed an assignment to different BinarySearchTree classes: e.g., half BinarySearchTree and half BinarySearchTreeRandomTree.
Also for their three highest matches most participants could still be matched to a BinarySearchTree data structure, as shown in figure 25. The limits for data structures also veered toward the unlimited side, as shown in figure 27.
Participants' insert strategy could very clearly be assigned for their highest match. Figure 20 shows this matching. About half of participants were assigned to ff or fff each. Only one participant could not clearly be assigned. This is a somewhat different distribution of insert strategies than shown in experiments by Ragni and Knauff (2013), since it veers much closer to an equal distribution.
When considering the three highest matches for participants, as shown in figure 26, this relationship is not as pronounced anymore. Only 7 participants could definitely be assigned to an insert type. One explanation could be that participants do not choose the same insert strategy for every problem that they choose. Even for the best match between participant and combination of data structure, only 32 problems were matched. It could be that e.g. the combinations for the highest match and those for the second highest match match different problems and contain different insert types - the highest match matching those problems that were solved with the ff-strategy, and the second highest match matching those problems that were solved with the fff-strategy, for instance.
For their highest match, one participant could clearly be linked to an activation function that showed forgetting, see figure 22. It was Yffelti (2016)'s function with a memory strength factor of S = 3. Presumably, three premises and therefore four tokens are still not enough to reliably show an activation function at work. However, this first participant clearly linked to one activation function could indicate that three premises are the starting point at which activation functions can start to be matched. For their three highest matches, the assignment doesn't change considerably, see figure 28.
Merge functions could not be matched clearly for the highest match, as shown in figure 23. Only three participants showed a clear preference for one strategy. However, as shown in figure 24, it is apparent that participants did perform a merge operation. Only one participant could clearly be matched to no merge (which would be equal to merge type supermodel and the limit for nested supermodels be set to 0). One participant could not clearly be matched to either an occurred merge or no occurred merge, all other participants clearly performed a merge. It's just not apparent which strategy they used for it.
For the three highest matches, all participants that could clearly be assigned a merge

strategy were assigned the supermodel strategy, see figure 29. It is still obvious that participants did perform a merge, as shown in figure 30, even if the merge strategy can't always be determined.

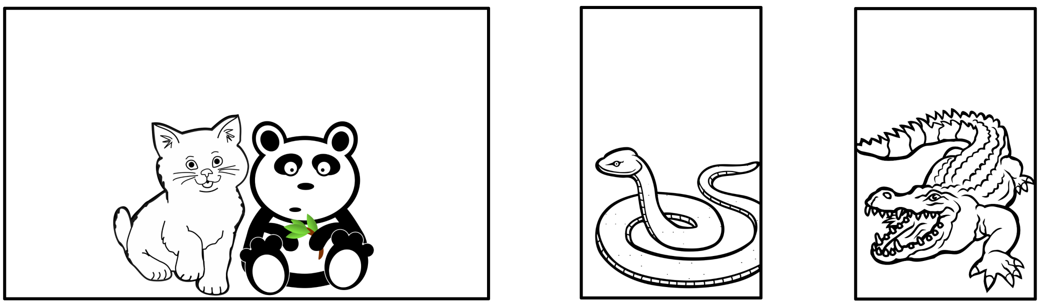
8 General discussion

There seems to be evidence for a left to right bias in humans, i.e. humans prefer to construct a model from the left to the right. This bias may be caused by cultural aspects, such as reading direction. (Chan and Bergen, 2005), (Spalek and H.) I did not take this bias into consideration. It may make problems that presume a right to left direction, e.g. with a premise such as "A right B" harder to solve.

In my strategies, fff- and ff-strategies were chosen once and for all. It may be the case that humans mix these strategies. This possibility was not considered in my implementation.

9 The Cave of Mysterious Monsters Revisited

Thank *you all* for all the chocolate, snickers and listening to diatribe. A particular thanks to Francesca Yffelti and Berta Brzenczeszczkiewicz for their mathematical contribution. A shout-out to the loudest group in the PC pool, thanks for the stone!
Plus, because I've been asked: The mysterious cave from the introduction was built by a fervent logician, but not a fervent biologist. Mistaking a panda for a polar bear was an honest mistake of his. In this case it all works out in your favor: Choosing the leftmost door or the one one position farther to the right won't determine your earthly fate. Both panda and kitten are extremely cuddly. :)



10 Picture Sources

- Figure 1: "Deciding Which Door to Choose 2" by Vic is licensed under CC BY 2.0
- Figures 2 and 3 are composed of the following images:
  - "Contour Drawing of Little Kitten Cat" by Brian Ramirez is licensed under CC BY 4.0
  - "Grizzly Bear Stencil Clipart" by Ryan Sanders is licensed under CC BY 4.0
  - "Outline Snake Clipart in Black and White" by Agnes Farmer is licensed under CC BY 4.0
  - "Stencil Design of Wild Crocodile in Black and White" by Robert Carter is licensed under CC BY 4.0
- Figure 4 is composed of the following images:
  - "Cat Outline Logo Drawing Design" by Carl Powell is licensed under CC BY 4.0
  - "Monkey Outline Clipart in Black and White" by Paul Foster is licensed under CC BY 4.0
  - "Outline Bear Drawing" by Emily Bennett is licensed under CC BY 4.0
  - "Drawing Outline of Horse in Black and White" by Frank Lewis is licensed under CC BY 4.0
- Figures 5, 6, 7, 8, 9, 10, 11 and 12 were made/are screenshots by me.
- Figure 9 is composed of the following images:
  - "Contour Drawing of Little Kitten Cat" by Brian Ramirez is licensed under CC BY 4.0
  - "Panda Bear Clipart with Branch in Hand" by Patricia Wood is licensed under CC BY 4.0
  - "Outline Snake Clipart in Black and White" by Agnes Farmer is licensed under CC BY 4.0
  - "Stencil Design of Wild Crocodile in Black and White" by Robert Carter is licensed under CC BY 4.0
- The reproduction of figures 13 and 14 was generously allowed by authors of Yffelti (2016).

Bibliography

A. D. Baddeley. *Working memory, thought, and action*, volume 45 of *Oxford psychology series*. Oxford Univ. Pr, Oxford u.a., 1. publ edition, 2007. ISBN 0-19-852801-9.

T. T. Chan and B. Bergen. Writing direction influences spatial cognition. In *Proceedings of the Twenty-Seventh Annual Conference of the Cognitive Science Society*, pages 412–417, 2005.

H. Ebbinghaus, H.A. Ruger, and C.E. Bussenius. *Memory: A Contribution to Experimental Psychology*. Columbia University. Teachers College. Educational reprints. Teachers College, Columbia University, 1913.

G. P. Goodwin and P. N. Johnson-Laird. Reasoning about relations. *Psychological review*, 112(2):468–493, 2005. ISSN 0033-295X. doi: 10.1037/0033-295X.112.2.468.

G. Jahn, M. Knauff, and P. N. Johnson-Laird. Preferred mental models in reasoning about spatial relations. *Memory & Cognition*, 35(8):2075–2087, 2007. ISSN 1532-5946. doi: 10.3758/BF03192939.

P. N. Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness*. Harvard University Press, Cambridge, MA, USA, 1983. ISBN 0-674-56882-6.

P. N. Johnson-Laird and R. M. J. Byrne. *Deduction*. Essays in cognitive psychology. Erlbaum, 1991. ISBN 9780863771484.

M. Knauff, L. Bucher, A. Krumnack, and J. Nejasmic. Spatial belief revision. *Journal of Cognitive Psychology*, 25(2):147–156, 2013. doi: 10.1080/20445911.2012.751910.

A. Krumnack, L. Bucher, J. Nejasmic, B. Nebel, and M. Knauff. A model for relational reasoning as verbal reasoning. *Cogn. Syst. Res.*, 12(3-4):377–392, September 2011. ISSN 1389-0417. doi: 10.1016/j.cogsys.2010.11.001.

G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

M. Ragni and M. Knauff. A theory and a computational model of spatial reasoning with preferred mental models. *Psychological review*, 120(3):561–588, 2013. ISSN 0033-295X. doi: 10.1037/a0032460.

R. Rauh, C. Hagen, M. Knauff, T. Kuss, C. Schlieder, and G. Strube. Preferred and alternative mental models in spatial reasoning. *Spatial Cognition & Computation*, 5 (2-3):239–269, 2005. doi: 10.1080/13875868.2005.9683805.

W. Schaeken, J-B. Van der Henst, and W. Schroyens. The mental models theory of relational reasoning: Premises' relevance, conclusions' phrasing and cognitive economy. In W. Schaeken, A. Vandierendonck, W. Schroyens, and d'Ydewalle. G., editors, *Mental models: Extensions and refinements*, chapter 7, pages 127–150. Lawrence Erlbaum Associates, Abingdon, 2007.

T. M. Spalek and Sherief H. The left-to-right bias in inhibition of return is due to the direction of reading. *Psychological Science*, 16(1):15–18. doi: 10.1111/j.0956-7976.2005.00774.x.

Wikipedia. Transitive relation — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Transitive2016. [Online; accessed 19-August-2016].

F. Yffelti. Simple mathematical model of successive forgetting, 2016.