decheem Documentation

Release latest

Dec 09, 2021

User Documentation

1 What is the deCheem Explorer? 1

2 How do I use the deCheem Explorer? 3

2.1 Theory behind deCheem 3

2.2 User Guide 8

2.3 deCheem rule writing guide 10

2.4 Frequently Asked Questions 12

CHAPTER 1

What is the deCheem Explorer?

deCheem Explorer is a web-app that uses the deCheem framework to provide the most efficient way for managing beliefs, knowledge and argumentation.

By extending our human limitations to deduction and memory, it allows us to apply a same level logical rigor to philosophical, legal or social science reasoning that has previously only been seen in the field of computing, mathematics and science.

CHAPTER 2

How do I use the deCheem Explorer?

The first step of any deCheem exercise is to use the *Belief-base editor* to build the belief system you want to analyse. You do so by documenting beliefs about relationship between situations in the discussion that you are trying to work on.

Once that's done, you can use the *Explorer* to explore the implications of such a belief system in various specific situations. Explores are designed to be *specific* and *independent*, allowing one to analyse very specific situations without worrying about any 'cross-contamination/confusion' between arguments and situation.

Take a look at this sample deCheem belief-base to find out how deCheem can be used in comparative studies of different belief systems and shows where exactly the disagreement is.

Please click on the Intro section below to learn more about the theory behind deCheem:

2.1 Theory behind deCheem

2.1.1 The parts of deCheem

To perform any deCheemic analysis, you will need to make use of three things:

- the deCheem Inference Engine
- the deCheem Belief Language
- a user interface.

The deCheem Inference Engine is makes use of set-theory for reasoning. It imagines the world as the universal set of situations, and beliefs as rules that asserts claims about the existence or non-existence of subsets of these situations.

The deCheem Belief Language is designed to be as close to natural language as possible. It eschews all forms of data type definitions in order to fully expose the expressive power of natural language. This is why deCheemBL statements reads like human language, and is also interpreted computationally exactly like how it reads in human language.

User interfaces uses deCheemBL to interact with instances of the deCheem IE in the backend, allowing one to explore belief systems intuitively without having to deal with any complicated syntax or code.

2.1.2 Sets vs Always, Never and Possibly.

deCheem operates on the belief that all beliefs can be constructed out of the subject(s) and statement(s) about the subject(s). Statements are about whether a situation applies, not applies or perhaps applies (Possibly, mostly, probably, often...) to a situation. deCheem encourages users to use 'Always' and 'Never' as modals in their statements, and pushes users to figure out why they are hesitant to make hard statements about the 'Always'ness or 'Never'ness of a situation.

By interpreting what 'Always' or 'Never' says about the existence of subsets of situations in the world, deCheem is able to help you derive the implications of any argument in this belief system.

This file here illustrates how deCheem statements are used to filter out sets of situations out of the set of all possible situations. Sheet A shows how 6 different beliefs translate to statements about the (non-)existence of certain subsets of situations. Sheet B shows 3 beliefs overlay to form a final world view, which can be 'explored' to find out what the implications are in each explore.

A live demo of deCheemBL, the deCheem Inference Engine and a suitable user-interface can be found here.

2.1.3 deCheemBL samples

Here are examples of how deCheem beliefs are stored and queried/explored in order to get the inference results we need.

Belief Base sample

The following is an example of how a belief base with 6 beliefs is stored in a data base.

You are free to store it in any database type you want as long as the object structure below is respected when the belief base is stored and retrieved.

```
{
  "then": [
    {
      "case": {
        "type": "LET",
        "modalPhrases": [
          {
            "modal": "Always",
            "properties": {
              "The world is in a state of war.": true
            }
          }
        ],
        "filterPhrases": [
          {
            "The world is in the state of nature.": true,
            "The individual in question is rational.": true,
            "The resources in the world is limitless.": false,
            "The philosophy of Thomas Hobbes is being considered.": true
          }
        ]
      },
      "beliefUniqueId": "426aa178-be38-432f-b9a5-9ab35e09d4ef"
    },
    {
      "case": {
        "type": "LET",
        "modalPhrases": [
```

(continues on next page)

(continued from previous page)


```
          {
            "modal": "Always",
            "properties": {
              "The collective in question is rational.": true
            }
          },
          {
            "modal": "Always",
            "properties": {
              "Force is deployed to bring about peace.": true
            }
          }
        ],
        "filterPhrases": [
          {
            "The world is in a state of war.": false,
            "The philosophy of Thomas Hobbes is being considered.": true
          }
        ]
      },
      "beliefUniqueId": "134c21d4-7005-4bdc-9ddc-e39991223328"
    },
    {
      "case": {
        "type": "LET",
        "modalPhrases": [
          {
            "modal": "Always",
            "properties": {
              "The individual's liberties are constrained by their innate morality.":␣
↪true
            }
          }
        ],
        "filterPhrases": [
          {
            "The individual has liberties.": true,
            "The world is in the state of nature.": true,
            "The philosophy of John Locke is being considered.": true
          }
        ]
      },
      "beliefUniqueId": "c1667128-8f29-4c29-82dc-55af15127a3b"
    },
    {
      "case": {
        "type": "LET",
        "modalPhrases": [
          {
            "modal": "Always",
            "properties": {
              "The world is in a state of war.": false
            }
          }
        ],
        "filterPhrases": [
          {
```


(continues on next page)

(continued from previous page)

```
            "The resources in the world is limitless.": true,
            "The philosophy of John Locke is being considered.": true
          }
        ]
      },
      "beliefUniqueId": "c6ac2033-905d-4e8b-bcb8-24b6c4c19b0c"
    },
    {
      "case": {
        "type": "LET",
        "modalPhrases": [
          {
            "modal": "Always",
            "properties": {
              "The collective is in agreement on the methods of punishment and␣
↪enforcement.": false
            }
          }
        ],
        "filterPhrases": [
          {
            "The world is in a state of war.": true,
            "The world is in the state of nature.": true,
            "The philosophy of John Locke is being considered.": true
          }
        ]
      },
      "beliefUniqueId": "791ea8f0-4779-4d64-b8b7-478bdf2b979e"
    },
    {
      "case": {
        "type": "LET",
        "modalPhrases": [
          {
            "modal": "Always",
            "properties": {
              "The collective in question is rational.": true
            }
          }
        ],
        "filterPhrases": [
          {
            "The collective is in agreement on the methods of punishment and␣
↪enforcement.": true
          }
        ]
      },
      "beliefUniqueId": "120dc259-2033-47b9-a5f3-cb316e4b883f"
    }
  ],
  "beliefBaseName": "Political philosophy demo",
  "beliefbaseowner": "guangmian@gmail.com"
}
```

Explore query

You submit a request to the deCheem Inference Engine in order to infer what the implications of an initial situation will be. The deCheem Inference Engine takes two arguments: the deCheem Belief Base object as show above, and the

simple object below representing the situation you want to explore.

The actual code of the deCheem Inference Engine will be made public at a later stage.

```
{
  "The philosophy of Thomas Hobbes is being considered.": true,
  "The philosophy of John Locke is being considered.": true,
  "The world is in the state of nature.": true,
  "The resources in the world is limitless.": true,
  "The individual's liberties are constrained by their innate morality.": false
}
```

Explore results

After the Inference Engine has received the Explore object and the Belief Base to base the inference on, it processes the situation and returns a result object like the one below:

```
{
  "resultCode": "Success",
  "resultReason": "Successful with no errors found",
  "exploreResults": {
    "possible": true,
    "reasoningSteps": [
      {
        "deducedProperty": {
          "The individual has liberties.": false
        },
        "sourceBeliefId": "c1667128-8f29-4c29-82dc-55af15127a3b"
      },
      {
        "deducedProperty": {
          "The world is in a state of war.": false
        },
        "sourceBeliefId": "c6ac2033-905d-4e8b-bcb8-24b6c4c19b0c"
      },
      {
        "deducedProperty": {
          "The collective in question is rational.": true
        },
        "sourceBeliefId": "134c21d4-7005-4bdc-9ddc-e39991223328"
      },
      {
        "deducedProperty": {
          "Force is deployed to bring about peace.": true
        },
        "sourceBeliefId": "134c21d4-7005-4bdc-9ddc-e39991223328"
      }
    ]
  }
}
```

This result object can then be used to power all kinds of visualisations and logic on the frontend of your application. Once again, an example showing this belief base in action can be seen here.

2.2 User Guide

2.2.1 Getting started

deCheem is a tool for sketching and testing rule/belief systems, and the layout is meant to help you with this process. deCheem was designed to allow everyone to start sketching and testing ideas and rules within a few minutes. deCheem is a tool for sketching and testing rule/belief systems, and the layout is meant to help you with this process. If you need some help to get started, here are some instructions for you!

Creating a deCheem account

1. To create an account, go to app.decheem.io and click on 'Sign Up'
2. Sign up either with your Google Account or create a fresh user name and password for deCheem.
3. Once done, you will be redirected to the working view, where you can start creating and testing your first rule systems.

Navigating deCheem

1. If it is your first time logging into deCheem, the first thing to do is to click on the big grey box on the left to create a new rule system.
2. deCheem is a tool for sketching and testing rule/belief systems, and the layout is meant to help you with this process.
    1. The basic building block in deCheem is simply a basic English sentence, which can be either **true** or **false** (basically a proposition).
    2. Sentences can then be used to build If-Then **Rules**.
    3. A collection of rules forms a **Rule System**, which can then be explored, analysed and tested.
3. Sentences (describing a situation) can be entered directly as part of a rule or simply pinned at the left side under 'Pinned sentences', until you figure out how to fit it into a rule.
    1. Note that sentences should read grammatically correct both standalone (e.g. **The person is guilty**) and as part of a rule (e.g. If the evidence is found then **the person is guilty**)
4. Once you have at least one rule, by default you should see nodes appearing in the graph on the right. This shows the **correlation** between various situations/ideas (aka sentences) and rules.
5. If you want to test and analyse the **causation** between various situations instead, selec 'Causation analysis' at the top-right instead.
    1. In this view, you can explore hypothetic situations described by any combination of sentences (true or false) mentioned in your rule system.
6. Whenever you want to save your rule system, just click on the save icon at the top.

User rights and permissions

1. deCheem utilises a graph-based user permission system, which basically means that instead of managing access rights through a static and hierarchical folder structure, it allows for any combination of user groups to fit your organisational needs.
2. There are three main kinds of permission entities in deCheem:

1. **Users** - Like the one you used to log into deCheem, which is represented by your email address.
2. **Rule Systems** - These are the rule systems that you create through the interface. They can be accessed by Users and/or User Groups.
3. **User groups** - These are entities that either users or other user groups groups can belong to. User groups also serve as folders in deCheem for grouping Rule Systems together.

3. Permissions for Rule Systems and User Groups are managed the same way.
    1. **User groups** can be created, viewed and managed by doing a mouse-over on your user avatar at the top right and clicking **Manage User Groups**.
    2. **Rule System** permissions can be viewed and managed by clicking on the permission management icon in the top right of the Rule System Editor.
4. In the Permission Management Overview, you will see on the left all the entities that have access to the entity you are examining. On the right you will see all the entities that it has access to (only applicable if you are managing User Groups).
5. To add permissions, click on the '+' sign and grant access by any User (Groups) to any User Group or Rule Systems.
6. To remove permissions, click on the '-' sign next to the entity on the Permission Management Overview to revoke its access.

2.2.2 Features for education

deCheem has several features that are meant to allow philosophy/law educators and students to work together more effectively.

deCheem Rule Systems (in combination with deCheem causation analysis test cases) could easily be used to replace essays arguing a certain standpoint, or to show where various systems of belief concur and disagree.

If you - as an educator - prefer to offer more guided assignments, deCheem also allows you to create 'Rule Assembly' assignments that help beginning students quickly understand the process of distilling ideas and rules from existing literature.

For teachers: Creating 'Rule Assembly' assignments

1. A Rule Assembly assignment is basically a **mix-and-match** exercise where the teacher determines the set of possible sentences to use, and the students have to use them to recreate the rules the teacher had in mind.
2. **Imagine you are a teacher of law and you:**
    - want your students to read 5 different past cases and distill the rules that led to the judgments in these cases.
    - have created the Rule System yourself and you want your students to try to recreate it, but want to set boundaries of the phrasings that the students can use for ease of marking.
    - want to programmatically mark the work of your students without having to read through every single essay painstakingly.
3. **To prepare such an assignment, you just need to do the following:**
    1. Create the Rule System that contain all the rules that you want your student to pick out from the past cases. Save this properly, as it will serve as your **marking scheme**.
    2. In the Rule System selector on the top left, click and duplicate the rule system, and choose to make a 'disassembled' copy of the rule system.

3. Share this disassembled copy with the User Group where your students are in. Grant only read-only rights so that students have to make a personal copy of it before working on it.
4. Create a new User Group as a submission folder (see instructions in previous section), and copy down the Submission Code at the bottom of the permission overview of the User Group.
    - Grant other teachers and teaching assistants access to this group so they can help you with reviewing the assignments later. Do not add your students to this group.

4. When communicating the assignment to your students, point them to the disassembled copy you shared with them, as well as the Submission Code for them to use once they are done.

For students: Submitting Rule Systems

1. All assignments in deCheem involve the creation of a Rule System, regardless of whether you were giving a disassembled rule system to start with or simply a blank slate to work from.
2. Once you have created and saved the rule system, go to the Permission Overview of the Rule System, and click on 'Submit this rule system'.
3. In the pop-up, enter the Submission Code given to you by your teacher, and adjust the submission name of the Rule System to fit the format given to you by your teacher.
4. Once done, click Submit. That's it, you have just submitted a timestamped and unlinked copy of your rule system to your teacher.

For teachers: Marking 'Rule Assembly' assignments

1. Once all your students have used the Submission Code to submit the assignment, you can mark all of them at one go using your marking scheme.
2. Click on the star icon at the top of the page to open up the deCheem scoring interface.
3. In the first dropdown, select the Rule System that you want to use as the marking scheme.
4. In the second dropdown, select the User Group that is linked to the Submission Code you gave to your students. Once done, all the submitted rule systems will be listed in the table below.
5. Click on 'Go!' to grade all these Rule Systems according to the marking scheme.
6. Each deCheem Rule is broken down and translated into a certain number of assertions in the background. deCheem then uses these assertions to score the students' Rule Systems on two measures:
    - **Completeness** - percentage of the marking scheme's assertions that are also made by the student's rule system.
    - **Correctness** - percentage of the total set of assertions implied by the student's rule system that are also made by the marking scheme.
7. In the last column of the interface, you will see an average score of the Completeness and Correctness of the rule system.

2.3 deCheem rule writing guide

Here you will find a number of best practices on how to phrase your rule in order to build the most scalable rule system possible.

2.3.1 How do I construct a deCheem rule?

Entering new deCheem rules is simple: you just need to phrase your rules in terms of IF-THEN statements. In these statements, you first describe 'situations', then use the words 'is', 'is possibly' or 'is not' to add what the situation should or should not be.

Here are a few examples:

Diabetes and sugar

- **Regular speech:**
  - People who have diabetes should eat less sugar
- **deCheem rule:**
  - If we are in a situation where **the person has diabetes** then it **is** a situation where **the person should eat less sugar**.

Coconut trees and climate

- **Regular speech:**
  - If the climate is cold, people should not be trying to plant coconut trees
- **deCheem rule:**
  - If we are in a situations in which **the climate is cold** then it **is not** a situation where **the person should be trying to plant coconut trees**.

2.3.2 Write rules with clear and limited scopes

deCheem analyses are most useful when specific cases can be used to derive very advice about other specific situations. In the deCheem world, general rules are derived from the exceptions. This means that the more specific and concise you write your rule, the more useful they become. Why is this so? The more specific the scope of a rule (provided it's correct), the higher the chance that it will not over-generalise and end up making an incorrect generalisation about a certain kind of situation.

A few guidelines:

Make rules narrow but deep

Use words like "the" and "in question" to narrow the scope of your rule, and refer to subjects singularly instead of in plural, as using the latter increases the chance of over-generalisations.

Sentence 1: If we are in a situation where **people are unhappy** then it **is** a situation where **work is not good**.*

> 'People' and 'Work' are used in a way that's too generic here. 'Work' here could mean an artwork, labour of a person or the work in the physics sense.

Sentence 2: If we are in a situation where **the person is unhappy** then it **is** a situation where **the work of the person is not good**.

> By introducing 'the' and transforming plural to specific singular (e.g. 'people' to 'person'), the rule lends itself to being used in more situations without risk of generalisation. However one question remains: which person am I talking about?

Sentence 3: If we are in a situation where **the person in question is unhappy** then it **is** a situation where **the work of the person in question is not good**.

> Using the words "in question" to denote the person we are talking about this very moment, we distinguish it from the 'abstract person', allow us to introduce phrases like "the person in question is Bill Gates" in other rules that will help sharpen the identity of the person we are talking about.

2.3.3 Using 'And' right

Using 'And' right in deCheem is essential to writing rules that behave correctly.

The following two rules sound similar, but actually can mean very different things:

'AND' within a situation

> If we are in a situation where **the cat is blue, furry** then it **is** a situation where **the cat is friendly**.

. . . versus. . .

'AND' between situations

> If we are in a situation where **the cat is blue** and or **the cat is furry** is **Always** a situation where **the cat is friendly**.

The first one implies that only when a cat is *both* blue *and* furry is it also friendly (meaning a cat that is only blue and not furry is not friendly).

The second one implies that when a cat is *either* blue *or* furry, it is also friendly.

Likewise, the same applies when 'and' is applied to the second part of the rule:

'AND' within a situation Situations in which **the cat is red** is **Never** a situation where **the cat is friendly, cute**.

. . . versus. . .

'AND' between situations

> Situations in which **the cat is red** is **Never** a situation where **the cat is friendly** and **Never** a situation where **the cat is cute**.

The first one implies that when a cat is red, it is not friendly *only when* it is also not cute, and not cute *only when* it is not friendly.

If the cat is red and cute, it could still be either friendly or not friendly.

The second one implies that when a cat is red, it will definitely be *neither* cute *nor* friendly.

2.4 Frequently Asked Questions

2.4.1 What are the most important characteristics to look for in the ideal expert system for managing legal or philosophical knowledge?

Here are a list of principles that were taken into account during the design process of deCheem.

- **Type-less knowledge entry**
  - A word or phrase can mean different things in different contexts (e.g. club as in golf club or disco club), so any form of 'type' declaration (in the broadest sense of the word) will inevitably limit the ability of that word to take on different meanings (and hence roles) in future situations.
- **Sequence-insensitive knowledge acquisition**
  - Expert systems are only considered useful if it can produce accurate advice faster and cover knowledge areas larger than any human can.
  - To achieve this, acquiring and assimilating the knowledge from a large number of sources is essential. To enable large scale knowledge aggregation to take place efficiently, it is crucial that the sequence in which knowledge is entered into the system has no effect on it's usefulness later on.
- **Deterministic**

  - Probabilistic methods (e.g. text-clustering, correlation-based) do not have a lot of use in philosophical and legal settings, as saying that something is 'maybe' or 'likely' true does not allow one to settle an argument completely.
  - It is therefore important that the framework deduces conclusions, rather than makes a guess of what it is (regardless of the likelihood that it is correct).

- **Syntax-less knowledge entry**
  - For expert systems to be truly useful in the fields of law and philosophy, it should not enforce specific syntax that the user needs to learn in order to document their knowledge. Not only does specific syntax decrease the usability of the system, it also can be seen as a form of 'typing', which has the downsides described above.

- **Directionless reasoning**
  - Forward and backward chaining are functionalities often looked at in expert systems, but any system that predefines types for reasoning/chaining will eventually be insufficiently generic for philosophical and legal use cases.

- **Implicit deduction**
  - A good expert system should be able to read what is explicitly mentioned in piece of documented knowledge, but also what is implicit. This allows a single statement to be more than just a declaration of a single relationship, but of all possible logically equivalent permutations of that relationship.

2.4.2 How does deCheem deal with forward and backward chaining?

deCheem intentionally does away with the technical distinction between these two forms of 'chaining'. In the spirit of 'typelessness', deCheem instead has just one way of exploring situations, without taking into account if it's the desired starting or ending situation.

The degree to which a particular explore is doing 'forward', 'backward' (or mixed??) chaining is determined by the human language used to describe those situations.

2.4.3 What is the best way to phrase a belief?

Refer to the style guide on this documentation website for extension guidelines.

2.4.4 How does deCheem represent beliefs about different subjects without using conditional/IF-ELSE statements?

IF-ELSE statements are needed when relating between two different subjects in terms of graphs/relations. In a set-based expert system like deCheem, this can be bypassed by phrasing everything as a property of a common subject. 'Situations' is the best choice of subject here, as it keeps everything in the same set-space by allowing anything under the sun to be phrased in terms of situations. This allows us to achieve conditionality without resorting to IF-ELSE clauses, and thus also achieves sequence-insensitivity at the same time.

2.4.5 Do relational databases fulfil the deCheem design principles?

Relation databases do not perform inference out of the box. If one were to (mis)use them for performing deduction, analyse 20 different possible situation types would need a table with 2^20 rows. This is unscalable and is not what relational databases should be used for.

2.4.6 Do graph databases fulfil the deCheem design principles?

Graph databases sees things as nodes with fixed relationships. deCheem forms relationships between different nodes based on certain conditions, and the inference engine layer is not native to graph databases.

2.4.7 Do decision-trees fulfil the deCheem design principles?

Decision trees are by nature hierachical and operates on branches. If an idea in a deep branch has links to another idea in an earlier branch, there is no efficient way to represent that relationship. Also, if the definition of a decision point at an earlier branch is changed, the validity of the decisions branches lower down will all be affected, which limits the maintainability of this solution.

Frameworks that share the same method and therefore the same pitfalls when used as philosophy and legal expert systems are:

- Decision Model and Notation (DMN)
- Argument-maps

2.4.8 Does Prolog fulfil the deCheem design principles?

Prolog is great for quantitative inferences and relationship deduction when properties share only inherit properties from a single parent. However, numerical methods are useless against analysis of beliefs, and the need for beliefs to take on different meanings (aka inherit properties) from any number of situations makes Prolog a bad choice to use for belief analysis. Prolog makes a distinction between 'rules' and 'facts', and that distinction takes away from the 'type-less' nature of a good general expert system.

2.4.9 Does the Carneades system fulfil the deCheem design principles?

When it comes to how knowledge is represented, the Carneades argumentation system is one of the closest to the deCheem belief language. Subjects and predicates are represented together in 'statements' (belief properties in deCheem's terms), which is one step closer to true 'typelessness'. Carneades also represents only relations between statements in a single direction, while deCheem does that but also allows statements to have true modality (e.g. represent assertions that are true in all cases/directions).

However, when it comes to how conclusions are generated (aka the inference engine), Carneades takes a graph-based approached (e.g. linking nodes to each other through edges) while deCheem goes for a set-based approach. Graphs are meant to show (cor)relation, and it can at best only deal with forward-chaining use-cases, and only for the situations that have been explictly documented either in part or full. deCheem does away with directionality altogether thanks to it's set-based approach, and also allows for deduction of all possible implicit conclusions.

2.4.10 Why are OWL or RDF-based formats not used for representing beliefs or statements in deCheem?

OWL Web Ontology Language and RDF make heavy use of object properties and relationship declarations (e.g. subClassOf, oneOf, childOf) to represent information, which takes away from the typelessness that deCheem trieds to strive for.

Other formats or frameworks that share the same pitfalls are :

- LegalRuleML
- Protége.

2.4.11 Does AceRules fulfil the deCheem design principles?

AceRules attempts to make rule entry very similar to typing regular English, which is admirable. However, it's strength but also its pitfall lies in its use of Attempto Controlled English as its foundation. As Attempto Controlled English restricts the set of standard English that can be used, it already introduces a limit to the kinds of ideas or relationships that we can express with it. Furthermore, AceRules requires relation types to come from a predefined list, which also takes away from the goal of typelessness.

2.4.12 What can deCheem not deal with (natively)?

Working with beliefs with a temporal or numerical nature is possible, but cumbersome without tools to help generate the arbitrary number of beliefs needed to cover temporal or numeric graduations.

deCheem does allow for plugins to be added on top as pre-processors, allowing data input sources that accepts graduated input (e.g. time, amounts, decimals) to be translated to input congruent with the set-based system of deCheem.

2.4.13 How do you compartmentalise belief-systems in deCheem?

If you want to categorise belief-systems based on their provenance, simply add that as an additional description of the situation.

2.4.14 Is deCheem a NLP project?

Nope. deCheem neither is nor aims to do Natural Language Processing in any way at this stage. NLP plugins are however possible on top of the deCheem framework to allow for beliefs to be generated in a much quicker way than human input.

2.4.15 Why can't deCheem automatically solve all confusion in conversations?

deCheem shifts the complexity of reasoning away from code and mathematics and into the realm of language. While this grants it enables the user to utilise any corner of his/her vocabulary, it cannot help the user extend or correct his/her vocabulary.

2.4.16 Why doesn't deCheem use any form of weighting?

Using weightage/votes to determine the correctness of a belief is fundamentally against the idea of deCheem, which is to use logical deduction to arrive at facts about our world. If you find yourself struggling with the correctness of a certain belief, think about a specific subset of situations with this belief that you for sure is correct, and document that instead.

2.4.17 Since 'not good' is not necessarily 'bad', how can things be binary?

Indeed, 'not good' is not the same as 'bad', just like 'not hot' is not necessarily 'cold'. deCheem leaves it to the user to determine what the opposite of each situation is, be it as a new situation or simply the negation of the former.

2.4.18 How do you deal with 'scales of things' or 'rankings' or 'priorities'.

'Scales' have similar pitfalls to weightage - something has to be more important than everything else, and when it's not, then something else is. This means that any arbitrary ranking-list can be expressed as a series of beliefs about the utmost importance of a certain thing under certain circumstances.

2.4.19 How efficient is deCheem in dealing with large numbers of beliefs and arguments?

deCheem uses set-theory to perform deduction, meaning it is able to perform implicit deduction without the computational overhead of generating 2^n scenarios or relationships in order to achieve it.

The limitations on the number of beliefs that can be in a belief base is only limited by the RAM available on your computer. As for the computability of the inference, deCheem can generate arguments with up to 500000 chained implications easily on any modern laptop.