UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Formal Analysis of Reddit Network Interactions

DEGREE FINAL WORK

Degree in Computer Engineering

Author: Padró Ferragut, Cristina
Tutor: Alpuente Frasnedo, María
Experimental director: Sapiña Sanchis, Julia

Course 2020-2021

Resum

Amb l'augment de popularitat de les plataformes de xarxes socials en les darreres dècades, es demana una comprensió més exhaustiva i formal del funcionament d'aquestes xarxes a causa de la rapidesa amb què es difon l'informació en aquestes xarxes. Les operacions d'aquestes xarxes haurien de ser consistents i correctes i amb el mínim d'incongruències possibles per mantindre la informació accessible i clara. Reddit, la plataforma escollida per a aquest projecte, ha augmentat en popularitat en els darrers anys i, tot i que no és la plataforma més utilitzada al mercat, és una de les xarxes més potents pel que fa a la difusió de la informació gràcies a la seva natura de dividir per categories. Aquest treball es centra en analitzar les propietats i característiques de la plataforma de xarxes socials Reddit. En primer lloc, es desenvolupa un model formal per a Reddit en el llenguatge d'especificació d'alt rendiment Maude. "Després d'això, es realitza una anàlisi d'accessibilitat al sistema Maude, que garanteix la seguretat del sistema demostrant que no es pot assolir cap estat considerat insegur. Finalment, es comproven les propietats de la vivència mitjançant el comprovador de models lògics (LMC) de Maude LTL."

Paraules clau: Maude, Mètodes Formals, Model Checking, Reddit, Verificació del codi

Resumen

Con el aumento de la popularidad de las plataformas de redes sociales en las últimas dos décadas, existe una demanda de una comprensión más completa y formal de cómo operan estas redes debido a la rapidez con que se difunde la información en estas redes. Las operaciones de estas redes deben ser consistentes y correctas y con la menor cantidad de inconsistencias posibles para mantener la información accesible y clara. Reddit, la plataforma elegida para este proyecto, ha ganado popularidad en los últimos años y, aunque no es la plataforma más utilizada en el mercado, es una de las redes más poderosas en cuanto a permitir que la información se difunda gracias a su naturaleza de división por categorías. Este trabajo se centra en analizar las propiedades y características de la plataforma de redes sociales Reddit. En primer lugar, se desarrolla un modelo formal para Reddit en el lenguaje de especificación de alto rendimiento Maude. "Posteriormente, se realiza un análisis de accesibilidad en el sistema Maude, que garantiza la seguridad del sistema al demostrar que no se puede alcanzar ningún estado considerado inseguro. Por último, las propiedades de vivacidad se comprueban utilizando el verificador de modelo lógico (LMC) de Maude LTL."

Palabras clave: Maude, Métodos Formales, Model Checking, Reddit, Verificación del código

Abstract

With the rise in popularity of social media platforms in the last couple of decades, there is a demand for a more thorough and formal understanding of how these networks operate due to how quickly information is disseminated in these networks. The operations of these networks ought to be consistent and correct and with as few inconsistencies as possible to keep information accessible and clear. Reddit, the platform chosen for this project, has risen in popularity in the last few years and, although it is not the most used platform in the market, it is one of the most powerful networks in terms of allowing information to spread thanks to it's divided-into-categories nature. This work focuses on analyzing the properties and characteristics of the social media platform Reddit. Firstly, a formal model is developed for Reddit in the high-performance specification language Maude. "After this, a reachability analysis is performed in the Maude system, which ensures the safety of the system by proving that no states considered unsafe can be reached. Finally, liveness properties are checked by using the Maude LTL logical model checker (LMC)."

Key words: Maude, Formal Methods, Model Checking, Reddit, Code Verification

Contents

Appendix

List of Figures

CHAPTER 1
Introduction

Online social networks are global media where people can collaborate, communicate, spread information and create spaces for dialogue. Reddit is a platform that groups people by interest. This platform has been a great forum for scientific and technological discussion, among other topics of interest, and has even been part of a controversy [6] that shows the power this platform has to create organized groups. The use of Reddit as a cooperative platform begs for a more attentive look at the interactions that are occurring on the network in terms of spread of information.

In this project, we have developed a formal model for the social network Reddit in the high-performance specification language Maude, so that it can then be formally verified with a model checker and other tools for software analysis that are available in Maude's ecosystem. In order to understand the process that was followed to develop the model, we must recall a few core concepts: formal languages, formal methods and model-checking.

A formal language is an arbitrary set of chains over some (finite or infinite) alphabet. In mathematical linguistics and the theory of automata, formal languages are specified by formal grammars and different types of automata that can be described as modifications of non-deterministic Turing machines [9]. A formal specification language is composed of three main components: the **syntax** that determines the specific notation that can be used to represent the specification; the **semantics** that provides the intended interpretation of such a representation; and the **set** of rules that indicate which of the objects properly satisfy the specification. The notion of "Formal Methods" refers to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems. The value of formal methods is that they provide a means to symbolically examine the entire state space of a digital design (whether hardware or software) and establish a correctness or safety property that is true for all possible inputs [5]. Formal specifications are mathematically-based techniques that can be used to accurately describe a system, to analyze its behavior and to verify its very properties through effective and rigorous reasoning tools such as model-checking. Model-checking is an automatic technique for verifying models of software or hardware systems against their specification. This analysis is based on an exploration of the checked system's state space.

Model-checking allows a specific formal system to be verified in a way that satisfies the given properties of interest thanks to the exhaustive analysis of all possible execution paths of the system. In this work, we chose to verify the online social network Reddit, on which we have created our model for automated analysis verification purposes. Our

formal model describes the logic of interactions among Reddit accounts. Since the specification is implemented in Maude, we are able to provide any language expression as an input and let Maude explore the computations arising from such an expression. This allows us to automatically verify Reddit's communication properties in a simple and effective way.

1.1 Motivation

Social media platforms are an ever-present part of the modern life. These social networks allow us to communicate, collaborate and connect with people, regardless of distance. The underlying software that supports the network is constantly evolving and increasing in complexity, allowing the spread of information, whether benign or not, at incredibly high speed. Therefore it is necessary to be able to rigorously guarantee its correctness. That being said, this correctness is difficult to provide, given the complexity behind social platform algorithms in terms of what, how and where information is being stored and displayed. For this, Reddit provides an interesting structure, given it's organized-into-subgroups nature.

Social networks are comprised of a set of actors (the users) that can have relationships with each other. Amounts of actors and variety of relationships vary between platforms, as each network is organized and maintained differently. To truly understand a social platform, a complete and rigorous description of a pattern of social relationships is required. This provides information such as all the relationships between each pair of actors in the platform [4]. Formal methods allow us to display information quickly and define a correct and thorough model of a network.

1.2 Objectives of this work

The objective of this project is to apply model verification techniques to the social network Reddit using the Maude programming language. Maude is a high-performance reflective language and system supporting both equational and rewriting logic specification and programming for a wide range of applications [8]. Once this model has been created, we can later apply automatic model verification to the formal specification of the platform. By using Maude's formal tools, we are able to achieve a deeper understanding of Reddit's interactions between users and how information is spread within this platform. Also, we are able to asses whether the system behaves as expected or it deviates from the user's intentions. This is done by modeling Reddit interactions in the programming language Maude, which allows us to use Maude's model checker to automatically verify Reddit's communication properties.

1.3 Project Structure

After this introductory chapter, we introduce the two main technologies that are used in this project: the language used and the social platform to be be analyzed. Chapter 2 briefly describes Maude, the language that is used to represent the model. Chapter 3 focuses on Model-checking. Chapter 4 provides a thorough overview of Reddit, the social media platform that we want to analyze. In Chapter 5 we explain in detail the model we developed for Reddit. In Chapter 6 we describe the analysis and model-checking process for the developed model. Finally, in Chapter 7, we present the final conclusions on the project as a whole.

CHAPTER 2
The Formal Specification Language Maude

In this chapter we explain Maude's basic syntax and how it works. Maude is a very flexible, expressive and high-performance language. Statements are simple to understand given its simple *rewriting semantics*. This simplicity allows Maude to express both *deterministic* computations and *concurrent, nondeterministic* computations, programmed by equations and rules respectively, making the language very expressive and versatile. Although maintaining such a flexible level of expressiveness without losing too much performance is no mean feat, it is achieved thanks to rewriting rules that are carefully analyzed and are semi-compiled for efficient matching module axioms. This semi-compilation allows every single rewriting step to be traced in an effective way.

In regards to its structure, a program written in Maude is made up of one or more modules, which can be functional modules or system modules. Functional modules differ from system modules in that the latter also support rules, integrating rewriting logic and enabling transitions between system states.

Maude can be publicly found and downloaded at `http://maude.cs.illinois.edu`.

2.1 Modules

Maude's basic units of specification and programming are called modules. Modules are syntax declarations that provide a notation to describe a system and statements that establish the system behavior [7]. In **Core Maude** we distinguish between two types of modules: Functional modules and System modules. The difference between these two is in the statements they can have.

Modules consist of `signatures` and `statements`. `Signatures` are the basic syntax declaration part of a module and consist of the following: `sorts`, `subsorts`, `operators` and `kinds` (we will not go over this last one in the explanation, as they were not used in our model).`Statements`, on the other hand, consist of `equations` and `rules`.

2.1.1. Imports

In Maude, modules can be imported as a submodule of another module in three different modes: `protecting`, `extending`, or `including`:

```
pr (ModuleExpression) .
ex (ModuleExpression) .
inc (ModuleExpression) .
```

- `Protecting`: means that it avoids adding elements belonging to a module that cannot be used or would otherwise cause confusion in our imported submodule.
- `Extending`: means that it will add the elements that might not be usable in a module, but does not allow elements would cause confusion in an imported submodule.
- `Including`: means that it both allows adding elements that cannot be used and does not avoid the ones that may cause confusion in our imported submodule.

2.1.2. Sorts and Subsorts

`Sorts` correspond to the types of data being defined in the code and must be the first thing to be declared in a specification. `Subsorts` serve to order `sorts` by expressing what data types are *a subtype of* another data type, similar to a parent-child relation.

2.1.3. Operators

An operator is a structure we define that consists of a list of `sorts` and returns another `sort` as a result. Operators in Maude can be overloaded. Operators have the following structure:

```
op (OpName) : (Sort-1) ... (Sort-n) -> (Sort) [(OparatorAttributes)] .
```

`Sort-1 ... Sort-n` are the sorts provided as arguments for the operator, `Sort` is the type of result and `OperatorAttributes` are the possible attributes the operator might have and express algebraic properties such as the Associativity or Commutativity. We can also declare various operators at the same time that share the same structure using the word `ops`, like this:

```
ops OpName1 OpName2 : (Sort-1) ... (Sort-n)
-> (Sort) [(OparatorAttributes)] .
```

2.1.4. Variables

Variables can be declared in either of two ways, on-the-fly or using the keyword `var`, and are constrained to being a particular `sort` or `kind`. Variable's names can be both in upper and lower case, but it is more customary in Maude to declare them in uppercase.

An on-the-fly variable's scope is the declaration's occurrence. Therefore such variable has to be accompanied by its `sort` or `kind`. On-the-fly variables consist of the `Variable`

`name`, a colon and the corresponding `sort` or `kind`. Here is an example of a variable declared within a rule:

```
rl [RuleName] OpName(VarName:Sort) .
rl [RuleName] OpName(VarName:[Kind]) .
```

When using the keyword `var` (or `vars` for multiple variables of the same `sort` or `kind`), the variable's scope is the entire module:

```
var VarName : Sort .
vars VarName1 VarName2 : Sort .
var VarName : [Kind] .
vars VarName1 VarName2 : [Kind] .
```

2.1.5. Parsing

Maude supports user-definable syntax and allows for prefix, suffix and mixfix operator declarations.

2.2 Functional Modules

Functional modules are comprised of equations that define data types and operations. These equations are used to simplify the rules by applying them only in the left-to-right direction and are assumed to be terminating, allowing us to repeatedly apply equations as simplification rules and reach a term where no more equations apply, thus obtaining its canonical form.

2.2.1. Equations

The keyword `eq` serves to declare unconditional equations. Both terms from the equation must be of the same kind and any variable in `RTerm` must also appear in `LTerm`. Equations are written thusly:

```
eq (LTerm) = (RTerm) [(Attributes)] .
```

The keyword `ceq` serves to declare conditional equations. A condition can be either a single equation or a conjunction of equations using the binary conjunction connective `/\` which is assumed to be associative [7]. The general structure of conditional equations is the following:

```
ceq (LTerm) = (RTerm)
    if (Condition-1) /\ ... /\ (Condition-n)
    [(Attributes)] .
```

2.3 System Modules

System modules specify *rewrite theories*, which have `sorts`, `kinds` and `operators` and three types of statements: equations, memberships and rules, which can all be conditional.

System modules contain equations *and* rules, which specify transitions between states and will change a state when it matches with the left side of a rule, transforming the state accordingly to the right-hand side of the rule.

2.3.1. Rules

The keyword `rl` serves to declare unconditional rules. Both terms from the rule are of the same kind. Rules describe *local concurrent transitions* in a system. Rules are written as follows:

```
rl [(Label)] : (LTerm) => (RTerm)
[(Attributes)] .
```

The keyword `crl` serves to declare conditional rules. A condition can be either a single equation, membership or rewrite or a conjunction of them using the binary conjunction connective `/\` which is assumed to be associative [7]. The general structure of conditional rules have the following syntax:

```
crl [(Label)] : (LTerm) => (RTerm)
    if (Condition-1) /\ ... /\
    (Condition-n)
    [(Attributes)] .
```

CHAPTER 3

Formal System Analysis by Model Checking

Formal methods are "the applied mathematics for modeling and analyzing systems". Actually, correct behavior can be accurately described by using mathematical logic, which can be used to demonstrate that the program behavior conforms to the intended specification. Formal methods establish system correctness with mathematical rigor and are one of the most recommended verification techniques for software development. Model-based verification techniques are based on models describing the possible system behavior in a mathematically precise and unambiguous manner. Model-checking is an agile verification technique for concurrent, distributed and reactive models that allows us to explore all the states of a system automatically and exhaustively by brute-forcing all possible system states in a systematic possibly implicit manner [2]. This verification technology provides an algorithmic means of determining whether an abstract model satisfies a formal specification expressed as a temporal logic (TL) formula [3].

Model-checking has some advantages over other verification techniques. Experimental methods such as simulation and testing are not comprehensive and exhaustive, with vulnerabilities going unnoticed, which model-checking would not ignore. However, its completeness can only be reliable in a system with a finite number of states that may be extremely high ($> 10^{120}$) but requires the use of abstraction techniques to address systems with infinite states. That being said, model-checking is currently the lightest and fastest verification technique and, additionally, it is fully automatic.

Important criteria for a logic are **expressiveness** and **efficiency**. Expressiveness reflects which kind of properties can and cannot be captured by the logic, such as **safety properties**, **liveness properties** and **fairness properties**. It is of utmost importance that all required properties can be expressed, otherwise there is not point in using the verification method at all. On the other hand, efficiency relates to the complexity of the model-checking problem for a logic and the performance of model-checking algorithms for the logic.

Properties of a model that can be verified can be classified in different types according to the aspects of the system that they contemplate:

- **Reachability properties**: these are the properties that guarantee that a desired state can occur.

- **Safety properties**: are those that guarantee that a harmful situation is not going to occur, which is the opposite of reachability.
- **Liveness properties**: ensure progress in the system, meaning they guarantee that a process does not end for no reason, that no process dies of starvation, and that no deadlocks occur.
- **Fairness properties**: guarantee that a property will be given attention an infinite number of times, intermittently or from a certain state. This includes **recurrence** (a property occurs every so often); and also **persistence** (a property is infinitely given and maintained).

3.1 Temporal Logic

In order for a model checker to be able to verify formulas, a logic capable of specifying the properties on infinite execution paths is required. That is where temporal logic comes in. If a program can be specified in TL, it can be realized as a finite state system. This suggested the idea of model-checking—to check if a finite state graph is a model of a TL specification. Such systems ideally exhibit nonterminating behavior so that they do not conform to the Hoare-style paradigm. They are also typically interactive, distributed and nondeterministic [3], i. e., a reactive system.

Temporal logic is a formalism for describing change over time that is used to manage the executions of the system and fairness issues to guarantee its correctness. It extends propositional or predicate logic by modalities that permit to referral to the infinite behavior of a reactive system, allowing for the specification of the relative order of events [2]. The temporal logic that we use in this project is the Linear Time Logic or LTL introduced by Pnueli.

LTL logic models a single timeline, so it is not suitable for asynchronous systems, where multiple futures are possible depending on which component evolves. The logical operators it uses are $\vee$, $\wedge$, $\neg$, $\rightarrow$, $\leftrightarrow$, `true` and `false`. These are some of the temporary operators that it incorporates:

- `G`: is used for **always**, indicating that `p` is globally true.
- `F`: is used for **finally**, indicating that `p` is going to be fulfilled at some future time.
- `X`: is used for **next**, indicating that `p` will be fulfilled in the next state.
- `U`: is used for **until**, indicating that `p` must be fulfilled until `q` is fulfilled, which must be done at some point.
- `R`: is used for **release**, that `q` must be true until and also in the state in which `p` is fulfilled; if `p` is never satisfied, `q` will remain infinitely true.
- `W`: is used for **weak until**, indicating that `p` must be fulfilled at least until `q` is fulfilled; if that never happens, `p` will remain infinitely.
- `M`: is used for strong **release**.

3.2 Model-checking in Maude

Maude's LTL model checker allows algorithmic verification of competing models that are expressed as rewrite theories. In any Maude system module, we can differentiate two levels of specification:

- **System specification level**: the rewriting theory that defines the behavior of the system.
- **Property specification level**: one or more properties of interest to be verified.

To examine the system specification, it must be executed in the Maude environment to observe if it is behaving in the expected and intended way. However, property verification requires their specification in a logic as well as a procedure that allows them to be verified in a finite range of states.

The LTL syntax is specified in the `model-checker.maude` file. To carry out property verification, Maude uses the modules defined in this file.

Maude's LTL model checker allows you to verify:

- **Reachability properties**:They ensure that a certain state will be reached.
- **Security properties**: They guarantee that a certain configuration will not be given.
- **Liveness properties**: They ensure that an action will have its reaction.

Nevertheless, although fairness is expressible in LTL, this model checker is unable to verify fairness properties, which guarantee that a situation will occur an infinite number of times. This is because it can only work with a finite number of specific states therefore not being prepared to make an equational abstraction to determine if these properties would be fulfilled in an eventual future.

CHAPTER 4
The Social Network Reddit

Reddit is a network comprised of communities based on people's interests. This means, essentially, that it is a network of networks, subdivided by topics. The main timeline is a mix of the different topics the user is interested in. That being said, depending on whether the user is registered or not, the website adapts its contents, appearance and available actions.

To unregistered users, Reddit presents the main page with trending news stories and the most popular posts (organized by `Hot`, `New`, `Top` and `Rising`), along with a search bar that is located on the top of the webpage and can be used to search for a topic of choice.

For registered users, the main page is divided into three sections: `News`, `Home`, and `Popular`. The `News` is a section organized by topics (US/World, Politics, Business, Technology, Science, Sports, Gaming, Entertainment, and Crypto) and shows relevant news for the day. `Home` presents the main feed of the site, whose content is determined by the "subreddits" (topics) of interest chosen by the user. While organized in a similar way to home, `Popular` shows the top popular posts of the whole website, regardless of what subreddits the user follows. Registered members submit content to the site such as links, text posts, and images, which are then voted up or down by other members and can be rewarded with purchased awards such as gold or silver.

Posts are organized by subject into user-created boards called "subreddits" (indicated with `"r/"`), which cover most topics one could think of. Submissions can be organized by Best, Hot, New, Top, Controversial, and Rising depending on the number of up-votes and the up-vote to down-vote ratio. `Top` is for the most up-voted posts of all time; `Best` is also for the most up-voted posts of all time but keeps into account the sample size; `New` is for viewing posts in chronological order with the possibility of seeing posts not yet processed by moderators; `Hot` is a mix between `Best` and `New`, sorting by up-votes but factoring in how recently they were cast; `Rising` is for seeing posts that are quickly becoming popular but do not yet belong to the `Hot` category; and `Controversial` is to see posts that have a very high up-vote **and** down-vote numbers. If a post gets a large number of up-votes, it will appear towards the top of their subreddit and, provided they receive enough up-votes, ultimately on the site's front page.

Subreddits can be managed by moderators, as determined by the creator of the subreddit. Moderators control and enforce the guidelines that must be followed to post and comment on a subreddit and have the ability to delete or disable (you can see but not interact with) a post from the subreddit they moderate. They can also ban (not allow

the user to access subreddit at all) or ghost (user can read and post, but the rest of the subreddit will not be able to see the user's actions) a user.

Within a post, users can also comment, creating threads of discussion. Comments can be either text or links (external or to a different subreddit). Reddit also allows users to use bots on the site, which are used by commenting the bot's name on a post. Bots are generally used for subreddit moderation (like removing posts based on keywords in user's profile) but can also help the user download a video, alter the text of the original post, up-vote or down-vote a specific user or subreddit or count words of a post, to name a few.

4.1 Reddit structure and features

In this section, we describe the basic aspects of Reddit, as well as outline a semiformal description of the four Reddit characteristics we have chosen to be worked on.

4.1.1. Structure of Reddit

To break down the structure of Reddit in a simplified manner, let us summarize three distinct components that comprise Reddit's social network:

- **Subreddit**: A subreddit is any distinguished category that has been created to form communities within the general website. It is the main place of interactions. All posts must belong to a subreddit. Subreddits are followed by users who are interested in the particular topic they are about.
- **Post**: Posts are the main form of interaction on the website. They are the content posted in the subreddits and are published by users, who can make a post on a subreddit they do not follow, but it is common practice to post on the subreddits you already follow.
- **User**: The user, as the name suggests, is the user of the platform. The user will be the one that will use the **Actions** described below. In the user's menu, they will be able to access the following:
  - Posts: This is an overview of the posts created by the user.
  - Comments: This is an overview of the comments created by the user.
  - Communities: This is a list of the user's followed subreddits.

4.1.2. Actions of Reddit

In this section, we describe the basic actions that a user can make in order to interact with the platform [10]:

- `Subscribe`: This action is used by the user to interact with a given subreddit by subscribing or unsubscribing. Once the user is subscribed to a subreddit, posts from this "sub" will be added to the user's main feed, sorted by whatever parameter the user sets (`Best`, `Hot`, `New`, `Top`, `Controversial` or `Rising`). For the user to be able to subscribe to a subreddit, either it is a public subreddit or the user has been invited to a private subreddit.

- `Submit`: This action is used to create a new post on the subreddit, which will be a link to another platform (if it is a link) or a `self-post` in the subreddit (a `self-post` is a post that does not link ouside of Reddit). Posts themselves consist of a title and a body.

- `Comment`: This action is used to create a comment on a post. The comment can be either a new comment or a reply to an already posted message. The basic structure of a `Comment` is the following:

  - New comment: This comment is a reply that will appear immediately underneath the original post and can be considered the parent of a thread of replies if other users reply to this first comment. A new comment can be:
    * Comment on `self-post`.
    * Comment on link post.
  - Comment to comment: This comment is a reply to a new comment made on a post. It will appear underneath the first comment with a vertical line on the left of the comment, visually indicating that it is related to a previous comment.

  Comments can be understood as a recursive structure, with comments within comments, all relating to one original post. This makes conversations about a topic easy to manage and be compartmentalized into threads. Comments can be clicked on to "collapse", meaning that if you click on the parent comment, the whole thread is hidden. This makes reading the comment section easier to manage. Comments can be sorted by the same parameters as posts, but this sorting will only affect the **parent comment**.

- `Delete`: This action is used to delete a link, `self-post` or `comment`.

- `Vote`: This action is used to cast a vote on a post or comment. There are three basic actions when voting: The user can **up-vote**, **down-vote** or "**un-vote**". **Up-voting** indicates the post has been "liked" by the user and adds +1 to the vote count (`Karma`). **Down-voting** indicates the post has been "disliked" by the user and adds -1 to the vote count. **Un-voting** is done by clicking on the already clicked highlighted arrow to "delete" the vote, equivalent to casting a neutral vote in the vote count. When a post is voted (up or down), the user who posted gets `karma` (positive or negative). `Karma` is the cumulative amount of votes a user has across all of their posts and/or comments.

4.2 Reddit action interactions

Reddit interactions seem quite simple, but they are not. Here we provide an example of three sequences of actions that show some subtleties, as will be explained later on:

- Post in a subreddit `p` - comment on `p` - delete `p`.

- Post in a subreddit `p` - comment `c` on `p` - answer `c` - delete `c`.

- Post in a subreddit `p` - up-vote `p` - undo up-vote.

In this section we do not seek to introduce a formal discussion yet, but simply provide an intuitivie explanation for the interaction sequences mentioned above.

Let us consider four Reddit users, say `u/aLynx`, `u/b0bbyPR`, `u/chocoSystems`, and `u/dedTech`. We suppose that the four accounts belong to four distinct high-tech workers: Alice, Bob, Charlie and Dave. Sequence 1, seen in Figure 4.1, involves two users, `u/aLynx` and `u/b0bbyPR`, where `u/aLynx` makes a post `p` on a subreddit `r/` and `u/b0bbyPR` comments on the post, then `u/aLynx` deletes `p`. Sequence 2, illustrated in Figure 4.2, involves two users, `u/aLynx` and `u/b0bbyPR`, where `u/aLynx` makes a post `p` on a subreddit `r/` and `u/b0bbyPR` makes a comment `c` on the post, `u/aLynx` answers `c`, and finally, `u/b0bbyPR` deletes `c`. Sequence 3 involves three users, `u/aLynx`, `u/b0bbyPR` and `u/chocoSystems`, where `u/aLynx` makes a post `p` on a subreddit `r/` and `u/b0bbyPR` up-votes the post, then `u/b0bbyPR` undoes the up-vote, as shown in Figure 4.3. We assume that for all three sequences, all users are subscribed to the same subreddit.

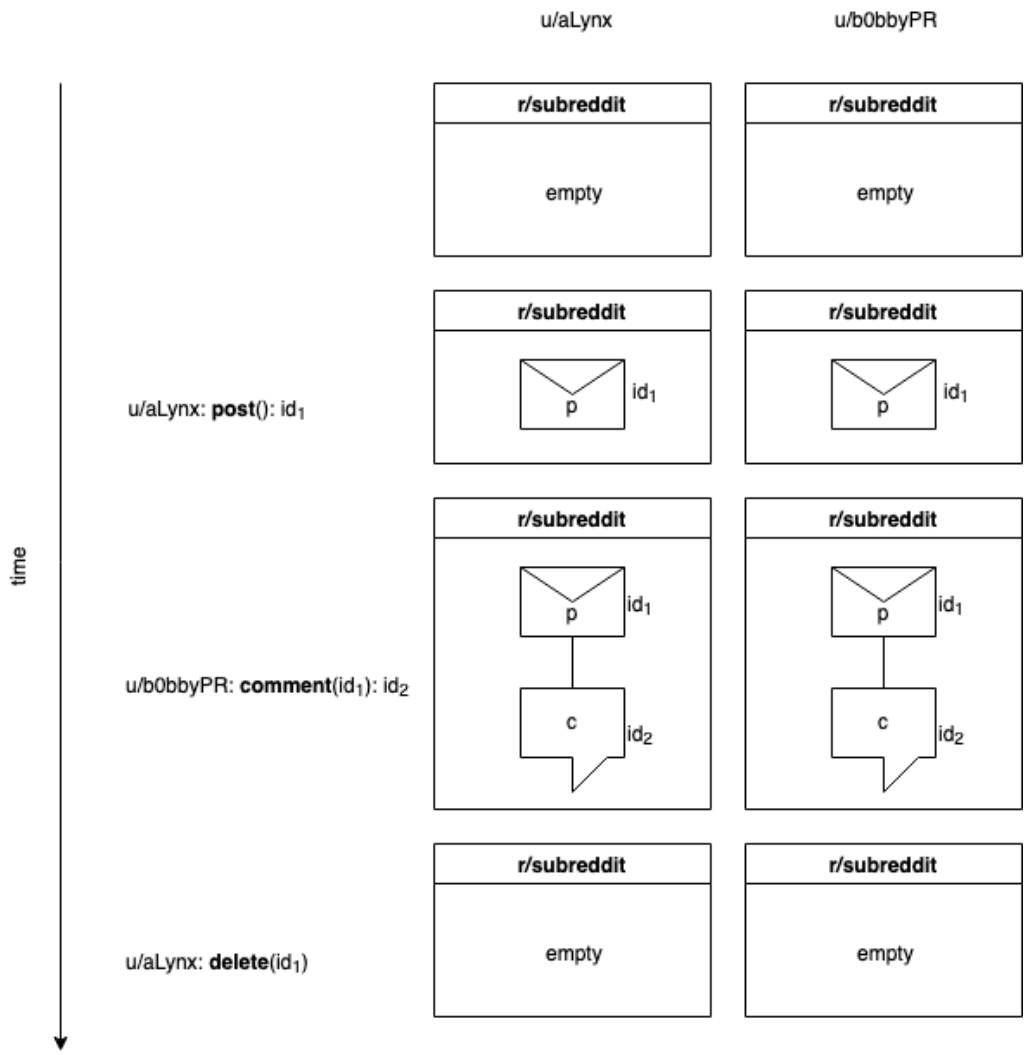

Figure 4.1: Sequence 1

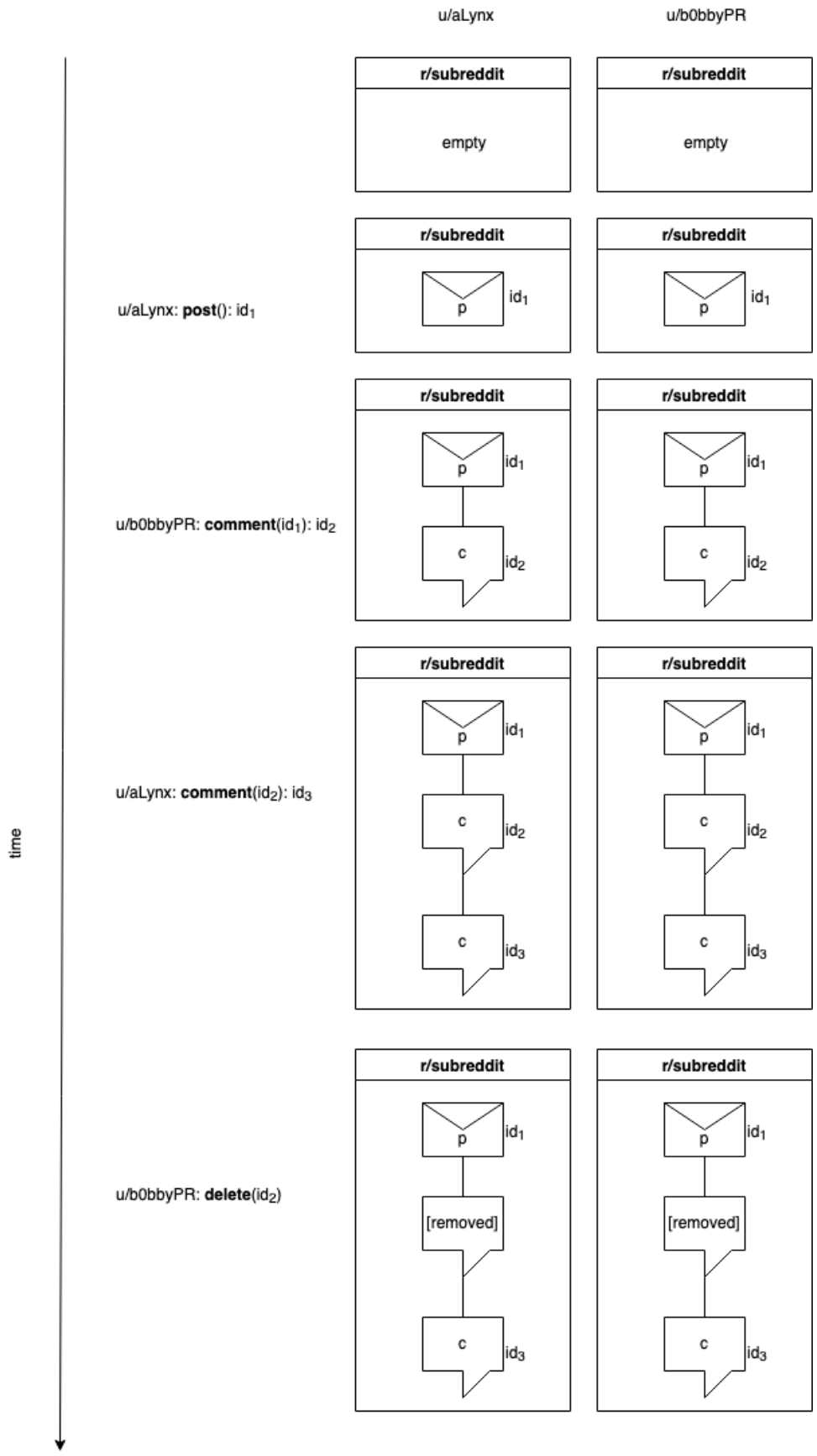

Figure 4.2: Sequence 2

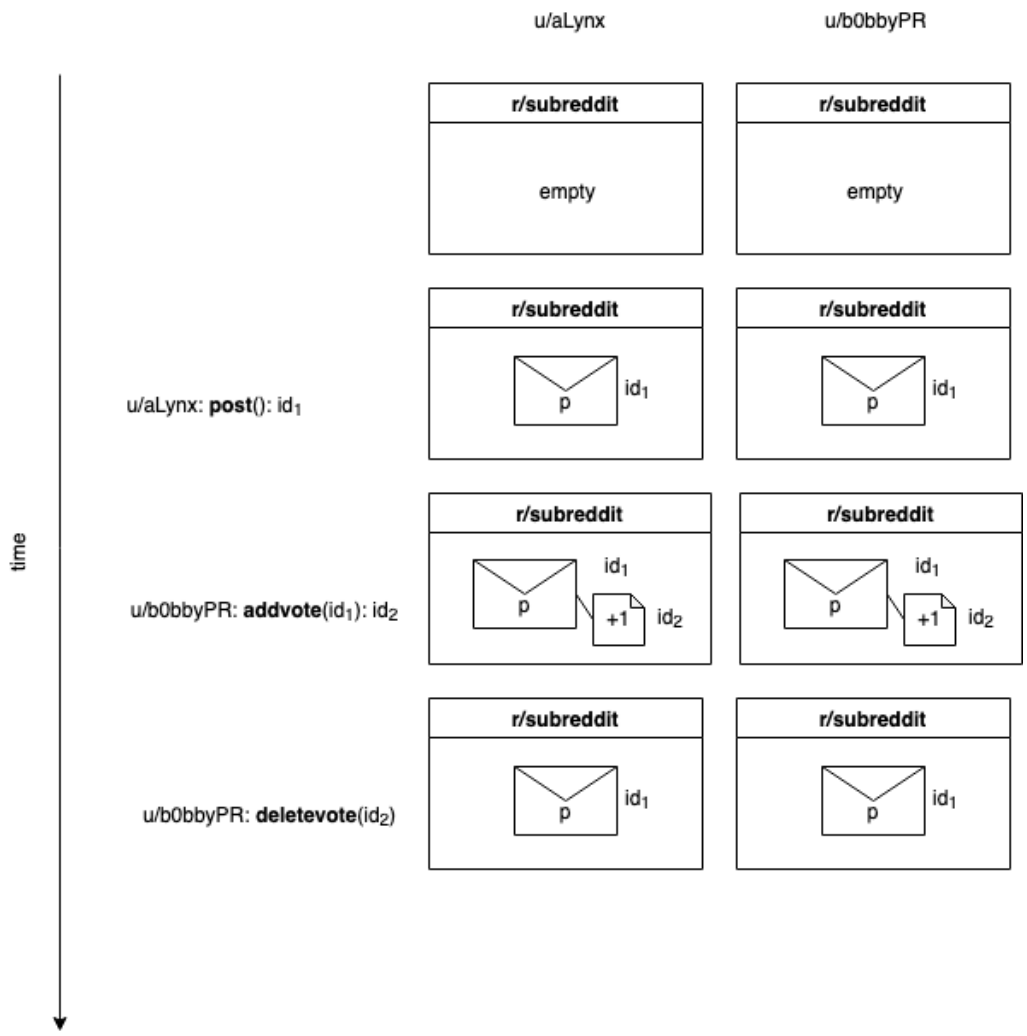

Figure 4.3: Sequence 3

In these three sequences we can see how the actions of deleting posts can provide different results. In the first sequence, p is removed from any timeline, along with all of the comments that could have been made. In the second sequence, c is removed from any timeline, while the original post remains along with all of the replies that were made to comment c. Comment c will appear in the thread of conversation as [removed]. Lastly, in the third sequence, when u/b0bbyPR cancels his up-vote, this does not cause any effect to p, which still exists at the end.

The presented sequences are just a few of a plethora of interactions that can occurr on the Reddit platform. As we can see, even if these examples are simple, there are some counterintuitive or even malicious effects that we explore in the next section. This is why a formal and accurate model is needed to trace and analyze Reddit interaction patterns.

4.3 Reddit action interaction with counterintuitive effects

Let us consider the four Reddit users mentioned before, u/aLynx, u/b0bbyPR, u/chocoSystems, and u/dedTech. Alice, Bob, and Dave frequently post on r/formalmethods, while Charlie only sees the posts from that subreddit when they appear on the **Popular** tab.

Alice has recently spoken at a conference on building high-performance code analyzers and has decided to post a link to a video of that talk in r/formalmethods. Bob, being a frequent user of this subreddit sees the new post on his feed and watches her talk. Since he found it interesting and informative, he decides to up-vote it and comments on the post by positively praising the talk. The post ends up becoming a popular post on the subreddit and eventually makes it to the **Popular** tab.

Charlie sees the post and comments under Bob's comment, asking about where he can find more information on formal methods since he is an experienced programmer but is new to formal methods. After Charlie comments, Alice notices the post's title contains a typo and decides to delete the post and redo it at another time.

After the post deletion, several things occur: 1) Bob gets notified of Charlie's comment and can find the post from the notification center but not searching the subreddit in question. He is also unable to reply to Charlie's reply, although he can still view the post. 2) Dan, a given user of the subreddit, for a few hours (the post remains "locked" for a few minutes before being fully deleted) can see the original post on the subreddit with all of the comments and votes but he is unable to interact with the post in any other way. 3) Alice can no longer access her deleted post, but Bob and Charlie can access it whenever they want because their replies to the original post have not been deleted, but the content of the post will.

4.4 Malicious interactions

In this section, let us consider three Reddit users, u/aLynx, u/b0bbyPR, u/computerWiz, corresponding to Alice, Bob and Chuck. We suppose that Alice's and Bob's accounts belong to two distinct tech workers, while Chuck is a user with malicious intent. Alice, Bob

and Chuck frequently post on `r/formalmethods`.

Alice has recently talked in a conference on building high-performance code analyzers and has decided to post a link to a video of that talk in r/formalmethods. Bob, being a frequent user of this subreddit,he sees the new post on his feed and watches her talk. Since he found it interesting and informative, he decides to up-vote it and comments on the post positively praising the talk.

This time, Chuck sees the post is starting to become popular and decides to use `Reddit bots` to 1) hide the original post and 2) popularize his own post. To do this, he utilizes some bots that will spam the down-vote, report and comment negatively on the original post, thus hiding the original post due to unpopularity and possibly forcing Alice to delete her post or even her whole account. Once the original post is hidden or deleted, he creates a new account with a similar name to `u/aLynx`, say `u/aLynx1`, and he reposts the original link in his own post, acting as Alice and uses bots to mass up-vote and comment on the new post.

CHAPTER 5
Reddit formal specification in Maude

This chapter focuses on describing the formal model of Reddit that has been specified in the Maude language. Before going into the model, it was necessary to gain fluency with Maude using the Maude Primer Manual [7], which through a series of explanations, examples and exercises was a crucial tool for coming to grips with the language. We also used Anima [1], which was invaluable as a visually intuitive tool to analyze the different `States` that can be reached during the program execution. Anima presents the execution graph of any input term in a given rewriting theory and describes the trace of the applied equations and rules and the changes they cause in the system. This tool greatly facilitates *reachability analysis* early in the design and allows the exploration of potential unwanted `States` that may be reached due to errors in the code.

Knowing that Reddit is composed by a series of users, SubReddits, posts, comments and votes, it is easy to create an initial `State` that contains all of these. The `State` is also given a counter that is used to designate the IDs of every new object created as well as a list of `Actions` we can take the system configuration to get to different `States`. The constructor that is used to represent the system `States` is defined by:

```
1 op _|_|_|_|_|_ : Nat UserSet SubRedditSet PostSet VoteSet ActionList ->
      State [ctor] .
```

5.1 Sorts

Let us explain in more detail the operator presented before, which allows the representation of any given `State` in the Reddit model, to be built by considering the following `sorts`:

- `Nat` equates to a counter that we used to assign an ID to each new object that is created of the four types that Reddit has: user, SubReddit, post/comment and vote.
- `UserSet` refers to the list of Users registered in Reddit, meaning, created in Reddit.
- `SubRedditSet` represents the list of SubReddits that exist in Reddit at any given moment, being able to delete an existing SubReddit or create a new one.
- `PostSet` constitutes the list of Posts created in Reddit, belonging to a particular SubReddit.

- `VoteSet` denotes the list of Votes that correspond to each post, whether positive or negative (upvotes or downvotes).

The last element represents an `ActionList`, which we explain in Chapter 5.3.

Corresponding to each of the sorts, we have built their respective values, defined by using suitable constructors. These elements are what can be created or removed from the network during the execution of the model:

```
*** Constructors ***
op user : Nat String NatList NatList NatList -> User [ctor] .
op subr : Nat String String NatList NatList -> SubReddit [ctor] .
op post : Nat String String Nat NatList NatList -> Post [ctor] .
op vote : Nat Nat Nat Bool -> Vote [ctor] .
```

5.1.1. User

Users have the following structure:

```
op user : Nat String NatList NatList NatList -> User [ctor] .
```

- `Nat`: Id of the user.
- `String`: Username, written with a "u/" before the username to make it more readable within the `State`.
- `NatList`: SubReddits the user is subscribed to.
- `NatList`: List of posts created by the user, in chronological order, regardless of SubReddit.
- `NatList`: Total votes obtained on posts the user has made. Also known as `karma`.

5.1.2. SubReddit

SubReddits have the following structure:

```
op subr : Nat String String NatList NatList -> SubReddit [ctor] .
```

- `Nat`: Id of the SubReddit.
- `String`: Name of the SubReddit, written with an "r/" before the name to make it more readable within the `State`.
- `String`: Description of what the SubReddit is about or for. Can be omitted.
- `NatList`: List of users subscribed to the SubReddit.
- `NatList`: List of posts created within the SubReddit.

5.1.3. Post

Posts have the following structure:

```
op post : Nat String String Nat NatList NatList -> Post [ctor] .
```

- `Nat`: Id of the post.
- `String`: Title of the post. Omitted if the post is a reply to another post, since the title is considered to be the parent's title.
- `String`: Text within the post.
- `Nat`: ID of the user who created the post.
- `NatList`: List of comments that have replied to the post.
- `NatList`: List of votes made on the post.

5.1.4. Vote

Votes have the following structure:

```
op vote : Nat Nat Nat Bool -> Vote [ctor] .
```

- `Nat`: ID of the vote.
- `Nat`: ID of the user who voted.
- `Nat`: ID of the post where the vote has been cast.
- `Bool`: Value of the vote (positive or negative).

5.2 Actions

In a system configuration,the `ActionList` component represents the list of `Actions` that can be taken. These actions allow for:

- The creation and deletion of users (`createuser` and `deleteuser`).
- The creation and deletion of subreddits (`createsub` and `deletesub`).
- The creation and deletion of posts (`createpost` and `deletepost`).
- The creation and deletion of comments (`reply` and `deletereply`).
- The creation and deletion of votes (`addvote` and `deletevote`).
- The deletion of comments in cascade when a parent post or comment is eliminated (`deletereplycascade`).

These `Actions` are designed for creating all the necessary elements that exist in the Reddit platform and to simulate all possible common interactions.

5.3 Reddit action characteristics

In this last section we recreated the different interactions that were set as an example in Chapter 4.2. For this we have defined the initial `State` as well as the initial `Actions` taken at a point `t` in time where the users have been added following the explanations in Chapter 4, as well as a few SubReddits where the posts can go.

```
--- Initial State
op init : -> State .
eq init = 0 | none | none | none | none | aListRed .
--- Initial Reddit actions
op aListInit : -> ActionList .
eq aListInit = createuser("u/aLynx") , createuser("u/b0bbyPR") , createuser
    ("u/chocoSystems") , createuser("u/dedTech") , createsub(0, "r/Maude",
    "Lorem ipsum") , createsub(0, "r/ModelChecking", "Lorem ipsum") ,
    createsub(0, "r/FormalMethods", "Lorem ipsum") .
```

Then, we record the different `ActionLists` that represent the interactions proposed formerly, which is shown in the following code excerpt:

```
*** Reddit action interactions ***
op aListInter1 : -> ActionList .
eq aListInter1 = createpost(0, 6, "Conference", "Conference link") , reply
    (1, 7, "Good job!") , deletepost(7) .
op aListInter2 : -> ActionList .
eq aListInter2 = createpost(0, 6, "Maude is cool", "Body of text") , reply
    (1, 7, "Nice") , reply(2, 8, "Delete this") , deletepost(8) .
op aListInter3 : -> ActionList .
eq aListInter3 = createpost(0, 6, "Post here", "Bottom text") , addvote(1,
    7, true) , deletevote(8) .
```

Note that these consequences are in correspondence with the interactions we presented in Chapter 4, namely:

- Post in a subreddit `p` - comment on `p` - delete `p`.
- Post in a subreddit `p` - comment `c` on `p` - answer `c` - delete `c`.
- Post in a subreddit `p` - up-vote `p` - undo up-vote.

Finally, recall that rules are what enable transitions between `States`. In this specification, the rules are used to mimic the `Actions` we can take so that the rule corresponding to a given `Action` will consume that `Action` from the given `ActionList`. The left part of this rule corresponds to the current `State` and the `Action` that is going to be consumed; on the right-hand side of the rule we have the resulting state after applying the consumed `Action`.

As an example, if we want to create a user we can use the following rule, which consumes the `createuser Action`, and so the `Action` disappears from the `ActionList` in the target state, while the new user has been added to the `State`, as we can see in Figure 5.1:

```
rl [createuser] :
        CNT |
        USet |
        SSet |
        PSet |
        VSet |
        createuser(UName) , AList
        =>
        CNT + 1 |
        USet user(CNT, UName, nil, nil, nil) |
        SSet |
        PSet |
        VSet |
        AList [narrowing] .
```

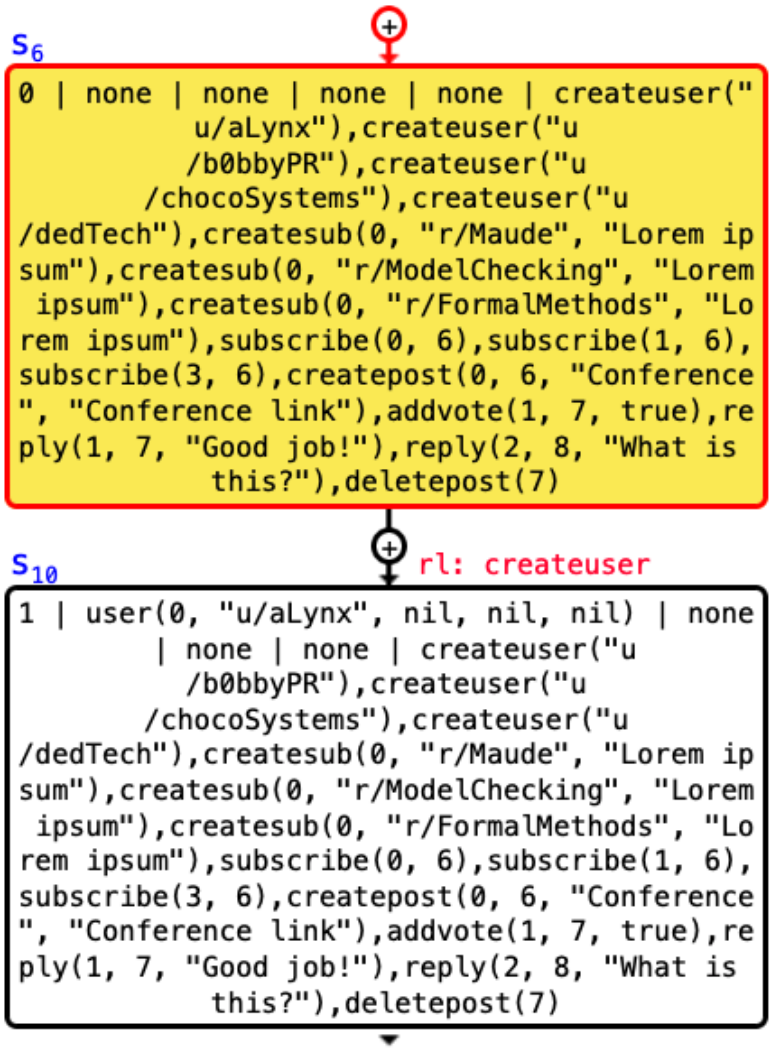

Figure 5.1: Anima state transition when consuming `createuser` action

In Reddit, when a user deletes a post, the post is removed from the SubReddit but if someone commented, that post (along with all other comments) remains for them. When you delete a comment, that comment now appears as `[removed]` but the list of answers to that comment remain. In addition, all `Karma` (votes) received from comments on deleted posts or replies to deleted comments remain for the user who received those votes. This leads to many inconsistencies in the network, where posts and comments remain in some places but not in others. One possibility to model this would be to add a `hidden` or `shown` element in the posts and comments, but the problem of having inconsistent access to information in the network remains. Therefore we opted for cascade deletion, meaning once you decide to delete a post or comment, everything linked to that "parent" will also be deleted. This creates a more controlled network environment that can be seen as a "repaired" version of the system that allows for better analysis and verification. Here we present the operations and equations that allow the rule `deletereply` to delete in cascade all comments and votes related to that post or comment:

```
op deleteReplyCascade : State NatList -> State .

eq deleteReplyCascade(CNT | USet | SSet | PSet post(N, T, Txt, UId, CList,
    VList) PSet' | VSet | AList, NL N NL') = deleteReplyCascade(CNT | USet
    | SSet | (PSet PSet') | VSet | AList, NL NL' NL'') .
eq deleteReplyCascade(CNT | USet | SSet | PSet | VSet vote(N, UId, PId, B)
    VSet' | AList, NL N NL') = deleteReplyCascade(CNT | USet | SSet | PSet
    | (VSet VSet') | AList, NL NL' NL'') .
eq deleteReplyCascade(CNT | USet | SSet | PSet | VSet | AList, NL) = CNT |
    USet | SSet | PSet | VSet | AList [owise].

rl [deletereply] :
        CNT |
        USet |
        SSet |
        PSet post(PId, T, Txt, UId, CList, VList) |
        VSet |
        deletereply(PId) , AList
        =>
        CNT |
        USet |
        SSet |
        PSet |
        VSet |
        deleteReplyCascade(CNT | USet | SSet | PSet | VSet | AList, CList,
            VList), AList [narrowing] .
```

It is worth mentioning that, in contrast to rules, Maude executes the proper equations without producing any State transitions, meaning that the State is deterministically simplified in an extremely efficient way.

CHAPTER 6
Analysis and Verification of the Reddit system

This chapter's core purpose is to provide an explanation on how the created model has been analyzed and model-checked. For illustrative purposes we use the Reddit properties that were mentioned and explained in Chapter 3.2: reachability, safety, liveness and fairness. While these properties were explained individually, they must be understood as interrelated and complimentary to one another. Knowing that any property of a reactive system can be expressed as the conjunction of a liveness property and a safety property, meaning it is an intersection between the two, we will focus on these two.

To allow for a more complete analysis, both a plain reachability analysis using Maude's interpreter and leveraging the more sophisticated Maude's model-checker, two models were created following the standard distinction from conceptual modeling, between deductive and operational models: a guided deductive model and an automatic operational model. In Chapter 6.1 we use the guided model to make the reachability analysis and in Chapter 6.2 the automatic model is used instead.

6.1 Reachability analysis

Before verifying the liveness and safety properties, making some simple reachability analysis is sensible. This analysis is geared towards ensuring that all the intended states are reached by the system and no unwanted or harmful states are found. The Anima tool [1] is especially useful when dissecting each reached `State` because it provides a more visual representation. That being said, the `search` command that is available when using Maude in the terminal was also used when a more exhaustive state search was required.

For this reachability analysis we will hearken back to Chapter 5.3, where we had some interactions that were modeled, the initial `State` and all the possible `Actions` we could use. Using the `search` command in the Maude terminal, we can find all possible reachable states for a given `ActionList`, and we can also look for some incorrect or harmful states, eventually ensuring that they are not reachable.

The following executions of the model demonstrate that all three cases, corresponding with the three `ActionLists` shown in Chapter 5.3 yield the expected results, where they reach their final `State` with the `ActionList` consumed.

The command `search init =>! X:State .` was used in order to obtain only the final resulting `State`. If the command `search init =>* X:State .` was used instead, we would be able to observe every intermediate `State`, as shown in Figures 6.1 and 6.2, which depict Anima's execution for these examples:

```
********* CASE 1 *********
***** SOLUTION STATE *****
Result of: search init =>! X:State .
search in REDDIT-RL : init =>! X:State .

Solution 1 (state 11)
states: 12  rewrites: 28 in 1ms cpu (3ms real) (19178 rewrites/second)
X:State --> 9 | user(0, "u/aLynx", nil, nil, nil) user(1, "u/b0bbyPR", nil,
     nil, nil)
     user(2, "u/chocoSystems", nil, nil, nil) user(3, "u/dedTech", nil, nil,
          nil) | subr(4,
     "r/Maude", "Lorem ipsum", 0, nil) subr(5, "r/ModelChecking", "Lorem
         ipsum", 0, nil)
     subr(6, "r/FormalMethods", "Lorem ipsum", 0, nil) | none | none | nil

No more solutions.
states: 12  rewrites: 28 in 1ms cpu (3ms real) (17565 rewrites/second)
```

```
********* CASE 2 *********
***** SOLUTION STATE *****
Result of: search init =>! X:State .
search in REDDIT-RL : init =>! X:State .

Solution 1 (state 12)
states: 13  rewrites: 30 in 0ms cpu (0ms real) (54644 rewrites/second)
X:State --> 10 | user(0, "u/aLynx", nil, 7, nil) user(1, "u/b0bbyPR", nil,
    nil, nil) user(
     2, "u/chocoSystems", nil, nil, nil) user(3, "u/dedTech", nil, nil, nil)
          | subr(4,
     "r/Maude", "Lorem ipsum", 0, nil) subr(5, "r/ModelChecking", "Lorem
         ipsum", 0, nil)
     subr(6, "r/FormalMethods", "Lorem ipsum", 0, 7) | post(7, "Maude is
         cool",
     "Body of text", 0, nil, nil) | none | nil

No more solutions.
states: 13  rewrites: 30 in 0ms cpu (0ms real) (46012 rewrites/second)
```

```
********* CASE 3 *********
***** SOLUTION STATE *****
Result of: search init =>! X:State .
search in REDDIT-RL : init =>! X:State .

Solution 1 (state 10)
states: 11  rewrites: 24 in 0ms cpu (0ms real) (27874 rewrites/second)
X:State --> 9 | user(0, "u/aLynx", nil, 7, nil) user(1, "u/b0bbyPR", nil,
    nil, nil) user(
    2, "u/chocoSystems", nil, nil, nil) user(3, "u/dedTech", nil, nil, nil)
        | subr(4,
    "r/Maude", "Lorem ipsum", 0, nil) subr(5, "r/ModelChecking", "Lorem
       ipsum", 0, nil)
    subr(6, "r/FormalMethods", "Lorem ipsum", 0, 7) | post(7, "Maude post",
        "Bottom text",
    1, nil, nil) | vote(8, 1, 7, true) | nil

No more solutions.
states: 11  rewrites: 24 in 0ms cpu (0ms real) (24615 rewrites/second)
```

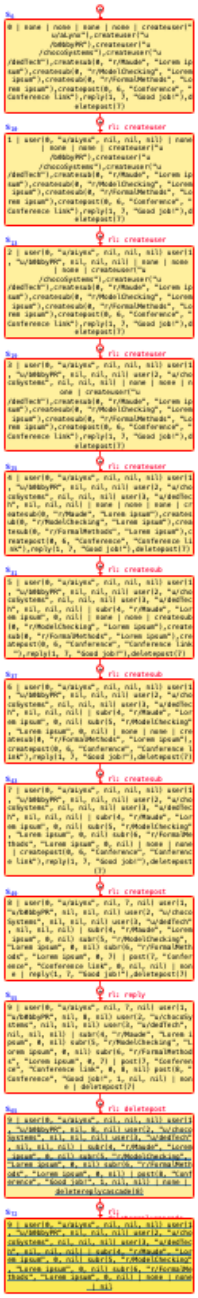

Figure 6.1: Anima execution tree of Case 1

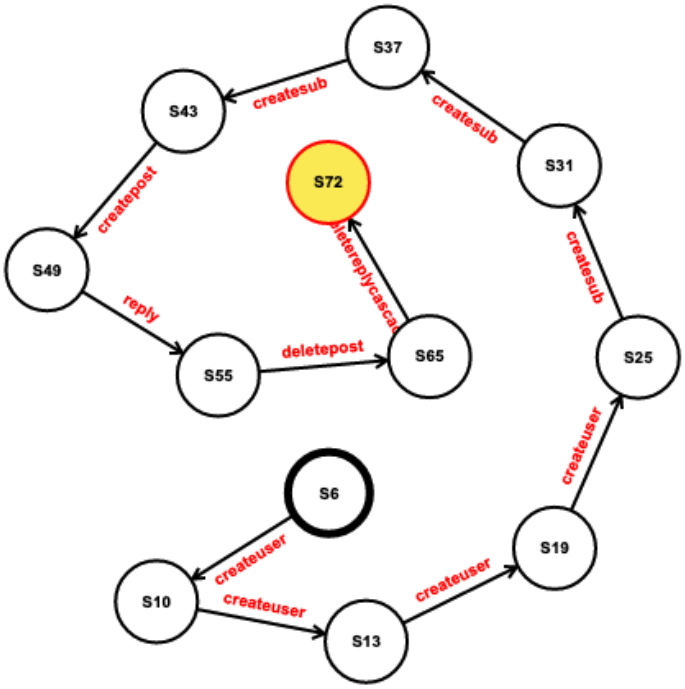

Figure 6.2: Anima execution graph of Case 1

We can also observe different tried searches to incorrect `States` (for the `ActionList` in Case 1, although this is true for all `ActionLists`) and it is demonstrated that they are in no way reachable, thus exemplifying how the model can be verified in terms of unreachability of forbidden states:

```
********* CASE 1 *********
*** UNREACHABLE STATES ***
search init =>! 9 | user(0, "u/aLynx", nil, nil, nil) user(1, "u/b0bbyPR",
    nil, 8, nil) user(2, "u/chocoSystems", nil, nil, nil) user(3, "u/
    dedTech", nil, nil, nil) | subr(4, "r/Maude", "Lorem ipsum", 0, nil)
    subr(5, "r/ModelChecking", "Lorem ipsum", 0, nil) subr(6, "r/
    FormalMethods", "Lorem ipsum", 0, nil) | none | none | nil .
search in REDDIT-RL : init =>! 9 | user(0, "u/aLynx", nil, nil, nil) user
    (1, "u/b0bbyPR",
     nil, 8, nil) user(2, "u/chocoSystems", nil, nil, nil) user(3, "u/
         dedTech", nil, nil,
     nil) | subr(4, "r/Maude", "Lorem ipsum", 0, nil) subr(5, "r/
         ModelChecking",
     "Lorem ipsum", 0, nil) subr(6, "r/FormalMethods", "Lorem ipsum", 0, nil
         ) | none | none
     | nil .

No solution.
states: 12  rewrites: 28 in 0ms cpu (0ms real) (32332 rewrites/second)
```

```
search init =>! 9 | user(0, "u/aLynx", nil, 7, nil) user(1, "u/b0bbyPR",
    nil, nil, nil) user(2, "u/chocoSystems", nil, nil, nil) user(3, "u/
    dedTech", nil, nil, nil) | subr(4, "r/Maude", "Lorem ipsum", 0, nil)
    subr(5, "r/ModelChecking", "Lorem ipsum", 0, nil) subr(6, "r/
    FormalMethods", "Lorem ipsum", 0, nil) | none | none | nil .
search in REDDIT-RL : init =>! 9 | user(0, "u/aLynx", nil, 7, nil) user(1,
    "u/b0bbyPR",
     nil, nil, nil) user(2, "u/chocoSystems", nil, nil, nil) user(3, "u/
         dedTech", nil, nil,
     nil) | subr(4, "r/Maude", "Lorem ipsum", 0, nil) subr(5, "r/
         ModelChecking",
     "Lorem ipsum", 0, nil) subr(6, "r/FormalMethods", "Lorem ipsum", 0, nil
         ) | none | none
     | nil .

No solution.
states: 12  rewrites: 28 in 0ms cpu (0ms real) (98939 rewrites/second)
```

```
search init =>! 9 | user(0, "u/aLynx", nil, 7, nil) user(1, "u/b0bbyPR",
    nil, 8, nil) user(2, "u/chocoSystems", nil, nil, nil) user(3, "u/
    dedTech", nil, nil, nil) | subr(4, "r/Maude", "Lorem ipsum", 0, nil)
    subr(5, "r/ModelChecking", "Lorem ipsum", 0, nil) subr(6, "r/
    FormalMethods", "Lorem ipsum", 0, nil) | none | none | nil .
search in REDDIT-RL : init =>! 9 | user(0, "u/aLynx", nil, 7, nil) user(1,
    "u/b0bbyPR",
     nil, 8, nil) user(2, "u/chocoSystems", nil, nil, nil) user(3, "u/
         dedTech", nil, nil,
     nil) | subr(4, "r/Maude", "Lorem ipsum", 0, nil) subr(5, "r/
         ModelChecking",
     "Lorem ipsum", 0, nil) subr(6, "r/FormalMethods", "Lorem ipsum", 0, nil
         ) | none | none
     | nil .

No solution.
states: 12  rewrites: 28 in 0ms cpu (0ms real) (64073 rewrites/second)
```

```
search init =>* 11 | user(0, "u/aLynx", nil, nil, nil) user(1, "u/b0bbyPR",
     nil, 8, nil) user(2, "u/chocoSystems", nil, nil, nil) user(3, "u/
    dedTech", nil, nil, nil) | subr(4, "r/Maude", "Lorem ipsum", 0, nil)
    subr(5, "r/ModelChecking", "Lorem ipsum", 0, nil) subr(6, "r/
    FormalMethods", "Lorem ipsum", 0, nil) | none | none | nil .
search in REDDIT-RL : init =>* 11 | user(0, "u/aLynx", nil, nil, nil) user
    (1, "u/b0bbyPR",
    nil, 8, nil) user(2, "u/chocoSystems", nil, nil, nil) user(3, "u/
        dedTech", nil, nil,
    nil) | subr(4, "r/Maude", "Lorem ipsum", 0, nil) subr(5, "r/
        ModelChecking",
    "Lorem ipsum", 0, nil) subr(6, "r/FormalMethods", "Lorem ipsum", 0, nil
        ) | none | none
    | nil .

No solution.
states: 12  rewrites: 28 in 0ms cpu (0ms real) (67307 rewrites/second)
```

For instance, the first search corresponds to searching for an irreducible `State` where the user `u/b0bbyPR` has access to his reply after the original post he replied to has been eliminated, but since no such `State` can be found, we can prove that cascade eliminations are made successfully and no inconsistent `States` where users can see eliminated posts can be found. The second search corresponds to searching for a normalized `State` where the user `u/aLynx` has access to his post, even after deletion. The third search corresponds to both `u/aLynx` and `u/b0bbyPR` having access to their posts after the original post's deletion. The last search is similar to the first, but we provide a more flexible searching condition that considers for matching all generated states in any possible branch. Just as the first search, the searched `State` is unreachable, thus proving the model generates no inconsistent `States`.

6.2 Model-cheking

Now that the reachability has been verified, we can move onto verifying the liveness and safety properties we mentioned previously. In this section we will employ Maude's LTL model-checker to verify a series of properties in our Reddit specification.

We have created a secondary module called `REDDIT-PREDS`, which contains the protected `system module` specification and includes the module `SATISFACTION` previously loaded into the Maude environment.

Safety and liveness verification

Liveness properties ensure that if a change or an action has occurred in the system, this change will have an impact, avoiding system stagnation. Safety properties ensure that an incorrect or harmful situation is unable to be reached.

For example, we should consider guaranteeing that all posts that have not been directly eliminated must remain on the platform, even if the user who created them has deleted their account:

«All posts created by a user will remain in the network, even if the user deletes their account.»

In order to examine this property, we need to translate the statement of the property into a LTL formula, which we do by equationally defining the following state predicates:

```
[] (postexists(N:Nat, M:Nat) W <> (userexists(M:Nat) ∨ ( ¬
                    userexists(M:Nat))))
```

```
eq < CNT | USet | SSet | PSet post(PId, T, Txt, UId, CList, VList) | VSet
    > |= postexists(PId, UId) = true .
eq < CNT | USet user(UId, UName, SList, PList, KList) | SSet | PSet | VSet
     > |= userexists(UId) = true .
```

Another item of interest to ensure that every vote belongs to the post where it was cast.

«Every vote is always associated to its corresponding post.»

In order to examine this property, we need to translate the statement of the property into a LTL formula, which we do by equationally defining the following state predicates:

```
[] (vote(N:Nat) -> post(N:Nat))
```

```
eq < CNT | USet | SSet | PSet | VSet vote(VId, UId, PId, B)  > |= vote(VId)
     = true .
eq < CNT | USet | SSet | PSet post(PId, T, Txt, UId', CList, VList VId
    VList') | VSet  > |= post(VId) = true .
```

CHAPTER 7
Conclusion

In this project, we have been able to define a formal specification and model Reddit in the Maude language and analyze its properties. Subsequently, this specification has been rigorously analyzed by completing some reachability tests and using model-checking. The reachability tests were able to prove that every `State` reached was intended and correct. We enhanced this analysis by using the Anima tool, which visually assisted us by providing a visual representation of the execution trees generated by the model. Therefore, we have been able to create a functional model that passes the tests we have created and fulfills the properties that were required of it. Furthermore, the properties have been formulated in LTL and verified using Maude's model-checker, verifying some security and liveliness properties, which are beyond what the extent of the reachability analysis can provide and guarantee a more faithful and correct behavior of the system.

There have been many difficulties encountered during the development of this work. Firstly, there had to be a lot of time dedicated to learning the Maude language from scratch. This took a significant amount of time to not only assimilate, but to practice before creating the model as well. This led to many hitches during the production of this work as well as the need to rework the model over and over until a definitive model was made. The stage of verification proved to be an even bigger challenge due to time constraints added to the steep learning curve it presented.

In terms of improving this work, there are several steps that could be taken. More properties could be analyzed with the mode-checker and more subtle Reddit characteristics could be added. They were deemed unnecessary due to exceeding the scope of a project like this but could be added in a later work. In addition, a more exhaustive analysis could be achieved with the use of `[narrowing]` in the verification process. These are all proposals that could be added to a future work.

Lastly, on a more personal note, we must acknowledge that although the field of expertise studied did not pertain to this particular branch of learning, as I have studied the networking branch, this project has allowed for an enlightening learning experience within this particular field of knowledge. This switch in field was hard to adapt to as Maude proved that, although the core concepts are easy to grasp, the depth and flexibility it has proved too steep a learning curve at certain points in the making of this project. That being said this project has been able to provide an extensive amount of knowledge in a short period of time. Here we include the practical use and theory of Maude language, language theory, model checking, temporal logic, and the expression and analysis

of properties. I now have a desire to expand my knowledge in this field further, as I think it is not only useful, but crucial in certain fields.

Bibliography

[1] M. Alpuente, D. Ballis, F. Frechina, J. Sapiña. Exploring Conditional Rewriting Logic Computations. *Journal of Symbolic Computation*, 69:3-39, 2015. Last consulted, July 2021. Article consulted at https://riunet.upv.es/handle/10251/61030. Tool consulted at http://safe-tools.dsic.upv.es/anima/.

[2] C. Baier and J. P. Katoen. *Principles of Model Checking*. MIT Press, January 2008.

[3] E. M. Clarke, E. A. Emerson, J. Sifakis. *Model Checking: Algorithmic Verification and Debugging*. Communications of the ACM, November 2009, vol. 52, no 11.

[4] R. Hanneman and M. Riddle. *Introduction to Social Network Methods*. Riverside, CA: University of California, Riverside. Can be consulted at https://faculty.ucr.edu/~hanneman/nettext/index.html.

[5] R. Butler. *Langley Formal Methods Program. What is Formal Methods*. Last consulted, June 2021. Consulted at https://shemesh.larc.nasa.gov/fm/fm-what.html.

[6] cbsnews.com. *How Reddit posters made millions as Wall Street lost billions on GameStop's wild stock ride*. Last consulted, June 2021. Consulted at https://www.cbsnews.com/news/gamestop-reddit-wallstreetbets-short-squeeze-2021-01-28/

[7] M. Clavel, F. Durán, S. Eker, S. Escobar, P.Lincoln, N. Martí-Oliet, J. Messeguer, C. Talcott. *Maude Manual (Version 3.1)*. Last consulted, July 2021. Consulted at http://maude.lcc.uma.es/maude31-manual-html/maude-manual.html.

[8] M. Clavel, F. Durán, S. Eker, S. Escobar, P.Lincoln, N. Martí-Oliet, J. Messeguer, C. Talcott. *The Maude System*. Last consulted, April 2021. Consulted at http://maude.cs.illinois.edu/w/index.php/The_Maude_System.

[9] Encyclopedia of Mathematics. *Formal language - Encyclopedia of Mathematics*. Last consulted, June 2021. Consulted at https://encyclopediaofmath.org/index.php?title=Formal_language.

[10] Reddit.com. *reddit.com: api documentation*. Last consulted, April 2021. Consulted at https://www.reddit.com/dev/api.

APPENDIX A
Appendices

A.1 The Maude deductive model for the Reddit Network

```
fmod REDDIT is
    pr STRING .
    pr NAT-LIST .

    *** State declaration ***
    sort State .

    *** User declaration ***
    sorts User UserSet .
    subsort User < UserSet .

    op none : -> UserSet [ctor] .
    op __ : UserSet UserSet -> UserSet [assoc comm id: none] .

    *** SubReddit declaration ***
    sorts SubReddit SubRedditSet .
    subsort SubReddit < SubRedditSet .

    op none : -> SubRedditSet [ctor] .
    op __ : SubRedditSet SubRedditSet -> SubRedditSet [assoc comm id: none]
        .

    *** Post declaration **
    sorts Post PostSet .
    subsort Post < PostSet .

    op none : -> PostSet [ctor] .
    op __ : PostSet PostSet -> PostSet [assoc comm id: none] .

    *** Vote declaration ***
    sorts Vote VoteSet .
    subsort Vote < VoteSet .

    op none : -> VoteSet [ctor] .
    op __ : VoteSet VoteSet -> VoteSet [assoc comm id: none] .

    *** Constructors ***
    op user : Nat String NatList NatList NatList -> User [ctor] . --- Id,
       Name, Subscribed SubReddits, Created Posts, Obtained total Votes (
       Total Karma)
    op subr : Nat String String NatList NatList -> SubReddit [ctor] . ---
       Id, Name, Description, Users subscribed to SubReddit, Posts in
       SubReddit
```

```
39      op post : Nat String String Nat NatList NatList -> Post [ctor] . --- Id
            , Title, Text, User ID (creator), Comments underneath, Votes on
            post
40      op vote : Nat Nat Nat Bool -> Vote [ctor] . --- Id, User who voted,
            Post where vote belongs, Value(positive or negative)
41
42  endfm
43
44  fmod REDDIT-EQ is
45      protecting REDDIT .
46
47      *** Actions ***
48      sorts Action ActionList .
49      subsort Action < ActionList .
50
51      --- Create an ActionList that will determine what actions can be
            performed in the State
52      op nil : -> ActionList [ctor] .
53      op _,_ : ActionList ActionList -> ActionList [assoc id: nil] .
54
55      --- Create a User (Add User to Reddit)
56      op createuser : String -> Action . --- Username
57      --- Delete a User (Delete User from Reddit)
58      op deleteuser : Nat -> Action . --- Post ID
59      --- Create a SubReddit (Add User to Reddit)
60      op createsub : Nat String String -> Action . --- User ID, SubReddit's
            Name, SubReddit's Description
61      --- Delete a SubReddit (Delete SubReddit from Reddit)
62      op deletesub : Nat -> Action . --- SubReddit ID
63      --- Create a Post (Add Post to User and SubReddit's PostLists)
64      op createpost : Nat Nat String String -> Action . --- User ID,
            SubReddit ID, Title, Text
65      --- Delete a Post (Delete Post from User and SubReddit's PostLists and
            delete all comments and votes made on that post)
66      op deletepost : Nat -> Action . --- Post ID
67      --- Subscribe User to SubReddit (User add SubReddit to their list of
            SubReddits and vice versa)
68      op subscribe : Nat Nat -> Action . --- User ID, SubReddit ID
69      --- Unsubscribe User (User remove SubReddit from their list of
            SubReddits and vice versa)
70      op unsubscribe : Nat Nat -> Action . --- User ID, SubReddit ID
71      --- Create a Comment (Add Post to Post's list of Comments)
72      op reply : Nat Nat String -> Action . --- User ID, Post ID, Text
73      --- Delete a Comment (Remove Post from Post's list of Comments)
74      op deletereply : Nat -> Action . --- Comment(Post) ID
75      --- Delete a Comment (Remove Post from Post's list of Comments)
76      op deletereplycascade : Nat -> Action . --- Comment(Post) ID
77      --- Cast a Vote on a Post (Add Vote to User and Post's lists of votes)
78      op addvote : Nat Nat Bool -> Action . --- User ID, Post ID, Upvote or
            Downvote
79      --- Delete a Vote from a Post (Delete Vote from User and Post's lists
            of votes)
80      op deletevote : Nat -> Action . --- Vote ID
81
82      *** Operators ***
83      op deletecascadeP : NatList -> ActionList .
84      op deletecascadeV : NatList -> ActionList .
85      op deleteall : NatList -> ActionList .
86
87      *** Variables ***
88      var Us : User .
89      var Sub : SubReddit .
90      var P : Post .
91      var V : Vote .
```

```
92      var N : Nat .
93      var NL : NatList .
94
95      *** Axioms ***
96      *** Idempotency ***
97      eq Us Us = Us .
98      eq Sub Sub = Sub .
99      eq P P = P .
100     eq V V = V .
101
102     *** Equations ***
103     eq deletecascadeP(nil) = nil .
104     eq deletecascadeP(N NL) = deletereplycascade(N) , deletecascadeP(NL) .
105
106     eq deletecascadeV(nil) = nil .
107     eq deletecascadeV(N NL) = deletereplycascade(N) , deletecascadeV(NL) .
108
109     eq deleteall(nil) = nil .
110     eq deleteall(N NL) = deletepost(N), deleteall(NL) .
111
112 endfm
113
114 mod REDDIT-RL is
115     protecting REDDIT-EQ .
116
117     *** Variables ***
118     --- For rules ---
119     vars T Txt T' Txt' UName SName SDesc : String . --- Variables for Title
            , Text, Username, SubReddit description
120     vars UList SList PList CList VList KList : NatList . --- Lists
121     vars UList' SList' PList' CList' VList' KList' : NatList . --- Lists'
122     vars UList'' SList'' PList'' CList'' VList'' KList'' : NatList . ---
            Lists''
123     vars UList''' SList''' PList''' CList''' VList''' KList''' : NatList .
            --- Lists'''
124     vars UId SId PId CId VId : Nat . --- IDs
125     vars UId' SId' PId' CId' VId' : Nat . --- ID's
126     vars X CNT : Nat . --- Counter for creating IDs
127     var B : Bool . --- Boolean to Upvote or Downvote
128     --- Sets ---
129     var USet : UserSet .
130     var SSet : SubRedditSet .
131     var PSet : PostSet .
132     var VSet : VoteSet .
133     var AList : ActionList .
134
135     *** Initial State ***
136     op _|_|_|_|_|_ : Nat UserSet SubRedditSet PostSet VoteSet ActionList ->
            State [ctor] .
137
138     *** Sequences as seen in the TFG paper ***
139     --- Initial State
140     op init : -> State .
141     eq init = 0 | none | none | none | none | aListInit , aListInter1 .
142     --- Initial Reddit actions
143     ---7 initial actions(0-6)
144     op aListInit : -> ActionList .
145     eq aListInit = createuser("u/aLynx") , createuser("u/b0bbyPR") ,
            createuser("u/chocoSystems") , createuser("u/dedTech") , createsub
            (0, "r/Maude", "Lorem ipsum") , createsub(0, "r/ModelChecking", "
            Lorem ipsum") , createsub(0, "r/FormalMethods", "Lorem ipsum") .
146
147     *** Reddit action interactions ***
148     op aListInter1 : -> ActionList .
```

```
    eq aListInter1 = createpost(0, 6, "Conference", "Conference link") ,
        reply(1, 7, "Good job!") , deletepost(7) .
    op aListInter2 : -> ActionList .
    eq aListInter2 = createpost(0, 6, "Maude is cool", "Body of text") ,
        reply(1, 7, "Nice") , reply(2, 8, "Delete this") , deletereply(8) .
    op aListInter3 : -> ActionList .
    eq aListInter3 = createpost(0, 6, "Maude post", "Bottom text") ,
        addvote(1, 7, true) , deletevote(8) .

    *** Rules ***
    rl [createuser] :
        CNT |
        USet |
        SSet |
        PSet |
        VSet |
        createuser(UName) , AList
        =>
        CNT + 1 |
        USet user(CNT, UName, nil, nil, nil) |
        SSet |
        PSet |
        VSet |
        AList [narrowing] .

    rl [deleteuser] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet subr(SId, SName, SDesc, UList UId UList', PList') |
        PSet |
        VSet |
        deleteuser(UId) , AList
        =>
        CNT |
        USet |
        SSet subr(SId, SName, SDesc, (UList UList'), PList') |
        PSet |
        VSet |
        AList [narrowing] .
-------------------------------------------------------------------------
    rl [createsub] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet |
        PSet |
        VSet |
        createsub(UId, SName, SDesc) , AList
        =>
        CNT + 1 |
        USet user(UId, UName, SList, PList, KList) |
        SSet subr(CNT, SName, SDesc, UId, nil)|
        PSet |
        VSet |
        AList [narrowing] .

    rl [deletesub] :
        CNT |
        USet |
        SSet subr(SId, SName, SDesc, UList, PList) |
        PSet |
        VSet |
        deletesub(SId) , AList
        =>
        CNT |
```

```
        USet |
        SSet |
        PSet |
        VSet |
        deleteall(PList) , AList [narrowing] .
---------------------------------------------------------------------------
    rl [createpost] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet subr(SId, SName, SDesc, UList, PList') |
        PSet |
        VSet |
        createpost(UId, SId, T, Txt) , AList
        =>
        CNT + 1 |
        USet user(UId, UName, SList, (PList CNT), KList) |
        SSet subr(SId, SName, SDesc, UList, (PList' CNT)) |
        PSet post(CNT, T, Txt, UId, nil, nil) |
        VSet |
        AList [narrowing] .

    rl [deletepost] :
        CNT |
        USet user(UId, UName, SList, PList PId PList', KList) |
        SSet subr(SId, SName, SDesc, UList, PList'' PId PList''') |
        PSet post(PId, T, Txt, UId, CList, VList) |
        VSet |
        deletepost(PId) , AList
        =>
        CNT |
        USet user(UId, UName, SList, (PList PList'), KList) |
        SSet subr(SId, SName, SDesc, UList, (PList'' PList''')) |
        PSet |
        VSet |
        deletecascadeP(CList) , deletecascadeV(VList) , AList [narrowing] .
---------------------------------------------------------------------------
    rl [subscribe] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet subr(SId, SName, SDesc, UList, PList') |
        PSet |
        VSet |
        subscribe(UId, SId) , AList
        =>
        CNT |
        USet user(UId, UName, (SList SId), PList, KList) |
        SSet subr(SId, SName, SDesc, (UList UId), PList') |
        PSet |
        VSet |
        AList [narrowing] .

    rl [unsubscribe] :
        CNT |
        USet user(UId, UName, SList SId SList', PList, KList) |
        SSet subr(SId, SName, SDesc, UList UId UList', PList') |
        PSet |
        VSet |
        unsubscribe(UId, SId) , AList
        =>
        CNT |
        USet user(UId, UName, (SList SList'), PList, KList) |
        SSet subr(SId, SName, SDesc, (UList UList'), PList') |
        PSet |
        VSet |
```

```
274         AList [narrowing] .
275 ------------------------------------------------------------------------
276     rl [reply] :
277         CNT |
278         USet user(UId, UName, SList, PList, KList) |
279         SSet |
280         PSet post(PId, T, Txt, UId', CList, VList) |
281         VSet |
282         reply(UId, PId, Txt') , AList
283         =>
284         CNT + 1 |
285         USet user(UId, UName, SList, (PList CNT), KList) |
286         SSet |
287         PSet post(CNT, T, Txt', UId, nil, nil) post(PId, T, Txt, UId', (
                CList CNT), VList) |
288         VSet |
289         AList [narrowing] .
290
291     rl [deletereply] :
292         CNT |
293         USet user(UId, UName, SList, PList PId PList', KList) |
294         SSet |
295         PSet post(PId, T, Txt, UId, CList, VList) post(PId', T', Txt', UId'
                , CList' PId CList'', VList') |
296         VSet |
297         deletereply(PId) , AList
298         =>
299         CNT |
300         USet user(UId, UName, SList, (PList PList'), KList) |
301         SSet |
302         PSet post(PId', T', Txt', UId', (CList' CList''), VList') |
303         VSet |
304         deletecascadeP(CList) , deletecascadeV(VList) , AList [narrowing] .
305
306     rl [deletereplycascade] :
307         CNT |
308         USet user(UId, UName, SList, PList PId PList', KList) |
309         SSet |
310         PSet post(PId, T, Txt, UId, CList, VList) |
311         VSet |
312         deletereplycascade(PId) , AList
313         =>
314         CNT |
315         USet user(UId, UName, SList, (PList PList'), KList) |
316         SSet |
317         PSet |
318         VSet |
319         deletecascadeP(CList) , deletecascadeV(VList) , AList [narrowing] .
320 ------------------------------------------------------------------------
321     op hasVoted? : VoteSet Nat Nat -> Bool .
322     eq hasVoted?(vote(VId, UId, PId, B) VSet, UId, PId) = true .
323     eq hasVoted?(VSet, UId, PId) = false [owise] .
324
325     rl [addvote] :
326         CNT |
327         USet user(UId, UName, SList, PList, KList) |
328         SSet |
329         PSet post(PId, T, Txt, UId', CList, VList) |
330         VSet |
331         addvote(UId, PId, B) , AList
332         =>
333         if hasVoted?(VSet, UId, PId)
334         then
335             CNT |
```

```
            USet user(UId, UName, SList, PList, KList) |
            SSet |
            PSet post(PId, T, Txt, UId', CList, VList) |
            VSet |
            AList
        else
            CNT + 1 |
            USet user(UId, UName, SList, PList, (KList CNT)) |
            SSet |
            PSet post(PId, T, Txt, UId, CList, (VList CNT)) |
            VSet vote(CNT, UId, PId, B) |
            AList
        fi [narrowing] .

    rl [deletevote] :
        CNT |
        USet user(UId, UName, SList, PList, KList VId KList') |
        SSet |
        PSet post(PId, T, Txt, UId, CList, VList VId VList') |
        VSet |
        deletevote(VId) , AList
        =>
        CNT |
        USet user(UId, UName, SList, PList, (KList KList')) |
        SSet |
        PSet post(PId, T, Txt, UId, CList, (VList VList')) |
        VSet |
        AList [narrowing] .

endm
```

A.2 The Maude operational model for the Reddit Network

```
fmod REDDIT is
    pr CONVERSION .
    pr NAT-LIST .

    *** State declaration ***
    sort State .

    *** User declaration ***
    sorts User UserSet .
    subsort User < UserSet .

    op none : -> UserSet [ctor] .
    op __ : UserSet UserSet -> UserSet [assoc comm id: none] .

    *** SubReddit declaration ***
    sorts SubReddit SubRedditSet .
    subsort SubReddit < SubRedditSet .

    op none : -> SubRedditSet [ctor] .
    op __ : SubRedditSet SubRedditSet -> SubRedditSet [assoc comm id: none]
        .

    *** Post declaration **
    sorts Post PostSet .
    subsort Post < PostSet .

    op none : -> PostSet [ctor] .
    op __ : PostSet PostSet -> PostSet [assoc comm id: none] .

```

```
29      *** Vote declaration ***
30      sorts Vote VoteSet .
31      subsort Vote < VoteSet .
32
33      op none : -> VoteSet [ctor] .
34      op __ : VoteSet VoteSet -> VoteSet [assoc comm id: none] .
35
36      *** Constructors ***
37      op user : Nat String NatList NatList NatList -> User [ctor] . --- Id,
            Name, Subscribed SubReddits, Created Posts, Obtained total Votes (
            Total Karma)
38      op subr : Nat String String NatList NatList -> SubReddit [ctor] . ---
            Id, Name, Description, Users subscribed to SubReddit, Posts in
            SubReddit
39      op post : Nat String String Nat NatList NatList -> Post [ctor] . --- Id
            , Title, Text, User ID (creator), Comments underneath, Votes on
            post
40      op vote : Nat Nat Nat Bool -> Vote [ctor] . --- Id, User who voted,
            Post where vote belongs, Value(positive or negative)
41
42      *** Variables ***
43      var Us : User .
44      var Sub : SubReddit .
45      var P : Post .
46      var V : Vote .
47      var N : Nat .
48      var NL : NatList .
49
50      *** Axioms ***
51      *** Idempotency ***
52      eq Us Us = Us .
53      eq Sub Sub = Sub .
54      eq P P = P .
55      eq V V = V .
56
57  endfm
58
59  mod REDDIT-RL is
60      protecting REDDIT .
61
62      *** Variables ***
63      --- For rules ---
64      vars T Txt T' Txt' UName SName SDesc : String . --- Variables for Title
            , Text, Username, SubReddit description
65      vars UList SList PList CList VList KList : NatList . --- Lists
66      vars UList' SList' PList' CList' VList' KList' : NatList . --- Lists'
67      vars UList'' SList'' PList'' CList'' VList'' KList'' : NatList . ---
            Lists''
68      vars UList''' SList''' PList''' CList''' VList''' KList''' : NatList .
            --- Lists'''
69      vars UId SId PId CId VId : Nat . --- IDs
70      vars UId' SId' PId' CId' VId' : Nat . --- ID's
71      vars X CNT : Nat . --- Counter for creating IDs
72      var B : Bool . --- Boolean to Upvote or Downvote
73      --- Sets ---
74      var USet : UserSet .
75      var SSet : SubRedditSet .
76      var PSet : PostSet .
77      var VSet : VoteSet .
78
79      *** Initial State ***
80      op _|_|_|_|_ : Nat UserSet SubRedditSet PostSet VoteSet -> State [ctor]
            .
81
```

```
    *** Sequences as seen in the TFG paper ***
    --- Initial State
    op init : -> State .
    eq init = 0 | none | none | none | none .

    *** Rules ***
    rl [createuser] :
        CNT |
        USet |
        SSet |
        PSet |
        VSet
        =>
        CNT + 1 |
        USet user(CNT, "u/User" + string(CNT, 10), nil, nil, nil) |
        SSet |
        PSet |
        VSet [narrowing] .

    rl [deleteuser] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet subr(SId, SName, SDesc, UList UId UList', PList') |
        PSet |
        VSet
        =>
        CNT |
        USet |
        SSet subr(SId, SName, SDesc, (UList UList'), PList') |
        PSet |
        VSet [narrowing] .
-------------------------------------------------------------------------
    rl [createsub] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet |
        PSet |
        VSet
        =>
        CNT + 1 |
        USet user(UId, UName, SList, PList, KList) |
        SSet subr(CNT, "r/SUB" + string(CNT, 10), "Lorem ipsum", UId, nil)|
        PSet |
        VSet [narrowing] .

    rl [deletesub] :
        CNT |
        USet |
        SSet subr(SId, SName, SDesc, UList, PList) |
        PSet |
        VSet
        =>
        CNT |
        USet |
        SSet |
        PSet |
        VSet [narrowing] .
-------------------------------------------------------------------------
    rl [createpost] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet subr(SId, SName, SDesc, UList, PList') |
        PSet |
        VSet
```

```
        =>
        CNT + 1 |
        USet user(UId, UName, SList, (PList CNT), KList) |
        SSet subr(SId, SName, SDesc, UList, (PList' CNT)) |
        PSet post(CNT, "Title" + string(CNT, 10), "Text", UId, nil, nil) |
        VSet [narrowing] .

    rl [deletepost] :
        CNT |
        USet user(UId, UName, SList, PList PId PList', KList) |
        SSet subr(SId, SName, SDesc, UList, PList'' PId PList''') |
        PSet post(PId, T, Txt, UId, CList, VList) |
        VSet
        =>
        CNT |
        USet user(UId, UName, SList, (PList PList'), KList) |
        SSet subr(SId, SName, SDesc, UList, (PList'' PList''')) |
        PSet |
        VSet [narrowing] .

--------------------------------------------------------------------------
    rl [subscribe] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet subr(SId, SName, SDesc, UList, PList') |
        PSet |
        VSet
        =>
        CNT |
        USet user(UId, UName, (SList SId), PList, KList) |
        SSet subr(SId, SName, SDesc, (UList UId), PList') |
        PSet |
        VSet [narrowing] .

    rl [unsubscribe] :
        CNT |
        USet user(UId, UName, SList SId SList', PList, KList) |
        SSet subr(SId, SName, SDesc, UList UId UList', PList') |
        PSet |
        VSet
        =>
        CNT |
        USet user(UId, UName, (SList SList'), PList, KList) |
        SSet subr(SId, SName, SDesc, (UList UList'), PList') |
        PSet |
        VSet [narrowing] .
--------------------------------------------------------------------------
    rl [reply] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet subr(SId, SName, SDesc, UList, PList') |
        PSet post(PId, T, Txt, UId', CList, VList) |
        VSet
        =>
        CNT + 1 |
        USet user(UId, UName, SList, (PList CNT), KList) |
        SSet subr(SId, SName, SDesc, UList, (PList' CNT)) |
        PSet post(CNT, T, "Txt", UId, nil, nil) post(PId, T, Txt, UId', (
            CList CNT), VList) |
        VSet [narrowing] .

    rl [deletereply] :
        CNT |
        USet |
```

```
        SSet |
        PSet post(PId, T, Txt, UId, CList, VList) |
        VSet
        =>
        CNT |
        USet |
        SSet |
        PSet |
        VSet [narrowing] .

------------------------------------------------------------------------
    op hasVoted? : VoteSet Nat Nat -> Bool .
    eq hasVoted?(vote(VId, UId, PId, B) VSet, UId, PId) = true .
    eq hasVoted?(VSet, UId, PId) = false [owise] .
    rl [upvote] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet |
        PSet post(PId, T, Txt, UId', CList, VList) |
        VSet
        =>
        if hasVoted?(VSet, UId, PId)
        then
            CNT |
            USet user(UId, UName, SList, PList, KList) |
            SSet |
            PSet post(PId, T, Txt, UId', CList, VList) |
            VSet
        else
            CNT + 1 |
            USet user(UId, UName, SList, PList, (KList CNT)) |
            SSet |
            PSet post(PId, T, Txt, UId, CList, (VList CNT)) |
            VSet vote(CNT, UId, PId, true)
        fi [narrowing] .

    rl [downvote] :
        CNT |
        USet user(UId, UName, SList, PList, KList) |
        SSet |
        PSet post(PId, T, Txt, UId', CList, VList) |
        VSet
        =>
        if hasVoted?(VSet, UId, PId)
        then
            CNT |
            USet user(UId, UName, SList, PList, KList) |
            SSet |
            PSet post(PId, T, Txt, UId', CList, VList) |
            VSet
        else
            CNT + 1 |
            USet user(UId, UName, SList, PList, (KList CNT)) |
            SSet |
            PSet post(PId, T, Txt, UId, CList, (VList CNT)) |
            VSet vote(CNT, UId, PId, false)
        fi [narrowing] .

    rl [deletevote] :
        CNT |
        USet user(UId, UName, SList, PList, KList VId KList') |
        SSet |
        PSet post(PId, T, Txt, UId', CList, VList VId VList') |
        VSet
```

```
        =>
        CNT |
        USet user(UId, UName, SList, PList, (KList KList')) |
        SSet |
        PSet post(PId, T, Txt, UId', CList, (VList VList')) |
        VSet [narrowing] .

endm
mod REDDIT-PREDS is
    pr REDDIT-RL .
    including SATISFACTION .

    *** Configuration (State) ***
    sort Reddit .
    subsort Reddit < State .

    *** Variables ***
    var CNT : Nat .
    var USet : UserSet .
    var SSet : SubRedditSet .
    var PSet : PostSet .
    var VSet : VoteSet .
    var R : Reddit .
    --- For rules ---
    vars T Txt T' Txt' UName SName SDesc : String . --- Variables for Title
        , Text, Username, SubReddit description
    vars UList SList PList CList VList KList : NatList . --- Lists
    vars UList' SList' PList' CList' VList' KList' : NatList . --- Lists'
    vars UList'' SList'' PList'' CList'' VList'' KList'' : NatList . ---
        Lists''
    vars UList''' SList''' PList''' CList''' VList''' KList''' : NatList .
        --- Lists'''
    vars UId SId PId CId VId : Nat . --- IDs
    vars UId' SId' PId' CId' VId' : Nat . --- ID's
    var B : Bool . --- Boolean to Upvote or Downvote

    *** Operators ***
    op vote : Nat -> Prop .
    op user : Nat -> Prop .
    op post : Nat -> Prop .

    *** Equations ***
    --- All existing posts will continue existing as long as they are not
        explicitly deleted.
    --- postexists(N:Nat, U:Nat) => [] (postexists(N:Nat, U:Nat) W
        deletePost(NP))
    ---eq < CNT | USet | SSet | PSet post(PId, T, Txt, UId, CList, VList) |
         VSet  > |= postexists(PId, UId) = true .
    ---eq < CNT | USet | SSet | PSet | VSet  > |= userexists(UId) = true .


    --- Every vote is always associated to its corresponding post
    eq < CNT | USet | SSet | PSet | VSet vote(VId, UId, PId, B)  > |= vote(
        VId) = true .
    eq < CNT | USet | SSet | PSet post(PId, T, Txt, UId', CList, VList VId
        VList') | VSet  > |= post(VId) = true .

endm
mod REDDIT-CHECK is
    protecting REDDIT-PREDS .
    including MODEL-CHECKER .
    including LTL-SIMPLIFIER .

    *** Variables ***
```

```
329     var CNT : Nat .
330     var USet : UserSet .
331     var SSet : SubRedditSet .
332     var PSet : PostSet .
333     var VSet : VoteSet .
334     var R : Reddit .
335 
336     --- For rules ---
337     vars T Txt T' Txt' UName SName SDesc : String . --- Variables for Title
            , Text, Username, SubReddit description
338     vars UList SList PList CList VList KList : NatList . --- Lists
339     vars UList' SList' PList' CList' VList' KList' : NatList . --- Lists'
340     vars UList'' SList'' PList'' CList'' VList'' KList'' : NatList . ---
            Lists''
341     vars UList''' SList''' PList''' CList''' VList''' KList''' : NatList .
            --- Lists'''
342     vars UId SId PId CId VId : Nat . --- IDs
343     vars UId' SId' PId' CId' VId' : Nat . --- ID's
344     var B : Bool . --- Boolean to Upvote or Downvote
345 
346 
347     ---op initConfig1 : -> Reddit .
348     ---eq initConfig1 = < 0 | none | none | none | none  > .
349 
350 endm
```