Practical Course – Contributing to an Open-Source Project

Project Report on Rust-Lang/Chalk

Florian Sextl[1]

[1]Technical University of Munich, sextl@in.tum.de

1 Introduction

Free/Libre and Open Source Software (FLOSS) describes software that comes with a certain set of freedoms guaranteed through a license. A large part of software used in todays IT infrastructure and personal devices follows the FLOSS ideas and is developed by Open Source communities. The practical course "Contributing to an Open-Source Project" [17] aims to teach the participants the importance of FLOSS principals and gives them all information necessary to successfully contribute to or even start own Open Source projects. As such, the practical course concentrates on the basics of FLOSS and requires the participants to apply this knowledge by contributing to an Open Source project they chose beforehand.

In this final project report I document and assess the contributions I made to the Chalk project and summarize the most important conclusions derived throughout the practical course. The Chalk project is introduced with a short profile in Section 2. After that section, a high-level description of the contributions to the Chalk project follows in Section 3. Subsequently, the project work is evaluated with respect to Free/Libre and Open Source Software best practices in Section 4. Lastly, Section 5 describes the lessons learned throughout the practical part of the course and related observations regarding the Chalk project.

2 Brief Profile of Chalk

The Rust programming language is a systems programming language that focuses on performance and safety [27]. One central feature of Rust are so called traits. Traits can be used to define certain properties about a type, e.g. that it can be printed for debugging purposes. As such, traits play a similar role as interfaces in Java or type classes in Haskell. Listing 1 shows an example trait `Debug` (that is part of the Rust standard library as `std::fmt::Debug`) which is implemented by a generic type `Point`. Every type that wants to implement `Debug` needs to provide its own implementation of the `fmt` method. `Point` implements `Debug` for all type arguments `T` that themselves implement `Debug` (i.e. `T: Debug`). Thus, it can use their `fmt` implementations as part of its own. In the example, `fmt` is called on a `Point` with type argument `u8` (i.e. the builtin type for 8 bit unsigned integers) by printing it with the `println!` macro, which uses `fmt` internally. Thus, `u8` must implement the `Debug` trait for this program to compile. In general, the compiler has to check whether a given type fulfils all trait constraints. This check is modeled as the Rust trait system and can be utilized by both the type checking and the type inference facilities of the compiler.

The Chalk project is a part of the Rust programming language development process and aims at developing a "library that implements the Rust trait system, based on Prolog-ish logic rules" [6], which is meant to replace the current trait system of the Rust compiler Rustc in the future. The Chalk library models trait constraints like `T: Debug` from the example Listing 1 as logic clauses and builds a logic program from these clauses. The so generated program can then be queried to find out, for example, whether a certain type can be used as a type argument in a specific place or whether there exists a type that could be used in this place. An example query could be whether for all types `T` a `Pair<T>` of this type would implement the `Debug` trait (in a rather mathematical notation):

$$\forall\ \texttt{T. Point<T>: Debug}$$

This is obviously false, as `Point` implements `Debug` only for types that implement `Debug` themselves. The clauses generated from the Rust trait system use quantifiers to model, for example, generic type parameters[1] (they are therefore first-order Hereditary Harrop clauses [29]), whereas standard Prolog solvers work only with Horn clauses which can't contain quantifiers.

As a part of the Rust development process, the Chalk project is maintained by the Rust programming language community (Rust-Lang for short) and a subsidiary of the Rust Foundation [24]. To be precise, Chalk is maintained by the traits working group, which itself is lead by members of the Rust-Lang compiler team. A more detailed description of the governance for the Chalk project can be found in the preliminary report on this course participation [28].

Chalk is mainly developed as part of the "library-ification" [cf. 16, 30] process of the Rustc compiler[2] and provides a model of the Rust trait system that can be used by the compiler and other language tools as well. One prominent example for such a tool is the

[1]The implementation type constraints would be lowered to this clause in Chalk's notations:

$$\texttt{forall<T>}\ \{\ \texttt{Pair<T>: Debug}\ \text{:-}\ \texttt{T: Debug}\ \}$$

[2]The idea behind this process is to extract models of central language components like the trait system (cf. Chalk) or the borrow checker (cf. Polonius [7]) into shared libraries. With this modularization, the compiler itself becomes less complex and the shared libraries can be used with a variety of projects to provide a common basis.

```rust
trait Debug {
    // Formats self as debug output
    fn fmt(&self, f: &mut Formatter<'_>) -> Result<(), Error>;
}

struct Point<T> {
    x: T,
    y: T,
}

impl<T: Debug> Debug for Point<T> {
    fn fmt(&self, f: &mut Formatter<'_>) -> Result<(), Error>{
        self.x.fmt(f)?;
        self.y.fmt(f)
    }
}

fn main(){
    let pair: Point<u8> = Point { x: 0, y: 1 };
    // Prints "The debug output is: 01"
    println!("The debug output is: {:?}", pair);
}
```

Listing 1: Traits example.

compiler frontend and language server Rust-Analyzer [26], that utilizes Chalk as part of its type checker [cf. 1].

Thus far, Chalk is not feature complete with regard to Rustc and includes two different incomplete solvers for the Hereditary Harrop clauses (i.e. the SLG solver and the recursive solver). Both solvers have their respective strengths and weaknesses and are still developed in parallel as none of the two is sophisticated enough yet to replace the current system [cf. 1]. As an example, the Rust-Analyzer utilizes only the recursive solver because it avoids a normalization issue the SLG solver is currently not capable of solving [cf. 9]. Despite its incompleteness, Chalk is also partially integrated into the unstable development version of Rustc [cf. 30].

3 Contributions to Chalk

The project phase of the practical course "Contributing to an Open-Source Project" lasted circa 15 weeks, in which the participants worked on and contributed to an Open Source project. In this period of time, I worked on the Chalk project and authored several contributions. These contributions are listed in Table 1 and are described alongside the most important aspects of the whole project work in the following section.

3.1 Timeline and Work Focus

Before the project phase, the practical course started with a theoretical part about the basics of Free/Libre and Open Source Software and the best practices in its development. After this initial phase, the course participants compiled preliminary reports about the projects they would work on [cf. 28]. After the preliminary reports were submitted on 2020-12-01, the course entered the project phase (cf. Table 1).

I started my work on the Chalk project by introducing myself and the practical course in the designated #new-members stream on the official Rust-Lang Zulip server [8] on 2020-12-04. At the same time, I started looking deeper into the inner workings of Chalk to get a better understanding of how Chalk is supposed to work and how this is achieved. To that end, I read through the Chalk book [29], looked through the source code, read the documentation there and asked some initial questions regarding the general structure on Zulip [5, 2]. Throughout this research process, several small problems with the documentation attracted my attention. These could be solved by the pull requests #663, #668, #674 and #675 [18, 19, 20, 21].

After I grew accustomed to the Chalk project's structure, I started looking for issues to work on. For this, I focused on bug reports as the search for the bugs' causes would provide a good opportunity to learn about the implementation in greater detail. The reported bugs were either related to infinite loops or wrong unification of goal clauses [12, 11]. I reproduced the bugs, investigated what happens and where the bugs are introduced and shared my findings with the working group [1, 3]. Due to a lack of deeper understanding to what causes these bugs and how they could be fixed, I refrained from working on patches for them.

Instead, I concentrated on gaining a better understanding of the recursive solver, for which the aforementioned bugs occurred. For this purpose, I looked into a missing feature in the recursive solver that would allow to handle specific corner cases correctly. This feature was described alongside a rough implementation sketch in issue #399 [10]. For the last weeks of the project phase, I developed three different implementation approaches based on the idea from the issue. In the end, I opened a pull request for the most sophisticated solution [23]. The whole contribution process is described in more detail in Section 3.3.

3.2 Communication about Chalk

The primary communication platform for exchange about the development of Rust-Lang projects is the Rust-Lang Zulip instance [25]. Thus, all discussions about the Chalk project are meant to happen in the designated #wg-traits stream. Accordingly, I opened dedicated threads for all general questions about Chalk in this stream. In these threads, I asked about the development status, the purpose of specific components and historic decisions about Chalk [5, 1, 2, accordingly]. In addition to these question threads, I also shared and discussed detailed debugging results with the wg-trait members [3, 1]. Answers to both questions and debugging results were given in most cases in less than one day, only in some cases it took a week or more to get an answer.

In contrast to the more extensive discussion threads in the #wg-traits Zulip stream, there were little to no discussions on the pull requests I issued through the course of the project phase. This is due to the fact that the contents of the pull requests were either already discussed in a Zulip thread, or simple enough that the reviewers had no need for any sort of additional exchange about them.

Besides Zulip threads dedicated to a specific topic and comments on GitHub issues and pull requests, the main form of exchange within the traits working group was weekly textual meetings. These meetings were held as Zulip conversation threads and were used to cooperate on issue triages and to coordinate development efforts in the near future [32, 33, 35, 36, 38, inter alia]. Although the traits working group maintains Chalk, not all weekly meetings were focused on the development of Chalk, as the working group is responsible for all trait system related topics. Thus, I only participated seldomly in the meeting discussions, albeit I attended all meetings and always followed them to the end.

3.3 A Contribution by Example

My final contribution in the project phase was to propose a feature addition to the recursive solver. This feature was aimed at improving the handling of coindutive cycles (i.e. goals that depend on each other cyclicly). The current way of handling these cycles could lead to false positive solutions for goals in rare edge cases. In these cases, positive results are cached before they are proved to be sound and can thus falsify other results. This shortcoming was described in issue #399 [10] alongside some solution ideas.

I started my work on this issue with examining why the problem arises and which parts of the recursive solver are involved. Based on this information, I developed a first implementation and opened a draft pull request for it [22]. Although this first implementation sufficed to solve all problematic test cases, I decided to develop a second implementation. The two implementation approaches differ in how much code they change and how they achieve the new feature. Whereas the first implementation is based on exploiting an existent way of delaying the caching of results, the second one introduces an own cache for result from coinductive cycles. The usage of this second cache requires to track where a result came from and if it depends directly or indirectly on a coinductive cycle. This tracking was implemented by adding a flag to results and change the handling based on this flag. To this end, changes in most components of the recursive solver were necessary. Both implementation approaches have in common that they come with a description of their functionality in the Chalk book.

After the pull request with the first approach was marked as ready to review, Jack Huey, one of the working group leaders, had a look at it on the same day. As they didn't have enough expertise with the recursive solver, they notified two other working group members who would be able to give a more insightful review. Unfortunately, they did not assign the other members as reviewers. The pull request was then pointed out to Niko Matsakis, one of the reviewers and the other working group leader, eleven days after the initial mention [37]. In the meeting the week after that the still outstanding review was mentioned again [38]. In the days after this meeting, I discovered a problem with the second implementation approach and started working on improving it. On this occasion, I also reworked the tracking of results that could lead to false positives. This effort then lead to a new approach; a dedicated coinduction handler component that manages a temporary cache and is able to handle even nested coinductive cycles correctly. The necessary changes for this coinduction handler were less than for the second approach. Thus, I closed the old pull request, which still hadn't been reviewed, in favor of the third approach and issued a new pull request for it [23].

About one week after the new pull request was opened, it was reviewed by the originally assigned reviewer. They commented on two smaller shortcomings and and announced that they would do another more detailed review later. In parallel to this review process, we discussed whether the first approach would actually have been reasonable as well [4]. At the point of this writing, both the review and the discussion are still in process.

4 Evaluation

The project work described in Section 3 resulted in several contributions to the Chalk project. In the context of the practical course, these contributions, all aspects of the project work and the motivations behind them shall be critically assessed. It is of special interest whether my work followed the best practices of Open Source software development and if it fulfilled the course's goals.

4.1 Autonomous Work

My contributions to the Chalk project were created in a mostly autonomous fashion. Both the initial research for the preliminary report and the subsequent investigation on the Chalk source code base were carried out without any help by other contributors. The motivation behind this approach was to gather all necessary information without bothering other contributors with questions, for which the solution has already been documented. Although asking for the required information might save time for new contributors, it seemed to me that it would require more effort by a seasoned team member than they could spare at that moment. This conception was based on the fact that there were no meetings and little to no significant contributions to Chalk between the mid of December, 2020 and the end of January, 2021 [cf. 5]. Even after this period, there was only little active development on Chalk and working group members did only find time to answer requests after up to five days [cf. 3]. In addition, the majority of the working group is based in the United States and communication with these members needs to be either asynchronous or in the limited time span in which both involved parties would be active. Due to these two limitations in effective communication with working group members, I decided to accumulate questions that arose while I worked on my contributions. These accumulated questions could then be posted when it was guaranteed that working group members would notice them in a timely manner.

In retrospect, my autonomous approach has proved to be reasonable yet not ideal. In particular, I was able to learn most necessary details about Chalk on my own and could work based on this information when the working group members were not able to share enough of their time with me. On the downside, autonomous working is not only slower than guided efforts, but also needs regular synchronization to not lose direction. As such, too much autonomous work is discouraged by the best practices of FLOSS. In the context of Chalk, the large amount of autonomous work resulted in another disadvantage. Small, regular interactions allow more developers to participate as they don't need much time to follow the conversations. If a conversation consists of longer texts and more topics at once, they become harder to follow and harder to take part in. Therefore, most interactions throughout the practical phase were only between me and the second leader of the traits working group, Jack Huey [3].

4.2 Reserved Commitment regarding Contributions

As mentioned before, I seldomly participated in the weekly working group meetings. The primary motivation for this reservation was my understanding that the meetings would be for interaction between the working group members only, as well as some shyness on my end. Although I grew gradually accustomed to interacting with members of the traits working group, a certain reluctance remained regarding what I could contribute to these meetings. In a similar fashion, I did not feel confident enough about how and when to inquire about reviews of both code changes and debugging results. This resulted in me waiting for the next working group meeting for updates instead of asking the parties concerned directly. The general feeling of insecurity regarding my knowledge of Chalk and the Rust trait system also resulted in me focusing on improvements of the Chalk book and investigation of bug reports. Both topics need only little knowledge about the way Chalk works on more then one abstraction level (cf. Section 5.3). Thus, I delayed my first feature contribution to the end of the project phase when I had gained enough understanding and felt able to provide a sufficient implementation.

In summary, the missing confidence and knowledge lead to a more passive way of interacting with the traits working group members and a focus on small contributions. Thus, I missed the opportunity to experience even more aspects of contributing to an Open Source project than I already did with my contributions.

4.3 Search for Knowledge

One central aspect of my project work on Chalk was, as described before, my effort in understanding the structure and inner workings of the library in greater detail. This focus on research was motivated by the desire to first achieve a certain level of knowledge before working on something. Due to this, I spent more time learning about the theory behind and the source code of Chalk than actually contributing to the project. In addition, most pull requests issued by me were either aimed at improving the documentation in the Chalk book ([18, 19],inter alia) or at least contained additions to it (primarily [23]).

In conclusion, the focus on gaining a better understanding resulted in less contributions and was therefore counterproductive to the goals of the practical course on one hand. On the other hand, I was able to gain enough knowledge about Chalk and especially about the recursive solver to now be able to implement more meaningful contributions. The pull request #690 can be seen as evidence for this newly acquired knowledge. In addition, I kept track of most discussions on the Rust-Lang Zulip server, which allowed me to learn not only about the development process of Chalk but also of the Rustc compiler and other Rust-Lang projects. The insight thus gained may prove helpful in future contributions to Chalk and other Rust-Lang projects. As such, the goal of the practical course - to prepare its participants for more participation in the FLOSS community - has been achieved to its fullest. On top of this, I also learned about several features of the Rust programming language that are currently considered unstable and actively developed, but which are already supported by Chalk. Due to these insights, my understanding of the Rust programming language and its development goals increased by a significant amount, which in turn also allows me to make more meaningful contributions to the Rust-Lang community in the future.

4.4 Reflection on the Preliminary Report

The preliminary report [28] is based on the results of my research about the organization and management aspects of the Chalk project. Thus, the preliminary report was the basis for all following project work and should be assessed as such. In retrospect, there are several assumptions and assessments in the preliminary report that have proved to be inaccurate.

First and foremost, my evaluation of the development status of Chalk was askew. Back then, I evaluated the integration of Chalk into Rustc as already usable in the development version. This is not entirely correct, as Chalk can be utilized via a compiler feature flag but results in many errors even for simple programs. In addition, the integration is also incomplete in that sense that not all of Chalks functionalities can actually be utilized. Other than the integration into Rustc, I also assessed the role of Chalk as part of the Rust-Analyzer. In particular, it was described as merely “usable” [28, page 2], whereas in reality Rust-Analyzer utilizes Chalk as its primary trait system model.

Another misconception in the preliminary report was to anticipate “broader interest and participation” [28, in section 3] by a growing number of dedicated contributors of the Chalk project. Over the course of the project phase, it became apparent that most contributors to Chalk, especially the traits working group members, work on Chalk as part of the development of the whole Rust programming language. The focus of the working group on traits related topics other than Chalk is evidence for this observation. In summary, my previous statement is too general and should be changed to “broader interest and participation” by the Rust-Lang community as a whole and Rustc contributors in particular.

One other aspect of the preliminary report, that proved to be inaccurate, was the anticipation of a highly beginner friendly variety of open GitHub issues. Whereas new issues are often solved in a timely manner, most of the open issues are older than one year and not actively handled or managed. This situation influenced my reluctance to work on bigger contributions described in Section 4.2, as well.

In spite of the misconceptions made in the preliminary report, the initial research for this report resulted in mostly reliable information about the Chalk project and its governance. It can, therefore, be seen as a successful preparation for the subsequent project phase.

5 Lessons learned

Independently of how the project work is assessed, it is possible to abstract aspects of the Chalk project and derive conclusions about similar FLOSS projects in general. These conclusions are then also evaluated with regard to how certain aspects might affect future participants of the practical course "Contributing to an Open-Source Project".

5.1 Subproject Relations in Open Source Communities

The Chalk project is developed as a project of the Rust-Lang community and, in particular, as a subproject of the Rustc compiler. Although it is part of the "library-ification" efforts that try to decouple parts of Rustc into distinct libraries, the development of Chalk is primarily aimed at the integration into Rustc [31, inter alia]. In addition, a majority of contributions to Chalk can be accredited to developers who already had experience with the Rustc compiler before starting to work on Chalk. As an example, of the 506 pull requests up to this date (2021-03-12), 330 were opened by working group members [14, 15]. As a result of these pull requests 1756/2879 commits with 557237/607947 additions and 509743/548179 deletions were contributed by working group members (all numbers were acquired through GitHub's REST API on 2021-03-12). Around half (6 of 14, [15]) of the working group members are in parallel members of the Rust-Lang compiler team with a rising tendency [34].

The relation between the Chalk and the Rustc projects can be described as a "hidden" subproject relation. Both the alignment to Rustc and the strong influence of compiler team members is not apparent at a first glance and could therefore be described as "hidden". This kind of relation increases the workload for new contributors as they have to get accustomed to the parent project as well. Thus, this structure makes the subproject less suited for the practical course "Contributing to an Open-Source Project" where the participants should concentrate on only one project. Apart from that, it allows the participants to learn about the intricate structure and relations of bigger Open Source communities, which in turn simplifies contributing to these projects after the end of the practical course.

5.2 Small, Global Teams and Slow Progress

The Chalk project is a small project; not only with regard to the lines of source code it contains, but also with regard to the number of contributors working on it. The traits working group has 14 members of which two are the team leaders [15]. As it is often the case with Open Source projects, the working group members and contributors are also located in several different time zones. Despite this, pull requests with trivial changes were often reviewed and merged not later than the next day (e.g. [18, 19, 20]). As could expected, pull requests with more extensive changes can still take more than a week to be reviewed if the responsible reviewers do not find the time for it (e.g. [23]). Another aspect of small teams and small projects is the amount of influence a single contributor can have. In the context of Chalk, this correlates with the new feature and documentation added in pull request #690.

In general, a small number of geographically distributed developers and maintainers makes it difficult to find a contact person which is reachable when needed. Furthermore, the general development progress of a smaller team is likely to be slower than with bigger teams. This property may become especially relevant if only few of these developers are able to work fully on the project. Both of these factors make small projects less optimal for this practical course as participants might be required to do more work on their own and are likely to have to wait longer for responses and reviews. Yet, projects that meet the described properties can become a good opportunity to become a regular contributor or even maintainer oneself and, thus, have a greater influence on future development decisions.

5.3 Mentoring and Introduction

Chalk is a rather complicated software library. Although it has not as many lines of code as, for example, the Rustc compiler, it requires knowledge about the whole trait system to understand it in its fullest. Therefore, Chalk differs from many other Open Source projects in that it contains another layer of complexity. Most software projects can be described by three distinguished levels of information that a developer needs to know; these are the underlying theory of what the software should do on a high level (i.e. its business logic), an overview of how the software is divided in modules and which module provides which functionalities and, lastly, the actual source code itself. Even though this partitioning might not be valid for all possible software projects, it is particularly apparent in the Chalk project. For the Chalk project, the high level theory describes how Rust code should be lowered to clauses and how the SLG solver and the recursive solver should prove or disprove goals with these clauses. The middle level describes how the lowering, the intermediate representation and the solvers work together, whereas the source code level describes the inner workings of these modules in detail. The difference for Chalk in comparison to other projects is the fourth level consisting of the theory of the Rust trait and type systems. This theory is essential for both the lowering process and the correct handling of corner cases in all parts of Chalk.

As a result, starting work on Chalk can be more difficult than with other software projects. This increased difficulty results in an increased necessity for beginers instructions. These instructions could be provided in the form of a detailed tutorial with extensive documentation or by a strong mentorship program. The first solution would require the project to be more or less stable as frequent and more extensive changes to the source code would also require frequent and extensive updates on the tutorial. On the other hand, the second solution requires a dedicated maintainer team that has enough time to spend on mentoring new contributors [cf. 13]. Thus, both solution approaches would not work well for small and experimental niche projects like Chalk and new contributors need to be willing to commit more time for getting to know the whole project to be able to contribute in the long run. Even though this fact increases the workload and difficulty for future participants of the practical course when choosing such a project, it does not make it impossible to develop a meaningful contribution.

5.4 Challenges in FLOSS Project Work

The most important realization that I made during the project phase was that contributors to Open Source projects have to challenge themselves and sometimes also other contributors. Challenges are mostly related to the personal capacity of each contributor and can be found in all aspects of Open Source development.

The most central challenge is the communication with other contributors and with maintainers. Even in smaller projects, the interaction with unfamiliar developers can require courage and might lead to a reserved demeanor (cf. Section 4.2). This effect can be amplified if the contrast in conversational tone between native and non-native speakers becomes too apparent in technical discussions (cf. [3] or the five minute difference between a message and my answer in [39]). Additionally, successful work on Open Source projects often requires collaboration of contributors. For this collaboration to succeed, an exchange of knowledge and a realistic advertisement of a contributors own strengths and abilities is inevitable, yet not trivial to do.

Another aspect of Open Source development that can become a challenge is coordination of work. Most Open Source projects have only a limited amount of open issues, that contributors can work on. Although cooperation platforms such as GitHub and Gitlab allow users to claim issues for themselves to work on, parallel work on issues is likely to occur even in small projects[3]. Such situations might either lead to tensions or cooperation depending on how both contributors behave.

6 Conclusion

Although my work on the Chalk project has not been as anticipated, it resulted in a number of lessons learned and newly acquired skills as well as several contributions. The project phase has especially proved to be helpful in becoming able to make meaningful contributions to Rust-Lang projects. In addition, I am motivated to work on FLOSS projects in the future in general and will continue to contribute to Rust-Lang projects and Chalk in particular. Hence, the participation in the practical course "Contributing to an Open Source Project" with regard to Chalk was successful in its own right.

[3] An example for this is issue #667 [12]. I started looking into it the same day a maintainer opened a pull request with a solution draft for this issue.

References

[1] *About the recursive solver Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/49904Abouttherecursivesolver.html (visited on 2021-01-14).

[2] *AliasEq vs ProjectionEq questions Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/89128AliasEqvsProjectionEqquestions.html (visited on 2020-12-28).

[3] *Ambiguous WF Results Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/14716AmbiguousWFResults.html (visited on 2021-01-28).

[4] *coinduction Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/71232coinduction.html (visited on 2021-03-11).

[5] *Development in the near future - Zulip Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/19517Developmentinthenearfuture.html (visited on 2020-12-13).

[6] *GitHub repository of Chalk.* URL: https://github.com/rust-lang/chalk (visited on 2021-03-04).

[7] *GitHub repository of Polonius.* URL: https://github.com/rust-lang/polonius (visited on 2020-11-26).

[8] *Hi from FireFighterDuck.* URL: https://zulip-archive.rust-lang.org/122652newmembers/04476HifromFireFighterDuck.html (visited on 2021-03-05).

[9] *Issue 234 on Chalk.* URL: https://github.com/rust-lang/chalk/issues/234 (visited on 2021-03-04).

[10] *Issue 399 on Chalk.* URL: https://github.com/rust-lang/chalk/issues/399 (visited on 2021-02-11).

[11] *Issue 587 on Chalk.* URL: https://github.com/rust-lang/chalk/issues/587 (visited on 2021-01-14).

[12] *Issue 667 on Chalk.* URL: https://github.com/rust-lang/chalk/issues/667 (visited on 2021-01-14).

[13] Stephan Kulla. *Guest Talk Motivation in Open Source Projects.* "invited talk at university". 2020-12-08.

[14] *List of Pull Request on Chalk.* URL: https://github.com/rust-lang/chalk/pulls (visited on 2021-03-09).

[15] *List of Traits Working Group Members.* URL: https://github.com/rust-lang/team/blob/master/teams/wg-traits.toml (visited on 2021-03-08).

[16] Nicholas D. Matsakis. *Library-ification and analyzing Rust.* 2020-04-09. URL: https://smallcultfollowing.com/babysteps/blog/2020/04/09/libraryification/ (visited on 2021-03-04).

[17] *Practical Course - Contributing to an Open-Source Project Website.* URL: https://www21.in.tum.de/teaching/osp/WS20/index.html (visited on 2021-03-04).

[18] *Pull Request 663 on Chalk*. URL: https://github.com/rust-lang/chalk/pull/663 (visited on 2020-12-13).

[19] *Pull Request 668 on Chalk*. URL: https://github.com/rust-lang/chalk/pull/668 (visited on 2020-12-28).

[20] *Pull Request 674 on Chalk*. URL: https://github.com/rust-lang/chalk/pull/674 (visited on 2021-01-14).

[21] *Pull Request 675 on Chalk*. URL: https://github.com/rust-lang/chalk/pull/675 (visited on 2021-01-14).

[22] *Pull Request 683 on Chalk*. URL: https://github.com/rust-lang/chalk/pull/683 (visited on 2021-02-11).

[23] *Pull Request 690 on Chalk*. URL: https://github.com/rust-lang/chalk/pull/690 (visited on 2021-03-05).

[24] *Rust Foundation Website*. URL: https://foundation.rust-lang.org/ (visited on 2021-03-04).

[25] *Rust Language Zulip Instance*. URL: https://rust-lang.zulipchat.com/ (visited on 2021-03-05).

[26] *rust-analyzer Website*. URL: https://rust-analyzer.github.io (visited on 2020-11-24).

[27] *rust-lang.org*. URL: https://rust-lang.org (visited on 2020-11-24).

[28] Florian Sextl. *Practical Course – Contributing to an Open-Source Project. Preliminary Report on Rust-Lang/Chalk*. Tech. rep. Technical University of Munich, 2020. URL: https://www21.in.tum.de/teaching/osp/WS20/assets/pr-sextl-rust-chalk.pdf.

[29] *The Chalk Book*. URL: https://rust-lang.github.io/chalk/book (visited on 2020-11-24).

[30] Jack Huey on behalf of The Traits WG. *Traits working group 2020 sprint 1 summary*. URL: https://blog.rust-lang.org/inside-rust/2020/03/28/traits-sprint-1.html (visited on 2021-03-04).

[31] *Weekly Workgroup Meeting 2020-11-17 Thread*. 2020-11-17. URL: https://zulip-archive.rust-lang.org/144729wgtraits/36754meeting20201117.html (visited on 2020-11-24).

[32] *Weekly Workgroup Meeting 2020-12-08 Thread*. URL: https://zulip-archive.rust-lang.org/144729wgtraits/33363meeting20201208.html (visited on 2020-12-13).

[33] *Weekly Workgroup Meeting 2020-12-15 Thread*. URL: https://zulip-archive.rust-lang.org/144729wgtraits/67825meeting20201215.html (visited on 2020-12-28).

[34] *Weekly Workgroup Meeting 2021-01-05 Thread*. URL: https://zulip-archive.rust-lang.org/144729wgtraits/05679meeting20210105.html (visited on 2021-03-09).

[35] *Weekly Workgroup Meeting 2021-01-19 Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/98900meeting20210119.html (visited on 2021-01-28).

[36] *Weekly Workgroup Meeting 2021-01-26 Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/33362meeting20210126.html (visited on 2021-01-28).

[37] *Weekly Workgroup Meeting 2021-02-16 Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/64434meeting20210216.html (visited on 2021-03-11).

[38] *Weekly Workgroup Meeting 2021-02-23 Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/98896meeting20210223.html (visited on 2021-02-25).

[39] *Weekly Workgroup Meeting 2021-03-09 Thread.* URL: https://zulip-archive.rust-lang.org/144729wgtraits/92117meeting20210309.html (visited on 2021-03-10).

Title	Work done	Start date	End date	Results	References
Preliminary Report	Research on project structure of Chalk	2020-11-17	2020-12-01	Preliminary Report	[28]
Pull Request #663	Update a JS library used with the Chalk book	2020-12-10	2020-12-11	Merged	[18]
Pull Request #668	Update the Chalk book section on `AliasEq`	2020-12-23	2020-12-24	Merged	[19]
Issue #667	Debugging	2021-01-04	2021-01-04	None	[12]
Issue #587	Debugging	2021-01-05	2021-02-01	Bug location narrowed down	[1, 3]
Pull Request #674	Remove duplicate code	2021-01-08	2021-01-08	Merged	[20]
Pull Request #675	Correct a formula in the Chalk book	2021-01-12	2021-01-13	Merged	[21]
Pull Request #683	Implement delayed caching for coinduction	2021-02-04	2021-03-02	Closed	[22]
Pull Request #690	Implement dedicated coinduction handler for the recursive solver	2021-03-02	-	Open	[23]

Table 1: Chronologically ordered Contributions in the Project Phase