Automated Theorem Proving in a Chat Environment

Rustam Zhumagambetov Mark Sterling

Department of Computer Science
School of Science and Technology
Nazarbayev University
Astana, Kazakhstan 010000

rustam.zhumagambetov@nu.edu.kz mark.sterling@nu.edu.kz

We present a chat bot interface for the Coq proof assistant system. The bot provides a new modality of interaction with Coq that functions across multiple devices and platforms. Our system is particularly suitable for mobile platforms, Android and iOS. The basic architecture of the bot software is reviewed as are the outcomes of several rounds of beta-testing. Potential implications of the system are discussed, such as the possibility of a seamless collaborative proof environment.

1 Introduction

Interactive theorem provers require a person to guide the assistant towards a correct proof. The user provides high-level guidance while the proof assistant produces the low-level implementation, or reports back to the user about various mistakes.

Traditional *integrated development environments* (IDEs) for programming are complex applications that effectively require a desktop operating system. While these IDEs provide a powerful workspace that allows the programmer to concentrate on development, they may also consume significant computational resources. Also, an IDE may not support development across systems and devices.

We present a new interface for theorem proving: a chat bot. As we demonstrate, a chat bot can offer the kinds of interactions that are found in existing proof assistants. A prototype chat bot built on the Coq language and the Telegram platform is presented below.

The primary benefit of the chat bot as a proof interface is that chat applications are typically platform agnostic. A theorem prover written against a popular chat application API will function uniformly across all supported devices and operating systems. There is no need to port the interface of a tool using the native UI kit of a platform.

Telegram allows for the definition of custom interfaces for specific chat bots. These interfaces allow the bot to accommodate users on different types of devices (for example, touch devices or devices with a pointer). This provides a promising direction for bringing interactive theorem proving in Coq to mobile platforms such as Android and iOS. For example, there is, currently, no simple procedure for installing Coq on Android. A user wishing to do so may require in-depth knowledge of OCaml, OPAM, and Android internals. Then the console editors, such as vim or emacs with plugins are installed. Such complications are avoided with a chat bot.

Our bot can be run easily on both personal machines and servers. The setup of the chat bot resembles the setup of the Coq itself. The application is packaged as a single `jar` file. The only other dependencies are the required Telegram API keys. These keys are free and easy to obtain (a special bot, the *BotFather* is provided for this purpose). Since the bot relies on the version of the Coq installed to the host system, it is trivial to customize the Coq instance with any necessary libraries/plugins. The project is open source. The code is available online under the MIT license and we welcome contributions.

Submitted to:
UITP 2018

2 Telegram Chat bot overview

Telegram is a cloud-based messenger application that focuses on security. One of its features is the ability to create chat bots using the provided API as well as SDKs for programming languages such as PHP, Java, Javascript (requires NodeJs), Python, C#, Ruby, Haskell, Scala, Swift and OCaml [1]. Another fact about Telegram is that its client API is open-source, so that anyone can build their own Telegram client, or customize an existing one. Install-ready solutions are available on multiple platforms, both mobile and desktop.

A Telegram chat bot is a chat user that is managed by an application. The powerful chat API allows for the development of complex interactive bots. A variety of bots already exist including a Gmail bot (gmail client) and a Weather bot (fetches weather conditions for selected cities) [2]. Aside from utilitarian bots, there are also bots for entertainment, such as an interactive fiction bot.

Telegram bot [3] messaging is based on HTTP requests. A Bot application operates in either of two modes: long polling (request updates from server) and webhook (receive updates). Bots can distinguish among message types available on Telegram. Depending on the content of the message and its type, the bot chooses an action. For example, if the user sends a picture, bot can comment on it, or download and save the photo. The Telegram API accepts messages in UTF-8 encoding. Any UTF-8 character can be displayed in Telegram, including non-latin alphabets, emojis, and special characters.

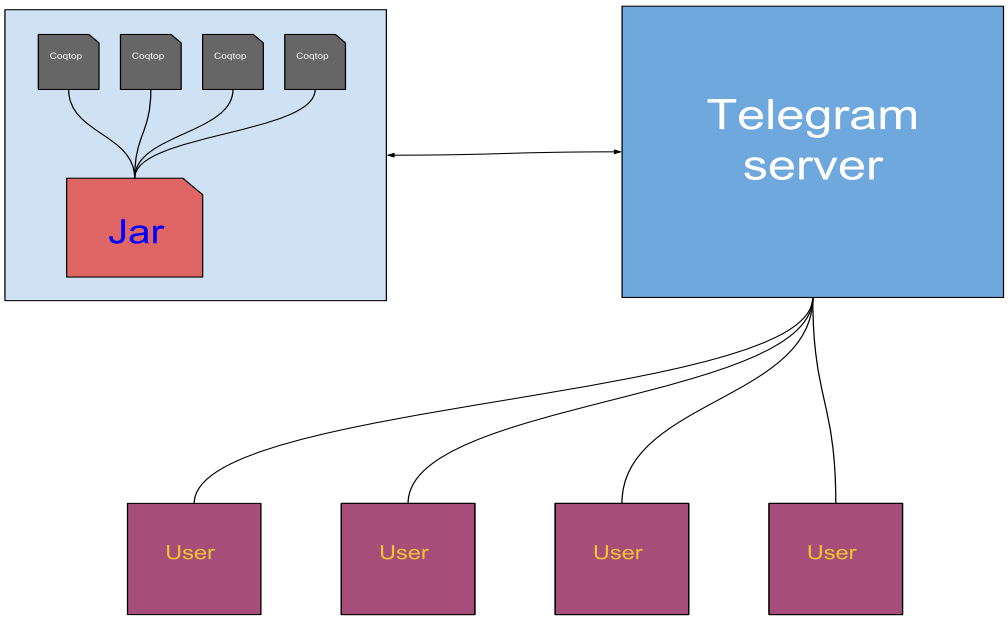

Figure 1: High level system overview

3 System overview

3.1 High level overview

As can be seen in Fig. 1 the system consists of coqtop instances, the chat bot jar file, and the Telegram server (accessed via the network). Figure 1 also shows the users. Chat users send commands to Telegram, which forwards them to the bot. Individual users are distinguished by chat ID and are each given separate coqtop processes.

Coqtop is the standard REPL of Coq. It is distributed together with the Coq compiler. The majority of plugins, IDEs, and utilities for Coq are built around coqtop.

3.2 Communication with Coq

To communicate with Coq, a Java module was implemented that utilizes the same XML protocol that CoqIDE uses for communication. It allows other processes to send vernacular commands to the coqtop instance for further interpretation. Coqtop then communicates a response back, either accepting the command or rejecting it. This Java module is a separate component of our chat bot and can easily be reused or adapted for other Java applications.

3.3 User interface

Bots may implement a large number of commands of which only a few are used regularly. For frequently used commands, Telegram provides a nice alternative to manual typing: custom keyboards. Custom keyboards can either be attached to a message or can substitute for the virtual keyboard. This feature allows the user to interact with the bot with greater comfort as and less tedium.

Figure 2 shows two types of keyboards that are used in the coq bot. The first is used to choose between loading external file, querying the current state of the coq and declaring a new entity. The second shows a set of tactics can be applied during a proof. In both cases, the custom keyboard is attached to the message.

Since Coq is an interactive proof assistant, writing proofs in Coq is similar to having a dialog. Since chat bots are based on simulating conversation with users, they fit well with the concept of interactive proof. Figure 3 demonstrates a simple proof with our chat bot. Note that when the bot needs input from the user it automatically adds a request in its reply.

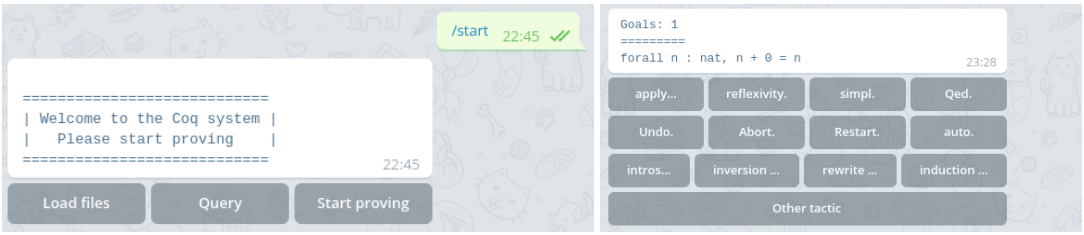

Figure 2: Screenshots of messages with custom keyboard

In addition, the bot currently supports import and compilation of files in **.v** and **.vo** formats. Users also have the option to upload files with drag-and-drop.

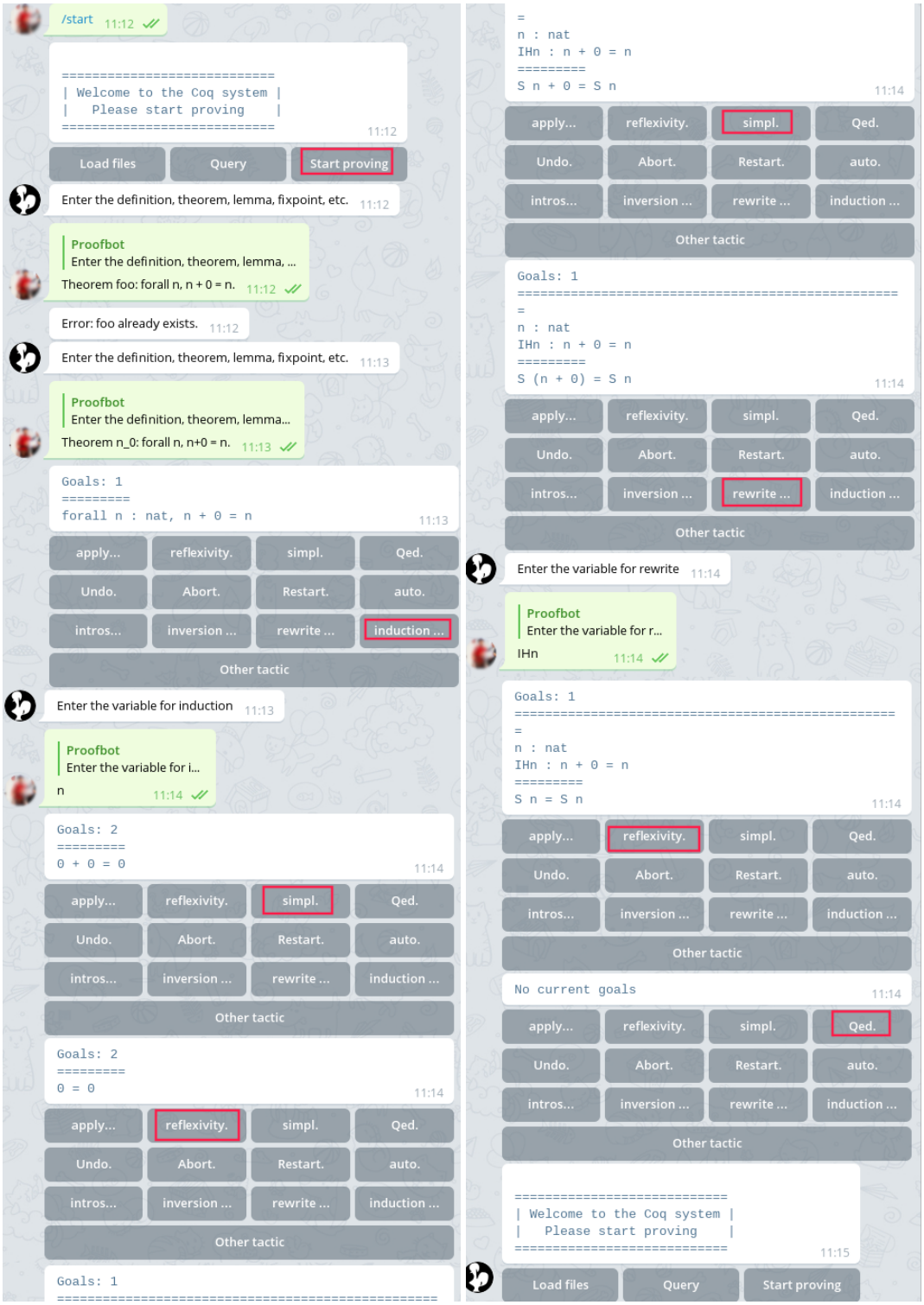

Figure 3: Screenshots of simple proof

3.4 Distribution

Currently, the bot is distributed as a single jar file and as a docker image. Because of the XML protocol changes in the latest versions of Coq (8.7 and above) the jar file requires the specific installation of Coq 8.6. To facilitate and standardize installation, a docker image is distributed in the repository. The Docker image, based on the official Alpine linux image, is compact a container ≈200MiB that packages all required software and only requires entry of Telegram API keys.

The distributed image contains default installation of the Coq and OPAM, so it is relatively easy to customize with any of the preferred libraries or plugins that are distributed through OPAM.

A public instance is available on our server. It can be found by searching **@CoqProofbot** [1] in Telegram.

4 Community feedback

Initially the bot was intended as a private, single user application. However, during development we also became interested in feedback from the community. Therefore, we extended the system to a multi-user environment.

We have organized two public beta tests through the Coq club mailing list. In both rounds we sent a short description of the bot with screenshots demonstrating a simple proof, and attached a brief survey with a link to the bot. The survey consisted of three questions:

1. What did you like about Coq telegram bot?
2. What did not you like about Coq telegram bot?
3. Any additional comments?

In the first round, we only tracked chat IDs that uniquely identify conversations between users and the bot. However, we did not map chat IDs to usernames (preserving anonymity). The results were that 12 people used bot, 2 provided feedback through the survey. We carefully considered the survey responses and redesigned error and state handling. After reviewing the code from the initial prototype, we also realized that it was poorly architected. So the code base was refactored to introduce a more modular structure.

In the second round, we attached a list of updates as well the same survey. Among the new features added were state persistence in a database, better error messages and general performance improvements. The results were that 30 people used bot with 1 person answering the survey.

Generally, survey feedback was positive. Most of the concerns expressed by respondents were flaws of the system, such as bad error recovery. They are related to the implementation details, rather than limitations of the system as a whole.

5 Related work

To accommodate interactive proving numerous systems were built. Here several systems that connect web and proof assistants, as well as plugins to the text editor are mentioned.

One of the recent works is jsCoq [6]. jsCoq is Coq inside browser. Using js_of_ocaml compiler, Coq's code was compiled into javascript file. Such construction allows seamlessly integrate Coq into browser.

[1] or visiting `http://t.me/CoqProofbot`

Although browser is available on many platforms, jsCoq brings traditional, desktop-like interfaces to the mobile. It fails to provide organically integrate proving interface with environment with touch interface.

Another work is CoqPea [7]. It is web-based front-end to the Coq. It is similar to the chat bot in a sense that it also similarly communicates with Coq.

List of plugins to the existing text editors that provide IDE like experience includes ProofGeneral [4], based on Emacs, and VSCoq [5], based on Visual Studio Code.

Finally, CoqIDE [8] is the standard graphical interface that is shipped along with Coq.

6 Future work

Currently bot is in beta stage. The list of features we desired to implement is fixed for now. However, there is a pile of features that would augment current user experience. They can be classified as following:

- User interface
- Proof persistence
- Collaboration

User interface Telegram allow editing of sent messages. Bot instead of creating new message bubble, can edit previous message to update the state of the proof. As we can attach custom keyboards to the messages, such message updates are natural to Telegram bots. Bot could have become one message – one proof system.

Proof persistence Currently, the bot uses MongoDB to save the state of Coq proofs. In the future, we hope to allow users to download proofs that were made with the bot.

Collaboration The Telegram API provides a number of social features. Bots can be added to multi-user conversations. This allows for the possibility of collaborative theorem proving.

7 Conclusion

We have presented a Coq chat bot that serves as a front-end to the Coq proof assistant, augmenting the user experience with features, such as unified interface across devices and support of touch interactions. We believe that this can be a valuable contribution to the Coq community and that has the potential to widen the audience for automated theorem provers. The source code of the tool is stored at `https://gitlab.com/rustamzh/telegramcoqbot`.

References

[1] *Bot Code Examples*. `https://core.telegram.org/bots/samples`. Accessed: 2018-04-10.

[2] *Bots: An introduction for developers*. `https://core.telegram.org/bot`. Accessed: 2018-04-10.

[3] *Telegram Bot API*. `https://core.telegram.org/bots/api`. Accessed: 2018-04-10.

[4] David Aspinall (2000): *Proof General: A Generic Tool for Proof Development*. In Susanne Graf & Michael Schwartzbach, editors: *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 38–43.

[5] Christian J. Bell: *A plugin for the Coq Proof Assistant 8.5 and 8.6 in Visual Studio Code*. https://github.com/siegebell/vscoq/.

[6] Emilio Jesús Gallego Arias, Benoît Pin & Pierre Jouvelot (2017): *jsCoq: Towards Hybrid Theorem Proving Interfaces*. In Serge Autexier & Pedro Quaresma, editors: *Proceedings of the 12th Workshop on User Interfaces for Theorem Provers, Coimbra, Portugal, 2nd July 2016*, *Electronic Proceedings in Theoretical Computer Science* 239, Open Publishing Association, pp. 15–27, doi:10.4204/EPTCS.239.2.

[7] Valentin Robert: *PeaCoq*. https://github.com/Ptival/PeaCoq.

[8] The Coq Development Team (2018): *The Coq Proof Assistant, version 8.7.2*, doi:10.5281/zenodo.1174360. Available at https://doi.org/10.5281/zenodo.1174360.