Exposé zur Bachelorarbeit
Ein Schach-Interpreter in Rosette

Robin Wroblewski

Heinrich-Heine Universität Düsseldorf
Institut für Informatik
Softwaretechnik und Programmiersprachen

November 24, 2023

1 Motivation

Rosette ist eine *solver-aided programming language* auf der Basis von Racket. Im Wesentlichen besteht es aus einer Teilmenge des Racket Kerns ergänzt um einen *symbolic compiler* und Konstrukte, mit denen Queries abgesetzt werden. Dieser Compiler kann ein Programm in logische Constraints übersetzen, die der integrierte SMT-Solver versteht und lösen kann. Das Ergebnis wird in das Programm zurückgeführt, sodass es beispielsweise für weitere Modifikationen oder zur Formulierung eines Ausdrucks verwendet werden kann.
Die Sprache wurde entworfen um sich als Host-Sprache für solver-aided DSLs zu eignen. Dies kennzeichnet sich durch die Unterstützung von Metaprogrammierung, eine von Racket vererbte Eigenschaft, die außerdem einfache Einbettung neuer Sprachen ermöglicht. Entwickler, die ihre Sprache in Rosette implementieren, sind davon befreit selbst eine Übersetzung in Constraints vorzunehmen und erhalten automatisch Zugang zu Werkzeugen für Verifikation, Fehlerlokalisierung, Programmsynthese und Programmreparatur.

Rosettes solver-aided Konstrukte basieren auf vier grundlegenden Konzepten: Assumptions, Assertions, Symbolic Values und Queries. Mit Assumptions und Assertions können Regeln ausgedrückt werden, denen jedes Programm folgen muss, um ohne Fehler zu laufen. Symbolic Values werden dann zum Zusammenstellen von Queries, die Fragen über das Verhalten des Programms an den Solver stellen, verwendet. Grundsätzlich gibt es vier Queries, die den Großteil der Anwendungsfälle für Solver in der Programmierung abdecken und diese werden alle von Rosette unterstützt:
1) verifizieren, dass ein Programm eine Spezifikation für einen gegebenen Eingaberaum erfüllt;
2) finden eines Satzes von Eingabewerten, die zur Erfüllung einer Spezifikation bei Programmausführung führen;
3) lokalisieren eines minimalen unerfüllbaren Kern, der das Programm daran hindert ein gewünschtes Verhalten zu demonstrieren;
4) synthetisieren eines Programms mit gewünschtem Verhalten nach Vorlage eines syntaktischen Sketches.

Diese Arbeit ist motiviert durch das Ziel die angesprochenen Werkzeuge im Zusammenhang mit B-Maschinen [Abr96] zu nutzen. Dies könnte beispielsweise in der Form einer direkten Übersetzung von B-Maschinen in Rosette Code oder der Entwicklung eines vollständigen B-Interpreters in Rosette realisiert werden. Ersteres würde Zugriff auf Werkzeuge für Verifizierung und Buglokalisierung liefern, während letzteres außerdem Programmsynthese ermöglicht.
Der mit diesem Projekt verbundene Aufwand ist jedoch vergleichsweise hoch, weshalb es für sinnvoll erachtet wird mit dieser Arbeit einen Zwischenschritt einzuführen, der gleichzeitig als Machbarkeitsprüfung dienen soll.

2 Aufgabenstellung

Die Aufgabe dieser Arbeit ist es einen Schach-Interpreter als Fallstudie zu entwickeln. Das beinhaltet die Regeln des Spiels, also wie sich Figuren auf dem Spielbrett bewegen dürfen oder wie das Spiel beendet wird, in Code abzubilden. Die Methoden dieses Interpreters sollen dann verwendet werden um Queries für verschiedene Aufgaben zu stellen, beispielsweise um zu überprüfen, ob ein Zustand in einen anderen übertragen werden kann oder um "Matt in drei Zügen"-Rätsel zu lösen.
Der Fokus liegt dabei jedoch zu allererst auf der Erkundung der Programmiersprache Rosette und ihrer Solver unterstützte Möglichkeiten. Der Schach-Interpreter liefert ein festgelegtes Ziel auf das hingearbeitet wird, aber der hauptsächliche Gewinn wird das im Prozess erlangte Wissen sein. Das bedeutet ein wichtiger Punkt wird es sein die bei der Entwicklung getroffenen Entscheidungen zu dokumentieren und auftretende Hürden und gegebenenfalls wie diese überwindet werden konnten zu beschreiben.
Wie umfangreich der Interpreter sein wird und welche Aufgaben mit dem Solver automatisiert werden können wird sich erst im Laufe der Entwicklungsphase herausstellen. Der Startpunkt wird es sein einen Prototypen in Racket zu entwickeln und dann die korrekte Umsetzung der Regeln mit konkreten Eingabewerten zu testen. Danach wird die Solver unterstützte Funktionalität Schritt für Schritt eingeführt, indem das Racket Programm zunächst in ein Rosette Programm übersetzt wird und dann die gewünschten Queries hinzugefügt werden. Für das Konvertieren des Programms wird der Code so angepasst, dass er nur noch Methoden des Rosette Kerns enthält und anschließend werden konkrete durch Symbolic Values ersetzt.
Das Endergebnis der Arbeit soll Auskunft darüber geben, ob Rosette als Solver unterstützte Sprache für Probleme dieser Art geeignet ist. Falls das angestrebte Ziel nicht erreicht werden kann, wird genauer untersucht werden warum dies der Fall ist. Andernfalls liefert die Arbeit eine praktische Beschreibung der verwendeten Features und der allgemeinen Vorgehensweise bei der Entwicklung einer Solver unterstützten Software, die als Hilfestellung für die weitere Erkundung des Themas dienen kann.

3 Verwandte Arbeit und Forschungsstand

Entscheidend für dieses Projekt sind die Arbeiten und Publikationen von Emina Torlak, der Entwicklering von Rosette. In ihrem Paper stellt sie Rosette vor und zeigt anhand des Beispiels einer Schaltkreis-Sprache wie eine Solver unterstützte DSL entwickelt werden kann [TB13]. Der Ansatz, der in diesem Beispiel beschrieben ist, deckt sich in etwa mit meinem Plan, nämlich vorerst einen kleinen Prototypen in Racket zu entwickeln und diesen dann schrittweise in einen Solver unterstützten Interpreter zu überführen. Diese Vorgehensweise ist nur sinnvoll, weil Rosette so in Racket integriert ist, dass sich Rosette Programme ohne Symbolic Values wie Racket Programme Verhalten. Außerdem Teil des

Papers sind drei weitere Fallstudien von Solver unterstützten Systemen, die in Rosette entwickelt wurden. Diese sind:
WebSynth, ein Werkzeug für *web scraping* welches ZPaths für Webseiten synthetisiert.
Ein Partitionierer für GA144 Programme, also Programme auf einer Architektur, die aus 144 Prozessoren mit sehr geringem Speicher besteht, laufen.
Ein Superoptimierer für Bitvector Programme.
Die ersten beiden wurden von Studenten entwickelt, indem sie zuerst einen Racket Interpreter für ihr Problem implementiert und diesen dann mit Rosette in einen Synthetisierer verwandelt haben.

Des Weiteren werden auf der Rosette Webseite [Tor] einige Applikationen vorgestellt. Diese automatisieren Aufgaben aus einer Vielzahl verschiedener Bereiche, darunter neben Feldern der Informatik auch beispielsweise Medizin oder Logik.

Applikationen beschreiben, habe in der englischen Version aber meiner Meinung nach nicht die passendsten rausgesucht, da diese nicht die erwähnten Felder abdecken. Den Teil also neu schreiben. Außerdem die Arbeit von Phillip Höfges erwähnen und die Publikationen zu npm etc.

4 Zeitplan

Generell habe ich mir vorgenommen schon beim Programmieren die wichtigen Gedanken und Ergebnisse zu dokumentieren. Dabei werde ich noch nicht so sehr auf gute Formulierung, wenig Wiederholungen et cetera achten, sondern erstmal geht es darum Inhalt zu produzieren. Aus dem Ertrag kann ich mir dann Passagen rausziehen, diese umschreiben, kürzen, anders zusammensetzen und mich somit Stück für Stück der richtigen Arbeit annähern. Wenn ich regelmäßig, beispielsweise nach jeder Woche, den Output ordne und damit gut formulierte Texte zu den jeweiligen Kapiteln schreibe, so wächst die Arbeit gleichmäßig mit dem Programm mit.

Ziele:

Woche 24.07 - 30.07: die ersten commits im GitLab, den Racket Interpreter möchte ich in der Woche soweit fertig programmieren (Regeln für einige Figuren, Spielende, also Schach-, Matt-, Remis-Regeln) und die Texte dazu schreiben, hier sollte ich auch schon anfangen meine Vorgehensweise mit der von Phillip Höfges zu vergleichen und etwas näher auf seine Arbeit eingehen.

Woche 31.07 - 06.08: hier bin ich nicht da.

Woche 07.08 - 13.08: hier bin ich nicht da.

Woche 14.08 - 20.08: das Programm wird in Rosette übertragen und mit den ersten Queries experimentiert (Lässt sich ein Zustand in einen anderen übertragen?) und die Ergebnisse beschrieben, außerdem anfangen die Implementierung im Text zu erklären.

Woche 21.08 - 27.08: mit den gesammelten Erfahrungen genauer überprüfen wie sich mein/e Problem/Vorgehensweise von den in den anderen Publikationen beschriebenen Problemen/Vorgehensweisen unterscheidet, außerdem anfangen über Ergebnisse zu reflektieren (war die Vorgehensweise gut? Eignet sich Rosette für das gestellte Problem?), und weitere Queries ausprobieren (kann der Gegner Matt gesetzt werden?).

Woche 28.08 - 03.09: die bisher entwickelten Queries versuchen auf eine anderen Weise umzusetzen, um Vergleiche ziehen zu können, werden hier andere Ausgaben produziert? Zum Beispiel: wurde bisher Programmsynthese mit Hilfe einer Grammatik umgesetzt → wie funktioniert das ohne Grammatik oder wie werden die gleichen Probleme mit Angelic Execution gelöst?

Woche 04.09 - 10.09: Das Programm sollte hier soweit sein, dass die vorhandenen Features entweder ausreichend wären oder ich zu dem Schluss gekommen bin, dass das erstrebte Ziel in Rosette nicht umsetzbar ist. Ist ersteres der Fall können hier Sachen ausprobiert werden, an die ich bisher nicht gedacht habe, die aber im Prozess eventuell aufgekommen sind.

Woche 11.09 - 17.09: Puffer, wenn anderes länger dauert als erwartet und der Plan sich nach hinten verschiebt.

Woche 18.09 - 24.09: Am Anfang der Woche die Arbeit zum Korrekturlesen rausgeben, Abstract und Conclusion könnten gut noch Kapitel sein, die in dieser Woche verfasst werden, falls vorher noch nicht die Zeit dafür da war.

Woche 25.09 - 01.10: Texte mit Hilfe von Korrekturen überarbeiten, Fehler, die bisher nicht aufgefallen sind oder Textpassagen, die bis hierher nicht ausformuliert wurden, anpassen, Abgabe spätestens am 30.09.

5 Gliederung

Ganz grob wird die Arbeit gegliedert sein in:
Abstract
Einleitung
Problem
Verwandte Arbeit / Forschungsstand
Implementierung
Details

(Ergebnis)
Folgerungen

Die Einleitung soll das Thema einordnen und die wichtigen Begriffe und Features erklären. Das umfasst den Anfang des Motivationsteils dieses Exposés erweitert um Features, die ich letztendlich im Laufe der Arbeit verwendet habe. Außerdem könnte auf einiges noch genauer eingegangen werden, zum Beispiel der Begriff solver-aided oder die erwähnten Konstrukte von Rosette.
Im Kapitel Problem finden sich Teile der Motivation und der Aufgabenstellung, also eine Auseinandersetzung damit warum diese Arbeit relevant ist.
Der Forschungsstand ist das gleiche wie im Exposé, jedoch mit weiteren Publikationen einbezogen, die in diesem Rahmen erstmal außer Acht gelassen wurden. Hier kann dann auch schon näher darauf eingegangen werden was diese Arbeit von den anderen unterscheidet.
Im Kapitel Implementierung wird darauf eingegangen was die fertige Software letztendlich kann, wie sie verwendet wird, grob wie diese Funktionalität umgesetzt wurde und wie die Software aufgebaut ist.
Detailliert wird dies dann im nächsten Kapitel beschrieben. Hier werden dann auch getroffene Entscheidung im Programmierprozess behandelt und auf mögliche Hürden und alternative Wege, die erkundet wurden, eingegangen. Es wird eben genau ausgeführt was ich in dieser ganzen Zeit gemacht habe.
Im Kapitel Ergebnis (bzw. Folgerungen) wird dann wieder der Bezug zur Motivation (B-Maschinen) hergestellt, indem nochmal explizit auf die gewonnenen Erfahrungen eingegangen wird und wie man sich diese zu Nutzen machen kann. Außerdem wird diskutiert ob das Ziel zufriedenstellend erreicht wurde und welche Implikationen dies mit sich trägt.

References

[Abr96] Jean-Raymond Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996.

[TB13] Emina Torlak and Ratislav Bodik. Growing solver-aided languages with rosette. *Onward! 2013: Proceedings of the 2013 ACM international symposium on New ideas, new paradigms and reflections on programming & software*, pages 135–152, October 2013.

[Tor] Emina Torlak. The rosette language. https://emina.github.io/rosette/index.html. Accessed: 2023-07-10.