Entwicklung einer Anwendung zum Vermitteln von Lerninhalten über E-Graphs und Equality Saturation für die Lehre

Ben Steinhauer

Exposé zur Bachelorarbeit

Softwaretechnik und Programmiersprachen
Heinrich-Heine-Universität Düsseldorf
12. November 2024

- Formaler Betreuer: Dr. John Witulski
- Täglicher Betreuer: Dr. John Witulski
- Empfehlung für Zweitgutachter: Dr. Carl Friedrich Bolz-Tereick
- Gewünschter Starttermin: 20. November 2024

1 Wissenschaftlicher Hintergrund und Motivation

E-Graphs steht für *Equality*- oder *Equivalence*-Graphs und beschreibt eine baumartige Datenstruktur. Diese erlaubt es, äquivalente (mathematische) Ausdrücke kompakt abzuspeichern, indem Äquivalenzklassen der Ausdrücke abgespeichert werden. Die Äquivalenzrelation ist dabei kongruent. Ursprünglich wurde diese Datenstruktur nur in der automatisierten Beweisführung verwendet, wird aber heute auch für *Equality Saturation* genutzt, mit deren Hilfe zum Beispiel Compiler-Optimierungen durchgeführt werden können (Willsey u. a. 2021).
Der Ausdruck *Equality Saturation* beschreibt eine Technik, mit dessen Hilfe äquivalente Programme eines gegebenen Input-Programms gefunden werden können. Dies geschieht durch das Anwenden von Rewrite-Regeln auf das Input-Programm, wodurch neue Programme logisch abgeleitet werden können. Aus den exponentiell vielen Programmen kann man dann ein optimales extrahieren (Goharshady, Lam und Parreaux 2024).

2 Problemstellung

Das Ziel dieser Bachelorarbeit ist es, ein sinnvolles Werkzeug für die Lehre zu erstellen, um Studentinnen und Studenten die Themen **E-Graphs** und **Equality Saturation** näher zu bringen. Dabei sollen sie die Möglichkeit haben, sich sowohl auf theoretischer als auch praktischer Ebene mit E-Graphs auseinander setzen zu können. Die theoretische Ebene soll den Studenten die notwendigen Hintergrundkenntnisse vermitteln sowie einen Enblick in die Implementierung geben. Die praktische Ebene soll Schritt für Schritt aufzeigen, wie der **E-Graph** aufgebaut wird, und wie an diesem **Equality Saturation** durchgeführt werden kann. Für größtmöglichen Nutzen soll die Anwendung plattformunabhängig sein und möglichst nur von *Open-Source-Software* (OSS) Gebrauch machen. Damit wird das Problem der unterschiedlichen Betriebssysteme der Studenten umgangen und zeitgleich die Hürden für Erweiterungen gesenkt.

3 Verwandte Arbeiten

egg Das Akronym *egg* steht für *e-graphs good* und ist eine in der Programmiersprache *Rust* geschriebene Bibliothek. Die Bibliothek implementiert die Datenstruktur E-Graph und stellt Funktionen für die Interaktion mit dieser zur Verfügung. Neben der Erzeugung von E-Graphs kann auch *Equality Saturation* auf diesen durchgeführt werden. Das gesamte Projekt wurde in einem Paper vorgestellt, auf welches sich diese Arbeit stützt (Willsey u. a. 2021).

Graphviz Der Fokus der Software *Graphviz* liegt auf dem Visualisieren von Graphen. Dabei werden gängige Dateiformate unterstützt sowie zahlreiche Funktionen angeboten (Gansner und North 2000). Mit ihrer Hilfe sollen E-Graphs visualisiert werden.

4 Methodik

(Teil 1) Erzeugung des E-Graphs visualisieren

Der erste Teil der Bachelorarbeit beschäftigt sich mit dem Erstellen eines E-Graphs aus einem mathematischen Ausdruck sowie mit der Visualisierung der Datenstruktur. Die Erstellung soll schrittweise ausgeführt und visualisiert werden können. Die Bibliothek **egg** eignet sich dafür nur bedingt, denn detailierte Informationen über die Erstellung eines E-Graphs lassen sich nur über die *Log-Ausgabe* im *DEBUG-Mode* einsehen. Aus diesem Grund wird auf eine eigene Implementierung in der Programmiersprache **Python** gesetzt. Die Visualisierungen werden mithilfe von **Graphviz** erstellt.
Die Implementierung in Python wird zusätzlich auf theoretischer Ebene beleuchtet, um Studentinnen und Studenten Einblicke in die Theorie hinter Equality Saturation sowie die Implementierung von E-Graphs zu ermöglichen. Dabei sollen folgende inhaltliche Themen ausgeführt werden:

1. *Der theoretische Teil beginnt mit einer Einführung in das Thema E-Graphs.*
2. *Anschließend werden E-Graphs mit einem naiven Ansatz (einfache Listen) verglichen. Dabei werden Unterschiede, Vorteile und Probleme festgehalten.*
3. *Basierend auf dem zu der Bibliothek **egg** veröffentlichten Paper (Willsey u. a. 2021) werden im nächsten Abschnitt*
    a) *die zur Erstellung von E-Graphs nötigen Datenstrukturen behandelt.*
    b) *die Funktionsweise von E-Graphs erläutert.*
    c) *eine grobe Implementierung von E-Graphs beschrieben.*
4. *Im letzten Abschnitt wird eine Liste mit weiterführenden Informationen und Quellen bereitgestellt.*

(Teil 2) Anwendung des E-Graphs visualisieren

Zu der Implementierung soll im zweiten Teil eine Benutzeroberfläche erstellt werden, die Implementierungsdetails wegabstrahiert und einem Nutzer die Möglichkeit geben, auf einfache Weise mit E-Graphs zu interagieren. Dazu wird als Benutzeroberfläche eine lokale Website im Browser des Benutzers laufen, um auf platformspezifische Frameworks für die GUI-Entwicklung verzichten zu können. Für die Erstellung der Benutzeroberfläche wird ***HTML***, ***CSS*** und ***Javascript*** eingesetzt. Das Backend bildet ein Webserver, der mithilfe des Framworks ***FastAPI*** erzeugt werden soll. Zusätzlich kann dieser auf die Implementierung der E-Graphs (*Teil 1*) zurückgreifen. Insgesamt sollen dem Benutzer folgende Funktionen zur Verfügung gestellt werden:

1. ***GUI*** *Die Anwendung stellt eine Benutzeroberfläche in Form einer HTML-Website zur Verfügung, die lokal im Browser läuft. Alle Funktionen sowie der theoretische Teil können darüber abgefragt werden.*
2. ***E-Graph*** *Mit der Anwendung ist es möglich, durch Eingabe eines mathematischen Terms, einen E-Graph schrittweise zu erzeugen und in der GUI durch eine SVG-Datei anzeigen zu lassen.*

3. ***Mathematische Terme*** *Terme bestehen aus Variablen (Buchstaben wie x, y, z), Operationen ($+, *, -, /, <<, >>$) und Zahlen.*
4. ***Rewrite Rules*** *Der Benutzer hat die Möglichkeit, Rewrite Rules zu definieren und diese unabhängig voneinander auf den E-Graph anzuwenden. Die dabei entstehenden Veränderungen spiegelt der E-Graph wieder, d.h. die Darstellung wird aktualisiert.*
5. ***Default Rewrite Rules*** *Wenn der Benutzer die Anwendung startet, sollen bereits einige vordefinierte Rewrite Rules eingespeichert werden, um dem Benutzer Arbeit zu sparen und ihm gleichzeitig als Beispiel dienen.*
6. ***Historie*** *Mithilfe der GUI ist es möglich, zwischen verschiedenen Versionen des E-Graphs zu wechseln und diese anzuzeigen.*
7. ***Extraktion*** *Der finale, optimierte Term kann aus dem E-Graph extrahiert werden.*
8. ***Export*** *Dem Benutzer ist es möglich, den momentan angezeigten E-Graph in ein Dateiformat wie PDF oder SVG zu exportieren.*
9. ***Sessions*** *Aus Effizienzgründen kann der Benutzer seine jeweilige Session in einem für Menschen lesbaren Dateiformat abspeichern und bei einem erneuten Start der Anwendung wieder laden. Eine Session besteht dabei aus: dem mathematischen Term, den eingegebenen Rewrite Rules inklusive der momentan angewendeten und dem evtl. bereits extrahierten, optimalen Term.*
10. ***Hilfe*** *Durch die Benutzeroberfläche kann der Nutzer auf eine Dokumentation zurückgreifen, in der die Bedienung der Anwendung erklärt wird.*

4.1 Optionale Funktionen

Im Folgenden werden optionale Funktionen aufgelistet, die, wenn es der Zeitplan zulässt, ebenfalls in die Anwendung integriert werden sollen.

- **Englisch-Support** Um auch internationalen Studenten ein hilfreiches Werkzeug bieten zu können, soll die Benutzeroberfläche sowie alle Ressourcen bei Bedarf auf Englisch zur Verfügung stehen. Dazu sollte nur minimaler Aufwand seitens des Benutzers erforderlich sein.
- **Dark Mode** Die grafische Benutzeroberfläche soll mit einem Button ausgestattet werden, der es dem Benutzer erlaubt, zwischen *Dark Mode* und *Light Mode* zu wechseln.

5 Ressourcen

Für die geplante Bachelorarbeit werden keine Ressourcen des Lehrstuhls oder der Universität benötigt.

6 Hindernisse und Schwierigkeiten

Der Entwicklung der geplanten Anwendung stellen sich momentan verschiedene Schwierigkeiten in den Weg, die im Folgenden aufgelistet werden:

- **Geschwindigkeit** Die Zeit zwischen einer Nutzereingabe und dem Aktualisieren der Visualisierung im Browser sollte möglichst klein sein. Da aber momentan noch nicht bekannt ist, wie schnell die eigene Implementierung sein wird, kann dies erst im Verlauf der Bachelorarbeit festgestellt werden.
- **Dynamische Generierung von Inhalten** Es gibt verschiedene Methoden, um dynamische Inhalte auf einer Website zu generieren (ModelAndView, Javascript). Bis jetzt steht noch nicht fest, welche dieser Methoden die schnellste und effizienteste ist. Dieses Problem könnte durch eine Recherche und Entwicklung eines kleinen Prototypen gelöst werden.
- **Visualisierung** Zum Visualisieren soll mit *Graphviz* gearbeitet werden. Diese erfordert jedoch Kenntnisse zum Umgang, die bis jetzt noch nicht erlangt wurden.
- **Sessions** Die Erstellung der Sessions ist noch nicht komplett durchdacht. Zum Beispiel steht das Dateiformat einer Session noch nicht fest. Denkbar wäre *JSON*, da das Format in Python gut verarbeitet werden kann.
- **CSS-Framework** Für das Frontend eignet sich die Einbindung eines *CSS-Frameworks*, das das Design vereinfacht und standardisiert. Hier wurde noch keine Auswahl getroffen. Möglich wäre *Bootstrap*.
- **FastAPI** Der Umgang mit dem Framework *FastAPI* ist noch nicht ganz klar. Tutorials und die Dokumentation sollten aber Abhilfe verschaffen.

7 Ungefährer Zeitplan

Der vorliegende Zeitplan beschreibt eine grobe Vorgehensweise und kann sich im Verlauf noch ändern. Im Folgenden werden die wichtigsten Meilensteine aufgelistet, die in dieser Reihenfolge erreicht werden sollten. Für jeden dieser Meilensteine sind ca. 2 Wochen eingeplant, wobei manche eventuell schneller erreicht werden könnten.

1. Implementierung von E-Graphs mit Python 1
2. Implementierung von E-Graphs mit Python 2
3. Theorie
4. Aufsetzen des Servers und Integration mit bestehender Komponente
5. Frontend mit grundlegenden Funktionen
6. Voller Funktionsumfang und letzte Tests

8 Betreuung

Die Betreuung der Bachelorarbeit übernimmt Dr. John Witulski. Dr. Carl Friedrich Bolz-Tereick unterstützt diese Arbeit ebenfalls. Geplant sind wöchentliche Meetings, die ca. eine Stunde dauern. Abweichungen können auftreten.

Literatur

[1] Max Willsey u. a. “egg: Fast and Extensible Equality Saturation”. In: *Proc. ACM Program. Lang.* 5.POPL (Jan. 2021). DOI: 10.1145/3434304. URL: https://doi.org/10.1145/3434304.

[2] Amir Kafshdar Goharshady, Chun Kit Lam und Lionel Parreaux. “Fast and Optimal Extraction for Sparse Equality Graphs”. In: *Proc. ACM Program. Lang.* 8.OOPSLA2 (Okt. 2024). DOI: 10.1145/3689801. URL: https://doi.org/10.1145/3689801.

[3] Emden R. Gansner und Stephen C. North. “An open graph visualization system and its applications to software engineering”. In: *Software: Practice and Experience* 30.11 (2000), S. 1203–1233. DOI: https://doi.org/10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/1097-024X%28200009%2930%3A11%3C1203%3A%3AAID-SPE338%3E3.0.CO%3B2-N. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/1097-024X%28200009%2930%3A11%3C1203%3A%3AAID-SPE338%3E3.0.CO%3B2-N.