Graphtransformationssyteme in Constraint Handling Rules: Verbesserte Methoden zur Programmanalyse

Frank Raiser

Universität Ulm
frank.raiser@uni-ulm.de

Abstract: Constraint Handling Rules (CHR) und Graphtransformationssysteme (GTS) sind regelbasierte Formalismen. In dieser Arbeit wird eine kompakte äquivalenzbasierte Semantik für CHR vorgestellt. Darauf aufbauend werden bestehende Programmanalysemethoden für CHR verbessert. Abschließend wird ein Vergleich von CHR und GTS durchgeführt, indem GTS in CHR eingebettet werden und CHR Programmanalysen auf Graphtransformationssysteme angewandt werden.

1 Einführung

In den letzten Jahren haben wir eine Renaissance der regel- und logikbasierten Ansätze beobachten können. Regelbasierte Programmierung gibt es bereits seit Jahrzehnten, wobei die Termersetzung (vgl. [BN98, Ohl02] für einen Überblick) ihr bekanntester Vertreter ist. Neben ihrer Bedeutung für die funktionale Programmierung, hat die regelbasierte Programmierung jedoch wenig Aufmerksamkeit außerhalb der Forschung erfahren.

Constraint Handling Rules (CHR) ist eine universelle Programmiersprache, die formal mittels eines Zustandsübergangssystems definiert ist und sowohl regel- als auch logikbasiert ist. Sie wurde ursprünglich von Frühwirth in den frühen 90igern konzipiert und wurde in den letzten beiden Jahrzehnten kontinuierlich weiterentwickelt [Frü98, Frü09, SVWSDK10]. CHR zeigt dieselben Tendenzen wie viele regel- oder logikbasierte Ansätze, indem hunderte wissenschaftliche Publikationen darüber existieren, aber wir gerade erst den Beginn der Anwendung durch Industriebenutzer beobachten.

Aus dem großen Fundus an Formalismen für regel- und logikbasierte Programmierung beschränkt sich diese Arbeit vorrangigg auf CHR, da es beide Paradigmen kombiniert. Es ist regelbasiert und bietet eine Semantik erster Ordnung [Frü09], sowie eine linearlogische Semantik [BF05].

Darüberhinaus wurde CHR als eine *lingua-franca* der regelbasierten Formalismen vorgeschlagen, d.h. “ein zentraler Formalismus, der Forschungsergebnisse aus verschiedenen angrenzenden Gebieten sammelt und an diese weiterreicht” (übersetzt aus [Sne08]). Es existiert bereits ein großer Fundus an Arbeiten über den Vergleich von CHR mit anderen regelbasierten Ansätzen (vgl. [Frü09, SVWSDK10]), was zu wertvollen Ergebnissen in den verglichenen Gebieten führte.

Im Rahmen dieser Arbeit soll dieses lingua-franca Argument weiter verfolgt werden. Dazu wird CHR mit algebraischen Graphtransformationssystemen (GTS) [EK04] verglichen. Diese bilden die Grundlagen für Graphersetzungssysteme und finden etwa Anwendung im Bereich des Model Driven Development. Neben der korrekten und vollständigen Einbettung von GTS in CHR, werden zudem Programmanalysemethoden beider Formalismen verglichen.

Ein Hauptaugenmerk dieser Arbeit liegt ferner darin, das lingua-franca Argument zu stärken, indem die operationelle Semantik von CHR vereinfacht wird. Dies macht Vergleiche zu anderen Formalismen auf formaler Ebene übersichtlicher, verständlicher und reduziert die Komplexität und Länge benötigter Beweise.

Um die Vorteile dieser vereinfachten Semantik ersichtlich zu machen führen wir diese zu Beginn in Kapitel 2 ein und wenden sie anschließend im restlichen Teil der Arbeit an. Kapitel 3 greift eine invariantenbasierte Programmanalysemethode von [DSS07] auf, welche auf der Basis unserer vereinfachten Semantik verallgemeinert wird. Abschließend werden diese Ergebnisse in Kapitel 4 zusammengeführt, um Graphtransformationssysteme in CHR einzubetten und die verbesserten Programmanalysemethoden darauf anzuwenden.

2 Äquivalenzbasierte Operationelle Semantik für Constraint Handling Rules

Die Grundlage für unsere vereinfachte operationelle Semantik bildet die Äquivalenz von CHR-Zuständen, welche unter CHR-Anwendern intuitiv sehr gut verstanden ist. Versuche, sie formal zu charakterisieren, haben jedoch zu widersprüchlichen Definitionen geführt. In [RBF09] wird eine fundierte axiomatische Definition eingeführt, die allen Anforderungen an eine Äquivalenzrelation für CHR-Zustände genügt und für die zudem ein notwendiges und hinreichendes Kriterium gezeigt wird.

Ein CHR Zustand $\langle \mathbb{G}; \mathbb{B}; \mathbb{V} \rangle$ besteht dabei aus einer Multimenge $\mathbb{G}$ von CHR Constraints, einer Konjunktion von Prädikaten $\mathbb{B}$ und einer Menge $\mathbb{V}$ von globalen Variablen. Variablen in $\mathbb{G}$ und $\mathbb{B}$, die nicht in $\mathbb{V}$ auftauchen, heißen entsprechend lokale Variablen. Die Zustandsäquivalenz lässt sich intuitiv leicht verstehen: So sind etwa die Zustände $\langle c(X); X = 1; \emptyset \rangle$ und $\langle c(Y); Y = 1; \emptyset \rangle$ äquivalent, nicht jedoch die Zustände $\langle c(X); X = 1; \emptyset \rangle$ und $\langle c(Y); Y = 2; \emptyset \rangle$.

2.1 Äquivalenzbasierte Operationelle Semantik ω_e

Basierend auf der formalen Definition der Äquivalenz von CHR-Zuständen lässt sich die operationelle Semantik von CHR aus einer neuen Sichtweise darstellen: Anstatt Zustände zu normalisieren und bei Bedarf deren Äquivalenz zu untersuchen, wird das Zustandsübergangssystem direkt über den Äquivalenzklassen der Zustände definiert.

Dies bringt mehrere Vorteile mit sich: Erstens vereinfacht sich die formale Definition des

$$\frac{r \mathbin{@} H_1 \setminus H_2 \Leftrightarrow G \mid B_c \uplus B_b}{[\langle H_1 \uplus H_2 \uplus \mathbb{G}; G \wedge \mathbb{B}; \mathbb{V}\rangle] \rightarrowtail_e^r [\langle H_1 \uplus B_c \uplus \mathbb{G}; G \wedge B_b \wedge \mathbb{B}; \mathbb{V}\rangle]}$$

Abbildung 1: Zustandsübergangssystem ω_e mit Äquivalenzklassen

Zustandsübergangssystems, wie weiter unten zu sehen ist. Zweitens wird die Behandlung von Regelwächtern vereinfacht. Diese dienen dazu, dass Regeln nur ausgeführt werden dürfen, wenn der Wächter erfüllt ist. Ein Zustand, welcher den Wächter jedoch erfüllt, ist äquivalent zu einem Zustand, in dem dieser Wächter in der Konjunktion $\mathbb{B}$ enthalten ist. Außerdem bieten die Zustandsäquivalenzklassen den Vorteil, dass im Rahmen formaler Analysen stets ein geeigneter Repräsentant der Klasse ausgewählt werden kann. Im Rahmen dieser Arbeit hat sich gezeigt, dass dadurch eine enorme Reduzierung der Komplexität und Länge von Beweisen möglich ist.

Die resultierende operationelle Semantik lässt sich dann wie folgt definieren. Dabei ist zu beachten, dass lediglich eine Inferenzregel benötigt wird, im Gegensatz zu existierenden Definitionen, welche bis zu einem Dutzend Inferenzregeln beinhalten.

Definition 2.1 (Operationelle Semantik ω_e) *Für ein CHR Programm $\mathcal{P}$ ist das Zustandsübergangssystem $(\Sigma_e/\equiv_e, \rightarrowtail_e)$ in Abbildung 1 gegeben. Die Übergänge basieren auf einer Variante einer Regel r in $\mathcal{P}$, so dass deren lokale Variablen disjunkt von den Variablen im Repräsentanten des initialen Zustands sind.*

Diese Definition eignet sich zudem didaktisch zur Vermittlung der operationellen Semantik von CHR, da anhand der Inferenzregel die grundlegende Funktionsweise anschaulich demonstriert werden kann. So sieht man in der Prämisse der Inferenzregel von Abbildung 1 den Aufbau einer CHR Regel: $r \mathbin{@} H_1 \setminus H_2 \Leftrightarrow G \mid B_c \uplus B_b$

Eine solche Regel besteht aus einem Regelnamen r, einem Kopf H_1 und H_2, einem Wächter G sowie einem Rumpf, der aus CHR Constraints B_c und Prädikaten B_b besteht. Man sieht ferner an Hand der Conclusio, dass die Anwendung einer solchen Regel auf einen CHR Zustand möglich ist, falls der gesamte Kopf, bestehend aus H_1 und H_2, sowie der Wächter G in diesem enthalten sind.

Die Auswirkung einer Regel auf den jeweiligen Zustand lässt sich durch den Vergleich der beiden Zustände vor bzw. nach dem Übergang bestimmen: Der Teil H_1 des Kopfes bleibt erhalten, während der zweite Teil H_2 entfernt wird. Dafür wird der Rumpf der Regel in die entsprechenden Komponenten des Zustands eingefügt.

Das Verständnis dieser intuitiven Funktionsweise von CHR-Regeln wurde bisher durch komplizierte Zustandsübergangssysteme erschwert. Mit unserer vereinfachten Semantik wird diese klar herausgestellt. Ferner können damit auch Berechnungen wesentlicher effizienter erfasst werden, wie das folgende Beispiel zeigt.

Beispiel 2.2 (Größter gemeinsamer Teiler) *Die folgenden beiden Regeln dienen der Berechnung des größten gemeinsamen Teilers auf Basis des Euklidschen Algorithmus. Die*

erste Regel stellt dabei den Spezialfall $H_1 = \emptyset$ dar, in dem H_1 zusammen mit $\setminus$ weggelassen wird.

$$\begin{array}{llll} \mathrm{ggT}_1 \ @ & \mathrm{gcd}(0) & \Leftrightarrow & \top \\ \mathrm{ggT}_2 \ @ & \mathrm{ggT}(N) \setminus \mathrm{ggT}(M) & \Leftrightarrow & N > 0 \wedge M \geq N \mid \ L = M\%N, \mathrm{ggT}(L) \end{array}$$

Eine Berechnung, etwa für die Eingabezahlen 6 und 15, kann mit der operationellen Semantik ω_e nun wie folgt formuliert werden. Dabei werden implizit Äquivalenzen ausgenutzt, wie etwa die von $\mathrm{ggT}(6)$ und $\mathrm{ggT}(M), M = 6$. Im Gegensatz dazu nimmt die Darstellung einer vollständigen Berechnung für die gleiche Eingabe mit anderen existierenden Formalismen ein Vielfaches an Platz in Anspruch und verliert dabei an Übersichtlichkeit und Aussagekraft (vgl. [Rai10, Kap. III]). In der folgendenen Berechnung wird das Zeichen $=$ verwendet, um anhand der Äquivalenzrelation $\equiv_e$ geeignete Repräsentanten auszuwählen. Da es sich dabei jeweils um die gleiche Äquivalenzklasse handelt, ließe sich die Berechnung auch in vier Zeilen darstellen.

$$\begin{array}{ll} & [\langle \mathrm{ggT}(6), \mathrm{ggT}(15); \top; \emptyset \rangle] \\ = & [\langle \mathrm{ggT}(N), \mathrm{ggT}(M); M \geq N \wedge N > 0 \wedge N = 6 \wedge M = 15; \emptyset \rangle] \\ \rightarrowtail_e^{\mathrm{ggT}_2} & [\langle \mathrm{ggT}(N), \mathrm{ggT}(L); M \geq N \wedge N > 0 \wedge L = M\%N \wedge N = 6 \wedge M = 15; \emptyset \rangle] \\ = & [\langle \mathrm{ggT}(6), \mathrm{ggT}(3); \top; \emptyset \rangle] \\ = & [\langle \mathrm{ggT}(N), \mathrm{ggT}(M); M \geq N \wedge N > 0 \wedge N = 3 \wedge M = 6; \emptyset \rangle] \\ \rightarrowtail_e^{\mathrm{ggT}_2} & \langle \mathrm{ggT}(N), \mathrm{ggT}(L); M \geq N \wedge N > 0 \wedge L = M\%N \wedge N = 3 \wedge M = 6; \emptyset \rangle] \\ = & [\langle \mathrm{ggT}(3), \mathrm{ggT}(0); \top; \emptyset \rangle] \\ \rightarrowtail_e^{\mathrm{ggT}_1} & [\langle \mathrm{gcd}(3); \top; \emptyset \rangle] \end{array}$$

2.2 Merge Operator

Bei der formalen Analyse von Programmen, etwa im Rahmen von Programmanalysemethoden, muss oft ein Teil eines CHR Zustandes untersucht werden. In [Rai10] wurde daher ein Operator definiert, der es erlaubt, den für die aktuelle Betrachtung irrelevanten Bestandteil des Zustands zu extrahieren. Dadurch kann die Analyse auf den relevanten Teil beschränkt werden und aufgrund der zugehörigen Monotonieresultate der extrahierte Zustandsteil implizit mit berücksichtigt werden. Dieser Merge Operator $\diamond_{\mathbb{V}}$ ist wie folgt definiert und stellt eine parameterisierbare Addition von CHR Zuständen, sowie Äquivalenzklassen von Zuständen, dar.

Definition 2.3 (Merge Operator $\diamond_{\mathbb{V}}$) *Seien $\sigma_1 = \langle \mathbb{G}_1; \mathbb{B}_1; \mathbb{V}_1 \rangle$ und $\sigma_2 = \langle \mathbb{G}_2; \mathbb{B}_2; \mathbb{V}_2 \rangle$, so dass die lokalen Variablen eines Zustands disjunkt von den Variablen des anderen Zustands sind. Dann gilt für eine Menge $\mathbb{V}$ von Variablen $\sigma_1 \diamond_{\mathbb{V}} \sigma_2 ::= \langle \mathbb{G}_1 \uplus \mathbb{G}_2; \mathbb{B}_1 \wedge \mathbb{B}_2; (\mathbb{V}_1 \cup \mathbb{V}_2) \setminus \mathbb{V} \rangle$. Diese Definition lässt sich zudem auf Äquivalenzklassen übertragen. In diesem Fall gehen wir davon aus, dass die beiden Repräsentanten die obige Variablenanforderung erfüllen: $[\sigma_1] \diamond_{\mathbb{V}} [\sigma_2] ::= [\sigma_1 \diamond_{\mathbb{V}} \sigma_2]$. Für $\mathbb{V} = \emptyset$ schreiben wir $\sigma_1 \diamond \sigma_2$, bzw. $[\sigma_1] \diamond [\sigma_2]$.*

Neben der oben genannten Möglichkeit, Zustandsbestandteile zu extrahieren, ergeben sich noch weitere praktische Konsequenzen dieses Operators. So bildet die Menge der

Äquivalenzklassen von CHR Zuständen damit ein kommutatives Monoid $(\Sigma/\equiv_e, \diamond)$, was wiederum eine partielle Ordnung $\lhd$ auf Äquivalenzklassen von CHR Zuständen impliziert. Diese partielle Ordnung spielt eine wichtige Rolle für die Programmanalysemethoden im folgenden Kapitel.

3 Verbesserte Programmanalysemethoden

Bettet man im Rahmen des lingua-franca Arguments einen Formalismus in CHR ein und versucht anschließend CHR Programmanalysemethoden auf das Ergebnis anzuwenden, so tritt ein inhärentes Problem zu Tage: die Einbettung verwendet üblicherweise nur eine Untermenge von CHR und die für diese Einbettung relevanten Zustände bilden ebenfalls nur eine Teilmenge aller möglichen CHR Zustände. Im Falle unserer Einbettung von Graphtransformationssystemen gibt es etwa bestimmte CHR Zustände, die der Kodierung eines Graphen entsprechen, während alle anderen Zustände als irrelevant angesehen werden.

Eine mögliche Lösung, wie Programmanalysemethoden sich dieses Problems annehmen können stellt [DSS07] am Beispiel der Konfluenzanalyse vor. Dabei wird eine bestehende Methode um eine Invariante erweitert, welche die relevanten Zustände charakterisiert.

Derartige Analysen bauen meist darauf auf, dass aus den Regeln des Programms zu testende Zustände abgeleitet werden. Im Normalfall sind Aussagen über diese Zustände aufgrund der darin verwendeten globalen Variablen allgemein genug, um daraus Aussagen für alle möglichen Zustände zu implizieren. Dabei ist die Überlegung bzgl. der Addition von Zuständen, die dem obigen Merge Operator zugrunde liegt, implizit enthalten. Ausgehend von den Ergebnissen der Untersuchung eines Zustands werden diese Ergebnisse auf alle "größeren" Zustände übertragen.

Bei Einbeziehung einer Invariante kann jedoch der Fall auftreten, dass die zu testenden Zustände selbige nicht erfüllen. Dann sind daraus gewonnene Ergebnisse irrelevant und vor allem auch nicht übertragbar auf weitere Zustände. Zu diesem Zweck lassen sich *minimale Elemente* (welche ebenfalls CHR Zustände sind) bezüglich der Ordnung $\lhd$ definieren (vgl. [Rai10, Kap. III]), so dass die betrachteten Zustände, erweitert um einen derartigen Zustand, die Invariante erfüllen. Aus dem kombinierten Zustand lassen sich dann wieder sinnvolle Aussagen ableiten und auf alle, bzgl. der Ordnung $\lhd$, größeren Zustände übertragen.

Dieser Sachverhalt ist in Abbildung 2 dargestellt, wobei $[\sigma]$ ein aus den Regeln des Programms abgeleiteter zu untersuchender Zustand ist. Die Zustände $[\sigma \diamond \sigma_i](i = 1, \ldots, k)$ werden dann aus den minimalen Elementen gewonnen. Dies sind die Zustände, welche von einer Programmanalysemethode letztlich untersucht werden müssen. Dahingegen lassen sich die daraus gewonnenen Ergebnisse auf alle größeren Zustände, die in der obersten Zeile von Abbildung 2 angedeutet sind, übertragen.

Diese erweiterten Methoden sind echt mächtiger als ihre klassischen Varianten. Im Falle eines Zustandes, welcher die Invariante erfüllt, besteht die Menge der minimalen Elemente stets aus dem leeren Zustand, welcher das neutrale Element des kommutativen Monoids darstellt. Für eine Invariante, welche für alle Zustände erfüllt ist, erhält man somit stets das

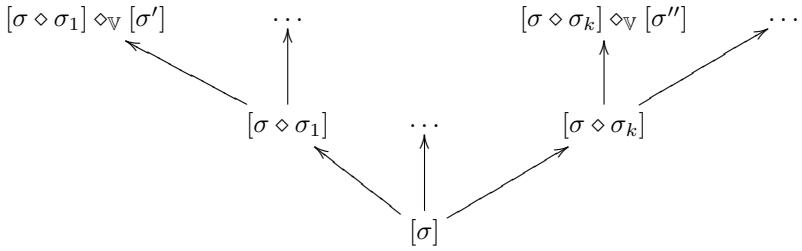

Abbildung 2: Invariantenbasierte Analyse mit minimalen Elementen $\sigma_1,\ldots,\sigma_k$

klassische Verfahren: die Erweiterungen der zu untersuchenden Zustände um minimale Elemente sind dann gerade die Zustände selbst.

In der Arbeit wird diese Parameterisierung von Programmanalysemethoden mit Invarianten exemplarisch für die Konfluenz- und Programmäquivalenzanalysen aus der CHR Literatur demonstriert.

4 Graphtransformationssysteme in Constraint Handling Rules

Dieses abschließende Kapitel befasst sich mit der Einbettung von Graphtransformationssystemen in Constraint Handling Rules sowie der Durchführung von Programmanalysemethoden auf einem eingebetteten GTS.

4.1 Kodierung

Eine Kodierung eines GTS in CHR ist verhältnismäßig leicht anzugeben. Man muss jedoch zeigen, dass mittels dieser Kodierung auch ein sinnvolles Verhalten erzielt wird. Dieser Nachweis von Korrektheit und Vollstndigkeit der Einbettung wurde in [Rai07] erstmalig erbracht. Der Vergleich mit einer Beweisführung auf Basis der oben vorgestellten äquivalenzbasierten operationellen Semantik von CHR [Rai10] hat deutlich deren Vorteile in Bezug auf geringere Komplexität und mehr Kompaktheit belegt.

Eine genauere Betrachtung der in [Rai10] vorgestellten Einbettung zeigt zudem, dass eine Simulation der Ausführung eines GTS in CHR auch für unvollständige Graphen möglich ist. Mit Hilfe globaler Variablen lassen sich Knoten eines Graphen auswählen, die im Rahmen der Transformation erhalten bleiben müssen. Dies erlaubt es, die Auswirkungen eines Graphtransformationssystems auf eine Menge von Graphen zu untersuchen. Diese Menge ist dabei durch den gegebenen Eingabegraph charakterisiert, wobei die durch globale Variablen designierten Knoten den Anschluss beliebiger Untergraphen erlauben.

4.2 Programmanalyse

Die im vorherigen Kapitel besprochenen Programmanalysemethoden können nun auf eingebettete Graphtransformationssysteme angewandt werden. Zu diesem Zweck wurde eine Invariante definiert, welche für Zustände erfüllt ist, die eine gültige Kodierung eines Graphen enthalten. Die Berücksichtigung dieser Invariante und damit der erweiterten Analysemethoden ist nötig, da die bestehenden Programmanalysemethoden nicht direkt anwendbar sind. So würden meist Zustände betrachtet, die bei Eingabe eines korrekt kodierten Graphs nie erreicht werden können. Unter Zuhilfenahme dieser Invariante wird zudem gezeigt, dass die Äquivalenz von CHR Zuständen, die Graphen kodieren, die Isomorphie dieser Graphen impliziert.

Eine besondere Programmanalyse im Vergleich von GTS und CHR ist die Konfluenzanalyse. Für terminierende Systeme ist diese im Fall der Graphtransformationssysteme unentscheidbar, im Fall von terminierenden CHR Programmen jedoch entscheidbar. Da aufgrund der korrekten und vollständigen Einbettung von Graphtransformationssystemen in CHR die Terminierung eines solchen Systems erhalten bleibt, stellt sich daher die Frage, wie das Ergebnis einer CHR-basierten Konfluenzanalyse eines eingebetteten GTS-Programms zu interpretieren ist.

Hier wurde im Rahmen der Arbeit festgestellt, dass die entscheidbare Konfluenzanalyse in CHR als hinreichendes, jedoch nicht notwendiges, Kriterium für die Konfluenz des eingebetteten Graphtransformationssystems dient. Es wird gezeigt, dass eine erfolgreiche Konfluenzanalyse mit einem hinreichenden Kriterium für die Konfluenz von GTS korrespondiert. Sollte die Konfluenzanalyse ein negatives Resultat ergeben, so lässt sich daraus keine Aussage über die Konfluenz des untersuchten Graphtransformationssystem ableiten. In diesem Fall existieren sowohl konfluente, als auch nicht-konfluente Systeme.

Im Gegensatz zur allgemeinen invariantenbasierten Programmanalyse lässt sich diese Art der Konfluenzanalyse problemlos automatisieren. Dies ergab eine genauere Untersuchung der involvierten minimalen Elemente: im Fall von Zuständen, welche nicht der Kodierung eines Graphs entsprechen, ist es durch Hinzufügen von minimalen Elementen nicht möglich, die Invariante zu erfüllen. Daher reduziert sich das im vorigen Kapitel vorgestellte allgemeine Verfahren zu einer Art Projektion auf relevante, d.h. Graphen kodierende, Zustände.

Zuletzt wurde mit der Programmäquivalenzanalyse eine aus dem CHR-Bereich stammende Analysemethode zur Anwendung auf Graphtransformationssysteme übertragen. Obwohl Programmäquivalenz im Allgemeinen unentscheidbar ist, hat sich diese Methode im CHR-Bereich bewährt, um damit bei automatisch generierten Programmen redundante Regeln zu erkennen und zu entfernen. In der Arbeit wurde erfolgreich gezeigt, das dies auch im Kontext der in CHR eingebetteten Graphtransformationssysteme möglich ist.

5 Zusammenfassung und Ausblick

Die vorgestellte Arbeit befasst sich in ihrem Kern mit der Idee von CHR als lingua-franca für regel- und logikbasierte Formalismen. In diesem Bezug wurde die Formulierung der operationellen Semantik von CHR entscheidend verbessert. Die zu diesem Zweck eingeführte Semantik ω_e basiert auf einer axiomatischen Definition der Zustandsäquivalenz in CHR, für die zudem ein Entscheidungskriterium gegeben wird. Dadurch ist eine neue Sichtweise auf die operationelle Semantik von CHR, basierend auf dem Ersetzen von Äquivalenzklassen von Zuständen, möglich.

Weiterhin haben wir einen Merge Operator für die Kombination von zwei CHR Zuständen, bzw. Zustandsäquivalenzklassen, eingeführt. Wir haben wichtige Eigenschaften dieses Operators identifiziert, insbesondere, dass er zusammen mit den Zustandsäquivalenzklassen ein kommutatives Monoid bildet. Dies wird genutzt, um eine partielle Ordnung über CHR Zuständen zu definieren, welche wiederum im Kontext der Programmanalysen Anwendung findet.

Zusammenfassend bietet unsere äquivalenzbasierte operationelle Semantik mehrere Vorteile gegenüber existierenden Formulierungen: deklarative Zustände, ein Zustandsübergangssystem, bei dem jede Transition direkt mit einer Regelanwendung korrespondiert und eine solide Zustandsäquivalenzdefinition. Zudem ist diese Formulierung geeignet zur Beschreibung von informell dargestellten Berechnungen. Diese können in der äquivalenzbasierten operationellen Semantik auf intuitiver Ebene verfasst, und dabei gleichzeitig formal korrekt aufgeschrieben werden.

Im dritten Kapitel untersuchen wir die Verwendung von Invarianten in der Analyse von CHR Programmen. Insbesondere diskutieren wir die Anwendung unserer partiellen Ordnung von CHR Zuständen für die Definition von minimalen Elementen. Nachdem wir die Auswirkungen der Einbeziehung einer Invariante auf die Programmanalysemethoden untersucht haben, können wir so die Ergebnisse zur invariantenbasierten Konfluenz von [DSS07] in unserem äquivalenzbasierten Kontext reproduzieren.

Ferner identifizieren wir mögliche Verbesserungen für die Programmäquivalenzergebnisse aus [AF99], insbesondere die Unterstützung von Programmschnittstellen. Außerdem erweitern wir ihre Ergebnisse, was zu einem notwendigen und hinreichenden Kriterium für die invariantenbasierte Programmäquivalenz führt.

Diese Arbeit macht die Programmanalysemethoden auf eine strikt größere Menge von CHR Programmen anwendbar. Die Einbeziehung einer Invariante stellt dabei sicher, dass implizite Annahmen explizit gemacht werden. Formulierungen dieser Methoden, die in der CHR Literatur existieren, haben üblicherweise Schwierigkeiten mit Programmen, die gängie Strukturen, wie Zahlen, (verkettete) Listen oder Graphen verwenden, da dabei inkonsistente Zustände getestet werden. Im Gegensatz dazu stellen unsere Methoden sicher, dass die getesten Zustände konsistente Datenstrukturen enthalten, was ein negatives Ergebnis eines solchen Tests aussagekräftiger macht.

Zuletzt findet unsere äquivalenzbasierte operationelle Semantik Anwendung in der Einbettung von Graphtransformationssystemen in Constraint Handling Rules. Dabei zeigt sich, dass dadurch elegantere Formulierungen und kompaktere Beweise ermöglicht werden. Wir

beweisen die Korrektheit und Vollständigkeit der Einbettung und untersuchen deren Eigenschaften. Dazu identifizieren wir eine Invariante für gültige Graphen und untersuchen die Korrespondenz zwischen Graphisomorphismus und Zustandsäquivalenz. Außerdem zeigen wir, wie die CHR Kodierung auf unvollständige Graphen angewandt werden kann, was in einem gewöhnlichen GTS nicht möglich wäre.

Die Arbeit gipfelt schließlich in der Untersuchung von Konfluenz und Programmäquivalenz eingebetteter Graphtransformationssysteme, was alle vorherigen Ergebnisse involviert: die Kodierung von GTS in CHR, unsere Ergebnisse zur invariantenbasierten Programmanalyse und die äquivalenzbasierte operationelle Semantik als formale Grundlage. Wir zeigen, dass die invariantenbasierte Konfluenzanalyse auf einem eingebetteten terminierenden GTS ein hinreichendes Kriterium für dessen Konfluenz ist und geben ein erstes Programmäquivalenzergebnis für Graphtransformationssysteme an, welches wir zum Entfernen redundanter Regeln anwenden.

Diese Arbeit bestätigt daher die Eignung von CHR als lingua-franca. Unsere äquivalenzbasierte operationelle Semantik erweist sich in diesem Forschungsbereich als vorteilhaft und unsere Ergebnisse demonstrieren erfolgreich die Möglichkeit Forschungsergebnisse aus den verglichenen Bereichen zu übertragen.

In den einzelnen Teilbereichen gibt es dennoch noch Forschungspotenzial: der Merge Operator stellt lediglich den Beginn einer allgemeineren algebraischen Betrachtungsweise von CHR dar. Seine Assoziativität ist zudem vorteilhaft, um die Parallelisierung von CHR auf algebraische Art zu beschreiben.

Am meisten Potenzial befindet sich jedoch noch im Zusammenspiel von Graphtransformationssystemen und Constraint Handling Rules. So führt eine Erweiterung unserer Ergebnisse auf attributierte GTS zu Einsatzmöglichkeiten im Bereich des Model-Driven Development. Mittels eines einzelnen CHR Programms könnten dann Transformationen und Constraints der Object Constraint Language [OCL10] modelliert und ausgeführt, sowie mit einer linearlogischen Semantik versehen werden.

Literatur

[AF99] Slim Abdennadher and Thom Frühwirth. Operational Equivalence of CHR Programs and Constraints. In J. Jaffar, editor, *CP '99*, volume 1713 of *Lecture Notes in Computer Science*, pages 43–57. Springer-Verlag, October 1999.

[BF05] Hariolf Betz and Thom Frühwirth. A Linear-Logic Semantics for Constraint Handling Rules. In P. van Beek, editor, *CP '05*, volume 3709 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, October 2005.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[DN07] V. Dahl and I. Niemelä, editors. *ICLP '07: Proc. 23rd Intl. Conf. Logic Programming*, volume 4670 of *Lecture Notes in Computer Science*. Springer-Verlag, September 2007.

[DSS07] Gregory J. Duck, Peter J. Stuckey, and Martin Sulzmann. Observable Confluence for Constraint Handling Rules. In Dahl and Niemelä [DN07], pages 224–239.

[EK04] Hartmut Ehrig and Barbara König. Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting. In I. Walukiewicz, editor, *FOSSACS '04*, volume 2987 of *Lecture Notes in Computer Science*, pages 151–166. Springer-Verlag, March/April 2004.

[Frü98] Thom Frühwirth. Theory and Practice of Constraint Handling Rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1–3):95–138, 1998.

[Frü09] Thom Frühwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.

[OCL10] Object Management Group. *Object Constraint Language, Specification Version 2.2*, February 2010.

[Ohl02] Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, 2002.

[Rai07] Frank Raiser. Graph Transformation Systems in CHR. In Dahl and Niemelä [DN07], pages 240–254.

[Rai10] Frank Raiser. *Graph Transformation Systems in Constraint Handling Rules: Improved Methods for Program Analysis*. PhD thesis, Ulm University, November 2010.

[RBF09] Frank Raiser, Hariolf Betz, and Thom Frühwirth. Equivalence of CHR States Revisited. In F. Raiser and J. Sneyers, editors, *CHR '09*, pages 34–48. K.U.Leuven, Department of Computer Science, Technical report CW 555, July 2009.

[Sne08] Jon Sneyers. *Optimizing Compilation and Computational Complexity of Constraint Handling Rules*. PhD thesis, K.U.Leuven, Leuven, Belgium, November 2008.

[SSD05] Jon Sneyers, Tom Schrijvers, and Bart Demoen. The Computational Power and Complexity of Constraint Handling Rules. In T. Schrijvers and T. Frühwirth, editors, *CHR '05*, pages 3–17. K.U.Leuven, Department of Computer Science, Technical report CW 421, October 2005.

[SVWSDK10] Jon Sneyers, Peter Van Weert, Tom Schrijvers, and Leslie De Koninck. As Time Goes By: Constraint Handling Rules – A Survey of CHR Research between 1998 and 2007. *Theory and Practice of Logic Programming*, 10(1):1–47, 2010.

Frank Raiser wurde am 23. September 1980 in Füssen geboren und hat dort sein Abitur gemacht. Von 2001 bis 2006 hat er an der Universität Ulm Informatik studiert und anschließend dort mit Auszeichnung promoviert. Neben seiner Forschungstätigkeit hat er sich zudem an der Organisation wissenschaftlicher Tagungen beteiligt und seine Ergebnisse auf der CHR Summer School 2010 präsentiert. Er bringt sich stets für Verbesserungen der Lehre ein und hat 2010 das Baden-Württemberg Zertifikat für Hochschuldidaktik verliehen bekommen. Er ist Autor von über 20 wissenschaftlichen Veröffentlichungen und Mitherausgeber des Buches "Constraint Handling Rules: Compilation, Execution, and Analysis". Derzeit arbeitet er als Software-Entwicklungs-Ingenieur für die Firma Konzept Informationssysteme GmbH.