Hochschule für Technik, Wirtschaft und Kultur Leipzig

Fakultät Informatik, Mathematik und Naturwissenschaften

Masterstudiengang Informatik

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

Erweiterung einer Haskell Web-Applikation um Analyse und Visualisierung mit Formaler Begriffsanalyse

Eingereicht von: **René Muhl**

Matrikelnummer: 61797

Betreuer: Prof. Dr. Johannes Waldmann

Leipzig 4. August 2016

Abstract

Bei der Termination-Competition treten Computer-Programme sogenannte Solver gegeneinander an, um die Termination, das Halten eines Programms in verschiedenen Berechnungsmodellen automatisch zu beweisen. Die Solver lösen in einem Wettstreit eine Vielzahl von Problemen, die Benchmarks.

Die StarExec-Plattform stellt Kapazitäten für die Berechnungen und eine rudimentäre Auflistung der Ergebnisse. Stefan von der Krone und Prof. Waldmann implementierten Star-Exec-Presenter um eine bessere und flexiblere Visualisierungsschicht für die StarExec-Plattform zu haben.

In dieser Arbeit soll Formale Begriffsanalyse genutzt werden um Visualisierung und Analyse in Star-Exec-Presenter anzureichern. Die Formale Begriffsanalyse ist eine Methode des Data-Minings und erlaubt es Resultatdaten der Wettbewerbe zu klassifizieren und besser auswerten zu können.

During the termination competition computer programs battle each other to automatically proove the termination in different models of computation. These computer programs, also known as solvers solve a lot of problems called benchmarks.

The StarExec-Platform provides the computation capacities and implements a primitive listing of the results. To achieve a better and more flexible visualization layer, Star-Exec-Presenter was developed by Stefan van der Krone and Prof. Waldmann.

The aim of this thesis is to enrich the visualization and analysis data from Star-Exec-Presenter by means of formal concept analysis. This technique, which is common in data mining, is used to classify the results of the termination competition in order to analyse them in more depth.

Danksagungen

Größter Dank gilt meiner Frau Elisabeth, die sich mit viel Zeit und Liebe unseren, während der Arbeit geborenen Sohn widmete. Mit einer Selbstverständlichkeit brachte Sie mir Geduld und Unterstützung entgegen.

In der Hoffnung meine bestehenden, sprachübergreifenden Programmierkenntnisse zu erweitern und sich verschiedenste Techniken anzueignen, entschied sich der Autor Prof. Waldmann um die Betreuung meiner Abschlussarbeit zu bitten. Für die Annahme dieser und dem ausgesprochenen Engagement in Zeit und Rat bedankt sich der Autor sehr.

Dadurch gab es nicht nur einen Einblick in die Programmiersprache Haskell, sondern auch eine Verbesserung meiner analytischen Fähigkeiten sowie meines selbstständigen Vorgehens.

Weitere Danksagungen seien an die drei Interview-Partnern Matthias Heizmann, Georg Moser und Marc Brockschmidt gerichtet, die sich als Nutzer von Star-Exec-Presenter Zeit nahmen meinen Fragenkatalog zu beantworten. Auch den Korrekturlesern Andreas Linz, Elisabeth Pabst und Christoph Polcin sei gedankt.

Inhaltsverzeichnis

1 Motivation

Bei der Termination-Competition treten verschiedene Computerprogramme (sogenannte Solver) auf der Plattform StarExec gegeneinander an. In diesem Wettstreit muss eine Menge von Aufgaben (Benchmarks) bezüglich ihrer Termination überprüft werden.

Termination beschreibt das (An-)Halten eines Programmes auf eine Eingabe nach einer endlichen Anzahl von Arbeitsschritten. Das algorithmische Entscheiden der Termination von allen Programmen für alle Eingaben ist nicht möglich und findet seine Beschreibung durch das Halteproblem.

Aufgrund der Entwicklung neuer Techniken zum Beweisen der Termination in den neunziger Jahren wurden mehrere Programme für das vollautomatische Lösen von speziellen Aufgabeklassen entwickelt. Daraus entwickelte sich ein jährliches Event. Ab 2004 wurde diese Competition vier Jahre am *Laboratoire de Recherche en Informatique (LRI)* in Paris und anschließend 2008 bis 2013 an der Universität Innsbruck durchgeführt. Der Wettstreit findet seit 2014 an der HTWK Leipzig statt, Organisator ist Prof. Waldmann. [Wal15a] Seitdem Stattfinden an der HTWK Leipzig dient Star-Exec-Presenter[1] als Präsentationschicht von StarExec. Diese Web-Applikation wurde durch Stefan von der Krone in Zusammenarbeit mit Prof. Waldmann entwickelt.

Die Termination eines Benchmarks kann von jedem Solver mit YES, NO oder MAYBE beantwortet werden. Die Antwort YES gibt an, dass der Solver einen Beweis für eine und NO - einen Beweis für keine Termination gefunden hat. Die Antwort MAYBE drückt aus, dass kein Beweis gefunden wurde. Zusätzlich zu dieser Antwort werden noch weitere Daten wie die Zeit, wie lange für dieses Ergebnis benötigt wurde, erfasst [Wal15b] [MZ07].

Aufgrund der Vielzahl an Daten können nur wenige Informationen aus der Resultatsdarstellung der internationalen Termination-Competition 2015 gewonnen werden. In der größten Kategorie *TRS Standard* treten sieben Solver in 1498 Benchmarks gegeneinander an und bilden im Produkt 10486 Resultate[2].

Ziel dieser Arbeit ist die Verbesserung der Visualisierung und Analyse dieser Resultat-Daten. Zur Erreichung des Ziels soll Formale Begriffsanalyse genutzt werden, um die Menge der Resultat-Daten in Teilmengen zu zerlegen. Die Formale Begriffsanalyse agiert innerhalb eines formalen Kontextes. Dieser Kontext besteht aus einer Menge von Gegenständen, einer Menge von Merkmalen und der Beziehung zwischen beiden. In einem Formalen Kontext werden Formale Begriffe bestimmt. Ein Formaler Begriff besteht aus einer Menge von Gegenständen und einer Menge von Merkmalen. Zwei Gegenstände gehören genau dann zum gleichen Begriff, wenn diese exakt die gleichen Merkmale teilen.

[1] https://github.com/jwaldmann/star-exec-presenter
[2] http://nfa.imn.htwk-leipzig.de/termcomp-2015/results/standard/noquery/10257

Beispiel 1.0.1 Formale Begriffsanalyse

Formaler Kontext $(\{1,2,3\},\{blau,rot\},\{(1,rot),(2,blau),(3,rot)\})$
Formale Begriffe $\{(\emptyset,\{blau,rot\}),(\{1,3\},\{rot\}),(\{2\},\{blau\}),(\{1,2,3\},\emptyset)\}$

Diese Mengen sollen die Analyse zwischen verschiedenen Solvern sowie den Vergleich der Ergebnisse vergangener Competitions vereinfachen. Aufgrund der Vielzahl von Daten ist es notwendig die Ergebnisse neu zu strukturieren. Durch eine geeignete Navigation innerhalb dieser, soll es möglich sein die vorliegenden Informationen besser zu untersuchen.

Die Stärken der Formalen Begriffsanalyse sind unter anderem das „Entfalten der Daten“, indem die Struktur der Daten sichtbar und zugänglich gemacht wird [GW, S.6] [Pet08, S.49]. So können Zusammenhänge, Muster, Regelmäßigkeiten und Ausnahmen gefunden werden. [GW, S.6]

Als Vorbereitung auf diese Arbeit fand bereits ein Masterprojekt [Muh15] statt. In dieser Vorarbeit wurde die bestehenden Applikation, die Programmiersprache Haskell und das Web-Framework Yesod untersucht. Im Rahmen des Masterprojektes wurden die Grundlagen der Formalen Begriffsanalyse recherchiert und rudimentär implementiert.

Nach der Vorstellung der Termination-Competition einschließlich *StarExec*, *Termination Problems Data Base* und Star-Exec-Presenter folgt eine Einführung in die Formale Begriffsanalyse. Nachdem der thematische und theoretische Rahmen vorgestellt wurde, werden die Anforderungen an die Arbeit formuliert. In Kapitel 5 wird auf die funktionale Programmiersprache Haskell eingegangen um in die Hauptkapitel Implementierung und Auswertung überzuleiten. Abschließende Worte sind in der Zusammenfassung zu finden.

Anmerkung: In der Arbeit werden die offiziellen StarExec-Bezeichnungen wie Competition, Job, JobPair verwendet. Um Verwirrung zu vermeiden werden keine neuen Bezeichnungen eingeführt.

2 Termination-Competition

Die Termination-Competition ist ein Wettbewerb zur automatischen Analyse der Termination von Problemen der *Termination Problems Data Base*. Solver untersuchen diese Probleme sogenannte Benchmarks und versuchen deren Termination zu beweisen. Seit dem Jahre 2014 findet diese Competition auf der Plattform StarExec statt. Die Ergebnisse werden auf Star-Exec-Presenter zusammengefasst und präsentiert. [Wal15a] Die Competition-Resultate bilden die Daten-Grundlage der Formalen Begriffsanalyse. Alle genannten Bestandteile der Termination-Competition werden in diesem Kapitel erläutert.

2.1 Termination Problems Data Base

Während der Termination-Comeptition werden alle Benchmarks der Termination Problems Data Base (abgekürzt TPDB) auf StarExec bearbeitet. Es existiert ein Mercurial[1]-Respository[2] zur Verwaltung der Benchmark-Dateien. Eine Benchmark-Datei beschreibt ein Problem. Je nach Problem-Kategorie ist das Dateiformat verschieden. Der Benchmark „TRS_Relative/Relative_05/rt3-1.xml“[3] gehört zur Kategorie der Termersetzungssysteme bzw. TRS. Eine XML-Datei beschreibt in dieser Kategorie über die Signatur $\sum = \{(a/0), (f/3), (g/1)\}$ das folgende Termersetzungssystem R:

- $f(g(x), y, z) \to f(x, y, g(z))$
- $f(x, y, g(z)) \to f(x, g(y), z)$
- $f(x, a(), z) \to f(x, g(a()), z)$
- $f(x, y, z) \to f(x, y, g(z))$

Das aktuelle Release ist 10.3 (Revision 4d76dd84fd49, Stand 11.7.2016). In TPDB 10.3 befindet sich eine Sammlung von 16006[4] Dateien. Der Aufbau der TPDB ist ähnlich der Competition-Kategorien [Gie+15]. Die Tabelle 2.1 gibt eine Übersicht über Verteilung der Kategorien in der TPDB.

[1] https://www.mercurial-scm.org/
[2] http://cl2-informatik.uibk.ac.at/mercurial.cgi/TPDB/summary
[3] http://cl2-informatik.uibk.ac.at/mercurial.cgi/TPDB/file/d085ae59ef47/TRS_Relative/Relative_05/rt3-1.xml
[4] find TPDB-4d76dd84fd49 | wc -l

Kategorie	Anzahl der Dateien	Dateinnamenerweiterung
Termersetzungssysteme	10811	xml
Haskell-Programme	1671	hs
Integer-Transitionssysteme	1222	SMT2
C-Programme	803	c
Prolog-Programme	633	pl
Java-Programme	443	jar
Integer Termersetzungssysteme	117	itrs

Tabelle 2.1: Übersicht über den Inhalt der TPDB

Anmerkung: Die Anzahlen wurden mit dem Befehl `find TPDB-4d76dd84fd49 -name "*.<extension>" | wc -l` ermittelt, wobei <extension> mit der zugehörigen Dateinnamenerweiterung ersetzt wurde.

2.2 Solver

Die an einer Competition teilnehmenden Solver müssen eine Menge von Benchmarks lösen. Ein Algorithmus [Rub14] wählt eine Untermenge an Benchmarks aus den existierenden Benchmarks der aktuellen Kategorie. Die ausgewählte Menge aus Solvern und Benchmarks wird als Job bezeichnet. Bei den Termination-Competitions laufende Jobs sind Competitions.

2.3 Plattformlatform StarExec

StarExec[5] ist ein Community-übergreifender Service der University of Iowa und der University of Miami. Die Ziele des Services sind unter anderem die experimentelle Auswertung von Logik-Solvern und das Ausführen von Solver-Competitions. Dabei dient StarExec als Plattform und stellt Speicher, Recheninfrastruktur und Bibliotheken zur Verfügung. [SST15]

2.4 Star-Exec-Presenter

Star-Exec-Presenter ist im Rahmen der Masterarbeit von Stefan von der Krone entstanden und wurde von Prof. Waldmann weiterentwickelt. Die Applikation ist in Haskell (Kapitel 5)/Yesod (Kapitel 5.2) implementiert. Die Features von Star-Exec-Presenter sind vielfältig, das wichtigste Feature ist die Visualisierung der Ergebnisse der Termination-Competition auf StarExec.

Die Daten der Competiton werden auf der Plattformlatform StarExec berechnet und falls auf Star-Exec-Presenter benötigt, angefordert und in einer lokalen Datenbank abgelegt. Aus der Datenbank werden die verschiedenen Daten geladen und auf entsprechenden Seiten dargestellt. Solver werden

[5] https://www.starexec.org/

als Spalten und die Benchmarks der TPDB (Kapitel 2.1) als Zeilen in der Tabelle aus Abbildung 2.1 dargestellt. Die Zellen/JobPairIDs als Kreuzungspunkte beschreiben die Resultat-Daten.

Overview of job-results

flexible query (experimental) | view original jobs on star-exec:5375,

Statistics

88 pairs, 13062.8 / 4027.5 s

Results

Show Legend

Jobs	TRS Relat 27688	
Solver	TTT2	AProVE_JRE2
Scores	24	35
TRS_Relative/Mixed_relative_TRS/abp.xml	235.7 / 59.7 s	957.9 / 300.1 s
TRS_Relative/Mixed_relative_TRS/abp2.xml	235.3 / 59.4 s	930.1 / 300.1 s
TRS_Relative/Mixed_relative_TRS/assoc.xml	220.3 / 59.3 s	2.2 / 0.9 s
TRS_Relative/Mixed_relative_TRS/carbridge.xml	20.6 / 5.5 s	4.4 / 1.5 s
TRS_Relative/Mixed_relative_TRS/gcd.xml	235.3 / 59.4 s	554.8 / 300.0 s
TRS_Relative/Mixed_relative_TRS/gcd_list.xml	235.3 / 59.4 s	1027.8 / 300.1 s
TRS_Relative/Mixed_relative_TRS/gcd_many.xml	1.4 / 0.6 s	1.5 / 0.7 s
TRS_Relative/Mixed_relative_TRS/ijcar2006.xml	3.3 / 1.1 s	4.3 / 1.5 s
TRS_Relative/Mixed_relative_TRS/relsubst.xml	235.6 / 59.4 s	555.3 / 300.0 s
TRS_Relative/Mixed_relative_TRS/rt-rw4.xml	235.2 / 59.4 s	16.0 / 4.7 s

Abbildung 2.1: Star-Exec-Presenter: Resultat-Tabelle bzgl. der Job-ID 5375

http://termcomp.imn.htwk-leipzig.de/results/standard/noquery/5375

2.5 Resultat-Daten

Jeder Solver untersucht die Benchmarks auf ihre Termination. Die Kombination aus Solver und Benchmark wird als JobPair oder kurz Pair bezeichnet. Das Resultat eines JobPairs wird in Tabelle 2.2 dargestellt.

Information	Beispiel	Erläuterung
JobId	5375	Id des Jobs
Score	Nothing	Punktzahl
PairId	26929910	Id des JobPairs
Benchmark	rt3-1.xml	Pfad der Benchmarkdatei
BenchmarkId	1123229	Id des Benchmarks
Solver	AProVE_JRE2	Name des Solvers
SolverId	1681	Id des Solvers
Configuration	trs	Konfiguration des Solvers
ConfigurationId	2656	Id der Konfiguration
Status	complete	Abarbeitungsstatus
CpuTime	3.46947	Zeit durch Nutzung mehrerer Kerne
WallclockTime	1.24966	real vergangene Zeit
Result	YES	Antwort bzgl. Termination

Tabelle 2.2: Resultat-Daten des JobPairs 26929910

Anmerkungen zur Tabelle 2.2:

- Auf Grund der Übersichtlichkeit wurde der Name des Benchmarks verkürzt. Der vollständige Name lautet: „TRS_Relative/Relative_05/rt3-1.xml“
- Ein Postprozessor prüft das Resultat des Solvers. Zur Vereinfachung wird sich auf die Antworten des Solvers beschränkt. Antworten des Postprozessors sind: CERTIFIED, ERROR, OTHER.

3 Formale Begriffsanalyse

Die Formale Begriffsanalyse ist eine Methode des Data-Minings und dient der Klassifikation von JobPairs in den Resultat-Daten auf Star-Exec-Presenter. Es werden mathematische Grundlagen aus der Ordnungs- und Verbandstheorie und darauf aufbauend benötigte Aspekte der Formalen Begriffsanalyse erläutert.

3.1 Allgemein

Die Formale Begriffsanalyse (engl. formal concept analysis/FCA) seltener Formale Konzeptanalyse genannt, stellt eine Anwendung der Verbandstheorie dar. Die Theorie der FCA wurde Anfang der 1980er Jahre von Rudolf Wille im Fachbereich Mathematik der Technischen Hochschule Darmstadt definiert [GW, S.1] [GW96, S.58].

Um sich von dem bereits vergebenen Begriff „Begriff“ abzuheben, wird mit der zusätzlichen Eigenschaft „formal“ die Absicht einer mathematischen Beschreibung verdeutlicht [GW96, S.17]. In der vorliegenden Auseinandersetzung wird der Ausdruck Begriffsanalyse dem der Konzeptanalyse vorgezogen, da in der deutschen Sprache ein Begriff für eine Sache eine Menge von Merkmalen besser beschreibt als der Begriff „Konzept“.

Anmerkung: Zur Theorie werden Modellierungen in Haskell[1] angegeben. Eine ausführliche Erläuterung der Sprache und deren Sprach-Features werden in Kapitel 5 vorgenommen.

Um die Formale Begriffsanalyse vorzustellen sind Grundlagen aus der Mengenlehre sowie der Ordnungstheorie notwendig. Mathematik ist die Grundlage der Informatik und die Ordnungstheorie als Teilgebiet der Mathematik die Basis für die Formale Begriffsanalyse.

Es wird die Kenntnis grundlegender mengentheoretischer Relationen und Operatoren vorausgesetzt. In der Masterarbeit werden die folgenden verwendet/bezeichnet:

- Relationen: Teilmenge $\subseteq$, Element $\in$
- Operationen: Kartesisches Produkt $\times$, Vereinigungsmenge $\cup$, Differenzmenge $\setminus$, Potenzmenge $\mathcal{P}(M)$, Mächtigkeit $|M|$

[1] https://www.haskell.org/

3.2 Ordnungs- und Verbandstheorie

Definition 3.2.1. *(M, R) mit $R \subseteq M^2$ ist eine **Halbordnung** oder **partiell geordnete Menge**, wenn folgende Eigenschaften erfüllt sind:*

- *Reflexivität:* $\forall a \in M : (a, a) \in R$
- *Transitivität:* $\forall a, b, c \in M : (a, b) \in R \wedge (b, c) \in R \rightarrow (a, c) \in R$
- *Antisymmetrie:* $\forall a, b \in M : (a, b) \in R \wedge (b, a) \in R \rightarrow a = b$

Beispiel 3.2.1 Halbordnung

$(\mathbb{N}, \leq), (\mathbb{N}, |), (\mathcal{P}(M), \subseteq)$ [Hac08, S.103]

Definition 3.2.2. *[Hac08, S.237] Ist in einer Halbordnung $(M, \preceq)$ die Menge $N \subseteq M$ nicht leer,*

- *so heißt $x \in M$ eine **untere Schranke** von N, falls $\forall n \in N : x \preceq n$.*
- *so heißt $y \in M$ eine **obere Schranke** von N, falls $\forall n \in N : n \preceq y$.*

Beispiel 3.2.2 untere/obere Schranken

Halbordnung $(\mathbb{N}, \leq)$ mit $N = \{1, 2, 3\} \subseteq \mathbb{N}$
untere Schranke von N: 0 obere Schranken von N: 3,4,7

Halbordnung $(\mathbb{N}, |)$ mit $N = \{6, 12, 18\} \subseteq \mathbb{N}$
untere Schranke von N: 1 obere Schranken von N: 72,144

Halbordnung $(\mathcal{P}(\{a, b, c\}), \subseteq)$ mit $N = \{\emptyset, \{a\}, \{c\}\} \subseteq \mathcal{P}(\{a, b, c\})$
untere Schranke von N: $\emptyset$ obere Schranke von N: $\{a, b, c\}$

Definition 3.2.3. *[Hac08, S.237] In einer Halbordnung $(M, \preceq)$ mit $N \subseteq M$ sei $x \in M$ eine untere Schranke von N und $y \in M$ eine obere Schranke von N.*

- *x ist ein **Infimum** oder eine **größte untere Schranke** von N, falls für alle weiteren unteren Schranken r von N $r \preceq x$ gilt.*
- *y ist ein **Supremum** oder eine **kleinste obere Schranke** von N, falls für alle weiteren oberen Schranken r von N $y \preceq r$ gilt.*

Aus der Antisymmetrie von $\preceq$ ist das Infimum und Supremum (sofern sie existieren) eindeutig bestimmt.

Notation:

- *x ist genau dann Infimum von N, wenn $x = \inf(N)$*
- *y ist genau dann Supremum von N, wenn $y = \sup(N)$*

Beispiel 3.2.3 Infimum & Supremum

Halbordnung $(\mathbb{N}, \leq)$ mit $N = \{1, 2, 3\} \subseteq \mathbb{N}$
$\inf(N) = 1$, $\sup(N) = 3$

Halbordnung $(\mathbb{N}, |)$ mit $N = \{6, 12, 18\} \subseteq \mathbb{N}$
$\inf(N) = 6$, $\sup(N) = 36$

Halbordnung $(\mathcal{P}(\{a, b, c\}), \subseteq)$ mit $N = \{\emptyset, \{a\}, \{c\}\} \subseteq \mathcal{P}(\{a, b, c\})$
$\inf(N) = \emptyset$, $\sup(N) = \{a, c\}$

Definition 3.2.4. *[Gan13, S.65] [Hac08, S.237] Eine Halbordnung $(M, \leq)$ ist genau dann ein **Verband**, wenn $\forall a, b \in M$:*

- $\inf(\{a, b\})$
- $\sup(\{a, b\})$

existieren.

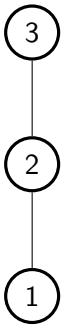
Beispiel 3.2.4 Verband

Die Halbordnungen $(\mathbb{N}, \leq), (\mathbb{N}, |), (\mathcal{P}(M), \subseteq)$ sind Verbände, da sie ein Infimum und Supremum (in Beispiel 3.2.3 gezeigt) besitzen.

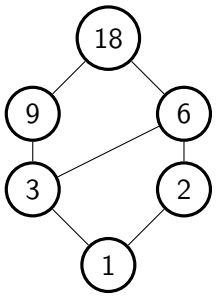
Definition 3.2.5. *Jede endliche Halbordnung $(M, \preceq)$ lässt sich als **Hasse[2]-Diagramm** (gelegentlich auch Liniendiagramm) darstellen. Die Elemente von M werden als Kreise dargestellt. Wenn $x, y \in M$ mit $x \prec y$, dann ist der Kreis für y im Diagramm oberhalb des Kreises von x. Beide Kreise werden durch eine Linie miteinander verbunden. Um eine Zuordnung zwischen Element und Kreis zu erhalten, wird das Element in den Kreis geschrieben. Die Ordnungsrelation lässt sich wie folgt aus dem Diagramm ablesen: $x \prec y$, wenn es der Kreis für x durch eine aufsteigende Linie den Kreis für y erreichbar ist. [GW96, S.2] Linien die sich aus der Reflexivität und Transitivität der Relation $\preceq$ über die Menge M ergeben werden ausgespart [IL14, S.48]. Im folgenden werden Kreise "Knoten" und Linien "Kanten" genannt.*

[2]Helmut Hasse (1898 - 1976): Professor für Mathematik in Berlin und Hamburg

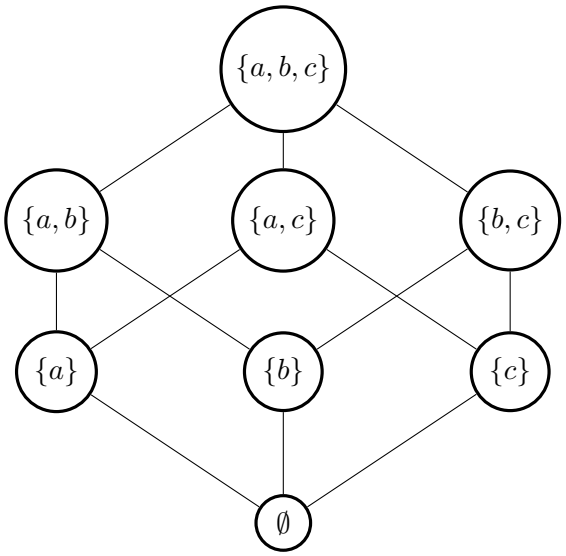
Beispiel 3.2.5 Hasse-Diagramm



Halbordnung $(\{1, 2, 3\}, \leq)$



Halbordnung $(\{n \in \mathbb{N} : n|18\}, |)$



Halbordnung $(\mathcal{P}(\{a, b, c\}), \subseteq)$

3.3 Formaler Kontext

Zur Veranschaulichung dient die Resultat-Tabelle 3.1 der JobID 9515. Die Gegenstände sind die Zellen der Tabelle, die sogenannten JobPairIDs. Als Merkmale dienen die Bestandteile: *Solver*, *Result* und *CpuTime* des JobPair-Datensatzes (Kapitel 2.5). Es werden Modellierungen in Haskell angegeben. Eine Beschreibung der Sprache Haskell erfolgt in Kapitel 5.

Definition 3.3.1. *Das Tripel (G, M, R) wird **Formaler Kontext** genannt, wenn:*

- *G die Menge der Gegenstände*
- *M die Menge der Merkmale*
- *R die Relation $R \subseteq G \times M$*

Ein Kontext wird oft mit einer Kreuztabelle beschrieben. Dabei befinden sich die Gegenstände in einer Zeile und die Merkmale in einer Spalte. Ein boolescher Wert in Form eines ✘ markiert, ob ein Gegenstand ein Merkmal besitzt. Ist kein ✘ vorhanden, besitzt der Gegenstand den Wert nicht.

```
1 data Context ob at = Context
2   { fore :: Map ob (Set at)
3   , back :: Map at (Set ob)
4   } deriving (Show)
```

Quelltext 3.1: FCA/Basic.hs-L26

G | M	TTT2	AProVE 2015	YES	MAYBE	CPU time >10s
118913107	✘			✘	✘
118913108		✘		✘	✘
118913109	✘		✘		✘
118913110		✘	✘		✘
118913111	✘			✘	✘
118913112		✘	✘		✘
118913113	✘			✘	✘
118913114		✘		✘	✘

Tabelle 3.1: Kreuztabelle zur JobID 9515

Beispiel 3.3.1 Formaler Kontext

Die Gegenstände $G = \{118913107, 118913108, \ldots, 118913113, 118913114\}$ mit den Merkmalen $M = \{$TTT2, AProVE 2015, YES, MAYBE, CPU time >10s$\}$ in der Relation $R = \{(118913107, \text{TTT2}), ..., (118913114, \text{ CPU time >10s})\}$ ergeben den Formalen Kontext (G, M, R) in Tabelle 3.1.

3.4 Projektionen

Die Menge der gemeinsamen Merkmale der Gegenstände aus $A \subseteq G$ sei:

- $A' := \{m \in M \mid \forall g \in A : (g, m) \in R\}$

```haskell
-- get all attributes of given context and specific objects
getAttributes :: (Ord ob, Ord at) => Context ob at -> Set ob -> Set at
getAttributes c obs = foldr Set.intersection (attributes c)
  $ map (\o -> fore c Map.! o) $ Set.toList obs
```

Quelltext 3.2: FCA/Basic.hs-L66

Beispiel 3.4.1 Projektion A'

$A = \{118913109\} \subseteq G$
$A' = \{118913109\}' = \{$TTT2, YES, CPU time $>$10s$\}$
Die Merkmale der JobID 118913109 sind TTT2, YES und CPU time $>$10s.

Entsprechend die Menge der Gegenstände, die alle Merkmale aus $B \subseteq M$ besitzen:

- $B' := \{g \in G \mid \forall m \in B : (g, m) \in R\}$

```haskell
-- get all objects of given context and specific attributes
getObjects :: (Ord ob, Ord at) => Context ob at -> Set at -> Set ob
getObjects c ats = foldr Set.intersection (objects c)
  $ map (\a -> back c Map.! a) $ Set.toList ats
```

Quelltext 3.3: FCA/Basic.hs-L71

Beispiel 3.4.2 Projektion B'

$B = \{$TTT2, YES, CPU time $>$10s$\} \subseteq M$
$B' = \{$TTT2, YES, CPU time $>$10s$\}' = \{118913109\}$
Der Gegenstand (in diesem Fall nur ein Einzelner), der alle
Merkmale TTT2, YES, CPU time $>$10s beseitzt, ist die JobID 118913109.

3.5 Formaler Begriff

Definition 3.5.1. *Ein Paar (A, B) ist ein **formaler Begriff** des Formalen Kontextes (G, M, R) mit [GW96, S.18]:*

- $A \subseteq G$ *(Begriffsumfang/extent)*
- $B \subseteq M$ *(Begriffsinhalt/intent)*
- $A' = B$ *(siehe Projektionen)*
- $B' = A$ *(siehe Projektionen)*

```haskell
data Concept ob at = Concept
  { obs :: Set ob
  , ats :: Set at
  } deriving (Show, Eq)
```

Quelltext 3.4: FCA/Basic.hs-L31

G | M	TTT2	AProVE 2015	YES	MAYBE	CPU time >10s
118913107	✘			✘	✘
118913108		✘		✘	✘
118913109	✘		✘		✘
118913110		✘	✘		✘
118913111	✘			✘	✘
118913112		✘	✘		✘
118913113	✘			✘	✘
118913114		✘		✘	✘

Tabelle 3.2: Beispiele für Formale Begriffe der JobID 9515

Beispiel 3.5.1 Formaler Begriff

Der Formale Kontext des Jobs 9515 beinhaltet 10 Formale Begriffe (Beispiel 3.5.5 zeigt alle). Die Markierungen in Tabelle 3.2 zeigen beispielhaft zwei der zehn Formalen Begriffe: $(\{118913109\}, \{\text{TTT2,YES,CPU time } >10\text{s}\})$, $(\{118913108, 118913114\}, \{\text{AProVE 2015,MAYBE,CPU time } >10\text{s}\})$.

3.5.1 Algorithmus zur Begriffsbestimmung

Innerhalb eines Formalen Kontext (G, M, R) lässt sich ein Formaler Begriff durch die Ausführung einer der beiden Schrittfolgen (jeweils eine Spalte) [Sch11] erreichen:

1. $A \subseteq G$ wählen
2. A' und A'' bestimmen
3. wenn $A'' = A$, neuer Begriff (A, A')

1. $B \subseteq M$ wählen
2. B' und B'' bestimmen
3. wenn $B'' = B$, neuer Begriff (B', B)

Die folgende Modellierung in Haskell beschreibt einen Algorithmus zur Bestimmung aller Formalen Begriffe innerhalb eines Formalen Kontextes. Die Umsetzung erfüllt die Spezifikation und wird in Kapitel 6.3.2 durch eine effizientere Variante ersetzt.

```haskell
-- determine all concepts of given context
concepts :: (Ord ob, Ord at) => Context ob at -> [(Set ob, Set at)]
concepts c = do
  ats <- map (\ats -> Set.fromList ats) $ subsequences $ Set.toList $
      attributes c
  guard $ ats == (getAttributes c $ getObjects c ats)
  return (getObjects c ats, ats)
```

Quelltext 3.5: FCA/Basic.hs-L53

Beispiel 3.5.2 erfolgreiches Bestimmen eines Begriffes

$B = \{\text{TTT2, YES, CPU time } >\text{10s}\} \subseteq M$
$B' = \{\text{TTT2, YES, CPU time } >\text{10s}\}' = \{118913109\}$
$B'' = \{118913109\}' = \{\text{TTT2, YES, CPU time } >\text{10s}\}$

Da die Gleichung $B = \{\text{TTT2, YES, CPU time } >\text{10s}\} = \{\text{TTT2, YES, CPU time } >\text{10s}\} = B''$ erfüllt ist, entsteht der neue Formale Begriff
$(B', B) = (\{118913109\}, \{\text{TTT2, YES, CPU time } >\text{10s}\})$.

Beispiel 3.5.3 nicht erfolgreiches Bestimmen eines Begriffes

$A = \{118913110\} \subseteq G$
$A' = \{118913110\}' = \{$AProVE 2015, YES, CPU time >10s$\}$
$A'' = \{$AProVE 2015, YES, CPU time >10s$\}' = \{118913110,118913112\}$

Da die Gleichung $A'' = \{118913110, 118913112\} = \{118913110\} = A$ nicht erfüllt ist, folgt kein Formaler Begriff.

Anmerkung: Bei der Wahl der Schrittfolgen zur Begriffsbestimmung sollte darauf geachtet werden, das die Menge mit der kleinsten Mächtigkeit (Gegenstände G oder Merkmale M) verwendet wird. Zur Berechnung aller Teilmengen der gewählten Ausgangsmenge dient die Potenzmenge dieser Menge. Die Potenzmenge einer Ausgangsmenge $C \subseteq M$ oder $C \subseteq G$ besitzt eine Mächtigkeit von $2^{|C|}$.

3.5.2 Hierachie zwischen Begriffen

Seien in einem Formalen Kontext (G, M, R) zwei Formale Begriffe $(A_1 \subseteq G, B_1 \subseteq M), (A_2 \subseteq G, B_2 \subseteq M)$ heißt der Begriff (A_1, B_1) **Unterbegriff** von (A_2, B_2) falls $A_1 \subseteq A_2$ oder $B_2 \subseteq B_1$. (A_2, B_2) heißt dann **Oberbegriff** von (A_1, B_1) und man schreibt $(A_1, B_1) \leq (A_2, B_2)$ [GW96, S.20].

Beispiel 3.5.4 Begriffshierachie

Gegeben seien die Formalen Begriffe:

- $f1 = (A_1, B_1) = (\{118913109\}, \{$TTT2,YES,CPU time >10s$\})$
- $f2 = (A_2, B_2) = (\{118913109, 118913110, 118913112\}, \{$YES,CPU time >10s$\})$

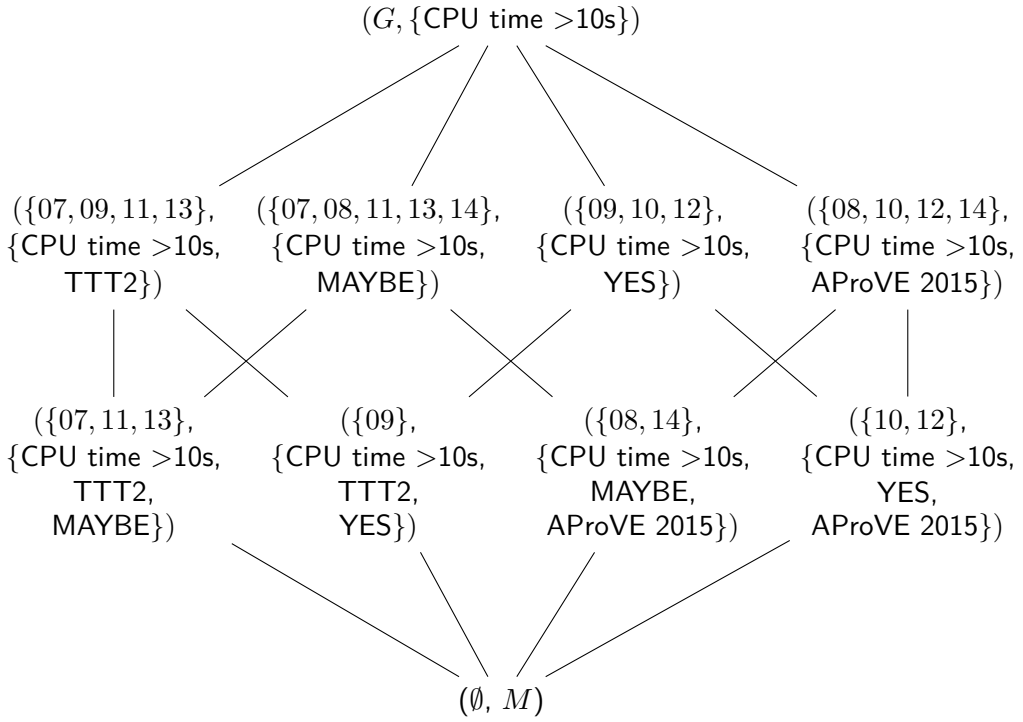
$f1$ ist Unterbegriff von $f2$, da $A_1 = \{118913109\} \subseteq \{118913109, 118913110, 118913112\} = A_2$ bzw $B_2 = \{$YES,CPU time >10s$\} \subseteq \{$TTT2,YES,CPU time >10s$\} = B_1$ gilt.

3.5.3 Begriffsverband

Satz 3.5.1. *Alle Formalen Begriffe des Formalen Kontextes (G,M,R) bilden mit der Relation $\leq$ (Kapitel 3.5.2) einen Verband [GW96, S.20].*

Beispiel 3.5.5 Begriffsverband

Aus Gründen der Übersichtlichkeit wurden in den Knoten des Hasse-Diagramms die Gegenstände auf die letzten zwei Nummern verkürzt (aus 118913109 wird 09).



Begriffsverband des Kontextes zum Job 9515

4 Anforderungen an die Arbeit

Es werden die Resultate der geführten Interviews ausgewertet und bestehende Möglichkeiten für die Nutzer auf Star-Exec-Presenter untersucht. Zusammenfassend wird eine Spezifikation an die Arbeit formuliert.

4.1 Interview

Es ist die Aufgabe mit Formaler Begriffsanalyse Star-Exec-Presenter zu verbessern. Um herauszufinden an welchen Stellen diese Methode eine Verbesserung erzielen kann, wird mit den Nutzern ein Interview geführt.

4.1.1 Fragen

Das Interview deckt drei Aspekte ab um die Nutzer und deren Wünsche einordnen zu können. Im Anhang 1 sind alle Fragen an die drei freiwilligen Interview-Partner aufgelistet. Hier wird nur eine Auswahl vorgestellt:

1. Verhalten auf der Webseite

a. Für was nutzen Sie Star-Exec-Presenter?
b. Was ist die von Ihnen am meisten genutzte Route? (Angabe von Bsp.-URL hilfreich)

2. benötigte Verbesserungen

a. Welche Darstellungsformen, würden bei der Visualisierung, Analyse, Navigation zwischen den Seiten, den Analyse-Daten helfen?
b. Welche Features/Funktionen wären hilfreich, um konkurrierende Solver besser zu analysieren?
c. Welche Informationen werden benötigt und sind derzeit nicht in Star-Exec-Presenter eingebaut?

3. Meinung zu eigenen Verbesserungsvorschlägen

a. Sollte es eine Navigationsleiste/Menü geben?
b. Sollte es einen Filter für die Job-Tabelle geben?
c. Würde Pagination auf den Seiten helfen? Wieviele Zeilen sollte pro Seite angezeigt werden? (10, 20, 50)
d. Sollte es mehr Informationen zu einem Benchmark geben?
e. Sollte es einen Vergleich von Benchmarks/Solvern (+ Config)/Termination-Competition geben?

4.1.2 Antworten

Die Antworten der einzelnen Nutzer werden zusammengefasst dargestellt. In Kapitel 4.3 wird geprüft, welche der Features mit Formaler Begriffsanalyse oder innerhalb des Rahmens der Masterarbeit umgesetzt werden können. Im Anhang 2 befindet sich eine ausführlichere Zusammenfassung der Antworten der Interview-Teilnehmer.

1. Verhalten auf der Webseite

a. Betrachtung Competition-Ergebnisse, Analyse der einzelnen Competition-Kategorien, Beobachtung der Tool-Entwicklung, Anzahl gelöste Benchmarks
b. Competition-Übersicht, Job-Resultate, *Flexible Table*

2. benötigte Verbesserungen

a. Übersichtstabellen mit Durchschnittszeiten, Plots
b. verbesserte Übersichtlichkeit
c. mehr Informationen über die Benchmarks (Inhalt, Größe, Anzahl Zeilen)

3. Meinung zu eigenen Verbesserungsvorschlägen

a. zwei Zustimmungen, eine Enthaltung
b. drei Zustimmungen
c. drei Ablehnungen
d. zwei Zustimmungen, eine Enthaltung
e. einmal kein Vergleich von Benchmarks, eine Enthaltung und eine generelle Zustimmung

4.2 Bestandsaufnahme

Es werden die bestehenden Routen mit ihren Möglichkeiten und Problemen vorgestellt. Die Gründe dafür sind das Verstehen der Nutzer-Wünsche und -Probleme. Außerdem soll es keine Implementierung zu bereits gelösten Problemen geben. Die gesamte Bestandsaufnahme bezieht sich auf die Star-Exec-Presenter-Version mit dem Commit cc84625[1] vom 7. August 2015.

[1] https://github.com/jwaldmann/star-exec-presenter/commit/cc846252ac57899c163bf6e0270e334129047013

4.2.1 Competition-Übersicht

Beschreibung

Die Competition-Übersicht zeigt die Competition-Resultate bezüglich eines Jahres.

Termination Competition 2015

General Information wc = 300 a = 1 b = 1 c = 0.1 (2015-08-05 18:48:18.60928 UTC) 51787 pairs, 12024641.5 / 6232857.3 s finished in 399686h 5m 50s

Termination of Term Rewriting (and Transition Systems) finished in 399686h 5m 50s, 33684 pairs, 6726808.6 / 3493948.5 s

Combined Ranking (Rules): 1. AProVE 2015 (20) 2. TTT2 (10) NaTT 1.3 (10) 4. matchbox2015-07-26.1 (7) 5. muterm 5.17 (6) 6. AProVE certified (3) 7. T2 - 2015-07-09 - 13745bd6 (2) Wanda (2) 9. AProVE certified TRS Standard (1) 10. cycsrs-29-07-2015.5 (0) AutoNon 1.21 (0) Ctrl (0)

category	post-proc	rankings	statistics
TRS Standard	plain.3	AProVE 2015 (1310), NaTT 1.3 (1023), TTT2 (989), muterm 5.17 (834), Wanda (636), matchbox2015-07-26.1 (524), AutoNon 1.21 (228),	10486 pairs, 1666958.3 / 824051.3 s
SRS Standard	plain.3	AProVE 2015 (832), TTT2 (598), matchbox2015-07-26.1 (365), NaTT 1.3 (202), muterm 5.17 (135), AutoNon 1.21 (58),	7890 pairs, 2570235.3 / 1324912.6 s
Cycles	plain.3	matchbox2015-07-26.1 (646), cycsrs-29-07-2015.5 (422),	2630 pairs, 950125.9 / 453572.0 s
TRS Relative	plain.3	NaTT 1.3 (70), AProVE 2015 (55), TTT2 (41), matchbox2015-07-26.1 (40),	392 pairs, 77146.9 / 34711.5 s
SRS Relative	plain.3	AProVE 2015 (88), matchbox2015-07-26.1 (32), TTT2 (24), NaTT 1.3 (17),	820 pairs, 274225.6 / 145649.5 s
TRS Standard certified	ceta-2.20-2	AProVE certified TRS Standard (1223), TTT2 (962),	2996 pairs, 251319.4 / 124453.7 s
SRS Standard certified	ceta-2.20-2	AProVE certified (816), TTT2 (570),	2630 pairs, 486311.2 / 223843.9 s
TRS Relative certified	ceta-2.20-2	AProVE certified (51), TTT2 (41),	196 pairs, 29112.5 / 17898.6 s
SRS Relative certified	ceta-2.20-2	AProVE certified (88), TTT2 (20),	410 pairs, 76372.8 / 46591.8 s
TRS Equational	plain.3	AProVE 2015 (67), muterm 5.17 (63),	152 pairs, 3067.0 / 3466.0 s
TRS Conditional	plain.3	muterm 5.17 (101), AProVE 2015 (85),	234 pairs, 5576.4 / 5099.8 s
TRS Context Sensitive	plain.3	muterm 5.17 (98), AProVE 2015 (97),	216 pairs, 7100.2 / 5007.1 s
TRS Innermost	plain.3	AProVE 2015 (273), muterm 5.17 (203),	732 pairs, 102735.3 / 77628.8 s
Integer Transition Systems	plain.3	T2 - 2015-07-09 - 13745bd6 (1061), AProVE 2015 (1034), Ctrl (423),	3666 pairs, 212567.8 / 200512.7 s
Integer TRS	plain.3	AProVE 2015 (102), Ctrl (85),	234 pairs, 13954.0 / 6549.1 s

Abbildung 4.1: Star-Exec-Presenter: Competition-Übersicht 2015 (Begrenzung auf Metakategorie *Termination of Term Rewriting (and Transition Systems)*)
http://nfa.imn.htwk-leipzig.de/termcomp-2015/competitions/4

Möglichkeiten

- Ermittlung des Scorings der Solver in einer Kategorie oder Metakategorie.
- Zu jeder Competition können teilnehmende Solver, der Post-Prozessor und Statistik-Werte wie Anzahl JobPairs, CPU-Zeit und WallclockTime angesehen werden.

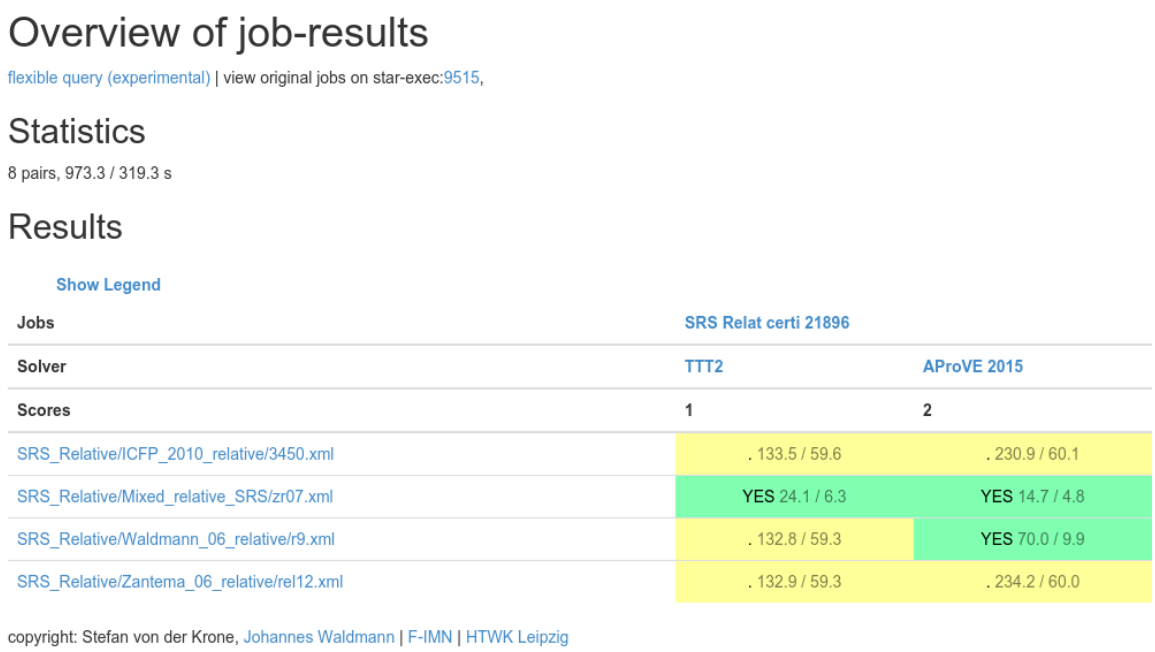
Probleme

- Es werden sehr viele Daten dargestellt.

4.2.2 Show Many Job Results

Beschreibung

Die Route *Show Many Job Results* zeigt ein einfaches Listing der Resultate eines oder mehrerer Jobs. Die Resultate eines Jobs werden in Form einer Tabelle visualisiert. Dabei sind alle teilnehmenden Solver (Kapitel 2.2) in Spalten und die zu lösenden Benchmarks der TPDB (Kapitel 2.1) in Zeilen dargestellt. Eine Zelle eines JobPairs (Paar aus Solver und Benchmark) zeigt die verschiedenen Resultatdaten (Kapitel 2.5) CPU-Zeit, (Wanduhr-)Zeit und als Zellen-Farbe[2] die Antwort. Zum Vergleich von mehreren Jobs müssen die JobIds in der URL mit einem Slash voneinander getrennt angegeben werden.



Overview of job-results

flexible query (experimental) | view original jobs on star-exec:9515,

Statistics

8 pairs, 973.3 / 319.3 s

Results

Show Legend

Jobs	SRS Relat certi 21896	
Solver	TTT2	AProVE 2015
Scores	1	2
SRS_Relative/ICFP_2010_relative/3450.xml	. 133.5 / 59.6	. 230.9 / 60.1
SRS_Relative/Mixed_relative_SRS/zr07.xml	YES 24.1 / 6.3	YES 14.7 / 4.8
SRS_Relative/Waldmann_06_relative/r9.xml	. 132.8 / 59.3	YES 70.0 / 9.9
SRS_Relative/Zantema_06_relative/rel12.xml	. 132.9 / 59.3	. 234.2 / 60.0

copyright: Stefan von der Krone, Johannes Waldmann | F-IMN | HTWK Leipzig

Abbildung 4.2: Star-Exec-Presenter: Show Many Job Results 9515
http://nfa.imn.htwk-leipzig.de/termcomp-devel/results/standard/noquery/9515

Möglichkeiten

- Anzeigen der Resultat-Daten.
- Vergleich von mehreren Job-IDs.

Probleme

- Mit steigender Anzahl von Benchmarks und Solvern steigt die Anzahl der Zeilen und Spalten. Damit wächst auch die Unübersichtlichkeit.
- Es ist kein Filtern der Daten möglich.

[2] Hellgrün=YES,Dunkelgrün=NO,Gelb=MAYBE

4.2.3 Flexible Table

Beschreibung

Die *Flexible Table* bietet Filterungen der Resultat-Tabelle durch eine Query an. Als Beispiel werden alle Benchmarks herausgefiltert, die von allen Solvern gelöst wurden. In Abbildung 4.3 wird dazu im Zeilenfilter die Zeile ausgewählt in der beide Solver mit `YES` geantwortet haben. Die ursprüngliche Tabelle (Abbildung 4.2) wird auf die zwei `JobPairs` mit `YES` reduziert. Abbildung 4.4 zeigt die gefilterte Resultat-Tabelle.

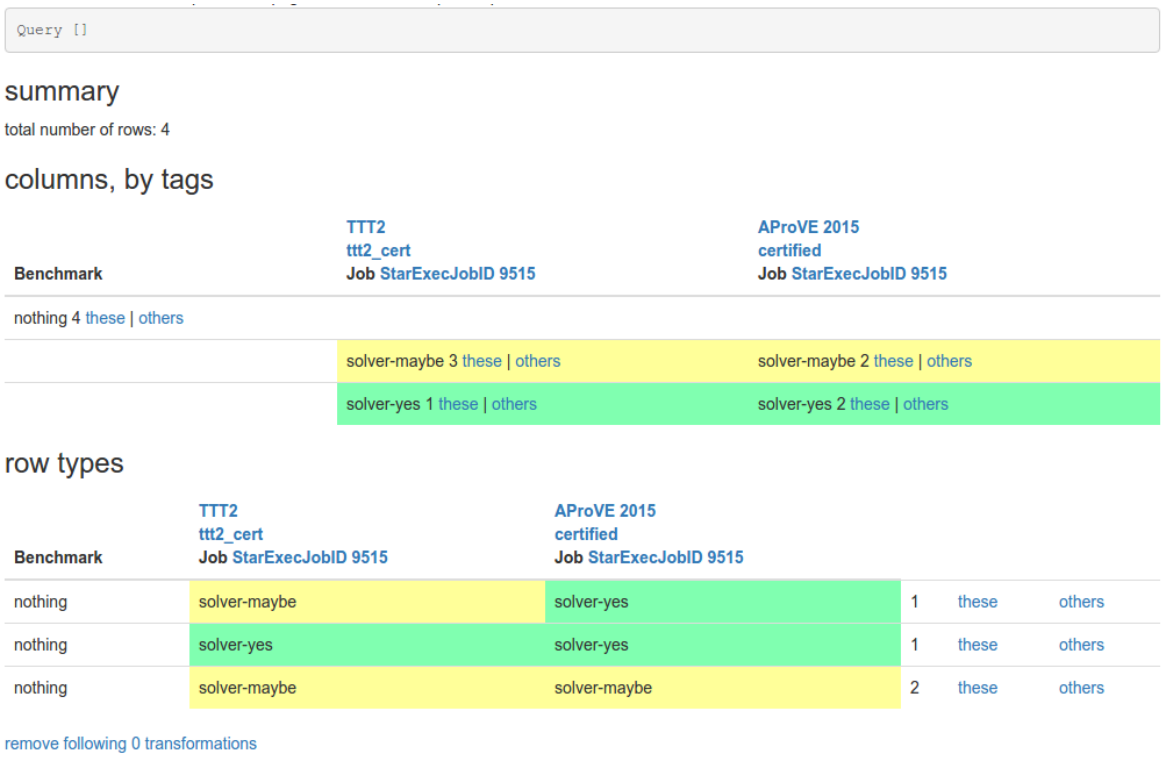


Abbildung 4.3: Star-Exec-Presenter: Flexible Table Auswahl von Spalten- und Zeilenfiltern 9515
http://nfa.imn.htwk-leipzig.de/termcomp-devel/flexible-table/Query%20%5B%5D/9515

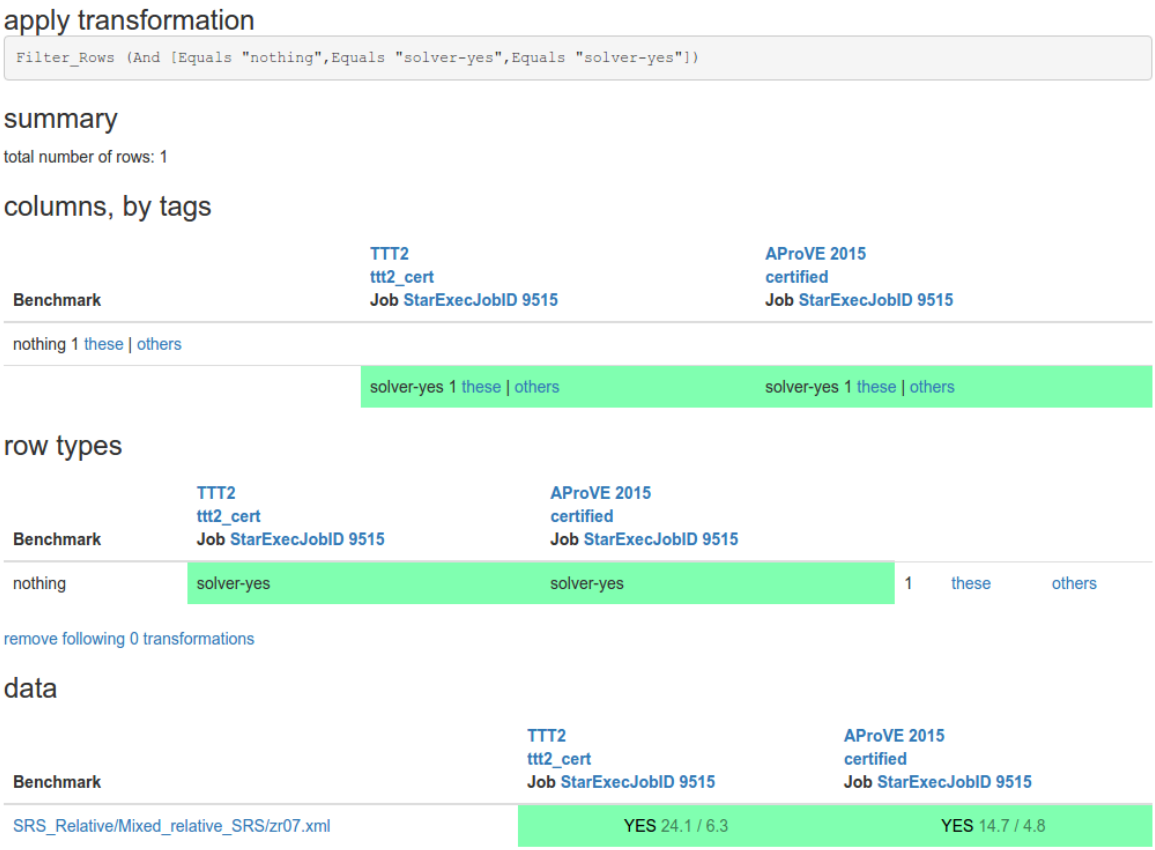


Abbildung 4.4: Star-Exec-Presenter: Flexible Table Resultat 9515
http://nfa.imn.htwk-leipzig.de/termcomp-devel/flexible-table/Query%20%5B%5D/9515

Möglichkeiten

- Dem Nutzer wird das Filtern der gesamten Resultat-Tabelle bezüglich der Solver-Antworten ermöglicht. Es gibt Filter für Spalten und Zeilen. Konkret lässt sich die Tabelle auf eine Solver-Antwort-Kombination YES-MAYBE in einer Zeile oder Zellen mit YES in einer Spalte beschränken. Die Negation der Auswahl ist auch möglich.

Probleme

- Die Vielzahl an möglichen Solver-Antwort-Kombinationen macht die Seite unübersichtlich.
- Nach mehrfacher Auswahl dieser Selektoren für Zeilen oder Spalten verlängert sich die Seite und der Nutzer erhält weitere Möglichkeiten.
- Es besteht Redundanz mit *Show Many Job Results*.

4.3 Spezifikation

Aus den Interview-Antworten und bestehenden Lösungen wird die Spezifikation erarbeitet.

Die Haupt-Anwendung für Nutzer von Star-Exec-Presenter ist das Betrachten und Analysieren der Competition-Resultate. Demnach sollte die Implementierung diese Anwendung unterstützen. Die Analyse der Resultat-Daten beinhaltet Vergleiche von Solvern/Konfigurationen bezüglich einer Auswahl von Benchmarks aus der TPDB 2.1. Es können auch ganze Competitions untereinander vergleichen werden.

Für Star-Exec-Presenter werden die folgenden Features gewünscht:

1. Einschränkung der Datenmenge durch Filter
2. mehr Übersichtlichkeit, besserer Zugang zu Informationen
3. eine verbesserte Navigation (z.B.: durch ein Menü)
4. mehr Informationen über die Benchmarks (Inhalt, Größe, Anzahl Zeilen)
5. Übersichtstabellen, Plots

Mit der Route *Show Many Job Results* (Kapitel 4.2.2) ist es möglich verschiedene Jobs zu vergleichen. Bei diesen Vergleichen empfängt der Nutzer eine Menge von Daten. In der größten Kategorie *TRS Standard* aus dem Jahre 2015 treten sieben Solver in 1498 Benchmarks gegeneinander an und bilden im Produkt 10486 Resultate[3]. Wenn der Nutzer diese Competition noch mit weiteren Competitions vergleichen möchte, ist es nachvollziehbar, weshalb ein Filter zur Einschränkung der Datenmenge ein Feature-Wunsch ist. Die Route *Flexible Table* (Kapitel 4.2.3) erfüllt die Anforderungen an einen Filter, jedoch nur in Bezug auf die Solver-Antwort. Die *Competition-Übersicht* (Kapitel 4.2.1) erfüllt den Zweck des rudimentären Listings der Resultate verschiedener Kategorien. Dieses Listing berührt den Kontext der Analyse nicht und steht außerhalb des Rahmens der Arbeit.

Mit dem Einsatz von Formaler Begriffsanalyse ist es denkbar die Features 1 und 2 zu erfüllen. Durch die Bestimmung des Begriffsverbandes entstehen Teilmengen von Zellen der Resultat-Tabelle, deren Inhalt angezeigt werden kann. Wenn weniger Zellen als Umfang eines Begriffes angezeigt werden, erfüllt dies den zweiten Feature-Wunsch.

Im Rahmen der Visualisierung und Verbesserung der Übersichtlichkeit soll eine Navigationsleiste hinzugefügt werden. Weitere Querverbindungen zwischen den Routen sollen die Navigation im Allgemeinen und gleichzeitig auch die Übersicht fördern.

Die Feature-Wünsche 4. und 5. überschneiden sich nicht mit dem Themenkomplex der Arbeit und werden nicht betrachtet.

Zusammenfassend soll die Implementierung die Features 1-3 erfüllen. Mithilfe Formaler Begriffsanalyse soll ein weiterer Filter für die Resultat-Tabelle erstellt werden. Eine Navigationsleiste und

[3] http://nfa.imn.htwk-leipzig.de/termcomp-2015/results/standard/noquery/10257

Verbindungen der Routen untereinander sollen die Navigation und auch den Zugang zu Informationen erleichtern.

Die Spezifikation ist erfüllt wenn die Zielsetzungen umgesetzt wurden. Weiter soll geprüft werden, ob die Implementierung für die Nutzer von Star-Exec-Presenter hilfreich ist. Zur Evaluation wurden die Interview-Partner erneut angeschrieben, die Features beschrieben und präsentiert. Die Auswertung befindet sich im Kapitel 7.

5 Haskell

Die Programmiersprache Haskell wird als Modellierungs- und Implementierungssprache verwendet. Der Vorteil der Verwendung von Haskell als Modellierungssprache ist die einfache Übertragung der mathematischen Spezifikation in Programmcode. Die Implementierungssprache ist durch die bestehende Web-Applikation Star-Exec-Presenter vorgegeben. Es werden verwendete Features von Haskell und des Yesod Web-Frameworks erläutert.

5.1 Die Sprache

> "Haskell is a general purpose, purely functional programming language incorporating many recent innovations in programming language design. Haskell provides higher-order functions, non-strict semantics, static polymorphic typing, user-defined algebraic datatypes, pattern-matching, list comprehensions, a module system, a monadic I/O system, and a rich set of primitive datatypes, including lists, arrays, arbitrary and fixed precision integers, and floating-point numbers. Haskell is both the culmination and solidification of many years of research on non-strict functional languages."[Mar10, S.3]

In diesem Zitat aus dem *Haskell 2010 Language Report*[Mar10] werden die wichtigsten Sprach-Feature genannt. Ergänzend sei anzumerken, das die Innovationen aus Haskell eine Vielzahl von Programmiersprachen wie *C#*, *Isabelle*, *Java*, *Python*, *Scala* und neue Sprachen wie *Rust* beeinflusst haben [Hud+07, S.45ff] [Kla+16]. Demnach spielt Haskell eine wichtige Rolle unter den Programmiersprachen. Erklärungen und allgemeine Beispiele führen einzelne Features der Sprache aus. Quellcode-Verweise für die Verwendung(en) der Sprach-Elemente in der Arbeit werden neben dem Feature in Klammern angegeben.

Anmerkung: Einige der Sprach-Feature verfügen über kein passendes deutsches Pendant und wurden aus dem Englischen übernommen.

Funktionale Programmierung

In der funktionalen Programmierung werden Programme durch Ausdrücke[1] in Form von Funktionen beschrieben. Im Gegensatz zu Programmen, die in imperativen Programmiersprachen wie *Java* oder *C* geschrieben sind. In diesen wird ein globaler Zustand durch eine Folge von Anweisungen[2] verändert. In der funktionalen Programmierung ist das nicht möglich. Ein Wert hat für den gesamten Programmablauf den gleichen Wert. [Mic13, S.XVff] Weitere entscheidende Eigenschaften sind *Lazy Evaluation*, *pure functions* und *Funktionen höherer Ordnung* (es sei auf die zugehörigen Abschnitte verwiesen).

Lazy Evaluation

Haskell ist *lazy* und wertet keinen Ausdruck aus, bevor er nicht tatsächlich benötigt wird [Mic13, S.XVff]. Das Beispiel 5.1 zeigt die Verwendung einer unendlichen Liste von der nur die ersten 7 Elemente berechnet werden. Mit dem Rest der Liste passiert nichts, da dieser nicht gebraucht wird.

List Comprehension

List Comprehension ist eine kompakte Form Listen durch Generator-Ausdrücke zu erzeugen, die speziellen Bedingungen genügen. Im Beispiel 5.1 werden durch `[x*x | x <- [1..]]` die Quadratzahlen aller Zahlen ab 1 berechnet. Der Ausdruck `even x` schränkt die generierten Werte für `x` auf gerade Zahlen ein.

```haskell
take 7 [x*x | x <- [1..], even x]
```

Quelltext 5.1: List Comprehension und Laziness

Pure Functions [3.2,3.3, 3.5]

In der rein funktionalen Programmiersprache Haskell sind Funktionen *pure*[3]. Das bedeutet eine Funktion ist frei von Nebenwirkungen. Bei mehrmaligem Aufruf einer *pure function* mit den gleichen Argumenten erhält man stets das gleiche Resultat. [Mic13, S.153ff] Das Beispiel 5.2 zeigt zwei Aufrufe von *pure functions*. Haskell bietet auch die Möglichkeiten Funktionen mit Nebeneffekten zu handhaben, dazu wurden Monaden eingeführt.

```haskell
length [0..99]

sin (pi/2)
```

Quelltext 5.2: Pure Function

[1] engl. expressions
[2] engl. statements
[3] engl. für rein/pur

Funktionen höherer Ordnung [3.2,3.3, 3.5, 6.13]

Funktionen höherer Ordnung sind Funktionen, die Funktionen als Argument oder Rückgabewert akzeptieren [Mic13, S.63]. Ein Beispiel (5.3) für eine Funktion höherer Ordnung ist `map`. Im Beispiel erhält `map` eine anonyme Funktion durch einen Lambda-Ausdruck und wendet diese Funktion auf jedes Element der Liste `[0..5]` an.

Lambda-Ausdruck [3.3, 3.2, 6.13, 6.6]

Ein Lambda-Ausdruck bezeichnet eine anonyme Funktion. Es wird ein Lambda-Ausdruck statt einer Funktion verwendet, wenn dieser nur einmal benötigt wird. Das Beispiel 5.3 zeigt eine einfache Funktion die jedes Argument `x` inkrementiert.

```
1 map (\x -> x +1) [0..5]
```

Quelltext 5.3: Funktion höherer Ordnung und Lambda

Algebraischer Datentyp [6.1,6.2, 6.16]

Ein algebraischer Datentyp erlaubt die Kombination von Typen durch die algebraischen Operationen Summe und Produkt. Die Summe der Konstruktoren ergibt den algebraischen Typ. Ein Konstruktor ist eine Alternative zum Instanziieren des algebraischen Typs. Jeder Konstruktor stellt mit seinen Subtypen ein Produkt dar. Demnach ist ein algebraischer Datentyp eine Summe von Produkten. Das Beispiel 5.4 zeigt den algebraischen Datentyp `Tree` mit der Summe aus den beiden Konstruktoren `Leaf` und `Branch`. `Leaf` ist ein einstelliges und `Branch` ein zweistelliges Produkt. [Mar10, S.15]

```
1 data Tree a = Leaf a | Branch (Tree a) (Tree a)
```

Quelltext 5.4: Algebraischer Datentyp

Statisch polymorphe Typisierung [3.1,3.4, 6.3]

Haskell ist eine statisch typisierte Programmiersprache - das heißt zur Compile-Zeit stehen alle Typen fest und ändern sich zur Laufzeit nicht, wie das bei dynamisch typisierten Sprachen möglich ist. Das hat den Vorteil das Fehler frühzeitig durch den Compiler erkannt und Typ-Fehler zur Laufzeit vorgebeugt werden. Das Beispiel 5.4 zeigt den Typparameter `a` für den beliebige Typen wie beispielsweise `String` oder `Maybe Int` eingesetzt werden können.

Record-Typ [3.1,3.4, 6.3, 6.7]

Haskell stellt mit dem Record-Typ einen Datentyp mit benannten Feldern zur Verfügung [Mar10, S.16]. Alle Felder müssen bei Instanziierung angegeben werden. Aus den Namen der Felder leiten sich Funktionen ab, die das Abfragen der Werte des Records ermöglichen. Im Beispiel 5.5 wird eine Person mit einem Namen und einem Alter deklariert.

```haskell
data Person = Person
              {name :: Text
              , age :: Int
              } deriving {Eq,Show}
```

Quelltext 5.5: Record-Typ und Typklassen

Typklassen [3.1,3.4, 6.2, 6.1]

Typklassen definieren durch eine Menge von Funktionen ein Verhalten [Mic13, S.63]. Je nach Typ für den die Typklasse implementiert wird, kann die Implementierung und das Verhalten unterschiedlich sein [OGS08, S.136]. Es ist für Funktionen möglich die Implementierung einer Typklasse vorauszusetzen, um sicher zu gehen, dass das Verhalten für den Typ sichergestellt ist. Das Beispiel 5.6 zeigt die Typklasse `Eq`, die instanziiert sein muss, um den Gleich- und Ungleichheitsoperator $==$, $/=$ verwenden zu können. Mit der Implementierung der Typklassenfunktionen für `Eq` wird definiert, wann ein Typ `a` gleich und ungleich ist. Der Haskell-Compiler kann die Instanzen für `Eq`, `Show` und andere Typklassen selbst ableiten [OGS08, S.136]. Bei einer Datentyp-Deklaration wird diese automatische Ableitung mit dem Schlüsselwort `deriving` (siehe Beispiel 5.5 für die Ableitung der Typklassen `Eq` und `Show`) eingeleitet.

```haskell
class Eq a where
    (==) :: a -> a -> Bool
    (/=) :: a -> a -> Bool
```

Quelltext 5.6: Typklassen

Pattern Matching [6.5 6.14 6.15]

Mit *pattern*[4] *matching* wird auf eine übersichtliche Art und Weise Datentypen von oben nach unten auf eine Reihe von Muster geprüft und bei Übereinstimmung entsprechende Aktionen ausgelöst. Für jeden Konstruktor des Datentyps sollte ein Fall der Behandlung existieren. Ein Muster erlaubt die Zerlegung des Typs in seine Bestandteile und den Zugriff auf dessen Werte. Das Beispiel 5.7 zeigt eine Reimplementierung der partiellen Funktion `head`, die das erste Element einer Liste zurückgibt. Im ersten Fall der Funktion `maybeHead` wird auf die leere Liste `[]` geprüft und `Nothing` zurückgegeben. Der zweite Fall spaltet die Liste in das erste Element und den Rest der Liste auf. Das erste Element wird mit `Just x` zurückgeben.

```haskell
maybeHead :: [a] -> Maybe a
maybeHead [] = Nothing
maybeHead (x:_) = Just x
```

Quelltext 5.7: Pattern Matching

[4] engl. für Muster

Monaden [6.5, 6.6, 6.9]

Eine Monade beschreibt einen Berechnungskontext und wird in Haskell genutzt, um *pure code* von *inpure code* zu trennen. Bei der Verwendung einer Monade wird ein Wert in den Berechnungskontext dieser Monade verpackt und muss zunächst wieder entpackt werden, um diesen in *pure functions* verwenden zu können. Eine Monade implementiert die Typklasse `Monad` und ihre Funktionen. Beispielanwendungen für Monaden sind der Zugriff auf eine Datenbank und Dateien. Beispiel-Monaden sind die `Handler`-Monade in Yesod 5.2, der `Maybe`-Datentyp sowie die Listen-Datenstruktur `[a]`. Die Verwendung des Schlüsselwortes `do` ist eine Kurzschreibweise um einfacher auf die Werte innerhalb Monaden zugreifen und diese mit Werten anderer Monaden verknüpfen zu können. Die Funktion `maybeHead` im Quelltext 5.7 gibt einen Wert oder nichts im Kontext der `Maybe` Monade zurück.

Modulsystem

Ein Modul beinhaltet Werte, Funktionen, Datentypen, Typklassen usw., ein Haskell-Programm wiederum eine Sammlung von Modulen [Mar10, S.61]. Mit dem Schlüsselwort `export` werden Bestandteile aus einem Modul anderen Modulen zur Verfügung gestellt. Mit `import` wird in einem Modul Bestandteile anderer Module verfügbar gemacht.

5.2 Yesod Web Framework

> "We want a language that gives us guarantees that our code is doing what it should. Instead of writing up a unit test to cover every bit of functionality in our application, wouldn't it be wonderful if the compiler could automatically ensure that our code is correct? And as an added bonus, wouldn't it be nice if our code ran quickly too? Yesod is a web framework bringing the strengths of the Haskell programming language to the web development world."[Sno15, S.5]

Da Star-Exec-Presenter in Yesod geschrieben ist, sollen die Eigenschaften von Yesod kurz zusammengefasst werden. Yesod[5] ist ein Haskell Web-Framework und wurde von Michael Snoyman[6] initiiert. Yesod besteht aus mehreren Sprachen sogenannten DSLs[7]. Diese Sprachen besitzen je nach ihrem Aufgabengebiet eine unterschiedliche Syntax. Es werden in der Arbeit verwendete DSLs weitere wichtige Features vorgestellt und ausgeführt.

[5] http://www.yesodweb.com/
[6] http://www.snoyman.com/
[7] (Domain Specific Language) engl. für anwendungsspezifische Sprache

Routing und Handler

Die Routing-DSL beschreibt die auf dem Server zur Verfügung gestellten Routen. Das Beispiel 5.8 zeigt eine einfache Route aus dem Star-Exec-Presenter. Der Pfad der Route ist `/flexible-table/#Query/*JobIds` und beinhaltet die zwei Routenparameter `#Query` und `*JobIds`. Es wird zwischen verschiedenen Arten von Routenparametern unterschieden. Parameter mit einem `*` beginnend werden als `Dynamic single` bezeichnet und besitzen beim Aufruf der URL nur ein Parameter. `Dynamic multi`-Routenparameter wird ein `#` vorangestellt. [Sno15, S.60] Diese Art von Routenparameter darf nur einmal am Ende einer URL vorkommen und besitzt mehrere Parameter. Der Grund für das Vorkommen am Ende einer URL ist die Zuordnung der URL-Parameter zu den richtigen Routenparametern. [Sno15, S.60] Der Name der Route ist `FlexibleTableR` und beantwortet `GET`-Requests[8]. [Sno15, S.16]

```
1 /flexible-table/#Query/*JobIds FlexibleTableR GET
```

Quelltext 5.8: config/routes-L48

Den Routen muss eine Funktion zugeordnet werden, die auf den Aufruf reagiert. Diese Funktion wird im Yesod Web-Framework `Handler` genannt. Der Name für die Funktion und das Modul in dem die Funktion beschrieben wird, leiten sich aus dem Routenname und der HTTP-Methode ab. [Sno15, S.17] Für das Beispiel 5.8 ergeben sich der Modulname `FlexibleTable.hs` und `getFlexibleTableR` als Funktionsnamen (siehe Beispiel 5.9).

```
1 getFlexibleTableR :: Query -> JobIds -> Handler Html
2 getFlexibleTableR = getShowManyJobResultsR Standard
```

Quelltext 5.9: Handler/FlexibleTable.hs-L6

Persistent als Datenbank-Backend

Eine weitere anwendungsspezifische Sprache ist Persistent[9] - die Datenbank-Schnittstelle mit verschiedenen Backends [Sno15, S.93ff]. Das Beispiel 5.10 zeigt ein Model zur Beschreibung einer `Competition` mit dessen Feldern. Eine Datenbank-Anfrage zum Abfragen von öffentlichen Competitions zeigt die Funktion `getPersistPublicCompetitions'` im Beispiel 5.11.

```
1 CompetitionInfo
2     competition Competition
3     date UTCTime default=now()
4     public Bool default=True
5     deriving Show
```

Quelltext 5.10: config/models-L180

[8] engl. für Anfragen
[9] https://www.stackage.org/package/persistent

```
getPersistPublicCompetitions' :: YesodDB App [Entity CompetitionInfo]
getPersistPublicCompetitions' = selectList [ CompetitionInfoPublic ==. True ] [
    Desc CompetitionInfoDate ]
```

Quelltext 5.11: Handler/Presenter/PersistHelper.hs-L93

Hamlet für HTML-Templates

Das Template-System von Yesod verwendet die Template-Sprachen der Shakespeare-Familie[10] [Sno15, S.23]. Das Shakespeare-Paket beinhaltet Template-Sprachen für HTML, CSS[11] und Javascript. Alle DSLs sind typsicher und werden statisch compiliert. Im Folgenden wird nur auf Hamlet als DSL für HTML eingegangen, da die Verwendung diese Template-Sprache im Rahmen der Arbeit den größten Anteil besaß. Das Beispiel 5.12 zeigt einen Ausschnitt des Hamlet-Templates zur Darstellung der Resultat-Tabelle (siehe Abbildung 4.2). Schließende HTML-Tags sind nicht notwendig, da durch Einrückungen der Scope geklärt wird. CSS-Klassen werden standardmäßig verwendet. Es stehen Template-typische Kontrollelemente wie eine for-Schleife zur Verfügung um im Beispiel über alle Benchmarks zu iterieren. Yesod-Routen können mit einem @-Zeichen beginnend verwendet werden.

```
<h2>All solvers listed in the database

<div class="container-fluid">
  <table class="table table-condensed">
    <thead>
      <tr>
        <th>Name
    <tbody>
      $forall benchmark <- benchmarks
        <tr>
          <td><a href=@{ShowBenchmarkInfoR $ toBenchmarkID
              benchmark}>#{toBenchmarkName benchmark}</a>
```

Quelltext 5.12: templates/listbenchmarks.hamlet

[10] https://hackage.haskell.org/package/shakespeare

[11] (Cascading Style Sheets) Deklarative Sprache, die die Darstellung einer Webseite kontrolliert (https://developer.mozilla.org/en-US/docs/Glossary/css)

6 Implementierung

Im Kapitel 4.3 wurden Ziele aus der bestehenden Implementierung und den Feature-Wünschen der Nutzer erarbeitet. Die Implementierung dieser Ziele wird in der Programmiersprache Haskell (Kapitel 5) umgesetzt. Ein Entwurf stellt die Teilaufgaben sowie deren Zusammenhänge vor und erläutert allgemeine Entscheidungen. Bedienelemente und zugehörige Entscheidungen zur Nutzeroberfläche werden anschließend diskutiert. Die Implementierung der Teilaufgaben folgt danach in einzelnen Unterkapiteln. Optimierungen für die bestehende Implementierung bilden den Abschluss.

6.1 Entwurf

Die Formale Begriffsanalyse soll genutzt werden, dem Nutzer eine weitere Filter-Möglichkeit bereitzustellen. Dazu muss die Formale Begriffsanalyse auf die Resultat-Daten der Jobs angewendet werden. Der Nutzer soll den neuen Filter auf einer neuen Seite der Yesod-Applikation (Kapitel 5.2) Star-Exec-Presenter verwenden können. Dazu muss eine neue Seite erstellt und diese in die bestehende Applikation integriert werden. Eine Nutzeroberfläche zur Visualisierung, Interaktion und Navigation dient dem Transport der Informationen der Formalen Begriffsanalyse.

Die Implementierung erfolgt durch die Lösung der folgenden Teilaufgaben:

1. Anwendung der Formalen Begriffsanalyse
2. Integration in Star-Exec-Presenter
3. Visualisierung der Formalen Begriffsanalyse
4. Interaktion und Navigation

Es werden verschiedene Aspekte der Teilaufgaben angesprochen und begründete Entscheidungen getroffen.

6.1.1 Wahl der Ausgangsmenge zur Berechnung des Begriffsverbandes

Zur Berechnung der Formalen Begriffe werden alle Teilmengen der Merkmale oder Gegenstände des Formalen Kontextes bestimmt. Wie in Kapitel 3.5.1 angegeben, bestimmt die geringere Mächtigkeit der Merkmale oder Gegenstände die Ausgangsmenge für den Algorithmus zur Berechnung des Verbandes. Im Falle der Termination-Resultat-Daten ist die Mächtigkeit der Merkmale oft geringer als die der Gegenstände.

Beispiel 6.1.1 Mächtigkeit der Gegenstände und Merkmale der JobID 10299

http://termcomp.imn.htwk-leipzig.de/concepts/0/Ids%20%5B%5D/10299

Die Mächtigkeit der Gegenstände entspricht der Anzahl der JobPairIDs. JobID 10299 besitzt 820 JobPairIDs. Diese resultieren aus vier Solvern und 205 Benchmarks.

Die Mächtigkeit der Merkmale ergibt sich aus den verschiedenen Merkmalen alle JobPairIDs. Im Falle der JobID 10299 gibt es vier Solver mit jeweils einer Konfiguration, 3 Resultat-Werte (YES, MAYBE, NO) und jeweils den booleschen Werten für CPU-Zeit, Anzahl-Regeln und links-linear. In Summe sind das $8 + 3 + 6 = 17$ Merkmale.

Mächtigkeit der Gegenstände $|G| = 820$ und Mächtigkeit der Merkmale $|M| = 17$.

Entscheidung: Auf Basis des Beispiels 6.1.1 wurde entschieden, dass die Merkmale der JobPairIDs als Ausgangsmenge für die Berechnung des Begriffsverbandes gewählt werden.

6.1.2 Reduktion der Merkmalmenge

Durch das Berechnen der Potenzmenge der Merkmale M werden alle Teilmengen der Menge M bestimmt. Jedoch schließen alle Teilmengen von M auch die Menge {Solvername1, Solvername2} ein. Eine JobPairId kann jedoch stets nur einen Solver als Merkmal besitzen. Das heißt, die Projektion (Kapitel 3.4) {Solvername1, Solvername2}′ ist die leere Menge, da die Projektionen {Solvername1}′ und {Solvername2}′ disjunkt[1] sind. Das gilt für alle Merkmale einer Merkmalsgruppe. Eine Merkmalsgruppe bezeichnet eine Menge von Merkmalen, die zu dem gleichen Merkmalstyp wie Solvername gehören.

[1] Zwei Mengen heißen disjunkt, wenn sie keine gemeinsamen Elemente besitzen.

Beispiel 6.1.2 Potenzmenge der Merkmale und Kombination von Merkmalsgruppen

Im Beispiel 6.1.1 wurde die Mächtigkeit der Merkmale für die JobPairId mit |M| = 17 bestimmt. Zur Berechnung aller Teilmengen von M wird mit *subsequences* die Potenzmenge der Menge |M| bestimmt. Das entspricht der Mächtigkeit von $2^{|M|} = 2^{17} = 131072$ Teilmengen.

Die 17 Merkmale der JobID 10299 zerlegen sich in die sechs Merkmalsgruppen: |Solver| = 4, |Konfigurationen| = 4, |Solver-Antworten| = 3, |CPU-Zeit| = 2, |Anzahl Regeln| = 2, |links-linear| = 2
Bei der Kombination der Merkmalsgruppen sei es auch erlaubt das kein Merkmal einer oder mehrerer Merkmalsgruppe gewählt wird. Aus diesem Grund wird imaginär zu jeder Merkmalsgruppe die Möglichkeit des Nichtwählens durch eine weitere Option hinzugefügt. Aus den Mächtigkeiten der Merkmalsgruppen ergibt sich die Obergrenze der Teilmenge von $\binom{5}{1} * \binom{5}{1} * \binom{4}{1} * \binom{3}{1} * \binom{3}{1} * \binom{3}{1} = 5 * 5 * 4 * 3 * 3 = 2700$.

Der Unterschied zwischen den beiden Werten für die Anzahl der Teilmengen zur Berechnung des Begriffsverbandes ist deutlich. Unter Verwendung von Kombination von Merkmalsgruppen werden deutlich weniger überflüssige Projektionen und Berechnungen ausgeführt.

Entscheidung: Im Beispiel 6.1.2 wird gezeigt, dass es für die StarExec-Daten sinnvoll ist, die Kombination der Merkmalsgruppen anstatt die Potenzmenge aller Merkmale zu berechnen.

6.1.3 Codierung der Merkmale

Jedes JobPair einer Resultat-Tabelle besitzt Merkmale. Jedes JobPair kann nach der Vorstellung einer Kreuztabelle ein Merkmal besitzen oder nicht. Bei dem Merkmal Solvername oder Solver-Konfiguration sind die möglichen Werte begrenzt und überschreiten nicht die Anzahl von zehn. Bei dem Merkmal *CPU-Zeit* können die JobPair jedoch alle Werte zwischen 0 Sekunden und 300 Sekunden annehmen. Ähnlich verhält es sich bei dem Merkmal *Anzahl der Regeln* eines Termersetzungssystems. Es existieren TRS mit den unterschiedlichsten Regel-Anzahlen.

Entscheidung: Bei Merkmalen *CPU-Zeit* und *Anzahl der Regeln* bestimmt der boolesche Wert des Ausdrucks den Merkmalswert. In der Tabelle 6.1 werden die Merkmale und ihre Ausdrücke aufgeführt.

Merkmal	Ausdruck
CPU-Zeit	>10 Sekunden (ist langsam)
Anzahl der Regeln	<10 Regeln (ist gering)

Tabelle 6.1: Merkmale und ihre booleschen Ausdrücke

6.2 Nutzeroberfläche

Die Formale Begriffsanalyse soll dem Nutzer, ohne Kenntnis über diese, zur Verfügung gestellt werden. Im Rahmen der Arbeit wurden einige Entscheidungen getroffen, um die Visualisierung der Daten, der Interaktion mit der neuen Seite so intuitiv wie möglich zu gestalten. Screenshots der Bedienelemente der Nutzeroberfläche zeigen die Umsetzung der Entscheidungen. Die Grafik 6.1 zeigt die Bedienelemente einschließlich URL und ihre Beeinflussung innerhalb der neuen Seite.

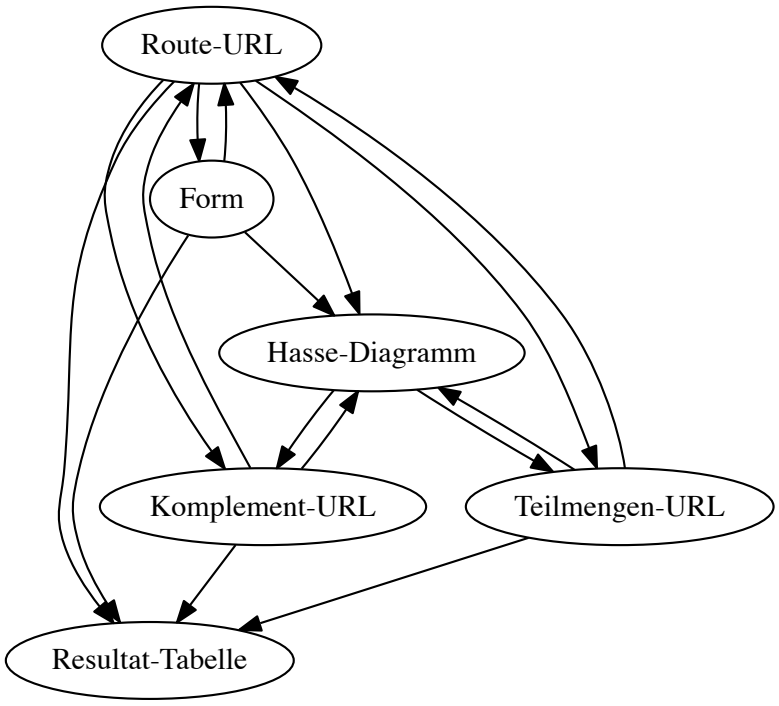


Abbildung 6.1: Bedienelemente und ihre gegenseitige Beeinflussung

6.2.1 Formular

Für einige JobIDs, speziell für die Competition ist die Anzahl der JobPairs sehr groß.

Entscheidung: Ein HTML-Formular (Abbildung 6.2) soll dem Nutzer die Möglichkeit geben, seine Daten bereits so frühzeitig wie möglich einzuschränken. Das HTML-Formular zeigt alle Merkmale der aktuellen JobPairs, die zur gewählten JobIDs gehören. Die Merkmale sind ihren Merkmalsgruppen zugeordnet. Nach dem Absenden des Formulars verbleiben nur JobPairs mit gewählten Merkmalen.

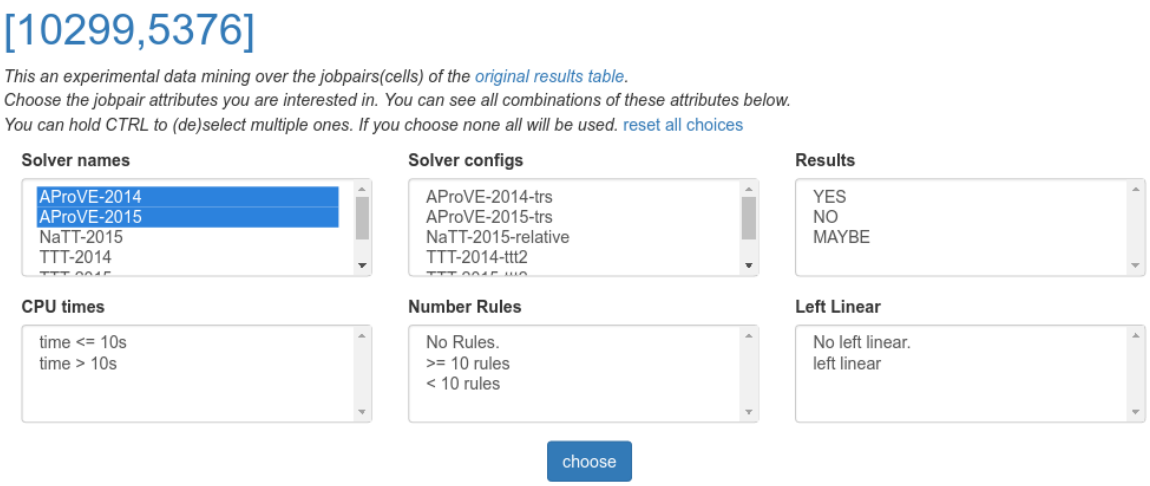


Abbildung 6.2: HTML-Formular zur Wahl der Merkmale aus den Merkmalgruppen
http://termcomp.imn.htwk-leipzig.de/concepts/0/Ids%20%5B%5D/10299/5376?_hasdata&
SolverNames=1&SolverNames=2

6.2.2 Hasse-Diagramm

Ein Hasse-Diagramm visualisiert den Begriffsverband eines Formalen Kontextes. Im Hasse-Diagramm befinden sich nach Definition im Kapitel 3.2.5 das Infimum oben und das Supremum unten. Die Lese- und Scrollrichtung auf einer Webseite ist von oben nach unten.

Entscheidung: Das Hasse-Diagramm ist im Gegensatz zur Definition in Kapitel 3.2.5 vertikal gespiegelt (Abbildung 6.3).

Für den Nutzer muss die Richtung der Kanten eindeutig erkennbar sein.

Entscheidung: Um die Richtung im Hasse-Diagramm für den Nutzer deutlich zu machen, wurden gerichtete Kanten verwendet (Abbildung 6.3).

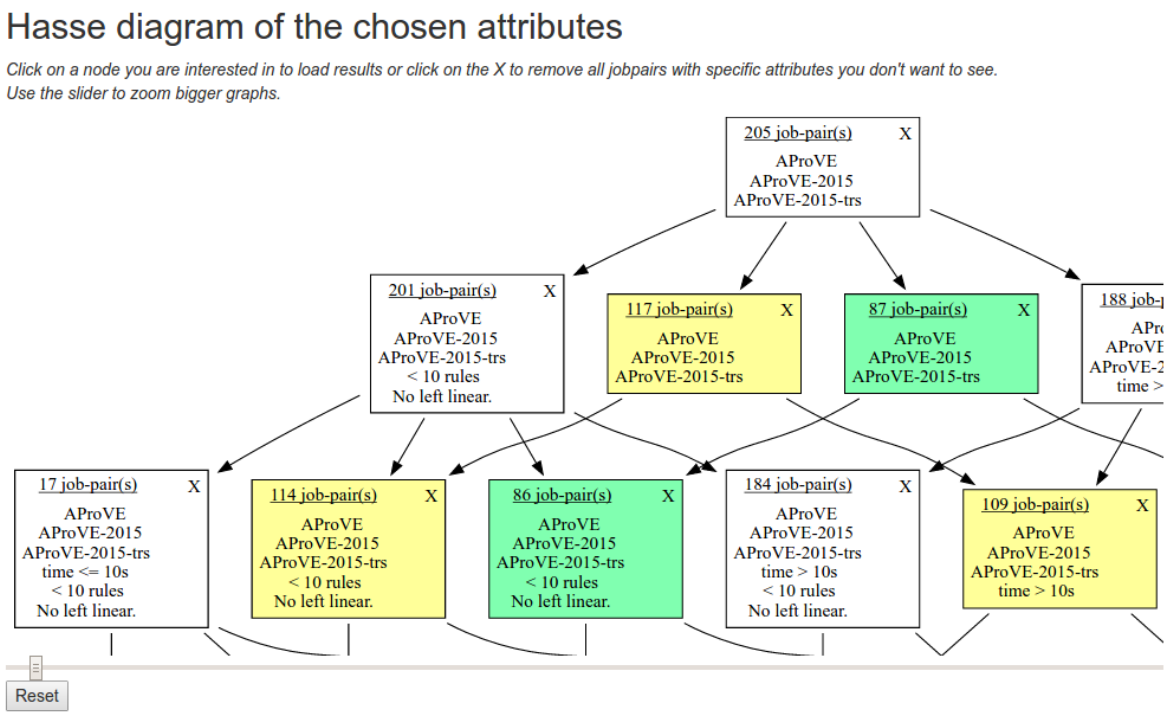


Abbildung 6.3: Hasse-Diagramm-Ausschnitt einschließlich des Zoom-Sliders
http://termcomp.imn.htwk-leipzig.de/concepts/0/Ids%20%5B%5D/10299?_hasdata&SolverNames=1

Das Hasse-Diagramm zeigt alle Begriffe des Begriffsverbandes. In jedem Knoten werden die Merkmale des Formalen Begriffs aufgeführt. Das führt zu Platzproblemen.

Entscheidungen:

- Merkmale werden innerhalb eines Knotens zeilenweise aufgeführt. Die Knoten-Farbe entspricht dem Merkmal Solver-Antwort und wurde aus der Zuordnung von Farbe und Solver-Antwort (zum Beispiel YES und MAYBE) übernommen.
- Das Infimum (siehe Kapitel 3.2.3) des Begriffsverbandes ist der Begriff in dem alle Merkmale vereinigt sind. Dieser Knoten wurde entfernt, da es kein JobPair das alle Merkmale besitzt.
- Zur Reduktion der Begriffe des Begriffsverbandes werden dem Nutzer zwei Links zur Verfügung gestellt:
  - Klick auf einen Knoten ermöglicht die Anzeige einer Teilmenge des Begriffsverband (Kapitel 6.6.1 und Abbildung 6.4).
  - Klick auf ein X in der rechten oberen Ecke des Knotens im Hasse-Diagramm um das Komplement des Begriffsverbandes (Kapitel 6.6.2) mit dem aktuellen Begriff zu bilden (Abbildung 6.5).
- Ein Zoom-Slider unterhalb des Diagramms ermöglicht das Zoomen im Hasse-Diagramm (Abbildung 6.3).

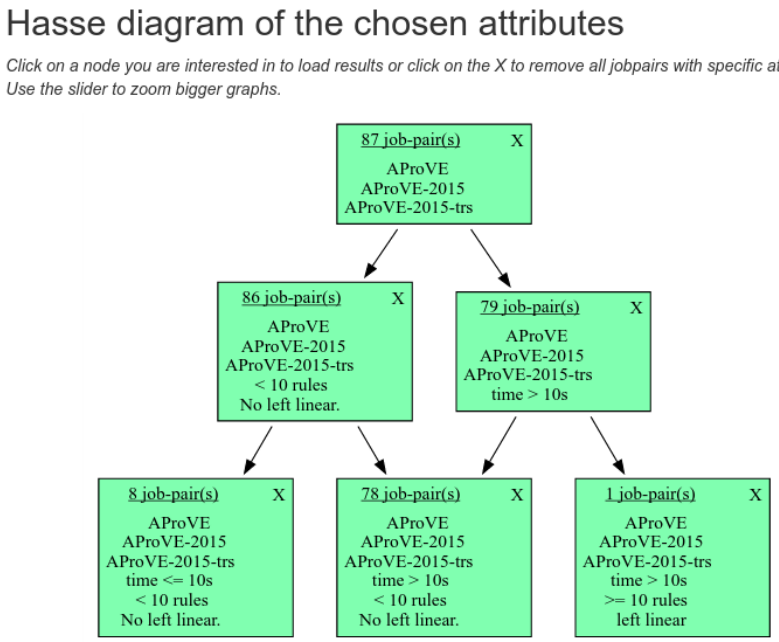


Abbildung 6.4: Hasse-Diagramm mit Begriffsverband-Teilmenge (Merkmale AProVE und YES)
http://termcomp.imn.htwk-leipzig.de/concepts/12/Ids%20%5B%5D/10299?_hasdata&SolverNames=1

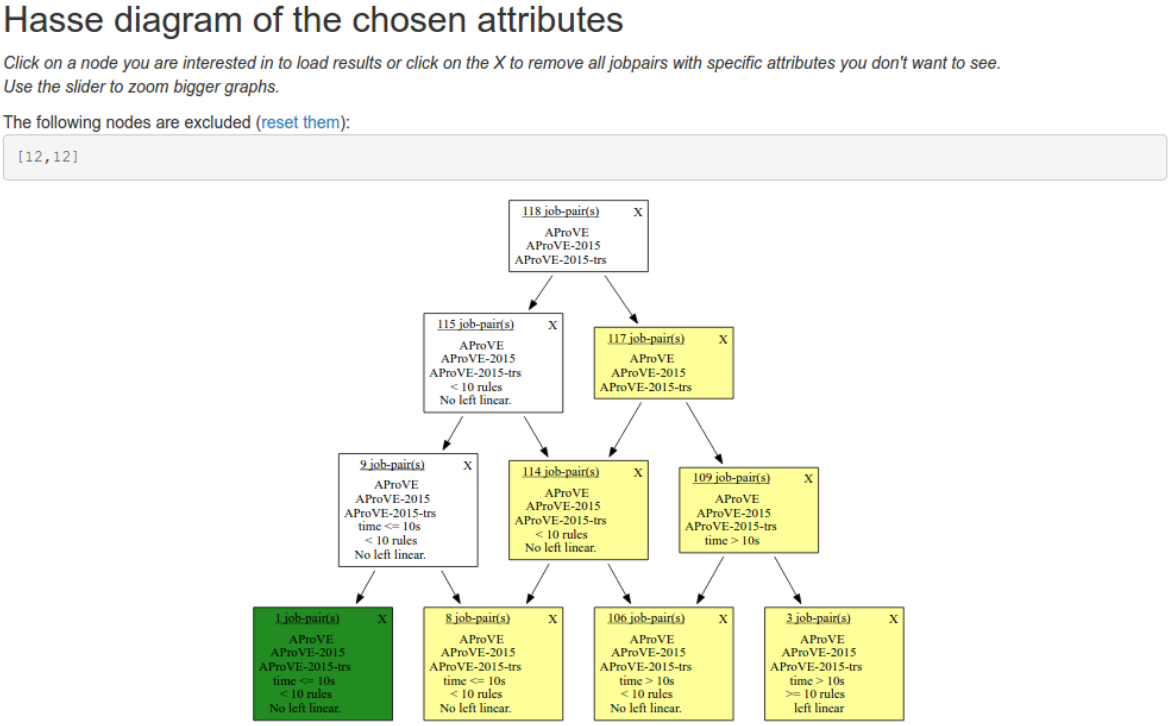


Abbildung 6.5: Hasse-Diagramm mit Begriffsverband-Komplement ohne YES
http://termcomp.imn.htwk-leipzig.de/concepts/0/Ids%20%5B12,12%5D/10299?_hasdata&SolverNames=1

6.2.3 gefilterte Resultat-Tabelle

Die Resultat-Tabelle zeigt die Ergebnisse der Jobs an und ist das zu filternde Objekt.

Entscheidung: Die bekannte Resultat-Tabelle bleibt bestehen und zeigt nur die Gegenstände des obersten Formalen Begriff des Hasse-Diagramms an.

Benchmark	AProVE 2015 trs Job StarExecJobID 10299
S_Relative/ICFP_2010_relative/107220.xml	. 421.8 / 300.0
S_Relative/ICFP_2010_relative/107236.xml	. 494.2 / 300.1
S_Relative/ICFP_2010_relative/123759.xml	. 422.8 / 300.0
S_Relative/ICFP_2010_relative/124211.xml	. 487.3 / 300.1
S_Relative/ICFP_2010_relative/124269.xml	. 563.9 / 300.0
S_Relative/ICFP_2010_relative/124791.xml	. 455.9 / 300.1
S_Relative/ICFP_2010_relative/128056.xml	. 584.5 / 300.1
S_Relative/ICFP_2010_relative/131633.xml	. 508.0 / 300.0
S_Relative/ICFP_2010_relative/131982.xml	. 371.2 / 300.1
S_Relative/ICFP_2010_relative/132235.xml	. 389.6 / 300.1
S_Relative/ICFP_2010_relative/132969.xml	. 507.6 / 300.1
S_Relative/ICFP_2010_relative/133159.xml	. 381.4 / 300.1
S_Relative/ICFP_2010_relative/133486.xml	. 529.6 / 300.0
S_Relative/ICFP_2010_relative/133881.xml	. 415.5 / 300.1
S_Relative/ICFP_2010_relative/134918.xml	. 437.8 / 300.1

Abbildung 6.6: Ausschnitt Resultat-Tabelle
http:
//termcomp.imn.htwk-leipzig.de/concepts/0/Ids%20%5B12,12%5D/10299?_hasdata&SolverNames=1

6.2.4 Navigation

Ein Feature-Wunsch ist die Einführung eines Menüs.

Entscheidung: Es wird ein Menü zur besseren Erreichbarkeit einzelner Routen eingeführt.

6.3 Anwendung der Formalen Begriffsanalyse

Aus dem Theorie-Kapitel sind die theoretischen Komponenten Formaler Kontext, Formaler Begriff und Begriffsverband bekannt. In diesem Kapitel soll die Formale Begriffsanalyse auf die Resultat-Daten der Jobs angewendet werden.

6.3.1 Formaler Kontext

Es wird die Implementierung der Gegenstände, Merkmale und der sich daraus zusammengesetzte Formale Kontext (Kapitel 3.3) vorgestellt.

Die zellenbeschreibenden `JobPairIDs` sind die Gegenstände. Die Merkmale für StarExec sind durch den algebraischen Datentyp `Attribute` definiert. Das Ausrufezeichen vor einem Typ bedeutet dass dieser Typ nicht lazy ist.

```
1 data JobPairID =
2   StarExecPairID Int
3   | LriPairID Int
4   | UibkPairID Int
5   deriving (Show, Read, Eq, Ord)
```

Quelltext 6.1: Presenter/Model/RouteTypes.hs-L113

```
1  data Attribute =
2    ASolverBasename !Text
3     | AJobResultInfoSolver !Text
4     | AYearSpecificSolverName !Text
5     | AJobResultInfoConfiguration !Text
6     | ASlowCpuTime !Bool
7     | ASolverResult !SolverResult
8     | ABenchmarkNumberRules !(Maybe Bool)
9     | ABenchmarkLeftLinear !(Maybe Bool)
10   deriving (Eq, Ord, Show)
```

Quelltext 6.2: FCA/StarExec.hs-L17

Aus den beiden Typen `JobPairID` als Gegenstand und `Attribute` als Merkmal wird aus dem Typparameter der polymorphen Definition in Kapitel 3.3.1 der Typ des Formalen Kontextes `Context` zur Compile-Zeit konkret.

```haskell
data Context JobPairID Attribute = Context
  { fore :: Map JobPairID (Set Attribute)
  , back :: Map Attribute (Set JobPairID)
  } deriving (Show)
```

Quelltext 6.3: FCA/StarExec.hs-L164 zur Compile-Zeit

Zur Berechnung des Formalen Kontextes dienen alle JobPairIDs aus den gewählten Tabellen als Gegenstände. Die Tabellen werden durch die Routenparameter (siehe Integration in StarExec-Presenter) für `JobIDs` bestimmt. Es werden für jede `JobPairID` alle Merkmale des Types `Attribute` kreiert.

Die Werte aller Konstruktoren von `Attribute` wurden aus dem `JobPair`-Datensatz der Datenbank (siehe Übersichtstabelle 2.2) abgeleitet. Eine besondere Rolle spielen die Merkmale `ABenchmarkNumberRules` und `ABenchmarkLeftLinear`. Diese beziehen sich auf die `BenchmarkId` des `JobPairs`. Um die Information über die Anzahl der Regeln und die Links-Linearität eines Benchmarks zur Verfügung zu haben, wurde das Haskell-Modul `FCA.Benchmarks` geschrieben. Das Modul durchläuft die aktuelle Version der TPDB (Kapitel 2.1) um jeden Benchmark zu untersuchen. Mit der neuen Version *1.3.3* des Haskell-Paketes `tpdb`[2] konnten die geforderten Merkmale für Termersetzungssysteme berechnet werden. Eine Erweiterung des Persistent-Models `BenchmarkInfo` durch die Felder `numberRules` und `leftLinear` in `config/models` erlaubt den Zugriff auf die Werte in der Datenbank.

```haskell
-- get attribute pairs of given job results
attributePairs :: [[JobResult]] -> Handler [(JobPairID, [Attribute])]
```

Quelltext 6.4: FCA/StarExec.hs-L61

Ein HTML-Formular schließt Gegenstände aus. Das Formular muss mit den zur Auswahl stehenden Merkmalen befüllt werden. Dazu ist es nicht notwendig den Formalen Kontext zu bestimmen. Stattdessen werden im ersten Schritt Paare vom Typ `attributePairs` berechnet. Im zweiten Schritt werden alle zur Verfügung stehenden Merkmale nach ihrem Konstruktor geordnet in eine Mehrfachauswahl[3] eingefügt. Die Abbildung 6.2 zeigt sechs der Felder mit ihren Optionen. Der Algebraische Datentyp `Attribute` hat acht Konstruktoren, jedoch werden `AJobResultInfoSolver` und `ASolverBasename` derzeit im Frontend nicht als Option verwendet.

```haskell
-- filter all attribute pairs by given attribute groups
filterPairsByAttributes :: [(JobPairID, [Attribute])] -> [[Attribute]]
                        -> Maybe [(JobPairID, [Attribute])]
filterPairsByAttributes pairs attrCombinations = do
  let anyMember = or . (\s -> map
                              (\v -> Set.isSubsetOf (Set.fromList v) s)
```

[2] https://hackage.haskell.org/package/tpdb
[3] https://getbootstrap.com/css/#selects

```
                                attrCombinations)
  let filteredJobResults = filter
                             (\(_,attrs) -> anyMember $ Set.fromList attrs) pairs
  case filteredJobResults of
    [] -> Nothing
    _ -> Just filteredJobResults
```

Quelltext 6.5: FCA/StarExec.hs-L80

Nach dem Absenden des Formulars werden die `attributePairs` mit `filterPairsByAttributes` gefiltert. Aus diesen gefilterten Paaren wird der Formale Kontext mit der Funktion `contextFromList` bestimmt.

```
-- create a context of given input data
contextFromList :: (Ord at, Ord ob) => [(ob, [at])] -> Context ob at
contextFromList l = Context
  { fore=Map.fromListWith
        Set.union $ map (\(ob, attrs) -> (ob, Set.fromList attrs)) l
  , back=Map.fromListWith
        Set.union $ do (ob, attrs) <- l; at <- attrs;
                       return (at,Set.singleton ob)
  }
```

Quelltext 6.6: FCA/Basic.hs-L86

6.3.2 Begriffsverband

Der Begriffsverband beinhaltet alle Formalen Begriffe (siehe 3.5) eines Formalen Kontextes. Im Implementierungskapitel 6.3.1 wird gezeigt, wie ein Formaler Kontext aus den Routenparametern berechnet wird. Auf Basis des Formalen Kontextes werden die einzelnen Formalen Begriffe berechnet. Der Begriffsverband wird als Hasse-Diagramm (siehe Kapitel 6.5.1) dargestellt.

Ähnlich zum Formalen Kontext wird der polymorphe Record-Typ `Concept` aus Kapitel 3.5 für die Implementierung des Formalen Begriffs konkret.

```
data Concept JobPairID Attribute = Concept
  { obs :: Set JobPairID
  , ats :: Set Attribute
  } deriving (Show, Eq)
```

Quelltext 6.7: FCA/Basic.hs-L31 zur Compile-Zeit

Zur Vermeidung von überflüssigen Teilmengen mit Elementen einer Merkmalsgruppe wurden die Berechnung aller Teilmengen der Menge M durch die Berechnung von Merkmalskombinationen

attributeCombination substituiert (siehe Entscheidung in 6.1.2). In Abbildung 6.2 wird eine Merkmalsgruppe durch eine Mehrfachauswahl präsentiert.

```haskell
-- create all attribute combinations from existing attributes without duplicates
attributeCombination :: (Ord at) => Context ob at -> [Set at]
attributeCombination context = do
  let attrs = Map.elems $ fore context
  -- using ordNub to reduce duplicate items and keep order
  ordNub . fmap Set.fromList . concatMap (subsequences . Set.toList) $ ordNub
      attrs
```

Quelltext 6.8: FCA/StarExec.hs-L157

Zur Berechnung valider - tatsächlich vorkommender Merkmalskombinationen - werden einzig die Potenzmengen von Merkmalen einer JobPairID vorgenommen. Das schließt Teilmengen mit mehreren Merkmalen einer Merkmalsgruppe aus. Das haskell-Paket ordNub[4] dient zum Entfernen von Duplikaten in einer Liste und das Beibehalten der Reihenfolge.

In der allgemeinen Vorschrift für die Berechnung des Begriffsverbandes aus dem Theorie-Kapitel (Definition 3.5.1) wird demnach subsequences durch die Funktion attributeCombination ersetzt. Daraus entsteht die neue Implementierung von concepts in Quelltext 6.9. Aus der Reihenfolge der Merkmale entsteht die Reihenfolge der Begriffe des Begriffsverbandes [Concept ob at].

```haskell
-- determine all concepts of given context with StarExec attributes
concepts :: (Ord at, Ord ob, Show ob, Show at)
         => Context ob at -> [Concept ob at]
concepts c = do
  attrs <- attributeCombination c
  let objs = getObjects c attrs
  unless (Set.null objs) . guard $ (attrs == (getAttributes c) objs)
  return (Concept objs attrs)
```

Quelltext 6.9: FCA/StarExec.hs-L164

Zur Visualisierung des Begriffsverbandes dient ein Hasse-Diagramm. Die Gegenstände eines Begriffs werden durch die Resultat-Tabelle dargestellt.

[4] https://github.com/nh2/haskell-ordnub#dont-use-nub

6.4 Integration in Star-Exec-Presenter

Um die Formale Begriffsanalyse nutzen zu können, bedarf es einer neuen Route und zugehörigen Handler in Star-Exec-Presenter. Es wird die Implementierung der Elemente des Yesod Web-Frameworks vorgestellt.

6.4.1 Concepts-Route

Die Concepts-Route hat den folgenden Aufbau:

`/concepts/#ConceptId/#ComplementIds/*JobIds ConceptsR GET`

Der erste Teil '`/concepts/#ConceptId/#ComplementIds/*JobIds` ist der Pfad der Route einschließlich der drei Routenparameter `#ConceptId/#ComplementIds/*JobIds`. Nach einem Leerzeichen folgt der Name der Route *ConceptsR*. Der letzte Teil `GET` gibt an, dass nur GET-Requests verarbeitet werden. Aus `ConceptsR` und `GET` ergibt sich der Name der Handler-Funktion `getConceptsR`, der bei Aufruf der URL die Verarbeitung übernimmt.

```haskell
-- Begriffs-ID
type ConceptId = Int

-- Komplement-IDs
data ComplementIds =
  Ids [ConceptId]
  deriving (Show, Read, Eq, Ord)

-- JobIDs
newtype JobIds = JobIds
  { getIds :: [JobID]
  }
  deriving (Show, Eq, Read)
```

Quelltext 6.10: Presenter/Model/RouteTypes.hs

An der Implementierung der Routenparameter lässt sich die Parameter-Anzahl erkennen. Es wird nur eine Begriffs-ID, mehrere Komplement-IDs und mehrere JobIDs verwendet. Die Gründe für die Anzahlen ergeben sich aus Erläuterungen in diesem Kapitel. Die JobIDs sind ein Beispiel für einen `Dynamic multi`-Routenparameter. Begriff-ID und Komplement-IDs sind `Dynamic single`-Routenparameter. Für die Implementierung der Komplement-IDs musste ein `Dynamic single` mit einer Liste verwendet werden. Die Gründe dafür sind die begrenzte Anzahl an `Dynamic multi`-Routenparametern in einer Route und die Notwendigkeit von mehreren Komplement-IDs. Es gibt ein HTML-Formular mit dem zusätzlich GET-Parameter in der URL (siehe Beispiel) möglich sind.

Beispiel 6.4.1 Concepts-Route

http://termcomp.imn.htwk-leipzig.de/concepts/0/Ids%20%5B%5D/10299/5376?_hasdata&SolverNames=1&SolverNames=2

ConceptId : 0

ComplementIds: `Ids%20%5B%5D` ($\hat{=}$ `URL-escaped Ids []`)

JobIDs : 10299 und 5376

GET-Parameter: SolverNames=1 und SolverNames = 2

Die Routenparameter skizzieren einen Formalen Kontext. Durch JobIDs werden die Tabellen ausgewählt, aus denen alle Zellen die Gegenstände bilden. Die GET-Parameter des Formulars beschränken diese Gegenstände nach einer Vorschrift (siehe Kapitel 6.3.1) auf gewählte Merkmale.

6.4.2 Concepts-Handler

Der Concepts-Handler ist eine Funktion. Diese Funktion wird mit den Parametern der Route aufgerufen (siehe Quellcode 6.11). Innerhalb des Concepts-Handler wird die Formale Begriffsanalyse gestartet und visualisiert. Auch alle weiteren Bestandteile wie die Form, das Hasse-Diagramm und die Resultat-Tabelle befinden sich in der Funktion.

```
1 -- route with multiselect to choose attributes of JobID
2 getConceptsR :: ConceptId -> ComplementIds -> JobIds -> Handler Html
```

Quelltext 6.11: Handler/Concepts.hs-L36

6.5 Visualisierung der Formalen Begriffsanalyse

Die Darstellung des Begriffsverbandes inklusive der Merkmale ist durch ein Hasse-Diagramm (siehe Definition 3.2.5) realisiert. Die Gegenstände werden in der Resultat-Tabelle angezeigt.

6.5.1 Hasse-Diagramm

Gemäß den Entscheidungen im Kapitel 6.2.2 wurde das Hasse-Diagramm implementiert. Den Ablauf für die Erzeugung des Hasse-Diagramm aus Formalen Begriffen `[Concept ob at]` bis zum SVG zeigt die folgende Zeile:

```
[Concept ob at] -> Graph -> DOT -> SVG
```

Das Hasse-Diagramm wird aus den Begriffen `[Concept ob FSE.Attribute]` und den beiden URLs in der Funktion `renderConceptSVG` berechnet. In der Funktion `dottedGraph` wird für jeden Formalen Begriff ein Knoten erstellt. Falls *Begriff I* Oberbegriff von *Begriff II* ist, wird eine gerichtete Kante von *Begriff I* zu *Begriff II* erzeugt. In den nächsten Schritten wird aus den Knoten und Kanten ein Graph und aus diesem DOT-Code[5] erzeugt. Aus dieser Graphviz[6]-internen Beschreibungssprache wird danach ein SVG generiert. Dieses SVG kann nativ im Hamlet-Template verwendet werden.

```
1  renderConceptSVG :: (Eq ob, Show ob, MonadIO m) => [Concept ob FSE.Attribute]
2                  -> [T.Text] -> [T.Text]-> m B.Markup
3  renderConceptSVG concepts' nodeURLs complURLs = do
4    svg <- liftIO . readProcess "dot" [ "-Tsvg" ]
5           $ dottedGraph concepts' nodeURLs complURLs
6    return . B.preEscapedLazyText . TL.pack . L.unlines
7           . L.dropWhile ( not . L.isPrefixOf "<svg" ) $ L.lines svg
8
9  dottedGraph :: (Eq ob, Show ob) => [Concept ob FSE.Attribute]
10             -> [T.Text] -> [T.Text] -> String
11 dottedGraph conceptLattice nodeURLs complURLs = do
12   let graph_params = getGraphParams conceptLattice nodeURLs complURLs
13   let graphWithTransEdges = G.graphToDot graph_params
14                             $ createGraph conceptLattice
15   (TL.unpack . renderDot) . toDot $ transitiveReduction graphWithTransEdges
```

Quelltext 6.12: FCA/DotGraph.hs-L24

Der oberste Knoten des Hasse-Diagramms entspricht dem Begriff mit dem Index der `ConceptId` (aus der `Concepts-Route`) in der Liste des Begriffsverbandes `[Concept ob at]`. Bei einem vollständigem Begriffsverband entspricht das dem Formalen Begriff mit allen Gegenständen des Formalen Kontext mit deren gemeinsamen Merkmalen.

In einem Formalem Kontext mit zu vielen Merkmalen und einem daraus resultierenden Begriffsverband mit mehr als 50 Begriffen[7] wird die Darstellung sehr problematisch. Dieses Problem tritt häufig bei dem Vergleich von mehreren JobIDs auf. Es ist nicht mehr möglich den Begriffsverband in einer Mindestgröße darzustellen. Um diesem Problem zu begegnen, wurde entschieden, dass nach Definition prädestinierte SVG in das jQuery-Plugin jQuery Panzoom[8] einzubetten.

Die Abbildung 6.5 zeigt deutlich die Hasse-Diagramm-Struktur einschließlich des Supremum. Einen Ausschnitt aus einem Hasse-Diagramm mit mehr Begriffen einschließlich Zoom zeigt die Grafik 6.3.

[5] http://www.graphviz.org/Documentation/dotguide.pdf
[6] https://hackage.haskell.org/package/graphviz
[7] Die 50 Begriffe wurden empirisch ermittelt. Ein Beispiel mit mehr als 50 Begriffen verdeutlicht die Problematik.
[8] https://github.com/timmywil/jquery.panzoom/

6.5.2 Resultat-Tabelle

Eine Tabelle dient zur Darstellung der JobPairs deren *JobPairIds* im aktuellen Begriff (Routenparameter `ConceptId`) vorhanden sind. Durch die Verwendung von Komplementen (siehe 6.6.2) kann der Formale Kontext außerdem um Gegenstände beschnitten sein. Um genau die Gegenstände anzuzeigen, werden in `currObjects` die aktuellen Gegenstände des Begriffsverbandes bestimmt. In dem Bezeichner `filteredJobResults` werden alle JobPairIds herausgefiltert, die sich nicht in `currObjects` befinden. Die bestehende Implementierung für die Darstellung der Resultate (bisherige Verwendung in Show Many Job Results und Flexible Table) wurde nur minimal angepasst. Die Anzeige des *Scorings* - die Anzahl der Solver-Antwort `YES` pro Solver - wurde entfernt, da diese Information bereits im Hasse-Diagramm verfügbar ist. Die Abbildung 6.6 zeigt einen Ausschnitt aus dem Begriffsumfang des obersten Begriffs des Hasse-Diagramms in 6.5.

```
1  let currObjects = maybe
2                    Set.empty
3                    (\ c -> safeGetConceptObjectsFromLattice c cid)
4                    concepts'
5
6  let filteredJobResults = fmap
7                           ((wrapResults .
8                             filter
9                             (\jr -> Set.member (getPairID jr) currObjects)
10                            . getStarExecResults)
11                            . snd . queryResult)
12                          qJobs
```

Quelltext 6.13: Handler/Concepts.hs-L71

6.6 Interaktion und Navigation

Jeder Knoten des Hasse-Diagramms besitzt zwei Arten von Links (Begriffsverband-Komplement und Begriffsverband-Teilmenge) zur Interaktion.

Innerhalb der `Concepts-Seite` gibt es verschiedene Möglichkeiten der Navigation. Es gibt zwei Reset-URLs, eine um die Auswahl im Formular und eine weitere um die Komplemente auf der `Concepts-Seite` zurückzusetzen. Beim Klicken auf einen Knoten im Hasse-Diagramm wird der Begriffsumfang angezeigt. Die Resultat-Tabelle ist nahezu die gleiche wie die in `ShowManyJobResults` und besitzt aus diesem Grund auch ähnliche Navigationsmöglichkeiten. Die Unterschiede sind in Kapitel 6.5.2 beschrieben.

Ein Menü soll unter anderem die `Concepts-Seite` auf der ganzen Seite besser erreichbar machen.

6.6.1 Begriffsverband-Teilmenge

Der aktuelle Begriffsverband beinhaltet alle Formalen Begriffe, die sich aus den gemeinsamen Merkmalen aller Gegenstände des Formalen Kontextes ergeben. Die Begriffsmenge soll sich auf eine Teilmenge einschränken lassen. Zur Auswahl der Teilmenge dient eine Merkmalmenge eines Begriffes.

```
1  -- reduce concepts to concepts with proper subsets of given concept id
2  reduceConceptsToProperSubsets :: (Ord at) => Maybe [Concept ob at] -> ConceptId
3                                -> Maybe [Concept ob at]
4  reduceConceptsToProperSubsets conceptLattice cid =
5      case conceptLattice of
6        Nothing -> Nothing
7        Just concepts' -> do
8          let concept = safeGetIndex concepts' cid
9          case concept of
10           Nothing -> Nothing
11           Just c  -> return $ c:filter (Set.isProperSubsetOf (ats c) . ats)
                 concepts'
```

Quelltext 6.14: FCA/Basic.hs-L76

In der Implementierung `reduceConceptsToProperSubsets` werden alle Begriffe herausgefiltert, deren Merkmale nicht echte Teilmenge der Merkmale des ausgewählten Begriffes sind. Der Begriff selbst soll Teil dieser Teilmenge des Begriffsverbandes bleiben. Abbildung 6.4 begrenzt den Begriffsverband auf Begriffe mit den Merkmalen {YES, AProVE}. An den `Concepts-Handler` wird per Routenparameter (Kapitel 6.4.1) `ConceptId` die ID eines Begriffes übergeben. Der erste Begriff in `[Concept ob at]` beschreibt den allgemeinsten Begriff, mit allen Gegenständen und deren gemeinsamen Merkmalen.

6.6.2 Begriffsverband-Komplement

Um Begriffe aus dem Begriffsverband zu lösen, wurde die Möglichkeit einer Komplementbildung implementiert. Durch die Wahl eines Begriffes werden alle zu diesem gehörenden Gegenstände aus dem Formalen Kontext entfernt. Um mehrere Begriffe aus dem Kontext entfernen zu können, wurde die Möglichkeit eines *chainings*, also dem verketteten Entfernen von Begriffen implementiert.

```haskell
-- recursive function to calculate concepts list and reduce attribute pairs
-- by objects of concept at first index of complement list.
-- Each complement list element refers to the index of the current concepts list
reducePairsByComplements :: [(JobPairID, [Attribute])] -> ComplementIds
                         -> [(JobPairID, [Attribute])]
reducePairsByComplements pairs (Ids complIds) = case complIds of
  []   -> pairs
  compl:compls -> case pairs of
    []         -> []
    _ -> do
      let concept = safeGetIndex (concepts $ contextFromList pairs) compl
      case concept of
        Nothing -> reducePairsByComplements pairs $ Ids compls
        Just c  -> reducePairsByComplements
                   (filterPairsByObjects pairs $ obs c) $ Ids compls
```

Quelltext 6.15: FCA/StarExec.hs-L211

Die rekursive Funktion `reducePairsByComplements` erhält eine Liste von Gegenstand-Merkmal-Paaren `pairs` als Vorstufe eines Formalen Kontextes und Komplement-Ids. Aus `pairs` wird der Formale Kontext und alle Begriffe bestimmt. Für die erste `ConceptId` der Komplement-Ids wird der Begriff via Index und dessen Gegenstände bestimmt. Diese Gegenstände werden aus der Liste von Paaren entfernt und es folgt ein weiterer Aufruf der Funktion mit der neuen Liste `pairs` und den verbleibenden Komplement-Ids. In der `Concepts-Route` werden die Komplement-Ids als Routenparameter übergeben. Die Implementierung der `ComplementIds` in Kapitel 6.4.1 als `Ids [ConceptId]` zeigt den Zusammenhang zwischen Komplementbildung und den Ids der Formalen Begriffen.

Die Abbildung 6.5 zeigt die verbleibenden Begriffe nachdem alle Gegenstände mit den Merkmalen {YES} entfernt wurden.

6.6.3 Menü

Mit Graphviz wurde manuell ein Graph (zu sehen in Abbildung 6.7) erstellt, der die Verlinkung ausgewählter Routen innerhalb von Star-Exec-Presenter aufzeigt. Ein Knoten entspricht einer Route auf Star-Exec-Presenter und eine gerichtete Kante von *Route I* nach *Route II* entspricht einem Link auf *Route I* nach *Route II*. Es ist deutlich die zentrale Position der grau gefärbten Routen für Show Many Job Results und Flexible Table zu sehen. Der Nutzer hat eingeschränkte Möglichkeiten um diese Routen von `HomeR` zu besuchen.

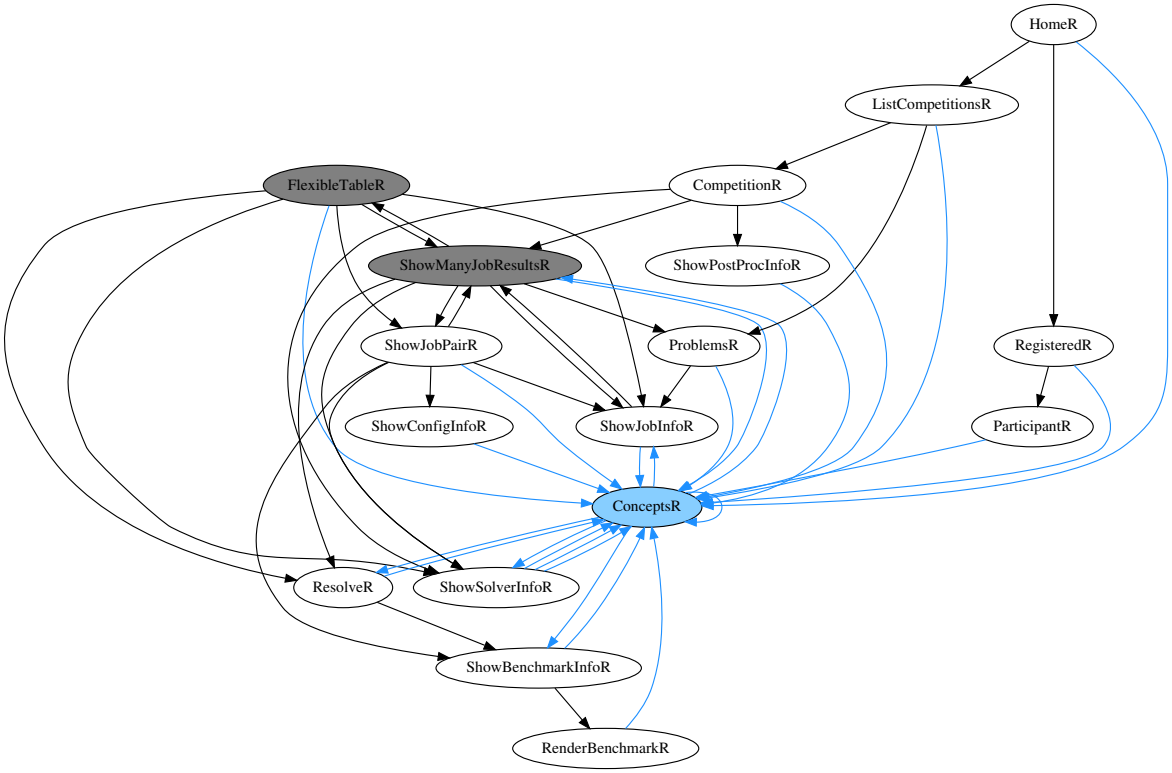


Abbildung 6.7: Star-Exec-Presenter: Navigation nach Optimierung

Die mit blau hervorgehobenen Knoten und Kanten sind durch die Implementierung der `Concepts`-Route und das Hinzufügen eines einfachen Navigationsmenüs (in Quellcode 6.16) entstanden. In der Implementierung des Menü wird zwischen zwei verschiedenen Menü-Elementen im Datentyp `MenuElement` unterschieden. Zum Einen ein einfacher Menü-Eintrag `MenuEntry` und zum Anderen ein Dropdown-Menü `MenuDropdown`. Ein `MenuEntry` besitzt nur einen Namen und eine Route, wohingegen `MenuDropdown` einen Text und eine Liste von `MenuEntry` besitzt. Die Dropdown-Menü-Einträge erlauben eine Gruppierung nach Themen wie „Analyse“ oder „StarExec-Interaktion“. Das Menü hat zur Folge, dass Einträge in diesem von allen Knoten verfügbar sind (im Beispiel `ConceptsR` deutlich in Abbildung 6.7 zu sehen). Die Menü-Einträge für `Competitions` existieren nur aus Gründen der Illustration und wurden nicht implementiert.

```
1  -- navigation bar types
2  data MenuElement = MenuEntry (Text, Route App) | MenuDropdown (Text,
       [MenuElement])
3
4  -- navigation bar elements
5  let menuElements = [MenuEntry ("Home", HomeR)
6                     , MenuDropdown ("Competitions"
7                       , [MenuEntry ("2015", CompetitionR)
8                       , MenuEntry ("2014", CompetitionR)
9                     ])
10                    , MenuDropdown ("Analysis"
11                      , [MenuEntry ("TRS Standard 2015", ConceptsR 0 (Ids [])
                            (JobIds [StarExecJobID 10257]))
12                      , MenuEntry ("TRS Standard 2014", ConceptsR 0 (Ids [])
                            (JobIds [StarExecJobID 5373]))
13                      , MenuEntry ("Example many jobs", ConceptsR 0 (Ids [])
                            (JobIds [StarExecJobID 9515, StarExecJobID 10299]))
14                    ])
15                    , MenuDropdown ("StarExec Interaction"
16                      , [MenuEntry ("Import", ImportR)
17                      , MenuEntry ("Install Solver", InstallSolversR Y2015)
18                      , MenuEntry ("Job Control", ControlR Y2015)
19                    ])]
```

Quelltext 6.16: Foundation.hs

6.7 Optimierungen

Die aktuelle Optimierung kann in verschiedenen Aspekten verbessert werden. Zwei konkrete Optimierungen werden erläutert. In der aktuellen Implementierung wird bereits die Bibliothek `begriff`[9] verwendet, die Optimierungen vorgenommen hat. Diese Optimierungen wurden nicht berücksichtigt.

6.7.1 Datentyp des Begriffsverband

Die Nutzung des Datentyps `[Concept ob at]` ist nicht ideal. Die Liste beinhaltet die einzelnen Formalen Begriffe des Begriffsverbandes, jedoch keine Information über die Beziehung zwischen diesen. Bei der Bildung der Begriffsverband-Teilmengen (Quelltext 6.6.1) und bei Berechnung der Kanten für das Hasse-Diagramm (Quelltext 6.5.1) sind die Beziehungen zwischen den Begriffen hilfreich. Bei der Erstellung des Hasse-Diagramms wird aus der Liste von Begriffen ein Graph aufgebaut. Wenn die Funktion zur Berechnung der Begriffe `concepts` bereits einen Graphen als Repräsentation des Begriffsverbandes als Resultat hätte, könnten verschiedene Berechnungen eingespart werden. Mit einer zusätzlichen Funktion könnten zu einem Knoten alle in Oberbegriff-Relation stehenden Knoten abgefragt werden. Das würde die angesprochenen Probleme optimieren.

6.7.2 Codierung der Merkmale

In der Implementierung der Merkmale `Attribute` (Quelltext 6.2) wird für einige Felder der Datentyp `Text` verwendet. Bei dem Algorithmus zur Bestimmung des Begriffsverbandes `concepts` (Quelltext 6.9) und der Berechnung der Merkmalkombinationen `attributeCombination` (Quelltext 6.8) werden exzessiv Vergleiche der Merkmale und somit Vergleiche von `Text` verwendet. Besser wäre jedes Merkmal durch einen Hash zu ersetzen, um in konstanter Zeit den Vergleich von zwei Merkmalen beantworten zu können.

[9] https://gitlab.imn.htwk-leipzig.de/waldmann/begriff

7 Auswertung

Die Implementierung wird in Bezug auf die Nutzer-Wünsche in Kapitel 4.3 ausgewertet. Eine erneute Befragung der Nutzer und deren Antworten wird vorgestellt. Nach quantifizierbaren Merkmalen der Lösung erfolgt eine Auswertung mit Fokus auf die Implementierung.

7.1 Implementierung

Die in dem Kapitel Spezifikation erarbeiteten Ziele waren:

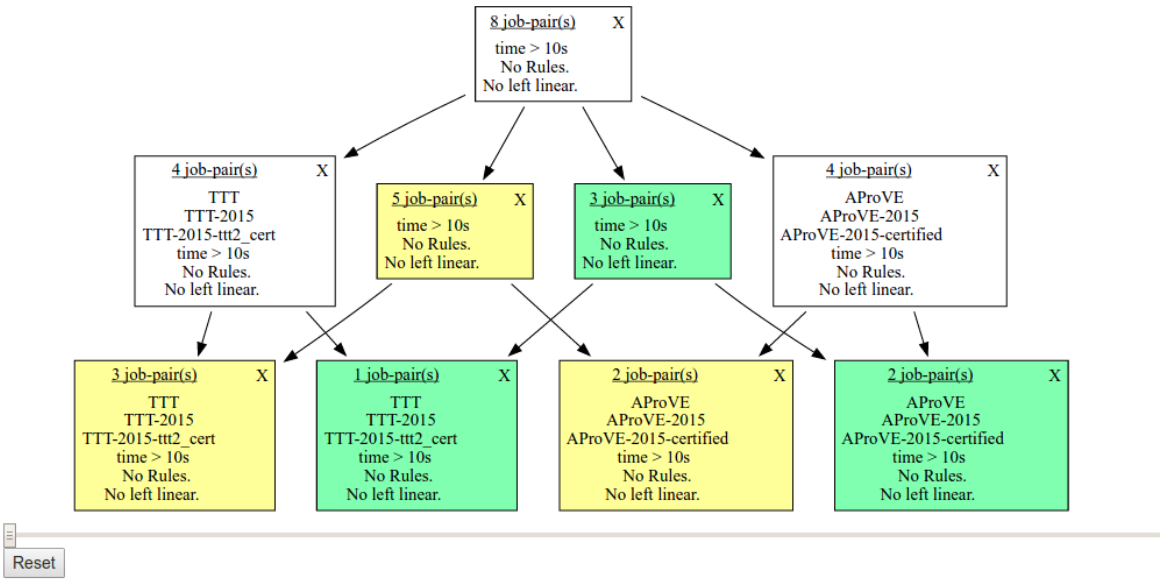
1. Einschränkung der Datenmenge durch Filter
2. mehr Übersichtlichkeit, besserer Zugang zu Informationen
3. eine verbesserte Navigation (z.B.: durch ein Menü)

Durch das Einführen der `Concepts-Route` inklusive `Concepts-Handler` werden dem Nutzer eine weitere Möglichkeit zur Analyse der Resultat-Daten zur Verfügung gestellt. `Flexible Table`, als bestehende Möglichkeit zur Analyse, erlaubt es die Daten nach Zeile und Spalte zu Filtern.

Bei Aufruf der `Concepts-Route` mit bestimmten Routenparametern wird eine oder mehrere Resultat-Tabellen ausgewählt. Die Zellen dieser Resultat-Tabellen, sogenannte `JobPairs` besitzen verschiedene Merkmale. Zur ID eines `JobPairs` werden durch die Konstruktion des Typs `Attribute` als Merkmale abgelegt. In Form von Paaren `[(JobPairID, [Attribute])]` werden diese Informationen verknüpft.

Ein Formular erlaubt es dem Nutzer interessante Merkmale auszuwählen. Alle Paare die keines der ausgewählten Merkmale enthalten, werden entfernt. Aus den verbleibenden Paaren wird durch Formale Begriffsanalyse ein Formaler Kontext erstellt. Der Kontext enthält die `JobPairs` der gewählte(n) Tabelle(n) als Gegenstände und deren `Attribute` als Merkmale.

Aus dem Formalen Kontext wird der Begriffsverband bestimmt. Alle `JobPairs` der Ausgangstabelle werden anhand von gleichen Merkmalen in Formalen Begriffen zusammengefasst. Der Begriffsverband beinhaltet all diese Formalen Begriffe. Ein Hasse-Diagramm visualisiert den Verband und macht die Teilmengen-Beziehung zwischen den Formalen Begriffen deutlich.



Result table

Benchmark	TTT2 ttt2_cert Job StarExecJobID 9515	AProVE 2015 certified Job StarExecJobID 9515
S_Relative/Zantema_06_relative/rel12.xml	. 132.9 / 59.3	. 234.2 / 60.0
SRS_Relative/Mixed_relative_SRS/zr07.xml	YES 24.1 / 6.3	YES 14.7 / 4.8
SRS_Relative/Waldmann_06_relative/r9.xml	. 132.8 / 59.3	YES 70.0 / 9.9
SRS_Relative/ICFP_2010_relative/3450.xml	. 133.5 / 59.6	. 230.9 / 60.1

Abbildung 7.1: Hasse-Diagramm und Resultat-Tabelle zur JobID 9515
http://termcomp.imn.htwk-leipzig.de/concepts/0/Ids%20%5B%5D/9515?_hasdata=&SolverNames=1&SolverNames=2

In Abbildung 7.1 ist zu erkennen, dass sich die acht JobPairs im Ausgangsbegriff in drei YES-Knoten und fünf MAYBE-Knoten aufteilen. In der nächsten Ebene wird ersichtlich, dass *AProVE 2015* zwei Benchmarks und *TTT2* nur einen Benchmark gelöst hat. Die Verteilung der Solver für die MAYBE-Knoten lässt sich analog ablesen.

Vor dieser Masterarbeit gab es in der *Show Many Job Results*-Route nur Aussagen über die erfolgreich gelösten Benchmarks. Wobei ohne die Formale Begriffsanalyse und das Hasse-Diagramm die Information nur kontextlose Zahl dargestellt war. Die kompakte Darstellung und Teilmengen-Beziehung zwischen den Begriffsknoten im Hasse-Diagramm transportiert die Informationen einfacher zum Anwender.

Durch einen Klick auf die Knoten zeigt die Resultat-Tabelle nur die *JobPair*, die die Merkmale des Knotens besitzen. Demnach ist der erste Punkt der Anforderungen - das Einschränken der Datenmenge - erfüllt. Mit dem Hasse-Diagramm wird ein Überblick über die verfügbaren Merkmale und deren Beziehungen visualisiert. Der Nutzer kann vom ersten Knoten mit allen gemeinsamen Merk-

malen im aktuellen Formalen Kontext den gerichteten Kanten bis zur gesuchten Merkmalmenge folgen. Bei einer kleinen Menge an Begriffsknoten ist der zweite Punkt der Anforderungen erfüllt. Mit dem Zoom-Plugin `jQuery Panzoom` können auch Informationen von größeren Begriffsverbänden zum Nutzer transportiert werden. Die zwei Links innerhalb eines Knotens ermöglichen es dem Nutzer die Menge der Begriffe im Begriffsverband zu reduzieren und die gesuchte Information zu finden. Die Einführung eines Menüs erfüllt den dritten Punkt der Anforderung. Demnach sind alle Anforderungen erfüllt.

7.2 Interview

Um die Seite gegenüber den Nutzer zu evaluieren, wurden diese erneut befragt. Das Interview bestand aus den folgenden Fragen:

1. Ist die derzeitige Implementierung hilfreich um die Tabelle zu filtern?
2. Können Sie sich vorstellen, das darüber für Sie interessante Datenmengen gefunden werden können?
3. Was stört Sie an der aktuellen Implementierung?
4. Welche Verbesserungen fallen Ihnen sofort ein?

Antworten:

1. 2x Ja, 1x Nein
2. 1x Ja, 1x Vielleicht, 1x Nein
3. a) Vergleich von Ergebnissen verschiedener Jahre
   b) Benchmark-Merkmale für nicht TRS-Competitions uninteressant
   c) Bedeutung von "CPU times" unklar
4. a) selbstdefinierte Auswahlkriterien (warum ausgerechnet $<= 10s$ und nicht $<= 42s$?)
   b) Erweiterung auf andere Competitions z.B. SMT.
   c) Vergleich zu selbst gestarteten Jobs

Die Antworten deuten auf einen insgesamt positiven Eindruck. Die neue Route hilft beim Filtern und vor allem Benchmark-Merkmale für andere Competitions sind von Interesse.

7.3 Quantifizierbare Merkmale

Eine Zeitmessung, die Übersicht über die geschriebenen Haskell-Module und Code-Metriken fassen die Implementierung in messbaren Größen und Zahlenwerten zusammen.

Zeit für Berechnung Begriffsverband

Als quantifizierbares Merkmal der Implementierung soll eine Zeitmessung dienen. Es wird die Zeitdauer für die Berechnung des Begriffsverbandes einschließlich der Konstruktion des Hasse-Diagramms gemessen. Zur Messung wird die neue `Concepts`-Route für eine JobID aufgerufen und gewartet bis auf dem Terminal des lokal laufenden Servers der erfolgreiche Response vermeldet wird. Zur Messung wurde Revision 92b031d von Star-Exec-Presenter verwendet, einschließlich Optimierung durch die Bibliothek begriff[1].

Die folgende Hard- und Software wurde für die Zeitmessung genutzt.

Hardware:

- Prozessor: Intel Core i5
- Prozessor-Geschwindigkeit: 2,6 GHz
- Anzahl Kerne: 2
- RAM: 16 GB

Software:

- Betriebssystem: OSX 10.10.5
- Graphviz 2.36.0
- GHC 7.10.3
- Yesod 1.4.3
- Google Chrome 51.0.2704.103 (64-bit)

Die Tabelle 7.1 zeigt die Ergebnisse der Messung. Die Berechnung des Begriffsverbandes zur JobID 10299 mit einer Anzahl von 95 Formalen Begriffen hat beispielsweise 1,04 Sekunden gedauert. Für jede JobID wurden fünf Messwerte aufgenommen und zur Berechnung der ∅-Zeit wurde das arithmetische Mittel dieser gebildet. Die Werte der Einzelmessungen befinden sich im Anhang 3.

JobID	|Gegenstände G|	|Merkmale M|	|Relationen R|	|B(G,M,R)|	∅-Zeit in Sekunden
9515	8	13	56	9	0.08
10299	820	24	5740	95	1,04
10257	10486	37	73402	658	17,05

Tabelle 7.1: Messwerte für die Jobs 9515, 10299 und 10257

[1] https://gitlab.imn.htwk-leipzig.de/waldmann/begriff

Haskell-Module

Die Umsetzung der Formalen Begriffsanalyse während der Masterarbeit brachte die folgenden sechs Haskell-Module hervor:

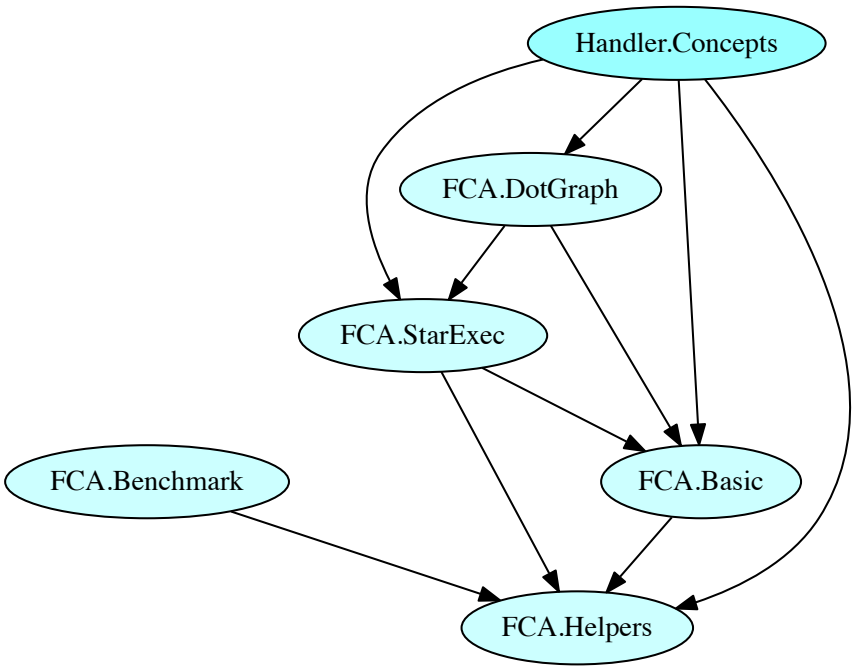


Abbildung 7.2: Abhängigkeiten der implementierten Haskell-Module

Der Graph in Abbildung 7.2 zeigt die Abhängigkeiten der Haskell-Module untereinander. Wenn *Modul1* von *Modul2* verwendet wurde, existiert eine Kante von *Modul2* zu *Modul1*. Zur Erstellung des Graphen wurde das graphmod[2] verwendet[3].

Code-Metriken

Ein weiteres quantifizierbares Merkmal in der Softwareentwicklung sind Code-Metriken. Die Tabelle 7.2 zeigt die Verteilung von LOC, Commits und Editierungen von Dateien auf die Entwickler von Star-Exec-Presenter. Die Original-Tabelle im Anhang 4 wurde mit `git-fame`[4] erstellt[5] und aufgrund mehrerer Nutzer mit unterschiedlicher Namen aggregiert. Das Verzeichnis static/ wurde ausgeschlossen, da das Einfügen von statischen Dateien der Tools `Bootstrap`, jQuery, `jQuery Panzoom` und `normalize.css` nur die LOC-Statistik verfälscht. Die Tabelle zeigt das 1580 LOC

[2] https://github.com/yav/graphmod
[3] find . -name '*.hs' | xargs graphmod -q –no-cluster | dot -Tpdf > concepts.pdf
[4] https://github.com/oleander/git-fame-rb
[5] git fame –exclude static/

(12,8 % Anteil an den gesamten LOC), 215 Commits und 75 Datei-Editierungen durch diese Arbeit entstanden sind.

Name	LOC	Commits	Dateien	Verteilung
Stefan von der Krone	6709	237	138	54,3/33,3/81,7
Johannes Waldmann	4078	260	93	33,0/36,5/39,6
René Muhl	1580	215	75	12,8/30,2/44,4

Tabelle 7.2: LOC, Commits und Datei-Editierungen der Entwickler von Star-Exec-Presenter

7.4 Resultat

Die erarbeiteten Anforderungen aus dem Kapitel 4.3 wurden durch die Implementierung (Kapitel 6) erfüllt. Die Nutzer nehmen die neue Seite positiv auf und entgegneten mit weiteren Wünschen. Die Zeitmessung zeigt, dass die aktuelle Implementierung die Begriffsverbände für große *Jobs* wie die Competition *TRS Standard* (JobID 10257) noch nicht schnell genug berechnet. Im Kapitel 6.7 wurden Optimierungen angegeben, die die Zeit bereits verbessern sollte. Die Implementierung besteht aus sechs Haskell-Modulen und wurde mit 1580 LOC und 215 Commits zu Star-Exec-Presenter hinzugefügt.

8 Zusammenfassung

Nach einer Einleitung in das Thema durch eine Motivation werden im Kapitel Termination-Competition beteiligte Komponenten und die Daten untersucht. Die Einführung in die Formale Begriffsanalyse erhält der Leser in Kapitel 3 nachdem grundlegende Mathematik der Ordnungs- und Verbandstheorie vorgestellt wurde. Im 4. Kapitel wird ein Interview mit Nutzern von Star-Exec-Presenter in Form einer Gegenüberstellung von Fragen und Antworten ausgeführt. Aus den Antworten und einer Betrachtung der aktuellen Möglichkeiten für den Nutzer auf Star-Exec-Presenter werden im Kapitel 4.3 Ziele erarbeitet. Bevor die Implementierung erläutert wird, gibt es eine Übersicht über die Programmiersprache Haskell und das Web-Framework Yesod. In Kapitel 6 wird ein Entwurf, Entscheidungen bezüglich der Implementierung im Allgemeinen und der Nutzeroberfläche und die Implementierung an Code-Beispielen erläutert. Optimierungen in Kapitel 6.7 leiten in die Auswertung in Kapitel 7 über. Eine Untersuchung der Implementierung auf die gestellten Anforderungen an die Arbeit mit anschließender Erhebung von quantifizierbaren Merkmalen schließen die Arbeit ab.

8.1 Einschätzung

Durch die Erweiterung von Star-Exec-Presenter durch Formaler Begriffsanalyse wurde die Applikation verbessert und das Ziel erreicht. Der Nutzer erhält auf der Seite `Concepts` eine neuartige Möglichkeit zur Analyse und Visualisierung der Resultatdaten der Termination-Competition.

Die aktuelle Implementierung erlaubt das Filtern mit bekannten und neuen Merkmalen der `JobPairs`. Durch die Einführung von Benchmark-Merkmalen wie *Anzahl der Regeln* und *Links-Linearität* wird eine weitere Dimension des Filtern eröffnet.

Die Visualisierung des Begriffsverbandes durch ein Hasse-Diagramm ermöglicht den Einblick in die Struktur der Daten und veranschaulicht die Beziehungen zwischen Begriffen und deren Merkmalen. Das erleichert die Navigation und Analyse der Daten.

Die Vielzahl an Daten erschwert die Visualisierung und ein Überblick über das gesamte Hasse-Diagramm ist nicht immer gewährleistet. Nachteil der aktuellen Implementierung ist die lange Zeitdauer für die Berechnung von großen Begriffsverbänden. Durch weitere Optimierungen bezüglich der Algorithmik und dem Verwenden anderer Technologien sind Verbesserungen möglich.

Die Formale Begriffsanalyse eignet sich zur Analyse der Resultatdaten der Termination-Competition.

8.2 Ausblick

Nachdem die `Concepts`-Seite intensiver durch die Nutzer von Star-Exec-Presenter verwendet wurde, ergeben sich wieder neue Anforderungen an diese. In den Nutzerantworten zur Evaluation der Implementierung in Kapitel 7.2 sind bereits erste Forderungen zu finden.

Mit dem Haskell-Paket tpdb ist es bereits jetzt möglich weitere Benchmark-Merkmale[1] zu berechnen. Die Einführung weiterer Merkmale in die aktuelle Implementierung erlaubt eine noch gezieltere Suche nach interessanten Teilmengen in den Resultat-Daten. Jedoch bedeuten weitere Merkmale eine weitere Verlangsamung der Berechnung und macht eine Optimierung des Algorithmus unerlässlich.

Das Haskell-Modul der Formalen Begriffsnalyse ist allgemein formuliert und erlaubt eine Nutzung in Star-Exec-Presenter. Mit anderen Merkmalen könnten Kategorie-spezifische Analysen und Visualisierungen entstehen. Denkbar ist eine Visualisierung der Kategorie über mehrere Competition-Jahre und die Anzahl an gelösten Benchmarks pro Solver.

Nach der Termination-Competition 2016 im September 2016[2] wird es neue Resultatdaten geben, die analysiert werden müssen.

[1] https://hackage.haskell.org/package/tpdb-1.3.3/docs/TPDB-Data-Attributes.html
[2] http://termination-portal.org/wiki/Termination_and_Complexity_Competition_2016

Glossar

Algebraischer Datentyp ist ein Haskell-Datentyp, der die Kombination von Typen mittels algebraischer Operationen Summe und Produkt erlaubt (siehe Kapitel 5).

Begriffsverband ist ein Verband in dem sich alle Formalen Begiffe eines Formalen Kontextes befinden (Kapitel 3.5.3).

Benchmark bezeichnet ein Problem der Termination Problems Data Base.

Competition ist die Bezeichnung für einen Job, der an der jählichen Termination-Competition teilnimmt.

DOT ist eine Sprache zur Beschreibung von Graphen für das Graphviz-Programm dot[3].

Formale Begriffsanalyse beschreibt eine Methode des Data-Minings (Kapitel 3).

Formaler Begriff ist ein grundlegender Begriff der Formalen Begriffsanalyse (Kapitel 3.5).

Formaler Kontext ist ein grundlegender Begriff der Formalen Begriffsanalyse (Kapitel 3.3).

GET HTTP-Methode zum Lesen von Web-Inhalten.

GHC (Glasgow Haskell Compiler) ist ein Compiler für Haskell[4].

Haskell funktionale Programmiersprache (Kapitel 5).

Hasse-Diagramm eine kompakte Darstellungform für Ordnungen (Kapitel 3.2).

Job beschreibt auf StarExec eine Kombination aus einer Menge Benchmarks und einer Menge Solver.

JobPair bezeichnet auf StarExec das Resultat eines Solvers bezüglich eines Benchmarks.

jQuery ist eine schnelle, kleine Javascript-Bibliothek die viele Funktionen unter anderem zum Traversieren und Manipulieren von HTML-Dokumenten besitzt. [5].

LOC (Lines Of Code) engl. für Anzahl der Zeilen eines Quelltextes.

Query Query engl. für eine Abfrage, die an ein System gestellt werden kann z.b. Datenbank via SQL.

[3] http://www.graphviz.org/Documentation/dotguide.pdf
[4] https://www.haskell.org/ghc/
[5] https://api.jquery.com/

Record-Typ ist ein Haskell-Datentyp mit benannten Feldern (siehe Kapitel 5).

SMT (Satisfiability Modulo Theories) „überprüft die Erfüllbarkeit von logischen Formeln bezüglich einer oder mehrerer Theorien“[6].

Solver bezeichnet ein Problem lösendes Computerprogramm (Kapitel 2.2).

SQL (Structured Query Language) ist eine Datenbank-Abfragesprache.

Star-Exec-Presenter ist eine in dem Haskell Web-Framework geschriebene Web-Applikation zur Visualisierung und Analyse von StarExec (Kapitel 2.4).

StarExec ein Service der University of Iowa und der University of Miami (Kapitel 2.3).

SVG (Scalable Vector Graphics), engl. für skalierbare Vektorgrafik.

Termination beschreibt das (An-)Halten eines Programmes auf eine Eingabe nach einer endlichen Anzahl von Arbeitsschritten.

Termination Problems Data Base ist eine Sammlung von Problemen die während der Termination-Competition gelöst werden (Kapitel 2.1).

Termination-Competition jährlich stattfindender Wettbewerk (Kapitel 2).

TPDB siehe Termination Problems Data Base.

TRS (Term Rewriting System) engl. für Termersetzungssystem.

URL (Uniform Resource Locator) engl. für einheitlicher Ressourcenzeiger ist die eindeutige Bezeichnung einer Resource in einem Netzwerk (häufige Anwendung im Web)[7].

XML (eXtensible Markup Language) ist eine generische vom W3C (World Wide Web Consortium)[8] spezifizierte Sprache.

Yesod Web Framework ist ein in Haskell von Michael Snoyman inital geschriebenes Web-Framework (Kapitel 5.2).

[6] http://research.microsoft.com/en-us/um/people/leonardo/sbmf09.pdf
[7] https://tools.ietf.org/html/rfc3986#section-1.1.3
[8] https://www.w3.org/

Literaturverzeichnis

[Gan13] Bernhard Ganter. *Diskrete Mathematik:Geordnete Mengen*. Springer-Verlag, 2013. ISBN: 978-3-642-37499-9.

[Gie+15] Jürgen Giesl u. a. „Termination Competition (termCOMP 2015)“. In: *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*. 2015, S. 105–108. DOI: 10.1007/978-3-319-21401-6_6. URL: http://dx.doi.org/10.1007/978-3-319-21401-6_6.

[GW] Bernhard Ganter und Rudolf Wille. *Applied Lattice Theory: Formal Concept Analysis*. URL: https://tu-dresden.de/Members/bernhard.ganter/fca/concept.pdf (besucht am 14. 07. 2016).

[GW96] Bernhard Ganter und Rudolf Wille. *Formale Begriffsanalyse - mathematische Grundlagen*. Springer, 1996. ISBN: 978-3-540-60868-4.

[Hac08] Dirk Hachenberger. *Mathematik für Informatiker*. Pearson Studium, 2008. ISBN: 3827373204.

[Hud+07] Paul Hudak u. a. „A history of Haskell: being lazy with class“. In: *Proceedings of the Third ACM SIGPLAN History of Programming Languages Conference (HOPL-III), San Diego, California, USA, 9-10 June 2007*. 2007, S. 1–55. DOI: 10.1145/1238844.1238856. URL: http://doi.acm.org/10.1145/1238844.1238856.

[IL14] Sebastian Iwanowski und Rainer Lang. *Diskrete Mathematik mit Grundlagen: Lehrbuch für Studierende von MINT-Fächern (German Edition)*. Springer Vieweg, 2014. ISBN: 978-3-658-07130-1.

[Kla+16] Steve Klabnik u. a. *The Rust Reference: 13 Appendix: Influences*. 2016. URL: https://doc.rust-lang.org/reference.html#appendix-influences (besucht am 04. 08. 2016).

[Mar10] Simon Marlow. *Haskell 2010 Language Report*. 2010. URL: https://www.haskell.org/definition/haskell2010.pdf (besucht am 04. 08. 2016).

[Mic13] Greg Michaelson. „*Learn You a Haskell for Great Good! A Beginner's Guide*, by Miran Lipovaca, No Starch Press, April 2011, ISBN-10: 1593272839; ISBN-13: 978-1593272838, 376 pp“. In: *J. Funct. Program.* 23.3 (2013), S. 351–352. DOI: 10.1017/S095679681300004X. URL: http://dx.doi.org/10.1017/S095679681300004X.

[Muh15] René Muhl. *Eine Haskell Web-Applikation zur Analyse und Visualisierung formaler Konzepte*. Techn. Ber. HTWK Leipzig, Juli 2015.

[MZ07] Claude Marché und Hans Zantema. „The Termination Competition“. In: *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*. 2007, S. 303–313. DOI: 10.1007/978-3-540-73449-9_23. URL: http://dx.doi.org/10.1007/978-3-540-73449-9_23.

[OGS08] Bryan O'Sullivan, John Goerzen und Don Stewart. *Real world Haskell - code you can believe in*. O'Reilly, 2008. ISBN: 978-0-596-51498-3. URL: http://book.realworldhaskell.org/.

[Pet08] Wiebke Petersen. „Zur Minimalität von Pāṇinis Śivasūtras : eine Untersuchung mit Methoden der formalen Begriffsanalyse". Diss. Heinrich-Heine-Universität Düsseldorf, 2008. URL: http://d-nb.info/1007126175 (besucht am 14. 07. 2016).

[Rub14] Albert Rubio. *Termination Competition Problem Selection Algorithm*. 2014. URL: http://termination-portal.org/mediawiki/index.php?title=Termination_Competition_Problem_Selection_Algorithm&oldid=1407 (besucht am 04. 08. 2016).

[Sch11] Sibylle Schwarz. *Wissensmanagement WS11/12 - Formale Begriffsanalyse (FCA)*. 2011. URL: http://whz-cms-10.zw.fh-zwickau.de/sibsc/lehre/ws11/wms/wms11_fca.pdf (besucht am 04. 08. 2016).

[Sno15] Michael Snoyman. *Developing Web Apps with Haskell and Yesod - Safety-Driven Web Development, Second Edition*. O'Reilly, 2015. ISBN: 978-1-491-91559-2. URL: http://www.oreilly.de/catalog/9781491915592/index.html.

[SST15] Aaron Stump, Geoff Sutcliffe und Cesare Tinelli. *StarExec*. 2015. URL: https://www.starexec.org/starexec/public/about.jsp (besucht am 04. 08. 2016).

[Wal15a] Johannes Waldmann. *Annual International Termination Competition*. 2015. URL: http://termination-portal.org/mediawiki/index.php?title=Termination_Competition&oldid=1620#Annual_International_Termination_Competition (besucht am 04. 08. 2016).

[Wal15b] Johannes Waldmann. *Competition Procedure*. 2015. URL: http://termination-portal.org/mediawiki/index.php?title=Termination_Competition_2015&oldid=1636#Competition_Procedure (besucht am 04. 08. 2016).

Anhang

1 Anforderungen an die Arbeit: Interview-Fragen

allgemein

- Für was nutzen Sie Star-Exec-Presenter?
  - Betrachtung der Termination-Competition-Ergebnisse
  - Analyse der Resultate einzelner Kategorien
  - Sonstiges
- Was ist die von Ihnen am meisten genutzte Route? (Angabe von Bsp.-URL hilfreich)
- Skizzieren Sie einen typischen Besuch der Seite bis zur gesuchten Informationen.
- Welche guten/schlechten Eigenschaften hat das query interface http://nfa.imn.htwk-leipzig.de/termcomp-devel/results/standard/Query%20%5BFilter_Rows%20%28And%20%5BEquals%20%22nothing%22,Equals%20%22solver-maybe%22,Equals%20%22solver-maybe%22,Equals%20%22solver-maybe%22,Equals%20%22solver-no%22,Equals%20%22solver-maybe%22,Equals%20%22solver-maybe%22%5D%29%5D/10296 ?
- Was sind die Probleme mit der derzeitigen Darstellung?
- Welche Darstellungsformen, wuerden bei der
  - Visualisierung
  - Analyse
  - Navigation zwischen den Seiten, den Analyse-Daten helfen?
- Welche Informationen werden benötigt und sind derzeit nicht in star-exec-presenter eingebaut?
- Welche Features/Funktionen wären hilfreich, um konkurrierende Solver besser zu analysieren?
- Was sind interessante Merkmale eines Job-Pairs?
- Gibt es Merkmale eines JobPairs, die wichtiger sind als andere? Kann man eindeutige Prioritäten vergeben?
  - Solvername
  - Solver-Konfiguration
  - Solver-Resultat (YES, NO, MAYBE)
  - CPU-Zeit

  - Wallclock-Zeit
  - Informationen ueber den gelösten Benchmark

konkrete Ansätze

- Sollte es eine Navigationsleiste/Menu geben?
- Sollten die Tabellenspalten sortierbar sein?
- Sollte es einen Filter fuer die Job-Tabelle geben?
- Sollte es eine Suche fuer Job, JobPair, Solver, Benchmark geben?
- Wuerde Pagination auf den Seiten helfen? Wieviele Zeilen sollte pro Seite angezeigt werden?
  - 10
  - 20
  - 50
- Sollte es eine zusammenfassende Tabelle zu einem Job geben? Was sollte diese beinhalten?
- Sollte es mehr Informationen zu einem Benchmark geben?
- Sollte es einen Vergleich von
  - Benchmarks
  - Solvern (+ Config)
  - Termination-Competition geben?
- Wenn ja, unter welchen Kriterien sollten diese verglichen werden?
- Sollten die Resultate zu einem Job exportierbar sein?
  - Wenn ja welches Format wird bevorzugt? CSV, XML, JSON, andere?

2 Anforderungen an die Arbeit: Interview-Antworten

Nutzung SEP

- Betrachtung Competition-Ergebnisse
- Analyse der einzelnen Competition-Kategorien
- Beobachtung der Tool-Entwicklung
- Identifikation von YES/NO-Konflikten (gut gelöst)
- Regression: Vergleich mit anderen Versionen des Tools
- Vergleich mit anderen Tools

Benutzte Routen

- Competition-Übersicht
- Flexible Table für drill down in Kombinationen
- Competition-Ergebnis, Job-Resultate, Flexible Table nicht

Gesuchte Information auf SEP

- Benchmarks, bei denen sich das eigene Tool von Anderen unterscheidet
- Ort von Flexible Table
- Übersichtstabellen (Job-Result?)
- Suche nach unterschiedlichem Result (YES, MAYBE) von versch. Solver-Versionen
- Wieviele der Benchmarks werden vom eigenen Solver geschafft. YES/(Anz. Benchmarks)

Eigenschaften Flexible-Table Query-Interface

- nicht bekannt, dass man eigene Queries absenden kann
  - (fehlende Dokumenation für die Features der Seite)
- vorhandene Funktionalität ist gut
- "solver-yes" sollte eher nur "yes" heißen
- In langen Tabellen wuerden Zwischenueberschriften / 'mit-laufende tabellen header' sehr helfen (s.u. 2.Link)
- Nutzungsmöglichkeiten unbekannt
- Fetchen der Informationen verwirrt, Nutzer weiß nicht, das man nur Warten muss

gewünschte Features

- Inhalt des Benchmarks
- C-Programm? XML? (s.u. 1.Link)
- direkte Darstellung von Prover-Inputs (fuer text-only formate) wäre schoen
- komplette Ausgabe des Tools (Solvers?)
- weitere Eigenschaften (Größe, Anzahl der Zeilen) eines Benchmarks

- Sortierung nach Benchmark-Eigenschaften
- Summe der Laufzeit pro Solver für die Benchmarks, die von allen Solvern gelöst wurden
- ein "Show examples with |tool1.runtime - tool2.runtime| $>$ k"-Feature
- Übersichtstabellen mit Durchschnittszeiten
- Plots: Scatterplots, Cactusplots, ... (z.B. Scatterplot in den man reinzoomen kann, und wo die Punkte ein mouseover haben, das Details anzeigt)
- Übersicht proofs
- Übersicht selbst abgefeuerte Queries/interessante Resultatmengen
- Übersichtlichkeit und intuitive Bedienbarkeit von (s.u. 3.Link) war gut

Navigationsleiste

- keine Präferenz
- 2x ja

Sortierung für Tabellenspalten

- 3x ja

Filter für Tabellen

- 3x ja

Suche für Job, JobPair, Solver, Benchmark

- keine Notwendigkeit
- nein

Pagination

- 2x nein
- keine bzw. default alle Ergebnisse anzeigen

Zusammenfassende Tabelle pro Job

- Anzahl (von was?)
- Count YES/NO/MAYBE/ERROR
- 2x avg. time for YES/NO/MAYBE/ERROR (s.u. 3.Link)
- à la (s.u. 3.Link)

Zusätzliche Informationen zu einem Benchmark

- Dateigröße und/oder Anzahl Zeilen

Vergleich von Benchmarks/Solver/Competition

- ja
- Vergleichskriterien: Anzahl gelöste Benchmarks

- nein, ja, ja

Export für Star-Exec-Presenter

- StarExec CSV-Export ausreichend
- CSV-Export gewünscht

Links

1. http://sv-comp.sosy-lab.org/2016/results/results-verified/Termination.table.html
2. http://stackoverflow.com/questions/17584702/how-to-add-a-scrollbar-to-an-html5-table
3. http://termcomp.uibk.ac.at/termcomp/competition/categoryResults.seam?cat=10235&comp=15991&cid=368094
4. http://cl-informatik.uibk.ac.at/software/tct/experiments/tct2/RaML/index.php#details

3 Auswertung: Einzelmesswerte Begriffsverband

JobID	Messwert 1	Messwert 2	Messwert 3	Messwert 4	Messwert 5	∅-Zeit in Sekunden
9515	0,09	0,07	0,08	0,06	0,09	0,08
10299	1,01	1,06	1,04	1,03	1,06	1,04
10257	16,66	17,31	17,33	17,25	16,70	17,05

Tabelle 1: Einzel-Messwerte für die Jobs 9515, 10299 und 10257 (alle Messwerte inklusive ∅-Zeit in Sekunden)

4 Auswertung: Code-Metriken Star-Exec-Presenter

```
Total number of files: 169
Total number of lines: 12,367
Total number of commits: 712
+--------------------------+-------+---------+-------+--------------------+
| name                     | loc   | commits | files | distribution       |
+--------------------------+-------+---------+-------+--------------------+
| Stefan von der Krone     | 5,872 | 212     | 108   | 47.5 / 29.8 / 63.9 |
| Johannes Waldmann        | 4,078 | 260     | 93    | 33.0 / 36.5 / 55.0 |
| René Muhl                | 1,467 | 182     | 67    | 11.9 / 25.6 / 39.6 |
| stefanvonderkrone@gmx.de | 777   | 17      | 27    | 6.3 / 2.4 / 16.0   |
| rm--                     | 113   | 33      | 8     | 0.9 / 4.6 / 4.7    |
| vagrant                  | 60    | 7       | 3     | 0.5 / 1.0 / 1.8    |
| stefanvonderkrone        | 0     | 1       | 0     | 0.0 / 0.1 / 0.0    |
+--------------------------+-------+---------+-------+--------------------+
```

Eigenständigkeitserklärung

Familienname, Vorname	Ort, Datum
Geburtsdatum	Studiengruppe / WS/SS

Erklärung

Gemäß §40 Abs. 1 i. V. m. §31 Abs. 7 RaPO

Hiermit erkläre ich, dass ich die Masterarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

.....

Unterschrift