Befehlsauswahl auf expliziten Abhängigkeitsgraphen

Sebastian Buchwald, Andreas Zwinkau

Abschlussvortrag Diplomarbeit

Betreuer:

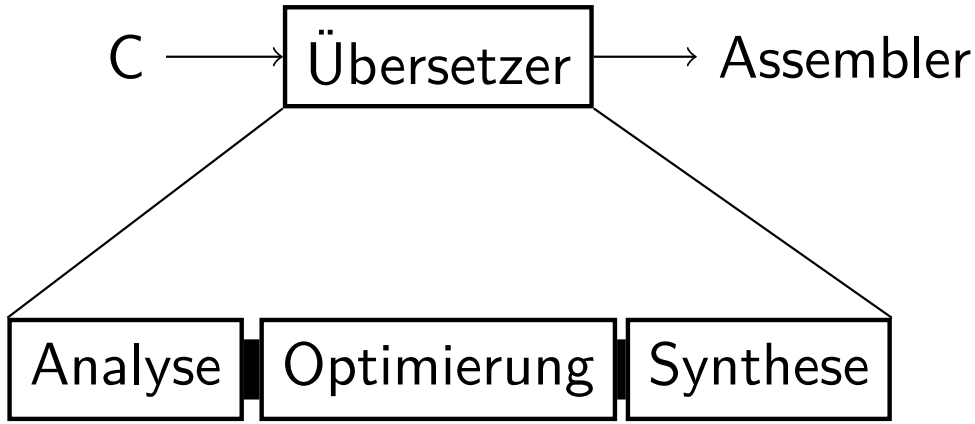
Dipl.-Inf. Michael Beck
Prof. em. Dr. Dr. h.c. Gerhard Goos

13. Februar 2009

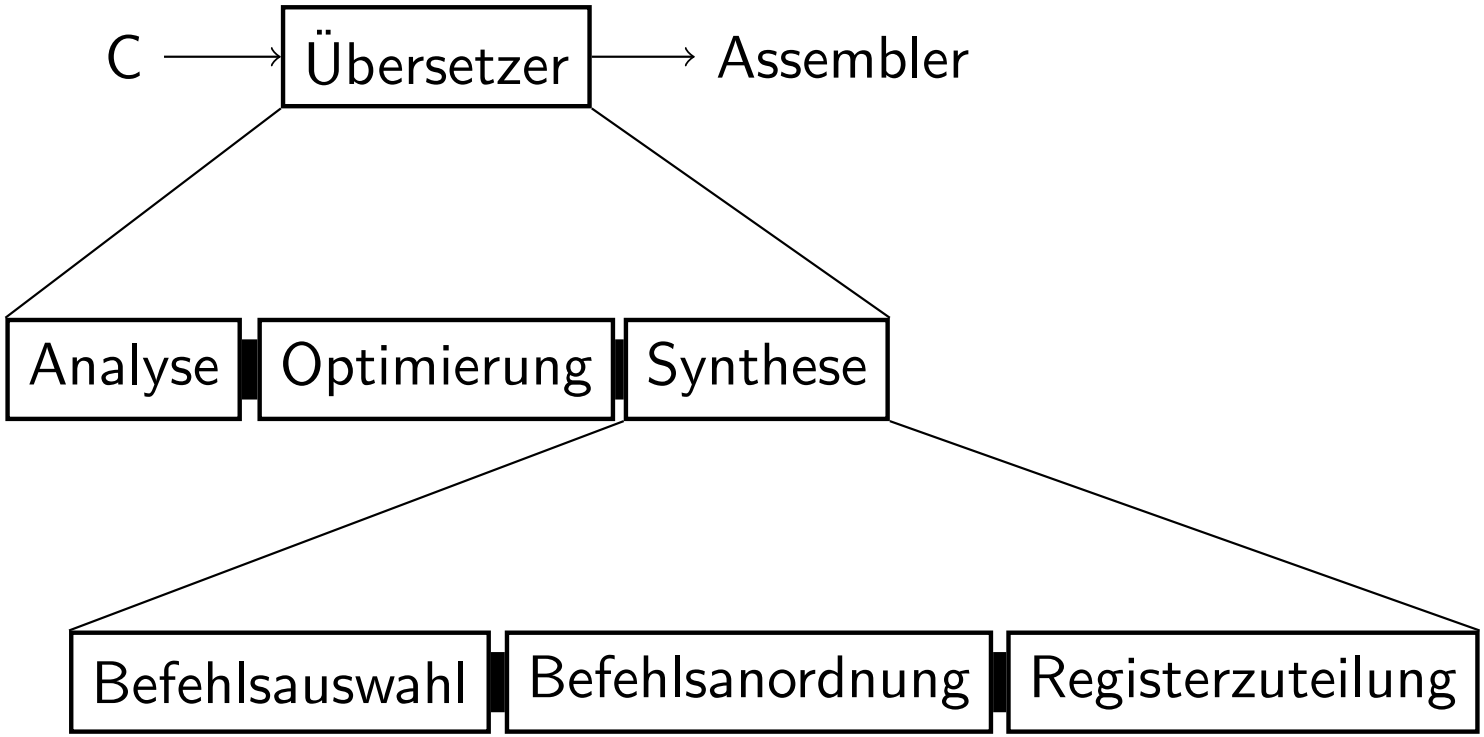
Übersicht

Übersetzer

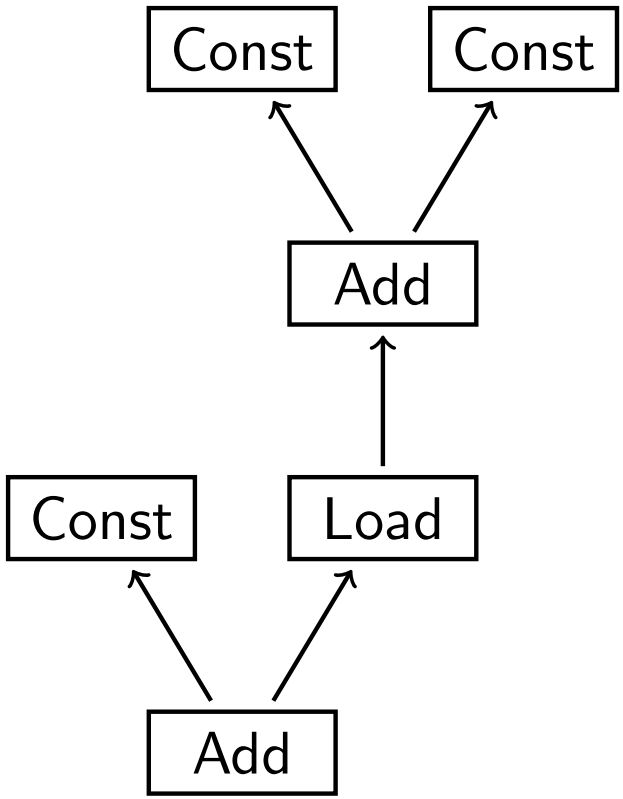
Übersetzer



Übersetzer

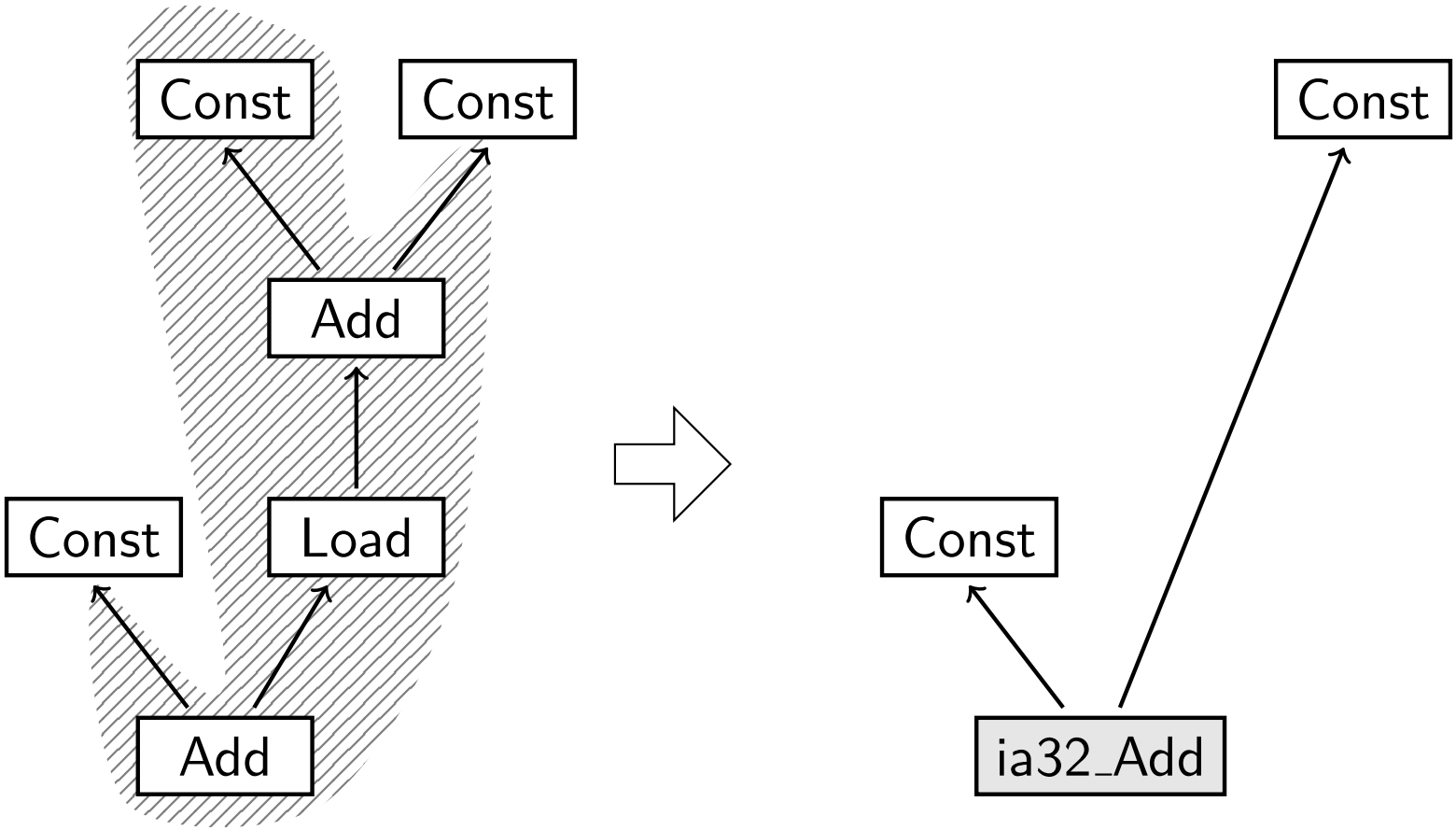

C
Übersetzer
Assembler
Analyse
Optimierung
Synthese
Befehlsauswahl
Befehlsanordnung
Registerzuteilung

Explizite Abhängigkeitsgraphen

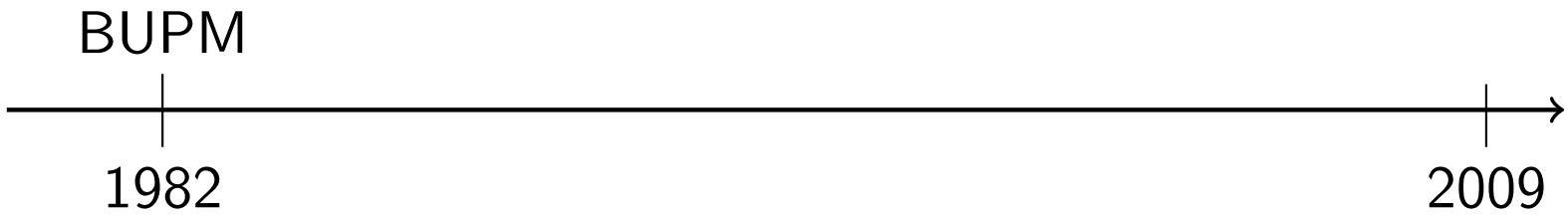


- Knoten repräsentieren Werte
- Kanten repräsentieren Abhängigkeiten
- SSA-Form (Statische Einmalzuweisung)

Befehlsauswahl

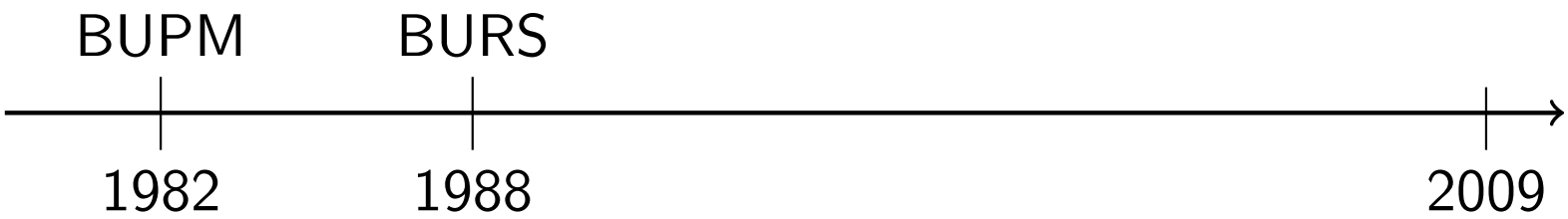


Generierte Befehlsauswahl



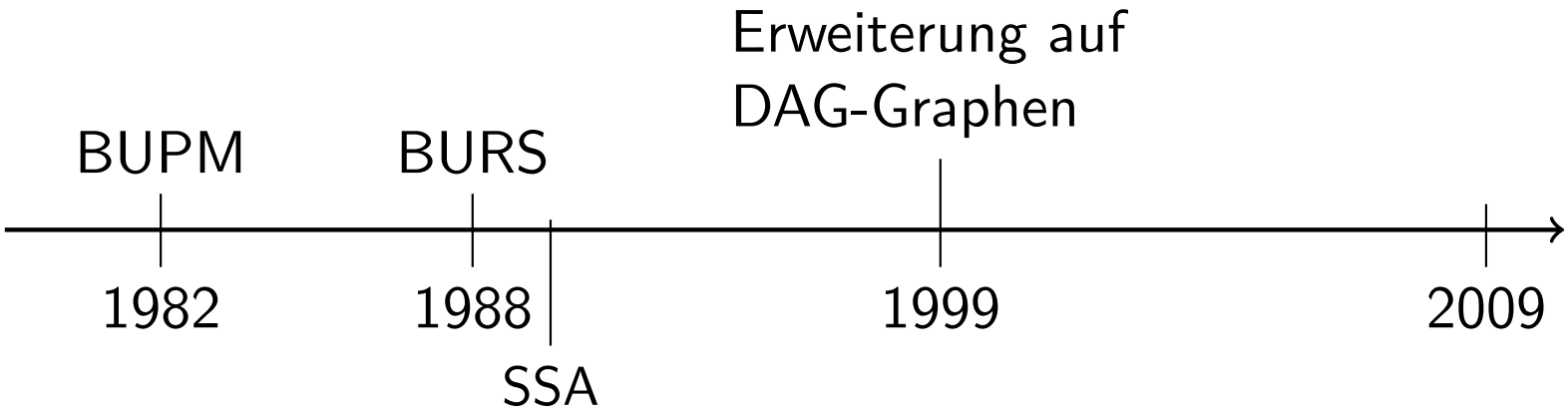
- Baum-Programmgraphen
- Baum-Muster (Spezifikation durch GTES)

Generierte Befehlsauswahl



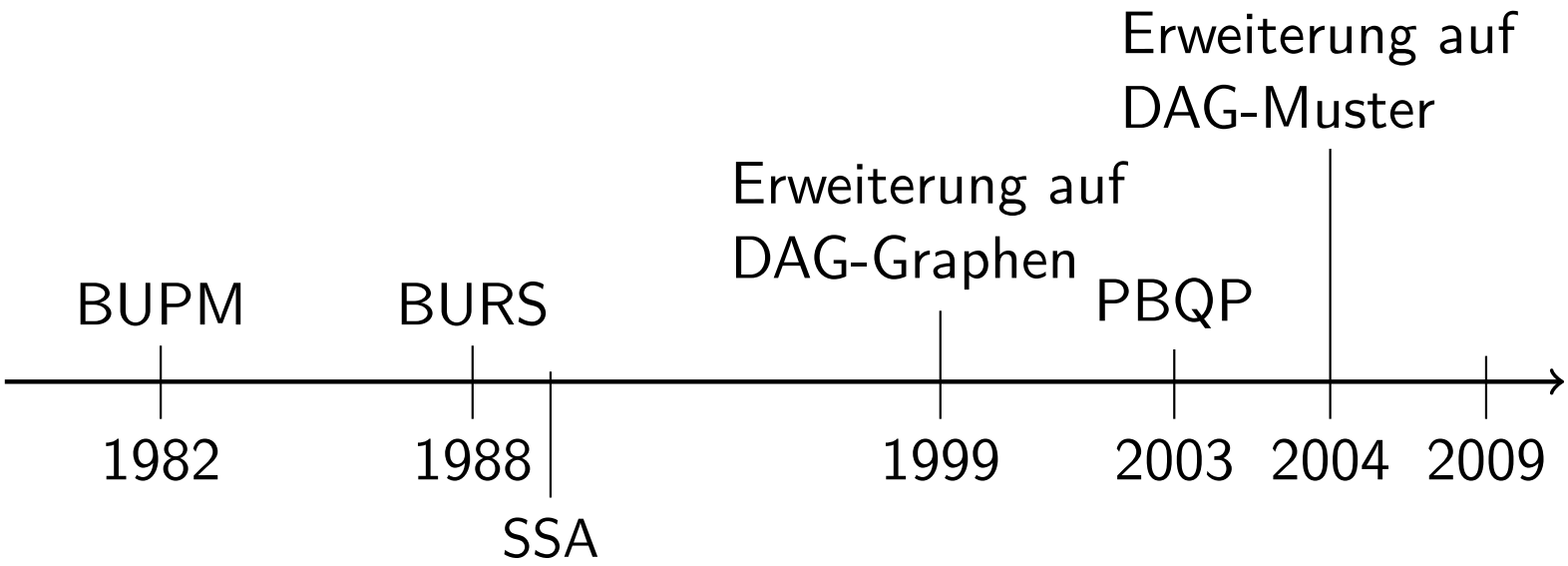
- Baum-Programmgraphen
- Baum-Muster (Spezifikation durch TES)

Generierte Befehlsauswahl



- DAG-Programmgraphen
- Baum-Muster (Spezifikation durch TES)

Generierte Befehlsauswahl



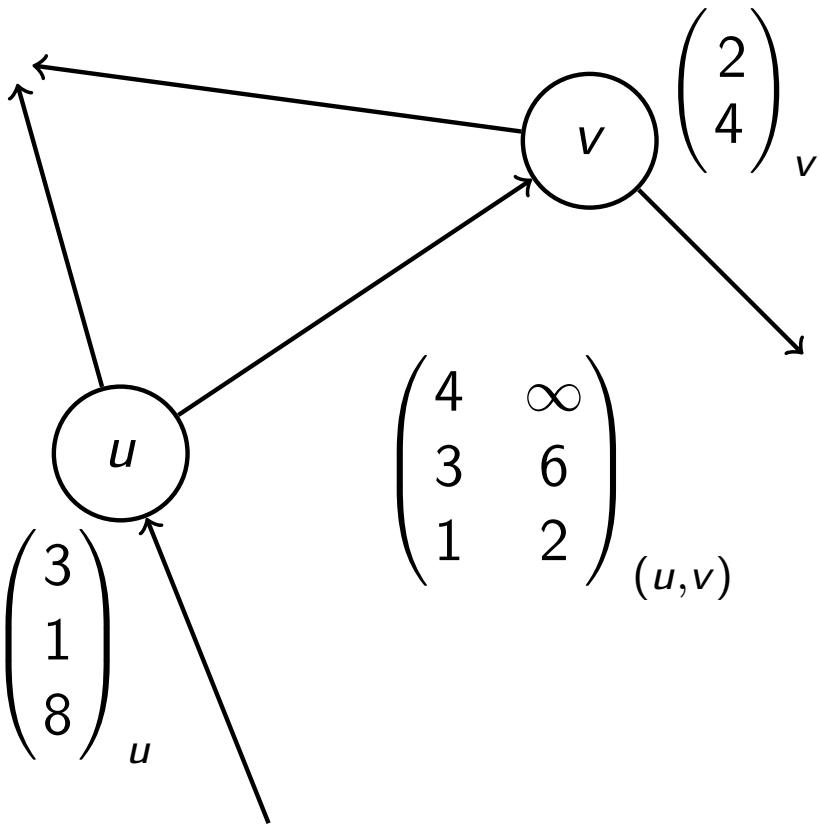
- DAG-Programmgraphen
- DAG-Muster

Überblick über unsere Arbeit

Eine PBQP-basierte, generierte Befehlsauswahl

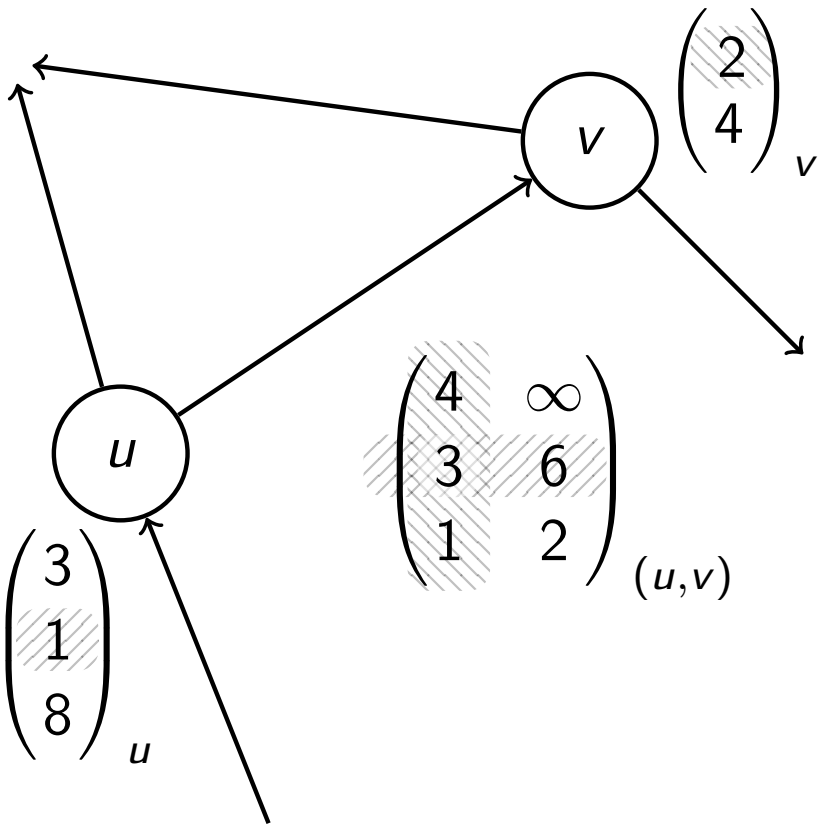
- Verifikation des Verfahrens
- Implementierung eines Prototyps
- Entwurf einer Spezifikationssprache

Partition Boolean Quadratic Problem (PBQP)



- Jeder Knoten besitzt einen Vektor
- Jede Kante besitzt eine Matrix
- Matrix-Dimension passt zu Vektor-Dimensionen der Quell- und Zielknoten

Partition Boolean Quadratic Problem (PBQP)



- Optimierungsproblem
- In jedem Vektor soll ein Eintrag ausgewählt werden
- Dadurch wird in jeder Matrix ebenfalls ein Eintrag ausgewählt
- Summe aller gewählten Einträge soll minimiert werden

Komplexitätsbetrachtung

- Finden der optimalen PBQP-Lösung ist NP-vollständig
- Finden einer PBQP-Lösung ist NP-vollständig

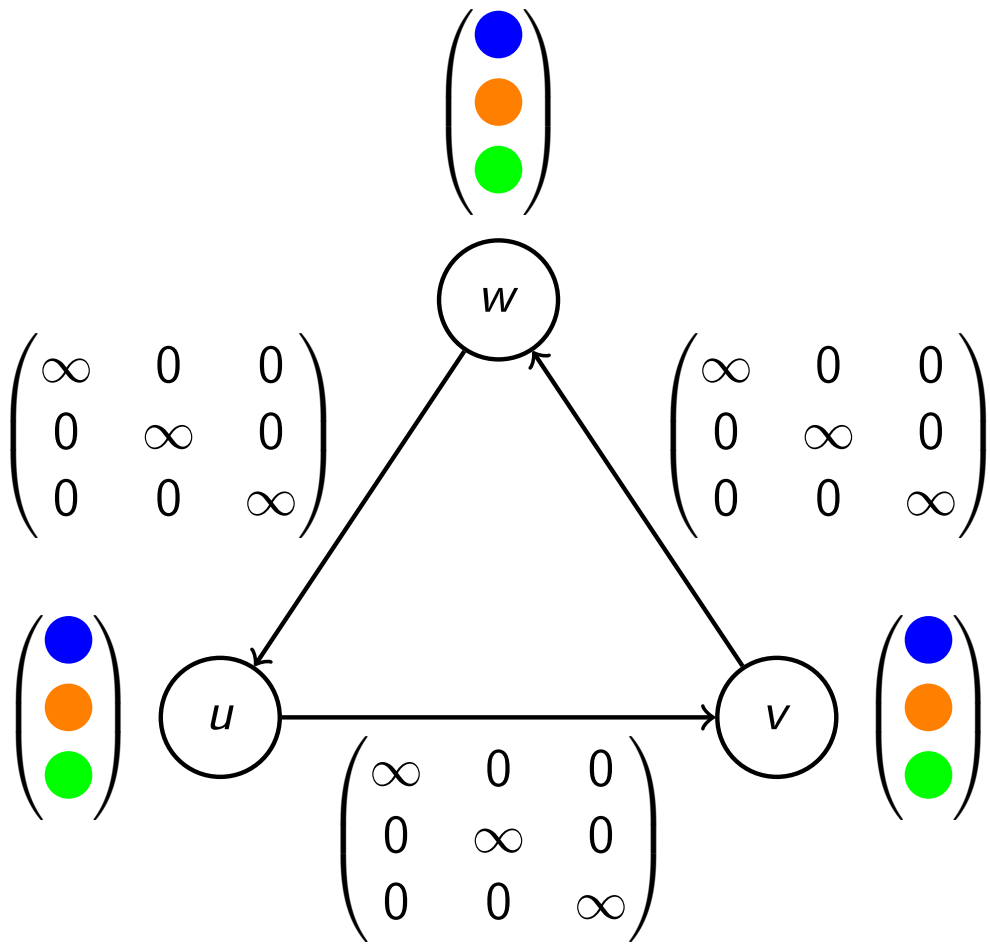
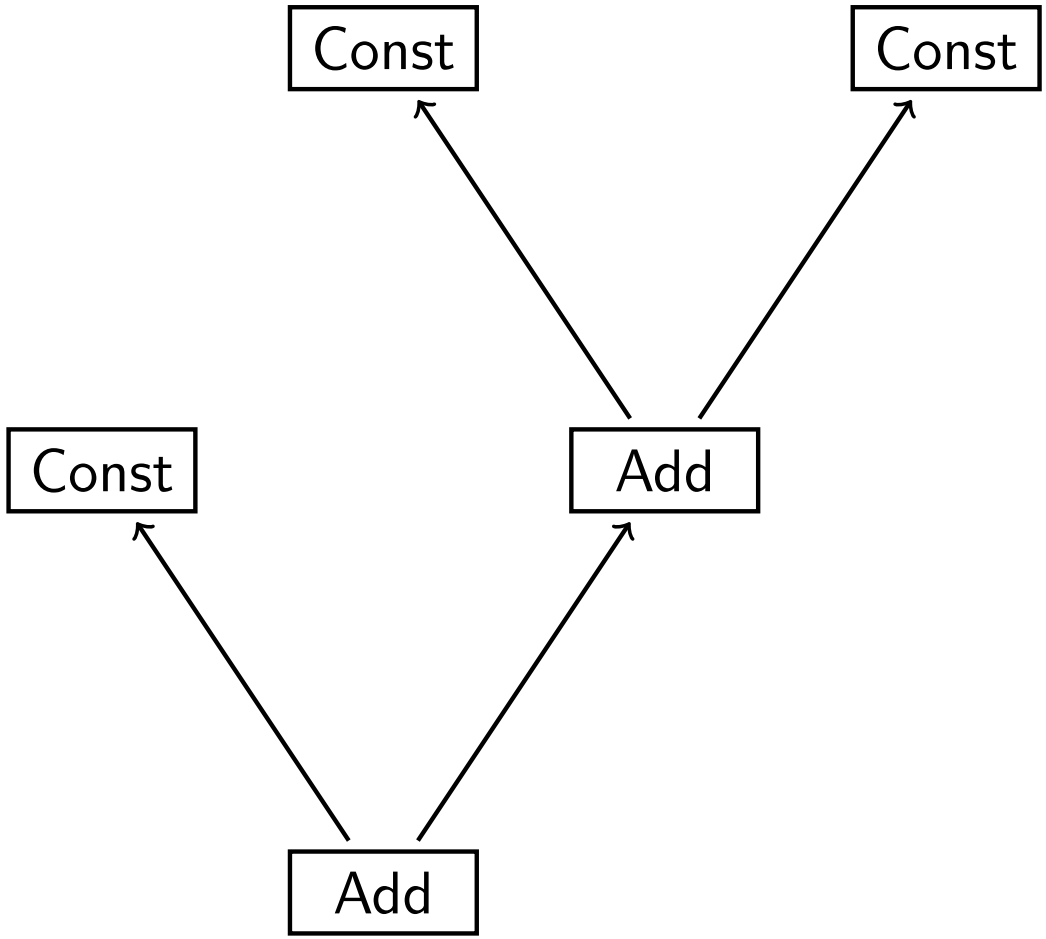
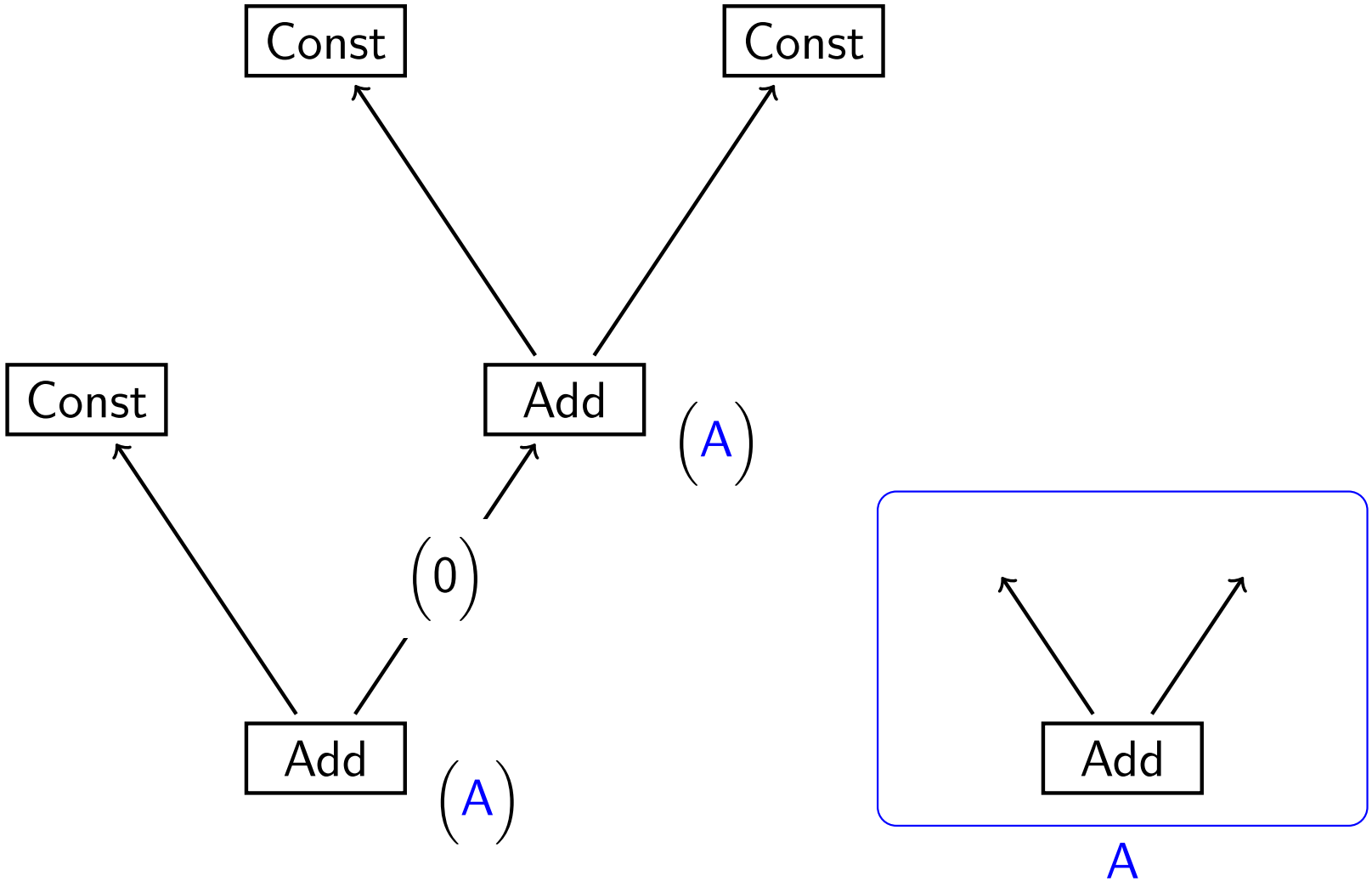


Abbildung von Befehlsauswahl auf PBQP



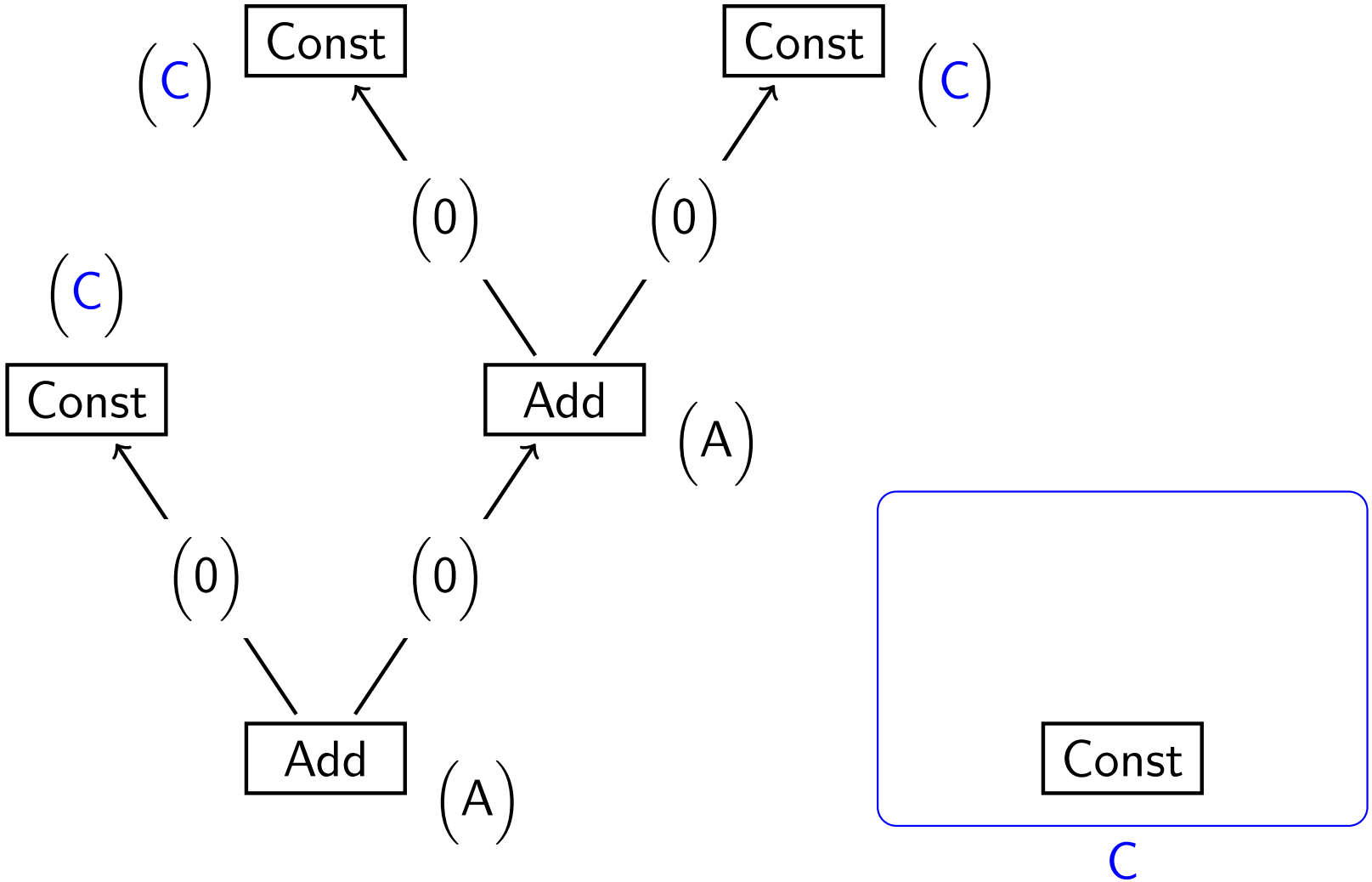
- PBQP-Graph $\widehat{=}$ Programmgraph

Abbildung von Befehlsauswahl auf PBQP



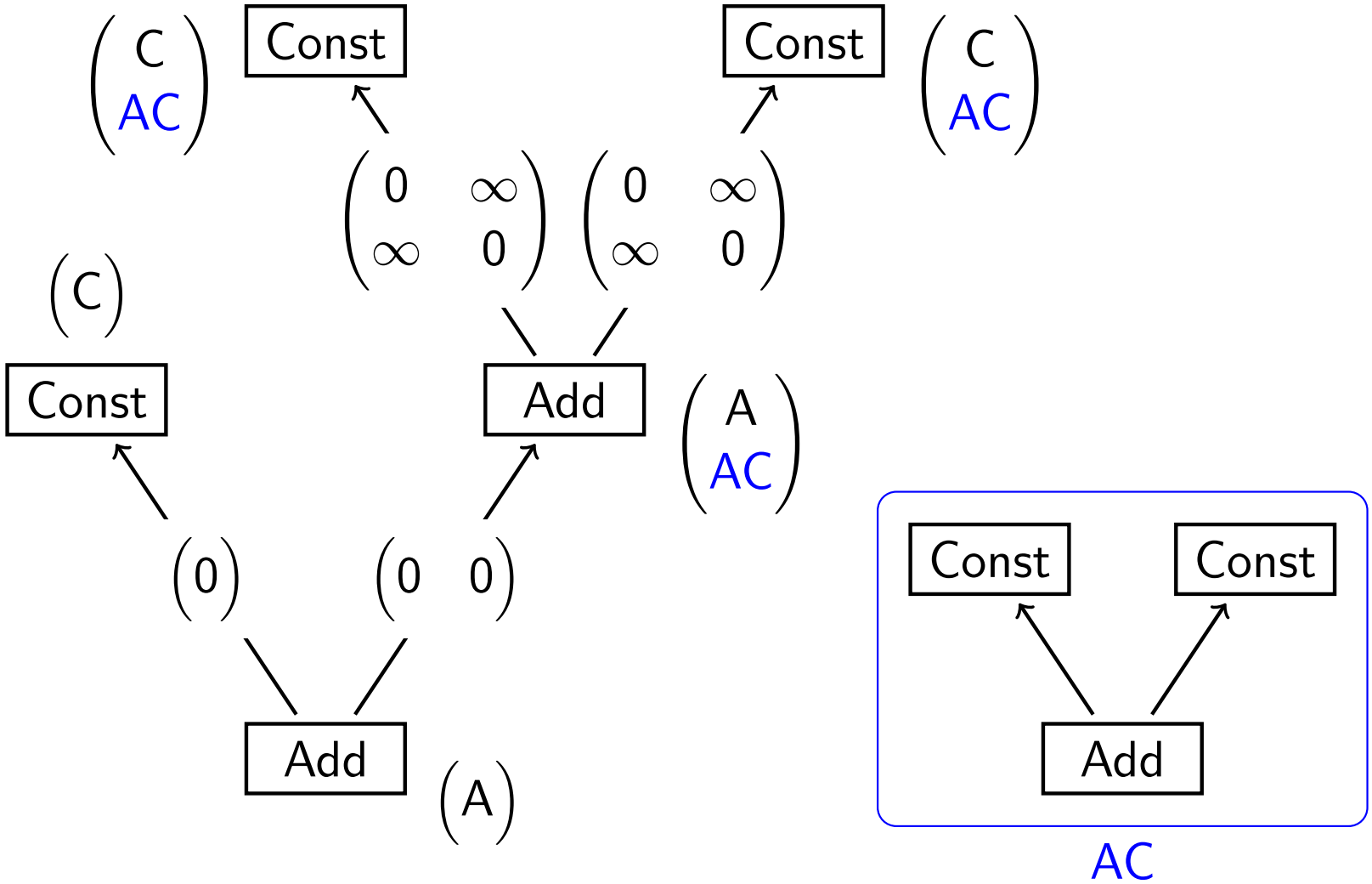
- Jeder Knotenvektor enthält Muster, die den Knoten überdecken

Abbildung von Befehlsauswahl auf PBQP



- Jeder Knotenvektor enthält Muster, die den Knoten überdecken

Abbildung von Befehlsauswahl auf PBQP



- Jeder Knotenvektor enthält Muster, die den Knoten überdecken
- ∞-Einträge in Matrizen für inkompatible Muster

PBQP-Reduktionen

Informationserhaltende Reduktionen:

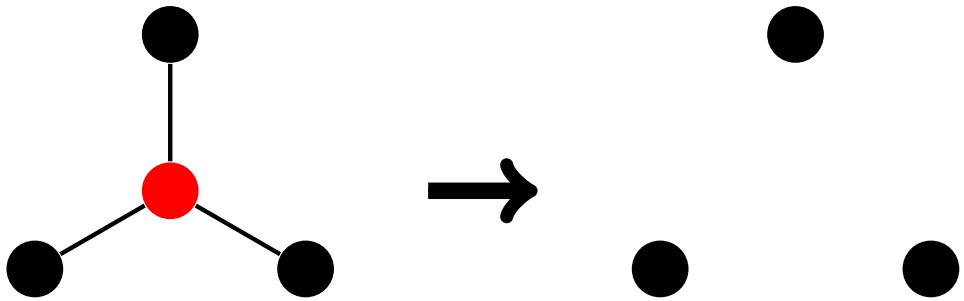
- Unabhängige Kanten (z.B. Kanten mit einer Nullmatrix)

- Knoten vom Grad Eins

- Knoten vom Grad Zwei

PBQP-Reduktionen

- Heuristische Reduktion

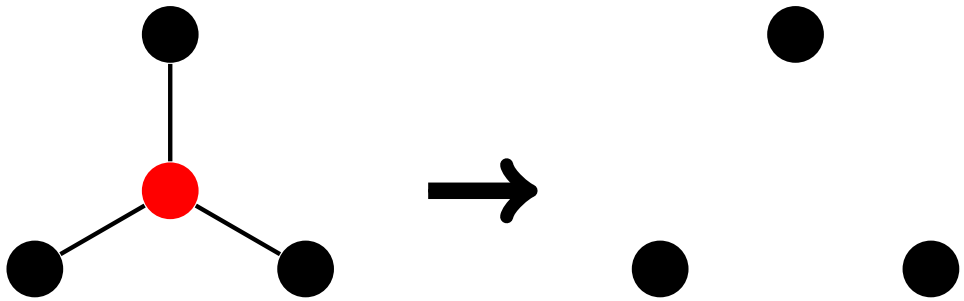


Es wird die lokal günstigste Alternative ausgewählt.

Der Algorithmus ist linear in Graphgröße.

PBQP-Reduktionen

- Heuristische Reduktion



Es wird die lokal günstigste Alternative ausgewählt.

Der Algorithmus ist linear in Graphgröße.

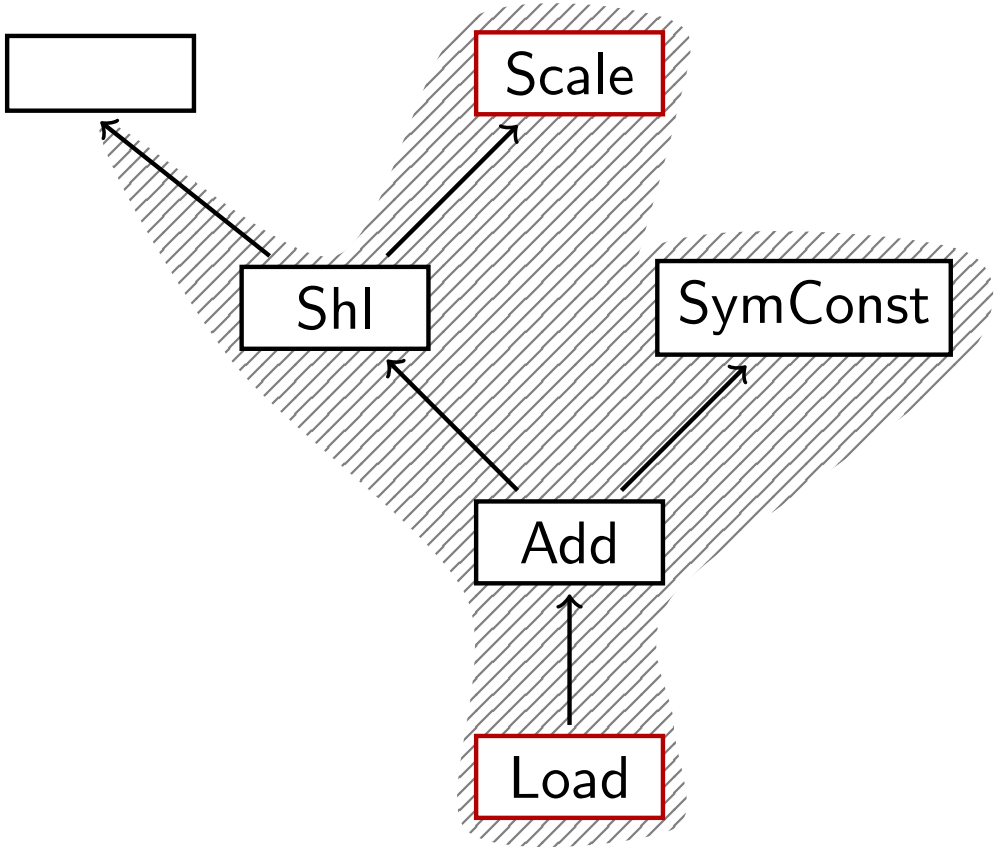
Lösungsgarantie trotz NP-Vollständigkeit?

Lösungsgarantie trotz NP-Vollständigkeit?

Ein bisher offenes Problem bei PBQP-basierter Befehlauswahl

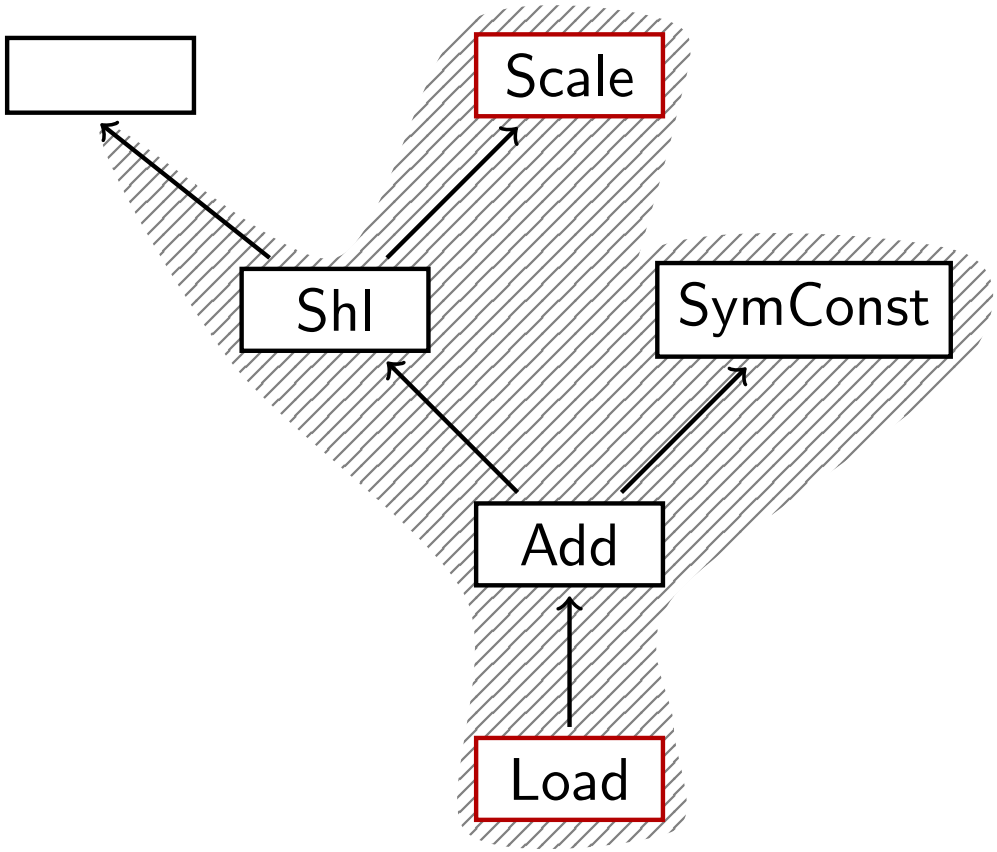
1. Hinreichende Voraussetzungen identifizieren
2. Lösungsgarantie beweisen

Erweiterung des PBQP-Lösungsalgorithmus



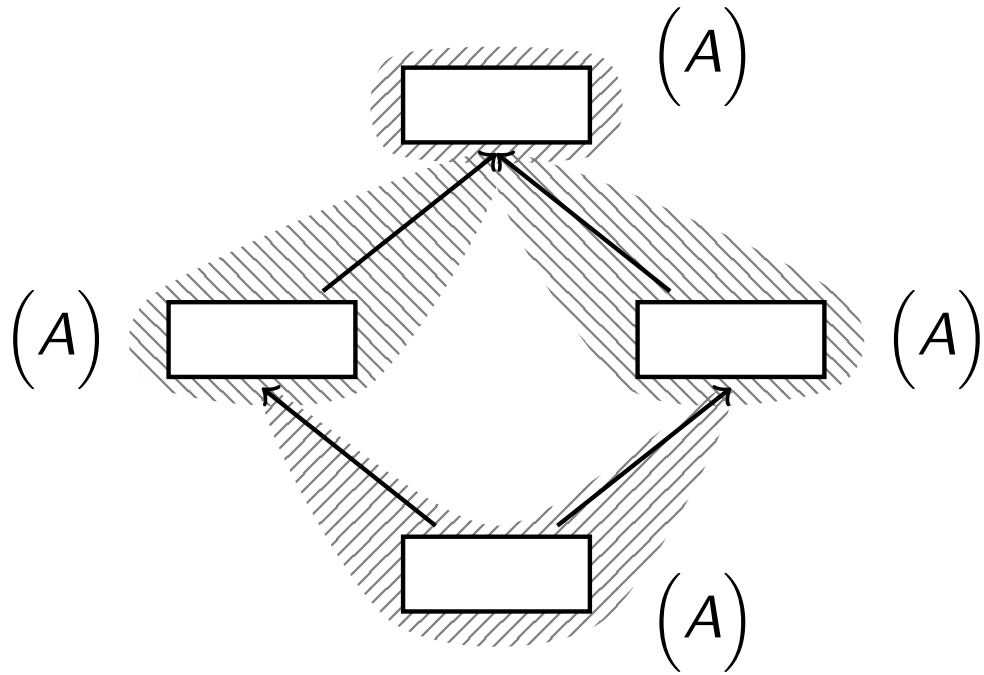
- Inkompatible Auswahl mit bisherigem Algorithmus möglich

Erweiterung des PBQP-Lösungsalgorithmus



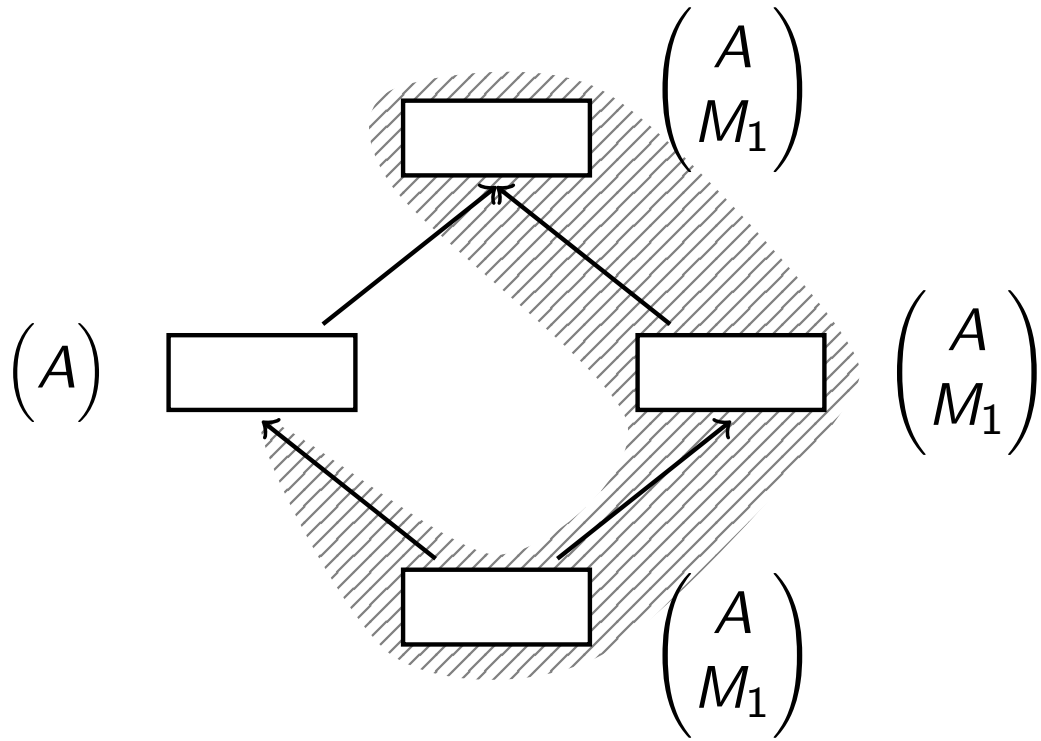
- Inkompatible Auswahl mit bisherigem Algorithmus möglich
- Rekursive Markierung ungültiger Alternativen notwendig

Motivation der Kompositionalität



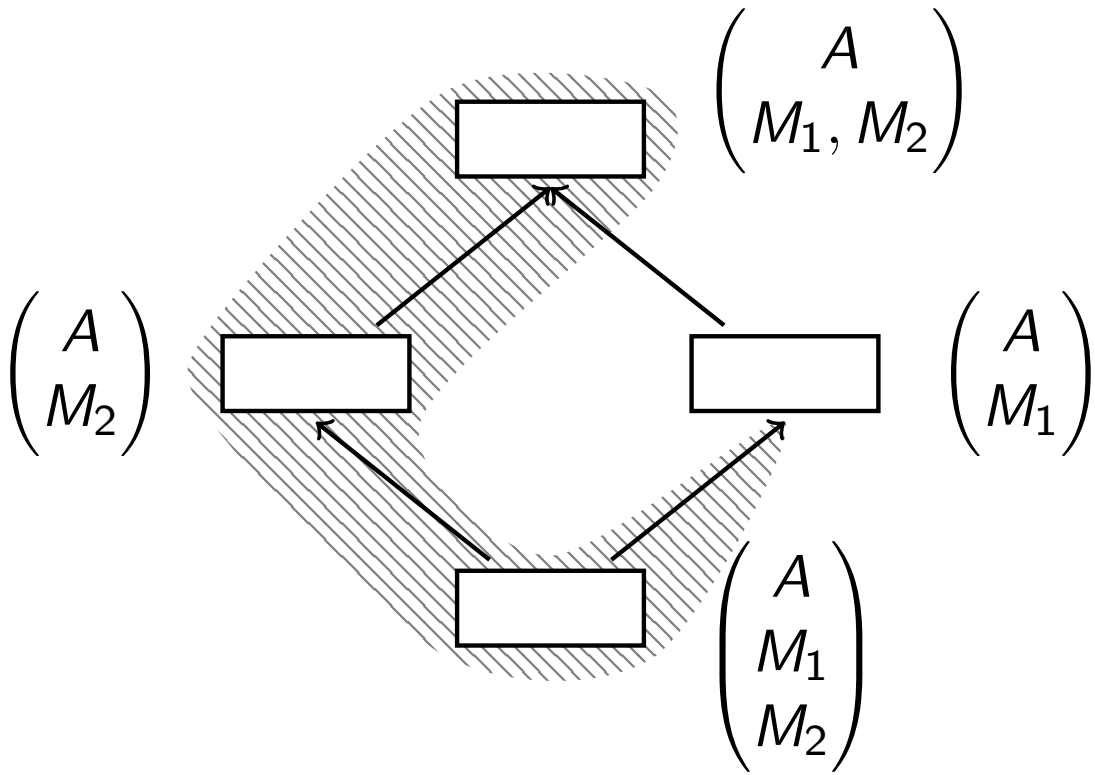
Atomare Muster überdecken den Programmgraph.

Motivation der Kompositionalität



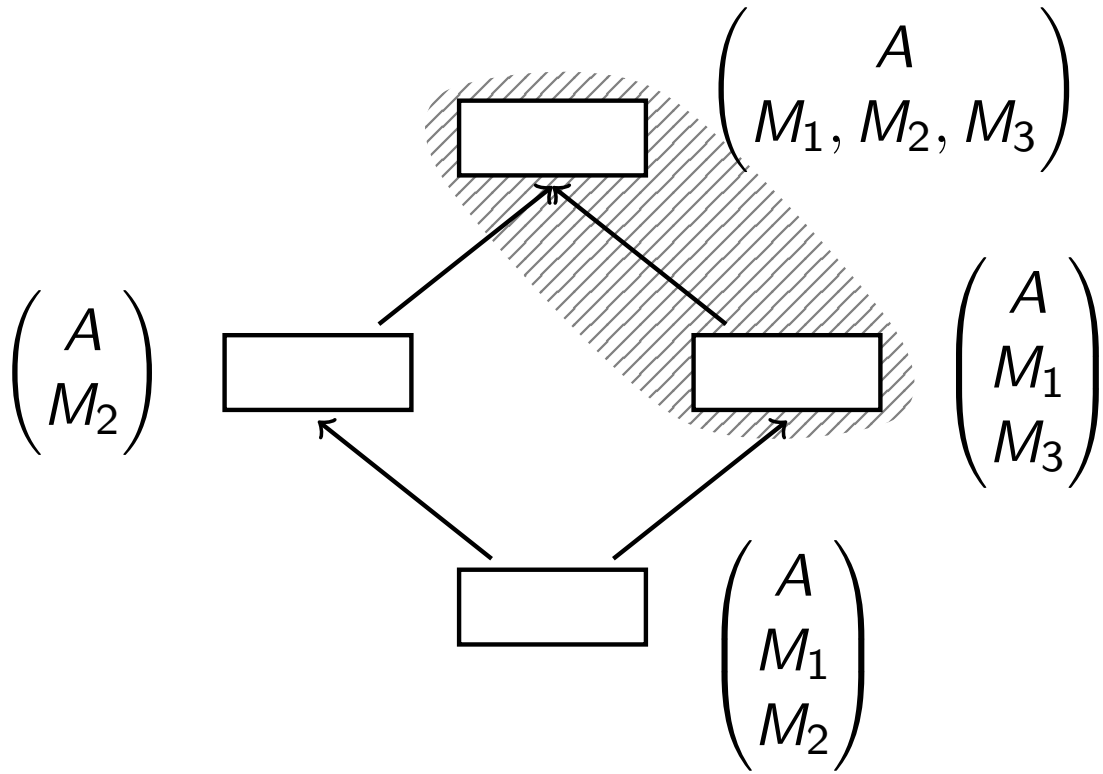
Muster M_1 ist an den überdeckten Knoten eingetragen.

Motivation der Kompositionalität



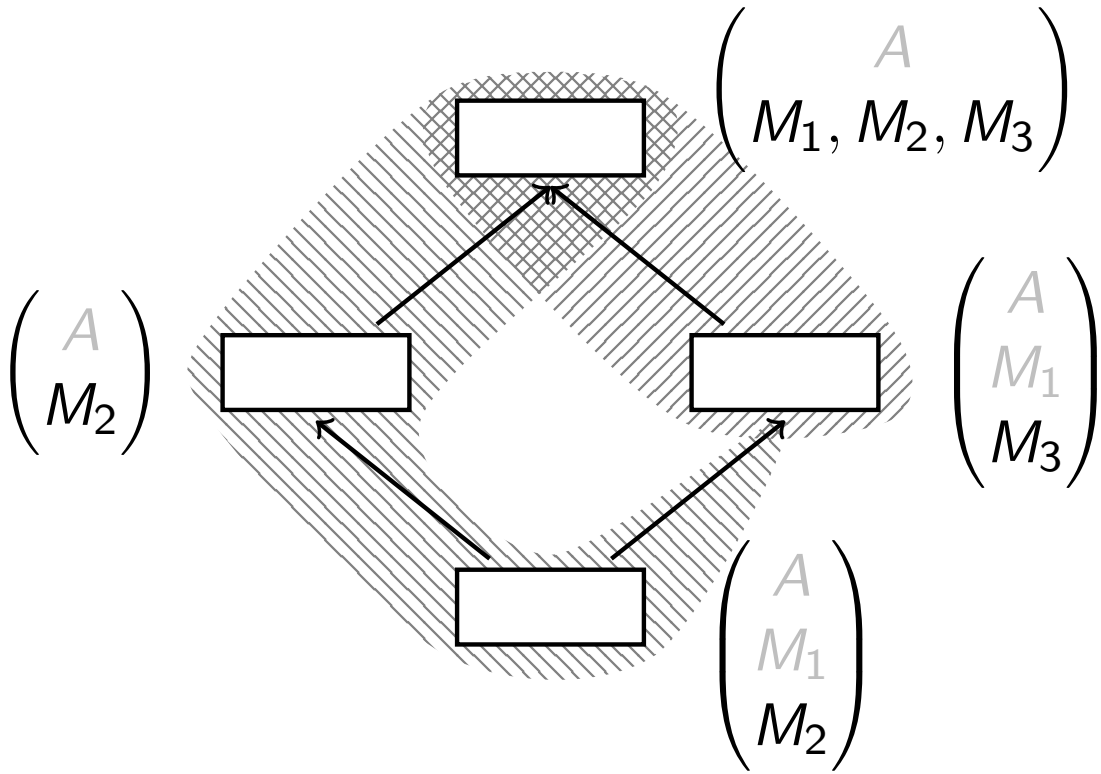
Am Blattknoten sind M_1 und M_2 dieselbe Alternative.

Motivation der Kompositionalität



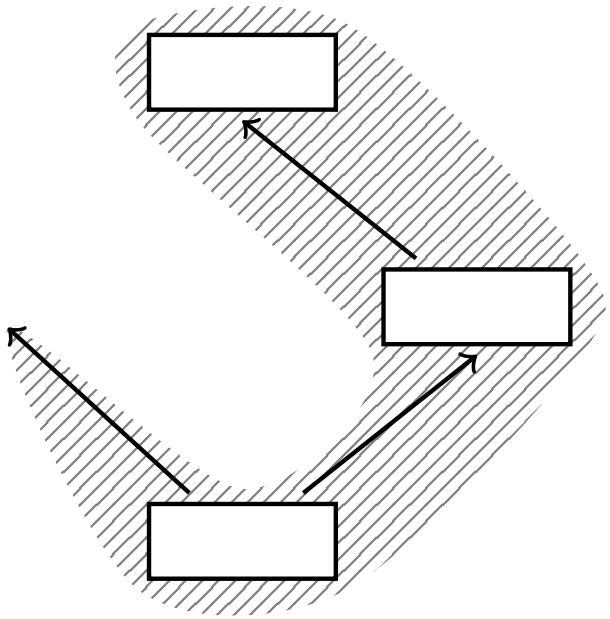
Muster M_3 gehört ebenfalls zu dieser Alternative.

Motivation der Kompositionalität



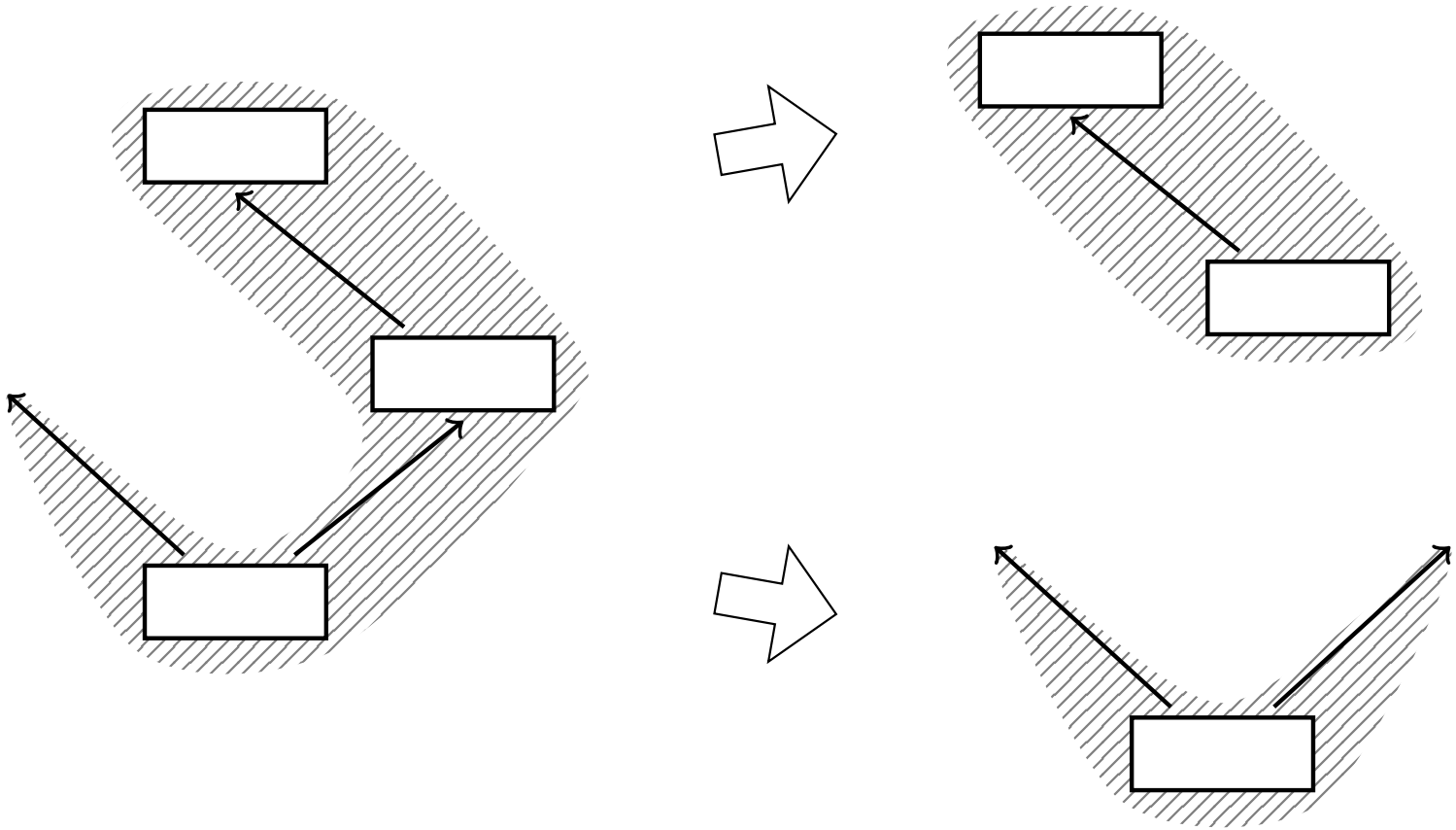
Mit Muster M_3 existiert eine weitere Lösung.

Definition: Kompositionalität



Jedes Muster . . .

Definition: Kompositionalität



Jedes Muster kann an Nachfolgern des Wurzelknotens „aufgespaltet“ werden.

Bedingungen für Korrektheit

- Erweiterter PBQP-Lösungsalgorithmus
- Mustermenge atomar-vollständig und kompositional

$\Rightarrow$ Lösungsgarantie

Automatische Vervollständigung

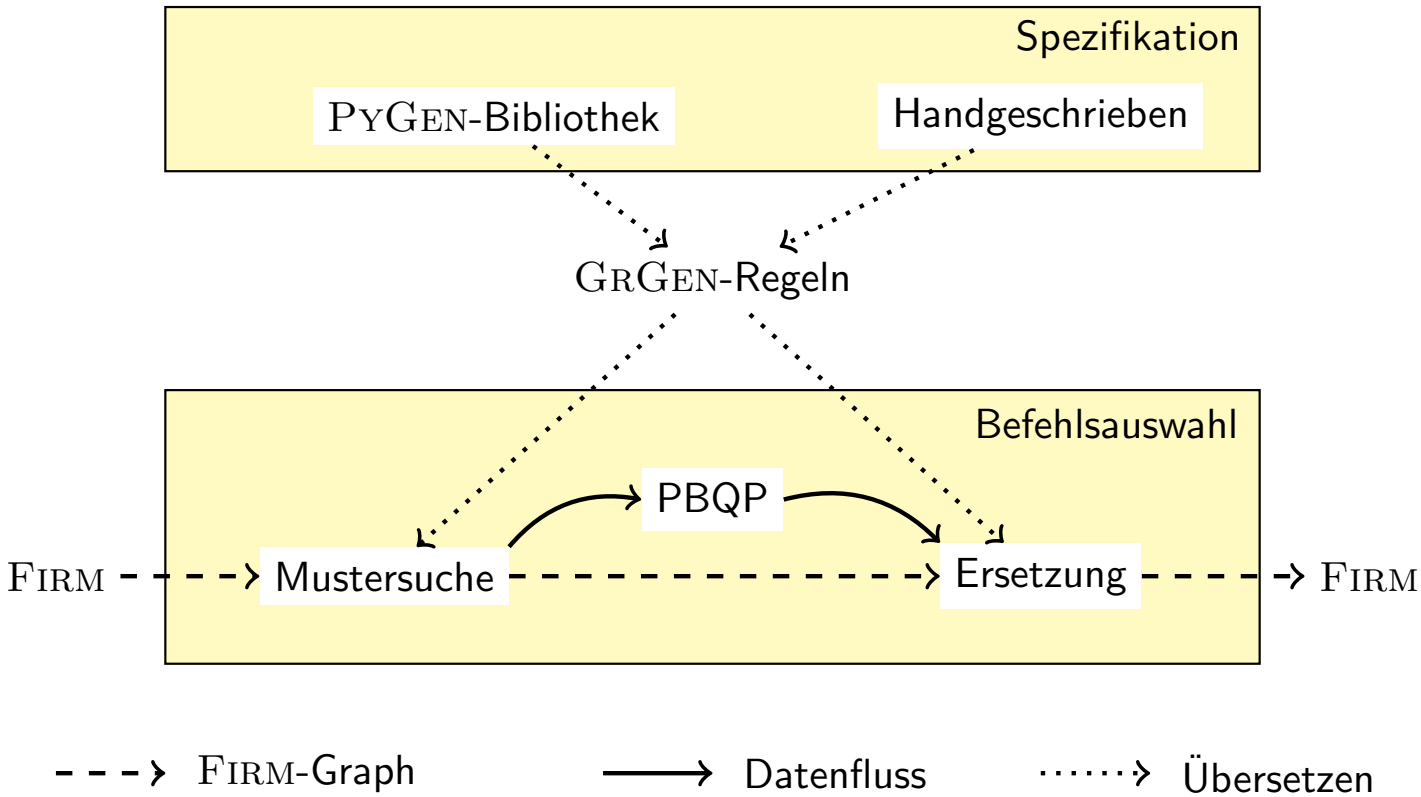
atomar-vollständig $\Rightarrow$ kompositional

- Fehlende Muster automatisch aus atomaren Mustern zusammensetzbar
- Meist effizientere Ersetzung möglich

Implementierung

Bild von http://flickr.com/photos/huladancer22/530743543/

Architektur



Bewertung unserer Implementierung

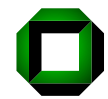
1. Laufzeiten der übersetzten Programme
2. Laufzeiten der Befehlsauswahl
3. Umfang der Spezifikation

SPEC CINT2000

Testprogramm	GCC	libFirm	PBQP	$\frac{\text{libFirm}}{\text{PBQP}}$
164.gzip	187	182	186	97.7%
175.vpr	276	296	292	101.3%
176.gcc	123	130	135	96.5%
181.mcf	327	329	327	100.5%
186.crafty	135	124	134	92.5%
197.parser	257	261	264	98.8%
253.perlbmk	250	196	220	89.1%
254.gap	128	139	142	97.9%
255.vortex	200	206	214	96.3%
256.bzip2	236	242	244	99.2%
300.twolf	459	512	479	107.0%
			Durchschnitt	97.9%

Warum nicht besser?

- Mustermenge
- Optimalität der Auswahl
- Kostenmodell



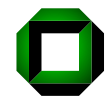
Optimalität

Testprogramm	$R1$	$R2$	heur.	sub.
164.gzip	6.550	581	0	0
175.vpr	18.412	4.528	42	0
176.gcc	244.541	33.426	52	2
181.mcf	1.782	233	1	0
186.crafty	24.263	6.819	12	0
197.parser	25.297	3.340	0	0
253.perlbmk	99.917	14.869	3	0
254.gap	80.558	19.402	47	0
255.vortex	112.548	17.750	9	0
256.bzip2	6.529	660	2	0
300.twolf	31.602	8.538	32	0
	651.999	110.146	200	2

Optimalität

- 99,9997 % der Reduktionen optimal
- 2 von 5973 Funktionen suboptimal

$\Rightarrow$ Die Schwachstelle liegt *nicht* im PBQP-Auswahlalgorithmus



Warum nicht besser?

- Mustermenge
- Optimalität der Auswahl
- Kostenmodell

Übersetzerlaufzeiten

Knoten	Suche	Auswahl	Ersetzung	PBQP	libFirm	$\frac{\text{PBQP}}{\text{libFirm}}$
71	401 ms	1 ms	1 ms	925 ms	1 ms	925
1.449	420 ms	134 ms	9 ms	1,2 s	10 ms	119
5.734	3,5 s	34 ms	20 ms	4,2 s	65 ms	65
54.948	7,7 s	259 ms	146 ms	8,8 s	588 ms	15
105.376	25,5 s	1,0 s	497 ms	29,3 s	2,3 s	13

Konstante Initialisierungszeit von ca. 0,5 Sekunden

Spezifikationsumfang

PyGen (1200 Zeilen)

Handgeschrieben (9000 Zeilen)

Ausblick

Es geht besser!

Bild von http://flickr.com/photos/kilobar/72769242/

Ausblick

- Verbesserte Mustersuche
- Verbessertes Kostenmodell und erweiterte Mustermenge
- Verifikation von Erweiterungen des PBQP-Aufbaus
- Schwächere Voraussetzungen durch automatische Verifikation

Zusammenfassung

PBQP-basierte Befehlauswahl ist

- gut geeignet für generative Verfahren
- leistungsfähig und flexibel

Lösungsgarantie

- unter einfachen Voraussetzungen
- bewiesen.

Vielen Dank für ihre Aufmerksamkeit!

Kostenmodell

1. Je größer das Muster, desto teurer
2. Vergünstigung für Immediate und Lea
3. Multiplikation mit der Ausführungshäufigkeit