Baubarkeitsprüfung von Kraftfahrzeugen durch automatisches Beweisen

Diplomarbeit

Carsten Sinz

Eberhard-Karls-Universität
Wilhelm-Schickard-Institut für Informatik
Arbeitsbereich Symbolisches Rechnen
Sand 13
72076 Tübingen

16. Dezember 1997

Hiermit versichere ich, die Arbeit selbständig verfaßt und keine anderen als die angegebenen Hilfsmittel benutzt zu haben.

Tübingen, im Dezember 1997

Dekan:	Professor Dr. U. Güntzer
1. Berichterstatter:	Professor Dr. W. Küchlin
2. Berichterstatter:	Privatdozent Dr. R. Bündgen

Betreuer bei der debis Systemhaus Engineering GmbH:
D. Bendrich

Inhaltsverzeichnis

1 Einleitung **6**

2 Problembeschreibung **8**

2.1 Aktueller Stand und Ziele 8

2.2 Das Datenbankmodell 13

2.2.1 Umsetzung der Baumuster in Codes 13

2.2.2 Baubarkeitskontrolle 13

2.2.3 Zusteuerung 16

2.2.4 Teilebedarfsermittlung 17

3 Formalisierung der Teilebedarfsermittlung **19**

3.1 Grundlegende Definitionen 19

3.2 Interpretation wichtiger Begriffe 23

3.2.1 Codes 23

3.2.2 Aufträge 24

3.2.3 Selektionsfunktionen 24

3.2.4 Interpretation der Baubarkeit 28

3.2.5 Interpretation der Zusteuerung 31

3.2.6 Teilebedarfsermittlung 33

3.3 Wichtige Kriterien und Eigenschaften 35

3.3.1 Pauschale Codebedingung 36

3.3.2 Baureihenabhängige Baubarkeit 36

3.3.3 Einschränkung der Lenkung und Ausführungsart 38

3.3.4 Die Baubarkeitsformel 38

3.3.5 Zusteuerung 39

3.3.6 Baumuster 41

3.4 Baubare Modelle mit Nebenbedingungen 41

3.5 Reihenfolgeabhängigkeit der Zusteuerung 42

4 Vergleich verschiedener autom. Beweistechniken **46**

4.1 Binäre Entscheidungsdiagramme 47

4.2 Termersetzungssysteme 51

4.3 Resolution . . . 53
4.4 Das Davis-Putnam-Verfahren . . . 55
4.5 Stålmarcks Methode . . . 57
4.6 Zur Berechnung konjunktiver Normalformen . . . 59
4.7 Zusammenfassung und Vergleich . . . 60

5 Implementation **61**
5.1 C++-Bibliothek . . . 61
5.1.1 Datenbank-Anbindung . . . 62
5.1.2 Aussagenlogischer Formelmanipulator . . . 64
5.2 Prototypen von Testprogrammen . . . 67

6 Ergebnisse **74**
6.1 Unzulässige und notwendige Codes . . . 74
6.2 Mehrdeutigkeiten in der Zusteuerung . . . 75
6.3 Konsistenz der Zusteuerung . . . 77
6.4 Konsistenz der Teilebedarfsermittlung . . . 78

7 Zusammenfassung und Aussichten **84**

A Statistische Größen **86**

Abbildungsverzeichnis

2.1 Vom Auftrag zur Teileliste 10
2.2 Struktur der baureihenabhängigen Baubarkeitskontrolle 15
2.3 Struktur der Stückliste 18

4.1 Auswirkung verschiedener Variablenordnungen auf die BDD-Größe 49
4.2 Ein vollständiges (kanonisches) Termersetzungssystem für boolesche Logik 52
4.3 Der Davis-Putnam Algorithmus 56

5.1 Klassendiagramm: Datenbank-Anbindung 62
5.2 Klassendiagramm: Aussagenlogischer Formelmanipulator 65
5.3 Schematischer Ablauf einer Konsistenzprüfung 68

Tabellenverzeichnis

4.1 BDD-Größen verschiedener Teilformeln von $\mathcal{B}_P$ . . . 50
4.2 Vergleich verschiedener dynamischer Variablen-Umordnungs-Strategien . . . 51
4.3 Ergebnisse mit ReDuX und JRAP am Beispiel der Formel $\mathcal{B}_P$ . . . 53
4.4 Ergebnisse mit OTTER am Beispiel der Formel $\mathcal{B}_P$ . . . 54
4.5 Messungen mit SATO anhand der Formeln $\mathcal{B}_P$, $\mathcal{B}_K$ und $\mathcal{Z}_V$ . . . 56
4.6 Konvertierung in konjunktive Normalform . . . 59

Kapitel 1

Einleitung

Die formale Beschreibung und Verifikation von Prozeßabläufen hat noch vor wenigen Jahren in der Industrie so gut wie keine Rolle gespielt, und auch heute ist der Einsatz von Beweismethoden auf diesem Gebiet nicht sehr weit verbreitet. Lediglich in der Halbleiterindustrie konnten mathematische Beweisverfahren schon relativ früh (Ende der 80er – Anfang der 90er Jahre) den Weg aus den Forschungslabors ins industrielle Umfeld schaffen. Dort konnten zum Teil beachtliche Erfolge bei der formalen Überprüfung zunehmend komplexerer Chips erreicht werden. In [Mar97] findet man eine Übersicht dieser Untersuchungen und der dabei auftretenden Probleme.

Die untergeordnete Rolle, die formale Beweisverfahren in der computerunterstützten industriellen Produktion spielen, hat vielfältige Ursachen. Zum einen gibt es kein einfaches, allgemeingültiges Verfahren, einen bestehenden Prozeßablauf zu formalisieren. Daneben stehen oft Verständigungsprobleme und unterschiedliche Begriffswelten einer zufriedenstellenden Modellierung im Weg. Zur Umsetzung der vorhandenen Betriebsabläufe in ein mathematisches Modell ist jedoch genau dieses Verständnis zwingend erforderlich. Der hohe zeitliche Aufwand eines solchen Abstimmungs- und Verständigungsprozesses wirkt oft abschreckend.

Darüberhinaus sind die vorhandenen Verifikationstechniken und Beweiser für komplexe Probleme häufig nicht ausreichend, in vielen Fällen stößt man schnell auf zu hohen Rechenzeit- und Speicherplatzbedarf. In letzter Zeit zeichnet sich ab, daß eine Kombination verschiedener Techniken weitere Fortschritte ermöglichen könnte [Hoa96]. Außer der bloßen Größe der untersuchten Systeme macht oftmals die zur Modellierung verwendete oder naheliegendste Sprache (Prädikatenlogik) schnelle Lösungen unmöglich.

Die Erfolge im Hardware-Sektor lassen sich in diesem Lichte leichter verstehen: Die Modellierung fast aller marktüblichen Chips (synchron, nur ein globaler Zeittakt) kann mit einem einheitlichen Verfahren vorgenommen werden; zur formalen Beschreibung der Chips reicht boolesche oder modale bzw. temporale Logik aus, für die es relativ effiziente Entscheidungsverfahren gibt; letztendlich sind die Begriffswelten in der Halbleiterentwicklung und der mathematischen Logik zumindest benachbart.

Innerhalb der Forschungsgemeinschaft im automatischen Beweisen besteht andererseits ein großes Interesse an praxisnahen Anwendungsbeispielen, die über einfache Problemstellungen hinausgehen. Bisher wurden Vergleiche verschiedener Beweisverfahren hauptsächlich anhand rein mathematischer Aussagen vorgenommen. Dabei konnten in speziellen Bereichen [Sla94] Aussagen bewiesen werden, deren Gültigkeit zuvor noch offen war. Neue Beweise, die nicht von Mathematikern sondern mit computerunterstützten Verfahren gefunden wurden, blieben trotzdem rar.

In der Industrie müssen sich formale Spezifikations- und Verifikationstechniken erst noch etablieren [BH94]. Weitere erfolgreiche Projekte in diesem Bereich könnten dieser Entwicklung Vorschub leisten.

Beispiele, in denen mittels formaler Verifikation industriell verwertbare Erfolge erzielt werden konn-

ten, gibt es inzwischen genug, stellvertretend für viele andere seien die folgenden genannt. Im Hardware-Bereich konnten verschiedene Prozessoren (LILITH [SS89], AAMP5 [SM95], FM9001 [HJ89], DLX [BM96]), Bildverarbeitungs-Prozessoren (Sobel, [NS89]) und ATM-Schaltelemente [Gar95] verifiziert werden. Der auch in der Tagespresse beachtete Pentium-Bug wurde ebenfalls nachträglich mittels formaler Methoden untersucht [Pra95].

Obwohl der Hardware-Sektor eine beherrschende Rolle beim Einsatz formaler Methoden spielt, gibt es neuerdings auch aus anderen industriellen Bereichen Fallbeispiele, die ernstzunehmen sind. Diese beschäftigen sich vor allem mit relativ überschaubaren Systemen aus dem sicherheitskritischen Umfeld. Auf diesem Gebiet herrscht verständlicherweise ein großes Interesse, die bereits vorhandenen computergesteuerten Anlagen auf Sicherheitsmängel hin zu überprüfen.

So wurden Teile einer chemischen Industrieanlage [TPP97] und die Steuerungssoftware eines hydroelektrischen Kraftwerks [PT96] formal spezifiziert und verifiziert. Die schwedische Firma Logikkonsult NP AB hat verschiedene Verifikations-Projekte durchgeführt, unter anderem zur Signalsteuerung auf Eisenbahnstrecken und zur Kontrolle sicherheitsrelevanter Aspekte von Bussen. Eine Zusammenfassung dieser Ergebnisse findet man in [Bor97].

Neben der Verifikation bieten sich für formal spezifizierte Systeme auch Testverfahren an. Gerade in der Überprüfung von Chip-Designs war dies die übliche Vorgehensweise bevor Beweiser (meist model-checker) eingesetzt wurden.

Gegenüber Testverfahren hat die Verifikation entschiedene Vorteile. So kann auch eine Vielzahl einzelner Tests nie die Sicherheit liefern, die man mittels mathematischer Beweisverfahren erzielt. Die Tests beschränken sich darüberhinaus auf gebräuchliche Beispiele, weniger häufig auftretende Fälle werden entweder nicht betrachtet oder übersehen. Kurz gesagt können Testverfahren nur die Anwesenheit von Fehlern zeigen, niemals deren völlige Abwesenheit. Darüberhinaus können All- und Existenz-Aussagen meist nur durch Beweiser bestätigt oder widerlegt werden.

In dieser Arbeit soll die Anwendbarkeit formaler Verifikationsmethoden zur Üperprüfung von speziellen Produktionsdatenbanken untersucht werden. Solche Datenbanken werden beispielsweise bei Mercedes-Benz in der Fahrzeugproduktion zur Baubarkeitskontrolle von Aufträgen verwendet.

Kapitel 2

Problembeschreibung

Die Fahrzeuge der Daimler-Benz AG können in einer Vielzahl von Variationen bestellt werden. Außer verschiedenen Modellreihen, die sich selbst wiederum aus verschiedenen Motor- und Getriebeausführungen zusammensetzen können, hat man es mit einem großen Sortiment an Sonderzubehör, Lackierungen und Innenausstattungen zu tun. Die gesamte Fahrzeugmodellpalette ist enorm umfangreich.

Allerdings gibt es viele Kombinationsmöglichkeiten, die nicht sinnvoll sind (zum Beispiel mehrere Motoren) oder sich nicht produzieren lassen. Neben geometrischen Problemen (Platzbedarf einzelner Teile) kann es auch technisch-, termin- und vertriebsbedingte oder länderspezifische Ausschlüsse geben.

Mercedes-Benz verwendet daher eine regelbasierte Datenbank, in der solche Ausschlüsse und Abhängigkeiten gespeichert sind. Jeder Auftrag muß diese Kontrolle durchlaufen haben, bevor er als korrekt und baubar akzeptiert wird.

Danach kann anhand des überprüften Auftrags der Teilebedarf für die gewünschte Ausstattungsvariante bestimmt werden.

Die Komplexität des Produkts macht jeden dieser Schritte kompliziert und fehleranfällig. Eine computerunterstützte Analyse und Verifikation des gesamten Prozesses kann manche dieser Fehler vermeiden helfen. Dazu ist allerdings eine formale Beschreibung der Abläufe unumgänglich.

Die Details des Prozesses vom Auftragseingang bis zur Generierung der Teileliste werden in den folgenden Abschnitten näher beleuchtet.

2.1 Aktueller Stand und Ziele

Vom Auftrag zur Teileliste

Zur Beschreibung der verschiedenen Fahrzeugmodelle werden "Codes" verwendet. Ein Kundenauftrag besteht aus einer Liste solcher Codes, die auch "Codeleiste" genannt wird. Sie gibt an, welche Merkmale das gewünschte Fahrzeugmodell aufweisen soll. So gibt es Codes für verschiedene Motorausführungen, Lackierungen, Sonderausstattungen wie Klimaanlage oder bestimmte Innenausstattungen. Durch Codes werden aber auch herstellungsspezifische Größen wie die Modellreihe und Ausführungsart (Limousine, Kombi, etc.) oder das Auslieferungsland festgelegt.

Welche Modelle baubar sind wird durch die "Produktdokumentation" [Mer95] festgelegt, die mittels Regeln beschreibt, welche Codekombinationen erlaubt sind und welche nicht.

Die Produktdokumentation besteht zum einen aus einer regelbasierten Datenbank, der Baubarkeitskontrolle, in der boolesche Formeln angeben, welche Modelle hergestellt werden können. Die Aussagenvariablen dieser Formeln sind gerade die Codes, die in der Bestellung auftreten können.

Darüberhinaus enthält sie eine weitere Komponente, "Zusteuerung" genannt, durch die Modifikationen an Aufträgen ermöglicht werden. Die Änderungen durch die Zusteuerung bestehen darin, daß zusätzlich zu den im Auftrag vorkommenden Codes weitere nicht vom Kunden spezifizierte Codes dem Auftrag hinzugefügt ("zugesteuert") werden. Damit lassen sich zum Beispiel Ausstattungspakete oder Default-Teile beschreiben. Die Zusteuerung ist genau wie die Baubarkeitskontrolle als regelbasierte Datenbank realisiert.

Anhand der Codes wird aus dem überprüften und eventuell modifizierten Auftrag der Teilebedarf für die Produktion ermittelt. Mit diesem letzten Schritt im Auftragsverarbeitungsprozeß wird also eine Decodierung in einzelne Teile der Stückliste vorgenommen.

Um die Aufträge für die Bestellung kompakter zu halten, gibt es für verschiedene oft auftretende Codekombinationen Abkürzungen, sogenannte "Baumuster". Diese Baumuster werden vor der Zusteuerung und Baubarkeitskontrolle durch eine eindeutige Zuordnung in Codes umgewandelt.

Der gesamte Prozeßablauf vom Auftragseingang bis zur Teileermittlung ist in Abbildung 2.1 – links schematisch, rechts anhand eines Beispiels – nochmals dargestellt.

Fehlerpotential

Fehler in der Produktdokumentation sollten natürlich weitestgehend vermieden werden, aber die Vielzahl der möglichen Aufträge macht dies zu einer nicht trivialen Aufgabe.

So sind alleine für die Limousinen der C-Klasse 1151 Codes und 1519 Regeln für die Zusteuerung und Baubarkeitskontrolle vonnöten. Die Teileermittlung selbst setzt sich sogar aus nahezu 18000 Regeln zusammen. Jede Produktänderung ist begleitet von einer Vielzahl von Regelanpassungen in der Datenbank.

Die Erstellung und Anpassung dieser Regeln erfordert eine genaue Kenntnis der Produktstruktur, und selbst wenn dieses Wissen vorhanden ist, können neben den erwünschten Eigenschaften des Regelsystems unerwünschte Nebeneffekte auftreten und lange Zeit unentdeckt bleiben.

Fehler in den Steuerungsregeln können vielfältige Auswirkungen haben:

- Aufträge, die eigentlich baubar sein sollten, werden als inkorrekt abgelehnt. Dies führt zu einem überflüssigen Abstimmungsprozeß zwischen Händler bzw. Kunde und Werk.
- Es kann passieren, daß ein bestimmtes Teil in keinem Auftrag vorkommen kann (zum Beispiel nach einer Produktänderung), trotzdem aber noch in der Stückliste geführt wird und daher unnötigerweise vorgehalten werden muß.
- Die Teilebedarfsermittlung kann fehlerhaft sein. Dies kann neben Auswirkungen auf die Produktionsplanung und Teilefertigung auch in der Montage zu unnötigen Wartezeiten durch Rückfragen führen, falls z.B. an einer Einbauposition mehrere oder gar kein Teil eingebaut werden soll.

Bisherige Testverfahren

Die Vermeidung von Fehlern in den Steuerungsregeln ist daher von großer Bedeutung. Üblicherweise werden zur Überprüfung der Datenbanken entweder von Hand oder maschinell generierte Beispielaufträge herangezogen. Diese Testverfahren haben allerdings Nachteile:

- Selbst eine hohe Anzahl von Tests kann Fehler nicht ausschließen.
- Normalerweise werden nur "typische" Aufträge betrachtet, weniger gebräuchliche fallen unter den Tisch.
- Um unnötige Teile in der Stückliste aufzufinden, ist es zwingend erforderlich, alle möglichen Aufträge zu betrachten.

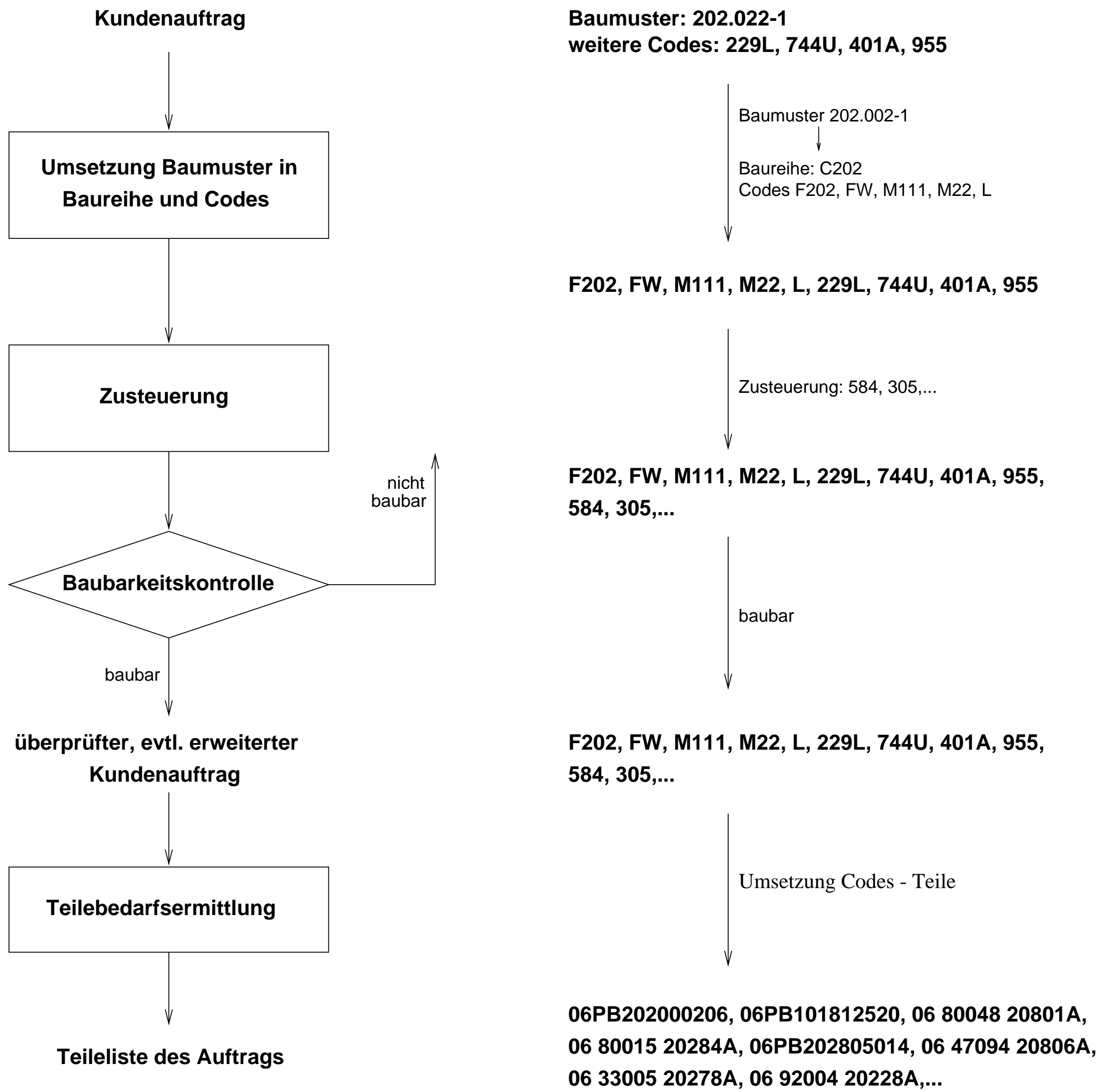


Abbildung 2.1: Vom Auftrag zur Teileliste

Es ist klar, daß es unmöglich ist, alle eventuell auftretenden Aufträge der Reihe nach zu testen. Denn schon bei 30 Codes gibt es über eine Milliarde theoretisch möglicher Aufträge, bei 1151 Codes sind es mehr als $3 \cdot 10^{346}$.

Vorteile formaler Verifikation

Der Einsatz von automatischen Beweisern bietet demgegenüber deutliche Vorteile.

- Es können zuverlässige Aussagen über *alle* möglichen Aufträge gemacht werden. Nicht mehr benötigte Teile können so mit Sicherheit aufgefunden werden.
- Viele Fehler können durch Verifikation, nicht aber mittels Testverfahren, gefunden werden.
- Das Generieren und Auswählen von Testaufträgen kann stark beschränkt werden.
- Schon während Änderungen an den Steuerungsregeln kann ein Beweiser unterstützend interaktiv eingesetzt werden. Dies kann zu einer Vereinfachung und Beschleunigung des Anpassungsprozesses führen.

Qualitätssicherung der bestehenden Produktdokumentation

Durch Verifikation lassen sich eine Reihe von Fragen beantworten, die im Hinblick auf mögliche Fehler in der Baubarkeitskomponente der Produktdokumentation relevant sind:

1. Gibt es Codes, die unabhängig von der Baureihe in keinem korrekten Auftrag vorkommen dürfen?
2. Gibt es Codes, die bei festgelegter Baureihe
   (a) in jedem korrekten Auftrag vorkommen müssen oder
   (b) in keinem korrekten Auftrag auftreten dürfen.

Die Zusteuerung betreffende Fragen zur Fehlervermeidung könnten wie folgt aussehen:

1. Gibt es baubare Aufträge, die durch Fehler in der Zusteuerung zu nicht mehr baubaren abgeändert werden? Wenn ja, wie sehen die betroffenen Aufträge aus?
2. Ist die Zusteuerung eindeutig bzw. nur aufgrund der zufälligen Reihenfolge, in der die Regeln in der Datenbank auftreten, eindeutig? Wenn ja, welche Aufträge und Zusteuerungsregeln sind davon betroffen?

Auch in Zusammenhang mit der Teileermittlung gibt es Kriterien, die durch formale Verifikation sichergestellt werden können:

1. Minimalität der Stückliste: Kein Teil der Stückliste sollte überflüssig sein, jedes Teil muß in mindestens einem baubaren Auftrag vorkommen.
2. Eindeutigkeit der Teileauswahl: Pro Stücklistenposition darf höchstens eine der möglichen Varianten[1] ausgewählt werden. Außerdem dürfen manche Positionen nicht freigelassen werden.

Durch formale Spezifikation und Verifikation werden Antworten auf diese Fragen möglich.

[1] Positionen und Varianten werden zusammen mit dem Datenbankmodell beschrieben. Man kann die Varianten einer Position vereinfacht als alle möglichen Teile auffassen, die an einem bestimmten Ort im Fahrzeug eingebaut werden können.

Unterstützung von Produktänderungen

Neben den oben erwähnten Kontrollmöglichkeiten gibt es weitere, die speziell bei der Anpassung der Datenbanken Unterstützung bieten.

Insbesondere kann der Prozeß vom Auftragseingang bis zur Teileermittlung transparenter werden, denn in jedem Prozeßstadium (z.B. direkt nach der Umsetzung der Baumuster oder vor der Teileermittlung) lassen sich Aussagen über alle auftretenden Zustände machen.

Beispielweise ließe sich überprüfen:

1. Gibt es nach der Baubarkeitsprüfung korrekte Aufträge, in denen eine bestimmte Codekombination vorkommt? Oder kommt eine Codekombination sogar in allen baubaren Aufträgen vor?

2. Gibt es zu einem bestimmten Baumuster überhaupt korrekte Aufträge?

In den nächsten Abschnitten werden notwendige Vorarbeiten hin zur formalen Spezifikation und Verifikation durchgeführt. Zuerst sollen dazu die bestehenden Datenbanken und die darin enthaltenen Regeln en detail vorgestellt werden.

2.2 Das Datenbankmodell

Die Produktdokumentation ist, wie schon erwähnt, eine relationale, regelbasierte Datenbank. Hier werden nur die für die Verifikation notwendigen Tabellen erläutert, und auch diese Tabellen werden nicht vollständig wiedergegeben, sondern auf das hier Wesentliche beschränkt. So werden beispielsweise terminliche Aspekte nicht berücksichtigt.

Da sämtliche im Rahmen dieser Diplomarbeit vorgenommenen Untersuchungen sich auf die Limousinen der C-Klasse beschränken, werden fast immer diese Tabellen in den Beispielen verwendet.

2.2.1 Umsetzung der Baumuster in Codes

Baumuster werden von den Händlern zur Erstellung von Aufträgen verwendet, Baureihen dienen der betriebsinternen Klassifikation. Jedes Baumuster enthält neben der Baureihe Informationen über Ausführungsart (z.B. Limousine, Kombi), motorbeschreibende Codes (z.B. M111, M24) und ob es für Linkslenker, Rechtslenker oder beide Lenkungsausführungen geeignet ist.

Die Tabelle zur Umsetzung stellt einen eindeutigen Zusammenhang zwischen jedem Baumuster und einem Quadrupel bestehend aus Baureihe, Ausführungsart, Codes und Lenkung her.

Beispielhafte Einträge sind zur Illustration in folgender Tabelle aufgeführt:

Baumuster	Baureihe	Ausführungsart	Codes	Lenkung
02C2020780	C202	FS	M111, M18	–
02C2020262	C202	FW	M112, M24	R
02C2020251	C202	FW	M111, M20, M001	L

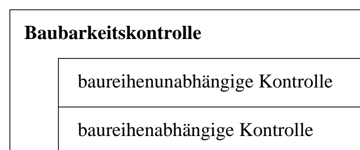
Die Baumuster werden in einem ersten Vorverarbeitungsschritt in die entsprechenden Codes aufgelöst. Aus Baumuster 02C2020251 wird so die Baureihe C202 und die Codemenge FW, M111, M20, M001 und L gewonnen.

Zur Spezifikation der Lenkung sind in den Tabellen die beiden Codes "L" und "R" vorgesehen. Ist ein Eintrag unabhängig von der Lenkung (in der Datenbank mit "–" bezeichnet), so wird angenommen, daß er für beide Lenkungsvarianten zulässig ist. Die wirklich realisierte Lenkungsvariante muß in diesem Fall explizit angegeben werden. Wir betrachten im folgenden allerdings nur solche Baumuster, die eine gültige Lenkungsvariante spezifizieren.

Anmerkung: Durch die Verwendung von Baumustern ist sichergestellt, daß jeder Auftrag einen Baureihen-, Ausführungsart- und Lenkungscode enthält.

2.2.2 Baubarkeitskontrolle

Die Baubarkeitskontrolle besteht aus zwei unabhängigen Kontrollinstanzen, die auch in unterschiedlichen Tabellen verwaltet werden.



Baureihenunabhängige Kontrolle

Die baureihenunabhängige Kontrolle faßt Ausschlüsse und Bedingungen zusammen, die für jedes Fahrzeug, unabhängig von Baureihe, Ausführungsart und Lenkung, eingehalten werden müssen.

Sie besteht im Wesentlichen aus einer Liste von Codes und diesen zugeordneten Bedingungen, die durch boolesche Formeln angegeben werden. In einem gültigen Auftrag muß dabei die Baubarkeitsbedingung eines jeden Codes, der im Auftrag vorkommt, erfüllt sein. Die Baubarkeitsbedingungen der Codes, die nicht im Auftrag enthalten sind, spielen keine Rolle.

Die baureihennunabhängigen Regeln stehen in einer Tabelle, die vereinfacht und beispielhaft wie folgt aussieht:

Code	Baubarkeitsbedingung
265A	$\neg(\text{M104} \wedge 908)$
315	$(274 \vee 313 \vee 930) \wedge \neg(494 \vee 498 \vee 625)$
677	$954 \wedge \neg(482 \vee 655 \vee 676)$
826	$\neg(494 \vee 498 \vee 623 \vee 625 \vee 823 \vee 825 \vee 828 \wedge \neg(\text{923L} \vee \text{924L}) \vee$ $829 \vee 831 \vee 832 \vee 833 \vee 835 \vee 836 \vee 839 \vee 982)$
994	$\neg\ (494 \vee 498 \vee 625 \vee 997)$

Ein Auftrag, der die Codes 315 und 994 enthielte, müßte also die entsprechenden Baubarkeitsbedingungen

$$(274 \vee 313 \vee 930) \wedge \neg(494 \vee 498 \vee 625)$$

und

$$\neg\ (494 \vee 498 \vee 625 \vee 997)$$

erfüllen. Die baureihenunabhängige Baubarkeitsbedingung eines Codes c wird auch pauschale Codebedingung des Codes c genannt.

Baureihenabhängige Kontrolle

Im Unterschied zur baureihenunabhängigen Baubarkeitskontrolle sind hier die Baubarkeitsbedingungen abhängig von Baureihe, Ausführungsart und Lenkung. Außerdem ist die Struktur der Regeln etwas komplizierter. Ein hierarchisches System bestehend aus Gruppen, Positionen und Varianten ermöglicht eine größere Flexibilität. Die Gesamtstruktur der baureihenabhängigen Baubarkeitskontrolle ist in Abbildung 2.2 dargestellt.

Innerhalb einer Baureihe dienen die Gruppen einer groben Gliederung, durch die Positionen wird eine feinere Aufschlüsselung gewährleistet. Für die formale Spezifikation der Baubarkeit ist eine inhaltliche Unterscheidung zwischen Baureihen, Gruppen und Positionen nicht erforderlich, sie können also zu Tupeln zusammengefaßt werden. Jeder Position innerhalb einer Baureihe und Gruppe ist eindeutig ein Code zugeordnet, die Umkehrung muß aber nicht gelten. Ein Code kann also in mehreren Gruppen und auf mehreren Positionen vorkommen. Code 494 tritt beispielsweise in der Gruppe CLA auf den Positionen 1002-1010 und auf den Positionen 1020 und 1200 auf. Der Code 2XXL kommt sowohl in der Gruppe CAA als auch in der Gruppe CLN auf Position 2200 vor. Ausführungsart und Lenkung sind mit den Varianten verknüpft. So ist jeder Positionsvariante eindeutig eine Lenkung und Ausführungsart zugeordnet; auch hier muß die Umkehrung nicht gelten.

Damit ein Code baubar ist, muß auf jeder seiner Positionen mindestens eine Variante erfüllt sein. Dies gilt für alle Gruppen, in denen der Code vorkommt. Im vereinfachten Beispiel aus Abbildung 2.2 muß daher für einen Auftrag, in dem Code 494 vorkommt, abhängig von der Lenkung entweder eine der beiden Regeln R1 und R3 (Linkslenker) oder eine der beiden Regeln R2 und R3 (Rechtslenker) gelten. Passende Baureihe und Ausführungsart des Auftrags sei in diesem Beispiel vorausgesetzt.

In der entsprechenden Datenbank-Tabelle wird dies wie folgt dargestellt:

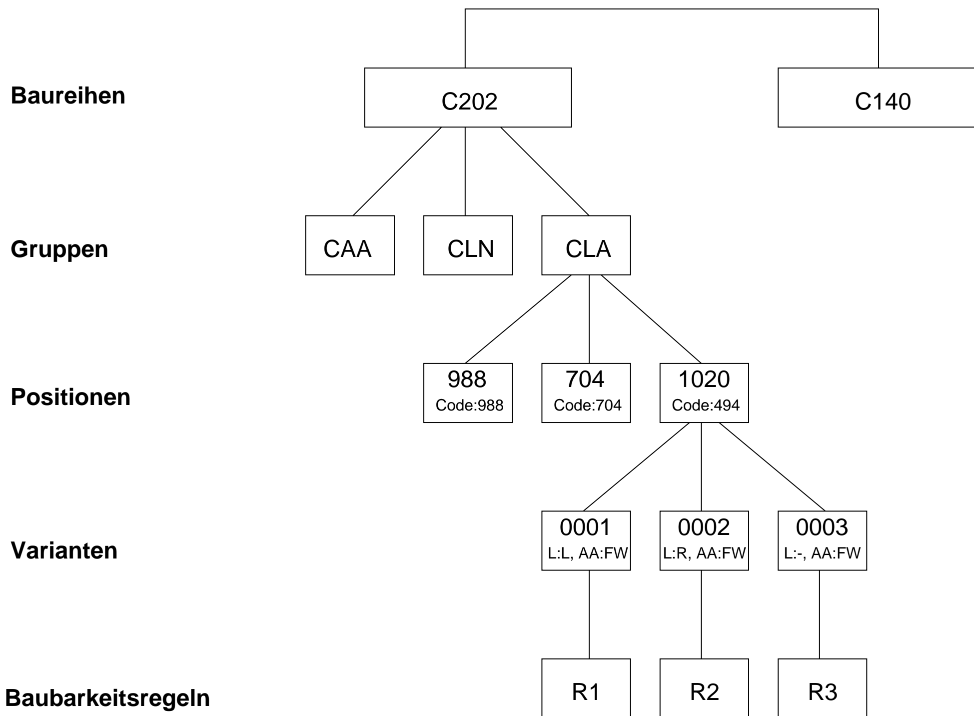


Abbildung 2.2: Struktur der baureihenabhängigen Baubarkeitskontrolle

Code	Gruppe	Pos.	Var.	AA[2]	BR[3]	Lenk.	Baubarkeitsbedingung
494	CLA	1020	0001	FW	C202	L	$M104 \wedge \neg(212 \vee 248 \vee 798 \vee 819 \vee 910 \wedge \neg(460 \vee 957))$
494	CLA	1020	0002	FW	C202	R	$M112 \wedge M28 \wedge \neg(772 \vee 774)$
494	CLA	1020	0003	FW	C202	–	$M111 \wedge (M23 \wedge \neg M001) \wedge \neg(280 \wedge \neg 460 \vee 654 \vee 772)$
400	CAU	0400	0001	FW	C202	–	$(040A \vee 740A) \wedge \neg 808$
423	CAG	0423	0007	FW	C202	R	M112

Zu dieser Tabelle sind noch einige Anmerkungen vonnöten: Die Ausführungsart bezieht sich auf die in jedem Auftrag zwingend enthaltene Grob-Klassifizierung des gewünschten Modells (z.B. Kombi, Cabriolet, Coupé). "FW" kennzeichnet beispielsweise Limousinen. Die Lenkung gibt, wie schon bei der Umsetzung der Baumuster, die gewünschte Lenkungsvariante an. Auch hier bedeutet ein Eintrag der Form "–", daß der entsprechende Datensatz sowohl für Links- als auch für Rechtslenker relevant ist. Ausführungsart und Lenkung wirken als eine Art Filter auf die Baubarkeitsregeln in dem Sinne, daß für einen Auftrag der Ausführungsart a und der Lenkung l nur die Varianten betrachtet werden, in denen die beiden Felder "AA" und "Lenkung" mit a und l übereinstimmen[4].

Es kann also der Fall eintreten, daß für einen Code c entweder überhaupt kein Eintrag in der Datenbank vorhanden ist, oder aufgrund der Filterwirkung von Auftragsart und Lenkung keine relevanten Datensätze übrigbleiben. Tritt dies ein, so wird Code c als nicht baubar angesehen.[5]

Die baureihenabhängige Baubarkeitsbedingung wird manchmal auch nur als "Baubarkeitsbedingung" bezeichnet, insbesondere dann, wenn sie zusammen mit dem Begriff "pauschale Codebedin-

[2]Ausführungsart
[3]Baureihe
[4]Ein Lenkungseintrag der Form "–" stimmt dabei mit jeder Lenkungsvariante überein.
[5]Wie wir später noch sehen werden, gibt es auch von dieser Regel Ausnahmen.

gung" auftritt.

2.2.3 Zusteuerung

Durch die Zusteuerung werden weitere Codes, die nicht explizit im Auftrag enthalten sind, diesem hinzugefügt. Dadurch können zum Beispiel Ausstattungspakete modelliert werden, bei denen ein Code mehrere andere zusammenfaßt.

Wann ein Code zugesteuert wird, ist abhängig von den anderen Codes des Auftrags. Diese Abhängigkeit wird durch Zusteuerbedingungen (boolesche Formeln über den Codes) ausgedrückt.

Die Zusteuerung eines Codes wird allerdings nicht alleine über die Zusteuerbedingung geregelt. Die Baubarkeitsregeln werden bei diesem Vorgang ebenfalls berücksichtigt. So ist jeder Zusteuerbedingung eine Baubarkeitsbedingung zugeordnet, die Teil der baureihenabhängigen Baubarkeitsregel desselben Codes ist. Nur wenn diese und darüberhinaus, falls vorhanden, auch die pauschale Codebedingung erfüllt ist, wird der Code wirklich zugesteuert.

Die Zusteuerbedingungen sind, genau wie die baureihenabhängige Baubarkeitsbedingungen, hierarchisch untergliedert. Die Struktur ist dabei annähernd gleich, wie schon bei der Baubarkeit (Abbildung 2.2) beschrieben: Es gibt Baureihen, Gruppen, Positionen und Varianten, die Zusteuerbedingung eines Codes ist wiederum an die Variante gebunden.

Positionen und Gruppen werden allerdings mit einer anderen Bedeutung verwendet als bei der Baubarkeitskontrolle: Um den Zusteuerungsvorgang eines Codes auszulösen, ist es ausreichend, wenn die Zusteuerungsregel einer einzigen Variante dieses Codes (an einer beliebigen Position und in einer beliebigen Gruppe) erfüllt ist. Die Gruppen und Positionen haben nicht die strukturgebende Funktion wie bei der Baubarkeit, man kann sich daher sämtliche Zusteuerbedingungen als weitestgehend unabhängig voneinander vorstellen. Die Struktur ist nur bei der Zuordnung von Baubarkeitsregeln zu Zusteuerungsregeln relevant.

Ein typischer Tabellenauszug der Zusteuerung könnte so aussehen:

Code	Gruppe	Pos.	Var.	AA	BR	Lenk.	Zusteuerbedingung
668	CAU	0668	0001	FW	C202	–	$494 \vee 498 \vee 625$
471	CFW	0471	0002	FW	C202	–	$\top$
471	CFW	0471	0003	FW	C202	–	$\top$
471	CFW	0471	0006	FW	C202	–	$808 \wedge (513L \vee 517L \vee 2XXL \vee 498)$
471	CFW	0471	0008	FW	C202	–	$808 \wedge (513L \vee 529L \vee 2XXL)$

Wie schon erwähnt, besteht zwischen den Zusteuerbedingungen und den baureihenabhängigen Baubarkeitsbedingungen ein Zusammenhang: Durch Baureihe, Gruppe, Position und Variante kann zu jeder Zusteuerbedingung die ihr zugeordnete Baubarkeitsbedingung eindeutig ausgewählt werden.

Diese Zuordnung von Zusteuerbedingung und Baubarkeitsbedingung wird dazu verwendet, die Zusteuerung zumindest teilweise dahingehend einzuschränken, daß ein baubarer Auftrag durch die Zusteuerung nicht zunichte gemacht wird.

Dies geschieht wie folgt: Wenn die Zusteuerbedingung eines Codes c erfüllt ist, wobei die Zusteuerbedingung zur Gruppe g, Position p, Variante v gehört, so wird dieser nur dann wirklich zugesteuert, wenn auch die baureihenabhängige Baubarkeitsbedingung von Gruppe g, Position p, Variante v erfüllt ist. Falls eine pauschale Codebedingung für Code c vorhanden ist, so muß auch diese erfüllt sein, damit die Zusteuerung in Kraft tritt.

Es sei noch angemerkt, daß durch die Überprüfung der Baubarkeit in dieser Form noch nicht zwangsläufig sichergestellt ist, daß kein baubarer Auftrag durch die Zusteuerung zu einem nicht mehr baubaren Auftrag umgestellt wird.

Intern wird darüberhinaus zwischen einer baureihenbezogenen und einer codegebundenen Zusteuerung unterschieden. Die baureihenbezogene Zusteuerung ist dabei mit keiner Zusteuerbedingung

verknüpft; die Zusteuerung wird in diesem Fall allein durch die Baubarkeitsbedingungen geregelt. Die codegebundene Zusteuerung tritt hingegen immer zusammen mit einer Zusteuerbedingung auf. Der zuerst genannte Fall wird hier durch eine Zusteuerbedingung, die immer wahr ist, simuliert. Damit muß diese Unterscheidung im weiteren nicht mehr vorgenommen werden.

Bevor wir zur Teilebedarfsermittlung übergehen, soll noch kurz auf das derzeit bei Mercedes Benz verwendete Zusteuerungsverfahren eingegangen werden. Bei diesem wird zwischen codegebundener und baureihengebundener Zusteuerung unterschieden. Die codegebundenen Zusteuerungen haben Priorität und werden in einer festgelegten Reihenfolge in mehreren Durchläufen angewendet. Erst danach kommen die baureihengebundenen Regeln zum Zuge, und das auch nur für einen Durchlauf.

2.2.4 Teilebedarfsermittlung

Der Teilebedarf für einen Auftrag wird nach der Zusteuerung und Baubarkeitsprüfung ermittelt. Die Aufschlüsselung in die einzelnen Teile wird ebenfalls anhand codebasierter Regeln vorgenommen.

Ähnlich dem hierarchischen Konzept der Baubarkeitsbedingungen liegt auch hier ein mehrstufiger Aufbau der Stückliste vor. Für jede Baureihe ist sie untergliedert in Module, Positionen und Varianten. Die Module selbst sind wiederum unterteilt in Hauptmodul, Modul und Submodul. Die Module sind nach funktionalen und geometrischen Gesichtspunkten gegliedert, wobei die Genauigkeit der Angaben vom Haupt- zum Submodul zunimmt. An einer Stücklistenposition sind alle Teile, die an einem Ort in derselben Funktion im Fahrzeug alternativ verwendet werden können, zusammengefaßt. Welche Alternativteile an einer Position eingebaut werden können, wird mittels Varianten erfaßt. Jeder Variante ist eine Coderegel zugeordnet, durch die die Auswahl vorgenommen wird. Wie bei der baureihenabhängigen Baubarkeit ist jeder Variante eine Ausführungsart und Lenkung zugeordnet, und auch hier wirken Ausführungsart und Lenkung wie ein Filter auf die Varianten: Nur diejenigen Varianten, deren Ausführungsart und Lenkung mit dem Auftrag übereinstimmen, spielen bei der Teileermittlung eine Rolle.

Die Struktur der Stückliste ist in Abbildung 2.3 schematisch dargestellt, ein typischer Auszug sieht wie folgt aus:

BR	Hauptmod.	Modul	Submod.	Pos.	Var.	AA	Lenk.	Coderegel
C202	08	04	12	1600	0120	FW	L	$\top$
C202	08	04	12	1600	0130	FW	R	$\top$
C202	08	04	12	1600	0140	FW	L	$955 \vee (955 \wedge 494)$
C202	08	04	12	1600	0150	FW	R	955
C202	08	04	12	1800	0010	FW	L	$551 \wedge \neg 494$

Die Regeln zur Teilebedarfsermittlung können nicht direkt ausgewertet werden, da sie nur in einer verkürzten Darstellung ("kurze Coderegel") vorliegen. Bevor also eine Regel zur Anwendung kommen soll, muß diese in die eigentliche boolesche Formel ("lange Coderegel") umgewandelt werden.

Anhand der hierarchischen Struktur läßt sich die Semantik der verkürzten Regeldarstellung, der kurzen Coderegeln, etwas genauer verstehen. Eine formelle Erläuterung der Umwandlung einer kurzen in eine lange Coderegel soll erst in Abschnitt 3.2.6 vorgenommen werden.

Eine kurze Coderegel an Modulposition p, Variante v_i ist nur zusammen mit den anderen Varianten $v_0, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n$ derselben Position, Ausführungsart und Lenkung aussagekräftig. Eine Variante enthält in gewisser Weise die anderen Varianten implizit als Ausschluß. Dadurch kann das Hinzufügen neuer Varianten vereinfacht und die Darstellung der Regeln komprimiert werden. Andererseits erhält man die eigentliche aussagenlogische Formel (lange Coderegel) erst durch eine Transformation der in der Datenbank enthaltenen Regel.

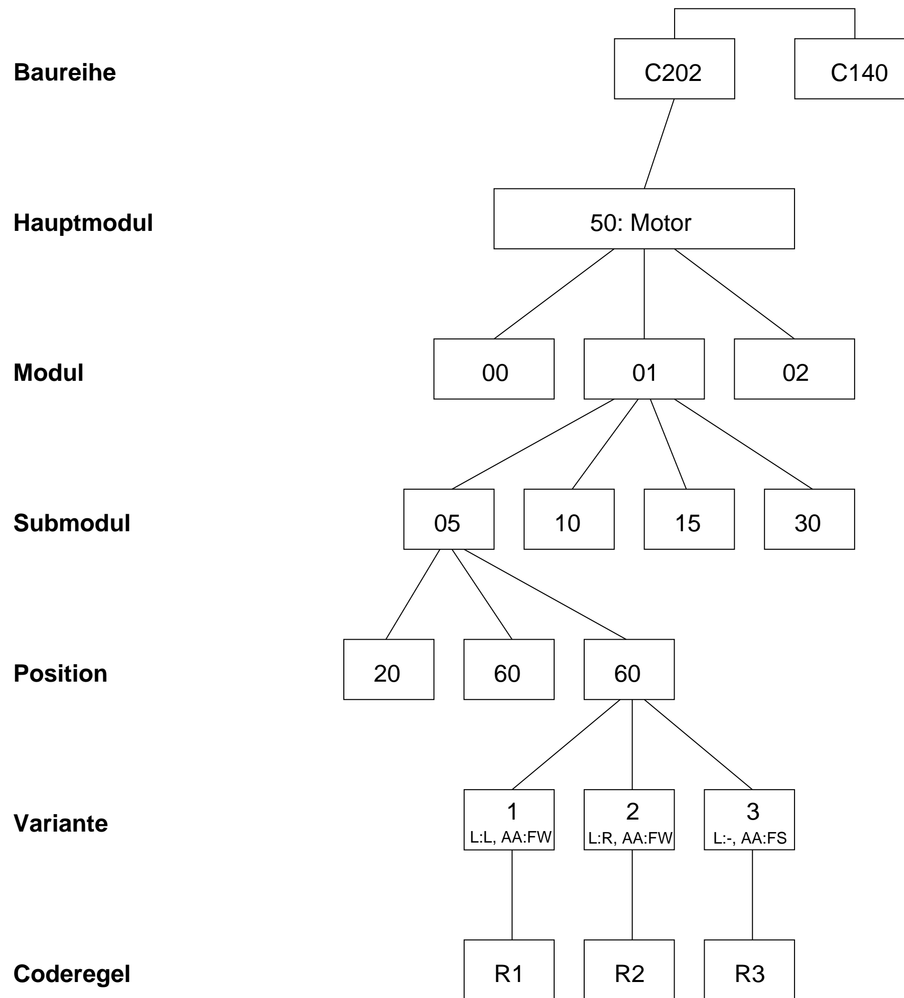


Abbildung 2.3: Struktur der Stückliste

Kapitel 3

Formalisierung der Teilebedarfsermittlung

In den folgenden Abschnitten soll eine formale Spezifikation des gesamten Prozesses vom Auftragseingang bis zur Ermittlung des Teilebedarfs vorgenommen werden. Dazu werden zuerst die später verwendeten grundlegenden Begriffe definiert, danach wird eine Interpretation der Objekte (wie Aufträge, Codes, Regeln) vorgenommen. Damit ist man dann in der Lage, die verschiedenen Stufen bis zur Teileermittlung formal zu beschrieben.

Darauf aufbauend können dann Kriterien angegeben werden, die bestimmte Auftragsmengen beschreiben. Besonderes Augenmerk liegt natürlich auf Kriterien zur Beschreibung der Menge aller baubaren Aufträge sowie aller baubaren, voll zugesteuerten Aufträge. Letzteres ist deshalb von Interesse, da die Teileermittlung von solchen Aufträgen ausgeht.

Die Formalisierung und die Entwicklung der Kriterien sind auf eine Baureihe beschränkt. Diese Einschränkung ändert nichts an den grundsätzlichen Ergebnissen, eine Erweiterung auf mehrere Baureihen ist fast immer durch minimale Änderungen möglich.

3.1 Grundlegende Definitionen

Die hier vorgestellten Definitionen lehnen sich an [Mon76] und [Ric78] an.

Definition 3.1.1 (Formeln) *Sei $V = (x_1, \ldots, x_n)$ eine endliche geordnete Menge (Menge der Prädikatvariablen). Die Menge der aussagenlogischen Formeln $\mathcal{F}(V)$ ist eine von V erzeugte absolut freie Algebra über den Operatoren $\vee$, $\wedge$ (2-stellig), $\neg$ (1-stellig), $\top$ und $\bot$ (Konstanten). Anstatt $\mathcal{F}(V)$ wird oft auch nur $\mathcal{F}$ geschrieben, wenn V aus dem Zusammenhang klar ist. Formeln der Form x_i und $\neg x_i$ werden (positive bzw. negative) Literale genannt.*

Die Operatoren $\vee$, $\wedge$, $\neg$, $\top$ und $\bot$ werden der Reihe nach als Disjunktion, Konjunktion, Negation, wahr und falsch bezeichnet.

Weitere Operationen lassen sich nach Bedarf anhand der vorhergehenden definieren. So kann man die Implikation $F \Rightarrow G$ durch $\neg F \vee G$ und die Äquivalenz $F \Leftrightarrow G$ durch $(F \Rightarrow G) \wedge (G \Rightarrow F)$ ausdrücken.

Alternativ zu obiger Definition kann man Formeln auch rekursiv definieren als die kleinste Menge $\mathcal{F}$ mit

1. $\top, \bot \in \mathcal{F}$,
2. $x \in \mathcal{F}$, falls $x \in V$,

3. $\neg F \in \mathcal{F}$, falls $F \in \mathcal{F}$ und
4. $F \wedge G \in \mathcal{F}$ und $F \vee G \in \mathcal{F}$, falls $F, G \in \mathcal{F}$.

Für endliche geordnete Formelmengen $M = (F_1, \ldots, F_n)$ werden allgemeine Disjunktionen $\bigvee_{1 \leq i \leq j} F_i$ und Konjunktionen $\bigwedge_{1 \leq i \leq j} F_i$ für $j \leq n$ rekursiv definiert durch

$$\begin{aligned}
\bigvee_{1 \leq i \leq 0} F_i &= \bot \\
\bigvee_{1 \leq i \leq j} F_i &= \left(\bigvee_{1 \leq i \leq j-1} F_i \right) \vee F_j \\
\bigwedge_{1 \leq i \leq 0} F_i &= \top \\
\bigwedge_{1 \leq i \leq j} F_i &= \left(\bigwedge_{1 \leq i \leq j-1} F_i \right) \wedge F_j
\end{aligned}$$

Anstatt $\bigvee_{1 \leq i \leq n} F_i$ wird oft auch die Schreibweise $\bigvee_{F \in M} F$ oder auch $\bigvee_{i \in \{1,\ldots,n\}} F_i$ (analog für allgemeine Konjunktionen) verwendet.

Definition 3.1.2 (Variablenbelegung, Bewertungsfunktion) *Die Menge der booleschen Konstanten $\{0,\ 1\}$ wird mit $\mathbf{B}$ bezeichnet. Eine Variablenbelegung ist eine Funktion $b : V \longrightarrow \mathbf{B}$. Zu jeder Variablenbelegung b kann man eine Bewertungsfunktion $b^* : \mathcal{F} \longrightarrow \mathbf{B}$ definieren als Fortsetzung von b durch*

$$\begin{aligned}
b^*(\top) &= 1 \\
b^*(\bot) &= 0 \\
b^*(\neg F) &= 1 - b^*(F) \\
b^*(F \vee G) &= \max(b^*(F), b^*(G)) \\
b^*(F \wedge G) &= \min(b^*(F), b^*(G))
\end{aligned}$$

Definition 3.1.3 (Modell, Tautologie, Erfüllbarkeit) *Eine Variablenbelegung b ist ein Modell einer Formel $F \in \mathcal{F}$, wenn $b^*(F) = 1$. Falls jede Belegung b ein Modell von F ist, so ist F eine Tautologie. Man schreibt dafür auch $\models F$. Gibt es eine Belegung b, die Modell von F ist, so ist F erfüllbar, ansonsten unerfüllbar. Zwei Formeln F und G heißen äquivalent, wenn $b^*(F) = b^*(G)$ für alle Variablenbelegungen b gilt.*

Lemma 3.1.4 *Sei b eine Variablenbelegung. Dann ist $b^*(F \Leftrightarrow G) = 1$ genau dann, wenn $b^*(F) = b^*(G)$.*

Beweis: Nachrechnen. □

Für äquivalente Formeln F und G gilt also $b^*(F \Leftrightarrow G) = 1$ für jede Belegung b, d.h. es gilt $\models F \Leftrightarrow G$. Dies rechtfertigt die doppelte Verwendung des Begriffs “Äquivalenz” sowohl als semantisches als auch als syntaktisches Konzept. Dieser Zusammenhang ist in der nächsten Definition festgehalten.

Definition 3.1.5 *Die Relation $\equiv \subseteq \mathcal{F} \times \mathcal{F}$ ist definiert durch*

$$F \equiv G \quad \textit{genau dann, wenn} \quad \models F \Leftrightarrow G$$

Lemma 3.1.6 $\equiv$ *ist eine Äquivalenzrelation.*

Beweis: Reflexivität, Transitivität und Symmetrie sind leicht zu überprüfen. □

Definition 3.1.7 (Negations-Normalform) *Eine Formel $F \in \mathcal{F}(V)$ ist in Negations-Normalform (NNF), wenn die einzigen negierten Subformeln von F Variablen $x \in V$ sind, d.h. außer bei negativen Literalen treten keine Negationssymbole in F auf.*

Lemma 3.1.8 *Zu jeder Formel $F \in \mathcal{F}(V)$ gibt es eine äquivalente Formel in Negations-Normalform.*

Beweis: Durch Anwendung der folgenden Äquivalenzen läßt sich jede Formel in NNF bringen:

$$\begin{aligned} \neg\top &\equiv \bot \\ \neg\bot &\equiv \top \\ \neg\neg F &\equiv F \\ \neg(F \vee G) &\equiv \neg F \wedge \neg G \\ \neg(F \wedge G) &\equiv \neg F \vee \neg G \end{aligned}$$

Dabei seien $F, G \in \mathcal{F}(V)$. □

Definition 3.1.9 (Disjunktive und konjunktive Normalform) *Sei $V = (x_1, \ldots, x_n)$ eine geordnete Variablenmenge. Eine Formel $F \in \mathcal{F}(V)$ ist in disjunktiver Normalform (DNF), wenn sie von der Gestalt*

$$\bigvee_{1 \le i \le p} \bigwedge_{1 \le j \le n} \phi_{ij}$$

für ein $p \geq 1$ ist, wobei $\phi_{ij} = x_j$ oder $\phi_{ij} = \neg x_j$.

Eine Formel F ist in konjunktiver Normalform (CNF), wenn sie von der Gestalt

$$\bigwedge_{1 \le i \le p} \bigvee_{1 \le j \le n} \phi_{ij}$$

für ein $p \geq 1$ ist, wobei $\phi_{ij} = x_j$ oder $\phi_{ij} = \neg x_j$.

Lemma 3.1.10 *Zu jeder erfüllbaren Formel gibt es eine äquivalente Formel in disjunktiver Normalform. Zu jeder Formel, die keine Tautologie ist, gibt es eine äquivalente Formel in konjunktiver Normalform.*

Beweis: Siehe zum Beispiel [Mon76], Theoreme 8.38 und 8.39. □

Anmerkung: Die Formeln in konjunktiver bzw. disjunktiver Normalform sind nach dieser Definition für minimales p eindeutig.

Zum Teil wird im folgenden auch eine allgemeinere Form der disjunktiven und konjunktiven Normalformen verwendet. Diese sehen dann wie folgt aus:

$$\bigvee_{i \in I} \bigwedge_{j \in J} L_{ij} \quad \text{und} \quad \bigwedge_{i \in I} \bigvee_{j \in J} L_{ij},$$

wobei I und J beliebige endliche Mengen und L_{ij} beliebige (positive oder negative) Literale sind. Im Gegensatz zu obiger Definition sind die Darstellungen dann nicht mehr eindeutig.

Zur Charakterisierung von Auftragsmengen wird es sich als zweckmäßiger erweisen, anstatt der (semantischen) Bewertungsfunktion spezielle boolesche Algebren zu verwenden.

Definition 3.1.11 (Boolesche Algebra) *Eine boolesche Algebra $\mathcal{A} = (A, +, \cdot, -, 0, 1)$ ist eine Algebra, wobei $+$ und $\cdot$ binäre Operatoren auf A, $-$ ein unärer Operator auf A, und $0, 1 \in A$ Konstanten sind. Außerdem müssen die folgenden Bedingungen gelten:*

1. *$+$ und $\cdot$ sind kommutativ und assoziativ,*

2. $x \cdot y + y = y$ *und* $(x + y) \cdot y = y$ *für alle* $x, y \in A$ *(Absorptionsgesetze),*

3. $x \cdot (y + z) = x \cdot y + x \cdot z$ *und* $x + y \cdot z = (x + y) \cdot (x + z)$ *für alle* $x, y, z \in A$ *(Distributivgesetze) und*

4. $x \cdot -x = 0$ *und* $x + -x = 1$ *für alle* $x \in A$.

Besonders interessant sind die folgenden booleschen Algebren. Bei der ersten handelt es sich um eine Mengenalgebra, die zweite stellt einen Zusammenhang zu den (syntaktischen) Formeln her.

Lemma 3.1.12 *Für eine endliche Menge* V *ist* $\mathrm{SBA}(V) = (\mathbf{P}^2(V), \cup, \cap, \sim, \emptyset, \mathbf{P}(V))$ *eine boolesche Algebra.*[1] *Dabei ist die Funktion* $\sim : \mathbf{P}^2(V) \longrightarrow \mathbf{P}^2(V)$ *definiert durch* $\sim M = \mathbf{P}(V) \setminus M$.

Beweis: Die Gesetze für boolesche Algebren überprüft man leicht. □

Lemma 3.1.13 *Für eine endliche Menge* V *ist* $\mathrm{FBA}(V) = (\mathcal{F}(V)/_\equiv, \vee, \wedge, \neg, \bot, \top)$ *eine boolesche Algebra.* FBA *wird auch Lindenbaumalgebra genannt.*

Anmerkung: Man kann leicht überprüfen, daß die Quotientenalgebra $\mathcal{F}(V)/_\equiv$ wohldefiniert ist.

Beweis: Auch hier sind lediglich die Gesetze für boolesche Algebren nachzuweisen, was durch einfaches Nachrechnen leicht möglich ist. □

Der Zusammenhang zwischen Formeln $F \in \mathcal{F}$ über der Variablenmenge V und Elementen der booleschen Mengen-Algebra $\mathrm{SBA}(V)$ wird durch die folgende Definition hergestellt:

Definition 3.1.14 *Die Funktion* $\tau : \mathcal{F}(V)/_\equiv \longrightarrow \mathbf{P}^2(V)$ *ist wie folgt definiert:*[2]

$$\begin{aligned}
\tau(\bot) &= \emptyset \\
\tau(\top) &= \mathbf{P}(V) \\
\tau(x) &= \{M \subseteq V \mid x \in M\} \text{ für } x \in V \\
\tau(\neg F) &= \sim \tau(F) \\
\tau(F \vee G) &= \tau(F) \cup \tau(G) \\
\tau(F \wedge G) &= \tau(F) \cap \tau(G)
\end{aligned}$$

Lemma 3.1.15 *Für eine endliche Menge* V *sind die booleschen Algebren* $\mathrm{SBA}(V)$ *und* $\mathrm{FBA}(V)$ *isomorph. Der Zusammenhang wird durch die Bijektion* $\tau : \mathcal{F}(V)/_\equiv \longrightarrow \mathbf{P}^2(V)$ *hergestellt.*

Beweis: Aus der Definition von τ ergibt sich sofort, daß τ ein Algebren-Homomorphismus ist. Bleibt also zu zeigen, daß τ eine Bijektion ist. Dies ist genau dann der Fall, wenn für den Kern $K = \{F \in \mathcal{F}/_\equiv | \tau(F) = \emptyset\}$ gilt, daß $K = \{\bot\}$. Dazu wählen wir ein beliebiges $F \in \mathcal{F}/_\equiv$ mit $F \not\equiv \bot$ und zeigen, daß $\tau(F) \neq \emptyset$ gilt. Da F erfüllbar ist (Lemma 3.1.4), können wir den Repräsentanten der Äquivalenzklasse in disjunktiver Normalform wählen: $F = \bigvee_{1 \le i \le q} F_i = \bigvee_{1 \le i \le q} \bigwedge_{1 \le j \le n} \phi_{ij}$. Betrachten wir nun ein beliebiges F_i ($i \in \{1, \ldots, q\}$). Dann gilt für dieses F_i:

$$\tau(F_i) = \tau\left(\bigwedge_{j \in I} x_j \wedge \bigwedge_{j \in J} \neg x_j\right) = \bigcap_{j \in I} \tau(x_j) \cap \bigcap_{j \in J} \sim \tau(x_j) = \{\{x_j | j \in I\}\} \neq \emptyset,$$

wobei die disjunkten Mengen I und J durch die DNF festgelegt sind und $I \dot\cup J = \{1, \ldots, n\}$. Damit gilt aber auch $\tau(F) = \bigcup_{1 \le i \le q} \tau(F_i) \neq \emptyset$, womit die Behauptung bewiesen ist. □

[1] $\mathbf{P}(M)$ bezeichnet die Potenzmenge von M, $\mathbf{P}^2(M)$ steht für $\mathbf{P}(\mathbf{P}(M))$.
[2] Die zu $F \in \mathcal{F}$ gehörige Äquivalenzklasse $[F]$ aus $\mathcal{F}/_\equiv$ wird hier verkürzt mit F bezeichnet.

Jeder Teilmenge M einer Variablenmenge V kann man mittels der charakteristischen Funktion von M eine Bewertungsfunktion b_M zuordnen:

$$b_M : V \longrightarrow \mathbf{B} : x \mapsto \begin{cases} 1 & \text{falls } x \in M \\ 0 & \text{sonst} \end{cases}$$

Aufgrund Lemma 3.1.4 gilt für jede Teilmenge M von V und Formeln F und G mit $F \equiv G$, daß $b^*_M(F) = b^*_M(G)$.

Außerdem gilt das folgende Lemma:

Lemma 3.1.16 *Sei $F \in \mathcal{F}(V)$ eine Formel und $M \subseteq V$. Dann ist b_M genau dann ein Modell von F, wenn $M \in \tau(F)$.*

Beweis: Induktion über den Aufbau von F. □

Zusammenfassend kann man die letzten Ergebnisse durch folgendes kommutative Diagramm darstellen:

$$\begin{array}{ccccc} \mathcal{F} & \xrightarrow{[\cdot]} & \mathcal{F}/_{\equiv} & \xrightarrow{\tau} & \mathbf{P}^2(V) \\ {\scriptstyle b^*_M}\downarrow & & {\scriptstyle b^*_M}\downarrow & & {\scriptstyle M\in\cdot}\downarrow \\ \mathbf{B} & \xrightarrow{id} & \mathbf{B} & \xrightarrow{id} & \mathbf{B} \end{array}$$

Die Auswertung einer Formel mittels einer Bewertungsfunktion einer Menge entspricht also der Elementbeziehung auf der Transformierten der Formel.

Im Zusammenhang mit der Zusteuerung werden wir später einzelnen Variablen einer Formel feste Werte zuweisen, sie ansonsten aber unverändert lassen. Die nächste Definition ermöglicht dies.

Definition 3.1.17 (Einschränkung) *Seien $F, G \in \mathcal{F}(V)$, $x, y \in V$ und $c \in \{\top, \bot\}$. Dann ist die Einschränkung $F|_{x=c}$ rekursiv definiert durch*

$$\begin{aligned} \top|_{x=c} &= \top \\ \bot|_{x=c} &= \bot \\ y|_{x=c} &= \begin{cases} c & \textit{falls } y = x \\ y & \textit{sonst} \end{cases} \\ (\neg F)|_{x=c} &= \neg(F|_{x=c}) \\ (F \vee G)|_{x=c} &= F|_{x=c} \vee G|_{x=c} \\ (F \wedge G)|_{x=c} &= F|_{x=c} \wedge G|_{x=c} \end{aligned}$$

3.2 Interpretation wichtiger Begriffe

Wie schon zu Beginn dieses Kapitels erwähnt, beschränken sich alle hier vorgenommenen Interpretationen auf eine einzige Baureihe. Dies stellt jedoch keine wesentliche Einschränkung dar, da eine Erweiterung auf mehrere Baureihen leicht möglich ist.

3.2.1 Codes

Ein Code ist eine boolesche Prädikatvariable. Diese Variablen werden, wie in den Datenbanktabellen, in dem Sinne verwendet, daß eine Belegung mit dem Wert 1 anzeigt, daß der entsprechende Code im Auftrag vorkommt. Eine Belegung mit 0 steht analog für das Fehlen des Codes im Auftrag.

Variablen für Codes, also Prädikatvariablen, werden im folgenden mit $a, b, c, \ldots, x, y, z, \ldots$ bezeichnet, die Menge aller Codes mit $\mathcal{C}$. Verschiedene wichtige Teilmengen von $\mathcal{C}$ und deren Bedeutungen sind in der folgenden Tabelle zusammengefaßt.

Codemenge	Bedeutung
$\mathcal{C}$	Menge aller auftretenden Codes
$\mathcal{C}_P$	Menge aller Codes, für die es eine pauschale Codebedingung gibt
$\mathcal{C}_B$	Menge aller Codes, für die es eine Baubarkeitsbedingung gibt
$\mathcal{C}_Z$	Menge aller Codes, für die es eine Zusteuerbedingung gibt
$\mathcal{C}_L = \{L, R\}$	Lenkungscodes
$\mathcal{C}_L^- = \{L, R, -\}$	Lenkungscodes inkl. unspezifizierter Lenkung
$\mathcal{C}_A$	Menge der Codes, die Ausführungsarten beschreiben
$\mathcal{C}_\top$	Menge der Codes, für die keine Baubarkeitsbed. notwendig ist
$\mathcal{C}_\bot = \mathcal{C} \setminus \mathcal{C}_\top$	Codes, für die eine Baubarkeitsbedingung erforderlich ist

Die beiden Mengen $\mathcal{C}_\top$ und $\mathcal{C}_\bot$ bedürfen noch näherer Erläuterung: Eine Reihe von Codes (Länder-, Ausstattungs- und Farbcodes, bezeichnet mit "xxxL", "xxxA", "xxxO" bzw. "xxxU") sind auch ohne eine baureihenabhängige Baubarkeitsbedingung baubar. Dies steht im Gegensatz zu der üblichen Interpretation, wonach ein Code nur dann baubar ist, wenn diesem eine Baubarkeitsbedingung zugeordnet (und erfüllt) ist. Die Menge $\mathcal{C}_\top$ enthält also jene (Länder-, Ausstattungs-, Farb-) Codes, die auch bei fehlender Baubarkeitsbedingung baubar sind. Alle anderen sind in der Menge $\mathcal{C}_\bot$ zusammengefaßt; für diese bedeutet das Fehlen einer Baubarkeitsbedingung, daß sie ungültig sind, also nicht verbaut werden dürfen.

Die beiden Lenkungscodes "L" und "R" sind immer baubar, also ist $\mathcal{C}_L \subseteq \mathcal{C}_\top$.

3.2.2 Aufträge

Ein Auftrag A ist eine Codemenge, also ist $A \subseteq \mathcal{C}$ bzw. $A \in \mathbf{P}(\mathcal{C})$. Zur Spezifikation werden später auch Auftragsmengen M betrachtet. Eine Auftragsmenge M ist Element der Menge $\mathbf{P}^2(\mathcal{C})$. So ist beispielsweise

$$\{\{x_2, x_{100}, x_{346}\}, \{x_{77}, x_{45}, x_{294}, x_{1021}\}, \{x_5, x_{66}, x_{224}, x_{571}, x_{985}\}\}$$

eine Auftragsmenge, die drei Aufträge beschreibt.

Manchmal wird es sich als zweckmäßiger erweisen, Aufträge als Variablenbelegungen aufzufassen. Dazu wird die charakteristische Funktion der Codemenge verwendet. Für einen Auftrag $A \subseteq \mathcal{C}$ ist die zugehörige Variablenbelegung b_A definiert durch

$$b_A(x) = \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{sonst} \end{cases}$$

Die Codemenge $\mathcal{C}$ wird dabei als fest gegeben vorausgesetzt, also ist b_A eine Funktion $b_A : \mathcal{C} \longrightarrow \mathbf{B}$.

Nimmt man z.B. als (geordnete) Codemenge $\mathcal{C} = (x_1, \ldots, x_5)$ und als Auftrag $A = \{x_2, x_5\}$ an, so ist $b_A = (0, 1, 0, 0, 1)$.

3.2.3 Selektionsfunktionen

Zur Spezifikation der Baubarkeit und Zusteuerung ist es erforderlich, einen Zusammenhang zu den verschiedenen Datenbanktabellen herzustellen. Dies wird durch Selektionsfunktionen bewerkstelligt, die einzelne Einträge in der Datenbank über verschiedene Attribute auswählen können. Die zuvor beschriebene Struktur der Datenbank wird von den Selektionsfunktionen beachtet und schlägt sich in deren Definition nieder.

Selektionsfunktionen sind nicht für alle Attribute notwendig, zum Teil ist eine Kombination verschiedener Felder zur Spezifikation ausreichend. In der nachfolgenden Zusammenstellung ist auch dies berücksichtigt.

Umsetzung der Baumuster in Codes

Die Funktion BMC ordnet jedem Baumuster $m \in \mathcal{R}$ ein Tupel bestehend aus Ausführungsart, Lenkung und zusätzlichen Codes zu. $\mathcal{R}$ bezeichnet dabei die Menge aller Baumuster mit spezifizierter Lenkung.

Selektionsfunktion	Beschreibung
$\mathrm{BMC} : \mathcal{R} \longrightarrow \mathcal{C}_A \times \mathcal{C}_L \times \mathbf{P}(\mathcal{C})$	Baumuster-Codes

Die Baureihen werden hierbei nicht berücksichtigt.

Baureihenunabhängige Kontrolle

Die Funktion PCB ordnet jedem Code $c \in \mathcal{C}_P$, also jedem Code, für den eine pauschale Codebedingung spezifiziert ist, dessen baureihenunabhängige Baubarkeitsbedingung zu. Da nicht für jeden Code $c \in \mathcal{C}$ eine pauschale Codebedingung vorhanden sein muß, ist $\mathrm{PCB} : \mathcal{C} \longrightarrow \mathcal{F}(\mathcal{C})$ keine totale Funktion. Für Codes $c \in \mathcal{C} \setminus \mathcal{C}_P$ ist $\mathrm{PCB}(c)$ undefiniert.

Im Zusammenhang mit der Zusteuerung wird die pauschale Codebedingung ebenfalls benötigt. Dort muß, damit Code c zugesteuert wird, dessen pauschale Codebedingung, sofern sie vorhanden ist, erfüllt sein. Eine nicht vorhandene pauschale Codebedingung wird immer als erfüllt betrachtet. Es liegt also nahe, durch Erweiterung von PCB auf alle Codes eine passende totale Funktion zur Verfügung zu stellen. Diese totale Funktion $\mathrm{PCB}^\top$ wird definiert durch

$$\mathrm{PCB}^\top : \mathcal{C} \longrightarrow \mathcal{F}(\mathcal{C}) : x \mapsto \begin{cases} \mathrm{PCB}(x) & \text{falls } x \in \mathcal{C}_P \\ \top & \text{sonst} \end{cases}$$

Damit kann man sich folgender Funktionen bedienen, um Selektionen aus der Datenbanktabelle zur baureihenunabhängigen Baubarkeit vorzunehmen:

Selektionsfunktion	Beschreibung
$\mathrm{PCB} : \mathcal{C}_P \longrightarrow \mathcal{F}(\mathcal{C})$	pauschale Codebedingung
$\mathrm{PCB}^\top : \mathcal{C} \longrightarrow \mathcal{F}(\mathcal{C})$	erweiterte pauschale Codebedingung

Baureihenabhängige Kontrolle

Entsprechend der hierarchischen Struktur ist bei der baureihenabhängigen Kontrolle nicht nur eine Selektionsfunktion für die Baubarkeitsregel vonnöten, sondern auch weitere strukturbeschreibende Funktionen.

Zuerst soll ein Hilfsprädikat $=_L$ definiert werden, das dazu dient, die Lenkungsvariante "–" (unabhängig von der Lenkung) zu berücksichtigen. Dieses Prädikat wird später bei der Baubarkeitskontrolle, Zusteuerung und Teilebedarfsermittlung benötigt.

Hilfsprädikat	Beschreibung
$=_L \;\subseteq\; \mathcal{C}_L^- \times \mathcal{C}_L$	passende Lenkung

Dabei ist $=_L \subseteq \mathcal{C}_L^- \times \mathcal{C}_L$ definiert als die kleinste Relation mit:

$$L =_L L \qquad\qquad - =_L L$$
$$R =_L R \qquad\qquad - =_L R$$

In der folgenden Tabelle steht $\mathcal{G}$ für die Menge aller Gruppen, die in der Datenbanktabelle zur baureihenabhängigen Baubarkeitskontrolle auftreten. $\mathcal{P}$ steht entsprechend für die Menge aller auftretenden Positionen und $\mathcal{V}$ für die Menge aller auftretenden Varianten.

Selektionsfunktion	Beschreibung
$\mathrm{Pos}_B : \mathcal{C} \longrightarrow \mathbf{P}(\mathcal{G} \times \mathcal{P})$	Gruppen und Positionen eines Codes
$\mathrm{Var}_B : \mathcal{G} \times \mathcal{P} \times \mathcal{C}_A \times \mathcal{C}_L \longrightarrow \mathbf{P}(\mathcal{V})$	Varianten einer Gruppe, Position, AA und Lenkung
$\mathrm{Var}_B^- : \mathcal{G} \times \mathcal{P} \times \mathcal{C}_A \times \mathcal{C}_L^- \longrightarrow \mathbf{P}(\mathcal{V})$	Var. einer Gruppe, Pos., AA und Lenk. (inkl. "–")
$\mathrm{BKB} : \mathcal{G} \times \mathcal{P} \times \mathcal{V} \longrightarrow \mathcal{F}(\mathcal{C})$	Baubarkeitsbed. einer Gruppe, Pos. und Variante

Die Funktion $\mathrm{Pos}_B(c)$ liefert Paare bestehend aus Gruppe und Position, an denen eine Baubarkeitsbedingung zu Code c spezifiziert ist. Für eine bestimmte Gruppe g, Position p, Ausführungsart a und Lenkung l (wobei auch "–" als Lenkung erlaubt ist) erhält man durch die Funktion $\mathrm{Var}_B^-(g, p, a, l)$ alle zu dieser Kombination passenden Baubarkeitsvarianten. D.h. aus allen an dieser Position vorhandenen Varianten werden diejenigen selektiert, die mit Ausführungsart und Lenkung übereinstimmen.

Anhand der (Hilfs-)Funktion Var_B^- läßt sich die Funktion Var_B wie folgt definieren:

$$\mathrm{Var}_B : \mathcal{G} \times \mathcal{P} \times \mathcal{C}_A \times \mathcal{C}_L \longrightarrow \mathbf{P}(\mathcal{V}) : (g, p, a, l) \mapsto \bigcup_{l' =_L l} \mathrm{Var}_B^-(g, p, a, l')$$

Var_B läßt also nur die beiden Lenkungvarianten L und R zu, die Variante einer Baubarkeitsregel mit unspezifizierter Lenkung gilt für beide Lenkungsvarianten.

Letztendlich extrahiert die Funktion $\mathrm{BKB}(g, p, v)$ die Baubarkeitsregel der Gruppe g, Position p und Variante v aus der Datenbanktabelle.

Üblicherweise ist keine der Funktionen Var_B^-, Var_B bzw. BKB total, da diese nur für bestimmte Gruppen, Positionen und Varianten definiert sind. Die Funktion Pos_B hingegen ist eine totale Funktion; falls für einen Code keine Baubarkeitsbedingung spezifiziert ist, so liefert sie die leere Menge.

Beispiel:
Wir betrachten den Datenbankauszug zur baureihenabhängigen Baubarkeit aus Abschnitt 2.2.2. Dann ist

$$\begin{aligned}
\mathrm{Pos}_B(494) &= \{(\mathrm{CLA}, 1020)\} \\
\mathrm{Var}_B^-(\mathrm{CLA}, 1020, \mathrm{FW}, \mathrm{L}) &= \{0001\} \\
\mathrm{Var}_B^-(\mathrm{CLA}, 1020, \mathrm{FW}, -) &= \{0003\} \\
\mathrm{Var}_B(\mathrm{CLA}, 1020, \mathrm{FW}, \mathrm{L}) &= \{0001, 0003\} \\
\mathrm{BKB}(\mathrm{CLA}, 1020, 0002) &= \mathrm{M112} \wedge \mathrm{M28} \wedge \neg(772 \vee 774)
\end{aligned}$$

Zusteuerung

Da die Zusteuerung strukturell sehr eng mit der baureihenabhängigen Baubarkeit zusammenhängt, lassen sich fast alle Definitionen von dort übernehmen. Dies gilt für Gruppen, Positionen und Varianten $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ ohne Änderung. Die entsprechenden Selektionsfunktionen Pos_B und Var_B werden hier Pos_Z und Var_Z genannt und unterscheiden sich von den ersteren nur dadurch, daß die mit Z indizierten Versionen die Positionen bzw. Varianten liefern, an denen eine Zusteuerbedingung spezifiziert ist (im Gegensatz zu einer Baubarkeitsbedingung im Falle der mit B indizierten Versionen). Da für jede Zusteuerbedingung auch eine Baubarkeitsregel in der Datenbank vorhanden ist, gilt für die Variantenselektion außerdem, daß $\mathrm{Var}_B(g, p, a, l)$ definiert ist, sofern dies auch für $\mathrm{Var}_Z(g, p, a, l)$ der Fall ist; darüberhinaus ist $\mathrm{Var}_Z(g, p, a, l) \subseteq \mathrm{Var}_B(g, p, a, l)$ für beliebige g, p, a, l.

Die einzige neue Selektionsfunktion ist die zur Auswahl der Zusteuerungsregel, die analog zur Selektion der Baubarkeitsregel weiter oben zu verstehen ist.

Selektionsfunktion	Beschreibung
$\mathrm{ZB} : \mathcal{G} \times \mathcal{P} \times \mathcal{V} \longrightarrow \mathcal{F}(\mathcal{C})$	Zusteuerbedingung

Auch hierzu wieder ein Beispiel anhand des im letzten Kapitel angegebenen Datenbankauszugs (Abschnitt 2.2.3):

Beispiel:

$$\begin{aligned}
\mathrm{Pos}_Z(471) &= \{(\mathrm{CFW}, 0471)\} \\
\mathrm{Var}_Z^-(\mathrm{CFW}, 0471, \mathrm{FW}, \mathrm{L}) &= \emptyset \\
\mathrm{Var}_Z^-(\mathrm{CFW}, 0471, \mathrm{FW}, -) &= \{0002, 0003, 0006, 0008\} \\
\mathrm{Var}_Z(\mathrm{CFW}, 0471, \mathrm{FW}, \mathrm{L}) &= \{0002, 0003, 0006, 0008\} \\
\mathrm{ZB}(\mathrm{CFW}, 0471, 0008) &= 808 \wedge (513\mathrm{L} \vee 529\mathrm{L} \vee 2\mathrm{XXL})
\end{aligned}$$

Teilebedarfsermittlung

Die Selektionsfunktionen die Teileermittlung betreffend lehnen sich ebenfalls eng an die Struktur der Datenbank, wie im vorigen Kapitel beschrieben, an.

Module werden aber nicht in drei hierarchischen Schichten dargestellt, sondern zu einer Schicht zusammengefaßt. Die Bezeichnungen der Module erhält man dann durch Konkatenation der drei Untermodulnamen (in absteigender Reihenfolge).

$\mathcal{M}$ bezeichnet hier die Menge aller Module (einer Baureihe), $\mathcal{P}_S$ die Menge aller Stücklistenpositionen und $\mathcal{V}_S$ die Menge aller Varianten, die in der Stückliste auftreten. Die Selektionsfunktionen sind dann:

Selektionsfunktion	Beschreibung
$\mathrm{Pos}_S : \mathcal{M} \longrightarrow \mathbf{P}(\mathcal{P}_S)$	Positionen eines Moduls
$\mathrm{Var}_S : \mathcal{M} \times \mathcal{P}_S \times \mathcal{C}_A \times \mathcal{C}_L \longrightarrow \mathbf{P}(\mathcal{V}_S)$	Varianten einer Position
$\mathrm{Var}_S^- : \mathcal{M} \times \mathcal{P}_S \times \mathcal{C}_A \times \mathcal{C}_L^- \longrightarrow \mathbf{P}(\mathcal{V}_S)$	Varianten einer Position (Lenk. inkl. "–")
$\mathrm{KCR} : \mathcal{M} \times \mathcal{P}_S \times \mathcal{V}_S \longrightarrow \mathcal{F}(\mathcal{C})$	(kurze) Coderegel

Die Funktion Pos_S liefert die Menge aller Positionen eines Moduls (das durch die Konkatenation von Hauptmodul, Modul und Submodul angegeben wird), die Hilfsfunktion Var_S^- extrahiert aus der Stückliste alle Varianten einer Modulposition, abhängig von Ausführungsart und Lenkung. Bei dieser Funktion ist auch die unspezifizierte Lenkung "–" erlaubt. Die zur Teileermittlung verwendete Funktion Var_S ist dagegen auf eine der Lenkungsvarianten "L" oder "R" angewiesen. Sie ist ähnlich der entsprechenden Funktion bei der Baubarkeit definiert:

$$\mathrm{Var}_S : \mathcal{M} \times \mathcal{P}_S \times \mathcal{C}_A \times \mathcal{C}_L \longrightarrow \mathbf{P}(\mathcal{V}_S) : (m, p, a, l) \mapsto \bigcup_{l' =_L l} \mathrm{Var}_S^-(m, p, a, l')$$

Um die (kurze) Coderegel einer bestimmten Variante (durch Position und Modul näher spezifiziert) auszuwählen, wird die Funktion KCR verwendet.

Beispiel:
Wird der Datenbankauszug aus Tabelle 2.2.4 zugrundegelegt, so ergeben die Selektionsfunktionen folgendes Bild:

$$\begin{aligned}
\mathrm{Pos}_S(080412) &= \{1600, 1800\} \\
\mathrm{Var}_S^-(080412, 1600, \mathrm{FW}, \mathrm{L}) &= \{0120, 0140\} \\
\mathrm{Var}_S^-(080412, 1600, \mathrm{FW}, -) &= \emptyset \\
\mathrm{Var}_S(080412, 1600, \mathrm{FW}, \mathrm{R}) &= \{0130, 0150\} \\
\mathrm{KCR}(080412, 1800, 0010) &= 551 \wedge \neg 494
\end{aligned}$$

3.2.4 Interpretation der Baubarkeit

Baureihenunabhängige Kontrolle

Zuerst soll die baureihenunabhängige Baubarkeit formalisiert werden. Dazu sei nochmal daran erinnert, daß ein Auftrag nur dann baubar ist, wenn für jeden im Auftrag vorkommenden Code dessen pauschale Codebedingung (falls vorhanden) erfüllt ist. Nach den Vorarbeiten des letzten Abschnitts läßt sich dies leicht anhand der Funktion PCB überprüfen.

Ein Auftrag $A \subseteq \mathcal{C}$ muß demnach, um baubar zu sein (nur baureihenunabhängige Baubarkeit), die folgende Bedingung erfüllen:

$$\boxed{\text{Für alle } c \in A \text{ ist } b_A^*(\mathrm{PCB}^\top(c)) = 1.}$$

Damit wird ausgedrückt, daß eine Bewertung der Formel $\mathrm{PCB}^\top(c)$ mit der dem Auftrag A zugeordneten Variablenbelegung b_A für jeden im Auftrag vorkommenden Code $c \in A$ den Wert 1 liefern muß.

Anmerkung: Falls für einen Code keine pauschale Codebedingung spezifiziert ist, so ist dieser Code baubar. Daher wird die Funktion $\mathrm{PCB}^\top$ anstelle der Funktion PCB verwendet.

Beispiel:
Der Auftrag $A = \{x_3, x_{33}, x_{56}\}$ soll auf (baureihenunabhängige) Baubarkeit geprüft werden. Die pauschalen Codebedingungen für die hier relevanten Codes seien:

$$\begin{aligned}
\mathrm{PCB}(x_3) &= \neg(x_4 \vee x_5) \\
\mathrm{PCB}(x_{33}) &= \neg(x_{32} \vee x_{34}) \wedge x_7 \\
\mathrm{PCB}(x_{56}) &= x_3 \vee x_4 \wedge x_{17}
\end{aligned}$$

Die Auswertung der Baubarkeitsregeln für die verschiedenen $c \in A$ mittels b_A^* liefert:

$$\begin{aligned}
b_A^*(\mathrm{PCB}(x_3)) &= 1 - \max(0,0) = 1 \\
b_A^*(\mathrm{PCB}(x_{33})) &= \min(1 - \max(0,0), 0) = 0 \\
b_A^*(\mathrm{PCB}(x_{56})) &= \max(1, \min(0,0)) = 1
\end{aligned}$$

Der Beispielauftrag ist also nicht baubar, da die Baubarkeitsregel von Code x_{33} nicht erfüllt ist.

Baureihenabhängige Kontrolle

Die zu überprüfende Formel ist hier aufgrund der komplizierteren, hierarchischen Struktur nicht ganz so einfach wie im Fall der baureihenunabhängigen Kontrolle.

Wie im letzten Kapitel beschrieben, gibt es eine Unterteilung in Gruppen, Varianten und Positionen. Damit ein im Auftrag vorkommender Code c baubar ist, muß hier gelten:

> Für alle Gruppen, in denen Code c vorkommt:
>> Für alle Positionen, an denen Code c auftritt:
>>> Die Baubarkeitsbedingung mindestens einer Variante,
>>> bei der Lenkung und Ausführungsart mit dem Auftrag
>>> übereinstimmen, muß erfüllt sein.

Darüberhinaus gilt es, die verschiedenen Sonderfälle zu berücksichtigen:

- Kommt ein Code c in der Datenbanktabelle zur baureihenabhängigen Baubarkeit nicht vor (d.h. es gibt keine Gruppe, in der Code c steht), so ist c baubar, falls $c \in \mathcal{C}_\top$. Ist $c \in \mathcal{C}_\bot$, so ist c nicht baubar.

- Gibt es für einen Code c eine Gruppe und Position, an der er in der Datenbank auftritt, sind aber alle Varianten dieser Gruppe und Position wegen der Filterwirkung von Ausführungsart und Lenkung irrelevant, so ist Code c nicht baubar.

Um eine übersichtlichere Darstellung zu ermöglichen, soll zuerst die Baubarkeitsformel $\mathcal{B}(c,a,l)$ eines Codes c definiert werden. Die Parameter a und l beziehen sich auf die im Auftrag vorhandene Ausführungsart und Lenkung.

$$\mathcal{B}(c,a,l) = \bigwedge_{(g,p)\in \mathrm{Pos}_B(c)} \bigvee_{v\in \mathrm{Var}_B(g,p,a,l)} \mathrm{BKB}(g,p,v)$$

Damit ein Auftrag $A \subseteq \mathcal{C}$ die baureihenabhängige Baubarkeitsbedingung erfüllt, muß nun gelten:

Für alle $c \in A$ gilt:

1. Ist $c \in \mathcal{C}_B$, $a,l \in A$ mit $a \in \mathcal{C}_A$ und $l \in \mathcal{C}_L$, so ist $b^*_A(\mathcal{B}(c,a,l)) = 1$ und
2. falls $c \notin \mathcal{C}_B$, so ist $c \in \mathcal{C}_\top$.

$(*)$

Dabei wird vorausgesetzt, daß der Auftrag A genau einen Ausführungsart-beschreibenden Code $a \in \mathcal{C}_A$ und genau einen Lenkungscode $l \in \mathcal{C}_L$ enthält.

Zur Erläuterung der Bedingung $(*)$: Die Formel $\mathcal{B}(c,a,l)$ gibt die oben angegebene Bedingung wieder, daß an allen Gruppen und Positionen eines Codes mindestens eine passende Variante erfüllt sein muß. Dies gilt jedoch nur für Codes, die überhaupt eine Baubarkeitsbedingung haben ($c \in \mathcal{C}_B$, erster Teil von $(*)$). Der zweite Teil von $(*)$ behandelt den oben erwähnten ersten Sonderfall: Ist keine Baubarkeitsbedingung vorhanden, so hängt die Baubarkeit von der Zugehörigkeit zur Menge $\mathcal{C}_\top$ ab.

Der zweite Sonderfall wird implizit durch die Formel $\mathcal{B}(c,a,l)$ berücksichtigt: Ist $\mathrm{Var}_B(g,p,a,l) = \emptyset$ für bestimmte g und p, so ist $\mathcal{B}(c,a,l) = \bot$ und der Auftrag damit nicht baubar.

Beispiel:
Der Auftrag $A = \{x_3, x_{33}, x_{56}, x_{100}, L\}$ soll hinsichtlich baureihenabhängiger Baubarkeit untersucht werden.

Die Codemengen seien dabei wie folgt:

$$\begin{aligned} \mathcal{C}_A &= \{x_{99}, x_{100}\} \\ \mathcal{C}_B &= \{x_3, x_{33}, x_{100}\} \\ \mathcal{C}_\top &= \{x_{56}, L, R\} \end{aligned}$$

Der Datenbankauszug soll die folgenden Baubarkeitsregeln enthalten:

Code	Gruppe	Pos.	Var.	AA	Lenk.	Baubarkeitsbedingung
x_3	g_1	p_1	v_1	x_{100}	L	$x_{33} \wedge \neg x_{56}$
x_3	g_1	p_1	v_2	x_{100}	R	$x_{35} \vee x_{36}$
x_3	g_1	p_1	v_3	x_{100}	–	$\neg x_5 \wedge \neg x_7$
x_{33}	g_1	p_2	v_1	x_{100}	–	$\neg x_{34} \wedge x_{56}$
x_{33}	g_2	p_2	v_1	x_{100}	L	$\top$
x_{100}	g_3	p_1	v_1	x_{100}	–	$\top$

Damit liefern die Selektionsfunktionen:

$$\begin{aligned}
\mathrm{Pos}_B(x_3) &= \{(g_1, p_1)\} \\
\mathrm{Pos}_B(x_{33}) &= \{(g_1, p_2), (g_2, p_2)\} \\
\mathrm{Pos}_B(x_{100}) &= \{(g_3, p_1)\} \\
\mathrm{Var}_B(g_1, p_1, x_{100}, L) &= \{v_1, v_3\} \\
\mathrm{Var}_B(g_1, p_2, x_{100}, L) &= \{v_1\} \\
\mathrm{Var}_B(g_2, p_2, x_{100}, L) &= \{v_1\} \\
\mathrm{Var}_B(g_3, p_1, x_{100}, L) &= \{v_1\} \\
\mathrm{BKB}(g_1, p_1, v_1) &= x_{33} \wedge \neg x_{56} \\
\mathrm{BKB}(g_1, p_1, v_3) &= \neg x_5 \wedge \neg x_7 \\
\mathrm{BKB}(g_1, p_2, v_1) &= \neg x_{34} \wedge x_{56} \\
\mathrm{BKB}(g_2, p_2, v_1) &= \top \\
\mathrm{BKB}(g_3, p_1, v_1) &= \top
\end{aligned}$$

Die Baubarkeitsformeln $\mathcal{B}(c, a, l)$ für die Codes $c \in \mathcal{C}_B = \{x_3, x_{33}, x_{100}\}$, $a = x_{100}$ und $l = L$ sehen dann wie folgt aus:

$$\begin{aligned}
\mathcal{B}(x_3, x_{100}, L) &= \mathrm{BKB}(g_1, p_1, v_1) \vee \mathrm{BKB}(g_1, p_1, v_3) \\
&= (x_{33} \wedge \neg x_{56}) \vee (\neg x_5 \wedge \neg x_7) \\
\mathcal{B}(x_{33}, x_{100}, L) &= \mathrm{BKB}(g_1, p_2, v_1) \wedge \mathrm{BKB}(g_2, p_2, v_1) \\
&= (\neg x_{34} \wedge x_{56}) \wedge \top \\
\mathcal{B}(x_{100}, x_{100}, L) &= \mathrm{BKB}(g_3, p_1, v_1) \\
&= \top
\end{aligned}$$

Daraus erhält man die baureihenabhängige Baubarkeitsbedingung nach $(*)$:

Für alle $c \in \{x_3, x_{33}, x_{56}, x_{100}, L\}$:

1. Ist $c \in \{x_3, x_{33}, x_{100}\}$, so ist $b_A^*(\mathcal{B}(c, x_{100}, L)) = 1$ und
2. falls $c \notin \{x_3, x_{33}, x_{100}\}$, so ist $c \in \{x_{56}, L, R\}$.

Dies kann man vereinfachen zu folgender äquivalenten Bedingung:

$b_A^*(\mathcal{B}(x_3, x_{100}, L)) = 1$ und
$b_A^*(\mathcal{B}(x_{33}, x_{100}, L)) = 1$ und
$b_A^*(\mathcal{B}(x_{100}, x_{100}, L)) = 1$ und

Die Auswertung der letzten drei Ausdrücke mittels b_A liefert:

$$\begin{aligned}
b_A^*(\mathcal{B}(x_3, x_{100}, L)) &= b_A^*((x_{33} \wedge \neg x_{56}) \vee (\neg x_5 \wedge \neg x_7)) \\
&= \max(\min(b_A(x_{33}), 1 - b_A(x_{56})), \min(1 - b_A(x_5), 1 - b_A(x_7))) \\
&= \max(\min(1, 1 - 1), \min(1 - 0, 1 - 0)) = 1 \\
b_A^*(\mathcal{B}(x_{33}, x_{100}, L)) &= b_A^*((\neg x_{34} \wedge x_{56}) \wedge \top) \\
&= \min(\min(1 - b_A(x_{34}), b_A(x_{56})), 1) \\
&= \min(\min(1 - 0, 1), 1) = 1 \\
b_A^*(\mathcal{B}(x_{100}, x_{100}, L)) &= b_A^*(\top) = 1
\end{aligned}$$

Da die Auswertung aller Ausdrücke 1 ergeben hat, ist die Baubarkeitsbedingung $(*)$ erfüllt, der Auftrag A also (hinsichtlich baureihenabhängiger Baubarkeit) baubar.

3.2.5 Interpretation der Zusteuerung

Die Zusteuerung dient der Modifikation von Aufträgen, um beispielsweise Ausstattungspakete zu beschreiben. Modifikationen sind allerdings nur insofern gestattet, als zu den im Auftrag bestehenden Codes weitere hinzugefügt werden. Von einem fest gewählten Auftrag aus können mehrere verschiedene Zusteuerungsschritte möglich sein, die Zusteuerung stellt also (auf der Ebene einzelner Zusteuerungsschritte) keinen funktionalen Zusammenhang zwischen nicht-zugesteuertem und zugesteuertem Auftrag her.

Die Zusteuerung wird im folgenden als Relation zwischen Aufträgen dargestellt. Diese Zusteuerungsrelation sei mit $\longrightarrow_z \subseteq \mathbf{P}(\mathcal{C}) \times \mathbf{P}(\mathcal{C})$ bezeichnet. Dabei soll $A \longrightarrow_z B$ gelten, wenn der Auftrag A durch einen Zusteuerungsschritt zu Auftrag B modifiziert werden kann. Definiert wird $\longrightarrow_z$ anhand der Zusteuerbedingungen der entsprechenden Datenbanktabelle, angereichert durch die zugehörigen Baubarkeitsbedingungen.

Bevor die Zusteuerungsrelation formal definiert wird, soll diese erweiterte Zusteuerbedingung, die die Baubarkeit mit berücksichtigt, genauer untersucht werden. Die erweiterte Zusteuerbedingung eines Codes c, einer Ausführungsart a und einer Lenkungsvariante l wird mit $\mathcal{Z}(c, a, l)$ bezeichnet. $\mathcal{Z}(c, a, l)$ läßt sich dann anhand der Selektionsfunktionen wie folgt definieren:

$$\mathcal{Z}(c, a, l) = \left(\bigvee_{\substack{(g,p) \in \mathrm{Pos}_Z(c) \\ v \in \mathrm{Var}_Z(g,p,a,l)}} (\mathrm{ZB}(g, p, v) \wedge \mathrm{BKB}(g, p, v)) \right) \wedge \mathrm{PCB}^\top(c)$$

Die erweiterte Zusteuerbedingung $\mathcal{Z}(c, a, l)$ ist also erfüllt, wenn irgendeine Zusteuerungsvariante, die ihr zugeordnete Baubarkeitsbedingung und die pauschale Codebedingung von Code c erfüllt sind. Die Baubarkeitsbedingungen beziehen sich dabei auf den nicht zugesteuerten Auftrag. Normalerweise besteht aber kein Unterschied in deren Gültigkeit im Vergleich zum zugesteuerten Auftrag, da der Code c nicht in der Baubarkeits- und pauschalen Codebedingung von Code c vorkommen sollte.

Die Formel $\mathcal{Z}(c, a, l)$ ist wohldefiniert, da $\mathrm{Var}_Z(g, p, a, l) \subseteq \mathrm{Var}_B(g, p, a, l)$. Darüberhinaus ist $\mathcal{Z}(c, a, l) = \perp$, wenn für c keine Zusteuerbedingung vorhanden ist.

Bevor die allgemeine Zusteuerungsrelation definiert wird, soll die Relation $\xrightarrow{c}_z$, eine Einschränkung von $\longrightarrow_z$, beschrieben werden. $A \xrightarrow{c}_z B$ ist für zwei Aufträge genau dann wahr, wenn Auftrag B aus Auftrag A durch Zusteuerung von Code c hervorgeht und diese Zusteuerung erlaubt ist.

Für $c \in \mathcal{C}$ ist $\xrightarrow{c}_z \subseteq \mathbf{P}(\mathcal{C}) \times \mathbf{P}(\mathcal{C})$ die kleinste Relation, für die gilt:

$A \xrightarrow{c}_z B$ falls gilt:

1. $B = A \dot{\cup} \{c\}$ und
2. $c \notin A$ und
3. $a, l \in A$ mit $a \in \mathcal{C}_A$ und $l \in \mathcal{C}_L$ und
4. $b^*_A(\mathcal{Z}(c, a, l)) = 1$.

(∗∗)

Auch hier wird wieder vorausgesetzt, daß der Auftrag A genau einen Ausführungsart-beschreibenden Code $a \in \mathcal{C}_A$ und genau einen Lenkungscode $l \in \mathcal{C}_L$ enthält.

Die einzelnen Bedingungen in (∗∗) haben die folgende Bedeutung:

1. B entspricht dem um den Code c erweiterten Auftrag A.
2. c darf in A nicht vorkommen.
3. Ausführungsart und Lenkung sind in A vorhanden (und werden mit a und l bezeichnet).
4. Die (erweiterte) Zusteuerbedingung von Code c ist bei Auftrag A erfüllt.

Die Zusteuerungsrelation $\longrightarrow_Z$ ist definiert als die Vereinigung aller auf einen Code eingeschränkten Zusteuerungsrelationen, also

$$\longrightarrow_Z \;=\; \bigcup_{c \in \mathcal{C}} \xrightarrow{c}_Z .$$

An einem Beispiel soll die Zusteuerung, wie sie gerade formalisiert wurde, nochmals verdeutlicht werden:

Beispiel:
Gegeben seien die Aufträge $A = \{x_5, x_{17}, x_{56}, x_{100}, L\}$, $B = \{x_5, x_{17}, x_{33}, x_{56}, x_{100}, L\}$ und $C = \{x_3, x_5, x_{17}, x_{56}, x_{100}, L\}$. Es soll überprüft werden, ob $A \longrightarrow_Z B$ und ob $A \longrightarrow_Z C$.

Die Codemenge der Ausführungsarten sei wiederum $\mathcal{C}_A = \{x_{99}, x_{100}\}$.

Die Baubarkeitsregeln seien dieselben wie in obigem Beispiel zur Baubarkeit, pauschale Codebedingungen seien keine vorhanden, die Zusteuerungsregeln entsprechen der folgenden Tabelle:

Code	Gruppe	Pos.	Var.	AA	Lenk.	Zusteuerbedingung
x_3	g_1	p_1	v_3	x_{100}	-	$\top$
x_{33}	g_1	p_2	v_1	x_{100}	-	$x_{17} \wedge \neg x_{32}$

Dann erhält man durch die Selektionsfunktionen (die Baubarkeitsregeln sind der Vollständigkeit halber mit angegeben):

$$\begin{aligned}
\text{Pos}_Z(x_3) &= \{(g_1, p_1)\} \\
\text{Pos}_Z(x_{33}) &= \{(g_1, p_2)\} \\
\text{Var}_Z(g_1, p_1, x_{100}, L) &= \{v_3\} \\
\text{Var}_Z(g_1, p_2, x_{100}, L) &= \{v_1\} \\
\text{ZB}(g_1, p_1, v_3) &= \top \\
\text{ZB}(g_1, p_2, v_1) &= x_{17} \wedge \neg x_{32} \\
\text{BKB}(g_1, p_1, v_3) &= \neg x_5 \wedge \neg x_7 \\
\text{BKB}(g_1, p_2, v_1) &= \neg x_{34} \wedge x_{56}
\end{aligned}$$

Man erhält damit die erweiterten Zusteuerbedingungen $\mathcal{Z}(c, a, l)$ für $a = x_{100}$, $l = L$ und jeden Code $c \in \{x_3, x_{33}\}$:

$$\begin{aligned}
\mathcal{Z}(x_3, x_{100}, L) &= \text{ZB}(g_1, p_1, v_3) \wedge \text{BKB}(g_1, p_1, v_3) \wedge \text{PCB}^{\top}(x_3) \\
&= \top \wedge (\neg x_5 \wedge \neg x_7) \wedge \top \\
\mathcal{Z}(x_{33}, x_{100}, L) &= \text{ZB}(g_1, p_2, v_1) \wedge \text{BKB}(g_1, p_2, v_1) \wedge \text{PCB}^{\top}(x_{33}) \\
&= (x_{17} \wedge \neg x_{32}) \wedge (\neg x_{34} \wedge x_{56}) \wedge \top
\end{aligned}$$

Es ist klar, daß Auftrag B aus Auftrag A nur durch Zusteuerung von Code x_{33} und Auftrag C nur durch Zusteuerung von x_3 erhalten werden kann. Also muß man feststellen, ob $A \xrightarrow{x_{33}}_Z B$ und ob $A \xrightarrow{x_3}_Z C$. Um dies zu erreichen, müssen die vier Punkte aus (**) überprüft werden.

Betrachten wir zuerst den Fall, ob A durch die Zusteuerung zu B transformiert werden kann. Die Punkte 1 bis 3 sind alle erfüllt. Man wählt dazu $a = x_{100}$ und $l = L$. Für den vierten Punkt muß man die Bewertung von $Z(x_{33}, x_{100}, L)$ unter b_A berechnen. Diese liefert:

$$\begin{aligned}
b_A^*(Z(x_{33}, x_{100}, L)) &= b_A^*((x_{17} \wedge \neg x_{32}) \wedge (\neg x_{34} \wedge x_{56}) \wedge \top) \\
&= \min(\min(\min(b_A(x_{17}), 1 - b_A(x_{32})), \min(1 - b_A(x_{34}), b_A(x_{56})))), 1) \\
&= \min(\min(\min(1, 1 - 0), \min(1 - 0, 1)), 1) = 1
\end{aligned}$$

Also sind alle vier Punkte von (∗∗) erfüllt, es gilt also $A \longrightarrow_z B$.

Betrachten wir nun C anstelle von B. Auch hier sind die ersten drei Kriterien erfüllt, wenn man a und l wie oben (und $c = x_3$) wählt. Die Auswertung von $\mathcal{Z}(x_3, x_{100}, L)$ unter b_A ergibt:

$$\begin{aligned} b_A^*(Z(x_3, x_{100}, L)) &= b_A^*(\top \wedge (\neg x_5 \wedge \neg x_7) \wedge \top) \\ &= \min(\min(1, \min(1 - b_A(x_5), 1 - b_A(x_7))), 1) \\ &= \min(\min(1, \min(1 - 1, 1 - 0)), 1) = 0 \end{aligned}$$

In diesem Fall ist das vierte Kriterium also nicht erfüllt, daher gilt $A \not\longrightarrow_z C$.

Die Zusteuerungskomponente im Auftragsbearbeitungsprozeß sollte idealerweise Zusteuerungsschritte mittels $\longrightarrow_z$ solange durchführen, bis keine weiteren Codes zugesteuert werden können.

Das derzeitige System führt allerdings nur eine feste Anzahl von Zusteuerungsschritten durch, und diese in einer festgelegten Reihenfolge. Dadurch kann es vorkommen, daß keine vollständige Zusteuerung erreicht wird oder die Zusteuerung von der Reihenfolge abhängt. Wir kommen später nochmal auf diese Probleme zurück.

3.2.6 Teilebedarfsermittlung

Die Teilebedarfsermittlung dient der Umsetzung im Auftrag vorkommender Codes in Teile der Stückliste.

Die Varianten der Modulpositionen werden über die Coderegeln angesteuert: Eine Variante wird genau dann ausgewählt, wenn der Auftrag ihre (lange) Coderegel erfüllt. Die Coderegeln liegen allerdings in einer verkürzten Darstellung vor. Wie man daraus die eigentlichen Auswahlformeln gewinnt, soll nun dargestellt werden.

Bevor die Transformation von kurzen in lange Coderegeln beschrieben wird, müssen noch einige Begriffe geklärt werden.

Definition 3.2.1 (Implikant) *Seien $F, G \in \mathcal{F}$. G ist ein Implikant von F, wenn $\models G \Rightarrow F$.*

Im folgenden werden nur spezielle Implikanten G betrachtet, nämlich Konjunktionen von Literalen, d.h. $G = L_1 \wedge \cdots \wedge L_m$, oder die Konstanten $\top$ und $\bot$.

Definition 3.2.2 (Subsumption) *Ein Implikant G subsumiert einen Implikanten H (in Zeichen: $G \trianglerighteq H$), wenn $\models G \Rightarrow H$. $G \triangleright H$ gilt, falls $G \trianglerighteq H$ und $G \not\equiv H$.*

Beispiel:

$$\begin{aligned} x_1 \wedge x_2 \wedge \neg x_3 &\trianglerighteq x_1 \wedge \neg x_3 \\ x_1 \wedge x_2 &\trianglerighteq \top \\ \bot &\trianglerighteq x_1 \\ x_1 &\ntrianglerighteq x1 \wedge x_2 \end{aligned}$$

Das folgende Lemma beschreibt einen wichtigen Aspekt der Subsumption und dient in erster Linie der Begriffsklärung.

Lemma 3.2.3 *Sei F eine Formel in disjunktiver Normalform, $F = K_1 \vee \cdots \vee K_n$, wobei die K_i für $1 \leq i \leq n$ Implikanten sind. Ferner sei $K_i \trianglerighteq K_j, i \neq j$ und $F' = K_1 \vee \cdots \vee K_{i-1} \vee K_{i+1} \vee \cdots \vee K_n$. Dann ist $F \equiv F'$.*

Damit stehen die erforderlichen Begriffe zur Erzeugung der langen Coderegeln zur Verfügung.

Sei $V = \{v_1, \ldots, v_n\}$ die Menge aller Varianten einer Modulposition mit zum Auftrag passender Ausführungsart und Lenkung (wie sie z.B. von der Selektionsfunktion Var_S geliefert wird). Die zu Variante v_i gehörige kurze Coderegel sei r_i, die lange, zu generierende, Coderegel sei mit $\overline{r_i}$ bezeichnet.

Alle r_i seien in disjunktiver Normalform dargestellt, also:

$$r_i = K_1^i \vee \cdots \vee K_{m_i}^i, \qquad m_i \geq 0,$$

womit K_j^i Konjunktionen von Literalen sind. Falls $r_i = \top$, so sei $m_i = 1$ und $K_1^i = \top$, für $r_i = \bot$ sei $m_i = 0$.

Ferner sei zu jedem K_j^i die Menge M_j^i für $1 \leq i \leq n$ und $1 \leq j \leq m_i$ definiert durch

$$M_j^i = \left\{ K_l^k \;\middle|\; 1 \leq k \leq n, k \neq i, 1 \leq l \leq m_k \text{ und } K_j^i \not\trianglerighteq K_l^k \right\}.$$

M_j^i enthält die Implikanten (Konjunktionen von Literalen) aller anderen Varianten ($k \neq i$), die K_j^i nicht subsumiert.

Dann ist die zu r_i gehörende lange Coderegel $\overline{r_i}$ definiert als

$$\overline{r_i} = \bigvee_{1 \leq j \leq m_i} \left(K_j^i \wedge \bigwedge_{K \in M_j^i} \neg K \right).$$

Beispiel:
Seien die folgenden kurzen Coderegeln gegeben. Diese sollen bezüglich Modul, Position, Ausführungsart und Lenkung übereinstimmen, sind also mögliche Varianten einer Position.

$$\begin{aligned} r_1 &= \top \\ r_2 &= x_1 \\ r_3 &= x_1 \vee x_2 \\ r_4 &= x_1 \wedge x_2 \end{aligned}$$

Alle kurzen Coderegeln sind schon in disjunktiver Normalform, es gilt also:

$$\begin{array}{lll} m_1 = 1 & K_1^1 = \top & \\ m_2 = 1 & K_1^2 = x_1 & \\ m_3 = 2 & K_1^3 = x_1 & K_2^3 = x_2 \\ m_4 = 1 & K_1^4 = x_1 \wedge x_2 & \end{array}$$

Die Mengen M_j^i kann man damit berechnen:

$$\begin{aligned} M_1^1 &= \{x_1, x_2, x_1 \wedge x_2\} \\ M_1^2 &= \{x_2, x_1 \wedge x_2\} \\ M_1^3 &= \{x_1 \wedge x_2\} \\ M_2^3 &= \{x_1, x_1 \wedge x_2\} \\ M_1^4 &= \emptyset \end{aligned}$$

Also erhält man als lange Coderegeln:

$$
\begin{aligned}
\overline{r_1} &= \top \wedge (\neg x_1 \wedge \neg x_2 \wedge \neg(x_1 \wedge x_2)) \\
&= \neg x_1 \wedge \neg x_2 \\
\overline{r_2} &= x_1 \wedge (\neg x_2 \wedge \neg(x_1 \wedge x_2)) \\
&= x_1 \wedge \neg x_2 \\
\overline{r_3} &= (x_1 \wedge \neg(x_1 \wedge x_2)) \vee (x_2 \wedge \neg x_1 \wedge \neg(x_1 \wedge x_2)) \\
&= (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \\
\overline{r_4} &= x_1 \wedge x_2 \wedge \top \\
&= x_1 \wedge x_2
\end{aligned}
$$

Anmerkung: Der im aktuellen System verwendete Algorithmus zur Teilebedarfsermittlung unterscheidet sich von dem hier angegebenen. Dies betrifft in erster Linie die Behandlung von negierten Codes.

Nachdem die Generierung der langen Coderegeln nun geklärt ist, soll noch kurz der Zusammenhang zu den Selektionsfunktionen hergestellt werden. Es gilt, die Menge der Varianten einer Stücklistenposition zu bestimmen, die von einem Auftrag ausgewählt werden. Diese Funktionalität wird durch die Funktion $\mathcal{T}$ bereitgestellt.

Sei $m \in \mathcal{M}, p \in \text{Pos}_S$ und $A \subseteq \mathcal{C}$ mit $a, l \in A, a \in \mathcal{C}_A$ und $l \in \mathcal{C}_L$. Ferner sei $V = \text{Var}_S(m, p, a, l)$ und $r_v = \text{KCR}(m, p, v)$ für alle $v \in V$. Dann ist $\mathcal{T}(A, m, p)$ wie folgt definiert:

$$\mathcal{T}(A, m, p) = \{v \in V \mid b^*_A(\overline{r_v}) = 1\}$$

$\mathcal{T}(A, m, p)$ liefert also genau die Varianten, die bei Auftrag A für Modul m und Position p aus der Stückliste ausgewählt werden.

Die Interpretation der Baubarkeit, Zusteuerung und Teilebedarfsermittlung ist damit abgeschlossen. In den nächsten Abschnitten werden verschiedene Kriterien und Eigenschaften hergeleitet, die wichtige Aspekte der einzelnen Verarbeitungsschritte vom Auftragseingang bis zur Teilebedarfsermittlung betreffen.

3.3 Wichtige Kriterien und Eigenschaften

Die folgenden Ausführungen sollen einerseits der Untersuchung bestimmter Eigenschaften der Baubarkeit, Zusteuerung und Teilebedarfsermittlung dienen, andererseits aber auch schrittweise zu einer vollständigen Klassifizierung aller baubaren Aufträge führen.

Ausgehend von den pauschalen Codebedingungen wird durch sukzessive Hinzunahme der baureihenabhängigen Baubarkeitsbedingungen, der Zusteuerung und der Umsetzung von Baumustern in Baureihen ein Kriterium entwickelt, das alle baubaren, voll zugesteuerten Aufträge beschreibt. Man erhält damit Formeln, die alle möglichen Zustände zwischen Baubarkeitskontrolle und Teilebedarfsermittlung charakterisieren.

Hilfreich ist dabei die am Anfang dieses Kapitels entwickelte Funktion τ, die in Verbindung mit Lemma 3.1.16 einen Zusammenhang zwischen Formeln und Auftragsmengen herstellt. Damit wird es ermöglicht, Auftragsmengen, wie zum Beispiel die Menge aller baubaren Aufträge, über eine einzige Formel zu beschreiben.

Definition 3.3.1 *Eine Formel F beschreibt eine Auftragsmenge M genau dann, wenn*

$$\tau(F) = M$$

Lemma 3.3.2 *Wenn F die Auftragsmenge M beschreibt, so gilt*

$$A \in M \text{ genau dann, wenn } b^*_A(F) = 1.$$

Die Umkehrung gilt ebenfalls.

Beweis: Die Behauptung folgt direkt aus Lemma 3.1.16. □

3.3.1 Pauschale Codebedingung

Nach den Vorarbeiten der letzten Abschnitte erhält man leicht die Formel zur Beschreibung aller Aufträge, die sämtliche pauschalen Codebedingungen erfüllen.

Theorem 3.3.3 *Alle Aufträge, die den pauschalen Codebedingungen genügen, werden durch die Formel $\mathcal{B}_P$ beschrieben, wobei*

$$\boxed{\mathcal{B}_P = \bigwedge_{c \in \mathcal{C}_P} (c \Rightarrow \mathrm{PCB}(c)).}$$

Beweis: Ein Auftrag A, der alle pauschalen Codebedingungen erfüllt, hat nach Abschnitt 3.2.4 folgende Eigenschaft:

$$\text{Für alle } c \in A \text{ ist } b^*_A(\mathrm{PCB}^\top(c)) = 1.$$

Es ist also nach Definition 3.3.1 zu zeigen, daß

$$\tau(\mathcal{B}_P) = \left\{ A \mid b^*_A(\mathrm{PCB}^\top(c)) = 1 \text{ für alle } c \in A \right\}. \tag{3.1}$$

Nach Definition von τ gilt:

$$\tau(\mathcal{B}_P) = \tau\left(\bigwedge_{c \in \mathcal{C}_P} (c \Rightarrow \mathrm{PCB}(c))\right) = \bigcap_{c \in \mathcal{C}_P} (\tau(\neg c) \cup \tau(\mathrm{PCB}(c)))$$

Die Mengengleichheit 3.1 wird durch zwei Inklusionen bewiesen:

"$\supseteq$": Sei $A \in \tau(\mathcal{B}_P)$ und $c \in A$, es gilt also $b^*_A(\neg c) = 0$ und damit $A \notin \tau(\neg c)$. Falls $c \notin \mathcal{C}_P$, so ist $\mathrm{PCB}^\top(c) = \top$, es gilt also $b^*_A(\mathrm{PCB}^\top(c)) = 1$. Andernfalls ($c \in \mathcal{C}_P$) gilt insbesondere $A \in \tau(\neg c) \cup \tau(\mathrm{PCB}(c))$ und wegen $A \notin \tau(\neg c)$ auch $A \in \tau(\mathrm{PCB}(c)) = \tau(\mathrm{PCB}^\top(c))$, somit also nach Lemma 3.1.16 $b^*_A(\mathrm{PCB}^\top(c)) = 1$.

"$\subseteq$": Sei $A \subseteq \mathcal{C}$ mit $b^*_A(\mathrm{PCB}^\top(c)) = 1$ für alle $c \in A$. Sei $c \in \mathcal{C}_P$ beliebig. Falls $c \notin A$, so gilt $b^*_A(\neg c) = 1$, also $A \in \tau(\neg c)$. Andernfalls ($c \in A$) ist $b^*_A(\mathrm{PCB}^\top(c)) = b^*_A(\mathrm{PCB}(c)) = 1$. Daher gilt auch $A \in \tau(\mathrm{PCB}(c))$.

□

3.3.2 Baureihenabhängige Baubarkeit

Wenden wir uns als nächstes der baureihenabhängigen Baubarkeit zu. Es gilt ein ähnliches Theorem wie obiges zur baureihenunabhängigen Baubarkeit.

Theorem 3.3.4 *Alle Aufträge, die den baureihenabhängigen Baubarkeitsbedingungen genügen, werden durch die Formel $\mathcal{B}_B$ beschrieben, wobei*

$$\begin{aligned}
\mathcal{B}_B &= \bigwedge_{c\in\mathcal{C}_B}\left(c \Rightarrow \bigwedge_{\substack{(g,p)\in\mathrm{Pos}_B(c)\\ a\in\mathcal{C}_A, l\in\mathcal{C}_L}}\left((a\wedge l) \Rightarrow \bigvee_{v\in\mathrm{Var}_B(g,p,a,l)} \mathrm{BKB}(g,p,v)\right)\right) \wedge \bigwedge_{c\in\mathcal{C}^{\perp}\setminus\mathcal{C}_B}\neg c\\
&= \bigwedge_{\substack{c\in\mathcal{C}_B\\ a\in\mathcal{C}_A, l\in\mathcal{C}_L}}\Big((c\wedge a\wedge l) \Rightarrow \mathcal{B}(c,a,l)\Big) \wedge \bigwedge_{c\in\mathcal{C}^{\perp}\setminus\mathcal{C}_B}\neg c
\end{aligned}$$

Anmerkung: Betrachtet man nur eine Lenkungsvariante l^* und eine Ausführungsart a^*, so vereinfacht sich die Formel $\mathcal{B}_B$ zu

$$\begin{aligned}
&\bigwedge_{c\in\mathcal{C}_B}\left(c \Rightarrow \bigwedge_{(g,p)\in\mathrm{Pos}_B(c)}\ \bigvee_{v\in\mathrm{Var}_B(g,p,a^*,l^*)} \mathrm{BKB}(g,p,v)\right) \wedge \bigwedge_{c\in\mathcal{C}^{\perp}\setminus\mathcal{C}_B}\neg c\\
=\ &\bigwedge_{c\in\mathcal{C}_B}(c \Rightarrow \mathcal{B}(c,a^*,l^*)) \wedge \bigwedge_{c\in\mathcal{C}^{\perp}\setminus\mathcal{C}_B}\neg c
\end{aligned}$$

Beweis: Wenn ein Auftrag A alle baureihenabhängigen Baubarkeitsbedingungen erfüllt, so hat er nach Abschnitt 3.2.4 für jeden Code $c \in A$ die folgenden Eigenschaften:

1. Ist $c \in \mathcal{C}_B$, $a, l \in A$ mit $a \in \mathcal{C}_A$ und $l \in \mathcal{C}_L$, so ist $b_A^*(\mathcal{B}(c,a,l)) = 1$ und
2. falls $c \notin \mathcal{C}_B$, so ist $c \in \mathcal{C}_{\top}$.

Auch hier wird wieder vorausgesetzt, daß jeder Auftrag A genau einen Lenkungscode $l \in \mathcal{C}_L$ und einen Ausführungsart-beschreibenden Code $a \in \mathcal{C}_A$ enthält.

Unter dieser Voraussetzung ist obige Bedingung äquivalent zu

$$\text{Für alle } c \in A \text{ gilt: } \quad c \in \mathcal{C}_B \Rightarrow b_A^*(\mathcal{B}(c,a_A,l_A)) = 1 \quad \text{und} \quad c \notin \mathcal{C}_B \Rightarrow c \in \mathcal{C}^{\top},$$

wobei a_A und l_A die eindeutigen Ausführungsart- und Lenkungscodes des Auftrags A sind.

Also muß nach Definition 3.3.1 gezeigt werden, daß

$$\tau(\mathcal{B}_B) = \big\{A \;\big|\; ((c \in \mathcal{C}_B \Rightarrow b_A^*(\mathcal{B}(c,a_A,l_A)) = 1) \wedge (c \notin \mathcal{C}_B \Rightarrow c \in \mathcal{C}^{\top})) \text{ für alle } c \in A\big\}. \qquad (3.2)$$

Nach Definition von τ gilt:

$$\begin{aligned}
\tau(\mathcal{B}_B) &= \tau\left(\bigwedge_{\substack{c\in\mathcal{C}_B\\ a\in\mathcal{C}_A, l\in\mathcal{C}_L}}\Big((c\wedge a\wedge l) \Rightarrow \mathcal{B}(c,a,l)\Big) \wedge \bigwedge_{c\in\mathcal{C}^{\perp}\setminus\mathcal{C}_B}\neg c\right)\\
&= \bigcap_{\substack{c\in\mathcal{C}_B\\ a\in\mathcal{C}_A, l\in\mathcal{C}_L}}\Big(\tau(\neg c)\cup\tau(\neg a)\cup\tau(\neg l)\cup\tau(\mathcal{B}(c,a,l))\Big) \cap \bigcap_{c\in\mathcal{C}^{\perp}\setminus\mathcal{C}_B}\tau(\neg c)
\end{aligned}$$

Auch hier wird die Mengengleichheit 3.2 durch zwei Inklusionen bewiesen:

"$\supseteq$": Sei $A \in \tau(\mathcal{B}_B)$ und $c \in A$ beliebig. Außerdem seien $a_A, l_A \in A$ die eindeutigen Ausführungsart- und Lenkungscodes des Auftrags. Es gilt also $b_A^*(\neg c) = b_A^*(\neg a_A) = b_A^*(\neg l_A) = 0$. Damit gilt nach Lemma 3.1.16

$$A \notin \tau(\neg c) \cup \tau(\neg a_A) \cup \tau(\neg l_A). \qquad (3.3)$$

Wir unterscheiden nun die beiden Fälle $c \in \mathcal{C}_B$ und $c \notin \mathcal{C}_B$.

$c \in \mathcal{C}_B$: In diesem Fall gilt nach Voraussetzung insbesondere auch $A \in \tau(\neg c) \cup \tau(\neg a_A) \cup \tau(\neg l_A) \cup \tau(\mathcal{B}(c, a_A, l_A))$ und wegen 3.3 $A \in \tau(\mathcal{B}(c, a_A, l_A))$. Mit Lemma 3.1.16 erhält man $b_A^*(\mathcal{B}(c, a_A, l_A)) = 1$.

$c \notin \mathcal{C}_B$: Dann muß wegen 3.3 und der Definition von $\tau(\mathcal{B}_B)$ gelten, daß $c \notin \mathcal{C}^\perp \setminus \mathcal{C}_B$. Also ist $c \in \mathcal{C} \setminus (\mathcal{C}^\perp \setminus \mathcal{C}_B) = (\mathcal{C} \setminus \mathcal{C}^\perp) \cup \mathcal{C}_B = \mathcal{C}^\top \cup \mathcal{C}_B$. Wegen $c \notin \mathcal{C}_B$ erhält man $c \in \mathcal{C}^\top$ und damit die Behauptung.

"$\subseteq$": Sei $A \subseteq \mathcal{C}$ mit $b_A^*(\mathcal{B}(c, a_A, l_A)) = 1$ für alle $c \in A \cap \mathcal{C}_B$ und $c \in \mathcal{C}^\top$ für alle $c \in A \setminus \mathcal{C}_B$. Wir unterscheiden nun zwei Fälle:

1. Sei $c \in \mathcal{C}_B$ beliebig. Ist $c \notin A$, so ist $b_A^*(\neg c) = 1$ und $A \in \tau(\neg c)$. Sei nun also $c \in A$. Dann gilt $b_A^*(\mathcal{B}(c, a_A, l_A)) = 1$ und somit $A \in \tau(\mathcal{B}(c, a_A, l_A))$. Da $a_A \in \mathcal{C}_A$ eindeutig ist, gilt für alle $a \in \mathcal{C}_A \setminus \{a_A\}$, daß $a \notin A$, also $b_A^*(\neg a) = 1$ und somit $A \in \tau(\neg a)$. Analog erhält man für alle Lenkungscodes $l \in \mathcal{C}_L, l \neq l_A$, daß $A \in \tau(\neg l)$.
2. Sei nun $c \in \mathcal{C}^\perp \setminus \mathcal{C}_B$. Falls $c \notin A$, so gilt wiederum $A \in \tau(\neg c)$. Der andere Fall $c \in A$ kann nicht auftreten, denn aus $c \in A$ folgte nach Voraussetzung $c \in \mathcal{C}^\top$, und daher $c \notin \mathcal{C}^\perp$, ein Widerspruch.

A ist also in jeder der einzelnen Schnittmengen enthalten, also auch in $\tau(\mathcal{B}_B)$.

□

3.3.3 Einschränkung der Lenkung und Ausführungsart

Im folgenden sollen Formeln entwickelt werden, die sicherstellen, daß jeder Auftrag genau einen Lenkungscode $l \in \mathcal{C}_L$ und genau einen Ausführungsart-Code $a \in \mathcal{C}_A$ enthält.

Theorem 3.3.5 *Die Menge aller Aufträge, die genau einen Lenkungscode enthalten, wird durch die Formel $\mathcal{B}_L$ beschrieben, wobei*

$$\mathcal{B}_L = \bigwedge_{\substack{l_1, l_2 \in \mathcal{C}_L \\ l_1 \neq l_2}} \neg(l_1 \wedge l_2) \quad \wedge \quad \bigvee_{l \in \mathcal{C}_L} l$$

Die Menge aller Aufträge, die genau einen Ausführungsart-Code enthalten, wird durch die Formel $\mathcal{B}_A$ beschrieben, wobei

$$\mathcal{B}_A = \bigwedge_{\substack{a_1, a_2 \in \mathcal{C}_A \\ a_1 \neq a_2}} \neg(a_1 \wedge a_2) \quad \wedge \quad \bigvee_{a \in \mathcal{C}_A} a$$

Beweis: Es ist leicht nachzuprüfen, daß die angegeben Formeln die gewünschten Auftragsmengen beschreiben. □

3.3.4 Die Baubarkeitsformel

Nimmt man die Ergebnisse der letzten Abschnitte zusammen, so erhält man eine Formel, die alle baubaren Aufträge beschreibt. Dabei sind beide Baubarkeitskriterien, also baureihenabhängig und baureihenunabhängig, ebenso berücksichtigt wie die naheliegenden Nebenbedingungen, daß ein Auftrag genau einen Lenkungscode und genau einen Ausführungsartcode enthalten muß.

Theorem 3.3.6 *Die Menge aller baubaren Aufträge (baureihenabhängige und -unabhängige Baubarkeit), die genau einen Lenkungs- und genau einen Ausführungsart-beschreibenden Code enthalten, wird durch die Formel $\mathcal{B}_K$ beschrieben, wobei*

$$\mathcal{B}_K = \mathcal{B}_P \wedge \mathcal{B}_B \wedge \mathcal{B}_L \wedge \mathcal{B}_A.$$

Beweis: Die Behauptung folgt direkt aus den Theoremen 3.3.3, 3.3.4 und 3.3.5, zusammen mit der Definition von τ und Definition 3.3.1. □

Wenden wir uns nun der Zusteuerung zu.

3.3.5 Zusteuerung

Das Ziel dieses Abschnitts ist es, eine Formel herzuleiten, die alle voll zugesteuerten Aufträge beschreibt. Mit dieser Formel und der Baubarkeitsformel des letzten Abschnitts erhält man so ein Kriterium, das entscheidet, ob ein Auftrag im Verarbeitungsprozeß zwischen Baubarkeitskontrolle (also nach der Zusteuerung) und Teilebedarfsermittlung auftreten kann. Damit wird letztendlich das Auffinden von unnötigen Teilen oder Mehrdeutigkeiten in der Stückliste ermöglicht.

Bevor wir uns diesem Kriterium zuwenden, sollen noch einige Eigenschaften der Zusteuerungsrelation untersucht werden. Die folgende Darstellung ist an [Bün97] angelehnt.

Definition 3.3.7 (transitive und transitiv-reflexive Hülle) *Sei M eine Menge und $\longrightarrow\ \subseteq M \times M$ eine irreflexive Relation (auch Reduktionsrelation genannt). Dann ist*

1. *$\longrightarrow^+$, die transitive Hülle von $\longrightarrow$, definiert als die kleinste Relation mit*
   (a) *$\longrightarrow\ \subseteq\ \longrightarrow^+$ und*
   (b) *falls $x \longrightarrow^+ y$ und $y \longrightarrow z$, so auch $x \longrightarrow^+ z$.*
2. *$\longrightarrow^*$, die reflexiv-transitive Hülle von $\longrightarrow$, definiert als die kleinste Relation mit*
   (a) *$x \longrightarrow^* x$ für alle $x \in M$ und*
   (b) *$\longrightarrow^+\ \subseteq\ \longrightarrow^*$.*

Anmerkung: Die Zusteuerungsrelation $\longrightarrow_z$ ist eine Reduktionsrelation.

Definition 3.3.8 (Termination, Normalform) *Eine Reduktionsrelation $\longrightarrow\ \subseteq M \times M$ terminiert, wenn es keine unendliche Kette $x_0 \longrightarrow x_1 \longrightarrow x_2 \longrightarrow x_3 \longrightarrow \dots$ gibt. $x \in M$ heißt reduzibel bezüglich $\longrightarrow$, falls es ein y gibt, so daß $x \longrightarrow y$, ansonsten heißt x irreduzibel. $n \in M$ heißt Normalform von $x \in M$ (bezüglich $\longrightarrow$), falls $x \longrightarrow^* n$ und n ist irreduzibel. n ist dann in ($\longrightarrow$-)Normalform.*

Anmerkung: Ein Auftrag ist genau dann voll zugesteuert, wenn er in $\longrightarrow_z$-Normalform ist.

Lemma 3.3.9 *Die Zusteuerungsrelation $\longrightarrow_z$ terminiert.*

Beweis: Wenn $A \longrightarrow_z B$, so ist nach Definition $B = A \dot{\cup} \{c\}$ für ein $c \in \mathcal{C}$ und $c \notin A$, also ist $|B| = |A| + 1$. Da $\mathcal{C}$ endlich ist, ist $|\mathcal{C}|$ eine obere Schranke von $|A|$ für alle $A \in \mathbf{P}(\mathcal{C})$. □

Theorem 3.3.10 *Die Menge aller voll zugesteuerten Aufträge wird durch die Formel $\mathcal{Z}_V$ beschrieben, wobei*

$$\begin{aligned}\mathcal{Z}_V &= \bigwedge_{\substack{c\in\mathcal{C}_Z\\ a\in\mathcal{C}_A, l\in\mathcal{C}_L}} \left((a\wedge l)\Rightarrow\left(\left(\Big(\bigvee_{\substack{(g,p)\in\mathrm{Pos}_Z(c)\\ v\in\mathrm{Var}_Z(g,p,a,l)}}(\mathrm{ZB}(g,p,v)\wedge\mathrm{BKB}(g,p,v))\Big)\wedge\mathrm{PCB}^\top(c)\right)\Rightarrow c\right)\right)\\ &= \bigwedge_{\substack{c\in\mathcal{C}_Z\\ a\in\mathcal{C}_A, l\in\mathcal{C}_L}} \Big((a\wedge l)\Rightarrow\big(\mathcal{Z}(c,a,l)\Rightarrow c\big)\Big)\end{aligned}$$

Anmerkung: Betrachtet man nur eine Lenkungsvariante l^* und eine Ausführungsart a^*, so vereinfacht sich die Formel $\mathcal{Z}_V$ zu

$$\begin{aligned}&\bigwedge_{c\in\mathcal{C}_Z}\left(\left(\Big(\bigvee_{\substack{(g,p)\in\mathrm{Pos}_Z(c)\\ v\in\mathrm{Var}_Z(g,p,a^*,l^*)}}(\mathrm{ZB}(g,p,v)\wedge\mathrm{BKB}(g,p,v))\Big)\wedge\mathrm{PCB}^\top(c)\right)\Rightarrow c\right)\\ =&\bigwedge_{c\in\mathcal{C}_Z}(\mathcal{Z}(c,a^*,l^*)\Rightarrow c)\end{aligned}$$

Beweis: Ist ein Auftrag A voll zugesteuert, also in $\longrightarrow_Z$-Normalform, so gilt nach 3.2.5, daß es keinen Auftrag B und kein $c\in\mathcal{C}$ gibt, so daß $A \xrightarrow{c}_Z B$. Dies ist genau dann der Fall, wenn für alle $c\in\mathcal{C}\setminus A$ gilt, daß $b^*_A(\mathcal{Z}(c,a_A,l_A))=0$, wobei a_A und l_A die eindeutigen Lenkungs- und Ausführungsart-Codes des Auftrags A sind.

Also muß nach Definition 3.3.1 gezeigt werden, daß

$$\tau(\mathcal{Z}_V) \;=\; \left\{A \,\middle|\, c\in\mathcal{C}\setminus A\Rightarrow b^*_A(\mathcal{Z}(c,a_A,l_A))=0\right\}. \tag{3.4}$$

Nach Definition von τ läßt sich $\tau(\mathcal{Z}_V)$ umformen zu

$$\begin{aligned}\tau(\mathcal{Z}_V) &= \tau\left(\bigwedge_{\substack{c\in\mathcal{C}_Z\\ a\in\mathcal{C}_A, l\in\mathcal{C}_L}}\Big((a\wedge l)\Rightarrow\big(\mathcal{Z}(c,a,l)\Rightarrow c\big)\Big)\right)\\ &= \bigcap_{\substack{c\in\mathcal{C}_Z\\ a\in\mathcal{C}_A, l\in\mathcal{C}_L}}\Big(\tau(\neg a)\cup\tau(\neg l)\cup\tau(\neg\mathcal{Z}(c,a,l))\cup\tau(c)\Big)\end{aligned}$$

Die Gleichheit der Mengen aus (3.4) wird wieder durch die beiden Inklusionen bewiesen:

"$\supseteq$": Sei $A\in\tau(\mathcal{Z}_V)$ und $c\in\mathcal{C}\setminus A$ beliebig. Weiter seien $a_A, l_A\in A$ die eindeutigen Ausführungsart- und Lenkungscodes des Auftrags A. Da $c\notin A$ ist, gilt $b^*_A(c)=0$. Außerdem ist $b^*_A(\neg a_A)=b^*_A(\neg l_A)=0$. Nach Lemma 3.1.16 gilt dann $A\notin\tau(c)\cup\tau(\neg a_A)\cup\tau(\neg l_A)$. Nach Voraussetzung gilt insbesondere $A\in\tau(\neg a_A)\cup\tau(\neg l_A)\cup\tau(\neg\mathcal{Z}(c,a_A,l_A))\cup\tau(c)$ und damit $A\in\tau(\neg\mathcal{Z}(c,a_A,l_A))$. Daraus folgt $b^*_A(\neg\mathcal{Z}(c,a_A,l_A))=1$ und $b^*_A(\mathcal{Z}(c,a_A,l_A))=0$.

"$\subseteq$": Sei $A\subseteq\mathcal{C}$ mit $b^*_A(\mathcal{Z}(c,a_A,l_A))=0$ für alle $c\in\mathcal{C}\setminus A$, $c\in\mathcal{C}_Z$ beliebig. Falls $c\in A$ ist, so gilt $b^*_A(c)=1$, also $A\in\tau(c)$. Andernfalls ($c\notin A$) ist nach Voraussetzung $b^*_A(\mathcal{Z}(c,a_A,l_A))=0$, $b^*_A(\neg\mathcal{Z}(c,a_A,l_A))=1$ und somit $A\in\tau(\neg\mathcal{Z}(c,a_A,l_A))$. Da $a_A\in\mathcal{C}_A$ eindeutig ist, gilt für alle $a\in\mathcal{C}_A\setminus\{a_A\}$, daß $a\notin A$, also $b^*_A(\neg a)=1$ und somit $A\in\tau(\neg a)$. Analog erhält man für alle Lenkungscodes $l\in\mathcal{C}_L, l\neq l_A$, daß $A\in\tau(\neg l)$.

□

3.3.6 Baumuster

Um eine gewisse Vollständigkeit zu erreichen, soll nun noch die Umwandlung der Baumuster in Codes untersucht werden. Ziel dieses Abschnitts ist eine Formel, die genau die Aufträge beschreibt, die aus der Umsetzung eines Baumusters in die ihm zugeordneten Codes entstehen.

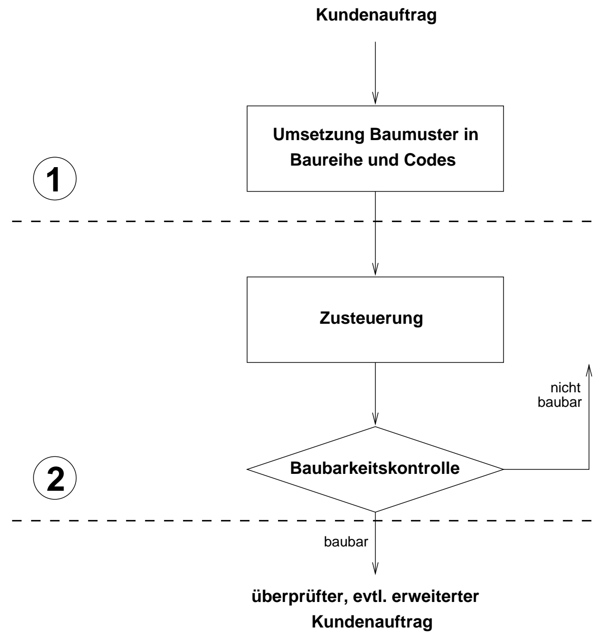
Theorem 3.3.11 *Die Formel $\mathcal{B}_R$ beschreibt genau die Aufträge, die nach Umsetzung eines Baumusters in seine Codes auftreten können. Dabei ist*

$$\mathcal{B}_R = \bigvee_{\substack{r \in \mathcal{R} \\ (a,l,X)=\mathrm{BMC}(r)}} \left(a \wedge l \wedge \bigwedge_{c \in X} c \right)$$

Beweis: Trivial. □

3.4 Baubare Modelle mit Nebenbedingungen

Vergegenwärtigen wir uns nochmals den Auftragsverarbeitungsprozeß:



Wir interessieren uns für die Menge aller Aufträge, die nach der Zusteuerung und nach erfolgreicher Baubarkeitskontrolle auftreten können. An den beiden Stellen ① und ② soll darüberhinaus eine Einschränkung auf Aufträge mit bestimmten Eigenschaften (oder Nebenbedingungen) vorgenommen werden können. Solche Nebenbedingungen werden durch Formeln F_N ausgedrückt. Eine Nebenbedingung F_N drückt dann aus, daß man sich nur für die Aufträge A interessiert, für die an der entsprechenden Stelle $A \in \tau(F_N)$ gilt. Ist $F_N = \top$, so wählt man alle möglichen Aufträge an der gewünschten Schnittstelle aus.

Wenden wir uns zuerst der Schnittstelle ② zu. Die Vorarbeiten der letzten Abschnitte erlauben ohne große Mühe, das gewünschte Kriterium anzugeben.

Theorem 3.4.1 *Sei $F_N \in \mathcal{F}(\mathcal{C})$ eine Nebenbedingung. Dann wird die Menge aller voll zugesteuerten, baubaren Aufträge, die an Stelle ② im Auftragsverarbeitungsprozeß der Nebenbedingung F_N*

genügen, durch die folgende Formel beschrieben:

$$\mathcal{B}_K \wedge \mathcal{B}_R \wedge \mathcal{Z}_V \wedge F_N$$

Beweis: Da $\tau(\mathcal{B}_K)$ die baubaren, $\tau(\mathcal{B}_R)$ die aus den Baumustern hervorgegangenen, $\tau(\mathcal{Z}_V)$ die voll zugesteuerten und $\tau(F_N)$ genau die Aufträge, die der Nebenbedingung F_N genügen, enthält, ist klar, daß der Schnitt über diese Mengen alle Aufträge liefert, die jede dieser vier Bedingungen erfüllen. $\tau(\mathcal{B}_K \wedge \mathcal{B}_R \wedge \mathcal{Z}_V \wedge F_N) = \tau(\mathcal{B}_K) \cap \tau(\mathcal{B}_R) \cap \tau(Z_B) \cap \tau(F_N)$ hat genau diese Eigenschaft. □

Nebenbedingungen an Stelle ① unterscheiden sich von denen an Stelle ② dadurch, daß die Zusteuerung noch Modifikationen an den zugehörigen Aufträgen vornehmen kann. Ein Kriterium wie das vorhergehende ist hier nicht so einfach zu erreichen.

Wir beschränken uns deshalb auf einen Spezialfall der Nebenbedingungen an Stelle ①, nämlich den, daß die Formel F_N keine Negationssymbole enthält. Auch wenn dies eine sehr starke Einschränkung ist, so lassen sich damit doch die aufgrund der Baumuster-Umsetzung gemachten Zusatzannahmen, die an dieser Schnittstelle wirken, beschreiben.

Lemma 3.4.2 *Sei $F \in \mathcal{F}$ eine Formel, die keine Negationssymbole enthält, und $A \overset{c}{\longrightarrow}_z B$ für Aufträge A, B und einen Code $c \in \mathcal{C}$. Dann gilt*

$$A \in \tau(F) \Rightarrow B \in \tau(F).$$

Beweis: Da $A \overset{c}{\longrightarrow}_z B$ ist, gilt $B = A \cup \{c\}$. Für die den Aufträgen A und B zugeordneten Variablenbelegungen gilt dann $b_B(x) \geq b_A(x)$ für alle $x \in \mathcal{C}$. Da die Fortsetzung b^* monoton für alle Operatoren außer der Negation ist und in F keine Negationssymbole enthalten sind, gilt $b^*_B(F) \geq b^*_A(F)$. Da $A \in \tau(F)$ äquivalent zu $b^*_A(F) = 1$ ist, erhält man $b^*_B(F) = 1$ und $B \in \tau(F)$. □

Die Behauptung des Lemmas läßt sich leicht auf $\longrightarrow_z$ und per Induktion auf $\longrightarrow_z^*$ erweitern. Damit hat man das stärkere Ergebnis erhalten, daß eine Nebenbedingung an der Stelle ①, die keine Negationssymbole enthält, sich ohne Veränderung als Nebenbedingung an der Stelle ② auffassen läßt.

Damit überträgt sich die Umsetzung der Baumuster auch ohne Änderung von ① nach ②.

3.5 Reihenfolgeabhängigkeit der Zusteuerung

Eine weitere interessante Eigenschaft der Zusteuerung – neben der Formel, die alle voll zugesteuerten Aufträge beschreibt – ist die Reihenfolgeabhängigkeit. Man kann diese Eigenschaft auch als Frage nach der Eindeutigkeit der $\longrightarrow_z$-Normalformen auffassen. Wünschenswert wäre es, wenn es einen funktionalen Zusammenhang zwischen einem Auftrag A und dessen $\longrightarrow_z$-Normalform geben würde. Damit wäre eine Reihenfolgeabhängigkeit der Zusteuerung ausgeschlossen.

Bevor diese Frage weiter bearbeitet wird, sollen die grundlegenden Begriffe zu deren Formalisierung festgelegt werden.

Definition 3.5.1 ((lokale) Konfluenz) *Eine Reduktionsrelation $\longrightarrow \subseteq M \times M$ ist konfluent, falls für alle $x, y, z \in M$ ein $u \in M$ existiert, so daß*

$$y \longleftarrow^* x \longrightarrow^* z \Rightarrow y \longrightarrow^* u \longleftarrow^* z.$$

$\longrightarrow$ ist lokal konfluent, wenn es für alle $x, y, z \in M$ ein $u \in M$ gibt, so daß

$$y \longleftarrow x \longrightarrow z \Rightarrow y \longrightarrow^* u \longleftarrow^* z.$$

Eine terminierende und konfluente Reduktionsrelation wird kanonisch genannt.

Lemma 3.5.2 (Diamond Lemma, Newman, 1942) *Sei $\longrightarrow$ eine terminierende Reduktionsrelation. Dann ist $\longrightarrow$ konfluent genau dann wenn $\longrightarrow$ lokal konfluent ist.*

Beweis: Siehe zum Beispiel [New42]. □

Lemma 3.5.3 *Sei $\longrightarrow$ eine kanonische Reduktionsrelation über M. Dann hat jedes $x \in M$ eine eindeutige Normalform.*

Beweis: Siehe beispielsweise [Bün97]. □

Um also die Reihenfolgeunabhängigkeit der Zusteuerung – und damit eindeutige Normalformen – sicherzustellen, muß nachgewiesen werden, daß die Relation $\longrightarrow_Z$ lokal konfluent ist.

Um die lokale Konfluenz zu überprüfen sind (mindestens) zwei Wege denkbar:

1. Man könnte alle Aufträge betrachten, die mehrere Zusteuerungen gleichzeitig zulassen. Jede dieser möglichen Zusteuerungen wird, so lange es geht, durch weitere Zusteuerungen fortgesetzt. Erhält man dadurch lauter gleiche Aufträge, so ist die lokale Konfluenz sichergestellt.

2. Man versucht, Kriterien zu finden, unter denen Aufträge mehrere Zusteuerungsmöglichkeiten haben. Danach bildet man die Normalformen der Aufträge, die durch dieses Kriterium beschrieben werden, sofern dies möglich ist.

Die erste Möglichkeit ist zwar prinzipiell möglich, da Aufträge endliche Mengen sind und die Zusteuerung terminiert. Allerdings ist dieser Weg praktisch kaum durchzuführen. Wir suchen also nach einem Kriterium, wann ein Auftrag zwei unterschiedliche Zusteuerungen ermöglicht.

Aus der Literatur ist das Konzept der kritischen Paare bekannt [KB70], das hier, entsprechend modifiziert, angewendet werden soll.

Definition 3.5.4 *Die Formel* $\mathrm{KA}(c_1, c_2)$ *ist für* $c_1, c_2 \in \mathcal{C}, c_1 \neq c_2$ *definiert durch*

$$\mathrm{KA}(c_1, c_2) = \neg c_1 \wedge \neg c_2 \wedge \bigwedge_{a \in \mathcal{C}_A, l \in \mathcal{C}_L} \Big((a \wedge l) \Rightarrow \mathcal{Z}(c_1, a, l) \wedge \mathcal{Z}(c_2, a, l) \Big).$$

KA *steht für "kritische Auftragsmenge", da* KA *Aufträge mit mehreren Zusteuerungsmöglichkeiten beschreiben soll.*

Ein Auftrag A ist Modell dieser Formel, wenn bei A die beiden Zusteuerungen $A \xrightarrow{c_1}_Z B$ und $A \xrightarrow{c_2}_Z C$ für zwei Aufträge B, C möglich sind. Dies wird präziser durch das folgende Theorem ausgedrückt.

Theorem 3.5.5 *Die Menge aller Aufträge, die Zusteuerungen mit zwei (oder mehr) unterschiedlichen Codes erlauben, wird durch die Formel $\mathcal{Z}_R$ beschrieben, wobei*

$$\begin{aligned} \mathcal{Z}_R &= \bigvee_{\substack{c_1, c_2 \in \mathcal{C}_Z \\ c_1 \neq c_2}} \Bigg(\neg c_1 \wedge \neg c_2 \wedge \bigwedge_{a \in \mathcal{C}_A, l \in \mathcal{C}_L} \Big((a \wedge l) \Rightarrow \mathcal{Z}(c_1, a, l) \wedge \mathcal{Z}(c_2, a, l) \Big) \Bigg) \\ &= \bigvee_{\substack{c_1, c_2 \in \mathcal{C}_Z \\ c_1 \neq c_2}} \mathrm{KA}(c_1, c_2) \end{aligned}$$

Die Menge aller Aufträge, die sowohl eine Zusteuerung von Code c_1 als auch von Code c_2 erlauben ($c_1 \neq c_2$), wird durch die Formel $\mathrm{KA}(c_1, c_2)$ *beschrieben.*

Beweis: Wir beweisen den zweiten Teil des Theorems. Der erste folgt dann direkt daraus.

Bei einem Auftrag A ist eine Zusteuerung von sowohl c_1 als auch c_2 möglich, falls $c_1, c_2 \notin A$, $b^*_A(\mathcal{Z}(c_1, a_A, l_A)) = 1$ und $b^*_A(\mathcal{Z}(c_2, a_A, l_A)) = 1$. Dabei sind a_A und l_A die eindeutigen Ausführungsart- und Lenkungscodes des Auftrags.

Nach Definition 3.3.1 ist also für beliebige c_1, c_2 mit $c_1 \neq c_2$ zu zeigen, daß

$$\tau(\mathrm{KA}(c_1, c_2)) = \{A \mid c_1, c_2 \notin A, b^*_A(\mathcal{Z}(c_1, a_A, l_A)) = 1 \text{ und } b^*_A(\mathcal{Z}(c_2, a_A, l_A)) = 1\}. \tag{3.5}$$

$\tau(\mathrm{KA}(c_1, c_2))$ ist nach Definition von τ:

$$\begin{aligned}\tau(\mathrm{KA}(c_1, c_2)) &= \tau\left(\neg c_1 \wedge \neg c_2 \wedge \bigwedge_{a \in \mathcal{C}_A, l \in \mathcal{C}_L} \Big((a \wedge l) \Rightarrow \mathcal{Z}(c_1, a, l) \wedge \mathcal{Z}(c_2, a, l)\Big)\right) \\ &= \tau(\neg c_1) \cap \tau(\neg c_2) \cap \bigcap_{a \in \mathcal{C}_A, l \in \mathcal{C}_L} \Big(\tau(\neg a) \cup \tau(\neg l) \cup (\mathcal{Z}(c_1, a, l) \cap \mathcal{Z}(c_2, a, l))\Big)\end{aligned}$$

Die Mengengleichheit (3.5) wird wiederum durch die beiden Inklusionen bewiesen:

“$\supseteq$”: Sei $A \in \tau(\mathrm{KA}(c_1, c_2))$. Ferner seien $a_A, l_A \in A$ die eindeutig festgelegten Ausführungsart- und Lenkungscodes des Auftrags A. Da nach Voraussetzung $A \in \tau(\neg c_1)$ und $A \in \tau(\neg c_2)$ ist, gilt nach Lemma 3.1.16 $b^*_A(\neg c_1) = b^*_A(\neg c_2) = 1$ und damit auch $c_1, c_2 \notin A$. Außerdem gilt $A \notin \tau(\neg a_A) \cup \tau(\neg l_A)$ wegen $a_A, l_A \in A$. Nach Voraussetzung gilt insbesondere $A \in \tau(\neg a_A) \cup \tau(\neg l_A) \cup (\mathcal{Z}(c_1, a_A, l_A) \cap \mathcal{Z}(c_2, a_A, l_A))$, und daher gilt $A \in \mathcal{Z}(c_1, a_A, l_A)$ und $A \in \mathcal{Z}(c_2, a_A, l_A)$. Eine weitere Anwendung von Lemma 3.1.16 führt zum Ergebnis $b^*_A(\mathcal{Z}(c_1, a_A, l_A)) = 1$ und $b^*_A(\mathcal{Z}(c_2, a_A, l_A)) = 1$.

“$\subseteq$”: Sei $A \subseteq \mathcal{C}$ mit $c_1, c_2 \notin A$ und $b^*_A(\mathcal{Z}(c_1, a_A, l_A)) = b^*_A(\mathcal{Z}(c_2, a_A, l_A)) = 1$. Lemma 3.1.16 liefert wegen $b^*_A(\neg c_1) = b^*_A(\neg c_2) = 1$ sofort $A \in \tau(\neg c_1) \cap \tau(\neg c_2)$. Aus der Voraussetzung folgt analog $A \in \tau(\mathcal{Z}(c_1, a_A, l_A)) \cap \tau(\mathcal{Z}(c_2, a_A, l_A))$. Da $a_A \in \mathcal{C}_A$ eindeutig ist, gilt für alle $a \in \mathcal{C}_A \setminus \{a_A\}$, daß $a \notin A$, also $b^*_A(\neg a) = 1$ und $A \in \tau(\neg a)$. Genauso zeigt man für Lenkungscodes $l \neq l_A$, daß $A \in \tau(\neg l)$.

□

Wir haben damit ein relativ effizientes Verfahren, mehrdeutige Ableitungsmöglichkeiten zu finden und darzustellen.

Aufträge aus den zuvor definierten kritischen Auftragsmengen sind allerdings nur dann problematisch, wenn die beiden Zusteuerungsmöglichkeiten bei weiterer Zusteuerung nicht dasselbe Ergebnis liefern.

Auch dies ist bei kritischen Paaren analog, so daß ein für diese bewiesenes Lemma hier ebenfalls gilt:

Lemma 3.5.6 *Falls für alle $c_1, c_2 \in \mathcal{C}$ und für jeden von $\mathrm{KA}(c_1, c_2)$ beschriebenen Auftrag A ein A' existiert, so daß $A \cup \{c_1\} \longrightarrow^*_z A' \longleftarrow^*_z A \cup \{c_2\}$ gilt, so ist $\longrightarrow_z$ (lokal) konfluent.*

Beweis: Lemma 3.5.2 und Theorem 3.5.5. □

Leider läßt sich mit den bisher zur Verfügung gestellten Mitteln die Voraussetzung des Lemmas nicht nachprüfen. Die Formalisierung über Aussagenlogik erlaubt es nicht, zwei Aufträge, die unterschiedlichen Modellen einer Formel entsprächen, gleichzeitig in voller Allgemeinheit zu behandeln.

Man kann einen Spezialfall des Lemmas aber auch mittels Aussagenlogik formalisieren, indem man sich auf $A \cup \{c_1\} \longrightarrow_z A' \longleftarrow_z A \cup \{c_2\}$ beschränkt. Damit kann man unter Umständen Reihenfolgeabhängigkeiten finden, die sich bei weiterer Normalisierung auflösen würden, also keine sind.

Wir beschränken uns also auf die Vertauschbarkeit von zwei aufeinanderfolgenden Zusteuerungsschritten. Führt die Vertauschung zweier aufeinanderfolgenden Zusteuerungen zu einem anderen Auftrag oder ist nach Vertauschung keine Zusteuerung mehr möglich, so hat man es (in unserem Sinne) mit einer Reihenfolgeabhängigkeit zu tun. Zwingend ist diese Abhängigkeit aber nicht. Andererseits hat man, wenn Vertauschungen zu keinen Änderungen führen, die Konfluenz der Zusteuerung nachgewiesen.

Wir versuchen nun also, ein Kriterium anzugeben, wann zwei aufeinanderfolgende Ableitungsschritte vertauscht werden können, ohne daß sich das Ergebnis ändert. Die zuvor definierten kritischen Aufträge sind trotz der Einschränkungen hilfreich.

Lemma 3.5.7 *Seien $c_1, c_2 \in \mathcal{C}$ mit $c_1 \neq c_2$. Die Formel* $\mathrm{RV}(c_1, c_2)$ *beschreibt genau die Aufträge, bei denen eine Zusteuerung von sowohl c_1 als auch c_2 möglich ist, nach Zusteuerung von c_1 aber c_2 auch noch zugesteuert werden kann und umgekehrt.*

$$\mathrm{RV}(c_1, c_2) = \neg c_1 \wedge \neg c_2 \wedge \bigwedge_{a \in \mathcal{C}_A, l \in \mathcal{C}_L} \Bigg((a \wedge l) \Rightarrow \Big(\mathcal{Z}(c_1, a, l) \wedge \mathcal{Z}(c_2, a, l) \wedge \mathcal{Z}(c_1, a, l)|_{c_2=\top} \wedge \mathcal{Z}(c_2, a, l)|_{c_1=\top} \Big) \Bigg)$$

Beweis: Ein Auftrag A läßt eine Zusteuerung von c_1 und c_2 zu, und nach Zusteuerung von c_1 noch die Zusteuerung von c_2 und umgekehrt, wenn gilt:

1. $c_1, c_2 \notin A$,
2. $b^*_A(\mathcal{Z}(c_1, a_A, l_A)) = b^*_A(\mathcal{Z}(c_2, a_A, l_A)) = 1$ und
3. $b^*_{A_1}(\mathcal{Z}(c_2, a_A, l_A)) = b^*_{A_2}(\mathcal{Z}(c_1, a_A, l_A)) = 1$, wobei $A_1 = A \cup \{c_1\}$ und $A_2 = A \cup \{c_2\}$.

Falls

$$b^*_A(\mathcal{Z}(c_1, a_A, l_A)|_{c_2=\top}) = b^*_{A_2}(\mathcal{Z}(c_1, a_A, l_A)) \tag{3.6}$$

gilt (analog für vertauschte c_1 und c_2), so kann man den Beweis wie ähnliche zuvor führen. 3.6 läßt sich leicht durch Induktion beweisen. Es sei hier nur noch angemerkt, daß $b^*_A(c|_{c=\top}) = b^*_A(\top) = 1 = b^*_{A \cup \{c\}}(c)$ gilt. □

Theorem 3.5.8 *Wenn für alle $c_1, c_2 \in \mathcal{C}$ mit $c_1 \neq c_2$*

$$\mathrm{RV}(c_1, c_2) \equiv \mathrm{KA}(c_1, c_2)$$

ist, dann ist die Zusteuerung konfluent.

Beweis: Lemma 3.5.6 zusammen mit Lemma 3.5.7. □

Kapitel 4

Vergleich verschiedener automatischer Beweistechniken

Nach den Ergebnissen des letzten Kapitels sind wir nun in der Lage, die verschiedenen ursprünglich aufgeworfenen Fragen als Erfüllbarkeits- bzw. Tautologieprobleme von aussagenlogischen Formeln darzustellen. Dies gilt für die Feststellung von nicht mehr benötigten Teilen in der Stückliste ebenso wie für die Reihenfolgeabhängigkeit der Zusteuerung und für die anderen Konsistenzkriterien.

Wir bleiben allerdings noch eine Antwort auf die Frage schuldig, wie diese Erfüllbarkeits- und Tautologieprobleme wirklich gelöst werden können. Seit der Arbeit von S. A. Cook [Coo71] ist bekannt, daß das Erfüllbarkeitsproblem der Aussagenlogik NP-vollständig ist, die Laufzeit aller bisher bekannten Algorithmen also im schlimmsten Fall exponentiell mit der Größe der Eingabeformel wächst. Für Probleme der hier betrachteten Größenordnung muß dies zwangsläufig einer praktischen Unlösbarkeit gleichkommen. Allerdings ist durch Cooks Ergebnis nicht ausgeschlossen, daß es für bestimmte Klassen von Formeln doch spezielle Verfahren gibt, die in vernünftiger Zeit Lösungen liefern.

Techniken, die auf unterschiedlichen Gebieten große Erfolge erzielen konnten, umfassen binäre Entscheidungsdiagramme (BDDs) in verschiedenen Varianten, das Verfahren von M. Davis und H. Putnam, Term-(Graph-)Ersetzungssysteme, resolutionsbasierte Verfahren sowie das von Stålmarck patentierte und nach ihm benannte Verfahren.

Eine Untersuchung der Anwendbarkeit dieser Verfahren bei den vorliegenden Formeln und das Herausarbeiten deren Stärken und Schwächen ist Ziel dieses Kapitels.

Ausgangspunkt des Vergleichs ist die im letzten Kapitel vorgestellte Formel $\mathcal{B}_P$, die alle Aufträge beschreibt, die den pauschalen Codebedingungen genügen. Die Wahl dieser Formel stellt einen vernünftigen Kompromiß zwischen noch handhabbarer Größe für die verwendeten Methoden und praktischer Problemnähe dar. In den meisten Fällen ist davon auszugehen, daß die Ergebnisse der Untersuchung von $\mathcal{B}_P$ sich auf andere Formeln (beispielsweise $\mathcal{B}_K$ oder $\mathcal{Z}_V$) übertragen lassen.

Die oben angegebenen Entscheidungsverfahren lassen sich grob in zwei Gruppen einteilen: In solche, die die Erfüllbarkeit von Aussagen feststellen können, und in Verfahren, die die Allgemeingültigkeit von Formeln beweisen. Verfahren der einen Gruppen lassen sich jedoch leicht in Verfahren der anderen Art umändern, da eine aussagenlogische Formel genau dann erfüllbar ist, wenn ihr Negat keine Tautologie ist. Die Vollständigkeit der Aussagenlogik garantiert in jedem Fall eine positive oder negative Antwort.

4.1 Binäre Entscheidungsdiagramme

Binäre Entscheidungsdiagramme (binary decision diagrams, BDDs) stellen Boolesche Funktionen als gerichtete azyklische Graphen dar. Bekannt wurden sie durch R. Bryants Artikel in [Bry86], der auf Ideen von Lee [Lee59] und Akers [Ake78] aufbaut. Die Begriffe OBDD (ordered binary decision diagram) und ROBDD (reduced ordered binary decision diagram) werden im folgenden, wie inzwischen üblich, gleichbedeutend mit dem Begriff BDD verwendet. Wir verstehen unter BDDs immer die reduzierte, geordnete Variante.

Obwohl der Zusammenhang zwischen Formeln und booleschen Funktionen intuitiv ist, sei dieser Zusammenhang des besseren Verständnisses wegen noch etwas näher beleuchtet.

Sei $V = (v_1, \ldots, v_n)$ eine geordnete (Prädikat-)Variablenmenge und $F \in \mathcal{F}(V)$. Ferner sei für alle $\vec{x} = (x_1, \ldots, x_n) \in \mathbf{B}^n$ die Bewertungsfunktion $b_{\vec{x}}$ definiert durch $b_{\vec{x}}(v_i) = x_i$ für $1 \leq i \leq n$. Dann ist die der Formel F zugeordnete boolesche Funktion f definiert durch

$$f : \mathbf{B}^n \longrightarrow \mathbf{B} : (x_1, \ldots, x_n) \mapsto b^*_{\vec{x}}(F).$$

Der für BDDs zentrale Begriff der Einschränkung, wie schon in 3.1.17 angegeben, überträgt sich auf boolesche Funktionen und läßt sich hier sogar noch einfacher darstellen: Für $b \in \mathbf{B}$ ist die Einschränkung $f|_{x_i=b}$ definiert durch

$$f|_{x_i=b}(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n).$$

Die Formelkonstruktoren $\vee, \wedge, \neg, \top$ und $\bot$ lassen sich entsprechend auf boolesche Funktionen übertragen.

Grundlegendes

Wir betrachten boolesche Funktionen, die abhängig sind von den Variablen $(x_1, \ldots, x_n)$. Die Knotenmenge des BDD-Graphen sei mit K bezeichnet und unterteilt in Terminalknoten und Nonterminalknoten. Jedem Nonterminalknoten k ist ein Index $j(k) \in \{1, \ldots, n\}$ zugeordnet, ein Terminalknoten k hat einen Wert $v(k) \in \{0, 1\}$. Außerdem hat jeder Nonterminalknoten k zwei Nachfolger k_L und k_H, wobei ein Nachfolgerknoten k' entweder ein Terminalknoten ist oder ein Nonterminalknoten mit $j(k') > j(k)$. Die von einem BDD-Graphen G mit Wurzelknoten k dargestellte Funktion f_k ist dann

$$f_k(x_1, \ldots, x_n) = \begin{cases} v(k) & \text{falls } k \text{ ein Terminalknoten ist,} \\ (\neg x_i \wedge f_{k_L}(x_1, \ldots, x_n)) \vee (x_i \wedge f_{k_H}(x_1, \ldots, x_n)) & \\ \quad \text{falls } k \text{ ein Nonterminalknoten mit } j(k) = i \text{ ist.} & \end{cases}$$

Für einen (reduzierten) BDD-Graphen gelten darüberhinaus die folgenden Einschränkungen:

1. Für alle Nonterminalknoten k ist $f_{k_L} \not\equiv f_{k_H}$.
2. Es gibt keine unterschiedlichen Knoten k_1, k_2 mit $f_{k_1} \equiv f_{k_2}$.

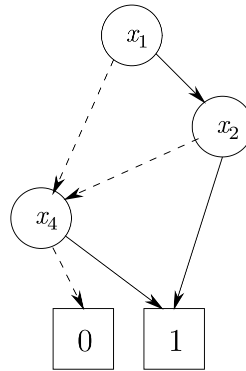
Mit diesen beiden Einschränkungen ist die BDD-Darstellung einer booleschen Funktion eindeutig; trotzdem läßt sich noch jede Funktion darstellen. Einen Beweis dieser Eigenschaft findet man beispielsweise in [Bry86].

Um zu einer gegebenen Funktion den entsprechenden BDD zu erhalten, wendet man rekursiv die Shannon-Expansion [Sha38] an:

$$f = (\neg x_i \wedge f|_{x_i=0}) \vee (x_i \wedge f|_{x_i=1})$$

Die beiden Einschränkungen $f|_{x_i=0}$ und $f|_{x_i=1}$ ergeben dann die L- und H-Nachfolgeknoten. Die weiter oben genannten Nebenbedingungen erhält man dann durch einfache Graph-Reduktionen, wie sie zum Beispiel in [Bry92] angegeben sind.

Beispiel: Wir betrachten die Funktion $f = (x_1 \wedge x_2) \vee x_4$. Der zugehörige BDD-Graph ist dann:



Terminalknoten sind hier als Quadrate, Nonterminalknoten als Kreise dargestellt. L-Nachfolgeknoten sind durch gestrichelte Linien, H-Nachfolger anhand durchgezogener Linien gekennzeichnet.

Stärken und Schwächen

BDDs beziehen ihre Stärke unter anderem aus der eindeutigen Darstellung isomorpher Funktionen im zugeordneten Graph [Moo92]. Im Vergleich zu der Darstellung über Binärbäume, die durch exponentiellen Speicherplatzbedarf in der Anzahl der Variablen gekennzeichnet sind, erhält man dadurch eine kompakte Darstellung vieler Funktionen. Außerdem sind alle gebräuchlichen Kompositionen ($\wedge, \vee$) effizient berechenbar, sofern man die Zwischenergebnisse der rekursiven Berechnung geschickt zwischenspeichert. Bryant hat in [Bry92] Algorithmen zur Funktionskomposition angegeben, in denen Hash-Tabellen verwendet werden, um mehrfache Berechnungen zu vermeiden. Die Verknüpfung von zwei durch BDDs dargestellte Funktionen f und g erhält man dann für die üblichen Operationen in der Zeit $O(|f| \cdot |g|)$, wobei $|f|$ die Anzahl der Knoten des BDDs von Funktion f ist. Die Größe des resultierenden Graphen ist ebenfalls durch $|f| \cdot |g|$ nach oben beschränkt.

Die Vorteile von BDDs für die Darstellung der Baubarkeits- und Zusteuerungsfunktionen liegen auf der Hand:

- Die BDDs der Baubarkeits- und Zusteuerungsformel müssen nur einmal berechnet werden. Nebenbedingungen lassen sich schnell hinzufügen.
- Die Erfüllbarkeits- und Tautologie-Eigenschaft kann direkt (in konstanter Zeit) aus dem BDD abgelesen werden: Besteht der BDD nur aus dem Terminalknoten 1, so ist die Formel allgemeingültig. Wenn der BDD nicht der Terminalknoten 0 ist, so ist die Formel erfüllbar.
- Beweise bestehen (wie im letzten Punkt schon angedeutet) lediglich aus dem Generieren der BDDs, eine nachfolgende Suche nach erfüllbaren Belegungen oder Widersprüchen ist nicht erforderlich.

Normalerweise ist bei der Erzeugung der BDDs der zeitliche Aufwand unbedeutend, der Speicherplatzbedarf hingegen das restriktive Element. Selbst wenn sich das Ergebnis kompakt darstellen läßt (was beispielsweise bei Tautologien oder unerfüllbaren Formeln der Fall ist), wird man häufig mit der Tatsache konfrontiert, daß die Berechnung der Teilformeln deutlich größere BDDs benötigt. Im Falle der Baubarkeits- und Zusteuerungsformeln kann a priori nicht einmal davon ausgegangen werden, daß deren BDDs sich kompakt darstellen lassen.

Ein weiteres vielbeachtetes Phänomen, das auch Impulse zur Entwicklung vieler neuer Verfahren gab, ist die starke Abhängigkeit der BDD-Größen von der gewählten Variablenordnung. Abbildung 4.1 soll dies verdeutlichen. Links wurde die Variablenordnung $a < b < c < d < e < f$, rechts die Ordnung $a < c < e < b < d < f$ zur Darstellung der Funktion $(a \wedge b) \vee (c \wedge d) \vee (e \wedge f)$ verwendet. Eine Verallgemeinerung dieses Beispiels zeigt, daß eine schlechte Variablenordnung

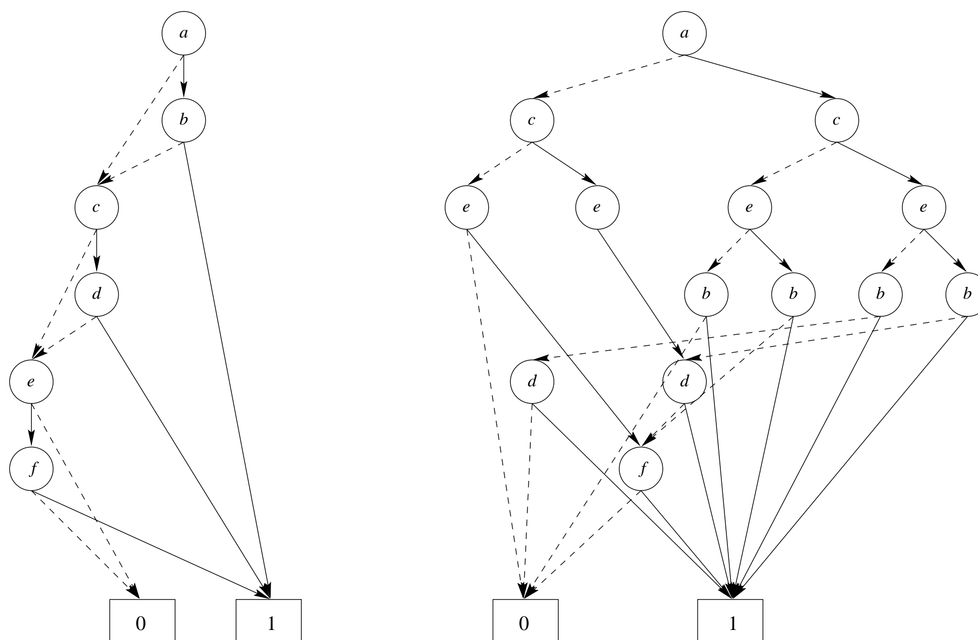


Abbildung 4.1: Auswirkung verschiedener Variablenordnungen auf die BDD-Größe

exponentiell größere BDDs als im optimalen Fall erzeugen kann: Nimmt man die allgemeine Funktion $f(n) = (a_1 \wedge b_1) \vee \cdots \vee (a_n \wedge b_n)$, die $2n$ Variablen enthält, so erhält man im optimalen Fall ($a_1 < b_1 < \cdots < a_n < b_n$) einen BDD mit $2(n+1)$ Knoten. Im ungünstigsten Fall ($a_1 < a_2 < \cdots < a_n < b_1 < \cdots < b_n$) hat man es mit einem BDD mit 2^{n+1} Knoten zu tun. Für manche Funktionen ist die Größe des BDDs – unabhängig von der Variablenordnung – exponentiell in der Anzahl der Variablen (siehe z.B. [Bry86]).

Der zur Zeit beste Algorithmus zum Auffinden einer optimalen Variablenordnung [FS90] ist exponentiell in der Anzahl der Variablen ($O(n^2 3^n)$) und hat einen Speicherplatzbedarf von $O(3^n/\sqrt{n})$. In [THY93] wurde gezeigt, daß das Auffinden optimaler Variablenordnungen für SBDDs (shared BDDs, mehrere Funktionen werden gleichzeitig dargestellt) NP-vollständig ist. Bollig und Wegner zeigten, daß dies auch für BDDs gilt [BW96].

Daher wurde eine Vielzahl von Heuristiken entwickelt, um möglichst gute Ordnungen zu finden. Hauptsächlich aus dem Hardware-Bereich kommen Heuristiken, die eine Ordnung festlegen, bevor BDDs aufgebaut werden. Viele dieser Heuristiken verwenden Eigenschaften der Schaltungs-Topologie und führen eine Tiefensuche ausgehend von den Ausgabe-Pins durch (z.B. [FOH93]). Auch wenn solche statischen Variablenordnungen Vorteile gegenüber einer zufälligen Ordnung liefern, leiden sie doch an gewissen Einschränkungen: Alle Funktionen, also auch die Zwischenergebnisse, müssen mit einer festen Ordnung auskommen; d.h. die Ordnung ist für alle Funktionen und während des gesamten Aufbauvorgangs konstant.

Dynamische Variablenordnungen umgehen diese Einschränkung. Sie versuchen, sobald eine bestimmte BDD-Größe überschritten ist, diesen zu minimieren, indem verschiedene Heuristiken angewandt werden. Inzwischen existieren viele unterschiedliche dynamische Ordnungsheuristiken, die verbreitetsten sind "Sifting" [Rud93] und "Window Permutation" [FYBSV93] sowie Varianten dieser Verfahren.

An den ursprünglichen BDDs wurde eine Vielzahl von Anpassungen vorgenommen, um neue Anwendungsgebiete für sie zu erschließen. So entstanden FDDs (functional decision diagrams, [KSR92]), ADDs (algebraic decision diagrams, [BFG+93]), ZDDs (zero-suppressed decision diagrams, [Min93]), OKFDDs (ordered Kronecker decision diagrams, [DST+94]), BMDs (binary moment diagrams, [BC95]) und viele weitere Variationen des anfänglichen Konzepts.

Implementationen von BDD-Paketen gibt es inzwischen reichlich, beispielsweise von Long ([Lon93], [Lon94]), Drechsler und Becker [DB95], sowie Somenzi [Som97]. Für die vorgenommenen Untersuchungen wurde das CUDD-Paket von Somenzi verwendet, da es zum einen die vollständigste Implementation mit Unterstützung von BDDs, ADDs und ZDDs anbietet, aber auch weil es eine umfangreiche Dokumentation enthält und ständig weiterentwickelt wird. Außerdem wurde bei diesem Paket durch den Einsatz von verschiedenen Hashing- und Garbage-Collection-Techniken eine Verschiebung vom normalerweise limitierenden Speicherplatz- zum eher unkritischen Rechenzeitbedarf erreicht.

Experimentelle Ergebnisse

Wie anfangs des Kapitels erwähnt, wurde die Formel $\mathcal{B}_P$, also die pauschalen Codebedingungen, den nachfolgend beschriebenen Experimenten zugrundegelegt. Es zeigte sich, daß eine Darstellung der kompletten Formel mittels BDDs nicht direkt zu erreichen ist. Daher wurden Teile der Formel einer weitergehenden Analyse unterzogen. Zum einen wurde anstatt der vollständigen Beschreibung der baureihenunabhängigen Baubarkeit, die Regeln für 515 Codes enthält, nur ein Ausschnitt dieser verwendet. In den nachfolgenden Tabellen ist die Formel mit allen 515 Regeln mit PCD-515 bezeichnet, Teile, die nur aus den ersten x Regeln bestehen mit PCD-x. Die anderen Formeln und weitere Untersuchungen werden später erläutert.

In Tabelle 4.1 sind BDD-Größen und Zeiten für verschiedene Teile der pauschalen Codebedingung wiedergegeben. Die BDD-Größe gibt dabei die Anzahl der Knoten des Ergebnisses (und nicht die maximale Knotenanzahl während des Aufbauvorgangs) an, die Zeiten sind in Sekunden wiedergegeben. Alle Messungen wurden auf einem Pentium-PC (100 MHz) unter Linux 2.1 durchgeführt. Die beiden linken Spalten ("ohne dyn. VO") geben die Werte ohne dynamische Variablenordnung wieder, die Zahlen auf der rechten Seite ("mit dyn. VO") wurden mit dynamischer Variablenordnung (Sifting) – inklusive einem Umordnungsschritt am Ende – ermittelt.

Problem	#Vars	#Knoten	Zeit	#Knoten	Zeit
		ohne dyn. VO		mit dyn. VO	
PCD-100	114	231	0.58	188	2.75
PCD-150	220	1542	1.03	1024	8.04
PCD-200	322	62859	11.36	4878	2:43.81
PCD-210	331	147207	51.61	5196	2:47.46
PCD-220	342	236557	5:32.67	6055	3:08.00
PCD-250	367	—	—	86252	36:33.41
PCD-515	580	—	—	—	—
PCD-100-B	183	10735	2.60	2262	17.41
PCD-125-B	213	189370	2:57.31	5362	1:34.41
PCD-150-B	235	—	—	13002	3:41.95
PCD-200-B	343	—	—	163825	1:10:09.99
PCD-373-B	429	—	—	—	—

Tabelle 4.1: BDD-Größen verschiedener Teilformeln von $\mathcal{B}_P$

Die Abhängigkeit von der Variablenordnung ist auch hier offensichtlich. Daher wurden weitere Experimente mit anderen Umordnungs-Strategien vorgenommen, die Ergebnisse sind in Tabelle 4.2 zusammengefaßt. Eine detaillierte Beschreibung der verschiedenen Strategien kann in [Som97] gefunden werden. Es zeigt sich, daß die Sifting-Technik den anderen Verfahren sowohl hinsichtlich Laufzeit als auch bezüglich BDD-Größen überlegen ist. Trotzdem ist, wie man den obigen Ergebnissen entnehmen kann, Sifting nicht ausreichend.

Bessere Variablenordnungen zu finden, blieb bei den Experimenten also weiterhin ein wichtiges Thema. Hauptproblem dabei war, daß die meisten Variablen fast gleichmäßig und größtenteils

	PCD-200		PCD-125-B	
Dyn. VO	#Knoten	Zeit	#Knoten	Zeit
keine	62859	11.65	189370	2:57.31
Sifting	4878	2:42.77	5362	1:34.41
Group Sifting	19326	10:06.55	5528	1:49.22
Window 2	60013	18.49	183626	3:07.33
Window 3	52201	23.82	155203	2:57.91
Window 4	47191	49.35	84967	2:57.53
Random Swap	22909	3:33.45	45267	5:15.42
Random Pivot	13251	3:35.32	25373	17:03.17

Tabelle 4.2: Vergleich verschiedener dynamischer Variablen-Umordnungs-Strategien

"zufällig" über alle Regeln verteilt sind. Die üblichen statischen Heuristiken konnten nicht überzeugen.

So wurde ein weiterer Versuch unternommen, Abhängigkeiten zwischen den Variablen (bzw. den Regeln) zu finden. Letztendlich sollte damit eine Zerlegung in unabhängige Teile ermöglicht werden. Variablen der einzelnen unabhängigen Teile werden in der Ordnung dann so gruppiert, daß Variablen derselben Teilformel benachbart auftreten, Variablen unterschiedlicher Teilformeln aber beliebig weit voneinander entfernt sein dürfen.

Die Zerteilung wurde auf Regel-Ebene vorgenommen. Dabei wurden zwei Regeln als voneinander unabhängig betrachtet, wenn sie keine gemeinsamen Variablen enthielten. Die Variablen einer Regel wurden dabei zu Mengen zusammengefaßt, und die innere Struktur der Regeln nicht weiter berücksichtigt. Diese Analyse lieferte 39 unabhängige Variablengruppen: die größte enthielt 429 Variablen in 373 Regeln, danach folgten weitere Gruppen, die aus 37, 23, 22, 9, 6 und 5 Regeln bestanden, zwei Gruppen mit 3 und acht Gruppen mit 2 Regeln; die restlichen 18 Gruppen beinhalteten jeweils nur die Variablen einer Regel.

In Tabelle 4.1 ist der größte Block mit 429 Variablen und 373 Regeln als PCD-373-B bezeichnet; Teile davon, die nur aus den ersten x Regeln des Blocks bestehen sind unter PCD-x-B verzeichnet.

Zusammenfassend kann man sagen, daß die Möglichkeiten von BDDs zur Darstellung der Formel $\mathcal{B}_P$ nicht ausreichend sind. Es ist nicht zu erwarten, daß die (sowohl hinsichtlich Variablenanzahl als auch Komplexität) deutlich größeren Formeln $\mathcal{B}_K$ und $\mathcal{Z}_V$ ein günstigeres Verhalten aufweisen.

4.2 Termersetzungssysteme

Termersetzungssysteme werden in erster Linie für gleichungsbasierte Theorien verwendet und erlauben dort durch Anwendung einer einzigen Ableitungsregel Beweise zu führen. Diese Ableitungsregel besteht aus der Ersetzung eines Terms durch einen anderen äquivalenten.

Voraussetzung zur Anwendbarkeit dieses Verfahrens ist allerdings eine entsprechende gleichungsbasierte Symbolisierung der gewünschten Theorie, in unserem Fall also der booleschen Logik. Um alle in der Theorie gültigen Sätze (bzw. Gleichungen) ableiten zu können, muß das Termersetzungssystem darüberhinaus terminierend sein und eindeutige Ergebnisse liefern (man nennt es dann kanonisch).

Für boolesche Logik gibt es ein solches kanonisches Termersetzungssystem (siehe z.B. [Hsi82]), es ist in Abbildung 4.2 wiedergegeben. Dieses verwendet zur Darstellung von Formeln die Operatoren $\oplus$ und $\wedge$, wobei $\oplus$ das Exklusiv-Oder bezeichnet.[1] Die Darstellung in dieser Form wird auch Reed-Muller-Form genannt.

[1] Man kann $F \oplus G$ durch $\neg(F \Leftrightarrow G)$ definieren.

$$
\begin{array}{rcl}
F \vee G & \longrightarrow & (F \wedge G) \oplus F \oplus G \\
\neg F & \longrightarrow & F \oplus \top \\
F \oplus F & \longrightarrow & \bot \\
F \oplus \bot & \longrightarrow & F \\
F \oplus (G \oplus G) & \longrightarrow & F \\
F \wedge \bot & \longrightarrow & \bot \\
F \wedge \top & \longrightarrow & F \\
F \wedge F & \longrightarrow & F \\
F \wedge (G \wedge G) & \longrightarrow & F \wedge G \\
F \wedge (G \oplus H) & \longrightarrow & (F \wedge G) \oplus (F \wedge H)
\end{array}
$$

Abbildung 4.2: Ein vollständiges (kanonisches) Termersetzungssystem für boolesche Logik

Um die Allgemeingültigkeit einer Formel F zu beweisen, wendet man die Regeln aus Tabelle 4.2 so lange an, indem man linke Regelseiten durch rechte ersetzt, bis keine weitere Regel mehr anwendbar ist.[2] Erhält man damit die Formel $\top$, so ist die Allgemeingültigkeit von F bewiesen.

Beispiel: Die Allgemeingültigkeit von $F = (x \wedge y) \vee \neg(x \wedge y)$ soll bewiesen werden. Man erhält die folgende Ableitungskette, wobei die von einem Schritt zum nächsten ersetzten Terme unterstrichen sind:

$$
\begin{array}{rcl}
F & = & \underline{(x \wedge y) \vee \neg(x \wedge y)} \\
 & \longrightarrow & (x \wedge y \wedge \underline{\neg(x \wedge y)}) \oplus (x \wedge y) \oplus \neg(x \wedge y) \\
 & \longrightarrow & (x \wedge y \wedge ((x \wedge y) \oplus \top)) \oplus (x \wedge y) \oplus \underline{\neg(x \wedge y)} \\
 & \longrightarrow & (x \wedge y \wedge ((x \wedge y) \oplus \top)) \oplus \underline{(x \wedge y) \oplus (x \wedge y)} \oplus \top \\
 & \longrightarrow & (x \wedge y \wedge ((x \wedge y) \oplus \top)) \oplus \underline{\bot \oplus \top} \\
 & \longrightarrow & \underline{(x \wedge y \wedge ((x \wedge y) \oplus \top))} \oplus \top \\
 & \longrightarrow & (\underline{x \wedge y \wedge x \wedge y}) \oplus (x \wedge y \wedge \top) \oplus \top \\
 & \longrightarrow & (x \wedge y) \oplus (\underline{x \wedge y \wedge \top}) \oplus \top \\
 & \longrightarrow & \underline{(x \wedge y) \oplus (x \wedge y)} \oplus \top \\
 & \longrightarrow & \underline{\bot \oplus \top} \\
 & \longrightarrow & \top
\end{array}
$$

An diesem Beispiel ist schon zu sehen, daß die Zwischenergebnisse bei einem Beweis sehr groß werden können, bevor sie durch Regeln der Art $F \oplus F \longrightarrow \bot$ wieder kollabieren. Indem man gleiche Formeln eindeutig darstellt, kann man die Expansion teilweise vermeiden.

Besonders auffällig ist das explosionsartige Wachstum der Zwischenergebnisse bei großen Disjunktionen. Werden Subterme nicht eindeutig dargestellt, so wächst die Darstellung einer Disjunktion exponentiell mit der Anzahl der Subterme der Disjunktion. Durch mehrfache Verwendung gleicher Teilformeln oder den Einsatz anderer Datenstrukturen (z.B. BDDs oder FDDs) lassen sich diese Probleme – zumindest teilweise – vermeiden. Da Termersetzungssysteme ein vergleichsweise allgemeines Konzept verwenden, ist von einer Spezialisierung der Datenstrukturen auf aussagenlogische Formeln eine deutliche Verbesserung zu erwarten.

Experimente mit termersetzungsbasierten Beweismethoden wurden anhand der beiden Pakete ReDuX [Bün93] und RAP [Hus85] vorgenommen. Bei ReDuX handelt es sich um ein Experimen-

[2] Die Operatoren $\wedge$ und $\otimes$ seien assoziativ und kommutativ.

tiersystem für Termersetzung aufbauend auf der SAC-2 Computer-Algebra-Bibliothek. Es enthält Programme, die mit Assoziativität und Kommutativität umgehen können und unterstützt die Verwendung von externen Evaluationsbereichen wie FDDs oder booleschen Polynomen. Auch in der Hardware-Verifikation wurde es schon erfolgreich eingesetzt [BKL96]. RAP wurde ursprünglich als System zur symbolischen Ausführung von hierarchisch gegliederten Spezifikationen entwickelt, später aber durch den Einbau von FDDs (JRAP) auch im Verifikationsbereich eingesetzt [GK97]. JRAP bietet darüberhinaus die Möglichkeit, gleiche Subterme eindeutig darzustellen. Dadurch ist dieses System für die Darstellung von booleschen Polynomen prädestiniert.

Ergebnisse

Die Experimente mit ReDuX wurden aus verschiedenen Gründen wieder eingestellt: Ohne externe Evaluationsbereiche war die Darstellung der meisten einzelnen Regeln nicht möglich. Durch die Verwendung von booleschen Polynomen oder FDDs wurden zwar Verbesserungen erreicht, da ReDuX aber keine eindeutige Darstellung gleicher Subterme anbietet (bis auf Variablen und Konstanten) waren die Ergebnisse selbst bei Verwendung nur weniger Regeln nicht mehr überschaubar. Für einige relativ kleine Teilformeln von $\mathcal{B}_P$ sind die Ergebnisse der Zeitmessung in Tabelle 4.3 aufgeführt.

		ReDuX		JRAP		
Problem	#Vars	$t_{Red.}$	t_{FDD}	$t_{Red.}$	t_{FDD}	#Knoten
PCD-9	12	0.05	1.27	0.14	0.20	41
PCD-10	14	0.07	6.07	0.15	0.20	43
PCD-11	17	0.97	13:33.58	0.16	0.21	49
PCD-100	114	—	—	33.77	3.30	527
PCD-125	185	—	—	41.30	6.15	29647
PCD-150	220	—	—	53.59	—	—
PCD-515	580	—	—	9:55.13	—	—

Tabelle 4.3: Ergebnisse mit ReDuX und JRAP am Beispiel der Formel $\mathcal{B}_P$

JRAP hatte demgegenüber durch das eingebaute Structure-Sharing Vorteile. Die mit JRAP erzielten Ergebnisse sind ebenfalls in Tabelle 4.3 zusammengestellt. Die Laufzeiten beziehen sich auf Parsing und Rewriting (Spalte “$t_{Red.}$”), sowie den Aufbau des FDDs der Formel (Spalte“t_{FDD}”). Alle Zeiten sind in Sekunden angegeben. Die Größe der erhaltenen FDDs ist ebenfalls mit verzeichnet.

Die Vorteile von Termersetzungs-basierten Ansätzen im Vergleich zu binären Entscheidungsdiagrammen sind kaum auszumachen, da ohne deren Einsatz bei JRAP und ReDuX keine eindeutige Darstellung der Formeln, wie sie zum automatischen Beweisen erforderlich ist, möglich war. Im Vergleich zu BDDs wäre denkbar, daß durch die Vorverarbeitung mittels Ersetzungsregeln Vereinfachungen an der Formel vorgenommen werden können, die das Generieren des Entscheidungsdiagramms erleichtern.

Darüberhinaus gibt es Formeln, die als BDDs eine exponentiell größere Darstellung haben als mit FDDs, und dies ist selbst bei optimalen Variablenordnungen möglich. Allerdings kann auch der umgekehrte Fall eintreten, daß die FDD-Darstellung exponentiell größer ist. In der Praxis treten zwischen den beiden Darstellungen allerdings meist keine gravierenden Unterschiede auf.

4.3 Resolution

Resolution [Rob65] ist ein Beweisverfahren für Aussagen- und Prädikatenlogik, das die Allgemeingültigkeit einer Formel F beweist, indem die negierte Aussage $\neg F$ zu einem Widerspruch

geführt wird. Einzige Ableitungsregel ist die Resolutionsregel

$$\frac{(A \vee B) \wedge (\neg B \vee C)}{A \vee C}$$

wobei A und C beliebige Formeln und B ein beliebiges Literal ist.

Die zu beweisende Formel muß dabei in konjunktiver Normalform vorliegen. Üblicherweise werden solche Eingabeformeln als Klauselmengen dargestellt. Eine Klausel ist dabei eine Menge $K = \{l_1, \ldots, l_n\}$ von Literalen, die als $l_1 \vee \cdots \vee l_n$ interpretiert wird. Die Klauseln einer Klauselmenge werden implizit als konjunktiv verknüpft gedacht, so daß eine Klauselmenge einer konjunktiven Normalform entspricht. Resolutionsbeweiser versuchen also die leere Klausel $K = \emptyset$ aus einer Klauselmenge herzuleiten. In dieser Notation sieht die Resolutionsregel dann wie folgt aus:

$$\frac{\{K_1, \ldots, K_{n-2}, K_{n-1} \cup \{x\}, K_n \cup \{\neg x\}\}}{\{K_1, \ldots, K_{n-2}, K_{n-1} \cup K_n\}}$$

Die neue Klausel $K_{n-1} \cup K_n$ wird Resolvente der beiden Klauseln $K_{n-1} \cup \{x\}$ und $K_n \cup \{\neg x\}$ genannt. In der Prädikatenlogik ist die Resolution das wohl prominenteste Verfahren im automatischen Beweisen. Verschiedene Strategien wie Unit-, Hyper-, N- oder P-Resolution wurden entwickelt, in erster Linie für das Haupteinsatzgebiet Prädikatenlogik.

Die bekannteste Implementation eines Resolutionsbeweisers ist das von McCune am Argonne National Laboratory entwickelte OTTER [McC94]. Für boolesche Logik verwendet OTTER geordnete Hyperresolution und eine Set-of-Support-Strategie. Ein Hyperresolutionsschritt eliminiert alle negativen Literale einer Klausel durch mehrere Resolutionsschritte mit Klauseln, die nur positive Literale enthalten.

			Zeit			
Problem	#Vars	#Klauseln	Einlesen	Kl.-Gen.	Suche	Total
PCD-100	114	726(+688)	1.08	0.25	2.12	3.45
PCD-200	322	1288(+809)	2.18	0.36	3.84	6.38
PCD-250	367	1573(+897)	3.06	0.41	5.36	8.83
PCD-515	580	3431(+2091)	10.31	1.02	27.58	39.91
PCD-515-K	580	3404(+2097)	15.84	0	26.37	42.21

Tabelle 4.4: Ergebnisse mit OTTER am Beispiel der Formel $\mathcal{B}_P$

Die Ergebnisse verschiedener Messungen mit OTTER sind in Tabelle 4.4 zusammengefaßt. Die Probleme sind bis auf PCD-515-K dieselben wie in den vorangegangenen Tests. OTTER kann auch Formeln bearbeiten, die nicht in konjunktiver Normalform gegeben sind. In einem Vorverarbeitungsschritt wird eine Konvertierung in Klauseln vorgenommen. Die entsprechende Spalte gibt in der Form "$x(+y)$" die Anzahl x der generierten Klauseln des Problems sowie die Anzahl y der während der Eingabeverarbeitung subsumierten Klauseln an. Das Problem PCD-515-K ist äquivalent zu PCD-515, bis auf die Tatsache, daß PCD-515-K schon in Klauseldarstellung gebracht wurde. Die Spalte "Einlesen" bezieht sich auf die zum Lesen der Formel und zum Löschen subsumierter Klauseln benötigte Zeit, wobei die Zeit zur Generierung der Klauseln nicht mitgerechnet wurde. Diese Zeit ist in der nächsten Spalte ("Kl.-Gen.") explizit registriert. Wie lange der durch Hyperresolution vorgenommene Beweisversuch dauert, ist in der mit "Suche" überschriebenen Spalte angegeben. Auch hier sind alle Laufzeiten in Sekunden gemessen.

Bei den vorliegenden Beispielen, die Teile der Formel $\mathcal{B}_P$ darstellen, ist natürlich davon auszugehen, daß diese Formeln erfüllbar sind. Daher kann durch Resolution kein Widerspruch hergeleitet werden. Ein Widerspruch hätte angezeigt, daß die Formel unerfüllbar ist.

Auch bei den letztendlich erforderlichen Beweisen zur Baubarkeit und Zusteuerung treten vergleichbare Formeln und Fragestellungen auf. Typischerweise ist man daran interessiert, ob es einen

baubaren Auftrag mit einer bestimmten Eigenschaft gibt. Dies entspricht der Erfüllbarkeit der zugeordneten Formel F und damit der Nicht-Existenz eines Resolutionsbeweises für F.[3]

Soll die Allgemeingültigkeit einer Formel F gezeigt werden, so muß $\neg F$ in konjunktive Normalform gebracht werden und mit dem Resolutionsverfahren ein Widerspruch hergeleitet werden.

Problematisch kann das Erzeugen der konjunktiven Normalformen sein. Insbesondere muß für den Nachweis einer Tautologie das Negat $\neg F$ der entsprechenden Formel in konjunktiver Normalform vorliegen, F selbst also in disjunktiver Normalform.

So muß eventuell eine Formel sowohl in CNF als auch in DNF konvertiert werden. Typischerweise ist aber nur eine der beiden Darstellungen kompakt. Es kann passieren, daß eine Darstellung exponentiell größer als ihre duale Darstellung ist. Alle hier relevanten Formeln sind große Konjunktionen, also leicht in konjunktive Normalform zu bringen. Erfüllbarkeitsaussagen sind also leicht zu überprüfen, Tautologiebeweise oft deutlich schwerer.

Sowohl auf die Konvertierung in CNF (bzw. Klauselform) als auch auf das soeben aufgeworfene Problem wird später noch genauer eingegangen. Man kann aber festhalten, daß die Voraussetzung einer konjunktiven Normalform Beschränkungen der handhabbaren Formeln mit sich bringt.

4.4 Das Davis-Putnam-Verfahren

Das Davis-Putnam-Verfahren ([DP60], [DLL62]) ist ein Entscheidungsverfahren für die Erfüllbarkeit aussagenlogischer Formeln. Es ist leicht dahingehend erweiterbar, daß bei erfüllbaren Formeln auch Modelle generiert werden. Diese Eigenschaft macht es für Probleme, bei denen man nicht nur einen Beweis, sondern auch erfüllende Belegungen erwartet, interessant.

Wie bei der Resolution, muß auch bei dem Davis-Putnam-Verfahren die zu überprüfende Formel in konjunktiver Normalform vorliegen. Die dadurch induzierten Probleme, wie sie weiter oben schon bei der Resolution aufgeführt wurden, sind also auch hier zu beobachten.

Grundsteine des Verfahrens sind Unit-Propagation und Fallunterscheidungen. Unit-Propagation bezeichnet die Zusammennahme von Unit-Resolution und Unit-Subsumption. Die Unit-Resolution schränkt die Resolutionsregel insofern ein, als eine der beiden zu resolvierenden Klauseln nur aus einem Literal bestehen darf (Unit-Klausel). Unter Unit-Subsumption versteht man das Löschen solcher Klauseln, die von einer Unit-Klausel (d.h. einem Literal) subsumiert werden. In der Mengendarstellung entspricht dies dem Wegfallen aller Klauseln, die das betreffende Literal enthalten.

Logisch rechtfertigen läßt sich die Unit-Propagation durch die beiden aussagenlogischen Äquivalenzen $F \wedge (\neg F \vee G) \equiv F \wedge G$ (Resolution) und $F \wedge (F \vee G) \equiv F$ (Subsumption). Die Fallunterscheidung basiert auf der Tatsache, daß eine Formelmenge M genau dann ein Modell hat, wenn $M \cup \{F\}$ oder $M \cup \{\neg F\}$ ein Modell besitzt. Eine Belegung b ist dabei Modell einer Formelmenge M, wenn b ein Modell von allen Formeln $F \in M$ ist.

In Abbildung 4.3 ist der Pseudo-Code einer Prozedur `satisfiable` angegeben, die den Davis-Putnam-Algorithmus skizziert. Die Effizienz dieses Algorithmus hängt neben der gewählten Datenstruktur zur Darstellung der Klauseln und der Implementation der Unit-Propagation in erster Linie von der Auswahl des Literals L in der Fallunterscheidung ab. In [HV95] wurde ein umfangreicher Vergleich verschiedener Heuristiken vorgenommen. Eine einfache Regel hat sich in vielen Fällen bewährt: Wähle ein beliebiges Literal aus einer kürzesten positiven Klausel. Dabei ist eine positive Klausel eine Klausel, die nur positive Literale enthält.

Von Zhang und Stickel wurde in [ZS94] und [ZS96] ein Algorithmus zur Unit-Propagation vorgestellt, dessen Komplexität sublinear in der Anzahl der Variablenvorkommen der Eingabe ist. Unit-Subsumption kann damit in konstanter, Unit-Resolution in linearer Zeit bewältigt werden. Zusammen mit einer guten Heuristik zur Literalauswahl bei der Fallunterscheidung ist das Davis-Putnam-Verfahren damit eines der effizientesten zur Lösung des Erfüllbarkeitsproblems. Eine Reihe

[3] Unter Resolutionsbeweis wird hierbei die Widerlegung von F verstanden.

```
function satisfiable (clause-set S) returns boolean
   repeat   // unit-propagation
      for each unit clause L in S do
         delete (L ∨ Q) from S           // unit-subsumption
         delete ¬L from (¬L ∨ Q) ∈ S     // unit-resolution
      end
      if S is empty then return TRUE
      else if the empty clause is in S then return FALSE
      endif
   until no further changes result
   choose a literal L occurring in S   // case-splitting
   if satisfiable(S ∪ {L}) then return TRUE
   else if satisfiable(S ∪ {¬L}) then return TRUE
   else return FALSE
   endif
end
```

Abbildung 4.3: Der Davis-Putnam Algorithmus

von offenen Fragen über Quasigruppen konnten mit diesem Verfahren gelöst werden [ZBH96].

Unter den vielen Implementationen des Davis-Putnam-Verfahrens befinden sich das von H. Zhang entwickelte SATO ([Zha93], [Zha97]) (und dessen parallele Variante, PSATO [ZBH96]) oder POSIT [Fre95]. Zu den nachfolgend beschriebenen Experimenten wurde SATO herangezogen, da es sich als etwas schneller als POSIT erwiesen hat.

Da SATO Eingaben in Klausel-Form erwartet (genauer: DIMACS-Format), mußten die zur Messung verwendeten Formeln in Klauselform gebracht werden, wie dies auch schon bei der Resolution (Problem PCD-515-K) der Fall war. Das dazu verwendete Verfahren wird später genauer beschrieben. Die Ergebnisse der Messungen sind in Tabelle 4.5 dargestellt.

				Zeit		
Problem	#Vars	#Klauseln	Speicher	Einlesen	Suche	Total
PCD-515-K	580	3404(+2097)	319	0.15	0.03	0.18
BBK-K	1151	16417(+2557)	1983	0.59	0.21	0.81
BBR-K	1151	16567(+2557)	1989	0.66	0.23	0.89
BBK-Z-K	1151	19608(+2567)	2386	0.54	0.28	0.82
BBR-Z-K	1151	19758(+2567)	2423	0.71	0.27	0.98

Tabelle 4.5: Messungen mit SATO anhand der Formeln $\mathcal{B}_P$, $\mathcal{B}_K$ und $\mathcal{Z}_V$

Zu den Messungen wurden neben der baureihenunabhängigen Baubarkeitsformel $\mathcal{B}_P$ (PCD-515-K) auch andere, größere Formeln herangezogen, da Teilformeln von $\mathcal{B}_P$ keine vernünftigen Messungen mehr ermöglicht hätten. Mit BBK-K ist die komplette Baubarkeitsformel $\mathcal{B}_K$ bezeichnet, BBR-K enthält darüberhinaus noch die Baumuster-Informationen, ist also $\mathcal{B}_K \wedge \mathcal{B}_R$. Bei den Problemen BBK-Z-K und BBR-Z-K wurde zusätzlich noch die Zusteuerungsformel $\mathcal{Z}_V$ dazugenommen, also entspricht das Problem BBR-Z-K der Formel $\mathcal{B}_K \wedge \mathcal{B}_R \wedge \mathcal{Z}_V$. Die Erfüllbarkeit der letzten Formel bedeutet demnach, daß es einen baubaren, voll zugesteuerten Auftrag gibt, der aus der Umwandlung eines Baumusters entstanden ist.

Die Spalte "#Klauseln" gibt wieder die Anzahl der Klauseln des Problems und die Anzahl der während der Aufarbeitung der Eingabe subsumierten Klauseln an. Der (Haupt-)Speicherplatzbedarf ("Speicher") ist in Kilo-Byte angegeben, die beiden zuerst genannten Laufzeitangaben (in Sekunden) beziehen sich auf das Einlesen der Klauseln und den Aufbau der Datenstruktur (trie, discri-

mination tree), sowie die für das eigentliche Davis-Putnam-Verfahren benötigte Zeit.

Im Vergleich zu den anderen Verfahren (läßt man einmal die Konvertierung in Klausel-Form außer acht) erweist sich der Davis-Putnam-Algorithmus als deutlich überlegen. Keines der bisher vorgestellten Verfahren konnte die komplette Baubarkeitsformel darstellen. Die Suche nach baubaren Aufträgen bezüglich baureihenunabhängiger Baubarkeit ist sogar fast nicht zu bemerken. Den Davis-Putnam-Beweiser interaktiv einzusetzen wird damit ebenfalls ermöglicht. Es bleibt noch anzumerken, daß die Zeiten für unerfüllbare Formeln ein ähnlich gutes Laufzeitverhalten aufweisen.

4.5 Stålmarcks Methode

Ståmarcks Methode [Stå92] ist ein patentiertes Beweisverfahren für aussagenlogische Formeln. Im Gegensatz zur Davis-Putnam-Methode wird nicht die Erfüllbarkeit, sondern die Allgemeingültigkeit von Formeln überprüft. Vorteilhaft ist, daß die Eingabeformel nicht in konjunktiver Normalform (oder Klausel-Darstellung) vorliegen muß.

Die grundlegenden Ideen erinnern teilweise an die Davis-Putnam-Methode, teilweise sind sie Kalkülen des natürlichen Schließens entnommen. Die folgende knappe Darstellung lehnt sich an die Beschreibung in [Har96] an.

Die zu überprüfenden Formeln werden in einem Vorverarbeitungsschritt so umgeformt, daß sie außer Literalen und Negationen nur noch die Konnektive $\wedge$ und $\Leftrightarrow$ enthalten. Diese Transformation kann man in linearer Zeit in der Größe der Eingabeformel durchführen, die resultierende Formel enthält außerdem – abgesehen von neuen Negationssymbolen – nicht mehr Konnektive als die Ursprungsformel. Dies ist ein wesentlicher Unterschied zu den konjunktiven Normalformen, die zur Resolution oder für das Davis-Putnam-Verfahren benötigt werden, und die im schlimmsten Fall exponentiell größer sein können als die Ausgangsformel.

Die Umformung kann beispielsweise durch die folgenden Reduktionen erreicht werden:

$$\begin{aligned} F \Rightarrow G &\longrightarrow \neg(F \wedge \neg G) & \qquad F \vee G &\longrightarrow \neg(\neg F \wedge \neg G) \\ \neg\top &\longrightarrow \bot & \neg\bot &\longrightarrow \top \\ F \wedge \top &\longrightarrow F & F \wedge \bot &\longrightarrow \bot \\ \neg\neg F &\longrightarrow F & & \end{aligned}$$

Ist die Formel damit zu $\top$ oder $\bot$ reduziert worden, ist nichts weiter zu tun. Andernfalls wird die Formel in “Triplets” zerlegt, die man durch Einführen neuer Variablen erhält. Diese Triplets müssen eine der beiden Formen

$$x \Leftrightarrow y \wedge z \qquad \text{oder} \qquad x \Leftrightarrow y \Leftrightarrow z$$

annehmen, wobei x, y und z (positive oder negative) Literale sind. Man kann sich diese Zerlegung (und die Definition neuer Variablen) als Benennung von Subtermen der Formel mit den entsprechenden Variablennamen vorstellen. In einer Implementation muß die Zerlegung also nicht wirklich vorgenommen werden. Eine konjunktive Verknüpfung der Triplets ist logisch äquivalent zu der ursprünglichen Formel.

Die zentrale Idee des Algorithmus ist, aus der Annahme, daß die zu beweisende Formel falsch ist, möglichst viele Variablenbelegungen abzuleiten, die aus dieser Annahme folgen. Außer den üblichen Variablenbelegungen mit 0 und 1 sucht man auch nach Gruppen von Variablen, die gleich (oder entgegengesetzt) belegt werden müssen. Solche Variablenbelegungen werden in Form von Gleichungen dargestellt. Die Belegung einer Variablen x mit 0 entspricht der Gleichung $x = 0$; müssen zwei Variablen x und y gleich belegt werden, so erhält man die Gleichung $x = y$. Analog drückt die Gleichung $x = \neg y$ aus, daß die Variablen x und y unterschiedlich belegt werden müssen. Man hat einen Beweis gefunden, wenn eine widersprüchliche Gleichung abgeleitet werden konnte, beispielsweise $x = \neg x$.

Die Ableitung neuer Gleichungen aus schon bestehenden wird anhand der Triplets vorgenommen und läßt sich durch einfache Regeln beschreiben. Eine vollständige Liste der erforderlichen Regeln ist in [Har96] zu finden.

Steht für Literale x, y und z das Triplet $x \Leftrightarrow y \wedge z$ zur Verfügung, so kann man an der Gleichungsmenge $G \cup \{x = \neg y\}$ den folgenden Ableitungsschritt vornehmen:

$$\frac{G \cup \{x = \neg y\}}{G \cup \{x = \neg y, y = 1, z = 0\}}[x \Leftrightarrow y \wedge z]$$

Man erhält also zwei zusätzliche Gleichungen, die neue Aussagen über die erforderliche Variablenbelegung machen.

Beispiel: Die Allgemeingültigkeit der Formel

$$F = \neg((a \Leftrightarrow b \wedge c) \wedge (b \Leftrightarrow \neg c) \wedge a)$$

soll bewiesen werden. Diese Formel enthält nur die Konnektive $\wedge$ und $\Leftrightarrow$, also kann der Vorverarbeitungsschritt entfallen.

Die Triplets dieser Formel sind dann (unter Einführung der neuen Variablen $x_1, \ldots, x_5$):

$$\begin{array}{ll} x_1 \Leftrightarrow b \wedge c & x_2 \Leftrightarrow a \Leftrightarrow x_1 \\ x_3 \Leftrightarrow b \Leftrightarrow \neg c & x_4 \Leftrightarrow x_3 \wedge a \\ x_5 \Leftrightarrow x_2 \wedge x_4 & \end{array}$$

Die gesamte Formel F ist äquivalent zu $\neg x_5$. Wir nehmen also an, daß die Formel falsch ist ($x_5 = 1$) und nehmen ausgehend von dieser Gleichung Ableitungsschritt anhand der Triplets vor (Ableitungsschritte sind hier als Pfeile dargestellt) :

$$\begin{array}{l} \{x_5 = 1\} \xrightarrow{[x_5 \Leftrightarrow x_2 \wedge x_4]} \{x_5 = 1, x_2 = 1, x_4 = 1\} \xrightarrow{[x_4 \Leftrightarrow x_3 \wedge a]} \{x_5 = 1, x_2 = 1, x_4 = 1, x_3 = 1, a = 1\} \\ \xrightarrow{[x_3 \Leftrightarrow b \Leftrightarrow \neg c]} \{x_5 = 1, x_2 = 1, x_4 = 1, x_3 = 1, a = 1, b = \neg c\} \\ \xrightarrow{[x_1 \Leftrightarrow b \wedge c]} \{x_5 = 1, x_2 = 1, x_4 = 1, x_3 = 1, a = 1, b = \neg c, x_1 = 0\} \\ \xrightarrow{[x_2 \Leftrightarrow a \Leftrightarrow x_1]} \{x_5 = 1, x_2 = 1, x_4 = 1, x_3 = 1, a = 1, b = \neg c, x_1 = 0, x_1 = 1\} \end{array}$$

Damit hat man einen Widerspruch ($x_1 = 0 = 1$) hergeleitet, womit der Beweis fertig ist.

Der gerade beschriebene Ableitungsprozeß wird auch "0-Sättigung" oder "0-saturation" genannt. 0-Sättigung alleine liefert aber kein vollständiges Beweisverfahren für die Aussagenlogik. Daher wird, wenn die 0-Sättigung keinen Widerspruch generieren konnte, eine Reihe von Fallunterscheidungen vorgenommen.

Dazu bildet man zu der durch 0-Sättigung erhaltenen Gleichungsmenge G zwei neue Gleichungsmengen der Form $G_1 = G \cup \{x = 0\}$ und $G_2 = G \cup \{x = 1\}$. An diesen beiden Gleichungsmengen wird wiederum eine 0-Sättigung vorgenommen, man erhält zwei neue Gleichungsmengen G'_1 und G'_2. Der Schnitt dieser Mengen wird als Ausgangspunkt für weitere Fallunterscheidungen mit anderen Variablen verwendet. Führt man diese Fallunterscheidung für sämtliche Variablen durch, wird dies 1-Sättigung genannt. Allgemein versteht man unter n-Sättigung eine Fallunterscheidung über alle möglichen Variablenbelegungen von n Variablen. Für eine 2-Sättigung müßte man also für alle Variablenpaare x, y die vier möglichen Fallunterscheidungen vornehmen.

Insgesamt erhält man dadurch ein vollständiges Verfahren. In den meisten in der Praxis auftretenden Fällen ist eine 1-Sättigung ausreichend.

Da sich Stålmarcks Methode in vielerlei industriellen Projekten als erfolgreich erwiesen hat, ist auch für die Baubarkeitskontrolle eine gute Prognose zu geben. Vergleiche mit dem Davis-Putnam-Verfahren wären interessant, wurden bisher aber noch nicht vorgenommen.

4.6 Zur Berechnung konjunktiver Normalformen

Sowohl für die Resolution als auch für die Davis-Putnam-Methode müssen die Formeln in konjunktive Normalform gebracht werden. Traditionelle Verfahren wenden das Distributivgesetz

$$F \vee (G \wedge H) \;\equiv\; (F \vee G) \wedge (G \vee H)$$

so lange an, bis die gewünschte Form erreicht ist. Dabei treten allerdings viele Klauseln auf, die von anderen subsumiert werden. Um bei größeren Formeln überhaupt ans Ziel zu kommen, müssen schon während der Konvertierung ständig Subsumptionstests vorgenommen werden. Daher ist bei großen Formeln der Zeitbedarf für die Konvertierung beträchtlich, oft übertrifft er sogar die für den eigentlichen Beweis benötigte Zeit.

So ist es nicht verwunderlich, daß in der Literatur verschiedene Verfahren zu finden sind, die sich mit diesem Problem beschäftigen ([Soc91], [RMBH95]). Allerdings konnten die meisten dieser Verfahren keine deutlichen Verbesserungen bringen.

Unerwartete Hilfe kam aus dem Hardware-Bereich. Dort tritt bei der zweistufigen Schaltungssynthese und -minimierung ein ähnliches Problem auf: Es sollen möglichst kleine Schaltnetze in (irredundanter) Sum-Of-Product-Form (ISOP, entspricht einer DNF) generiert werden. Die in diesem Bereich bekannten und erfolgreichen BDDs und ZDDs konnten gewinnbringend eingesetzt werden. Inzwischen sind hinreichend optimierte Algorithmen vorhanden (siehe z.B. [CMF93], [Min93]), die solche Darstellungen berechnen.

Um die verschiedenen Verfahren zu vergleichen, wurde eine parametrisierte Testformel T_n, definiert durch

$$T_n \;=\; \bigvee_{i \in n} \left(\bigwedge_{j \in n} x_{i+j} \wedge \bigwedge_{j \in n} \neg x_{i+j+n} \right) \;\wedge\; \bigwedge_{i \in n} \left(\bigvee_{j \in n} x_{i+j} \vee \bigvee_{j \in n} \neg x_{i+j+n} \right)$$

verwendet. Die untersuchten Konvertierungsmethoden waren

1. der in Otter eingebaute Algorithmus,
2. ein selbst implementiertes Standardverfahren,
3. das im CUDD-Paket implementierte, ZDD-basierte Verfahren und
4. eine Kombination der letzten beiden.

Die zuletzt aufgeführte Methode verwendet dabei für kleine Formeln das Standard-, für größere das ZDD-basierte Verfahren. Die Ergebnisse sind in Tabelle 4.6 aufgeführt. In der ersten Spalte ist die Probleminstanz angegeben, gefolgt von Variablenanzahl und Anzahl der Funktionssymbole der Formel in den nächsten beiden Spalten. In den letzten vier Spalten sind die Laufzeiten der oben angeführten Konvertierungsmethoden in Sekunden verzeichnet.

			Zeit			
Problem	#Vars	#Symbole	Otter	Standard	BDD	Mix
T5	14	249	0.29	2.29	0.09	0.04
T8	23	639	40.73	4:53.01	0.19	0.08
T10	29	999	14:55.37	—	0.27	0.14
T20	59	3999	—	—	1.40	0.89
T50	149	24999	—	—	16.58	16.67
T100	299	99999	—	—	1:44.29	1:45.42

Tabelle 4.6: Konvertierung in konjunktive Normalform

Es ist offensichtlich, daß die BDD-basierten Verfahren der traditionellen Methode weit überlegen sind. Automatischen Beweisern, die auf eine Klausel-Darstellung angewiesen sind, können so neue Einsatzgebiete erschlossen werden, die bisher nicht erreichbar waren.

Die in [BHMR95] beschriebene Idee, durch einen Vorverarbeitungsschritt mittels Anti-Links subsumierte Klauseln zu erkennen und zu entfernen, könnte weitere Verbesserungen bringen. Untersuchungen in diese Richtung wurden aber nicht vorgenommen.

4.7 Zusammenfassung und Vergleich

Zwischen den verschiedenen Beweisverfahren bestehen hinsichtlich Speicherplatzbedarf und Laufzeiten enorme Unterschiede. Angesichts der NP-Vollständigkeit des Erfüllbarkeitsproblems war mit einem solchen Ergebnis zu rechnen. Es zeigte sich auch, daß von den Erfolgen eines Verfahrens auf einem benachbarten Gebiet nicht einfach auf den hier vorliegenden Fall geschlossen werden konnte.

Binäre Entscheidungsdiagramme leiden in den modernen Implementationen immer weniger an den Speicherplatzproblemen früherer Untersuchungen. Durch verschiedene dynamische Umordnungsstrategien konnte eine Verschiebung von der Platz- auf die Zeitkomponente erreicht werden. Trotzdem spielt die Variablenordnung eine dominierende Rolle. Die in [US94] angegebenen Ergebnisse konnten bestätigt werden. Die typische Fragestellung im Bereich der Hardware-Verifikation (Äquivalenz von Formeln) ist anscheinend zu weit von den vorliegenden Erfordernissen entfernt.

Problematisch für binäre Entscheidungsdiagramme ist die annähernd zufällige Verteilung der Variablen in den Baubarkeitsformeln, so daß wahrscheinlich keine guten Variablenordnungen existieren. Man kommt damit der (exponentiell großen) Darstellung über Funktionstabellen bedenklich nahe.

Termersetzungssysteme haben unter dem starken Anwachsen der Formeln durch die Reed-Muller-Transformation zu leiden. Ohne Structure-Sharing können größere Formeln nicht dargestellt werden. Die Erweiterung um FDDs oder andere Evaluationsbereiche führt zu denselben Problemen, wie sie bei binären Entscheidungsdiagrammen alleine auftreten. Spezielle BDD-Implementationen verwenden darüberhinaus oft aktuellere Algorithmen.

Die auf die allgemeinere Prädikatenlogik ausgelegte Resolution konnte recht überzeugende Resultate erzielen. Im Vergleich zu dem Davis-Putnam-Verfahren sind aber doch zu große Unterschiede auszumachen. Beide Beweismethoden teilen die Einschränkung, nur mit Formeln in Klausel-Form umgehen zu können, so daß sie auf effiziente Konvertierungsverfahren angewiesen sind.

Das Davis-Putnam-Verfahren lieferte mit Abstand die besten Ergebnisse. Wie in [US94] beschrieben, ist diese Methode für schwierige Erfüllbarkeitsprobleme mit Einschränkungen (constraint-satisfaction problems) anderen deutlich überlegen. Einzig die erforderliche Klausel-Darstellung der Eingabe erfordert schnelle Verfahren zur Konvertierung. Wie gesehen, läßt sich dies durch Hinzunahme von BDD-basierten Methoden in den Griff bekommen.

Ein Vergleich der Stålmarck-Methode mit dem Davis-Putnam-Verfahren wäre höchst interessant, da beide gute Ergebnisse versprechen. Da der Stålmarck-Algorithmus in der Literatur bisher nur mit BDDs und Resolution verglichen wurde, wäre dies eine interessante Richtung für zukünftige Untersuchungen.

Abschließend sei noch bemerkt, daß die Kombination verschiedener Techniken (BDDs zur Klauselgenerierung, Davis-Putnam-Verfahren als Beweiser) enormes Potential beinhaltet.

Kapitel 5

Implementation

In diesem Kapitel wird zuerst die im Rahmen dieser Arbeit erstellte C++-Bibliothek beschrieben, die boolesche Formelmanipulation, BDD-Funktionalität und einen Davis-Putnam-Beweiser vereint. Daneben enthält sie eine Komponente, die es erlaubt, aus vorverarbeiteten Datenbanktabellen die notwendigen Regeln (Baubarkeit, Zusteuerung, Teilebedarfsermittlung) zu extrahieren und diese zu neuen Formeln zusammenzustellen.

Danach werden verschiedene Programme vorgestellt, mit denen die in Kapitel 2 aufgeworfenen Probleme bearbeitet werden können. Darunter findet man beispielsweise Programme zur Reihenfolge-Analyse der Zusteuerung oder zum Auffinden von unnötigen Codes und Teilen.

Im Unterschied zu Kapitel 3 werden bei der Implementation keine Ausführungsarten unterschieden. Es gibt also nur eine einzige Ausführungsart (FW, Limousinen), die Formeln aus Kapitel 3 vereinfachen sich dementsprechend.

5.1 C++-Bibliothek

Die im folgenden beschriebene C++-Bibliothek besteht grob aus zwei großen Blöcken:

- Einer Schnittstelle zu den Datenbanktabellen mit entsprechenden Funktionen zur Selektion und Zusammenstellung der relevanten Daten und
- einem aussagenlogischen Teil mit Formelmanipulator, Klausel-Generator, BDD/ZDD-Unterstützung und Davis-Putnam-Beweiser.

Der erste Teil ist speziell auf die Mercedes-Benz-Datenbanken der Dialog-Produktdokumentation [Mer95] abgestimmt, wohingegen der zweite Block universell einsetzbar ist.

Entsprechend den Ergebnissen des letzten Kapitels enthält dieser zweite Block die Komponenten und Verfahren, die sich in den Experimenten bewährt haben. So wurde für das BDD-Paket von Somenzi (CUDD, [Som97]) eine C++-Schnittstelle geschaffen. Zhangs Davis-Putnam-Implementation SATO ([Zha93],[Zha97]) wurde ebenfalls mit einer C++-Schnittstelle versehen. Die CUDD-Bibliothek ist normalerweise über ein C-Interface anzusprechen, für das eigenständige Programm SATO ist keine Programmier-Schnittstelle vorgesehen.

Die Implementation verwendet die "Standard Template Library" (STL, [SL94]) und den GNU-C/C++-Compiler. Damit ist der Einsatz sowohl unter UNIX als auch unter Windows 95/NT (unter Verwendung von CYGWIN) möglich.

5.1.1 Datenbank-Anbindung

Die Klassenstruktur der Datenbank-Anbindung ist in Abbildung 5.1 dargestellt. Die Anbindung besteht hauptsächlich aus verschiedenen Parsern, die aus den vorverarbeiteten Datenbanktabellen die relevante Information extrahieren und entsprechend den vorliegenden Bedürfnissen zusammenstellen. Die verschiedenen Parser und zugehörigen Datenstrukturen sollen nun näher beschrieben werden. Es ist zu beachten, daß der zugrundeliegende Parser (`code_parser`) speziell auf die vorverarbeiteten Dateien abgestimmt ist.[1] Die anderen Klassen zur Regeldarstellung sind aber unabhängig von der Art und Weise, wie hier die Anbindung an die Datenbanken vorgenommen wurde.

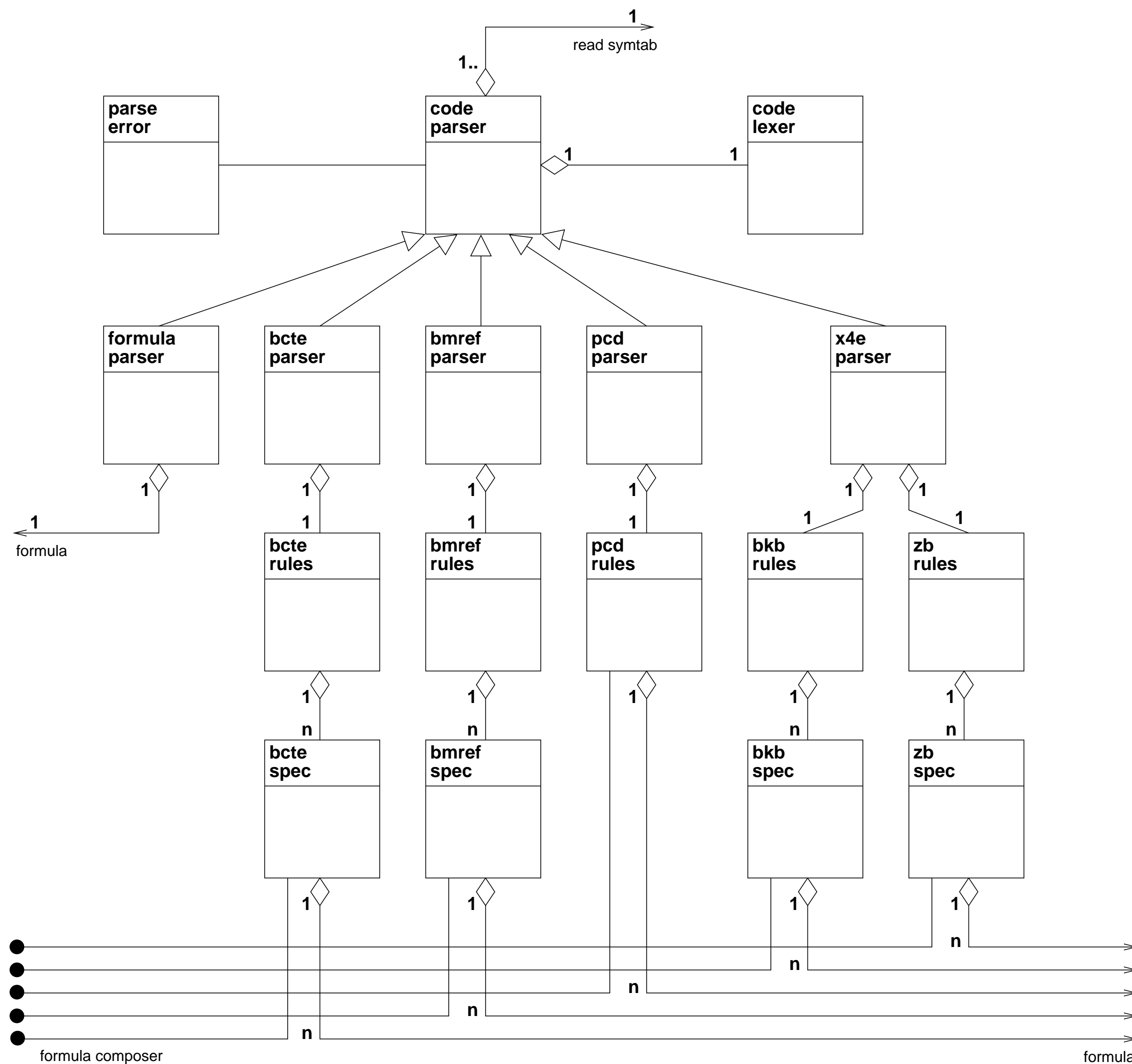


Abbildung 5.1: Klassendiagramm: Datenbank-Anbindung

`pcd_parser`: Der PCD-Parser erlaubt das Auslesen der baureihenunabhängigen Baubarkeitsregeln (ursprünglich in der Datenbanktabelle PCD dargestellt). Diese werden dann in der zugehörigen Datenstruktur (`pcd_rules`, weiter unten beschrieben) abgelegt. Die Datenstruktur ist

[1]Mit einem Perl-Skript werden in diesem Vorverarbeitungsschritt die ASCII-Datenbankabzüge auf das Wesentliche reduziert.

hier besonders einfach, da weder Lenkungen noch Ausführungsarten berücksichtigt werden müssen.

`x4e_parser`: Dieser Parser dient der Anbindung an die X4E Datenbanktabelle. Diese enthält neben den baureihenabhängigen Baubarkeitsregeln auch die Zusteuerungsregeln. Die beiden Regeltypen spiegeln sich in der Unterteilung in `bkb_rules` und `zb_rules` wieder, die der Darstellung der Regeln inklusive der ihnen zugeordneten Informationen wie Gruppe, Position, Variante und Lenkung dienen.

`bmref_parser`: Die Umsetzung der Baumuster in Baureihen und Codes sind in der Datenbanktabelle BMREF abgelegt. Dieser Parser ermöglicht das Auslesen dieser Information, die dann in den entsprechenden Datenstrukturen abgelegt wird.

`bcte_parser`: Die Tabelle BCTE enthält die eigentliche Stückliste, zusammen mit den (kurzen) Coderegeln, die die Teileauswahl steuern. Diese Regeln werden zusammen mit der strukturbildenden Information (Modul, Position, Variante) in der `bcte_rules`-Klasse abgelegt.

`formula_parser`: Dieser Parser ist unabhängig von den Datenbanken zum Lesen von Formeln aus Dateien oder von der Standard-Eingabe vorgesehen. Das Eingabeformat ist dasselbe, wie es auch in den vorverarbeiteten Datenbanktabellen Verwendung findet.

Alle Parser sind so ausgelegt, daß sie (über die Basisklasse `code_parser`) von beliebigen C++-Streams lesen können. Der Scanner `code_lexer` ist ein GNU-FLEX++-Scanner, der über eine C++-Schnittstelle angesprochen wird. Bei `parse_error` handelt es sich um eine Fehlerklasse, die zur Ausnahmebehandlung während des Parse-Vorgangs dient. Wenden wir uns nun der internen Verwaltung der aus der Datenbank gelesenen Regeln zu.

`pcd_rules`: Die baureihenunabhängigen Baubarkeitsregeln werden hier gespeichert. Sie sind als Abbildung von Codes auf Formeln organisiert, d.h. jedem Code wird seine pauschale Codebedingung zugeordnet. Durch die Darstellung als Abbildung ist ein schneller Zugriff anhand der Codes möglich. Intern werden sie bei der derzeitigen STL-Implementation als Red-Black-Trees abgelegt. Da die pauschalen Codebedingungen nicht weiter strukturiert sind, ist keine zusätzliche Information erforderlich.

`bkb_rules` und `bkb_spec`: Diese Klassen dienen der Darstellung der baureihenabhängigen Baubarkeitsregeln. Ähnlich den pauschalen Codebedingungen sind auch hier die Baubarkeitsregeln schnell über den ihnen zugeordneten Code zu erreichen. Da zu einem Code aber verschiedene Formeln vorhanden sein können, die zu unterschiedlichen Positionen, Varianten und Lenkungsausführungen gehören, ist jedem Code nicht nur eine Formel zugeordnet, sondern mehrere `bkb_spec`-Objekte. Diese beinhalten neben der Baubarkeitsformel auch deren Gruppe, Position, Variante und Lenkung. Für jede Teilformel der Baubarkeitsbedingung eines Codes wird ein solches `bkb_spec`-Objekt angelegt. Alle `bkb_spec`-Objekte eines Codes, also alle dessen Teilformeln, sind in einer Liste zusammengefaßt. Ein `bkb_rules`-Objekt enthält dann eine Abbildung von Codes auf diese Listen. Man erhält somit einen schnelle Zugriff auf alle Teilformeln der Baubarkeitsbedingung eines Codes.

`zb_rules` und `zb_spec`: Die Zusteuerungsregeln sind in Objekten dieser beiden Klassen abgelegt. Die Organisation ist annähernd analog zu den entsprechenden Klassen für die Baubarkeitsregeln. Auch hier werden Teilformeln der Zusteuerungsbedingung eines Codes in Listen von `zb_spec`-Objekten gespeichert. Diese enthalten neben Gruppe, Position, Variante und Lenkung auch noch das in den Datenbanken enthaltene Zusteuerungskennzeichen (CG: Codegebunden, BG: Baureihen-gebunden). Auch hier ist die Liste der `zb_spec`-Objekte eines Codes über die Abbildung in der Klasse `zb_rules` zu erreichen.

`bmref_rules` und `bmref_spec`: Alle Informationen, die zur Umsetzung eines Baumusters in eine Baureihe benötigt werden, sind hier abgelegt. In jedem `bmref_spec`-Objekt ist neben dem

Baumuster eine Formel abgelegt. Diese Formel ist die Konjunktion aller dem Baumuster zugeordneten Codes, was der Semantik der Umsetzung entspricht. Einzige Ausnahme ist die Lenkung, die nicht in dieser Formel mit enthalten ist. Alle Baumuster/Formel-Paare einer bestimmten Lenkungsvariante sind nämlich in einer Liste zusammengefaßt. Ein `bmref_rules`-Objekt speichert solche Listen für jede Lenkungsvariante. Die drei Listen für Links- und Rechtslenker sowie unspezifizierte Lenkung sind dabei in einer (Lenkungs-indizierten) Abbildung enthalten.

`bcte_rules` und `bcte_spec`: Die Stückliste ist in diesen beiden Klassen organisiert. Jedes `bcte_spec`-Objekt faßt die Daten eines Stücklisteneintrags zusammen. Neben der (kurzen) Coderegel ist die Position, die Positionsvariante und die Lenkungsvariante gespeichert. Alle in einem Stücklisten-Modul enthaltene Information ist in einer Liste solcher `bcte_spec`-Objekte gespeichert. Ein `bcte_rules`-Objekt enthält die Daten der kompletten Stückliste. In einer Modul-indizierten Abbildung sind zu jedem Modul dessen `bcte_spec`-Objekte abgelegt. Aufgrund der gewählten Darstellung kann man die zu einem Modul gehörigen Daten schnell ansprechen.

Die Zusammenstellung der Daten zu neuen Formeln wird erst später beschrieben, da die entsprechende Klasse `formula_composer` erst im nächsten Teil, der den Formelmanipulator beschreibt, auftritt.

5.1.2 Aussagenlogischer Formelmanipulator

Der Formelmanipulator enthält den von den Datenbanken unabhängigen Teil der Bibliothek.

Verschiedene Darstellungen für Formeln werden angeboten: Die Standardmethode stellt Formeln als Bäume dar. Die Knoten bestehen dabei aus Formelkonstruktoren, die Blätter sind boolesche Variablen oder Konstanten. Als alternative Darstellungen stehen BDDs und Klauselmengen (genauer: Klausel-Listen) zur Verfügung. Die Struktur dieses Teils der Bibliothek ist in dem Klassendiagramm in Abbildung 5.2 veranschaulicht.

Wenden wir uns zuerst der Standarddarstellung über Bäume (oder auch Terme) zu. Formeln in dieser Darstellung können vom Benutzer verwendet werden, ohne daß er sich um die Speicherverwaltung kümmern muß. Intern wird ein Mark&Sweep Garbage-Collector verwendet, der zusammen mit dem `formula_manager` für eine transparente Darstellung sorgt. Bevor die Speicherverwaltung genauer beschrieben wird, soll die Funktion der einzelnen Klassen näher erläutert werden.

`formula_node`: Von dieser Klasse können keine Objekte angelegt werden, da es sich um eine virtuelle Basisklasse der im folgenden beschriebenen speziellen Formelknoten handelt. Gemeinsame Konzepte aller Knoten wie Speicherverwaltung und Knotentypinformation sind in dieser Klasse zusammengefaßt.

`cf_node`: Formelknoten dieser Klasse sind boolesche Konstanten. Es gibt hiervon nur die beiden $\top$ und $\bot$ repräsentierenden Objekte.

`lit_node`: Ein Objekt der Klasse `lit_node` speichert boolesche Variablen. Jede Variable ist mit einem (nicht negativen) ganzzahligen Index versehen.

`uf_node`: Dieser Knotentyp dient der Darstellung von Negationen. Er enthält lediglich einen Zeiger auf die (negierte) Subformel.

`cxf_node`: Konjunktionen und Disjunktionen werden durch Objekte dieser Formelknoten-Klasse dargestellt. Um eine flache Darstellung von langen Konjunktionen und Disjunktionen zu ermöglichen, kann ein solcher Knoten beliebig viele Nachfolgerknoten haben. Zeiger auf diese sind in einer Liste gespeichert. Knoten dieser Art werden auch als komplexe Knoten bezeichnet.

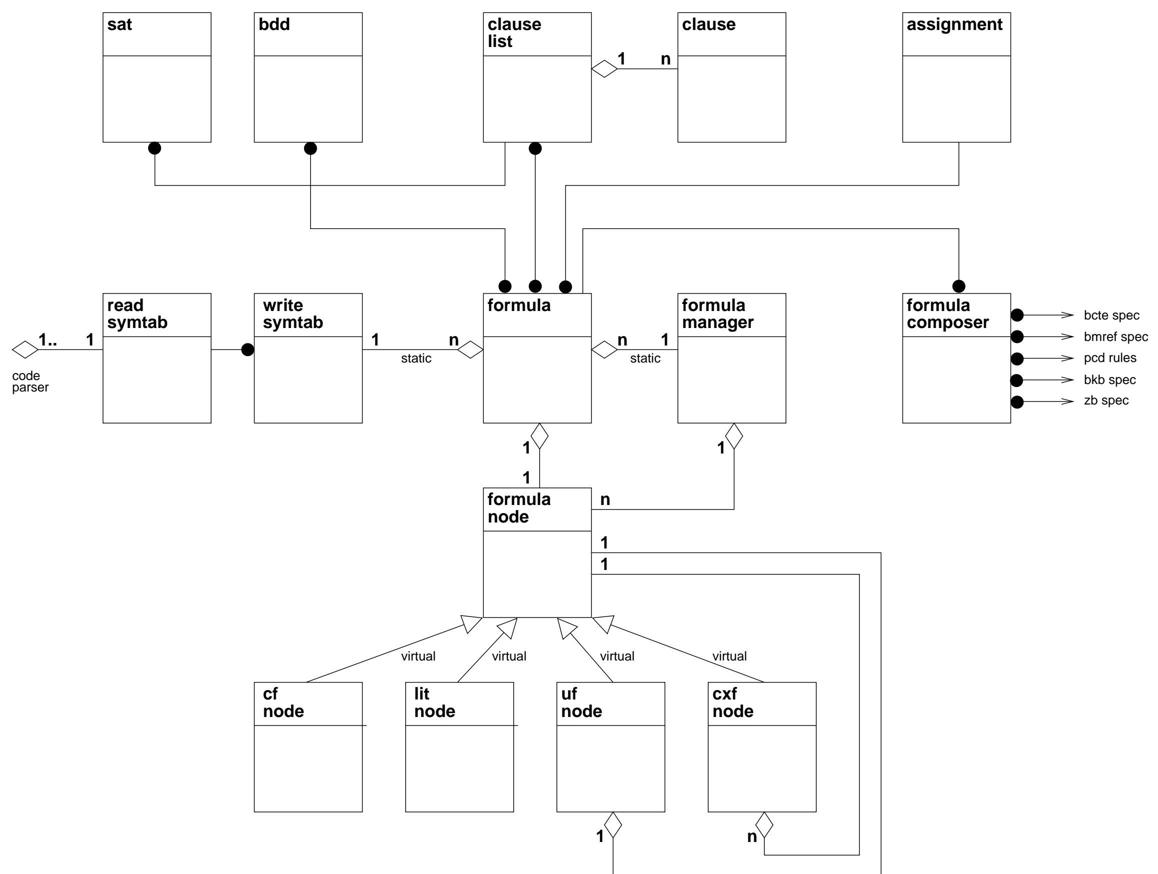


Abbildung 5.2: Klassendiagramm: Aussagenlogischer Formelmanipulator

formula: Diese Klasse ist eine Handle-Klasse zu `formula_node`. Sie ist die dem Benutzer zur Verfügung gestellte Klasse und erlaubt Formeln als einheitliche Objekte aufzufassen. Die komplette Speicherverwaltung ist vor dem Anwender verborgen. Komposition von neuen Formeln aus schon bestehenden, Normalformtransformationen, Restriktionen und Bewertungen sind nur ein Teil der vorhandenen Funktionalität. Jedes `formula`-Objekt enthält einen Zeiger auf den Wurzelknoten der dargestellten Formel.

formula_manager: Diese Klasse dient der Verwaltung der Formelknoten. Sie sorgt für eine eindeutige Darstellung der booleschen Konstanten und Variablen, löst bei Bedarf Garbage-Collections aus und stellt Funktionen auf Formelknoten zur Verfügung. Für alle Formeln wird ein einheitlicher Manager verwendet.

Nach diesem groben Überblick über die verschiedenen Klassen der Standarddarstellung für Formeln soll nun die Speicherverwaltung genauer beschrieben werden.

Speicherverwaltung

Der `formula_manager` hält neben den eindeutig dargestellten booleschen Konstanten eine Liste von Variablen, Negations- und komplexen Knoten. Die Variablen sind, wie die Konstanten, eindeutig dargestellt, ihre Anzahl muß bei der Initialisierung des Managers festgelegt werden und bleibt konstant. Um Variablen schnell ansprechen zu können, sind diese in einem Vektor gespeichert. Über die Variablenindizes kann dann direkt der entsprechende Formelknoten angewählt werden.

Negations- und komplexe Knoten werden anders gehandhabt. Beim Start des Managers wird eine festgelegte Anzahl von Knoten jedes Typs angelegt und in einer Liste gespeichert. Jedesmal wenn ein neuer Knoten gebraucht wird, kann er dann einer dieser beiden Listen entnommen werden. Zwei Listen werden verwendet um Speicherplatz zu sparen, da komplexe Knoten mehr Speicherplatz belegen als Negationsknoten.

Jeder Knoten verwaltet einen Referenzzähler, der die Anzahl der Zeiger speichert, die von `formula`-Objekten auf diesen Knoten bestehen. Zeiger innerhalb des Managers werden nicht mitgezählt. Nach Freigabe eines `formula`-Objekts werden dessen Knoten nicht sofort freigegeben. Dies passiert erst, wenn keine Referenzen mehr von Außen auf diesen Knoten zeigen und eine Garbage-Collection ausgelöst wurde. Dann werden die Listen der Negations- und komplexen Knoten des Managers durchlaufen und alle erreichbaren Knoten markiert, wobei auch Nachfolgeknoten von erreichbaren Knoten mit berücksichtigt werden. Die nicht erreichbaren Knoten werden danach wieder zur Verfügung gestellt.

Um eine korrekte Speicherverwaltung sicherzustellen, darf auf Formelknoten niemals direkt zugegriffen werden, sondern nur über den Manager. Verwendet man ausschließlich die Handle-Klasse `formula`, so ist diese Bedingung immer erfüllt.

Symboltabellen

Zur Ein- und Ausgabe von Formeln werden Symboltabellen verwendet, die die Umsetzung von Variablenindizes auf die in der Datenbank verwendeten Namen vornehmen. Für Ein- und Ausgabe sind dazu aus Effizienzgründen unterschiedliche Symboltabellen vorgesehen.

read_symtab: Die Eingabe-Symboltabelle wird beim Lesen von Formeln verwendet und muß daher für jeden Bezeichner, der schon einmal vorgekommen ist, dessen Variablenindex feststellen. Daher wird für diese Tabelle eine Abbildung von Bezeichnern auf Variablenindizes verwendet. Die Verwendung einer Hash-Tabelle oder eines Tries (discrimination tree) anstatt der Abbildung wäre ebenfalls denkbar.

write_symtab: Bei der Ausgabe sind die Erfordernisse genau umgekehrt: Zu einem gegebenen Index muß der Variablenname (Bezeichner) nachgeschlagen werden. Daher wird bei dieser

Symboltabelle eine Abbildung von Indizes auf Bezeichner verwendet. Typischerweise wird nach dem Einlesen der Formeln die Ausgabe-Symboltabelle aus der Eingabe-Symboltabelle generiert.

Da verschiedene Datenbanktabellen von unterschiedlichen Parsern ausgelesen werden, liegen normalerweise auch mehrere Symboltabellen (für jeden Parser eine) vor. Da dies oft nicht wünschenswert ist, können sich mehrere Parser auch eine einzige Symboltabelle teilen.

Alternative Darstellungsmöglichkeiten für Formeln

Neben der Darstellung von Formeln als Bäume sind in der Bibliothek eine Klausel-Darstellung und eine BDD-Darstellung vorgesehen.

Die BDD-Darstellung bietet neben den üblichen Formelkonstruktoren und Konvertierungsroutinen von und zur Standarddarstellung auch Konvertierungen in konjunktive und disjunktive Normalform an. Diese Konvertierungsfunktionen werden auch von den Normalform-Funktionen der Baum-Darstellung verwendet. Die Klasse `bdd` beinhaltet die BDD-Darstellung zusammen mit den kurz umrissenen Methoden.

Die Klauseldarstellung ist notwendig für den Einsatz des Davis-Putnam-Beweisers, kann aber auch unabhängig davon verwendet werden. Dazu stehen die Klassen `clause` und `clause_list` zur Verfügung. In diesen werden Klauseln als Listen von ganzen Zahlen dargestellt. Für negative Literale werden negative Zahlen, für positive Literale positive Zahlen verwendet. Die Zahl 0 wird nicht verwendet. Es ist naheliegend, daß für Klauselmengen (`clause_list`) Listen von Klauseln verwendet werden.

Die Konvertierung einer Formel in eine Klauselmenge ist zur Zeit nur möglich, wenn die Formel schon in konjunktiver Normalform vorliegt. Zur Berechnung dieser Normalformen werden von der Klasse `formula` die entsprechenden Funktionen bereitgestellt. Von den in 4.6 vorgestellten Methoden wurden die Standard-Methode, die BDD-Konvertierung (genauer: ZDD) und die als "Mix" bezeichnete Variante implementiert.

Die Klasse `sat` bildet die Schnittstelle zum Davis-Putnam-Beweiser SATO. Für eine gegebene Formel in Klausel-Form wird die Erfüllbarkeit überprüft und im positiven Fall ein Modell der Formel zurückgeliefert. Verschiedene Kombinationsmöglichkeiten von Klauselmengen können ebenfalls verwendet werden. Beispielsweise kann eine Klauseldatei hinzugeladen werden.

Die `assignment`-Klasse stellt Variablenbelegungen dar und kann dazu verwendet werden, Formeln unter bestimmten Belegungen auszuwerten. Unter anderem ist damit die Baubarkeitskontrolle einzelner Aufträge möglich.

Letztendlich dient die Klasse `formula_composer` zur Erstellung komplexer Formeln anhand der aus der Datenbank extrahierten Regeln. Die Teilformeln der Baubarkeit eines Codes oder auch die der Zusteuerung können hier zu den Gesamtformeln zusammengesetzt werden. Dabei sind verschiedene Einschränkungen und Interpretationen der Regeln möglich.

5.2 Prototypen von Testprogrammen

Aufbauend auf der gerade beschriebenen C++-Bibliothek wurde eine Reihe von Programm-Prototypen entwickelt, um die anfangs aufgeworfenen Probleme bearbeiten zu können. Diese Programme und die verwendeten Algorithmen werden in diesem Abschnitt vorgestellt.

Die folgenden Programme wurden implementiert:

aufchk: Überprüfung einzelner Aufträge, wie bisher schon bei Mercedes-Benz üblich, ist mit diesem Programm möglich. Dabei durchläuft der Auftrag zuerst die Zusteuerung, danach wird die Baubarkeitskontrolle vorgenommen. Ziel bei der Erstellung dieses Programms war es, eine möglichst genau Simulation des bestehenden Verfahrens zu erreichen.

codchk: Die Suche nach notwendigen und unzulässigen Codes erlaubt diese Programm. Einzelne Codes, die in jedem Auftrag vorkommen müssen oder in keinem baubaren Auftrag vorkommen dürfen, werden von **codchk** gefunden.

gencls: Dieses Programm generiert Klauseldarstellungen von immer wieder benötigten Formeln: Die komplette Baubarkeitsformel oder die Formel, die voll zugesteuerte Aufträge beschreibt, können so einmal im Voraus berechnet und dann von verschiedenen Programmen verwendet werden.

minchk: Mit diesem Programm kann man feststellen, ob nach Zusteuerung und Baubarkeitskontrolle Aufträge mit bestimmten Codemustern ("Miniaufträge", Nebenbedingungen) auftreten können. Dieses Programm kann auch interaktiv zur Überprüfung der Stückliste eingesetzt werden.

modchk: Zur Überprüfung verschiedener Aspekte der Stückliste kann **modchk** verwendet werden. Unnötige Teile, Aufträge, die mehrere Varianten an einer Stücklistenposition auswählen und vieles mehr, findet das Programm.

ordchk: Eine Untersuchung der Abhängigkeit von der Zusteuerungsreihenfolge erlaubt dieses Programm. Problematische Aufträge und Codes werden gefunden und angezeigt.

pcdchk: Dieses Programm hat eine ähnliche Funktion wie **codchk**. Allerdings werden nur die pauschalen Codebedingungen überprüft. Die Ergebnisse von **pcdchk** sind daher für alle Baureihen und Ausführungsarten gültig.

zustchk: Mit diesem Programm kann nach Aufträgen gesucht werden, die durch die Zusteuerung von baubaren zu nicht mehr baubaren transformiert werden. Solche Aufträge deuten auf Fehler in der Zusteuerungskomponente hin.

Der schematische Ablauf einer Konsistenzprüfung, wie er bei den meisten der genannten Programme auftritt, ist in Abbildung 5.3 dargestellt.

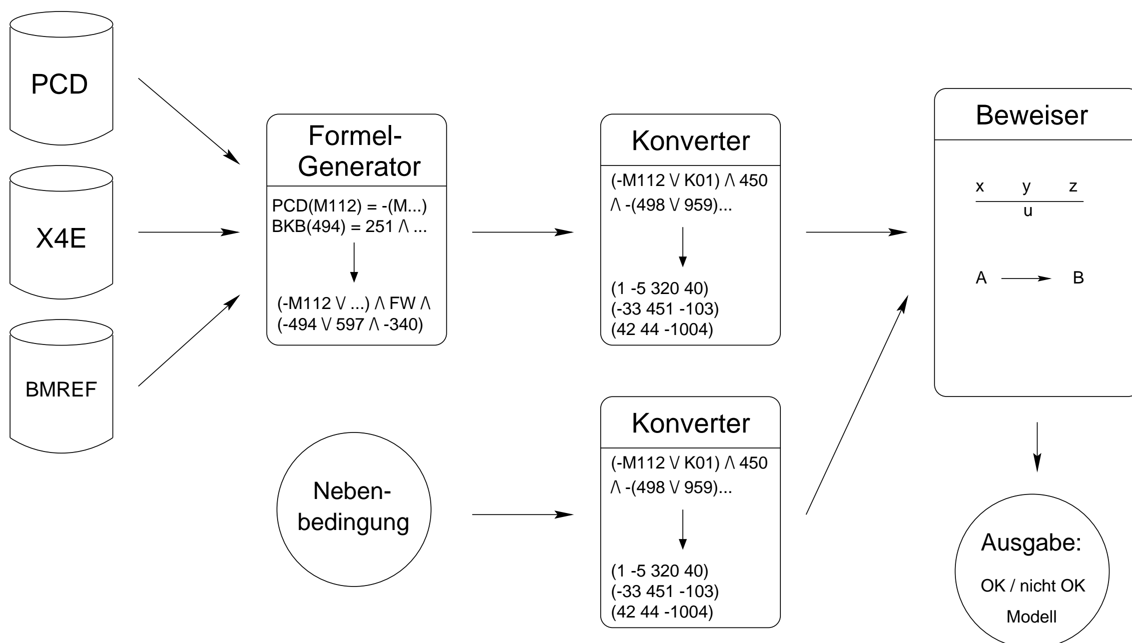


Abbildung 5.3: Schematischer Ablauf einer Konsistenzprüfung

Formeln und strukturelle Information aus den Tabellen X4E, PCD und BMREF der Datenbank der Produktdokumentation werden im Formelgenerator zu einer großen Formel zusammengesetzt. Diese Formel wird dann durch den Konverter, der in unserem Fall Klauseln generiert, für den Beweiser aufbereitet. Dazu kommen dann üblicherweise zusätzliche Bedingungen. Soll beispielsweise nach unnötigen Teilen in der Stückliste gesucht werden, so werden hier die langen Codebedingungen einer Position dazugenommen. Diese Nebenbedingung wird ebenfalls durch einen Konverter in eine für den Beweiser handhabbare Form gebracht. Der fügt dann die konvertierten Datenbankauszüge und Nebenbedingungen zusammen und sucht nach Aufträgen mit den geforderten Eigenschaften. Typischerweise besteht die Ausgabe dann zum einen aus der Antwort, ob es solche Aufträge gibt oder nicht, zum anderen aber auch aus Beispielen oder Gegenbeispielen.

aufchk: Baubarkeitstest einzelner Aufträge

Dieses Programm simuliert den bisher bei Mercedes-Benz schon vorhandenen Test einzelner Aufträge. Dabei wird zuerst ein Auftrag eingelesen, und dieser dann den beiden Zusteuerungsprozessen zugeführt: Zuerst wird eine codegebundene Zusteuerung, danach eine baureihengebundene vorgenommen. Dazu werden die entsprechenden Varianten der Zusteuerbedingungen unter der durch den Auftrag festgelegten Bewertungsfunktion ausgewertet. Ist eine Variante erfüllt, so wird der Code zugesteuert. Die Anzahl der Durchläufe für jede Zusteuerungsart kann eingestellt werden.

Nach der Zusteuerung werden die Baubarkeitsbedingungen der Codes des Auftrags überprüft. Die pauschalen Codebedingungen müssen nur ausgewertet werden, für die (baureihenabhängigen) Baubarkeitsbedingungen wird die Formel eines jeden Codes durch die Klasse `formula_composer` zusammengestellt.

Weder die Zusteuerung noch die Baubarkeitskontrolle sind auf Geschwindigkeit optimiert, dieses Programm dient in erster Linie zu Vergleichszwecken mit der vorhandenen Methode.

codchk: Suche nach unzulässigen und notwendigen Codes

Bei diesem Programm werden einzelne Codes auf ihre Zulässigkeit hin überprüft. Dazu wird für jeden Code $c \in \mathcal{C}$ die Allgemeingültigkeit der folgenden beiden Formeln getestet:[2]

$$\mathcal{B}_K \wedge \mathcal{Z}_V \Rightarrow c \qquad \text{und} \qquad \mathcal{B}_K \wedge \mathcal{Z}_V \Rightarrow \neg c$$

Ist die erste Formel allgemeingültig, so kommt Code c in jedem Auftrag vor, ist die zweite allgemeingültig, so darf Code c in keinem baubaren Auftrag vorkommen. Der Tautologietest wird durch Negation der Formel und Überprüfung der Unerfüllbarkeit bewerkstelligt. Aus der ersten Formel wird so durch Negation $\mathcal{B}_K \wedge \mathcal{Z}_V \wedge \neg c$, aus der zweiten $\mathcal{B}_K \wedge \mathcal{Z}_V \wedge c$. Die Klauseldarstellung der Baubarkeit und Zusteuerung ist, gemäß Abbildung 5.3, durch das Programm `gencls` schon vorhanden, so daß der Davis-Putnam-Beweiser nur mit diesen Klauseln und einer zusätzlichen (c oder $\neg c$) aufgerufen werden muß. Eine Konvertierung in Klauselform muß nicht bei jedem Schritt neu vorgenommen werden.

Offensichtlich müssen nicht alle Codes auf beide Möglichkeiten hin getestet werden, da sie sich gegenseitig ausschließen (die Erfüllbarkeit von $\mathcal{B}_K \wedge \mathcal{Z}_V$ vorausgesetzt). Weitere Tests lassen sich vermeiden, wenn man die generierten Modelle jedes einzelnen Beweises mitberücksichtigt.

Der implementierte Algorithmus sieht wie folgt aus:

[2] Alternativ kann die Formel $\mathcal{B}_R$ mitberücksichtigt werden.

```
function check_codes(code-set C) returns code-set-pair
   C0+ = C   // upper bound for not admissible codes
   C0- = ∅   // lower bound for not admissible codes
   C1+ = C   // upper bound for necessary codes
   C1- = ∅   // lower bound for necessary codes
   X = (C0+ \ C0-) ∪ (C1+ \ C1-)           // test code set
   while X ≠ ∅ do
      choose c ∈ X
      if c ∈ C0+ do                        // check whether c is admissible
         if satisfiable(B_K ∧ Z_V ∧ c) do
            get satisfying model M
            C0+ = C0+ \ M
            C1+ = C1+ \ (C \ M)
         else C0- = C0- ∪ {c}              // code c is not admitted
         endif
      endif
      if c ∈ C1+ do                        // check whether c is necessary
         if satisfiable(B_K ∧ Z_V ∧ ¬c) do
            get satisfying model M
            C0+ = C0+ \ M
            C1+ = C1+ \ (C \ M)
         else C1- = C1- ∪ {c}              // code c is necessary
         endif
      endif
      X = (C0+ \ C0-) ∪ (C1+ \ C1-)        // new test code set
   end
   return (C0+, C1+)                       // not admitted, necessary codes
end
```

Dieser Algorithmus terminiert, da die Testmenge X in jedem Schleifendurchlauf kleiner wird. Das erfüllende Modell M ist bei diesem Algorithmus die Menge aller mit 1 belegten Variablen.

genlcs: Der Klausel-Generator

Das Programm **genlcs** generiert häufig verwendete Klauselmengen. Insbesondere die zur Formel $\mathcal{B}_K \wedge \mathcal{Z}_V$ gehörige Klauselmenge wird immer wieder benötigt. In Abbildung 5.3 entspricht die von **gencls** bereitgestellte Funktionalität dem oberen Teil aus Formelgenerator und Konverter.

Genauer verwendet das Programm die vom `formula_composer` erstellten Regeln und baut daraus die Baubarkeitsformel auf. Diese wird dann mit den BDD-basierten Konvertierungsalgorithmen in konjunktive Normalform gebracht. Aus dieser konjunktiven Normalform werden danach Klauseln generiert und in einer Datei abgelegt. Diese Datei kann dann vom Davis-Putnam-Beweiser in den verschiedenen Testprogrammen wieder eingelesen werden.

Derzeit kann das Programm die folgenden Formeln (mit kleinen Varianten) erzeugen:

$$\begin{array}{cc} \mathcal{B}_P & \mathcal{B}_P \wedge \mathcal{B}_R \\ \mathcal{B}_P \wedge \mathcal{B}_L \wedge \mathcal{B}_B & \mathcal{B}_P \wedge \mathcal{B}_L \wedge \mathcal{B}_B \wedge \mathcal{B}_R \\ \mathcal{B}_P \wedge \mathcal{Z}_V & \mathcal{B}_P \wedge \mathcal{B}_R \wedge \mathcal{Z}_V \\ \mathcal{B}_P \wedge \mathcal{B}_L \wedge \mathcal{B}_B \wedge \mathcal{Z}_V & \mathcal{B}_P \wedge \mathcal{B}_L \wedge \mathcal{B}_B \wedge \mathcal{B}_R \wedge \mathcal{Z}_V \end{array}$$

Die möglichen Varianten beziehen sich auf die Behandlung von Codes ohne Baubarkeitsbedingung und Einschränkungen der Lenkungsvariante.

minchk: Baubarkeitstests mit Nebenbedingungen

Dieses Programm erlaubt die in 3.4 angegebenen baubaren, voll zugesteuerten Aufträge mit Nebenbedingungen zu untersuchen. Die dort angegebene Formel $\mathcal{B}_K \wedge \mathcal{Z}_V \wedge F_N$ für verschiedene Nebenbedingungen wird mit dem Davis-Putnam-Beweiser auf Erfüllbarkeit untersucht. F_N kann beliebig gewählt werden und wird dann, wie bei dem Programm **codchk** der schon in Klauseldarstellung vorliegenden Baubarkeitsformel hinzugefügt. Ist die Formel erfüllbar, gibt es also Aufträge mit der Nebenbedingung F_N, so wird ein Beispielauftrag ausgegeben.

Typische Zeiten für diesen Baubarkeitstest liegen unter einer Sekunde. Daher ist das Programm auch gut geeignet, interaktiv zur Kontrolle der Stückliste eingesetzt zu werden. Dies ist möglich, da die Auswahl von Positionsvarianten der Stückliste anhand der an Schnittstelle ② vorliegenden Aufträgen vorgenommen wird. Bei Änderungen an der Stückliste kann dieses Programm wertvolle Dienste leisten.

modchk: Konsistenzprüfung der Stückliste

Das Programm **minchk** ist durch minimale Erweiterungen in der Lage, in der Stückliste nach Inkonsistenzen zu suchen. Diese Inkonsistenzen können

- nie benötigte Teile,
- mehrere gleichzeitig ausgewählte Varianten oder
- freibleibende Positionen

sein. Dazu werden verschiedene Testformeln (entsprechend den Nebenbedingungen) von **modchk** generiert, die die erwähnten Inkonsistenzen aufspüren.

Diese Testformeln verwenden die langen Coderegeln, die demnach zuerst aus den Stücklisten-Regeln (mittels `formula_composer`) erzeugt werden müssen.

Nehmen wir an, daß für eine Position die Varianten $v_1, \ldots, v_n$ mit zugeordneten langen Coderegeln $r_1, \ldots, r_n$ in der Datenbank vorhanden sind. Eine nie verwendete Variante v_i, und damit ein nicht benötigtes Teil der Stückliste, ist dann an der Unerfüllbarkeit der Baubarkeitsformel mit Nebenbedingung $F_N = r_i$ zu erkennen. Analog zeigt die Erfüllbarkeit der Baubarkeitsformel mit Nebenbedingung $F_N = r_i \wedge r_j$ für $i \neq j$ an, daß es Aufträge gibt, die gleichzeitig die Varianten v_i und v_j anwählen. Eine Position bleibt bei manchen Aufträgen ganz frei, wenn mit der Nebenbedingung $F_N = \neg(r_1 \vee \cdots \vee r_n)$ eine erfüllende Belegung gefunden wurde.

ordchk: Reihenfolgeabhängigkeit der Zusteuerung

Die Reihenfolgeabhängigkeit der Zusteuerung kann in dem in 3.5 beschriebenen Rahmen mit dem Programm **ordchk** untersucht werden.

Die dort angegebenen Formeln KA und RV wurden allerdings etwas modifiziert, um zusätzliche Informationen erhalten zu können. Mit dieser neuen Formel stellt sich das Problem dann so dar:

Eine Reihenfolgenabhängigkeit kann zwischen den beiden Zusteuerungsmöglichkeiten der Codes c_1 und c_2 bestehen, falls

$$\bigwedge_{l \in \{L,R\}} \left(l \Rightarrow \left(\mathcal{Z}(c_1, a_A, l)|_{c_2=\bot} \wedge \mathcal{Z}(c_2, a_A, l)_{c_1=\bot} \;\Rightarrow\; \mathcal{Z}(c_1, a_A, l)|_{c_2=\top} \wedge \mathcal{Z}(c_2, a_A, l)_{c_1=\top} \right) \right)$$

erfüllbar ist. Im Programm **ordchk** wird, wenn die Bedingung für zwei Codes gilt, noch ein Baubarkeitstest mit der angegebenen Formel als Nebenbedingung nachgeschaltet. So können Mehrdeutigkeiten ausgefiltert werden, die sowieso keinen baubaren Aufträgen zugeordnet sind.

Die Erfüllbarkeit obiger Formel wird nicht mit der Davis-Putnam-Prozedur berechnet, sondern über BDDs. Da für den nachfolgenden Baubarkeitstest die Formel sowieso in Klauselform gebracht werden muß, bekommt man den Erfüllbarkeitstest praktisch geschenkt.

pcdchk: Unzulässige Codes unabhängig von der Baureihe

Das Programm **pcdchk** ist praktisch analog zu **codchk** aufgebaut, nur werden in diesem Fall lediglich die pauschalen Codebedingungen betrachtet. Auch die Zusteuerung fällt hier unter den Tisch.

Man testet also die Allgemeingültigkeit der Formel

$$\mathcal{B}_P \Rightarrow \neg c.$$

Der zweite Test ist nicht erforderlich, da die pauschale Codebedingung für den leeren Auftrag immer wahr ist. **codchk** sucht also nur nach nicht zulässigen Codes.

Da hier weniger zu testende Codes vorhanden sind, wird auch nicht der bei **codchk** angegebene Algorithmus verwendet, sondern die Codes einfach der Reihe nach als zusätzliche Klauseln hinzugefügt.

zustchk: Konsistenzprüfung der Zusteuerung

Die Konsistenzprüfung der Zusteuerung soll nachweisen, daß durch die Zusteuerung kein baubarer Auftrag zu einem nicht mehr baubaren umgebaut wird. Dazu betrachtet man für jeden Code $c \in \mathcal{C}_Z$ die folgende Formel:

$$\left(\mathcal{B}_K \wedge \bigwedge_{l \in \{L,R\}} (l \Rightarrow \mathcal{Z}(c, a_A, l))\right) \Rightarrow \mathcal{B}_K|_{c=\top}$$

Diese Formel drückt aus, daß wenn ein Auftrag baubar ist und eine Zusteuerung von Code c möglich, auch der zugesteuerte Auftrag baubar ist. Nun muß die Allgemeingültigkeit dieser Formel gezeigt werden, was äquivalent zur Unerfüllbarkeit von

$$\mathcal{B}_K \wedge \bigwedge_{l \in \{L,R\}} (l \Rightarrow \mathcal{Z}(c, a_A, l)) \wedge \neg(\mathcal{B}_K|_{c=\top})$$

ist. Die Formel $\mathcal{B}_K$ liegt schon in Klauseldarstellung vor, muß also nicht weiter beachtet werden. Problematischer ist die Teilformel $T = \neg(\mathcal{B}_K|_{c=\top})$. Da $\mathcal{B}_K$ eine große Konjunktion ist, ist T eine Disjunktion, eine Konvertierung in CNF also schwierig. Man kann die Formel T aber vereinfachen, denn T kann (bei entsprechender Benennung der Baubarkeits-Teilformeln B) als

$$T = \neg\left(B(c)|_{c=\top} \wedge \bigwedge_{\substack{x \in \mathcal{C} \\ x \neq c}} (x \Rightarrow B(x)|_{c=\top})\right)$$

dargestellt werden. Dann ist

$$T = \neg B(c)|_{c=\top} \vee \bigvee_{\substack{x \in \mathcal{C} \\ x \neq c}} (x \wedge \neg B(x)|_{c=\top})$$

Von der großen Disjunktion muß man nur die Subformeln betrachten, in denen es ein negatives c-Vorkommen in $B(x)$ gibt. Die Menge aller x mit dieser Eigenschaft sei N. Dann ist die endgültige Formel für T

$$T = \neg B(c)|_{c=\top} \vee \bigvee_{\substack{x \in N \\ x \neq c}} (x \wedge \neg B(x)|_{c=\top}),$$

die zu überprüfende Gesamtformel also

$$\mathcal{B}_K \wedge \bigwedge_{l \in \{L,R\}} (l \Rightarrow \mathcal{Z}(c, a_A, l)) \wedge \left(\neg B(c)|_{c=\top} \vee \bigvee_{\substack{x \in N \\ x \neq c}} (x \wedge \neg B(x)|_{c=\top}) \right).$$

Diese Formel wird von dem Programm **zustchk** für alle $c \in \mathcal{C}_Z$ generiert und überprüft. Ist sie für ein c erfüllbar, so erhält man als Modell ein Beispiel, bei dem die Zusteuerung einen Auftrag zerstört.

Alle implementierten Programme sind nun beschrieben. Insgesamt ermöglichen sie eine Vielzahl von Konsistenz-Checks, die ohne Beweiser-Einsatz nicht möglich gewesen wären. Von den am Anfang dieser Arbeit aufgeworfenen Problemstellungen konnte jede einer zumindest annähernd vollständigen Lösung zugeführt werden.

Die mit diesen Programmen erhaltenen Ergebnisse sind im nächsten Kapitel zusammengefaßt.

Kapitel 6

Ergebnisse

In diesem Kapitel sollen die Ergebnisse zusammengefaßt werden, die eine Untersuchung der Produktdokumentation anhand der Limousinen der Baureihe C202 ergeben hat. Die zu Beginn dieser Arbeit aufgeworfenen möglichen Inkonsistenten seinen zuvor nochmals kurz zusammengestellt.

- Unzulässige und notwendige Codes
- Reihenfolgeabhängigkeit der Zusteuerung
- Inkonsistenzen in der Zusteuerung
- Unnötige Teile der Stückliste
- Mehrdeutigkeiten in der Stückliste

Durch die im letzten Kapitel beschriebenen Testprogramme konnte jeder dieser fünf Problemfälle bearbeitet werden. In vielen Fällen ist ohne genaueres Hintergrundwissen allerdings nicht klar, ob es sich bei den aufgedeckten Inkonsistenzen um Fehler in der Produktdokumentation oder beabsichtigte Spezialfälle handelt.

6.1 Unzulässige und notwendige Codes

Zwei unabhängig von der Baureihe nicht baubare Codes wurden mit dem Programm `pcdchk` gefunden:

Code	Begründung
513	pausch. Codebed. von Code 513
930	pausch. Codebed. von Code 930

Speziell bei der betrachteten Baureihe C202, Ausführungsart FW, waren die folgenden Codes nicht baubar, die mit dem Programm `codchk` ermittelt wurden:

Code	Begründung
924L	Code 826 nicht baubar
009U	Code 024 nicht baubar

Eine Reihe anderer Codes waren aufgrund fehlender Baubarkeitsbedingungen nicht baubar. Diese seien der Vollständigkeit halber hier benannt:

F163, F129, F140, F168, F170, F203, F208, F210, F211, F215, F220, 420, 425, G330, G337, G338, G372, G373, G374, G375, G376, G377, G378, G379, G380, G381, G383, G384, G385, G388, G389, G391, G392, G393, G395, G396, G438, G439, G500, G501, G502, G503, G504, G505, G700, G701, ME01, M14, M16, M17, M19, M27, M29, M30, M32, M35, M42, M50, M58, M60, M003, M102, M119, M120, M131, M137, M166, M602, M603, M606, M612, M613, M668, Q05, Q06, S01, S02, U02, U03, 669, U04, U10, U13, U11, U12, U14, U30, U47, 841, 846, 849, U50, Z001, 979, Z002, Z003, Z005, Z006, Z04, Z06, Z07, 954, 177, 178, 210, 415, 211, 213, 214, 217, 787, 788, 789, 908, 262, 241, 571, 242, 572, 223, 224, 226, 228, 230, 245, 310, 323, 325, 350, 513, 514, 530, 537, 538, 759, 812, 254, 255, 750, 752, 753, 754, 756, 257, 259, 268, 269, 930, 839, 870, 915, 295, 296, 298, 303, 326, 305, 306, 307, 308, 316, 330, 973, 690, 692, 693, 698, 699, 417, 421, 430, 431, 670, 705, 488, 655, 676, 677, 486, 640, 642, 644, 646, 648, 649, 651, 658, 659, 660, 661, 662, 667, 780, 783, 786, 791, 792, 793, 794, 795, 797, 799, 490, 565, 631, 826, 836, 501, 515, 541, 569, 590, 592, 596, 598, 599, 591, 619, 620, 647, 663, 664, 790, 887, 691, 702, 700, 701, 708, 719, 709, 711, 723, 847, 731, 732, 740, 744, 746, 747, 758, 757, 777, 785, 796, 811, 813, 940, 871, 878, 883, 886, 902, 947, 903, 948, 909, 914, 983, 991, 992, 722, 024

Gerade bei diesen Codes ist davon auszugehen, daß es sich um keine Inkonsistenzen handelt, sondern diese Codes absichtlich ausgeschlossen wurden. Es ist aber nicht auszuschließen, daß einzelnen Codes unbeabsichtigt keine Baubarkeitsbedingung zugeordnet wurde, wie man an den letzten Beispielen (z.B. Baubarkeit von 924L) sehen konnte.

Einziger notwendiger Code war der Baureihe-beschreibende Code F202. Bei diesem Code ist ebenfalls davon auszugehen, daß er in jedem korrekten Auftrag dieser Baureihe vorhanden sein muß, die Notwendigkeit dieses Codes also keinen Fehler darstellt.

Code	Begründung
F202	Zusteuerbedingung von Code F202

6.2 Mehrdeutigkeiten in der Zusteuerung

In den folgenden Fällen ist die Zusteuerung abhängig von der Reihenfolge, in der die Zusteuerregeln getestet werden. Die angegebenen Beispielaufträge lassen immer eine Zusteuerung beider Codes zu. Insgesamt traten 10 Codepaare auf, bei denen eine Reihenfolgeabhängigkeit festzustellen war.

Codes GA und GM :

Code 1:	GA
Code 2:	GM
Beispiel-Auftrag:	M104, 957, ...
Problem:	nur einer der beiden Codes kann zugesteuert werden; nach Zusteuerung von GA ist Auftrag baubar, nach Zusteuerung von GM nicht
akt. Zustand:	GA wird zugesteuert, o.k.

Codes GM und 423 :

Code 1:	GM
Code 2:	423
Beispiel-Auftrag:	M112, 494, ...
Problem:	nur einer der beiden Codes kann zugesteuert werden; nach Zusteuerung von 423 ist Auftrag baubar, nach Zusteuerung von GM

nicht
akt. Zustand: 423 wird zugesteuert, o.k.

Codes 2XXL und 828 :

Code 1: 2XXL
Code 2: 828
Beispiel-Auftrag: 200L, 617, ...
Problem: nur einer der beiden Codes kann zugesteuert werden;
nach Zusteuerung von 828 ist Auftrag baubar, nach Zusteuerung von 2XXL nicht
akt. Zustand: 2XXL wird zugesteuert, nicht o.k.

Codes 270 und 531 :

Code 1: 270
Code 2: 531
Beispiel-Auftrag: 314, ...
Problem: nur einer der beiden Codes kann zugesteuert werden;
nach Zusteuerung von 270 ist Auftrag baubar, nach Zusteuerung von 531 nicht
akt. Zustand: 270 wird zugesteuert, o.k.

Codes 570 und 959 :

Code 1: 570
Code 2: 959
Beispiel-Auftrag: M111, M20, 625, ...
Problem: Falls zuerst 570 zugesteuert wird, so kann 959 auch noch zugesteuert werden, wird zuerst 959 zugesteuert, ist eine Zusteuerung von 570 nicht mehr möglich
akt. Zustand: nur 959 wird zugesteuert; unklar, ob o.k.

Codes 593 und 959 :

Code 1: 593
Code 2: 959
Beispiel-Auftrag: M111, M20, 625, 740A, ...
Problem: Falls zuerst 593 zugesteuert wird, so kann 959 auch noch zugesteuert werden, wird zuerst 959 zugesteuert, ist eine Zusteuerung von 593 nicht mehr möglich
akt. Zustand: nur 959 wird zugesteuert; unklar, ob o.k.

Codes 654 und 808 :

Code 1: 654
Code 2: 808
Beispiel-Auftrag: M112, 956, 498, ...
Problem: nur einer der beiden Codes kann zugesteuert werden;
nach Zusteuerung von 808 ist Auftrag baubar, nach Zusteuerung von 654 nicht
akt. Zustand: 808 wird zugesteuert, o.k.

Codes 652 und 656 :

Code 1: 652
Code 2: 656
Beispiel-Auftrag: M20, M111, ...
Problem: nur einer der beiden Codes kann zugesteuert werden; der Auftrag ist in beiden Varianten baubar
akt. Zustand: 652 wird zugesteuert; unklar, ob o.k.

Codes 761 und 762 :

Code 1: 761
Code 2: 762
Beispiel-Auftrag: M111, 498, 817L, ...
Problem: nur einer der beiden Codes kann zugesteuert werden; nach Zusteuerung von 761 ist Auftrag baubar, nach Zusteuerung von 762 nicht
akt. Zustand: 761 wird zugesteuert, o.k.

Codes 583 und 584 :

Code 1: 583
Code 2: 584
Beispiel-Auftrag: 955, 808, ...
Problem: nur einer der beiden Codes kann zugesteuert werden; nach Zusteuerung von 584 ist Auftrag baubar, nach Zusteuerung von 583 nicht
akt. Zustand: 584 wird zugesteuert, o.k.

6.3 Konsistenz der Zusteuerung

18 Zusteuerungsregeln, die Aufträge nicht mehr baubar machen, wurden gefunden. In all diesen Fällen ist der Auftrag ohne Zusteuerung des links angegebenen Codes baubar. Die Zusteuerung nimmt ihn (eventuell, abhängig von der Reihenfolge) hinzu, womit der Auftrag nicht mehr baubar ist.

Code	Beispiel-Auftrag	Begründung
2XXL	F202, GM, M23, M111, 100A, 189U, 200L, 617, L	pausch. Codebed. von Code 617
200B	M20, M604, 781A, 959, 808, 519L, R	Baubark. von 200B in Grp. CLN
202B	M20, M604, 742A, 959, 203L, L	Baubark. von 202B in Grp. CLN
203B	M20, M604, 742A, 959, 516L, L	Baubark. von 202B in Grp. CLN
205B	M20, M604, 925L, L	Baubark. von 205B in Grp. CLN
206B	M20, M604, 511L, L	Baubark. von 206B in Grp. CLN
207B	M20, M604, 514L, L	Baubark. von 207B in Grp. CLN
208B	M20, M604, 533L, L	Baubark. von 208B in Grp. CLN
211B	M20, M604, 601L, L	Baubark. von 211B in Grp. CLN
212B	M20, M604, 701L, L	Baubark. von 212B in Grp. CLN
231B	M20, M604, 719L, L	Baubark. von 231B in Grp. CLN
292	M20, M111, 625, 770L, 808, R	Baubark. von Code 625
583	M23, M111, 640A, 625, 956, 807, R	Baubark. von Code 625
652	M20, M111, 100A, 808, 955, 498, 675, L	Baubark. von Code 675
654	M24, M112, 809, 956, R	Baubark. von Code 956
656	M20, M111, 100A, 808, 955, 675, L	Baubark. von Code 675
762	M20, M111, 200A, 808, 823L, 584, 955, 875L, L	pausch. Codebed. von Code 875L
923	M28, M112, 100A, 494, 2XXL, L	pausch. Codebed. von Code 494

6.4 Konsistenz der Teilebedarfsermittlung

Stücklisten-Positionsvarianten, die von keinem Auftrag ausgewählt werden

Die nachfolgende Liste enthält nur einen Teil der aufgefundenen Positionsvarianten, die von keinem Auftrag verwendet werden. Eine komplette Darstellung aller 517 Varianten hätte zu viel Platz benötigt. In der folgenden Tabelle bedeutet "kein Modell mit ...", daß es keine möglichen Aufträge gibt, die nach Zusteuerung und Baubarkeitskontrolle noch die entsprechende Eigenschaft (oder Nebenbedingung) haben.

In den angegebenen Formeln ist "∗" als Konjunktions-, "+" als Disjunktions- und "−" als Negationssymbol verwendet.

Modul	Pos.	Var.	Begründung
60412	775	1	Kein Modell mit 957*-772
100404	4500	1	Kein Modell mit 333*970 (wegen PCD)
100404	4650	1	Kein Modell mit 333*970 (wegen PCD)
100404	4700	1	Kein Modell mit 333*970 (wegen PCD)
100404	4710	1	Kein Modell mit 333*970 (wegen PCD)
101608	100	70	Kein Modell mit M113*494
102008	30	50	Kein Modell mit M112*460
102008	30	60	Kein Modell mit M112*460
120404	300	7	Kein Modell mit M111*M001*220*-M20
120404	300	17	Kein Modell mit 460*040*-M111
120408	1200	5	Kein Modell mit 460*-494
120408	1300	5	Kein Modell mit 460*-494
122004	10	110	Kein Modell mit 612*494*-613
122004	100	120	Kein Modell mit 612*494*-617
122004	1010	110	Kein Modell mit 612*494*-617
122004	1100	120	Kein Modell mit 612*494*-613
122032	2010	1	Kein Modell mit 961*-970*-974*-975
180834	200	999	Kein Modell mit 498*-500
180867	2400	1	Kein Modell mit 930*-(450+934)
180867	2410	1	Kein Modell mit 930*-(450+934)
180867	2415	1	Kein Modell mit 930*-(450+934)
180867	2450	1	Kein Modell mit 930*-(450+934)
180867	2460	1	Kein Modell mit 930*-(450+934)
180867	2470	1	Kein Modell mit 930*-(450+934)
180867	2480	1	Kein Modell mit 930*-(450+934)
181208	2900	10	Kein Modell mit 930
181220	500	2	Kein Modell mit M112*-423*R
181240	520	2	Kein Modell mit M112*-423*R
220814	2000	3	Kein Modell mit (M111+M604)*423*-M23*-M20*-M18*-M22
220820	100	8	Kein Modell mit M605*M25*-M002
220822	100	10	Kein Modell mit M605*M25*-M002
220828	100	7	Kein Modell mit M605*M25*-M002
240404	125	330	Kein Modell mit M604*906*-654*-656*-675
240404	470	280	Kein Modell mit M604*906*-643*-653*-654
240404	475	280	Kein Modell mit M605*906*-653*-654*-657
240408	125	330	Kein Modell mit M604*906*-654*-656*-675
240408	470	280	Kein Modell mit M604*906*-643*-653*-654
240408	475	280	Kein Modell mit M605*906*-653*-654*-657
240412	125	30	Kein Modell mit M605*906*-653*-654*-657

Modul	Pos.	Var.	Begründung
240412	130	100	Kein Modell mit M605*498
240412	130	180	Kein Modell mit M605*498
240412	470	20	Kein Modell mit M604*906*-643*-653*-654
240412	475	20	Kein Modell mit M605*906*-653*-654*-657
240412	480	20	Kein Modell mit M611*906
261624	100	10	Kein Modell mit 024*956
261632	340	3	Kein Modell mit 715*L
281204	5440	3	Kein Modell mit M112*494*809
281608	100	3	Kein Modell mit M112*494*809
281608	340	2	Kein Modell mit M112*494*809
281608	400	1	Kein Modell mit M112*494*809
281608	420	1	Kein Modell mit M112*494*809
281608	440	1	Kein Modell mit M112*494*809
300416	200	999	Kein Modell mit 472*-M112
300416	240	999	Kein Modell mit 472*-M112
300416	400	999	Kein Modell mit 472*-M112
300416	800	999	Kein Modell mit 472*-M112
300416	840	999	Kein Modell mit 472*-M112
490080	210	1	Kein Modell mit 718*580*-423
490080	220	1	Kein Modell mit M111*M20*718*-423
490080	230	1	Kein Modell mit M104*703*-(420+423)
490080	270	1	Kein Modell mit M104*M28*704*-423
490420	10	4	Kein Modell mit M605*-M002
490420	20	4	Kein Modell mit M605*-M002
490420	740	2	Kein Modell mit M112*-423*R
490428	10	6	Kein Modell mit M605*-M002
490428	10	9	Kein Modell mit M111*M22*-807
490800	70	3	Kein Modell mit 420 (X4E)
490800	140	1	Kein Modell mit 420 (X4E)
491620	1460	4	Kein Modell mit M605*-M002
491620	1500	2	Kein Modell mit M605*-M002
492000	120	3	Kein Modell mit 420 (X4E)
492000	120	4	Kein Modell mit 420 (X4E)
492404	100	10	Kein Modell mit M104*625
492408	100	11	Kein Modell mit 460*-M104*-M111*-M112
492804	100	2	Kein Modell mit 420 (X4E)
492808	60	1	Kein Modell mit 420 (X4E)
492808	70	2	Kein Modell mit 420 (X4E)
492808	120	1	Kein Modell mit 420 (X4E)
492808	160	1	Kein Modell mit 420 (X4E)
492808	200	1	Kein Modell mit 420 (X4E)
492808	260	2	Kein Modell mit 420 (X4E)
492808	360	1	Kein Modell mit 420 (X4E)
492808	380	1	Kein Modell mit 420 (X4E)
492808	400	1	Kein Modell mit 420 (X4E)
492808	420	1	Kein Modell mit 420 (X4E)
492808	440	1	Kein Modell mit 420 (X4E)
492808	460	1	Kein Modell mit 420 (X4E)
492808	500	1	Kein Modell mit 420 (X4E)
492812	100	8	Kein Modell mit M112*-423*R
492812	300	8	Kein Modell mit M112*-423*R
492816	200	1	Kein Modell mit 420 (X4E)
492816	320	2	Kein Modell mit 715*-M111

Modul	Pos.	Var.	Begründung
492816	340	2	Kein Modell mit 715*-M111
503040	40	4	Kein Modell mit M605*-M002
503040	60	4	Kein Modell mit M605*-M002
503040	80	3	Kein Modell mit M605*-M002
503500	140	2	Kein Modell mit M605*-M002
505520	100	2	Kein Modell mit M605*-M002
506510	1180	2	Kein Modell mit M605*-M002
600001	120	1	Kein Modell mit 715*-581
600001	130	1	Kein Modell mit 718*-580
600001	140	1	Kein Modell mit 718*-423
600001	160	1	Kein Modell mit 706*-580
600002	100	2	Kein Modell mit 706*-580
600404	100	2	Kein Modell mit 706*-580
630003	100	3	Kein Modell mit 420 (X4E)
630003	100	6	Kein Modell mit 420 (X4E)
630003	100	8	Kein Modell mit 420 (X4E)
630802	100	2	Kein Modell mit M111*(704+703+706)*423*-580
700004	300	2	Kein Modell mit M112*706
800001	60	3	Kein Modell mit M104*704*654
800002	100	999	Kein Modell mit (703+...+719)*-M111*-M611*-M112...
800002	120	999	Kein Modell mit (703+...+719)*-M111*-M611*-M112...
800002	1100	7	Kein Modell mit M104*704*654
800002	1200	7	Kein Modell mit M104*704*654
800006	100	7	Kein Modell mit M104*704*654*VL
800008	200	7	Kein Modell mit M104*704*654*VR
802008	100	5	Kein Modell mit (703+...+718)*-M111*-M112*-M113...
802412	100	2	Kein Modell mit M111*M001*-494*...
802412	100	6	Kein Modell mit M111*494*-(653+654+675)...
900001	100	3	Kein Modell mit HA*M111*M18*706*-580
900001	100	4	Kein Modell mit HA*M111*M18*706*423*-580
900001	160	1	Kein Modell mit HA*M111*M20*718*-423
900001	310	1	Kein Modell mit HA*M104*704*-423
900001	320	2	Kein Modell mit HA*M104*703*-423
900002	170	50	Kein Modell mit M104*703*-423
900008	170	50	Kein Modell mit M104*703*-423*HA
900008	170	52	Kein Modell mit M104*704*HA*-423
900404	30	50	Kein Modell mit M111*M20*(704+718)*423*HA*M001
900404	30	51	Kein Modell mit M111*M20*(704+718)*423*HA*M001
900404	30	99	Kein Modell mit M111*M20*706*HA*M001
900428	140	8	Kein Modell mit 420
900804	180	7	Kein Modell mit M604*-M20*-M22
901212	400	5	Kein Modell mit HA*M111*M20*704*M001
901212	430	3	Kein Modell mit HA*M111*M20*704*M001
901608	100	100	Kein Modell HA*(703+...+718)*-M111...
902412	10	3	Kein Modell mit M111*M001*M18
902412	10	5	Kein Modell mit M111*M001*M18
902412	10	9	Kein Modell mit M111*M001*M18
902412	10	11	Kein Modell mit M111*M001*M18
902412	10	13	Kein Modell mit 718*M18
902412	1000	2	Kein Modell mit M111*M001*M18
902412	1000	5	Kein Modell mit M111*M001*M18
902416	160	2	Kein Modell mit M104*956*494*-(-715+-M111...)

Mehrdeutigkeiten in der Auswahl einer Positionsvariante

Es wurden 264 Mehrdeutigkeiten gefunden. Diese sind, wiederum aus Platzgründen, nur teilweise wiedergegeben.

Modul	Pos.	Lenk.	Var. 1	Var. 2
20808	160	L	2	3
20808	160	R	2	3
20808	161	L	2	3
20808	161	R	2	3
20808	1600	L	1	2
20808	1600	R	1	2
40484	460	L	1	2
40832	225	R	1	2
80420	6000	L	2	10
80420	6000	R	2	10
80432	500	L	1	2
80432	510	L	1	2
80616	1620	L	10	20
80616	1620	R	10	20
300800	100	R	1	5
300800	100	R	2	3
300800	100	R	2	4
300800	100	R	2	5
300800	100	R	3	4
300800	100	R	3	5
300800	100	R	4	5
301208	500	L	1	2
301208	500	R	1	2
301208	660	L	1	2
301208	660	R	1	2
301208	780	L	1	2
301208	780	R	1	2
490004	110	R	0	4
490004	110	R	0	10
490004	120	L	0	7
490004	120	L	0	20
490004	210	R	0	3
490004	210	R	0	14
490004	220	L	0	3
490004	220	L	0	25
490004	400	L	0	2
490004	400	L	0	20
490004	410	R	0	5
490004	500	L	0	2
490004	500	L	0	10
490004	510	R	0	4
490004	800	L	0	11
490004	810	R	0	7
490004	5000	L	0	3
490004	5000	L	0	24

Modul	Pos.	Lenk.	Var. 1	Var. 2
490004	5010	R	0	2
490004	5010	R	0	5
490004	5020	L	0	3
490004	5020	L	0	11
490004	5200	L	0	7
490004	5200	L	0	9
490004	5210	R	0	3
490408	1200	L	1	2
490408	1200	R	1	2
490420	420	L	1	2
490420	420	R	1	2
490420	520	L	1	2
490420	520	R	1	2
490420	2220	L	1	2
490420	2220	R	1	2
490424	1200	L	1	2
490424	1200	R	1	2
490816	3020	L	1	4
492408	140	L	2	3
492808	320	L	2	3
492808	320	R	2	3
500005	100	R	1	3
500005	100	R	2	4
500005	200	R	1	5
500005	200	R	2	4
500005	200	R	3	6
500005	300	R	1	5
500005	300	R	2	6
500005	300	R	3	7
500005	300	R	4	8
500005	400	R	1	2
500005	400	R	3	4
500005	420	L	1	3
500005	1000	R	1	2
500005	1100	L	1	2
500005	3000	R	1	4
500005	3000	R	2	3
802008	100	L	1	3
802008	100	R	1	3
802008	120	R	1	2
802008	140	R	1	2
802008	1120	R	1	2
802008	1140	R	1	2
802416	125	L	4	10
802416	125	R	4	10
802416	130	L	2	7
802416	130	R	2	7
900400	3000	L	4	5
900400	3000	R	4	5
900400	3030	L	1	2
900428	190	L	1	4
900428	190	R	1	4
900804	200	L	1	5

Modul	Pos.	Lenk.	Var. 1	Var. 2
900804	200	R	1	5
901212	1400	L	1	2
901212	1400	L	1	3
901212	1400	L	2	3
901212	1400	R	1	2
901212	1400	R	1	3
901212	1400	R	2	3
901212	1420	L	1	2
901212	1420	R	1	2
901212	1440	L	1	2
901212	1440	R	1	2
901212	1600	L	1	2
901212	1600	L	1	3
901212	1600	L	2	3
901212	1600	R	1	2
901212	1600	R	1	3
901212	1600	R	2	3
901212	1620	L	1	2
901212	1620	R	1	2
901212	1640	L	1	2
901212	1640	R	1	2
903212	140	L	1	2
903212	140	R	1	2
903212	160	L	1	2
903212	160	R	1	2
903212	200	L	1	2
903212	200	R	1	2

Kapitel 7

Zusammenfassung und Aussichten

In dieser Arbeit wurde die Anwendbarkeit formaler Methoden zur Konsistenzprüfung der Produktdokumentation der Mercedes-Benz Fahrzeuge untersucht. Durch deren Einsatz wurden Möglichkeiten eröffnet, die ohne formale Spezifikation und Verifikation nicht erreichbar gewesen wären.

Neben der ursprünglich anvisierten Suche nach Inkonsistenzen in der Stückliste konnten viele weitere Kontrollmöglichkeiten geschaffen werden. Diese erlauben neben der Feststellung von notwendigen und unzulässigen Codes auch eine Untersuchung der Reihenfolgeabhängigkeit der Zusteuerung sowie das Auffinden von (möglicherweise) fehlerhaften Zusteuerungsregeln.

Die Formalisierung machte Probleme deutlich, die bei der Beschreibung ähnlich komplexer Prozeßabläufe wohl immer wieder zu beobachten sind:

- Durch die komplizierte Struktur und beträchtliche Größe sind die Systeme schlecht zu überschauen und zu warten. Fehler sind leicht zu verursachen und schwer zu finden.
- Da viele verschiedene Personen, auch mit unterschiedlichem Wissensstand, diese Systeme verwenden und modifizieren, ist es kaum möglich, einen bestimmten erwünschten Zustand über längere Zeit aufrechtzuerhalten.

Bei der Untersuchung der Produktdokumentation manifestierten sich diese Probleme in den gefundenen Inkonsistenzen ebenso wie in den Ausnahmefällen, die bei der Formalisierung zu berücksichtigen waren: Bei der Baubarkeitskontrolle ist beispielsweise die Behandlung nicht angegebener Codes nicht einheitlich geregelt. Daneben kann bei der Generierung der langen Coderegeln das Vorhandensein negierter Codes zu Mehrdeutigkeiten führen.

Doch gerade die durch solche Unklarheiten entstehenden Fehlerquellen lassen sich durch den Einsatz formaler Methoden leichter finden.

Der Vergleich verschiedener Beweisverfahren hat in erster Linie deutlich gemacht, daß bewährte Methode aus benachbarten Gebieten nicht bedenkenlos übernommen werden können. So lieferte der Einsatz von BDDs nicht dieselben guten Ergebnisse wie bei der Hardware-Verifikation. Es scheint keine einheitlichen, einfachen Kriterien zu geben, wann ein bestimmter Beweiser erfolgreich einzusetzen ist und wann nicht.

Die überzeugenden Ergebnisse unter Verwendung des Davis-Putnam-Verfahrens hängen in erster Linie mit der Struktur des Problems (und damit der Formeln) zusammen. Wie in [US94] festgestellt wurde, ist dieses für Erfüllbarkeitsprobleme mit uneinheitlichen Lösungen besser geeignet als binäre Entscheidungsdiagramme.

Ohne eine schnelle Konvertierungsroutine für konjunktive Normalformen wäre manches der erzielten Resultate so nicht möglich gewesen. Durch den Einsatz von BDDs zur Konvertierung wurden enorme Beschleunigungen erreicht. Dies läßt vermuten, daß auch in anderen Bereichen durch die Kombination verschiedener Verfahren noch manche Verbesserung zu erwarten ist.

Durch das Zusammenspiel von binären Entscheidungsdiagrammen und dem Davis-Putnam-Beweiser ist so bei der Überprüfung der Stücklisten sogar interaktive Unterstützung durch die vorgestellten Programme möglich. Die Gesamtheit der Programm-Prototypen läßt für die Zukunft neben der Reduzierung von Fehlern in der Produktdokumentation auch auf Hilfe bei Änderungen an den Datenbanken hoffen.

Anhang A

Statistische Größen

In der folgenden Tabelle sind typische Laufzeiten und Speicherplatzanforderungen der Programm-Prototypen zusammengefaßt. Alle Messungen wurden auf einem Pentium PC mit 100MHz und 64 MB Hauptspeicher durchgeführt.

Programm	Laufzeit	Speicher
Generieren der Beweiser-Daten	7-8 min.	ca. 35 MB
notw./unzulässige Codes suchen	ca. 19 min.	ca. 9 MB
Mehrdeutigkeiten in Zusteuerung	ca. 8 min.	ca. 20 MB
Konsistenz der Zusteuerung	ca. 17 min.	ca. 28 MB
Unnötige Teile/Mehrdeutigkeiten in SL	ca. 30 h	ca. 35 MB
Beispiel-Aufträge mit Nebenbed.	1-3 sec.	5-10 MB

Die Dateigrößen der ASCII-Abzüge der Datenbanktabellen, der daraus mittels eines Perl-Programms erzeugten, komprimierten Darstellungen und der Klauseldatei der Baubarkeitsformel sind in der nächsten Tabelle angegeben.

Datenbank-Abzüge	11,2 MB
ohne Stückliste	485 KB
komprimiert	747 KB
ohne Stückliste	96 KB
Baubarkeitsformel	480 KB
mit Zusteuerung	591 KB

Die folgende Tabelle enthält schlußendlich noch statistische Angaben zu der generierten Baubarkeitsformel.

Anzahl Codes	1151
Baubarkeitsformel	ca. 140000 Konnektive
mit Zusteuerung	ca. 170000 Konnektive

Literaturverzeichnis

[Ake78] S. B. Akers. Binary decision diagrams. In *IEEE Transactions on Computers*, volume C-27(6), pages 509–516, June 1978.

[BC95] R. E. Bryant and Y.-A. Chen. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of the 32nd Design Automation Conference*, June 1995.

[BFG+93] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the International Conference on Computer-Aided Design*, pages 188–191, Santa Clara, CA, Nov. 1993.

[BH94] J. P. Bowen and M. G. Hinchey. Seven more myths of formal methods: Dispelling industrial prejudices. In M. Naftalin, T. Denvir, and M. Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, volume 873 of *Lecture Notes in Computer Science*, pages 105–117. Springer-Verlag, 1994.

[BHMR95] B. Beckert, R. Hähnle, N. V. Murray, and A. Ramesh. Anti-links for boolean function manipulation, 1995. Workshop on Computational and Propositional Logic, KI95.

[BKL96] R. Bündgen, W. Küchlin, and W. Lauterbach. Verification of the sparrow processor. In *1996 IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, pages 86–93. IEEE Computer Society Press, Mar. 1996.

[BM96] E. Börger and S. Mazzanti. A correctness proof for pipelining in RISC architectures. Technical Report YY-NN, DIMACS, Pisa, Feb. 1996.

[Bor97] Arne Borälv. The industrial success of verification tools based on Stålmarck's method. In O. Grumberg, editor, *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 7–10. Springer-Verlag, 1997.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computers*, volume C-35(8), pages 677–691, Aug. 1986.

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. In *ACM Computing Surveys*, volume 24(3), pages 293–318, Sept. 1992.

[Bün93] R. Bündgen. Reduce the redex → ReDuX. In C. Kirchner, editor, *RTA'93: Rewrite Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 446–450, Montreal, Canada, June 1993. Springer-Verlag.

[Bün97] R. Bündgen. Termersetzungssysteme, 1997. Vorlesungsskriptum.

[BW96] B. Bollig and I. Wegner. Improving the variable ordering of OBDDs is NP-complete. In *IEEE Transactions on Computers*, volume 45(9), pages 993–1002, Sept. 1996.

[CMF93] O. Coudert, J. C. Madre, and H. Fraisse. A new viewpoint on two-level logic minimization. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 625–630, Dallas, TX, June 1993.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[DB95] R. Drechsler and B. Becker. *PUMA: An OKFDD-Package and its Implementation.* Johann Wolfgang Goethe-Universität, Frankfurt am Main, 1995. (Part of the PUMA package).

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. In *Communications of the ACM*, volume 5, pages 394–397, July 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. In *Journal of the ACM*, volume 7, pages 201–215, 1960.

[DST+94] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski. Efficient representation and manipulation of switching functions based on ordered kronecker functional decision diagrams. In *Proceedings of the Design Automation Conference*, pages 415–419, 1994.

[FOH93] H. Fujii, G. Ootomo, and C. Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 38–41, 1993.

[Fre95] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms.* PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, May 1995.

[FS90] S. J. Friedman and K. J. Supowit. Finding the optimal variable ordering for binary decision diagrams. In *IEEE Transactions on Computers*, volume 39(5), pages 710–713, May 1990.

[FYBSV93] E. Felt, G. York, R. Brayton, and A Sangiovanni-Vincentelli. Dynamic variable reordering for BDD minimization. In *Proceedings of the EuroDAC'93*, pages 130–135, 1993.

[Gar95] E. H. A. Garcez. The verification of an ATM switching fabric using the HSIS tool. Technical Report WSI-95-13, Universität Tübingen, 1995.

[GK97] Alfons Geser and Wolfgang Küchlin. Structured formal verification of a fragment of the IBM 390 Clock Chip. Technical Report 97-50, RISC-Linz Report Series, Schloß Hagenberg bei Linz, Austria, Oct. 1997.

[Har96] J. Harrison. The stålmarck method as a HOL derived rule. In *TPHOL'96: Theorem Proving in Higher Order Logic*, pages 221–234. Springer Verlag, 1996.

[HJ89] W. A. Hunt Jr. Microprocessor design verification. In L. Wos, editor, *Journal of Automated Reasoning*, volume 5, pages 429–460. Kluwer Academic Publications, Dec. 1989.

[Hoa96] C. A. R. Hoare. How did software get so reliable without proof? In M.-C. Gaudel and J. Woodcock, editors, *FME'96: Industrial Benefit and Advances in Formal Methods*, volume 1051 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, 1996.

[Hsi82] J Hsiang. *Topics in Automated Theorem Proving and Program Generation.* PhD thesis, University of Illinois, Urbana, Illinois, Dec. 1982.

[Hus85] H. Hussmann. Rapid prototyping for algebraic specifications – RAP-system user's manual. Technical Report MIP-8504, Universität Passau, Mar. 1985. (Second Edition).

[HV95] J. N. Hooker and V. Vinay. Branching rules for satisfiability. In *Journal of Automated Reasoning*, volume 15(3), pages 359–383, 1995.

[KB70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra*. Pergamon Press, 1970.

[KSR92] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic based on functional decision diagrams. In *Proceedings of the European Design Automation Conference*, pages 43–47, 1992.

[Lee59] C. Y. Lee. Representation of switching circuits by binary-decision programs. In *Bell System Technical Journal*, volume 4, pages 985–999, July 1959.

[Lon93] D. E. Long. *Model Checking, Abstraction and Compositional Verification*. PhD thesis, School of Computer Science, Carnegie Mellon University, July 1993.

[Lon94] D. E. Long. *BDDLIB: A BDD package*. E-mail: long@research.att.com, 1994. (Part of the BDDLIB package).

[Mar97] F. E. Marschner. Practical challenges for industrial formal verification tools. In O. Grumberg, editor, *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 1–2. Springer-Verlag, 1997.

[McC94] W. W. McCune. OTTER 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, Illinois, Jan. 1994.

[Mer95] Mercedes-Benz Personenwagen, Abt. NED/GFP. *DIALOG: Die neue Produktdokumentation, Ergänzungswerk*, Nov. 1995. (interne Dokumentation).

[Min93] S.-I. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 272–277, Dallas, TX, June 1993.

[Mon76] J. D. Monk. *Mathematical Logic*, volume 37 of *Graduate Texts in Mathematics*. Springer-Verlag, 1976.

[Moo92] J. S. Moore. Introduction to the OBDD algorithm for the ATP community. Technical Report 84, Computational Logic, Inc., Austin, Texas, Oct. 1992.

[New42] M. H. A. Newman. On theories with a combinatorial definition of “equivalence”. In *Annals of Mathematics*, volume 43, pages 223–243. Princeton University Press, 1942.

[NS89] P. Narendran and J. Stillman. Formal verification of the Sobel image processing chip. In G. Birtwistle and P. A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 92–127. Springer, New York, NY, 1989.

[Pra95] V. Pratt. Anatomy of the Pentium bug. In *TAPSOFT’95: Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 97–107, New York, 1995. Springer-Verlag.

[PT96] R. Pugliese and E. Tronci. Automatic verification of a hydroelectric power plant. In M.-C. Gaudel and J. Woodcock, editors, *FME’96: Industrial Benefit and Advances in Formal Methods*, volume 1051 of *Lecture Notes in Computer Science*, pages 425–444. Springer-Verlag, 1996.

[Ric78] M. M. Richter. *Logikkalküle*, volume 43 of *Teubner-Studienbücher: Informatik*. Teubner, Stuttgart, 1978.

[RMBH95] A. Ramesh, N. V. Murray, B. Beckert, and R. Hähnle. Fast subsumption checks using anti-links. Technical Report 24/95, Universität Karlsruhe, Fakultät für Informatik, Apr. 1995.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. In *Journal of the ACM*, volume 12, pages 23–41, 1965.

[Rud93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 42–47, 1993.

[Sha38] C. E. Shannon. A symbolic analysis of relay and switching circuits. In *Trans. AIEE*, volume 57, pages 713–723, 1938.

[SL94] A. A. Stepanov and M. Lee. The standard template library. Technical Report HPL-94-34, Hewlett-Packard Laboratories, Apr. 1994.

[Sla94] J. Slaney. The crises in finite mathematics: Automated reasoning as cause and cure. In A. Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 1–13. Springer-Verlag, 1994.

[SM95] M. K. Srivas and S. P. Miller. Formal verification of the AAMP5 microprocessor. In M. G. Hinchey and J. P. Bowen, editors, *Applications of Formal Methods*, International Series of Computer Science, pages 125–180. Prentice Hall, 1995.

[Soc91] R. Socher. Optimizing the clausal normal form transformation. In *Journal of Automated Reasoning*, volume 7, pages 325–336. Kluwer Academic Publishers, 1991.

[Som97] F. Somenzi. *CUDD: CU Decision Diagram Package, Release 2.1.2.* University of Colorado, Boulder, 1997. (Part of the CUDD package).

[SS89] R. C. Sekar and M. K. Srivas. Formal verification of a microprocessor using equational techniques. In G. Birtwistle and P. A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 171–217. Springer-Verlag, New York, NY, 1989.

[Stå92] G. Stålmarck. A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula. Swedisch Patent No. 467 076 (approved 1992), U.S. Patent No. 5 276 897 (1994), European Patent No. 0403 454 (1995), 1992.

[THY93] S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable ordering of a shared binary decision diagrams. In *Proceedings of the 4th ISAAC*, volume 762 of *Lecture Notes in Computer Science*, pages 389–398, New York, 1993. Springer-Verlag.

[TPP97] A. L. Turk, S. T. Probst, and G. J. Powers. Verification of a chemical process leak test procedure. In O. Grumberg, editor, *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 84–94. Springer-Verlag, 1997.

[US94] T. E. Uribe and M. E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In *1st International Conference on Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*. Springer-Verlag, Sept. 1994.

[ZBH96] H. Zhang, M. P. Bonacina, and J. Hsiang. PSATO: A distributed propositional prover and its application to quasigroup problems. In *Journal of Symbolic Computation*, volume 21, pages 543–560. Academic Press, 1996.

[Zha93] H. Zhang. SATO: A decision procedure for propositional logic. In *Association for Automated Reasoning Newsletter*, volume 22, pages 1–3, Mar. 1993.

[Zha97] H. Zhang. SATO: An efficient propositional prover. In *CADE'97: 14th International Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[ZS94] H. Zhang and M. Stickel. Implementing the Davis-Putnam algorithm by tries. Technical report, Department of Computer Science, The University of Iowa, Iowa City, IA, Aug. 1994.

[ZS96] H. Zhang and M. Stickel. An efficient algorithm for unit propagation. In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, 1996.