Diss. ETH 5941

INFORMATIONSVERDICHTUNG VON MODELLEN IN DER AUSSAGENLOGIK UND DAS P = NP - PROBLEM

ABHANDLUNG

zur Erlangung

des Titels eines Doktors der Mathematik

der

EIDGENOESSISCHEN TECHNISCHEN HOCHSCHULE ZUERICH

vorgelegt von

KARL LIEBERHERR
dipl. Math. ETH
geboren am 27. August 1948
von Nesslau/SG

angenommen auf Antrag von
Prof.Dr. E. Engeler, Referent
Prof.Dr. E. Specker, Korreferent

1977

Sehr herzlich moechte ich Herrn Prof. Dr. E. Engeler fuer die stets wohlwollende Betreuung meiner Arbeit und Herrn Prof. Dr. E. Specker fuer die Uebernahme des Korreferates danken. Meiner Frau Dr. R. Lieberherr, meinen Kollegen H.R. Thomann, Dr. R. Schoenberger, E. Graf, M. Wietlisbach und M. Fuerer danke ich fuer ihre Kritik und ihre Anteilnahme an meiner Arbeit. Dankbar bin ich A. Mueller fuer das Trennungsprogramm, das er mir zur Verfuegung stellte.

Weiterhin danke ich allen Professoren und den Sekretaerinnen des Instituts fuer Informatik und meinen Institutskollegen fuer das angenehme Arbeitsklima, das meine Arbeit indirekt unterstuetzt hat.

Abstract:

The thesis deals with the problem of deciding, whether a given formula of the propositional calculus is satisfiable. In the first part a deduction rule, called Superresolution, is introduced and compared with known proof-systems and decision procedures. In the second part probabilistic algorithms for NP-complete languages are investigated.

Keywords:
Superresolution, decision procedures for Satisfiability, polynomial reduction of proof-systems, length of proofs, lower bounds, probabilistic algorithms for NP-complete languages.

Zusammenfassung

In dieser Arbeit wird die Komplexitaet des Entscheidens, ob eine konjunktive Normalform (KNF) der Aussagenlogik erfuellbar ist, analysiert. Dieses Entscheidungsproblem hat eine zentrale Stellung in einer grossen Klasse von kombinatorischen Entscheidungs- resp. Optimierungsproblemen.

Die bisher untersuchten Beweissysteme fuer die unerfuellbaren KNF haben lokalen Charakter. In der Arbeit wird ein Beweissystem Superresolution eingefuehrt, das als global bezeichnet werden darf, d.h. ein Schluss kann von der ganzen bisherigen Formel abhaengen. Superresolution wird verglichen mit einem auf dem Gebiete des automatischen Beweisens haeufig verwendeten Beweissystem, naemlich mit Resolution und deren Abarten. Weil Resolution lokal ist, so ist es einleuchtend, dass eine globale Schlussregel wie Superresolution kuerzere Beweise erlaubt als Resolution. Dies wird in der Arbeit auch bewiesen. Dafuer ist der Aufwand fuer das Finden, resp. Ueberpruefen eines Beweisschrittes groesser als bei Resolution. Beide Aufgaben sind jedoch linear loesbar. Superresolution ist eine grosse Einschraenkung von Resolution, denn keine Input-Resolvente kann eine Superresolvente sein. Deshalb hat Superresolution gegenueber Resolution wesentliche Vorzuege, die in der Arbeit untersucht werden.

Um die Komplexitaet von Superresolution zu analysieren, fuehrt man einen Entscheidungsalgorithmus SR fuer KNF ein, der nur Superresolutionsbeweise in einer kurzen Normalform erzeugen kann. Es stellt sich dabei heraus, dass Superresolution und Resolution polynomial aequivalent sind. Ausserdem wird fuer eine spezielle Art von Superresolution eine exponentielle untere Schranke bewiesen. Neben anderen Vorzuegen hat SR die Tendenz, "einfache" Formeln effizient zu behandeln.

Im zweiten Teil der Arbeit werden probabilistische Entscheidungsalgorithmen fuer NP-vollstaendige Sprachen untersucht. Probabilistische Algorithmen sind keine Algorithmen im eigentlichen Sinn, denn sie verwenden bei ihren Berechnungen Zufallsprozesse und koennen deshalb falsche Entscheide liefern. Wenn die Fehlerwahrscheinlichkeit aber mit relativ wenig Rechenaufwand gegenueber den bekannten deterministischen Verfahren beliebig klein gemacht werden kann, sind probabilistische Algorithmen fuer praktische Anwendungen sehr interessant.

In der Arbeit wird der Begriff "Erfuelltheitsgrad" fuer KNF eingefuehrt und auf NP-vollstaendige Sprachen erweitert. Die Komplexitaet von probabilistischen Algorithmen fuer eine Sprache S haengt direkt mit dem Erfuelltheitsgrad von S zusammen. Die Relation wird durch eine Formel beschrieben, die ausdrueckt: Je groesser der Erfuelltheitsgrad ist, desto effi-

zientere Algorithmen resultieren. Deshalb wird in der Arbeit versucht, den Erfuelltheitsgrad moeglichst gross zu machen. Es wird behauptet und durch verschiedene Ueberlegungen und Hilfssaetze begruendet, dass dies mit wenig Aufwand moeglich ist, beim Versuch, es zu beweisen, stoesst man aber auf erhebliche Schwierigkeiten.

Inhaltsverzeichnis

Bezeichnungen

Dieser Text wurde mit Hilfe einer computergesteuerten Schreibmaschine geschrieben, welche die ueblichen mathematischen Symbole nicht im Zeichensatz hat. Deshalb verwenden wir folgende Bezeichnungen:

abs	abs(a) : absoluter Betrag von a
binomial	binomial(n,k) ist der Binomialkoeffizient n tief k
card	card(a) : die Maechtigkeit der Menge a
in	a in B : a ist ein Element von B
trunc	Fuer nicht negative reelle Zahlen a ist trunc(a) die groesste natuerliche Zahl, die kleiner oder gleich a ist.
**	Potenzierung
{}	leere Menge
+	Vereinigung von Mengen (und Addition)
*	Durchschnitt von Mengen (und Multiplikation)
#	Ende eines Beweises
v	Disjunktion (oder)
&	Konjunktion (und)
´	Negation (a´ ist die negierte Variable a)

Tiefstellungen werden durch eckige oder runde Klammern dargestellt, z.B. a[1] oder a(1).

INFORMATIONSVERDICHTUNG VON MODELLEN IN DER AUSSAGENLOGIK UND DAS P = NP - PROBLEM

****

Uebersicht

****

In dieser Arbeit untersuche ich einen Entscheidungsalgorithmus SR und einige seiner Abwandlungen fuer das Erfuellbarkeitsproblem (ERF) der konjunktiven Normalformen (KNF) der Aussagenlogik. Dabei diente mir folgende Frage als Leitmotiv : Sei s eine erfuellbare KNF mit Modell M und r eine Klausel, die eine Folgerung von s ist. Wie kann man r so waehlen, dass r viele Informationen ueber M enthaelt, d.h. dass M relativ viele Literale in r erfuellt ? Die Modellinformation Inf(r,M) von r in bezug auf M ist der Quotient der Anzahl durch M erfuellter Literale und der Literalanzahl von r. Wenn man die Erzeugung von Klauseln r mit Inf(r,M) groesser als 1/2 iteriert, erhaelt man zuverlaessige Informationen ueber M durch blosse zufaellige Auswahl von Literalen in den erzeugten Klauseln. Diese Tatsache laesst sich dazu verwenden, probabilistische Algorithmen [RAB76,SOL76] zu entwickeln, die Modelle mit vorgegebener Fehlerwahrscheinlichkeit finden, und zwar rascher als die trivialen deterministischen Algorithmen.

Uebersicht ueber die Resultate der Arbeit:

1. Ich habe ein vollstaendiges Beweissystem (Superresolution) fuer unerfuellbare KNF entwickelt, das stets besser ist als Resolution. D.h. fuer jeden Resolutionsbeweis, der nicht trivial ist (d.h. die leere Klausel ist nicht die erste erzeugte Klausel), existiert ein Superresolutionsbeweis, der weniger Klauseln enthaelt. (Es ist zu erwarten, dass im verkuerzten Beweis die Anzahl Anwendungen der Schlussregel "Superresolution" meistens etwa die Quadratwurzel der Anzahl Anwendungen der Schlussregel "Resolution" im unverkuerzten Beweis ist.)
   Eine Anwendung der Schlussregel "Superresolution" auf eine Formel s beruecksichtigt meistens die "ganze" Formel s. Deshalb kann Superresolution als globales Beweissystem betrachtet werden. Das Ueberpruefen eines Superresolutionsbeweisschrittes ist in linearer Zeit moeglich. Superresolution ist in folgendem Sinn staerker als Resolution: Man braucht mindestens 3 Anwendungen von Resolution, um ir-

gendeine Klausel (ausser die leere) zu erhalten, die mittels Superresolution mit einer einzigen Anwendung der Schlussregel erzeugbar ist. Nach meinen Literaturkenntnissen (siehe Bibliographie Teile A und D) sind diese Resultate neu, obwohl schon in verschiedenen Arbeiten nach einem solchen Ergebnis gesucht wurde. Superresolution ist auch besser als die bekannten Verfeinerungen von Resolution wie: semantische Resolution, Hyperresolution, set-of-support-Resolution, Lockresolution, lineare Resolution, Resolution mit Mischen etc. Ich habe Superresolution zu einem Beweissystem "erweiterte Superresolution" ergaenzt und bewiesen, dass "erweiterte Superresolution" stets besser ist als "erweiterte Resolution".

2. Resolution und Superresolution sind zwar als Beweissysteme fuer unerfuellbare KNF polynomial aequivalent, doch kann ich in Abschnitt 1.5 beweisen, dass von einem bestimmten algorithmischen Gesichtspunkt her Superresolution exponentiell besser ist als Resolution.

3. Weiter beweise ich in Abschnitt 1.8 fuer bestimmte Mengen von KNF [TS68, COO76], dass SR auf diesen Mengen nur polynomialen Aufwand hat. Hingegen haben gewisse, nicht-triviale klassische Entscheidungsalgorithmen (Davis-Putnam, regulaere Resolution) fuer diese Klasse erwiesenermassen exponentiellen Aufwand.

4. Es zeigt sich in Abschnitt 1.7, dass auf den bekannten Teilmengen von ERF (Horn-KNF, Krom-KNF, stark erfuellbare KNF), die polynomial entscheidbar sind, SR uniform die Entscheidung in polynomialer Zeit liefert. Dabei werden die eingegebenen KNF nicht in verschiedene Typen eingeteilt, sondern alle werden nach derselben universellen Methode behandelt. Dies deutet an, dass SR die Tendenz hat, sich auf "einfachen" Formeln guenstig zu verhalten. Damit wird die Effizienz von SR noch weiter unterstrichen.

5. Weiter zeige ich, wie Algorithmus SR dazu verwendet werden kann, Boolesche Funktionen beschleunigt zu berechnen (Abschnitt 1.9).

6. Ich zeige, dass die Mittelwertuntersuchung von gewissen Resolutionsbeweissystemen ein System ergibt, das eine besonders gute Verdichtung von Modellinformation garantiert (Abschnitt 2.4).

7. Sei s eine erfuellbare KNF mit Modell M, und der Erfuelltheitsgrad eq(s,M) sei der Bruchteil der von M erfuellten Literale in s. Ich gebe NP-vollstaendige Teilmengen Q von ERF an, so dass fuer alle KNF q in Q und fuer alle Modelle M von q der Erfuelltheitsgrad beliebig nahe 1 ist (Abschnitt 2.3).

8. Es gibt polynomiale Verfahren, denen eine Erfolgswahrscheinlichkeit d zugeordnet werden kann, fuer die gilt: Je groesser d ist, desto effizientere Erkennungsalgorithmen mit vorgegebener Fehlerwahrscheinlichkeit koennen fuer eine gewisse NP-vollstaendige Sprache angegeben werden. d = 1 wuerde bedeuten, dass P = NP ist. Ich zeige, dass $d \geq 2/3$. Das bedeutet, dass nur O(1.5**n) Interpretationen konstruiert werden muessen, damit eine beliebig kleine Fehlerwahrscheinlichkeit erhalten wird (Abschnitte 2.2, 2.3).

9. Ich zeige fuer eine NP-vollstaendige Teilsprache Q von ERF, dass fuer jedes q in Q der Aufwand fuer eine Variante von SR von der Ordnung O(p(Laenge(q))*2**(2*n/3)) ist. Dabei ist n die Anzahl Variabeln in q und p ist ein Polynom kleinen Grades. Kuerzlich haben Tarjan und Trojanowski in [TAR76] einen O(2**(n/3)) Algorithmus fuer eine andere NP-vollstaendige Sprache angegeben.
Diese beiden Resultate zeigen, dass sogar fuer gewisse NP-vollstaendige Sprachen Algorithmen existieren, die im schlimmsten Fall wesentlich besser sind als die naheliegenden Aufzaehlungsalgorithmen (Abschnitt 1.14).

10. Unter Verwendung eines Resultates von Galil [GAL76] ueber Aufzaehlungsalgorithmen zeige ich in Abschnitt 1.13 exponentielle untere Schranken fuer eine spezielle Art von Superresolution.

11. Als Hauptresultat meiner Arbeit entwickle ich in Abschnitt 2.6 einen Entscheidungsalgorithmus fuer ERF. Er hat vermutlich polynomialen Aufwand, wobei aber die Entscheidung nur mit beliebig kleiner Fehlerwahrscheinlichkeit gefaellt wird. Erweist sich meine Vermutung als richtig, so ist fuer praktische Zwecke P=NP, d.h. dass jedes Problem in NP in polynomialer Zeit mit beliebig kleiner Fehlerwahrscheinlichkeit geloest werden kann. Es gibt bereits Resultate in dieser Richtung : Solovay und Strassen [SOL76] haben fuer die Primzahlen (die, wie auch ihr Komplement, in NP sind) sogar einen linearen Algorithmus angegeben, der die Entscheidung mit beliebig kleiner Fehlerwahrscheinlichkeit faellt.

Diss. ETH 5941

Autor: Karl Lieberherr

Titel: Informationsverdichtung von Modellen
in der Aussagenlogik und das P = NP - Problem

Referent: Prof. Dr. E. Engeler
Korreferent: Prof. Dr. E. Specker

Abstract:

The thesis deals with the problem of deciding, whether a given formula of the propositional calculus is satisfiable. In the first part a deduction rule, called Superresolution, is introduced and compared with known proof-systems and decision procedures. In the second part probabilistic algorithms for NP-complete languages are investigated.

Keywords:
Superresolution, decision procedure for Satisfiability, polynomial reduction of proof-systems, length of proofs, lower bounds, probabilistic algorithms for NP-complete languages.

Diss. ETH 5941

Autor: Karl Lieberherr

Titel: Informationsverdichtung von Modellen
in der Aussagenlogik und das P = NP - Problem

Referent: Prof. Dr. E. Engeler
Korreferent: Prof. Dr. E. Specker

Zusammenfassung

In dieser Arbeit wird die Komplexitaet des Entscheidens, ob eine konjunktive Normalform (KNF) der Aussagenlogik erfuellbar ist, analysiert. Dieses Entscheidungsproblem hat eine zentrale Stellung in einer grossen Klasse von kombinatorischen Entscheidungs- resp. Optimierungsproblemen.

Die bisher untersuchten Beweissysteme fuer die unerfuellbaren KNF haben lokalen Charakter. In der Arbeit wird ein Beweissystem Superresolution eingefuehrt, das als global bezeichnet werden darf, d.h. ein Schluss kann von der ganzen bisherigen Formel abhaengen. Superresolution wird verglichen mit einem auf dem Gebiete des automatischen Beweisens haeufig verwendeten Beweissystem, naemlich mit Resolution und deren Abarten. Weil Resolution lokal ist, so ist es einleuchtend, dass eine globale Schlussregel wie Superresolution kuerzere Beweise erlaubt als Resolution. Dies wird in der Arbeit auch bewiesen. Dafuer ist der Aufwand fuer das Finden, resp. Ueberpruefen eines Beweisschrittes groesser als bei Resolution. Beide Aufgaben sind jedoch linear loesbar. Superresolution ist eine grosse Einschraenkung von Resolution, denn keine Input-Resolvente kann eine Superresolvente sein. Deshalb hat Superresolution gegenueber Resolution wesentliche Vorzuege, die in der Arbeit untersucht werden.

Um die Komplexitaet von Superresolution zu analysieren, fuehrt man einen Entscheidungsalgorithmus SR fuer KNF ein, der nur Superresolutionsbeweise in einer kurzen Normalform erzeugen kann. Es stellt sich dabei heraus, dass Superresolution und Resolution polynomial aequivalent sind. Ausserdem wird fuer eine spezielle Art von Superresolution eine exponentielle untere Schranke bewiesen. Neben anderen Vorzuegen hat SR die Tendenz, "einfache" Formeln effizient zu behandeln.

Im zweiten Teil der Arbeit werden probabilistische Entscheidungsalgorithmen fuer NP-vollstaendige Sprachen untersucht. Probabilistische Algorithmen sind keine Algorithmen im eigentlichen Sinn, denn sie verwenden bei ihren Berechnungen Zufallsprozesse und koennen deshalb falsche Entscheide liefern.

Wenn die Fehlerwahrscheinlichkeit aber mit relativ wenig Rechenaufwand gegenueber den bekannten deterministischen Verfahren beliebig klein gemacht werden kann, sind probabilistische Algorithmen fuer praktische Anwendungen sehr interessant.

In der Arbeit wird der Begriff "Erfuelltheitsgrad" fuer KNF eingefuehrt und auf NP-vollstaendige Sprachen erweitert. Die Komplexitaet von probabilistischen Algorithmen fuer eine Sprache S haengt direkt mit dem Erfuelltheitsgrad von S zusammen. Die Relation wird durch eine Formel beschrieben, die ausdrueckt: Je groesser der Erfuelltheitsgrad ist, desto effizientere Algorithmen resultieren. Deshalb wird in der Arbeit versucht, den Erfuelltheitsgrad moeglichst gross zu machen. Es wird behauptet und durch verschiedene Ueberlegungen und Hilfssaetze begruendet, dass dies mit wenig Aufwand moeglich ist, beim Versuch, es zu beweisen, stoesst man aber auf erhebliche Schwierigkeiten.

Ø. Grundbegriffe

Die Komplexitaetstheorie beschaeftigt sich unter anderem mit der Frage : Welche Funktionen sind praktisch berechenbar ? Bevor wir diese Frage beantworten, benoetigen wir einige Definitionen.

Als Computermodell verwenden wir die Direktzugriffmaschine (DZM), wie sie in [AHO75,Seite 5] beschrieben ist. Dabei setzen wir voraus, dass die Ausfuehrung einer beliebigen Anweisung eine Zeiteinheit benoetigt.

Sei B* die Menge der endlichen Worte ueber dem Alphabet B={Ø,1}. (Jedes Element von B* ist als Codierung eines Objektes zu betrachten.) Eine Funktion f : B* -> B* heisst partiell berechenbar, wenn es ein Programm p fuer die DZM gibt, das fuer jedes x im Definitionsbereich von f nach endlich vielen Schritten stoppt, nachdem es f(x) auf das Ausgabeband geschrieben hat. Die Laenge l(x) eines Elementes x in B* ist die Anzahl Zeichen, die x enthaelt.

Sei f eine durch das Programm p berechnete Funktion. Wir betrachten die Menge c(n) saemtlicher Elemente x im Definitionsbereich von f, welche die feste Laenge l(x)=n haben. Sei s(x) die Zeit, die p benoetigt, um f(x) zu berechnen. Wir definieren

$$K[f,p](n) = \max[x \text{ in } c(n)]\ s(x)/l(f(x))$$

K[f,p] ist eine Funktion von N nach N, die wir die Zeitkomplexitaet von Programm p fuer f nennen.

Wir interessieren uns lediglich dafuer, mit welcher Groessenordnung die oben definierte Funktion waechst. Deshalb definieren wir : Eine Funktion T : N -> N ist von derselben Ordnung wie die Funktion S(n), wenn Konstanten c1,c2,n1 > Ø existieren, so dass fuer alle n > n1 :

$$c1 * S(n) \leq T(n) \leq c2 * S(n) .$$

Θ(S(n)) definieren wir als die Menge aller Funktionen, welche die Ordnung S(n) haben. Meistens wird als Repraesentant der Ordnung eine "allgemein bekannte" Funktion angegeben, z.B. eine Komposition von Polynomen, Logarithmusfunktionen oder Exponentialfunktionen. Die Ordnung der Zeitkomplexitaet von Programm p nennen wir die asymptotische Zeitkomplexitaet von p. Sie bestimmt unmittelbar die Groesse der Eingabewerte, die vom Programm p mit vernuenftigem Aufwand verarbeitet werden koennen. Allerdings stimmt das nur fuer Eingabewerte, die genuegend gross sind. Ein Programm, dessen Zeitkomplexitaet eine

hohe Wachstumsrate, aber einen kleinen konstanten Koeffizienten hat, ist naemlich fuer kleine Eingabewerte einem Programm ueberlegen, dessen Zeitkomplexitaet eine kleine Wachstumsrate, aber einen grossen konstanten Koeffizienten hat.

Heute werden allgemein die Funktionen, deren asymptotische Zeitkomplexitaet durch ein Polynom beschraenkt ist, als praktisch berechenbar betrachtet. Wir nennen diese Funktionen polynomial berechenbar und bezeichnen die Klasse aller polynomial berechenbaren Funktionen mit P. Sicher sind alle "einfachen" Funktionen in P enthalten. Zudem ist P invariant bei gewissen [PR74] Veraenderungen des Maschinenmodells. Wenn man z.B. die klassische Turingmaschine statt der DZM verwendet (1 Schritt = 1 Zeiteinheit), erhaelt man dieselbe Klasse P.

Im folgenden interessieren wir uns fuer Entscheidungsfunktionen, die auf Worten des Alphabets B={Ø,1} definiert sind. Jede Teilmenge L von B* nennen wir eine Sprache. Wir sagen, L ist in P, wenn die charakteristische Funktion von L in P ist. Meistens sind die Sprachen, die wir betrachten, gewisse Formelmengen. Die Details der Codierung in (Ø,1)-Folgen geben wir nicht an.

Wir definieren nun die Klasse NP [CO71]. Die Sprachen in NP erscheinen in vielen Gebieten der Mathematik und sind deshalb besonders interessant (siehe Teil C der Bibliographie). Eine Sprache L ist in NP, wenn es ein nicht-deterministisches Programm p fuer eine DZM mit folgenden Eigenschaften gibt (bei einer nicht-deterministischen Anweisung darf es nur endlich viele Auswahlmoeglichkeiten geben) :

1. p akzeptiert nur Elemente in L, unabhaengig von der Wahl in den nicht-deterministischen Anweisungen.
2. Fuer jedes x in L gibt es eine akzeptierende Berechnung von polynomialer Laenge bezueglich der Laenge von x.

Die folgende Sprache ERFUELLBARKEIT (ERF) ist in NP:
Elemente : Mengen von Klauseln s = {c(1), c(2), ... c(m)} in Variabeln x(1), x(2), ... x(n). Jede Klausel ist eine Menge von Literalen, wobei jeder Literal entweder die Variable x(i) ($1 \leq i \leq n$) oder deren Negation x(i)´ ist.
Eigenschaft : Es gibt eine Zuweisung von Wahrheitswerten an die Variabeln, so dass alle Klauseln erfuellt sind. Dabei heisst eine Klausel erfuellt, wenn mindestens einer ihrer Literale den Wahrheitswert wahr hat.

ERF kann auch anders definiert werden : Eine Formel G der Aussagenlogik heisst eine konjunktive Normalform (KNF), gdw. G die Form F(1) & F(2) & ... & F(m) hat, wobei jedes F(i) ($1 \leq i \leq m$) eine Disjunktion (Oder-Operation) von Literalen ist.

Sei G eine Formel der Aussagenlogik, und seien x(1),x(2), ... x(n) die Variabeln, welche in G vorkommen. Eine Interpretation

von G ist eine Zuweisung von Wahrheitswerten an x(1), x(2), ... x(n). Eine Interpretation kann beschrieben werden durch eine Menge I von Literalen, welche fuer jede Variable von G genau einen Literal enthaelt. Wenn eine Variable x in I positiv vorkommt (d.h. x ist in I), dann hat x unter I den Wahrheitswert "wahr". Wenn x negativ in I vorkommt (d.h. x´ ist in I), dann hat x den Wahrheitswert "falsch". Falls eine Formel G wahr ist unter einer Interpretation I, so sagen wir, dass I ein Modell von G ist. Eine Formel, die ein Modell hat, heisst erfuellbar. Also entspricht der Sprache ERF die Menge der erfuellbaren KNF. Eine beliebige aussagenlogische Formel kann durch lineare Vergroesserung in eine aequivalente KNF uebersetzt werden (siehe z.B. [TS68]).

Ein wichtiges Problem ist die Frage, ob P=NP. Vieles weist darauf hin, dass diese Mengen verschieden sind. Davon sind auch die meisten Mathematiker ueberzeugt. Trotzdem bin ich aufgrund der Resultate dieser Arbeit noch nicht ganz ueberzeugt von der Vermutung P≠NP.

Die Frage, ob P=NP, laesst sich reduzieren auf die Frage, ob die Sprache ERF in P ist [CO71]. Seien L und M Sprachen. Dann heisst L polynomial reduzierbar auf M (L<M), wenn es eine Funktion f in P von B* nach B* gibt, so dass f(x) in M ist, gdw. x in L ist. Wenn L<M und M in P ist, so ist auch L in P. Diese Reduzierbarkeit nennt man mehrdeutige Reduzierbarkeit.

Eine Sprache L heisst in polynomialer Zeit Turing-reduzierbar zu einer Sprache M, wenn es eine Orakel-Turingmaschine TM und ein Polynom p gibt, so dass x in L ist, gdw. TM das Element x mit M als Orakel in p(Laenge(x)) Schritten akzeptiert. Interessante Eigenschaften dieser und anderer polynomialer Reduzierbarkeiten finden sich in [LA74,LA75] (siehe auch Bibliographie Teil F).

Sei D eine Menge von Sprachen. Eine Sprache ds in D heisst D-schwierig, wenn fuer alle Sprachen d in D gilt : d<ds. Fuer eine Sprache ds, die D-schwierig ist, gilt : Wenn ds in P ist, so ist die ganze Menge D in P. Wenn hingegen fuer eine D-schwierige Sprache ds bewiesen ist, dass sie nicht in P ist, so gilt dies nicht fuer ganz D. Deshalb brauchen wir folgende Definition : Eine Sprache dv in D heisst D-vollstaendig, wenn dv D-schwierig ist und es eine Sprache d in D gibt mit dv<d. Wenn eine D-vollstaendige Sprache nicht in P ist, so ist P verschieden von D. Denn, wenn jede Sprache d in D in P waere, so waere auch jede D-vollstaendige Sprache in P.

Sei nun D=NP. Cook [CO71] hat bewiesen, dass L in NP, gdw. L < ERF. Deshalb ist ERF NP-schwierig. ERF ist auch NP-vollstaendig, weil ERF in NP liegt. Mit ERF(k) bezeichnen wir die Teilsprache von ERF, fuer die gilt: Jede Klausel einer KNF in ERF(k) hat hoechstens Laenge k, d.h. sie enthaelt hoechstens k Literale. ERF(k) ist NP-vollstaendig fuer k>2. Hingegen ist

ERF(2) in P.

Wenn P=NP, so ist NP abgeschlossen unter Komplementbildung, denn P ist abgeschlossen unter Komplementbildung. Die Frage, ob NP unter Komplementbildung abgeschlossen ist, steht in direktem Zusammenhang mit der Untersuchung von Beweissystemen [CO74]. Sei L eine Sprache ueber B={0,1}. Ein Beweissystem fuer L ist ein nicht-deterministischer Algorithmus BW(L) mit Eingabewerten x in B*. Falls es fuer ein x eine Berechnung w gibt, die x akzeptiert, so ist w ein Beweis, dass x in L ist. Wir setzen voraus, dass BW(L) korrekt ist, d.h. nur Elemente x in L akzeptiert. Weiter verlangen wir, dass es deterministisch in polynomialer Zeit bezueglich der Laenge von w moeglich ist, zu ueberpruefen, ob eine Berechnung w entsprechend den Vorschriften des Algorithmus BW(L) ausgefuehrt wurde. Beweissysteme koennen auch aufgefasst werden als Schlussregelsysteme.

Sei L eine Sprache und BW(L) ein Beweissystem fuer L. Sei c(n) die Menge saemtlicher Elemente x in L, welche feste Laenge n haben. s(x) sei die Laenge des kuerzesten Beweises fuer x im Beweissystem BW[L]. Wir nennen

$$K[BW(L)]\ (n) = \max\ [x \text{ in } c(n)]\ s(x)$$

die Komplexitaet des Beweissystems BW(L). Die asymptotische Komplexitaet eines Beweissystems definieren wir wie bei Programmen.

Ein Beweissystem BW(L) heisst polynomial, wenn die Laenge des kuerzesten Beweises fuer alle x in L beschraenkt ist durch ein Polynom in der Laenge von x, d.h. wenn die asymptotische Komplexitaet durch ein Polynom beschraenkt ist. Wenn eine Sprache ein polynomiales Beweissystem hat, so ist sie in NP.

Sei UNERF die Sprache der unerfuellbaren KNF. Dann gilt : NP ist abgeschlossen unter Komplementbildung, gdw. UNERF ein polynomiales Beweissystem hat [CO74].

Ein bekanntes Beweissystem fuer UNERF ist Resolution. Zwei Klauseln c1 und c2 stossen zusammen, wenn es genau einen Literal in c1 gibt, der in c2 komplementiert vorkommt. Falls die Klauseln d1 = c1 + {x} und d2 = c2 + {x'} zusammenstossen, dann ist r = c1 + c2 die Resolvente von d1 und d2. Wir sagen, r wurde erhalten durch Anwendung von Resolution. d1 und d2 heissen Elternklauseln, x heisst aufgeloeste Variable.

Ein Resolutionsbeweis fuer die Unerfuellbarkeit einer KNF s ist eine Folge von Klauseln c(1), c(2), ... c(k), so dass c(k) die leere Klausel ist. Fuer $1 \leq i \leq k-1$ muss gelten, dass c(i+1) eine Resolvente von Klauseln in s + {c(1), c(2), ... c(i)} ist. Man kann zeigen, dass eine KNF s unerfuellbar ist, gdw. ein Resolutionsbeweis existiert. Solche Beweissysteme nennt man vollstaendig. Regulaere Resolution, eine einschraen-

kende Art von Resolution, ist auch vollstaendig. In [TS68, GA74] wird gezeigt, dass dieses Beweissystem nicht polynomial ist. Fuer Resolution wird ebenfalls vermutet, dass sie nicht polynomial ist [COO76].

Ein Beweissystem BW1(L1) fuer L1 heisst polynomial reduzierbar auf ein Beweissystem BW2(L2) fuer L2 (BW1(L1) < BW2(L2)), wenn
1. L1 < L2 (Die Uebersetzungsfunktion sei f.)
2. fuer jeden Beweis w1(x1) fuer ein x1 in L1 ein Beweis w2(f(x1)) fuer f(x1) in L2 existiert, so dass die Laenge von w2(f(x1)) beschraenkt ist durch ein Polynom in der Laenge von w1(x1).

Zwei Beweissysteme BW1(L1) und BW(L2) heissen aequivalent, wenn BW1(L1) < BW2(L2) und BW2(L2) < BW1(L1) .

In [CO74] werden verschiedene Beweissysteme fuer UNERF in bezug auf ihre gegenseitige Reduzierbarkeit untersucht. Als ein starkes Beweissystem erweist sich erweiterte Resolution.

Bei der erweiterten Resolution koennen Hilfsvariabeln eingefuehrt werden. Sei s eine KNF. Fuer zwei Literale x,y in s und eine neue Hilfsvariable h sei

$$A(h,x,y) = \{(x',h) , (y',h) , (x,y,h')\} .$$

Man beachte, dass die Menge der Klauseln A(h,x,y) aequivalent ist zu h = x v y . Die Erweiterungsregel ist folgendermassen definiert:
1. Waehle x und y in s.
2. Erweitere s um die Klauseln A(h,x,y).

Auch von diesem Beweissystem wird in [CO75] vermutet, dass es nicht polynomial ist.

Als Sprache zur Beschreibung von Algorithmen verwende ich Pidgin-Algol, wie es in [AHO75] beschrieben ist. Kommentare werden zwischen (* und *) eingeschlossen, z.B. : Sei s eine KNF und l ein Literal in s. Die Prozedur REDUZIERE(s,l) leiste folgendes : Alle Klauseln, die l enthalten, werden gestrichen. In allen Klauseln, welche l' enthalten, wird l' gestrichen.

Beschreibung in Pidgin-Algol:

```
procedure REDUZIERE(s,l);
(* s ist eine KNF und l ein Literal von s *)
begin
   for jede Klausel c, die l enthaelt do
   entferne c aus s;
   for jede Klausel c, die l' enthaelt do
   entferne den Literal l' aus c
end
```

Wenn ein Parameter in einem Prozeduraufruf irrelevant ist, wird er durch * ersetzt.

1. Superresolution

Superresolution ist eine effiziente Konkretisierung der in [RO68] eingefuehrten verallgemeinerten Resolution. Sie ordnet jeder misslungenen Modellkonstruktion eine verallgemeinerte Resolvente zu. Superresolution ist nicht nur ein weiteres der zahlreichen vollstaendigen Resolutionsbeweissysteme, sondern in verschiedener Hinsicht beweisbar besser als Resolution und deren Abarten.

Auf dem Gebiete des automatischen Beweisens werden verschiedene einschraenkende Varianten von Resolution untersucht, die aber immer noch vollstaendig sind (Siehe Bibliographie Teil A). Diese Untersuchungen werden durch folgende Erfahrung motiviert: Automatische Beweisprogramme haben die unangenehme Eigenschaft, enorm viele Folgerungen zu ziehen, die fuer den gesuchten Beweis irrelevant sind (siehe z.B. [MAR75]). Wenn man eingeschraenkte Schlussregeln verwendet, ist diese Gefahr vermindert. Allerdings verlaengern solche eingeschraenkten Schlussregeln meistens die Beweise [KI72, ME71, SHO76]. Es ist intuitiv einleuchtend, dass mit eingeschraenkten Schlussregeln im allgemeinen laengere Beweise entstehen. Fuer Superresolution kann ich hingegen beweisen:

1. Superresolution ist eine starke Einschraenkung von Resolution (siehe Abschnitt 1.5) und
2. Superresolutionsbeweise sind kuerzer als Resolutionsbeweise (siehe Abschnitt 1.6).

Deshalb eignet sich Superresolution (mehr als alle anderen bekannten Abarten von Resolution) vortrefflich als Schlussregel in einem automatischen Beweisprogramm.

1.1 Das vollstaendige Beweissystem Superresolution

Zur Definition des Begriffs Superresolvente benoetigen wir die im folgenden beschriebene Prozedur SETZE.

SETZE findet fuer eine KNF s und einen Literal l diejenigen Literale, die wahr gesetzt werden muessen, wenn l wahr gesetzt wird. Ein Literal ll heisst durch l eindeutig bestimmt, falls eine unerfuellte Klausel entsteht, wenn l wahr und ll falsch gesetzt wird. Die Boolesche Variable WIDERSPRUCH wird wahr, wenn die eindeutig bestimmten Literale eine Klausel von s unerfuellt lassen.

```
procedure SETZE(l,s,WIDERSPRUCH);
begin
   el := {l} ; WIDERSPRUCH := false ;
   repeat
      waehle ein Element l1 aus el ;
      el := el - {l1} ;
      seien kpl1 die Klauseln in s,
      welche den Literal l1 enthalten ;
      for jede Klausel c in kpl1 do entferne c aus s;
      seien knl1 die Klauseln in s, welche den Literal l1´
      enthalten;
      for jede KLausel c in knl1 do
      begin
         streiche den Literal l1´ in der KLausel c von s;
         if c enthaelt genau einen Literal l2 then
         (* l2 heisst eindeutig bestimmt *)
         if l2´ in el then WIDERSPRUCH := true else
         el := el + {l2}
      end
   until el={} ;
end;
```

Informell ist eine Superresolvente sr einer KNF s eine Klausel, fuer die folgendes gilt: Seien alle Literale in sr mit Hilfe von SETZE so gesetzt, dass sr unerfuellt ist. Die Variablen, die zu anderen Literalen als denen in sr gehoeren, seien noch nicht interpretiert. Dann darf keine Klausel in s unerfuellt sein. Jetzt muss es eine Variable v geben, deren beide Interpretationen durch Konsequenzenbildung (d.h. durch Beruecksichtigen der eindeutig bestimmten Literale) eine unerfuellte Klausel entstehen lassen. Die Variable v nennen wir lernend.

Nun geben wir die exakte Definition fuer den allgemeineren Begriff Superresolvente k-ten Grades.

Definition 1.1.1 : Eine Klausel {l(1), l(2), ... l(n)} heisst Superresolvente k-ten Grades (k>1) einer KNF s, wenn man SETZE(l(i),s,w) der Reihe nach fuer i = 1, 2, ... n ausfuehren kann und immer w=false erhaelt, und wenn dann

1. k Variablen v(1), v(2), ... v(k) existieren, so dass fuer alle Folgen vv(1), vv(2), ... vv(k) (vv(j) in {v(j),v(j)´} (1<j<k)) die Prozedur SETZE(vv(i),s,w), der Reihe nach fuer i = 1, 2, ... k ausgefuehrt, immer w=true liefert und
2. fuer jedes k1<k eine Folge vv(1), vv(2), ... vv(k1) existiert, so dass die Prozedur SETZE(vv(i),s,w), der Reihe nach fuer i = 1, 2, ... k1 ausgefuehrt, w=false liefert.

Eine Klausel {l(1), l(2), ... l(n)} heisst Superresolvente einer KNF s, wenn man SETZE(l(i),s,w) der Reihe

nach fuer i = 1, 2, ... n ausfuehren kann und immer w=false erhaelt, und wenn dann eine Variable v existiert, fuer welche sowohl bei SETZE(v,s,w) als auch bei SETZE(v´,s,w) die boolesche Variable w=true wird.

Bemerkungen:
1. Man beachte, dass eine Superresolvente ersten Grades eine Superresolvente ist.
2. Bei der Konstruktion einer Superresolventen sr einer KNF s werden viele Klauseln von s, manchmal sogar alle (wie im folgenden Beispiel), beruecksichtigt. Eine Superresolvente ist also ein "Kondensat" von vielen Klauseln, eine Resolvente hingegen nur von 2.
3. Fuer eine Resolvente ist es einfach zu ueberpruefen, dass sie bei jedem existierenden Modell erfuellt ist. Man hat dazu nur die beiden Elternklauseln zu finden. Eine Superresolvente ist auch bei jedem existierenden Modell erfuellt, d.h. eine Superresolvente ist wie eine Resolvente eine Folgerung. Diese Eigenschaft ist zwar verborgen, doch durch iterierte Anwendung von SETZE einzusehen.

Beispiel:
1. a b
2. c d
3. e f
4. a´ c´
5. a´ e´
6. c´ e´
7. b´ d´
8. b´ f´
9. d´ f´

Die leere Klausel ist eine Superresolvente dieser KNF, da jede Variable lernend ist.

Fuer diejenigen Leser, denen die Programmiersprachterminologie nicht vertraut ist, folgt nun eine Definition des Begriffs Superresolvente in der ueblichen mathematischen Terminologie. Sei s eine beliebige aussagenlogische Formel und LIT(s) die Menge der Literale in s. Eine Teilinterpretation von s ist eine Teilmenge einer Interpretation aller Variablen von s. Ein Literal l in LIT(s) heisst durch eine Teilinterpretation I einfach bestimmt, wenn s unter der Interpretation I+{l´} unerfuellt ist oder wenn l in I ist (wir schreiben (s,I)->l). Eine Formel s heisst unerfuellt unter der partiellen Interpretation I, wenn nach der Elimination der Konstanten, die durch I in s eingefuehrt werden, "falsch" erhalten wird. Ein Literal l in LIT(s) heisst durch eine Teilinterpretation I bestimmt in der Formel s, wenn es Literale g(1), g(2), ... g(n)=l (n>0) in LIT(s) gibt, so dass (s, I+{g(1), g(2), ... g(i-1)})->g(i) fuer 1<i<n (wir schreiben (s,I)=>l).
Sei D(s,I) die Menge aller Literale von s, die durch I be-

stimmt sind, d.h.

$$D(s,I) = \{l \text{ in } LIT(s) : (s,I) \Rightarrow l\}.$$

D(s,I) kann komplementaere Paare von Literalen enthalten, d.h. eine Variable v und ihr Komplement v´.

Definition 1.1.1A : Sei s eine aussagenlogische Formel und c eine Klausel mit Literalen in s. Sei I die Menge der komplementierten Literale in c. c heisst eine Superresolvente von s, wenn
1. D(s,I) kein komplementaeres Literalpaar enthaelt und wenn
2. eine Variable v existiert, so dass sowohl D(s,I+{v}) als auch D(s,I+{v´}) ein komplementiertes Literalpaar enthaelt.

Lemma 1.1.1 : Sei s eine erfuellbare KNF und M ein Modell von s. Dann ist jede Superresolvente beliebigen Grades von M erfuellt.

Beweis:
Sei sr eine Superresolvente k-ten Grades von s. Annahme: sr sei unerfuellt beim Modell M. Wegen der Definition 1.1.1 gibt es k Variablen v(1),v(2), ... ,v(k) in s - sr, so dass bei allen Interpretationen dieser Variablen mit Hilfe von SETZE eine unerfuellte Klausel in s entsteht. Also kann eine Interpretation, welche sr nicht erfuellt, sicher nicht zu einem Modell erweitert werden. Widerspruch. #

Definition 1.1.2 : Ein Superresolutionsbeweis k-ten Grades fuer eine KNF s ist eine Folge von Klauseln c(1),c(2), ... c(m), so dass c(m)={} und fuer i zwischen 1 und m-1 die Klausel c(i+1) eine Superresolvente hoechstens k-ten Grades der KNF s + {c(1), c(2), ... c(i)} ist.
Einen Superresolutionsbeweis ersten Grades nennen wir einen Superresolutionsbeweis.

Bemerkung:
Automatischer Gleichheits- und Teilklauseltest: Man beachte, dass gemaess Definition des Begriffs "Superresolvente" nicht zweimal dieselbe Superresolvente oder eine Superresolvente, die eine frueher erzeugte enthaelt, in einem Superresolutionsbeweis vorkommen kann. Dies ist bei Resolutionsbeweisen nicht garantiert, weshalb ein aufwendiger Test noetig ist.

Lemma 1.1.2 : Fuer jedes k gilt : Eine KNF s ist unerfuell-

bar genau dann, wenn ein Superresolutionsbeweis k-ten Grades existiert.

Beweis:
Wenn ein Superresolutionsbeweis k-ten Grades fuer s existiert, so ist s nach Lemma 1.1.1 unerfuellbar. Die Umkehrung wird aus dem folgenden klar. #

Bemerkung :
Fuer eine KNF mit n Variablen ist Superresolution log(n)-ten Grades immer noch ein Beweissystem, d.h. das Ueberpruefen des Beweises ist polynomial moeglich. k braucht also nicht konstant zu sein. Superresolution ersten Grades und Superresolution log(n)-ten Grades sind trotzdem polynomial aequivalente Beweissysteme, denn jede Superresolvente log(n)-ten Grades kann durch Superresolventen linear simuliert werden.

Im naechsten Kapitel beschreiben wir einen universellen Algorithmus SR, der jede interessante Superresolvente erzeugen kann. Die Existenz und die Eigenschaften dieses Algorithmus werden uns behilflich sein, verschiedene Einsichten in Superresolution zu gewinnen, insbesondere wie man "kurze" Superresolventen finden kann. Eine wichtige Eigenschaft von SR ist, dass sich ein beliebiger Superresolutionsbeweis auf einen von SR erzeugten verkuerzen laesst. Deshalb haben die von SR erzeugten Superresolutionsbeweise eine zentrale Stellung. Algorithmus SR wird in dieser Arbeit haeufig verwendet, und deshalb ist eine genaue Kenntnis von ihm unerlaesslich fuer das Verstaendnis der spaeteren Abschnitte.

1.2 Beschreibung von Algorithmus SR

SR ist ein Algorithmus mit nicht-deterministischen Anweisungen, der auf KNF operiert. In jeder nicht-deterministischen Anweisung hat er eine endliche Auswahl von Alternativen. Wenn nicht ausdruecklich erwaehnt, gelten die Aussagen ueber SR fuer jede beliebige Wahl in den nicht-deterministischen Anweisungen. SR kann deshalb als das erzeugende Element einer ganzen Algorithmenklasse aufgefasst werden.

Auf erfuellbaren KNF berechnet SR ein Modell und auf unerfuellbaren KNF einen Superresolutionsbeweis, unabhaengig von der Variante, welche bei nicht-deterministischen Anweisungen gewaehlt wird. Diese Wahl hat jedoch bei erfuellbaren KNF einen wesentlichen Einfluss auf die Berechnungszeit von SR, weil die Interpretation von Variabeln im wesentlichen auch ueber nicht-deterministische Anweisungen geschieht. Waehlt SR naemlich ein Modell, so terminiert er sofort.

Sei s eine Eingabe-KNF. SR versucht fuer s ein Modell zu konstruieren, indem SR nach bestimmten Regeln eine Interpretation konstruiert. Einer Interpretationskonstruktion von SR entspricht ein Schritt von SR. Falls SR im laufenden Schritt kein Modell findet, wird eine Superresolvente r in s aufgenommen ("gelernt"). r hat die Aufgabe, zu verhindern, dass bei einer spaeteren Interpretationskonstruktion nochmals der "Fehler" (nur bei erfuellbaren KNF ist es ein eigentlicher Fehler), der beim letzten Schritt begangen wurde, wiederholt wird. Falls r leer ist, hat man Unerfuellbarkeit bewiesen. Sonst wird ein weiterer Schritt ausgefuehrt. Fuer die Konstruktion von r verwendet SR eine Menge von Baeumen, die bei der Interpretationskonstruktion aufgebaut werden.

Ist I eine von SR bereits konstruierte Teilinterpretation von s, die erweitert wird, und l ein Literal, der als naechster in I aufgenommen werden koennte, dann muss genau einer der drei Faelle eintreten:

1. Fall "waehle"
   Kein Literal im noch vorhandenen s ist durch die Anwendung von SETZE eindeutig bestimmt. Auch l´ koennte anstelle von l in I aufgenommen werden, ohne dass durch SETZE ein Widerspruch entsteht. Wir nennen die Variable, die zu l gehoert, waehlbar.
2. Fall "unerfuellt"
   Sowohl wenn l, als auch wenn l´ in I aufgenommen wuerde, entstuende ein Widerspruch. Deshalb kann I nicht zu einem Modell erweitert werden. Es wird eine Superresolvente sr gebildet, die eine Teilmenge der komplementierten Literale von I ist. Die Variable, die zu l gehoert, ist die lernende Variable der Superresolventen sr.
3. Fall "isoliert"
   Wenn l´ in I aufgenommen wuerde, so wuerde ein Widerspruch entstehen. Wir nennen die Variable, die zu l gehoert, isoliert.

Zuerst beschreibe ich Algorithmus SR in der ueblichen mathematischen Terminologie. Anschliessend folgt eine Beschreibung in Pidgin-Algol [AHO75].

Um das Wesentliche herausheben zu koennen, beschreibe ich zuerst einen etwas einfacheren Algorithmus SR´. Mit Hilfe von SR´ kann SR einfach erklaert werden.

Sei s die Eingabe-KNF. In einem Schritt von SR´ wird nach folgenden Regeln eine Interpretation konstruiert und, wenn kein Modell gefunden wird, eine Superresolvente gebildet:

1. Die folgenden Operationen werden auf einer Kopie von s durchgefuehrt. Das momentane s wird nach diesem Schritt wieder verwendet. Es wird eine beliebige Variable v von s gewaehlt und festgestellt, welche der Eigenschaften "waehlbar", "lernend" und "isoliert" gilt. (Es muss genau eine gelten.) Meistens wird v waehlbar sein.

a) waehlbar
   Es wird ein Literal l von v gewaehlt. Dann wird l in die Menge GEWAEHLT aufgenommen.
b) isoliert
   Sei l ein Literal von v, so dass kein Widerspruch entsteht, wenn l wahr gesetzt wird.
c) lernend
   Es wird ein Literal l von v gewaehlt und eine Superresolvente sr gebildet, die in eine Menge sres aufgenommen wird. sr enthaelt gewisse komplementierte Literale der momentanen Menge GEWAEHLT. Diese Teilmenge garantiert bereits, dass durch Konsequenzenbildung nachgeprueft werden kann, dass die entsprechende Teilinterpretation (die komplementierten Literale der Teilmenge) nicht zu einem Modell erweitert werden kann.

   Man beachte, dass die Menge G der komplementierten Literale von GEWAEHLT auch eine Superresolvente ist. Die Eigenschaften von sr sind in Abschnitt 1.3 etwas genauer analysiert.

2. Der Literal l wird nun mit SETZE wahr gesetzt, d.h. l und die durch l eindeutig bestimmten Literale werden wahr gesetzt und die betroffenen Klauseln entweder gestrichen oder gekuerzt. Dabei wird die Geschichte der Konsequenzenbildung festgehalten, indem fuer jeden Literal l1, der durch l eindeutig bestimmt ist, die Menge der durch l1 eindeutig bestimmten Literale gebildet wird. Diese Mengen werden bei der Bestimmung der Superresolventen verwendet.

3. Falls s noch nicht leer ist, werden die Punkte 1. und 2. wiederholt, bis s leer ist.

4. s wird fuer den naechsten Schritt vorbereitet, falls kein Modell oder die leere Superresolvente erhalten wurde: s erhaelt wieder den Wert vor dem letzten Schritt, ergaenzt um eine aus sres gewaehlte Superresolvente. Anschliessend werden allenfalls Vereinfachungen durchgefuehrt, z.B. Klauseln der Laenge 1 eliminiert.

Im naechsten Schritt wird analog mit der erweiterten KNF eine Interpretationskonstruktion durchgefuehrt. Bei zwei verschiedenen Schritten kann nicht dieselbe Superresolvente gebildet werden. Deshalb bricht der Algorithmus nach endlich vielen Schritten ab, weil nur endlich viele verschiedene Superresolventen existieren. Ueber die Anzahl der Superresolventen kann ich nur relativ zur Anzahl der Resolventen etwas Interessantes aussagen (Siehe Abschnitt 1.5). Es gibt wesentlich weniger Superresolventen als Resolventen.

Soweit die Beschreibung von SR'. Algorithmus SR unterscheidet sich folgendermassen von SR': Bevor eine Variable v gewaehlt wird, evaluiert SR fuer jede Variable von s, ob sie waehlbar,

lernend oder isoliert ist. Wenn es eine lernende resp. isolierte Variable gibt, wird eine solche gewaehlt und sonst eine waehlbare. Ausser dieser Wahlvorschrift ist SR genau gleich wie SR´.

Nun folgt die Beschreibung in Pidgin-Algol. Der Algorithmus SR benuetzt vier separat definierte Prozeduren, die auf den folgenden Seiten beschrieben sind. Danach wird die Arbeitsweise anhand eines Beispiels erlaeutert.

Globale Daten von SR:

GEWAEHLT, I1: sind Mengen von Literalen, die als Interpretationen der Variabeln in s aufzufassen sind. GEWAEHLT ist eine Teilmenge von I1.

I,I1: sind Array´s von Literalmengen, die als Interpretationen der Variabeln in s aufzufassen sind.

LERNEND: ist ein array von Booleschen Variabeln. Die Indizes sind die Variabeln in s. LERNEND[v] wird wahr, wenn die Interpretation I1 sich nicht zu einem Modell erweitern laesst.

s: ist eine KNF, zu Beginn die Eingabe-KNF.

SCHRITT: zaehlt, wieviele Interpretationskonstruktionen durchgefuehrt werden.

UEBRIG: ist eine KNF, die aus s entsteht durch Entfernen von Klauseln und Markieren von Literalen.

UNERFUELLBAR: Diese Boolesche Variable wird wahr, wenn die Eingabe-KNF unerfuellbar ist.

ISOLIERT: ist ein array von Booleschen Variabeln. Die Indizes sind die Variabeln in s. ISOLIERT[v] wird wahr, wenn die Interpretation von v eindeutig bestimmt ist mit Hilfe der Prozedur SETZE.

WALD: ist ein array von Baeumen, in dem die Information ueber gewisse Klauseln, die aus UEBRIG gestrichen wurden, gespeichert ist. Die Punkte eines Baumes b sind Literale. Den Kanten von b ist als zusaetzliche Information eine Menge von Literalen zugeordnet. Die Indizes von WALD sind die Literale von s. Die Wurzel von WALD[l] ist der Literal l.

wvar: ist ein array von Variablenmengen. Indizes sind die Literale in s.

Die Prozedur VEREINFACHE(s)

Globale Variable : UNERFUELLBAR

VEREINFACHE fuehrt triviale Verkuerzungen der KNF s durch, naemlich:

1. Elimination von Klauseln der Laenge 1.
2. Anwendung von Subsumption (wenn eine Klausel d in einer Klausel c enthalten ist, so wird nur d behalten).
3. Wenn eine Variable v nur positiv vorkommt in s (nur v kommt in s vor), so koennen alle Klauseln, die v enthalten, gestrichen werden. Analoges gilt, wenn v nur negativ vorkommt.
4. Wenn l1=l2 gilt, wird l1 ueberall durch l2 ersetzt.

Beschreibung in Pidgin-Algol:

```
procedure VEREINFACHE(s);
   begin
      AENDERUNG := false ;
      repeat
         while s enthaelt einen Literal l in einer Klausel der
         Laenge 1 do
         begin
            REDUZIERE(s,l) ;
            UNERFUELLBAR wird wahr, wenn eine Klausel der
            Laenge Ø entstanden ist ;
            AENDERUNG := true ;
         end;
         while s enthaelt eine Klausel c, die eine andere
         Klausel enthaelt do
         begin
            eliminiere c in s ;
            AENDERUNG := true ;
         end;
         while s enthaelt einen Literal l, ohne dass l' in s
         vorkommt do
         begin
            REDUZIERE(s,l) ;
            AENDERUNG := true ;
         end;
         while es gibt Klauselpaare {l1,l2'},{l1',l2} in s do
         begin
            ersetze in allen Klauseln von s den Literal l1
            durch l2 ;
            AENDERUNG := true ;
         end;
      until not AENDERUNG ;
   end;
```

Bemerkung : VEREINFACHE laesst sich geschickter formulieren, weil nicht jede while-Schleife die Bedingungen in allen anderen while-Anweisungen veraendert.

Die Prozedur SETZEB(l,UEBRIG,WIDERSPRUCH)

Globale Variablen: I[l], WALD[l], wvar[l]

Im Unterschied zu SETZE speichert SETZEB seine Geschichte in der Datenstruktur WALD. Diese Information wird spaeter dazu verwendet, die Superresolvente zu konstruieren.
Der Literal l wird wahr gesetzt, und alle dadurch eindeutig bestimmten Literale werden auch wahr gesetzt und in I[l] gespeichert. Wenn ein eindeutig bestimmter Literal gefunden wird, erweitert SETZEB den Baum WALD[l] durch eine Kante. Als Label dieser Kante werden Literale derjenigen Klausel eingetragen, welche den eindeutig bestimmten Literal verursachte. Die Variable WIDERSPRUCH wird wahr, wenn beim Setzen der eindeutig bestimmten Literale eine unerfuellte Klausel entsteht. Wenn WIDERSPRUCH wahr ist, so ist wvar[l] die Menge von Variabeln, die sowohl wahr wie auch falsch bestimmt sind.

Beschreibung in Pidgin-Algol:

```
procedure SETZEB(l,UEBRIG,WIDERSPRUCH);
   begin
      el := {l} ;
      WIDERSPRUCH := false; wvar[l] := {};
      initialisiere WALD[l] durch seine Wurzel l;
      repeat
         (* Nicht-deterministische Wahl [el] *)
         sei l1 ein Element aus el; el := el - {l1} ;
         I[l] := I[l] + {l1} ; seien kpl1 die Klauseln in UEB-
         RIG, welche den Literal l1 enthalten;
         for jede Klausel c in kpl1 do entferne c aus UEBRIG;
         seien knl1 die Klauseln in UEBRIG, welche den Literal
         l1´ enthalten ;
         for jede Klausel c in knl1 do
         begin
            markiere den Literal l1´ in der Klausel c von UEB-
            RIG ;
            if c enthaelt genau einen Literal l2, der nicht
            markiert ist then
            begin
               waehle einen Punkt l1 in WALD[l], dem der Lite-
               ral l1 zugeordnet ist;
               erweitere WALD[l] durch eine Kante ka von l1
               nach dem neuen Punkt l2;
               label[ka] := c - {l1} - {l2} ;
               if l2´ in el then
               begin
                  wvar[l] := wvar[l] + { Variable, die zu Li-
                  teral l2 gehoert } ;
                  WIDERSPRUCH := true ;
               end else el := el + {l2};
            end;
         end;
      until el={} ;
   end;
```

Die Prozedur LABELSAMMELN(l,vp,vn,w)

Globale Variable: WALD[l]

Nach der Ausfuehrung enthaelt die Menge w die Literale, welche als Label der Kanten auf dem Weg von der Wurzel von WALD[l] nach den Literalen vp resp. vn auftreten.

Beschreibung in Pidgin-Algol

```
procedure LABELSAMMELN(l,vp,vn,w);
   begin
      verwende WALD[l] und bestimme die Menge w der Labels auf
```

```
      den Wegen von l nach vp und nach vn ;
   end;
```

Die Prozedur AUFROLLEN(w)

Globale Variabeln: GEWAEHLT, WALD

Vor der Ausfuehrung von AUFROLLEN enthaelt die Menge w eventuell einen Literal ng, der durch andere Literale eindeutig bestimmt wurde, d.h. es gibt einen Literal g, so dass WALD[g] den Literal ng´ an einem Punkt vng enthaelt. ng´ ist eindeutig bestimmt, weil g in Il und die komplementierten Literale der Label der Kanten auf dem Weg von g nach ng´ in Il sind. Deshalb wird ng in w ersetzt durch g´ + {Label der Kanten auf dem Weg von g nach vng im WALD[g]}. Diese Operation wird solange wiederholt, bis w nur noch Elemente enthaelt, die in der elementweise komplementierten Menge GEWAEHLT sind.

Beschreibung in Pidgin-Algol

```
procedure AUFROLLEN;
   begin
      while w enthaelt Literale, die durch andere bestimmt
      wurden do
      begin
         (* nicht-deterministische Wahl [w] *)
         sei ng ein Literal in w, der in SETZEB durch einen
         anderen Literal g bestimmt wurde ; d.h. WALD[g] enthaelt den Literal ng´ an den Punkten ; waehle ein
         Vorkommen vng von ng´ an den Punkten von WALD[g]; ersetze ng in w durch g´ zusammen mit den Labels des
         Weges von g nach vng in WALD[g] ;
         w := w - {ng} ;
      end
      (* w ist eine Teilmenge von GEWAEHLT *)
   end
```

Algorithmus SR

Die Grobstruktur von SR

```
Eingabe von s;
VEREINFACHE(s);
while Entscheid nicht gefaellt do
begin
   (* naechster Schritt: *)
   UEBRIG := s ;
   while in UEBRIG gibt es Variabeln zu interpretieren do
   begin
      Evaluiere die Variabeln in UEBRIG mit der for-Schleife
```

```
      EVAL. Die Evaluation kann 3 verschiedene Ergebnisse
      liefern : 1.Fall : "waehle", 2.Fall : "unerfuellt" und
      3.Fall : "isoliert". In jedem Fall wird ein Literal l
      gewaehlt, der anschliessend mit SETZEB(l,UEBRIG,*) in-
      terpretiert wird.
   end;
   (* Falls kein Modell gefunden wurde, so wurde s um eine Su-
   perresolvente erweitert. *)
   VEREINFACHE(s);
end;
Ausgabe der Entscheidung. (* Die Entscheidung lautet "erfuell-
bar", falls ein Modell gefunden wurde und "unerfuellbar",
falls die leere Superresolvente gebildet wurde *)
```

Nun folgt die detaillierte Beschreibung von SR:

```
begin
   UNERFUELLBAR := false ;
   SCHRITT := Ø ; VEREINFACHE(s) ;
   while nicht alle Klauseln von s sind erfuellt and not UNER-
   FUELLBAR do
   begin
      SCHRITT := SCHRITT + 1 ;
      UEBRIG := s ; GEWAEHLT := {} ; I1 := {} ;
      V1 :=  {die Menge der Variabeln in UEBRIG, die
      noch nicht definitiv interpretiert sind} ;
      while V1 # {} and
         in UEBRIG sind nicht alle Klauseln gestrichen
             do (* while-Schleife V1 :  *)
      begin
         for jede Variable v in V1 do  (* for-Schleife EVAL:*)
         begin
            UEBRIG1 := UEBRIG ; UEBRIG2 := UEBRIG ;
            SETZEB(v,UEBRIG1,WID1);
            SETZEB(v´,UEBRIG2,WID2);
            if WID1 and not WID2 then
            begin
               ISOLIERT[v] :=  true ;
            end else
            if not WID1 and WID2 then
            begin
               ISOLIERT[v] := true ;
            end else
            if WID1 and WID2 then
            begin
               LERNEND[v] := true;
               (* Die Variable v nennen wir lernend *)
            end else WAEHLBAR[v] := true;
            if LERNEND[v] then
            begin
               (* nicht-deterministische Wahl [wvar] *)
               waehle eine Variable wv in wvar[v] und waehle
               ein Vorkommen
```

```
      vpwv von wv und ein Vorkommen vnwv
      von wv´ an den Punkten von WALD[v]
      begin
         LABELSAMMELN(WALD[v],vpwv,vnwv,w1);
         (* nicht-deterministische Wahl [wvar] *)
         waehle eine Variable wv1 in wvar[v´]
         und waehle ein Vorkommen vpwv1 von wv1
         und ein Vorkommen vnwv1 von wv1´ an den
         Punkten von WALD[v´] ;
         begin
            LABELSAMMELN(WALD[v´],vpwv1,vnwv1,w2);
            AUFROLLEN(w1+w2);
            UNERFUELLBAR := w=[] ;
            res[v] := w ;
         end;
      end;
   end
end; (* Ende der for-Schleife EVAL *)
if fuer alle Variabeln v in V1 gilt WAEHLBAR[v] =
true then
(* Fall "waehle" :  *)
begin
   if es gibt Variabeln in V1, die nicht nur
   positiv oder nur negativ in UEBRIG vorkommen then
   begin
      waehle einen Literal l einer Variabeln in V1,
      der nicht nur positiv oder nur
      negativ in UEBRIG vorkommt;
      GEWAEHLT := GEWAEHLT + {l} ;
   end else waehle einen Literal l einer Variabeln in
   V1;
end else
if es gibt lernende Variabeln in V1 then
(* Fall "unerfuellt" :  *)
begin
   sei resver die Vereinigungsmenge der Klauseln in
   res[v] fuer alle lernenden Variabeln v;
   konstruiere eine neue Menge sres aus resver, indem
   in resver
   alle Klauseln entfernt werden, die in andern
   Klauseln von resver enthalten sind.
   (* nicht-deterministische Wahl [sres] *)
   waehle eine Klausel sr aus sres;
   s := s + {sr} ;
   (* s wird um die Superresolvente sr erweitert *)
   sei v1 eine lernende Variable mit der
   sr erhalten wurde;
   waehle einen Literal l von v1;
   entferne alle Klauseln von UEBRIG, die unerfuellt
   sind, wenn
   SETZEB(l,UEBRIG,*) angewendet wuerde ;
end else
(* mindestens eine Variable in V1 ist eindeutig be-
```

```
        stimmt, Fall "isoliert" :  *)
        begin
           sei l ein Literal, der eindeutig als wahr bestimmt
           ist;
           (* nicht-deterministische Wahl [wvar] *)
           waehle eine Variable wv in wvar[l´] und waehle
           ein Vorkommen vpwv von wv und ein Vorkommen
           vnwv von wv´ an den Punkten von WALD[l´] ;
           LABELSAMMELN(WALD[l´],vpwv,vnwv,w) ;
           AUFROLLEN ;
           if w#{} then
           begin
              sei ll der Literal von w, der zuletzt in GE-
              WAEHLT aufgenommen wurde ;
              erweitere WALD[ll] durch eine Kante ka von ll
              nach l´ ;
              label[ka] := w -{ll} ;
           end else s := s + {l} ;
        end;
        SETZEB(l,UEBRIG,*);
        Il := Il + I[l] ;
        Vl := Vl - {Variable, die zu l gehoert} ;
     end; (* Ende der while-Schleife Vl *)
     VEREINFACHE(s) ;
  end;
  if UNERFUELLBAR then write(´ unerfuellbar ´)
  else write(´ erfuellbar ´);
end.
```

Beispiel VG:

Eingabe-KNF :

```
1.  a b c
2.        d e f
3.              g h i
4.                    j k l
5.  a´    d´
6.  a´          g´
7.  a´                j´
8.        d´    g´
9.        d´          j´
10.             g´    j´
11.   b´    e´
12.   b´          h´
13.   b´                k´
14.         e´    h´
15.         e´          k´
16.               h´    k´
17.     c´    f´
18.     c´          i´
19.     c´                l´
20.         f´    i´
21.         f´          l´
22.               i´    l´
```

1. Schritt :
Nachdem die for-Schleife EVAL ausgefuehrt ist, tritt der Fall "waehle" ein. Annahme : a wird gewaehlt. Bei der Ausfuehrung von SETZEB(a,UEBRIG,*) am Schluss der while-Schleife V1 wird der Baum WALD[a] aufgebaut.

Wir stellen einen Baum folgendermassen dar : Eine Kante wird durch eine Folge von Sternen repraesentiert. Wenn eine Kante eine nicht leere Labelmenge hat, unterbrechen wir in der Mitte die Folge von Sternen und schreiben dort die Labelmenge hin. Die Kanten sind von links nach rechts oder von oben nach unten gerichtet. WALD[a] hat folgende Gestalt :

```
a **** d´
**
* *
*  *
*   *
j´    g´
```

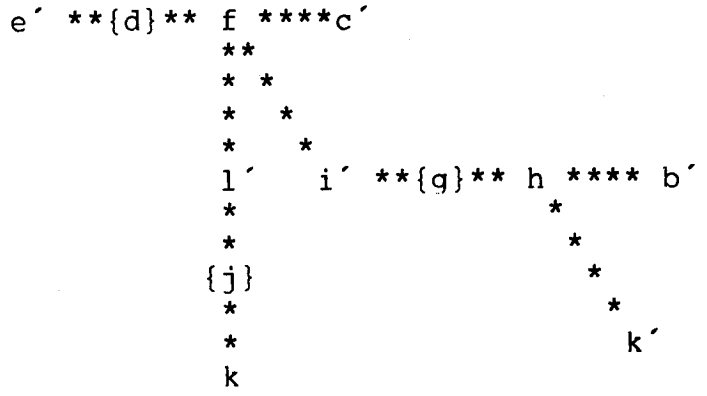
D.h. I[a] = {d´,g´,j´}. Die resultierende KNF UEBRIG sieht folgendermassen aus:
(Markierte Literale sind durch * ersetzt. Die Literale von gestrichenen Klauseln sind durch - ersetzt.)

```
-   - - -
2.          * e f
3.                * h i
4.                      * k l
-   *       -
-   *               -
-   *                       -
-           -       -
-           -               -
-                   -       -
11.   b´      e´
12.   b´              h´
13.   b´                      k´
14.           e´      h´
15.           e´              k´
16.                   h´      k´
17.       c´      f´
18.       c´              i´
19.       c´                      l´
20.               f´      i´
21.               f´              l´
22.                       i´      l´
```

In UEBRIG sind nicht alle Klauseln gestrichen, deshalb wird die for-Schleife EVAL nochmals ausgefuehrt. Jetzt tritt der Fall "unerfuellt" ein, und zwar sind die Variabeln e,f,h,i,k und l lernend. WALD[e] sieht z.B. folgendermassen aus :

```
        e**** b´
        **
        * *
        *  *
        *   *
        k´   h´ **{g}** i **** c´
        *                 **
        *                 * *
       {j}                *  *
        *                 *   *
        *                 l´   f´
        l
```

D.h. nach der Ausfuehrung von SETZEB(e,UEBRIG1,WID1) in der for-Schleife EVAL ist wvar[e] = 1 und WID1 ist wahr. WALD[e´] sieht z.B. folgendermassen aus :

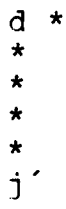


D.h. nach der Ausfuehrung von SETZEB(e´,UEBRIG2,WID2) ist wvar[e´] = k und WID2 ist wahr. Weil sowohl WID1 und WID2 wahr sind, ist die Variable e lernend.

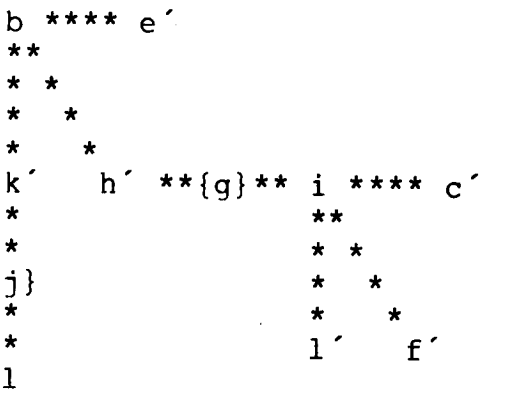
Der Aufruf der Prozedur LABELSAMMELN(e,l,l´,w1) liefert die Menge w1 = {g,j}. Der Aufruf von LABELSAMMELN(e´,k,k´,w2) liefert die Menge w2 = {d,g,j}. Deshalb ist w = w1 + w2 = {d,g,j}. Der Aufruf von AUFROLLEN liefert die Menge w = {a´}. Also ist a´ die (einzige) Klausel, welche in res[e] aufgenommen wird. Bei der Ausfuehrung der Anweisungen, die zum Fall "unerfuellt" gehoeren, wird die Klausel {a´} in sres aufgenommen. Wir nehmen an, dass diese Klausel bei der nicht-deterministischen Wahl [sres] gewaehlt wird. Also wird die Eingabe-KNF s um die Klausel {a´} erweitert. Nun wird die while-Schleife V1 verlassen, weil UEBRIG nach der Ausfuehrung von SETZEB(l,UEBRIG,*) keine Klausel mehr enthaelt. Nach der Ausfuehrung von VEREINFACHE(s) sieht s folgendermassen aus :

```
1.    b c
2.        d e f
3.              g h i
4.                    j k l
8.        d´    g´
9.        d´ .        j´
10.             g´    j´
11.   b´    e´
12.   b´          h´
13.   b´                k´
14.         e´    h´
15.         e´          k´
16.               h´    k´
17.     c´    f´
18.     c´          i´
19.     c´                l´
20.           f´    i´
21.           f´          l´
22.                 i´    l´
```

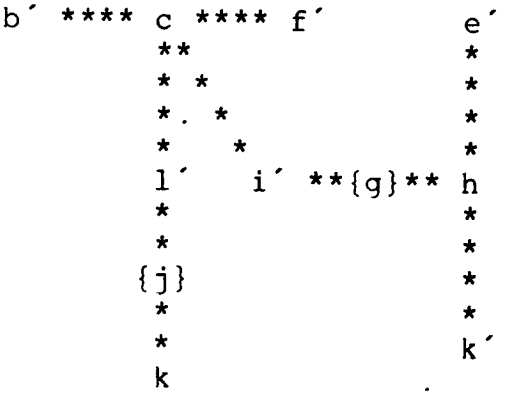
2. Schritt :
Nachdem die for-Schleife EVAL ausgefuehrt ist, tritt der Fall "waehle" ein. Annahme : d wird gewaehlt. Bei der Ausfuehrung von SETZEB(d,UEBRIG,*) wird WALD[d] folgendermassen aufgebaut :



In der resultierenden KNF UEBRIG sind nicht alle Klauseln gestrichen. Deshalb wird die for-Schleife EVAL nochmals ausgefuehrt. Jetzt tritt der Fall "unerfuellt" ein, und zwar sind die Variabeln b,c,h,i,k und l lernend. WALD[b] sieht z.B. folgendermassen aus :



Wald[b´] hat z.B. folgende Form :



Der Aufruf der Prozedur LABELSAMMELN(b,l,l´,wl) liefert die Menge wl = {g,j}. Der Aufruf von LABELSAMMELN(b´,k,k´,w2) liefert w2 = wl. Also ist w = {g,j}. Der Aufruf von AUFROLLEN liefert w = {d´}. Also ist {d´} eine Superresolvente. Nach der Ausfuehrung von VEREINFACHE sieht s folgendermassen aus :

```
1.    b c
2.          e f
3.              g h i
4.                    j k l
10.             g´    j´
11.   b´    e´
12.   b´          h´
13.   b´                k´
14.         e´    h´
15.         e´          k´
16.               h´    k´
17.     c´    f´
18.     c´          i´
19.     c´                l´
20.           f´    i´
21.           f´          l´
22.                 i´    l´
```

3. Schritt :
Nachdem die for-Schleife EVAL ausgefuehrt ist, tritt bereits der Fall "unerfuellt" ein. Weil die Menge GEWAEHLT leer ist, wird die leere Superresolvente gebildet.

Der Superresolutionsbeweis wurde also nach drei missglueckten Modellkonstruktionen gefunden. Er hat folgende Gestalt:
(In Klammer steht eine lernende Variable.)

```
1.
2.

  .
  .     Eingabe-KNF
  .

21.
22.
23. a´ (e)    .
24. d´ (b)
25.    (b)
```

Kommentar zum Aufwand der Beweisueberpruefung:
Jeder Beweisschritt (Erzeugung einer Superresolvente) ist auf einer DZM (Direktzugriffmaschine) linear in der Laenge der momentanen KNF s ueberpruefbar. Man verwendet dazu die Prozedur SETZE. Bei der wiederholten Ausfuehrung dieser Prozedur wird jeder Literal von s hoechstens zweimal bearbeitet, d.h. aus seiner Klausel entfernt. Um die Ueberlegenheit von Superresolution gegenueber Resolution am Beispiel VG zeigen zu koennen, geben wir einen Resolutionsbeweis fuer die Klausel {a´} an, der nicht stark verkuerzt werden kann. Der ganze Resolutionsbeweis waere zu lang. Hinter jeder Resolvente stehen in Klammern die Nummern der Elternklauseln.

1.	a	b	c										
2.				d	e	f							
3.							g	h	i				
4.										j	k	l	
5.	a′			d′									
6.	a′						g′						
7.	a′									j′			
8.				d′			g′						
9.				d′						j′			
10.							g′			j′			
11.		b′			e′								
12.		b′						h′					
13.		b′									k′		
14.					e′			h′					
15.					e′						k′		
16.								h′			k′		
17.			c′			f′							
18.			c′						i′				
19.			c′									l′	
20.						f′			i′				
21.						f′						l′	
22.									i′			l′	
23.							g	h				l′	(3,22)
24.					e′		g					l′	(14,23)
25.					e′					j		l	(4,15)
26.					e′		g			j			(24,25)
27.	a′				e′					j			(6,26)
28.	a′				e′								(7,27)
29.							g		i		k′		(3,16)
30.						f′	g				k′		(20,29)
31.						f′				j	k		(4,21)
32.						f′	g			j			(30,31)
33.				d	e		g			j			(2,32)
34.	a′				e		g			j			(5,33)
35.	a′				e					j			(6,34)
36.	a′				e								(7,35)
37.	a′												(28,36)

1.3 Der Zusammenhang zwischen SR und Superresolution

Ich zeige in diesem Abschnitt, dass jede von SR erzeugte Klausel im Fall einer missglueckten Modellkonstruktion eine Superresolvente ist. SR kann aber im allgemeinen fuer eine KNF s nicht alle moeglichen Superresolventen von s erzeugen. Das ist kein Nachteil von SR, sondern ein wesentlicher Vorteil, denn SR erzeugt nur gewisse "kurze" Superresolventen. Eine grosse Menge von Ballast-Superresolventen kann nicht erzeugt werden. Trotzdem ist SR, als Beweissystem aufgefasst, vollstaendig. Deshalb garantiert die Anwendung von SR auf eine unerfuellbare KNF, dass ein relativ kurzer, ballastfreier Superresolutionsbeweis entsteht.

Lemma 1.3.1 : Jede von Algorithmus SR in die Menge sres eingebrachte Klausel ist eine Superresolvente von s, d.h. jede von SR bei einer missglueckten Modellkonstruktion erzeugte Klausel ist eine Superresolvente.

Beweis:
Ich beziehe mich auf Algorithmus SR, Fall "unerfuellt". Sei sr eine Klausel in sres. Wenn genau die Literale in sr mit SETZE so gesetzt sind, dass sr unerfuellt ist, dann ist v lernend. Seien alle Literale in sr mit Hilfe von SETZE so gesetzt, dass sr unerfuellt ist. Es gebe eine unerfuellte Klausel in s. Dann kann sr nicht in sres sein, weil sr einen Literal enthaelt, der nicht in GEWAEHLT ist. Widerspruch. #

Das folgende Lemma zeigt, dass SR gewisse uninteressante Superresolventen nicht erzeugen kann.

Lemma 1.3.2 : Zu einer gegebenen KNF s koennen die Superresolventen in der Ausnahmemenge A, die im folgenden definiert wird, nicht von SR erzeugt werden.

Eine Superresolvente sr ist in A, gdw. es im Verlauf der Konstruktion von sr einen noch nicht gesetzten Literal l in der Literalmenge von s - sr gibt, so dass folgendes gilt :
1. Wenn l erfuellt wird, entsteht eine unerfuellte Klausel beim Anwenden von SETZE, d.h. l´ ist isoliert.
2. Sei I[l´] die Menge der durch l´ eindeutig bestimmten Literale. Reduziere s, indem nacheinander fuer alle Literale ll in I[l´] die Prozedur REDUZIERE(s,ll) ausgefuehrt wird. Dann darf die reduzierte Klausel sr im reduzierten s keine Superresolvente der reduzierten KNF mehr sein. #

Beispiel einer Superresolventen
in der Ausnahmemenge A :

1. a b c
2. a´ d e
3. a´ d e´
4. a b´ c
5. a f
6. a f´

7. c d

{c,d} ist eine Superresolvente der KNF s, bestehend aus den Klauseln 1 bis 6. {c,d} liegt aus folgendem Grunde in A : Wenn 1 = a erfuellt wird, entsteht beim Anwenden von SETZE eine unerfuellte Klausel. Also muss 1´ = a´ erfuellt werden. {c,d} ist aber keine Superresolvente mehr von s, nachdem REDUZIERE(s,a´) auf s angewendet wurde.

Beweis von Lemma 1.3.2 :
Sei sr eine Superresolvente von s, die in A ist. Dann kann sr nicht von SR erzeugt werden, weil SR jede isolierte Variable feststellt.

Es gibt noch weitere Superresolventen, die von SR nicht erzeugt werden koennen. Eine Superresolvente sr heisst verfremdet, gdw. es eine Teilmenge t von sr gibt, so dass gilt:
1. t ist eine Superresolvente von s.
2. Wenn t mit SETZE unerfuellt gesetzt wird, ist mindestens ein Literal l in sr nicht eindeutig bestimmt (falls sr nicht in s vorhanden ist). Wir nennen l einen fremden Literal der Superresolventen sr.

Beispiel einer verfremdeten Superresolventen:
Wir verwenden die KNF s des Beispiels VG vom letzten Abschnitt. {a´,b} ist eine verfremdete Superresolvente von s, denn {a´} ist auch eine Superresolvente von s. Auch wenn a wahr gesetzt wird, ist die Variable b nicht eindeutig bestimmt. Deshalb ist b ein fremder Literal der Superresolventen {a´,b}. Man beachte, dass SR die Superresolvente {a´,b} nicht erzeugen kann.

Es bleibt spaeteren Untersuchungen vorbehalten, zu ueberpruefen, unter welchen Umstaenden SR nur nicht verfremdete Superresolventen erzeugt, und SR allenfalls so abzuaendern, dass er nur nicht verfremdete Superresolutionsbeweise erzeugt. Diese Untersuchungen sind deshalb interessant, weil das Beweissystem "nicht verfremdete Superresolution" (keine Superresolvente darf verfremdet sein) vollstaendig ist, die Beweise jedoch kuerzer sind.

In den Abschnitten 1.10 und 1.11 werden unabhaengige und maximal gekuerzte Superresolventen untersucht. Beide Begriffe be-

zeichnen, wie der Begriff "nicht verfremdet", eine Einschraenkung auf kuerzere Superresolventen. Wir nennen eine Superresolvente technisch minimal, wenn sie die folgenden drei Eigenschaften besitzt: nicht verfremdet, unabhaengig und maximal gekuerzt. Beim Beweissystem "technisch minimale Superresolution" TMSR muss jede Superresolvente technisch minimal sein. TMSR ist vollstaendig, obwohl es eine starke Einschraenkung von Superresolution ist. Zudem sind aber die Beweise in TMSR im allgemeinen kuerzer als im Beweissystem Superresolution, und technisch minimale Superresolventen koennen mit kleinem polynomialem Aufwand konstruiert werden. Deshalb ist TMSR ein sehr attraktives Beweissystem.

Bemerkung zum Aufwand der Konstruktion einer Superresolventen:

Ein Schritt von SR (Konstruktion einer Interpretation) liefert eine Superresolvente ausser, wenn in diesem Schritt ein Modell gefunden wird. Dieser Fall ist jedoch selten. Der Aufwand von SR auf einer DZM (Direktzugriffmaschine) fuer einen Schritt ist - grob abgeschaetzt - beschraenkt durch l*l*l, wobei l die Literalanzahl der Eingabe-KNF ist. Diese obere Schranke ist unabhaengig von der Wahl der Varianten bei den nicht-deterministischen Anweisungen. (Es koennen auch Algorithmen fuer das Erzeugen von Superresolventen mit Aufwand O(l) angegeben werden, was vor allem fuer Anwendungen wichtig ist. Man beachte, dass SR mehr leistet, als was zur Erzeugung einer Superresolventen notwendig ist.) Also ist auf einer DZM ein Schritt von SR polynomial in der Laenge der behandelten KNF und somit ist auch das Erzeugen einer Superresolventen polynomial moeglich. Deshalb ist das Erzeugen eines Superresolutionsbeweises mit m Superresolventen und urspruenglichen Klauseln auf einer DZM polynomial in m.

1.4 Der Zusammenhang zwischen Resolution und Superresolution

In diesem Abschnitt zeige ich zuerst, dass jede von SR erzeugte Superresolvente eine Resolvente ist. Die Anzahl der Resolventenbildungen, die man benoetigt, um eine Superresolvente zu erhalten, wird nach oben abgeschaetzt. Daraus folgt, dass SR polynomial auf Resolution reduzierbar ist. Weil ein beliebiger Superresolutionsbeweis auf einen von SR erzeugbaren Superresolutionsbeweis verkuerzt werden kann, gilt: Superresolution ist polynomial reduzierbar auf Resolution.

Lemma 1.4.1 : Jede von SR erzeugte Superresolvente einer KNF s ist gleich dem Resultat einer Folge von Resolutionsoperationen.

Beweis:
Sei w die Menge von Literalen, die von LABELSAMMELN(l,vp,vn,w) berechnet wird. Dann ist die Menge w+{l} eine Resolvente von s, wobei bei jeder Resolventenbildung genau eine Elternklausel in s ist. Man bemerke, dass an jeder Kante ka eines Baumes in WALD im wesentlichen eine Klausel von s gespeichert ist. Sei h1 der Literal des Anfangspunktes, h= label[ka] und h2 der Literal des Endpunktes. Dann entspricht dieser Kante die Klausel c={h1´,h,h2}. Ebenso kann die Berechnung von AUFROLLEN mit Resolution simuliert werden. Auch hier ist bei jeder Resolventenbildung genau eine Elternklausel in s. #

Sei sr eine von SR erzeugte Superresolvente und v eine lernende Variable. Sei m die Anzahl Klauseln in s. Zur Bildung von sr + {v} werden hoechstens m Resolutionen benoetigt, denn jede Klausel in s kann hoechstens einmal als Elternklausel vorkommen. (Die Baeume in WALD mit Indizes in GEWAEHLT enthalten jede Klausel hoechstens einmal.)

Analoges gilt fuer sr + {v´} . Also sind zur Bildung von sr hoechstens 2*m + 1 Resolutionen noetig. Bei genau einer Anwendung von Resolution ist keine Elternklausel in s.

Lemma 1.4.2 : Sei s eine unerfuellbare KNF mit m1 Klauseln, und sei m die Anzahl Klauseln (inklusive die Klauseln in s) in einem von SR erzeugten Superresolutionsbeweis von s. Dann kann in polynomialer Zeit, abhaengig von der Laenge von s und des Superresolutionsbeweises, ein Resolutionsbeweis mit O(m*m) Resolutionen angegeben werden.

Beweis:
Die Anzahl Resolutionen ist beschraenkt durch

$$\sum_{k=0}^{m-m1-1} (2*(m1+k)+1).$$

#

Sei sr eine beliebige Superresolvente einer unerfuellbaren KNF s. Wir wenden SR auf s an und setzen dabei die Literale in sr unerfuellt. Sei sr1 die Superresolvente, die dabei von SR erzeugt werden muss. Es ist moeglich, dass sr1 eine Teilmenge von sr ist. Mit dieser Methode kann jeder Superresolutionsbeweis von hinten (beginnend mit der leeren Superresolventen) auf einen Beweis, der durch SR erzeugbar ist, verkuerzt werden. Deshalb haben wir bewiesen:

Satz 1.4.3 : Superresolution ist polynomial reduzierbar auf Resolution.

1.5 Die implizite Leistung von Superresolution

Obwohl ich spaeter beweisen werde, dass Resolution und Superresolution polynomial aequivalent sind, zeige ich in diesem Abschnitt, dass von einem anderen Gesichtspunkt her Superresolution exponentiell besser ist als Resolution.

Sei s eine erfuellbare KNF und R(s) die Menge aller Resolventen von s. Auf den "meisten" Formelmengen ist die Anzahl der Klauseln in R(s) eine exponentielle Funktion bezueglich der Klauselanzahl von s. Es wird sich zeigen, dass die Menge EE(s) der Klauseln in R(s), die keine Superresolventen von s sein koennen, gross ist. Jede KNF s kann polynomial so transformiert werden, dass EE(s) exponentiell in der Laenge von s waechst. Also kann man sagen, dass Superresolution exponentiell weniger Wahlen als Resolution fuer die Fortsetzung eines Beweises offen laesst. Deshalb beruecksichtigt SR, wenn ein Literal l gewaehlt wird, gleichzeitig exponentiell viele Resolventen im folgenden Sinn: Keine dieser Resolventen kann unerfuellt werden, wenn l und die durch l eindeutig bestimmten Literale wahr gesetzt werden.

Definition 1.5.1 : Sei s eine KNF. Dann heisst eine Klausel c durch einseitige Resolution erzeugt, wenn c in der folgenden Menge E(s) liegt:

1. Alle Klauseln in s sind in E(s).

2. Alle Resolventen von zwei Klauseln c1,c2 sind in E(s), falls c1 in E(s) und c2 in s ist.

3. Es gibt keine andern Klauseln in E(s).

Eine Klausel c heisst durch einseitige Resolution mit Einerklauselelimination erzeugt, wenn c in folgender Menge EE(s) liegt:

1. Alle Klauseln in E(s) sind in EE(s).

2. Sei l eine Klausel der Laenge 1, welche in EE(s) ist. Sei neg(l) die Menge der Klauseln in EE(s), welche den Literal l´ enthalten. Sei red die Menge von Klauseln, die erhalten wird, wenn bei jeder Klausel in neg(l) der Literal l´ weggelassen wird. Dann ist red auch in EE(s).

3. Es gibt keine andern Klauseln in EE(s).

Bemerkung :
E(s) ist die Menge aller durch Input-Resolution [CH73] erzeug-

baren Klauseln. Man kann zeigen: E(s) enthaelt die leere Klausel genau dann, wenn ein Resolutionsbeweis fuer s existiert, in dem bei jedem Schritt mindestens eine Elternklausel die Laenge 1 hat.

Lemma 1.5.1 : Sei s eine KNF und EE(s) die Menge der durch einseitige Resolution mit Einerklauselelimination erhaltenen Resolventen. Wenn EE(s) die leere Klausel nicht enthaelt, kann keine Klausel in EE(s) durch SR erzeugt werden. Wenn EE(s) die leere Klausel enthaelt, so erzeugt SR die leere Klausel im ersten Schritt.

Beweis:
1. Die leere Klausel sei nicht in E(s). Wir zeigen zuerst, dass keine Superresolvente in E(s) sein kann. Wir beweisen folgendes mit Induktion ueber die Struktur von E(s): Wenn eine Klausel in E(s) unerfuellt ist, so ist nach der Anwendung von SETZE eine Klausel in s unerfuellt. Die Aussage ist richtig fuer Klauseln in s. Sei c1 eine Klausel in E(s) und c2 eine Klausel in s. Sei r die Resolvente von c1 und c2.
a) r ist leer. Dies ist nicht moeglich, denn sonst waere die leere Klausel in E(s).
b) r ist nicht leer. Wenn r unerfuellt ist, so ist entweder c1 oder c2 als unerfuellt bestimmt. Wenn c2 unerfuellt ist, so ist eine Klausel in s unerfuellt. Wenn c1 unerfuellt ist, so wenden wir die Induktionsvoraussetzung an.

Also kann keine Klausel in E(s) eine Superresolvente sein. Wegen Lemma 1.3.1 kann somit keine Klausel in E(s) von SR erzeugt werden.

2. Die leere Klausel sei nicht in EE(s). Wir zeigen, dass SR keine Superresolvente in EE(s) erzeugt unter der bereits bewiesenen Annahme, dass keine Superresolvente in E(s) ist. Wir verwenden Induktion ueber die Struktur von EE(s). Fuer E(s) ist die Behauptung bewiesen. Sei l eine Klausel der Laenge 1 in EE(s). Dieser eindeutig bestimmte Literal wird von SR erkannt und stets erfuellt, falls das Komplement nicht auch eindeutig bestimmt ist. Sei c2 eine Klausel in EE(s), welche den Literal l´ enthaelt. Sei r die Resolvente von l und c2.
a) r ist leer. Dies ist nicht moeglich, denn sonst waere die leere Klausel in EE(s).
b) r ist nicht leer. Wenn r unerfuellt ist, so ist entweder l oder c2 unerfuellt. Da l nicht unerfuellt sein kann, ist stets c2 unerfuellt. Also kann r nicht von SR erzeugt werden, da sonst die Induktionsannahme verletzt waere.

Wenn E(s) die leere Klausel enthaelt, gilt folgendes: Sei v die Variable, die aufgeloest wurde, als die leere Klausel entstand. Wenn v von SR gesetzt wird, entsteht bei beiden Interpretationen eine unerfuellte Klausel. GEWAEHLT ist aber zu Beginn von SR leer, deshalb erzeugt SR die leere Klausel im ersten Schritt.

Wenn EE(s) die leere Klausel enthaelt, werden die eindeutig bestimmten Literale im ersten Schritt von SR wahr gesetzt. Fuer die resultierende reduzierte KNF s´ gilt E(s´)=EE(s´). Deshalb erzeugt SR die leere Klausel im ersten Schritt. #

Korollar 1.5.2 : Man braucht mindestens 3 Anwendungen von Resolution, um irgendeine Klausel (ausser die leere) zu erhalten, die mittels Superresolution mit einer einzigen Anwendung der Schlussregel erzeugbar ist.

Beweis:
Verwende die Tatsache, dass E(s) keine Superresolventen (ausser allenfalls die leere) enthaelt.
#

Korollar 1.5.3 : Sei s eine erfuellbare KNF. Dann erzeugt SR die Menge EE(s) in folgendem Sinn implizit: SR waehlt die Interpretation 1 der ersten zu setzenden Variabeln so, dass keine Klausel in EE(s) unerfuellt wird, wenn 1 und die durch 1 eindeutig bestimmten Literale gesetzt werden.

Beweis:
Annahme: Sei c eine Klausel in EE(s), die bei der ersten Anwendung von SETZE unerfuellt wird. Nach Beweis von Lemma 1.5.1 ist dann eine Klausel in s unerfuellt. Dies ist bei Algorithmus SR nur moeglich, wenn eine lernende Variable existiert. Wir erhalten einen Widerspruch zur Voraussetzung, dass s erfuellbar ist. #

Das Folgende soll darauf hinweisen, dass EE(s) normalerweise eine sehr grosse Menge ist.

Sei s eine erfuellbare KNF mit m verschiedenen Klauseln c(1),c(2), ... ;c(m). Wir betrachten folgenden Algorithmus, der eine Teilmenge En(s) von E(s) konstruiert :

```
for k:=1 to m do
begin
   bilde die Menge R der Resolventen zwischen c(k) und allen
   andern Klauseln in s ;
   erweitere s um R;
   eliminiere Klauseln in s, die mehr als einmal vorkommen,
   bis die Klauseln in s nur einmal auftreten;
end;
```

Lemma 1.5.4 :
1. Wenn R fuer ein k leer ist, so ist c(k) ohne Einfluss auf die Modelle von s.
2. En(s) - s enthaelt alle wichtige Information ueber s,

denn es gilt : En(s) - s ist erfuellbar, gdw. s erfuellbar ist. Es kann jedoch Modelle von En(s) - s geben, die keine Modelle von s sind. Jedes derartige Modell laesst sich jedoch leicht in ein Modell von s abaendern.

Beweis :
Siehe [ME68].

Obiger Algorithmus simuliert vollstaendig die Elimination einer beliebigen Variabeln in der Davis-Putnam-Prozedur [DA60]. Die weiteren Variabelneliminationen werden nur fragmentarisch simuliert. Fuer die Davis-Putnam-Prozedur ist bewiesen, dass sie schon auf einfachen Formeln exponentiell ist [TS68,KIR74]. Das ist ein Hinweis, dass En(s) und somit auch EE(s) normalerweise gross sind.

Definition 1.5.2 : Sei s eine erfuellbare KNF, auf die SR angewendet wird. Sei l der erste Literal, der von SR im ersten Schritt gewaehlt wird. Wir definieren VS(s) als die Anzahl der Resolventen von s, die nicht unerfuellt sind, bevor der naechste Literal gewaehlt wird.

Definition 1.5.3 : Sei s eine erfuellbare KNF mit m verschiedenen Klauseln und n Variabeln. s heisst normal, wenn card(EE(s)) $\geq$ 2**(n*1/3).

Wenn eine KNF s mit m Klauseln und n Variabeln nicht normal ist, kann sie auf natuerliche Art und mit polynomialem Aufwand folgendermassen in eine normale KNF verwandelt werden: Suche eine Interpretation I von s, die kein Modell ist. Erweitere s um eine Klausel c, in der jeder Literal durch I unerfuellt ist. Teile die Literale in c in n/2 disjunkte Literalpaare ein und fuehre fuer jedes dieser Paare eine Hilfsvariable ein. Die so erweiterte KNF s enthaelt nun n1 = n + n/2 Variabeln. E(s) enthaelt mindestens 2**(n1*1/3) = 2**(n/2) verschiedene Klauseln, denn die Klausel c kann mit einseitiger Resolution auf 2**(n/2) verschiedene Arten gekuerzt werden. Somit ist bewiesen:

Lemma 1.5.5 : Fuer alle erfuellbaren KNF s, die normal sind, ist VS(s) eine exponentielle Funktion in der Klauselanzahl von s.

SR hat fuer erfuellbare KNF die Eigenschaft, im Normalfall mit polynomialem Aufwand eine exponentielle Anzahl von Resolventen gleichzeitig zu beruecksichtigen, d.h. keine dieser Resolventen wird unerfuellt, wenn der naechste Literal und die durch

ihn eindeutig bestimmten Literale gesetzt werden. Von diesem Gesichtspunkt her kann man sagen, dass Superresolution exponentiell besser ist als Resolution.

Korollar 1.5.6 : Fuer die unerfuellbaren KNF s, fuer welche die leere Klausel in EE(s) liegt, kann effektiv ein polynomialer Resolutionsbeweis angegeben werden.

Beweis:
Verwende Saetze 1.4.2 und 1.5.1. #

Wir betrachten im folgenden Resolution und Superresolution als Schlussregelsysteme fuer UNERF. Sei UNERFNT die Menge der unerfuellbaren KNF s, bei denen fuer keine Variable v die Klausel v und die Klausel v´ in s vorkommt. Sei T ein Schlussregelsystem fuer UNERF, und sei s eine KNF. Dann ist T*(s) die Menge aller Klauseln, welche mit T aus s herleitbar sind.

Ein Schlussregelsystem T1 heisst eine Verbesserung des Schlussregelsystems T2, wenn fuer alle s in UNERFNT die Menge T1*(s) eine echte Teilmenge von T2*(s) ist [ME71]. Wir nennen einen Superresolutionsbeweis normal, wenn er von SR erzeugt wurde.

Lemma 1.5.7 : Normale Superresolution ist eine Verbesserung von Resolution.

Beweis :
Nach dem Beweis des Lemmas 1.5.1 gibt es fuer KNF in UNERFNT stets Resolventen, die keine Superresolventen sein koennen.

1.6 Resolution ist polynomial verkuerzbar auf Superresolution

In diesem Abschnitt zeige ich, dass Superresolution, ausser in trivialen Faellen, stets kuerzere Beweise erlaubt als Resolution, d.h. Beweise, die weniger Klauseln enthalten. Ein Beweissystem BW1 heisst polynomial verkuerzbar auf ein Beweissystem BW2, wenn fuer alle KNF s in UNERFNT gilt :

1. Zu jedem Beweis w1 von s in BW1 kann in polynomialer Zeit abhaengig von der Laenge von w1, ein Beweis w2 in BW2 fuer s angegeben werden.
2. Die Laenge von w2 (im Komplexitaetsmass des Beweissystems BW2) ist kuerzer als die Laenge von w1 (im Komplexitaetsmass des Beweissystems BW1).

Lemma 1.6.1 : Sei s eine KNF und E(s) die Menge der durch einseitige Resolution erhaltenen Resolventen. Wenn E(s) die leere Klausel nicht enthaelt, dann kann keine Klausel in E(s) eine Superresolvente sein.

Beweis:
Analog zum Beweis von Lemma 1.5.1 (1.Teil).

Lemma 1.6.2 : Sei s eine KNF und r(1), r(2), ... r(k)={} ein Resolutionsbeweis fuer s, der nicht trivial ist. Dann kann in polynomialer Zeit, abhaengig von der Laenge von s und des Beweises, ein kuerzerer Superresolutionsbeweis fuer s angegeben werden. d.h. Resolution ist polynomial verkuerzbar auf Superresolution (Komplexitaetsmass: Anzahl Resolventen, resp. Superresolventen).

Beweis:
Wir betrachten einen Abkuerzungsalgorithmus fuer r(1), r(2), ... r(k)={}. Sei r(j) (j zwischen 1 und k) die erste Klausel mit folgender Eigenschaft W: Wenn sie mit SETZE unerfuellt gesetzt wird, ist keine Klausel in s unerfuellt. r(j) muss aus folgenden Gruenden eine Superresolvente von s sein:

a) Wenn mit SETZE r(j) unerfuellt gesetzt wird, ist keine Klausel in s unerfuellt.
b) Wenn r(j) unerfuellt gesetzt ist, muss es eine lernende Variable v geben, naemlich die Variable, die aufgeloest wurde, als durch Resolution r(j) entstand. Denn wenn diese Variable gesetzt wird, muss bei beiden Interpretationen eine unerfuellte Klausel entstehen. Man beachte, dass r(j) die erste Klausel im Beweis ist, fuer die W gilt.

Falls r(j)#{}, so erweitere s um r(j) und wiederhole den Algorithmus so lange, bis j=k. Die ausgewaehlten Resolventen sind Superresolventen. Somit haben wir einen Superresolutionsbeweis

konstruiert. Weil k>1 und wegen Lemma 1.6.1, ist der Superresolutionsbeweis kuerzer als der gegebene Resolutionsbeweis. #

Betrachten wir folgenden Resolutionsbeweis :

```
1.  a  b  c
2.  a  b  c´
3.  a  b´    d
4.  a  b´    d´
5.  a´ b        e
6.  a´ b        e´
7.  a´ b´          f
8.  a´ b´          f´

9.  a  b                (1,2)
10. a  b´               (3,4)
11. a                   (9,10)
12. a´ b                (5,6)
13. a´ b´               (7,8)
14. a´                  (12,13)
15.                     (11,14)
```

Die KNF, deren Unerfuellbarkeit bewiesen wird, besteht aus den Klauseln 1 bis 8. Der Resolutionsbeweis besteht aus den Klauseln 9 bis 15. Die erste Superresolvente in diesem Beweis ist die Klausel 11 und die zweite die Klausel 15.

Lemma 1.6.4 : Resolution und Superresolution sind polynomial aequivalente Beweissysteme fuer UNERF.

Beweis :
Resolution < Superresolution (Lemma 1.6.2)
Superresolution < Resolution : verwende Lemma 1.4.3.
#

Satz 1.6.5 : Auch die folgenden Beweissysteme, die Abarten von Resolution sind, sind polynomial verkuerzbar auf Superresolution (Komplexitaetsmass: Anzahl Klauseln, d.h. Anzahl semantischer Resolventen, Anzahl Hyperresolventen etc.): semantische Resolution, Hyperresolution, set-of-support Resolution, Lockresolution, lineare Resolution, Resolution mit Mischen etc. (Fuer die Definitionen siehe [CH73].)

Beweis:
Verwende obigen Abkuerzungsalgorithmus.
#

1.7 Polynomiales Verhalten von SR auf polynomialen Mengen

In diesem Abschnitt analysiere ich das Verhalten von SR auf speziellen Formelmengen, von denen bekannt ist, dass sie in P sind [CO71, AA76]. Ich zeige, dass SR fuer alle diese Formelmengen uniform ein polynomialer Entscheidungsalgorithmus ist. Das weist darauf hin, dass SR generell die Eigenschaft hat, "einfache" Formeln rasch zu behandeln. Deshalb ist es vermutlich nicht noetig, erst viele weitere polynomiale Klassen zu suchen, sondern SR verhaelt sich vermutlich automatisch auf diesen Klassen polynomial.

Man beachte, dass SR ein Algorithmus mit nicht-deterministischen Anweisungen ist und dass die Aussagen, die wir ueber SR machen, fuer jede beliebige Wahl in den nicht-deterministischen Anweisungen richtig sind.

Definition 1.7.1 : Eine KNF s heisst Krom-KNF, wenn jede Klausel von s hoechstens zwei Literale enthaelt [KR67].

Lemma 1.7.1 : Auf erfuellbaren Krom-KNF erzeugt SR ein Modell im ersten Schritt, und auf unerfuellbaren Krom-KNF wird die leere Superresolvente im ersten Schritt konstruiert.

Beweis:
1. Sei s eine erfuellbare Krom-KNF. Annahme: SR findet das Modell nicht im ersten Schritt. Dann wird eine Superresolvente sr gebildet. sr muss leer sein, weil jede Kante ka in einem beliebigen Baum in WALD eine leere Labelmenge label[ka] hat. Widerspruch.
2. Sei s eine unerfuellbare Krom-KNF. Dann muss die leere Superresolvente im ersten Schritt von SR gebildet werden, weil auf Krom-KNF keine nicht leere Superresolvente konstruiert wird. #

Definition 1.7.2 : Eine KNF s heisst Horn-KNF, wenn jede Klausel in s hoechstens einen positiven Literal enthaelt.

Lemma 1.7.2 : Auf erfuellbaren Horn-KNF erzeugt SR ein Modell im ersten Schritt, und auf unerfuellbaren Horn-KNF wird die leere Superresolvente im ersten Schritt gebildet.

Beweis:
1. Sei s eine erfuellbare Horn-KNF. Wir fuehren den Beweis mit

Induktion ueber die Variablenanzahl n von s. Fuer n=1 ist die Behauptung richtig. Sei s eine erfuellbare Horn-KNF mit n Variablen. Wir wenden SR auf s an.
1.Fall: SR findet eine lernende Variable. Dies ist nicht moeglich, sonst waere s unerfuellbar (siehe 2. und 3. Fall). 2.Fall: SR findet eine Variable, deren Interpretation erzwungen ist. Dann muss die resultierende KNF UEBRIG immer noch erfuellbar sein, falls s erfuellbar war. UEBRIG enthaelt mindestens eine Variable weniger als s.
3.Fall: SR findet eine Variable v, die gewaehlt werden kann. Dann ist nach der Anwendung von SETZE keine Klausel in s unerfuellt. Die resultierende KNF UEBRIG enthaelt keine Klausel der Laenge 1. Also ist UEBRIG eine erfuellbare Horn-KNF. UEBRIG enthaelt mindestens eine Variable weniger als s.
Also erzeugt SR ein Modell im ersten Schritt.

2. Sei s eine unerfuellbare Horn-KNF. s muss mindestens eine Klausel {l} der Laenge 1 enthalten, die einen positiven Literal l enthaelt. Wir wenden SR auf s an. SR findet mindestens einen Literal ll, der eindeutig bestimmt ist. Dann enthaelt die reduzierte KNF REDUZIERE(s,ll) wieder eine Klausel der Laenge 1, denn sonst waere die reduzierte KNF erfuellbar. Man beachte, dass die Horn-Eigenschaft in bezug auf die Operation REDUZIERE invariant ist.
Deshalb erzeugt SR die leere Superresolvente im ersten Schritt. #

Definition 1.7.3 : Ein Modell M einer KNF heisst stark, wenn in jeder Klausel hoechstens ein Literal unerfuellt ist. Eine KNF heisst stark erfuellbar, wenn sie ein starkes Modell hat.

Bemerkung: Eine Horn-KNF, die keine Klausel der Laenge 1 enthaelt, ist stark erfuellbar.

Lemma 1.7.3 : Auf stark erfuellbaren KNF erzeugt SR ein Modell im ersten Schritt, d.h. fuer stark erfuellbare KNF ist die Modellkonstruktion in polynomialer Zeit moeglich.

Beweis:
Sei s eine mit Modell M stark erfuellte KNF. Wir fuehren den Beweis mit Induktion ueber die Anzahl der Variablen von s. Fuer n=1 ist die Behauptung richtig. Sei s eine stark erfuellbare KNF mit n Variablen. Wir wenden SR auf s an.
1.Fall: SR findet eine lernende Variable v. Dies ist nicht moeglich, sonst waere s unerfuellbar (siehe 2. und 3. Fall).
2.Fall: SR findet eine Variable v, deren Interpretation erzwungen ist. Falls s erfuellbar ist, muss die resultierende KNF UEBRIG immer noch erfuellbar sein. Ferner ist die KNF UEB-

RIG durch M stark erfuellt. UEBRIG enthaelt mindestens eine Variable weniger als s.
3.Fall: SR findet eine Variable v, die gewaehlt werden kann. Dann ist nach der Anwendung von SETZE keine Klausel in s unerfuellt. Die resultierende KNF UEBRIG enthaelt keine Klausel der Laenge 1. Deshalb laesst M in jeder Klausel von UEBRIG hoechstens einen Literal unerfuellt. Somit ist UEBRIG immer noch stark erfuellbar. UEBRIG enthaelt weniger Literale als s. Also erzeugt SR ein Modell im ersten Schritt. #

Bemerkung: Das von SR erzeugte Modell von s kann in gewissen Klauseln mehr als einen Literal unerfuellt lassen.

Lemma 1.7.4 : Es gibt einen polynomialen Algorithmus, der entscheidet, ob eine KNF stark erfuellbar ist.

Beweis:
Ersetze jede Klausel {l(1),l(2), ... ,l(n)} der Laenge groesser zwei durch binomial(n,2) Klauseln der Laenge zwei : {l(j),l(k)}, wobei j<k und j und k zwischen 1 und n variieren. Die erhaltene KNF ist eine Krom-KNF. Sie ist erfuellbar genau dann, wenn die urspruengliche KNF stark erfuellbar ist. #

1.8 Exponentielle Verbesserungen

Intuitiv ist folgendes klar: Fuer eine erfuellbare KNF s1 der Laenge g ein Modell zu finden, ist einfacher, als fuer eine unerfuellbare KNF s2, auch etwa der Laenge g, einen Resolutionsbeweis zu finden. Zu dieser Einsicht kommt man durch eine einfache Wahrscheinlichkeitsbetrachtung: Die erfuellbare KNF s1 enthalte n Variablen. Dann muss man n mal richtig "raten", um ein Modell zu finden.
Der kuerzeste Resolutionsbeweis fuer die unerfuellbare KNF s2 enthalte m Klauseln. Dann muss man mindestens m Klauseln richtig "raten", um einen Resolutionsbeweis zu finden. Da s1 und s2 etwa gleich gross sind, ist m im allgemeinen wesentlich groesser als n. Deshalb hat man eine wesentlich kleinere Chance, in einer vorgegebenen Zeiteinheit fuer s2 einen Resolutionsbeweis als fuer s1 ein Modell zu finden.

Wir bezeichnen dieses Phaenomen als "Einseitigkeitseffekt". Trotz des Einseitigkeitseffekts gilt: Genau dann, wenn fuer jede erfuellbare KNF ein Modell in polynomialer Zeit konstruiert werden kann, ist P=NP, d.h. das Konstruktionsproblem (fuer Modelle) und das Entscheidungsproblem (ob ein Modell existiert) sind etwa gleich schwierig. Dieses Paradoxon laesst sich nur dadurch erklaeren, dass die traditionellen Beweissysteme fuer Unerfuellbarkeit schlecht sind. Ich kann keine Alternative anbieten, weil eine solche einen tiefliegenden Satz ueber erfuellbare KNF als Voraussetzung haette. Hingegen kann ich fuer spezielle Formelmengen beweisen, dass einige traditionelle Beweissysteme schlecht sind.

In diesem Abschnitt beweise ich fuer eine Formelmenge TS, dass das Modellkonstruktionsproblem sogar exponentiell einfacher ist als das Konstruieren eines regulaeren Resolutionsbeweises.

Wir untersuchen zuerst eine Formelmenge TS von KNF, die von einer einfachen graphentheoretischen Konstruktion herruehrt [TS68]. Sei TSE die Menge der erfuellbaren KNF und TSU die Menge der unerfuellbaren KNF in TS. Es ist bewiesen, dass regulaere Resolution kein polynomiales Beweissystem ist fuer TSU [TS68,GA74]. Wir werden aber den Satz beweisen, dass SR auf den KNF in TSE das Modell im ersten Schritt findet. Deshalb sagen wir, dass das Modellkonstruktionsproblem fuer TS exponentiell einfacher ist als das Konstruieren eines Beweises im Beweissystem "regulaere Resolution". Um mit SR zu beweisen, dass eine KNF in TS unerfuellbar ist, muss man nur eine Interpretationskonstruktion ausfuehren. Wenn sie kein Modell liefert, ist bewiesen, dass die KNF unerfuellbar ist. Man beachte, dass SR ein Algorithmus mit nicht-deterministischen Anweisungen ist und dass die Aussagen ueber SR fuer jede beliebige Wahl in den nicht-deterministischen Anweisungen gelten.

Nachher interessiert uns eine Klasse V von KNF, welche mit der Faerbbarkeit von Graphen zusammenhaengt. Sei UV die Menge der unerfuellbaren KNF in V und EV die Menge der erfuellbaren. Es ist bewiesen, dass Gentzen ohne Schnitt [CO74] kein polynomiales Beweissystem fuer UV ist [WI76]. Es wird auch allgemein vermutet [COO76], dass Resolution kein polynomiales Beweissystem fuer UV ist. Die Menge V enthaelt nebst den unerfuellbaren KNF verschiedene Arten erfuellbarer KNF, die auf natuerliche Art von den unerfuellbaren abgeleitet sind. Wir werden SR mit einem Literalauswahlkriterium zu einem Algorithmus SRMAX einschraenken und zeigen, dass SRMAX das Modell fuer KNF in EV stets im ersten Schritt findet. Also tritt hier eine exponentielle Vereinfachung gegenueber dem Beweissystem "Gentzen ohne Schnitt" auf. SRMAX ist immer noch ein Algorithmus mit nichtdeterministischen Anweisungen, und obige Aussage gilt fuer jede beliebige Wahl bei den nicht-deterministischen Anweisungen.

Ich vermute, dass die oben erwaehnten exponentiellen Vereinfachungen auch fuer andere Beweissysteme und fuer interessantere Formelmengen gelten.

a) Eine KNF heisst Tseitin-KNF, wenn sie wie folgt konstruiert wird: Sei G = (V,E) ein ungerichteter Graph ohne Schleifen, d.h. es gibt keine Kante (v,v). Sei L1 eine Funktion, die jedem Punkt v den Wert L1(v) in {0,1} zuordnet. Sei L2 eine eineindeutige Funktion, die jeder Kante eine Variable zuordnet. Betrachte einen Punkt v in V. v habe Grad g und z(1),z(2), ... ,z(g) seien die Variablen, welche zu den mit v inzidenten Kanten gehoeren. Eine Klausel c gehoert zum Punkt v, gdw.

$$c = \{y(1),y(2), \ldots ,y(g)\} ,$$

wobei y(i) entweder z(i) oder das Komplement von z(i) ist. Die Paritaet der Anzahl komplementierter z(i) in c muss der Paritaet des Wertes L1(v) entgegengesetzt sein.

Wir bezeichnen mit TS(G(L1,L2)) die Menge aller Klauseln, die zu allen Punkten von G gehoeren, wobei L1,L2 zwei beliebige Funktionen sind, die obigen Bedingungen genuegen. Sei e(G(L1)) die Summe modulo 2 der Funktionswerte von L1 ueber alle Punkte von G. Es gilt TS(G(L1,L2)) ist erfuellbar, gdw. e(G(L1))=0.

Satz 1.8.1 : Fuer eine erfuellbare KNF TS(G(L1,L2)) findet Superresolution das Modell im ersten Schritt.

Beweis :
Wir verwenden Induktion ueber die Kantenanzahl n von G.
Fuer n=1 ist die Aussage richtig.
Sei s eine KNF TS(G(L1,L2)) mit e(G(l1))=0. s enthalte n

Literale.

1. Bei der Anwendung von SR trete der Fall "waehle" ein. SR waehlt einen beliebigen Literal y und setzt ihn wahr. Die Variable zu y gehoere zur Kante k. Wir untersuchen fuer jeden Endpunkt v von k, wie sich die zu v gehoerende Klauselmenge aendert, falls y wahr gesetzt wird. Sei C(v) die Menge der Klauseln, die zu v gehoeren, und sei D(v) die KNF, die nach der Anwendung von REDUZIERE(C(v),y) erhalten wird. D(v) ist gerade die Klauselmenge, die im reduzierten Graph zu v gehoert, wobei aber L1(v) geaendert wurde (Ø wird zu 1 und umgekehrt).

Seien v1 und v2 die Endpunkte der Kante k. Wir aendern die Funktion L1 an diesen beiden Punkten und nennen die neue Funktion L1´ (Ø wird zu 1 und umgekehrt). Diese Aenderung hat keinen Einfluss auf die Erfuellbarkeit. Wir aendern L2 zu L2´ ab, wobei L2´ auf der Kante k nicht mehr definiert ist. Auf den restlichen Kanten ist L2´=L2. Die nach der Ausfuehrung von REDUZIERE(s,y) vorhandene KNF ist aber TS(G(L1´,L2)) und somit erfuellbar.

2. Fall "eindeutig" : Analog wie bei "waehle".
3. Fall "unerfuellt" : Ist nicht moeglich.
#

b) Im folgenden betrachten wir eine eingeschraenktere Version SRMAX von SR, die ein Literalauswahlkriterium verwendet. SRMAX arbeitet wie SR, ausser wenn der Fall "waehle" eintritt. Dann wird ein Literal so gewaehlt, dass die resultierende KNF UEBRIG moeglichst wenige Klauseln enthaelt.

Sei vg(p,q) die KNF, welche ausdrueckt, dass der vollstaendige Graph mit p Punkten nicht q faerbbar ist. vg(p,q) ist erfuellbar, gdw. p < q ist. vg(p,q) hat die Form :

$$\bigwedge_{i=1..p} \Big(\bigvee_{j=1..q} a(i,j) \Big) \;\&$$

$$\bigwedge_{\substack{j=1..q \\ i1=1..p \\ i2=i1+1..p}} (a(i1,j)' \vee a(i2,j)')$$

vg(p,q) enthaelt p Klauseln der Lange q. Sie druecken aus, dass jeder Punkt mindestens eine Farbe haben muss. vg(p,q) enthaelt q * binomial(p,2) Klauseln der Laenge 2. Sie druecken aus, dass keines der binomial(p,2) Punktepaare dieselbe Farbe haben kann.

Sei VG die Menge der KNF vg(p,q). Sei VG1(p) die Menge der KNF vg(p,p-1), wobei eine Klausel der Laenge p-1 weggelassen ist. Sei VG2(p) die Menge der KNF vg(p,p-1), wobei eine Klausel der Laenge 2 weggelassen ist. Sei VG1 (VG2) die Vereinigung der Mengen VG1(p) (VG2(p)), wobei p=1,2,.. . VG1 und VG2 koennen als die schwierigsten erfuellbaren Umgebungen der unerfuellbaren KNF in VG angesehen werden. Sei V = VG + VG1 + VG2.

Satz 1.8.2 : Auf den erfuellbaren KNF in V findet SRMAX ein Modell im ersten Schritt.

Beweis:
1.VG2(p),p>3.
O.B.d.A sei die Klausel {a(p-1,p-1)´, a(p,p-1)´} weggelassen. Wenn eine Variable aus a(1,1),a(1,2), ... a(p-2,p-1) wahr gesetzt wird, so werden h = p*(p-1) * 1/2 + (p-1) * (p-2) + 1 Klauseln gestrichen. Wenn eine solche Variable falsch gesetzt wird, so werden p-1 Klauseln gestrichen. Wenn eine Variable a(p-1,1), a(p-1,2), ... a(p,p-2) (ohne die Variablen a(p-1,p-1) und a(p,p-1)) wahr gesetzt wird, so werden h-1 Klauseln gestrichen. Wenn eine solche Variable falsch gesetzt wird, so werden p-1 Klauseln gestrichen. Wenn eine der Variablen a(p-1,p-1) oder a(p,p-1) wahr gesetzt wird, so werden h-1 Klauseln gestrichen. Wenn eine dieser Variablen falsch gesetzt wird, so werden p-2 Klauseln gestrichen. Also waehlt das Literalauswahlkriterium einen der Literale a(1,1),a(1,2), ... a(p-2,p-1) aus. Dadurch wird VG2(p) auf eine Formel VG2(p-1) reduziert.

2.VG2(p),p≤3.
VG2(p) enthaelt Klauseln hoechstens der Laenge 2, deshalb findet SRMAX ein Modell im ersten Schritt.

3.VG1(p),p>3.
O.B.d.A. sei die Klausel {a(p,1), a(p,2), ... a(p,p-1)} weggelassen. Nun kommen die Variablen a(p,1), a(p,2), ... a(p,p-1) nur noch negativ vor, also werden sie von SRMAX falsch gesetzt. Dabei werden (p-1)*(p-1) Klauseln gestrichen. Die restliche KNF VG3(p-1) drueckt aus: " der Graph mit p-1 Punkten ist (p-1)-faerbbar". Wir untersuchen nun das Verhalten von SRMAX auf den Formeln VG3(p).

a) VG3(p),p>3
VG3(p) enthaelt p Klauseln der Laenge p und p*(p-1)/2 * p Klauseln der Laenge 2. VG3(p) ist symmetrisch, d.h. jede Variable ist gleichwertig. Wenn a(1,1) wahr gesetzt wird, so werden p*(p-1)/2 + (p-1)*(p-1)+1 Klauseln gestrichen. Wenn a(1,1) falsch gesetzt wird, so werden p Klauseln gestrichen. Also waehlt das Literalauswahlkriterium einen positiven Literal aus. Dies fuehrt zu einer Formel VG3(p-1).

b) VG3(p),2≤p≤3
Bei diesen Formeln wird ein Modell im ersten Schritt gefunden, weil sie Klauseln hoechstens der Laenge 2 enthalten.

4. VG1(p),p≤3.
Bei der ersten Interpretationskonstruktion wird ein Modell gefunden.

Fuer die Formeln vg(p,q) mit q>p findet SRMAX ein Modell im ersten Schritt. (Beweis wie bei VG3.)

#

Beweisstruktur:

VG2(p),p>3 -> VG2(p-1),p>2
VG2(p),2≤p≤3 SRMAX findet Modell bei der ersten Interpretationskonstruktion.

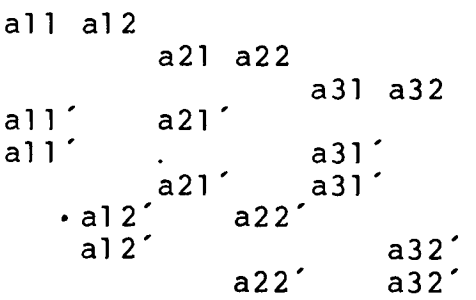
VG1(p),p>3 -> VG3(p-1),p>2

VG3(p),p>3 -> VG3(p-1),p>2
VG3(p),2≤p≤3 SRMAX findet Modell bei der ersten Interpretationskonstruktion.

VG1(p),2≤p≤3 SRMAX findet Modell bei der ersten Interpretationskonstruktion.

Beispiele:

VG(3):



VG(4):

```
a11 a12 a13
            a21 a22 a23
                        a31 a32 a33
                                    a41 a42 a43
a11'        a21'
a11'                    a3'
a11'                                a41'
            a21'        a31'
            a21'                    a41'
                        a31'        a41'
    a12'        a22'
    a12'                    a32'
    a12'                                a42'
                a22'        a32'
                a22'                    a42'
                                a32'    a42'
        a13'        a23'
        a13'                    a33'
        a13'                                a43'
                    a23'        a33'
                    a23'                    a43'
                                a33'        a43'
```

VG3(3):

```
a11 a12 a13
            a21 a22 a23
                        a31 a32 a33
a11'        a21'
a11'                    a31'
            a21'        a31'
    a12'        a22'
    a12'                    a32'
                a22'        a32'
        a13'        a23'
        a13'                    a33'
                    a23'        a33'
```

1.9 Beschleunigtes Berechnen Boolescher Funktionen

Sei s eine gegebene KNF mit n Variablen x(1),x(2), ... x(n) , welche eine Funktion f darstellt. Wir nehmen an, wir haetten diese Funktion f immer wieder auszuwerten. Dabei setzen wir voraus, dass bei der Berechnung von f fuer eine Interpretation I1 die Wahrheitswerte der Variablen x(1),x(2), ... x(n) nicht zu Beginn geliefert werden. Vielmehr koennen wir bestimmen, fuer welche Variablen wir die Wahrheitswerte wissen moechten. Der Aufwand fuer die Berechnung des Wahrheitswertes einer der n Variablen sei sehr gross, z.B. c. Wir sagen, das Abfragen eines Wahrheitswertes sei ueberfluessig, wenn aus den bereits erhaltenen Wahrheitswerten geschlossen werden kann, ob f wahr oder falsch ist. Unser Ziel ist, ueberfluessiges Abfragen zu vermeiden. Obige Fragestellung ist eng verknuepft mit Problemen, welche bei der Vereinfachung von Schaltkreisen auftauchen.

Zuerst beschreibe ich den Algorithmus SRE, der KNF geschickt auswertet. Wir nennen eine KNF gesaettigt, wenn Algorithmus SR im ersten Schritt entweder immer ein Modell findet oder die leere Superresolvente erzeugt (bei beliebiger Wahl in den nicht-deterministischen Anweisungen). Wir zeigen, dass SRE auf gesaettigten KNF optimal arbeitet, d.h. keine ueberfluessigen Abfragen macht. Zuletzt beschreiben wir Algorithmus SROPT, der eine gegebene KNF in eine gesaettigte ueberfuehrt.

Algorithmus SR kann bequem dazu verwendet werden, den Wahrheitswert einer KNF s bei einer Interpretation I1 geschickt zu berechnen. Wenn in SR eine Variable zu interpretieren ist, so koennen folgende Faelle auftreten : 1.waehle, 2.unerfuellt, 3.isoliert. Im Fall "waehle" ist die Interpretation der Variablen, fuer die der Wahrheitswert gewaehlt wird, wie in I1 zu setzen. Im Fall "unerfuellt" ist s unerfuellt. Im Fall "isoliert" ist zu ueberpruefen, ob der eindeutig bestimmte Literal in I1 ist. Wenn er nicht in I1 vorkommt, dann erfuellt I1 die KNF s nicht.
Falls noch kein Entscheid getroffen wurde, dann ist bei der anschliessenden Anwendung von SETZEB zu ueberpruefen, ob jeder eindeutig bestimmte Literal in I1 ist. Wenn er nicht in I1 vorkommt, dann erfuellt I1 die KNF s nicht. Wenn UEBRIG keine Klauseln mehr enthaelt, dann ist I1 ein Modell.

Wir nennen diesen ergaenzten Evaluationsalgorithmus SRE. Wir nehmen an, dass SRE abbricht, sobald der Entscheid gefunden wurde.

Man beachte, dass SRE ein Algorithmus mit nicht-deterministischen Anweisungen ist und dass alle Aussagen ueber SRE fuer jede beliebige Wahl in den nicht-deterministischen Anweisungen

gelten.

Wir nennen eine KNF s optimal, wenn SRE auf s keine ueberfluessigen Abfragen macht.

Lemma 1.9.1 : Wenn eine KNF s gesaettigt ist, so ist s optimal.

Beweis:
Annahme: SRE mache eine ueberfluessige Abfrage fuer eine Interpretation I.

a) I ist ein Modell. Die Variable v sei ueberfluessig abgefragt, d.h. es sind schon alle Klauseln erfuellt, ohne dass v interpretiert ist. Widerspruch.
b) I ist kein Modell. Die Variable v sei ueberfluessig abgefragt, d.h. jede Erweiterung der bisher abgefragten Interpretation ergibt kein Modell. Wenn eine KNF gesaettigt ist, so ist auch jede mit REDUZIERE erhaltene KNF gesaettigt. Deshalb kann s nicht gesaettigt sein. Widerspruch. #

Bemerkung:
Wir nennen eine KNF prim, wenn sie nur Prim-Implikanten enthaelt. Wenn eine KNF prim ist, so ist sie gesaettigt und optimal. Es gibt KNF, die gesaettigt, hingegen nicht prim sind, z.B. gewisse Horn-KNF. Die Eigenschaft "gesaettigt" ist in dieser Hinsicht eine Verallgemeinerung der klassischen Eigenschaft "prim". Superresolution kann analog wie Resolution dazu verwendet werden, alle Prim-Implikanten einer KNF zu finden [HU68, REU76]. Bekanntlich ist aber Superresolution wesentlich oekonomischer als Resolution.

Sei s eine KNF und IM eine Menge von Interpretationen von s. s heisst IM-optimal, wenn SRE bei jeder Interpretation in IM keine ueberfluessige Abfrage macht.

Im folgenden skizzieren wir einen Algorithmus SROPT, der eine beliebige KNF s in eine IM-optimale KNF verwandelt. Der Aufwand kann im schlimmsten Fall exponentiell sein. SROPT verwendet folgendes Prinzip: Wenn SRE eine Interpretation in IM als Nicht-Modell erkennt, wird versucht, ein Modell zu konstruieren. Dabei wird die Interpretation der Variablen, die eine unerfuellte Klausel verursacht hat, geaendert. Wenn kein Modell gefunden wird, so wird s um eine Superresolvente erweitert. Interpretationsauswertungen von Interpretationen in IM werden solange wiederholt, bis die erweiterte KNF IM-gesaettigt ist. Dabei heisst eine KNF s IM-gesaettigt, falls folgendes gilt: Wenn s unerfuellbar ist, erzeugt SR die leere Klausel im ersten Schritt, und falls s erfuellbar ist, gilt fuer jede Teilinterpretation I einer Interpretation in IM, die noch zu einem Modell erweitert werden kann: Wenn I + {l} noch zu einem Modell erweitert werden kann, hingegen I + {l´} nicht

mehr, so ist l entweder eindeutig bestimmt oder isoliert.

Dieser Algorithmus ist bemerkenswert, weil er sich automatisch an eine Menge IM von Interpretationen anpasst und auf diesen Interpretationen den Wert mit immer weniger ueberfluessigen Abfragen berechnet, bis das Optimum erreicht ist. Unter Umstaenden werden Interpretationsauswertungen nicht nur von Interpretationen in IM optimiert.

1.10 Unabhaengige Superresolventen

Bei Algorithmus SR koennen die Literale einer durch ihn erzeugten Superresolvente in folgendem Sinn abhaengig sein: es kann einen Literal in sr geben, der durch andere Literale in sr eindeutig bestimmt ist. Abhaengige Superresolventen koennen also durch kleinere ersetzt werden. Wir aendern deshalb SR ab zu einem Algorithmus SRU, der nur unabhaengige Superresolventen erzeugt. SRU besitzt nur den Vorteil, dass die durch ihn erzeugten Superresolventen i.a. kuerzer sind als die von SR erzeugten. Die Anzahl der Superresolventen hingegen wird nicht kleiner.

Betrachten wir folgendes Beispiel:

1.		b	c	x´			
2.		b´		x			
3.			c´	x			
4.	a			x	y´		
5.	a´				y		
6.				x´	y		
7.					y´	d e	f
8.					y´	d´ e	f
9.					y´	d e´	f
10.					y´	d e	f´
11.		b´				d´ e´	f
12.		b´				d´ e	f´
13.		b´				d e´	f´
14.		b´				d´ e´	f´

Wir betrachten den Berechnungsablauf von SR im ersten Schritt, wobei die Literale in folgender Reihenfolge gewaehlt werden: a´,y,b,d. Der Baum, der bei der Wahl von a´ entsteht, besteht nur aus a´. Bei der Wahl von y entsteht folgender Baum:

y **{a}** x

Der Baum, der bei der Wahl von b entsteht, besteht nur aus b. Analoges gilt fuer d. Nun ist die Variable e lernend. Der Baum WALD[e] hat folgende Gestalt:

```
      e **{b´,d´}** f
      *
      *
   {b´,d´}
      *
      *
      f´
```

Der Baum WALD[e´] sieht folgendermassen aus:

```
      e´ **{y´,d´}** f
      *
      *
   {b´,d´}
      *
      *
      f´
```

Also wird die Superresolvente {b´,d´,y} gebildet, obwohl auch {b´,d´} eine Superresolvente ist.

Wir nennen eine Superresolvente sr abhaengig, gdw. fuer die literalweise komplementierte Klausel sr, die wir sr´ nennen, gilt: sr´ besitzt einen Literal, der durch andere Literale in sr´ eindeutig bestimmt ist. Wie kann man nun erreichen, dass SR nur unabhaengige Superresolventen erzeugt ? Eine naheliegende Methode waere, eine gegebene Superresolvente zu testen, ob sie abhaengige (d.h. durch die anderen Literale bestimmte) Literale enthaelt. Dies wuerde linearen Suchaufwand bedeuten.

Durch eine Aenderung in der Prozedur SETZEB kann man ohne zu suchen erreichen, dass SR nur noch unabhaengige Superresolventen erzeugt. Wir aendern SETZEB wie folgt zu einer Prozedur SETZEB1 ab:

a) Klauseln, die erfuellt sind, werden nicht gestrichen, sondern inaktiviert. Erst wenn sie durch zwei Literale erfuellt sind, werden sie aus UEBRIG gestrichen.
b) SETZEB1 stellt mit Hilfe der inaktivierten Klauseln fest, wenn ein Literal nachtraeglich eindeutig bestimmt ist. Solche Literale nennen wir sekundaer eindeutig bestimmt. Falls ein sekundaer eindeutig bestimmter Literal in GEWAEHLT ist, wird er aus GEWAEHLT entfernt, falls er nicht durch sich selbst eindeutig bestimmt ist.
c) Als Labelmengen der Kanten werden urspruenglich nur Literale verwendet, die momentan in GEWAEHLT sind. (Verwendung von AUFROLLEN)
d) Es wird erreicht, dass die Zuordnung von Literalen zu Punkten in WALD eineindeutig ist.

Nun folgt die detaillierte Beschreibung von SETZEB1 :

```
procedure SETZEB(l,UEBRIG,WIDERSPRUCH);
   begin
      sel := {}; selalt := {};
      el := {l} ;
      WIDERSPRUCH := false; wvar[l] := {};
      initialisiere WALD[l] durch seine Wurzel l;
      repeat
         (* Nicht-deterministische Wahl [el] *)
         sei l1 ein Element aus el;
         el := el - {l1} ;
         I[l] := I[l] + {l1} ;
         seien kpl1 die Klauseln in UEBRIG, welche den Literal
         l1 enthalten;
         for  jede Klausel c in kpl1 do
         if c ist inaktiviert then entferne c aus UEBRIG
         else inaktiviere c;
         seien knl1 die Klauseln in UEBRIG, welche den Lite-
         ral l1´ enthalten ;
         for jede Klausel c in knl1, welche nicht inaktiviert
         ist do
         begin
            markiere den Literal l1´ in der Klausel c von UEB-
            RIG ;
            if c enthaelt genau einen Literal l2, der nicht
            markiert ist then
            begin
               if es gibt einen Punkt in WALD[l], dem der Li-
               teral l2 zugeordnet ist then
               erweitere WALD[l] durch eine Kante ka von l1
               nach dem alten Punkt l2 else
               erweitere WALD[l] durch eine Kante ka von l1
               nach dem neuen Punkt l2;
               w := c - {l1} - {l2} ;
               if w # {} then AUFROLLEN;
               label[ka] := w;
               if l2´ in el then
               begin
                  wvar[l] := wvar[l] + { Variable, die zu Li-
                  teral l2 gehoert } ;
                  WIDERSPRUCH := true ;
               end else el := el + {l2};
            end;
         end;
         QUER;
      until el={} ;
      while sel # {} do
      begin
         (* Nicht-deterministische Wahl [sel] *)
         sei l1 ein Element aus sel;
         sel := sel - {l1};
         selalt := selalt + {l1};
         seien knl1 die Klauseln in UEBRIG, welche den Literal
         l1´ enthalten;
```

```
         QUER;
      end;
      entferne alle o-Kanten aus UEBRIG;
   end;
```

Dabei ist die Prozedur QUER folgendermassen definiert:

```
procedure QUER;
   begin
      for jede Klausel in knl1, die inaktiviert ist do
      begin
         if c enthaelt genau einen Literal l2, der nicht mar-
         kiert ist then
         begin
            erweitere WALD durch eine o-Kante ka von l1 nach
            dem alten Punkt l2;
            w := c - {l1} - {l2};
            if w # {} then AUFROLLEN;
            label[ka] := w;
            if not (l2 in GEWAEHLT) then
            begin
               if not (l2 in selalt) then sel := sel + {l2}
            end else
            begin
               if (es gibt einen Weg we von l nach l2, sodass
               keine Kante den Literal l2 in der mit AUFROLLEN
               auf Literale in GEWAEHLT transformierten Label-
               menge hat)
               and (der Punkt l ist verschieden von Punkt l2)
               then
               begin
                  sei w die Labelmenge des Weges we;
                  GEWAEHLT := GEWAEHLT - {l2};
                  erweitere WALD[l] durch eine Kante ka von l
                  nach l2;
                  label[ka] := w;
               end else (* l2 hat sich selbst zur Vorausset-
               zung *)
            end;
         end;
      end;
   end;
```

Wir ersetzen in SR am Schluss der while-Schleife V1 die Prozedur SETZEB durch SETZEB1. Den Algorithmus, den wir erhalten, nennen wir SRU.

Lemma 1.10.1 : Algorithmus SRU erzeugt nur unabhaengige Superresolventen.

Beweis:

Nach Konstruktion von SRU.
#

Wir wenden SRU auf das zuletzt behandelte Beispiel an. Der Baum, der bei der Wahl von a´ entsteht, besteht nur aus seiner Wurzel a´. Bei der Wahl von y entsteht folgender Baum:

```
y **{a}** x
```

Also ist Literal x eindeutig bestimmt und Klausel 5 wird aus UEBRIG gestrichen, weil sie zweifach erfuellt ist. Die Klauseln 2,3,4 und 6 sind inaktiviert. Bei der Wahl von b entsteht folgender WALD:
(o-Kanten sind durch eine Folge von o dargestellt, die Kantenrichtung ist durch einen Pfeil markiert.)

```
a

  <ooooooo
y          x <oooo b
  **{a}**>
```

Also ist y in GEWAEHLT sekundaer eindeutig bestimmt und hat sich nicht selbst zur Voraussetzung. Deshalb wird y aus GEWAEHLT entfernt. Nachher hat WALD folgende Gestalt:

```
a

b **** y **{a}** x
```

Der Baum, der bei der Wahl von d entsteht, besteht nur aus der Wurzel d. Nun ist die Variable e lernend. Der Baum WALD[e] sieht folgendermassen aus:

```
   e **{b´,d´}** f
   *
   *
{b´,d´}
   *
   *
   f´
```

Der Baum WALD[e´] hat folgende Gestalt:

```
   e´ **{b´,d´}** f
   *
   *
{b´,d´}
   *
   *
   f´
```

Also wird die Superresolvente {b´,d´} gebildet.

Kommentar zum Aufwand:
Der Aufwand wird nur wenig groesser. SRU laesst sich so implementieren, dass bei Eingabe von s die Erzeugung einer unabhaengigen Superresolventen hoechstens Aufwand O(l**2) hat. Dabei ist l die Literalanzahl in s.

1.11 Maximal gekuerzte Superresolventen

Bei Algorithmus SRU kann eine durch ihn erzeugte Superresolvente sr im folgenden Sinn kuerzbar sein: Es kann eine Teilmenge t von sr mit card(t) > 1 geben, die durch einen Literal l so ersetzt werden kann, dass die resultierende Klausel immer noch eine Superresolvente ist. Dies ist z.B. dann der Fall, wenn ein Literalpaar durch eine Hilfsvariable ersetzt werden kann. Wie wir in Abschnitt 1.12 sehen werden, koennen Superresolutionsbeweise durch die Einfuehrung von Hilfsvariablen stark verkuerzt werden bezueglich der Anzahl Superresolventen. (In [CO076] wird eine Folge von KNF angegeben, bei der man vermutet, dass jeder Superresolutionsbeweis exponentiell ist. Hingegen ist ein polynomialer Superresolutionsbeweis bekannt, wenn die Einfuehrung von Hilfsvariablen erlaubt ist.) Deshalb wollen wir SRU so abaendern, dass er vorhandene Hilfsvariablen verwendet. Wenn er nur maximal gekuerzte Superresolventen erzeugt, so verwendet er automatisch vorhandene Hilfsvariablen.

Betrachten wir folgendes Beispiel:

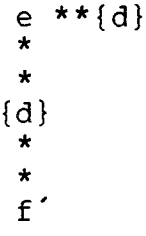
```
1.  a  b  c  z´
2.  a´        z
3.     b´     z
4.        c´  z
5.            z  d  e  f
6.               d´ e  f
7.               d  e´ f
8.               d  e  f´
9.               d´ e´ f
10.              d´ e  f´
11.              d  e´ f´
12.              d´ e´ f´
```

Wir betrachten den Berechnungsablauf von SR im ersten Schritt, wobei die Literale in folgender Reihenfolge in GEWAEHLT aufgenommen werden: a´,b´,c´,d´. Der Baum, der bei der Wahl von a´ entsteht, besteht nur aus a´. Analoges gilt fuer b´. Bei der Wahl von c´ entsteht folgender Baum:

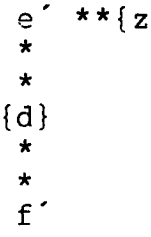
```
      c´ **{a,b}** z´
```

Der Baum, der bei der Wahl von d´ entsteht, besteht nur aus der Wurzel d´. Nun ist die Variable e lernend.

WALD[e] hat folgende Gestalt:



WALD[e´] sieht folgendermassen aus:



Also wird die Superresolvente {a,b,c,d} gebildet. Sie ist kuerzbar, denn {z,d} ist auch eine Superresolvente.

Wir nennen eine Superresolvente sr einer KNF s maximal gekuerzt, wenn fuer keine Teilmenge t von s mit card(t) > 1 ein Literal l in s existiert, so dass die Klausel (sr - t) + l eine Superresolvente von s ist.

Wie kann man erreichen, dass SRU nur maximal reduzierte KNF erzeugt ? Eine naheliegende Methode waere, fuer jede Teilmenge einer Superresolvente sr zu testen, ob eine Ersetzung moeglich ist. Dies wuerde aber exponentiellen Suchaufwand bedeuten. Durch eine Aenderung in der Prozedur SETZEB1 kann man mit polynomialem Aufwand erreichen, dass SR nur noch maximal gekuerzte Superresolventen erzeugt.

Dazu verwenden wir folgende Prozedur SETZEB2: Wir ersetzen in SETZEB1 die Anweisung "entferne alle o-Kanten aus UEBRIG" durch folgende Anweisungen:

```
ABKUERZEN := true;
while es hat sekundaer eindeutig
bestimmte Literale 12 in GEWAEHLT,
die sich selbst zur Voraussetzung haben and ABKUERZEN do
begin
   sei SEG die Menge der o-Kanten, die auf einem Weg von l
   nach einem sekundaer eindeutig bestimmten Literal (#l) in
   GEWAEHLT liegen;
   sei G der Teilgraph von WALD, der aus allen Kanten in SEG
   besteht;
   P := Punktmenge von G;
   ABKUERZEN := false;
   while P enthaelt Punkte and not ABKUERZEN do
```

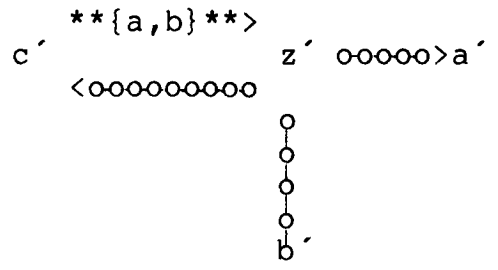
```
   begin
      sei p ein Punkt in P, und sei H(p) der Kantenteilgraph
      von G, der alle Kanten von G enthaelt, die man von p aus
      erreichen kann;
      sei GEW(p) die Menge der Blaetter von H(p);
      (* sie sind in GEWAEHLT *)
      sei WERT(p) die Menge der Punkte
      in GEW(p), die eindeutig
      bestimmt sind, wenn GEW(p) aus
      GEWAEHLT entfernt und p in GEWAEHLT
      aufgenommen wird;
      if card(WERT(p)) > 1  then ABKUERZEN := true;
      P := P - {p};
   end;
   if ABKUERZEN then
   begin
      GEWAEHLT := (GEWAEHLT + {p}) - GEW(p);
      die o-Kanten in H(p) werden zu normalen Kanten;
      entferne die Kanten, die zu p fuehren;
      l := p;
   end;
end;
entferne alle o-Kanten aus UEBRIG;
```

Wir ersetzen in SRU am Schluss der while-Schleife V1 die Prozedur SETZEB1 durch SETZEB2. Den Algorithmus, den wir erhalten, nennen wir SRUG.

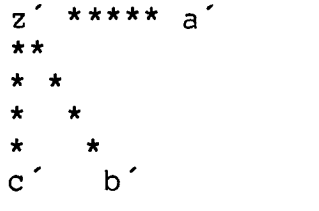
Lemma 1.11.1 : Algorithmus SRUG erzeugt nur unabhaengige und maximal gekuerzte Superresolventen.

Beweis:
Wegen Lemma 1.10.1 erzeugt SRUG nur unabhaengige Superresolventen. Annahme: SRUG erzeuge eine Superresolvente sr, die kuerzbar ist. Also existiert eine Teilmenge t von sr mit card(t) > 1 und ein Literal l, so dass (sr - t) + l eine Superresolvente ist. Deshalb sind die komplementierten Literale in t durch l´ eindeutig bestimmt unter der Voraussetzung, dass die Literale in sr - t unerfuellt sind. Dann existiert ein Graph H(l) mit der Eigenschaft, dass GEW(l) = {die komplementierten Literale in t}. Deshalb wird ABKUERZEN wahr und die Literale in t koennen nicht mehr in einer Superresolvente vorkommen. Widerspruch.

Wir wenden SRUG auf das zuletzt behandelte Beispiel an. Die Literale werden in folgender Reihenfolge in GEWAEHLT aufgenommen: a´,b´,c´,d´. Der Baum, der bei der Wahl von a´ entsteht, besteht nur aus der Wurzel a´. Analoges gilt fuer b´. Bei der Wahl von c´ entsteht folgender WALD:



Also sind die Literale a´,b´ und c´ sekundaer eindeutig bestimmt, sie haben jedoch sich selbst zur Voraussetzung. Die Funktion WERT(z´) hat den Wert 3. Deshalb werden die Literale a´,b´ und c´ aus GEWAEHLT entfernt und z´ wird in GEWAEHLT aufgenommen. WALD[z´] hat folgende Gestalt:

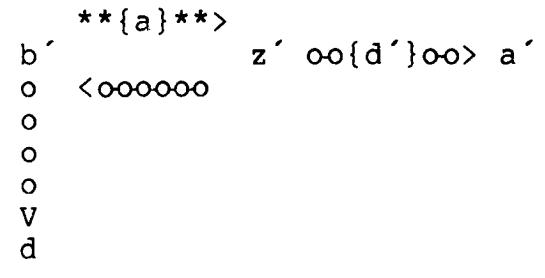


Der Baum, der bei der Wahl von d´ entsteht, besteht nur aus der Wurzel d´. Nun ist die Variable e lernend und es wird die Superresolvente {z,d} gebildet.

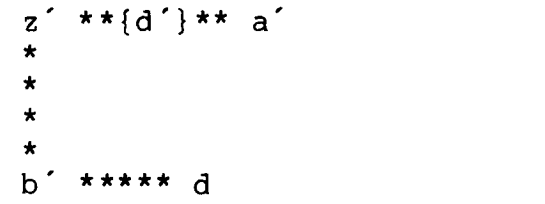
Wir untersuchen ein weiteres Beispiel:

1.	a	b	z´		
2.	a´		z	d´	
3.		b´	z		
4.		b		d	
5.				d	e f
6.				d´	e f
7.				d	e´ f
8.				d	e f´
9.	a			d´	e´ f
10.				d´	e f´
11.				d	e´ f´
12.		b		d´	e´ f´

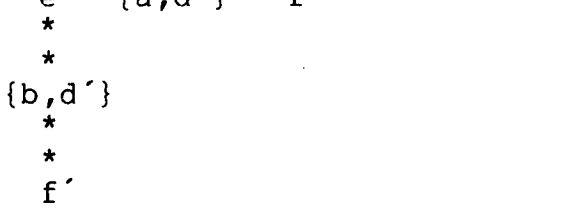
Wir betrachten den Berechnungsablauf von SRUG im ersten Schritt, wobei die Literale in folgender Reihenfolge in GEWAEHLT aufgenommen werden: d,a´,b´. Der Baum, der bei der Wahl von d entsteht, besteht nur aus der Wurzel d. Analoges gilt fuer a´. Bei der Wahl von b´ entsteht folgender WALD:



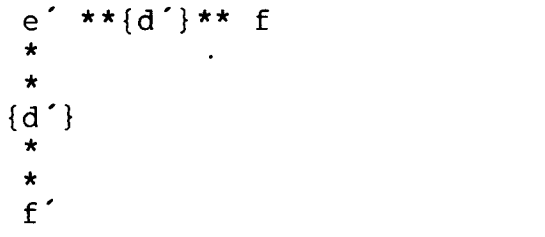
Also sind die Literale a´,b´ und d eindeutig bestimmt. a´ und b´ haben sich selbst zur Voraussetzung, hingegen d nicht. Deshalb wird die Kante b´ oooo> d durch eine gewoehnliche Kante ersetzt. Weil WERT(z´)=2 ist, wird anschliessend WALD wie folgt transformiert:



Nun ist die Variable e lernend.
WALD[e] hat folgende Gestalt:



WALD[e´] sieht folgendermassen aus:



Also wird die Superresolvente z gebildet.

Kommentar zum Aufwand:
Es ist ueberraschend, dass maximal gekuerzte Superresolventen mit polynomialem Aufwand erzeugt werden koennen. SRUG kann so implementiert werden, dass die Erzeugung einer Superresolvente hoechstens Aufwand O(l**3) hat. l ist die Anzahl Literale in der Eingabe-KNF.

1.12 Erweiterte Superresolution

Analog zur erweiterten Resolution [TS68, CO74] fuehren wir die erweiterte Superresolution ein. Wir zeigen, dass erweiterte Superresolution einschraenkender als erweiterte Resolution ist, d.h. es gibt weniger Superresolventen als Resolventen. Trotzdem erlaubt erweiterte Superresolution, ausser in trivialen Faellen, kuerzere Beweise als erweiterte Resolution, d.h. Beweise mit weniger Superresolventen. Wir definieren einen Algorithmus ESR, der erweiterte Superresolutionsbeweise in einer "kurzen" Normalform erzeugt. Fuer eine Eingabe-KNF mit n Variablen erzeugt ESR erweiterte Superresolventen hoechstens der Laenge n, unabhaengig davon, wieviele Hilfsvariablen eingefuehrt wurden.

Sei s eine KNF. Eine erweiterte Superresolvente von s ist eine Superresolvente der KNF s, in der durch iterierte Anwendung der Erweiterungsregel Hilfsvariablen eingefuehrt wurden. Seien x und y beliebige Literale in s, sei h eine neue Hilfsvariable und A(h,x,y) die Klauselmenge {(x´,h), (y´,h), (x,y,h´)}, welche h = x v y ausdrueckt. Die Erweiterungsregel ist folgendermassen definiert:
1. Waehle x und y in s.
2. Erweitere s um die Klauseln A(h,x,y).

Sinngemaess ist der Begriff erweiterter Superresolutionsbeweis definiert.

Die Zweckmaessigkeit der Einfuehrung von Hilfsvariablen ist in folgenden aussagenlogischen Regeln begruendet:

$$(a \vee b)' = a' \,\&\, b'$$
$$a \vee (c \,\&\, d) = (a \vee c) \,\&\, (a \vee d)$$

Dank diesen Regeln ist es moeglich, KNF, die viele Klauseln enthalten, durch die Einfuehrung von Hilfsvariablen kurz darzustellen.

Betrachten wir ein Beispiel. Die KNF s(i) sei wie folgt definiert:
1. s(i) enthaelt i+1 Variablen x(1), x(2), ... x(i+1).
2. s(i) ist wahr, gdw. die Anzahl wahr interpretierter Variablen ungerade ist.

s(2) hat z.B. folgende Form:

1. x(1) x(2)
2. x(1)´ x(2)´

Bei trivialer Darstellung enthaelt s(i) abhaengig von i expo-

nentiell viele Klauseln. Wenn Hilfsvariablen eingefuehrt werden, ist eine polynomiale Darstellung moeglich. Um s(i) (i > 2) kurz darzustellen, gehen wir so vor: Wir spalten zwei Variablen x(1) und x(2) ab und fuehren die folgenden drei Hilfsvariablen ein:

$$\begin{aligned} h(1) &= x(1) \;\; v \;\; x(2) \\ h(2) &= x(1)' \; v \;\; x(2)' \\ h(3) &= h(1)' \; v \;\; h(2)' \end{aligned}$$

h(3) ist wahr, gdw. genau eine der beiden Variablen x(1), x(2) wahr interpretiert wird. Nun haben wir das urspruengliche Problem P(i+1) reduziert auf folgendes Problem P(i): s(i) ist wahr, gdw. eine ungerade Anzahl der Variablen h(3), x(3), ... x(i+1) wahr interpretiert ist. Wir wenden das skizzierte Verfahren so lange an, bis wir beim Problem P(2) angelangt sind.

Mit aehnlichen Methoden kann man dank Hilfsvariablen Resolutionsbeweise kurz darstellen. Diese Tatsache hat Tseitin in [TS68] bemerkt. Er hat fuer die Tseitin-KNF einen polynomialen Beweis mit erweiterter Resolution angegeben, obwohl jeder regulaere Beweis ueberpolynomial ist. Fuer die unerfuellbaren KNF, die in [COO76] beschrieben sind, wird vermutet, dass jeder Resolutionsbeweis ueberpolynomial waechst. Hingegen existiert ein polynomialer Beweis fuer erweiterte Resolution.

Im folgenden aendern wir Algorithmus SRUG ab, so dass er erweiterte Superresolventen erzeugen kann:

1. Die Prozedur VEREINFACHE wird so ergaenzt, dass Hilfsvariablen eingefuehrt werden koennen.
2. Nach der nicht-deterministischen Wahl [sres], bei der eine Superresolvente sr gewaehlt wurde, wird folgende Anweisungsfolge eingefuegt:

```
while (Laenge(sr) > 3) do
begin
   (* nicht-deterministische Wahl [hvar] *)
   waehle zwei beliebige Literale l1,l2 in sr und fuehre
   eine neue Hilfsvariable z = l1 v l2 ein; erweitere s um
   die entsprechenden Klauseln; ersetze in allen Klauseln,
   die das Literalpaar l1,l2 enthalten, dieses Literalpaar
   durch z;
end;
```

3. Bei der nicht-deterministischen Wahl [sres] soll stets eine kuerzeste Superresolvente gewaehlt werden.

Diesen Algorithmus nennen wir ESR.

Lemma 1.12.1 : ESR fuehrt fuer kein Literalpaar zweimal eine Hilfsvariable ein.

Beweis:
Nach Lemma 1.11.1 erzeugt ESR nur maximal gekuerzte Superresolventen. Deshalb kann kein Literalpaar in einer Superresolventen vorkommen, fuer das bereits eine Hilfsvariable gebildet wurde.

Lemma 1.12.2 : Sei s eine KNF, auf die ESR angewendet werde. Von den urspruenglichen Variablen in s seien n1 entweder in GEWAEHLT oder eindeutig bestimmt. Dann ist eine Hilfsvariable, welche durch die Interpretationen der n1 Variablen eindeutig bestimmt waere in GEWAEHLT, gdw. sie mindestens einen urspruenglichen Literal eindeutig bestimmt.

Beweis:
Verwende die Tatsache, dass ESR nur unabhaengige und maximal gekuerzte Superresolventen (und Mengen GEWAEHLT) erzeugt.

Lemma 1.12.3 : Sei s eine KNF mit anfaenglich n Variablen, auf die ESR angewendet werde. Dann enthaelt sres bei jedem Schritt eine Superresolvente hoechstens der Laenge n, unabhaengig davon, wieviele Hilfsvariablen durch ESR gebildet wurden.

Beweis:
Verwende Lemma 1.12.2. Weil es eine Superresolvente geben muss, welche durch Resolution nur aus urspruenglichen Klauseln gebildet wird, muss es eine Superresolvente hoechstens der Laenge n geben.

Lemma 1.12.4 :
1. Erweiterte Superresolution ist polynomial verkuerzbar auf erweiterte Resolution.
2. Erweiterte Resolution und erweiterte Superresolution sind polynomial aequivalent.

Beweis:
Analog 1.6.

1.13 Exponentielle untere Schranken fuer Aufzaehlungsalgorithmen und Superresolution

In diesem Abschnitt gebe ich exponentielle untere Schranken fuer eine spezielle Art von Superresolution und fuer eine Variante von SR an. Sei {SR} die Menge aller Varianten des nicht-deterministischen Algorithmus SR. Ich vergleiche {SR} mit der Klasse der Aufzaehlungsalgorithmen und weise darauf hin, dass {SR} effizientere Algorithmen enthaelt als die Klasse aller Aufzaehlungsalgorithmen. Die Algorithmen in {SR} sind somit keine klassischen Backtrackalgorithmen. Meine Ueberlegungen deuten an, dass es schwierig ist, exponentielle untere Schranken fuer Superresolution zu erhalten. Weil Superresolution und Resolution polynomial aequivalent sind, ist es ebenso schwierig, exponentielle untere Schranken fuer Resolution zu finden. Als wichtiges Hilfsmittel verwende ich ein Resultat von Galil [GAL76] ueber Aufzaehlungsalgorithmen.

Sei s eine unerfuellbare KNF. Ein Aufzaehlungsbaum fuer s ist ein binaerer Baum T, so dass gilt:

1. Jedem Punkt von T ist eine KNF zugeordnet.
2. Die KNF, welche zur Wurzel gehoert, ist eine Teilmenge der Klauseln von s.
3. Eine KNF, die zu einem Blatt gehoert, enthaelt die leere Klausel.
4. Falls s(i) die KNF eines inneren Punktes in T ist, dann gilt:
   a) s(i) enthaelt die leere Klausel nicht.
   b) Falls i1,i2 die Nachfolger von i sind, dann gibt es eine Variable x in s(i), so dass
      A) s(i1) eine Teilmenge der durch Anwendung von REDUZIERE(s(i),x) erhaltenen KNF s(i) ist und
      B) s(i2) eine Teilmenge der durch Anwendung von REDUZIERE(s(i),x´) erhaltenen KNF s(i) ist.

Bemerkung: s ist unerfuellbar, gdw. ein Aufzaehlungsbaum fuer s existiert.

Jedem inneren Punkt eines Aufzaehlungsbaumes entspricht eine partielle Interpretation, die durch den Weg zu diesem Punkt bestimmt ist. Zu jedem inneren Punkt gehoert auch eine Variable, die auf zwei Arten interpretiert wird. Bei den Blaettern laesst die partielle Interpretation eine Klausel unerfuellt. Wir sagen, dass diese Klausel den Ast von der Wurzel bis zum Blatt abschliesst.

Die Komplexitaet N(T) eines Aufzaehlungsbaumes T ist die Anzahl verschiedener KNF, die den Punkten von T zugeordnet sind. Wir nennen einen Aufzaehlungsbaum einen semantischen Baum, wenn wir die KNF, die den Punkten zugeordnet sind, nicht beruecksichtigen. Die Komplexitaet eines semantischen Baumes ist

deshalb die Anzahl der Punkte des Baumes.

In [GAL76] ist folgender Satz bewiesen:

> Satz 1.13.1 : Es gibt eine Folge t(1), t(2), ... von KNF und Konstanten iØ und c>Ø, so dass fuer jeden Aufzaehlungsbaum T(i) fuer t(i) und fuer alle i>iØ gilt: N(T(i)) $\geq$ 2**(c * Laenge(t(i))).

Sei s eine KNF und I eine partielle Interpretation der Variabeln in s. Sei s[I] die KNF, die entsteht, wenn die Variablen in s gemaess I interpretiert werden. s[I] ist die Menge der Klauseln c', wobei c' eine Teilklausel einer Klausel c in s ist, welche durch I nicht erfuellt wird. c' enthaelt genau die Literale von c, die durch I nicht interpretiert werden, also sind alle Literale in c-c' durch I "falsch" interpretiert.

Eine Menge von Klauseln c[1], c[2], ... c[n] heisst baumeinbettbar, wenn

1. eine Variable v in s existiert mit der Eigenschaft, dass jede nicht-leere Klausel in s einen Literal einer Variablen v in s enthaelt und
2. fuer jede partielle Interpretation I der Variablen von s gilt: Es gibt eine Variable v von s, die durch I nicht interpretiert wird, so dass jede nicht-leere Klausel in s[I] einen Literal der Variabeln v enthaelt.

Ein Superresolutionsbeweis R(s) = sr[1], sr[2], ... sr[n] einer KNF s heisst einfach, wenn R(s) baumeinbettbar ist.

Die Laenge NS(R(s)) eines Superresolutionsbeweises R fuer die KNF s ist die Anzahl verschiedener Superresolventen in R plus die Anzahl Klauseln in s.

Im folgenden erlaeutern wir, wie ein Superresolutionsbeweis in einen Aufzaehlungsbaum uebersetzt werden kann, und zeigen, weshalb diese Uebersetzung i.a. nicht polynomial ist. Fuer einfache Superresolution ist die Uebersetzung hingegen polynomial und liefert deshalb eine untere Schranke. Sei s eine KNF und R(s) = s + sr(1) + sr(2) + ... + sr(m) ein Superresolutionbeweis fuer s mit m Superresolventen. Wir ordnen der Superresolventen sr(i) die Menge e(i) der - nach Bildung der Superresolvente sr(i) - global eindeutig bestimmten Literale zu. Dabei heisst ein Literal l global eindeutig bestimmt, wenn bei der Anwendung von SETZE(l,s,*) eine Klausel in s unerfuellt ist. Ohne Beschraenkung der Allgemeinheit nehmen wir an, dass die eindeutig bestimmten Literale als Klauseln in R(s) vorkommen.

Wir entfernen aus e(k) diejenigen Literale l, fuer die folgendes gilt: Wenn die Variablen in s so interpretiert werden, wie e(k) - {l} als Interpretation aufgefasst angibt, bleibt eine

gesaettigte KNF uebrig. Sei e(k) die so gereinigte Klausel und sei x(R(s)) diejenige Variable in e(k), die zuerst in der Folge e(1),e(2), ... e(k) vorgekommen ist. Wir nennen x(R(s)) die Wurzelvariable des Superresolutionsbeweises R(s).

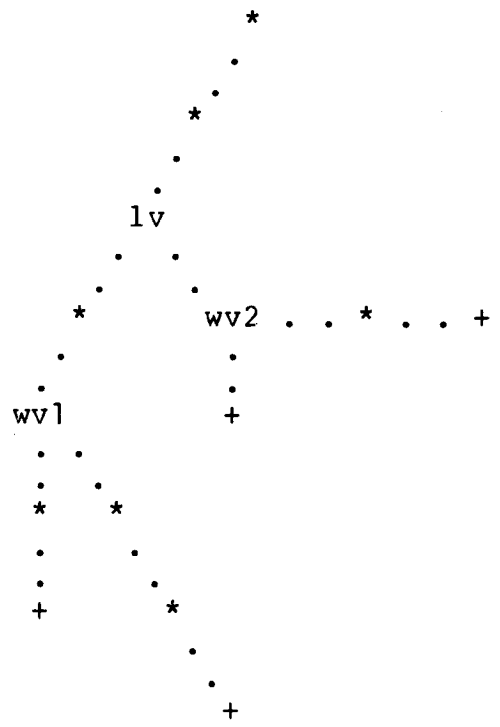
Nun ordnen wir einem Superresolutionsbeweis durch folgenden Algorithmus einen Aufzaehlungsbaum zu:

1. Zuerst bestimmen wir rekursiv eine Folge w(1), w(2), ... w(n) von Wurzelvariablen. w(1) ist die Wurzelvariable von R(s). Sei l die eindeutig bestimmte Interpretation von w(k). Dann ist w(k+1) die Wurzelvariable des Superresolutionsbeweises, der erhalten wird, wenn in R(s) der Literal l durch SETZE wahr gesetzt wird.

2. Sei s eine KNF und sr eine Superresolvente von s. Wir bezeichnen mit SE(s,sr) die KNF, die erhalten wird, wenn die Literale in sr mit SETZE unerfuellt gesetzt werden. Sei R[i](s) der Teil des Superresolutionsbeweises R(s) bis zur i-ten Superresolventen. Wir ordnen jeder Superresolventen sr(i) einen Aufzaehlungsbaum von SE(R[i](s), sr(i)) zu.
   Wir uebersetzen jede Superresolvente sr(i) folgendermassen in 4 Klauseln: Sei lv eine lernende Variable von sr(i) und sei wv1 eine Widerspruchsvariable, wenn lv wahr gesetzt, und wv2 eine Widerspruchsvariable, wenn lv falsch gesetzt wird. Sei ef(c) eine Folge der Literale, die eindeutig bestimmt sind, wenn zusaetzlich c mit SETZE unerfuellt gesetzt wird. Die Literale muessen in der Reihenfolge eindeutig bestimmt sein, die die Folge angibt. Bei der Variablen, deren beide Interpretationen eindeutig bestimmt sind, wird ein Literal gewaehlt. Die 4 Klauseln sind:

   ef(sr(i)´) + lv + ef(lv) + wv1 + ef(wv1)
   ef(sr(i)´) + lv + ef(lv) + wv1´ + ef(wv1´)
   ef(sr(i)´) + lv´ + ef(lv) + wv2 + ef(wv2)
   ef(sr(i)´) + lv´ + ef(lv) + wv2´ + ef(wv2´)

   sr(i)´ entsteht dadurch, dass die Klausel sr(i) elementweise komplementiert wird. Die obigen 4 Klauseln werden, als Interpretationen aufgefasst, in einen unvollstaendigen binaeren Baum Th eingebettet. Th hat etwa folgende Gestalt:
   (Wir markieren jeden Punkt nur mit der Variablen, die zu ihm gehoert. Ein Stern bedeutet eine beliebige Variable, ein + entspricht einer unerfuellten Klausel.)



Sei s1 = SE(R[i](s),sr(i)) . Durch Anhaengen einer weiteren Kante an jeden Punkt in Th, der nicht bereits zwei Nachfolger hat, laesst sich Th in einen Aufzaehlungsbaum T(s1) transformieren. Jedem Blatt von T(s1) entspricht eine unerfuellte Klausel von s1. Dies ist eine Konsequenz der Eigenschaft "eindeutig bestimmt". Es gilt:

N(T(s1)) < 4 * (Variablenanzahl(s) - Laenge(sr(i))) * 2

3. Sei sr(i) die erste Superresolvente der Laenge 1 im Superresolutionsbeweis R(s). Wir betrachten die Folge von Aufzaehlungsbaeumen, die wir den KNF SE(R[k](s),sr(k)) = srb(k) ($1 \leq k \leq i$) zugeordnet haben. Wir wollen eine Folge von Aufzaehlungsbaeumen fuer SE(s,sr(k)) = srs(k) konstruieren. Die Baeume srb(k) verwenden naemlich teilweise Superresolventen sr(k1) (k1 < k) zum Abschliessen von Aesten. Diese Superresolventen wollen wir ersetzen.

   Zuerst ersetzen wir in allen Baeumen srb(k) ($1 \leq k \leq i$) die Superresolvente sr(1). Zum Abschliessen von Aesten in srb(1) werden nur Klauseln in s verwendet. Im Baum srb(k) wird ein Ast mit Endpunkt p, der durch sr(1) abgeschlossen ist, ergaenzt durch einen Teilbaum von srb(1). Der Teilbaum von srb(1) ist dabei definiert, wie im folgenden beschrieben wird. Sei I die bei p bestimmte Teilinterpretation, vereinigt mit den komplementierten Literalen in sr(k). Sei l ein Literal in I, dessen Variable v in srb(1) an einem inneren Punkt (d.h. nicht an einem Blatt) vorkommt. Wir betrachten den Teilbaum T von srb(1), der die Wurzel v hat.

Sei Tl(v) der linke Teilbaum von T und Tr(v) der rechte Teilbaum von T. In Tl sei v wahr interpretiert. Falls der Literal v in I ist, so sei TT = Tl(v) und sonst TT = Tr(v). srb(l) wird nun verkuerzt zu srb(l) - T + TT. Dabei wird TT bei v angehaengt und vl wird ersetzt durch die Variable, welche der Wurzel von TT entspricht oder durch eine Klausel, welche den Ast abschliesst. Diesen Prozess wiederholen wir fuer jeden Literal l in I. Wir nennen den erhaltenen Baum Red(srb(l),I).

Wir betrachten die Komplexitaet des Aufzaehlungsbaumes Red(srb(l),I). Wenn I nur einen Literal der Wurzel von srb(l) enthaelt, dann enthaelt Red(srb(l),I) keine anderen KNF als srb(l). Sei w ein Weg in srb(l) von der Wurzel zu einem Punkt. Sei v die Variable eines Punktes (aber keines Endpunkts) auf dem Weg w. Wenn w im Teilbaum Tl(v) weiterlaeuft, sei l(v) = v, sonst l(v) = v´. Sei I(w) die Menge der Literale l(v) auf dem Weg w. Auch jetzt enthaelt Red(srb(l),I(w)) keine anderen Literale als srb(l).
Fuer ein allgemeines I enthaelt Red(srb(l),I) andere KNF als srb(l). Sei v die Variable eines Punktes in srb(l), dessen beide Nachfolger Endpunkte sind. Sei I = {v}. Dann enthaelt Red(srb(l),I) auf dem Ast von der Wurzel bis zum Punkt, dem v zugeordnet war, nur KNF, die nicht in srb(l) sind.

4. Wir eliminieren nach obigem Verfahren die Superresolventen sr(k) ($1 \leq k \leq i$) und erhalten zuletzt den Aufzaehlungsbaum srs(i). Bei diesem Baum ist jeder Ast durch eine Klausel in s abgeschlossen. Welche Komplexitaet hat srs(i) ? Sei c(k) die Komplexitaet von srb(k) ($1 \leq k \leq i$). Sei cl(k) die Komplexitaet von srs(k) ($1 \leq k \leq i$). Von cl(k) kann man im allgemeinen nur sagen, dass

$$cl(k) \leq c(k) + \text{Summe}[j \text{ in } K(k)]\ cl(j),$$

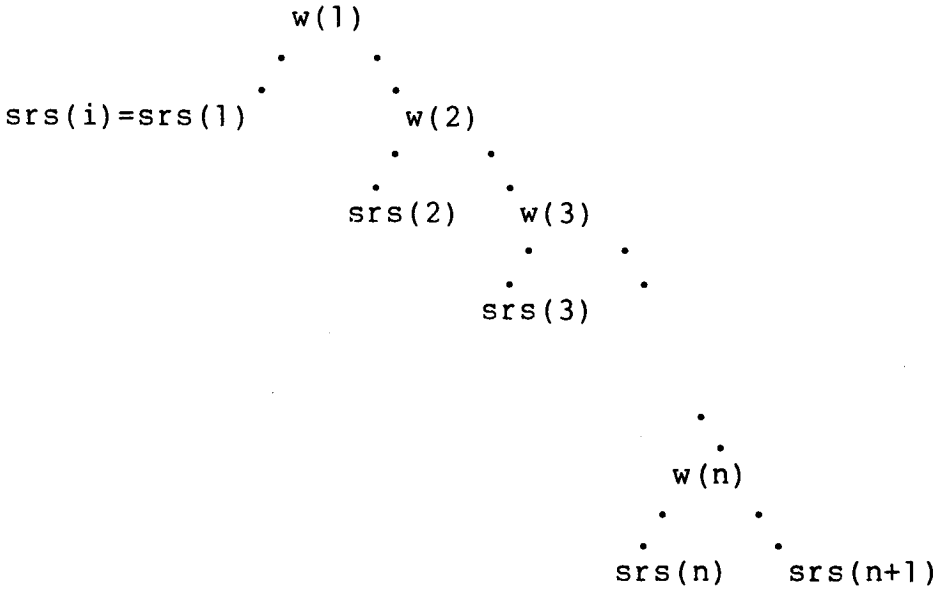
wobei K(k) eine Teilmenge der Zahlen {1, 2, ... k-1} ist. Also ist cl(i) im allgemeinen eine exponentielle Funktion in i (z.B. eine Funktion, welche der Fibonacci-Funktion aehnlich ist).

Im Fall von einfacher Superresolution ist cl(i) durch ein Polynom in i beschraenkt, denn die Baeume srb(k) verwenden bei einfacher Superresolution nach der lernenden Variablen lv (d.h. bei den Nachfolgerpunkten des Punktes, der zu lv gehoert) keine frueheren Superresolventen zum Abschliessen von Aesten. Man beachte, dass bei einfacher Superresolution alle frueheren Superresolventen erfuellt sind, falls die momentane Superresolvente unerfuellt ist. Deshalb entspricht jeder Superresolventen vor sr(i) hoechstens ein polynomialer Teilbaum von srs(i).

Wir haben somit bewiesen, dass im Fall von einfacher Super-

resolution die Komplexitaet von srs(i) beschraenkt ist durch ein Polynom in der Laenge des Superresolutionsbeweises R[i](s).

5. Wir wenden die Prozedur SETZE(R(s),sr(i),*) an und erhalten so einen Superresolutionsbeweis, der die Variable, die zum Literal in sr(i) gehoert, nicht mehr enthaelt. (Man beachte, dass sr(i) die erste Superresolvente der Laenge 1 in R(s) war.) Sei nun sr(i) wieder die erste Superresolvente im reduzierten Beweis. Wir wenden analog das oben beschriebene Verfahren an, um einen Aufzaehlungsbaum zu konstruieren, dessen Aeste nur durch Klauseln in s abgeschlossen werden. Wenn wir dieses Verfahren fuer jede Variable in s iterieren, erhalten wir folgenden Aufzaehlungsbaum TSRA(R(s)) fuer s. (w(1), w(2), ... w(n) sind die unter 1. bestimmten Wurzelvariablen.)



Im Fall von einfacher Superresolution ist die Komplexitaet $N(TSRA(R(s)) \leq p(NS(R(s)))$ fuer ein Polynom p festen Grades.

Bemerkung:

Die Rekursionsungleichung fuer cl(k) deutet an, dass die Uebersetzung von Superresolutionsbeweisen in Aufzaehlungsbaeume auf vielen Folgen exponentiell ist. Das bedeutet, dass Superresolution (resp. Resolution) auf vielen Folgen polynomial ist, wie folgende Ueberlegungen zeigen.

Sei s(1), s(2), ... eine Folge von KNF und R(s(1)),R(s(2)), ... eine entsprechende Folge von Superresolutionsbeweisen. Wir nehmen an, die Uebersetzung TSRA sei exponentiell, d.h. die Folge N(TSRA(R(s(1)))), N(TSRA(R(s(2)))), ... wachse exponentiell in der Laenge der

R(s(i)). Dies bedeutet, dass es ein c>0 gibt, so dass N(TSRA(R(s(i)))) > 2 ** (c * NS(R(s(i))))). Dann kann die Folge NS(R(s(1))), NS(R(s(2))), ... nicht exponentiell wachsen in der Laenge der s(i), d.h. Superresolution ist polynomial auf dieser Folge. Wuerde naemlich die Folge NS(R(s(1))), NS(R(s(2))), ... exponentiell wachsen, so wuerde die Folge N(TSRA(R(s(1)))), N(TSRA(R(s(2)))), ... doppelt exponentiell wachsen in der Laenge der s(i). Dies fuehrt zu einem Widerspruch, denn ein Aufzaehlungsbaum einer KNF mit n Variablen hat hoechstens 2**n Punkte. Aufgrund dieser Ueberlegungen vermute ich, dass es schwierig bis unmoeglich ist, exponentielle untere Schranken fuer Superresolution zu erhalten.

Beispiel:
Wir betrachten die KNF s, auf die wir in Abschnitt 1.2 Algorithmus SR angewendet haben.

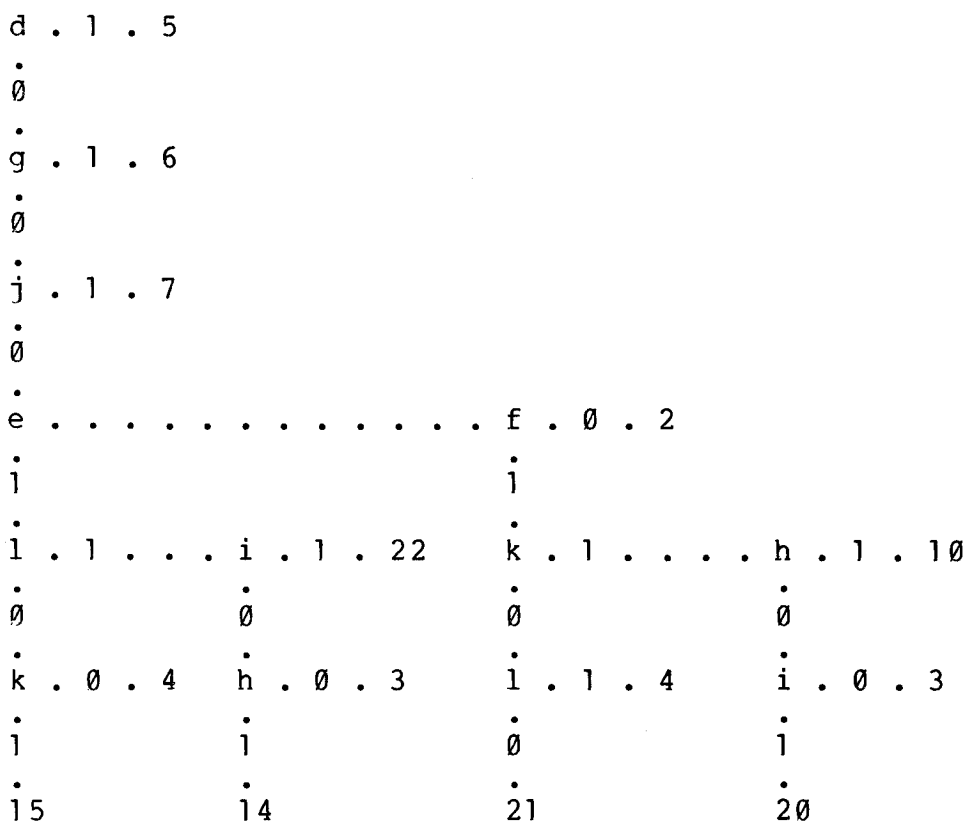
R(s) = s + {a´} + {d´} + {}
= s + sr(1) + sr(2) + sr(3)

e(1) = {a´}
e(2) = {a´,d´}
e(3) = {a´,d´}

Die Folge der Wurzelvariablen ist: a,d. Eine lernende Variable der ersten Superresolventen ist e. Dann sind 1 resp. k Widerspruchsvariablen. Der ersten Superresolventen werden die folgenden 4 Klauseln zugeordnet:

1.	d´ g´ j´	e		l	i´ h
2.	d´ g´ j´	e		l´	k
3.	d´ g´ j´	e´	f	k	h´ i
4.	d´ g´ j´	e´	f	k´	l

Diese Aufteilung in 5 Gruppen weist auf die Mengen ef(sr(i)´), lv, ef(lv), wv, ef(wv) hin. Die Klauseln werden folgendermassen in einen Aufzaehlungsbaum eingebettet (jeder Ast wird durch die Nummer der Klausel, die unerfuellt ist, abgeschlossen):



Satz 1.13.2 : Es gibt eine Folge t(1),t(2), ... von KNF und Konstanten iØ und c>Ø, so dass fuer jeden einfachen Superresolutionsbeweis R(i) fuer t(i) und fuer alle i>iØ gilt: NS(R(i)) $\geq$ 2**(c*Laenge(t(i))).

Beweis:
Wir betrachten die Folge t(1),t(2), ... von Galil. Annahme: Fuer diese Folge existiert eine nicht exponentielle Beweisfolge im Beweissystem "einfache Superresolution". Weil wir jeden einfachen Superresolutionsbeweis polynomial in einen Aufzaehlungsbaum uebersetzen koennen, existiert auch eine nicht exponentiell wachsende Folge von Aufzaehlungsbaeumen. Widerspruch.
#

Ein Beweissystem BW fuer UNERF heisst exponentiell, wenn es eine Folge s(1),s(2), ... von KNF gibt, so dass jede Beweisfolge BW(s(1)),BW(s(2)), ... exponentiell waechst. Galil hat in Satz 1.13.1 bewiesen, dass das Beweissystem "Aufzaehlungsbaum" exponentiell ist.

Korollar 1.13.3 : Einfache Superresolution ist exponentiell.

Beweis:
Verwende Satz 1.13.2. #

Ein Aufzaehlungsalgorithmus ist eine Prozedur, die bei gegebener unerfuellbarer KNF s einen Aufzaehlungsbaum fuer s konstruiert. Die Komplexitaet C[AP] der Klasse aller Aufzaehlungsalgorithmen AP ist definiert durch

$$C[AP](n) = \max_{[Laenge(s)=n]} \min(N(T)),$$

wobei das Minimum ueber alle Aufzaehlungsbaeume fuer s genommen ist. Laenge(s) ist die Anzahl Literale in s. Aus Satz 1.13.1 folgt [GAL76]:

Satz 1.13.5 : Es gibt eine Konstante c>0, so dass $C[AP] \geq 2^{**}(c*n)$, d.h. alle Aufzaehlungsalgorithmen sind exponentiell.

Wir betrachten eine Variante von SR, die ein Aufzaehlungsalgorithmus ist. Sei I(k) die Interpretation, welche im k-ten Schritt von SR konstruiert wurde. Im (k+1)-ten Schritt wird im Fall "waehle" ein Literal folgendermassen gewaehlt: Wenn es einen Literal in I(k) gibt, der nicht in I(k+1) ist, so wird er in I(k+1) aufgenommen. Sonst wird ein beliebiger Literal gewaehlt. Wir bezeichnen mit SR(AUF) den Algorithmus SR mit obigem Literalauswahlkriterium. Die erste Interpretation werde beliebig gewaehlt.

Korollar 1.13.6 : Auf unerfuellbaren KNF ist SR(AUF) exponentiell.

Beweis:
SR(AUF) erzeugt einen einfachen Superresolutionsbeweis, also direkt einen Aufzaehlungsbaum. #

Die klassischen Backtrackalgorithmen sind ebenfalls Aufzaehlungsalgorithmen und somit exponentiell im Aufwand. Sie durchlaufen den Aufzaehlungsbaum in einer festen Reihenfolge, z.B. pre-order. Wir nennen solche Aufzaehlungsalgorithmen geordnet. Sie haben den Vorteil, dass die bereits durchprobierten Interpretationen nicht explizit abgespeichert werden muessen. Ein Zaehler genuegt, um anzugeben, welche Interpretationen schon durchprobiert wurden und welche noch getestet werden muessen. Geordnete Aufzaehlungsalgorithmen haben auf den erfuellbaren KNF einen wesentlichen Nachteil: Sei B der binaere Baum der Interpretationen fuer n Variablen, und sei s eine erfuellbare KNF. Wir nehmen an, die linke Haelfte von B enthalte keine Modelle, hingegen die rechte Haelfte relativ viele. Wenn nun ein geordneter Aufzaehlungsalgorithmus auf s angesetzt wird, so

findet er sehr lange kein Modell, wenn er zuerst den linken Teilbaum durchsucht.

SR hat diesen Nachteil nicht bei geeigneter Wahl der Literale im Fall "waehle". SR kann z.B. eine Folge von Interpretationen untersuchen, wobei die mit gerader Nummer im linken Teilbaum und die mit ungerader Nummer im rechten Teilbaum sind. Deshalb hat SR bei dieser Reihenfolge eine gute Chance, eines der relativ haeufigen Modelle im rechten Teilbaum zu finden.

SR hat einen weiteren wesentlichen Vorteil gegenueber den klassischen Backtrackalgorithmen: Wie die Ueberlegungen zur Uebersetzung von Superresolutionsbeweisen in Aufzaehlungsbaeume zeigen, gibt es Varianten von SR, die keine Aufzaehlungsalgorithmen sind. Deshalb sind diese Varianten eventuell asymptotisch besser als alle Aufzaehlungsalgorithmen. Ich vermute, dass es Folgen von KNF gibt, fuer die jeder Aufzaehlungsalgorithmus exponentiellen Aufwand hat, fuer die hingegen eine polynomiale Variante von SR existiert.

SR bietet die Moeglichkeit, die Suche nach einem Modell den bisher untersuchten Interpretationen geschickt anzupassen. SR hat hingegen den Nachteil, dass die schon untersuchten Interpretationen in den Superresolventen abgespeichert werden muessen. Algorithmus SRUG leistet dies auf optimale Art. Im zweiten Teil der Arbeit werden wir die Vorteile von SR, resp. SRUG ausnuetzen.

1.14 Laenge der Superresolventen und schnelle Aufzaehlungsalgorithmen

In diesem Abschnitt zeige ich, dass die laengste Superresolvente in einem Superresolutionsbeweis fuer eine KNF s in UNERF(3) kuerzer sein kann als die laengste Resolvente im guenstigsten Resolutionsbeweis fuer s. Umgekehrt kann die laengste Resolvente in einem Resolutionsbeweis fuer eine KNF s in UNERF(3) nicht kuerzer sein als die laengste Superresolvente im guenstigsten Superresolutionsbeweis fuer s. Deshalb ist Superresolution auch in dieser Hinsicht oekonomischer als Resolution. Weiter zeige ich, dass fuer eine KNF s in UNERF(3) die Superresolventen hoechstens die Laenge 2/3*n haben muessen, wobei n die Variablenanzahl von s ist. Aus dieser Bemerkung resultiert ein O(2**(n*2/3)) Entscheidungsalgorithmus fuer ERF(3).

Fuer Resolution ist bewiesen [GA75], dass es fuer jedes k unerfuellbare KNF in UNERF(3) gibt, fuer die jeder Resolutionsbeweis mindestens Klauseln der Laenge k benoetigt. Dies gilt ohne Zweifel auch fuer Superresolution. Superresolution

benoetigt hingegen im allgemeinen kuerzere Klauseln als Resolution, wie die folgenden Ueberlegungen zeigen.

Wenn jeder Superresolutionsbeweis fuer eine KNF Klauseln mindestens der Laenge k benoetigt, so benoetigt auch jeder Resolutionsbeweis Klauseln mindestens der Laenge k. Denn wenn es einen Resolutionsbeweis mit Resolventen der Laenge k1<k gibt, so existiert auch ein Superresolutionsbeweis mit Superresolventen der Laenge k1<k. (Anwendung des Abkuerzungsalgorithmus)

Die Umkehrung hingegen ist bereits falsch fuer den Fall k=4. Es gibt KNF in UNERF(3), fuer die jeder Resolutionsbeweis mindestens eine Klausel der Laenge 4 verwendet. Hingegen existiert ein Superresolutionsbeweis mit Klauseln hoechstens der Laenge 3.

Beispiel:
Sei B(5,3) die KNF mit 5 Variablen, die alle Klauseln der Laenge 3 enthaelt, bei denen entweder alle Literale positiv oder alle Literale negativ sind. B(5,3) ist unerfuellbar, weil jede Interpretation entweder 3 Variablen wahr oder 3 Variablen falsch interpretiert.

Wir haben also ein weiteres Beurteilungskriterium gefunden, bei dem Superresolution besser abschneidet als Resolution. Nun beweisen wir eine obere Schranke fuer die Laenge der benoetigten Superresolventen fuer UNERF(3).

Wir schraenken Algorithmus SR durch folgendes Literalauswahlkriterium ein: Im Fall "waehle" ist derjenige Literal zu waehlen, der veranlasst, dass moeglichst viele Literale durch SETZE eindeutig bestimmt sind. Wir nennen diesen Algorithmus SRME.

Satz 1.14.1 : Sei s eine KNF in ERF(3) + UNERF(3) mit n Variablen, auf die SRME angewendet wird. Dann haben die durch SRME erzeugten Superresolventen hoechstens Laenge n*2/3.

Beweis:
Wir beweisen, dass von 3 Literalen in s hoechstens zwei Literale in GEWAEHLT aufgenommen werden koennen. Es werde ein Literal a gewaehlt. Wenn durch a ein Literal eindeutig bestimmt ist, so ist die Behauptung bewiesen. Wenn durch a kein Literal eindeutig bestimmt ist, dann wird in einer Klausel der Laenge drei ein Literal bei der Ausfuehrung von SETZEB(a,UEBRIG,*) markiert. Beim naechsten Durchlauf der for-Schleife V1 koennen die folgenden drei Faelle auftreten:

"waehle": Es existiert ein Literal l, durch den mindestens ein Literal eindeutig bestimmt ist. Weil das Literalauswahlkriterium stets denjenigen Literal

waehlt, der moeglichst viele andere Literale eindeutig bestimmt, ist die Behauptung bewiesen.
"unerfuellt": Kein Literal wird mehr in GEWAEHLT aufgenommen.
"isoliert": Ein Literal ist durch a eindeutig bestimmt. #

Korollar 1.14.2 : Fuer alle KNF in UNERF(3) existiert ein Superresolutionsbeweis, bei dem keine Klauseln auftreten, welche laenger als n*2/3 sind.

Die maximale Laenge der Superresolventen hat einen Einfluss auf die maximale Anzahl Interpretationen, die durchprobiert werden muessen, bevor ein Modell gefunden wird.

Satz 1.14.3 : ERF(3) ist entscheidbar mit Aufwand O(p(s)*2**(n*2/3)). Dabei ist p ein Polynom kleinen Grades, das den Aufwand fuer eine Interpretationskonstruktion angibt. s ist eine KNF in ERF(3), und n ist die Anzahl Variablen in s.

Beweis:
Wir verwenden SR und geben an, in welcher Reihenfolge die Interpretationen durchprobiert werden muessen. Die Variablen von s seien numeriert. Sei I = l(1), l(2), ... l(k) eine konstruierte Teilinterpretation, durch die noch keine Klausel unerfuellt ist. Dann wird ein Literal l der kleinsten Variablen mit folgender Eigenschaft in I aufgenommen: l bestimmt mindestens einen weiteren Literal eindeutig oder, wenn es kein solches l gibt, ist l so gewaehlt, dass l´ in einer Klausel der Laenge 3 vorkommt.

Die erste Interpretation wird entsprechend der obigen Vorschriften gewaehlt. Bei der i-ten Interpretation werden die Literale moeglichst wie bei der (i-1)-ten Interpretation gewaehlt, sonst nach obiger Vorschrift. Der so entstehende semantische Baum hat hoechstens 2**(n*2/3) Blaetter. Die Superresolventen muessen nicht abgespeichert werden, sondern jede Superresolvente bestimmt eindeutig, was noch durchprobiert werden muss.

2. Probabilistische Algorithmen fuer NP-vollstaendige Sprachen

2.1 Semientscheidbarkeit mit vorgegebener Fehlerwahrscheinlichkeit

Fuer viele Funktionen ist die exakte Berechnung mit den bisher bekannten Methoden sehr aufwendig. Das kann an mangelndem Wissen oder an der Nichtexistenz effizienter Algorithmen liegen. Auf jeden Fall ist folgender Ansatz interessant : Man verzichtet auf die Garantie der Richtigkeit der Resultate, wenn die Wahrscheinlichkeit w, dass das Resultat einen grossen Fehler aufweist, sehr klein ist. Dadurch kann sich der Rechenaufwand wesentlich reduzieren.

Im folgenden interessieren wir uns fuer Entscheidungsprobleme. Es gibt die folgenden zwei Moeglichkeiten, bei Entscheidungsalgorithmen ein Wahrscheinlichkeitsmodell anzusetzen.

1. Verteilung auf Eingabe-Daten [KAR76]
Man nimmt auf den Eingabe-Daten eine geeignete Verteilung an und zeigt, dass "fast ueberall " ein gegebener Algorithmus die Entscheidung richtig faellt. Bei der Anwendung solcher Methoden liegt die Hauptschwierigkeit darin, zu beurteilen, ob die gewaehlte Verteilung fuer den Anwendungsbereich realistisch ist. In [POS76] wird bei einer gewissen Verteilung auf den Graphen ein polynomialer Algorithmus angegeben, der "fast ueberall" einen Hamilton-Kreis findet, falls einer existiert. Dies ist bemerkenswert wegen der NP-Vollstaendigkeit des Hamilton-Kreis-Problems.

2. Ausnuetzen des Einseitigkeitseffekts
Sei L eine entscheidbare Sprache ueber {0,1}*. Fuer viele Sprachen L (z.B. einige in NP) gibt es Beweissysteme BW(L), deren asymptotische Komplexitaet schwaecher waechst als die aller bisher bekannten Beweissysteme fuer das Komplement von L. Deshalb ist die Wahrscheinlichkeit unter Umstaenden erheblich groesser, fuer ein x in L den Beweis zu finden, dass x in L ist (z.B. fuer eine KNF ein Modell anzugeben), als fuer ein y, das etwa gleich lang wie x ist, den Beweis zu finden, dass y nicht in L ist (z.B fuer eine KNF die Unerfuellbarkeit zu beweisen). (Siehe auch Einleitung zu Abschnitt 1.8.) Zudem gibt es meistens viele Beweise, dass x in L ist, was die Wahrscheinlichkeit, dass man einen solchen Beweis findet, weiter vergroessert.

Man sucht Entscheidungsverfahren mit moeglichst kleiner Irrtumswahrscheinlichkeit w und mit moeglichst kleinem Aufwand T

und definiert deshalb: Eine Sprache L heisst (w,T)-semientscheidbar (Ø < w < 1 und T : N -> N), wenn es eine DZM M und ein Programm p fuer M mit folgenden Eigenschaften gibt:

1. p verwendet eine Zufallsvariable Z (Gleichverteilung auf einer endlichen Menge) als Orakel.
2. Die asymptotische Zeitkomplexitaet von p ist O(T(n)), wobei das Abfragen des Orakels einen Schritt benoetigt.
3. Wenn p entscheidet, dass x in L ist, so ist dieser Entscheid richtig.
4. Wenn p entscheidet, dass x nicht in L ist, so ist dieser Entscheid falsch mit der Wahrscheinlichkeit fw(T(n)) < w. Dabei soll die Fehlerwahrscheinlichkeit fw wie folgt bestimmt sein: Jedem x in {Ø,1}* ordnet man eine endliche Menge V(x) und eine Eigenschaft P[x] auf V(x) so zu, dass gilt:
   a) Falls x nicht in L ist, so hat kein Element in V(x) die Eigenschaft P[x].
   b) Falls x in L ist, so gilt fuer einen Prozentsatz q(x) der Elemente y in V(x) die Eigenschaft P[x](y) nicht.

   Nun prueft man das Praedikat "x in L", indem man von einer zufaelligen Auswahl einiger y aus V(x) den Wahrheitswert P[x](y) bestimmt. fw ist die Fehlerwahrscheinlichkeit dieses Tests.

   Sei x in L. Wenn mit Hilfe der Zufallsvariabeln Z ein Element y in V(x) erzeugt wird, so ist die Wahrscheinlichkeit q(x), dass P[x](y) nicht gilt. Nach der Erzeugung von k unabhaengigen Zufallswerten ist die Wahrscheinlichkeit q(x)**k, dass kein erzeugtes Element die Eigenschaft P[x] hat. Anders ausgedrueckt : Die Wahrscheinlichkeit, dass mindestens ein Element mit Eigenschaft P[x] erzeugt wurde, ist 1-(q(x)**k).

Der Algorithmus von Solovay und Strassen [SOL76,RAB76], der eine natuerliche Zahl auf Primalitaet testet, faellt in obiges Definitionsschema, d.h. die Primzahlen sind (2**(-m),6*m*n) - semientscheidbar fuer jedes m $\geq$ 1 (n ist die Laenge im Binaersystem). Dieser Ansatz ist besonders dann interessant, wenn die Fehlerwahrscheinlichkeit des Algorithmus kleiner ist als die Wahrscheinlichkeit einer Fehlfunktion der Maschine, auf der das Programm ausgefuehrt wird. (Der bisher beste bekannte deterministische Algorithmus fuer Primalitaet hat Aufwand O(2**(n*1/7)) [MIL75].)

Die Ueberlegenheit dieser Methode gegenueber einer Schlimmster-Fall-Analyse ist offensichtlich, wenn die Mengen V(x) viele Elemente enthalten. Wir nehmen an, dass card (V(x)) = 2 ** (Laenge von x) und dass qØ < q(x) < Ø.5. Dann muessen fuer ein x in L bei linearer Suche durch ganz V(x) (anstelle der Pruefung nur eines einzelnen Wertes) im schlimmsten Fall card (V(x)) * qØ Elemente durchprobiert werden, was exponentiellen Aufwand bedeutet. Hingegen ist L schon fuer langsam wachsendes T (w,T)-semientscheidbar mit kleinem w. Sei naemlich p1 ein Polynom, das den Aufwand fuer das Testen der Ei-

genschaft P[x] angibt. Dann ist der Aufwand T[w](x), den man benoetigt, um die Fehlerwahrscheinlichkeit hoechstens w zu erreichen, beschraenkt durch p1(Laenge(x))*log[q](w). Dabei ist log[q](w) der Logarithmus von w zur Basis q.

In der Definition der (w,T)-Semientscheidbarkeit wird ein einfacher statistischer Test dazu verwendet, die Hypothese zu stuetzen, dass x nicht in L ist. Die Statistik und die Wahrscheinlichkeitstheorie stellen viele solche Tests zur Verfuegung, welche vielleicht dazu verwendet werden koennen, interessante (w,T)-Semientscheidbarkeitsresultate zu erhalten.

2.2 Literalauswahlkriterien und Semientscheidbarkeit

In diesem Abschnitt werden Literalauswahlkriterien und ihre Erfolgswahrscheinlichkeiten definiert und deren Zusammenhang mit der Wahrscheinlichkeit, durch iterierte Anwendung eines Literalauswahlkriteriums ein Modell zu finden, untersucht (Satz 2.2.1).

Sei Q eine NP-vollstaendige Teilsprache von ERF. Dann bezeichnen wir mit R(Q) die folgende Menge von KNF:

a) Die KNF in Q sind in R(Q).
b) Sei q eine KNF in R(Q), und sei l ein Literal irgendeines Modells von q. (Man beachte, dass eine Interpretation als eine Menge von Literalen aufgefasst wird.) Dann ist auch diejenige KNF in R(Q), die durch Anwendung von SETZE(l,q,*) entsteht.
c) Es gibt keine anderen KNF in R(Q).

R(Q) enthaelt also alle KNF in Q und alle erfuellbaren KNF, die man durch iterierte Anwendung von SETZE auf KNF in Q erhaelt.

Ein Literalauswahlkriterium L mit Erfolgswahrscheinlichkeit 1 fuer eine Menge Q1 von KNF ist ein Algorithmus mit folgender Eigenschaft: Sei q eine KNF in Q1. Falls q erfuellbar ist, waehlt L in q einen Literal aus, der in einem Modell von q ist. Falls q unerfuellbar ist, waehlt L irgendeinen Literal.

Wenn ein polynomiales Literalauswahlkriterium mit Erfolgswahrscheinlichkeit 1 fuer eine Menge R(Q) existiert, wobei Q eine NP-vollstaendige Menge ist, so ist P=NP. Deshalb ist es zweckmaessig, nach geeigneten Q und Literalauswahlkriterien fuer R(Q) zu suchen, welche polynomialen Aufwand haben und das richtige Resultat mindestens mit einer bestimmten bekannten Erfolgswahrscheinlichkeit liefern.

Wir betrachten Literalauswahlkriterien L folgender Art: Sei q eine KNF in R(Q). Dann waehlt L eine gewisse Multiteilmenge T[L](q) der Literale von q und, falls T[L](q) nicht leer ist, waehlt L in dieser Multimenge zufaellig mit Gleichverteilung einen Literal aus. Wir sagen, dass L die Erfolgswahrscheinlichkeit x hat, falls es ein Modell M von q gibt, das in T[L](q) mindestens den Prozentsatz x der Literale erfuellt. Wir verlangen, dass die Teilmenge der KNF q in R(Q) mit T[L](q)={} polynomial entscheidbar und die Berechnung von T[L](q) polynomial in der Laenge von q ist.

Wir koennen die Erfolgswahrscheinlichkeit x auch folgendermassen formulieren: Sei s eine Multimenge von Literalen und I eine Interpretation der Variablen in s. Sei r die Anzahl erfuellter und b die Anzahl unerfuellter Literale in s. Wir nennen r/(r+b) den Erfuelltheitsgrad von s bei der Interpretation I (Abkuerzung eg(s,I)). Dann hat das Literalauswahlkriterium Erfolgswahrscheinlichkeit x, gdw. es ein Modell M fuer q gibt, fuer das gilt: eg(T[L](q),M) $\geq$ x.

Die Menge der KNF q in R(Q) mit T[L](q)#{} nennen wir durch L erreichbar. Wir bezeichnen diese Teilmenge von R(Q) mit R[L](Q).

Sei Q eine NP-vollstaendige Teilsprache von ERF, und wir nehmen an, wir haetten ein Literalauswahlkriterium fuer R(Q) mit Erfolgswahrscheinlichkeit x. Wir untersuchen, wie sich die Erfolgswahrscheinlichkeit des Literalauswahlkriteriums auswirkt auf die Wahrscheinlichkeit, durch iterierte Anwendung des Literalauswahlkriteriums ein Modell zu finden.

Sei q eine KNF in R[L](Q) mit n Variablen. Da die Erfolgswahrscheinlichkeit fuer die richtige Interpretation einer Variablen mindestens x ist, so ist die Erfolgswahrscheinlichkeit, nach n Anwendungen ein Modell gefunden zu haben, mindestens x**n. Also ist die Wahrscheinlichkeit, nach einer Interpretationskonstruktion kein Modell gefunden zu haben, hoechstens 1 - x**n. Nach m Interpretationskonstruktionen ist die Wahrscheinlichkeit, dass kein Modell gefunden wurde, hoechstens gleich (1 - x**n)**m. Wir nennen diese Wahrscheinlichkeit die Fehlerwahrscheinlichkeit f(m) nach m Schritten. Wie gross muss bei gegebenem q und n die Zahl m sein, damit f(m) kleiner als eine vorgegebene Konstante eps ist ? m(q,n,eps) soll also $\geq$ h sein, wobei

$$h = \ln(eps) \;/\; \ln(1-x^{**}n).$$

Wir verwenden folgende Ungleichung, um m abzuschaetzen:

$$x/(1+x) < \ln(1+x) < x,$$

wobei x > -1 und x # 0. Dann folgt:

ln(1/eps)*(1-x**n)/x**n > h > ln(1/eps)/x**n

Deshalb ist m in O(ln(eps) * (1/x)**n). Also haben wir folgenden Satz bewiesen:

Satz 2.2.1 : Wenn fuer eine NP-vollstaendige Sprache Q ein Literalauswahlkriterium L mit Erfolgswahrscheinlichkeit x fuer R(Q) existiert, dann existiert ein probabilistischer Algorithmus PR fuer Q mit folgender Eigenschaft: PR benoetigt O((1/x)**n) Interpretationskonstruktionen, damit PR eine beliebig kleine Fehlerwahrscheinlichkeit erreicht.

Wir koennen das Resultat auch folgendermassen exakter formulieren: Eine NP-vollstaendige Sprache Q mit einem polynomialen Literalauswahlkriterium L fuer R(Q) mit Erfolgswahrscheinlichkeit x ist

(eps,(p1(l(q))*n(q)+p2(l(r(q))))*(1/x)**n(q))-semientscheidbar

fuer jedes eps>0. Dabei verwenden wir folgende Bezeichnungen:
q: Element in R[L](q)
n: Variablenanzahl von q
l(q): Laenge von q
n(q): Anzahl Variablen, die im schlimmsten Fall gewaehlt werden muessen. (n(q) $\leq$ n)
r(q): Laengste KNF, die im schlimmsten Fall uebrig bleibt, wenn T[L](q) leer ist. (l(r(q)) $\leq$ l(q))
p1: Polynom, das den Aufwand des Literalauswahlkriteriums angibt.
p2: Polynom, das den Aufwand der Entscheidung fuer q mit T[L](q)={} angibt.

2.3 Der Erfuelltheitsgrad NP-vollstaendiger Sprachen

Sei Q eine NP-vollstaendige Teilsprache von ERF. Der Erfuelltheitsgrad der KNF in R(Q) hat unmittelbar einen Einfluss auf die asymptotische Zeitkomplexitaet der probabilistischen Algorithmen fuer Q, wie Satz 2.2.1 zeigt. Zuerst untersuche ich in diesem Abschnitt, wie gross der Erfuelltheitsgrad auf NP-vollstaendigen Mengen Q (d.h. nicht auf den ganzen Mengen R(Q)) werden kann. Neben dem Erfuelltheitsgrad interessieren uns aber auch noch einige andere Funktionen [LIE76].

Sei s eine KNF, c eine Klausel von s und I eine Interpretation der Variablen in s. Wir definieren:
eg(s,I) = (Anzahl Literale, die bei der Interpretation I in s erfuellt sind) / (Anzahl Literale in s). (Erfuellt-

heitsgrad)
deg(s,I) = (Summe aller eg(c,I) fuer alle Klauseln c in s) / (Anzahl Klauseln in s). (durchschnittlicher Erfuelltheitsgrad)
pnmin(s,I) = (Anzahl Klauseln in s mit eg(c,I) > (1 / Laenge(c))) / (Anzahl Klauseln in s). (Prozentsatz Klauseln mit nicht minimalem Erfuelltheitsgrad)
gew(c) = 2 ** (-Anzahl Literale in c). (Gewicht)
geg(s,I) = (Summe der Gewichte aller Literale in s, die bei Interpretation I erfuellt sind) / (Summe der Gewichte aller Literale in s). Dabei ist das Gewicht eines Literals das Gewicht der Klausel, die den Literal enthaelt. (gewichteter Erfuelltheitsgrad)

Der hier verwendete Begriff "Gewicht einer Klausel" wurde in [JO73] eingefuehrt und dort und in [LIE75] bei der Untersuchung von Approximationsalgorithmen fuer ERF benutzt.

Sei NPQ die Klasse der NP-vollstaendigen Teilsprachen von ERF.

Lemma 2.3.1 : Fuer alle d in [Ø,1) kann effektiv ein Q in NPQ angegeben werden, so dass fuer jede erfuellbare KNF q in Q und fuer jedes Modell M von q gilt :
1. eg(q,M) > d
2. deg(q,M) > d
3. pnmin(q,M) > d
4. geg(q,M) > d.

Beweis :
Beweisidee : Wir uebersetzen eine Klasse von NP-vollstaendigen Sprachen, die von einem Parameter i abhaengen, nach Erfuellbarkeit und zeigen, dass bei wachsendem i fuer alle Modelle eg, deg, pnmin und geg nach 1 konvergieren.

Bemerkung: Der Satz ist trivial und laesst sich auch durch Einfuehrung von zusaetzlichen Variablen und Klauseln beweisen.

Im folgenden definieren wir die Sprache GRAPHFAERB(i):

Elemente : ungerichtete Graphen G(N,A), wobei N die Menge der Punkte und A die Menge der Kanten ist.
Eigenschaft : Es gibt eine Faerbung mit i Farben.

Man kann zeigen, dass GRAPHFAERB(i) fuer alle i > 2 NP-vollstaendig ist (siehe z.B. [SP75]). Folgende Methode dient dazu, dieses Problem polynomial nach ERF zu transformieren. Fuer jede Kante von G fuehren wir folgende KNF s(i) ein : Fuer jede Farbe der beiden Endpunkte verwenden wir eine Variable. Wir

druecken fuer jeden Endpunkt aus, dass er genau mit einer Farbe gefaerbt sein muss. Das ist moeglich mit ncm = 2 * binomial(i,2) Klauseln cm(1), cm(2), ... cm(ncm) der Laenge 2 (hoechstens eine Farbe) und 2 Klauseln cl(1), cl(2) der Laenge i (mindestens eine Farbe). Mit i Klauseln cs(1), cs(2), ... cs(i) druecken wir aus, dass die beiden Endpunkte nicht dieselbe Farbe haben koennen.

Beispiel:
s(3) =

							Anzahl erfuellter Literale bei I
1.	a	b	c				1
2.				d	e	f	1
3.	a´	b´					1
4.	a´		c´				1
5.		b´	c´				2
6.				d´	e´		1
7.				d´		f´	2
8.					e´	f´	1
9.	a´			d´			1
10.		b´			e´		1
11.			c´			f´	2
	1	Ø	Ø	Ø	1	Ø	(Interpretation I)

Fuer jedes Modell M der KNF s(i) gilt : Die Anzahl erfuellter Literale ist 2 * binomial(i-1,2) * 2 + 2 * (i-1) + 4 + (i-2) * 2.

Die Anzahl Literale ist : 2 * binomial(i,2) * 2 + 4 * i. Also gilt :

lim [i -> unendlich] eg(s(i),M) = 1

fuer alle Modelle M von s(i).

Fuer jedes Modell M der KNF s(i) gilt : Genau 2 * binomial(i-1,2) der Klauseln cm(1), cm(2), ... cm(ncm) haben Erfuelltheitsgrad 1. Genau 2 * (i-1) haben Erfuelltheitsgrad 1/2. Die Klauseln cl(1) und cl(2) haben Erfuelltheitsgrad 1/i. Zwei der Klauseln cs(1), cs(2), ... cs(i) haben Erfuelltheitsgrad 1/2, und i-2 haben Erfuelltheitsgrad 1. Also ist die Summe der Erfuelltheitsgrade aller Klauseln von s(i) gleich 2/i + (binomial(i-1,2) + (i-1) * 1/2) * 2 + 2 * 1/2 + i-2. Die Anzahl Klauseln ist m(s(i)) = 2 + 2 * binomial(i,2) + i. Also gilt :

lim [i -> unendlich] deg(s(i),M) = 1

fuer alle Modelle M von s(i).

Fuer jedes Modell M von s(i) gilt : Das Gewicht der Literale in s(i) ist : i * 2**(-i) * 2 + binomial(i,2) + i * 1/2. Das Gewicht der erfuellten Literale ist :

```
2**(-i) * 2 + ((i-1) * 1/4 +
binomial(i-1,2) * 1/2) * 2 +
   2 * 1/4 + (i-2) * 1/2.
```

Also gilt :

```
lim [i -> unendlich] geg(s(i),M) = 1
```

fuer alle Modelle M von s(i).

Fuer jedes Modell M von s(i) gilt : 2 * binomial(i-1,2) der Klauseln cm(1), cm(2), ... cm(ncm) haben nicht minimalen Erfuelltheitsgrad, und i-2 der Klauseln c(1), cs(2), ... cs(i) haben nicht minimalen Erfuelltheitsgrad. Also gilt :

```
lim [i -> unendlich] pnmin(s(i),M) = 1
```

fuer alle Modelle M von s(i).

Sei d gegeben. Berechne i, so dass deg (s(i),M) > d, eg (s(i),M) > d, pnmin (s(i),M) > d und geg(s(i)M) > d fuer alle Modelle M von s(i).
Nun besteht Q aus allen KNF, die aus KNF s[1](i), s[2](i), ... s[n](i) (n beliebig) zusammengesetzt sind. Ein Element q in Q ist erfuellbar genau dann, wenn der entsprechende Graph mit i Farben faerbbar ist. q kann Klauseln enthalten, die mehrfach vorkommen.
#

Es gibt also NP-vollstaendige Teilmengen Q von ERF, fuer deren Elemente q gilt: Fuer alle Modelle M von q ist der Erfuelltheitsgrad eg(q,M) beliebig nahe 1. Leider ist diese Eigenschaft nicht erblich bei der Anwendung von SETZE auf die KNF von Q.

Im folgenden betrachten wir nicht nur die KNF in Q, sondern auch diejenigen erfuellbaren KNF, die man aus Elementen von Q durch Interpretation von Variablen mit SETZE erhaelt.

Sei Q in NPQ. R(Q) enthaelt alle KNF in Q und alle erfuellbaren KNF, die man durch iterierte Anwendung von SETZE auf KNF in Q erhaelt. Nun interessiert folgende Frage : Welches ist das groesste d1 und welches das groesste d2 in [0,1], fuer die effektiv ein Q in NPQ angegeben werden kann, so dass fuer jede erfuellbare KNF q in R(Q) ein Modell M von q existiert, so dass gilt:
1. eg (q,M) > d1
2. geg (q,M) > d2 ?

Aus der Analyse der NP-vollstaendigen Sprachen Q, welche zu den i-faerbbaren Graphen gehoeren, kann nur geschlossen werden: d1 $\geq$ 1/2 und d2 $\geq$ 1/2, denn die folgende KNF s ist fuer alle i > 2 in R(Q).

1. a b
2. d e
3. a´ d´
4. b´ e´
5. a´ b´
6. d´ e´

Jedes Modell von s hat Erfuelltheitsgrad 1/2. Dasselbe gilt fuer den gewichteten Erfuelltheitsgrad, weil alle Klauseln gleich lang sind. Fuer alle andern KNF in R(Q) hat jedes Modell mindestens (gewichteten) Erfuelltheitsgrad 1/2.

Fuer andere, naheliegende NP-vollstaendige Teilsprachen von ERF ist die Situation aehnlich. Deshalb untersuchen wir, wie polynomial berechenbare Funktionen die Situation verbessern koennen.

Sei Q in NPQ, und sei Pl die Menge der polynomial berechenbaren Funktionen R(Q) -> ERF, so dass gilt:
1. Fuer jedes pl in Pl ist die Menge der Formeln q in R(Q) mit pl(q) = leer in P.
2. Fuer jedes pl in Pl und jedes q in R(Q) entsteht pl(q) durch Streichen von Literalen in q.

Definition 2.3.1 : Eine Sprache in NPQ hat (gewichteten) Erfuelltheitsgrad x, wenn es ein pl in Pl gibt mit folgender Eigenschaft: Fuer jede KNF q in R(Q) mit pl(q)#{} gibt es ein Modell M von q mit:

eg(pl(q),M) $\geq$ x (geg(pl(q),M) $\geq$ x).

Falls P=NP, so gibt es trivialerweise NP-vollstaendige Sprachen mit Erfuelltheitsgrad 1. Was koennen wir ueber den Erfuelltheitsgrad NP-vollstaendiger Sprachen sagen, ohne zu wissen, ob P=NP ?

Satz 2.3.2 : Es gibt NP-vollstaendige Sprachen mit Erfuelltheitsgrad 2/3.

Beweis :
Sei Q folgende NP-vollstaendige Teilsprache von ERF(3), die durch polynomiale Uebersetzung von ERF(3) erhalten wird. Sei s eine KNF in ERF(3). Fuer jede Klausel {a,b,c} der Laenge 3 fuehren wir folgende KNF t({a,b,c}) ein, die zu {a,b,c} aequi-

valent ist:

1.			c	d	
2.		b			e
3.	a				f
4.			d	e	
5.			d		f
6.				e	f
7.	a	b		d´	
8.	a´			d	
9.		b´		d	
10.	a		c		e´
11.	a´				e
12.			c´		e
13.		b	c		f´
14.		b´			f
15.			c´		f

d (e,f) ist eine Hilfsvariable fuer a v b (a v c, b v c). Fuer jede Klausel {a,b,c} wird i.a. ein neues Trippel (d,e,f) von Hilfsvariablen eingefuehrt. Wir verlangen jedoch, dass fuer kein Literalpaar zweimal eine Hilfsvariable eingefuehrt wird, sondern dass alte Klauseln mehrfach verwendet werden.

Sei q eine KNF in R(Q). pl(q) entsteht aus q wie folgt:
1. Alle Klauseln der Laenge 2 in q, welche nicht zu einer Klauselmenge t gehoeren, werden weggelassen.
2. Von jeder Klauselmenge t werden nur die Klauseln 4 bis 6 beruecksichtigt.
3. Von jeder Klauselmenge t, bei der bereits eine oder mehrere Variablen interpretiert sind, werden alle Klauseln weggelassen.

pl ist polynomial berechenbar. (Wir setzen voraus, dass fuer jedes q in Q die Menge t({a,b,c}) direkt nach der Klausel {a,b,c} folgt.). Falls pl(q) leer ist, wird das Modell von SR im ersten Schritt gefunden, weil dann q im wesentlichen nur noch Klauseln der Laenge 2 enthaelt.

Fuer jedes q in R(Q) mit pl(q) nicht leer und fuer jedes Modell M von q ist der Erfuelltheitsgrad $eg(pl(q),M) \geq 2/3$ und auch der gewichtete Erfuelltheitsgrad $geg(pl(q),M) \geq 2/3$. Man kann leicht nachpruefen, dass jedes Modell von t in den Klauseln 4 bis 6 mindestens 2/3 der Literale erfuellt.

2.4 Lokale Analyse von Resolution

Die gefundene NP-vollstaendige Sprache Q mit Erfuelltheitsgrad 2/3 ermoeglicht nach Satz 2.2.1 die Entscheidung mit beliebig kleiner Irrtumswahrscheinlichkeit in

$$O((3/2)^{**}n) = O(2^{**}(\emptyset.58*n))$$

Schritten. Leider ist dieses Resultat fuer praktische Anwendungen uninteressant:

1. Bei der Uebersetzung nach ERF(3) werden Hilfsvariablen benoetigt. Deshalb kann ein trivialer Aufzaehlungsalgorithmus fuer die unuebersetzte KNF besser sein als ein probabilistischer Algorithmus fuer die uebersetzte KNF.
2. Der Erfuelltheitsgrad 2/3 ist zu klein, als dass interessante probabilistische Algorithmen erhalten wuerden. Wie H.R. Thomann bemerkt, wuerde man polynomiale Algorithmen fuer eg $\geq 1 - O(\log(n)/n)$ erhalten.

Damit wir den Erfuelltheitsgrad erhoehen koennen, analysieren wir in diesem Abschnitt das Verhalten von Resolution in bezug auf die Fortpflanzung von Modellinformation. Wir wollen eine gegebene erfuellbare KNF durch Resolution in eine KNF transformieren, bei der eine Multiteilmenge ihrer Literale einen hohen Erfuelltheitsgrad hat.

Zuerst untersuchen wir das Mittelwertverhalten von verschiedenen Beweissystemen, damit wir einen Hinweis bekommen, welches System besonders interessant ist.

Seien c1, c2 zwei Klauseln, die l1, resp. l2 Literale enthalten und sei c3 die Resolvente von c1 und c2. c3 habe Laenge l3. Sei M ein Modell von c1 und c2 (somit auch von c3), und M erfuelle e1 Literale in c1 und e2 Literale in c2. In c3 seien e3 Literale erfuellt.

Wir setzen voraus, dass $l1 \leq k$ und $l2 \leq k$, wobei k eine Konstante ist. Welche 6-Tupel (l1,e1,l2,e2,l3,e3) sind dann moeglich bei festem k ? Die Zahlen l1,e1,l2,e2,l3,e3 muessen folgenden Ungleichungen genuegen, damit sie zu drei Klauseln c1, c2 und c3 und einem Modell M, das e1,e2 und e3 bestimmt, gehoeren koennen:

1. $l1 \leq k$ und $l2 \leq k$ und $l3 \leq l1 + l2 - 2$.
2. $e1 \leq l1$ und $e2 \leq l2$ und $e3 \leq l3$ und $e3 \leq e1 + e2 - 1$.

3. Das Praedikat Test(l1,e1,l2,e2,l3,e3) sei wie folgt definiert :
Setze g = l1 + l2 - l3 und g1 = e1 + e2 - e3. Test ist wahr, wenn g1 $\leq$ e1 und g1 $\leq$ e2 und g1 $\leq$ g und e1 - (l1-g) $\leq$ g1 und e2 - (l2-g) $\leq$ g1 gilt und sonst ist Test falsch.
Es muss mindestens einer der folgenden Terme wahr sein :
Test(l1-1,l2-1,l3,e1-1,e2,e3),
Test(l1-1,l2-1,l3,e1,e2-1,e3).

Eine Schlussregel der (k,M)-Resolution ist ein Tripel (c1,c2,c3) von Klauseln, wobei c3 die Resolvente von c1 und c2 ist. M ist irgendein festes Modell der KNF, auf welche die Schlussregeln angewendet werden. Es muss gelten:
1. l1 $\geq$ 2 und l2 $\geq$ 2 und l3 $\geq$ 2.
2. l1 $\leq$ k und l2 $\leq$ k.

Das 6-Tupel (l1,e1,l2,e2,l3,e3) nennen wir den Genotyp der Schlussregel (c1,c2,c3). Er genuegt den Bedingungen 1. bis 3. Eine Schlussregel der (k,M)-Resolution heisst Schlussregel der beschraenkten (k,M)-Resolution, falls l3 $\leq$ k.

Sei (c1,c2,c3) eine Schlussregel der (k,M)-Resolution. Sei f eine Funktion, die jeder Klausel c und jeder Interpretation I von c eine rationale Zahl f(c,I) zuordnet. Die Verbesserung v(f,r1) von f bei der Schlussregel (c1,c2,c3) ist f(c3,M) - f(c1 + c2,M).

Sei gew eine Funktion, die jedem Genotyp r1 ein Gewicht gew(r1) zuordnet. Sei R[k] die Menge aller wesentlich verschiedenen Genotypen (l1,e1,l2,e2,l3,e3), die zu Schlussregeln der (k,M)-Resolution gehoeren, und sei G die Summe der Gewichte aller Genotypen in R[k]. Dabei heissen zwei Genotypen nicht wesentlich verschieden, falls

$$(l1,e1,l2,e2,l3,e3) = (l2,e2,l1,e1,l3,e3).$$

Der mit gew gewichtete Erwartungswert E[f,gew](k) der Verbesserungen von f ueber alle Elemente in R[k] ist definiert durch:

$$E[f,gew](k) = (\text{Summe } [r1 \text{ in } R[k]]\ v(f,r1) * gew(r1)) / G$$

Wir betrachten die folgenden Gewichtungen:
gew1: Alle Genotypen haben dasselbe Gewicht.
gew2: Jeder Genotyp hat das Gewicht: 1/(Prozentsatz der erfuellten Literale der Elternklauseln).
gew3: Jeder Genotyp hat das Gewicht: Prozentsatz der erfuellten Literale der Elternklauseln.

gew2 benachteiligt die Schlussregeln, deren Elternklauseln einen hohen Erfuelltheitsgrad haben, waehrend gew3 diese Schlussregeln bevorzugt. Wir berechnen die durchschnittlichen Verbesserungen fuer zwei f, naemlich den Erfuelltheitsgrad und

den gewichteten Erfuelltheitsgrad(geg). Den gewichteten Erfuelltheitsgrad untersuchen wir, weil er haeufig groesser ist als der Erfuelltheitsgrad. Die erhaltenen Resultate bestaetigen diese Tendenz.

Die Tabellen im Anhang 1 geben Auskunft ueber die Abhaengigkeit von E[f,gew](k) von seinen Parametern. Sie zeigen, dass bei allen Gewichtungen die (3,M)-Resolution in bezug auf die durchschnittliche Fortpflanzung von Modellinformation der (k,M)-Resolution fuer k>3 ueberlegen ist. Dasselbe gilt fuer beschraenkte (3,M)-, bzw. beschraenkte (k,M)-Resolution. Weiter ist die beschraenkte (3,M)-Resolution besser als die gewoehnliche (3,M)-Resolution. Dies gilt sowohl fuer den Erfuelltheitsgrad wie fuer den gewichteten Erfuelltheitsgrad. Deshalb untersuchen wir im folgenden die beschraenkte (3,M)-Resolution genauer. Wir bezeichnen sie mit BM3. Bei der pessimistischen Gewichtung gew2 waechst bei BM3 der Erfuelltheitsgrad bei einer Anwendung einer Schlussregel durchschnittlich um 1/2Ø (bei der optimistischen Gewichtung gew3 um 1/1Ø).

Zuerst muessen wir begruenden, weshalb es sinnvoll ist, ein stark reduziertes Beweissystem wie BM3 zu betrachten. Deshalb zeigen wir, dass das zu BM3 gehoerende Schlussregelsystem im wesentlichen vollstaendig ist. Damit ist garantiert, dass die Regeln in BM3 haeufig anwendbar sind. Eine Schlussregel der (k)-Resolution ist eine Schlussregel der (k,M)-Resolution, wobei das Modell M nicht beruecksichtigt wird. Analog ist die beschraenkte (k)-Resolution definiert.

Bemerkung:
Im Gegensatz zur beschraenkten (k,M)-Resolution wird bei der beschraenkten (k)-Resolution nur die aeussere Form der Schlussregeln betrachtet. Die folgenden Schlussregeln gehoeren zur beschraenkten (3)-Resolution :

(I)
```
A  B
A´    C
-------
   B  C
```

(II)
```
A  B  C
A´ B     D
----------
   B  C  D
```

(III)
```
A  B  C
A´ B  C
-------
   B  C
```

(IV)
```
A  B  C
A´       D
----------
   B  C  D
```

(V)
```
A  B  C
A´ B
-------
   B  C
```

Die beschraenkte (3)-Resolution hat die angenehme Eigenschaft, dass gegenueber der (3)-Resolution nur eine Schlussregel fehlt:

```
A  B  C
A´       D  E
-------------
   B  C  D  E
```

Mit EB3 bezeichnen wir das mit der Erweiterungsregel ergaenzte Beweissystem beschraenkte (3)-Resolution.

Wir nennen ein Beweissystem B vollstaendig in bezug auf Superresolution, wenn jede unerfuellbare KNF mit Hilfe von B und SETZE in eine gesaettigte KNF transformiert werden kann. Dabei heisst eine unerfuellbare KNF gesaettigt, wenn die leere Klausel eine Superresolvente ist. SETZE darf angewendet werden, wenn ein Literal durch SETZE eindeutig bestimmt ist.

Lemma 2.4.1 : EB3 ist vollstaendig in bezug auf Superresolution.

Beweis:
Jeder Resolutionsbeweis kann durch Einfuehrung von Hilfsvariablen so umgeformt werden, dass jede Klausel im Beweis hoechstens Laenge 3 hat. Wenn eine Klausel der Laenge 1 erzeugt wird, ist sie durch SETZE eindeutig bestimmt. Die Schlussregel

(VI)
```
A  B  C
A´       D  E
-------------
   B  C  D  E
```

kann z.B folgendermassen mit Schlussregeln von EB3 simuliert werden: Wir fuehren eine Hilfsvariable Z = C + D und folgende Klauseln ein:

C D Z´
C´ Z
D´ Z

Durch Anwendung der Schlussregel (IV) erhaelt man folgende Klauseln:

A B Z
A´ E Z

Durch Anwendung der Schlussregel (II) kann man

B E Z

herleiten. Diese Klausel ist aequivalent zur Klausel {B,C,D,E}. Das bedeutet, dass ein Literal durch SETZE eindeutig bestimmt ist. #

Lemma 2.4.1 zeigt, dass EB3 genuegend starke Schlussregeln enthaelt. Deshalb werden auf den erfuellbaren KNF die Schlussregeln von EB3 genuegend haeufig anwendbar sein. Das ist wichtig fuer eine schnelle Fortpflanzung der Modellinformation bei beschraenkter (3,M)-Resolution.

Im folgenden wollen wir Einblick in beschraenkte (3,M)-Resolution gewinnen. Sei (c1,c2,c3) eine Schlussregel von BM3. Die folgenden Lemmata kann man leicht durch Testen aller Faelle beweisen.

Lemma 2.4.2 :
1. Wenn c1 und c2 minimalen Erfuelltheitsgrad haben, so gilt das auch fuer c3.
2. eg(c3,M) $\geq$ min(eg(c1,M),eg(c2,M))
3. Wenn c1 und c2 nur die aufgeloeste Variable gemeinsam haben und wenn c1 nicht minimalen Erfuelltheitsgrad hat, dann gilt: eg(c3,M) > min(eg(c1,M),eg(c2,M)) und c3 hat nicht minimalen Erfuelltheitsgrad.
4. Wenn c1 und c2 nicht minimalen Erfuelltheitsgrad haben, dann gilt : eg(c3,M) $\geq$ max(eg(c1,M),eg(c2,M)) und eg(c3,M) $\geq$ 2/3.

Bemerkung:
2. gilt nicht fuer (4,M)-Resolution. Gegenbeispiel:

```
                              eg

c1      A  B  C               1/3
c2      A´          D  E      1/3
        -------------
c3         B  C  D  E         1/4
```

Modell M = {A,B´,C´,D,E´}

4. gilt nicht fuer (5,M)-Resolution. Gegenbeispiel:

```
                                 eg

c1      A  B  C                  2/3
c2      A´          D  E  F      1/2
        ----------------
c3         B  C  D  E  F         3/5
```

Modell M = {A,B,C´,D,E,F´}

Lemma 2.4.3 : Sei s eine erfuellbare KNF in ERF(3). Sei r eine Klausel, die aus s durch wiederholte Anwendung von folgenden Schlussregeln gewonnen wurde:

```
A  B
A´    C
-------
   B  C
```

```
A  B
A´    C  D
----------
   B  C  D
```

Seien d(1), d(2), ... d(n) die n Klauseln aus s, welche die Blaetter des Resolutionsbaumes der Resolventen r sind. Dann gilt fuer jedes Modell M von s:

1. Wenn mindestens ein d(i) nicht minimalen Erfuelltheitsgrad hat, dann hat auch r nicht minimalen Erfuelltheitsgrad.
2. Wenn zwei d(i) nicht minimalen Erfuelltheitsgrad haben, so ist eg(r,M)=1.
3. Es koennen hoechstens zwei d(i) nicht minimalen Erfuelltheitsgrad haben.

Lemma 2.4.3 zeigt, dass sich die Informationen ueber M bei Anwendung der obigen zwei Schlussregeln in der Resolvente ansammeln. Wir nennen deshalb diese beiden Schlussregeln informationsansammelnd.

Die beiden Lemmata koennen wie folgt zusammengefasst werden: Klauseln mit nicht minimalem Erfuelltheitsgrad pflanzen sich bei beschraenkter (3,M)-Resolution erstaunlich gut fort. Bei den informationsansammelnden Schlussregeln geht keine Information verloren.

2.5 Simulation der Modellinformationsfortpflanzung

In diesem Abschnitt simulieren wir ein Teilbeweissystem der beschraenkten (3,M)-Resolution. Wir untersuchen die folgenden drei Schlussregeln der beschraenkten (3,M)-Resolution:

```
A  B  C
A´       D
----------
   B  C  D
```

```
A  B
A´    C
-------
   B  C
```

```
A  B  C
A´ B     D
----------
   B  C  D
```

Wir nennen dieses Beweissystem RBM3. Die von BM3 weggelassenen Schlussregeln sind:

```
A  B  C
A´ B  C
-------
   B  C
```

```
A  B  C
A´ B
-------
   B  C
```

Man beachte, dass die ersten beiden Regeln von RBM3 die informationsansammelnden Regeln sind.

Sei c eine Klausel und I eine Interpretation von c. Sei e die Anzahl durch I erfuellter Literale in c, und sei l die Anzahl Literale von c. Wir nennen das Tupel(l,e) den Typ von c bei I. Fuer die Typenfortpflanzung bei RBM3 gilt folgende symmetrische Tabelle TR:

	(3,3)	(3,2)	(3,1)	(2,2)	(2,1)
(3,3)	-	(3,3)	(3,2)	-	(3,3)
(3,2)	(3,3)	(3,2)	*	(3,3)	(3,2)
(3,1)	(3,2)	*	(3,1)	(3,2)	(3,1)
(2,2)	-	(3,3)	(3,2)	-	(2,2)
(2,1)	(3,3)	(3,2)	(3,1)	(2,2)	(2,1)

Bei * ist sowohl (3,2) als auch (3,1) moeglich. Jeder Zeile, resp. Kolonne entspricht der Typ einer Elternklausel. Im Schnittpunkt steht der Typ (bei * die Typen) der Resolvente. Ein - bedeutet, dass sich diese beiden Typen nicht fortpflanzen koennen.

Nun folgt die Beschreibung des Simulationsalgorithmus. Sei s eine KNF in ERF(3), die fuer ein bestimmtes Modell genau a(i,j) Klauseln vom Typ (i,j) enthaelt. Der Simulationsalgorithmus waehlt zwei Typen und bestimmt nach den beiden folgenden Regeln mit der Tabelle TR den Typ der Resolvente:

1. Der Typ der ersten Elternklausel wird zuerst unter allen Typen folgendermassen gewaehlt: Sei at die Anzahl aller Klauseln, d.h. die Summe der a(i,j) ueber alle moeglichen i und j. Dann wird der Typ (i,j) mit Wahrscheinlichkeit a(i,j)/at gewaehlt. Sei (i1,j1) der gewaehlte Typ.

2. Der Typ der zweiten Elternklausel wird zuerst unter allen Klauseln gewaehlt, deren Typen fuer eine Resolventenbildung mit dem Typ (i1,j1) in Frage kommen. Sei atr die Anzahl aller solcher Typen. Dann wird der erlaubte Typ (i,j) mit Wahrscheinlichkeit a(i,j)/atr gewaehlt.

Falls genuegend Resolventen gebildet sind, wird die Auswahl des Typs der ersten Elternklausel auf die letzten d1 Resolventen beschraenkt. Der Typ der zweiten Elternklausel wird unter den letzten d2 Resolventen gewaehlt, welche fuer eine Resolventenbildung in Frage kommen. Die Auswahlwahrscheinlichkeiten werden wie oben proportional zu den Typanzahlen gebildet. Wir werden Simulationen mit verschiedenen Werten von d1 und d2 durchfuehren, z.B. d1=1 und d2 = 2 * Klauselanzahl der Eingabe-KNF (lineare Resolution).

Ist der Typ der Resolvente nicht eindeutig bestimmt, so wird einer der beiden Typen mit Wahrscheinlichkeit 1/2 gewaehlt. Begruendung: Bei folgender Schlussregel tritt dieser Fall ein:

```
c1   A  B  C
c2   A' B     D
     ----------
c3      B  C  D
```

Wenn in c1 genau 2 Literale erfuellt sind, koennen nicht B und C erfuellt sein, sonst waeren in c2 auch mindestens 2 Literale erfuellt. Also koennen nur entweder A und C oder A und B erfuellt sein. Wir nehmen an, diese beiden Faelle seien gleich wahrscheinlich.

Bevor wir Simulationsresultate analysieren, diskutieren wir die Typenzusammensetzung NP-vollstaendiger Sprachen. Dabei sind vor allem Klauseln mit nicht-minimalem Erfuelltheitsgrad wichtig, denn diese tragen zur Erhoehung der Modellinformation bei.

Sei q eine KNF einer NP-vollstaendigen Sprache Q in ERF(3). Wir fuehren in dieser KNF zufaellig Hilfsvariablen ein, d.h. wir waehlen zufaellig ein Literalpaar (a,b) und fuehren fuer dieses Literalpaar eine Hilfsvariable z ein. Weil die Literalpaare zufaellig gewaehlt werden, ist zu erwarten, dass das gesuchte Modell M die gewaehlten Literalpaare folgendermassen erfuellt:
In 1/4 der Literalpaare sind beide Literale erfuellt.
In 1/4 der Literalpaare ist kein Literal erfuellt.
In 1/2 der Literalpaare ist genau ein Literal erfuellt.

Die folgende Tabelle zeigt den Zusammenhang zwischen den Typen der fuer die Hilfsvariable z eingefuehrten Klauseln und der Belegung des Literalpaares (a,b). a=1 bedeutet, a ist erfuellt, und a=Ø bedeutet, a ist unerfuellt.

	a=1 b=1	a=1 b=Ø	a=Ø b=1	a=Ø b=Ø
a b z'	(3,2)	(3,1)	(3,1)	(3,1)
a' z	(2,1)	(2,1)	(2,2)	(2,1)
b'z	(2,1)	(2,2)	(2,1)	(2,1)

Summieren wir die Typen in dieser Tabelle, so erhalten wir:
6 mal Typ (2,1)
2 mal Typ (2,2)
3 mal Typ (3,1)
1 mal Typ (3,2)
1/3 der Typen haben nicht-minimalen Erfuelltheitsgrad.

Aufgrund dieser Ueberlegungen gilt, dass jede KNF nach genuegend haeufiger zufaelliger Einfuehrung von Hilfsvariablen obige prozentuale Typenzusammensetzung hat. (1/2 Typ (2,1), 1/6 Typ (2,2), 1/4 Typ (3,1), 1/12 Typ (3,2))

Bemerkung: Man kann effektiv NP-vollstaendige Sprachen Q in ERF(3) angeben, fuer die gilt: Fuer jede KNF in Q hat bei jedem Modell mindestens 1/5 der Klauseln den Typ (2,2).

Die erhaltenen Simulationsresultate sind im Anhang 2 aufgefuehrt. Sie koennen folgendermassen zusammengefasst werden: Unter obigen Annahmen pflanzt sich die Modellinformation rasch fort, d.h. es entstehen schnell Resolventen mit Erfuellheitsgrad 1. Die Geschwindigkeit der Fortpflanzung ist bei den gewaehlten Parametern praktisch unabhaengig von der Groesse der Eingabe-KNF. Sie nimmt nicht, wie zu erwarten waere, beim Wachsen der Eingabe-KNF rasch ab. Die Resultate deuten an, dass die Modellinformationsfortpflanzung algorithmisch interessant ist.

2.6 Algorithmus SR1

In diesem Kapitel beschreibe ich einen Algorithmus SR1, der die in den vorherigen Kapiteln gewonnenen Erkenntnisse ausnuetzt. Wir nehmen an, dass SR1 nur auf erfuellbaren KNF als Modellkonstruktionsalgorithmus operiert. Die Modellkonstruktion fuer eine erfuellbare KNF ist etwa gleich schwierig wie das Finden einer Entscheidung fuer eine KNF [SC76].

SR1 besteht aus mehreren Prozessen, die parallel ablaufen und stoppen, sobald einer die Entscheidung gefunden hat. Als gemeinsame Datenstruktur verwenden die Prozesse die KNF s. Jeder Prozess ist eine spezielle Variante von ESR, resp. SRUG und hat zwei Parameter. Der erste Parameter ist das Literalauswahlkriterium und der zweite Parameter die Art der Erweiterung von s, wenn SR kein Modell gefunden hat. Das Literalauswahlkriterium wird in einem Prozess verwendet, wenn der Fall "waehle" vorliegt. Wir beschreiben ein Literalauswahlkriterium durch Angabe einer Menge von Klauseln in UEBRIG und einer Auswahlmethode, die sich auf diese Menge von Klauseln bezieht.

Die Prozesse warten aufeinander, d.h. die Werte der Variablen SCHRITT der verschiedenen Prozesse sind in jedem Zeitpunkt gleich. Sei s1 die Eingabe-KNF. Sie enthalte n Variablen. Wir transformieren s1 durch Einfuehrung von Hilfsvariablen in eine KNF s in ERF(3).

Prozess 1 :

Literalauswahl :
Zufaellig unter allen Literalen in UEBRIG.

Erweiterung :
1 Superresolvente der Laenge hoechstens n.

Motivation :
Der erzeugte Superresolutionsbeweis soll moeglichst nicht einfach sein. Dadurch wird die KNF s rascher gesaettigt, weil die Uebersetzung von Superresolution in "Aufzaehlungsbaum" die Beweise eher verlaengert. (Siehe Abschnitt 1.13).

Prozess 2 :

Literalauswahl :
Methode : LGMD, die im folgenden beschrieben ist : Sei c eine Klausel und gew(c) = 2 ** (-Anzahl Literale in c) das Gewicht von c. Sei v eine Variable in UEBRIG, und sei gap(v) das Gewicht der Klauseln in UEBRIG, die v enthalten, und gan(v) das Gewicht der Klauseln in UEBRIG, die v´ enthalten. Sei vmax eine Variable in UEBRIG, fuer die abs(gap(v) - gan(v)) maximal ist. Dann waehlt LGMD den Literal vmax, falls gap(v) > gan(v), und sonst den Literal vmax´.

Erweiterung :
1 Superresolvente der Laenge hoechstens n.

Motivation :
Wenn UEBRIG ein Modell M mit hohem gewichtetem Erfuelltheitsgrad hat und sich dieser in jeder Variablen ausdrueckt, wird M rasch gefunden. Der gewichtete Erfuelltheitsgrad pflanzt sich im Durchschnitt besser fort als der Erfuelltheitsgrad (siehe Anhang 1).

Prozess 3 :

Literalauswahl :
Klauseln : Alle Klauseln in UEBRIG.
Methode : LGMD1, die folgende Eigenschaften hat : Wenn LGMD den Literal 1 waehlen wuerde, so waehlt LGMD1 den Literal 1´.

Erweiterung :
1 Superresolvente der Laenge hoechstens n.

Motivation :
Wenn UEBRIG ein Modell M mit niederem gewichtetem Erfuelltheitsgrad hat und sich dieser auf die Variablen uebertraegt, wird M rasch gefunden.

Prozess 4 :

Literalauswahl :
Klauseln : Die letzten k Resolventen, die durch beschraenkte (3)-Resolution erhalten wurden. Falls es keine hat, verwende alle Klauseln in UEBRIG.
Methode : Waehle beliebigen Literal in der bestimmten Klauselmenge.

Erweiterung :
1 Superresolvente der Laenge hoechstens n. Kuerze die durch Prozesse 1 bis 5 erzeugten Superresolventen durch maximal 5 * (n-4) Hilfsvariablen ab. Erzeuge maximal n Resolventen der beschraenkten (3)-Resolution, die noch nicht in s sind.

Motivation :
Beschraenkte (3,M)-Resolution garantiert eine gute Modellinformationsfortpflanzung. (Siehe die Durchschnittsuntersuchungen im Anhang 1 und die Simulation eines Teilsystems in Anhang 2.) Die Einfuehrung von Hilfsvariablen hat zwei Vorteile: Erstens ermoeglicht sie, dass Regeln der beschraenkten (3)-Resolution haeufig anwendbar sind, und zweitens, dass ein Beweis im starken Beweissystem "erweiterte Resolution" erzeugt wird. Es ist aber nicht klar, wie stark sich diese Vorteile auswirken.

Vergroesserung :
Pro Schritt hoechstens

$$1 + 3*(n-4)*5 + n = 16*n - 59$$

Klauseln der Laenge hoechstens 3.

Prozess 5 :

Literalauswahl :
Bestimme den informationsansammelnden Resolutionsbaum im bisherigen Resolutionsbeweis, der moeglichst viele Blaetter hat. (Eventuell besteht der Baum nur aus einer Klausel.) Setze nacheinander alle Literale der Resolvente r der Wurzel des Baumes wahr. (Jedesmal wird ein Literal aus r wahr gesetzt, wenn der Fall "waehle" eintritt. Eventuell wird dies verhindert wegen eindeutig bestimmten Literalen. Der groesste informationsansammelnde Resolutionsbaum wird nur bestimmt, wenn die Literale der Wurzel des letzten Baumes interpretiert sind.)

Erweiterung :
1 Superresolvente der Laenge hoechstens n.

Motivation :
Eine Resolvente, die an der Wurzel eines informationsansam-

melnden Resolutionsbaumes mit vielen Blaettern ist, hat mit grosser Wahrscheinlichkeit Erfuelltheitsgrad 1. Es ist nicht klar, wie schnell sich solche Baeume vergroessern und wieviele entstehen. Pro Schritt hoechstens c5 Klauseln der Laenge hoechstens n.

Bemerkungen zu SR1:

1. Aufwand:
Sei s1 die zu analysierende erfuellbare KNF mit n Variablen. Nach der Uebersetzung von s1 in eine KNF s in ERF(3) enthalte s genau c0 Klauseln und v0 Variablen. Sei sch der Wert der Variablen SCHRITT der Prozesse 1 bis 5. Dann enthaelt s hoechstens

$$c0 + sch * (5 + 3*(n-4)*5 + n) = c0 + sch*(16*n - 55)$$

Klauseln und hoechstens $v0 + sch * 5 * (n-4)$ Variablen. Sei $l(s1)$ die Anzahl Literale, die s1 enthaelt, und p ein festes Polynom. Dann gilt: Falls $sch \leq p(l(s1))$ ist, so ist der Aufwand von SR1 polynomial.

2. Modellinformationsfortpflanzung bei Resolution:
Sei p ein Polynom vom Grad 100, und wir nehmen an, dass $sch = p(l(s1))$. Wie hoch sind dann die informationsansammelnden Resolutionsbaeume gewachsen und wie gross ist der Erfuelltheitsgrad der letzten k Resolventen, die durch beschraenkte (3)-Resolution gebildet wurden ? Leider kann eine solche Frage nicht theoretisch beantwortet werden. Die Resultate meiner Simulation deuten an, dass sehr viele Resolventen Erfuelltheitsgrad 1 haben und deshalb praezise Informationen ueber ein gesuchtes Modell enthalten. Das bedeutet fuer SR1, dass er mit hoher Wahrscheinlichkeit ein Modell gefunden hat, nachdem $p(l(s1))$ Schritte durchgefuehrt wurden.

2.7 Schlussbemerkungen

Der in dieser Arbeit untersuchte Algorithmus SR1 ist in bezug auf seine Faehigkeit, ein Modell einer erfuellbaren KNF zu finden, nicht zu unterschaetzen. Leider ist eine Wahrscheinlichkeitsanalyse nicht moeglich, weil die Fortpflanzung der Typen nicht ueberblickbar ist. Exponentielle untere Schranken fuer Resolution und erweiterte Resolution wuerden nicht implizieren, dass SR1 exponentiell ist, d.h. exponentiell viele Modellkonstruktionen benoetigt, um ein Modell zu finden. Wir fassen die wichtigsten Vorteile SR1-aehnlicher Algorithmen nochmals zusammen:

1. Modellinformationsansammlung:
Die Modellinformation sammelt sich bei beschraenkter (3)-Resolution rasch an, weil die Fortpflanzungsregeln vorteil-

haft sind.
2. Flexibilitaet:
   Die Interpretationen werden nicht in einer festen Reihenfolge durchprobiert. Dadurch wird es wahrscheinlicher, dass existierende Modelle gefunden werden, denn die Algorithmen verhalten sich anpassend.
   3. Erweiterte Resolution:
   Die Interpretationen, die keine Modelle sein koennen, werden eventuell dank Hilfsvariablen mit wenig Speicheraufwand dargestellt. Deshalb kann eine erfuellbare KNF bei der Anwendung von SR1 rasch gesaettigt werden. (Eine erfuellbare KNF heisst gesaettigt, wenn SR bei der ersten Interpretationskonstruktion ein Modell findet.)
4. Effizienz auf "einfachen" Formeln:
   Die Algorithmen haben die Tendenz, sich auf "einfachen" Formeln guenstig zu verhalten (Siehe Abschnitt 1.7).
5. Beruecksichtigung von vielen Resolventen:
   Bei der Modellkonstruktion beruecksichtigt SR1 sehr viele (im Normalfall exponentiell viele) nicht explizit vorhandene Resolventen (Siehe Abschnitt 1.5).

Falls die Modellinformationsfortpflanzung bei beschraenkter 3-Resolution so gut ist, wie ich vermute, hat diese Entdeckung weitreichende Konsequenzen auf die weitere Entwicklung der Komplexitaetstheorie von Algorithmen fuer NP-vollstaendige Sprachen. Deshalb bezeichne ich Algorithmus SR1 als das Hauptresultat meiner Arbeit. Dass ich ueber die Effizienz von SR1 nicht viel b e w e i s e n kann, ist verstaendlich, wenn man bedenkt, dass Bestrebungen im Gange sind, zu zeigen, dass die P=NP-Frage von den Axiomen der Mengenlehre unabhaengig ist [HART76].

Neben der Modellinformationsansammlung habe ich das Beweissystem Superresolution eingehend analysiert. Superresolution hat die folgenden Vorzuege:
1. Superresolution ist eine starke Einschraenkung von Resolution. Deshalb ist die Gefahr, dass viele ueberfluessige Schluesse erzeugt werden, erheblich vermindert.
2. Superresolution ist trotzdem kuerzer als Resolution. Bisher war keine Einschraenkung von Resolution mit dieser Eigenschaft bekannt.

Das Beweissystem Superresolution laesst sich wie Resolution auch in die Praedikatenlogik uebertragen.

Die Komplexitaet von Superresolution habe ich relativ bestimmt und als Resultat erhalten, dass Superresolution und Resolution polynomial aequivalent sind. Fuer eine spezielle Art von Superresolution habe ich exponentielle untere Schranken gefunden.

Um Superresolution genauer analysieren zu koennen, habe ich einen Algorithmus SR eingefuehrt, der Superresolutionsbeweise nur in einer kurzen Normalform erzeugen kann. Ich habe ge-

zeigt, dass dieser Algorithmus die Tendenz hat, sich auf "einfachen" Formeln effizient zu verhalten. Insbesondere haben sich die erfuellbaren Formelmengen in der "Umgebung" von schwierigen, unerfuellbaren Formelmengen, fuer die exponentielle untere Schranken fuer gewisse Beweissysteme bekannt sind, als einfach entpuppt. Von Algorithmus SR wurde gezeigt, dass er sehr viele Resolventen bei seinen Interpretationskonstruktionen beruecksichtigt, ohne sie explizit zu erzeugen.

Im zweiten Teil der Arbeit habe ich probabilistische Algorithmen (im Sinne von Rabin [RAB76]) fuer NP-vollstaendige Sprachen untersucht. Ich habe den Zusammenhang zwischen dem Erfuelltheitsgrad von NP-vollstaendigen Sprachen und der Komplexitaet von probabilistischen Algorithmen fuer diese Sprachen hergestellt. Wie schnell man "reiche Zeugen" ("abundant witnesses" (Rabin)) fuer Erfuellbarkeit bekommt, haengt bei meinem Ansatz davon ab, wie rasch sich die Modellinformation bei beschraenkter 3-Resolution und bei Superresolution fortpflanzt.

Anhang 1

Die folgenden Tabellen geben Auskunft ueber die durchschnittliche Modellinformationsfortpflanzung bei verschiedenen Beweissystemen und Gewichtungen. In einer Tabelle werden fuer eine feste Gewichtung gew die Werte E[f,gew](k) fuer die Funktionen f = eg, f = geg dargestellt (nur fuer kleine Werte von k). Zuerst folgen die Tabellen der beschraenkten (k,M)-Resolution, anschliessend die der gewoehnlichen (k,M)-Resolution.

Beschraenkte (k,M)-Resolution

Gewichtung mit gew1

k	eg	geg	genotypenanzahl
2	0.1250000	0.1250000	2
3	0.0768519	0.0800265	18
4	0.0513859	0.0550817	67
5	0.0363646	0.0409103	180
6	0.0275164	0.0332161	406
7	0.0217510	0.0290137	811
8	0.0179240	0.0269147	1489

Gewichtung mit gew2

k	eg	geg	genotypenanzahl
2	0.1000000	0.1000000	2
3	0.0566881	0.0574545	18
4	0.0325747	0.0328377	67
5	0.0193677	0.0195120	180
6	0.0122525	0.0128013	406
7	0.0080652	0.0095361	811
8	0.0056328	0.0083492	1489

Gewichtung mit gew3

k	eg	geg	genotypenanzahl
2	0.1500000	0.1500000	2
3	0.0973658	0.1024302	18
4	0.0696851	0.0760060	67
5	0.0524326	0.0603321	180
6	0.0415242	0.0510947	406
7	0.0339867	0.0455278	811
8	0.0286291	0.0421575	1489

(k,M)-Resolution

Gewichtung mit gew1

k	eg	geg	genotypenanzahl
2	0.1250000	0.1250000	2
3	0.0710145	0.0734990	23
4	0.0437408	0.0483554	97
5	0.0296141	0.0370790	290
6	0.0216430	0.0324095	712
7	0.0168144	0.0310579	1532

Gewichtung mit gew2

k	eg	geg	genotypenanzahl
2	0.1000000	0.1000000	2
3	0.0487031	0.0492944	23
4	0.0233924	0.0251702	97
5	0.0114770	0.0153325	290
6	0.0055151	0.0120056	712
7	0.0024147	0.0118155	1532

Gewichtung mit gew3

k	eg	geg	genotypenanzahl
2	0.1500000	0.1500000	2
3	0.0924737	0.0964744	23
4	0.0626110	0.0693798	97
5	0.0458892	0.0559542	290
6	0.0356751	0.0493607	712
7	0.0289814	0.0463595	1532

Anhang 2

Die graphischen Darstellungen der Simulationsresultate (Abschnitt 2.5) haben folgende Bedeutung: Die Achse von links nach rechts gibt den Erfuelltheitsgrad der letzten m/10 Resolventen an, wobei m die Klauselnanzahl der Eingabe-KNF ist. Die Zahl mit 12 Stellen nach dem Komma bezeichnet den genauen Wert dieses Erfuelltheitsgrades fuer die naechste Zeile. Die Achse von oben nach unten gibt den Vergroesserungsfaktor der Klauselnanzahl der durch Resolventen erweiterten KNF in bezug auf m an. Der Vergroesserungsfaktor wird in Form einer Zahl mit einer Kommastelle angegeben.

Die Typenzusammensetzung der behandelten KNF ist diejenige, die man zu erwarten hat, wenn zufaellig genuegend Hilfsvariablen eingefuehrt werden.

Die graphischen Darstellungen gliedern sich in 3 Gruppen:

1. Vier graphische Darstellungen D(i) ($1 \leq i \leq 4$), wobei fuer D(i) gilt:
   a) die in D(i) behandelte KNF enthaelt i * 120 = m(i) Klauseln.
   b) d1 = 1 und d2 = 2 * m(i).
2. Vier graphische Darstellungen wie in 1., wobei aber d2 = 4 * m(i).
3. Vier graphische Darstellungen wie in 1., wobei aber d1 = 10.

1. Gruppe
****

Eingabe-KNF :

Typ (2,1) :	60	Klauseln
Typ (2,2) :	20	Klauseln
Typ (3,1) :	30	Klauseln
Typ (3,2) :	10	Klauseln
Typ (3,3) :	0	Klauseln

Klauselnanzahl der Eingabe-KNF : 120

```
d1=     1     d2=   240

     Ø                                                    1
     ---------------------------------------------------->
     Ø.861111111111
  1.5                                          .
     Ø.805555555556
  2.Ø                                        .
     Ø.750000000000
  2.5                                      .
     Ø.888888888889
  3.Ø                                         .
     Ø.833333333333
  3.5                                       .
     Ø.916666666667
  4.Ø                                           .
     Ø.916666666667
  4.5                                           .
     1.000000000000
  5.Ø                                                .
     1.000000000000
  5.5                                                .
     1.000000000000
  6.Ø                                                .
```

KNF am Schluss :

Typ (2,1):	61	Klauseln
Typ (2,2):	22	Klauseln
Typ (3,1):	72	Klauseln
Typ (3,2):	206	Klauseln
Typ (3,3):	524	Klauseln

Eingabe-KNF :

Typ (2,1) :	120	Klauseln
Typ (2,2) :	40	Klauseln
Typ (3,1) :	60	Klauseln
Typ (3,2) :	20	Klauseln
Typ (3,3) :	0	Klauseln

Klauselnanzahl der Eingabe-KNF : 240

```
d1=     1     d2=   480

     0                                                    1
     ---------------------------------------------------->
     0.861111111111
  1.5                                          .
     0.763888888889
  2.0                                       .
     0.888888888889
  2.5                                          .
     0.888888888889
  3.0                                          .
     0.944444444444
  3.5                                             .
     1.000000000000
  4.0                                                .
     1.000000000000
  4.5                                                .
     1.000000000000
  5.0                                                .
```

KNF am Schluss :

```
Typ (2,1):  120  Klauseln
Typ (2,2):   40  Klauseln
Typ (3,1):   95  Klauseln
Typ (3,2):  269  Klauseln
Typ (3,3): 1076  Klauseln
```

Eingabe-KNF :

```
Typ (2,1) :  180  Klauseln
Typ (2,2) :   60  Klauseln
Typ (3,1) :   90  Klauseln
Typ (3,2) :   30  Klauseln
Typ (3,3) :    0  Klauseln
```

Klauselnanzahl der Eingabe-KNF : 360

```
d1=     1     d2=   720

     0                                                    1
     ---------------------------------------------------->
     0.787037037037
  1.5                                    .
     0.638888888889
  2.0                                .
     0.666666666667
  2.5                                 .
     0.712962962963
  3.0                                   .
     0.833333333333
  3.5                                     .
     0.962962962963
  4.0                                       .
     0.907407407407
  4.5                                     .
     0.990740740741
  5.0                                       .
     0.981481481481
  5.5                                      .
     1.000000000000
  6.0                                      .
     1.000000000000
  6.5                                      .
     1.000000000000
  7.0                                      .
```

KNF am Schluss :

Typ (2,1): 182 Klauseln
Typ (2,2): 62 Klauseln
Typ (3,1): 244 Klauseln
Typ (3,2): 603 Klauseln
Typ (3,3): 1727 Klauseln

Eingabe-KNF :

Typ (2,1) : 240 Klauseln
Typ (2,2) : 80 Klauseln
Typ (3,1) : 120 Klauseln
Typ (3,2) : 40 Klauseln
Typ (3,3) : 0 Klauseln

Klauselnanzahl der Eingabe-KNF : 480

```
d1=     1     d2=   960

     0                                                    1
     ---------------------------------------------------->
     0.590277777778
  1.5                          .
     0.743055555556
  2.0                             .
     0.590277777778
  2.5                          .
     0.652777777778
  3.0                             .
     0.680555555556
  3.5                              .
     0.805555555556
  4.0                                 .
     0.902777777778
  4.5                                    .
     0.965277777778
  5.0                                       .
     0.993055555556
  5.5                                          .
     1.000000000000
  6.0                                          .
     1.000000000000
  6.5                                          .
```

KNF am Schluss :

Typ (2,1): 248 Klauseln
Typ (2,2): 80 Klauseln
Typ (3,1): 440 Klauseln
Typ (3,2): 944 Klauseln
Typ (3,3): 2444 Klauseln

2. Gruppe
****

Eingabe-KNF :

Typ (2,1) : 60 Klauseln
Typ (2,2) : 20 Klauseln
Typ (3,1) : 30 Klauseln
Typ (3,2) : 10 Klauseln
Typ (3,3) : 0 Klauseln

Klauselnanzahl der Eingabe-KNF : 120

```
d1=     1     d2=    480

      Ø                                                        1
      ---------------------------------------------------------->
      Ø.861111111111
  1.5                                          .
      Ø.805555555556
  2.Ø                                        .
      Ø.750000000000
  2.5                                       .
      Ø.888888888889
  3.Ø                                          .
      Ø.944444444444
  3.5                                             .
      Ø.805555555556
  4.Ø                                        .
      Ø.944444444444
  4.5                                             .
      Ø.944444444444
  5.Ø                                             .
      Ø.861111111111
  5.5                                          .
      1.000000000000
  6.Ø                                                 .
      Ø.916666666667
  6.5                                              .
      Ø.972222222222
  7.Ø                                                .
      Ø.972222222222
  7.5                                                .
      1.000000000000
  8.Ø                                                 .
      1.000000000000
  8.5                                                 .
      1.000000000000
  9.Ø                                                 .
```

KNF am Schluss :

Typ (2,1):	61	Klauseln
Typ (2,2):	22	Klauseln
Typ (3,1):	76	Klauseln
Typ (3,2):	277	Klauseln
Typ (3,3):	954	Klauseln.

```
Eingabe-KNF :

Typ (2,1) :  120  Klauseln
Typ (2,2) :   40  Klauseln
Typ (3,1) :   60  Klauseln
Typ (3,2) :   20  Klauseln
Typ (3,3) :    0  Klauseln

Klauselnanzahl der Eingabe-KNF :    240

d1=     1     d2=   960

     0                                                    1
     ----------------------------------------------------->
     0.805555555556
  1.5
     0.750000000000
  2.0
     0.861111111111
  2.5
     0.847222222222
  3.0
     0.791666666667
  3.5
     0.833333333333
  4.0
     0.861111111111
  4.5
     0.875000000000
  5.0
     0.916666666667
  5.5
     0.972222222222
  6.0
     0.986111111111
  6.5
     0.944444444444
  7.0
     1.000000000000
  7.5
     1.000000000000
  8.0
     1.000000000000
  8.5


KNF am Schluss :

Typ (2,1):  120  Klauseln
Typ (2,2):   41  Klauseln
Typ (3,1):  148  Klauseln
Typ (3,2):  536  Klauseln
Typ (3,3): 2035  Klauseln
```

Eingabe-KNF :

Typ (2,1) : 180 Klauseln
Typ (2,2) : 60 Klauseln
Typ (3,1) : 90 Klauseln
Typ (3,2) : 30 Klauseln
Typ (3,3) : 0 Klauseln

Klauselnanzahl der Eingabe-KNF : 360

```
d1=      1      d2=   1440

      Ø                                                        1
      ---------------------------------------------------------->
      Ø.629629629630
  1.5
      Ø.712962962963
  2.Ø
      Ø.666666666667
  2.5
      Ø.759259259259
  3.Ø
      Ø.805555555556
  3.5
      Ø.879629629630
  4.Ø
      Ø.750000000000
  4.5
      Ø.768518518519
  5.Ø
      Ø.870370370370
  5.5
      Ø.935185185185
  6.Ø
      Ø.916666666667
  6.5
      Ø.990740740741
  7.Ø
      Ø.981481481481
  7.5
      1.000000000000
  8.Ø
      Ø.981481481481
  8.5
      Ø.981481481481
  9.Ø
      1.000000000000
  9.5
      1.000000000000
 10.Ø
      1.000000000000
 10.5
      1.000000000000
 11.Ø
```

KNF am Schluss :

```
Typ (2,1):  181  Klauseln
Typ (2,2):   60  Klauseln
Typ (3,1):  285  Klauseln
Typ (3,2):  870  Klauseln
Typ (3,3): 2924  Klauseln
```

Eingabe-KNF :

Typ (2,1) :	240	Klauseln
Typ (2,2) :	80	Klauseln
Typ (3,1) :	120	Klauseln
Typ (3,2) :	40	Klauseln
Typ (3,3) :	0	Klauseln

Klauselnanzahl der Eingabe-KNF : 480

```
d1=     1     d2=  1920

     0                                                    1
     ---------------------------------------------------->
     0.631944444444
  1.5                          .
     0.763888888889
  2.0                           .
     0.750000000000
  2.5                           .
     0.722222222222
  3.0                          .
     0.743055555556
  3.5                           .
     0.763888888889
  4.0                           .
     0.756944444444
  4.5                           .
     0.770833333333
  5.0                            .
     0.819444444444
  5.5                              .
     0.902777777778
  6.0                                .
     0.909722222222
  6.5                                .
     0.937500000000
  7.0                                 .
     0.937500000000
  7.5                                 .
     0.951388888889
  8.0                                  .
     0.965277777778
  8.5                                  .
     0.993055555556
  9.0                                    .
     1.000000000000
  9.5                                    .
     0.993055555556
 10.0                                    .
     1.000000000000
 10.5                                    .
     1.000000000000
 11.0                                    .
```

KNF am Schluss :

```
Typ (2,1):  241  Klauseln
Typ (2,2):   80  Klauseln
Typ (3,1):  453  Klauseln
Typ (3,2): 1340  Klauseln
Typ (3,3): 3646  Klauseln
```

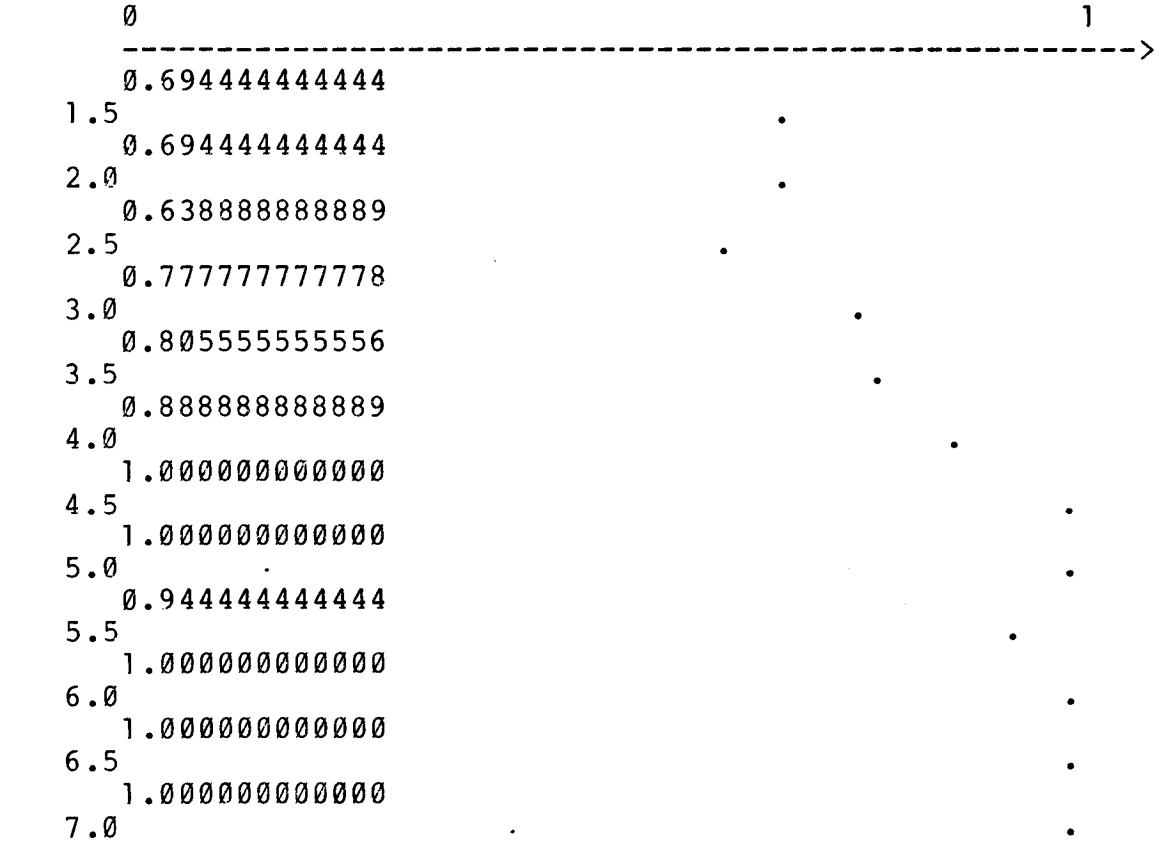
3. Gruppe
****

Eingabe-KNF :

Typ (2,1) : 60 Klauseln
Typ (2,2) : 20 Klauseln
Typ (3,1) : 30 Klauseln
Typ (3,2) : 10 Klauseln
Typ (3,3) : 0 Klauseln

Klauselnanzahl der Eingabe-KNF : 120

d1= 10 d2= 240



KNF am Schluss :

Typ (2,1): 67 Klauseln
Typ (2,2): 24 Klauseln
Typ (3,1): 89 Klauseln

```
Typ (3,2):  192  Klauseln
Typ (3,3):  531  Klauseln


Eingabe-KNF :

Typ (2,1) :  120  Klauseln
Typ (2,2) :   40  Klauseln
Typ (3,1) :   60  Klauseln
Typ (3,2) :   20  Klauseln
Typ (3,3) :    0  Klauseln

Klauselnanzahl der Eingabe-KNF :    240

d1=    10     d2=   480

     0                                                    1
     ----------------------------------------------------->
     0.736111111111
  1.5                                  .
     0.611111111111
  2.0                                .
     0.611111111111
  2.5                                .
     0.652777777778
  3.0                                 .
     0.819444444444
  3.5                                   .
     0.930555555556
  4.0                                     .
     0.861111111111
  4.5                                    .
     0.986111111111
  5.0                                      .
     1.000000000000
  5.5                                       .
     1.000000000000
  6.0                                       .
     1.000000000000
  6.5                                       .


KNF am Schluss :

Typ (2,1):  127  Klauseln
Typ (2,2):   51  Klauseln
Typ (3,1):  209  Klauseln
Typ (3,2):  425  Klauseln
Typ (3,3): 1252  Klauseln
```

```
Eingabe-KNF :

Typ (2,1) :  180  Klauseln
Typ (2,2) :   60  Klauseln
Typ (3,1) :   90  Klauseln
Typ (3,2) :   30  Klauseln
Typ (3,3) :    0  Klauseln

Klauselnanzahl der Eingabe-KNF :    360

d1=    10     d2=   720

     0                                                    1
     ----------------------------------------------------->
     0.648148148148
  1.5                          .
     0.546296296296
  2.0                       .
     0.666666666667
  2.5                          .
     0.842592592593
  3.0                             .
     0.814814814815
  3.5                             .
     0.962962962963
  4.0                                .
     0.962962962963
  4.5                                .
     1.000000000000
  5.0                                   .
     0.981481481481
  5.5                                .
     0.981481481481
  6.0                                .
     1.000000000000
  6.5                                   .
     1.000000000000
  7.0       .                           .
     1.000000000000
  7.5                                   .


KNF am Schluss :

Typ (2,1):  190  Klauseln
Typ (2,2):   61  Klauseln
Typ (3,1):  270  Klauseln
Typ (3,2):  580  Klauseln
Typ (3,3): 1878  Klauseln
```

```
Eingabe-KNF :

Typ (2,1) :  240  Klauseln
Typ (2,2) :   80  Klauseln
Typ (3,1) :  120  Klauseln
Typ (3,2) :   40  Klauseln
Typ (3,3) :    0  Klauseln

Klauselnanzahl der Eingabe-KNF :    480

d1=    10     d2=   960

     0                                                    1
     ----------------------------------------------------->
     0.763888888889
  1.5
     0.618055555556
  2.0
     0.715277777778
  2.5
     0.729166666667
  3.0
     0.972222222222
  3.5
     0.958333333333
  4.0
     0.930555555556
  4.5
     0.958333333333
  5.0
     0.986111111111
  5.5
     1.000000000000
  6.0
     1.000000000000
  6.5
     1.000000000000
  7.0


KNF am Schluss :

Typ (2,1):  252  Klauseln
Typ (2,2):   81  Klauseln
Typ (3,1):  323  Klauseln
Typ (3,2):  717  Klauseln
Typ (3,3): 2580  Klauseln
```

ZITIERTE ARBEITEN
****

DIE VERWENDETEN ABKUERZUNGEN VON ZEITSCHRIFTENNAMEN SIND AUF DER ZWEITEN SEITE DER ANSCHLIESSEND FOLGENDEN BIBLIOGRAPHIE ERKLAERT.

AA76. AANDERAA S., HORN FORMULAS AND THE P=NP PROBLEM, LOGIC COLLOQUIUM 76, OXFORD.

AHO75. AHO A.V., HOPCROFT J.E., ULLMAN J.D., THE DESIGN AND ANALYSIS OF COMPUTER ALGORITHMS, ADDISON-WESLEY.

CH73. CHANG C.L., LEE R.C., SYMBOLIC LOGIC AND MECHANICAL THEOREM PROVING, ACADEMIC PRESS, NEW YORK AND LONDON, 1973.

CO71. COOK S.A., THE COMPLEXITY OF THEOREM-PROVING PROCEDURES, 3.STOC (1971), 151-158.

CO74. COOK S., RECKHOW R., ON THE LENGTH OF PROOFS IN THE PROPOSITIONAL CALCULUS, 6.STOC (1974), 135-148.

CO75. COOK S.A., FEASIBLY CONSTRUCTIVE PROOFS AND THE PROPOSITIONAL CALCULUS, 7.STOC (1975).

COO76. COOK S.A., A SHORT PROOF OF THE PIGEON HOLE PRINCIPLE USING EXTENDED RESOLUTION, SIGACT NEWS, VOL 8, NO 4.

DA6Ø. DAVIS M., PUTNAM H., A COMPUTING PROCEDURE FOR QUANTIFICATION THEORY, J.ACM, VOL 7, 2Ø1-215.

GA74. GALIL Z., ON THE COMPLEXITY OF RESOLUTION PROCEDURES FOR THEOREM PROVING, TECHNICAL REPORT, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY, 1974.

GA75. GALIL Z., ON THE VALIDITY AND COMPLEXITY OF BOUNDED RESOLUTION PROCEDURES, 7.STOC (1975).

GAL76. GALIL Z., ON ENUMERATION PROCEDURES FOR THEOREM PROVING AND FOR INTEGER PROGRAMMING, IN AUTOMATA, LANGUAGES AND PROGRAMMING, THIRD INTERNATIONAL COLLOQUIUM, EDINBURGH, 1976.

HART76. HARTMANIS J., HOPCROFT J., INDEPENDENCE RESULTS IN COMPUTER SCIENCE, SIGACT NEWS, VOL 8, NO 4.

HU68. HU S., MATHEMATICAL THEORY OF SWITCHING CIRCUITS AND AUTOMATA, UNIVERSITY OF CALIFORNIA PRESS, BERKELEY AND LOS ANGELES.

JO73. JOHNSON D.S., APPROXIMATION ALGORITHMS FOR COMBINATORIAL PROBLEMS, 5.STOC (1973), 38-49.

KA72. KARP R.M., REDUCIBILITY AMONG COMBINATORIAL PROBLEMS, IN ''COMPLEXITY OF COMPUTER COMPUTATIONS'', R.E. MILLER AND J.W. THATCHER, EDS., PLENUM PRESS, NEW YORK, 1972, 85-104.

KAR76. KARP R.M., THE PROBABILISTIC ANALYSIS OF SOME COMBINATORIAL SEARCH ALGORITHMS, ELECTRONIC RESEARCH LABORATORY, COLLEGE OF ENGINEERING, UNIVERSITY OF CALIFORNIA, BERKELEY.

KI72. KIEBURTZ R.B., LUCKHAM D., COMPATIBILITY AND COMPLEXITY OF REFINEMENTS OF THE RESOLUTION PRINCIPLE, SIAM J. COMPUT., VOL. 1, 313-332.

KIR74. KIRKPATRICK D.G., TOPICS IN THE COMPLEXITY OF COMBINATORIAL ALGORITHMS, PH.D. THESIS, TORONTO.

KR67. KROM M.R., THE DECISION PROBLEM FOR A CLASS OF FIRST-ODER FORMULAS IN WHICH ALL DISJUNCTIONS ARE BINARY, ZEITSCHR. F. MATH. LOGIK U. GRUNDLAGEN D. MATH., 13, 15-20.

LA74. LADNER R., LYNCH N., SELMAN A.L., COMPARISON OF POLYNOMIAL-TIME REDUCIBILITIES, 6.STOC (1974), 110-122 AND TCS 1 (1975), 103-123.

LA75. LADNER R.E., ON THE STRUCTURE OF POLYNOMIAL TIME REDUCIBILITY, J.ACM, VOL 22, NO 1, 155-171.

LA75. LADNER R.E., THE CIRCUIT VALUE PROBLEM IS LOG SPACE COMPLETE FOR P, SIGACT NEWS, JANUARY 1975, 18-20.

LIE75. LIEBERHERR K., TOWARD FEASIBLE SOLUTIONS OF NP-COMPLETE PROBLEMS, ETH-ZUERICH, INSTITUT FUER INFORMATIK, CH-8092 ZUERICH, BERICHT NR.14.

LIE76. LIEBERHERR K., NP-COMPLETENESS AND DEGREES OF SATISFACTION, ABSTRACT FOR THE EUROPEAN SUMMER MEETING OF THE ASSOCIATION FOR SYMBOLIC LOGIC, OXFORD 1976.

MAR75. MARMIER E., AUTOMATIC VERIFICATION OF PASCAL PROGRAMS, DISS. ETH NR.5629.

ME68. MELTZER B., SOME NOTES ON RESOLUTION STRATEGIES, MI 3,71-76.

ME71. MELTZER B., PROLEGOMENA TO A THEORY OF EFFICIENCY OF PROOF PROCEDURES, IN ARTIFICIAL INTELLIGENCE AND HEURISTIC PROGRAMMING, AMERICAN ELZEVIER, 1971, 15 - 33.

MIL75. MILLER G.L., RIEMANN´S HYPOTHESIS AND TESTS FOR PRIMALITY, 7.STOC.

POS76. POSA L., HAMILTON CIRCUITS IN RANDOM GRAHS, DISCRETE MATHEMATICS 1976.

PR74. PRATT V.R., RABIN M., STOCKMEYER L.J., A CHARACTERIZATION OF THE POWER OF VECTOR MACHINES, 6.STOC (1974), 122-134.

REU76. REUSCH B., ON THE GENERATION OF PRIME IMPLICANTS, TECHNICAL REPORT, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY.

RAB76. RABIN M.O., PROBABILISTIC ALGORITHMS, RC 6164, IBM T.J.WATSON RESEARCH CENTER AND IN ALGORITHMS AND COMPLEXITY, ED. J.F. TRAUB, ACADEMIC PRESS.

RO65. ROBINSON J.A., A MACHINE-ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE, J.ACM, VOL 12, NO 1, 23-41.

RO68. ROBINSON J.A., THE GENERALIZED RESOLUTION PRINCIPLE, MI 3,77-93.

SC76. SCHNORR C.P., OPTIMAL ALGORITHMS FOR SELF-REDUCIBLE PROBLEMS, THIRD INTERNAT. COLL. ON AUTOMATA, LANG.AND PROG., EDINBURGH.

SHO76. SHOSTAK R.E., ON THE COMPLEXITY OF RESOLUTION, IN ALGORITHMS AND COMPLEXITY, ED. J.F. TRAUB, ACADEMIC PRESS, P.493.

SOL76. SOLOWAY R., STRASSEN V., A FAST MONTE-CARLO TEST FOR PRIMALITY, SUBMITTED FOR PUBLICATION.

SP75. SPECKER E., STRASSEN V., KOMPLEXITAET VON ENTSCHEIDUNGSPROBLEMEN, LNCS 43.

TAR76. TARJAN R.E., TROJANOWSKI A.E., FINDING A MAXIMUM INDEPENDENT SET, STAN-CS-76-55Ø, STANFORD UNIVERSITY.

TS68. TSEITIN G.S., ON THE COMPLEXITY OF DERIVATIONS IN THE PROPOSITIONAL CALCULUS, STRUCTURES IN CONSTRUCTIVE MATHEMATICAL LOGIC, PART II, A.O. SILENKO(ED), 1968, 115-125.

WI76. WIETLISBACH M., EINE UNTERE SCHRANKE FUER GENTZEN OHNE SCHNITT, PRIVATE MITTEILUNG.

BIBLIOGRAPHIE UEBER
´´P=NP UND VERWANDTE PROBLEME´´
****

THE BIBLIOGRAPHIE DOES NOT CLAIM TO BE COMPLETE. IT IS DIVIDED INTO THE FOLLOWING PARTS:

PART A : RESOLUTION PROOF PROCEDURES FOR PROPOSITIONAL LOGIC

PART B : APPROXIMATION ALGORITHMS FOR COMBINATORIAL PROBLEMS

PART C : NP-COMPLETE LANGUAGES, CLASSIFICATION OF NP

PART D : COMPLEXITY OF PROOF PROCEDURES AND LENGTH OF PROOFS

PART E : FAST ALGORITHMS

PART F : POLYNOMIAL-TIME REDUCIBILITIES

PART G : COMPLETE LANGUAGES (OTHER THAN NP-COMPLETE)

PART H : MACHINES FOR WHICH THE P=NP QUESTION IS SOLVED

PART J : ALGORITHMS FOR PROBLEMS IN NP WHICH WORK QUICKLY ON PRACTICAL PROBLEMS

PART K : COMBINATIONAL COMPLEXITY OF BOOLEAN FUNCTIONS

PART L : LENGTH OF BOOLEAN FORMULAS

PART M : PROBABILISTIC METHODS IN COMPLEXITY THEORY

PART N : RANDOM SEARCH

PART O : RANDOM GENERATION

PART Q : LOWER BOUNDS

BIBLIOGRAPHIE

ABBREVIATIONS

BELL SYST.TECH.J.
BELL SYSTEM TECHNICAL JOURNAL

DCM DISCRETE MATHEMATICS

EIK ELEKTRONISCHE INFORMATIONSVERARBEITUNG

FCS FOUNDATIONS OF COMPUTER SCIENCE

IC INFORMATION AND CONTROL

IPL INFORMATION PROCESSING LETTERS

J.ACM JOURNAL OF THE ASSOCIATION FOR COMPUTING MACHINERY

JCS
JOURNAL OF COMPUTER AND SYSTEM SCIENCES

LNCS LECTURE NOTES IN COMPUTER SCIENCE

MI MACHINE INTELLIGENCE (B. MELTZER AND D. MICHIE, EDS.), AMERICAN ELSEVIER, NEW YORK

STOC PROCEEDINGS OF THE ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING

SWAT PROCEEDINGS OF THE ANNUAL IEEE SYMPOSIUM ON SWITCHING AND AUTOMATA THEORY (FOUNDATIONS OF COMPUTER SCIENCE)

TCS THEORETICAL COMPUTER SCIENCE

PART A
RESOLUTION PROOF PROCEDURES
FOR PROPOSITIONAL LOGIC

AN68. ANDREWS P.B., RESOLUTION WITH MERGING, J.ACM, VOL 15, NO 3, 367-381.

AND70. ANDERSON R., BLEDSOE W.W., A LINEAR FORMAT FOR RESOLUTION WITH MERGING AND A NEW TECHNIQUE FOR ESTABLISHING COMPLETENESS, J.ACM, VOL 17, NO 3, 525-534.

CH73. CHANG C.L., LEE R.C., SYMBOLIC LOGIC AND MECHANICAL THEOREM PROVING, ACADEMIC PRESS, NEW YORK AND

LONDON, 1973.

DA6Ø. DAVIS M., PUTNAM H., A COMPUTING PROCEDURE FOR QUANTIFICATION THEORY, J.ACM, VOL 7, 2Ø1-215.

LOV7Ø. LOVELAND D.W., A LINEAR FORMAT FOR RESOLUTION, PROC. IRIA SYMP. AUTOMATIC DEMONSTRATION, SPRINGER-VERLAG, NEW YORK, 147-162.

MIC72. MICHIE D., G-DEDUCTION, MI 7.

MIN72. MINICOZZI E., A NOTE ON LINEAR RESOLUTION STRATEGIES IN CONSEQUENCE-FINDING, ARTIFICIAL INTELLIGENCE 3.

ME68. MELTZER B., SOME NOTES ON RESOLUTION STRATEGIES, MI 3,71-76.

RO65. ROBINSON J.A., A MACHINE-ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE, J.ACM, VOL 12, NO 1, 23-41.

ROB65. ROBINSON J.A., AUTOMATIC DEDUCTION WITH HYPERRESOLUTION, INTERNAT. J. COMPUT. MATH., 227-234.

RO68. ROBINSON J.A., THE GENERALIZED RESOLUTION PRINCIPLE, MI 3,77-93.

SH76. SHOSTAK, REFUTATION GRAPHS, ARTIFICIAL INTELLIGENCE, 7(1976), 51-64.

SL67. SLAGLE J.R., AUTOMATIC THEOREM PROVING WITH RENAMABLE AND SEMANTIC RESOLUTION, J.ACM 14(4), 687-697.

PART B
APPROXIMATION ALGORITHMS
FOR COMBINATORIAL PROBLEMS

AUS75. AUSIELLO G., ON THE COMPARISON OF NOTIONS OF APPROXIMATIONS, LNCS, BD.32, MATHEMATICAL FOUNDATIONS OF COMPUTER SCIENCE.

CHA75. CHANDRA K.C., APPROXIMATE ALGORITHMS FOR THE KNAPSACK PROBLEM AND ITS GENERALIZATIONS, RC 5616, IBM RES. REPORT.

GAR72. GAREY M.R., GRAHAM R.L., ULLMAN J.D., WORST CASE ANALYSIS OF MEMORY ALLOCATION ALGORITHMS, 4.STOC (1972), 143-15Ø.

GAJ76. GAREY M.R., JOHNSON D.S., THE COMPLEXITY OF

NEAR-OPTIMAL GRAPH COLORING, J. ACM, VOL 23.

JO72. JOHNSON D.S., FAST ALLOCATION ALGORITHMS, 13.SWAT (1972), 144-154.

JOG76. JOHNSON D.S., GAREY M.R., APPROXIMATION ALGORITHMS FOR COMBINATORIAL PROBLEMS: AN ANNOTATED BIBLIOGRAPHY, IN ALGORITHMS AND COMPLEXITY, ED. J.F. TRAUB, ACADEMIC PRESS.

JO73. JOHNSON D.S., APPROXIMATION ALGORITHMS FOR COMBINATORIAL PROBLEMS, 5.STOC (1973), 38-49.

KO75. KOHLER W.H., STEIGLITZ K., EXACT, APPROXIMATE, AND GUARANTEED ACCURACY ALGORITHMS FOR THE FLOW-SHOP PROBLEM N/2/F/F, J.ACM, VOL 22, NO 1, 106-114.

LIE75. LIEBERHERR K., TOWARD FEASIBLE SOLUTIONS OF NP-COMPLETE PROBLEMS, ETH-ZUERICH, INSTITUT FUER INFORMATIK, CH-8092 ZUERICH, BERICHT NR.14.

ROS74. ROSENKRANTZ D.J., STEARNS R.E., LEWIS P.M., APPROXIMATE ALGORITHMS FOR THE TRAVELING SALESPERSON PROBLEM, 15.SWAT (1974)

SA74. SAHNI S., GONZALES T., P-COMPLETE PROBLEMS AND APPROXIMATE SOLUTIONS, 15.SWAT (1974)

SA75. SAHNI S., APPROXIMATE ALGORITHMS FOR THE 0/1 KNAPSACK PROBLEM, J.ACM, VOL 22, NO 1, 115-124.

PART C
NP-COMPLETE LANGUAGES
CLASSIFICATION OF NP

ADL75. ADLEMAN L., MANDERS K., COMPUTATIONAL COMPLEXITY OF DECISION PROCEDURES FOR POLYNOMIALS, 15.FCS.

AA76. AANDERAA S., HORN FORMULAS AND THE P=NP PROBLEM, LOGIC COLLOQUIUM 76, OXFORD.

AU75. AUSIELLO G., ON THE COMPLEXITY OF DECISION PROBLEMS FOR CLASSES OF SIMPLE PROGRAMS ON STRINGS, INF. FACHBER. (5), GI-6.JAHRESTAGUNG.

BAR76. BARROW H.G., BURSTALL R.M., SUBGRAPH ISOMORPHISM, MATCHING RELATIONAL STRUCTURES AND MAXIMAL CLIQUES, IPL 76(4),83-84.

BO072. BOOK R.V., ON LANGUAGES ACCEPTED IN POLYNOMIAL TIME, SIAM J. COMPUTING,VOL. 1.

BO72. BOOK R.V., COMPLEXITY CLASSES OF FORMAL LANGUAGES, IN : AUTOMATA, LANGUAGES AND PROGRAMMING, PROCEEDINGS OF A SYMPOSIUM ORGANIZED BY IRIA, EDITED BY M.NIVAT, 1972.

BO74. BOOK R.V., COMPARING COMPLEXITY CLASSES, JCS VOL. 9.

BOO74. BOOK R.V., TALLY LANGUAGES AND COMPLEXITY CLASSES, IC VOL. 26.

BOO75. BOOK R.V., TRANSLATIONAL LEMMAS, POLYNOMIAL TIME AND (LOG N)**J-SPACE, TCS VOL. 1.

BRU76. BRUNO J., SETHI R., CODE GENERATION FOR A ONE REGISTER MACHINE, J. ACM, VOL. 23, 502-510.

CON74. CONSTABLE R.L., HUNT B.H., ON THE COMPUTATIONAL COMPLEXITY OF SCHEME EQUIVALENCE, TECHNICAL REPORT TR 74-201, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY, ITHACA, NEW YORK 14850.

CO71. COOK S.A., THE COMPLEXITY OF THEOREM-PROVING PROCEDURES, 3.STOC (1971), 151-158.

DO74. DOBKIN D., LIPTON R.J., ON SOME GENERALIZATION OF BINARY SEARCH, 6.STOC (1974), 310-316.

DU74. DUNHAM B., WANG H., TOWARD FEASIBLE SOLUTIONS OF THE TAUTOLOGY PROBLEM, RC 4924, IBM THOMAS J. WATSON RESEARCH CENTER, YORKTOWN HEIGHTS, N.Y. 10598 AND ANNALS OF MATHEMATICAL LOGIC, 10(2), 1976.

EV75. EVEN S., ON THE COMPLEXITY OF TIME TABLE AND MULTI-COMMODITY FLOW PROBLEMS, 16.FCS.

FA73. FAGIN R., GENERALIZED FIRST-ORDER SPECTRA AND POLYNOMIAL TIME RECOGNIZABLE SETS, SIAM-AMS PROCEEDINGS, VOLUME 7, EDITED BY R.M. KARP.

GAR74. GAREY M.R., JOHNSON D.S., SOME SIMPLIFIED NP-COMPLETE PROBLEMS, 6.STOC (1974), 47-63.

GA76. GAREY M.R., SOME NP-COMPLETE GEOMETRIC PROBLEMS, 8.STOC.

GAR76. GAREY M.R., JOHNSON D.S., SCHEDULING TASKS WITH NONUNIFORM DEADLINES ON TWO PROCESSORS, J. ACM, VOL. 23, 461-467.

HAR76. HARTMANIS J., ON ISOMORPHISMS AND DENSITY OF NP AND OTHER COMPLETE SETS, 8.STOC.

JOS72. JONES N., SELMAN A., TURING MACHINES AND THE SPECTRA OF FIRST-ORDER FORMULAS WITH EQUALITY, 4. STOC.

KA72. KARP R.M., REDUCIBILITY AMONG COMBINATORIAL PROBLEMS, IN ´´COMPLEXITY OF COMPUTER COMPUTATIONS´´, R.E. MILLER AND J.W. THATCHER, EDS., PLENUM PRESS, NEW YORK, 1972, 85-104.

LAE73. LAEUCHLI H., A REMARK ON A PAPER BY A. LEVY, JSRAEL JOURNAL.

LYN75. LYNCH N., ´´HELPING´´: SEVERAL FORMALIZATIONS, JSL VOL. 40(4).

MAC76. MACHTEY M., SIMPLE GOEDEL NUMBERINGS, TRANSLATIONS AND THE P-HIERARCHY, 8.STOC.

MAN76. MANDERS K., NP-COMPLETE DECISION PROBLEMS FOR QUADRATIC POLYNOMIALS, 8.STOC.

PAP76. PAPADIMITRIOU C.H., ON THE COMPLEXITY OF EDGE TRAVERSING, J.ACM, VOL.23, 544-554.

PUD75. PUDLAK P., POLYNOMIALLY COMPLETE PROBLEMS IN THE LOGIC OF AUTOMATED DISCOVERY, LNCS, BD. 32, MATH. FOUND. OF COMP. SCIENCE.

RA74. RABIN M.O., THEORETICAL IMPEDIMENTS TO ARTIFICIAL INTELLIGENCE, IFIP 74, 615-619.

SA72. SAHNI S., SOME RELATED PROBLEMS FROM NETWORK FLOWS, GAME THEORY, AND INTEGER PROGRAMMING, 13.SWAT (1972), 130-138.

SC75. SCHNORR C.P., SATISFIABILITY IS QUASI-LINEAR COMPLETE IN NQL, UNIVERSITAET FRANKFURT, 1975.

SC76. SCHNORR C.P., OPTIMAL ALGORITHMS FOR SELF-REDUCIBLE PROBLEMS, THIRD INTERNAT. COLL. ON AUTOMATA, LANG.AND PROG., EDINBURGH.

SE73. SETHI R., COMPLETE REGISTER ALLOCATION PROBLEMS, 5.STOC (1973), 182-195.

SP75. SPECKER E., STRASSEN V., KOMPLEXITAET VON ENTSCHEIDUNGSPROBLEMEN, LNCS 43.

ST75. STOCKMEYER L.J., THE POLYNOMIAL TIME HIERARCHY,RC 5379, IBM THOMAS J. WATSON RESEARCH CENTER.

STO75. STOCKMEYER L.J., THE SET BASIS PROBLEM IS NP-COMPLETE, RC 5431, MAT. SCIENCE DEPARTMENT, IBM THOMAS J. WATSON RESEARCH CENTER, YORKTOWN HEIGHTS, NEW YORK 10598.

UL73. ULLMAN J.D., POLYNOMIAL COMPLETE SCHEDULING PROBLEMS, 4TH SYMPOSIUM ON OPERATING SYSTEM

PRINCIPLES (1973), 96-101 AND IN J.COMPUTER + SYSTEM SCIENCES 10, (1975), 384-393.

WEG76. WEGBREIT B., COMPLEXITY OF SYNTHESIZING INDUCTIVE ASSERTIONS, XEROX PALO ALTO RESEARCH CENTER.

PART D
COMPLEXITY OF PROOF PROCEDURES
AND LENGTH OF PROOFS

AUSI76. AUSIELLO G., DIFFICULT LOGICAL THEORIES AND THEIR COMPUTER APPROXIMATIONS,ASTERISQUE 38-39.

BI75. BIBEL W., EFFIZIENZVERGLEICHE VON BEWEISPROZEDUREN, GI 74, LNCS 26, 153-160.

BIB75. BIBEL W., SCHREIBER J., PROOF SEARCH IN A GENTZEN-LIKE SYSTEM OF FIRST-ORDER LOGIC, PROC. INT. COMP. SYMP. (ICS'75), NORTH-HOLLAND, 205-212.

CO74. COOK S., RECKHOW R., ON THE LENGTH OF PROOFS IN THE PROPOSITIONAL CALCULUS, 6.STOC (1974), 135-148.

CO75. COOK S.A., FEASIBLY CONSTRUCTIVE PROOFS AND THE PROPOSITIONAL CALCULUS, 7.STOC (1975).

COO76. COOK S.A., A SHORT PROOF OF THE PIGEON HOLE PRINCIPLE USING EXTENDED RESOLUTION, SIGACT NEWS, VOL 8, NO 4.

GA74. GALIL Z., ON THE COMPLEXITY OF RESOLUTION PROCEDURES FOR THEOREM PROVING, TECHNICAL REPORT, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY, 1974.

GA75. GALIL Z., ON THE VALIDITY AND COMPLEXITY OF BOUNDED RESOLUTION PROCEDURES, 7.STOC (1975).

GAL76. GALIL Z., ON ENUMERATION PROCEDURES FOR THEOREM PROVING AND FOR INTEGER PROGRAMMING, IN AUTOMATA, LANGUAGES AND PROGRAMMING, THIRD INTERNATIONAL COLLOQUIUM, EDINBURGH, 1976.

HAR76. HARTMANIS J., ON EFFECTIVE SPEED UP AND LONG PROOFS OF TRIVIAL THEOREMS IN FORMAL THEORIES, R.A.I.R.O., INFORMATIQUE THEORETIQUE, VOL.10, NO.3.

HART76. HARTMANIS J., HOPCROFT J., INDEPENDENCE RESULTS IN COMPUTER SCIENCE, SIGACT NEWS, VOL 8, NO 4.

KI72. KIEBURTZ R.B., LUCKHAM D., COMPATIBILITY AND COMPLEXITY OF REFINEMENTS OF THE RESOLUTION

PRINCIPLE, SIAM J. COMPUT., VOL. 1, 313-332.

KIR74. KIRKPATRICK D.G., TOPICS IN THE COMPLEXITY OF COMBINATORIAL ALGORITHMS, PH.D. THESIS, TORONTO.

KOW71. KOWALSKI R., KUEHNER D., LINEAR RESOLUTION WITH SELECTION FUNCTION, ARTIFICIAL INTELLIGENCE 2(1971), 227-260.

LO74. LONGO G., VENTURINI ZILLI M., COMPLEXITY OF THEOREM-PROVING PROCEDURES : SOME GENERAL PROPERTIES, REVUE FRANCAISES D'AUTOMATIQUE, INFORMATIQUE ET RECHERCHE OPERATIONELLE, DEC. 1974,R-3, 5-18.

ME71. MELTZER B., PROLEGOMENA TO A THEORY OF EFFICIENCY OF PROOF PROCEDURES, IN ARTIFICIAL INTELLIGENCE AND HEURISTIC PROGRAMMING, AMERICAN ELZEVIER, 1971, 15 - 33.

SHO76. SHOSTAK R.E., ON THE COMPLEXITY OF RESOLUTION, IN ALGORITHMS AND COMPLEXITY, ED. J.F. TRAUB, ACADEMIC PRESS, P.493.

TS68. TSEITIN G.S., ON THE COMPLEXITY OF DERIVATIONS IN THE PROPOSITIONAL CALCULUS, STRUCTURES IN CONSTRUCTIVE MATHEMATICAL LOGIC, PART II, A.O. SILENKO(ED), 1968, 115-125.

WI76. WIETLISBACH M., EINE UNTERE SCHRANKE FUER GENTZEN OHNE SCHNITT, PRIVATE MITTEILUNG.

PART E
FAST ALGORITHMS

AHJ76. AHO A.V., JOHNSON S.C., OPTIMAL CODE GENERATION FOR EXPRESSION TREES, J.ACM VOL. 23(3).

AHO75. AHO A.V., HOPCROFT J.E., ULLMAN J.D., THE DESIGN AND ANALYSIS OF COMPUTER ALGORITHMS, ADDISON-WESLEY.

BA74. BATNI R.P., RUSSELL J.D., KIME CH.R., AN EFFICIENT ALGORITHM FOR FINDING AN IRREDUNDANT SET COVER, J.ACM, VOL 12, NO 3, 351-355.

EVE73. EVEN S., ALGORITHMIC COMBINATORICS, THE MAC MILLAN COMPANY, NEW YORK.

EVE75. EVEN S., AN O(N**2.5) ALGORITHM FOR MAXIMUM MATCHING IN GENERAL GRAPHS, 16.FCS.

GOO75. GOODMAN S.E.,HEDETNIEMI S.T., ADVANCES ON THE HAMILTONIAN COMPLETION PROBLEM, J. ACM, VOL. 22, 352-360.

HAR72. HARRIS R.V., A POLYNOMIAL BOUND ON THE COMPLEXITY OF THE DAVIS PUTNAM PROCEDURE AS APPLIED TO SYMMETRIZABLE PROPOSITIONS, PHD. THESIS, DEPT. OF COMPUTER SCIENCE CORNELL UNIVERSITY, AUGUST 1972.

HO74. HOPCROFT J.E., WONG J.K., LINEAR TIME ALGORITHM FOR ISOMORPHISM OF PLANAR GRAPHS, 6.STOC (1974), 172-184.

KAR75. KARP R.M., LI S.R., TWO SPECIAL CASES OF THE ASSIGNMENT PROBLEM, DCM VOL. 13.

MIL75. MILLER G.L., RIEMANN'S HYPOTHESIS AND TESTS FOR PRIMALITY, 7.STOC.

SHA75. SHAMOS M.I., CLOSEST-POINT PROBLEMS, 16.FCS.

TAR75. TARJAN R.E., SOLVING PATH PROBLEMS ON DIRECTED GRAPHS, STAN-CS-75-528, STANFORD UNIVERSITY.

PART F
POLYNOMIAL TIME REDUCIBILITIES

GI74. GILL J.T., AXIOMATIC INDEPENDENCE OF THE QUESTION NP=P, DEPARTMENT OF ELECTRICAL ENGINEERING, STANFORD UNIVERSITY, STANFORD, CALIFORNIA 94305.

LA74. LADNER R., LYNCH N., SELMAN A.L., COMPARISON OF POLYNOMIAL-TIME REDUCIBILITIES, 6.STOC (1974), 110-122 AND TCS 1 (1975), 103-123.

LA75. LADNER R.E., ON THE STRUCTURE OF POLYNOMIAL TIME REDUCIBILITY, J.ACM, VOL 22, NO 1, 155-171.

LEW76. LEWIS F.D., ON COMPUTATIONAL REDUCIBILITY, JCS 12, 122-131.

LYN75. LYNCH N., ON REDUCIBILITY TO COMPLEX OR SPARSE SETS, J. ACM, VOL. 22, 341-345.

MEH74. MEHLHORN K., POLYNOMIAL AND ABSTRACT SUBRECURSIVE CLASSES, 6.STOC (1974), 96-109.

PART G
COMPLETE LANGUAGES
(OTHER THAN NP-COMPLETE)

CAR76. CARDOZA E., LIPTON R., MEYER A., EXPONENTIAL SPACE COMPLETE PROBLEMS FOR PETRI NETS AND COMMUTATIVE SEMIGROUPS, 8. STOC.

EVEN75. EVEN S., TARJAN R.E., A COMBINATORIAL PROBLEM WHICH IS COMPLETE IN POLYNOMIAL SPACE, 7. STOC.

GALI75. GALIL Z., HIERARCHIES OF COMPLETE PROBLEMS, RC 5512, IBM T.J.WATSON RESEARCH CENTER.

JON74. JONES N.D., LAASER W.T., COMPLETE PROBLEMS FOR DETERMINISTIC POLYNOMIAL TIME, 6.STOC (1974), 40-46.

LA75. LADNER R.E., THE CIRCUIT VALUE PROBLEM IS LOG SPACE COMPLETE FOR P, SIGACT NEWS, JANUARY 1975, 18-20.

SCH76. SCHAEFER T.J., COMPLEXITY OF DECISION PROBLEMS BASED ON FINITE TWO-PERSON PERFECT-INFORMATION GAMES, 8.STOC.

PART H
MACHINES FOR WHICH THE
P=NP QUESTION IS SOLVED

BOO76. BOOK R.V., NIVAT M., P#NP FOR REVERSAL-BOUNDED ACCEPTORS, IN PREPARATION.

HA74. HARTMANIS J., SIMON J., ON THE POWER OF MULTIPLICATION IN RANDOM ACCESS MACHINES, 15.SWAT(1974).

PR74. PRATT V.R., RABIN M., STOCKMEYER L.J., A CHARACTERIZATION OF THE POWER OF VECTOR MACHINES, 6.STOC (1974), 122-134.

PART J
ALGORITHMS FOR PROBLEMS IN NP
WHICH WORK QUICKLY ON PRACTICAL PROBLEMS

AHR75. AHRENS J.H., MERGING AND SORTING APPLIED TO THE ZERO-ONE KNAPSACK PROBLEM, OPERATIONS RESEARCH, VOL.23.

CO69. COMPUTATION OF MINIMAL LENGTH FULL STEINER TREES ON THE VERTICES OF A CONVEX POLYGON, MATH. COMP. 23(1969), 521-531.

KE70. KERNIGHAN B.W., LIN S., AN EFFICIENT HEURISTIC FOR PARTITIONING GRAPHS, BELL SYST.TECH.J. 49 (1970), 291-308.

LI73. LIN S., KERNIGHAN B.W., AN EFFECTIVE HEURISTIC ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM , BELL SYST.TECH.J. 52 (1973), 498-516.

TAR76. TARJAN R.E., TROJANOWSKI A.E., FINDING A MAXIMUM INDEPENDENT SET, STAN-CS-76-550, STANFORD UNIVERSITY.

UL76. ULLMANN J.R., AN ALGORITHM FOR SUBGRAPH ISOMORPHISM, J.ACM, VOL.23, 31-42.

PART K
COMBINATIONAL COMPLEXITY
OF BOOLEAN FUNCTIONS

FI74. FISCHER M.J., LECTURES ON NETWORK COMPLEXITY, UNIVERSITY OF FRANKFURT, JUNE 1974.

MEH75. MEHLHORN K., GALIL Z., MONOTONE SWITCHING CIRCUITS AND BOOLEAN MATRIX PRODUCT, LNCS, BD.32, MATHEMATICAL FOUNDATIONS OF COMP. SCIENCE.

PA74. PAUL W.J., 2,25*N-LOWER BOUND ON THE COMPUTATIONAL COMPLEXITY OF BOOLEAN FUNCTIONS, TR74-222, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY, ITHACA, NEW YORK 14853.

PAT75. PATERSON M.S., AN INTRODUCTION TO BOOLEAN FUNCTION COMPLEXITY, ASTERISQUE, 38-39.

PI76. PIPPENGER N., SHIFTING GRAPHS AND THEIR APPLICATIONS, J. ACM, VOL.23, 423-432.

SCHM76. SCHMIDT D.C., A FAST BACKTRACKING ALGORITHM TO TEST DIRECTED GRAPHS FOR ISOMORPHISM USING DISTANCE MATRICES, J. ACM VOL. 23, 433-445.

SCH76. SCHNORR C.P., THE BREATH OF NETWORKS AND THE TAPE COMPLEXITY OF ON-LINE COMPUTATIONS, UNIVERSITY OF FRANKFURT.

UH72. UHLIG, UEBER DIE ANZAHL DER UNTERFUNKTIONEN EINER BOOLESCHEN FUNKTION ZUR CHARAKTERISIERUNG DER

KOMPLIZIERTHEIT IHRER REALISIERUNG, EIK 8 (1972)5, 255-268.

VAL76. VALIANT L.G., UNIVERSAL CIRCUITS, 8. STOC.

ZA75. ZASLAVSKI I.D., ON SOME MODELS OF COMPUTABILITY OF BOOLEAN FUNCTIONS, LNCS, BD.32, MATH. FOUND. OF COMP. SCIENCE.

PART L
LENGTH OF BOOLEAN FORMULAS

FIS75. FISCHER M.J., MEYER A.R., PATERSON M.S., LOWER BOUNDS ON THE SIZE OF BOOLEAN FORMULAS, 7.STOC.

HOD68. HODES L., SPECKER E., LENGTH OF FORMULAS AND ELIMINATION OF QUANTIFIERS I, CONTR. TO MATH. LOG. (1968) 175-188.

PART M
PROBABILISTIC METHODS
IN COMPLEXITY THEORY

FUC76. FUCHS P.H., SCHNORR C.P., MONTE CARLO METHODS AND PATTERNLESS SEQUENCES, SYMPOSIUM OPERATIONS RESEARCH, HEIDELBERG.

GIL76. GILL III J.T., COMPUTATIONAL COMPLEXITY OF PROBABILISTIC TURING MACHINES, FORSCHUNGSINSTITUT FUER MATHEMATIK ETH ZUERICH.

KAR76. KARP R.M., THE PROBABILISTIC ANALYSIS OF SOME COMBINATORIAL SEARCH ALGORITHMS, ELECTRONIC RESEARCH LABORATORY, COLLEGE OF ENGINEERING, UNIVERSITY OF CALIFORNIA, BERKELEY.

KNU76. KNUTH D.E., MATHEMATICS AND COMPUTER SCIENCE : COPING WITH FINITENESS, STAN-CS-76-541, FEBR.76, STANFORD UNIVERSITY.

PEA75. PEARL J., ON THE COMPLEXITY OF INEXACT COMPUTATIONS, IPL, VOL. 4.

POS76. POSA L., HAMILTON CIRCUITS IN RANDOM GRAHS, DISCRETE MATHEMATICS 1976.

RAB76. RABIN M.O., PROBABILISTIC ALGORITHMS, RC 6164, IBM T.J.WATSON RESEARCH CENTER AND IN ALGORITHMS AND

COMPLEXITY, ED. J.F. TRAUB, ACADEMIC PRESS.

SH76. SHAMOS M.I., GEOMETRY AND STATISTICS: PROBLEMS AT THE INTERFACE, IN ALGORITHMS AND COMPLEXITY, ED. J.F. TRAUB, ACADEMIC PRESS.

SOL76. SOLOWAY R., STRASSEN V., A FAST MONTE-CARLO TEST FOR PRIMALITY, SUBMITTED FOR PUBLICATION.

TRA75. TRAKHTENBROT B.A., ON PROBLEMS SOLVABLE BY SUCCESSIVE TRIALS, LNCS, BD.32, MATH. FOUND. OF COMP. SCIENCE.

PART N
RANDOM SEARCH

KAT72. KATONA G., COMBINATORIAL SEARCH PROBLEMS, INTERNATIONAL CENTRE FOR MECHANICAL SCIENCES, NO.145, UDINE 1972, SPRINGER VERLAG.

REN61. RENYI A., ON RANDOM GENERATING ELEMENTS OF A FINITE BOOLEAN ALGEBRA, ACTA SCI. MATH. SZEGED, 22, 75-81.

REN62. RENYI A., STATISTICAL LAWS OF ACCUMULATION OF INFORMATION, BULLETIN DE L´INST. INTERNAT. DE STAT., 39, 311-316.

REN65. RENYI A., ON THE THEORY OF RANDOM SEARCH, BULL. AMER. MATH. SOC., VOL.71, 809-828.

PART O
RANDOM GENERATION

KNU74. KNUTH D.E., ON THE GENERATION OF RANDOM MATROIDS, STANFORD UNIVERSITY, STAN-CS-74-453.

KNU76. KNUTH D.E., THE COMPLEXITY OF NONUNIFORM RANDOM NUMBER GENERATION, IN ALGORITHMS AND COMPLEXITY, ED. J.F. TRAUB, ACADEMIC PRESS.

NIJ75. NIJENHUIS A., COMBINATORIAL ALGORITHMS, ACADEMIC PRESS.

PART Q
LOWER BOUNDS

ENG75. ENGELER E., STRUCTURAL RELATIONS BETWEEN PROGRAMS AND PROBLEMS, INSTITUT FUER INFORMATIK, ETH-ZUERICH, BERICHT 15.

ENGE75. ENGELER E., LOWER BOUNDS BY GALOIS THEORY, ASTERISQUE, 38-39.

FIS74. FISCHER M.J., RABIN M.O., SUPER-EXPONENTIAL COMPLEXITY OF PRESBURGER ARITHMETIC, COMPLEXITY OF COMPUTATIONAL PROCESSES, AMERICAN MATHEMATICAL SOCIETY, PROVIDENCE, R.I.

FLE76. FLEISCHMANN K., MAHR B., SIEFKES D., BOUNDED CONCATENATION THEORY AS A UNIFORM METHOD FOR PROVING LOWER COMPLEXITY BOUNDS, PERDUE UNIV., COMPUTER SCI. DEPT., TECHNICAL REPORT 1976.

FRE75. FREDMAN M.L., ON THE DECISION TREE COMPLEXITY OF THE SHORTEST PATH PROBLEMS, 16. FCS.

HSZ76. HUNT (III) H.B., SZYMANSKI T.G., DICHOTOMIZATION, REACHABILITY, AND THE FORBIDDEN SUBGRAPH PROBLEM, 8. STOC.

POH75. POHL I., MINIMEAN OPTIMALITY IN SORTING ALGORITHMS, 16.FCS.

RAC75. RACKOFF CH., THE COMPLEXITY OF THEORIES OF THE MONADIC PREDICATE CALCULUS, IRIA, RAPPORT DE RECHERCHE 136.

RAC76. RACKOFF CH., ON THE COMPLEXITY OF THE THEORIES OF WEAK DIRECT POWERS, JSL 41, 561-573.

ROU75. ROUNDS W.C., A GRAMMATICAL CHARACTERIZATION OF EXPONENTIAL-TIME LANGUAGES, 16.FCS.

SCK76. SCHNORR C.P., KLUPP H., A UNIVERSALLY HARD SET OF FORMULAE WITH RESPECT TO NON-DETERMINISTIC TURING ACCEPTORS, FACHBEREICH MATHEMATIK, UNIVERSITAET FRANKFURT.

STO73. STOCKMEYER L., MEYER A.R., WORD PROBLEMS REQUIRING EXPONENTIAL TIME, 5.STOC.

VAL75. VALIANT L.G., ON NON-LINEAR LOWER BOUNDS IN COMPUTATIONAL COMPLEXITY, 7. STOC.

YAO75. YAO A.C., ON THE COMPLEXITY OF COMPARISON PROBLEMS USING LINEAR FUNCTIONS, 16. FCS.

Lebenslauf:

Am 27. August 1948 wurde ich als als Sohn des Karl und der Berta Lieberherr-Weder in Wattwil geboren. In Ebnat-Kappel besuchte ich die Primar- und Sekundarschule und in St. Gallen die kantonale Oberrealschule. Diese schloss ich im Herbst 1968 mit der Matura Typus C ab. Anschliessend immatrikulierte ich mich an der ETH Zuerich an der Abteilung fuer Mathematik und Physik. Im Herbst 1972 bestand ich das Diplom in Mathematik, im folgenden Semester schrieb ich meine Diplomarbeit bei Professor Dr. Ernst Specker. Seit Fruehjahr 1973 war ich Assistent bei Professor Dr. Erwin Engeler am Institut fuer Informatik der ETH Zuerich.