Verbesserte SAT basierte Fehlerdiagnose durch Widerspruchanalyse

André Sülflow Görschwin Fey Rolf Drechsler

Fachbereich 3 – Mathematik und Informatik
Universität Bremen
28359 Bremen
{suelflow,fey,drechsle}@informatik.uni-bremen.de

Zusammenfassung

Durch Simulation oder formale Verifikation wird ein Design automatisch gegen spezifizierte Eigenschaften überprüft. Schlägt die Prüfung fehl, ist manuelles Debugging notwendig, um den oder die Fehler im Design oder der Spezifikation zu finden. Erste automatische Verfahren zur Eingrenzung der Fehlerkandidaten wurden entwickelt und bereits erfolgreich eingesetzt. Die Anzahl der Fehlerkandidaten ist aber noch immer in einer Größenordnung, die den Einsatz in der Praxis erschwert. In dieser Arbeit wird eine Erweiterung des Verfahrens von [8] um eine Widerspruchsanalyse vorgestellt, die die Anzahl der Kandidaten für Fehlerstellen deutlich reduzieren kann. In Experimenten konnte für kombinatorische Schaltkreise bei angenommenen Zweifachfehlern eine Reduzierung der Fehlerkandidaten um bis zu 94% erreicht werden.

1 Einleitung

Der Entwicklungsprozess eines Schaltkreises von der Idee bis zum fertigen Produkt enthält viele Stufen. Aus der grundlegenden Idee wird eine Spezifikation entworfen und darauf aufbauend das Schaltkreis-Design entwickelt. Aus dem Design wird im Anschluss ein synthesefähiges Modell entwickelt, welches sich gegen die Spezifikation prüfen lässt. Es können dabei sowohl Fehler im Design als auch der Spezifikation auftreten, welche manuell untersucht werden müssen. In dieser Arbeit werden nun Fehler im Design näher betrachtet.

Zur Überprüfung eines Designs werden automatische und teilautomatische Verfahren wie formale Verifikation und Simulation eingesetzt. Diese liefern eine Menge von Gegenbeispielen, welche durch manuelles und damit zeitintensives Debugging näher untersucht werden.

Für ein gegebenes fehlerhaftes Design, eine Spezifikation und eine Menge von Gegenbeispielen wurden erste unterstützende automatische Verfahren beispielsweise von [5, 4, 9] entwickelt. Diese identifizieren Fehlerkandidaten und können so den bisherigen komplett manuellen Debugging-Prozess enorm unterstützen. Speziell für Mehrfachfehler ist die Menge der Fehlerkandidaten aber noch in einem Bereich, der einen praxisrelevanten Einsatz nur schwer möglich macht.

In dieser Arbeit wurden daher Mehrfachfehler bei kombinatorischen Schaltkreisen näher untersucht. Als Diagnoseverfahren wurde [7, 8] eingesetzt, welches bereits gute Anwendbarkeit bei Einfachfehlern gezeigt hat. Allerdings ergaben sich dabei auch nichtdeterministische Korrekturen, welche in der Praxis häufig nicht gesucht sind. Im Unterschied zu den bisher veröffentlichten Verfahren garantiert das hier vorgestellte eine deterministische Korrektur. Dadurch wird insgesamt eine deutliche Reduktion der Anzahl der Fehlerkandidaten erreicht. Dazu wurden in dieser Arbeit gezielt die Belegung der Eingangs- und Ausgangswerte der Fehlerkandidaten betrachtet. Die Untersuchung bei Mehrfachfehlerkandidaten ergab, dass häufig nicht widerspruchsfreie Belegungen auftraten. Ein Widerspruch tritt dabei immer dann auf, wenn die Diagnose für einen Fehlerkandidaten bei gleichem Eingabevektor unterschiedliche Ausgabewerte fordert. Wenn die Funktion eines Gatters aber deterministisch sein soll, kann es keine deterministische Reparatur für diesen Fehlerkandidaten geben. Durch den Ausschluss von Widersprüchen in dem Verfahren [7, 8] lässt sich die Anzahl von Fehlerkandidaten deutlich reduzieren.

Diese Arbeit ist wie folgt gegliedert: In Abschnitt 2 wird zunächst die Grundidee des verwendeten Verfahrens erläutert, gefolgt von der Beschreibung des Widerspruchanalyse-Verfahrens in Abschnitt 3. Eine experimentelle Evaluation und ein Vergleich der vorgestellten Verfahren wird abschließend in Abschnitt 4 durchgeführt.

2 Grundlagen

Bevor auf das Auftreten und die Verhinderung von Widersprüchen weiter eingegangen wird, sollen zunächst die Grundlagen zu Fehlern in Schaltkreisen und des Verfahrens von [8] beschrieben werden.

Durch *Equivalence-Checking* (EC) lassen sich für einen gegebenen fehlerhaften Schaltkreis Gegenbeispiele erzeugen, die die Spezifikation nicht erfüllen. Diese Gegenbeispiele werden dann beim Prozess des Debugging genauer untersucht, und versucht der Ort und die Art des Fehlers zu lokalisieren.

Nach der Synthese der RT-Ebene sind oftmals noch manuelle Korrekturen auf Gatter-Ebene notwendig. Bei den manuellen Korrekturen werden dabei aber auch häufig Fehler injiziert. Zehn häufig vorkommende Fehlertypen wurden beispielsweise in [1] zusammengetragen. Dies sind zum Einen einfache Fehler wie vergessene Gatter oder Gattervertauschungen (AND statt OR), aber auch komplexere, wie zum Beispiel fehlerhaft angeschlossene Signalleitungen. Die Aufgabe beim Debugging ist es nun auf der Basis von Erfahrung manuell die Fehlerkandidaten einzugrenzen, den Fehler zu identifizieren und zu beheben.

Die Verfahren von [7, 8] versuchen die Identifikation von Fehlerkandidaten automatisiert anzugehen. Wird ein Gegenbeispiel an den primären Eingängen des fehlerhaften Schaltkreises angelegt, so wird die Spezifikation nicht erfüllt. Die Verfahren versuchen nun diesen Konflikt zu lösen, indem sie automatisiert nach Reparaturmöglichkeiten suchen, welche die Spezifikation wieder erfüllen. Hierzu wird, wie in Abbildung 1 dargestellt, der Ausgang eines jeden Gatters des fehlerhaften Schaltkreises mit einem Multiplexer verbunden, dessen zweiter Eingang durch einen neuen primären Eingang belegt ist. Nun lässt sich mit Hilfe eines Kontrollsignals (Fehlerprädikat) bestimmen, ob das originale Ausgangssignal des Gatters oder das des neuen primären Einganges durchgeschaltet wird.

Mit dieser Erweiterung des fehlerhaften Schaltkreises können die Verfahren nun gezielt die Ausgangssignale eines Gatters für die nachfolgenden Gatter verändern.

Die Verfahren testen für alle Gatter, ob nach der Veränderung des Ausganges dieses Gatters die Spezifikation erfüllt ist. Dazu wird jeweils exakt ein Fehlerprädikat umgestellt,

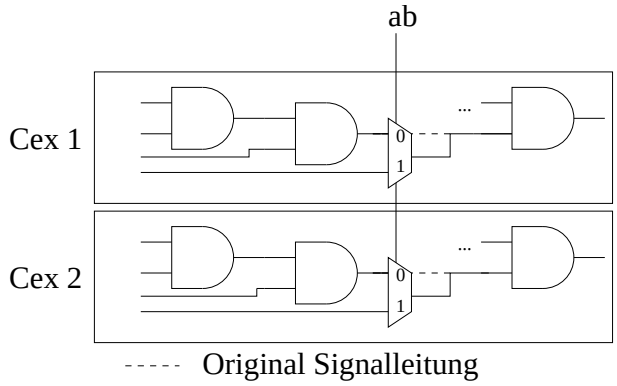


Abbildung 1: Verknüpfung von Fehlerprädikaten, Multiplexern und Gegenbeispielen

so dass der neue primäre Eingang den Ausgang des Gatters ersetzt. Simulativ werden auf den primären Eingang danach nacheinander die Werte 0 und 1 gelegt. Wenn sich bei einem dieser zwei Werte die Spezifikation erfüllen lässt, dann ist eine Korrekturstelle zur Behebung des Fehlers im Schaltkreis gefunden worden. Andernfalls wird das Fehlerprädikat wieder zurückgeschaltet und mit dem nächsten Gatter fortgefahren. Auf diese Art und Weise werden systematisch alle Gatter überprüft.

Da für große Schaltkreise, mit vielen Gattern, die Laufzeiten für die Diagnose stark ansteigen, wurde in [2] ein hierarchischer Ansatz für das Verfahren entwickelt. Hier wird eine Menge von Gattern zu Komponenten zusammengefasst, die dann eine weitaus geringere Menge an Ausgängen haben. Auf diesen Ausgängen lässt sich dann das oben beschriebene Verfahren wieder durchführen. Im Weiteren wird der Begriff Komponente als Synonym für Gatter verwendet, da hier ein Gatter einer Komponente entspricht.

Die Verfahren [7, 8] setzen für die Diagnose einen inkrementellen SAT-Beweiser [10] ein. Die Aufgabe des SAT-Beweisers ist es nun zum Einen die Fehlerprädikate und gleichzeitig die Werte der neuen primären Eingänge zu setzen. Beides geschieht durch eine entsprechende Formulierung als *Konjunktive Normal Form* (KNF). Bei Einfachfehlern wird exakt ein Fehlerprädikat zur gleichen Zeit gesetzt, während bei n-fach Fehlern $1, .., n$ Fehlerprädikate gleichzeitig gesetzt werden. Durch die Vielzahl an Kombinationsmöglichkeiten für die Fehlerprädikate erhöht sich bei Mehrfachfehlern die Komplexität und damit die Laufzeit entsprechend stark.

Das oben beschriebene Vorgehen wird entweder iterativ [7] oder direkt [8] für alle Gegenbeispiele durchgeführt. Mit jedem Gegenbeispiel erhält man für ein Fehlerprädikat eine korrekte Belegung für die Eingänge und Ausgänge der Komponente, die die Spezifikation erfüllt; somit also eine partielle Funktion, die eine Komponente ausführen muss. Jedes Gegenbeispiel liefert somit eine partielle Funktion und die Gesamtheit aller partiellen Funktionen für ein Fehlerprädikat bildet dann eine partielle Korrekturfunktion. Die Korrekturfunktion ist nur partiell, da die gewählten Gegenbeispiele in der Regel nicht den kompletten Belegungsraum abdecken. Je unterschiedlicher die durch die Gegenbeispiele hervorgerufenen Eingangs- und Ausgangsbelegungen der Komponenten an den Fehlerprädikaten sind, desto genauer ist die Korrekturfunktion.

Beispiel 1. *In Abbildung 2 wurde die Bildung der Korrekturfunktion an einem Beispiel dargestellt. In dem Beispiel ist das Gatter G2 ein ermittelter Fehlerkandidat. Vom SAT-Beweiser wurden dabei für jedes Gegenbeispiel die primären Eingangswerte des Schaltkreises und die korrigierte Belegung des Ausganges von G2 festgelegt. Unter dieser Konfiguration wird die Spezifikation erfüllt. Die Eingangs- und die korrigierte Ausgangsbelegung von*

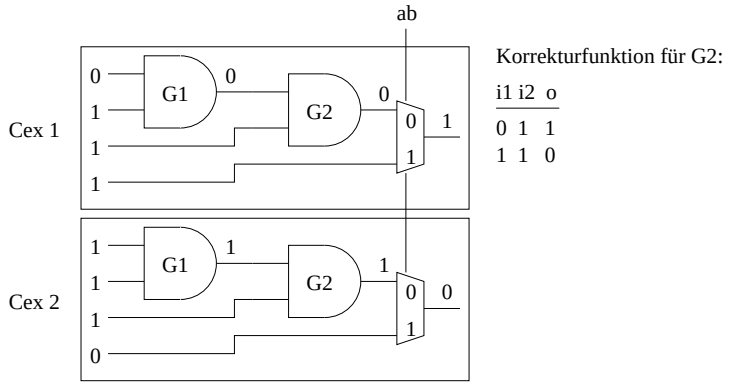


Abbildung 2: Beispiel für die Bildung der Korrekturfunktion

```
1  bool isContradictionFree (ab_1,..,ab_m)
2  {
3    tmpCnf = σ_keep;
4    for ab ∈ {ab_1,..,ab_m}
5    {
6      tmpCnf += addDistinctIOClause(ab);
7    }
8    cnf += tmpCnf;
9    result = (cnf.solve() == SATISFIABLE);
10   cnf -= tmpCnf; //Entfernen der CNF Erweiterung
11   return result;
12 }
```

Abbildung 3: Pseudocode zur Ermittlung der Widerspruchsfreiheit

G2 geben nun für jedes Gegenbeispiel eine partielle Funktion an, die das Gatter G2 für einen äquivalenten Schaltkreis liefern muss. Die auf diese Weise für alle Gegenbeispiele gewonnenen partiellen Funktionen können anschliessend in einer partiellen Korrekturfunktion kombiniert werden. In diesem Fall lässt sich durch Austausch des AND-Gatters G2 durch beispielsweise ein XOR-Gatter die partielle Korrekturfunktion erfüllen.

3 Widerspruchsfreie Diagnose

Bei der Bildung der Korrekturfunktion aus den partiellen Funktionen kann es zu Widersprüchen kommen. Dies ist dann der Fall, wenn bei gleicher Eingangsbelegung unterschiedliche Ausgangswerte folgen. Genau dann lässt sich keine deterministische Reparatur durchführen und die Komponente, deren Fehlerprädikat gesetzt ist, kann nicht repariert und damit als Fehlerkandidat ausgeschlossen werden. Die Diagnosegenauigkeit wird dadurch verbessert. In diesem Abschnitt wird die Erkennung von Widersprüchen während der Diagnose als Erweiterung für das vorgestellte Verfahren von [8] beschrieben.

Nachdem der SAT-Beweiser eine Kombination von Fehlerprädikaten gefunden hat, die die Spezifikation erfüllt, kann nun die Widerspruchsfreiheit für diese Kombination überprüft werden. Es wird dafür die KNF um Klauseln für die aktuellen Fehlerprädikate erweitert, die die Widerspruchsfreiheit der Korrektur garantieren. Mit dem inkrementellen SAT-Beweiser lässt sich dann effizient herausfinden, ob ein Widerspruch in der Belegung besteht.

In Abbildung 3 ist die Methode `isContradictionFree` zum Ausschließen von Widersprüchen in Pseudocode dargestellt. Die Parameter dieser Methode sind die aktuell gesetzten Fehlerprädikate (engl.: abnormal predicates). Zunächst werden alle aktuell ge-

```
1  void addDistinctIOClause (ab)
2  {
3    tmpCnf = '';
4    for (int i = 0; i < C.size(); i++) {
5      for (int j = i+1; j < C.size(); j++) {
6        tmpCnf += σ_consistent(C_i, C_j, ab);
7      }
8    }
9    cnf += tmpCnf;
10 }
```

Abbildung 4: Pseudocode zur Erstellung einer Widerspruchsklausel für ein Fehlerprädikat

setzten Fehlerprädikate konstant gehalten, um Widersprüche an diesen analysieren zu können.

Dies wird wie folgt formuliert: Seien $ab_1..ab_m$ die aktuellen Fehlerprädikate der Fehlerkandidaten, dann werden durch die Bedingung

$$\sigma_{keep} = ab_1 \text{ AND } ... \text{ AND } ab_m \tag{1}$$

die Fehlerprädikate konstant gehalten.

Im Anschluss wird dann mit der Methode *addDistinctIOClause* (Abbildung 4) für jedes Fehlerprädikat eine Widerspruchsklausel über alle Paare von Gegenbeispielen wie folgt formuliert und hinzugefügt:

Seien $I_{1C_i}..I_{kC_i}$ die Eingänge und $O_{1C_i}..O_{lC_i}$ die Ausgänge des Fehlerkandidaten für das Gegenbeispiel C_i, der dem Fehlerprädikat ab entspricht. Dann wird eine konsistente Bedingung für die Gegenbeispiele C_1 und C_2 wie folgt formuliert:

$$\begin{aligned}\sigma_{consistent}(C_i, C_j, ab) = ((I_{1C_1} \;==\; I_{1C_2}) \text{ AND } ... \text{ AND } (I_{kC_1} \;==\; I_{kC_2})) \\ \Rightarrow ((O_{1C_1} \;==\; O_{1C_2}) \text{ AND } ... \text{ AND } (O_{lC_1} \;==\; O_{lC_2}))\end{aligned} \tag{2}$$

Die Formel (2) erfordert somit $k + l$ $NXOR$ Gatter, $k + l - 2$ AND Gatter und eine Implikation. Wie ein Schaltkreis lässt sich dieser Ausdruck in einen KNF umwandeln, wobei die Anzahl der Klauseln hierin die Anzahl der Operatoren in $\sigma_{consistent}$ ist und die Anzahl der Variablen der Anzahl der Operatoren entspricht.

Nachdem die KNF um alle konsistenten Bedingungen erweitert worden ist, wird der SAT-Beweiser aufgerufen, um zu überprüfen, ob die KNF erfüllbar ist. Wird mindestens eine Lösung gefunden, dann existiert eine deterministische Korrekturfunktion und der Fehlerkandidat kann nicht ausgeschlossen werden. Andernfalls kann der Fehlerkandidat verworfen werden. Im letzten Schritt werden dann noch die hinzugefügten Widerspruchsbedingungen σ_{keep} und $\sigma_{consistent}$ wieder entfernt. Ein inkrementeller SAT-Beweiser garantiert dabei eine effiziente Verarbeitung der KNF und damit des Suchprozesses [10]. Im Anschluss führt der SAT-Beweiser die Suche nach weiteren Fehlerkandidaten fort.

Für komplexe Fehlertypen (z.B. *missing wire* [1]) kann es möglich sein, dass keine deterministische Korrektur für einen oder alle Fehlerkandidaten existiert. In diesem Fall sollte der hierarchische Ansatz [2] ergänzend eingesetzt werden.

Als Erweiterung der oben beschriebenen widerspruchsfreien Diagnose lässt sich nicht nur eine Korrekturfunktion ermitteln, d.h. eine mögliche Ersetzung durch kombinatorische Logik, sondern die Menge aller möglichen Ersetzungen, die zur Korrektur führen. Dies macht die Diagnose genauer und kann das Debugging enorm unterstützen.

```
1  int  getNumberOfDifferentAssignments (ab_1,..,ab_m)
2  {
3          tmpCnf = σ_keep;
4          for ab ∈ {ab_1,..,ab_m}
5          {
6             tmpCnf += addDistinctIOClause(ab);
7          }
8          cnf += tmpCnf;
9          int numberOfDifferentAssignments = 0;
10         while (cnf.solve() == SATISFIABLE)
11         {
12           numberOfDifferentAssignments++;
13           blockingClause = blockCurrentAssignment();
14           tmpCnf    += blockingClause;
15           cnf       += blockingClause;
16         }
17         cnf -= tmpCnf; //Entfernen der CNF Erweiterung
18         return numberOfDifferentAssignments;
19 }
```

Abbildung 5: Pseudocode zur Ermittlung aller Korrekturfunktionen

Beispiel 2. *Zunächst ein Beispiel, wie es zu mehreren Korrekturfunktionen kommen kann. Seien zwei Gatter $G1$ und $G2$ Fehlerkandidaten einer Zweifachfehleranalyse, welche durch einen Pfad verbunden sind. Das bedeutet, dass die Eingangsbelegung von $G2$ abhängig von dem Ausgang (den Ausgängen) des Gatters $G1$ ist. Nun werden bei der Analyse die primären Eingänge und die Ausgänge der Gatter $G1$ und $G2$ verändert. Für beide Gatter existieren nun unterschiedliche Belegungsmöglichkeiten für die Ausgänge. Beispielsweise kann eine Reparatur den Ausgang von $G1$ auf 0 und von $G2$ auf 0 setzen. Die Änderung des Ausganges von $G1$ hat dann direkten Einfluss auf die Belegung der Eingänge von $G2$, und damit auf deren Korrekturfunktion. Damit kann es dann auch zu mehreren Korrekturfunktionen kommen. Dies kann zum Beispiel dem Einfügen eines AND-Gatters bei $G1$ und einem OR-Gatter bei $G2$ bei einer ersten Korrekturfunktion entsprechen, während eine zweite Reparatur einem OR-Gatter an $G1$ und dafür ein AND-Gatter an $G2$ erfordert.*

Bei k gesetzten Fehlerprädikaten und exakt einem Ausgang pro Fehlerprädikat ergeben sich somit bis zu 2^k Kombinationsmöglichkeiten für die Belegung der Ausgänge. Da in dem Verfahren alle n Gegenbeispiele gleichzeitig untersucht werden, und für jedes dieser Gegenbeispiele 2^k Kombinationen möglich sind, ergeben sich insgesamt in der Theorie $2^{k \cdot n}$ mögliche Wertkombinationen. In der Praxis führt aber nur ein geringer Teil zur Korrektur und damit zu einer Korrekturfunktion.

Die Analyse aller Korrekturfunktionen soll Aufschluss geben, ob die theoretisch mögliche Anzahl in der Praxis auch erreicht wird, oder aber ob sie in einem überschaubaren Bereich bleiben.

Der Pseudocode zur Ermittlung aller möglichen Korrekturfunktionen wurde dafür in Abbildung 5 dargestellt und ist eine Erweiterung von Abbildung 3. Es wird nun die Methode $cnf.solve()$ des SAT-Beweisers solange aufgerufen, bis alle Korrekturfunktionen ermittelt worden sind. Um doppelte Korrekturfunktionen zu vermeiden, wird die KNF nach jeder gefundenen Korrekturfunktion um blockierende Klauseln erweitert. Diese beschreiben die aktuelle Belegung der Ein- und Ausgänge für alle Fehlerkandidaten bezüglich aller Gegenbeispiele.

Tabelle 1: Vergleich der Einfach-, Zweifach- und Dreifachfehler-diagnose

Typ	Verfahren	t	$\#ab$	γ
Einfach	Standard	148.41	1357	
	Widerspruchsfrei	199.84	1356	$< 1\%$
	Δ im Mittel	2.57	99%	
Zweifach	Standard	296.10	8224	
	Widerspruchsfrei	2086.76	6721	43%
	Δ im Mittel	6.70	86%	
Dreifach	Standard	968.54	28631	
	Widerspruchsfrei	20673.04	18717	45%
	Δ im Mittel	19.24	78%	

4 Experimentelle Ergebnisse

In diesem Abschnitt werden die theoretischen Ergebnisse durch Experimente belegt. Hierfür wurden 119 Schaltkreise aus der `LGSynth93` Schaltkreisbibliothek genutzt und in jedem dieser Schaltkreise n Fehler, $n = 1, 2, 3$, zufällig injiziert. Es wurde dabei aus acht verschiedenen Fehlertypen zufällig ausgewählt: *gate replacement*, *extra inverter*, *extra wire*, *extra gate*, *missing gate*, *missing inverter*, *missing wire* und *incorrectly placed wire* [1]. Wenn nicht anders beschrieben, wurden für jeden der Schaltkreise zehn Gegenbeispiele mittels SAT-basiertem Äquivalenz-Vergleich erzeugt und bei der weiteren Analyse eingesetzt. Die ausgewählten Schaltkreise enthalten Gatter mit n Eingängen, wobei n auch größer zwei sein kann. Als SAT-Beweiser wurde Zchaff [6] mit inkrementeller SAT Erweiterung [10] eingesetzt.

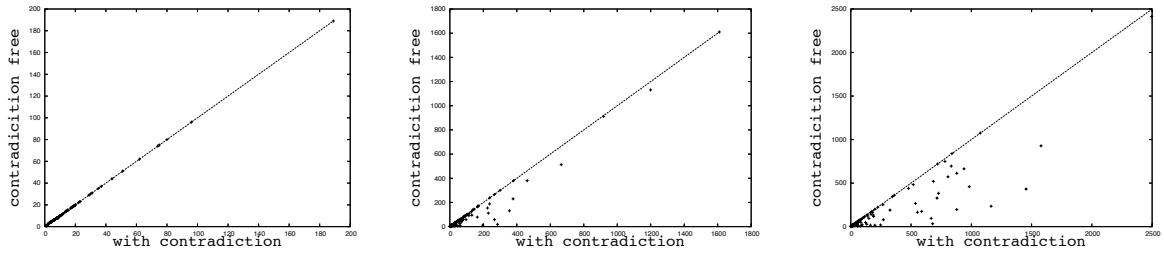
Alle Experimente wurden auf einem AMD Athlon 64 3700+ Prozessor mit 2.2 Ghz und 1 GB Speicher durchgeführt, wobei bei der Ausführung der Speicher auf 512 MB und die Laufzeit auf 3600 Sekunden begrenzt wurde. Die 119 Schaltkreise liegen alle innerhalb dieser Begrenzungskriterien. In den Experimenten werden die drei vorgestellten Ansätze hinsichtlich Laufzeit und Diagnosequalität miteinander verglichen.

Im ersten Teil soll das bisherige Standardverfahren mit der widerspruchsfreien Diagnose verglichen werden. Es wurden dazu die Verfahren mit einer maximalen Anzahl von Einfach-, Zweifach- und Dreifachfehlern auf allen korrespondierenden fehlerhaften 119 Schaltkreisen ausgeführt. Die Ergebnisse sind in Tabelle 1 dargestellt. Es wurden die Berechnungszeit (t), die Anzahl der ermittelten Fehlerkandidaten ($\#ab$) und der Anteil der Schaltkreise mit Widersprüchen in Prozent (γ) angegeben. Die Zeilen "Δ im Mittel" geben die ermittelten zusätzlichen Laufzeiten/Anzahl Fehlerkandidaten pro Schaltkreis als Mittelwert über alle Schaltkreise an.

Bei Einfachfehler-Analyse konnten nur in einem Schaltkreis Widersprüche festgestellt werden. Dort ließen sich die Fehlerstellen nur gering von 9 auf 8 senken. Damit ist die Kosten-Nutzen-Relation bei einer zusätzlichen Laufzeiterhöhung von im Mittel 2.57 sehr gering.

Es konnte bei maximal angenommenen Zweifachfehlern eine Reduzierung der Kandidaten von maximal 94% erreicht werden. Insgesamt wurden bei 43% der Schaltkreise Verbesserungen in der Diagnose um durchschnittlich 14% erreicht. Dabei lag die mittlere Zeiterhöhung für die widerspruchsfreie Diagnose bei einem Faktor von 6.70.

Ähnlich sind auch die Ergebnisse für Dreifachfehler mit Verbesserungen bei 45% der Schaltkreise, einer maximalen Reduzierung von 96% und einer mittleren Reduzierung von



(a) Einfachfehler (b) Zweifachfehler (c) Dreifachfehler

Abbildung 6: Anzahl der ermittelten Fehlerkandidaten mit der Standard- und Widerspruchsfreien-Fehleranalyse

Tabelle 2: Vergleich der Anzahl der Korrekturfunktionen bei Zweifachfehler-Diagnose

Verfahren	t	Anzahl Lösungen
Standard	296.10	8224–Kandidaten
Widerspruchsfrei	2086.76	6721–Kandidaten
Alle Belegungen	3234.14	10758–Korrekturen

22% der Fehlerstellen. Der Aufwand der Dreifachfehler-Diagnose steigt aufgrund der zu verarbeitenden erhöhten Menge an Klauseln um einen Faktor von 19.

Ein Zeitvergleich der Verfahren (Tabelle 1) zeigt, dass mit steigender Anzahl an maximalen Fehlerkandidaten (einfach, zweifach, dreifach) die Laufzeit insbesondere für die widerspruchsfreie Diagnose nicht-linear zunimmt. So benötigt die dreifach widerspruchsfreie Diagnose etwa 10 mal soviel Rechenzeit wie die zweifach Analyse.

Die ermittelte Anzahl an Kandidaten für Fehlerstellen wurde noch einmal in Abbildung 6 zusammenfassend dargestellt. Auf der X-Achse sind die Anzahl Fehlerkandidaten der Standarddiagnose und auf der Y-Achse die der widerspruchsfreien Diagnose für alle 119 Schaltkreise abgetragen. Die Verbesserung der Diagnosegenauigkeit ist insbesondere für Dreifachfehler deutlich zu erkennen.

Die Ermittlung aller Korrekturfunktionen (Tabelle 2) für alle 119 Schaltkreise zeigt einen gering höheren Zeitaufwand im Vergleich zur einfachen widerspruchsfreien Diagnose. Bei etwa 55% höherem Zeitaufwand ergab die vollständige Diagnose dabei im Mittel 1,6 Korrekturfunktionen pro Fehlerkandidat. Somit existieren für den überwiegenden Anteil an Fehlerkandidaten nur ein oder zwei realisierbare Korrekturvorschläge.

Tabelle 3: Vergleich aller Verfahren bei Zweifachfehler-Diagnose

		Standard		Widerspruchsfrei		Alle Belegungen	
Schaltkreis	Anzahl Gatter	t	Anzahl Kandidaten	t	Anzahl Kandidaten	t	Anzahl Korrekturen
5xp1	175	0.65	88	5.84	88	9.63	177
9sym	261	3.54	25	5.26	8	5.32	12
dalu	2625	12.97	238	65.02	238	126.14	543
ex4p	1554	16.03	165	30.52	165	39.00	165
parker1986	2558	28.54	62	38.39	6	37.62	6

Beispielhaft wurden in Tabelle 3 für fünf Schaltkreise die drei Verfahren detailliert gegenübergestellt. Mit moderatem Mehraufwand lässt sich mit der Widerspruchsanalyse für die Schaltkreise 9*sym* und *parker*1986 eine Reduzierung um 68% bzw. 90% der Fehlerkandidaten erreichen. Die Diagnosegenauigkeit ließ sich somit erhöhen. Für die Schaltkreise 5*xp*1, *dalu* und *ex*4*p* liess sich hingegen keiner der Fehlerkandidaten ausschliessen. Die Analyse aller widerspruchsfreien Belegungen für zwei dieser Schaltkreise ergab eine etwa doppelte Anzahl an konsistenten Korrekturmöglichkeiten. Mit Hilfe der Korrekturmöglichkeiten ist es nun möglich gezielt Gatter zur Korrektur auszuwählen.

Zusammenfassend lässt sich feststellen, dass die widerspruchsfreie Diagnose eine Reduzierung der Fehlerkandidaten nicht garantiert, bei einer Reduzierung aber die Fehlerkandidaten mit moderatem Zeitaufwand signifikant verringern kann.

5 Zusammenfassung und Diskussion

Es wurden mit der Widerspruchsanalyse eine Verbesserung der Fehlerdiagnose von [7] durch deterministische Reparaturen erreicht. Mit ihr konnte eine signifikante Reduzierung der Fehlerkandidaten bei über 40% der Schaltkreise erreicht werden. Die Reduzierungsrate lag dabei bei bis zu 96%.

Der Fokus für weitere Arbeiten liegt in der Vervollständigung der partiellen Korrekturfunktion. So werden beispielsweise Belegungen, die nicht zu einem Gegenbeispiel führen, noch nicht in die Korrekturfunktion mit einbezogen. Weiterhin ist zu vermuten, dass die verwendeten Gegenbeispiele nur wenig Unterschiede in den primären Eingängen aufweisen. Es sollten daher möglichst unterschiedliche Gegenbeispiele gewählt werden, um schon mit wenig Gegenbeispielen höhere Diagnosegenauigkeit und damit Reduzierung zu erreichen. Diese Vermutung deckt sich mit den Untersuchungen von [3].

Literatur

[1] M. S. Abadir, J. Ferguson, and T.E. Kirkland. Logic design verification via test generation. *IEEE Trans. on CAD*, 7:172–177, 1988.

[2] M. Ali, S. Safarpour, A. Veneris, M. Abadir, and R. Drechsler. Post-verification debugging of hierarchical designs. In *Int'l Conf. on CAD*, pages 871–876, 2005.

[3] G. Fey and R. Drechsler. Finding good counter-examples to aid design verification. In *MEMOCODE*, pages 51–52, 2003.

[4] S.-Y. Huang and K.-T. Cheng. Errortracer: Design error diagnosis based on fault simulation techniques. *IEEE Trans. on CAD*, 18(9):1341–1352, 1999.

[5] A. Kuehlmann, D. I. Cheng, A. Srinivasan, and D. P. LaPotin. Error diagnosis for transistor-level verification. In *Design Automation Conf.*, pages 218–224, 1994.

[6] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[7] A. Smith, A. Veneris, M.F. Ali, and A. Viglas. Fault diagnosis and logic debugging using boolean satisfiability. *IEEE Trans. on CAD*, 24:1606–1621, 2005.

[8] S. Staber, G. Fey, R. Bloem, and R. Drechsler. Automatic fault localization for property checking. In *Haifa Verification Conference*, volume 4383 of *Lecture Notes in Computer Science*, page not yet appeared. Springer-Verlag, 2006.

[9] A. Veneris and I. N. Hajj. Design error diagnosis and correction via test vector simulation. *IEEE Trans. on CAD*, 18(12):1803–1816, 1999.

[10] J. Whittemore, J. Kim, and K. Sakallah. SATIRE: A new incremental satisfiability engine. In *Design Automation Conf.*, pages 542–545, 2001.