ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG
INSTITUT FÜR INFORMATIK

Lehrstuhl für Rechnerarchitektur
Prof. Dr. Bernd Becker

Bounded Model Checking für unvollständige Designs unter Verwendung von QBF-Codierung

Studienarbeit

Karina Gitina
22. Januar 2010

Inhaltsverzeichnis

1 Einleitung

Digitale Systeme sind heutzutage zu einem festen Bestandteil unseres Alltags geworden. Der technische Fortschritt begleitet uns in fast allen Lebensbereichen, so dass wir von der korrekten Funktionsweise vieler verschiedener technischer Systeme unmittelbar abhängig sind. Vor allem in sicherheitskritischen Systemen, wie in Autos, Flugzeugen oder Maschinensteuerungen, kann deren fehlerhafte Funktionsweise verheerende Folgen haben. Gleichzeitig werden digitale Systeme durch weiter fortschreitende Technologien immer mächtiger und komplexer; viele bestehen heutzutage bereits aus mehreren Millionen von Modulen.

Die korrekte Funktionsweise solch komplexer Systeme sicherzustellen ist keine triviale Aufgabe. Simulation alleine ist nicht ausreichend, da dadurch nur ein kleiner Teil aller möglichen Abläufe untersucht werden kann.

Formale Verifikation ermöglicht es, mit mathematisch exakten Methoden Fehler im System zu finden bzw. die Fehlerfreiheit zu beweisen. Im Wesentlichen kann man die Verifikationsmethoden in zwei Gruppen unterteilen. Eine Gruppe beweist die Äquivalenz zweier Systeme [19]. Diese Verfahren kommen beispielsweise dann zum Einsatz, wenn nach einer Optimierung des Systems gezeigt werden soll, dass sich das funktionale Verhalten nicht verändert hat. Die andere Gruppe befasst sich mit dem Beweisen von Systemeigenschaften. Dabei wird überprüft, ob das System bestimmte Eigenschaften, die in der Spezifikation verlangt werden, aufweist. Diese Eigenschaften werden üblicherweise in einer temporalen Aussagenlogik wie CTL [6] oder LTL [21] spezifiziert.

Model Checking ist ein automatisches, auf Exploration des Zustandsraumes basierendes Verfahren zum Beweis von Systemeigenschaften. Ein großes Problem für die Anwendbarkeit von Model Checking ist die Größe praktisch relevanter Systeme: Die Zahl der Zustände wächst exponentiell in der Anzahl der speichernden Elemente. Dadurch sind Verfahren, die auf dem Durchlaufen eines explizit dargestellten Zustandsraums basieren, nur auf sehr kleine Systeme anwendbar. Abhilfe schaffen symbolische Methoden, z. B. die implizite Darstellung des Zustandsraums mit Hilfe binärer Entscheidungsdiagramme (BDDs) [3]. Doch leider scheitern auch diese Verfahren für viele praktisch relevante Schaltungen, da BDDs bei manchen Schaltungen, wie z.B. Multiplizierer, exponentiell groß in der Zahl der Variablen werden können [4].

Ein in der Praxis sehr erfolgreicher Ansatz zum Wiederlegen von Eigenschaften ist Bounded Model Checking (BMC) [1, 2]. Dabei wird das System bis zur einer vorgegebenen Grenze k induktiv abgerollt und als aussagenlogisches Erfüllbarkeitsproblem (SAT) formuliert. Jede BMC-Instanz wird mit Hilfe eines SAT-Solvers gelöst. Da das Erfüllbarkeitsproblem NP-vollständig ist, kann dies auch exponentielle Laufzeit in Anspruch nehmen. Dank großer Fortschritte bei SAT-Solvern führt dieses Verfahren in der Praxis jedoch in vielen Fällen zum Erfolg.

Diese Arbeit beschäftigt sich mit BMC von Schaltkreisen, die nicht vollständig implementiert/spezifiziert sind. Die unbekannten Teilschaltkreise werden als Blackboxes, deren Eingänge und Ausgänge, aber nicht deren Funktion bekannt sind, bezeichnet. In der Praxis werden sie bei der Verifikation zu einem frühem Zeitpunkt eingesetzt, wenn noch nicht der ganze Schaltkreis vorliegt. Eine andere Anwendung können Blackboxes bei der Fehlerlokalisation während des Entwicklungsprozesses finden: Sucht man einen Fehler, legt man nacheinander über mögliche Fehlerorte Blackboxes. Wenn der Fehler dabei verschwindet, ist ein Fehlerort gefunden (Fehler liegt in der Blackbox). Nicht zuletzt können Blackboxes zur Effizienzsteigerung bei der Verifikation eingesetzt werden. Dafür werden irrelevante Module, die die Gültigkeit der Eigenschaft nicht beeinflussen (sollten), aber teuer zu verifizieren sind (beispielsweise Speicher oder Multiplizierermodule) ausgeblendet.

Mit BMC für unvollständige Designs kann überprüft werden, ob für alle möglichen Ersetzungen der Blackboxes die geforderte Invariante (eine Eigenschaft, die in jedem Zustand gelten muss) verletzt ist. Ist dies der Fall, sind die bereits vorhandenen Schaltungsteile fehlerhaft.

Für die Codierung der BMC Instanzen sind zwei verschiedene Vorgehensweisen möglich. Zum Einen kann man eine BMC-Abrollung als SAT-Problem formulieren [12, 10]. Um hier das unbekannte Verhalten der Blackboxes zu modellieren, wird ein zusätzlicher logischer Wert X eingeführt. Dieses Verfahren ist zwar effizient, allerdings für manche Probleme ungenau: In einigen Fällen, in denen die Invariante nicht realisierbar ist, wird kein Gegenbeispiel gefunden.

Um dieses Problem zu umgehen, kann das BMC-Problem als *quantifizierte* Boolesche Formel (QBF) formuliert werden, wobei das unbekannte Verhalten der Blackboxes durch universelle Quantifizierung deren Ausgänge modelliert wird. Eine QBF zu lösen ist aufwändiger (genauer gesagt ist es PSPACE-vollständig), bietet aber in diesem Kontext eine höhere Genauigkeit.

Ziele dieser Arbeit In dieser Studienarbeit werden wir verschiedene SAT- und QBF-Formulierungen des BMC-Problems für unvollständige Schaltkreise untersuchen. Für die QBF-Formulierung sollen mehrere alternative Codierungen erzeugt werden, die in unterschiedlich mächtigen Gegenbeispielen resultieren. Weiterhin soll es möglich sein, für verschiedene Blackboxes unterschiedliche Formulierungen zu verwenden, d. h. manche Blackboxes werden mit SAT, andere mit QBF modelliert. Dazu erweitern wir ein vorhandenes SAT-basiertes BMC-Tool für unvollständige Schaltkreise so, dass damit auch verschiedene QBF-Formulierungen des BMC-Problems erzeugt und mit einer Reihe von QBF-Solvern gelöst werden können. Die Effizienz und die Genauigkeit der verschiedenen Ansätze zur Modellierung der Blackboxes sollen an Hand einiger Benchmarks ausgewertet werden.

Gliederung dieser Arbeit Als nächstes werden im Kapitel 2 die für das Verständnis dieser Arbeit notwendige Grundlagen vermittelt. Dies sind insbesondere die Grundlagen des Erfüllbarkeitsproblems und Quantifizierten Booleschen Formeln. Weiterhin gehen

wir auf BMC für Schaltkreise ein. Die Kapitel 3 und 4 sind BMC für unvollständig spezifizierte Schaltkreise gewidmet. In Kapitel 3 werden wir zeigen, wie BMC auch in diesem Fall als SAT-Problem formuliert werden kann und auf dessen Probleme eingehen. In Kapitel 4 stellen wir vor, wie verschieden mächtige QBF-Formulierungen für BMC unvollständiger Schaltkreise erzeugt werden können. Im fünften Kapitel werden wir die experimentellen Ergebnisse an Hand verschiedener Benchmarks und deren Auswertung vorstellen. Schließlich folgt die Zusammenfassung und der Ausblick im Kapitel 6.

2 Grundlagen

Dieses Kapitel behandelt die Grundlagen für BMC für unvollständige Designs. Dabei handelt es sich um das aussagenlogische Erfüllbarkeitsproblem, Quantifizierte Boolesche Formeln und den BMC-Algorithmus an sich. Dieser wird dann in den nächsten Kapiteln für unvollständige Designs erweitert.

2.1 SAT

Das Erfüllbarkeitsproblem für aussagenlogische Formeln ist ein zentrales Problem in der Informatik. Viele praktisch relevante Probleme wie z. B. der Äquivalenz-Beweis zweier Schaltkreise oder das BMC-Problem, lassen sich als Erfüllbarkeitsprobleme formulieren. In der theoretischen Informatik war es das erste Problem, für das nachgewiesen wurde, dass es NP-vollständig ist [5].

Definition 2.1
*Sei $X = \{x_1, \ldots, x_n\}$ eine Menge Boolescher Variablen. Zu einer Variable $x_i \in X$ ist x_i das **positive** und $\neg x_i$ das **negative Literal**. Eine **Klausel** $K = (l_1 \vee \cdots \vee l_n)$ ist eine Disjunktion von Literalen l_i. Eine Formel φ ist in **konjunktiver Normalform (KNF)**, wenn sie eine Konjunktion $(K_1 \wedge \cdots \wedge K_m)$ von Klauseln K_i $(1 \leq i \leq m)$ ist.*

Definition 2.2
*Das **SAT-Problem** besteht darin zu entscheiden, ob es für eine vorgegebene Formel $\varphi(X)$ in KNF eine Variablenbelegung β von Variablen X gibt, so dass $\beta \vDash \varphi(X)$ gilt.*

Im Folgenden setzen wir stets voraus, dass die Formel, deren Erfüllbarkeit überprüft werden soll, in KNF gegeben ist. Jede Boolesche Funktion lässt sich in KNF transformieren. Dabei kann es allerdings passieren, dass die Formel exponentiell länger wird, wie man an folgendem Beispiel sieht.

Beispiel 2.1
Betrachten wir die Funktion

$$f_4(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4.$$

Eine KNF minimaler Länge für diese Funktion ist gegeben durch

$$\begin{aligned} f_4(x_1, x_2, x_3, x_4) = {} & (\neg x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge \\ & (\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge \\ & (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4) \wedge \\ & (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4) \end{aligned}$$

Für das Exklusiv-Oder von vier Variablen benötigen wir also mindestens 8 Klauseln mit jeweils 4 Literalen. Bei n Variablen wären es 2^{n-1} Klauseln mit jeweils n Literalen.

Wollen wir zu einer Booleschen Funktion eines vorgegebenen Schaltkreises eine äquivalente Darstellung in KNF, stoßen wir schnell an unsere Grenzen: Die Länge der Formel kann exponentiell in der Größe des Schaltkreises sein. Abhilfe schafft die sogenannte Tseitin-Transformation [24], die durch Einführung von Hilfsvariablen eine erfüllbarkeitsäquivalente Darstellung erzeugt, deren Länge linear in der Größe des Schaltkreises ist. Auf die Tseitin-Transformation werden wir in Kapitel 2.3 genauer eingehen.

2.2 Quantifizierte Boolesche Formeln

Eine Erweiterung des SAT-Problems sind Quantifizierte Boolesche Formeln, bei denen die Variablen entweder existentiell ($\exists$) oder universell ($\forall$) quantifiziert werden. Dies wird im sogenannten Quantorenpräfix spezifiziert.

Definition 2.3
*Eine **Quantifizierte Boolesche Formel (QBF)** ist eine Formel ψ der Form:*

$$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n : \varphi(x_1, x_2, \ldots, x_n),$$

wobei $\varphi(x_1, x_2, \ldots, x_n)$ eine Boolesche Formel ist, in der genau die Booleschen Variablen $x_1, x_2, \ldots, x_n$ vorkommen. $Q_i \in \{\forall, \exists\}$ sind universelle ($\forall$) bzw. existentielle ($\exists$) Quantoren, $i \in \{1, \cdots, n\}$, $Q_1 x_1 Q_2 x_2 \cdots Q_n x_n$ ist das Quantorenpräfix.

Wir setzen wie auch bei SAT voraus, dass φ in KNF gegeben ist.

Für eine Boolesche Formel φ und einen Wert $v \in \{0, 1\}$ sei $\varphi[v/x]$ die Formel, die aus φ entsteht, indem jedes Vorkommen der Variablen x durch die Konstante v ersetzt wird. Dann ist die Semantik einer QBF wie folgt definiert.

Definition 2.4
*Sei $\psi = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n : \varphi(x_1, x_2, \ldots, x_n)$ eine Quantifizierte Boolesche Formel. Wir definieren induktiv, wann ψ **gültig** ist (geschrieben $\vDash \psi$):*

$$\vDash \exists x_1 Q_2 x_2 \cdots Q_n x_n : \varphi \quad \Leftrightarrow \quad \vDash Q_2 x_2 \cdots Q_n x_n : \varphi[0/x_1] \text{ oder } \vDash Q_2 x_2 \cdots Q_n x_n : \varphi[1/x_1]$$
$$\vDash \forall x_1 Q_2 x_2 \cdots Q_n x_n : \varphi \quad \Leftrightarrow \quad \vDash Q_2 x_2 \cdots Q_n x_n : \varphi[0/x_1] \text{ und } \vDash Q_2 x_2 \cdots Q_n x_n : \varphi[1/x_1].$$

*Ist ψ nicht gültig, nennen wir sie **ungültig**.*

Beispiel 2.2
Die QBF

$$\forall x \exists y : (x \vee y) \wedge (\neg x \vee \neg y)$$

ist gültig, denn für jeden x-Wert (egal, ob 1 oder 0) existiert mindestens ein y-Wert (d. h. 0 bzw. 1), so dass die Formel erfüllbar ist. Die QBF

$$\exists y \forall x : (x \vee y) \wedge (\neg x \vee \neg y)$$

dagegen ist ungültig, denn es gibt keinen festen y-Wert, so dass unabhängig vom Wert von x die Formel $(x \vee y) \wedge (\neg x \vee \neg y)$ erfüllt ist.

Definition 2.5
*Das **QBF-Problem** besteht darin zu entscheiden, ob eine gegebene QBF gültig ist.*

Das QBF-Problem ist PSPACE-vollständig [7, Abschn. 7.4].

Es gibt verschiedene Verfahren um die Gültigkeit von quantifizierten Booleschen Formel zu entscheiden, siehe z. B. [8, 17, 20].

2.3 Bounded Model Checking

BMC [1, 2] ist ein SAT-basiertes Verfahren, um Invarianteneigenschaften für Schaltkreise zu falsifizieren. Dabei wird die Existenz eines Pfades fester Länge k, der die Invariante verletzt, als ein Erfüllbarkeitsproblem formuliert. Dieses Erfüllbarkeitsproblem wird mit Hilfe eines SAT-Solvers gelöst. Liefert der Solver zurück, dass das Problem erfüllbar ist, entspricht die erfüllende Belegung der Variablen der primären Eingänge einem Gegenbeispiel, d.h. einem Pfad, der im Anfangszustand beginnt und in einem Zustand endet, in dem die Invariante nicht gilt.

Sei $T(s^i, x^i, s^{i+1})$ die Transitionsrelation, die genau dann erfüllt ist, wenn s^{i+1} der Nachfolgezustand von s^i bei Eingabe von x^i ist. Außerdem konstruiert man eine Formel $I(s^0)$, die erfüllt ist, wenn s^0 der Anfangszustand des Schaltkreises ist. Die zu falsifizierende Invarianteneigenschaft wird ebenfalls als Formel $P(s^k)$ kodiert, so dass $P(s^k)$ erfüllt ist, wenn im Zustand s^k die Invariante gilt.

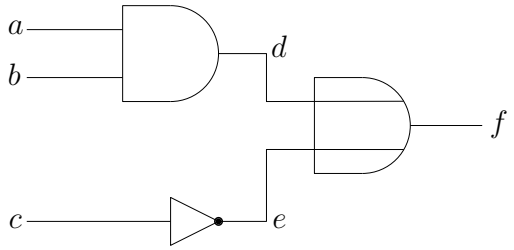
Formel 2.1 beschreibt Pfade einer vorgegebenen Länge k, die im Anfangszustand s^0 beginnen und in einem Zustand s^k, der die Invariante verletzt, enden.

$$BMC(k) = I(s^0) \wedge \bigwedge_{i=0}^{k-1} T(s^i, x^i, s^{i+1}) \wedge \neg P(s^k) \tag{2.1}$$

Um die Invarianteneigenschaft zu widerlegen, beginnt man mit $k = 0$. Hat man $BMC(k)$ konstruiert und mittels Tseitin-Transformation die Formel in KNF gebracht, dann prüft man ihre Erfüllbarkeit mit Hilfe eines SAT-Solvers. Falls der Solver eine erfüllende Belegung findet, entspricht diese einem Gegenbeispiel für die Invarianteneigenschaft. Falls die Formel unerfüllbar ist, erhöht man die Abwicklungstiefe k um 1 und beginnt von Neuem.

Wir wollen nun die Funktionsweise der Tseitin-Transformation an Hand eines Beispiels erläutern.

Beispiel 2.3
Wir betrachten einen Schaltkreis mit drei Gattern, Eingangssignalen a, b, c, einem Ausgangssignal f und neu eingeführten Hilfsvariablen für interne Signale d und e. Wir wollen eine KNF erzeugen, die genau dann erfüllbar ist, wenn es eine Eingangsbelegung des Schaltkreises gibt, so dass der Ausgang f den Wert 1 annimmt.



Es gilt, dass das Signal d genau dann mit 1 belegt ist, wenn sowohl a als auch b den Wert 1 trägt. Dies führt zur Formel $d \leftrightarrow (a \wedge b)$. Entsprechend erhalten wir für die anderen Gatter $e \leftrightarrow \neg c$ und $f \leftrightarrow (d \vee e)$. Diese drei Äquivalenzen formen wir in konjunktive Normalform um. Dabei erhalten wir folgende Formeln:

$$\begin{array}{rcccl} d \leftrightarrow (a \wedge b) & \equiv & (d \rightarrow (a \wedge b)) \wedge ((a \wedge b) \rightarrow d) & \equiv & (a \vee \neg d) \wedge (b \vee \neg d) \wedge (d \vee \neg a \vee \neg b) \\ e \leftrightarrow \neg c & \equiv & (e \rightarrow \neg c) \wedge (\neg c \rightarrow e) & \equiv & (\neg c \vee \neg e) \wedge (c \vee e) \\ f \leftrightarrow (d \vee e) & \equiv & (f \rightarrow (d \vee e)) \wedge ((d \vee e) \rightarrow f) & \equiv & (d \vee e \vee \neg f) \wedge (f \vee \neg d) \wedge (f \vee \neg e) \end{array}$$

Die konsistenten Belegungen im der Signale lassen sich durch die Konjunktion dieser Formeln beschreiben, d. h. durch

$$(a \vee \neg d) \wedge (b \vee \neg d) \wedge (d \vee \neg a \vee \neg b) \wedge (\neg c \vee \neg e) \wedge (c \vee e) \wedge (d \vee e \vee \neg f) \wedge (f \vee \neg d) \wedge (f \vee \neg e)$$

Wir wollen aber nicht nur, dass die Signale im Schaltkreis konsistent belegt sind, sondern zusätzlich, dass das Ausgangssignal f den Wert 1 hat. Dies erreichen wir, indem wir die Konjunktion der obigen Formel mit der Variablen f bilden.

$$(a \vee \neg d) \wedge (b \vee \neg d) \wedge (d \vee \neg a \vee \neg b) \wedge (\neg c \vee \neg e) \wedge (c \vee e) \wedge (d \vee e \vee \neg f) \wedge (f \vee \neg d) \wedge (f \vee \neg e) \wedge f$$

Man sieht, dass die Länge dieser Formel und die Zahl der Hilfsvariablen linear in der Anzahl der Gatter des Schaltkreises sind.

Nachdem wir nun die Grundlagen kennengelernt haben, werden wir uns in den nächsten beiden Kapiteln damit beschäftigen, wie die Existenz eines Gegenbeispiels bei unvollständigen Designs mittels SAT bzw. QBF geprüft werden kann.

3 Bounded Model Checking für unvollständige Designs unter Verwendung von SAT

Im Folgenden beschäftigen wir uns mit BMC für *unvollständige* Designs, bei denen Teile des Schaltkreises unbekannt sind. Diese Teile bilden die sogenannten Blackboxes, deren Ein- und Ausgänge bekannt sind, nicht aber deren Funktion. Abbildung 3.1 zeigt ein unvollständiges Design mit Eingang x, Ausgang y, vier speichernden Elementen und einer Blackbox mit einem Ausgang.

Für unvollständige Designs müssen wir bei BMC eine neue Fragestellung betrachten.

Definition 3.1
Eine Invariante ϕ heißt für ein unvollständiges Design ***nicht-realisierbar****, wenn es keine Ersetzung der Blackboxes durch Schaltkreise gibt, so dass das entstehende vollständige Design die Invariante ϕ erfüllt.*

Da die Werte der Blackboxausgänge unbekannt sind, benötigen wir hierfür eine geeignete Modellierung. Dieses Kapitel behandelt die Modellierung von unvollständigen Designs mit der dreiwertigen Logik über $\{0, 1, X\}$, die bei manchen BMC-Problemen zu ungenau ist, sich aber (wie bei BMC für vollständige Designs) durch eine SAT-Formel codieren lässt.

3.1 Dreiwertige Logik

Anstelle der Booleschen (zweiwertigen) Logik $\{0, 1\}$ verwenden wir zur Modellierung von Schaltkreisen mit Blackboxes die dreiwertige Logik $\{0, 1, X\}$. Dabei wird den Blackboxausgängen der Wert X zugewiesen. Die Booleschen Operationen $\neg$, $\wedge$ und $\vee$ lassen sich, wie in der Tabelle 3.1 dargestellt, auf die dreiwertige Logik übertragen.

Wenn wir BMC auf unvollständig spezifizierte Schaltkreise anwenden wollen, um die Nicht-Realisierbarkeit einer Eigenschaft zu beweisen, müssen wir die Existenz eines Gegenbeispiels vorgegebener Länge als Erfüllbarkeitsproblem formulieren. Dieses Problem für dreiwertige Logik kann mit Hilfe der Jain-Codierung [15] auf die Boolesche Logik reduziert werden.

3.2 Jain-Codierung

Die Idee bei der Jain-Codierung für dreiwertige Logik ist, die drei logischen Werte mit zwei booleschen Signalen zu codieren. Anstelle jedes dreiwertigen Signals haben wir also

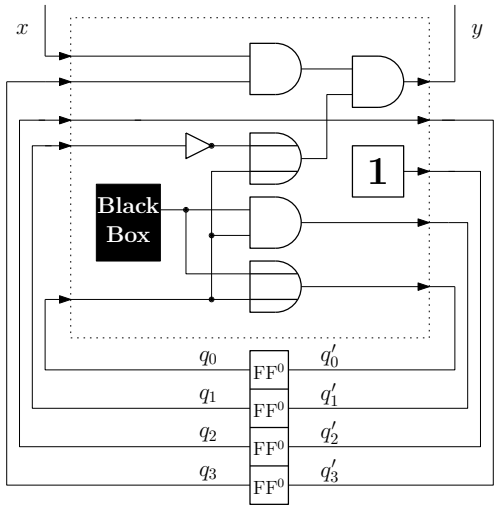


Abbildung 3.1: Schaltkreis mit einer Blackbox

a	b	$\neg a$	$a \wedge b$	$a \vee b$
0	0	1	0	0
0	1	1	0	1
0	X	1	0	X
1	0	0	0	1
1	1	0	1	1
1	X	0	X	1
X	0	X	0	X
X	1	X	X	1
X	X	X	X	X

Tabelle 3.1: Wertetabelle für dreiwertige Logik

ein Paar zweiwertiger Signale nämlich $1_{01X} = (1, 0)$, $0_{01X} = (0, 1)$ und $X_{01X} = (0, 0)$.

Der Wert $(1, 1)$ wird für die Codierung nicht benötigt und ist deshalb ungültig. Wir können die Operationen AND, OR und NOT nun auf Wertepaaren angeben, so dass sie den entsprechenden Operationen für die dreiwertige Logik (siehe Tabelle 3.1) entsprechen.

Seien $a = (a_{\text{high}}, a_{\text{low}})$ und $b = (b_{\text{high}}, b_{\text{low}})$ zwei Jain-codierte Variablen. Dann können wir die Operationen AND, OR und NOT auf folgende Weise ausdrücken:

$$\begin{aligned} \text{NOT}_{01X}(a) &= (a_{\text{low}}, a_{\text{high}}) \\ \text{AND}_{01X}(a, b) &= (a_{\text{high}} \wedge b_{\text{high}}, a_{\text{low}} \vee b_{\text{low}}) \\ \text{OR}_{01X}(a, b) &= (a_{\text{high}} \vee b_{\text{high}}, a_{\text{low}} \wedge b_{\text{low}}) \end{aligned}$$

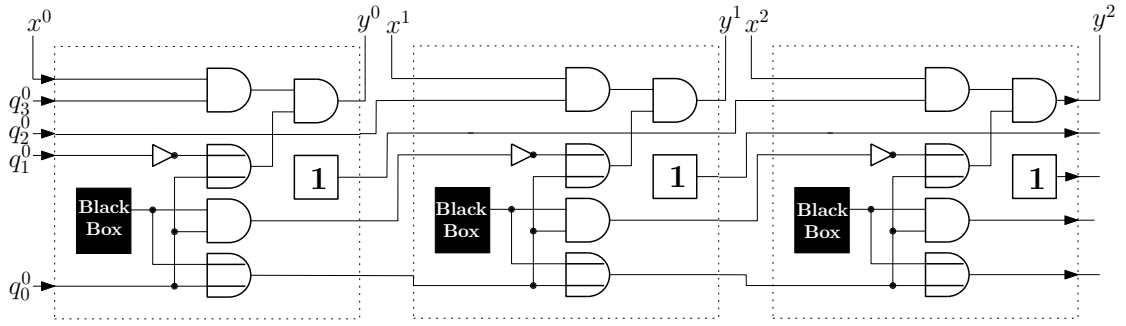


Abbildung 3.2: Dreimalige Abwicklung des Schaltkreises aus Abbildung 3.1

Wir werden für die effizientere Codierung die Booleschen Operationen auch für gemischte Signaltypen benötigen, wenn ein Eingang Jain-codiert ist und der andere nicht. Dafür können wir die benötigten zweistelligen Operationen leicht anpassen. Sei f eine boolesche (zweiwertige) Variable. Dann entspricht ihr das Jain-codierte Paar $(f, \neg f)$. Setzen wir dies oben ein, erhalten wir

$$\mathrm{AND}_{01\mathrm{X}}(a, f) = \mathrm{AND}_{01\mathrm{X}}\big(a, (f, \neg f)\big) = (a_{\mathrm{high}} \wedge f, a_{\mathrm{low}} \vee \neg f)$$
$$\mathrm{OR}_{01\mathrm{X}}(a, f) = \mathrm{OR}_{01\mathrm{X}}\big(a, (f, \neg f)\big) = (a_{\mathrm{high}} \vee f, a_{\mathrm{low}} \wedge \neg f)$$

3.3 BMC mit Jain-Codierung

Anhand eines ausführlichen Beispiels werden wir zeigen, wie mit Hilfe der Jain-Codierung BMC auf unvollständige Schaltkreise angewendet werden kann und warum diese Art der Codierung zu ungenau sein kann [12, 10].

Beispiel 3.1
Betrachten wir den Schaltkreis in Abbildung 3.1. Am Anfang seien alle Speicherelemente mit 0 initialisiert. Wir wollen zeigen, dass die Invariante $AG\neg y$ (der Ausgang y ist nie 1) unabhängig von der Implementierung der Blackbox verletzt ist. Das kürzeste Gegenbeispiel hat die Länge 3. Deshalb gehen wir hier nur auf die Abwicklungstiefe $k = 2$ ein. Abbildung 3.2 zeigt den Schaltkreis, den wir durch dreimaliges Abwickeln enthalten.

Wir entfernen nun aus diesem Schaltkreis all diejenigen Teile, die auf den Ausgang y^2 keinen Einfluss haben, indem wir ausgehend von y^2 rückwärts laufen und die nicht erreichten Teile des Schaltkreises entfernen (Cone-of-influence reduction). Die Teile des Schaltkreises, die dann übrig bleiben, sind in Abbildung 3.3 dargestellt.

Überlegen wir uns kurz, dass die Eigenschaft $AG\neg y$ in diesem Schaltkreis tatsächlich verletzt ist. Nehmen wir an, die linke Blackbox gebe den Wert $Z^0 = 0$ aus. Da $q_0^0 = 0$ ist (alle Flipflops haben den Initialwert 0), folgt $a = 0$, $b = 0$ und damit $d = 1$. Deshalb nimmt das Signal f den Wert 1 an. Setzen wir den Eingang $x^2 = 1$, so folgt $y^2 = 1$ und die Eigenschaft ist verletzt. Gibt die linke Blackbox hingegen den Wert $Z^0 = 1$ aus, folgt $a = 1$, $c = 1$ und $f = 1$. Mit $x^2 = 1$ folgt wiederum $y^2 = 1$. Damit ist die Eigenschaft

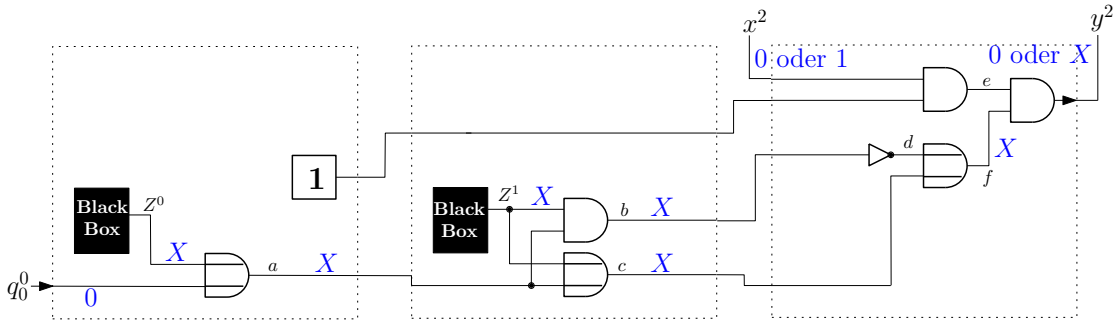


Abbildung 3.3: Wir entfernen alle Teile des Schaltkreises aus Abbildung 3.2, die keinen Einfluss auf den Wert von y^2 haben

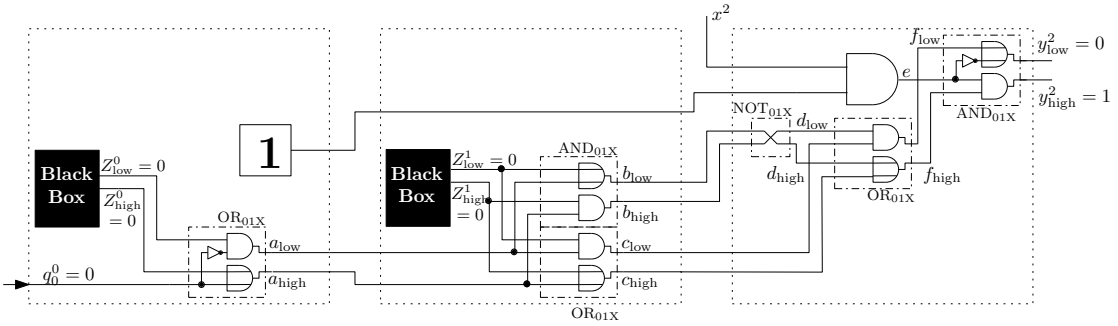


Abbildung 3.4: Jain-Codierung im Schaltkreis aus Abbildung 3.3 eingeführt

unabhängig vom Wert der Blackboxausgänge verletzt, sofern wir $x^2 = 1$ setzen. Da die Werte der Blackboxes jedoch unbekannt sind, versehen wir diese mit X.

Im nächsten Schritt führen wir die Jain-Codierung für die beiden Blackboxes ein. Dazu müssen wir die ausgehenden Leitungen der Blackboxes verdoppeln. Dasselbe müssen wir mit allen ausgehenden Leitungen von Gattern tun, bei denen mindestens eine Eingangsleitung verdoppelt wurde. Dazu laufen wir ausgehend von den verdoppelten Blackbox-Ausgängen vorwärts und verdoppeln alle Leitungen, die wir auf dem Weg zu y^2 entlanglaufen. Die Gatter auf diesem Weg ersetzen wir durch AND_{01X}, OR_{01X} bzw. NOT_{01X}. Dadurch erhalten wir den Schaltkreis in Abbildung 3.4. Den Ausgängen der Blackboxes weisen wir den Wert $X_{01X} = (0, 0)$ zu, dem Eingang q_0^0 den Initialwert 0 des Flipflops. Dies erreichen wir, indem wir für diese Signale entsprechende Unit-Klauseln hinzufügen. Wenn die Leitung $Z_{\text{low}} = 0$ sein soll, fügen wir die Unit-Klausel $(\neg Z_{\text{low}})$ hinzu. Entsprechend verfahren wir für die anderen Leitungen, die festen Werten erhalten sollen.

Um die Invariante $AG\neg y$ im Schritt 2 zu verletzen, muss $y^2 = 1$ gelten. Da diese Leitung verdoppelt wurde, muss der Wert für $(y^2_{\text{high}}, y^1_{\text{low}}) = 1_{01X} = (1, 0)$ sein. Diese Werte werden durch entsprechende Unit-Klauseln sichergestellt.

Auf den so entstandenen Schaltkreis wenden wir die Tseitin-Transformation an, um eine Formel in konjunktiver Normalform zu erhalten. Wenn sie erfüllbar ist, ist die

Invariante für alle möglichen Blackbox-Implementierungen verletzt. Allerdings liefert der SAT-Solver das Ergebnis UNSAT, wie man folgendermaßen einsehen kann. Wegen $Z^0_{\text{low}} = 0$ folgt $a_{\text{low}} = 0$, und wegen $q^0_0 = 0$ und $Z^0_{\text{high}} = 0$ gilt $a_{\text{high}} = 0$. Entsprechend kann man wegen $Z^1_{\text{high}} = 0$ und $a_{\text{high}} = 0$ auf $c_{\text{high}} = 0$ schließen und wegen $a_{\text{low}} = 0$ auf $c_{\text{low}} = 0$. Entsprechend erhalten wir $b_{\text{low}} = b_{\text{high}} = 0$. Damit gilt auch $d_{\text{low}} = d_{\text{high}} = 0$. Zusammen mit $c_{\text{high}} = 0$ implizieren sie $f_{\text{low}} = f_{\text{high}} = 0$. Folglich gilt $y^2_{\text{high}} = 0$. Damit muss das Ergebnis des SAT-Solver-Aufrufs UNSAT sein.

Das sehen wir auch, wenn wir nochmals Abbildung 3.3 betrachten. Wir haben dort die logischen Werte eingetragen, die zum Ausgang propagiert werden. Die Ausgänge der beiden Blackboxes haben per Definition den Wert X. Dadurch erhalten auch die anderen Leitungen zwischen den Blackboxes und dem Ausgang diesen Wert. Den logischen Wert 1 zu erhalten, ist so nicht möglich.

Wie wir im obigen Beispiel gesehen haben, kann es passieren, dass Eigenschaften unabhängig von der Ausgabe der Blackboxes verletzt sind, der Ansatz mit 01X-SAT aber dennoch UNSAT liefert. Dies kann zwei verschiedene Ursachen haben. Zum einen werden Abhängigkeiten zwischen Signalen nicht berücksichtigt. Es gilt $a \wedge \neg a = 0$ unabhängig vom Wert von a. Bei der dreiwertigen Logik gilt jedoch, wenn $a = X$ ist: $\neg a = X$ und $\text{AND}_{\text{01X}}(a, \neg a) = \text{AND}_{\text{01X}}(X, X) = X$. Wären die beiden Argumente von AND_{01X} zwei von einander unabhängige Signale, die beide einen unbekannten Wert haben, so wäre die die Ausgabe X tatsächlich der einzig sinnvolle Wert. Da wir aber wissen, dass a und $\neg a$ immer entgegengesetzte logische Werte tragen, könnten wir folgern, dass das Ergebnis 0 sein muss. Dies ist der Grund, warum der 01X-SAT-Ansatz im obigen Beispiel fehlschlägt.

Der andere Grund, warum der 01X-SAT-Ansatz kein Ergebnis liefern kann, ist, dass Eingabewerte manchmal in Abhängigkeit von den genauen Ausgabewerten der Blackboxes in den vorigen Zeitschritten gewählt werden müssen, um einen Zustand zu erreichen, in dem die Invariante nicht gilt.

Im nächsten Kapitel werden wir ein Beispiel sehen, in dem dies der Fall ist.

Beide Probleme können durch die Verwendung quantifizierter Boolescher Formeln gelöst werden – im Allgemeinen jedoch auf Kosten der Laufzeit. Darauf werden wir im nächsten Kapitel eingehen.

4 Bounded Model Checking für unvollständige Designs unter Verwendung von QBF

In diesem Kapitel werden wir uns mit BMC für unvollständige Schaltkreise mittels QBF-Codierung beschäftigen. Wir werden zwei Varianten der Formulierung des Quantorenpräfixes vergleichen und eine neue Möglichkeit betrachten, die QBF-Codierung mit der 01X-Codierung zu kombinieren.

4.1 Reine QBF-Formulierungen

Bei der reinen QBF-Formulierung werden alle Signale des Schaltkreises – im Gegensatz zum 01X-Verfahren – mit nur einer Variable codiert. Die Blackboxausgänge müssen all-quantifiziert werden, da das Gegenbeispiel für alle möglichen Werte der Blackboxausgänge existieren soll. Die übrigen Variablen werden existenz-quantifiziert. Da die Werte der Blackboxausgänge im Zeitschritt i von den Werten der primären Eingänge in den Zeitschritten $j \leq i$ abhängen können, müssen die Eingänge für alle Zeitschritte $j \leq i$ im Quantorenpräfix vor den Blackboxausgängen zum Zeitschritt i stehen.

Um diesen QBF-Ansatz formal zu beschreiben, werden wir zunächst einige Bezeichnungen einführen.

Sei $BMC(k)$ die BMC-Formel für den zu untersuchenden Schaltkreis. Mit I^j bezeichnen wir die Menge der Variablen für die primären Eingänge in Zeitschritt j, mit B^j die Variablen für die Blackbox-Ausgänge in Zeitschritt j und mit H^j die entsprechenden Hilfsvariablen, die durch die Tseitin-Transformation eingeführt werden. Die Werte der letzteren werden durch die Werte der Eingangsvariablen und der Blackboxausgänge impliziert. Wir setzen außerdem

$$I = \bigcup_{j=0}^{k} I^j \qquad B = \bigcup_{j=0}^{k} B^j \qquad H = \bigcup_{j=0}^{k} H^j.$$

Gegeben die oben beschriebene Partitionierung der Variablen in $BMC(k)$, können wir zwei verschiedene Quantorenpräfixe definieren, die die oben beschriebene Bedingung an die Reihenfolge erfüllen:

- *nicht-uniformes Präfix:*

$$\exists I^0 \forall B^0 \exists H^0 \exists I^1 \forall B^1 \exists H^1 \ldots \exists I^k \forall B^k \exists H^k : BMC(k)$$

- *uniformes Präfix:*

$$\exists I \forall B \exists H : BMC(k)$$

Wir werden nun näher auf jede Variante eingehen und die Unterschiede verdeutlichen.

4.1.1 Nicht-uniforme Quantorenpräfixe

Wie schon oben erwähnt, haben nicht-uniforme Quantorenpräfixe die Form:

$$\exists I^0 \forall B^0 \exists H^0 \exists I^1 \forall B^1 \exists H^1 \ldots \exists I^k \forall B^k \exists H^k : BMC(k)$$

Zuerst werden also die primären Eingänge im Zeitschritt 0 belegt.[1] Dann werden für alle möglichen Werte der Blackboxausgänge in Zeitschritt 0 die primären Eingänge in Zeitschritt 1 belegt, usw. Diese Alternierung zwischen primären Eingängen mit Existenz-Quantor und Blackboxausgängen mit All-Quantor setzt sich bis zum Zeitschritt k fort. Das bedeutet, die Werte für jeden primären Eingang in jedem Zeitschritt werden erst gewählt, wenn die Werte aller Variablen der vorhergehenden Zeitschritte schon bekannt sind.

Wie wir weiter in Abschnitt 4.1.3 an einem Beispiel demonstrieren werden, eignet sich diese QBF-Formulierung sehr gut für die Gegenbeispielsuche. Allerdings hängt die Anzahl der Quantorenwechsel bei nicht-uniformen Präfixen von der maximalen Abwicklungstiefe der Schaltung ab und beträgt $2 \cdot (k+1)$ bei der maximalen Abwicklungstiefe k.

4.1.2 Uniforme Quantorenpräfixe

Die uniformen Quantorenpräfixe haben die Form:

$$\exists I \forall B \exists H : BMC(k).$$

Man legt also ganz am Anfang die Belegung der primären Eingänge für alle Zeitschritte auf einmal fest, dann wird für alle möglichen Blackboxausgaben aller Zeitschritte versucht, eine konsistente Belegung der Hilfsvariablen zu finden.

Diese QBF-Formulierung kann in manchen Fällen dazu führen, dass kein Gegenbeispiel gefunden wird, obwohl eins existiert (siehe das Beispiel in Abschnitt 4.1.3). Trotzdem sind uniforme Quantorenpräfixe von großem Interesse, denn unabhängig von der Abwicklungstiefe hat ein uniformes Präfix nur zwei Quantorenwechsel. Das lässt uns leichtere QBF-Probleme erwarten.

Ein weiterer Vorteil der uniformen Quantifizierung liegt daran, dass ein uniformes Gegenbeispiel aus genau einer Eingangsbelegung besteht. Im Gegensatz dazu werden bei nicht-uniformen Präfixen die Eingänge in Zeitschritt i abhängig von den Eingängen und den Ausgängen der Blackboxes in den früheren Zeitschritten gewählt, so dass im Prinzip für jede mögliche Wertekombination der Blackboxausgänge in früheren Zeitschritten

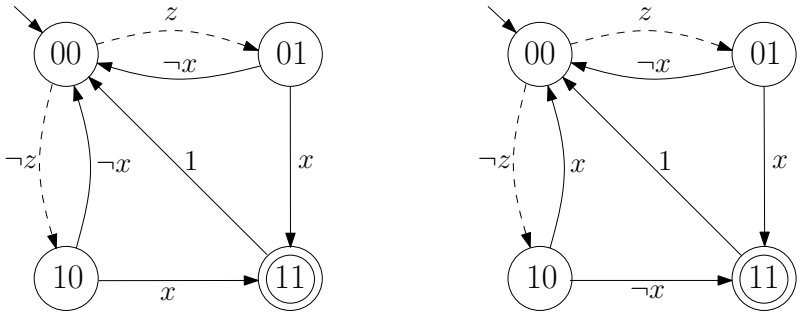
[1] Zu beachten ist hier, dass die Ausgänge der Flipflops im Zeitschritt 0 als primäre Eingänge aufgefasst werden, falls für die Flipflops kein fester initialer Wert vorgegeben ist. Sonst und in allen anderen Zeitschritten werden sie wie Tseitin-Variablen behandelt und in die Menge H aufgenommen.

eine eigene Eingangsbelegung erzeugt wird. Ein Gegenbeispiel, das mit einem uniformen Präfix erzeugt wurde, erleichtert also die Arbeit des Entwicklers beim Reproduzieren des Fehlers.

4.1.3 Beispiel (nicht-uniform versus uniform)

Wir wollen anhand eines Beispiels demonstrieren, wie die Quantorenreihenfolge einen Einfluss darauf haben kann, ob für ein vorgegebenes Modell ein Gegenbeispiel gefunden wird oder nicht.

Wir betrachten zwei Zustandsautomaten, die die Funktionsweise jeweils eines unvollständigen Schaltkreises darstellen. Dabei zeigen die gestrichelten Kanten das Verhalten



(a) Schaltung mit uniformem Gegenbeispiel

(b) Schaltung ohne uniformes Gegenbeispiel

Abbildung 4.1: Unvollständige Schaltkreise

abhängig von der Blackboxausgabe z („mögliche Übergänge“), die durchgezogenen Kanten stehen für das Verhalten, das zwar abhängig von der Eingangsbelegung x, aber unabhängig vom Wert z des Blackboxausgangs ist („sichere Übergänge“). Die Zustände codieren wir mit zwei Variablen q_0 und q_1 und erhalten für beide Zustandsautomaten die folgende Formel für den Anfangszustand (den Zustand 00):

$$I(q_0, q_1) = \neg q_0 \wedge \neg q_1$$

und für die verletzte Invariante (der Zustand 11 soll nie erreicht werden):

$$\neg P(q_0, q_1) = q_0 \wedge q_1.$$

Zukünftig werden sie durch I^0 und $\neg P^i$ für den Zeitschritt i entsprechend abgekürzt. Die Transitionsrelation für den Automaten aus Abbildung 4.1a lautet:

$$\begin{aligned} T_a(q_0^i, q_1^i, x^i, z^i, q_0^{i+1}, q_1^{i+1}) = (q_0^{i+1} \equiv \neg q_0^i \neg q_1^i \neg z^i + x(q_0^i \oplus q_1^i)) \\ \wedge (q_1^{i+1} \equiv \neg q_0^i \neg q_1^i z^i + x(q_0^i \oplus q_1^i)) \end{aligned}$$

wobei i der aktuelle Zustand ist. Wir werden die Transitionsrelation zukünftig mit $T_a^{i,j}$ abkürzen, wobei i der aktuelle Zustand und j der Folgezustand ist.

Die Transitionsrelation für den Automaten auf Abbildung 4.1b lautet:

$$\begin{aligned} T_b(q_0^i, q_1^i, x^i, z^i, q_0^{i+1}, q_1^{i+1}) = (q_0^{i+1} \equiv \neg q_0^i \neg q_1^i \neg z^i + \neg q_0^i q_1^i z^i + q_0^i \neg q_1^i \neg z^i \\ \wedge (q_1^{i+1} \equiv \neg q_0^i \neg q_1^i z^i + \neg q_0^i q_1^i z^i + q_0^i \neg q_1^i \neg z^i. \end{aligned}$$

Sie wird analog mit $T_b^{i,j}$ abgekürzt.

Für jeden von den beiden Automaten versuchen wir sowohl mit einer nicht-uniformen als auch mit einer uniformen Quantorenreihenfolge, ein Gegenbeispiel zu finden. Die bei der Tseitin-Transformation entstehenden zusätzlichen Variablen fassen wir in einer Menge H zusammen. Der Übersichtlichkeit halber verschieben wir alle Hilfsvariablen an das Ende des Quantorenpräfixes. Da deren Werte durch die Wahl des Startzustandes, der primären Eingänge und der Blackboxausgänge eindeutig festgelegt sind, verändert sich die Semantik des resultierenden BMC-Problems nicht, wenn wir diese nach „innen schieben“.

Wir betrachten zuerst die nicht-uniform quantifizierte BMC-Formel für den Zustandsautomaten aus Abbildung 4.1a:

$$\begin{aligned} BMC^{(a)}_{\text{nicht-uniform}}(0) =& \exists q_0^0 \exists q_1^0 \exists H^0 : I^0 \wedge \neg P^0 \\ BMC^{(a)}_{\text{nicht-uniform}}(1) =& \exists x^0 \forall z^0 \exists q_0^0 \exists q_1^0 \exists H^0 \exists q_0^1 \exists q_1^1 \exists H^1 : I^0 \wedge T_a^{0,1} \wedge \neg P^1 \\ BMC^{(a)}_{\text{nicht-uniform}}(2) =& \exists x^0 \forall z^0 \exists x^1 \forall z^1 \exists q_0^0 \exists q_1^0 \exists H^0 \exists q_0^1 \exists q_1^1 \exists H^1 \exists q_0^2 \exists q_1^2 \exists H^2 : \\ & I^0 \wedge T_a^{0,1} \wedge T_a^{1,2} \wedge \neg P^2 \end{aligned}$$

Die Abwicklungstiefen 0 und 1 sind unerfüllbar. Dies sieht man auch direkt am Automaten, da der „schlechte“ Zustand in mindestens 2 Schritten vom Startzustand zu erreichen ist. Wir betrachten also direkt Abwicklungstiefe 2.

Zuerst belegen wir x^0 mit einem beliebigen Wert (siehe Abbildung 4.2), da sowohl die 1 als auch die 0 zum gleichen Folgezustand führen. Die Blackboxausgabe z^0 kann entweder den Wert 0 oder 1 annehmen. Wir müssen, da z^0 all-quantifiziert ist, beide Fälle betrachten. Für x^1 wählen wir in beiden Fällen die 1. Für z^1 betrachten wir wieder beide Fälle. Nun fehlt noch die Belegung der Zustands- und Hilfsvariablen. Abhängig vom Wert der Blackboxausgaben z^0 und z^1 kann eine gültige Belegung der Zustandsvariablen gewählt werden, so dass $BMC(2)$ erfüllt ist. Im Zustandsautomat entspricht das den Pfaden $00 - 10 - 11$ bzw. $00 - 01 - 11$. Wie man sieht, ist die Folge der x-Belegungen gleich, unabhängig von den Blackboxausgaben z^0 bzw. z^1. Das bedeutet, wir können die x-Werte schon im Voraus festlegen. Dies führt uns zur uniformen Quantorenreihenfolge.

Diese stellen wir analog für den ersten Zustandsautomaten (Abbildung 4.1a) auf:

$$\begin{aligned} BMC^{(b)}_{\text{uniform}}(0) =& \exists q_0^0 \exists q_1^0 \exists H^0 : I^0 \wedge \neg P^0 \\ BMC^{(b)}_{\text{uniform}}(1) =& \exists x^0 \forall z^0 \exists q_0^0 \exists q_1^0 \exists H^0 \exists q_0^1 \exists q_1^1 \exists H^1 : I^0 \wedge T_b^{0,1} \wedge \neg P^1 \\ BMC^{(b)}_{\text{uniform}}(2) =& \exists x^0 \exists x^1 \forall z^0 \forall z^1 \exists q_0^0 \exists q_1^0 \exists H^0 \exists q_0^1 \exists q_1^1 \exists H^1 \exists q_0^2 \exists q_1^2 \exists H^2 : \\ & I^0 \wedge T_b^{0,1} \wedge T_b^{1,2} \wedge \neg P^2 \end{aligned}$$

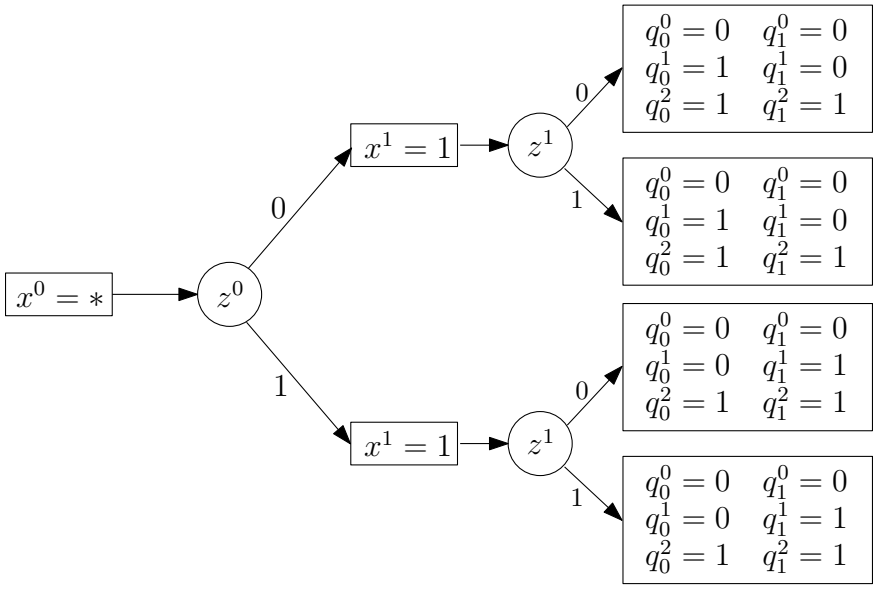


Abbildung 4.2: Nicht-uniformes Gegenbeispiel für den Automaten aus Abbildung 4.1a

Man sieht auch hier (siehe Abbildung 4.3), dass mit dieser Belegungsreihenfolge ein gültiges QBF-Problem entsteht. Wir belegen x^0 mit einem beliebigen Wert und x^1 mit dem Wert 1. Durch die Ausgaben der Blackboxes z^0 und z^1 wird dann die jeweilige Belegung der Zustandsvariablen impliziert. Im Worten lässt sich das uniforme Gegenbeispiel wie folgt ausdrücken: Für alle Blackboxersetzungen kommt man mit derselben Eingabefolge über evtl. verschiedene Pfade gleicher Länge in einen Fehlerzustand.

Für den anderen Zustandsautomaten aus Abbildung 4.1b ist es dagegen von großer Bedeutung, ob eine nicht-uniforme oder eine uniforme Quantorenreihenfolge gewählt wird. Das werden wir im Folgenden demonstrieren, indem wir die BMC-Formeln analog zum anderen Automaten aufstellen.

Im Falle der nicht-uniformen Quantifizierung kann ein Gegenbeispiel gefunden werden (siehe Abbildung 4.5). Allerdings führen, abhängig von der Blackboxausgabe, verschiedene Eingangsbelegungen in den Fehlerzustand; deswegen darf man im Quantorenpräfix x^1 nicht vorziehen. Hier kann kein uniformes Gegenbeispiel gefunden werden (siehe Abbildung 4.4).

Die Quantorenreihenfolge hat unmittelbar Einfluss darauf, ob ein Gegenbeispiel gefunden wird, und ist deswegen von sehr großer Bedeutung. Immer wenn ein uniformes Gegenbeispiel existiert, führt auch die nicht-uniforme BMC-Formel zum Ziel. Die Umkehrung gilt jedoch nicht.

4.2 Neue Kombinationsmöglichkeiten von 01X und QBF

Für manche BMC-Probleme genügt es, nur manche Blackboxausgänge mit QBF zu modellieren, ohne die Genauigkeit zu verlieren. Die restlichen Blackboxausgänge werden dann mit 01X modelliert. Ein Verfahren zur heuristischen Ermittlung der mit QBF zu

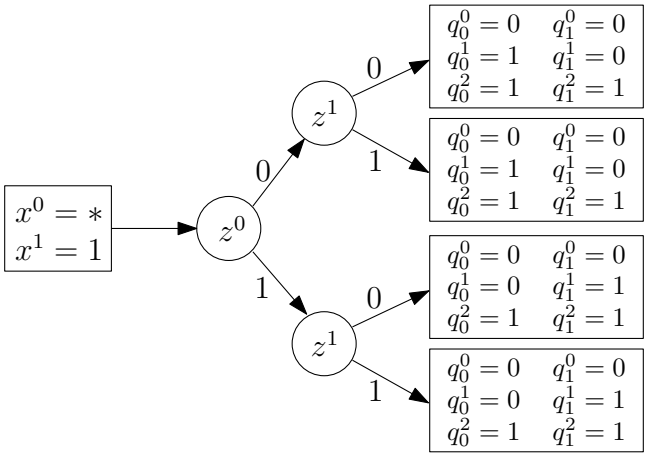


Abbildung 4.3: Uniformes Gegenbeispiel für den Automaten aus Abbildung 4.1a

modellierenden Ausgänge wird in [18] beschrieben. Die resultierende QBF enthält durch eine solche Kombination weniger universelle Quantoren, somit wird das QBF-Problem eventuell leichter zu lösen.

Eine Möglichkeit für diese Kombination wurde in [11] vorgestellt. Dort wird so vorgegangen, dass zunächst der gesamte Schaltkreis Jain-codiert wird. Soll dann ein Blackboxausgangspaar $Z = (Z_{\text{high}}, Z_{\text{low}})$ mit QBF modelliert werden, fügt man dem Quantorenpräfix $\forall Z_{\text{high}} \exists Z_{\text{low}}$ und der KNF zwei Klauseln $(Z_{\text{low}} \vee Z_{\text{high}})$ und $(\neg Z_{\text{low}} \vee \neg Z_{\text{high}})$ hinzu, die dafür sorgen, dass das binäre codierte Signal nur die Werte $0_{01\text{X}}$ und $1_{01\text{X}}$ annehmen kann.

Soll das Ausgangspaar $Z = (Z_{\text{high}}, Z_{\text{low}})$ jedoch mit 01X modelliert werden, so werden zwei Existenzquantoren verwendet: $\exists Z_{\text{high}} \exists Z_{\text{low}}$. Der KNF werden zwei Klauseln $(\neg Z_{\text{high}})$ und $(\neg Z_{\text{low}})$ mit jeweils einem Literal hinzugefügt, die dafür sorgen, dass der Blackboxausgang den Wert $X_{01\text{X}}$ annimmt.

Insgesamt haben wir also bisher drei verschiedene Arten gesehen, wie Ausgänge von Blackboxes modelliert werden können:

1. mit 01X (siehe Kapitel 3),
2. mit QBF mit Jain-codiertem Blackboxausgang, wie in [9] beschrieben worden ist, und
3. mit QBF mit einem Signal (siehe Abschnitt 4.1).

Werden nicht sämtliche Blackbox-Ausgänge mit 01X modelliert, d. h. ist die Formel nicht ein reines SAT-Problem, sondern enthält sie mindestens einen Allquantor, so kann zusätzlich zwischen den beiden oben vorgestellten Quantorenpräfixen aus 4.1.1 bzw. aus 4.1.2 ausgewählt werden.

Die drei Modellierungsvarianten können wir leicht in einem Schaltkreis kombiniert verwenden. Wie in Kapitel 3 an einem Beispiel gezeigt, verwenden wir im Schaltkreis beim

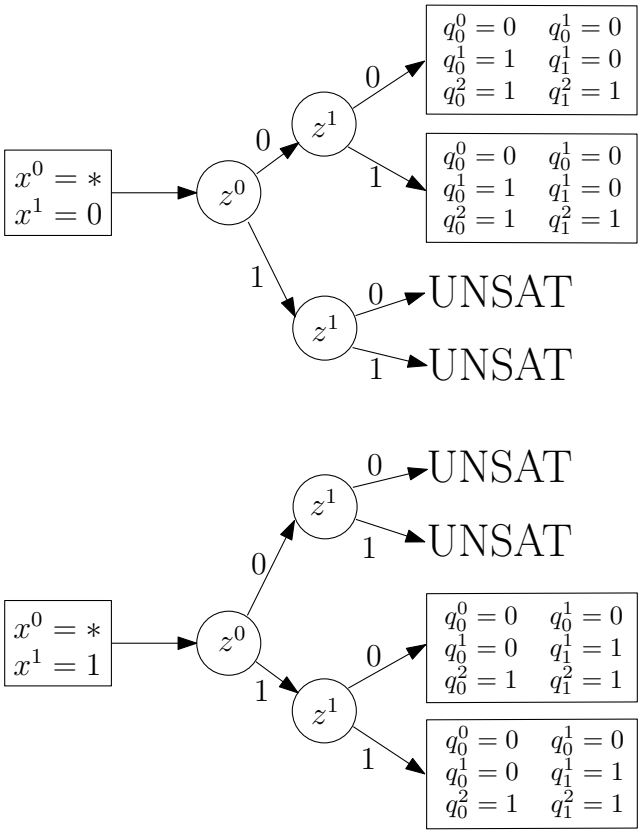


Abbildung 4.4: Gescheiterte Suche nach einem uniformen Gegenbeispiel für den Automaten aus Abbildung 4.1b

01X-Ansatz lediglich für diejenigen Leitungen die Jain-Codierung, die im Einflussbereich eines 01X-modellierten Blackbox-Ausgangs liegen, d. h. für diejenigen Leitungen, die potentiell den Wert X annehmen können. Alle anderen Leitungen werden nicht verdoppelt.

Wollen wir für einen Blackboxausgang die 01X-Codierung verwenden, verwenden wir für ihn die Jain-Codierung und fügen der Formel zwei Unit-Klauseln hinzu, die dafür sorgen, dass der Wert beider Leitungen $X_{01\mathrm{X}} = (0, 0)$ ist. Außerdem merken wir uns, dass beide Leitungen existenz-quantifiziert werden müssen; sie werden also wie diejenigen Variablen behandelt, die durch die Tseitin-Transformation eingeführt werden.

Im zweiten Fall, wenn der Blackboxausgang zwar Jain-codiert, aber trotzdem mit QBF modelliert werden soll, wird der Ausgang ebenfalls verdoppelt. Wir fügen der Formel die beiden Klauseln hinzu, die dafür sorgen, dass die Leitungen nur einen booleschen Wert aus $\{0, 1\}$ annehmen. Ferner merken wir uns, dass bei dem Variablenpaar die erste all- und die zwei existenz-quantifiziert werden muss.

Im dritten Fall, der QBF-Codierung mit nur einem Signal, wird der Blackboxausgang nicht verdoppelt. Wir merken uns die Variable, damit sie später allquantifiziert werden kann.

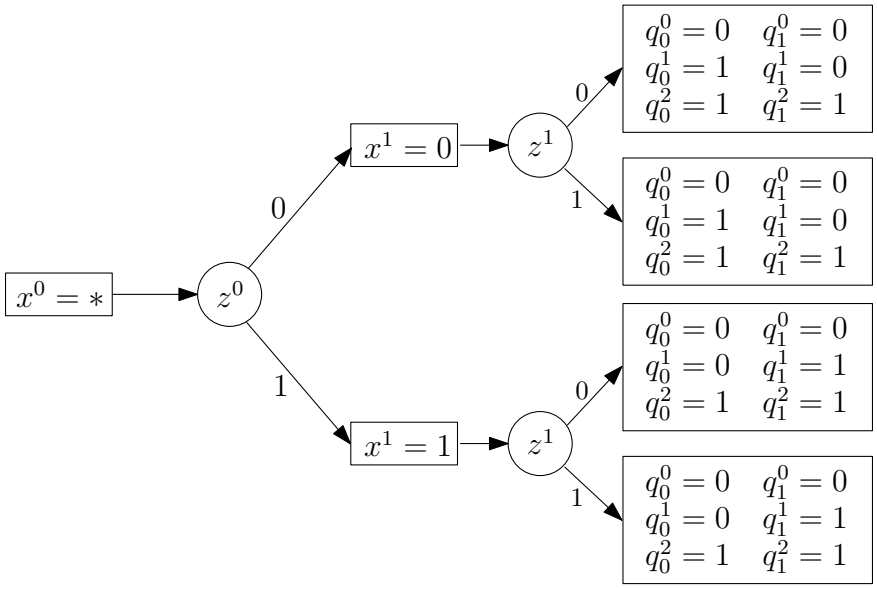


Abbildung 4.5: Nicht-uniformes Gegenbeispiel für den Automaten aus Abbildung 4.1b

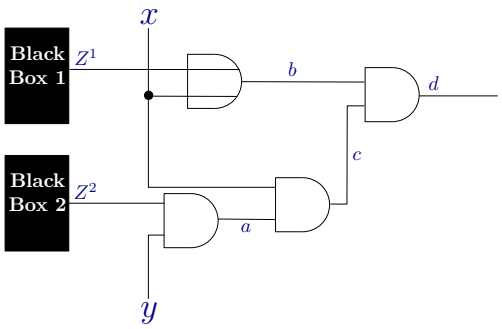
Ausgehend von den Jain-codierten Blackboxausgängen werden all diejenigen Leitungen verdoppelt, die von einem Gatter getrieben werden, von dem mindestens ein Eingang Jain-codiert ist.

Die Codierung mit QBF mit nur einem Signal hat gegenüber der mit zwei Signalen den großen Vorteil, dass zum einen dadurch, dass Teile des Schaltkreises nicht verdoppelt werden müssen, die BMC-Formel kürzer ist und weniger Variablen enthält. Zum anderen wird im Vergleich zu [11, 9] die Zahl der Quantorenwechsel reduziert, weil nicht hinter jedem All-Quantor direkt ein Existenz-Quantor stehen muss. Wir werden es anhand eines Beispiels demonstrieren.

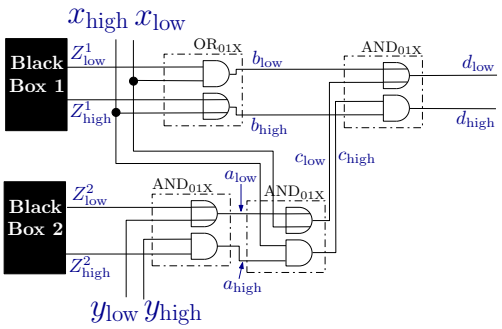
Beispiel 4.1
Abbildung 4.6a zeigt eine Schaltung mit zwei Blackboxes und zwei primären Eingängen x und y. Der Einfachheit halber enthält die Schaltung keine Flipflops, so dass eine einzige Abwicklung genügt. Um die Eigenschaft zu widerlegen, dass der Ausgang d immer den Wert 0 hat, wollen wir die obere Blackbox mit 01X modellieren, und die untere mit QBF. Um die Kombination von 01X und QBF aus [9] mit unserer optimierten Kombination zu vergleichen, geben wir beide resultierenden Schaltungen und Formeln an.

Für die QBF-Codierung mit zwei Leitungen wird zunächst der gesamte Schaltkreis Jain-codiert; alle Gatter werden durch ihre Pendants für Jain-Codierung ersetzt. Dadurch erhalten wir die Schaltung aus Abbildung 4.6b. Sei φ^b die KNF, die wir durch Tseitin-Transformation dieser Schaltung erhalten. Für die negierte Eigenschaft fügen wir zusätzlich zu φ^b die Klauseln (d_{high}) und $(\neg d_{\text{low}})$ hinzu. Ferner fassen wir die eingeführten Hilfsvariablen in H^b zusammen, d. h.

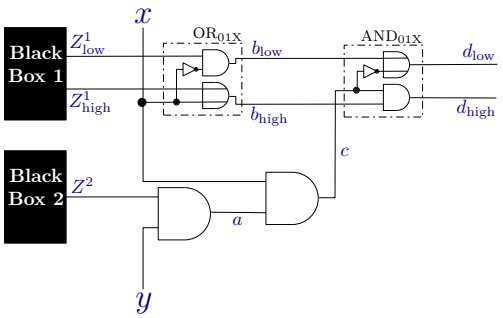
$$H^b = \{a_{\text{high}}, a_{\text{low}}, b_{\text{high}}, b_{\text{low}}, c_{\text{high}}, c_{\text{low}}, d_{\text{high}}, d_{\text{low}}\}.$$



(a) Schaltung mit mehreren Blackboxes



(b) Kombination von QBF mit zwei Leitungen und 01X



(c) Kombination von QBF mit einer Leitung und 01X

Abbildung 4.6: Kombination von 01X und QBF

Dann lautet die entsprechende Quantifizierte Boolesche Formel:

$$BMC^b = \exists x_{\text{high}} \exists x_{\text{low}} \exists y_{\text{high}} \exists y_{\text{low}} \forall Z^2_{\text{high}} \exists Z^2_{\text{low}} \exists Z^1_{\text{high}} \exists Z^1_{\text{low}} \exists H^b :$$
$$\varphi^b \wedge (\neg Z^1_{\text{high}}) \wedge (\neg Z^1_{\text{low}}) \wedge (x_{\text{high}} \vee x_{\text{low}}) \wedge (\neg x_{\text{high}} \vee \neg x_{\text{low}})$$
$$\wedge (y_{\text{high}} \vee y_{\text{low}}) \wedge (\neg y_{\text{high}} \vee \neg y_{\text{low}}) \wedge (Z^2_{\text{high}} \vee Z^2_{\text{low}}) \wedge (\neg Z^2_{\text{high}} \vee \neg Z^2_{\text{low}}).$$

Dabei mussten wir Klauseln hinzufügen, um sicherzustellen, dass $(x_{\text{high}}, x_{\text{low}})$, $(y_{\text{high}}, y_{\text{low}})$ und $(Z^2_{\text{high}}, Z^2_{\text{low}})$ nur die Werte 1_{01X} oder 0_{01X} annehmen können.

Nun erzeugen wir die BMC-Formel, indem wir die obere Blackbox wieder 01X-codieren, die untere mit QBF und nur einem Signal. Dann müssen wir nur das OR-Gatter und das oberste AND-Gatter Jain-codieren, die beiden anderen AND-Gatter bleiben unverändert. Ebenso müssen die primären Eingänge nicht Jain-codiert werden, da sie nur die Werte 1 oder 0 annehmen dürfen. Die resultierende Schaltung ist in Abbildung 4.6c dargestellt. Wir erzeugen wiederum für diesen Schaltkreis mittels Tseitin-Transformation eine KNF φ^c inklusive den Klauseln, so dass $(c_{\text{high}}, c_{\text{low}}) = 1_{01X}$ ist. Die Hilfsvariablen seien

$$H^c = \{a, b_{\text{high}}, b_{\text{low}}, c, d_{\text{high}}, d_{\text{low}}\}.$$

Dann können wir die BMC-Formel angeben:

$$BMC^c = \exists x \exists y \forall Z \exists Z^1_{\text{high}} \exists Z^1_{\text{low}} \exists H^c : \varphi^b \wedge \neg Z^1_{\text{high}} \wedge \neg Z^1_{\text{low}}.$$

Dass die Formel, die mit der optimierten Kombination erzeugt wird, kompakter ist, dürfte an diesem Beispiel offensichtlich geworden sein.

5 Experimentelle Ergebnisse

Die Ansätze, die wir im vorigen Kapitel beschrieben haben, haben wir implementiert und auf einige Schaltkreise mit Blackboxes angewendet. In diesem Kapitel werden wir unsere Experimente beschreiben und eine Auswertung vornehmen.

Implementierung

Ein bestehendes BMC-Tool für unvollständige Designs, das bisher lediglich die Modellierung der Blackboxausgänge mit 01X unterstützte, wie sie in Kapitel 3 beschrieben wurde, wurde so erweitert, dass für jeden Blackboxausgang einzeln zwischen 01X- und QBF-Modellierung ausgewählt werden kann. Falls mindestens ein Blackboxausgang mit QBF modelliert wird, kann zwischen uniformem und nicht-uniformem Quantorenpräfix ausgewählt werden. Zur Lösung von QBF-Problemen kann der Benutzer zwischen drei verschiedenen QBF-Solvern auswählen: AIGSolve [20] ist ein struktureller QBF-Solver auf der Basis von AIGs, QMiraXT [17] und QuBE [8] sind DPLL-basierte QBF-Solver. Zur Lösung reiner SAT-Formeln, wenn alle Blackboxausgänge 01X-modelliert sind, wird der SAT-Solver MiraXT [16] verwendet.

Benchmarks

Wir haben vier verschiedene Benchmarkgruppen verwendet:

- Die ISCAS-Benchmarks [14] sind industrielle Schaltkreise. Dort wurden die Blackboxes zufällig gesetzt. Wir prüfen die Eigenschaft, dass eine gewisse Hammingdistanz zum Startzustand nicht überschritten werden darf.
- Die FFB-Benchmarks, wie in [22] beschrieben, sind das Modell eines Schienennetzes mit Segmenten, auf denen sich Züge nach vorgegebenen Fahrplänen fortbewegen. Die enthaltenen Blackboxes sind einzelne Gleissegmente. Die zu prüfende Eigenschaft ist, dass Züge nie zusammenstoßen können.
- Die Benchmarks von der Hardware-Model-Checking-Competition 2008 [13] sind industrielle Schaltkreise mit zufällig gesetzten Blackboxes. Die zu prüfende Eigenschaft ist schon in die Schaltkreise eingebaut.
- Die QUALU-Benchmarks [23] sind eine VLIW-ALU mit verschiedenen Wortbreiten, wobei eine funktionale Einheit einen Fehler bei der EXOR-Berechnung hat. Wir haben den Addierer, den Multiplizierer und den Multiplexer-Baum in Blackboxes gesetzt. Die zu prüfende Eigenschaft ist, dass die EXOR-Operation korrekt funktioniert.

Ergebnisse und Auswertung

Alle Experimente wurden auf einem Rechner mit AMD Opteron 2.4 GHz-Prozessor, 1 MB Cache und 4 GB Hauptspeicher durchgeführt. Das Zeitlimit wurde auf 1800 Sekunden gesetzt.

Die experimentellen Ergebnisse haben wir in den folgenden Tabellen aufgeführt. Für jeden Benchmark haben wir beginnend mit $k = 0$ geprüft, ob die Invarianteneigenschaft unrealisierbar ist. Solange der Solver kein Gegenbeispiel finden konnte, haben wir den Wert von k erhöht.

Die erste Spalte der Tabellen enthält den Namen des jeweiligen Benchmarks, die zweite Spalte die Zahl der Klauseln in der höchsten betrachteten Abwicklungstiefe. Diese Tiefe, bei der ein Gegenbeispiel gefunden werden konnte, ist in der dritten Spalte angegeben. Für jeden der drei verwendeten QBF-Solver (AIGSolve, QuBE und QMiraXT) haben wir sowohl für die uniforme Quantorenreihenfolge (Spalte „unf“) als auch für die nicht-uniforme (Spalte „n-unf“) die Zeit in Sekunden gemessen, die der Solver benötigt hat, um ein Gegenbeispiel zu finden. Dieser Wert enthält die Zeit für alle Abwicklungstiefen, auch für diejenigen, für die der Solver UNSAT zurückgegeben hat. Enthält eine Zelle der Tabelle den Wert „TO“, so wurde das Zeitlimit von 1800 Sekunden überschritten. Entsprechend bedeutet „MEM“, dass der verfügbare Speicher von 4 GB aufgebraucht war, bevor ein Ergebnis ermittelt werden konnte.

ISCAS-Benchmarks

			AIGsolve		**QuBE**		**QMiraXT**	
Name	# Klauseln	k	n-unf	unf	n-unf	unf	n-unf	unf
s9234-003	38031	14	38.21	101.85	115.52	TO	TO	TO
s9234-010	57449	18	57.96	105.56	27.82	26.82	5.40	TO
s9234-011	54653	17	52.91	123.29	11.08	9.59	5.79	TO
s13207-007	23920	10	1.80	2.55	1.66	2.04	1.10	1.14
s13207-008	34552	12	3.76	5.33	2.80	3.24	2.56	5.10
s13207-013	23920	10	1.80	2.51	1.64	2.02	1.11	1.12
s13207-014	34552	12	3.77	5.21	5.37	3.26	TO	TO
s15850-005	68388	20	47.13	TO	TO	TO	TO	TO

FFB-Benchmarks

			AIGsolve		**QuBE**		**QMiraXT**	
Name	# Klauseln	k	n-unf	unf	n-unf	unf	n-unf	unf
ffb-a1	8216	10	7.20	12.47	0.99	TO	TO	TO
ffb-a2	8453	10	23.25	MEM	1.01	TO	TO	TO
ffb-b1	40254	25	MEM	MEM	87.88	TO	TO	TO
ffb-b2	42248	25	MEM	MEM	131.65	TO	TO	TO
ffb-b3	41247	25	MEM	MEM	90.13	TO	TO	TO
Koeln	80514	6	10.01	9.63	3.53	5.59	TO	TO

Benchmarks von der Hardware-Model-Checking-Competition 2008

			AIGsolve		QuBE		QMiraXT	
Name	# Klauseln	k	n-unf	unf	n-unf	unf	n-unf	unf
139464p5.@05.1	132903	3	86.91	87.52	274.84	8.84	2.90	2.69
texastwoprocp1.@5.1	16899	14	0.81	0.82	0.81	0.98	0.42	0.52
139462p6neg.@05.1	59667	3	15.51	14.96	7.83	7.74	1.34	1.51
139453p24.@05.2	77727	4	12.13	11.29	3.58	3.73	1.59	1.63
139462p6.@05.1	59667	3	15.42	14.89	5.54	10.90	1.36	1.50
139463p24.@05.2	134868	4	83.27	81.80	9.41	6.71	78.94	306.00
139444p0neg.@05.1	46350	3	6.27	10.17	1.16	1.24	0.76	0.77
139464p6.@05.2	133170	3	83.96	86.45	80.12	5.58	2.61	2.60
139444p24.@05.0	70356	4	9.56	9.17	2.65	2.62	1.38	1.35
139462p22.@05.2	88560	4	14.78	14.57	5.55	5.39	2.06	2.08
pdtviscoherence1.@5.2	40494	11	13.93	MEM	8.82	TO	TO	TO
139444p23.@05.0	70212	4	13.69	13.41	2.65	2.59	1.41	1.37
texasparsesysp3.@5.0	10206	8	1.10	1.03	2.70	0.99	TO	TO
texastwoprocp5.@5.1	17367	14	0.64	0.65	0.67	0.72	0.41	0.42
139464p5neg.@05.1	132903	3	87.24	86.35	269.19	8.12	2.97	2.87
139464p23.@05.0	193032	4	189.27	185.86	84.59	160.94	5.12	4.77
139464p23.@05.2	192816	4	185.67	187.44	95.20	415.36	4.92	4.73
139464p6neg.@05.2	133170	3	85.31	86.84	60.12	5.65	2.63	2.52
139464p5neg.@05.0	132963	3	106.17	104.61	374.02	99.34	TO	TO
139463p22.@05.1	134271	4	80.48	79.68	11.24	8.35	3.03	3.06
139464p22.@05.1	192939	4	187.42	193.80	57.62	78.98	4.80	4.62
abp4pold.@5.2	35115	17	7.61	7.55	199.75	21.51	TO	TO
139464p5.@05.0	132963	3	105.42	105.19	1213.89	22.64	TO	TO
139454p1.@05.2	77688	3	23.74	23.41	2.39	2.45	1.44	1.40
viseisenberg.@5.2	35718	20	11.58	11.54	11.76	49.21	TO	4.78
139453p24.@05.1	77568	4	17.33	17.89	3.34	4.51	1.61	1.60
139444p1.@05.0	46911	3	6.48	10.88	1.19	1.24	0.80	0.80
139444p23.@05.1	70242	4	13.73	13.46	8.72	2.58	1.43	1.39

QUALU-Benchmarks

			AIGsolve		QuBE		QMiraXT	
Name	# Klauseln	k	n-unf	unf	n-unf	unf	n-unf	unf
qualu2	2700	4	0.10	0.09	0.09	0.11	0.04	0.80
qualu4	5060	4	0.18	0.20	0.21	0.23	0.14	0.80
qualu16	19220	4	1.29	1.36	876.86	1.49	TO	0.39
qualu24	28660	4	2.55	2.41	TO	2.95	TO	0.70
qualu32	38100	4	4.01	4.04	TO	4.90	TO	0.98
qualu40	47540	4	7.86	7.99	TO	7.50	TO	1.35
qualu48	56980	4	7.78	9.75	TO	10.65	TO	1.74
qualu64	75860	4	14.47	14.57	TO	17.84	TO	1.35

Betrachten wir die Werte in den Tabellen genauer, fällt auf, dass die Laufzeit von AIGSolve nicht vom Quantorenpräfix abhängt. Die Laufzeiten für das uniforme und das nicht-uniforme Quantorenpräfix unterscheiden sich oft nur um wenige Sekunden. Die Laufzeiten der beiden anderen Solver unterscheiden sich zum Teil beträchtlich in Abhängigkeit vom gewählten Quantorenpräfix. Dies kann damit zusamenhängen, dass

AIGSolve ein struktureller QBF-Solver ist, der die Quantoren eliminiert und hinterher die Formel immer wieder vereinfacht, während QMiraXT und QuBE Erweiterungen des DPLL-Algorithmus sind.

Bei nicht gewürfelten Blackboxes (qualu, ffb) gibt es große Unterschiede der Laufzeiten von QuBE und QMiraXT je nach Wahl des Quantorenpräfixes: QuBE löst die ffb-Benchmarks mit nicht-uniformem Präfix in knapp über zwei Minuten, während mit uniformem Präfix 30 Minuten nicht ausreichen. QMiraXT löst hingegen keinen dieser Benchmarks innerhalb der vorgegebenen Zeit.

Bei den qualu-Benchmarks ist es umgekehrt: sowohl QuBE als auch QMiraXT lösen alle qualu-Instanzen in wenigen Sekunden, wenn das uniforme Präfix verwendet wird, mit dem nicht-uniformen Präfix werden nur die drei (bzw. zwei) kleinsten Instanzen innerhalb der vorgegebenen Zeit gelöst.

Bei den Benchmarks mit zufällig gewählter Blackbox gibt es bei QuBE keine eindeutige Tendenz: beispielsweise benötigt QuBE bei 139464p5.@05.1 von der Hardware-Model-Checking-Competition mit uniformem Präfix 8.84 Sekunden, mit nicht-uniformem hingegen 274.84 Sekunden. Andererseits ist die Laufzeit von QuBE auf 139464p23.@05.2 mit uniformem Präfix 415.36 Sekunden, mit nicht-uniformem dagegen nur 95.20 Sekunden.

Bei QMiraXT scheint bei zufällig gewählten Blackboxes das nicht-uniforme Präfix effizienter zu sein. Insgesamt erkennt man bei QMiraXT die Tendenz, dass Benchmarks entweder in wenigen Sekunden oder überhaupt nicht gelöst werden.

Auffallend ist auch, dass bei allen Benchmarks sowohl bei Verwendung des uniformen als auch des nicht-uniformen Präfixes in derselben Abwicklungstiefe ein Gegenbeispiel gefunden wurde.

Bei den Laufzeiten lässt sich folglich keine generelle Tendenz zugunsten einer der beiden Quantorenreihenfolgen beobachten.

6 Zusammenfassung und Ausblick

In dieser Arbeit haben wir BMC für unvollständige Designs untersucht. Wir haben vorgestellt, wie Blackboxausgänge mit Hilfe dreiwertiger Logik als Erfüllbarkeitsproblem modelliert werden können und deren Genauigkeitsprobleme aufgezeigt. Eine Abhilfe dafür ist die Modellierung mit Quantifizierten Booleschen Formeln. Dafür haben wir zwei verschiedene Varianten beschrieben – einmal mit einem uniformen Quantorenpräfix und einmal mit einem nicht-uniformen. Das nicht-uniforme Quantorenpräfix bietet theoretisch eine höhere Genauigkeit, allerdings enthält es mehr Quantorenwechsel als das uniforme Präfix, was typischerweise die Laufzeit der QBF-Solver erhöht. Uniforme Gegenbeispiele haben den Vorteil, dass sie leichter zu simulieren sind, wenn der Fehler im Schaltkreis reproduziert werden soll. Weiterhin haben wir gezeigt, wie beide Techniken – die Modellierung mittels dreiwertiger Logik und die Modellierung mittels QBF – in einem Schaltkreis kombiniert werden können.

Wir haben die QBF-Modellierung in ein bestehendes Tool integriert, das bisher nur die 01X-Modellierung von Blackboxausgängen unterstützte und eine Reihe von Experimenten durchgeführt.

Die experimentellen Ergebnisse stimmen für diese Benchmarks nicht mit der Erwartung überein, dass die Laufzeit beim uniformen Quantorenpräfix geringer ist, da die Zahl der Quantorenwechsel in der Formel im Vergleich zum nicht-uniformen Präfix kleiner ist. Auch die verbesserte Genauigkeit des nicht-uniformen Präfixes gegenüber dem uniformen konnte anhand dieser Benchmarks nicht beobachtet werden. Für beide Präfixe wurde stets in derselben Abwicklungstiefe ein Gegenbeispiel gefunden.

Ausblick

In diesem Abschnitt wollen wir kurz zwei Techniken beschreiben, die die Genauigkeit bzw. die Effizienz des Verfahrens verbessern können. Beide sind bereits in das BMC-Tool integriert worden, die experimentelle Evaluation steht allerdings noch aus.

Dynamisches Quantorenpräfix

In Kapitel 4.1 haben wir zwei verschiedene Quantorenpräfixe eingeführt. Bei dem einen sind die Variablen gemäß ihrer Abwicklungstiefe geordnet, beim anderen nicht. Wichtig ist bei beiden, dass die Eingangssignale vor den Blackboxausgängen quantifiziert wurden, da die Werte der Blackboxausgänge von den Eingangswerten abhängen können und deshalb nicht unabhängig von ihnen gewählt werden können.

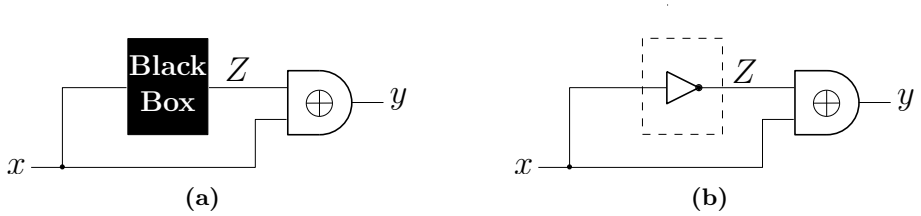


Abbildung 6.1: Kombinatorischer Schaltkreis mit einer Blackbox und eine mögliche Implementierung davon

Beispiel 6.1
Betrachten wir den kombinatorischen Schaltkreis mit einer Blackbox, der in Abbildung 6.1a dargestellt ist, zusammen mit der Eigenschaft AGy, d. h. wir verlangen, dass der Ausgang des Schaltkreises den konstanten Wert 1 trägt.

Da der Schaltkreis keine speichernden Elemente enthält und somit nur $k = 0$ betrachtet werden muss, stimmen die uniforme und die nicht uniforme Reihenfolge überein. Wir erhalten in beiden Fällen die Formel

$$\exists x \forall Z \exists y : \big((y \leftrightarrow x \oplus Z) \wedge \neg y\big).$$

Sie ist unerfüllbar, denn für $x \neq Z$ ist das Ausgangssignal 1. Tatsächlich existiert eine Implementierung der Blackbox mit der gewünschten Eigenschaft. Sie ist in Abbildung 6.1b dargestellt. Mit $Z = \neg x$ folgt $y = x \oplus Z = x \oplus \neg x = 1$.

Hätten wir jedoch die schwächere Quantorenreihenfolge

$$\forall Z \exists x \exists y : \big((y \leftrightarrow x \oplus Z) \wedge \neg y\big)$$

gewählt, bei der die Eingangsbelegung in Abhängigkeit vom Wert des Blackboxausgangs gewählt werden darf, hätten wir vom Solver die Erfüllbarkeit gemeldet bekommen, d. h. die Nichtrealisierbarkeit der Eigenschaft.

Das Beispiel zeigt, dass wir diejenigen Eingangsvariablen, die einen Einfluss auf den Wert eines Blackboxausgangs haben können, *vor* dem entsprechenden Blackboxausgang quantifizieren müssen. Eingangsvariablen, die jedoch keinen Einfluss auf den Wert des Blackboxausgangs haben können, dürfen wir jedoch nach dem Blackboxausgang quantifizieren. Das hat den Vorteil, dass wir mehr Gegenbeispiele erhalten können, wie wir nachher an einem Beispiel sehen werden.

Die Mengen I^j der Eingangsvariablen von Zeitschritt j lassen sich in folgende zwei Teile aufspalten:

- Die Menge I^j_{indep} enthält alle Eingangsvariablen aus Zeitschritt j, die keinen Einfluss auf den Wert der Blackbox-Ausgänge haben können.
- Die Menge $I^j_{\text{dep}} = I^j \setminus I^j_{\text{indep}}$ enthält die Eingänge, von deren Wert die Blackboxausgänge abhängen können.

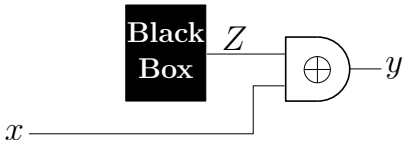


Abbildung 6.2: Schaltkreis mit einer Blackbox, deren Ausgang nicht vom primären Eingang abhängt

Wir setzen auch hier

$$I_{\text{dep}} = \bigcup_{j=0}^{k} I_{\text{dep}}^{j} \qquad \text{und} \qquad I_{\text{indep}} = \bigcup_{j=0}^{k} I_{\text{indep}}^{j}.$$

Die Mengen I_{dep}^{j} und I_{indep}^{j} lassen sich bestimmen, indem der Schaltkreis rückwärts ausgehend von den Blackboxeingängen bis zu den primären Eingängen durchlaufen wird. Jedes dabei erreichte Signal wird markiert. Die primären Eingänge, die am Ende markiert sind, bilden die Mengen I_{dep}^{j}, die nicht markierten die I_{indep}^{j}.

Mit Hilfe der oben beschriebenen Partitionierung der Variablen in $BMC(k)$ können wir zwei weitere Quantorenpräfixe definieren:

- uniform+dynamisch:
$$\exists I_{\text{dep}} \forall B \exists I_{\text{indep}} \exists H : BMC(k)$$
- nicht-uniform+dynamisch:
$$\exists I_{\text{dep}}^{0} \forall B^{0} \exists I_{\text{indep}}^{0} \exists H^{0} \exists I_{\text{dep}}^{1} \forall B^{1} \exists I_{\text{indep}}^{1} \exists H^{1} \ldots \exists I_{\text{dep}}^{k} \forall B^{k} \exists I_{\text{indep}}^{k} \exists H^{k} : BMC(k).$$

Das erste entspricht dem uniformen Quantorenpräfix, das zweite dem nicht-uniformen. Allerdings werden diejenigen Eingänge, die die Blackboxausgänge nicht beeinflussen, nach den Blackboxausgängen quantifiziert.

Beispiel 6.2
Abbildung 6.2 zeigt einen Schaltkreis mit einer Blackbox, deren Ausgang nicht vom primären Eingang x abhängt. Als Eigenschaft verwenden wir wie vorhin AGy, d. h. das Ausgangssignal soll den konstanten Wert 1 haben. Sie ist nicht realisierbar, denn dafür müsste immer $Z = \neg x$ gelten. Da aber die Blackbox den Wert von x nicht kennt, kann sie nicht immer den korrekten Wert liefern.

Sowohl die BMC-Formel mit uniformem als auch mit nicht-uniformem Quantorenpräfix sind unerfüllbar. Jedoch dürfen wir hier, da Z nicht von x abhängen kann, die umgekehrte Quantorenreihenfolge verwenden

$$\forall Z \exists x \exists y : \big((y \leftrightarrow x \oplus Z) \wedge \neg y\big),$$

die zu einer erfüllbaren Formel führt. Dazu setzen wir $x = Z$ und $y = 0$.

Mehrere Properties

Die beschriebenen Methoden gehen alle davon aus, dass die verschiedenen Abwicklungstiefen nacheinander abgearbeitet werden. Man beginnt also mit $k = 0$ und erhöht k solange um 1, bis man ein Gegenbeispiel findet. Fängt man mit einer größeren Abwicklungstiefe an oder lässt manche Abwicklungstiefen aus, kann es sein, dass Gegenbeispiele nicht gefunden werden, weil nur Gegenbeispiele gefunden werden, deren Länge exakt mit k übereinstimmt.

Wenn man will, dass bei einem Aufruf für Abwicklungstiefe k alle Gegenbeispiele der Länge $\leq k$ gefunden werden, so wird die BMC-Formel folgendermaßen abgeändert.

$$BMC(\leq k) = I(s^0) \wedge \bigwedge_{i=0}^{k-1} T(s^i, x^i, s^{i+1}) \wedge \left(\bigvee_{i=0}^{k} \neg P(s^i) \right).$$

Die Quantorenpräfixe bleiben dabei unverändert.

Der Nachteil ist, dass nicht mehr das kürzeste Gegenbeispiel gefunden wird, sondern ein beliebiges der Länge $\leq k$.

Wir haben die Hoffnung, dass durch die Verwendung dynamischer Präfixe mehr Gegenbeispiele gefunden werden. Indem wir für jeden Zeitschritt die Klauseln für die Eigenschaft einfügen, können wir uns Abwicklungstiefen einsparen. Dadurch sinkt hoffentlich der Zeitaufwand, auch wenn wir nicht mehr die kürzest möglichen Gegenbeispiele erhalten. Ob sich diese beiden Techniken in der Praxis bewähren, ist noch zu untersuchen.

Literaturverzeichnis

[1] Biere, Armin, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita und Yunshan Zhu: *Symbolic Model Checking Using SAT Procedures instead of BDDs.* In: *Proceedings of the Design Automation Conference (DAC)*, Seiten 317–320, 1999.

[2] Biere, Armin, Alessandro Cimatti, Edmund M. Clarke und Yunshan Zhu: *Symbolic Model Checking without BDDs.* In: Cleaveland, Rance (Herausgeber): *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Band 1579 der Reihe *Lecture Notes in Computer Science*, Seiten 193–.207, Amsterdam, The Netherlands, März 1999. Springer-Verlag.

[3] Bryant, Randal E.: *Graph-Based Algorithms for Boolean Function Manipulation.* IEEE Transactions on Computers, 35(8):677–691, 1986.

[4] Bryant, Randal E.: *On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication.* IEEE Transactions on Computers, 40(2):205–213, 1991.

[5] Cook, Stephen A.: *The Complexity of Theorem-Proving Procedures.* In: *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, Seiten 151–158, 1971.

[6] Emerson, E. Allen und Joseph Y. Halpern: *"Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic.* Journal of the ACM, 33(1):151–178, 1986.

[7] Garey, Michael R. und David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[8] Giunchiglia, Enrico, Massimo Narizzano und Armando Tacchella: *Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas.* Journal of Artificial Intelligence Research (JAIR), 26:371–416, 2006.

[9] Herbstritt, Marc: *Satisfiability & Verification: From Core Algorithms to Novel Application Domains.* Südwestdeutscher Verlag für Hochschulschriften (SVH), 2009.

[10] Herbstritt, Marc und Bernd Becker: *On SAT-based Bounded Invariant Checking of Blackbox Designs.* In: *Proceedings of the Sixth International Workshop on Microprocessor Test and Verification (MTV)*, Austin, TX, USA, 2005. IEEE Computer Society.

[11] Herbstritt, Marc und Bernd Becker: *On Combining 01X-Logic and QBF*. In: Moreno-Díaz, Roberto und Franz Pichler (Herausgeber): *Proceedings of the 11th International Conference on Computer Aided Systems Theory (EUROCAST)*, Band 4739 der Reihe *Lecture Notes in Computer Science*, Seiten 531–538, Las Palmas de Gran Canaria, Canary Islands, Spain, February 2007. Springer-Verlag. Revised Selected Papers.

[12] Herbstritt, Marc, Bernd Becker und Christoph Scholl: *Advanced SAT-Techniques for Bounded Model Checking of Blackbox Designs*. In: Abadir, Magdy S., Li-C. Wang und Jayanta Bhadra (Herausgeber): *Proceedings of the Seventh International Workshop on Microprocessor Test and Verification (MTV 2006), Common Challenges and Solutions, 4-5 December 2006, Austin, Texas, USA*, Seiten 37–44. IEEE Computer Society, 2006.

[13] *Hardware Model Checking Competition*, 2008. `http://fmv.jku.at/hwmcc08/`.

[14] *ISCAS'89 benchmark suit.* `http://www.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html`.

[15] Jain, Ankur, Vamsi Boppana, Rajarshi Mukherjee, Jawahar Jain, Masahiro Fujita und Michael S. Hsiao: *Testing, Verification, and Diagnosis in the Presence of Unknowns*. In: *Proceedings of the 18th IEEE VLSI Test Symposium (VTS 2000)*, Seiten 263–270, Montreal, Canada, 2000. IEEE Computer Society.

[16] Lewis, Matthew, Tobias Schubert und Bernd Becker: *Multithreaded SAT Solving*. In: *Proceedings of the ASP Design Automation Conf.*, Seiten 926–921, Yokohama, Japan, January 2007.

[17] Lewis, Matthew, Tobias Schubert und Bernd Becker: *QMiraXT – A Multithreaded QBF Solver*. In: *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, 2009.

[18] Miller, Christian, Stefan Kupferschmid und Bernd Becker: *Exploiting Craig Interpolants in Bounded Model Checking for Incomplete Designs*. In: *GI/ITG/GMM Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"*, Dresden, February 2010. to be published.

[19] Molitor, Paul und Janett Mohnke: *Equivalence Checking of Digital Circuits: Fundamentals, Principles, Methods*. Springer-Verlag, 2004.

[20] Pigorsch, Florian und Christoph Scholl: *Exploiting Structure in an AIG Based QBF Solver*. In: *Proceedings of the Conf. on Design, Automation and Test in Europe*, Seiten 1596–1601, April 2009.

[21] Pnueli, Amir: *The Temporal Logic of Programs*. In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, Seiten 46–57, Providence, Rhode Island, USA, 1977. IEEE.

[22] S1, AVACS: *The FFB Benchmarks*, 2007. `http://www.avacs.org/`.

[23] S1, AVACS: *The VLIW ALU Benchmark*, 2007. `http://www.avacs.org/`.

[24] Tseitin, Grigori S.: *On the Complexity of Derivation in Propositional Calculus*. Studies in Constructive Mathematics and Mathematical Logic, Part 2:115–125, 1970.