Kombinatorischer Äquivalenzvergleich

Bisher...

- BDDs
  - eindeutige Darstellung
  - symbolische Simulation
- SAT
  - Erfüllbarkeit des Miter-Schaltkreises

⇨ Wie vergleicht man damit *große* Schaltungen?

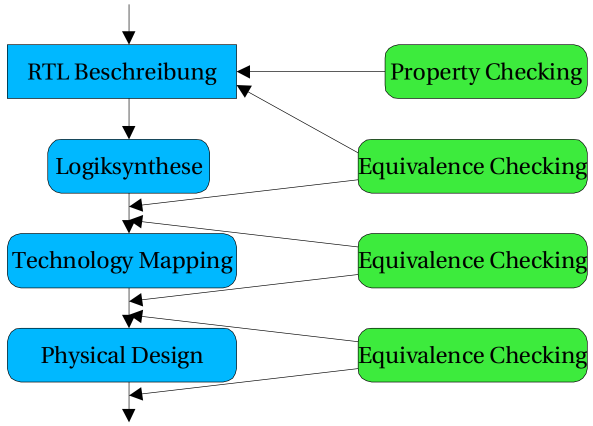
In diesem Kapitel...

- Anwendung von CEC
- Struktureller Ansatz
  - Identifizierung von Gattern
- Term Rewriting
- Probleme
- Vergleich BDDs – SAT bei CEC

Äquivalenzvergleich

- Gegeben:
  zwei (sequentielle) Schaltungen
- Gesucht:
  sind die beiden Schaltungen (funktional) äquivalent?
- Nicht betrachtet:
  - Timing
  - Power
  - Fläche
  - etc.

Equivalence Checking



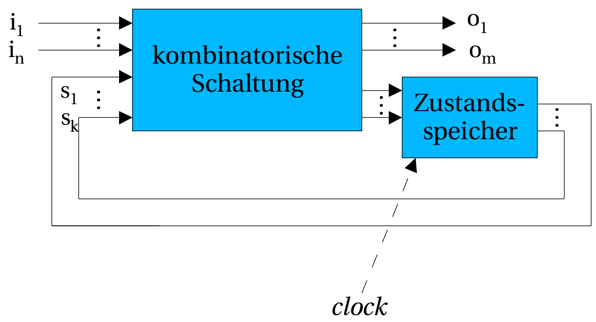
Equivalence Checking (2)

- kombinatorischer Äquivalenzvergleich
  - kombinatorische Schaltungen, oder
  - sequentielle Schaltungen mit gleicher Zustandskodierung
- sequentiellen Äquivalenzvergleich
  - sequentielle Schaltungen

Mealy-Maschine

- Synchrone sequentielle Schaltungen werden meist durch Mealy-Maschinen modelliert
- Definition: Eine Mealy-Maschine ist ein endlicher Automat $M = (I, O, S, S_{init}, \delta, \lambda)$
  - I: endliche Menge von Eingabesymbolen
  - O: endliche Menge von Ausgabesymbolen
  - S: endliche Menge von Zuständen
  - $S_{init} \subseteq S$: Menge von erlaubten Anfangszuständen
  - $\delta : S \times I \rightarrow S$: die Zustandsübergangsfunktion
  - $\lambda : S \times I \rightarrow O$: die Ausgabefunktion

Blockdiagramm



Äquivalenzbegriffe

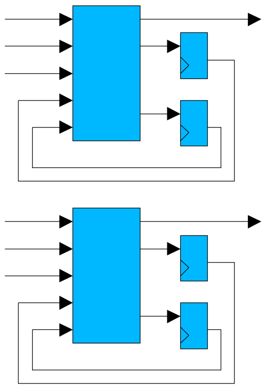
- Seien M und M' zwei Automaten mit gleichen Mengen von Ein- und Ausgabesymbolen
- Automatenäquivalenz (Verhaltensäquivalenz):
  - Für beliebige Eingabefolgen wird die gleiche Ausgabefolge geliefert, beginnend bei Startzustand
- Zustandsäquivalenz
  - Äquivalenzrelation für Zustände:
    $s \sim s' \;:\Leftrightarrow\; \forall\, i \in I : \lambda(s, i) = \lambda'(s', i)$ und $\delta(s, i) \sim \delta'(s', i)$.
  - M, M' sind äquivalent, wenn $s_{init} \sim s_{init}'$.

Zustandskodierung

- Voraussetzung für ***CEC***:
  Die Zustandskodierung in beiden Schaltungen muss gleich sein
- In späten Phasen des Entwurfs wird die Kodierung nicht verändert
- Beim Vergleich RTL-Netzliste kann das Synthesetool in manchen Fällen die Kodierung ändern (Optimierungsmöglichkeit)
  - Oft kennt der Designer eine gute Kodierung
  - (bisher) keine große Einschränkung in der Praxis

State Matching

- Welche Inputs, Outputs, FlipFlops korrespondieren zueinander?



Matching: Ansätze

- Matching basierend auf den Namen
  - Namen müssen übereinstimmen / ähnlich sein
- Matching anhand der Funktion
  - Übereinstimmen der Funktion
  - „Ähnlichkeit" der Funktion (Support, etc.)

Matching: Schwierigkeiten

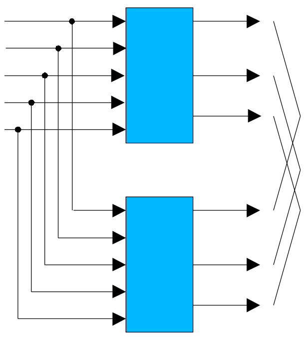
- Matching basierend auf Namen
  - Funktioniert gut bei Inputs und Outputs
    - Schwierigkeiten bei Black Boxing
  - Namen von States können sehr unterschiedlich sein
- Matching basierend auf der Funktion
  - Je genauer die Funktion betrachtet wird, desto
    - aufwändiger ist das Verfahren
    - anfälliger ist das Verfahren bei Nichtäquivalenz
- Eine gute Heuristik basiert auf der Kombination verschiedener Ansätze

Matching: Äquivalenzklassen

- Ein Algorithmus für das Matching von Zustandsbits
  - Anfangs sind alle in der selben Äquivalenzklasse
  - Die Äquivalenzklassen werden anhand der vorherigen Kriterien immer weiter verfeinert
  - Dies wird solange wiederholt, bis ein Matching gefunden wurde

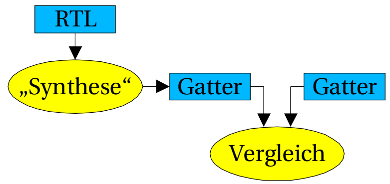
nach dem Matching...

- korrespondierende Inputs, Outputs, FlipFlops sind gefunden
- FlipFlops sind aufgebrochen
  - Zusätzliche Ein- und Ausgänge
- Korrespondierende Eingänge sind identifiziert



Modellbildung

- Basis ist (zunächst) die Gatterebene
- Wenn eine (beide) Schaltung in RTL gegeben ist, muss sie zuerst in eine Netzliste übersetzt werden
  - Ähnlich zur Aufgabe eines Synthese-Tools
  - andere Optimierungsziele

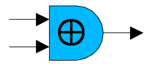


Erste Idee

- Für korrespondierende Ausgänge
  - Konstruktion der beiden BDDs
  - Vergleich der BDDs
- Vorteil: Vergleichen der BDDs kann in konstanter Zeit erfolgen
- Nachteil: Konstruktion der BDDs kann zu viel Speicher erfordern

Zweite Idee

- Für korrespondierende Ausgänge
  - Konstruktion eines Miter-Schaltkreises
  - Testen von Erfüllbarkeit mittels SAT



- Vorteil: Größe der Problembeschreibung wächst **linear** in der Größe der Schaltung
- Nachteil: Lösung des SAT-Problems kann dauern

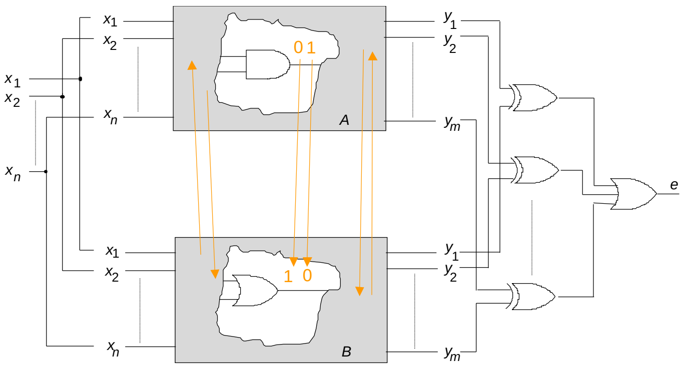
Nächste Idee

- Kann man BDDs und SAT kombinieren, so dass die Vorteile beider Ansätze bleiben?
- Äquivalenzvergleich ist co-NP-vollständig.

Beobachtung

- Die beiden Schaltungen sind oft strukturell ähnlich
  - Nur geringe Änderungen in späten Designphasen
    - Z.B. Einfügen von Clock-Trees
  - Auch beim Vergleich RTL – Netzliste gibt es viele äquivalente Punkte
- Kann man dies irgendwie ausnutzen?

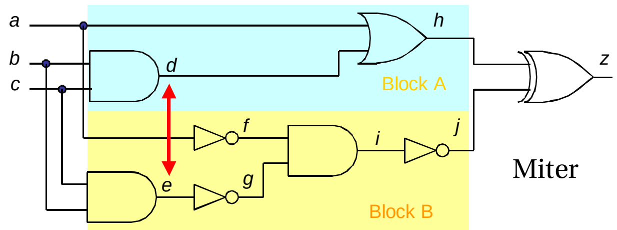
Äquivalente Punkte



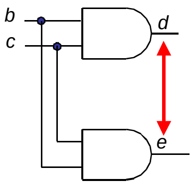
Struktureller Ansatz

- Von den Eingängen zu den Ausgängen
  - Identifiziere Äquivalenzen an internen Signalen im Miter
  - Substituiere Knoten entsprechend den Äquivalenzen
- Beweise/Widerlege Erfüllbarkeit des Ausgangs des modifizierten Miters

Beispiel



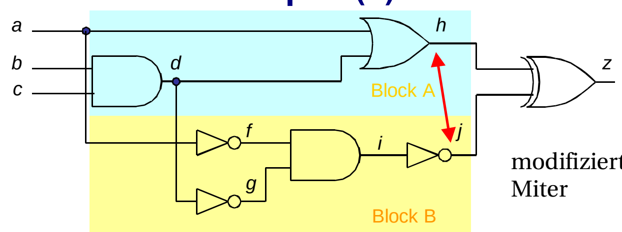
Miter



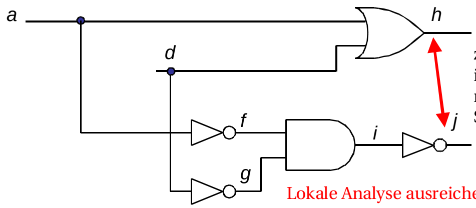
Zur Bestimmung der internen Äquivalenz $d = e$ relevante Schaltungspartition

Lokale Analyse ausreichend !

Beispiel (2)



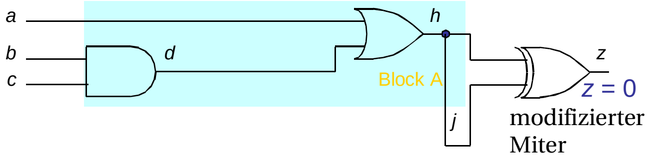
modifizierter Miter



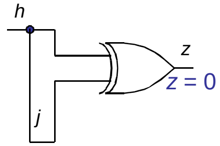
zur Bestimmung der internen Äquivalenz $h = j$ relevante Schaltungspartition

Lokale Analyse ausreichend !

Beispiel (3)



modifizierter Miter



relevante Schaltungspartition zur Bestimmung von $z = 0$

Lokale Analyse ausreichend !

Probleme

1) Gibt es ausreichend interne äquivalente Knoten?
2) Wie findet man interne äquivalente Knoten?
3) Wie nutzt man äquivalente Knoten aus?

Zahl äquivalenter Knoten

- Beobachtung (Kühlmann, 1997):

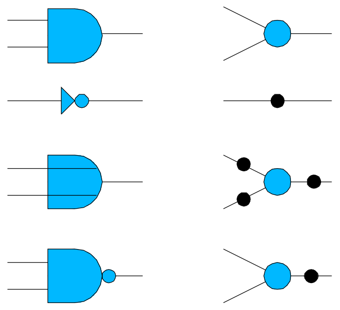
  In 80% aller Schaltkreis-Paare gibt es zu mehr als 80% der Knoten einen äquivalenten Knoten in der anderen Schaltung.

Struktureller Äquivalenzbeweis

- Die Äquivalenz zweier Ausgänge wird bewiesen, indem
  - Primäre Eingänge paarweise identifiziert werden
  - Gatter, die gleiche Eingänge haben und die gleiche Funktion realisieren, identifiziert werden.

Kühlmann, DAC 1997

- Nicht-kanonische graphenbasierte Darstellung von Schaltkreisen (And, Inverter-Netzliste)



Kühlmann, DAC 1997 (2)

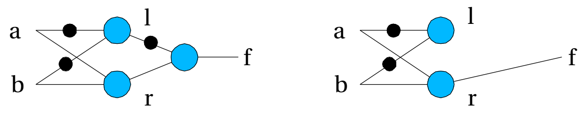
- Eine Hash-Tabelle wird verwendet, um strukturelle Redundanz zu vermeiden (ähnlich zur Unique Table bei BDDs)
- Termersetzungsregeln beim Einfügen neuer Knoten
  - Betrachten von zwei Leveln gleichzeitig
  - Ersetzen jeder 4-Input Sub-Struktur durch eine kanonische Darstellung
    - Feine Granularität
    - Überlappen der Regionen


```
Node* and(Node* p1, Node* p2) {
        // constant folding
        if (p1 == CONST_0)     return CONST_0;
        if (p2 == CONST_0)     return CONST_0;
        if (p1 == CONST_1)     return p2;
        if (p2 == CONST_1)     return p1;
        if (p1 == p2)          return p1;
        if (p1 == ~p2)         return CONST_0;
        // rank order inputs
        if (rank(p1) > rank(p2)) swap(p1, p2);
        // check for isomorphic entry in hash table
        p = hash_lookup(p1, p2);  if (p) return p;
        // 3 cases depending on position in circuit
        if (is_var(p1) && is_var(p2))  return new_and_vertex(p1, p2);
        else if (is_var(p1))           return and_3(p1, p2);
        else if (is_var(p2))           return and_3(p2, p1);
        else                           return and_4(p1, p2);
}
```



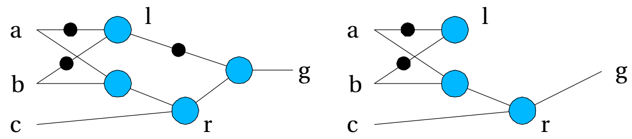
```
Node* and_4(Node* l, Node* r) {
        ll = left_child(p1);
        lr = right_child(p1);
        rl = left_child(p2);
        rr = right_child(p2);
        index = get_canonical_case(l, r);
        switch(index) {
                // first set: grandchildren are shared
                ...
                case 144: // f = (a + b) * (a b)
                        return and(rl, rr);
                ...
                // second set: grandchildren are not shared
                ...
                // look for sharing of grandchildren and
                // great-grandchildren
}
```

Beispiel 1: *f* = (*a* + *b*) * (*a b*)



- Strukturen mit identischen oder komplementierten Enkeln werden auf isomorphe Implementierungen abgebildet
  - Lokale Redundanz wird vermieden
- Wird rekursiv aufgerufen

Beispiel 2: *g* = (*a* + *b*) * (*a b c*)



- Wenn es keine identische Enkel gibt, so werden Strukturen mit identischen Enkeln und *Urenkeln* berücksichtigt
  - Wird rekursiv aufgerufen
  - Schleifen der rekursiven Aufrufe sind zu verhindern

Bewertung

- AND-Inverter-Netzlisten bieten
  - eine kompakte Netzlistendarstellung
  - sind lokal kanonisch

	BDDs	AND-Inv	SAT
Darstellungsgröße	exp.	linear	linear
Kanonizität	ja	lokal	nein
Lösung CEC	O(1)	NP	NP

Bestimmung interner Äquivalenzen

- AND-Inverter-Netzlisten
- Simuliere **Zufallsmuster**, um geeignete Kandidaten zu finden
  - Durch unvollständige Simulation kann nur Nicht-Äquivalenz gezeigt werden.
- Verwende **BDDs**, um die Äquivalenz zu beweisen
  - Kann für zwei Kandidaten Äquivalenz bewiesen werden, so können sie **identifiziert** werden
  - Nicht-Äquivalenz der Kandidaten bedeutet nicht, dass die Schaltungen inäquivalent sind

Verwendung von Äquivalenzen

- Äquivalenzen können als „interne Schnittpunkte“ („cut points“) verwendet werden
  - Der Knoten wird zu einer neuen Eingangsvariablen
  - Wenn die restliche Schaltung äquivalent ist, so war auch die Original-Schaltung äquivalent

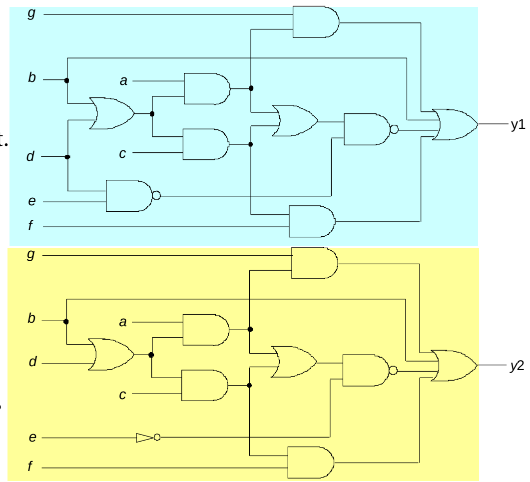
Wo liegt das Problem?

Beispiel

Die Schaltungen sind äquivalent.

(Konstruktion des globalen BDDs zeigt dies.)

Lässt sich die Äquivalenz auch mittels äquivalenten Knoten zeigen?

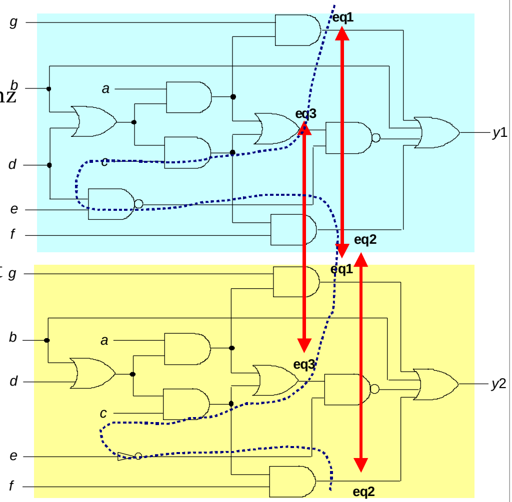


Beispiel

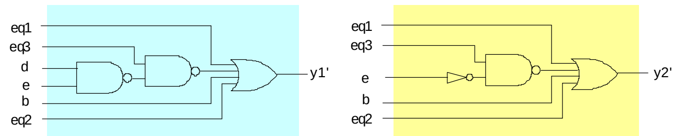
Beweise Äquivalenz von eq1, eq2, eq3 durch BDDs mit Eingangssignalen als Variablen

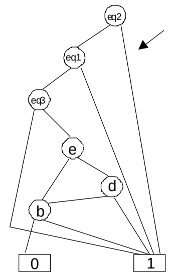
Schnitt durch Schaltung (enthält nur äquivalente Signale)

Generiere BDDs für die Ausgänge mit Variablen der Schnittlinie



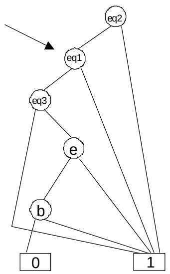
Beispiel (3)





BDDs sind klein, aber nicht äquivalent!

"False negative"!

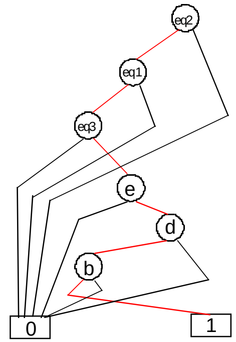


False Negatives

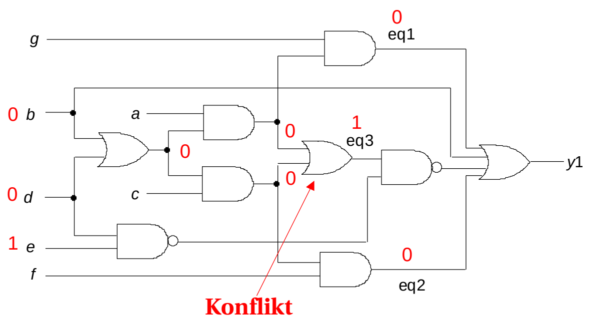
- Durch Verwendung neuer Variablen für die internen Schnittpunkte werden für diese alle Werte zugelassen
- Der hintere Teil der Schaltung braucht aber nur für diejenigen Werte äquivalent sein, die im vorderen eingestellt werden können.

Beispiel (4)

- BDD für XOR der Ausgänge
- Jede erfüllende Belegung repräsentiert eine Wertebelegung, unter der die Schaltungen verschieden sind.
- Bei einem False Negative ist diese Belegung nicht auf der Schaltung einstellbar



Beispiel (5)



Konflikt in Originalschaltung → false negative

Erkennung von False Negatives

- Ansatz 1:
  Führe für alle erfüllenden Belegungen des BDDs einen Erfüllbarkeitstest auf der Schaltung durch
  - ✔ Erfüllbarkeitstest auf der Schaltung kann mittels eines SAT-Solvers erfolgen
  - ✗ Es kann exponentiell viele erfüllende Belegungen geben

Erkennung von False Negatives (2)

- Ansatz 2:
  Substituiere im BDD die Variablen der Schnittlinie durch ihre Funktion
  d.h. Verschieben der Schnittlinie in Richtung der Inputs, bis die Unterschiede verschwinden
  - ✔ Es sind höchstens linear viele Substitutionen notwendig
  - ✗ Die BDDs können explodieren

Erkennung von False Negatives (3)

- Ansatz 3:
  BDDs werden für sich überlappende Partitionen der Schaltung aufgebaut
  - ✔ False Negatives werden vermieden
  - ✔ Größere Robustheit
  - ✗ Viele BDDs müssen konstruiert werden
  - ✗ Der richtige Schnitt kann dennoch verpasst werden

SAT

- SAT-basierte Techniken vermeiden das Problem der „false negatives“
  - die Schaltung muss nicht partitioniert werden
  - von äquivalenten Knoten kann dennoch profitiert werden

BDDs versus SAT bei CEC

BDDs

- Komplexität liegt in der kanonischen Darstellung
- Besser auf kleineren Instanzen (< 10,000 Knoten)
- Äquivalente Knoten können leicht gefunden werden

SAT

- Komplexität liegt im Erfüllbarkeitstest
- Besser auf größeren Instanzen

Kombination verschiedener Verfahren

- Kombinieren verschiedener Verfahren ist sinnvoll
  - AND-Inverter-Netzliste zum schnellen Finden äquivalenter Teile
  - Random-Pattern-Simulation (schnelles Finden von Gegenbeispielen)
  - BDDs, mit hartem Knotenlimit
  - SAT mit Resourcenbeschränkung
  - ...
- Zunehmende Komplexität der Ansätze

Fazit

- Beim **kombinatorischen Äquivalenzvergleich** gibt es üblicherweise viele äquivalente Knoten
  - Nutzen der Äquivalenzen ist unbedingt notwendig
  - Strukturelle Verfahren (etwa AND-Inverter-Netzlisten) sind sinnvoll
- Eine Mischung verschiedener Verfahren ist unabdingbar
- Arithmetische Schaltungen machen Probleme
- Der kombinatorische Äquivalenzvergleich ist ausgereift