Prof. Dr. Rolf Drechsler, drechsler@informatik.uni-bremen.de, MZH 4330
Jan Kleinekathöfer, ja_kl@uni-bremen.de, MZH 4300

Formale Beweismethoden - Programmieraufgabe

Aufgabe 1 (40 %)

Implementiere den in der Vorlesung vorgestellten Algorithmus zur Synthese von ROBDDs durch den ITE-Operator. Achte dabei darauf den Algorithmus so zu implementieren, dass die Komplexität der in der Vorlesung vorgestellten Komplexität entspricht.

Mögliche technische Stolpersteine:

- **Vergleich von Knoten**: Knoten müssen zwingend über einen eindeutigen Identifikator verglichen werden (z.B. Speicheradresse). Wird ein Vergleich durch einen Vergleich der Nachfolger umgesetzt, ist die Operation linear zur Graphgröße.
- **Datenstrukturen**: Die 'unique table' und 'computed table' sind Datenstrukturen, welche eine konstante Suchkomplexität haben (bzw. zumindest näherungsweise).
- **Hashfunktion für Knoten**: Um Hashtables(Hashmaps) verwenden zu können, muss eine Hashfunktion auf Knoten definiert sein. Diese sollte von der Variable und den Identifikatoren der Nachfolger abhängig sein.

Aufgabe 2 (30 %)

Erweitere dein Paket auf KFDDs. Füge deinem Paket dazu die Möglichkeit, Variablen mit einem spezifischen Dekompositionstyp zu assoziieren, hinzu. Zusätzlich müssen die Dekompositionen selbst in die Synthese eingebaut werden. Um eine effiziente Synthese sicherzustellen, implementiert den XOR-Operator rekursiv.

Aufgabe 3 (30 %)

Verwende das von dir verwendete Paket, um Schaltkreise im Aiger Format zu simulieren. Dazu muss ein Parser für Schaltkreise im Aiger Format realisiert werden. Anschließend kann eine symbolische Simulation durchgeführt werden.

Wir stellen zum Testen zwei Typen von Schaltkreisen zur Verfügung: 1. Kleine Schaltkreise, welche eine unbekannte Funktion realisieren. 2. Addiererschaltkreise in unterschiedlichen Größen.

Für die kleinen Schaltkreise stellen wir jeweils zwei Graphen zur Verfügung. Einer der beiden ist das Referenzdesign, der zweite eine optimierte Version. Für die Verifikation gilt es zu überprüfen, ob die realisierten Funktionen der beiden Schaltkreise äquivalent sind. Für die Addierer kann der Ripple-Carry-Addierer als Referenz für alle Addierer der gleichen bit-Größe verwendet werden.

Um eine effiziente Verifikation sicherzustellen, ist die Variablenordnung entscheidend. Diese ist jedoch nur bekannt, wenn klar ist, welche Funktion ein Schaltkreis implementiert.

a) Führe die Verifikation für alle Addierer mit BDDs für die bekannte Variablenordnung aus. Messe dabei die Ausführungszeit und die maximale Anzahl an Knoten.

b) Simuliere alle anderen Schaltkreise mit BDDs für 10 unterschiedliche, zufällige Variablenordnungen. Gebe für die beiden Metriken aus a) jeweils das Minimum, den Durchschnitt sowie das Maximum an.

c) Wiederhole a), b) für KFDDs. Verwende für a) und b) jeweils zufällige DTLs.

d) Protokolliere deine Ergebnisse.

e) Stelle ein Skript/Programm zur Verfügung, mit welchem wir die Experimente wiederholen können.

Gruppenbearbeitung: Grundsätzlich sind die drei Aufgaben zur Einzelbearbeitung ausgelegt. Ihr wollt die Programmaufgabe in einer Gruppe bearbeiten? Dann müssen wir den Umfang natürlich etwas erweitern. Kommt dazu bei Bedarf auf uns zu.

Bearbeitungszeitraum: Mit der Bearbeitung könnt ihr jederzeit beginnen. Abgabe der Programmieraufgabe ist zwei Wochen vor eurem Fachgespräch, spätestens aber zwei Wochen nach Ende der Vorlesungszeit (16.02.25). Wir teilen euch dann spätestens eine Woche vor dem Fachgespräch eure Vornote mit.

Programmiersprache: Verwendet C/C++ oder Java. Falls ihr andere Programmiersprachen verwenden wollt, sprecht das bitte mit uns ab.

Abgabe: Eure Abgabe besteht aus drei Teilen.

- Einer kurzen Beschreibung eurer Implementierung: Wie ist die Implementierung strukturiert? Welche Besonderheiten gibt es? Wie lässt sich euer Paket bauen und ausführen?
- Das Paket selbst als kommentierter lesbarer Code. Wie in Aufgabe 3 beschrieben soll, auch eine Möglichkeit enthalten sein, eure Experimente zu wiederholen.
- Eine Auswertung der Experimente aus Aufgabe 3. Diese soll die Resultate in Form von Tabellen oder Graphen enthalten. Ordnet außerdem die Ergebnisse ein, skaliert beispielsweise die Verifikation von Addierer, wie es theoretisch zu erwarten wäre?