Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας

ECE431 - Αλγόριθμοι CAD I - Λογική Σύνθεση

Χειμερινό Εξάμηνο — Ακαδημαϊκό Έτος 2023-2024

Εργαστηριακή Εργασία 3^{η}

Working with BDDs Using CUDD Package

17/12/2023 έως 14/1/2024

Χ. Σωτηρίου

1 Στόχος της 3ης Εργασίας

Οι στόχοι της 3ης εργασίας είναι:

1. Η ανάλυση μιας λογικής έκφρασης και η κατασκευή του αντίστοιχου Binary Decision Diagram (BDD) για αυτή.
2. Η ταξινόμηση των gatepins του design σε Levels (Levelization).

Για την κατασκευή των BDDs σας προτείνεται να χρησιμοποιήσετε το πακέτο Colorado University Decision Diagram Package (CUDD) καθώς προσφέρει ένα έτοιμο API για την κατασκευή και διαχείριση BDDs.

2 Colorado University Decision Diagram Package (CUDD)

Το CUDD αποτελεί μια βιβλιοθήκη λογισμικού, η οποία παρέχει εξελιγμένες δυνατότητες για τη δημιουργία, τη διαχείριση και την επεξεργασία Binary Decision Diagrams (BDDs), αλλά και άλλων συναφών δομών δεδομένων όπως τα Algebraic Decision Diagrams (ADDs) και τα Zero-Suppressed BDDs (ZDDs). Αναπτύχθηκε στο Πανεπιστήμιο του Colorado, και είναι ιδιαίτερα χρήσιμο στην αυτόματη σχεδίαση ηλεκτρονικών κυκλωμάτων (EDA), στην ανάλυση και επαλήθευση συστημάτων και στην τεχνητή νοημοσύνη. Οι αλγόριθμοι του

CUDD επιτρέπουν την αποδοτική επίλυση πολύπλοκων προβλημάτων που απαιτούν εκτεταμένους υπολογισμούς λογικής και πιθανοτήτων, καθιστώντας το πολύτιμο εργαλείο για ερευνητές και μηχανικούς στον τομέα του CAD αλλά και της υπολογιστικής λογικής.

Χρησιμοποιώντας το πακέτο CUDD δε θα χρειαστεί να υλοποιήσετε τις δικές σας δομές ή συναρτήσεις διαχείρισης δέντρων όπως επίσης, γλυτώνετε και από την ανάγκη να υλοποιήσετε αλγορίθμους ή πράξεις μεταξύ των BDDs που μπορεί να χρειαστούν στα πλαίσια της εργασίας. Ένα χαρακτηριστικό παράδειγμα είναι το variable reordering για το οποίο το API προσφέρει εσωτερική συνάρτηση (`Cudd_ReduceHeap()`) που κάνει minimize το BDD ώστε να έχετε το μικρότερο δυνατό βάθος στο δέντρο.

Περισσότερες πληροφορίες για την εγκατάσταση όπως και την χρήση του πακέτου μπορείτε να βρείτε εδώ.

3 Μέρος A: Logic Expression Analysis and BDD Construction

Σε αυτό το μέρος της εργασίας σας ζητείται δεδομένης μιας λογικής έκφρασης να κατασκευάσετε το BDD της. Αυτή μπορεί είναι ο τύπος μιας πύλης του κυκλώματος που έχετε διαβάσει από το Practical Format ή μία οποιαδήποτε άλλη boolean έκφραση. Μιας και το CUDD διευκολύνει πολύ τη διαδικασία χτισίματος του δέντρου ή όλη ουσία αυτού του μέρους είναι να εξοικειωθείτε με το πακέτο και να δείτε πως γίνεται η ανάλυση μιας λογικής έκφρασης στους υπολογιστές.

Σε αυτά τα πλαίσια λοιπόν, σας ζητείται για την κατασκευή του BDD μιας boolean έκφρασης να τη μετατρέψετε από infix σε postfix λόγω της απλότητας και της αποδοτικότητας που προσφέρει αυτή η μορφή στην ανάλυση και επεξεργασία λογικών εκφράσεων. Για τη μετατροπή της συνάρτησης μπορείτε να δείτε αλγορίθμους όπως ο Shunting yard algorithm του Dijkstra ο οποίος χρησιμοποιεί μια stack και έχει πολυπλοκότητα γραμμική $O(n)$.

Infix	**Postfix**
!((A2 * A1) + B)	A2 A1 * ! B +
!((I0 * !(S)) + (I1 * S) + (I0 * I1))	I0 S ! * I1 S * + I0 I1 * + !

Table 1: Infix and Postfix demostration

Οι σχετικες TCL εντολες που θα πρέπει να υλοποιήσετε είναι οι εξής:

- `report_library_cell_BDD <library cell name>`
- `report_component_BDD <component name>`
- `compute_expression_BDD <boolean expression>`

Οι δύο πρώτες θα παίρνουν ως παράμετρο το όνομα από το library cell ή το όνομα από ένα instance του κυκλώματος αντίστοιχα ενώ η τελευταία θα παίρνει ως παράμετρο μια λογική έκφραση και όλες θα επιστρέφουν το αντίστοιχο DBB. Το πακέτο CUDD προσφέρει εσωτερική συνάρτηση που γράφει το δέντρο σε dot format (`Cudd_DumpDot()`), οπότε μπορείτε από το πρόγραμμα σας να καλείτε το πακέτο Graphviz για να οπτικοποιήσετε τα διαγράμματά σας.

Προαιρετικά και για δική σας διευκόλυνση κατά την ανάπτυση της εργασία μπορείτε να υλοποιήσετε και τις ακόλουθες TCL εντολές:

- `report_component_postfix_boolean_function <component name>`
- `convert_infix_to_postfix <infix expression>`

Η πρώτη θα τυπώνει τη συνάρτηση ενός component του σε postfix μορφή ενώ η δεύτερη θα λαμβάνει ως είσοδο μια infix έκφραση και θα την μετατρέπει σε postfix.

4 Μερος B: Circuit Levelization

Σε αυτό το μέρος τις εργασίας καλείστε να ταξινομήσετε τα gatepins του κυκλώματος κατά τοπολογική σειρά (Toposort). Η ταξινόμηση τους θα ξεκινάει από τις εισόδους του κυκλώματος (Primary inputs) και τις εξόδους των Flip-Flops, αλλιώς STA startpoints, και θα φτάνει μέχρι τις εξόδους του κυκλώματος (Primary Outputs) και τις εισόδους των Flip-Flops, αλλιώς STA endpoints.

Οι σχετικές TCL εντολές που θα πρέπει να υποστηρήξετε στο εργαλείο σας είναι οι εξής:

- `report_gatepin_level <gatepin name>`
- `report_gatepins_levelized`
- `report_level_gatepins <level>`

Η πρώτη θα παίρνει ως παράμετρο το όνομα από ένα gatepin και θα ενημερώνει τον χρήστη για το level στο οποίο αυτό ανήκει. Προαιρετικά μπορείτε να κάνετε την εντολή να δέχεται μια λίστα από gatepins και να τυπώνει το επίπεδο για κάθε ένα από αυτά. Η δεύτερη θα τυπώνει για όλα τα επίπεδα του κυκλώματος μια λίστα με τα gatepins που ανήκουν στο εκάστοτε επίπεδο. Η τελευταία θα παίρνει ως παράμετρο ένα level του design και θα τυπώνει μια λίστα με όλα τα gatepins αυτού του level.

5 Παράδειγμα χρήσης του CUDD

Έστω ότι θέλουμε να κατασκευάσουμε το BDD της συνάρτησης $((A * B) + C)$. Στο παρακάτω listing 1 παρουσιάζεται ένας τρόπος με τον οποίο μπορεί να γίνει αυτό. Παρατηρούμε λοιπόν ότι στην αρχή αρχικοποιούμε τον BDD manager με τη συνάρτηση `Cudd_Init()` *(line 10)*, τα ορίσματα της οποίας είναι τα default προτεινόμενα από το decumentation του πακέτου. Έπειτα πάμε να ορίσουμε τις μεταβλητές του bdd μας με τη συνάρτηση `Cudd_bddNewVar()` *(lines 14-16)* και μετά κατασκευάζουμε το BDD χρησιμοποιώντας τις εσωτερικές boolean συναρτήσεις που προσφέρει το πακέτο *(lines 18-19)*.

Προσέξτε στην γραμμή 22 ότι εφόσον φτιάξουμε το BDD μας καλούμε την συνάρτηση `Cudd_Ref()` για να αυξήσουμε τον αριθμό αναφορών του node του BDD. Αυτό το κάνουμε γιατί το CUDD έχει εσωτερικό σύστημα garbage collection που βασίζεται σε αριθμό αναφορών, για πιο αποδοτική χρήση της μνήμης. Όταν λοιπόν μια εφαρμογή φτιάχνει ένα διάγραμμα αποφάσεων πρέπει να αυξήσει των αριθμό αναφορών του καλώντας την προαναφερθείσα συνάρτηση. Όμοια όταν ένα διάγραμμα δεν είναι πλέον απαραίτητο στην εφαρμογή θα πρέπει μέσω της συνάρτησης `Cudd_RecursiveDeref()` ώστε να ανακυκλωθεί η μνήμη του.

```
1 int main(int argc, char const *argv[])
2 {
3   DdManager *gbm; // Global BDD manager. //
```

```
  DdNode *bdd;
  DdNode *var_a;
  DdNode *var_b;
  DdNode *var_c;

  // Initialize a new BDD manager. //
  gbm = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0); /* Initialize a new BDD manager. */
  // ((A & B) | C) //

  // create bdd variables //
  var_a = Cudd_bddNewVar(gbm);
  var_b = Cudd_bddNewVar(gbm);
  var_c = Cudd_bddNewVar(gbm);

  bdd = Cudd_bddAnd(gbm, var_a, var_b); // create A & B //
  bdd = Cudd_bddOr(gbm, bdd, var_c); // create (A & B) | C //

  // update the reference count for the node just created //
  Cudd_Ref(bdd);

  // quit manager //
  Cudd_Quit(gbm);

  return 0;
}
```

Listing 1: Παράδειγμα χρήσης τους CUDD για την κατασκευή του BDD της συνάρτησης ((A B) | C).

6 Παρουσίαση και Demo

Προετοιμάστε ενα σύντομο ppt και μια επίδεξη του εκτελέσιμου για την ώρα του μαθήματος. Η προθεσμία υποβολής 3ης Εργασίας είναι η **14/1/2024**. Μέχρι τότε θα πρέπει να έχετε υποβάλλει τα αρχεία της εργασίας στο `e-Class`