Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Απόδειξη ορθότητας μίας υλοποίησης του αλγορίθμου των Ford-Fulkerson για την εύρεση της ελάχιστης τομής γράφου χωρίς βάρη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΧΑΗΛ ΓΙΑΚΚΟΥΠΗΣ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Επίκ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2009

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Απόδειξη ορθότητας μίας υλοποίησης του αλγορίθμου των Ford-Fulkerson για την εύρεση της ελάχιστης τομής γράφου χωρίς βάρη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΧΑΗΛ ΓΙΑΚΚΟΥΠΗΣ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Επίκ. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Ιουλίου 2009.

Νικόλαος Παπασπύρου Επίκ. Καθηγητής Ε.Μ.Π.	Κωστής Σαγώνας Αν. Καθηγητής Ε.Μ.Π.	Στάθης Ζάχος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2009

Μιχαήλ Γιακκούπης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της εργασίας αυτής είναι η τυπική επαλήθευση της λύσης ενός προβλήματος παρόμοιου με τα προβλημάτα που εξετάζονται στην Ολυμπιάδα Πληροφορικής. Η τυπική επαλήθευση ενός προγράμματος αποτελεί τη διαδικασία απόδειξης ορθότητας του προγράμματος με βάση κάποια τυπική προδιαγραφή ή ιδιότητα, χρησιμοποιώντας ως εργαλεία τυπικές μεθόδους και μαθηματική λογική.

Στην περίπτωσή μας υλοποιήσαμε μία λύση για το πρόβλημα με τίτλο `damage2`, που ήταν ένα από τα θέματα του διαγωνισμού πληροφορικής του Μαρτίου 2009 της USACO. Το πρόβλημα αυτό ανάγεται εύκολα στο γνωστό από τη θεωρία γράφων πρόβλημα της εύρεσης ελάχιστης τομής (min cut) γράφου χωρίς βάρη, για τη λύση του οποίου μπορεί να χρησιμοποιηθεί ο αλγόριθμος των Ford-Fulkerson, Στη συνέχεια αποδείξαμε την ορθότητα της υλοποίησης αυτού του αλγορίθμου σε γλώσσα C, χρησιμοποιώντας τα εργαλεία Caduceus και Coq.

Λέξεις κλειδιά

Αλγόριθμος Ford-Fulkerson, πρόβλημα μέγιστης ροής/ ελάχιστης τομής, απόδειξη ορθότητας, πρόβλημα damage2 από USACO, Coq.

Abstract

The purpose of this diploma project is the formal verification of a problem similar to those that are used in the International Olympiad in Informatics. Formal verification of a program is the process of proving its correctness, based on a formal specification or property, using formal methods and mathematical logic.

In this case, we implemented a solution to the problem `damage2`, which was used in USACO's computing contest of March 2009. This problem is easily reduced to the well-known graph-theoretic problem of finding the minimum cut in a graph with no weights, for which we can obtain a solution using the Ford-Fulkerson algorithm. We then proved the correctness of an implementation of this algorithm in C, using the tools Caduceus and Coq.

Key words

Ford-Fulkerson algorithm, max-flow/ min-cut problem, correctness proof, USACO problem damage2, Coq.

Ευχαριστίες

Ευχαριστώ πολύ την οικογένειά μου και όλους όσους μου στάθηκαν, με στήριξαν και με βοήθησαν όλα αυτά τα χρόνια των σπουδών μου. Επίσης ευχαριστώ ιδιαιτέρως τον καθηγητή μου κ. Παπασπύρου για την πολύτιμη βοήθειά του κατά τους τελευταίους μήνες, αλλά και καθ' όλη τη διάρκεια της φοίτησης μου στο ΕΜΠ.

Μιχαήλ Γιακκούπης,

Αθήνα, 20 Ιουλίου 2009.

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-5-09, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Ιούλιος 2009.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός της εργασίας

Στην παρούσα διπλωματική εργασία στόχος μας είναι η τυπική επαλήθευση ενός προγράμματος που υλοποιεί τον αλγόριθμο των Ford-Fulkerson για το πρόβλημα του min cut στην περίπτωση που ο γράφος μας είναι κατευθυνόμενος και δεν έχουμε βάρη στις ακμές, χρησιμοποιώντας εργαλεία που δίνουν τη δυνατότητα μηχανιστικής απόδειξης θεωρημάτων.

Η τυπική επαλήθευση ενός προγράμματος αποτελεί τη διαδικασία απόδειξης της ορθότητας του προγράμματος με βάση κάποια τυπική προδιαγραφή ή ιδιότητα χρησιμοποιώντας ως εργαλεία τυπικές μεθόδους και μαθηματική λογική. Η χρησιμότητα της διενέργειας της τυπικής επαλήθευσης πηγάζει από το γεγονός ότι ο δυναμικός έλεγχος δεν επαρκεί για την απόδειξη ότι κάποιο λογισμικό δεν περιέχει σφάλματα ή ότι έχει κάποια συγκεκριμένη ιδιότητα. Βέβαια ούτε η τυπική επαλήθευση μπορεί να πετύχει το πρώτο, καθώς δεν είναι πάντα εφικτός ο προσδιορισμός της έννοιας της απουσίας λαθών από ένα πρόγραμμα. Είναι όμως δυνατό να αποδειχθεί ότι το συγκεκριμένο λογισμικό έχει κάποιες ιδιότητες που το καθιστούν λειτουργικό και χρήσιμο.

Όπως αναφέραμε η απόδειξη της ορθότητας που θα επιχειρήσουμε θα γίνει χρησιμοποιώντας εργαλεία μηχανιστικής απόδειξης θεωρημάτων. Η μηχανιστική απόδειξη θεωρημάτων ασχολείται με τη διατύπωση και απόδειξη θεωρημάτων με τυπικό τρόπο με την υποστήριξη ειδικών εργαλείων λογισμικού (proof assistants). Η χρήση του όρου «μηχανιστική» δεν είναι τυχαία καθώς, παρόλο που τα εργαλεία αυτά υποστηρίζουν αυτόματη απόδειξη θεωρημάτων, στην περίπτωση που αυτή δεν είναι εφικτή γίνεται ημι-αυτόματη ή και πλήρως χειροκίνητη και οδηγούμενη από τον προγραμματιστή απόδειξη.

Στην παρούσα εργασία χρησιμοποιήσαμε τη γλώσσα C και τα εργαλεία απόδειξης θεωρημάτων (proof assistants) Caduceus και Coq. Το Caduceus είναι ένα εργαλείο ελεύθερου λογισμικού που κατασκευάζει ένα σύνολο υποχρεώσεων απόδειξης (proof obligations), τις οποίες προσπαθεί να αποδείξει αυτόματα. Όσες δεν καταφέρνει να αποδείξει, τις αποδεικνύουμε με τη βοήθεια του Coq που είναι ένα εργαλείο μηχανικής απόδειξης θεωρημάτων.

Μέσα από αυτή τη διαδικασία εκτός από την ίδια την απόδειξη θέλουμε να επικεντρωθούμε στη δυσκολία της μηχανιστικής απόδειξης θεωρημάτων την οποία χειριζόμαστε ως εργαλείο. Θέλουμε δηλαδή να ερευνήσουμε κατά πόσο η διαδικασία μηχανιστικής απόδειξης θεωρημάτων προσφέρει ουσιαστικά έναν εύκολο και αποδοτικό τρόπο για την απόδειξη θεωρημάτων και έχει φτάσει σε ένα σημείο που να μπορεί να μας βοηθήσει στην τυπική επαλήθευση προγραμμάτων όπως αυτό που μελετούμε. Από τις δυσκολίες που συναντήσαμε κατά τη διάρκεια της απόδειξης και γενικά το σύνολο των προβλημάτων που είχαμε, θα γίνει εμφανές ότι παρόλο που τα εργαλεία αυτά είναι σε μεγάλο βαθμό αξιόπιστα και σε περίπτωση που χρησιμοποιηθούν σωστά μπορούν να οδηγήσουν σε έγκυρες αποδείξεις, η δυσκολία της διαδικασίας της απόδειξης παραμένει σε αρκετά υψηλά επίπεδα. Μάλιστα σε συγκεκριμένες περιπτώσεις θα οδηγηθούμε στο συμπέρασμα ότι η απόδειξη με χρήση των εργαλείων αυτών είναι ακόμη πιο επίπονη από την αντίστοιχη απόδειξη με το χέρι.

1.2 Δομή της εργασίας

Η δομή της παρούσας εργασίας ακολουθεί την παρακάτω λογική. Στο Κεφάλαιο 2 παρουσιάζουμε με συντομία τα εργαλεία που χρησιμοποιήσαμε για την απόδειξη της ορθότητας του αλγορίθμου. Πρόκειται για τα εργαλεία Caduceus και Coq, τα οποία είναι βοηθητικά εργαλεία απόδειξης θεωρημάτων (proof assistants). Στη συνέχεια και στο Κεφάλαιο 3, παρουσιάζουμε το ακριβές πρόβλημα του USACO (USA Computing Olympiad) με το οποίο ασχοληθήκαμε, καθώς και τη σχέση του με το γνωστό πρόβλημα της θεωρίας γράφων min cut και τον τρόπο με τον οποίο κάναμε αναγωγή του πρώτου στο δεύτερο. Στο Κεφάλαιο 4 περιγράφουμε τον αλγόριθμο των Ford-Fulkerson που επιλύει το πρόβλημα του min cut στη γενική περίπτωση και την απλοποίησή του που έχουμε στην περίπτωσή μας. Επιπλέον αναφέρουμε τη γενική περιγραφή του τρόπου απόδειξης του αλγορίθμου. Στο επόμενο κεφάλαιο, καταγράφουμε τον κώδικα στη γλώσσα C που υλοποιεί τον παραπάνω αλγόριθμο, καθώς και τις υποχρεώσεις απόδειξης που συμπληρώσαμε, ώστε να είναι εφικτή η απόδειξη της ορθότητας του προγράμματος. Στο Κεφάλαιο 6, έχουμε την αναλυτική εξέταση της απόδειξης. Τέλος, στο Κεφάλαιο 7, παραθέτουμε τα αποτελέσματά και τα συμπεράσματά μας.

Κεφάλαιο 2

Τα εργαλεία που χρησιμοποιήσαμε

Σε αυτό το κομμάτι της εργασίας θα αναφερθούμε συνοπτικά στα εργαλεία που χρησιμοποιήσαμε για να αποδείξουμε την ορθότητα του αλγορίθμου των Ford-Fulkerson που επιλύει το πρόβλημα του min cut. Χωρίζουμε το κεφάλαιο αυτό σε δύο τμήματα τα οποία και αφιερώνουμε στο εργαλείο Caduceus και Coq αντίστοιχα.

2.1 Το εργαλείο Caduceus

Εδώ παρουσιάζουμε συνοπτικά το εργαλείο Caduceus, το οποίο και χρησιμοποιήσαμε στα πρώτα στάδια της απόδειξής μας, ώστε να εξάγουμε από τον κώδικα που γράψαμε σε γλώσσα C και από τις αντίστοιχες υποχρεώσεις αποδείξεων (proof obligations) κώδικα συμβατό με το εργαλείο Coq.

Σύμφωνα με το [Fili09b] το Caduceus είναι ένα εργαλείο απόδειξης θεωρημάτων για τους προγραμματιστές της γλώσσας C. Δίνει τη δυνατότητα στον προγραμματιστή να εισάγει πληροφορίες όσον αφορά τις ιδιότητες και τα χαρακτηριστικά του προγράμματός του υπό μορφή σχολίων μέσα στον κώδικα. Ακολούθως, το εργαλείο αυτό αναλαμβάνει να μετατρέψει τα σχόλια αυτά, που προφανώς έχουν κάποια τυπική μορφή, σε υποχρεώσεις αποδείξεων (proof obligations) χρησιμοποιώντας ένα εργαλείο το οποίο ονομάζεται Why.

Το εργαλείο Why [Fili09a] έχει πάρει το όνομά του ακριβώς από την ιδιότητά του να αποδεικνύει το λόγο για τον οποίο ένα πρόγραμμα είναι ορθό. Ως γνωστόν, ένα πρόγραμμα σε οποιαδήποτε γλώσσα προγραμματισμού οδηγεί μία μηχανή λέγοντας της πώς να υπολογίσει μία συγκεκριμένη έξοδο συναρτήσει της εισόδου. Όμως δε μας λέει γιατί το πρόγραμμα αυτό είναι σωστό, αν εξαιρέσουμε πιθανότατα κάποια σχόλια που μπορεί να δίνουν κάποια στοιχεία προς αυτή την κατεύθυνση. Αυτό ακριβώς μας προσφέρει το εργαλείο Why μέσα από τις υποχρεώσεις αποδείξεων (proof obligations) που βοηθάει να παραχθούν.

Η πρωτοτυπία του εργαλείου Caduceus σε σχέση με όσα προϋπήρχαν βρίσκεται ακριβώς στο γεγονός που αναφέραμε παραπάνω, δηλαδή στη δυνατότητά του να επεξεργάζεται απευθείας κώδικα C, στον οποίο έχουμε τοποθετήσει υπό μορφή σχολίων τις προδιαγραφές (specification) του προβλήματός μας. Αυτό μοιάζει σε μεγάλο βαθμό με το μοντέλο της γλώσσας Java, όπου σε μορφή σχολίων μας δίνονται διάφορα χαρακτηριστικά του προγράμματος σε ορισμένες περιπτώσεις.

Έτσι το Caduceus [Hube05] είναι ένα εργαλείο παραγωγής αποδεικτικών συνθηκών στηριζόμενο σε μια σειρά από προαπαιτούμενα και μετασυνθήκες που θέτουμε στο πρόγραμμά μας. Οι παραγόμενες αυτές συνθήκες δημιουργούνται χρησιμοποιώντας το λογισμό της πιο αδύναμης συνθήκης και μια μοναδική μετάφραση σε συναρτησιακό προγραμματισμό. Ένα μοναδικό στοιχείο που προσθέτει το Caduceus σε σχέση με παλαιότερα εργαλεία είναι η δυνατότητα του χρήστη να χρησιμοποιήσει πολλούς διαφορετικούς βοηθούς αποδείξεων (proof assistants), οι οποίοι μπορεί να είναι είτε αυτόματοι είτε διαδραστικοί ανάλογα με την επιλογή του χρήστη. Σε αυτή την εργασία, όπως έχουμε ήδη αναφέρει χρησιμοποιούμε το εργαλείο Coq για την περαιτέρω απόδειξη θεωρημάτων.

2.2 Το εργαλείο Coq

Σε αυτό το τμήμα παρουσιάζουμε εν συντομία τη λογική με την οποία δουλεύει το Coq και τη χρησιμότητά του. Τέλος αναφερόμαστε στην έκδοση που χρησιμοποιείται σήμερα και την οποία έχουμε χρησιμοποιήσει και στην παρούσα εργασία για την απόδειξή μας.

Το Coq [Bert04] είναι ένα βοηθητικό εργαλείο (proof assistant) απόδειξης θεωρημάτων της μαθηματικής λογικής και όχι μόνο. Επιτρέπει τη διατύπωση εικασιών, προτάσεων κτλ. των μαθηματικών και ελέγχει την απόδειξη τους κατά αυστηρό και μηχανικό τρόπο. Όταν ολοκληρώσει την εν λόγω διαδικασία εξάγει μία κατασκευαστική απόδειξη (constructive proof) των προδιαγραφών που έχουν τεθεί από το χρήστη, η οποία στην ουσία αποτελεί τη «διαδρομή» μέχρι τη λύση. Η αποδεικτική διαδικασία είναι κατασκευαστική υπό την έννοια ότι εκφράζει την ύπαρξη ενός αντικειμένου που αντιστοιχεί στην πρόταση προς απόδειξη και παράγει το πρόγραμμα που αντιστοιχεί στην απόδειξη αυτή. Η εν λόγω προσέγγιση αναφέρεται στη βιβλιογραφία ως constructive ή intuitionistic και διαφέρει από την κλασική λογική στην οποία ισχύει η αρχή: $A \vee (notA) = True$, γνωστή και ως κανόνας του αποκλεισμένου μέσου (excluded middle rule). Ωστόσο μέχρι σήμερα έχουν αναπτυχθεί βιβλιοθήκες που δίνουν τη δυνατότητα εφαρμογής της συγκεκριμένης αρχής στα διάφορα στάδια των αποδείξεων. Επίσης διαθέτει ισχυρό διαδραστικό σύστημα αλληλεπίδρασης με το χρήστη (interactive theorem proving), ενώ παράλληλα παρέχει ένα πλήρες σύνολο εντολών tactics για την βήμα προς βήμα απόδειξη των θεωρημάτων που διατυπώνει ο χρήστης. Στις προδιαγραφές/δυνατότητες του Coq περιλαμβάνεται σύστημα εξαρτημένων τύπων (dependent types) και επαγωγικών ορισμών (inductive definitions).

Το Coq χρησιμοποιείται για να διασφαλίζει την ορθότητα προγραμμάτων (τυπική επαλήθευση). Είναι επίσης ένα εργαλείο που μας επιτρέπει να αναπτύσσουμε μαθηματικές αποδείξεις με μία πολύ εκφραστική γλώσσα λογικής, η οποία ονομάζεται λογική υψηλού επιπέδου (higher-order logic). Αυτές οι αποδείξεις κατασκευάζονται με ένα διαδραστικό τρόπο όπως περιγράφηκε και παραπάνω. Μπορεί επίσης να χρησιμοποιηθεί σαν ένα λογικό πλαίσιο στο οποίο μπορούν να δοθούν τα αξιώματα μιας νέας λογικής και να παραχθούν αποδείξεις για αυτή τη λογική.

Το Coq στηρίζεται στη συναρτησιακή γλώσσα OCAML και αναπτύχθηκε από το ινστιτούτο INRIA της Γαλλίας, ενώ και στις μέρες μας επεκτείνεται από ανεξάρτητους προγραμματιστές. Τη σημερινή του μορφή την έχει πάρει το 2004 όποτε και παρουσιάστηκε η έκδοση 8. Η τελευταία έκδοση που βρίσκεται σε χρήση σήμερα και έχει χρησιμοποιηθεί στην παρούσα απόδειξη είναι η έκδοση 8.2. η οποία ανακοινώθηκε το Μάρτιο του 2009.

Κεφάλαιο 3

Περιγραφή του προβλήματος και της λογικής επίλυσης

Στην παρούσα ενότητα δίνουμε μια περιγραφή του προβλήματος που καλούμαστε να επιλύσουμε και στη συνέχεια να αποδείξουμε την ορθότητά του. Ο ορισμός του προβλήματος σύμφωνα με την εκφώνηση που υπάρχει στην ιστοσελίδα του USACO [USAC] είναι ο εξής:

Έχουμε τη φάρμα του κυρ Γιάννη, η οποία αποτελείται από ένα σύνολο βοσκοτοπιών και ένα σύνολο μονοπατιών μεταξύ τους. Σε ορισμένα από αυτά τα βοσκοτόπια υπάρχουν αγελάδες οι οποίες βόσκουν. Μετά από ένα σεισμό ένα σύνολο βοσκοτοπιών καταστρέφεται, αλλά, παραδόξως, κανένα από τα μεταξύ τους μονοπάτια δεν υφίσταται κάποια καταστροφή. Ακολούθως, οι αγελάδες πρέπει να γυρίσουν πίσω στον αχυρώνα όπου τους περιμένει ο κυρ Γιάννης. Όμως, κάποιες από αυτές δεν καταφέρνουν να επιστρέψουν, καθώς στο δρόμο προς τη φάρμα συναντούν κάποιο κατεστραμμένο βοσκοτόπι, το οποίο δεν μπορούν να διασχίσουν. Αυτές οι αγελάδες ειδοποιούν τον κυρ Γιάννη μέσω κινητού τηλεφώνου (moobile) ότι δεν καταφέρνουν να προσεγγίσουν τη φάρμα και του αναφέρουν σε ποιο βοσκοτόπι βρίσκονται. Είναι προφανές ότι δεν είναι δυνατό να βρίσκονται σε κάποιο γειτονικό βοσκοτόπι της φάρμας, γιατί τότε δε θα ήταν δυνατό να μην έχουν πρόσβαση σε αυτήν, διότι αναφέραμε ότι κανένα από τα μονοπάτια δεν έχει υποστεί κάποια καταστροφή. Τέλος, ο κυρ Γιάννης συγκεντρώνοντας τις αναφορές από τις αγελάδες επιχειρεί να υπολογίσει ποιος είναι ο μικρότερος δυνατός αριθμός βοσκοτοπιών ο οποίος καταστράφηκε.

Εκφράζοντας το πρόβλημα με αυστηρότερους όρους έχουμε τον παρακάτω ορισμό:

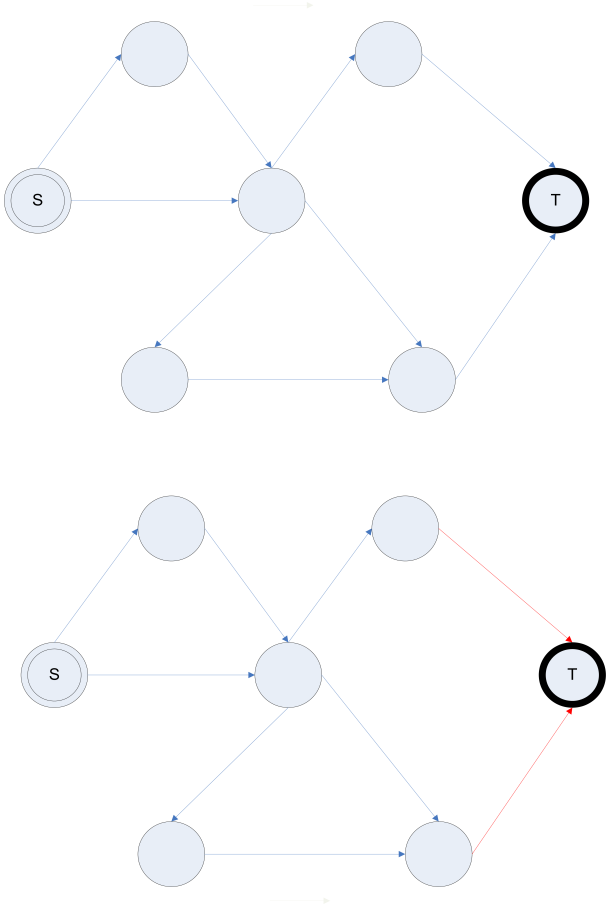
Ορισμός 3.1: Έχουμε ένα μη-κατευθυνόμενο γράφο, για τον οποίο κάθε κόμβος αντιστοιχεί σε ένα βοσκοτόπι του προβλήματος και κάθε ακμή σε ένα μονοπάτι. Σε αυτόν επιλέγουμε έναν κόμβο-προορισμό (το σημείο όπου βρίσκεται η φάρμα), καθώς και μία σειρά από κόμβους-πηγές μη γειτονικούς προς αυτόν, από τους οποίους θεωρούμε ότι δεν υπάρχει κάποιο μονοπάτι προς τον κόμβο-προορισμό (τα βοσκοτόπια από όπου είχαμε αναφορές αγελάδων που δεν μπορούν να προσεγγίσουν τη φάρμα). Στη συνέχεια, προσπαθούμε να εντοπίσουμε ποιος είναι ο ελάχιστος αριθμός κόμβων που αν αφαιρεθούν από το γράφο δε θα υπάρχει όντως πρόσβαση από τους κόμβους-πηγές προς τον κόμβο-προορισμό. Βέβαια, στους κόμβους αυτούς δεν είναι δυνατό να συμπεριλαμβάνονται ούτε ο κόμβος-προορισμός, αλλά ούτε οι κόμβοι-πηγές.

Από τον ορισμό του προβλήματος αυτού παρατηρούμε ότι υπάρχει μια μεγάλη ομοιότητα με το γνωστό από τη θεωρία γράφων πρόβλημα του min cut. Σύμφωνα με τον ορισμό που δίνει η Wikipedia (`http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem#Definition`) για το min cut έχουμε:

Ορισμός 3.2: Το min cut ενός κατευθυνόμενου γράφου είναι ίσο με την ελάχιστη χωρητικότητα που πρέπει να αφαιρεθεί από αυτόν ώστε να μην υπάρχει κάποιο μονοπάτι από έναν κόμβο-πηγή σε κάποιο κόμβο-προορισμό. Ως χωρητικότητα ορίζεται το άθροισμα των βαρών των ακμών που θα πρέπει να αφαιρεθούν από το γράφο.

Γίνεται αντιληπτό ότι οι διαφορές μεταξύ του προβλήματός μας και του min cut όπως το ορίσαμε είναι αυτές που αναφέρονται στον πίνακα 1.1.

Για να μπορέσουμε να αναγάγουμε το πρόβλημά μας στο min cut θα χρειαστούμε ένα επιπλέον θεώρημα το οποίο έχει αποδείξει ο Karl Menger το 1927 [Meng27, Thol08].



Σχήμα 3.1: Σχηματική παρουσίαση του προβληματος min cut

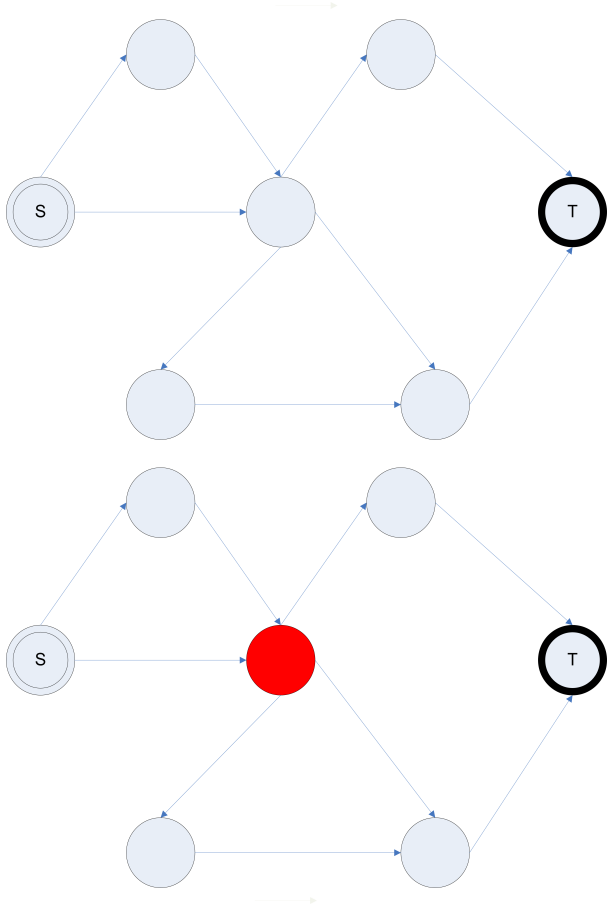
Πίνακας 3.1: Σύγκριση μεταξύ των προβλημάτων

Πρόβλημα USACO	Min cut
Μη κατευθυνόμενος γράφος	Κατευθυνόμενος γράφος
Πολλοί κόμβοι-πηγές	Ένας κόμβος-πηγή
Δεν υπάρχει μονοπάτι μετά από αφαίρεση κόμβων του γράφου	Δεν υπάρχει μονοπάτι μετά από αφαίρεση ακμών του γράφου

Θεώρημα 3.1: *Έστω ένας μη-κατευθυνόμενος πεπερασμένος γράφος και x, y δύο μη γειτονικοί κόμβοι του. Τότε το μέγεθος της ελάχιστης τομής όσον αφορά τους κόμβους (minimum vertex cut) για τους κόμβους x και y είναι ίσο με το μέγιστο αριθμό μονοπατιών από το x στο y, τα οποία έχουν ανά δύο διαφορετικούς κόμβους (κανένας κόμβος ενός μονοπατιού δεν περιέχεται σε οποιοδήποτε άλλο μονοπάτι).*

Παρατηρούμε ότι αυτό το θεώρημα γεφυρώνει σε μεγάλο βαθμό τις διαφορές που υπάρχουν μεταξύ του min cut και του προβλήματος που έχουμε, καθώς συνδέει την έννοια του πλήθους των μονοπατιών μεταξύ δύο κόμβων με την έννοια της ελάχιστης τομής όσον αφορά τους κόμβους (minimum vertex cut). Όμως και πάλι έχουμε μια βασική διαφορά, διότι στην περίπτωση του min cut αναφερόμαστε σε αφαίρεση ακμών από το γράφο, ενώ στην περίπτωση του θεωρήματος του Menger (όπως και στη δική μας) καλούμαστε να αφαιρέσουμε μια σειρά από κόμβους από το γράφο. Για να το πετύχουμε αυτό χρησιμοποιούμε τα αποτελέσματα που υπάρχουν στην εργασία [Thol08], σύμφωνα με την οποία θα πρέπει να μετασχηματίσουμε τον αρχικό μας γράφο χρησιμοποιώντας μια σειρά από διαδοχικά βήματα:

1. Έστω ο μη κατευθυνόμενος γράφος που έχουμε στο πρόβλημά μας.



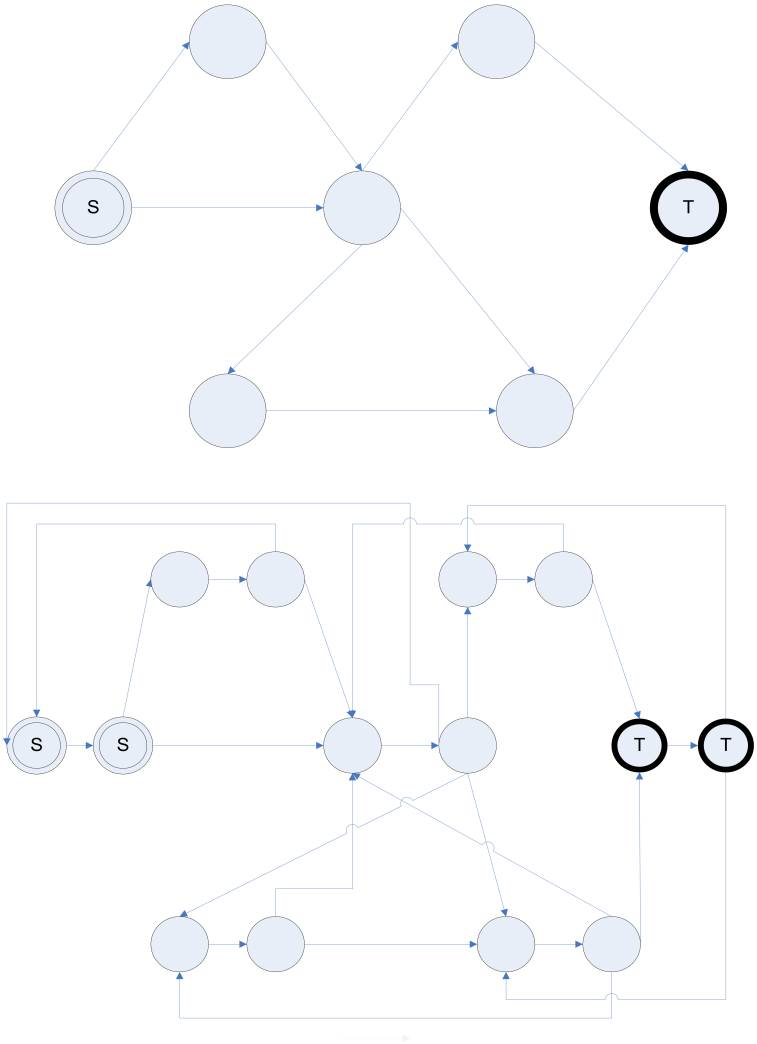
Σχήμα 3.2: Σχηματική απεικόνιση του θεωρήμστος του Menger

2. Για κάθε κόμβο v του αρχικού γράφου δημιουργούμε δύο διαφορετικούς κόμβους τους οποίους ονομάζουμε v_{in} και v_{out}. Ανάμεσά τους τοποθετούμε μία ακμή με κατεύθυνση από τον κόμβο v_{in} προς τον κόμβο v_{out}.

3. Για κάθε ακμή του παλιού γράφου από έναν κόμβο u σε έναν άλλο κόμβο v τοποθετούμε δύο ακμές: μία ακμή από τον κόμβο u_{out} προς τον κόμβο v_{in} και μία από τον κόμβο v_{out} προς τον u_{in}.

4. Εκτελούμε στο γράφο αυτό κάποιον αλγόριθμο εύρεσης του min cut και το αποτέλεσμα που θα βρούμε είναι ίσο με το minimum vertex cut του αρχικού γράφου.

Με αυτό τον τρόπο έχουμε καταφέρει να αναγάγουμε το αρχικό μας πρόβλημα σε ένα ισοδύναμο στο οποίο θα πρέπει να εκτελέσουμε απλά κάποιο γνωστό αλγόριθμο εύρεσης του min cut. Όμως και πάλι έχουμε το πρόβλημα ότι στην κλασική περίπτωση του min cut έχουμε έναν κόμβο-πηγή κι έναν κόμβο-προορισμό. Στην περίπτωσή μας, αντιθέτως, μπορούμε να έχουμε πολλούς κόμβους-πηγές. Γι αυτό το λόγο θα ήταν δυνατό να προσθέσουμε έναν επιπλέον κόμβο-πηγή που θα περιείχε ακμές προς όλους τους άλλους κόμβους-πηγές, έτσι ώστε να πετύχουμε να έχουμε ακριβή ταύτιση των δύο προβλημάτων. Επειδή όμως αυτό δημιουργούσε μια σειρά από προβλήματα που κρίθηκαν δυσεπίλυτα, επιλέξαμε να χρησιμοποιήσουμε μια διαφορετική προσέγγιση για να επιλύσουμε το τελευταίο αυτό πρόβλημα. Έτσι αντιστρέψαμε τους ρόλους των κόμβων-πηγών και του κόμβου-προορισμού, θέτοντας τους πρώτους ως κόμβους-προορισμούς και τον τελευταίο ως κόμβο-πηγή. Στο αμέσως παρακάτω κεφάλαιο παρουσιάζουμε τον αλγόριθμο των Ford-Fulkerson και δείχνουμε πως αυτή η αλλαγή στον αρχικό μας γράφο δε δημιουργεί κανένα πρόβλημα στον αλγόριθμο ενώ επιλύει το πρόβλημά μας.



Σχήμα 3.3: Παρουσίαση της αναγωγής του αρχικού πρβλήματος στο min cut

Σε επόμενο μέρος της εργασίας παρουσιάζουμε επακριβώς τον κώδικα που έχουμε γράψει, όπως επίσης και τα χαρακτηριστικά (specification) του προβλήματος όπως τα εκφράζουμε για να επιτύχουμε την απόδειξη του ζητούμενου. Στο αμέσως επόμενο τμήμα όμως παρουσιάζουμε τη γενική ιδέα της απόδειξης.

Κεφάλαιο 4

Περιγραφή του αλγορίθμου των Ford-Fulkerson και της γενικής προσέγγισης της απόδειξης

Όπως ήδη αναφέραμε στο τμήμα αυτό θα παρουσιάσουμε τον αλγόριθμο των Ford-Fulkerson, καθώς και τη λύση του προβλήματος σε γλώσσα C.

4.1 Αλγόριθμος των Ford-Fulkerson

Σύμφωνα με τον ορισμό του [Corm01] για τον αλγόριθμο των Ford-Fulkerson έχουμε:

Ορισμός 4.1: Η μέθοδος των Ford-Fulkerson είναι επαναληπτική. Αρχικά θέτουμε $f(u, v) = 0$ για όλα τα ζεύγη κόμβων u, $v \in V$, οπότε έχουμε αρχική τιμή ροής ίση με 0. Σε κάθε επανάληψη, αυξάνουμε την τιμή της ροής εντοπίζοντας μια αυξητική διαδρομή την οποία μπορούμε να αντιλαμβανόμαστε απλώς ως μία διαδρομή από τον αφετηριακό κόμβο s προς τον τερματικό κόμβο t η οποία μπορεί να δεχθεί επιπλέον ροή, και στη συνέχεια αυξάνουμε τη ροή κατά μήκος της διαδρομής αυτής. Επαναλαμβάνουμε τη διαδικασία μέχρις ότου να εξαντληθούν οι αυξητικές διαδρομές. Όπως αποδεικνύεται από το θεώρημα μέγιστης ροής-ελάχιστης τομής, η διαδικασία αυτή δίνει κατά τον τερματισμό της μια μέγιστη ροή.

Στην περίπτωσή μας δεν έχουμε βάρη στις ακμές και επομένως έχουμε τη δυνατότητα να απλοποιήσουμε σε πολύ μεγάλο βαθμό τον παραπάνω ορισμό. Ακολούθως, ορίζουμε σε μορφή βημάτων τον αλγόριθμο που υλοποιήσαμε, ο οποίος είναι ο αλγόριθμος των Ford-Fulkerson στην περίπτωση γράφου χωρίς βάρη.

1. Έχουμε έναν κατευθυνόμενο γράφο με ακμές χωρίς βάρη, στον οποίο έχουμε έναν κόμβο-πηγή και μια σειρά από κόμβους-προορισμούς.

2. Αρχικοποιούμε την τιμή του min cut σε 0.

3. Βρίσκουμε έναν μονοπάτι στο γράφο από τον κόμβο-πηγή σε ένα οποιονδήποτε κόμβο-προορισμό.

4. Στην περίπτωση που δεν βρίσκουμε ένα τέτοιο μονοπάτι ο αλγόριθμος τερματίζει και το min cut του αρχικού μας γράφου είναι ίσο με την τρέχουσα τιμή του min cut.

5. Στην περίπτωση που βρήκαμε ένα τέτοιο μονοπάτι αυξάνουμε κατά ένα το τρέχον min cut και αφαιρούμε από το γράφο το σύνολο των ακμών που αποτελούν το μονοπάτι, ενώ ταυτοχρόνως, προσθέτουμε στο γράφο όλες τις αντίστροφες αυτών των ακμών (η αντίστροφη μίας ακμής ορίζεται ως η ακμή που ξεκινάει από τον κόμβο που η αρχική ακμή κατέληγε και καταλήγει στον κόμβο από τον οποίο η αρχική ακμή ξεκινούσε).

6. Οδηγούμαστε και πάλι στην κατάσταση 3, όπου τώρα ο γράφος μας είναι αυτός που κατασκευάζουμε στο βήμα 5.

Όπως είναι προφανές από τα παραπάνω, η ύπαρξη ενός μονοπατιού από έναν κόμβο s σε κάποιον κόμβο t ισοδυναμεί με ροή ίση με 1 για τη μεταξύ τους διαδρομή. Συνεπώς, σε κάθε επανάληψη της διαδικασίας, η μέγιστη ροή-ελάχιστη τομή του γράφου αυξάνεται κατά ένα. Επίσης, η φορά του μονοπατιού δε μας ενδιαφέρει, καθώς ο αρχικός μας γράφος δεν είναι κατευθυνόμενος και η αναγωγή που έχουμε κάνει σέβεται αυτό τον περιορισμό.

Ακολούθως, θα παρουσιάσουμε μια γενική προσέγγιση της απόδειξης που ακολουθήσαμε για τον αλγόριθμο των Ford-Fulkerson. Σε επόμενα κεφάλαια, παραθέτουμε τον ακριβή κώδικα σε γλώσσα C, καθώς και λεπτομερή ανάλυση της αποδεικτικής διαδικασίας.

4.2 Γενική προσέγγιση της απόδειξης

Στο παρόν τμήμα της εργασίας παρουσιάζουμε με συντομία τη βασική ιδέα που ακολουθήσαμε για την απόδειξη, καθώς και τις βασικότερες δυσκολίες που αντιμετωπίσαμε. Η περιγραφή των σπουδαιότερων από τα λήμματα που έπρεπε να αποδείξουμε γίνεται σε επόμενο κεφάλαιο.

Αρχικά, θα πρέπει να παρατηρήσουμε ότι η απόδειξη μας χωρίζεται σε δύο επιμέρους τμήματα. Στο πρώτο, αποδεικνύουμε ότι ο αλγόριθμος των Ford-Fulkerson επιλύει πραγματικά το πρόβλημα του min cut. Στο δεύτερο κομμάτι θα πρέπει να δείξουμε ότι η υλοποίησή μας είναι πραγματικά μια σωστή υλοποίηση του αλγορίθμου των Ford-Fulkerson.

Για να μπορέσουμε να αποδείξουμε την ορθότητα του αλγορίθμου των Ford-Fulkerson, θα πρέπει να έχουμε στη διάθεσή μας έναν ορισμό για το πρόβλημα του min cut, ο οποίος να μας βολεύει στην απόδειξη μας. Αυτός ο ορισμός θα πρέπει να μας επιτρέπει να εκφράσουμε μία αναλλοίωτη ποσότητα για το βρόχο που αναφέραμε ότι περιέχεται στον αλγόριθμο. Επειδή ο ορισμός του προβλήματος όπως τον έχουμε αναφέρει παραπάνω δεν μπορεί να χρησιμοποιηθεί, παραθέτουμε στη συνέχεια έναν ισοδύναμο ορισμό, ο οποίος μας επιτρέπει να εκφράσουμε με ευκολία ακριβώς την αναλλοίωτη αυτή ποσότητα.

Ορισμός 4.2: Έχουμε έναν κατευθυνόμενο γράφο G. Χωρίζουμε τους κόμβους που περιέχονται στο γράφο αυτό σε δύο υποσύνολα vls και vlt. Στο πρώτο υποσύνολο περιέχεται ο κόμβος-πηγή του προβλήματός μας, ενώ στο δεύτερο περιλαμβάνεται το σύνολο των κόμβων-προορισμών του γράφου μας. Το πλήθος των ακμών του γράφου που συνδέουν έναν κόμβο του συνόλου vls με έναν κόμβο του συνόλου vlt με φορά από τον κόμβο του vls προς τον κόμβο του vlt αποτελεί ένα cut του γράφου μας. Το μικρότερο δυνατό τέτοιο πλήθος αποτελεί το min cut του γράφου μας.

Ο ορισμός αυτός προκύπτει άμεσα από την έκφραση του προβλήματος του min cut, σε αντίθεση με τον προηγούμενο που είχαμε αναφέρει ο οποίος θα μπορούσαμε να πούμε ότι πηγάζει από την έκφραση του δυαδικού προβλήματος της μέγιστης ροής. Έτσι με τον ορισμό αυτό θα μπορούσαμε να ισχυριστούμε ότι αν από το γράφο μας αφαιρέσουμε το σύνολο των ακμών που χρησιμοποιούμε για τον υπολογισμό του min cut, τότε δε θα έχουμε κανέναν μονοπάτι από τον κόμβο-πηγή προς οποιονδήποτε κόμβο-προορισμό.

Χρησιμοποιώντας τον παραπάνω ορισμό, θα πρέπει να αποδείξουμε ότι το min cut του τρέχοντος γράφου μειώνεται κατά ένα (εφόσον δεν έχουμε βάρη στις ακμές) σε κάθε επανάληψη του αλγορίθμου των Ford-Fulkerson. Επομένως, ουσιαστικά, θα πρέπει να αποδείξουμε ότι ο αλγόριθμος αυτός σε κάθε επανάληψη αφαιρεί ακριβώς μία από τις ακμές που προσμετρούνται στον υπολογισμό του min cut. Όμως η μόνη αλλαγή που γνωρίζουμε ότι κάνει ο αλγόριθμος πάνω στο γράφο είναι ότι αφαιρεί το σύνολο των ακμών που απαρτίζουν το μονοπάτι που βρήκαμε από τον κόμβο-πηγή προς κάποιον κόμβο-προορισμό και ότι προσθέτει στο γράφο τις ανάποδες αυτών των ακμών. Με την απόδειξη αυτή θα ολοκληρώσουμε το πρώτο τμήμα της εργασίας μας, το οποίο είναι η απόδειξη της ορθότητας του αλγορίθμου των Ford-Fulkerson για το πρόβλημα του min cut.

Όσον αφορά το δεύτερο σημαντικό κομμάτι της απόδειξής μας, περιλαμβάνει την απόδειξη της ισοδυναμίας μεταξύ του αλγορίθμου των Ford-Fulkerson και της λύσης που έχουμε γράψει. Από τα παραπάνω γίνεται φανερό ότι θέλουμε να δείξουμε ότι το πρόγραμμά μας εξασφαλίζει τα εξής τρία:

1. Όσο υπάρχει κάποιο μονοπάτι στο γράφο που έχουμε, υπάρχει μία συνάρτηση η οποία βρίσκει το μονοπάτι.

2. Αφού βρούμε το μονοπάτι, βγάζουμε από το γράφο όλες τις ακμές που το απαρτίζουν και προσθέτουμε όλες τις αντίθετές τους.

3. Όταν τελικά δε βρούμε μονοπάτι στο γράφο θα πρέπει να σταματάμε τον αλγόριθμο και να επιστρέφουμε το min cut, το οποίο είναι το πλήθος των επιτυχημένων αναζητήσεων για μονοπάτι που κάναμε στο γράφο.

Το πρώτο ζητούμενο όπως και το δεύτερο θα πρέπει να αποδεικνύεται ότι γίνονται στη συνάρτηση find_path, ενώ η συνάρτηση min_cut αναλαμβάνει το τρίτο μέρος μόνο. Για να αποδείξουμε τα παραπάνω έχουμε εισάγει μια σειρά από ορισμούς και λήμματα, τα οποία και μας βοηθούν να αποδείξουμε τελικά το ζητούμενο που είναι η ισοδυναμία του προγράμματός μας με τον αλγόριθμο των Ford-Fulkerson για το min cut.

Στο επόμενο κεφάλαιο παρουσιάζουμε το σύνολο των ορισμών και των βοηθητικών λημμάτων που χρησιμοποιήσαμε για την απόδειξη, καθώς και τα βασικότερα λήμματα που προέκυψαν από το caduceus και τη λογική με την οποία τα αποδείξαμε.

Κεφάλαιο 5

Υλοποίηση αλγορίθμου σε γλώσσα C και περιγραφή υποχρεώσεων απόδειξης

Σε αυτό το τμήμα παρουσιάζουμε το βασικότερο κομμάτι του κώδικα που έχουμε γράψει, καθώς και τις υποχρεώσεις απόδειξης (proof obligations) που προσθέσαμε στον κώδικα ώστε να γίνει εφικτή η απόδειξή του στη συνέχεια. Παραθέτουμε το κομμάτι της υλοποίησης του αλγορίθμου των Ford-Fulkerson σε γλώσσα C με τα απαραίτητα σχόλια. Στο πρώτο κομμάτι έχουμε τη συνάρτηση find_path:

```
/*@ requires 1 <= P <= MAXN
  @       && 0 <= count <= MAXEDGE
  @       && good(2*P, count, start, e)
  @       && residualG(makeG(2*P, count, start, e))
  @       && \valid_range(visited, 1, MAXNODE)
  @       && 1 <= x <= 2*P
  @       && visited[x] == false
  @       && (\forall int it;
  @            targetG(makeG(2*P, count, start, e), index(it)) =>
  @            visited[it] == false)
  @ ensures \forall graph oldg, graph newg;
  @           oldg == \old(makeG(2*P, count, start, e)) &&
  @           newg == makeG(2*P, count, start, e) =>
  @           good(2*P, count, start, e) &&
  @           residualG(newg) &&
  @           (\forall int vnum; \old(visited[vnum]) != false =>
  @             visited[vnum] != false) &&
  @           (\forall int vnum; visited[vnum] == false =>
  @             \old(visited[vnum]) == false) &&
  @           (\result != false =>
  @             (\exists Vertex t, V_list vl, A_list al;
  @               targetG(oldg, t) &&
  @               \old(path(oldg, index(x), t, vl, al)) &&
  @               (\forall int vnum; InV(index(vnum), vl) =>
  @                 !\old(visited[vnum])) &&
  @               (\forall Arc a; InA(a, al) =>
  @                 presentG(oldg, a) && presentG(newg, revA(a))) &&
  @               (\forall Arc a; !InA(a, al) && !InA(revA(a), al) =>
  @                 (presentG(newg, a) <=> presentG(oldg, a))))) &&
  @           (\result == false => visited[x] != false &&
  @             (\forall Arc a; presentG(oldg, a) <=>
  @               presentG(newg, a)) &&
  @             (\forall int knum; \old(e[knum].present) <=>
  @               e[knum].present) &&
  @             (\forall int it; targetG(oldg, index(it)) =>
  @               visited[it] == false) &&
  @             (\forall int y, Vertex t, V_list vl, A_list al;
  @               1 <= y <= 2*P &&
  @               targetG(oldg, t) =>
  @               \old(visited[y]) == false && visited[y] != false &&
```

```
  @                (\forall int z; 1 <= z <= 2*P && InV(index(z), vl) =>
  @                  \old(visited[z]) == false) =>
  @                  !path(oldg, index(y), t, vl, al)))
  @*/
bool find_path (int x)
{
  int k;

  if (start[x].target)
    return true;
  visited[x] = true;

  /*@ invariant 0 <= k <= count
    @         && residualG(makeG(2*P, count, start, e))
    @         && good(2*P, count, start, e)
    @         && (\forall int y, Vertex t, V_list vl, A_list al;
    @               1 <= y <= 2*P &&
    @               !InV(index(x), vl) &&
    @               targetG(\old(makeG(2*P, count, start, e)), t) &&
    @               \old(visited[y]) == false &&
    @               visited[y] != false &&
    @               (\forall int z; 1 <= z <= 2*P &&
    @                 InV(index(z), vl) =>
    @                 \old(visited[z]) == false) =>
    @                 !path(\old(makeG(2*P, count, start, e)),
    @                       index(y), t, vl, al))
    @         && edge_from(e, start[x].edge, k)
    @         && (\forall int vnum; \old(visited[vnum]) != false =>
    @              visited[vnum] != false)
    @         && (\forall int vnum; visited[vnum] == false =>
    @              \old(visited[vnum]) == false)
    @         && (\forall Arc a;
    @              presentG(\old(makeG(2*P, count, start, e)), a) <=>
    @              presentG(makeG(2*P, count, start, e), a))
    @         && (\forall int knum;
    @              \old(e[knum].present) <=> e[knum].present)
    @         && visited[x] != false
    @         && (\forall int kr;
    @              1 <= kr <= count &&
    @              presentG(\old(makeG(2*P, count, start, e)),
    @                       make_arc(index(x), index(e[kr].to))) &&
    @              edge_from(e, e[kr].next, k) =>
    @                visited[e[kr].to] != false)
    @         && (\forall int it;
    @              targetG(\old(makeG(2*P, count, start, e)),
    @                      index(it)) => visited[it] == false)
    @*/
  for (k = start[x].edge; k != 0; k = e[k].next)
    if (e[k].present && !visited[e[k].to] && find_path(e[k].to)) {
      e[k].present = false;
      e[e[k].reverse].present = true;
      return true;
    }
  return false;
}
```

Η find_path είναι η συνάρτηση που ψάχνει για κάποιο μονοπάτι μέσα στο γράφο κάνοντας διάσχιση DFS. Σε περίπτωση που βρει ένα τέτοιο μονοπάτι αφαιρεί τις ακμές που το απαρτίζουν από

το γράφο και προσθέτει όλες τις αντίθετες αυτών και επιστρέφει τον καινούργιο γράφο. Στην περίπτωση που δε βρει κάποιο μονοπάτι επιστρέφει την πληροφορία ότι δεν υπάρχει μονοπάτι μεταξύ του κόμβου-πηγής και οποιουδήποτε κόμβου- προορισμού. Ο συγκεκριμένος τρόπος διάσχισης του γράφου επιλέχθηκε ώστε να έχουμε άμεσα τις ακμές που απαρτίζουν το μονοπάτι μετά την εύρεσή του και να μπορούμε εύκολα να τροποποιήσουμε το γράφο κατάλληλα.

Στη συνέχεια μελετούμε τις υποχρεώσεις αποδείξεων που τοποθετήσαμε τόσο σα συνθήκες εισόδου και μετασυνθήκες στη συνάρτηση, όσο και σαν αναλλοίωτη του βρόχου. Παρατηρούμε ότι υπάρχει μεγάλη ομοιότητα μεταξύ της αναλλοίωτης του βρόχου και της μετασυνθήκης στην περίπτωση που η συνάρτηση επιστρέψει false σαν αποτέλεσμα. Αυτό συμβαίνει διότι σε κάποιες περιπτώσεις θα πρέπει, όπως θα δούμε αναλυτικότερα παρακάτω, να εξάγουμε την αναλλοίωτη του βρόχου από τις πληροφορίες που μας επιστρέφει η μετασυνθήκη της συνάρτησης, αλλά και το ανάποδο, Έτσι στη μετασυνθήκη ζητούμε να αποδειχθούν τα εξής:

1. Η εγκυρότητα του εκάστοτε γράφου όσον αφορά ιδιότητες που θα εξηγήσουμε στα επόμενα κεφάλαια.

2. Η διατήρηση του χαρακτηρισμού ενός κόμβου ως κόμβο τον οποίο έχουμε επισκεφθεί για όλους τους γράφους από το σημείο αυτό και έπειτα, αλλά και της ιδιότητας της μη επισκεψιμότητας σε όλους τους προηγούμενους γράφους για όλους τους κόμβους που στον παρόντα γράφο θεωρείται ότι δεν τους έχουμε επισκεφθεί.

3. Η ύπαρξη μονοπατιού από τον τρέχοντα κόμβο προς κάποιο κόμβο-προορισμό και η διενέργεια των απαραίτητων αλλαγών στο γράφο όπως αυτές περιγράφονται σε επόμενο κεφάλαιο στην περίπτωση που η συνάρτηση επιστρέφει true.

4. Η μη ύπαρξη μονοπατιού από τον τρέχοντα κόμβο προς οποιονδήποτε κόμβο-προορισμό, καθώς και η διατήρηση του αναλλοίωτου του γράφου στην περίπτωση που η συνάρτηση επιστρέφει false. Επίσης στην περίπτωση αυτή αποδεικνύεται και ότι δεν έχουμε επισκεφθεί κανέναν από τους κόμβους-προορισμούς

Όσον αφορά την αναλλοίωτη του βρόχου τώρα, οι αποδείξεις που χρειαζόμαστε περιλαμβάνουν τα παρακάτω:

1. Την εγκυρότητα του εκάστοτε γράφου όσον αφορά ιδιότητες που θα εξηγήσουμε στα επόμενα κεφάλαια.

2. Την ιδιότητα ότι δεν υπάρχει μονοπάτι προς οποιονδήποτε κόμβο-προορισμό από κανένα κόμβο που έχουμε επισκεφθεί κι έχουμε εξετάσει και το σύνολο των γειτονικών του κόμβων μέχρι αυτή τη στιγμή.

3. Τη διατήρηση του χαρακτηρισμού ενός κόμβου ως κόμβο τον οποίο έχουμε επισκεφθεί για όλους τους γράφους από το σημείο αυτό και έπειτα, αλλά και της ιδιότητας της μη επισκεψιμότητας σε όλους τους προηγούμενους γράφους για όλους τους κόμβους που στον παρόντα γράφο θεωρείται ότι δεν τους έχουμε επισκεφθεί.

4. Τη διατήρηση του αναλλοίωτου του γράφου.

5. Τη διατήρηση του χαρακτηρισμού του κόμβου που τώρα εξετάζουμε ως κόμβου τον οποίο έχουμε επισκεφθεί

6. Το χαρακτηριμό ως κόμβο που έχουμε επισκεφθεί για κάθε κόμβο που είναι γειτονικός του κόμβου που εξετάζουμε και υπάρχει ακμή που συνδέει τον τελευταίο με τον πρώτο.

7. Τη διατήρηση της μη επισκεψιμότητας του συνόλου των κόμβων-προορισμών

Έχοντας την πληροφορία που μας δίνει η αναλλοίωτη του βρόχου, έχουμε τη δυνατότητα να αποδείξουμε την ισχύ της μετασυνθήκης σε κάθε περίπτωση. Έπειτα χρησιμοποιώντας την πληροφορία της μετασυνθήκης, που επιστρέφεται στη συνάρτηση min_cut, μπορούμε τελικά να αποδείξουμε την ορθότητα του προγράμματός μας που είναι και το τελικό ζητούμενο.

Ακολούθως, παρουσιάζουμε τον κώδικα της δεύτερης συνάρτησης του προγράμματος που ολοκληρώνει την υλοποίηση του αλγορίθμου των Ford-Fulkerson, της min_cut:

```
/*@ requires  1 <= P <= MAXN
  @         && 0 <= count <= MAXEDGE
  @         && good(2*P, count, start, e)
  @         && residualG(makeG(2*P, count, start, e))
  @         && \valid_range(visited, 1, MAXNODE)
  @ ensures   \result == mincutG(\old(makeG(2*P, count, start, e)))
  @*/
int min_cut ()
{
  int num;
  int i;

  num = 0;e
  /*@ invariant num+mincutG(makeG(2*P, count, start, e)) ==
    @                    \old(mincutG(makeG(2*P, count, start, e)))
    @         && residualG(makeG(2*P, count, start, e))
    @*/
  // variant should be the out-degree of SOURCE
  while (true) {
    /*@ invariant 1 <= i <= 2*P+1
      @         && \forall int j; 1 <= j < i => visited[j] == false
      @*/
    for (i=1; i<=2*P; i++)
      visited[i] = false;
    if (find_path(SOURCE))
      num++;
    else
      return num;
  }
}
```

Η συνάρτηση min_cut είναι αυτή που στέλνει το γράφο στην find_path ζητώντας από την τελευταία να εντοπίσει πιθανό μονοπάτι μεταξύ του κόμβου-πηγή και κάποιου κόμβου-προορισμού. Όσο επιστρέφει η πληροφορία εύρεσης μονοπατιού μαζί με έναν καινούργιο γράφο όπως αυτός περιγράφηκε παραπάνω, επαναλαμβάνεται η κλήση της find_path με τον καινούργιο γράφο κάθε φορά. Ταυτόχρονα, αυξάνεται μια μεταβλητή η οποία αρχικά αρχικοποιείται στην τιμή 0. Όταν επιστραφεί η πληροφορία μη εύρεσης άλλου μονοπατιού, η min_cut επιστρέφει τη μεταβλητή που είναι ίση με το min cut του αρχικού γράφου.

Όσον αφορά τις υποχρεώσεις αποδείξεων, παρατηρούμε ότι εδώ είναι πολύ πιο λίγες και ευκολονόητες. Όπως θα δούμε όμως σε επόμενα κεφάλαια, η απόδειξή τους σε καμία περίπτωση δεν μπορεί να θεωρηθεί ευκολότερη. Η μετασυνθήκη μας στην παρούσα συνάρτηση ζητάει, όπως είναι αναμενόμενο, να αποδεικνύεται ότι το αποτέλεσμα που παίρνουμε είναι πραγματικά η ελάχιστη τομή (min cut) του αρχικού μας γράφου. Όσον αφορά τις αναλλοίωτες βρόχων που έχουμε είναι και αυτές αρκετά απλοϊκές. Η πρώτη αναλλοίωτη μας ζητάει να δείξουμε ότι ο γράφος παραμένει έγκυρος όσον αφορά χαρακτηριστικά που θα αναφέρουμε παρακάτω και ότι η ελάχιστη τομή του αρχικού γράφου μας θα είναι ίση με την ελάχιστη τομή του εκάστοτε γράφου που θα έχουμε συν το πλήθος των επαναλήψεων που έχουμε εκτελέσει μέχρι εκείνο το σημείο. Αποδεικνύοντας αυτό θα δούμε στη συνέχεια ότι η απόδειξη της ισχύος της μετασυνθήκης μετατρέπεται σε αρκετά εύκολη υπόθεση. Από την άλλη, η δεύτερη αναλλοίωτη ζητάει να αποδείξουμε ότι η μεταβλητή του βρόχου παραμένει στα όρια που

έχουμε θέσει και ότι σε κάθε επανάληψη του εξωτερικού βρόχου ο πίνακας visited αρχικοποείται με τιμή false, όπως είναι απαραίτητο.

Από τα παραπάνω είναι προφανές ότι τελικά αποδεικνύεται το ζητούμενο, δηλαδή ότι το πρόγραμμά μας επιλύει σωστά τον αλγόριθμο των Ford-Fulkerson για το πρόβλημα της ελάχιστης τομής. Επίσης φαίνεται ότι οι δύο συναρτήσεις αυτές αποτελούν την καρδιά του κώδικα που υλοποιεί τον αλγόριθμο αυτό. Στο επόμενο κεφάλαιο, θα αναλύσουμε σε μεγαλύτερο βάθος την απόδειξη του αλγορίθμου, αναφέροντας μια σειρά από ορισμούς και από λήμματα που κληθήκαμε να αποδείξουμε.

Κεφάλαιο 6

Απόδειξη ορθότητας

Σε αυτό το τμήμα της εργασίας μας παρουσιάζουμε το σύνολο των ορισμών που χρησιμοποιήσαμε για να υλοποιήσουμε την απόδειξη, καθώς και τα λήμματα που κληθήκαμε να αποδείξουμε. Στο τελευταίο μέρος παρουσιάζουμε εκτενέστερα όσα λήμματα που παρουσιάζουν το μεγαλύτερο ενδιαφέρον.

6.1 Ορισμοί που χρησιμοποιήσαμε

Στο παρόν τμήμα της εργασίας μας θα αναφέρουμε με συντομία το σύνολο των ορισμών που χρειαστήκαμε για να ολοκληρώσουμε την απόδειξή μας. Οι ορισμοί αυτοί αναφέρονται είτε σε αντιστοίχιση δομών δεδομένων που χρησιμοποιήσαμε στον κώδικα με ανάλογες δομές που ορίσαμε στη λογική, είτε σε αναδρομικούς ορισμούς που χρειαστήκαμε για τις αναλλοίωτες των βρόχων που ορίσαμε.

Αρχίζουμε από τους ορισμούς που αντιστοιχούν σε δομές δεδομένων που χρησιμοποιήσαμε στην επίλυση του προβλήματος:

1. Vertex: Ως κόμβο ορίσαμε μια αμφιμονοσήμαντη σχέση μεταξύ ενός φυσικού αριθμού και μίας δομής που ονομάσαμε κόμβο.

2. Arc: Ως ακμή ορίσαμε μια διατεταγμένη δυάδα κόμβων η οποία αναπαριστά μια ακμή ενός γράφου.

3. graph: Ως γράφο ορίσαμε μια δομή που περιέχει ένα σύνολο κόμβων, ένα σύνολο ακμών που περιλαμβάνονται σε αυτόν, ένα σύνολο ακμών που είναι παρούσες σε ένα στιγμιότυπο του γράφου κι ένα σύνολο κόμβων που αποτελούν τους κόμβους- προορισμούς. Η ανάγκη για την ύπαρξη των ακμών που είναι παρούσες προκύπτει από το γεγονός ότι για να αποφύγουμε την εισαγωγή και εξαγωγή ακμών από το γράφο, για κάθε ακμή του αρχικού μας γράφου τοποθετούμε και την αντίθετή της, αλλά χαρακτηρίζοντας την ως μη παρούσα.

4. V_list: Ορίσαμε μία δομή λίστας κόμβων που περιλαμβάνει ένα σύνολο κόμβων.

5. A_list: Ορίσαμε μία δομή λίστας ακμών που περιλαμβάνει ένα σύνολο ακμών.

Παρατηρούμε ότι οι παραπάνω ορισμοί αναφέρονται κατά το πλείστον σε δομές που έχουν σχέση με γράφους. Θα πρέπει να παρατηρήσουμε ότι το Coq περιείχε ήδη μια βιβλιοθήκη η οποία εισάγει την έννοια των γράφων και αρκετούς βοηθητικούς ορισμούς. Η βιβλιοθήκη αυτή ονομάζεται GraphBasics και μπορούμε να τη βρούμε στην ιστοσελίδα του Coq:
`http://coq.inria.fr/coq/distrib/current/contribs/GraphBasics.html`.
Όμως επιλέξαμε να μη χρησιμοποιήσουμε την παραπάνω βιβλιοθήκη, καθώς περιείχε διάφορα χαρακτηριστικά άχρηστα σε εμάς, ενώ δε μας έδινε τη δυνατότητα να εισάγουμε την έννοια της παρούσας ακμής σε ένα γράφο, η οποία πιθανότατα θα μπορούσε να εκφραστεί με όρους βάρους των ακμών του γράφου (μία ακμή με βάρος 1 υπάρχει στο γράφο, ενώ μία ακμή με βάρος 0 δεν υπάρχει).

Σε δεύτερη φάση ορίσαμε μια σειρά από συναρτήσεις, που μας βοηθούν να κατασκευάσουμε τις δομές που αναφέραμε παραπάνω χρησιμοποιώντας τις δομές που παίρνουμε από το πρόγραμμά μας:

1. index: Ορισμός για την κατασκευή ενός κόμβου.

2. make_arc: Ορισμός για την κατασκευή μίας ακμής.

3. add_vertex: Ορισμός που χρησιμεύει για την προσθήκη ενός κόμβου στο γράφο.

4. add_arc: Ορισμός που χρησιμεύει για την προσθήκη μιας ακμής στο γράφο.

5. add_target: Ορισμός που χρησιμεύει για την προσθήκη ενός κόμβου-προορισμού στο γράφο.

6. revA: Ορισμός για την κατασκευή της αντίστροφης μιας ακμής.

7. emptyG: Ορισμός του κενού γράφου.

8. makeV: Ορισμός συνάρτησης που προσθέτει σε ένα γράφο το σύνολο των κόμβων που το αποτελούν.

9. makeA: Ορισμός συνάρτησης που προσθέτει σε ένα γράφο το σύνολο των ακμών που ξεκινούν από κάποιο συγκεκριμένο κόμβο.

10. makeAs: Ορισμός συνάρτησης που προσθέτει σε ένα γράφο το σύνολο των ακμών που περιέχονται σε αυτόν.

11. makeT: Ορισμός συνάρτησης που προσθέτει σε ένα γράφο το σύνολο των κόμβων-προορισμών που περιέχονται σε αυτόν.

12. makeG: Ορισμός συνάρτησης κατασκευής ενός γράφου.

Επίσης ορίσαμε μερικές ακόμα βοηθητικές δομές, οι οποίες μας δίνουν κάποιες ιδιότητες και χαρακτηριστικά για τις δομές που ορίσαμε στο πρώτο μέρος:

1. InV: Ορισμός συνάρτησης που ελέγχει την ύπαρξη ενός κόμβου σε μία λίστα κόμβων.

2. InA: Ορισμός συνάρτησης που ελέγχει την ύπαρξη μιας ακμής σε μία λίστα ακμών.

3. length: Ορισμός συνάρτησης υπολογισμού του μήκους μίας λίστας.

4. vertexG: Ορισμός συνάρτησης που ελέγχει την ύπαρξη ενός κόμβου σε ένα γράφο.

5. edgeG: Ορισμός συνάρτησης που ελέγχει την ύπαρξη μιας ακμής σε ένα γράφο.

6. presentG: Ορισμός συνάρτησης που ελέγχει την παρουσία μιας ακμής σε ένα γράφο.

7. targetG: Ορισμός συνάρτησης που ελέγχει το αν ένας κόμβος που περιέχεται σε ένα γράφο είναι κόμβος-προορισμός.

8. residualG: Ορισμός συνάρτησης που ελέγχει αν ένας γράφος έχει την ιδιότητα να έχει μόνο μία από δυο αντίστροφες ακμές παρούσα.

9. good: Ορισμός συνάρτησης που ελέγχει ότι ο γράφος έχει μια σειρά από ιδιότητες που σχετίζονται με παραδοχές για την είσοδο που έχουμε στο πρόγραμμά μας.

10. vertex_eq: Ορισμός που μας δίνει την πληροφορία αν δύο κόμβοι είναι ίσοι.

11. arc_eq: Ορισμός που μας δίνει την πληροφορία αν δύο ακμές είναι ίσες.

12. path: Ορίσαμε αναδρομικά την έννοια του μονοπατιού θεωρώντας ότι υπάρχει ένα μονοπάτι από έναν κόμβο σε κάποιον άλλο αν και μόνο αν υπάρχει ένα σύνολο κόμβων που περιέχεται στο γράφο και συνδέονται με ένα σύνολο ακμών που είναι παρούσες στο γράφο και οδηγούν από τον αρχικό κόμβο στον τελικό. Τόσο το σύνολο των κόμβων όσο και το σύνολο των ακμών δεν περιέχουν διπλές καταχωρήσεις (σε ένα μονοπάτι δεν έχουμε δυο φορές ούτε τον ίδιο κόμβο ούτε την ίδια ακμή).

13. edge_from: Ένας άλλος ορισμός που κατασκευάσαμε είναι μια αναδρομική σχέση που μας επιτρέπει να ελέγχουμε τους άμεσους γείτονες ενός κόμβου με τη σειρά που αυτοί εμφανίζονται στη λίστα του προγράμματός μας (στην υλοποίηση μας κάθε κόμβος αντιστοιχίζεται σε μία συνδεδεμένη λίστα που περιλαμβάνει το σύνολο των κόμβων- γειτόνων της).

14. vertex_in_list: Ορισμός που μας δίνει την πληροφορία για την ύπαρξη ενός κόμβου μέσα σε μία λίστα κόμβων.

15. arc_is_right: Ορισμός που μας δίνει την πληροφορία ότι από τις άκρες μιας ακμής η πρώτη ανήκει σε ένα σύνολο κόμβων και η δεύτερη σε κάποιο άλλο σύνολο κόμβων.

16. arc_in_list: Ορισμός που μας δίνει την πληροφορία για την ύπαρξη μίας ακμής μέσα σε μία λίστα ακμών.

17. filter_list: Ορισμός συνάρτησης που φιλτράρει μία λίστα χρησιμοποιώντας μια συγκεκριμένη ιδιότητα για τα περιεχόμενά της.

18. remove_first: Ορισμός συνάρτησης που αφαιρεί από μία λίστα το πρώτο στοιχείο το οποίο είναι ίδιο με ένα δοσμένο στοιχείο Α.

19. alist_diff: Ορισμός συνάρτησης που αφαιρεί τα στοιχεία μίας λίστας Λ2 από μία άλλη λίστα Λ1. Παρουσιάζει όμως την εξής ιδιαιτερότητα: για κάθε στοιχείο της Λ2 βρίσκουνε την πρώτη εμφάνισή του στη Λ1 και αφαιρούμε μόνο αυτή και όχι άλλες πιθανές εμφανίσεις του ίδιου στοιχείου.

Τέλος, χρησιμοποιήσαμε κάποιους βοηθητικούς ορισμούς για να κατασκευάσουμε τον ορισμό του προβλήματος min cut όπως το έχουμε περιγράψει σε προηγούμενο τμήμα της εργασίας:

1. pathG: Ορισμός ενός μονοπατιού σε ένα λογικό γράφο.

2. cutG: Ορισμός της έννοιας του cut.

3. mincutG: Ορισμός της έννοιας του min cut.

Αυτοί είναι όλοι οι ορισμοί που χρησιμοποιήσαμε στην απόδειξή μας και στους οποίους αναφερόμαστε στα επόμενα τμήματα της εργασίας όπου γίνεται εκτενέστερη περιγραφή της απόδειξης.

6.2 Σύντομη περιγραφή του συνόλου των λημμάτων

Όπως ήδη αναφέραμε στο σημείο αυτό θα παρουσιάσουμε εν συντομία το σύνολο των λημμάτων που κληθήκαμε να αποδείξουμε. Αυτά τα λήμματα χωρίζονται σε δύο κατηγορίες, όπου στην πρώτη έχουμε το σύνολο των λημμάτων που δημιουργήθηκαν αυτόματα με τη χρήση του Caduceus πάνω στον κώδικά μας, ενώ στη δεύτερη όλα τα βοηθητικά λήμματα που χρειάστηκε να γράψουμε για να αποδείξουμε το ζητούμενο.

Τα λήμματα που δημιουργήθηκαν αυτόματα από το Caduceus για τη συνάρτηση min_cut είναι τα εξής:

1. Αρχικοποίηση της αναλλοίωτης του εξωτερικού βρόχου.

2. Αρχικοποίηση της αναλλοίωτης του εσωτερικού βρόχου.

3. Απόδειξη της ισχύος του δείκτη visited[i].

4. Διατήρηση της αναλλοίωτης βρόχου του εσωτερικού βρόχου.

5. Απόδειξη της ισχύος των προαπαιτούμενων συνθηκών για την κλήση της find_path.

6. Διατήρηση της αναλλοίωτης βρόχου του εξωτερικού βρόχου.

7. Απόδειξη της μετασυνθήκης στην περίπτωση που έχουμε είσοδο στη δομή else και επιστροφή από τη συνάρτηση.

8. Απόδειξη μη προσβασιμότητας στο τελευταίο κομμάτι του κώδικα.

Από τα παραπάνω δε θα ασχοληθούμε καθόλου με τα λήμματα 1-5 και με το λήμμα 8. Αυτά τα λήμματα αναφέρονται σε ισχύ δεικτών πινάκων και σε αρχικοποιήσεις βρόχων και, συνεπώς, τα περισσότερα αποδεικνύονται αυτόματα χωρίς να χρειαστεί να γράψουμε κάτι ιδιαίτερο στο coq, αλλά και σε όσα πρέπει να γράψουμε κάτι είναι ιδιαιτέρως εύκολα στην απόδειξη. Με τα υπόλοιπα θα ασχοληθούμε εκτενέστερα στο επόμενο μέρος της εργασίας.

Αντιστοίχως, τα λήμματα που δημιουργήθηκαν αυτόματα από το Caduceus για τη συνάρτηση find_path είναι τα εξής

1. Απόδειξη της μετασυνθήκης στην περίπτωση που ο κόμβος που εξετάζουμε ανήκει στο σύνολο των κόμβων.

2. Απόδειξη της ισχύος του δείκτη visited[x].

3. Αρχικοποίηση της αναλλοίωτης του βρόχου.

4. Απόδειξη της ισχύος του δείκτη visited[e[k].to].

5. Απόδειξη της ισχύος των προαπαιτούμενων συνθηκών για την αναδρομική κλήση της find_path.

6. Απόδειξη της μετασυνθήκης στην περίπτωση που έχουμε είσοδο στη δομή if και επιστροφή από τη συνάρτηση.

7. Διατήρηση της αναλλοίωτης του βρόχου στην περίπτωση που η αναδρομική κλήση της find_path επιστρέψει false.

8. Διατήρηση της αναλλοίωτης του βρόχου στην περίπτωση που ο κόμβος e[k].to είχε ήδη εξεταστεί προηγουμένως.

9. Διατήρηση της αναλλοίωτης του βρόχου στην περίπτωση που η ακμή από τον τρέχοντα κόμβο προς κάποιο πιθανό επόμενο δεν είναι παρούσα.

10. Απόδειξη της ισχύος της μετασυνθήκης στην περίπτωση όπου δεν βρεθεί μονοπάτι και η συνάρτηση επιστρέψει false.

Από τα παραπάνω λήμματα και πάλι δε θα ασχοληθούμε με τα λήμματα 1-5, τα οποία αναφέρονται σε ισχύ δεικτών πινάκων, στην αρχικοποίηση του βρόχου, καθώς και στην ισχύ της μετασυνθήκης στην περίπτωση επιστροφής όταν εντοπίσουμε ότι ο επισκεπτόμενος κόμβος είναι κόμβος-προορισμός. Η απόδειξή των λημμάτων αυτών είναι ιδιαιτέρως εύκολη σε σχέση με τα υπόλοιπα λήμματα. Με τα υπόλοιπα λήμματα θα ασχοληθούμε στο επόμενο μέρος της εργασίας.

Τέλος, θα αναφέρουμε επιγραμματικά και τα υπόλοιπα λήμματα που έπρεπε να προσθέσουμε για να ολοκληρώσουμε την απόδειξή μας.

1. vertex_dec_eq: Λήμμα που μας δίνει τη δυνατότητα να εξετάσουμε το κατά πόσο 2 κόμβοι είναι ίσοι ή όχι.

2. arc_dec_eq: Λήμμα που μας δίνει τη δυνατότητα να εξετάσουμε το κατά πόσο 2 ακμές είναι ίσες ή όχι.

3. arc_eq_sane: Λήμμα που αποδεικνύει ότι αν και μόνο αν μία ακμή είναι ίση με μία άλλη, τότε η arc_eq επιστρέφει true.

4. arc_eq_sane_inv: Λήμμα που αποδεικνύει ότι αν και μόνο αν μία ακμή είναι διαφορετική με μία άλλη, τότε η arc_eq επιστρέφει false.

5. path_vl_al: Λήμμα που αποδεικνύει ότι για οποιαδήποτε ακμή περιλαμβάνεται σε ένα μονοπάτι, ο κόμβος-πηγή της ακμής επίσης υπάρχει μέσα στο μονοπάτι αυτό.

6. path_al_vl: Λήμμα που αποδεικνύει ότι για οποιονδήποτε κόμβο υπάρχει σε ένα μονοπάτι, υπάρχει και μια αντίστοιχη ακμή από τον κόμβο αυτό προς κάποιον άλλο κόμβο, η οποία επίσης περιλαμβάνεται στο μονοπάτι.

7. path_no_dup: Λήμμα που αποδεικνύει ότι ένα μονοπάτι δεν μπορεί να περιέχει δύο φορές την ίδια ακμή.

8. path_to_path: Λήμμα που αποδεικνύει ότι εφόσον έχουμε ένα μονοπάτι από κάποιο κόμβο Α σε κάποιο κόμβο Β, τότε έχουμε μονοπάτι προς τον κόμβο Β από οποιονδήποτε κόμβο βρίσκεται μέσα στο μονοπάτι αυτό.

9. cons_length: Λήμμα που αποδεικνύει ότι το μήκος μιας λίστας στην όποια έχουμε τοποθετήσει στην αρχή κάποιο στοιχείο είναι ίσο με το μήκος της χωρίς το στοιχείο αυτό αυξημένο κατά 1.

10. app_length: Λήμμα που αποδεικνύει ότι το μήκος της συνένωσης δύο λιστών είναι ίσο με το άθροισμα των μηκών των δύο λιστών ξεχωριστά.

11. numberToVertex: Λήμμα που αντιστοιχίζει αμφημονοσήμαντα ένα φυσικό αριθμό με έναν κόμβο.

12. present_info: Λήμμα που αποδεικνύει ότι αν μία ακμή είναι παρούσα στο γράφο, θα είναι σίγουρα ακμή του γράφου.

13. filter_no_dup: Λήμμα που αποδεικνύει ότι αν έχουμε μια λίστα που δεν έχει διπλές καταχωρήσεις κι εφαρμόσουμε μία συνάρτηση φιλτραρίσματος πάνω σε αυτήν (filter), η καινούργια λίστα επίσης δε θα περιέχει διπλές καταχωρήσεις.

14. map_no_dup: Λήμμα που αποδεικνύει ότι στην περίπτωση που έχουμε μία συνάρτηση ένα προς ένα (1-1) και επί και την εφαρμόσουμε σε μία λίστα που δεν περιέχει διπλές καταχωρήσεις, τότε και η λίστα που θα πάρουμε ως αποτέλεσμα δεν περιέχει διπλές καταχωρήσεις.

15. vertex_eq_eq: Λήμμα που αποδεικνύει ότι όταν έχουμε δύο ίδιους κόμβους η vertex_eq επιστρέφει την τιμή true.

16. vertex_eq_true: Λήμμα που αποδεικνύει ότι όταν η vertex_eq επιστρέφει την τιμή true, τοτέ οι δύο κόμβοι που συγκρίναμε είναι ίδιοι.

17. filter_list_in: Λήμμα που αποδεικνύει ότι εφόσον ένα αντικείμενο ανήκει σε μία λίστα και ικανοποιεί κάποια συγκεκριμένη ιδιότητα, τότε ανήκει και στην αντίστοιχη φιλτραρισμένη λίστα.

18. filter_list_not_in: Λήμμα που αποδεικνύει ότι εφόσον ένα αντικείμενο δεν ικανοποιεί κάποια συγκεκριμένη ιδιότητα, τότε δεν μπορεί να ανήκει στην αντίστοιχη φιλτραρισμένη λίστα.

19. filter_list_rev: Λήμμα που αποδεικνύει ότι το μήκος μιας φιλτραρισμένης λίστας είναι ίδιο με αυτό της ίδιας φιλτραρισμένης λίστας, όπου έχουμε αντιστρέψει κάθε στοιχείο της (η πράξη της αντιστροφής αποτελεί μια πράξη ένα προς ένα (1-1) και επί).

20. in_remove_first: Λήμμα που αποδεικνύει ότι αν ένα στοιχείο Α ανήκει σε μία λίστα και είναι διαφορετικό από ένα στοιχείο Β, τότε το στοιχείο Α ανήκει και στη λίστα αυτή αν αφαιρέσουμε το πρώτο από τα στοιχεία Β που θα βρούμε στη λίστα.

21. in_alist_diff: Λήμμα που αποδεικνύει ότι αν ένα στοιχείο βρίσκεται σε μία λίστα Λ1, αλλά δε βρίσκεται σε μία λίστα Λ2, τότε βρίσκεται και στη διαφορά των δύο λιστών όπως την έχουμε ορίσει παραπάνω.

22. extract_in: Λήμμα που αποδεικνύει ότι αν ένα αντικείμενο Α βρίσκεται σε μία λίστα, τότε υπάρχει μία άλλη λίστα, στην οποία προσθέτοντας το αντικείμενο Α παίρνουμε μία αναδιάταξη της αρχικής.

23. remove_pres_permute: Λήμμα που αποδεικνύει ότι εφόσον μία λίστα είναι αναδιάταξη μίας άλλης, τότε αν αφαιρέσουμε και από τις δύο μία φορά το ίδιο στοιχείο, τότε οι παραγόμενες λίστες θα είναι και πάλι η μία αναδιάταξη της άλλης.

24. alist_diff_permute: Λήμμα που αποδεικνύει ότι εφόσον μία λίστα είναι αναδιάταξη μιας άλλης, τότε αν αφαιρέσουμε και από τις δύο την ίδια λίστα με την έννοια της αφαίρεσης που έχουμε ορίσει παραπάνω, τότε οι παραγόμενες λίστες θα είναι και πάλι η μία αναδιάταξη της άλλης.

25. alist_diff_not_in: Λήμμα που αποδεικνύει ότι αν το πρώτο στοιχείο μίας λίστας Λ1 δεν υπάρχει μέσα σε μία λίστα Λ2, τότε η διαφορά των δύο λιστών είναι ίση με τη διαφορά της λίστας Λ1, από την οποία αφαιρέσαμε το πρώτο στοιχείο, με τη λίστα Λ2, εφόσον στη διαφορά τους προσθέσουμε στην αρχή το στοιχείο που αφαιρέσαμε από τη Λ1.

26. alist_diff_permute_one: Λήμμα που αποδεικνύει ότι εφόσον έχουμε ένα στοιχείο Α το οποίο ανήκει σε μία λίστα Λ1 και δεν ανήκει σε μία λίστα Λ2, τότε η διαφορά των δύο λιστών έχει τα ίδια στοιχεία με τη λίστα που προκύπτει από τη διαφορά των Λ1 και Λ2, εφόσον από την Λ1 έχουμε αφαιρέσει το στοιχείο Α και το προσθέσουμε στην αρχή της λίστας της διαφοράς τους.

27. alist_diff_permute: Λήμμα που αποδεικνύει ότι αν κάθε στοιχείο μιας λίστας Λ2 είναι στοιχείο και μίας άλλης λίστας Λ1 και η Λ2 δεν περιέχει διπλά στοιχεία, τότε η λίστα Λ1 έχει τα ίδια ακριβώς στοιχεία με τη λίστα που προκύπτει αν αφαιρέσουμε από τη λίστα Λ1 τη Λ2 και, στη συνέχεια, προσθέσουμε και πάλι τη Λ2.

28. existsb_not_exists: Λήμμα που αποδεικνύει ότι αν και μόνο αν ένα στοιχείο μίας λίστας δεν ικανοποιεί μία συγκεκριμένη ιδιότητα, τότε η συνάρτηση existsb επιστρέφει false.

29. vertex_eq_not_eq: Λήμμα που αποδεικνύει ότι όταν έχουμε δύο κόμβους που έχουν έστω και μία διαφορετική ιδιότητα, τότε η vertex_eq επιστρέφει την τιμή false.

30. all_lists_can_become_bigger: Λήμμα που αποδεικνύει ότι για κάθε λίστα μήκους n (όπου n φυσικός αριθμός) μπορούμε να έχουμε μια λίστα μεγαλύτερου μήκους m (όπου m φυσικός αριθμός).

31. Zpos_of_nat: Λήμμα που αποδεικνύει ότι κάθε θετικός ακέραιος μπορεί να αντιστοιχηθεί σε κάποιο φυσικό αριθμό.

32. Z_of_nat_non_neg: Λήμμα που αποδεικνύει ότι κάθε μη αρνητικός ακέραιος μπορεί να αντιστοιχηθεί σε κάποιο φυσικό αριθμό.

33. all_lists_can_become_bigger_z: Λήμμα που αποδεικνύει ότι για κάθε λίστα μήκους n (όπου n ακέραιος) μπορούμε να έχουμε μία λίστα μεγαλύτερου μήκους m (όπου m ακέραιος).

34. all_bigger_cuts: Λήμμα που αποδεικνύει ότι αν ένας γράφος έχει ένα cut κάποιου μεγέθους n, τότε έχει cut για κάθε αριθμό μεγαλύτερο του n.

35. no_smaller_cut: Λήμμα που αποδεικνύει ότι αν ένας γράφος δεν έχει cut για έναν ακέραιο n, τότε δεν έχει cut για κανένα μικρότερο ή ίσο με αυτό ακέραιο.

36. simple_path_start_s: Λήμμα που αποδεικνύει ότι αν χωρίσουμε τους κόμβους ενός γράφου σε δύο σύνολα A και B και έχουμε ένα μονοπάτι το οποίο καταλήγει σε κάποιον κόμβο του B, τότε το μονοπάτι αυτό περιέχει ίσο αριθμό ακμών από το A στο B και από το B στο A αν ο αρχικός κόμβος του μονοπατιού βρίσκεται επίσης στο σύνολο B. Αντιθέτως, αν βρίσκεται στο σύνολο A, το πλήθος των ακμών από το A στο B είναι κατά ένα μεγαλύτερο σε σχέση με το πλήθος των ακμών από το B στο A.

37. simple_path_s_to_t: Λήμμα-εφαρμογή του προηγούμενου, που αποδεικνύει ότι αν χωρίσουμε τους κόμβους ενός γράφου σε δύο σύνολα A και B και έχουμε ένα μονοπάτι το οποίο ξεκινάει από κάποιο κόμβο του συνόλου A και καταλήγει σε κάποιον κόμβο του B, τότε το μονοπάτι αυτό περιέχει μία ακμή περισσότερη από το σύνολο A προς το σύνολο B σε σχέση με το πλήθος των ακμών από το B στο A.

38. permutation_length: Λήμμα που αποδεικνύει ότι εφόσον δύο λίστες περιέχουν τα ίδια στοιχεία, έχουν και το ίδιο μήκος.

39. cut_succ: Λήμμα άμεσα βοηθητικό στην απόδειξη της ορθότητας της βασικής αναλλοίωτης βρόχου του αλγορίθμου των Ford-Fulkerson που θα παρουσιαστεί εκτενώς στο επόμενο τμήμα της εργασίας.

40. mincut_succ: Λήμμα που αποδεικνύει την ορθότητα της βασικής αναλλοίωτης βρόχου του αλγορίθμου των Ford-Fulkerson και θα παρουσιαστεί εκτενώς στο επόμενο τμήμα της εργασίας.

41. path_preserved: Λήμμα που αποδεικνύει ότι εφόσον οι ακμές που είναι παρούσες σε ένα γράφο διατηρούνται αναλλοίωτες, η ύπαρξη μονοπατιού στο γράφο αυτό διατηρείται επίσης.

Με αυτό τον τρόπο ολοκληρώσαμε το σύνολο των λημμάτων που χρειαστήκαμε σε αυτή την απόδειξη. Στο επόμενο κομμάτι της εργασίας αναλύουμε εκτενέστερα τα σημαντικότερα από τα παραπάνω λήμματα, που είτε κρίθηκαν ιδιαιτέρως δύσκολα, είτε είχαν μεγάλη σημασία για την απόδειξή μας.

6.3 Παρουσίαση βασικότερων λημμάτων

Όπως έχουμε ήδη αναφέρει, σε αυτό το κομμάτι της εργασίας θα ασχοληθούμε με την περιγραφή κι ανάλυση των βασικότερων λημμάτων που κληθήκαμε να αποδείξουμε για να κατασκευάσουμε την απόδειξη ορθότητας του min cut. Θα βασιστούμε στα λήμματα που πήραμε ως έξοδο από το εργαλείο Caduceus και με βάση αυτά θα αναφέρουμε όσα λήμματα παρουσιάζουν κάποια ιδιαιτερότητα ως προς την επίλυσή τους ή όσα θεωρούμε ότι είναι βασικά για την κατανόηση της απόδειξης. Ακολούθως, θα αναφερθούμε αρχικά στα λήμματα της min_cut και έπειτα σε αυτά της find_path.

6.3.1 Βασικότερα λήμματα της min_cut

Όπως ήδη αναφέραμε από τα λήμματα της min_cut θα ασχοληθούμε μόνο με τα λήμματα 6 και 7. Το πρώτο από αυτά αναφέρεται στην αναλλοίωτη του βρόχου που θα χρησιμοποιήσουμε ή οποία αναφέρει τα εξής:

1. Ο γράφος παραμένει έγκυρος όσον αφορά το γεγονός ότι για κάθε ακμή που είναι παρούσα στο γράφο η αντίστροφή της δεν είναι παρούσα.

2. Το min_cut του αρχικού γράφου είναι ίσο με το min_cut του γράφου που έχουμε τώρα αυξημένο κατά το πλήθος των επαναλήψεων που έχουμε κάνει μέχρι τώρα.

Η απόδειξη του πρώτου κομματιού του λήμματος είναι απλή, καθώς η ισχύς του προέρχεται άμεσα από τα όσα μας επιστρέφει η κλήση της find_path. Όσον αφορά το δεύτερο κομμάτι του λήμματος

είναι εύκολο να δείξουμε ότι για να ισχύει αυτό θα πρέπει σε κάθε επανάληψη να ισχύει ότι το min cut του τρέχοντος γράφου έχει μειωθεί κατά ένα από την προηγούμενη επανάληψη. Η απόδειξη αυτού του τμήματος είναι πιθανότατα και η πιο δύσκολη απόδειξη που κληθήκαμε να υλοποιήσουμε.

Για να δείξουμε το ζητούμενο, θα πρέπει να χωρίσουμε αυτό το τμήμα της απόδειξής μας σε δύο επιμέρους κομμάτια. Στο πρώτο, θα δείξουμε ότι αφαιρώντας τις ακμές ενός μονοπατιού μεταξύ του κόμβου-πηγής και ενός κόμβου προορισμού και προσθέτοντας τις αντίθετές τους σε ένα γράφο πετυχαίνουμε να μειώσουμε το min cut του κατά 1. Στο δεύτερο, θα δείξουμε ότι το πρόγραμμά μας υλοποιεί ακριβώς αυτή τη διαδικασία.

Η απόδειξη του πρώτου τμήματος γίνεται ουσιαστικά στο λήμμα simple_path_from_s_to_t. Για την εξήγηση του παρόντος λήμματος θα πρέπει να θυμίσουμε εν συντομία τον ορισμό που έχουμε δώσει στο min cut σε προηγούμενα κεφάλαια. Το min cut ενός γράφου έχει οριστεί ως εξής: έχουμε ένα γράφο και χωρίζουμε το σύνολο των κόμβων του σε δύο υποσύνολα, όπου το πρώτο περιέχει τουλάχιστον τον κόμβο-πηγή, ενώ το δεύτερο τουλάχιστον όλους τους κόμβους-προορισμούς. Το πλήθος των ακμών που συνδέει το πρώτο σύνολο κόμβων με το δεύτερο είναι ίσο με ένα cut του γράφου. Το ελάχιστο τέτοιο cut είναι το min cut του γράφου αυτού. Συνεπώς, για να αποδείξουμε ότι το min cut του καινούργιου γράφου είναι κατά ένα μικρότερο του παλιού θα πρέπει να αποδείξουμε ότι το πλήθος των ακμών από οποιοδήποτε σύνολο κόμβων Λ1 που περιέχει τον κόμβο-πηγή προς οποιοδήποτε σύνολο κόμβων Λ2 που περιέχει όλους τους κόμβους-προορισμούς μειώνεται κατά 1 στον καινούργιο γράφο. Όμως, όπως αποδεικνύουμε και παρακάτω, η μόνη διαφορά μεταξύ των δύο γράφων είναι ότι στον καινούργιο γράφο έχουμε αφαιρέσει τις ακμές που περιέχονται στο μονοπάτι που μας επέστρεψε η find_path κι έχουμε προσθέσει τις αντίστροφές τους. Επομένως, τελικά, αυτό που θέλουμε να δείξουμε είναι ότι σε ένα μονοπάτι από τον κόμβο-πηγή προς οποιονδήποτε κόμβο-προορισμό περιέχονται κατά μία περισσότερες ακμές που συνδέουν κάποιο κόμβο του Λ1 με κάποιο κόμβο του Λ2 από ότι ακμές που συνδέουν κάποιο κόμβο του Λ2 με κάποιο κόμβο του Λ1. Αυτό ακριβώς αποδεικνύεται στο λήμμα που αναφέραμε.

Η απόδειξη του λήμματος αυτού προκύπτει σαν άμεση εφαρμογή ενός γενικότερου λήμματος, το οποίο ονομάζεται simple_path_start_s. Σε αυτό το λήμμα αποδεικνύουμε ότι αν έχουμε ένα μονοπάτι το οποίο καταλήγει σε έναν κόμβο του Λ2, τότε έχουμε δύο περιπτώσεις. Εφόσον το μονοπάτι αυτό ξεκινάει από κάποιο κόμβο του Λ2, το πλήθος των ακμών που περιέχονται στο μονοπάτι και ενώνουν κάποιο κόμβο του Λ1 με κάποιο κόμβο του Λ2 είναι ίσο με αυτών που ενώνουν κάποιο κόμβο του Λ2 με κάποιον του Λ1. Αντιθέτως, αν το μονοπάτι ξεκινάει από κάποιο κόμβο του Λ1, τότε το πλήθος των ακμών από το Λ1 στο Λ2 είναι κατά 1 μεγαλύτερο. Αυτό το αποδεικνύουμε με επαγωγή πάνω στο μήκος του μονοπατιού. Έτσι έχουμε ότι αν το μήκος του μονοπατιού είναι μηδενικό, τότε έχουμε ένα μονοπάτι με αφετηρία και προορισμό τον ίδιο κόμβο, που επειδή είναι ο κόμβος-προορισμός, ξέρουμε από την υπόθεση ότι ανήκει στο Λ2. Το μονοπάτι αυτό δεν έχει καμία ακμή και, προφανώς, το πλήθος των ακμών από το Λ1 στο Λ2 είναι ίσο με αυτό από το Λ2 στο Λ1 και ίσο με μηδέν. Ακολούθως, χρησιμοποιώντας την επαγωγική υπόθεση, θα πρέπει να αποδείξουμε ότι εφόσον για ένα μονοπάτι ισχύει το παραπάνω, τότε θα ισχύει και για ένα μονοπάτι το οποίο θα έχει μία επιπλέον ακμή. Για να αποδείξουμε αυτό έχουμε 4 υποπεριπτώσεις:

1. Το αρχικό μονοπάτι να ξεκινάει από κάποιο κόμβο του Λ1 και ο νέος κόμβος που προστίθεται στο μονοπάτι να βρίσκεται επίσης στο Λ1. Σε αυτή την περίπτωση από την επαγωγική υπόθεση έχουμε ότι το πλήθος των ακμών από το Λ1 στο Λ2 είναι αρχικά κατά 1 μεγαλύτερο. Εφόσον ο επιπλέον κόμβος βρίσκεται επίσης στο Λ1, δεν προστίθεται στο μονοπάτι ούτε κάποια ακμή από το Λ1 στο Λ2, αλλά ούτε κάποια ακμή από το Λ2 στο Λ1. Συνεπώς, διατηρείται η διαφορά μεταξύ των δύο συνόλων και αποδεικνύεται το ζητούμενο, εφόσον ο αρχικός κόμβος βρίσκεται και πάλι στο Λ1.

2. Το αρχικό μονοπάτι να ξεκινάει από κάποιο κόμβο του Λ1 και ο νέος κόμβος που προστίθεται στο μονοπάτι να βρίσκεται στο Λ2. Σε αυτή την περίπτωση από την επαγωγική υπόθεση έχουμε ότι το πλήθος των ακμών από το Λ1 στο Λ2 είναι αρχικά κατά 1 μεγαλύτερο. Εφόσον ο επιπλέον κόμβος βρίσκεται στο Λ2, δεν προστίθεται στο μονοπάτι κάποια ακμή από το Λ1 στο Λ2, αλλά

προστίθεται ακριβώς μία ακμή από το Λ2 στο Λ1. Οπότε εδώ τα δύο σύνολα αποκτούν πλέον το ίδιο πλήθος ακμών και αποδεικνύεται το ζητούμενο, καθώς ο αρχικός κόμβος βρίσκεται πλέον στο Λ2.

3. Το αρχικό μονοπάτι να ξεκινάει από κάποιο κόμβο του Λ2 και ο νέος κόμβος που προστίθεται στο μονοπάτι να βρίσκεται στο Λ1. Σε αυτή την περίπτωση από την επαγωγική υπόθεση έχουμε ότι το πλήθος των ακμών από το Λ1 στο Λ2 είναι αρχικά ίσο με το πλήθος των κόμβων από το Λ2 στο Λ1. Εφόσον ο επιπλέον κόμβος βρίσκεται στο Λ1, δεν προστίθεται στο μονοπάτι κάποια ακμή από το Λ2 στο Λ1, αλλά προστίθεται ακριβώς μία ακμή από το Λ1 στο Λ2. Οπότε εδώ τα δύο σύνολα αποκτούν πλέον διαφορά ίση με 1 υπέρ του συνόλου των ακμών από το Λ1 στο Λ2 και αποδεικνύεται το ζητούμενο, διότι ο αρχικός κόμβος είναι πλέον στο Λ1.

4. Το αρχικό μονοπάτι να ξεκινάει από κάποιο κόμβο του Λ2 και ο νέος κόμβος που προστίθεται στο μονοπάτι να βρίσκεται επίσης στο Λ2. Σε αυτή την περίπτωση από την επαγωγική υπόθεση έχουμε ότι το πλήθος των ακμών από το Λ1 στο Λ2 είναι αρχικά ίσο με το πλήθος των κόμβων από το Λ2 στο Λ1. Εφόσον ο επιπλέον κόμβος βρίσκεται επίσης στο Λ2, δεν προστίθεται στο μονοπάτι ούτε κάποια ακμή από το Λ1 στο Λ2, αλλά ούτε κάποια ακμή από το Λ2 στο Λ1. Συνεπώς, διατηρείται η ισότητα μεταξύ των δύο συνόλων και αποδεικνύεται το ζητούμενο, εφόσον ο αρχικός κόμβος βρίσκεται και πάλι στο Λ2.

Για την απόδειξη του πρώτου μέρους αυτού του λήμματος χρειαστήκαμε επιπλέον κάποια μικρότερα λήμματα, τα οποία αναφέρονταν κυρίως σε ιδιότητες που προκύπτουν από την εφαρμογή της συνάρτησης φιλτραρίσματος στη λίστα των ακμών του μονοπατιού.

Για να αποδείξουμε το δεύτερο τμήμα εισάγουμε ένα καινούργιο λήμμα το οποίο ονομάζεται mincut_succ. Και σε αυτή την περίπτωση επιλέγουμε να αποδείξουμε κάτι γενικότερο, του οποίου εφαρμογή είναι το ζητούμενο. Έτσι αποδεικνύουμε ότι για οποιοδήποτε cut του αρχικού γράφου, στον καινούργιο γράφο έχουμε ένα cut κατά ένα μικρότερο. Αυτό προκύπτει άμεσα από το πρώτο τμήμα εφόσον αποδείξουμε ότι το πρόγραμμά μας προσθέτει και αφαιρεί τις κατάλληλες ακμές από το γράφο μας. Την πληροφορία αυτή την παίρνουμε από την επιστροφή της find_path, στην οποία περιέχεται η πληροφορία ότι οι γράφοι μας είναι ίδιοι με εξαίρεση την αφαίρεση των κόμβων του μονοπατιού και την πρόσθεση των αντίστροφών τους.

Ενώ όμως, σύμφωνα με τα παραπάνω, θα έπρεπε στη συνέχεια η απόδειξη να είναι προφανής, εδώ παρατηρούμε μία αδυναμία του Coq, η οποία οφείλεται κυρίως στην έλλειψη αρκετών δομών για τη διαχείριση των λιστών. Πιο συγκεκριμένα, δεν περιλαμβάνει κάποιον ορισμό για την αφαίρεση μίας λίστας από μία άλλη και, προφανώς, ούτε συνοδευτικά λήμματα που να υποστηρίζουν την πράξη αυτή. Αυτό μας ανάγκασε να δώσουμε μια σειρά από ορισμούς για να μπορέσουμε τελικά μέσω την alist_diff να ορίσουμε τη διαφορά αυτή. Επίσης, ακολούθως, αποδείξαμε μια σειρά από λήμματα, των οποίων η απόδειξη δεν ήταν πάντα προφανής, ώστε να μπορέσουμε να χρησιμοποιήσουμε καταλλήλως τη διαφορά των λιστών στην απόδειξή μας. Πιθανότατα, θα μπορούσαμε να αποφύγουμε τους παραπάνω ορισμούς χρησιμοποιώντας αντί για τη δομή της λίστας (list) τη δομή του συνόλου (Set) που επίσης περιλαμβάνεται στο Coq. Όμως σε αυτή την περίπτωση συναντούσαμε μια σειρά από άλλα προβλήματα σε άλλα στάδια της απόδειξης και έτσι προτιμήσαμε τη δομή της λίστας.

Μετά από αυτά καταφέραμε να ορίσουμε ένα cut του καινούργιου γράφου συναρτήσει ενός cut του παλιού μας γράφου. Αποδείξαμε ότι το cut του καινούργιου θα περιέχει όλες της ακμές που περιείχε το cut του παλιού εκτός από αυτές που περιλαμβάνονται στο μονοπάτι. Επίσης θα περιέχει το σύνολο των αντίστροφων των ακμών που περιλαμβάνονται στο μονοπάτι και που συνδέουν το ένα σύνολο κόμβων (αυτό που περιέχει τους κόμβους-προορισμούς) με το άλλο σύνολο κόμβων (αυτό που περιέχει τον κόμβο-πηγή).

Τέλος, ένα άλλο πρόβλημα που αντιμετωπίσαμε εξαιτίας του Coq στο σημείο αυτό ήταν η απόδειξη του λήμματος που αναφέρει ότι αν ένας γράφος έχει ένα cut μεγέθους n, τότε θα έχει ένα cut και για οποιοδήποτε m μεγαλύτερο του n. Αυτό είναι προφανές για την ανθρώπινη λογική, καθώς, όπως γνωρίζουμε, ο ορισμός του cut μας επιτρέπει σε περίπτωση που έχουμε ένα cut μεγέθους n για ένα γράφο να επιλέξουμε τυχαία μία επιπλέον ακμή του γράφου και να ισχυριστούμε ότι ο γράφος αυτός

έχει cut μεγέθους n+1. Αυτή η διαδικασία μπορεί να συνεχιστεί μέχρι περάτωσης του συνόλου των ακμών του γράφου. Οπότε το ζητούμενο μπορεί να θεωρηθεί προφανές. Στην περίπτωση όμως του Coq, για να αποδείξουμε το ζητούμενο κληθήκαμε να αποδείξουμε ότι για κάθε φυσικό αριθμό υπάρχει κάποιος άλλος φυσικός που μπορεί να είναι μεγαλύτερος (δεδομένο για την ανθρώπινη λογική) και ακολούθως ότι για κάθε λίστα υπάρχει μια άλλη λίστα που έχει μήκος μεγαλύτερο της πρώτης (και πάλι αυτονόητο). Με τον τρόπο αυτό επιτύχαμε να αποδείξουμε και αυτό που αρχικά θέλαμε.

Χρησιμοποιώντας όλα όσα αναφέραμε παραπάνω καταφέραμε τελικά να αποδείξουμε την ισχύ του λήμματος 6 όσον αφορά την αναλλοίωτη του βρόχου της min_cut.

Όσον αφορά το λήμμα 7, πρόκειται για την απόδειξη ότι η συνάρτησή μας επιστρέφει τελικά το ζητούμενο αποτέλεσμα, δηλαδή το min cut του αρχικού γράφου. Επειδή όμως η συνάρτηση find_path μας επιστρέφει ότι στον τρέχοντα γράφο δεν έχουμε κάποιο μονοπάτι από τον κόμβο-πηγή προς οποιονδήποτε κόμβο-προορισμό που να χρησιμοποιεί μόνο κόμβους που δεν τους έχουμε επισκεφθεί αρχικά (δηλαδή το σύνολο των κόμβων για την πρώτη επανάληψη της find_path), με βάση τον ορισμό του min cut, καταλαβαίνουμε ότι για το γράφο αυτό το min cut είναι μηδέν. Συνεπώς, σύμφωνα με το αποτέλεσμα του προηγούμενου λήμματος όσον αφορά την αναλλοίωτη του βρόχου, συμπεραίνουμε ότι το πλήθος των επαναλήψεων που έχουν εκτελεστεί είναι ίσο με το min cut του αρχικού γράφου και, επομένως, το επιστρεφόμενο αποτέλεσμα είναι και το ζητούμενο.

6.3.2 Βασικότερα λήμματα της find_path

Όπως αναφέραμε και προηγουμένως, στην περίπτωση της find_path θα ασχοληθούμε μόνο με τα λήμματα 6-10. Πρόκειται για τα λήμματα που αναφέρονται στη διατήρηση της αναλλοίωτης του βρόχου και στην επιστροφή της συνάρτησης όταν εισέρχεται στο if και επιστρέφει true, αλλά και όταν τελειώνει ο βρόχος και επιστρέφει false.

Το λήμμα 6 αναφέρεται στην περίπτωση όπου έχουμε είσοδο στη δομή if του βρόχου και επιστροφή της συνάρτησης με αποτέλεσμα true. Σε αυτή την περίπτωση καλούμαστε να αποδείξουμε μια σειρά από ιδιότητες που προκύπτουν κατά κύριο λόγο από την αναλλοίωτη του βρόχου που ίσχυε μέχρι την προηγούμενη επανάληψη του βρόχου και από την επιστροφή της αναδρομικής κλήσης της find_path.

Αρχικά, το να αποδείξουμε ότι ο γράφος παραμένει έγκυρος όσον αφορά την παρουσία ακριβώς μίας από δύο αντίστροφες ακμές θα έπρεπε να προκύπτει άμεσα από την επιστροφή της αναδρομικής κλήσης και από τα αποτελέσματα του κώδικα που μεσολαβεί από την κλήση μέχρι τν επιστροφή της συνάρτησης. Όμως σε αυτό το σημείο παρατηρούμε την αδυναμία του Coq να διαχειριστεί με ευκολία τις αλλαγές που επέρχονται σε μια μνήμη. Θα πρέπει εδώ να παρατηρήσουμε ότι το Coq ορίζει μία μνήμη για κάθε μέλος ενός struct όταν έχουμε ένα πίνακα από τέτοια struct. Έτσι στην περίπτωσή μας έχει μία ξεχωριστή μνήμη που αναφέρεται στην πληροφορία για την παρουσία μίας ακμής στο γράφο. Σε αυτή τη μνήμη για να δείξουμε ότι ο γράφος παραμένει έγκυρος μετά τις ενημερώσεις που γίνονται σε αυτόν, κληθήκαμε να αποδείξουμε ότι κάθε στοιχείο διάφορο αυτών των δύο που άλλαξαν παρέμεινε αναλλοίωτο. Κι ενώ για την ανθρώπινη λογική αυτό είναι προφανές βάση του κώδικα, για να αποδειχθεί με τη βοήθεια του Coq χρειάστηκαν αρκετές γραμμές κώδικα.

Η απόδειξη της διατήρησης του αναλλοίωτου της πληροφορίας για την επίσκεψη των κόμβων όπως αυτή εκφράζεται στη συνθήκη επιστροφής, μπορούμε να πούμε ότι ήταν σχετικά εύκολη συνδυάζοντας πληροφορίες τόσο από την αναλλοίωτη του βρόχου όσο και από την επιστροφή της αναδρομική κλήσης.

Στην απόδειξη της ύπαρξης του μονοπατιού από τον κόμβο που εξετάζουμε προς κάποιο κόμβο-προορισμό, οι δυσκολίες που αντιμετωπίσαμε οφείλονταν και πάλι στην ύπαρξη διαφορετικών και ανανεωμένων μνημών σε μεγάλο βαθμό. Αυτός είναι και ο λόγος που αναγκαστήκαμε να τοποθετήσουμε τόσες πολλές συνθήκες όσον αφορά τις ακμές που είναι παρούσες στο γράφο και τους κόμβους του γράφου που έχουμε επισκεφθεί ήδη ή που δεν έχουμε επισκεφθεί ακόμα. Αν εξαιρέσουμε όμως αυτή τη δυσκολία, η απόδειξη αυτή στηρίχθηκε στα όσα μας επιστρέφονται από την αναδρομική κλήση και παρόλο που είναι αρκετά μεγάλη σε μέγεθος κώδικα Coq, δεν παρουσιάζει κάποια ιδιαίτερη δυσκολία.

Ακολούθως, θα ασχοληθούμε με την απόδειξη του λήμματος 7, το οποίο αναφέρεται στην περίπτωση που δεν έχουμε είσοδο στη δομή if εξαιτίας της επιστροφής της αναδρομικής κλήσης (έχουμε επιστροφή false). Οι αποδείξεις που ζητάμε και σε αυτή την περίπτωση προέρχονται είτε άμεσα είτε με κάποια επεξεργασία από την αναλλοίωτη του βρόχου και από τα όσα μας επιστρέφονται από την αναδρομική κλήση. Η μονή ιδιαιτερότητα που θα πρέπει να παρατηρήσουμε είναι ότι χρειάστηκε να ξεχωρίσουμε όλες τις επαναλήψεις του βρόχου από την τελευταία. Στην πρώτη περίπτωση η απόδειξη των θεωρημάτων προέρχεται σχεδόν αποκλειστικά από την προηγούμενη αναλλοίωτη του βρόχου. Αντιθέτως, όπως είναι και λογικό, στη δεύτερη περίπτωση οι αποδείξεις προέρχονται από την αναδρομική κλήση. Με αυτό τον τρόπο αποδείξαμε την εγκυρότητα του γράφου όσον αφορά την παρουσία μίας από τις δύο αντίστροφες ακμές, τη διατήρηση της πληροφορίας όσον αφορά το αν έχουμε επισκεφθεί έναν κόμβο ή όχι, τη διατήρηση της πληροφορίας σχετικά με την παρουσία μίας ακμής στο γράφο, το γεγονός ότι δεν έχουμε επισκεφθεί κανέναν από τους κόμβους- προορισμούς και ότι έχουμε επισκεφθεί τόσο τον τρέχοντα κόμβο όσο και τον άμεσα γειτονικό του που εξετάσαμε μόλις. Τέλος, αποδείξαμε και ότι μέχρι την τρέχουσα επανάληψη έχουμε επισκεφθεί όλους τους κόμβους που ήταν άμεσοι γείτονες του εξεταζόμενου και οι αντίστοιχες ακμές ήταν παρούσες στο γράφο και δεν βρήκαμε μονοπάτι από κάποιον από αυτούς τους κόμβους προς κάποιο κόμβο-προορισμό.

Όσον αφορά τα λήμματα 8 και 9, πρόκειται για τις περιπτώσεις που δεν έχουμε είσοδο στο if λόγω του ότι έχουμε ήδη εξετάσει αυτό τον κόμβο ή η ζητούμενη ακμή δεν είναι παρούσα στο γράφο αντιστοίχως. Σε αυτή την περίπτωση τις περισσότερες αποδείξεις τις παίρνουμε απευθείας από την ισχύ της προηγούμενης αναλλοίωτης του βρόχου διαχωρίζοντας όλες τις προηγούμενες επαναλήψεις του βρόχου από την τελευταία, όπως και προηγουμένως. Το μόνο σημείο της απόδειξης που εξάγεται από τα αποτελέσματα που μας δίνει η συνθήκη του if αναφέρεται στην απόδειξη ότι μετά το πέρας του βρόχου έχουμε επισκεφθεί όλους τους κόμβους που είναι άμεσοι γείτονες του αρχικού και παρόντες στο γράφο. Βέβαια, αυτό προκύπτει άμεσα από τη συνθήκη του if, διότι στην πρώτη περίπτωση έχουμε ήδη επισκεφθεί τον κόμβο και στη δεύτερη η συγκεκριμένη ακμή δεν είναι παρούσα στο γράφο. Κατόπιν αυτού η υπόλοιπη διαδικασία της απόδειξης μπορεί να θεωρηθεί αρκετά εύκολη, καθώς δεν έχουμε καμία αλλαγή τιμών από το πρόγραμμα.

Τέλος, θα αναφερθούμε στη διαδικασία απόδειξης του τελευταίου λήμματος, η οποία είναι και μία τις δύο βασικότερες που κληθήκαμε να κατασκευάσουμε. Η πλειοψηφία αυτών που πρέπει να αποδείξουμε προέρχεται άμεσα από τη διατήρηση της αναλλοίωτης του βρόχου. Με αυτό τον τρόπο αποδεικνύεται η εγκυρότητα του γράφου όσον αφορά την παρουσία της μίας εκ των δύο αντιστρόφων ακμών σε κάθε στιγμή, η διατήρηση της παρουσίας των ακμών μεταξύ του παλιού και του καινούργιου γράφου, και η πληροφορία ότι δεν έχουμε επισκεφθεί κανέναν από τους κόμβους-προορισμούς. Το μόνο σχετικά δύσκολο σημείο ήταν η απόδειξη ότι τελικά δεν υπάρχει μονοπάτι από τον κόμβο που εξετάζουμε προς οποιονδήποτε κόμβο-προορισμό. Αυτό εξάγεται από το γεγονός ότι ο κόμβος που εξετάζουμε δεν είναι κόμβος-προορισμός, όπως αποδεικνύεται από σχετικό έλεγχο που γίνεται στην αρχή της συνάρτησης, και από το ότι έχουμε επισκεφθεί το σύνολο των γειτόνων του κόμβου στους οποίους μπορούμε να μεταφερθούμε μέσω κάποιας ακμής που είναι παρούσα στον κόμβο και δεν υπάρχει μονοπάτι ούτε από αυτούς προς οποιονδήποτε κόμβο-προορισμό. Το τελευταίο αποδεικνύεται από την αναλλοίωτη του βρόχου. Έχοντας τα παραπάνω κάτι τελευταίο που χρειάστηκε να αποδείξουμε ώστε να οδηγηθούμε σε άτοπο σε ορισμένες περιπτώσεις είναι ότι εφόσον υπάρχει μονοπάτι από έναν κόμβο Α σε έναν κόμβο Β, τότε υπάρχει μονοπάτι και από οποιονδήποτε ενδιάμεσο κόμβο του μονοπατιού προς τον κόμβο Β.

Με αυτό τον τρόπο ολοκληρώσαμε το σύνολο των λημμάτων που χρειάστηκε να αποδείξουμε για να ολοκληρωθεί η τυπική επαλήθευση του προγράμματός μας. Στο τελευταίο τμήμα της εργασίας παρουσιάζουμε συνοπτικά τα αποτελέσματά μας και κάνουμε κάποιες παρατηρήσεις όσον αφορά το εργαλείο Coq που χρησιμοποιήσαμε για την απόδειξη αυτή.

Κεφάλαιο 7

Συμπεράσματα

Στο τελευταίο τμήμα της εργασίας μας παραθέτουμε εν συντομία τα αποτελέσματα της απόδειξής μας και τα προβλήματα που συναντήσαμε κατά τη διάρκεια της υλοποίησης της.

Αυτό που κάναμε είναι η απόδειξη της ορθότητας μιας υλοποίησης του αλγορίθμου των Ford-Fulkerson στη γλώσσα C με χρήση των εργαλείων Caduceus και Coq. Μέχρι τη στιγμή που γράφεται αυτή η εργασία, από όσο γνωρίζουμε, δεν υπάρχει άλλη παρόμοια απόδειξη σε αυτά τα εργαλεία. Συνεπώς, θα μπορούσαμε να πούμε ότι μελετάμε τη δυνατότητα απόδειξης ενός προβλήματος της θεωρίας γράφων, η απόδειξη του οποίου δεν είναι τετριμμένη αλλά μπορεί σχετικά εύκολα να γίνει με χρήση της ανθρώπινης λογικής και της θεωρητικής πληροφορικής, σε ένα εργαλείο μηχανιστικής απόδειξης θεωρημάτων, όπως είναι το Coq.

Από την παραπάνω μελέτη συμπεραίνουμε ότι η απόδειξη ορθότητας τέτοιων προγραμμάτων, καθώς και των λημμάτων που αυτά περιλαμβάνουν, χρησιμοποιώντας εργαλεία μηχανιστικών αποδείξεων (proof assistants) όπως το Coq είναι μία διαδικασία αρκετά επίπονη. Σε αυτό σημαντικό ρόλο παίζει όχι μόνο η ίδια η δυσκολία καταγραφής των αποδείξεων, όπου σε ορισμένες περιπτώσεις, όπως και αυτή που μελετάμε, ακόμα και η καταγραφή των υποχρεώσεων απόδειξης (proof obligations) αποδεικνύεται αρκετά επίπονη, αλλά και από τη δυνατότητα της ανθρώπινης λογικής να χρησιμοποιεί μια αφαιρετική λογική η οποία δεν εγκυάται πάντα την ορθότητα των αποτελεσμάτων που παίρνουμε. Δηλαδή ο άνθρωπος τείνει να θεωρεί ορισμένα πράγματα δεδομένα και τετριμμένα, ενώ αυτό όχι απλά δε συμβαίνει σε κάθε περίπτωση, αλλά ορισμένες φορές οδηγεί ακόμα και σε λανθασμένα συμπεράσματα. Τέτοιες όμως εσφαλμένες αντιλήψεις είναι εξαιρετικά δύσκολο να υπεισέλθουν σε αποδείξεις σε εργαλεία σαν το Coq.

Συνεπώς, από τα παραπάνω γίνεται εμφανές ότι παρόλο που τα εργαλεία μηχανιστικής απόδειξης θεωρημάτων είναι δυνατό να μας δυσκολεύουν μέχρι κάποιο βαθμό σε ορισμένες περιπτώσεις στο να ολοκληρώνουμε την απόδειξη κάποιου θεωρήματος, από την άλλη διασφαλίζουν την ακεραιότητα της απόδειξής μας. Γι' αυτό το λόγο, όπως αναφέραμε και σε προηγούμενο κεφάλαιο, εργαλεία σαν το Coq χρησιμοποιούνται για την απόδειξη της ορθότητας προγραμμάτων σε περιπτώσεις όπου η σημασία της ορθής εκτέλεσης τους είναι εξαιρετικά κρίσιμη.

Βιβλιογραφία

[Bert04] Yves Bertot and Pierre Casteran, *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*, Springer, 2004.

[Corm01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms 2nd edition*, MIT Press, 2001.

[Fili09a] Jean-Christophe Filiâtre, *The WHY verification tool*, INRIA Team Project Proval, 2009.

[Fili09b] Jean-Christophe Filiâtre, Thierry Hubert and Claude Marché, *The Caduceus verification tool for C programmers*, INRIA Team Project Proval, 2009.

[Hube05] Thierry Hubert and Claude Marché, "A case study of C source code verification: The Schorr-Waite algorithm", in *Proceedings of the 3rd IEEE International Conference on Software Engineering and Formal Methods*, pp. 190–199, 2005.

[Meng27] Karl Menger, "Zur allgemeinen Kurventheorie", *Fundamenta Mathematicae*, vol. 10, pp. 96–115, 1927.

[Thol08] Namrata P. Tholiya, "Vertex disjoint paths", Mtech project stage i report, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, 2008.

[USAC] "USA Computing Olympiad". URL: `http://www.uwp.edu/sws/usaco/`.