Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου, Δρ Γιάννη Δημόπουλο, ο οποίος ήταν ο επιβλέπον καθηγητής της διπλωματικής αυτής εργασίας και με βοήθησε ώστε να ολοκληρωθεί με επιτυχία.

Επίσης θα ήθελα να ευχαριστήσω τους γονείς μου, για τη στήριξη και τη βοήθεια που μου πρόσφεραν κατά τη διάρκεια της φοίτησής μου στο πανεπιστήμιο Κύπρου.

Τέλος θα ήθελα να ευχαριστήσω όλους όσους ήταν δίπλα μου όλα αυτά τα χρόνια και με βοήθησαν να πραγματοποιήσω τους στόχους μου.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή

Ένα πρόβλημα ικανοποίησης περιορισμών είναι ένα πρόβλημα του οποίου η λύση απαιτεί την ικανοποίησης πολλών περιορισμών ταυτόχρονα. Μπορεί να εκφραστεί ως ένα σύνολο από περιορισμούς σε ένα πεπερασμένο σύνολο από μεταβλητές. Κάθε μεταβλητή έχει ένα πεπερασμένο πεδίο ορισμού. Μια λύση σε ένα πρόβλημα ικανοποίησης περιορισμών είναι μια ανάθεση σε κάθε μεταβλητή από το πεδίο ορισμού της, έτσι ώστε όλοι οι περιορισμοί να ικανοποιούνται. Ένα τέτοιο πρόβλημα μπορεί να έχει πολλές λύσεις.

Ο προγραμματισμός περιορισμών είναι ένας τρόπος προγραμματισμού που χρησιμοποιείται για τη μελέτη και την επίλυση προβλημάτων ικανοποίησης περιορισμών. Σε αυτό το είδος προγραμματισμού ο χρήστης γράφει ένα πρόγραμμα για ένα πρόβλημα ικανοποίησης περιορισμών, δηλώνοντας τι πρέπει να λυθεί και όχι πώς πρέπει να λυθεί.

Ο πιο γνωστός τρόπος επίλυσης ενός προβλήματος ικανοποίησης περιορισμών είναι ο αλγόριθμος οπισθοδρόμησης, ο οποίος επεκτείνει μια μερική λύση του προβλήματος σε μια ολοκληρωμένη λύση. Αναθέτει σε μια μεταβλητή μια τιμή από το πεδίο ορισμού της και ελέγχει αν αυτή η ανάθεση είναι συνεπής με τις τιμές που έχουν ανατεθεί ήδη σε άλλες μεταβλητές. Αν αυτό ισχύει, τότε η διαδικασία συνεχίζεται αλλιώς ο αλγόριθμος οπισθοδρομεί και αναθέτει τιμή στην επόμενη μεταβλητή. Παράλληλα διατηρεί κάποιου είδους συνέπεια για να μειώσει τα πεδία ορισμού στα οποία αναζητεί τιμές. Οι τεχνικές συνέπειας αφαιρούν τις τιμές στα πεδία ορισμών που δεν είναι συνεπείς με τους περιορισμούς, με αποτέλεσμα ο χώρος αναζήτησης να μειώνεται. Οι τεχνικές συνέπειας μπορούν να εφαρμοστούν πριν την αναζήτηση ή κατά την διάρκεια της αναζήτησης.

Σε ένα πρόβλημα προτασιακής ικανοποιησιμότητας, ένα πρόβλημα περιορισμών εκφράζεται από ένα σύνολο λογικών προτάσεων. Μια λύση στο πρόβλημα προτασιακής ικανοποιησιμότητας είναι μια ανάθεση στις δυαδικές μεταβλητές , η οποία ικανοποιεί όλες τις προτάσεις.

Ένα πρόβλημα προτασιακής ικανοποιησιμότητας μπορεί επίσης να λυθεί με ένα αλγόριθμο οπισθοδρόμησης. Ο πιο γνωστός είναι ο αλγόριθμος Davis-Putnam-Logemann-Loehand. Σε αυτό τον αλγόριθμο χρησιμοποιείται η ιδέα της lookahead. Δηλαδή ανατίθεται μια τιμή σε μια μεταβλητή και ελέγχεται αν αυτή η μεταβλητή οδηγεί σε αντίφαση. Αν υπάρχει αντίφαση τότε η lookahead, αναθέτει την αντίθετη τιμή στην μεταβλητή. Με αυτό τον τρόπο ανατίθεται τιμή στη μεταβλητή χωρίς να γίνει αναζήτηση. Για να παραχθεί η αντίφαση, χρησιμοποιούνται μέθοδοι απαλοιφής, όπως είναι η μοναδιαία απαλοιφή.

Σε ένα πρόβλημα συνόλων απαντήσεων, ένα πρόβλημα περιορισμών αναπαριστάται από ένα λογικό πρόγραμμα το οποίο προσδιορίζει τους περιορισμούς που πρέπει να ικανοποιηθούν. Ένα σύνολο απαντήσεων για ένα λογικό πρόγραμμα, αντιστοιχεί σε μια λύση του προβλήματος περιορισμών. Μια υλοποίηση συστήματος που επιλύει ένα πρόβλημα συνόλων απαντήσεων είναι το Smodels , το οποίο χρησιμοποιεί την ιδέα της lookahead. Ένα μερικό σύνολο απαντήσεων επεκτείνεται σε ένα ολοκληρωμένο σύνολο απαντήσεων με βάση τέσσερις κανόνες διάδοσης περιορισμών, όπως αυτοί υλοποιούνται στη συνάρτηση expand, την οποία καλεί η συνάρτηση lookahead.

Στόχος αυτής της διπλωματικής εργασίας είναι να συγκρίνουμε τις διάφορες μεθόδους που χρησιμοποιούνται για την επίλυση προβλημάτων περιορισμών. Στο τελευταίο κεφάλαιο της διπλωματικής παραθέτουμε τον τρόπο με τον οποίο μπορεί να χρησιμοποιηθεί η συνάρτηση lookahead του συστήματος Smodels για την επίτευξη γενίκευσης συνέπειας σε λογικά προγράμματα συνόλων απαντήσεων έτσι ώστε να μειωθεί ο χώρος αναζήτησης.

Κεφάλαιο 2

Προβλήματα Ικανοποίησης Περιορισμών

2.1 Το πρόβλημα ικανοποίησης περιορισμών(Constraint satisfaction problem)

Ορισμός 2.1: Το πρόβλημα ικανοποίησης περιορισμών (CSP)[6]

Ένα πρόβλημα ικανοποίησης περιορισμών(CSP) είναι μια τριάδα A(X,D,C).Το C είναι ένα πεπερασμένο σύνολο από περιορισμούς $\{c_1,c_2,\ldots, c_n\}$ πάνω σε ένα πεπερασμένο σύνολο από μεταβλητές X= $\{x_1,x_2,\ldots, x_m\}$.Το πεδίο ορισμού D=$\{D_{x1},D_{x2},\ldots,D_{xm}\}$ αντιστοιχεί κάθε μεταβλητή $x_i \in$ X στο πεπερασμένο σύνολο από τιμές D_{xi}. Ένας περιορισμός c_ι είναι μια σχέση R_i που ορίζεται σε ένα υποσύνολο μεταβλητών $S_i \subseteq$ X. Το υποσύνολο S_i ονομάζεται πεδίο του c_i. Αν το υποσύνολο S_i =$\{x_{i1},x_{i2},\ldots, x_{im}\}$,τότε η σχέση R_i είναι ένα υποσύνολο του καρτεσιανού γινομένου $D_{xi1} \times\ldots\times D_{xim}$. Ο βαθμός του περιορισμού αναφέρεται στον αριθμό των μεταβλητών που συμμετέχουν σε αυτό. Ο μοναδιαίος περιορισμός ορίζεται σε μια μεταβλητή. Ο δυαδικός περιορισμός ορίζεται σε δύο μεταβλητές. Ο n-αδικός περιορισμός ορίζεται σε n μεταβλητές.

Ορισμός 2.2: Στιγμιότυπο συνόλου μεταβλητών

Ένα στιγμιότυπο συνόλου μεταβλητών είναι μια ανάθεση σε κάθε μεταβλητή του συνόλου μιας τιμής από το πεδίο ορισμού της. Σε ένα πρόβλημα ικανοποίησης περιορισμών A(X,D,C) , ένα στιγμιότυπο συνόλου μεταβλητών $\{x_{i1},x_{i2},\ldots, x_{im}\}$ είναι ένα σύνολο από ζεύγη $\{(x_{i1},\alpha_{i1}),\ldots,(x_{im}, \alpha_{im})\}$,όπου $\{x_{i1},x_{i2},\ldots, x_{im}\} \subseteq$ X και $\alpha_{ik} \subseteq D_{xik}$. Δηλώνουμε με x → α ή με (x,α) ότι στην μεταβλητή x ανατίθεται η τιμή α $\subseteq D_x$.

Ορισμός 2.3:Ικανοποίηση περιορισμού

Ένας περιορισμός c_i ικανοποιείται από ένα στιγμιότυπο αν και μόνο αν το στιγμιότυπο του συνόλου μεταβλητών που συμμετέχουν στον περιορισμό c_i, έχει ως αποτέλεσμα μια πλειάδα στη σχέση R_i.

Ορισμός 2.4:Μερικό συνεπές στιγμιότυπο

Ένα μερικό στιγμιότυπο είναι συνεπές με ένα περιορισμό c_i αν και μόνο αν έχει ως αποτέλεσμα μια προβολή μιας πλειάδας στη σχέση R_i. Ένα μερικό στιγμιότυπο είναι συνεπές αν και μόνο αν είναι συνεπές με κάθε περιορισμό. Ένα μερικό στιγμιότυπο ονομάζεται επίσης μερική λύση του προβλήματος ικανοποίησης περιορισμών.

Ορισμός 2.5:Λύση στο πρόβλημα ικανοποίησης περιορισμών

Μια λύση σε ένα πρόβλημα ικανοποίησης περιορισμών A(X,D,C) είναι ένα στιγμιότυπο όλων των μεταβλητών του X = $\{x_1,x_2,\ldots, x_m\}$, όπου σε κάθε μεταβλητή x_i ανατίθεται μια τιμή από το πεδίο ορισμού της D_{xi} έτσι ώστε όλοι οι περιορισμοί στο σύνολο C= $\{c_1,c_2,\ldots, c_n\}$ να ικανοποιούνται.

Παράδειγμα 2.1

Ορίζουμε ένα πρόβλημα ικανοποίησης περιορισμών A(X,D,C), όπου X={x,y,z}, D_x= D_y = D_z = {0,1,2} και C_{xy} = {(0,1), (0,2), (1,2)}, C_{yz} = {(0,1),(0,2),(1,2)}. Τότε το σύνολο {x=0,y=1,z=2} είναι μια λύση για το A.

2.2 Τεχνικές συνέπειας περιορισμών (Consistency techniques)[2]

Ένα πρόβλημα ικανοποίησης περιορισμών λύνεται με ένα αλγόριθμο οπισθοδρόμησης, ο οποίος προσπαθεί να επεκτείνει μια μερική λύση σε μια ολοκληρωμένη λύση. Σε κάθε βήμα ανατίθεται στην τρέχουσα μεταβλητή μια τιμή και αν η μερική ανάθεση είναι συνεπής , συνεχίζουμε με την επόμενη μεταβλητή, αν όχι συνεχίζουμε με την επόμενη τιμή στο πεδίο ορισμού της τρέχουσας μεταβλητής. Αν όλες οι τιμές στο πεδίο τιμών έχουν δοκιμαστεί και αποτύχει να δώσουν μια συνεπή ανάθεση, κάνουμε οπισθοδρόμηση στην προηγούμενη μεταβλητή και δοκιμάζουμε εναλλακτικές τιμές από το πεδίο ορισμού της. Ο χώρος αναζήτησης μπορεί να περιοριστεί σε ένα αλγόριθμο οπισθοδρόμησης εφαρμόζοντας λειτουργίες που μικραίνουν τα πεδία ορισμών. Η ιδέα είναι να μειώσουμε τα πεδία των μεταβλητών στις οποίες δεν έχει ανατεθεί τιμή ακόμα, διατηρώντας κάποιο είδος συνέπειας με τους περιορισμούς πριν να επιλέξουμε κάποια τιμή. Η μείωση σε ένα κενό πεδίο ορισμού έχει ως συνέπεια την οπισθοδρόμηση του αλγορίθμου.

Ορισμός 2.6: Μοναδιαία συνέπεια (Node consistency)[2]

Η μοναδιαία συνέπεια απαιτεί κάθε μοναδιαίος περιορισμός σε μια μεταβλητή να ικανοποιείται από όλες τις τιμές στο πεδίο ορισμού της μεταβλητής . Αυτή η συνθήκη μπορεί να γίνει εφικτή μειώνοντας το πεδίο ορισμού της κάθε μεταβλητής στις τιμές που ικανοποιούν όλους τους μοναδιαίους περιορισμούς σε αυτή τη μεταβλητή. Ως αποτέλεσμα οι μοναδιαίοι περιορισμοί μπορούν να αγνοηθούν και να γίνει η υπόθεση ότι είναι ενσωματωμένοι στα πεδία ορισμού.

Παράδειγμα 2.2

Δεδομένης μιας μεταβλητής V με πεδίο ορισμού {1,2,3,4} και ένας περιορισμός $V \leq 3$, η μοναδιαία συνέπεια θα περιορίσει το πεδίο ορισμού στο {1,2,3} και ο περιορισμός μπορεί να αγνοηθεί.

Ορισμός 2.7: Συνέπεια ακμής (Arc consistency)[2]

Η συνέπεια ακμής ορίζει ότι μια μεταβλητή x είναι συνεπής με μια άλλη μεταβλητή y αν για κάθε τιμή a στο πεδίο ορισμού της x υπάρχει μια τιμή b στο πεδίο ορισμού της y έτσι ώστε (a,b) ικανοποιεί το δυαδικό περιορισμό μεταξύ των μεταβλητών x και y. Ένα πρόβλημα CSP έχει συνέπεια ακμής αν όλοι οι δυαδικοί του περιορισμοί έχουν συνέπεια ακμής.

Παράδειγμα 2.3

Έστω οι μεταβλητές x με πεδίο ορισμού {2…6} και y με πεδίο ορισμού {3…7} και ο περιορισμός x<y.Το πρόβλημα έχει συνέπεια ακμής επειδή για κάθε ανάθεση τιμής στη μεταβλητή x , υπάρχει μια ανάθεση τιμής στη μεταβλητή y, έτσι ώστε ο περιορισμός x<y να ικανοποιείται.

Παράδειγμα 2.4

Έστω οι μεταβλητές x με πεδίο ορισμού {2…7} και y με πεδίο ορισμού {3…7} και ο περιορισμός x<y.Το πρόβλημα δεν έχει συνέπεια ακμής αφού αν θέσουμε x=7, δεν υπάρχει τιμή για το y που να ικανοποιεί τον περιορισμό.

Ορισμός 2.8:Αλγόριθμος επιβολής συνέπειας ακμής (AC)[6]

Ο αλγόριθμος επιβολής συνέπειας ακμής παίρνει ως εισαγωγή ένα πρόβλημα ικανοποίησης περιορισμών A (X, D, C) και εκτελεί τις ακόλουθες διαδικασίες μείωσης πεδίων επανειλημμένα έως ότου δεν μπορεί να μειωθεί περαιτέρω κανένα πεδίο.

1. Για οποιοδήποτε περιορισμό c ∈ C , εάν υπάρχει ακριβώς μια μεταβλητή x που δεν της έχει δοθεί τιμή ακόμα από τη μερική περίπτωση(λύση) , αφαιρεί την τιμή d από το πεδίο ορισμού D_x εάν η τιμή d είναι ασυνεπής με την τιμή της μεταβλητής στην οποία έχει δοθεί τιμή στον περιορισμό c.

2. Για οποιοδήποτε περιορισμό c ∈ C όπου και στις δύο μεταβλητές x και y στο πεδίο του c δεν έχει δοθεί ακόμα τιμή, αφαιρεί οποιαδήποτε τιμή d από το D_x για την οποία δεν υπάρχει καμία τιμή στο D_y που, μαζί με την τιμή d, είναι συνεπής με το c.

Παράδειγμα 2.5

1. Υποθέτουμε ένα CSP όπως περιγράφεται από τους περιορισμούς C={x<y,y<z, z≤3}, όπου D_x=D_y={1,2,3 } και D_z={2,3,4 }.Επιβάλλοντας τη μοναδιαία συνέπεια, ο τελευταίος περιορισμός μειώνει το D_z σε D'_z= {2,3 }. Επιβάλλοντας την συνέπεια ακμής, ο πρώτος περιορισμός μειώνει το D_x σε D'_x={1,2 }.Παρόμοια το D_y μειώνεται σε D'_y ={2,3 }. Με βάση τον περιορισμό y<z για τη μεταβλητή y , το D'_y μειώνεται περαιτέρω σε {2 }, το οποίο αναγκάζει το D'_x να γίνει {1 } από τον πρώτο περιορισμό, και το D'_z να γίνει {3 } από τον τελευταίο περιορισμό. Καμία περαιτέρω μείωση δεν είναι δυνατή

Ορισμός 2.9: Συνέπεια μονοπατιού(Path consistency)[2]

Η συνέπεια μονοπατιού είναι μια διαδικασία παρόμοια με την συνέπεια ακμής, αλλά εξετάζει ζευγάρια μεταβλητών αντί μόνο μια μεταβλητή. Τυπικά οι μεταβλητές x_i και x_j έχουν μονοπάτι ακμής με τη μεταβλητή x_k εάν, για κάθε ζευγάρι τιμών (a,b) που ικανοποιεί το δυαδικό περιορισμό μεταξύ x_i και x_j , υπάρχει μια τιμή c στο πεδίο ορισμού του x_k έτσι ώστε (a,c) και (b,c) ικανοποιεί τον περιορισμό μεταξύ x_i και x_k και μεταξύ του x_j και x_k, αντίστοιχα. Ένα πρόβλημα CSP έχει συνέπεια μονοπατιού αν για κάθε ζευγάρι μεταβλητών x_i και x_j ,όπου i ≠ j και για κάθε συνεπή ανάθεση στις μεταβλητές αυτές υπάρχει συνέπεια μονοπατιού.

Παράδειγμα 2.6

Λαμβάνοντας υπόψη ένα CSP το οποίο έχει τρεις περιορισμούς σε τρεις μεταβλητές x,y και z με πεδία ορισμού D_x= D_y = D_z ={0,1}, C_{xy} = {(0,1) (1,0)}, C_{xz}= {(0,1)(1,0)}, και C_{zy} = {(0,1) (1,0)}.

Είναι προφανές ότι αυτό το CSP έχει συνέπεια ακμής. Όμως για τον περιορισμό (0,1) Є C_{xy} δεν υπάρχει κάποια τιμή a για το z έτσι ώστε να ισχύει ταυτόχρονα (0,α) Є C_{xz} και (α,1) Є C_{zy}. Για αυτό τον λόγο το CSP δεν έχει συνέπεια μονοπατιού.

Ορισμός 2.10: Επιβολή συνέπειας μονοπατιού[2]

Για να εφαρμόσουμε συνέπεια μονοπατιού σε ένα πρόβλημα ικανοποίησης περιορισμών θα πρέπει να αφαιρέσουμε ζευγάρια τιμών από τα πεδία ορισμών των μεταβλητών ή να αλλάξουμε τους περιορισμούς του προβλήματος.

Παράδειγμα 2.7

Δεδομένου του CSP με τις μεταβλητές x,y και z με πεδία ορισμού D_x= D_y = D_z={0,1} και τους περιορισμούς x<y, y<z και z<x, συνδυάζουμε τους δύο περιορισμούς x<y και y<z και συμπεραίνουμε ότι ισχύει ο περιορισμός x<z.Ο τελευταίος περιορισμός έρχεται σε αντίφαση με τον περιορισμό z<x,άρα το πρόβλημα δεν έχει συνέπεια μονοπατιού. Επίσης εξορισμού της συνέπειας μονοπατιού, για την ανάθεση τιμών ((x,0),(z,1)) ή για την ανάθεση τιμών ((x,1),(z,0)) δεν υπάρχει τιμή για το y έτσι ώστε να ισχύουν οι περιορισμοί x<y και y<z. Επίσης για την ανάθεση τιμών ((y,0),(x,1)) ή για την ανάθεση τιμών ((y,1),(x,0)) δεν υπάρχει τιμή για το z έτσι ώστε να ισχύουν οι περιορισμοί y<z και z<x.Για αυτό το πρόβλημα δεν έχει συνέπεια μονοπατιού.Για να επιβάλλουμε συνέπεια μονοπατιού στο πιο πάνω πρόβλημα θα πρέπει να αφαιρέσουμε δυάδες τιμών από τα πεδία ορισμών. Παρατηρούμε ότι δεν είναι δυνατή η αφαίρεση οποιοδήποτε τιμών από τα πεδία ορισμών για να επιβάλουμε συνέπεια μονοπατιού. Το πιο πάνω πρόβλημα δεν έχει λύση.

Παράδειγμα 2.8

Έστω το CSP με τις μεταβλητές x_1, x_2 και x_3 όλες με πεδίο ορισμού {1,2}. Έστω ότι ισχύουν οι περιορισμοί $x_1 \neq x_2$ και $x_2 \neq x_3$. Λαμβάνοντας υπόψη το συνδυασμό αναθέσεων τιμών ((x_1,1),(x_3, 2)) ,συμπεραίνουμε ότι αυτό το πρόβλημα ικανοποίησης περιορισμών δεν έχει συνέπεια μονοπατιού επειδή αυτός ο συνδυασμός αναθέσεων τιμών στο ζευγάρι μεταβλητών x_1 και x_3 δεν μπορεί να επεκταθεί αναθέτοντας μια

τιμή στη μεταβλητή x_2 που να ικανοποιεί και τους δύο περιορισμούς του προβλήματος. Προσθέτοντας τον περιορισμό $x_1 = x_3$, επιβάλλουμε συνέπεια μονοπατιού στο πρόβλημα αφού για ((x_1,1),(x_3, 1)) υπάρχει η ανάθεση x_2= 2 που ικανοποιεί τους περιορισμούς $x_1 \neq x_2$ και $x_2 \neq x_3$ και για ((x_1,2),(x_3, 2)) υπάρχει η ανάθεση x_2= 2, που ικανοποιεί τους περιορισμούς $x_1 \neq x_2$ και $x_2 \neq x_3$. Χωρίς την πρόσθεση του επιπλέον περιορισμού, η επιβολή συνέπειας μονοπατιού στο πιο πάνω πρόβλημα πραγματοποιείται με την αφαίρεση των μη επιτρεπτών πλειάδων τιμών από τα πεδία ορισμού των μεταβλητών. Δηλαδή αφαιρώντας την τιμή 2 από τα πεδία ορισμών D_{x1} και D_{x3}, παίρνουμε τα πεδία D'_{x1} = {1} και D'_{x3} ={1} οπότε για ((x_1,1),(x_3, 1)) ,που είναι η μόνη δυνατή ανάθεση τιμών στις μεταβλητές x_1 και x_3, υπάρχει η ανάθεση x_2= 2 που ικανοποιεί τους περιορισμούς $x_1 \neq x_2$ και $x_2 \neq x_3$. Συνέπεια ακμής μπορεί επίσης να επιτευχθεί αν αφαιρέσουμε την τιμή 1 αντί την τιμή 2 από τα πεδία ορισμών D_{x1} και D_{x3}.

Ορισμός 2.11: Γενικευμένη Συνέπεια(k-consistency)[2]

Δεδομένου ενός CSP A(X,D,C), αν για κάθε συνεπή ανάθεση k-1 μεταβλητών, υπάρχει μια ανάθεση σε κάποια k μεταβλητή έτσι ώστε οι k τιμές ικανοποιούν όλους τους περιορισμούς μεταξύ των k μεταβλητών, τότε το για A ισχύει κ-γενίκευση συνέπειας. Αν το κ=1 τότε έχουμε μοναδιαία συνέπεια. Αν το κ=2, τότε έχουμε συνέπεια ακμής. Αν το κ=3, τότε έχουμε συνέπεια μονοπατιού. Για να δείξουμε γενικευμένη συνέπεια θα πρέπει να εξετάσουμε κάθε συνεπή k-1 ανάθεση και κάθε άλλη μεταβλητή και να δείξουμε ότι αυτή η επέκταση ανάθεσης είναι συνεπής.

Παράδειγμα 2.9

Δεδομένου ενός CSP το οποίο έχει τρεις μεταβλητές x_1, x_2, x_3 με D_{x1} = D_{x2}=D_{x3}={1,2} και τέσσερις περιορισμούς $x_1 \neq x_2$, $x_1 \neq x_3$, $x_2 \neq x_3$. Αφού θέσουμε την τιμή x_1= 1,που αποτελεί μερική ανάθεση(λύση) του προβλήματος, διαλέγουμε οποιαδήποτε άλλη μεταβλητή π.χ. την x_3 και μπορούμε να βρούμε μια συνεπή ανάθεση που ικανοποιεί όλους τους περιορισμούς, π.χ. x_3=2.Παρατηρούμε ότι το πιο πάνω CSP έχει συνέπεια ακμής. Για να δούμε αν το πιο πάνω CSP έχει συνέπεια μονοπατιού, αναθέτουμε στις μεταβλητές x_1 = 1 και x_2 = 2.Παρατηρούμε ότι δεν υπάρχει μια τρίτη μεταβλητή που όταν της αναθέσουμε μια τιμή θα υπάρχει λύση για τρεις μεταβλητές. Για αυτό το λόγω το πιο πάνω CSP δεν έχει συνέπεια μονοπατιού.

Ορισμός 2.12: Μοναδιαία συνέπεια ακμής(Singleton arc-consistency)[2]

Υποθέτουμε ότι αναθέτουμε μια τιμή v σε μια μεταβλητή x, και ακολούθως επιβάλουμε συνέπεια ακμής . Εάν οποιοδήποτε πεδίο γίνει κενό, αυτό σημαίνει ότι η τιμή v για τη μεταβλητή x δεν μπορεί να οδηγήσει σε οποιαδήποτε λύση. Έτσι, μπορούμε ακίνδυνα να αφαιρέσουμε την τιμή v από το πεδίο D_x.

Παράδειγμα 2.10

Δεδομένου του CSP με τις μεταβλητές x,y και z με πεδία ορισμού $D_x = D_y = D_z$= {0,1,2,3}. Έχουμε τους εξής περιορισμούς

C_{xy} = {(0,0),(0,1),(1,2),(1,3),(2,0),(2,1),(3,2),(3,3)}

C_{xz} = {(0,0),(0,1),(1,2),(1,3),(2,0),(2,1),(3,2),(3,3)}

C_{zy} = {(0,2),(1,3),(2,0),(3,1),(1,2),(0,3),(3,0),(2,1)}

Αυτό το πρόβλημα ικανοποίησης περιορισμών P έχει συνέπεια ακμής αλλά δεν έχει συνέπεια μοναδιαίας ακμής. Υποθέτουμε ότι περιορίζουμε το D_x στο {0}, και παίρνουμε το εξής A(P):

C_{xy} = {(0,0),(0,1)}

C_{xz} = {(0,0),(0,1)}

C_{zy} = {(0,2),(1,3),(2,0),(3,1),(1,2),(0,3),(3,0),(2,1)}

Επιβάλλοντας συνέπεια ακμής αφαιρούμε τις τιμές 2 and 3 από το D_y και όλες τις τιμές από το D_z.Το γεγονός ότι το πεδίο ορισμού γίνεται κενό σημαίνει ότι η τιμή a δεν μπορεί να οδηγήσει σε οποιαδήποτε λύση και την αφαιρούμε από το πεδίο D_x

Ορισμός 2.13: Μοναδιαία γενικευμένη συνέπεια (Singleton k-consistency)[2]

Ομοίως, η ιδέα της μοναδιαίας συνέπειας ακμής μπορεί να εφαρμοστεί σε οποιαδήποτε γενίκευση συνέπειας με τον εξής τρόπο: Περιορίζουμε μια μεταβλητή σε μια τιμή, και επιβάλλουμε γενίκευση συνέπειας . Εάν αυτό οδηγεί σε ασυνέπεια επειδή κάποιος περιορισμός δεν μπορεί να ικανοποιηθεί, τότε η τιμή αφαιρείται από το πεδίο ορισμού της μεταβλητής. Η διαδικασία γίνεται για κάθε τιμή και για κάθε μεταβλητή. Έτσι, μηδέν ή περισσότερες τιμές μπορούν να αφαιρεθούν.

Κεφάλαιο 3

Προτασιακή Ικανοποιησιμότητα

3.1 Προτασιακή λογική(Propositional logic)

Η προτασιακή λογική λειτουργεί σε προτάσεις και συνδέσμους. Μια πρόταση είναι μια λογική κατασκευή που αξιολογείται σε αληθινή ή ψευδή. Ένα παράδειγμα μιας πρότασης είναι η εξής "Είναι ασφαλές να προχωρήσουμε ". Η λογική αλήθεια της πιο πάνω πρότασης είναι ότι μπορούμε να προχωρήσουμε χωρίς να πρέπει να ανησυχήσουμε ότι υπάρχει περίπτωση να κτυπήσουμε σε ένα εμπόδιο.

Υπάρχουν δύο τύποι προτάσεων που μπορούμε να προσδιορίσουμε. Οι ατομικές και σύνθετες προτάσεις. Οι ατομικές προτάσεις αποτελούνται από μια μοναδιαία δήλωση. Ένα τέτοιο παράδειγμα είναι η ατομική πρόταση " Είναι ασφαλές να προχωρήσουμε". Μπορούμε να αντιπροσωπεύσουμε μια τέτοια πρόταση με ένα μοναδιαίο σύμβολο, π.χ. το P. Εάν το P είναι αληθές, σημαίνει ότι μπορούμε να προχωρήσουμε ακίνδυνα. Εάν είναι ψευδές, θα μπορούσαμε να υποστούμε συνέπειες αν προχωρούσαμε. Μια αληθής ατομική πρόταση αναφέρεται ως μια θετική δυαδική μεταβλητή (positive literal) ενώ μια ψευδής ατομική πρόταση καλείται ως μια αρνητική δυαδική μεταβλητή(negative literal). Οι σύνθετες προτάσεις χτίζονται από τις ατομικές προτάσεις χρησιμοποιώντας τους λογικούς συνδέσμους που περιγράφονται στον πίνακα 3.1.

Πίνακας 3.1.Λογικοί σύνδεσμοι

¬	Μοναδιαίος τελεστής άρνησης.
Λ	Δυαδικός τελεστής και
V	Δυαδικός τελεστής ή
⇒	Λογικός τελεστής συνεπαγωγής
⇔	Λογικός τελεστής διπλής συνεπαγωγής

Κάθε σύνθετη πρόταση μπορεί να αξιολογηθεί ως αληθής ή ψευδής. Η σημασιολογία της προτασιακής λογικής καθορίζει αν μια πρόταση είναι αληθής ή ψευδής . Σε σχέση με τη βασική αριθμητική, η πρόταση "x + y = 4" είναι αληθής στην περίπτωση όπου "x = 2 και y = 2" , αλλά η πρόταση "x + y = 4" είναι ψευδής εάν "x = 1 και y = 1". Η σημασιολογία της προτασιακής λογικής συνοψίζεται σε έναν πίνακα αληθείας όπως φαίνεται στον πίνακα 3.2.

Πίνακας 3.2.Πίνακας αληθείας

P	Q	¬P	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	P ⇔ Q
Ψευδής	Ψευδής	αληθής	ψευδής	ψευδής	αληθής	αληθής
ψευδής	αληθής	αληθής	ψευδής	αληθής	αληθής	ψευδής
αληθής	ψευδής	ψευδής	ψευδής	αληθής	ψευδής	ψευδής
αληθής	αληθής	ψευδής	αληθής	αληθής	αληθής	αληθής

Ορισμός 3.1:Η βάση γνώσης στην προτασιακή λογική

Μια βάση γνώσεων είναι ένα σύνολο προτάσεων που είναι γνωστό ότι είναι αληθείς. Για ένα ευφυή πράκτορα, θα μπορούσαμε να δημιουργήσουμε αυτή τη βάση γνώσης από προτάσεις που εκ των προτέρων γνωρίζουμε ότι είναι αληθείς. Ο πράκτορας έπειτα μπορεί να ενισχύσει τη βάση γνώσης του συγκεντρώνοντας πληροφορίες κατά τη διάρκεια της αποστολής του μέσω των αισθητήρων του.

Ορισμός 3.2: Ερμηνεία στην προτασιακή λογική

Μια ερμηνεία στην προτασιακή λογική είναι μια αναπαράσταση που καθορίζει την αλήθεια ή την αναλήθεια κάθε πρότασης. Τυπικά, είναι μια ανάθεση τιμών στις δυαδικές μεταβλητές των προτάσεων. Αξίζει να σημειωθεί ότι καθορισμός της αλήθειας ή της αναλήθειας μιας λογικής πρότασης μπορεί να μην είναι εφικτός αν δεν υπάρχει ανάλογη ερμηνεία. Η πρόταση P: "είναι ασφαλές να προχωρήσουμε " έχει δύο πιθανές ερμηνείες. Η P είναι αληθής ή η P είναι ψευδής. Αν δεν υπάρχει κάποια ερμηνεία δεν μπορούμε να ξέρουμε αν η πρόταση P είναι αληθής ή ψευδής. Η πρόταση Q :"x + y = 4" έχει ένα αριθμό ερμηνειών που έχει άμεση σχέση με τα πεδία ορισμών των μεταβλητών x και y. Έστω οι μεταβλητές x και y έχουν πεδίο ορισμού {1,2}.Η ερμηνεία x=2 και y=2 καθορίζει ότι η πρόταση Q είναι αληθής. Αντίθετα η ερμηνεία x=1 και y=2 καθορίζει ότι η πρόταση Q είναι ψευδής.

Ορισμός 3.3: Συνεπαγωγή μεταξύ των λογικών προτάσεων και λογικό συμπέρασμα

Η συνεπαγωγή μεταξύ των προτάσεων της προτασιακής λογικής δηλώνει τη λογική σχέση που ισχύει έτσι ώστε μια λογική πρόταση να ακολουθεί λογικά μια άλλη πρόταση. Συμβολικά η συνεπαγωγή αναπαριστάται ως a |= b αν και μόνο αν σε κάθε ερμηνεία στην οποία η πρόταση a είναι αληθής συνεπάγεται επίσης ότι και η πρόταση b είναι αληθής. Το λογικό συμπέρασμα είναι μια διαδικασία που καθορίζει εάν μια πρόταση συνεπάγεται από τη βάση γνώσεων.

Ορισμός 3.4: Λογική ισοδυναμία και έγκυρη πρόταση.

Δύο προτάσεις P και Q είναι λογικά ισοδύναμες αν είναι αληθείς στο ίδιο σύνολο ερμηνειών. Οι βασικές λογικές ισοδυναμίες δηλώνονται στον πίνακα 3.3.Μια πρόταση είναι έγκυρη εάν είναι αληθής σε όλες τις ερμηνείες. Παραδείγματος χάριν, η πρόταση (P ∨ ¬P) είναι μια έγκυρη πρόταση. Οι έγκυρες προτάσεις είναι επίσης γνωστές ως ταυτολογίες.

Πίνακας 3.3: Βασικές λογικές ισοδυναμίες

Ισοδυναμία διπλής άρνησης ¬(¬p) ⇔ p
Ισοδυναμίες αντιμεταθετικότητας p ∨ q ⇔ q ∨ p p ∧ q ⇔ q ∧ p
Ισοδυναμίες προσεταιριστικότητας
(p ∨ q) ∨ r ⇔ p ∨ (q ∨ r) (p ∧ q) ∧ r ⇔ p ∧ (q ∧ r)
Ισοδυναμίες Επιμεριστικότητας p ∨ (q ∧ r) ⇔ (p ∨ q) ∧ (p ∨ r) p ∧ (q ∨ r) ⇔ (p ∧ q) ∨ (p ∧ r)
Ισοδυναμίες De Morgan ¬(p ∨ q) ⇔ ¬p ∧ ¬ q ¬(p ∧ q) ⇔ ¬p ∨ ¬ q

Ορισμός 3.5: Ικανοποιησιμότητα.

Μια λογική πρόταση μπορεί να ικανοποιηθεί αν είναι αληθής σε κάποια ερμηνεία. Αν μια πρόταση a είναι αληθής σε κάποια ερμηνεία m,δηλώνουμε ότι η ερμηνεία m ικανοποιεί την πρόταση a.

Συμπέρασμα 3.1:

Με βάση τους πιο πάνω ορισμούς και δεδομένης μιας βάσης γνώσεων, που είναι γνωστό ότι περιέχει αληθείς προτάσεις, μπορούμε να ελέγξουμε αν η βάση γνώσεων υποστηρίζει αν μια άλλη πρόταση εκτός της βάσης γνώσεων είναι αληθής. Εάν θέλουμε να απαντήσουμε στην ερώτηση "είναι ασφαλές να προχωρήσουμε;", μπορούμε να το δηλώσουμε ως πρόταση P: "είναι ασφαλές να προχωρήσουμε". Κατόπιν ελέγχουμε εάν η βάση γνώσεων συνεπάγεται την P. Αν η βάση γνώσεων συνεπάγεται την P, τότε η P είναι αληθής και καταλήγουμε στο συμπέρασμα ότι "είναι ασφαλές να προχωρήσουμε". Αν η βάση δεν συνεπάγεται την P, μπορούμε να ελέγξουμε εάν η βάση γνώσεων συνεπάγεται την πρόταση ¬P. Αν σε αυτή την περίπτωση η βάση γνώσεων συνεπάγεται την ¬P, η P είναι ψευδής και συμπεραίνουμε ότι "δεν είναι ασφαλές να προχωρήσουμε". Αν δεν συνεπάγεται η P ούτε η ¬P, συμπεραίνουμε ότι δεν μπορούμε να ξέρουμε από τη βάση γνώσεων την αλήθεια της πρότασης.

Δεδομένου ότι η βάση γνώσεων είναι μια ένα σύνολο από προτάσεις, μπορούμε να την δούμε ως ενιαία σύνθετη πρόταση KB. Ένας τρόπος να εξετάσουμε αν η πρόταση P συνεπάγεται ή όχι από τη σύνθετη πρόταση KB είναι η χρησιμοποίηση του ορισμού της συνεπαγωγής. Ο ορισμός αυτός απαιτεί να βρούμε όλες τις ερμηνείες που είναι αληθείς στη σύνθετη πρόταση KB και να ελέγξουμε αν όλες αυτές οι ερμηνείες είναι επίσης αληθείς στη πρόταση P.Αυτή όμως η διαδικασία είναι εξαιρετικά υπολογιστικά εξαντλητική. Χρησιμοποιώντας την έννοια της ικανοποιησιμότητας, εξετάζουμε αν η συνεπαγωγή KB ⇒ P είναι έγκυρη οπότε η πρόταση ¬(KB ⇒ P) δεν μπορεί να ικανοποιηθεί. Την τελευταία πρόταση μπορούμε να γράψουμε και ως (KB Λ ¬P). Έτσι, για να ελέγξουμε αν ισχύει η συνεπαγωγή KB ⇒ P ελέγχουμε εάν η πρόταση (KB Λ ¬P) μπορεί να ικανοποιηθεί. Αν η πρόταση μπορεί να ικανοποιηθεί τότε συμπεραίνουμε ότι η πρόταση P συνεπάγεται από το KB.

3.2 Το πρόβλημα ικανοποιησιμότητας (The satisfiability problem)[4]

Ένα προτασιακό πρόβλημα ικανοποιησιμότητας είναι πρόβλημα που στόχο έχει να καθορίσει αν κάποια απάντηση ικανοποιεί μια δεδομένη λογική πρόταση. Συνήθως είναι ευκολότερο να αναπαριστούμε την πρόταση σε κανονική συζευκτική μορφή (CNF)

Μια σύνθετη πρόταση σε κανονική συζευκτική μορφή είναι μια σύζευξη προτάσεων, δηλαδή οι προτάσεις συνδέονται με τον δυαδικό τελεστή και. Αυτές οι προτάσεις είναι διαζεύξεις των δυαδικών μεταβλητών, δηλαδή οι λογικές μεταβλητές συνδέονται με το λογικό τελεστή ή. Πρέπει να σημειωθεί ότι η κανονική συζευκτική μορφή επιτρέπει μόνο τις αρνητικές δυαδικές μεταβλητές και όχι την άρνηση των προτάσεων. Από τις σχέσεις ισοδυναμίας μπορούμε να μετασχηματίσουμε οποιαδήποτε σύνολο προτάσεων σε κανονική συζευκτική μορφή.

Ειδικότερα ένα προτασιακό πρόβλημα ικανοποιησιμότητας έχει στόχο να καθορίσει την ικανοποιησιμότητα μιας λογικής πρότασης. Παίρνει ως είσοδο ένα σύνολο προτάσεων σε κανονική συζευκτική μορφή και απαντά στην έξοδο αν οι προτάσεις στο σύνολο είναι ικανοποιήσιμες. Αν οι προτάσεις είναι ικανοποιήσιμες δίνει στην έξοδο την ερμηνεία που ικανοποιεί τις προτάσεις.

Παράδειγμα 3.1:

Υποθέτουμε το πρόβλημα ικανοποιησιμότητας δυαδικών προτάσεων, το οποίο ορίζεται από τη σύνθετη πρόταση F= (x ∨ y) ∧ (¬x ∨ ¬y).Το πρόβλημα F μπορεί να ικανοποιηθεί θέτοντας στην μεταβλητή x=αληθής και στην μεταβλητή y=ψευδής οπότε η απάντηση είναι ΝΑΙ, δηλαδή η πρόταση είναι αληθής. Αν δεν υπήρχε καμιά δυνατή ανάθεση, η απάντηση θα ήταν ΟΧΙ, δηλαδή η πρόταση είναι ψευδής.

Παράδειγμα 3.2:

Υποθέτουμε το πρόβλημα ικανοποιησιμότητας τριαδικών προτάσεων, το οποίο ορίζεται από τη σύνθετη πρόταση E = $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_4)$.Το πρόβλημα Ε μπορεί να ικανοποιηθεί με την ανάθεση (x_1= αληθής, x_2= αληθής, x_3= αληθής, x_4= αληθής),οπότε η απάντηση είναι ΝΑΙ, δηλαδή η πρόταση είναι αληθής. Αν δεν υπήρχε καμιά δυνατή ανάθεση. Αυτή είναι μια από πολλές πιθανές αναθέσεις. Για παράδειγμα, οποιοδήποτε σύνολο αναθέσεων περιλαμβάνει το x_1 = true είναι ικανοποιητική. Αν δεν υπήρχε καμιά δυνατή ανάθεση, η απάντηση θα ήταν ΟΧΙ, δηλαδή η πρόταση είναι ψευδής.

3.3 Κανόνες απαλοιφής[4]

Ορισμός 3.6: Συμπληρωματικές δυαδικές μεταβλητές και αντίφαση

Δύο δυαδικές μεταβλητές είναι συμπληρωματικές αν αναφέρονται στην ίδια λογική πρόταση P αλλά η μια μεταβλητή δηλώνει ότι η πρόταση είναι αληθής και συμβολίζεται με p και η άλλη μεταβλητή δηλώνει ότι η πρόταση είναι ψευδής και συμβολίζεται με ¬p. Αν ισχύει η σύνθετη πρόταση p ∧¬p,τότε υπάρχει αντίφαση και η σύνθετη πρόταση είναι πάντα ψευδής.

Ορισμός 3.7: Ο γενικός κανόνας απαλοιφής (Resolution rule)

Ο γενικός κανόνας απαλοιφής στην προτασιακή λογική είναι ένας έγκυρος κανόνας συμπεράσματος που παράγει, από δύο προτάσεις, μια νέα πρόταση που υπονοείται από αυτές. Ο κανόνας απαλοιφής παίρνει δύο λογικές προτάσεις σε κανονική συζευκτική μορφή, οι οποίες μπορεί να περιέχουν και συμπληρωματικές δυαδικές μεταβλητές , και παράγει μια νέα πρόταση με όλες τις μεταβλητές και από τις

δύο προτάσεις εκτός από τα συμπληρωματικές τους. Τυπικά, εκτιμώντας ότι οι μεταβλητές a_i και b_j είναι δυαδικές μεταβλητές και δεδομένων των λογικών προτάσεων:

$a_1 \vee \ldots \vee a_{i-1} \vee a_i \vee a_{i+1} \ldots \vee a_n$ και $b_1 \vee \ldots \vee b_{j-1} \vee b_j \vee b_{j+1} \vee \ldots \vee b_m$,το αποτέλεσμα του κανόνα απαλοιφής θα είναι η λογική πρόταση :

$a_1 \vee \ldots \vee a_{i-1} \vee a_{i+1} \ldots \vee a_n \vee b_1 \vee \ldots \vee b_{j-1} \vee b_{j+1} \vee \ldots \vee b_m$

Η πρόταση που παράγεται από τον κανόνα απαλοιφής καλείται διαλυτική πρόταση των δύο εισαγόμενων προτάσεων. Όταν οι δύο προτάσεις περιέχουν περισσότερα από ένα ζευγάρια από συμπληρωματικές δυαδικές μεταβλητές, ο κανόνας απαλοιφής μπορεί να εφαρμοστεί ανεξάρτητα για κάθε τέτοιο ζευγάρι. Εντούτοις, μόνο ένα ζευγάρι των συμπληρωματικών μεταβλητών μπορεί να αφαιρεθεί. Τα άλλα ζευγάρια των συμπληρωματικών μεταβλητών παραμένουν στη διαλυτική πρόταση. Λαμβάνοντας υπόψη τη λογική πρόταση C ∨ {v} και τη λογική πρόταση D ∨ {¬v} συμπεραίνουμε τη λογική πρόταση C U D,αφαιρώντας τις συμπληρωματικές μεταβλητές v και ¬v.

Ορισμός 3.8:Μοναδαία απαλοιφή(Unit resolution)

Μια μοναδιαία πρόταση είναι μια πρόταση με μια δυαδική μεταβλητή. Στη κανονική συζευκτική μορφή μια μοναδιαία πρόταση αναγκάζει τη δυαδική μεταβλητή της να πάρει την τιμή αληθής. Ο κανόνας μοναδιαίας απαλοιφής είναι μια εφαρμογή του γενικού κανόνα απαλοιφής, όπου μια πρόταση είναι μια μοναδιαία πρόταση. Η μοναδιαία απαλοιφή ονομάζεται επίσης μοναδιαία μετάδοση. Το μοναδιαία απαλοιφή είναι μια επαναληπτική διαδικασία αφαίρεσης των μοναδιαίων προτάσεων μέχρι (α) μια αντίφαση να επιτευχθεί, ή (β) δεν υπάρχουν άλλες μοναδιαίες προτάσεις στην είσοδο.

Παράδειγμα 3.3

Θεωρούμε τη λογική σύνθετη πρόταση (a ∨ b) ∧ (¬a). Εφαρμόζοντας μοναδιαία απαλοιφή στη μεταβλητή a, έχουμε ως αποτέλεσμα τη λογική πρόταση b, οπότε η αρχική λογική πρόταση ικανοποιείται για b = αληθές.

Ορισμός 3.9: Δυαδική απαλοιφή (Binary resolution)

Η δυαδική απαλοιφή εστιάζει πάντα σε δύο λογικές προτάσεις από τις οποίες τουλάχιστον μια έχει δύο δυαδικές μεταβλητές και σε μια δυαδική μεταβλητή στην κάθε λογική πρόταση. Για να αναγνωρίσει η δυαδική απαλοιφή ένα συμπέρασμα, πρέπει οι δύο μεταβλητές στις οποίες εστιάζει, να είναι συμπληρωματικές.

Παράδειγμα 3.4

Λαμβάνοντας υπόψη την πρόταση a ∨ b και την πρόταση ¬ a ∨ c συμπεραίνουμε την πρόταση c v b,αφαιρώντας τις συμπληρωματικές μεταβλητές a και ¬a.

Παράδειγμα 3.5

Εφαρμόζουμε δυαδική απαλοιφή στη σύνθετη πρόταση Q = (a ∨ b) ∧ (¬a ∨ c) ∧ (¬b ∨ c),έχουμε ως λογικό συμπέρασμα τη σύνθετη πρόταση P = (b ∨ c) ∧ (¬b ∨ c). Εφαρμόζοντας τη δυαδική απαλοιφή στην πρόταση P έχουμε τη λογική μοναδιαία πρόταση c. Οπότε συμπεραίνουμε ότι η πρόταση Q ικανοποιείται για c = αληθές Παρατηρούμε ότι η μοναδιαία απαλοιφή δεν μπορεί να εφαρμοστεί στη σύνθετη πρόταση Q, επειδή αυτή δεν περιέχει μοναδιαίες προτάσεις.

Συμπέρασμα 3.2:

Το παράδειγμα 3.5, δείχνει ότι αν εφαρμόσουμε δυαδική απαλοιφή αντί μοναδιαία απαλοιφή θα πάρουν τιμή περισσότερες μεταβλητές.

Ορισμός 3.10: Τριαδική απαλοιφή (Ternary resolution)

Η τριαδική απαλοιφή εστιάζει πάντα σε δύο λογικές προτάσεις από τις οποίες τουλάχιστον μια έχει τρεις δυαδικές μεταβλητές η κάθε μια και σε μια μεταβλητή λογικής στην κάθε μια. Για να αναγνωρίσει η τριαδική απαλοιφή ένα συμπέρασμα θα πρέπει οι δύο αυτές μεταβλητές να είναι συμπληρωματικές.

Παράδειγμα 3.6

Λαμβάνοντας υπόψη την πρόταση a ∨ b ∨ c και την πρόταση ¬a ∨ c ∨ d συμπεραίνουμε την πρόταση d ∨ b ∨ c,αφαιρώντας τα συμπληρωματικές μεταβλητές λογικής a και ¬a.

Παράδειγμα 3.7

Εφαρμόζοντας τριαδική απαλοιφή εστιάζοντας στη δυαδική μεταβλητή a στη λογική πρόταση Q= (a ∨ ¬b ∨ d) ∧ (¬a ∨ ¬b ∨ d) ∧ (¬c ∨ b ∨ d) ∧ (c ∨ b ∨ d) συμπεραίνουμε τη λογική πρόταση P = (¬b ∨ d) ∧ (¬c ∨ b ∨ d) ∧ (c ∨ b ∨ d). Ακολούθως εφαρμόζοντας τριαδική απαλοιφή εστιάζοντας στη δυαδική μεταβλητή c στην πρόταση P παίρνουμε την πρόταση S = (¬b ∨ d) ∧ (b ∨ d).Εφαρμόζοντας δυαδική απαλοιφή εστιάζοντας στη δυαδική μεταβλητή b στην πρόταση S παίρνουμε την πρόταση d. Από τα πιο πάνω συμπεραίνουμε ότι η πρόταση Q ικανοποιείται για d = αληθές. Παρατηρούμε ότι η δυαδική απαλοιφή δεν μπορεί να εφαρμοστεί στην πρόταση Q.

Συμπέρασμα 3.3:

Το παράδειγμα 3.7 πιστοποιεί ότι αν τρέξουμε τριαδική απαλοιφή αντί δυαδική απαλοιφή παίρνουν τιμή περισσότερες μεταβλητές.

3.4 Μετάφραση του προβλήματος ικανοποιησιμότητας σε πρόβλημα ικανοποίησης περιορισμών.

Υπάρχουν διάφοροι τρόποι με τους οποίους ένα πρόβλημα ικανοποιησιμότητας μπορεί να μεταφραστεί σε πρόβλημα ικανοποίησης περιορισμών Αυτές είναι η δυαδική μετάφραση, η μετάφραση κριμένης μεταβλητής, η μετάφραση λογικής μεταβλητής και η μη δυαδική μετάφραση.

Ορισμός 3.11: Δυαδική μετάφραση (Dual Encoding).[4]

Συσχετίζουμε μια δυαδική μεταβλητή u_i με κάθε λογική πρόταση c_i. Το πεδίο ορισμού D_{ui} περιέχει εκείνες τις πλειάδες τιμών αληθείας οι οποίες ικανοποιούν την πρόταση c_i. Οι δυαδικοί περιορισμοί καθορίζονται ανάμεσα στις δυαδικές μεταβλητές οι οποίες συσχετίζονται με λογικές προτάσεις που έχουν κοινές προτασιακές μεταβλητές.

Παράδειγμα 3.8

Συσχετίζουμε την πρόταση $x_1 \vee x_3$ με τη δυαδική μεταβλητή u_1 με πεδίο ορισμού {(T,F),(F,T),(T,T)} και την πρόταση $x_1 \vee \neg x_3$ με τη δυαδική μεταβλητή u_2 με πεδίο ορισμού {(T,T),(F,F),(T,F)} .Αυτές είναι οι αναθέσεις x_1 και x_3 που ικανοποιούν τις προτάσεις $x_1 \vee x_3$ και $x_1 \vee \neg x_3$. Μεταξύ των δυαδικών μεταβλητών u_1 και u_2 που σχετίζονται με τις προτάσεις $x_1 \vee x_3$ και $x_1 \vee \neg x_3$ αντίστοιχα υπάρχει ο δυαδικός περιορισμός ότι το δεύτερο στοιχείο της πλειάδας που συσχετίζεται με τη μεταβλητή u_1 πρέπει να είναι το συμπλήρωμα του δεύτερου στοιχείου της πλειάδας που συσχετίζεται με τη μεταβλητή u_2.

Ορισμός 3.12: Μετάφραση της κρυμμένης μεταβλητής (Hidden variable encoding)[4]

Συσχετίζουμε μια δυαδική μεταβλητή u_i με κάθε λογική πρόταση c_i. Το πεδίο ορισμού της u_i περιέχει εκείνες τις πλειάδες τιμών αληθείας οι οποίες ικανοποιούν την πρόταση c_i.Επίσης έχουμε προτασιακές μεταβλητές x_i με πεδία ορισμού {T,F}. Ένας δυαδικός περιορισμός ορίζεται μεταξύ μιας δυαδικής μεταβλητής και μιας προτασιακής μεταβλητής, αν η πρόταση που συσχετίζεται με τη δυαδική μεταβλητή περιέχει την προτασιακή μεταβλητή. Δηλαδή δεν υπάρχουν ευθείς περιορισμοί μεταξύ των δυαδικών μεταβλητών.

Παράδειγμα 3.9

Συσχετίζουμε την πρόταση $x_1 \vee x_3$ με τη δυαδική μεταβλητή u_1 με πεδίο ορισμού {(T,F),(F,T),(T,T)} και την πρόταση $x_1 \vee \neg x_3$ με τη δυαδική μεταβλητή u_2 με πεδίο ορισμού {(F,F),(T,T),(T,F)}.Αυτές είναι οι αναθέσεις x_1 και x_3 που ικανοποιούν τις προτάσεις $x_1 \vee x_3$ και $x_1 \vee \neg x_3$. Επίσης ορίζουμε τις μεταβλητές x_1, x_3 με πεδίο ορισμού {T,F}

Μεταξύ της δυαδικής μεταβλητής u_2,η οποία συσχετίζεται με την πρόταση $x_1 \vee \neg x_3$ και την προτασιακή μεταβλητή x_3, υπάρχει ένας δυαδικός περιορισμός. Ο περιορισμός αυτός περιορίζει το δεύτερο στοιχειό της πλειάδας που ανατίθεται στη μεταβλητή u_2 να είναι το συμπλήρωμα της τιμής που ανατίθεται στην προτασιακή μεταβλητή x_3.

Ορισμός 3.13: Μετάφραση προτασιακής μεταβλητής (Literal encoding)[4]

Συσχετίζουμε μια μεταβλητή u_i με κάθε λογική πρόταση c_i. Το πεδίο ορισμού της μεταβλητής u_i αποτελείται από τις προτασιακές μεταβλητές που ικανοποιούν την πρόταση c_i. Σχηματίζουμε δυαδικούς περιορισμούς μεταξύ των μεταβλητών u_i και u_j, εάν και μόνο εάν η συσχετιζόμενη με τη μεταβλητή u_i πρόταση c_i περιέχει μια προτασιακή μεταβλητή της οποίας το συμπλήρωμα περιέχεται στη συσχετιζόμενη με τη μεταβλητή u_j πρόταση c_j. Αυτοί οι περιορισμοί αποκλείουν τις αναθέσεις που είναι λογικές αντίφασης.

Παράδειγμα 3.11

Συσχετίζουμε την πρόταση $x_1 \vee x_3$ με τη δυαδική μεταβλητή u_1 με πεδίο ορισμού $\{x_1, x_3\}$ και την πρόταση $x_2 \vee - x_3$ με τη δυαδική μεταβλητή u_2 με πεδίο ορισμού $\{x_2, - x_3\}$.

Μεταξύ των μεταβλητών u_1 και u_2 υπάρχει ο περιορισμός που επιτρέπει την ανάθεση $u_1 = x_1$ και $u_2 = x_2$, την ανάθεση $u_1 = x_1$ και $u_2 = - x_3$ ή την ανάθεση $u_1 = x_3$ και $u_2 = - x_2$.Ομως η ανάθεση $u_1 = x_3$ και $u_2 = - x_3$ είναι μια λογική αντίφαση για αυτό θα πρέπει να αποκλειστεί.

Συμπέρασμα 3.4[4]

Παρατηρούμε ότι η μετάφραση λογικής μεταβλητής χρησιμοποιεί μεταβλητές με μικρότερα πεδία ορισμού σε σχέση με τη μετάφραση κρυμμένης μεταβλητής και τη δυαδική μετάφραση. Οι δυαδικές μεταβλητές έχουν πεδία ορισμού μεγέθους $O(2^k)$,όπου το k είναι το μέγεθος της λογικής πρότασης, ενώ οι μεταβλητές που χρησιμοποιούνται στη μετάφραση λογικής μεταβλητής έχουν πεδία ορισμού μεγέθους O(κ).Αυτό το γεγονός έχει σημαντική διαφορά στους χρόνους εκτέλεσης.

Ορισμός 3.14: Μη δυαδική μετάφραση (Non binary encoding)[4]

Ένα πρόβλημα ικανοποίησης περιορισμών έχει μεταβλητές x_i με πεδία ορισμού {T,F}. Ένας μη δυαδικός περιορισμός καθορίζεται μεταξύ των μεταβλητών οι οποίες παρουσιάζονται μαζί σε μια λογική πρόταση. Αυτός ο περιορισμός αποκλείει εκείνες τις μερικές αναθέσεις που αποτυγχάνουν να ικανοποιήσουν τη λογική πρόταση.

Παράδειγμα 3.10

Έστω η λογική πρόταση $x_1 \vee x_2 \vee \neg x_3$. Δημιουργούμε τις δυαδικές μεταβλητές x_1, x_2 και x_3 με πεδία ορισμού {T,F}.Ο μη δυαδικός περιορισμός που συσχετίζεται με τις μεταβλητές x_1, x_2 και x_3 αποκλείει την ανάθεση x_1=F, x_2=F και x_3=T, διότι αυτή η ανάθεση στις μεταβλητές δεν ικανοποιεί τη λογική πρόταση.

Συμπέρασμα 3.5

Μπορούμε να χρησιμοποιήσουμε τεχνικές συνέπειας για να λύσουμε ένα πρόβλημα ικανοποιησιμότητας, αφού αρχικά το μεταφράσουμε σε ένα πρόβλημα ικανοποίησης περιορισμών.

3.5 Μετάφραση του προβλήματος ικανοποίησης περιορισμών σε πρόβλημα Ικανοποιησιμότητας.

Υπάρχουν διάφοροι τρόποι με τους οποίους ένα πρόβλημα ικανοποίησης περιορισμών μπορεί να μεταφραστεί σε πρόβλημα ικανοποιησιμότητας. Αυτές είναι η ευθεία μετάφραση, η λογαριθμική μετάφραση και η μετάφραση συνέπειας ακμής.

Ορισμός 3.15:Ευθεία μετάφραση (Direct Encoding)[8]

Η ευθεία μετάφραση είναι η μετάφραση που χρησιμοποιείται συχνότερα από τις υπόλοιπες. Υπάρχει μια δυαδική μεταβλητή x_u για κάθε τιμή u της κάθε μεταβλητής x του προβλήματος ικανοποίησης περιορισμών. x_u = T σημαίνει ότι η τιμή u ανατίθεται στη μεταβλητή x. Αυτές οι μεταβλητές εμφανίζονται σε δύο σύνολα λογικών προτάσεων:

α) Τουλάχιστο μια πρόταση: Υπάρχει μια τέτοια πρόταση για κάθε μεταβλητή και η σημασία τους είναι ότι μια τιμή από το πεδίο ορισμού πρέπει να δοθεί σε αυτή τη μεταβλητή. Έστω x μια μεταβλητή με πεδίο ορισμού $D_x = \{u_1, u_2, \ldots, u_n\}$, τότε προσθέτουμε τουλάχιστον μια πρόταση της μορφής $x_{u1} \vee x_{u2} \vee \ldots \vee x_{un}$.

β)Πρόταση αντίφασης: Υπάρχει μια τέτοια πρόταση για κάθε περιορισμό και η σημασία τους είναι ότι αυτή η πλειάδα τιμών απαγορεύεται. Έστω C_{xyz} ο περιορισμός στις μεταβλητές x,y,z και [u, v, w] ανήκει στο καρτεσιανό γινόμενο $D_x \times D_y \times D_z$, και η πλειάδα [u, v, w] απαγορεύεται από τον περιορισμό C_{xyz},τότε προσθέτουμε την πρόταση αντίφασης $\neg x_u \vee \neg x_v \vee \neg x_w$.

Παράδειγμα 3.11

Έστω το πρόβλημα ικανοποίησης περιορισμών CSP(X,D,C) το οποίο έχει τις μεταβλητές a, b και c με πεδία ορισμού {1,2,3}. Έστω ότι ισχύουν οι περιορισμοί a < b , b < c και c < a. Η ευθεία μετάφραση αυτού του προβλήματος ικανοποίησης περιορισμών σε πρόβλημα προτασιακής ικανοποιησιμότητας παράγει τις ακόλουθες λογικές προτάσεις:

Τουλάχιστον μια πρόταση:

$x_{\alpha 1} \vee x_{\alpha 2} \vee x_{\alpha 3}$ $\quad$ $x_{b1} \vee x_{b2} \vee x_{b3}$ $\quad$ $x_{c1} \vee x_{c2} \vee x_{c3}$

Παραδείγματος χάριν η πρόταση $x_{b1} \vee x_{b2} \vee x_{b3}$, εκφράζει το γεγονός στη μεταβλητή x ανατίθεται τουλάχιστον μια τιμή από το πεδίο ορισμού της {1,2,3}.

Προτάσεις Αντίφασης:

$\neg x_{\alpha 1} \vee \neg x_{b1}$	$\neg x_{b1} \vee \neg x_{c1}$	$\neg x_{c1} \vee \neg x_{a1}$
$\neg x_{\alpha 2} \vee \neg x_{b1}$	$\neg x_{b2} \vee \neg x_{c1}$	$\neg x_{c2} \vee \neg x_{a1}$
$\neg x_{\alpha 3} \vee \neg x_{b3}$	$\neg x_{b3} \vee \neg x_{c3}$	$\neg x_{c3} \vee \neg x_{a3}$
$\neg x_{\alpha 3} \vee \neg x_{b2}$	$\neg x_{b3} \vee \neg x_{c2}$	$\neg x_{c3} \vee \neg x_{a2}$
$\neg x_{\alpha 3} \vee \neg x_{b1}$	$\neg x_{b3} \vee \neg x_{c1}$	$\neg x_{c3} \vee \neg x_{a1}$

Παραδείγματος χάριν η πρόταση $\neg x_{\alpha 1} \vee \neg x_{b1}$ αναφέρεται στον περιορισμό a < b και εκφράζει το γεγονός ότι η ανάθεση a = 1 και b = 1, δεν επιτρέπεται επειδή παραβιάζει τον περιορισμό a < b.

Θεώρημα 3.1:[8]

Η εφαρμογή συνέπειας ακμής στο αρχικό πρόβλημα είναι πιο αποτελεσματική από την εφαρμογή του κανόνα μοναδιαίας απαλοιφής, μετά την ευθέως μετάφραση του αρχικού προβλήματος ικανοποίησης περιορισμών σε πρόβλημα ικανοποιησιμότητας.

Ορισμός 3.16: Μετάφραση συνέπειας ακμής (AC enconding)[8]

Η μετάφραση συνέπειας ακμής επιτρέπει στο μεταφρασμένο πρόβλημα ικανοποιησιμότητας να έχει συνέπεια ακμής μέσω της μοναδιαίας απαλοιφής. Δεν μεταφράζει μόνο τη δομή του προβλήματος ικανοποίησης περιορισμών. Μέσω της μετάφρασης υλοποιεί και ένα αλγόριθμο συνέπειας ο οποίος χρησιμοποιείται για να λύσει το πρόβλημα. Διαφέρει από την ευθεία μετάφραση στο γεγονός ότι προσθέτουμε ακόμα ένα σύνολο λογικών προτάσεων:

Το πολύ μια πρόταση: Υπάρχει μια τέτοια πρόταση για κάθε ζεύγος τιμών της κάθε μεταβλητής και η σημασία τους είναι ότι αυτή η μεταβλητή δεν μπορεί να πάρει περισσότερες από μία τιμή. Έστω οι τιμές u_i και u_j οι οποίες ανήκουν στο πεδίο ορισμού D_x και i ≠ j, τότε προσθέτουμε το πολύ μια πρόταση της μορφής ¬ x_{ui} ∨ ¬ x_{uj}.

Επίσης οι προτάσεις αντίφασης της ευθείας μετάφρασης, αντικαθιστούνται από τις προτάσεις στήριξης. Μια πρόταση στήριξης ορίζεται ως εξής:

Έστω x και y δύο μεταβλητές του προβλήματος ικανοποίησης περιορισμών A(X,D,C).Η τιμή u ανήκει στο πεδίο ορισμού D_x και το σύνολο { $w_1, \ldots, w_k$} είναι οι τιμές που στηρίζουν την ανάθεση x = u στο πεδίο ορισμού D_y. Τότε υπάρχει η λογική πρόταση στήριξης ¬ x_u ∨ y_{w1} ∨ y_{w2} ∨ . . . ∨ y_{wk}. Αυτή η λογική πρόταση είναι λογικά ισοδύναμη με την λογική πρόταση $x_u \Rightarrow (y_{w1} \vee y_{w2} \vee \ldots \vee y_{wk})$ η οποία σημαίνει εφόσον η x_u είναι αληθής, δηλαδή η τιμή u παραμένει στο πεδίο ορισμού D_x, τότε τουλάχιστον μία δυαδική μεταβλητή y_{wi} θα πρέπει να είναι αληθής. Γι αυτό όταν όλες οι μεταβλητές y_{wi} πάρουν την τιμή ψευδής, τότε και η μεταβλητή x_u θα πρέπει να πάρει την τιμή ψευδής.

Παράδειγμα 3.12

Έστω το πρόβλημα ικανοποίησης περιορισμών CSP(X,D,C) το οποίο έχει τις μεταβλητές a, b και c με πεδία ορισμού {1,2,3}. Έστω ότι ισχύουν οι περιορισμοί a < b , b < c και c < a. Η μετάφραση συνέπειας ακμής αυτού του προβλήματος ικανοποίησης περιορισμών σε πρόβλημα προτασιακής ικανοποιησιμότητας παράγει τις ακόλουθες λογικές προτάσεις:

Τουλάχιστον μια πρόταση:

$x_{\alpha 1} \vee x_{\alpha 2} \vee x_{\alpha 3}$ $\quad$ $x_{b1} \vee x_{b2} \vee x_{b3}$ $\quad$ $x_{c1} \vee x_{c2} \vee x_{c3}$

Παραδείγματος χάριν η πρόταση $x_{b1} \vee x_{b2} \vee x_{b3}$, εκφράζει το γεγονός στη μεταβλητή x ανατίθεται τουλάχιστον μια τιμή από το πεδίο ορισμού της {1,2,3}.

Προτάσεις στήριξης:

$\neg x_{\alpha 3}$	$\neg x_{b3}$	$\neg x_{c3}$
$x_{b3} \vee \neg x_{\alpha 2}$	$x_{c3} \vee \neg x_{b2}$	$x_{\alpha 3} \vee \neg x_{c2}$
$x_{b2} \vee x_{b3} \vee \neg x_{\alpha 1}$	$x_{c2} \vee x_{c3} \vee \neg x_{b1}$	$x_{\alpha 2} \vee x_{\alpha 3} \vee \neg x_{c1}$
$\neg x_{b3}$	$\neg x_{c1}$	$\neg x_{\alpha 1}$
$x_{\alpha 1} \vee \neg x_{b2}$	$x_{b1} \vee \neg x_{c2}$	$x_{c1} \vee \neg x_{\alpha 2}$
$x_{\alpha 2} \vee x_{\alpha 1} \vee \neg x_{b3}$	$x_{b2} \vee x_{b1} \vee \neg x_{c3}$	$x_{c2} \vee x_{c1} \vee \neg x_{\alpha 3}$

Παραδείγματος χάριν η πρόταση $x_{c3} \vee \neg x_{b2}$ μεταφράζει τη στήριξη b = 2 στον περιορισμό b < c. Επειδή ισχύει b < c και c ≤ 3, η μόνη στήριξη για την ανάθεση b = 2 είναι η ανάθεση c=3. Η πρόταση αυτή εκφράζει ότι είτε το c=3 είτε το b ≠ 2.

Το πολύ μια πρόταση:

$\neg x_{\alpha 1} \vee \neg x_{\alpha 2}$	$\neg x_{b1} \vee \neg x_{b2}$	$\neg x_{c1} \vee \neg x_{c2}$
$\neg x_{\alpha 2} \vee \neg x_{\alpha 3}$	$\neg x_{b2} \vee \neg x_{b3}$	$\neg x_{c2} \vee \neg x_{c3}$
$\neg x_{\alpha 3} \vee \neg x_{\alpha 1}$	$\neg x_{b3} \vee \neg x_{b1}$	$\neg x_{c3} \vee \neg x_{c1}$

Παραδείγματος χάριν οι προτάσεις $\neg x_{c1} \vee \neg x_{c2}$, $\neg x_{c2} \vee \neg x_{c3}$ και $\neg x_{c3} \vee \neg x_{c1}$ δηλώνουν ότι στη μεταβλητή c μπορεί να ανατεθεί το πολύ μια τιμή από το πεδίο ορισμού της {1,2,3}.

3.6 Ο αλγόριθμος Davis Putman Logemann-Loveland(DPLL) για εύρεση λύσης προβλημάτων προτασιακής ικανοποιησιμότητας.

Ο αλγόριθμος DPLL[5] καθορίζει εάν μια λογική πρόταση σε κανονική συζευκτική μορφή είναι ικανοποιήσιμη. Μια λογική πρόταση είναι ικανοποιήσιμη εάν υπάρχει κάποια ανάθεση στις μεταβλητές λογικής που έχει ως αποτέλεσμα η πρόταση να είναι αληθής .

Παράδειγμα 3.13

Έστω η σύνθετη πρόταση Q:

(a ∨ ¬b ∨ c) ∧ (¬a ∨ b ∨ d) ∧ (a ∨ b ∨ ¬d) ∧ (¬a ∨ ¬d ∨ ¬d)

Στο παράδειγμα 3.15, υπάρχουν τέσσερις προτάσεις. Για να είναι η σύνθετη πρόταση ικανοποιήσιμη θα πρέπει οι λογικές προτάσεις που την απαρτίζουν να είναι ικανοποιήσιμες. Ο αλγόριθμος DPLL επιλέγει τιμές για μια δυαδική προτασιακή μεταβλητή κάθε φορά. Μετά από κάθε ανάθεση, απλοποιεί το σύνολο προτάσεων που πρέπει να ικανοποιηθούν. Μετά από κάθε απλοποίηση, ελέγχει για να δει εάν :

- Όλες οι προτάσεις ικανοποιούνται, οπότε επιστρέφει αληθές.
- Κάποια πρόταση δεν ικανοποιείται. Εάν αυτό συμβαίνει, ο αλγόριθμος DPLL οπισθοδρομεί και δοκιμάζει μια διαφορετική τιμή για κάποια μεταβλητή που δεν της έχει δοθεί τιμή ακόμα.

Εδώ είναι ο ψευδοκώδικας για ένα αναδρομικό αλγόριθμο που το κάνει αυτό:

function DPLL(*clauses, assignment*)

1. **If** all clauses are satisfiable, **then return** *true*.

2. **If** some clause is not satisfiable, **then return** *false.*

3. Simplify clauses using forward checking.

4. u ← choose_variable(*clauses, assignment*).

5. **Return** DPLL(simplify(*clauses, assignment, u = true*)) **or**
DPLL(simplify(*clauses, assignment, u = false*))

Ένας επαναλαμβανόμενος αλγόριθμος DPLL είναι μια καλή ιδέα επειδή, εάν οι μεταβλητές καλούνται δια τιμής τότε η οπισθοδρόμηση είναι εύκολη. Εάν μια κλήση επιστρέφει ανεπιτυχώς, οι παλαιές τιμές των δομών δεδομένων αποκαθίστανται αυτόματα. Πώς ο αλγόριθμος DPLL απλοποιεί τις προτάσεις. Ο DPLL παρακολουθεί ένα σύνολο προτάσεων που πρέπει ακόμα να ικανοποιηθούν; Για κάθε μια από αυτές τις προτάσεις, παρακολουθεί τις μεταβλητές λογικής που δεν τους έχει δοθεί ακόμα τιμή, και εάν είναι αληθείς, θα ικανοποιήσουν την πρόταση. Κατά συνέπεια, υπάρχουν δύο τρόποι που ο DPLL μπορεί να απλοποιήσει το σύνολο προτάσεών του.

- Για κάθε πρόταση, εάν έχει δοθεί τιμή σε μια από τις μεταβλητές της έτσι ώστε η πρόταση είναι αληθής, κατόπιν η πρόταση μπορεί να διαγραφεί από την λίστα προτάσεων που πρέπει να ικανοποιηθούν.

- Εάν σε μια από τις μεταβλητές λογικής μιας πρότασης έχει δοθεί η τιμή ψευδής, κατόπιν η πρόταση μπορεί να απλοποιηθεί για να είναι μια κοντύτερη πρόταση με ακριβώς τις μεταβλητές λογικής που έχουν απομείνει.

Παράδειγμα 3.14

Υποθέτουμε ότι θέτουμε στη λογική μεταβλητή d = αληθής στη λογική πρόταση Q = (a ∨ ¬b ∨ c) ∧ (¬a ∨ b ∨ d) ∧ (a ∨ b ∨ ¬d) ∧ (¬a ∨ ¬d ∨ ¬d) .Μπορούμε έπειτα να αφαιρέσουμε όλες τις προτάσεις που περιέχουν την μεταβλητή d. Αυτό απλοποιεί την πρόταση Q στην λογική πρόταση P= (a ∨ ¬b ∨ c) ∧ (a ∨ b ∨ ¬d) ∧ (¬a ∨ ¬d ∨ ¬d) Μπορούμε επίσης να αφαιρέσουμε τη μεταβλητή λογικής ¬d από όλες τις προτάσεις. Αυτό απλοποιεί την πρόταση P στην πρόταση R= (a ∨ ¬b ∨ c) ∨ (a ∨ b) ∧ (¬a).

Πιο κάτω είναι ο ψευδοκώδικας για την απλούστευση των προτάσεων:

Function simplify(clauses,assignments, u= v)

1. for each clause i satisfied by u = v, remove_clause (clauses, i)
2. for each clause i satisfied by ¬u = v, remove_literal (clauses[i], ¬u)
3. assignment[u] ⟵v
4. **return** clauses, assignment

Στον πιο πάνω ψευδοκώδικα η μεταβλητή v έχει πεδίο ορισμού {true,false}. Επίσης η μεταβλητή u αναφέρεται σε μια προτασιακή μεταβλητή του προβλήματος ικανοποιησιμότητας. Η ανάθεση u = v, δηλώνει ότι θέτουμε στη μεταβλητή u την τιμή true ή την τιμή false. Αν θέσουμε την τιμή true τότε αφαιρούμε όλες τις προτάσεις που περιέχουν τη μεταβλητή u και αφαιρούμε τη μεταβλητή ¬u από όλες τις προτάσεις που έχουν απομείνει. Αν θέσουμε την τιμή false τότε αφαιρούμε όλες τις προτάσεις που περιέχουν τη μεταβλητή ¬u και αφαιρούμε τη μεταβλητή u από όλες τις προτάσεις που έχουν απομείνει.

Λαμβάνοντας υπόψη αυτές τις απλοποιήσεις:

• Όλες οι προτάσεις είναι ικανοποιήσιμες εάν το σύνολο προτάσεων που πρέπει ακόμα να ικανοποιηθούν είναι κενό.

• Κάποια πρόταση δεν είναι ικανοποιήσιμη εάν το σύνολο προτάσεων περιέχει μια κενή πρόταση.

Μπορούμε επομένως να ενημερώσουμε τον ψευδοκώδικα DPLL μας:

Function DPLL(clauses,assignment)

1. **If** the set of clauses is empty, then return true.
2. **If** some clause in the set of clauses is empty, then return false.
3. Simplify clauses using forward checking.
4. u ⟵ choose_variable(clauses,assignment).
5. **Return** DPLL(simplify(clauses,assignment,u=true)) or DPLL(simplify(clauses,assignment,u=false)).

Ο DPLL απλοποιεί περαιτέρω τις προτάσεις μέσω της χρησιμοποίησης του αλγορίθμου εμπρόσθιος έλεγχος (forward checking), ο όποιος είναι αντίστοιχος με την μοναδιαία απαλοιφή. Αφού ο αλγόριθμος DPPL αναθέσει τιμή σε μια μεταβλητή, ελέγχει αν υπάρχουν μοναδιαίες προτάσεις που δεν έχουν ακόμα ικανοποιηθεί. Όταν έχουμε μια μοναδιαία πρόταση, η σωστή τιμή της μεταβλητής σε αυτή είναι προφανής. Μπορούμε να ορίσουμε σε αυτήν την μεταβλητή τη σωστή αυτή τιμή, και απλοποιούμε τις προτάσεις.

Παράδειγμα 3.15

Ας προσπαθήσουμε να εφαρμόσουμε μοναδιαία απαλοιφή στη σύνθετη λογική πρόταση R = (a ∨ ¬b ∨ c) ∨ (a ∨ b) ∧ (¬a). Η τελευταία πρόταση της R είναι μια μοναδιαία πρόταση. Έτσι μπορούμε να θέσουμε την τιμή α = ψευδής. Πρέπει έπειτα να απλοποιήσουμε την πρόταση R. Αφαιρούμε όλες τις προτάσεις που περιέχουν τη μεταβλητή λογικής ¬a, οπότε έχουμε την πρόταση B= (a ∨ ¬b ∨ c) ∧ (a ∨ b). Αφαιρούμε έπειτα τη λογική μεταβλητή a από όλες τις προτάσεις, με συνέπεια την πρόταση.A= (b) ∧ (¬b ∨ c). Τώρα υπάρχει μια άλλη μοναδιαία πρόταση. Θέτουμε b = αληθής, και απλοποιούμε τον τύπο, με κατάληξη σε (c). Έχουμε τώρα μια άλλη πρόταση μονάδων. Θέτουμε c = αληθής, και απλοποιούμε τη πρόταση, με συνέπεια ένα κενό σύνολο προτάσεων. Άρα συμπεραίνουμε ότι η αρχική πρόταση Q είναι ικανοποιήσιμη.

Πιο κάτω είναι ο ψευδοκώδικας της μοναδιαίας απαλοιφής:

Function unit_propagation(clauses,assignment)

1. **For each** clause i in the set of clauses,
   If clause i is a unit clause satisfied by u = v, then
   Clauses,assigment ⟵simplify(clauses,assignment,u=v)
   Return unit_propagation(clauses,assignment)
2. **Return** clauses,assignment

Προσθέτοντας το βήμα της μοναδιαίας απαλοιφής , παίρνουμε τον ακόλουθο αλγόριθμο:

Function DPPL(clauses,assignment)

1. **If** the set of clauses is empty, then return true.
2. **If** some clause in the set of clauses is empty, then return false.
3. **If** some clause i in the set of clauses is a unit clause, then
   Return DPLL (unit_propagation (clauses, assignment, i)).
4. u ⟵ choose_variable(clauses,assignment).
5. **Return** DPPL (simplify (clauses,assignment, u=true)) **or**
   DPLL (simplify (clauses, assignment, u = false)).

Η σύνθετη πρόταση μπορεί να περιέχει επίσης πολλές δυαδικές λογικές προτάσεις και είναι δυνατό να γίνουν τα διάφορα είδη επιλύσεων της σύνθετης πρότασης εισαγωγής λαμβάνοντας υπόψη και αυτές τις προτάσεις. Όλες αυτές οι επιλύσεις γίνονται από κοινού με τη μοναδιαία απαλοιφή. Αρχικά ο αλγόριθμος θα πρέπει να επιλύσει όλα τα πιθανά ζευγάρια των δυαδικών προτάσεων. Τέτοιες μειώσεις παράγουν μόνο νέες δυαδικές προτάσεις ή νέες προτάσεις μονάδων.

Χρησιμοποιώντας τη δυαδική απαλοιφή στον αλγόριθμο DPPL θα έχουμε

α. προσθέτουμε στη σύνθετη πρόταση όλες τις νέες δυαδικές ή μοναδιαίες προτάσεις μέσο της επίλυσης των ζευγαριών των δυαδικών προτάσεων,

β. εφαρμόζουμε μοναδιαία απαλοιφή σε οποιεσδήποτε νέες μοναδιαίες προτάσεις εμφανίζονται (το οποίο στη συνέχεια μπορεί να παράγει περισσότερες δυαδικές προτάσεις προκαλώντας μια άλλη επανάληψη του (α)), μέχρι

1. να υπάρξει αντίφαση , ή

2. τίποτα νέο δεν μπορεί να προστεθεί στη σύνθετη πρόταση από τα βήματα α. ή β.

Παράδειγμα 3.16

Εφαρμόζουμε τον πιο πάνω αλγόριθμο στο τύπο Q= (a ∨ b) ∧ (¬a ∨ c) ∧ (¬b ∨ c),οπότε παίρνουμε τις δυαδικές προτάσεις (b ∨ c) , (a ∧ c) και την μοναδιαία πρόταση c.Μετά εφαρμόζεται κανονικά η μοναδιαία απαλοιφή.

Ορισμός 3.17: Γενικευμένη δυαδική απαλοιφή (Hyper binary resolution)[5]

Η γενικευμένη δυαδική απαλοιφή είναι ένας λογικός κανόνας συμπεράσματος, ο οποίος περιλαμβάνει ένα βήμα γενικής απαλοιφής, δηλαδή ένα βήμα συμπεράσματος που περιλαμβάνει περισσότερες από δύο προτάσεις εισόδου. Παίρνει ως εισαγωγή μια ενιαία n-αδική πρόταση, της μορφής (l_1, l_2, l_3, . . . , l_n), όπου (n >=2) και n - 1 δυαδικές προτάσεις της μορφής (¬l_i, l) ,όπου(i = 1, . . ., n - 1). Παράγει ως έξοδο τη νέα δυαδική πρόταση (l, l_n).

Παράδειγμα 3.17

Χρησιμοποιώντας τη γενικευμένη δυαδική απαλοιφή στη σύνθετη πρόταση Q = (a ∨ b ∨ c ∨ d) ∧ (h ∨ ¬a) ∧ (h ∨ ¬c) ∧ (h ∨¬d), θα έχουμε τη νέα δυαδική πρόταση (h ∨ b).

Η γενικευμένη δυαδική απαλοιφή είναι ισοδύναμη με μια σειρά συνηθισμένων δυαδικών βημάτων συμπεράσματος, δηλαδή βημάτων συμπεράσματος που περιλαμβάνουν μόνο δύο προτάσεις. Εντούτοις, μια τέτοια ακολουθία θα παρήγε και τις προτάσεις ενδιάμεσου μήκους ενώ το η γενίκευση δυαδικής απαλοιφή το παραβλέπει αυτό, παράγοντας μόνο την τελική δυαδική πρόταση. Σε ένα αλγόριθμο επίλυσης προβλημάτων προτασιακής ικανοποιησιμότητας, όπως είναι ο αλγόριθμος DPLL είναι γενικά αντιπαραγωγικό να προστεθούν όλες αυτές οι ενδιάμεσες προτάσεις στη βάση γνώσεων. Θα ήταν πιο χρήσιμο να προστεθεί μόνο η τελική δυαδική πρόταση. Πρέπει επίσης να σημειωθεί ότι εάν η n-αδική σύνθετη πρόταση εισαγωγής είναι δυαδική, η γενικευμένη δυαδική απαλοιφή είναι η ίδια με την απλή δυαδική απαλοιφή των προτάσεων, όπως φαίνεται στο πιο κάτω παράδεγμα.

Παράδειγμα 3.18

Η γενικευμένη δυαδική απαλοιφή στην n- αδική πρόταση P = (a ∨ b) ∧ (h ∨ ¬a) παράγει τη νέα δυαδική πρόταση (h ∨ b).

Η γενικευμένη δυαδική απαλοιφής μπορεί επίσης να χρησιμοποιηθεί για να παραγάγει και μοναδιαίες προτάσεις, εάν της επιτρέψουμε να εξετάσει μια ακόμα δυαδική πρόταση, όπως φαίνεται στο πιο κάτω παράδειγμα.

Παράδειγμα 3.19

Λαμβάνοντας υπόψη τη σύνθετη πρόταση
Q = (a ∨ b ∨ c ∨ d) ∧ (h ∨ ¬a) ∧ (h ∨ ¬b) ∧ (h ∨ ¬c) ∧ (h ∨ ¬d) , συμπεραίνουμε τη μοναδιαία πρόταση (h).

Συμπέρασμα 3.6

'Αρα στον αλγόριθμο DPLL μπορούμε να εφαρμόσουμε γενικευμένη δυαδική απαλοιφή όπως διευκρινίζεται πιο πάνω και έπειτα ένα χωριστό ενιαίο βήμα του συνηθισμένου συμπεράσματος των δυαδικών προτάσεων. Στο παράδειγμά 3.21, η γενικευμένη δυαδική απαλοιφή, χρησιμοποιώντας μόνο τις πρώτες τρεις δυαδικές προτάσεις, θα παρήγε την πρόταση (h ∨ d), κατόπιν ένα συνηθισμένο βήμα συμπεράσματος με την πρόταση (h ∨ ¬d) παράγει την πρόταση (h).

Ορισμός 3.18: Απαλοιφή ισότητας (Equality resolution) [5]

Μόλις οι δυαδικές προτάσεις είναι διαθέσιμες μπορούμε να εκτελέσουμε το λογικό κανόνα συμπεράσματος απαλοιφή ισότητας. Εάν βάση γνώσεων περιέχει τη σύνθετη πρόταση Q = (¬a ∨ b) ∧ (a ∨ ¬b) μπορούμε να παραγάγουμε μια νέα πρόταση. Η απαλοιφή ισότητας αντικαθιστά όλα τις μεταβλητές λογικής b στη σύνθετη πρόταση Q με τις μεταβλητές λογικής α, ακόλουθος αφαιρεί όλες τις προτάσεις που περιέχουν τώρα και το α και ¬α, και τέλος αφαιρεί όλες τις διπλές περιπτώσεις a (ή ¬a) από όλες τις προτάσεις. Αυτή η διαδικασία παραγάγει νέες δυαδικές προτάσεις, όπως φαίνεται στο παράδειγμα 4.7.

Παράδειγμα 3.20

Εφαρμόζοντας την απαλοιφή ισότητας στην πρόταση
Q= (a ∨ ¬b) ∧ (¬a ∨ b) ∧ (a ∨ ¬b ∨ c) ∧ (b ∨ ¬ d) ∧ (α ∨ b ∨ d) θα έχουμε ως αποτέλεσμα τη λογική πρόταση P = (a ∨ ¬d) ∧ (a ∨ d).

Συμπέρασμα 3.7[5]

Μπορούμε να εφαρμόσουμε μοναδιαία απαλοιφή , γενικευμένη δυαδική απαλοιφή και απαλοιφή ισότητας στη βάση γνώσεων μέχρι να μην μπορούν να εφαρμοστούν ξανά. Μια βάση γνώσεων στην οποία αυτοί οι τρεις κανόνες συμπεράσματος δεν μπορούν να συμπεράνουν τίποτα νέο καλείται κλειστότητα ως προς γενίκευμένη δυαδικής απαλοιφής και απαλοιφή ισότητας.

Θεώρημα 3.2[5]

Η κλειστότητα ως προς γενικευμένη δυαδική απαλοιφή και απαλοιφή ισότητας μιας βάσης γνώσεων που είναι γραμμένη σε κανονική συζευκτική μορφή είναι μοναδική. Δηλαδή η σειρά στην οποία οι κανόνες συμπεράσματος εφαρμόζονται είναι άσχετη, εφ' όσον συνεχίζουμε έως ότου δεν μπορούμε να τους εφαρμόσουμε άλλο.

Η πρακτική σημασία του θεωρήματος 3.2 είναι ότι είμαστε ελεύθεροι να εφαρμόσουμε αυτούς τους κανόνες συμπεράσματος με οποιαδήποτε σειρά και θα φθάσουμε στο ίδιο τελικό αποτέλεσμα.

Ορισμός 3.19: Η ανίχνευση μεταβλητής αποτυχίας (Failed literal detection) [5]

Η μοναδιαία απαλοιφή μπορεί επίσης να γίνει σε δοκιμαστική βάση. Δηλαδή μπορούμε να επιλέξουμε μια μεταβλητή λογικής και να θέσουμε σε αυτή την τιμή αληθής και μετά να εφαρμόσουμε τη μοναδιαία απαλοιφή. Αυτή η διαδικασία όπως έχουμε δει πιο πάνω, ονομάζεται **μοναδιαία μετάδοση(unit propagation)** και την αναπαριστούμε ως UP(a),όπου το a είναι η μεταβλητή στην οποία έχουμε αναθέσει αρχικά την τιμή αληθής. Όταν η Up(a) έχει ως αποτέλεσμα κάποιο άλλη μεταβλητή l να γίνει αληθής , χρησιμοποιούμε την αναπαράσταση Up(a) $\vdash l$. Εάν ισχύει Up(a) $\vdash l$ και Up(a) $\vdash \neg l$, έχουμε ανιχνεύσει ότι η a είναι μια μεταβλητή αποτυχίας και για αυτό τη θέτουμε να είναι ψευδής στη βάση γνώσεων . Μπορούμε έπειτα να μειώσουμε τη βάση γνώσεων εκτελώντας UP(¬a).Η πιο πάνω διαδικασία ονομάζεται ανίχνευση μεταβλητής αποτυχίας.

θεώρημα 3.3[5]

Η ανίχνευση μεταβλητής αποτυχίας είναι ισοδύναμη με τη γενικευμένη δυαδική απαλοιφή.

Συμπέρασμα 3.8[5]

Η κλειστότητα ως προς τη γενικευμένη δυαδική απαλοιφή και την απαλοιφή ισότητας περιλαμβάνει επανειλημμένη εφαρμογή της γενικευμένης δυαδικής απαλοιφής, της μοναδιαίας απαλοιφής και της απαλοιφής ισότητας. Το θεώρημα 3.4 δείχνει ότι μπορούμε να επιτύχουμε την κλειστότητα ως προς τη γενικευμένη δυαδική απαλοιφή και την απαλοιφή ισότητας εφαρμόζοντας επανειλημμένα τη μοναδιαία μετάδοση μιας μεταβλητής στις μεταβλητές της θεωρίας. Ακριβέστερα,

α) Μειώνουμε αρχικά τη θεωρία εκτελώντας τη μοναδιαία μετάδοση μιας μεταβλητής σε όλες τις μοναδιαίες προτάσεις .

β)Κατόπιν για κάθε literal l που απομένει εφαρμόζουμε UP(l), προσθέτοντας στη θεωρία μια νέα δυαδική πρόταση (¬l , a) για κάθε μεταβλητή a έτσι ώστε Up(l) ⊢ a .Ένα πέρασμα από όλες τις μεταβλητές εξασφαλίζει την εύρεση όλων των δυαδικών προτάσεων που μπορούν να προκύψουν από ένα βήμα γενικευμένης δυαδικής απαλοιφής. Η προσθήκη των συνεπαγόμενων δυαδικών προτάσεων εξασφαλίζει ότι το δεύτερο πέρασμα μπορεί να βρει όλες τις δυαδικές προτάσεις που συνεπάγονται αφού εφαρμόσουμε δύο βήματα γενικευμένης δυαδικής απαλοιφής. Πρέπει να προσθέσουμε τις συνεπαγόμενες δυαδικές προτάσεις που βρήκαμε στο πρώτο πέρασμα, αλλιώς η μοναδιαία μετάδοση μιας μεταβλητής δεν θα είναι αρκετά ισχυρή. Η προσθήκη αυτών των προτάσεων κάνει όλες τις εισαγωγές στο δεύτερο βήμα της γενικευμένης δυαδικής απαλοιφής να είναι διαθέσιμες στη βάση γνώσεων και επιτρέπει στη μοναδιαία μετάδοση μιας μεταβλητής να ακολουθήσει το δεύτερο βήμα της γενικευμένης δυαδικής απαλοιφής. Σε αυτή τη φάση έχουμε επιτύχει την κλειστότητα ως προς τη γενικευμένη δυαδική απαλοιφή και την απαλοιφή ισότητας αφού δεν υπάρχει καμιά κάποια άλλη περίπτωση εφαρμογής του κανόνα της γενικευμένης δυαδικής απαλοιφής.

γ)Η απαλοιφή ισότητας και η μοναδιαία απαλοιφή εφαρμόζονται τώρα για να υπολογίσουν την κλειστότητα ως προς τη γενικευμένη δυαδική απαλοιφή και την απαλοιφή ισότητας. Ένας προφανής τρόπος είναι η επαναληπτική διαδικασία μέχρι να επιτευχθεί η κλειστότητα ως προς τη γενικευμένη δυαδική απαλοιφή. Μετά εκτελούμε όλες τις απαλοιφές ισότητας, και τέλος επαναλαμβάνουμε αυτά τα βήματα έως ότου δεν εξαχθεί τίποτα καινούργιο.

Συμπέρασμα 3.9

Από το θεώρημα 3.4 αυτή η συγκεκριμένη ακολουθία διαδικασιών του συμπεράσματος 3.8 υπολογίζει την κλειστότητα ως προς τη γενικευμένη δυαδική απαλοιφή και την απαλοιφή ισότητας. Στην πράξη, εντούτοις, η ευελιξία που εξασφαλίζεται από το θεώρημα 3.4 είναι πολύ σημαντική για την αποδοτικότητα. Παραδείγματος χάριν, είναι πάντα καλή ιδέα να εκτελέσουμε μοναδιαία απαλοιφή οποτεδήποτε βρίσκουμε μια μοναδιαία πρόταση. Το θεώρημα 3.4 δηλώνει ότι δεν

μπορούμε να επιτύχουμε τίποτα μεγαλύτερο από τη γενικευμένη δυαδική απαλοιφή χρησιμοποιώντας πολλές εφαρμογές της μοναδιαίας μετάδοσης. Εντούτοις η διαδικασία που περιγράφηκε ακριβώς πιο πάνω υπολογίζει ακριβώς την κλειστότητα ως προς τη γενικευμένη δυαδική απαλοιφή και την απαλοιφή ισότητας.

Κεφάλαιο 4

Λογικά Προγράμματα και Σύνολα απαντήσεων

4.1 Η σημασιολογία του λογικού προγράμματος

Ορισμός 4.1: Τα θεμελιώδη συστατικά της προτασιακής λογικής

Τα θεμελιώδη συστατικά της προτασιακής λογικής είναι οι σταθερές αντικειμένων (object constants) του πεδίου ορισμού που μας ενδιαφέρει όπως είναι ο Γιώργος, ένα ποδήλατο ή ένα αυτοκίνητο. Επίσης στην προτασιακή λογική ορίζουμε τις σταθερές λειτουργιών (function constants) οι οποίες έχουν ως είσοδο κάποιο αντικείμενο και ως έξοδο ένα άλλο αντικείμενο. Παραδείγματος χάριν η λειτουργία πατέρας_του(Γιώργος) επιστρέφει τον Ανδρέα ο οποίος είναι ο πατέρας του Γιώργου αν φυσικά κάτι τέτοιο συνεπάγεται από τη βάση γνώσεων. Το τελευταίο θεμελιώδες συστατικό της προτασιακής λογικής είναι οι σταθερές κατηγορημάτων (predicate constants), οι οποίες δηλώνουν ιδιότητες των αντικειμένων. Δηλαδή έχουν ως είσοδο ένα αντικείμενο και επιστρέφουν την αλήθεια ή την αναλήθεια κάποιας ιδιότητας για αυτό το αντικείμενο. Παραδείγματος χάριν το κατηγόρημα είναι_κοντός(Γιώργος) θα επιστρέψει αληθές εάν το γεγονός ότι ο Γιώργος είναι κοντός συνεπάγεται από τη βάση γνώσεων ή θα επιστρέψει ψευδές εάν δεν συνεπάγεται το γεγονός ότι ο Γιώργος είναι κοντός από τη βάση γνώσεων. Οι σταθερές κατηγορημάτων χρησιμοποιούνται για να περιγράψουν τις σχέσεις μεταξύ των αντικειμένων.

Ορισμός 4.2: Άτομο στην προτασιακή λογική (Atom)

Άτομο στην προτασιακή λογική είναι μια βασική δομική μονάδα λογικής, η οποία αποτελείται από μια σταθερά κατηγορήματος η οποία έχει ως είσοδο ένα σύνολο από αντικείμενα. Παραδείγματος χάριν η σταθερά κατηγορήματος είναι_ συγγενείς(Γιώργος,Κώστας) είναι ένα άτομο.

Ορισμός 4.3:Προτασιακό κανονικό πρόγραμμα (logic program)[1]

Ένα προτασιακό κανονικό πρόγραμμα είναι ένα σύνολο κανονικών κανόνων της μορφής:

$A_0 \leftarrow A_1,\ldots,A_m$,not $A_{m+1},\ldots,$not A_n .

Όπου $n \geq m \geq 0$ και A_0, A_1 ,....., A_n είναι προτασιακά άτομα. Το άτομο A_0 λέγεται κεφαλή και το σύνολο των ατόμων $A_1,\ldots,A_m$,not $A_{m+1},\ldots,$not A_n λέγεται σώμα του κανόνα. Αν το σώμα είναι άδειο (n=0) δηλαδή ο κανόνας είναι της μορφής $A_0 \leftarrow$,τότε ο κανόνας αυτός ονομάζεται γεγονός και μπορούμε να το αναπαραστήσουμε εναλλακτικά ως A_0.

Εκτός από τους κανονικούς κανόνες, υποθέτουμε ότι η γλώσσα μας έχει και περιορισμούς, παραδείγματος χάριν κανόνες με μια κεφαλή, με άλλα λόγια, κανόνες της μορφής:

$\leftarrow A_1,\ldots,A_m$,not $A_{m+1},\ldots,$not A_n

Ένας τέτοιος κανόνας δηλώνει ότι οποιοδήποτε σύνολο ατόμων το οποίο περιέχει όλα τα $A_1,\ldots,A_m$ και κανένα από τα $A_{m+1}, \ldots, A_n$ δεν μπορεί να είναι λύση. Τυπικά, ένας περιορισμός είναι μια συντομογραφία του κανονικού κανόνα:

p ←not p, $A_1,\ldots,A_m$,not $A_{m+1},\ldots,$not A_n

όπου το p είναι ένα νέο άτομο. Ένα κλασσικό πρόγραμμα είναι ένα πεπερασμένο σύνολο από κανόνες.

Ένας κανόνας είναι θετικός αν θέσουμε n=m και έχει την μορφή $A_0 \leftarrow A_1,\ldots,A_m$.

Δηλαδή το πρόγραμμα

s ←

p ←q,s

είναι θετικό.

Αντίθετα το πρόγραμμα

p ←not q

q ←not r

δεν είναι θετικό.

Το παράδειγμα 4.1. παραθέτει ένα κλασσικό λογικό πρόγραμμα.

Παράδειγμα 4.1

p←q,s

t←

p←not q

q←not r

4.2 Σύνολο απαντήσεων θετικού προγράμματος[1]

Αρχικά θα μελετήσουμε το σύνολο για θετικά προγράμματα.

Ορισμός 4.4 Σύνολο απαντήσεων για θετικά προγράμματα

Ένα σύνολο ατόμων Χ ικανοποιούν το θετικό πρόγραμμα Π αν για όλους τους κανόνες της μορφής $A_0 \leftarrow A_1,\ldots,A_m$ στο πρόγραμμα Π, $A_0 \in X$ όταν $A_1,\ldots,A_m \in X$.

Έστω το παρακάτω πρόγραμμα Π1

s←t

p←q,s

t←

q←

Το σύνολο απαντήσεων του προγράμματος Π1 είναι Sp={q,t,s,p}.Στο σύνολο αυτό όλα τα άτομα λαμβάνουν την τιμή True.Το Sp ονομάζεται σύνολο απαντήσεων του Π1.Τα σύνολα απαντήσεων αναπαριστούνται από τα σύνολα των ατόμων που παίρνουν την τιμή True.

Ένα σύνολο S είναι σύνολο απαντήσεων ενός προγράμματος Π αν και μόνο αν το S είναι ελάχιστο σύνολο απαντήσεων του Π.'Ενα σύνολο S είναι ελάχιστο σύνολο απαντήσεων ενός προγράμματος Π αν $\nexists$ S' $\subset$ S τέτοιο ώστε S' είναι σύνολο απαντήσεων του Π. Αυτό σημαίνει ότι το σύνολο απαντήσεων είναι το μικρότερο σύνολο ατόμων που ικανοποιούν το πρόγραμμα Π.

Παράδειγμα 4.2

p←q

q←p

Στο παράδειγμα 4.2 το πρόγραμμα έχει δύο πιθανά σύνολα απαντήσεων

S_1={p, q},δηλαδή p=true, q=true

S_2=∅, δηλαδή p=false, q=false

Το S_1={p,q} δεν είναι το ελάχιστο σύνολο απαντήσεων επειδή ∅ ⊂ S_1 είναι σύνολο απαντήσεων του προγράμματος. Άρα από τα δύο αυτά μοντέλα μόνο το S_2 είναι σύνολο απαντήσεων .

Παράδειγμα 4.3

p←

r←p,q

Στο παράδειγμα 4.3 ο πρώτος κανόνας p← μας επιτρέπει να συμπεριλάβουμε το p στο σύνολο απαντήσεων.Ο δεύτερος κανόνας r←p,q μας λέει ότι μπορούμε να προσθέσουμε το r στο σύνολο απαντήσεων αν έχουμε ήδη συμπεριλάβει το p και το q.

Το πρόγραμμα έχει τρία πιθανά σύνολα απαντήσεων

S1={p}

S2={p, r}

S3={p, q, r}

Το S1={p} είναι το σύνολο απαντήσεων.

Αν στο πιο πάνω πρόγραμμα προσθέσω το q←p τότε το σύνολο απαντήσεων είναι το {p,q,r}.

4.3 Σύνολο απαντήσεων(answer set) ενός προγράμματος με άρνηση[1]

Ορισμός 4.5: Σύνολο απαντήσεων για προγράμματα με άρνηση

Επεκτείνουμε τα λογικά προγράμματα με τον τελεστή not. Ένα λογικό πρόγραμμα είναι ένα σύνολο από κανόνες της μορφής:

$A_0 \leftarrow A_1,\ldots,A_m$,not $A_{m+1},\ldots,$not A_n.

Ο τελεστής not ονομάζεται τελεστής άρνηση ως αποτυχία. Διαισθητικά, το not An είναι αληθές εάν δεν υπάρχει τρόπος να αποδειχθεί το An.Αυτό βασίζεται στην υπόθεση του κλειστού κόσμου. Αυτή η δήλωση υποδηλώνει ότι όλα τα γεγονότα που δεν είναι λογικές συνέπειες του προγράμματος είναι ψευδή.

Ορισμός 4.6: Reduct λογικού προγράμματος

Μας δίνεται ένα πρόγραμμα Π και ένα σύνολο ατόμων X. Το reduct του Π ως προς το X συμβολίζεται με το Π^X και ορίζεται σαν:

- Για κάθε q ∈ X αφαιρούμε όλους τους κανόνες με not q στο σώμα τους.
- Από όλους τους υπόλοιπους κανόνες αφαιρούμε όλα τα not από το σώμα τους

Ένα σύνολο S είναι το σύνολο απαντήσεων ενός προγράμματος Π αν X=closure(Π^X), όπου closure(Π^X) είναι το υπερσύνολο του Π^X που λαμβάνουμε εφαρμόζοντας επανειλημμένα τους δύο πιο πάνω κανόνες στο πρόγραμμα Π

Συμπέρασμα 4.1.

Αν το X είναι σύνολο απαντήσεων του προγράμματος Π τότε κάθε στοιχείο του X είναι κεφαλή σε ένα από τους κανόνες του Π. Για οποιοδήποτε πρόγραμμα Π και οποιαδήποτε σύνολα X,Y από άτομα, ισχύουν τα εξής

α)Αν X ⊆ Y τότε $\Pi^Y \subseteq \Pi^X$

β)Αν το X είναι σύνολο απαντήσεων σε ένα πρόγραμμα Π τότε κανένα υποσύνολο του X δεν μπορεί να είναι σύνολο απαντήσεων του Π.

Παράδειγμα 4.4

Έχουμε το πιο κάτω αντιφατικό πρόγραμμα:

ασθένεια←not(ασθένεια).

Στο παράδειγμα 4.4, το πρόγραμμα που παρατίθεται δεν έχει κάποιο σύνολο απαντήσεων γιατί δεν μπορούμε να συμπεράνουμε κατά πόσο έχει κάποια ασθένεια ή όχι. Αν υποθέσουμε ότι δεν υπάρχει ασθένεια τότε συμπεραίνουμε ότι υπάρχει ασθένεια, δηλαδή φτάνει σε αντίφαση. Έτσι δεν μπορούμε να συμπεράνουμε ότι δεν υπάρχει ασθένεια αλλά ούτε ότι υπάρχει ασθένεια.

Παράδειγμα 4.5

Έχουμε το πιο κάτω πρόγραμμα:

υγιής(Χ) ←άνθρωπος(Χ),not(ασθένεια(Χ)).

άνθρωπος(Νίκος).

Στο παράδειγμα 4.5 ,το πρόγραμμα που παρατίθεται εκφράζει ότι όλοι οι άνθρωποι που δεν έχουν κάποια ασθένεια, είναι υγιείς και ότι ο Νίκος είναι άνθρωπος. Αν κάποιος πράκτορας είχε στη βάση γνώσης του αυτό το πρόγραμμα τότε πιστεύει ότι ο Νίκος δεν έχει κάποια ασθένεια, δηλαδή θεωρείται not(ασθένεια(Νίκος)) ως αληθές, αφού στο πρόγραμμα δεν δηλώνει το αντίθετο. Και γι αυτό το λόγο ο πράκτορας συμπεραίνει ότι είναι υγιής. Στο πιο πάνω πρόγραμμα το σύνολο απαντήσεων είναι {άνθρωπος(Νίκος), υγιής(Νίκος)}. Αν επεκτείνουμε το πιο πάνω πρόγραμμα προσθέτοντας ότι ο Νίκος πάσχει από χρόνιο άσθμα τότε το πρόγραμμα θα γίνει:

υγιής(Χ) ←άνθρωπος(Χ), not(ασθένια(Χ)).

άνθρωπος(Νίκος).
πάσχει_από χρονιο_άσθμα(Νίκος).

ασθένεια(Χ) ←πάσχει_από_χρόνιο_άσθμα(Νίκος))

Έτσι ο πράκτορας με βάση το πιο πάνω πρόγραμμα συμπεραίνει ότι ο Νίκος δεν είναι υγιής. Στο πιο πάνω πρόγραμμα το σύνολο απαντήσεων είναι:

{άνθρωπος(Νίκος),πάσχει_από_χρόνιο_άσθμα(Νίκος),ασθένεια(Νίκος)}.

Παράδειγμα 4.6

Έστω ότι έχουμε το πιο κάτω πρόγραμμα:

υγιής(Νίκος) ←not(πάσχει_από χρόνιο_άσθμα(Νίκος)).

πάσχει_από χρόνιο_άσθμα(Νίκος) ←not υγιείς(Νίκος).

Το πιο πάνω πρόγραμμα του παραδείγματος 4.6 στοχεύει στο να εκφράσει ότι ο Νίκος είναι υγιείς αν δεν πάσχει από χρόνιο άσθμα και ότι ο Νίκος πάσχει από κάποια ασθένεια αν δεν είναι υγιής. Ο πράκτορας με βάση αυτό το πρόγραμμα θα συμπέραινε ότι ο Νίκος είναι υγιής αφού δεν μπορεί να συμπεράνει ότι πάσχει από χρόνιο άσθμα. Συμμετρικά θα συμπέραινε ότι ο Νίκος πάσχει από χρόνιο άσθμα αφού δεν μπορεί να συμπεράνει ότι είναι υγιής. Έτσι αυτό το πρόγραμμα έχει δύο ευσταθή μοντέλα το{πάσχει_από_χρόνιο_άσθμα(Νίκος)} και το {υγιείς(Νίκος)}.

4.4. Το Σύστημα Smodels.

Το Σύστημα Smodels αποτελείται από δύο προγράμματα, το Lparse και το Smodels. Το Lparse είναι ένας αναλυτής ο οποίος μετατρέπει το πρόγραμμα του λογικού προγραμματισμού σε κάποια μορφή, την οποία μπορεί να κατανοήσει το λογισμικό Smodels. Αφού το Smodels πάρει το πρόγραμμα αυτό σε μια συγκεκριμένη μορφή, μας δίνει το σύνολο απαντήσεων. Το Smodels είναι ένα ευρέως διαδεδομένο πρόγραμμα για εύρεση συνόλων απαντήσεων.

Τα βήματα που απαιτούνται για την εύρεση ενός ευσταθούς μοντέλου είναι:

α)Γράφουμε το λογικό πρόγραμμα στο αρχείο Input.pl

β)Το lparse μετατρέπει το πρόγραμμα σε μορφή που την κατανοεί το Smodels.

γ)Το Smodels δίνει στην έξοδο τα σύνολα απαντήσεων.

‘Οπως και στην Prolog, το ← στην lparse συμβολίζεται με :-. Επιπλέον βάζουμε ένα κανόνα σε κάθε γραμμή και κάθε κανόνας τερματίζει με τελεία(.)

Παράδειγμα 4.7

Αν θέλουμε να βρούμε το σύνολο απαντήσεων του πιο κάτω προγράμματος:

k←not m

m←not k

r←m

r←k

τότε δημιουργούμε ένα αρχείο και το ονομάζουμε program.pl και στη συνέχεια τρέχουμε το πρόγραμμα ως εξής:

%lparse program| smodels 0

Το 0 στο τέλος της εντολής δείχνει ότι θέλουμε να υπολογίσουμε όλα τα σύνολα απαντήσεων βάλουμε κάποιο θετικό αριθμό k στη θέση του 0 τότε το Smodels θα τερματίσει όταν υπολογίσει τα πρώτα k σύνολα απαντήσεων . Η προεπιλεγμένη τιμή για το k είναι το 1.Η έξοδος του προγράμματος είναι μια λίστα με τα σύνολα απαντήσεων του προγράμματος:

Smodels version 2.28.Reading...done

Answer:1

Stable Model: r m

Answer: 2

Stable Model: r k

False

Duration: 0.15

Number of choice points: 1

Number of wrong choices: 1

Number of atoms: 3

Number of rules: 4

Number picked atoms: 4

Number of forced atoms: 1

Number of truth assignments: 10

Size of searchspace (removed) : 2(0)

Με το stable model εννοούμε σύνολο απαντήσεων. Στην πρώτη γραμμή τυπώνει πληροφορίες για την έκδοση του Smodels. Οι επόμενες γραμμές μας δίνουν τα δύο ευσταθή μοντέλα του πιο πάνω προγράμματος που είναι {r,m} και {r,k}.Η λέξη False κάτω από τα ευσταθή μοντέλα δηλώνει ότι δεν υπάρχουν άλλα ευσταθή μοντέλα που

να ικανοποιούν το πιο κάτω πρόγραμμα. Η λέξη False εμφανίζεται και όταν το πρόγραμμα δεν έχει καθόλου λύσεις. Στην περίπτωση που θα ήταν True δηλώνει ότι πιθανών υπάρχουν κι άλλα ευσταθή μοντέλα τα οποία το smodels δεν υπολόγισε. Επιπλέον η λέξη Duration μας λέει πόσο χρόνο χρειάστηκε σε δευτερόλεπτα να γίνει η αναζήτηση. Ο αριθμός των choice points δηλώνει πόσες φορές το Smodels έπρεπε να μαντέψει μια τιμή για κάθε άτομο. Ο αριθμός των wrong choices δηλώνει πόσες φορές έδωσε λάθος τιμές στις μεταβλητές. Στην συνέχεια μας δίνει πληροφορίες για τον αριθμό των ατόμων και τον αριθμό των κανόνων που αποτελούν το πρόγραμμα. Ο αριθμός των picked atoms μας δηλώνει πόσες φορές η συνάρτηση lookahead του smodels κατάφερε να επιλέξει τη σωστή τιμή αληθείας για ένα άτομο. Ο αριθμός των forced atoms μας λέει πόσα άτομα προστέθηκαν στο σύνολο απαντήσεων λόγω των κανόνων διάχυσης περιορισμών που θα προκαλούσαν αντίφαση. Ο αριθμός των truth assignments μας δηλώνει πόσες φορές ανατέθηκε μια αληθής τιμή σε ένα άτομο. Τέλος το Size of searchspace δηλώνει τον μέγιστο αριθμό επιλογών που μπορεί να κάνει το Smodels πριν σιγουρευτεί αν ένα σύνολο απαντήσεων υπάρχει ή όχι. Στο lparse μπορούμε να χρησιμοποιήσουμε μεταβλητές. Όπως και στην Prolog, οι μεταβλητές αναπαριστώνται με κεφαλαία γράμματα.

4.5 Η διάδοση περιορισμών στο σύστημα Smodels[6]

Παραθέτουμε τον τρόπο διάδοσης περιορισμών στο σύστημα Smodels. Αρχικά όμως δίνουμε τους πιο κάτω ορισμούς.

Ένα σύνολο μεταβλητών λογικής είναι συνεπές αν δεν υπάρχει κάποιο προτασιακό άτομο a έτσι ώστε το a και το not a να είναι και τα δύο στο σύνολο, αλλιώς θα έχουμε αντίφαση στο σύνολο. Δεδομένου ενός συνόλου μεταβλητών λογικής A , ορίζουμε το A^+ = { a| a ∈ A, a είναι ένα προτασιακό άτομο} και A^- = { a| not a ∈ A, a είναι ένα προτασιακό άτομο}. Δεδομένου του συνόλου μεταβλητών λογικής B, ορίζουμε το σύνολο not(B) = { not b| b ∈ B, b είναι ένα προτασιακό άτομο}. Για ένα πρόγραμμα P, το σύνολο atoms(P) δηλώνει το σύνολο όλων των προτασιακών ατόμων x έτσι ώστε να μην υπάρχει στο σύνολο το x ή το not(x) στο P. Ένα σύνολο ατόμων A συμφωνεί με το B αν $B^+ \subseteq$ A και $B^- \cap$ not(A) = ∅.Η άρνηση της λογικής μεταβλητής x ορίζεται ως not(x), όπου το x είναι ένα προτασιακό άτομο, not(x) = not x και not(not(x)) = x. Το B καλύπτει το A αν A ⊆ { x | x ∈ B ή (not x) ∈ B}.

Ο αλγόριθμος του Smodels

Πιο κάτω παρατίθεται ο αλγόριθμος του Smodels.

```
Function smodels (P,A)
A  : = expand(P,A)
A := lookahead (P,A)

If conflict (P,A) then
        Return  false
Else if A covers Atoms(P) then
        Return true { A+ is a stable model}
Else
        x : = heuristic(P,A)
        If smodels (P, A ∪ {x}  then
                Return true
        Else
                Return smodels(P, A  ∪ {not x})
        End if
End if
```

Στον πιο πάνω αλγόριθμο, το Π είναι ένα λογικό πρόγραμμα συνόλου απαντήσεων και το Α είναι ένα σύνολο από μεταβλητές λογικής. Η συνάρτηση smodels(P,A) επιστρέφει αληθές αν υπάρχει ένα σύνολο απαντήσεων (stable model) του P το οποίο συμφωνεί με το σύνολο Α. Δεδομένου ενός συνόλου από μεταβλητές λογικής, ο αλγόριθμος αρχικά εφαρμόζει διάδοση περιορισμών, η οποία αποτελείται από δύο συναρτήσεις, την expand(P, A) και τη lookahead(P,A). Ένα υπερσύνολο του Α, το οποίο ονομάζουμε Α', επιστρέφεται. Έχει αποδεικτεί ότι οποιοδήποτε σύνολο απαντήσεων συμφωνεί με το Α, συμφωνεί και με το Α'. Ακολούθως ελέγχει αν υπάρχει αντίφαση στο Α'. Αν αυτό ισχύει, επιστρέφει false, και επιστρέφει πίσω για να αναθέσει σε άλλη μεταβλητή τιμή. Αν το Α' είναι συνεπές και καλύπτει όλα τα προτασιακά άτομα στο πρόγραμμα P, τότε το σύνολο ατόμων A'^{+} είναι ένα σύνολο απαντήσεων και γι αυτό ο αλγόριθμος τερματίζει. Αν το Α' δεν καλύπτει όλα τα άτομα στο P, τότε ο αλγόριθμος διαλέγει μια μεταβλητή λογικής x που δεν ανήκει στο Α', η οποία υπολογίζεται από τη συνάρτηση heuristic(P, A'), και την προσθέτει στο Α'. Αν η επιλεγμένη μεταβλητή μαζί με το Α' μπορεί να επεκταθεί σε ένα σύνολο απαντήσεων, επιστρέφεται true,αλλιώς η συνάρτηση ελέγχει αν το not x ∪ Α' μπορεί να επεκταθεί σε ένα σύνολο απαντήσεων.

Η συνάρτηση Expand (P,A)

Παρακάτω παρατίθεται η συνάρτηση expand(P, A).

```
Function expand(P,A)
Repeat
        A' : = A
        A : = atleast (P,A)
        A : = A ∪ { not x | x  ∈ Atoms(P)
                and x ∈ atmost(P, A) }
until A = A'
return A.
```

Η συνάρτηση expand(P,A) επεκτείνει το σύνολο ατόμων Α χρησιμοποιώντας δύο συναρτήσεις, την atleast (P,A) και την atmost(P, A). Η συνάρτηση atleast (P,A) επιστρέφει ένα υπερσύνολο Α' του Α εφαρμόζοντας τέσσερις κανόνες διάδοσης. Οποιοδήποτε σύνολο απαντήσεων συμφωνεί με το Α πρέπει να συμφωνεί και με το σύνολο ατόμων Α'.

Η συνάρτηση atleast(P,A) ορίζεται με βάση τη σημασιολογία των λογικών προγραμμάτων συνόλων απαντήσεων και χρησιμοποιεί τους πιο κάτω κανόνες:

α)Η κεφαλή του κανόνα θα πρέπει να είναι true αν το σώμα είναι true στην Α.

β) Αν δεν υπάρχει κανένας κανόνας με το a ως κεφαλή του οποίου το σώμα δεν είναι αληθές στο Α, τότε το a δεν μπορεί να παραχθεί από οποιοδήποτε συνεπές μοντέλο που συμφωνεί με το Α.

γ)Αν το a ∈ Α, και υπάρχει μόνο ένας κανόνας με το a ως κεφαλή του οποίου το σώμα δεν είναι ακόμα ψευδές στο Α, τότε το σώμα του κανόνα θα έπρεπε να είναι αληθές.

δ) Το σώμα του κανόνα θα πρέπει να είναι ψευδές αν η κεφαλί είναι ψευδές στο Α.

Οι τέσσερις πιο πάνω κανόνες διάδοσης μπορούν να παράξουν νέες μεταβλητές λογικής με τις οποίες το σύνολο απαντήσεων συμφωνεί.

Παράδειγμα 4.8

Έστω το P το πιο κάτω λογικό πρόγραμμα

a ← c, not e (κ1)
b ← not a (k2)
c ← a, not b (k3)
f ← b, not c (k4)

Δεδομένου ενός συνόλου A = { b} υπολογίζουμε την atleast (P,A) ως ακολούθως:

1. Επειδή το e δεν εμφανίζεται στην κεφαλή κανενός κανόνα του P, εφαρμόζοντας τον κανόνα 2, παίρνουμε A = {b, not e}.
2. Εφαρμόζοντας τον κανόνα 3, αφού το b ∈ A, και ο κ2 είναι ο μόνος κανόνας που έχει το b ως κεφαλή, το not a θα προστεθεί στο A και παίρνουμε A = { b, not a, not e}.
3. Αφού το not a ∈ A και το a είναι η κεφαλή του κανόνα κ1, τότε εφαρμόζοντας τον κανόνα 4, προσθέτουμε το not c στο A και παίρνουμε A = { b, not a, not e, not c}.
4. Εφαρμόζοντας τον κανόνα κ1, προσθέτουμε και το f στο A και παίρνουμε A = {b, not a, not e, not c, f}.
5. Αφού καμιά νέα μεταβλητή λογικής δεν μπορεί να προστεθεί στο A, σταματούμε.

Η συνάρτηση Lookahead (P,A)

Παρακάτω παρατίθεται η συνάρτηση Lookahead (P, A).

```
Function lookahead(P,A)
repeat
        A' : = A
        A : = lookahead_once(P,A)
Until A = A'
return  A.

Function lookahead_once(P,A)
B : = Atoms(P) – Atoms(A)
B : = B  ∪ not (B)
while B ≠ Ø do
        take any literal x ∈ B
        A' = expand(P, A ∪ {x})
        B : = B – A'
```

```
        If conflict(P,A') then
                return expand (P,A ∪ { not (x)})
        end if
end while
return A.
```

Η συνάρτηση expand χρησιμοποιεί τη σημασιολογία των λογικών προγραμμάτων συνόλων απαντήσεων για να προσδιορίσει ένα μερικό σύνολο απαντήσεων και έτσι να μειώσει το χώρο αναζήτησης για εξεύρεση λύσης του προβλήματος. Αφού εφαρμόσουμε τη συνάρτηση expand, φτάνουμε το σημείο στο οποίο δεν μπορούμε να προσθέσουμε άλλες μεταβλητές λογικής στο σύνολο Α. Μετά μπορούμε να επεκτείνουμε ακόμα περισσότερο το μερικό σύνολο απαντήσεων χρησιμοποιώντας τη συνάρτηση lookahead, η οποία ελέγχει κάθε πιθανή επιλογή πριν προσθέσει κάποιο άτομο στο σύνολο Α. Δεδομένου του προγράμματος Π και του συνόλου μεταβλητών λογικής Α, διαλέγουμε μια επιπρόσθετη μεταβλητή x που δεν ανήκει στο Α και θέτουμε το Α' = expand(P,A ∪ {x}). Αν υπάρχει αντίφαση στο Α΄ , τότε η not x προστίθεται στο μερικό σύνολο απαντήσεων.

Αυτό βασίζεται στον ακόλουθο κανόνα μετάδοσης:

Αν το σύνολο απαντήσεων Μ συμφωνεί με το σύνολο από λογικές μεταβλητές Β αλλά δεν συμφωνεί με το Β ∪ {x}, ισχύει ότι το Μ θα πρέπει να συμφωνεί με το Β ∪ { not x}.

Η συνάρτηση lookahead(P,A) καλεί τη συνάρτηση lookahead_once(P,A) επανειλημμένα μέχρι καμία νέα μεταβλητή λογικής να μπορεί να προστεθεί στο σύνολο απαντήσεων Α. Η lookahead_once(P,A) διαλέγει μια μεταβλητή λογικής που δεν ανήκει στο σύνολο Α, ας πούμε το x, και επεκτείνει το Α με το x. Έστω Α' = expand(P, A ∪ {x}). Αν υπάρχει αντίφαση στο Α', τότε επιστρέφει expand(P, AU{ not x}), αλλιώς η συνάρτηση lookahead_once(P,A), δοκιμάζει όλες τις μεταβλητές λογικής που δεν ανήκουν στο Α, μέχρι να υπάρξει αντίφαση ή έχουν δοκιμαστεί όλες μεταβλητές λογικής που δεν ανήκουν στο Α. Σε αυτή την περίπτωση επιστρέφει το σύνολο απάντησης Α.

Λήμμα 4.1[6]

Η συνάρτηση lookahead(P,A) επιστρέφει το ίδιο σύνολο απαντήσεων ανεξάρτητα της σειράς με την οποία οι μεταβλητές διαλέγονται από το σύνολο B στο βρόγχο επανάληψης while της συνάρτησης lookahead_once(P,A).

Παράδειγμα 4.8

Έστω το πιο κάτω λογικό πρόγραμμα:

a ← not b.
b ← not a.
a ← b.

Το σύνολο απαντήσεων για αυτό το πρόγραμμα είναι το {a}. Αν η lookahead ελέγξει πρώτα το not a, θα υπάρξει αντίφαση. Για αυτό το a προστίθεται στο σύνολο απαντήσεων. Υποθέτουμε ότι μετά ελέγχει το b, και εμφανίζεται ακόμα μια αντίφαση. Ως αποτέλεσμα το σύνολο {a, not b} προστίθεται στο σύνολο απαντήσεων. Έχει αποδεικτή ότι η συνάρτηση lookahead επιστρέφει τα ίδια αποτελέσματα ασχέτως της σειράς που επιλέγει τις μεταβλητές λογικής.

Κεφάλαιο 5

Σύγκριση λογικών προγραμμάτων απαντήσεων συνόλων με άλλες μεθόδους

5.1 Μετάφραση προβλημάτων ικανοποίησης περιορισμών σε προβλήματα συνόλων απαντήσεων.

Υπάρχουν διάφορες μεταφράσεις προβλημάτων ικανοποίησης περιορισμών σε προβλήματα συνόλων απαντήσεων. Για να συγκρίνουμε όμως τη συνάρτηση lookahead του συστήματος Smodels με τη συνέπεια ακμής των προβλημάτων ικανοποίησης περιορισμών στο υποκεφάλαιο 5.2, παραθέτουμε πιο κάτω τη βασική μετάφραση.

Ορισμός 5.1: Η βασική μετάφραση[6]

Η μετάφραση αυτή ονομάζεται βασική, επειδή είναι ο πιο ευθύς τρόπος να μεταφράσουμε ένα πρόβλημα ικανοποίησης περιορισμών σε λογικό πρόγραμμα συνόλων απαντήσεων.

Έστω A(X, D, C) ένα πρόβλημα ικανοποίησης περιορισμών. Ορίζουμε ως P_A το λογικό πρόγραμμα συνόλου απαντήσεων που προκύπτει από το A. Η βασική

μετάφραση αποτελείται από τρία μέρη. Το πρώτο μέρος προσδιορίζει την ιδιότητα μοναδικότητας, δηλαδή σε κάθε δυαδική μεταβλητή του προβλήματος ικανοποίησης περιορισμών μπορεί να ανατεθεί μόνο μια τιμή.

Για κάθε μεταβλητή $x_i \in$ X με πεδίο ορισμού $D_{xi} = \{a_1, \ldots, a_l\}$, χρησιμοποιούμε ένα προτασιακό άτομο, x_i (α_j), για να δηλώσουμε αν το x_i παίρνει την τιμή α_j ή όχι. Άρα για κάθε $1 \leq j \leq l$, προσθέτουμε ένα κανόνα της μορφής:

x_i (α_j) ← not x_i (α_1) , . . . , not x_i (α_{j-1}), not x_i (α_{j+1}), . . ., not x_i (α_l).

Στο δεύτερο μέρος της μετάφρασης, εκφράζονται οι περιορισμοί. Για κάθε περιορισμό $c_i(x_1, \ldots, x_n\}$ και για κάθε πλειάδα $(\alpha_1 , \ldots , \alpha_n) \in c_i$ προσθέτουμε τον κανόνα

sat(c_i) ← $x_1(a_1)$, . . . , $x_n(a_n)$.

Στο τελευταίο βήμα εκφράζουμε ότι κάθε περιορισμός στο σύνολο C πρέπει να ικανοποιεί τους κανόνες:

sat ← sat(c_1) , . . . , sat(c_m).

f ← not f, not sat

Οι δύο πιο πάνω κανόνες μπορούν να αγνοηθούν αν ζητήσουμε από το σύστημα Smodels να υπολογίσει τα σύνολο απαντήσεων που περιέχουν sat(c_1) , . . . , sat(c_m). Θα έχουμε δηλαδή το sat(C) = {sat(c_1) , . . . , sat(c_m)}.

Παράδειγμα 5.1

Έστω πιο κάτω ένα πρόβλημα ικανοποίησης περιορισμών με μεταβλητές x και y και τον περιορισμό c_{xy} ={(0,0),(1,1)}.

Όπου D_x={0,1,2} και D_y= {0,1}.Το μεταφράζουμε σε ένα κανονικό πρόγραμμα.

Μια μεταβλητή παίρνει ακριβώς μια τιμή από το πεδίο ορισμού της:

x(0)←not x(1),not x(2).

x(1)←not x(0),not x(2).

x(2)←not x(0),not x(1).

y(0)←not y(1).

y(1)←not y(0).

Οι επιτρεπτές πλειάδες στον περιορισμό αναπαριστούνται ως:

sat(c)←x(0),y(0).

sat(c)←x(1),y(1).

και θα υπολογιστούν τα σύνολα απαντήσεων που περιέχονται στο sat(C).

Πρόταση 5.1.

Έχει αποδεικτή ότι ένα πρόβλημα ικανοποίησης περιορισμών A έχει μια λύση εάν και μόνο εάν το P_A ένα σύνολο απαντήσεων.

5.2 Σύγκριση της συνάρτησης lookahead του συστήματος Smodels με τη συνέπεια ακμής των προβλημάτων ικανοποίησης περιορισμών [6]

Λαμβάνοντας υπόψη τη βασική μετάφραση, συγκρίνουμε τη συνάρτηση lookahead του συστήματος Smodels με τη συνέπεια ακμής.

Παράδειγμα 5.2

Το πιο κάτω πρόβλημα ικανοποίησης περιορισμών αποτελείται από τρεις μεταβλητές x, y και z όλες με πεδίο ορισμού {0,1} και έχει τους εξής περιορισμούς

c_{xy} = {(0,0),(1,1)}

c_{yz} = {(0,1),(1,0)}

c_{zx} = {(0,0),(1,1)}

Το μεταφράζουμε σε ένα κανονικό πρόγραμμα.

Μια μεταβλητή παίρνει ακριβώς μια τιμή από το πεδίο ορισμού της:

x(0)←not x(1).

x(1)←not x(0).

y(0)←not y(1).

y(1)←not y(0).

z(0)←not z(1).

z(1)←not z(0).

Οι επιτρεπτές πλειάδες στον περιορισμό c_{xy} αναπαριστούνται ως:

sat(c_1)←x(0),y(0).

sat(c_1)←x(1),y(1).

Οι επιτρεπτές πλειάδες στον περιορισμό c_{yz} αναπαριστούνται ως:

sat(c_2)←y(0),z(1).

sat(c_2)←y(1),z(0).

Οι επιτρεπτές πλειάδες στον περιορισμό c_{zx} αναπαριστούνται ως:

sat(c_3)←z(0),x(0).

sat(c_3)←z(1),x(1).

Παρατηρούμε ότι η συνέπεια ακμής δεν μπορεί να αφαιρέσει τιμές από τα πεδία ορισμού των μεταβλητών του προβλήματος ικανοποίησης περιορισμών. Αν η συνάρτηση lookahead δοκιμάσει να αναθέσει την τιμή 0 στη μεταβλητή x στο μεταφρασμένο λογικό πρόγραμμα, τότε θα έχουμε αντίφαση και έτσι η μεταβλητή not x(0) θα προστεθεί στο σύνολο απαντήσεων. Αυτό θα έχει αποτέλεσμα να προστεθούν και οι μεταβλητές not x(1), not y(0), not y(1), not z(0) και not z(1).

Συμπέρασμα 5.1

Παρατηρούμε ότι η εφαρμογή της lookahead στο πρόβλημα του παραδείγματος 5.2 έχει ως αποτέλεσμα να πάρουν τιμή περισσότερες μεταβλητές σε σύγκριση με την εφαρμογή της συνέπειας ακμής στο ίδιο πρόβλημα. Για αυτό το λόγο η συνάρτηση lookahead είναι αυστηρά πιο δυνατή από τη συνέπεια ακμής.

Ορισμός 5.2: Συνέπεια ακμής με μοναδιαία μετάδοση[6]

Έχουμε αποδείξει ότι η συνέπεια ακμής είναι αυστηρά πιο αδύνατη από τη συνάρτηση lookahead του συστήματος Smodels. Πιο κάτω προσδιορίζουμε αυτό που λείπει από τη συνέπεια ακμής.

Δεδομένου ενός προβλήματος ικανοποίησης περιορισμών A(X,D,C) και ένα σύνολο μεταβλητών Π στις οποίες έχει ανατεθεί τιμή , ορίζουμε μια συνάρτηση η οποία επεκτείνει το Π ως ακολούθως:

unit_propagate(A,Π)[6]

= Π ∪ { x → a | c_{yx} ∈ C, ή c_{xy} ∈ C και y → b ∈ Π , έτσι ώστε η τιμή a είναι η μοναδική τιμή στο πεδίο ορισμού D_x η οποία είναι συνεπής με την τιμή b.

Στην πιο πάνω συνάρτηση με τον ορισμό x → a εννοούμε ότι στη μεταβλητή x ανατίθεται η τιμή a.

Η πιο πάνω συνάρτηση δηλώνει ότι δεδομένου ενός δυαδικού περιορισμού, αν το πεδίο ορισμού μιας μεταβλητής x στην οποία δεν έχει ανατεθεί τιμή ακόμα, έχει ακριβώς μια τιμή a η οποία είναι συνεπής με μια μεταβλητή y στην οποία έχει ανατεθεί μια τιμή b, τότε θα πρέπει να ανατεθεί η τιμή a στην μεταβλητή x.

Ένα ζεύγος αναθέσεων έρχεται σε αντίφαση αν και μόνο αν υπάρχουν διακριτές τιμές d και d' ώστε x→d και x→ d' ∈ Π. Ο ορισμός της αντίφασης είναι αναγκαίος επειδή όπως θα δούμε παρακάτω η συνάρτηση unit_propagate(A,P) μπορεί να οδηγήσει σε μια τέτοια αντίφαση.

Η συνάρτηση unit_propagate* (A,Π) πιο κάτω καλεί τη συνάρτηση unit_propagate(A,Π) επανειλημμένα μέχρι τίποτα να μη μπορεί να διαδοθεί ή μέχρι να υπάρξει κάποια αντίφαση.

```
Function unit_propagate*(A,Π)
        repeat
        Π' : = Π
        Π : = unit_propagate(A,Π')
        If Π is in conflict then
        Return "conflict"
        until Π = Π'
  return Π.
```

Παράδειγμα 5.3

Έστω το πρόβλημα ικανοποίησης περιορισμών A(X,D,C) με X = {x , y} , D_x = D_y = {0, 1} και το σύνολο περιορισμών C να αποτελείται από c_{xy} ={(0,1)}, c_{yz} = {(1,1)}, και c_{zx} ={(1,1)}.

Δεδομένου του Π = { x→0} , η συνάρτηση unit_propagate* (Α,Π) επιστρέφει αντίφαση επειδή το σύνολο που επιστρέφεται από τη συνάρτηση unit_propagate(Α,Π) είναι το {χ→0, y→1, z→1, x→1}.

Με τον πιο πάνω τρόπο έχουμε δυναμώσει τη διαδικασία επιβολής συνέπειας ακμής, προσθέτοντας τη μοναδιαία διάδοση. Ονομάζουμε αυτή τη συνάρτηση AC^+ και την παραθέτουμε πιο κάτω:

Η συνάρτηση AC^+ [6]

Η συνάρτηση AC^+ παίρνει ως είσοδο ένα πρόβλημα ικανοποίησης περιορισμών Α(Χ,D,C) και εκτελεί επανειλημμένα τις ακόλουθες μειώσεις στα πεδία ορισμού μέχρι κανένα πεδίο ορισμού να μην μπορεί να μειωθεί περεταίρω.

1. Για κάθε περιορισμό c ∈ C, αν υπάρχει ακριβώς μια μεταβλητή x στην οποία δεν έχει ανατεθεί τιμή ακόμα, αφαιρούμε οποιαδήποτε τιμή d από το πεδίο ορισμού D_x, αν η τιμή d δεν είναι συνεπής με μια διαφορετική τιμή e που έχει δοθεί στην μεταβλητή x. Δηλαδή εφαρμόζουμε μοναδιαία συνέπεια.
2. Για κάθε περιορισμό c ∈ C, αν υπάρχουν δύο μεταβλητές x και y στις οποίες δεν έχουν δοθεί τιμές ακόμα, αφαιρούμε οποιαδήποτε τιμή d οπό το πεδίο ορισμού D_x αν
   (α) δεν υπάρχει κάποια τιμή στο πεδίο ορισμού D_y έτσι ώστε μαζί με το d, είναι συνεπείς με τον περιορισμό c.Δηλαδή εφαρμόζουμε συνέπεια ακμής.
   ή
   (β) αν η συνάρτηση unit_propagate*(A, {x→d}) επιστρέφει αντίφαση. Δηλαδή εφαρμόζουμε συνέπεια ακμής με μοναδιαία διάδοση.

Σύγκριση των αλγορίθμων AC και AC^+[6]

Είναι φανερό ότι ο αλγόριθμος AC^+ είναι ορθός. Το μόνο βήμα που έχει προστεθεί σε σχέση με τον αλγόριθμο AC, είναι το βήμα 2.β, στο οποίο η αντίφαση είναι αρκετή για την αφαίρεση της μεταβλητής d.Θα πρέπει να σημειωθεί ότι αν τα πεδία ορισμών των μεταβλητών που επηρεάζονται έχουν περισσότερες από μια συνεπείς τιμές, η διαδικασία της μοναδιαίας μετάδοσης σταματά. Για αυτό το λόγο ο περεταίρω χρόνος που χρειάζεται από τον αλγόριθμο AC^+, είναι ανάλογος της ύπαρξης των μοναδικών αυτών τιμών για τις μεταβλητές κατά τη διάρκεια της μοναδιαίας μετάδοσης.

Θεώρημα 5.1[6]

Έστω το πρόβλημα ικανοποίησης περιορισμών A(X,D,C).Υποθέτουμε ότι μετά την εφαρμογή του αλγόριθμου AC^+, το σύνολο των πεδίων ορισμού D μειώνεται στο σύνολο D'. Τότε για οποιαδήποτε μεταβλητή x ∈ X, μια τιμή d αφαιρείται από το πεδίο ορισμού D_x, εάν και μόνο εάν η μεταβλητή not x(d) είναι στο σύνολο απαντήσεων A που επιστρέφεται από τη συνάρτηση lookahead(P_A,sat(C)).

Λαμβάνοντας υπόψη το πιο λήμμα 4.1 και το θεώρημα 5.1 τότε συμπεραίνουμε το εξής:

Συμπέρασμα 5.2[6]

Έστω το πρόβλημα ικανοποίησης περιορισμών A(X,D,C). Ο αλγόριθμος AC^+ επιστρέφει το ίδιο σύνολο από τιμές που έχουν αφαιρεθεί από τα πεδία ορισμών D, ανεξάρτητα της σειράς με την οποία οι μεταβλητές διαλέγονται από το σύνολο X και οι τιμές διαλέγονται από το σύνολο D.

5.3 Μετάφραση προβλημάτων προτασιακής ικανοποιησιμότητας σε προβλήματα συνόλων απαντήσεων.

Η μετάφραση προβλημάτων προτασιακής ικανοποιησιμότητας σε προβλήματα συνόλων απαντήσεων μπορεί να γίνει ευθέως.

Δηλαδή για κάθε λογική πρόταση της μορφής $x_1 \vee x_2, \ldots, \vee x_n$ του προβλήματος ικανοποιησιμότητας δημιουργούμε ένα σύνολο από κανόνες του προβλήματος συνόλου απαντήσεων όπως φαίνεται στο πιο κάτω παράδειγμα:

Παράδειγμα 5.4

Έστω η λογική πρόταση P = a ∨ ¬b ∨ c
Η πρόταση P μεταφράζεται στους πιο κάτω κανονικούς κανόνες

r ←not a'.
r ←not b.
r ← not c'.
← not r.

Στο πιο πάνω παράδειγμα ο τελευταίος κανόνας δηλώνει ότι η νέα μεταβλητή r είναι πάντα αληθής. Επίσης μια θετική μεταβλητή προτασιακής λογικής p μεταφράζεται στη μεταβλητή not p', όπου το άτομο p' είναι το λογικό συμπλήρωμα της p. Επίσης μια αρνητική μεταβλητή προτασιακής λογικής ¬p μεταφράζεται στη μεταβλητή not p.

5.4 Μετάφραση προβλημάτων συνόλων απαντήσεων σε προβλήματα προτασιακής ικανοποιησιμότητας.

Με τη μέθοδο ολοκλήρωσης του Clark(Clark's completion) μπορούμε να μεταφράσουμε προβλήματα συνόλου απαντήσεων σε προβλήματα προτασιακής ικανοποιησιμότητας .

Δεδομένου ενός προβλήματος συνόλου απαντήσεων P, ονομάζουμε τη μετάφρασή του σε πρόβλημα ικανοποιησιμότητας ως Comp(P).Για να γίνει η μετάφραση ακολουθούμε τους πιο κάτω κανόνες:

α) Αν το άτομο p δεν περιέχεται στην κεφαλή κανενός κανόνα τότε το θέτουμε να είναι ψευδές στη βάση γνώσεων, δηλαδή προσθέτουμε τη λογική πρόταση ¬p

β) Αν υπάρχουν ακριβώς n κανόνες που έχουν το άτομο p ως κεφαλή και το Body_i ως σώμα, δηλαδή κανόνες τις μορφής p← Body_i , όπου το i ϵ [1 . . .n], και το (Body_i)' παράγεται από το Body_i αν αντικαταστήσουμε κάθε παρουσία του not q στο Body_i με ¬q, τότε προσθέτουμε τη λογική πρόταση

p ⇔ (Body_1)' ∨ . . . . ∨ (Body_n)

Παράδειγμα 5.5

Έστω το πρόβλημα συνόλου απαντήσεων P

P: a ← b, not c.
a ← not d.
c ←not a.
Το πρόβλημα προτασιακής ικανοποιησιμότητας που προκύπτει είναι το εξής

Comp(P) = {
(a ⇔ (b ∧ ¬c) ∨ -d)) ∧ (c ⇔ ¬a) ∧ (¬d)
}

Το πιο πάνω πρόβλημα ικανοποιησιμότητας Comp(P) μπορούμε εύκολα να το μεταφέρουμε σε κανονική μορφή σύζευξης χρησιμοποιώντας τη λογική ισοδυναμία (a ⇔ b) ⇔ (¬a ∨ b) ∧ (a ∨ ¬b)

Θεώρημα 5.2[9]

Έστω P ένα πρόβλημα συνόλων απαντήσεων. Αν το X είναι ένα σύνολο απαντήσεων του Π, τότε το X ικανοποίει το Comp(P).

Το πιο πάνω θεώρημα συσχετίζει τα σύνολα απαντήσεων κάποιου προβλήματος συνόλων απαντήσεων P, με τη λογική απάντηση του προβλήματος ικανοποιησιμότητας Comp(P) που ικανοποιεί τις προτάσεις του Comp(P). Δηλαδή ένα σύνολο είναι σύνολο απαντήσεων του P αν ικανοποιεί το Comp(P).

5.5 Αλγόριθμος βασισμένος σε προβλήματα προτασιακής ικανοποιησιμότητας για την εύρεση συνόλων απαντήσεων.

Χρησιμοποιώντας τον αλγόριθμο DPLL μπορούμε να υπολογίσουμε το Comp(P) ενός προβλήματος συνόλου απαντήσεων σε κλασική λογική και μετά να υπολογίσουμε τις λογικές απαντήσεις του Comp(P). Ακολούθως ελέγχουμε αν οι παραγόμενες λογικές απαντήσεις είναι σύνολα απαντήσεων του P.

Πιο κάτω παρατίθεται ο αλγόριθμος DPLL [9]

DPLL(Γ, S)
if Γ =Ø then return *True*;
if ; Ø $\in$ Γ then return *False*;
if {l} $\in$ Γ then return DPLL(assign(l, Γ), S ∪{l});
p := an atom occurring in Γ;
return DPLL(assign(p, Γ), S ∪ {p}) or
DPLL(assign(¬p, Γ), S ∪ {¬p}).

Στον πιο πάνω αλγόριθμο το l δηλώνει μια μεταβλητή λογικής ,το Γ είναι το σύνολο από λογικές προτάσεις του προβλήματος ικανοποιησιμότητας και το S μια μερική ανάθεση δηλαδή ένα συνεπές σύνολο από μεταβλητές .Δεδομένου ενός ατόμου a , το assign(a,Γ) είναι ένα σύνολο από λογικές προτάσεις το οποίο παράγεται από το Γ αν αφαιρέσουμε τις λογικές προτάσεις στις οποίες ανήκει το a και αν αφαιρέσουμε το ¬a από όλες τις άλλες λογικές προτάσεις στο Γ. Με άλλα λόγια το assign(a,Γ) εφαρμόζει μοναδιαία απαλοιφή με βάση το a.Το assign(¬a,Γ) ορίζεται παρόμοια. Στην αρχική

κλίση του αλγορίθμου το S είναι το κενό σύνολο.DPLL(Γ, Ø) επιστρέφει αληθές οποτεδήποτε το Γ ικανοποιείται και αληθές αλλιώς.

Δεδομένου του αλγορίθμου DPLL μπορούμε να έχουμε ένα αλγόριθμο βασισμένο σε προβλήματα προτασιακής ικανοποιησιμότητας για την εύρεση συνόλων απαντήσεων λογικών προγραμμάτων P.

1.Αλλάζουμε την πρώτη γραμμή του αλγορίθμου αντικαθιστώντας το “return true” με το “return test(S,Π)”, που είναι μια νέα συνάρτηση η οποία τυπώνει το σύνολο atoms(S) = S ∩ P και επιστρέφει αληθές αν το atoms(S) είναι ένα σύνολο απαντήσεων του P και επιστρέφει ψευδές αλλιώς.

2.Ορίζουμε την συνάρτηση ASP-SAT(P),η οποία καλεί την DPLL(Γ, Ø) όπου το Γ είναι ένα σύνολο από λογικές προτάσεις το οποίο αντιστοιχεί στο Comp(P). Το σύνολο Γ μπορεί να υπολογιστεί με πολλούς τρόπους. Οι μόνες υποθέσεις μας είναι ότι το σύνολο Γ επεκτείνει το P, και ότι για κάθε υποσύνολο X ατόμων στο σύνολο Γ , το X ικανοποιεί το Γ αν και μόνο αν το σύνολο S ∩ X ικανοποιεί το Comp(Π).

Παρατηρούμε για κάποιο άτομο a στο P, τα άτομα a και ¬a μπορεί να μην ανήκουν στο σύνολο S. Κατά συνέπεια το σύνολο S ∪ {a} συνεπάγεται το Comp(P) και επίσης θα πρέπει να ελέγξουμε εάν η atoms(S ∪ {a}) είναι ένα σύνολο απαντήσεων του P. Όμως αυτός ο πρόσθετος έλεγχος δεν απαιτείται, όπως συνεπάγεται από την ακόλουθη πρόταση.

Προτάση 5.1[9]

Ας υποθέσουμε ότι το P είναι ένα λογικό πρόγραμμα. Το X και Y είναι δύο σύνολα από άτομα που ικανοποιούν το Comp(P).Αν ισχύει ότι το X ⊂ Y τότε το Y δεν είναι σύνολο απαντήσεων του P.

Από την πιο πάνω πρόταση και το γεγονός ότι κάθε σύνολο απαντήσεων είναι επίσης μια λογική απάντηση στο Comp(P) συμπεραίνουμε την ορθότητα του αλγορίθμου.

Επιπλέον ο αλγόριθμος ASP-SAT(P) εκτελεί την αναζήτηση στο Comp(P) και έτσι δεν εισαγάγει οποιεσδήποτε πρόσθετες μεταβλητές εκτός από εκείνες που απαιτούνται τελικά για τη μετάφραση του P σε Comp(P) . Ο αλγόριθμος ASP-SAT(P) μπορεί να τροποποιηθεί εύκολα για τον υπολογισμό όλων των συνόλων απαντήσεων του P. Είναι αρκετό να τροποποιηθεί η συνάρτηση test(S,P),έτσι ώστε να επιστραφεί ψευδές και στην περίπτωση που τα άτομα (S) είναι ένα σύνολο απαντήσεων.

Παρακάτω παρατίθεται ο αλγόριθμος ASP-SAT(P)

Ο αλγόριθμος ASP-SAT(P)[9]

function ASP-SAT(P)
return DPLL(CNF(Comp(P)), Ø ,P);

function DPLL(Γ, S, P)
if (Γ= Ø) then return test(S, Π);
if (Ø ∈ Γ) then return False;
if ({l} ∈ Γ) then return DPLL(assign(l, Γ), S ∪{l},Π);
p := an atom occurring in Γ
return DPLL(assign(p, Γ), S ∪ {p}, Π) or
DPLL(assign(¬p,Γ), S ∪ {¬p},Π).

5.6 Τροποποίηση της συνάρτησης lookahead του Smodels για επίτευξη γενικευμένης συνέπειας.

Ορισμός 5.3: Γενικευμένη τριαδική απαλοιφή (Hyper ternary resolution)

Η γενικευμένη τριαδική απαλοιφή είναι ένας λογικός κανόνας συμπεράσματος, ο οποίος περιλαμβάνει ένα βήμα γενικής απαλοιφής, δηλαδή ένα βήμα συμπεράσματος που περιλαμβάνει περισσότερες από δύο προτάσεις εισόδου. Παίρνει ως εισαγωγή μια ενιαία n-αδική πρόταση, της μορφής (l_1, l_2, l_3, . . . , l_n), όπου (n >=3) και n - 1 τριαδικές προτάσεις της μορφής (¬l_i, l', l'') ,όπου(i = 1, . . ., n - 1). Παράγει ως έξοδο τη νέα τριαδική πρόταση (l', l'', l_n),όπως φαίνεται στο πιο κάτω παράδειγμα.

Παράδειγμα 5.6

Χρησιμοποιώντας τη γενικευμένη τριαδική απαλοιφή στη σύνθετη πρόταση Q = (a ∨ b ∨ c ∨ d) ∧ (g ∨ h ∨ ¬a) ∧ (g ∨ h ∨ ¬c) ∧ (g ∨ h ∨¬d) ∧ (g ∨ h ∨¬b) θα έχουμε τη νέα δυαδική πρόταση (g ∨ h).

Παράδειγμα 5.7

Εφαρμόζοντας γενικευμένη τριαδική απαλοιφή στη λογική πρόταση Q = (a ∨ b ∨ c ∨ d) ∧ (g ∨ h ∨ ¬a) ∧ (g ∨ h ∨ ¬c) ∧ (g ∨ h ∨¬d) ∧ (g ∨ h ∨¬b) ∧ (¬g ∨ ¬h ∨ q)

συμπεραίνουμε τη λογική πρόταση P= (g ∨ h) ∧ (¬g ∨ ¬h ∨ q).Ακολούθως εφαρμόζοντας δυαδική απαλοιφή στην πρόταση P συμπεραίνουμε τη λογική πρόταση q.

Οπότε συμπεραίνουμε ότι η πρόταση Q ικανοποιείται για q =αληθές. Παρατηρούμε ότι η γενικευμένη δυαδική απαλοιφή δεν μπορεί να εφαρμοστεί στην πρόταση Q.

Συμπέρασμα 5.3:

Το παράδειγμα 5.7 πιστοποιεί ότι η εφαρμογή γενικευμένης τριαδικής απαλοιφής έχει ως αποτέλεσμα να πάρουν τιμή περισσότερες μεταβλητές σε σύγκριση με την εφαρμογή της γενικευμένης δυαδικής απαλοιφής.

Ορισμός 5.4: Γενικευμένη απαλοιφή σε προτάσεις i μεταβλητών (Hyper i- resolution)

Η γενικευμένη απαλοιφή σε προτάσεις i μεταβλητών είναι ένας λογικός κανόνας συμπεράσματος, ο οποίος περιλαμβάνει ένα βήμα γενικής απαλοιφής, δηλαδή ένα βήμα συμπεράσματος που περιλαμβάνει περισσότερες από δύο προτάσεις εισόδου. Παίρνει ως εισαγωγή μια ενιαία n-αδική πρόταση, της μορφής (l_1, l_2, l_3, . . . , l_n), όπου (n >= i) και n - 1 προτάσεις i μεταβλητών της μορφής (¬l_κ,l'_1, . . . , $l'_{\iota-1}$) ,όπου(κ = 1, . . , n - 1). Παράγει ως έξοδο τη νέα πρόταση I μεταβλητών της μορφής (l'_1, . . . , $l'_{\iota-1}$, l_n), Αν το i = 2 τότε ονομάζεται γενικευμένη δυαδική απαλοιφή, αν το i = 3, τότε ονομάζεται γενικευμένη τριαδική απαλοιφή.

Παράδειγμα 5.8

Χρησιμοποιώντας τη γενικευμένη απαλοιφή i μεταβλητών στη σύνθετη πρόταση Q = (a ∨ b ∨ c ∨ d) ∧ (g ∨ h ∨ ¬a) ∧ (g ∨ h ∨ ¬c) ∧ (g ∨ h ∨¬d) ∧ (g ∨ h ∨¬b) θα έχουμε τη νέα δυαδική πρόταση (g ∨ h).

Παράδειγμα 5.9

Εφαρμόζοντας γενικευμένη απαλοιφή σε προτάσεις i μεταβλητών στη λογική πρόταση Q = (a ∨ b ∨ c ∨ d ∨ e) ∧ (s ∨ g ∨ h ∨ ¬a) ∧ (s ∨ g ∨ h ∨ ¬b) ∧ (s ∨ g ∨ h ∨¬c) ∧ (s ∨ g ∨ h ∨ ¬d) ∧ (s ∨ g ∨ h ∨¬e) ∧ (¬s ∨ ¬g ∨ ¬h ∨ q) συμπεραίνουμε τη λογική πρόταση P= (s ∨ g ∨ h) ∧ (¬s ∨ ¬g ∨ ¬h ∨ q).Ακολούθως εφαρμόζοντας τριαδική απαλοιφή στην πρόταση P συμπεραίνουμε τη λογική πρόταση q.Οπότε συμπεραίνουμε ότι η πρόταση Q ικανοποιείται για q =αληθές. Παρατηρούμε ότι η γενικευμένη τριαδική απαλοιφή δεν μπορεί να εφαρμοστεί στην πρόταση Q.

Συμπέρασμα 5.4:

Το παράδειγμα 5.9 πιστοποιεί ότι η εφαρμογή γενικευμένης απαλοιφής σε προτάσεις ι μεταβλητών έχει ως αποτέλεσμα να πάρουν τιμή περισσότερες μεταβλητές σε σύγκριση με την εφαρμογή της γενικευμένης απαλοιφής σε προτάσεις i – 1 μεταβλητών.

Συμπέρασμα 5.5:

Γνωρίζουμε ότι η ανίχνευση μεταβλητής αποτυχίας είναι ένα βήμα lookahead με μοναδιαία διάδοση. Επίσης η ανίχνευση μεταβλητής αποτυχίας που εφαρμόζεται στον αλγόριθμο DPLL είναι ισοδύναμη με τη γενικευμένη δυαδική απαλοιφή. Από τα πιο πάνω και το συμπέρασμα 5.4 συμπεραίνουμε ότι η εφαρμογή γενικευμένης απαλοιφής σε προτάσεις i μεταβλητών είναι ισοδύναμη με την τροποποίηση της συνάρτησης lookahead του Smodels έτσι ώστε να αναθέτει τιμή δοκιμαστικά σε οποιεσδήποτε i -1 μεταβλητές. Για αυτό το λόγο αν θέλουμε να εφαρμόσουμε γενικευμένη απαλοιφή σε προτάσεις i μεταβλητών στη συνάρτηση lookahead του συστήματος Smodels θα πρέπει να τροποποιήσουμε τη συνάρτηση έτσι ώστε να αναθέτει τιμή δοκιμαστικά σε οποιεσδήποτε i -1 μεταβλητές.

Θεώρημα 5.1[6]

Έστω το πρόβλημα ικανοποίησης περιορισμών A(X,D,C). Για οποιοδήποτε 1 < i $\leq$ |X| , έστω { $y_1, \ldots, y_{i-1}$} το σύνολο οποιοδήποτε i – 1 μεταβλητών στο x και y_i έστω ότι είναι οποιαδήποτε i – οστή μεταβλητή. Για κάθε συνεπή ανάθεση Π = { $y_1 \rightarrow b_1, \ldots, y_{i-1} \rightarrow b_{i-1}$}, αν η μεταβλητή y_i δεν έχει κάποια ανάθεση συνεπή με το Π, τότε η συνάρτηση expand (P(A), { $y_1(b_1), \ldots, y_{i-1}(b_{i-1})$}) θα παράγει αντίφαση. Το P(A) είναι το λογικό πρόγραμμα που λαμβάνουμε αφού μεταφράσουμε το A με βάση τη βασική μετάφραση.

Το θεώρημα 5.1 αναφέρει ότι αν θέλουμε να πετύχουμε γενικευμένη συνέπεια βαθμού i στη συνάρτηση lookahead του Smodels, θα πρέπει να την τροποποιήσουμε έτσι ώστε να αναθέτει τιμή δοκιμαστικά σε i -1 μεταβλητές.

Λαμβάνοντας υπόψη το θεώρημα 5.1 ,τροποποιούμε τη συνάρτηση lookahead_once(P,A) σε lookahead_once*(P,A) έτσι ώστε να αναθέτει δοκιμαστικά σε i -1 μεταβλητές τιμή, όπου i είναι ο βαθμός της συνέπειας που εφαρμόζεται στο αντίστοιχο πρόβλημα ικανοποίησης περιορισμών. Πιο κάτω παρατίθεται η συνάρτηση lookahead(P,A)* η οποία καλεί την lookahead_once*(P,A)

Function lookahead*(P,A)

repeat
    A' : = A
    A : = lookahead_once*(P,A)
Until A = A'
return A.

Function lookahead_once*(P,A)
B : = Atoms(P) – Atoms(A)
B : = B ∪ not (B)
while B ≠ ∅ do
    take any $l_1, \ldots, l_{\iota-1}$ literals ∈ B
    A' = expand(P, A ∪ {$l_1, \ldots, l_{\iota-1}$})
    B : = B – A'
    If conflict(P,A') then
        return expand (P,A ∪ { not (l_1), . . . , not ($l_{\iota-1}$)})
    end if
end while
return A.

Στο παράδειγμα 5.10 συγκρίνουμε τη συνάρτηση lookahead** η οποία χρησιμοποιεί τη συνάρτηση lookahead_once** με την εφαρμογή της γενικευμένης συνέπειας.

Παράδειγμα 5.10

Το πιο κάτω πρόβλημα ικανοποίησης περιορισμών αποτελείται από τέσσερις μεταβλητές x, y, z και w όλες με πεδίο ορισμού {0,1,2} και έχει τους εξής περιορισμούς x ≠ y, x ≠ z, y ≠ z, y ≠ w, z ≠ w και w ≠ x.

Παρατηρούμε ότι το πιο πάνω πρόβλημα έχει συνέπεια ακμής αφού όλοι οι δυαδικοί περιορισμοί έχουν συνέπεια ακμής. Άρα η συνέπεια ακμής δεν μπορεί να αφαιρέσει τιμές από τα πεδία ορισμού των μεταβλητών του προβλήματος.

Παρατηρούμε ότι το πιο πάνω πρόβλημα έχει συνέπεια μονοπατιού αφού για

κάθε ζευγάρι μεταβλητών x_i και x_j ,όπου i ≠ j και για κάθε συνεπή ανάθεση στις μεταβλητές αυτές υπάρχει συνέπεια μονοπατιού. Άρα η συνέπεια μονοπατιού δεν μπορεί να αφαιρέσει τιμές από τα πεδία ορισμού των μεταβλητών του προβλήματος.

Παρατηρούμε ότι η συνέπεια δεν μπορεί να επεκταθεί σε 4 μεταβλητές. 'Αρα το πιο πάνω πρόβλημα δεν έχει 4 – συνέπεια(4 – consistent) .Αν εφαρμόσουμε συνέπεια σε 4 μεταβλητές παρατηρούμε ότι το πεδίο ορισμού της τέταρτης μεταβλητής είναι το κενό σύνολο. Για αυτό το λόγο το πιο πάνω πρόβλημα δεν έχει λύση.

Μεταφράζουμε το πιο πάνω πρόβλημα ικανοποίησης περιορισμών σε ένα κανονικό πρόγραμμα.

Μια μεταβλητή παίρνει ακριβώς μια τιμή από το πεδίο ορισμού της:

x(0)←not x(1),not x(2).

x(1)←not x(0),not x(2).

x(2)←not x(0),not x(1).

y(0)←not y(1),not y(2).

y(1)←not y(0),not y(2).

y(2)←not y(0),not y(1).

z(0)←not z(1),not z(2).

z(1)←not z(0),not z(2).

z(2)←not z(0),not z(1).

w(0)←not w(1),not w(2).

w(1)←not w(0),not w(2).

w(2)←not w(0),not w(1).

Οι επιτρεπτές πλειάδες στον περιορισμό x ≠ y αναπαριστούνται ως:

sat(c_1)←x(0),y(1).

sat(c_1)←x(0),y(2).

sat(c_1)←x(1),y(0).

sat(c_1)←x(1),y(2).

sat(c_1)←x(2),y(0).

sat(c_1)←x(2),y(1).

Οι επιτρεπτές πλειάδες στον περιορισμό x ≠ z αναπαριστούνται ως:

sat(c_2)←x(0),z(1).

sat(c_2)←x(0),z(2).

sat(c_2)←x(1),z(0).

sat(c_2)←x(1),z(2).

sat(c_2)←x(2),z(0).

sat(c_2)←x(2),z(1).

Οι επιτρεπτές πλειάδες στον περιορισμό y ≠ z αναπαριστούνται ως:

sat(c_3)←y(0),z(1).

sat(c_3)←y(0),z(2).

sat(c_3)←y(1),z(0).

sat(c_3)←y(1),z(2).

sat(c_3)←y(2),z(0).

sat(c_3)←y(2),z(1).

Οι επιτρεπτές πλειάδες στον περιορισμό y ≠ w αναπαριστούνται ως:

sat(c_4)←y(0),w(1).

sat(c_4)←y(0),w(2).

sat(c_4)←y(1),w(0).

sat(c_4)←y(1),w(2).

sat(c_4)←y(2),w(0).

sat(c_4)←y(2),w(1).

Οι επιτρεπτές πλειάδες στον περιορισμό z ≠ w αναπαριστούνται ως:

sat(c_5)←z(0),w(1).

sat(c_5)←z(0),w(2).

sat(c_5)←z(1),w(0).

sat(c_5)←z(1),w(2).

sat(c_5)←z(2),w(0).

sat(c_5)←z(2),w(1).

Οι επιτρεπτές πλειάδες στον περιορισμό w ≠ x αναπαριστούνται ως:

sat(c_6)←w(0),x(1).

sat(c_6)←w(0),x(2).

sat(c_6)←w(1),x(0).

sat(c_6)←w(1),x(2).

sat(c_6)←w(2),x(0).

sat(c_6)←w(2),x(1).

Αν η συνάρτηση lookahead δοκιμάσει να αναθέσει την τιμή 0 στη μεταβλητή x, την τιμή 0 στη μεταβλητή y και την τιμή 1 στην μεταβλητή z στο μεταφρασμένο λογικό πρόγραμμα, τότε θα έχουμε αντίφαση με αποτέλεσμα να προσθέσει τις μεταβλητές not x(0), not y(0) και not z(1) στο σύνολο απαντήσεων. Αυτό έχει ως αποτέλεσμα να προστεθούν οι μεταβλητές x(1),x(2),y(1),y(2),z(0) και z(2) στο σύνολο απαντήσεων. Αν έπειτα η lookahead δοκιμάσει την ανάθεση x = 1 , z =2 και w =1 ,τότε θα έχουμε αντίφαση με αποτέλεσμα να προσθέσει τις μεταβλητές not x(1) ,not z(2) και not w(1) στο σύνολο απαντήσεων. Όμως αυτό θα έχει ως αποτέλεσμα αντίφαση, οπότε συμπεραίνουμε ότι το πιο πάνω πρόβλημα δεν έχει λύση.

Συμπέρασμα 5.6

Παρατηρούμε ότι η εφαρμογή της lookahead η οποία αναθέτει δοκιμαστικά σε i=3 μεταβλητές τιμή στο πρόβλημα του παραδείγματος 5.10 έχει το ίδιο αποτέλεσμα

με την εφαρμογή γενικευμένης συνέπειας σε προτάσεις i = 4 μεταβλητών για το ίδιο πρόβλημα.

5.7 Εφαρμογή της γενικευμένης απαλοιφής σε προτάσεις i μεταβλητών στη συνάρτηση lookahead χρησιμοποιώντας τη μέθοδο ολοκλήρωσης του Clark.

Τροποποιουμε τη συνάρτηση lookahead_once(P,A) του συστήματος Smodels σε lookahead_once**(P,A) έτσι ώστε να εκτελεί :

α) S = CNF(Comp(P)). Δηλαδή αρχικά μεταφράζει το λογικό πρόγραμμα P σε πρόβλημα προτασιακής ικανοποιησιμότητας χρησιμοποιώντας τη μέθοδο ολοκλήρωσης του Clark. Ακολούθος μετατρέπει το μεταφρασμένο πρόβλημα σε κανονική συζευκτική μορφή και αποθηκεύει το αποτέλεσμα στο σύνολο S.

Εφόσον το σύνολο S δεν είναι κενό ή δεν υπάρχει αντίφαση ή δεν μπορεί να ανατεθεί οποιαδήποτε άλλη συνεπής τιμή στις μεταβλητές μετά από οποιοδήποτε βήμα.

β) Εφαρμόζει γενικευμένη απαλοιφή σε προτάσεις i μεταβλητών στο σύνολο S.

γ)Αν παράγονται νέες προτάσεις i -1 μεταβλητών που δημιουργούνται στο σύνολο S εφαρμόζει γενικευμένη απαλοιφή σε προτάσεις i – 1 μεταβλητών.

δ) Αν υπάρχουν οποιεσδήποτε μοναδιαίες προτάσεις στο σύνολο S εφαρμόζει μοναδιαία απαλοιφή στο σύνολο S.

ε) Κάθε φορά που εφαρμόζει ένα κανόνα απαλοιφής, επεκτείνει το σύνολο A των μεταβλητών που είδη τους έχει ανατεθεί τιμή στο σύνολο A’ , το οποίο περιέχει τις και τις νέες μεταβλητές που έχουν πάρει τιμή μετά την εφαρμογή του κανόνα απαλοιφής. Ακολούθως αν υπάρχει αντίφαση στο σύνολο A’, επιστρέφει την επέκταση του συνόλου A με βάση το πρόγραμμα P (expand(P,A)). Αν το σύνολο S είναι το κενό σύνολο επεκτείνει το σύνολο A’ με βάση το πρόγραμμα P και αποθηκεύει το αποτέλεσμα στο σύνολο A’’. Αν υπάρχει αντίφαση στο σύνολο A’’ με βάση το πρόγραμμα P, επιστρέφει την επέκταση του συνόλου A με βάση το πρόγραμμα P (expand(P,A)), αλλιώς επιστρέφει το σύνολο A’’.

Πιο κάτω παρατίθεται η συνάρτηση lookahead** η οποία καλεί επανειλημμένα τη lookahead_once**

Function lookahead(P,A,i)**

```
repeat
        A' : = A
        A : = lookahead_once*(P,A,i)
Until A = A'
return  A.
```

Function lookahead_once(P,A,i)**

```
S = CNF(Comp(P))

while S ≠ ∅

If there is a unit clause  in S
(A') :=  Unit_resolution(S,A)
    if  A' contains a contradiction then
        return expand(P,A)
        elseif  S is empty
                A'' := expand(P,A')
                If conflict(P,A'') then
                return expand (P,A)
                else
                return (A'')
                endif
    end if
end if
```

```
 (A') :=Hyper_resolution(S,A,i)
    if  A' contains a contradiction then
```

```
        return expand(P,A)
        elseif  S  is empty
                A'' := expand(P,A')
                If conflict(P,A'') then
                return expand(P,A)
                else
                return (A'')
                endif
    end if
```

If there are clauses with i – 1 variables in S then
i = i -1

return expand(P,A)

Κεφάλαιο 6

Επίλογος

6.1 Περίληψη

Σε αυτή τη διπλωματική μελετήσαμε τις διάφορες μεθόδους μελέτης και επίλυσης προβλημάτων περιορισμών, δηλαδή τον προγραμματισμό με βάση τους περιορισμούς, τον προγραμματισμό με βάση την προτασιακή ικανοποιησιμότητα και τον προγραμματισμό με βάση τα σύνολα απαντήσεων.

Είδαμε ότι η καθεμιά από αυτές τις μεθόδους χρησιμοποιεί μια παραλλαγή του αλγόριθμου οπισθοδρόμησης, ο οποίος επεκτείνει μια μερική λύση του προβλήματος σε μια ολοκληρωμένη λύση. Ο αλγόριθμος αυτός αναθέτει σε μια μεταβλητή μια τιμή από το πεδίο ορισμού της και ελέγχει αν αυτή η ανάθεση είναι συνεπής με τις τιμές που έχουν ανατεθεί είδη σε άλλες μεταβλητές. Αν αυτό ισχύει, τότε η διαδικασία συνεχίζεται αλλιώς ο αλγόριθμος οπισθοδρομεί και αναθέτει τιμή στην επόμενη μεταβλητή.

Για να μην χρειάζεται ο αλγόριθμος να δοκιμάσει όλους τους πιθανούς συνδυασμούς τιμών στις μεταβλητές του προβλήματος μέχρι να βρει την ανάθεση τιμών που ικανοποιεί όλους τους περιορισμούς ταυτόχρονα, κάθε μια από τις πιο πάνω μεθόδους χρησιμοποιεί κάποιες τεχνικές μείωσης του πεδίου αναζήτησης.

Πιο συγκεκριμένα ο προγραμματισμός με βάση τους περιορισμούς διατηρεί παράλληλα κάποιου είδους συνέπεια για να μειώσει τα πεδία ορισμού στα οποία αναζητεί τιμές. Οι τεχνικές συνέπειας αφαιρούν τις τιμές στα πεδία ορισμών που δεν είναι συνεπείς με τους περιορισμούς, με αποτέλεσμα ο χώρος αναζήτησης να μειώνεται. Ο προγραμματισμός με βάση την προτασιακή ικανοποιησιμότητα χρησιμοποιεί διάφορες περιπτώσεις του γενικού κανόνα συμπερασμού απαλοιφής για να αφαιρέσει λογικές μεταβλητές από τη βάση γνώσης του προβλήματος. Τέλος ο προγραμματισμός με βάση σύνολα απαντήσεων χρησιμοποιεί την ιδέα της lookahead, δηλαδή αναθέτει μια λογική τιμή σε κάποια μεταβλητή και αν αυτή η ανάθεση

οδηγήσει σε αντίφαση τότε είναι σίγουρο ότι στην μεταβλητή πρέπει να ανατεθεί η αντίθετη λογική τιμή, χωρίς όμως να γίνεται κάποια μείωση των πεδίων ορισμών κατά τη διάρκεια ή πριν την αναζήτηση. Ένα μερικό σύνολο απαντήσεων επεκτείνεται σε ένα ολοκληρωμένο σύνολο απαντήσεων με βάση τέσσερις κανόνες διάδοσης περιορισμών, όπως αυτοί υλοποιούνται στη συνάρτηση expand, την οποία καλεί η συνάρτηση lookahead.

Μεταφράζοντας την κάθε μέθοδο προγραμματισμού στις άλλες δύο και συγκρίνοντας τις διάφορες τεχνικές μείωσης του πεδίου αναζήτησης που χρησιμοποιούν για την επίλυση προβλημάτων περιορισμών καταλήξαμε στο συμπέρασμα ότι οι τεχνικές αυτές έχουν κοινά σημεία τόσο στην τεχνοτροπία τους όσο και στην αποδοτικότητα τους να μειώνουν το πεδίο αναζήτησης.

Είδαμε ότι αν θέλουμε να εφαρμόσουμε γενικευμένη απαλοιφή σε προτάσεις i μεταβλητών στη συνάρτηση lookahead του συστήματος Smodels θα πρέπει να τροποποιήσουμε τη συνάρτηση έτσι ώστε να αναθέτει τιμή δοκιμαστικά σε οποιεσδήποτε i -1 μεταβλητές.

Τέλος είδαμε ότι αν θέλουμε να πετύχουμε γενικευμένη συνέπεια βαθμού i στη συνάρτηση lookahead του Smodels, θα πρέπει να την τροποποιήσουμε έτσι ώστε να αναθέτει τιμή δοκιμαστικά σε i -1 μεταβλητές.

6.2 Μελλοντική εργασία

Στην διπλωματική αυτή τροποποιήσαμε θεωρητικά τη συνάρτηση lookahead του συστήματος Smodels για επίτευξη γενικευμένης συνέπειας ή για εφαρμογή γενικευμένης απαλοιφής σε προτάσεις i μεταβλητών. Επίσης εφαρμόσαμε θεωρητικά την γενικευμένη απαλοιφή σε προτάσεις i μεταβλητών στη συνάρτηση lookahead χρησιμοποιώντας τη μέθοδο ολοκλήρωσης του Clark.

Μια επέκταση της διπλωματικής αυτής είναι η υλοποίηση στο σύστημα Smodels της τροποποιημένης συνάρτησης lookahead έτσι ώστε να γίνει εφικτή η γενίκευση συνέπειας.

Βιβλιογραφία

[1].Vladimir Lifschitz: Introduction το Answer Set Programming.University of Texas at Austin. Draft.

[2].Romuald Debruyne and Christian Bessiere: Domain Filtering Consistencies. J. Artif. Intell. Res. (JAIR) 14: 205-230 (2001).

[3].Ilkka Niemela and Mirostaw Truszczynski: Answer-Set Programming:a Declarative knowledge Representation Paradigm.ESSLLI 2001-Logic and Computing:an introductory course.

[4].Toby Walsh: SAT V CSP. CP 2000: 441-456.

[5].Fahiem Bacchus: Enhancing Davis Putnam with Extended Binary Clause Reasoning. AAAI/IAAI 2002: 613-619.

[6].Jia-Huai You and Guiwen Hou: Arc-Consistency + Unit Propagation = Lookahead. ICLP 2004: 314-328.

[7]Jia-Huai You, Guohua Liu,Li Yan Yuan and Curtis Onuczko: Lookahead in Smodels Compared to Local Consistencies in CSP. LPNMR 2005: 266-278.

[8]Christian Bessiere, Emmanuel Hebrand and Toby Walsh: Local Consistencies in SAT. SAT 2003: 299-314.

[9]Enrico Giunchiglia, Yuliya Lierler and Marco Maratea: SAT-Based Answer Set Programming. AAAI 2004: 61-66