Μέρος Β’

Εξαγωγή συμπερασμάτων προς τα εμπρός για προτάσεις Horn Προτασιακής Λογικής

- *Literal.java*

Η συγκεκριμένη κλάση, παρέχει έναν βασικό τρόπο αναπαράστασης ενός συμβόλου/όρου (πρακτικά μιας προτασιακής μεταβλητής). Λόγω της φύσης των προτάσεων Horn και δεδομένου ότι θα χρησιμοποιηθούν μόνο οριστικές προτάσεις, η άρνηση ενός συμβόλου δεν είναι απαραίτητη αλλά έχει διατηρηθεί στην κλάση για λόγους πληρότητας.

- *HornClause.java*

Η κλάση χρησιμοποιείται για την αναπαράσταση μιας οριστικής πρότασης Horn που είναι είτε της μορφής True -> A (γεγονός) είτε της μορφής A ∧ B -> C (κανόνας). Οι ιδιότητες της κλάσης είναι μια λίστα που περιέχει όρους οι οποίοι αποτελούν της προϋποθέσεις ενός κανόνα (premises) -ένα γεγονός έχει κενή λίστα γιατί δεν έχει προϋποθέσεις- καθώς και έναν ακόμη όρο (conclusion) που αναπαριστά το συμπέρασμα του κανόνα (αν μια πρόταση είναι γεγονός, τότε έχει μόνο συμπέρασμα). Ο κατασκευαστής της κλάσης αρχικοποιεί κατάλληλα τις ιδιότητες, σε περίπτωση που ο προς δημιουργία τύπος είναι γεγονός σημαίνει ότι σαν όρισμα στη θέση των προϋποθέσεων θα δωθεί η τιμή null. Για λόγους συμβατότητας με την υλοποίηση του forward chaining ο κατασκευαστής αρχικοποιεί το πεδίο premises κάθε γεγονότος με μια κενή λίστα. Οι μέθοδοι που παρέχει η κλάση, είναι μέθοδοι πρόσβασης στις προυποθέσεις και στο συμπέρασμα, μέθοδος αναγνώρισης γεγονότος καθώς και μια *toString()* για την αναπαράσταση μιας πρότασης Horn, λαμβάνοντας υπόψιν και το είδος της (γεγονός ή κανόνας).

- *ForwardChaining.java*

Η συγκεκριμένη κλάση χρησιμοποιείται για την υλοποίηση του αλγόριθμου ForwardChaining. Ένα αντικείμενο ForwardChaining έχει τρεις ιδιότητες, μια λίστα που αναπαριστά την βάση γνώσης, ένα HashSet (inferred) στο οποίο αποθηκεύονται όσοι όροι έχουν εξαχθεί και μια ουρά (agenda) στην οποια αποθηκεύονται οι προς εξέταση όροι και αρχικώς περιέχει όλα τα γεγονότα. Ο κατασκευαστής της κλάσης αρχικοποιεί τα αντικείμενα αυτά και γεμίζει την ουρά (agenda), διατρέχοντας την βάση γνώσης και ελέγχοντας αν ο προς εξέταση τύπος είναι γεγονός, με την μέθοδο *isFact()* της κλάσης HornClause.

Μέθοδος isEntailed(String query):

Η συγκεκριμένη μέθοδος, είναι υπεύθυνη για την υλοποίηση του αλγορίθμου ForwardChaining. Σκοπός της είναι να ελέγξει αν η είσοδος που της έχει δωθεί μπορεί να εξαχθεί από την δεδομένη βάση γνώσης ενός αντικειμένου ForwardChaining. Εφόσον έχουν αρχικοποιηθεί οι δομές δεδομένων του αντικειμένου, εξετάζει την ουρά (agenda) μέχρις ότου αυτή αδειάσει. Για κάθε όρο που εξάγεται από την agenda, η οποία αρχικώς περιέχει όλα τα γεγονότα, ελέγχει αρχικά αν ισούται με το ζητούμενο query (έλεγχος ονόματος εφόσον δεν μπορεί να είναι αρνητικός ο όρος) και αν είναι τότε επιστρέφει *True* και τερματίζει. Στη συνέχεια ελέγχεται αν ο προς εξέταση όρος έχει ήδη εξαχθεί και αν δεν έχει τότε εισάγεται στο σύνολο inferred. Ο έλεγχος αυτός, παρ' όλο που μπορεί να μην φαίνεται σημαντικός (εφόσον το inferred είναι σύνολο, αν ο όρος υπήρχε ήδη δεν θα εισαγόταν) γλιτώνει από την διπλή επεξεργασία όρων που έχουν ήδη εξαχθεί κάτι το οποίο μειώνει την πολυπλοκότητα της μεθόδου και κρατά την ουρά (agenda) σωστά ενημερωμένη αποφεύγοντας την πολλαπλή ύπαρξη όρων. Σε σπάνιες περιπτώσεις, αν η agenda γεμίσει με τους ίδιους όρους, το σύστημα μπορεί να εισέλθει σε έναν άπειρο βρόχο (ή να καθυστερήσει υπερβολικά). Μετά, για κάθε πρόταση Horn της βάσης γνώσης ελέγχεται αν οι υποθέσεις της περιέχουν τον προς εξέταση όρο, και αν ναι τότε αυτός αφαιρείται από την λίστα προϋποθέσεων της πρότασης Horn με την χρήση της μεθόδου *remove* (αφαιρεί το πρώτο στοιχείο που ισούται με τον προς εξέταση όρο που περνάμε ως όρισμα, με βάση τη μέθοδο *equals* της κλάσης Literal). Τέλος, ελέγχει αν όλες οι προϋποθέσεις της πρότασης Horn έχουν ικανοποιηθεί (όταν η λίστα των προϋποθέσεων της πρότασης είναι κενή) και αν αυτό ισχύει τότε ελέγχει αν τόσο το σύνολο των όρων που έχουν εξαχθεί (inferred) όσο και η ουρά με τους προς εξέταση όρους (agenda) περιέχουν ήδη το συμπέρεσμα. Ο λόγος που γίνεται είναι για να αποφευχθεί να επανεξεταστεί ένας όρος που βρίσκεται ήδη στην agenda (αν γινόταν θα αυξάναμε την πολυπλοκότητα της μεθόδου) ή/και έχει ήδη εξαχθεί (δηλαδή βρίσκεται στο inferred) και συνεπώς δεν έχει νόημα να μπει ξανά στην agenda για να επανεξεταστεί. Εφόσον οι προυποθέσεις έχουν ικανοποιηθεί και το συμπέρασμα της πρότασης δεν περιέχεται ήδη σε καμία από της δύο δομές δεδομένων τότε εισάγεται στην agenda για να εξεταστεί σε επόμενη επανάληψη. Σε κάθε κρίσιμο σημείο της επεξεργασίας του αλγορίθμου, γίνονται και οι αντίστοιχες εκτυπώσεις για να ενημερώνεται ο χρήστης για την εξέλιξη της διαδικασίας.

- *Main.java*

Η συγκεκριμένη κλάση, περιλαμβάνει το κυρίως πρόγραμμα που υλοποιεί τον αλγόριθμο Forward Chaining για προτάσεις Horn Προτασιακής λογικής. Το πρόγραμμα εκκινεί ζητώντας από τον χρήστη να εισάγει το μονοπάτι προς το αρχείο κειμένου που είναι αποθηκευμένη η Βάση Γνώσης. Διαβάζει επαναληπτικά το αρχείο, απομονώνοντας κάθε του γραμμή και ελέγχοντας,

μέσω της μεθόδου *processClause*, αν είναι μια έγκυρη πρόταση Horn. Στη συνέχεια εκτυπώνεται η Βάση Γνώσης και δημιουργείται ένα αντικείμενο Forward Chaining με βάση την συγκεκριμένη βάση γνώσης. Εφόσον ο χρήστης εισάγει τον τύπο που θέλει να ελέγξει, εκτελείται, μέσω της μεθόδου *isEntailed* του αντικειμένου, ο αλγόριθμος Forward Chaining. Εφόσον εκτελεστεί ο αλγόριθμος, τυπώνεται το αποτέλεσμα, αν εξάγεται δηλαδή η είσοδος του χρήστη από την δεδομένη βάση γνώσης και ανεξαρτήτως αποτελέσματος, τυπώνεται και ο χρόνος που χρειάστηκε για να εκτελεστεί ο αλγόριθμος.

Μέθοδος processClause(String line, List<HornClause> knowledgeBase):

Η μέθοδος αυτή ελέγχει αν μια γραμμή του αρχείου είναι μια έγκυρη πρόταση Horn, αν είναι την μετατρέπει σε ένα αντικείμενο HornClause και την προσθέτει στην Βάση Γνώσης. Διαχωρίζει κάθε γραμμή χρησιμοποιώντας ως διαχωριστικό το σύμβολο της συνεπαγωγής (->) και προσθέτει τα διασπασμένα μέρη της γραμμής σε έναν πίνακα. Αν το μέγεθος του πίνακα είναι 1, τότε σημαίνει πως δεν είχαμε διάσπαση, δηλαδή ο τύπος δεν περιείχε το σύμβολο της συνεπαγωγής, άρα είναι, πιθανώς, ένα γεγονός. Συνεπώς, ο τύπος θα είναι έγκυρος αν αναπαρίσταται από κάποιο γράμμα, είτε πεζό είτε κεφαλαίο, της αγγλικής αλφάβητου, αν είναι έγκυρος τότε δημιουργείται ένα νέο αντικείμενο Literal με όνομα το γράμμα της αλφάβητου και στη συνέχεια, με βάση το αντικείμενο Literal, δημιουργεί ένα νέο αντικείμενο HornClause το οποιο προστίθεται στην Βάση Γνώσης. Αν το μέγεθος του πίνακα είναι 2, τότε έχει πραγματοποιηθεί διάσπαση, άρα ο τύπος ενδεχομένως αναπαριστά ένα κανόνα. Το πρώτο μέρος της διασπασμένης γραμμής αναπαριστά τις προϋποθέσεις, συνεπώς διασπάται με βάση το διαχωριστικό της σύζευξης (^). Ένας κανόνας μπορεί να έχει μια ή περισσότερες προϋποθέσεις, οπότε αν υπάρχει διάσπαση έχουμε πολλές ενώ αν δεν υπάρχει έχουμε μια. Κάθε μια προϋπόθεση καθώς και το δεύτερο μέρος της διασπασμένης γραμμής, που αναπαριστά το συμπέρεσμα, επεξεργάζεται με τον ίδιο τρόπο που επεξεργάστηκε και το γεγονός παραπάνω, ως ένα μεμονωμένο σύμβολο. Ανάλογα με τον αν είναι έγκυρο, δημιουργείται για κάθε ένα σύμβολο ένα νέο αντικείμενο Literal και στο τέλος δημιουργείται ο τελικός τύπος HornClause με μια λίστα από Literal που αναπαριστούν τις προυποθέσεις και το μοναδικό Literal που αναπαριστά το συμπέρασμα και ο τύπος προστίθεται στην Βάση Γνώσης.

Πειραματικές Εκτελέσεις

Βάση Γνώσης (παράδειγμα διαφανειών)

A A ^ P -> L

B B ^ L -> M P -> Q

A ^ B -> L L ^ M -> P

- Ερώτηση: Εξάγεται ως συμπέρασμα ο Q, ο αλγόριθμος απάντα ορθά ΝΑΙ και αποκρίνεται σε 0.022 δευτερόλεπτα

Starting Forward Chaining...

Processing literal: A

Inferred: A

Inferred set: [A]

Literal A satisfies a premise in clause: A ^ B -> L

Literal A satisfies a premise in clause: A ^ P -> L

Processing literal: B

Inferred: B

Inferred set: [A, B]

Literal B satisfies a premise in clause: B -> L

All premises satisfied. Adding conclusion to agenda: L

Literal B satisfies a premise in clause: B ^ L -> M

Processing literal: L

Inferred: L

Inferred set: [A, B, L]

Literal L satisfies a premise in clause: L -> M

All premises satisfied. Adding conclusion to agenda: M

Literal L satisfies a premise in clause: L ^ M -> P

Processing literal: M

Inferred: M

Inferred set: [A, B, L, M]

Literal M satisfies a premise in clause: M -> P

All premises satisfied. Adding conclusion to agenda: P

Processing literal: P

Inferred: P

Inferred set: [P, A, B, L, M]

Literal P satisfies a premise in clause: P -> L

Literal P satisfies a premise in clause: P -> Q

All premises satisfied. Adding conclusion to agenda: Q

Processing literal: Q

Answer: Yes, 'Q' is entailed.

- Ερώτηση: Εξάγεται ως συμπέρασμα ο S, ο αλγόριθμος απάντα ορθά ΟΧΙ και αποκρίνεται σε 0.013 δευτερόλεπτα.

Starting Forward Chaining...

Processing literal: A

Inferred: A

Inferred set: [A]

Literal A satisfies a premise in clause: A ^ B -> L

Literal A satisfies a premise in clause: A ^ P -> L

Processing literal: B

Inferred: B

Inferred set: [A, B]

Literal B satisfies a premise in clause: B -> L

All premises satisfied. Adding conclusion to agenda: L

Literal B satisfies a premise in clause: B ^ L -> M

Processing literal: L

Inferred: L

Inferred set: [A, B, L]

Literal L satisfies a premise in clause: L -> M

All premises satisfied. Adding conclusion to agenda: M

Literal L satisfies a premise in clause: L ^ M -> P

Processing literal: M

Inferred: M

Inferred set: [A, B, L, M]

Literal M satisfies a premise in clause: M -> P

All premises satisfied. Adding conclusion to agenda: P

Processing literal: P

Inferred: P

Inferred set: [P, A, B, L, M]

Literal P satisfies a premise in clause: P -> L

Literal P satisfies a premise in clause: P -> Q

All premises satisfied. Adding conclusion to agenda: Q

Processing literal: Q

Inferred: Q

Inferred set: [P, A, Q, B, L, M]

Answer: No, 'S' is not entailed.

Εξαγωγή συμπερασμάτων προς τα εμπρός για προτάσεις Horn Κατηγορηματικής Λογικής

- *Term.java*

Η κλάση αυτή χρησιμοποιείται για να μοντελοποιήσει την απλούστερη οντότητα της γραμματικής της Κατηγορηματικής Λογικής, δηλαδή τον όρο. Κάθε ορός είναι είτε μια μεταβλητή, είτε μια σταθερά. Ο διαχωρισμός τους πραγματοποιείται με την χρήση μιας λογικής μεταβλητής που αρχικοποιείται κατά την δημιουργία του αντικειμένου. Ο διαχωρισμός αυτός είναι πολύ σημαντικός, καθώς παίζει τον βασικότερο ρόλο στην διαδικασία της ενοποίησης διότι οι μεταβλητές μπορούν να αντικατασταθούν από άλλες μεταβλητές ή σταθερές ενώ οι σταθερές προφανώς παραμένουν ως έχουν.

- *Predicate.java*

Η κλάση αυτή χρησιμοποιείται για να μοντελοποιήσει τον απλούστερο, ατομικό, τύπο (οι όροι δεν είναι τύποι) της Κατηγορηματικής Λογικής, δηλαδή το κατηγόρημα. Ένα κατηγόρημα, έχει τόσους όρους όσους και η τάξη του. Συνεπώς, για την δημιουργία του, πέρα από το όνομά του θέλουμε και μια λίστα από έναν ή περισσότερους όρους.

- *HornClause.java*

Η κλάση αυτή χρησιμοποιείται για να μοντελοποιήσει τις προτάσεις Horn, οι οποίες αποτελούν τους τύπους πάνω στους οποίους θα βασίσουμε τον αλγόριθμο Forward Chaining. Μια πρόταση αποτελεί έναν κανόνα, ο οποίος έχει καμία, μία ή περισσότερες προϋποθέσεις και πάντοτε ένα συμπέρεσμα. Αν ο κανόνας δεν περιέχει υποθέσεις τότε έχουμε ένα γεγονός, δηλαδή ένα συμπέρασμα, κάτι το οποίο ισχύει πάντα, αλλιώς ανάλογα με το πλήθος των υποθέσεων (κατηγορημάτων) αρχικοποιούμε κατάλληλα και την λίστα κάθε κανόνα. Συνεπώς, η κλάση αυτή πέρα από την κατασκευή των άνω αντικειμένων, παρέχει και μεθόδους πρόσβασης στα συστατικά τους μέρη καθώς και έλεγχο για το αν ένας κανόνας είναι γεγονός, πρακτική πολύ χρήσιμη κατά την εκκίνηση του αλγορίθμου Forward Chaining.

- *Unifier.java*

Η κλάση αυτή χρησιμοποιείται για την υλοποίηση της διαδικασίας της ενοποίησης μεταξύ δύο όρων ή μεταξύ δύο λιστών από όρους. Η βασική λειτουργία αφορά τις μεθόδους που προσπαθούν να ενοποιήσουν δύο όρους. Δύο όροι ενοποιούνται αν τουλάχιστον ένας από τους δύο είναι μεταβλητή. Σε περίπτωση που και οι δύο είναι σταθερές, υπάρχει μια τετριμμένη περίπτωση ενοποίησης που ενοποιεί δύο σταθερές με το ίδιο όνομα, δηλαδή την ίδια σταθερά. Εφόσον κάποιος εκ των όρων είναι μεταβλητή, τότε πραγματοποιείται ενοποίηση, αντικαθιστώντας το όνομα της πρώτης μεταβλητής με το όνομα του δεύτερου όρου(συνήθως σταθερά αλλά μπορεί να είναι και μεταβλητή). Οι μέθοδοι που υλοποιούν τις ίδιες λειτουργίες σε λίστα όρων, εφόσον διασφαλίσουν ότι οι δύο λίστες έχουν το ίδιο πλήθος από όρους, απλά διατρέχουν κάθε όρο στις δύο λίστες και ελέγχουν αν μπορούν να ενοποιηθούν ζεύγη όρων (ένας όρος από την πρώτη και ένας από την δεύτερη λίστα) καλώντας κάθε φορά τις μεθόδους που χειρίζονται δύο όρους. Η κλάση αυτή διατηρεί ένα *HashMap* με όλες τις ενοποιήσεις που έχουν πραγματοποιηθεί και επίσης κάθε φορά που πάει να γίνει μια αντικατάσταση, ελέγχεται η μεταβλητή προς ενοποίηση έχει προηγουμένως αντιστοιχηθεί σε κάποια άλλη σταθερά. Αν αυτό ισχύει τότε δεν μπορεί να ξανά-αντιστοιχηθεί και απλώς επιστρέφεται η προηγούμενη αντιστοίχιση. Ο λόγος που δεν επιτρέπονται πολλαπλές αντικαταστάσεις μιας μεταβλητής είναι ότι αν σε έναν κανόνα, είχαμε πολλαπλή εμφάνιση μιας μεταβλητής σε διαφορετικά κατηγορήματα και για κάθε κατηγόρημα είχαμε μια

διαφορετική ενοποίηση της μεταβλητής τότε ο κανόνας θα έπαυε να είναι συνεπής. Για την διατήρηση της συνέπειας του κανόνα, θα έπρεπε για κάθε ενοποίηση να δημιουργούταν διαφορετική εκδοχή του κανόνα, αυξάνοντας σημαντικά την πολυπλοκότητα του Forward Chaining και οδηγώντας τον σε πιθανές αντιφάσεις.

- *ForwardChaining.java*

Η βασική λογική και δομή της κλάσης είναι ίδια με την αντίστοιχη που υλοποιεί τον αλγόριθμο σε Προτασιακή λογική. Το μόνο επιπλέον στοιχείο είναι ένα αντικείμενο *Unifier* που θα χρησιμοποιηθεί για την ενοποίηση όρων καθώς και η προσθήκη ενός άνω ορίου βημάτων που σε συνδυασμό με τον μετρητή που αυξάνεται κατά την εκτέλεση του αλγορίθμου, προσθέτουν μια επιπλέον συνθήκη τερματισμού. Ο λόγος που γίνεται αυτό είναι η ιδιότητα της ημι-αποκρισιμότητα της Κατηγορηματικής Λογικής. Εφόσον ένας ατομικός τύπος μπορεί να εξαχθεί από την βάση γνώσης, ο αλγόριθμος θα το βρει και θα τερματίσει, αν όμως ο τύπος δεν εξάγεται, είναι πολύ πιθανό ο αλγόριθμος να τρέχει επ΄ άπειρον. Για την λύση αυτού, αν για έναν τύπο ο αλγόριθμος χρειάζεται να πραγματοποιήσει πάνω από 5000 επαναλήψεις, θεωρεί ότι λογικά δεν εξάγεται από την Βάση Γνώσης και τερματίζει.

Μέθοδος isEntailed(Predicate query):

Η συγκεκριμένη μέθοδος, είναι υπεύθυνη για την υλοποίηση του αλγορίθμου ForwardChaining. Σκοπός της είναι να ελέγξει αν η είσοδος που της έχει δωθεί μπορεί να εξαχθεί από την δεδομένη βάση γνώσης ενός αντικειμένου ForwardChaining. Όσο η agenda (ουρά με τα προς εξέταση κατηγορήματα) είναι μη κενή και ο αριθμός των βημάτων δεν έχει ξεπεράσει το άνω όριο, η μέθοδος εκτελεί τα εξής. Το πρώτο κατηγόρημα αφαιρείται από την agenda και γίνεται προσπάθεια ενοποίησης με το ερώτημα προς εξέταση (είσοδος χρήστη), αν η ενοποίηση είναι επιτυχής (είτε δηλαδή ενοποιήθηκε η μεταβλητή του κατηγορήματος με την σταθερά ή την μεταβλητή του ερωτήματος είτε το κατηγόρημα είχε την ίδια σταθερά με το ερώτημα) τότε το ερώτημα εξάγεται από την βάση γνώσης και ο αλγόριθμος τερματίζει επιτυχώς. Εφόσον δεν ήταν επιτυχής η ενοποίηση και εφόσον το κατηγόρημα δεν έχει ήδη εξαχθεί (αν έχει εξαχθεί δεν υπάρχει λόγος για επιπλέον έλεγχο, συνεπώς ο αλγόριθμος μεταβαίνει στην επόμενη επανάληψη) εκτελούνται τα εξής. Για κάθε κανόνα στην βάση γνώσης, διασπά τον κανόνα στα επιμέρους συστατικά του (προϋποθέσεις και συμπέρασμα) και από την λίστα των προϋποθέσεων του δημιουργεί έναν *iterator* για να διατρέξει άμεσα όλες τις προυποθέσεις του κατηγορήματος χωρίς να υπάρξουν προβλήματα κατά την τροποποίηση της λίστας. Για κάθε προϋπόθεση στον *iterator* ελέγχεται αν οι όροι του μπορούν να ενοποιηθούν με τους όρους του κατηγορήματος, αν αυτό είναι εφικτό πραγματοποιείται η ενοποίηση και η προϋπόθεση αφαιρείται από την λίστα.

Κάθε φορά που ικανοποιείται μια προϋπόθεση, ελέγχεται αν η αρχική λίστα προϋποθέσεων του κανόνα είναι κενή, δηλαδή αν έχουν ικανοποιηθεί όλες οι προϋποθέσεις του, εφόσον αυτό ισχύει το συμπέρασμα του κανόνα προστίθεται στην agenda.

- *Main.java*

Η συγκεκριμένη κλάση, περιλαμβάνει το κυρίως πρόγραμμα που υλοποιεί τον αλγόριθμο Forward Chaining για προτάσεις Horn Κατηγορηματικής Λογικής. Η δομή της κλάσης και ο τρόπος επεξεργασίας του αρχείου και των επί μέρους κανόνων είναι ίδιος ακριβώς με την αντίστοιχη κλάση για την Προτασιακή Λογική.

Πειραματικές Εκτελέσεις

Βάση Γνώσης (παράδειγμα διαφανειών)

(American(x) ^ Weapon(y) ^ Sells(x, y, z) ^ Hostile(z)) => Criminal(x)

Missile(M1)

Owns(Nono, M1)

(Missile(x) ^ Owns(Nono, x)) => Sells(West, x, Nono)

Missile(x) => Weapon(x)

Enemy(x, America) => Hostile(x)

American(West)

Enemy(Nono, America)

- Ερώτηση: Εξάγεται ο ατομικός τύπος American(West), ο αλγόριθμος απάντα ορθά ΝΑΙ και αποκρίνεται σε 0.011 δευτερόλεπτα.

Starting Forward Chaining...

Processing Predicate: Missile(M1)

Inferred: Missile(M1)

Agenda: [Owns(Nono, M1), American(West), Enemy(Nono, America)]

Inferred Set: [Missile(M1)]

Unification successful for premise: American(x)

Unification successful for premise: Weapon(y)

Unification failed for premise: Sells(x, y, z)

Unification successful for premise: Hostile(z)

Unification successful for premise: Missile(x)

Unification failed for premise: Owns(Nono, x)

Unification successful for premise: Missile(x)

Adding to agenda: Weapon(M1)

Unification failed for premise: Enemy(x, America)

Processing Predicate: Owns(Nono, M1)

Inferred: Owns(Nono, M1)

Agenda: [American(West), Enemy(Nono, America), Weapon(M1)]

Inferred Set: [Owns(Nono, M1), Missile(M1)]

Unification failed for premise: American(x)

Unification failed for premise: Weapon(y)

Unification failed for premise: Sells(x, y, z)

Unification failed for premise: Hostile(z)

Unification failed for premise: Missile(x)

Unification successful for premise: Owns(Nono, x)

Unification failed for premise: Missile(x)

Unification failed for premise: Enemy(x, America)

Processing Predicate: American(West)

Unification successful with query: American(West)

Answer: Yes, 'American(West)' is entailed.

- Ερώτηση: Εξάγεται ο ατομικός τύπος Criminal(Nono), ο αλγόριθμος απάντα ορθά ΟΧΙ έχοντας φτάσεις στο μέγιστο όριο επαναλήψεων, αποκρίνεται σε 19.118 δευτερόλεπτα. (Δεν έχουμε συμπεριλάβει το output διότι από ένα σημείο κι έπειτα γίνεται επανάληψη).