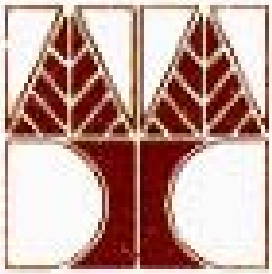
Ατομική Διπλωματική Εργασία

Μοντελοποίηση Προβλημάτων Προγραμματισμού Δράσεων Σε Προγραμματισμό Συνόλου Απαντήσεων

Ξένια Χριστοδούλου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δεκέμβριος 2007

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μοντελοποίηση Προβλημάτων Προγραμματισμού Δράσεων Σε Προγραμματισμό Συνόλου Απαντήσεων

Ξένια Χριστοδούλου

Επιβλέπων Καθηγητής
Δρ. Γιάννης Δημόπουλος

Η Ατομική αυτή Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου.

Δεκέμβριος 2007

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κ. Γιάννη Δημόπουλο, για την ευκαιρία που μου έδωσε να εργαστώ πάνω στο αντικείμενο της παρούσας εργασίας. Επίσης με τη καθοδήγηση του και την πολύτιμη βοήθεια του, κατάφερα να ολοκληρώσω την εργασία αυτή.

Ευχαριστώ,
Ξένια Χριστοδούλου

Περιεχόμενα:

Κεφάλαιο 1 - Εισαγωγή:

Κεφάλαιο 2 - Προγραμματισμός Δράσης(Planning Domain Definition Language) :

Κεφάλαιο 3 - Προγραμματισμός Συνόλου Απαντήσεων(Answer Set Programming):

Κεφάλαιο 4 – Lparse και Smodels:

Κεφάλαιο 5 - Μοντελοποίηση Προβλημάτων Προγραμματισμού Δράσης Σε Λογικό Προγραμματισμό:

Κεφάλαιο 6 – Χρονικοί Περιορισμοί:

Κεφάλαιο 7 – Πειραματικά Αποτελέσματα:

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή

Η παρούσα Ατομική Διπλωματική Εργασία έχει σαν αντικείμενο τη Μοντελοποίηση Προβλημάτων Προγραμματισμού Δράσης Σε Σύνολα Απαντήσεων. Δηλαδή ασχολείται με δυο διαφορετικά ειδή προγραμματισμού, πρώτον ο προγραμματισμός δράσης (PDDL) και δεύτερον ο προγραμματισμός συνόλου απαντήσεων (Answer Set Programming).

Αρχικά θα μελετήσω το προγραμματισμό δράσης, τα γενικά χαρακτηριστικά του, το τρόπο σκέψης του καθώς επίσης και τη δημιουργία πλάνου για ένα πρόβλημα.

Στη συνέχεια θα μελετήσω το προγραμματισμό συνόλου απαντήσεων, τα χαρακτηριστικά του και επίσης το τρόπο σκέψης του. Με αυτή τη γλώσσα θα ασχοληθώ στη πράξη χρησιμοποιώντας τα συστήματα SMODELS και CLASP για εξαγωγή αποτελεσμάτων και συμπερασμάτων. Τέλος, θα επιλέξω τρία προβλήματα προγραμματισμού δράσης, τα οποία θα μετατρέψω σε προγραμματισμό συνόλου απαντήσεων.

1.2 Σκοπός Διπλωματικής Εργασίας

Η Διπλωματική αυτή Εργασία σκοπό έχει την κωδικοποιήση τριών συγκεκριμένων προγραμμάτων προγραμματισμού δράσεων σε προγραμματισμό συνόλου απαντήσεων. Μετά την κωδικοποίηση σε προγραμματισμό συνόλου απαντήσεων, θα ακολουθήσει η κωδικοποίηση διαφόρων αρχικών καταστάσεων και τελικών σκοπών για κάθε ένα πρόβλημα, σύμφωνα πάντα με το κόσμο του προβλήματος, και με τη χρήση των συστημάτων Lparse, Smodels και Clasp θα πάρω αποτελέσματα.

1.3 Δομή Διπλωματικής Εργασίας

Η παρούσα Διπλωματική Εργασία με θέμα Μοντελοποίηση Προβλημάτων Προγραμματισμού Δράσης σε Σύνολα Απαντήσεων αποτελείται από οχτώ κεφάλαια.

Το πρώτο Κεφάλαιο είναι η εισαγωγή της Διπλωματικής Εργασίας.

Στο δεύτερο Κεφάλαιο, εξηγείται και αναλύεται η έννοια του Προγραμματισμού δράσεων, γενικά χαρακτηριστικά της γλώσσας προγραμματισμού δράσης, το πώς ορίζονται οι δράσεις στο προγραμματισμό δράσεων, και πως ορίζονται τα προβλήματα.

Στο τρίτο Κεφάλαιο, εξηγείται και αναλύεται η έννοια του προγραμματισμού συνόλου απαντήσεων, γενικά χαρακτηριστικά και συντακτικά χαρακτηριστικά της γλώσσας καθώς επίσης και το πως ορίζονται τα προβλήματα.

Στο τέταρτο Κεφάλαιο αναλύεται ο τρόπος λειτουργίας του Lparse, Smodels και Clasp και εξηγείται ο τρόπος που δίνουν τα αποτελέσματα.

Στο πέμπτο Κεφάλαιο περιγράφονται τα προβλήματα προγραμματισμού δράσεων και πως αυτά έχουν μετατραπεί σε προγράμματα προγραμματισμού συνόλου απαντήσεων.

Στο έκτο Κεφάλαιο κωδικοποιώ στα προβλήματα χρονικούς περιορισμούς και απλές προτιμήσεις.

Στο έβδομο Κεφάλαιο δίνονται τα πειραματικά αποτελέσματα των προβλημάτων, δηλαδή τα αποτελέσματα που θα πάρουμε βάζοντας διαφορετικά δεδομένα.

Στο όγδοο Κεφάλαιο υπάρχουν τα συμπεράσματα από όλη τη διαδικασία.

Κεφάλαιο 2

Προγραμματισμός Δράσης
(Planning Domain Definition Language)

2.1 Εισαγωγή στον Προγραμματισμό Δράσης

Πλάνο (planning) είναι ο στόχος που πηγάζει από μια σειρά δράσεων (actions) και θέλει να πετύχει ένα συγκεκριμένο σκοπό. Με άλλα λόγια, δεδομένων κάποιων δράσεων με γνωστά χαρακτηρίστικα, όπου με συγκεκριμένη σειρά και δεδομένης μιας αρχικής κατάστασης δίνουν ένα αποτέλεσμα το οποίο μπορούμε να το γνωρίζουμε εύκολα. Αντίθετα στα προβλήματα κλασσικού έλεγχου και ταξινόμησης, οι λύσεις είναι σύνθετες, άγνωστες και στη συνέχεια πρέπει να ανακαλυφθούν και να βελτιστοποιηθούν.

Η γλώσσα PDDL είναι μια γλώσσα προγραμματισμού δράσεων, όπου εμπνέεται από τα γνωστές φόρμουλες των STRIPS (**St**anford **R**esearch **I**nstitute **P**roblem **S**olver) όπου είναι αυτοματοποιημένο σύστημα δημιουργίας πλάνων. Είναι μια γλώσσα πολύ απλή συντακτικά, αφού εκφράζονται τα σημαντικά κάθε δράσης, δηλαδή οι συνθήκες που πρέπει να ισχύουν για να μπορεί να γίνει γεγονός μια δράση (preconditions) και τα αποτελέσματα μιας δράσης (post conditions). Δεδομένης μιας αρχικής κατάστασης καθώς επίσης και μιας τελικής, ένα πρόγραμμα δράσης θα δώσει ένα πλάνο, μια σειρά από δράσεις όπου δίνουν το τελικό, επιθυμητό αποτέλεσμα.

Στο προγραμματισμό δράσεων οπωσδήποτε πρέπει να ασχοληθούμε με δυο μέρη, πρώτον στο ορισμό δράσεων (domain) και δεύτερο το ίδιο το πρόβλημα (problem definition). Η σύνταξη της γλώσσας είναι πολύ τυποποιημένη και μπορεί εύκολα να καταλάβει κανείς την περιγραφή του προβλήματος (domain).

2.2 Γενικά χαρακτηριστικά τις γλώσσας προγραμματισμού δράσης (PDDL)

Ο ορισμός της γλώσσας προγραμματισμού δράσεων αποτελείται από δυο μέρη, ορισμό δράσεων (domain) και δεύτερο το ίδιο το πρόβλημα (problem definition). Παρόλα αυτά πολλοί προγραμματιστές ενώνουν τα δυο αυτά μέρη σε ένα αρχείο χωρίς να υπάρχει πρόβλημα.

Τα σχόλια σε ένα αρχείο προγραμματισμού δράσης, αρχίζουν με ελληνικό ερωτηματικό (;). Σε κάθε γραμμή όπου στην αρχή υπάρχει το ελληνικό ερωτηματικό ακολουθούν σχόλια μέχρι το τέλος της γραμμής.

Υπάρχουν οι απαιτήσεις για το προγραμματισμό δράσης όπου πρέπει να δηλώνονται σε κάθε πρόβλημα (domain). Σε ένα πρόβλημα προγραμματισμούς δράσης μπορεί να υποστηριχθεί μόνο ένα υποσύνολο απαιτήσεων. Τέτοιες είναι οι εξής:

:strips

Το πιο μεγάλο υποσύνολο του προγραμματισμού δράσης περιέχει μόνο strips

:equality

Το πρόβλημα χρησιμοποιεί κατηγόρημα = και ερμηνεύεται σαν ισότητα. equality.

:typing

Το πρόβλημα χρησιμοποιεί τύπους.

:adl

Το πρόβλημα χρησιμοποιεί μερικά ή όλα τα ADL. Για παράδειγμα διαζεύξεις και ποσοδείκτες στις προϋποθέσεις και στους στόχους.

2.3 Ορισμός των δράσεων του προβλήματος στον προγραμματισμό δράσης

Στον ορισμό των δράσεων του προβλήματος περιέχονται τα κατηγορήματα που ασχολείται το πρόβλημα, καθώς επίσης και δράσεις.

Η γενική μορφή ενός ορισμού προβλήματος φαίνεται πιο κάτω:

```
(define (domain DOMAIN_NAME)
   (:requirements [:strips] [:equality] [:typing] [:adl])
   (:predicates (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
                (PREDICATE_2_NAME [?A1 ?A2 ... ?AN])
          ...)
```

Τα στοιχεία που είναι μέσα στις αγκύλες [] είναι προαιρετικά. Μπροστά από τις παραμέτρους των κατηγορημάτων και των δράσεων υπάρχει το αγγλικό ερωτηματικό (?).

Οι παράμετροι που χρησιμοποιούνται στους ορισμούς των κατηγορημάτων δεν έχουν κάποια συνάρτηση που να καθορίζει τον αριθμό των χαρακτηριστικών που έχει κάθε κατηγόρημα.

```
(:action ACTION_1_NAME
      [:parameters (?P1 ?P2 ... ?PN)]
      [:precondition PRECOND_FORMULA]
      [:effect EFFECT_FORMULA]
    )

  (:action ACTION_2_NAME
    ...)

 ...)
```

Στις δράσεις φαίνονται τα κατηγορήματα που έχουν σχέση με τη δράση, τα οποία φαίνονται στο ορισμό του προβλήματος. Τα κατηγορήματα αυτά αποτελούν τις παραμέτρους των δράσεων. Στη συνέχεια ακολουθούν οι συνθήκες που πρέπει να ισχύουν για να συμβεί μια δράση (preconditions), καθώς και τα αποτελέσματα μια δράσης, δηλαδή οι αλλαγές που παρουσιάζονται στο κόσμο του προβλήματος (effects).

Μορφή συνθηκών:

Στα προβλήματα που χρησιμοποιούν STRIPS, η μορφή των συνθηκών μιας δράσης (preconditions) είναι η εξής:

(PREDICATE_NAME ARG1 ... ARG_N)

Οι παράμετροι κάθε συνθήκης πρέπει να είναι παράμετροι της ίδιας της δράσης. Το πιο πάνω μπορεί να θεωρηθεί και ένας άτομο τις συνθήκης. Η συνθήκη μπορεί να αποτελείται από τη σύζευξη αρκετών ατόμων, όπως φαίνεται το παράδειγμα πιο κάτω.

(and ATOM1 ... ATOM_N)

Στα προβλήματα που χρησιμοποιείται η ισότητα (equality) η μορφή ενός ατόμου είναι (= ARG1 ARG2). Χρησιμοποιείται και η αρνητική μορφή της ισότητας (not (= ARG1 ARG2)).

Στα προβλήματα που χρησιμοποιείται ADL μπορεί επιπρόσθετα να περιέχει άρνηση, σύζευξη ή και διάζευξη:

(not CONDITION_FORMULA)
(and CONDITION_FORMULA1...CONDITION_FORMULA_N)
(or CONDITION_FORMULA1 ... CONDITION_FORMULA_N)

Επισης μπορεί να περιέχονται και συνθήκες του τύπου:

(forall(?V1 ?V2...)CONDITION_FORMULA)

που αντιπροσωπεύει το καθολικό συντελεστή, δηλαδή για κάθε V να ισχύει η συνθήκη CONDITION_FORMULA

(exists (?V1 ?V2 ...) CONDITION_FORMULA)

που αντιπροσωπεύει το υπαρξιακό συντελεστή, δηλαδή υπάρχει V για το οποίο ισχύει η συνθήκη CONDITION_FORMULA.

Μορφή αποτελεσμάτων:

Στα προβλήματα που χρησιμοποιούν STRIPS, αν ισχύουν οι συνθήκες της δράσης τότε έχουμε σαν αποτέλεσμα να προστίθενται ή και να αφαιρούνται κάποια κατηγορήματα. Είναι πιθανόν τα αποτελέσματα να εκφραστούν σαν σύζευξη. Πιο κάτω φαίνονται πως αναπαριστώνται οι περιπτώσεις αυτές.

Άτομο το οποίο προστίθεται ορίζεται ως εξής:

(PREDICATE_NAME ARG1 ... ARG_N)

Άτομο το οποίο αφαιρείται ορίζεται ως εξής:

(not (PREDICATE_NAME ARG1 ... ARG_N))

Σύζευξη ατόμων σαν αποτέλεσμα ορίζεται ως εξής:

(and ATOM1 ... ATOM_N)

2.4 Ορισμός του προβλήματος στον προγραμματισμό δράσης

Στον ορισμό του προβλήματος περιέχονται τα αντικείμενα για το συγκεκριμένο πρόβλημα, η αρχική κατάσταση του κόσμου στον οποίο βρισκόμαστε και ο στόχος του προβλήματος.

Η μορφή ενός απλού προβλήματος είναι η εξής:

```
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects OBJ1 OBJ2 ... OBJ_N)
  (:init ATOM1 ATOM2 ... ATOM_N)
  (:goal CONDITION_FORMULA)
  )
```

Η περιγραφή της αρχική κατάσταση (:init) είναι απλά μια λίστα από όλα τα γεγονότα όπου είναι αληθής στην αρχική κατάσταση. Όλα τα άλλα άτομα είναι εξ' ορισμού είναι μη αληθές. Η περιγραφή του τελικού σκοπού (:goal) έχει ακριβώς τη ίδια μορφή με τη μορφή των συνθηκών στους ορισμούς των δράσεων. Όλα τα κατηγορήματα τα οποία χρησιμοποιούνται στην αρχική κατάσταση και στο σκοπό του προβλήματος είναι ορισμένα στον ορισμό των δράσεων.

Σε αντίθεση με τις συνθήκες των δράσεων όπου τα κατηγορήματα παίρνουν σαν παραμέτρους μεταβλητές, τα κατηγορήματα στη αρχική κατάσταση και στο τελικός σκοπός παίρνουν σαν παραμέτρους αντικείμενα ή σταθερά ονόματα.

Κεφάλαιο 3

Προγραμματισμός Συνόλου Απαντήσεων
(Answer Set Programming)

3.1 Εισαγωγή στον Προγραμματισμό Συνόλου Απαντήσεων (Answer Set Programming)

Ο Προγραμματισμός Συνόλου Απαντήσεων (Answer Set Programming) είναι ένα νέο είδος δηλωτικού και λογικού προγραμματισμού. Βασίζεται στην ιδέα της καθορισμένης σημασιολογίας μιας απάντησης (answer set semantics). Έχει τη δυνατότητα να υπολογίζει δύσκολα συνδυαστικά προβλήματα. Ο σκοπός του Προγραμματισμού Συνόλου Απαντήσεων είναι να αναπαραστήσει ένα υπολογιστικό πρόβλημα, ως ένα λογικό πρόβλημα οπού το σύνολο των απαντήσεων που θα υπολογιστούν θα αντιστοιχεί στις λύσεις του προβλήματος. Ο Προγραμματισμός Συνόλου Απαντήσεων διαφέρει από

τον παραδοσιακό λογικό προγραμματισμό αλλά και από τον λογικό προγραμματισμό σταθερών. Στο Προγραμματισμό Συνόλου Απαντήσεων αναπαρίστανται λύσεις σε ένα πρόβλημα δίνοντας το σύνολο απαντήσεων (answer set), ενώ στο παραδοσιακό λογικό προγραμματισμό αλλά και στον λογικό προγραμματισμό σταθερών αναπαρίστανται το σύνολο αντικαταστάσεων (answer substitutions). Η βασική προσέγγιση του Προγραμματισμού Συνόλου Απαντήσεων είναι να γράψουμε ένα πρόγραμμα στο οποίο θα μπορούν να υπάρχουν κανόνες επιλογής με χρήση σταθερών. Ο Προγραμματισμού Συνόλου Απαντήσεων επίσης συσχετίζεται πολύ με τον προγραμματισμό περιορισμών.

Ο Προγραμματισμός Συνόλου Απαντήσεων αναπαριστά ένα λογικό πρόγραμμα υπό μορφή περιορισμών σ'ένα σύνολο από κανόνες προτασιακής λογικής δίνοντας τιμές αληθείας στα άτομα. Ακόμη ο Προγραμματισμός Συνόλου Απαντήσεων έχει εφαρμοστεί στην παραγωγή σχεδίων και προβλημάτων στην Τεχνητή Νοημοσύνη.

3.2 Ομοιότητες και Διαφορές Προγραμματισμού Συνόλου Απαντήσεων με Prolog

Ο Προγραμματισμός Συνόλου Απαντήσεων και η Prolog έχουν αρκετές ομοιότητες αλλά και αρκετές διαφορές, τόσο στον συντακτικό τρόπο γραφής των προγραμμάτων, όσο στην γενική ιδέα σκέψης.

Συντακτικά τα προγράμματα στον Προγραμματισμό Συνόλου Απαντήσεων μοιάζουν σε μεγάλο βαθμό με προβλήματα γραμμένα σε Prolog. Διαφέρουν όμως στους υπολογιστικούς μηχανισμούς και στις ιδέες στους οποίους είναι βασισμένα.

Μια απάντηση από το σύνολο απαντήσεων (answer set) του Προγραμματισμού Συνόλου Απαντήσεων αποτελεί ένα πιθανό ορισμό μιας σωστής απάντησης, για μια ερώτηση στη Prolog.

Ο Προγραμματισμός Συνόλου Απαντήσεων αναπαριστά της λύσεις του σε ένα πρόβλημα μέσω του συνόλου απαντήσεων (answer set) αντί των αντικαταστάσεων απάντησης (answer substitutions). Επίσης φαίνεται η αναζήτηση των απαντήσεων που δίνει.

Η Prolog είναι ευαίσθητη στην σειρά με την οποία γράφονται οι κανόνες και την σειρά των κατηγορημάτων στο σώμα ενός κανόνα. Λανθασμένη σειρά των κατηγορημάτων στο σώμα του κανόνα ή σειρά των κανόνων οδηγεί σε βρόγχους με

άπειρες επαναλήψεις (infinite loops). Δίνει μια απάντηση και επιπλέων “yes” η “no” αν υπάρχει και άλλη πιθανή απάντηση η όχι στο πρόβλημα που δίνεται.

3.3 Πλεονεκτήματα του Προγραμματισμού Συνόλου Απαντήσεων

Το σημαντικότερο πλεονέκτημα του Προγραμματισμού Συνόλου Απαντήσεων είναι η απλότητα στη σύνταξη και η σχέση του με τον προγραμματισμό περιορισμών. Δεν υπάρχει κανένα σύμβολο συνάρτησης, και δεν υπάρχει καμία ανάγκη για ενοποίηση (unification).

Οι προτάσεις και τα προγράμματα αποτελούν τους περιορισμούς στα σύνολα. Το πρόγραμμα αναπτύσσεται με την αντιπροσώπευση των περιορισμών προβλήματος.

3.4 Κανόνες στο Προγραμματισμό Συνόλου Απαντήσεων

Ένα κανονικό λογικό πρόγραμμα *(normal logic program)* είναι ένα σύνολο κανονικών κανόνων (*normal rules*) της πιο κάτω μορφής:

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

Όπου $n \geq m \geq 0$ και $A_0, A_1, \ldots..,A_n$ είναι προτασιακά άτομα. Το άτομο A_0 λέγεται κεφαλή (head) του κανόνα και το σύνολο των ατόμων $A_1, ...,A_m$,not $A_{m+1},\ldots,$not A_n λέγεται *σώμα* (body) του κανόνα.

Αν το σώμα είναι άδειο (n = 0) δηλαδή ο κανόνας είναι της μορφής $A_0 \leftarrow$ τότε ο κανόνας αυτός ονομάζεται *γεγονός* (fact) και μπορούμε να το αναπαραστήσουμε εναλλακτικά ως A_0.

Ένας κανόνας είναι θετικός (positive) αν n = m και έχει την μορφή

$$\mathbf{A_0 \leftarrow A_1, ... ,A_m .}$$

Εκτός από τους κανονικούς κανόνες , υποθέτουμε ότι η γλώσσα μας έχει και περιορισμούς, για παράδειγμα κανόνες με μια κενή κεφαλή, με άλλα λόγια, κανόνες της πιο κάτω μορφής

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

Ένας τέτοιος κανόνας δηλώνει ότι οποιοιδήποτε 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:

$$\boldsymbol{f} \quad \leftarrow \quad \mathbf{not}\ \boldsymbol{f}, \mathbf{A_1, ...,A_m\ ,not\ A_{m+1},\ldots,not\ A_n}$$

όπου το *f* είναι ένα νέο άτομο.

Ένα λογικό πρόγραμμα είναι ένα πεπερασμένο σύνολο από κανόνες.

Για παράδειγμα,

p ← not q
q ← not r
r ← p
r ← q

Ένα πρόγραμμα P είναι θετικό (positive) αν όλοι οι κανόνες του P είναι θετικοί.

Δηλαδή το πρόγραμμα :

r ← p
r ← q
p ← t

είναι θετικό (positive).

Αντίθετα όμως το πρόγραμμα:

p ← not q
q ← not t

δεν είναι θετικό (positive).

Κάθε πρόγραμμα έχει στη πραγματικότητα μοναδικό σύνολο απαντήσεων (answer set).

3.5 Σύνολο Απαντήσεων ενός θετικού ενός προγράμματος

Αρχικά, θα ορίσουμε το σύνολο απαντήσεων (answer set) για θετικά (positive) προγράμματα. Έχουμε ένα σύνολο X ατόμων που ικανοποιούν το θετικό (positive) πρόγραμμα αν για όλους τους κανόνες της μορφής $A_0 \leftarrow A_1, ...,A_m$ το πρόγραμμα, $A_0 \in X$ όταν $A_1, ...,A_m \in X$.

Έχοντας το παρακάτω πρόγραμμα:

p ← q , s
s ←
q ←

Το σύνολο συμπερασμάτων του προγράμματος μας είναι S = { s, q, p }. Το σύνολο αυτό μπορούμε να το δούμε σαν το μοντέλο του, όπου όλα τα άτομα λαμβάνουν την τιμή True (T). Το S ονομάζεται σύνολο απαντήσεων του προγράμματος.

Το Σύνολο Απαντήσεων και τα μοντέλα ενός προγράμματος αναπαρίστανται από το σύνολο των ατόμων που παίρνουν την τιμή True (T).

Ένα σύνολο S είναι το Σύνολο Απαντήσεων ενός προγράμματος μόνο αν όταν το S είναι το ελάχιστο μοντέλο του προγράμματος. Ένα σύνολο S είναι το ελάχιστο μοντέλο ενός προγράμματος Π αν δεν υπάρχει ένα σύνολο S' που να είναι γνήσιο υποσύνολο του S τέτοιο ώστε S' να είναι μοντέλο του προγράμματος. Αυτό σημαίνει ότι Σύνολο Απαντήσεων είναι το μικρότερο σύνολο ατόμων που ικανοποιούν το πρόγραμμα μας.

Παράδειγμα 2.1:

$p \leftarrow q$
$q \leftarrow p$

Το πρόγραμμα έχει δύο διαφορετικές λύσεις

Λύση 1:
S = {p, q},
δηλαδή p = T (true), q = T (true)

Λύση 2:
$S = \{ Ø \}$,
Δηλαδή, p = F (false), q = F (false)

Στο παραπάνω παράδειγμα το S = {p, q} δεν είναι ελάχιστο μοντέλο επειδή το κενό είναι υποσύνολο του S ($Ø \subset S$). Άρα, από αυτά τα δύο μοντέλα μόνο το S στην $2^{η}$ λύση είναι Σύνολο Απαντήσεων.

3.6 Σύνολο Απαντήσεων ενός προγράμματος με χρήση άρνησης

Επεκτείνοντας τα λογικά προγράμματα χρησιμοποιώντας τον τελεστή not, δημιουργείται ένα λογικό πρόγραμμα είναι ένα σύνολο από κανόνες της μορφής:

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

Ο τελεστής not ονομάζεται τελεστής άρνησης και θεωρείτε αποτυχία (Negation as Failure). Διαισθητικά, το **not** A_n είναι αληθές εάν δεν υπάρχει τρόπος να αποδειχθεί το A_n.

Αυτό βασίζεται στην υπόθεση του κλειστού κόσμου. Αυτή η δήλωση υποδηλώνει ότι όλα τα γεγονότα που δεν είναι λογικές συνέπειες του προγράμματος είναι ψευδή.

Παράδειγμα 2.2:

$$p \leftarrow \text{not } p$$

Το πιο πάνω αντιφατικό πρόγραμμα δεν έχει κάποιο ευσταθές μοντέλο γιατί δεν μπορούμε να συμπεράνουμε κατά πόσο το p είναι αληθές ή όχι.

Αν υποθέσουμε ότι δεν είναι αληθές το p τότε συμπεραίνουμε ότι ισχύει το p, δηλαδή φτάνουμε σε αντίφαση με τον κανόνα. Έτσι δεν μπορούμε να συμπεράνουμε ούτε ότι το p είναι αληθές αλλά ούτε ότι το p είναι ψευδές.

3.7 Θέματα στα οποία έχει χρησιμοποιηθεί ο Προγραμματισμός Συνόλου Απαντήσεων

Στο Προγραμματισμό Συνόλου Απαντήσεων δεδομένου κάποιου υπολογιστικού προβλήματος σαν λογικό πρόβλημα, το σύνολο απαντήσεων (answer set) θα ανταποκρίνεται στις λύσεις τις οποίες θα βρει. Ο Προγραμματισμός Συνόλου Απαντήσεων έχει χρησιμοποιηθεί για να υπολογιστούν διαφορετικά είδη υπολογιστικών προβλημάτων, όπως:

3.7.1. Προβλήματα προγραμματισμού (Planning Problems):

Προβλήματα Προγραμματισμού είναι τα προβλήματα στα οποία ψάχνουμε να βρούμε μια ακολουθία από πράξεις οι οποίες αν υλοποιηθούν θα μας οδηγήσουν στο επιθυμητό αποτέλεσμα. Αυτά τα προβλήματα αποτελούν και το αντικείμνο της εργασλίας αυτής.

3.7.2. Προβλήματα Δρομολόγησης (Routing):

Προβλήματα Δρομολόγησης είναι τα προβλήματα στα οποία ψάχνουμε να βρούμε μια λογική και επιπλέον συμφέρουσα σειρά, ανάλογα με τα δεδομένα κάποιου προβλήματος, για δρομολόγηση διάφορων αντικειμένων. Ένα πρόβλημα δρομολόγησης είναι το προβλήματα δρομολόγησης καλωδίων και απαιτείται ο προσδιορισμός των φυσικών θέσεων των καλωδίων που ενώνουν το κύκλωμα.

3.7.3. Προβλήματα Αναδημιουργίας (Phylogeny Reconstruction):

Προβλήματα Αναδημιουργίας είναι τα προβλήματα στα οποία πρέπει να δημιουργηθεί ένα εξελικτικό δέντρο για ένα σύνολο αντικειμένων, το οποίο περιγράφει την εξέλιξη ενός τέτοιου δέντρου αρχίζοντας από τον πιο πρόσφατο πρόγονο του.

Κεφάλαιο 4

Lparse, Smodels και Clasp

4.1 Εισαγωγή στα Lparse, Smodels και Clasp

Τα Smodels και Clasp είναι συστήματα που εφαρμόζονται στο Προγραμματισμό Συνόλου Απαντήσεων (Answer Set Programming). Τα συστήματα του Smodels και Clasp αποτελούντε από δυο προγράμματα, το Smodels ή Clasp αντίστοιχα, και το Lparse. Τα Smodels και Clasp είναι μια αποδοτική υλοποίηση της σημασιολογίας ευσταθούς μοντέλου (stable model) για τα λογικά προγράμματα, δηλαδή είναι τα προγράμματα που θα δώσουν τις λύσεις για ένα πρόγραμμα συνόλου απαντήσεων. Το Lparse είναι ένας αναλυτής ο οποίος μετατρέπει το λογικό πρόγραμμα σε κάποια μορφή την οποία μπορεί να κατανοήσει το Smodels και Clasp έτσι ώστε να μας δώσουν τα σύνολα απαντήσεων (answer sets) που θέλουμε.

Ο Προγραμματισμός Συνόλου Απαντήσεων (Answer Set Programming) είναι ένα είδος προγραμματισμού, διαφορετικό από τον παραδοσιακό προγραμματισμό. Στο Προγραμματισμός Συνόλου Απαντήσεων ο προγραμματιστής δίνει στο σύστημα την περιγραφή του προβλήματος σε μια τυπική γλώσσα και στην συνέχεια η μηχανή λύνει το πρόβλημα βασισμένη στους κανόνες που της έχουμε δώσει, δίνοντας λύσεις στο

πρόβλημα μας, ενώ σε μια γλώσσα παραδοσιακού προγραμματισμού δίνουμε τον αλγόριθμο για την λύση του προβλήματος μας.

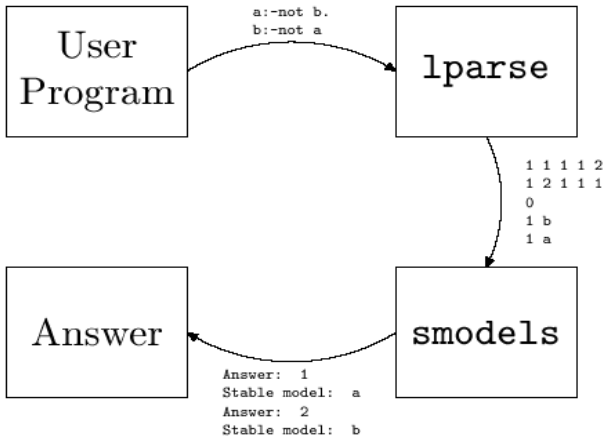
Το Smodels αποτελείται από δύο προγράμματα: Το Lparse και το Smodels. Το πρώτο πρόγραμμα, το Smodels που είναι η μηχανή προγραμματισμού λογικής που κάνει όλη τη σκληρή εργασία για να δώσει λύσεις και το Lparse, απλά προσθέτει ακριβώς ένα πεδίο συντακτικής κορυφής (syntactic sugar) πάνω σ' αυτήν. Το Smodels αναπτύχθηκε στο εργαστήριο θεωρίας της Πληροφορικής του Πανεπιστημίου του Ελσίνκι και το Lparse αναπτύχθηκε από τον Tommy Syrjanen.

Το Clasp αποτελείται επίσης από δύο προγράμματα: Το Lparse και το Clasp. Το Clasp κάνει ακριβώς τη δουλειά που κάνει και το Smodels. Το Clasp αναπτύχθηκε στο εργαστήριο Πληροφορικής του Πανεπιστημίου του Πότσταμ.

Τα προγράμματα στο Smodels και Clasp γράφονται χρησιμοποιώντας τον λογικό προγραμματισμό. Δηλαδή τα προγράμματα αποτελούνται από τα άτομα και τους κανόνες συμπεράσματος τους. Ένα άτομο αντιπροσωπεύει ένα κομμάτι του κόσμου του προβλήματος και μπορεί να είναι αληθινό ή όχι. Οι κανόνες συμπεράσματος χρησιμοποιούνται για να κωδικοποιήσουν τις σχέσεις μεταξύ των ατόμων και να δημιουργήσουν ένα σταθερό πρότυπο (stable model), το οποίο θα λέει ποια άτομα είναι αληθινά.

Ένα πρόγραμμα στο Smodels ή Clasp μπορεί να έχει ένα, κανένα, ή πολλά σύνολα απαντήσεων. Τα σταθερά μοντέλο ενός προγράμματος είναι οι λύσεις που μπορούν να υπάρξουν από το πρόγραμμα που έχουμε δώσει στο Smodels ή Clasp και μπορούν να υπάρξουν ως σύνολο λογικών λύσεων για το πρόγραμμα. Μπορούμε να σκεφτούμε ότι ένα πρόγραμμα είναι μια βάση με γνώσεις που κωδικοποιεί τις σχέσεις μεταξύ των αντικειμένων μας και ένα σταθερό μοντέλο είναι ένα σύνολο εκείνων των πραγμάτων στον κόσμο μας που δημιουργούμε.

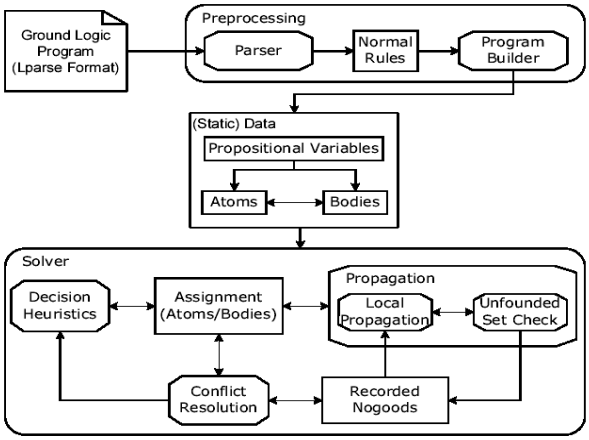
Τα βήματα που απαιτούνται για την εύρεση ενός συνόλου απαντήσεων είναι στο Smodels είναι :



Σχήμα 4.1: Έυρεση συνόλου απαντήσεων με το Smodels

Το Lparse όπως βλέπουμε στο Σχήμα 4.1, ως αναλυτής μετατρέπει το πρόγραμμα του λογικού προγραμματισμού σε κάποια μορφή την οποία κατανοεί το Smodels. Αφού το Smodels πάρει το πρόγραμμα αυτό σε μια συγκεκριμένη μορφή, μας δίνει το αποτέλεσμα που θέλουμε να πάρουμε από το πρόβλημα.

Τα βήματα που απαιτούνται για την εύρεση ενός συνόλου απαντήσεων είναι στο Clasp είναι :



Σχήμα 4.2: Έυρεση συνόλου απαντήσεων με το Clasp

Το Lparse όπως βλέπουμε στο Σχήμα 4.2, και πάλι ως αναλυτής μετατρέπει το πρόγραμμα του λογικού προγραμματισμού σε κάποια μορφή την οποία κατανοεί το Clasp. Αφού το Clasp πάρει το πρόγραμμα αυτό σε μια συγκεκριμένη μορφή, μας δίνει το αποτέλεσμα που θέλουμε να πάρουμε από το πρόβλημα. Τα πλεονεκτήματα του Clasp σε σχέση με το Smodels είναι ότι μπορεί να αντιληφθεί συγκρούσεις και θυμάτε από που ξεκίνησε έτσι δεν χρειάζεται να ξεκινήσει το ψάξιμο των λύσεων από την αρχή. Άρα αναμένεται να είναι πιο γρήγορο σε σχέση με το Clasp.

4.2 Τα τέσσερα βασικά αντικείμενα που συνθέτουν τον λογικό προγραμματισμό

Στην γλώσσα του λογικού προγραμματισμού υπάρχουν τέσσερα είδη αντικειμένων τα οποία χρησιμοποιούνται για την γραφή προγραμμάτων. Αυτά είναι τα άτομα, οι σταθερές, οι μεταβλητές και οι κανόνες.

Σταθερές:
Οι σταθερές είναι οι ανεξάρτητες τιμές που υπάρχουν στον κόσμο του προβλήματος μας: Σταθερές σε ένα πρόβλημα μπορεί να είναι είτε αριθμοί είτε κάποιοι συμβολισμοί λέξεων. Όπως και στις πιο πολλές γλώσσες προγραμματισμού έτσι και στη περίπτωσή μας, το πρώτο γράμμα των σταθερών γράφεται με μικρό χαρακτήρα του αγγλικού αλφαβήτου.
Παραδείγματα σταθερών είναι: var, p, q, 10.

Μεταβλητές:
Οι μεταβλητές στον λογικό προγραμματισμό υπάρχουν για να γενικεύουν το πρόβλημα (δηλαδή μπορούν να πάρουν οποιαδήποτε τιμή τους δοθεί). Η μηχανή που λύνει το πρόβλημα ψάχνει και βρίσκει τη σωστή τιμή που μπορεί να τους ανατεθεί, ενώ στο παραδοσιακό προγραμματισμό αναθέτεις μια τιμή απευθείας. Ακόμη οι μεταβλητές αντίθετα πάλι από τον παραδοσιακό προγραμματισμό ξεκινούν πάντα με κεφαλαίο χαρακτήρα του αγγλικού αλφαβήτου
Παραδείγματα μεταβλητών είναι: X, Y, Z, Var,

Άτομα:
Ένα άτομο συνθέτετε από ένα κατηγόρημα που ακολουθείτε από μια λίστα μεταβλητές ή σταθερές μέσα στην παρένθεση. Τα άτομα χρησιμοποιούνται για να δημιουργούμε σχέσεις μεταξύ των σταθερών του προγράμματος. Τα άτομα μπορούν να πάρουν τιμές ορθό(true) και λάθος(false).
Παράδειγμα ατόμου είναι: on(block1,location1,1).
Όπου σημαίνει ότι αντικείμενο block1 βρίσκεται πάνω στη τοποθεσία location1 τη χρονική στιγμή 1.

Κανόνες:
Οι κανόνες σε ένα λογικό πρόγραμμα μας επιτρέπουν να αναφερόμαστε, δηλαδή να κάνουμε αλλαγές σε κατηγορήματα μέσω άλλων κατηγορημάτων. Ένας κανόνας αποτελείτε από δύο μέρη την κεφαλή και το σώμα του κανόνα. Η κεφαλή είναι το κομμάτι πριν από το :- και το σώμα το μέρος μετά από αυτό. Κάθε άτομο στο σώμα του κανόνα πρέπει να ισχύει για να ισχύσει η κεφάλι του κανόνα.

Παράδειγμα κανόνα είναι:
on(X,Z,T+1) :- on(X,Y,T), move(X,Z,T), block(X), block(Y), block(Z).
Το πιο πάνω παράδειγμα θα μας έλεγε πως το X είναι τοποθετημένο στο Y τη χρονική στιγμή T και το X επίσης μετακινείται στο Z τη χρονική στιγμή T, όπου X, Y και Z είναι αντικείμενα (blocks), τότε σαν αποτέλεσμα είναι ότι το X, βρίσκεται στο Z την επόμενη χρονική στιγμή.

4.3 Τρόπος Λειτουργία του Smodels και Clasp

Το Smodels και Clasp εκτελεί τις εργασίες με έναν διαφορετικό τρόπο. Στην πρώτη φάση όλες οι μεταβλητές αντικαθιστούνται από το πρόγραμμα με όλες τις πιθανές τιμές για όλους τους κανόνες. Αυτή η φάση ονομάζεται grounding και είναι η δουλεία του Lparse. Στην επόμενη φάση το Smodels και Clasp υπολογίζει το σταθερό σύνολο απαντήσεων (stable model) του προγράμματος.

Το σύνολο απαντήσεων αποτελείται από μοντέλο. Ένα μοντέλο είναι ένα σύνολο ατόμων που τηρεί κάθε κανόνα στο λογικό πρόγραμμα. Ένας κανόνας ισχύει όταν η κεφάλι(head) του είναι αληθινή στο μοντέλο. Επίσης μπορούμε να πούμε ότι ένας κανόνας ισχύει, όταν τα αληθή κατηγορήματα του σώματος ισχύουν ενώ τα μη αληθή κατηγορήματα δεν ισχύουν.

Μπορούμε να πούμε ότι ένα σύνολο ατόμων είναι ένα πρότυπο στο σύνολο απαντήσεων, εάν για κάθε άτομο στο πρότυπο υπάρχει κάποιος κανόνας που έχει το άτομο ως κεφάλι έτσι ώστε το σώμα του κανόνα είναι αληθινό στο μοντέλο.

Δηλαδή εάν το σώμα είναι αληθινό, τότε και το κεφάλι πρέπει επίσης να είναι αληθινό. Το ακόλουθο τμήμα προγράμματος επεξηγεί περαιτέρω αυτήν την πρακτική:

```
p :- not q.
q :- not p.
```

Αν το q δεν είναι αληθινό τότε το p πρέπει να είναι αληθινό και αντίστροφα. Εντούτοις, αυτή η κατασκευή δεν περιλαμβάνει τα άτομα, δεδομένου ότι είναι δυνατό κάποιο άλλο μέρος του προγράμματος να αναγκάζει το p και το q να είναι αληθινά. Για να ισχύει η πράξη του XOR κάποιο άτομο πρέπει να προστεθεί στο πρόγραμμα έναν κανόνα της μορφής:

```
:- p, q.
```

Ο πιο πάνω κανόνας ερμηνεύεται ως εξής: δεν μπορεί να ισχύει ποτέ και το p και το q. Κανόνες χωρίς κεφαλή συμπεριφέροντε ως πράξη ως περιορισμού ακεραιότητας, δηλαδή εάν το σώμα ενός τέτοιου κανόνα ικανοποιείται, τότε ο κανόνας αποτυγχάνει.

4.4 Χρησιμοποίηση του Lparse

Έχουμε πει ότι συντακτικά ο προγραμματισμός συνόλου απαντήσεων και Prolog είναι πανομοιότητες. Όπως και στην Prolog, το ← στην lparse συμβολίζεται με :- . Επιπλέον βάζουμε ένα κανόνα σε κάθε γραμμή και κάθε κανόνας τελειώνει με τελεία (.).

Αν θέλουμε να βρούμε το answer set του πιο κάτω προγράμματος:

s(2).
x(1).
x :- y.
y :- not s(x).
z :- m,y.
m :- not x.

τότε δημιουργούμε ένα αρχείο και το ονομάζουμε program.lp και στην συνέχεια τρέχουμε το πρόγραμμα στο Smodels ως εξής:

```
% lparse program.lp | smodels 0
```

Το 0 στο τέλος της εντολής δείχνει ότι θέλουμε να υπολογίσουμε όλες τις απαντήσεις (answer sets). Αν βάλουμε κάποιο θετικό αριθμό n στην θέση του 0 τότε το Smodels θα τερματίσει όταν υπολογίσει τις πρώτες n απαντήσεις (answer sets). Η προκαθορισμένη τιμή για το n είναι 1. Δημάδη αν δεν βάλουμε τιμή στο τέλος θα δώσει μια μόνο λύση. Η έξοδος του προγράμματος είναι μια λίστα με τα answer sets του προγράμματος όπως φαίνεται πιο κάτω:

smodels version 2.32. Reading...done
Answer: 1
Stable Model: x y s(2) x(1)
False
Duration: 0.000

Number of choice points: 0
Number of wrong choices: 0
Number of atoms: 5
Number of rules: 4
Number of picked atoms: 0
Number of forced atoms: 0
Number of truth assignments: 4
Size of searchspace (removed): 0 (0)

Με το stable model εννοούμε απάντηση (answer set). Στην πρώτη γραμμή τυπώνει πληροφορίες για την έκδοση του Smodels. Οι επόμενες γραμμές μας δίνουν το ευσταθές μοντέλο του πιο πάνω προγράμματος που είναι: x y s(2) x(1). Η λέξη False κάτω από το ευσταθή μοντέλο δηλώνει ότι δεν υπάρχουν άλλα ευσταθή μοντέλα που να ικανοποιούν το πιο πάνω πρόγραμμα. Η λέξη False εμφανίζεται και όταν το πρόγραμμα δεν έχει καθόλου λύσεις. Στην περίπτωση που 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 κάποιο 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 δηλώνει πόσες φορές το Smodels κατάφερε να πάρει μια αληθής τιμή για ένα άτομο. Ο αριθμός των forced atoms μας λέει πόσα άτομα προστέθηκαν στο μοντέλο λόγω των κανόνων διάχυσης περιορισμών (propagation rules) που θα προκαλούσαν αντίφαση. Ο αριθμός των truth assignments 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 μοντέλο υπάρχει ή όχι.
Στο Lparse μπορούμε να χρησιμοποιήσουμε μεταβλητές. Όπως και στην Prolog, οι μεταβλητές αναπαριστούνται με κεφαλαία γράμματα.

Το ίδιο πρόγραμμα το τρέχουμε και στο Clasp ως εξής:

```
% lparse program.lp | clasp 0
```

Το 0 στο τέλος της εντολής, και πάλι δείχνει ότι θέλουμε να υπολογίσουμε όλες τις απαντήσεις (answer sets). Αν βάλουμε κάποιο θετικό αριθμό n στην θέση του 0 τότε το Clasp θα τερματίσει όταν υπολογίσει τις πρώτες n απαντήσεις (answer sets). Η προκαθορισμένη τιμή για το n είναι 1. Δημάδη αν δεν βάλουμε τιμή στο τέλος θα δώσει μια μόνο λύση. Η έξοδος του προγράμματος είναι μια λίστα με τα answer sets του προγράμματος όπως φαίνεται πιο κάτω:

clasp version 1.0.4. Reading...done
Answer: 1
x y s(2) x(1)

Models : 1
Time : 0.000 (Parsing: 0.000)

Στην πρώτη γραμμή τυπώνει πληροφορίες για την έκδοση του Clasp. Οι επόμενες γραμμές μας δίνουν το ευσταθές μοντέλο του πιο πάνω προγράμματος που είναι: x y s(2) x(1). Με το Answer εννοούμε απάντηση (answer set). Στην ένδυξη models φαίνεται ο αριθμός απαντήσεων που το Clasp μας έχει παρουσιάσει, ενώ αν υπήρχαν πιο πολλές απαντήσεις από τον αριθμό απαντήσεων που ζητήσαμε να μας παρουσιάσε το Clasp θα υπήρχε το σύμβολο ‘+’ δίπλα από τον αριθμό. Επίσης στη τελευταία γραμμή φαίνεται ο χρόνος αναζήτησης απαντήσεων του Clasp σε δευτερόλεπτα.

4.5 Πρακτικό Παράδειγμα

Οι βασικές έννοιες εισάγονται εδώ με τη χρησιμοποίηση του προβλήματος χρωματισμού κόμβων (node coloring) ως παράδειγμα. Σε μια περίπτωση ο χρωματισμός

κόμβων μας δίνεται ως ένα σύνολο κόμβων και ένα σύνολο ακρών που συνδέουν τους κόμβους. Το πρόβλημα είναι να χρησιμοποιηθεί κάποιος σταθερός αριθμός χρωμάτων για να χρωματίσει κάθε κόμβο έτσι ώστε δύο γειτονικοί κόμβοι να μην έχουν το ίδιο χρώμα. Σε αυτό το παράδειγμα χρησιμοποιούμε τρία χρώματα: κόκκινο, μπλε και κίτρινο.
Ο χρωματισμός κόμβων μπορεί να υλοποιηθεί με το ακόλουθο πρόγραμμα:

```
color(red). color(blue). color(yellow).
col(X, red) :- node(X), not col(X,blue), not col(X,yellow).
col(X, blue) :- node(X), not col(X,red), not col(X,yellow).
col(X, yellow) :- node(X), not col(X,blue), not col(X,red).
fail :- edge(X,Y), color(C), col(X,C), col(Y,C).
node(a). node(b).
node(c). node(d).
edge(a,b). edge(b,c).
edge(c,d). edge(d,a).
compute 1 { not fail }.
```

Η πρώτη γραμμή:

color(red). color(blue). color(yellow).

καθορίζει τα χρώματα που έχουμε την ελευθερία να χρησιμοποιήσουμε. Το κατηγόρημα " color " καθορίζεται με τέτοιο τρόπο έτσι ώστε να μπορεί να χρησιμοποιηθεί ως κατηγόρημα περιοχών που θα υπάρξουν αργότερα στο πρόγραμμα.

Ο κανόνας,

col(X, red) :- node(X), not col(X,blue), not col(X,yellow).

δηλώνει ότι εάν ένας κόμβος δεν είναι μπλε και ούτε κίτρινο τότε ο κόμβος πρέπει να είναι κόκκινο. Οι επόμενοι δύο κανόνες είναι ίδιοι αλλά είναι για τα χρώματα μπλε και κίτρινο. Εδώ το κατηγόρημα node(X) ενεργεί ως κατηγόρημα περιοχών που απαριθμεί τις πιθανές τιμές της μεταβλητής X.

Ο κανόνας,

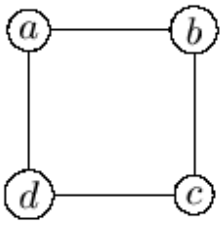
fail :- edge(X,Y), color(C), col(X,C), col(Y,C).

εξασφαλίζει ότι οι γειτονικοί κόμβοι έχουν διαφορετικά χρώματα. Το άτομο "fail" είναι αληθινό ακριβώς όταν έχουν δύο συνδεδεμένοι κόμβοι το ίδιο χρώμα. Αργότερα αναγκάζουμε Smodels να ψάξει για τα πρότυπα μόνο όπου "fail" δεν είναι αληθινό. Εδώ ο κόμβος κατηγορήματος δεν απαιτείται για να δώσει την περιοχή για τον κόμβο μεταβλητών X και Y επειδή η άκρη είναι επίσης ένα κατηγόρημα περιοχών.

Το επόμενο κομμάτι του προγράμματος:

node(a). node(b).
node(c). node(d).
edge(a,b). edge(b,c).
edge(c,d). edge(d,a).

καθορίζει ένα απλό γράφο με τέσσερις κόμβους και τέσσερις άκρες. Αυτός ο γράφος είναι αυτός που βλέπουμε στο Σχήμα 2.2. Συνήθως, ο γράφος αποθηκεύεται σε ένα άλλο αρχείο έτσι ώστε να μην είναι απαραίτητο να αναπαραγάγουμε τους κανόνες συμπεράσματος για κάθε γραφική παράσταση.



Σχήμα 4.3: Ο γράφος που δημιουργείται από το πρόγραμμα.
Η τελευταία γραμμή του προγράμματος:

compute 1 { not fail }.

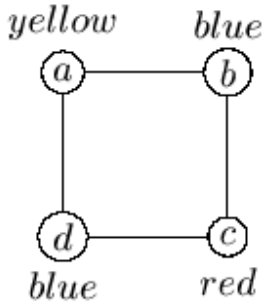
λέει στο Smodels ότι θέλουμε μόνο ένα πρότυπο και το άτομο "fail" μπορεί να μην είναι στο μοντέλο. Αυτό αποκλείει όλα τα μοντέλα όπου δύο γειτονικοί κόμβοι έχουν το ίδιο χρώμα.

Το άτομο "fail" χρησιμοποιείται για να επισημάνει ότι υπάρχει κάποιο λάθος με το μοντέλο, δηλαδή ότι δύο γειτονικοί κόμβοι έχουν το ίδιο χρώμα. Αργότερα στο πρόγραμμα απαιτούμε ότι το"fail μπορεί να μην είναι αληθινό σε οποιοδήποτε σύνολο απαντήσεων του προγράμματος.

Το ίδιο αποτέλεσμα θα μπορούσε να επιτευχθεί με τη χρησιμοποίηση της κατασκευής:

:- edge(X,Y), color(C), col(X,C), col(Y,C).

Κανόνες με κενό στην κεφάλι, δουλεύουν ως περιορισμός στο πρότυπο του προγράμματος. Εάν μια μεταβλητή καθιστά το σώμα του κανόνα αληθινό, η σύνδεση απορρίπτεται δεδομένου ότι δεν υπάρχει εκεί κανένας τρόπος να ικανοποιήσει το κεφάλι του κανόνα.



Σχήμα 3.4: Μια πιθανή απάντηση στο πρόγραμμα.

Όταν τρέξουμε το συγκεκριμένο πρόγραμμα με το Smodels, η απάντηση που θα πάρουμε θα είναι η παρακάτω:

% lparse color.lp | smodels
Smodels version 2.10. Reading... done
Answer: 1
Stable Model:edge(d,a) edge(c,d) edge(a,b) edge(b,c)
node(a) node(d) node(c) node(c) node(b) col(a,yellow) col(c,red)
col(d,blue) col(b,blue) color(yellow) color(red) color(blue)
True
Duration: 0.030
Number of choice points: 3
Number of wrong choices: 0
Number of atoms: 24
Number of rules: 35
Number of picked atoms: 45
Number of forced atoms: 0
Number of truth assignments: 152
Size of searchspace (removed): 12 (0)

Όταν τρέξουμε το συγκεκριμένο πρόγραμμα με το Clasp, η απάντηση που θα πάρουμε θα είναι η παρακάτω:
%lparse color.lp | clasp
clasp version 1.0.4. Reading...done
Answer: 1
col(a,yellow) col(c,yellow) col(b,blue) col(d,red) edge(d,a) edge(c,d) edge(b,c)
edge(a,b) node(d) node(c) node(b) node(a) color(yellow) color(blue) color(red)

Models : 1+
Time : 0.000 (Parsing: 0.000)

Κεφάλαιο 5

Μοντελοποίηση Προβλημάτων Προγραμματισμού Δράσης Σε Λογικό Προγραμματισμό

5.1 Εισαγωγή στη μοντελοποίηση προβλημάτων προγραμματισμού δράσης σε λογικό προγραμματισμό

Ο προγραμματισμός δράσης ασχολείται με τον ορισμό δράσεων για κάποιο κόσμο, οπού δεδομένης μιας αρχικής κατάστασης και ενός τελικού σκοπού, δίνει ένα πλάνο. Μια ακολουθία από δράσεις δηλαδή, όπου οδηγεί την αρχική κατάσταση στη τελική.

Ο λογικός προγραμματισμός ασχολείται με την δημιουργία κανόνων για κάποιο κόσμο. Στο λογικό προγραμματισμό υπάρχουν κανόνες όπου καθορίζουν ποτέ ισχύει ένα γεγονός και πότε όχι.

Στόχος μας είναι να μετατρέψουμε προβλήματα προγραμματισμού δράσης σε λογικό προγραμματισμό συνόλου απαντήσεων. Θα ασχοληθώ με τρία συγκεκριμένα προβλήματα προγραμματισμού όπου και θα αναλυθούν πιο κάτω. Έχω ως δεδομένο τα προγράμματα σε προγραμματισμό δράσεις και πρέπει να τα μετατρέψω σε λογικό πρόγραμμα.

Στο προγραμματισμό δράσης όπως έχουμε ξαναπεί ορίζονται οι δράσεις, δηλαδή ορίζονται οι παραμέτροι της, οι συνθήκες που πρέπει να ισχύουν για να γίνει γεγονός μια δράση καθώς επίσης και τα αποτελέσματα της. Πολλές φορές συμβαίνει ένα αποτέλεσμα μιας δράσης να είναι η άρνηση ενός ήδη υπάρχοντος κατηγορήματος. Αυτό δεν μπορεί να συμβεί και στο προγραμματισμό συνόλου απαντήσεων. Από τη στιγμή που ισχύει κάτι στη συνεχεία δεν μπορούμε να το αναιρέσουμε. Άρα κατά τη μετατροπή σε λογικό προγραμματισμό συνόλου απαντήσεων, σε κάθε δράση θα πρέπει να ληφθεί υπόψη και ο παράγοντας χρόνος.
Πιο κάτω θα αναλυθεί και θα επεξηγηθεί λεπτομερώς η μετατροπή του προγραμματισμού δράσεων σε λογικό προγραμματισμό συνόλου απαντήσεων.

5.2 Γενική περιγραφή προβλημάτων που θα μοντελοποιήσω

Τα δυο προβλήματα που θα αναλύσω στη συνέχεια είναι το πρόβλημα δρομολόγησης φορτηγών (trucks), το πρόβλημα ανοικτής στοίβας εργασιών (openstacks) και το πρόβλημα βιολογικών μονοπατιών (pathways).

Περιγραφή προβλήματος δρομολόγησης φορτηγών:

Υπάρχουν κάποια φορτηγά τα οποία έχουν σκοπό να μεταφέρουν πακέτα από μια περιοχή στην άλλη. Τα φορτηγά κινούνται από μια περιοχή σε άλλη φτάνει να οι δυο αυτές περιοχές να είναι ενωμένες. Επίσης τα φορτηγά έχουν συγκεκριμένη χωρητικότητα. Οι χώροι φορτίων στο φορτηγό έχουν σειρά προτεραιότητας. Κατά το φόρτωμα των πακέτων, προτεραιότητα έχουν οι χώροι που είναι πιο μέσα στο φορτηγό δεδομένου ότι είναι άδειοι. Κατά το ξεφόρτωμα ενός φορτίου από το φορτηγό, προτεραιότητα έχουν τα φορτία που είναι φορτωμένα στους πιο έξω χώρους φορτίων, αφού για να ξεφορτωθούν τα μέσα πακέτα θα πρέπει πρώτα να ξεφορτωθούν τα έξω. Τα πακέτα και τα φορτηγά βρίσκονται αρχικά σε μια περιοχή και έχουν ένα και μόνο προορισμό.

Περιγραφή προβλήματος ανοικτής στοίβας:

Υπάρχουν κάποιες παραγγελίες που συμπεριλαμβάνουν κάποια προϊόντα, μια μηχανή που μπορεί να παράγει τα προϊόντα που χρειάζονται οι παραγγελίες καθώς επίσης και μια στοίβα όπου φυλάει τα προϊόντα που παράγονται. Στόχος είναι να παραχθούν τα προϊόντα από τη μηχανή και οι παραγγελίες να παραδοθούν. Αρχικά γνωρίζουμε ότι οι παραγγελίες δεν έχουν ξεκινήσει ακόμη και ότι η μηχανή είναι διαθέσιμη για να παράγει οποιοδήποτε προϊόν φτάνει να γίνει η κατάλληλη ρύθμιση. Επίσης γνωρίζουμε ότι η στοίβα είναι άδεια.

Περιγραφή προβλήματος μονοπατιών:

Το πρόβλημα αυτό έχει σχέση με χημικές αντιδράσεις. Στόχος έχει να παράξει κάποιες χημικές ενώσεις. Υπάρχουν δεδομένα μόρια δυο τύπων, απλά και σύνθετα. Κάποια από αυτά είναι διαθέσιμα. Επίσης είναι γνωστές οι διάφορες χημικές αντιδράσεις μεταξύ μορίων. Γνωρίζουμε δηλαδή το αποτέλεσμα χημικής αντίδρασης δυο μορίων, το αποτέλεσμα χημικής αντίδρασης με καταλύτη και το αποτέλεσμα της σύνθεσης δυο μορίων. Στόχος είναι να παραχθούν συγκεκριμένα μόρια ακολουθώντας μια λογική σειρά οι χημικές αντιδράσεις.

5.3 Περιγραφή προβλημάτων σε προγραμματισμό δράσης

5.3.1. Πρόβλημα δρομολόγησης φορτηγών:

Πιο κάτω φαίνεται η περιγραφή του προβλήματος σε προγραμματισμό δράσης.

```
; IPC5 Domain: Trucks Propositional
; Authors: Yannis Dimopoulos, Alfonso Gerevini and Alessandro Saetti

(define (domain Trucks)
(:requirements :typing :adl)
```

```
(:types truckarea time location locatable - object
        truck package - locatable)
(:predicates (at ?x - locatable ?l - location)
             (in ?p - package ?t - truck ?a - truckarea)
             (connected ?x ?y - location)
             (free ?a - truckarea ?t - truck)
            (time-now ?t - time)
            (next ?t1 - time ?t2 - time)
            (le ?t1 - time ?t2 - time)
            (delivered ?p - package ?l - location ?t - time)
            (at-destination ?p - package ?l - location)
            (closer ?a1 - truckarea ?a2 - truckarea))

(:action load
  :parameters (?p - package ?t - truck ?a1 - truckarea ?l - location)
  :precondition (and (at ?t ?l) (at ?p ?l) (free ?a1 ?t)
                     (forall (?a2 - truckarea)
                             (imply (closer ?a2 ?a1) (free ?a2 ?t))))
  :effect (and (not (at ?p ?l)) (not (free ?a1 ?t)) (in ?p ?t ?a1)))

(:action unload
  :parameters (?p - package ?t - truck ?a1 - truckarea ?l - location)
  :precondition (and (at ?t ?l) (in ?p ?t ?a1)
                     (forall (?a2 - truckarea)
                             (imply (closer ?a2 ?a1) (free ?a2 ?t))))
  :effect (and (not (in ?p ?t ?a1)) (free ?a1 ?t) (at ?p ?l)))

(:action drive
  :parameters (?t - truck ?from ?to - location ?t1 ?t2 - time)
  :precondition (and (at ?t ?from) (connected ?from ?to)
                     (time-now ?t1) (next ?t1 ?t2))
```

```
  :effect (and (not (at ?t ?from)) (not (time-now ?t1))
                    (time-now ?t2) (at ?t ?to)))

(:action deliver
  :parameters (?p - package ?l - location ?t1 ?t2 - time)
  :precondition (and (at ?p ?l) (time-now ?t1) (le ?t1 ?t2))
  :effect (and (not (at ?p ?l)) (delivered ?p ?l ?t2) (at-destination ?p ?l)))
)
```

Τα κατηγορήματα του προγράμματος είναι τα εξής:

(at ?x - locatable ?l - location)

Δηλώνει ότι ένα πακέτο (x), ή ένα φορτηγό (x) βρίσκεται σε συγκεκριμένη τοποθεσία (l).

(in ?p - package ?t - truck ?a - truckarea)

Ένα πακέτο (p) βρίσκεται μέσα στο φορτηγό (t) σε συγκεκριμένο χώρο του φορτηγού(a).

(connected ?x ?y - location)

Μια τοποθεσία (x) είναι ενωμένη με μια άλλη τοποθεσία (y).

(free ?a - truckarea ?t - truck)

Ένας χώρος (a) σε συγκεκριμένο φορτηγό (t) είναι διαθέσιμος.

(time-now ?t - time)

Η παρούσα χρονική στιγμή (t).

(next ?t1 - time ?t2 - time)

Δηλώνει ότι τη χρονική στιγμή (t1) ακολουθεί η χρονική στιγμή (t2).

(le ?t1 - time ?t2 - time)

Δηλώνει ότι η χρονική στιγμή (t1) είναι μικρότερη της χρονικής στιγμής (t2).

(delivered ?p - package ?l - location ?t - time)
Το πακέτο (p) παραδίνεται στη συγκεκριμένο τοποθεσία (l) τη συγκεκριμένη χρονική στιγμή (t).

(at-destination ?p - package ?l - location)
Το πακέτο (p) βρίσκεται στο προορισμό του, δηλαδή σε μια συγκεκριμένη τοποθεσία (l).

(closer ?a1 - truckarea ?a2 - truckarea)
Ο χώρος φορτηγού (a1) βρίσκεται πιο έξω στο φορτηγό από το χώρο (a2).

Στο πρόβλημα αυτό υπάρχουν 4 δράσεις:

Δράση 1: Φόρτωμα

Έχει ως παραμέτρους το πακέτο το οποίο φορτώνεται, το φορτηγό, ο χώρος του φορτηγού και η τοποθεσία που βρίσκονται. Για να μπορεί να γίνει γεγονός η δράση αυτή πρέπει να πακέτο και το φορτηγό να βρίσκονται στην ίδια τοποθεσία καθώς επίσης και ο χώρος του φορτηγού που θα φορτωθεί να είναι άδειος. Επίσης για κάθε χώρο του φορτηγού που βρίσκεται πιο έξω στο φορτηγό από το χώρο που θα φορτωθεί το πακέτο, δηλαδή είναι πιο κοντά στη πόρτα εξόδου, πρέπει να ισχύει ότι είναι άδειο. Σαν αποτέλεσμα της δράσης αυτής είναι το πακέτο να μην βρίσκεται στη τοποθεσία, να βρίσκεται μέσα στο φορτηγό στο συγκεκριμένο χώρο και επιπλέον ο χώρος να μην είναι άδειος.

```
(:action load
  :parameters (?p - package ?t - truck ?a1 - truckarea ?l - location)
  :precondition (and (at ?t ?l) (at ?p ?l) (free ?a1 ?t)
                     (forall (?a2 - truckarea)
                             (imply (closer ?a2 ?a1) (free ?a2 ?t))))
  :effect (and (not (at ?p ?l)) (not (free ?a1 ?t)) (in ?p ?t ?a1)))
```

Δράση 2: Ξεφόρτωμα

Έχει ως παραμέτρους το πακέτο το οποίο φορτώνεται, το φορτηγό, ο χώρος του φορτηγού που βρίσκεται πακέτο και η τοποθεσία που βρίσκονται. Για να μπορεί να γίνει γεγονός η δράση αυτή πρέπει να πακέτο να βρίσκεται μέσα στο φορτηγό και το φορτηγό να βρίσκονται στη τοποθεσία. Επίσης για κάθε χώρο του φορτηγού που βρίσκεται πιο έξω στο φορτηγό από το χώρο που βρίσκεται το πακέτο, δηλαδή είναι πιο κοντά στη πόρτα εξόδου, πρέπει να ισχύει ότι είναι άδειο. Σαν αποτέλεσμα της δράσης αυτής είναι το πακέτο να βρίσκεται στη τοποθεσία, να μην βρίσκεται μέσα στο φορτηγό στο συγκεκριμένο χώρο και επιπλέον ο χώρος να είναι άδειος.

```
(:action unload
  :parameters (?p - package ?t - truck ?a1 - truckarea ?l - location)
  :precondition (and (at ?t ?l) (in ?p ?t ?a1)
                     (forall (?a2 - truckarea)
                             (imply (closer ?a2 ?a1) (free ?a2 ?t))))
  :effect (and (not (in ?p ?t ?a1)) (free ?a1 ?t) (at ?p ?l)))
```

Δράση 3: Οδήγημα

Έχει ως παραμέτρους το φορτηγό, η τοποθεσία που βρίσκεται και η τοποθεσία η οποία θα μεταφερθεί καθώς επίσης και ο χρόνος που κινείται. Για να μπορεί να γίνει γεγονός η δράση αυτή πρέπει το φορτηγό να βρίσκεται στη τοποθεσία. Επίσης η τοποθεσία που βρίσκεται το φορτηγό να είναι ενωμένο με τη τοποθεσία που θα παει. Σαν αποτέλεσμα θα ισχύει ότι το φορτηγό την επόμενη χρονική στιγμή δεν βρίσκεται στο πρώτη τοποθεσία αλλά στη δεύτερη.

```
(:action drive
  :parameters (?t - truck ?from ?to - location ?t1 ?t2 - time)
  :precondition (and (at ?t ?from) (connected ?from ?to)
```

(time-now ?t1) (next ?t1 ?t2))
:effect (and (not (at ?t ?from)) (not (time-now ?t1))
(time-now ?t2) (at ?t ?to)))

Δράση 4: Παράδοση

Έχει ως παραμέτρους το πακέτο, τη τοποθεσία που θα παραδοθεί και το χρόνος που θα παραδοθεί. Για να μπορεί να γίνει γεγονός η δράση αυτή πρέπει το πακέτο να βρίσκεται στη τοποθεσία παράδοσης τη συγκεκριμένη χρονική στιγμή. Σαν αποτέλεσμα θα ισχύει ότι το πακέτο δεν είναι απλά στη τοποθεσία, αλλά το πακέτο παραδόθηκε και επίσης είναι στο προορισμό του.

```
(:action deliver
  :parameters (?p - package ?l - location ?t1 ?t2 - time)
  :precondition (and (at ?p ?l) (time-now ?t1) (le ?t1 ?t2))
  :effect (and (not (at ?p ?l)) (delivered ?p ?l ?t2) (at-destination ?p ?l)))
```

5.3.2 Πρόβλημα ανοικτής στοίβας εργασιών:

Πιο κάτω φαίνεται η περιγραφή του προβλήματος σε προγραμματισμό δράσης.

```
; IPC5 Domain: Openstacks Propositional -- strips forced sequential version
; Author: Patrik Haslum
; The problem instances for this domain were taken from the problem
; collection of the 2005 Constraint Modelling Challenge, organized by
; Barbara Smith & Ian Gent (see http://www.dcs.st-and.ac.uk/~ipg/challenge/).

(define (domain openstacks-sequencedstrips)
   (:requirements :typing :adl)
   (:types order product count)
   (:predicates (includes ?o - order ?p - product)
```

```
            (waiting ?o - order)
            (started ?o - order)
            (shipped ?o - order)
            (made ?p - product)
            (machine-available)
            (machine-configured ?p - product)
            (stacks-avail ?s - count)
            (next-count ?s ?ns - count))

(:action setup-machine
   :parameters (?p - product ?avail - count)
   :precondition (and (machine-available)
                      (not (made ?p))
                      (stacks-avail ?avail))
   :effect (and (not (machine-available)) (machine-configured ?p)))

(:action make-product
   :parameters (?p - product ?avail - count)
   :precondition (and (machine-configured ?p)
                      (forall (?o - order)
                              (imply (includes ?o ?p) (started ?o)))
                      (stacks-avail ?avail))
   :effect (and (not (machine-configured ?p))
            (machine-available)
            (made ?p)))

(:action start-order
   :parameters (?o - order ?avail ?new-avail - count)
   :precondition (and (waiting ?o)
                      (stacks-avail ?avail)
                      (next-count ?new-avail ?avail))
```

```
    :effect (and (not (waiting ?o))
                 (started ?o)
                 (not (stacks-avail ?avail))
                 (stacks-avail ?new-avail))
    )

  (:action ship-order
    :parameters (?o - order ?avail ?new-avail - count)
    :precondition (and (started ?o)
                       (forall (?p - product)
                               (imply (includes ?o ?p) (made ?p)))
                       (stacks-avail ?avail)
                       (next-count ?avail ?new-avail))
    :effect (and (not (started ?o))
                 (shipped ?o)
                 (not (stacks-avail ?avail))
                 (stacks-avail ?new-avail))
    )

  (:action open-new-stack
    :parameters (?open ?new-open - count)
    :precondition (and (stacks-avail ?open)
                       (next-count ?open ?new-open))
    :effect (and (not (stacks-avail ?open))
                 (stacks-avail ?new-open))
    )

  )
```

Τα κατηγορήματα του προγράμματος είναι τα εξής:

```
(includes ?o - order ?p - product)
```

Το προϊόν (p) περιέχεται στη παραγγελία (o).

(waiting ?o - order)
Η παραγγελία (o) βρίσκεται σε κατάσταση αναμονής, δηλαδή δεν έχει ξεκινήσει ακόμα.

(started ?o - order)
Η παραγγελία (o) έχει ξεκινήσει.

(shipped ?o - order)
Η παραγγελία (o) έχει παραδοθεί.

(made ?p - product)
Το προϊόν (p) έχει κατασκευαστεί.

(machine-available)
Η μηχανή είναι διαθέσιμη για να ξεκινήσει την κατασκευή οποιουδήποτε προϊόντος.

(machine-configured ?p - product)
Η μηχανή είναι έτοιμη να παράξει το προϊόν (p).

(stacks-avail ?s - count)
Δηλώνει τη διαθέσιμη θέση (s) στη στοίβα.

Στο πρόβλημα αυτό υπάρχουν 5 δράσεις:

Δράση 1: Ρύθμιση μηχανής

Η συγκεκριμένη δράση έχει τους παραμέτρους προϊόν που θα παρασκευαστεί και θέση στη στοίβα. Για να γίνει γεγονός αυτή η δράση πρέπει να ισχύει ότι η μηχανή είναι διαθέσιμη, η θέση της στοίβας να είναι διαθέσιμη καθώς επίσης πρέπει να ισχύει ότι το προϊόν το οποίο θα ρυθμιστεί να κατασκευάσει η μηχανή να μην έχει κατασκευαστεί.

Σαν αποτέλεσμα της δράσης θα ισχύει ότι η μηχανή δεν θα είναι διαθέσιμη, και η μηχανή θα είναι έτοιμη να κατασκευάσει το προϊόν.

```
(:action setup-machine
      :parameters (?p - product ?avail - count)
      :precondition (and (machine-available)
                         (not (made ?p))
                         (stacks-avail ?avail))
      :effect (and (not (machine-available)) (machine-configured ?p)))
```

Δράση 2: Κατασκευή προϊόντος

Η συγκεκριμένη δράση έχει τους παραμέτρους προϊόν και θέση στη στοίβα. Για να γίνει γεγονός αυτή η δράση πρέπει να ισχύει ότι η μηχανή είναι έτοιμη να κατασκευάσει το συγκεκριμένο προϊόν και η θέση στη στοίβα να είναι διαθέσιμη. Επίσης , για κάθε παραγγελία που περιέχει το συγκεκριμένο προϊόν να ισχύει ότι έχει ξεκινήσει παραγγελία, δηλαδή δεν βρίσκεται σε αναμονή. Σαν αποτέλεσμα θα δώσει η μηχανή δεν θα είναι ρυθμισμένη, και η μηχανή θα είναι διαθέσιμη και το προϊόν θα είναι πλέον κατασκευασμένο.

```
(:action make-product
      :parameters (?p - product ?avail - count)
      :precondition (and (machine-configured ?p)
                         (forall (?o - order)
                                 (imply (includes ?o ?p) (started ?o)))
                         (stacks-avail ?avail))
     :effect (and (not (machine-configured ?p))
                  (machine-available)
                  (made ?p)))
```

Δράση 3: Ξεκίνημα παραγγελίας

Η συγκεκριμένη δράση έχει τους παραμέτρους τη παραγγελία, τη παρούσα διαθέσιμη θέση στη στοίβα και την επόμενη διαθέσιμη θέση στη στοίβα.. Για να γίνει γεγονός αυτή η δράση πρέπει να ισχύει ότι η παραγγελία βρίσκεται σε αναμονή, η παρούσα θέση στη στοίβα είναι διαθέσιμη, και η επόμενη διαθέσιμη θέση στη στοίβα να είναι ακριβώς πάνω από τη παρούσα διαθέσιμη θέση. Σαν αποτέλεσμα θα δώσει ότι η παραγγελία δεν βρίσκεται πλέον σε αναμονή, η παραγγελία έχει αρχίσει, η μέχρι τώρα διαθέσιμη θέση στη στοίβα δεν είναι πλέον διαθέσιμη και η επόμενη θέση στη στοίβα είναι διαθέσιμη.

```
(:action start-order
   :parameters (?o - order ?avail ?new-avail - count)
   :precondition (and (waiting ?o)
                      (stacks-avail ?avail)
                      (next-count ?new-avail ?avail))
   :effect (and (not (waiting ?o))
                (started ?o)
                (not (stacks-avail ?avail))
                (stacks-avail ?new-avail))
   )
```

Δράση 4: Παράδοση παραγγελίας

Η συγκεκριμένη δράση έχει τους παραμέτρους τη παραγγελία, τη παρούσα διαθέσιμη θέση στη στοίβα και την επόμενη διαθέσιμη θέση στη στοίβα.. Για να γίνει γεγονός αυτή η δράση πρέπει να ισχύει ότι η παραγγελία έχει ξεκινήσει, η παρούσα θέση στη στοίβα είναι διαθέσιμη, και η επόμενη διαθέσιμη θέση στη στοίβα να είναι ακριβώς πάνω από τη παρούσα διαθέσιμη θέση. Επίσης όλα τα προϊόντα που περιέχονται στη παραγγελία να ισχύει ότι έχουν κατασκευαστεί. Σαν αποτέλεσμα θα δώσει ότι η παραγγελία δεν είναι ξεκινημένη, η παραγγελία έχει παραδοθεί, η μέχρι τώρα διαθέσιμη θέση στη στοίβα δεν είναι πλέον διαθέσιμη και η επόμενη θέση στη στοίβα είναι διαθέσιμη.

```
(:action ship-order
   :parameters (?o - order ?avail ?new-avail - count)
   :precondition (and (started ?o)
                      (forall (?p - product)
                              (imply (includes ?o ?p) (made ?p)))
                      (stacks-avail ?avail)
                      (next-count ?avail ?new-avail))
   :effect (and (not (started ?o))
                (shipped ?o)
                (not (stacks-avail ?avail))
                (stacks-avail ?new-avail))
   )
```

Δράση 5: Άνοιγμα καινούριας θέσης στη στοίβα

Η συγκεκριμένη δράση έχει τους παραμέτρους τη παρούσα διαθέσιμη θέση στη στοίβα και την επόμενη διαθέσιμη θέση στη στοίβα.. Για να γίνει γεγονός αυτή η δράση πρέπει να ισχύει ότι η παρούσα θέση στη στοίβα είναι διαθέσιμη, και η επόμενη διαθέσιμη θέση στη στοίβα να είναι ακριβώς πάνω από τη παρούσα διαθέσιμη θέση. Σαν αποτέλεσμα θα δώσει ότι η μέχρι τώρα διαθέσιμη θέση στη στοίβα δεν είναι πλέον διαθέσιμη και η επόμενη θέση στη στοίβα είναι διαθέσιμη.

```
(:action open-new-stack
   :parameters (?open ?new-open - count)
   :precondition (and (stacks-avail ?open)
                      (next-count ?open ?new-open))
   :effect (and (not (stacks-avail ?open))
                (stacks-avail ?new-open))
   )
```

5.3.3 Πρόβλημα βιολογικών μονοπατιών:

Πιο κάτω φαίνεται η περιγραφή του προβλήματος σε προγραμματισμό δράσης.

```
; IPC5 Domain: Pathways Propositional
; Authors: Yannis Dimopoulos, Alfonso Gerevini and Alessandro Saetti

(define (domain Pathways-Propositional)
(:requirements :typing :adl)

(:types level molecule - object
        simple complex - molecule)

(:constants pCAF-p300 pRbp1p2-AP2    - complex)

(:predicates
                (association-reaction ?x1 ?x2 - molecule ?x3 - complex)
                (catalyzed-association-reaction ?x1 ?x2 - molecule ?x3 - complex)
                (synthesis-reaction ?x1 ?x2 - molecule)
                    (possible ?x - molecule)
                (available ?x - molecule)
                    (chosen ?s - simple)
                (next ?l1 ?l2 - level)
                (num-subs ?l - level)
                (goal1))

(:action choose
  :parameters (?x - simple ?l1 ?l2 - level)
  :precondition (and (possible ?x) (not (chosen ?x))
                        (num-subs ?l2) (next ?l1 ?l2))
  :effect (and (chosen ?x) (not (num-subs ?l2)) (num-subs ?l1)))
```

```
(:action initialize
    :parameters (?x - simple)
    :precondition (and (chosen ?x))
    :effect (and (available ?x)))

(:action associate
  :parameters (?x1 ?x2 - molecule ?x3 - complex)
  :precondition (and (association-reaction ?x1    ?x2    ?x3)
                     (available ?x1) (available ?x2))
  :effect (and    (not (available ?x1)) (not (available ?x2)) (available ?x3)))

(:action associate-with-catalyze
  :parameters (?x1 ?x2 - molecule ?x3 - complex)
  :precondition (and (catalyzed-association-reaction ?x1 ?x2 ?x3)
                     (available ?x1) (available ?x2))
  :effect (and (not (available ?x1)) (available ?x3)))

(:action synthesize
  :parameters (?x1 ?x2 - molecule)
  :precondition (and (synthesis-reaction ?x1 ?x2) (available ?x1))
  :effect (and (available ?x2)))

(:action DUMMY-ACTION-1
  :parameters ()
  :precondition
       (or (available pRbp1p2-AP2)
            (available pCAF-p300))
  :effect (and (goal1)))
)
```

Τα κατηγορήματα του προγράμματος είναι τα εξής:

(association-reaction ?x1 ?x2 - molecule ?x3 – complex)
Ενώνονται τα απλά μόρια (x1) και (x2), αντιδρούν και παράγουν το σύνθετο μόριο (x3).

(catalyzed-association-reaction ?x1 ?x2 - molecule ?x3 – complex)
Ενώνονται τα μόρια (x1) και (x2) με καταλύτη, αντιδρούν και παράγουν το σύνθετο μόριο (x3).

(synthesis-reaction ?x1 ?x2 – molecule)
Με σύνθεση από το μόρια (x1) παράγεται το μόριο (x2) .

(possible ?x - molecule)
Δηλώνει ότι το μόριο (x) είναι πιθανόν διαθέσιμο.

(available ?x – molecule)
Δηλώνει ότι το μόριο (x) είναι διαθέσιμο.

(chosen ?s - simple)
Δηλώνει ότι το μόριο (s) είναι επιλεγμένο.

(next ?l1 ?l2 - level)
Το στάδιο (l1) ακολουθείται από το στάδιο (l2).

(num-subs ?l – level)
Δηλώνει τον αριθμό του υποστρώματος σε που ιχσύει.

(goal1))
Ο σκοπός.

Στο πρόβλημα αυτό υπάρχουν 5 δράσεις:

Δράση 1: Επιλογή πιθανού μορίου

Έχει ως παραμέτρους ένα απλό μόριο και δυο υποστρώματα. Για να γίνει αυτή η δράση πρέπει να ισχύει ότι το μόριο είναι πιθανόν και μέχρις στιγμής δεν έχει επιλεχθεί. Επίσης το υπόστρωμα που ισχύει να είναι το προηγούμενο του επόμενου υποστρώματος. Και σαν αποτέλεσμα έχουμε ότι το μόριο να επιλέγετε, το προηγούμενο υπόστρωμα να ισχύει ενώ το προηγούμενο που ίσχυες να μην ισχύει πλέον.

```
(:action choose
  :parameters (?x - simple ?l1 ?l2 - level)
  :precondition (and (possible ?x) (not (chosen ?x))
                     (num-subs ?l2) (next ?l1 ?l2))
  :effect (and (chosen ?x) (not (num-subs ?l2)) (num-subs ?l1)))
```

Δράση 2: Αρχικοποίηση του μορίου

Στη δράση αυτή έχουμε ως μοναδική παράμετρο ένα απλό μόριο. Προϋπόθεση για αυτή τη δράση είναι το μόριο να έχει επιλεχθεί και σαν αποτέλεσμα θα ισχύει ότι το μόριο θα είναι πλέον διαθέσιμο.

```
(:action initialize
   :parameters (?x - simple)
   :precondition (and (chosen ?x))
   :effect (and (available ?x)))
```

Δράση 3: Απλή Αντίδραση

Η δράση αυτή έχει σαν παραμέτρους δυο μόρια και ακόμα ένα σύνθετο μόριο. Προϋποθέσεις για αυτή τη δράση είναι τα δυο πρώτα μόρια να είναι διαθέσιμα και να ισχύει ότι με την μεταξύ τους αντίδραση, παράγετε το σύνθετο μόριο. Σαν αποτέλεσμα αυτής της αντίδρασης, λογικά , έχουμε ότι τα μόρια που έχουν κάνει την αντίδραση πλέον δεν είναι διαθέσιμα και είναι διαθέσιμο μόνο το σύνθετο μόριο που έχει παραχθεί.

```
(:action associate
  :parameters (?x1 ?x2 - molecule ?x3 - complex)
  :precondition (and (association-reaction ?x1  ?x2  ?x3)
                     (available ?x1) (available ?x2))
  :effect (and  (not (available ?x1)) (not (available ?x2)) (available ?x3)))
```

Δράση 4: Αντίδραση

Η δράση αυτή έχει σαν παραμέτρους δυο μόρια και ακόμα ένα σύνθετο μόριο. Προϋποθέσεις για αυτή τη δράση είναι τα δυο πρώτα μόρια να είναι διαθέσιμα και να ισχύει ότι με την μεταξύ τους αντίδραση στη παρουσία καταλύτη παράγετε το σύνθετο μόριο. Σαν αποτέλεσμα αυτής της αντίδρασης, έχουμε ότι το πρώτο μόριο που αντιδρά πλέον δεν είναι διαθέσιμο ενώ το δεύτερο εξακολουθεί να είναι διαθέσιμο. Επίσης το σύνθετο μόριο είναι διαθέσιμο.

```
(:action associate-with-catalyze
  :parameters (?x1 ?x2 - molecule ?x3 - complex)
  :precondition (and (catalyzed-association-reaction ?x1 ?x2 ?x3)
                     (available ?x1) (available ?x2))
  :effect (and (not (available ?x1)) (available ?x3)))
```

Δράση 5: Σύνθεση

Η δράση αυτή έχει σαν παραμέτρους δυο μόρια. Προϋποθέσεις για αυτή τη δράση είναι το πρώτο μόριο να είναι διαθέσιμο και μεταξύ τους να ισχύει η σύνθεση μεταξύ τους. Σαν αποτέλεσμα αυτής της δράσης αυτής, έχουμε ότι και το δεύτερο μόριο είναι διαθέσιμο.

```
(:action synthesize
  :parameters (?x1 ?x2 - molecule)
  :precondition (and (synthesis-reaction ?x1 ?x2) (available ?x1))
  :effect (and (available ?x2)))
```

Δράση 6: Σκοπός

Πιο κάτω είναι ένα παράδειγμα κάποιου σκοπού όπου για να ισχύσει αυτή η δράση πρέπει ένα από τα δυο μόρια (pRbp1p2-AP2 ή pCAF-p300) να γίνει διαθέσιμο.

```
  (:action DUMMY-ACTION-1
  :parameters ()
  :precondition
       (or (available pRbp1p2-AP2)
            (available pCAF-p300))
  :effect (and (goal1)))
)
```

5.4 Περιγραφή προβλημάτων σε λογικό προγραμματισμό

Μετά από μελέτη των προβλημάτων στο προγραμματισμό δράσεων μετέτρεψα τα προβλήματα σε προγραμματισμό συνόλου απαντήσεων.

5.4.1 Πρόβλημα δρομολόγησης φορτηγών:

Πιο κάτω θα περιγράψω τις δράσεις για το πρόβλημα αυτό τις οποίες έχω μετατρέψει σύμφωνα με τις απαιτήσεις του προγραμματισμού συνόλου απαντήσεων.

Δράση 1: Φόρτωμα

Υποθέτω ότι το φορτηγό έχει τρεις θέσεις και έτσι τη δράση φόρτωμα τη διαχωρίζω σε τρεις διαφορετικές περιπτώσεις.

Περίπτωση 1:

Ένα πακέτο μπορεί να φορτωθεί σε ένα φορτηγό που βρίσκεται σε μια συγκεκριμένη τοποθεσία, στη εξωτερική θέση του φορτηγού για μια χρονική στιγμή αν ισχύει ότι το φορτηγό και το πακέτο βρίσκονται στη συγκεκριμένη τοποθεσία και η εξωτερική θέση είναι άδεια τη συγκεκριμένη χρονική στιγμή .

{load(Pa,Tr,L,a1,T)} :- at(Tr,L,T),
at(Pa,L,T),
free(a1,Tr,T),
package(Pa),
truck(Tr),location(L),truckarea(a1),time(T).

Περίπτωση 2:

Ένα πακέτο μπορεί να φορτωθεί σε ένα φορτηγό που βρίσκεται σε μια συγκεκριμένη τοποθεσία, στη δεύτερη θέση του φορτηγού για μια χρονική στιγμή αν ισχύει ότι το φορτηγό και το πακέτο βρίσκονται στη συγκεκριμένη τοποθεσία και επίσης η εξωτερική θέση και δεύτερη θέση του είναι άδεια τη συγκεκριμένη χρονική στιγμή .

{load(Pa,Tr,L,a2,T)} :- at(Tr,L,T),
at(Pa,L,T),
free(a1,Tr,T),free(a2,Tr,T),
package(Pa),truck(Tr),location(L),
truckarea(a1),truckarea(a2),time(T).

Περίπτωση 3:

Ένα πακέτο μπορεί να φορτωθεί σε ένα φορτηγό που βρίσκεται σε μια συγκεκριμένη τοποθεσία, στη τρίτη θέση (στην πιο μέσα) του φορτηγού για μια χρονική στιγμή αν ισχύει ότι το φορτηγό και το πακέτο βρίσκονται στη συγκεκριμένη τοποθεσία και επίσης η εξωτερική θέση και δεύτερη θέση του είναι άδεια τη συγκεκριμένη χρονική στιγμή .

{load(Pa,Tr,L,a3,T)} :- at(Tr,L,T),
at(Pa,L,T),
free(a1,Tr,T),free(a2,Tr,T),free(a3,Tr,T),
package(Pa),truck(Tr),location(L),
truckarea(a1),truckarea(a2),truckarea(a3),time(T).

Σαν αποτέλεσμα ου φορτώματος έχουμε το πακέτο να είναι μέσα στο φορτηγό στη συγκεκριμένη θέση .

in(Pa,Tr,Ta,T+1):- load(Pa,Tr,L,Ta,T), package(Pa),

truck(Tr),location(L),truckarea(Ta),time(T).

Η κατάσταση της θέσης έχει αλλάξει.

change_free_ta(Ta,Tr,T):- load(Pa,Tr,L,Ta,T),package(Pa),truck(Tr),
location(L),truckarea(Ta),time(T).

Η κατάσταση του πακέτου να έχει αλλάξει

change(Pa,T):- load(Pa,Tr,L,Ta,T),package(Pa),truck(Tr),
location(L),truckarea(Ta),time(T).

Το φορτηγό εξακολουθεί να βρίσκεται στη συγκεκριμένη τοποθεσία.

at(Tr,L,T+1):- load(Pa,Tr,L,Ta,T)package(Pa),truck(Tr),
location(L),truckarea(Ta),time(T).

Δράση 2: Ξεφόρτωμα

Το φορτηγό ξεφορτώνει ένα πακέτο όταν στη τοποθεσία την οποία βρίσκεται είναι ο προορισμός του πακέτου. Διαχωρίζω τη δράση πάλι σε τρεις περιπτώσεις ανάλογα με τη θέση που έχει το πακέτο στο φορτηγό.

Περίπτωση 1:

Αν το πακέτο βρίσκεται τοποθετημένο στη εξωτερική θέση του φορτηγού τότε το πακέτο μπορεί να ξεφορτωθεί.

{unload(Pa,Tr,L,a1,T)}:-in(Pa,Tr,a1,T),
at(Tr,L,T),

package(Pa),truck(Tr),location(L),truckarea(a1),time(T).

Περίπτωση 2:

Αν το πακέτο βρίσκεται τοποθετημένο στη μεσαία θέση του φορτηγού και η εξωτερική θέση είναι άδεια τότε το πακέτο μπορεί να ξεφορτωθεί.

{unload(Pa,Tr,L,a2,T)}:-in(Pa,Tr,a2,T),
at(Tr,L,T),
free(Tr,a1,T),
package(Pa),truck(Tr),location(L), truckarea(a1),truckarea(a2),time(T).

Περίπτωση 3:

Αν το πακέτο βρίσκεται τοποθετημένο στη μέσα θέση του φορτηγού και επίσης η μεσαία θέση και η εξωτερική θέση είναι άδεια τότε το πακέτο μπορεί να ξεφορτωθεί.

{unload(Pa,Tr,L,a3,T)}:-in(Pa,Tr,a3,T),
at(Tr,L,T),
free(a1,Tr,T),free(a2,Tr,T),
package(Pa),truck(Tr),location(L),truckarea(a1),truckarea(a2),truckarea(a3),
time(T).

Αποτέλεσμα αυτής της δράσης είναι η θέση του φορτηγού που ήταν φορτωμένο το πακέτο πλέον είναι άδεια.

free(Ta,Tr,T+1):- unload(Pa,Tr,L,Ta,T),
package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

Το φορτηγό εξακολουθεί να βρίσκεται στη συγκεκριμένη τοποθεσία.

at(Tr,L,T+1) :- unload(Pa,Tr,L,Ta,T),
package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

Η κατάσταση του πακέτου αλλάζει.

change(Pa,T):- unload(Pa,Tr,L,Ta,T)
,package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

Το πακέτο πάει στο προορισμό του.

at_destination(Pa,L) :-
unload(Pa,Tr,L,Ta,T),package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

Το πακέτο παραδίνεται στο προορισμό του την επόμενη χρονική στιγμή από το ξεφόρτωμά του.

delivered(Pa,L,T+1) :-
unload(Pa,Tr,L,Ta,T),package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

Δράση 3: Οδήγημα

Το φορτηγό οδηγεί από μια τοποθεσία σε άλλη αυτό πρώτα βρίσκεται στη πρώτη τοποθεσία τη συγκεκριμένη χρονική στιγμή και επίσης οι δυο τοποθεσίες είναι ενωμένες..

{drive(Tr,L1,L2,T)}:-at(Tr,L1,T),
connected(L1,L2),
truck(Tr),
location(L1),location(L2),L1!=L2,

time(T).

Επομένως την επόμενη χρονική στιγμή το φορτηγό βρίσκεται στη δεύτερη τοποθεσία.

at(Tr,L2,T+1) :-drive(Tr,L1,L2,T),truck(Tr),location(L1),location(L2),time(T).

Και η κατάσταση του φορτηγού αλλάζει.

change(Tr,T):- drive(Tr,L1,L2,T) ,truck(Tr),location(L1),location(L2),time(T).

Γενικοί κανόνες:

Αν ένα πακέτο μια συγκεκριμένη χρονική στιγμή βρίσκεται σε μια τοποθεσία και δεν έχει υποστεί κάποια αλλαγή τότε και την επόμενη χρονική στιγμή εξακολουθεί να βρίσκεται στην ίδια τοποθεσία.

at(Pa,L,T+1) :-at(Pa,L,T),not change(Pa,T),package(Pa),location(L),time(T).

Αν ένα πακέτο μια συγκεκριμένη χρονική στιγμή βρίσκεται μέσα σε ένα φορτηγό σε κάποια θέση και δεν έχει υποστεί κάποια αλλαγή τότε και την επόμενη χρονική στιγμή εξακολουθεί να βρίσκεται μέσα στο φορτηγό στην ίδια θέση.

in(Pa,Tr,Ta,T+1):-in(Pa,Tr,Ta,T),not change(Pa,T),package(Pa), truck(Tr),truckarea(Ta),time(T).

Αν ένα φορτηγό μια συγκεκριμένη χρονική στιγμή βρίσκεται σε μια τοποθεσία και δεν έχει υποστεί κάποια αλλαγή τότε και την επόμενη χρονική στιγμή εξακολουθεί να βρίσκεται ίδια τοποθεσία.

at(Tr,L,T+1) :-at(Tr,L,T),not change(Tr,T),truck(Tr),location(L),time(T).

Αν μια θέση σε ένα φορτηγό είναι άδεια μια συγκεκριμένη χρονική στιγμή και δεν έχει υποστεί έχει αλλάξει η κατάστασή της τότε εξακολουθεί να είναι άδεια κ την επόμενη χρονική στιγμή.

free(Ta,Tr,T+1):- free(Ta,Tr,T),not change_free_ta(Ta,Tr,T), truck(Tr), truckarea(Ta),time(T).

Αν ένα πακέτο μια συγκεκριμένη χρονική στιγμή βρίσκεται μέσα σε ένα φορτηγό και δεν έχει υποστεί κάποια αλλαγή τότε και την επόμενη χρονική στιγμή εξακολουθεί να βρίσκεται μέσα στο φορτηγό.

intruck(Pa,Tr,T):-in(Pa,Tr,Ta,T), package(Pa), truck(Tr), truckarea(Ta), time(T).

Δεν γίνεται να φορτώνονται δυο διαφορετικά πακέτα στο ίδιο φορτηγό την ίδια χρονική στιγμή σε δυο διαφορετικές θέσεις.

:-load(Pa1,Tr,L,Ta1,T),load(Pa2,Tr,L,Ta2,T),

package(Pa1),package(Pa2),Pa1!=Pa2,location(L),truck(Tr),truckarea(Ta1),truckarea(Ta2),Ta1!=Ta2,time(T).

Δεν γίνεται να φορτώνονται δυο διαφορετικά πακέτα στο ίδιο φορτηγό την ίδια χρονική στιγμή στην ίδια θέση.

:-load(Pa1,Tr,L,Ta,T),load(Pa2,Tr,L,Ta,T),

package(Pa1),package(Pa2),Pa1!=Pa2,location(L),truck(Tr),truckarea(Ta),time(T).

package(Pa1),package(Pa),location(L),truck(Tr),truckarea(Ta1),truckarea(Ta2),time(T),Ta1!=Ta2.

Δεν γίνεται να φορτώνεται το ίδιο πακέτο την ίδια χρονική στιγμή σε δυο διαφορετικές θέσεις.

:-load(Pa,Tr,L,Ta1,T),load(Pa,Tr,L,Ta2,T),

Δεν γίνεται να ένα φορτηγό να κατευθύνεται σε δυο διαφορετικές τοποθεσίες..

:-drive(Tr,L1,L2,T),drive(Tr,L1,L3,T),
location(L1),location(L2),location(L3),truck(Tr),time(T),L2!=L3.

package(Pa),location(L1),location(L2),truckarea(Ta),truck(Tr),time(T),L1!=L2.

Δεν γίνεται να ένα φορτηγό να φορτώνει και να ξεφορτώνει την ίδια χρονική στιγμή.

:-drive(Tr,L1,L2,T),load(Pa,Tr,L1,Ta,T),

Δεν γίνεται να ένα φορτηγό να οδηγά και να ξεφορτώνει την ίδια χρονική στιγμή.

:-drive(Tr,L1,L2,T),unload(Pa,Tr,L1,Ta,T),

package(Pa),location(L1),location(L2),truckarea(Ta),truck(Tr),time(T),L1!=L2.

Δεν γίνεται να ένα φορτηγό να φορτώνει και να ξεφορτώνει την ίδια χρονική στιγμή σε δυο διαφορετικές θέσεις του φορτηγού**.**

:-load(Pa1,Tr,L,Ta1,T),unload(Pa2,Tr,L,Ta2,T),

package(Pa1),package(Pa2),truck(Tr),location(L),truckarea(Ta1),truckarea(Ta2),Ta1!=Ta2,time(T),Pa1!=Pa2.

5.4.2 Πρόβλημα ανοικτής στοίβας εργασιών:

Πιο κάτω θα περιγράψω τις δράσεις για το πρόβλημα αυτό τις οποίες έχω μετατρέψει σύμφωνα με τις απαιτήσεις του προγραμματισμού συνόλου απαντήσεων.

Δράση 1: Ρύθμιση μηχανής

Η δράση ρύθμιση της μηχανής για κάποιο προϊόν, μια χρονική στιγμή, σε μια θέση στη στοίβα προϋποθέτει ότι η μηχανή τη συγκεκριμένη χρονική στιγμή είναι διαθέσιμη το προϊόν δεν έχει κατασκευαστεί και η συγκεκριμένη θέση στη στοίβα είναι η διαθέσιμη.

{setup_machine(Prod,Avail,T)}:- machine_available(T),
not made(Prod,T),
stacks_avail(Avail,T),
product(Prod),count(Avail),time(T).

Σαν αποτέλεσμα η μηχανή την επόμενη χρονική στιγμή είναι έτοιμη να παράγει το συγκεκριμένο προϊόν.

machine_configured(Prod,T+1):-
setup_machine(Prod,Avail,T),product(Prod),count(Avail),time(T).

Επίσης η κατάσταση της μηχανής έχει αλλάξει.

delete_mach_available(T):-
setup_machine(Prod,Avail,T),product(Prod),count(Avail),time(T).

Δράση 2: Κατασκευή προϊόντος

Υποθέτω ότι υπάρχουν πέντε παραγγελίες και έτσι τη δράση κατασκευή προϊόντος τη διαχωρίζω σε πέντε διαφορετικές περιπτώσεις.

Περίπτωση 1:

Κατασκευάζεται ένα προϊόν, σε συγκεκριμένη θέση στη στοίβα, για τη πρώτη παραγγελιά, μια χρονική στιγμή, αν η μηχανή είναι έτοιμη να παράγει το συγκεκριμένο προϊόν, τη συγκεκριμένη χρονική στιγμή και αν η θέση στη στοίβα είναι η διαθέσιμη. Επίσης πρέπει να ισχύει ότι το συγκεκριμένο προϊόν συμπεριλαμβάνεται στη πρώτη παραγγελιά και η πρώτη παραγγελιά έχει ήδη ξεκινήσει, δεν βρίσκεται δηλαδή σε κατάσταση αναμονής.

{make_product(Prod,Avail,o1,T)} :- machine_configured(Prod,T),
stacks_avail(Avail,T),
includes(o1,Prod),started(o1,T),
product(Prod),count(Avail),time(T).

Περίπτωση 2:

Κατασκευάζεται ένα προϊόν, σε συγκεκριμένη θέση στη στοίβα, για τη δεύτερη παραγγελιά, μια χρονική στιγμή, αν η μηχανή είναι έτοιμη να παράγει το συγκεκριμένο προϊόν, τη συγκεκριμένη χρονική στιγμή και αν η θέση στη στοίβα είναι η διαθέσιμη. Επίσης πρέπει να ισχύει ότι το συγκεκριμένο προϊόν συμπεριλαμβάνεται στη δεύτερη παραγγελιά και η δεύτερη παραγγελιά έχει ήδη ξεκινήσει, δεν βρίσκεται δηλαδή σε κατάσταση αναμονής.

{make_product(Prod,Avail,o2,T)} :- machine_configured(Prod,T),
stacks_avail(Avail,T),
includes(o2,Prod),started(o2,T),
product(Prod),count(Avail),time(T).

Περίπτωση 3:

Κατασκευάζεται ένα προϊόν, σε συγκεκριμένη θέση στη στοίβα, για τη τρίτη παραγγελιά, μια χρονική στιγμή, αν η μηχανή είναι έτοιμη να παράγει το συγκεκριμένο προϊόν, τη συγκεκριμένη χρονική στιγμή και αν η θέση στη στοίβα είναι η διαθέσιμη. Επίσης πρέπει να ισχύει ότι το συγκεκριμένο προϊόν συμπεριλαμβάνεται στη τρίτη παραγγελιά και η τρίτη παραγγελιά έχει ήδη ξεκινήσει, δεν βρίσκεται δηλαδή σε κατάσταση αναμονής.

{make_product(Prod,Avail,o3,T)} :- machine_configured(Prod,T),
stacks_avail(Avail,T),

includes(o2,Prod),started(o3,T),
product(Prod),count(Avail),time(T).

Περίπτωση 4:

Κατασκευάζεται ένα προϊόν, σε συγκεκριμένη θέση στη στοίβα, για τη τέταρτη παραγγελιά, μια χρονική στιγμή, αν η μηχανή είναι έτοιμη να παράγει το συγκεκριμένο προϊόν, τη συγκεκριμένη χρονική στιγμή και αν η θέση στη στοίβα είναι η διαθέσιμη. Επίσης πρέπει να ισχύει ότι το συγκεκριμένο προϊόν συμπεριλαμβάνεται στη τέταρτη παραγγελιά και η τέταρτη παραγγελιά έχει ήδη ξεκινήσει, δεν βρίσκεται δηλαδή σε κατάσταση αναμονής.

{make_product(Prod,Avail,o4,T)} :- machine_configured(Prod,T),
stacks_avail(Avail,T),
includes(o4,Prod),started(o4,T),
product(Prod),count(Avail),time(T).

Περίπτωση 5:

Κατασκευάζεται ένα προϊόν, σε συγκεκριμένη θέση στη στοίβα, για τη πέμπτη παραγγελιά, μια χρονική στιγμή, αν η μηχανή είναι έτοιμη να παράγει το συγκεκριμένο προϊόν, τη συγκεκριμένη χρονική στιγμή και αν η θέση στη στοίβα είναι η διαθέσιμη. Επίσης πρέπει να ισχύει ότι το συγκεκριμένο προϊόν συμπεριλαμβάνεται στη πέμπτη παραγγελιά και η πέμπτη παραγγελιά έχει ήδη ξεκινήσει, δεν βρίσκεται δηλαδή σε κατάσταση αναμονής.

{make_product(Prod,Avail,o5,T)} :- machine_configured(Prod,T),
stacks_avail(Avail,T),
includes(o5,Prod),started(o2,T),
product(Prod),count(Avail),time(T).

Αποτέλεσμα της πιο πάνω δράσης είναι η μηχανή να είναι διαθέσιμη τη συγκεκριμένη χρονική στιγμή.

machine_available(T+1) :-
make_product(Prod,Avail,Ord,T),product(Prod),count(Avail),order(Ord),time(T).

Επίσης το προϊόν που παρασκευάστηκε θα είναι διαθέσιμο την επόμενη χρονική στιγμή,

made(Prod,T+1):-
make_product(Prod,Avail,Ord,T),product(Prod),count(Avail),order(Ord),time(T).

Και επιπλέον έχει σημειωθεί αλλαγή στη κατάσταση της μηχανής.

delete_mach_config(Prod,T):-
make_product(Prod,Avail,Ord,T),product(Prod),count(Avail),order(Ord),time(T).

Δράση 3: Ξεκίνημα παραγγελίας

Η δράση ξεκίνημα παραγγελιάς, σε μια θέση της στοίβας, μια χρονική στιγμή προϋποθέτει ότι η παραγγελιά βρίσκεται σε κατάσταση αναμονής, η θέση στη στοίβα είναι διαθέσιμη, και υπάρχει μια θέση ακριβώς πιο κάτω από τη θέση αυτή η οποία στη συνεχεία θα είναι διαθέσιμη.

{start_order(Ord,Avail,NewAvail,T)}:- waiting(Ord,T),

stacks_avail(Avail,T),next_count(NewAvail,Avail),

order(Ord),count(Avail),count(NewAvail),time(T).

Σαν αποτέλεσμα της δράσης η παραγγελία ξεκινά την επόμενη χρονική στιγμή.

started(Ord,T+1):-
start_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail),time(T).

Η θέση της στοίβας ακριβώς πιο κάτω από τη θέση που ξεκινά η παραγγελία είναι διαθέσιμη την επόμενη χρονική στιγμή.

stacks_avail(NewAvail,T+1):-
start_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail),time(T).

Η κατάσταση της θέσης στη στοίβα που ξεκινά η παραγγελία ξαλλάζει.

delete_stacks_avail(Avail,T):-
start_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail),time(T).

Η κατάστασης αναμονής της παραγγελίας ξαλλάζει.

delete_waiting(Ord,T):-
start_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail),time(T).

Δράση 4: Παράδοση παραγγελίας

Αφού υπάρχουν πέντε παραγγελίες και πάλι χωρίζουμε τη δράση σε πέντε διαφορετικές περιπτώσεις.

Περίπτωση 1:

Η πρώτη παραγγελία παραδίνεται από μια θέση στη στοίβα , μια χρονική στιγμή αν η παραγγελία έχει ξεκινήσει και αν όλα τα προϊόντα τα οποία περιέχονται αυτή έχουν παραχθεί.

{ship_order(o1,Avail,NewAvail,T)}:-
order(o1),started(o1,T),product(p1),product(p2),includes(o1,p1),includes(o1,p2),
made(p1,T),made(p2,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).

Περίπτωση 2:

Η δεύτερη παραγγελία παραδίνεται από μια θέση στη στοίβα , μια χρονική στιγμή αν η παραγγελία έχει ξεκινήσει και αν όλα τα προϊόντα τα οποία περιέχονται αυτή έχουν παραχθεί.

{ship_order(o2,Avail,NewAvail,T)}:-
order(o2),started(o2,T),product(p3),product(p4),includes(o2,p3),includes(o2,p4),
made(p3,T),made(p4,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).

Περίπτωση 3:

Η τρίτη παραγγελία παραδίνεται από μια θέση στη στοίβα , μια χρονική στιγμή αν η παραγγελία έχει ξεκινήσει και αν όλα τα προϊόντα τα οποία περιέχονται αυτή έχουν παραχθεί.

{ship_order(o3,Avail,NewAvail,T)}:-
order(o3),started(o3,T),product(p1),product(p5),includes(o3,p1),includes(o3,p5),
made(p1,T),made(p5,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).

Περίπτωση 4:

Η τέταρτη παραγγελία παραδίνεται από μια θέση στη στοίβα , μια χρονική στιγμή αν η παραγγελία έχει ξεκινήσει και αν όλα τα προϊόντα τα οποία περιέχονται αυτή έχουν παραχθεί.

{ship_order(o4,Avail,NewAvail,T)}:-
order(o4),started(o4,T),product(p3),product(p5),includes(o4,p3),includes(o4,p5),
made(p3,T),made(p5,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).

Περίπτωση 5:

Η πέμπτη παραγγελία παραδίνεται από μια θέση στη στοίβα , μια χρονική στιγμή αν η παραγγελία έχει ξεκινήσει και αν όλα τα προϊόντα τα οποία περιέχονται αυτή έχουν παραχθεί.

{ship_order(o5,Avail,NewAvail,T)}:-
order(o5),started(o5,T),product(p2),product(p4),includes(o5,p2),includes(o5,p4),
made(p2,T),made(p4,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).

Σαν αποτέλεσμα η παραγγελία παραδίνεται.

shipped(Ord):-
ship_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail),time(T).

Η αμέσως πιο πάνω θέση στη στοίβα είναι διαθέσιμη την επόμενη χρονική στιγμή.

stacks_avail(NewAvail,T+1):-
ship_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail),time(T).

Επίσης η θέση στη στοίβα που ήταν διαθέσιμη έχει υποστεί αλλαγές.

delete_stacks_avail(Avail,T):-
ship_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail),time(T).

Και η παραγγελία δεν θεωρείται πλέον ότι είναι ξεκινημένη.

delete_started(Ord,T):-
ship_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail),time(T).

Δράση 5: Άνοιγμα καινούριας θέσης στη στοίβα

Για να ανοιχθεί μια καινούρια θέση στη στοίβα μια χρονική στιγμή πρέπει να υπάρχει μια θέση στη στοίβα διαθέσιμη, πάνω από την οποία υπάρχει μια άλλη.

{open_new_stack(Open,NewOpen,T)}:-
stacks_avail(Open,T),next_count(Open,NewOpen),
count(Open),count(NewOpen),time(T).

Έτσι η καινούρια διαθέσιμη θέση στη στοίβα είναι αυτή που υπάρχει πάνω από τη προηγούμενη.

stacks_avail(NewOpen,T+1) :-
open_new_stack(Open,NewOpen,T),count(Open),count(NewOpen),time(T).

Η προηγούμενη θέση στη στοίβα έχει αλλάξει κατάσταση αφού πλέον δεν είναι διαθέσιμη.

delete_stacks_avail(Open,T):-
open_new_stack(Open,NewOpen,T),count(Open),count(NewOpen),time(T).

Γενικοί κανόνες:

Όταν μια παραγγελία βρίσκεται σε κατάσταση αναμονής μια στιγμή και δεν έχει υποστεί αλλαγή τότε και την επόμενη χρονική στιγμή βρίσκεται σε κατάσταση αναμονής.

waiting(Ord,T+1):-waiting(Ord,T),not delete_waiting(Ord,T),order(Ord),time(T).

Μια θέση στη στοίβα είναι διαθέσιμη και την επόμενη χρονική στιγμή αν την προηγουμένη χρονική στιγμή ήταν διαθέσιμη και δεν έχει υποστεί κάποια αλλαγή.

stacks_avail(Avail,T+1):-stacks_avail(Avail,T),not
delete_stacks_avail(Avail,T),count(Avail),time(T).

Η μηχανή είναι διαθέσιμη και τη επόμενη χρονική στιγμή αν δεν έχει υποστεί αλλαγή την προηγούμενη χρονική στιγμή.

machine_available(T+1):-machine_available(T),not
delete_mach_available(T),time(T).

Η μηχανή είναι ρυθμισμένη για να κατασκευάσει ένα προϊόν και τη επόμενη χρονική στιγμή αν δεν έχει υποστεί αλλαγή την προηγούμενη χρονική στιγμή.

machine_configured(Prod,T+1):-machine_configured(Prod,T),not
delete_mach_config(Prod,T),product(Prod),time(T).

Μια παραγγελία θεωρείται ξεκινημένη και την επόμενη χρονική στιγμή αν δεν έχει υποστεί αλλαγή.

started(Ord,T+1):-started(Ord,T),not delete_started(Ord,T),order(Ord),time(T).

Ένα προϊόν είναι κατασκευασμένο αν τη προηγουμένη χρονική στιγμή ήταν κατασκευασμένο. Φτάνει μόνο να κατασκευαστεί μια φορά.

made(Prod,T+1):-made(Prod,T),product(Prod),time(T).

Δεν επιτρέπεται δυο δράσεις να γίνονται ταυτόχρονα. Ο λόγος είναι ότι κάθε δράση επιφέρει αλλαγή στη διαθέσιμη θέση στη στοίβα. σε κάθε χρόνο μόνο μια θέση πρέπει να είναι διαθέσιμη. Έτσι δεν επιτρέπεται δυο δράσεις να συμβαίνουν την ίδια στιγμή.

Δεν επιτρέπεται να γίνεται ταυτόχρονα η ρύθμιση της μηχανής και το ξεκίνημα παραγγελίας.

:-setup_machine(Prod,Avail,T),
start_order(Ord,Avail,NewAvail,T),
product(Prod),order(Ord),count(Avail),count(NewAvail),time(T).

Δεν επιτρέπεται να γίνεται ταυτόχρονα η ρύθμιση της μηχανής και η παράδοση παραγγελίας.

```
:-setup_machine(Prod,Avail,T),
  ship_order(Ord,Avail,NewAvail,T),
  product(Prod),order(Ord),count(Avail),count(NewAvail),time(T).
```

Δεν επιτρέπεται να γίνεται ταυτόχρονα η ρύθμιση της μηχανής και το άνοιγμα καινούριας θέσης στη στοίβα.

```
:-setup_machine(Prod,Avail,T),
  open_new_stack(Avail,NewOpen,T),
  product(Prod),count(Avail),count(NewOpen),time(T).
```

Δεν επιτρέπεται να γίνεται ταυτόχρονα η παραγωγή κάποιου προϊόντος και το ξεκίνημα παραγγελίας.

```
:-make_product(Prod,Avail,Ord1,T),
  start_order(Ord2,Avail,NewAvail,T),
  product(Prod),order(Ord1),order(Ord2),
  count(Avail),count(NewAvail),time(T),Ord1!=Ord2.
```

Δεν επιτρέπεται να γίνεται ταυτόχρονα η παραγωγή κάποιου προϊόντος και η παράδοση κάποιας παραγγελίας.

```
:-make_product(Prod,Avail,Ord1,T),
  ship_order(Ord2,Avail,NewAvail,T),
  product(Prod),order(Ord1),order(Ord2),
  count(Avail),count(NewAvail),time(T),Ord1!=Ord2.
```

Δεν επιτρέπεται να γίνεται ταυτόχρονα η παραγωγή κάποιου προϊόντος και το άνοιγμα καινούριας θέσης στη στοίβα.

```
:-make_product(Prod,Avail,Ord1,T),
  open_new_stack(Avail,NewOpen,T),
  product(Prod),order(Ord1),count(Avail),count(NewOpen),time(T).
```

Δεν επιτρέπεται να γίνεται ταυτόχρονα το ξεκίνημα παραγγελίας και η παράδοση κάποιας παραγγελίας άλλης.

```
:-start_order(Ord1,Avail,NewAvail,T),
  ship_order(Ord2,Avail,NewAvail_1,T),
  order(Ord1),order(Ord2),Ord1!=Ord2,

count(Avail),count(NewAvail),count(NewAvail_1),NewAvail!=NewAvail_1,time(T).
```

Δεν επιτρέπεται να γίνεται ταυτόχρονα το ξεκίνημα παραγγελίας και το άνοιγμα καινούριας θέσης στη στοίβα.

```
:-start_order(Ord,Avail,NewAvail,T),
  open_new_stack(Avail,NewOpen,T),
```

order(Ord),count(Avail),count(NewAvail),count(NewOpen),
NewAvail!=NewOpen,time(T).

Δεν επιτρέπεται να γίνεται ταυτόχρονα η παράδοση παραγγελίας και το άνοιγμα καινούριας θέσης στη στοίβα.

:-ship_order(Ord,Avail,NewAvail,T),
open_new_stack(Avail,NewAvail,T),order(Ord),
count(Avail),count(NewAvail),time(T).

Δεν επιτρέπεται να γίνεται ταυτόχρονα το ξεκίνημα δυο παραγγελίων.

:-2{start_order(Ord,Avail,NewAvail,T):count(Avail):count(NewAvail):order(Ord)},time(T).

Δεν επιτρέπεται να γίνεται ταυτόχρονα ρύθμιση μηχανής για δυο προϊόντα.

:-2{setup_machine(Prod,Avail,T):count(Avail):product(Prod)},time(T).

Δεν επιτρέπεται να γίνεται ταυτόχρονα η παράδοση δυο παραγγελιών.

:-2{ship_order(Ord,Avail,NewAvail,T):count(Avail):count(NewAvail):order(Ord)},time(T).

5.4.3 Πρόβλημα βιολογικών μονοπατιών:

Πιο κάτω θα περιγράψω τις δράσεις για το πρόβλημα αυτό τις οποίες έχω μετατρέψει σύμφωνα με τις απαιτήσεις του προγραμματισμού συνόλου απαντήσεων.

Δράση 1: Επιλογή πιθανού μορίου

Ένα μόριο επιλέγεται αν είναι απλό, αν είναι πιθανόν και επιπλέον αν δεν έχει επιλεχθεί μέχρι στιγμής. Επίσης πρέπει να γνωρίζουμε σε πιο υπόστρωμα βρισκόμαστε τη συγκεκριμένη χρονική στιγμή, καθώς επίσης και το επόμενο υπόστρωμα.

{choose(X,L1,L2,T)}:-simple(X),level(L1),level(L2),possible(X),
not chosen(X,T), num_subs(L2,T), next(L1,L2),time(T).

Αποτέλεσμα της δράσης αυτής είναι το μόριο πλέον να έχει επιλεχθεί.

chosen(X,T+1):-choose(X,L1,L2,T),simple(X),level(L1),level(L2),time(T).

Το νέο υπόστρωμα να ισχύει από τη επόμενη χρονική στιγμή.

num_subs(L1,T+1):-choose(X,L1,L2,T),simple(X),level(L1),level(L2),time(T).

Και το προηγούμενο υπόστρωμα να έχει υποστεί αλλαγή αφού πλέον δεν θα είναι το τρέχων υπόστρωμα.

change_subs(L2,T):-choose(X,L1,L2,T),simple(X),level(L1),level(L2),time(T).

Δράση 2: Αρχικοποίηση του μορίου

Ένα απλό μόριο αρχικοποιείται σε κάποια χρονική στιγμή αν έχει επιλεχθεί.

{initialize(X,T)} :- chosen(X,T),simple(X),time(T).

Σαν αποτέλεσμα της αρχικοποίησης του μορίου έχουμε ότι το μόριο την επόμενη χρονική στιγμή είναι διαθέσιμο.

available(X,T+1):-initialize(X,T), simple(X),time(T).

Δράση 3: Απλή αντίδραση

Για να γίνει αυτή η δράση γεγονός κάποια στιγμή πρέπει να υπάρχουν διαθέσιμα δυο μόρια τα οποία μπορούν να αντιδράσουν μεταξύ τους παράγοντας ένα σύνθετο μόριο.

{associate(X1,X2,X3,T)}:- molecule(X1),molecule(X2),complex(X3),
association_reaction(X1,X2,X3),
available(X1,T),
available(X2,T),time(T).

Σαν αποτέλεσμα της πιο πάνω δράσης είναι το σύνθετο μόριο να είναι διαθέσιμο την επόμενη χρονική στιγμή.

available(X3,T+1) :- associate(X1,X2,X3,T),molecule(X1),molecule(X2),

complex(X3),time(T).

Επίσης η κατάσταση των δυο αντιδρώντων μορίων αλλάζει.

change(X1,T):-associate(X1,X2,X3,T), molecule(X1), molecule(X2), complex(X3), time(T).

change(X2,T):-associate(X1,X2,X3,T), molecule(X1), molecule(X2), complex(X3), time(T).

Δράση 4: Αντίδραση

Για να γίνει αυτή η δράση γεγονός κάποια στιγμή πρέπει να υπάρχουν διαθέσιμα δυο μόρια τα οποία μπορούν να αντιδράσουν μεταξύ τους στη παρουσία καταλυτή παράγοντας ένα σύνθετο μόριο.

{associate_with_catalyze(X1, X2, X3, T)}:-molecule(X1), molecule(X2),
complex(X3), catalyzed_association_reaction(X1,X2,X3),
available(X1,T),
available(X2,T),
time(T).

Σαν αποτέλεσμα της πιο πάνω δράσης είναι το σύνθετο μόριο να είναι διαθέσιμο την επόμενη χρονική στιγμή.

available(X3,T+1):-
associate_with_catalyze(X1,X2,X3,T),molecule(X1),molecule(X2),complex(X3), time(T).

Επίσης η κατάσταση του πρώτου μορίου από τα αντιδρώντων μορίων αλλάζει.

change(X1,T):-
associate_with_catalyze(X1,X2,X3,T),molecule(X1),molecule(X2),complex(X3), time(T).

Δράση 5: Σύνθεση

Για να γίνει αυτή η δράση γεγονός κάποια στιγμή πρέπει να υπάρχει διαθέσιμο ένα μόρια το οποίο μπορεί να συνθέσει ένα άλλο μόριο.

{synthesize(X1,X2,T)}:-molecule(X1),molecule(X2),
synthesis_reaction(X1,X2),
available(X1,T),time(T).

Σαν αποτέλεσμα της πιο πάνω δράσης είναι το παραγόμενο μόριο να είναι διαθέσιμο την επόμενη χρονική στιγμή.

available(X2,T+1):-synthesize(X1,X2,T), molecule(X1),molecule(X2),time(T).

Γενικοί κανόνες:

Όλα παραμένουν ως έχουν την επόμενη χρονική στιγμή αν και μόνο εάν δεν υποστούν κάποια αλλαγή τη προηγουμένη στιγμή.

Ένα μόριο είναι διαθέσιμο την επόμενη χρονική στιγμή αν την προηγούμενη χρονική στιγμή ήταν και πάλι διαθέσιμο και δεν έχει υποστεί κάποια αλλαγή.

available(X,T+1) :- available(X,T), not change(X,T), molecule(X), time(T).

Το υπόστρωμα είναι το τρέχων υπόστρωμα για μια χρονική στιγμή αν και μόνο αν ίσχυε και τη προηγούμενη στιγμή και δεν έχει υποστεί κάποια αλλαγή.

num_subs(L,T+1) :- num_subs(L,T), not change_subs(L,T), level(L), time(T).

Από τη στιγμή που ένα απλό μόριο έχει επιλεγεί μια στιγμή τότε θεωρείτε επιλεγμένο όλες τις επόμενες χρονικές στιγμές.

chosen(X,T+1) :- chosen(X,T), simple(X), time(T).

Ένα μόριο είναι διαθέσιμο μια χρονική στιγμή τότε θεωρείται διαθέσιμο.

available_molecule(X):-available(X,T),molecule(X),time(T).

Δεν γίνεται να επιλέγονται ταυτόχρονα δυο διαφορετικά μόρια.

:-choose(X1,L1,L2,T),choose(X2,L1,L2,T),
simple(X1),simple(X2),level(L1),level(L2),time(T),X1!=X2.

Ένα μόρια αρχικοποιείται μια μόνο φορά.

:-initialize(X,T1),initialize(X,T2),simple(X),time(T1),time(T2),T1<T2.

Σκοπός του προγράμματος αυτού είναι να παραχθούν ένα από τα μόρια (prbp1p2_ap2 η pcaf_p300).

goal1:-available_molecule(prbp1p2_ap2).

goal1:-available_molecule(pcaf_p300).

Κεφάλαιο 6

Χρονικοί Περιορισμοί

6.1 Εισαγωγή

Ο προγραμματισμός (planning) είναι το πρόβλημα που βρίσκει μια σειρά δράσεων που ικανοποιούν κάποιο συγκεκριμένο σκοπό. Οι πιο πολλές έρευνες λογικού προγραμματισμού έχουν ασχοληθεί κυρίως με τις μεθοδολογίες και θέματα σχετικά με την ανάπτυξη αποδοτικών πλάνων. Έχουν αναπτυχθεί διάφορα συστήματα αποδοτικού προγραμματισμού που έχουν την ιδιότητα να ανακαλύπτουν ανεξάρτητες μεθόδους και να χρησιμοποιούν συγκεκριμένες γνώσεις. Τα συστήματα αυτά και η κατασκευή αποδοτικών δομών δεδομένων χρησιμοποιούνται στη υλοποίηση αλγορίθμων προγραμματισμού.

Ο λογικός προγραμματισμός έχει παίξει σημαντικό ρόλο στις έρευνες, παρέχοντας ορισμένα πλαίσια για κωδικοποίηση διάφορων ειδών γνώσεων και την αποδοτικότητα του κατά τη διάρκεια της επεξεργασίας προγραμματισμού.

Παρόλα αυτά, έχει σημειωθεί σχετικά περιορισμένη προσπάθεια για να αντιμετώπιση σημαντικών διαστάσεων για προβλήματα στο πραγματικό κόσμο, όπως για παράδειγμα ποιοτικό πλάνο και προτιμήσεις και χρονικοί περιορισμοί για πλάνα.

Σε πολλά πραγματικά πλαίσια, το διάστημα για εφικτά πλάνα για ικανοποίηση του στόχου είναι μεγάλο, αλλά πολλά τέτοια πλάνα μπορεί να παρουσιάζουν ανεπυθήμητα χαρακτηριστικά. Έτσι θα πρέπει να βρούμε λύση που να παράγει πλάνα που θεωρούνται ικανοποιητικά για τις προτιμήσεις του χρήστη.

Έτσι τα εφικτά πλάνα, μόνο αν τηρούν τις προτιμήσεις του χρήστη θα θεωρούνται αποδεκτά.

Χωρίζουμε τις προτιμήσεις όπου ο χρήστης θα μπορούσε να έχει σε διάφορες κατηγορίες:

- **Προτιμήσεις σχετικά με τη κατάσταση:** ο χρήστης προτιμά να βρίσκεται στη κατάσταση s που ικανοποιεί την ιδιότητα φ παρά να βρίσκεται στη κατάσταση s χωρίς να την ικανοποιεί, παρόλο που και οι δυο περιπτώσεις οδηγούν στην ικανοποίηση του στόχου του.
- **Προτιμήσεις σχετικά με δράση:** Ο χρήστης προτιμά να εκπληρώσει τη δράση α, όπου είναι εφικτή και επιτρέπει το σκοπό να πραγματοποιηθεί.
- **Προτιμήσεις σχετικά με κατεύθυνση:** ο χρήστης προτιμά την πορεία όπου ικανοποιεί την σίγουρη ιδιότητα ψ παρά αυτή που δεν την ικανοποιεί.
- **Πολυδιάστατες προτιμήσεις:** ο χρήστης προτιμά ένα σύνολο από προτιμήσεις με μια σειρά. Μια πορεία που ικανοποιεί την πιο θεμιτή προτίμηση της δίνεται μεγαλύτερη προτεραιότητα από μια άλλη όχι τόσο θεμιτή.

6.2 Χρονικοί περιορισμοί στο πρόβλημα δρομολόγησης φορτηγών

Πιο κάτω θα προσθέσουμε χρονικούς περιορισμούς και στο πρόβλημα δρομολόγησης φορτηγών.

Ας υποθέσουμε ότι στο κόσμο του προβλήματος δρομολόγησης φορτηγών υπάρχουν τρία πακέτα και ένα φορτηγό με δυο θέσεις. Τα πακέτα βρίκονται αρχικά σε μια περιοχή 2 και σκοπός του προβλήματος είναι τι πακέτο 1 να δρομολογηθέι στη περιοχή 3 και τα άλλα δυο πακέτα στη περιοχή 1.

Η κωδικοποίηση του πιο πάνω προβλήματος μέχρι στιγμής γίνεται ως εξής:

```
truck(truck1).

package(p1).
package(p2).
package(p3).

location(lo1).
location(lo2).
location(lo3).

truckarea(a1).
truckarea(a2).

% initial configuration
at(truck1,lo1,0).
free(a1,truck1,0).
free(a2,truck1,0).

at(p1,lo2,0).
at(p2,lo2,0).
at(p3,lo2,0).

connected(lo1,lo2).
connected(lo1,lo3).
connected(lo2,lo1).
connected(lo2,lo3).
connected(lo3,lo1).
connected(lo3,lo2).

goal :- at_destination(p1,lo3), at_destination(p2,lo1), at_destination(p3,lo1).
compute 1 {goal}.
```

Οι χρονικοί περιορισμοί που θέλω να θέσω είναι ότι το πακέτο 1 επιθυμώ να παραδοθεί μεταξύ της χρονικής στιγμής 2 και 8 και επιπλέων το πακέτο 3 να παραδοθεί μεταξύ της χρονικής στιγμής 4 και 8. Η κωδικοποίηση του πιο πάνω χρονικού περιορισμού γίνεται ως εξής:

```
  time_constraint(p1) :- package(p1), location(lo3), delivered(p1,lo3,T), time(T), T>=2,
T<=8.
  time_constraint(p3) :-    package(p3), location(lo1), delivered(p3,lo1,T), time(T), T>=4,
T<=8.
  goal :- at_destination(p1,lo3), time_constraint (p1),
            at_destination(p2,lo1),
            at_destination(p3,lo1), time_constraint (p3).

  compute 1 {goal}.
```

Κεφάλαιο 7

Πειραματικά Αποτελέσματα

7.1 Εισαγωγή

Μετά την υλοποίηση των πιο πάνω προγραμμάτων σε λογικό προγραμματισμό ακολούθησε το πειραματικό μέρος. Δημιούργησα τα αρχεία trucks.lp, openstacks.lp, pathway.lp όπου περιέχουν του κανόνες που επεξήγησα αναλυτικά στη προηγουμένη ενότητα. Επίσης πρόσθεσα το παράγοντα χρόνο (time(0..length)) σε αυτά τα αρχεία.
Για κάθε ένα πρόβλημα δημιούργησα αλλά πέντε αρχεία όπου περιέχουν συγκεκριμένα παραδείγματα, δηλαδή τα αντικείμενα του προβλήματος και την αρχική κατάσταση τους.
Στη συνεχεία ακολούθησε το πειραματικοί στάδιο όπου για κάθε ένα παράδειγμα έπαιρνα διαφορετικά αποτελέσματα για διαφορετικούς χρόνους. Χρησιμοποίησα τα προγράμματα SMODELS και CLASP για να μου δώσει τις λύσεις και στη συνέχεια θα συγκρίνω το χρόνο που θα πάρουν τα δυο προγράμματα για να τρέξουν.
Το CLASP παίζει ακριβώς το ρολό του SMODELS. Δηλαδή βρίσκει τις λύσεις που ικανοποιούν του κανόνες μετά την κωδικοποίηση του LPARSE.
Σε κάθε δοκιμή αύξανα κατά ένα τον χρόνο κ σταματούσα όταν έβρισκα τον πρώτο χρόνο που μου έδινε λύση.
Χρησιμοποίησα τις πιο κάτω εντολές που καθόριζα το χρόνο.
lparse -c length=6 pathway_p02.lp p02.lp | smodels 1 για το smodels και
lparse -c length=6 pathway_p02.lp p02.lp | clasp 1 για το clasp.

7.2 Πειραματικά αποτελέσματα για πρόβλημα δρομολόγησης φορτηγών

Πείραμα 1.1:

Τα αντικείμενα στο παράδειγμα αυτό είναι ένα φορτηγό και τρία πακέτα καθώς επίσης και τρις περιοχές οι οποίες είναι ενωμένες μεταξύ τους. Κάθε φορτηγό έχει τρις θέσεις οι οποίες αρχικά είναι άδειες. Το φορτηγό είναι στη πρώτη τοποθεσία αρχικά ενώ και τα τρία πακέτα βρίσκονται στη δεύτερη τοποθεσία. Σκοπός του παραδείγματος αυτού είναι να υπολογίσει τις απαντήσεις που έχουν σαν αποτέλεσμα το πρώτο πακέτο να βρίσκεται στη τρίτη περιοχή και το δεύτερο και τρίτο πακέτο στη πρώτη περιοχή. Πιο κάτω φαίνονται οι χρόνοι που πήρα και με τα δυο προγράμματα.

trucks-p01		
time	SMODELS(sec)	CLASP
4	0,119	0,010
5	0,163	0,020
6	0,261	0,040
7	0,464	0,100
8	0,950	0,190
9	0,419	0,050

trucks-p01	SMODELS	CLASP
average time	0,396	0,068

Πείραμα 1.2:

Τα αντικείμενα στο παράδειγμα αυτού είναι ένα φορτηγό και τέσσερα πακέτα καθώς επίσης και τρις περιοχές οι οποίες είναι ενωμένες μεταξύ τους. Κάθε φορτηγό έχει τρις θέσεις οι οποίες αρχικά είναι άδειες. Το φορτηγό είναι στη δεύτερη τοποθεσία αρχικά ενώ και τα τέσσερα πακέτα βρίσκονται στη πρώτη τοποθεσία. Σκοπός του παραδείγματος αυτού είναι να υπολογίσει τις απαντήσεις που έχουν σαν αποτέλεσμα το πρώτα τρία πακέτα να βρίσκεται στη δεύτερη περιοχή και το τέταρτο πακέτο στη τρίτη περιοχή. Πιο κάτω φαίνονται οι χρόνοι που πήρα και με τα δυο προγράμματα.

trucks-p02		
time	SMODELS(sec)	CLASP
4	0,191	0,020
5	0,257	0,030
6	0,419	0,050
7	0,718	0,120
8	1,544	0,350
9	4,591	0,450
10	23,615	1,020
11	90,517	2,030
12	1,275	0,680

trucks-p02	SMODELS	CLASP
average time	13,681	0,528

Πείραμα 1.3:

Τα αντικείμενα στο παράδειγμα αυτού είναι ένα φορτηγό και πέντε πακέτα καθώς επίσης και τρις περιοχές οι οποίες είναι ενωμένες μεταξύ τους. Κάθε φορτηγό έχει τρις θέσεις οι οποίες αρχικά είναι άδειες. Το φορτηγό είναι στη δεύτερη τοποθεσία αρχικά ενώ και τα τέσσερα πρώτα πακέτα βρίσκονται στη πρώτη τοποθεσία και το πέμπτο στη δεύτερη. Σκοπός του παραδείγματος αυτού είναι να υπολογίσει τις απαντήσεις που έχουν σαν αποτέλεσμα τα πρώτα δυο πακέτα να βρίσκεται στη δεύτερη περιοχή και τα υπόλοιπα στη τρίτη περιοχή.

trucks-p03		
time	SMODELS(sec)	CLASP
4	0,272	0,020
5	0,437	0,030
6	0,865	0,050
7	1,771	0,120
8	5,149	0,350
9	25,765	0,450
10	226,141	1,020
11	16014,453	2,030
12	22351,901	0,680
13	45610,559	0,230
14	213,189	0,110

trucks-p03	SMODELS	CLASP
average time	7677,318	0,463

Πείραμα 1.4:

Τα αντικείμενα στο παράδειγμα αυτού είναι ένα φορτηγό και έξι πακέτα καθώς επίσης και τρις περιοχές οι οποίες είναι ενωμένες μεταξύ τους. Κάθε φορτηγό έχει τρις θέσεις οι οποίες αρχικά είναι άδειες. Το φορτηγό είναι στη τρίτη τοποθεσία αρχικά ενώ και τα έξι πακέτα βρίσκονται στη πρώτη τοποθεσία. Σκοπός του παραδείγματος αυτού είναι να υπολογίσει τις απαντήσεις που έχουν σαν αποτέλεσμα το πρώτο, δεύτερο, πέμπτο και έκτο πακέτο να βρίσκεται στη τρίτη περιοχή και το τρίτο και τεταρτο πακέτο στη δεύτερη περιοχή.

trucks-p04		
time	SMODELS(sec)	CLASP
4	0,378	0,040
5	0,515	0,050
6	0,801	0,120
7	1,450	0,270
8	4,085	0,780
9	11,096	1,710
10	6,525	2,120
11	491,580	4,990
12	3166,580	11,960
13	45045,335	18,550
14	78302,452	51,810
15	82350,785	226,900
16	112152,213	351,920
17	18119,892	11,060

trucks-p04	SMODELS	CLASP
average time	24260,978	48,734

Πείραμα 1.5:

Τα αντικείμενα στο παράδειγμα αυτού είναι δυο φορτηγά και έξι πακέτα καθώς επίσης και τρις περιοχές οι οποίες είναι ενωμένες μεταξύ τους. Κάθε φορτηγό έχει τρις θέσεις οι οποίες αρχικά είναι άδειες. Τα φορτηγά είναι στη τρίτη τοποθεσία αρχικά ενώ και τα έξι πακέτα βρίσκονται στη δεύτερη τοποθεσία. Σκοπός του παραδείγματος αυτού είναι να υπολογίσει τις απαντήσεις που έχουν σαν αποτέλεσμα το πρώτο και το δεύτερο πακέτο να βρίσκεται στη τρίτη περιοχή και τα υπόλοιπα στη πρώτη περιοχή.

trucks-p05		
time	SMODELS(sec)	CLASP
4	2,020	0,250
5	29,100	1,100
6	610,640	5,950
7	11619,700	27,960
8	18325,544	144,680
9	25325,787	2,080

trucks-p05	SMODELS	CLASP
average time	9318,799	30,337

Πείραμα 1.6:

Πιο κάτω φαίνονται οι χρόνοι που πήρα και με τα δυο προγράμματα με το πρόβλημα που περίγραψα στο κεφάλαιο 6 όσον αφορά τους χρονικούς περιορισμούς.

trucks		
time	SMODELS	CLASP
4	0.113	0.010
5	0.136	0.020
6	0.222	0.040
7	0.432	0.100
8	0.804	0.150
9	0.322	0.080
10	0.517	0.040

7.3 Πειραματικά αποτελέσματα για πρόβλημα ανοικτής στοίβας εργασιών

Πείραμα 2.1:

Τα αντικείμενα στο παράδειγμα αυτού είναι πέντε προϊόντα και πέντε παραγγελίες καθώς επίσης και πέντε θέσεις στη στοίβα. Η μηχανή την αρχική στιγμή είναι διαθέσιμη και η διαθέσιμη θέση στη στοίβα είναι η πρώτη εφόσον δεν έχει γίνει καμιά εργασία ακόμα. Οι πέντε παραγγελίες βρίσκονται σε αναμονή. Η κάθε παραγγελία περιέχει από δυο προϊόντα. Σκοπός του παραδείγματος είναι τουλάχιστον η παράδοση της πρώτης παραγγελίας.

openstacks-p01		
time	SMODELS	CLASP
4	2,310	0,400
5	2,901	0,570
6	8,065	0,670

openstacks-p01	SMODELS	CLASP
average-time	4,425	0,547

Πείραμα 2.2:

Τα αντικείμενα στο παράδειγμα αυτού είναι πέντε προϊόντα και πέντε παραγγελίες καθώς επίσης και πέντε θέσεις στη στοίβα. Η μηχανή την αρχική στιγμή είναι διαθέσιμη και η διαθέσιμη θέση στη στοίβα είναι η πρώτη εφόσον δεν έχει γίνει καμιά εργασία ακόμα. Οι πέντε παραγγελίες βρίσκονται σε αναμονή. Η κάθε παραγγελία περιέχει από δυο προϊόντα. Σκοπός του παραδείγματος είναι τουλάχιστον η παράδοση των πρώτων δυο παραγγελιών.

openstacks-p02		
time	SMODELS	CLASP
4	2,379	0,410
5	2,956	0,580
6	6,681	0,690
7	143,338	0,740
8	42760,720	1,860
9	56215,120	5,450

10	58145,578	12,870
11	-	62,320
12	48456,152	9,150

openstacks-p02	SMODELS	CLASP
average-time	25716,616	10,452

Πείραμα 2.3:

Τα αντικείμενα στο παράδειγμα αυτού είναι πέντε προϊόντα και πέντε παραγγελίες καθώς επίσης και πέντε θέσεις στη στοίβα. Η μηχανή την αρχική στιγμή είναι διαθέσιμη και η διαθέσιμη θέση στη στοίβα είναι η πρώτη εφόσον δεν έχει γίνει καμιά εργασία ακόμα. Οι πέντε παραγγελίες βρίσκονται σε αναμονή. Η κάθε παραγγελία περιέχει από δυο προϊόντα. Σκοπός του παραδείγματος είναι τουλάχιστον η παράδοση των πρώτων τριών ο παραγγελιών.

openstacks-p03		
time	SMODELS	CLASP
4	2,309	0,400
5	2,844	0,580
6	5,653	0,720
7	36,747	1,000
8	8066,726	1,500
9	21322,459	4,490
10	45745,356	13,010
11	80745,877	50,670
12	200325,46	161,520
13	-	535,340
14	-	1646,830
15	-	4294967078,249
16	81988,351	1121,645

openstacks-p03	SMODELS	CLASP
average-time	43824,178	294,809

Πείραμα 2.4:

Τα αντικείμενα στο παράδειγμα αυτού είναι πέντε προϊόντα και πέντε παραγγελίες καθώς επίσης και πέντε θέσεις στη στοίβα. Η μηχανή την αρχική στιγμή είναι διαθέσιμη και η διαθέσιμη θέση στη στοίβα είναι η πρώτη εφόσον δεν έχει γίνει καμιά εργασία ακόμα. Οι πέντε παραγγελίες βρίσκονται σε αναμονή. Η κάθε παραγγελία περιέχει από δυο προϊόντα. Σκοπός του παραδείγματος είναι τουλάχιστον η παράδοση των πρώτων τεσσάρων παραγγελιών.

openstacks-p04		
time	SMODELS	CLASP
4	2,381	0,400
5	2,943	0,540
6	4,827	0,750
7	17,474	0,920
8	4952,145	1,480
9	9875,788	3,580
10	40452,154	10,410
11	142445,565	35,640
12	288122,451	106,130
13	-	420,620
14	-	650,120
15	-	754,450
16	-	654,120
17	-	523,140
18	-	186,885
19	79766,475	938,340

openstacks-p04	SMODELS	CLASP
average-time	56564,2203	267,970

Πείραμα 2.5:

Τα αντικείμενα στο παράδειγμα αυτού είναι πέντε προϊόντα και πέντε παραγγελίες καθώς επίσης και πέντε θέσεις στη στοίβα. Η μηχανή την αρχική στιγμή είναι διαθέσιμη και η διαθέσιμη θέση στη στοίβα είναι η πρώτη εφόσον δεν έχει γίνει καμιά εργασία ακόμα. Οι πέντε παραγγελίες βρίσκονται σε αναμονή. Η κάθε παραγγελία περιέχει από δυο προϊόντα. Σκοπός του παραδείγματος είναι η παράδοση όλων των παραγγελιών.

openstacks-p05		
time	SMODELS	CLASP
4	2,381	0,410
5	2,943	0,530
6	4,827	0,670
7	17,474	0,920
8	5065,861	1,350
9	11032,124	4,160
10	45244,145	9,260
11	148245,475	32,730
12	291542,176	113,100
13	-	312,430
14	-	1549,920
15	-	1645,210
16	-	1932,780
17	-	2125,450
18	-	3421,870
19	-	5032,680
20	-	6035,680
21	-	6931,450
22	81475,746	54,240

openstacks-p05	SMODELS	CLASP
average-time	58263,3152	1537,097

7.4 Πειραματικά αποτελέσματα για πρόβλημα βιολογικών μονοπατιών

Στα πιο κάτω πειράματα δημιούργησα παραδείγματα όπου σκοπό είχαν να παράγονται συγκεκριμένες χημικές ενώσεις. Στο πρώτο πείραμα να παράγεται τουλάχιστον μια χημική ένωση, στο δεύτερο τουλάχιστον δυο χημικές ενώσεις, στο τρίτο τουλάχιστον τρις, στο τέταρτο τουλάχιστον τέσσερις χημικές ενώσεις και στο πέμπτο να παράγονται τουλάχιστον πέντε χημικές ενώσεις.

Πείραμα 3.1:

pathway-p01		
time	SMODELS	CLASP
3	4,483	0,610
4	5,663	0,790
5	6,927	0,960

pathway-p01	SMODELS	CLASP
average-time	5,691	0,787

Πείραμα 3.2:

pathway-p02		
time	SMODELS	CLASP
3	16,118	2,330
4	20,238	3,020
5	24,36	3,630
6	28,846	4,400

pathway-p02	SMODELS	CLASP
average-time	22,3905	3,345

Πείραμα 3.3:

pathway-p03		
time	SMODELS	CLASP
3		11,050
4	82,362	14,330
5	96,146	17,720
6	112,699	20,950
7	129,525	25,060
8	149,849	28,360
9	165,793	32,190
10	181,053	36,570
11	202,972	50,310
12	246,013	48,160
13	atom name too long or end of file	94,200
14		71,330

pathway-p03	SMODELS	CLASP
average-time	142,3566	34,445

Πείραμα 3.4 και Πείραμα 3.5:

Δεν πήραμε κανένα αποτέλεσμα αφού το lparse δεν κατάφερε να διαβάσει τα αρχεία που του δώσαμε.

Κεφάλαιο 8

Συμπεράσματα

8.1 Εισαγωγή

Στη Διπλωματική αυτή Εργασία μελετήσαμε την γλωσσά προγραμματισμού δράσης και την γλωσσά προγραμματισμού συνόλου απαντήσεων. Στην πράξη, αφού μελέτησα και τις δυο γλώσσες λογικού προγραμματισμού επέλεξα τρία συγκεκριμένα προβλήματα και τα κωδικοποίησα στη γλωσσά προγραμματισμού συνόλου απαντήσεων και στη συνεχεία προέθεσα προτιμήσεις.

8.2 Γενικά Συμπεράσματα

Παρόλο που με τη γλωσσά προγραμματισμού δράσεων ασχολήθηκα μονό στη θεωρία, μπορώ να πω ότι είναι μια απλή γλωσσά. Οι δράσεις λειτουργούν ανεξάρτητα η μια από την άλλη, χωρίς δηλαδή να υπάρχουν συγκρούσεις μεταξύ του. Δεν χρειάζεται να οριστούν περιορισμοί εκτός από αυτοί που ήδη υπάρχουν στη δήλωση μιας δράση. Σαν αποτέλεσμα δίνει ένα πλάνο το οποίο αποτελείται από μια σειρά δράσεων που εξελίσσονται κατά το πέρασμα του χρόνου.

Η γλωσσά προγραμματισμού συνόλου απαντήσεων είναι πολύ απλή συντακτικά γλωσσά στην οποία πρέπει να δηλώνεται το καθετί. Με την κωδικοποίηση των δράσεων στη γλωσσά προγραμματισμού συνόλου απαντήσεων δεν συνεπάγεται ότι οι δράσεις

λειτουργούν ανεξάρτητα σε σχέση με το χρόνο. Τα αποτελέσματα μια δράσης ισχύουν την αμέσως επόμενη χρονική στιγμή. Αν μια κατάσταση δεν έχει αλλάξει μια χρονική στιγμή, τότε παραμένει στη ιδία κατάσταση. Ανάλογα με τα αποτελέσματα μια δράσεις μπορούμε να κρίνουμε αν μπορεί να γίνεται ταυτόχρονα με μια άλλη δράση. Έτσι στο προγραμματισμό συνόλου απαντήσεων χρειάζεται να έχουμε περιορισμούς για να πάρουμε τη σωστές λύσεις. Επίσης υπάρχει η δυνατότητα να ορίσουμε προτιμήσεις και καθώς επίσης και χρονικούς περιορισμούς στα προβλήματα μας ούτως ώστε να πάρουμε πιο επιθυμητά αποτέλεσμα

Το σύστημα Lparse και Smodels μας δίνουν τη δυνατότητα να σπαράξουμε τις λύσεις κάποιου προβλήματος. Το Lparse σαν αναλυτής του προβλήματος, όπως έχουμε πει, διαβάζει το πρόβλημα και το μετατρέπει σε μια γλώσσα κατανοητή για το Smodels. Στην ουσία δημιουργεί λογικά δέντρα με όλες τους πιθανούς συνδυασμούς συμφωνά με το πρόβλημα που του δίνουμε. Σχετικά λειτουργεί γρήγορα, παρόλο που στα μεγάλα προβλήματα που υπάρχουν πολλές πιθανές λύσεις, δηλαδή τα λογικά δέντρα είναι πολλά, καθυστερεί. Υπάρχει και η πιθανότητα να μην καταφέρει να δημιουργήσει αυτά τα δέντρα αν είναι πολύ μεγάλα. Αυτό συνέβηκε στα πιο πολύπλοκα προβλήματα που πειραματιστήκαμε, κυρίως στο πρόβλημα μονοπατιών όπου οι πιθανοί συνδυασμοί ήταν πολλοί και λόγω μνήμης δεν κατάφερε να δημιουργήσει τα δέντρα..

Το Smodel έχοντας ως σκοπό να αποδεκτή τις σωστές λύσεις που του έδωσε ο αναλυτής Lparse σχετικά δεν λειτουργεί καθόλου γρήγορα. Ακόμα και σε μικρές τιμές του length καθυστερούσε πολύ με αποτέλεσμα πολλές φορές αν και το δίναμε πολλή χρόνο, να μην καταφέρει να δώσει λύση. Αντίθετα το αντίστοιχο εργαλείο Clasp λειτουργούσε πολύ πιο γρήγορα. Μπορούσε να καταλάβει αν θα έδινε λύσεις η όχι.

Τέλος, πιστεύω ότι το πιο πάνω σύστημα δεν προσφέρεται για λύση πιο δύσκολων προβλημάτων παρόλο που η υλοποίηση ενός προγράμματος σε λογικό προγραμματισμό συνόλου απαντήσεων είναι εύκολη.

Βιβλιογραφία:

[1.] Εγχειρίδιο χρήσης Lparse και Smodels
htpp://www.tcs.hut.fi/Software/Smodels

[2.] Εγχειριδιο χρησης Clasp
http://www.cs.uni-potsdam.de/clasp/?page=downloads

[3.] CS 395T, spring 2005 Answer Set Programming

[4.] Vladimir. Lifschitz: 'Introduction to Answer Set Programming'.

[5.] Ilkka Niemela:
Answer – Set Programming: a Declarative Knowledge Representation Paradigm
Helsinky University of Technology

[6.] Esra Erdem, Theory and Applications of Answer Set Programming,
The University of Texas at Austin.

[7.] The Benchmark Domains of the Deterministic Part of IPC-5
http://ipc5.ing.unibs.it

[8.] Εγχειριδιο χρησης PDDL – The Planning Domain Definition Language, Version 1.2

[9.] Tao Cao Son, Enrico Pontelli:
Planning with Preferences unsing Logic Programming
Department of Computer Science – New Mexico State University

ПАРАРТНМА

%trucks.lp

```
% time
time(0..length).


%actions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


{load(Pa,Tr,L,a1,T)} :- at(Tr,L,T),
                         at(Pa,L,T),
                         free(a1,Tr,T),%not free(a2,Tr,T),not
free(a3,Tr,T),
                         package(Pa),
                         truck(Tr),location(L),truckarea(a1),time(T).

{load(Pa,Tr,L,a2,T)} :- at(Tr,L,T),
                         at(Pa,L,T),
                         free(a1,Tr,T),free(a2,Tr,T),%not free(a3,Tr,T),

package(Pa),truck(Tr),location(L),truckarea(a1),truckarea(a2),time(T).


{load(Pa,Tr,L,a3,T)} :- at(Tr,L,T),
                         at(Pa,L,T),
                         free(a1,Tr,T),free(a2,Tr,T),free(a3,Tr,T),

package(Pa),truck(Tr),location(L),truckarea(a1),truckarea(a2),truckarea
(a3),time(T).

%effects
in(Pa,Tr,Ta,T+1):- load(Pa,Tr,L,Ta,T)
,package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

change_free_ta(Ta,Tr,T):- load(Pa,Tr,L,Ta,T)
,package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

change(Pa,T):- load(Pa,Tr,L,Ta,T)
,package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

at(Tr,L,T+1):- load(Pa,Tr,L,Ta,T)
,package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


{unload(Pa,Tr,L,a1,T)}:-in(Pa,Tr,a1,T),
                        at(Tr,L,T),

package(Pa),truck(Tr),location(L),truckarea(a1),time(T).

{unload(Pa,Tr,L,a2,T)}:-in(Pa,Tr,a2,T),
                        at(Tr,L,T),
                        free(Tr,a1,T),
```

```
package(Pa),truck(Tr),location(L),truckarea(a1),truckarea(a2),time(T).

{unload(Pa,Tr,L,a3,T)}:-in(Pa,Tr,a3,T),
                        at(Tr,L,T),
                        free(a1,Tr,T),free(a2,Tr,T),

package(Pa),truck(Tr),location(L),truckarea(a1),truckarea(a2),truckarea
(a3),time(T).

%effects

free(Ta,Tr,T+1):-  unload(Pa,Tr,L,Ta,T),
package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).


at(Tr,L,T+1) :- unload(Pa,Tr,L,Ta,T),
package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

change(Pa,T):- unload(Pa,Tr,L,Ta,T)
,package(Pa),truck(Tr),location(L),truckarea(Ta),time(T).

at_destination(Pa,L) :-
unload(Pa,Tr,L,Ta,T),package(Pa),truck(Tr),location(L),truckarea(Ta),ti
me(T).

delivered(Pa,L,T+1) :-
unload(Pa,Tr,L,Ta,T),package(Pa),truck(Tr),location(L),truckarea(Ta),ti
me(T).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%drive from L1 to L2
%drive(Truck,Location1,Location2,time)

{drive(Tr,L1,L2,T)}:-at(Tr,L1,T),
                     connected(L1,L2),
                     truck(Tr),
                     location(L1),location(L2),L1!=L2,
                     time(T).


%effects

at(Tr,L2,T+1) :-
drive(Tr,L1,L2,T),truck(Tr),location(L1),location(L2),time(T).

change(Tr,T):- drive(Tr,L1,L2,T)
,truck(Tr),location(L1),location(L2),time(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% frame Axioms
```

```
at(Pa,L,T+1) :-at(Pa,L,T),not
change(Pa,T),package(Pa),location(L),time(T).

in(Pa,Tr,Ta,T+1):-in(Pa,Tr,Ta,T),not
change(Pa,T),package(Pa),truck(Tr),truckarea(Ta),time(T).

at(Tr,L,T+1) :-at(Tr,L,T),not
change(Tr,T),truck(Tr),location(L),time(T).

free(Ta,Tr,T+1):- free(Ta,Tr,T),not
change_free_ta(Ta,Tr,T),truck(Tr),truckarea(Ta),time(T).

intruck(Pa,Tr,T):-in(Pa,Tr,Ta,T), package(Pa), truck(Tr),
truckarea(Ta), time(T).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-load(Pa1,Tr,L,Ta1,T),load(Pa2,Tr,L,Ta2,T),
package(Pa1),package(Pa2),Pa1!=Pa2,location(L),truck(Tr),truckarea(Ta1)
,truckarea(Ta2),Ta1!=Ta2,time(T).

:-load(Pa1,Tr,L,Ta,T),load(Pa2,Tr,L,Ta,T),
package(Pa1),package(Pa2),Pa1!=Pa2,location(L),truck(Tr),truckarea(Ta),
time(T).

:-load(Pa,Tr,L,Ta1,T),load(Pa,Tr,L,Ta2,T),
package(Pa1),package(Pa),location(L),truck(Tr),truckarea(Ta1),truckarea
(Ta2),time(T),Ta1!=Ta2.

:-drive(Tr,L1,L2,T),drive(Tr,L1,L3,T),
  location(L1),location(L2),location(L3),truck(Tr),time(T),L2!=L3.

:-drive(Tr,L1,L2,T),load(Pa,Tr,L1,Ta,T),
package(Pa),location(L1),location(L2),truckarea(Ta),truck(Tr),time(T),L
1!=L2.

:-drive(Tr,L1,L2,T),unload(Pa,Tr,L1,Ta,T),
package(Pa),location(L1),location(L2),truckarea(Ta),truck(Tr),time(T),L
1!=L2.

:-load(Pa1,Tr,L,Ta1,T),unload(Pa2,Tr,L,Ta2,T),

package(Pa1),package(Pa2),truck(Tr),location(L),truckarea(Ta1),truckare
a(Ta2),Ta1!=Ta2,time(T),Pa1!=Pa2.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


hide connected(X,Y).
hide location(X).
hide in(P,Tr,Ta,T).
hide intruck(P,Tr,T).
hide at(P,L,T).
hide time(T).
```

```
hide package(P).
hide truck(T).
hide free(Ta,Tr,T).
hide change(P,T).
hide truckarea(Ta).
hide change_free_ta(Ta,Tr,T).
```

%openstacks.lp

```
time(0..length).


%%%%%%%%%%%%%%
%%% ACTIONS %%%
%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%action setup machine
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
{setup_machine(Prod,Avail,T)}:- machine_available(T),
                                not made(Prod,T),
                                stacks_avail(Avail,T),
                                product(Prod),count(Avail),time(T).

%%% EFFECTS

machine_configured(Prod,T+1):-
setup_machine(Prod,Avail,T),product(Prod),count(Avail),time(T).


delete_mach_available(T):-
setup_machine(Prod,Avail,T),product(Prod),count(Avail),time(T).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%action make product


{make_product(Prod,Avail,o1,T)} :- machine_configured(Prod,T),
                                   stacks_avail(Avail,T),
                                   includes(o1,Prod),started(o1,T),
                                   product(Prod),count(Avail),time(T).

{make_product(Prod,Avail,o2,T)} :- machine_configured(Prod,T),
                                   stacks_avail(Avail,T),
                                   includes(o2,Prod),started(o2,T),
                                   product(Prod),count(Avail),time(T).

{make_product(Prod,Avail,o3,T)} :- machine_configured(Prod,T),
                                   stacks_avail(Avail,T),
                                   includes(o3,Prod),started(o3,T),
                                   product(Prod),count(Avail),time(T).


{make_product(Prod,Avail,o4,T)} :- machine_configured(Prod,T),
                                   stacks_avail(Avail,T),
                                   includes(o4,Prod),started(o4,T),
                                   product(Prod),count(Avail),time(T).

{make_product(Prod,Avail,o5,T)} :- machine_configured(Prod,T),
```

```
                                    stacks_avail(Avail,T),
                                    includes(o5,Prod),started(o5,T),
                                    product(Prod),count(Avail),time(T).


%effects

machine_available(T+1) :-
make_product(Prod,Avail,Ord,T),product(Prod),count(Avail),order(Ord),ti
me(T).

made(Prod,T+1):-
make_product(Prod,Avail,Ord,T),product(Prod),count(Avail),order(Ord),ti
me(T).

delete_mach_config(Prod,T):-
make_product(Prod,Avail,Ord,T),product(Prod),count(Avail),order(Ord),ti
me(T).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%action start order
{start_order(Ord,Avail,NewAvail,T)}:- waiting(Ord,T),

stacks_avail(Avail,T),next_count(NewAvail,Avail),

order(Ord),count(Avail),count(NewAvail),time(T).

%effects

started(Ord,T+1):-
start_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvai
l),time(T).

stacks_avail(NewAvail,T+1):-
start_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvai
l),time(T).

delete_stacks_avail(Avail,T):-
start_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvai
l),time(T).

delete_waiting(Ord,T):-
start_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvai
l),time(T).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%ship order
```

```
{ship_order(o1,Avail,NewAvail,T)}:-
order(o1),started(o1,T),product(p1),product(p2),includes(o1,p1),include
s(o1,p2),

made(p1,T),made(p2,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).

{ship_order(o2,Avail,NewAvail,T)}:-
order(o2),started(o2,T),product(p3),product(p4),includes(o2,p3),include
s(o2,p4),

made(p3,T),made(p4,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).

{ship_order(o3,Avail,NewAvail,T)}:-
order(o3),started(o3,T),product(p1),product(p5),includes(o3,p1),include
s(o3,p5),

made(p1,T),made(p5,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).

{ship_order(o4,Avail,NewAvail,T)}:-
order(o4),started(o4,T),product(p3),product(p5),includes(o4,p3),include
s(o4,p5),

made(p3,T),made(p5,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).

{ship_order(o5,Avail,NewAvail,T)}:-
order(o5),started(o5,T),product(p2),product(p4),includes(o5,p2),include
s(o5,p4),

made(p2,T),made(p4,T),stacks_avail(Avail,T),

next_count(Avail,NewAvail),count(Avail),count(NewAvail),time(T).


%effects
shipped(Ord):-
ship_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail
),time(T).

stacks_avail(NewAvail,T+1):-
ship_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail
),time(T).

delete_stacks_avail(Avail,T):-
ship_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail
),time(T).

delete_started(Ord,T):-
ship_order(Ord,Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail
),time(T).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%action open new stact

{open_new_stack(Open,NewOpen,T)}:-
stacks_avail(Open,T),next_count(Open,NewOpen),
                                   count(Open),count(NewOpen),time(T).

stacks_avail(NewOpen,T+1) :-
open_new_stack(Open,NewOpen,T),count(Open),count(NewOpen),time(T).
delete_stacks_avail(Open,T):-
open_new_stack(Open,NewOpen,T),count(Open),count(NewOpen),time(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%everything stays at the same state

waiting(Ord,T+1):-waiting(Ord,T),not
delete_waiting(Ord,T),order(Ord),time(T).

stacks_avail(Avail,T+1):-stacks_avail(Avail,T),not
delete_stacks_avail(Avail,T),count(Avail),time(T).

machine_available(T+1):-machine_available(T),not
delete_mach_available(T),time(T).

machine_configured(Prod,T+1):-machine_configured(Prod,T),not
delete_mach_config(Prod,T),product(Prod),time(T).

started(Ord,T+1):-started(Ord,T),not
delete_started(Ord,T),order(Ord),time(T).

made(Prod,T+1):-made(Prod,T),product(Prod),time(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-setup_machine(Prod,Avail,T),
  start_order(Ord,Avail,NewAvail,T),
  product(Prod),order(Ord),count(Avail),count(NewAvail),time(T).

:-setup_machine(Prod,Avail,T),
  ship_order(Ord,Avail,NewAvail,T),
  product(Prod),order(Ord),count(Avail),count(NewAvail),time(T).

:-setup_machine(Prod,Avail,T),
  open_new_stack(Avail,NewOpen,T),
  product(Prod),count(Avail),count(NewOpen),time(T).

:-make_product(Prod,Avail,Ord1,T),
```

```
  start_order(Ord2,Avail,NewAvail,T),

product(Prod),order(Ord1),order(Ord2),count(Avail),count(NewAvail),time
(T),Ord1!=Ord2.

:-make_product(Prod,Avail,Ord1,T),
  ship_order(Ord2,Avail,NewAvail,T),

product(Prod),order(Ord1),order(Ord2),count(Avail),count(NewAvail),time
(T),Ord1!=Ord2.

:-make_product(Prod,Avail,Ord1,T),
  open_new_stack(Avail,NewOpen,T),
  product(Prod),order(Ord1),count(Avail),count(NewOpen),time(T).

:-start_order(Ord1,Avail,NewAvail,T),
  ship_order(Ord2,Avail,NewAvail_1,T),
  order(Ord1),order(Ord2),Ord1!=Ord2,

count(Avail),count(NewAvail),count(NewAvail_1),NewAvail!=NewAvail_1,tim
e(T).

:-start_order(Ord,Avail,NewAvail,T),
  open_new_stack(Avail,NewOpen,T),

order(Ord),count(Avail),count(NewAvail),count(NewOpen),NewAvail!=NewOpe
n,time(T).

:-ship_order(Ord,Avail,NewAvail,T),

open_new_stack(Avail,NewAvail,T),order(Ord),count(Avail),count(NewAvail
),time(T).


:-
start_order(Ord1,Avail,NewAvail,T),start_order(Ord2,Avail,NewAvail,T),

count(Avail),count(NewAvail),order(Ord1),order(Ord2),Ord1!=Ord2,time(T)
.

:-setup_machine(Prod1,Avail,T),setup_machine(Prod2,Avail,T),
  product(Prod1),product(Prod2),count(Avail),time(T),Prod1!=Prod2.

:-ship_order(Ord1,Avail,NewAvail,T),ship_order(Ord2,Avail,NewAvail,T),

count(Avail),count(NewAvail),order(Ord1),order(Ord2),time(T),Ord1!=Ord2
.


hide product(Prod).
hide order(Ord).
hide time(T).
hide count(Avail).
hide next_count(Avail1,Avail2).
hide delete_stacks_avail(Avail,T).
hide delete_waiting(Ord,T).
hide delete_mach_available(T).
```

```
hide delete_mach_config(Prod,T).
hide machine_available(T).
hide machine_configured(Prod,T).
hide stacks_avail(Count,T).
hide includes(O,P).
hide waiting(O,T).
hide started(O,T).
hide delete_started(O,T).
hide made(P,T).
```

pathways.lp

```
time(0..length).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%

molecule(X):- simple(X).

molecule(X):- complex(X).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%


%%%%%%%%%%%%%
%% ACTIONS %%
%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%

%% CHOOSE

{choose(X,L1,L2,T)}:-simple(X),level(L1),level(L2),possible(X),
                     not chosen(X,T), num_subs(L2,T),
next(L1,L2),time(T).

%% EFFECTS CHOOSE

chosen(X,T+1):-choose(X,L1,L2,T),simple(X),level(L1),level(L2),time(T).

num_subs(L1,T+1):-
choose(X,L1,L2,T),simple(X),level(L1),level(L2),time(T).

change_subs(L2,T):-
choose(X,L1,L2,T),simple(X),level(L1),level(L2),time(T).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%

%% INITIALIZE

{initialize(X,T)} :- chosen(X,T),simple(X),time(T).

%% EFFECTS INITIALIAZE

available(X,T+1):-initialize(X,T), simple(X),time(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%

%% ASSOCIATE

{associate(X1,X2,X3,T)}:- molecule(X1),molecule(X2),complex(X3),
```

```
                          association_reaction(X1,X2,X3),
                          available(X1,T),
                          available(X2,T),time(T).
                          %X1!=X2,X1!=X3,X2!=X3.

%% EFFECTS ASSOCIATE

available(X3,T+1) :- associate(X1,X2,X3,T),
molecule(X1),molecule(X2),complex(X3),time(T).

change(X1,T):-associate(X1,X2,X3,T),
molecule(X1),molecule(X2),complex(X3),time(T).

change(X2,T):-associate(X1,X2,X3,T),
molecule(X1),molecule(X2),complex(X3),time(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%

%% ASSOCIATE WITH CATALYZE

{associate_with_catalyze(X1,X2,X3,T)}:-
molecule(X1),molecule(X2),complex(X3),

catalyzed_association_reaction(X1,X2,X3),
                                       available(X1,T),
                                       available(X2,T),
                                       time(T).

%% EFFECTS ASSOCIATE WITH CATALYZE

available(X3,T+1):-
associate_with_catalyze(X1,X2,X3,T),molecule(X1),molecule(X2),complex(X
3),time(T).

change(X1,T):-
associate_with_catalyze(X1,X2,X3,T),molecule(X1),molecule(X2),complex(X
3),time(T).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%

%% SYNTHESIZE

{synthesize(X1,X2,T)}:-
molecule(X1),molecule(X2),synthesis_reaction(X1,X2),available(X1,T),
time(T).%X1!=X2.

% EFFECTS SYNTHESIZE

available(X2,T+1):- synthesize(X1,X2,T), molecule(X1), molecule(X2),
time(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
% EFFECTS
```

```
available(X,T+1) :- available(X,T), not change(X,T), molecule(X),
time(T).

num_subs(L,T+1) :- num_subs(L,T), not change_subs(L,T), level(L),
time(T).

chosen(X,T+1) :- chosen(X,T), simple(X), time(T).

available_molecule(X):-available(X,T),molecule(X),time(T).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%


:-choose(X1,L1,L2,T),choose(X2,L1,L2,T),
simple(X1),simple(X2),level(L1),level(L2),time(T),X1!=X2.

:-initialize(X,T1),initialize(X,T2),simple(X),time(T1),time(T2),T1<T2.


goal1:-available_molecule(prbp1p2_ap2).
goal1:-available_molecule(pcaf_p300).




hide simple(X).
hide complex(X).
hide possible(X).
hide catalyzed_association_reaction(X1,X2,X3).
hide association_reaction(X1,X2,X3).
hide catalyzed_association_reaction(X1,X2,X3).
hide next(L1,L0).
hide time(T).
hide molecule(X).
hide level(L).
hide chosen(X,T).
hide change(X,T).
hide change_subs(L,T).
hide available(X,T).
```