ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΑΚΑΔΗΜΑΙΚΟ ΕΤΟΣ 2007 – 2008

ΘΕΜΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ:

"Πιθανοκρατικός Έλεγχος Μοντέλων με το PRISM: Ανάλυση του Host Identity Protocol (HIP)"

ΑΛΕΞΙΟΥ ΝΙΚΟΛΑΟΣ
ΑΕΜ 873

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ
ΠΑΝΑΓΙΩΤΗΣ ΚΑΤΣΑΡΟΣ

26/2/2008

ΠΕΡΙΕΧΟΜΕΝΑ

Πρόλογος

Ο χώρος του ελέγχου μοντέλων είναι ένας νέος και δυναμικά αναπτυσσόμενος τομέας της πληροφορικής λόγω των πολλών πλεονεκτημάτων που προσφέρει, όπως είναι η αποτελεσματικότητα, το μικρό κόστος, η ταχύτητα και η ευκολία που εξάγονται τα αποτελέσματα από το εκάστοτε μοντέλο που αναλύεται.

Έχουν αναπτυχθεί πολλά αξιόλογα εργαλεία ελέγχου μοντέλων τα τελευταία χρόνια και κάθε ένα από αυτά βλέπει τα προβλήματα που καλείται να λύσει από διαφορετική οπτική γωνία. Ένα από τα νεότερα και πολύ αξιόλογα εργαλεία είναι το prism το οποίο εκτελεί πιθανοκρατικό (probabilistic) έλεγχο του μοντέλου που αναλύει. Η χρησιμότητα των εργαλείων ελέγχου μοντέλων πηγάζει από το ευρύ φάσμα δραστηριοτήτων και συμπεριφορών που μπορούν να μοντελοποιηθούν. Ένα τέτοιο παράδειγμα είναι και τα πρωτόκολλα επικοινωνίας. Με τη βοήθεια αυτών των εργαλείων είναι δυνατόν να δημιουργηθεί ένα ακριβές μοντέλο όπου αναπαριστά την συμπεριφορά ενός πρωτοκόλλου επικοινωνίας με σκοπό την μελέτη για τις αδυναμίες που μπορεί να κρύβει, παίρνοντας σαν αποτέλεσμα ποιοτικά και ποσοτικά στοιχεία.

Η διπλωματική αυτή ασχολείται με την μοντελοποίηση του πρωτοκόλλου HIP (host identity protocol). Το πρωτόκολλο αυτό κάνει χρήση κρυπτογραφίας με χρήση κλειδιού και έχει έμφυτα ορισμένα χαρακτηριστικά ώστε να είναι ανθεκτικό σε επιθέσεις τύπου άρνησης υπηρεσίας (DoS attack). Με τη χρήση του εργαλείου prism διεξάγεται πιθανοκρατικός έλεγχος (probabilistic model checking) στο μοντέλο του πρωτοκόλλου και εξάγονται ποσοτικά αποτελέσματα σχετικά με την πιθανότητα να επιτύχει επίθεση τύπου άρνησης υπηρεσίας εναντίον του πρωτοκόλλου.

Το πρωτόκολλο HIP μοντελοποιείται χρησιμοποιώντας μαρκοβιανές αλυσίδες διακριτού χρόνου σε γλώσσα Prism και έπειτα ελέγχεται με ιδιότητες τύπου pctl (probabilistic computational tree logic).

Στόχος της εργασίας

Ο στόχος της εργασίας είναι να παρουσιαστεί το εργαλείο ελέγχου μοντέλων prism και να μοντελοποιηθεί το πρωτόκολλο επικοινωνίας HIP. Μοντελοποιείται η λειτουργία ενός εισβολέα, ο οποίος θα επεμβαίνει στην ομαλή λειτουργία του πρωτοκόλλου, με σκοπό να επιτύχει επίθεση άρνησης υπηρεσίας. Ως αποτελέσματα της όλης πτυχιακής εργασίας, λαμβάνονται τα συνολικά κόστη όπου θεωρούνται ως υπολογιστικός φόρτος των διεργασιών του πρωτοκόλλου, επιτρέποντας την μελέτη της απόδοσης του πρωτοκόλλου, όταν αυτό βρίσκεται σε κατάσταση επίθεσης. Σαν αποτέλεσμα μας επιστρέφεται μια σημαντική επίθεση άρνησης υπηρεσιών από την πλευρά των συμμετεχόντων, ενώ παρουσιάζεται και η συγκριτική μελέτη του υπολογιστικού κόστους όπου θα χρειαστεί ένας επίδοξος εισβολέας σε σχέση με το κόστος των νόμιμων χρηστών. Τα αποτελέσματα αυτής της εργασίας παρουσιάζονται στο τελευταίο κεφάλαιό της.

ΚΕΦΑΛΑΙΟ Ι

1. ΕΛΕΓΧΟΣ ΜΟΝΤΕΛΩΝ ΚΑΙ ΕΡΓΑΛΕΙΑ ΜΟΝΤΕΛΟΠΟΙΗΣΗΣ

1.1 Εισαγωγή στο Model Checking

Κατά την διάρκεια ανάπτυξης ενός έργου π.χ. ανάπτυξη λογισμικού, hardware ή και ακόμη μεγαλύτερων βιομηχανικών έργων η απαίτηση ικανοποίησης ορισμένων προδιαγραφών – ιδιοτήτων είναι απαραίτητη. Τα έργα αυτά θα πάρουν την τελική τους μορφή έπειτα από κάποιες δοκιμές. Επίσης πολλές φορές κρίνεται αναγκαία η βελτίωση ενός υπάρχοντος συστήματος, υπολογιστικού, βιομηχανικού και άλλων.

Η ανάπτυξη όμως αυτή ή η βελτίωση έχει κάποιο κόστος, όχι μόνο οικονομικό αλλά και άλλων ειδών που δεν είναι δυνατόν να αγνοηθεί. Για παράδειγμα κατά την ανάπτυξη ενός πρωτοκόλλου επικοινωνίας δύο δικτύων εκτός από την λειτουργικότητά του πρέπει να ελεγχθεί και το επίπεδο ασφάλειάς του, πράγμα το οποίο δεν μπορεί να γίνει στην πράξη, όταν δηλαδή θα έχει υιοθετηθεί το εν λόγω πρωτόκολλο. Επίσης στο σύστημα ελέγχου των πτήσεων ενός αεροδρομίου για παράδειγμα δεν είναι τόσο απλή η εφαρμογή αλλαγών χωρίς πρώτα να πραγματοποιηθούν κάποιοι έλεγχοι, καθώς εκτός από το προφανές οικονομικό κόστος που θα είχε αν τύχαινε κάποια δυσλειτουργία υπάρχει και ο κίνδυνος απώλειας ζωών.

Αυτή η αδυναμία πραγματοποίησης ελέγχων των συστημάτων κατά την λειτουργία τους καθώς και η απαίτηση ικανοποίησης ιδιοτήτων των συστημάτων όταν θα τεθούν σε λειτουργία ή θα βγουν στο εμπόριο είναι που δημιούργησε τον έλεγχο μοντέλων των πραγματικών συστημάτων (model checking).

1.2 Ο Έλεγχος Μοντέλων

Ο έλεγχος μοντέλων (model checking) είναι μια διαδικασία εξέτασης αν ένα μοντέλο ικανοποιεί μια δοθείσα φόρμουλα. Η έννοια είναι γενική και έχει εφαρμογή σε κάθε είδους λογική και τα αντίστοιχα μοντέλα. Ένα απλό πρόβλημα ελέγχου μοντέλου είναι να ελέγξουμε αν μια δοθείσα φόρμουλα σε προτασιακή λογική ικανοποιείται από κάποιο μοντέλο [1].

Μια σημαντική κατηγορία μεθόδων ελέγχου μοντέλων έχει αναπτυχθεί με σκοπό την αλγοριθμική επαλήθευση των συστημάτων. Αυτό γίνεται επαληθεύοντας αν το μοντέλο, που συχνά προέρχεται από σχεδιασμό hardware ή software, ικανοποιεί μια λογική φόρμουλα σε κάποιο χρόνο.

Ο έλεγχος μοντέλου συνήθως γίνεται σε έργα hardware και αυτό γιατί σε έργα software η προσέγγιση δεν μπορεί να γίνει αμιγώς αλγοριθμικά (λόγω του υψηλού βαθμού αβεβαιότητας). Δηλαδή στην πράξη είναι πολύ πιθανό να αποτύχει να αποδείξει μια ιδιότητα του μοντέλου. Τα εργαλεία ελέγχου μοντέλων (model checking tools) αντιμετωπίζουν ένα πρόβλημα μεγάλου αριθμού πιθανών καταστάσεων που είναι πιο γνωστό ως state explosion problem. Για να υπερβούν αυτό το πρόβλημα ακολουθούνται τρεις διαφορετικές προσεγγίσεις σύμφωνα με [1].

1. Οι συμβολικοί αλγόριθμοι αποφεύγουν να χτίσουν ένα γράφο, αντίθετα αναπαριστούν το γράφο χρησιμοποιώντας την προτασιακή λογική. Επίσης χρησιμοποιούνται και BDDs (binary decision diagrams) για την λήψη αποφάσεων. Πιο πρόσφατη είναι η χρήση SAT solvers για την επίλυση του γράφου.
2. Η επιλεκτική μείωση τάξης (Partial order reduction) μπορεί να χρησιμοποιηθεί για να μειωθεί ο αριθμός των διαδικασιών που χρειάζεται να ληφθούν υπόψη. Η βασική ιδέα είναι ότι αν δεν έχει σημασία για την προς απόδειξη ιδιότητα αν το A ή το B εκτελούνται πρώτα, τότε είναι χάσιμο χρόνου να υπολογίζονται και η AB και η BA ενέργεια.

3. Η αφαίρεση προσπαθεί να αποδείξει ιδιότητες σε ένα σύστημα απλοποιώντας το σύστημα. Το απλοποιημένο σύστημα συνήθως δεν ικανοποιεί ακριβώς τις ίδιες ιδιότητες με το αρχικό και γι' αυτό μια διαδικασία επαναπροσδιορισμού ίσως είναι απαραίτητη. Γενικά η αφαίρεση πρέπει να γίνεται σωστά, δηλαδή η ιδιότητα που αποδεικνύεται με την αφαίρεση να ισχύει και για το αρχικό σύστημα. Ωστόσο πολύ συχνά η αφαίρεση δεν είναι ολοκληρωμένη, δηλαδή δεν ισχύει η παραπάνω ιδιότητα.

1.3 Εργαλεία Αυτόματου Ελέγχου Μοντέλων

Παρουσιάζονται ενδεικτικά ορισμένα εργαλεία αυτόματου ελέγχου μοντέλων. Δίνονται περιληπτικά ορισμένα στοιχεία και πληροφορίες για ορισμένα εργαλεία προσομοίωσης με σκοπό μια πρώτη γνωριμία με τον τρόπο λειτουργίας και λογικής τους. Τα εργαλεία που παρουσιάζονται είναι τα NuSMV, MRMC, UPPAAL, KRONOS, ALLOY, APMC και CADP.

1.3.1 NuSMV

Το NuSMV είναι ένας συμβολικός model-checker που αναπτύχθηκε από το Formal Methods Group του Carnegie Mellon University , Γένοβα και του πανεπιστημίου του Trento σύμφωνα με [2].

Το NuSMV είναι μια επανέκδοση και επέκταση του SNN, του πρώτου model-checker βασισμένου σε BDD's (binary decision diagrams). Έχει σχεδιαστεί για να είναι μια ανοιχτή αρχιτεκτονική για model-checking και μπορεί να χρησιμοποιηθεί αξιόπιστα για την επαλήθευση εργαστηριακών σχεδίων, σαν πυρήνας για συνηθισμένα εργαλεία επαλήθευσης, είτε σαν testbed για τυπικές μεθόδους επαλήθευσης (formal verification methods).

Η γλώσσα του NuSMV είναι σχεδιασμένη έτσι ώστε να επιτρέπει τον ορισμό αυτομάτων πεπερασμένων καταστάσεων (Finite State Machines, FSM), που μπορεί να ποικίλλουν από σύγχρονα έως ασύγχρονα αυτόματα και από σαφώς ορισμένα έως αφηρημένα αυτόματα. Αφού σκοπός της γλώσσας είναι να περιγράψει FSM ο τύπος δεδομένων που δέχεται σαν είσοδο το NuSMV μπορεί να είναι Booleans, μονόμετρα (scalars), και σταθερούς πίνακες (fixed arrays). Επίσης μπορούν να κατασκευαστούν στατικές μορφές δεδομένων.

Το NuSMV χρησιμοποιεί LTL model checking καθώς και model checking βασισμένο σε SAT. Αυτά τα δύο χαρακτηριστικά αποτελούν επίσης και βελτιώσεις του NuSMV σε σχέση με το SMV.

```
MODULE main
VAR
request : boolean;                          μεταβλητή τύπου boolean
state : fready, busyg;                      μεταβλητή τύπου scalar
3
ASSIGN
init(state) := ready;                       αρχική κατάσταση του state
next(state) := case                         επόμενη κατάσταση του state
state = ready & request = 1 : busy;         και προϋποθέσεις που πρέπει
1 : fready, busyg;                          να ικανοποιούνται για να
esac;                                       γίνει η μετάβαση
```

1.3.2 MRMC

Το MRMC είναι ένας model checker για διακριτού και συνεχούς χρόνου μοντέλα Markov (Markov reward models). Δημιουργήθηκε από την ομάδα Software Modelling and Verification (MOVES) του πανεπιστημίου RWTH του Aachen της Γερμανίας και την ομάδα Formal Methods & Tools του πανεπιστημίου Twente της Ολλανδίας. Το MRMC είναι ένας διάδοχος ενός πολύ γνωστού εργαλείου, του ETMCC, το οποίο είναι ένα πρωτότυπο εργαλείο model checker, για τον έλεγχο συνεχών αλυσίδων Markov (continuous Markov chains) σύμφωνα με [3].

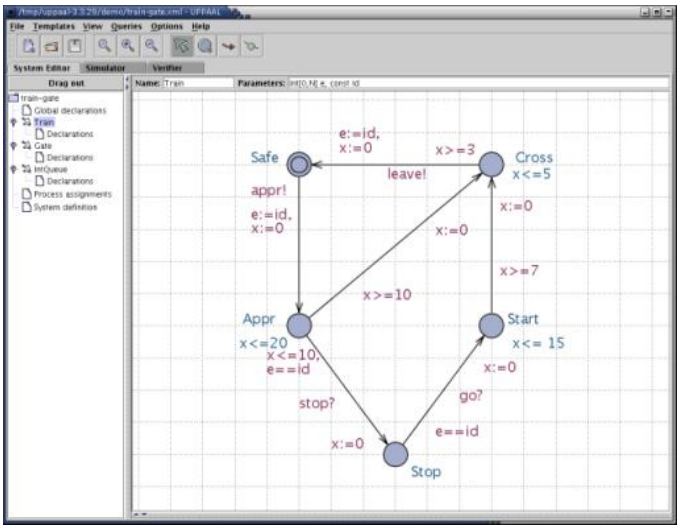
Υποστηρίζει επεκτάσεις της PCTL και της CSL (PRCTL και CSRL) και επιτρέπει την αυτόματη επαλήθευση ιδιοτήτων που αφορούν μακροχρόνιες και στιγμιαίες ανταμοιβές όπως και συσσωρευμένες ανταμοιβές. Ακόμη υποστηρίζει αν ένα σύνολο από καταστάσεις στόχους μπορεί να ικανοποιηθεί από το μοντέλο, σε εξάρτηση από το χρόνο και συσσωρευμένες αμοιβές.

Το MRMC είναι ένα εργαλείο γραμμής εντολών (command line tool) γραμμένο σε γλώσσα C. Αυτό επιτρέπει το εργαλείο να είναι μικρό και γρήγορο. Το MRMC τρέχει σε περιβάλλον linux μόνο ωστόσο μπορεί να εκτελεστεί και σε windows αν χτιστεί και εκτελεστεί σε Cygwin. Το εργαλείο περιμένει τεσσάρων ειδών αρχεία για είσοδο, αρχεία με κατάληξη .tra, .lab, .rew, .rewi, τα οποία καθορίζουν τις πιθανότητες των μεταβάσεων, τις επιβραβεύσεις για τις μεταβάσεις και άλλα.

1.3.3 UPPAAL

Το UPPAAL είναι ένας ελεγκτής μοντέλων που αναπτύχθηκε από την συνεργασία δύο πανεπιστημίων. Συγκεκριμένα από το Basic Research in computer science του πανεπιστημίου Aalborg της Δανίας και του Department of information technology του πανεπιστημίου της Uppsala στη Σουηδία σύμφωνα με [4].

Είναι κατάλληλο για συστήματα που μπορούν να μοντελοποιηθούν σαν μια συλλογή από μη ντετερμινιστικές διεργασίες, με πεπερασμένο αριθμό καταστάσεων και πραγματικών τιμών, ρολόγια που επικοινωνούν μεταξύ τους με κανάλια ή διαμοιραζόμενες μεταβλητές. Τυπικές περιπτώσεις εφαρμογής είναι οι ελεγκτές πραγματικού χρόνου και τα πρωτόκολλα επικοινωνίας όπου η διάσταση του χρόνου είναι κρίσιμο μέγεθος.



Εικόνα 1.1 η οθόνη του UPPAAL.

Το UPPAAL αποτελείται από τρία βασικά μέρη: μια περιγραφική γλώσσα, έναν εξομοιωτή και τον model checker ενώ η περιγραφική του γλώσσα είναι μια μη ντετερμινιστική γλώσσα εντολών με τύπους δεδομένων όπως bounded integers, arrays κλπ. Χρησιμεύει σαν γλώσσα μοντελοποίησης και σχεδίασης για να περιγράψει τη συμπεριφορά του συστήματος σαν συστήματα αυτομάτων με ρολόγια και μεταβλητές δεδομένων. Ο εξομοιωτής είναι ένα εργαλείο επαλήθευσης που επιτρέπει την εξέταση των πιθανών δυναμικών εκτελέσεων του συστήματος στην αρχή της σχεδίασης και γι' αυτό παρέχει ένα φτηνό μέσο εντοπισμού λαθών πριν την επαλήθευση από τον model checker, ο οποίος καλύπτει εξαντλητικά τη συμπεριφορά του συστήματος. Το UPPAAL για την επαλήθευση του μοντέλου χρησιμοποιεί ένα υποσύνολο της γλώσσας CTL (computer tree logic) που αποτελεί μια σαφώς και καλά ορισμένη γλώσσα.

1.3.4 KRONOS

Το KRONOS [5] είναι ένα εργαλείο που αναπτύχθηκε με σκοπό την επαλήθευση συστημάτων πραγματικού χρόνου. Τα στοιχεία αυτών των συστημάτων μοντελοποιούνται από χρονισμένα αυτόματα (timed automata) και οι επιτρεπτές καταστάσεις περιγράφονται με την TCTL λογική. Η TCTL είναι μια επέκταση της TCL λογικής.

Το KRONOS ελέγχει αν ένα αυτόματο ικανοποιεί μια TCTL φόρμουλα. Ο αλγόριθμος του model-checking βασίζεται πάνω σε μια συμβολική αναπαράσταση του χώρου καταστάσεων με σετ από γραμμικούς περιορισμούς.

Διανέμεται ελεύθερα για ακαδημαϊκή μη κερδοσκοπική χρήση. Μπορεί να λειτουργήσει σε συστήματα Solaris, Linux και Windows και δημιουργήθηκε από τους: Conrado Daws, Alfredo Olivero , Stavros Tripakis , SergioYovine.

1.3.5 ALLOY

Ο Alloy Analyzer [6] είναι ένα εργαλείο που αναπτύχθηκε από το Software design group για την ανάλυση μοντέλων που έχουν αναλυθεί στην γλώσσα Alloy, η οποία είναι μια απλή δομημένη γλώσσα βασισμένη στην first order λογική. Το εργαλείο μπορεί να παράγει τιμές μεταβλητών, να υπολογίζει την εκτέλεση πράξεων και να επιβεβαιώνει ή να απορρίπτει ιδιότητες του μοντέλου. Το εργαλείο μπορεί να παράγει περιπτώσεις αμετάβλητων, να εξομοιώσει την εκτέλεση πράξεων και να ελέγξει τις ιδιότητες ενός μοντέλου. Ο alloy analyzer προσφέρει αυτόματη προσομοίωση και έλεγχο. Έχει αναπτυχθεί από το Software Design Group του MIT.

Ο alloy analyzer είναι ένας model finder δηλαδή δοθείσης μιας λογικής φόρμουλας στη γλώσσα που καταλαβαίνει το alloy, προσπαθεί να βρει ένα μοντέλο, ένα όριο στις τιμές των μεταβλητών, που κάνει την φόρμουλα να ισχύει.

Η λογική του alloy:

1. Όλα είναι μια σχέση: Το alloy χρησιμοποιεί τις σχέσεις για όλους τους τύπους δεδομένων και τις δομές στο χώρο και το χρόνο. Η σχέση στη γλώσσα του alloy συμβολίζεται με την τελεία «.» . Οι σχέσεις χρησιμοποιούνται γιατί είναι εύκολο να κατανοηθούν ιδιαίτερα μέσω ενός γραφήματος και είναι εύκολο να αναλυθούν.
2. Μη εξειδικευμένη λογική: όχι ειδικές δομές για ίχνη, συγχρονισμό.
3. Αντιπαραδείγματα και σφαίρα: Παρατηρήσεις για την ανάλυση σχεδίασης δείχνουν τα δύο εξής στοιχεία. Πρώτον ότι οι περισσότερες υποθέσεις είναι λανθασμένες και επίσης πως τα περισσότερα λάθη έχουν αντιπαραδείγματα.
4. Ανάλυση με το SAT.

1.3.6 APMC

Το APMC (Approximate Probabilistic Model Checker) είναι ένας κατανεμημένος model checker για πλήρως πιθανοκρατικά συστήματα που χρησιμοποιεί ένα υπολογιστικό μοντέλο πελάτη/εξυπηρέτη για να κατανέμει τη δημιουργία μονοπατιού και την επαλήθευση κάποιας φόρμουλας σε μια ομάδα από σταθμούς εργασίας [1].

Η προσέγγιση του APMC χρησιμοποιεί μία αποτελεσματική μέθοδο Monte carlo, για προσεγγιστική ικανοποίηση ιδιοτήτων πλήρως πιθανοκρατικών συστημάτων. Οι ιδιότητες που ελέγχονται εκφράζονται σε LTL (Linear Temporal Logic). Το μοντέλο, η φόρμουλα και οι λοιποί παράγοντες παραδίδονται στον server, που αποκαλείται και master. Το μοντέλο και η φόρμουλα μεταφράζονται σε πηγαίο κώδικα C, μεταγλωττίζονται και το αποτέλεσμα στέλνεται στους πελάτες-εργάτες. Συνήθως οι εργάτες στέλνουν αποτελέσματα κάποιας επαλήθευσης και λαμβάνουν μήνυμα από τον master ώστε να ξέρουν αν θα συνεχίσουν ή θα σταματήσουν τον υπολογισμό.

Πρόσφατα το APMC ξαναγράφτηκε για την έκδοση 3.0 σε γλώσσα JAVA. Στην Τρίτη του έκδοση το APMC μπορεί να χειριστεί συστήματα διακριτού αλλά και συνεχούς χρόνου.

1.3.7 CADP

Το CADP (construction and analysis of distributed systems, πρωτύτερα γνωστό ως CAESAR/ALDEBARAN πακέτο ανάπτυξης) είναι ένα ευρέως γνωστό πακέτο εργαλείων για την σχεδίαση πρωτοκόλλων επικοινωνίας και κατανεμημένων συστημάτων [7]. Έχει αναπτυχθεί από την ομάδα VASY στο INRIA Rhone-Alpes και συνδέεται με πολλά συμπληρωματικά εργαλεία. Το CADP αναβαθμίζεται συχνά και χρησιμοποιείται σε πολλές βιομηχανικές εφαρμογές.

Επίσης προσφέρει μια μεγάλη λίστα από λειτουργίες που ποικίλλουν από βήμα προς βήμα εξομοίωση μέχρι μαζικό παράλληλο model checking.

Δέχεται τρεις διαφορετικούς τύπους εισόδου:

1. Υψηλού επιπέδου αναλύσεις πρωτοκόλλων γραμμένα σε γλώσσα LOTTO. Το CADP έχει 2 compilers (Cæsar και Cæsar.adt) που μεταφράζουν την γλώσσα LOTTO σε C κώδικα που μπορεί να χρησιμοποιηθεί για την εξομοίωση, την επαλήθευση και την εξομοίωση.
2. Χαμηλού επιπέδου αναλύσεις πρωτοκόλλων που καθορίζονται σαν Labeled Transition Systems (εν συντομία LTS). Για παράδειγμα Αυτόματα πεπερασμένων καταστάσεων των οποίων οι μεταβάσεις μεταξύ των καταστάσεων χαρακτηρίζονται με ονόματα ενεργειών.
3. Μεσαίου επιπέδου δίκτυα από επικοινωνούντα LTSs, για παράδειγμα αυτόματα πεπερασμένων καταστάσεων που υπάρχουν παράλληλα και συγχρονίζονται χρησιμοποιώντας «σημεία» συνάντησης. Αυτά τα δίκτυα μπορούν νε περιγραφούν με 2 διαφορετικούς τύπους εισόδου. EXP: LTSs που συνδυάζονται χρησιμοποιώντας τελεστές παράλληλης σύνδεσης και απόκρυψης. FC2: LTSs που συνδυάζονται χρησιμοποιώντας ένα προϊόν συγχρονισμού.

ΚΕΦΑΛΑΙΟ ΙΙ

2. ΤΟ ΕΡΓΑΛΕΙΟ PRISM

2.1 Εισαγωγή στο Prism

Το prism είναι ένα πιθανοκρατικό εργαλείο ελέγχου μοντέλων (model checker), ένα εργαλείο για ανάλυση και μοντελοποίηση συστημάτων που παρουσιάζουν συμπεριφορά σχετική με πιθανότητες. Ο έλεγχος βασισμένος σε πιθανότητες είναι μια τυπική μέθοδος επαλήθευσης (formal verification technique) και βασίζεται στην κατασκευή ενός ακριβούς μαθηματικού μοντέλου ενός συστήματος το οποίο και θα αναλυθεί. Το prism υποστηρίζει τρεις τύπους μοντέλων με βάση την πιθανοκρατική ανάλυση που στοχεύουμε να πραγματοποιήσουμε, σύμφωνα με [8].

- Αλυσίδες markov διακριτού χρόνου (discrete-time Markov chains, DTMCs)
- Διαδικασίες απόφασης Markov (Markov Decision processes, MDPs)
- Αλυσίδες Markov συνεχούς χρόνου (continuous time Markov chains, CTMCs)

Τα μοντέλα εκφράζονται στο εργαλείο περιγράφοντάς τα με τη γλώσσα prism, μια απλή και υψηλού επιπέδου γλώσσα μοντελοποίησης. Οι ιδιότητες των μοντέλων αναλύονται στο εργαλείο χρησιμοποιώντας την γλώσσα ορισμού ιδιοτήτων του Prism η οποία βασίζεται σε δύο πιθανοκρατικές χρονικές λογικές [8].

- Την PCTL (probabilistic computation tree logic), για μοντέλα τύπου DTMC και MDP.
- Και την CSL (continuous stochastic logic), για τα CTMCs μοντέλα.

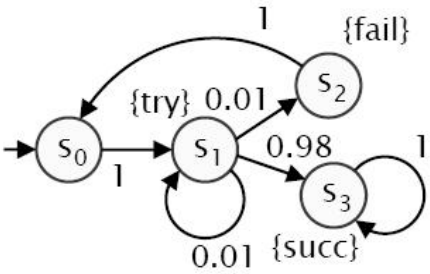
Το prism για να ελέγξει τις ιδιότητες των μοντέλων, είτε χρησιμοποιεί τυπικές μεθόδους επαλήθευσης βασισμένες σε αριθμητικούς υπολογισμούς, είτε ανάλυση με προσομοίωση διακριτών γεγονότων.

2.1.1 DTMCs

Τα DTMCs συνήθως εκφράζονται με (S,s',P,L) όπου σύμφωνα με [14]:

- Το S είναι μια πεπερασμένη συλλογή καταστάσεων
- $s' \in S$ είναι η αρχική κατάσταση
- $P: S \times S \rightarrow [0,1]$ είναι ένας πίνακας με τις πιθανότητες μεταβάσεων για όλα τα s που ανήκουν στο S να ισχύει το εξής: $\sum_{s' \in S} P(s,s') = 1$
- $L : S \rightarrow 2^{AP}$ είναι μια συνάρτηση που δίνει ονόματα (ταμπέλες-labels) σε καταστάσεις χρησιμοποιώντας ατομικούς τύπους.

Όπως αναφέρεται στο [8] τα DTMC's έχουν ένα διακριτό χώρο καταστάσεων και οι μεταβάσεις γίνονται σε διακριτά βήματα χρόνου. Από κάθε κατάσταση η επίλογή του απογόνου γίνεται βάση διακριτών πιθανοκρατικών επιλογών. Για παράδειγμα:



Εικόνα 2.1 : από την κατάσταση So μπορεί να μεταβεί στην S1 με πιθανότητα 1, ενώ από την S1 με πιθανότητα 0.98 στην S3 και με 0.01 στην S2 ή παραμένει στην S1.

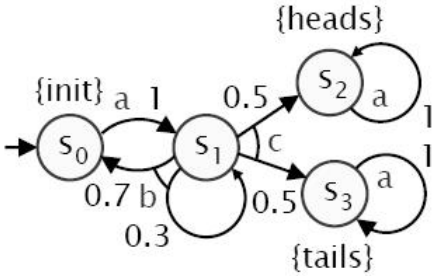
Ορισμένα όμως στοιχεία του μοντέλου μπορεί να μην είναι βασισμένα σε πιθανότητες και συνεπώς δεν πρέπει να μοντελοποιηθούν με πιθανότητες. Για παράδειγμα άγνωστα περιβάλλοντα μοντελοποίησης και άγνωστοι παράμετροι των μοντέλων.

2.1.2 MDPs

Τα MDPs συνήθως εκφράζονται με τη μορφή(S, s',Steps,L) όπου σύμφωνα με [9]:

- Το S είναι μια πεπερασμένη συλλογή καταστάσεων (state space)
- s'∈S είναι η αρχική κατάσταση
- Steps: S →2Act×Dist(S) είναι η συνάρτηση με τις πιθανότητες μετάβασης και όπου Act είναι ένα σύνολο ενεργειών και Dist(S) είναι το σύνολο των πιθανοκρατικών μεταβάσεων επί του συνόλου S
- L : S →2AP είναι μια συνάρτηση που δίνει ονόματα (ταμπέλες-labels) σε καταστάσεις χρησιμοποιώντας ατομικούς τύπους.

Αποτελούν μια προέκταση των DTMC's και επιτρέπουν μη ντετερμινιστική επιλογή. Σε κάθε κατάσταση υπάρχει μη ντετερμινιστική επιλογή μεταξύ αρκετών καταστάσεων απογόνων.



Εικόνα 2.2: από την κατάσταση S1 με μετάβαση b, υπάρχει πιθανότητα 70% να μεταβεί στην S0 και 30% να παραμείνει στην S1.

2.1.3 CTMCs

Τα continuous-time Markov chains έχουν τα εξής χαρακτηριστικά:

- Έχουν διακριτές καταστάσεις
- Τα βήματα του χρόνου είναι συνεχή
- Οι καθυστερήσεις κατανέμονται εκθετικά

Μπορούν να χρησιμοποιηθούν για να μοντελοποιήσουν :

- Βιολογικά μονοπάτια
- Συστήματα ελέγχου
- Χημικές αντιδράσεις

Για παράδειγμα η χημική αντίδραση $Na + Cl \leftrightarrow Na^{+} + Cl^{-}$,μοντελοποιείται στο prism με CTMC μοντέλο [8].

```
// ctmc model
stochastic

// constants
const int N1 = 10; // number of Na molecules
const int N2 = N1; // number of Cl molecules

// Na and Na+ module
module na

na : [0..N1] init N1;
// total number of Na and Na+ molecules is fixed at N1
// na is the number of Na molecules
// therefore N1-na gives the number of Na+ molecules

[e1] na>0  ->     na  : (na'=na-1);
[e2] na<N1 -> (N1-na) : (na'=na+1);

...
```

Εικόνα 2. 3: το μοντέλο ορίζεται σαν stochastic δηλαδή CTMC.

2.2 Η Γλώσσα του Prism

Στην ενότητα 2.2 αναλύονται οι βασικές έννοιες της γλώσσας του prism σύμφωνα με το [8].

2.2.1 Εισαγωγή

Για να κατασκευάσουμε και να αναλύσουμε ένα μοντέλο στο prism ,πρέπει πρώτα να οριστεί με τη γλώσσα του prism, μια απλή γλώσσα βασισμένη σε σύνολα καταστάσεων και σε αλληλεπιδρώντα στοιχεία (modules). Σε αυτή την ενότητα θα περιγραφεί η γλώσσα prism και θα δοθούν αντιπροσωπευτικά παραδείγματα αυτής.

Τα βασικά στοιχεία της γλώσσας είναι οι μεταβλητές και τα αλληλεπιδρώντα στοιχεία. Από αυτό το σημείο και έπειτα τα αλληλεπιδρώντα στοιχεία θα αναφέρονται ως modules και οι μεταβλητές ως variables. Ένα μοντέλο αποτελείται από ένα αριθμό modules που αλληλεπιδρούν και το κάθε ένα από αυτά περιλαμβάνει έναν αριθμό από τοπικές μεταβλητές. Οι τιμές των μεταβλητών αυτών σε κάθε χρονική στιγμή ορίζουν την κατάσταση του module, ενώ η συνολική κατάσταση του μοντέλου ορίζεται από τις καταστάσεις των modules κάθε χρονική στιγμή. Η συμπεριφορά του κάθε module περιγράφεται από ένα σύνολο εντολών. Οι εντολές έχουν την εξής μορφή:

```
[] guard - >prob_1 : update_1 + ... + prob_n : update_n;
```

Το guard είναι ένα κατηγόρημα για όλες τις μεταβλητές του μοντέλου, ακόμα και αυτών που δεν ανήκουν στο module που ανήκει η εντολή. Το κάθε update περιγράφει μια μετάβαση που μπορεί να κάνει το μοντέλο εάν η τιμή του guard είναι true. Επίσης

σε κάθε update δίνεται και κάποια πιθανότητα, ή σε κάποιες περιπτώσεις ρυθμός, η οποία και χαρακτηρίζει την μετάβαση. Μια μετάβαση καθορίζεται δίνοντας νέες τιμές στις μεταβλητές του module.

Έστω ότι έχουμε ένα σύστημα στο οποίο υπάρχουν δύο όμοιες διεργασίες ,οι οποίες πρέπει να λειτουργήσουν κάτω από αμοιβαίο αποκλεισμό. Κάθε διεργασία μπορεί να είναι σε μία από τις τρείς καταστάσεις {0,1,2}. Από την κατάσταση 0 μπορεί να μεταβεί στην κατάσταση 1 με πιθανότητα 0.2 είτε να μείνει στην ίδια κατάσταση με πιθανότητα 0.8. Από την κατάσταση 1, η διεργασία προσπαθεί να μεταβεί στην κατάσταση 2 ,δηλαδή στο κρίσιμο τμήμα της. Αυτό συμβαίνει μόνο αν η άλλη διεργασία δεν έχει εισέλθει στο κρίσιμο τμήμα της. Τέλος από την κατάσταση 2, μπορεί να παραμείνει στην 2 είτε να μεταβεί στην 0 με ίση πιθανότητα. Παρακάτω φαίνεται ο κώδικας του prism που περιγράφει το παραπάνω σύστημα.

```
// Example 1
// Two process mutual exclusion
mdp

module M1
    x : [0..2] init 0;

    [] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
    [] x=1 & y!=2 -> (x'=2);
    [] x=2 -> 0.5:(x'=2) + 0.5:(x'=0);
endmodule

module M2
    y : [0..2] init 0;

    [] y=0 -> 0.8:(y'=0) + 0.2:(y'=1);
    [] y=1 & x!=2 -> (y'=2);
    [] y=2 -> 0.5:(y'=2) + 0.5:(y'=0);
endmodule
```

Εικόνα 2.4: Αμοιβαίος αποκλεισμός εργασιών.

2.2.2 Modules και Μεταβλητές

Ένα module ορίζεται ως: `module name ... endmodule`. Στο παραπάνω παράδειγμα, ορίζονται μεταβλητές με αρχική τιμή: `x : [0..2] init 0;` Όπως η `x` με αρχική τιμή 0. Επίσης στο prism μπορούμε να ορίσουμε Boolean μεταβλητές: `b : bool init false;` Ορίζεται η μεταβλητή `b` ως Boolean και παίρνει αρχική τιμή false.

Τα ονόματα που δίνονται σε μεταβλητές και σε modules αναφέρονται ως αναγνωριστικά (identifiers) .Τα αναγνωριστικά μπορούν να δημιουργηθούν από χαρακτήρες, ψηφία και το στοιχείο '_'(undescore), αλλά δεν μπορούνε να ξεκινούν με ψηφίο. Επίσης τα identifiers είναι case sensitive και δεν μπορούν να πάρουν κανένα από τα παρακάτω ονόματα που αποτελούν λέξεις κλειδιά για το prism: `bool, C, ceil, const, CTMC, double, DTMC, endinit, endmodule, endrewards, endsystem, F, false, floor, formula, G, global, I, init, int, label, max, MDP, min, module, nondeterministic, P, Pmin, Pmax, prob, probabilistic, R, rate, rewards, Rmin, Rmax, S, stochastic, system, true, U, X.`

2.2.3 Αρχικές Καταστάσεις

Ο χώρος καταστάσεων ενός μοντέλου με πιθανότητες όπως περιγράφεται στη γλώσσα του prism , είναι μια συλλογή από όλες τις πιθανές τιμές που μπορούν να πάρουν οι μεταβλητές του. Το σύνολο των αρχικών καταστάσεων για ένα μοντέλο μπορεί να οριστεί με 2 τρόπους. Ο πιο συχνός τρόπος είναι να ορίσουμε αρχική κατάσταση για κάθε μεταβλητή του μοντέλου, αυτό γίνεται στην δήλωση της μεταβλητής όπως φαίνεται και στο προηγούμενο παράδειγμα. Εάν στην δήλωση της μεταβλητής παραλειφθεί η αρχική τιμή της, τότε αυτόματα στην μεταβλητή αυτή ορίζεται σαν αρχική τιμή η μικρότερη τιμή από το πεδίο τιμών της.

Επίσης μπορούμε να κατασκευάσουμε ένα μοντέλο με πολλαπλές αρχικές καταστάσεις. Αυτό γίνεται χρησιμοποιώντας την δομή `init...endinit` που μπορεί να χρησιμοποιηθεί οπουδήποτε μέσα στο αρχείο εκτός εντός του ορισμού ενός module. Επίσης αν χρησιμοποιηθεί η παραπάνω δομή απαγορεύεται η χρήση αρχικών τιμών για κάθε μεταβλητή χωριστά. Ανάμεσα στις λέξεις κλειδιά `init...endinit` πρέπει να υπάρχει ένα κατηγόρημα για όλες τις μεταβλητές του μοντέλου. Όποια κατάσταση ικανοποιεί το κατηγόρημα είναι μια αρχική κατάσταση.

Για παράδειγμα στο προηγούμενο παράδειγμα αν αφαιρέσουμε τον ορισμό αρχικών τιμών στις δηλώσεις των μεταβλητών και γράψουμε: `init x=0 endinit;` τότε θα έχουμε τρεις αρχικές καταστάσεις. Τις (0,0),(0,1) και (0,2). Αν τώρα δηλώσουμε `init x+y=1 endinit;` τότε έχουμε δύο αρχικές καταστάσεις. Τις (1,0) και (0,1).

2.2.4 Εντολές

Η συμπεριφορά του κάθε module περιγράφεται με εντολές. Για παράδειγμα έστω η εντολή: `[] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);` Οι δύο αγκύλες δηλώνουν πως πρόκειται για μια εντολή. Το αριστερό μέλος της εντολής, αριστερά από το βελάκι, ορίζει το πότε μπορεί να ισχύσει η εντολή. Στην συγκεκριμένη περίπτωση η εντολή μπορεί να εκτελεστεί όταν `x=0`. Το δεξιό σκέλος δηλώνει τις πιθανές μεταβάσεις και την πιθανότητα να πραγματοποιηθούν. Στο παράδειγμά μας η μεταβλητή x μπορεί να μεταβεί στην κατάσταση 1 με πιθανότητα 0.2 ή να παραμείνει στην 0 με πιθανότητα 0.8. Για μοντέλα τύπου MDP και DTMC πρέπει το άθροισμα των πιθανοτήτων στο δεξιό μέλος να είναι ίσο με ένα.

Στην δεύτερη εντολή του παραδείγματος `[] x=1 & y!=2 -> (x'=2);` ,στο αριστερό σκέλος της εντολής περιλαμβάνεται ο έλεγχος μιας μεταβλητής, της y , που δεν ορίζεται μέσα στο module που βρίσκεται η εντολή. Δηλαδή η συμπεριφορά ενός module μπορεί να βασιστεί σε μεταβλητές που ορίζονται σε κάποιο άλλο module του μοντέλου. Οι νέες τιμές των μεταβλητών ωστόσο, δεξιό μέρος, μπορούν να αφορούν μόνο τις μεταβλητές του module που βρίσκεται η εντολή. Επίσης όταν στο δεξιό μέλος έχουμε μόνο μια πιθανή μετάβαση για κάποια μεταβλητή τότε η πιθανότητα μπορεί να παραλειφθεί όπως φαίνεται στο παράδειγμα.

Έστω τώρα η εντολή `[] x1=1 & x2!=2 -> (x1'=2) & (x2'=3);` Παρατηρούμε ότι οι μεταβάσεις ενώνονται με το χαρακτήρα '`&`'. Αυτό σημαίνει ότι αν ισχύει το αριστερό σκέλος τότε η `x1` θα πάρει τιμή 2 και η `x2` την τιμή 3.

2.2.5 Παράλληλη Σύνθεση

Για ένα μοντέλο πιθανοτήτων που έχει υλοποιηθεί με τη γλώσσα prism, δομείται σαν την παράλληλη σύνθεση των modules του. Σε κάθε κατάσταση του μοντέλου υπάρχει ένα πλήθος εντολών που μπορεί να πραγματοποιηθεί. Η εντολές αυτές όπως έχει προαναφερθεί μπορεί να ανήκουν σε οποιαδήποτε από τα modules του μοντέλου. Η επιλογή ανάμεσα στο ποιά εντολή θα εκτελεστεί εξαρτάται από τον τύπο του μοντέλου. Για ένα MDP μοντέλο η επιλογή είναι μη ντετερμινιστική, ενώ για ένα DTMC μοντέλο η επιλογή είναι βασισμένη σε πιθανότητες. Κάθε εντολή επιλέγεται με ίση πιθανότητα. Για ένα CTMC μοντέλο η επιλογή της εντολής περιγράφεται ως ένας «αγώνας» μεταξύ των μεταβάσεων.

2.2.6 Μετονομασία Module

Το prism υποστηρίζει την μετονομασία(renaming) των modules. Στο παράδειγμα 1 το module `M1` είναι πανομοιότυπο με το `M2` και επομένως μπορούμε να αλλάξουμε τη δήλωσή του ως εξής: `module M2 = M1 [ x=y, y=x ] endmodule`

2.2.7 Σταθερές

Το prism υποστηρίζει τη δήλωση σταθερών. Οι σταθερές μπορεί να είναι integers, doubles ή Booleans και ορίζονται χρησιμοποιώντας τη λέξη κλειδί `const` όπως και στα παρακάτω παραδείγματα:

```
const int radius = 12;
const double pi = 3.141592;
const double area = pi * radius * radius;
const bool yes = true;
```

Οι σταθερές μπορούν να χρησιμοποιηθούν όπου αναμένεται μια σταθερή τιμή όπως για παράδειγμα ένα άνω ή κάτω όριο για μια μεταβλητή. Όπως στο παρακάτω παράδειγμα:

```
const int N = 10;
q : [0..N];
```

Όπως θα αναφερθεί παρακάτω μπορεί να μην οριστεί τιμή σε μια μεταβλητή και να οριστεί αργότερα μια μοναδική τιμή ή ένα πλήθος τιμών χρησιμοποιώντας πειράματα.

2.2.8 Εκφράσεις

Ο ορισμός μιας σταθεράς όπως περιγράφτηκε πιο πάνω χρησιμοποιεί μια έκφραση. Στο τμήμα αυτό θα οριστεί ακριβέστερα ποιοι τύποι εκφράσεων επιτρέπονται από το prism. Οι εκφράσεις μπορεί να περιλαμβάνουν τιμές (12, 3.143456, true, false κλπ), identifiers – αναγνωριστές που μπορεί να αναφέρονται σε μεταβλητές σταθερές κλπ, και τέλος τελεστές από την ακόλουθη λίστα:

- `*, /` (πολλαπλασιασμός, διαίρεση)
- `+, -` (πρόσθεση, αφαίρεση)
- `<, <=, >=, > , =, !=` (συσχετιστικοί τελεστές)

- `!` (λογικό όχι)
- `&` (λογικό και)
- `|` (λογικό ή)
- ? (εκτιμητής κατάστασης που ορίζεται ως εξής: `προϋποθεση ? a : b`, που σημαίνει "αν η προϋπόθεση είναι true then a else b")

Οι εκφράσεις εξετάζονται από αριστερά προς τα δεξιά και υποστηρίζουν προτεραιότητα τελεστών.(Η προτεραιότητα ορίζεται όπως και στην παραπάνω λίστα). Παρενθέσεις μπορεί να χρησιμοποιηθούν για να υπερβούμε την προτεραιότητα τελεστών. Οι εκφράσεις στη γλώσσα prism είναι παρόμοιες με τις συμβατικές γλώσσες προγραμματισμού όπως η C ,η C++ και η Java. Μια σημαντική διαφορά είναι ότι ο τελεστής της διαίρεσης επιστρέφει πάντα αποτέλεσμα τύπου float και όχι integer. Για παράδειγμα το αποτέλεσμα της διαίρεσης 22/7 είναι 3.142857 και όχι 3. Με τη χρήση εκφράσεων μπορούμε να εκφράσουμε τις τιμές που θα έχει μια μεταβλητή. Για παράδειγμα η έκφραση: `x = 1..5,7,10..13` είναι ίση με: `(x>=1 & x<=5) | (x=7) | (x>=10 & x<=13)`. Επίσης μπορούμε να χρησιμοποιήσουμε τον τελεστή ! για ολόκληρες εκφράσεις όπως για παράδειγμα η έκφραση: `x = b-1..b+1,b+6..b+8` είναι ίση με : `!((x>=b-1 & x<=b+1) | (x>=b+6 & x<=b+8)`.

Με εκφράσεις μπορούμε να χρησιμοποιήσουμε και μερικές συναρτήσεις που υποστηρίζει το prism. Η συναρτήσεις καλούνται χρησιμοποιώντας τη λέξη κλειδί func και το όνομα της συνάρτησης. `func(name, arg1, arg2, ...)`.
Προς το παρών οι υποστηριζόμενες από το prism συναρτήσεις είναι οι:

- `min,max` : που υπολογίζουν το ελάχιστο και το μέγιστο αντίστοιχα, δύο ή περισσοτέρων αριθμών
- `fllor, ceil`: που προσεγγίζουν το μικρότερο και μεγαλύτερο ακέραιο αριθμό από την τιμή που δέχεται η συνάρτηση σαν όρισμα.
- `Pow` : που υψώνει έναν ακέραιο σε έναν άλλο
- `Mod` : που βρίσκει το υπόλοιπο της διαίρεσης ενός ακεραίου με έναν άλλο.

Παραδείγματα της χρήσης των συναρτήσεων φαίνονται παρακάτω:

```
func(min, x+1, x_max)
func(max, a,b,c)
func(floor, 13.5)
func(ceil, 13.5)
func(pow, 2, 8)
func(pow, 9.0, 0.5)
func(mod, 1977, 100)
```

Οι εκφράσεις μπορούν να χρησιμοποιηθούν σε πολλά σημεία μέσα σε μια περιγραφή γλώσσας prism. Για παράδειγμα έκφραση μπορεί να χρησιμοποιηθεί για :

- Ορισμό σταθερών
- Ορισμό των ορίων μιας μεταβλητής
- Ορίσματα σε μια εντολή
- Να εκφράσουν πιθανότητα
- Αναβαθμίσει

2.2.9 Συγχρονισμός

Ένα άλλο στοιχείο του prism είναι ο συγχρονισμός. Όπως έχει προαναφερθεί μπορούμε να δώσουμε ονόματα στις εντολές τού κώδικα prism, δηλαδή ετικέτες, γράφοντας το όνομα της εντολής ανάμεσα στις αγκύλες που προηγούνται στον ορισμό της εντολής. Για παράδειγμα: `[serve] q>0 -> lambda:(q'=q-1);` Αυτές οι εντολές μπορούν να χρησιμοποιηθούν για να αναγκάσουν 2 ή περισσότερα modules να εκτελέσουν μεταβάσεις ταυτόχρονα, δηλαδή να συγχρονιστούν. Ας υποθέσουμε τον παρακάτω κώδικα που μοντελοποιεί έναν εξυπηρέτη με μια ουρά. Στην κατάσταση `q=3 και s=0`, το μοντέλο μπορεί να μεταβεί στην κατάσταση `q=2` και `s=1`, συγχρονισμένο πάνω στην ενέργεια με ετικέτα serve.

```
ctmc

const int N = 10;
const double mu = 1/10;
const double lambda = 1/2;
const double gamma = 1/3;

module queue
     q : [0..N];

     [] q<N -> mu:(q'=q+1);
     [] q=N -> mu:(q'=q);
     [serve] q>0 -> lambda:(q'=q-1);
endmodule

module server
     s : [0..1];

     [serve] s=0 -> 1:(s'=1);
     [] s=1 -> gamma:(s'=0);
endmodule
```

Εικόνα 2.5: Εξυπηρέτης με μια ουρά

2.2.10 Proccess Algebra Operators

Για να γίνει πιο ισχυρή η έννοια του συγχρονισμού, το prism επιτρέπει τον ορισμό του ακριβούς τρόπου με τον οποίο τα modules θα συγκροτηθούν παράλληλα. Αυτό γίνεται μέσω της δομής `system ….endsystem`, που προστίθεται στο τέλος της περιγραφής του μοντέλου. Η δομή αυτή πρέπει να περιέχει μια αλγεβρική έκφραση, που μπορεί να χαρακτηρίζει κάθε module ακριβώς μια φορά και μπορεί να χρησιμοποιεί τους παρακάτω τελεστές.

- `M1||M2` ,πλήρης παράλληλη σύνθεση των modules `M1` και `M2`, που συγχρονίζονται πάνω σε όλες τις ενέργειες που εμφανίζονται με και στα δύο modules.
- `M1|||M2` ,ασύγχρονη παράλληλη σύνθεση των δύο modules
- `M1|[a,b,….]|M2` ,παράλληλη σύνθεση με περιορισμούς των δύο modules, δηλαδή συγχρονισμός πάνω στις ενέργειες από το σύνολο `{a,b,.....}`.
- `M/{a,b,….}` ,απόκρυψη των ενεργειών του συνόλου `{a,b,….}` από το module `M`
- `M {a<-b,c<-d}` ,μετονομασία των εντολών `a` σε `b`, `c` σε `d` στο module `M`.

Οι δύο πρώτοι τύποι παράλληλης σύνθεσης (`||` και `|||`) είναι συσχετιστικοί και μπορούν να εφαρμοστούν σε περισσότερα των δύο modules στην ίδια έκφραση. Για παράδειγμα M1||M2||M3 κλπ. Στην αξιολόγηση μιας έκφρασης, οι τελεστές της μετονομασίας και της απόκρυψης , «/» και «<-» αντίστοιχα, συνδέονται ισχυρότερα από τους υπόλοιπους τρεις τελεστές. Δεν υπάρχουν άλλοι κανόνες προτεραιότητας και οι παρενθέσεις χρησιμοποιούνται για να ορίσουν την σειρά με την οποία συνθέτονται τα modules. Μερικά παραδείγματα εκφράσεων που μπορεί να περιληφθούν στην δομή `system…endsystem` μπορεί να είναι τα παρακάτω:

- `(station1|||station2|||station3)|[serve]|server`
- `((P1|[a]|P2)/{a})||Q`
- `((P1|[a]|P2) {a<-b})|[b]| Q`

Όταν δεν ορίζεται είδος παράλληλης σύνθεσης από το χρήστη, το prism αυτόματα υιοθετεί το πρώτο είδος παράλληλης σύνθεσης, δηλαδή M1||M2||M3…, για όλα τα modules του μοντέλου.

2.2.11 Καθολικές Μεταβλητές

Εκτός από τις τοπικές μεταβλητές που ανήκουν σε κάθε ένα module ξεχωριστά, το prism επιτρέπει και τον ορισμό καθολικών μεταβλητών, που αφορούν όλα τα modules του μοντέλου. Όπως και οι τοπικές μεταβλητές , οι καθολικές μπορεί να είναι integers και Booleans. Ορίζονται με παρόμοιο τρόπο όπως και οι τοπικές μεταβλητές, αλλά η δήλωση τους δεν μπορεί να συμπεριληφθεί μέσα στον ορισμό ενός module. Μερικά παραδείγματα δηλώσεων μπορεί να είναι τα ακόλουθα:

- `global g : [1..10];`
- `global b : bool init true;`

Η τιμή μιας καθολικής μεταβλητής μπορεί να αλλαχθεί από οποιοδήποτε module του μοντέλου ,γεγονός που δίνει άλλον ένα τρόπο για να αλληλεπιδρούν τα modules ενός μοντέλου. Ένα σημαντικό σημείο περιορισμού είναι ότι εντολές που συγχρονίζουν δύο modules, δεν μπορούν να μεταβάλλουν την τιμή μιας καθολικής μεταβλητής. Αν συμβεί αυτό το prism παράγει μήνυμα λάθους.

2.2.12 Formulas

Στο prism μπορούν να οριστούν formulas τα οποία χρησιμοποιούνται για αποφυγή επαναληπτικών κομματιών κώδικα. Μια formula αποτελείται από το όνομά της και μια έκφραση. Το όνομα της formula μπορεί να χρησιμοποιηθεί σαν συντομογραφία που καλεί την έκφραση που συνοδεύει, οπουδήποτε μέσα στον ορισμό του μοντέλου μπορεί να γίνει δεκτή μια έκφραση. Μια formula ορίζεται ως : `formula lfree =p2=0..4,6,10;` Δηλαδή εδώ ορίζεται η formula με όνομα `lfree` που ισχύει όταν η μεταβλητή `p2` είναι ίση με 0,1..4,6 ή 10. Επίσης οι formulas μπορούν να χρησιμοποιηθούν από εντολές όπως εδώ:

```
[] p1=2 & lfree -> (p1'=4);
```

Η παραπάνω εντολή είναι η ίδια με την εξής:

```
[] p1=2 & (p2=0..4,6,10) -> (p1'=4);
```

2.2.13 Αρχεία Γλώσσας Prism

Τα αρχεία που γράφονται με τη γλώσσα prism για την περιγραφή ενός μοντέλου μπορούν να περιέχουν όσους κενούς χαρακτήρες επιθυμεί ο σχεδιαστής (tabs,spaces κλπ) καθώς αυτά αγνοούνται κατά την γραμματική ανάλυση του μοντέλου. Επίσης μπορούν να προστεθούν και σχόλια με τη μορφή που προστίθενται και στη γλώσσα C, δηλαδή με //. Τα αρχεία prism παίρνουν τις προεκτάσεις .nm, .pm, .sm ανάλογα με το αν πρόκειται για μοντέλα τύπου MDP, DTMC ή CTMC αντίστοιχα.

2.2.14 Κόστος και Ανταμοιβές

Το prism στις τελευταίες εκδόσεις του επιτρέπει τον ορισμό και την ανάλυση ιδιοτήτων των μοντέλων βασισμένων σε κόστος και ανταμοιβή. Αυτό σημαίνει ότι το prism μπορεί να χρησιμοποιηθεί για να αιτιολογήσει όχι μόνο την πιθανότητα ένα μοντέλο να συμπεριφέρεται με συγκεκριμένο τρόπο, αλλά και για ένα ευρύτερο πεδίο ποσοτικών μετρήσεων που αφορούν την συμπεριφορά του μοντέλου. Για παράδειγμα το prism μπορεί να χρησιμοποιηθεί για να υπολογίσει ιδιότητες του τύπου: αναμενόμενος χρόνος, αναμενόμενος αριθμός χαμένων μηνυμάτων ή αναμενόμενη κατανάλωση ενέργειας. Η εισαγωγή του κόστους και των ανταμοιβών στο εργαλείο και των τεχνικών που βασίζονται σε αυτά έχει μόνο μερικώς εκπληρωθεί και συνεχίζεται.

Ωστόσο η βασική ιδέα πίσω από τα κόστη και ανταμοιβές είναι ότι καταστάσεις του μοντέλου ή μεταβάσεις και εκτελέσεις εντολών, μπορούν να συνδεθούν με πραγματικές τιμές. Στην πραγματικότητα δεν υπάρχει σαφής διαχωρισμός μεταξύ cost και reward, εκτός του ότι οι πρώτες θεωρούνται σαν κακό και οι δεύτερες σαν κάτι καλό. Έτσι το prism υποστηρίζει μόνο ανταμοιβές και από αυτό το σημείο και έπειτα ο όρος κόστος και ανταμοιβή θα είναι ταυτόσημος. Ο χρήστης είναι ελεύθερος να μεταφράσει τις τιμές που του επιστρέφει το εργαλείο σαν αποτέλεσμα. Σε αυτήν την ενότητα θα παρουσιαστεί πως τα μοντέλα που περιγράφονται σε γλώσσα prism μπορούν να εμπλουτιστούν με κόστη. Τα κόστη του μοντέλου περιγράφονται με την χρήση της δομής `rewards…endrewards`, η οποία μπορεί να εμφανιστεί οπουδήποτε μέσα στην περιγραφή του μοντέλου εκτός από τη δομή ενός module. Αυτές οι δομές περιλαμβάνουν ένα ή περισσότερα αντικείμενα, για παράδειγμα:

```
rewards
    true : 1;
endrewards
```

Αυτή η δήλωση αναθέτει την τιμή ένα σε κάθε κατάσταση του μοντέλου. Είναι μια απλή δήλωση αντικειμένου όπου όταν το αριστερό μέλος είναι αληθές τότε δίνεται η τιμή του δεξιού μέλους, δηλαδή ένα. Οι καταστάσεις του μοντέλου που ικανοποιούν το αριστερό μέλος δέχονται reward. Γενικότερα, rewards καταστάσεων μπορούν να ορισθούν χρησιμοποιώντας πολλαπλά αντικείμενα, κάθε ένα της μορφής guard:reward ,όπου guard είναι ένα κατηγόρημα (για οποιαδήποτε από τις μεταβλητές του μοντέλου) και reward είναι μια έκφραση (που μπορεί να περιλαμβάνει οποιαδήποτε από τις μεταβλητές, σταθερές κλπ του μοντέλου).
Για παράδειγμα:

```
rewards
    x=0 : 100;
    x>0 & x<10 : 2*x;
    x=10 : 100;
endrewards
```

Εδώ δίνεται reward 100 στις καταστάσεις που ικανοποιούν το `x=0 η x=10`, και `reward 2*x` στις καταστάσεις που ικανοποιούν το `x>0` και `x<10`. Αξίζει να σημειωθεί ότι ένα αντικείμενο κόστους μπορεί να δίνει διαφορετικά κόστη σε διαφορετικές καταστάσεις ανάλογα με τις τιμές των μεταβλητών στην κάθε μια κατάσταση. Οι καταστάσεις που δεν ικανοποιούν τα guards (αριστερά σκέλη) των αντικειμένων reward δεν θα τους δοθεί reward. Αν κάποια κατάσταση ικανοποιεί περισσότερα του ενός αντικείμενα reward, τότε το σύνολο των reward σε όλα τα αντικείμενα είναι αυτό που θα δοθεί στην κατάσταση. Επίσης κόστη μπορούν να δοθούν και στις μεταβάσεις μεταξύ των καταστάσεων.

2.3 Προσδιορισμός Ιδιοτήτων

Στην ενότητα 2.3 αναλύεται ο τρόπος ορισμού των ερωτημάτων με την PCTL λογική. Η ανάλυση έγινε σύμφωνα με το [8].

2.3.1 Εισαγωγή

Για να αναλύσουμε ένα πιθανοκρατικό μοντέλο που έχει περιγραφεί και κατασκευαστεί με το prism πρέπει να ορίσουμε μία ή περισσότερες ιδιότητες του μοντέλου (από αυτό το σημείο και έπειτα οι ιδιότητες αναφέρονται και ως properties) που να μπορέσουν να αξιολογηθούν από το εργαλείο. Στο prism αυτό γίνεται χρησιμοποιώντας χρονική λογική και χρησιμοποιώντας ιδιότητες που εκφράζονται σε PCTL για μοντέλα τύπου DTMC και MDP, και σε CSL για μοντέλα τύπου CTMC. Και οι δύο λογικές είναι προεκτάσεις της κλασικής χρονικής λογικής CTL. Στην πραγματικότητα το prism υποστηρίζει μια πληθώρα προεκτάσεων και προσθηκών των δύο αυτών λογικών. Παρακάτω φαίνονται ορισμένα παραδείγματα ορισμού ιδιοτήτων στο prism.

- `P>=1[true U terminate]`, σημαίνει ότι ο αλγόριθμος τερματίζει με επιτυχία.
- `"init"=>P<0.1 [trueU <=100num_errors>5]`,από μια αρχική κατάσταση, η πιθανότητα να συμβούν περισσότερα των 5 λάθη τις 100 πρώτες μονάδες χρόνου, είναι μικρότερη του 0.1 .
- `"down"=> P>0.75[!"fail" U[1,2]"up"]` ,όταν συμβεί ένας τερματισμός, η πιθανότητα επανεκκίνησης του συστήματος σε μία ή δύο ώρες, χωρίς να συμβούν περαιτέρω λάθη, είναι μεγαλύτερη του 0.75.
- `S<0.01[num_sensors<min_sensors]`,στη διάρκεια εκτέλεσης, η πιθανότητα ένας αριθμός από ακατάλληλους αισθητήρες να είναι λειτουργικός ,είναι μικρότερη του 0.75.

Πρέπει να σημειωθεί ότι οι παραπάνω πιθανότητες είναι ισχυρισμοί που επιβεβαιώνονται ή απορρίπτονται. Για παράδειγμα παίρνουμε σαν απάντηση ναι ή όχι. Αυτό συμβαίνει διότι οι αναφορές στις πιθανότητες σχετίζονται με ένα άνω ή κάτω όριο το οποίο μπορεί να ελέγξει αν είναι σωστό ή λάθος. Ωστόσο στο prism μπορούμε να ορίσουμε ιδιότητες που έχουν αριθμητική τιμή. Τα παρακάτω παραδείγματα επιτυγχάνουν ακριβώς αυτό.

```
P=? [ !proc2_terminate U proc1_terminate ]
```
Η πιθανότητα να τερματίσει η διεργασία ένα πριν την δύο.
```
Pmax=? [ true U<=T messages_lost > 10 ]
```
Την μέγιστη πιθανότητα να χαθούν περισσότερα των δέκα μηνυμάτων μέχρι τον χρόνο T.
```
S=? [ queue_size / max_size > 0.75 ]
```
Στη διάρκεια της εκτέλεσης, η ουρά να είναι τουλάχιστον 75% γεμάτη.

Επίσης το prism προσφέρει την δυνατότητα να υπολογιστούν τιμές για ιδιότητες ενός εύρους παραμέτρων και τον σχεδιασμό γραφημάτων που σχετίζονται με τα αποτελέσματα των πειραμάτων. Αυτός είναι συνήθως ένας πολύ χρήσιμος τρόπος για την αναγνώριση συγκεκριμένων προτύπων και γενικών κατευθύνσεων στην συμπεριφορά του συστήματος.

2.3.2 Αναγνώριση Συνόλου Καταστάσεων

Μια από τις σημαντικότερες διεργασίες κατά τον ορισμό των ιδιοτήτων ενός μοντέλου είναι η αναγνώριση επιμέρους συνόλων ή κλάσεων καταστάσεων του μοντέλου. Για παράδειγμα, για την επαλήθευση μιας ιδιότητας όπως ο αλγόριθμος τερματίζει με πιθανότητα 1, είναι πρώτα απαραίτητο να αναγνωριστούν πρώτα οι καταστάσεις που αναφέρονται στο «ο αλγόριθμος τερμάτισε επιτυχώς».

Στο prism αυτό επιτυγχάνεται γράφοντας μια έκφραση σε γλώσσα prism που αντιστοιχεί σε τιμή Boolean. Αυτή η έκφραση θα περιέχει αναφορές σε μεταβλητές και σταθερές από το μοντέλο που αναφέρεται. Οι καταστάσεις που ανταποκρίνονται στο μοντέλο αυτό είναι αυτές για τις οποίες η έκφραση παίρνει τιμή true.

2.3.3 Σύνταξη

Η βασική σύνταξη της γλώσσας για να εκφράσουμε μια ιδιότητα στο prism (prop), δίνεται από την ακόλουθη γραμματική:

```
prop ::= true | false |
         expr |
         !prop |
         prop & prop |
         prop | prop |
         prop => prop |
         P bound [ pathprop ] |
         S bound [ prop ]

bound ::= >=p | >p | <=p | <p

pathprop ::= X prop |
             prop U prop |
             prop U time prop

time ::= >=t | <=t | [t,t]
```

- expr είναι μια έκφραση σε γλώσσα prism που αντιστοιχεί σε μια Boolean τιμή.
- p είναι μια έκφραση σε γλώσσα prism που αντιστοιχεί σε μεταβλητή τύπου double στο διάστημα [0,1]

- t είναι μια έκφραση σε γλώσσα prism που αντιστοιχεί σε μη αρνητικό ακέραιο ή double.

Μια ιδιότητα αξιολογείται σύμφωνα με μια κατάσταση του μοντέλου. Για την παραπάνω γραμματική όλες οι ιδιότητες αντιστοιχούν σε Boolean τιμές. Για κάθε κατάσταση του μοντέλου, μια ιδιότητα είναι είτε true είτε false για αυτήν την κατάσταση. Ισοδύναμα μπορεί να θεωρηθεί πως η ιδιότητα ισχύει ή δεν ισχύει για αυτήν την κατάσταση. Η σημασιολογία είναι η ίδια όπως και στην προτασιακή λογική:

- true αν είναι true σε όλες τις καταστάσεις
- false αν είναι false σε όλες τις καταστάσεις
- expr είναι true αν η έκφραση expr είναι true
- !prop είναι true αν δεν είναι true η prop
- prop1 & prop2 είναι true αν και η prop1 και η prop2 είναι true
- prop1 | prop2 είναι true αν η prop1 είτε η prop2 είναι true
- prop1 => prop2 είναι αληθής αν η prop1 παράγει την prop2

2.3.4 Ο Τελεστής P

Δύο βασικοί τελεστές για να ορίζονται ιδιότητες είναι ο τελεστής P και ο τελεστής S. Ο τελεστής P μπορεί να χρησιμοποιηθεί και στα τρία είδη μοντέλων και επιτρέπει την εξαγωγή συμπερασμάτων για την πιθανότητα να παρατηρηθεί μια συγκεκριμένη συμπεριφορά στο μοντέλο. Γενικά: `P bound[ pathprop ]` είναι true για μια κατάσταση s, αν η πιθανότητα να ικανοποιείται η ιδιότητα pathprop ικανοποιεί το όριο bound. Ένα τυπικό παράδειγμα είναι το εξής:
`P>0.98[ pathprop]`, αυτό σημαίνει πως η πιθανότητα να ικανοποιείται η `pathprop` ξεπερνά το 0.98.

Για ένα DTMC η μέτρηση της πιθανότητας για ένα σύνολο μονοπατιών που ξεκινούν από μια κατάσταση s, είναι σαφώς ορισμένη. Επίσης για ένα CTMC η πιθανότητα αυτή μπορεί να οριστεί. Αντιθέτως για ένα MDP, η μέτρηση της πιθανότητας μπορεί μόνο να προσεγγιστεί, αφού όλα τα στοιχεία μη-ντετερμινισμού έχουν απομακρυνθεί. Ως εκ τούτου η πραγματική ερμηνεία του :
`P bound [ pathprop ]` ,είναι η πιθανότητα ώστε η ιδιότητα pathprop να ικανοποιείται από τα μονοπάτια που ξεκινούν από την αρχική κατάσταση s και να βρίσκουν το όριο bound για όλες τις πιθανές μη-ντετερμινιστικές αποφάσεις. Αυτό σημαίνει ότι, για ένα MDP, οι ιδιότητες που χρησιμοποιούν τον τελετή P, ουσιαστικά αναφέρονται στην ελάχιστη και μέγιστη πιθανότητα, από όλες τις μη-ντετερμινιστικές αποφάσεις ,ώστε να παρατηρείται μια συμπεριφορά. Αυτό εξαρτάται από ένα όριο που συνοδεύει τον τελεστή P. Ένα κατώτερο όριο(> ή >=) αναφέρεται σε ελάχιστες πιθανότητες και ένα ανώτερο όριο(< ή <=) σε μέγιστες πιθανότητες.

2.3.5 Ιδιότητες Μονοπατιών

Όπως αναφέρθηκε προηγουμένως στην παράγραφο «Σύνταξη», υπάρχουν τρείς διαφορετικοί τύποι ιδιοτήτων μονοπατιών που μπορούν να οριστούν στον τελεστή P.

- next : X (επόμενο)
- until : U (έως - μέχρι)
- bounded until : U time (εντός ορίων μέχρι)

Η ιδιότητα `X prop` είναι αληθής για κάποιο μονοπάτι εάν η prop είναι αληθής στην δεύτερή της κατάσταση. Παράδειγμα: `P<0.01 [ X y=1 ]` ,που είναι `true` για μια κατάσταση, εάν η πιθανότητα της έκφρασης `y=1` να είναι `true` στην επόμενη κατάσταση, είναι μικρότερη του 0.01.

Η ιδιότητα `prop1 U prop2` είναι αληθής για ένα μονοπάτι, εάν η `prop2` είναι αληθής για κάποια κατάσταση και η `prop1` είναι αληθής σε όλες τις προηγούμενες καταστάσεις. Μια κοινή εφαρμογή για αυτού του είδους την ιδιότητα είναι όταν η `prop1` είναι true. Ένα τυπικό παράδειγμα είναι το εξής: `P>0.5 [ true U z=2 ]` ,το οποίο είναι αληθές σε μια κατάσταση, εάν η πιθανότητα του z να είναι ίσο με 2 είναι μεγαλύτερη του 0.5 .

Οι ιδιότητες του τύπου bounded until είναι μια γενίκευση των ιδιοτήτων until, όπου ένα όριο προστίθεται ενώ το δεύτερο όρισμα (`prop2`) πρέπει να ικανοποιείται. Λόγω του ότι τα DTMC's και τα MDP's προχωρούν με διακριτά βήματα χρόνου, εν αντιθέσει με τα CTMC's που είναι μοντέλα συνέχουν χρόνου, οι δύο περιπτώσεις αντιμετωπίζονται με διαφορετικό τρόπο από τον τελεστή.

Σε μια ιδιότητα τύπου `"bounded until"` με `prop1 U time prop2` που αφορούν DTMC ή MDP, το τμήμα του ορισμού που αφορά τον χρόνο πρέπει να είναι της μορφής `"<=t"`, όπου `t` είναι μια έκφραση prism που αντιστοιχεί σε έναν μη αρνητικό ακέραιο. Η ιδιότητα ικανοποιείται για ένα μονοπάτι αν η `prop2` γίνει αληθής εντός κάποιων χρονικών βημάτων και η `prop1` είναι αληθής σε όλα τα βήματα πριν από το σημείο όπου η `prop2` γίνεται αληθής. Ένα παράδειγμα είναι το: `P>=0.98 [ true U<=7 y=4 ]` ,που είναι αληθές σε μια κατάσταση αν η πιθανότητα του υ να είναι ίσο με 4, εντός 7 χρονικών βημάτων, είναι μεγαλύτερη ή ίση από 0.98 .

Σε μια ιδιότητα τύπου `"bounded until"` με `prop1 U time prop2` που αφορούν CTMC, το τμήμα του ορισμού που αφορά τον χρόνο μπορεί να είναι οποιοδήποτε από τα παρακάτω: `>=t, <=t η [t1,t2]`, όπου `t,t1` και `t2` είναι εκφράσεις prism που αντιστοιχούν σε μη αρνητική μεταβλητή τύπου double και με `t1<=t2`. Το μονοπάτι ικανοποιεί την ιδιότητα αν η prop2 είναι αληθής σε χρόνο που η `prop1` είναι true, αλλά και σε όλους τους προηγούμενους χρόνους πριν τη στιγμή αυτή. Για παράδειγμα: `P>0 [ y<=1 U[5.5,6.5] y>1 ]` ,μεταφράζεται ως η πιθανότητα του y να ξεπεράσει το 1 σε χρόνο μεταξύ του 5.5 και 5.6 είναι μεγαλύτερη του μηδέν. Με την χρήση του τελεστή `"bounded until"` μπορούμε να αναφερθούμε σε μια συγκεκριμένη χρονική στιγμή, όπως εδώ: `P<0.01 [ true U[10,10] y=6 ]`.

2.3.6 Ο Τελεστής S

Ο τελεστής S χρησιμοποιείται όταν θέλουμε να κάνουμε αναφορά σε μια σταθερή κατάσταση συμπεριφοράς του μοντέλου, για παράδειγμα η συμπεριφορά του μοντέλου στη μακρά διάρκεια προσομοίωσης. Παρόλο που εξ ορισμού ο παραπάνω τελεστής θα μπορούσε να εφαρμοστεί και στα τριών ειδών μοντέλα, το prism υποστηρίζει τον τελεστή μόνο για μοντέλα τύπου CTMC προς το παρόν. Για παράδειγμα η ιδιότητα `S bound [ prop ]` ,είναι αληθής για μια κατάσταση s ενός CTMC, όταν ξεκινώντας από την s , η πιθανότητα στην μακρά πορεία της προσομοίωσης να βρεθεί σε μια κατάσταση που ικανοποιεί την ιδιότητα `prop`, είναι εντός του ορίου `bound`. Για παράδειγμα: `S<0.05 [ queue_size / max_size`

`> 0.75 ]` ,που ερμηνεύεται ως την πιθανότητα (long-run probability) της ουράς να είναι άνω του 75% γεμάτη, είναι μικρότερη από 0.05 .

2.3.7 Ικανοποίηση Ιδιοτήτων

Όλες οι ιδιότητες που περιγράφτηκαν στην προηγούμενες ενότητες αντιστοιχούν σε μια τιμή bool. Αν και η ικανοποίηση μιας ιδιότητας ορίζεται για μια κατάσταση συγκεκριμένα, όταν αναλύεται ένα μοντέλο το PRISM θεωρεί την ιδιότητα αληθή, αν ικανοποιείται σε όλες τις καταστάσεις του μοντέλου και ψευδή στην αντίθετη περίπτωση.

Για να ελεγχθεί αν μια ιδιότητα ικανοποιείται σε ένα υποσύνολο καταστάσεων, για παράδειγμα τις αρχικές καταστάσεις, οι ιδιότητες μπορούν να έχουν ένα πρόθεμα κάποιας συγκεκριμένης σημασίας. Για παράδειγμα:

```
"init" => P>=1 [ true U leader_elected=true ]
```

2.3.8 Καθορίζοντας την Πραγματική Πιθανότητα

Είναι χρήσιμο να γνωρίζουμε την πραγματική πιθανότητα να παρατηρηθεί κάποια συγκεκριμένη συμπεριφορά στο μοντέλο και όχι μόνο αν κάποια πιθανότητα ξεπερνά ένα όριο ή όχι. Το PRISM επιτρέπει ιδιότητες του παρακάτω τύπου:

```
P=?[...]
S=?[...]
```

Αυτές οι πιθανότητες επιστρέφουν μια αριθμητική τιμή ως αποτέλεσμα. Πρέπει να σημειωθεί πως το όριο της πιθανότητας που εφαρμόζεται σε τελεστές P ή S μπορεί να αντικατασταθεί με « =? », εάν είναι έσχατος τελεστής της ιδιότητας που εμφανίζεται, αλλιώς η σημασιολογία δεν ορίζεται σαφώς. Για ένα MDP οι πιθανότητες μπορούν να υπολογιστούν από όταν ο μη-ντετερμινισμός έχει επιλυθεί. Έτσι το PRISM υπολογίζει την ελάχιστη ή μέγιστη πιθανότητα να ικανοποιείται η ιδιότητα ενός μονοπατιού καταστάσεων, ποσοτικοποιώντας όλες τις πιθανές λύσεις (για παράδειγμα καλές και κακές λύσεις). Έτσι ένα MDP μπορεί να έχει δύο πιθανούς τύπους ιδιοτήτων:

```
Pmin=? [ ... ]
Pmax=? [ ... ]
```

Όλοι αυτοί οι τελεστές επιστρέφουν μια αριθμητική τιμή. Στην απλούστερη περίπτωση όπου το μοντέλο έχει μια μοναδική αρχική κατάσταση, η αριθμητική τιμή που επιστρέφει είναι αυτή που αντιστοιχεί σε αυτήν την κατάσταση. Για παράδειγμα: `P=? [ true U x=5&y=5 ]`, επιστρέφει την πιθανότητα του: «από την αρχική κατάσταση, το μοντέλο να μεταβεί σε κατάσταση όπου ικανοποιούνται τα εξής: `x=5` και `y=5`».

Είναι επίσης πιθανό να βρούμε την πιθανότητα για μια «κρίσιμη θέση» ορίζοντας μια ατομική πρόταση αληθή μόνο σε αυτή την κατάσταση εντός αγκυλών «{}». Αυτό είναι γνωστό ως φίλτρο. Για παράδειγμα: `P=? [ true U x=5&y=5 {x=1&y=2} ]`

Η παραπάνω πρόταση ερμηνεύεται ως: «η πιθανότητα από μια κατάσταση `(x=1,y=2)`, να φτάσει το μοντέλο σε κατάσταση `(x=5,y=5)`». Είναι πιθανό η έκφραση στο φίλτρο να ικανοποιεί περισσότερες της μίας καταστάσεις. Σε αυτή την περίπτωση εξ ορισμού η πρώτη τέτοια κατάσταση επιλέγεται και το PRISM εμφανίζει μια προειδοποίηση πως το φίλτρο ικανοποιεί πολλές καταστάσεις.

Η ίδια προσέγγιση εφαρμόζεται αν δεν υπάρχει φίλτρο αλλά το μοντέλο έχει πολλές αρχικές καταστάσεις. Αν το φίλτρο δεν ικανοποιείται από καμία κατάσταση τότε εμφανίζεται ένα μήνυμα λάθους.

Μπορεί επίσης να βρεθεί η μέγιστη και ελάχιστη τιμή από ένα σύνολο τιμών. Για παράδειγμα το παρακάτω:

```
P=? [ true U x=5&y=5 {y=2}{min} ]
P=? [ true U x=5&y=5 {y=2}{max} ]
```

που επιστρέφει την ελάχιστη και μέγιστη πιθανότητα αντίστοιχα να φτάσει το μοντέλο σε κατάσταση `x=5` και `y=5` από όλες τις καταστάσεις που ικανοποιούν το `y=2`.

Τέλος τα φίλτρα μπορούν να χρησιμοποιηθούν για ιδιότητες που χρησιμοποιούν τελεστές P και S. Για παράδειγμα: `P>0.5 [true U x=5&y=5 {y=2}]`, επιστρέφει true ή false αν η ιδιότητα `x=5` και `y=5` ικανοποιείται για όλες τις καταστάσεις όπου `y=2`.

2.3.9 Αρχεία Ιδιοτήτων

Τα αρχεία που περιέχουν ιδιότητες μπορούν επίσης να περιέχουν σταθερές. Για παράδειγμα:

```
const int k = 7;
const double T = 9.5;
const double p = 0.01;
P<p [ true U<=T x=k ]
```

Οι σταθερές αυτές μπορούν και να μην έχουν κάποια συγκεκριμένη τιμή εξ ορισμού αλλά να πάρουν αργότερα κάποιο εύρος τιμών χρησιμοποιώντας ένα πείραμα.

Τα όρια των πιθανοτήτων για τους τελεστές P και S και τα άνω και κάτω όρια για τον τελεστή U μπορεί να είναι σταθερές. Ακόμη οι ιδιότητες μπορούν να αναφέρονται σε σταθερές που έχουν οριστεί προηγουμένως στο μοντέλο (το αρχείο μοντέλου).

Ένα ακόμα χαρακτηριστικό των αρχείων ιδιοτήτων είναι τα labels. Αυτά χρησιμοποιούνται με τον ίδιο τρόπο και για τον ίδιο σκοπό με τα formulas που χρησιμοποιούνται στο αρχείο μοντέλου. Τα labels είναι ένας τρόπος για τον ορισμό και επαναχρησιμοποίηση ατομικών προτάσεων. Έχουν δύο διαφορές με τα formulas. Πρώτον πρέπει να είναι τύπου Boolean και επίσης ορίζονται χρησιμοποιώντας εισαγωγικά (“…”) όπως φαίνεται στο παρακάτω παράδειγμα:

```
Label "safe" = temp <=100 | alarm=true ;
label "fail" = temp>100 & alarm=false ;
P>=0.99 [ "safe" U "fail" ]
```

Δύο ιδιαίτερες περιπτώσεις είναι τα `"init"` και `"deadlock"` labels που ορίζονται πάντα. Έχουν τιμή true στις αρχικές καταστάσεις του μοντέλου και σε καταστάσεις όπου εμφανίζεται deadlock (και διορθώθηκαν με εισαγωγή self-loop) αντίστοιχα. Το πρώτο έχει χρησιμότητα μόνο για να διαπιστωθεί αν μια ιδιότητα είναι αληθής στις αρχικές καταστάσεις. Για παράδειγμα:

```
"init" => P>=0.99 [ "safe" U "fail" ]
```

Τα αρχεία ιδιοτήτων μπορούν να περιέχουν οποιονδήποτε αριθμό από ιδιότητες. Αποθηκεύονται με την προέκταση .pctl αν το μοντέλο είναι τύπου dtmc ή MDP, και .csl για μοντέλα τύπου CTMC.

2.3.10 Ορισμός Ιδιοτήτων Κόστους και Ανταμοιβών

Τα μοντέλα του prism μπορούν να περιέχουν πληροφορία για ανταμοιβές και κόστος (Το εργαλείο δεν διαχωρίζει το κόστος από την ανταμοιβή και ο διαχωρισμός γίνεται έπειτα στην ποιοτική ανάλυση των αποτελεσμάτων). Το εργαλείο μπορεί να αναλύσει ιδιότητες που σχετίζονται με αναμενόμενες τιμές από αυτές τις ανταμοιβές. Αυτό γίνεται με την χρήση του τελεστή R που δουλεύει με παρόμοιο τρόπο με τους τελεστές P και S. Για παράδειγμα: `R bound [ rewardprop ]`, όπου το bound παίρνει την μορφή form <r, <=r, >r or >=r για r που αντιστοιχεί σε μη αρνητικό double ή ερώτημα πραγματικής τιμής. Για παράδειγμα: `R query [ rewardprop ]`, όπου query είναι =? για ένα DTMC ή CTMC και min=? ή max=? για MDP.
Υπάρχουν τέσσερεις τύποι ιδιοτήτων reward:

- "reachability reward": `F prop`
- "cumulative reward" : `C<=t`
- "instantaneous reward" : `I=t`
- "steady-state reward" : `S`.

2.3.11 Προσεγγιστικές Ιδιότητες Κόστους

Οι ιδιότητες τύπου προσεγγιστικού κόστους (Reachability reward) σχετίζουν μια ανταμοιβή με κάθε μέρος του μοντέλου. Συγκεκριμένα, αναφέρονται σε ανταμοιβή που έχει μαζευτεί κατά μήκος ενός μονοπατιού καταστάσεων μέχρι να φτάσει ένα συγκεκριμένο όριο. Ο τρόπος που οι ανταμοιβές μαζεύονται εξαρτάται από το μοντέλο. Για ένα MDP και ένα DTMC μοντέλο, η συνολική ανταμοιβή που μαζεύεται είναι ίσο με το άθροισμα των ανταμοιβών των καταστάσεων του μονοπατιού, συν τις ανταμοιβές των μεταβάσεων μεταξύ των καταστάσεων του μονοπατιού. Για ένα μοντέλο CTMC το σύνολο των ανταμοιβών υπολογίζεται με παρόμοιο τρόπο μόνο που οι ανταμοιβές των καταστάσεων μεταφράζονται ως ο ρυθμός με τον οποίο τα rewards αθροίζονται σε κάθε κατάσταση. Για παράδειγμα αν έχουν περάσει t μονάδες χρόνου σε κατάσταση με κόστος-ανταμοιβή r, τότε το κόστος που έχει αθροιστεί στην κατάσταση αυτή είναι r x t. Έτσι το συνολικό κόστος είναι το άθροισμα όλων αυτών των γινομένων συν τα κόστη για τις μεταβάσεις μεταξύ των καταστάσεων.

Η ιδιότητα F που σχετίζεται με reward (`F prop`) αναφέρεται σε ένα κόστος που έχει μαζευτεί κατά μήκος ενός μονοπατιού μέχρι να ικανοποιείται η συνθήκη prop. Τα rewards μαζεύονται όπως περιγράφτηκε παραπάνω. Τα κόστη της κατάστασης prop που ικανοποιεί το ερώτημα pctl, δεν υπολογίζονται στο τελικό αποτέλεσμα. Στην περίπτωση που δεν υπάρχει μονοπάτι που φτάνει στην κατάσταση prop το αποτέλεσμα είναι infinity. Παρακάτω φαίνεται ένα τυπικό ερώτημα της ιδιότητας F.
`R<=9.5 [F z=2]` ,το οποίο είναι αληθές αν το κόστος που μαζεύεται κατά μήκος του μονοπατιού έως την κατάσταση `z=2` είναι μικρότερο του 9.5.

2.3.12 Συσσωρευτικές Ιδιότητες Κόστους

Η ιδιότητα τύπου συσσωρευτικού κόστους επίσης αντιστοιχούν ένα κόστος με κάθε μονοπάτι του μοντέλου αλλά μέχρι ένα ορισμένο χρονικό σημείο. Η ιδιότητα `C<=t` αντιστοιχεί στο κόστος που έχει μαζευτεί κατά μήκος ενός μονοπατιού μέχρι `t` μονάδες χρόνου να έχουν περάσει. Για μοντέλα τύπου DTMC και MDP το `t` πρέπει να είναι ένας ακέραιος. Για μοντέλα τύπου CTMC το t μπορεί να είναι double.

2.3.13 Στιγμιαίες Ιδιότητες Κόστους

Η ιδιότητα αυτού του τύπου αναφέρεται σε κάποια ανταμοιβή του μοντέλου σε συγκεκριμένη χρονική στιγμή. Η ιδιότητα `I=t` συσχετίζει ένα μονοπάτι με την ανταμοιβή μιας κατάστασης του μονοπατιού όταν ακριβώς t μονάδες χρόνου έχουν περάσει.

Για μοντέλα τύπου DTMC και MDP το t πρέπει να είναι ένας ακέραιος. Για μοντέλα τύπου CTMC το `t` μπορεί να είναι double. Η ιδιότητα: `R<4.4 [ I=100]` ,σημαίνει ότι μετά από 100 μονάδες χρόνου ακριβώς, η ανταμοιβή της κατάστασης που βρίσκεται το μοντέλο πρέπει να είναι μικρότερο του 4.4.

2.3.14 Steady-State Ιδιότητες Κόστους

Σε αντίθεση με τους προηγούμενους τρεις τύπους ιδιοτήτων δεν σχετίζονται με μονοπάτια, αλλά με κόστος στη διάρκεια της εκτέλεσης του προγράμματος. Ένα τυπικό παράδειγμα αυτής της ιδιότητας είναι το `R < 0.7 [S]`,που αν υποτεθεί πως τα κόστη του μοντέλου σχετίζονται με κατανάλωση ενέργειας, τότε η παραπάνω ιδιότητα σημαίνει η κατανάλωση ενέργειας στην εκτέλεση του προγράμματος να μην ξεπερνά τις 0.7 μονάδες.

2.3.15 Επιλογή δομής reward

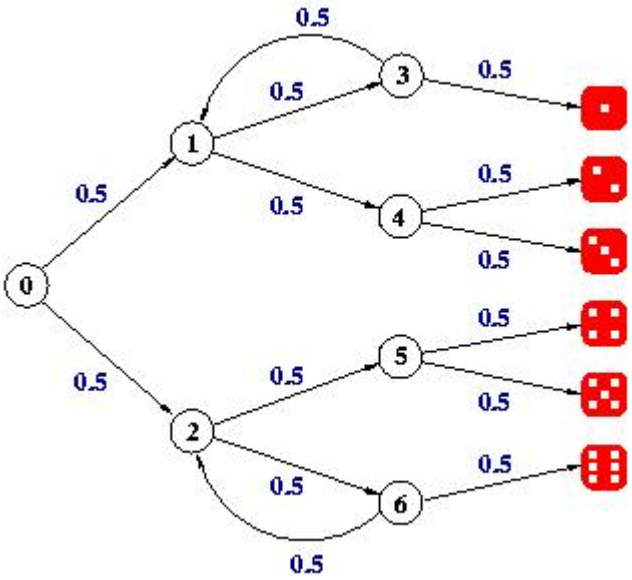
Σε περίπτωση που το μοντέλο έχει πολλαπλές δομές ανταμοιβών, για παράδειγμα `rewards num_failures …endrewards`, `rewards num_errors…endrewards` κλπ, τότε μπορεί να επιλεχθεί η επιθυμητή δομή μόνο και να ελεγχθεί μια ιδιότητα για αυτή μόνο. Για παράδειγμα: `R{"num_failures"}=? [C<=10.0]`.

2.4 Ο αλγόριθμος του Knuth

Στη συνέχεια παρατίθεται ένα παράδειγμα για να παρουσιαστούν στην πράξη η γλώσσα και η pctl λογική. Η ανάλυση του παραδείγματος έγινε σύμφωνα με το [8].

2.4.1 Εισαγωγή

Το παράδειγμα αυτό είναι ένας απλός πιθανοκρατικός αλγόριθμος του Knuth που σύμφωνα με το [11] μιμείται ένα ζάρι με έξι πλευρές με ένα δίκαιο νόμισμα. Παρακάτω δίνεται μια γραφική αναπαράσταση του αλγορίθμου:



Εικόνα 2.4 γραφική παράσταση του αλγορίθμου

Η ιδέα του αλγορίθμου είναι η εξής: Ξεκινώντας από το βήμα 0 (ο κύκλος με την τιμή 0), σε κάθε βήμα ένα νόμισμα στρέφεται και υπάρχει πιθανότητα 50% να πάρει οποιαδήποτε από τις δύο τιμές (κεφαλή ή γράμμα). Ο αλγόριθμος τερματίζει όταν έχουμε μια από τις τιμές του ζαριού σαν αποτέλεσμα. Ο κώδικας και η γραφική παράσταση είναι σύμφωνα με όσα αναφέρονται στο [8].
Ο κώδικας του PRISM είναι ο παρακάτω:

```
1    dtmc
2
3  module die
4
5          // local state
6          s : [0..7] init 0;
7          // value of the die
8          d : [0..6] init 0;
9
10         [] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=2);
11         [] s=1 -> 0.5 : (s'=3) + 0.5 : (s'=4);
12         [] s=2 -> 0.5 : (s'=5) + 0.5 : (s'=6);
13         [] s=3 -> 0.5 : (s'=1) + 0.5 : (s'=7) & (d'=1);
14         [] s=4 -> 0.5 : (s'=7) & (d'=2) + 0.5 : s'=7 & (d'=3);
15         [] s=5 -> 0.5 : (s'=7) & (d'=4) + 0.5 : (s'=7) & (d'=5);
16         [] s=6 -> 0.5 : (s'=2) + 0.5 : (s'=7) & (d'=6);
17         [] s=7 -> (s'=7);
18
19 endmodule
20
```

Εικόνα 2.6: ο κώδικας του prism για τον αλγόριθμο του Knuth

2.4.2 Ανάλυση κώδικα

Στην πρώτη γραμμή δηλώνεται ότι το μοντέλο είναι τύπου dtmc. Στην τρίτη γραμμή ξεκινά ο ορισμός του module με το όνομα `die` και το σώμα της δομής αυτής εκτείνεται ως την γραμμή 19. Στην γραμμή 6 δηλώνεται η μεταβλητή s που δείχνει το βήμα του αλγορίθμου που εκτελείται (στο παράδειγμα σε ποιον κύκλο του διαγράμματος βρισκόμαστε) και στην 8 μεταβλητή `d` που δείχνει την τιμή του ζαριού

(όταν `d=0` δεν έχουμε τιμή ζαριού). Στην δέκατη γραμμή η μεταβλητή `s` από την κατάσταση `s=0`, η μεταβλητή `s` είτε θα πάρει τιμή 1 με πιθανότητα 0.5 είτε τιμή 2 με πιθανότητα 0.5. Έπειτα και έως την 17 γραμμή η λειτουργία των εντολών είναι πανομοιότυπη με αυτή της εντολής 10.

2.4.3 Ιδιότητες που ελέγχονται

Παρακάτω φαίνεται η ιδιότητα που θα ορίσουμε για το μοντέλο:

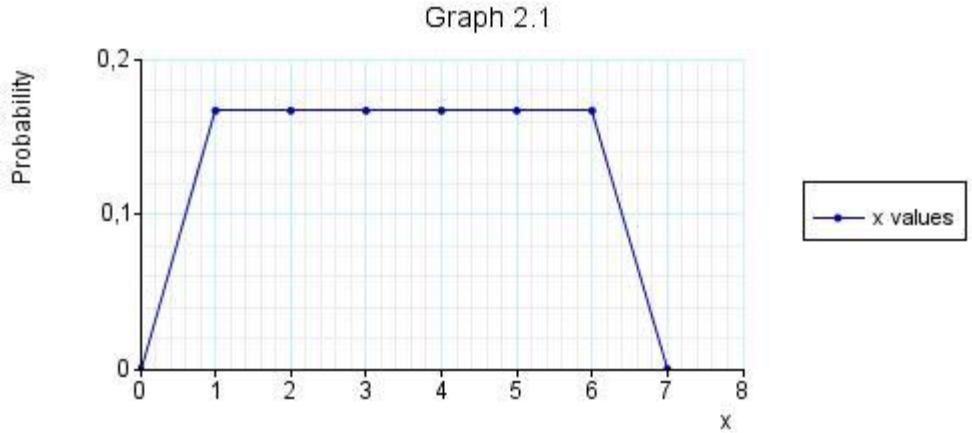
```
Const int x;
P=? [ true U s=7 & d=x ]
```

Η ιδιότητα είναι της μορφής: `P=? [ true U phi]` ,που σημαίνει «ποια είναι η πιθανότητα από την αρχική κατάσταση να φτάσουμε σε κατάσταση που να ικανοποιείται το `phi`». Στις ιδιότητα του μοντέλου ορίζεται και μια σταθερά τύπου int που χρησιμεύει για να διενεργηθεί το πείραμα.

2.4.4 Αποτελέσματα

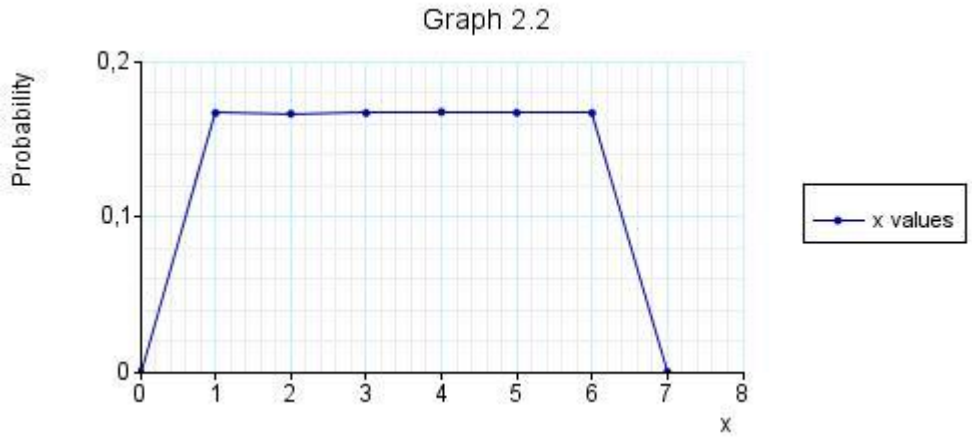
Παρακάτω φαίνονται τα αποτελέσματα που δίνει το prism με την εκτέλεση πειράματος και για `x=4`:
με μέθοδο verification, P=0.16666
με μέθοδο simulation, P=0.16711
Παρακάτω φαίνονται οι γράφοι που σχηματίζει το prism με μέθοδο verification,

α) αρχική κατάσταση `x=0` ,τελική `x=7` και βήμα 1



Γράφος 2.1: Γράφος πειράματος με μέθοδο verification

β) με μέθοδο simulation, αρχική κατάσταση `x=0` ,τελική `x=7` και βήμα 1



Γράφος 2.2: Γράφος πειράματος με μέθοδο simulation

ΚΕΦΑΛΑΙΟ ΙΙΙ

3. ΑΝΑΛΥΣΗ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ HIP

3.1 Εισαγωγή

Το HIP (Host Identity Protocol) είναι ένας μηχανισμός ασφαλείας και διευθυνσιοδότησης που έχει οριστεί από την IETF και αναπτύχθηκε από τον Moskowitz. Αργότερα ο Tuomas Aura, ο Aarthi Nagarajan και Andrei Gurtov βρήκανε ορισμένες τρύπες ασφαλείας του πρωτοκόλλου και προτείνανε λύσεις για την βελτίωση του [11].

Το internet αποτελείται από δύο βασικά στοιχεία : Τις IP (Internet Protocol) διευθύνσεις και τα ονόματα DNS (Domain Name Service) [16]. Αυτά τα δύο στοιχεία έχουν έναν αριθμό από χαρακτηριστικά και στοιχεία που έχουν δώσει στο internet την σημερινή του μορφή. Έχουν επίσης και έναν αριθμό από αδυναμίες. Η σημασιολογική υπερφόρτωση και οι λειτουργικές επεκτάσεις έχουν περιπλέξει πολύ το IP και το DNS. Το HIP γεμίζει ένα σημαντικό κενό ανάμεσα σε αυτά τα δύο.

Βασικός στόχος του HIP είναι ο διαχωρισμός των host identifiers από τις τοποθεσίες στο Ipv4 και Ipv6 internet [13]. Το HIP προσθέτει ένα ακόμη επίπεδο μεταξύ του επιπέδου μεταφοράς (transport) και δικτύου (network) στη στοίβα του TCP/IP. Το νέο αυτό επίπεδο αντιστοιχίζει αναγνωριστές σταθμών (host) σε τοποθεσίες δικτύου και αντίστροφα. Έτσι επιτυγχάνεται ο βασικός στόχος του HIP που είναι ο διαχωρισμός των αναγνωριστών από τις τοποθεσίες.

Το HI (Host Identity) στο HIP είναι ένα δημόσιο κλειδί. Αυτού του τύπου ο αναγνωριστής πιστοποιεί τον εαυτό του υπό την έννοια πως μπορεί να χρησιμοποιηθεί για πιστοποίηση υπογραφών χωρίς να χρειάζεται πρόσβαση σε πιστοποιητικά ή δομές πιστοποίησης αυθεντικότητας βάση δημοσίου κλειδιού. Το αναγνωριστικό σταθμού (host identity) συνήθως παριστάνεται με το Host Identity Tag (HIT), που είναι το αποτέλεσμα της εφαρμογής μιας 128 bits hash συνάρτησης στο HI. Οι διευθύνσεις IPv4 και IPv6 στο HIP είναι γνήσιες διευθύνσεις.

Το πρωτόκολλο αποτελείται από τρία βασικά μέρη. Πρώτα οι δύο σταθμοί που επιθυμούν να επικοινωνήσουν εγκαθιδρύουν κλειδιά συνόδου με τη βάση συναλλαγής (base exchange) του HIP. Σύμφωνα με αυτή τα πακέτα προστατεύονται χρησιμοποιώντας τυπικά IPsec ESP. Τα HI χρησιμοποιούνται για να δημιουργήσουν τις απαραίτητες σχέσεις ασφαλείας (Security Associations, SAs) και για να πιστοποιήσουν τους σταθμούς. Όταν χρησιμοποιείται το IPSec τότε τα πακέτα IP δεν διαφέρουν σε καμία περίπτωση από τα κλασικά πακέτα IP που προστατεύονται με IPSec. Τέλος υπάρχει ένας μηχανισμός επαναδιευθυνσιοδότησης για να υποστηρίξει την αλλαγή των IP διευθύνσεων με ευκινησία και multi-homing.

Η ευκινησία του HIP περιλαμβάνει την αλλαγή των διευθύνσεων IP (με οποιαδήποτε μέθοδο) για κάθε μία οντότητα. Έτσι ένα σύστημα θεωρείται κινητό αν οι IP διευθύνσεις του μπορούν να αλλάζουν δυναμικά για οποιοδήποτε λόγο. Παρόμοια ένα σύστημα θεωρείται multi-homing αν έχει περισσότερες του ενός παγκόσμιες δρομολογήσιμες IP διευθύνσεις την ίδια στιγμή, δηλαδή ένας σταθμός έχει πολλά παράλληλα μονοπάτια επικοινωνίας που μπορεί να χρησιμοποιήσει. Το HIP ενώνει μαζί τις διευθύνσεις όταν πολλές IP διευθύνσεις αναφέρονται στην ίδια HI και αν κάποια διεύθυνση γίνει μη διαθέσιμη, ή αν μια πιο επιθυμητή γίνει διαθέσιμη, τότε οι υπάρχουσες σχέσεις μεταφοράς μπορούν εύκολα να μεταφερθούν σε κάποια άλλη διεύθυνση.

3.2 HIP Base Exchange

Το κύριο δομικό τμήμα των πρωτοκόλλων HIP είναι το base exchange. Χρησιμοποιείται για να εγκαθιδρύσει ένα ζεύγος IPsec συνδέσεων μεταξύ των δύο σταθμών. Το base exchange βασίζεται στην ανταλλαγή κλειδιού βάση του πρωτοκόλλου ανταλλαγής κλειδιού Diffie Hellman, αλλά υπάρχουν και ορισμένα ιδιαίτερα χαρακτηριστικά που σχετίζονται με την προστασία ενάντια του Dos-attack. Δεν χρειάζονται πιστοποιητικά για την απόδειξη της αυθεντικότητας καθώς τα HIT είναι αναπροσδιοριζόμενα [10].

Αρχικά ο initiator στέλνει ένα κενό μήνυμα I1 στον responder. Το μήνυμα αυτό ακολουθεί ένα μήνυμα R1 από τον responder. Όλα τα πακέτα περιέχουν την επιγραφή της ταυτότητας του initiator και του responder στην κεφαλή (HIT-I και HIT-R).

Ο responder πριν ακόμη πάρει το μήνυμα I1 προϋπολογίζει ένα μέρος του R1. Το προϋπολογισμένο R1 περιλαμβάνει το HIT-R, το κλειδί Diffie Hellman του responder, την ταυτότητα του σταθμού (host identity) HI-R του responder, τους προτεινόμενους κρυπτογραφικούς και IPsec αλγορίθμους για το υπόλοιπο του πρωτοκόλλου και ένα πεδίο Echo-Request. Το πεδίο αυτό περιέχει πληροφορίες που ο initiator επιστρέφει χωρίς αλλαγή στο μήνυμα I2. Είναι σημαντικό το μήνυμα R1 να στέλνεται χωρίς ο responder να χρειαστεί να δημιουργήσει κάποια κατάσταση πρωτοκόλλου. Το πεδίο Echo-Request μπορεί να χρησιμοποιηθεί για αποθήκευση δεδομένων χωρίς να χρειάζεται νέα κατάσταση πρωτοκόλλου. Έπειτα ο responder υπογράφει το μήνυμα. Τα πεδία HIT-I και puzzle είναι κενά μέχρι τη δημιουργία του προϋπολογισμένου R1 και συμπληρώνονται μετά τη λήψη του I1. Τα πεδία αυτά δεν προστατεύονται από την υπογραφή του responder.

Η παράμετρος puzzle στο μήνυμα R1 περιέχει ένα κρυπτογραφικό puzzle το οποίο ο initiator πρέπει να λύσει πριν στείλει το επόμενο πακέτο I2. Η βασική ιδέα είναι ότι ο initiator καλείται να εκτελέσει μια δαπανηρή ενέργεια brute-force προτού ο responder αναθέσει τους υπολογιστικούς του πόρους στο πρωτόκολλο ή δημιουργήσει μια κατάσταση πρωτοκόλλου.

Το puzzle έχει τρία στοιχεία: Το puzzle nonce I, την δυσκολία του puzzle K, την λύση J. Είναι ευκολότερο να εξηγηθεί το πως επαληθεύεται η λύση του puzzle. Πρώτα συνδέονται (concatenation) το I, τα HIT-I και HIT-R και η λύση J. Έπειτα υπολογίζεται το SHA-1 hash της παραπάνω σύνδεσης. Τέλος ελέγχεται αν όλα τα K χαμηλής τάξης bits του αποτελέσματος της hash συνάρτησης, είναι όλα μηδέν.
Ltrunc(SHA-1(I|HIT-I|HIT-R|J),K) = =0.

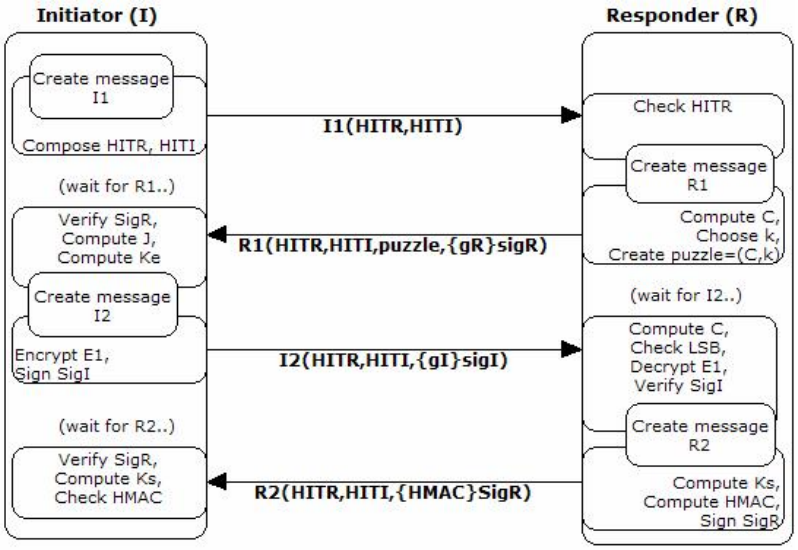
Ο initiator πρέπει να εκτελέσει μια brute-force αναζήτηση για τη λύση J, που παίρνει O(2κ) δοκιμές. Αντιθέτως ο responder μπορεί να επαληθεύσει την λύση του puzzle απλά υπολογίζοντας μια hash συνάρτηση.

Όταν λαμβάνει το R1, ο initiator ελέγχει αν έχει στείλει ένα αντίστοιχο I1 και επαληθεύει την υπογραφή του με το δημόσιο κλειδί HI-R. Εάν η υπογραφή είναι σωστή, τότε ο initiator λύνει το puzzle και δημιουργεί το μήνυμα I2. Το I2 περιέχει το puzzle και τη λύση του, το κλειδί Diffie Hellman του initiator, οι μετασχηματισμοί ESP του HIP που προτείνει και μια παράμετρο ασφαλείας για το Responder-to-Initiator IPSesc SA. Επίσης περιλαμβάνει το δημόσιο κλειδί του initiator (HI-I) κρυπτογραφημένο με το νέο κλειδί συνόδου και το Echo-Response.Το κλειδί συνόδου υπολογίζεται ως μια τιμή SHA-1 hash του μοιραζόμενου Diffie Hellman μυστικού κλειδιού Kij. Το μήνυμα καλύπτεται από μια υπογραφή.
KEYMAT_k=SHA-1(Kij,|sort(HIT-I|HIT-R)|k) for k=1,2,…

Όταν λαμβάνει το I2, ο responder επαληθεύει τη λύση του puzzle. Αν είναι σωστή, υπολογίζει τα κλειδιά συνόδου, αποκρυπτογραφεί το HI-I και επαληθεύει την

υπογραφή του I2. Έπειτα ο responder στέλνει το R2 που περιέχει το SPI για το Responder-to-Initiator IPSec SA, και ένα HMAC υπολογισμένο χρησιμοποιώντας ένα κλειδί συνόδου και μια υπογραφή.

Για τον initiator η συναλλαγή ολοκληρώνεται με την λήψη του R2 και την επαλήθευση του HMAC και της υπογραφής. Το HMAC επιβεβαιώνει την εγκαθίδρυση του κλειδιού συνόδου. Για τον responder η επαλήθευση του κλειδιού γίνεται με το πρώτο IPSec πακέτο που λαμβάνει και προστατεύεται με την νέα συμφωνημένη σχέση ασφαλείας.



Εικόνα 3.1 Σχηματική αναπαράσταση του πρωτοκόλλου HIP[14]

3.3 Σημασιολογία Πρωτοκόλλου

Ο παρακάτω πίνακας δίνει την ερμηνεία των συμβόλων που χρησιμοποιούνται για την περιγραφή του πρωτοκόλλου.[10]

ΣΥΜΒΟΛΟ	ΕΡΜΗΝΕΙΑ
HIT	Δεδομένο μήκους 128 bit που δημιουργήθηκε με εφαρμογή κρυπτογραφικής hash συνάρτησης στο HI.
HIT-I	Αναγνωριστικό σταθμού (host identity) initiator
HIT-R	Αναγνωριστικό σταθμού (host identity) responder
Responder	Η οντότητα που αποκρίνεται στις αιτήσεις του initiator
Initiator	Ο πελάτης, που εκκινεί το πρωτόκολλο
I	Το puzzle nonce
J	Η λύση του puzzle
K	Η δυσκολία του puzzle
Sig	Υπογραφή

SHA-1	Συνάρτηση hash
Echo-Request	Πεδίο του HIP μηνύματος
HMAC	MAC(Message authentication code) κρυπτογραφημένο με συνάρτηση hash
HIP BASE EXCHANGE	Κρυπτογραφικό πρωτόκολλο
ESP	Μια από τις πιο γνωστές μορφές IPSec
Diffie Hellman	Κρυπτογραφικό πρωτόκολλο εγκατάστασης διαμοιραζόμενου μυστικού κλειδιού
DNS	Μεταφράζει hostnames (π.χ. www.csd.auth.gr) σε διευθύνσεις IP
IP διεύθυνση	Μοναδική διεύθυνση κάθε σταθμού που χρησιμοποιεί για να αναγνωρίσει και να επικοινωνήσει με άλλους
TCP/IP	Σύνολο πρωτοκόλλων επικοινωνίας με τα πιο βασικά το TCP και IP. Πάνω σε αυτό το σύνολο πρωτοκόλλων τρέχει και το internet

Πίνακας 3.1 Σημασιολογία πρωτοκόλλου

3.4 Επίθεση άρνησης υπηρεσίας

Σύμφωνα και με το [14], επίθεση άρνησης υπηρεσίας ορίζεται ως η προσπάθεια ενός ή πολλών εισβολέων να αποτρέψουν νόμιμους χρήστες μιας υπηρεσίας από το να μπορούν να χρησιμοποιήσουν αυτήν την υπηρεσία.

Ως ανθεκτική ιδιότητα ενάντια σε επίθεση άρνησης υπηρεσίας ορίζεται ως η χαμηλή πιθανότητα για έναν εισβολέα να αποτρέψει νόμιμους χρήστες ενός πρωτοκόλλου από τη χρήση κάποιας υπηρεσίας του [13].

Η προστασία έναντι σε επίθεση άρνησης υπηρεσίας συνήθως βασίζεται σε δύο στρατηγικές. Είτε σε κάποιο cookie είτε σε επίλυση ενός puzzle όπου ο ένας από τους συμμετέχοντες του πρωτοκόλλου στέλνει ένα cookie (κάποια τιμή hash συνάρτησης που δεν μπορεί να αποκαλυφθεί) ή ένα puzzle στον άλλο συμμετέχων, με σκοπό την εγκαθίδρυση αμοιβαίας εμπιστοσύνης, συνήθως με την μορφή διαμοίρασης κάποιου κοινού μυστικού. Η βασική ιδέα είναι ότι ο ανταποκρινόμενος σε αυτό το cookie πρέπει να μην παράγει κάποια κατάσταση πρωτοκόλλου, αλλά να παραμείνει stateless, για να μην διενεργήσει ακριβές υπολογιστικά πράξεις. Υπάρχουν ωστόσο τρεις βασικές στρατηγικές για έναν εισβολέα στην παραπάνω περίπτωση [13]:

- Counterfeit: αποστολή πλαστών cookies και puzzles ή λύσεις puzzles.
- Time shifting: Ο εισβολέας είναι έτοιμος για την επίθεση υπολογίζοντας εσφαλμένα διαμοιραζόμενα μυστικά (είτε λύνοντας puzzles είτε χειριζόμενος τα cookies) με σκοπό να τα χρησιμοποιήσει σε μια μαζική επίθεση άρνησης υπηρεσίας.
- Επανάληψη μηνυμάτων (message replays): Ο εισβολέας στέλνει τα ίδια cookies ή λύσεις puzzle πολλές φορές.

3.5 Το Μοντέλο του Prism

Η βασική ιδέα της εργασίας είναι να αναδειχθεί μια αδυναμία του HIP πρωτοκόλλου με τη βοήθεια του εργαλείου prism. Αυτή παρουσιάζεται όταν ένας εισβολέας επιχειρήσει επίθεση άρνησης υπηρεσίας και ο τρόπος που μπορεί να εκμεταλλευτεί την αδυναμία του πρωτοκόλλου είναι να παρεμβληθεί στην επικοινωνία των δύο οντοτήτων, του initiator και του responder, και να κάνει αλλαγή στο puzzle του μηνύματος.

Η επικοινωνία γίνεται μέσω ανταλλαγής μηνυμάτων και κάθε συμμετέχων στο πρωτόκολλο βρίσκεται σε τρεις καταστάσεις, κατασκευής μηνύματος, επεξεργασίας μηνύματος και αποστολής μηνύματος. Τα μηνύματα που στέλνονται είναι τέσσερα, δύο από πλευράς initiator και δύο από πλευράς responder. Τα modules του πρωτοκόλλου είναι τέσσερα . Το πρώτο είναι το meso, που αντιπροσωπεύει το κανάλι επικοινωνίας που χρησιμοποιείται από τους συμμετέχοντες στο πρωτόκολλο για να ανταλλάσουν μηνύματα. Τα υπόλοιπα τρία είναι τα initiator, intruder και responder. Το module meso μπορεί να έχει τρεις καταστάσεις που καθορίζονται από τις τιμές που παίρνουν οι μεταβλητές `c1` και `c2`. Αν `c1=2` και `c2=2` τότε δεν μεταδίδεται κάποιο μήνυμα στο κανάλι. Αν `c1=1` και `c2=2` τότε ο initiator είναι αυτός που στέλνει μήνυμα ενώ αν `c1=2` και `c2=1` τότε είτε ο intruder είτε ο responder στέλνει μήνυμα. Δύο βασικές μεταβλητές σχετίζονται με την υλοποίηση του πρωτοκόλλου, η μεταβλητή `proc_limit` και `B`. Η πρώτη αναπαριστά τον αριθμό μηνυμάτων που μπορεί να επεξεργάζεται ταυτόχρονα ο initiator και η δεύτερη τον μέγιστο αριθμό μηνυμάτων που μπορεί να δεχθεί η ουρά μηνυμάτων προς εξυπηρέτηση. Επίσης η μεταβλητή `M` αντιπροσωπεύει τον αριθμό των διακριτών μηνυμάτων που αλλάζει ο εισβολέας κάθε φορά.

Σύμφωνα με το [13], η επίθεση που κάνει ο εισβολέας βασίζεται στο ότι το puzzle που καλείται να λύσει ο initiator δεν βρίσκεται στο προϋπολογισμένο και υπογεγραμμένο τμήμα του δεύτερου μηνύματος. Το puzzle παράγεται με βάση ένα τυχαίο nonce και μιας παραμέτρου k που εκφράζει την δυσκολία επίλυσης του puzzle. Η κατασκευή puzzle και το προϋπολογισμένο τμήμα του δευτέρου μηνύματος προστατεύουν τον πρωτόκολλο από επιθέσεις τύπου replay DoS attack αλλά το καθιστούν ευάλωτο στην επίθεση τύπου άρνησης υπηρεσίας που μοντελοποιείται.

Όλες οι ενέργειες του επιτιθέμενου ορίζονται μέσα στο module intruder, και στις πράξεις αυτές ανατίθεται κάποιο κόστος, όπως βέβαια και στις πράξεις που κάνει ο initiator και ο responder [14]. Σε όλες τις περιπτώσεις το κόστος αναφέρεται στο ίδιο αγαθό που μπορεί να είναι bandwidth, αποθηκευτική ή επεξεργαστική ισχύ κλπ.

Ο εισβολέας χρησιμοποιεί ένα καθορισμένο αριθμό από N μηχανές που χρησιμοποιούν νόμιμοι χρήστες και στις οποίες έχει αποκτήσει πρόσβαση. Αυτοί οι υπολογιστές καλούνται zombie υπολογιστές. Οι υπολογιστές αυτοί δημιουργούν λανθασμένα μηνύματα με puzzles προς λύση και μπορούν να οδηγήσουν στην δέσμευση όλων των υπολογιστικών πόρων του initiator [14].

Αναλυτικότερα, όταν ο initiator θελήσει να επικοινωνήσει με κάποιον responder με βάση το πρωτόκολλο HIP, τότε θα πρέπει να στείλει ένα αρχικό μήνυμα για να εκκινήσει την συνομιλία με τον responder. Ο intruder σε αυτή την φάση απλά επιβλέπει την αποστολή του πρώτου μηνύματος και δεν κάνει τίποτα παραπάνω από το να προωθήσει το μήνυμα του initiator και να περιμένει για την απάντηση του responder.

Έπειτα, σύμφωνα με το πρωτόκολλο ο responder απαντά στέλνοντας μήνυμα που περιέχει ένα puzzle προς λύση από τον initiator. Το puzzle δεν βρίσκεται εντός του υπογεγραμμένου μηνύματος και έτσι ο intruder μπορεί εύκολα να το κλέψει και να το αλλάξει. Αυτό συμβαίνει γιατί το πρωτόκολλο θέλει να ευνοήσει τον responder με την έννοια ότι δεν τον αναγκάζει να δεσμεύσει πολλούς υπολογιστικούς πόρους για την παραγωγή υπογραφών κλπ. Με άλλα λόγια ο initiator δεσμεύει περισσότερους πόρους από τον responder μέχρι και να λύσει το puzzle και αυτό γιατί η παραγωγή του puzzle έχει ελάχιστο κόστος ενώ η λύση του αρκετά μεγαλύτερο.

Εφόσον ο εισβολέας παραλάβει το puzzle, το αλλάζει μέσω της μεθόδου που περιγράφτηκε προηγουμένως και στέλνονται «ψεύτικα» μηνύματα προς λύση στον initiator. Εκείνος δεσμεύει τους πόρους του για να ικανοποιήσει τις όλο και περισσότερες εισερχόμενες αιτήσεις, μιας και ο εισβολέας στέλνει μεγάλο αριθμό μηνυμάτων. Έτσι προκαλείται η κατάρρευση του συστήματος του initiator και επομένως ο intruder επιτυγχάνει μια επίθεση άρνησης υπηρεσίας. Ο τρόπος που μοντελοποιούνται τα παραπάνω με γλώσσα prism αναλύονται παρακάτω.

ΚΕΦΑΛΑΙΟ IV

4. ΑΝΑΛΥΣΗ ΤΟΥ ΚΩΔΙΚΑ

4.1 Εισαγωγή

Στην ενότητα αυτή θα αναλυθεί ο κώδικας που γράφτηκε σε γλώσσα prism, για να μοντελοποιηθεί το πρωτόκολλο HIP που περιγράφτηκε σε προηγούμενη ενότητα. Ακολουθεί ο κώδικας χωρισμένος σε κομμάτια και περιγραφή της κάθε σειράς του.

```
1  //THIS IS A dtmc
2  dtmc
3  const double MAX_TIME=2000;
4  ////////////////////
5  //const int INIT_=10; //xronos gia thn apostolh kai thn epanapostolh tou prwtou mhnumatos
6  ////////////////////
7  const double GEN_I1 = 0.05;
8  const double PR_I1_GEN_R1=0.05; //gen/process I1
9  const double PR_R1_GEN_I2_k5=1;
10 const double PR_R1_GEN_I2_k10=1.35;
11 const double PR_R1_GEN_I2_k15=1.6;
12 const double PR_R1_GEN_I2_k20=13;
13 const double PR_R1_GEN_I2_k25=68;
14 const double PR_I2_GEN_R2=1.48;
15 const double PR_R2=0.05;
16 const double TRNSMT_=0.002;
17
18 const double PR_R1_GEN_SP_k5=0.9;
19 const double PR_R1_GEN_SP_k10=0.9;
20 const double PR_R1_GEN_SP_k15=1;
21 const double PR_R1_GEN_SP_k20=13;
22 const double PR_R1_GEN_SP_k25=20;
23
24 //THESE ARE THE FORMULAS
25
26 //no action over the channel
27 formula no_action = c1=2 & c2=2;
28 //initiator sending
29 formula sending_init = c1=1 & c2=2;
30 //responder sending
31 formula sending_resp = c1=2 & c2=1;
32 //exei dimiourgithei to message 1
33 formula message1 = HITi=1 & HITr_sent=1;
34 formula message1_int = HITi_stolen=1 & HITr_stolen=1;
35 formula message2 = HITi_received=1 & HITr=1 & puzzle=true & HITr=1 & gr=1 & HITi_received=1 & signr1=1;
36
37
38 //END OF FORMULAS
39
```

Εικόνα 4.1: ο ορισμός των μεταβλητών και των formulas

4.2 Ορισμός σταθερών - formulas

Στην δεύτερη γραμμή ορίζεται το είδος του μοντέλου σαν discrete time markov chain. Το μοντέλο είναι τύπου DTMC γιατί η επιλογή της μετάβασης μεταξύ των εκάστοτε καταστάσεων του μοντέλου γίνεται με βάση κάποια πιθανότητα (πιθανοκρατικά). Δεν επιλέχθηκε μοντέλο CTMC αφού η παράμετρος του χρόνου δεν μας αφορά. Από την Τρίτη έως την δέκατη έκτη γραμμή ορίζονται όλες οι σταθερές του μοντέλου που αφορούν τον χρόνο παραγωγής των μηνυμάτων από initiator και responder. Από την γραμμή 18 έως 22 φαίνονται οι χρόνοι επεξεργασίας του πρώτου μηνύματος του responder από τον intruder. Όλοι οι χρόνοι είναι σύμφωνα με το [12]. Από την γραμμή 27 έως και 35 ορίζονται οι εξής φόρμουλες:

- `no_action` ,ισχύει όταν το κανάλι επικοινωνίας είναι ελεύθερο
- `sending_init` ,είναι true όταν στέλνει μήνυμα ο initiator
- `sending_resp` ,αντίστοιχα για τον responder
- `message1, message2, message_int` ισχύουν όταν, το πρώτο και το δεύτερο μήνυμα είναι έτοιμα προς αποστολή και ο intruder έχει υποκλέψει το πρώτο μήνυμα, αντίστοιχα.

```
40 //THESE ARE THE CONSTANTS
41 const int M = 100;
42 //anw orio counter tou initiator
43 const int B = 400;
44 const int proc_limit = 200;
45 const int MAX = 200;
46
47 //aritmos minymatwn pou ftanoun sti monada xronoy
48 global a : [0..50] init 0;
49
50 global fail : [0..2] init 0;
51 //0 no failure
52 // 1 connection failure
53 // 2 dos attack failure
54
55 global puzzle_solution : bool init false;
56 global reject : bool init false;
57 global connection : bool init false;
58 //END OD CONSTANTS
59
60
61 module meso
62
63 //c1=1: no action towards responder
64 //c1=2: action towards responder
65 c1 : [1..2] init 2;
66 c2 : [1..2] init 2;
67
68 //MESSAGE 1
69 [m1]c1=2 & c2=2 -> c1'=1;
70 [finish_m1](c1=1 & c2=2 & s3=1) -> (c1'=2) & (c2'=2);
71 ///////////////
72
73 //MESSAGE 1, INTR->RESPONDER
74 [intr_1]no_action -> c1'=1;
75 [finish_intr_1]sending_init & s3=4-> c1'=2;
76 ////////////////
77
78 //MESSAGE 2
```

Εικόνα 4.2: Συνέχεια ορισμού μεταβλητών και το module meso

Στην γραμμή 41 έως 45 ορίζονται σταθερές ως integers οι οποίες αφορούν κατά σειρά, τον αριθμό των αλλαγών του puzzle που κάνει ο intruder, το άνω όριο της ουράς του initiator που δέχεται τις αιτήσεις που περιμένουν να εξυπηρετηθούν, το όριο των αιτήσεων που μπορούν να περιμένουν στην ουρά και θα εξυπηρετηθούν και τέλος το άνω όριο του αριθμού των εξυπηρετών που μπορούν να εξυπηρετούν παράλληλα.

Έπειτα η μεταβλητή a που ορίζεται στην 48 είναι τύπου global και αφορά τον αριθμό των μηνυμάτων που θα στέλνονται παράλληλα στον initiator από τον intruder και στην 50 η μεταβλητή `fail` έχει αρχική τιμή 0 και αποκτά τιμή 1 όταν υπάρξει connection failure και 2 όταν συμβεί η άρνηση υπηρεσίας.

Στην γραμμή 55 global η μεταβλητή `puzzle_solution` παίρνει την τιμή true εάν ο initiator λύσει σωστά το puzzle ενώ στην 56 η global μεταβλητή `reject` παίρνει την τιμή true εάν ο responder απορρίψει τον initiator, για παράδειγμα αν η λύση του puzzle είναι λάθος. Στην γραμμή 57 η μεταβλητή παίρνει την τιμή true εάν όλα τα βήματα του πρωτοκόλλου ολοκληρωθούν με επιτυχία.

4.3 Module Meso

Στην γραμμή 61 αρχίζει ο ορισμός του module meso, στην 65 και 66 δηλώνονται μεταβλητές που σχετίζονται με την κατάσταση του καναλιού. Η αρχική τιμή των μεταβλητών είναι 2. Αν c1 έχει την τιμή 1 τότε ο initiator είναι αυτός που στέλνει μήνυμα, ενώ αν `c2=1` τότε ο initiator είναι αυτός που θα λαμβάνει το μήνυμα.

Στις γραμμές 69 και 70 ορίζονται δύο εντολές που συγχρονίζονται με τα modules initiator και intruder και σχετίζονται με την αποστολή του πρώτου μηνύματος. Η πρώτη εντολή θέτει την τιμή 1 στην μεταβλητή `c1`, ενώ η δεύτερη την επαναφέρει στην αρχική της κατάσταση απελευθερώνοντας το κανάλι.

Έπειτα στις γραμμές 74 έως 95 ορίζονται εντολές με την ίδια λειτουργία όπως αυτή περιγράφτηκε για τις δύο προηγούμενες εντολές και η μόνη διαφορά τους είναι πως ενεργοποιούνται σε διαφορετικές φάσεις του πρωτοκόλλου δηλαδή όταν στέλνονται διαφορετικά μηνύματα.

```
78  //MESSAGE 2
79  [m2]no_action & message2 -> c2'=1;
80  [finish_m2]sending_resp & s1=4 -> (c1'=2) & (c2'=2);
81  ////////////////
82
83  //SPOOFED MESSAGE 2  INTR->INITIATOR
84  [spoof]no_action -> c2'=1;
85  [finish_spoof]sending_resp -> c2'=2;
86  ////////////////
87
88  //MESSAGE 3
89  [m3]no_action -> c1'=1;
90  [finish_m3]sending_init -> c1'=2;
91  ////////////////
92
93  //MESSAGE 4
94  [m4]no_action -> c2'=1;
95  [finish_m4]sending_resp -> c2'=2;
96  endmodule
97
98
99
100 module initiator
101 s1 : [1..7] init 1;
102 //states for initiator
103 //1: initial state
104 //2: prepare 1st message
105 //3: sending first message
106 //4: wait for responder's answer and send message3
107 HITi : [0..1] init 0;//initiator's host identity tag
108 HITr_sent : [0..1] init 0;//responder's host identity tag
109 counter : [0..B] init 0;//number of requests waiting to be answered
110 verify_sign : bool init false;//signature verification for received message 2
111 ex : [0..MAX] init 0;//indicates the proccessors
112 verify_HMAC : bool init false;//HMAC verification
113 msgs : [0..500] init 0;
114
115 // CREATE MESSAGE 1
116 []s1=1 -> s1'=2;
```

Εικόνα 4.3: έναρξη του module initiator

4.4 Module Initiator

Έπειτα στην γραμμή 100 ορίζεται η έναρξη του ορισμού του module initiator που υλοποιεί τις λειτουργίες που επιτελεί στο πρωτόκολλο η οντότητα που ξεκινά το πρωτόκολλο στέλνοντας το πρώτο μήνυμα.
Στην επόμενη γραμμή η μεταβλητή s1 χρησιμοποιείται για να υποδεικνύει την κατάσταση που βρίσκεται ο initiator. Αναλυτικότερα:

- `s1=1`, αρχική κατάσταση του initiator.
- `s1=2`, αρχίζει την κατασκευή του πρώτου μηνύματος.
- `s1=3`, το πρώτο μήνυμα είναι έτοιμο προς αποστολή και αποστέλλεται.
- `s1=4`, το πρώτο μήνυμα έχει αποσταλεί και περιμένει την απάντηση του responder. Σε αυτήν την κατάσταση επίσης λύνει το puzzle και αποστέλλει την λύση του.
- `s1=5`, λαμβάνει το τέταρτο κατά σειρά μήνυμα, επιβεβαιώνει το HMAC και εγκαθίσταται η επικοινωνία.
- `s1=7`, είναι η κατάσταση λάθους.

```
114
115 // CREATE MESSAGE 1
116 []sl=1 -> sl'=2;
117 []HITi=0 & HITr_sent=0 & sl=2 -> (HITi'=1) & (HITr_sent'=1);
118 []sl=2 & message1 -> sl'=3;
119 [m1]no_action & sl=3 ->sl'=3;
120 [finish_m1]sl=3 -> sl'=4;
121 ////////////////////
122
123 //PROCCESS R1 & CREATE AND SEND MESSAGE 3
124 [finish_spoof]sl=4 & counter<=B-a -> 0.999 : (counter'=counter+a) + 0.001:(sl'=7);
125 []counter>=a & ex<=MAX-a & a!=0 -> ex'=ex+a & counter'=counter-a;
126 [m3]ex>=a & ex>0 -> ex'=ex-a;
127 ////////////////////
128
129 //PROCCESS R2
130 [finish_m4]sl=4 & verify_HMAC=false ->sl'=5;
131 []sl=5 -> verify_HMAC'=true & connection'=true;
132 ////////////////////
133
134 //FAIL CONDITIONS
135 []ex=MAX & counter>=proc_limit ->fail'=2;
136 []sl=7 -> fail'=1;
137 ////////////////////
138 endmodule
139
140
141
142 module intruder
143 s3 : [1..9] init 1;
144 //1=initial state
145 //2=message 1 received
146 //3=get hit's
147 //4=send 1st message to responder
148 //5=finished sending 1st message and waiting for responder's message
149 //6=responder'a message stolen
150 //7=choose the value of a, which means the amount of messages arriving
151   //at the initiator simultaneously
152 //8=created spoofed puzzles and ready to send them
```

Εικόνα 4.4: module initiator και intruder

Από την γραμμή 116 έως 120 μοντελοποιείται η παραγωγή του πρώτου μηνύματος, δηλαδή η δημιουργία των HIT του initiator και του responder. Τα labels των εντολών χρησιμοποιούνται για τον συγχρονισμό με τα άλλα modules και στις συγκεκριμένες εντολές με αυτό του meso και του intruder.

Στην γραμμή 124 ο initiator παραλαμβάνει τα αλλαγμένα μηνύματα που στέλνει ο intruder και τα τοποθετεί στην ουρά ενώ στην επόμενη αν υπάρχουν ελεύθεροι εξυπηρέτες τότε τα μηνύματα από την ουρά αναμονής προωθούνται και απασχολούν του εξυπηρέτες. Έπειτα αν τα μηνύματα εξυπηρετηθούν, δηλαδή λυθεί το puzzle, τότε αποδεσμεύονται εξυπηρέτες για να εξυπηρετήσουν άλλα μηνύματα που περιμένουν στην ουρά.

Όταν εκτελεστεί η εντολή της σειράς 130, τότε έχει ληφθεί το τέταρτο μήνυμα και με την εκτέλεση της εντολής επόμενης σειράς, λύνεται το HMAC και τερματίζει το πρωτόκολλο δίνοντας στην μεταβλητή connection την τιμή true. Η μεταβλητή fail της γραμμής 135 παίρνει την τιμή 2 όταν οι εξυπηρέτες είναι όλοι απασχολημένοι και η ουρά είναι γεμάτη πάνω από το όριο εξυπηρέτησης. Αυτό αντιστοιχεί στην κατάσταση όπου ο intruder έχει επιτύχει επίθεση άρνησης υπηρεσίας. Στην γραμμή 136, η μεταβλητή fail παίρνει την τιμή 1 όταν υπάρξει κάποιο σφάλμα κατά την παραλαβή κάποιου μηνύματος.

4.5 Module Intruder

Από την 142, αρχίζει ο ορισμός του module intruder που μοντελοποιεί την συμπεριφορά του εισβολέα στο πρωτόκολλο και στην επόμενη, η μεταβλητή `s3` δείχνει την κατάσταση που βρίσκεται ο intruder.

Πιο συγκεκριμένα:

- `s3=1`, είναι η αρχική κατάσταση του intruder.
- `s3=2`, έχει υποκλέψει το πρώτο μήνυμα που στέλνει ο initiator.

- `s3=3`, αντιγράφει τα host identity tags του responder και του initiator.
- `s3=4`, προωθεί το μήνυμα στον responder.
- `s3=5`, έχει τελειώσει την αποστολή του πρώτου μηνύματος και περιμένει μέχρι να απαντήσει ο responder.
- `s3=6`, έχει υποκλέψει την απάντηση του responder, δηλαδή το μήνυμα που περιέχει το puzzle.
- `s3=7`, επιλογή τιμής για a που δείχνει τα μηνύματα που φτάνουν ταυτόχρονα στον initiator.
- `s3=8`, έχει δημιουργήσει τα μηνύματα με τα αλλαγμένα puzzle και είναι έτοιμος να τα στείλει.
- `s3=9`, έχει στείλει τα μηνύματα.

```
153 //9=spoofed messages sent
154
155 //this is the initiator's host identity tag
156 HITi_stolen : [0..1] init 0;
157 //this is the responder's host identity tag
158 HITr_stolen : [0..1] init 0;
159 //puzzle_change=1 means that intruder has received and changed
160   //the responder's puzzle
161 puzzle_received : [0..1] init 0;
162 //this is a variable that shows the number of times the intruder has sent
163   //messages (the real amount of messages the initiator receives is puzzle_change*a)
164 puzzle_change : [0..M] init 0;
165 //this is a flag used to force an action at a certain point of the protocol
166 flag : bool init false;
167
168 //GETS M1 AND SENDS IT TO RESPONDER
169 ////////////////////////////////////////////////////
170 //message1 is a formula and is true when the first message
171 //this action indicates that the intruder has received the initiator's 1st message
172 [finish_m1]s3=1 & message1 & sending_init -> s3'=2;
173 //the intruder steals the host identity tags of the intruder and the responder
174 []s3=2 -> HITi_stolen'=1 & HITr_stolen'=1 & s3'=3;
175 //since the intruder has stolen the host identity tags, he is ready to forward the
176   //1st message to responder
177 []s3=3 & message1 -> s3'=4;
178 //forwards the message
179 [intr_1]no_action & s3=4 & flag=false -> s3'=4 & flag'=true;
180 //intruder is waiting for responder's response to the 1st message
181 [finish_intr_1]s3=4 -> s3'=5;
182 ////////////////////////////////////////////////////////
183
184 //RECEIVES M2 AND CREATES AND SENDS THE SPOOFED MESSAGES
185 ////////////////////////////////////////////////////////
186 //message2 is a formula and is true when responder's 1st message is ready
187 //intruder has stolen the responder's first message
188 [finish_m2]s3=5 & message2 & sending_resp ->s3'=6;
189 //choose the number of simultaneous messages that will be sent to the initiator
190 []s3=6 -> 0.05:(a'=10) & (s3'=7) + 0.95:(a'=50) & (s3'=7);
191 //intruder has changed the puzzle
```

Εικόνα 4.5: το module του intruder

Στις γραμμές 156 και 158 ορίζονται οι μεταβλητές για τα host identity tags του initiator και responder αντίστοιχα που παίρνουν τιμή 1 όταν τα παραλάβει ο intruder. Στην 161 η μεταβλητή puzzle_received παίρνει την τιμή 1 όταν ο intruder υποκλέψει την απάντηση του responder και στην 164 η μεταβλητή `puzzle_change` που έχει εύρος τιμών από 0 έως `M`, δείχνει πόσες φορές έχει κάνει αλλαγή puzzle ο intruder σε `a` (στον αριθμό) μηνύματα.

Η μεταβλητή της γραμμής 166 είναι απλά βοηθητική ενώ με την εκτέλεση της εντολής που βρίσκεται στην 172, ο intruder έχει υποκλέψει το πρώτο μήνυμα Με την εντολή της γραμμής 174 ο intruder που βρίσκεται στην κατάσταση 2 υποκλέπτει τα host identity tags του initiator και του responder. Στην 177 εφόσον ο intruder βρίσκεται στην κατάσταση 3 και έχει κατασκευαστεί το πρώτο μήνυμα, μεταβαίνει στην κατάσταση 4. Με την εντολή της 179, προωθεί το μήνυμα στον responder. Η εκτέλεση της εντολής στην γραμμή 181 σημαίνει πως ο intruder περιμένει την

αποστολή του δεύτερου μηνύματος από τον responder ενώ η αποστολή του πρώτου μηνύματος έχει τελειώσει. Με την εντολή της γραμμής 188 ο intruder έχει υποκλέψει την απάντηση του responder και μεταβαίνει στην κατάσταση 6. Από αυτήν την κατάσταση και με την εκτέλεση της εντολής στην 190 επιλέγεται η τιμή για το `a`.

```
192 []s3=7 -> puzzle_received'=1 & s3'=8;
193 //intruder is sending the messages
194 [spoof]s3=8 & puzzle_change<M & no_action -> puzzle_change'=puzzle_change+1 & s3'=9;
195 //intruder has finished sending messages and getting ready to send spoofed messages again
196 [finish_spoof]s3=9-> s3'=8;
197 ////////////////////////////////
198 endmodule
199
200
201
202 module responder
203 s2 : [1..10] init 1;
204 signrl : [0..1] init 1;   //i ypografh tou prwtou mhnymatos
205 HITr : [0..1] init 1;   //ti HIT tou responder
206 c : [0..1] init 1;      // h parametros c(proipologismenh), aparaithth gia dhmiourgia puzzle
207 puzzle : bool init false;   // to puzzle
208 k : [0..25] init 0;             //h parametros k gia ti dyskolia tou puzzle
209 gr : [0..1] init 1;             // einai g^r, parametros tou message2
210 HITi_received : [0..1] init 0;  //to Hit tou initiator pou lamvanei o responder
211 HMAC : bool init false;
212 signr2 : bool init false;
213
214
215
216 [finish_intr_1]s2=1 & message1_int & sending_init ->0.999:(s2'=2) + 0.001:(s2'=10);
217 //afou to exei lavei perna se katastash check prwtou nhnymatos
218 []s2=2 -> s2'=3;
219 //epilogh tou k, me ish pithanothta, apo tis times  0  1  10  20  40
220 []HITr=HITr_sent & s2=3 & c=1 & k=0 -> 0.20:(k'=1)
221 + 0.20:(k'=5)+ 0.20:(k'=10)+ 0.20:(k'=15)+ 0.20:(k'=25);
222 //afou exei epilextei to k, kai yparxei to c, dhmiourgei to puzzle
223 []c=1 & k!=0 & s2=3 -> (puzzle'=true)&(HITi_received'=HITi_stolen) & (s2'=4);
224 //otan yparxei to message4 paei se katastash apostolhs tou
225 []s2=4 & message2 -> s2'=5;
226 //an den yparxei kinhsh sto kanali kai responder=5 , tote aposteletai to mhnyma
227 [m2]no_action & s2=5 ->s2'=5;
228 [finish_m2]s2=5 & s1=4-> s2'=6;
229
230 //[]s2=6 & s1=5 -> s2'=7;
```

Εικόνα 4.6: το module του responder

Όταν ο intruder υποκλέψει το puzzle τότε μεταβαίνει στην κατάσταση `s3=8` και στην κατάσταση αυτή αλλάζει το puzzle και αποστέλλει τα αλλαγμένα μηνύματα στον initiator. Στην γραμμή 196 τελειώνει η αποστολή των αλλαγμένων puzzles και ο intruder επανέρχεται σε κατάσταση `s3=8` για να επαναλάβει την διαδικασία αλλαγής και αποστολής των μηνυμάτων. Στην γραμμή 198 έχουμε το τέλος του module intruder.

4.6 Module Responder

Στην γραμμή 203 η μεταβλητή s3 δείχνει την κατάσταση που βρίσκεται ο responder.
Αναλυτικότερα:

- `s2=1`, είναι η αρχική κατάσταση του responder.
- `s2=2`, έχει παραλάβει το πρώτο μήνυμα που στέλνει ο initiator.
- `s2=3`, επιλέγει την δυσκολία του puzzle `k`.
- `s2=4`, έχει ολοκληρωθεί η δημιουργία του δευτέρου μηνύματος.
- `s2=5`, αποστέλλεται το δεύτερο μήνυμα.
- `s2=6`, περιμένει για το τρίτο κατά σειρά μήνυμα που περιέχει τη λύση του puzzle. Επίσης παραμένει στην κατάσταση αυτή όταν παραλάβει το τρίτο μήνυμα.
- `s2=8`, δημιουργεί το HMAC, υπογράφει το μήνυμα και το αποστέλλει.

- `s2=9`, είναι η τελική κατάσταση. Εδώ τελειώνει το πρωτόκολλο για τον responder.
- `s2=10`, κατάσταση λάθους.

Από την γραμμή 204 έως και 212 ορίζονται οι μεταβλητές για τον responder και η επεξήγηση φαίνεται στα σχόλια του κώδικα. Στην 216 γίνεται η παραλαβή του πρώτου μηνύματος και έπειτα η επιλογή της δυσκολίας του puzzle σύμφωνα με [15].

```
230 //[]s2=6 & s1=5 -> s2'=7;
231 //[]s2=6 -> true;
232 [finish_m3]s2=6 -> s2'=6;
233 []s2=6 & puzzle_solution=false ->reject'=true;
234 []s2=6 & puzzle_solution=true -> s2'=8 & HMAC'=true & signr2'=true;
235
236 [m4]s2=8 & no_action -> s2'=8;
237 [finish_m4]s2=8 -> s2'=9;
238
239 []s2=10 -> fail'=1;
240 endmodule
241
242 rewards "responder_cost"
243 [m2] true : TRNSMT_+PR_I1_GEN_R1;
244 [m4] true : TRNSMT_+PR_I2_GEN_R2;
245 endrewards
246
247 rewards "initiator_cost"
248 [m1] true :TRNSMT_+GEN_I1;
249 [m3] k=1 : (TRNSMT_+PR_R1_GEN_I2_k5);
250 [m3] k=10 : (TRNSMT_+PR_R1_GEN_I2_k10);
251 [m3] k=15 : (TRNSMT_+PR_R1_GEN_I2_k15);
252 [m3] k=20 : (TRNSMT_+PR_R1_GEN_I2_k20);
253 [m3] k=25 : (TRNSMT_+PR_R1_GEN_I2_k25);
254 [finish_m4] true : PR_R2;
255 endrewards
256
257 rewards "intruder_cost"
258 [intr_1] true : TRNSMT_;
259 [m3] k=1 : TRNSMT_+PR_R1_GEN_SP_k5;
260 [m3] k=10 : TRNSMT_+PR_R1_GEN_SP_k10;
261 [m3] k=15 : TRNSMT_+PR_R1_GEN_SP_k15;
262 [m3] k=20 : TRNSMT_+PR_R1_GEN_SP_k20;
263 [m3] k=25 : TRNSMT_+PR_R1_GEN_SP_k25;
264 endrewards
265
266 rewards "messages_sent"
267 [spoof] true : a;
268 endrewards
```

Εικόνα 4.7: ορισμός των rewards

Στην γραμμή 232 παραλαμβάνεται το τρίτο μήνυμα από τον responder και αν η λύση δεν είναι σωστή τότε απορρίπτει τον initiator ενώ αν η λύση είναι σωστή τότε προχωρά στην δημιουργία του τέταρτου μηνύματος.
Μετά το module του responder ορίζονται τα rewards για τους τρεις συμμετέχοντες του πρωτοκόλλου σύμφωνα με [15].

ΚΕΦΑΛΑΙΟ V

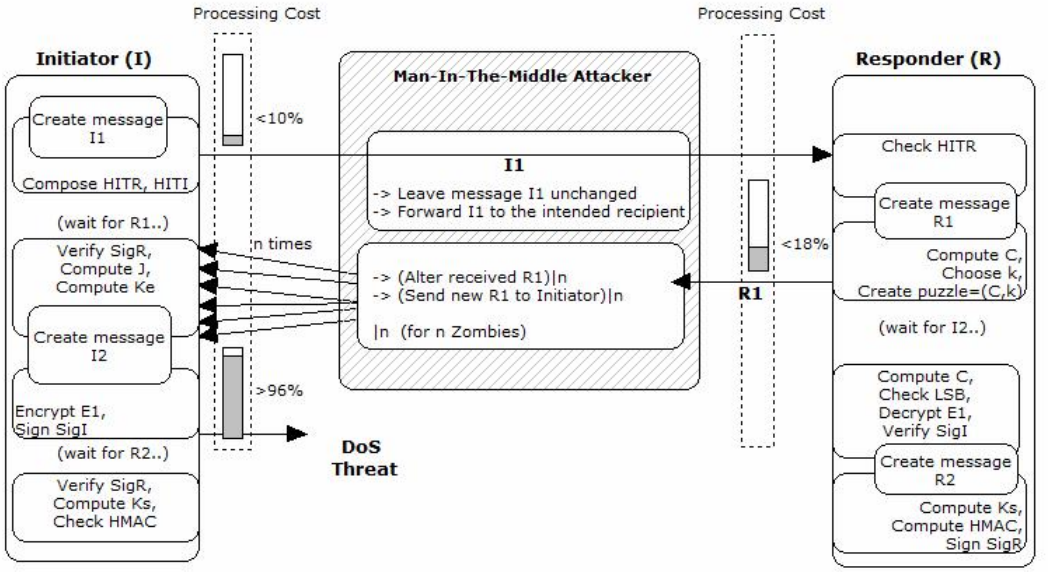
5. ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΗΣ ΑΝΑΛΥΣΗΣ

5.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα που προέκυψαν από την ανάλυση του μοντέλου με το prism. Ο κάθε γράφος ή πιθανότητα παρουσιάζεται μαζί με το ερώτημα-ιδιότητα που ελέγχθηκε σε κάθε περίπτωση.

5.2 Πιθανότητα επιτυχίας επίθεσης άρνησης υπηρεσίας

Στην ενότητα αυτή εξετάζεται η ιδιότητα `P=? [ true U fail=2 ]`, που ελέγχει την πιθανότητα η μεταβλητή `fail` να πάρει τιμή ίση με 2. Αυτό συμβαίνει μόνο όταν όλοι οι εξυπηρέτες είναι απασχολημένοι και η ουρά έχει ξεπεράσει το όριο εξυπηρέτησής της, δηλαδή όταν έχει επιτύχει μια επίθεση άρνησης υπηρεσίας κατά του πρωτοκόλλου. Η πιθανότητα να επιτύχει η άρνηση υπηρεσίας είναι πολύ υψηλή και περίπου ίση με 0.89. Η επόμενη εικόνα παρουσιάζει την λειτουργία του εισβολέα για κάθε φάση του πρωτοκόλλου. Στο παράρτημα Γ παρουσιάζεται το αρχείο log που δημιουργείται από το εργαλείο όταν εξετάζεται η παραπάνω ιδιότητα.



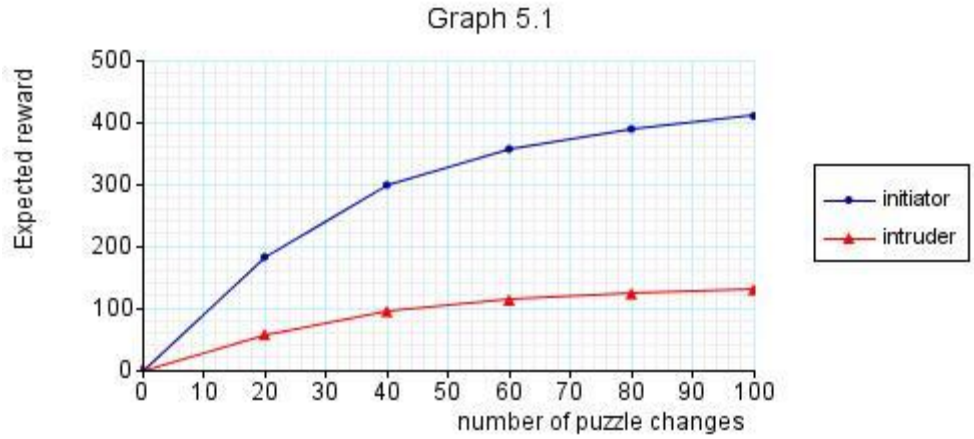
Εικόνα 5.1: Σχηματική αναπαράσταση της λειτουργίας του εισβολέα

5.3 Κόστος Ανάλογα με την Δυσκολία του Puzzle

Η εικόνα 5.2 παρουσιάζει το κόστος που έχει ο initiator και ο εισβολέας, ο μεν πρώτος για να λύσει το puzzle και ο δεύτερος για να παράγει τα αλλαγμένα puzzle που θα στείλει στον initiator. Ο παρακάτω γράφος αφορά όλους τους πιθανούς βαθμούς δυσκολίας του puzzle.

Στον άξονα x μετριέται ο αριθμός αλλαγών puzzle που έκανε ο intruder ενώ στον άξονα y το αναμενόμενο κόστος. (Σε κάθε αλλαγή puzzle ο intruder στέλνει περισσότερα του ενός μηνύματα).

```
R{"intruder_cost"}=? [ F fail=1 | fail=2 | puzzle_change>num]
R{"ininiator_cost"}=? [ F fail=1 | fail=2 | puzzle_change>num]
```



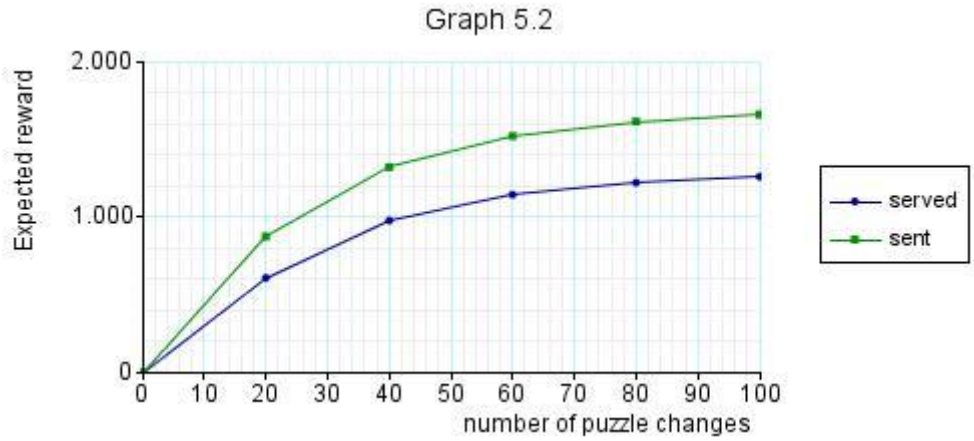
Η εικόνα 5.2: κόστος ανάλογα με δυσκολία puzzle

Η εικόνα 5.2 δείχνει την πορεία του συνολικού κόστους που επιβαρύνει τον initiator και τον intruder για την λύση και την παραγωγή puzzles αντίστοιχα. Το αποτέλεσμα δείχνει το κατά πολύ μικρότερο κόστος που έχει ο intruder σε σχέση με τον initiator. Η δυσκολία του puzzle είναι τυχαία επιλεγμένη σύμφωνα με [15] για τιμές 1,10,15,20,25.

5.4 Μηνύματα που Στέλνονται και Εξυπηρετούνται

Στην εικόνα 5.3 παρουσιάζεται ο αριθμός των μηνυμάτων που στέλνονται και ο αντίστοιχος αυτών που εξυπηρετούνται. Τα στοιχεία στους δύο άξονες είναι τα ίδια με της εικόνας 5.2.

```
const int num;
R{"messages_sent"}=? [ F fail=1 | fail=2 | puzzle_change>=num]
R{"messages_served"}=? [ F fail=1 | fail=2 |
puzzle_change>=num ]
```



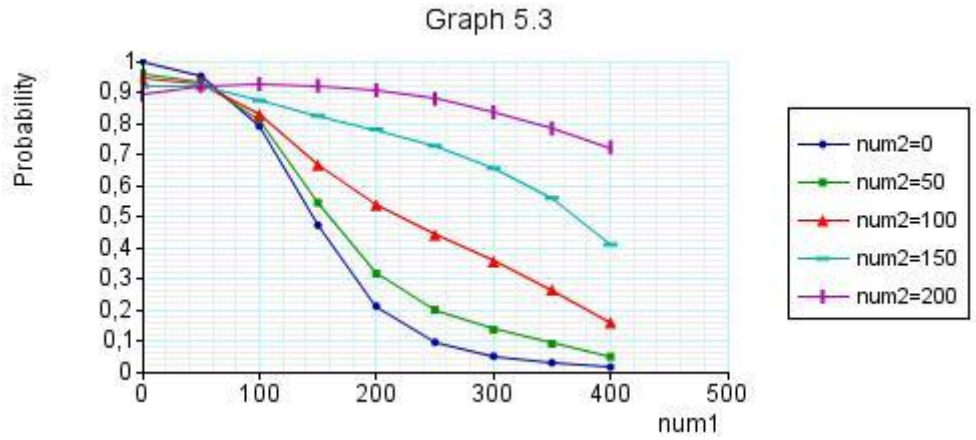
Εικόνα 5.3: μηνύματα εξυπηρετούμενα και απεσταλμένα

Είναι φανερό πως καθώς όλο και περισσότερα puzzles προς λύση στέλνονται από τον εισβολέα, μεγάλος αριθμός από αυτά δεν επιλύονται με αποτέλεσμα να γεμίζει η ουρά των εργασιών προς λύση και την απασχόληση όλων των εξυπηρετών. Με άλλα λόγια την επιτυχία ενός denial of service attack.

5.5 Πορεία Εξυπηρετών και Ουράς

Στην εικόνα 5.4 παρουσιάζεται η πορεία των εξυπηρετών με την μεταβλητή num2 και της ουράς με την μεταβλητή num1. Στον άξονα y βρίσκεται η πιθανότητα. Οι πορεία των εξυπηρετών εξετάζεται για διάφορες τιμές που φαίνονται στο πλαίσιο δίπλα στον γράφο ενώ η τιμή της ουράς κυμαίνεται από 0 έως 400.

```
P=? [ true U counter=num1 & ex=num2 ]
```



Εικόνα 5.4: φόρτος ουράς - εξυπηρετών

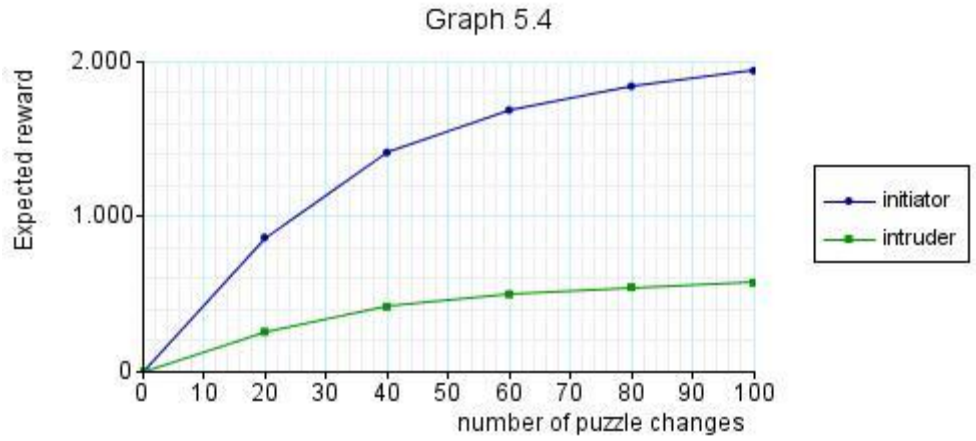
Από τον γράφο συμπεραίνεται ότι μια μη επιθυμητή και υψηλή πιθανότητα για να φτάσει το σύστημα σε κατάσταση που δεν μπορεί να εξυπηρετήσει άλλες αιτήσεις. Σύμφωνα και με το [13], η αντοχή του συστήματος σε επίθεση τύπου άρνησης υπηρεσίας βελτιστοποιείται για ουρές εργασιών προς εξυπηρέτηση μεγαλύτερες των 250 θέσεων χωρητικότητας, υπό την προϋπόθεση ότι ο initiator δεν θα επεξεργάζεται ταυτόχρονα πάνω από 50 μηνύματα.

Όσο πιο γεμάτη είναι η ουρά τόσο περισσότεροι εξυπηρέτες είναι απασχολημένοι. Αυτό δείχνει τον αυξανόμενο ρυθμό άφιξης μηνυμάτων προς εξυπηρέτηση στον initiator και την αδυναμία του να τα εξυπηρετήσει.

5.6 Κόστος Intruder και Initiator

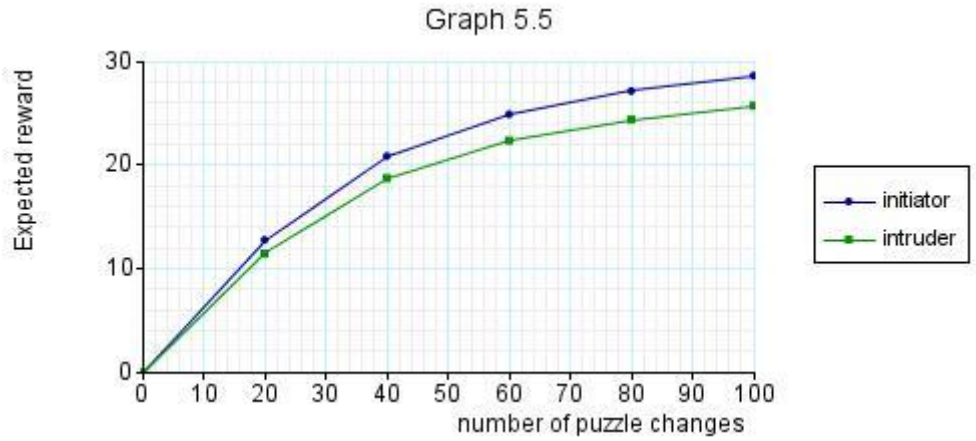
```
R{"initiator_cost"}=?[ F fail=1 | fail=2 | puzzle_change>=num]
R{"intruder_cost"}=? [ F fail=1 | fail=2 | puzzle_change>=num]
```

Για k=25.



Εικόνα 5.5: κόστος με δυσκολία puzzle k=25.

Σύμφωνα και με το [14], η εικόνα 5.5 παρουσιάζει το κόστος των intruder και initiator για την παραγωγή και λύση puzzles αντίστοιχα όταν αυτά έχουν τον μεγαλύτερο βαθμό δυσκολίας (κ=25). Είναι φανερό το κατά πολύ μεγαλύτερο κόστος για τη λύση των puzzles σε σχέση με την παραγωγή τους.



Εικόνα 5.6: κόστος με δυσκολία puzzle k=1.

Στην εικόνα 5.6 εξετάζεται η ίδια ιδιότητα αλλά για βαθμό δυσκολίας puzzle κ=1. Ο intruder έχει και πάλι μικρότερο κόστος αλλά αυτό έχει μειωθεί αισθητά. Επίσης σύμφωνα με το [14] για ουρές μεγέθους μεταξύ 0 και 60 το υπολογιστικό κόστος για τον initiator μπορεί να αυξηθεί δραματικά, ειδικά όταν η δυσκολία του puzzle είναι μεγαλύτερη ή ίση με 20.

Επίλογος

Στην εργασία αυτή έγινε η ανάλυση του πρωτοκόλλου HIP, πρωτοκόλλου προσανατολισμένου να είναι ανθεκτικό σε επιθέσεις τύπου άρνησης υπηρεσίας. Μέσω του εργαλείου ελέγχου μοντέλων βασισμένο σε πιθανότητες PRISM, μοντελοποιήθηκε το παραπάνω πρωτόκολλο και ελέγχθηκε η αντοχή του σε γνωστές μεθόδους επίθεσης άρνησης υπηρεσίας μέσω ενός μοντέλου εισβολέα που παρεμβάλλεται στην ομαλή λειτουργία του HIP. Ελέγχθηκε η πιθανότητα να φτάσει το πρωτόκολλο σε κατάσταση όπου κάποιος χρήστης του τίθεται εκτός λειτουργίας και το κόστος διενέργειας της επίθεσης για τον εισβολέα και τον initiator [13]. Η μοντελοποίηση στο PRISM έγινε χρησιμοποιώντας μέθοδο επίλυσης puzzle για κάποιον initiator-honest client και η ποσοτικοποίηση των αποτελεσμάτων βασίστηκε στην ικανότητα του εργαλείου να προσαρτεί κόστη (rewards), ανάλογα με τις προδιαγραφές του πρωτοκόλλου, διακριτά για κάθε διεργασία (module). Τα αποτελέσματα δείχνουν την σοβαρότητα του κενού ασφαλείας που παρουσιάζει το πρωτόκολλο σε ενδεχόμενη επίθεση άρνησης υπηρεσίας τύπου όμοιου με αυτήν που μοντελοποιήθηκε.

Αποδείχθηκε ότι ένας εισβολέας μπορεί να ξεγελάσει κάποιον νόμιμο χρήστη του πρωτοκόλλου αναγκάζοντας τον να λύνει λανθασμένα puzzles κάνοντας σαφές πως χρειάζονται σημαντικές αλλαγές για να προσφέρεται το επιθυμητό επίπεδο ασφαλείας [10].

Αναφορές

[1]Wikipedia, http://en.wikipedia.org/wiki/Wikipedia/, (τελευταία πρόσβαση: 14/01/2007).

[2]NuSMV website, http://nusmv.fbk.eu, (τελευταία πρόσβαση: 14/01/2007).

[3]MRMC website, http://wwwhome.cs.utwente.nl/~zapreevis/mrmc/, (τελευταία πρόσβαση: 14/01/2007).

[4]UPPAL website, http://www.uppal.com, (τελευταία πρόσβαση: 20/01/2007).

[5] Kronos webpage, http://www-verimag.imag.fr/TEMPORISE/kronos/, (τελευταία πρόσβαση: 17/12/2007).

[6]Alloy website, http://alloy.mit.edu, (τελευταία πρόσβαση: 2/2/2007).

[7]CADP webpage, http://www.inrialpes.fr/vasy/cadp/, (τελευταία πρόσβαση: 12/2/2007).

[8] Prism website, http://prismodelchecker.org, (τελευταία πρόσβαση: 5/2/2008)

[9] Marta Kwiatkowska, "Probabilistic Model Checking of Randomised Distributed Protocols using PRISM" ,tutorial at the VPSM PhD school, Copenhagen, Denmark, October 2006.

.

[10] T. Aura, A. Nagarajan, and A. Gurtov, “Analysis of the hip base exchange protocol”, In ACISP, pages 481-493, 2005.

[11] Suratose, T.,Boyd, C., Foo, E., Nieto, J., “Cost-based and Time-based Analysis of DoS-resistance in HIP”, Pages: 191 - 200 , Information Security Institute, Queensland University of Technology, Australia, 2007, ISBN: 1445-1336.

[12] Performance of Host Identity Protocol on Nokia Internet Tablet, Andrey Khurri, HIP Research Group, Helsinki Institute for Information Technology, IETF 68 Prague, March 23, 2007.

.

[13] Basagiannis, S., Katsaros, P., Pombortsis, A. and Alexiou N., “Probabilistic model checking for the quantification of Dos Security threats”, Technical Report 2008, (submitted for evaluation).

[14] Basagiannis, S., Katsaros, P., Pombortsis, A. and Alexiou N., “A Probabilistic attacker Model for Quantitative Verification of Dos Security Threats”, Technical Report 2008, (submitted for evaluation).

[15] Tritilanunt, S., Boyd, C., Foo, E., Gonzalez Nieto, J., “Using Coloured Petri Nets to Simulate DoS-resistant Protocols”, In Seventh Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, pages 261--280. University of Aarhus, Denmark, October 2006. ISSB: 0105-8517.

[16] Basagiannis, S., Katsaros, P. and Pombortsis, A. “Intrusion Attack Tactics for the model checking of e-commerce security guarantees”, Proceedings of the 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP), Nuremberg, Germany, LNCS 4680, Springer Verlag, 238-251, 2007.

Παράρτημα Α

Στο παράρτημα Α παρουσιάζεται ο κώδικας σε γλώσσα prism που προσωμοιώνει το πρωτόκολλο HIP.

```
//THIS IS A dtmc
dtmc
const double MAX_TIME=2000;
//////////////////
//const int INIT_=10; //xronos gia thn apostolh kai thn
epanapostolh tou prwtou mhnumatos
////////////////////
const double GEN_I1 = 0.05;
const double PR_I1_GEN_R1=0.05; //gen/process I1
const double PR_R1_GEN_I2_k5=1;
const double PR_R1_GEN_I2_k10=1.35;
const double PR_R1_GEN_I2_k15=1.6;
const double PR_R1_GEN_I2_k20=13;
const double PR_R1_GEN_I2_k25=68;
const double PR_I2_GEN_R2=1.48;
const double PR_R2=0.05;
const double TRNSMT_=0.002;

const double PR_R1_GEN_SP_k5=0.9;
const double PR_R1_GEN_SP_k10=0.9;
const double PR_R1_GEN_SP_k15=1;
const double PR_R1_GEN_SP_k20=13;
const double PR_R1_GEN_SP_k25=20;

//THESE ARE THE FORMULAS

//no action over the channel
formula no_action = c1=2 & c2=2;
//initiator sending
formula sending_init = c1=1 & c2=2;
//responder sending
formula sending_resp = c1=2 & c2=1;
//exei dimiourgithei to message 1
formula message1 = HITi=1 & HITr_sent=1;
formula message1_int = HITi_stolen=1 & HITr_stolen=1;
formula message2 = HITi_received=1 & HITr=1 & puzzle=true &
HITr=1 & gr=1 & HITi_received=1 & signr1=1;


//END OF FORMULAS

//THESE ARE THE CONSTANTS
const int M = 100;
//anw orio counter tou initiator
const int B = 400;
const int proc_limit = 200;
const int MAX = 200;
```

```
//aritmos minymatwn pou ftanoun sti monada xronoy
global a : [0..50] init 0;

global fail : [0..2] init 0;
//0 no failure
// 1 connection failure
// 2 dos attack failure

global puzzle_solution : bool init false;
global reject : bool init false;
global connection : bool init false;
//END OD CONSTANTS


module meso

//c1=1: no action towards responder
//c1=2: action towards responder
c1 : [1..2] init 2;
c2 : [1..2] init 2;

//MESSAGE 1
[m1]c1=2 & c2=2 -> c1'=1;
[finish_m1](c1=1 & c2=2 & s3=1) -> (c1'=2) & (c2'=2);
///////////////

//MESSAGE 1, INTR->RESPONDER
[intr_1]no_action -> c1'=1;
[finish_intr_1]sending_init & s3=4-> c1'=2;
////////////////

//MESSAGE 2
[m2]no_action & message2 -> c2'=1;
[finish_m2]sending_resp & s1=4 -> (c1'=2) & (c2'=2);
////////////////

//SPOOFED MESSAGE 2  INTR->INITIATOR
[spoof]no_action -> c2'=1;
[finish_spoof]sending_resp -> c2'=2;
///////////////

//MESSAGE 3
[m3]no_action -> c1'=1;
[finish_m3]sending_init -> c1'=2;
///////////////

//MESSAGE 4
[m4]no_action -> c2'=1;
[finish_m4]sending_resp -> c2'=2;
endmodule

module initiator
s1 : [1..7] init 1;
//states for initiator
//1: initial state
//2: prepare 1st message
```

```
//3: sending first message
//4: wait for responder's answer and send message3
HITi : [0..1] init 0;//initiator's host identity tag
HITr_sent : [0..1] init 0;//responder's host identity tag
counter : [0..B] init 0;//number of requests waiting to be
answered
verify_sign : bool init false;//signature verification for
received message 2
ex : [0..MAX] init 0;//indicates the proccessors
verify_HMAC : bool init false;//HMAC verification
msgs : [0..500] init 0;

// CREATE MESSAGE 1
[]s1=1 -> s1'=2;
[]HITi=0 & HITr_sent=0 & s1=2 -> (HITi'=1) & (HITr_sent'=1);
[]s1=2 & message1 -> s1'=3;
[m1]no_action & s1=3 ->s1'=3;
[finish_m1]s1=3 -> s1'=4;
//////////////////

//PROCCESS R1 & CREATE AND SEND MESSAGE 3
[finish_spoof]s1=4 & counter<=B-a -> 0.999 :
(counter'=counter+a) + 0.001:(s1'=7);
[]counter>=a & ex<=MAX-a & a!=0 -> ex'=ex+a &
counter'=counter-a;
[m3]ex>=a & ex>0 -> ex'=ex-a;
////////////////////

//PROCCESS R2
[finish_m4]s1=4 & verify_HMAC=false ->s1'=5;
[]s1=5 -> verify_HMAC'=true & connection'=true;
////////////////////

//FAIL CONDITIONS
[]ex=MAX & counter>=proc_limit ->fail'=2;
[]s1=7 -> fail'=1;
////////////////////
endmodule


module intruder
s3 : [1..9] init 1;
//1=initial state
//2=message 1 received
//3=get hit's
//4=send 1st message to responder
//5=finished sending 1st message and waiting for responder's
message
//6=responder'a message stolen
//7=choose the value of a, which means the amount of messages
arriving
  //at the initiator simultaneously
//8=created spoofed puzzles and ready to send them
//9=spoofed messages sent

//this is the initiator's host identity tag
```

```
HITi_stolen : [0..1] init 0;
//this is the responder's host identity tag
HITr_stolen : [0..1] init 0;
//puzzle_change=1 means that intruder has received and changed
  //the responder's puzzle
puzzle_received : [0..1] init 0;
//this is a variable that shows the number of times the
intruder has sent
  //messages (the real amount of messages the initiator
receives is puzzle_change*a)
puzzle_change : [0..M] init 0;
//this is a flag used to force an action at a certain point of
the protocol
flag : bool init false;

//GETS M1 AND SENDS IT TO RESPONDER
////////////////////////////////////////////////
//message1 is a formula and is true when the first message
//this action indicates that the intruder has received the
initiator's 1st message
[finish_m1]s3=1 & message1 & sending_init -> s3'=2;
//the intruder steals the host identity tags of the intruder
and the responder
[]s3=2 -> HITi_stolen'=1 & HITr_stolen'=1 & s3'=3;
//since the intruder has stolen the host identity tags, he is
ready to forward the
  //1st message to responder
[]s3=3 & message1 -> s3'=4;
//forwards the message
[intr_1]no_action & s3=4 & flag=false -> s3'=4 & flag'=true;
//intruder is waiting for responder's response to the 1st
message
[finish_intr_1]s3=4 -> s3'=5;
////////////////////////////////////////////////////////

//RECEIVES M2 AND CREATES AND SENDS THE SPOOFED MESSAGES
////////////////////////////////////////////////////////
//message2 is a formula and is true when responder's 1st
message is ready
//intruder has stolen the responder's first message
[finish_m2]s3=5 & message2 & sending_resp ->s3'=6;
//choose the number of simultaneous messages that will be sent
to the initiator
[]s3=6 -> 0.05:(a'=10) & (s3'=7) + 0.95:(a'=50) & (s3'=7);
//intruder has changed the puzzle
[]s3=7 -> puzzle_received'=1 & s3'=8;
//intruder is sending the messages
[spoof]s3=8 & puzzle_change<M & no_action ->
puzzle_change'=puzzle_change+1 & s3'=9;
//intruder has finished sending messages and getting ready to
send spoofed messages again
[finish_spoof]s3=9-> s3'=8;
////////////////////////////////
endmodule
```

```
module responder
s2 : [1..10] init 1;
signr1 : [0..1] init 1;   //i ypografh tou prwtou mhnymatos
HITr : [0..1] init 1; //ti HIT tou responder
c : [0..1] init 1;      // h parametros c(proipologismenh),
aparaithth gia dhmiourgia puzzle
puzzle : bool init false;   // to puzzle
k : [0..25] init 0;        //h parametros k gia ti dyskolia
tou puzzle
gr : [0..1] init 1;        // einai g^r, parametros tou
message2
HITi_received : [0..1] init 0;   //to Hit tou initiator pou
lamvanei o responder
HMAC : bool init false;
signr2 : bool init false;



[finish_intr_1]s2=1 & message1_int & sending_init -
>0.999:(s2'=2) + 0.001:(s2'=10);
//afou to exei lavei perna se katastash check prwtou nhnymatos
[]s2=2 -> s2'=3;
//epilogh tou k, me ish pithanothta, apo tis times  0  1  10
20  40
[]HITr=HITr_sent & s2=3 & c=1 & k=0 -> 0.20:(k'=1)
+ 0.20:(k'=5)+ 0.20:(k'=10)+ 0.20:(k'=15)+ 0.20:(k'=25);
//afou exei epilextei to k, kai yparxei to c, dhmiourgei to
puzzle
[]c=1 & k!=0 & s2=3 ->
(puzzle'=true)&(HITi_received'=HITi_stolen) & (s2'=4);
//otan yparxei to message4 paei se katastash apostolhs tou
[]s2=4 & message2 -> s2'=5;
//an den yparxei kinhsh sto kanali kai responder=5 , tote
aposteletai to mhnyma
[m2]no_action & s2=5 ->s2'=5;
[finish_m2]s2=5 & s1=4-> s2'=6;

//[]s2=6 & s1=5 -> s2'=7;
//[]s2=6 -> true;
[finish_m3]s2=6 -> s2'=6;
[]s2=6 & puzzle_solution=false ->reject'=true;
[]s2=6 & puzzle_solution=true -> s2'=8 & HMAC'=true &
signr2'=true;

[m4]s2=8 & no_action -> s2'=8;
[finish_m4]s2=8 -> s2'=9;

[]s2=10 -> fail'=1;
endmodule

rewards "responder_cost"
[m2] true : TRNSMT_+PR_I1_GEN_R1;
[m4] true : TRNSMT_+PR_I2_GEN_R2;
endrewards
```

```
rewards "initiator_cost"
[m1] true :TRNSMT_+GEN_I1;
[m3] k=1 : (TRNSMT_+PR_R1_GEN_I2_k5);
[m3] k=10 : (TRNSMT_+PR_R1_GEN_I2_k10);
[m3] k=15 : (TRNSMT_+PR_R1_GEN_I2_k15);
[m3] k=20 : (TRNSMT_+PR_R1_GEN_I2_k20);
[m3] k=25 : (TRNSMT_+PR_R1_GEN_I2_k25);
[finish_m4] true : PR_R2;
endrewards

rewards "intruder_cost"
[intr_1] true : TRNSMT_;
[m3] k=1 : TRNSMT_+PR_R1_GEN_SP_k5;
[m3] k=10 : TRNSMT_+PR_R1_GEN_SP_k10;
[m3] k=15 : TRNSMT_+PR_R1_GEN_SP_k15;
[m3] k=20 : TRNSMT_+PR_R1_GEN_SP_k20;
[m3] k=25 : TRNSMT_+PR_R1_GEN_SP_k25;
endrewards

rewards "messages_sent"
[spoof] true : a;
endrewards

rewards "messages_served"
[m3] true : a;
```
Endrewards

Παράρτημα Β

Στο παράρτημα Β παρουσιάζονται οι ιδιότητες που ελέγχθηκαν για το μοντέλο του παραρτήματος Α.

```
P=? [ true U fail=2 ]
R{"intruder_cost"}=? [ F fail=1 | fail=2 | puzzle_change>num]
R{"ininiator_cost"}=? [ F fail=1 | fail=2 | puzzle_change>num]
P=? [ true U counter=num1 & ex=num2 ]
```

Επίσης παρουσιάζεται το αρχείο log που δημιουργεί το prism όταν εξετάζει την ιδιότητα `P=? [ true U fail=2 ]`.

```
Building model...
Computing reachable states...
Reachability: 520 iterations in 110.02 seconds (average
0.211569, setup 0.00)
Time for model construction: 255.547 seconds.
States:      8733343 (1 initial)
Transitions: 31988778
Transition matrix: 155297 nodes (16 terminal), 31988778
minterms, vars: 78r/78c
Transition rewards (0): 173 nodes (2 terminal), 5 minterms
Transition rewards (1): 9387 nodes (6 terminal), 2203809
minterms
Transition rewards (2): 9378 nodes (5 terminal), 2203809
minterms
Transition rewards (3): 9439 nodes (3 terminal), 2892600
minterms

Model checking: P=? [ true U fail=2 ]

Prob1: 1052 iterations in 25.67 seconds (average 0.024403,
setup 0.00)
Prob0: 103 iterations in 3.61 seconds (average 0.035039, setup
0.00)
yes = 3288740, no = 3324002, maybe = 2120601
Computing remaining probabilities...
Building hybrid MTBDD matrix... [levels=78, nodes=104514]
[2449.5 KB]
Adding explicit sparse matrices... [levels=24, num=1141,
compact] [785.4 KB]
Creating vector for diagonals... [dist=6, compact] [17057.4
KB]
Creating vector for RHS... [dist=2, compact] [17057.3 KB]
Allocating iteration vectors... [2 x 68229.2 KB]
TOTAL: [173808.1 KB]
Starting iterations...
Jacobi: 500 iterations in 148.11 seconds (average 0.280094,
setup 8.06)
Time for model checking: 178.532 seconds.

Result: 0.8948948113273193
```