ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

"ΠΡΟΗΓΜΕΝΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γενικοί Παίκτες: Αλγόριθμοι και Τεχνικές

Ελένη Δ. Βλαχομήτρου
Νίκος Π. Πρίντεζης

Επιβλέποντες: **Παναγιώτης Σταματόπουλος,** Επίκουρος Καθηγητής

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2012

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γενικοί παίκτες: Αλγόριθμοι και Τεχνικές

Ελένη Δ. Βλαχομήτρου

Α.Μ.: Μ1104

Νίκος Ν. Πρίντεζης

Α.Μ: Μ1103

Επιβλέποντες: **Παναγιώτης Σταματόπουλος,** Επίκουρος Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Ιζαμπώ Καράλη,** Επίκουρος Καθηγητής

Σεπτέμβριος 2012

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία έχει σαν αντικείμενο τη μελέτη γενικών παικτών (General Game Players). Οι γενικοί παίκτες είναι υπεύθυνοι για το παίξιμο οποιουδήποτε παιχνιδιού, αρκεί οι κανόνες του να περιγράφονται σε μια συγκεκριμένη μορφή – γλώσσα. Η αρχή για την υλοποίηση γενικών παικτών έγινε τη δεκαετία του 2000, όμως η ιδέα γεννήθηκε πολλά χρόνια πριν, σε πολύ πρώιμη μορφή. Στην εργασία αυτή γίνεται μια καταγραφή της βασικής αρχιτεκτονικής ενός γενικού παίκτη, των εξελίξεων που υπάρχουν στον τομέα τα τελευταία χρόνια, καθώς και των δυσκολιών που υπάρχουν ακόμα.

Προκειμένου να γίνει πιο αποτελεσματική η μελέτη, στα πλαίσια της διπλωματικής εργασίας, αναπτύχθηκε ο γενικός παίκτης SweetGGP, ώστε να εφαρμοστούν στην πράξη, όσο περισσότερο γίνεται, οι γνώσεις που αποκτήθηκαν, και να εξαχθούν συμπεράσματα.

Τέλος, αναπτύχθηκαν επεκτάσεις στη γλώσσα περιγραφής παιχνιδιών, προκειμένου να γίνει ευκολότερη η περιγραφή ορισμένων ομάδων παιχνιδιών.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: γενικοί παίκτες, αναζήτηση, αντιπαλότητα, αναπαράσταση παιχνιδιών, ευριστικές

ABSTRACT

The object of this master thesis is the study of general game players. General game players are able to play any game, if the game's rules are represented in a certain format – language. The development of the first general game players began on 2000s, but the idea was born many years earlier, in an early version. In this thesis, the basic architecture of a general game player, the developments in recent years, and the difficulties found are presented.

In order for the study to be more accurate, the SweetGGP general game player was developed, in order to use the knowledge gained, as much as possible, and come to conclusions.

Finally, extensions of the game description language were developed, so that the description of some groups of games becomes easier.

SUBJECT AREA: Artificial Intelligence

KEYWORDS: general game players, search, rivalry, game representation, heuristics

Στους γονείς μας

ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστούμε τον Καθηγητή μας κ. Παναγιώτη Σταματόπουλο για την καθοδήγηση του και τη βοήθεια του, καθώς επίσης και τους γονείς μας για την στήριξη τους σε όλη την περίοδο της μελέτης μας. Τέλος ευχαριστούμε όλους τους φίλους μας που παρόλο που τους παραμελήσαμε αυτούς τους τελευταίους μήνες έδειξαν κατανόηση.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΠΡΟΛΟΓΟΣ

Η συγκεκριμένη διπλωματική εργασία πραγματοποιήθηκε κυρίως στη Χαλκίδα και Αθήνα που διαμένουμε καθώς επίσης και αρκετές ώρες στη σχολή μας στο τμήμα Πληροφορικής και Τηλ/νιων ΕΚΠΑ. Θα θέλαμε να αναφέρουμε ότι ήταν μία άψογη συνεργασία κάτω από αμοιβαία εκτίμηση, σεβασμό και χρωστάμε ένα μεγάλο ευχαριστώ ο ένας στον άλλον για τη στήριξη του. Ευχαριστούμε επίσης τον καθηγητή μας κ. Παναγιώτη Σταματόπουλο για την πρόταση και προτροπή του να αναλάβουμε το συγκεκριμένο θέμα γιατί ήταν άκρως ενδιαφέρον και για τους δύο μας λόγω της φύσης του αλλά και της δουλειάς που απαιτούσε.

1. ΕΙΣΑΓΩΓΗ

Το Γενικό Παίξιμο (General Game Playing, GGP) αποτελεί την τέχνη της σχεδίασης προγραμμάτων τα οποία είναι ικανά να παίζουν ως παίκτες σε παιχνίδια τα οποία τους είναι «άγνωστα», διαφόρων ειδών παιχνίδια, όπως σκάκι, ντάμα, κτλ., με μοναδική πληροφορία τους κανόνες του εκάστοτε παιχνιδιού. Η έννοια του γενικού παιξίματος έρχεται σε αντίθεση με τη φύση προηγούμενων κλασικών προγραμμάτων παιχνιδιών, όπως το Deep Blue για σκάκι, διότι τα συγκεκριμένα προγράμματα είναι σχεδιασμένα να παίζουν ένα συγκεκριμένο παιχνίδι και είναι εφοδιασμένα με όλες τις πληροφορίες που χρειάζονται εξαρχής. [1]

Τα συστήματα γενικού παιξίματος είναι σχεδιασμένα να παίζουν ως καλοί παίκτες σε παιχνίδια, για τα οποία γνωρίζουν μόνο τους κανόνες τους. Το κυριότερο πρόβλημα που προκύπτει σε αυτή την περίπτωση είναι ότι στον προγραμματιστή είναι άγνωστο το ποια παιχνίδια θα δοθούν στο πρόγραμμα και το τι κανόνες περιέχουν, οπότε είναι αδύνατον να κατασκευάσει εξαρχής συναρτήσεις αξιολόγησης, ευριστικές συναρτήσεις, βιβλιοθήκες, κτλ., προκειμένου να ενισχύσει την απόδοση και την ικανότητα του προγράμματος. Για τους λόγους αυτούς, ένας γενικός παίκτης (general game player) σχεδιάζεται με στρατηγικές μεθόδους γενικού σκοπού. [1]

Σε αυτό το κεφάλαιο παρουσιάζεται μια σύντομη περιγραφή της προηγούμενης δουλειάς που είχε γίνει σε αυτό τον τομέα. Στα επόμενα κεφάλαια γίνεται παρουσίαση των χαρακτηριστικών ενός γενικού παίκτη, καθώς και πληροφορίες για την υλοποίηση του γενικού παίκτη SweetGGP και την υλοποίηση επεκτάσεων για τη γλώσσα περιγραφής παιχνιδιών GDL (Game Description Language).

1.1 Προηγούμενη Δουλειά

Η κυριότερη δουλειά πάνω στον τομέα του γενικού παιξίματος έγινε την τελευταία δεκαετία και ειδικά απο το 2005 και μετά. Στην πραγματικότητα όμως οι πρώτες βάσεις για το συγκεκριμένο τομέα είχαν δοθεί απο την δεκαετία του 90. Παρακάτω ακολουθούν μερικές δουλειές που αποτέλεσαν την ιδέα για το ξεκίνημα του:

- Pell στην πτυχιακή του, το 1993, παρουσιάζει την κατασκευή γενικών παικτών για παιχνίδια που είναι όμοια με το σκάκι, καθώς και το προσωπικό του πρόγραμμα, που ονομάζεται Metagamer. [2].
- Οι Fawcett και Utgoff, το 1993, παρουσιάζουν έναν αλγόριθμο για εύρεση χαρακτηριστικών στο Othello. Τα χαρακτηριστικά παράγονται εφαρμόζοντας επαγωγικά τελεστές μετασχηματισμού ξεκινώντας απο την περιγραφή του στόχου. [3].
- Η IBM κατασκευάζει τον Μάιο του 1997 μία πολύπλοκη σκακιστική μηχανή η οποία είχε μεν πολύ μεγάλο όγκο, αλλά παρόλαυτα περιείχε σαν ιδιότητες, αναπτυγμένες τεχνικές αναζήτησης για την εποχή, καθώς είχε δοθεί ιδιαίτερη έμφαση σε αυτή και πολύπλοκη ευριστική συνάρτηση. Αξίζει να σημειωθεί ότι η συγκεκριμένη μηχανή είχε καταφέρει να κερδίσει δύο φορές τον Παγκόσμιο πρωταθλητή στο σκάκι Garry Kasparov. [13].
- Το 2005 οι Michael Genesereth και Nathaniel Love στη δουλειά τους έκαναν μία περιεκτική περιγραφή ενός Γενικού Παίχτη καθώς επίσης και της σχετικής γλώσσας περιγραφής παιχνιδιών. [14].
- Το 2009 οι Yngvi Björnsson και Hilmar Finnsson δημιουργείται ο Γενικός Παίχτης CADIAPLAYER η συνάρτηση αναζήτησης του οποίου στηρίζεται στις

προσομοιώσεις Monte – Carlo που θα αναλυθούν παρακάτω, έναντι της παραδοσιακής μεθόδου αναζήτησης βασισμένη σε δέντρο αναζήτησης που χρησιμοποιούνταν μέχρι τότε. [18].

- Το 2010, 2011 και μέχρι τώρα ο Michael Thielscher ο οποίος έχει κάνει ιδιαίτερη έρευνα πάνω στον τομέα του Γενικού Παιξίματος βοήθησε στην ανάπτυξη της Γλώσσας Περιγραφής Παιχνιδιών να αναγνωρίζει και παιχνίδια με ατελή πληροφορία και έτσι βοηθήθηκε παραπάνω η ανάπτυξη των Γενικών Παιχτών. [17].

2. ΓΕΝΙΚΟΙ ΠΑΙΚΤΕΣ

Σε αυτό το κεφάλαιο γίνεται εκτενής περιγραφή των χαρακτηριστικών ενός γενικού παίκτη. Τα χαρακτηριστικά ενός γενικού παίκτη, τα οποία θα μελετηθούν, είναι τα εξής [1]:

- Η περιγραφή του παιχνιδιού. Ο τρόπος επιλογής των νόμιμων κινήσεων και ο καθορισμός των αποτελεσμάτων τους στην κατάσταση του παιχνιδιού.
- Η αναζήτηση για την εύρεση της καλύτερης κίνησης.
- Η αξιολόγηση των καταστάσεων του παιχνιδιού.
- Οι ευριστικές.
- Η εξακρίβωση της πιθανής δομής αναπαράστασης ενός παιχνιδιού.

2.1 Περιγραφή Παιχνιδιών

Για να μπορέσει ένας γενικός παίκτης να δεχτεί τους κανόνες διαφόρων παιχνιδιών, θα πρέπει να αναπαρίστανται ακολουθώντας κάποια συγκεκριμένα πρότυπα. Για αυτό το σκοπό έχει δημιουργηθεί η γλώσσα αναπαράστασης παιχνιδιών (Game Description Language, GDL) [5]. Οι κανόνες κάθε παιχνιδιού πρέπει να περιγράφονται σε αυτή τη γλώσσα. Υπάρχουν πολλά είδη τέτοιων γλωσσών, αλλά η GDL είναι η πιο συχνή γλώσσα που χρησιμοποιείται. Σε παρακάτω κεφάλαιο γίνεται εκτενής παρουσίαση της γλώσσας GDL.

Η GDL είναι γλώσσα που αποτελεί επέκταση της Datalog με συναρτησιακά σύμβολα ενός επιπέδου, οπότε αποτελείται από λογικές προτάσεις.

Παρακάτω παρουσιάζεται μια μικρή περιγραφή του παιχνιδιού της τρίλιζας σε GDL ως παράδειγμα για την παρουσίαση των κυριότερων σημείων: [5]

- Σε κάθε παιχνίδι πρέπει να δηλώνονται οι **παίκτες**, χρησιμοποιώντας το κατηγόρημα $\boldsymbol{role}, \pi.\chi: role(x), role(o)$.
- Πρέπει να δηλώνεται η **αρχική κατάσταση** για τα διάφορα στοιχεία του παιχνιδιού, με τη χρήση του κατηγορήματος $\boldsymbol{init}$, π.χ $\mathrm{init}(\mathrm{cell}(1,1,\mathrm{b}))$, $\mathrm{init}(\mathrm{control}(\mathrm{x}))$, κτλ.
- Πρέπει να δηλώνονται οι **νόμιμες κινήσεις** που μπορεί να κάνει κάθε παίκτης σε κάθε γύρο, με τη χρήση των κατηγορημάτων $\boldsymbol{legal}$ και $\boldsymbol{true}$. π.χ. $\mathrm{legal}(\mathrm{P}, \mathrm{mark}(\mathrm{X}, \mathrm{Y})) :- \mathrm{true}(\mathrm{control}(\mathrm{P})), \mathrm{true}(\mathrm{cell}(\mathrm{X}, \mathrm{Y}, \mathrm{b}))$
- Πρέπει να δηλώνονται οι **συνέπειες** μιας κίνησης στην κατάσταση του παιχνιδιού, με τη χρήση των κατηγορημάτων $\boldsymbol{next}$ και $\boldsymbol{does}$, $(\pi.\chi: \mathrm{next}(\mathrm{cell}(\mathrm{X}, \mathrm{Y}, \mathrm{x})) :- \mathrm{does}(\mathrm{x}, \mathrm{mark}(\mathrm{X}, \mathrm{Y})))$.
- Πρέπει να δηλώνονται οι **τερματικές καταστάσεις** του παιχνιδιού, με τη χρήση του κατηγορήματος $\boldsymbol{terminal}$, $\pi.\chi: terminal :- line(x)\ ;\ line(o)\ ;\ not\ open$.
- Τέλος, πρέπει να δηλώνονται οι **στόχοι** του παιχνιδιού όταν τελειώσει, με τη χρήση του κατηγορήματος $\boldsymbol{goal}, \pi.\chi: goal(x, 100) :- line(x)$.

Σε επόμενο κεφάλαιο παρουσιάζεται εκτενέστερα το παραπάνω παράδειγμα.

2.2 Αναζήτηση

Ένας γενικό παίκτης δεν πρέπει απλά να επιλέγει μια απο τις διαθέσιμες έγκυρες κινήσεις και να «παίζει» εκείνη, θα πρέπει να επιλέγει την (όσο το δυνατόν) καλύτερη κίνηση από τις διαθέσιμες που έχει. Προκειμένου να γίνει εφικτό αυτό, θα πρέπει να μελετηθούν και οι μελλοντικές κινήσεις που προκύπτουν απο κάθε διαθέσιμη κίνηση, δηλαδή γίνεται αναζήτηση στο δέντρο κινήσεων.

Η αναζήτηση όμως διαφέρει ανάλογα με το είδος του εκάστοτε παιχνιδιού, για παράδειγμα, διαφορετική είναι η διαδικασία της αναζήτησης για παιχνίδια ενός παίκτη και διαφορετική για παιχνίδια δύο παικτών, όπως σκάκι, ντάμα, κτλ. Επίσης διαφέρει ανάλογα με το μέγεθος του παιχνιδιού, για παράδειγμα, η τρίλιζα έχει μικρό μέγεθος, καθώς ο συνολικός αριθμός των κινήσεων που απαιτούνται για να τελειώσει το παιχνίδι είναι μικρό και υπάρχουν λίγες διαθέσιμες κινήσεις σε κάθε γύρο, οπότε το δέντρο κινήσεων έχει μικρό βάθος με συνέπεια η διαδικασία της αναζήτησης να έχει τη δυνατότητα να σκανάρει ολόκληρο το δέντρο σε λογικό χρόνο, αντίθετα όμως, στο σκάκι υπάρχουν πολλές διαθέσιμες κινήσεις σε κάθε γύρο και συνήθως απαιτούνται πολλές κινήσεις μέχρι να τελειώσει το παιχνίδι, με αποτέλεσμα να είναι αδύνατο για τη διαδικασία της αναζήτησης να σκανάρει ολόκληρο το δέντρο κινήσεων.

Για όλους αυτούς τους λόγους υπάρχουν διάφορες τακτικές για την αναζήτηση στο δέντρο κινήσεων, όπως:

- Αλγόριθμοι αναζήτησης μέχρι κάποιο συγκεκριμένο βάθος.
- Ειδικοί αλγόριθμοι αναζήτησης με αντιπαλότητα (για παιχνίδια πολλαπλών παικτών)
- Πίνακες μετάθεσης (transposition tables) για την προσωρινή αποθήκευση καταστάσεων του παιχνιδιού, ώστε να συλλέγονται πληροφορίες γι’ αυτές χωρίς να γίνεται περαιτέρω αναζήτηση.
- Ταξινόμηση των διαθέσιμων κινήσεων κάθε γύρου ώστε να ελέγχονται πρώτες οι πιο «ενδιαφέρουσες».
- κ.α.

Οι παραπάνω τεχνικές είναι ιδανικές για ντετερμινιστικά παιχνίδια (όπως το σκάκι, η ντάμα και η τρίλιζα) αλλά όχι τόσο ιδανική για μη-ντετερμινιστικά παιχνίδια, όπου οι διαθέσιμες κινήσεις επηρεάζονται άμεσα απο παράγοντες τύχης (όπως στο τάβλι, όπου οι διαθέσιμες κινήσεις βασίζονται στο αποτέλεσμα που φέρνουν τα ζάρια). Τα τελευταία χρόνια έχει παρουσιαστεί μια νέο μέθοδος αναζήτησης, που είναι ιδανική στον τομέα του γενικού παιξίματος, τόσο για μη - ντετερμινιστικά παιχνίδια, όσο και για ντετερμινιστικά. Η μέθοδος αυτή ονομάζεται αναζήτηση δέντρου Monte - Carlo. Η μέθοδος αυτή συνδυάζει στοιχεία κλασσικών μεθόδων αναζήτησης δέντρου και προσομοιώσεων Monte - Carlo. Η επιτυχία της αναζήτησης Monte - Carlo στον τομέα του γενικού παιξίματος βασίζεται κυρίως δυσκολία εφαρμογής ευριστικών συναρτήσεων. Η αναζήτηση Monte - Carlo χρησιμοποιείται από πολλούς διάσημους γενικού παίκτες. [8]

2.2.1 Αναζήτηση Monte – Carlo

Η αναζήτηση δέντρου Monte - Carlo (Monte - Carlo Tree Search, MCTS) [8] είναι μια, βασισμένη σε προσομοιώσεις, μέθοδος αναζήτησης, που επεκτείνει τις απλές προσομοιώσεις Monte - Carlo ώστε να είναι προσαρμοσμένες για παιχνίδια. Οι προσομοιώσεις Monte - Carlo βασίζονται σε τυχαία δειγματοληψία για τον υπολογισμό

αποτελεσμάτων. Η αναζήτηση Monte - Carlo ξεκινάει τρέχοντας μια απλή προσομοίωση Monte - Carlo χτίζοντας όμως, σταδιακά, ένα δέντρο παιχνιδιού στη μνήμη με κάθε νέα προσομοίωση. Η διαδικασία περιέχει τέσσερα στρατηγικά βήματα: επιλογή, επέκταση, παίξιμο, διάδοση προς τα πίσω.

- Στο βήμα της **επιλογής**, διασχίζεται το δέντρο από την κορυφή μέχρι να βρεθεί κάποιος κόμβος-φύλλο.
- Όταν βρεθεί ο κόμβος-φύλλο στο βήμα της επιλογής, τότε ξεκινά το βήμα της **επέκτασης**. Στο βήμα της επέκτασης, ο κόμβος-φύλλο αυξάνεται κατά ένα επίπεδο, δηλαδή προστίθεται σε αυτόν, ένας κόμβος-παιδί.
- Ύστερα, το βήμα του **παιξίματος** τρέχετε από το νέο κόμβο, που δημιουργείται στο προηγούμενο βήμα, μέχρι το τέλος του παιχνιδιού ή μέχρι να υπάρξουν τερματικές συνθήκες.
- Τέλος, στο βήμα της **διάδοσης προς τα πίσω**, το αποτέλεσμα επιστρέφεται στην κορυφή του δέντρου, όπου και ανανεώνονται τα στατιστικά στο δέντρο.

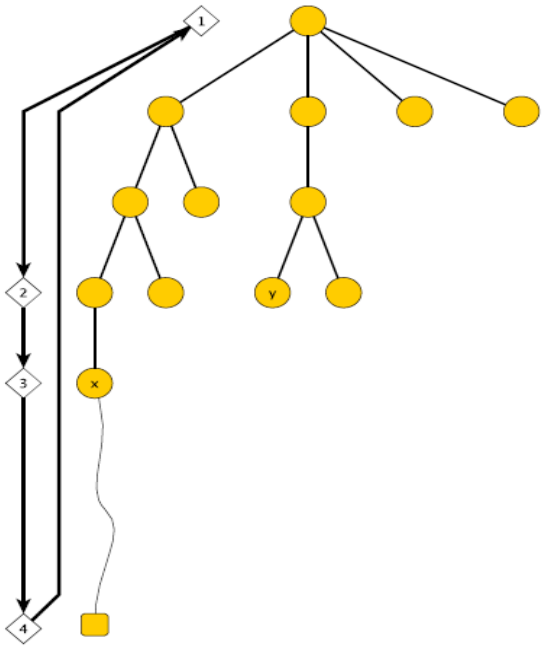
Η διαδικασία της αναζήτησης συνεχίζει να εκτελεί τα παραπάνω βήματα μέχρις ότου τελειώσει ο χρόνος μελέτης και επιλεχθεί η πιο πολύ υποσχόμενη κίνηση.

Η φόρμουλα που χρησιμοποιείται για την επιλογή της ιδανικής κίνησης είναι η εξής:

$$a^* = argmax_{a \in A(s)} \left\{ Q(s,a) + C \sqrt{\frac{lnN(s)}{N(s,a)}} \right\}$$

Οι συναρτήσεις και οι μεταβλητές που χρησιμοποιούνται στη φόρμουλα έχουν τις παρακάτω έννοιες:

- $\boldsymbol{N(s)}$: Ο αριθμός των δειγμάτων που συλλέχτηκαν στην κατάσταση s.
- $\boldsymbol{N(s,a)}$: Ο αριθμός των δειγμάτων που συλλέχτηκαν στην κατάσταση s όταν έχει επιλεχθεί η κίνηση a.
- $\boldsymbol{A(s)}$: Ο αριθμός των πιθανών κινήσεων στην κατάσταση s.
- $\boldsymbol{Q(s,a)}$: Η προβλεπόμενη επιστρεφόμενη τιμή για την επιλογή της κίνησης a στην κατάσταση s. Συνήθως είναι ο αριθμητικός μέσος των N(s, a) που συλλέγονται για την κίνηση a.
- $\boldsymbol{C}$: Ο όρος $\boldsymbol{Q(s,a)}$ δηλώνει το πόσο μεγάλη είναι η επιθυμία για εξερεύνηση. Η σταθερά C δηλώνει το πόση επίδραση έχει η εξερεύνηση ενάντια στην εκμετάλλευση. Με C = 0, τα δείγματα επιλέγονται άπληστα, επιλέγοντας πάντοτε την κίνηση με τη μεγαλύτερη αξιολόγηση για κάθε παίξιμο.



Εικόνα 1: Τα τέσσερα βήματα της αναζήτησης Monte-Carlo

Έστω ότι μετά το τρέξιμο καθορισμένου αριθμού προσομοιώσεων στο δέντρο, δύο από τις διαθέσιμες κινήσεις στην κορυφή έχουν καθοριστεί ως καλύτερες από τις υπόλοιπες, έστω με σκορ 0.85 και 0.88 αντίστοιχα. Σε ένα μη-ντετερμινιστικό παιχνίδι με ένα ουσιαστικό στοιχείο ευκαιρίας, ή ένα παιχνίδι στο οποίο οι αξιολογήσεις κυμαίνονται στο διάστημα [0.00, 1.00], θα μπορούσαν να γίνουν περισσότερες προσομοιώσεις ώστε να βρεθεί ποια από τις δύο κινήσεις είναι καλύτερη. Σε αντίθεση, σε ένα ντετερμινιστικό παιχνίδι, όπου τα αποτελέσματα είναι μόνο νίκη (=1) ή ήττα (=0), αυτό δεν αποτελεί λύση, γιατί και οι δύο κινήσεις είναι πιθανό να οδηγούν σε νικηφόρα κατάσταση και οποιαδήποτε κίνηση κι αν παιχτεί, το πραγματικό αποτέλεσμα είναι εξασφαλισμένο, οπότε αντί να σπαταλούνται πόροι για παραπάνω προσομοιώσεις με σκοπό την εύρεση της καλύτερης, εκ των δύο, κίνησης, οι πόροι αυτοί μπορούν να αξιοποιηθούν καλύτερα. Αν υπάρχουν μόνο αποτελέσματα νίκης ή ήττας, τότε μιας και οι τιμές του $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})$ πλησιάζουν σε κάποιο θεμιτό αποτέλεσμα με βάση αρκετές προσομοιώσεις, δεν είναι σοφό το ξόδεμα παραπάνω προσομοιώσεων για την επιλογή μεταξύ κοντινών τιμών. Επιπλέον, μόνο με αποτελέσματα νίκης, ήττας και ισοπαλίας, μόλις βρεθούν νικηφόρες γραμμές, τότε τα αποτελέσματα αργούν να διαδοθούν προς τα πίσω στο δέντρο γιατί οι τιμές των $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})$ βασίζονται σε μέσους όρους.

Για ένα μη - ντετερμινιστικό παιχνίδι, οι τιμές $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})$ μπορεί να θεωρούνται λογικοί εκτιμητές του πραγματικού αποτελέσματος του παιχνιδιού, ενώ για ντετερμινιστικά παιχνίδια με ελάχιστα διακριτά αποτελέσματα, οι τιμές αυτές δεν προσεγγίζουν σωστά το πραγματικό αποτέλεσμα. Σε ένα ντετερμινιστικό παιχνίδι, που οι τιμές αυτές προσεγγίζουν ένα από τα διακριτά αποτελέσματα, χρησιμοποιούνται δύο μέθοδοι για την εκτίμηση του αποτελέσματος: το **όριο επάρκειας** και ο **μέσος όρος κίνησης**.

Έστω ένα ντετερμινιστικό παιχνίδι στο οποίο υπάρχουν δύο κινήσεις με αξιολόγηση 0.75 και 0.70 αντίστοιχα, που σημαίνει ότι προσφέρουν πιθανότητες 75% και 70% νίκης, αντίστοιχα. Έστω ένα παράδειγμα από το σκάκι, στο οποίο ο λευκός παίκτης έχει την ευκαιρία να αιχμαλωτίσει έναν πύργο ή έναν ίππο. Έπειτα από αρκετές προσομοιώσεις, υπάρχουν υψηλές εκτιμήσεις και για τις δύο κινήσεις. Υπάρχει μεγάλη πιθανότητα και οι δύο κινήσεις να οδηγούν σε νίκη, δηλαδή σε αποτέλεσμα ίσο με 1, όμως είναι άκρως σημαντικό να αποκτηθούν πιο αξιόπιστες πληροφορίες για μία από τις κινήσεις παρά να γίνει προσπάθεια για επιλογή μεταξύ δύο, εξίσου καλών, κινήσεων.

Η εκτίμηση για μια κίνηση ενδέχεται, στη συνέχεια, να αυξηθεί, οπότε αυξάνεται και η εμπιστοσύνη, ή η εκτίμηση πέφτει, οπότε η αρχική εκτίμηση θεωρείται εσφαλμένη. Ορίζεται ως **όριο επάρκειας** μια σταθερά α, για την οποία ισχύει ότι αν υπάρχει εκτίμηση $Q(s,a) > \alpha$, τότε αποσυνδέεται η εξερεύνηση. Για να επιτευχθεί αυτό, αντικαθίσταται η σταθερά C με τη συνάρτηση $\hat{C}$, η οποία ορίζεται ως εξής:

$$\hat{C} = \begin{cases} C & when\ all\ Q(s,a) \leq a, \\ 0 & when\ any\ Q(s,a) > a. \end{cases}$$

Άμα η εκτίμηση πέσει κάτω από το α, τότε χρησιμοποιείται η αρχική διαδικασία.

Κατά τη συλλογή των πρώτων δειγμάτων, η εκτίμηση για μια κίνηση ενδέχεται να μην είναι σωστή, αλλά κατά τη συλλογή όλο και περισσότερων δειγμάτων, η εκτίμηση αυτή γίνεται όλο και πιο αξιόπιστη και κινείται προς το 0 ή το 1. Όσο λαμβάνονται όλο και περισσότερα δείγματα, είναι προτιμότερο να αφήνονται παλαιότερα δείγματα γιατί μπορεί να επισκιάσουν ξαφνικές ανόδους στην εκτίμηση μιας κίνησης. Προκειμένου να επιτευχθεί αυτό, χρησιμοποιείται η συνάρτηση λ και η φόρμουλα ανανέωσης του Q(s, a) μετατρέπεται στον εξής:

$$Q(s,a) = Q_{old}(s,a) + \lambda(r - Q_{old}(s,a))$$

Όπου r είναι το αποτέλεσμα της προσομοίωσης. Αυτή η μέθοδος αποτελεί τον **μέσο όρο κίνησης**. Λόγω του ότι η αρχική εκτίμηση παίζει πολύ σημαντικό ρόλο, ο μέσος όρος κίνησης δεν ξεκινάει μέχρι να έχει μαζευτεί ένα ελάχιστο ποσοστό δειγμάτων M. Με λιγότερα δείγματα χρησιμοποιείται ο αριθμητικός μέσος. Σύμφωνα με τα παραπάνω, η συνάρτηση λ ορίζεται ως εξής:

$$\lambda = \begin{cases} \dfrac{1}{N(s,\alpha)} & when\ N(s,a) \leq M, \\ \lambda_0 & when\ N(s,a) > M. \end{cases}$$

Όπου το λ_0 αποτελεί μια σταθερά.

2.3 Αξιολόγηση

Κάθε πρόγραμμα, το οποίο κατασκευάζεται με σκοπό να παίζει «καλά» κάποιο παιχνίδι, περιέχει συνάρτηση αξιολόγησης [7], η οποία φροντίζει ώστε να αξιολογούνται οι καταστάσεις του παιχνιδιού και να επιλέγεται η καλύτερη επιλογή σε κάθε γύρο του παιχνιδιού. Ένας γενικός παίκτης, λόγω του ότι δε γνωρίζει ποιο παιχνίδι θα κληθεί να παίξει, δε μπορεί να έχει συγκεκριμένη συνάρτηση αξιολόγησης, οπότε είναι αναγκαία η χρήση κάποια γενικής συνάρτησης αξιολόγησης. Αυτό επιτυγχάνεται κυρίως με τη χρήση ασαφούς λογικής.

Παρακάτω ακολουθούν κάποια γενικά βήματα που γίνονται για την επίτευξη της αξιολόγησης καταστάσεων:

- Για την κατασκευή της συνάρτησης αξιολόγησης γίνεται επεξεργασία στους στόχους (goals) που βρίσκονται στους κανόνες του παιχνιδιού. Εφόσον οι στόχοι περιγράφονται ως λογικές εκφράσεις, τότε γίνεται επεξεργασία σε αυτές τις εκφράσεις κι επιστρέφονται σε δεντρική δομή για περαιτέρω επεξεργασία. Η κατασκευή της δεντρικής δομής μπορεί να γίνεται με έναν αναδρομικό αλγόριθμο σα τον παρακάτω:

```
1 expand(true(Fluent), true(Fluent)) :- !.
2 expand(not(Expr), not(Child)) :- !,
3   expand(Expr, Child).
4 expand(Expr, and(Children)) :-
5   is_conjunction(Expr, Conjuncts), !,
6   expand_children(Conjuncts, Children).
7 expand(Expr, or(Children)) :-
8   is_disjunction(Expr, Disjuncts), !,
9   expand_children(Disjuncts, Children).
10 expand(Expr, R) :-
11   is_game_specific_pred(Expr, Expansions), !,
12   (Expansions = [OneExpansion] ->
13     expand(OneExpansion, R)
14   ;
15     R = or(Children)
16     expand_children(Expansions, Children)
17   ).
18 expand(Expr, reason(Expr)).
19 expand_children([], []).
20 expand_children([C|Children], [E|Expansions]) :-
21   expand(C, E),
22   expand_children(Children, Expansions).
```

Εικόνα 2: Αλγόριθμος κατασκευής δεντρικής δομής εκφράσεων

- Από τη στιγμή που έχει κατασκευαστεί η δεντρική δομή που αποτελεί το δέντρο αξιολόγησης, μπορεί πλέον να εφαρμοστεί μια διαδικασία η οποία λαμβάνει το δέντρο αξιολόγησης ως όρισμα, την κατάσταση του παιχνιδιού προς αξιολόγηση και επιστρέφει μια αριθμητική τιμή που αποτελεί την τιμή αξιολόγησης της συγκεκριμένης κατάστασης του παιχνιδιού. Η διαδικασία αυτή χρησιμοποιεί ασαφή λογική προκειμένου να υπολογίσει την τιμή αξιολόγησης. Η διαδικασία αυτή έχει τον παρακάτω σκελετό:

```
1 eval(not(Child), S, V) :-
2   eval(Child, S, V1), not(V1, V).
3 eval(true(Fluent), S, V) :-
4   fluent_holds(Fluent, S),
5   !, fz_true(V).
6 eval(true(_Fluent), _, V) :- fz_false(V).
7 eval(and([]), _, V) :- !, true(V).
8 eval(and([C|Children]), S, V) :-
9   eval(C, S, V1),
10  eval(and(Children), S, V2),
11  fz_and(V1, V2, V).
12 eval(or([]), _, V) :- !, false(V).
13 eval(or([C|Children]), S, V) :-
14  eval(C, S, V1),
15  eval(or(Children), S, V2),
16  fz_or(V1, V2, V).
17 eval(reason(Expr), S, V) :-
18  expression_holds(Expr, S),
19  !, fz_true(V).
20 eval(reason(_Expr), _, V) :- fz_false(V).
```

Εικόνα 3: Αλγόριθμος αξιολόγησης δεντρικής δομής εκφράσεων

Σύμφωνα με τις αρχές της ασαφούς λογικής, τίποτα δε θεωρείται πως είναι απόλυτα αληθές και τίποτα δε θεωρείται πως είναι απόλυτα ψευδές, οπότε αν κάποιος κανόνας ισχύει, του δίνεται η τιμή fz_true, ενώ αν δεν ισχύει, τότε του δίνεται η τιμή fz_false. Οι τιμές των $\boldsymbol{true}$ και $\boldsymbol{false}$ χρησιμοποιούνται ως άνω και κάτω όρια. Παρακάτω ακολουθεί η υλοποίηση των παραστάσεων της ασαφούς λογικής:

```
1 fz_true(0.9). true(1).
2 fz_false(0.1). false(0).
3 not(X, Y) :- Y is 1 - X.
4 fz_and(X, Y, Z) :- Z is Y * X.
5 fz_or(X, Y, Z) :- Z is X + Y - X * Y.
```

Εικόνα 4: Παραστάσεις ασαφούς λογικής

Η κατάσταση παιχνιδιού με τη μεγαλύτερη ασαφή τιμή, αποτελεί κατά πάσα πιθανότητα την καλύτερη επιλογή για το συγκεκριμένο γύρο.

- Στη διαδικασία κατασκευής του δέντρου αξιολόγησης, όταν μια έκφραση, που περιέχει μεταβλητές, δε μπορεί να ξεδιπλωθεί (expand) περαιτέρω, τότε πρέπει να συγκεκριμενοποιηθεί, δηλαδή να αντικατασταθεί από μια διάζευξη που περιέχει την ίδια έκφραση αλλά με συγκεκριμένη τιμή στη θέση της μεταβλητής. Αυτό είναι εφικτό, λόγω του ότι το πεδίο ορισμού κάθε μεταβλητής είναι πεπερασμένο. Αυτή η τακτική όμως έχει σαν αποτέλεσμα το μέγεθος του δέντρου αξιολόγησης να αυξάνεται εκθετικά, γι' αυτό το λόγο συνήθως εφαρμόζονται τεχνικές ώστε να αποφεύγεται ή να περιορίζεται η συγκεκριμενοποίηση εκφράσεων.

2.4 Ευριστικές

Στην ενότητα αυτή παρουσιάζονται ευριστικές που μπορούν να χρησιμοποιούνται απο γενικούς παίκτες.

2.4.1 Απλές Ευριστικές

Όπως αναφέρθηκε και στην προηγούμενη ενότητα, με την τεχνική της συγκεκριμενοποίησης εκφράσεων, το μέγεθος του δέντρου αξιολόγησης ενδέχεται να αυξάνεται εκθετικά. Για το λόγο αυτό, χρησιμοποιούνται ευριστικές για να μειωθεί το μέγεθος του δέντρου. Αντί να γίνεται πάντοτε συγκεκριμενοποίηση, σε κάποιες περιπτώσεις αποφεύγεται και αντ' αυτού εφαρμόζονται ευριστικές για να υπολογιστεί η τιμή αξιολόγησης της έκφρασης στην οποία θα εφαρμοζόταν συγκεκριμενοποίηση. Οι ευριστικές αποτελούν χαρακτηριστικά [7] τα οποία εμφανίζονται στους κανόνες των παιχνιδιών. Υπάρχουν δύο μορφές χαρακτηριστικών: τα χαρακτηριστικά εκφράσεων και τα χαρακτηριστικά στιγμιότυπων.

Τα χαρακτηριστικά εκφράσεων προέρχονται από εκφράσεις και κανόνες που περιέχουν μεταβλητές, στιγμιότυπα με μεταβλητές, συζεύξεις εκφράσεων που μοιράζονται ίδιες μεταβλητές. Μερικά από τα χαρακτηριστικά εκφράσεων είναι τα εξής:

- **Το χαρακτηριστικό του πλήθους λύσεων**. Το συγκεκριμένο χαρακτηριστικό μπορεί να εφαρμοστεί για εκφράσεις που δεν έχουν μεταβλητές ως βάση. Παράδειγμα αποτελεί η παρακάτω έκφραση:

```
1 open :- true(cell(X, Y, b)).
```

Εικόνα 5: Σχέση ύπαρξης ανοιχτού κελιού στην τρίλιζα

Η έκφραση αυτή αφορά το παιχνίδι της τρίλιζας. Η έκφραση αυτή μπορεί να είναι αληθής για οποιαδήποτε τιμή των X και Y. Σε τέτοιες περιπτώσεις, αντί να γίνεται συγκεκριμενοποίηση, θεωρείται η έκφραση πως είναι ψευδής αν δεν υπάρχει λύση ώστε να είναι αληθής, αλλιώς έχει μεγαλύτερη τιμή, ανάλογα με το πόσες

λύσεις υπάρχουν. Επίσης, πρέπει να γίνει η αλλαγή στον αλγόριθμο αξιολόγησης, ώστε, για παράδειγμα, να έχει παρόμοια μορφή με την εξής:

```
1 eval(true(cell(X, Y, b)), S, V) :-
2   count_solutions(cell(X, Y, b), S, SolCnt),
3   count_instances(cell(X, Y, b), Inst),
4   fz_generalized_or(SolCnt, Inst, V).
```

Εικόνα 6: Εφαρμογή χαρακτηριστικού πλήθους λύσεων

Το κατηγόρημα $\boldsymbol{count\_instances}$ επιστρέφει τα έγκυρα στιγμιότυπα της έκφρασης που λαμβάνει ως όρισμα (στη συγκεκριμένη περίπτωση 9, μιας και μπορούν να υπάρξουν ως και 9 κενά κελιά στην τρίλιζα). Το κατηγόρημα $\boldsymbol{fz\_generalized\_or}$ υπολογίζει την ασαφή τιμή που προκύπτει όταν κάτι μπορεί να είναι $\boldsymbol{Inst}$ – $\boldsymbol{SolCnt}$ φορές ψευδές και $\boldsymbol{SolCnt}$ φορές αληθές.

- **Το χαρακτηριστικό της διάταξης**. Έστω ότι παρακάτω στόχος για το παιχνίδι της ντάμας:

```
1 goal(white, 100) :-
2   true(piece_count(white, W)),
3   true(piece_count(black, B)),
4   greater(W, B).
```

Εικόνα 7: Στόχος στο παιχνίδι της ντάμας

Ο συγκεκριμένος στόχος παίρνει τον αριθμό των κομματιών του λευκού παίκτη και τον αριθμό των κομματιών του μαύρου παίκτη και ελέγχει αν ο λευκός παίκτης έχει περισσότερα κομμάτια από τον μαύρο παίκτη. Η σχέση που προκύπτει από το κατηγόρημα $\boldsymbol{greater}$ παριστάνει διάταξη, οπότε μπορεί να γίνει αντιστοίχηση των στοιχείων της στο σύνολο των φυσικών αριθμών. Το χαρακτηριστικό της διάταξης εμφανίζεται σε συζεύξεις όπου μοιράζονται μεταβλητές και τουλάχιστον μία μεταβλητή στιγμιότυπου εμφανίζεται σε κάποιο στατικό κατηγόρημα το οποίο παριστάνει διάταξη. Και σε αυτή την περίπτωση πρέπει να αλλάξει κατάλληλα ο αλγόριθμος αξιολόγησης για να λαμβάνει υπόψη του το συγκεκριμένο χαρακτηριστικό. Για παράδειγμα, ο παραπάνω στόχος θα έπρεπε να αξιολογηθεί ως εξής:

```
1 eval(order(goal(white, 100)), S, V) :-
2   fluent_holds(piece_count(white, W), S),
3   fluent_holds(piece_count(black, B), S),
4   term_to_int(greater_2, W, WI),
5   term_to_int(greater_2, B, BI),
6   domain_size(greater_2, Size),
7   V is 0.5 + (WI-BI)/(2*Size).
```

Εικόνα 8: Εφαρμογή χαρακτηριστικού της διάταξης

Το πλεονέκτημα αυτού του χαρακτηριστικού είναι ότι μια κατάσταση έχει μεγαλύτερη τιμή για όσο περισσότερα κομμάτια έχει ο λευκός παίκτης σε σχέση με το μαύρο παίκτη. Το μειονέκτημα του είναι ότι δεν υπάρχει εύκολος τρόπος έκφρασης της εκάστοτε τέτοιας σχέσης σε συνδυασμό με την προσκόλληση στις αρχές της ασαφούς λογικής.

- **Σχετικές Αποστάσεις Ι – Αποστάσεις Μεταξύ στιγμιότυπων**. Στιγμιότυπα που βρίσκονται σε μια σύζευξη και μοιράζονται κοινές μεταβλητές στις ίδιες θέσεις ορίσματος, είναι υποψήφιες για σχετικές αποστάσεις. Έστω ο παρακάτω στόχος στο παιχνίδι "**Pacman"** όπου ο pacman κερδίζει αν φτάσει σε ένα συγκεκριμένο σημείο το οποίο περιέχει την τελίτσα:

```
1 goal(inky, 100) :-
2   true(location(pacman, X, Y)),
3   true(location(inky, X, Y))).
```

Εικόνα 9: Στόχος σε παραλλαγή του pacman

Για να αναγνωριστεί αν πράγματι αποτελούν πρότυπο σχετικής απόστασης, πρέπει να ελεγχθεί αν μπορούν να υπολογιστούν αποστάσεις στις θέσεις ορίσματος που βρίσκονται οι προαναφερθείσες κοινές μεταβλητές. Για να γίνει αυτό πρέπει να ελεγχθεί το πως εξελίσσονται οι μεταβλητές στο χρόνο, για παράδειγμα αν το κατηγόρημα $\boldsymbol{location}$ αποτελεί ταμπλό όπου ο pacman μπορεί να κινείται βόρεια, τότε θα υπάρχει κανόνας όπως ο εξής:

```
1 next(location(pacman, X, Z)) :-
2   true(location(pacman, X, Y)),
3   a_predicate(Y, Z).
```

Εικόνα 10: Αλλαγή κατάστασης σε παραλλαγή του pacman.

Αναγνωρίζοντας στατικά κατηγορήματα, όπως το $\boldsymbol{a\_predicate}$, μπορεί να κατασκευαστεί γράφος, του οποίου τα άκρα αποτελούν όλα τα έγκυρα στιγμιότυπα του $\boldsymbol{a\_predicate}$. Ακόμα, η σχέση που παρουσιάζεται από το $\boldsymbol{a\_predicate}$ παριστάνει διάταξη, οπότε παρόμοια με το χαρακτηριστικό της διάταξης, ο στόχος θα μπορούσε να αξιολογηθεί ως εξής:

```
1 eval(relative_dist(goal(inky, 100)), S, V) :-
2   fluent_holds(location(pacman, XP, YP), S),
3   fluent_holds(location(inky, XI, YI), S),
4   term_to_int(location_3_arg_2, XP, XPI),
5   term_to_int(location_3_arg_2, XI, XII),
6   domain_size(location_3_arg_2, XSize),
7   XDist is abs(XPI-XII)/XSize,
8   term_to_int(location_3_arg_3, YP, YPI),
9   term_to_int(location_3_arg_3, YI, YII),
10  domain_size(location_3_arg_3, YSize),
11  YDist is abs(YPI-YII)/YSize,
12  metric(XDist, YDist, Dist),
13  V is 1-Dist.
```

Εικόνα 11: Εφαρμογή του χαρακτηριστικού αποστάσεων μεταξύ στιγμιότυπων.

Γίνεται υπολογισμός των αποστάσεων ανά όρισμα, συνδυάζονται σύμφωνα με κάποια μετρική και επιστρέφεται μεγαλύτερη τιμή όσο πιο μικρές είναι οι αποστάσεις.

Αν το $\boldsymbol{a\_predicate}$ ήταν τέτοιο ώστε να μην παριστάνει διάταξη, τότε ο γράφος πρέπει να είναι διαφορετικός, πιο συγκεκριμένα, τα άκρα του γράφου αποτελούν οι σχέσεις που δίνονται από το $\boldsymbol{a\_predicate}$. Η μικρότερη απόσταση μεταξύ δύο στοιχείων του γράφου αποτελεί την απόσταση. Αν δεν υπάρχει μονοπάτι, τότε η απόσταση θεωρείται άπειρη. Αντί για το μέγεθος του πεδίου ορισμού (domain

size) χρησιμοποιείται η μέγιστη κοντινότερη απόσταση μεταξύ δύο στοιχείων, για την κανονικοποίηση της απόστασης.

Τα χαρακτηριστικά στιγμιότυπων προέρχονται από στιγμιότυπα και τροποποιούν τις εκφράσεις της μορφής $\boldsymbol{true}(\boldsymbol{Fluent})$.

- **Σχετικές Αποστάσεις ΙΙ – Απόσταση από καθορισμένα στιγμιότυπα**. Αποτελεί ειδίκευση του χαρακτηριστικού σχετικής απόστασης μεταξύ δύο στιγμιότυπων και αναφέρεται στη σχετική απόσταση από ένα καθορισμένο σημείο. Στο χαρακτηριστικό σχετικής απόστασης μεταξύ δύο στιγμιότυπων υπάρχουν στιγμιότυπα τα οποία μοιράζονται κοινές μεταβλητές στις ίδιες θέσεις ορίσματος, ενώ αντίθετα στο χαρακτηριστικό απόστασης από καθορισμένα στιγμιότυπα, ελέγχεται ένα μόνο στιγμιότυπο το οποίο δεν έχει μεταβλητές, όπως το παρακάτω:

```
timeout :- true(step(50)).
```

Εικόνα 12 : Στιγμιότυπο χωρίς μεταβλητές.

Το όρισμα στο κατηγόρημα $\boldsymbol{step}$ αυξάνεται με βάση κάποιο στατικό κατηγόρημα, παρόμοιο με το $\boldsymbol{a\_predicate}$ που αναφέρθηκε προηγουμένως. Ακόμα ένα παράδειγμα αποτελεί η απλοποίηση του pacman, στην οποία ο pacman κερδίζει αν φτάσει σε ένα συγκεκριμένο σημείο:

```
goal(pacman, 100) :-
  true(location(pacman, 8, 8)).
```

Εικόνα 13: Στόχος σε παραλλαγή του pacman

Αυτό το πρότυπο μπορεί να επεκταθεί και στα υπόλοιπα στιγμιότυπα, έτσι ώστε για κάθε στιγμιότυπο το οποίο αξιολογείται από το $\boldsymbol{fluent\_holds}$, να ελέγχεται διάταξη σε ορισμένα απο τα ορίσματα του, και αν, πράγματι, υπάρχει διάταξη, τότε να γίνεται εκτίμηση της απόστασης στο στιγμιότυπο. Η δύναμη της εκτίμησης της απόστασης έγκειται στο γεγονός ότι, ενώ μόνο το $\boldsymbol{location}(\boldsymbol{pacman}, \mathbf{8}, \mathbf{8})$ υπάρχει ως στόχος, μπορεί να εξαχθεί χρήσιμη πληροφορία από άλλα στιγμιότυπα της μορφής $\boldsymbol{location}(\boldsymbol{pacman}, \boldsymbol{X}, \boldsymbol{Y})$ τα οποία ισχύουν στην εκάστοτε κατάσταση του παιχνιδιού. Τα στιγμιότυπα των οποίων τα ορίσματα X και Y είναι πιο κοντά στο 8, είναι πιο πιθανά να οδηγούν στον επιθυμητό στόχο.

Συγκεκριμενοποιημένα στιγμιότυπα υπάρχουν σχεδόν σε κάθε στόχο, γι' αυτό το λόγο μπορούν να συμβούν σφάλματα κατανόησης όπως στην παρακάτω περίπτωση:

```
next(cell(X, Y2, red)) :-
  does(red, drop(X)),
  cell(X, Y2, blank),
  succ(Y1, Y2),
  cell(X, Y1, red).
```

Εικόνα 14: Μετάβαση σε νέα κατάσταση (χαρακτηριστικό απόστασης απο καθορισμένα στιγμιότυπα)

Το νόημα του παραπάνω κανόνα είναι ότι αν σε κάποιο κελί υπάρχει κόκκινο, τότε και στο από πάνω του κελί μπορεί να μπει κόκκινο. Σύμφωνα όμως με την προηγούμενη διατύπωση του χαρακτηριστικού, ένας γενικός παίκτης το αντιλαμβάνεται ως ότι το κόκκινο μετακινείται από το ένα κελί στο άλλο, το οποίο είναι λάθος. Ένα ακόμα πρόβλημα που μπορεί να προκύψει φαίνεται στην παρακάτω περίπτωση:

```
goal(x, 100) :- true(cell(Y, 8, x)).
```

Εικόνα 15: Στόχος (χαρακτηριστικό απόστασης από καθορισμένα στιγμιότυπα)

Σε αυτή την περίπτωση, ο παίκτης νικάει αν κάποιο από τα πιόνια του (που συμβολίζονται με X) φτάσει σε κάποιο κελί (που συμβολίζεται με Y) της $8^{ης}$ σειράς. Και σε αυτή την περίπτωση υπάρχουν κατηγορήματα τα οποία παριστάνουν διάταξη για τα δύο πρώτα ορίσματα του στιγμιότυπου, γεγονός που επιτρέπει τον υπολογισμό αποστάσεων. Σε αυτό το σημείο υπάρχει σύγχυση για το ποιο χαρακτηριστικό να χρησιμοποιηθεί, το χαρακτηριστικό πλήθους λύσεων ή το χαρακτηριστικό απόστασης από καθορισμένα στιγμιότυπα. Η λύση σε αυτό το πρόβλημα είναι η εφαρμογή του χαρακτηριστικού πλήθους λύσεων μόνο όταν όλα τα διατεταγμένα ορίσματα είναι μεταβλητές.

- **Το χαρακτηριστικό της επιμονής**. Παρατηρώντας τον παρακάτω κανόνα στην περιγραφή των κανόνων της τρίλιζας:

```
next(cell(X, Y, C)) :-
  true(cell(X, Y, C)), C \= b.
```

Εικόνα 16: Αλλαγή κατάστασης στην τρίλιζα

γίνεται αντιληπτό ότι αν ένα κελί μαρκαριστεί είτε με $\boldsymbol{x}$, είτε με $\boldsymbol{o}$, τότε παραμένει μαρκαρισμένο για όλη τη διάρκεια του παιχνιδιού. Το χαρακτηριστικό αυτό ονομάζεται χαρακτηριστικό της επιμονής επειδή αναφέρεται σε στιγμιότυπα τα οποία επιμένουν στην κατάσταση τους για όλη τη διάρκεια του παιχνιδιού. Τα στιγμιότυπα, σε αυτό το χαρακτηριστικό διακρίνονται σε δύο κατηγορίες: στα αληθή επίμονα και στα ψευδή επίμονα. Για τα αληθή επίμονα στιγμιότυπα ισχύει ότι αν ισχύουν σε κάποια κατάσταση του παιχνιδιού, τότε ισχύουν και για όλες τις καταστάσεις διαδόχους, ενώ αντίθετα, για τα ψευδή επίμονα στιγμιότυπα ισχύει ότι αν δεν ισχύουν σε κάποια κατάσταση, τότε δεν ισχύουν και για όλες τις καταστάσεις διαδόχους. Τα επίμονα στιγμιότυπα έχουν μεγαλύτερες επιπτώσεις σε μελλοντικές καταστάσεις του παιχνιδιού, οπότε ο αλγόριθμος αξιολόγησης μπορεί να τροποποιηθεί ως εξής, εκμεταλλευμένος τα επίμονα στιγμιότυπα:

```
eval(true(Fluent), S, V) :-
  fluent_holds(Fluent, S), !,
  (persistent_true(Fluent) -> true(V);
  fz_true(V)).
eval(true(Fluent), _, V) :-
  (persistent_false(Fluent) -> false(V);
  fz_false(V)).
```

Εικόνα 17: Εφαρμογή χαρακτηριστικού της επιμονής

.

Επίσης, τα επίμονα στιγμιότυπα μπορούν να χρησιμοποιηθούν προκειμένου να επιταχυνθεί η αξιολόγηση, όπως με το εξής: Αν η κατάφαση κάποιου ψευδούς επίμονου στιγμιότυπου βρίσκεται σε κάποια σύζευξη, τότε η σύζευξη θεωρείται αυτόματα ψευδής, χωρίς να χρειάζεται να αξιολογηθούν περαιτέρω τα υπόλοιπα μέλη της σύζευξης:

```
1 eval(and([]), S, V) :- true(V).
2 eval(and([C|Children], S, V) :-
3   eval(C, S, V1),
4   (V1 == 0 ->
5     V = 0
6   ;
7     eval(and(Children), S, V2),
8     fz_and(V1, V2, V)
9   ).
```

Εικόνα 18: Εφαρμογή χαρακτηριστικού της επιμονής στην αξιολόγηση σύζευξης

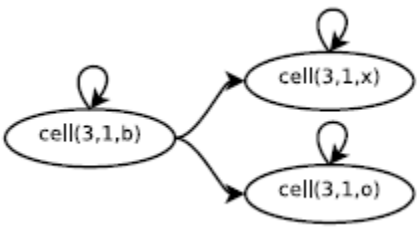
Ανάλογη λογική ακολουθείται και για τις διαζεύξεις. Οι εκτιμήσεις απόστασης που επιστρέφουν άπειρη απόσταση, θεωρούνται ψευδή επίμονα στιγμιότυπα.

Στην επόμενη ενότητα ακολουθούν περισσότερες λεπτομέρειες πάνω στις ευριστικές αποστάσεων.

2.4.2 Ευριστικές Αποστάσεων

Στην προηγούμενη ενότητα παρουσιάστηκαν κάποια χαρακτηριστικά αποστάσεων. Στην ενότητα αυτή παρουσιάζονται γενικές μέθοδοι για την κατασκευή χαρακτηριστικών αποστάσεων [5], μελετώντας τους κανόνες του παιχνιδιού και βρίσκοντας εξαρτήσεις μεταξύ τους.

Στόχος είναι η απόκτηση γνώσης για το πως εξελίσσονται τα στιγμιότυπα κατά τη διάρκεια του παιχνιδιού. Η διαδικασία ξεκινά με την κατασκευή ενός γράφου στιγμιότυπων, ο οποίος περιέχει όλα τα στιγμιότυπα των κανόνων του παιχνιδιού, σαν κόμβους στο γράφο. Στη συνέχεια δημιουργούνται κατευθυνόμενες ακμές $(\boldsymbol{fi}, \boldsymbol{f})$ αν τουλάχιστον ένα στιγμιότυπο – πρόγονος απαιτείται να ισχύει, στην τωρινή κατάσταση του παιχνιδιού, ώστε να ισχύει και το στιγμιότυπο f, στην τωρινή κατάσταση του παιχνιδιού. Η παρακάτω εικόνα απεικονίζει ένα κομμάτι γράφου στιγμιότυπων για το παιχνίδι της τρίλιζας:



Εικόνα 19: Γράφος στιγμιότυπων στην τρίλιζα

Για να είναι το κελί (3,1) κενό (b) θα πρέπει να ήταν κενό και προηγουμένως. Για να είναι ένα κελί x ή o, θα πρέπει να ισχύει μια από τις ακόλουθες συνθήκες:

- Να ήταν x (ή o αντίστοιχα) προηγουμένως.
- Να ήταν κενό (b) προηγουμένως.

Χρησιμοποιώντας το γράφο αυτό βγαίνει το συμπέρασμα ότι, για παράδειγμα, μια μετάβαση από το $cell(3,1,b)$ στο $cell(3,1,x)$ είναι δυνατή σε ένα βήμα, ενώ η μετάβαση από το $cell(3,1,o)$ στο $cell(3,1,x)$ είναι αδύνατη.

Ο ορισμός του γράφου στιγμιότυπων είναι ο εξής: Έστω Γ ένα παιχνίδι πάνω σε όρους Σ. Ένας γράφος G = (V, E) ονομάζεται γράφος στιγμιότυπων του παιχνιδιού Γ αν και μόνο αν:

- $V = \Sigma \cup \{ \emptyset \}$
- για κάθε στιγμιότυπο $f \in \Sigma$ και δύο έγκυρα στιγμιότυπα s και s', ισχύει:

  (το s' είναι απόγονος του s) Λ f $\in \Sigma \rightarrow (\exists f)$ (f, f') $\in$ E Λ (f $\in$ s $\cup \{ \emptyset \}$).

Στον ορισμό αυτό προστίθεται ένας επιπλέον κόμβος ∅ και επιτρέπεται το ∅ να αποτελεί την πηγή των ακμών. Ο λόγος είναι ότι ενδέχεται να υπάρχουν στιγμιότυπα του παιχνιδιού που να μην έχουν προϋποθέσεις, όπως για παράδειγμα στο στιγμιότυπο g με τον εξής κανόνα $\boldsymbol{next}$:

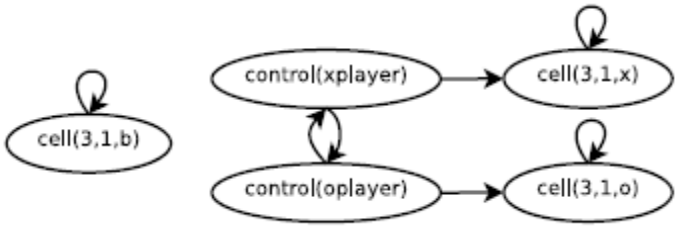
$$\boldsymbol{next}(g) :- \boldsymbol{distinct}(a, b)$$

Στην αντίπερα όχθη, ενδέχεται να υπάρχουν στιγμιότυπα που να μη μπορούν να είναι αληθή σε καμία κατάσταση του παιχνιδιού, λόγω του ότι το σώμα του κανόνα next, που τα αφορά, είναι μη – ικανοποιήσιμος. Ένα παράδειγμα αποτελεί ένα στιγμιότυπο g με τον παρακάτω κανόνα $\boldsymbol{next}$:

$$\boldsymbol{next}(g) :- \boldsymbol{distinct}(a, a)$$

Τα στιγμιότυπα τα οποία δεν έχουν προϋποθέσεις ξεχωρίζονται απο αυτά που είναι μη – ικανοποιήσιμα, συνδέοντας τα πρώτα με τον κόμβο ∅, ενώ τα δεύτερα δεν έχουν ακμές στον γράφο στιγμιότυπων.

Ο παραπάνω ορισμός καλύπτει ορισμένες μόνο απαραίτητες προϋποθέσεις για στιγμιότυπα, ως εκ τούτου, ένας γράφος στιγμιότυπων δεν είναι απαραίτητα μοναδικός, όπως φαίνεται στην παρακάτω εικόνα, που απεικονίζει ένα διαφορετικό γράφο για το παιχνίδι της τρίλιζας:



Εικόνα 20: Διαφορετικός γράφος στιγμιότυπων στην τρίλιζα

Η συνάρτηση απόστασης $\Delta(s, f')$ μεταξύ της τωρινής κατάστασης s και μια κατάστασης στην οποία ισχύει το f' ορίζεται ως εξής: Έστω $\Delta G(f, f')$ το ελάχιστο μονοπάτι μεταξύ του κόμβου f και του κόμβου f' ή ∞ αν δεν ορίζεται τέτοιο μονοπάτι. Τότε:

$$\Delta(s, f') = \min_{f \in s \cup \{ \emptyset \}} \Delta_G(f, f')$$

Στην ουσία, ως $\Delta(s, f')$ υπολογίζεται το ελάχιστο μονοπάτι από οποιοδήποτε στιγμιότυπο στην κατάσταση s προς το f'. Ωστόσο, ο γράφος στιγμιότυπων αποτελεί απλοποίηση του πραγματικού παιχνιδιού, μιας και πολλές προϋποθέσεις, για τις μεταβάσεις μεταξύ καταστάσεων, αγνοούνται. Αυτό έχει ως αποτέλεσμα η απόσταση $\Delta(s, f')$ να χρησιμοποιείται ως κάτω όριο για τον πραγματικό αριθμό βημάτων που χρειάζονται για τη μετάβαση από την κατάσταση s σε μια κατάσταση στην οποία το f'

ισχύει, οπότε η απόσταση $\Delta(s, f')$ αποτελεί αποδεκτή ευριστική για τη μετάβαση στο f' από μια κατάσταση s.

Έστω οι παρακάτω υποθέσεις:

- Ένα παιχνίδι $\Gamma = (R, s0, T, l, u, g)$ με όρους Σ και καταστάσεις S.
- Μια κατάσταση $s_1 \in S$ του παιχνιδιού Γ.
- Ένα στιγμιότυπο $f \in \Sigma$ του παιχνιδιού Γ.
- Ένας γράφος στιγμιότυπων $G = (V, E)$ του παιχνιδιού Γ.
- $s1 \rightarrow s2 \rightarrow \ldots \rightarrow sm + 1$ μια νόμιμη ακολουθία από καταστάσεις του παιχνιδιού Γ, έτσι ώστε για κάθε i, από 0 έως m, να υπάρχει μια συνδυαστική ενέργεια A_i, τέτοια ώστε:

$$s_{i+1} = u(A_i, s_i) \wedge (\forall r \in R)\ l(r, A_i(r), s)$$

Αν $\Delta(s_1, f) = n$, τότε δεν υπάρχει νόμιμη ακολουθία από καταστάσεις $s1 \rightarrow s2 \rightarrow \ldots \rightarrow sm + 1$ με $f \in s_{m+1}$ και $m < n$.

Στη συνέχεια ακολουθεί ένας αλγόριθμος για την κατασκευή γράφου στιγμιότυπων βασισμένου στους κανόνες παιχνιδιού. Για κάθε συγκεκριμενοποιημένο στιγμιότυπο f' του παιχνιδιού:

- Κατασκευάζεται μια συγκεκριμενοποιημένη κανονική μορφή φ του $next(f')$. Μια φόρμουλα φ τέτοια ώστε $next(f') \supset \varphi$.
- Για κάθε κομμάτι διάζευξης ψ στο φ:
  - Γίνεται επιλογή ενός όρου $\boldsymbol{true(f)}$ απο το ψ ή τίθεται $f = \emptyset$ αν δεν υπάρχει κανένα.
  - Γίνεται προσθήκη της ακμής (f, f') στο γράφο στιγμιότυπων.

Επιλέγεται ένας μόνο όρος από κάθε κομμάτι διάζευξης στο φ, μιας και η συνάρτηση απόστασης $\Delta(s, f')$ που αποκτιέται από τον γράφο είναι αποδεκτή. Στόχος είναι η κατασκευή ενός γράφου στιγμιότυπων, ο οποίος αυξάνει τα μήκη των ελάχιστων μονοπατιών μεταξύ των στιγμιότυπων, όσο περισσότερο γίνεται. Ο αλγόριθμος όμως αφήνει κάποια ανοιχτά ζητήματα:

- Πως κατασκευάζεται η συγκεκριμενοποιημένη φόρμουλα φ που αποτελεί την διαζευκτική κανονική μορφή (Disjunctive Normal Form, DNF) του $next(f')$;
- Ποιος όρος $\boldsymbol{true(f)}$ επιλέγεται αν υπάρχουν περισσότεροι από ένας;

Η φόρμουλα φ είναι ένα σύνολο από φόρμουλες $\{\psi 1, \psi 2, \ldots, \psi n\}$ συνδεδεμένες απο διαζεύξεις έτσι ώστε κάθε φόρμουλα ψi να αποτελεί ένα σύνολο από όρους που συνδέονται από συζεύξεις. Ο αλγόριθμος κατασκευής της φόρμουλας φ παρουσιάζεται στην παρακάτω εικόνα (Εικόνα 21: Αλγόριθμος κατασκευής φόρμουλας φ):

Input: game description D, ground fluent f'
Output: ϕ, such that $\texttt{next}(f') \supset \phi$

1: $\phi := \texttt{next}(f')$
2: $finished := false$
3: **while** $\neg finished$ **do**
4: Replace every positive occurrence of $\texttt{does}(r, a)$ in ϕ with $\texttt{legal}(r, a)$.
5: Select a positive literal l from ϕ such that $l \neq \texttt{true}(t), l \neq \texttt{distinct}(t_1, t_2)$ and l is not a recursively defined predicate.
6: **if** there is no such literal **then**
7: $finished := true$
8: **else**
9: $\hat{l} := \bigvee_{h:-b \in D, l\sigma = h\sigma} b\sigma$
10: $\phi := \phi\{l/\hat{l}\}$
11: **end if**
12: **end while**
13: Transform ϕ into disjunctive normal form, i.e., $\phi = \psi_1 \vee \ldots \vee \psi_n$ and each formula ψ_i is a conjunction of literals.
14: **for all** ψ_i in ϕ **do**
15: Replace ψ_i in ϕ by a disjunction of all ground instances of ψ_i.
16: **end for**

Εικόνα 21: Αλγόριθμος κατασκευής φόρμουλας φ

Η διαδικασία ξεκινά θέτοντας $\varphi = next(f')$ κι έπειτα επιλέγει ένα θετικό όρο l στο φ και τον αντικαθιστά με τα σώματα όλων των κανόνων, των οποίων η κεφαλή ξεκινά με το l. Αυτή η διαδικασία της αντικατάστασης επαναλαμβάνεται ως ότου όλα τα κατηγορήματα που έχουν απομείνει είναι είτε της μορφής $true(t)$, $distinct(t1, t2)$ είτε ορίζονται αναδρομικά. Τα κατηγορήματα τα οποία ορίζονται αναδρομικά δεν υφίστανται την προηγούμενη διαδικασία προκειμένου να διασφαλιστεί ο ασφαλής τερματισμός του αλγορίθμου. Τέλος, γίνεται μετατροπή της φ σε διαζευκτική κανονική μορφή και αντικαθίστανται κάθε κομμάτι διάζευξης ψi της φ με μια διάζευξη όλων των συγκεκριμενοποιημένων στιγμιότυπων έτσι ώστε να δημιουργηθεί μια συγκεκριμενοποιημένη φόρμουλα φ.

Ας σημειωθεί επίσης ότι στη γραμμή 4, γίνεται αντικατάσταση κάθε εμφάνισης του ***does*** με ***legal***, ώστε να συμπεριληφθούν οι προυποθέσεις των ενεργειών που εκτελούνται στη φ. Αυτό έχει σαν αποτέλεσμα, η παραχθείσα φόρμουλα φ να μην είναι ίδια με το $next(f\text{‘})$. Ωστόσο, ισχύει ότι $next(f\text{‘}) \supset \varphi$, με την προυπόθεση ότι μόνο νόμιμες $(legal)$ κινήσεις είναι δυνατό να εκτελούνται, δηλαδή ότι $does(r, \alpha) \supset legal(r, \alpha)$, το οποίο είναι αναγκαία συνθήκη για την κατασκευή γράφου στιγμιότυπων από το φ.

Επίσης, δεν επιλέγονται αρνητικοί όροι για τη διαδικασία αντικατάστασης, επειδή αυτό θα έκανε πιο περίπλοκο τον αλγόριθμο, θα αύξανε το μέγεθος της παραγόμενης φόρμουλας φ και θα γέμιζε με αρνητικές προϋποθέσεις την φ, οι οποίες δε χρησιμοποιούνται από το γράφο στιγμιότυπων.

Όπως αναφέρθηκε προηγουμένως, αν σε κάποιο κομμάτι διάζευξης υπάρχουν παραπάνω από ένας όροι $true(f)$, τότε πρέπει να επιλεχθεί μόνο ένας ως πηγή της ακμής στο γράφο στιγμιότυπων. Η απόσταση $\Delta(s, f)$, που υπολογίζεται με βάση το γράφο στιγμιότυπων, είναι το κάτω όριο των βημάτων που χρειάζονται για τη μετάβαση. Ένα καλό κάτω όριο πρέπει να είναι όσο το δυνατό μεγαλύτερο. Η επιλογή του

καλύτερου γράφου στιγμιότυπων, δηλαδή εκείνου που μεγιστοποιεί τις αποστάσεις, αποτελεί ακατόρθωτο έργο. Το ποιος γράφος είναι καλύτερος εξαρτάται από τις καταστάσεις του παιχνιδιού που βρίσκονται, κατά τη διάρκεια του παιχνιδιού, αλλά η εκ των προτέρων γνώση αυτών των καταστάσεων του παιχνιδιού δεν είναι δυνατή. Προκειμένου να επιλεχθεί κάποιος ικανοποιητικός γράφος στιγμιότυπων, ο οποίος κάνει καλές εκτιμήσεις αποστάσεων, χρησιμοποιούνται διάφορες ευριστικές για την επιλογή των όρων από τις διαζεύξεις της διαζευκτικής κανονικής μορφής του $next(f\ ')$.

Κατ' αρχάς, προστίθενται νέες ακμές μόνο αν είναι απαραίτητο, δηλαδή, αν υπάρχει ένας όρος $true(f)$, σε κάποιο κομμάτι διάζευξης, τέτοιος ώστε η ακμή $(f, f\ ')$ να υπάρχει ήδη στο γράφο στιγμιοτύπων, τότε επιλέγεται εκείνος ο όρος $true(f)$. Η λογική πίσω από αυτή την ευριστική είναι ότι, κατά μέσο όρο, τα μονοπάτια στο γράφο στιγμιότυπων είναι μεγαλύτερα, όσο λιγότερες συνδέσεις υπάρχουν μεταξύ των κόμβων. Επίσης, προτιμάται ο όρος $true(f)$ έναντι του όρου $true(g)$ αν το f έχει περισσότερες ομοιότητες με το $f\ '$, απ' ότι το g με το $f\ '$, δηλαδή $sim(f, f\ ') > sim(g, f\ ')$. Η συνάρτηση ομοιότητας sim ορίζεται ως εξής:

$$sim(t,t') = \begin{cases} 1 & \tau\alpha\ t, t'\ \acute{\epsilon}\chi ov\nu\ \beta\alpha\theta\mu\acute{o}\ 0\ \kappa\alpha\iota\ t = t' \\ \sum_i sim(t,t') & t = f(t_1, t_2, \dots, t_n)\ \kappa\alpha\iota\ t' = f(t'_1, t'_2, \dots, t'_n) \\ 0 & \Delta\iota\alpha\phi o\rho\epsilon\tau\iota\kappa\acute{\alpha} \end{cases}$$

Σε περιγραφές παιχνιδιού, κατασκευασμένες από ανθρώπους, όμοια στιγμιότυπα έχουν ισχυρούς δεσμούς. Για παράδειγμα, στην τρίλιζα, το στιγμιότυπο cell(3, 1, x) είναι περισσότερο όμοιο με το $cell(3, 1, b)$ απ' ότι με το $cell(b, 1, x)$. Χρησιμοποιώντας όμοια στιγμιότυπα, κατά την προσθήκη νέων ακμών στο γράφο στιγμιότυπων, υπάρχει μεγαλύτερη πιθανότητα εύρεσης του ίδιου στιγμιότυπου ξανά σε διαφορετικό κομμάτι διάζευξης της φ, με αποτέλεσμα τη μεγιστοποίηση της πιθανότητας επαναχρησιμοποίησης ακμών.

Για να χρησιμοποιηθεί η συνάρτηση απόστασης στη συνάρτηση αξιολόγησης, ορίζεται η κανονικοποιημένη απόσταση $\delta(s, f)$ ως εξής:

$$\delta(s,f) = \frac{\Delta(s,f)}{\Delta_{max}(s,f)}$$

Η τιμή $\Delta_{max}(s, f)$ αποτελεί τη μεγαλύτερη απόσταση $\Delta G(g, f)$ από οποιοδήποτε στιγμιότυπο g στο f. Η τιμή $\Delta_{max}(s, f)$ αποτελεί τη μεγαλύτερη δυνατή απόσταση $\Delta(s, f)$ που δεν είναι άπειρη. Η κανονικοποιημένη απόσταση $\delta(s, f)$ είναι άπειρη αν $\Delta(s, f) = \infty$, δηλαδή αν δεν υπάρχει μονοπάτι απο κάποιο στιγμιότυπο στην κατάσταση s, προς το f, στο γράφο στιγμιότυπων. Σε κάθε άλλη περίπτωση ισχύει ότι $0 \leq \delta(s, f) \leq 1$. Αξίζει να σημειωθεί, ότι η κατασκευή του γράφου στιγμιοτύπων και ο υπολογισμός της ελάχιστης απόστασης μεταξύ όλων των στιγμιοτύπων, δηλαδή η συνάρτηση απόστασης Δg, χρειάζεται να γίνουν μόνο μια φορά για κάθε παιχνίδι. Η κατασκευή του γράφου στιγμιότυπων είναι πιο δαπανηρή όσο πιο περίπλοκο είναι ένα παιχνίδι, ενώ το κόστος του υπολογισμού του χαρακτηριστικού απόστασης $\delta(s, f)$ (ή $\Delta(s, f)$) εξαρτάται (γραμμικά) μόνο απο το μέγεθος της κατάστασης s.

2.5 Εξακρίβωση Δομής

Οι ευριστικές συναρτήσεις που αναφέρθηκαν προηγουμένως αξιολογούν μια κατάσταση με έναν αριθμό από 0 έως 1. Για καλύτερη αξιολόγηση, μπορεί να γίνει εξακρίβωση της δομής του παιχνιδιού (για παράδειγμα, ταμπλό, κτλ.) προκειμένου να εφαρμοστούν επιπρόσθετες ευριστικές συναρτήσεις ανάλογα με τη δομή στην οποία βασίζεται το εκάστοτε παιχνίδι.

Για να εξακριβωθεί η δομή ενός παιχνιδιού, μελετώνται οι σημασιολογικές ιδιότητες των κανόνων του παιχνιδιού προκειμένου να ανακαλυφθούν στατικές δομές που είναι ανεξάρτητες της εκάστοτε κατάστασης του παιχνιδιού. Παραδείγματα ενεργειών αποτελούν η κατασκευή γράφου εξάρτησης μεταξύ των ορισμάτων κανόνων προκειμένου να βρεθεί το πεδίο ορισμού κάθε ορίσματος ώστε να βρεθούν σχέσεις μεταξύ ορισμάτων. [1]

Για παράδειγμα, προκειμένου να εξακριβωθεί αν η δομή του παιχνιδιού αποτελεί δισδιάστατο ταμπλό, τότε μπορεί ακολουθείται η εξής διαδικασία, που περιγράφεται συνοπτικά:

- Στην αρχή θεωρείται ότι κάθε τριαδική συνάρτηση στην περιγραφή των κανόνων του παιχνιδιού αποτελεί ταμπλό. Ένα από τα ορίσματα κάθε τέτοιας συνάρτηση αποτελεί την κατάσταση του σημείου του ταμπλό, το οποίο δε μπορεί να έχει παραπάνω από μία τιμές ταυτόχρονα. Συνολικά, τα δύο ορίσματα αποτελούν τα ορίσματα εισόδου και το τρίτο όρισμα αποτελεί το όρισμα εξόδου.
- Ένα υποψήφιο ταμπλό απορρίπτεται αν ποτέ έχει στο ίδιο σημείο παραπάνω από ένα αντικείμενα (για παράδειγμα, δύο πιόνια στο ίδιο τετράγωνο).
- Αν τα ορίσματα εισόδου κάθε τριαδικής συνάρτησης είναι διατεταγμένα με βάση κάποια σχέση διαδόχων (για παράδειγμα, ότι μετά το τετράγωνο 1 βρίσκεται το τετράγωνο 2), τότε το ταμπλό είναι και αυτό διατεταγμένο.
- Αφότου διαπιστωθεί ότι το παιχνίδι βασίζεται σε ταμπλό, τότε μπορούν να βρεθούν και οι δομές των αντικειμένων που βρίσκονται πάνω στο ταμπλό, όπως τα «κομμάτια» και οι «δείκτες». Στην αρχή θεωρούνται όλα τα αντικείμενα δείκτες, όμως αν ένας δείκτης μπορεί να βρίσκεται μόνο σε ένα σημείο του ταμπλό κάθε φορά, τότε ονομάζεται κομμάτι (όπως η τα κομμάτια στο σκάκι). Για παράδειγμα, τα πούλια στο “**Othello**” είναι δείκτες γιατί το μαύρο πούλι μπορεί να βρίσκεται σε πολλά τετράγωνα ταυτόχρονα. [6]

2.6 Η Γλώσσα περιγραφής παιχνιδιών GDL

Στη συνέχεια ακολουθεί εκτενέστερη περιγραφή της γλώσσας GDL, ύστερα από την εισαγωγή στο κεφάλαιο 2.1.

2.6.1 Περιγραφή

Για να περιγραφεί ένας γενικός παίκτης δημιουργήθηκε η ανάγκη ύπαρξης μίας εξειδικευμένης γενικής γλώσσας η οποία να μπορεί να τον περιγράφει. Έτσι δημιουργήθηκε η γλώσσα GDL (Game Description Language) με σκοπό να μπορούν να περιγραφούν γενικά παιχνίδια από τον γενικό παίκτη. Η GDL [5] είναι επέκταση της γλώσσας Datalog η οποία όμως περιλαμβάνει συναρτήσεις, ισοδυναμίες, μερικούς συντακτικούς περιορισμούς με σκοπό να διατηρηθεί η παρεμφατικότητα και κάποιες προσδιορισμένες λέξεις κλειδιά.

Η έμφαση στην περιγραφή της GDL είναι υψηλού επιπέδου, εμπεριέχονται οι δηλωτικοί κανόνες του παιχνιδιού που είναι εύκολο να καταλάβει κανείς και να διατηρήσει. Ταυτόχρονα, η GDL εμπεριέχει ακριβή σημασιολογία και είναι μηχανικά πλήρως επεξεργάσιμη. Επιπλέον, δεν απαιτείται γνωστικό υπόβαθρο - ένα σύνολο κανόνων είναι όλα όσα πρέπει ένας παίκτης να γνωρίζει για να είναι σε θέση να συμμετάσχει σε ένα άγνωστο παιχνίδι μέχρι εκείνη τη στιγμή. Παρακάτω παρατίθεται σαν παράδειγμα ένα κομμάτι κώδικα από το παιχνίδι της τρίλιζας γραμμένο σε GDL με σκοπό να υπάρξει μια αρχική εξοικείωση με τη γλώσσα.

```
1  role(xplayer). role(oplayer).
2
3  init(cell(1,1,b)). init(cell(1,2,b)).
4  init(cell(1,3,b)). ...
5  init(cell(1,3,b)). init(control(xplayer)).
6
7  legal(P, mark(X, Y)) :-
8    true(control(P)), true(cell(X, Y, b)).
9  legal(P,noop) :-
10   role(P), not true(control(P)).
11
12 next(cell(X,Y,x)) :- does(xplayer,mark(X,Y)).
13 next(cell(X,Y,o)) :- does(oplayer,mark(X,Y)).
14 next(cell(X,Y,C)) :-
15   true(cell(X,Y,C)), distinct(C, b).
16 next(cell(X,Y,b)) :- true(cell(X,Y,b)),
17   does(P, mark(M, N)),
18   (distinct(X, M) ; distinct(Y, N)).
19
20 goal(xplayer, 100) :- line(x).
21 ...
22 terminal :- line(x) ; line(o) ; not open.
23
24 line(P) :- true(cell(X, 1, P)),
25   true(cell(X, 2, P)), true(cell(X, 3, P)).
26 ...
27 open :- true(cell(X, Y, b)).
```

Εικόνα 22: Περιγραφή τρίλιζας σε GDL.

- Η πρώτη σειρά τού κώδικα δηλώνει τους ρόλους του παιχνιδιού. Δηλαδή τους παίχτες του παιχνιδιού. Η ανάθεση ρόλων γίνεται με το κατηγόρημα $\boldsymbol{role}$.
- Στις επόμενες σειρές έχουμε τη δήλωση των αρχικών καταστάσεων του παιχνιδιού με τη βοήθεια του μοναδιαίου κατηγορήματος $\boldsymbol{init}$.
- Στις σειρές 7-10 δηλώνονται οι **νόμιμες κινήσεις** που μπορεί να κάνει κάθε παίκτης σε κάθε γύρο, με τη χρήση του κατηγορήματος $\boldsymbol{legal}$**.** Για παράδειγμα το $\boldsymbol{mark(X,Y)}$ είναι μία νόμιμη κίνηση για το ρόλο P αν η συνθήκη $control(P)$ είναι αληθής στη τρέχουσα κατάσταση (π.χ στη σειρά του ρόλου P) και το κελί X,Y είναι κενό $(cell(X,Y,b))$.
- Οι κανόνες για το κατηγόρημα $\boldsymbol{next}$ ορίζουν τις συνθήκες που πρέπει να τηρεί η επόμενη κατάσταση, για παράδειγμα το $cell(M,N,x)$ είναι αληθές εφόσον ο παίχτης χ έβαλε το σύμβολο του στο κελί (M,N), ενώ το κελί $cell(M,N,b)$ είναι

αληθές - δηλαδή δεν αλλάζει κατάσταση - και παραμένει κενό εφόσον οο παίχτης έβαλε το σύμβολο του σε κάποιο άλλο κελί. Το ειδικό κατηγόρημα $\boldsymbol{distinct}(\boldsymbol{X},\boldsymbol{Y})$ δηλώνει ότι οι όροι X και Y είναι συντακτικά διαφορετικοί.

- Στις σειρές από 20 – 22 ορίζονται οι στόχοι των παιχτών και οι προϋποθέσεις για να προσέλθει το παιχνίδι σε κάποια τελική κατάσταση με τη βοήθεια των κατηγορημάτων $\boldsymbol{goal}$ και $\boldsymbol{terminal}$ αντίστοιχα. Οι κανόνες οι οποίοι περιέχουν τα βοηθητικά κατηγορήματα $\boldsymbol{line}$ και $\boldsymbol{open}$ κωδικοποιούν τη δημιουργία τρίλιζας και την ύπαρξη κενών κελιού αντίστοιχα.
- Τέλος έχουμε τα βοηθητικά κατηγορήματα $\boldsymbol{line}$ και $\boldsymbol{open}$ τα οποία εμπεριέχονται σε κανόνες και κωδικοποιούν τη δημιουργία τρίλιζας και την ύπαρξη κενού κελιού αντίστοιχα.

Μία σημείωση των παραπάνω είναι ότι αναφερόμαστε στα ορίσματα των κατηγορημάτων $\boldsymbol{init}$, $\boldsymbol{true}$ και $\boldsymbol{next}$ σαν «στιγμιότυπα» . Στο παράδειγμα μας έχουμε σαν ορίσματα το $control\,(X)$ όπου $X \in \{xplayer, oplayer\}$ και $cell(X,Y,Z)$ όπου $X,Y \in \{1,2,3\}$ και $Z \in \{b,x,o\}$.

2.6.2 Σημασιολογία

Ας υποθέσουμε ότι έχουμε ένα σύνολο από συγκεκριμενοποιημένους όρους και 2^{Σ} το σύνολο των υποσυνόλων του Σ. Το παιχνίδι πάνω από το Σ είναι ένα σύστημα μετάβασης κατάστασης $\Gamma = (R, s0, T, l, u, g)$ πάνω από τα σύνολα καταστάσεων $S \subseteq 2^{\Sigma}$ και ενεργειών $A \subseteq \Sigma$ με

- $R \subseteq \Sigma$ ένα πεπερασμένο σύνολο από ρόλους.
- $s0 \in S$, η αρχική κατάσταση του παιχνιδιού.
- $T \subseteq S$, το σύνολο των τερματικών καταστάσεων.
- $l : R \times A \times S$, οι έγκυρες σχέσεις.
- $u : (R \mapsto A) \times S \to S$, η μετάβαση ή η συνάρτηση ανανεωμένης θέσης.
- $g : R \times S \mapsto \mathbb{N}$, η αμοιβή ή η συνάρτηση στόχου.

Αυτή η επίσημη σημασιολογία βασίζεται στο σύνολο των συγκεκριμενοποιημένων όρων Σ. Αυτό το σύνολο είναι το σύνολο όλων αυτών των όρων πάνω από την περιγραφής της συγκεκριμένης περιγραφής παιχνιδιού. Άρα , στιγμιότυπα, ενέργειες και ρόλοι του παιχνιδιού είναι συγκεκριμενοποιημένοι όροι του Σ. Οι καταστάσεις είναι πεπερασμένα σύνολα στιγμιότυπων, δηλαδή πεπερασμένα υποσύνολα του Σ. Η σύνδεση μεταξύ μίας περιγραφής παιχνιδιού D και ενός παιχνιδιού Γ που την περιγράφει, καθιερώνεται χρησιμοποιώντας ένα standard μοντέλο λογικού προγραμματισμού D. Για παράδειγμα η συνάρτηση ανανεωμένης θέσης u ορίζεται σαν ,

$$u(A,s) = \{f \in \Sigma : D \cup s^{true} \cup A^{does} \vDash next(f)\}$$

όπου το s^{true} και το A^{does} είναι κατάλληλες κωδικοποιήσεις μίας κατάστασης s και μίας ενέργειας σύζευξης A για όλους τους παίχτες όπως ένα λογικό πρόγραμμα. Οπότε, η διάδοχη κατάσταση $u(A,s)$ είναι το σύνολο όλων των συγκεκριμενοποιημένων όρων (στιγμιότυπων) f όπως το $next(f)$ το οποίο εμπεριέχεται στην περιγραφή του παιχνιδιού D μαζί με τη κατάσταση s και την ενέργεια σύζευξης A.

2.7 Η Γλώσσα περιγραφής παιχνιδιών GDL – II

Στο κεφάλαιο αυτό γίνεται περιγραφή της γλώσσας GDL – II, που αποτελεί επέκταση της γλώσσας GDL.

2.7.1 Περιγραφή

Η GDL – II [10] είναι μία πρόσφατη επέκταση της GDL η οποία έγινε με σκοπό να μπορέσει να καλύψει παιχνίδια αυθαίρετου τύπου με ελλιπή/ατελή πληροφορία (incomplete/imperfect).

Ο στόχος του γενικού παίκτη είναι ουσιαστικά η δημιουργία ενός έξυπνου συστήματος το οποίο να κατανοεί τους κανόνες από νέα παιχνίδια αυθαίρετου τύπου και χωρίς ανθρώπινη παρέμβαση να μαθαίνει να παίζει αυτά τα παιχνίδια καλά. Η GDL έχει οριστεί σαν επίσημη, μηχανικά επεξεργάσιμη γλώσσα για την περιγραφή παιχνιδιών αυθαίρετου τύπου. Αρχικά ήταν περιορισμένη στην κατανόηση και διαχείριση παιχνιδιών με τέλεια πληροφορία, αλλά πρόσφατα επεκτάθηκε στη διαχείριση παιχνιδιών αποτελούμενα από άνω των δύο παιχτών με ατελή / ελλιπή πληροφορία.

Παρόλο που είναι βασισμένες στη λογική η GDL και η διάδοχος της GDL II (από: GDL with incomplete information), είναι μινιμαλιστικές με την έννοια ότι πρέπει να παρέχονται μόνο οι κανόνες. Για τον τρόπο όμως, τον οποίο πρέπει να χειρίζονται αυτοί οι κανόνες, όπως για παράδειγμα στην αναφορά των νόμιμων κινήσεων η για την ανανέωση μία συγκεκριμένης θέσης στο παιχνίδι, αφήνεται στην κρίση των παιχτών.

Η γλώσσα περιγραφής γενικών παιχνιδιών GDL – II βασίζεται στο καθορισμένο συντακτικό και σημασιολογία του λογικού προγραμματισμού. Ακολουθείται η σύμβαση της Prolog όσον αφορά τη δήλωση μεταβλητών η οποία πραγματοποιείται από κεφαλαία γράμματα, ενώ οι περιορισμοί και τα σύμβολα συναρτήσεων ορίζονται από μικρά γράμματα. Η GDL II χαρακτηρίζεται από τα παρακάτω ειδικά keywords.

`role(R)`	R is a player
`random`	the random player
`init(F)`	F holds in the initial position
`true(F)`	F holds in the current position
`legal(R,M)`	R can do move M in the current position
`does(R,M)`	player R does move M
`next(F)`	F holds in the next position
`sees(R,P)`	R perceives P in the next position
`terminal`	the current position is terminal
`goal(R,N)`	R gets N points in the current position

Εικόνα 23: Λέξεις – κλειδιά της GDL – II

Επιπλέον έχουμε το βοηθητικό κατηγόρημα $distinct\ (X, Y)$ το οποίο επισημαίνει την ατομικότητα δύο ορισμάτων. Επιπλέον μη δεσμευμένες μεταβλητές σε διατάξεις ή άλλους τύπους υπονοείται ότι είναι καθολικά ποσοτικές.

Παρακάτω παρατίθεται ένα παράδειγμα το οποίο περιγράφει το παιχνίδι της τρίλιζας στη γλώσσα GDL II αλλά με μια ενδιαφέρουσα παραλλαγή στην οποία οι παίχτες δε

γνωρίζουν τις κινήσεις των αντιπάλων τους (όπως στην Kriegspiel παραλλαγή σκακιού (Pritchard 1994)).

```
1  role(xplayer).
2  role(oplayer).
3  init(control(xplayer)).
4  init(cell(1,1,b)).
5  ...
6  init(cell(3,3,b)).
7
8  legal(R,mark(M,N))  :- true(control(R)),
9                         true(cell(M,N,Z)),
10                        not true(tried(M,N)).
11 legal(xplayer,noop) :- true(control(oplayer)).
12 legal(oplayer,noop) :- true(control(xplayer)).
13
14 validmove :- does(R,mark(M,N)), true(cell(M,N,b)).
15
16 next(F)           :- not validmove, true(F).
17 next(tried(M,N))   :- not validmove, does(P,mark(M,N)).
18 next(cell(M,N,x)):- validmove, does(xplayer,mark(M,N)).
19 next(cell(M,N,o)):- validmove, does(oplayer,mark(M,N)).
20 next(cell(M,N,Z)):- validmove, true(cell(M,N,Z)),
21                       does(P,mark(I,J)), distinct(M,I).
22 next(cell(M,N,Z)):- validmove, true(cell(M,N,Z)),
23                       does(P,mark(I,J)), distinct(N,J).
24 next(control(oplayer)):- validmove,
25                          true(control(xplayer)).
26 next(control(xplayer)):- validmove,
27                          true(control(oplayer)).
28
29 sees(R,yourmove)       :- not validmove,
30                          true(control(R)).
31 sees(xplayer,yourmove) :- validmove,
32                          true(control(oplayer)).
33 sees(oplayer,yourmove) :- validmove,
34                          true(control(xplayer)).
```

Εικόνα 24: Περιγραφή παραλλαγής της τρίλιζας χρησιμοποιώντας GDL – II

Παραπάνω έχουμε τα εξής:

- Στις σειρές από 1 – 6 έχουμε τα ονόματα των παιχτών και τις αρχικές θέσεις του παιχνιδιού με τα κατηγορήματα $\boldsymbol{role}()$ και $\boldsymbol{init}()$.
- Στις σειρές από 8 – 12 ορίζονται οι κανόνες με το κατηγόρημα $\boldsymbol{legal}()$, το οποίο υποδηλώνει τις επιτρεπόμενες κινήσεις που μπορούν να γίνουν από τον εκάστοτε παίχτη. Το όρισμα $\boldsymbol{noop}$ δηλώνει ότι ο άλλος παίχτης κάνει κίνηση χωρίς επίδραση.
- Η ενημέρωση των θέσεων του παιχνιδιού υποδηλώνεται με το κατηγόρημα $\boldsymbol{next}(\boldsymbol{m},\boldsymbol{n})$ στις σειρές 16 – 27. Αν η υποβληθείσα κίνηση ακολουθούμενη από το κατηγόρημα $mark()$, είναι μη έγκυρη, τότε το κάθε χαρακτηριστικό της τρέχουσας κατάστασης του παιχνιδιού παραμένει ως έχει και το μόνο που αλλάζει είναι η τιμή του κατηγορήματος $\boldsymbol{tried}(\boldsymbol{m},\boldsymbol{n})$, που αποκτά την τιμή $\boldsymbol{true}$. Αντιθέτως αν η κίνηση είναι έγκυρη τότε το κατηγόρημα $\boldsymbol{cell}\ (\boldsymbol{m},\boldsymbol{n})$ αποκτά το στίγμα του συγκεκριμένου παίχτη ενώ όλα τα άλλα κελιά διατηρούν το περιεχόμενο τους ακέραιο. Επιπλέον το δικαίωμα της επόμενης κίνησης μεταφέρεται στον άλλο παίχτη. Ο αναγνώστης πρέπει επιπλέον να προσέξει ότι όλες οι οντότητες του κατηγορήματος $\boldsymbol{tried}(\boldsymbol{m},\boldsymbol{n})$ παύουν να ισχύουν στην περίπτωση που δεν υπάρχει όρος με κεφαλή το $\boldsymbol{next}(\boldsymbol{tried}(\boldsymbol{m},\boldsymbol{n}))$ και εφόσον το όρισμα $\boldsymbol{validmove}$ έχει την τιμή $\boldsymbol{true}$.
- Τέλος στις σειρές 29-34 με το κατηγόρημα $\boldsymbol{sees}()$ δηλώνει ότι ο παίχτης που ακολουθεί στη σειρά να παίξει, ενημερώνεται για την κατάσταση του παιχνιδιού.

2.7.2 Επίσημο συντακτικό και Σημασιολογία

Με σκοπό την αποτροπή μίας διφορούμενης καταγραφής των κανόνων ενός παιχνιδιού, η περιγραφή των κανόνων σε GDL II πρέπει να υπακούει σε βασικούς περιορισμούς γενικού συντακτικού. Συγκεκριμένα ένα έγκυρο παιχνίδι πρέπει να επιδέχεται ένα συγκεκριμένο standard μοντέλο. Ένας επιπλέον συντακτικός περιορισμός εξασφαλίζει, ότι μόνο ένας πεπερασμένος αριθμός από θετικά δείγματα είναι αληθή στο συγκεκριμένο μοντέλο. Τέλος τα ειδικά keywords πρέπει να χρησιμοποιηθούν όπως παρακάτω:

- $\boldsymbol{role}$ εμφανίζεται μόνο σαν κεφαλή στα γεγονότα.
- $\boldsymbol{init}$ εμφανίζεται μόνο σαν κεφαλή σε ορισμούς και δεν εξαρτάται από κανένα από τα παρακάτω κατηγορήματα $true$, $legal$, $does$, $next$, $sees$, $terminal$, $goal$.
- $\boldsymbol{true}$ εμφανίζεται μόνο στο σώμα των όρων και δεν εξαρτάται από κανένα από τα παρακάτω $legal$, $terminal$, $goal$.
- $\boldsymbol{next}$ και $\boldsymbol{sees}$ τα οποία εμφανίζονται μόνο στην κεφαλή των όρων.

Κάτω από αυτούς τους περιορισμούς, κάθε έγκυρη GDL II περιγραφή παιχνιδιού καθορίζει ένα σύστημα μετάβασης κατάστασης ως εξής:
Ξεκινώντας, κάθε σετ από κανόνες καθορίζει ένα έμμεσο σετ από συγκεκριμενοποιημένους όρους Σ οι οποίοι είναι εξαρτώμενοι τομέα όπως για παράδειγμα το $cell(1,1,b)$, $mark(1,3)$, $yourmove$ κ.ο.κ. Οι θέσεις στο παιχνίδι αντικατοπτρίζονται από υποσύνολα από τα αναφερόμενα σύνολο Σ αποτελούνται από ανεξάρτητα χαρακτηριστικά. Παρόλο που το Σ είναι συνήθως άπειρο, οι συντακτικοί περιορισμοί στην GDL II εξασφαλίζουν ότι το σετ ρόλων , οι προσβάσιμες καταστάσεις, και το σετ από τα επιτρεπόμενα βήματα είναι πάντα πεπερασμένα υποσύνολα του Σ. Ειδικότερα οι υπολογιζόμενοι όροι του κατηγορήματος $role(R)$ καθορίζει τους παίχτες και την αρχική κατάσταση η οποία αποτελεί τους υπολογιζόμενους όρους του κατηγορήματος $init(F)$.
Με σκοπό να καταγραφούν και καθοριστούν τα επιτρεπόμενα βήματα ενός παίχτη στην δοθείσα κατάσταση, η συγκεκριμένη κατάσταση πρέπει πρώτα να κωδικοποιηθεί χρησιμοποιώντας τη λέξη κλειδί $true$: Ας υποθέσουμε ότι έχουμε την κατάσταση $S = \{f1, \ldots . fn\}$ (π.χ ένα πεπερασμένο σύνολο από συγκεκριμενοποιημένους όρους), τότε το G επεκτείνεται κατά n όρους

$$S^{true} \stackrel{\text{def}}{=} \{true(f1) \ldots \ldots . true(fn).\}$$

Οι συγκεκριμένοι όροι του $legal(R, M)$ είναι υπολογίσιμές από την ένωση $G \cup S^{true}$ η οποία ορίζει όλα τα επιτρεπόμενα βήματα M για τον παίχτη R στη θέση S. Με τον ίδιο τρόπο, οι όροι για το $terminal$ και το $goal(R, N)$ ορίζουν τον τερματισμό και τους στόχους που σχετίζονται με την κωδικοποίηση της δοθείσας θέση.
Αναλογίζοντας την ανανεωμένη θέση και τις αντιλήψεις των παιχτών, απαιτείται η κωδικοποίηση για την τρέχουσα θέση και του βήματος που ακολουθεί. Συγκεκριμένα αν το M είναι τέτοιο ώστε οι παίχτες r1,…rk κάνουν κινήσεις m1,….mk τότε ισχύει το παρακάτω:

$$M^{does} \stackrel{\text{def}}{=} \{does(r1, \quad m1) \ldots \ldots . does(rk, \quad mk).\}$$

Οι οντότητες του $next\,(F)$ οι οποίες προέρχονται από το $G \cup M^{does} \cup S^{true}$ αποτελούν την νέα θέση, όπως για παράδειγμα οι παραγόμενες οντότητες του κατηγορήματος $sees\ (R, P)$ οι οποίες περιγράφουν τι ένας παίχτης αντιλαμβάνεται σε μία τρέχουσα θέση.

Ορισμός 1

Η σημασιολογία μίας έγκυρης περιγραφής παιχνιδιού GDL – II G είναι το σύστημα μετάβασης κατάστασης που δίνεται από :

- $R = \{ r : G \models role(r) \}$ (Ονόματα παιχτών).
- $s1 = \{ f : G \models init(f) \}$ (Αρχική κατάσταση).
- $t = \{ S : G \cup S^{true} \models terminal \}$ (Τελικές καταστάσεις).
- $l = \{ (r, m, S) : G \cup S^{true} \models legal(r, m) \}$ (Έγκυρα βήματα).
- $u(M, S) = \{ f : G \cup M^{does} \cup S^{true} \models next(f) \}$, για όλα τα συζευγμένα βήματα M και τις πεπερασμένες καταστάσεις S (ανανεωμένη θέση).
- $I = \{(r, M, S, p): G \cup M^{does} \cup S^{true} \models sees(r, p) \}$, για όλους τους ρόλους $r \in R \setminus \{ random\}$ (αντιλήψεις παιχτών).
- $g = (r, n, S) : G \cup S^{true} \models goal(r, n) \}$ (στόχοι).

2.7.3 Η GDL – II στη τεχνική Situation Calculus

Σύμφωνα με την περιγραφή του παιχνιδιού στην (Εικόνα 24: Περιγραφή παραλλαγής της τρίλιζας χρησιμοποιώντας GDL – II**)** τα κατηγορήματα στις σειρές 29 – 34 δηλώνουν τι αντιλαμβάνονται οι παίχτες στη διάρκεια του παιχνιδιού, υποδηλώνοντας ότι ο παίχτης ο οποίος έχει σειρά να παίξει, θα ενημερωθεί για το γεγονός. Αυτός ο αφαιρετικός ορισμός επαρκεί εφόσον οι παίχτες πρέπει πάντα να γνωρίζουν αν είναι η σειρά τους η όχι. Στην περίπτωση την οποία ένας παίχτης δε μπορεί να παίξει, θα πρέπει να είναι σε θέση να καταλαβαίνει ότι πρέπει να είναι η σειρά του αντιπάλου του. Ένα άλλο παράδειγμα (στρατηγικά πολύ χρήσιμο) ενός συμπεράσματος είναι η δυνατότητα να μπορεί να συμπεριληφθεί στο κελί η δυνατότητα να φέρει το σημάδι του αντιπάλου, οπότε αν προσπαθήσει κάποιος άλλος παίχτης να σημαδέψει ο ίδιος αυτό το κελί να μην τα καταφέρει υπονοώντας ότι η προσπάθεια του ήταν ανεπιτυχής.

Εξάγοντας συμπεράσματα αυτού του είδους είναι ο τομέας που πραγματεύεται θεωρίες ενεργειών , και η **Situation Calculus** θεωρείται η παλαιότερη τεχνική για επισημοποίηση και αυτοματοποιημένη συμπερασματολογία για τις ενέργειες. Για την επίτευξη εξερεύνησης αυτής της σειράς έρευνας στο πλαίσιο του τομέα General Game Playing χρειάζεται να ενσωματωθεί πλέον η γλώσσα περιγραφής GDL - II στην μέθοδο Situation Calculus. Η αντιστοίχηση η οποία θα περιγραφεί σε αυτή την ενότητα βασίζεται στην εξελιγμένη αυτή μέθοδο με ειδική ευχέρεια στο να αντιπροσωπεύει τη γνώση από τους agent. Θα πρέπει ελαφρώς να τροποποιηθεί για να επεκταθεί ώστε να εξυπηρετήσει τους σκοπούς αυτής της ενότητας. Γενικά η μέθοδος αυτή είναι με λογικούς περιορισμούς και μερικώς προκαθορισμένα λεκτικά στοιχεία.

- Συνεχές s_0, υποδηλώνοντας την αρχική κατάσταση και τον κατασκευαστή $DO(\alpha, \sigma)$, υποδηλώνοντας ότι η κατάσταση καταλήγει πραγματοποιώντας την ενέργεια α στην κατάσταση σ.
- Ο περιορισμός $HOLDS(\varphi, \sigma)$, υποδηλώνει ότι το στιγμιότυπο φ είναι αληθές στην κατάσταση σ.
- Ο περιορισμός $POSS(\alpha, \sigma)$, υποδηλώνει ότι η ενέργεια α είναι πιθανή στην κατάσταση σ.

2.7.3.1 Σύνθετες Ενέργειες

Στα παιχνίδια με δύο η παραπάνω παίχτες, μία νέα θέση οδηγεί στο όλοι οι παίχτες κινούνται ταυτόχρονα. Μία επαρκή τυποποίηση αυτής της μεθόδου χρειάζεται για την αναγνώριση ενός σεναρίου ενέργειας α με διάνυσμα <m1,….mκ>, περιέχοντας μία κίνηση για τον κάθε παίχτη. Σε μία δοθείσα περιγραφή σε GDL-II, ο χώρος των κινήσεων καθορίζεται εμμέσως από τις παραμέτρους της λέξης κλειδιού $legal$ και των δευτερευόντων ορισμάτων της λέξης κλειδιού $does$. Για παράδειγμα το κατηγόρημα $mark\,(M,N)$ και της λέξης κλειδί $noop$ στη Krieg – Tictactoe. Υποθέτοντας μια αυθαίρετη αλλά καθορισμένη σειρά των παιχτών, όπως (x παίχτης , o παίχτης). Καθορίζουμε ένα απλό αξίωμα το οποίο αναγνωρίζει τη κίνηση του τρέχοντα παίχτη σε ένα διάνυσμα ενεργειών:

$$ACT\,(r, < m1, \dots, mr, \dots, mk >) = mr \quad (1)$$

Για παράδειγμα $ACT\,(xplayer, < mark(1,3), noop >) = mark(1,3)$.

2.7.3.2 Συμπληρωματικά κατηγορήματα ενεργειών

Δοθείσας μίας περιγραφής παιχνιδιού σε GDL – II, αναγνωρίζονται σαν αρχικά στιγμιότυπα εκείνοι οι όροι στα πλαίσια καθεμίας από τις λέξεις κλειδιά $init\,(F)$, $true\,(F)$, ή το $next\,(F)$. Στην (Εικόνα 24: Περιγραφή παραλλαγής της τρίλιζας χρησιμοποιώντας GDL – II) αυτοί είναι οι $control\,(R)$, $cell\,(M,N,Z)$, και $tried\,(M,N)$. Σαν συμπληρωματικά στιγμιότυπα λαμβάνουμε υπόψη όλα εκείνα τα κατηγορήματα τα οποία είναι ειδικά τομέα και τα οποία εξαρτώνται από το $true$ αλλά όχι από το $does$. Τα συμπληρωματικά στιγμιότυπα δεν απαιτούν τα δικά τους διαδοχικά αξιώματα καταστάσεων, επειδή οι τιμές αληθείας τους είναι πλήρως προσδιορισμένες από τους κανόνες του παιχνιδιού, εφόσον όλες οι τιμές των αρχικών στιγμιότυπων είναι καθορισμένες σε μία διάδοχη κατάσταση. Οι λέξεις κλειδιά $terminal$ και $goal\,(R,N)$ αντιμετωπίζονται σαν συμπληρωματικά στιγμιότυπα, επίσης.

Επιπλέον όσον αφορά τα συμπληρωματικά στιγμιότυπα , η αντιστοίχηση της GDL – II στη μέθοδο **Situation Calculus** απαιτεί μία εισαγωγή ενός νέου σεναρίου για τα συμπληρωματικά κατηγορήματα ενεργειών. Αυτά είναι κατηγορήματα τύπου ειδικά τομέα και τα οποία εξαρτώνται από τις τιμές $true$ και $does$. Παρόμοια στα συμπληρωματικά στιγμιότυπα, η τιμή αληθείας ενός συμπληρωματικού αξιώματος ενέργειας είναι πλήρως καθορισμένη στους κανόνες του παιχνιδιού εφόσον είναι καθορισμένες οι τιμές των αρχικών στιγμιότυπων σε μία κατάσταση και στην ενέργεια την οποία εκτελείται στη συγκεκριμένη κατάσταση. Ένα παράδειγμα αυτού είναι το κατηγόρημα $validmove$ στο Krieg-Tictactoe.

2.7.3.3 Η αντιστοίχηση

Όσον αφορά την κάθε περιγραφή G σε GDL – II μπορεί να αντιστοιχηθεί με ένα πρισματικό τρόπο στην θεωρία **Situation Calculus**. Αρχικά, μερικά άτομα τα οποία εμφανίζονται στη G είναι αποτυπωμένα όπως παρακάτω:

1. Όλα τα συμπληρωματικά στιγμιότυπα $f(\vec{X})$ αντικαθίστανται από το $f(\vec{X},\, S)$ και όλα τα συμπληρωματικά κατηγορήματα ενεργειών $p(\vec{X})$ από το $p(\overrightarrow{X},\, A,\, S)$, υποδεικνύοντας την εξάρτηση στην κατάσταση S και στην ενέργεια A, αντίστοιχα.
2. Κάθε κατηγόρημα $init(\varphi)$ αντικαθίσταται από το $HOLDS\,(\varphi, s0)$.
3. Κάθε κατηγόρημα $true\,(\varphi)$ αντικαθίσταται από το $HOLDS\,(\varphi, S)$.

4. Κάθε κατηγόρημα $next\ (\varphi)$ αντικαθίσταται από το $HOLDS\ (\varphi, Do(A,S))$.
5. Κάθε κατηγόρημα $does\ (\rho,\mu)$ αντικαθίσταται από το $ACT\ (\rho,A) = \mu$.

   Σαν παράδειγμα, ο όρος στη σειρά 14 στην (Εικόνα 24: Περιγραφή παραλλαγής της τρίλιζας χρησιμοποιώντας GDL – II) μετατρέπεται:

$$validmove\ (A,S) \subset ACT(R,A) = mark(M,N) \bigwedge HOLDS(cell(M,N,b),S)$$

Οι GDL-II περιγραφές παιχνιδιών βασίζονται στον κανόνα «Άρνηση ως Αποτυχία», όπου σύμφωνα με αυτόν, κάθε πρόταση η οποία δεν μπορεί να παραχθεί από τους κανόνες του παιχνιδιού θεωρείται ως λανθασμένη. Για να αντικατοπτριστεί αυτό στη **Situation Calculus** θεωρία, χρησιμοποιούμε το συμπλήρωμα όλων των όρων σύμφωνα με τον παρακάτω τρόπο: Για κάθε κατηγόρημα $p(\vec{X})$, αντικαθιστούμε τους όρους με το κατηγόρημα στην κεφαλή από

$$p(\vec{X}) \equiv \bigvee_{p(\vec{t}):\ -body \in G} \vec{X} = \vec{t}\ \wedge body$$

Αρχική κατάσταση: Η μετατροπή που ορίζεται παραπάνω αποφέρει το ακόλουθο αξιωματισμό της αρχικής κατάστασης

$$HOLDS(F,s0) \equiv \bigvee_{init(\varphi):\ -body \in G} F = \varphi\ \wedge body$$

Προϋποθέσεις: Βασιζόμενοι στην ολοκλήρωση του $legal$, ορίζουμε το προϋποθετικό αξίωμα για τις σύνθετες ενέργειες όπως παρακάτω:

$$POSS(A,S) \equiv \forall R.LEGAL(R,ACT(R,A),S)$$

Επιδράσεις: Σαν αποτέλεσμα της παραπάνω μετατροπής, αποκτούμε ένα γενικό αξίωμα διάδοχης κατάστασης όπως ακολουθεί:

$$HOLDS(F,Do(A,S)) \equiv \bigvee_{next(\varphi):\ -body \in G} F = \varphi\ \wedge body$$

Γνώση: Το 2003 εισήχθη ένα νέο ειδικό στιγμιότυπο το $K(S',S)$ – το οποίο διαβάζεται όπως παρακάτω: Η κατάσταση S' είναι προσβάσιμη από την κατάσταση S – για τον αξιωματισμό καταστάσεων γνώσης, από κάθε παίκτη στην μέθοδο Situation Calculus. Χρησιμοποιείται μια ευθύς γενίκευση για την περίπτωση των πολλαπλών παικτών, όπου το K(R, S', S) χρησιμοποιείται για να εκφράσει ότι ο παίχτης R θεωρεί την S' σαν πιθανή κατάσταση της S. Αυτό επιτρέπει το φορμαλισμό της υποκειμενικής γνώσης όπως:

$$KNOWS\ (\rho,\Phi,\sigma) \stackrel{\text{def}}{=} \forall S'.K(\rho,S',\sigma)\ \supset \Phi[S']$$

Εδώ το Φ είναι μία μεταποιημένη φόρμουλα όπου οι παράμετροι καταστάσεως σε όλα τα στιγμιότυπα καταστέλλονται και το Φ[S'] σημαίνει ότι όλοι οι παράμετροι καταστάσεως πραγματοποιούνται στο S'. Για παράδειγμα:

$$KNOWS\ (xplayer, \forall X,Y.cell(X,Y,b),S')$$
$$\forall S'.K(xplayer,\ s0) \supset\ \forall X,Y.HOLDS(cell(X,Y,b),S').$$

Ο συγκεκριμένος ορισμός της μακροεντολής αυτής μπορεί εύκολα να επεκταθεί σε εμφωλευμένες εκφράσεις, όπως:

$$KNOWS\Big(xplayer, KNOWS\big(oplayer, control(oplayer)\big),\qquad DO(\langle mark(1,1), noop\rangle, s0)\Big).$$

Στη GDL-II όλοι οι παίχτες έχουν πλήρη γνώση της αρχικής κατάστασης. Από την άποψη της μεθόδου Situation Calculus έχουμε,

$$K(R, S, s0) \equiv S = s0$$

Οι συνέπειες των ενεργειών και αντιλήψεων πάνω στις καταστάσεις γνώσης των παιχτών ορίζονται από το αξίωμα διάδοχης κατάστασης για το ειδικό στιγμιότυπο K, για το οποίο υιοθετούμε τον ορισμό όπως παρακάτω:

$$K(R, S'', Do(A, S)) \equiv \exists A', S'.S''$$
$$= Do(A', S') \wedge K(R, S', S) \wedge POSS(A', S') \wedge ACT(R, A') \wedge \forall P.SEES(R, P, A, S)$$
$$\equiv SEES(R, P, A', S')$$

Για να το εκφράσουμε με λόγια, ένας παίχτης θεωρεί την πιθανή κατάσταση S'' μετά από τη σύνθετη ενέργεια A στην κατάσταση S αν και μόνο αν η S'' λαμβάνεται να κάνει κάποια ενέργεια A' σε μία κατάσταση S' η οποία ήταν αντιληπτή στην S. Η A' προϋποθέτει να ήταν εκτελέσιμη στην S'. Ο παίχτης έκανε την ίδια κίνηση A' όπως και στην A, και ο παίχτης αντιλαμβάνεται ότι τα αποτελέσματα στην A', S' είναι όμοια με τις A,S , οπότε η διαφορά δεν μπορεί να γίνει διακριτή ανάμεσα στις δύο.

3. ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΓΕΝΙΚΟΥ ΠΑΙΚΤΗ SWEETGGP

Σε αυτό το κεφάλαιο παρουσιάζονται σημαντικά κομμάτια του γενικού παίκτη SweetGGP, που αναπτύχθηκε στα πλαίσια της διπλωματικής εργασίας, και λεπτομέρειες για την υλοποίηση του.

3.1 Διάβασμα παιχνιδιού

Τα προγράμματα, που περιγράφουν παιχνίδια, είναι διατυπωμένα σε GDL, η οποία αποτελεί παραλλαγή της Datalog. Αυτό σημαίνει ότι είναι γραμμένα σε μορφή παρόμοια με την παρακάτω:

$$\boldsymbol{role}(\boldsymbol{white}).$$

$$\boldsymbol{role}(\boldsymbol{black}).$$

$$\boldsymbol{init}(\boldsymbol{control}(\boldsymbol{white})).$$

...

$$\boldsymbol{legal}(\boldsymbol{white}, \boldsymbol{noop}) :- \boldsymbol{true}(\boldsymbol{control}(\boldsymbol{black})).$$

...

$$\boldsymbol{terminal} :- \boldsymbol{not}(\boldsymbol{open}).$$

...

Στο SweetGGP τα παιχνίδια που χρησιμοποιούνται είναι διατυπωμένα σε μορφή Αανταλλαγής Γνώσης (Knowledge Interchange Format, KIF). Η μορφή ανταλλαγής γνώσης αποτελεί γλώσσα ανταλλαγής γνώσης μεταξύ διαφορετικών προγραμμάτων. Στη συνέχεια γίνεται περιγραφή της μορφής ανταλλαγής γνώσης.

3.1.1 Μορφή Ανταλλαγής Γνώσης (Knowledge Interchange Format, KIF)

Προκειμένου να είναι δυνατή η περιγραφή παιχνιδιών, ακολουθώντας τα πρότυπα της GDL, πρέπει να είναι δυνατή η περιγραφή των παρακάτω στοιχείων στη μορφή ανταλλαγής γνώσης:

- Σταθερές.
- Μεταβλητές.
- Συναρτησιακά σύμβολα.
- Σχέσεις
- Συμβολισμός σύζευξης (AND).
- Συμβολισμός διάζευξης (OR).
- Συμβολισμός άρνησης (NOT).

Στη μορφή ανταλλαγής γνώσης είναι δυνατή η περιγραφή των παραπάνω στοιχείων, αφού αποτελεί μια εναλλακτική μορφή της Datalog. Οι τρόποι περιγραφής των παραπάνω στοιχείων καταγράφεται στη συνέχεια.

3.1.1.1 Σταθερές

Οι σταθερές αποτελούν ορίσματα σχέσεων και συναρτησιακών συμβόλων και η τιμή τους παραμένει σταθερή καθ' όλη την εκτέλεση των επερωτήσεων. Στη datalog δηλώνονται ως λέξεις που το πρώτο του γράμμα είναι αναγκαστικά πεζό, για παράδειγμα, $noop$, $white$, $black$, κτλ. Στη μορφή ανταλλαγής πληροφορίας οι σταθερές δηλώνονται ως λέξεις που μπορούν να ξεκινούν είτε με πεζό γράμμα είτε με κεφαλαίο, δηλαδή δεν υπάρχει ο περιορισμός που υπάρχει στη datalog. Για παράδειγμα, μπορούν να υπάρχουν οι εξής σταθερές στο παιχνίδι: $White$, $Black$, $Noop$, κτλ.

3.1.1.2 Μεταβλητές

Οι μεταβλητές αποτελούν και αυτές ορίσματα σχέσεων και συναρτησιακών συμβόλων, όπως και οι σταθερές, αλλά οι τιμές τους δεν είναι σταθερές και εξακριβώνονται μόνο ύστερα από απάντηση επερώτησης. Στη datalog δηλώνονται ως λέξεις που το πρώτο γράμμα είναι αναγκαστικά κεφαλαίο, για παράδειγμα, X, Y, $Noop$, κτλ. Στη μορφή ανταλλαγή γνώσης, αντίθετα, οι μεταβλητές δηλώνονται ως λέξεις οι οποίες ξεκινούν με ένα ερωτηματικό "?". Τα υπόλοιπα γράμματα μπορούν να είναι είτε πεζά, είτε κεφαλαία. Μερικά παραδείγματα μεταβλητών στη μορφή ανταλλαγής γνώσης είναι τα εξής: $?x$, $?y$, $?player$, $?X$, $?Y$, $?Role$, κτλ.

3.1.1.3 Συναρτησιακά σύμβολα

Τα συναρτησιακά σύμβολα αποτελούν εκφράσεις οι οποίες έχουν ένα ή περισσότερα ορίσματα (αν δεν έχουν κανένα όρισμα, τότε αποτελούν σταθερές). Τα ορίσματα αυτά μπορούν να είναι μόνο σταθερές ή μεταβλητές. Στην datalog, τα συναρτησιακά σύμβολα περιγράφονται ως εξής: $όνομα\ (ορισμα_1\ ορισμα_2\ \ldots.ορισμα_v)$

Μερικά παραδείγματα είναι τα εξής:

$\boldsymbol{role(white)}$

$\boldsymbol{control(white)}$

$\boldsymbol{mark(X,Y)}$

Στη μορφή ανταλλαγής γνώσης περιγράφονται ως εξής: $(όνομα\ ορισμα_1\ ορισμα_2\ \ldots.ορισμα_v)$

Μερικά παραδείγματα είναι τα εξής:

$\boldsymbol{(role\ white)}$

$\boldsymbol{(control\ white)}$

$\boldsymbol{(mark\ ?x\ ?y)}$

3.1.1.4 Σχέσεις

Οι σχέσεις αποτελούν εκφράσεις οι οποίες έχουν δύο χαρακτηριστικά:

Περιέχουν ορίσματα τα οποία μπορούν να είναι είτε σταθερές, είτε μεταβλητές, είτε συναρτησιακά σύμβολα. Μια σχέση μπορεί να μην έχει και κανένα όρισμα.

Περιέχουν μια λίστα από άλλες σχέσεις, οι οποίες πρέπει να αληθεύουν προκειμένου να αληθεύει και η ίδια η σχέση.

Οι σχέσεις στη datalog περιγράφονται ως εξής: $όνομα\ (ορισμα_1\ ορισμα_2\ ….ορισμα_v) : -\ σχεση_1, σχεση_2, \ldots, σχεση_\mu$.

Μερικά παραδείγματα είναι τα εξής:

$$\boldsymbol{legal(P, mark(X,Y)) : -\ true(control(P)), true(cell(X,Y))}.$$

Η λίστα των περαιτέρω σχέσεων βέβαια μπορεί να είναι κενή, οπότε τότε η σχέση συντάσσεται ως εξής:

$$\boldsymbol{greater(2,1)}.$$

Οι σχέσεις αυτές είναι πάντοτε αληθείς.

Αντίθετα, στη μορφή ανταλλαγής γνώσης, οι σχέσεις περιγράφονται ως εξής:

$$(<=\ \ (όνομα\ ορισμα_1\ ορισμα_2\ ….ορισμα_v)\ \ (σχεση_1)\ \ (σχεση_2)\ \ \ldots\ \ (σχεση_\mu)).$$

Μερικά παραδείγματα είναι τα εξής:

$$(<=\ \ (\boldsymbol{legal\ ?p\ (mark\ ?x\ ?y))}\ \ (\boldsymbol{true\ (control\ ?p))}\ \ (\boldsymbol{true\ (cell\ ?x\ ?y)))}$$

Αν μια σχέση δεν έχει ορίσματα, τότε δε μπαίνουν παρενθέσεις. Για παράδειγμα:

$$(<=\ \ \boldsymbol{terminal}\ \ (\boldsymbol{true\ (step\ 50)}))$$

Αν η λίστα των περαιτέρω σχέσεων είναι κενή, τότε δε χρησιμοποιείται το σύμβολο <= και η σχέση περιγράφεται ως εξής:

$$(\boldsymbol{greater\ 2\ 1})$$

3.1.1.5 Συμβολισμός Σύζευξης (AND)

Όπως σημειώθηκε και προηγουμένως, για να είναι αληθής μια σχέση θα πρέπει να είναι αληθείς οι σχέσεις στη λίστα της (εκτός κι αν η λίστα είναι κενή). Στη datalog, οι σχέσεις που χωρίζονται με κόμμα “,”, πρέπει να είναι αληθείς και οι 2. Το ίδιο επεκτείνεται και για όλες τις υπόλοιπες σχέσεις της λίστας. Για παράδειγμα, στην παρακάτω σχέση:

$$\boldsymbol{brothers(X,Y) : -\ distinct(X,Y), parent(X,Z), parent(Y,Z)}.$$

Για να αληθές το $\boldsymbol{brothers(X,Y)}$, θα πρέπει να ισχύει και το $\boldsymbol{distinct(X,Y)}$ και το $\boldsymbol{parent(X,Z)}$ και το $\boldsymbol{parent(Y,Z)}$.

Στη μορφή ανταλλαγής γνώσης, η σύζευξη αυτή περιγράφεται απλά αφήνοντας ένα κενό μεταξύ των σχέσεων. Για παράδειγμα:

$$(<=\ \ (\boldsymbol{brothers\ ?x\ ?y})\ \ (\boldsymbol{distinct\ ?x\ ?y})\ \ (\boldsymbol{parent\ ?x\ ?z})\ \ (\boldsymbol{parent\ ?y\ ?z}))$$

3.1.1.6 Συμβολισμός Διάζευξης (OR)

Προηγουμένως αναφέρθηκε ότι για να αληθεύει μια σχέση, πρέπει να αληθεύει και κάθε σχέση στη λίστα των περαιτέρω σχέσεων. Ενδέχεται όμως να εφαρμοστεί και διάζευξη, δηλαδή, για να ισχύει μια σχέση, αρκεί να ισχύει μια σχέση μόνο από τη διάζευξη. Στη datalog η διάζευξη δηλώνεται με το ερωτηματικό “;”. Στο παρακάτω παράδειγμα:

$$\boldsymbol{illegal(white, X, Y)}$$
$$\boldsymbol{: -\ true(control(white)), (true(cell(X,Y,x))\ ;\ true(cell(X,Y,o)))}$$

Για να αληθεύει η σχέση $illegal$ θα πρέπει να αληθεύει η σχέση $\boldsymbol{true(control(white))}$ και να αληθεύει είτε η σχέση $\boldsymbol{true(cell(X,Y,x))}$, είτε η σχέση $\boldsymbol{true(cell(X,Y,o))}$.

Στη μορφή ανταλλαγής πληροφορίας, η διάζευξη σχέσεων ορίζεται με το σύμβολο OR και περικλείοντας τις σχέσεις με παρενθέσεις. Το παραπάνω παράδειγμα μετατρέπεται ως εξής:

$$(\leq \ (\boldsymbol{illegal\ white}\ ?\boldsymbol{x}\ ?\boldsymbol{y})\ (\boldsymbol{true}\ (\boldsymbol{control\ white}))$$
$$(\boldsymbol{OR}\ (\boldsymbol{true}\ (\boldsymbol{cell}\ ?\boldsymbol{x}\ ?\boldsymbol{y}\ \boldsymbol{x}))\ (\boldsymbol{true}\ (\boldsymbol{cell}\ ?\boldsymbol{x}\ ?\boldsymbol{y}\ ?\boldsymbol{o}))))$$

3.1.1.7 Συμβολισμός Άρνησης (NOT)

Πολλές φορές, για να αληθεύει μια σχέση πρέπει να μην αληθεύει κάποια άλλη σχέση στη λίστα των περαιτέρω σχέσεων. Η άρνηση στη datalog δηλώνεται με το σύμβολο "not", είτε ως εξής: $\boldsymbol{terminal} :- \boldsymbol{not}(\boldsymbol{open})$, είτε ως εξής: $\boldsymbol{terminal} :- \boldsymbol{not\ open}$.

Και στη μορφή ανταλλαγής γνώσης, χρησιμοποιείται το σύμβολο NOT, περικλείοντας, και σε αυτή την περίπτωση, το σύμβολο NOT και τη σχέση της λίστας με παρενθέσεις, όπως στο παρακάτω παράδειγμα:

$$(<= \ \boldsymbol{terminal}\ (\boldsymbol{NOT\ open}))$$

3.1.2 Κριτήρια Επιλογής της Μορφής Ανταλλαγής Γνώσης

Η μορφή ανταλλαγής γνώσης προσφέρει τις ίδιες δυνατότητες με τη datalog, απλά σε διαφορετική μορφή. Ο κυριότερος λόγος για τον οποίο επιλέξαμε να χρησιμοποιήσουμε τη μορφή ανταλλαγής γνώσης είναι ότι καθιστά πιο εύκολη και δομημένη τη διαδικασία της συντακτικής ανάλυσης. Ένας ακόμα λόγος είναι ότι υπάρχουν στο διαδίκτυο πολλά παιχνίδια τα οποία περιγράφονται σε μορφή ανταλλαγής γνώσης.

3.1.3 Λεκτική Ανάλυση

Η λεκτική ανάλυση είναι η διαδικασία κατά την οποία το πρόγραμμα διαβάζει το κείμενο του αρχείου εισόδου και σχηματίζει λεξήματα, τα οποία αποτελούν στοιχεία της γλώσσας (στη συγκεκριμένη περίπτωση, η γλώσσα είναι η μορφή ανταλλαγής γνώσης).

Ο λεκτικός αναλυτής (lexer) αυτό που κάνει ουσιαστικά είναι να παίρνει με τη σειρά κάθε λέξημα του αρχείου εισόδου, όπου βρίσκεται η περιγραφή του παιχνιδιού που εκτελείται, εξετάζει την φύση του και έπειτα τα "πακετάρει" στην κατάλληλη κατηγορία ανάλογα με το που ανήκει καθένα από αυτά. Τέλος τα στέλνει στον συντακτικό αναλυτή για να τα εξετάσει.

Ο χαρακτηρισμός του κάθε λεξήματος (token) του αρχείου ανήκει σε μία από τις παρακάτω κατηγορίες.

- **Αριστερή παρένθεση**, στην περίπτωση που το λέξημα είναι η αριστερή παρένθεση ¨(¨.
- **Δεξιά παρένθεση**, στην περίπτωση που το λέξημα είναι η δεξιά παρένθεση ¨)¨.
- **Συνεπαγωγή**, στην περίπτωση που έχουμε το σύμβολο "<=" . Σε μία λογική πρόταση αυτό το σύμβολο προηγείται και δηλώνει ότι για να ισχύει το πρώτο μέρος της πρότασης, πρέπει να είναι αληθές το σύνολο των κατηγορημάτων που ακολουθούν. Στο παρακάτω παράδειγμα που είναι ένα μικρό απόσπασμα από το παιχνίδι «τρίλιζα» αποδεικνύεται η αλήθεια των παραπάνω:

$$(\leq \ (DIAGONAL\ ?X)\ (TRUE\ (CELL\ 1\ 3\ ?X))$$

$$(TRUE\ (CELL\ 2\ 2\ ?X))\ (TRUE\ (CELL\ 3\ 1\ ?X)))$$

Το παραπάνω παράδειγμα ουσιαστικά περιγράφει τι πρέπει να ισχύει για να έχουμε διαγώνιο.

- **Μεταβλητή**, μπορεί να θεωρηθεί οποιοδήποτε λέξημα συντροφεύεται από το σύμβολο "?" στο μπροστινό του μέρος (πχ, ?X) και χρησιμοποιείται για να περιγράψει καθένα από τους άγνωστους όρους.
- **Αρνητικό Λέξημα**, στην κατηγορία αυτή καταχωρείται το λέξημα "not" όπου βρεθεί μέσα στο αρχείο. Ουσιαστικά το συγκεκριμένο λέξημα σε μία πρόταση συμβολίζει ότι δεν θα πρέπει να ισχύει η συγκεκριμένη πρόταση. Ένα παράδειγμα από την περιγραφή του παιχνιδιού «Τρίλιζα» είναι το παρακάτω.

$$(<=\ TERMINAL\ (NOT\ OPEN))$$

Που εκφράζει ότι μία περίπτωση για να έχουμε τελική κατάσταση στο παιχνίδι θα πρέπει να μην είναι κανένα κελί κενό, η διαφορετικά κανένα κελί "ανοιχτό" .

- **Διαζευκτικό λέξημα**, το οποίο παρόμοια με το αρνητικό λέξημα καταχωρείται το λέξημα "or" όπου υπάρχει μέσα στο αρχείο, και εκφράζει τη διάζευξη μέσα σε δύο η παραπάνω προτάσεις. Για παράδειγμα έχουμε το παρακάτω απόσπασμα από το παιχνίδι «Τρίλιζα».

$$(\leq\ (NEXT\ (CELL\ ?M\ ?N\ B))\ (DOES\ ?W\ (MARK\ ?J\ ?K))$$

$$(TRUE\ (CELL\ ?M\ ?N\ B))\ (OR\ (DISTINCT\ ?M\ ?J)\ (DISTINCT\ ?N\ ?K)))$$

Στην συγκεκριμένη λογική πρόταση εκφράζεται ότι θα πρέπει να είναι διαφορετικές οι μεταβλητές (?M ?J) μεταξύ τους ή οι μεταβλητές (?N ?K) και αρκεί να ισχύει μία από τις δύο προτάσεις.

- **Λεκτικό Λέξημα**, είναι κάθε λέξημα το οποίο δεν ανήκει σε καμία από τις παραπάνω κατηγορίες ενώ παράλληλα δεν είναι κάποιος άγνωστος χαρακτήρας. Λέξη μπορεί να χαρακτηριστεί επίσης οποιοδήποτε νούμερο η επιτρεπτός χαρακτήρας.

Αξίζει να αναφερθεί ότι ο λεκτικός αναλυτής αναγνωρίζει τα σχόλια και κάθε αλλαγή γραμμής η κενό και τα προσπερνάει.

Γενικά ο λεκτικός αναλυτής μετά την αρχικοποίηση του καλείται μέσω της συνάρτησης $NextToken()$ η οποία είναι υπεύθυνη να επιστρέφει ένα - ένα τα λεξήματα βάσει των παραπάνω κατηγοριών που αναφέρθηκαν. Η διαδικασία την οποία ακολουθεί η συγκεκριμένη συνάρτηση είναι η παρακάτω:

1. Έλεγχος για το **τέλος του αρχείου** (end of file) και επιστροφή κατάλληλου μηνύματος σε περίπτωση αληθείας.
2. Έλεγχος του σωρού για χαμένους χαρακτήρες και έλεγχος αυτών. Ο σωρός χρησιμοποιείται ανάμεσα στους ελέγχους για τους χαρακτήρες για να μην υπάρχουν απώλειες αυτών, οπότε όπου κρίνεται απαραίτητο τους αποθηκεύει. Στην περίπτωση που ο σωρός είναι άδειος, τότε διαβάζεται χαρακτήρας από το αρχείο.
    - Για εισαγωγή και εξαγωγή από τον σωρό χρησιμοποιούνται ειδικές συναρτήσεις που έχουν δημιουργηθεί γι' αυτόν το σκοπό.
3. Έλεγχος για αλλαγή γραμμής. Σε περίπτωση αληθείας ο αντίστοιχος χαρακτήρας παρακάμπτεται.

4. Έλεγχος για την ύπαρξη κάθε είδους σχολίων. Σε περίπτωση ύπαρξης αυτών τότε παρακάμπτονται.
5. Έλεγχος για ύπαρξη κενού όπου σε περίπτωση ύπαρξης αυτού παρακάμπτεται.
6. Έλεγχος για την ύπαρξη αριστερής η δεξιάς παρένθεσης. Σε περίπτωση που ο χαρακτήρας αυτός ανήκει σε μία από τις δύο παραπάνω κατηγορίες, τότε επιστρέφεται στον συντακτικό αναλυτή στην κατάλληλη κατηγορία.
7. Έλεγχος για την ύπαρξη του συμβόλου συνεπαγωγής, όπου σε περίπτωση ύπαρξης αυτού επιστρέφεται στην αντίστοιχη κατηγορία στον συντακτικό αναλυτή.
8. Έλεγχος για την ύπαρξη μεταβλητής, όπου στην περίπτωση που το σύνολο των χαρακτήρων αυτών αποτελούν μεταβλητή τότε το συγκεκριμένο λέξημα επιστρέφει στον συντακτικό αναλυτή στην αντίστοιχη κατηγορία που ανήκει.
9. Έλεγχος για την κατηγορία την οποία ανήκει το λέξημα, οι οποίες μπορεί να είναι είτε Αρνητικό λέξημα είτε Διαζευκτικό λέξημα είτε να είναι Λεκτικό λέξημα. Όποτε επιστρέφει στον συντακτικό αναλυτή ανάλογα με την κατηγορία στην οποία εντάσσεται.
10. Τέλος εφόσον ο χαρακτήρας μας δεν ακολουθεί μία από τις παραπάνω κατηγορίες, τότε χαρακτηρίζεται σαν άγνωστος χαρακτήρας και επιστρέφεται αντίστοιχο αναγνωριστικό μήνυμα σφάλματος.

3.1.4 Συντακτική Ανάλυση

Η συντακτική ανάλυση (parsing) είναι η διαδικασία στην οποία ένας μεταγλωττιστής δέχεται τα λεξήματα ενός πηγαίου αρχείου κώδικα και ελέγχει αν ικανοποιούν το συντακτικό της γλώσσας. Για παράδειγμα, σε έναν γενικό παίκτη που δέχεται αρχεία κώδικα σε μορφή ανταλλαγής γνώσης, θα θεωρούσε σωστό συντακτικό σχηματισμό τον εξής:

$$left\_parenthesis\ \ word\ \ word\ \ variable\ right\_parenthesis$$

που σημαίνει ($σχέση\ όρισμα\_σταθερά\ όρισμα\_μεταβλητή$), ενώ θα έβγαζε συντακτικό σφάλμα για τον εξής συντακτικό σχηματισμό:

$$right\_parenthesis\ \ word\ \ word\ \ variable\ left\_parenthesis$$

που δεν ταιριάζει σε περιγραφή παιχνιδιού σε μορφή ανταλλαγής γνώσης.

Στο SweetGGP η διαδικασία της συντακτικής ανάλυσης γίνεται σε δύο επίπεδα: το συντακτικό αναλυτή πρώτου στρώματος (First Layer Parser) και το συντακτικό αναλυτή δεύτερου στρώματος (Second Layer Parser).

Ο συντακτικός αναλυτής πρώτου στρώματος αναλαμβάνει να διαβάσει όλα τα λεξήματα του πηγαίου αρχείου του παιχνιδιού και κατασκευάζει δύο κατηγορίες εκφράσεων:

- Τις απλές εκφράσεις.
- Τις λίστες.

Οι απλές εκφράσεις περιέχουν ένα λέξημα, το οποίο μπορεί να είναι:

- Λέξη.
- Μεταβλητή.
- Σύμβολο NOT.

- Σύμβολο OR.
- Σύμβολο συνεπαγωγής.

Οι λίστες περιέχουν, όπως δηλώνει το όνομα τους, μια λίστα απο άλλες εκφράσεις, που μπορούν να είναι είτε απλές εκφράσεις, είτε λίστες. Οι λίστες περιέχουν τα κομμάτια της περιγραφής του παιχνιδιού τα οποία βρίσκονται μέσα σε παρενθέσεις. Για παράδειγμα, η παρακάτω έκφραση:

$$(legal\ ?p\ (mark\ ?x\ ?y))$$

είναι μια λίστα που περιέχει την απλή έκφραση $legal$, την απλή έκφραση $?p$ και τη λίστα η οποία, με τη σειρά της, περιέχει τις απλές εκφράσεις $mark$, $?x$ και $?y$.

Ο συντακτικός αναλυτής δεύτερου στρώματος αναλαμβάνει να πάρει τις εκφράσεις που κατασκευάζει ο συντακτικός αναλυτής πρώτου στρώματος και να κατασκευάσει, με βάση αυτές, αντικείμενα δομών στο πρόγραμμα που αναπαριστούν τα εξής:

- Σταθερές.
- Μεταβλητές.
- Συναρτησιακά σύμβολα.
- Σχέσεις.
- Συζεύξεις.
- Διαζεύξεις.
- Αρνήσεις.

Στη συνέχεια ακολουθεί αναλυτική περιγραφή για τα δύο στρώματα συντακτικής ανάλυσης.

3.1.4.1 Συντακτικός Αναλυτής Πρώτου Στρώματος (First Layer Parser)

Η διαδικασία του συντακτικού αναλυτή πρώτου στρώματος είναι πολύ απλή:

1. Όσο υπάρχουν λεξήματα για να διαβαστούν:
    1.1. Διαβάζεται το επόμενο λέξημα.
    1.2. Αν το λέξημα δεν είναι αριστερή παρένθεση
        1.2.1. Αν το λέξημα είναι δεξιά παρένθεση:
            1.2.1.1. Επιστρέφεται συντακτικό σφάλμα.
        1.2.2. Αλλιώς:
            1.2.2.1. Επιστρέφεται απλή έκφραση με βάση το λέξημα.
    1.3. Αλλιώς:
        1.3.1. Κατασκευή έκφρασης λίστας.
        1.3.2. Όσο δεν έχει διαβαστεί λέξημα που να είναι δεξιά παρένθεση:
            1.3.2.1. Διαβάζεται το επόμενο λέξημα.
            1.3.2.2. Αν είναι το τέλος της εισόδου:
                1.3.2.2.1. Επιστρέφεται συντακτικό σφάλμα.
            1.3.2.3. Αν δεν είναι δεξιά παρένθεση:
                1.3.2.3.1. Επιστροφή στο βήμα 1.2 και αποθήκευση της έκφρασης που επιστρέφεται στη λίστα.
        1.3.3. Επιστρέφεται η λίστα.

3.1.4.2 Συντακτικός Αναλυτής Δεύτερου Στρώματος (Second Layer Parser)

Ο συντακτικός αναλυτής δεύτερου στρώματος αναλαμβάνει στη συνέχεια να κατασκευάσει τις δομές οι οποίες αναπαριστούν τις οντότητες του παιχνιδιού (σχέσεις, μεταβλητές, σταθερές, κτλ.), ακολουθώντας την παρακάτω διαδικασία:

1. Για κάθε έκφραση που έχει κατασκευαστεί στο πρώτο στρώμα:
    1.1. Καλείται η διαδικασία ανάλυσης έκφρασης με βάση τη συγκεκριμένη έκφραση.

Η διαδικασία ανάλυσης έκφρασης είναι η εξής:

1. Αν η έκφραση δεν είναι λίστα:
    1.1. Επιστρέφεται συντακτικό σφάλμα.
2. Αν η έκφραση είναι λίστα και το πρώτο στοιχείο περιέχει το σύμβολο συνεπαγωγής:
    2.1. Καλείται η διαδικασία ανάλυσης λίστας.
3. Αλλίως:
    3.1. Καλείται η διαδικασία ανάλυσης σχέσης.

Η διαδικασία ανάλυσης λίστας είναι η εξής:

1. Αν η λίστα περιέχει μόνο 1 στοιχείο:
    1.1. Επιστρέφεται συντακτικό σφάλμα.
2. Καλείται η διαδικασία ανάλυσης σχέσης με βάση το δεύτερο στοιχείο της λίστας και κατασκευάζεται νέα σχέση με βάση το αποτέλεσμα.
3. Για κάθε μία απο τις υπόλοιπες εκφράσεις της λίστας:
    3.1. Καλείται η διαδικασία ανάλυσης εκφράσεων σύζευξης με βάση τη συγκεκριμένη έκφραση και το αποτέλεσμα αποθηκεύεται στις εκφράσεις που αποτελούν τη σύζευξη της σχέσης.
4. Επιστρέφεται ως αποτέλεσμα η σχέση.

Η διαδικασία ανάλυσης σύζευξης είναι η εξής:

1. Άν η έκφραση που εξετάζεται δεν είναι λίστα:
    1.1. Σε αυτή την περίπτωση πρόκειται για σχέση χωρίς ορίσματα, οπότε καλείται η διαδικασία ανάλυσης σχέσης για τη συγκεκριμένη έκφραση.
2. Επιλέγεται το πρώτο στοιχείο της έκφρασης (που είναι λίστα).
3. Αν αυτό το στοιχείο είναι λίστα:
    3.1. Επιστρέφεται συντακτικό σφάλμα.
4. Αν το στοιχείο δεν είναι ούτε λέξη, ούτε κάποιο απο τα ειδικά σύμβολα OR και NOT:
    4.1. Επιστρέφεται συντακτικό σφάλμα.
5. Αν το στοιχείο είναι το σύμβολο διάζευξης OR:
    5.1. Καλείται η διαδικασία ανάλυσης διάζευξης.
6. Αλλιώς αν το στοιχείο είναι το σύμβολο άρνησης NOT:
    6.1. Καλείται η διαδικασία ανάλυσης άρνησης.
7. Σε διαφορετική περίπτωση καλείται η διαδικασία ανάλυσης έκφρασης.

Η διαδικασία ανάλυσης άρνησης είναι η εξής:

1. Το όρισμα της διαδικασίας είναι λίστα.
2. Αν η λίστα δεν έχει ακριβώς δύο ορίσματα:

2.1. Επιστρέφεται συντακτικό σφάλμα.
3. Κατασκευάζεται νέα άρνηση.
4. Επιλέγεται το δεύτερο στοιχείο της λίστας.
5. Καλείται η διαδικασία ανάλυσης σύζευξης για το επιλεγμένο στοιχείο της λίστας και το αποτέλεσμα που επιστρέφεται αποθηκεύεται ως η έκφραση της άρνησης.
6. Επιστρέφεται η άρνηση ως αποτέλεσμα.

Η διαδικασία ανάλυσης διάζευξης είναι η εξής:

1. Το όρισμα της διαδικασίας είναι λίστα.
2. Αν η λίστα έχει λιγότερα απο τρία στοιχεία:
   2.1. Επιστρέφεται συντακτικό σφάλμα.
3. Κατασκευάζεται νέα διάζευξη.
4. Για κάθε στοιχείο της λίστας μετά το πρώτο (που είναι το σύμβολο OR):
   4.1. Καλείται η διαδικασία ανάλυσης σύζευξης.
   4.2. Το αποτέλεσμα αποθηκεύεται στα στοιχεία της διάζευξης.
5. Επιστρέφεται η διάζευξη ως αποτέλεσμα.

Η διαδικασία ανάλυσης σχέσης είναι η εξής:

1. Αν το όρισμα είναι απλή έκφραση:
   1.1. Κατασκευάζεται νέα σχέση.
   1.2. Σαν όνομα της σχέσης δίνεται το όνομα του λεξήματος της έκφρασης.
   1.3. Επιστρέφεται η σχέση ως αποτέλεσμα.
2. Αλλιώς:
   2.1. Κατασκευάζεται νέα σχέση.
   2.2. Επιλέγεται το πρώτο στοιχείο του ορίσματος (που είναι λίστα σε αυτή την περίπτωση).
   2.3. Σαν όνομα της σχέσης δίνεται το όνομα του λεξήματος του στοιχείου.
   2.4. Για κάθε στοιχείο του ορίσματος μετά το πρώτο (που είναι το όνομα της σχέσης):
      2.4.1. Αν το στοιχείο είναι απλή έκφραση:
         2.4.1.1. Καλείται η διαδικασία ανάλυσης απλής έκφρασης.
      2.4.2. Αλλιώς:
         2.4.2.1. Καλείται η διαδικασία ανάλυσης συναρτησιακού συμβόλου.
      2.4.3. Το αποτέλεσμα αποθηκεύεται στα ορίσματα της σχέσης.
   2.5. Επιστρέφεται η σχέση ως αποτέλεσμα.

Η διαδικασία ανάλυσης συναρτησιακού συμβόλου είναι η εξής:

1. Το όρισμα είναι λίστα.
2. Για κάθε στοιχείο του ορίσματος:
   2.1. Αν το στοιχείο είναι λίστα:
      2.1.1. Επιστρέφεται συντακτικό σφάλμα.
3. Αν το όρισμα έχει μόνο ένα στοιχείο:
   3.1. Επιστρέφεται συντακτικό σφάλμα:
4. Κατασκευάζεται νέο συναρτησιακό σύμβολο.

5. Σαν όνομα του συναρτησιακού συμβόλου δίνεται το πρώτο στοιχείο του ορίσματος.
6. Για κάθε στοιχείο του ορίσματος μετά το πρώτο (που είναι το όνομα του συναρτησιακού συμβόλου):
    6.1. Το στοιχείο αποθηκεύεται στα ορίσματα του συναρτησιακού συμβόλου.
7. Επιστρέφεται το συναρτησιακό σύμβολο ως αποτέλεσμα.

Η διαδικασία ανάλυσης απλής σχέσης είναι η εξής:

1. Αν το λέξημα που περιέχεται στο όρισμα παριστάνει μεταβλητή:
    1.1. Κατασκευάζεται νέα μεταβλητή που έχει σαν όνομα, το όνομα του ορίσματος.
    1.2. Επιστρέφεται η μεταβλητή ως αποτέλεσμα.
2. Αλλιώς:
    2.1. Κατασκευάζεται νέα σταθερά που έχει σαν όνομα, το όνομα του ορίσματος.
    2.2. Επιστρέφεται η σταθερά ως αποτέλεσμα.

3.2 Αποδεικτικός Αναλυτής Σχέσεων

Ο αποδεικτικός αναλυτής σχέσεων (relation prover), ή διαφορετικά prover, είναι ένα πολύ σημαντικό κομμάτι του προγράμματος καθώς είναι υπεύθυνος για την απόδειξη κάθε λογικής πράξης όπου απαιτείται. Επιπλέον αναλαμβάνει κάθε σχέση που του τίθεται και φροντίζει να αποδείξει την αλήθεια αυτής καθώς επίσης να βρει και τις λύσεις της εφόσον είναι αληθής. Το έργο του διαχωρίζεται σε πέντε κομμάτια καθένα από τα οποία αναλαμβάνουν ένα διαφορετικό σκοπό, μια διαφορετική διαδικασία ή εργασία, και συλλογικά έχουν σκοπό την επίτευξη των παραπάνω. Τα πέντε αυτά μέρη θα μπορούσαμε να τα καλέσουμε και επίπεδα, γιατί χρησιμοποιεί το ένα το άλλο. Παρόλα αυτά διαχωρίζονται σε πέντε διαφορετικές κλάσεις και είναι οι εξής:

1. *RelationCollection*
2. *KnowledgeBase*
3. *Unifier*
4. *RelationProver*
5. *GameState*

3.2.1 RelationCollection

Ένα δομικό κομμάτι του συντακτικού αναλυτή είναι η κλάση $RelationCollection$ η οποία ουσιαστικά είναι ότι δηλώνει το όνομα της, μια συλλογή από σχέσεις (relations). Πιο συγκεκριμένα είναι ένας ταξινομημένος πίνακας από σχέσεις. Ο πίνακας αυτός αποτελείται από το όνομα της κάθε σχέσης της περιγραφής του παιχνιδιού καθώς επίσης και την λίστα των τελεστών αυτής που μπορεί να είναι και επιπλέον σχέσεις. Η κλάση αυτή έχει κάποιες βασικές συναρτήσεις για την περιγραφή της. Οι συναρτήσεις αυτές είναι οι παρακάτω:

1. Η συνάρτηση $\boldsymbol{Get}(\ \boldsymbol{string\ name})$ τύπου λίστας σχέσεων λαμβάνει σαν όρισμα το όνομα μίας σχέσης και εφόσον υπάρχει καταχωρημένο το όνομα αυτό στην

σχετική λίστα της κλάσης επιστρέφει μία λίστα από σχέσεις σχετικά με το όνομα αυτό, ειδάλλως επιστρέφεται μία κενή λίστα από σχέσεις.

2. Η συνάρτηση $\boldsymbol{Add(string\ name, Relation\ expr)}$ τύπου void είναι εκείνη η οποία λαμβάνει σαν είσοδο το όνομα μίας σχέσης καθώς επίσης και όλη την έκφραση αυτής και ελέγχει αν το συγκεκριμένο όνομα είναι καταχωρημένο στη λίστα της κλάσης με τις σχέσεις. Εφόσον είναι τότε πάει στην θέση της λίστας που είναι καταχωρημένες οι σχέσεις σχετικές με το όνομα αυτό και καταχωρεί τη συγκεκριμένη έκφραση εκεί, ειδάλλως την καταχωρεί στο τέλος της λίστας.
3. Η συνάρτηση $\boldsymbol{Sort()}$ τύπου void είναι εκείνη η συνάρτηση η οποία είναι υπεύθυνη για την ταξινόμηση της λίστας με τις σχέσεις βάσει του ονόματος τους.

3.2.2 KnowledgeBase

Η συγκεκριμένη κλάση είναι ένα πολύ βασικό κομμάτι του προγράμματος καθώς είναι υπεύθυνη για την ανάγνωση του αρχείου εισόδου, που περιέχει τους κανόνες του τρέχοντος παιχνιδιού, την κλήση του λεκτικού και συντακτικού αναλυτή καθώς επίσης και την συμπλήρωση της λίστας με τις σχέσεις της κλάσης $RelationCollection$ που αναφέρθηκε προηγουμένως. Περιλαμβάνει με την σειρά τις παρακάτω διαδικασίες – συναρτήσεις.

1. Αρχικά στην δημιουργία της συγκεκριμένης κλάσης ελέγχεται το όνομα του αρχείου εισόδου το οποίο έχει δοθεί από την είσοδο αν όντως υπάρχει, και εκτυπώνεται ανάλογο μήνυμα σφάλματος στην περίπτωση που δεν υπάρχει. Εφόσον το αρχείο βρεθεί τότε καλείται η κλάση του λεκτικού αναλυτή για την δημιουργία αυτού και επιπλέον δημιουργείται ένα δείκτης στο αρχείο εισόδου με το παιχνίδι. Έπειτα η κλάση του λεκτικού αναλυτή δίνεται σαν είσοδο στον συντακτικό αναλυτή ο οποίος αρχίζει να παίρνει λέξημα – λέξημα από το αρχείο και να τα εξετάζει. Τέλος καλείται η συνάρτηση $FillRelations()$ από την τρέχουσα κλάση η οποία θα περιγραφεί αμέσως παρακάτω.
2. Η συνάρτηση $\boldsymbol{FillRelations(List < Relation > relation)}$ τύπου void παίρνει σαν είσοδο μία λίστα από σχέσεις και ακολουθεί την παρακάτω διαδικασία. Αρχικά δημιουργεί ένα αντικείμενο τύπου $RelationCollection$ που αναφέραμε λίγο παραπάνω και έπειτα παίρνει την λίστα από τις σχέσεις οι οποίες έχουν δοθεί σαν είσοδο στην συνάρτηση και τις προσθέτει στη λίστα με την βοήθεια της συνάρτησης $Add()$ της κλάσης $RelationCollection$. Αφού τις προσθέσει όλες, καλεί την $Sort()$ πάλι της ίδιας κλάσης για να τα ταξινομήσει.
3. Η συνάρτηση $\boldsymbol{UnifiableRelations(Relation\ relation)}$ είναι τύπου λίστας $Unifier$ η οποία αναφέρθηκε και παραπάνω και είναι υπεύθυνη για την ενοποίηση των σχέσεων που θα της δοθούν. Η διαδικασία την οποία ακολουθεί η συνάρτηση αυτή είναι η εξής: Αρχικά δημιουργείται ένα αντικείμενο τύπου λίστας $Unifier$ και έπειτα δημιουργείται ένα αντικείμενο τύπου λίστας $Relations$ στην οποία αποθηκεύονται οι σχέσεις από τη συνάρτηση $Get()$ της κλάσης $RelationCollection$ βάση του ονόματος των σχέσεων που θα δοθούν στην συνάρτηση. Έπειτα για κάθε από τις σχέσεις καλείται η συνάρτηση $UnifyRelations()$ της κλάσης $Unifier$ με σκοπό να επιτευχθεί η ενοποίηση των σχέσεων και έπειτα εφόσον έχουμε αποτελέσματα, προστίθενται στην λίστα $Unifier$ που δημιουργήθηκε στην αρχή της συνάρτησης. Στο τέλος επιστρέφεται η λίστα.

3.2.3 Unifier

Το πιο σημαντικό και κρίσιμο σημείο στην απόδειξη αληθείας μιας σχέσης είναι η ενοποίηση (unification). Έστω η ακόλουθη περίπτωση: πρέπει να αποδειχτεί ότι η σχέση add(1, 2, X) είναι αληθής και να βρεθούν και λύσεις για το X. Στη βάση γνώσης υπάρχει η σχέση add(1, 2, 3). Η διαδικασία της ενοποίησης είναι υπεύθυνη ώστε να αντιλαμβάνεται πότε δύο σχέσεις «ταιριάζουν» και να δίνει κατάλληλες τιμές στις μεταβλητές ώστε να είναι ίδιες οι σχέσεις. Στο προηγούμενο παράδειγμα η λύση είναι { X = 3} προκειμένου να είναι ίδιες οι σχέσεις.

Η διαδικασία της ενοποίησης αποτελείται από τα εξής βήματα για να ελέγξει αν δύο σχέσεις «ταιριάζουν», καθώς και για να δώσει τιμές στις μεταβλητές:

1. Αν οι σχέσεις έχουν διαφορετικό όνομα, τότε δεν ταιριάζουν.
2. Αν οι σχέσεις δεν έχουν ίδιο αριθμό ορισμάτων, τότε δεν ταιριάζουν.
3. Αλλάζουν τα ονόματα των ορισμάτων, των σχέσεων, που είναι μεταβλητές, ώστε να μην υπάρχουν συγκρούσεις ονομάτων κατά τη διαδικασία ενοποίησης μεταβλητών. Για παράδειγμα, οι σχέσεις add(1, 2, X) και add(1, X, 3), θα γινόντουσαν add(1, 2, X1) και add(1, X2, 3).
4. Για κάθε όρισμα των σχέσεων ακολουθείται η εξής διαδικασία:
    4.1. Αν και τα δύο ορίσματα είναι σταθερές και τα ονόματα τους είναι διαφορετικά, τότε οι σχέσεις δεν ταιριάζουν.
    4.2. Αν κάποιο απο τα δύο ορίσματα είναι μεταβλητή, τότε ακολουθείται η διαδικασία ενοποίησης μεταβλητών.
    4.3. Αν και τα δύο ορίσματα είναι συναρτησιακά σύμβολα, τότε ακολουθείται η διαδικασία ενοποίησης συναρτησιακών συμβόλων. Η διαδικασία ενοποίησης συναρτησιακών συμβόλων είναι η ίδια με αυτή τη διαδικασία, μόνο που αντί για σχέσεις ελέγχονται συναρτησιακά σύμβολα, γι' αυτό το λόγο παραλείπεται η περιγραφή της.
    4.4. Σε αντίθετη περίπτωση, οι σχέσεις δεν ταιριάζουν.

Προκειμένου να είναι εφικτή η ενοποίηση μεταβλητών, είναι αναγκαία η χρήση δομής η οποία κρατάει για κάθε μεταβλητή την τιμή (σταθερή, μεταβλητή ή συναρτησιακό σύμβολο) στην οποία έχει αντιστοιχιστεί. Η διαδικασία ενοποίησης μεταβλητών περιλαμβάνει τα εξής βήματα (ως πρώτη μεταβλητή ορίζεται η μεταβλητή της πρώτης σχέσης, ως δεύτερη μεταβλητή ορίζεται η μεταβλητή της δεύτερης σχέσης, ως πρώτη τιμή ορίζεται η τιμή της πρώτης μεταβλητής και ως δεύτερη τιμή ορίζεται η τιμή της δεύτερης μεταβλητής):

1. Αν στην πρώτη μεταβλητή έχει αντιστοιχηθεί κάποια τιμή, τότε κρατείται αυτή ως τιμή της μεταβλητής, αλλιώς κρατείται το όνομα της μεταβλητής.
2. Αν στη δεύτερη μεταβλητή έχει αντιστοιχηθεί κάποια τιμή, τότε κρατείται αυτή ως τιμή της μεταβλητής, αλλιώς κρατείται το όνομα της μεταβλητής.
3. Αν και οι δύο τιμές είναι σταθερές και δεν έχουν ίδιο όνομα, τότε οι σχέσεις δεν ταιριάζουν.
4. Αν και οι δύο τιμές είναι συναρτησιακά σύμβολα, τότε ακολουθείται η διαδικασία ενοποίησης συναρτησιακών συμβόλων.
5. Η πρώτη μεταβλητή αντιστοιχίζεται στη δεύτερη τιμή. Αν η πρώτη τιμή είναι μεταβλητή, τότε αντιστοιχίζεται και αυτή στη δεύτερη τιμή. Έπειτα, σαρώνεται η δομή με τις αντιστοιχίες μεταβλητών και τιμών. Για κάθε μεταβλητή στη δομή:
    5.1. Αν η τιμή της είναι η πρώτη μεταβλητή, τότε της ανατίθεται η δεύτερη τιμή.
    5.2. Αν η τιμή της είναι συναρτησιακό σύμβολο, τότε σαρώνεται ώστε αν έχει σαν όρισμα την πρώτη μεταβλητή, τότε να αντικατασταθεί με τη

δεύτερη τιμή. Αν γίνει τέτοια αντικατάσταση και η δεύτερη τιμή είναι συναρτησιακό σύμβολο, τότε οι σχέσεις δεν ταιριάζουν γιατί παραβιάζεται βασικός κανόνας της GDL που δεν επιτρέπει ένα συναρτησιακό σύμβολο να έχει σαν όρισμα ένα άλλο συναρτησιακό σύμβολο.

6. Η δεύτερη μεταβλητή αντιστοιχίζεται στην πρώτη τιμή. Αν η δεύτερη τιμή είναι μεταβλητή, τότε αντιστοιχίζεται και αυτή στην πρώτη τιμή.

3.2.4 RelationProver

Η συγκεκριμένη κλάση ουσιαστικά αναλαμβάνει να αποδείξει αν μία σχέση είναι αληθής η όχι χρησιμοποιώντας τα παραπάνω δομικά στοιχεία που αναφέρθηκαν, και τις λειτουργίες αυτών. Αποτελείται από πέντε βασικές συναρτήσεις για να πετύχουν αυτό το σκοπό και είναι οι παρακάτω:

Η συνάρτηση $\boldsymbol{Prove}$ ($\boldsymbol{Expression\ expr}$) η οποία είναι υπεύθυνη να εντάξει την σχέση που της δίνεται σε μία από τις τέσσερις παρακάτω κατηγορίες ανάλογα με την κατηγορία που ανήκει:

- ***Relation*** είναι μια απλή σχέση.
- ***Conjuction*** όπου ουσιαστικά είναι μία σύζευξη από σχέσεις.
- ***Disjunction*** όπου είναι μια διάζευξη από σχέσεις.
- ***Negation*** όπου είναι η αρνητική έκφρασης μίας σχέσης και πρέπει να αποδειχτεί η αλήθεια αυτής.

Οπότε αυτό που κάνει ουσιαστικά η συνάρτηση Prove() είναι να ελέγχει την έκφραση που της δίνεται αν ανήκει σε μία από τις παραπάνω κατηγορίες και ανάλογα με την κατηγορία που ανήκει καλείται μία από τις επόμενες συναρτήσεις.

Η συνάρτηση $\boldsymbol{ProveRelation}(\boldsymbol{Relation\ relation})$ τύπου λίστας $Unifier$ είναι η υπεύθυνη συνάρτηση για την απόδειξη αληθείας σε μία σχέση καθώς επίσης και για την εύρεση των τιμών που την επιλύουν. Η διαδικασία που ακολουθεί αρχίζει με τον έλεγχο αν η σχέση είναι τύπου $\boldsymbol{DISTINCT}$ και ελέγχει τις δύο παραμέτρους της σχέσης αν είναι διαφορετικές μεταξύ τους. Στην περίπτωση που δεν είναι διαφορετικές επιστρέφεται μία κενή λίστα τύπου κλάσης $Unifier$ ειδάλλως μία λίστα τύπου $Unifier$ που εμπεριέχει όμως ένα αντικείμενο κλάσης $Unifier$. Σε περίπτωση όμως που δεν έχουμε τον όρο DISTINCT στο relation τότε η διαδικασία είναι διαφορετική. Έχουμε μια λίστα με όλες τις ενοποιημένες σχέσεις βάσει της συγκεκριμένης σχέσης που έχει δοθεί για εξέταση και μία κενή λίστα τύπου κλάσης $Unifier$ η οποία θα περιέχει της λύσεις της σχέσης που είναι υπό εξέταση. Έπειτα ελέγχεται αν η κάθε σχέση από τη λίστα των ενοποιημένων σχέσεων που είναι αποθηκευμένα στην κλάση $KnowledgeBase$ έχει παραπάνω από έναν όρους που μπορεί να είναι επιπλέον σχέσεις, όπου σε αυτή την περίπτωση καλείται επιπλέον η συνάρτηση $Prove()$ για την εξέταση των νέων σχέσεων και αποθηκεύονται τα αποτελέσματα τους διασταυρωμένα στην αρχικά κενή λίστα με τα αποτελέσματα, ειδάλλως αποθηκεύονται τα αποτελέσματα της απλής σχέσης. Στο τέλος επιστρέφεται η λίστα με τα αποτελέσματα.

Η συνάρτηση $\boldsymbol{ProveNegation}(\boldsymbol{Relation\ relation})$ η οποία είναι υπεύθυνη να αποδείξει την αλήθεια σε μία αρνητική έκφραση και να βρει τις λύσεις αυτής. Στην αρχή αυτό το οποίο κάνει είναι να εξετάζει μέσω της συνάρτησης $Prove()$ την αρνητική έκφραση που συνοδεύει τον όρο **NOT** για τις λύσεις αυτού. Ένα τυπικό παράδειγμα μπορεί να είναι το παρακάτω:

$$(<= \ (GOAL\ OPLAYER\ 50)\ (NOT\ (LINE\ X))\ (NOT\ (LINE\ O))\ (NOT\ OPEN))$$

οπότε για το $(\boldsymbol{NOT}\ (\boldsymbol{LINE\ X}))$ για παράδειγμα θα πρέπει να εξεταστεί αν έχει λύσεις το $(\boldsymbol{LINE\ X})$. Αν βρεθούν λύσεις για την αρνητική έκφραση που έχουμε τότε επιστρέφεται μία κενή λίστα τύπου κλάσης $Unfier$ πίσω ειδάλλως μία λίστα ίδιου τύπου απλά με ένα αντικείμενο $Unifier$ μέσα.

Η συνάρτηση $\boldsymbol{ProveDisjunction}(\boldsymbol{Relation\ relation})$ έχει σαν στόχο να αποδείξει τη διάζευξη σε μία σχέση. Είναι απλούστερη σαν διαδικασία γενικά από τις υπόλοιπες συναρτήσεις. Αυτό που γίνεται συνοπτικά είναι να έχουμε έναν επαναληπτικό βρόγχο ο οποίος να παίρνει όλους τους όρους μίας σχέσης όπου μπορεί να είναι και επιπλέον σχέσεις να τους αποδεικνύει έναν - έναν καλώντας την συνάρτηση $Prove()$ και να προσθέτει τα αποτελέσματα καθεμίας σε μία λίστα τύπου $Unifier$. Να αναφερθεί επιπλέον ότι για να αποδειχτεί η αλήθεια σε μία διάζευξη αρκεί να ισχύει η λύση σε μία από τις σχέσεις. Ένα παράδειγμα των παραπάνω είναι το παρακάτω:

$$(\leq \ (NEXT\ (CELL\ ?M\ ?N\ B))\ (DOES\ ?W\ (MARK\ ?J\ ?K))$$

$$(TRUE\ (CELL\ ?M\ ?N\ B))\ (OR\ (DISTINCT\ ?M\ ?J)\ (DISTINCT\ \ ?N\ ?K)))$$

Που ουσιαστικά δηλώνει ότι για να είναι κενό το συγκεκριμένο κελί θα πρέπει το κελί που θα καταληφθεί δηλαδή το CELL(?J ?K ?X) να ισχύουν τα εξής: ότι το το (CELL ?M ?N B) να είναι όντως κενό εκείνη τη στιγμή κάτι το οποίο διαπιστώνεται με το κατηγόρημα TRUE και επιπλέον να ισχύει η διάζευξη, δηλαδή είτε η μεταβλητή ?M να είναι διαφορετική από την ?J σαν τιμές η $?N$ διάφορη από την $?K$ για να επιβεβαιωθεί ότι το κελί που θα καταληφθεί δεν θα είναι το $(CELL\ ?M\ ?N\ B))$ το οποίο και θα παραμείνει και κενό.

Τέλος έχουμε την συνάρτηση $\boldsymbol{ProveConjunction}(\boldsymbol{Relation\ relation})$ που έχει σαν σκοπό να αποδείξει τη σύζευξη σε ένα σύνολο από σχέσεις. Η διαδικασία που ακολουθείται είναι περίπου όμοια με προηγουμένως. Παίρνουμε έναν - έναν τους όρους από τις σχέσεις και καλούμε την συνάρτηση $Prove()$ για να πάρουμε τις λύσεις της κάθε μίας από τις σχέσεις, μόνο που σε αυτή την περίπτωση δεν μας αρκεί μόνο να ισχύει κάποια από τις τιμές που μπορεί να έχουμε σαν λύση αλλά σκοπός είναι να ισχύει μία ένωση από τιμές η οποία φυσικά εξαρτάται από το μέγεθος της σύζευξης που υπάρχει. Αυτό επιτυγχάνεται μέσω της συνάρτησης $union()$ της κλάσης $unifier$ η οποία πράττει την ένωση αυτή. Έστω ότι έχουμε το παράδειγμα με τη διαγώνιο που αναφέραμε σε προηγούμενη ενότητα, όπου έχουμε την παρακάτω σύζευξη:

$$(\leq \ (DIAGONAL\ ?X)\ (TRUE\ (CELL\ 1\ 1\ ?X))$$

$$(TRUE\ (CELL\ 2\ 2\ ?X))\ (TRUE\ (CELL\ 3\ 3\ ?X)))$$

Ουσιαστικά για να ισχύει η παραπάνω σύζευξη και να έχουμε διαγώνιο πρέπει να ισχύουν οι τρείς προτάσεις – προϋποθέσεις που ακολουθούν. Δηλαδή τα τρία διαγώνια κελιά να καταληφθούν από τον ίδιο παίχτη. Άρα αν η μεταβλητή ?X έχει παντού την ίδια τιμή τότε είναι αληθής η σχέση ειδάλλως όχι. Τέλος προσθέτουμε τις ενοποιημένες λύσεις στη λίστα τύπου $Unifier$ που έχουμε στην αρχή της συνάρτησης και την επιστρέφουμε.

3.2.5 GameState

Ουσιαστικά η κλάση αυτή είναι υπεύθυνη για τους βασικούς ελέγχους του παιχνιδιού και είναι αυτή που διαχειρίζεται πιο άμεσα τις κινήσεις των παιχτών χρησιμοποιώντας ότι έχει ειπωθεί μέχρι στιγμής. Περιέχει μία λίστα από σχέσεις στην οποία αποθηκεύονται κάθε φορά όλες οι σχέσεις που ισχύουν και έχουν πραγματοποιηθεί σε κάθε δεδομένη

στιγμή του παιχνιδιού. Επιπλέον αποτελείται από ένα σύνολο από συναρτήσεις κάθε μία από τις οποίες έχει αναλάβει κάποια συγκεκριμένη διαδικασία. Αυτές είναι οι παρακάτω:

Η συνάρτηση $\boldsymbol{Clone}()$ η οποία «αντιγράφει» ουσιαστικά την τρέχουσα κατάσταση του παιχνιδιού που έχουμε κάθε φορά για να μπορούμε να πραγματοποιούμε ότι δοκιμές χρειάζονται και απαιτούνται από τις διαδικασίες του παιχνιδιού χωρίς να επηρεάζεται η τρέχουσα κατάσταση του παιχνιδιού.

Οι συναρτήσεις $\boldsymbol{ProveTrueRelation}(\boldsymbol{Relation\ relation})$ και $\boldsymbol{InitializeTrueRelations}()$ αφορούν τη λίστα με τις σχέσεις $TrueRelations$ και έχουν σαν αντικείμενο την απόδειξη αληθείας των σχέσεων που είναι αποθηκευμένες στην πρώτη περίπτωση και την αρχικοποίηση της λίστας στην δεύτερη περίπτωση αντίστοιχα.

Η συνάρτηση $\boldsymbol{GetRoles}()$ έχει σαν αντικείμενο να πάρει από την κλάση $Knowledgebase$ κάθε κατηγόρημα με το όνομα "$role$" το οποίο ουσιαστικά κατέχει το κάθε όνομα από τους παίχτες του παιχνιδιού και να τα επιστρέφει.

Η συνάρτηση $\boldsymbol{HasNoOperations}(\boldsymbol{string\ player})$ είναι υπεύθυνη να εξετάζει για το όνομα του παίχτη που της δίνεται σαν όρισμα αν μπορεί να πραγματοποιήσει κίνηση, εν ολίγοις αν είναι η σειρά του συγκεκριμένου παίχτη στο παιχνίδι να παίξει. Αυτό γίνεται βάσει του κατηγορήματος "$legal$" η οποία αν έχει σαν όρισμα τον όρο "noop" και στην περίπτωση που είναι αληθές το κατηγόρημα αυτό τότε σημαίνει ότι ο συγκεκριμένος παίχτης δεν μπορεί να παίξει κάποια άλλη κίνηση πέρα από την «κενή κίνηση» τη δεδομένη στιγμή. Οπότε δημιουργείται μία σχέση τύπου $(legal\ player\ noop)$ στέλνεται στον αποδεικτικό αναλυτή μέσω της συνάρτηση $Prove()$ και αν επιστραφεί κάποιο αποτέλεσμα τότε σημαίνει ότι είναι αληθές και ο παίχτης δεν έχει δικαίωμα να παίξει ειδάλλως όχι.

Στην συνάρτηση $\boldsymbol{IsMoveLegal}(\boldsymbol{string\ player}, \boldsymbol{Atom\ move})$ ακολουθείται μία παρόμοια διαδικασία όπως και πριν, δηλαδή δημιουργείται μία σχέση του $τύπου\ (legal\ player\ move)$ και στέλνεται στον αποδεικτικό αναλυτή, αν ο αποδεικτικός αναλυτής επιστρέψει αποτελέσματα σημαίνει ότι ο παίχτης δικαιούται να παίξει τη συγκεκριμένη κίνηση την δεδομένη στιγμή ειδάλλως αυτή η κίνηση που επιθυμεί να πραγματοποιήσει ο παίχτης θεωρείται μη έγκυρη.

Η συνάρτηση $\boldsymbol{GetLegalMoves}(\boldsymbol{string\ player})$ χρησιμοποιείται για να συγκεντρώσει όλες τις νόμιμες κινήσεις για τον συγκεκριμένο παίχτη τη δεδομένη στιγμή. Αυτό επιτυγχάνεται με παρόμοιο τρόπο όπως και πριν, δηλαδή τη δημιουργία σχέσης που σαν τελευταία παράμετρο θα έχει μία μεταβλητή έτσι ώστε να αποθηκευτεί εκεί κάθε νόμιμη κίνηση. Η σχέση αυτή τύπου $(legal, player, var)$ στέλνεται στον αποδεικτικό αναλυτή και τα αποτελέσματα από τον αποδεικτικό αναλυτή αποθηκεύονται σε μία λίστα τύπου $Unifier$. Εκεί εξετάζεται το κάθε αντικείμενο της λίστας στην μεταβλητή καταχώρησης κίνησης var και αυτές με τη σειρά τους αποθηκεύονται σε μία λίστα τύπου $Atom$, όπου ουσιαστικά η λίστα αυτή είμαι υπεύθυνη να αποθηκεύει όλες τις νόμιμες κινήσεις του συγκεκριμένου παίχτη στο συγκεκριμένο στιγμιότυπο παιχνιδιού. Στο τέλος της συνάρτησης η λίστα αυτή επιστρέφεται.

Η συνάρτηση $\boldsymbol{IsTerminalState}()$ έχει σαν στόχο να εξετάζει αν το παιχνίδι έχει περιέλθει σε τελική κατάσταση. Οπότε στέλνεται στο αποδεικτικό αναλυτή μία σχέση με το κατηγόρημα "$terminal$" και αν έχουμε αποτελέσματα από τον αποδεικτικό αναλυτή αυτό σημαίνει ότι το παιχνίδι έχει τερματίσει ειδάλλως ότι συνεχίζεται κανονικά.

Η συνάρτηση $\boldsymbol{GetGoalValue}(\boldsymbol{string\ player})$ είναι υπεύθυνη για την αξιολόγηση του δοθέντος παίχτη. Δηλαδή ουσιαστικά επιστρέφει τη βαθμολογία του συγκεκριμένου

παίχτη. Η διαδικασία που ακολουθεί είναι απλή. Στέλνεται στον αποδεικτικό αναλυτή μία κατασκευασμένη σχέση με βάση το κατηγόρημα “goal”, δηλαδή του $τύπου\ (goal\ player\ ?\ goal)$ και στην μεταβλητή goal επιστρέφεται η αξιολόγηση του συγκεκριμένου παίχτη ανάλογα με το τι έχει επιτύχει στο παιχνίδι πάνω στους κανόνες του παιχνιδιού. Στο τέλος της συνάρτησης η βαθμολογία του παίχτη επιστρέφεται. Ένα παράδειγμα βαθμολογίας για το παιχνίδι «Τρίλιζα» από τους κανόνες του παιχνιδιού είναι το παρακάτω, όπου δείχνει τη βαθμολογία του παίχτη X σε κάθε μια από τις τρείς περιπτώσεις ανάλογα με το ποια θα είναι αληθής:

$$(<=\ (GOAL\ XPLAYER\ 100)\ (LINE\ X))$$

$$(<=\ (GOAL\ XPLAYER\ 50)\ (NOT\ (LINE\ X))\ (NOT\ (LINE\ O))\ (NOT\ OPEN))$$

$$(<=\ (GOAL\ XPLAYER\ 0)\ (LINE\ O))$$

Η συνάρτηση $\boldsymbol{PlayMove(string\ player, Atom\ move)}$ πραγματοποιεί ουσιαστικά αυτό που λέει το όνομα της, δηλαδή παίζει τη συγκεκριμένη κίνηση για τον δοθέντα παίχτη. Αυτό επιτυγχάνεται δημιουργώντας μια σχέση βασισμένη στο κατηγόρημα “$does$”. Δηλαδή μία σχέση του τύπου $(does\ player\ move)$ η οποία προστίθεται στη λίστα με τα $TrueRelations$ η οποία ουσιαστικά περιλαμβάνει όλες τις κινήσεις του παιχνιδιού που ισχύουν μέχρι στιγμής.

Τέλος είναι η συνάρτηση $\boldsymbol{GetNextState()}$ η οποία είναι υπεύθυνη να πραγματοποιήσει τις αλλαγές στο παιχνίδι όπου θα φέρουν το παιχνίδι σε νέα κατάσταση. Οπότε ανάλογα με το τι κινήσεις έχουν πραγματοποιηθεί μέχρι τη δεδομένη στιγμή στο παιχνίδι δημιουργείτε μία σχέση με βάση το κατηγόρημα “$next$” και μία μεταβλητή και στέλνεται στον αποδεικτικό αναλυτή. Έπειτα ανάλογα με τα αποτελέσματα του αποδεικτικού αναλυτή και την μεταβλητή σε καθένα από αυτά δημιουργούνται σχέσεις με βάση το κατηγόρημα “true” τα οποία και αποθηκεύονται στη λίστα $TrueRelations$ της κλάσης. Στο τέλος επιστρέφεται η νέα κατάσταση του παιχνιδιού.

3.3 Αξιολόγηση

Όπως έχει αναφερθεί και σε προηγούμενα κεφάλαια, η εξαντλητική αναζήτηση του δέντρου καταστάσεων, προκειμένου να βρεθεί η καλύτερη κίνηση, είναι πρακτικά αδύνατη, γιατί το δέντρο καταστάσεων έχει τις περισσότερες φορές μεγάλο παράγοντα διακλάδωσης, κάτι που καθιστά την αναζήτηση, ακόμα και σε όχι μεγάλα βάθη, χρονοβόρα. Αυτό σημαίνει ότι πρέπει να υπάρχει κάποιου είδους ενδιάμεση αξιολόγηση προκειμένου η αναζήτηση να γίνεται μέχρι ένα συγκεκριμένο βάθος και να αξιολογούνται οι καταστάσεις που βρίσκονται στο βάθος – όριο.

Η υλοποίηση ενδιάμεσης αξιολόγησης είναι απλή υπόθεση για προγράμματα που σχετίζονται με συγκεκριμένα παιχνίδια μόνο, αλλά για έναν *General Game Player* αποτελεί δύσκολη και πολύπλοκη υπόθεση μιας και οφείλει να παίζει οποιοδήποτε παιχνίδι. Για το λόγο αυτό, η ενδιάμεση αξιολόγηση σε έναν *General Game Player* συνήθως βασίζεται σε ασαφή λογική σε ευριστικές που προκύπτουν από χαρακτηριστικά τα οποία ανακαλύπτονται έπειτα από τη λήψη πειραμάτων.

Το SweetGGP χρησιμοποιεί τη δομή που παρουσιάζεται στο κεφάλαιο 2.4 και βασίζεται εκτενώς σε ασαφή λογική για την αξιολόγηση του πόσο κοντά βρίσκεται κάθε παίκτης σε κάθε στόχο, σε μια κατάσταση του παιχνιδιού.

Η αξιολόγηση στο SweetGGP βασίζεται στην εξής ιδέα: Κάθε παίκτης προσπαθεί να φτάσει στον πιο κερδοφόρο στόχο του, αλλά αν δε τα καταφέρει προσπαθεί τουλάχιστον να φτάσει στους υπόλοιπους λιγότερο κερδοφόρους στόχους, αποφεύγοντας όμως τα ταυτόχρονα τους μη κερδοφόρους στόχους. Για παράδειγμα, αν

ένας παίκτης μπορεί να φτάσει σε στόχους με κέρδη 100, 75, 50, 25 και 0, τότε πρώτα απ' όλα θέλει να φτάσει στο στόχο με κέρδος 100 και αν δε τα καταφέρει, να φτάσει τουλάχιστον στο στόχο με κέρδος 75. Με τίποτα όμως δε θέλει να φτάσει στους στόχους με κέρδη 25 και 0.

Αυτή η διαδικασία υλοποιείται ως εξής:

1. Βρίσκεται το μέγιστο και το ελάχιστο κέρδος.
2. Υπολογίζεται ο μέσος όρος τους.
3. Το συνολικό κέρδος αρχικοποιείται στο 0.
4. Για κάθε στόχο:
    4.1. Γίνεται η αξιολόγηση του στόχου.
    4.2. Προστίθεται στο συνολικό κέρδος η αξιολόγηση του στόχου με βάρος ίσο με το κέρδος του στόχου μείον το μέσο όρο (έτσι οι μη κερδοφόροι στόχοι έχουν αρνητική επίπτωση).
5. Επιστρέφεται το συνολικό κέρδος ως αποτέλεσμα.

Η διαδικασία αξιολόγησης ενός στόχου ακολουθεί τον αλγόριθμο που παρουσιάστηκε στο κεφάλαιο 2.4:

```
1 eval(not(Child), S, V) :-
2   eval(Child, S, V1), not(V1, V).
3 eval(true(Fluent), S, V) :-
4   fluent_holds(Fluent, S),
5   !, fz_true(V).
6 eval(true(_Fluent), _, V) :- fz_false(V).
7 eval(and([]), _, V) :- !, true(V).
8 eval(and([C|Children]), S, V) :-
9   eval(C, S, V1),
10  eval(and(Children), S, V2),
11  fz_and(V1, V2, V).
12 eval(or([]), _, V) :- !, false(V).
13 eval(or([C|Children]), S, V) :-
14  eval(C, S, V1),
15  eval(or(Children), S, V2),
16  fz_or(V1, V2, V).
17 eval(reason(Expr), S, V) :-
18  expression_holds(Expr, S),
19  !, fz_true(V).
20 eval(reason(_Expr), _, V) :- fz_false(V).
```

Εικόνα 25: Αλγόριθμος αξιολόγησης δεντρικής δομής εκφράσεων

Χρησιμοποιώντας τις παραστάσεις ασαφούς λογικής, που επίσης παρουσιάζονται στο ίδιο κεφάλαιο:

```
1 fz_true(0.9). true(1).
2 fz_false(0.1). false(0).
3 not(X, Y) :- Y is 1 - X.
4 fz_and(X, Y, Z) :- Z is Y * X.
5 fz_or(X, Y, Z) :- Z is X + Y - X * Y.
```

Εικόνα 26: Παραστάσεις ασαφούς λογικής

Όσον αφορά τις συζεύξεις, χρησιμοποιείται αναζήτηση κατά βάθος (Depth First Search, DFS) για να βρεθούν οι εκφράσεις της σύζευξης που μοιράζονται κοινές μεταβλητές. Για κάθε τέτοια ομάδα εκφράσεων, όπου υπάρχουν τουλάχιστον δύο εκφράσεις που μοιράζονται κοινές μεταβλητές, τότε χρησιμοποιείται ο prover για να ελεγχθεί αν έχει λύση η ομάδα. Αν έχει λύση η ομάδα τότε επιστρέφεται η τιμή της παράστασης fz_true, αλλιώς επιστρέφεται η τιμή της παράστασης fz_false. Η διαδικασία αυτή απεικονίζεται στον παραπάνω αλγόριθμο στη γραμμή 17, με την έκφραση $\boldsymbol{eval(reason(Expr), S, V)}$.

3.4 Αναζήτηση

Η αναζήτηση καλύτερης κίνησης σε έναν *General Game Player* είναι διαφορετική, λόγω του ότι πρέπει να είναι σε θέση να δεχτεί παιχνίδια οσονδήποτε παικτών, από έναν έως άπειρους παίκτες. Το SweetGGP χρησιμοποιεί μια παραλλαγή των αλγορίθμων αναζήτησης $minimax$ και άλφα – βήτα κλαδέματος. Στη συγκεκριμένη παραλλαγή, οι παραπάνω αλγόριθμοι προσαρμόζονται ώστε να λειτουργούν για οποιονδήποτε αριθμό παικτών. Στη συνέχεια ακολουθεί περιγραφή των αλγορίθμων $minimax$ και άλφα – βήτα κλαδέματος και ύστερα περιγραφή της παραλλαγής που χρησιμοποιείται στο SweetGGP.

3.4.1 Αλγόριθμος Minimax

Ο αλγόριθμος $minimax$ [10] είναι ένας από τους κυριότερους αλγορίθμους που χρησιμοποιούνται στη θεωρία παιγνίων και η ουσιαστική διαφορά του από τους υπόλοιπους αλγορίθμους αναζήτησης είναι ότι λαμβάνει υπόψη του τον αριθμό των παικτών σε ένα παιχνίδι. Αν πάρουμε σαν παράδειγμα το παιχνίδι «Σκάκι», έστω μια τιμή V η οποία αποτελεί τη μέγιστη τιμή κέρδους, τότε η μέγιστη απολαβή για τον παίκτη που αναλαμβάνει η σκακιστική μηχανή είναι V, ενώ για τον άλλο παίκτη είναι –V. Θα μπορούσαμε να πούμε πως στην τελική κατάσταση στην οποία κάνει ματ ο λευκός παίκτης έχουμε κέρδος 1, στην τελική κατάσταση που κάνει ματ ο μαύρος παίκτης έχουμε κέρδος -1 και στην τελική κατάσταση στην οποία έχουμε ισοπαλία έχουμε κέρδος 0.

Ο λόγος που ονομάστηκε $minimax$ είναι γιατί σκοπός του είναι, για ένα παίκτη, να μειώνει τις πιθανές απώλειες και να αυξάνει τα κέρδη. Για παράδειγμα, στο «Σκάκι», αν υποθέσουμε πως η σκακιστική μηχανή αναλαμβάνει το ρόλο του μαύρου παίκτη, τότε ο αλγόριθμος έχει ως σκοπό να επιλέγει κινήσεις που, για παράδειγμα, μειώνουν την πιθανότητα ο μαύρος παίκτης να χάσει δυνατά κομμάτια και αυξάνουν τις δυνατότητες του μαύρου παίκτη να κάνει ματ στον λευκό παίκτη.

Σε κάθε περίπτωση, ο αλγόριθμος $minimax$ κοιτάει αρκετές κινήσεις μπροστά και προσπαθεί να μεγιστοποιήσει το παραπάνω κέρδος για τον παίκτη που έχει αναλάβει.

```
chSCORE minimax(chPOSITION node) {
      if (isLeaf(node) ) return evaluate(node);
      else if (isMaxOnMove(node) ) {
            g = -maxSCORE;   // maxSCORE is a huge positive
            //number
            x = left_most_child(node);
            while (isNotNull (x) ) {
                  g = max(g, minimax(x) );
                  x = next_brother(x);
            }
      }
      else {          // MIN on the move
            g = +maxSCORE;
            x = left_most_child(node);
            while (isNotNull (x) ) {
                  g = min(g, minimax(x) );
                  x = next_brother(x);
            }
      }
      return g;
} // end of minimax
```

Εικόνα 27: Ο αλγόριθμος minimax

Ο παίκτης του οποίου προσπαθούμε να μεγιστοποιήσουμε το κέρδος ονομάζεται max και ο παίκτης του οποίου προσπαθούμε να ελαχιστοποιήσουμε το κέρδος ονομάζεται min. Για την καλύτερη κατανόηση του ψευδοκώδικα ας θεωρήσουμε το παρακάτω παράδειγμα: Έστω ότι ο max είναι ο λευκός παίκτης και ο min είναι ο μαύρος παίκτης και είναι η σειρά του λευκού παίκτη να παίξει. Αν είμαστε σε τελική κατάσταση (ματ ή ισοπαλία) τότε το κέρδος που έχει ο λευκός παίκτης είναι 1 αν κάνει ματ, -1 αν κάνει ο μαύρος παίκτης ματ και 0 αν είναι ισοπαλία, αλλιώς πρέπει για κάθε πιθανή κίνηση που μπορεί να κάνει ο λευκός παίκτης να υπολογίσει το κέρδος τους και να διαλέξει την κίνηση που θα του αποφέρει το μέγιστο κέρδος, οπότε εφαρμόζει την ίδια μέθοδο σε όλες τις πιθανές κινήσεις. Αν κάποια κατάσταση είναι τελική τότε επιστρέφει το κατάλληλο σκορ, αλλιώς, είναι η σειρά του μαύρου παίκτη να παίξει οπότε εξετάζει όλες τις πιθανές κινήσεις και διαλέγει εκείνη που ελαχιστοποιεί το κέρδος. Αυτή η διαδικασία συνεχίζεται μέχρι να βρεθεί μια τελική κατάσταση ή να φτάσει η διαδικασία σε κάποιο βάθος – όριο, όπου και εφαρμόζει αξιολόγηση της κατάστασης.

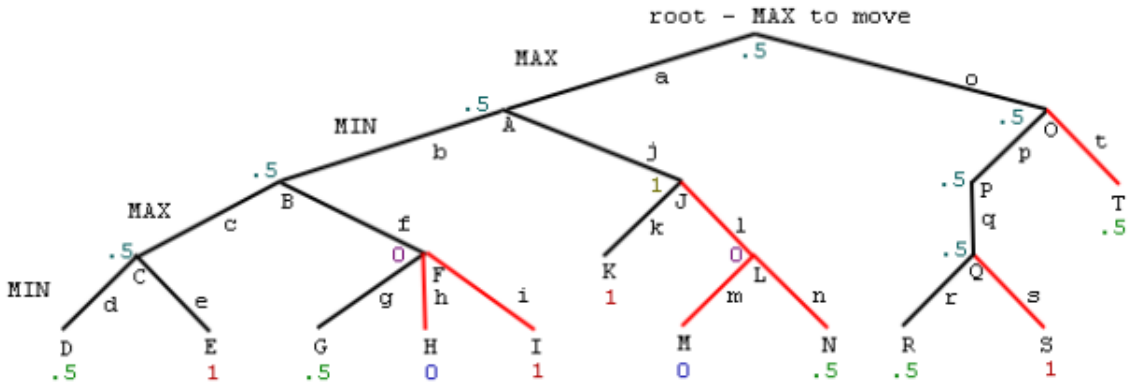
3.4.2 Άλφα – Βήτα Κλάδεμα (Alpha – Beta Pruning)

Το κλάδεμα άλφα-βήτα [10] αποτελεί προσθήκη του αλγορίθμου $minimax$ και σκοπός του είναι να κόβει κόμβους που δε χρειάζεται να εξετάσουμε γιατί το κέρδος που μας προσφέρουν μας είναι αδιάφορο. Πιο συγκεκριμένα, δεν υπάρχει λόγος να ελεγχθεί για τον max παίκτη ένας κόμβος που του δίνει κέρδος 0.5 από τη στιγμή που έχει ήδη βρεθεί κόμβος ο οποίος του δίνει κέρδος 1.

```
chSCORE alphabeta(chPOSITION node, chSCORE alpha, beta) {
     if (isLeaf(node) ) return evaluate(node);
     else if (isMaxOnMove(node) ) {
          g = -maxSCORE;  // maxSCORE is a huge
          //positive number
          x = left_most_child(node);
          while ( (g < beta) && (isNotNull (x) ) ) {
               g = max(g, alphabeta(x, alpha,beta) );
               alpha = max (alpha, g)
               x = next_brother(x);
          }
     }
       else {          // MIN on the move
          g = +maxSCORE;
          x = left_most_child(node);
          while ( (g > alpha) && (isNotNull (x) ) ) {
               g = min(g, alphabeta(x, a, b) );
               beta = min (beta, g);
               x = next_brother(x);
          }
       }
       return g;
} // end of alphabeta
```

Εικόνα 28: Ο αλγόριθμος κλαδέματος άλφα – βήτα

Ο αλγόριθμος μπορεί να ξεκινάει είτε με άλφα = -∞ και βήτα = +∞ είτε με άλφα = ελάχιστο κέρδος και βήτα = μέγιστο κέρδος (π.χ άλφα = -1, βήτα = 1).



Εικόνα 29: Παράδειγμα εφαρμογής κλαδέματος άλφα – βήτα

Για την καλύτερη κατανόηση της μεθόδου ας δούμε ένα παράδειγμα βασισμένο στο παραπάνω δέντρο. Έστω ότι ελέγχεται ο κόμβος F, μέχρι στιγμής έχουμε ότι άλφα = 0.5

και βήτα = 0.5. Αν χρησιμοποιούσαμε μόνο τον αλγόριθμο $minimax$ τότε θα ελέγχαμε τους κόμβους G, H, I, αλλά χρησιμοποιώντας το κλάδεμα άλφα-βήτα ελέγχουμε μόνο τον κόμβο G, γιατί δεν υπάρχει λόγος να ελέγξουμε τον κόμβο H, αφού αποτελεί τελική κατάσταση στην οποία ο max παίκτης χάνει ενώ έχει σίγουρη καλύτερη θέση (άλφα = 0.5) και δεν υπάρχει λόγος να ελέγξουμε τον κόμβο I αφού αποτελεί τελική κατάσταση στην οποία νικάει ο max παίκτης οπότε δεν υπάρχει περίπτωση ο min παίκτης να επιλέξει αυτή την κίνηση.

Η απόδοση της μεθόδου εξαρτάται άμεσα από την ταξινόμηση των πιθανών κινήσεων. Αν οι καλύτερες, από θέμα κέρδους, κινήσεις ελέγχονται πρώτες τότε όλο και περισσότεροι κόμβοι θα αποκοπούν, ενώ στην αντίθετη περίπτωση, όπου πρώτα ελέγχονται οι χειρότερες, από θέμα κέρδους, κινήσεις, τότε το κλάδεμα άλφα-βήτα τείνει να λειτουργεί ακριβώς όπως ο αλγόριθμος $minimax$.

3.4.3 Ο αλγόριθμος αναζήτησης του SweetGGP

Στην υλοποίηση του SweetGGP έπρεπε να ληφθούν υπ' όψη οι εξής απαιτήσεις:

- Σε ένα γύρο, για κάποιο παιχνίδι, ενδέχεται να έχουν τη δυνατότητα να παίξουν παραπάνω από ένας παίκτης.
- Κάθε παιχνίδι έχει απροσδιόριστο αριθμό παικτών.

Ακολουθώντας πιστά τις αρχές των αλγορίθμων $minimax$ και κλαδέματος άλφα – βήτα, η αναζήτηση στο SweetGGP θα έπρεπε, σε κάθε επίπεδο του δέντρου καταστάσεων, να ψάχνει την καλύτερη κίνηση για διαφορετικό παίκτη κάθε φορά. Μια τέτοια τροποποίηση για τους συγκεκριμένους αλγορίθμους παρουσιάζεται στο [12]. Προκειμένου να απλουστευτεί η διαδικασία, στο SweetGGP χρησιμοποιείται το εξής αξίωμα: Για έναν παίκτη, η χειρότερη περίπτωση δεν είναι να παίζουν οι υπόλοιποι παίκτες βέλτιστα ως προς τον εαυτό τους, αλλά να παίζουν κινήσεις οι οποίες τον φέρνουν στη χειρότερη θέση. Έτσι, σε κάθε επίπεδο του δέντρου καταστάσεων, πρέπει να επιλεγεί η καλύτερη κίνηση για τον παίκτη, θεωρώντας ότι οι υπόλοιποι παίκτες θα παίξουν τις χειρότερες, για τον ίδιο, κινήσεις. Επίσης, θεωρείται ότι σε κάθε γύρο μπορεί να κάνει κίνηση κάθε παίκτης, ακόμα κι αν αυτή η κίνηση είναι η $noop$.

Η διαδικασία που ακολουθεί το SweetGGP για την αναζήτηση της καλύτερης κίνησης είναι η εξής:

Αναζήτηση(κατάσταση, βάθος, α, β):

1. Αν η κατάσταση είναι τερματική
    1.1. Επιστρέφεται η τιμή στόχου που αντιστοιχεί στον παίκτη για τη συγκεκριμένη κατάσταση.
2. Αν το βάθος ισούται με το βάθος – όριο:
    2.1. Γίνεται αξιολόγηση της κατάστασης.
    2.2. Επιστρέφεται η τιμή της αξιολόγησης.
3. bestMove := NULL.
4. Για κάθε κίνηση του παίκτη στη συγκεκριμένη κατάσταση:
    4.1. minScore := β.
    4.2. Για κάθε συνδυασμό κινήσεων των υπολοίπων παικτών:
        4.2.1. Παίζεται η κίνηση του παίκτη
        4.2.2. Για κάθε κίνηση στο συνδυασμό των κινήσεων των υπολοίπων παικτών που εξετάζεται:
            4.2.2.1. Παίζεται η κίνηση του συνδυασμού.
        4.2.3. Υπολογίζεται η επόμενη κατάσταση του παιχνιδιού (κατάσταση') με βάση τις κινήσεις που παίχτηκαν.

4.2.4. score := Αναζήτηση(κατάσταση', βάθος + 1, α, minScore).
4.2.5. Αν score < minScore:
4.2.5.1. minScore := score.
4.2.6. Αν temp ≤ α:
4.2.6.1. Σταματάει η επανάληψη.
4.3. Αν minScore > α:
4.3.1. α = minScore.
4.3.2. Στο bestMove ανατίθεται η κίνηση του παίκτη που εξετάζεται.
4.4. Αν α ≥ β:
4.4.1. Σταματάει η επανάληψη.
5. Επιστρέφεται το bestMove με σκορ α.

4. ΕΠΕΚΤΑΣΕΙΣ ΣΤΗ ΓΛΩΣΣΑ GDL

Τα περισσότερα παιχνίδια τα οποία χρησιμοποιήθηκαν στα πλαίσια αυτής της διπλωματικής εργασίας, είναι παιχνίδια τα οποία παίζονται πάνω σε ταμπλό. Επίσης, ένα τεράστιο ποσοστό παιχνιδιών παίζονται πάνω σε ταμπλό, το οποίο έχει κελιά. Παρατηρήθηκε ότι αυτά τα παιχνίδια, πολλές φορές, μοιράζονται κοινές εντολές και γενικά κοινές τακτικές, στην περιγραφή τους σε GDL. Ακόμα, πολλά τμήματα της περιγραφής αυτών των παιχνιδιών βασίζεται στην επαναλαμβανόμενη χρήση ίδιων εντολών, για παράδειγμα, στο σκάκι, για να περιγραφεί το ταμπλό, που έχει 64 τετράγωνα, χρειάζονται 64 εντολές της μορφής $init(cell(x, y, z))$ για να περιγραφούν οι θέσεις των κομματιών, καθώς και περίπου 200 εντολές της μορφής $succ\ x\ y$, για τον ορισμό των διαδόχων αριθμών. Το μεγάλο, αυτό, πλήθος ίδιων εντολών δυσκολεύει την περιγραφή των παιχνιδιών και οδηγεί με μεγαλύτερη πιθανότητα σε λάθη.

Στα πλαίσια αυτής της διπλωματικής εργασίας, αναπτύχθηκαν ορισμένες εντολές, ως επεκτάσεις στη γλώσσα GDL, που βοηθούν στην περιγραφή παιχνιδιών που παίζονται σε ταμπλό. Παρατηρήθηκε ότι η χρήση αυτών των επεκτάσεων είχε ως αποτέλεσμα, σε παιχνίδια με μεγάλες περιγραφές, όπως η ντάμα και το σκάκι, ο κώδικας του παιχνιδιού να μειωθεί στα 2/3 του περίπου. Στη συνέχεια ακολουθεί περιγραφή των εντολών που αναπτύχθηκαν, καθώς και παρουσίαση παιχνιδιού που χρησιμοποιεί τις επεκτάσεις, ως παράδειγμα.

4.1 Περιγραφή των επεκτάσεων

Σε αυτό το κεφάλαιο γίνεται περιγραφή των εντολών, που υλοποιήθηκαν ως επεκτάσεις στη γλώσσα GDL από το γενικό παίκτη SweetGGP.

4.1.1 Η εντολή board

Η εντολή $board$ είναι η σημαντικότερη εντολή των επεκτάσεων. Ο τρόπος σύνταξης της είναι ο εξής:

$$\boldsymbol{board}(\boldsymbol{κελί_1}, \boldsymbol{κελί_2}, \boldsymbol{κελί_3}, \dots, \boldsymbol{nextline}, \dots \boldsymbol{κελί_N}, \boldsymbol{κελί_{N+1}}, \dots, \boldsymbol{nextline}, \dots).$$

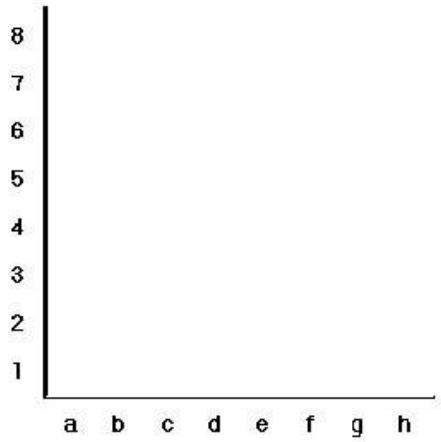
Σαν ορίσματα δέχεται το περιεχόμενο κάθε κελιού και την ειδική σταθερά **nextline** η οποία δηλώνει ότι τα επόμενα περιεχομένα κελιών, που δίνονται σαν ορίσματα, θα τοποθετηθούν στην επόμενη γραμμή του ταμπλό. Στην ουσία περιγράφει την κατάσταση του ταμπλό κατά την έναρξη του παιχνιδιού. Η εντολή board λειτουργεί ως εξής: Για κάθε όρισμα, αν το όρισμα είναι η ειδική σταθερά **nextline**, τότε ο δείκτης της τρέχουσας γραμμής αυξάνεται και ο δείκτης της τρέχουσας στήλης μηδενίζεται. Σε διαφορετική περίπτωση, αυξάνεται ο δείκτης της τρέχουσας στήλης και τοποθετείται το όρισμα στο κελί που δηλώνεται απο το δείκτη της τρέχουσας γραμμής και το δείκτη της τρέχουσας στήλης. Τα ορίσματα τοποθετούνται ως περιεχόμενα στα κελιά μέσω κατάλληλων εντολών **init**. Έστω η παρακάτω δήλωση της εντολής board:

$$\boldsymbol{board}(\boldsymbol{bp}, \boldsymbol{bp}, \boldsymbol{bp}, \boldsymbol{bp}, \boldsymbol{nextline}, \boldsymbol{b}, \boldsymbol{b}, \boldsymbol{b}, \boldsymbol{b}, \boldsymbol{nextline}, \boldsymbol{wp}, \boldsymbol{wp}, \boldsymbol{wp}, \boldsymbol{wp}).$$

Αυτό σημαίνει πως το παιχνίδι περιέχει ένα ταμπλό 3x4 και οι εντολές $\boldsymbol{init}$ που εισάγονται είναι οι εξής:

$$init(cell(a, 1, wp)), init(cell(b, 1, wp)), init(cell(c, 1, wp)), init(cell(d, 1, wp)),$$
$$init(cell(a, 2, b)), init(cell(b, 2, b)), init(cell(c, 2, b)), init(cell(d, 2, b)),$$
$$init(cell(a, 3, bp)), init(cell(b, 3, bp)), init(cell(c, 3, bp)), init(cell(d, 3, bp))$$

Οι στήλες του αρχείου χωρίζονται με κενά. Όπως παρατηρείται, η αρίθμηση των γραμμών και των στηλών ακολουθεί την εξής λογική:



Εικόνα 30: Αρίθμηση γραμμών και στηλών στην εντολή board

Οι γραμμές αριθμούνται από το 1 ως το άπειρο, ξεκινώντας από κάτω προς τα πάνω, και οι στήλες αριθμούνται από το a ως το άπειρο $(a, b, c, \dots, z, a2, b2, c2, \dots, z2, a3, \dots)$, ξεκινώντας από αριστερά προς τα δεξιά.

Μαζί με τις εντολές $init$, εισάγονται και κάποιες ακόμα εντολές, που συντελούν στην περιγραφή των συντεταγμένων του ταμπλό. Οι εντολές αυτές είναι οι εξής:

- $\boldsymbol{coordinate}$: Περιγράφει ποιες σταθερές συμβολίζουν συντεταγμένες του ταμπλό, για παράδειγμα: $coordinate(a), coordinate(b), coordinate(1)$, κτλ.
- $\boldsymbol{next\_file}$: Περιγράφει ποια στήλη ακολουθεί μετά από μια άλλη στήλη, για παράδειγμα: $next\_file(a, b), next\_file(b, c), next\_file(c, d)$, κτλ.
- $\boldsymbol{next\_rank}$: Περιγράφει ποια γραμμή ακολουθεί μετά από μια άλλη γραμμή, για παράδειγμα: $next\_rank(1, 2), next\_rank(2, 3), next\_rank(3, 4)$, κτλ.

Ανάλογα με το πόσες γραμμές και στήλες έχει το ταμπλό, εισάγονται στη βάση γνώσης οι κατάλληλες εντολές $coordinate, next\_file$ και $next\_rank$. Για παράδειγμα, για το παραπάνω ταμπλό:

$$bp \;\; bp \;\; bp \;\; bp$$

$$b \;\; b \;\; b \;\; b$$

$$wp \;\; wp \;\; wp \;\; wp$$

εισάγονται στη βάση γνώσης οι εξής εντολές:

$$coordinate(a), coordinate(b), coordinate(c), coordinate(d),$$

$$coordinate(1), coordinate(2), coordinate(3),$$

$$next\_file(a, b), next\_file(b, c), next\_file(c, d),$$

$$next\_rank(1, 2), next\_rank(2, 3)$$

4.1.2 Η εντολή succ

Η εντολή succ συντάσσεται ως εξής:

$$\boldsymbol{succ(x, y).}$$

Η εντολή $succ$ υπολογίζει ποιος αριθμός είναι ο διάδοχος (επόμενος) κάποιου άλλου αριθμού, για παράδειγμα οι εξής εντολές είναι αληθείς: $succ(1,2)$, $succ(2,3)$, $succ(3,4)$, κτλ. Η εντολή $succ$ μπορεί να συνταχτεί έχοντας το πολύ μία μεταβλητή (π.χ. $succ(X,4)$, $succ(3,X)$), αλλά αν συνταχτεί με δύο μεταβλητές, τότε δεν επιστρέφεται αληθές αποτέλεσμα.

4.1.3 Η εντολή succlimit

Η εντολή $succlimit$ συντάσσεται ως εξής:

$$\boldsymbol{succlimit}(\boldsymbol{x},\boldsymbol{y}).$$

Η εντολή $succlimit$ βάζει όρια στην εντολή $succ$. Το πρώτο όρισμα της $succlimit$ καθορίζει το κάτω όριο της $succ$ και το δεύτερο όρισμα της $succlimit$ καθορίζει το άνω όριο της $succ$. Για παράδειγμα, με τη $succlimit(1,10)$, οι εξής εντολές είναι ψευδείς: $succ(0,1)$, $succ(11,12)$.

4.1.4 Η εντολή sum

Η εντολή sum συντάσσεται ως εξής:

$$\boldsymbol{sum}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}).$$

Η εντολή sum κάνει πρόσθεση ακεραίων αριθμών (π.χ.: $sum(1,2,3)$, $sum(4,5,9)$, κτλ.). Η εντολή sum μπορεί να συνταχτεί έχοντας το πολύ μία μεταβλητή (π.χ.: $sum(1,2,X)$, $sum\ (1,X,5)$, κτλ.). Σε αντίθετη περίπτωση δεν επιστρέφεται αληθές αποτέλεσμα.

4.1.5 Η εντολή minus

Η εντολή $minus$ συντάσσεται ως εξής:

$$\boldsymbol{minus}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}).$$

Η εντολή $minus$ κάνει αφαίρεση ακεραίων αριθμών (π.χ.: $minus(5,2,3)$, $minus(14,5,9)$, κτλ.). Η εντολή $minus$ μπορεί να συνταχτεί έχοντας το πολύ μία μεταβλητή (π.χ.: $minus(3,2,X)$, $minus\ (20,X,5)$, κτλ.). Σε αντίθετη περίπτωση δεν επιστρέφεται αληθές αποτέλεσμα.

4.1.6 Η εντολή different_cells

Η εντολή $different\_cells$ συντάσσεται ως εξής:

$$\boldsymbol{different\_cells}(\boldsymbol{x1},\boldsymbol{y1},\boldsymbol{x2},\boldsymbol{y2}).$$

Η εντολή $different\_cells$ επιστρέφει ψευδές αποτέλεσμα αν $x1 = x2$ και $y1 = y2$, ενώ επιστρέφει αληθές αποτέλεσμα σε αντίθετη περίπτωση. Παραδείγματα χρήσης: $different\_cells(a,1,b,2)$, $different\_cells(X,Y,d,5)$, κτλ. Η εντολή $different\_cells$ είναι παρόμοια με την εντολή $distinct$.

4.2 Παράδειγμα παιχνιδιού

Στη συνέχεια παρουσιάζεται η περιγραφή ενός παιχνιδιού, που είναι παραλλαγή του σκακιού με μικρότερο ταμπλό (4x4), όπου ο λευκός παίκτης έχει βασιλιά και πύργο, ενώ

ο μαύρος παίκτης έχει μόνο βασιλιά. Ο κώδικας που ακολουθεί βασίζεται σε κώδικα του πανεπιστημίου του Stanford.

Ο κώδικας του παιχνιδιού:

$(role\ white)$

$(role\ black)$

$(board\ bk\ b\ b\ b\ nextline\ b\ b\ b\ b\ nextline\ b\ b\ b\ b\ nextline\ b\ b\ wk\ wr)$

$(init\ (control\ white))$

$(init\ (step\ 1))$

$(<=\ (next\ (cell\ ?x\ ?y\ ?p))$

$(does\ ?player\ (move\ ?p\ ?u\ ?v\ ?x\ ?y)))$

$(<=\ (next\ (cell\ ?u\ ?v\ b))$

$(does\ ?player\ (move\ ?p\ ?u\ ?v\ ?x\ ?y)))$

$(<=\ (next\ (cell\ ?w\ ?z\ b))$

$(does\ ?player\ (move\ ?p\ ?u\ ?v\ ?x\ ?y))$

$(true\ (cell\ ?w\ ?z\ b))$

$(or\ (distinct\ ?w\ ?x)\ (distinct\ ?z\ ?y)))$

$(<=\ (next\ (cell\ ?w\ ?z\ ?q))$

$(does\ ?player\ (move\ ?p\ ?u\ ?v\ ?x\ ?y))$

$(true\ (cell\ ?w\ ?z\ ?q))$

$(distinct\ ?p\ ?q)$

$(distinct\ ?q\ b))$

$(<=\ (next\ (control\ white))$

$(true\ (control\ black)))$

$(<=\ (next\ (control\ black))$

$(true\ (control\ white)))$

$(<=\ (next\ (step\ ?y))$

$(true\ (step\ ?x))$

$(succ\ ?x\ ?y))$

$(<=\ (legal\ white\ (move\ wk\ ?u\ ?v\ ?x\ ?y))$

$(true\ (control\ white))$

$(true\ (cell\ ?u\ ?v\ wk))$

$(kingmove\ ?u\ ?v\ ?x\ ?y)$

$(true\ (cell\ ?x\ ?y\ b))$

$(not\ (restricted\ ?x\ ?y)))$

$(<=\ (legal\ white\ (move\ wr\ ?u\ ?v\ ?x\ ?y))$

$(true\ (control\ white))$

$(true\ (cell\ ?u\ ?v\ wr))$

$(rookmove\ ?u\ ?v\ ?x\ ?y)$

$(true\ (cell\ ?x\ ?y\ b))$

$(not\ (restricted\ ?x\ ?y)))$

$(<=\ (legal\ white\ noop)$

$(true\ (control\ black)))$

$(<=\ (legal\ black\ (move\ bk\ ?u\ ?v\ ?x\ ?y))$

$(true\ (control\ black))$

$(true\ (cell\ ?u\ ?v\ bk))$

$(kingmove\ ?u\ ?v\ ?x\ ?y)$

$(true\ (cell\ ?x\ ?y\ b))$

$(not\ (attacked\ bk\ ?x\ ?y))$

$(not\ (guarded\ ?x\ ?y)))$

$(<=\ (legal\ black\ noop)$

$(true\ (control\ white)))$

$(<=\ (kingmove\ ?u\ ?v\ ?u\ ?y)$

$(or\ (next\_rank\ ?v\ ?y)\ (next\_rank\ ?y\ ?v))$

$(coordinate\ ?u))$

$(<= \ (kingmove\ ?u\ ?v\ ?x\ ?v)$

$(or\ (next\_file\ ?u\ ?x)\ (next\_file\ ?x\ ?u))$

$(coordinate\ ?v))$

$(<= \ (kingmove\ ?u\ ?v\ ?x\ ?y)$

$(or\ (next\_file\ ?u\ ?x)\ (next\_file\ ?x\ ?u))$

$(or\ (next\_rank\ ?v\ ?y)\ (next\_rank\ ?y\ ?v)))$

$(<= \ (rookmove\ ?u\ ?v\ ?u\ ?y)$

$(clearcolumn\ wr\ ?u\ ?v\ ?y))$

$(<= \ (rookmove\ ?u\ ?v\ ?u\ ?y)$

$(clearcolumn\ wr\ ?u\ ?y\ ?v))$

$(<= \ (rookmove\ ?u\ ?v\ ?x\ ?v)$

$(clearrow\ wr\ ?u\ ?x\ ?v))$

$(<= \ (rookmove\ ?u\ ?v\ ?x\ ?v)$

$(clearrow\ wr\ ?x\ ?u\ ?v))$

$(<= \ checkmate$

$check$

$stuck)$

$(<= \ check$

$(true\ (cell\ ?u\ ?v\ bk))$

$(attacked\ bk\ ?u\ ?v))$

$(<= \ stuck$

$(not\ canmove))$

$(<= \ canmove$

$(true\ (cell\ ?u\ ?v\ bk))$

$(true\ (cell\ ?x\ ?y\ b))$

$(kingmove\ ?u\ ?v\ ?x\ ?y)$

$(not\ (attacked\ bk\ ?x\ ?y))$

$(not\ (guarded\ ?x\ ?y)))$

$(<=\ (restricted\ ?x\ ?y)$

$(true\ (cell\ ?u\ ?v\ bk))$

$(kingmove\ ?u\ ?v\ ?x\ ?y))$

$(<=\ (guarded\ ?x\ ?y)$

$(true\ (cell\ ?u\ ?v\ wk))$

$(kingmove\ ?u\ ?v\ ?x\ ?y))$

$(<=\ (attacked\ ?p\ ?u\ ?w)$

$(true\ (cell\ ?u\ ?v\ wr))$

$(clearcolumn\ ?p\ ?u\ ?v\ ?w))$

$(<=\ (attacked\ ?p\ ?u\ ?v)$

$(true\ (cell\ ?u\ ?w\ wr))$

$(clearcolumn\ ?p\ ?u\ ?v\ ?w))$

$(<=\ (attacked\ ?p\ ?u\ ?v)$

$(true\ (cell\ ?x\ ?v\ wr))$

$(clearrow\ ?p\ ?u\ ?x\ ?v))$

$(<=\ (attacked\ ?p\ ?u\ ?v)$

$(true\ (cell\ ?x\ ?v\ wr))$

$(clearrow\ ?p\ ?x\ ?u\ ?v))$

$(<=\ (clearcolumn\ ?p\ ?x\ ?y1\ ?y2)$

$(next\_rank\ ?y1\ ?y2)$

$(coordinate\ ?x)$

$(piece\ ?p))$

$(<=\ (clearcolumn\ ?p\ ?x\ ?y1\ ?y3)$

$(next\_rank\ ?y1\ ?y2)$

$(next\_rank\ ?y2\ ?y3)$

$(or\ (true\ (cell\ ?x\ ?y2\ b))\ (true\ (cell\ ?x\ ?y2\ ?p)))$

$(piece\ ?p))$

$(<=\ (clearcolumn\ ?p\ ?x\ ?y1\ ?y4)$

$(next\_rank\ ?y1\ ?y2)$

$(next\_rank\ ?y2\ ?y3)$

$(next\_rank\ ?y3\ ?y4)$

$(or\ (true\ (cell\ ?x\ ?y2\ b))\ (true\ (cell\ ?x\ ?y2\ ?p)))$

$(or\ (true\ (cell\ ?x\ ?y3\ b))\ (true\ (cell\ ?x\ ?y3\ ?p)))$

$(piece\ ?p))$

$(<=\ (clearrow\ ?p\ ?x1\ ?x2\ ?y)$

$(next\_file\ ?x1\ ?x2)$

$(coordinate\ ?y)$

$(piece\ ?p))$

$(<=\ (clearrow\ ?p\ ?x1\ ?x3\ ?y)$

$(next\_file\ ?x1\ ?x2)$

$(next\_file\ ?x2\ ?x3)$

$(or\ (true\ (cell\ ?x2\ ?y\ b))\ (true\ (cell\ ?x2\ ?y\ ?p)))$

$(piece\ ?p))$

$(<=\ (clearrow\ ?p\ ?x1\ ?x4\ ?y)$

$(next\_file\ ?x1\ ?x2)$

$(next\_file\ ?x2\ ?x3)$

$(next\_file\ ?x3\ ?x4)$

$(or\ (true\ (cell\ ?x2\ ?y\ b))\ (true\ (cell\ ?x2\ ?y\ ?p)))$

$(or\ (true\ (cell\ ?x3\ ?y\ b))\ (true\ (cell\ ?x3\ ?y\ ?p)))$

$(piece\ ?p))$

$(piece\ wk)$

$(piece\ bk)$

$(piece\ wr)$

$(<= \ (goal\ white\ 100)$

$checkmate)$

$(<= \ (goal\ white\ 0)$

$(not\ checkmate))$

$(<= \ (goal\ black\ 100)$

$(not\ checkmate))$

$(<= \ (goal\ black\ 0)$

$checkmate)$

$(<= \ terminal$

$(true\ (step\ 10)))$

$(<= \ terminal$

$stuck)$

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Οι γενικοί παίκτες εξαρτώνται απο δύο παράγοντες: την υλοποίηση των εσωτερικών μεθόδων τους και τις δυνατότητες που προσφέρει η γλώσσα περιγραφής παιχνιδιών. Τα τελευταία χρόνια έχουν γίνει ραγδαίες εξελίξεις στον τομέα των γενικών παικτών, αλλά ακόμα χρειάζονται βελτίωση ώστε να μπορούν να παίξουν «καλά» παιχνίδια όπως το σκάκι και το go που είναι άκρως απαιτητικά.

Όσον αφορά τη γλώσσα περιγραφής παιχνιδιών, δηλαδή τη γλώσσα GDL, ακόμα και για παιχνίδια με απλούς κανόνες, χρειάζονται, πολλές φορές, αρκετές γραμμές κώδικα για να περιγραφούν. Είναι σημαντικό να προστεθούν επεκτάσεις στη GDL ώστε να γίνει πιο εύκολη η περιγραφή παιχνιδιών ή τουλάχιστον ομάδων παιχνιδιών.

Όσον αφορά το SweetGGP, είναι σε πολύ βασικό στάδιο και χρειάζεται μεγάλη βελτίωση, κυρίως όσον αφορά την αξιολόγηση που κάνει. Πρέπει να προστεθούν περισσότερες ευριστικές και πιθανότατα να αλλάξει η μέθοδος αναζήτησης και να χρησιμοποιηθεί αναζήτηση Monte – Carlo.

Ο τομέας του γενικού παιξίματος έχει πολλά περιθώρια εξέλιξης και μπορεί να βοηθήσει και σε άλλους τομείς της τεχνητής νοημοσύνης, όπως η κατάστρωση σχεδίου.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
General Game Playing	Γενικό παίξιμο
General Game Player	Γενικός παίκτης
Game Description Language	Γλώσσα αναπαράστασης παιχνιδιού
Transposition tables	Πίνακες μετάθεσης
Retrograde analysis	Μέθοδος οπισθοδρομικής ανάλυσης
Monte - Carlo Tree Search	Αναζήτηση δέντρου Monte - Carlo
Domain size	Πεδίο ορισμού
Cell	Κελί
Relation	Σχέση
Relation Collection	Συλλογή από σχέσεις
Goal	Στόχος
Role	Ρόλος
Disjunctive Normal Form	Διαζευκτική κανονική μορφή
Lexer	Λεκτικός αναλυτής
Parser	Συντακτικός ανάλυτης
Relation prover	Αποδεικτικός αναλυτής σχέσεων
Depth First Search	Αναζήτηση κατά βάθος
Sum	Άθροισμα
Expand	Ξεδίπλωμα
Knowledge Interchange Format	Μορφή ανταλλαγής γνώσης
Unification	Ενοποίηση
Alpha – Beta Pruning	Κλάδεμα άλφα – βήτα

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

GGP	General Game Player
GDL	Game Description Language
DFS	Depth First Search
DNF	Disjunctive Normal Form,
MCTS	Monte – Carlo Tree Search
KIF	Knowledge Interchange Format

ΑΝΑΦΟΡΕΣ

[1] Schiffel, S., Thielscher, M.: *Fluxplayer: A successful general game player*. In: Proceedings of the National Conference on Artificial Intelligence, AAAI Press, Vancouver 2007, pp. 1191-1196.

[2] B. D. Pell. *Strategy generation and evaluation for meta-game playing*. Ph.D. dissertation, University of Cambridge, August 1993.

[3] Fawcett, T. E, and Utgoff, P. E. *Feature discovery for inductive concept learning*. Technical Report UM-CS-1990-015, Department of Computer Science, University of Massachusetts, 1993.

[4] Bonet, B., & Geffner, H. (1999). *Planning as heuristic search: new results*. In Proceedings of ECP-99, 1999.

[5] Daniel Michulke and Stephan Schiffel. *Distance features for general game playing*. In: Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA), pages 7-14, Barcelona, 2011.

[6] Kuhlmann, G.; Dresner, K.; and Stone, P. *Automatic heuristic construction in a complete general game player.* In Proc. of AAAI, 1457–62, 2006.

[7] Michulke D. *Heuristic Interpretation of Predicate Logic Expressions in General Game Playing*. In: Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA), pages 61-68, Barcelona, 2011.

[8] Gudmundsson S. F., Bjornsson Y. *MCTS: Improved Action Selection Techniques for Deterministic Games*. School of Computer Science, Reykjavik University, Iceland, 2011.

[9] Schiffel, S., Thielscher, M. *Reasoning About General Games Described in GDL-II.* In Proc. of AAAI-11, 2011.

[10] Ruan J., Thielscher M. *On the Comparative Expressiveness of Epistemic Models and GDL-II*. School of Computer Science and Engineering, The University of New South Wales, Australia, 2011.

[11] Marek Strejczek, *Some aspects of chess programming*. Master's thesis, Technical University of Łódź, Faculty of Electrical and Electronic Engineering, Department of Computer Science, 2004.

[12] Richard E. Korf, *Multi player alpha – beta pruning*. Computer Science Department, University of California, Los Angeles, 1991.

[13] Feng-hsiung Hsu, Murray Campbell and A. Joseph Hoane Jr. *Deep Blue.* 2001; http://sjeng.org/ftp/deepblue.pdf .

[14] Michael Genesereth, Nathaniel Love, and Barney Pell. *General Game Playing: Overview of the AAAI Competition*. In AAAI-05 AI Magazines, 2005.

[15] Gregory Kuhlmann, Kurt Dresner, Peter Stone , *Automatic Heuristic Construction in a Complete General Game Player.* In Proc. of AAAI-06, 2006.

[16] Nathaniel Love, Timothy Hinrichs, Michael Genesereth. *General Game Playing: Game Description Language Specification.* Stanford Logic Group, Computer Science Department, Stanford University, 2006.

[17] Michael Thielscher. *A General Game Description Language for Incomplete Information Games.* In Proc. of AAAI-10, 2010.

[18] Yngvi Björnsson, Hilmar Finnsson. *CADIAPLAYER: A Simulation-Based General Game Player.* IEEE transactions on computational intelligence and AI in Games, Vol. 1, No. 1, March 2009.