Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Interactive theorem proving with Machine Learning

Diploma Thesis

by

Apostolos Kyteas

Επιβλέπων: Νικόλαος Παπασπύρου
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Γλωσσών Προγραμματισμού

Interactive theorem proving with Machine Learning

Diploma Thesis

by

Apostolos Kyteas

Επιβλέπων: Νικόλαος Παπασπύρου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή τη 19η Οκτωβρίου, 2023.

.....	.....	.....
Νικόλαος Παπασπύρου	Γεώργιος Στάμου	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2023

.....
Αποστολος Κυτεας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η Αυτοματοποιημένη Διαδραστική Σύνθεση Αποδείξεων με Μηχανική Μάθηση έχει δει σημαντική ανάπτυξη τα τελευταία χρόνια. [47], [33]. Είναι ένας τομέας που επιχειρεί να υποκαταστήσει την αλληλεπίδραση των χρηστών με βοηθούς απόδειξης με μοντέλα μηχανικής μάθησης τα οποία είναι υπεύθυνα για την παροχή των υποψήφιων τακτικών.

Το CoqGym είναι ένα σύνολο δεδομένων και περιβάλλον μάθησης για Διαδραστική Σύνθεση Αποδείξεων που περιέχει 71 χιλιάδες αποδείξεις γραμμένες από ανθρώπους για τον βοηθό αποδείξεων Coq, το οποίο δημιουργήθηκε εξάγοντας αποδείξεις από 123 σύνολα αποδείξεων [46]. Το CoqGym αποτέλεσε τη βάση για τη δημιουργία διάφορων εργαλείων σύνθεσης αποδείξεων με προβλέψεις μηχανικής μάθησης, όπως το ASTactic [46], το TacTok [11] και το Passport [33].

Σε αυτή την εργασία επιχειρούμε να βελτιώσουμε τις αρχιτεκτονικές που προτάθηκαν από τη βιβλιογραφία, αντικαθιστώντας τους παραδοσιακούς αλγόριθμους αναζήτησης που χρησιμοποιούνται για την εξερεύνηση του χώρου ταακτικών ο οποίος παράγεται από τα μοντέλα μηχανικής μάθησης, και να χρησιμοποιήσουμε Αναζήτηση Δέντρου Μόντε Κάρλο (MCTS). Εισάγουμε τρεις πολιτικές ανταμοιβών για την υλοποίηση μας: την πολιτική ανταμοιβής "σε τερματικές καταστάσεις", την πολιτική ανταμοιβής "βάσει του βάθους" και την πολιτική ανταμοιβής "με βαση τη μείωση των στόχων". Δείχνουμε ότι το MCTS με περιορισμένη ρύθμιση των παραμέτρων του είναι ικανό να παράγει αποτελέσματα πολύ κοντά στα κορυφαία αποτελέσματα που παρουσιάζονται στη βιβλιογραφία.

Λέξεις-κλειδιά — Coq, CoqGym, Αναζήτηση Μόντε Κάρλο σε Δέντρα (MCTS), ASTactic, TacTok, Passport, Απόδειξη Θεωρημάτων, Διαδραστική Απόδειξη Θεωρημάτων, Καθοδηγούμενη Σύνθεση Αποδείξεων, Βοηθοί Αποδείξεων, Μοντέλα Μακράς - Βραχυπρόθεσμης Μνήμης

Abstract

Automated Interactive Theorem Proving with Machine Learning has seen significant advances [47], [33]. It is a domain which attempts to replace the human element in the interaction with proof assistants with machine learning models which are responsible for providing the candidate tactics.

CoqGym is a dataset and learning environment for ITP containing 71K human-written proofs, which was created by extracting proofs from 123 open-source Coq projects [46]. CoqGym has been the basis for the creation of several machine learning prediction tools for guided proof synthesis, like ASTactic [46], TacTok [11] and Passport [33].

In this thesis we attempt to improve the architectures suggested from previous work by replacing the traditional search algorithms used for the exploration of the tactic space generated by the machine learning models with Monte Carlo Tree Search. We introduce three reward policies for our MCTS implementation: the "reward on terminal states" policy, the "reward on depth" policy and the "reward on goal reduction" policy. We show that MCTS with limited tuning is capable of producing results very close the the top results presented in literature.

Keywords — Coq, CoqGym, Monte Carlo Tree Search (MCTS), ASTactic, TacTok, Passport, Theorem Proving, Interactive Theorem Proving, Proof Assistants, Long Short Term Memory Networks

Ευχαριστίες

Ευχαριστώ θερμά την Αγγελική Δημητρίου για την υπέροχη συνεργασία κατά την εκπόνηση αυτής της διπλωματικής εργασίας, το Δημήτρη Βυτινιώτη για την πρόταση αυτού του καταπληκτικού θέματος και την καθοδήγησή του και το καθηγητή Γιώργο Στάμου που επέμεινε να δουλέψουμε σε αυτό και για όλες τις ενδιαφέρουσες ιδέες που είχε στις συζητήσεις μας.

Ιδιαίτερα, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στον καθηγητή Νίκο Παπασπύρου, για τη στήριξη, την υπομονή και την εμπιστοσύνη που έδειξε ώστε να ολοκληρωθεί αυτή η διπλωματική εργασία.

Τέλος, ένα μεγάλο ευχαριστώ στους συμφοιτητές και φίλους μου, που ομόρφυναν την κάθε μου μέρα στη σχολή, μου χάρισαν τις γνώσεις και τις ιδέες τους και που τώρα έχουν σκορπίσει και δημιουργούν σε όλα τα μέρη της γης.

Κυτέας Απόστολος, Οκτώβριος 2023

Contents

List of Figures

Chapter 1

Coq & CoqGym

Coq

Το σύστημα COQ σχεδιάστηκε για την ανάπτυξη μαθηματικών αποδείξεων και ιδιαίτερα για τη σύνταξη τυπικών προδιαγραφών, προγραμμάτων και τον έλεγχο της ορθότητας των προγραμμάτων σε σχέση με τις προδιαγραφές τους [2]. Χρησιμοποιείται εκτενώς για ερευνητικούς και βιομηχανικούς σκοπούς για την ανάπτυξη λογισμικού υψηλής αξιοπιστίας. Παραδείγματα της χρήσης του περιλαμβάνουν την απόδειξη θεωρημάτων στα μαθηματικά [13], την επαλήθευση λογισμικού και πρωτοκόλλων ασφαλείας και τον σχεδιασμό και δημιουργία γλωσσών προγραμματισμού. Χρησιμοποιήθηκε επίσης στον επαληθευμένο με το CompCert επεξεργαστή C, ο οποίος χρησιμοποιείται για την τυπική επαλήθευση ρεαλιστικών μεταγλωττιστών (realistic compilers). Τέτοιοι επαληθευμένοι μεταγλωττιστές διαθέτουν μαθηματική, μηχανική επαλήθευση ότι ο παράγομενο εκτελέσιμος κώδικα συμπεριφέρεται ακριβώς όπως προβλέπεται από τη σημασιολογία του πηγαίου κώδικα (source code) [22].

Ο βοηθός απόδειξης Coq είναι ένα ανθεκτικό και ευέλικτο εργαλείο λογισμικού που χρησιμοποιείται εκτενώς για την τυπική επαλήθευση στα μαθηματικά, την επιστήμη των υπολογιστών και σε τομείς πέραν αυτών. Παρέχει ένα στέρεο πλαίσιο για την θεμελίωση της ορθότητας μαθηματικών θεωρημάτων και προγραμμάτων λογισμικού. Στην καρδιά του, το Coq βασίζεται στον Λογισμό Επαγωγικών Κατασκευών (Calculus of Inductive Constructions), ένα ισχυρό λογικό θεμέλιο που συγχωνεύει μια λογική υψηλότερης τάξης και μια richly-typed συναρτησιακή γλώσσα προγραμματισμού.

Κεντρικό στοιχείο της δομής του Coq είναι η γλώσσα GALLINA, μια γλώσσα υψηλού επιπέδου. Η GALLINA επιτρέπει στους χρήστες να αναπαραστούν όχι μόνο προγράμματα αλλά και ιδιότητες αυτών των προγραμμάτων και αποδείξεις αυτών των ιδιοτήτων. Αυτή η εκφραστική γλώσσα αποτελεί τη γέφυρα μεταξύ του τυπικού μαθηματικού συλλογισμού και της πρακτικής επαλήθευσης λογισμικού.

Η GALLINA είναι εξοπλισμένη με μια ευρεία γκάμα χαρακτηριστικών που διευκολύνουν τον τυπικό συλλογισμό. Υποστηρίζει τον ορισμό αναδρομικών τύπων δεδομένων, συναρτήσεων και θεωρημάτων, τα οποία υπόκεινται σε ακριβές έλεγχο τύπων από τον πυρήνα του Coq. Επιπλέον, η διαδραστική της φύση επιτρέπει στους χρήστες να κατασκευάζουν εκ των υστέρων σύνθετες αποδείξεις, βελτιώνοντάς τες επαναληπτικά.

Στο οικοσύστημα του Coq, ο πυρήνας του Coq λειτουργεί ως το θεμελιώδες επίπεδο, υπεύθυνο για τον έλεγχο τύπων, τη δημιουργία αντικειμενικών αποδείξεων ελέγξιμων από μηχανή και τη διατήρηση του λογικού πλαισίου. Πάνω από αυτόν τον πυρήνα, ο μηχανισμός απόδειξης ερμηνεύει τα σενάρια αποδείξεων που παρέχονται από τον χρήστη, προσφέροντας ένα διαδραστικό και χρηστικό περιβάλλον για τον τυπικό συλλογισμό.

Η επεκτασιμότητα του Coq είναι σημαντικό πλεονέκτημα, επιτρέποντας την ανάπτυξη προσαρμοσμένων τύπων δεδομένων, κανόνων απόδειξης και τακτικών, καθιστώντας το προσαρμόσιμο σε μια ποικιλία διαφορετικών εργασιών τυπικής επαλήθευσης.

Ατομικές & Σύνθετες τακτικές

Ατομικές Τακτικές: Οι ατομικές τακτικές στο Coq είναι βασικές και απλές τακτικές που εκτελούν απλές, εστιασμένες λειτουργίες στην απόδειξη [2]. Αυτές οι τακτικές χρησιμοποιούνται συνήθως για να κάνουν μικρά, σταδιακά βήματα στην κατασκευή της απόδειξης. Παραδείγματα ατομικών τακτικών στο Coq είναι το intros (για την εισαγωγή μεταβλητών ή υποθέσεων), το apply (για την εφαρμογή ενός λήμματος ή θεωρήματος), το simpl (για την απλοποίηση εκφράσεων) και το rewrite (για την αντικατάσταση όρων βάσει ισοτήτων). Οι ατομικές τακτικές είναι τα βασικά στοιχεία των πιο πολύπλοκων σεναρίων απόδειξης.

Σύνθετες Τακτικές: Από την άλλη πλευρά, οι σύνθετες τακτικές είναι τακτικές που συνδυάζουν πολλαπλές ατομικές τακτικές σε ένα μόνο βήμα [2]. Αυτές επιτρέπουν στους χρήστες να αυτοματοποιήσουν ή να συντάξουν πιο περίπλοκα βήματα απόδειξης, καθορίζοντας μια ακολουθία ενεργειών. Οι σύνθετες τακτικές μπορούν να περιλαμβάνουν συνθήκες διακλάδωσης, επανάληψη κα. Παραδείγματα σύνθετων τακτικών στο Coq περιλαμβάνουν το repeat (για επανειλημμένη εφαρμογή μιας άλλης τακτικής), το ; (ερωτηματικό, χρησιμοποιείται για την εφαρμογή τακτικών σειριακά) και τα tacticals ;, | και +, τα οποία παρέχουν διάφορους τρόπους ελέγχου της διαδικασίας απόδειξης.

Ένα παράδειγμα ατομικών τακτικών:

```
intros x y.
apply Nat.add_comm.
```

An example of compound tactics:

```
intros x y; apply Nat.add_comm.
```

CoqGym

CoqGym: Ένα σύνολο δεδομένων και περιβάλλον μάθησης για το CoqGym

Το CoqGym είναι ένα σύνολο δεδομένων και περιβάλλον μάθησης για Διαδραστική Σύνθεση Αποδείξεων που περιέχει 71 χιλιάδες αποδείξεις γραμμένες από ανθρώπους [46] για τον βοηθό αποδείξεων Coq. Δημιουργήθηκε από την εξαγωγή αποδείξεων από 123 projects ανοιχτού κώδικα στο Coq και καλύπτει ένα εύρος πεδίων εφαρμογής, συμπεριλαμβανομένων των μαθηματικών, του υλικού υπολογιστών, των γλωσσών προγραμματισμού, κ.λπ.

Προηγούμενες προσπάθειες δημιουργίας συνόλων δεδομένων για τον βοηθό αποδείξεων Coq οδήγησαν σε σύνολα που αποτελούνταν από μερικές χιλιάδες θεωρήματα. Επιπλέον, κάλυπταν μόνο έναν περιορισμένο εύρος πεδίων, όπως η αριθμητική Peano [10] ή το θεώρημα περιττής τάξης Feit–Thompson [14]. Το CoqGym είναι μεγαλύτερο και διαθέτει περισσοότερη ποικιλία, επομένως διευκολύνει την εκπαίδευση μοντέλων μηχανικής μάθησης. Δίνει επιπλέον τη δυνατότητα αξιολόγησης του διατομεακά.

Το περιβάλλον του CoqGym σχεδιάστηκε για να εκπαιδεύει και να αξιολογεί πράκτορες αυτόματης διαδραστικής σύνθεσης αποδείξεων. Ο πράκτορας ξεκινά από το θεώρημα που πρέπει να αποδειχθεί μαζί με ένα σύνολο προκειμένων. Η αλληλεπίδραση με τον βοηθό απόδειξης περιλαμβάνει την παραγωγή μιας σειράς τακτικών. Ο βοηθός απόδειξης εκτελεί κάθε μία από αυτές και επιστρέφει αποτελέσματα στη μορφή νέων στόχων. Η απόδειξη έχει επιτυχώς βρεθεί από τον πράκτορα όταν δεν υπάρχουν πλέον ανοιχτοί στόχοι προς απόδειξη.

Δομή του συνόλου δεδομένων

Όπως ήδη αναφέρθηκε, το CoqGym περιλαμβάνει ένα μεγάλης κλίμακας σύνολο δεδομένων από 71 χιλιάδες αποδείξεις γραμμένες από ανθρώπους οι οποίες εξήχθησαν από 123 projects ανοιχτού κώδικα στο Coq. Εκτός από τα αρχεία πηγαίου κώδικα, περιλαμβάνει επίσης τα δέντρα συντακτικής ανάλυσης (ASTs) και πλούσιες πληροφορίες λειτουργίας των αποδείξεων, συμπεριλαμβανομένων των περιβάλλοντων, των στόχων και των δέντρων απόδειξης. Τα ASTs έχουν εξαχθεί από τον διερμηνέα (interpreter) του Coq ως τύποι δεδομένων OCaml. Τα αφηρημένα συντακτικά δέντρο AST μετατρέπονται σε Σ-εκφράσεις (συμβολικές) σε Lisp [26]. Το περιβάλλον CoqGym παρέχει εργαλεία για τη χρήση τους σε Python.

Επεξεργασία projects και αρχείων Coq

Τα αρχεία πηγαίου κώδικα που αποτελούν το σύνολο δεδομένων οργανώνονται ανά project (τομέα αποδείξεων). Κάθε project περιλαμβάνει ένα σύνολο σχετικών αποδείξεων για συγκεκριμένους τομείς. Τα projects στο CoqGym περιλαμβάνουν την πρότυπη βιβλιοθήκη του Coq και τα πακέτα που αναφέρονται στον Δείκτη Πακέτων του Coq [7]. Ορισμένα από αυτά ενδέχεται να μην μεταγλωττίζονται επειδή απαιτούν μια συγκεκριμένη έκδοση του Coq ή έχουν εξαρτήσεις που δεν είναι διαθέσιμες. Μόνο τα projects που μπορούν να μεταγλωττιστούν συμπεριλαμβάνονται στο σύνολο δεδομένων.

Τα σύνολα εκπαίδευσης, επικύρωσης και ελέγχου αποτελούνται από διαφορετικά projects. Αυτό έγινε για τους ακόλουθους λόγους:

- Καθώς οι αποδείξεις σε κάθε project είναι σχετικές μεταξύ τους, ήταν απαραίτητο να εξασφαλιστεί ότι οι αποδείξεις ελέγχου δεν χρησιμοποιούνταν κατά τη διάρκεια της εκπαίδευσης.
- Ο στόχος της εκπαίδευσης των μοντέλων είναι η γενίκευση σε διαφορετικούς τομείς.

Η διαίρεση έγινε ως εξής:

- Σύνολο δεδομένων εκπαίδευσης: 43,844 αποδείξεις
- Σύνολο δεδομένων επικύρωσης: 13,875 αποδείξεις
- Σύνολο δεδομένων ελέγχου: 13,137 αποδείξεις

SerAPI

"SerAPI" είναι "μια βιβλιοθήκη για τη μηχανική αλληλεπίδραση με το Coq" [1], που διευκολύνει την αλληλεπίδραση με αυτό χρησιμοποιώντας σειριοποιημένα δεδομένα και ασύγχρονη επικοινωνία. Σχεδιάστηκε για να επικοινωνεί με εξωτερικά εργαλεία και εφαρμογές χρησιμοποιώντας σειριοποιημένες αναπαραστάσεις των εσωτερικών τύπων δεδομένων OCaml του Coq. Τα σειριοποιημένα δεδομένα μπορούν να είναι σε διάφορες μορφές όπως JSON ή Σ-εκφράσεις, κάτι που διευκολύνει τα εξωτερικά εργαλεία να αναλύουν και να επεξεργάζονται κώδικα Coq. Είναι ιδιαίτερα χρήσιμο για εργασίες όπως η ανάλυση κώδικα, η μετασχηματισμός και η εξαγωγή δεδομένων στο πλαίσιο του Coq.

Συνθετικές αποδείξεις από ενδιάμεσους στόχους

Το σύνολο δεδομένων CoqGym έχει εμπλουτιστεί με συνθετικές αποδείξεις. Οι συνθετικές αποδείξεις [46] είναι αποδείξεις που συντάχθηκαν για να αποδείξουν τους ενδιάμεσους στόχους υπαρχόντων μακρών αποδείξεων. Ο κύριος σκοπός ήταν ο εμπλουτισμός του υπάρχοντος συνόλου δεδομένων με μικρότερες και λιγότερο πολύπλοκες αποδείξεις. Αυτοί οι ενδιάμεσοι στόχοι θεωρήθηκαν ως πιο εύκολοι να αποδειχθούν και χρήσιμοι για την εκμάθηση των μοντέλων. Επιπλέον, τα δεδομένα εκπαίδευσης επεκτάθηκαν με περισσότερα παραδείγματα. Για να δημιουργήσουν αυτές τις συνθετικές αποδείξεις, οι Yang et al. [46] δημιούργησαν αποδείξεις μήκους 1, 2, 3 και 4 για κάθε ενδιάμεσο στόχο μιας απόδειξης γραμμένης από άνθρωπο.

Δομή απόδειξης - περιβάλλοντα, στόχοι & δέντρα αποδείξεων

Τα περιβάλλοντα των αποδείξεων περιέχουν τα θεμελιώδη στοιχεία πάνω στα οποία θα κατασκευαστεί η απόδειξη. Περιλαμβάνουν όρους Coq ως προκείμενες για τις αποδείξεις. Τα περιβάλλοντα για τις αποδείξεις αναπαρίστανται ως συλλογή όρων πυρήνα, οι οποίοι είναι εσωτερικές αναπαραστάσεις που χρησιμοποιούνται από το Coq. Αυτή η μορφή του περιβάλλοντος δημιουργήθηκε μέσω της εκτέλεσης των αποδείξεων και εν συνεχεία της σειριοποίησης των εσωτερικών του Coq. Δεδομένου ότι ο πηγαίος κώδικας καθορίζει πλήρως το περιβάλλον, αυτός θεωρήθηκε ως μια από τις πιθανές αναπαραστάσεις του που θα μπορούσαν να χρησιμοποιηθούν για την εκπαίδευση. Αυτό θα την καθιστούσε δυσκολότερη, καθώς θα σήμαινε επίσης την εκμάθηση της σημασιολογίας του κώδικα Coq.

Το περιβάλλον για κάθε απόδειξη - οι προκείμενες στο πλαίσιό της - καθορίζεται τόσο στο ίδιο αρχείο πηγαίου κώδικα όσο και σε άλλες βιβλιοθήκες. Σημαντικό είναι να σημειωθεί ότι οι αποδείξεις στο CoqGym περιέχουν το πλήρες περιβάλλον, πράγμα που δεν ίσχυε σε προηγούμενη εργασία [18]. Το όφελος της παρουσίας του πλήρους περιβάλλοντος είναι ότι το μοντέλο μηχανικής μάθησης είναι σε θέση να έχει πρόσβαση σε όλες τις σχετικές πληροφορίες σε δομημένες μορφές.

Οι αποδείξεις στο σύνολο δεδομένων αναπαρίστανται ως δέντρα αποδείξεων. Οι κόμβοι των δέντρων είναι οι στόχοι και το τοπικό πλαίσιο. Οι ακμές είναι τακτικές που μετασχηματίζουν τον τρέχοντα στόχο σε υπο-στόχους. Αυτό επιτεύχθηκε μέσω της σειριοποίησης των τρεχόντων στόχων από τον διερμηνέα του Coq σε κάθε βήμα της απόδειξης. Οι ακμές αναγνωρίζονται με κριτήριο το πώς εμφανίζονται και απλοποιούνται ή εξαφανίζονται οι στόχοι κατά τη διάρκεια της απόδειξης. Τα περιβάλλοντα, οι στόχοι και τα δέντρα απόδειξης σχηματίζουμ μια δομημένη αναπαράσταση των αποδείξεων του Coq. Σε σύγκριση με τον ακατέργαστο πηγαίο κώδικα, μια δομημένη αναπαράσταση επιτρέπει στα μοντέλα μηχανικής μάθησης να αξιοποιήσουν πιο εύκολα τις συντακτικές και σημασιολογικές δομές. Πρέπει να σημειωθεί ότι αυτή η δομημένη αναπαράσταση δεν είναι ευκολό να εξαχθεί επειδή το Coq δεν παρέχει τα κατάλληλα APIs. Κατά τη δημιουργία του CoqGym, το Coq τροποποιήθηκε και μαζί με το SerAPI [1] η ροή πληροφοριών εκτέλεσης σειριοποιήθηκε. Το κύριο μοντέλο ελέγχου αποδείξεων του Coq δεν επηρεάστηκε. Έτσι, η ορθότητα των αποδείξεων δεν θέτεται υπό αμφισβήτηση.

Chapter 2

Μοντέλα

ASTactic: Παραγωγή τακτικών ως προγράμματα

Σε αυτή την ενότητα θα γίνει μια προσπάθεια παρουσίασης της αρχιτεκτονικής του ASTactic, του σχεδιασμού και της δομής του [46]. Αυτό είναι σημαντικό καθώς μοντέλα όπως το Passport [33] και το TacTok [11], τα οποία θα παρουσιαστούν επίσης, δομούνται γύρω από την αρχιτεκτονική του ASTactic και χρησιμοποιούν τμήματα της υλοποίησής του.

Το ASTactic είναι ένα μοντέλο βαθιάς μάθησης που παράγει τακτικές ως προγράμματα [46]. Εκπαιδεύεται στο CoqGym και διακρίνεται από προηγούμενα προτεινόμενα αυτοματοποιημένα συστήματα απόδειξης θεωρημάτων, επειδή κατά τη φάση αναζήτησης της απόδειξης δεν επιλέγει τακτικές από ένα σταθερό σύνολο. Αντ' αυτού, οι τακτικές παράγονται δυναμικά από το ASTactic ως αφηρημένα συντακτικά δέντρα (ASTs). Η έξοδος του ASTactic χρησιμοποιείται στη συνέχεια και δοκιμάζεται μέσω δειγματοληψίας. Σε κάθε κατάσταση (state) της απόδειξης, επιλέγεται ένας αριθμός τακτικών, βάσει του προκαθορισμένου μεγέθους BEAM. Αυτές είναι οι δυνατές ενέργειες που μπορούν να γίνουν από την τρέχουσα κατάσταση. Η αναζήτηση συνεχίζεται μέσω αναζήτησης κατά βάθος (DFS) μέχρι να βρεθεί μια σωστή απόδειξη, να εξαντληθεί ο μέγιστος αριθμός τακτικών ή να λήξει το χρονικό όριο που έχει τεθεί.

Αρχιτεκτονική

Γενική αρχιτεκτονική του μοντέλου

Το ASTactic διαθέτει μια αρχιτεκτονική κωδικοποιητή-αποκωδικοποιητή. Τόσο η είσοδος όσο και η έξοδος του μοντέλου είναι δένδρα. Ο κωδικοποιητής ενσωματώνει όλους τους όρους εισόδου Coq: τον στόχο και τις προκείμενες εκφρασμένες σε ASTs. Εξαρτώμενος από τα embeddings, ο αποκωδικοποιητής παράγει μια τακτική με δομή προγράμματος μέσω της σειριακής ανάπτυξης ενός AST.

Κωδικοποιητής

Το ASTactic κωδικοποιεί τον στόχο και τις προκείμενες κάθε κατάστασης σε διανύσματα.

Συγκεκριμένα, στην κωδικοποίηση περιλαμβάνεται το τοπικό πλαίσιο της απόδειξης και έως 10 προκείμενες στο περιβάλλον. Οι συγγραφείς αποφάσισαν να εξαιρέσουν έναν μεγάλο αριθμό προκειμένων που εισάγωνται από βιβλιοθήκες, οι οποίες δεν σχετίζονταν με την απόδειξη. Ο στόχος και οι προκείμενες είναι όροι του Coq σε μορφή AST και κωδικοποιούνται χρησιμοποιώντας ένα δίκτυο TreeLSTM 4.2.4.

Ειδικότερα, κάθε κόμβος σε ένα AST έχει ένα σύμβολο n που δηλώνει τον συντακτικό του ρόλο. Το δίκτυο συσχετίζει κάθε κόμβο με μια κρυφή κατάσταση h και ένα κελί μνήμης c που ενημερώνονται από τα παιδιά του ως εξής:

$(c, h) = f_{\text{update}}(n, c_1, \ldots, c_K, \sum_{i=1}^{K} h_i)$

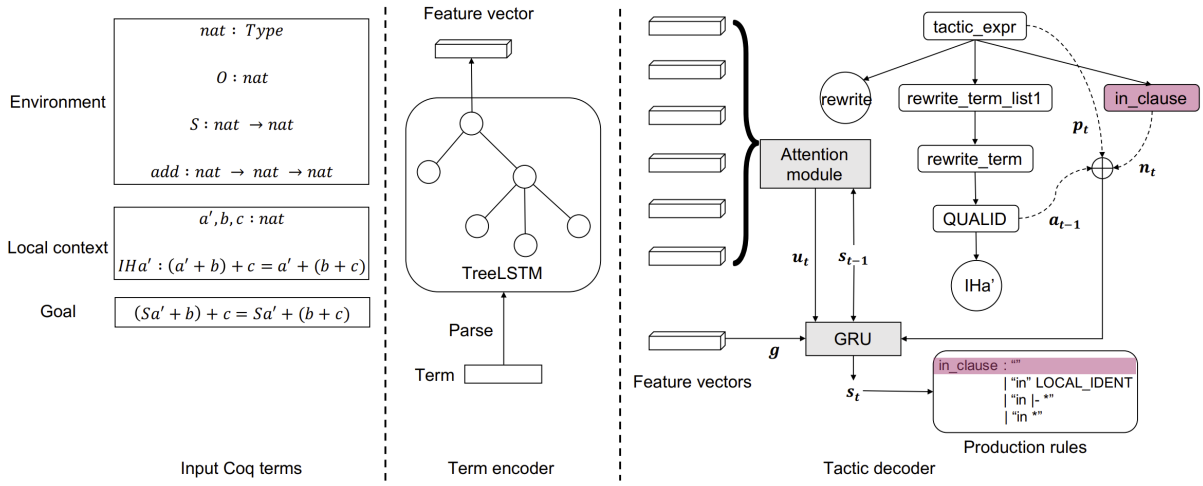


Figure 2.0.1: Η Αρχιτεκτονική του ASTactic. Ο κωδικοποιητής κατάστασης απόδειξης (α) παίρνει ως είσοδο τον στόχο, το τοπικό πλαίσιο και τους όρους του περιβάλλοντος σε μορφή AST και δημιουργεί embeddings (διανύσματα χαρακτηριστικών) για κάθε όρο. Ο αποκωδικοποιητής (β) συνδυάζει τα embeddings εισόδου και δημιουργεί μια τακτική στη μορφή ενός AST, εξαρτώμενο από αυτές τις εισόδους. [46]

Η συνάρτηση ενημέρωσης f_{update} είναι η παραλλαγή Child-Sum του TreeLSTM, n είναι το σύμβολο του κόμβου σε one-hot encoding, και c_i και h_i είναι η κατάσταση κελιού και η κρυφή κατάσταση του κόμβου - παιδιού i.

Αυτός ο υπολογισμός πραγματοποιείται από τα κάτω προς τα πάνω. Ολόκληρο το δένδρο αναπαρίσταται από το h_{root}, την κρυφή κατάσταση της ρίζας. Τελικά, το h_{root} προστίθεται με ένα τρισδιάστατο one-hot διάνυσμα. Το διάνυσμα δείχνει εάν ο όρος είναι ο στόχος, μια προκείμενη στο περιβάλλον, ή μια προκείμενη στο τοπικό πλαίσιο.

Αποκωδικοποιητής

Ρόλος του αποκωδικοποιητή είναι να συνθέσει τακτικές. Ο χώρος αναζήτησης των ορισμάτων περιορίζεται από σημασιολογικούς περιορισμούς οι οποίοι δίνονται μέσω της γραμματικής χωρίς συμφραζόμενα (CFG).

Ο αποκωδικοποιητής του ASTactic δημιουργεί τακτικές με δομή προγράμματος ως Αφηρημένα Συντακτικά Δένδρα (ASTs) ακολουθώντας τη μέθοδο των Yin & Neubig (2017) [48].

Η βάση της μεθόδου τους είναι ένα μοντέλο γραμματικής το οποίο τυποποιεί τη διαδικασία δημιουργίας ενός derivation AST ως μια σειρά ενεργειών. Αυτές οι ενέργειες είναι είτε η εφαρμογή κανόνων παραγωγής είτε η παραγωγή τερματικών συμβόλων. Συνεπώς, το συντακτικό της γλώσσας προγραμματισμού προκαθορίζεται μέσω του μοντέλου γραμματικής ως ένα σύνολο πιθανών ενεργειών. Έτσι η προσέγγισή τους εξαλείφει την ανάγκη το μοντέλο να μάθει τη γραμματική αποκλειστικά με χρήση των δεδομένων εκπαίδευσης (training dataset). Αντ' αυτού, επιτρέπει στο σύστημα να επικεντρωθεί στο πώς οι υφιστάμενοι κανόνες γραμματικής αλληλεπιδρούν και συνδυάζονται [48].

Η μέθοδος ξεκινά με έναν αρχικό κόμβο και αναπτύσσει ένα μερικό δέντρο κατά depth-first με τον εξής τρόπο:

- Επεκτείνει μη τερματικούς κόμβους επιλέγοντας έναν κανόνα παραγωγής από τη γραμματική (Context-Free Grammar - CFG) του χώρου τακτικών.
- Για τερματικούς κόμβους, επιστρέφει μία λεκτική μονάδα που αντιστοιχεί σε ένα όρισμα της τακτικής.

Αυτή η σειριακή διαδικασία παραγωγής ελέγχεται από ένα Gated Recurrent Unit - GRU 4.3.

TacTok

Το TacTok είναι ένα μοντέλο για την πρόβλεψη της επόμενης τακτικής στο Coq, που δημιούργησαν οι First et al. [11]. Η ιδέα για τη δημιουργία του TacTok βασίζεται σε δύο παρατηρήσεις:

1. Όταν οι χρήστες αλληλεπιδρούν με το Coq για να αποδείξουν ένα θεώρημα ή να επαληθεύσουν μια απόδειξη, συχνά πρέπει να εξετάζουν την κατάσταση της απόδειξης για να επιλέξουν την επόμενη τακτική που θα εφαρμόσουν. Αυτό έδωσε την ιδέα να δοκιμαστεί το αν τα μοντέλα πρόβλεψης της επόμενης τακτικής θα μπορούσαν να επωφεληθούν από την πρόσβαση στις πληροφορίες της κατάστασης της απόδειξης.
2. Καθώς οι χρήστες του Coq, κατά τη δημιουργία μιας απόδειξης, γνωρίζουν όλες τις τακτικές που έχουν εφαρμοστεί μέχρι εκείνο το σημείο, ένα μοντέλο πρόβλεψης της επόμενης τακτικής μπορεί επίσης να επωφεληθεί από την πρόσβαση στις προηγούμενες τακτικές στο σενάριο απόδειξης.

Το μοντέλο εκπαιδεύεται στο CoqGym, μέσω της ακόλουθης τεχνικής:

- Διασχίζει τα υπάρχοντα scripts απόδειξης, διασχίζοντάς τα, μια τακτική ανά βήμα. Για κάθε βήμα υπολογίζει τις ενδιάμεσες καταστάσεις απόδειξης που προκύπτουν.
- Δημιουργεί μια ενσωμάτωση (embedding) τόσο των καταστάσεων απόδειξης όσο και του script απόδειξης σε κάθε βήμα. Οι ενσωματώσεις αντιστοιχίζονται σε ένα αφηρημένο συντακτικό δέντρο (AST) της επόμενης γραμμής στο script απόδειξης.

Έτσι, οι είσοδοι του μοντέλου είναι το μερικώς κατασκευασμένο script απόδειξης και η κατάσταση απόδειξης.

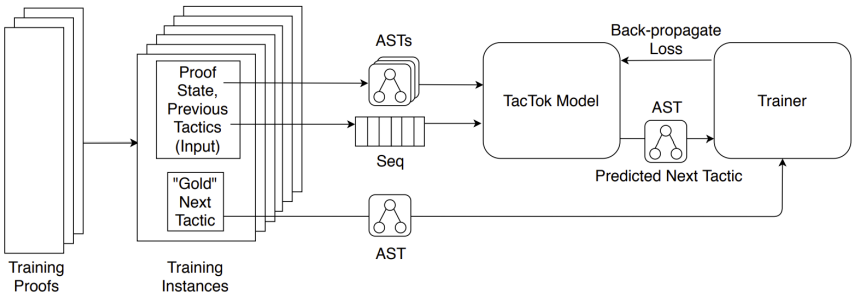
Η έξοδος του μοντέλου είναι ένα AST που το TacTok αποκωδικοποιεί δημιουργώντας την επόμενη προβλεπόμενη τακτική και τα ορίσματά της.

Κάθε σενάριο απόδειξης διασπάται σε στιγμιότυπα εκπαίδευσης. Το στιγμιότυπο εκπαίδευσης περιλαμβάνει:

1. την κατάσταση απόδειξης μετά την εφαρμογή μιας τακτική από το script της απόδειξης,
2. το script της απόδειξης μέχρι την τελευταία τακτική,
3. την επόμενη τακτική στο script.

Η κατάσταση απόδειξης περιλαμβάνει:

1. τον τρέχοντα στόχο,
2. το τοπικό πλαίσιο,
3. το περιβάλλον.



(a) Διαδικασία εκπαίδευσης του TacTok [11].

Κάθε όρος στην κατάσταση απόδειξης έχει ένα υποκείμενο AST. Το script της απόδειξης αναπαρίσταται ως μια ακολουθία από λεκτικές μονάδες (tokens). Το μοντέλο TacTok μαθαίνει κοινά embeddings για αυτά τα AST και τις ακολουθίες. Χρησιμοποιεί αυτά τα embeddings για να παράξει ένα προβλεπόμενο επόμενο βήμα στο script απόδειξης, σε μορφή AST. Το προβλεπόμενο επόμενο βήμα σε συνδυασμό με το AST της πραγματικής επόμενης τακτικής που παίρνουμε από την υπάρχουσα απόδειξη χρησιμοποιούνται για την εκπαίδευση του μοντέλου. Ο εκπαιδευτής στη συνέχεια συγκρίνει αυτές τις τακτικές και αναδιανέμει την απώλεια.

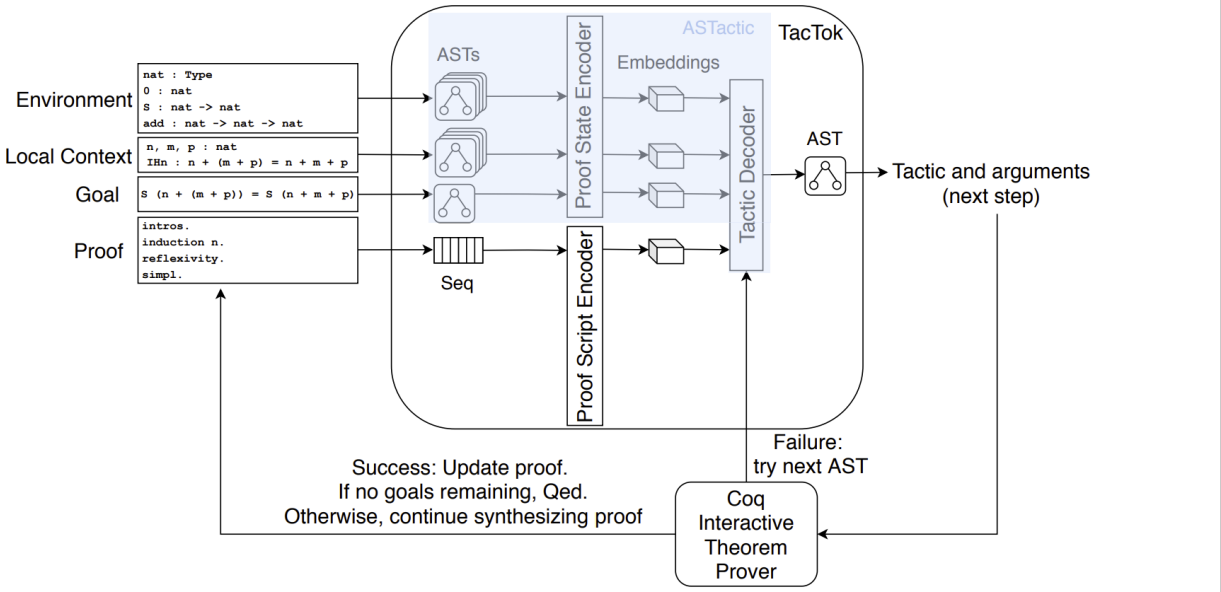


Figure 2.0.3: Το TacTok, κατά τη διαδικασία ολοκλήρωσης του σεναρίου απόδειξης της προσεταιριστικότητας για την πρόσθεση, μετά την εκτέλεση της τακτικής simpl.
[11]

Κωδικοποιητής Κατάστασης Απόδειξης

Ο κωδικοποιητής της κατάστασης απόδειξης είναι αυτός που εισήχθη από το ASTactic και αναλύεται λεπτομερώς στο προηγούμενο τμήμα αυτού του κεφαλαίου. Οι είσοδοι είναι ο στόχος, το τοπικό πλαίσιο, και το περιβάλλον σε μορφή AST.

Κωδικοποιητής Script Απόδειξης

Το script απόδειξης αποτελείται από λεκτικές μονάδες (tokens) που είναι είτε τακτικές, είτε ορίσματα, είτε άλλα σύμβολα. Ο κωδικοποιητής του σεναρίου απόδειξης αναλύει συντακτικά (parsing) την ακολουθία των tokens με δύο διαφορετικές μεθόδους.

- Tac: Κάνει συντακτική ανάλυση της ακολουθίας περιλαμβάνοντας τις πιο κοινές τακτικές του Coq που εμφανίζονται στα σενάρια απόδειξης των δεδομένων εκμάθησης, εξαιρώντας custom τακτικές, και ασαφή ορίσματα ώστε τα ονόματά τους να μην είναι κομμάτι της μάθησης.
- Tok: περιλαμβάνει ολόκληρη την ακολουθία λεκτικών μονάδων, εξαιρώντας μόνο τα σημεία στίξης.

Η αναλυθείσα ακολουθία λεκτικών μονάδων κωδικοποιείται με ένα αμφίδρομο LSTM 4.2.3, που παράγει ένα embedding για την ακολουθία. Με τον τρόπο αυτό η ακολουθία εισόδου υφίσταται επεξεργασία προς δύο κατευθύνσεις. Οι δύο εξόδοι, μία για κάθε κατεύθυνση, συγχωνεύονται για να καταγράψουν το πλαίσιο από τα προηγούμενα και μελλοντικά βήματα απόδειξης.

Το TacTok αποτελείται από τα μοντέλα Tac και Tok, τα οποία εκπαιδεύονται χωριστά, και χρησιμοποιούνται συμπληρωματικά στην απόπειρα σύνθεσης αποδείξεων.

Passport

Στην εργασία τους, ο Sanchez-Stern, A. κ.ά. [33] επικεντρώνονται στη μοντελοποίηση αναγνωριστικών (identifiers). Ο στόχος τους είναι να εκμεταλλευτούν τον πλούτο των δεδομένων απόδειξης στα οποία εκπαιδεύεται το μοντέλο, βελτιώνοντας έτσι την απόδοση των μοντέλων.

Το Passport έχει κατασκευαστεί πάνω στο ASTactic, εμπλουτίζοντας τα υπάρχοντα μοντέλα με την εισαγωγή τριών νέων μηχανισμών κωδικοποίησης για τα αναγνωριστικά:

1. κατηγοριοποίηση λεξιλογίου (Category Vocabulary Indexing),
2. μοντελοποίηση ακολουθουθιών υπολέξεων (subword sequence modeling),
3. επεξεργασία μονοπατιού (path elaboration).

Σε αυτή την ενότητα θα παρουσιάσουμε μια σύντομη επισκόπηση των παραπάνω μηχανισμών.

Κατηγορίες Αναγνωριστικών στο Passport

Τα "αναγνωριστικά" (identifiers) είναι σύμβολα ή ονόματα που χρησιμοποιούνται για να προσδιορίζουν μοναδικά μεταβλητές, συναρτήσεις, σταθερές, ή άλλα στοιχεία σε ένα πρόγραμμα. Στο πλαίσιο των αποδείξεων και καταστάσεων απόδειξης του Coq, τα αναγνωριστικά είναι "τα ονόματα που προσδιορίζουν με μοναδικό τρόπο θεωρήματα, τύπους δεδομένων, συναρτήσεις, κατασκευαστές τύπων και τοπικές μεταβλητές" [33].

Το Passport κωδικοποιεί κάθε αναγνωριστικό ανάλογα με την κατηγορία του:

1. καθολικός ορισμός,
2. τοπική μεταβλητή,
3. κατασκευαστής τύπου.

Μηχανισμοί Κωδικοποίησης στο Passport

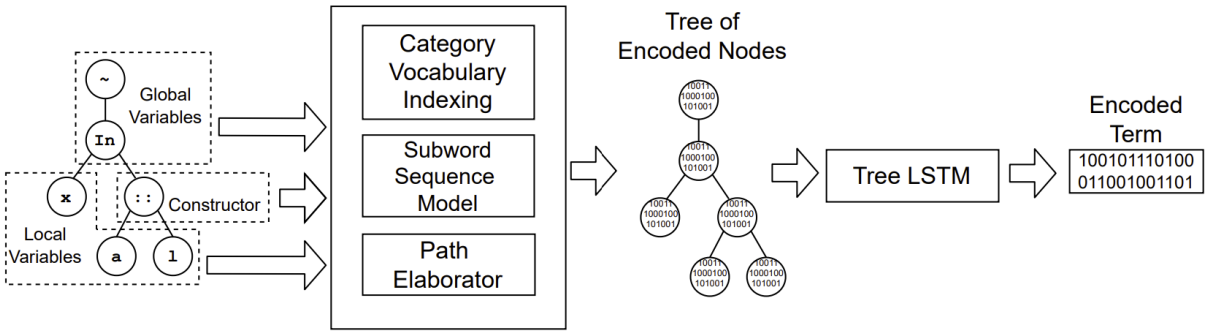


Figure 2.0.4: Η αρχιτεκτονική της επεξεργασίας των αναγνωριστικών στο Passport. [33]

Κατηγοριοποίηση λεξιλογίου:

Οι Sanchez-Stern, A. et al. [33] σημειώνουν ότι, καθώς το Coq δεν διαθέτει πρωτογενείς τύπους δεδομένων, κάθε αναφερόμενος τύπος είναι ένα αναγνωριστικό. Αυτό τους καθιστά πολύ σημαντικους, καθώς μπορούν να κουβαλούν σημαντική πληροφορία η οποία να βρίσκεται στα ονόματα των θεωρημάτων που αναφέρονται σε αυτούς.

Με την κατηγοριοποίηση λεξιλογίου, σε κάθε αναγνωριστικό ανατίθεται μία ετικέτα ή δείκτης με την κατηγορία από την οποία προέρχεται. Αυτό διαχωρίζει αναγνωριστικά με το ίδιο όνομα από διαφορετικές κατηγορίες. Επιπλέον παρέχει στο μοντέλο χρήσιμες πληροφορίες αναγνωριστικά τα οποία δεν συναντόνται συχνά. Τα πιο κοινά αναγνωριστικά σε κάθε κατηγορία λαμβάνουν μια μοναδική ετικέτα, η οποία περιλαμβάνεται στην κωδικοποίησή τους. Έτσι το Passport δημιουργεί μια συσχέτιση μεταξύ όλων των χρήσεων αυτών των αναγνωριστικών. Τέλος, το μοντέλο μπορεί να δημιουργήσει γενικεύσεις σχετικά με τη συμπεριφορά και τις χρήσεις τους, και να προβλέψει τακτικές που λειτούργησαν αποτελεσματικά με αυτά σε προηγούμενες περιπτώσεις.

Μοντελοποίηση Ακολουθιών Υπολέξεων:

Η μοντελοποίηση ακολουθιών υπολέξεων (subword sequence modeling) είναι μια τεχνική που χρησιμοποιείται στην επεξεργασία φυσικής γλώσσας (NLP) και στη μηχανική μάθηση για τον χειρισμό λέξεων και συμβόλων σε πιο λεπτομερές επίπεδο από αυτό των ολόκληρων λέξεων. Εμπλέκει τον χωρισμό των λέξεων σε μικρότερα κομμάτια, όπως τμήματα υπολέξεων ή χαρακτήρες, και τη μοντελοποίηση ακολουθιών αυτών των υπομονάδων [38]. Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη για τον χειρισμό γλωσσών με πολύπλοκη μορφολογία, συγκολλητικές γλώσσες (agglutinative languages) - γλώσσες που δημιουργούν λέξεις μέσω του συνδυασμού μικρότερων μορφημάτων για την έκφραση σύνθετων ιδεών, ή όταν συναντόνται λέξεις που βρίσκονται εκτός λεξιλογίου.

Με αυτήν την τεχνική, οι Sanchez-Stern, A. et al. [33] προσπαθούν να εκμεταλλευτούν τις συσχετίσεις που μπορούν να δημιουργηθούν μεταξύ των αναγνωριστικών με βάση τα ονόματά τους και τις γενικές συμβάσεις ονοματοδοσίας που χρησιμοποιούνται στις αποδείξεις. Για όλα τα αναγνωριστικά, το Passport χρησιμοποιεί ένα μοντέλο ακολουθίας υπολέξεων για να δημιουργήσει γέφυρες μεταξύ σχετικών ονομάτων. Δηλαδή, τα αναγνωριστικά χωρίζονται σε κοινά τμήματα λέξεων και επεξεργάζονται με ένα μοντέλο ακολουθίας.

Για παράδειγμα, το όνομα της μεταβλητής `orderedListAsc` μπορεί να χωριστεί σε "ordered", "list" και "asc".

Επεξεργασία Μονοπατιού:

Η επεξεργασία μονοπατιού είναι η τελευταία τεχνική κωδικοποίησης που χρησιμοποιείται στο Passport: η κωδικοποίηση των απόλυτων μονοπατιών διαφορετικών αναγνωριστικών. Τα απόλυτα μονοπάτια είναι "τα ονόματα των καταλόγων, των αρχείων και των μονάδων" εντός των οποίων περιέχονται τα αναγνωριστικά. Το Passport μπορεί να εκμεταλλευτεί οποιαδήποτε ομαδοποίηση αναγνωριστικών σε κοινές μονάδες και αρχεία που χρησιμοποιούνται ήδη από τους προγραμματιστές του Coq. Μπορεί επίσης να εκμεταλλευτεί το στυλ αποδείξεων που εφαρμόζεται σε κλάσεις σχετικών θεωρημάτων.

Chapter 3

Εξερεύνηση προβλεπόμενων τακτικών με MCTS

Στα τρία εργαλεία πρόβλεψης μηχανικής μάθησης για καθοδηγούμενη σύνθεση αποδείξεων που παρουσιάστηκαν, (ASTactic 5.1, TacTok 5.2, Passport 5.3), χρησιμοποιήθηκαν συμβατικοί αλγόριθμοι αναζήτησης για την εξερεύνηση του χώρου των παραγόμενων τακτικών που προτείνονται από τα μοντέλα πρόβλεψης (DFS, διερευνητική βαθμιαία βελτίωση κλπ). Κατά τη διαδικασία παραγωγής της απόδειξης, ο αλγόριθμος αναζήτησης σε κάθε βήμα δειγματολειπτεί έναν συγκεκριμένο αριθμό τακτικών που παράγονται από το μοντέλο, ανάλογα με το καθορισμένο μέγεθος της ακτίνας(beam) αναζήτησης, και θα συνέχιζε την εξερεύνησητου χώρου.

Για τρία μοντέλα, χρησιμοποιήθηκε ακτίνα μεγέθους 20 κατά τη διάρκεια των πειραμάτων. Αυτή η συγκεκριμένη τιμή αποφασίστηκε μέσω δοκιμών όταν δοκιμάστηκε το ASTactic και υιοθετήθηκε στα πειράματα των υπολοίπων μοντέλων χωρίς περαιτέρω βελτιώσεις. Το μέγεθος της δέσμης "ελέγχει το trade-off μεταξύ ταχύτητας και ακρίβειας" [44]. Ένα μεγάλο πλάτος δέσμης οδηγεί σε μια ευρεία περιοχή αναζήτησης, αυξάνοντας έτσι το ποσοστό επιτυχίας. Το μοντέλο έχει τη δυνατότητα να εξερευνά μεγαλύτερο χώρο αναζήτησης εις βάρος αυξημένου χρόνου αναζήτησης. Ωστόσο, παρατηρήθηκε ότι όταν το πλάτος της δέσμης θα ορίζεται σε τιμές μεγαλύτερες από 20, το ποσοστό επιτυχίας μειωνόταν. Η εξήγηση πίσω από αυτό είναι ότι το μοντέλο "παγιδεύεται" σε έναν μη υποσχόμενο κλάδο για πολύ χρόνο, και η αναζήτηση λήγει λόγω χρονικού ορίου.

Η πρότασή μας σε αυτή τη διπλωματική εργασία είναι να αντικαταστήσουμε τους παραδοσιακούς αλγορίθμους αναζήτησης με τον επαναλαμβανόμενο αλγόριθμο αναζήτησης Monte Carlo Tree Search 6. Υποστηρίζουμε ότι η χρήση του MCTS στον χώρο των παραγόμενων tactics μπορεί να μειώσει τον απαιτούμενο χρόνο αναζήτησης για τον εντοπισμό της σωστής ακολουθίας των tactics που συνιστούν μία πλήρη απόδειξη ενός θεωρήματος. Η ταχύτερη αναζήτηση θα επιτρέψει την αύξηση του αριθμού των δειγματοληπτούμενων tactics σε κάθε βήμα της αναζήτησης (beam), επεκτείνοντας έτσι τον χώρο εξερεύνησης. Η επέκταση του χώρου εξερεύνησης μπορεί να αυξήσει τον συνολικό αριθμό των αποδείξεων που αποδεικνύονται από τα παραπάνω μοντέλα.

Υλοποίηση του Monte Carlo Tree Search

Ορισμός κατάστασης (state)

Για να εκτελέσουμε την αναζήτησή μας πρέπει να καθορίσουμε την κατάσταση(state) της απόδειξης - και με βάση αυτό να κατασκευάσουμε τους κόμβους του δένδρου αναζήτησης. Η κατάσταση(state) της απόδειξης περιέχει το περιβάλλον της απόδειξης (global context), το τοπικό πλαίσιο (local context) και τους στόχους εστιασμένους και μη εστιασμένους (focused and unfocused). Το περιβάλλον της απόδειξης μπορεί να είναι κοινό για όλες τις καταστάσεις και να διαμοιράζεται από αυτές, αλλά κάθε κατάσταση καθορίζεται από το τοπικό της πλαίσιο και τους στόχους της.

Στην υλοποίησή μας, το κάθε αντικείμνο κατάστασης (state object) περιλαμβάνει το tactic (ενέργεια) από το οποίο το δημιουργήθηκε με την εφαρμογή του στην κατάσταση - γονέα. Επίσης περιλαμβάνει το "observation_result" δηλαδή το αποτέλεσμα της κατάστασης (SUCCESS, ERROR κ.λπ.) και το script μέχρι εκείνη την

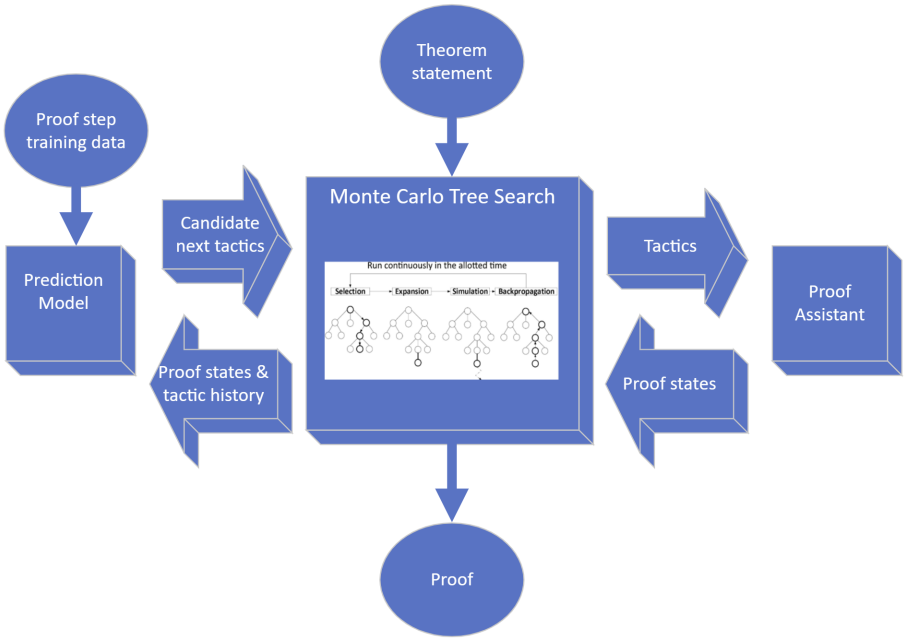


Figure 3.0.1: Η αρχιτεκτονική των εργαλείων πρόβλεψης μηχανικής μάθησης για καθοδηγούμενη σύνθεση αποδείξεων.

κατάσταση, που είναι η ακολουθία των tactics που εφαρμόστηκαν από τη ρίζα του δένδρου μέχρι την κατάσταση αυτή. Για να εκτελέσουμε την αναζήτηση είναι απαραίτητο επίσης να προσθέσουμε τον γονικό κόμβο, το βάθος της κατάστασης, τον αριθμό των επισκέψεων σε αυτή την κατάσταση, τους κόμβους - παιδιά αν αυτά υπάρχουν και τις συνολικές ανταμοιβές (rewards) για την κατάσταση.

Τέλος, προσθέσαμε κάποιες επιπλέον παραμέτρους που διευκόλυναν την υλοποίησή μας (timeouts, αριθμός συνολικών tactics που χρησιμοποιήθηκαν μέχρι εκείνο το σημείο, reference στο περιβάλλον της απόδειξης και άλλα).

Τερματικές και μη τερματικές καταστάσεις

Στην υλοποίησή μας, οι τερματικές καταστάσεις είναι οι καταστάσεις στις οποίες δεν μπορούν να εφαρμοστούν άλλες τακτικές (tactics). Χαρακτηρίζουμε τις καταστάσεις του δένδρου αναζήτησής μας ως τερματικές ή μη τερματικές με βάση την ανατροφοδότηση που λαμβάνουμε από το περιβάλλον απόδειξης μετά την εφαρμογή του βήματος (τακτικής ή εντολής Coq) που δημιούργησε αυτή την κατάσταση. Στο σύνολο των τερματικών καταστάσεων μας συμπεριλαμβάνουμε επίσης τις καταστάσεις MAX_TIME_REACHED και MAX_NUM_TACTICS_REACHED που επιστρέφονται από το περιβάλλον απόδειξης 5.1.2. Αυτές οι καταστάσεις επιστρέφονται όταν έχει συμπληρωθεί το μέγιστο χρονικό όριο ή όταν έχει συμπληρωθεί ο μέγιστος αριθμός τακτικών. Συμπεριλαμβάνοντας αυτές τις καταστάσεις στο σύνολο των τερματικών καταστάσεων διευκόλυνε την υλοποίηση του τερματισμού της αναζήτησης για αυτές τις δύο περιπτώσεις.

Εδώ είναι μια λίστα με όλες τις τερματικές καταστάσεις:

- ALREADY_SUCCEEDED
- ALREADY_FAILED
- MAX_TIME_REACHED
- MAX_NUM_TACTICS_REACHED
- ERROR
- GIVEN_UP

- SUCCESS

Οι μη τερματικές καταστάσεις είναι "PROVING" και "Initializing". Το "Initializing" ήταν μια κατάσταση που προσθέσαμε ως μέρος της υλοποίησης MCTS για να δηλώσει ότι η αναζήτηση δεν είχε ακόμη ξεκινήσει για τον τρέχοντα κόμβο.

Επιλογή - Selection

Κατά το βήμα της επιλογής χρειάζεται να επιλέξουμε την κατάσταση(state) που πρέπει να επεκταθεί, μεταξύ όλων των καταστάσεων απόδειξης που είναι στα φύλλα του δέντρου αναζήτησης.

Στην υλοποίησή μας χρησιμοποιήσαμε τον τύπο Upper Confidence Bound for Trees (UCT) σε κάθε κόμβο, ο οποίος είναι μια παραλλαγή του τύπου UCB, προσαρμοσμένος για αναζήτηση σε δένδρα 6.3.

$$UCT(i) = \frac{w_i}{n_i} + \sqrt{2\frac{\ln N_i}{n_i}}$$

Όπου:

- w_i είναι η συνολική ανταμοιβή που προκύπτει από τον κόμβο i.
- n_i είναι ο αριθμός επισκέψεων στον κόμβο i.
- N_i είναι ο αριθμός επισκέψεων στον γονεϊκό κόμβοςi.
- Θέσαμε την παράμετρο εξερεύνησης C να είναι ίση με $\sqrt{2}$.

Επέκταση - Expansion

Σε αυτό το στάδιο, μία ή περισσότερες νέες καταστάσεις απόδειξης προστίθενται στο δένδρο. Αυτοί οι νέοι κόμβοι αντιπροσωπεύουν καταστάσεις απόδειξης που δημιουργήθηκαν εφαρμόζοντας μια συγκεκριμένη τακτική στην προηγούμενη κατάσταση (που αντιπροσωπεύεται από τον κόμβο φύλλο).

Στην υλοποίησή μας πρέπει να συλλέξουμε τις δυνατές τακτικές (ενέργειες) που μπορούν να εφαρμοστούν στη τρέχουσα κατάσταση απόδειξης. Οι δυνατές ενέργειες για την τρέχουσα κατάσταση επιστρέφονται από το μοντέλο μηχανικής μάθησης, αφού του δώσουμε ως είσοδο το περιβάλλον της απόδειξης, το local context και τους στόχους που μένουν να αποδειχθούν. Οι τακτικές επιστρέφονται μετά το κάλεσμα της μεθόδου `beam_search` του μοντέλου. Αυτή η μέθοδος καλεί τον αποκωδικοποιητή που δημιουργεί τις υποψήφιες τακτικές για την τρέχουσα κατάσταση. Ο αριθμός των τακτικών καθορίζεται από την παράμετρο `beam` που έχουμε αναφέρει παραπάνω.

Το δεύτερο μέρος της επέκτασης είναι να εφαρμόσουμε τις υποψήφιες τακτικές στην τρέχουσα κατάσταση και να δημιουργήσουμε τις νέες καταστάσεις. Αυτό γίνεται παρέχοντας τις τακτικές μία μία στο περιβάλλον αξιολόγησης (evaluation environment) 5.1.2 και λαμβάνοντας την ανατροφοδότηση για τις εφαρμοσμένες ενέργειες. Η ανατροφοδότηση περιέχει την πρόοδο της απόδειξης (PROVING, ERROR κλπ.) και μπορεί επίσης να περιλαμβάνει το περιβάλλον απόδειξης καθώς και τους εστιασμένους, μη εστιασμένους, ανασταλμένους και εγκαταλελειμμένους στόχους, ανάλογα με την πρόοδο. Η ανατροφοδότηση μπορεί επίσης να περιέχει τα σφάλματα του Coq, αν η απόδειξη έχει φτάσει σε σφάλμα.

Η ανατροφοδότηση που επιστρέφεται από το Coq μέσω του περιβάλλοντος αξιολόγησης χρησιμοποιείται για να δημιουργήσουμε τις νέες καταστάσεις.

Προσομοίωση - Simulation

Σε αυτό το στάδιο πραγματοποιούμε μια προσομοίωση ή εξάπλωση από έναν νέο κόμβο προς έναν τερματικό κόμβο ακολουθώντας μια τυχαία πολιτική. Ακολουθούμε την ίδια διαδικασία για την λήψη και την εφαρμογή των δυνατών τακτικών σε κάθε κατάσταση, όπως περιγράφεται στις προηγούμενες ενότητες 3.

Στην υλοποίηση της προσομοίωσης διατηρούμε τα scripts αποδείξεων που δημιουργούνται κατά τη διάρκεια της εξάπλωσης. Στην περίπτωση που όλοι οι στόχοι αποδειχθούν κατά τη διάρκεια της προσομοίωσης, το script

που περιέχει τις τακτικές που εφαρμόστηκαν στο μονοπάτι από τη ρίζα στον τερματικό κόμβο με κατάσταση "PROVED" είναι μια απόδειξη του θεωρήματος.

Οπισθοδιάδοση - Backpropagation

Σε αυτό το στάδιο ανανεώνουμε τους κόμβους που επισκεφθήκαμε κατά τη διάρκεια της προσομοίωσης 6.2 με την κατάλληλη ανταμοιβή. Κατευθυνόμαστε από τα φύλλα προς τη ρίζα ενημερώνοντας την τιμή και τον αριθμό επισκέψεων κάθε κόμβου κατά μήκος της διαδρομής.

Πολιτικές Ανταμοιβής

Η αξιολόγηση για το αν μια κατάσταση απόδειξης είναι "καλή" ή "κακή", ή πόσο κοντά είναι στην απόδειξη δεν είναι κάτι που μπορούμε να κάνουμε. Αν είχαμε τη δυνατότητα να το πραγματοποιήσουμε με μία τυπική διαδικασία θα σήμαινε ότι θα μπορούσαμε χρησιμοποιώντας την να φτάσουμε στην τελική απόδειξη του θεωρήματος που προσπαθούμε να αποδείξουμε.

Το στάδιο προσομοίωσης του MCTS στοχεύει να κάνει ακριβώς αυτό - να εφαρμόσει μια σειρά τακτικών με μια τυχαία πολιτική μετά από μια δεδομένη κατάσταση απόδειξης και να αξιολογήσει την κατάσταση απόδειξης βάσει αυτών των αποτελεσμάτων. Εφόσον δεν είμαστε σε θέση να αξιολογήσουμε μη τερματικές καταστάσεις που επιτεύχθηκαν κατά τη διάρκεια της προσομοίωσης, η πρώτη μας προσέγγιση είναι να περιμένουμε η προσομοίωση να φτάσει σε μία από τις τερματικές καταστάσεις και να τους αναθέσουμε την αντίστοιχη ανταμοιβή.

Η πρώτη πολιτική ανταμοιβής είναι:

- Ανταμοιβή +100 εάν έχουμε φτάσει σε μια κατάσταση που έχει status 'ALREADY_SUCCEEDED' ή 'SUCCESS'.
- Ανταμοιβή -1 εάν έχουμε φτάσει σε μια κατάσταση που έχει status που υποδηλώνει αποτυχία - ALREADY_FAILED, ERROR, GIVEN_UP.

Η παραπάνω πολιτική είναι πολύ απλή στην εφαρμογή της και προωθεί μια εξερεύνηση στους τερματικούς κόμβους, ψάχνοντας έτσι για βαθύτερες αποδείξεις. Επιπλέον, καταφέρνει να αποτρέπει την αναζήτηση από το να παγιδεύεται σε μη υποσχόμενους κλάδους, καθώς θα ανταμείβονται αρνητικά. Από την άλλη πλευρά, οι θετικές ανταμοιβές είναι πολύ σπάνιες. Επιπροσθέτως, μόλις φτάσουμε σε μια τερματική κατάσταση που θα επιστρέφει μια θετική ανταμοιβή έχουμε φτάσει ουσιαστικά στο τέλος της αναζήτησής μας. Το script απόδειξης από τη ρίζα στο τερματικό κόμβο είναι μια ακολουθία τακτικών που αποδεικνύουν το τρέχον θεώρημα.

Μια άλλη προσέγγιση για να αποθαρρύνουμε την εξερεύνηση των μη υποσχόμενων κλάδων ήταν η αρνητική ανταμοιβή όταν φτάναμε σε βαθύτερους κόμβους. Ο μέσος όρος των βημάτων στις αποδείξεις του CoqGym είναι 9.1 [46].

Η πολιτική ανταμοιβής μας βασισμένη στο βάθος εξερεύνησης είναι:

- Ανταμοιβή +100 εάν έχουμε φτάσει σε μια θέση που έχει κατάσταση 'ALREADY_SUCCEEDED' ή 'SUCCESS'.
- Ανταμοιβή -1 εάν έχουμε φτάσει σε μια θέση που έχει κατάσταση που υποδηλώνει αποτυχία - ALREADY_FAILED, ERROR, GIVEN_UP.
- Ανταμοιβή $-\lambda_d \cdot \lfloor \frac{D}{5} \rfloor$, $D \in \{5k \mid k \in \mathbb{Z}\}$ όπου λ_d είναι μια σταθερά στην οποία πειραματιστήκαμε και D είναι το βάθος που έχουμε φτάσει.

Η παραπάνω προσέγγιση δεν αντιμετωπίζει το ζήτημα των σπάνιων θετικών ανταμοιβών. Για να το κάνουμε αυτό, πρέπει να βρούμε έναν τρόπο να αξιολογούμε εάν η εφαρμογή μιας συγκεκριμένης τακτικής οδηγεί σε καλύτερη κατάσταση από την προηγούμενη.

Σε αυτό το σημείο αξίζει να αναφερθεί ότι στο Coq, όταν εφαρμόζεται μια τακτική, συνήθως εφαρμόζεται μόνο στον πρώτο στόχο στο προσκήνιο, και όχι σε όλους τους στόχους. Οι στόχοι καταχωρούνται σε μια στοίβα, και η πλειοψηφία των τακτικών εφαρμόζεται μόνο στον τρέχοντα στόχο στην κορυφή της στοίβας (τον "εστιασμένο" στόχο).

Όταν μια τακτική εφαρμόζεται σε έναν στόχο στο Coq, μπορεί να συμβούν διάφορα πράγματα βάσει της χρησιμοποιημένης τακτικής:

- Ο στόχος επιλύεται: Εάν η τακτική επιλύσει επιτυχώς τον στόχο, αυτός ο συγκεκριμένος υποστόχος εξαφανίζεται γιατί έχει αποδειχθεί.
- Ο στόχος μετασχηματίζεται: Εάν η τακτική επιλύσει μερικώς τον στόχο ή τον μετασχηματίζει, ο αρχικός υστόχος εξαφανίζεται και εμφανίζονται νέοι στόχοι, αντικατοπτρίζοντας την υπολειπόμενη εργασία που πρέπει να γίνει. Ένας στόχος μπορεί να αντικατασταθεί από πολλούς νέους στόχους αν η εφαρμοζόμενη τακτική τον διασπά σε μικρότερα τμήματα.
- Ο στόχος παραμένει αμετάβλητος: Εάν η τακτική δεν είναι εφαρμόσιμη ή δεν είναι επιτυχής, ο στόχος θα παραμείνει αμετάβλητος.
- Μεταβολή του πλαισίου (context): Η τακτική ενδέχεται επίσης να προσθέσει νέους ορισμούς, μεταβλητές ή υποθέσεις στο πλαίσιο (context), επηρεάζοντας την επόμενη απόδειξη.

Εδώ παρουσιάζουμε ένα παράδειγμα του πώς οι στόχοι μπορούν να μετασχηματιστούν με την εφαρμογή τακτικών.

Listing 3.1: Παράδειγμα μετασχηματισμού στόχων

```
Goal forall x y : nat, x + y = y + x.
Proof.
  intros x y.
  (*
     - `intros` moves universally quantified variables (x and y)
     - from the goal to the context.
     - The goal transforms to `x + y = y + x` without
     - the forall quantifiers.
  *)
  rewrite Nat.add_comm.
  (*
     - `rewrite` uses the provided lemma (`Nat.add_comm`)
     - to change the goal.
     - Here, it solves the goal, so no subgoals remain,
     - and we're done.
  *)
Qed.
```

Με αυτά κατά νου μπορούμε να δούμε ότι όταν εφαρμόζουμε μια τακτική κατά τη διάρκεια της αναζήτησής μας, εάν ο αριθμός των εστιασμένων στόχων μειώνεται, τότε η τακτική έχει επιλύσει έναν από τους στόχους, και έχουμε φτάσει σε μια πιο βελτιωμένη κατάσταση σε σχέση με την προηγούμενη στην οποία βρισκόμασταν.

Η πολιτική ανταμοιβής με βάση τη μείωση των εστιασμένων στόχων είναι:

- Ανταμοιβή +100 εάν έχουμε φτάσει σε μια κατάσταση με status 'ALREADY_SUCCEEDED' ή 'SUCCESS'.
- Ανταμοιβή -1 εάν έχουμε φτάσει σε μια κατάσταση με status που υποδηλώνει αποτυχία - ALREADY_FAILED, ERROR, GIVEN_UP.
- Επιστροφή ανταμοιβής $+\lambda_g$ κάθε φορά που ο αριθμός των εστιασμένων στόχων μειώνεται, όπου λ_g είναι μια σταθερά πάνω στην οποία πειραματιστήκαμε.

Σημειώνεται ότι δεν ανταμοίβουμε αρνητικά τις περιπτώσεις στις οποίες ο αριθμός των στόχων αυξάνεται ή παραμένει αμετάβλητος, καθώς αυτό δεν είναι απαραίτητα ένα αρνητικό αποτέλεσμα.

Chapter 4

Πειράματα

Σε αυτό το κεφάλαιο θα αναφερθούμε στα αποτελέσματα των πειραμάτων μας στον συνδυασμό του MCTS με τα υπάρχοντα μοντέλα βαθιάς μάθησης που δημιουργούν τακτικές ως προγράμματα. Αρχικά θα παρουσιάσουμε τη διάταξη των πειραμάτων μας και στη συνέχεια θα αναλύσουμε τα αποτελέσματα που λάβαμε για κάθε μία από τις πολιτικές ανταμοιβής που εξετάσαμε στο προηγούμενο κεφάλαιο.

Setup

Μοντέλα

Πραγματοποιήσαμε τα πειράματά μας χρησιμοποιώντας τα εκπαιδευμένα μοντέλα ASTactic και TacTok που ήταν διαθέσιμα δημόσια. Τα μοντέλα Passport δεν ήταν διαθέσιμα δημόσια, επομένως έπρεπε να τα εκπαιδεύσουμε μόνοι μας. Εκπαιδεύσαμε το Passport με τα ASTactic, Tac και Tok, έτσι ώστε να είμαστε σε θέση να συγκρίνουμε την απόδοση του MCTS συνδυασμένου με τα μοντέλα αυτά. Τα λεπτομερή αποτελέσματα της εκτέλεσης των μοντέλων με την αναζήτηση DFS παρουσιάζονται στην ενότητα 8.9.

Benchmark

Όπως περιγράφεται στο 3.2, το CoqGym περιλαμβάνει 123 Coq projects ανοιχτού κώδικα, χωρισμένα σε τρία σύνολα. Για την αξιολόγησή μας, εκπαιδεύσαμε σε 97 από αυτά (περιλαμβάνοντας συνολικά 57.719 θεωρήματα) και συνθέσαμε αποδείξεις για 26 (περιλαμβάνοντας συνολικά 10.782 θεωρήματα). Ακολουθούμε τους First et al. [11] και Sanchez-Stern, A. et al. [33] και εξαιρούμε το σύνολο coq-library-undecidability από την αξιολόγηση. Όπως αναφέρεται από τους Sanchez-Stern, A. κ.α. [33] "Η αξιολόγηση του TacTok [First et al. 2020] δεν μπόρεσε να αναπαράγει τα προηγούμενα αποτελέσματα για την απόδοση του ASTactic [Yang and Deng 2019] [46] σε αυτό το έργο λόγω εσωτερικών σφαλμάτων του Coq κατά την επεξεργασία των σεναρίων απόδειξης".

Υλικό

Χρησιμοποιήσαμε GPUs για την εκπαίδευση των μοντέλων και CPUs για τη σύνθεση αποδείξεων. Είχαμε την τύχη η διαδικασία σύνθεσης αποδείξεων να είναι εφικτή σε CPUs καθώς, όπως αναφέρεται από τον Yang et al. [46], το υπολογιστικό bottleneck βρίσκεται στην αλληλεπίδραση με το Coq και την εκτέλεση των τακτικών, όχι στην παραγωγή τους.

Για την εκπαίδευση του Passport χρησιμοποιήσαμε μηχάνημα με μια GPU NVIDIA T4 με 16 GB μνήμης.

Παράμετροι

Στα πειράματά μας χρησιμοποιήσαμε τις πειραματικές παραμέτρους που ακολούθησαν οι Yang και Deng 2019 [46], First et al. [11] και Sanchez-Stern, A. et al. [33].

Συνθέτουν αποδείξεις ορίζοντας ένα χρονικό όριο, μετά το πέρας του οποίου, αν η αναζήτηση αποτύχει να φτάσει στο Qed, θεωρούν πως η απόδειξη απέτυχε. Έχουν ορίσει το χρονικό όριο σε 10 λεπτά, και αυτή είναι η τιμή που χρησιμοποιήσαμε στα πειράματά μας.

Για την εκπαίδευση του Passport χρησιμοποιήσαμε επίσης τις παραμέτρους που περιγράφονται από τους Sanchez-Stern, A. et al. [33].

Ορίσαμε:

- default category vocabulary threshold: 200
- byte-pair merge threshold: 4096
- default vector dimension for term, grammar and terminal/non-terminal symbol embeddings: 128
- dimension of LSTM controller: 128

Καθορισμός Ακτίνας αναζήτησης - Beam Size

Λόγω του τεράστιου χρονικού κόστους της διαδικασίας πειραμάτων, δεν ήμασταν σε θέση να δοκιμάσουμε μεγάλο εύρος βάθους αναζήτησης σε ολόκληρο το σύνολο δοκιμών. Πραγματοποιήσαμε δοκιμές για βάθη 20, 25 και 30. Αποφασίσαμε να διαλέξουμε projects με υψηλό αριθμό αποδείξεων στις οποίες τα μοντέλα είχαν είτε πολλές επιτυχημένες είτε πολύ λίγες επιτυχημένες αποδείξεις. Τα projects που επιλέξαμε ήταν: dblib, UnifySL, PolTac και verdi-raft.

Κατά τη διάρκεια των πειραμάτων είδαμε ότι η αύξηση του ακτίνας σε 25 στο MCTS έδωσε βελτιωμένα αποτελέσματα σε σύγκριση με τα αποτελέσματα με ακτίνα αναζήτησης 20. Η αύξηση του δέσμης σε 30 δεν πρόσθεσε καμία αξία. Αυτό είναι πιθανώς ένδειξη του γεγονότος ότι οι επιπλέον τακτικές που προτείνονται από τα μοντέλα δεν είναι αρκετά ακριβείς για να παράγουν περισσότερες αποδείξεις. Τα παρακάτω αποτελέσματα παρουσιάζονται χρησιμοποιώντας ακτίνα αναζήτησης 25.

Πολιτική Ανταμοιβής σε Τερματικές Καταστάσεις

Κατά την αξιολόγηση των μοντέλων που εκπαιδεύσαμε για το Passport, δεν ήμασταν σε θέση να αναπαραγάγουμε τα ακριβή αποτελέσματα που αναφέρονται από τους Sanchez-Stern, A. et al. [33]. Για αυτό τον λόγο, θα συγκρίνουμε τα αποτελέσματά μας με τα αποτελέσματα που καταφέραμε να αναπαραγάγουμε, αλλά θα παρουσιάσουμε και εκείνα που αναφέρονται στη βιβλιογραφία.

Σύγκριση MCTS & DFS μεταξύ των μοντέλων

Στο σχήμα 4.0.1 μπορούμε να δούμε την απόδοση του MCTS που εκτελέστηκε με την πολιτική "ανταμοιβή σε τερματικές καταστάσεις" 7.1.6.

Παρατηρούμε ότι όταν εφαρμόζουμε το MCTS στο ASTactic και στο TacTok (τα μοντέλα Tac & Toc συνδυασμένα), το DFS υπερτερεί της μεθόδου αναζήτησής μας, αλλά όχι σημαντικά.

Όταν συγκρίνουμε MCTS και DFS σε συνδυασμό με τα μοντέλα του Passport, βλέπουμε ότι το DFS υπερτερεί της αναζήτησής μας σύμφωνα με τα αποτελέσματα της της βιβλιογραφίας [33]. Από την άλλη πλευρά, η MCTS υπερτερεί της DFS όταν συγκρίνουμε τα αποτελέσματα από τα μοντέλα Passport που εκπαιδεύσαμε εμείς. Επιπλέον παρατηρούμε ότι συνδυάζοντας και τα τρία μοντέλα η MCTS κατάφερε να αποδείξει 29 περισσότερα θεωρήματα από το DFS.

Νέα θεωρήματα που αποδεικνύονται με MCTS

Θα ακολουθήσουμε την προσέγγιση που υιοθέτησαν οι First et al. [11] και οι Sanchez-Stern, A. et al. [33] για να αξιολογήσουμε πώς η εφαρμογή της MCTS στις προτάσεις των μοντέλων οδηγεί στην απόδειξη νέων θεωρημάτων.

Στον πίνακα 4.1 βλέπουμε ότι όταν υπολογίζουμε την πληθικότητα της ένωσης των αποδείξεων που συνθέτονται από όλα τα μοντέλα που ενισχύονται από το Passport σε συνδυασμό με το MCTS, έχουμε 33 νέες αποδείξεις.

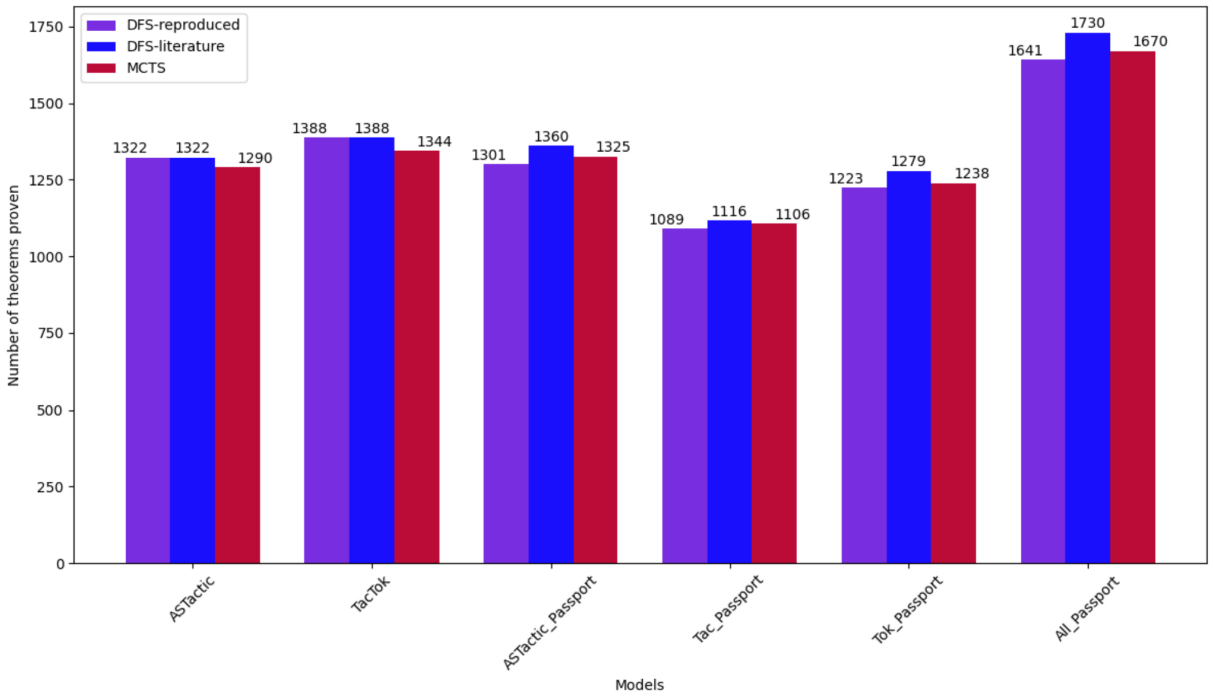


Figure 4.0.1: Ο αριθμός των θεωρημάτων που αποδεικνύονται για κάθε μοντέλο χρησιμοποιώντας DFS και MCTS με την πολιτική ανταμοιβής σε τερματικές καταστάσεις. Οι μώβ ιστοί αντιπροσωπεύουν τα βασικά μοντέλα που βασίζονται στο ASTactic, TacTok, ASTactic + Passport, Tok + Passport και Tac + Passport. Οι μπλε ιστοί αντιπροσωπεύουν τα αποτελέσματα που βρίσκουμε στη βιβλιογραφία [33], [11]. Οι κόκκινοι ιστοί αντιπροσωπεύουν τα αποτελέσματα της αξιολόγησης των μοντέλων με MCTS. Οι ιστοί με την ετικέτα "AllPassport", είναι ο αριθμός των θεωρημάτων που αποδεικνύονται επιτυχώς χρησιμοποιώντας είτε DFS είτε MCTS από τουλάχιστον ένα από τα μοντέλα που ενισχύονται από το Passport.

Όταν συγκρίνουμε τον αριθμό των αποδείξεων που συνθέτονται από το ASTactic σε συνδυασμό με το MCTS, καταφέραμε να αποδείξουμε 8 νέα θεωρήματα παρά το γεγονός ότι χάσαμε κάποιες αποδείξεις που βρέθηκαν με το DFS. Παρατηρούμε επίσης ότι στο μοντέλο ASTactic+Passport, καταφέραμε να αποδείξουμε 27 νέα θεωρήματα, αν και αποδείξαμε μόνο 24 περισσότερα θεωρήματα σε απόλυτες τιμές. Αυτό δείχνει και πάλι ότι χάσαμε μερικές από τις αποδείξεις που αποδείχθηκαν με τη χρήση DFS.

Τα αποτελέσματα που παίρνουμε με την εφαρμογή MCTS με πολιτική 'ανταμοιβή σε τερματικές καταστάσεις' στα υπάρχοντα μοντέλα είναι πολύ παρόμοια με τα αποτελέσματα που παίρνουμε από το DFS. Αυτό δεν αποτελεί έκπληξη. Η σπανιότητα των θετικών ανταμοιβών που χαρακτηρίζει αυτή την πολιτική, καθώς και το γεγονός πως όταν τελικά θα λάβουμε την θετική ανταμοιβή θα έχουμε αποδείξει όλους τους στόχους, εκφυλίζουν την MCTS σε παραδοσιακό αλγόριθμο αναζήτησης.

Μοντέλο	ASTactic	TacTok	ASTactic+Passport	Tac+Passport	Tok+Passport	*+Passport
DFS	1322	1388	1301	1089	1223	1641
MCTS	1290	1344	1325	1106	1238	1670
Διαφορά	-32	-44	+24	+27	+15	+29
Νέα Θεωρήματα	8 (0.07%)	0 (0%)	27 (0.25%)	18 (0.17%)	16 (0.15%)	33 (0.31%)

Table 4.1: Αριθμός θεωρημάτων που αποδεικνύονται στα διάφορα μοντέλα με το MCTS όταν χρησιμοποιείται η πολιτική ανταμοιβής στις τερματικές καταστάσεις και ακτίνα αναζήτησης (beam) 25. Τα ποσοστά υπολογίζονται σε σχέση με το συνολικό αριθμό αποδείξεων 10782 των δεδομένων αξιολόγησης (evaluation dataset).

Όπως αναφέρεται στη βιβλιογραφία [46], το μέσο μήκος των αποδείξεων που αποδεικνύουν τα μοντέλα είναι μικρό - λιγότερο από 10 τακτικές. Το μέσο μήκος των θεωρημάτων που αποδείχτηκαν σε διάφορα μοντέλα με το MCTS ήταν 6.7. Συγκριτικά, το μέσο μήκος αποδείξεων στα δεδομένα αξιολόγησης (evaluation dataset) είναι 12.5. Αυτό δεν μας εκλπήσσει - καθώς επιβεβαιώνει την προσδοκία μας πως όσο πιο μακρύ είναι το θεώρημα, τόσο δυσκολότερο θα είναι να δημιουργήσουμε την απόδειξή του. Το μέσο μήκος των 33 νέων θεωρημάτων που αποδείχτηκαν ήταν 7.8 τακτικές. Αυτό είναι λίγο υψηλότερο από το μέσο μήκος όλων των αποδείξεων που αναφέρουμε παραπάνω. Αυτό υποδηλώνει ότι το MCTS διευκολύνει την απόδειξη μεγαλύτερων θεωρημάτων.

Πολιτική Ανταμοιβής βάσει του Βάθους Αναζήτησης

Καθορισμός Παραμέτρων Πολιτικής

Προκειμένου να καθορίσουμε την τιμή του λ_d ακολουθήσαμε την ίδια προσέγγιση που χρησιμοποιήσαμε για τον καθορισμό του της ακτίνας αναζήτησης (beam) 8.2. Διεξάγαμε δοκιμές για $\lambda_d \in \{0.5, 0.75, 1\}$ στα σύνολα αποδείξεων dblib, UnifySL, PolTac και verdi-raft.

Τα αποτελέσματα των πειραμάτων των αποδείξεων σε αυτά τα σύνολα δεν είχαν σημαντικές διαφορές για τις διαφορετικές τιμές του λ_d. Επιλέξαμε να χρησιμοποιήσουμε την τιμή $\lambda_d = 0.5$ για να μην τιμωρούμε υπερβολικά την αναζήτηση σε βαθύτερο χώρο.

Σύγκριση MCTS & DFS μεταξύ των μοντέλων

Στο σχήμα 4.0.2 μπορούμε να δούμε την απόδοση της MCTS στην οποία εφαρμόσαμε την πολιτική "ανταμοιβής βάσει του βάθους αναζήτησης". 7.1.6.

Παρατηρούμε πως όταν εφαρμόζουμε MCTS στο ASTactic και το TacTok, η DFS αποδίδει καλύτερα από την μέθοδο αναζήτησης μας, αποδεικνύοντας 65 περισσότερα θεωρήματα.

Όταν συγκρίνουμε MCTS και DFS σε συνδυασμό με τα μοντέλα Passport, βλέπουμε ότι η DFS και πάλι αποδίδει καλύτερα από την αναζήτησή μας σύμφωνα με τα αποτελέσματα της βιβλιογραφίας [33], αποδεικνύοντας 138 περισσότερα θεωρήματα. Η DFS επίσης έχει καλύτερη απόδοση όταν συγκρίνουμε με τα αποτελέσματα των πειραμάτων που πήραμε από τα μοντέλα του Passport που εμείς εκπαιδεύσαμε. Μπορούμε να δούμε ότι με τη DFS και τα τρία μοντέλα συνδυασμένα μπόρεσαν να αποδείξουν 49 περισσότερα θεωρήματα απ' ό,τι με το MCTS.

Νέα θεωρήματα που αποδεικνύονται με MCTS

Στον πίνακα 4.2 βλέπουμε ότι όταν υπολογίζουμε την πληθικότητα της ένωσης των αποδείξεων που συντάχθηκαν από όλα τα μοντέλα ενισχυμένα με το Passport σε συνδυασμό με το MCTS, έχουμε 17 νέες αποδείξεις.

Δεν πήραμε καμία νέα απόδειξη όταν συνδυάσαμε το MCTS με το ASTactic και το TacTok.

Μοντέλο	ASTactic	TacTok	ASTactic+Passport	Tac+Passport	Tok+Passport	*+Passport
DFS Repro	1322	1388	1301	1089	1223	1641
MCTS Repro	1257	1282	1244	1027	1151	1592
Διαφορά	-65	-106	-57	-62	-72	-49
Νέα Θεωρήματα	0	0	14 (0.13%)	11 (0.1%)	11 (0.1%)	17 (0.16%)

Table 4.2: Αριθμός θεωρημάτων που αποδεικνύονται στα διάφορα μοντέλα με το MCTS όταν χρησιμοποιείται η πολιτική ανταμοιβής βάσει του βάθους αναζήτησης και ακτίνα αναζήτησης (beam) 25. Τα ποσοστά υπολογίζονται σε σχέση με το συνολικό αριθμό αποδείξεων 10782 των δεδομένων αξιολόγησης (evaluation dataset).

Η εφαρμογή της πολιτικής ανταμοιβής βάσει του βάθους αναζήτησης για ακτίνα 25 δεν βελτίωσε τα αποτελέσματα της αναζήτησής μας. Εικάζουμε πως αυτό συνέβη για τους ακόλουθους λόγους:

- Η πρώτη παρατήρηση είναι ότι οι αρνητικές ανταμοιβές βάσει του βάθους πιθανώς να εμποδίζουν την ανακάλυψη βαθύτερων αποδείξεων. Όπως έχουμε δει παραπάνω, ο μέσος όρος του μήκους των αποδείξεων

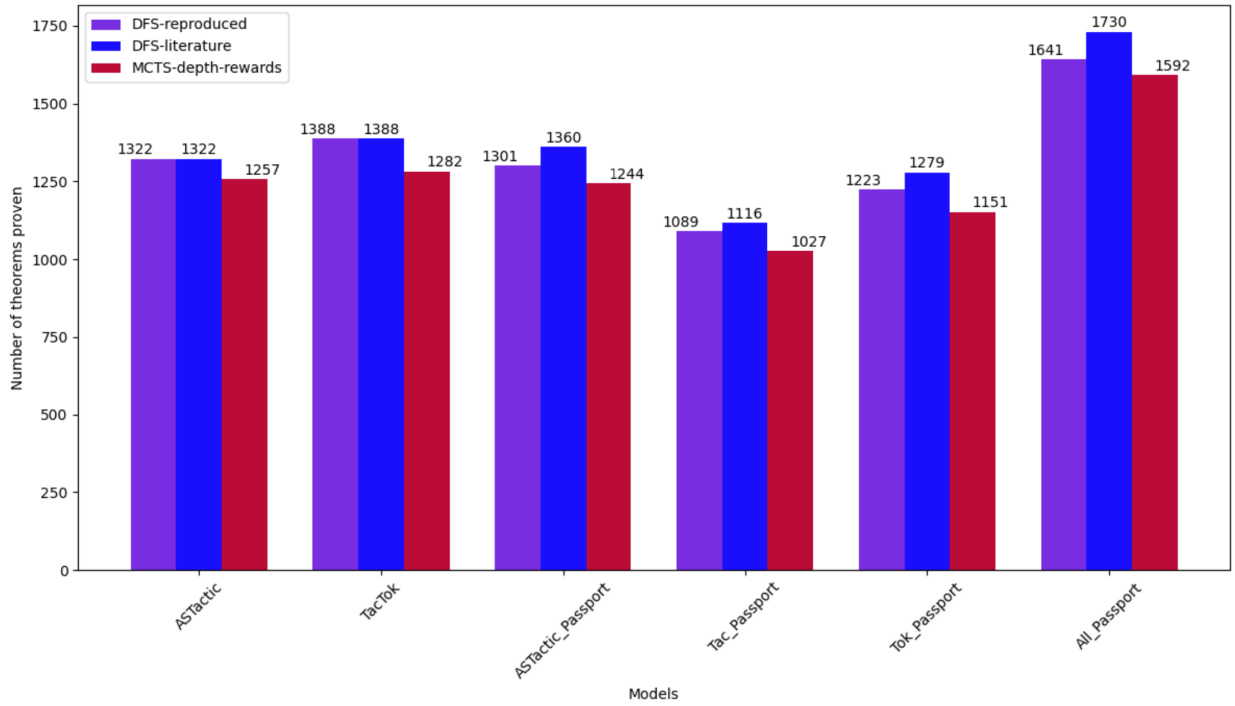


Figure 4.0.2: Αριθμός θεωρημάτων που αποδείχθηκαν για κάθε μοντέλο χρησιμοποιώντας DFS και MCTS με πολιτική Ανταμοιβής βάσει του Βάθους Αναζήτησης. Οι μώβ ιστοί αντιπροσωπεύουν τα βασικά μοντέλα βασισμένα στο ASTactic, TacTok, ASTactic + Passport, Tok + Passport και Tac + Passport. Οι μπλε ιστοί αντιπροσωπεύουν τη βασική αξιολόγηση που αναφέρεται στη βιβλιογραφία [33], [11]. Οι κόκκινοι ιστοί αντιπροσωπεύουν τα αποτελέσματα της εκτέλεσης των μοντέλων με MCTS. Οι ιστοί με την ετικέτα "AllPassport" είναι ο αριθμός των θεωρημάτων που αποδείχθηκαν επιτυχώς χρησιμοποιώντας είτε DFS είτε MCTS από τουλάχιστον ένα από τα μοντέλα που ενισχύονται από το Passport.

που παράγονται ανά μοντέλο είναι 6.7 τακτικές ανά απόδειξη. Αυτό δείχνει ότι τα μοντέλα είναι πιο αποτελεσματικά στην σύνθεση μικρών αποδείξεων και ότι η αναζήτηση DFS είναι ικανή να εντοπίσει αποτελεσματικά αυτές τις αποδείξεις, δεδομένων των παραμέτρων αναζήτησης που καθορίστηκαν (ακτίνα αναζήτησης 20, χρονικό όριο 10 λεπτών). Η αρνητική ανταμοιβή σε βαθύτερες καταστάσεις δεν βελτίωσε την ανακάλυψη νέων συντομότερων αποδείξεων και απέτρεψε την αναζήτηση από το να βρει μακρύτερες αποδείξεις που ανακαλύφθηκαν από την προηγούμενη πολιτική που εφαρμόσαμε.

- Η δεύτερη παρατήρηση είναι ότι το χρονικό κόστος των πειραμάτων μας εμπόδισε να βελτιστοποιήσουμε τις παραμέτρους του MCTS σε συνδυασμό με την πολιτική ανταμοιβής βάσει του βάθους αναζήτησης. Είναι πιθανό ότι αν είχαμε πειραματιστεί με περισσότερους συνδυασμούς των παραμέτρων λ_d και $beam$, θα μπορούσαμε να έχουμε καλύτερα αποτελέσματα. Για παράδειγμα, αυξάνοντας την παράμετρο λ_d σε τιμές μεγαλύτερες από 1 και επεκτείνοντας την παράμετρο $beam$ σε τιμές μεγαλύτερες από 30, ίσως να ενθαρρύναμε την ανακάλυψη νέων συντομότερων αποδείξεων, παρόλο που θα εμποδίζαμε την ανακάλυψη βαθιών αποδείξεων.

Πολιτική Ανταμοιβής με βάση τη μείωση των στόχων

Καθορισμός Παραμέτρων Πολιτικής

Αυτή ήταν η τελευταία πολιτική που δοκιμάσαμε και δεν είχαμε χρόνο να ολοκληρώσουμε τα πειράματά μας μέχρι τη στιγμή που ολοκληρώθηκε αυτή η διπλωματική.

Για να καθορίσουμε την τιμή του λ_g ακολουθούμε την ίδια προσέγγιση που χρησιμοποιήσαμε για τον καθορισμό της δέσμης 8.2 και του λ_d. Εκτελούμε δοκιμές για $\lambda_g \in \{0.5, 1, 2\}$ στα σύνολα αποδείξεων dblib, UnifySL,

PolTac και verdi-raft. Τα αποτελέσματα που πήραμε από αυτά τα πειράματα ήταν σημαντικά (γύρω στο 20%) χειρότερα από τα αποτελέσματα που πήραμε από τις προηγούμενες πολιτικές ανταμοιβής που χρησιμοποιήσαμε στο MCTS. Μια εξήγηση γι' αυτό είναι ότι κατά τη φάση της προσομοίωσης, πολλές καταστάσεις ανταμείβονται θετικά ως αποτέλεσμα των τακτικών που μειώνουν τον αριθμό των εστιασμένων στόχων στα υποδέντρα τους, αλλά δεν οδηγούν κλαδιά που βρίσκεται η απόδειξη του δέντρου.

Επιπλέον, αξίζει να αναφερθεί πως με αυτήν την πολιτική προσπαθούμε να αναγνωρίσουμε το θετικό αποτέλεσμα της εφαρμογής μιας τακτικής σε μια δεδομένη κατάσταση (state) αλλά ανταμείβουμε τις προκύπτουσες καταστάσεις και όχι τις τακτικές. Αυτό μπορεί να οδηγήσει σε συνθήκες όπου οι καταστάσεις σε μη υποσχόμενα παρακλάδια της αναζήτησης θα ανταμείβονται υψηλά κατά τη διάρκεια της αναζήτησης.

Πιθανές Βελτιώσεις της Πολιτικής Ανταμοιβής

Θα χρειαζόμασταν περισσότερο χρόνο για να βελτιώσουμε τις παραμέτρους της πολιτικής μας (λ_g, τιμές θετικών ανταμοιβών, τιμές αρνητκών ανταμοιβών) και την ακτίνα αναζήτησης (beam) ώστε να εκμεταλλευτούμε τα πιθανά οφέλη αυτής της πολιτικής. Ήταν κάτι που δεν είχαμε χρόνο να κάνουμε λόγω του υπολογιστικού κόστους των πειραμάτων.

Μία προσέγγιση που θα μπορούσε επίσης να βελτιώσει την αναζήτησή μας είναι να συνδυάσουμε το MCTS με μια προσωρινή μνήμη "καλών" τακτικών που προτάθηκαν από τα μοντέλα - τακτικές που έχουν συμβάλει στη μείωση των εστιασμένων στόχων κατά την αναζήτηση. Οι τακτικές αυτές θα δειγματοληπτούνται και θα ανταμείβονται κατά τη διάρκεια της αναζήτησης. Ένα επιπλέον βήμα σε αυτό θα ήταν να έχουμε μια προσωρινή μνήμη για κάθε επικεντρωμένο στόχο του θεωρήματος.

Συμπεράσματα

Στα πειράματά μας δοκιμάσαμε την αντικατάσταση των παραδοσιακών μεθόδων αναζήτησης στα εργαλεία πρόβλεψης μηχανικής μάθησης για καθοδηγούμενη σύνθεση αποδείξεων με Monte Carlo Tree Search - MCTS. Τα πειράματά μας έδειξαν ότι η χρήση της MCTS μπορεί να παράγει αποτελέσματα πολύ κοντά στα αποτελέσματα της βιβλιογραφίας. Επιπλέον, τα πειράματά μας παρήγαγαν αποδείξεις που δεν παρήχθησαν από τις παραδοσιακές μεθόδους αναζήτησης. Ωστόσο, δεν καταφέραμε να υπερβούμε τα καλύτερα αποτελέσματα που αναφέρονται στη βιβλιογραφία.

Από τις τρεις πολιτικές ανταμοιβής που χρησιμοποιήσαμε η πολιτική ανταμοιβής στις τερματικές καταστάσεις ήταν η πιο αποτελεσματική. Όταν συνδυάστηκε με τα μοντέλα Passport που εμείς εκπαιδεύσαμε, το MCTS κατάφερε να βρει περισσότερες αποδείξεις σε σύγκριση με αυτές που πήραμε με την DFS. Παρ' όλα αυτά, δεν κατάφερε να υπερβεί τα καλύτερα αποτελέσματα που αναφέρονται στη βιβλιογραφία. Η έλλειψη θετικών ανταμοιβών σε αυτή την πολιτική κάνει το MCTS να συμπεριφέρεται πολύ παρόμοια με τους παραδοσιακούς αλγορίθμους αναζήτησης όπως το DFS, δίνοντάς μας πολύ παρόμοια αποτελέσματα.

Προσπαθήσαμε να εκμεταλλευτούμε τα οφέλη του MCTS εισάγοντας τις πολιτικές "ανταμοιβή βάσει του βάθους αναζήτησης" και "ανταμοιβή βάσει της μείωσης στόχων". Η πολιτική "ανταμοιβή βάσει του βάθους αναζήτησης" μας έδωσε θετικά δείγματα, αλλά δεν ήταν τόσο αποτελεσματική στην ανακάλυψη μεγαλύτερων αποδείξεων. Η δοκιμασία της πολιτικής "ανταμοιβή βάσει της μείωσης στόχων" είναι ακόμη σε εξέλιξη καθώς ολοκληρώνεται αυτής η διπλωματική εργασία, καθώς η ρύθμισή της ήταν πιο δύσκολη από τη ρύθμιση παραμέτρων στις προηγούμενες μεθόδους.

Περιορισμοί

Ο κύριος περιορισμός που είχαμε στα πειράματά μας ήταν το χρονικό κόστος εκτέλεσής τους. Η εκτέλεση των πειραμάτων στο σύνολο των δεδομένων αξιολόγησης (evaluation dataset) για ένα σύνολο παραμέτρων σε όλα τα 26 σύνολα δεδομένων του Coq διαρκούσε αρκετές εβδομάδες με τον υλικό (hardware) που είχαμε διαθέσιμο. Για το λόγο αυτό, όπως αναφέρουμε και προηγούμενα, καταφέραμε να δοκιμάσουμε μόνο ένα περιορισμένο εύρος πιθανών τιμών για τις διάφορες παραμέτρους. Επιπλέον, πραγματοποιήσαμε τις δοκιμές των παραμέτρων αυτών σε ένα υποσύνολο των δεδομένων αξιολόγησης. Επομένως, δεν μπορούμε να είμαστε βέβαιοι για την βελτιστότητά τους.

Λόγω της έλλειψης hardware, δεν μπορέσαμε επίσης να πειραματιστούμε με τα μοντέλα και να τα εκπαιδεύσουμε εκ νέου. Το κόστος του υλικού αναφέρεται επίσης από τους περισσότερους ερευνητές που εργάζονται στον τομέα Sanchez-Stern, A. et al. [33] ως ένας οριακά απαγορευτικός περιορισμός.

Κείμενο στα αγγλικά

Chapter 1

Introduction

Theorem proving has stood as the cornerstone of scientific progress. In political debates winners are not always the politicians with the most concrete arguments and the law does not consistently require indisputable evidence for a verdict to be determined. But in mathematics, unlike our everyday lives or areas like law or politics, arguments and propositions need to be indisputable. The only way to evaluate the validity of a mathematical statement is through the rigorous task of constructing a mathematical proof, a convincing logical argument which demonstrates the truth of a proposition given a set of assumptions. Scientists through the millennia have created several proof techniques. Some examples are proof by construction, proof by induction, proof by contradiction, proof by exhaustion, proof by counter example and more.

Creating a machine that can verify the validity of a theorem and provide its proof with the press of a button has been an object of research and experimentation for many mathematicians and computer scientists through the ages. This has led to the evolution of the field of automated theorem proving or automated proof synthesis which in the context of logic and computer science involves automatically generating a proof for a given statement, provided that the statement is indeed provable. Unlike proof checking, which verifies the correctness of a given proof, proof synthesis is the process of actually deriving a proof, potentially without human intervention.

1.1 Motivation

The advances in the area of Artificial Intelligence and particularly in Neural Networks and Machine Learning could not bypass the field of Automated Theorem Proving (ATP). There have been significant research attempts to apply those advances to the fields of Automated Proof Synthesis [37] and Interactive Theorem Proving. Researchers have collected and structured datasets of significant sizes ([16], CoqGym [46]) which contain human written and auto generated proofs, and have tried to train models which will either compose entire proofs or interact with proof assistance and compose proof scripts of the theorems to be proved. These data have been used to create models which initially were based on Recurrent Neural Network architectures ([46]) and have no shifted to incorporate Large Language models and Transformers [47].

In this thesis we will focus on the domain of Interactive Theorem Proving and specifically on the models trained on the CoqGym data set which contains proofs of the Coq Proof Assistant.

1.2 Contribution

The principal proposal of this thesis revolves around enhancing the existing architectures of machine learning prediction guided proof synthesis tools built for the CoqGym dataset with more elaborate proof search methods and specifically with Monte Carlo Tree Search.

To that end we implemented and tested MCTS on the architectures of ASTactic [46], TacTok [11] and Passport [33] which are tactic prediction models for Coq Proofs trained on CoqGym dataset [46]. We introduce three

reward policies for our MCTS implementation:

1. the "reward on terminal states" policy
2. the "reward on depth" policy
3. the "reward on goal reduction" policy.

1.3 Thesis Structure

In this thesis we start by presenting to the reader the problem of proof synthesis and automated proof synthesis as well as the domain of interactive theorem proving in chapter 2. We continue by presenting the interactive theorem prover Coq 3 and then try to briefly familiarize the reader with the theoretical background on Long Short Term Memory networks 4. Chapter 5 presents the architectures three machine learning prediction models for guided proof synthesis: ASTactic [46], Passport [33] and TacTok [11]. In chapter 6 we describe the Monte Carlo Tree Search algorithm and in chapter 7 we describe how we propose to use it with the above prediction models for guided proof synthesis. Lastly, in chapter 8 we conclude this thesis by presenting and analyzing the results of our experiments, discussing the limitations that we encountered and proposing ideas and opportunities for future work.

Chapter 2

Theroetical Background on Proof Synthesis

2.1 Automated Theorem Proving (ATP)

Automated theorem proving (ATP) [Bibel 2013; Fitting 2012; Pfenning 2004] is a set of techniques that prove logical formulas automatically.

ATP is a tool used in formal mathematics to produce mathematical proofs whose validity can be then checked by experts [5]. Additionally, it has wide application beyond mathematics, in areas like system verification, where one states the correctness of a system as a theorem and justifies it in the form of proofs [37].

2.1.1 Forms of ATP

There are two main forms of ATP:

- Automated proof synthesis (APS) [37]:

  Given a logical formula P, if P holds, return a proof M of P. In the light of the importance of theorem proving, APS serves as a useful tool for activities based on formal reasoning. For example, from the perspective of the aforementioned system verification, APS serves for automating formal system verification. Various methods for (semi)automated static program verification, essential for proving the correctness and safety of high-stakes applications [22], can be seen as APS procedures.

- Interactive theorem proving (ITP)

  In ITP, proofs are constructed by human experts interacting with software tools called proof assistants, such as Coq [2], Isabelle [29], and Lean [28]. Proof assistants allow users to state theorems and their proofs formally in the form of certain programming languages and automatically check that the proofs correctly prove the theorems [37]. The interaction with the human becomes essential due to the enormous search space that needs to be explored by the machine in order to fully generate a correct proof. Machine learning can automate such interactive theorem proving, opening up a new avenue for theorem proving [46]. The model can learn to interact with proof assistants, given data containing human-written proofs.

2.1.2 A look at APS methodologies

Automated proof synthesis for implicational logics

The traditional approach taken by the programming-language community to tackle APS has been with the use of symbolic methods. An APS algorithm inspects the syntactic structure of a formula P and, using the obtained information, tries to construct a proof derivation of P [37]. A seminal work in this regard is

by Ben-Yelles [1979][3] in which a sound and complete APS algorithm for an implicational fragment of the propositional logic is proposed. This algorithm, given a formula of implicational logic, determines the set (possibly empty) of λ - terms in long β - normal form which have a given formula as a type.

An implicational logic can be specified by the rule

$$\frac{\alpha \to \beta \qquad \alpha}{\beta}$$

and a set of axiom schemes. Getting into more detail on this topic is not on the goals of this thesis. There have been several advances on this domain and improvements on the Ben-Yelles work [17]. Furthermore, similar algorithms have been implemented and used by Hammers, which are tools that provide general purpose automation for formal proof assistants [8].

Automated proof synthesis with machine learning

The first work of Sekiyama and Suenaga views the APS problem as a machine translation problem from the language of logical formulas to the language of proofs [36]. Their second work introduces the use of statistical machine learning. They experiment with the application of deep neural networks (DNN), introducing the DNN architecture named proposition-to-proof model tailored to the APS problem [37].

In short, they created a statistical model of APS problem in terms of probabilities, which serves for quantifying how a partially constructed proof is likely to lead to a correct proof of the given proposition P. Based on that, they defined a proof-synthesis procedure that searches for a proof of given proposition P in the order of the likelihood. This proof synthesis procedure requires a function to estimate the likelihood of an inference rule being applied at a specific step of a proof (or, equivalently, a specific position of a partially constructed proof). For this estimation, they used DNN based on the aforementioned proposition-to-proof architecture.

The performance evaluation of their network revealed that it can predict the inference rules which fill a partially constructed proof of a propositional-logic formula with 96.79% accuracy.

2.1.3 A look at ITP methodologies

As mentioned above, Interactive theorem proving (ITP) is a method of formal verification in computer science and mathematics that combines human-guided reasoning with the assistance of computer software. It involves the interaction between a human user (typically a mathematician or software engineer) and a computer-based tool, known as a proof assistant or interactive theorem prover.

A large part of the process of proof formalisation consists of providing justifications for smaller goals [9]. And although mathematicians would not face any serious complications when tackling these goals, users of modern interactive theorem provers have to spend an important part of the formalisation effort on them. This effor usually includes:

1. library search,
2. minor transformations on the already proved theorems (such as reordering assumptions or reasoning modulo associativity-commutativity) [9],
3. combining a small number of simple known lemmas.

ITP automation techniques are able to reduce this effort significantly. Automation techniques are most developed for systems that are based on somewhat simple logics, such as those based on first-order logic, higher-order logic, or the untyped foundations of ACL2, which is a logic and programming language in which one can model computer systems [19].

Hammers

The strongest general purpose proof assistant automation technique is today provided by tools called “hammers” [4]. Hammers combine learning from previous proofs with translation of the problems to the logics of automated systems and reconstruction of the successfully found proofs.

Blanchette et al. [4] present the three main components of a hammer in most cases.

1. The premise selector heuristically identifies a fraction of the available theorems as potentially relevant to discharge the current interactive goal.
2. The translation module constructs an ATP problem from the selected premises and the current goal, converting from the proof assistant's rich logic to the ATP's logic, often a variant of many-sorted first-order logic (FOL).
3. The proof reconstruction module processes a proof found by an ATP so that it is accepted by the proof assistant.

The advantage of a hammer is that it is a general system not depending on any domain-specific knowledge [9]. It can use not only the lemmas from standard or other predefined libraries but all currently accessible lemmas, including those proven earlier in a given formalisation.

When it comes to performance, hammers are very effective in assisting and facilitating ITP users, especially for routine proofs. As standalone provers, their performance is tied to the complexity of the domain and of the goals they are given. CoqHammer [9], can prove 40.8% of the theorems of the Coq standard library in a push-button mode. For each theorem in the library only the previous theorems and proofs could be used during evaluation.

2.1.4 ITP and Machine Learning

Gauthier et al. [12] present the problem of using Machine Learning to guide theorem provers with the following formulation: "Can we learn a mapping from a given proof state to the next tactic (or sequence of tactics) that will productively advance the proof towards a solution"?

Starting from this question, a number of different attempts have been made using various techniques in the past years. Here we will try to mention the most interesting and important of those attempts, and will focus on those which are most relevant to the Coq proof assistant for the context of this thesis.

In order to do that we will first define the problem of ITP and Machine Learning by breaking it down to its parts.

Gathering data

The first step to tackle this problem is to create the datasets necessary and build the necessary tooling and APIs to interact with them. These datasets are pairs of problems (goals) together with their proofs which is the sequence of tactics which solve the goals. The size, variety and quality of these datasets are essential for the effective training of the models. Examples of such datasets are LeanDojo Benchmark, LeanDojo Benchmark 4 - with 98,641 and 100,780 theorems respectively [47], LEANSTEP [16] and CoqGym [46].

Data Representation

The second step is to create a data representation of these proofs which will be used as an input for the model training. Here we have various approaches to the problem which include Long Short Term Memory models [46], Graph Neural Network models [31], Language models and Transformers [47].

Output representation

Here it has to be decided how the new tactics will be represented. This is done either using a tactic grammar [46], tactic templates or language modeling [47].

Lemma/Premise selection

Another important aspect of interactive theorem proving is the lemma or premise selection. This is mainly used by Hammers, and is further analyzed above 2.1.3. Language models just memorize the lemmas from the training data [47].

Term Generation

The next step is to set up a way of generating tactics. There are also various approaches for this problem, for example generating tactics as programs [46], using dependent type theory or language models [47].

Proof search strategies

The final element needed for the problem of automating interactive theorem proving with ML, is how to search for the proof. After the model which generates the tactics is available, a search method is needed to find the sequence of model generated tactics which will lead to the solution of all the goals. This search can be done with traditional methods like DFS [46], BFS, iterative deepening, best first search and Monte Carlo Tree Search [20].

In figure 2.1.1 we can see the system architecture of a machine-learning-prediction-guided proof-synthesis tool.

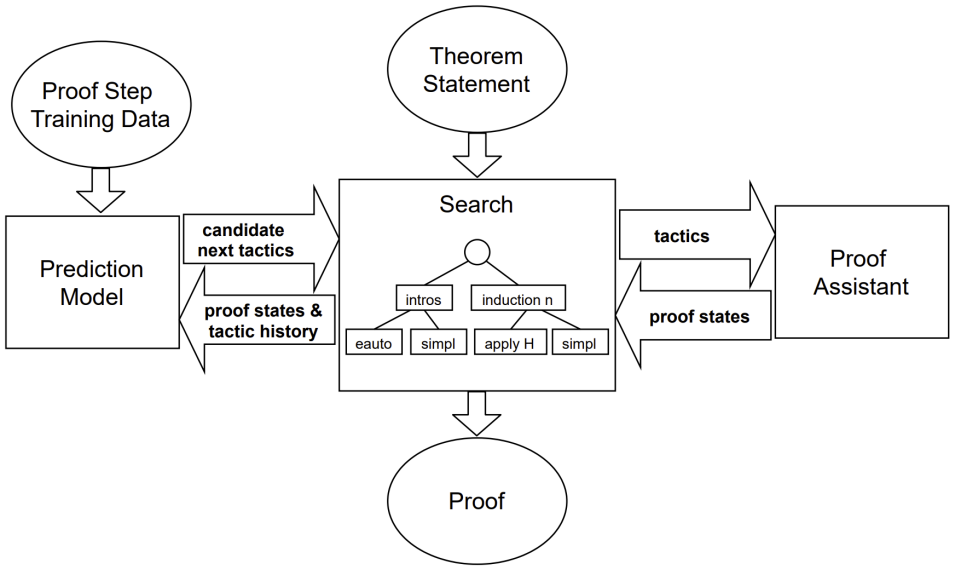


Figure 2.1.1: The system architecture of a machine-learning-prediction-guided proof-synthesis tool. [33]

Limitations

The main problems of applying machine learning to APS are:

1. There is no guarantee that the output of a DNN will be correct. Although neural networks thrive in domains like language processing, for theorem proving the answer cannot be approximate but is required to be fully correct.
2. The encoding of an APS problem as input to a DNN is non-trivial [36].

Chapter 3

Coq & CoqGym

3.1 Coq

The COQ system is designed to develop mathematical proofs, and especially to write formal specifications, programs and to verify that programs are correct with respect to their specification [2]. It is being widely used for research and industrial purposes for the development of high-assurance software systems. Examples of its usage are theorem proving in mathematics [13], software and security protocol verification, and the design and construction of programming languages. It was also being used in the CompCert verified C compiler, which is used for the formal verification of realistic compilers. Such verified compilers come with a mathematical, machine-checked proof that the generated executable code behaves exactly as prescribed by the semantics of the source program[22].

The Coq proof assistant is a robust and versatile software tool used extensively for formal verification in mathematics, computer science, and beyond. It provides a rigorous framework for establishing the correctness of mathematical theorems and software programs. At its core, Coq relies on the Calculus of Inductive Constructions (CIC), a powerful logical foundation that merges a higher-order logic and a richly-typed functional programming language.

Central to Coq's structure is its GALLINA language, a specification and mathematical higher-level language. GALLINA allows users to represent not only programs but also properties of those programs and proofs of these properties. This expressive language forms the bridge between formal mathematical reasoning and practical software verification.

GALLINA is equipped with a wide array of features that facilitate formal reasoning. It supports the definition of inductive data types, functions, and theorems, all subject to precise type checking by the Coq kernel. Furthermore, its interactive nature enables users to incrementally construct complex proofs, refining them iteratively.

In the Coq ecosystem, the Coq kernel serves as the foundational layer, responsible for type-checking, generating machine checkable proof objects, and maintaining the logical framework. Above this kernel, the proof engine interprets user-provided proof scripts, offering an interactive and user-friendly environment for formal reasoning.

Coq's extensibility is a notable asset, enabling the development of custom data types, proof rules, and tactics, making it adaptable to a diverse range of formal verification tasks.

3.1.1 Atomic & Compound tactics

Atomic Tactics: Atomic tactics in Coq are basic and elementary tactics that perform simple, focused operations in the proof [2]. These tactics are typically used to make small, incremental steps in the proof construction. Examples of atomic tactics in Coq include intros (for introducing variables or hypotheses), apply (for applying a lemma or theorem), simpl (for simplifying expressions), and rewrite (for rewriting terms

using equalities). Atomic tactics are the building blocks of more complex proof scripts.

Compound Tactics: Compound tactics, on the other hand, are tactics that combine multiple atomic tactics into a single step [2]. They allow users to automate or script more intricate proof steps by specifying a sequence of actions to be taken. Compound tactics can include conditional branching, iteration, and more. Examples of compound tactics in Coq include repeat (to repeatedly apply another tactic), ; (semicolon, used to apply tactics sequentially), and tacticals like ;, |, and +, which provide various ways to control the proof process.

An example of atomic tactics:

```
intros x y.
apply Nat.add_comm.
```

An example of compound tactics:

```
intros x y; apply Nat.add_comm.
```

3.2 CoqGym

3.2.1 CoqGym: A large-scale Interactive Theorem Prover dataset and learning environment

The CoqGym dataset and learning environment for ITP was created by extracting proofs from 123 open-source Coq projects. It contains 71K human-written proofs [46] and covers a plethora of application domains, including mathematics, computer hardware, programming languages, etc.

Previous attempts to create datasets of projects in the Coq proof assistant resulted in sets which consisted of a few thousand theorems. Furthermore, they were only covering a limited range of domains such as Peano arithmetic [10] or the Feit–Thompson Odd Order theorem [14]. CoqGym is both larger and more diverse thus facilitating training machine learning models and the evaluating cross-domain generalization.

The CoqGym environment is designed to train and evaluate auto-ITP agents. The agent begins with specific theorem to prove together with a defined set of premises. Interaction with the proof assistant involves issuing a series of tactics, with the proof assistant executing each one and providing results in the form of new goals. The proof has been successfully found by the agent when no more objectives remain.

3.2.2 Dataset structure

As already mentioned, CoqGym includes a large-scale dataset of 71K human-written proofs from 123 open-source Coq projects. In addition to the source files, it also contains abstract syntax trees (ASTs) and rich runtime information of the proofs, including the environments, the goals, and the proof trees. The ASTs have been extracted from the internals of Coq's interpreter as OCaml datatypes. They are serialized into Lisp - style S-expressions [26]. The CoqGym environment provides tools for using them in Python.

Processing Coq projects and files

The source files comprising the data set are organized under projects. Each project contains a set of inter-related proofs about specific domains. The projects in CoqGym include the Coq standard library and the packages listed on the Coq Package Index [7]. Some of them may not compile because they require a specific version of Coq, or there is a missing dependency. Only projects that compile are included in the dataset.

The training, validation and test sets are composed of different projects. This was done for the following reasons:

- Since the proofs in each project are inter-related, it was necessary to ensure that testing proofs were not used during training.
- Goal of the training of the models is to generalize accross domains.

The split was as follows:

- Training dataset: 43,844 proofs
- Training dataset: 13,875 proofs
- Testing dataset: 13,137 proofs

3.2.3 SerAPI

"SerAPI" is "a library for machine-to-machine interaction with the Coq" [1], facilitating interaction with it by using serialized data and asynchronous communication. It is designed to communicate with external tools and applications by using serialized representations of Coq's internal OCaml datatypes. The serialized data can be in various formats like JSON or S-expressions, making it easier for external tools to analyze and manipulate Coq code. It is particularly useful for tasks like code analysis, transformation, and extraction in the context of Coq developments.

3.2.4 Synthetic proofs from intermediate goals

The CoqGym dataset has also been enriched with synthetic proofs. Synthetic proofs [46] are proofs synthesized to prove the intermediate goals of existing long proofs. The main reasoning behind it was to enrich the existing dataset with smaller and less complex proofs. The hypothesis was that these intermediate goals are easier to prove and more conducive to learning. Additionally, the training data was augmented with more examples. To create these synthetic proofs Yang et al. [46] generated proofs of length 1, 2, 3, and 4 for each intermediate goal in a human-written proof.

3.2.5 Proof structure - environments, goals & proof trees

The proof environments contain the foundational elements upon which the proof will be built. They contain Coq terms as premises for the proofs. The environments for the proofs are represented as a collection of kernel terms, which are internal representations used by Coq and stripped of syntactic sugar. This form of the environment was generated through the execution of the proofs and the subsequent serialization of Coq's internals. As the source code completely defines the environment, it was considered as a possible representation for it. This would make the training of the machine learning models more challenging, as it would also imply the learning of the semantics of Coq code.

The environment for each proof - the premises in its scope - is defined both in the same source file and in other libraries. It is important to note that proofs in CoqGym contain the complete environment, which was not the case in prior work [18], The benefit of having the complete environment is that the machine learning model is able to access all relevant information in structured forms.

Proofs in the dataset are represented as proof trees. The nodes of the trees are the goals and the local context. The edges are tactics which manipulate and transform the current goal into sub-goals. This was achieved by serializing the current goals from Coq's interpreter at each step of the proof. The edges are identified by tracking how goals emerge and disappear during the lifetime of the proof. Environments, goals, and proof trees together form a structured representation of Coq proofs. Compared to raw source code, a structured representation allow machine learning models to more easily exploit the syntactic and semantic structures. It is worth noting that this structured representation is nontrivial to extract because Coq does not provide APIs exposing its internals. In constructing CoqGym, Coq was modified and together with SerAPI [1] the runtime information was serialized. The core proof-checking module of Coq was not touched, so as the correctness of the proofs would not be compromised.

Chapter 4

Machine Learning Background

4.1 Introduction to Machine Learning

4.1.1 Recurrent Neural Networks (RNNs)

Introduction to RNNs: Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed for processing sequential data or data with temporal dependencies. Unlike traditional feedforward neural networks, where information flows in one direction from input to output, RNNs have connections that loop back on themselves, allowing them to maintain and use information from previous time steps while processing the current input.

The key components of an RNN include:

1. **Hidden State (h_t):** At each time step 't', an RNN maintains a hidden state that captures information from previous time steps. This hidden state is updated at each time step based on the current input, the previous hidden state, and learned parameters (weights and biases). It serves as a summary of the data seen so far.
2. **Input (x_t):** The input at time step 't', denoted as x_t, represents the data point or feature vector from the sequence being processed. It can represent elements of a sequence, such as words in a sentence, time steps in a time series, or any other relevant information.
3. **Weights and Biases:** RNNs have weights and biases that are learned during training. These parameters determine how the input and hidden state are combined to produce the updated hidden state.

Architecture of traditional RNNs:

Hidden State Update:

The core operation in an RNN is the update of the hidden state (h_t) at each time step (t). This update is based on the current input (x_t), the previous hidden state (h_{t-1}), and learned parameters (weights and biases).

The mathematical expression for updating the hidden state in a basic RNN can be written as follows:

$$h_{t,i} = f(h_{t-1,i}, x_t)$$

where f is a non-linear activation function. An example is given below:

$$h_t = \sigma(W_{hx} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h)$$

- h_t: The updated hidden state at time step 't'. - x_t: The input at time step 't'. - h_{t-1}: The previous hidden state at time step 't-1'. - W_{hx}: Weight matrix that connects the input to the hidden state. - W_{hh}: Weight

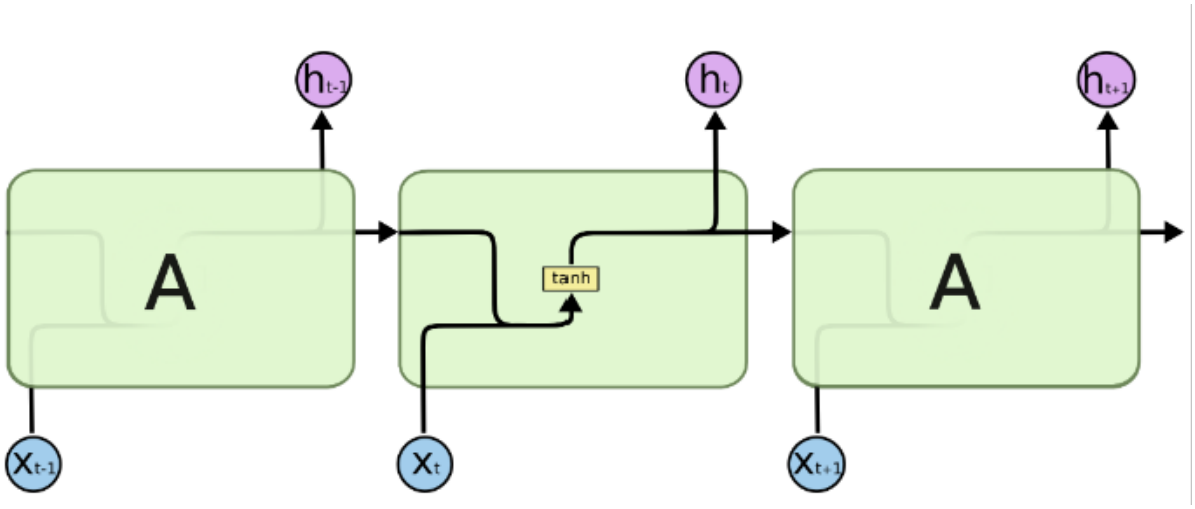


Figure 4.1.1: The architecture of a RNN. The repeating module in a standard RNN contains a single layer. [30]

matrix that connects the previous hidden state to the current hidden state. - b_h: Bias vector. - σ: Activation function, often the hyperbolic tangent (tanh) or the rectified linear unit (ReLU).

Output Calculation:

RNNs can produce an output at each time step based on the hidden state. The output at time step 't' (y_t) is calculated as:

$$y_t = \sigma(W_{yh} \cdot h_t + b_y)$$

- y_t: The output at time step 't'. - W_{yh}: Weight matrix that connects the hidden state to the output. - b_y: Bias vector.

Learning Probability Distributions with RNNs:

As described by Cho et al., 2014 [6], an RNN can learn a probability distribution over a sequence by being trained to predict the next symbol in a sequence. In that case, the output at each timestep 't' is the conditional distribution $p(x_t|x_{t-1}, \ldots, x_1)$. For example, a multinomial distribution (1-of-K coding) can be output using a softmax activation function:

$$p(x_t, j = 1|x_{t-1}, \ldots, x_1) = \frac{e^{w_j h_{t,i}}}{\sum_{j'=1}^{K} e^{w_{j'} h_{t,i}}}$$

for all possible symbols $j = 1, \ldots, K$, where w_j are the rows of a weight matrix W.

By combining these probabilities, we can compute the probability of the sequence 'x' using:

$$p(x) = \prod_{t=1}^{T} p(x_t|x_{t-1}, \ldots, x_1)$$

From this learned distribution, it is straightforward to sample a new sequence by iteratively sampling a symbol at each time step.

3. **Sequential Processing**:

RNNs process sequences by iterating through the time steps. For a sequence of length 'T', the RNN updates its hidden state from $t = 1$ to T. The final hidden state (h_T) can be used for making predictions or further processing.

4. **Backpropagation Through Time (BPTT)**:

Training an RNN involves adjusting the model's parameters (weights and biases) to minimize a loss function. This is typically done using an optimization algorithm like stochastic gradient descent (SGD) or its variants. The backpropagation algorithm is applied through time (BPTT) to compute gradients and update the parameters.

5. **Vanishing Gradient Problem**:

RNNs can struggle with long sequences due to the vanishing gradient problem. This occurs because gradients become very small when backpropagated through many time steps, making it challenging to learn long-term dependencies. More advanced RNN architectures like LSTMs and GRUs were designed to mitigate this problem.

6. **Bidirectional RNNs**:

In some cases, bidirectional RNNs are used. These models process sequences in both forward and reverse directions, allowing them to capture information from past and future time steps simultaneously.

7. **Sequence-to-Sequence Models**:

RNNs are used in sequence-to-sequence tasks, where an input sequence is transformed into an output sequence. For example, in machine translation, an RNN can encode a sentence in one language and then decode it into another language.

In practice, RNNs are often stacked in multiple layers to capture more complex features and dependencies in data. While basic RNNs are conceptually simple, the introduction of LSTM and GRU architectures has addressed some of their limitations and made them more effective for processing sequential data. These advanced architectures incorporate gating mechanisms and memory cells to better capture long-range dependencies.

4.2 Long Short-Term Memory Networks

LSTMs, or Long Short-Term Memory networks, are a type of recurrent neural network (RNN) architecture that was specifically designed to overcome some of the limitations of traditional RNNs when dealing with sequences of data. LSTMs are a fundamental component of deep learning and have found widespread use in various fields, including natural language processing, speech recognition, and time series analysis.

4.2.1 LSTMs definition

As mentioned by Zaremba and Sutskever [49] the key feature that sets LSTMs apart from traditional RNNs is their ability to capture and remember long-range dependencies and relationships within sequential data. Traditional RNNs suffer from the vanishing gradient problem, which makes it challenging for them to learn and remember information from distant time steps in a sequence. LSTMs address this issue by introducing a more complex memory mechanism.

We define the LSTM unit at each time step t to be a collection of vectors in $\mathbb{R}^d$:

- an input gate i_t,
- a forget gate f_t,
- an output gate o_t,
- a memory cell c_t,
- a hidden state h_t.

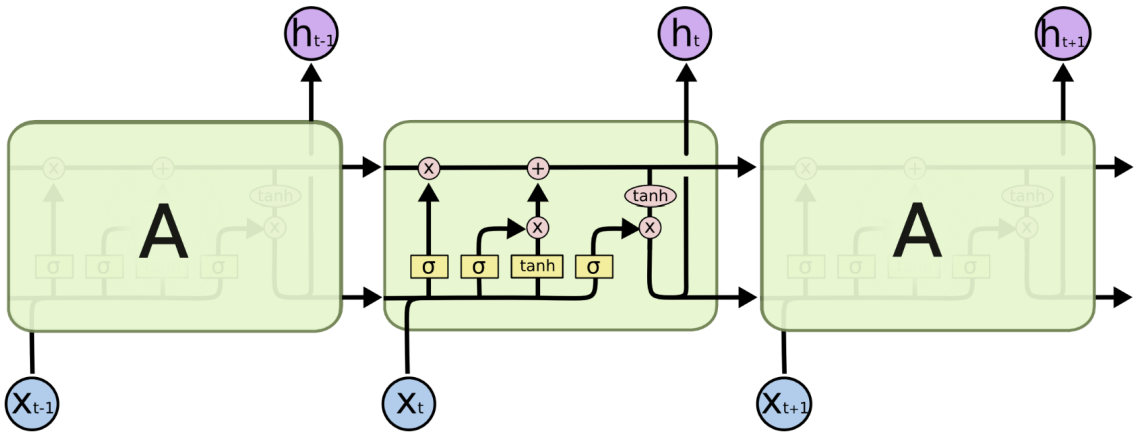


Figure 4.2.1: The architecture of an LSTM. The repeating module in an LSTM contains four interacting layers.
[30]

The entries of the gating vectors i_t, f_t, and o_t are in the range [0, 1]. The memory dimension of the LSTM is refered to as d.

4.2.2 LSTM transition equations:

The LSTM transition equations are the following:

$$it = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}\right) \tag{4.2.1}$$

$$ft = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\right) \tag{4.2.2}$$

$$ot = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}\right) \tag{4.2.3}$$

$$ut = \tanh\left(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}\right) \tag{4.2.4}$$

$$ct = it \odot ut + ft \odot ct_{t-1} \tag{4.2.5}$$

$$ht = ot \odot \tanh(ct) \tag{4.2.6}$$

In the equations above: x_t is the input at the current time step, σ denotes the logistic sigmoid function, $\odot$ denotes elementwise multiplication.

Cell State (C_t): The cell state serves as the memory of the LSTM and allows information to flow through the network over long sequences. It can selectively retain or forget information, making it suitable for modeling dependencies over time.

Hidden State (h_t): The hidden state is responsible for capturing and passing on relevant information to the next time step. It is updated based on the current input, the previous hidden state, and the cell state.

Input Gate (i_t): The input gate controls how much each unit is updated It takes into account the current input and the previous hidden state.

Forget Gate (f_t): The forget gate controls the extent to which the previous memory cell is forgotten. It considers the current input and the previous hidden state.

Output Gate (o_t): The output gate controls the exposure of the internal memory state, and more specifically which parts of the cell state are used to compute the current hidden state. It also influences the output of the LSTM cell.

The hidden state vector in an LSTM unit is therefore a gated, partial view of the state of the unit's internal memory cell. Since the value of the gating variables varies for each vector element, the model can learn to represent information over multiple time scales.

Variants of LSTMS are Bidirectional LSTMs and Multilayer LSTMs (Graves et al., 2013 [15]).

4.2.3 Bidirectional LSTMs

Bidirectional LSTMs (Long Short-Term Memory networks) (Graves et al., 2013 [15]) are a type of recurrent neural network (RNN) architecture that process sequences in both forward and backward directions. This bidirectional processing allows them to capture dependencies in the past and future context of each time step in a sequence.

Forward LSTM:

The forward LSTM processes the sequence from left to right. It computes a hidden state h_t^{forward} at each time step t based on the input x_t and the previous hidden state h_{t-1}^{forward}. The computation is done based on the transition equations presented above 4.2.2.

Backward LSTM:

The backward LSTM processes the sequence from right to left. It computes a hidden state h_t^{backward} at each time step t based on the input x_t and the previous hidden state $h_{t+1}^{\text{backward}}$. The backward LSTM equations are analogous to those of the forward LSTM but with different weight matrices and biases.

Concatenation:

The outputs of the forward and backward LSTMs at each time step can be concatenated to form the final bidirectional output $h_t^{\text{bi}} = [h_t^{\text{forward}}, h_t^{\text{backward}}]$. This concatenated output captures information from both directions.

4.2.4 Tree-Structured LSTMs

Traditional LSTM architectures only allow for strictly sequential information propagation. This limitation has spurred the development of more sophisticated and versatile models. Tai et al., 2015 [43] in their work have introduced the Child-Sum Tree-LSTM and the N-ary Tree-LSTM. Both of them enable richer network topologies, in which each LSTM unit can assimilate information from multiple child units. The two models maintain the basic structure of standard LSTM units. Each unit contains j contains input and output gates i_j and o_j, a memory cell c_j and hidden state h_j. However, the key distinction lies in how these units handle gating vectors and memory cell updates, which can depend on potentially **many** child units. Furthermore, the TreeLSTM unit contains one forget gate f_jk for each child k, instead of a single forget gate.

In practical applications, each Tree-LSTM unit receives an input vector x_j. For example, in some NLP tasks where the input consists of sentences, x_j represents a vector representation of each word within the sentence. The choice of input vectors and the way they are organized depend on the specific tree structure used in the network.

Child-Sum Tree-LSTM

Given a tree, let $C(j)$ denote the set of children of node j. The Child-Sum Tree-LSTM transition equations are the following:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k \tag{2}$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right) \tag{3}$$

$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right) \tag{4}$$

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right) \tag{5}$$

$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right) \tag{6}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \tag{7}$$

$$h_j = o_j \odot \tanh(c_j) \tag{8}$$

where in Eq. 4, $k \in C(j)$ [43].

An intuitive interpretation of each parameter matrix in these equations is to view it as encoding correlations between the component vectors of the Tree-LSTM unit, the input x_j, and the hidden states h_k of the unit's children. For example, in a dependency tree application, the model can learn parameters $W^{(i)}$ such that the components of the input gate i_j have values close to 1 (i.e., "open") when a semantically important content word (such as a verb) is given as input, and values close to 0 (i.e., "closed") when the input is a relatively unimportant word (such as a determiner) [43].

4.3 GRU

Cho et al., 2014 [6] motivated by the LSTM architecture presented above, proposed a new type of hidden unit - "a hidden unit that adaptively remembers and forgets"

Those units, now known as GRUs, or Gated Recurrent Units, are a type of recurrent neural network (RNN) architecture that, like LSTMs (Long Short-Term Memory networks), is designed to address the vanishing gradient problem and facilitate the learning of long-range dependencies in sequential data.

GRU architecture:

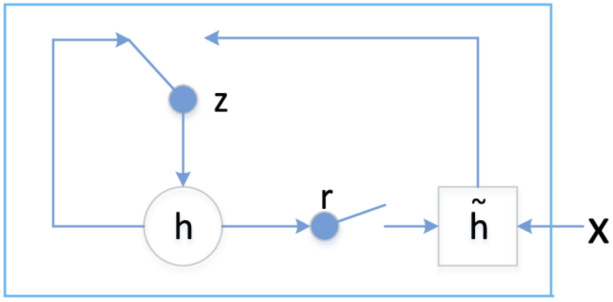


Figure 4.3.1: Hidden activation function of GRU.
[23]

Hidden State (ht): Like traditional RNNs and LSTMs, GRUs maintain a hidden state that captures information from previous time steps. This hidden state is updated at each time step.

Reset Gate (rt): The reset gate in a GRU controls how much of the previous hidden state ('ht-1') should be forgotten or reset at the current time step. It takes into account the current input and previous hidden state and produces values between 0 and 1, indicating what information to retain from the past.

$$r_j = \sigma\left([W_r x]_j + [U_r h_{\langle t-1\rangle}]_j\right)$$

Update Gate (zt or ut): The update gate, sometimes denoted as 'zt' or 'ut', determines how much of the new candidate state should be mixed with the previous hidden state to compute the current hidden state. Similar to the reset gate, it also takes into account the current input and previous hidden state.

$$z_j = \sigma\left([W_z x]_j + [U_z h_{\langle t-1\rangle}]_j\right)$$

Candidate State ($\hat{\text{ht}}$): The candidate state is computed based on the current input and the previous hidden state. It represents the new information that could potentially be added to the hidden state.

The actual activation of the proposed unit h_j is computed by

$$h_j^{\langle t\rangle} = z_j h_j^{\langle t-1\rangle} + (1 - z_j)\tilde{h}_j^{\langle t\rangle}$$

where

$$\tilde{h}_j^{\langle t\rangle} = \varphi\left(W_x^{\langle t\rangle} + [U\left(r \odot h_{\langle t-1\rangle}\right)]_j\right)$$

The update gate and reset gate are crucial components that control the flow of information in and out of the GRU cell. These gates allow GRUs to selectively update their hidden states, making them capable of capturing both short-term and long-term dependencies in sequential data. Those units that learn to capture short-term dependencies will tend to have reset gates that are frequently active, but those that capture longer-term dependencies will have update gates that are mostly active [6].

Compared to LSTMs, GRUs have fewer parameters and are computationally less intensive, which can make them easier to train on smaller datasets and in situations where computational resources are limited. However, LSTMs tend to perform slightly better on tasks that require modeling complex long-term dependencies due to their more elaborate memory cells.

4.4 Encoder - Decoder Architecture

The encoder-decoder proposed by Cho et al., 2014 [6] consists of two main components: an encoder and a decoder, which respectively learn to "encode a variable-length sequence into a fixed-length vector representation and to decode a given fixed-length vector representation back into a variable-length sequence".

From a probabilistic perspective, this new model is a general method to learn the conditional distribution over a variable-length sequence conditioned on yet another variable-length sequence, e.g., $p(y_1, \ldots, y_{T_0}|x_1, \ldots, x_T)$, where one should note that the input and output sequence lengths T and T_0 may differ [6].

This architecture is particularly prevalent in natural language processing tasks, machine translation, text summarization, speech recognition, and more. Here's an overview of each component:

Encoder:

The encoder is the first part of the architecture and is responsible for processing the input data and encoding it into a fixed-dimensional representation, often referred to as a "context vector" or "thought vector." It takes variable-length input sequences, such as sentences or time series data, and transforms them into a compressed representation with fixed dimensions. Typically, the encoder is built using recurrent neural networks (RNNs), long short-term memory (LSTM) networks, gated recurrent units (GRUs), or more recently, transformer-based models like the Attention Is All You Need architecture. As the encoder reads each symbol, the hidden state of the RNN changes according to Eq. (1). After reading the end of the sequence (marked by an end-of-sequence symbol), the hidden state of the RNN is a summary c of the whole input sequence. The encoder's output is a high-level representation of the input sequence, which is then passed to the decoder.

Decoder:

The decoder is the second part of the architecture and is responsible for generating an output sequence from the encoded representation produced by the encoder. It takes the context vector from the encoder and, step by step, generates elements of the output sequence. Like the encoder, the decoder can also be constructed using recurrent networks or transformer-based models. During training, the decoder is typically provided with the correct target sequence, and its output is compared to the target sequence to calculate the loss for training purposes.

The decoder of the proposed model is another RNN which is trained to generate the output sequence. It is typically provided with the correct target sequence and tries predicting the next symbol y_t given the hidden state h_t^i. Its output is compared to the target sequence to calculate the loss for training purposes. However, unlike the RNN described in Sec. 2.1, both y_t and h_t^i are also conditioned on y_{t-1} and on the summary c of the input sequence. Hence, the hidden state of the decoder at time t is computed by,

$$h_t^i = f(h_{t-1}^i, y_{t-1}, c)$$

and similarly, the conditional distribution of the next symbol is

$$P(y_t|y_{t-1}, y_{t-2}, \ldots, y_1, c) = g(h_t^i, y_{t-1}, c)$$

for given activation functions f and g (the latter must produce valid probabilities, e.g., with a softmax function). During inference or testing, the decoder generates the output sequence one element at a time, often autoregressively, using its own previously generated elements as input.

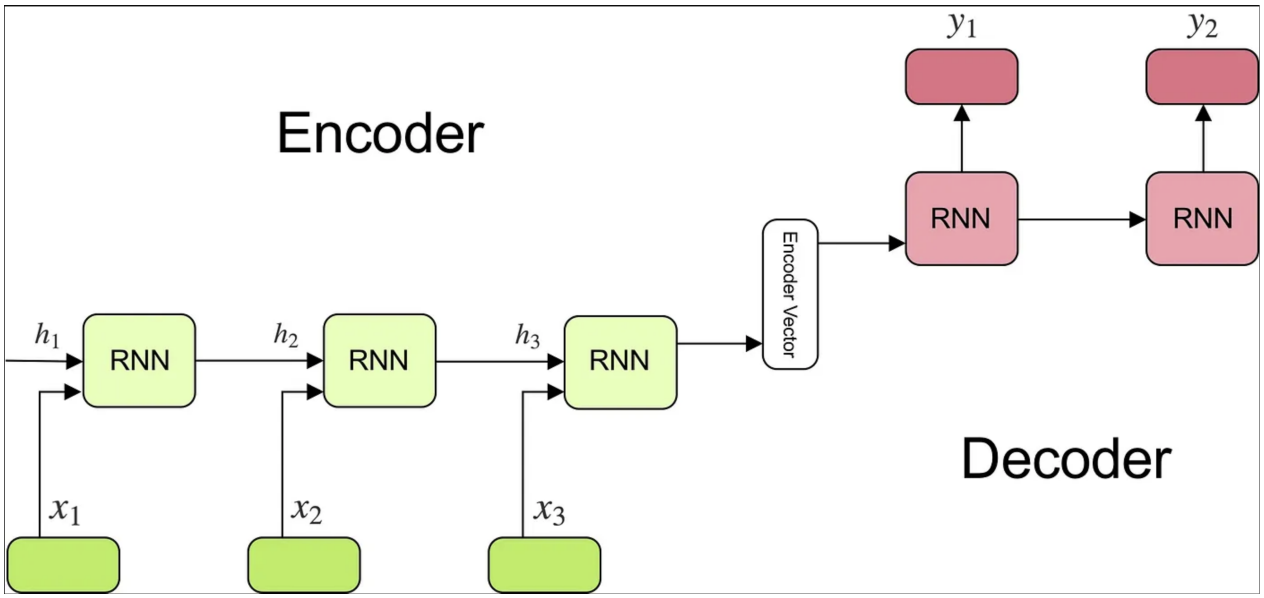


Figure 4.4.1: An illustration of the RNN Encoder–Decoder Sequence to Sequence model

The two components of the proposed RNN Encoder-Decoder are jointly trained to maximize the conditional log-likelihood [6]:

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(y_n|x_n)$$

where θ is the set of model parameters, and each (x_n, y_n) is an (input sequence, output sequence) pair from the training set. In our case, as the output of the decoder, starting from the input, is differentiable, we can use a gradient-based algorithm to estimate the model parameters.

The resulting trained model can be used in two ways. One approach is for the model to generate target sequences from input sequences. The second approach is for the model to produce a probability value $p_\theta(y|x)$ from Eqs. (3) and (4) for a given input and output.

The encoder-decoder architecture is versatile and has been applied to various tasks beyond natural language processing, such as image captioning, speech synthesis, and more. It is particularly useful for tasks where the input and output sequences have different lengths or where the mapping between input and output is complex and context-dependent. Additionally, attention mechanisms, often used in conjunction with encoder-decoder architectures, have improved their performance by allowing the decoder to focus on different parts of the input sequence when generating the output.

Chapter 5

Models

5.1 ASTactic: generating tactics as programs

In this section an attempt will be made to present the architecture of ASTactic and details of its design and structure [46]. This is important since models like Passport [33] and TacTok [11] which will be also be presented are building on top of the ASTactic architecture and utilize parts of its implementation.

ASTactic is a deep learning model that generates tactics as programs [46]. It is trained on CoqGym, and it stood out from previously suggested automated theorem provers because during the proof search phase it does not select tactics from a fixed set. Instead, tactics are dynamically generated by ASTactic as abstract syntax trees (ASTs). The output of ASTactic is then utilized and tested by sampling. On each state of the proof, a number of Tactics is selected, based on the predefined BEAM size. These are the possible actions to take from the current state. The search continues via depth-first search (DFS) until a correct proof is found, or until the maximum number of tactics is reached or the search times out.

5.1.1 Space of tactics

The output space of ASTactic is "specified by a context-free grammar (CFG) that is fixed during training and testing" [46]. A statistical analysis of the proofs in CoqGym showed that many of those valid tactics are seldom used in the data set, and thus omitting them could facilitate the learning, at the expense of not successfully finding a proof for the theorems in the test data which require those omitted tactics. For that reason, the tactic grammar does not contain all the possible valid tactics that can be used in a Coq proof. Additionally, ASTactic can only generate atomic tactics 3.1.1. Compound tactics and user defined tactics have been excluded to simplify the tactic generation. This is not a severe handicap because all proofs can be completed without compound tactics. When a tactic requires a Coq term as its argument, the term to be an identifier the term is constrained to be an identifier.

5.1.2 Architecture

Overall model architecture

ASTactic has an encoder - decoder architecture (Fig. 2). Both the input and the output of the model are trees. The encoder embeds all input Coq terms: the goal and the premises expressed in ASTs. Conditioned on the embeddings, the decoder generates a program-structured tactic by sequentially growing an AST.

Encoder

ASTactic encodes the goal and premises of each state into vectors. What is included in the encoding is very significant as it deeply impacts the training of the model. In this case the entire local context of the proof and up to 10 premises in the environment are included. The authors decided τo exclude a large number of premises imported from libraries, which were not related to the proof. A significant case could be made for

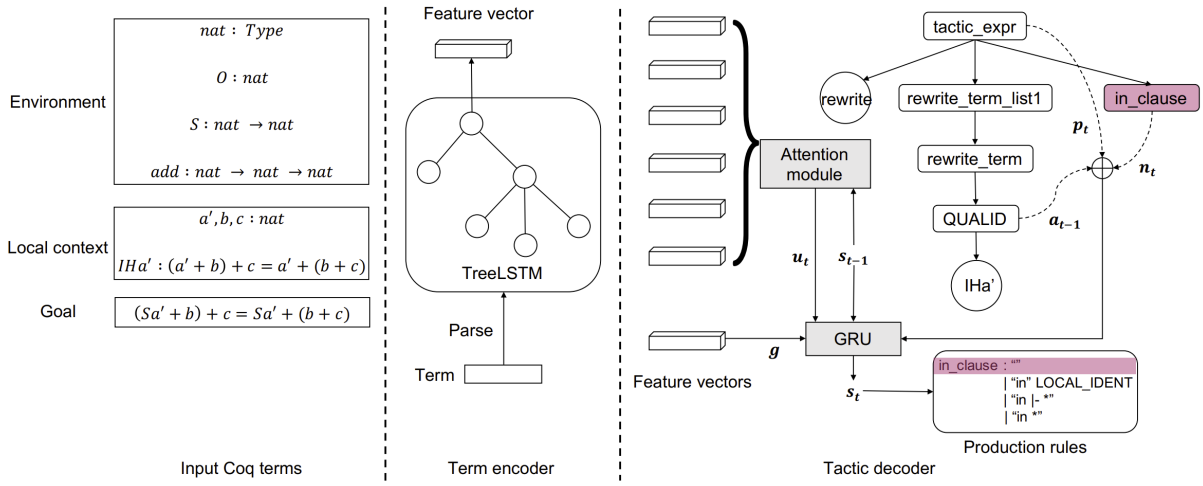


Figure 5.1.1: ASTactic architecture. The proof state encoder (a), takes as input the goal, local context, and environment terms in AST form and generates embeddings (feature vectors) for each term. The tactic decoder (b) concatenates the input embeddings and generates a tactic in the form of an AST, conditioned on these inputs.
[46]

adding logic to select relevant premises from the entire environment. This could potentially generate a more powerful model, and it is something that we will also mention in possible improvements and future work. Both the goal and the premises are Coq terms in the form of ASTs and are encoded using a TreeLSTM network 4.2.4.

Specifically, each node in an AST has a symbol n indicating its syntactical role. The network associates each node with a hidden state h and a memory cell c which are updated by its children as follows:

$(c, h) = f_{update}(n, c_1, \ldots, c_K, \sum_{i=1}^{K} h_i)$

The update function f_{update} is the Child-Sum variant of TreeLSTM, n is the symbol of the node in one-hot encoding, and c_i and h_i are the cell state and hidden state of the children.

This computation is performed bottom-up. The entire tree is represented by h_{root}, the hidden state of the root. Finally, the h_{root} is appended with a 3-dimensional one-hot vector. The vector indicates whether the term is the goal, a premise in the environment, or a premise in the local context.

Decoder

The challenging task of the decoder is to synthesize tactic arguments. Within the decoder, constraints based on semantics are integrated to restrict the search space for arguments.

ASTactic's decoder generates program-structured tactics as Abstract Syntax Trees (ASTs) following the method in Yin & Neubig (2017) [48].

The foundation of their method revolves around a grammar model. This model formalizes the process of generating a derivation AST as a sequence of actions. These actions can either apply production rules or produce terminal tokens. Consequently, the syntax of the programming language is predefined within the grammar model as a set of potential actions. Thus, their approach eliminates the need for the model to deduce the grammar solely from limited training data. Instead, it allows the system to concentrate on understanding how the existing grammar rules interact and combine [48].

It begins with a single node and grows a partial tree in depth-first order in the following manner:

- It expands non-terminal nodes by choosing a production rule in the Context-Free Grammar (CFG) of the tactic space.

- For terminal nodes, it emits a token corresponding to a tactic argument.

This sequential generation process is controlled by a Gated Recurrent Unit (GRU) 4.3, whose hidden state is updated by the input embeddings and local information in the partially generated AST.

All symbols and production rules in the tactic grammar have learnable embeddings.

At time step t, let n_t be the symbol of the current node; a_{t-1} is the production rule (also referred as action) used to expand the previous node; p_t is the parent node's state concatenated with the production rule used to expand the parent; g is the goal, which is fixed during the generation process. The state s_t is updated by:

$$s_t = f_{GRU}(s_{t-1}, [a_{t-1} : p_t : n_t : g : u_t]) \tag{1}$$

where ":" denotes vector concatenation.

- u_t is a weighted sum of premises.
- s_{t-1} is used to compute an attention mask on the premises, which selectively attends to the relevant premises for the current generation step.

The mask is then used to retrieve u_t:

$$w_i = f_{att}(s_{t-1} : r_i) \tag{2}$$

$$u_t = \sum_i w_i r_i \tag{3}$$

where

- r_i is the i-th premise,
- w_i is its weight,
- f_{att} is a two-layer fully-connected network.

Expanding ASTs and synthesizing arguments

In the context of expanding the ASTs and generating associated arguments, the pivotal factor is the state variable denoted as s_t. This variable plays a crucial role in dictating the expansion strategy, determining the application of production rules and the generation of tokens.

To facilitate the selection of a production rule, Yang et al. [46] in their approach are modeling the probabilities associated with these rules, selecting the node with the highest probability. Specifically, $\mathbf{p}_t$ is defined as the softmax output of a transformation, computed as $\text{softmax}(W_R \cdot f(s_t))$,

where:

- f denotes a linear layer followed by the application of a hyperbolic tangent function,
- $\mathbf{W}_R$ represents the embedding matrix corresponding to production rules.

Within the AST representation, tokens correspond to tactic arguments. Synthesizing these arguments presents a challenge due to the large syntactic space, which effectively includes all valid identifiers within the Coq language. Nonetheless, there are strong semantic constraints on these arguments. For example, in the case of a tactic such as "apply H', where 'H' represents a premise, 'H' must invariably correspond to a valid premise either within the environment or the local context.

In order to effectively incorporate these semantic constraints into the argument synthesis process, Yang et al. [46] did the following:

Identifiers of premises (as in "apply H"): Each premise is scored using s_t in the same way as computing the attention masks (Equation 3). The probability for each premise is given by a softmax on the scores.

Integers (as in "constructor 2"): Most integers in the data are in the range of [1, 4]. They are generated using a 4-way classifier.

Quantified variables in the goal (as in "simple induction n"): A universally quantified variable is randomly picked from the goal.

Evaluation environment

ASTactic also provides an evaluation environment - an environment of interacting with Coq theorems during testing. It receives a command from the model(in this case ASTactic) agent and interacts with Coq using SerAPI 3.2.3. The environment can perform single interactions - steps, providing commands to Coq and getting the results of the application of those commands. Commands can include Coq tactics to be applied or commands other valid Coq commands like `Admitted` - for giving up the proof and `Undo` - for backtracking.

The environment returns the feedback given from Coq to the agent. The possible responses that can be returned on the progress of the proof are:

- ALREADY_SUCCEEDED
- ALREADY_FAILED
- MAX_TIME_REACHED
- MAX_NUM_TACTICS_REACHED
- ERROR
- GIVEN_UP
- PROVING
- SUCCESS

The evaluation environment is used for the evaluation of ASTactic and the models presented below. We also used it for our own experiments, which we present in a following chapter.

5.2 TacTok

TacTok is a model of next-tactic prediction for Coq proposed by First et al. [11]. The core motivation behind the creation of TacTok comes from two observations:

1. When human users interact with Coq to prove a theorem or verify a proof, they very often have to examine the proof state in order to choose the next tactic to apply. This inspired the idea to test how next-tactic prediction models would benefit from having access to information in the proof state.
2. Since Coq users, when building a proof, are aware of all the tactics that have been applied up to each proof state, a model of next tactic prediction may also benefit from having access to the previous tactics in the proof script.

The model is trained on CoqGym, through the following technique:

- It traverses the existing proof scripts by stepping through them, one tactic at a time. For each step it computes the resulting intermediate proof states.
- It creates an embedding of both the proof states and the partially written proof script at each step. The embeddings are mapped to an abstract syntax tree (AST) of the next line in the proof script.

Thus, the model's inputs are the partially written proof script and the proof state.

The model's output is an AST that TacTok decodes to the next predicted tactic and its arguments.

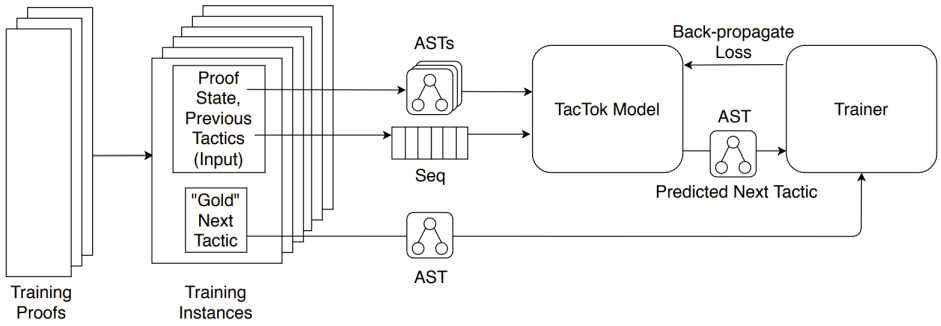
Each proof script is broken down into training instances. A training instance consists of the following:

1. the proof state after a tactic from the proof script has been executed,
2. the proof script up to the last tactic,

3. the next tactic in the script.

The proof state includes:

1. the current goal,
2. the local context,
3. the environment.



(a) TacTok training process [11].

Each term in the proof state has an underlying AST. The proof script is represented as a sequence of tokens. The TacTok model jointly learns embeddings for these ASTs and sequences. The TacTok model uses these embeddings to output a predicted next proof script step, in the form of an AST, and sends that along with the AST form of the ground-truth next tactic to the trainer. The trainer then compares these tactics and back-propagates the loss.

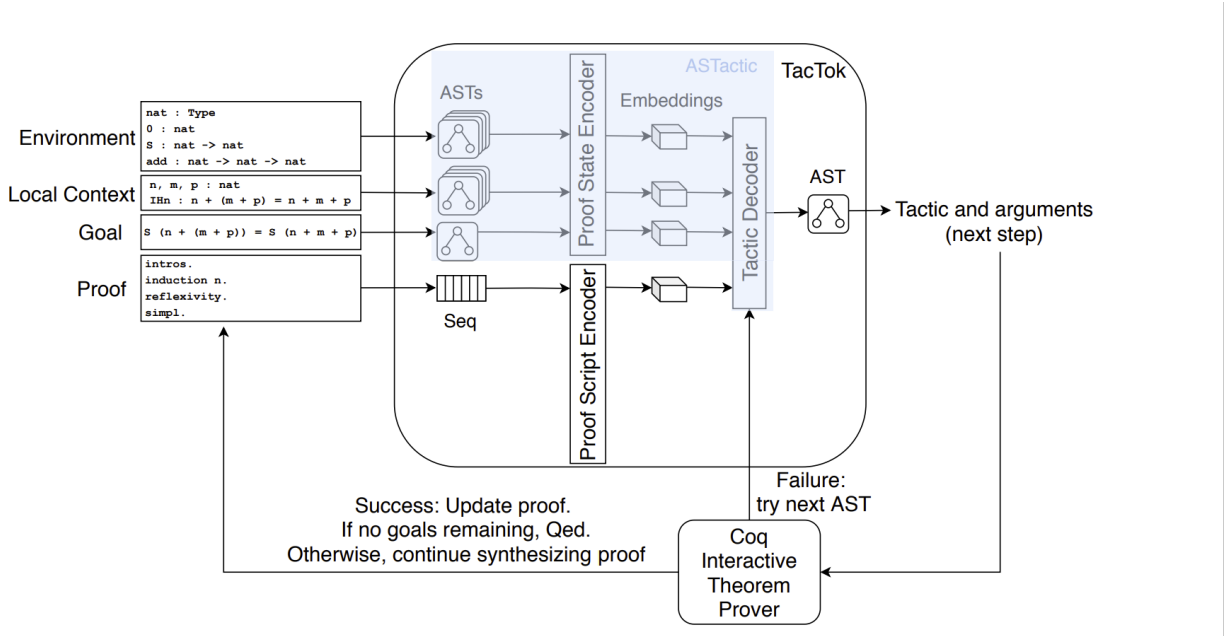


Figure 5.2.2: TacTok, in the process of completing the proof script of associativity for the add function, after the execution of simpl.
[11]

5.2.1 Proof State Encoder

The encoder of the proof state is the one which was introduced from ASTactic and is analyzed in detail in the previous section of this chapter 5.1.2. The inputs are the goal, local context, and environment in AST form.

5.2.2 Proof Script Encoder

The proof script consists of tokens that are either tactics, arguments to tactics, or other symbols. The proof script encoder parses the sequence of these tokens in two different modes.

- Tac: Parses the sequence including the most common Coq tactic tokens appearing in the training proof scripts, excluding custom tactics, and obscure arguments so that their names are not learned.
- Tok: includes the entire token sequence, only excluding punctuation.

The parsed sequence of previous tokens is encoded with a Bidirectional LSTM 4.2.3, which generates an embedding for the sequence. In this way, the input sequence is processed in both the forward and backward directions, and the outputs of these directions are concatenated to capture context from both past and future proof steps.

TacTok comprises both the Tac and Tok models, trained separately, and uses either one when attempting to synthesize a proof script.

5.3 Passport

In their work Sanchez-Stern, A. et al. [33] focus on modeling identifiers. Their goal is to exploit the rich information of the proof data the model is trained on, thus improving performance.

Passport is the resulting proof-synthesis tool for Coq built on top of ASTactic, enriching the existing models by introducing three new encoding mechanisms for identifiers:

1. category vocabulary indexing,
2. subword sequence modeling,
3. path elaboration.

In this section we will be giving a brief overview of the above mechanisms, without analyzing all the technical and implementation details of the work, since it is beyond the scope of this thesis.

5.3.1 Identifier Categories in Passport

In computer science and programming, "identifiers" are symbols or names used to uniquely identify variables, functions, constants, or other elements in a program. In the context of Coq proofs and proof states, identifiers are "the names that uniquely identify theorems, datatypes, functions, type constructors, and local variables" [33].

Passport encodes each identifier according to its category:

1. global definition,
2. local variable,
3. type constructor.

Global Definitions

The most straightforward of our categories to include was identifiers referencing global definitions. These identifiers refer to objects defined globally directly by the user, using the keywords Definition, Theorem, Inductive, or one of their variants. Global definitions are generally either an inductive type name, or a name given to some Gallina term (function, constant value, etc). Crucially, since proof objects themselves are terms, theorems are global definitions with their names bound to their proof objects.

Local Variables

Besides global definitions, local variables are the most common type of identifier in Coq terms. Local variables can be bound to an explicit term, as in a let definition, but in many cases (function parameters, forall bindings, and existential pairs) are given only a type binding. This is in contrast to global definitions, which are always bound directly to terms. Encoding local variables is often critical to determining the correct next step in a proof, or even understanding its basic structure. Even when the local variable's name isn't particularly informative, knowing when local variables repeat is often critical.

Type Constructors

Unlike global definitions and local variables, type constructors are not bound on their own, but are instead defined as part of inductive type definitions.

5.3.2 Encoding Mechanisms in Passport

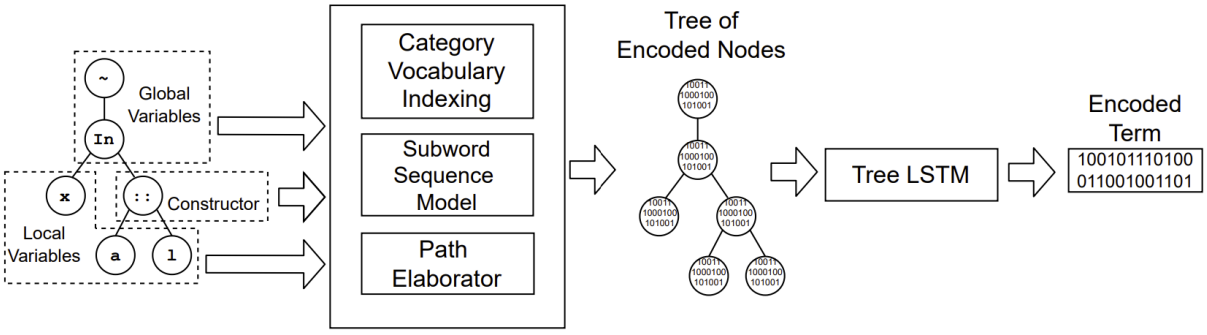


Figure 5.3.1: The architecture of Passport's identifier processing.
[33]

Category Vocabulary Indexing:

Sanchez-Stern, A. et al. [33] point that since Coq has no primitive datatypes; every referenced type is an identifier. This makes them very significant as they can "carry a lot of meaning—and that meaning can be reflected in the names of theorems that refer to them".

With Category Vocabulary Indexing, each identifier is tagged with the category it comes from. This distincts identifiers with the same name from different categories. It can even provide the model with useful information about uncommon identifiers. Additionally, the most common identifiers in each category are given a unique tag, which is included in their encoding. In this way Passport creates an association between all usages of the exact identifier. Furthermore, the model can create generalization about their behavior and usages, and predict tactics that worked effectively with them in past usages.

Subword Sequence Modeling:

Subword sequence modeling is a technique used in natural language processing (NLP) and machine learning to handle words and tokens at a more granular level than whole words. It involves breaking down words into smaller units, such as subword pieces or characters, and modeling sequences of these subword units [38]. This approach is particularly useful for handling languages with complex morphology, agglutinative languages - languages which form words through the combination of smaller morphemes to express compound ideas -, or when dealing with out-of-vocabulary words.

With this technique Sanchez-Stern, A. et al. [33] are trying to take advantage of the correlations that can be created between identifiers based on their names and the overall naming conventions used across proofs. For all identifiers, Passport uses a subword sequence model to draw bridges between related names. That is, identifiers are broken down into common word-pieces, and processed with a sequence model.

For example the variable name `orderedListAsc` can be broken down to "ordered", "list" and "asc".

Path Elaboration:

Path elaboration is the last encoding technique used in Passport: the encoding of fully-qualified paths of different identifiers. Fully-qualified paths are "the names of directories, files, and modules" within which different identifiers - type constructors and global definitions are contained. In this way, Passport can take advantage of any grouping of identifiers into common modules and files already used by Coq developers to organize development. Passport can also capitalize on proof development styles that dispatch proofs for entire classes of related theorems using powerful tactics.

Chapter 6

Monte Carlo Tree Search

6.1 Markov Decision Processes

A Markov Decision Process (MDP) is a discrete, stochastic, and generally finite model of a system to which some external control can be applied [44]. When used for reinforcement learning, firstly the parameters of an MDP are learned from data, and then the MDP is processed to choose a behavior.

MDP is a process defined as a tuple $< \mathcal{S}, \mathcal{A}, T, R >$, where:

- $\mathcal{S}$ is the set of states that are possible in an environment (state space). A specific state s0∈S is distinguished as the initial state.
- $\mathcal{A}_s$ is the set of actions available to perform in state s. The subscript S can be omitted if all actions are always available in the given environment.
- $P_a(s, s')$ is the transition function modelled by a probability that action αperformed in state s will lead to state s'. In deterministic games, the probability is equal to 1 if the action in state s leads to s', whereas 0 if it does not.
- $Ra(s)$ is the immediate reward (payoff) for reaching state s by action α. In Markov games, where states incorporate all the necessary information (that summarizes the history), the action component can be omitted.

We assume the Markov Property: "For a Markov chain the conditional distribution of any future state X_{n+1}, given the ppast states $X_0, X_1, ..., X_{n-1}$ and the present state X_n, is independent of the past states and depends only on the present states" [32].

6.2 MCTS algorithm

MCTS is an iterative search algorithm, directly applicable to problems which can be modelled by a Markov decision process (MDP) [25]. Additionally, certain modifications of MCTS make it possible to apply it to partially observable MDPs (POMDPs). It collects and utilizes statistical evidence to assess the decisions available in particular states and navigate the search space.

MDP allows to model a simulated environment with sequential decisions to make in order to receive a reward after certain sequence of actions. The next state of the environment can be derived only using the current state and the performed action.

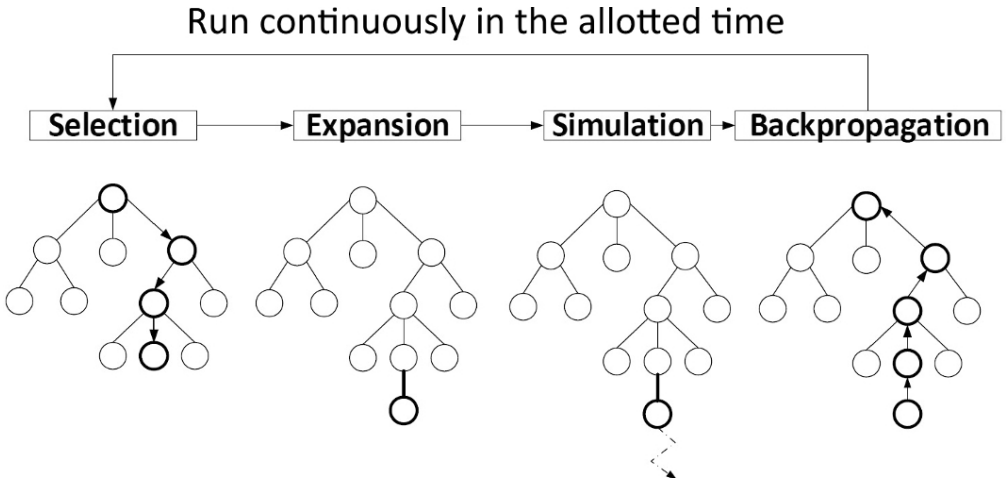
In complicated problems or games, like chess and go, the size of the state space (e.g. a game tree) makes an exhaustive search impossible. In practical applications, MCTS is allotted some computational resources which can be either specified by the number of iterations or the time available for making a decision. MCTS is an anytime algorithm. This property means that it can be stopped at anytime and provide the currently best action (decision) using Eq. 1 [42]:

$$a^* = \arg\max_{a \in A(s)} Q(s,a), \quad (1)$$

where $A(s)$ is a set of actions available in state s, in which decision is to be made and $Q(s,a)$ denotes the empirical average result of playing action $a$in state s. Naturally, the more the iterations, the more confident the statistics are and the more likely it is that the recommended best action is indeed the optimal one.

Each iteration consists of four phases as depicted in the image.

1. Selection: In this phase, the algorithm starts at the root of the tree (the current game state or decision point) and navigates down the tree following a selection policy. The selection policy often balances exploration and exploitation, aiming to visit promising nodes while also exploring new possibilities. This phase continues until a leaf node (a node with unexplored actions or unvisited states) is reached.
2. Expansion: Once a leaf node is reached, the expansion phase comes into play. In this step, the algorithm selects one or more child nodes corresponding to untried actions or unvisited states from the current node. These actions or states represent potential future moves or decisions. When expansion reaches the terminal state, which is extremely rare, then the current iteration skips directly to backpropagation.
3. Simulation (Rollout): After expansion, the algorithm conducts a simulation (also known as a rollout) from one of the newly added child nodes. The simulation often involves using a simple heuristic or a random policy to play out the game or explore the potential outcomes from the selected child node until a terminal state or a predefined depth is reached. This step helps estimate the value of the current child node.
4. Backpropagation: After completing the simulation, the algorithm backpropagates the result (outcome) of the simulation back up the tree to update the statistics associated with each node along the path from the root to the expanded child node. This update includes adjusting the visit counts and the accumulated values (e.g., wins or rewards). The backpropagation phase helps refine the value estimates of nodes in the tree.



(a) Monte Carlo Tree Search phases [42]

6.3 MCTS selection policy

The aim of the selection policy is to maintain a proper balance between the exploration (of not well-tested actions) and exploitation (of the best actions identified so far). The most common algorithm, which has become de facto the enabler of the method is called Upper Confidence Bounds applied for Trees (UCT) introduced by Kocsis and Szepesvári (2006) [21]. First, it advises to check each action once and then according to the following formula:

$$a^* = \arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln\,[N(s)]}{N(s,a)}} \right\},$$

where A(s) is a set of actions available in state s, Q(s, a) denotes the average result of playing action a in state s in the simulation performed so far, N(s) is a number of times state s has been visited in previous iterations and N(s, a)—a number of times action a has been sampled in state s. Constant C controls the balance between exploration and exploitation. The usually recommended value of C to test first is $\sqrt{2}$, assuming that Q values are normalized to the [0, 1] interval, but it is a game-dependent parameter, in general.

Due to the UCT formula and random sampling, MCTS searches the game tree in an asymmetric fashion in contrast to traditional tree search methods such as minimax. The promising lines of play are searched more thoroughly. Figure 2 illustrates the type of asymmetrical MCTS search.

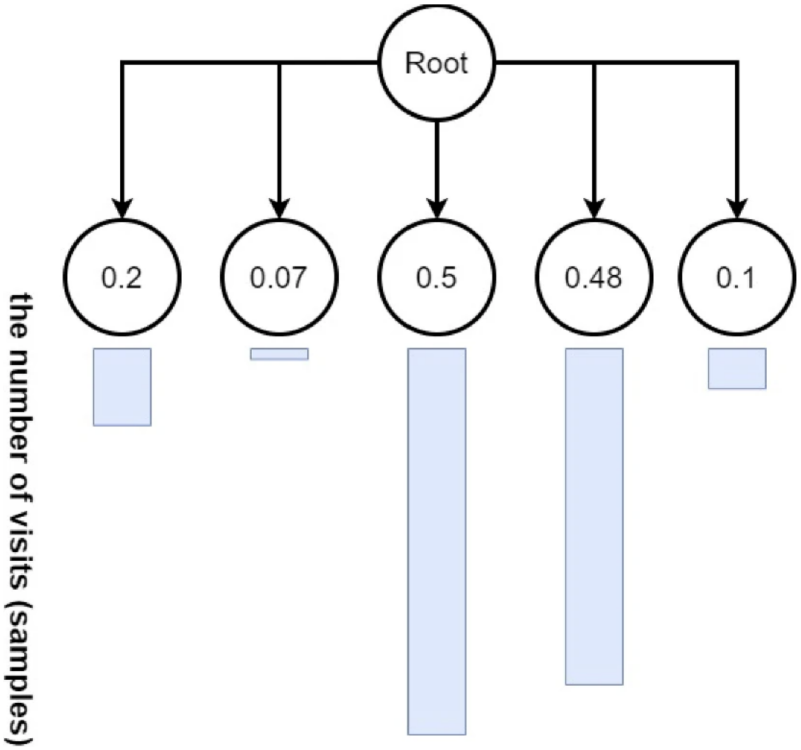


Figure 6.3.1: The root denotes the starting state. The average scores [Q(s, a)] of actions leading to next states are shown inside the circles. The bars denote how many simulations started from a particular action. This is an abstract example—not taken from any particular game.
[42]

6.4 Disadvantages of MCTS

MCTS is not a silver bullet when it comes to solving search problems. A few reasons are combinatorial complexity, sparse rewards or other kinds of inherent difficulty in the problem domain [42]. Other disadvantages are the high computation cost and the high memory requirements (both of which we encountered when implementing and executing MCTS for the current thesis), as well as lack of guarantees as MCTS does not guarantee finding an optimal solution [39]. Whenever the vanilla MCTS algorithm, i.e., implemented in its base unmodified form, fails to deliver the expected performance, it needs to be equipped with some kind of enhancements [42].

6.5 MCTS and Machine Learning

MCTS combined with deep reinforcement learning (RL) has emerged as a formidable force, exemplified by Google DeepMind's AlphaGo, as detailed in Silver et al. [40]. This groundbreaking development not only

revolutionized the game of Go but also made a significant impact on artificial intelligence (AI) as a whole. It's worth noting that MCTS has found applications in a wide array of combinatorial games and even ventured into real-time video games [42].

This integration of MCTS and ML not only enhances strategic decision-making but also opens up possibilities across diverse domains. AlphaGo's success story has ignited a wave of innovation, inspiring numerous approaches that integrate MCTS with machine learning (ML) models. Some of these approaches are outlined below.

Policy Networks:

In many applications, deep neural networks, known as policy networks, can be trained to estimate the likelihood of taking specific actions in a given state. These policy networks can be used within the MCTS algorithm to guide action selection. During the selection phase of MCTS, instead of using purely random or heuristic-based selection, the policy network can suggest actions that are more likely to be promising based on a policy function. The policy, denoted by p(a|s) informs which action—a—should be chosen given state s. A policy can be deterministic or stochastic. It is often modelled as a probability distribution of actions in a given state [42]. AlphaGo, used a policy network to guide its Monte Carlo simulations in the game of Go [40].

Value Networks:

In addition to policy networks, value networks can be employed to estimate the expected outcome or value of a particular state. These networks help in the evaluation phase of MCTS, using a function $v^*(s)$ that approximates the outcome of the game in a given state s. A perfect (non-approximate) value function is often used when solving a game, e.g. Checkers by Schaeffer et al. [34]. This is a similar concept to the state evaluation function. The use of them can be technically the same, however, the term "state evaluation function" is usually used when the function is static, whereas value function is self-improving through ML. Value networks are particularly useful in domains where it is challenging to perform exhaustive simulations [42].

Rollout Policies:

Rollouts (simulations) are an essential part of MCTS, but they can be improved by using learned policies instead of random actions. Machine learning techniques can be used to train rollout policies that guide the simulations. These policies may be trained to explore the state space more effectively, leading to better estimates of node values during the selection process. Used in [35]

Experience Replay:

In some cases, MCTS can benefit from experience replay mechanisms commonly used in deep reinforcement learning. Experience replay involves storing and randomly sampling previously encountered states and actions into a replay memory [27] to train and improve the performance of neural networks. In this way an agent can mentally experience the effects of its actions without actually executing them [24]. This technique can help MCTS algorithms learn from past simulations, making them more efficient over time.

Domain-Specific Heuristics:

Machine learning can be employed to learn domain-specific heuristics or features that enhance MCTS. These heuristics can help in the selection and evaluation of nodes in the search tree. By incorporating learned features, MCTS can make more informed decisions during the tree traversal.

Adaptive Strategies:

Machine learning models can be used to adapt MCTS parameters dynamically based on the characteristics of the problem or environment. For example, the exploration-exploitation trade-off parameters, such as the exploration constant in the UCB formula [41], can be learned and adjusted over time.

Hybrid Approaches:

Some applications may benefit from hybrid approaches that combine MCTS with other machine learning methods like reinforcement learning. In such cases, MCTS may be used for policy optimization, while reinforcement learning techniques are employed to fine-tune the learned policies.

Chapter 7

Explore Predicted Tactics with MCTS

In all three machine learning prediction tools for guided proof synthesis presented above (ASTactic 5.1, TacTok 5.2, Passport 5.3), conventional search algorithms were utilized to explore the space of generated tactics proposed by the prediction models (DFS, iterative deepening etc.). During the proof generation process, the search algorithm at each step would sample a specific number of model generated tactics, according to the specified beam size, and continue the search.

For all three models a Beam of size 20 has been used during testing. This specific value was decided through a trial and error process when ASTactic was tested and was adopted for the testing of the rest of the models without further improvements. The beam size "controls the trade-off between speed and accuracy" [46]. A large beam width results in an expanded search space, thus increasing the success rate. The model is enabled to explore larger search space at the expense of a longer search runtime. It was noticed however that when the beam width would be set to values greater than 20, the success rate would drop. The explanation behind that is that the model gets "trapped" in an unpromising branch for too long, and the search times out.

Our proposal in this thesis is to replace the traditional search algorithms with the Monte Carlo Tree Search iterative search algorithm 6. We claim that the usage of MCTS in the space of generated tactics can reduce the search time required to identify the correct sequence of tactics that constitute a full proof of a theorem. Faster search will allow to increase the number of sampled tactics at each step of the search (beam), thus expanding the exploration space. The expansion of the exploration space can increase the overall number of proofs proved by the above models.

7.1 Implementation of MCTS

7.1.1 State definition

In order to perform our search we need to define the proof state - and based on that construct the nodes of our search tree. The proof state contains the proof environment (global context), the local context and the goals (focused and unfocused). The proof environment can be shared across states, but each state is defined by its local context and its goals.

In our implementation, each state object (tree node) contains the tactic (action) which generated it by being applied in its parent state, the "observation_result" which holds the state status (SUCCESS, ERROR etc.) and the script up to that state, which is the sequence of tactics applied from the root of the tree until that state. In order to perform the search it is necessary to also add have the parent node, the depth, the number of visits to this state, the total rewards for the state and the children of the node, if it has any. Lastly we have included some additional parameters which facilitated our implementation (timeouts, number of overall tactics used up to this point, reference to the proof environment and more).

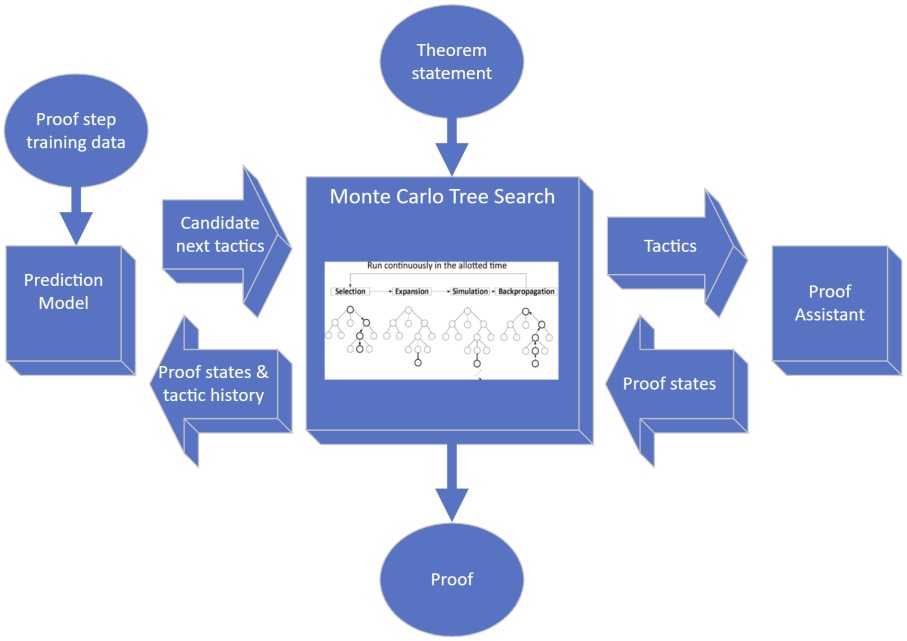


Figure 7.0.1: The architecture of machine learning prediction tools for guided proof synthesis with MCTS as a search component.

Terminal and non-terminal states

In the general case, a terminal state refers to a state in which the game or decision-making process ends. In our implementation, terminal states are the states in which no more tactics can be applied to the proof. We label the proof states of our search tree as terminal or non-terminal based on the feedback we get from the proof environment after the application of the step (tactic or Coq command) which generated this state. In our set of terminal states we also include the states MAX_TIME_REACHED and MAX_NUM_TACTICS_REACHED which are returned by the proof environment 5.1.2. These states are returned when the maximum time limit has been reached or when the maximum number of tactics has been reached. Including these states in the set of terminal states facilitated the implementation of the termination of the search for these two cases.

Here is a list of all the terminal states:

- ALREADY_SUCCEEDED
- ALREADY_FAILED
- MAX_TIME_REACHED
- MAX_NUM_TACTICS_REACHED
- ERROR
- GIVEN_UP
- SUCCESS

Non-terminal states are "PROVING" and "Initializing". Initializing was a state we added as part of the MCTS implementation to indicate that the search had not yet started for the current node.

7.1.2 Selection

During the selection step we need to pick the proof state that should be expanded next, among all the proof states we have reached so far.

For the selection process of our implementation we used the Upper Confidence Bound for Trees (UCT) formula which is a variant of the UCB formula, tailored for tree search 6.3.

We used the following formula to calculate the UCT value of each node:

$$UCT(i) = \frac{w_i}{n_i} + \sqrt{2\frac{\ln N_i}{n_i}}$$

Where:

- w_i is the total reward obtained from node i.
- n_i is the number of times node i has been visited.
- N_i is the number of times the parent of node i has been visited.
- We set the exploration parameter C to be equal to $\sqrt{2}$.

7.1.3 Expansion

In this stage, one or more new proof states are added to the tree, branching from a leaf node. These new nodes represent proof states that have been reached by applying a particular tactic to the previous state (represented by the leaf node).

In our implementation of the expansion stage we have to get the possible tactics (actions) which can be applied in the current proof state. The possible actions for the current state are returned by the trained model, after providing as input the environment of the proof, the local contexts and the goals that remain to be proved. The tactics are returned after calling the `beam_search` method of the model. This method calls the decoder which generates the candidate tactics for the current state. The number of generated tactics is specified by the `beam` parameter which we have mentioned above.

The second part of the expansion is to apply the candidate tactics in the current state and create the new states. This is done by providing the tactics one at a time to the proof environment 5.1.2 and getting the feedback on the applied actions. The feedback includes the progress of the proof (PROVING, ERROR etc.) and can also include the proof environment as well as the focused, background, shelved and given up goals, depending on the progress. The feedback can also contain the Coq exceptions, if the proof has reached an error.

The feedback returned from Coq through the proof environment is used to set up the new states.

7.1.4 Simulation

In this stage we perform a simulation or rollout from a new node to a terminal state following a random policy. We follow the same process for getting and applying the possible tactics at each state as described in the previous sections 7.1.3.

In our implementation of the simulation logic we are maintaining the proof scripts that are being generated during the rollout. In the case that all goals are proved during the rollout, the proof script containing the tactics applied in the path from the root to the terminal node in state "PROVED" is a proof of the theorem.

7.1.5 Backpropagation

In this stage we are simply updating the nodes that were visited during the simulation 6.2 with the appropriate rewards. We navigate back from the leaf node to the root updating the value and visit count of each node along the path.

7.1.6 Reward Policies

When MCTS is used in games like chess, it is usually combined with a component which can evaluate the states or, in this case, the chess positions during the simulation stage. And, although evaluating a chess

position as winning or losing is not trivial, there are extremely advanced software tools or machine learning models that surpass the human evaluation. Unfortunately, evaluating if a proof state is "good" or "bad", or how close it is to the proof is not something we can do. Being able to do that in a formal way would mean that we would be able to reach the actual proof of the theorem we are trying to prove.

The simulation stage of MCTS is aiming to do just that - apply series of tactics with a random policy after a given proof state and evaluate the proof state based on these results. Since we are not able to evaluate non-terminal states reached during the rollout, our first approach is to wait for the rollout to reach one of the terminal states and assign them the corresponding reward.

Our first reward policy is:

- Return a reward of +100 if we have reached a state which has status 'ALREADY_SUCCEEDED' or 'SUCCESS'.
- Return a reward of -1 if we have reached a state which has status which indicates failure - ALREADY_FAILED, ERROR, GIVEN_UP.

The above policy has the benefits of being very simple to apply and of promoting an exploration to the terminal nodes, thus searching for deeper proofs. Furthermore, it manages to prevent the search of being trapped in unpromising branches, since they will be negatively rewarded. On the other hand, the positive rewards are very sparse. More importantly, once we reach a terminal state which will be returning a positive reward we have essentially reached the end of our search. The proof script from the root node to the terminal node is a sequence of tactics which prove the current theorem.

Another approach to discourage the exploration of unpromising branches was to give negative rewards when reaching deeper nodes. The average number of steps in the proofs of CoqGym is 9.1 [46].

Our reward policy based on the exploration depth is:

- Return a reward of +100 if we have reached a state which has status 'ALREADY_SUCCEEDED' or 'SUCCESS'.
- Return a reward of -1 if we have reached a state which has status which indicates failure - ALREADY_FAILED, ERROR, GIVEN_UP.
- Return a reward of $-\lambda_d \cdot \lfloor \frac{D}{5} \rfloor$, $D \in \{5k \mid k \in \mathbb{Z}\}$ where λ_d is a constant on which we experimented and D is the depth that we have reached.

The above approach still does not address the issue of sparse positive rewards. In order to do that we need to find a way to evaluate if the application of a certain state leads to a better state than the previous one.

In this point it is worth mentioning that in Coq, when a tactic is applied, it typically applies only to the first goal in the foreground, not to all goals. The goals are arranged in a stack, and most tactics apply only to the current goal at the top of the stack (the "focused" goal).

When a tactic is applied to a goal in Coq, a few different things might happen based on the tactic used:

- The goal is solved: If the tactic successfully solves the goal, that particular subgoal disappears because it has been proved.
- The goal is transformed: If the tactic partially solves the goal or transforms it in some way, the original subgoal disappears and new subgoal(s) appear, reflecting the remaining work to be done. A single goal might be replaced by multiple subgoals if the applied tactic breaks it into smaller pieces.
- The goal is unchanged: If the tactic is not applicable or unsuccessful, the goal will remain unchanged.
- Additional context: The tactic might also add new definitions, variables, or hypotheses to the context, affecting the subsequent proof.

Here is an example of how goals can transform with tactic application.

Listing 7.1: Example of goal transformation

```
Goal forall x y : nat, x + y = y + x.
```

```
Proof.
  intros x y.
  (*
     - `intros` moves universally quantified variables (x and y)
     - from the goal to the context.
     - The goal transforms to `x + y = y + x` without
     - the forall quantifiers.
  *)
  rewrite Nat.add_comm.
  (*
     - `rewrite` uses the provided lemma (`Nat.add_comm`)
     - to change the goal.
     - Here, it solves the goal, so no subgoals remain,
     - and we're done.
  *)
Qed.
```

With these in mind we can see that when we apply a tactic during our search, if the number of focused goals decreases then the tactic has actually solved one of the goals, and we have reached an improved state from the previous on we were in.

Our reward policy based on the reduction of focused goals is:

- Return a reward of +100 if we have reached a state which has status 'ALREADY_SUCCEEDED' or 'SUCCESS'.
- Return a reward of -1 if we have reached a state which has status which indicates failure - ALREADY_FAILED, ERROR, GIVEN_UP.
- Return a reward of $+\lambda_g$ every time the number of focused goals decreases where λ_g is a constant on which we experimented.

Note that we do not penalize the cases in which the number of goals increases or remains unchanged, since this is not necessarily a negative outcome.

In the following section we will be presenting the results of our experiments.

Chapter 8

Experiments

In this chapter we will present the results of our experiments on combining MCTS with the existing deep learning models which generate tactics as programs. We will be presenting the setup of our experiments and then we will analyze the results that we got for each of the reward policies which we analyzed in the previous chapter 7.1.6.

8.1 Setup

8.1.1 Models

We conducted our experiments on pre-trained and shared models of ASTactic and TacTok. Passport models where not publically available, so we had to run the training for the models ourselves. We trained Passport with ASTactic, Tac and Tok, so that we would be able to compare the performance of MCTS combined to these models. The detailed results of running the models with the default search (DFS) are presented in section 8.9.

8.1.2 Benchmark

As described in 3.2, the CoqGym benchmark includes 123 open-source Coq projects, split into three sets. For our evaluation, we trained on 97 projects (containing a total of 57,719 theorems) and synthesized proofs for 26 projects (containing a total of 10,782 theorems). We follow First et al. [11] and Sanchez-Stern, A. et al. [33] and exclude the coq-library-undecidability project from the evaluation. As it is mention by Sanchez-Stern, A. et al. [33] "TacTok's evaluation [First et al. 2020] was unable to reproduce prior results for ASTactic's performance [Yang and Deng 2019] [46] on that project due to internal Coq errors when processing the proof scripts".

8.1.3 Machines

We ran this paper's experiments GPUs for training and CPUs for synthesizing proofs. We were fortunate that the process of synthesizing proofs could be done on CPUs since, as it is mentioned by Yang et al. [46], the bottleneck of the process is found in the interaction with Coq and the execution of tactics, not their generation.

For training Passport we used a machine with one NVIDIA T4 GPU with 16 GB of memory.

For synthesizing the proofs we used:

- One 12th Generation Core i7 Processor @ 2.40GHz, 32GB RAM and 512GB local SSD disk.
- One Standard_D2s_v3 Azure VM with Xeon E5-2673 v3 @ 2.40GHz, 8GB RAM and 50 GB SSD disk.

8.1.4 Parameters

In our experiments we used the experimental parameters followed by Yang and Deng 2019 [46], First et al. [11] and Sanchez-Stern, A. et al. [33].

They synthesize proofs by setting a timeout, after which, if the search has failed to reach Qed it fails. The timeout was set to 10 minutes by all of them, and this is the value we used in our experiments.

For training passport we also used the setup described by Sanchez-Stern, A. et al. [33].

We set:

- default category vocabulary threshold: 200
- byte-pair merge threshold: 4096
- default vector dimension for term, grammar and terminal/non-terminal symbol embeddings: 128
- dimension of LSTM controller: 128

8.2 Determining the Beam size

Due to the huge time expense of the testing process, we were not able to test a large range of depths across the entire testing dataset. We run tests for depth 20, 25 and 30. We chose to select projects with a high count of proofs in which the models had either too many or too few successful proofs. The projects that we picked were: dblib, UnifySL, PolTac and verdi-raft.

Upon experimentation we saw that increasing the beam to 25 in MCTS gave improved results compared to a beam of 20. Increasing the beam to 30 did not add any value. This is probably an indication of the fact that the additional tactics suggested by the models are not accurate enough to generated more proofs. The results presented below, have been generated using a beam of 25.

8.3 Reward on terminal states policy

When doing the evaluation on the models we trained for Passport we were not able to replicate the exact results which are mentioned in Sanchez-Stern, A. et al. [33]. For this reason we will compare our results with the results which we were able to replicate, but we will also be showing those which are mentioned in literature.

8.3.1 MCTS & DFS comparison across models

In figure 8.3.1 we can see the performance of MCTS executed with the "reward on terminal states" policy 7.1.6.

We observe that when applying MCTS to ASTactic and TacTok (Tac & Toc models combined), DFS outperforms our search method, but not significantly.

When we compare MCTS and DFS combined with the Passport models, we see that DFS outperforms our search according to literature results [33]. On the other hand, MCTS outperforms DFS on the evaluation results we got for the Passport models we trained ourselves. Across all models, we can see that with MCTS all three models were able to prove 29 more theorems than with DFS.

8.3.2 New theorems proven with MCTS

We will be following the approach taken by First et al. [11] and by Sanchez-Stern, A. et al. [33] to evaluate how MCTS applied to the model suggestions leads to new theorems proven.

In table 8.1 we see that when we calculate the union of the proofs synthesized by all the Passport-enhanced models combined with MCTS, we have 33 new proofs. When we compare the number of proofs synthesized by ASTactic combined with MCTS we were able to prove 8 new theorems, despite the fact that we lost some

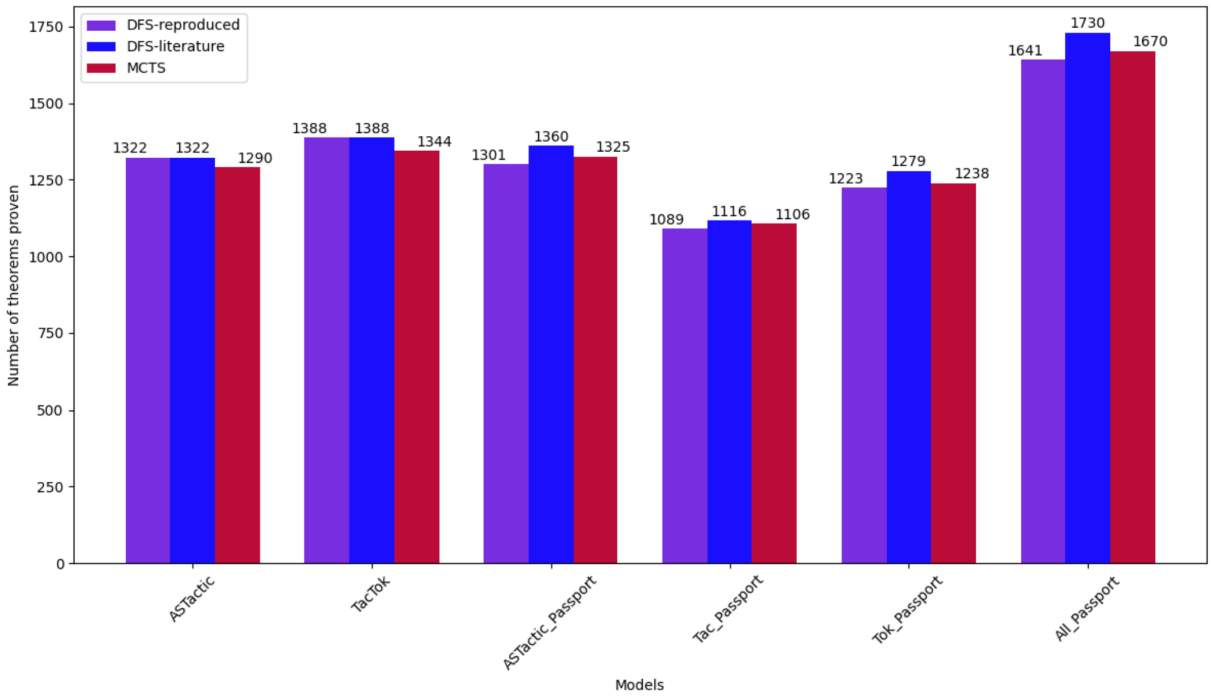


Figure 8.3.1: Number of theorems proved for each model by using DFS and MCTS with reward on terminal states policy. The purple bars represent baseline models based on ASTactic, TacTok, ASTactic + Passport, Tok + Passport and Tac + Passport. The blue bars represent the baseline mentioned in literature [33], [11]. The red bars represent the results of running the models with MCTS. The bars labeled "AllPassport", is the number of theorems successfully proven using either DFS or MCTS by at least one of the Passport-enhanced models.

proofs which were found with DFS. We also notice that in the ASTactic+Passport model, we were able to prove 27 new theorems, although we only proved 24 more theorems in absolute values. This again shows that we lost some of the proof that were proved with DFS.

Overall we can see that the results that we get with the application of MCTS with the 'reward terminal states' policy to the existing models are very similar to the results that we get from DFS. This is not surprising. The scarcity of positive rewards in MCTS, and the fact that when we finally get a positive reward we have essentially solved our search problem, "reduce" MCTS to a traditional search algorithm.

Model	ASTactic	TacTok	ASTactic+Passport	Tac+Passport	Tok+Passport	*+Passport
DFS Repro	1322	1388	1301	1089	1223	1641
MCTS Repro	1290	1344	1325	1106	1238	1670
Difference	-32	-44	+24	+27	+15	+29
New Theorems	8 (0.07%)	0 (0%)	27 (0.25%)	18 (0.17%)	16 (0.15%)	33 (0.31%)

Table 8.1: Number of theorems proven across models with MCTS when using the reward on terminal states policy and a beam of 25. The percentages are calculated over the total 10782 of the evaluation dataset.

As it is also mentioned in literature [46], the average length of theorem proofs is small - less than 10 tactics. The average length of the theorems proved across models with MCTS was 6.7. In comparison, the average length of the proofs in the testing dataset is 12.5. This is not surprising - as it validates the expectation that the longer the theorem, the harder it will be to generate its proof. The average length of the 33 new theorems that were proved was 7.8 tactics. This is a bit higher than the average length of all the proofs which we mention above. This indicates that MCTS facilitates the proving of longer theorems.

8.4 Reward on depth policy

8.4.1 Determining policy parameters

As mentioned in the previous chapter 7.1.6, the reward on depth policy was defined as follows:

- Return a reward of +100 if we have reached a state which has status 'ALREADY_SUCCEEDED' or 'SUCCESS'.
- Return a reward of -1 if we have reached a state which has status which indicates failure - ALREADY_FAILED, ERROR, GIVEN_UP.
- Return a reward of $-\lambda_d \cdot \lfloor \frac{D}{5} \rfloor$, $D \in \{5k \mid k \in \mathbb{Z}\}$ where λ_d is a constant on which we experimented and D is the depth that we have reached.

In order to determine the value of λ_d we have followed the same approach we used for determining the beam 8.2. We run tests for $\lambda_d \in \{0.5, 0.75, 1\}$ in projects dblib, UnifySL, PolTac and verdi-raft.

The results of evaluating the proofs for these projects did not have significant differences for the different values of λ_d. We chose to use a value of $\lambda_d = 0.5$ to not significantly penalize searching in deeper space.

8.4.2 MCTS & DFS comparison across models

In figure 8.4.1 we can see the performance of MCTS executed with the "reward on depth" policy 7.1.6.

We observe that when applying MCTS to ASTactic and TacTok (Tac & Toc models combined), DFS outperforms our search method proving 65 theorems more.

When we compare MCTS and DFS combined with the Passport models, we see that DFS outperforms our search according to literature results [33] by 138 theorems. DFS also outperforms MCTS when comparing with the evaluation results we got for the Passport models we trained ourselves. We can see that with DFS all three models combined were able to prove 49 more theorems than with MCTS.

8.4.3 New theorems proven with MCTS

In table 8.2 we see that when we calculate the union of the proofs synthesized by all the Passport-enhanced models combined with MCTS, we have 17 new proofs.

We did not get any new proofs when combining MCTS with ASTactic and TacTok.

Model	ASTactic	TacTok	ASTactic+Passport	Tac+Passport	Tok+Passport	*+Passport
DFS Repro	1322	1388	1301	1089	1223	1641
MCTS Repro	1257	1282	1244	1027	1151	1592
Difference	-65	-106	-57	-62	-72	-49
New Theorems	0	0	14 (0.13%)	11 (0.1%)	11 (0.1%)	17 (0.16%)

Table 8.2: Number of theorems proven across models with MCTS when using the reward on depth policy and a beam of 25. The percentages are calculated over the total 10782 of the evaluation dataset.

Applying the "reward on depth policies" for a beam of 25 did not improve our search results. We have the following assumptions on the cause behind that.

- The first observation is that a negative reward on depth could hinder the discovery of deeper proofs. As we have seen above, the average length of proofs generated across models is 6.7 tactics per proof. This indicates that the models are more effective in synthesizing shorter proofs, and that DFS is able to effectively locate those proofs, given the search parameters specified (beam of 20, 10 minute timeout). Giving negative reward for reaching deeper states did not improve the discovery of new shorter proofs, and prevented the search from finding longer proofs which were discovered by the "reward on terminal states" policy.

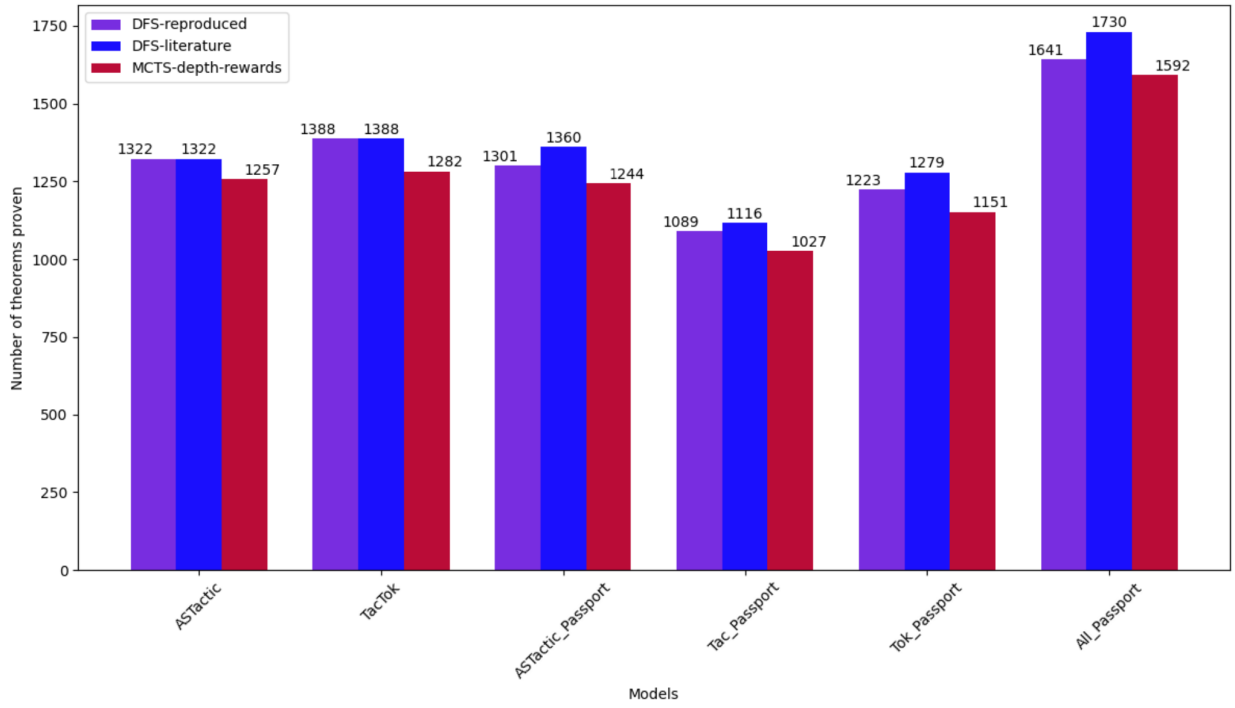


Figure 8.4.1: Number of theorems proved for each model by using DFS and MCTS with reward on depth policy. The purple bars represent baseline models based on ASTactic, TacTok, ASTactic + Passport, Tok + Passport and Tac + Passport. The blue bars represent the baseline mentioned in literature [33], [11]. The red bars represent the results of running the models with MCTS. The bars labeled "AllPassport", is the number of theorems successfully proven using either DFS or MCTS by at least one of the Passport-enhanced models.

- The second observation is that the evaluation cost prevented us from fine-tuning the MCTS combined with the "reward on depth" policy. It is possible that by trying more combinations of λ_d and *beam* parameters, we could have gotten better results. For instance, increasing the λ_d parameter to values greater than 1, and widening to values greater than 30, could encourage the discovery of new shorter proofs, despite hindering the discovery of deep proofs. This is something we will also discuss in our conclusions and the limitations that we faced for our experiments 8.7.

8.5 Reward on goal reduction policy

8.5.1 Determining policy parameters

As mentioned in the previous chapter 7.1.6, the reward policy based on the reduction of focused goals was defined as follows:

- Return a reward of +100 if we have reached a state which has status 'ALREADY_SUCCEEDED' or 'SUCCESS'.
- Return a reward of -1 if we have reached a state which has status which indicates failure - ALREADY_FAILED, ERROR, GIVEN_UP.
- Return a reward of $+\lambda_g$ every time the number of focused goals decreases where λ_g is a constant on which we experimented.

This was the last policy that we considered testing, and we did not have time to complete our experiments by the time this thesis was finalized.

To determine the value of λ_g we follow the same approach we used for determining the beam 8.2 and λ_d. We run tests for $\lambda_g \in \{0.5, 1, 2\}$ in projects dblib, UnifySL, PolTac and verdi-raft. The results that we got from these experiments were significantly (around 20%) lower than the results we got from the previous reward policies which we used in MCTS. An explanation to that is that during the simulation phase, many states are highly rewarded as a result of tactics reducing the number of focused goals in their branches, but they are not leading to the proof branches of the tree.

To examine this deeper we need to remember that with this policy we are trying to recognise the positive effect of applying a tactic in a given state, but we reward the resulting states and not the tactics. This can lead in situations where states in unpromissing branches of the search will be highly rewarded during the search.

8.5.2 Possible improvements of the reward policy

We would need more time to fine tune the policy parameters (λ_g, reward on proof, rewards on failure) and the beam in order to exploit possible benefits of this policy. It was something that we did not have time to do due to the computational cost of the evaluation phase.

One approach which could also add value to the search is to combine MCTS with a cache of "good" model generated tactics - tactics that have contributed to the reduction of focused goals during search, which will be sampled and rewarded during the search. An extra step to that would be to have such a cache for every focused goal of the theorem.

8.6 Conclusion

In our experiments we tried replacing the traditional search components of machine learning prediction tools for guided proof synthesis with Monte Carlo Tree Search. Our experimentation showed that using MCTS can produce results very close to the top literature results. Furthermore, our experiments produced proofs which were not being generated be the traditional search methods. Still, we were not able to outperform the best results as found in literature.

From the three MCTS reward policies we used the "reward on terminal states" policy was the most effective. When combined with our Passport trained models, MCTS was able to find more proofs compared to the DFS results we were able to reproduce. Still it was not able to outperform the best results mentioned in literature. The scarcity of positive rewards in this policy makes MCTS behave very similarly to traditional search algorithms like DFS, thus giving us very similar results.

We attempted to leverage the benefits of MCTS by introducing the "reward on depth" policy and the "reward on goal reduction" policy. The "reward on depth" policy showed potential, but was not as effective in the discovery of longer proofs. The experimentation on the "reward on goal reduction" policy is still ongoing by the time this thesis is being completed, since fine+tuning it was harder than the previous methods.

8.7 Limitations

The major limitation we had for our experimentation was the cost of the evaluation experiments. Doing a full evaluation run for a set of parameters across all 26 Coq projects could take several weeks with the hardware we had available. Due to this reason, as mentioned in the previous sections 8.2, we could only test a limited range of potential values for the different parameters. Additionally, we performed the testing in a subset of the evaluation set. Hence, we cannot be certain for their optimality.

Due to lack of hardware we were also not able to experiment with the models and retrain them. The hardware cost is also raised as a concern by most researchers working on the domain. Sanchez-Stern, A. et al. [33] for instance mention that "expensive hardware is required to train these models, limiting who can develop them, and often requiring the use of shared clusters which can slow development".

Another limitation was the complexity of preparing the environment to run the experiments.

8.8 Future Directions

Fine-tuning the reward policies would be the first step going forward. As already explained, this is a task that can have a big computational cost but MCTS has already given promising results with very little experimentation on its parameters.

Our approach to fine-tuning would be to test different combinations of the reward policy parameters and the search beam. Specifically it is worth trying even larger values of the Beam combined with the reward on depth policy and test if this approach can generate new proofs and not necessarily more proofs than previous search methods or combination. The goal from this approach would be to test if a greater beam and a search focused on small depths could benefit the discovery of new short proofs. The second step would be to fine tune the "reward on goal reduction" policy. This policy could give significant benefits since it will increase the number of positive rewards given in MCTS. We would also like to test the above policies in combination with CoqHammer - where its application would be represented as applying the hammer tactic. Finally, combining the reward on depth policy with the reward on goal reduction policy would also be interesting to test.

Another possibility for future work would be to replace the LSTM components of Graph Neural Networks and evaluate if incorporating Graph Neural Networks (GNNs) enhances the capability to identify and exploit intricate structures inherent in proof strategies. This approach has already been explored by Paliwal et al. [31]. GNNs, are inherently capable of managing relational data and propagating information across graph-structured entities [45], and could facilitate a more nuanced understanding and navigation through the logical space of proofs. Given that proof synthesis could be conceived as a graph where each theorem and lemma serve as nodes connected through logical entailment or dependencies, GNNs might be uniquely positioned to learn and leverage these structural relations effectively.

Another option would be to retrain the policy networks that we have available (ASTactic, TacTok, Passport) with MCTS as was famously done for Alpha Go [40].

Finally, a direction which could significantly improve the performance of all the models would be to enrich CoqGym with information about which premises of the proof context and the environment are being used for the proof. This is also mentioned by Yang et al. [46] as a complex endeavor which could have significant benefits. Having such information and including it in the vector encoding during training could improve the performance of the trained models.

Table 8.3: The evaluation results of running TacTok and ASTactic models with MCTS using the reward on terminal state policy 8.3

Name	TacTok-MCTS	ASTactic-MCTS	Dataset Total
weak-up-to	18 (12.9%)	20 (14.4%)	139
buchberger	72 (9.9%)	70 (9.7%)	725
jordan-curve-theorem	21 (3.3%)	19 (3.0%)	628
dblib	44 (24.4%)	41 (22.8%)	180
disel	83 (13.1%)	77 (12.1%)	634
zchinese	5 (11.6%)	5 (11.6%)	43
zfc	31 (13.1%)	32 (13.5%)	237
dep-map	11 (25.6%)	9 (20.9%)	43
chinese	35 (26.7%)	31 (23.7%)	131
UnifySL	180 (18.6%)	182 (18.8%)	968
hoare-tut	3 (16.7%)	1 (5.6%)	18
huffman	28 (8.9%)	25 (8.0%)	314
PolTac	112 (30.9%)	116 (32.0%)	363
angles	4 (6.5%)	4 (6.5%)	62
coq-procrastination	6 (75.0%)	5 (62.5%)	8
tree-automata	104 (12.6%)	95 (11.5%)	828
coquelicot	100 (6.8%)	95 (6.5%)	1467
fermat4	10 (7.7%)	13 (10.0%)	130
demos	51 (75.0%)	49 (72.1%)	68
coqoban	0 (0.0%)	0 (0.0%)	2
goedel	59 (9.7%)	53 (8.7%)	606
verdi-raft	113 (5.3%)	111 (5.2%)	2127
verdi	44 (8.6%)	37 (7.2%)	514
zorns-lemma	12 (8.1%)	10 (6.7%)	149
coqrel	183 (71.5%)	179 (69.9%)	256
fundamental-arithmetic	15 (10.6%)	11 (7.7%)	142
Totals	1344 (12.5%)	1290 (12.0%)	10782

Table 8.4: The evaluation results of running ASTactic+Passport, Tac+Passport, Tok+Passport models with MCTS using the reward on terminal state policy 8.3.

Name	ASTactic-Passport-MCTS	Tac-Passport-MCTS	Tok-Passport-MCTS	Dataset Total
weak-up-to	23 (16.5%)	12 (8.6%)	14 (10.1%)	139
buchberger	70 (9.7%)	70 (9.7%)	69 (9.5%)	725
jordan-curve-theorem	19 (3.0%)	16 (2.5%)	22 (3.5%)	628
dblib	41 (22.8%)	41 (22.8%)	36 (20.0%)	180
disel	83 (13.1%)	63 (9.9%)	80 (12.6%)	634
zchinese	5 (11.6%)	4 (9.3%)	4 (9.3%)	43
zfc	33 (13.9%)	28 (11.8%)	28 (11.8%)	237
dep-map	9 (20.9%)	7 (16.3%)	8 (18.6%)	43
chinese	31 (23.7%)	27 (20.6%)	33 (25.2%)	131
UnifySL	189 (19.5%)	126 (13.0%)	168 (17.4%)	968
hoare-tut	1 (5.6%)	3 (16.7%)	3 (16.7%)	18
huffman	25 (8.0%)	22 (7.0%)	23 (7.3%)	314
PolTac	118 (32.5%)	87 (24.0%)	110 (30.3%)	363
angles	4 (6.5%)	3 (4.8%)	4 (6.5%)	62
coq-procrastination	5 (62.5%)	5 (62.5%)	6 (75.0%)	8
tree-automata	96 (11.6%)	83 (10.0%)	99 (12.0%)	828
coquelicot	95 (6.5%)	77 (5.2%)	88 (6.0%)	1467
fermat4	13 (10.0%)	9 (6.9%)	8 (6.2%)	130
demos	50 (73.5%)	49 (72.1%)	52 (76.5%)	68
coqoban	0 (0.0%)	0 (0.0%)	0 (0.0%)	2
goedel	56 (9.2%)	57 (9.4%)	58 (9.6%)	606
verdi-raft	117 (5.5%)	97 (4.6%)	108 (5.1%)	2127
verdi	37 (7.2%)	37 (7.2%)	42 (8.2%)	514
zorns-lemma	10 (6.7%)	10 (6.7%)	7 (4.7%)	149
coqrel	184 (71.9%)	163 (63.7%)	159 (62.1%)	256
fundamental-arithmetic	11 (7.7%)	10 (7.0%)	9 (6.3%)	142
Totals	1325 (12.3%)	1106 (10.3%)	1238 (11.5%)	10782

8.9 Model Benchmarks

Name	TacTok	ASTactic & CoqHammer	ASTactic	Dataset Total
weak-up-to	18 (12.9%)	36 (25.9%)	23 (16.5%)	139
buchberger	76 (10.5%)	192 (26.5%)	70 (9.7%)	725
jordan-curve-theorem	22 (3.5%)	168 (26.8%)	19 (3.0%)	628
dblib	45 (25.0%)	67 (37.2%)	41 (22.8%)	180
disel	89 (14.0%)	194 (30.6%)	83 (13.1%)	634
zchinese	5 (11.6%)	13 (30.2%)	5 (11.6%)	43
zfc	33 (13.9%)	70 (29.5%)	33 (13.9%)	237
dep-map	11 (25.6%)	16 (37.2%)	9 (20.9%)	43
chinese	35 (26.7%)	58 (44.3%)	31 (23.7%)	131
UnifySL	180 (18.6%)	367 (37.9%)	189 (19.5%)	968
hoare-tut	5 (27.8%)	6 (33.3%)	1 (5.6%)	18
huffman	28 (8.9%)	81 (25.8%)	25 (8.0%)	314
PolTac	112 (30.9%)	308 (84.8%)	118 (32.5%)	363
angles	4 (6.5%)	15 (24.2%)	4 (6.5%)	62
coq-procrastination	6 (75.0%)	5 (62.5%)	5 (62.5%)	8
tree-automata	111 (13.4%)	311 (37.6%)	96 (11.6%)	828
coquelicot	100 (6.8%)	299 (20.4%)	95 (6.5%)	1467
fermat4	10 (7.7%)	47 (36.2%)	13 (10.0%)	130
demos	53 (77.9%)	55 (80.9%)	50 (73.5%)	68
coqoban	0 (0.0%)	0 (0.0%)	0 (0.0%)	2
goedel	67 (11.1%)	128 (21.1%)	53 (8.7%)	606
verdi-raft	121 (5.7%)	351 (16.5%)	117 (5.5%)	2127
verdi	47 (9.1%)	127 (24.7%)	37 (7.2%)	514
zorns-lemma	12 (8.1%)	21 (14.1%)	10 (6.7%)	149
coqrel	183 (71.5%)	191 (74.6%)	184 (71.9%)	256
fundamental-arithmetic	15 (10.6%)	41 (28.9%)	11 (7.7%)	142
Totals	1388 (12.9%)	3167 (29.4%)	1322 (12.3%)	10782

Table 8.5: Success rates, as evaluated on the CoqGym benchmark by Yang and Deng 2019 [46], the three tools, TacTok [11], ASTactic [46], and CoqHammer [Czajka and Kaliszyk 2018] [9].

Name	CoqHammer	Tac	Tok	Dataset Total
weak-up-to	30 (21.6%)	12 (8.6%)	12 (8.6%)	139
buchberger	166 (22.9%)	70 (9.7%)	65 (9.0%)	725
jordan-curve-theorem	165 (26.3%)	16 (2.5%)	22 (3.5%)	628
dblib	55 (30.6%)	41 (22.8%)	35 (19.4%)	180
disel	185 (29.2%)	63 (9.9%)	72 (11.4%)	634
zchinese	12 (27.9%)	4 (9.3%)	4 (9.3%)	43
zfc	64 (27.0%)	28 (11.8%)	28 (11.8%)	237
dep-map	14 (32.6%)	7 (16.3%)	8 (18.6%)	43
chinese	56 (42.7%)	27 (20.6%)	30 (22.9%)	131
UnifySL	303 (31.3%)	126 (13.0%)	153 (15.8%)	968
hoare-tut	6 (33.3%)	3 (16.7%)	3 (16.7%)	18
huffman	74 (23.6%)	22 (7.0%)	21 (6.7%)	314
PolTac	289 (79.6%)	87 (24.0%)	110 (30.3%)	363
angles	15 (24.2%)	3 (4.8%)	4 (6.5%)	62
coq-procrastination	3 (37.5%)	5 (62.5%)	6 (75.0%)	8
tree-automata	292 (35.3%)	83 (10.0%)	96 (11.6%)	828
coquelicot	273 (18.6%)	77 (5.2%)	78 (5.3%)	1467
fermat4	47 (36.2%)	9 (6.9%)	8 (6.2%)	130
demos	54 (79.4%)	49 (72.1%)	52 (76.5%)	68
coqoban	0 (0.0%)	0 (0.0%)	0 (0.0%)	2
goedel	120 (19.8%)	57 (9.4%)	58 (9.6%)	606
verdi-raft	337 (15.8%)	99 (4.7%)	97 (4.6%)	2127
verdi	120 (23.3%)	36 (7.0%)	33 (6.4%)	514
zorns-lemma	19 (12.8%)	8 (5.4%)	10 (6.7%)	149
coqrel	179 (70.0%)	174 (68.0%)	177 (69.1%)	256
fundamental-arithmetic	39 (27.5%)	11 (7.7%)	10 (7.0%)	142
Totals	2850 (26.4%)	1097 (10.2%)	1159 (10.7%)	10782

Table 8.6: Continuation of the previous table. 8.5

Table 8.7: Success rates as evaluated on Passport benchmark trained with ASTatic, Tac and Tok. Note that these are the results that we managed to replicate. The performance of the model is different to the performance found in Sanchez-Stern, A. et al. [33]

Name	ASTactic+Passport	Tac+Passport	Tok+Passport	Dataset Total
weak-up-to	23 (16.5%)	12 (8.6%)	14 (10.1%)	139
buchberger	70 (9.7%)	70 (9.7%)	69 (9.5%)	725
jordan-curve-theorem	17 (2.7%)	16 (2.5%)	22 (3.5%)	628
dblib	41 (22.8%)	41 (22.8%)	36 (20.0%)	180
disel	78 (12.3%)	63 (9.9%)	80 (12.6%)	634
zchinese	5 (11.6%)	4 (9.3%)	4 (9.3%)	43
zfc	33 (13.9%)	23 (9.7%)	23 (9.7%)	237
dep-map	9 (20.9%)	7 (16.3%)	8 (18.6%)	43
chinese	31 (23.7%)	27 (20.6%)	33 (25.2%)	131
UnifySL	182 (18.8%)	123 (12.7%)	162 (16.7%)	968
hoare-tut	1 (5.6%)	3 (16.7%)	3 (16.7%)	18
huffman	25 (8.0%)	21 (6.7%)	23 (7.3%)	314
PolTac	118 (32.5%)	87 (24.0%)	110 (30.3%)	363
angles	4 (6.5%)	3 (4.8%)	4 (6.5%)	62
coq-procrastination	5 (62.5%)	5 (62.5%)	6 (75.0%)	8
tree-automata	96 (11.6%)	83 (10.0%)	99 (12.0%)	828
coquelicot	95 (6.5%)	77 (5.2%)	88 (6.0%)	1467
fermat4	13 (10.0%)	8 (6.2%)	8 (6.2%)	130
demos	50 (73.5%)	49 (72.1%)	52 (76.5%)	68
coqoban	0 (0.0%)	0 (0.0%)	0 (0.0%)	2
goedel	56 (9.2%)	56 (9.2%)	58 (9.6%)	606
verdi-raft	112 (5.3%)	97 (4.6%)	104 (4.9%)	2127
verdi	37 (7.2%)	37 (7.2%)	42 (8.2%)	514
zorns-lemma	10 (6.7%)	10 (6.7%)	7 (4.7%)	149
coqrel	179 (69.9%)	157 (61.3%)	159 (62.1%)	256
fundamental-arithmetic	11 (7.7%)	10 (7.0%)	9 (6.3%)	142
Totals	**1301 (12.1%)**	**1089 (10.1%)**	**1223 (11.3%)**	**10782**

Chapter 9

Bibliography

[1] Arias, E. J. G. "SerAPI: Machine-Friendly, Data-Centric Serialization for COQ". In: 2016. URL:

[2] Barras, B. et al. "The Coq Proof Assistant Reference Manual : Version 6.1". In: (June 1997).

[3] Ben-Yelles, C. "Type Assignment in the Lambda-Calculus: Syntax and Semantics". PhD thesis. Department of Pure Mathematics, University College of Swansea, Sept. 1979.

[4] Blanchette, J. C. et al. "Hammering towards QED". In: *Journal of Formalized Reasoning* 9.1 (2016), pp. 101–148.

[5] Buzzard, K. *The future of mathematics*. CRNS-Imperial Lecture. 2019.

[6] Cho, K. et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL:

[7] *Coq Package Index*. Accessed: October 9, 2023.

[8] Czajka, L. and Kaliszyk, C. "Goal Translation for a Hammer for Coq (Extended Abstract)". In: *Proceedings First International Workshop on Hammers for Type Theories, HaTT@IJCAR 2016, Coimbra, Portugal, July 1, 2016*. Ed. by J. C. Blanchette and C. Kaliszyk. Vol. 210. EPTCS. 2016, pp. 13–20. DOI: 10.4204/EPTCS.210.4. URL:

[9] Czajka, Ł. and Kaliszyk, C. "Hammer for Coq: Automation for Dependent Type Theory". In: *Journal of Automated Reasoning* 61.1 (June 2018), pp. 423–453. ISSN: 1573-0670. DOI: 10.1007/s10817-018-9458-4. URL:

[10] Dixon, L. and Fleuriot, J. "IsaPlanner: A Prototype Proof Planner in Isabelle". In: *Automated Deduction – CADE-19*. Ed. by F. Baader. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 279–283. ISBN: 978-3-540-45085-6.

[11] First, E., Brun, Y., and Guha, A. "TacTok: Semantics-Aware Proof Synthesis". In: *Proceedings of the ACM on Programming Languages* 4.OOPSLA (2020), pp. 1–31. DOI: 10.1145/3428299.

[12] Gauthier, T. et al. "Learning to Prove with Tactics". In: *CoRR* abs/1804.00596 (2018). arXiv: 1804.00596. URL:

[13] Gonthier, B. *Formal proof of the Four Color Theorem*. Version 1.3.0. May 2023. URL:

[14] Gonthier, G. et al. "A Machine-Checked Proof of the Odd Order Theorem". In: *Interactive Theorem Proving*. Ed. by S. Blazy, C. Paulin-Mohring, and D. Pichardie. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 163–179. ISBN: 978-3-642-39634-2.

[15] Graves, A., Jaitly, N., and Mohamed, A.-r. "Hybrid speech recognition with Deep Bidirectional LSTM". In: *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*. 2013, pp. 273–278. DOI: 10.1109/ASRU.2013.6707742.

[16] Han, J. M. et al. "Proof Artifact Co-training for Theorem Proving with Language Models". In: *CoRR* abs/2102.06203 (2021). arXiv: 2102.06203. URL:

[17] Hindley, J. R. "Counting a Type's Inhabitants". In: *Basic Simple Type Theory*. Cambridge Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press, 1997. Chap. 9, pp. 108–139. DOI: 10.1017/CBO9780511608865.009.

[18] Huang, D. et al. "GamePad: A Learning Environment for Theorem Proving". In: *International Conference on Learning Representations*. 2019. URL:

[19] Joosten, S., Kaliszyk, C., and Urban, J. "Initial Experiments with TPTP-Style Automated Theorem Provers on ACL2 Problems". In: *ACL2 Theorem Prover and Its Applications (ACL2 2014)*. Ed. by F. Verbeek and J. Schmaltz. Vol. 152. Electronic Proceedings in Theoretical Computer Science (EPTCS). 2014, pp. 77–85. DOI: 10.4204/EPTCS.152.7.

[20] Kaliszyk, C. et al. "Reinforcement Learning of Theorem Proving". In: *CoRR* abs/1805.07563 (2018). arXiv: 1805.07563. URL:

[21] Kocsis, L. and Szepesvári, C. "Bandit Based Monte-Carlo Planning". In: *Machine Learning: ECML 2006*. Ed. by J. Fürnkranz, T. Scheffer, and M. Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293. ISBN: 978-3-540-46056-5.

[22] Leroy, X. *The Compcert verified compiler, software and commented proof*. 2009. URL:

[23] Li, L. et al. "Biomedical event extraction based on GRU integrating attention mechanism". In: *BMC Bioinformatics* 19 (Aug. 2018), pp. 177–184. DOI: 10.1186/s12859-018-2275-2.

[24] Lin, L.-J. *Reinforcement learning for robots using neural networks*. Tech. rep. DTIC Document, 1993.

[25] Lizotte, D. J. and Laber, E. B. "Multi-objective Markov decision processes for data-driven decision support". English. In: *J. Mach. Learn. Res.* 17 (2016). Id/No 211, p. 28. ISSN: 1532-4435. URL:

[26] McCarthy, J. "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I". In: *Communications of the ACM* 3.4 (1960), pp. 184–195. DOI: 10.1145/367177.367199.

[27] Mnih, V. et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL:

[28] Moura, L. M. de et al. "The Lean Theorem Prover (System Description)". In: *CADE*. 2015. URL:

[29] Nipkow, T., Paulson, L. C., and Wenzel, M. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Vol. 2283. LNCS. Springer, 2002.

[30] Olah, C. *Understanding LSTM Networks*. Blog post. Aug. 2015. URL:

[31] Paliwal, A. et al. "Graph Representations for Higher-Order Logic and Theorem Proving". In: *CoRR* abs/1905.10006 (2019). arXiv: 1905.10006. URL:

[32] Ross, S. M. *Stochastic Processes*. 2nd. John Wiley and Sons, 1995.

[33] Sanchez-Stern, A. et al. "Passport: Improving Automated Formal Verification Using Identifiers". In: *ACM Transactions on Programming Languages and Systems* 45.2 (June 2023), pp. 1–30. DOI: 10.1145/3593374. URL:

[34] Schaeffer, J. et al. "Checkers Is Solved". In: *Science* 317.5844 (2007), pp. 1518–1522. DOI: 10.1126/science.1144079. eprint: URL:

[35] Segler, M. H. S., Preuss, M., and Waller, M. P. "Planning chemical syntheses with deep neural networks and symbolic AI". In: *Nature* 555.7698 (Mar. 2018), pp. 604–610. ISSN: 1476-4687. DOI: 10.1038/nature25978. URL:

[36] Sekiyama, T., Imanishi, A., and Suenaga, K. "Towards Proof Synthesis Guided by Neural Machine Translation for Intuitionistic Propositional Logic". In: *CoRR* abs/1706.06462 (2017). arXiv: 1706.06462. URL:

[37] Sekiyama, T. and Suenaga, K. *Automated proof synthesis for propositional logic with deep neural networks*. 2018. arXiv: 1805.11799 [cs.AI].

[38] Sennrich, R., Haddow, B., and Birch, A. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. URL:

[39] Sharma, A. et al. *Robust and Adaptive Planning under Model Uncertainty*. 2019. arXiv: 1901.02577 [cs.AI].

[40] Silver, D. et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[41] Sironi, C. F. et al. "Self-adaptive MCTS for General Video Game Playing". In: *Applications of Evolutionary Computation*. Ed. by K. Sim and P. Kaufmann. Cham: Springer International Publishing, 2018, pp. 358–375. ISBN: 978-3-319-77538-8.

[42] Świechowski, M., Godlewski, K., Sawicki, B., et al. "Monte Carlo Tree Search: a review of recent modifications and applications". In: *Artificial Intelligence Review* 56 (2023), pp. 2497–2562. DOI: 10.1007/s10462-022-10228-y.

[43] Tai, K. S., Socher, R., and Manning, C. D. "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks". In: *CoRR* abs/1503.00075 (2015). arXiv: 1503.00075. URL:

[44] Uther, W. "Markov Decision Processes". In: *Encyclopedia of Machine Learning*. Ed. by C. Sammut and G. I. Webb. Boston, MA: Springer US, 2010, pp. 642–646. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_512. URL:

[45] Wu, Z. et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[46] Yang, K. and Deng, J. *Learning to Prove Theorems via Interacting with Proof Assistants*. 2019. arXiv: 1905.09381 [cs.LO].

[47] Yang, K. et al. "LeanDojo: Theorem Proving with Retrieval-Augmented Language Models". In: *Neural Information Processing Systems (NeurIPS)*. 2023.

[48] Yin, P. and Neubig, G. *A Syntactic Neural Model for General-Purpose Code Generation*. 2017. arXiv: 1704.01696 [cs.CL].

[49] Zaremba, W. and Sutskever, I. "Learning to Execute". In: *CoRR* abs/1410.4615 (2014). arXiv: 1410.4615. URL: