TacticZero: Learning to Prove Theorems from Scratch with Deep Reinforcement Learning Supplementary Materials

Anonymous Author(s)
Affiliation
Address
`email`

1 Embedding HOL4 in an MDP — A Sketch of the Software Environment

Rather than interacting directly with the HOL4 ITP engine, our agent acts on the more abstract MDP formulation introduced in main submission. In terms of software, we achieve this by defining a reinforcement learning environment which appropriately wraps HOL4. The API of our environment is inspired by that of Gym Brockman et al. [2016]. An instance e of the environment can be created by calling `HolEnv(g)` with an initial goal g (which represents the theorem to be proved). An action a can be taken by executing `e.step(a)` and the return value of this function is a pair consisting of the immediate reward and a boolean indicating whether the proof attempt has finished. Internally, the environment keeps track of the states encountered during the proof search.

Given a goal `g`, a tactic `t` and an argument list `c`, one may call `e.query(g,t,c)` independently of the `e.step(a)` function, to inspect the result of applying a tactic. The execution of a single query takes around 15 milliseconds on a regular laptop on average.

Once the environment detects a successful proof attempt (*i.e.*, an empty fringe), it re-constructs a HOL4 proof script from the MDP state sequence (along with some book-keeping information) and sends it to HOL4 for verification. The implied proof search can also be visualized by our software as an interactive tree which depicts all the information—see Figure 1 (which is the same search as Figure 6b in the main submission) and the interactive HTML version of that plot in the included file `proof_search_trees/Figure1.html`.

The proof search represented by Figure 1 is neither a breadth first search nor a depth first search, but a unique proof search managed by the agent itself, as indicated by the "Step" entry in the edge labels of the interactive version of the plot. We have also separated "assumptions" and "goal" in the fringes for better readability. From the agent's point of view, the "assumptions" and the "goal" are merged into one formula by chained implications, as mentioned in footnote 1 of the paper.

2 Space of Arguments

For each episode, a set of candidate theorems that can be selected as arguments is computed at the beginning of the episode. Duplicates are allowed in the list of arguments. The set of candidate theorems consists of all the theorems coming from the theories mentioned in the theorem to be proved, except for those that come after the theorem that is being proved in the library.

For example, if the agent is trying to prove theorem $\forall x, \bigcup\{x\} = x$ which is a theorem from the `pred_set` theory, the candidate set will contain *a)* all the theorems from the `bool` theory (because the symbol $\forall$ comes from `bool` theory), and *b)* all the theorems whose proof precedes that of $\forall x, \bigcup\{x\} = x$ from within `pred_set` theory.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

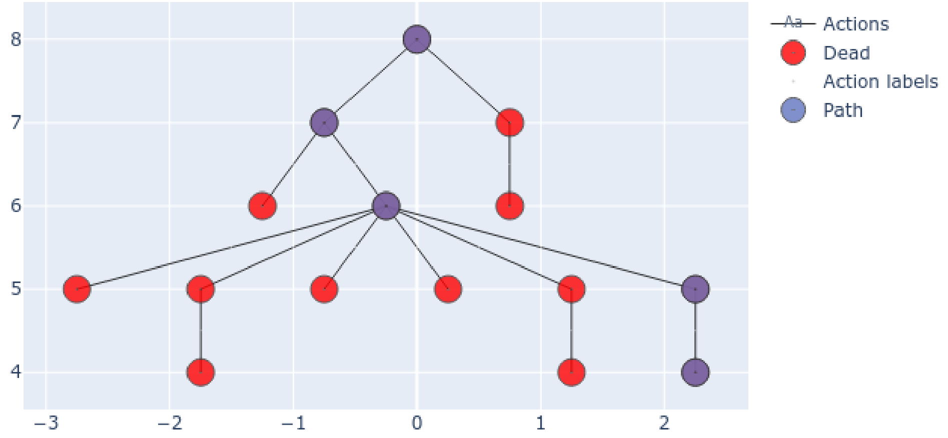

Figure 1: The visualization (in HTML) of a successful proof search of the theorem $\forall\ x\ s.\ x \in s \Rightarrow \forall f.\ f(x) \in \texttt{IMAGE}\ f\ s$. This particular proof was found in 13 steps. Red nodes represent the fringes that never lead to a successful proof, and blue nodes consist of a path from which a valid HOL4 proof can be re-constructed. Detailed information for both the edges and vertices can be revealed by hovering the mouse over the corresponding part of the HTML version of the plot, which is included in the supplementary materials as file `proof_search_trees/Figure1.html`.

The average size of such a set of candidate theorems for each theorem in our dataset (see `dataset.pdf` in the supplementary materials) is 468. In other words, the argument policy has to look at 468 theorems on average whenever the agent chooses an argument to a tactic. If the number of required arguments L is n, then the search space of the argument list is 468^n.

3 Use of HOL4

There are many extant interactive theorem-proving systems currently used, and being developed in the world today. Though some may share similar logical underpinnings (*e.g.,* HOL LIGHT and HOL4 are *incompatible* systems implementing the same logic), this is not necessarily the case (*e.g.,* COQ and HOL4 are fundamentally different). In addition, systems are implemented in different underlying programming languages; do not generally inter-operate at all; and prove different libraries of results. This fractured landscape makes comparing learning results across different systems impossible to do in any principled way. Moreover, as our bibliography and related work make clear, existing work in the application of machine learning to interactive theorem-proving has already "embraced" this chaos and chosen to work with a variety of different systems.

Our choice of HOL4 as a background system is grounded both in our own expertise with it, and some technical advantages crucial to reinforcement learning:

- HOL4's use of theory files stored on disk means it is quick to start in any chosen logical context;
- HOL4's implementation language, Poly/ML has good support for multi-threading and concurrent execution

4 Example Proofs

The subsequent pages of this document include several example proofs presented in the same format as Figure 6a of the main submission.

```
(* TacticZero proof *)

Theorem EVERY_CONJ:
  ∀P Q l. EVERY (λ(x:'a). (P x) ∧ (Q x)) l = (EVERY P l ∧ EVERY Q l)
Proof
  rw[] » Induct_on ‘l’
  >- (rw[listTheory.EVERY_DEF])
  >- (rw[listTheory.EVERY_DEF] » metis_tac[])
QED
(*
  (* Original human proof *)
   NTAC 2 GEN_TAC THEN LIST_INDUCT_TAC THEN
   ASM_REWRITE_TAC [EVERY_DEF] THEN
   CONV_TAC (DEPTH_CONV BETA_CONV) THEN
   REPEAT (STRIP_TAC ORELSE EQ_TAC) THEN
   FIRST_ASSUM ACCEPT_TAC);
*)
```

```
(* TacticZero proof *)

Theorem MONO_EXISTS:
  (∀x. P x ⇒ Q x) ⇒ (EXISTS P l ⇒ EXISTS Q l)
Proof
  rw[listTheory.EXISTS_MEM] » metis_tac[]
QED
(*
  (* Original human proof *)
  Q.ID_SPEC_TAC ‘l’ THEN LIST_INDUCT_TAC THEN
  ASM_SIMP_TAC (srw_ss()) [DISJ_IMP_THM]);
*)
```

```
(* TacticZero proof *)

Theorem FLAT_APPEND:
  ∀l1 l2. FLAT (APPEND l1 l2) = APPEND (FLAT l1) (FLAT l2)
Proof
  strip_tac » strip_tac »
  Induct_on ‘l1’
  >- (rw[listTheory.APPEND, listTheory.FLAT])
  >- (fs[listTheory.APPEND]
      » fs[listTheory.APPEND_ASSOC, listTheory.FLAT])
QED
(*
  (* Original human proof *)
  LIST_INDUCT_TAC
  THEN REWRITE_TAC [APPEND, FLAT]
  THEN ASM_REWRITE_TAC [APPEND_ASSOC]);
*)
```

```
(* TacticZero proof *)

Theorem MEM_MAP_f:
  ∀f l a. MEM a l ⇒ MEM (f a) (MAP f l)
Proof
  strip_tac » strip_tac » strip_tac
  » rw[listTheory.MEM_MAP] » (metis_tac[listTheory.MEM_MAP])
QED
(*
  (* Original human proof *)
  PROVE_TAC[MEM_MAP]
*)
```

```
(* TacticZero proof *)

Theorem REVERSE_11:
  (∀l1 l2:'a list. (REVERSE l1 = REVERSE l2) ⇔ (l1 = l2)
Proof
  strip_tac » strip_tac
  » metis_tac[listTheory.REVERSE_REVERSE]
QED
(*
  (* Original human proof *)
   REPEAT GEN_TAC THEN EQ_TAC THEN1
     (DISCH_THEN (MP_TAC o AP_TERM “REVERSE : 'a list → 'a list”) THEN
      REWRITE_TAC [REVERSE_REVERSE]) THEN
   STRIP_TAC THEN ASM_REWRITE_TAC []);
*)
```

```
(* TacticZero proof *)

Theorem FILTER_COMM:
  ∀f1 f2 l. FILTER f1 (FILTER f2 l) = FILTER f2 (FILTER f1 l)
Proof
Induct_on ‘l’
>- rw[]
>- rw[]
QED
(*
  (* Original human proof *)
  NTAC 2 GEN_TAC
  THEN BasicProvers.Induct
  THEN REWRITE_TAC [FILTER]
  THEN GEN_TAC
  THEN REPEAT COND_CASES_TAC
  THEN ASM_REWRITE_TAC [FILTER]);
*)
```

References

G. Brockman, Vicki Cheung, Ludwig Pettersson, J. Schneider, John Schulman, Jie Tang, and W. Zaremba. OpenAI Gym. *ArXiv*, abs/1606.01540, 2016.

```
(* TacticZero proof *)

Theorem ABSORPTION:
  ∀x:'a. ∀s. (x IN s) ⇔ (x INSERT s = s)
Proof
  strip_tac
  » rw[pred_setTheory.INSERT_DEF]
  » fs[pred_setTheory.GSPEC_ETA, pred_setTheory.INSERT_DEF]
  » metis_tac[pred_setTheory.SPECIFICATION]
QED
(*
  (* Original human proof *)
  REWRITE_TAC [EXTENSION,IN_INSERT] THEN
  REPEAT (STRIP_TAC ORELSE EQ_TAC) THEN
  ASM_REWRITE_TAC [] THEN
  FIRST_ASSUM (fn th => fn g => PURE_ONCE_REWRITE_TAC [SYM(SPEC_ALL th)] g)
  THEN DISJ1_TAC THEN REFL_TAC
*)
```

```
(* TacticZero proof *)

Theorem DISJOINT_INSERT:
  (∀(x:'a) s t. DISJOINT (x INSERT s) t ⇔ DISJOINT s t ∧ x NOTIN t
Proof
  strip_tac » strip_tac » strip_tac
  » fs[pred_setTheory.IN_INSERT, pred_setTheory.INSERT_DEF, pred_setTheory.IN_DISJOINT]
  » metis_tac[]
QED
(*
  (* Original human proof *)
  REWRITE_TAC [IN_DISJOINT,IN_INSERT] THEN
  CONV_TAC (ONCE_DEPTH_CONV NOT_EXISTS_CONV) THEN
  REWRITE_TAC [DE_MORGAN_THM] THEN
  REPEAT GEN_TAC THEN EQ_TAC THENL
  [let val v = genvar (==‘:’a‘==)
       val GTAC = X_GEN_TAC v
   in DISCH_THEN (fn th => CONJ_TAC THENL [GTAC,ALL_TAC] THEN MP_TAC th)
      THENL [DISCH_THEN (STRIP_ASSUME_TAC o SPEC v) THEN ASM_REWRITE_TAC [],
             DISCH_THEN (MP_TAC o SPEC (“x:’a”)) THEN REWRITE_TAC[]]
   end,
   REPEAT STRIP_TAC THEN ASM_CASES_TAC (“x’:’a = x”) THENL
   [ASM_REWRITE_TAC[], ASM_REWRITE_TAC[]]]
*)
```

```
(* TacticZero proof *)

Theorem INSERT_INTER:
  (∀x:’a. ∀s t. (x INSERT s) INTER t = (if x IN t then x INSERT (s INTER t) else s INTER t)
Proof
strip_tac » strip_tac »
rw[pred_setTheory.INSERT_DEF, pred_setTheory.SPECIFICATION, pred_setTheory.INTER_DEF]
>- (rw[pred_setTheory.GSPEC_ETA] » metis_tac[])
>- (rw[] » (rw[pred_setTheory.GSPEC_ETA] >> metis_tac[]))
QED
(*
  (* Original human proof *)
  REPEAT GEN_TAC THEN COND_CASES_TAC THEN
  ASM_REWRITE_TAC [EXTENSION,IN_INTER,IN_INSERT] THEN
  GEN_TAC THEN EQ_TAC THENL
  [STRIP_TAC THEN ASM_REWRITE_TAC [],
   STRIP_TAC THEN ASM_REWRITE_TAC [],
   PURE_ONCE_REWRITE_TAC [CONJ_SYM] THEN
   DISCH_THEN (CONJUNCTS_THEN MP_TAC) THEN
   STRIP_TAC THEN ASM_REWRITE_TAC [],
   STRIP_TAC THEN ASM_REWRITE_TAC []]);
*)


(* TacticZero proof *)

Theorem SET_MINIMUM:
  (∀s:’a → bool. ∀M. (∃x. x IN s) ⇔ ∃x. x IN s ∧ ∀y. y IN s ⇔ M x <= M y
Proof
rw[]
» fs[boolTheory.IMP_CONG, boolTheory.EQ_TRANS, boolTheory.EQ_IMP_THM]
» rw[arithmeticTheory.WOP_measure, boolTheory.COND_ABS]
» metis_tac[boolTheory.ONE_ONE_THM]
QED
(*
  (* Original human proof *)
  REPEAT (STRIP_TAC ORELSE EQ_TAC) THENL
  [IMP_RES_THEN (ASSUME_TAC o ISPEC (“M:’a→num”)) lemma THEN
   let val th = SET_SPEC_CONV (“(n:num) IN M x | (x:’a) IN s”)
   in IMP_RES_THEN (STRIP_ASSUME_TAC o REWRITE_RULE [th]) NUM_SET_WOP
   end THEN EXISTS_TAC (“x’:’a”) THEN CONJ_TAC THENL
   [FIRST_ASSUM ACCEPT_TAC,
    FIRST_ASSUM (SUBST_ALL_TAC o SYM) THEN
    REPEAT STRIP_TAC THEN FIRST_ASSUM MATCH_MP_TAC THEN
    EXISTS_TAC (“y:’a”) THEN CONJ_TAC THENL
    [REFL_TAC, FIRST_ASSUM ACCEPT_TAC]],
   EXISTS_TAC (“x:’a”) THEN FIRST_ASSUM ACCEPT_TAC]
*)
```

```
(* TacticZero proof *)

Theorem INJ_DELETE:
  (∀f s t. INJ f s t ==> ∀e. e IN s ==> INJ f (s DELETE e) (t DELETE (f e))
Proof
strip_tac » strip_tac » strip_tac
» fs[] » rw[]
» (fs[pred_setTheory.INJ_DEF] »
    (strip_tac » fs[pred_setTheory.IN_DELETE, boolTheory.IMP_DISJ_THM]
     >- (metis_tac[pred_setTheory.IN_APP])
     >- (fs[] » (fs[] » (fs[] » (metis_tac[]))))))
QED
(*
  (* Original human proof *)
  RW_TAC bool_ss [INJ_DEF, DELETE_DEF] THENL
  [‘~(e = x)’ by FULL_SIMP_TAC bool_ss
                 [DIFF_DEF,DIFF_INSERT, DIFF_EMPTY, IN_DELETE] THEN
  FULL_SIMP_TAC bool_ss [DIFF_DEF,DIFF_INSERT, DIFF_EMPTY, IN_DELETE] THEN
  METIS_TAC [],
  METIS_TAC [IN_DIFF]]);
*)
```

```
(* TacticZero proof *)

Theorem IMAGE_SURJ:
  (∀f:'a->'b. ∀s t. SURJ f s t = ((IMAGE f s) = t)
Proof
strip_tac » strip_tac » rw[pred_setTheory.SURJ_DEF]
» fs[] » fs[pred_setTheory.EXTENSION]
» fs[pred_setTheory.SPECIFICATION] » fs[]
» fs[pred_setTheory.IMAGE_applied]
» fs[pred_setTheory.IN_APP, boolTheory.RES_EXISTS_THM]
» metis_tac[]
QED
(*
  (* Original human proof *)
  PURE_REWRITE_TAC [SURJ_DEF,EXTENSION,IN_IMAGE] THEN
  REPEAT GEN_TAC THEN EQ_TAC THENL
  [REPEAT (STRIP_TAC ORELSE EQ_TAC) THENL
   [RES_TAC THEN ASM_REWRITE_TAC [],
    MAP_EVERY PURE_ONCE_REWRITE_TAC [[CONJ_SYM],[EQ_SYM_EQ]] THEN
    FIRST_ASSUM MATCH_MP_TAC THEN FIRST_ASSUM ACCEPT_TAC],
   DISCH_THEN (ASSUME_TAC o CONV_RULE (ONCE_DEPTH_CONV SYM_CONV)) THEN
   ASM_REWRITE_TAC [] THEN REPEAT STRIP_TAC THENL
   [EXISTS_TAC (“x:'a”) THEN ASM_REWRITE_TAC [],
    EXISTS_TAC (“x':'a”) THEN ASM_REWRITE_TAC []]])
*)
```