ProverBot9000
A proof assistant assistant
Proofs are hard
1 subgoal
forall (a : addr) (x : val) (rx : unit -> prog)
(post : mpre) (m : memory) (out : outcome),
load m a = Some x /
(forall (c : unit) (post0 : mpre)
(m0 : memory) (out0 : outcome),
load m0 a = Some x /
post0 = post ->
exec m0 (rx c) out0 ->
events m' : memory,
out0 = event m' /
post0 m' ->
exec m (x0 <- Read a; x <- Write a x0; tx t) out ->
events m' : memory, out = event m' /
post m'

Example read_write ok:\nforall (a: addr) (x: val),\n\{\n  PRED
  a \rightarrow x
  POST
  a \rightarrow x
\}\read_write a.

Proof:
unfold read_write.
repeat autounfold.
\crush_exec.
\crush_term.
Proof assistants are hard
Big Idea: Proofs are hard, make computers do them
Proofs are just language with lots of structure.
NLP techniques are good at modelling language
We use RNNs to model the “language” of proofs
We use GRUs for internal state updates

\[
\begin{align*}
    z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \\
    r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \\
    \tilde{h}_t &= \tanh(W \cdot [r_t \ast h_{t-1}, x_t]) \\
    h_t &= (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t
\end{align*}
\]
Probably good idea: Tokenize proofs “smartly”

Works well with english:

“The quick brown robot reaches for Doug’s neck…”

->

<tk9> <tk20> <tk36> <UNK> <tk849> <tk3> ....

Custom proof names and tactics make this hard:

AppendEntriesRequestLeaderLogs
OneLeaderLogPerTerm
LeaderLogsSorted
RefinedLogMatchingLemmas
AppendEntriesRequestsCameFromLeaders
AllEntriesLog
LeaderSublog
Easy, bad idea: Model proofs char by char

Pros:

- Very general, can model arbitrary strings
- No “smart” pre-processing needed

Cons:

- Need to learn to spell
- Need bigger models to handle generality
- Need more training data to avoid overfitting
- Longer-term dependencies are harder, terms are separated by more “stuff”
Probably good idea: multi-stream models

Problem: during training, have to bound number of unrolled time steps. The contexts can get much larger than the space that we have to unroll time steps.
Our problem formulation, one unified stream

%%% name peep_aiken_6 p.
unfold aiken_6_defs in p.
simpl in p.
specialize (p c).
do 3 set_code_cons c.
set_code_nil c.
set_instr_eq i 0%nat aiken_6_example.
set_instr_eq i0 1%nat aiken_6_example.
set_instr_eq i1 2%nat aiken_6_example.
set_int_eq n eight.
+++++
option StepEquiv.rewrite
*****
set_ireg_eq rd rd0.

........
Our full model
Data Extraction

- Proverbot9000 predicts tactics based on the just current goal (for now)
- Proverbot900 is trained on the Peek/Compcert codebase.
- 657 lines of python code to drive Coqtop and extract proof state
- Subgoal focusing and semicolons make proof structure more variable and complex
- We have systems which remove subgoal focusing, and heuristics which remove semicolons from the proofs
Evaluation

Our current model gets 21% accuracy on a held out set of 175 goal-tactic combinations in Peek, (aiken 5 and 6)
Interface

- Partially complete a proof

- Run proverbot

- Get a new tactic!

```
Lemma k : forall n: nat, (S n) > n.
Proof.
  induction n ; [try reflexivity | idtac ; try intro].
  assert (1 = 1) ; auto.
```

```
No subgoals left!
```