

ProverBot9000

A proof assistant assistant

Proofs are hard





scratch crash0.v

```

- context [ load (store _ ?A _) ?A ] => rewrite load store
- context [ store (store _ ?A _) ?A _ ] => rewrite store
- store ?M _ = store ?M _ => f_equal
- Some ?X = Some ?Y => f_equal
- context [ load (store _ ?A _) ?A ] | - _ => rewrite load
[ H: Some ?X = Some ?Y | - _ ] => assert (X = Y); try congr
[ H1: ?P = Some ?X, H2: ?P = Some ?Y | - _ ] => replace X w
=> omega
end.

Ltac crush_exec :=
repeat match goal with
| [ H: step _ _ _ | - _ ] => inv H
| [ H: exec _ (progseq _ _ ) | - _ ] => inv H
end.

Ltac crush_term :=
match goal with
| H: context [ _ = Terminated _ ], _ : exec ?M ( _ ?C) ?O | - c
end.

(** Examples *)

Definition read_write (a: addr) rx :=
x <- Read a ;
Write a x ;;
rx tt.

Example read_write_ok:
forall (a: addr) (x: val), {<
PRE
a |-> x
POST
a |-> x
>} read_write a.
Proof.
unfold read_write.
repeat autounfold.
crush_exec.
crush_term.

```

```

1 subgoal
(1/1)
forall (a : addr) (x : val) (rx : unit -> prog)
(post : mpred) (m : memory) (out : outcome),
load m a = Some x /\
(forall (c : unit) (post0 : mpred)
(m0 : memory) (out0 : outcome),
load m0 a = Some x /\ post0 = post ->
exec m0 (rx c) out0 ->
exists m' : memory,
out0 = Terminated m' /\ post0 m') ->
exec m (x0 <- Read a; _ <- Write a x0; rx tt) out ->
exists m' : memory, out = Terminated m' /\ post m'

```

Messages

Errors

Jobs

Error: No matching clauses for match.

Proof assistants are hard



Big Idea: Proofs are hard, make computers do them



Proofs are just language with lots of structure

The screenshot shows the Coq IDE interface. The left pane contains a proof script with several sections: a context definition, a tactic definition for `crush exec`, a tactic definition for `crush term`, an example definition for `read write`, and a proof attempt for `read write ok`. The `crush term` tactic is circled in red, with the text "Want to generate this!" next to it. The right pane shows the current goal, which is a complex logical statement involving memory, addresses, and program execution. The goal is annotated with "Local Context" and "Goal" in red text. The bottom status bar indicates "Ready, proving read_write_ok" and "Line: 168 Char: 14".

```
- context [ load (store _ ?A _) ?A ] => rewrite load store
- context [ store (store _ ?A _) ?A _ ] => rewrite store
- store ?M _ = store ?M _ => f_equal
- Some ?X = Some ?Y => f_equal
: context [ load (store _ ?A _) ?A ] |- _ => rewrite load
[ H: Some ?X = Some ?Y |- _ ] => assert (X = Y); try congruence
[ H1: ?P = Some ?X, H2: ?P = Some ?Y |- _ ] => replace X with Y
=> omega
end.

Ltac crush exec :=
repeat match goal with
| [ H: step _ _ _ _ ] => inv H
| [ H: exec _ (progseq _ _ ) _ ] => inv H
end.

Ltac crush term :=
match goal with
| H: context [ _ = Terminated _ ], _ : exec ?M ( _ ?C) ?O |- _
end.

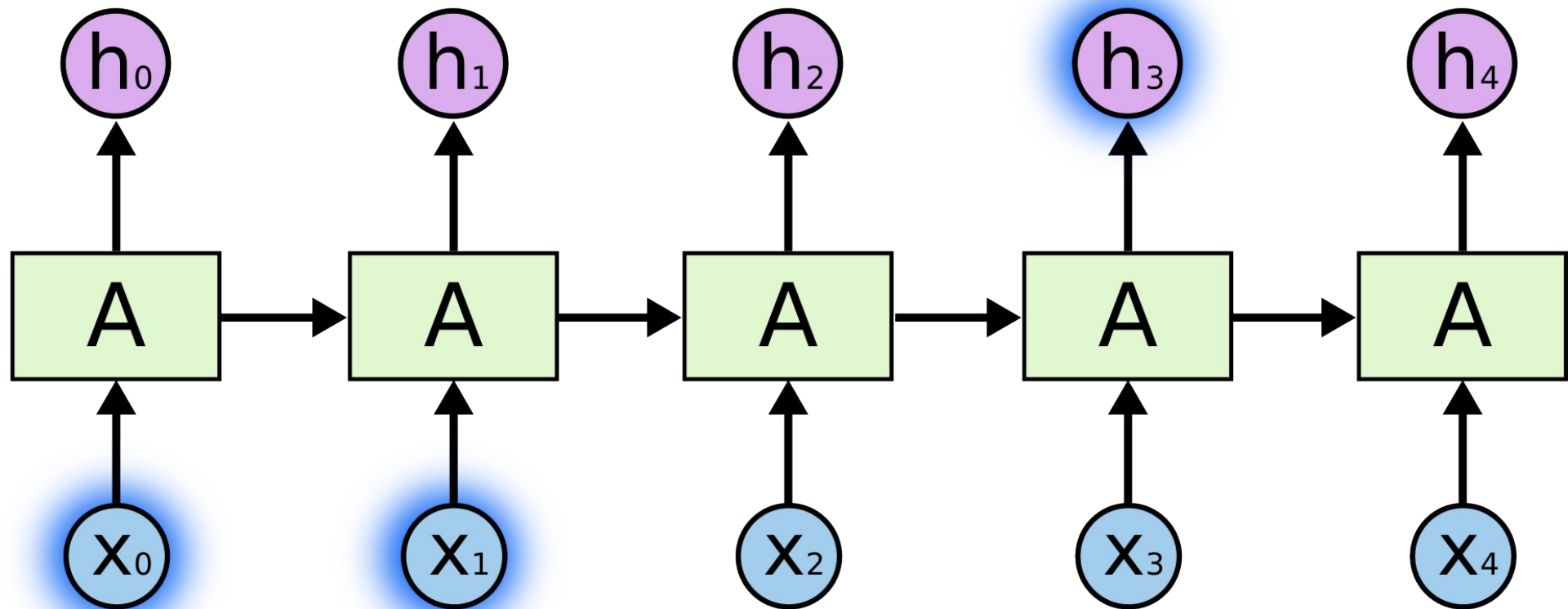
(** Examples *)

Definition read write (a: addr) rx :=
x <- Read a ;
Write a x ;;
rx tt.

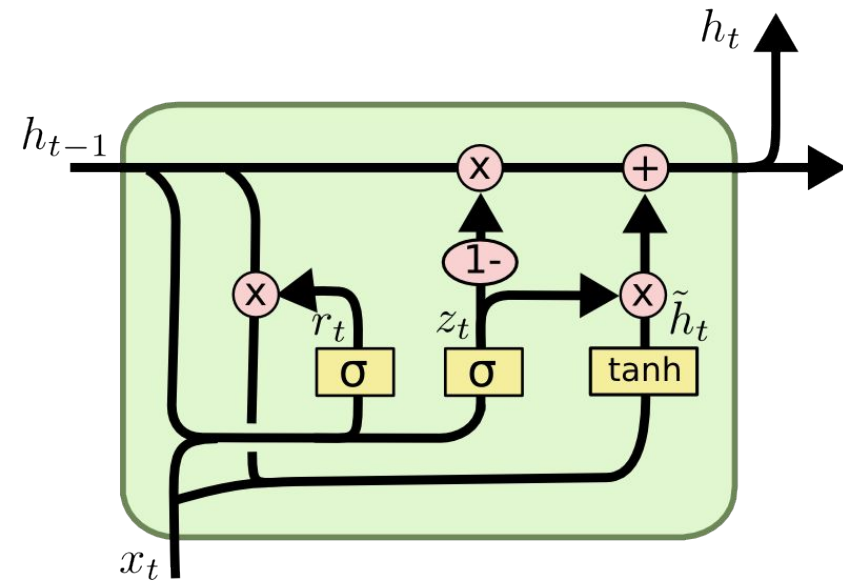
Example read write ok:
forall (a: addr) (x: val), {<
PRE
a |-> x
POST
a |-> x
>} read_write a.
Proof.
unfold read write.
repeat autounfold. intros.
crush exec.
crush term.

1 subgoal
a : addr
x : val
rx : unit -> prog
post : mpred
out : outcome
m' : memory
v : val
H7 : load m' a = Some v
H : load m' a = Some x /\
(forall (c : unit) (post0 : mpred)
(m : memory) (out : outcome),
load m a = Some x /\ post0 = post ->
exec m (rx c) out ->
exists m' : memory,
out = Terminated m' /\ post0 m')
H1 : exec (store m' a v) (rx tt) out
(1/1)
exists m'0 : memory,
out = Terminated m'0 /\ post m'0
```


We use RNNs to model the “language” of proofs



We use GRUs for internal state updates



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

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:

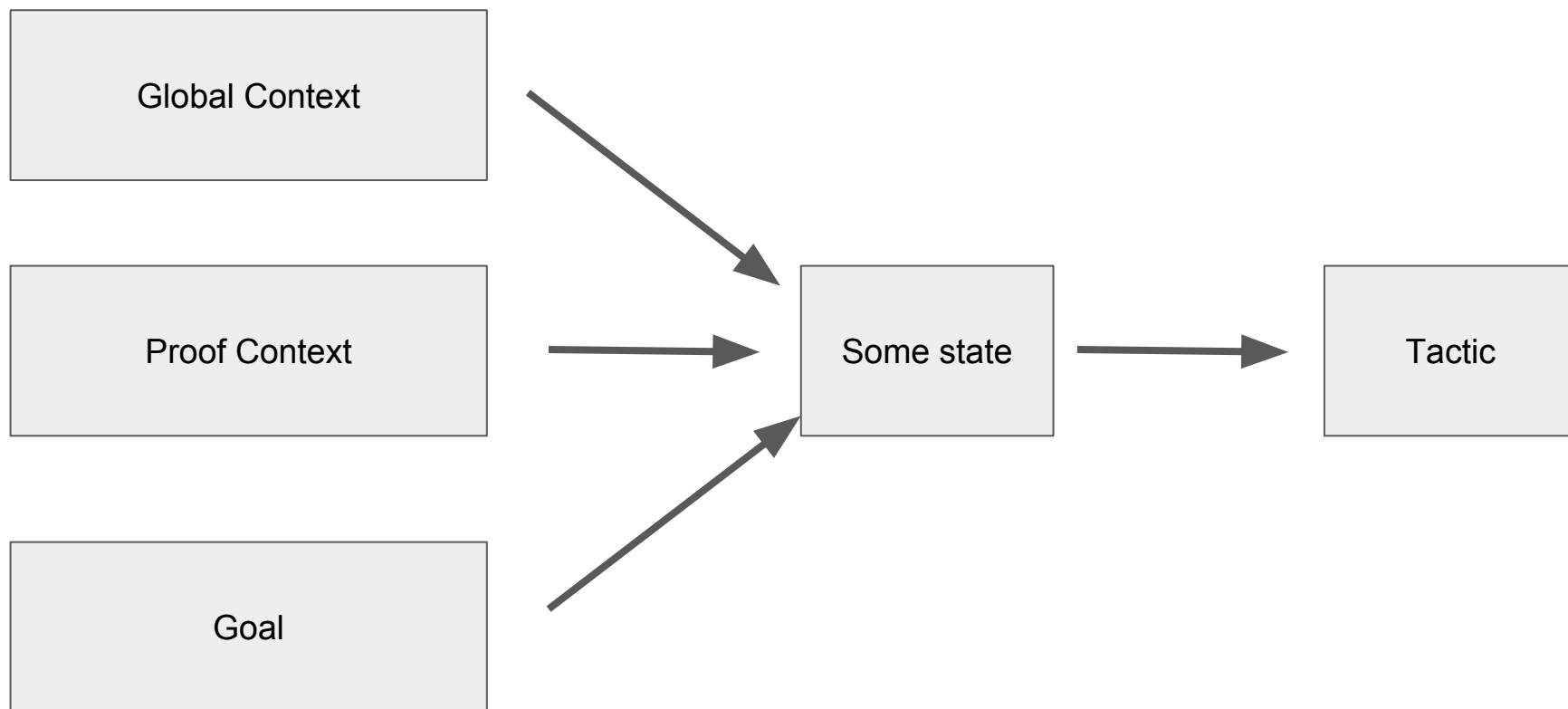
- Need to learn to spell

- 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.
```

Start tokens

Previous tactics

Dividing tokens

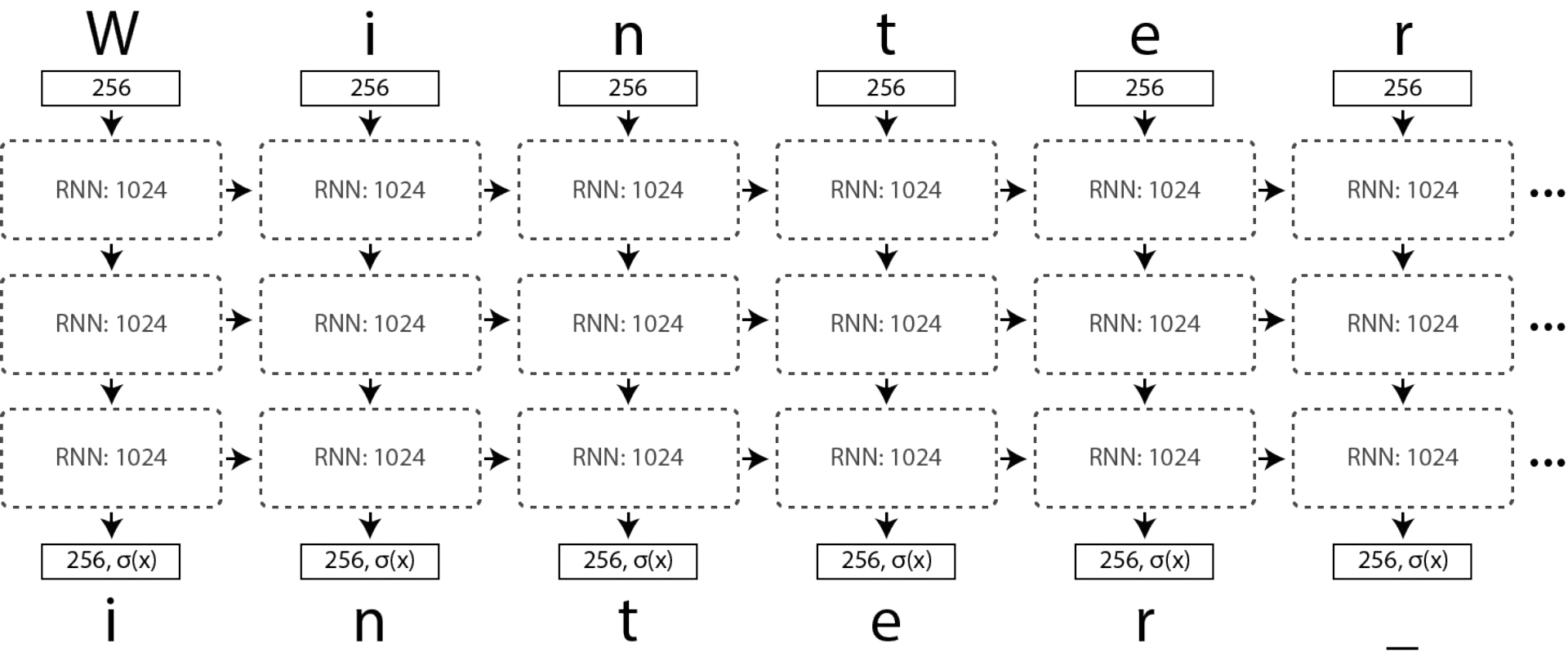
Current goal

Dividing tokens

Next tactic

.....

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

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

from the proofs

```
Lemma k : forall n : nat, (S n) > n.  
Proof.  
  induction n.  
  try reflexivity.  
  assert (1 = 1).  
  auto.  
  auto.  
  idtac.  
  try intro.  
  omega.  
Qed.
```

```
forall n : nat, S n > n  
*****  
induction n.  
%%%%%%%%%%  
1 > 0  
*****  
try reflexivity.  
%%%%%%%%%%  
1 > 0  
*****  
assert (1 = 1).  
%%%%%%%%%%  
1 = 1  
*****  
auto.  
%%%%%%%%%%  
1 > 0  
*****  
auto.  
%%%%%%%%%%  
S (S n) > S n  
*****  
idtac.  
%%%%%%%%%%  
S (S n) > S n  
*****  
try intro.  
%%%%%%%%%%  
S (S n) > S n  
*****  
omega.  
%%%%%%%%%%
```

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.  
  █
```

```
ns/proverbot9000 $ ./predict.py simple-proof.v █
```

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

No subgoals left!

DEMO

