universität
innsbruck

Learning theorem proving through self-play

Stanisław Purgał

The goal

Learn to prove theorems without:

- any proofs
- any theorems

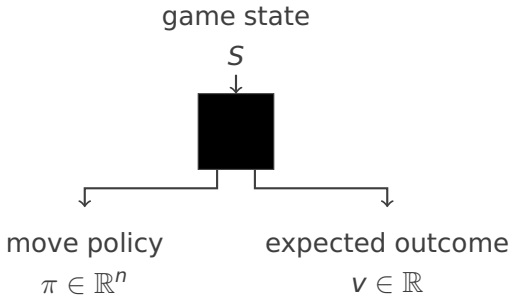
What we get:

- a list of axioms defining the logic

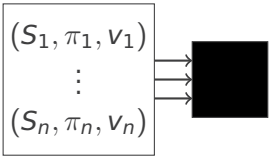
Overview

- AlphaZero (briefly)
- Proving game
- adjusting MCTS for proving game
- some results

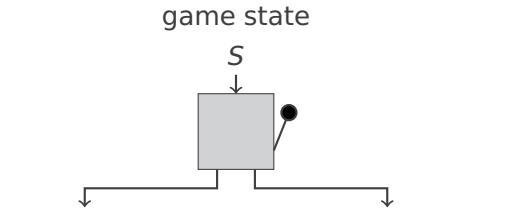
Neural black box



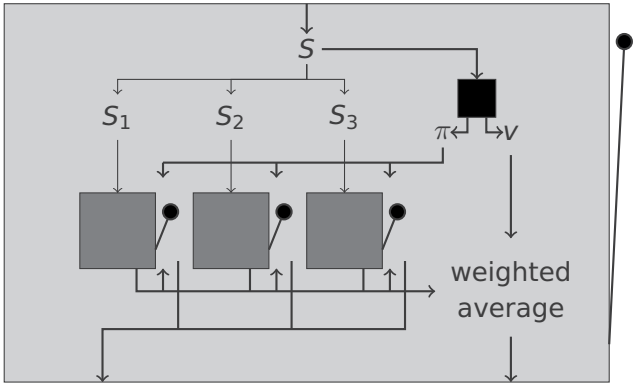
Neural black box



Monte-Carlo Tree Search



Monte-Carlo Tree Search



choose a child according to the formula:

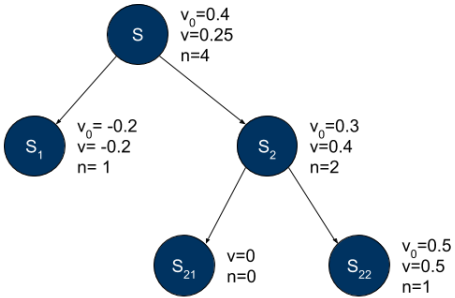
$$c \cdot \frac{\sqrt{n}}{n_i}\pi_i + v_i$$

$$c = \left(\log \frac{n + c_{base} + 1}{c_{base}} + c_{init}\right)$$

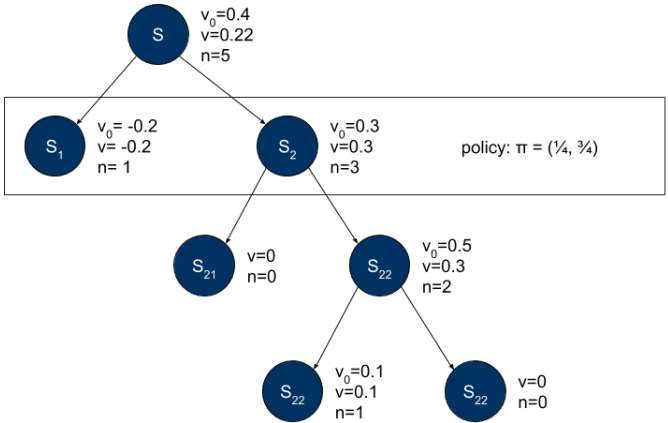
$$c_{base} = 19652$$

$$c_{init} = 1.25$$

Monte-Carlo Tree Search



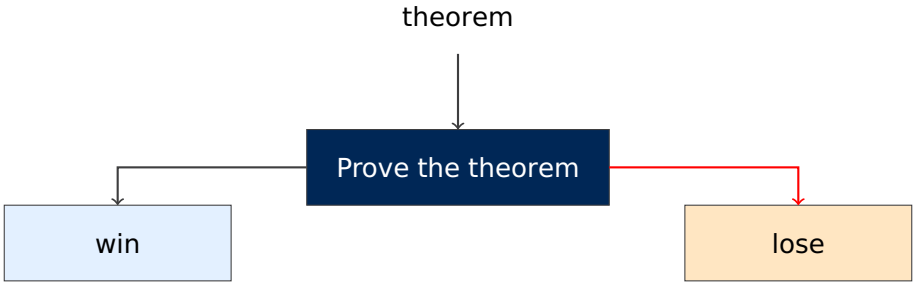
Monte-Carlo Tree Search



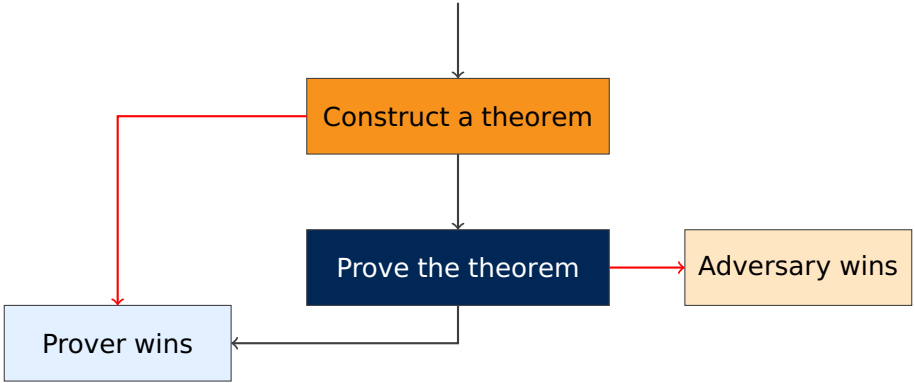
Closing the loop

- play lots of games
- choose moves randomly, according to MCTS policy
- use finished games for training:
  - target value in the result of the game
  - target policy is the MCTS policy
- also add noise to neural network output to increase exploration

Proving game



Proving game

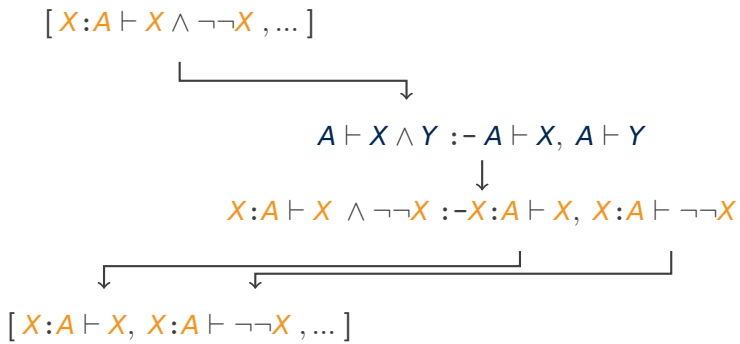


Prolog-like proving

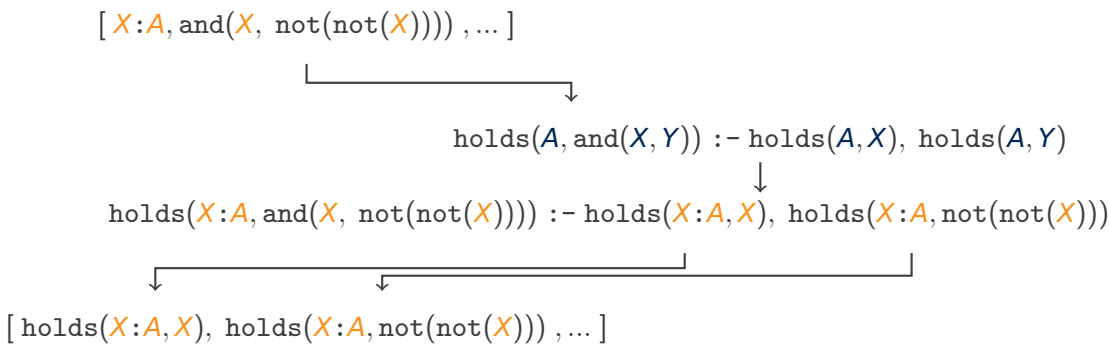
$$\frac{A \vdash X \qquad A \vdash Y}{A \vdash X \wedge Y} \tag{1}$$

$$\texttt{holds}(A, \texttt{and}(X, Y)) \texttt{ :- } \texttt{holds}(A, X),\ \texttt{holds}(A, Y) \tag{2}$$

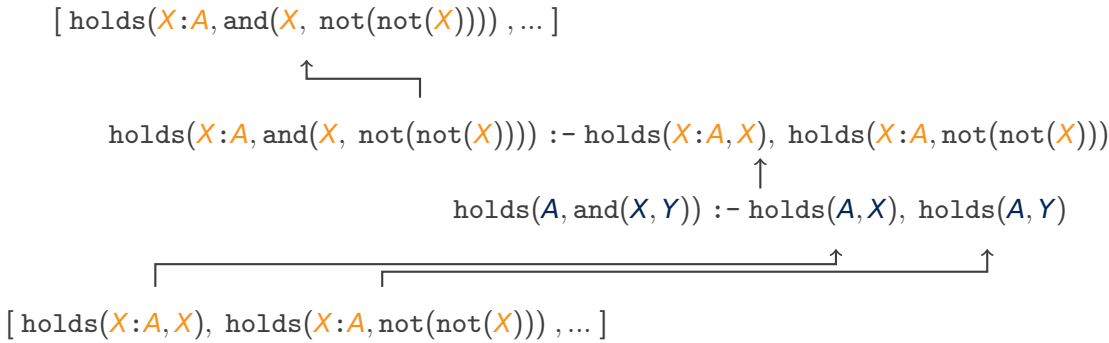
Prolog-like proving



Prolog-like proving

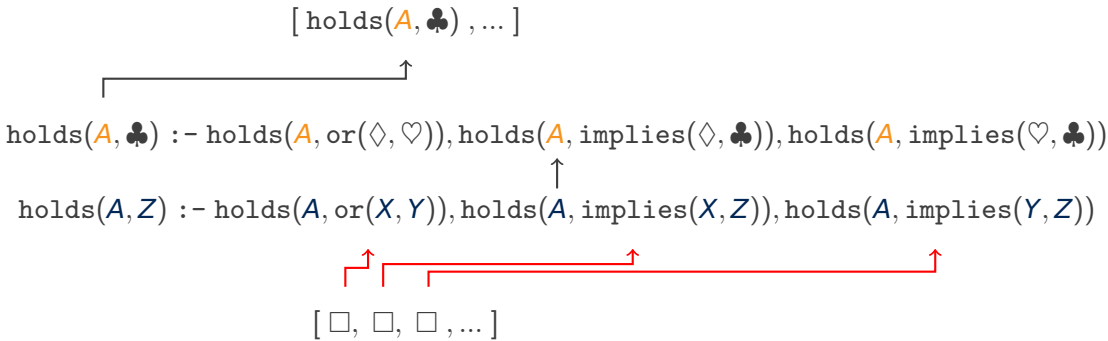


Prolog-like theorem constructing

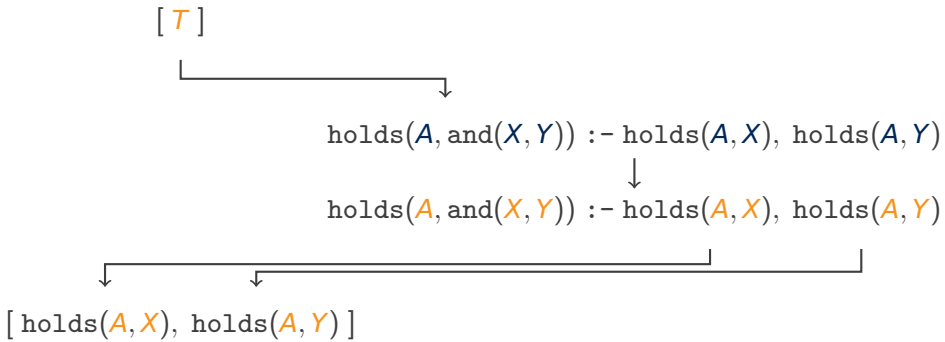


bad idea

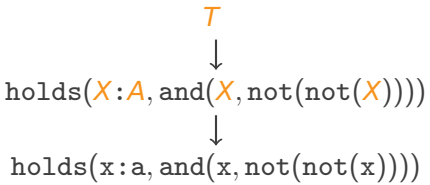
Prolog-like theorem constructing



Prolog-like theorem constructing



Prolog-like theorem constructing



Forcing termination of the game

Step limit:

- ugly extension of game state
- strategy may depend on number of steps left
- even if we hide it, there is a correlation:
  large term constructed $\sim$ few steps left $\sim$ will likely lose

Forcing termination of the game

Sudden death chance:

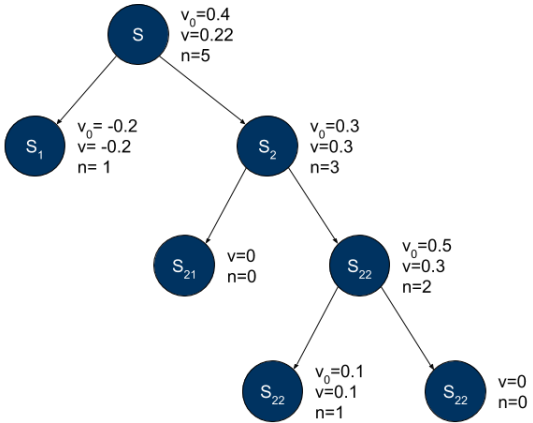
- game states nicely equal
- no hard limit for length of a theorem

During training playout, randomly terminate game with chance p_d.
In MCTS, adjust value $v' = (-1) \cdot p_d + v \cdot (1 - p_d)$.

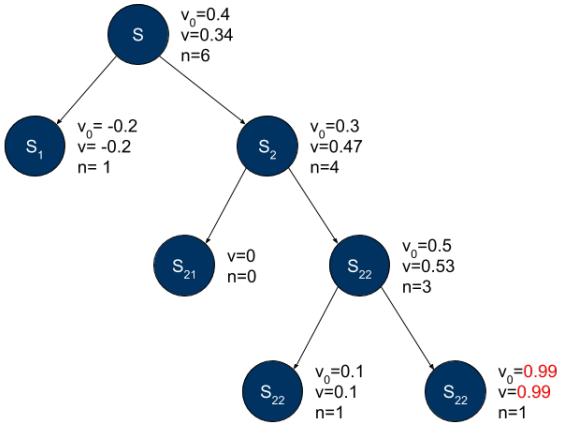
Disadvantages of this game

- two different players - if one player starts winning every game, we can’t learn much
- proof use single inference steps - inefficient
- players don’t take turns - MCTS not designed for that situation

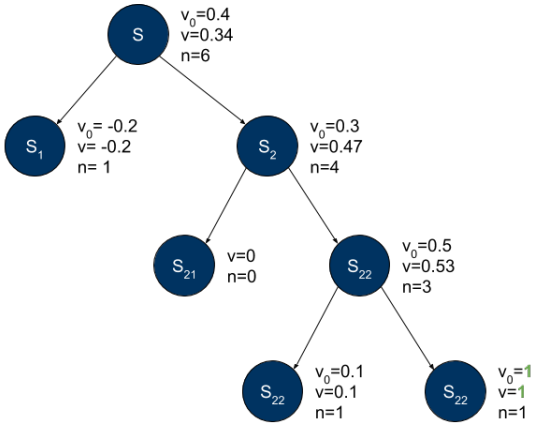
Not using maximum



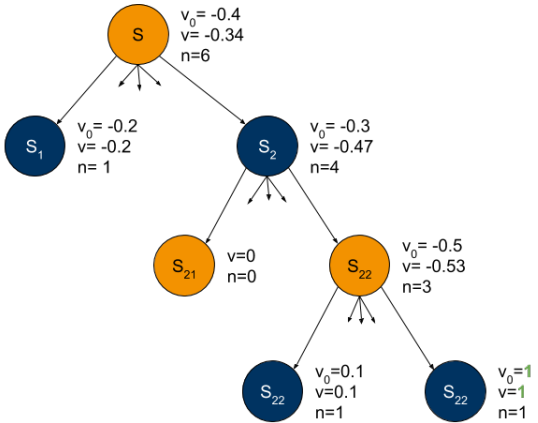
Not using maximum



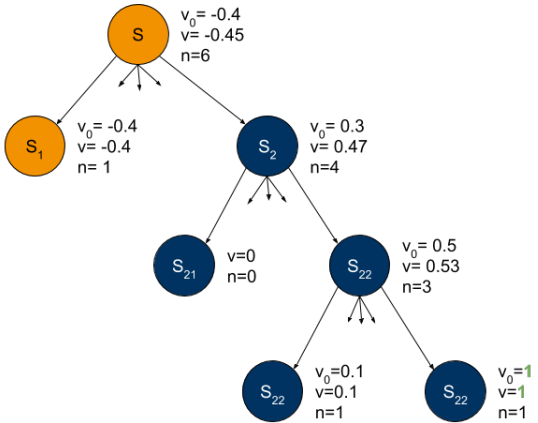
Not using maximum



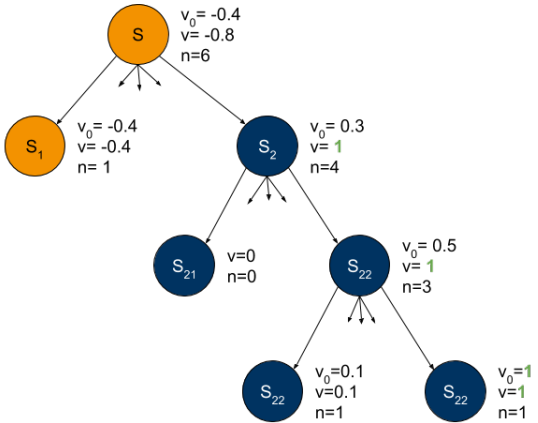
Not using maximum



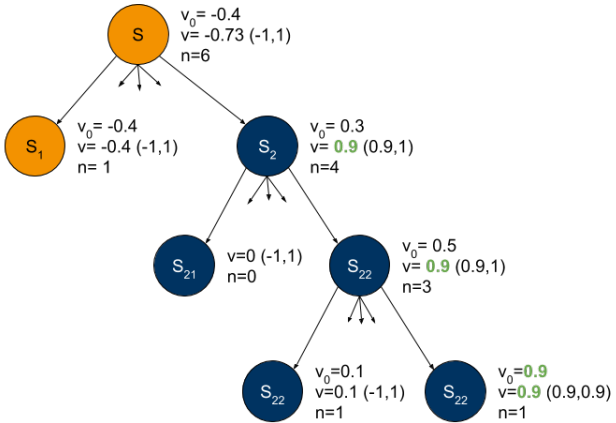
Certainty propagation



Certainty propagation



Certainty propagation



Certainty propagation

for uncertain leafs:

$v = \blacksquare$

$a = \blacksquare$

$l = -1$

$u = 1$

for certain leafs:

$v = \text{result}$

$a = \text{result}$

$l = \text{result}$

$u = \text{result}$

recursively:

$v = \min(u, \max(l, a))$

$a = \frac{\blacksquare + \Sigma v_i \cdot n_i}{n+1}$

$l = \max_i l_i$

$u = \max_i u_i$

when player changes:

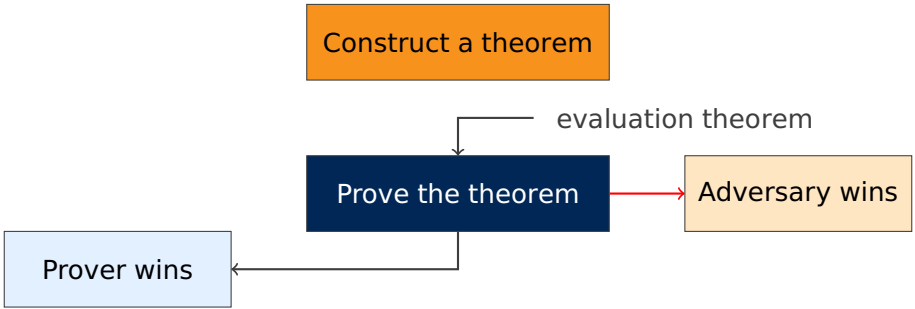
- values and bounds flip
- lower and upper bound switch places

Learning the proving game

Like AlphaZero, with few differences:

- using Transformer (encoder) for ■
- for theorems that prover failed to prove, show proper path with additional training samples
- during evaluation, greedy policy and step limit instead of sudden death
- balance training batches to have even split of won and lost games

Proving game evaluation



Potential problems

Players are non symmetrical:

- Prover could be winning everything
- Adversary could be winning everything
  to some extent this is handled by additional training samples

can be solved by more exploration

Uninteresting space of hard theorems

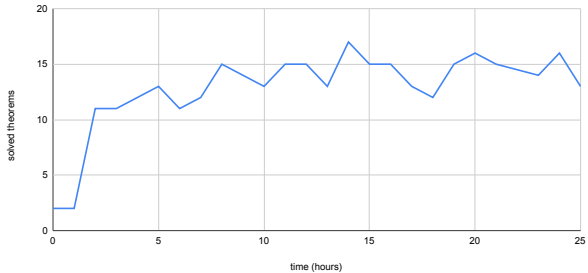
$\exists_x f(x) = y$ (where f is a one-way function)

- easy to prove if you can choose what y is
- hard to prove if y is fixed
  so hard that we can't expect the prover to learn it

this is stable - more learning and/or exploration won't help

Results

(intuitionstic first-order - sequential calculus)



Results

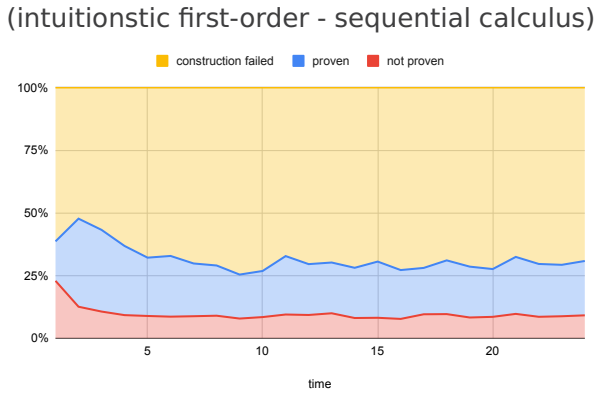
Solved:

$$\vdash (\forall_a \forall_b p_c(f_c(a,b)) \to \exists_d \exists_e p_c(f_c(d,e)))$$
$$\vdash (\neg(p_a(\emptyset) \to p_b(\emptyset)) \to (p_b(\emptyset) \to p_a(\emptyset)))$$

Unsolved:

$$\vdash (\exists_a p_b(a) \to \exists_c p_b(c)) \tag{3}$$

Results


(intuitionstic first-order - sequential calculus)
construction failed
proven
not proven
100%
75%
50%
25%
0%
5
10
15
20
time

Results

(intuitionistic first-order - sequential calculus)
unproven theorems - first hour:

$$A, \bot \vdash C$$
$$\vdash (\bot \to B)$$
$$(A \to B), A \vdash B$$
$$A, B, C, D, E, F, G, H \vdash H$$
$$A, B, C, D, E, F, G, H, I, J, K, L, M \vdash M$$
$$A, B, C, D, E, F, G, H, I \vdash I$$

Results

(intuitionistic first-order - sequential calculus)
unproven theorems - second hour:

$$\forall_a \Omega_a C \vdash \Omega_a C$$
$$\vdash (B \vee (\neg\bot \vee C))$$
$$(A \wedge \Omega_c \Omega_e F) \vdash \exists_e \Omega_c \Omega_e F$$
$$(A \wedge B) \vdash (D \to B)$$
$$(A \wedge B) \vdash (D \vee A)$$
$$\vdash ((B \wedge (C \wedge D)) \to C)$$

Results

(intuitionistic first-order - sequential calculus)
unproven theorems - third hour:

$$\forall_a(\Omega_c\Omega_a E \wedge \Omega_g(\Omega_a J \star \Omega_a L)) \vdash \Omega_g(\Omega_a J \star \Omega_a L)$$
$$A, B, C, D, E, F, G, ((H \wedge \bot) \wedge I) \vdash \neg K$$
$$A, B, C, D, E, F, G, H, \bot \vdash (J \vee K)$$
$$A, \neg B, C, (D \wedge B) \vdash (F \vee G)$$
$$\forall_a(p_b(f_c(f_d(a, \emptyset), \emptyset)) \wedge \bot), \neg\neg E \vdash \exists_g \Omega_g I$$
$$A, B, \neg C, D, E, (C \wedge F) \vdash (H \leftrightarrow \neg\bot)$$

Results

(intuitionistic first-order - sequential calculus)
unproven theorems - twelth hour:

$$A, B, (\forall_c \Omega_e(\Omega_c H \star \neg\neg\neg\neg\Omega_j\Omega_l\neg(\neg\bot \star (\neg\neg(\bot \star \Omega_c Q) \star \neg\neg\Omega_c S))) \leftrightarrow A)$$
$$\vdash \Omega_e(\Omega_c H \star \neg\neg\neg\neg\Omega_j\Omega_l\neg(\neg\bot \star (\neg\neg(\bot \star \Omega_c Q) \star \neg\neg\Omega_c S)))$$

$$A, B, (\forall_c X \leftrightarrow A) \vdash X$$

How to do better

- train longer and/or harder
  costly
- relegate low-level reasoning to some more efficient solver
  need to invent some other mechanism for generating theorems
- allow use of theorems, not only axioms
  action space becomes large and changing over time

  all above still face uninteresting theorem space
- use some other objective
  would be nice to find theorems that are useful in proving other theorems
  – but how exactly would that work?

Thank you for your attention!

Stanisław Purgał