universität
innsbruck

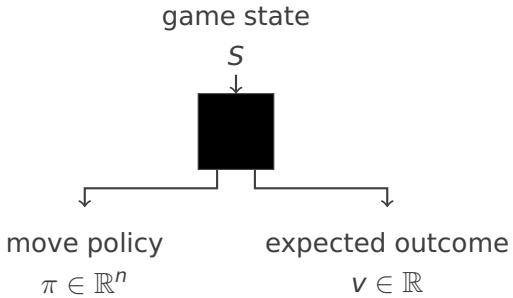
Learning theorem proving through self-play

Stanisław Purgał

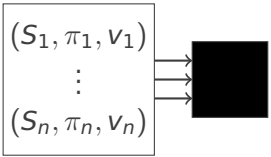
Overview

- AlphaZero
- Proving game
- adjusting MCTS for proving game
- some results

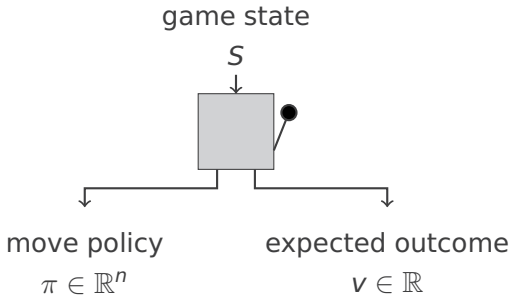
Neural black box



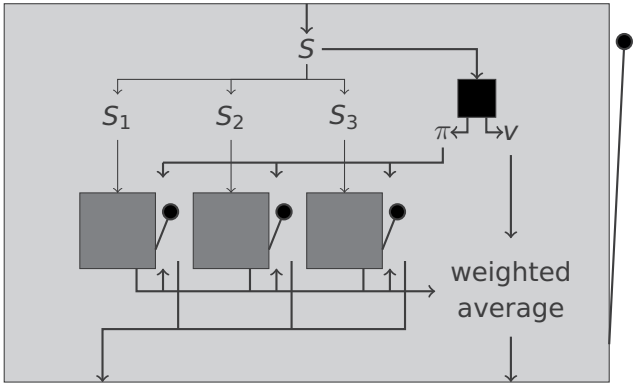
Neural black box



Monte-Carlo Tree Search



Monte-Carlo Tree Search



choose a child according to the formula:

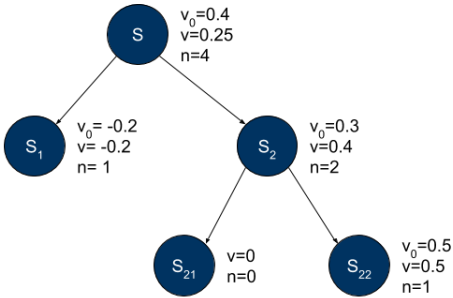
$$c \cdot \frac{\sqrt{n}}{n_i}\pi_i + v_i$$

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

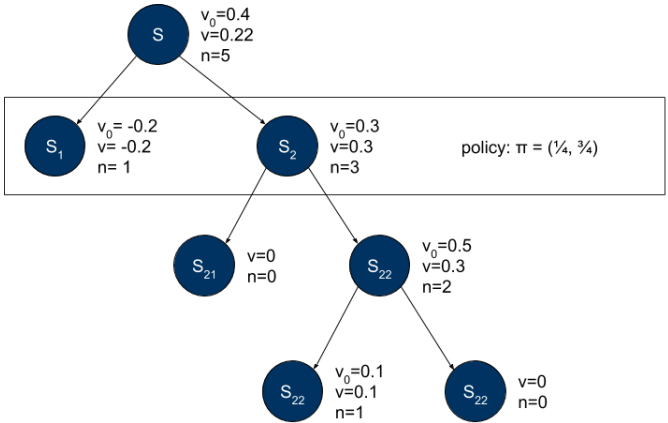
$$c_{base} = 19652$$

$$c_{init} = 1.25$$

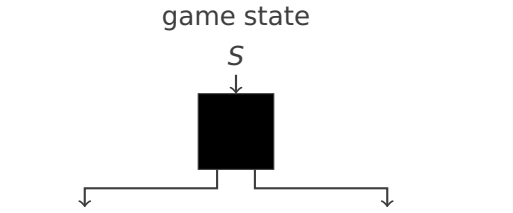
Monte-Carlo Tree Search



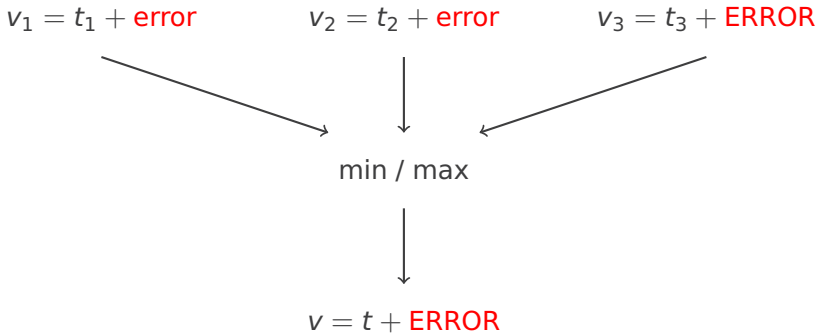
Monte-Carlo Tree Search



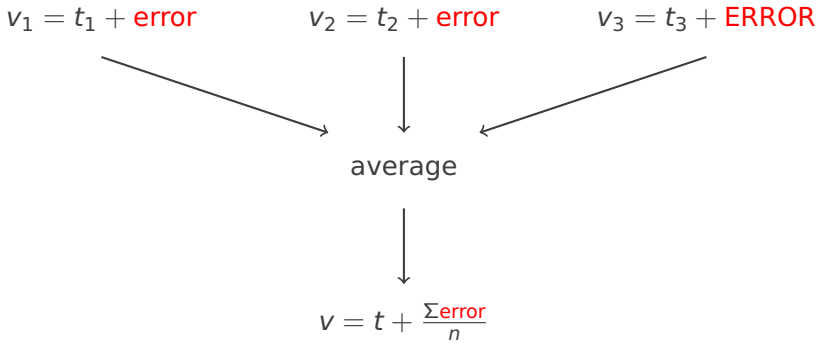
Why not maximum?



Why not maximum?



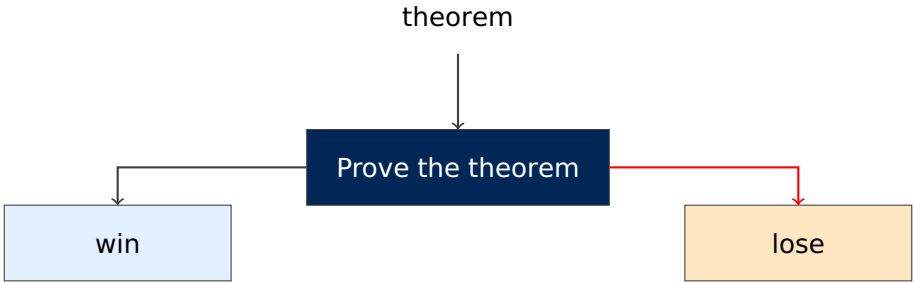
Why not maximum?



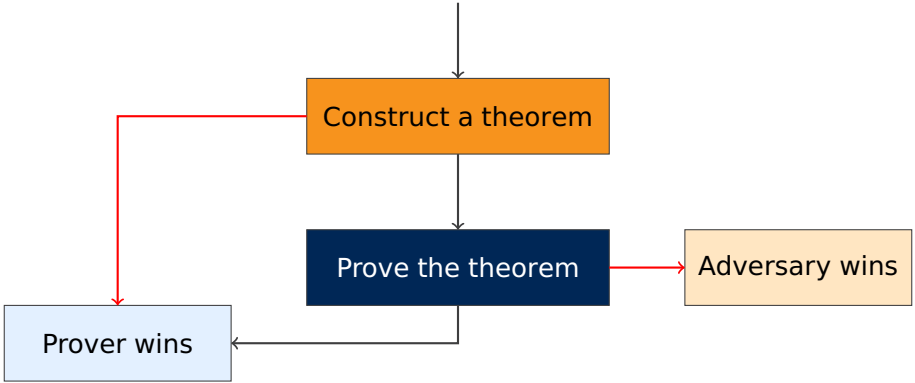
Closing the loop

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

Proving game



Proving game

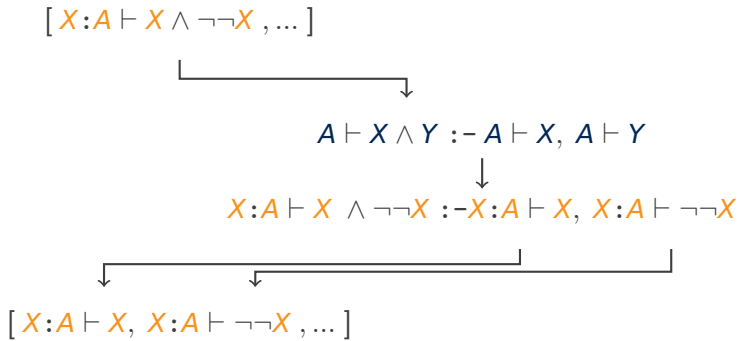


Prolog-like proving

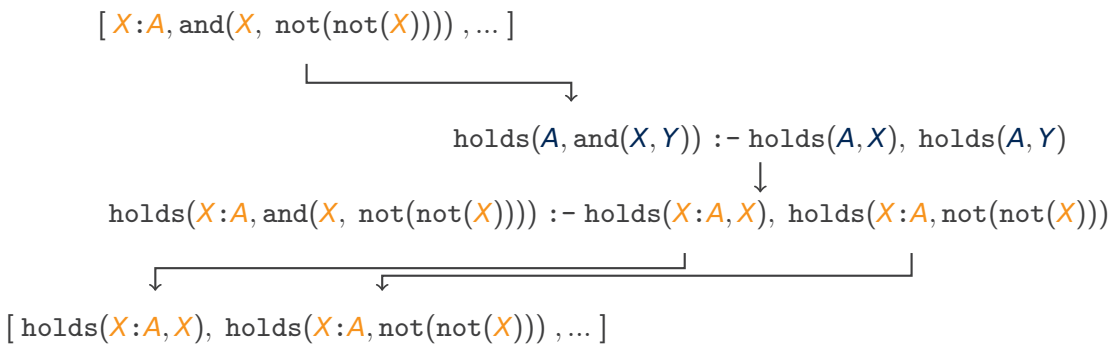
$$\frac{A \vdash X \qquad A \vdash Y}{A \vdash X \wedge Y} \tag{1}$$

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

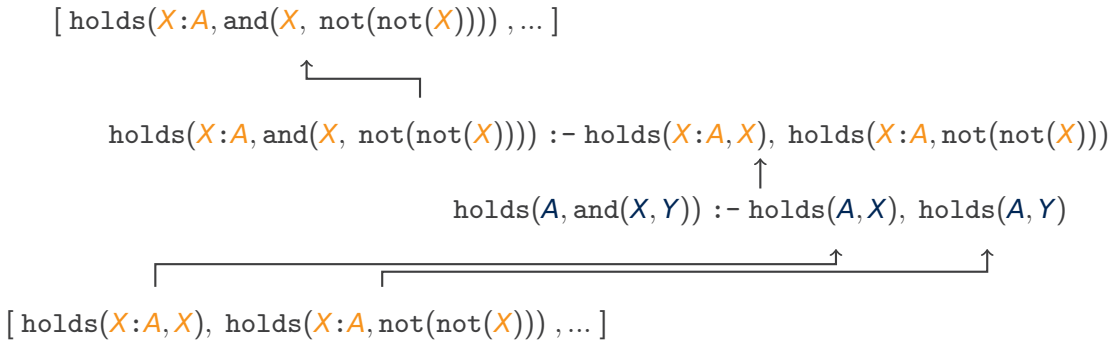
Prolog-like proving



Prolog-like proving

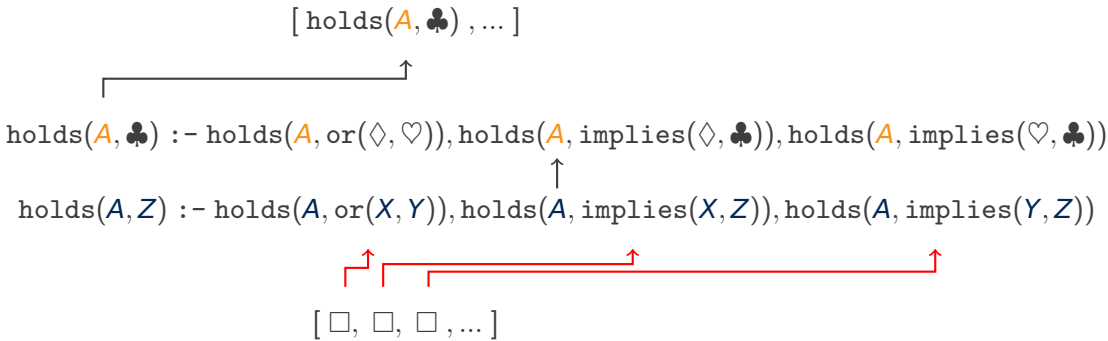


Prolog-like theorem constructing

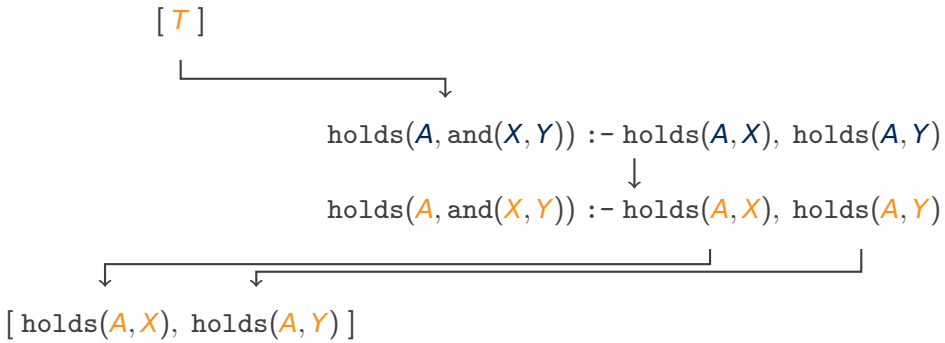


bad idea

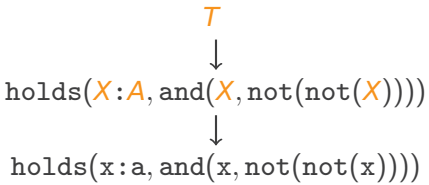
Prolog-like theorem constructing



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:

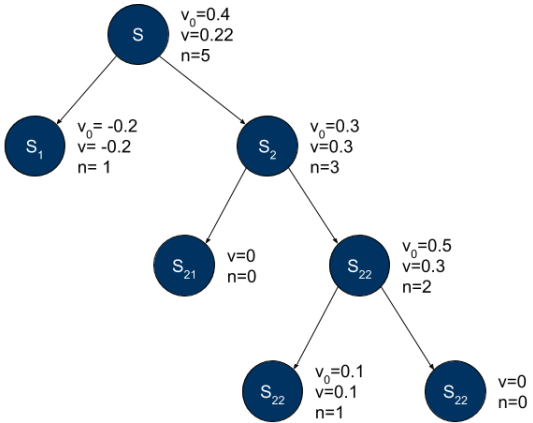
- game states nicely equal
- no hard limit for length of a theorem

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

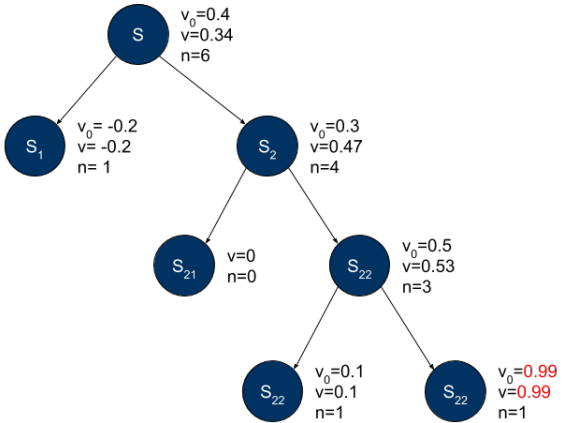
Disadvantages of this game

- 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

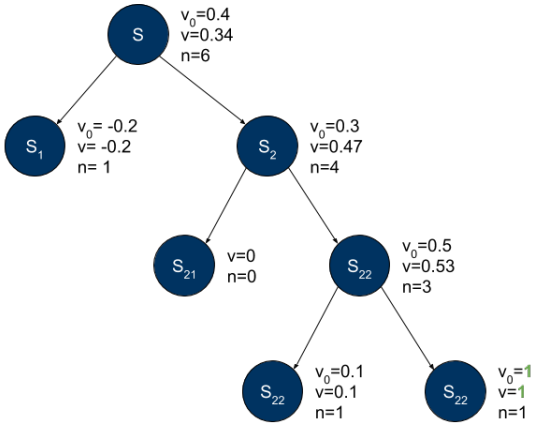
Not using maximum



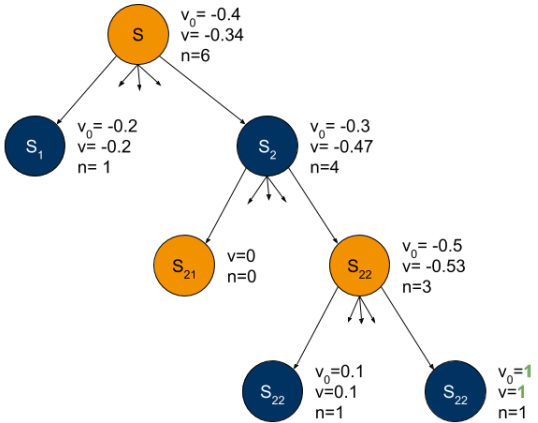
Not using maximum



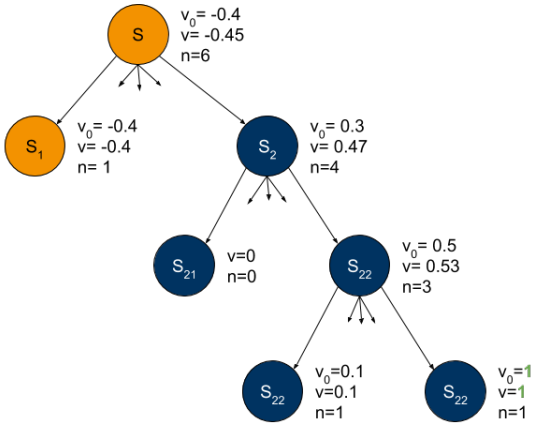
Not using maximum



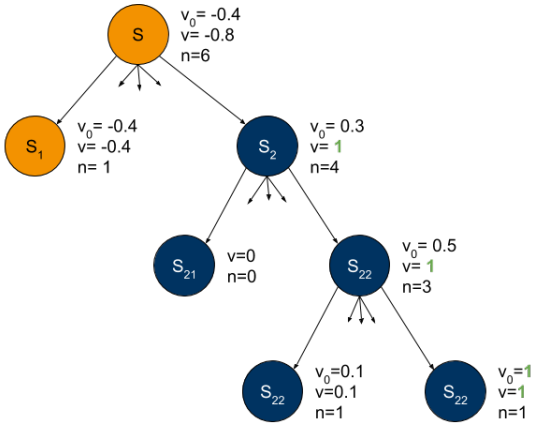
Not using maximum



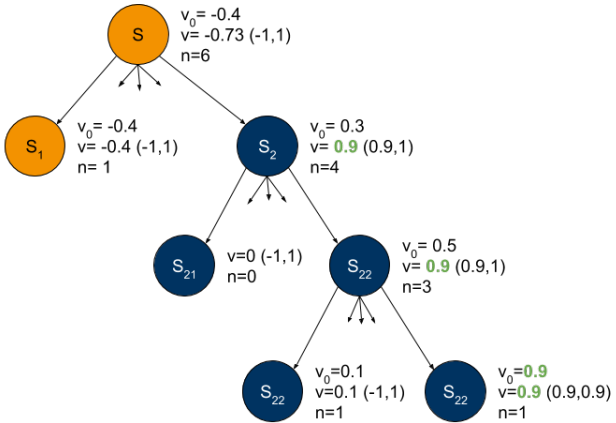
Certainty propagation



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

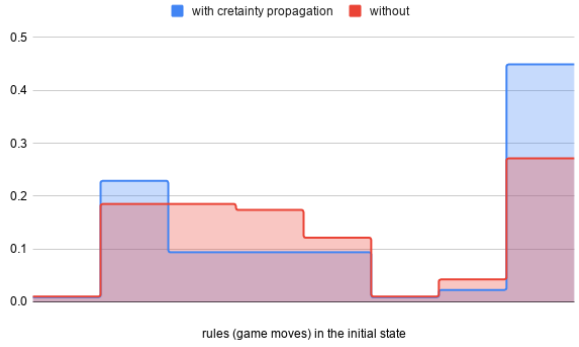
Toy problem

```
ablist([]).
ablist([a|L]) :- ablist(L).
ablist([b|L]) :- ablist(L).
ablist([c|L]) :- ablist(L).
ablist([d|L]) :- ablist(L).
rev3([],L,L).
rev3([H|T],L,Acc) :- rev3(T, L, [H|Acc]).
revablist(L) :- ablist(T), rev3(L, T, []).
```

Toy problem evaluation

```
ablist([a,b,a,b,a,b,b]),
revablist([]),
revablist([a]),
revablist([b]),
revablist([c,d]),
revablist([c,a,b]),
revablist([a,d,c,b]),
revablist([a,d,c,a,a]),
revablist([a,b,c,d,b,d]),
revablist([d,b,c,a,d,a,b]),
revablist([a,c,b,a,c,a,d,d])
```

Certainty propagation effect

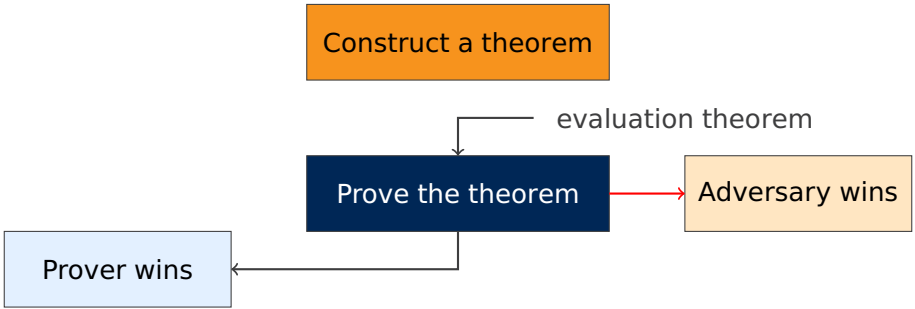


Learning the proving game

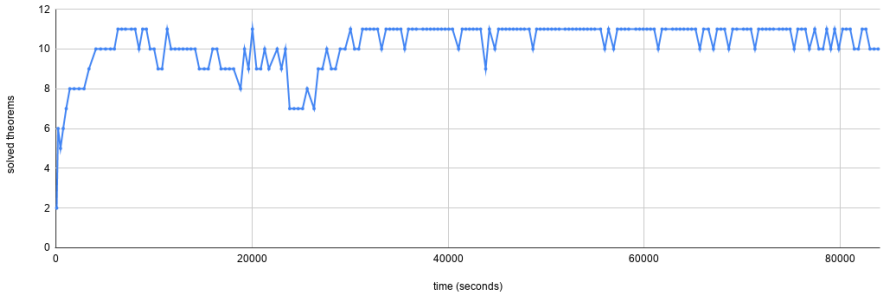
Like AlphaZero, with few differences:

- using Graph Attention Network for ■
- for theorems that prover failed to prove, show proper path with additional policy training samples
- during evaluation, greedy policy and step limit instead of sudden death

Proving game evaluation



Learning toy problem



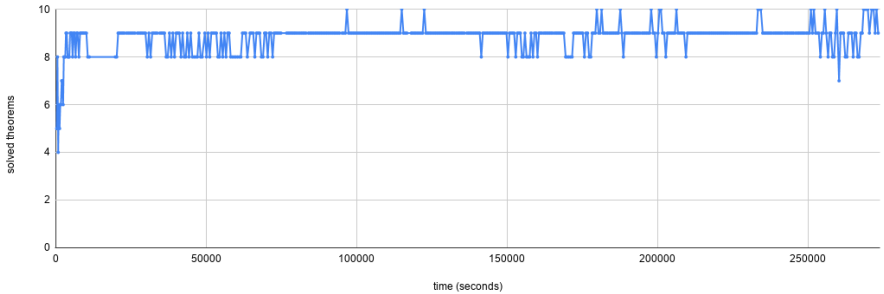
Intuitionistic propositional logic

```
holds([A|T], A).
holds(T, A) :- holds([B|T], A), holds(T, B).
holds([H|T], A) :- holds(T, A).
holds(T, impl(A, B)) :- holds([A|T], B).
holds(T, B) :- holds(T, A), holds(T, impl(A, B)).
holds(T, or(A, B)) :- holds(T, A).
holds(T, or(A, B)) :- holds(T, B).
holds(T, C) :- holds(T, or(A, B)), holds([A|T], C), holds([B|T], C).
holds(T, and(A, B)) :- holds(T, A), holds(T, B).
holds(T, A) :- holds(T, and(A, B)).
holds(T, B) :- holds(T, and(A, B)).
holds([false|T], A).
```

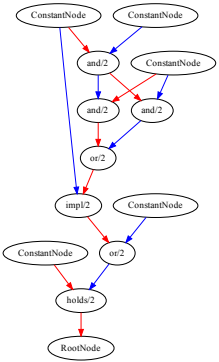
Classical propositional logic

```
holds([A|T], A).
holds(T, A) :- holds([B|T], A), holds(T, B).
holds([H|T], A) :- holds(T, A).
holds(T, impl(A, B)) :- holds([A|T], B).
holds(T, B) :- holds(T, A), holds(T, impl(A, B)).
holds(T, or(A, B)) :- holds(T, A).
holds(T, or(A, B)) :- holds(T, B).
holds(T, C) :- holds(T, or(A, B)), holds([A|T], C), holds([B|T], C).
holds(T, and(A, B)) :- holds(T, A), holds(T, B).
holds(T, A) :- holds(T, and(A, B)).
holds(T, B) :- holds(T, and(A, B)).
holds([false|T], A).
holds(T, A) :- holds([impl(A, false)|T], false).
```

Learning classical propositional logic



Constructed theorem example



$$\vdash (((d \wedge b \wedge c) \vee (b \wedge c \wedge d)) \implies b) \vee e$$
$$\bot \vdash a \vee b \vee c$$
$$\vdash ((((a \wedge \bot \wedge b) \implies c) \implies d) \implies d)$$
$$((a \wedge b) \implies a) \implies (\bot \wedge c) \vdash d$$
$$((a \implies \bot) \implies b)\,,\ c\,,\ (a \implies b) \vdash b$$

First-order logic

```
%some classical logic

neq(var([a|_]), var([b|_])).
neq(var([b|_]), var([a|_])).
neq(var([_|A]), var([_|B])) :- neq(var(A), var(B)).

repl(var(A), R, var(A), R).
repl(var(A), R, var(B), var(B)) :- neq(var(A), var(B)).
repl(var(A), R, op(O, X1, Y1), op(O, X2, Y2)) :- repl(var(A), R, X1, X2), repl(var(A), R, Y1, Y2).
repl(var(A), R, q(O, var(A), P), q(O, var(A), P)).
repl(var(A), R, q(O, var(B), P1), q(O, var(B), P2)) :- neq(var(A), var(B)), repl(var(A), R, P1, P2).
repl(var(A), R, false, false).
repl(var(A), R, [], []).
repl(var(A), R, [H1|T1], [H2|T2]) :- repl(var(A), R, H1, H2), repl(var(A), R, T1, T2).

holds(T, q(forall, var(A), Phi)) :- repl(var(A), var(B), Phi, PhiBA), repl(var(B), false, [Phi|T], [Phi|T]), holds(T, PhiBA).
holds(T, Phi) :- holds(T, q(forall, var(A), PhiA)), repl(var(A), B, PhiA, Phi).

holds(T, q(exists, var(A), Phi)) :- repl(var(A), R, Phi, PhiR), holds(T, PhiR).
holds(T, P) :- holds(T, q(exists, var(A), Phi)), repl(var(B), false, Phi, Phi), repl(var(A), var(B), Phi, PhiB), holds([PhiB|T], P).
```

Future work

- better rule representation?
- proper prover with a different construction mechanism?
- different use cases?
- more computational power?

Thank you for your attention!

Stanisław Purgał