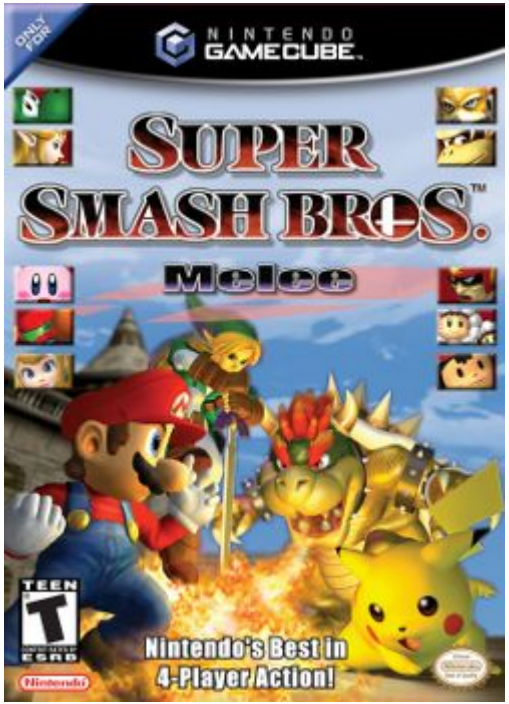
# Tackling Super Smash Bros. Melee with Deep RL

Vlad Firoiu

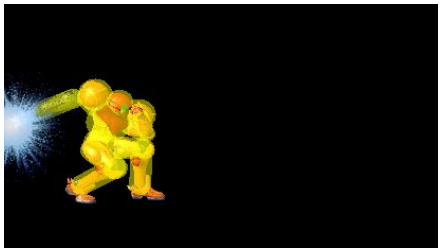

NINTENDO GAMECUBE
SUPER SMASH BROS.
Melee
TEEN
T
Nintendo's Best in 4-Player Action!

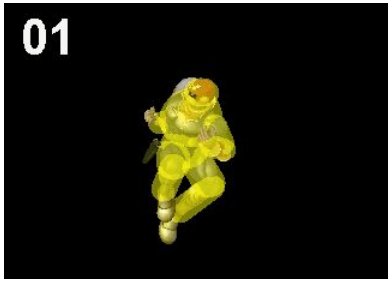

NINTENDO GAMECUBE
NINTENDO GAMECUBE


A series of attacks ending in a KO.

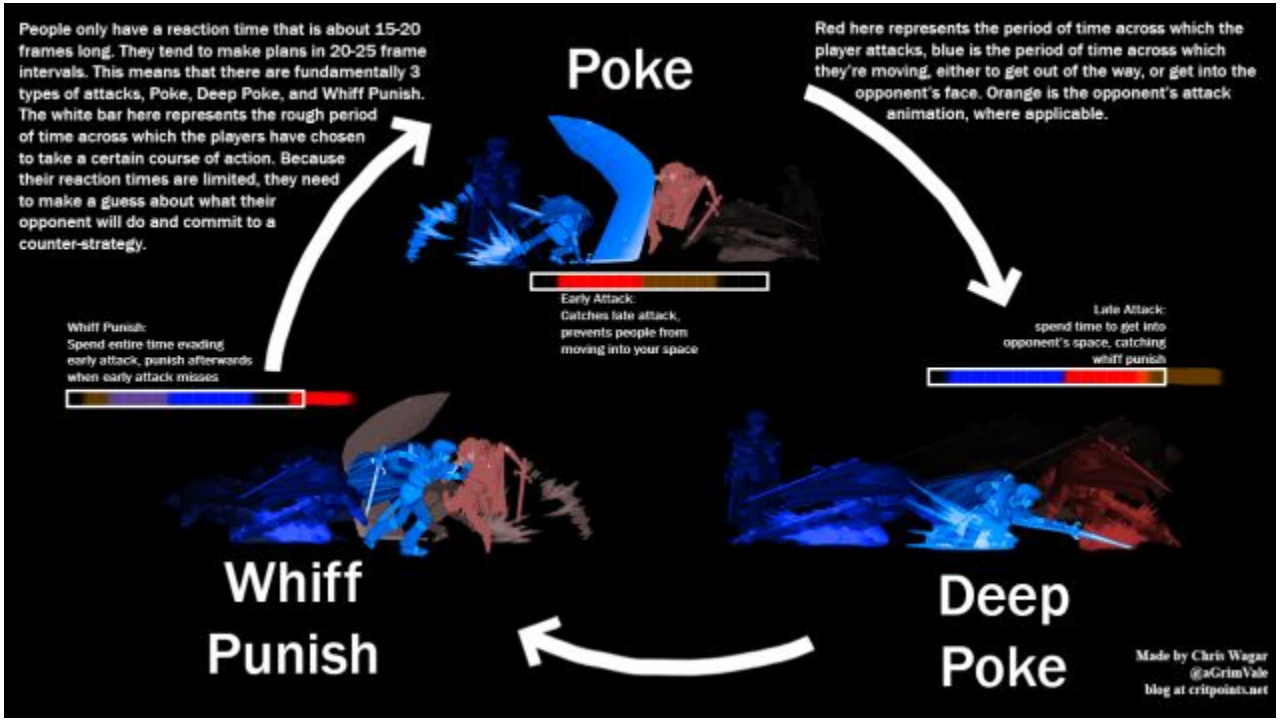

Top-level gameplay. Not sped up!

# “Yomi” in Melee





# “Yomi” in Melee



# The RL Environment

- Environment simulated with the Dolphin emulator
  - 1-2x real time on server hardware (no graphics)
- Game state is read from RAM on each frame
  - Player positions, velocities, facing direction, etc
  - 382 (!) action states (running, jumping, attacking)
  - mostly observable
- Action space
  - Full controller has 2 analog control sticks, 7 buttons, 2 triggers.
  - Compressed down to ~50 discrete actions.
- Reward structure
  - +/- 1 for kills/deaths
  - +/- 0.01 for damage dealt/taken

# Methods

- RL: Importance-Weighted Advantage Actor-Critic (IMPALA)
  - 50-200 environments per experiment.
  - Early versions had no importance weighting.
- DL: small MLP
  - Initially frame stacking.
  - Later added a GRU.
- Symmetric self-play
  - Against exact same parameters → double throughput.
  - Against mixture of old checkpoints.
    - Indicates rate of improvement over time.
  - Modern solution is AlphaStar League.

# Results

After 1 day: garbage. Left the experiment running and forgot about it.

After 1 week… very strong, better than me.

After 1 more week, could beat professional players.

| Opponent | Rank | Kills | Deaths |
|---|---|---|---|
| S2J | 16 | 4 | 2 |
| Zhu | 31 | 4 | 1 |
| Gravy | 41 | 8 | 5 |
| Crush | 49 | 3 | 2 |
| Mafia | 50 | 4 | 3 |
| Slox | 51 | 6 | 4 |
| Redd | 59 | 12 | 8 |
| Darkrain | 61 | 12 | 5 |
| Smuckers | 64 | 8 | 5 |
| Kage | 70 | 4 | 1 |

# Limitations

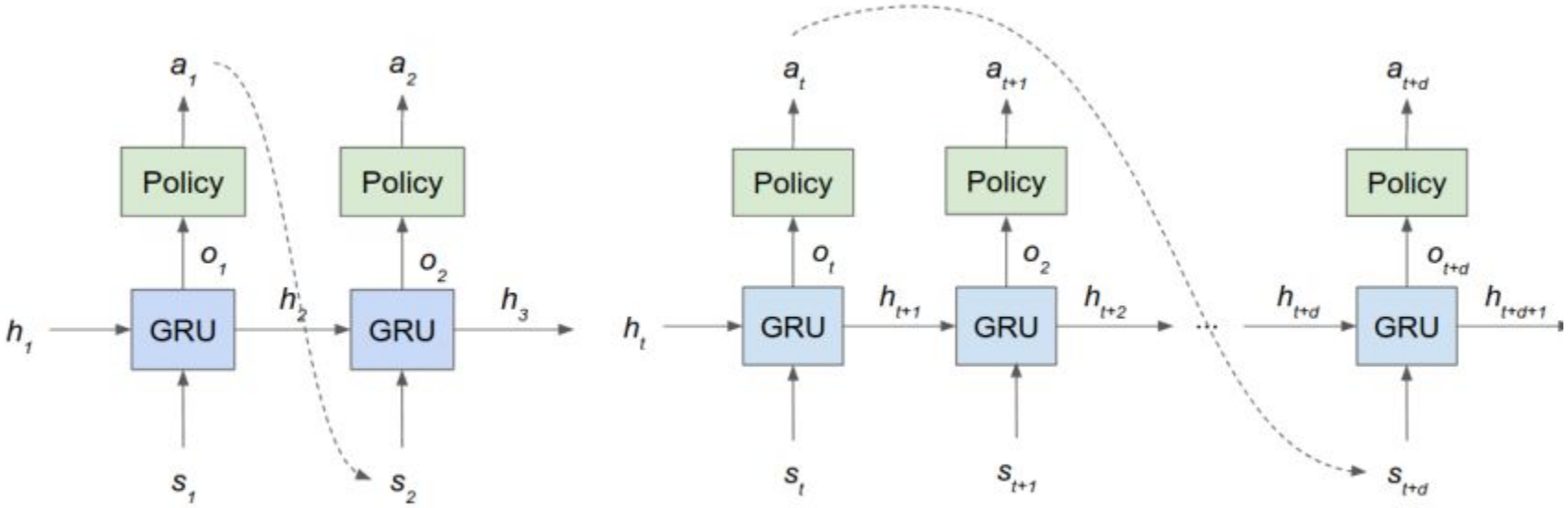

A simple exploit.


Inhuman speed and reaction time.

# Deep RL with Action Delay

- Action delay levels the playing field between human and AI
  - Removes “degenerate” inhuman behavior
- Humans have ~300ms visual reaction time
  - Varies with complexity of task (250-800)
  - Corresponds to 6 agent actions
- Issues for Deep RL
  - Makes credit assignment harder
  - Makes control much harder
  - Correct action depends on heavily on unknown future state

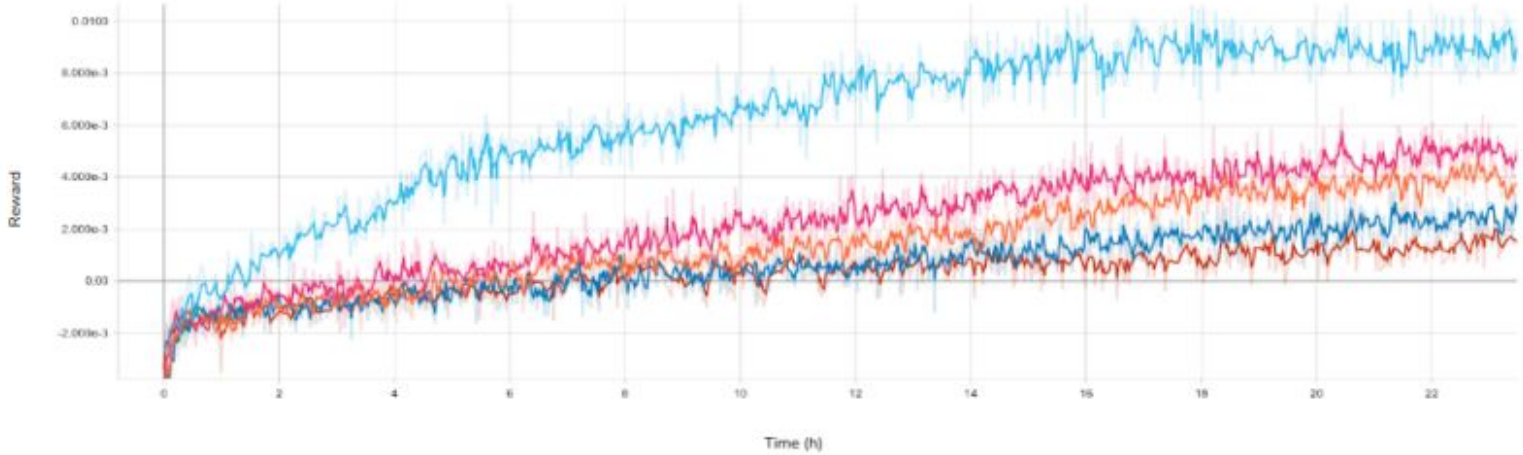
# Deep RL with Action Delay




(a) An agent unrolled over time.

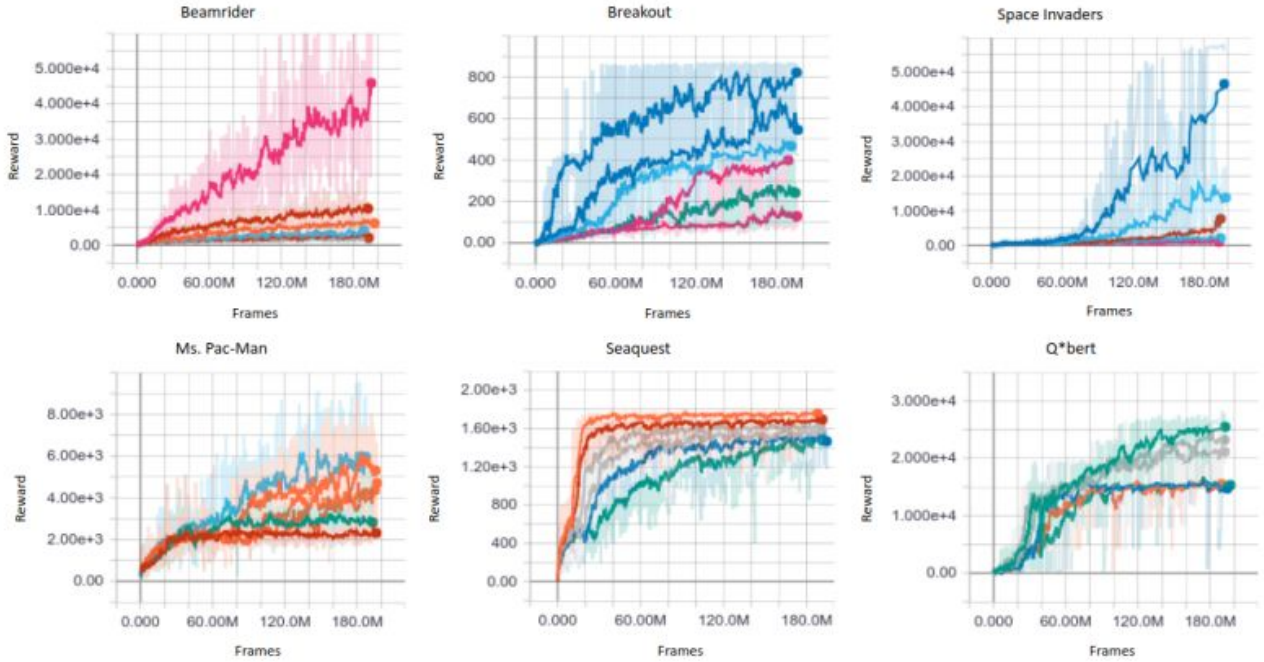
(b) A delayed agent unrolled over time.

# The Action Delay Problem: SSBM




Training vs. the in-game AI with delays of 0 (light blue), 1(magenta), 2 (orange), 4 (dark blue) and 5 (brown) agent steps. Each agent step is 50 ms.

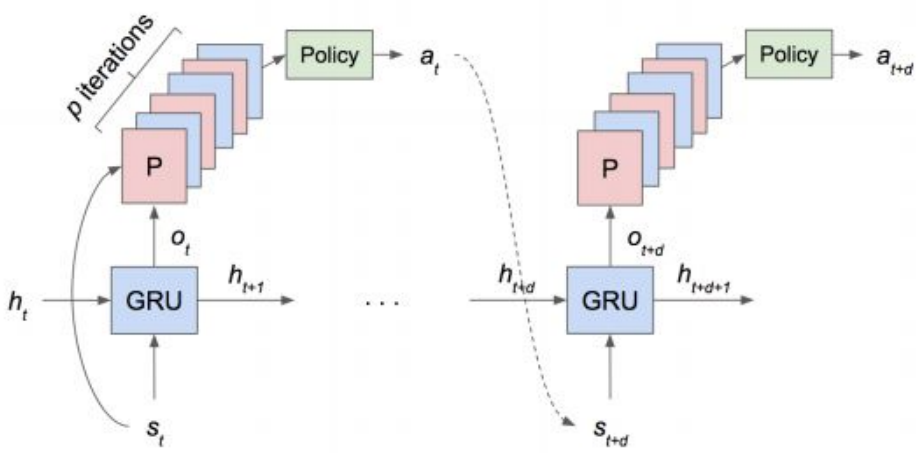
# The Action Delay Problem: Atari



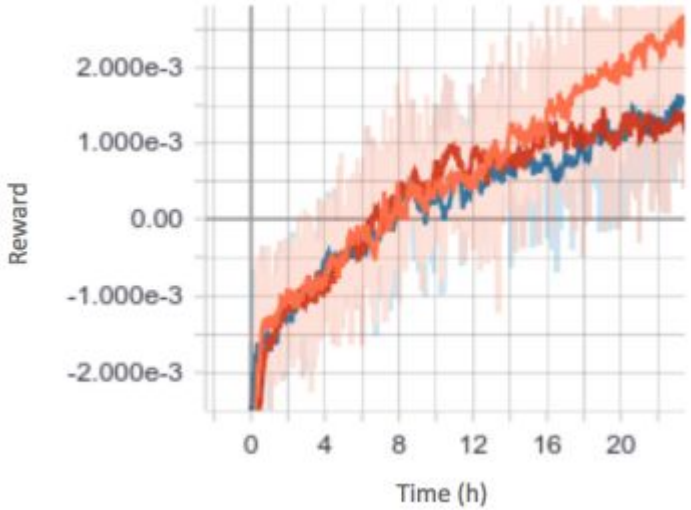

IMPALA trained on Atari with between 0 and 5 steps of delay (up to 333ms).

# Solution: “Undoing” Delay

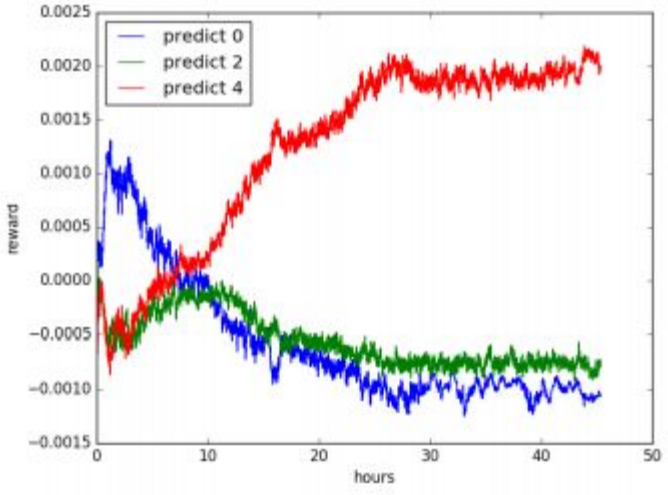
- We perceive moving objects slightly *ahead in space*.
  - See “Flash-Lag Effect”, an optical illusion.
- Use a learned environment model to do the same.
  - Gives the policy a better estimate of the future state.



# Results of Undoing Delay




Delay 4 against in-game AI, predicting 0 (blue), 2 (red) and 4 (orange) frames into the future.




Three delay=4 agents co-training for two days.

# Results vs. Human Opponents

Good, but not superhuman.

More “human-like” than before, but still relies on very precise reactions.

Later agents went up to 300ms, but fell short of pro level.

| Agent | | | | |
|---|---|---|---|---|
| Delay | Prediction Steps | Days Trained | Wins | Losses |
| 6 | 0 | 7 | 0 | 6 |
| | 6 | 3 | 3 | 5 |
| 7 | 7 | 10 | 2 | 5 |

Performance against Professor Pro, a top-50 player.
Each win/loss is in a 4-stock match.
For this agent, 7 frames = 233ms.

# Remarks & Future Work

- Delay not solved yet
  - Opponent is considered part of environment
  - Could combine MuZero with delay
- Exploration not solved yet
  - RL alone can’t discover some important techniques and strategies
  - OpenAI 5 needed heavy reward shaping
  - AlphaStar used millions of human games
- Imitation learning for Smash Bros
  - Already a dataset of 100K tournament matches
  - Many more games are being recorded in the online “covid” era

# Training a First-Order Theorem Prover from Synthetic Data

**Vlad Firoiu**, Eser Aygun, Ankit Anand, Zafarali Ahmed, Xavier Glorot, Laurent Orseau, Lei Zhang, Doina Precup, Shibl Mourad

# What is (Formal) Theorem Proving?

Just two easy steps:

1. Encode the rules of logic in a computer.
2. State and prove theorems in this system.

Takes a lot of skill and patience. Not generally used by mathematicians.

An analogy:

- Informal proofs are like sketches or design documents.
- Formal proofs are their implementations in code.

# Theorem Proving as a Domain for AI

Many advantages:

- Environment is fully specified (by logic) and efficient to simulate.
- Rich, composable, and open-ended task space.
- Easy to express very challenging intellectual tasks.
- Includes many problems of practical interest.
  - Software verification, program synthesis, physical and chemical simulations

Some challenges:

- Expanding state and action spaces.
- **Relative lack of supervised data.**

# Our Approach

Dataset: **T**housands of **P**roblems for **T**heorem **P**rovers (TPTP)

- Benchmark for Automated Theorem Provers (ATPs) for over a decade.
- Best ATPs are *E* and *Vampire*. Kind of like Stockfish and built for TPTP.
- We use a subset of 606 “first-order without equality” problems.
  - 10 axioms sets covering algebra, geometry, number theory, and set theory

No test theorems ever used during training!

- Synthetically **propose** training problems from axioms.

Use “Reinforcement Learning” to train a neural prover.

# The Resolution Calculus

Resolution Calculus: a set of inference rules for first-order **clauses**.

A clause is a type of first-order proposition, e.g. "$\forall$ X, $p$(X) $\Rightarrow$ $q$(X)".

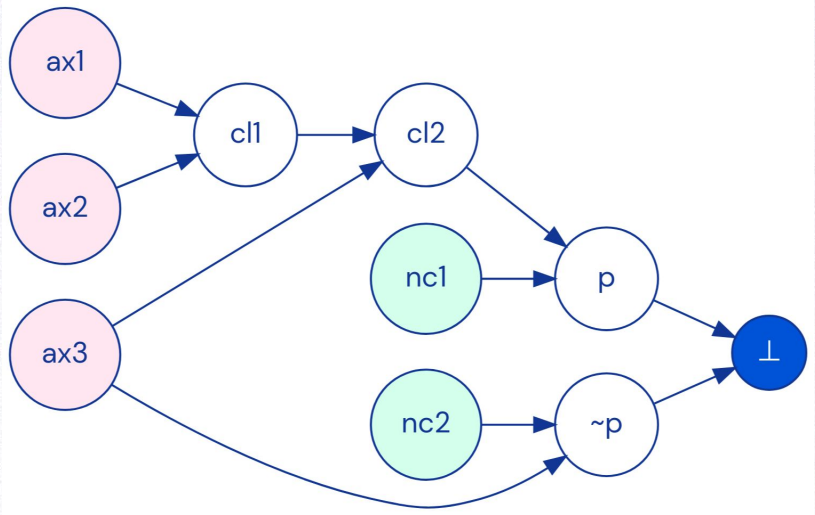
An inference is the implication of one clause by one or more other clauses.

- A, (A $\Rightarrow$ B) $\vdash$ B

Refutation proving:

- Start with **negated** conjecture as set of clauses.
- Use the resolution calculus to derive **false**.

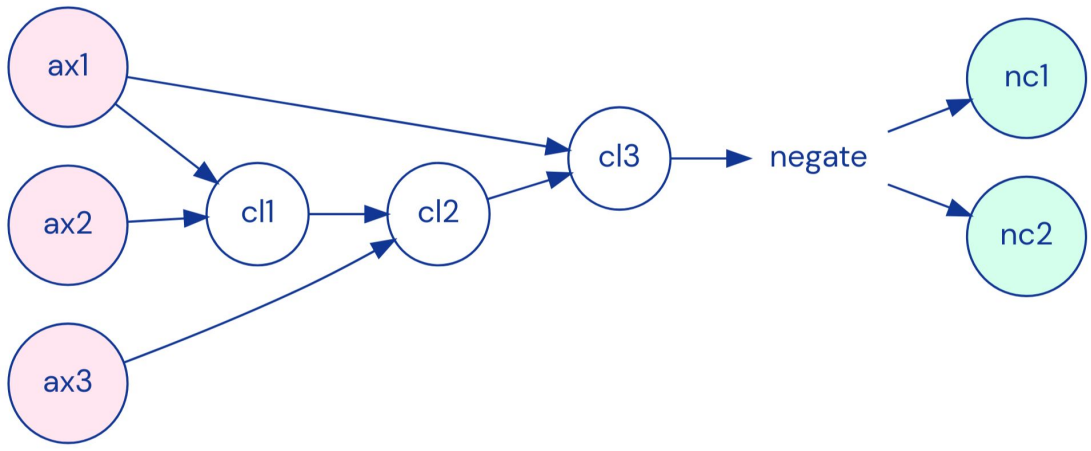
# Refutation Proving with the Resolution Calculus




[Conjecture] = [Axioms ⇒ Conclusion]

~[Axioms ⇒ Conclusion] → [Axioms & ~Conclusion]

# The Forward Proposer




Take random(*) resolution steps and negate the final clause.

# The Given-Clause Algorithm

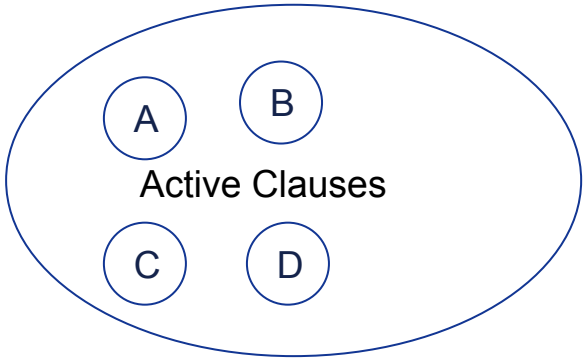
Naively applying RL to resolution doesn't work too well.

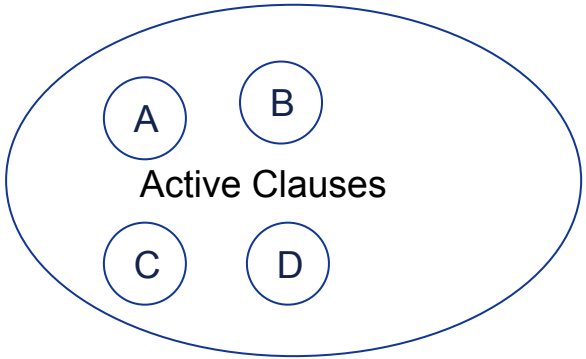
Given-Clause Algorithm: an A*-like search procedure built on top of resolution.

- Used by all the best first-order ATPs

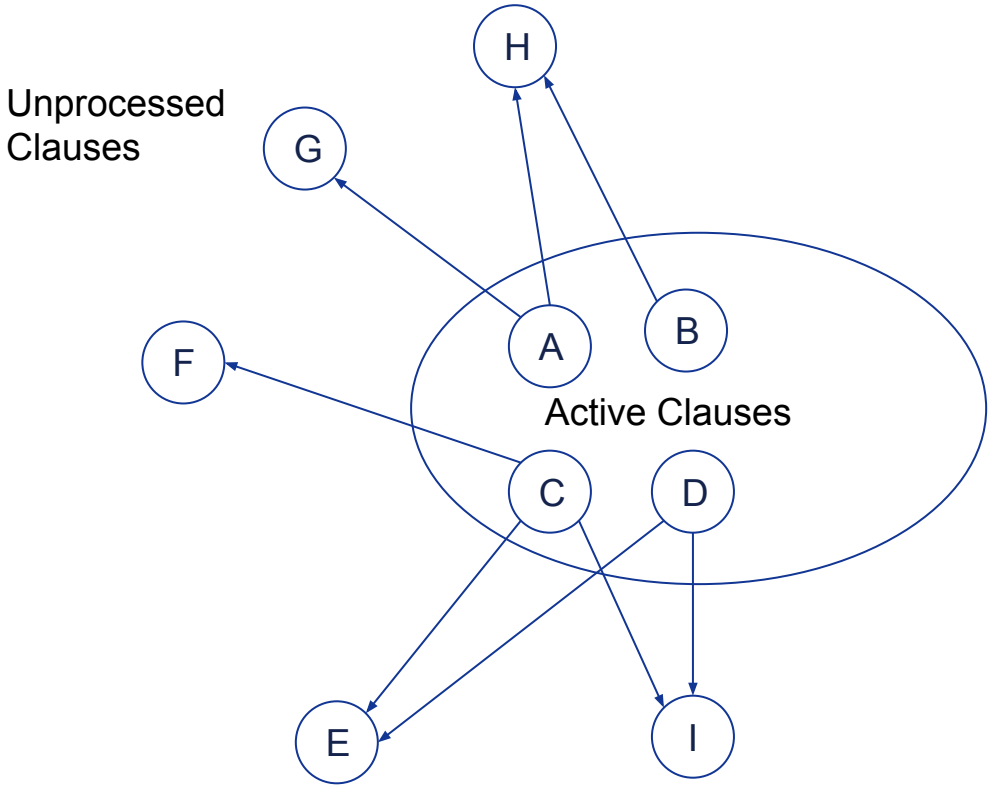
Outline in pseudo-pseudo-code:

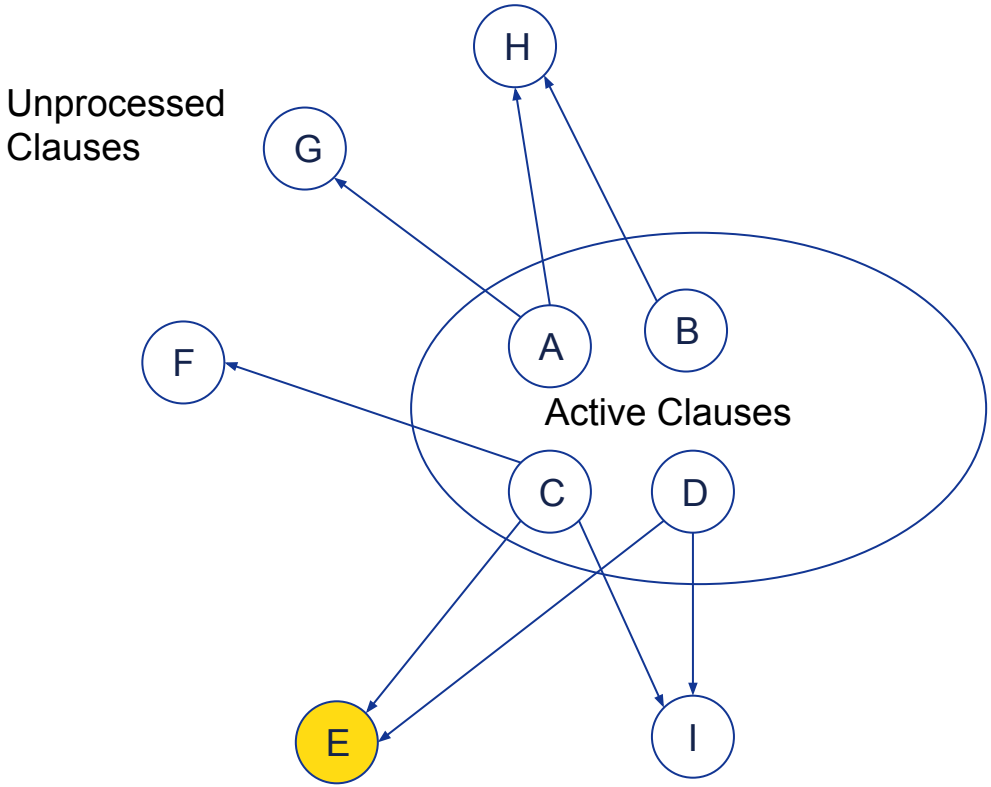
- Each new clause is assigned a *cost* and placed into a priority queue.
- At each step, pick the min-cost clause and generate its consequences.
- Repeat until QED or timeout.

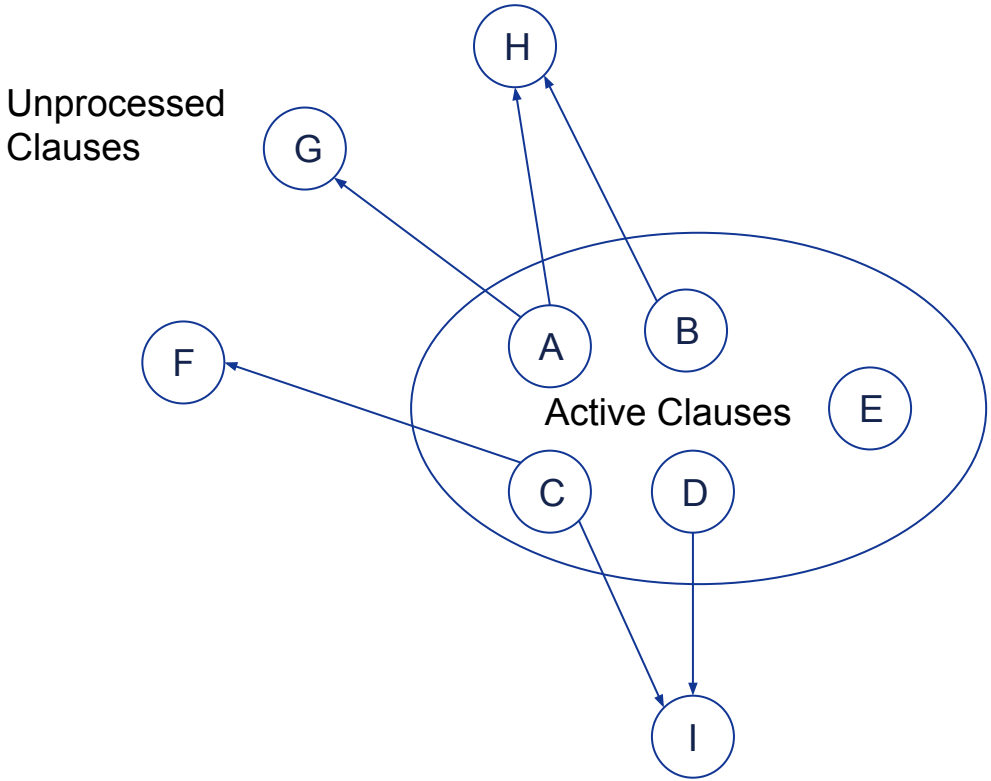

A
B
Active Clauses
C
D

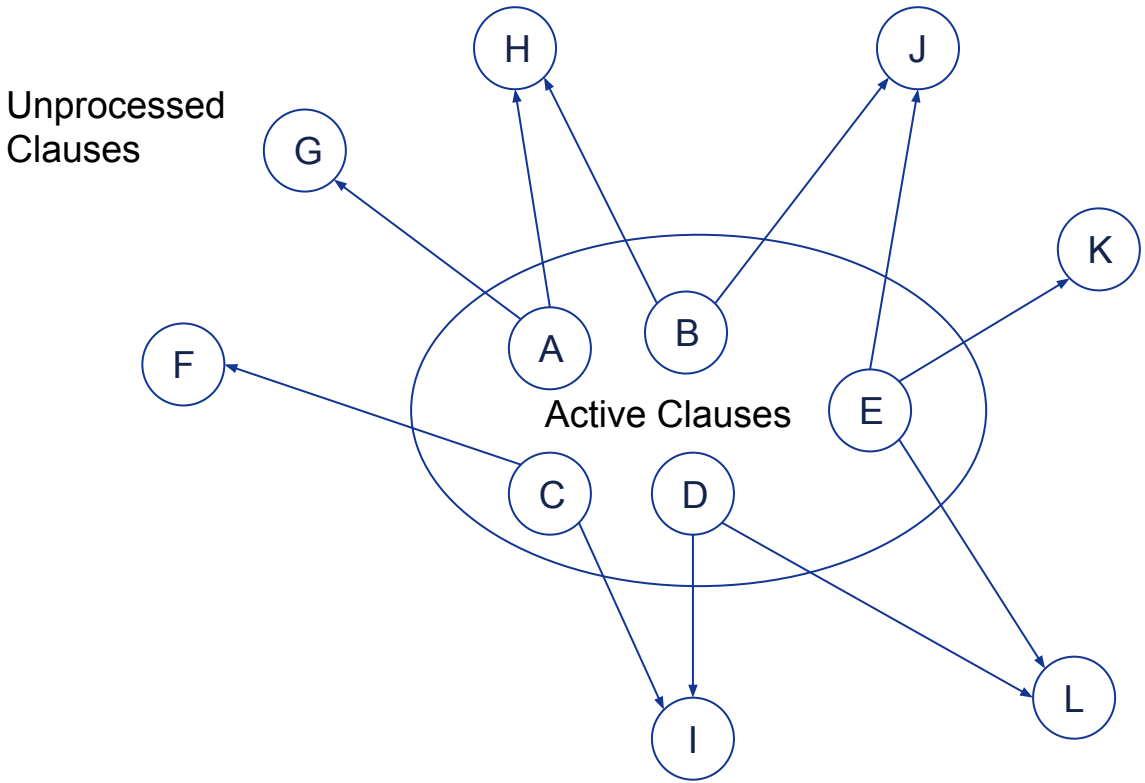


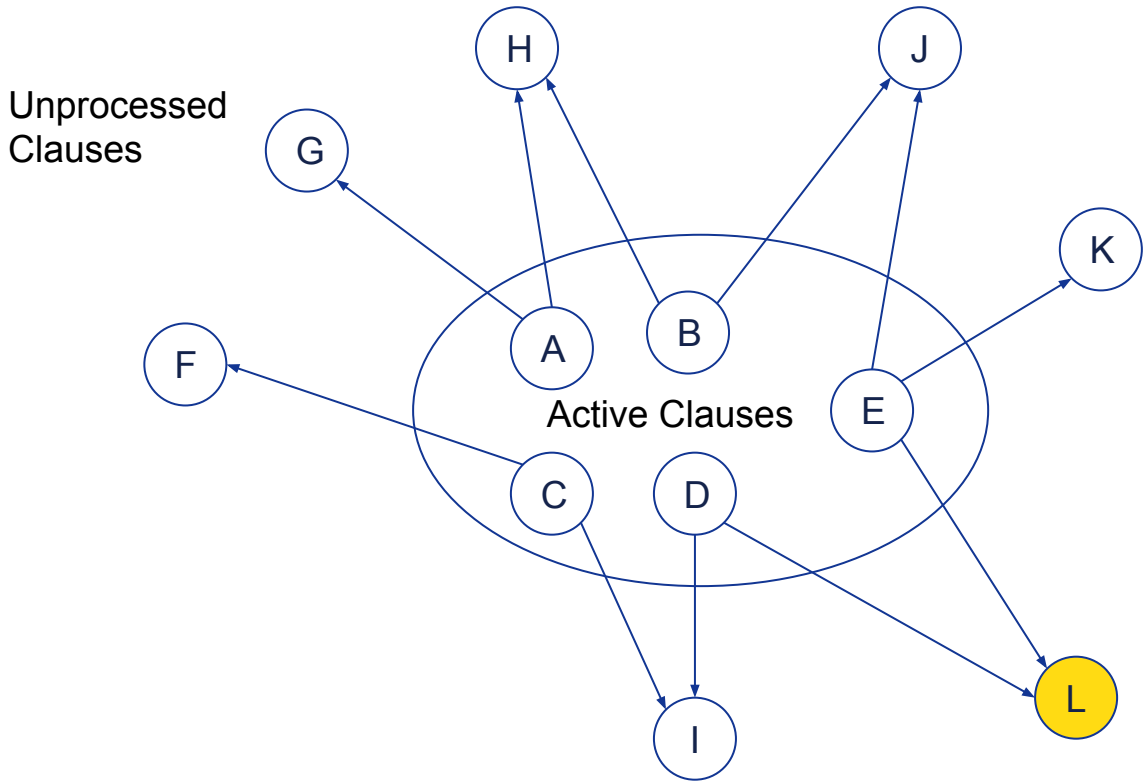

Initially, the axioms and negated conjecture.

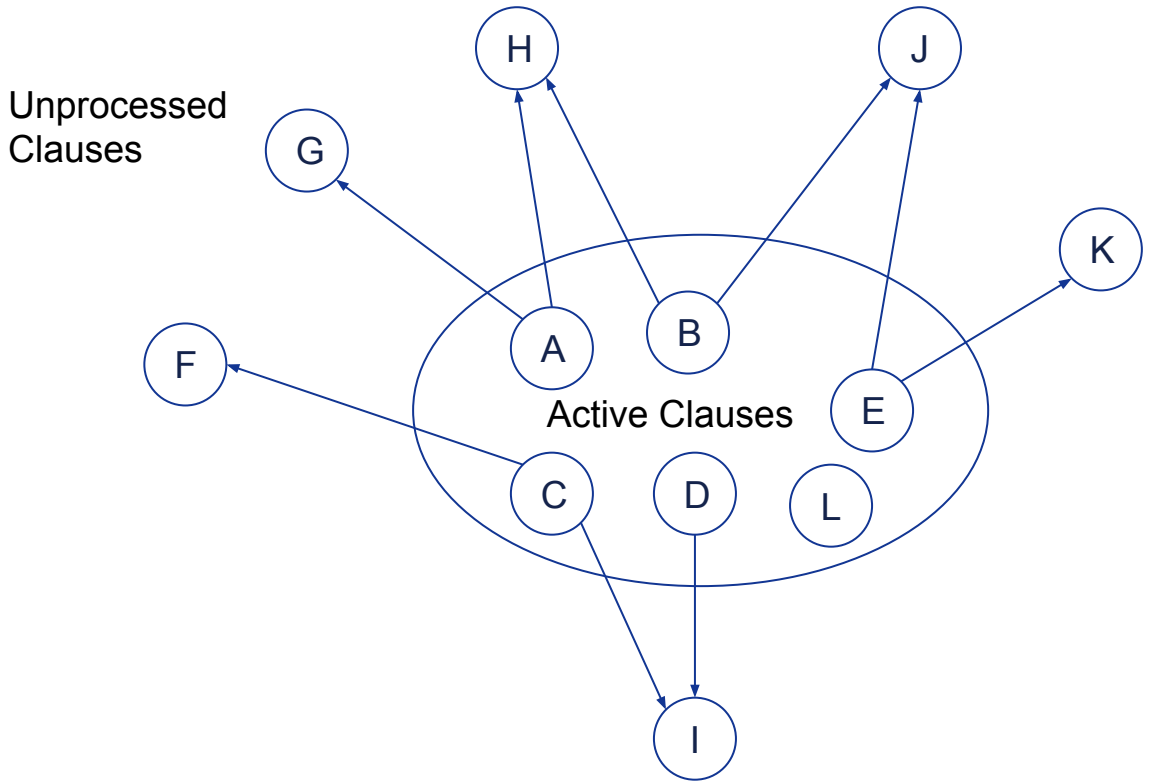

Unprocessed
Clauses
H
G
A
B
F
Active Clauses
C
D
E
I


Unprocessed
Clauses
H
G
A
B
F
Active Clauses
C
D
E
I

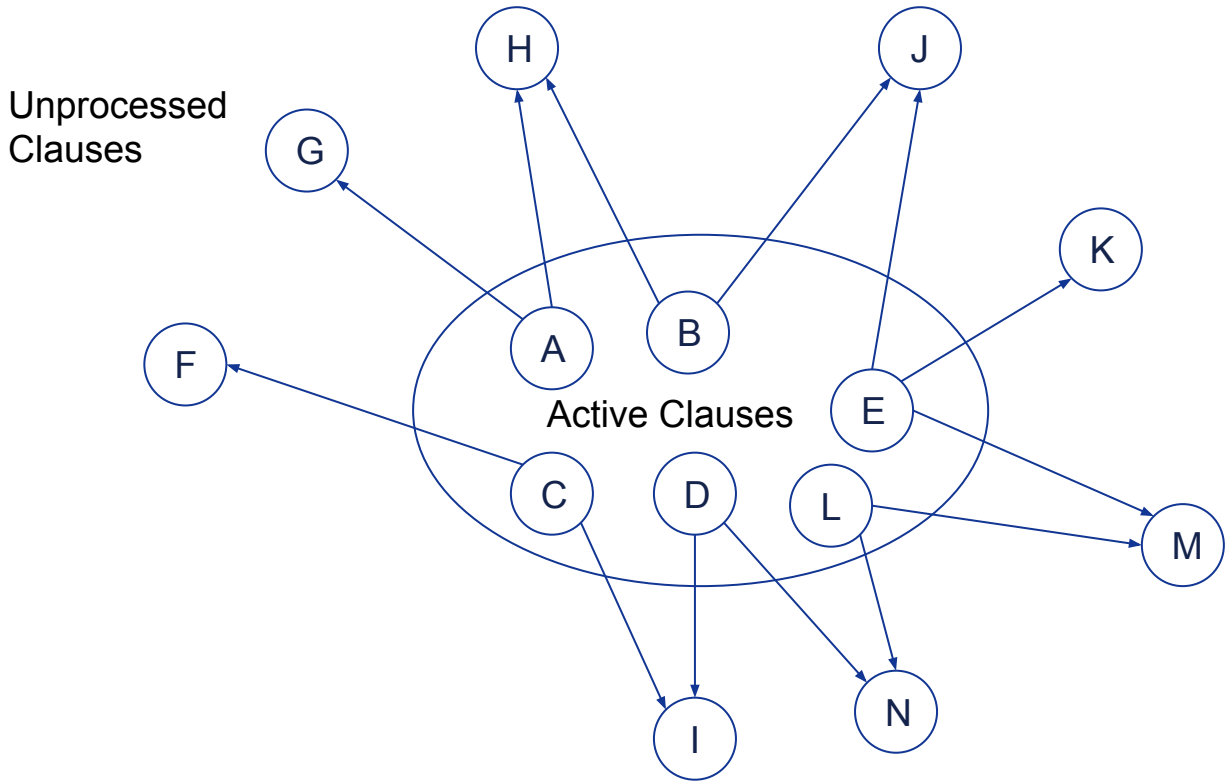

Unprocessed
Clauses
H
G
A
B
F
Active Clauses
E
C
D
I


Unprocessed
Clauses
H
J
G
K
F
A
B
Active Clauses
E
C
D
L
I


Unprocessed
Clauses
H
J
G
K
F
A
B
Active Clauses
E
C
D
L
I


Unprocessed
Clauses
H
J
G
K
A
B
F
Active Clauses
E
C
D
L
I

Private & Confidential



# Learning a Cost Function

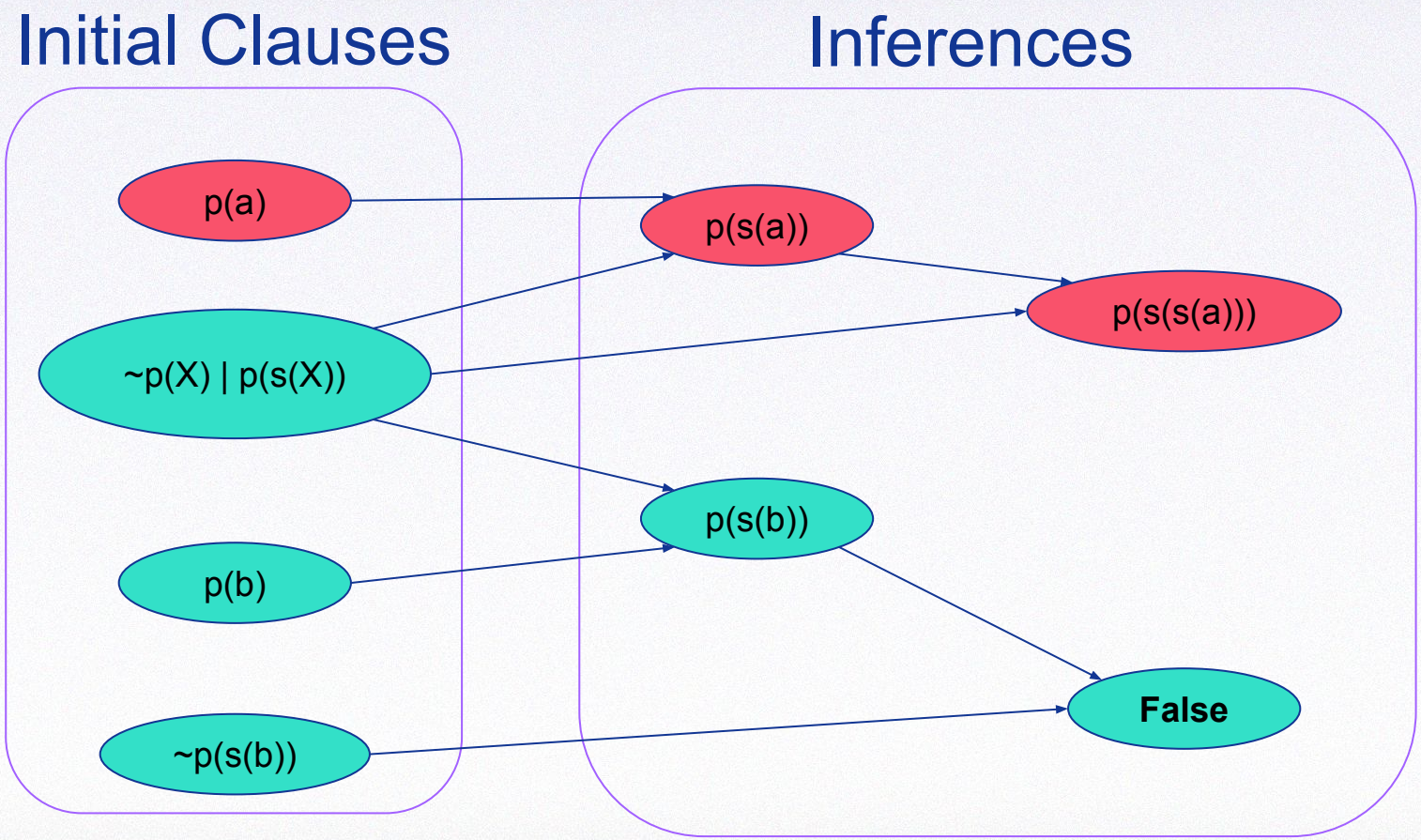
Let's train a neural network to evaluate the clauses.

An optimal cost function would avoid useless clauses.

- A clause is used if it is on the proof path (ancestor of False).

**Appears-in-Proof** binary classification objective.

- Search distillation similar to AlphaZero.
- How to sample positive vs. negative examples?
- What do we do if we couldn't find the proof?


Initial Clauses
Inferences
p(a)
~p(X) | p(s(X))
p(b)
~p(s(b))
p(s(a))
p(s(s(a)))
p(s(b))
False

# Naive RL vs Given-Clause

| Naive RL | Given-Clause |
| --- | --- |
| Linearly growing state: the whole active set. | **Only initial conjecture in state.** |
| Slow policy. Rescores all available inferences. | **Fast policy, only scores new clauses.** |
| Not clause/proof aware. Relies on linear order. | **Rewards only clauses used in proof.** |
| **Dynamic clause scores reflect evolving state.** | Static and independent clause scores. |
| **Theoretically sound value updates given by the Bellman equation.** | Clause scores don't have a well-defined meaning. |
| **Bootstrapping allows learning from failed proofs.** | Can't learn from failed proofs. |

# Naive RL vs Given-Clause

| Naive RL | Given-Clause |
| --- | --- |
| Linearly growing state: the whole active set. | **Only initial conjecture in state.** |
| Slow policy. Rescores all available inferences. | **Fast policy, only scores new clauses.** |
| Not clause aware. Relies on linear order. | **Rewards only clauses used in proof.** |
| **Dynamic clause scores reflect evolving state.** | Static and independent clause scores. |
| **Theoretically sound value updates given by the Bellman equation.** | Clause scores don't have a well-defined meaning. |
| **Bootstrapping allows learning from failed proofs.** | Can't learn from failed proofs. |

# Neural Networks

## MLP

Small fixed-size clause representation from simple statistics like number of variables, functions, literals, etc. that appear in the clause.

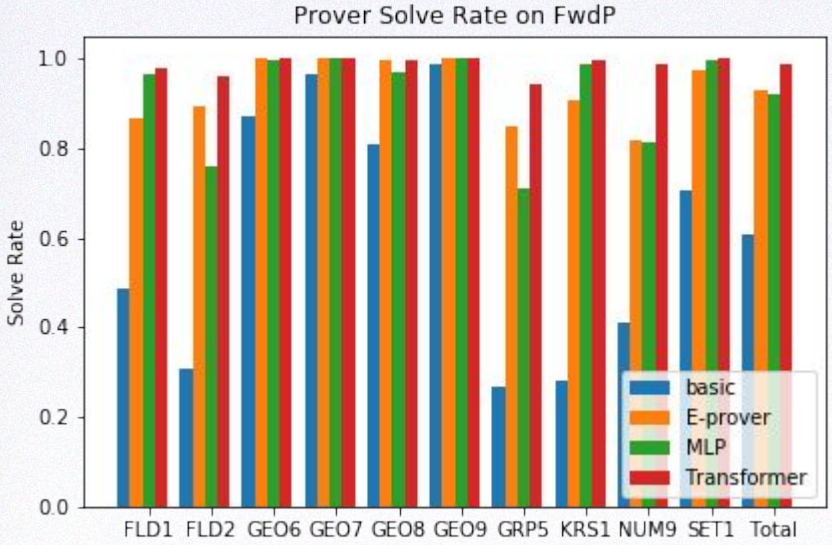
Very low-capacity. Few parameters and major information loss in features.

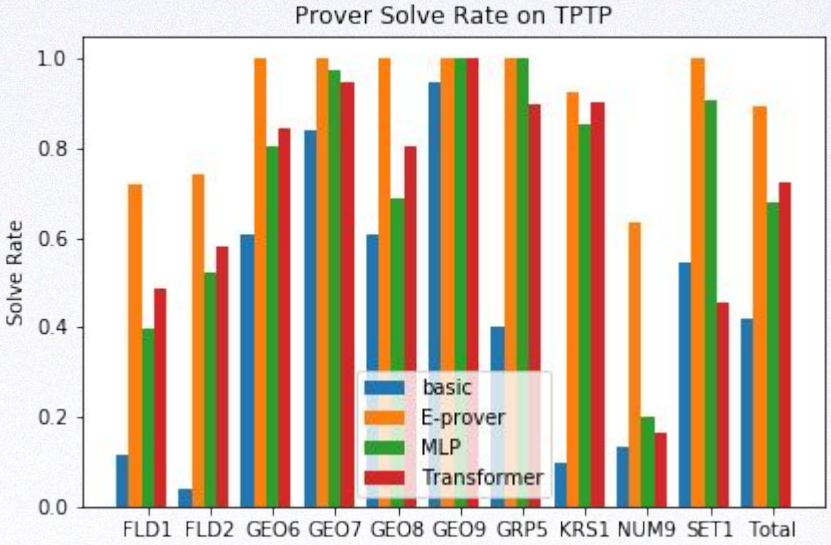
## Transformer

Graphical representation based on syntax tree with spectral features for node embeddings. Function names exposed via stable hash vectors.

High-capacity. Many parameters with full representation of the input clause.

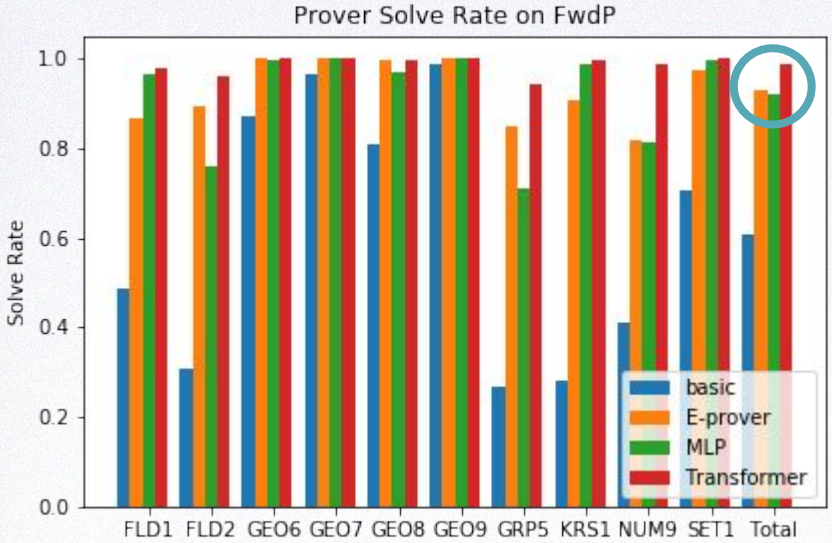
# Results

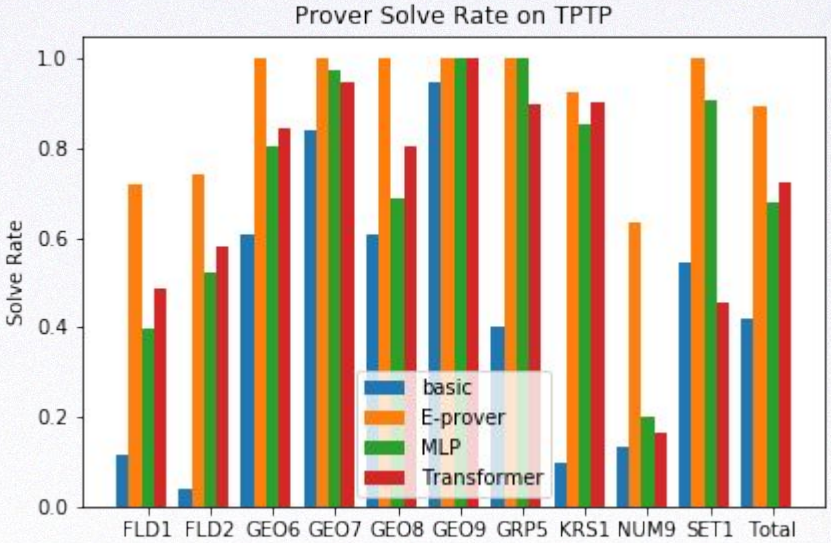



Solidly beating E-prover on FwdP. Definite positive transfer to TPTP, but still a ways to go.

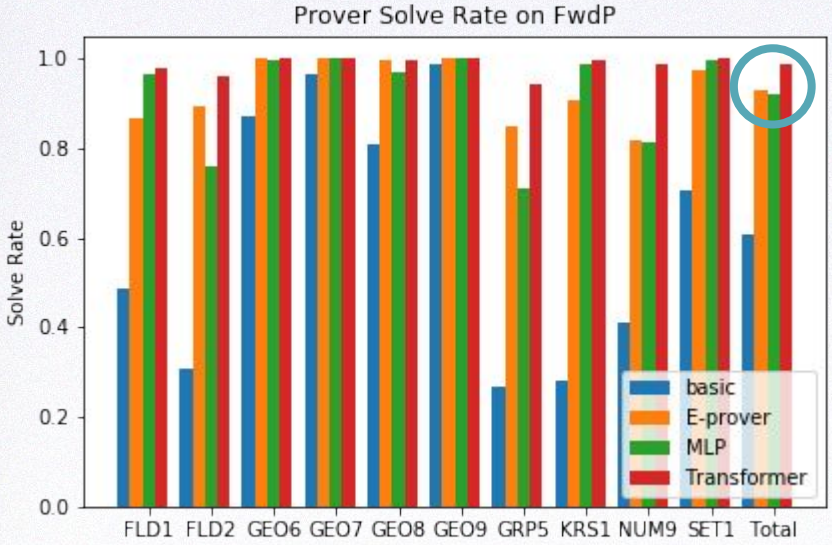
# Results

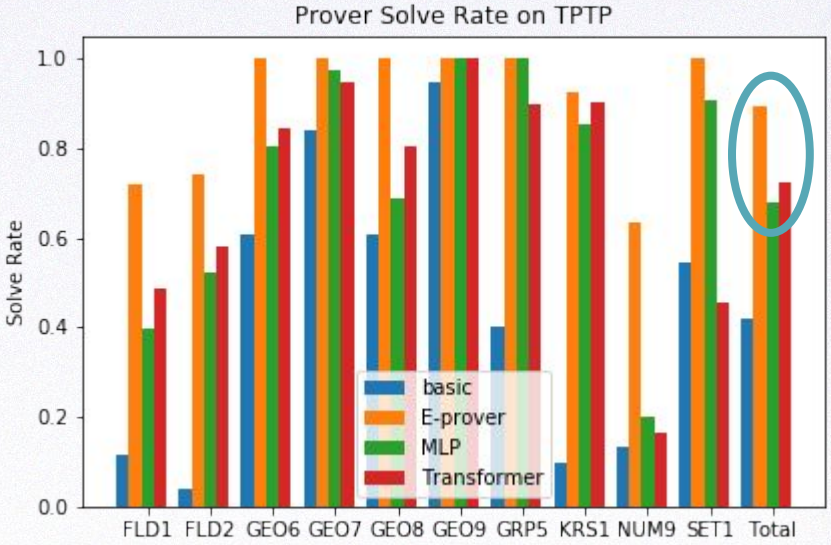



Solidly beating E-prover on FwdP. Definite positive transfer to TPTP, but still a ways to go.

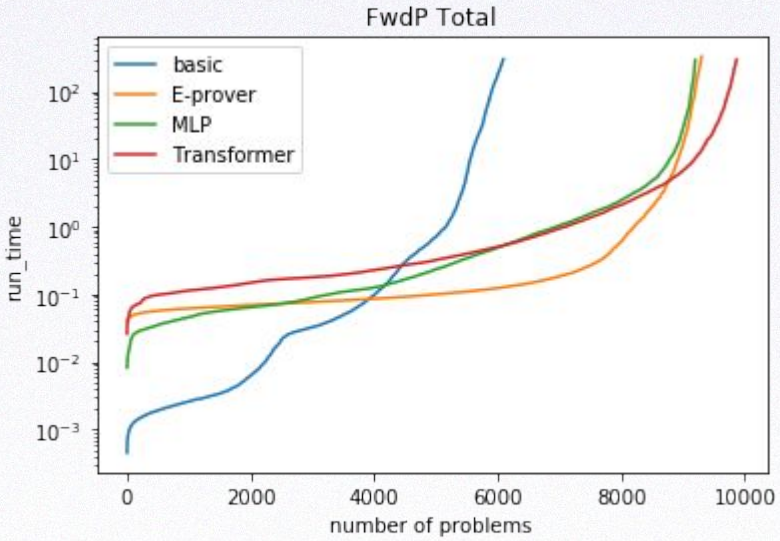
# Results

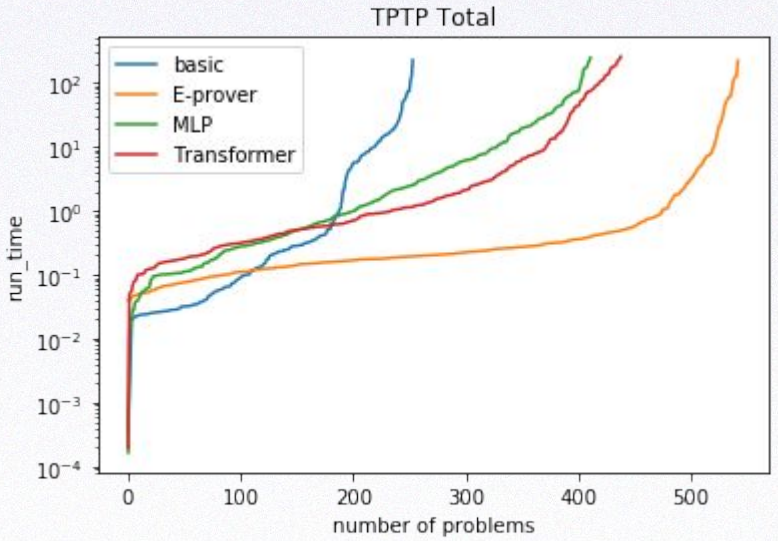



Solidly beating E-prover on FwdP. Definite positive transfer to TPTP, but still a ways to go.

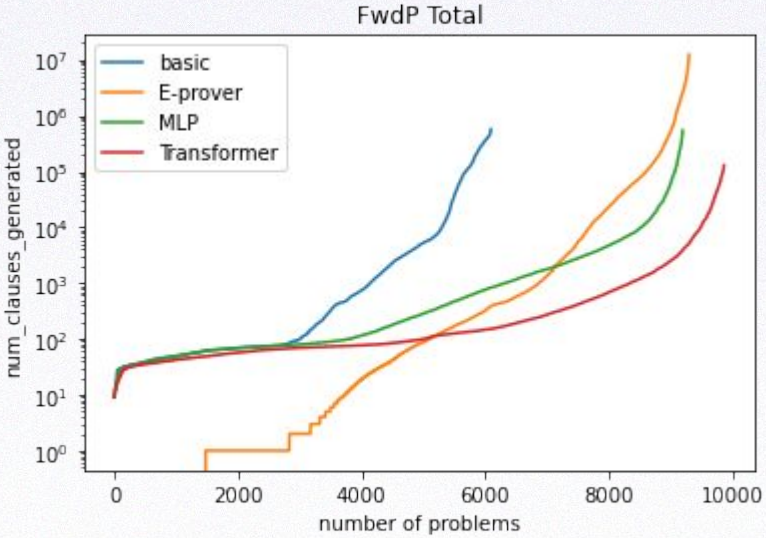
# Search Time Efficiency

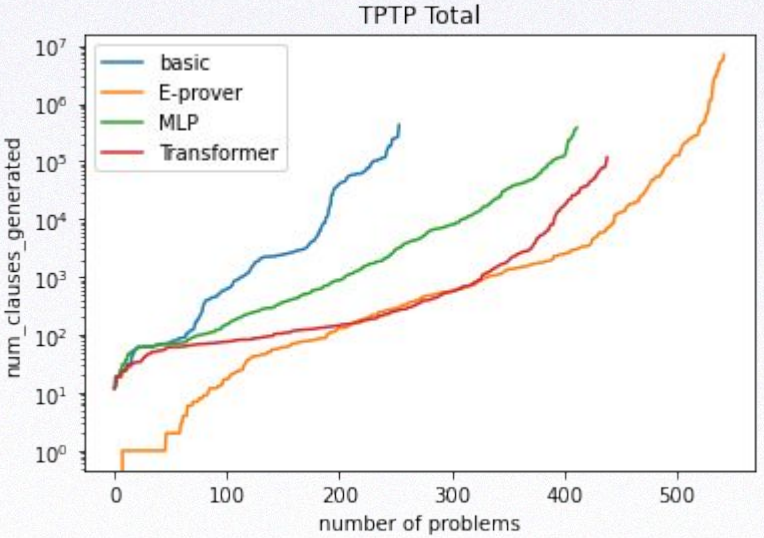



Much slower Transformer still does better on both problem sets.
No overfitting: similar advantage between MLP and Transformer on FwdP vs TPTP.

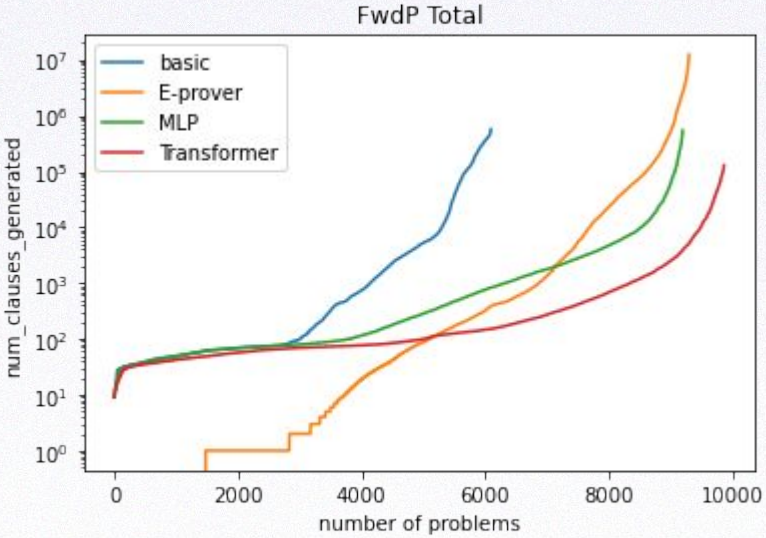
# Search Step Efficiency

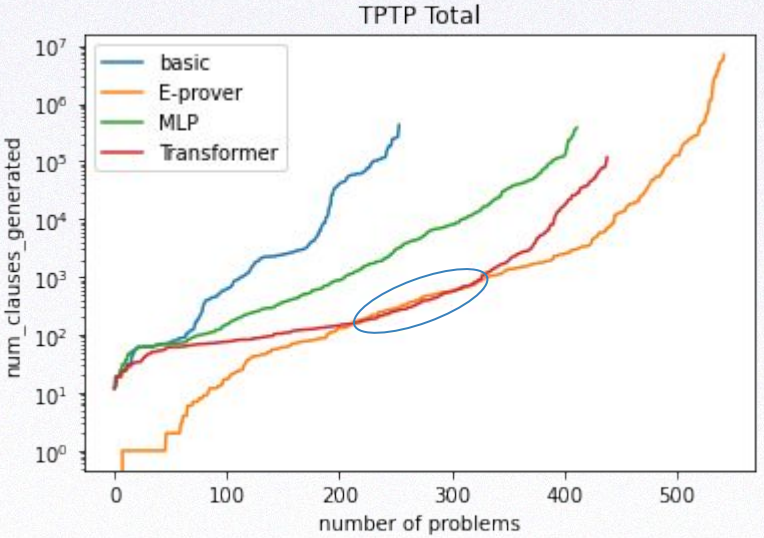



ML does even better when network inference is free.
Shows that the search decisions are being made fairly intelligently.

# Search Step Efficiency




ML does even better when network inference is free.
Shows that the search decisions are being made fairly intelligently.

# Future Work

Clearly we need better synthetic data. How do we get it?

Use ML in the proposer to optimize for… something.

- Small problem size (Occam's Razor).
- Prover difficulty: 2-player game.
- While maintaining diversity.

What is an interesting problem (theorem)?

- A theorem which is useful (as a lemma) for solving other problems.
- Usefulness can be objectively measured and thus optimized for.