Fast Numerical Program Analysis with Reinforcement Learning

Gagandeep Singh

Markus Püschel

Martin Vechev

Department of Computer Science
ETH Zürich

ML-Based Solvers

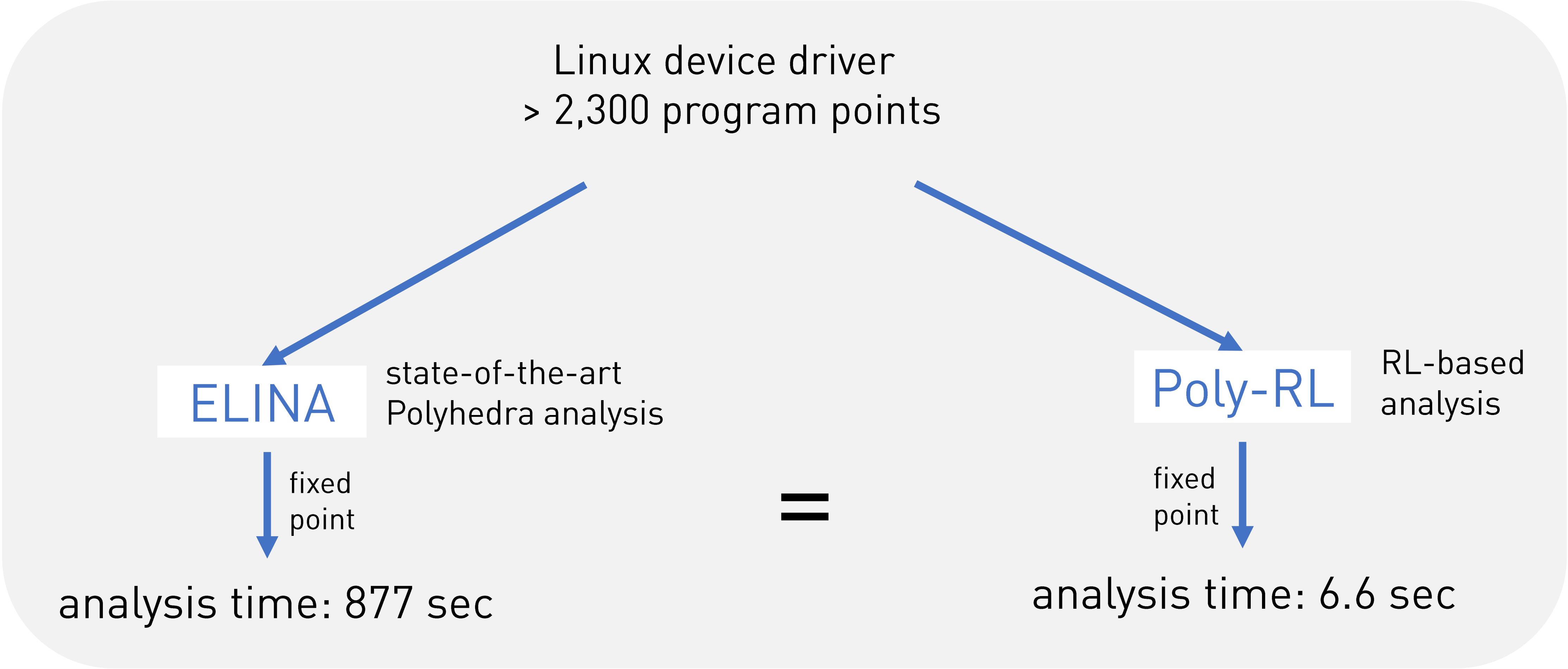
Many solvers and analyzers are based on heuristics

Trade-off precision vs. scalability

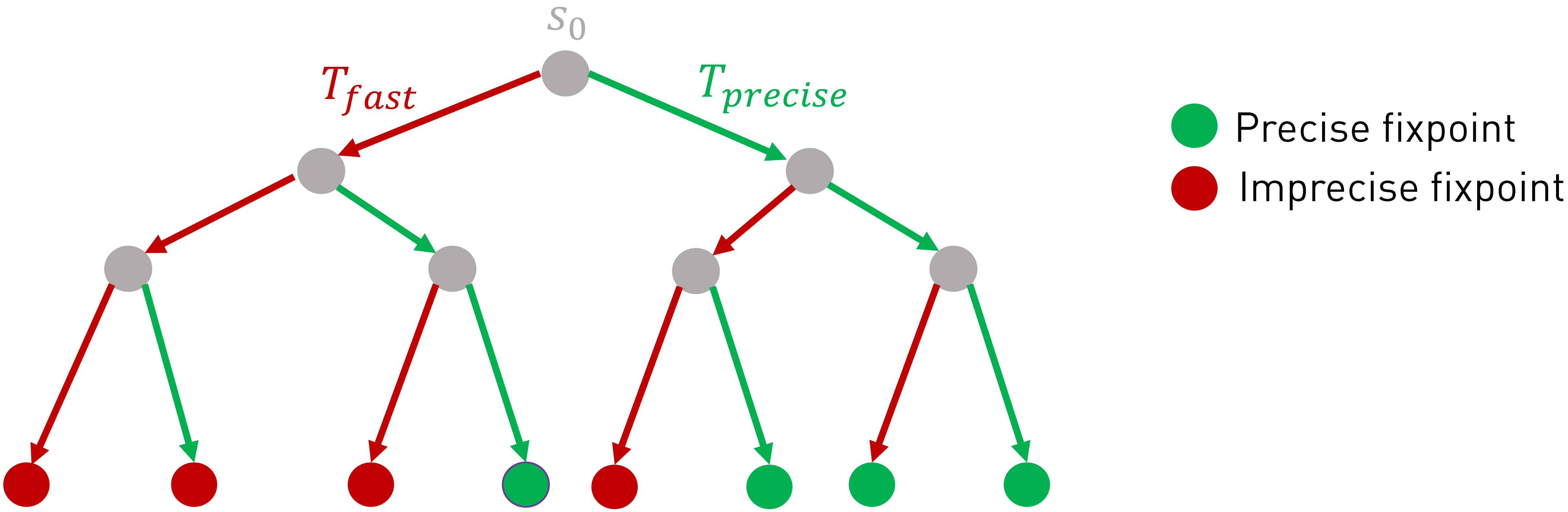
Key idea: apply machine learning to learn optimal strategy

This work: ML for numerical static analysis

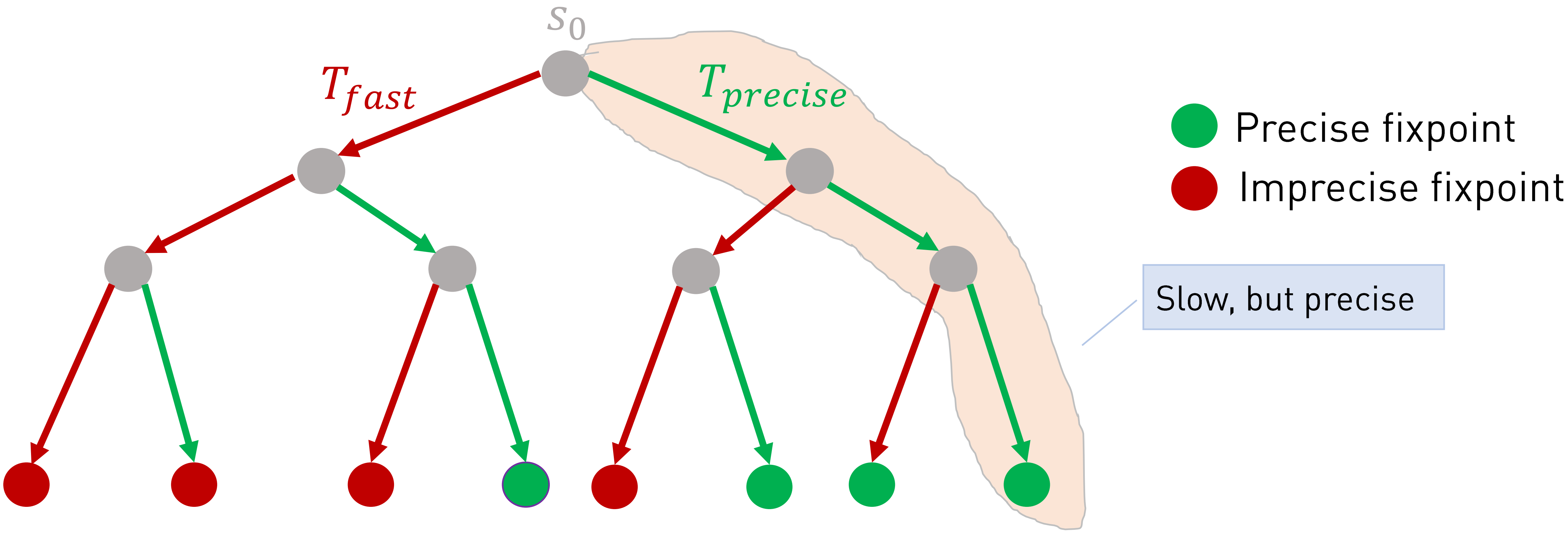
Reinforcement learning for analysis



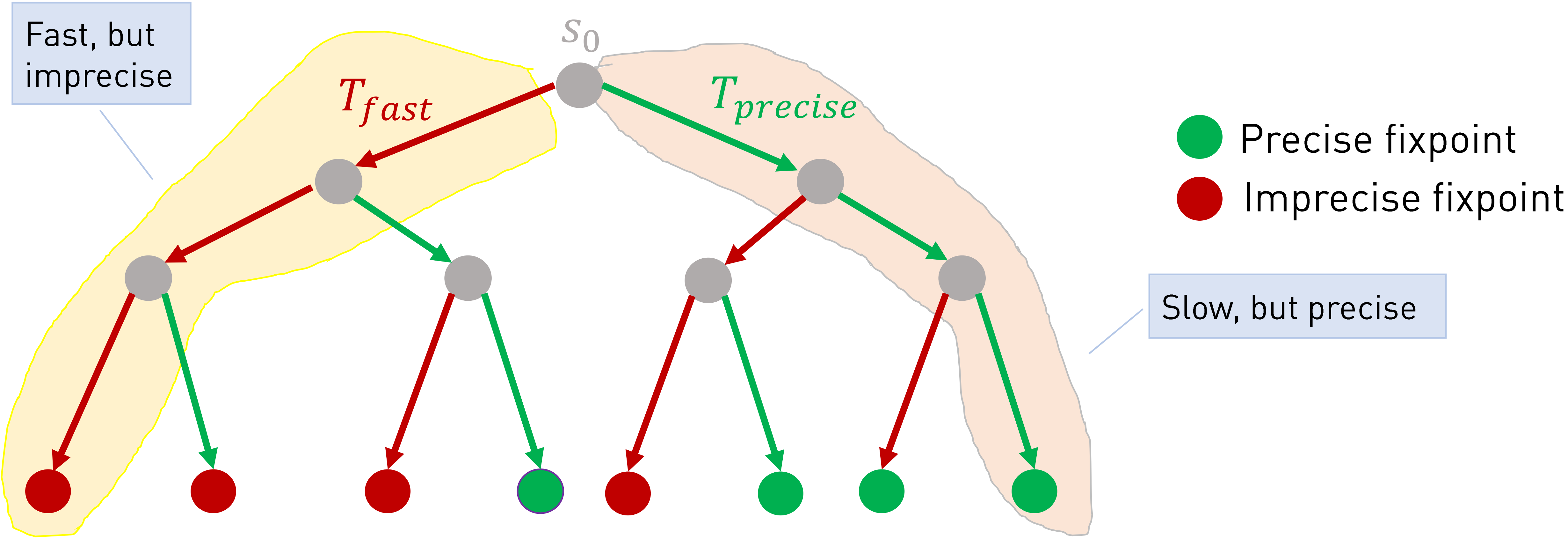
Static analysis: trade-offs



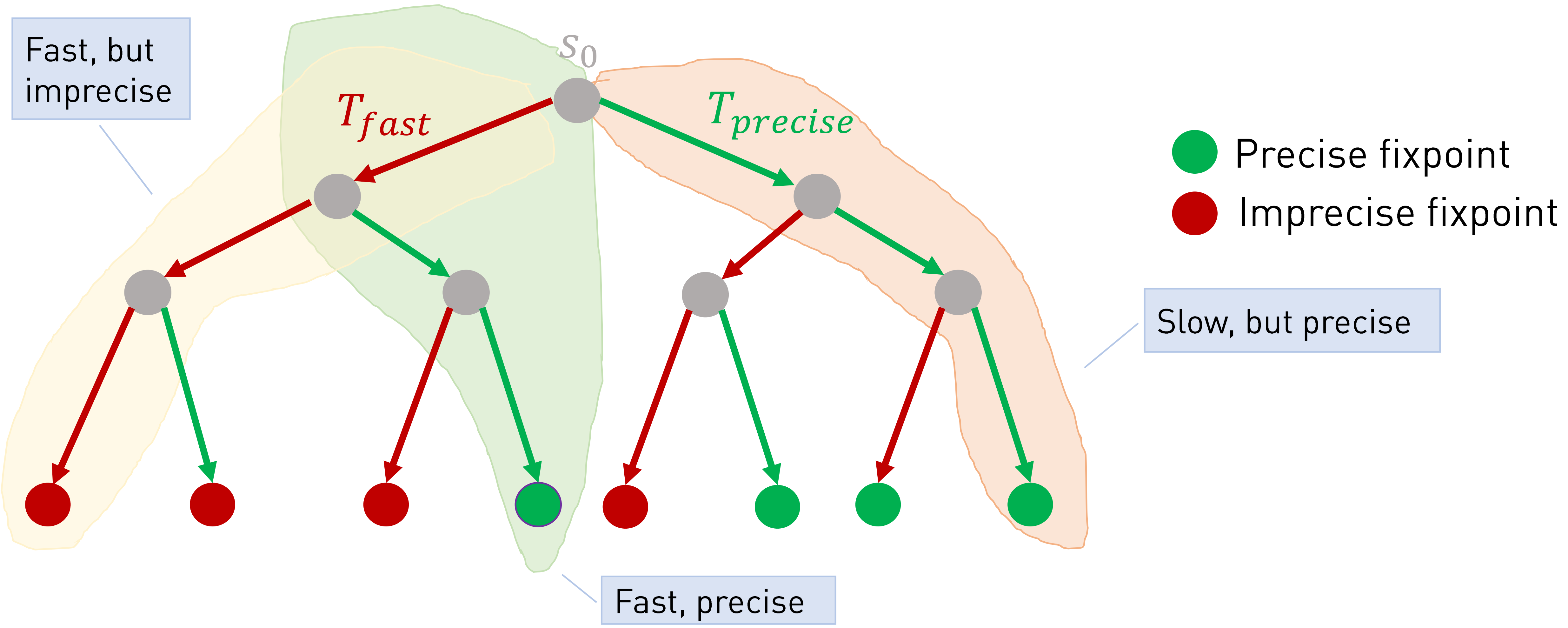
Static analysis: trade-offs



Static analysis: trade-offs

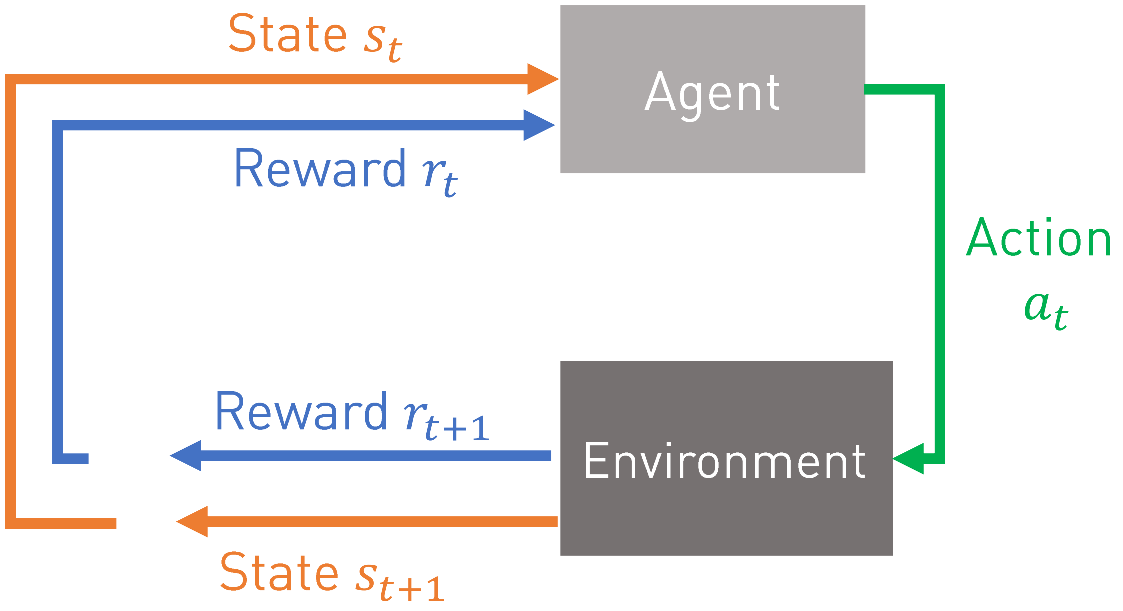


Static analysis: trade-offs



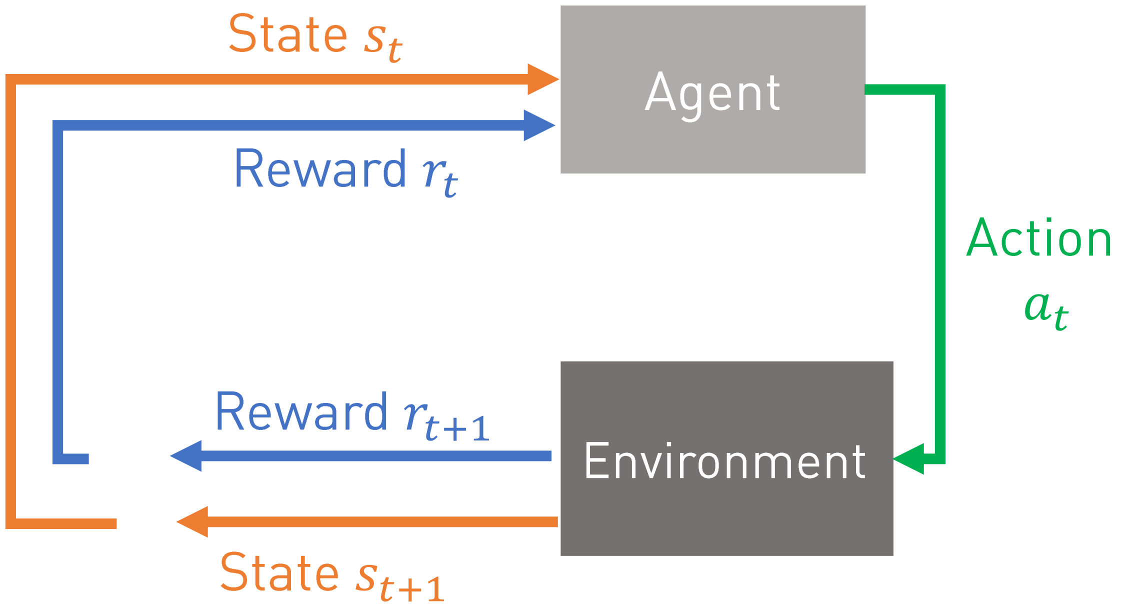
Goal: find such a sequence of transformers

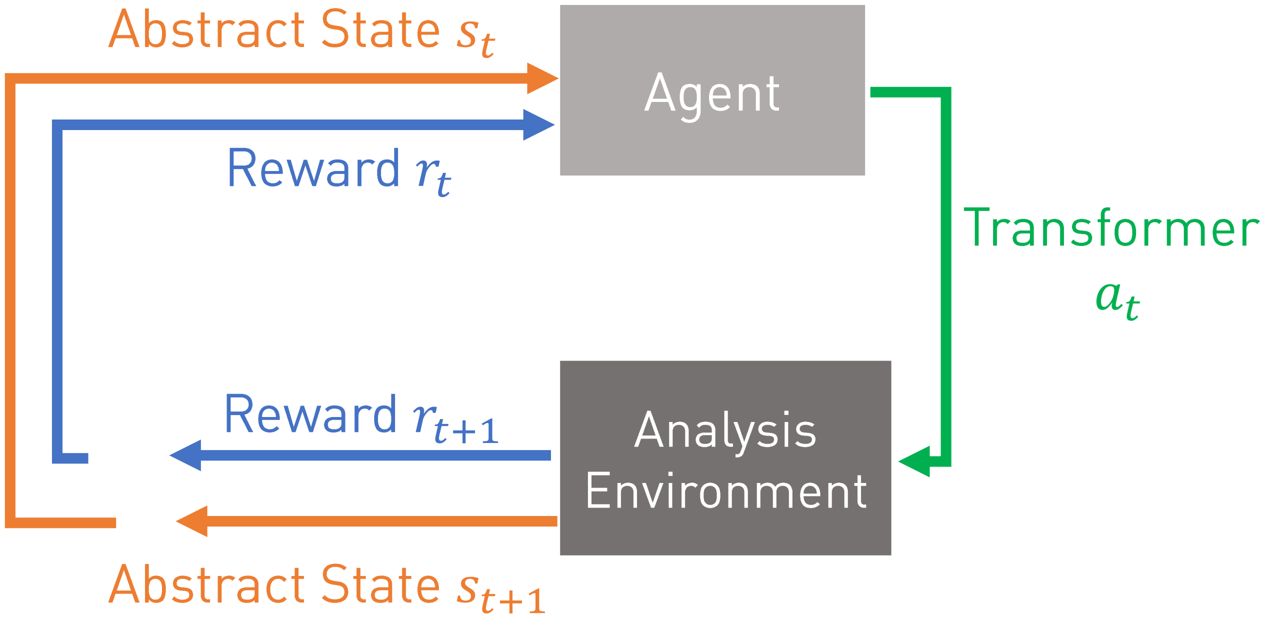
Reinforcement learning



Learn a policy that for any state selects the action maximizing long term rewards

Reinforcement learning for analysis





Learn a policy that for any state selects the action maximizing long term rewards

Learn a policy that for any abstract state selects the transformer maximizing speed and precision of analysis

What is the agent doing?

Agents maintains a function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

higher Q is proxy for higher precision
and performance at fixpoint

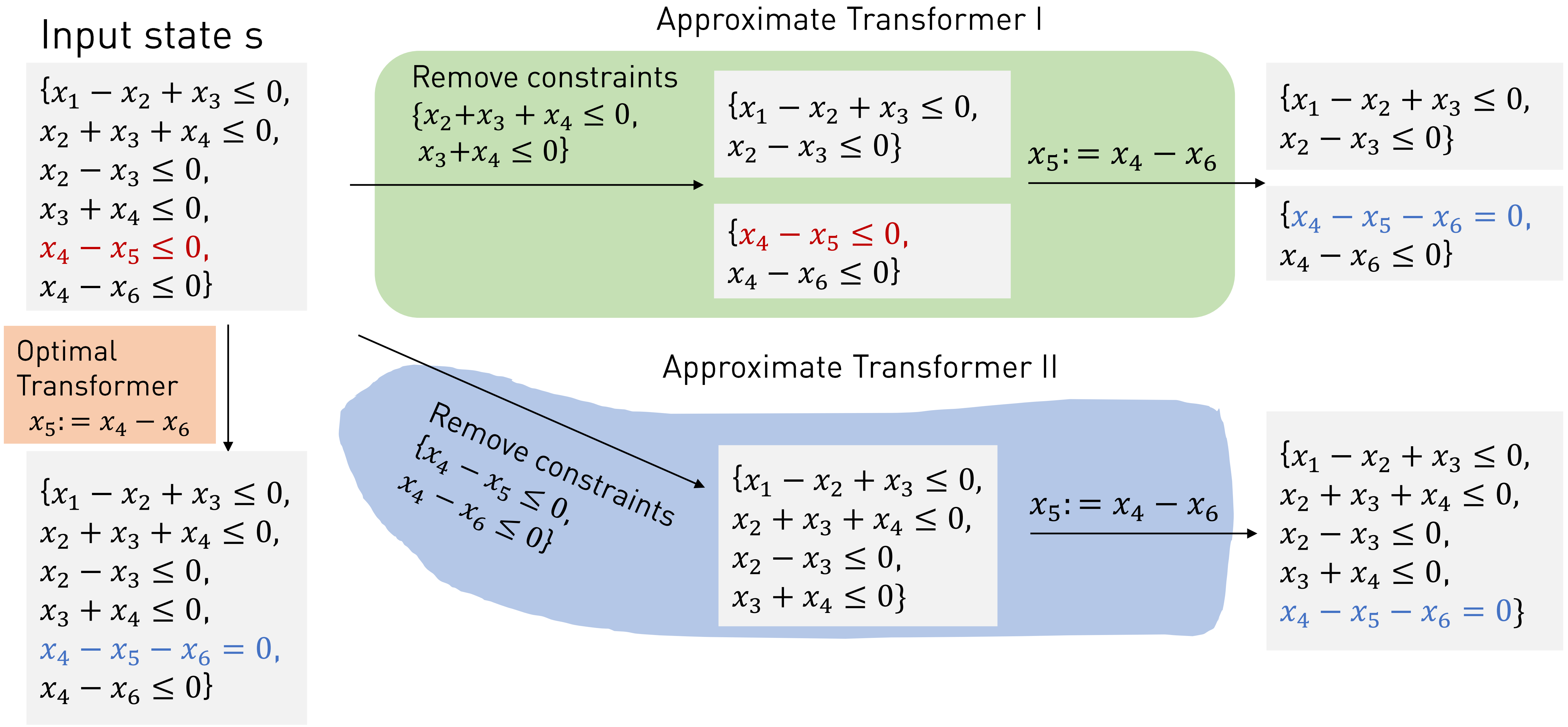
$$Q(s, a) = \sum_{j=1}^{l} \theta_j \, \phi_j(s, a)$$

linear function approximation,
can also use deep learning to represent Q

ϕ_j are features on (state, action) pairs
θ_j are parameters to be learned via Q-learning,
learning uses reward function $r(s_t, a_t, s_{t+1})$

$$a_t = argmax_{a \in \mathcal{A}} Q(s_t, a)$$

Step 1: Define space of transformers $\mathcal{A}$



Step 2: Define features $\phi_j(s, a)$

Feature are proxy for **precision** of input s and **performance** of transformer a

State s

$\{x_1 - x_2 + x_3 \leq 0,$
$x_2 - x_3 \leq 0\}$

$\{x_4 - x_5 \leq 0,$
$x_4 - x_6 \leq 0\}$

$\{x_7 = 0,$
$x_8 + x_7 \leq 0\}$

Precision features	Value
# of variables with finite upper and lower bound	1 (x_7)
# of variables with either finite upper or lower bound	2 (x_1, x_8)

Performance features	Value
# of blocks	3
Maximum # of variables in a block	3
Minimum # of variables in a block	2
Average # of variables in a block	8/3

Step 3: Define reward function $r(s_t, a_t, s_{t+1})$

Reward favors high precision of output state s_{t+1} and speed for transformer a_t

Output state s_{t+1}

$\{x_1 - x_2 + x_3 \leq 0,$
$x_2 - x_3 \leq 0\}$

$\{x_4 - x_5 - x_6 = 0,$
$x_5 = 0,$
$x_6 \leq 2, -x_6 \leq 0\}$

$\boldsymbol{n_s}$: # of variables with singleton bounds

$\boldsymbol{n_b}$: # of variables with finite lower and upper bound

$\boldsymbol{n_{hb}}$: # of variables with only finite lower or upper bound

$\boldsymbol{cyc}$: # of cycles for computing the transformer

$$r(s_t, a_t, s_{t+1}) = 3.\boldsymbol{n_s} + 2.\boldsymbol{n_b} + \boldsymbol{n_{hb}} - log_{10}(\boldsymbol{cyc})$$

Feature	Value on s_{t+1}
n_s	1 (x_5)
n_b	2 (x_4, x_6)
n_{hb}	1 (x_1)
cyc	10
$r(s_t, a_t, s_{t+1})$	7

Putting it all together

Finally, using the features ϕ_j, the reward function $r(s_t, a_t, s_{t+1})$, and transformers $\mathcal{A}$, we can apply Q-learning and learn θ_j

Then, we can perform **RL-based analysis**

$$Q(s, a) = \sum_{j=1}^{l} \theta_j \, \phi_j(s, a)$$

$$a_t = argmax_{a \in \mathcal{A}} Q(s_t, a)$$

Experimental setup

Dataset from SVCOMP

70 benchmarks for training, 30 benchmarks for testing

Poly-RL vs

- ELINA: state-of-the-art Polyhedra library [**ground truth**]
- Poly-Fixed: fixed strategy
- Poly-Init: use precise transformer 50% of the time

Precision = % of program points with same invariants as ELINA

Experimental results

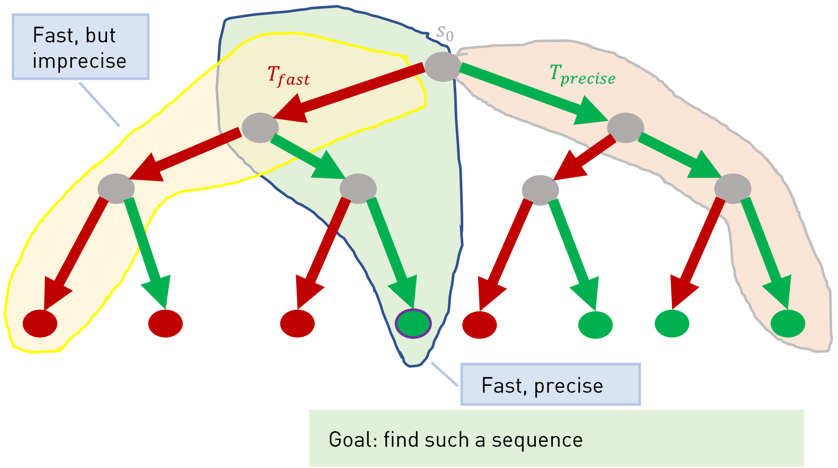
Timeout: 1hr, Memory limit: 100 GB

Benchmark	Number of program points	ELINA	Poly-RL		Poly-Fixed		Poly-Init	
		time (s)	time(s)	precision	time(s)	precision	time(s)	precision
wireless_airo	2372	877	6.6	100	6.7	100	5.2	74
net_ppp	689	2220	9.1	87	TO	34	7.7	55
mfd_sm501	369	1596	3.1	97	1421	97	2	64
ideapad_laptop	461	172	2.9	100	157	100	OOM	41
pata_legacy	262	41	2.8	41	2.5	41	OOM	27
usb_ohci	1520	22	2.9	100	34	100	OOM	50
usb_gadget	1843	66	37	60	35	60	TO	40
wireless_b43	3226	19	13	66	TO	28	83	34
lustre_llite	211	5.7	4.9	98	5.4	98	6.1	54
usb_cx231xx	4752	7.3	3.9	≈100	3.7	≈100	3.9	94
netfilter_ipvs	5238	20	17	≈100	9.8	≈100	11	94

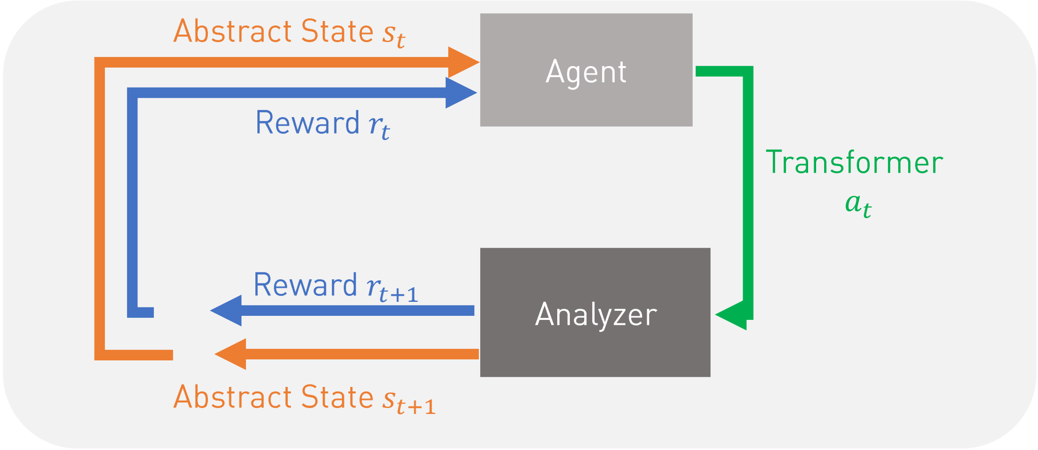
Poly-RL produces the same invariant at assertion points as ELINA on all benchmarks

Summary

Many analyzers/solvers based on heuristics



Use Reinforcement Learning to find heuristics



Instantiate for Polyhedra via Q-learning

Step 1: Define space of transformers $\mathcal{A}$

Step 2: Define features $\phi_j(s, a)$

Step 3: Define reward function $r(s_t, a_t, s_{t+1})$

$$Q(s,a) = \sum_{j=1}^{l} \theta_j \, \phi_j(s,a) \qquad a_t = argmax_{a \in \mathcal{A}} Q(s_t, a)$$

Promising results and future work