PEPR IA

Nathanaël Fijalkow
CNRS
LaBRI, Bordeaux

SAIF

SAFE — IA — FORMAL METHODS

01/10/2023 → 31/09/2027

5M € (LaBRI : 900k€)

Coordinator :
Caterina Urban (INRIA PARIS)

PARTNERS :

INRIA Paris Saclay
IRISA
LMF
CEA LIST

PERSONNEL :

1 PhD student
2 years research engineer
6 years postdoc

SYNTHESIS

INPUT: Specification

OUTPUT: model satisfying the specification

LEARNING

INPUT: data

OUTPUT: model explaining the data

GENERAL APPROACH

inspired by insights from program synthesis,

give a new spin to:

(1) Model learning

(2) Reinforcement learning

(3) Reactive synthesis

INSIGHTS FROM PROGRAM SYNTHESIS

* Neuro- Symbolic approaches are powerful
  - Neuro: efficient
  - Symbolic: reliable

* Programs are a good abstraction for synthesis:
  - explainable, readable, verifiable
  - expressive, succinct
  - searchable, enumerable

preferred communication between humans and machines

PART I: MODEL LEARNING

automata learning
specification mining
⋮
$\subseteq$ model learning

logical formulas

General problem: given traces, construct a generating model

→ explain (bugs, security breaches)

→ use as specification

→ use as intermediate model for verification or reasoning

EXAMPLE: Consider **learning Boolean circuits**:

Variables $x_1, \dots, x_n$ $\quad f : \{0,1\}^n \to \{0,1\}$

The learner can:

Membership query: ask for the value of $f(\bar{x})$ for some concrete $\bar{x}$

Equivalence query: check whether some circuit C computes f

Beimel et al, JACM 2000

IDEA :

$f : \{0,1\}^n \to \{0,1\}$ is a special case of $f : A^* \to \mathbb{R}$

↳ learn f as a weighted automaton !

axe IA : a transformer without activation functions

M2F : automaton with real values

Lemma : if $\mathcal{A}$ is a **minimal** weighted automaton computing $f : \{0,1\}^* \to \mathbb{R}$ such that :

(1) $f(w) \neq 0 \Rightarrow w \in \{0,1\}^n$

(2) $\forall w,\ f(w) \in \{0,1\}$

in other words : $f : \{0,1\}^n \to \{0,1\}$

Then $\mathcal{A}$ is $\simeq$ a Boolean circuit

(also true for polynomials, Boolean automata, ...)

WEIGHTED AUTOMATA LEARNING

existing results:

elegant combination of linear algebra and algorithms

OPEN QUESTION

can we learn probabilistic automata?

Example (Xavier Hinaut):

can we generate (= understand the structure of) bird songs?

Research directions:

- symbolic algorithms
- neuro-symbolic algorithms
- most importantly: representation for probabilistic distributions as probabilistic programs ?

SPECIFICATION MINING

very hard to write complete specification

idea: allow users to give examples of good / bad behaviours

OPEN QUESTION

can we learn SMALL formulas from positive traces?

(LTL / regular expressions)

very active in robotics / cyber physical systems

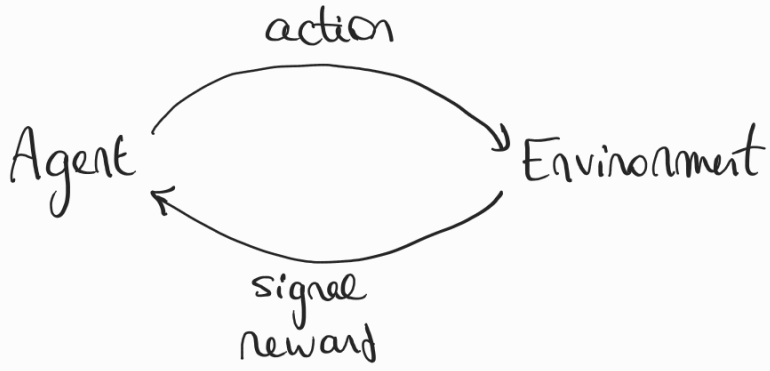
WIDE OPEN:

- no good theoretical definition!
- existing tools show limited results, purely symbolic

Research directions:

- neuro-symbolic algorithms
- GPU implementations are very powerful for formulas enumeration!

PART II REINFORCEMENT LEARNING

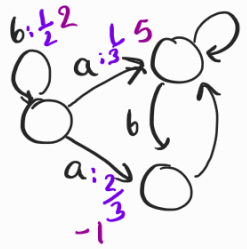


Goal: construct a policy for the agent to maximise rewards

Two classical settings:

(1) Tabular

- environment is an Markov Decision Process (MDP)



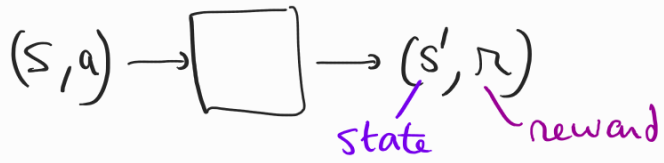
probabilities
rewards

- policy is $\sigma : S \to A$ (state → action)

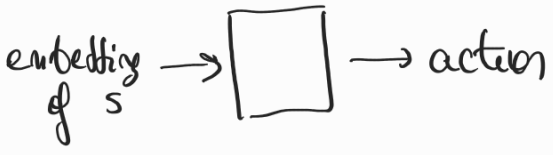
ISSUE when S is large, σ is unmanageable

(2) Deep RL

- environment is complicated but can be simulated:



- policy is represented by a neural network



ISSUE neural networks !

Experiment on MountainCar using DeepSynth (program synthesis)

In ~20s we found: (input variables (position, velocity))

```
If ( sign(Var1) ≥ Var0) :
    left
Else :
    Right
```

(actions)

a program

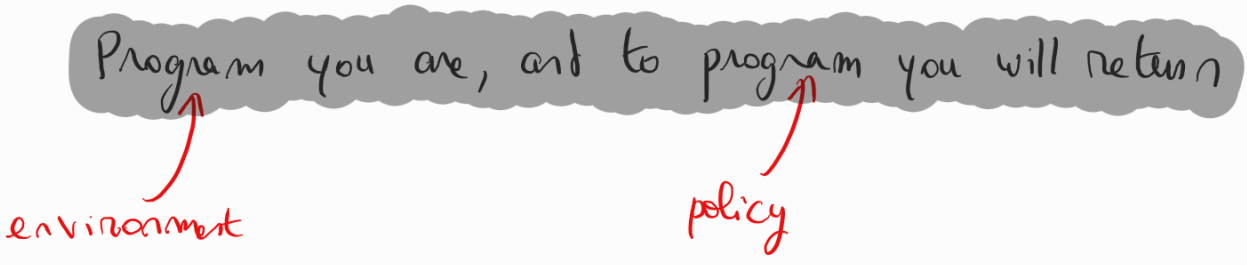
Baseline Deep learning (DQN): 5 mn for a neural policy ...

The third setting I want to explore:

(3) Programmatic RL

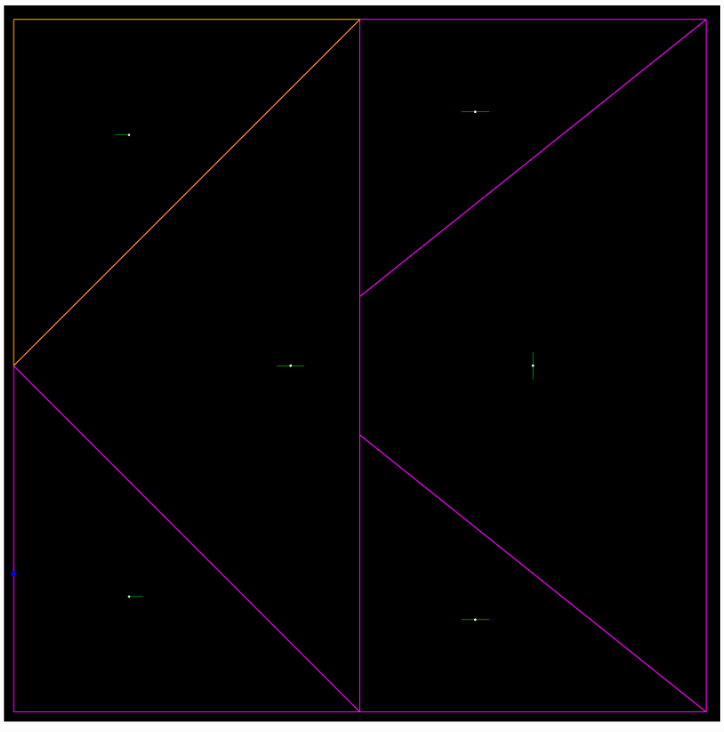
- environment given by a program which describes the dynamics
- policy represented by a program

Motto:

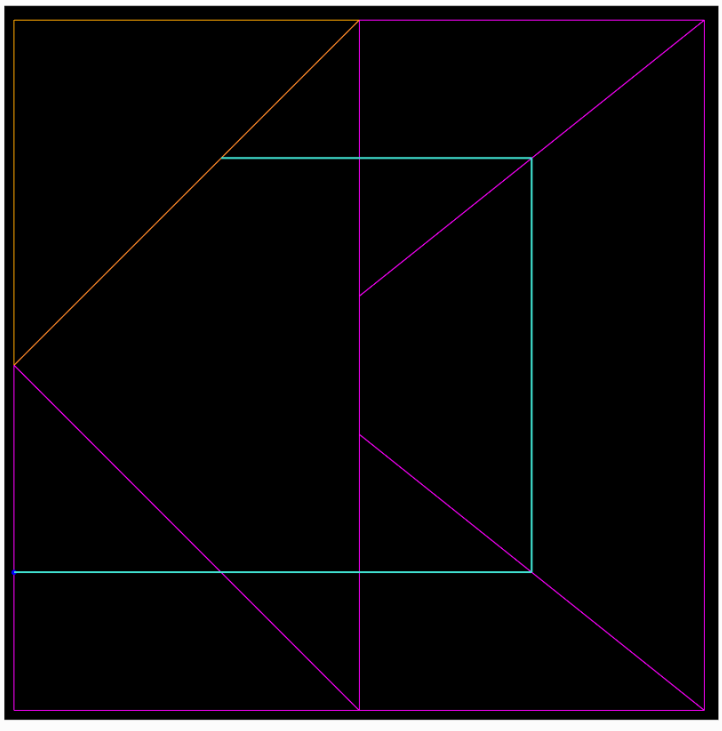


GRIDWORLDS (Guruprerana Shabadi)

problem

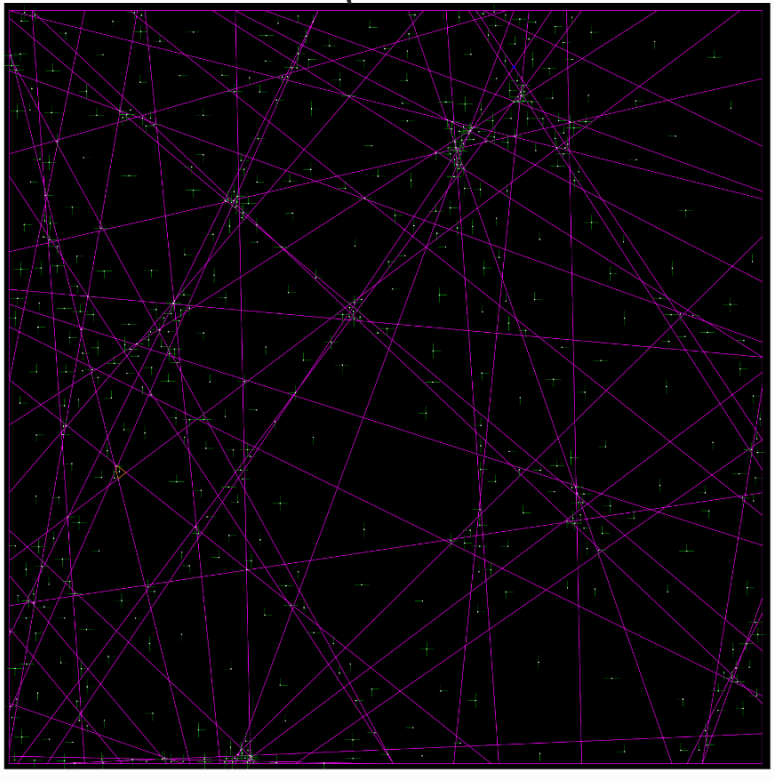


Solution

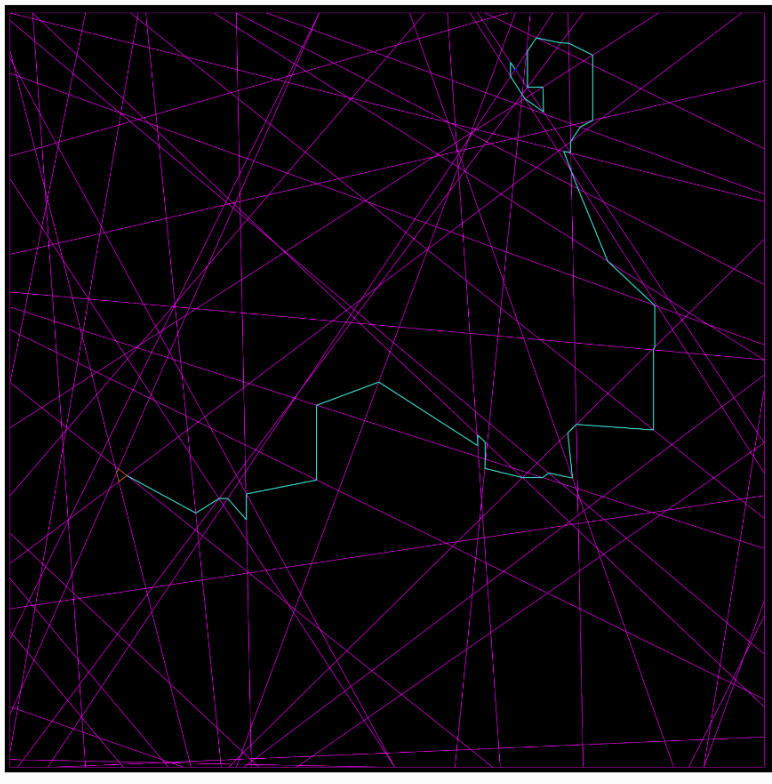


Key feature: the environment is given as a PRBT program

problem

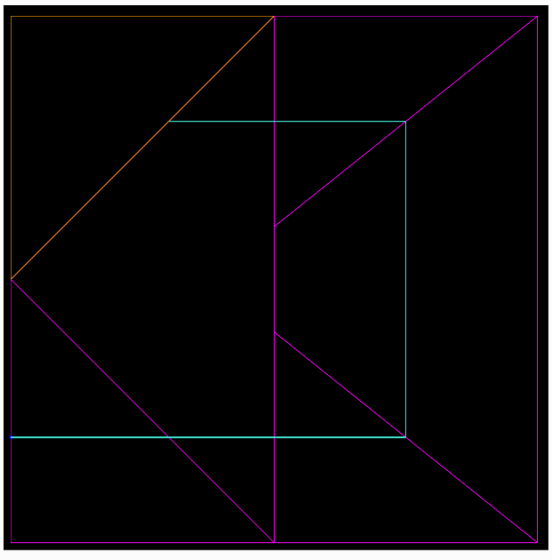


Solution



Program representation inspired by Shannon

Example



generated policy

```
Until([(0, 0), (0, 25)]):
  [(0, 25), (0, 0)] ->
    [(5, 20), (25, 0)], Preference: Neutral, Edge: [(0, 25), (25, 0)]

Until([(0, 25), (25, 0)]):
  [(0, 25), (25, 0)] ->
    [(25, 0), (25, 20)], Preference: Neutral, Edge: [(25, 20), (25, 0)]

Until([(25, 0), (25, 20)]):
  [(25, 20), (25, 0)] ->
    [(25, 20), (50, 0)], Preference: Neutral, Edge: [(25, 20), (50, 0)]

Until([(25, 20), (50, 0)]):
  [(25, 20), (50, 0)] ->
    [(25, 30), (50, 50)], Preference: Neutral, Edge: [(25, 30), (50, 50)]

Until([(25, 30), (50, 50)]):
  [(25, 30), (50, 50)] ->
    [(25, 30), (25, 50)], Preference: Neutral, Edge: [(25, 30), (25, 50)]

Until([(25, 30), (25, 50)]):
  [(25, 30), (25, 50)] ->
    [(0, 25), (25, 50)], Preference: Neutral, Edge: [(0, 25), (25, 50)]

Until([(0, 25), (25, 50)]):

  Win!
```

Summary of experimental results:

the program as policies are infinitely more succinct !

and Verifiable, explainable, readable, executable, ...

- environment as program : SMALL
- environment as explicit MDP : LARGE
- policy as program : SMALL
- policy as explicit $S \to A$: LARGE

Programmatic RL is mostly unexplored !

GOALS

(1) A Gym-like framework for Programmatic RL (collecting benchmarks, baseline approaches)

(2) Clear theoretical framework to show that optimal policies resemble environments

(3) Neuro-symbolic algorithms

PART III REACTIVE SYNTHESIS

Boolean circuit synthesis already back in 1930's by Church !

INPUTS: i_1 i_2 i_3 ... I inputs

OUTPUTS: o_1 o_2 o_3 ... O outputs

Specification: (LTL) formula over $(I \times O)^\omega$

Reactive system: $S : I^+ \to O$

Example ARBITER

n users

input : at each time step, a subset of users can request access to some resource

ouput : at each time step, the arbiter can grant access to at most one user

Specification :

"every request is granted eventually"

$\wedge$ "no user is granted access without request"

$\wedge$ ⋮

Symbolic Solution :

$S : (I \times O)^* \to O$

typically represented as **finite state machine** (large, hard to interpret)

Issue : the canonical queue solution grows exponentially with number of users !

Change representation ! As a program :

update the data structure

```
def arbiter(X):          # set of requests received at current time step
    for x in X:
        if not memQueue(x):
            addQueue(x)
```

Choose output

```
if not emptyQueue():
    popQueue()
```

Short simple parametric

Programmatic RS is mostly unexplored !

GOALS

(1) Design a reactive programming language (ex: Verilog, ...) and show completeness

(2) Neuro-symbolic algorithms

↑ recently neural approaches crushed the SYNTCOMP benchmarks

TAKE-AWAY MESSAGE / RESEARCH COMPASS

* Neuro-Symbolic approaches are powerful
  - Neuro: efficient
  - Symbolic: reliable

* Programs are a good abstraction for synthesis:
  - explainable, readable, verifiable
  - expressive, succinct
  - searchable, enumerable

preferred communication between humans and machines