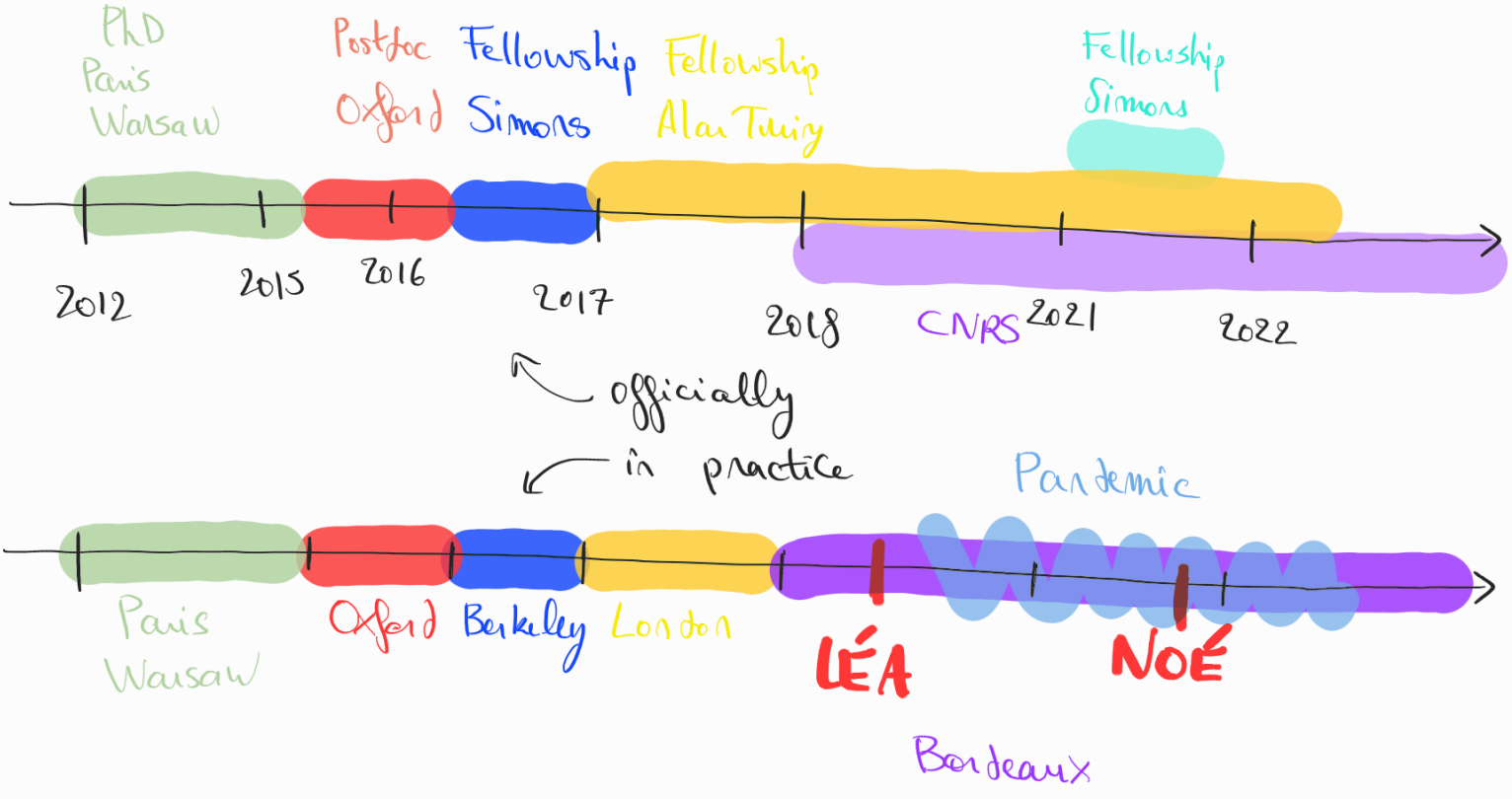
HABILITATION Á DIRIGER DES RECHERCHES

LaBRI 11/02/2022

Nathanaël Fijalkow
CNRS
The Alan Turing Institute



The Game of Synthesis

PROGRAM SYNTHESIS

INPUT: Specification

OUTPUT: program satisfying the specification

Two examples:

(1) **Reactive Synthesis** :

I set of inputs
O set of outputs

$S: I^+ \to O$

Inputs i_1 i_2 i_3 ...
Outputs o_1 o_2 o_3 ...

Logical specification:
φ over $(I \times O)^\omega$

(2) Inductive Synthesis :

Specification : a few examples (input, output)

Ex: $[1,5,4,2] \longrightarrow [2,4]$

$[6,3,0,8] \longrightarrow [0,6,8]$

Solution: SORT(FILTER(even))

What can a program represent ?

Hardware
Controller
Algorithm
Macro
Strategy
Software
Graphics

Invariant
Specification
Proof
Explanation
Model
Equation
Predictor

IN THIS TALK:

I'll tell you where I want to go
while referring to what I've learnt

FIRST LESSON LEARNT:

REPRESENTATION IS EVERYTHING !

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

IDEA: ← Beimel et al, JACM 2000

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

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

↳ learn f as a weighted automaton !

↑ automaton with real values

Learning weighted automata:

Hankel matrix ↓

elegant combination of linear algebra and algorithms

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, ...)

So: to understand something, it may be useful to change its representation

Two examples:

(1) Clark & F. 2020 (TACL): learning PCFGs

(2) F., Lagarde, Ohlmann, Serre | 2020 (STACS)
2021 (Complexity)

Lower bounds on arithmetic circuits

Strong learning of PCFGs:

Input: words sampled from a distribution $\mathcal{D}$

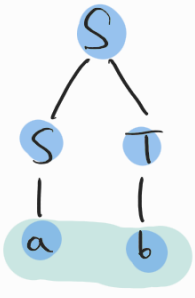
Objective: Construct a PCFG G computing $\mathcal{D}$

Ex:

$$\begin{cases} S \xrightarrow{1/2} ST \\ S \xrightarrow{1/2} a \\ T \xrightarrow{1} b \end{cases}$$

defines a distribution over words

generates



with probability 1/4

1/8 ...

Key idea: PCFG can be re-parameterised as "bottom-up" processes which are easier to learn

Lower bounds on arithmetic circuits

Theorem (Nisan '91):

Let P polynomial. Define N_P "Nisan matrix"

The smallest algebraic branching program computing P has size $\text{rank}(N_P)$

↑ some class of arithmetic circuits

This has been extended to many classes of circuits

Theorem (Fliess '71):

Let $f : A^* \to \mathbb{R}$. Define H_f "Hankel matrix"

The smallest weighted automaton computing f has size $\text{rank}(H_f)$

A foundational result in automata theory

for minimisation, equivalence, learning, ...

Starting point of our work:

Nisan's theorem is a special case of Fliess' !

But this is for linear classes of circuits

↳ ≃ words

↳ extensions to trees: we embarked on a long journey revisiting arithmetic circuit complexity using Hankel's matrix

SO: REPRESENTATION MATTERS!

I would like to explore grammar-based program representations

↑ classical in programming languages

Examples: (Finally some games!)

(1) Reinforcement Learning

(2) Reactive Synthesis

(3) Specification Mining (not discussed today)

Reinforcement Learning

See Video

Typical Deep learning approach: data hungry

the controller is represented by a neural network

hard to interpret / verify

We (Theo Matricon) experimented with our program synthesis tool DEEPSYNTH

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

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

actions

a program generated from a grammar

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

Reactive Synthesis

I inputs

O outputs

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

INPUTS : i_1 i_2 i_3 ...

OUTPUTS : o_1 o_2 o_3 ...

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

Example ARBITER : grants access to users

Specification : "every request is granted eventually"

Symbolic Solution :

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

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

But as a program : short and simple

update the data structure

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

Choose

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

output

popQueue()

SECOND LESSON:

Machine Learning
efficient

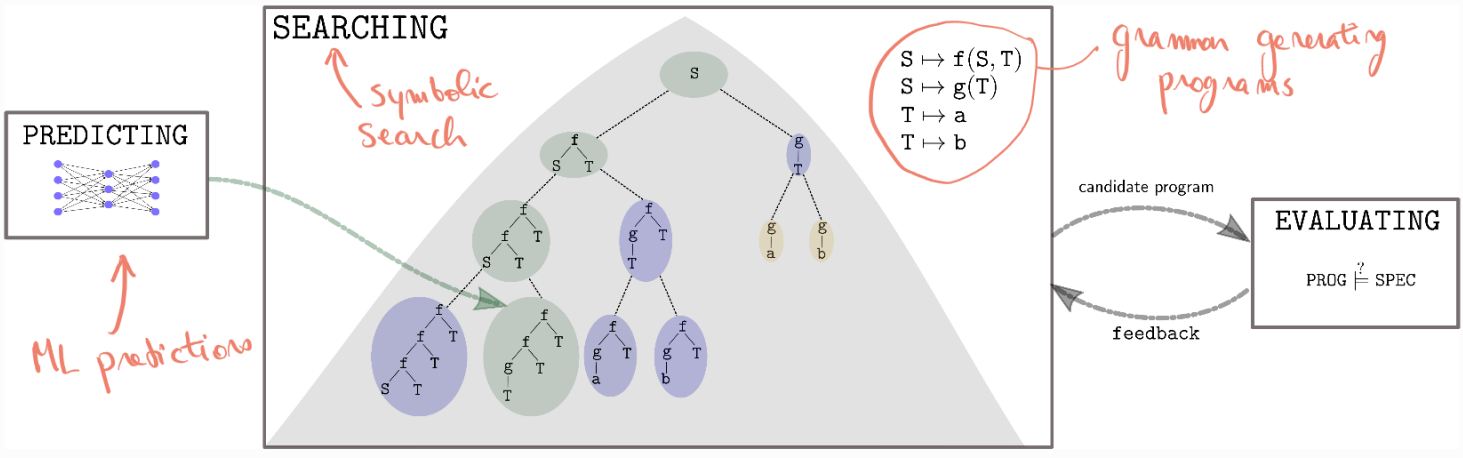
the future is HERE !

Formal Methods
reliable

(1) FM → ML : improving ML's guarantees using FM

(2) ML → FM : improving FM's performance using ML

Predictions-guided program synthesis



First practical instance: DeepCoder for inductive synthesis, 2017

Key idea patterns found in examples reveal which primitives

are used in a solution program

Example: $[1,5,4,2] \rightarrow [2,4]$

$[6,3,0,8] \rightarrow [0,6,8]$

likely primitives: SORT, EVEN, ...

Benefit: predictions affect performance NOT correctness

Rationale: predictions are the best scaling path to address combinatorial explosion

A lot of questions remain open:

- How to make good predictions ?

  → Clymo et al (AI&STAT 20): good data generation improves predictions

- How to deploy probabilistic models ?

  → F. et al (AAAI 22): new symbolic algorithms for grammar enumeration

...

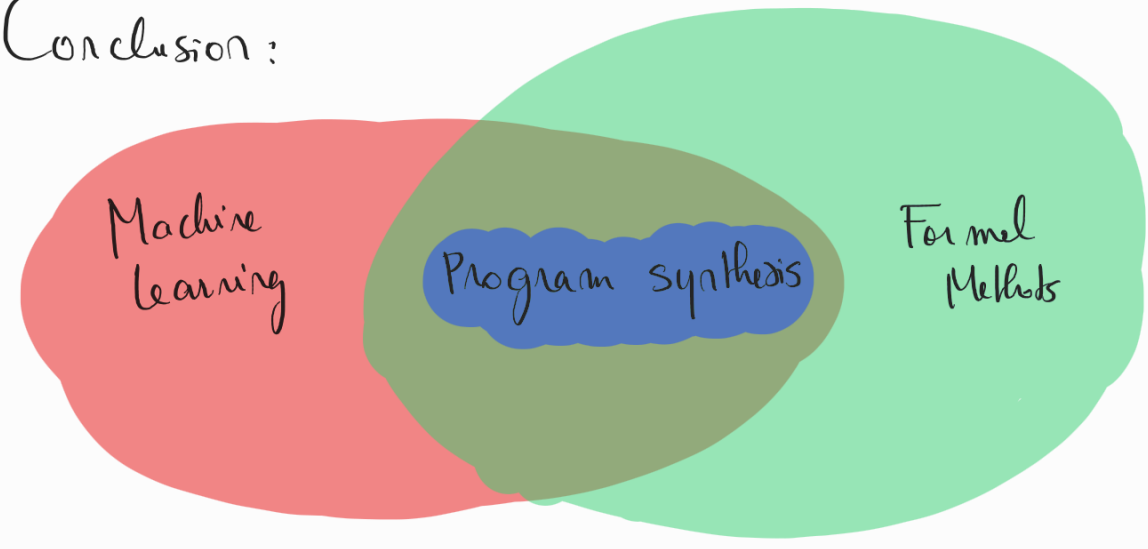
It's almost over ...

I haven't told you about:

- parity games
- invariants in linear dynamical systems
- LTL learning
- controlling populations

...

Conclusion:



Goal: extending the scope of program synthesis to other domains and develop methods at the intersection of ML and FM