Learning programs with magic values

Céline Hocquette and Andrew Cropper

Departement of Computer Science
University of Oxford
celine.hocquette@cs.ox.ac.uk

1 - Introduction

A *magic value* is a constant symbol in a program which has no clear explanation for its choice: it 'magically' works.

```
f(List)←head(List,7)
f(List)←tail(List,Tail),f(Tail)

nextval(5)←does(player,press)
nextval(V)←does(player,noop),
           true_val(T), succ(V,T)

drug(D) ← atom(D,A1),atom(D,A2),
          atomtype(A1,o),atomtype(A2,h),
          distance(A1,A2,0.53)
```

Fig. 1: Example of programs with magic values.

Existing program synthesis approaches rely on enumeration of candidate magic values and thus cannot scale to large or infinite domains.

We introduce an approach, implemented in MAGICPOPPER, which can:

1. learn programs with magic values,
2. improve learning performance,
3. scale to large, even infinite, domains.

Existing approaches	MAGICPOPPER
f(L)←head(L,H),c_1(H)	
f(L)←head(L,H),c_2(H)	f(L)←head(L,H),
f(L)←head(L,H),c_3(H)	@magic(H)
...	

Fig. 2: Existing approaches enumerate possible constant symbols while our approach does not.

3 - Experiment 1

Q1 How well does MAGICPOPPER perform compared to other approaches?

Task	ALEPH	POPPER	MAGICPOPPER
md	**100**	**100**	**100**
buttons	81	**100**	**100**
coins	50	**100**	**100**
buttons-g	**100**	98	**100**
coins-g	50	**100**	**100**
krk	**100**	96	99
list	50	49	**100**
powerof2	83	**100**	**100**
append	95	**96**	**96**

Table 1: Predictive accuracies[1].

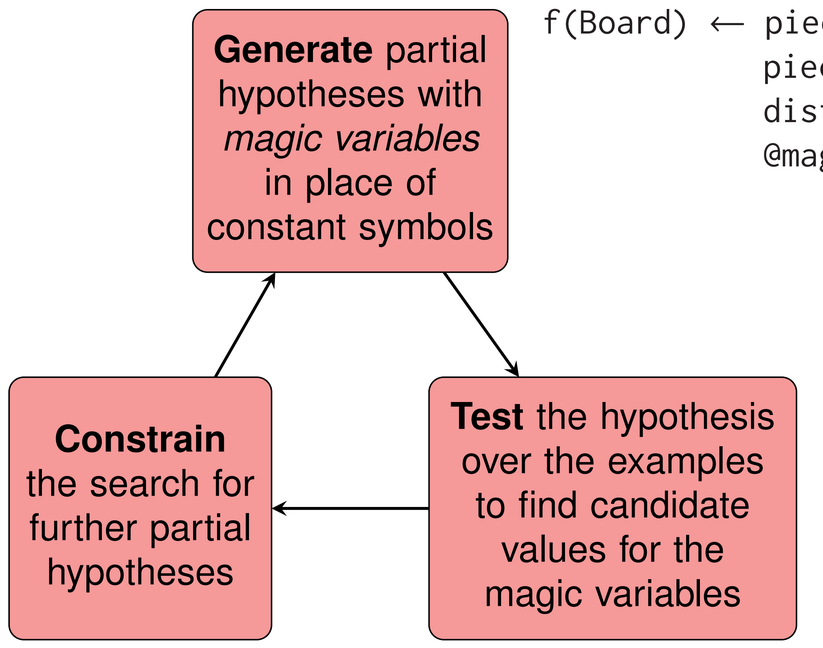
Task	ALEPH	POPPER	MAGICPOPPER
md	**0**	1	**0**
buttons	32	**3**	4
coins	timeout	**53**	99
buttons-g	**0**	1	**0**
coins-g	**0**	**0**	**0**
krk	**0**	35	6
list	66	timeout	**2**
powerof2	**0**	182	**0**
append	1	298	**0**

Table 2: Learning times[1].

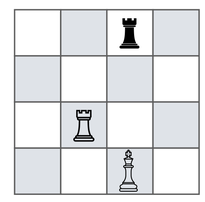
- **MAGICPOPPER can outperform existing approaches.**

2 - Learning framework

Key idea: do not enumerate every possible constant symbols, but instead use *magic variables* which represent the possible constant symbols. Inspired by ALEPH's lazy evaluation procedure [5].




```
f(Board) ← piece(Board,Piece1,Color1,Type1),@magic(Type1),
           piece(Board,Piece2,Color2,Type2),@magic(Type2),
           distance(Piece1,Piece2,Int),@magic(Int)
           @magic(Color1),@magic(Color2)
```



```
f(Board) ← piece(Board,Piece1,white,rook),
           piece(Board,Piece2,white,king),
           distance(Piece1,Piece2,1)
f(Board) ← piece(Board,Piece1,white,king),
           piece(Board,Piece2,black,king),
           distance(Piece1,Piece2,3)
...
```

Fig. 3: Our learning framework follows a generate, test and constrain loop (Learning From Failures [2] setting of ILP [1]).

4 - Experiment 2

Q2 How well does MAGICPOPPER scale?

Fig. 4: List: learning times.

Fig. 5: List: predictive accuracies.

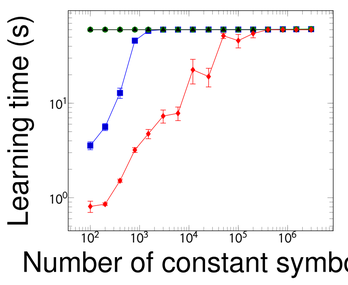


Fig. 6: Md: learning times.

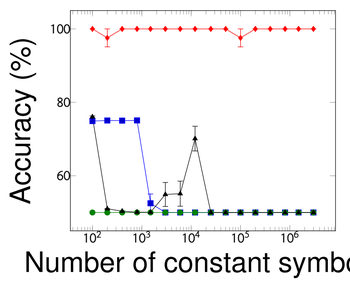


Fig. 7: Md: predictive accuracies.

- **MAGICPOPPER can scale well with the number of constant symbols, up to millions.**

5 - Experiment 3

Q3 Can MAGICPOPPER learn in infinite domains?

Task	ALEPH	POPPER	MAGICPOPPER
pi	**100**	50	99
physics	**100**	62	86
drug	63	50	**98**
next	50	49	**100**
sumk	50	50	**100**

Table 3: Predictive accuracies[1].

Task	ALEPH	POPPER	MAGICPOPPER
pi	4	timeout	**1**
physics	**0**	209	72
drug	5	**1**	6
next	**0**	1	25
sumk	**0**	**0**	99

Table 4: Learning times[1].

- **MAGICPOPPER can learn programs with magic values from infinite domains.**

6 - Conclusion

- **Learning programs with magic values from large, potentially infinite domains.**
- **MAGICPOPPER can outperform existing approaches.**

Future work and Limitations:

- numerical reasoning from multiple examples (eg learning thresholds)
- learning from noisy examples

[1] Results are statistically significant.

References

[1] A. Cropper and A. Dumančić. Inductive logic programming at 30: A new introduction. *J. Artif. Intell. Res.*, 2022.

[2] A. Cropper and R. Morel. Learning programs by learning from failures. *Mach. Learn.*, 2021.

[3] S. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Mach. Learn.*, 2015.

[4] A. Srinivasan. The ALEPH manual. *Mach. Learn. at the Computing Laboratory*, 2001.

[5] A. Srinivasan and R. Camacho. Numerical reasoning with an ILP system capable of lazy evaluation and customised search. *J. Logic Prog.*, 1999.

Article

Code