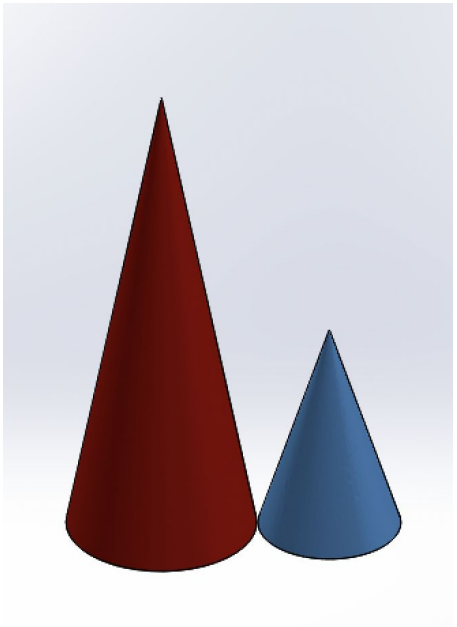
Relational Program Synthesis with Numerical Reasoning

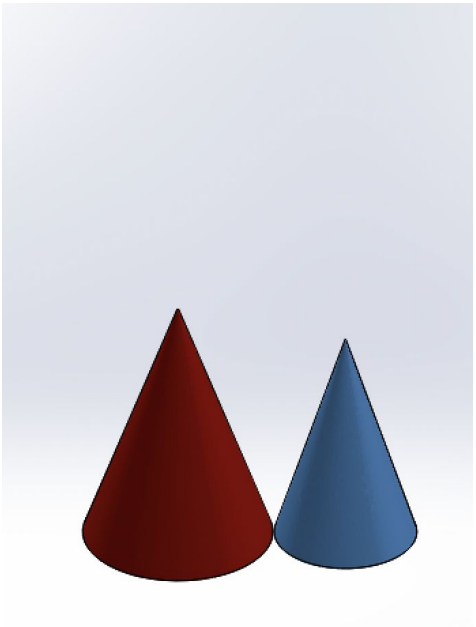
Céline Hocquette, Andrew Cropper
University of Oxford

UK Research and Innovation

UNIVERSITY OF OXFORD

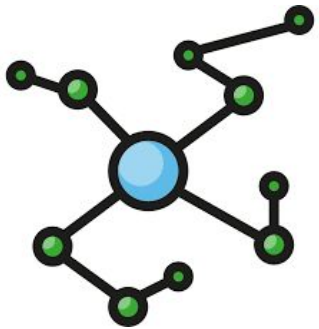
Motivation: learning programs with numerical values





zendo(Structure) ← *piece(Structure,Piece1),*
contact(Piece1,Piece2),
size(Piece2,Size),
*geq(Size,**7**).*

Motivation: learning programs with numerical values



pharma3(Molecule) ← *zincsite(Molecule,Zinc),*
hacc(Molecule,Hydrogen),
dist(Molecule,Zinc,Hydrogen,Distance),
*geq(Distance,**1.7**),*
*leq(Distance,**3.5**).*
pharma3(Molecule) ← *hacc(Molecule,Hydrogen1),*
hacc(Molecule,Hydrogen2),
bond(Molecule,Hydrogen1,Hydrogen2,du),
dist(Molecule,Hydrogen1,Hydrogen2,Distance),
*leq(Distance,**2.7**).*

Related Work

- Existing approaches:
  - cannot scale to infinite domains (ASPAL, ILASP, HEXMIL, ProSynth, Popper, δILP)
  - cannot reason from multiple examples jointly (Progol, MagicPopper)
  - cannot learn chained numerical literals sharing variables (Aleph)
  - cannot learn recursive programs, optimal programs (textually minimal) (Progol, Aleph)

Our approach

Program synthesis approach to learn programs with numerical values

- infinite and continuous domains
- complex numerical reasoning
- reason from multiple examples

Decompose the learning task in two stages:

- **Program search**: generate partial hypotheses with variables in place of numerical values
- **Numerical search**: searches for numerical values to fill in the numerical variables.

Our approach

Program search: generate partial hypotheses with variables in place of numerical values

H : f(List) ← length(List,Length), leq(Length,N), @numerical(N)

Our approach

Numerical search: searches for numerical values to fill in the numerical variables.

Our approach

- finds values for the intermediate variables given the positive and negative examples

*H : f(List) ← length(List,Length), geq(**Length**,N), @numerical(N)*

S_P(Length) = {5, 4} and S_N(Length) = {3, 2}

Positive	Negative
f([3,7,8,2,4]) f([2,7,0,1])	f([8,0,4]) f([1,2])

Our approach

- translates the numerical search as a SMT formula

*H : f(List) ← length(List,Length), geq(**Length**,N), @numerical(N)*

S_P(Length) = {5, 4} and S_N(Length) = {3, 2}

$5 \geq N \wedge 4 \geq N \wedge \neg(3 \geq N) \wedge \neg(2 \geq N)$

Positive	Negative
f([3,7,8,2,4]) f([2,7,0,1])	f([8,0,4]) f([1,2])

Our approach

- substitutes numerical variables with a solution to the SMT formula

*H : f(List) ← length(List,Length), geq(**Length**,N), @numerical(N)*

S_P(Length) = {5, 4} and S_N(Length) = {3, 2}

$5 \geq N \wedge 4 \geq N \wedge \neg(3 \geq N) \wedge \neg(2 \geq N)$

*f(List) ← length(List,Length), geq(Length,**4**)*

Positive	Negative
f([3,7,8,2,4]) f([2,7,0,1])	f([8,0,4]) f([1,2])

Implementation

We implement our approach in NumSynth:

- It is based on the program synthesis system Popper.
- It uses the SMT solver Z3 in the numerical search stage.

Implementation: built-in numerical literals

Literal	Definition	Example
`geq(A,N)`	$A \geq N$	`geq(A,3)`
`leq(A,N)`	$A \leq N$	`leq(A,5.2)`
`add(A,B,C)`	$A + B = C$	`add(A,B,C)`
`mult(A,N,C)`	$A * N = C$	`mult(A,2,C)`

Implementation: supported fragments

Fragment	NumSynth	Example
Linear real arithmetic	✓	$X + 6.3 * Y \leq 3$
Linear integer arithmetic	✓	$U + 6 * V \leq 3$
Mixed real / integer	✓	$X + 6.3 * V \leq 3$
Integer difference logic	✓	$U - V \leq 4$
Real difference logic	✓	$X - Y \leq 4$
Unit two-variable / inequality	✓	$X + Y \leq 4$
Polynomial real arithmetic	X	$X^2 + Y^2 = 2$
Non-linear integer arithmetic	X	$U^2 = 2$

Experiments

Q1: Can NumSynth learn programs with numerical values?

Q2: How well does NumSynth perform compared to other approaches?

Q1: comparison with other approaches

Task	ALEPH	MAGICPOPPER	NUMSYNTH
interval	69 ± 1	70 ± 0	$\mathbf{99 \pm 1}$
halfplane	$\mathbf{99 \pm 0}$	84 ± 7	96 ± 1
zendo1	98 ± 0	68 ± 3	$\mathbf{99 \pm 0}$
zendo2	51 ± 1	56 ± 1	$\mathbf{96 \pm 1}$
zendo3	71 ± 1	51 ± 1	$\mathbf{96 \pm 1}$
zendo4	63 ± 1	52 ± 1	$\mathbf{94 \pm 1}$
pharma1	82 ± 1	64 ± 3	$\mathbf{99 \pm 0}$
pharma2	83 ± 1	77 ± 2	$\mathbf{95 \pm 1}$
pharma3	81 ± 1	82 ± 1	$\mathbf{98 \pm 1}$
pharma4	76 ± 1	62 ± 2	$\mathbf{92 \pm 1}$
member_between	49 ± 0	75 ± 4	$\mathbf{97 \pm 1}$
last_leq	50 ± 0	51 ± 1	$\mathbf{98 \pm 1}$
next_geq	50 ± 0	50 ± 0	$\mathbf{92 \pm 5}$

Predictive accuracies

Q1: comparison with other approaches

Task	ALEPH	MAGICPOPPER	NUMSYNTH
interval			**0 ± 0**
halfplane	**1 ± 0**	60 ± 26	2 ± 1
zendo1	25 ± 8		**10 ± 1**
zendo2			**17 ± 1**
zendo3			**69 ± 2**
zendo4			**76 ± 2**
pharma1	2 ± 0		**1 ± 0**
pharma2	10 ± 2	7 ± 0	**2 ± 0**
pharma3	24 ± 3	66 ± 5	**20 ± 1**
pharma4	**3 ± 0**		20 ± 0
member_between		161 ± 38	2 ± 0
last_leq			13 ± 1
next_geq			39 ± 6

Learning times

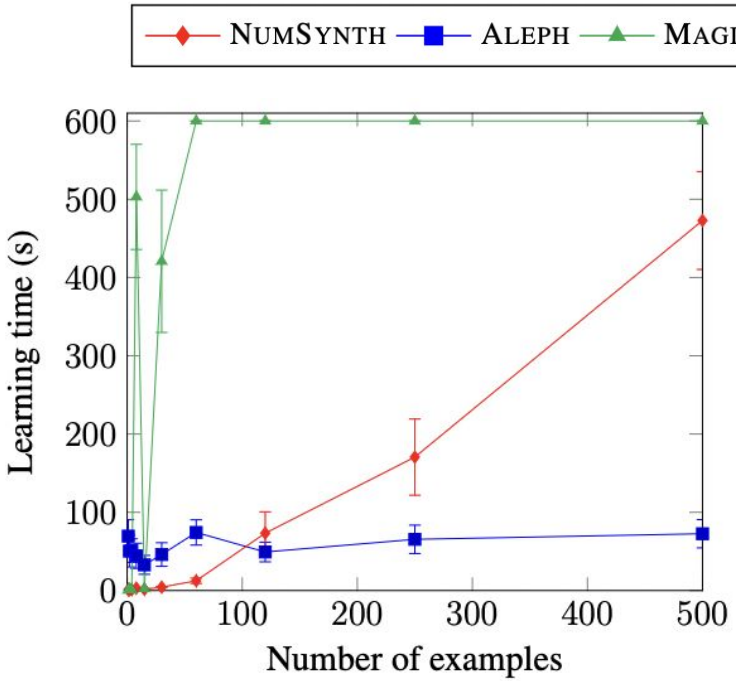
Q1: comparison with other approaches

- **NumSynth can learn programs with numerical values**
- **NumSynth can outperform existing approaches in terms of learning times and predictive accuracies**

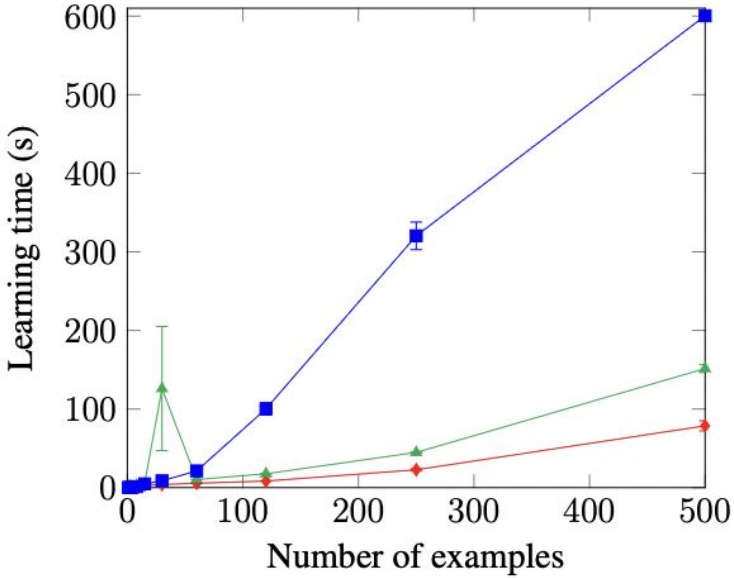
Experiments

Q3: How well does NumSynth scale with the number of examples?

Q3: scalability with respect to the number of examples



zendo1



pharma2

Q3: scalability with respect to the number of examples

- **NumSynth can scale well better than Aleph and MagicPopper with respect to the number of examples**
- **Numerical search stage can be expensive and limit scalability**

Conclusion

NumSynth, approach to learn programs with numerical values. It can:

- outperform state-of-the art approaches,
- learn in infinite and continuous domains,
- learn optimal programs and recursive programs.

Future Work and Limitations

- Scalability with respect to the complexity of the numerical reasoning stage

- Noise: identify numerical values from noisy examples

References

- Corapi, D., Russo, A., Lupu, E.: Inductive logic programming in answer set programming. In: Inductive Logic Programming - 21st International Conference, (2011).
- Raghothaman, M., Mendelson, J., Zhao, D., Naik, M., Scholz, B.: Provenance-guided synthesis of datalog programs. Proceedings of the ACM on Programming Languages 4(POPL), (2019).
- Cropper, A., Morel, R.: Learning programs by learning from failures. Machine Learning 110(4), 801–856 (2021).
- Hocquette, C.; and Cropper, A. 2022. Learning programs with magic values. *Machine Learning*.
- Srinivasan, A.: The ALEPH manual. Machine Learning at the Computing Laboratory (2001).
- Srinivasan, A., Camacho, R.: Numerical reasoning with an ILP system capable of lazy evaluation and customised search. The Journal of Logic Programming 40(2), 185–213 (1999).
- Moura, L. d.; and Bjørner, N. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algo- rithms for the Construction and Analysis of Systems*, 337–340. Springer.

Tasks

*zendo4(A):- piece(A,B), size(B,C), geq(C,**1.23**), leq(C,**4.66**).*
*zendo4(A):- piece(A,B), position(B,X,Y), leq(X,**3.45**), leq(Y,**6.87**).*
*zendo4(A):- piece(A,B), contact(B,C), rotation(C,D), leq(D,**1.18**).*

*pharma4(A):- zincsite(A, B), hacc(A, C), dist(A, B, C, D), leq(D,**4.18**),geq(D,**2.22**).*
*pharma4(A):- hacc(A, C), hacc(A, E), dist(A, B, C, D), geq(D,**1.23**), leq(D,**3.41**).*
*pharma4(A):- zincsite(A, C), zincsite(A, B), bond(B,C,du), dist(A, B, C, D), leq(D,**1.23**).*

*f(A):- head(A,19),tail(A,E),head(E,C),geq(C,**27**).*
f(A):- tail(A,B),f(B).