Download slides: https://polgreen.github.io/pdfs/synth.pdf

Program Synthesis

(A brief introduction)

Elizabeth Polgreen, 10th October

Automatically generating code that satifies the user's specification

Hasn’t this all been solved?

GitHub Copilot

DEVOPS GENERAL SECURITY

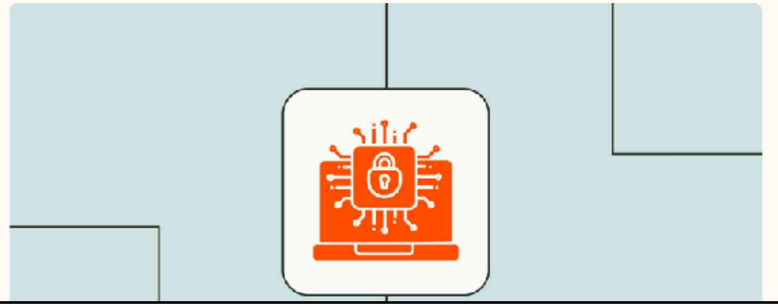
GitHub Copilot: A potential security risk?

By **arnavsharma**

MAY 10, 2023

5 security risks of generative AI and how to prepare for them

By Elisa Silverman · June 14, 2023



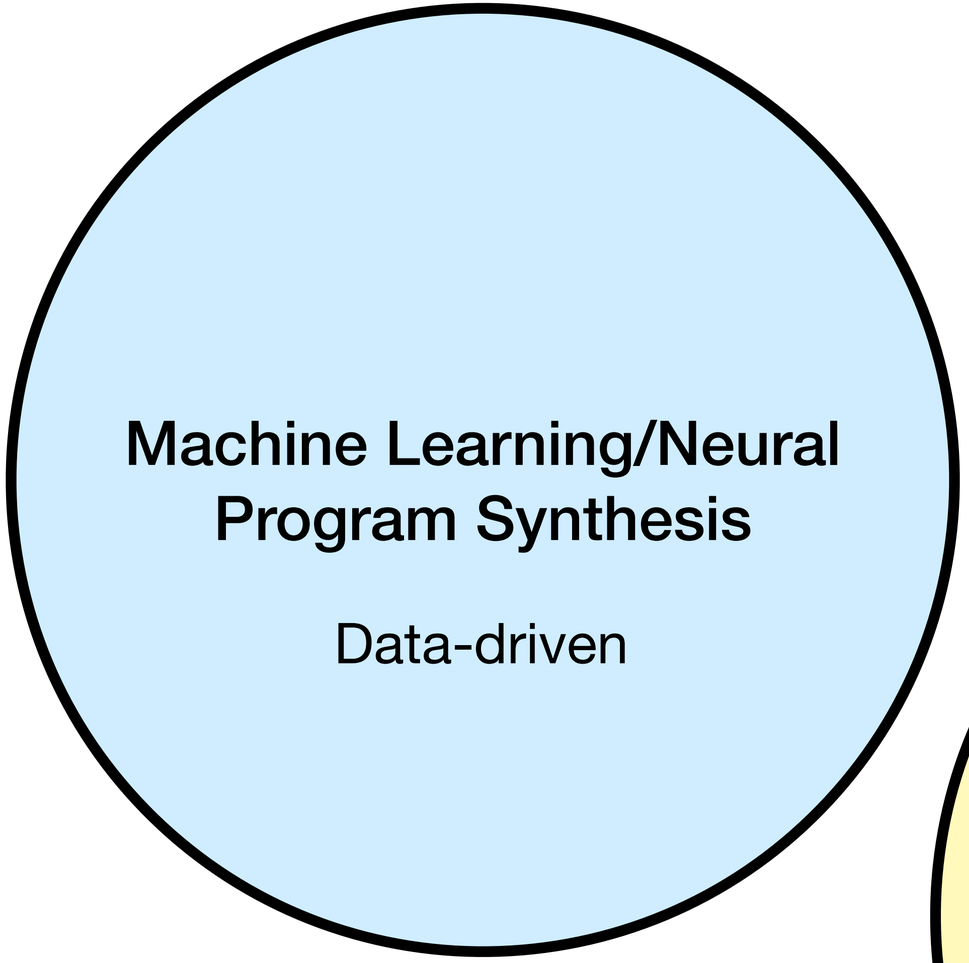
[Submitted on 20 Aug 2021 (v1), last revised 16 Dec 2021 (this version, v3)]

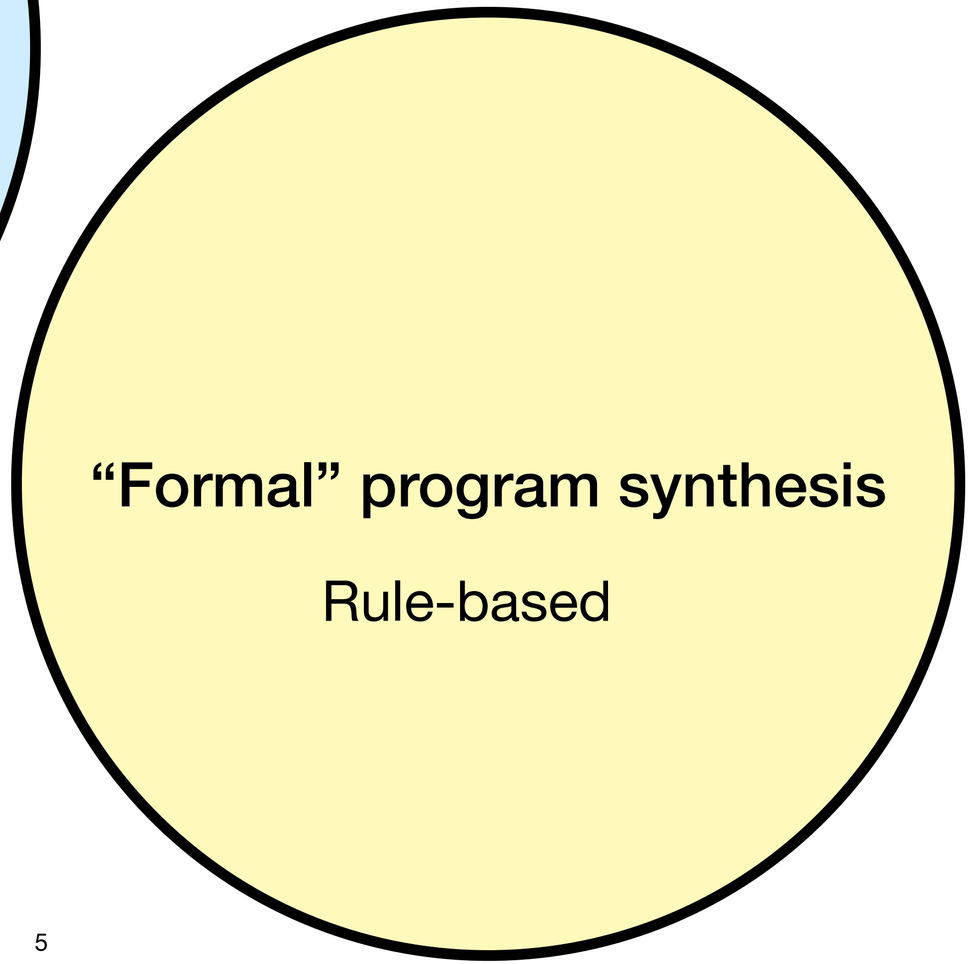
Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions

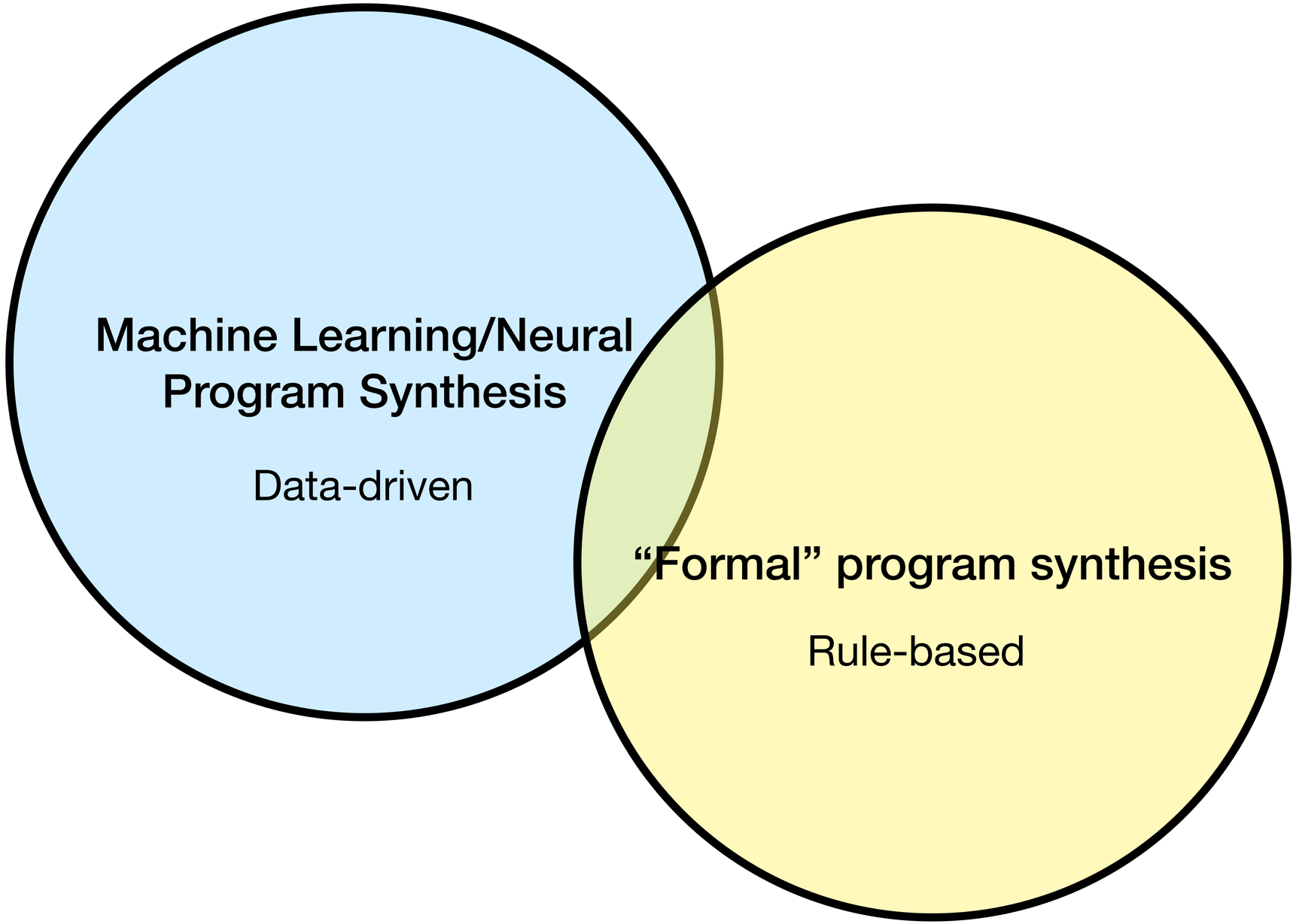
Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri

There is burgeoning interest in designing AI-based systems to assist humans in designing computing systems, including tools that automatically generate computer code. The most notable of these comes in the form of the first self-described `AI pair programmer', GitHub Copilot, a language model trained over open-source GitHub code. However, code often contains bugs – and so, given the vast quantity of unvetted code that Copilot has processed, it is certain that the language model will have learned from exploitable, buggy code. This raises concerns on the security of Copilot's code contributions. In this work, we systematically investigate the prevalence and conditions that can cause GitHub Copilot to recommend insecure code. To perform this analysis we prompt Copilot to generate code in scenarios relevant to high-risk CWEs (e.g. those from MITRE's "Top 25" list). We explore Copilot's performance on three distinct code generation axes -- examining how it performs given diversity of weaknesses, diversity of prompts, and diversity of domains. In total, we produce 89 different scenarios for Copilot to complete, producing 1,689 programs. Of these, we found approximately 40% to be vulnerable.

Comments: Accepted for publication in IEEE Symposium on Security and Privacy 2022


Machine Learning/Neural
Program Synthesis
Data-driven


“Formal” program synthesis
Rule-based


Machine Learning/Neural Program Synthesis
Data-driven
“Formal” program synthesis
Rule-based

This lecture

- What is formal synthesis
    - CounterExample Guided Inductive Synthesis:
        - Guessing
        - Checking
- Applications of formal synthesis
    - Translating code from one language to another
    - Synthesising invariants/ranking functions
    - Learning from sparse data
    - Helping machine learning to play Atari games

Formal Program Synthesis

$$\exists P \forall x \,.\, \sigma(P, x)$$

Does there exist a function P such that, for all possible inputs x, the specification σ will evaluate to true for P and x.

Formal Program Synthesis

$$\exists P \forall x \,.\, \sigma(P, x)$$

Does there exist a function P such that, for all possible inputs x, the specification σ will evaluate to true for P and x.

σ is a quantifier free formula in a background theory, e.g., Linear Integer Arithmetic

NB: we can write specs with input-output examples as quantifier free formula

Formal Program Synthesis

```
int f(int x, int y)
{
  ???
}
@ensures: @ret ≥ x ∧ @ret ≥ y ∧ (@ret = x ∧ @ret = y)
```

Formal Program Synthesis

```
int f(int x, int y)
{
   ???
}
@ensures: @ret ≥ x ∧ @ret ≥ y ∧ (@ret = x ∧ @ret = y)
```

$$\exists f.\, \forall x, y.\;\; f(x,y) \geq y \;\; \wedge \;\; f(x,y) \geq x \;\; \wedge \;\; (f(x,y) = x \vee f(x,y) = y)$$

Defining the search space

Syntax-Guided Synthesis

```
int f(int x, int y)
{
  ???
}
```

@ensures: $@ret \geq x \wedge @ret \geq y \wedge (@ret = x \wedge @ret = y)$

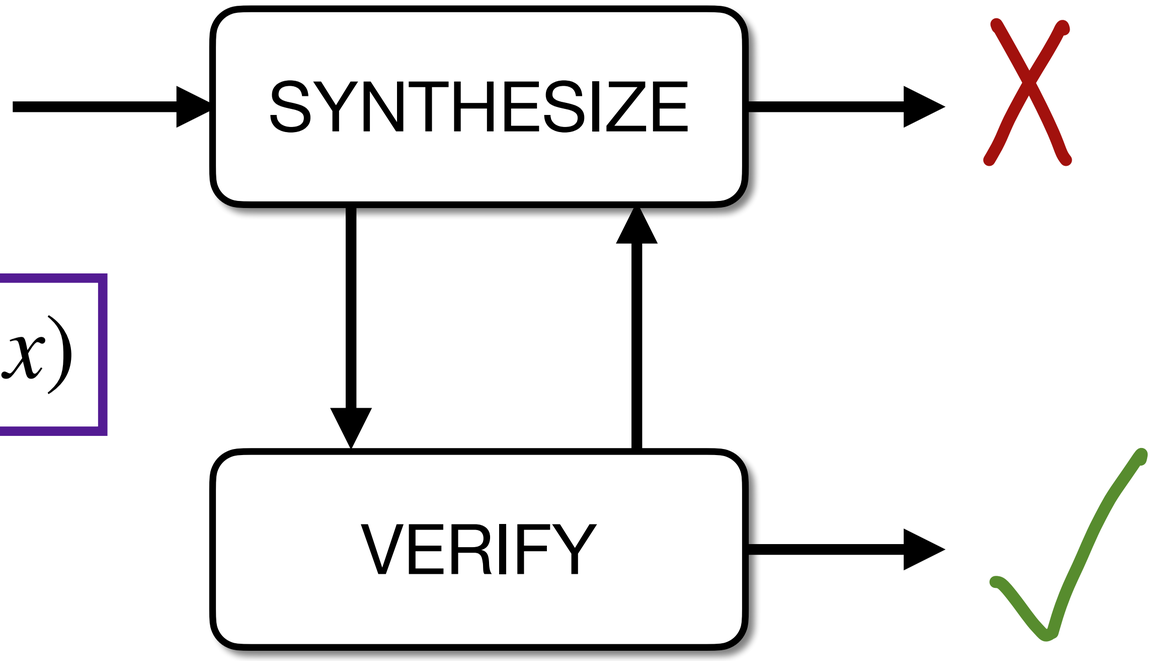
```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

Context Free Grammar (with no cycles)

Algorithms for formal synthesis

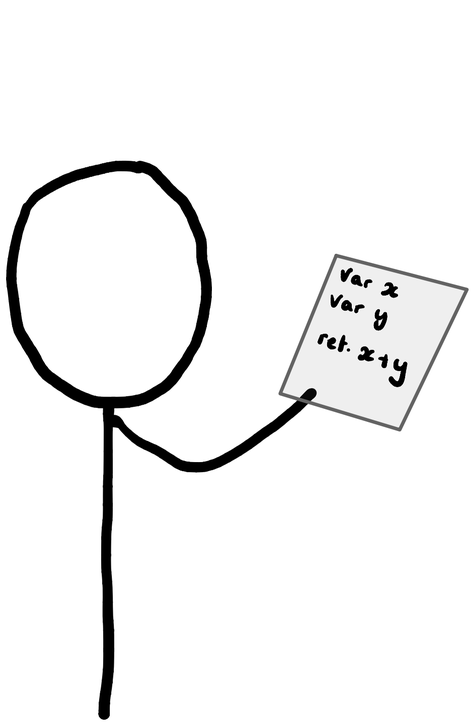
Counterexample Guided Inductive Synthesis

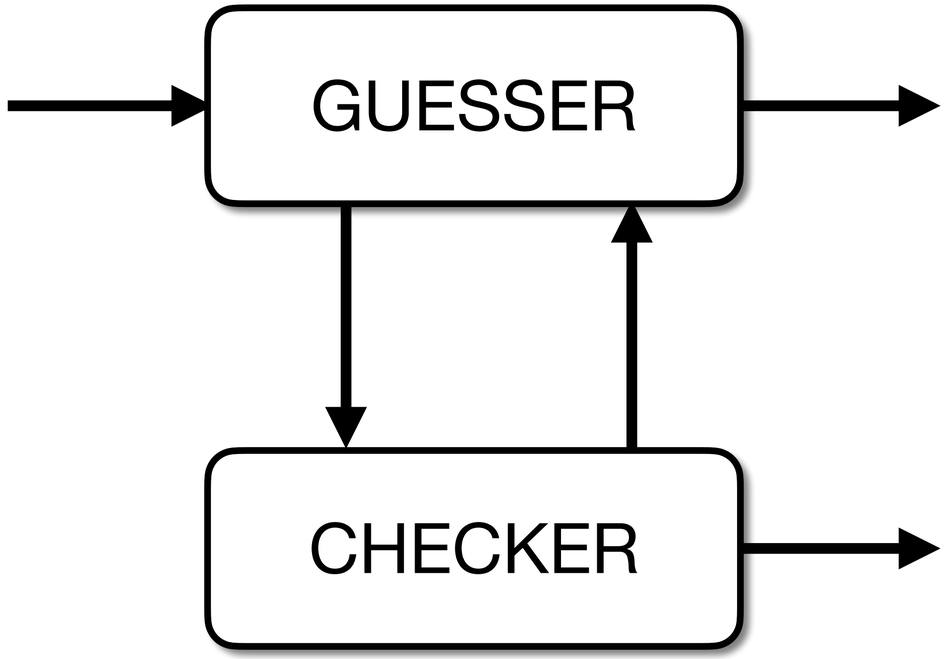
$$\exists P \forall x \,.\, \sigma(P, x)$$



Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis

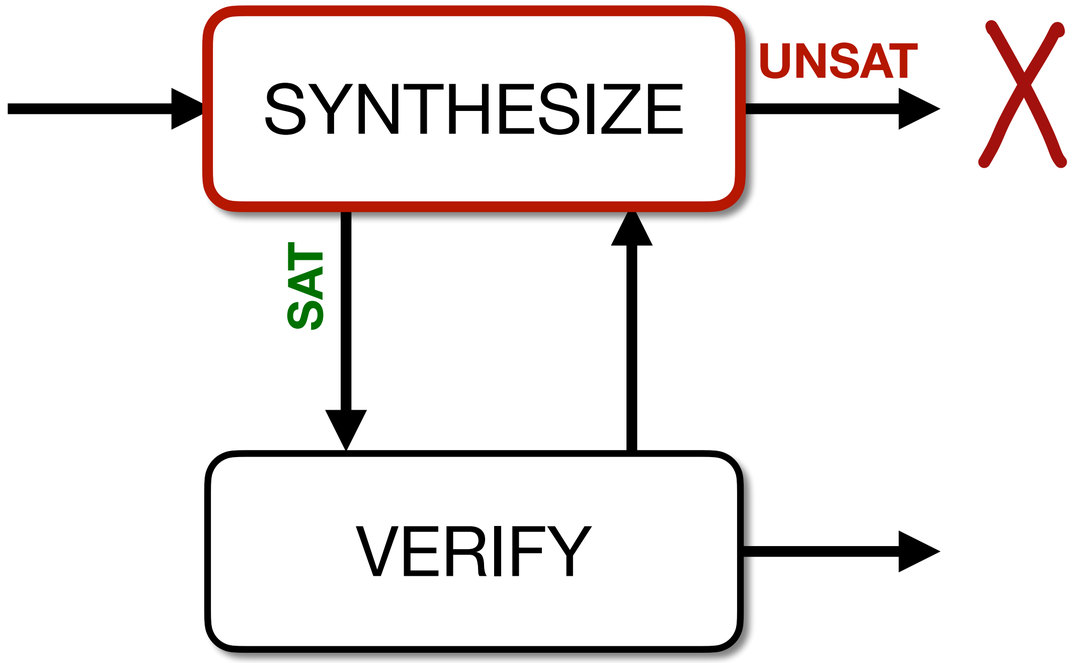




Algorithms for formal synthesis

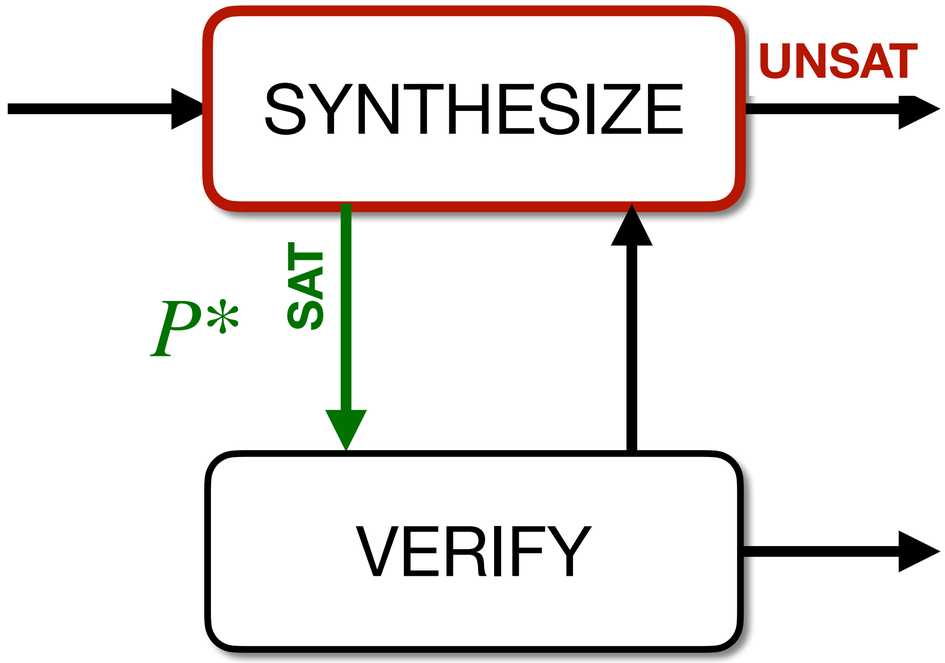
Counterexample Guided Inductive Synthesis

$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$



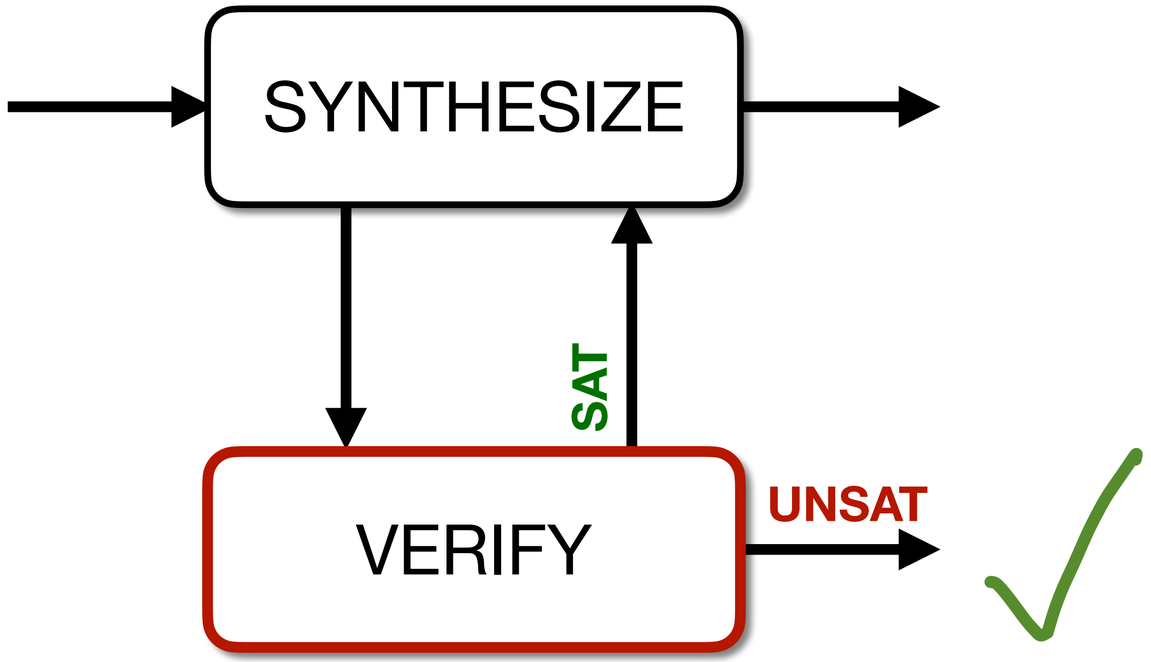
Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis



Algorithms for formal synthesis

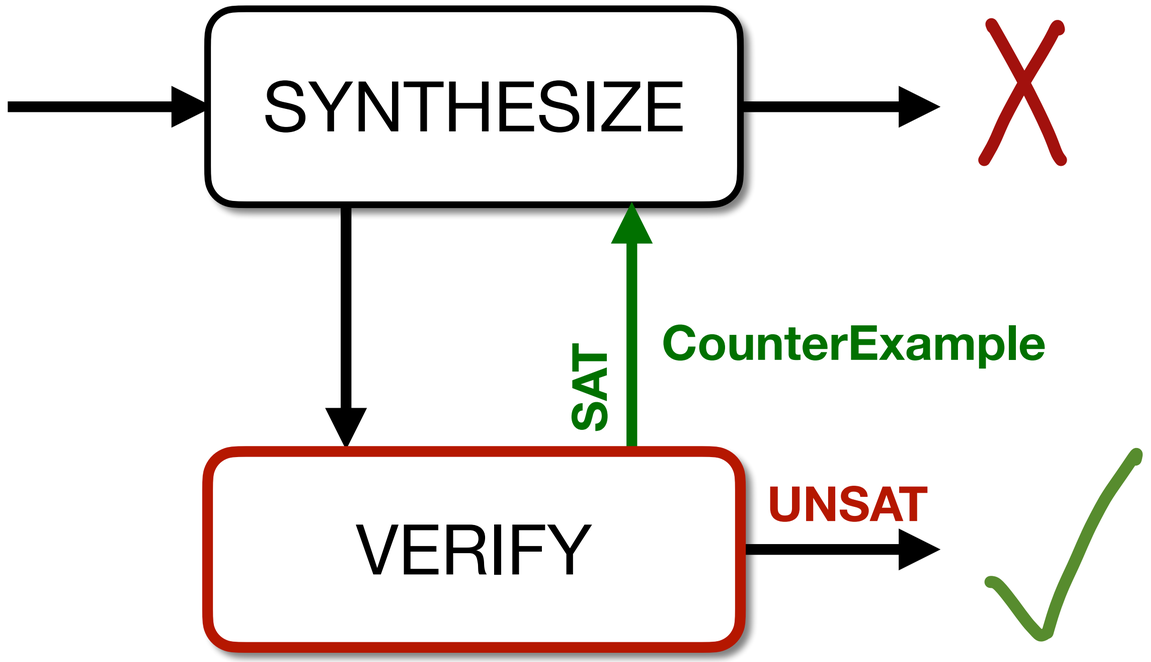
Counterexample Guided Inductive Synthesis



$$\exists x \,.\, \neg\sigma(P*, x)$$

Algorithms for formal synthesis

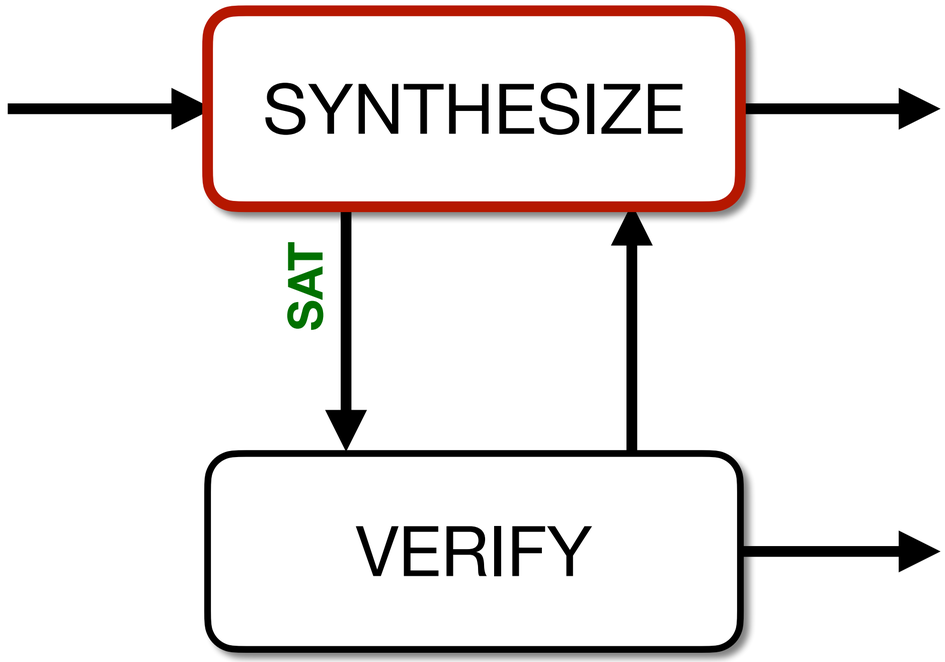
Counterexample Guided Inductive Synthesis



Algorithms for formal synthesis

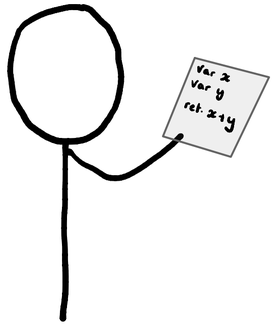
Counterexample Guided Inductive Synthesis

$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$



Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



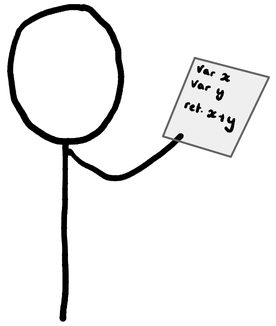
$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$

Guess	Counterexamples
x	

$\exists x\,.\,\neg\sigma(P*, x)$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



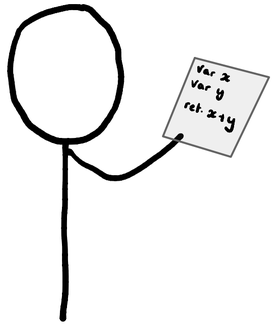
$$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$$

Guess	Counterexamples
x	x=1, y=2

$$\exists x\,.\,\neg\sigma(P*, x)$$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



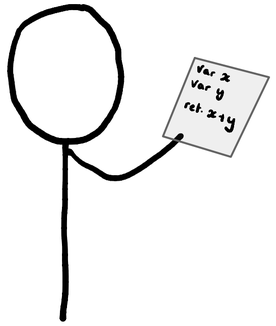
$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$

Guess	Counterexamples
x	x=1, y=2
y	

$\exists x\,.\,\neg\sigma(P*, x)$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



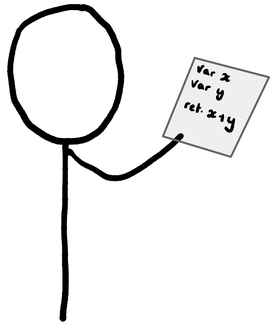
$$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$$

Guess	Counterexamples
x	x=1, y=2
y	x=2, y=1

$$\exists x\,.\,\neg\sigma(P*, x)$$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



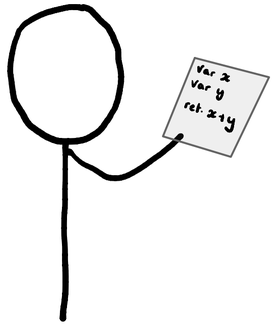
$$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$$

Guess	Counterexamples
x	x=1, y=2
y	x=2, y=1
$(x \geq y)?x : y$	

$$\exists x\,.\,\neg\sigma(P*, x)$$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



$$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$$

Guess	Counterexamples
x	x=1, y=2
y	x=2, y=1
$(x \geq y)?x : y$	Correct!

$$\exists x\,.\,\neg\sigma(P*, x)$$

The Checker

$$\exists x \,.\, \neg \sigma(P*, x)$$

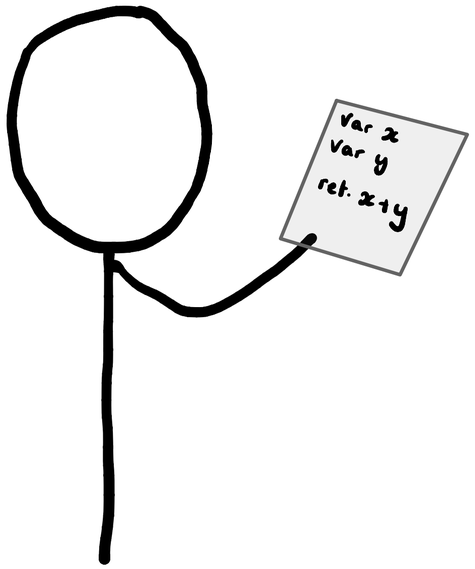
The Checker

- Checking against input-output specifications is easy: just execute the code
- Checking against arbitrary formula is harder:
  - Recall Inf2D: DPLL
  - We can extend DPLL to handle theories (SMT solvers)

$$\exists x \,.\, \neg\sigma(P*, x)$$

The Guesser

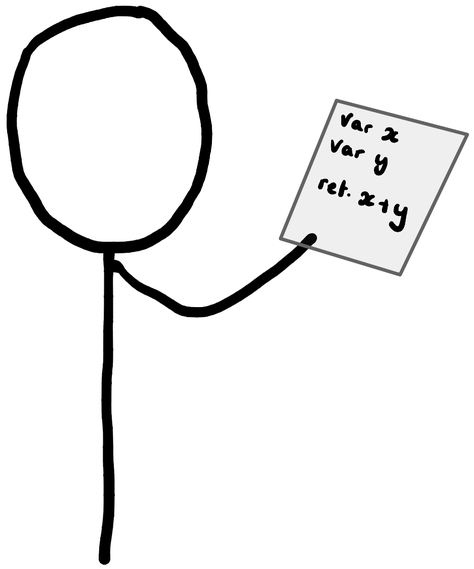
$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$



- Enumerate through the grammar (with heuristics)
- Symbolic encoding

The Guesser

$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$



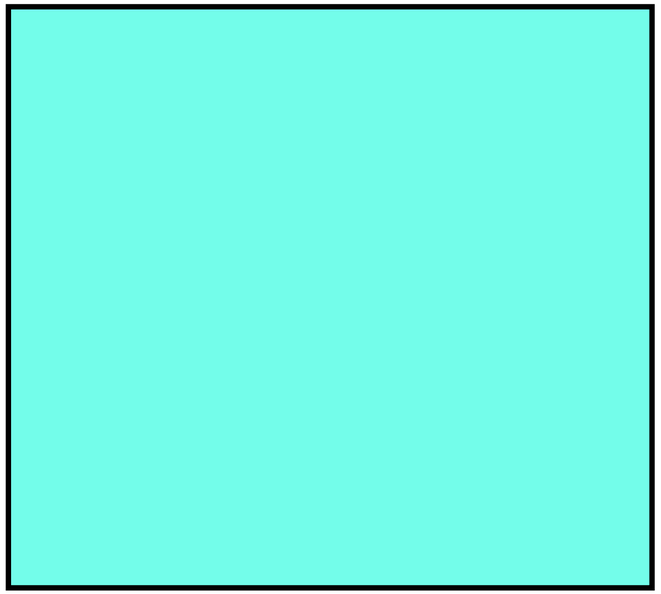
- Enumerate through the grammar (with heuristics)

```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

The Guesser

```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

Programs so far



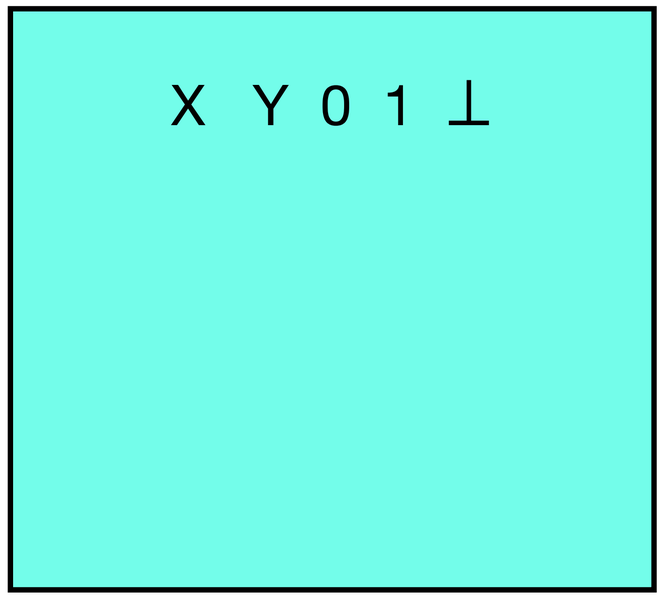
Programs of length 1:

X Y 0 1 ⊥

The Guesser

```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

Programs so far



Programs of length 2:

X+X Y+Y X+0 X+1 Y+0 Y+1 X+Y

-X -Y -0 -1

ite(⊥, X, X) ite(⊥, X, Y) ite(⊥, X, 0). …

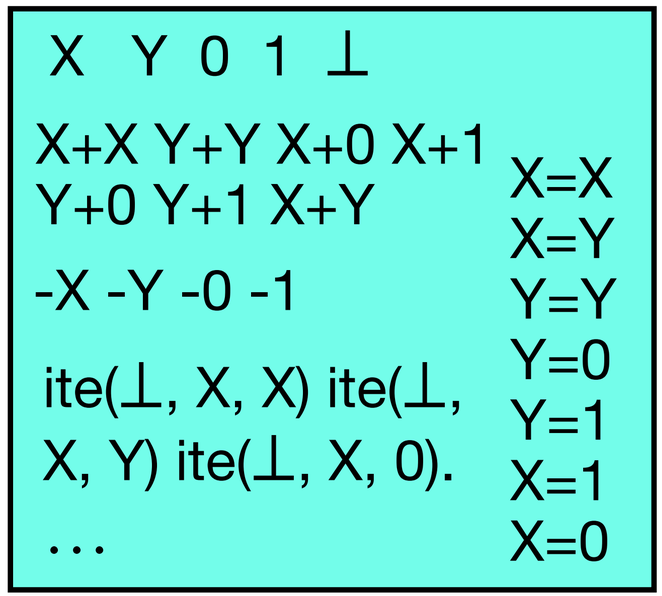
X=X X=Y Y=Y Y=0 Y=1 X=1 X=0

…

The Guesser

```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

Programs so far

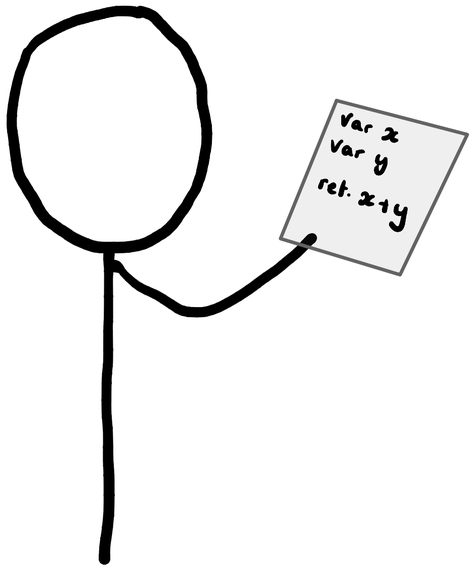


…

Programs of length 3:

The Guesser

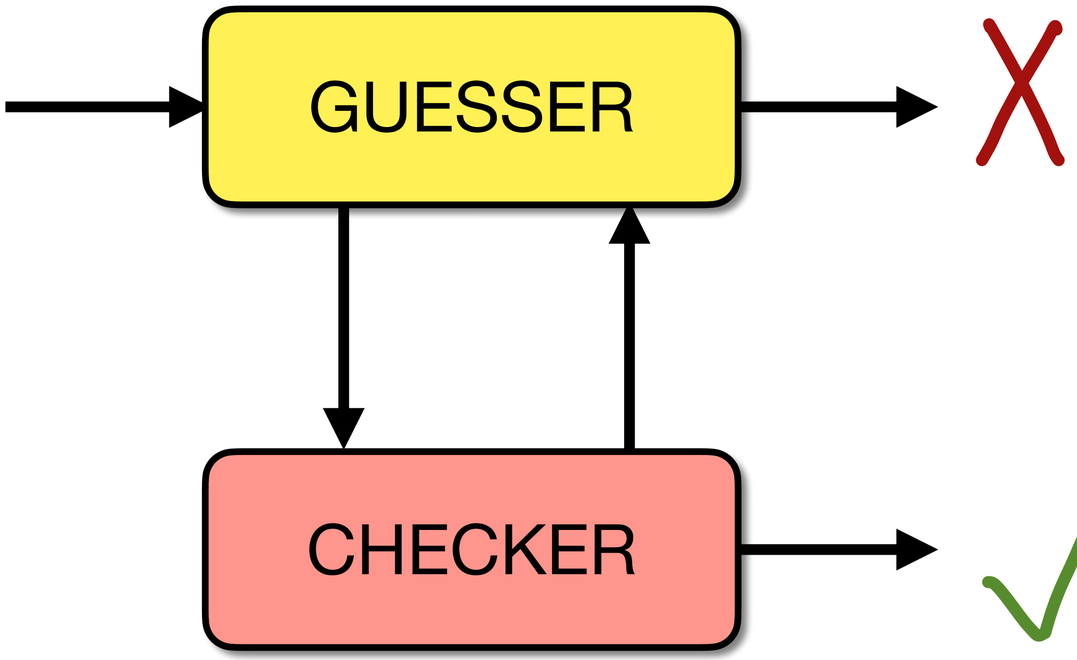
$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$



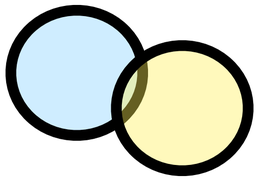
- Enumerate through the grammar (with heuristics)
- Symbolic encoding:
  - use selector variables to choose which parts of the grammar to use.
  - Ask an SMT solver to assign values to those selector variables.

```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

New synthesis algorithms:



- New guessers: clever heuristics, use machine learning
  - Type directed
  - Reinforcement learning
  - Probabilistic grammars

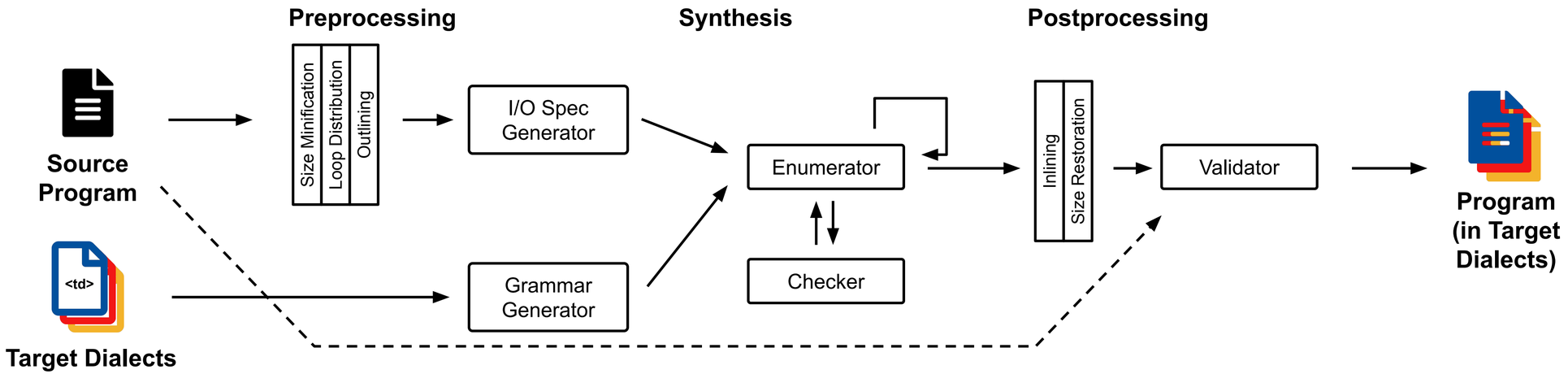


- New checkers: pass more information back to the guesser
  - “Can this partial program ever satisfy the specification”
- New ways to break the problem into smaller problems

Applications

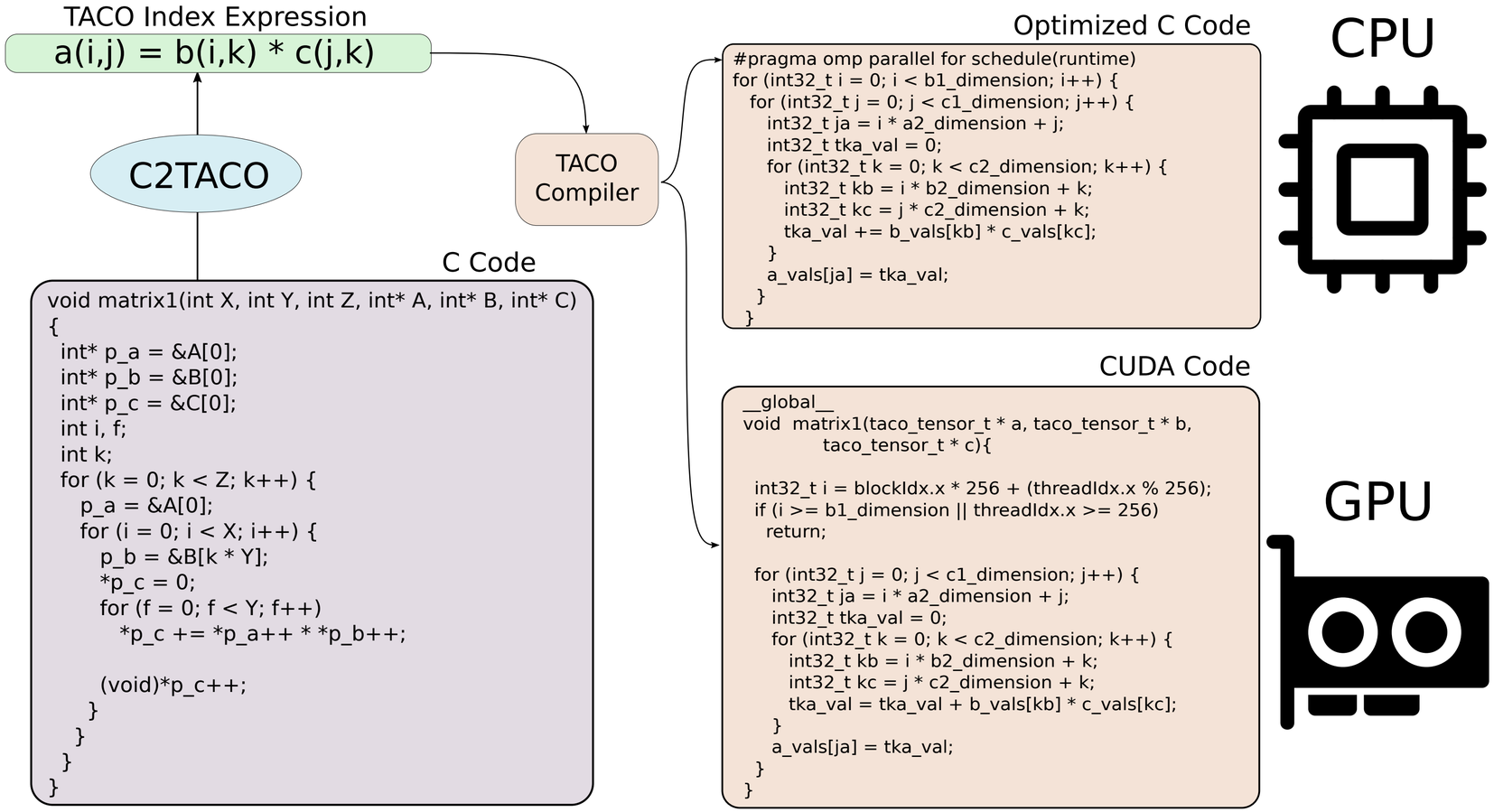
Lifting Code

- Translating code from one language to another
- Without manually writing a syntactic transpiler
- Improved performance



mlirSynth: Automatic, Retargetable Program Raising in Multi-Level IR using Program Synthesis - Alexander Brauckmann, Elizabeth Polgreen, Tobias Grosser, Michael O'Boyle

Lifting Code



C2TACO: Lifting Tensor Code to TACO - José Wesley de Souza Magalhães, Jackson Woodruff, Elizabeth Polgreen, Michael O'Boyle

Synthesizing Invariants

```
int a=0;
int b=1;
while(!button_pressed)
{
  a’ = b;
  b’ = a+b;
  a = a’; b=b’
}
assert(a <= b);
```

How can we prove this property?

Synthesizing Invariants

```
int a=0;
int b=1;
while(!button_pressed)
{
  if(a+b<MAX_INT)
  {
    a’ = b;
    b’ = a+b;
    a = a’; b=b’
  }
}
assert(a <= b);
```

How can we prove this property?

Synthesizing Invariants

```
int a=0;
int b=1;
while(!button_pressed)
{
  if(a+b<MAX_INT)
  {
    a' = b;
    b' = a+b;
    a = a'; b=b'
  }
}
assert(a <= b);
```

How can we prove this property?

$$\exists inv \forall a, b \,.\, init(a,b) \implies inv(a,b) \bigwedge$$

$$inv(a,b) \wedge trans(a,b,a',b') \implies inv(a',b') \bigwedge$$

$$inv(a,b) \implies a \leq b$$

Synthesizing Invariants

```
int a=0;
int b=1;
while(!button_pressed)
{
  if(a+b<MAX_INT)
  {
    a' = b;
    b' = a+b;
    a = a'; b=b'
  }
}
assert(a <= b);
```

How can we prove this property?

$$\exists inv \forall a, b \,.\, init(a, b) \implies inv(a, b) \bigwedge$$

$$inv(a, b) \wedge trans(a, b, a', b') \implies inv(a', b') \bigwedge$$

$$inv(a, b) \implies a \leq b$$

Synthesizing Invariants

```
int a=0;
int b=1;
while(!button_pressed)
{
  if(a+b<MAX_INT)
  {
    a' = b;
    b' = a+b;
    a = a'; b=b'
  }
}
assert(a <= b);
```

How can we prove this property?

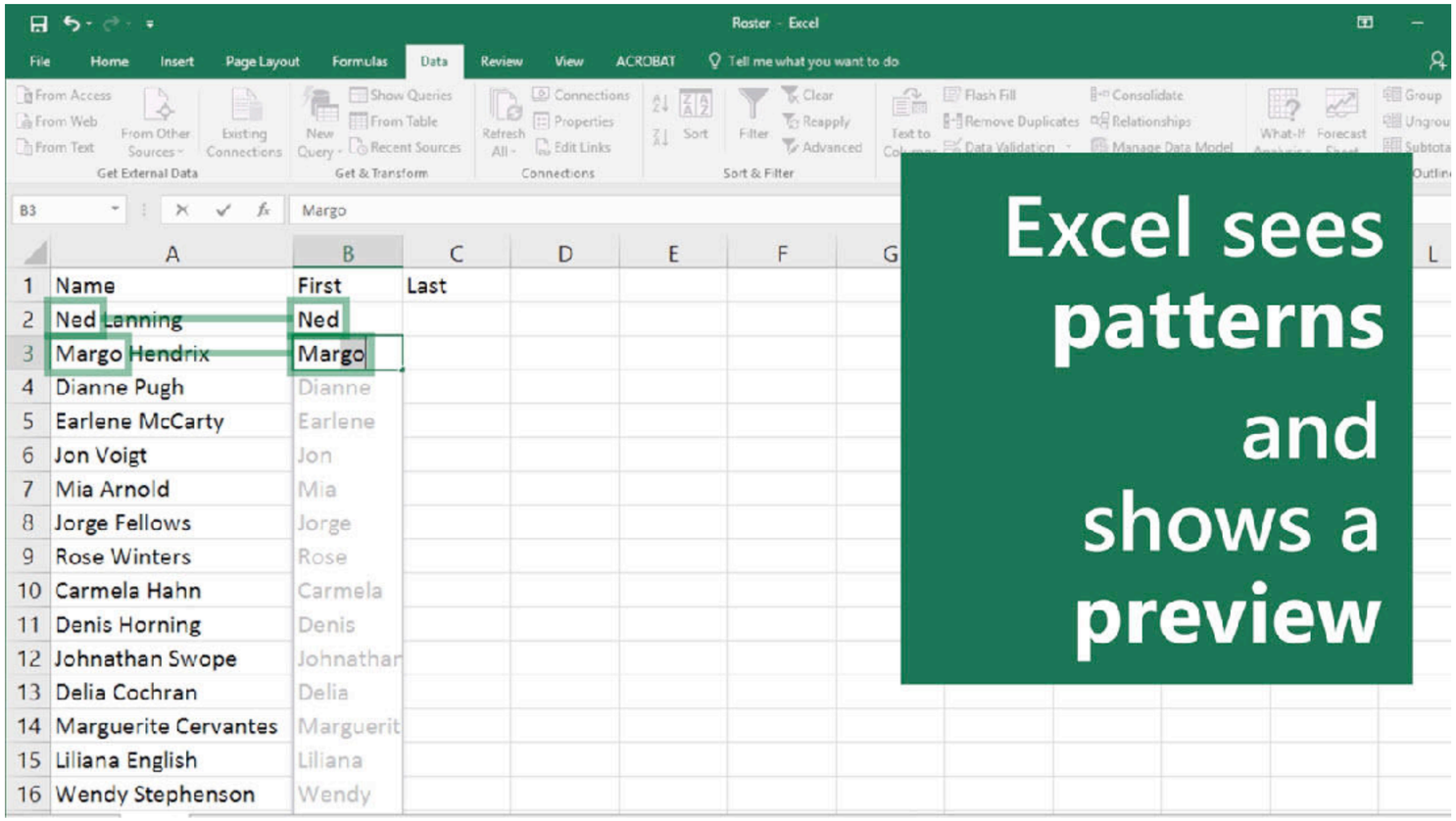
$$\exists inv \forall a, b \,.\, a = 0 \wedge b = 1 \implies inv(a, b) \bigwedge$$

$$inv(a, b) \wedge a' = b \wedge b' = a + b \implies inv(a', b') \bigwedge$$

$$inv(a, b) \implies a \leq b$$

This looks a lot like a program synthesis problem

Data wrangling (FlashFill)



	A	B	C
1	Name	First	Last
2	Ned Lanning	Ned	
3	Margo Hendrix	Margo	
4	Dianne Pugh	Dianne	
5	Earlene McCarty	Earlene	
6	Jon Voigt	Jon	
7	Mia Arnold	Mia	
8	Jorge Fellows	Jorge	
9	Rose Winters	Rose	
10	Carmela Hahn	Carmela	
11	Denis Horning	Denis	
12	Johnathan Swope	Johnathan	
13	Delia Cochran	Delia	
14	Marguerite Cervantes	Marguerit	
15	Liliana English	Liliana	
16	Wendy Stephenson	Wendy	

Programming by example:

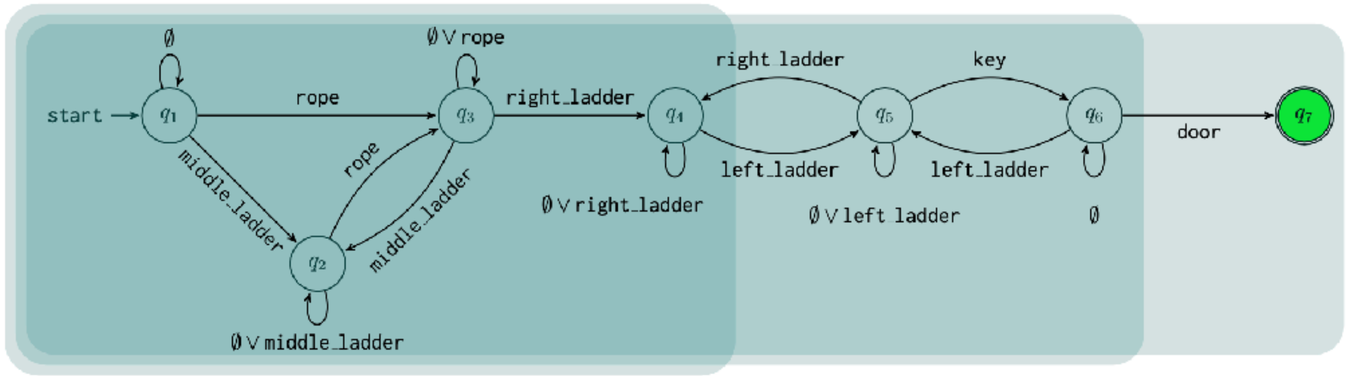
$$\exists P \,.\, P(\text{Ned Lanning}) = \text{Ned} \bigwedge$$

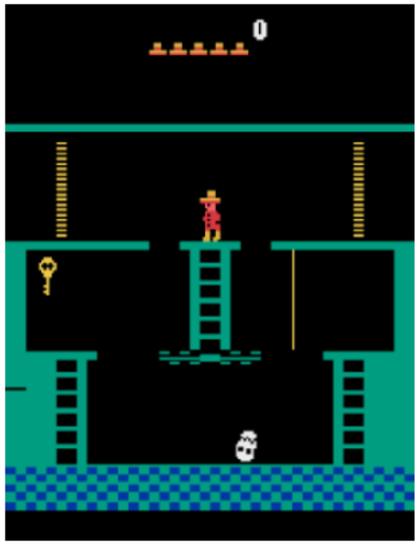
$$P(\text{Margo Hendrix}) = \text{Margo} \bigwedge$$

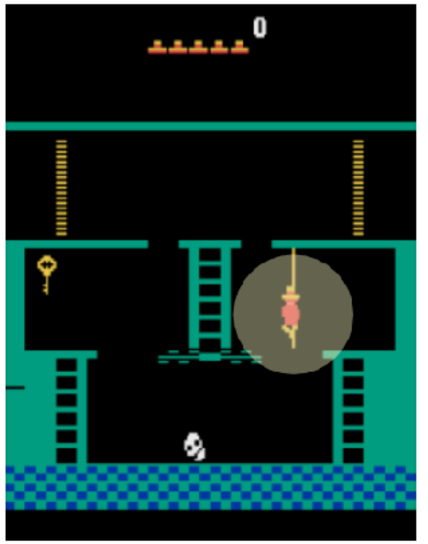
$$\ldots$$

Helping machine learning plan

- Use synthesis to learn the order tasks should be done in. Then give SafeRL the ordering



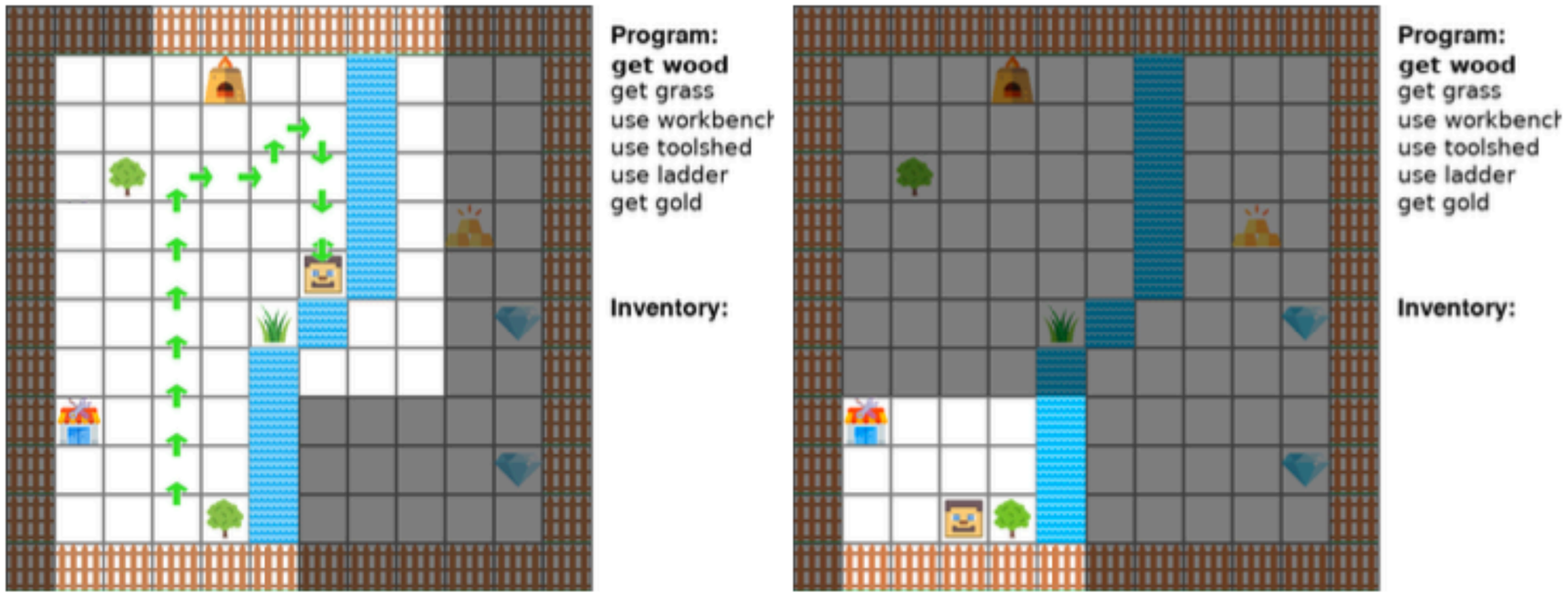




DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning
-Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, Daniel Kroening

Helping machine learning plan

- Model Predictive Program Synthesis: Predict possible worlds based on observations, synthesise a program maximising probability of success, execute program, repeat.

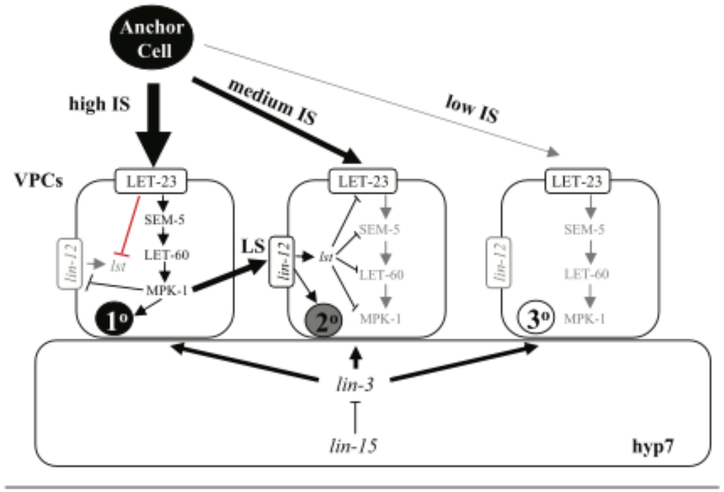


Program Synthesis Guided Reinforcement Learning for Partially Observed Environments - Yichen Yang, Jeevana Priya Inala, Osbert Bastani, Yewen Pu, Armando Solar-Lezama, Martin Rinard

Learning from Sparse Data

- Program synthesis can learn from less data than classic ML methods
- Resulting program is “human readable”

Cell fate decision in C. Elegans: synthesise the model that matches all observations



Synthesis of biological models from mutation experiments - Ali Sinan Koksal, Yewen Pu, Saurabh Srivastava, Rastislav Bodik, Jasmin Fisher, Nir Piterman

Resources

elizabeth.polgreen@ed.ac.uk

- TedX talk by Armando Solar Lezama: https://youtu.be/RzMk7XWVMWw?si=VnH_T-2G_R7gs2Zd
- CAV keynote by Sumit Gulwani: https://youtu.be/PeeFc9Js9ZU?si=GRXpKbQKXUgAFdwx
- Course notes from MIT: https://people.csail.mit.edu/asolar/SynthesisCourse/TOC.htm
- Fourth year projects? Talk to me
- These slides on https://polgreen.github.io/pdfs/synth.pdf