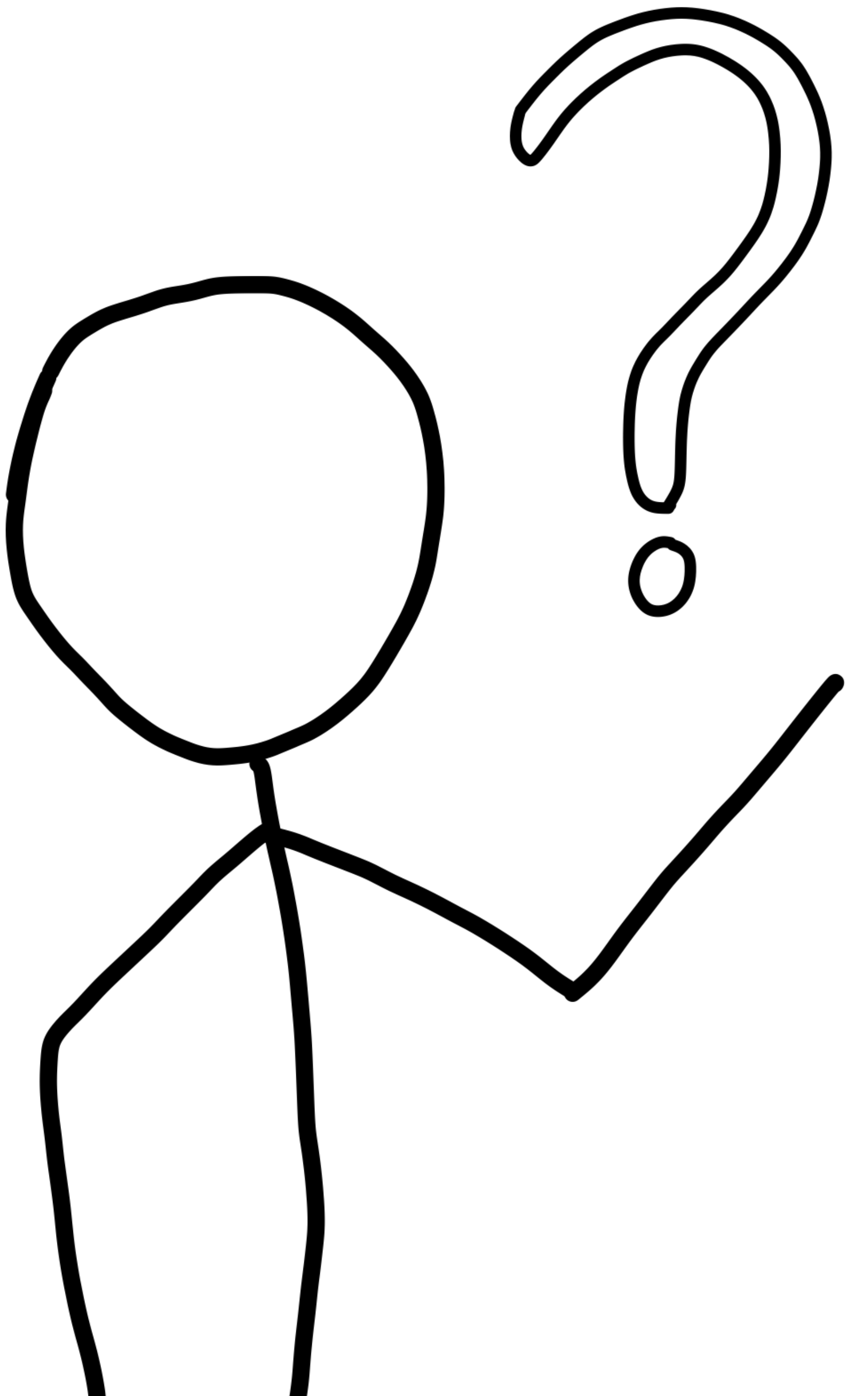
SYNTHESIS is the new SAT

Dr. Elizabeth Polgreen

Berkeley
UNIVERSITY OF CALIFORNIA

SYNTHESIS is the new SAT



SAT

A : Boolean

B : Boolean

$$\boxed{\exists A, B \quad A \land \neg B}$$

A : true

B : false

SMT

(program) Synthesis

SAT

A : Boolean

B : Boolean

$$\exists A, B$$
$$A \wedge \neg B$$

A : true

B : false

SMT

A : Integer

B : Integer

$$\exists A, B$$
$$A > 0 \wedge B < 0$$

A : 10

B : -3

(program) Synthesis

SAT

A : Boolean

B : Boolean

$$\boxed{\exists A, B \quad A \wedge \neg B}$$

A : true

B : false

SMT

A : Integer

B : Integer

$$\boxed{\exists A, B \quad A > 0 \wedge B < 0}$$

A : 10

B : -3

(program) Synthesis

A : integer

B : integer

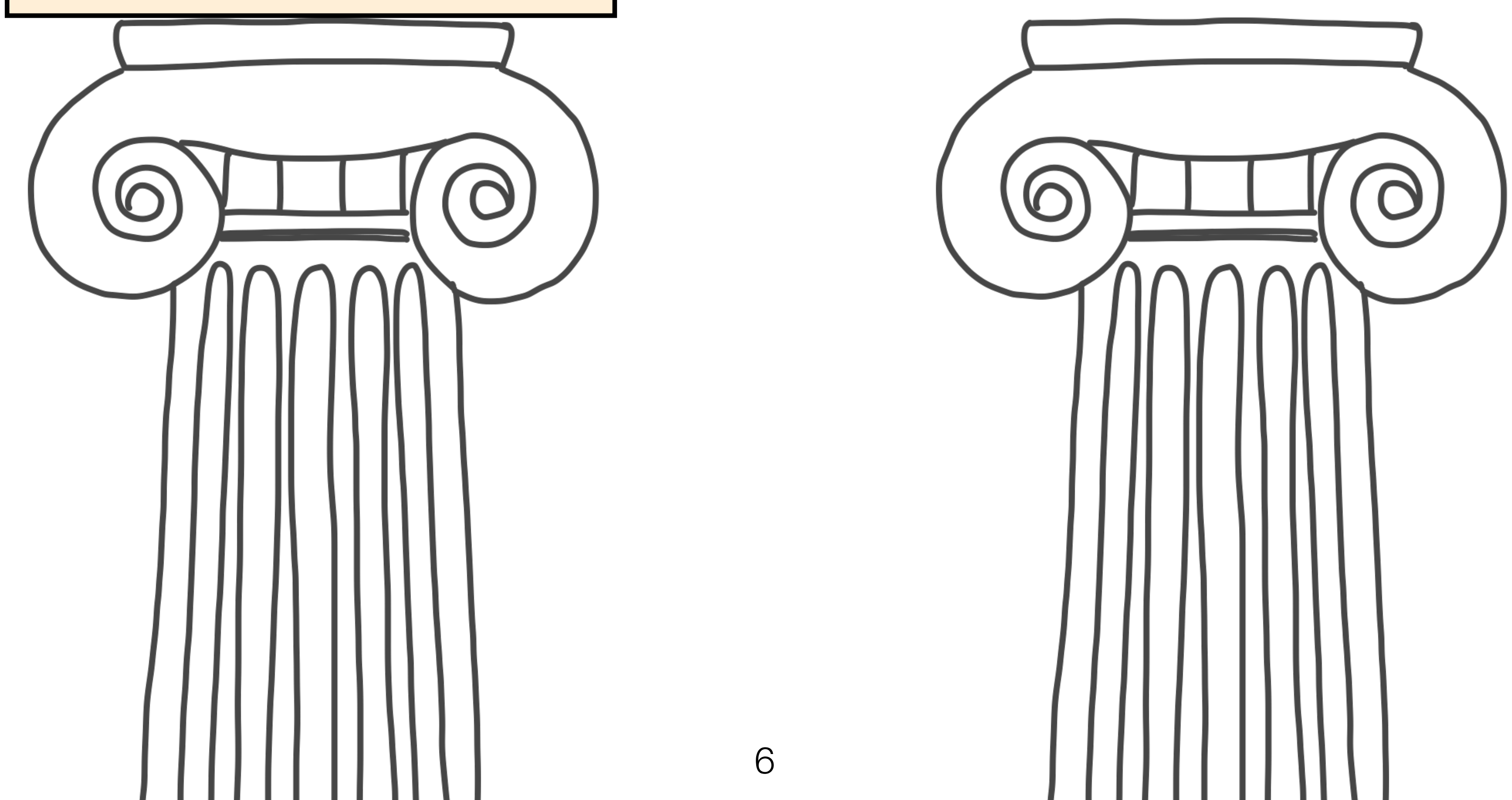
F : integer $\times$ integer $\rightarrow$ integer

$$\boxed{\begin{array}{c} \exists F\, \forall A, B \\ F(A,B) \geq A \wedge \\ F(A,B) \geq B \wedge \\ (F(A,B) = A \vee \\ F(A,B) = B) \end{array}}$$

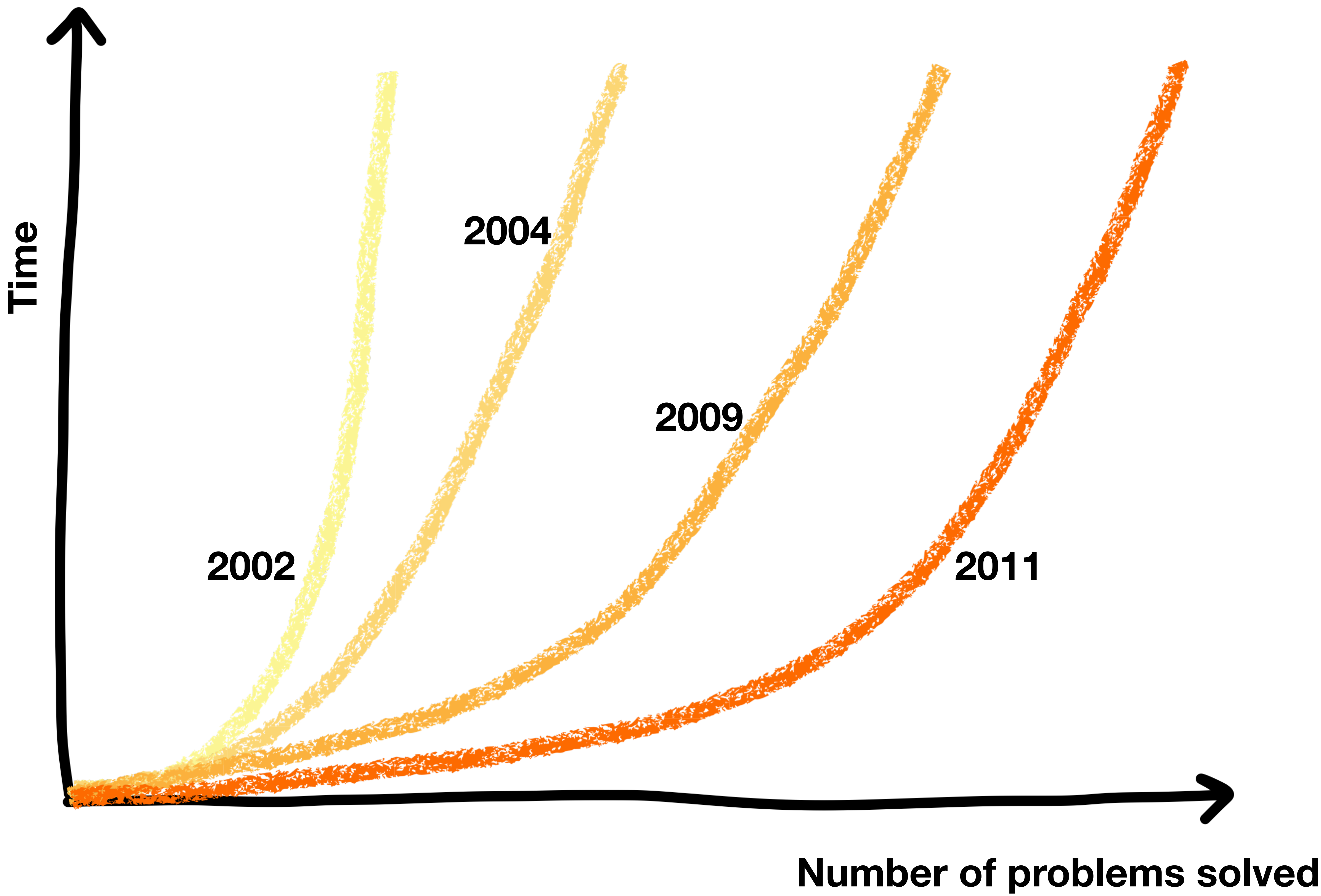
F: max(A,B)

The Success of Boolean Satisfiability Solvers

Algorithmic Improvements

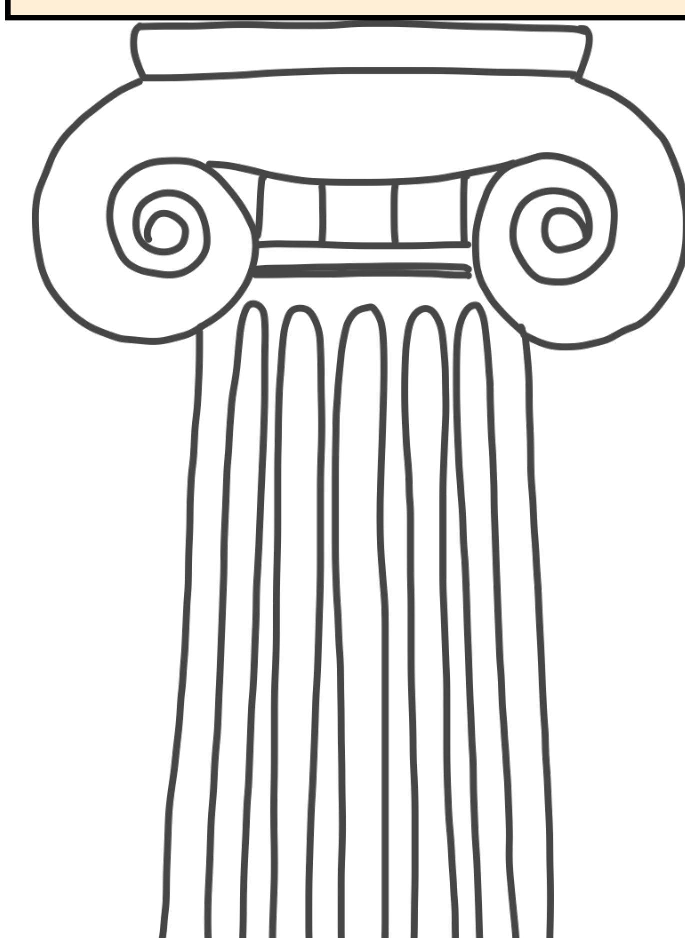


Progress of SAT solvers in SAT competition

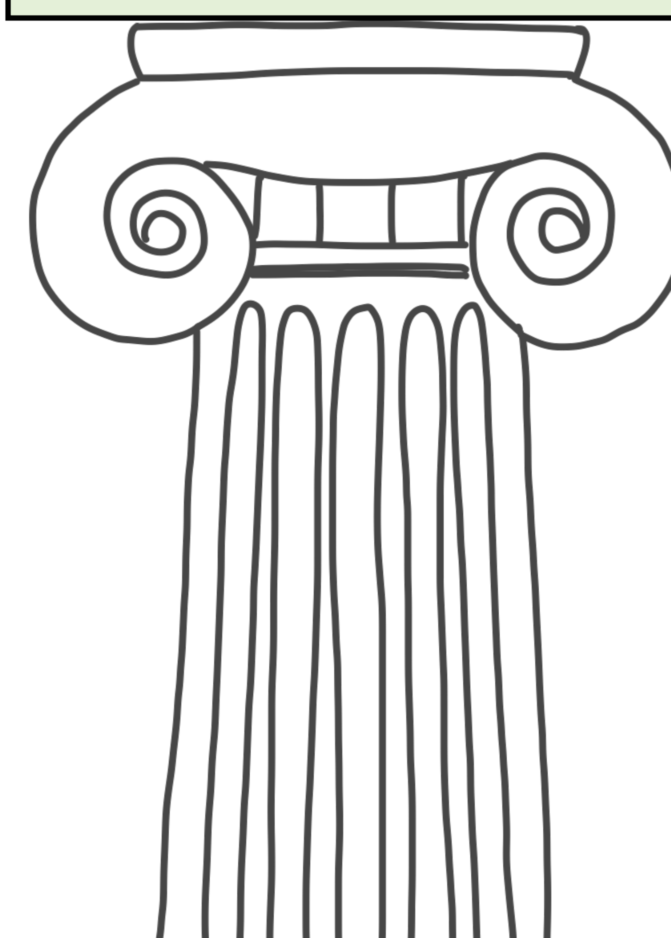


The Success of Boolean Satisfiability Solvers

Algorithmic Improvements



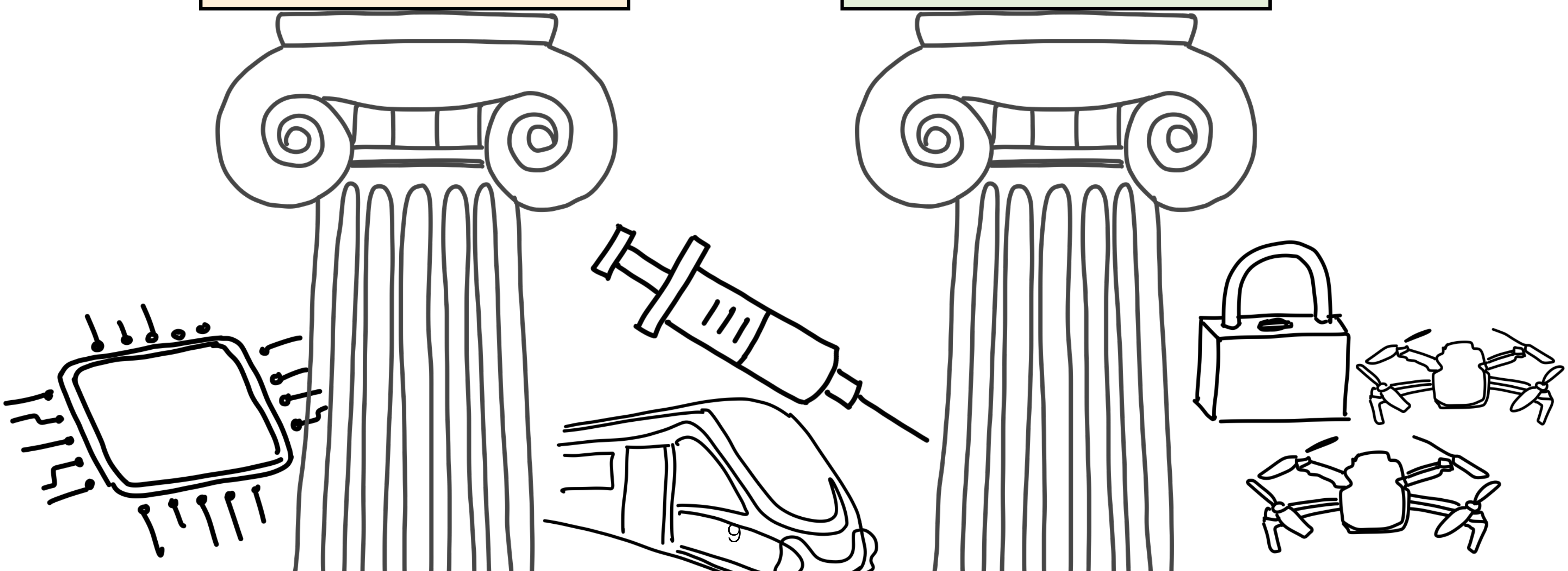
Applications



The Success of Boolean Satisfiability Solvers

Algorithmic Improvements

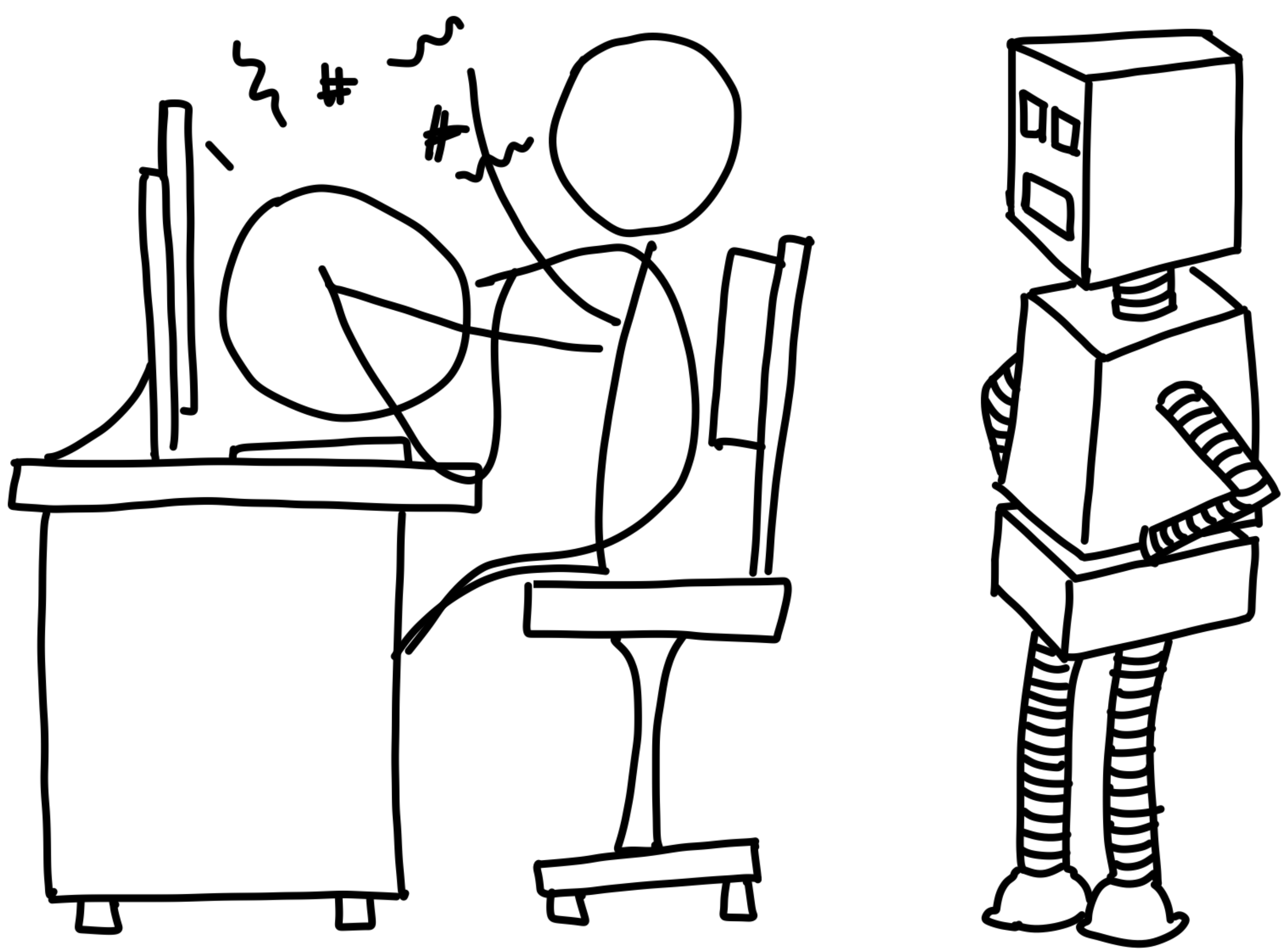
Applications



We all write code

But writing correct code is hard...

SAT solvers allow computers to check code for us.

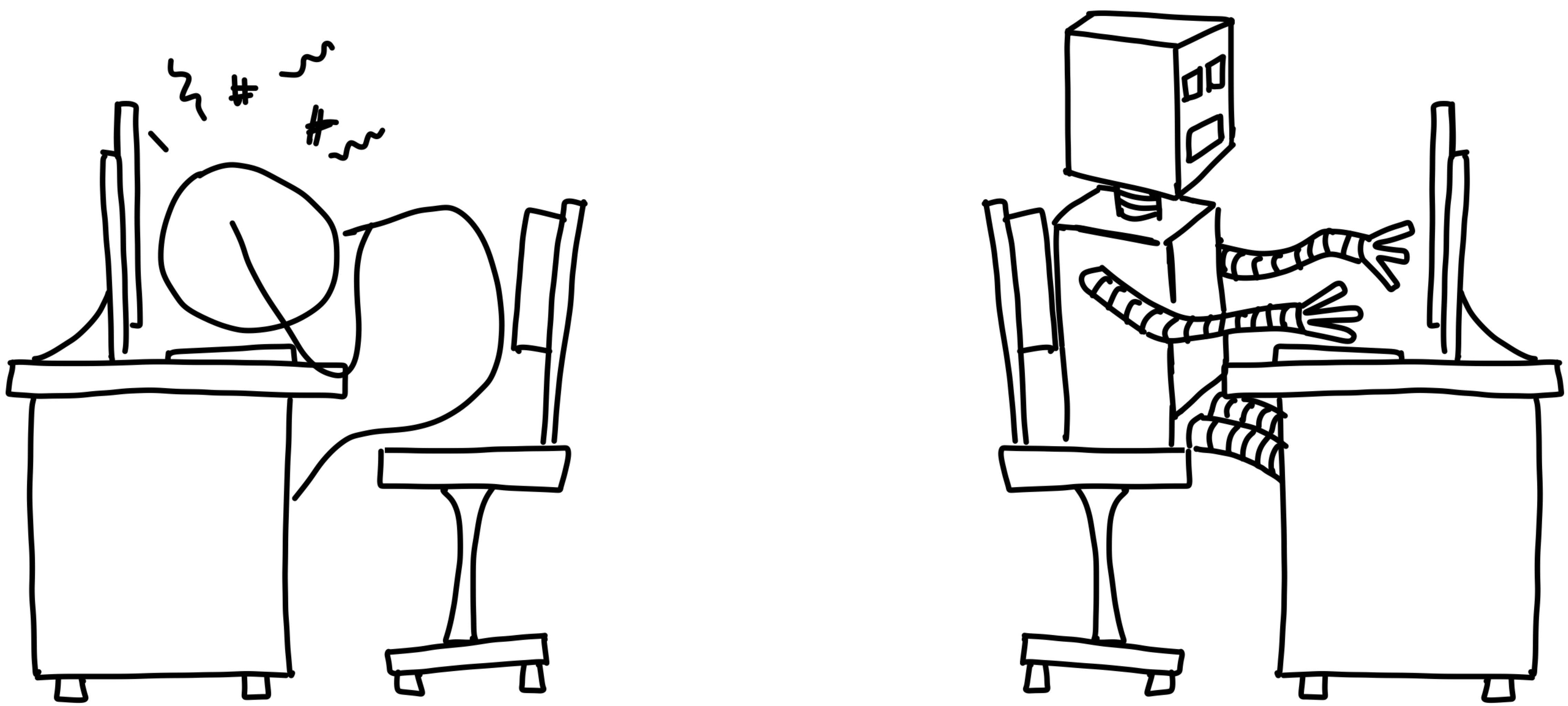


We all write code

But writing correct code is hard...

SAT solvers allow computers to check code for us.

Synthesis could allow computers to repair code for us.

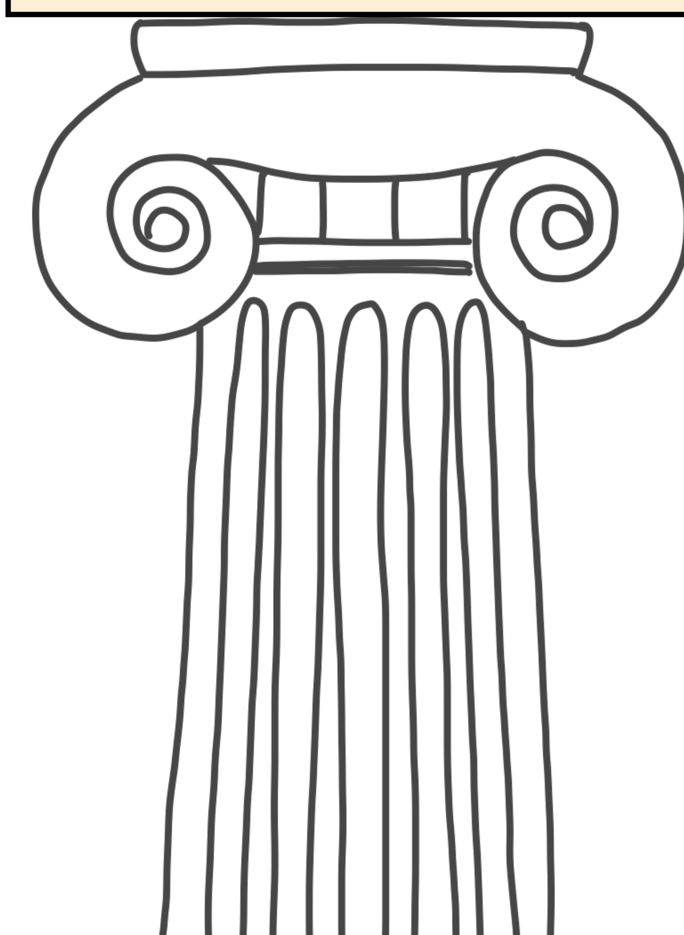


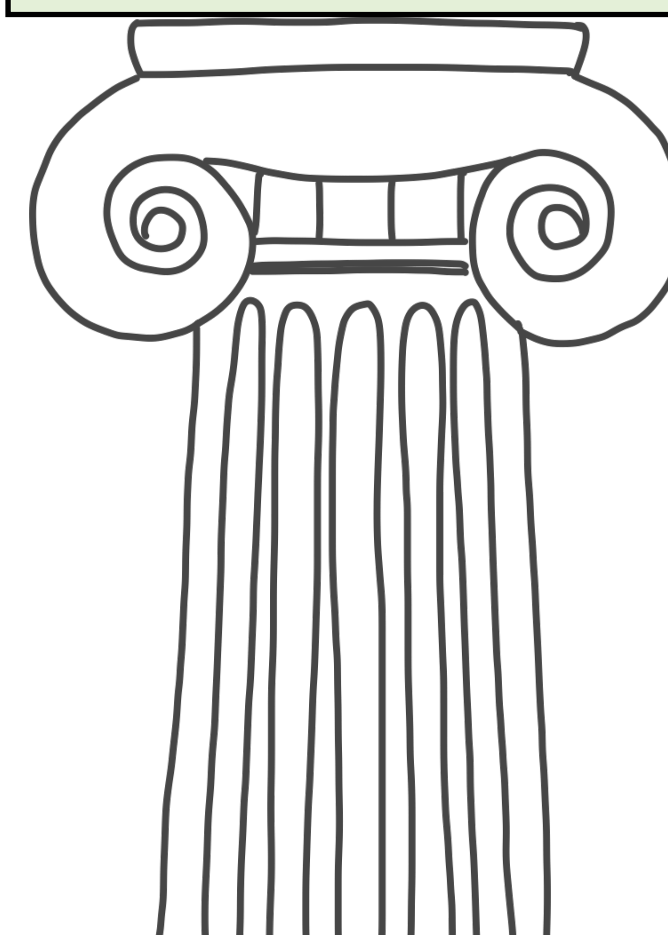
(program) Synthesis

(program) Synthesis

Algorithmic Improvements

Applications

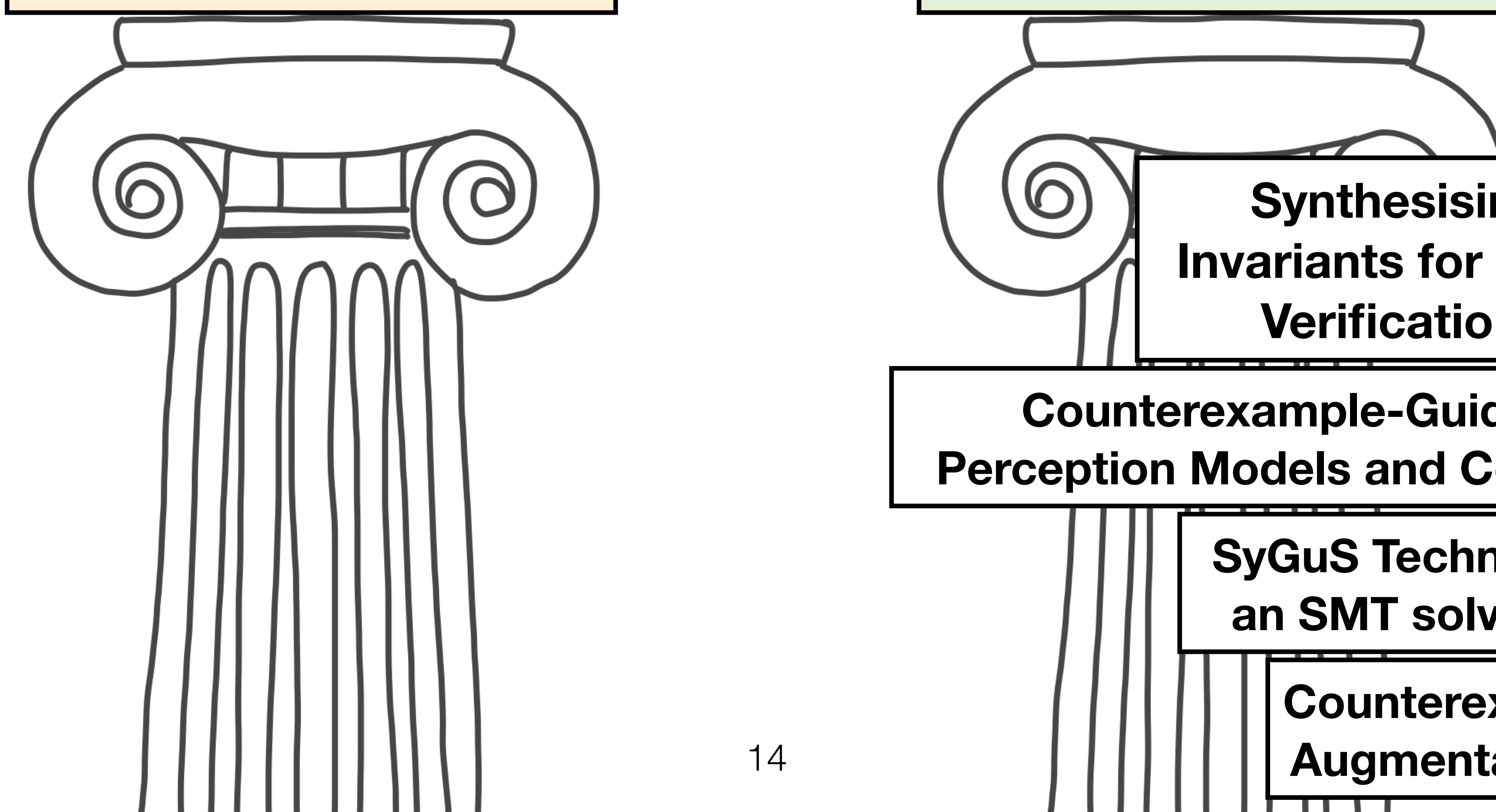




(program) Synthesis

Algorithmic Improvements

Applications

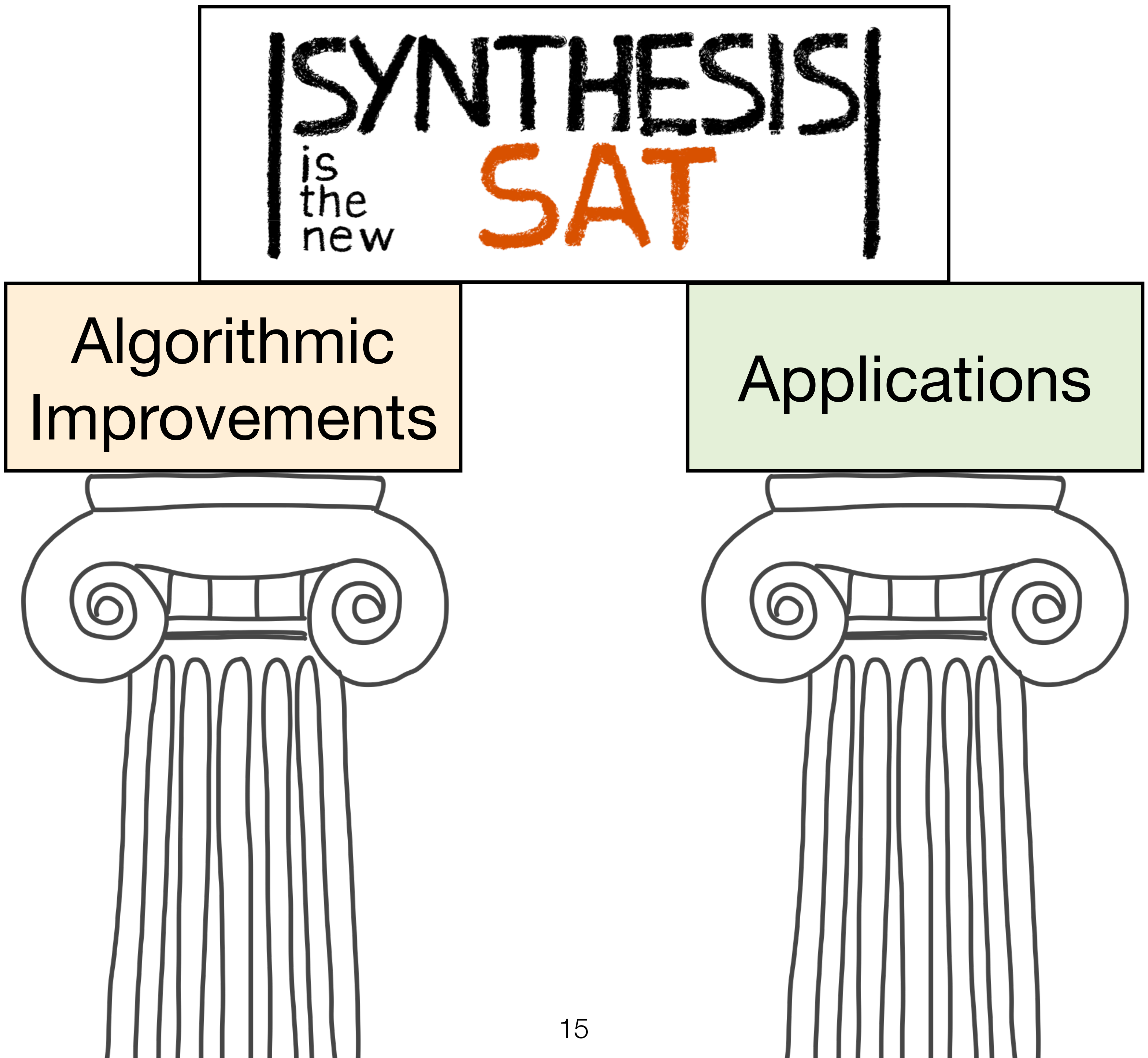


Synthesising Environment Invariants for Modular Hardware Verification - Zhang et al.

Counterexample-Guided Synthesis of Perception Models and Control - Ghosh et al.

SyGuS Techniques in the Core of an SMT solver - Reynolds et al.

Counterexample-Guided Data Augmentation - Dreossi et al.


SYNTHESIS is the new SAT
Algorithmic Improvements
Applications

In this talk

- Define synthesis
- Describe how my research fits into this vision
- Details: CounterExample Guided Inductive Synthesis modulo Theories
- Future

What is formal synthesis?

- Synthesis that satisfies a specification σ.

Input-output examples

constraints

What is formal synthesis?

- Synthesis that satisfies a specification σ.

Input-output examples

constraints

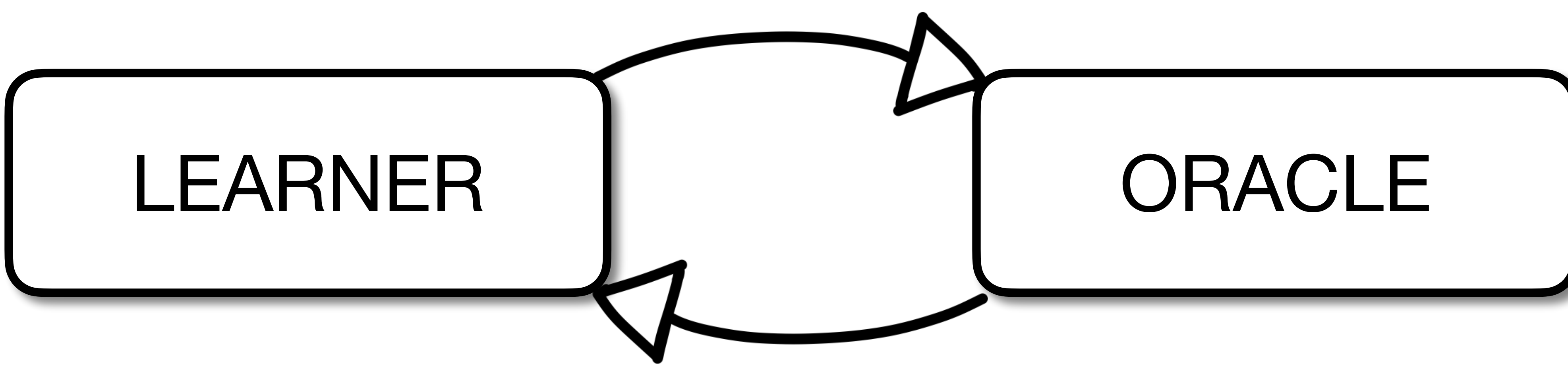
- Can be framed as Oracle Guided Learning.

LEARNER

ORACLE

What is formal synthesis?

- Synthesis that satisfies a specification σ.
- Can be framed as Oracle Guided Learning.



Positive examples

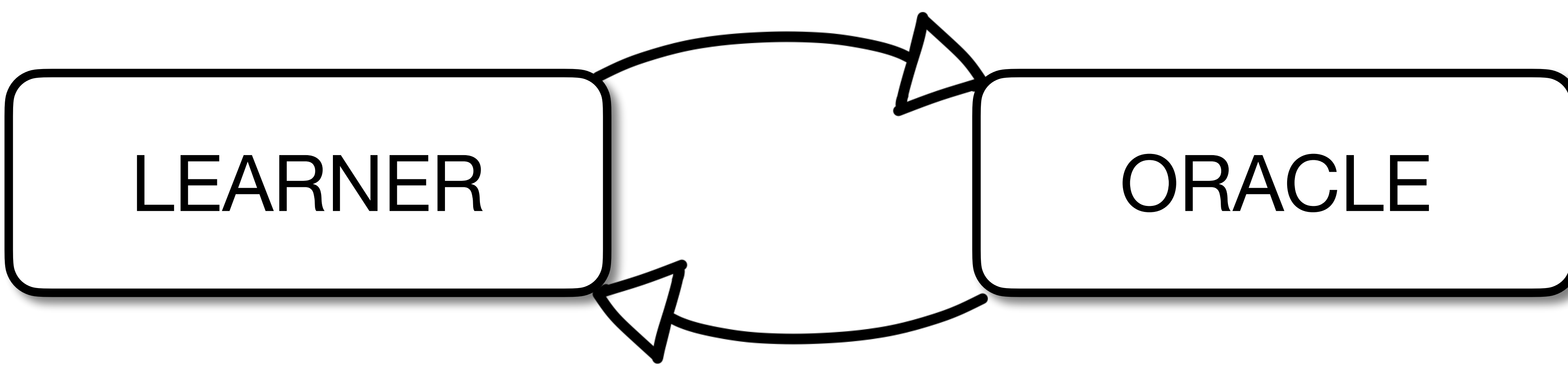
Negative examples

Positive and Negative examples

What is formal synthesis?

CounterExample Guided Inductive Synthesis (CEGIS)[1]

- Logical specification σ.
- Synthesizes expressions/loop-free programs



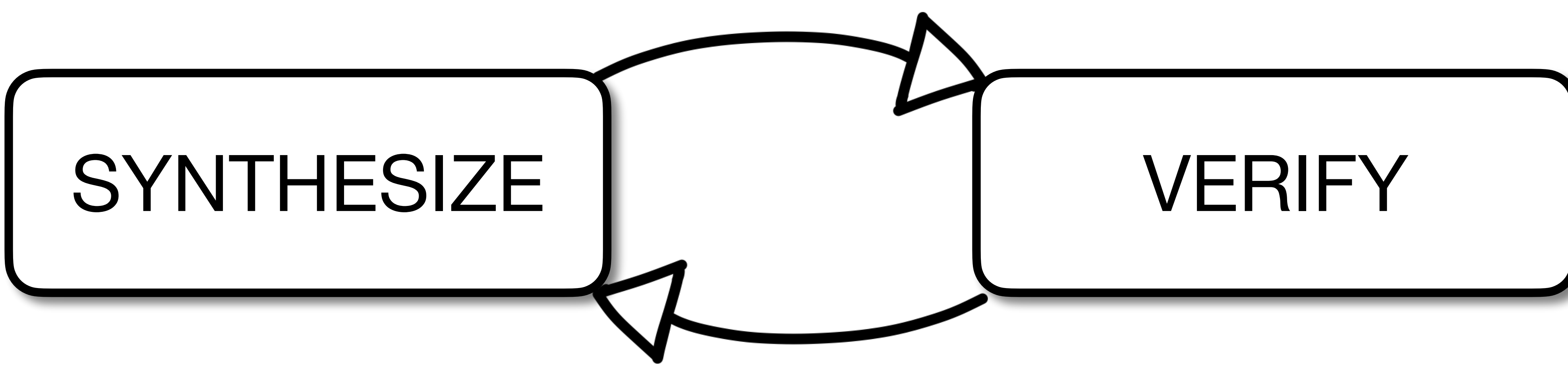
Counterexamples

[1] Sketching stencils, Solar-Lezama et al. PLDI 2007

What is formal synthesis?

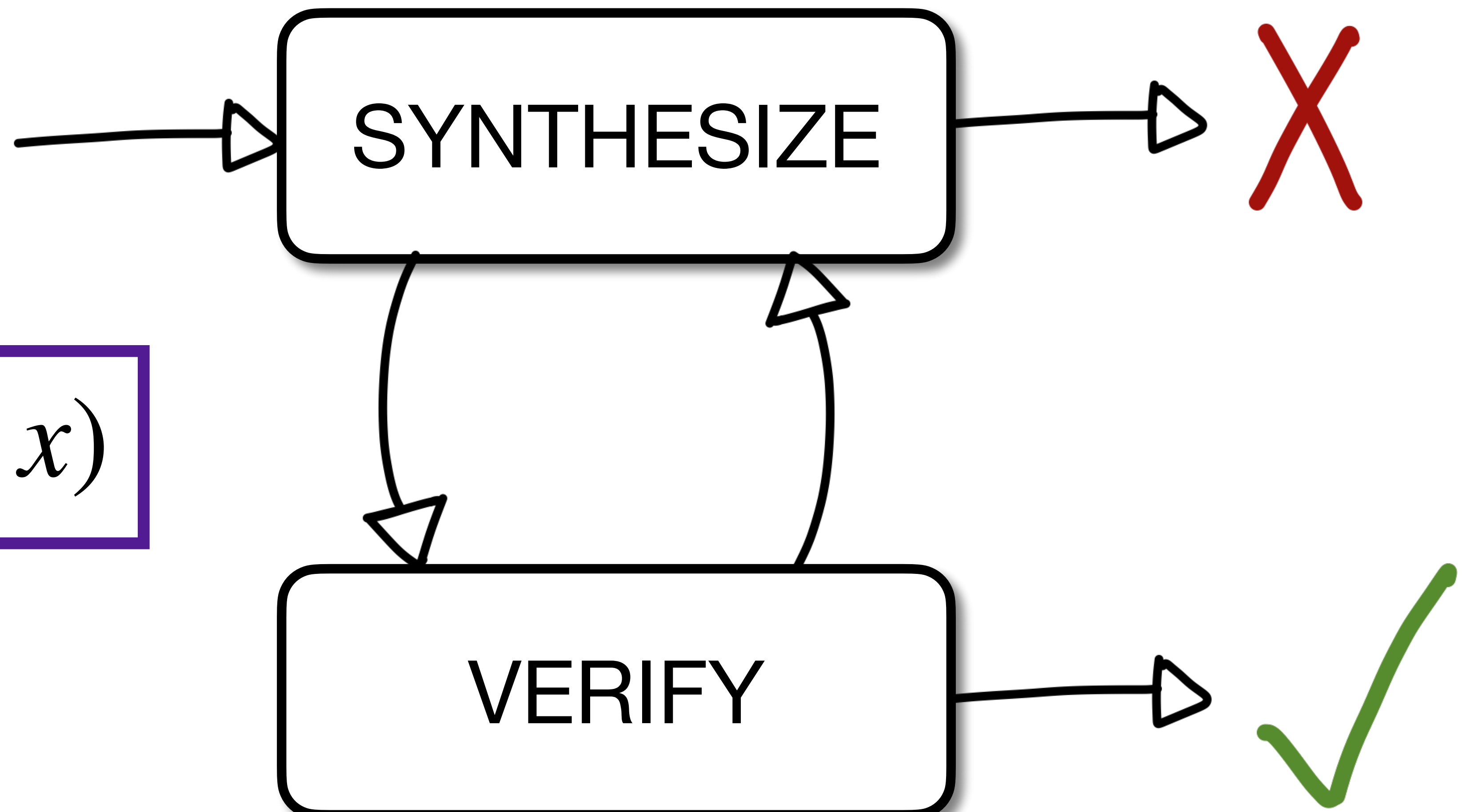
CounterExample Guided Inductive Synthesis (CEGIS)

- Logical specification σ.
- Synthesizes expressions/loop-free programs



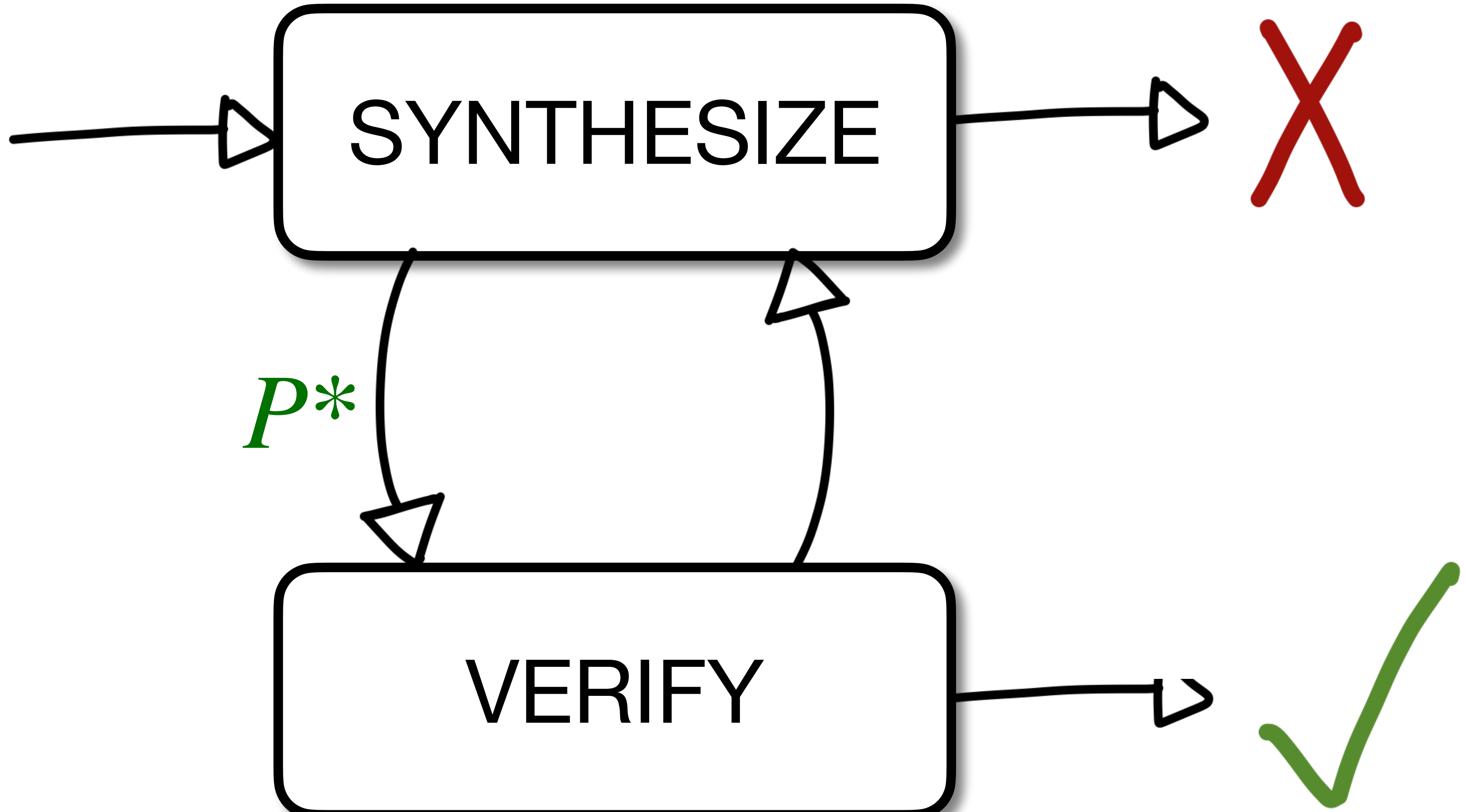
Counterexamples

CEGIS

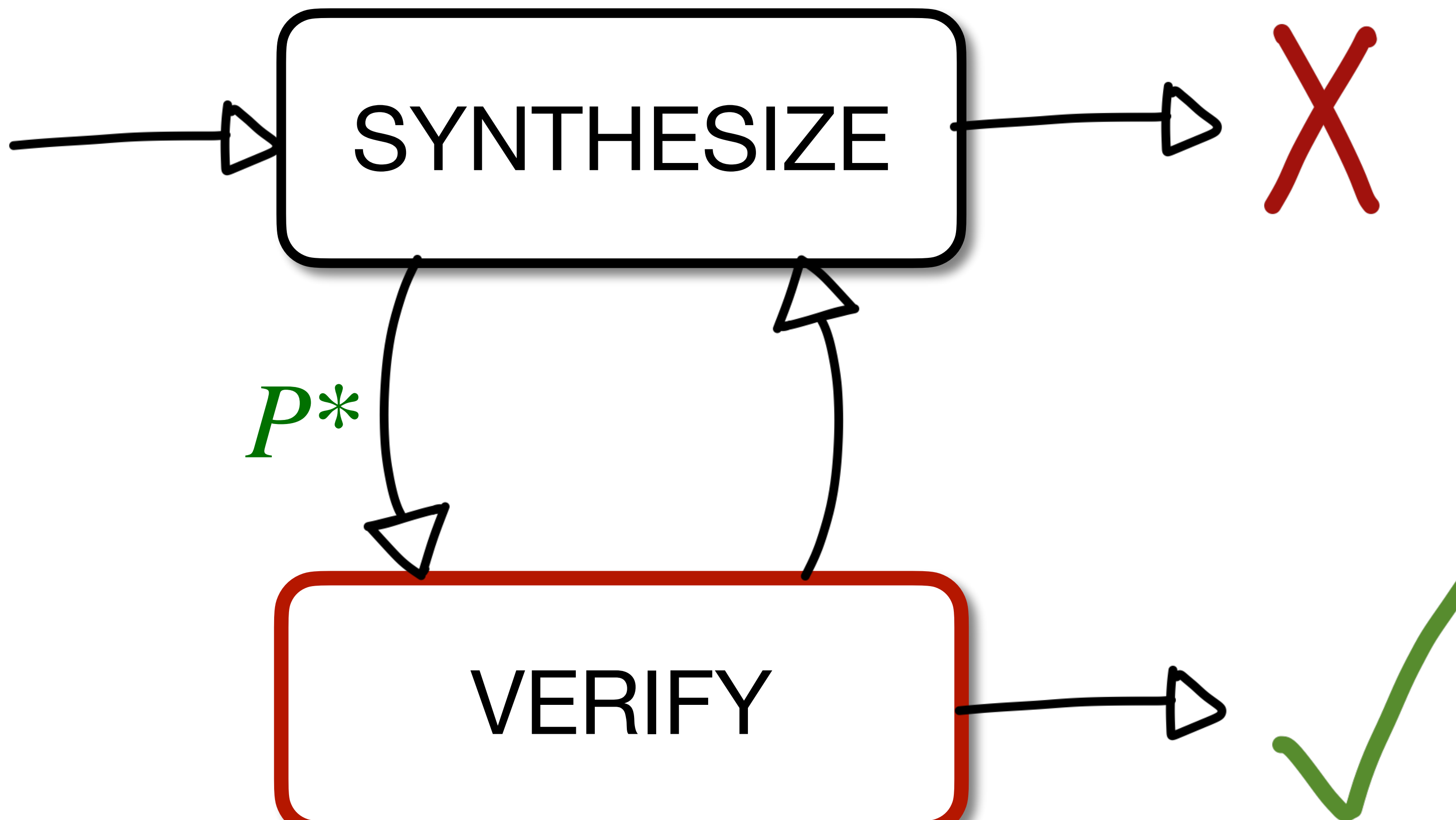


$$\boxed{\exists P \forall x \,.\, \sigma(P, x)}$$

CEGIS

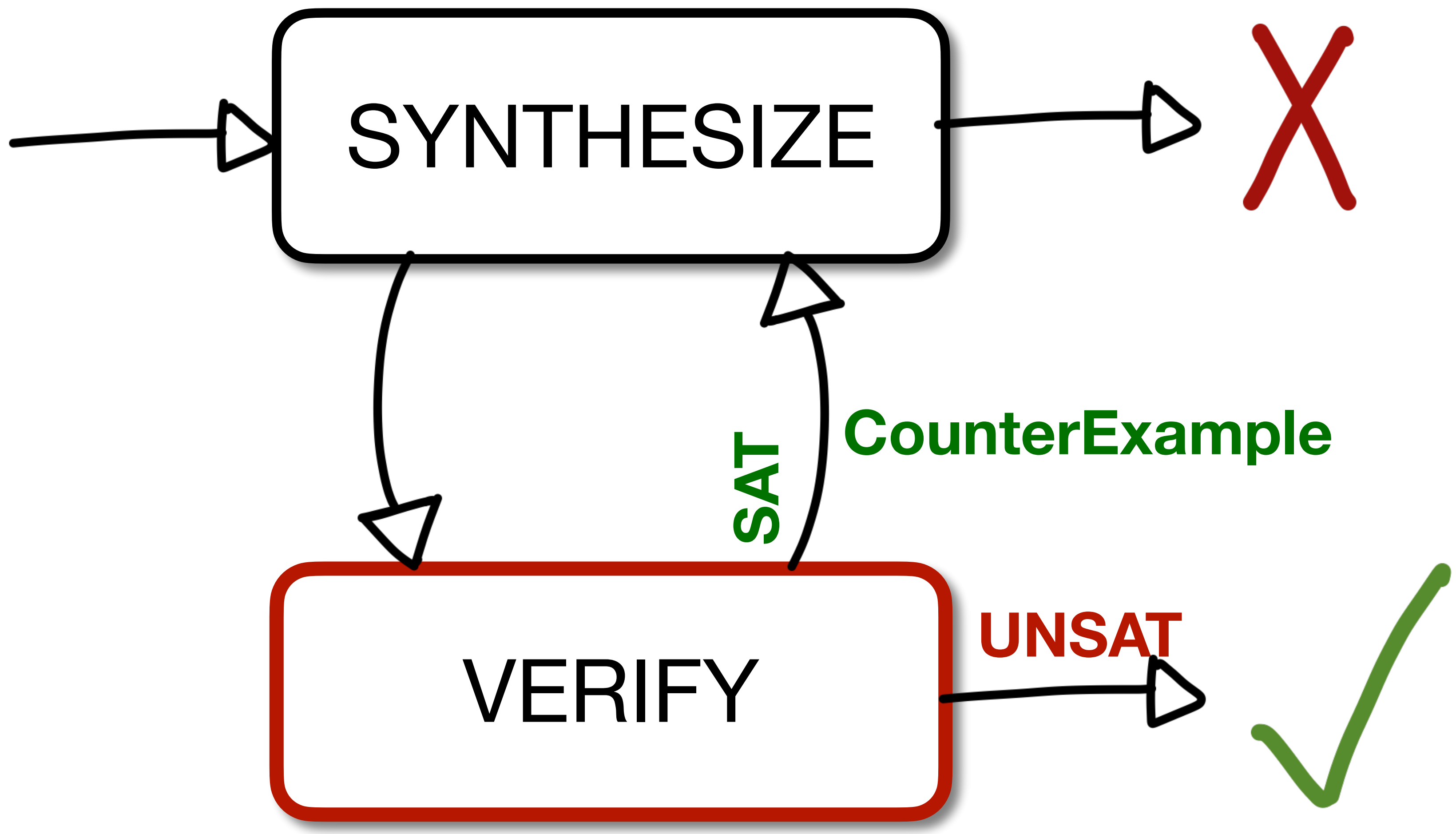


CEGIS



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

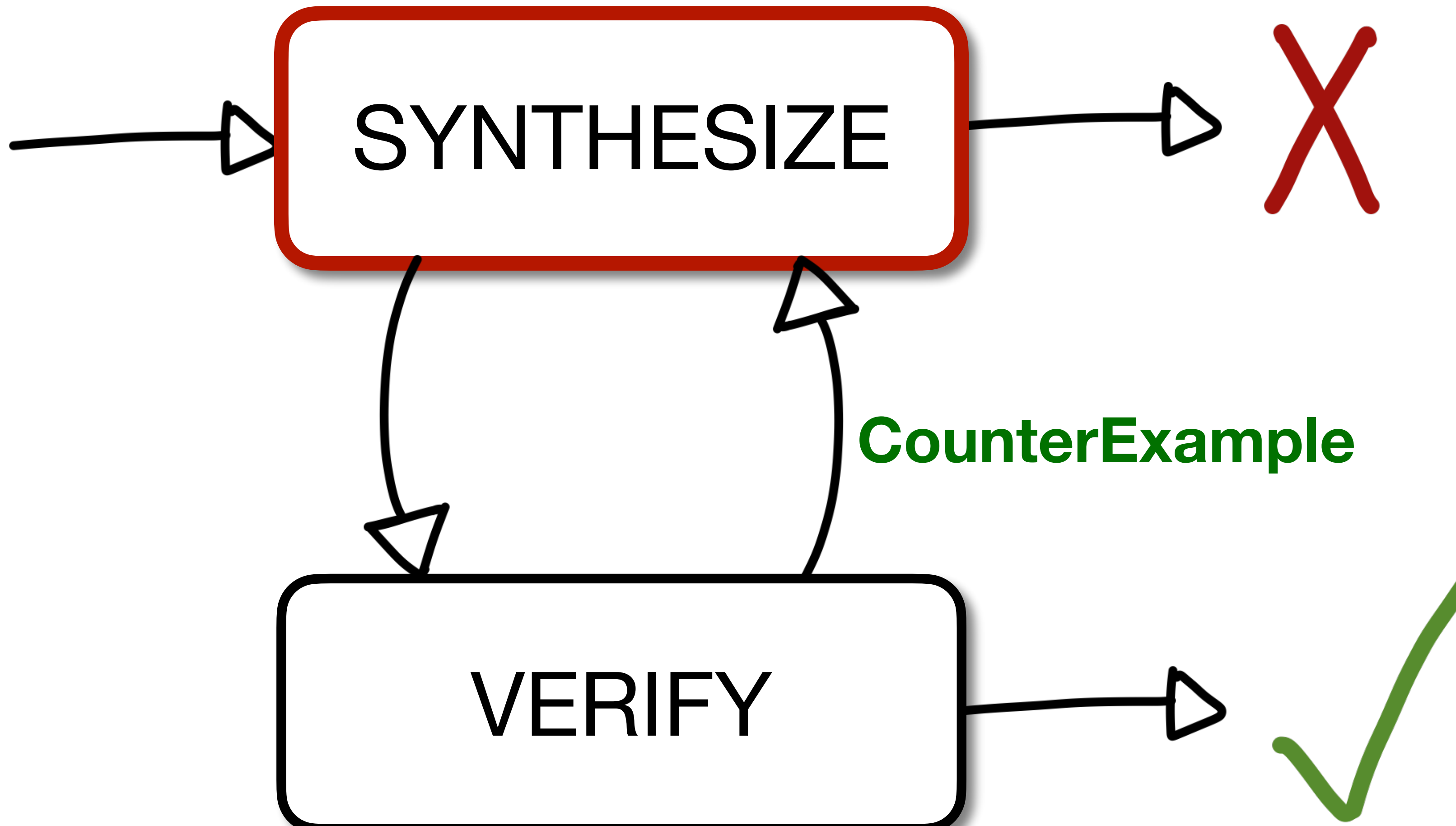
CEGIS



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

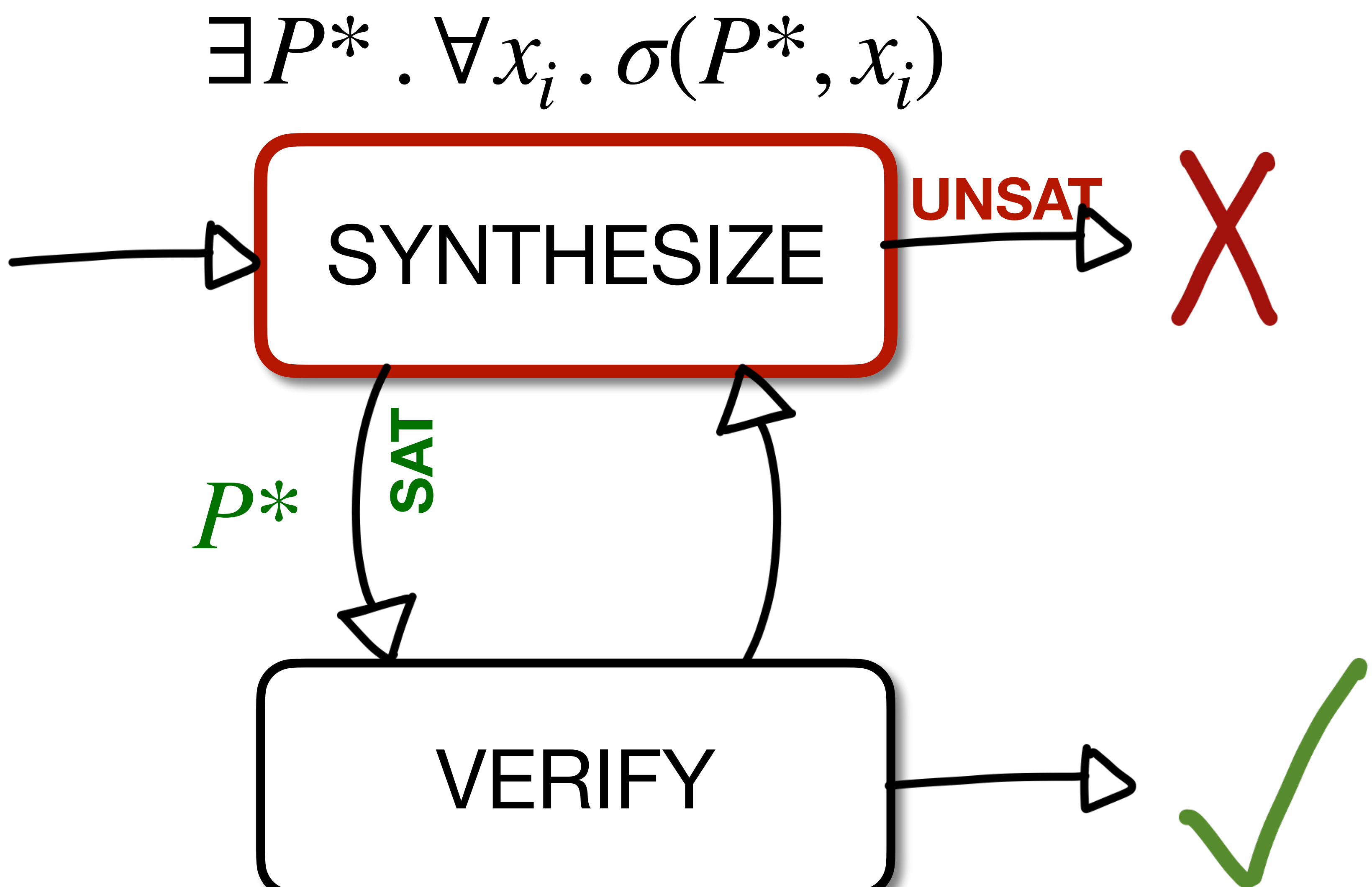
CEGIS

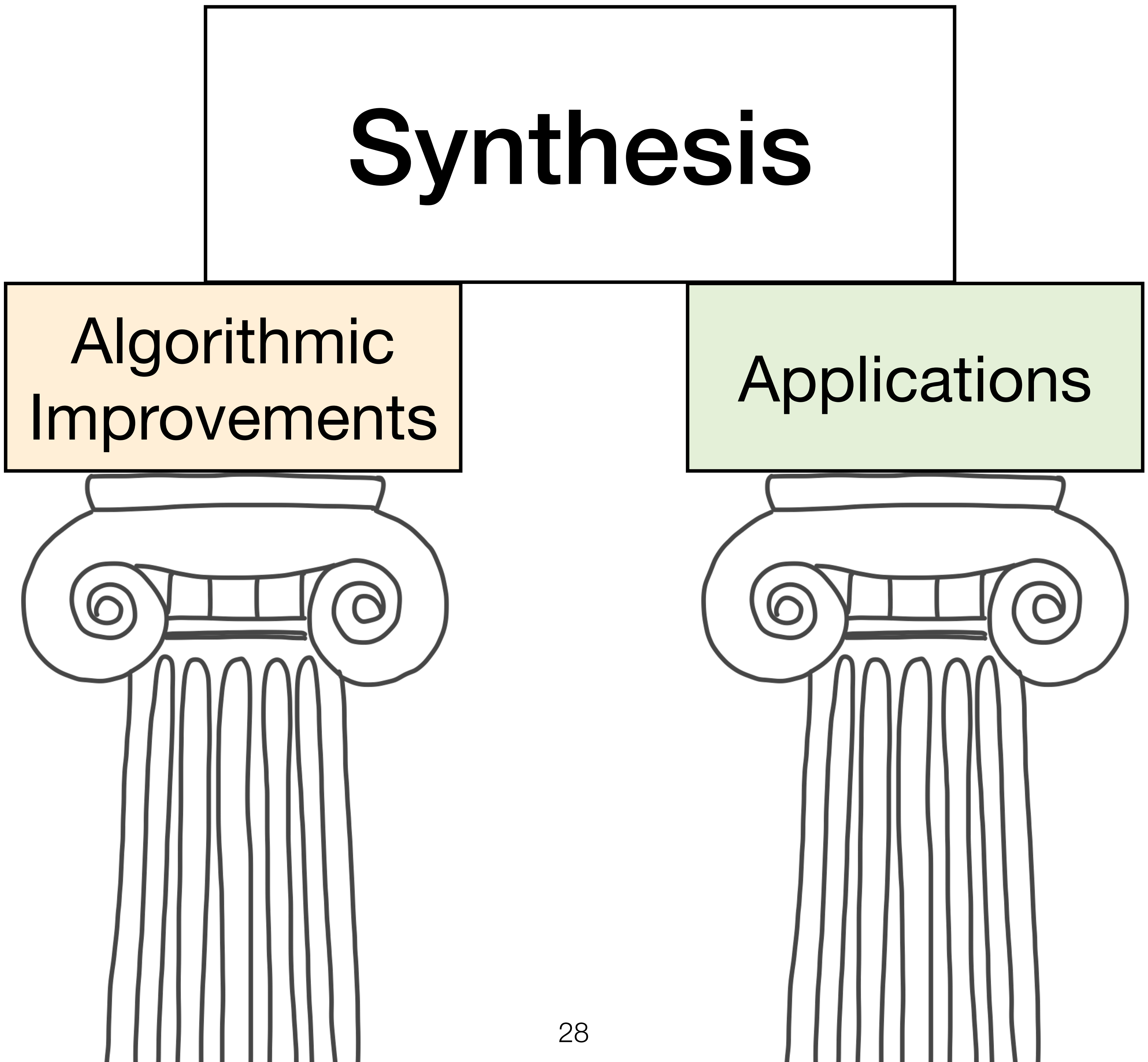
$$\exists P^* \,.\, \forall x_i \,.\, \sigma(P^*, x_i)$$

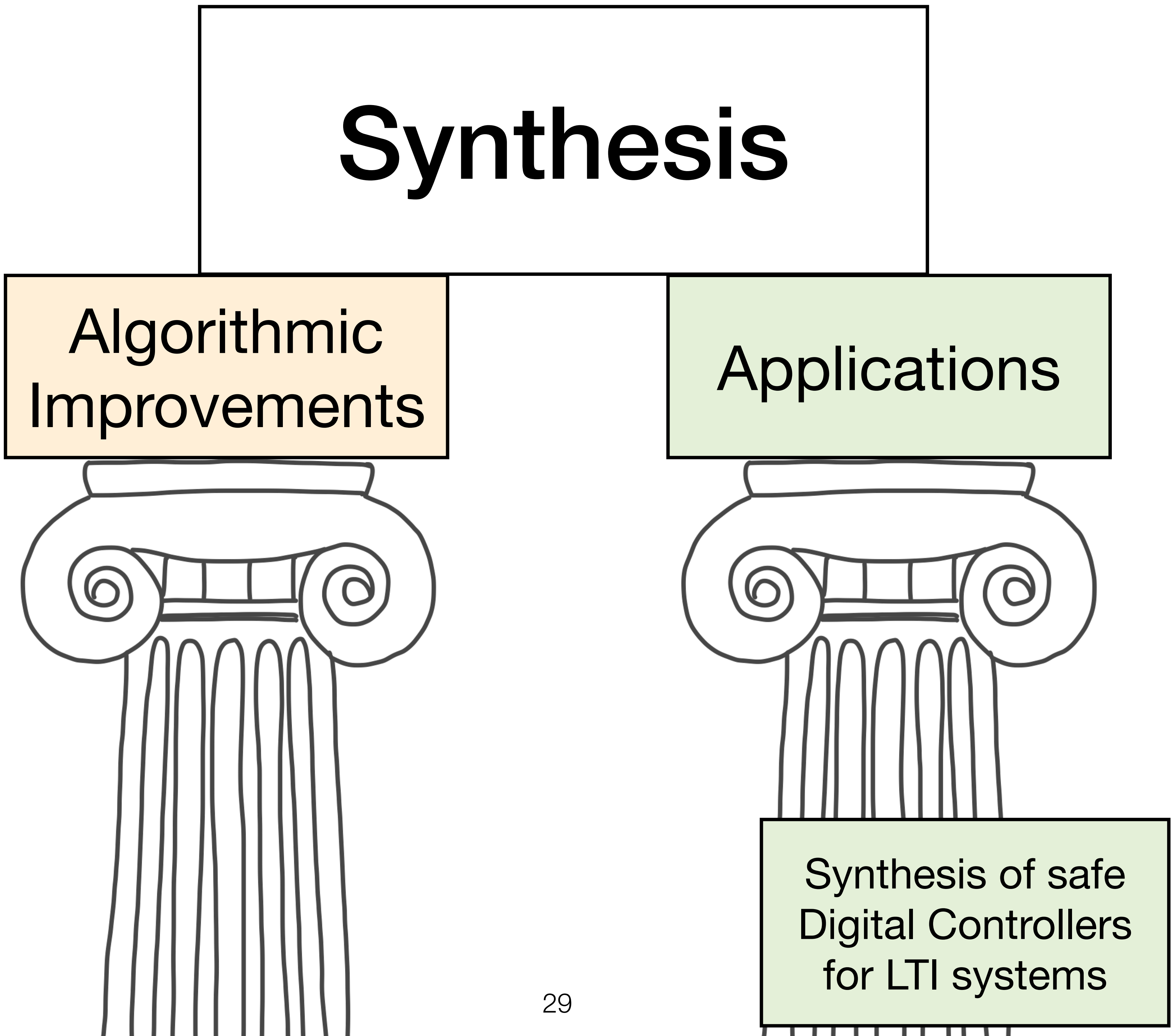


$$\exists x \,.\, \neg\sigma(P^*, x)$$

CEGIS




Synthesis
Algorithmic Improvements
Applications

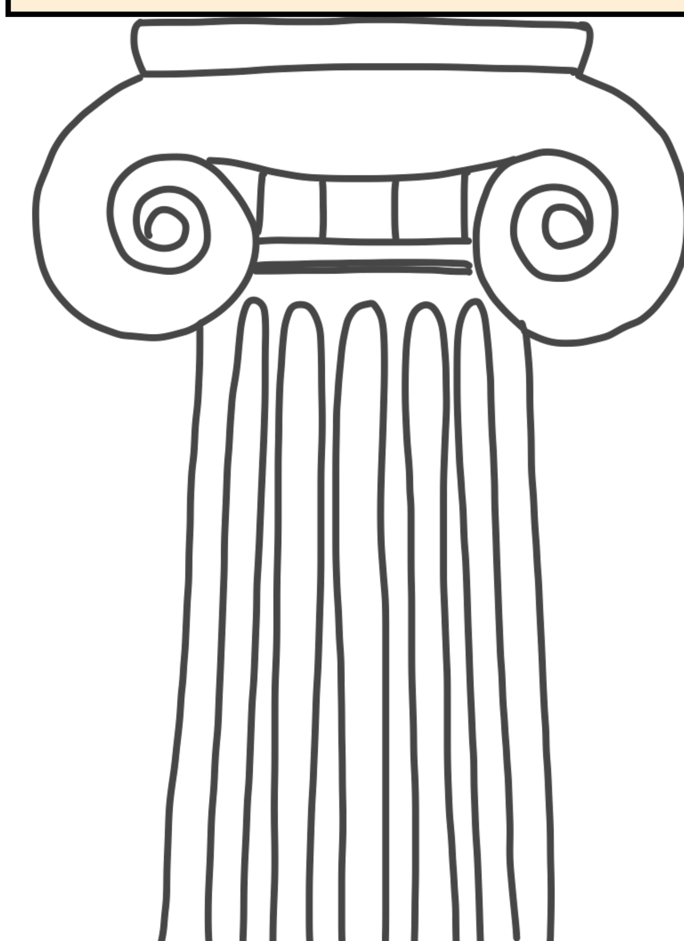


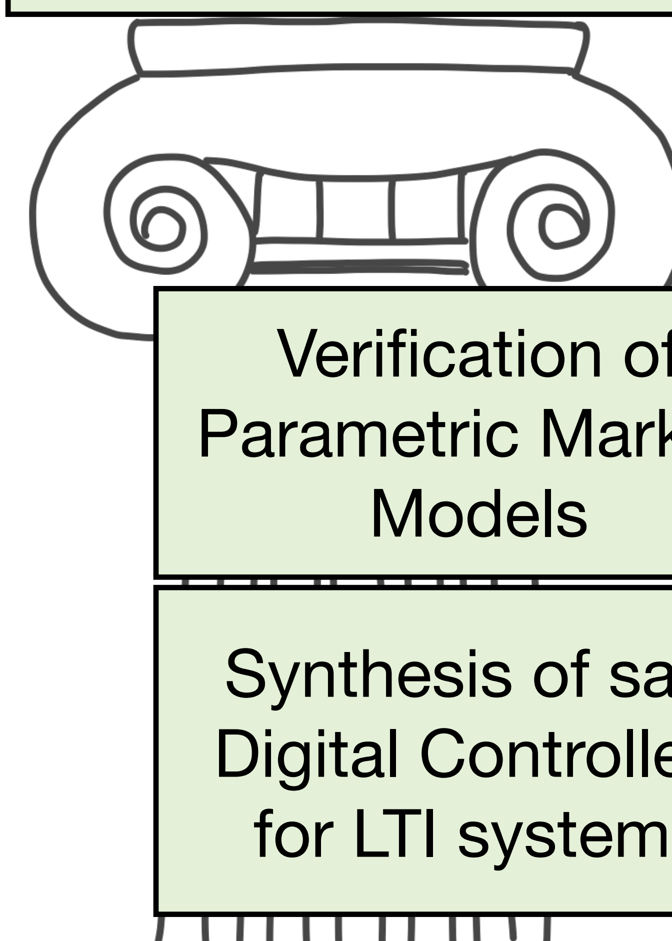
CAV 2017
Acta Inf. 2020
ASE 2017

Synthesis

Algorithmic Improvements

Applications





Verification of Parametric Markov Models

QEST 2016

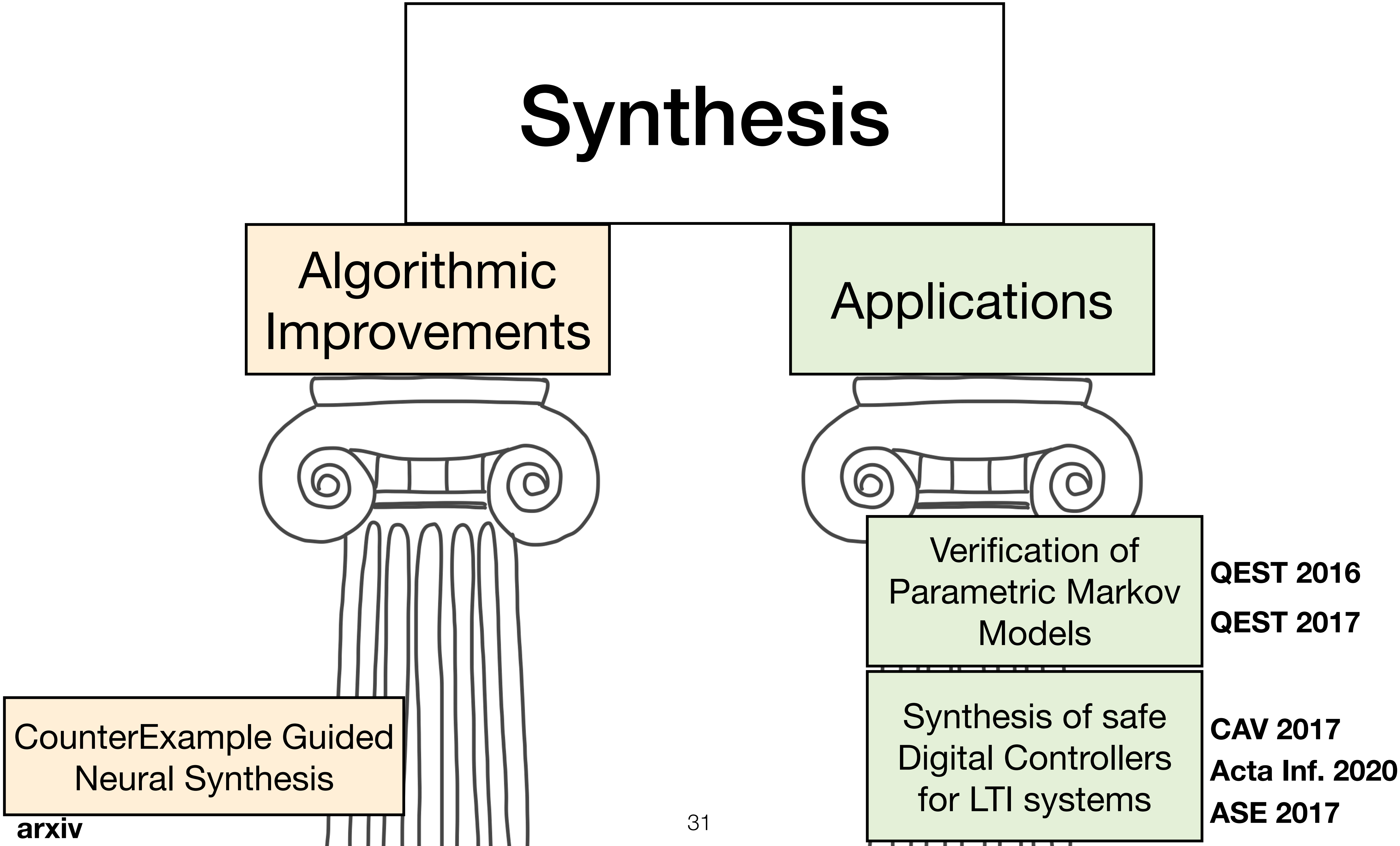
QEST 2017

Synthesis of safe Digital Controllers for LTI systems

CAV 2017

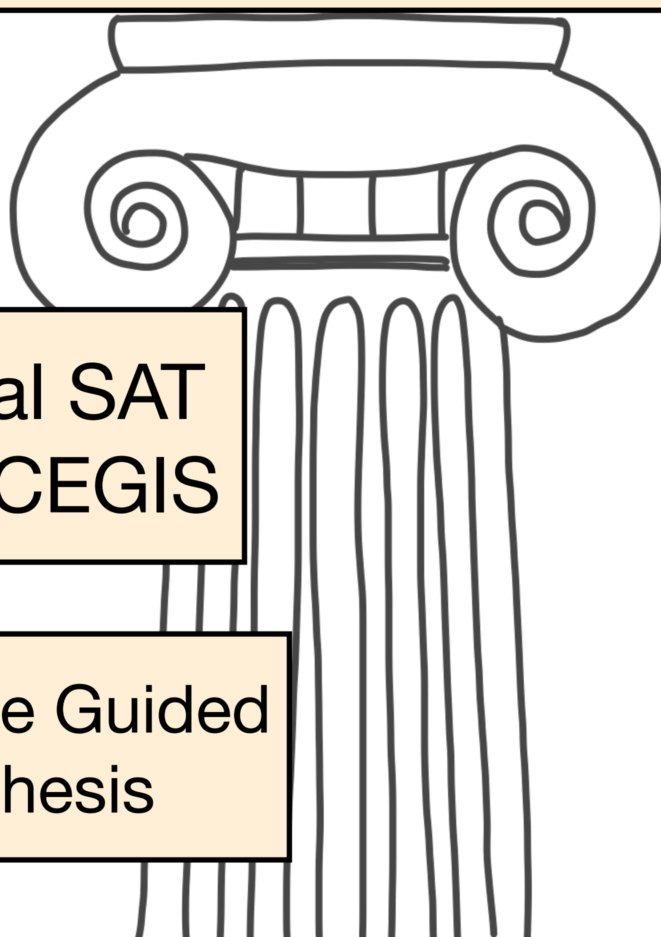
Acta Inf. 2020

ASE 2017


Synthesis
Algorithmic Improvements
Applications
Verification of Parametric Markov Models
QEST 2016
QEST 2017
Synthesis of safe Digital Controllers for LTI systems
CAV 2017
Acta Inf. 2020
ASE 2017
CounterExample Guided Neural Synthesis
arxiv

Synthesis

Algorithmic Improvements



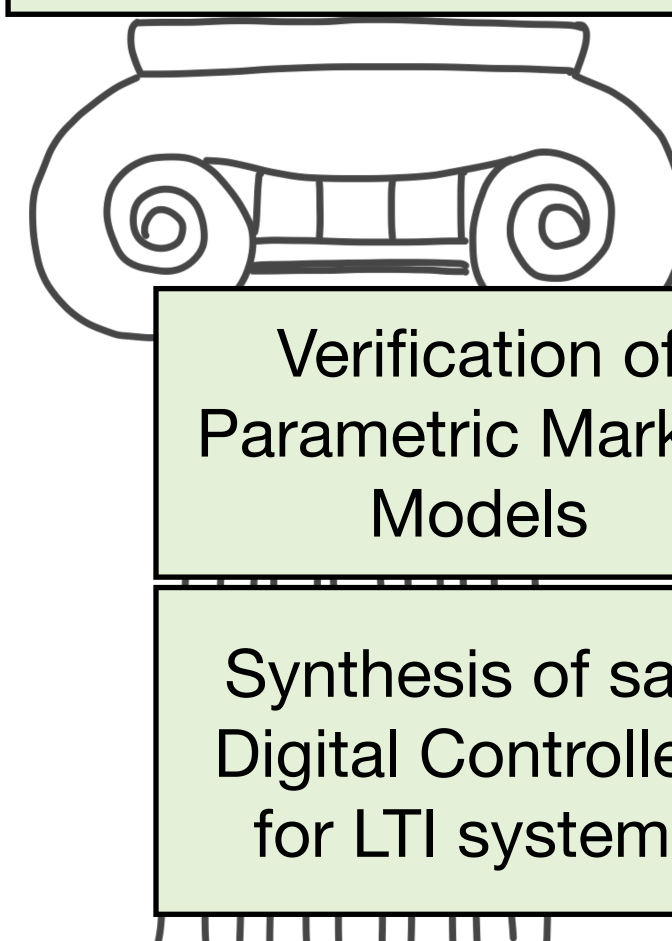
Thesis

Incremental SAT solving in CEGIS

CounterExample Guided Neural Synthesis

arxiv

Applications



Verification of Parametric Markov Models

QEST 2016

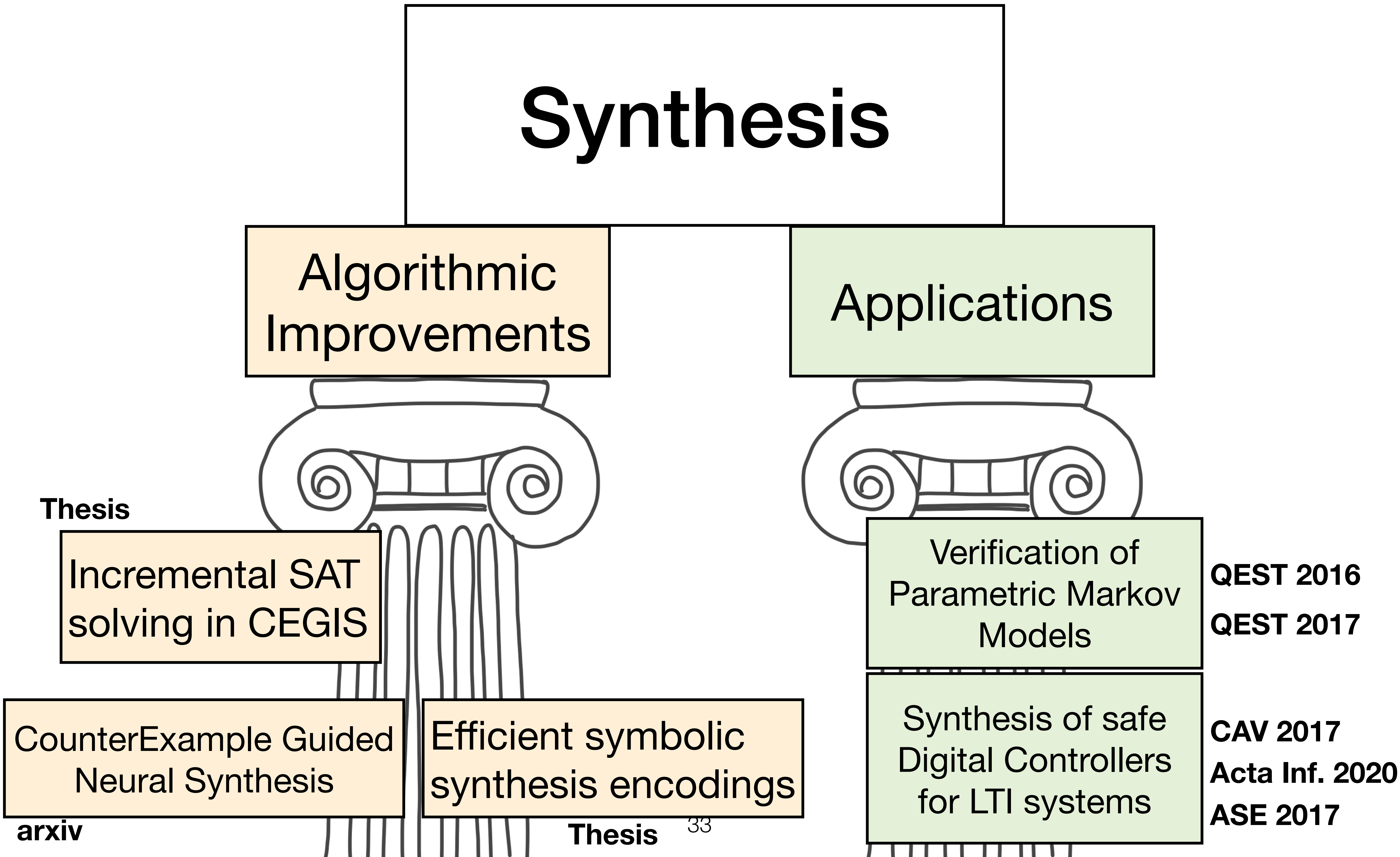
QEST 2017

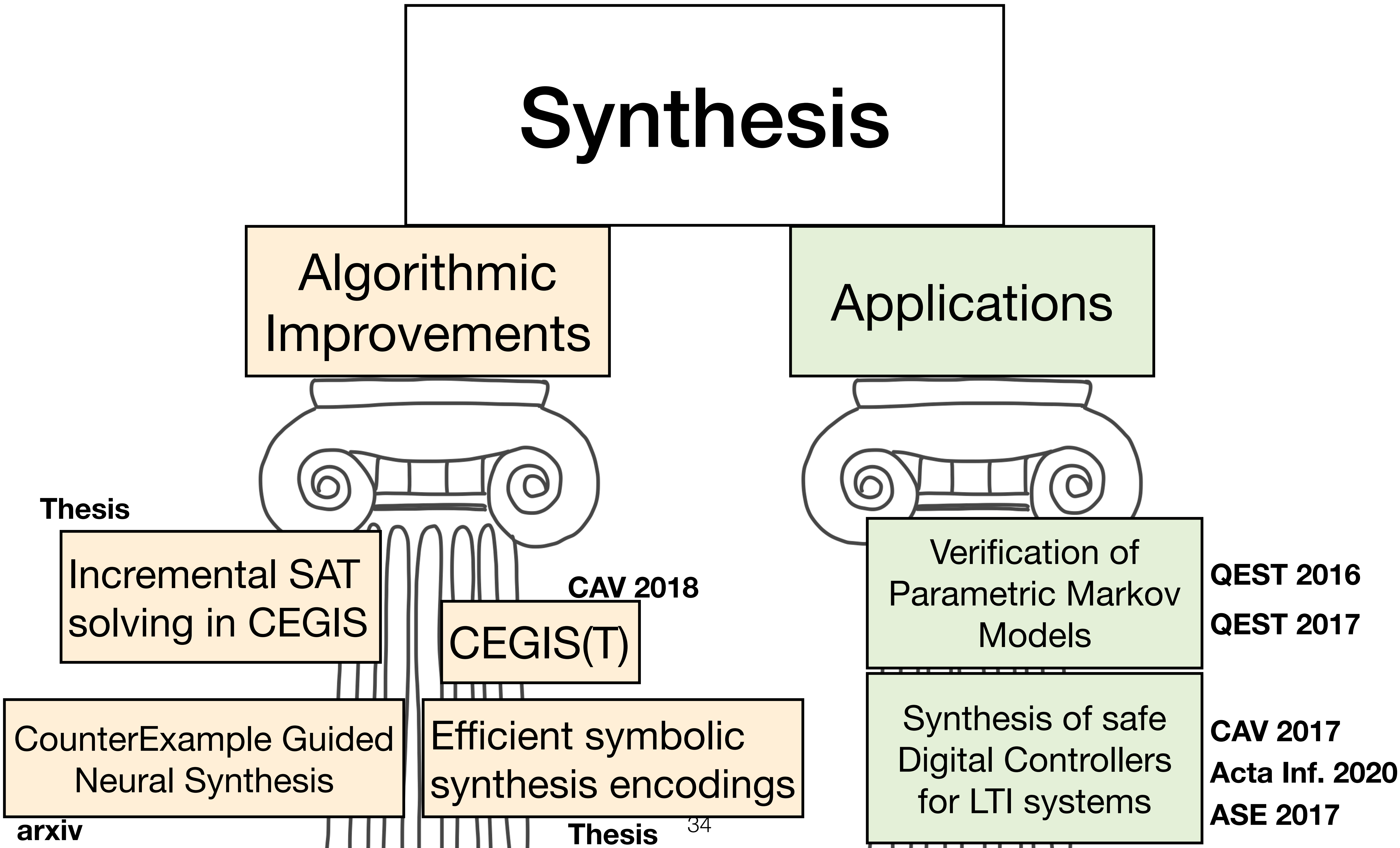
Synthesis of safe Digital Controllers for LTI systems

CAV 2017

Acta Inf. 2020

ASE 2017


Synthesis
Algorithmic Improvements
Applications
Thesis
Incremental SAT solving in CEGIS
CounterExample Guided Neural Synthesis
arxiv
Efficient symbolic synthesis encodings
Thesis
Verification of Parametric Markov Models
QEST 2016
QEST 2017
Synthesis of safe Digital Controllers for LTI systems
CAV 2017
Acta Inf. 2020
ASE 2017


Synthesis
Algorithmic Improvements
Applications
Thesis
Incremental SAT solving in CEGIS
CAV 2018
CEGIS(T)
CounterExample Guided Neural Synthesis
arxiv
Efficient symbolic synthesis encodings
Thesis
Verification of Parametric Markov Models
QEST 2016
QEST 2017
Synthesis of safe Digital Controllers for LTI systems
CAV 2017
Acta Inf. 2020
ASE 2017

CounterExample Guided Inductive Synthesis Modulo Theories

CAV 2018

Extends CEGIS framework to

- verify generalized candidate solutions and
- return more general counterexamples.

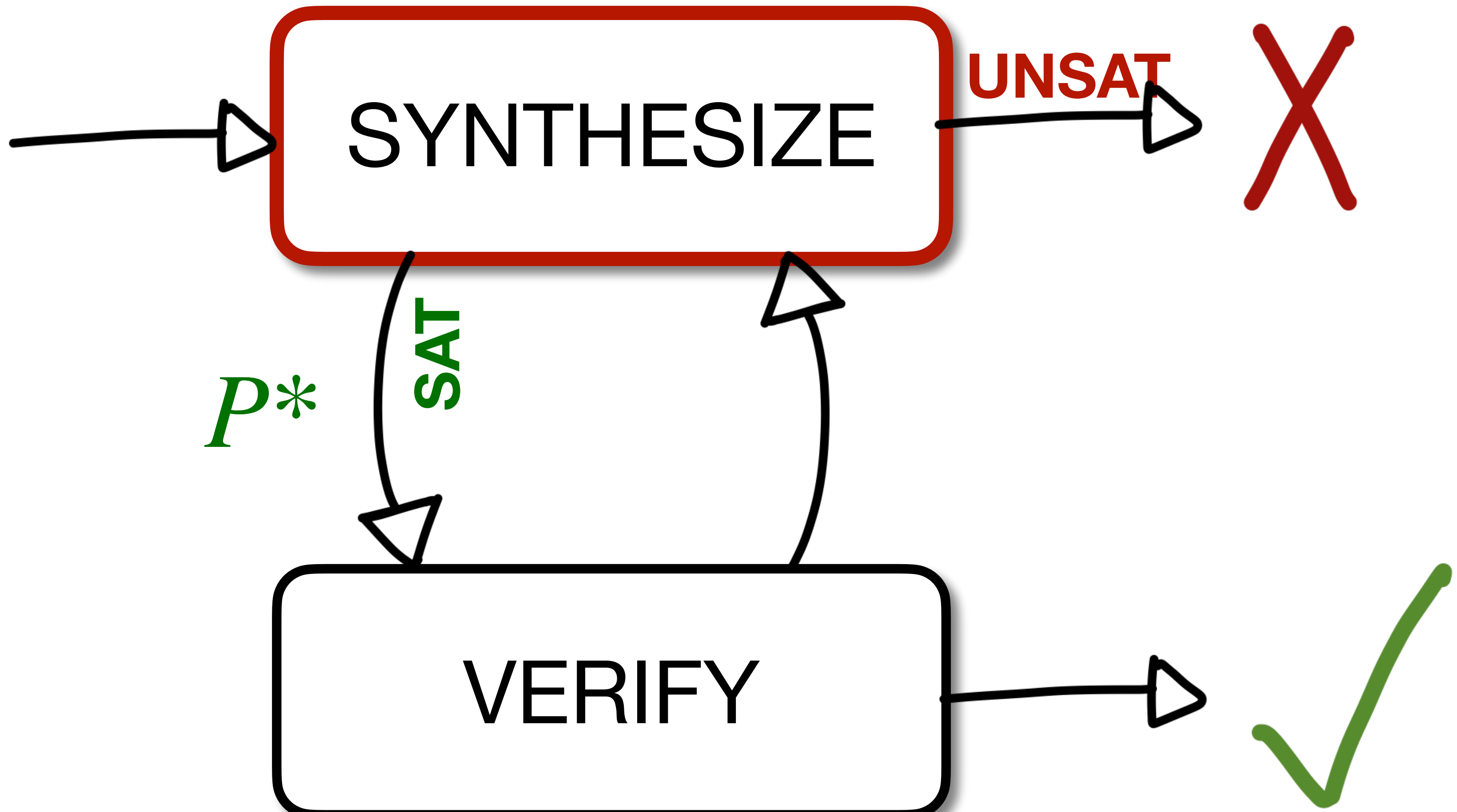
CEGIS(T) is able to synthesize programs containing arbitrary constants that elude other solvers.

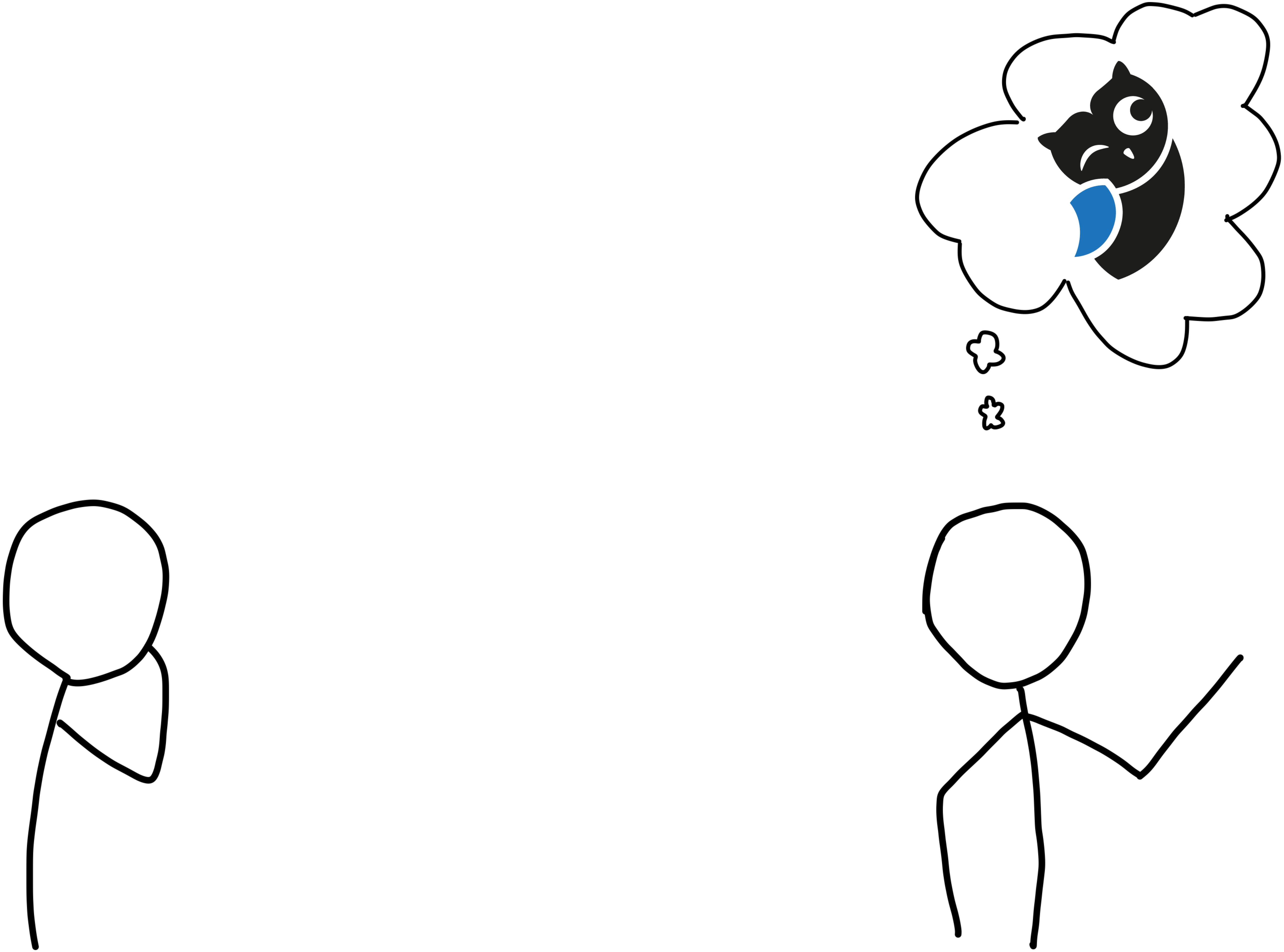
- Do not appear in the synthesis problem

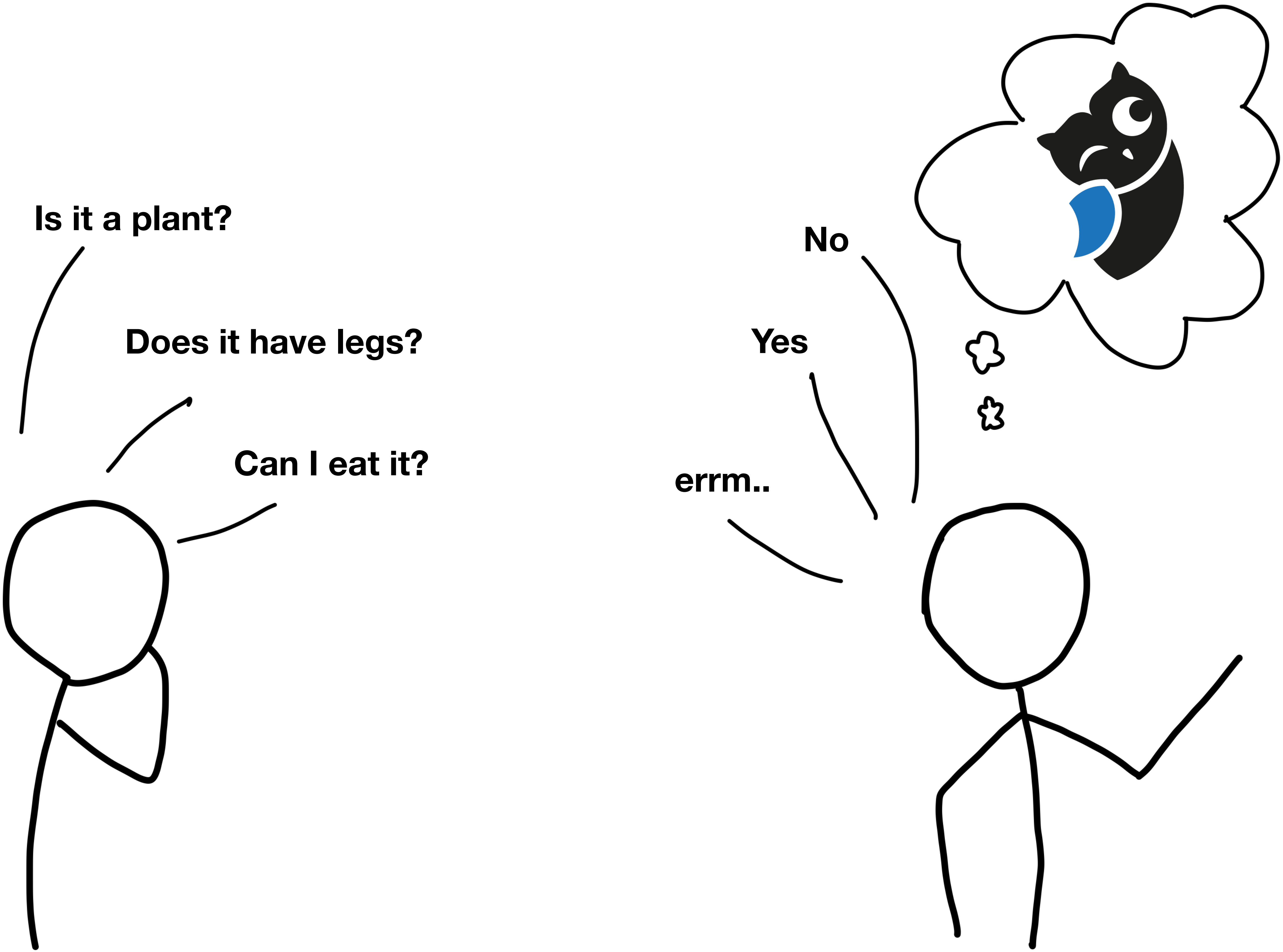
- Not 0, 1 or FFFF

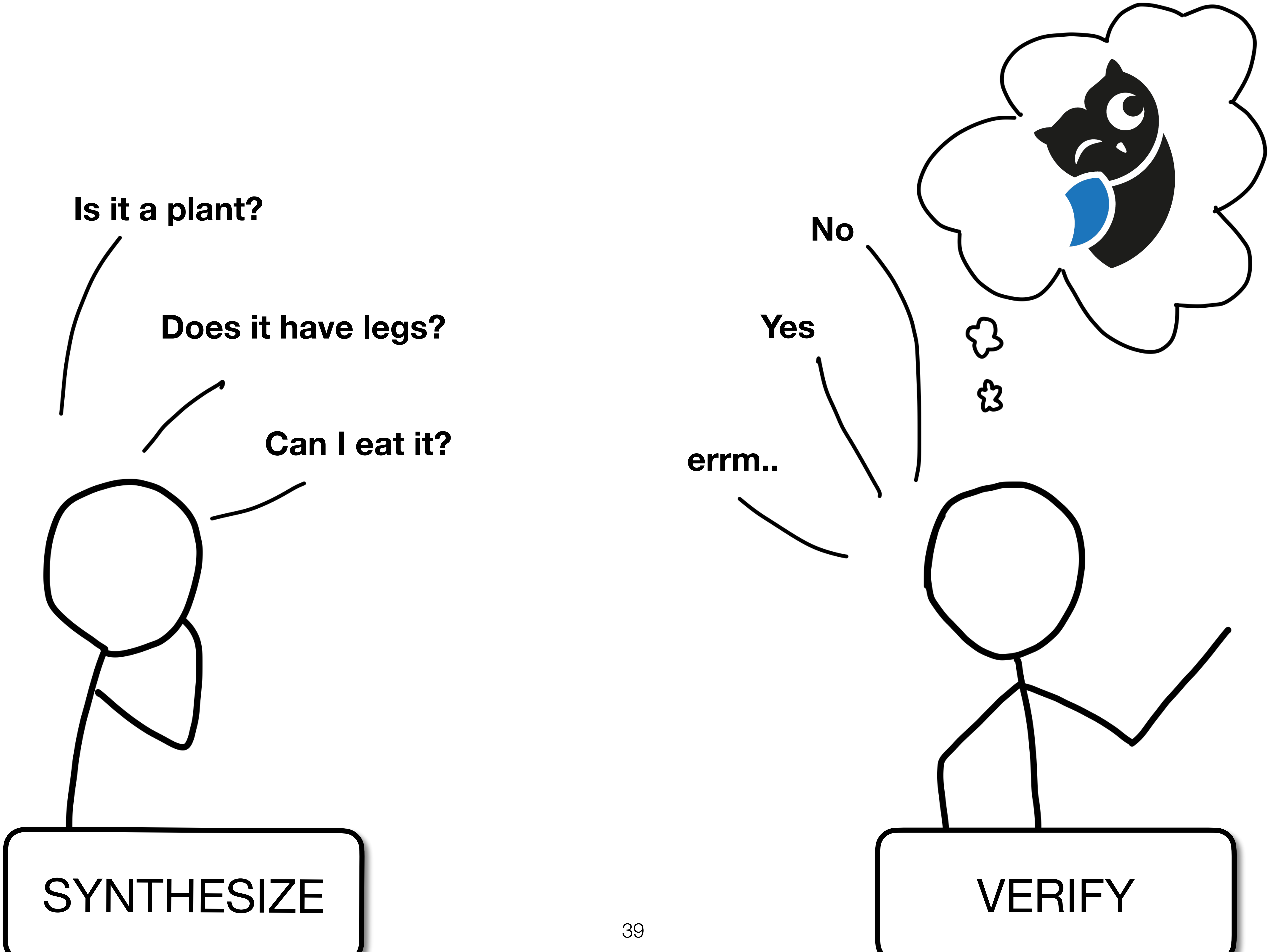
CEGIS

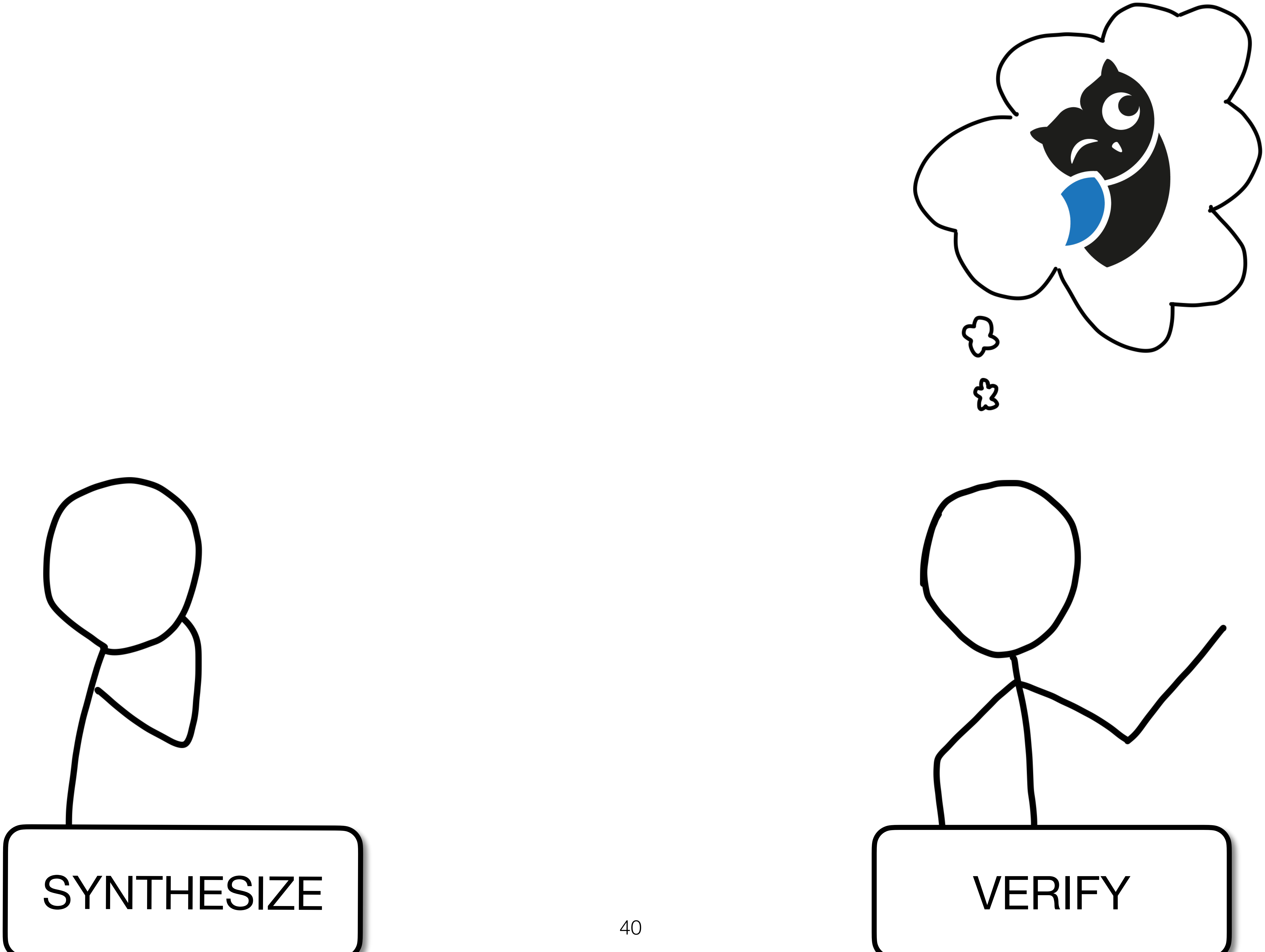
$$\exists P^* . \forall x_i . \sigma(P^*, x_i)$$

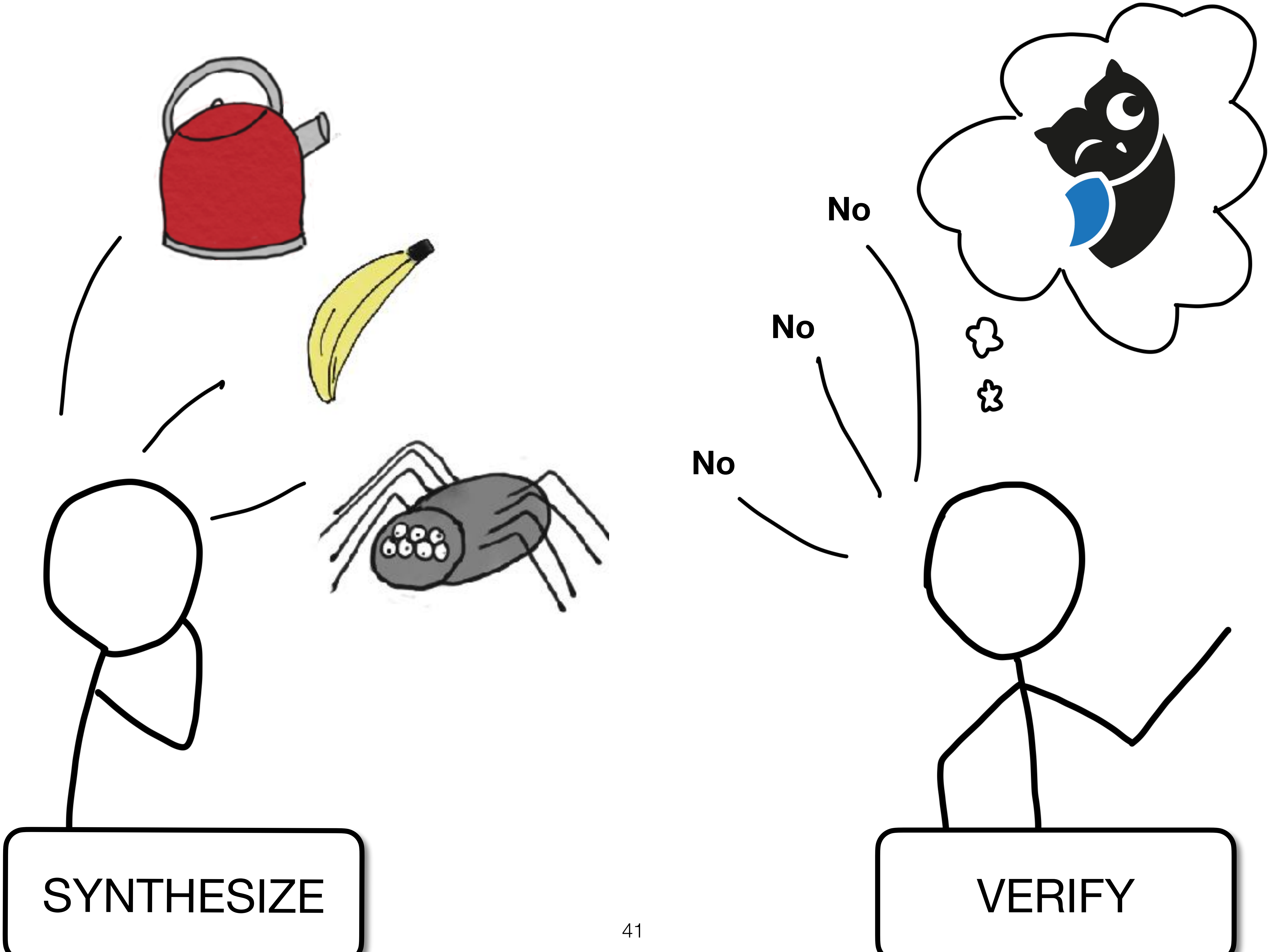


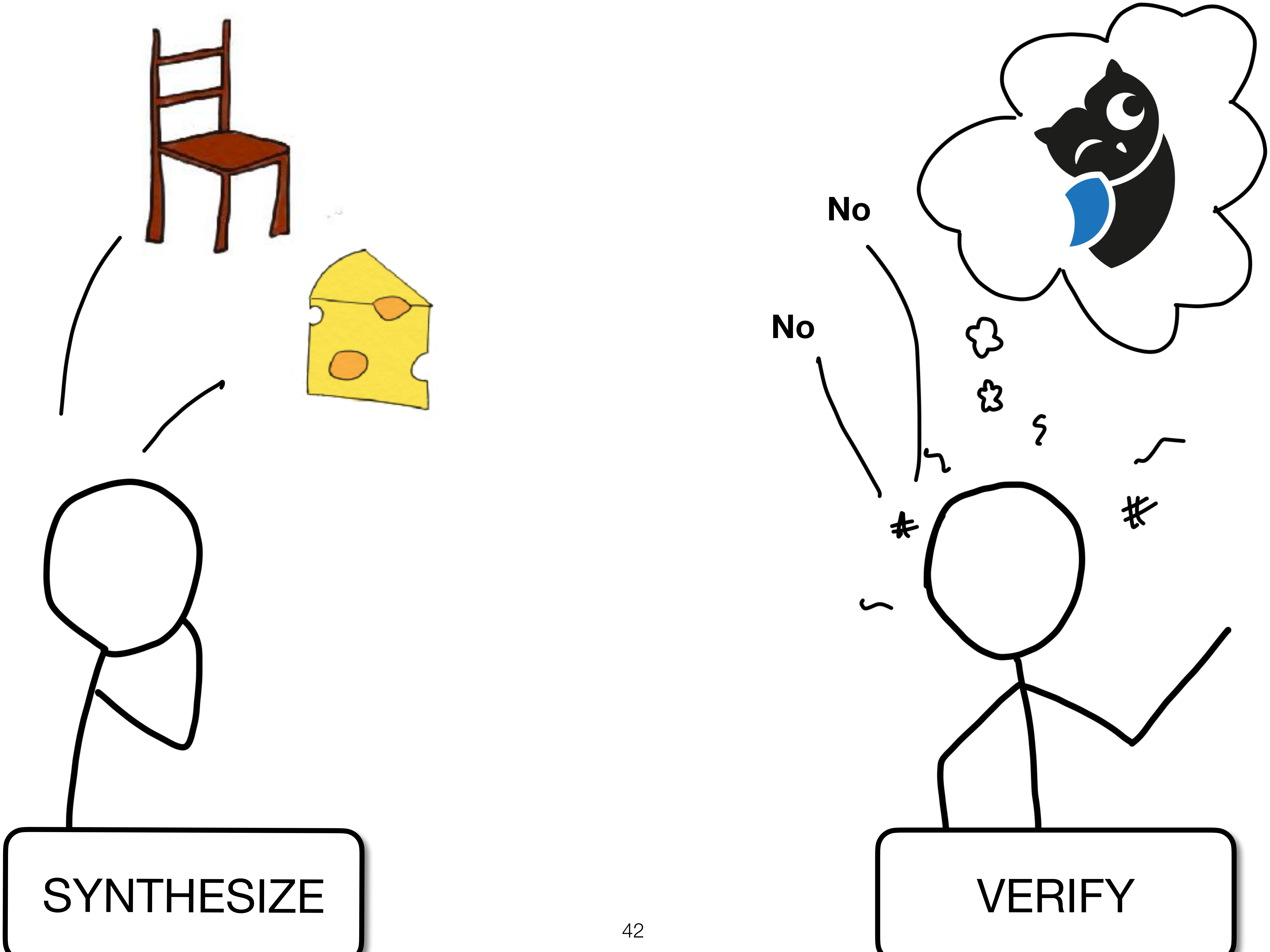


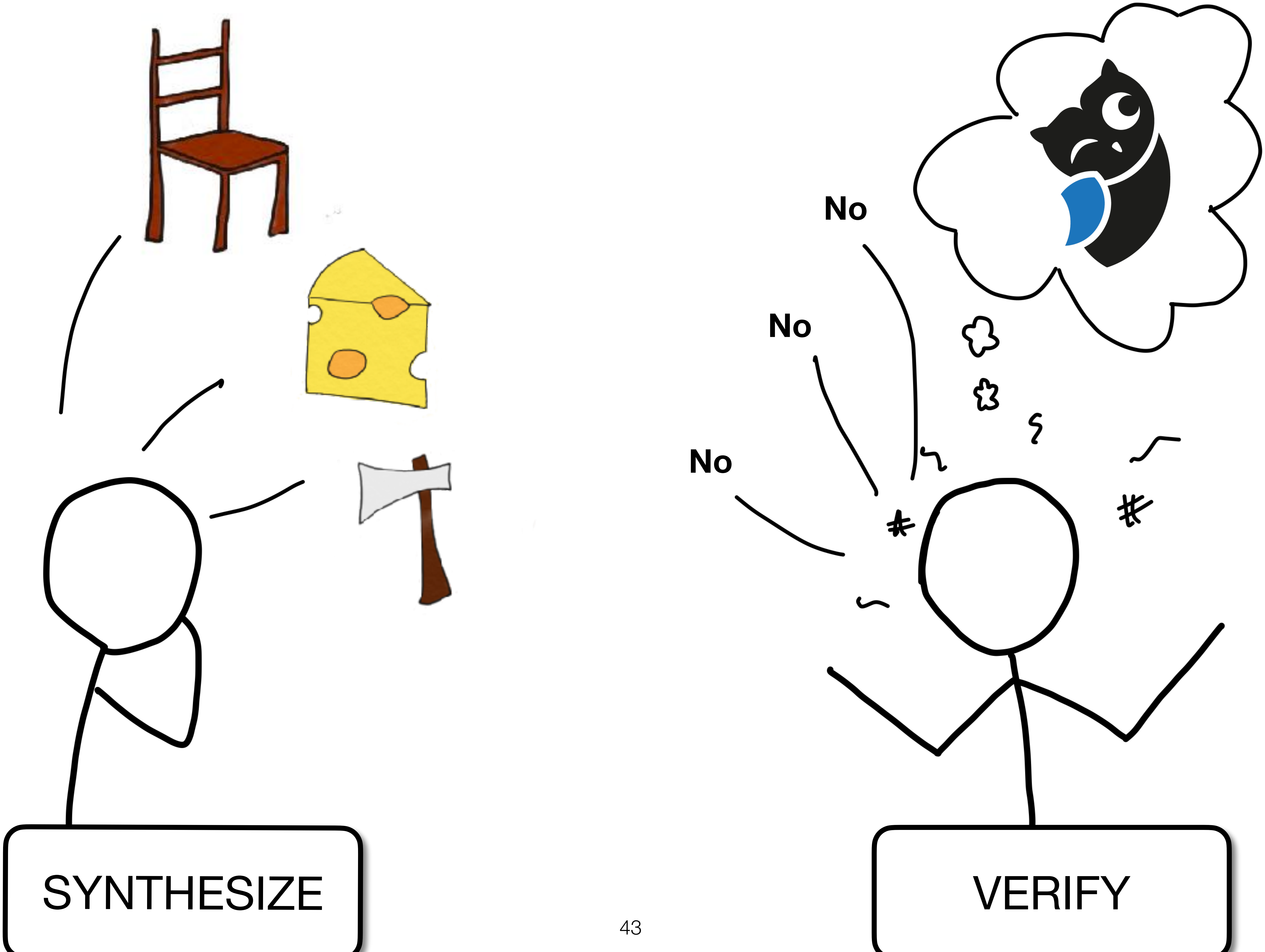

Is it a plant?
No
Does it have legs?
Yes
Can I eat it?
errm..


Is it a plant?
Does it have legs?
Can I eat it?
No
Yes
errm..
SYNTHESIZE
VERIFY


SYNTHESIZE
VERIFY


No
No
No
SYNTHESIZE
VERIFY


No
No
SYNTHESIZE
VERIFY


No
No
No
SYNTHESIZE
VERIFY

Safety Invariant

```
int x = 5;

while ( x < 1000)
  x++;

assert( 5 < x && x < 1005)
```

$$init(x) \iff x = 0$$

$$trans(x, x') \iff x' = x + 1$$

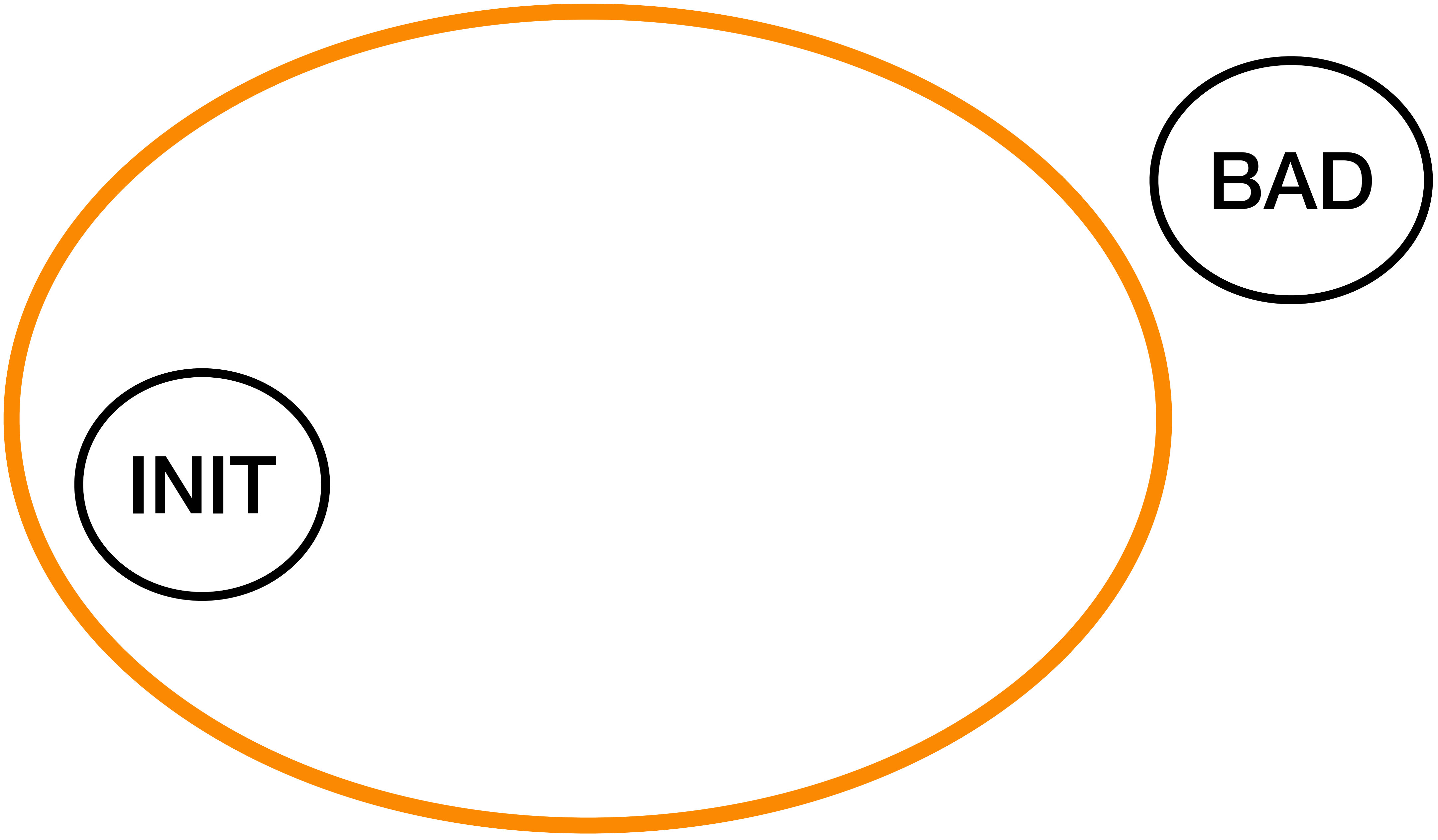
find $inv(x)$ **such that:**

$$init(x) \implies inv(x)$$

$$inv(x) \land (x < 1000) \land trans(x, x') \implies inv(x')$$

$$inv(x) \land \neg(x < 1000) \implies (x < 1005) \land (x > 5)$$

Safety Invariant



Safety Invariant

```
int x = 5;

while ( x < 1000)
  x++;

assert( 5 < x && x < 1005)
```

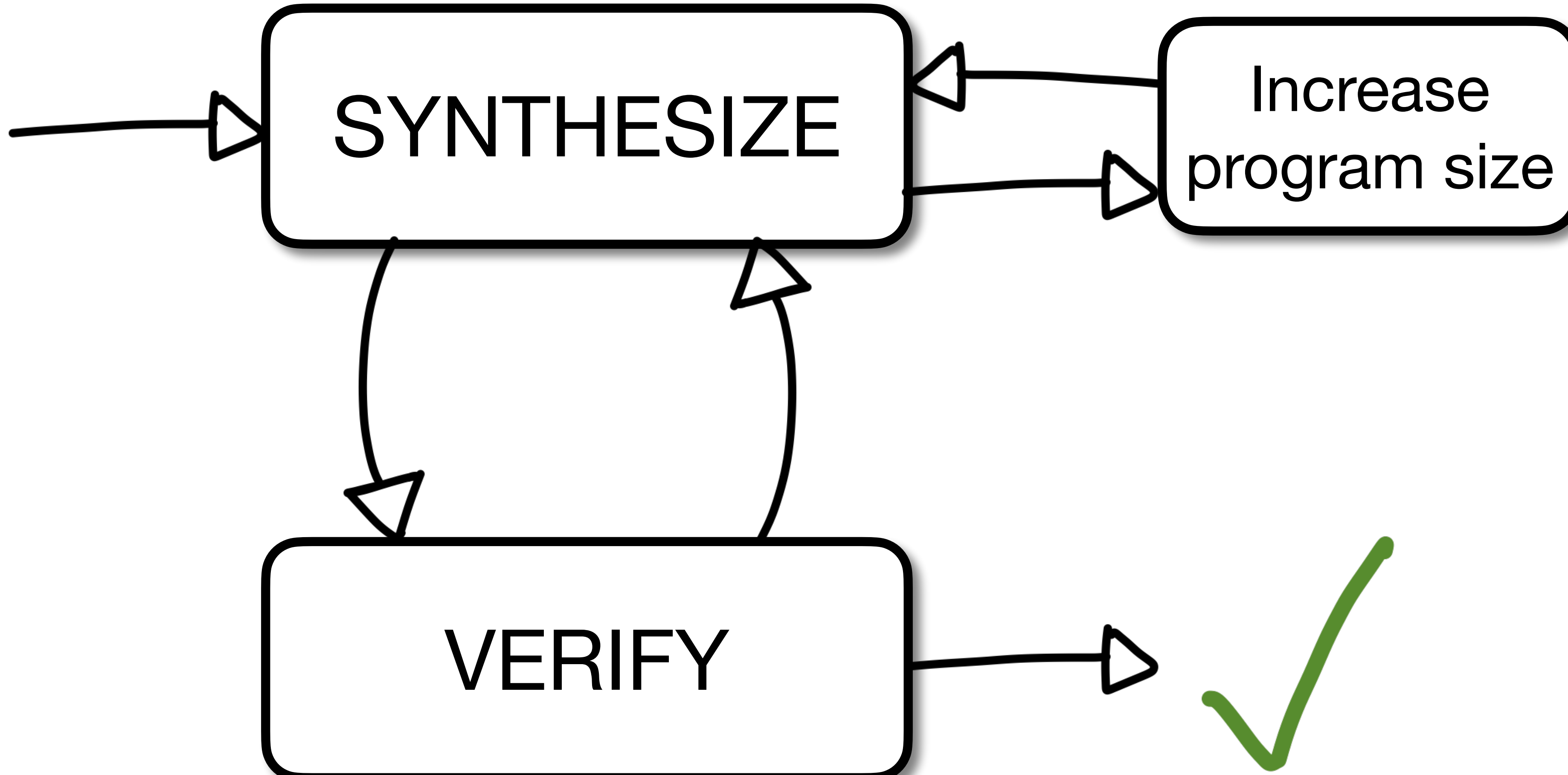
$$init(x) \iff x = 0$$

$$trans(x, x') \iff x' = x + 1$$

$$\boxed{inv(x) = (4 < x) \wedge (x < 1003)}$$

Synthesis Encoding

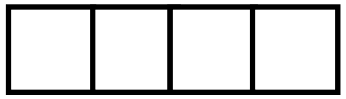
$$\exists \boxed{P^*} . \forall x_i . \sigma(P^*, x_i)$$



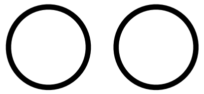
Synthesis Encoding

$$\exists P^* . \forall x_i . \sigma(P^*, x_i)$$

$$C_0 ::= 0000 \,|\, 0001 \,|\, \ldots \,|\, 1111$$



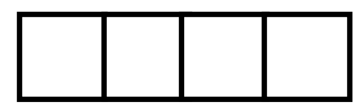
$$P_1 ::= arg_1 \,|\, arg_2 \,|\, C_0$$



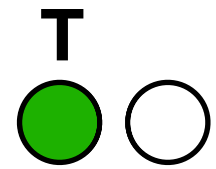
Synthesis Encoding

$$\exists P^* . \forall x_i . \sigma(P^*, x_i)$$

$$C_0 ::= 0000 \,|\, 0001 \,|\, \ldots \,|\, 1111$$



$$P_1 ::= arg_1 \,|\, arg_2 \,|\, C_0$$

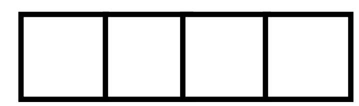


$$P_1 = arg_1$$

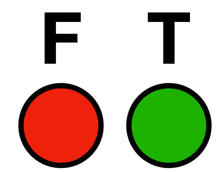
Synthesis Encoding

$$\exists \boxed{P*} \, . \, \forall x_i \, . \, \sigma(P*, x_i)$$

$C_0 ::= 0000 \,|\, 0001 \,|\, \ldots \,|\, 1111$



$P_1 ::= arg_1 \,|\, \boxed{arg_2} \,|\, C_0$

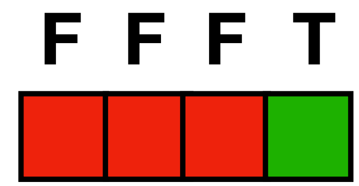


$$\boxed{P_1 = arg_2}$$

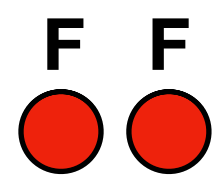
Synthesis Encoding

$$\exists P^* . \forall x_i . \sigma(P^*, x_i)$$

$$C_0 ::= 0000 \mid 0001 \mid \ldots \mid 1111$$



$$P_1 ::= arg_1 \mid arg_2 \mid C_0$$



$$P_1 = 1$$

Synthesis Encoding

$$\exists \boxed{P*} \,.\, \forall x_i \,.\, \sigma(P*, x_i)$$

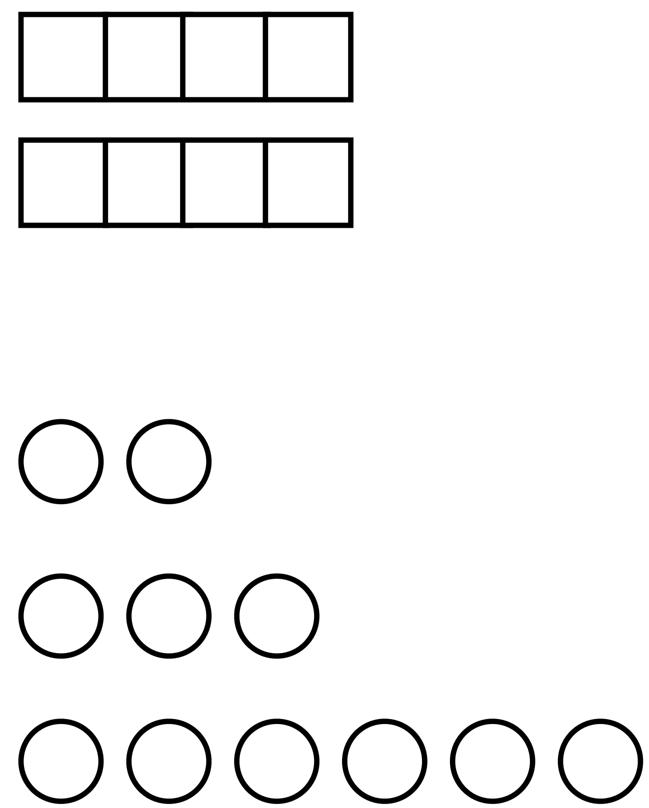
$$C_0 ::= 0000 \,|\, 0001 \,|\, \ldots \,|\, 1111$$

$$C_1 ::= 0000 \,|\, 0001 \,|\, \ldots \,|\, 1111$$

$$P_1 ::= arg_1 \,|\, arg_2 \,|\, C_0$$

$$P_2 ::= P_1 + P_1 \,|\, arg1 \,|\, arg_2 \,|\, C_1$$

$$P_3 ::= P_2 + P_1 \,|\, P_2 - P_1 \,|\, \ldots$$



Synthesis Encoding

$$\exists \boxed{P^*} \,.\, \forall x_i \,.\, \sigma(P^*, x_i)$$

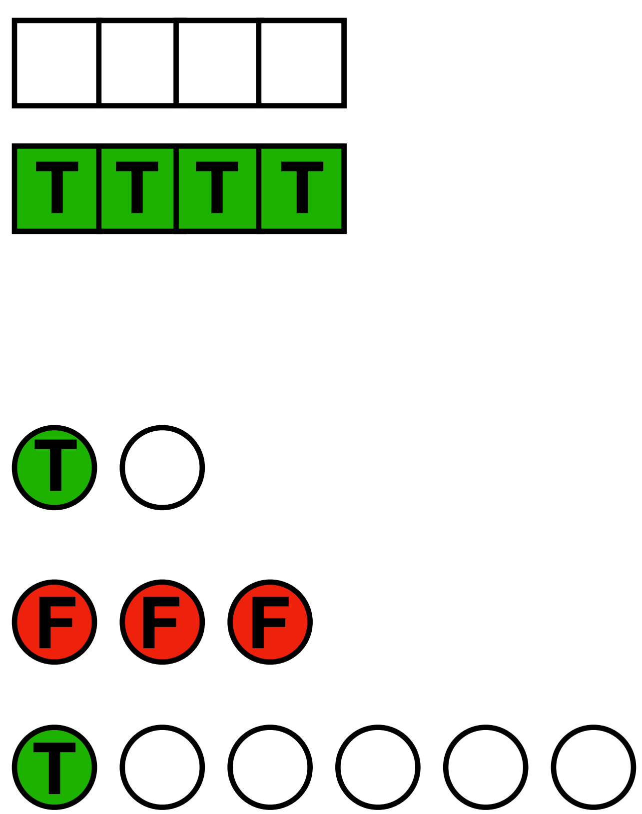
$$C_0 ::= 0000 \,|\, 0001 \,|\, \ldots \,|\, 1111$$

$$C_1 ::= 0000 \,|\, 0001 \,|\, \ldots \,|\, 1111$$

$$P_1 ::= arg_1 \,|\, arg_2 \,|\, C_0$$

$$P_2 ::= P_1 + P_1 \,|\, arg1 \,|\, arg_2 \,|\, C_1$$

$$P_3 ::= P_2 + P_1 \,|\, P_2 - P_1 \,|\, \ldots$$



$$\boxed{P_3 = 15 + arg_1}$$

Safety Invariant

```
int x = 5;

while ( x < 1000)
  x++;

assert( 5 < x && x < 1005)
```

$$init(x) \iff x = 0$$

$$trans(x, x') \iff x' = x + 1$$

$$\boxed{inv(x) = (4 < x) \wedge (x > 1003)}$$

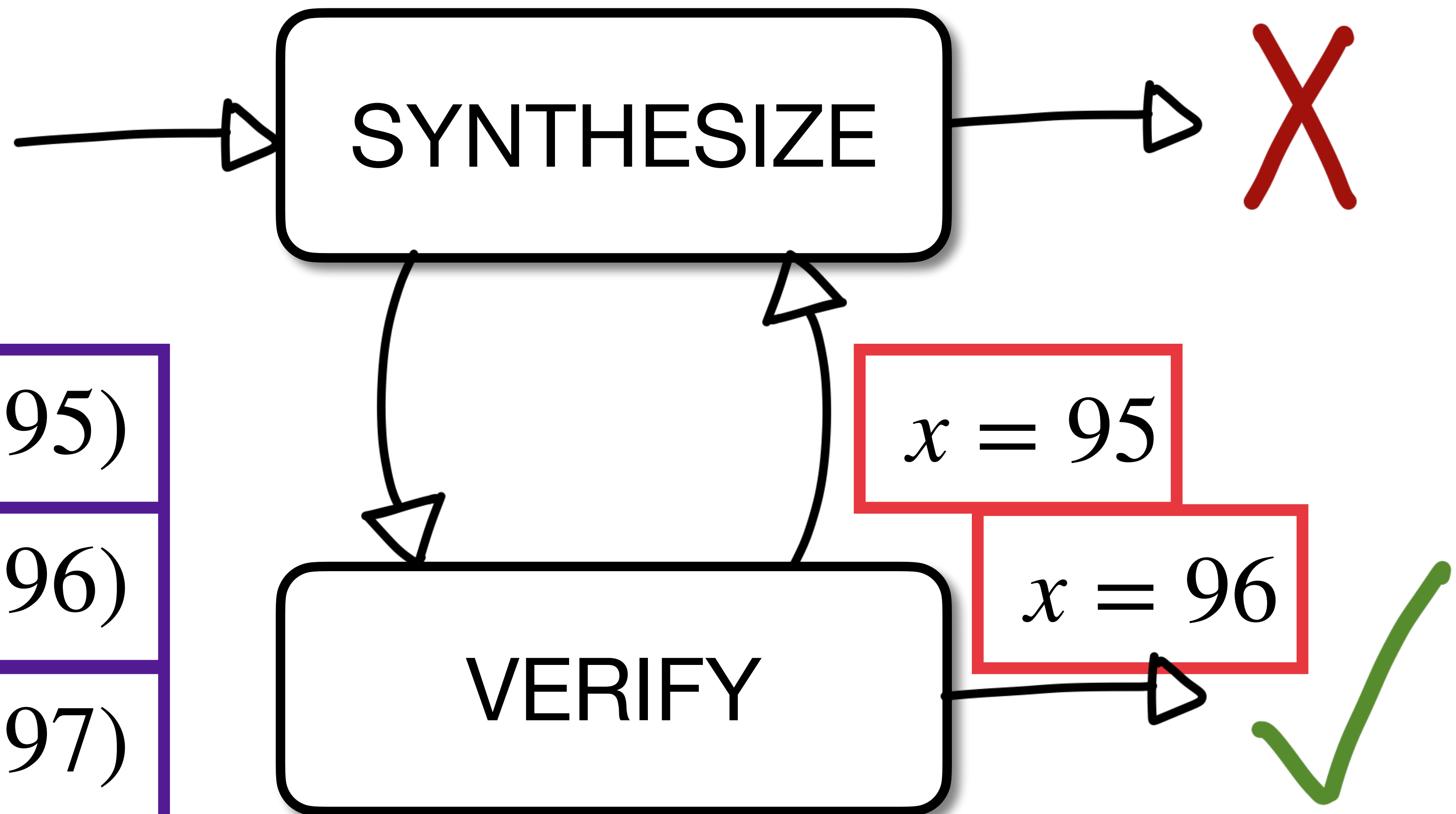
Target:

$$inv(x) = (4 < x) \wedge (x < 1003)$$

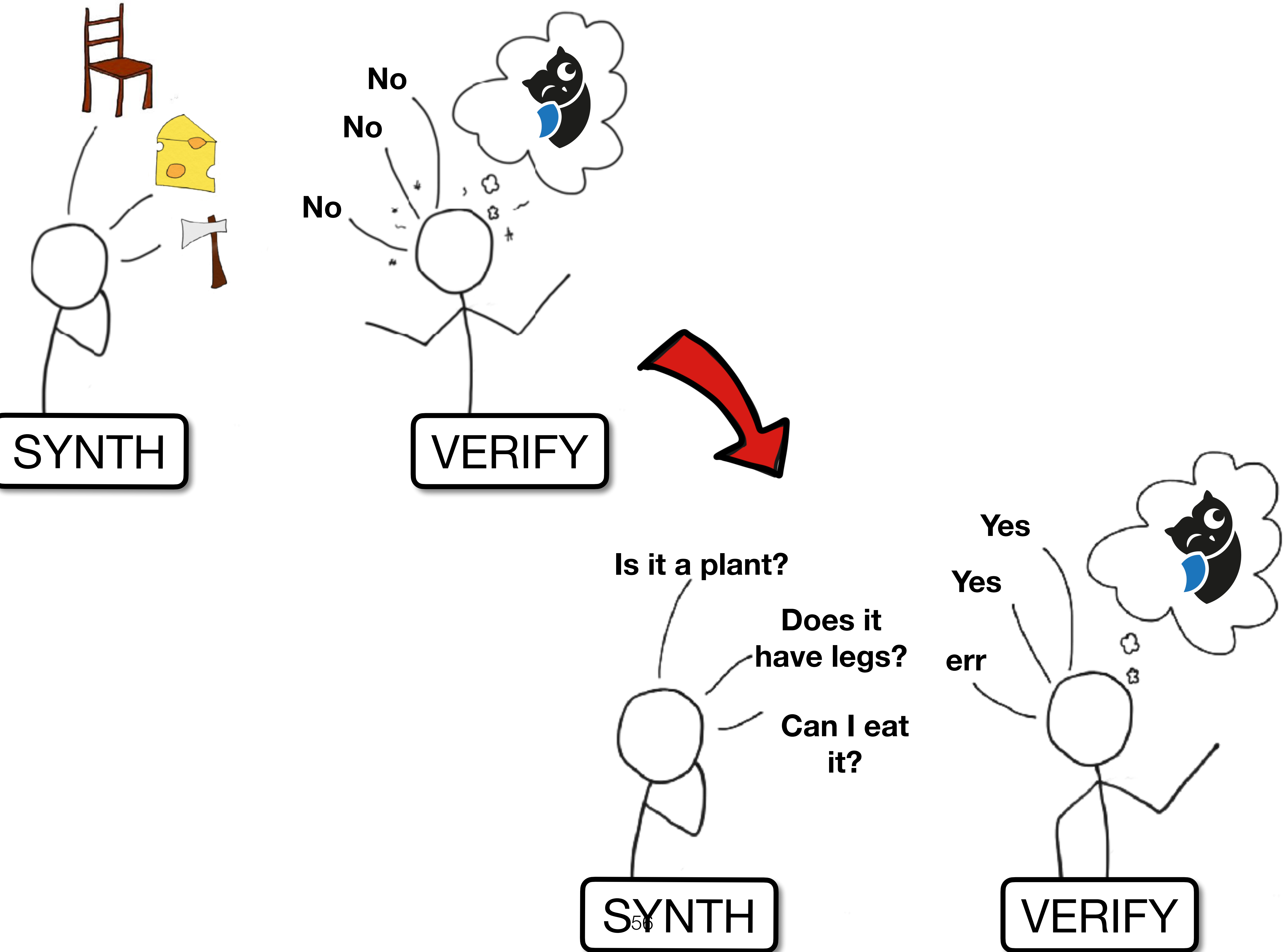
$$inv(x) = (x < 95)$$

$$inv(x) = (x < 96)$$

$$inv(x) = (x < 97)$$

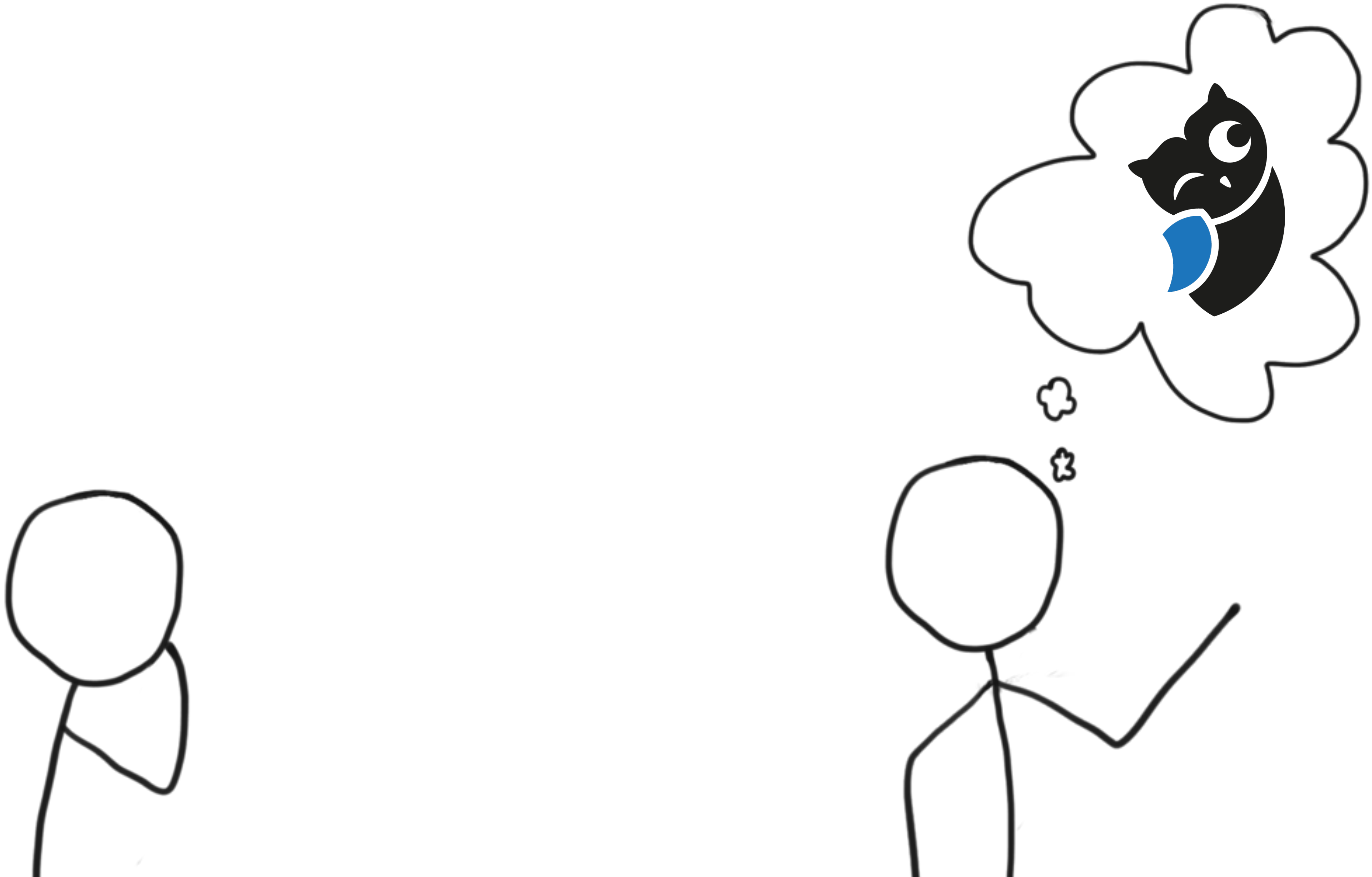


And so on ..



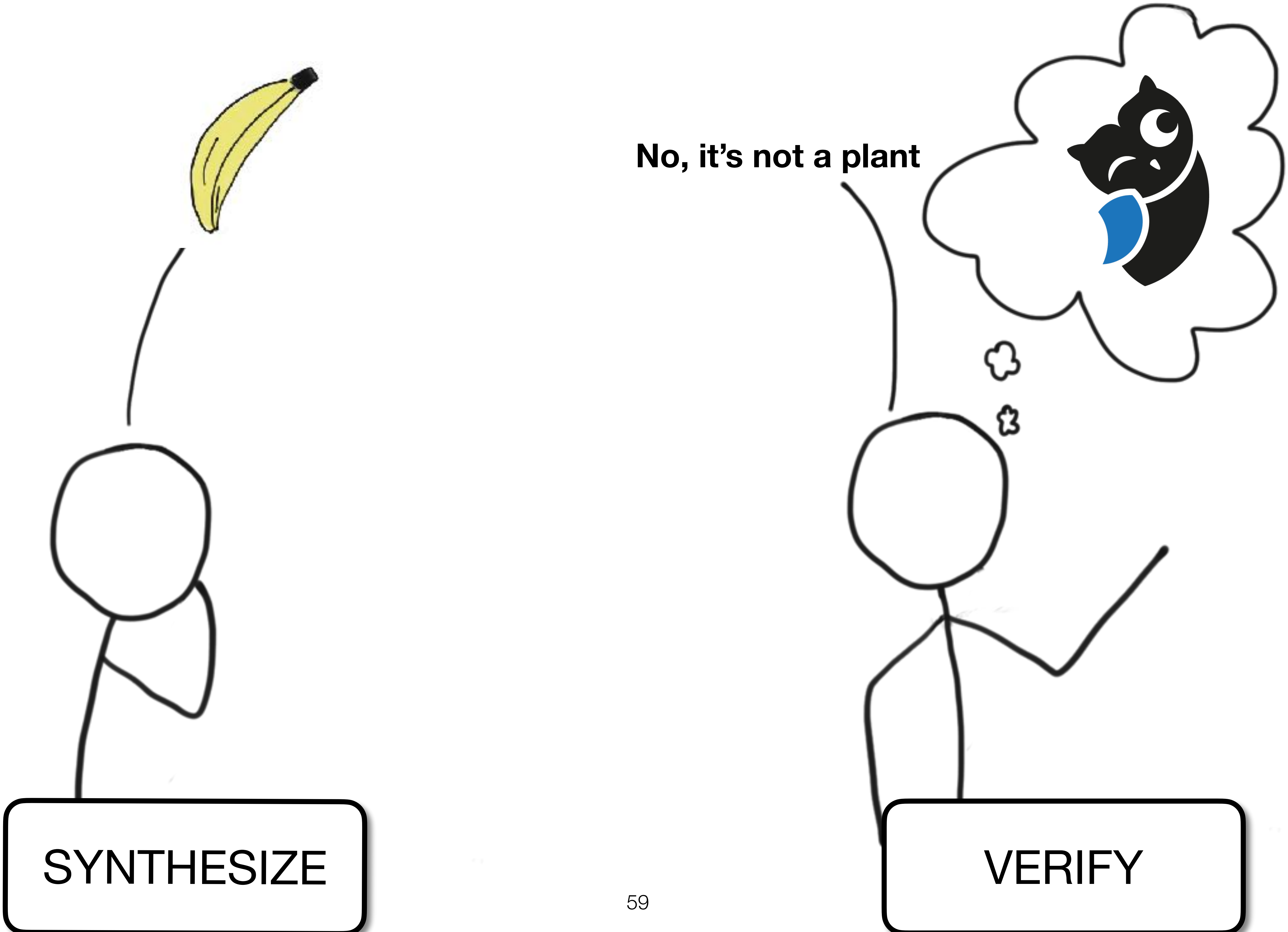
No
No
No
SYNTH
VERIFY
Is it a plant?
Does it have legs?
Can I eat it?
Yes
Yes
err
SYNTH
VERIFY

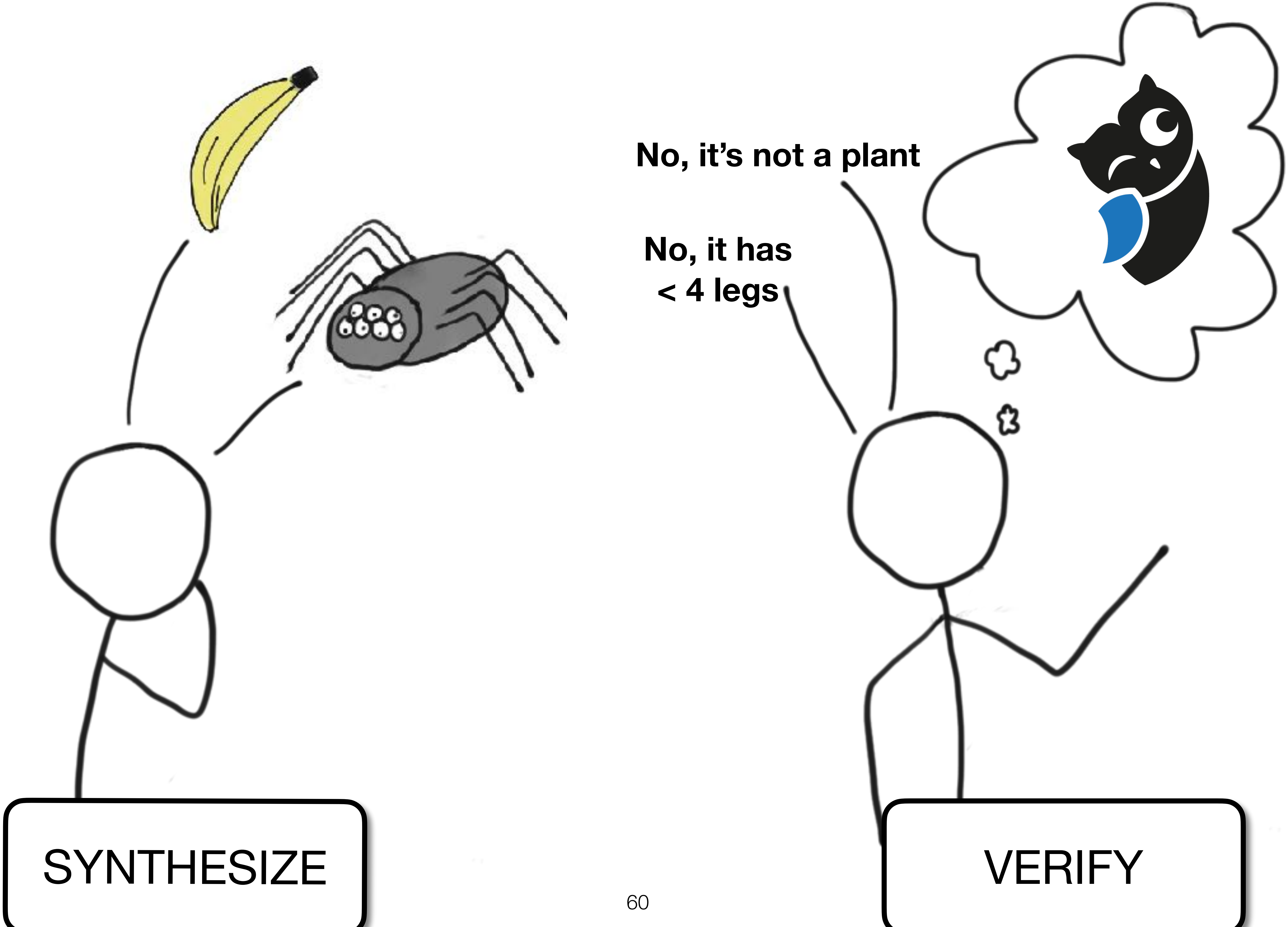
Can we ask more general questions?

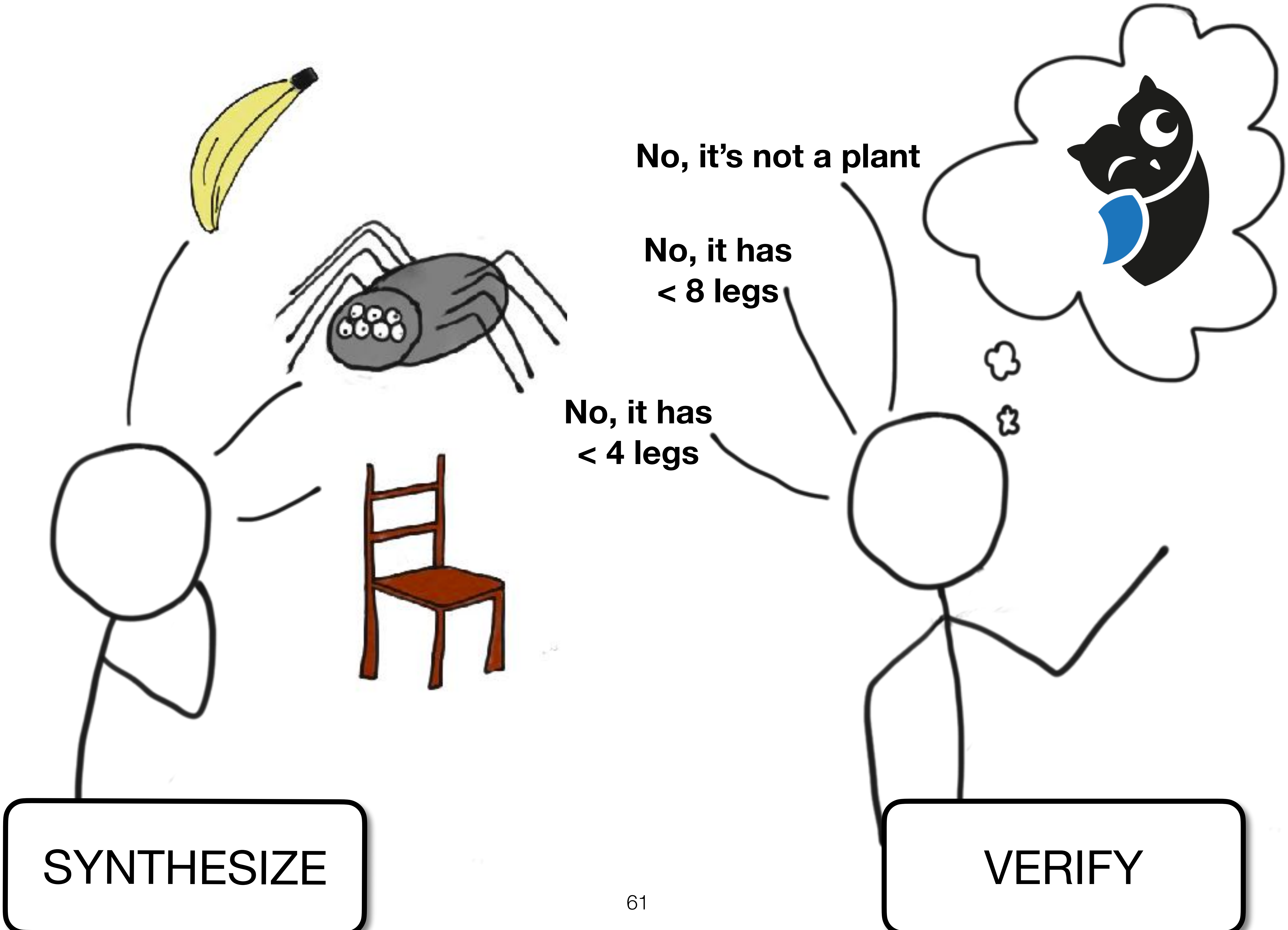


SYNTHESIZE

VERIFY


No, it's not a plant
SYNTHESIZE
VERIFY


No, it's not a plant
No, it has
< 4 legs
SYNTHESIZE
VERIFY


No, it's not a plant
No, it has
< 8 legs
No, it has
< 4 legs
SYNTHESIZE
VERIFY

Can we give more general answers?

More general questions

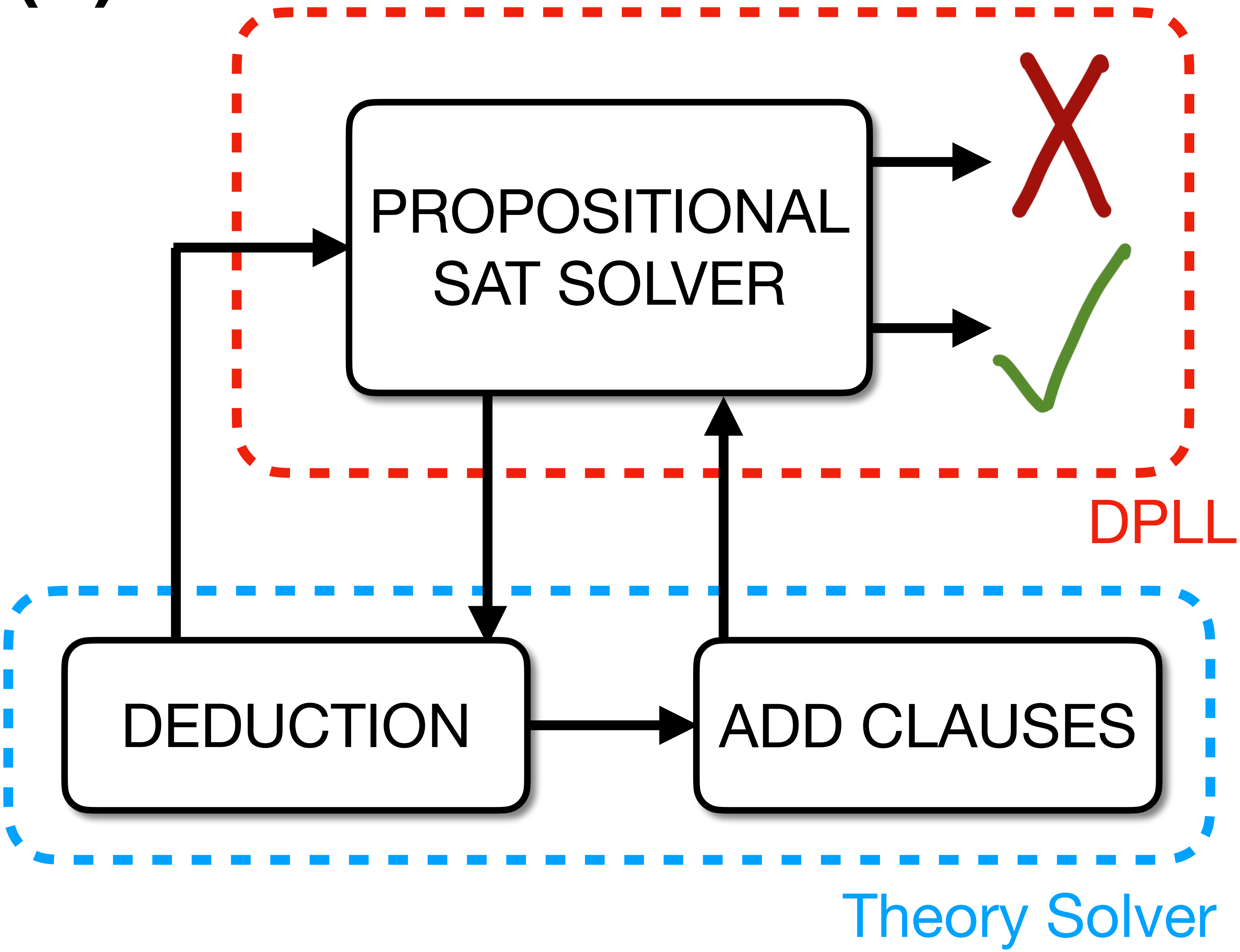
+

More general answers

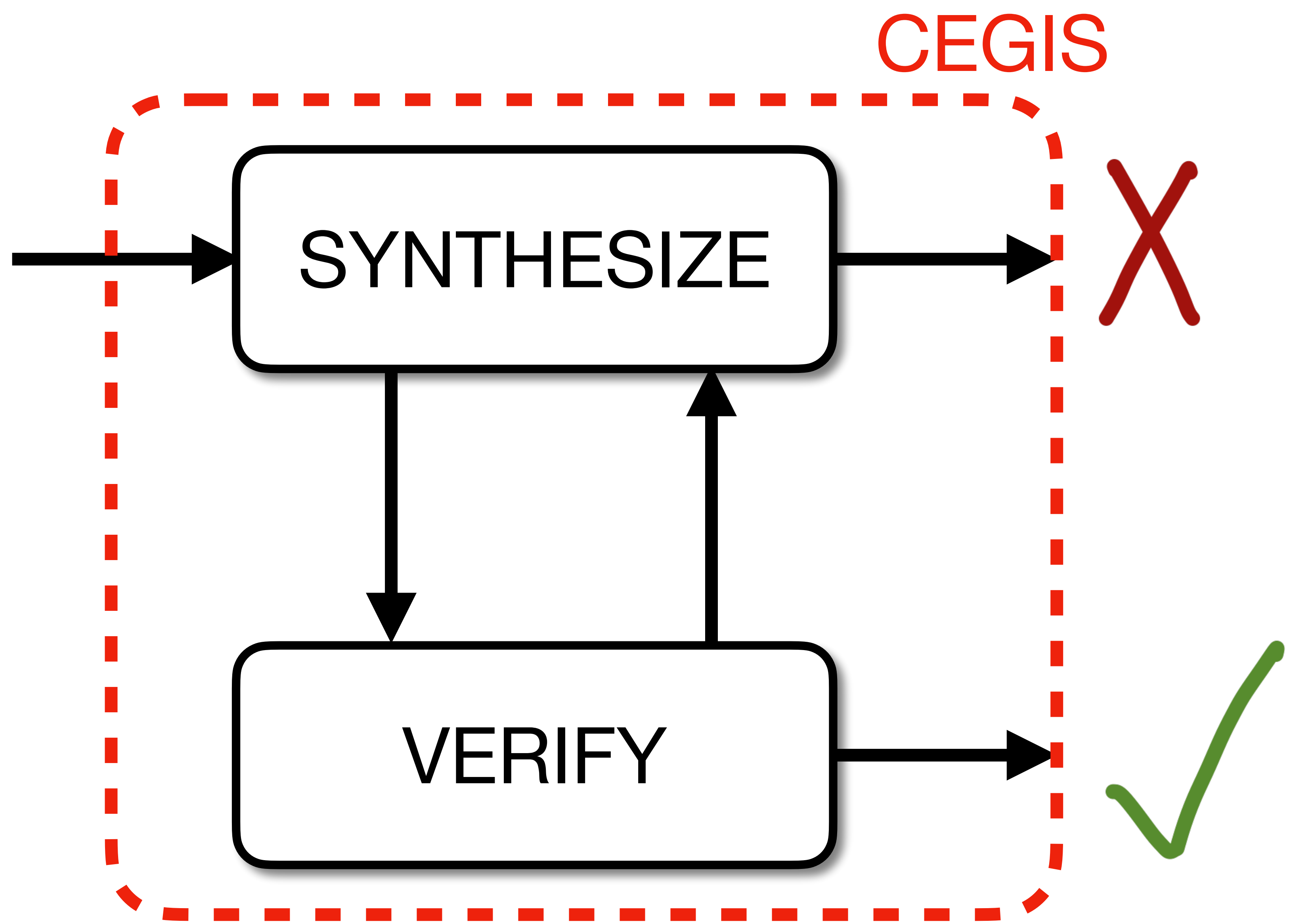
=

CEGIS(T)

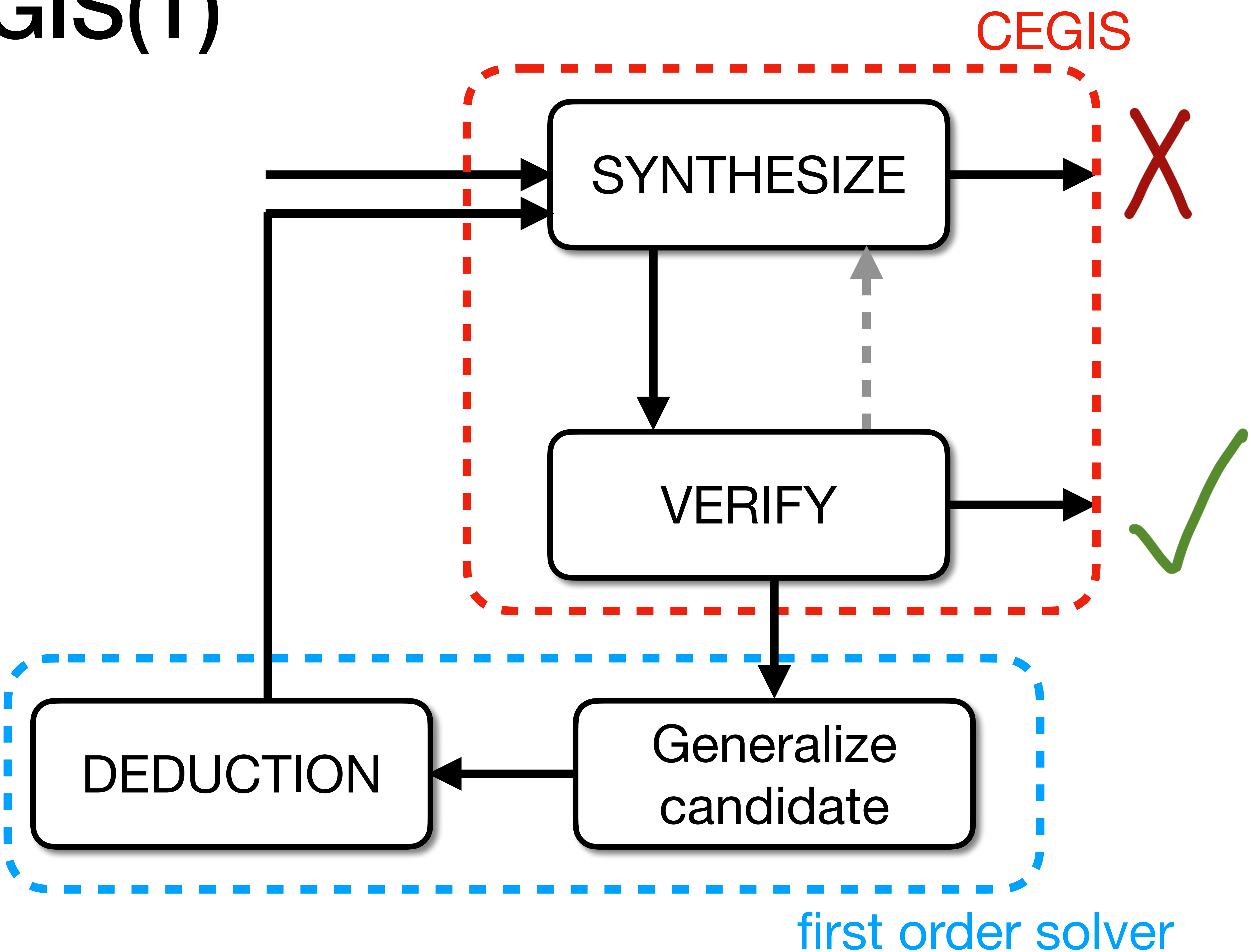
DPLL(T)


PROPOSITIONAL SAT SOLVER
DPLL
DEDUCTION
ADD CLAUSES
Theory Solver

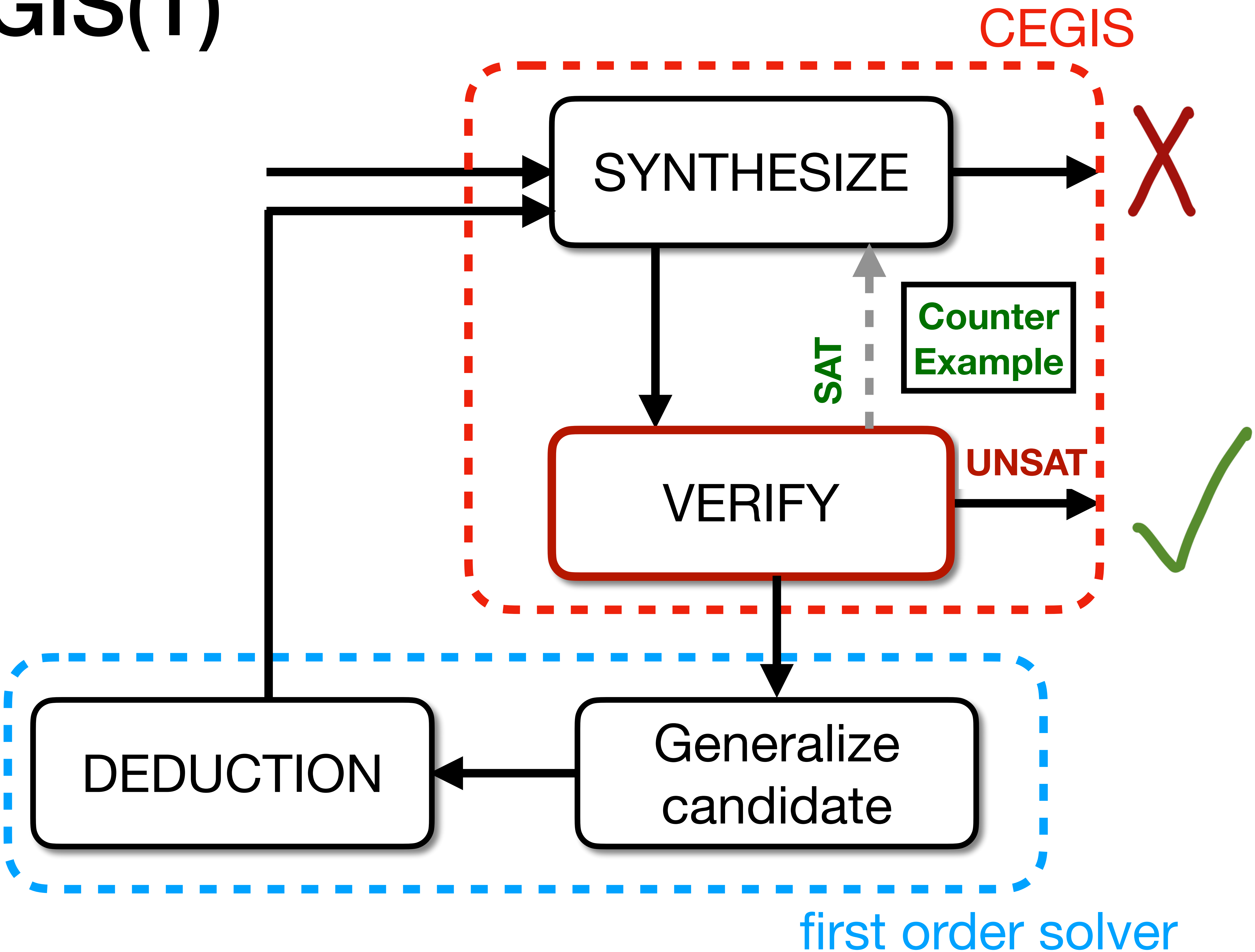
CEGIS(T)



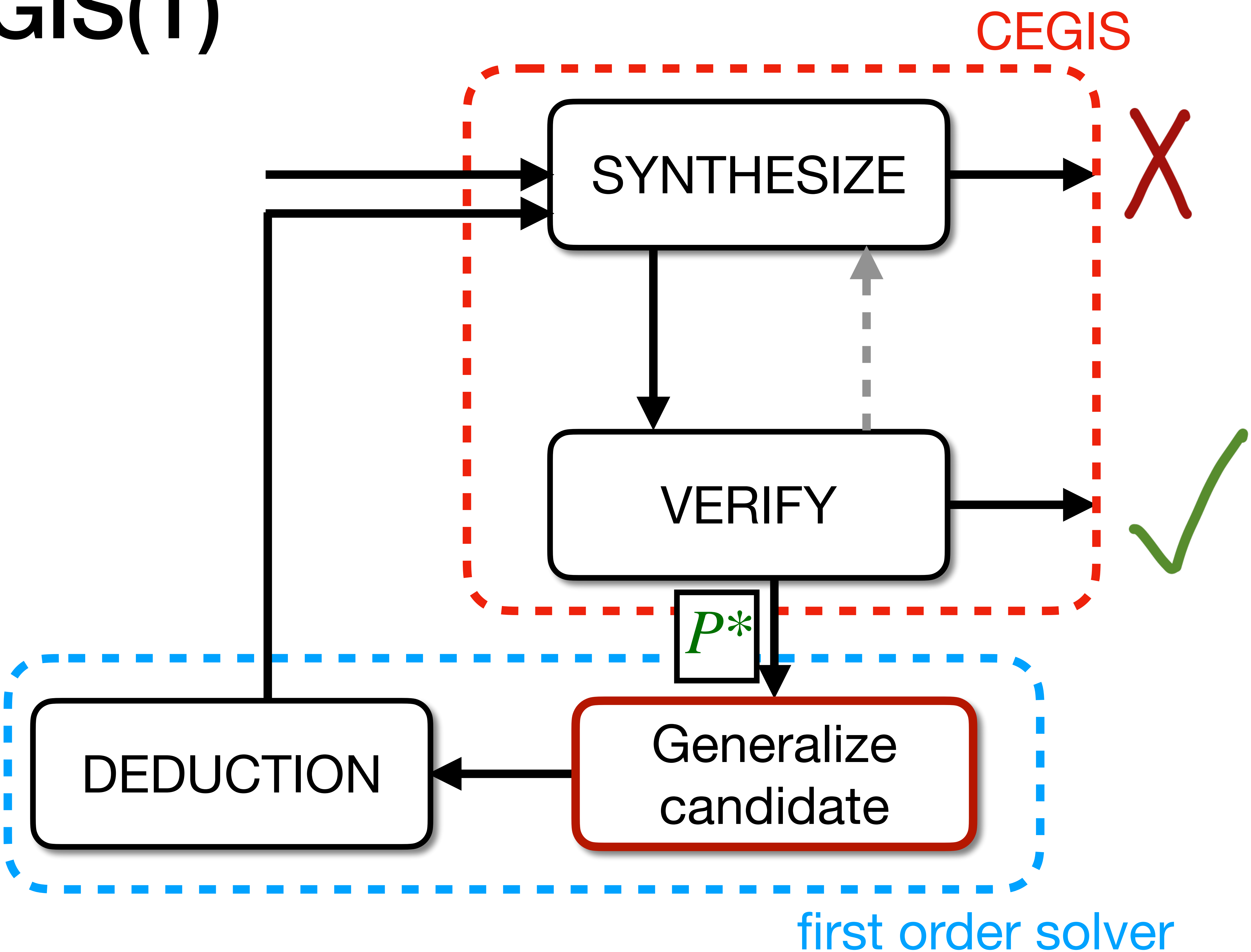
CEGIS(T)


CEGIS
SYNTHESIZE
VERIFY
DEDUCTION
Generalize candidate
first order solver

CEGIS(T)


CEGIS
SYNTHESIZE
SAT
Counter Example
VERIFY
UNSAT
DEDUCTION
Generalize candidate
first order solver

CEGIS(T)


CEGIS
SYNTHESIZE
VERIFY
P^*
DEDUCTION
Generalize candidate
first order solver

Generalize

Candidate

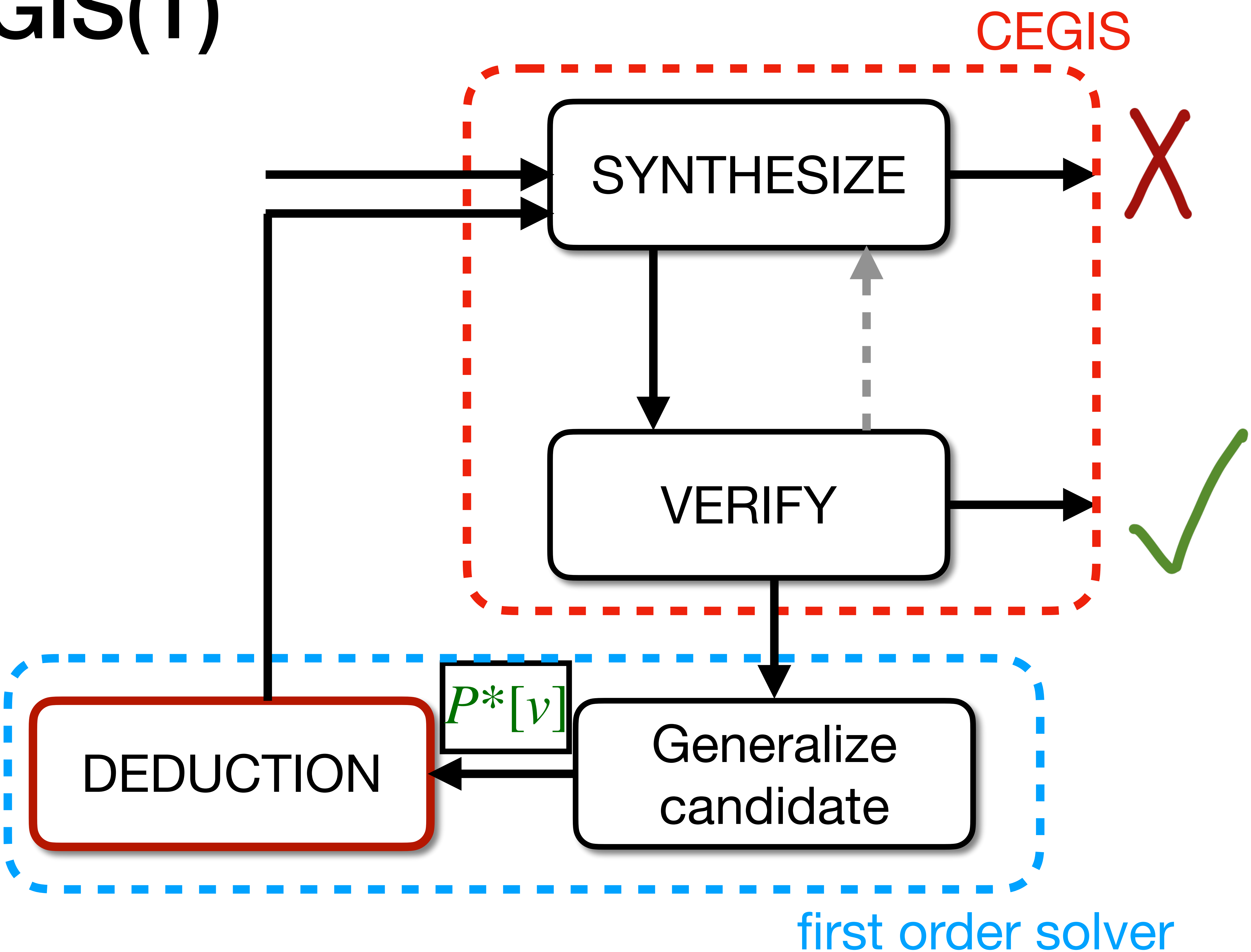
P^* $(x < 95)$

$\downarrow$

$P^*[v]$ $(x < v)$

Generalized candidate

CEGIS(T)



CEGIS
SYNTHESIZE
VERIFY
DEDUCTION
P*[v]
Generalize candidate
first order solver

Deduction

$$\exists v \forall x \,.\, \sigma(P^*[v], x)$$

UNSAT

SAT

$\neg P^*[v]$

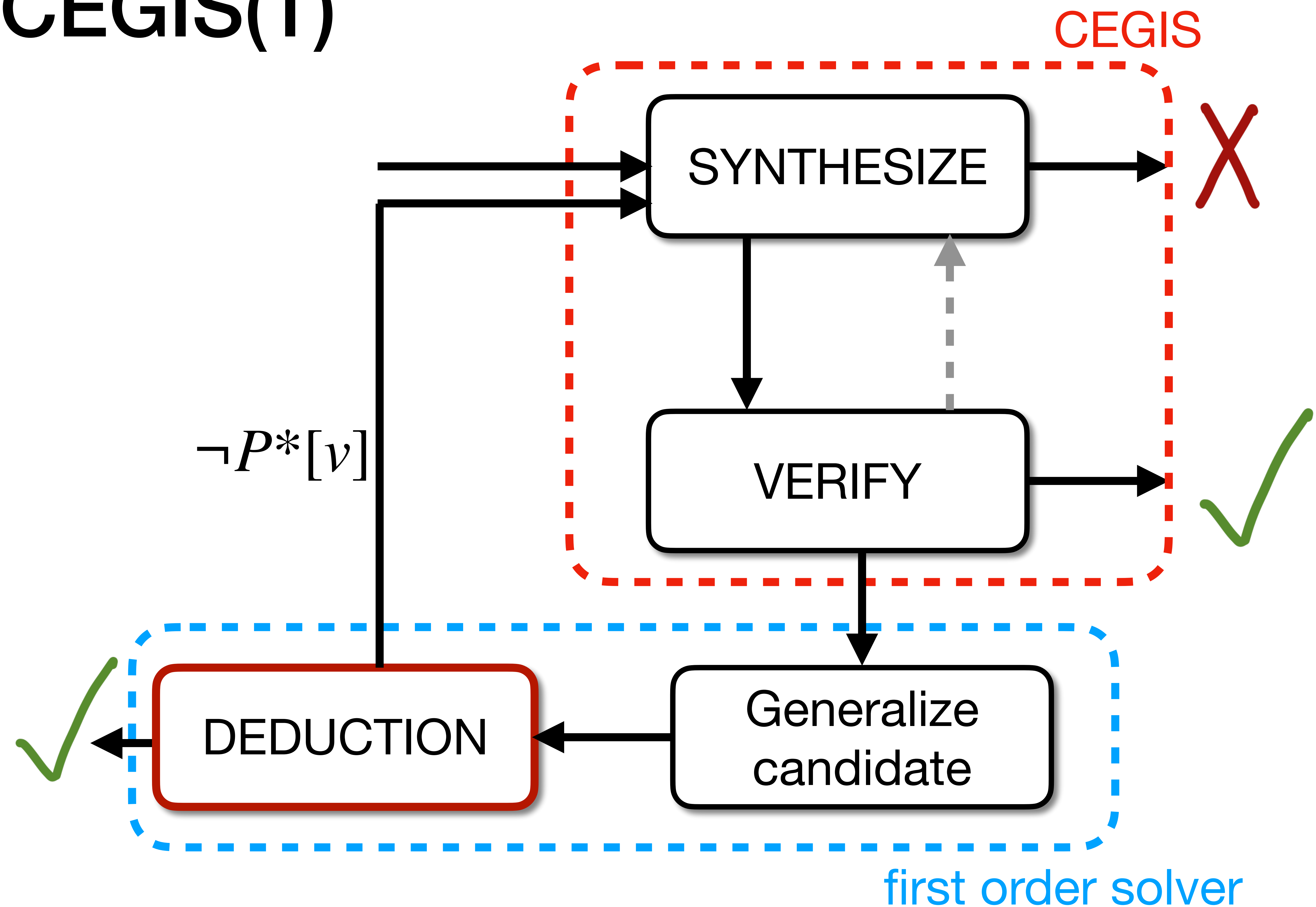
BLOCK

$P^*[v]$

SOLUTION

is there a value for v that makes $(x < v)$ a valid invariant

CEGIS(T)


CEGIS
SYNTHESIZE
VERIFY
$\neg P^*[v]$
DEDUCTION
Generalize candidate
first order solver

CEGIS(T) - SMT

$$\boxed{\exists v \forall x \,.\, \sigma(P^*[v], x)}$$

UNSAT

SAT

$$\neg P^*[v]$$

BLOCK

$$P^*[v]$$

SOLUTION

First Order Solver

Solves first order formula with:

- Arbitrary propositional structure
- 1 quantifier alternation

Paper presents 2 versions:

- SMT (Z3) [1]
- Fourier Motzkin

[1] Z3: An Efficient SMT Solver. De Moura et al. TACAS 2008

CEGIS(T) - SMT

$$\boxed{\exists v \forall x \,.\, \sigma(P^*[v], x)}$$

UNSAT → $\neg P^*[v]$

BLOCK

SAT → $P^*[v]$

SOLUTION

CEGIS(T) - SMT

$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v < c)$

$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v > c)$

$\neg P^*[v]$
BLOCK

$v > c$
CONSTRAINT

$v < c$
CONSTRAINT

$P^*[v]$
SOLUTION

Target:
$$inv(x) = (4 < x) \land (x < 1003)$$

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \land (v < c)$$

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \land (v > c)$$

$$P^* = (x < 95)$$

$$P^*[v] = (x < v)$$

$\neg P^*[v]$ **BLOCK**

$v > c$ **CONSTRAINT**

$v < c$ **CONSTRAINT**

$P^*[v]$ **SOLUTION**

Target:
$$inv(x) = (4 < x) \land (x < 1003)$$

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \land (v < 95)$$

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \land (v > 95)$$

$$P^* = (x < 95)$$

$$P^*[v] = (x < v)$$

$\neg P^*[v]$
BLOCK

$v > 95$
CONSTRAINT

$v < 95$
CONSTRAINT

$P^*[v]$
SOLUTION

Target:

$$inv(x) = (4 < x) \wedge (x < 1003)$$

UNSAT

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v < 95)$$

UNSAT

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v > 95)$$

$\neg P^*[v]$

BLOCK

$v > 95$

CONSTRAINT

$v < 95$

CONSTRAINT

$P^*[v]$

SOLUTION

Target:

$$inv(x) = (4 < x) \wedge (x < 1003)$$

UNSAT

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v < 95)$$

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v > 95)$$

$\neg P^*[v]$
BLOCK

$v > 95$
CONSTRAINT

$v < 95$
CONSTRAINT

$P^*[v]$
SOLUTION

Target:

$$inv(x) = (4 < x) \land (x < 1003)$$

$\exists v \forall x . \sigma(P^*[v], x) \land (v < 95)$

UNSAT

$\exists v \forall x . \sigma(P^*[v], x) \land (v > 95)$

$\neg P^*[v]$	$v > 95$	$v < 95$	$P^*[v]$
BLOCK	**CONSTRAINT**	**CONSTRAINT**	**SOLUTION**

Target:
$inv(x) = (4 < x) \wedge (x < 1003)$

SAT

$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v_1 < 95)$

$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v_1 > 95)$

$\neg P^*[v]$
BLOCK

$v > 95$
CONSTRAINT

$v < 95$
CONSTRAINT

$P^*[v]$
SOLUTION

Target:
$$inv(x) = (4 < x) \wedge (x < 1003)$$

SAT

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v_1 < 95)$$

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v_1 > 95)$$

$\neg P^*[v]$
BLOCK

$v > 95$
CONSTRAINT

$v < 95$
CONSTRAINT

$P^*[v]$
SOLUTION

Target:

$$inv(x) = (4 < x) \wedge (x < 1003)$$

TIMEOUT

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v_1 < 95)$$

TIMEOUT

$$\exists v \forall x \,.\, \sigma(P^*[v], x) \wedge (v_1 > 95)$$

$$\times$$

$\neg P^*[v]$	$v > 95$	$v < 95$	$P^*[v]$
BLOCK	**CONSTRAINT**	**CONSTRAINT**	**SOLUTION**

Experiments

Benchmarks:

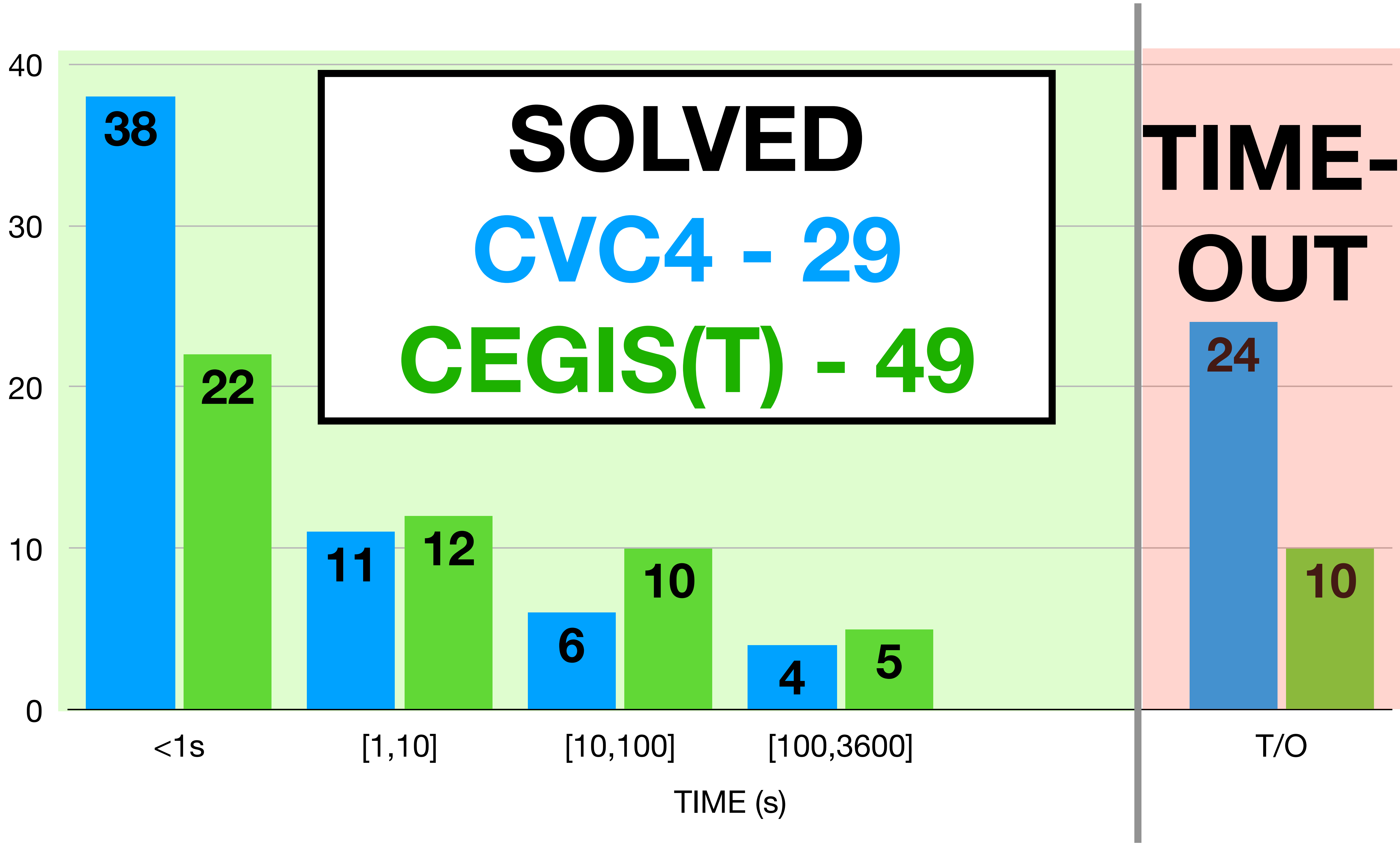
- Bitvectors
- Syntax-guided Synthesis competition (without the syntax)
- Loop invariants
- Danger invariants

Solvers:

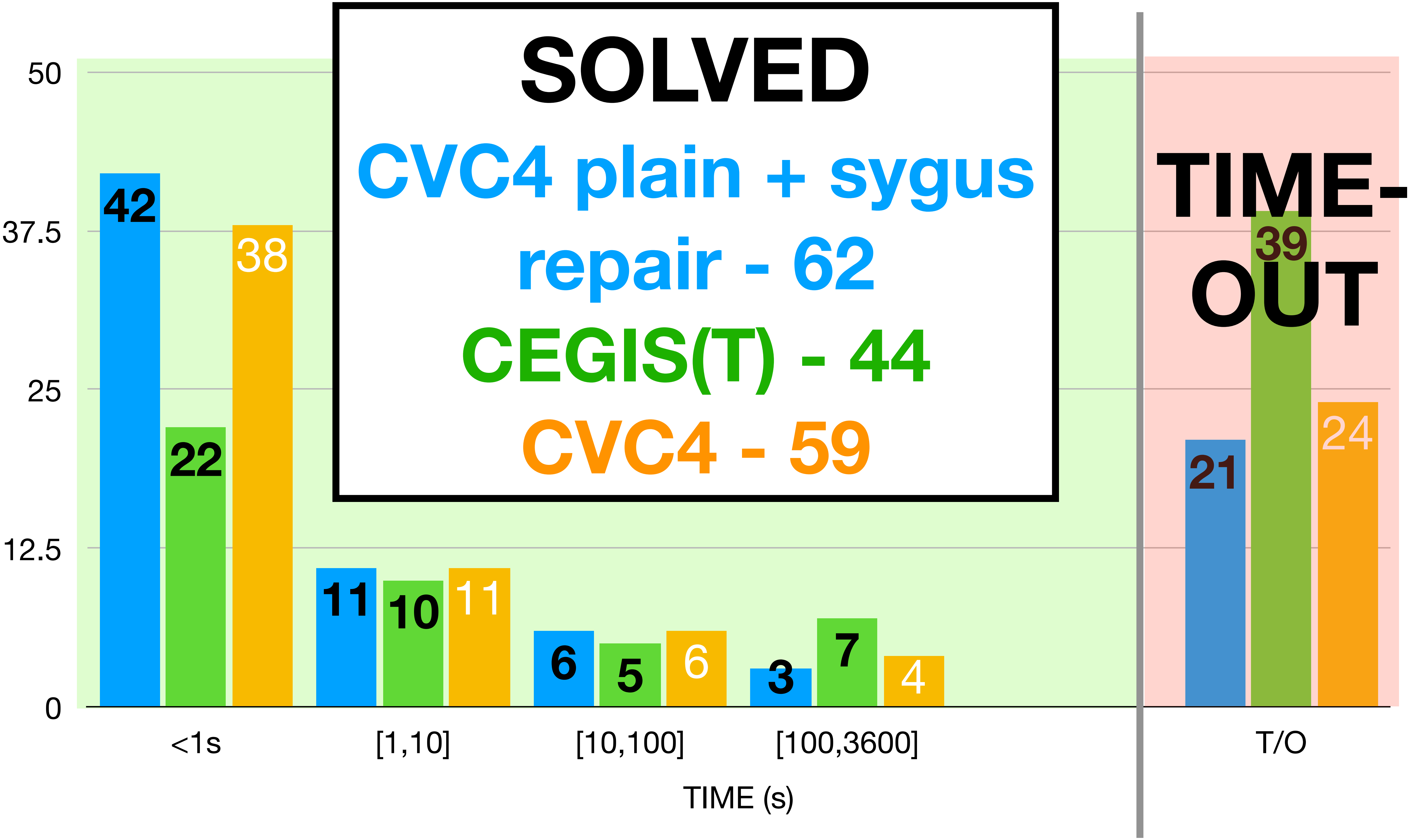
- CVC4 [1]
- EUSolver, E3Solver, LoopInvGen – bitvectors with no grammar unsupported

[1] CVC4. Barrett et al. CAV 2011

Experiments



Experiments



CEGIS(T) - Conclusions

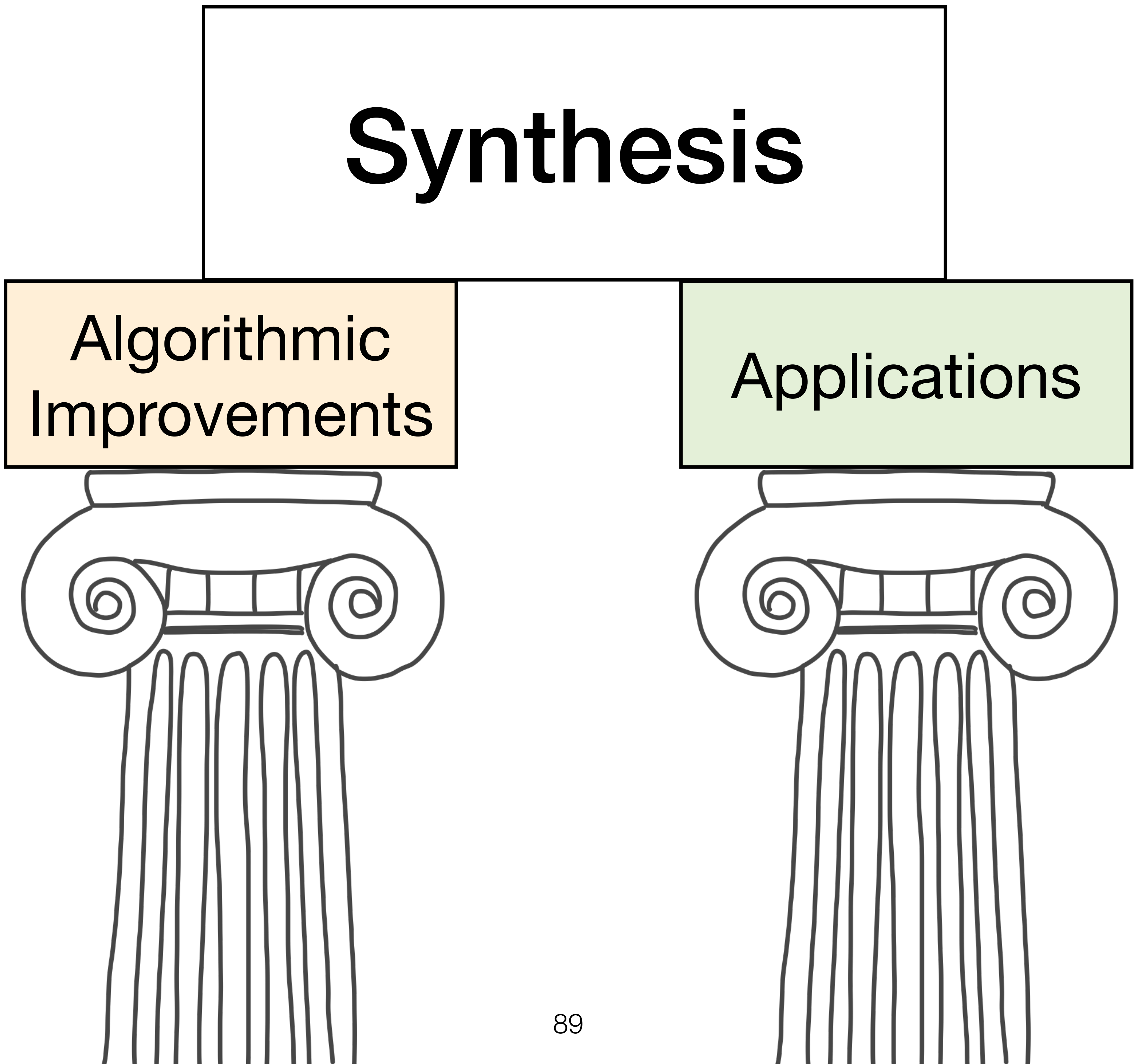
CEGIS(T) solves program synthesis via 1st order solvers that support quantifiers:

- Enables use of existing solvers

Algorithmic insights:

- verify generalized candidate solutions
- return generalized counterexamples

Future Work

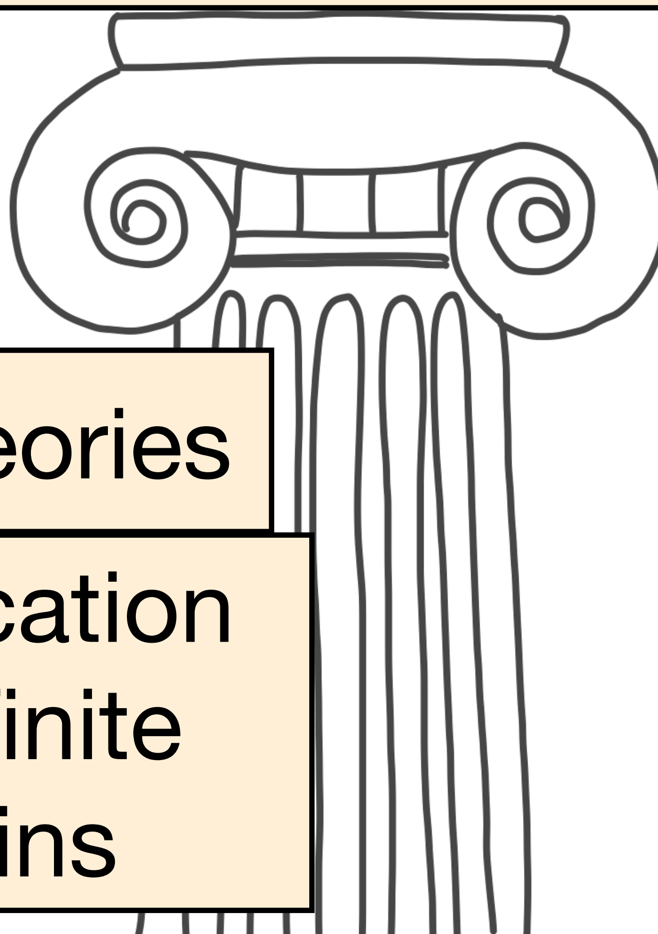


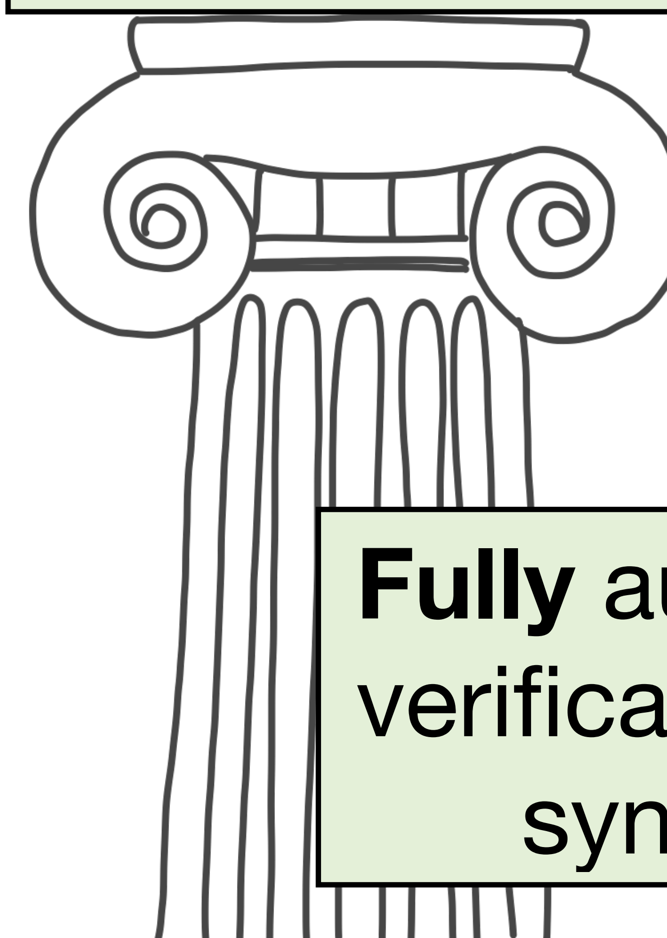
Future Work

Synthesis

Algorithmic Improvements

Applications





More theories

Quantification over infinite domains

Fully automating verification using synthesis

Quantification Over Infinite Domains

- Reasoning about unbounded or large data structures requires quantification

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

Quantifier free **Quantifier free**

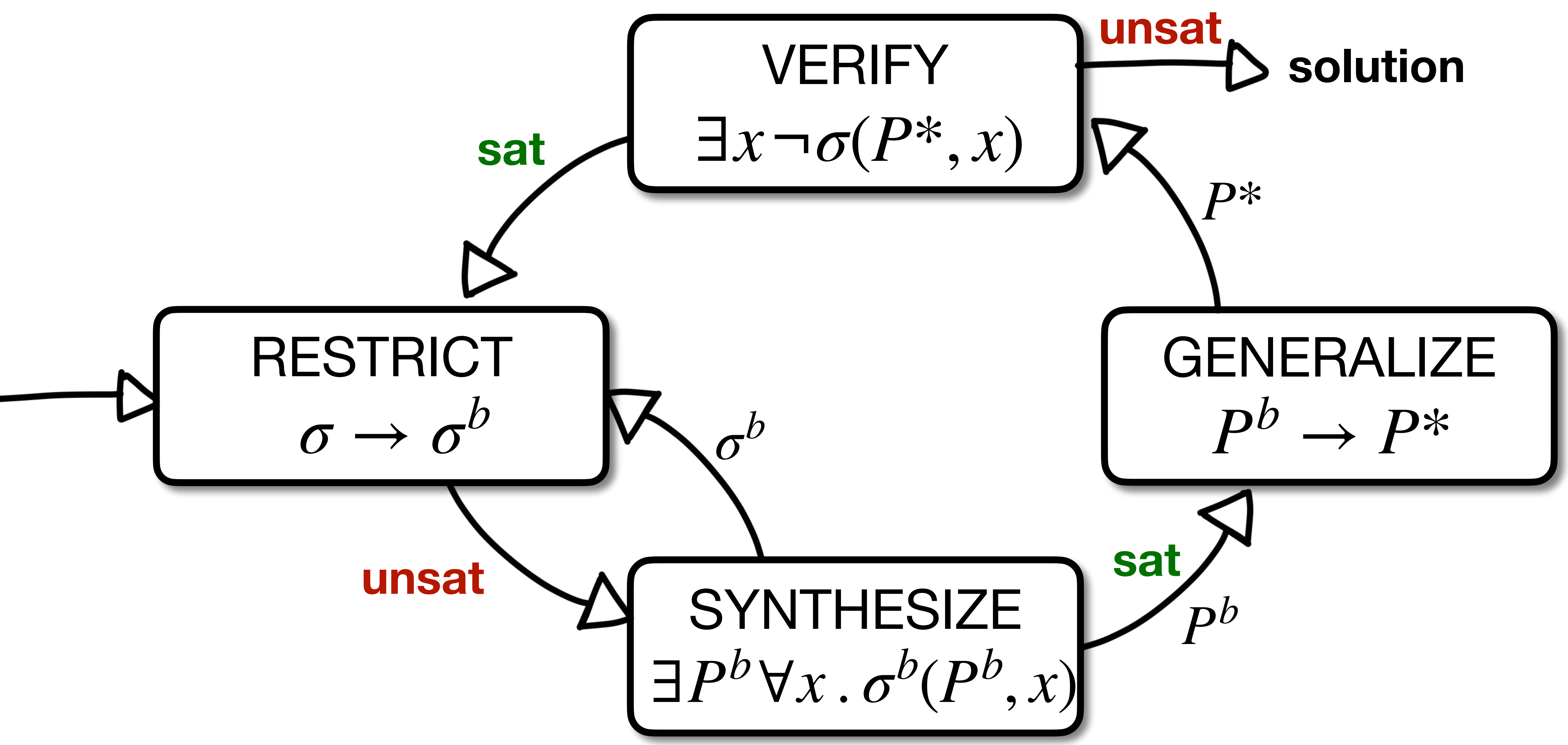
Quantification Over Infinite Domains

- Reasoning about unbounded or large data structures requires quantification

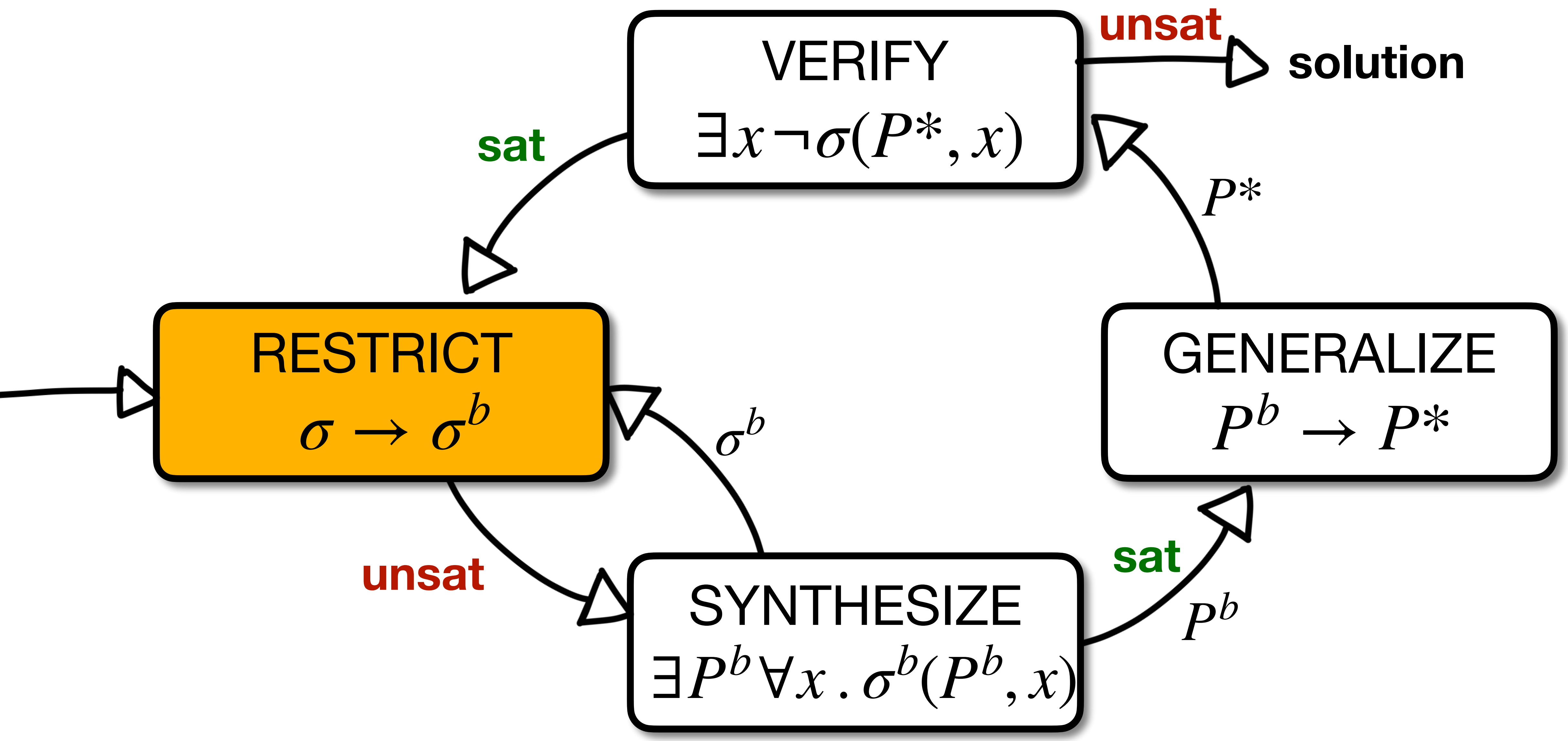
$$\exists P \forall x \,.\, \boxed{\sigma}(\boxed{P}, x)$$

Quantifiers **Quantifiers**

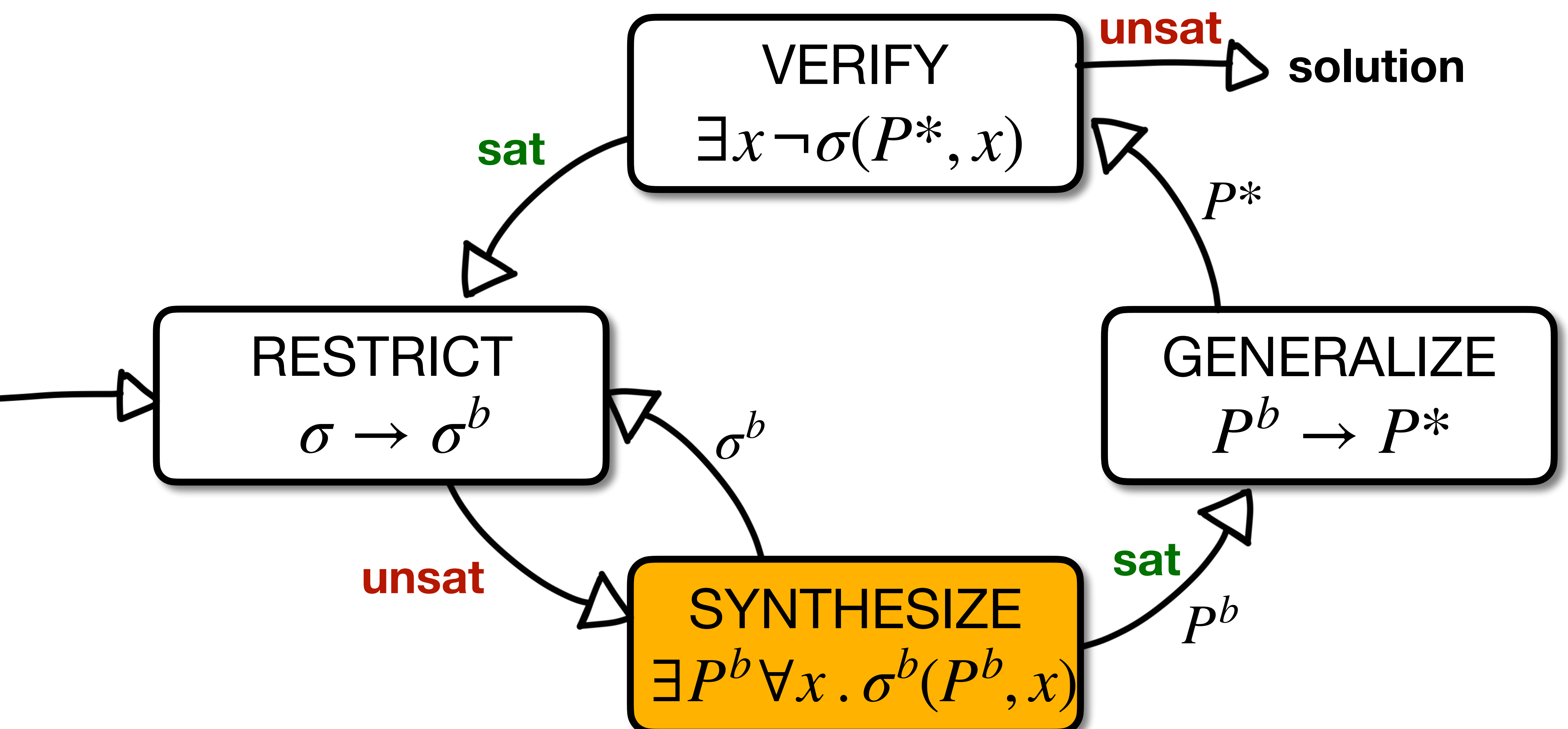
Quantification Over Infinite Domains



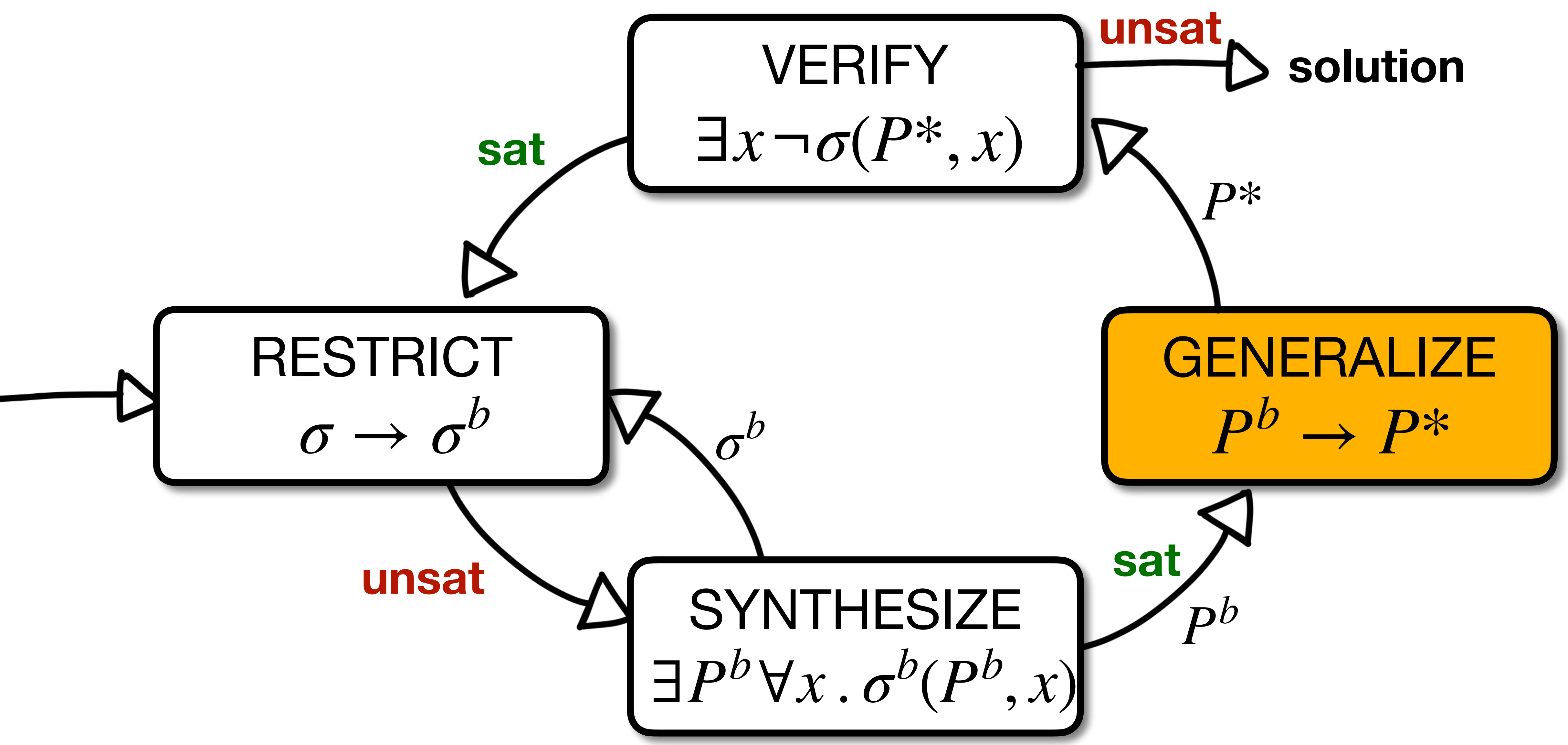
Quantification Over Infinite Domains



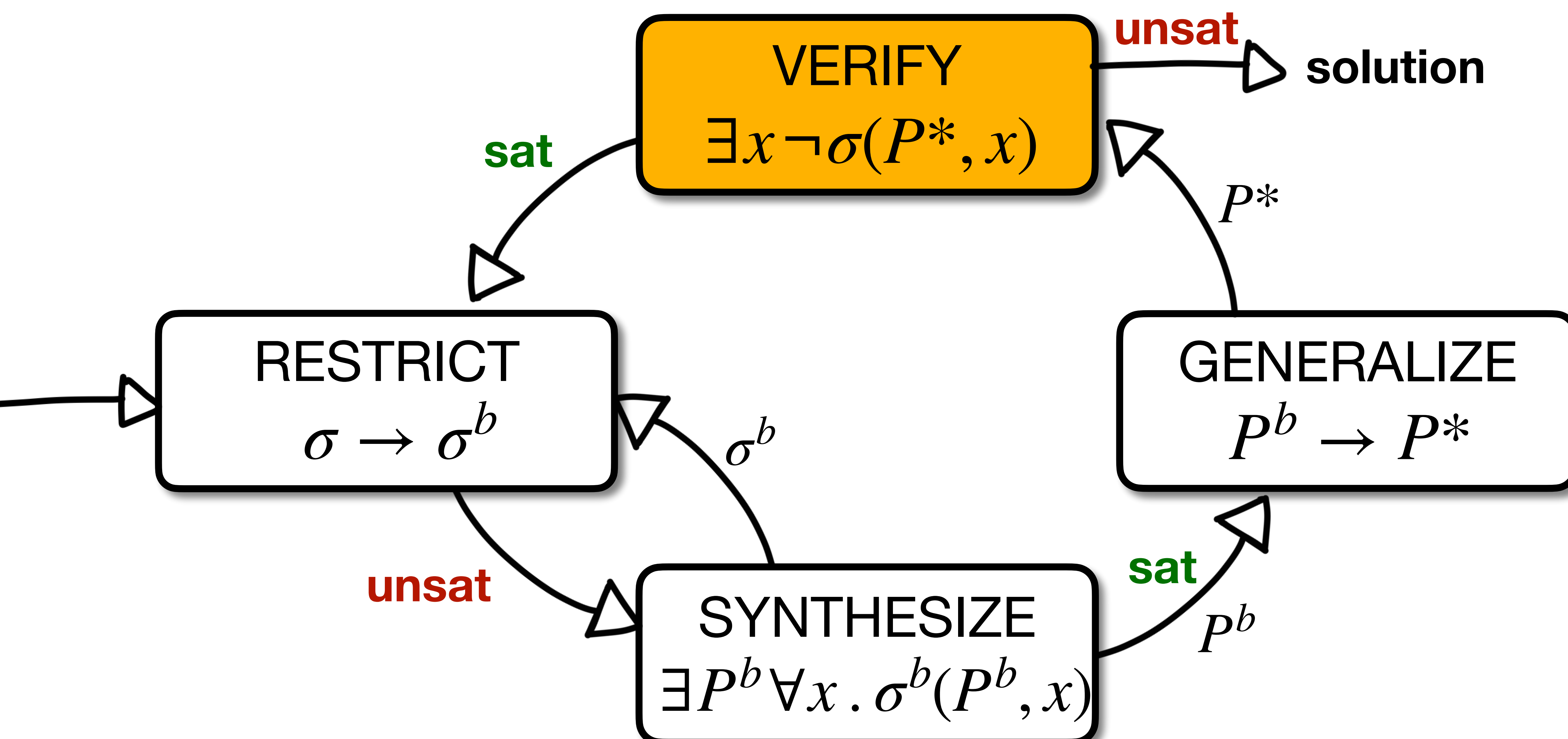
Quantification Over Infinite Domains



Quantification Over Infinite Domains



Quantification Over Infinite Domains



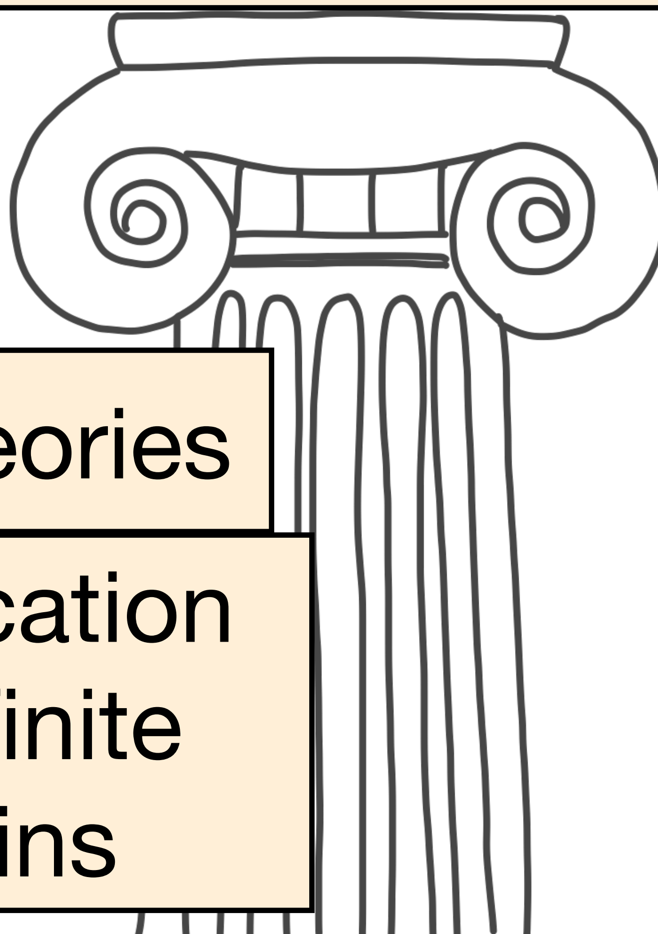
Example	Z3-Horn solver	QUIC3	SynRG
duplication	t/o	t/o	t/o
equal arrays 1	✓	✓	✓
equal arrays 2	**u**	**u**	t/o
exists 1	**u**	**u**	✓
Fibonacci	t/o	t/o	t/o
fill 1	t/o	t/o	✓
fill 2	t/o	t/o	t/o
find first 1	✓	✓	✓
find first 2	**u**	**u**	✓
permutation 1	**u**	**u**	t/o
permutation 2	t/o	t/o	✓
permutation 3	t/o	t/o	✓
permutation 4	t/o	t/o	✓
permutation 5	t/o	t/o	✓
simple array	t/o	t/o	✓
array and constant	t/o	t/o	✓
two indices 1	t/o	✓	✓

Table 1. Examples solved by each solver. We ran the experiments with a 600 s timeout but all the solved examples were solved within 10 s. t/o indicates the time-out was exceeded. **u** indicates the solver returned "unknown".

Future Work

Synthesis

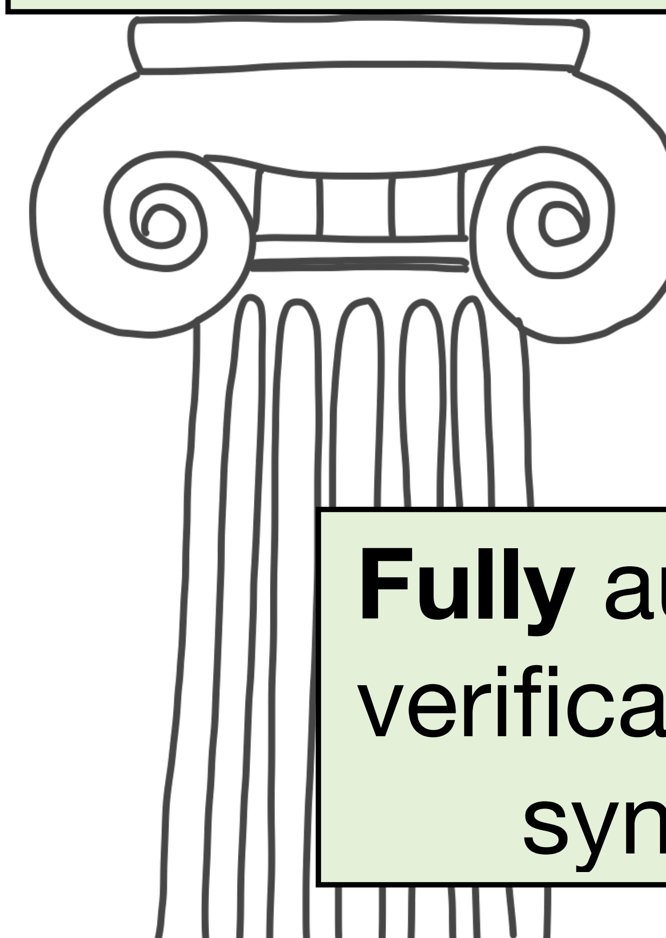
Algorithmic Improvements



More theories

Quantification over infinite domains

Applications



Fully automating verification using synthesis

Fully Automated Assertion Verification

- Verification of real-world software is not yet fully automated

Manual writing:

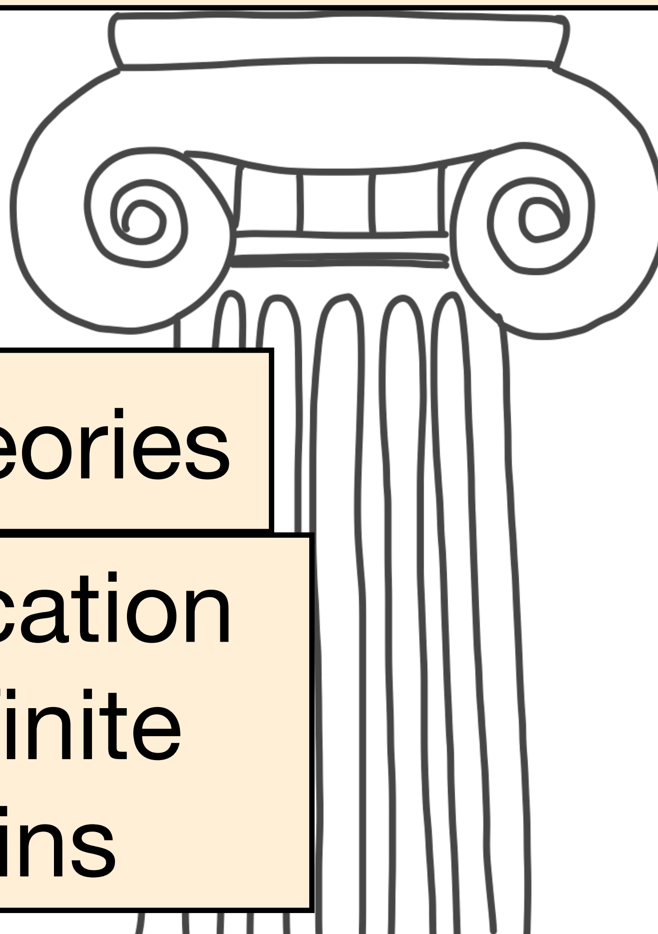
- invariants
- pre-and-post-conditions
- code summaries

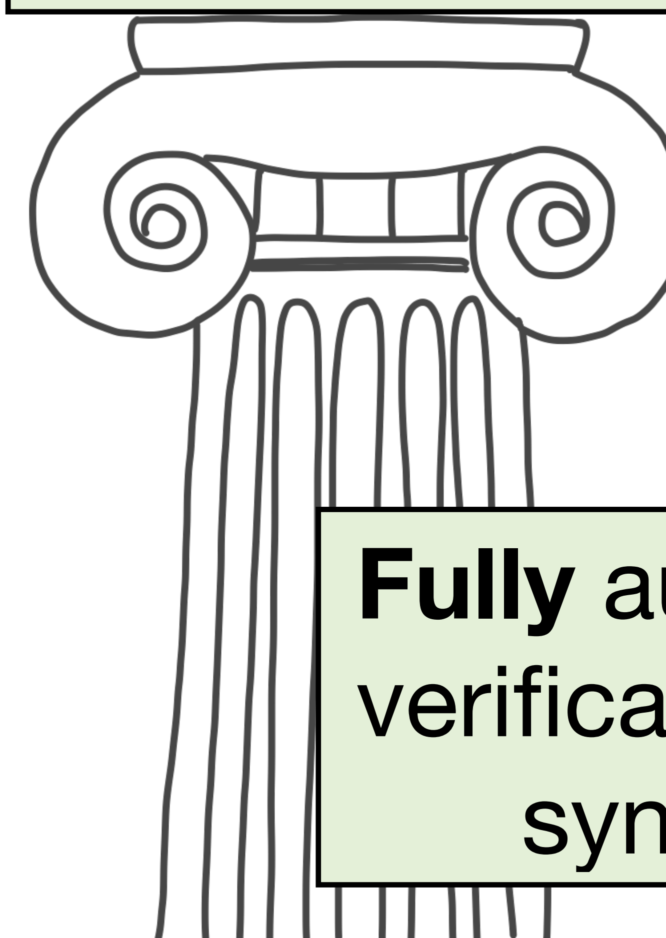
Future Work

SYNTHESIS is the new SAT

Algorithmic Improvements

Applications



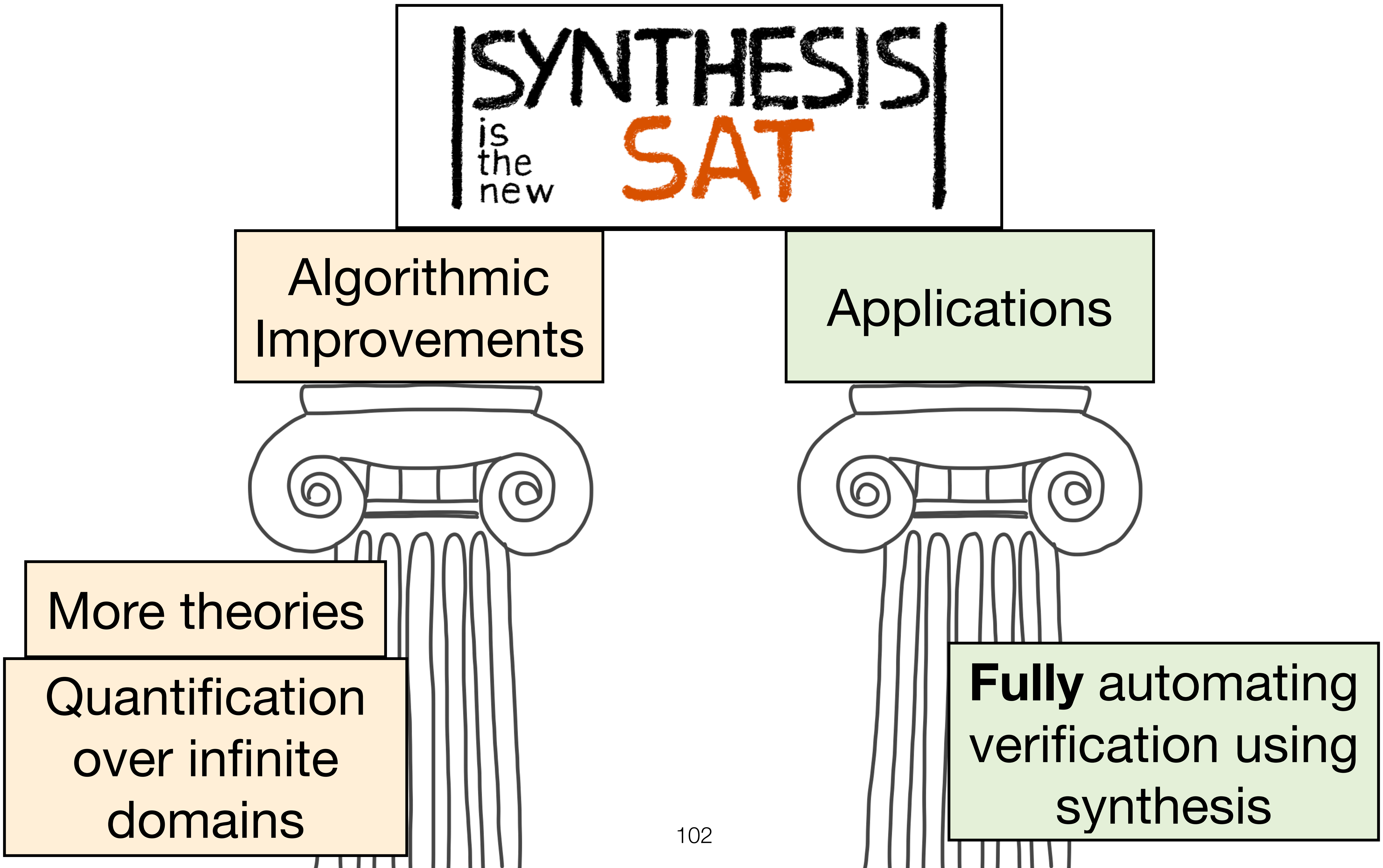


More theories

Quantification over infinite domains

Fully automating verification using synthesis

Questions?


SYNTHESIS is the new SAT
Algorithmic Improvements
Applications
More theories
Quantification over infinite domains
Fully automating verification using synthesis