THE UNIVERSITY of EDINBURGH

CEGIS(T)

CounterExample Guided Inductive Synthesis modulo Theories

CAV 2018

Alessandro Abate[1], Cristina David[2], Pascal Kesseli[3], Daniel Kroening[13],
Elizabeth Polgreen[0]

[0]University of Edinburgh, [1]University of Oxford, [2]University of Bristol, [3]DiffBlue Ltd

UNIVERSITY OF OXFORD

University of BRISTOL

diffblue
AI for Code


Counterexample Guided Inductive Synthesis Modulo Theories

Alessandro Abate[1], Cristina David[2,3](✉), Pascal Kesseli[3], Daniel Kroening[1,3], and Elizabeth Polgreen[1]

[1] University of Oxford, Oxford, UK
[2] University of Cambridge, Cambridge, UK
cd652@cam.ac.uk
[3] Diffblue Ltd., Oxford, UK

Abstract. Program synthesis is the mechanised construction of software. One of the main difficulties is the efficient exploration of the very large solution space, and tools often require a user-provided syntactic restriction of the search space. We propose a new approach to program synthesis that combines the strengths of a counterexample-guided inductive synthesizer with those of a theory solver, exploring the solution space more efficiently without relying on user guidance. We call this approach CEGIS($\mathcal{T}$), where $\mathcal{T}$ is a first-order theory. In this paper, we focus on one particular challenge for program synthesizers, namely the generation of programs that require non-trivial constants. This is a fundamentally difficult task for state-of-the-art synthesizers. We present two exemplars, one based on Fourier-Motzkin (FM) variable elimination and one based on first-order satisfiability. We demonstrate the practical value of CEGIS($\mathcal{T}$) by automatically synthesizing programs for a set of intricate benchmarks.

CEGIS(T)

CounterExample Guided Inductive Synthesis modulo Theories

CAV 2018

Alessandro Abate[1], Cristina David[2], Pascal Kesseli[3], Daniel Kroening[13], **Elizabeth Polgreen[0]**

[0]University of Edinburgh, [1]University of Oxford, [2]University of Bristol, [3]DiffBlue Ltd

CEGIS(T)

Program synthesis is hard

CEGIS(T)

Program synthesis is hard

CEGIS framework that uses a theory solver to

- verify generalized candidate solutions
- return more general counterexamples

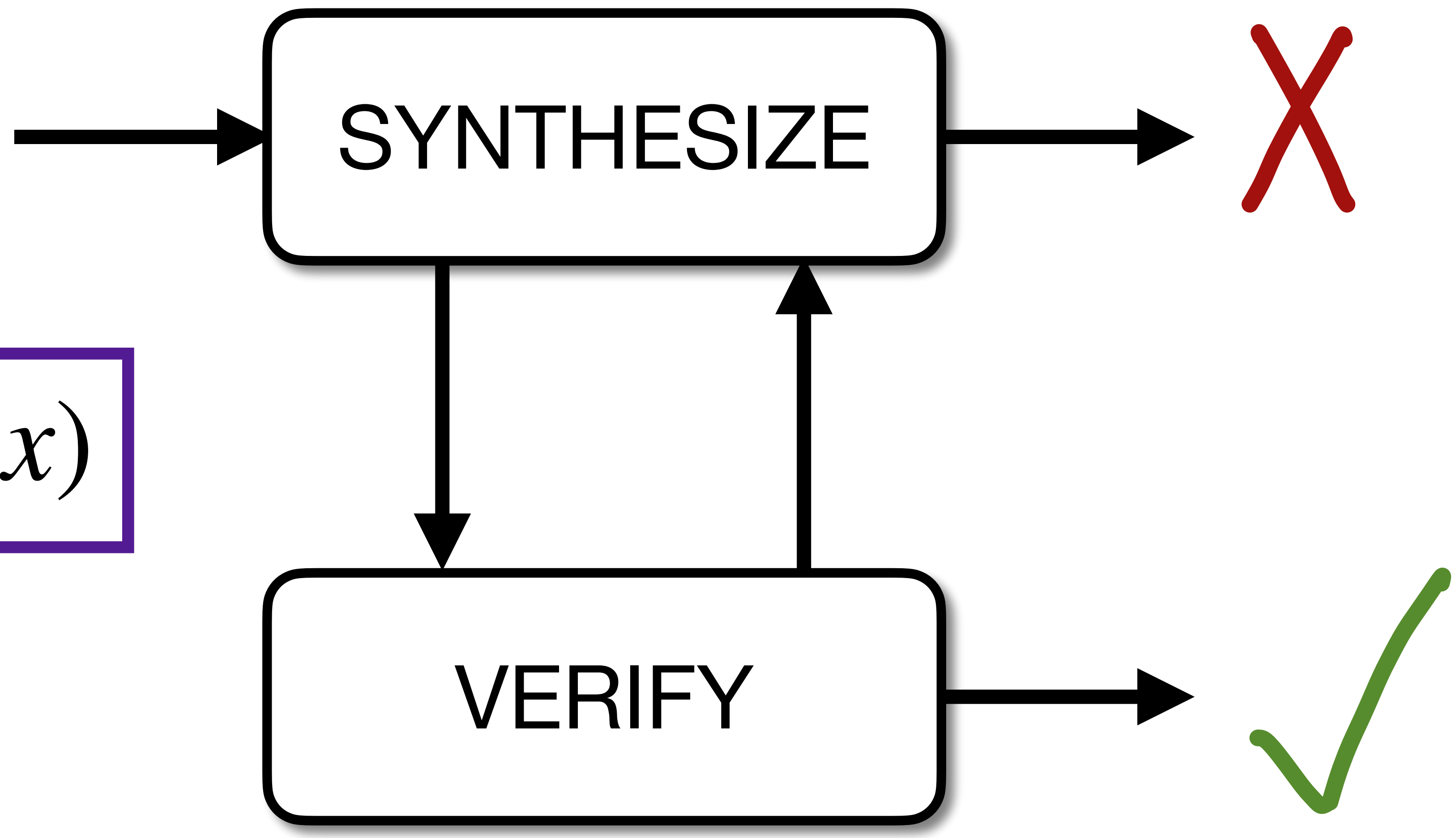
CEGIS(T) is able to synthesize programs containing arbitrary constants that elude other solvers.

Outline

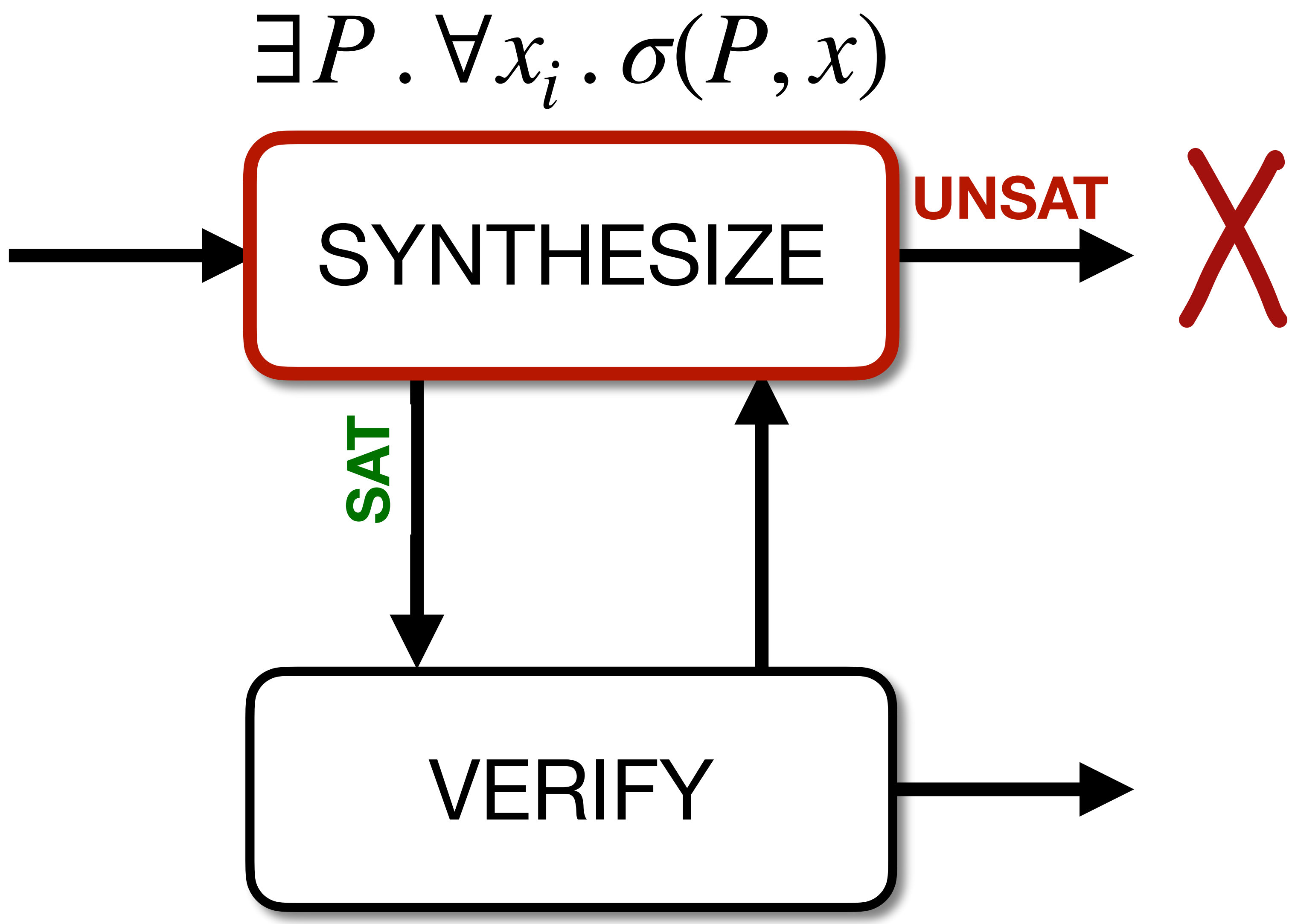
- **Overview of CEGIS and motivation for CEGIS(T)**
- CEGIS(T): algorithm in detail
- Evaluation
- CEGIS(T) in CVC4
- Ongoing work: beyond constants

CEGIS

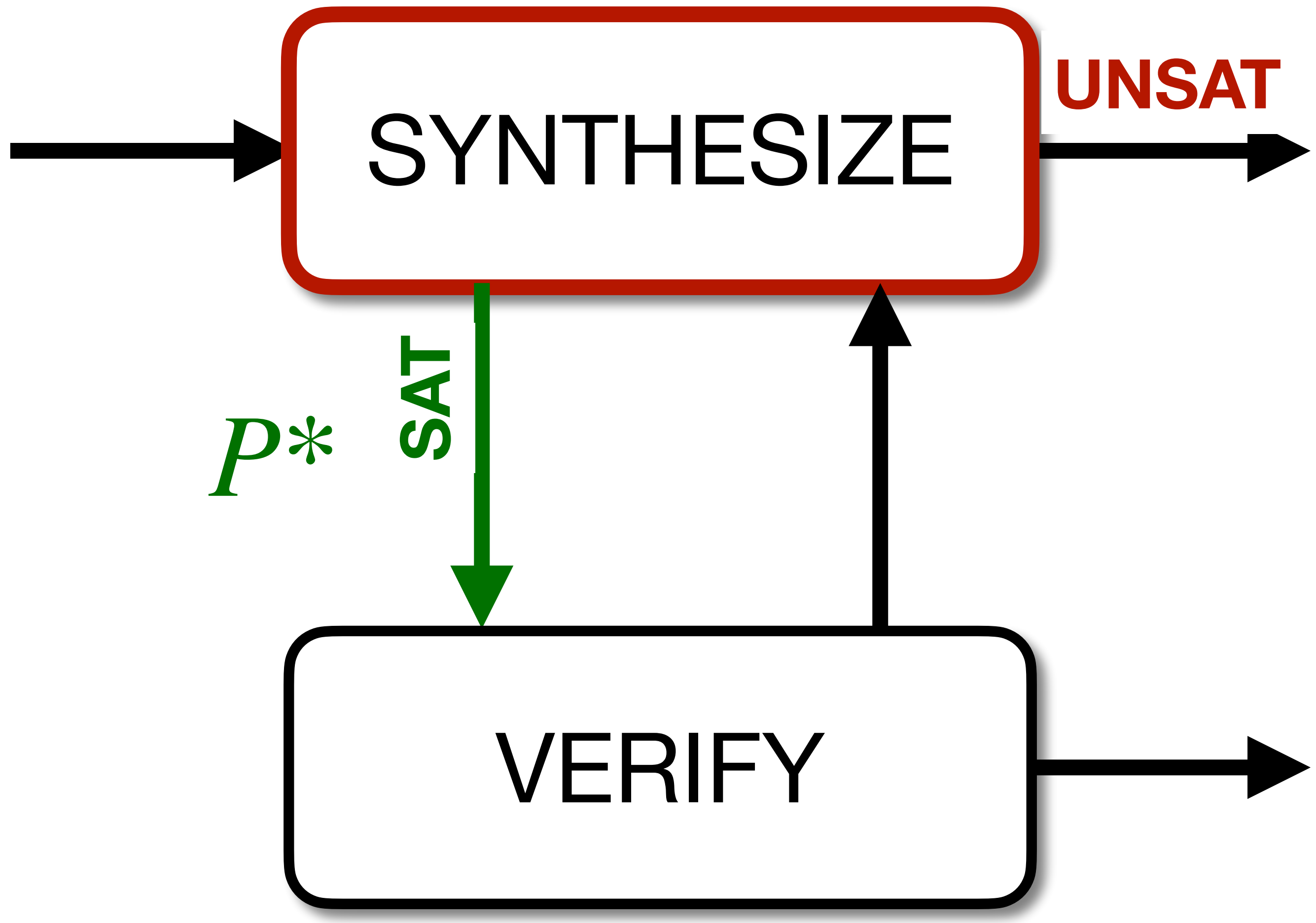
$$\boxed{\exists P \forall x \,.\, \sigma(P, x)}$$



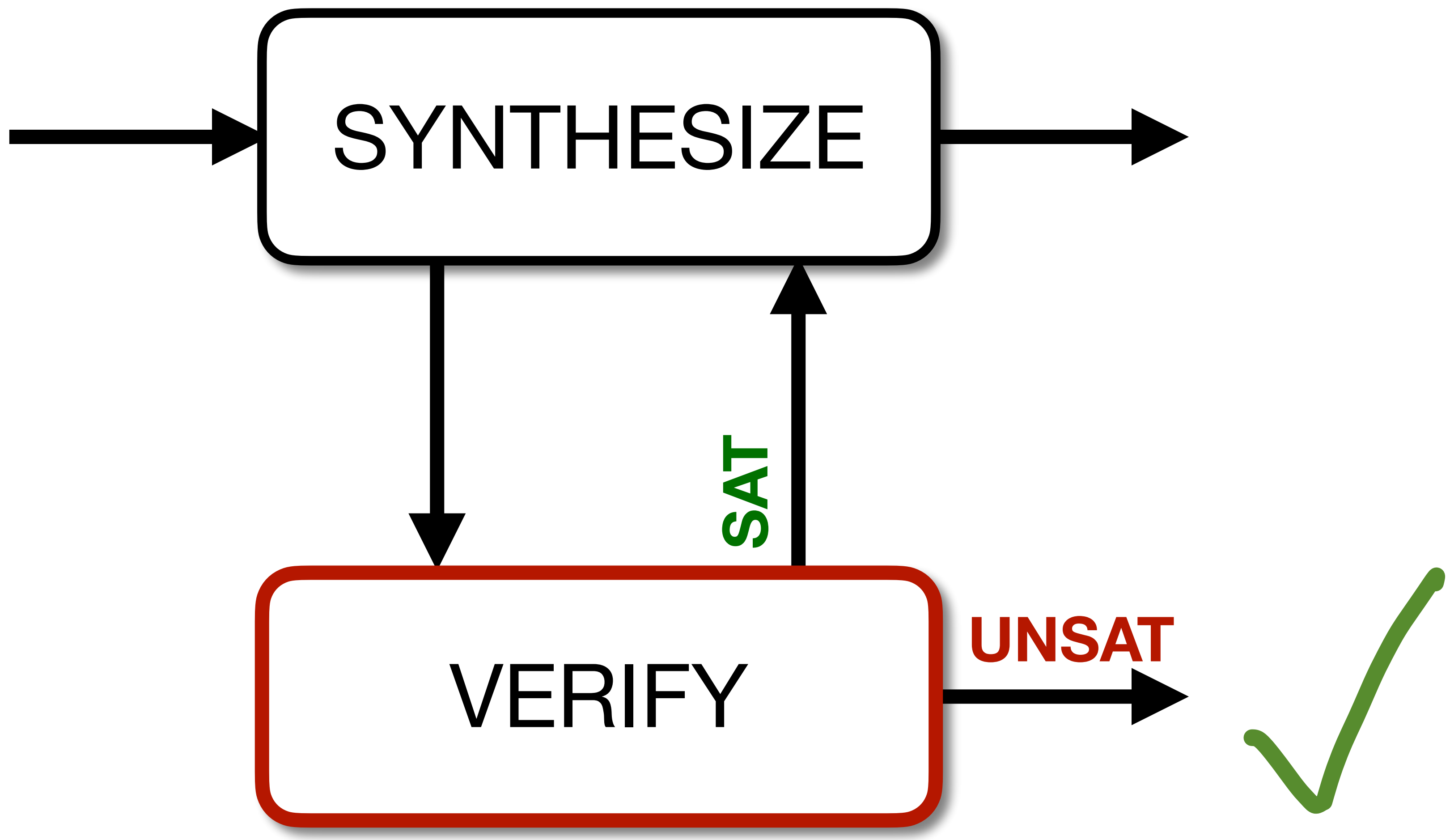
CEGIS



CEGIS

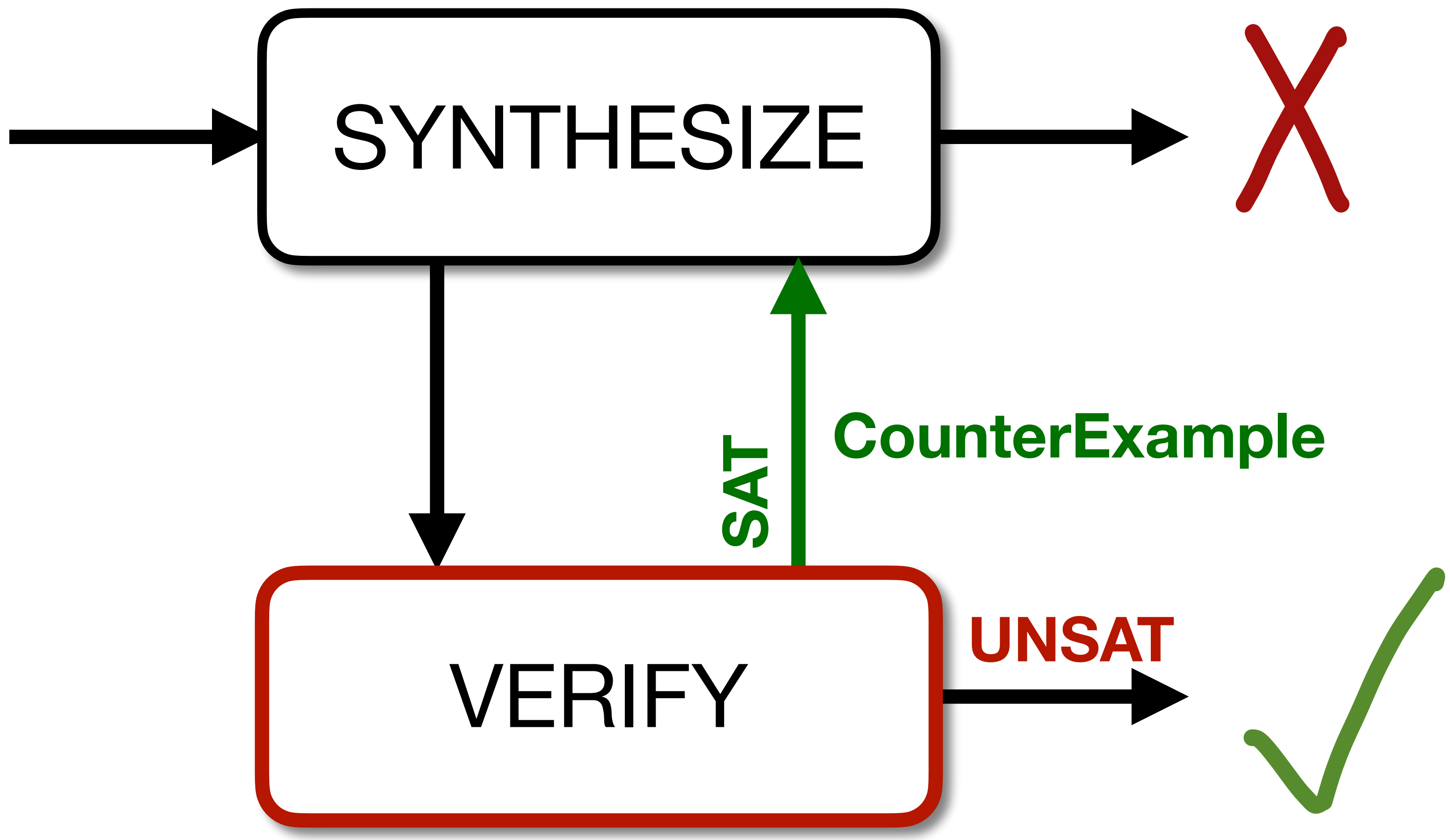


CEGIS



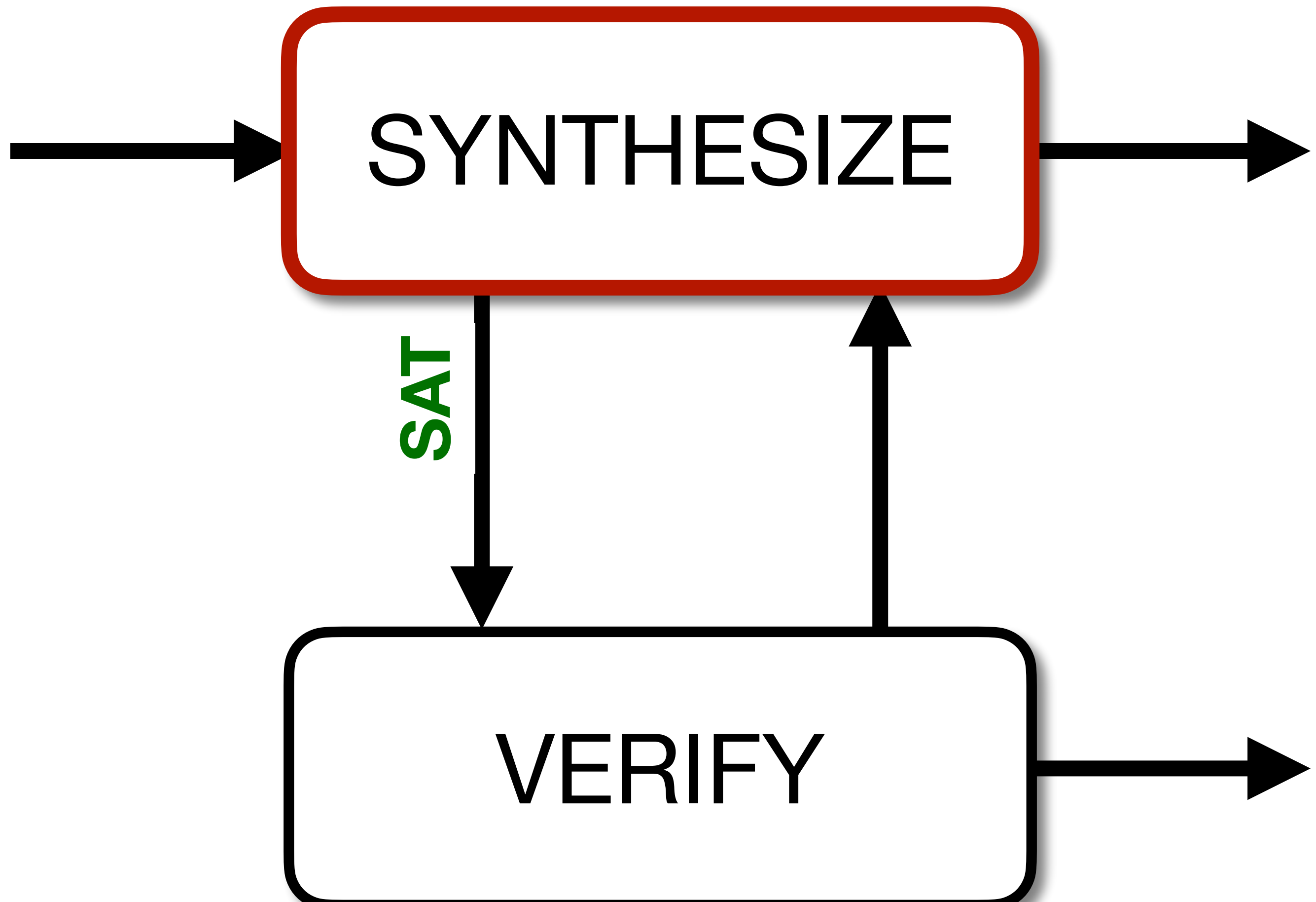
$$\exists x\,.\,\neg\sigma(P^*, x)$$

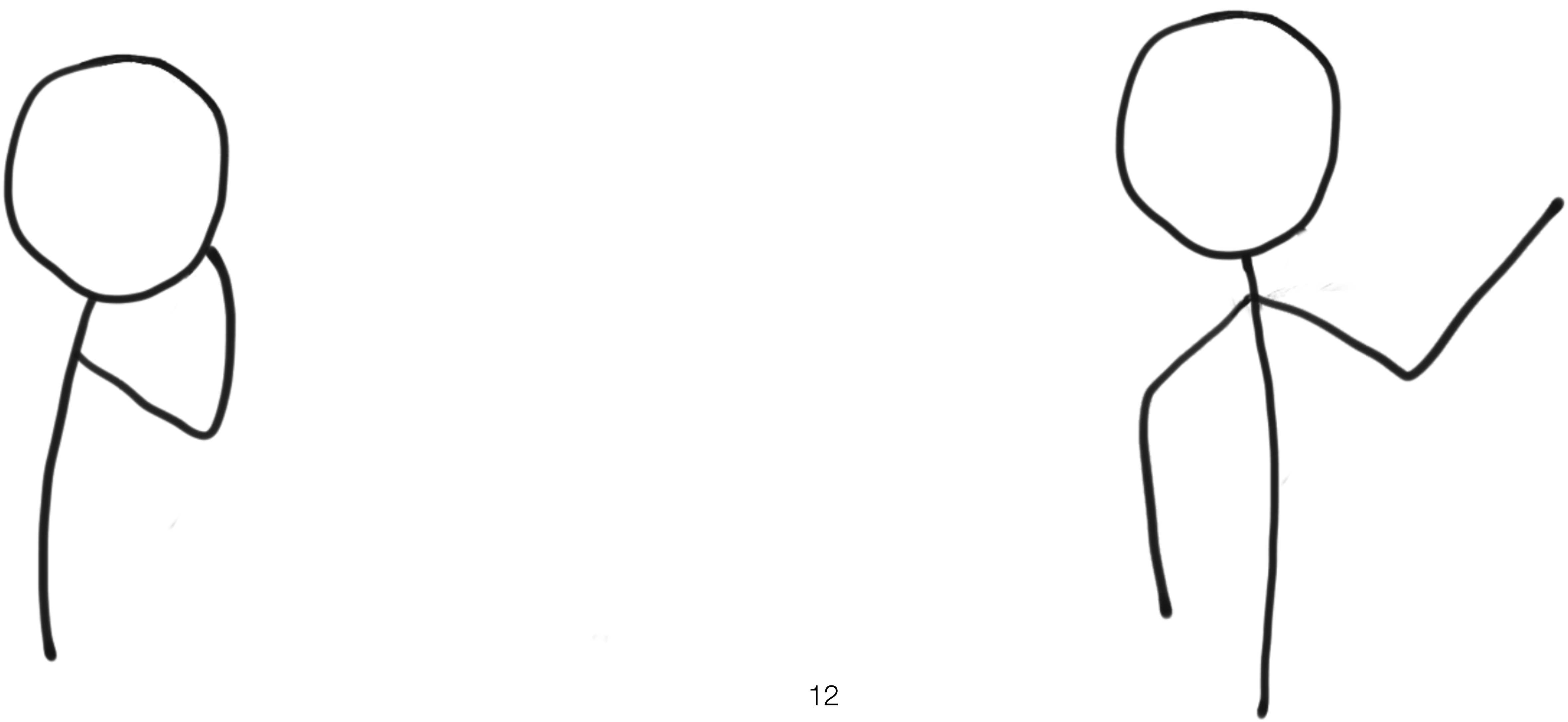
CEGIS

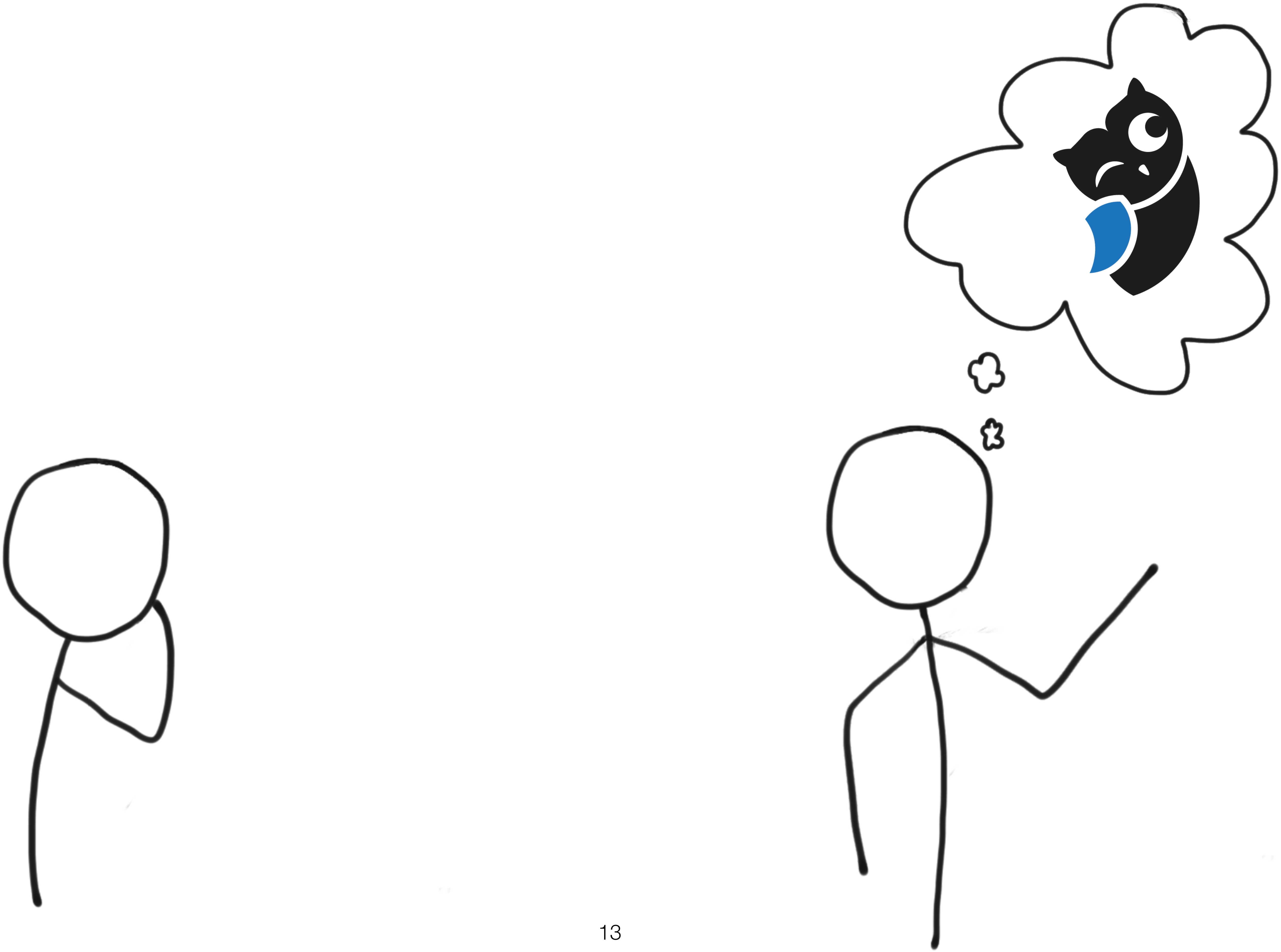


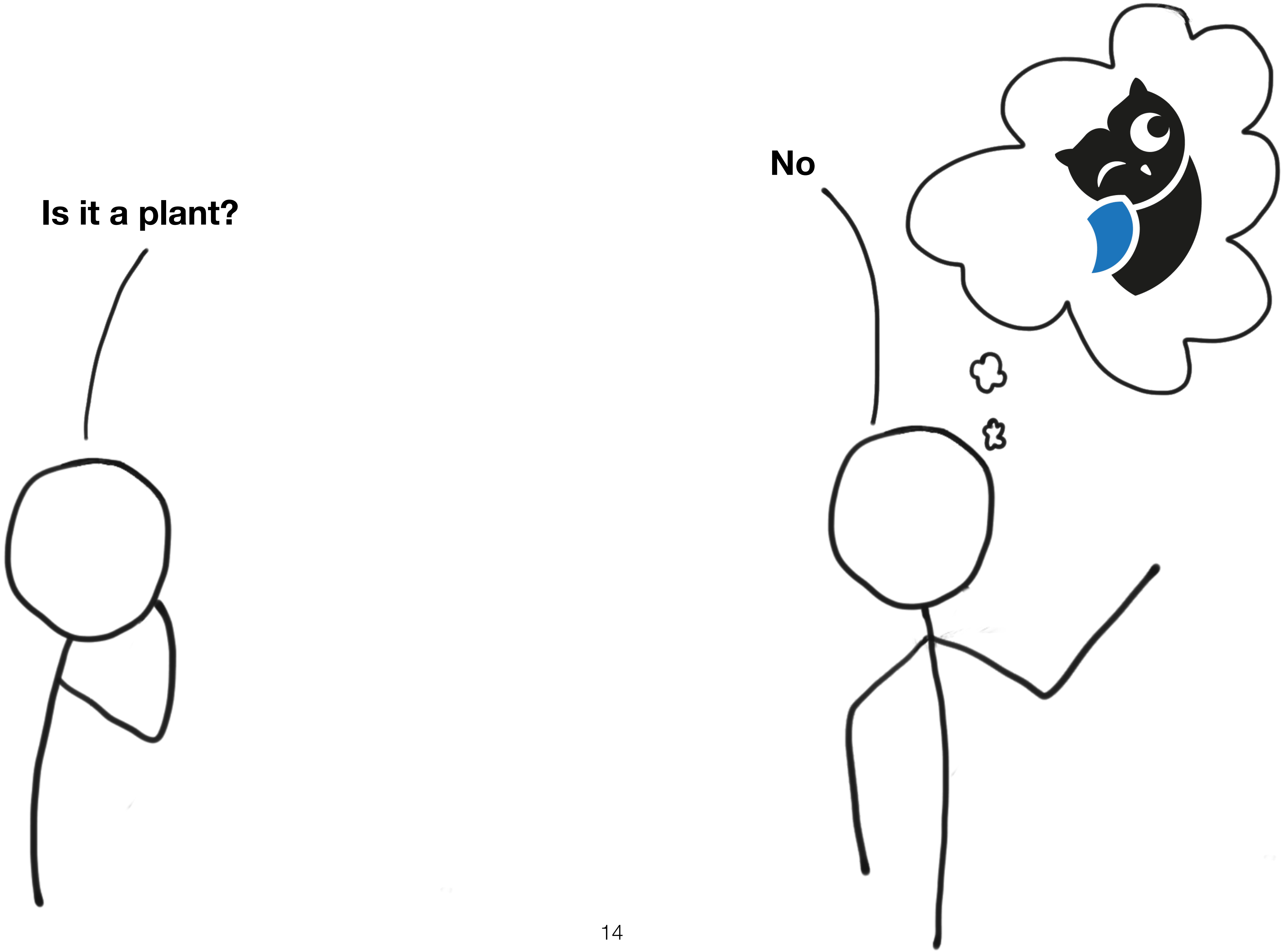
CEGIS

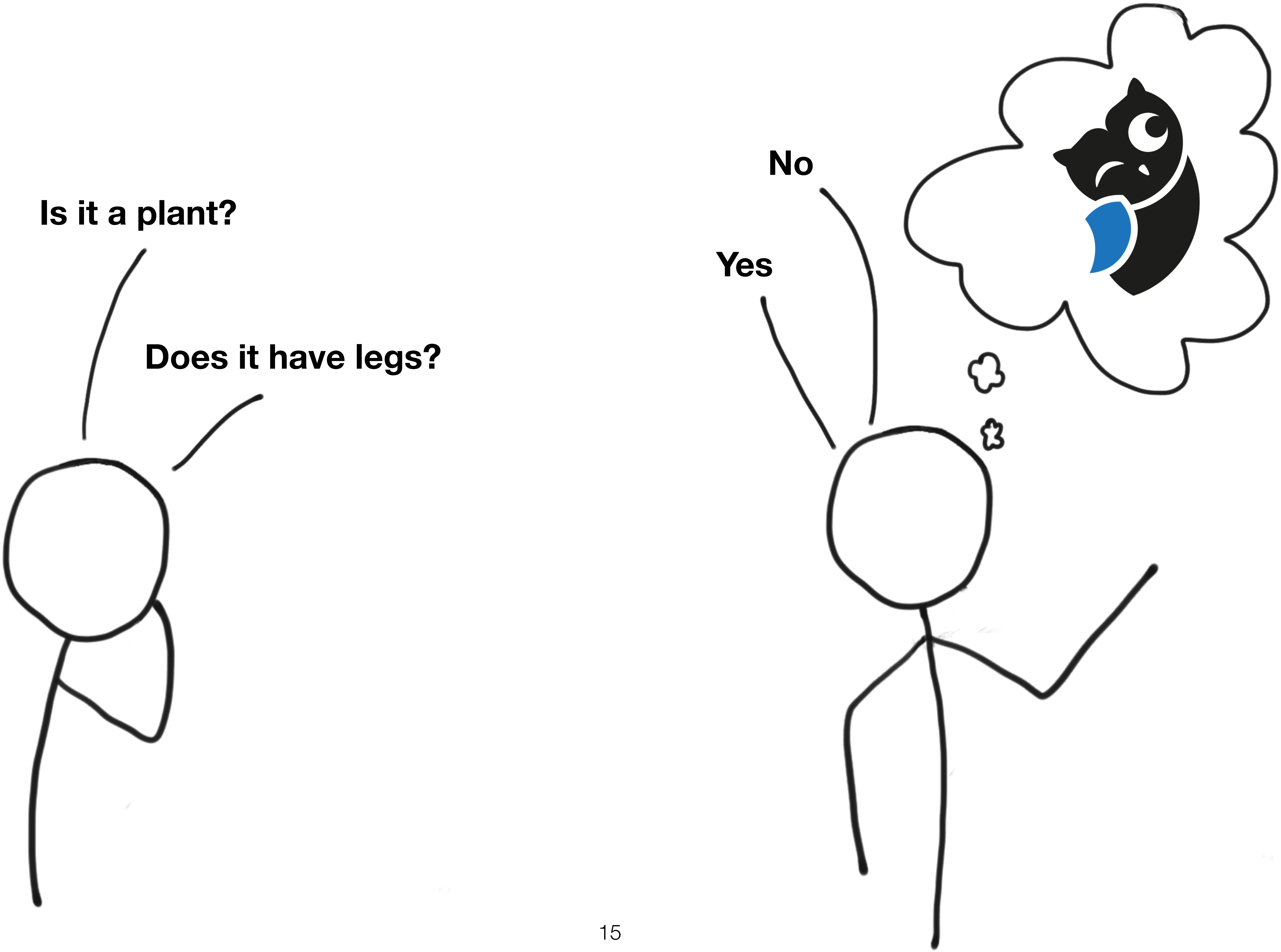
$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$

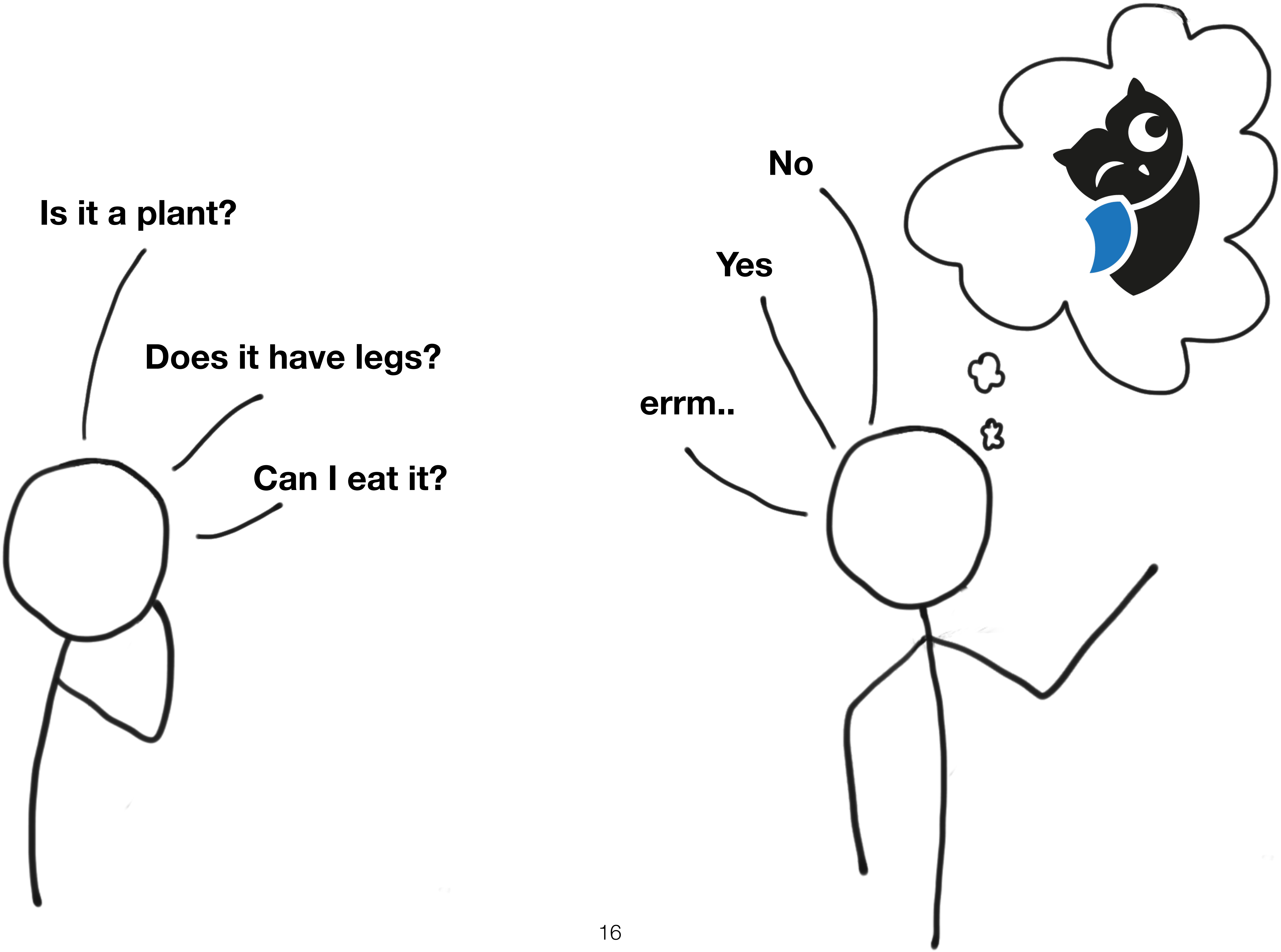


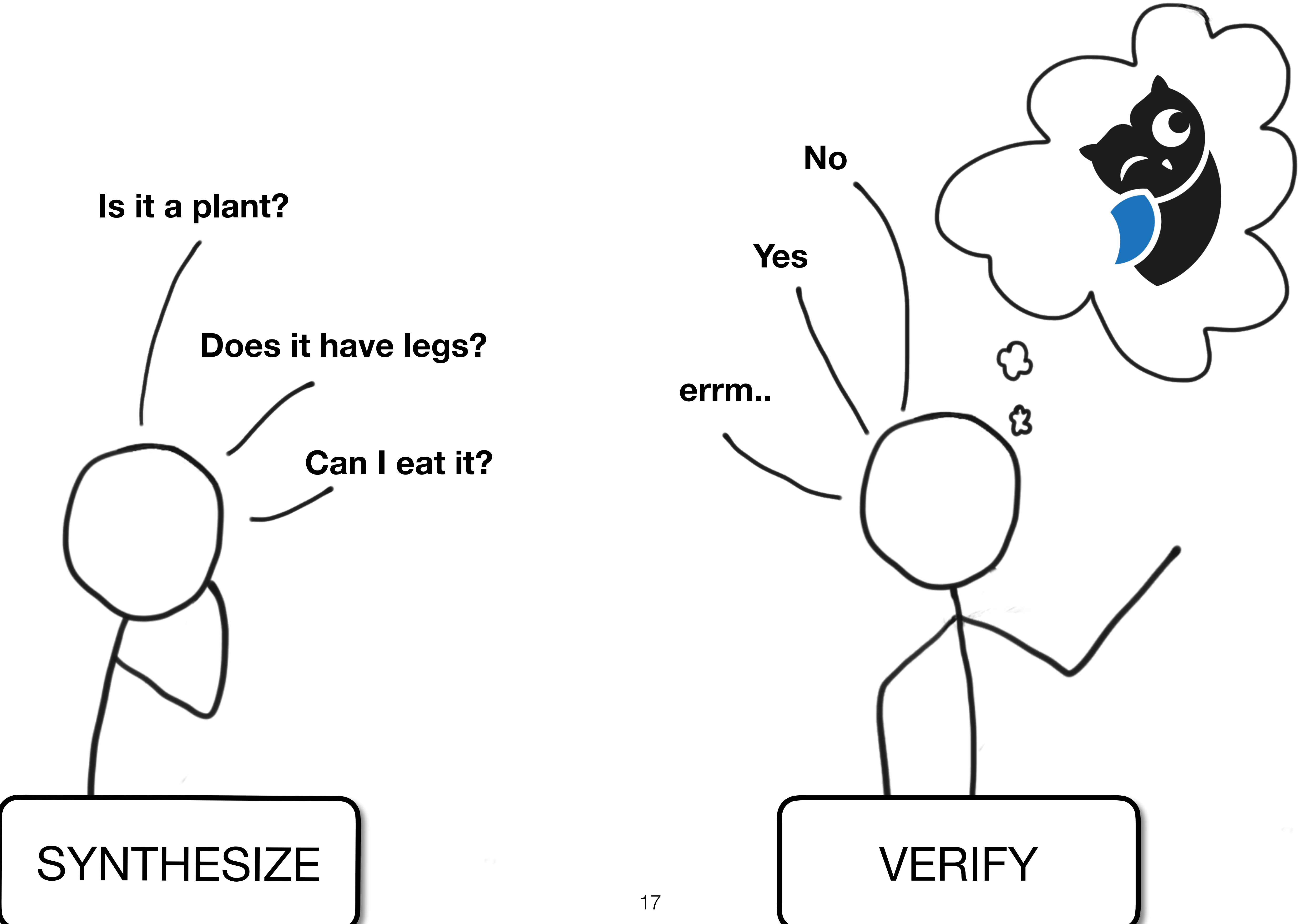




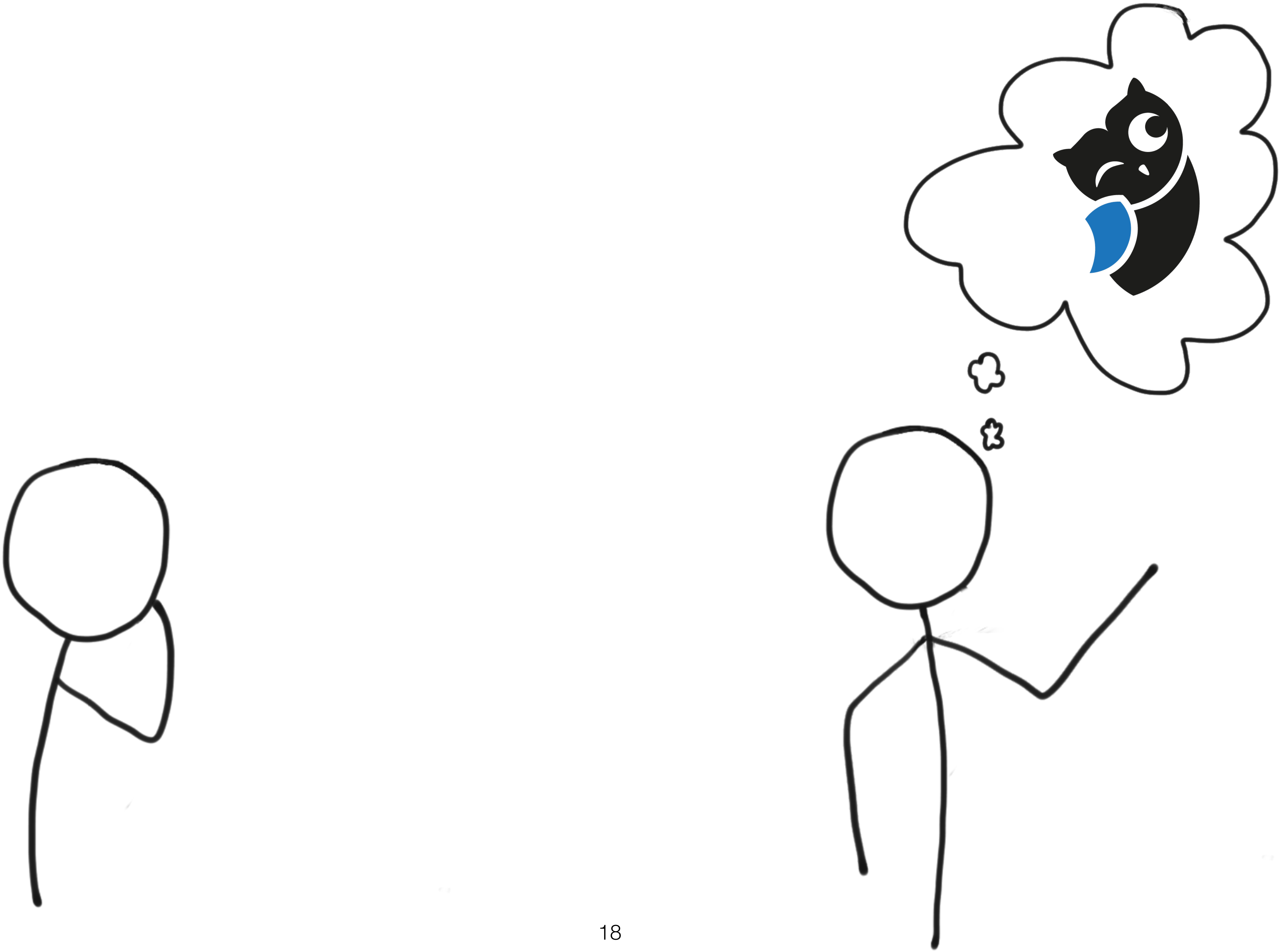

Is it a plant?
No

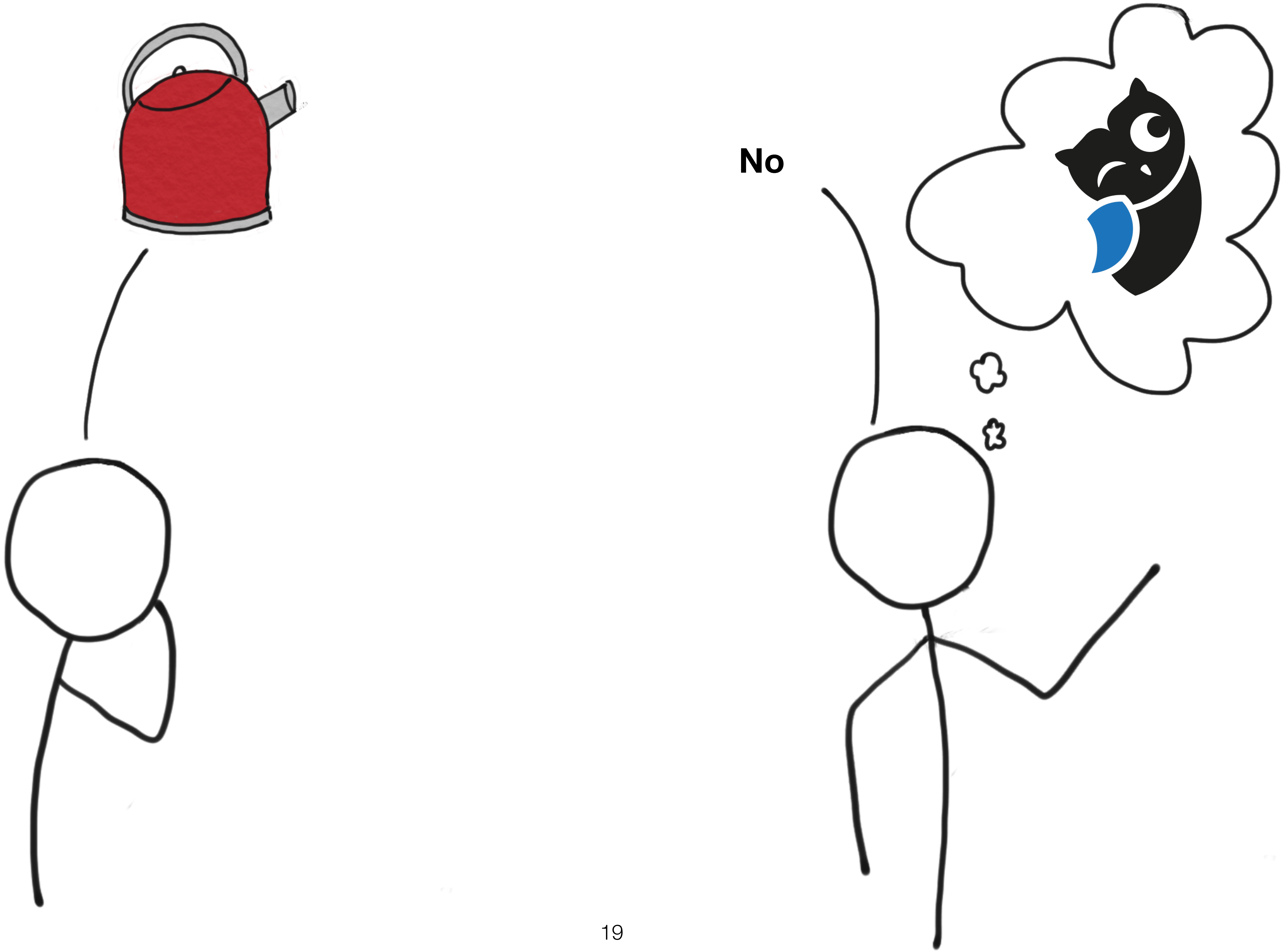

Is it a plant?
Does it have legs?
No
Yes

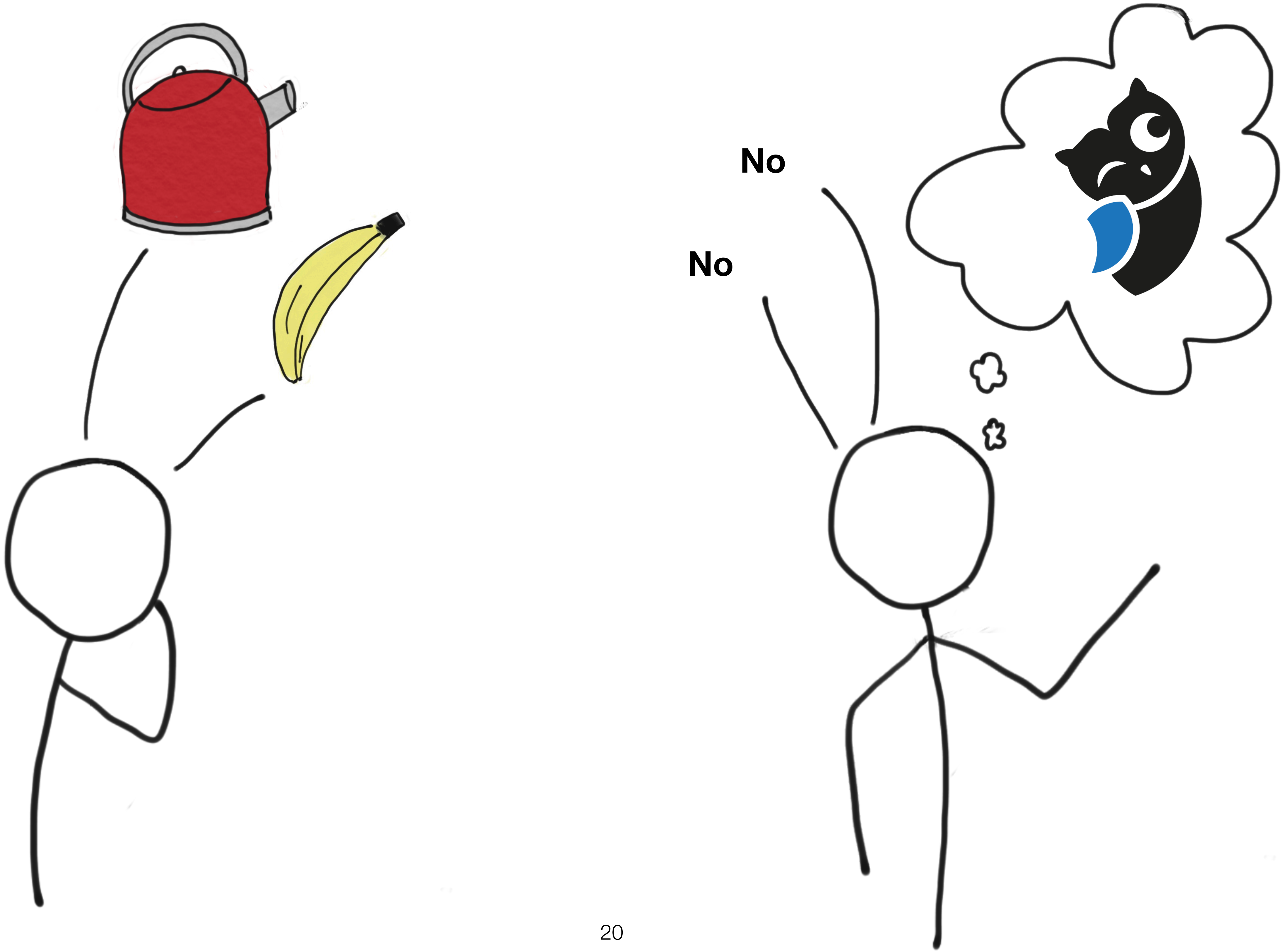

Is it a plant?
Does it have legs?
Can I eat it?
No
Yes
errm..

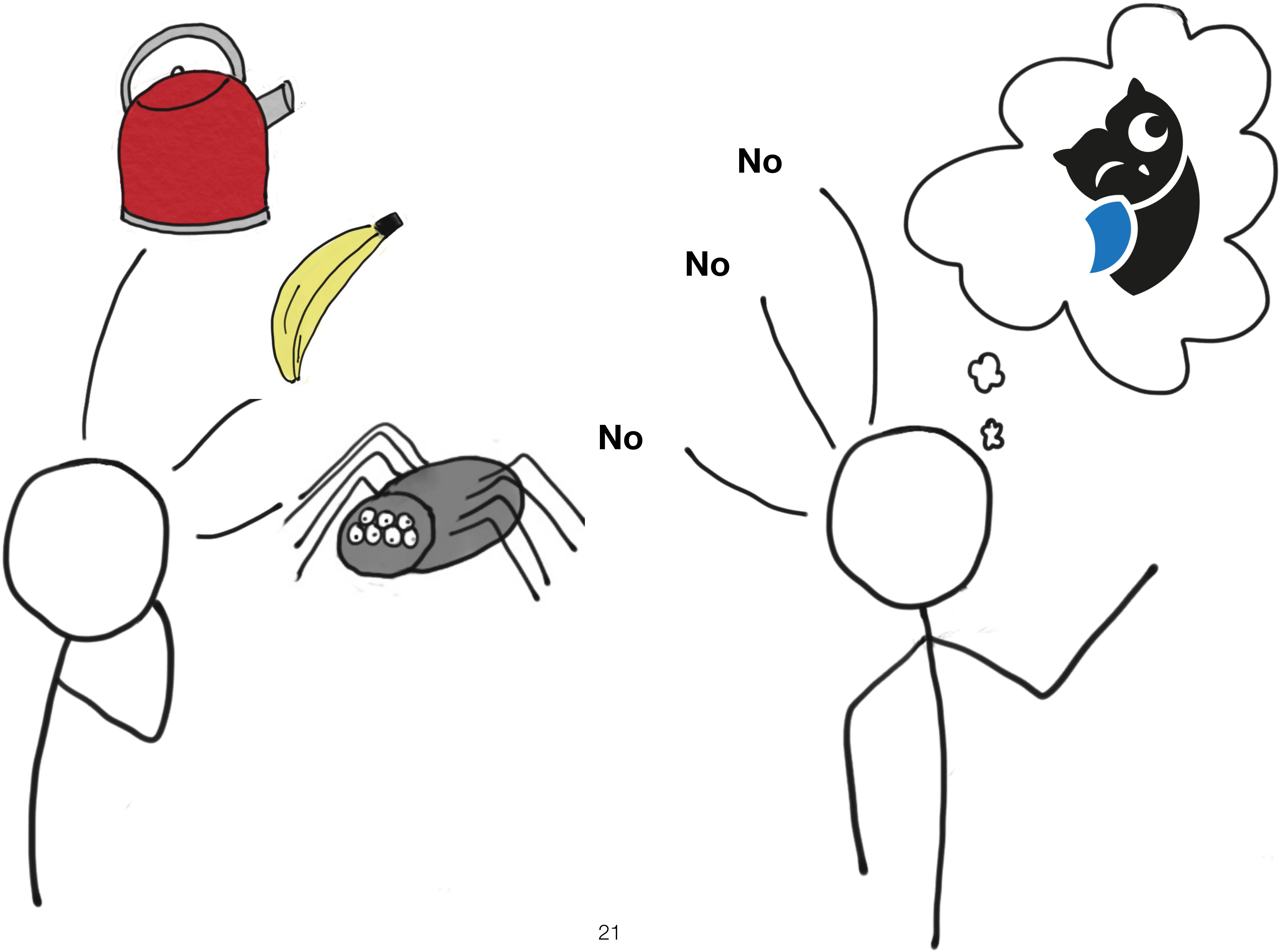


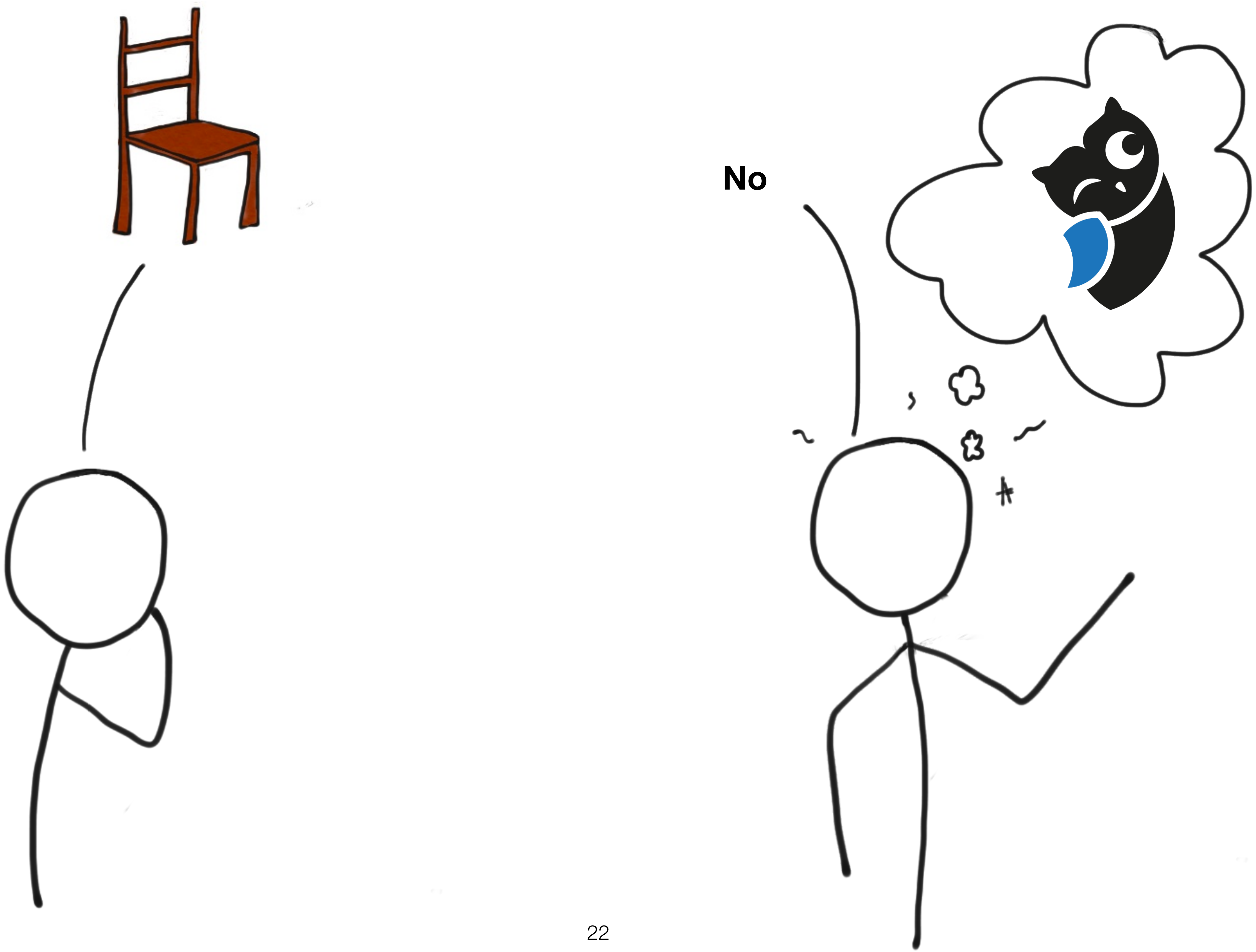
Is it a plant?
Does it have legs?
Can I eat it?
No
Yes
errm..
SYNTHESIZE
VERIFY

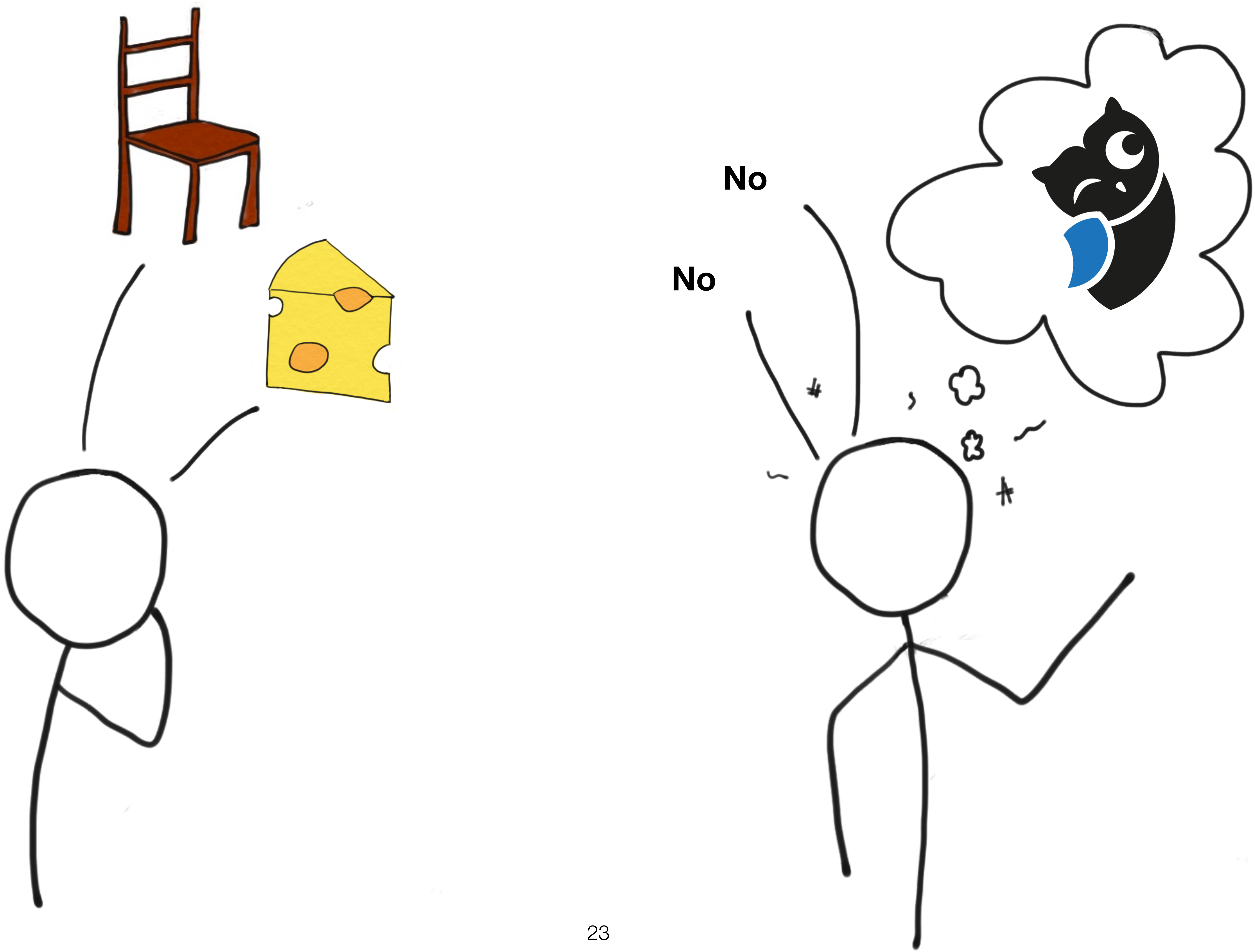


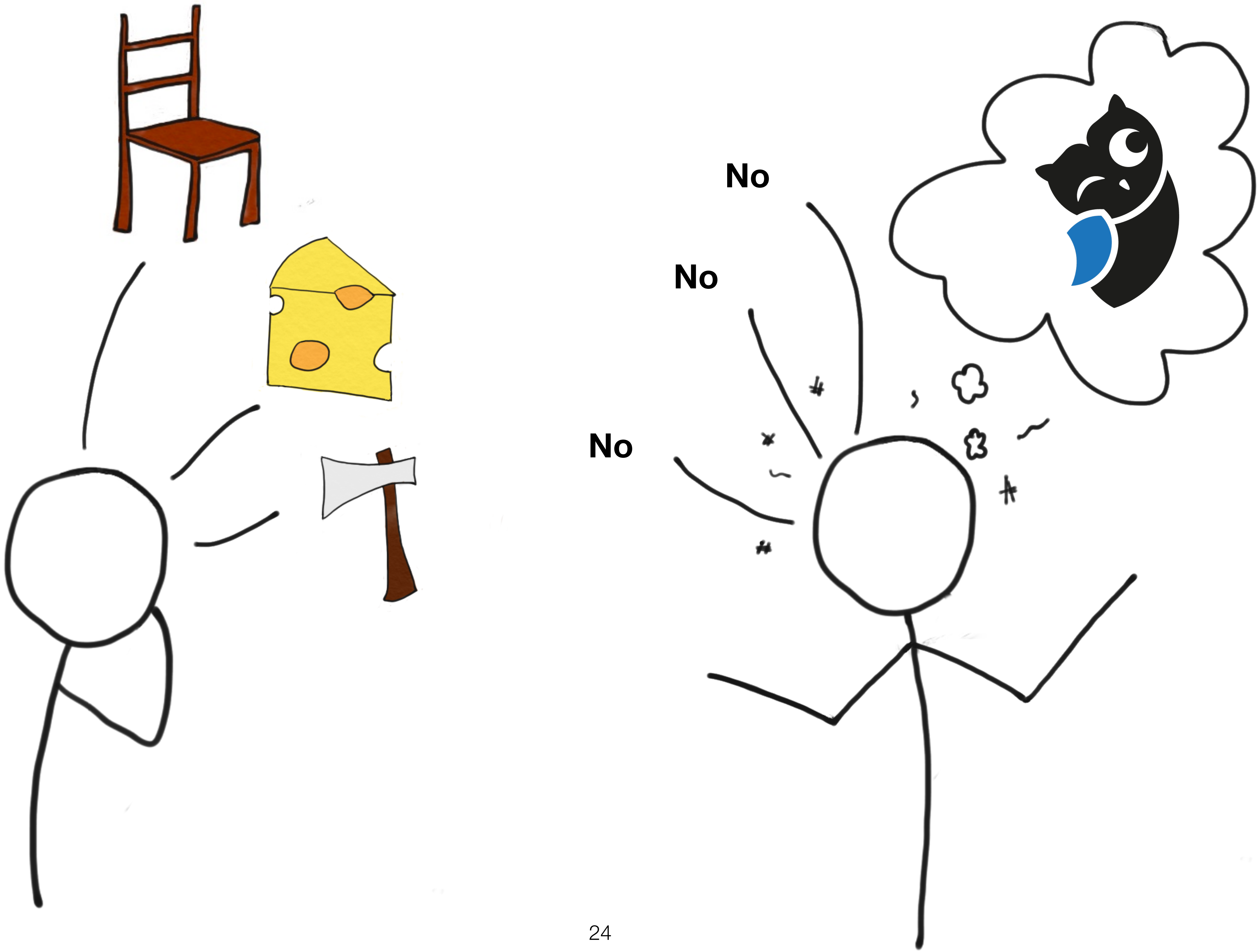

No


No
No


No
No
No


No


No
No


No
No
No

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:**

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

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

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

Safety invariant

```
int x = 5;

while ( x < 1000)
  x++;

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

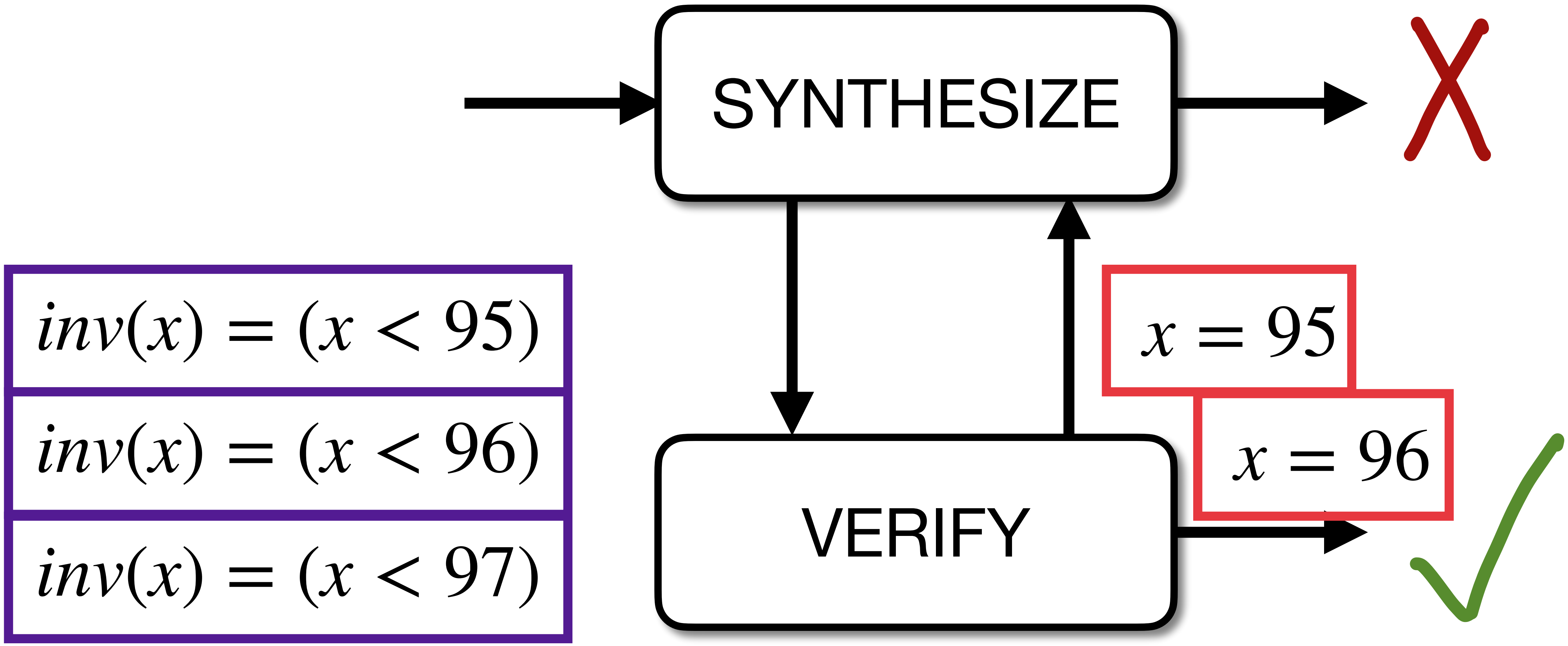
$$init(x) \iff x = 0$$

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

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

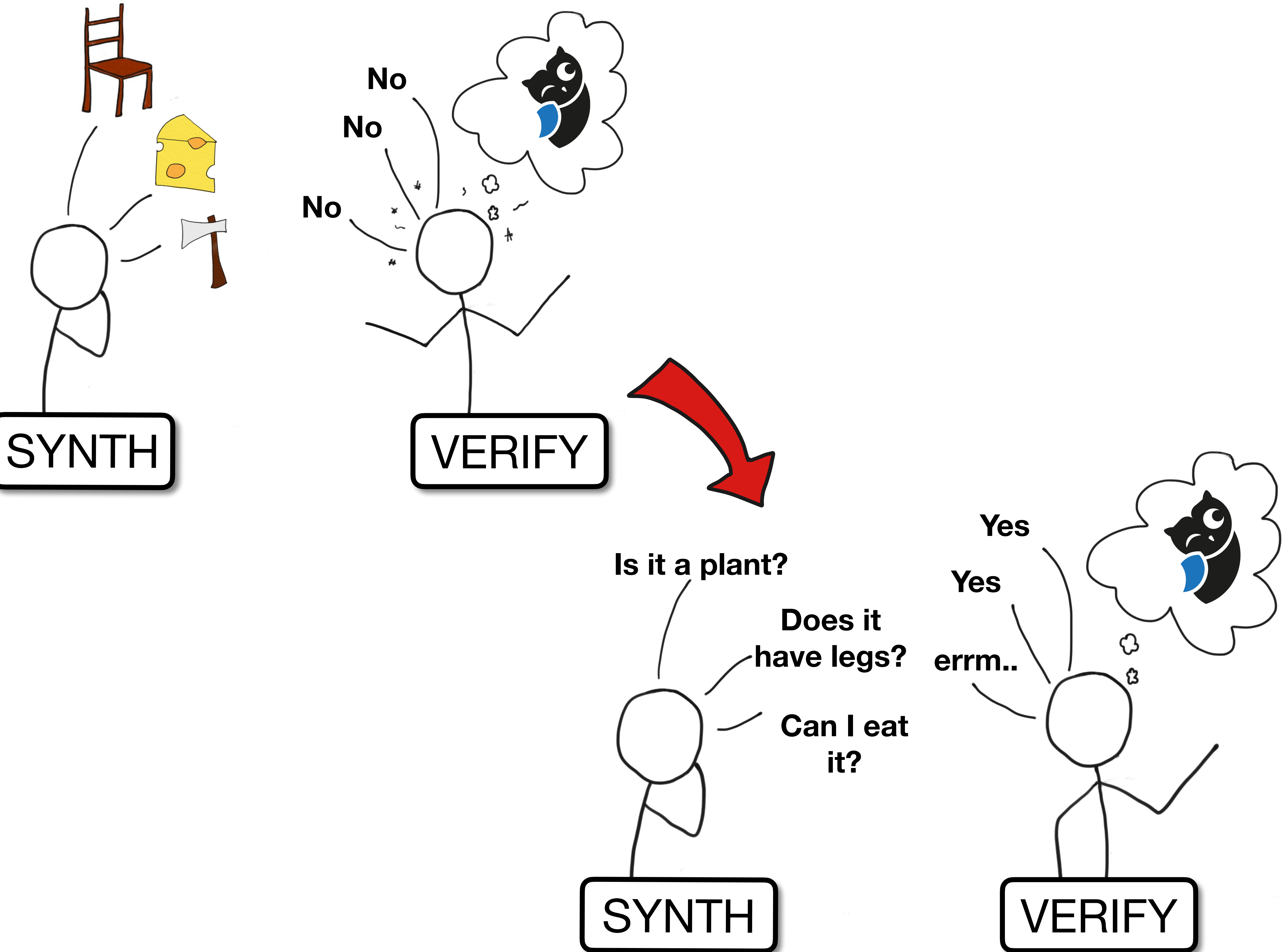
Possible solution:

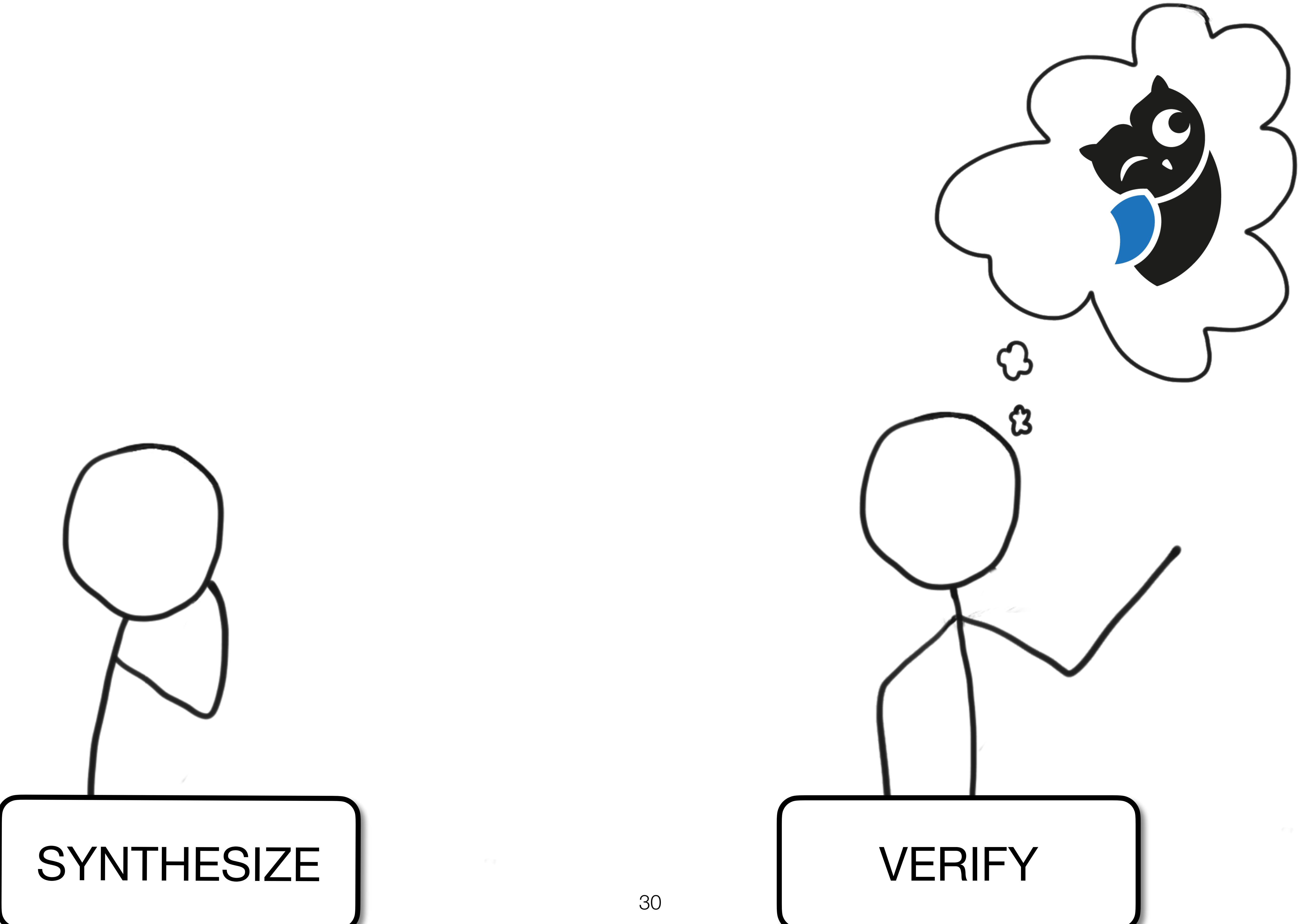
$$inv(x) = (4 < x) \wedge (x < 1003)$$

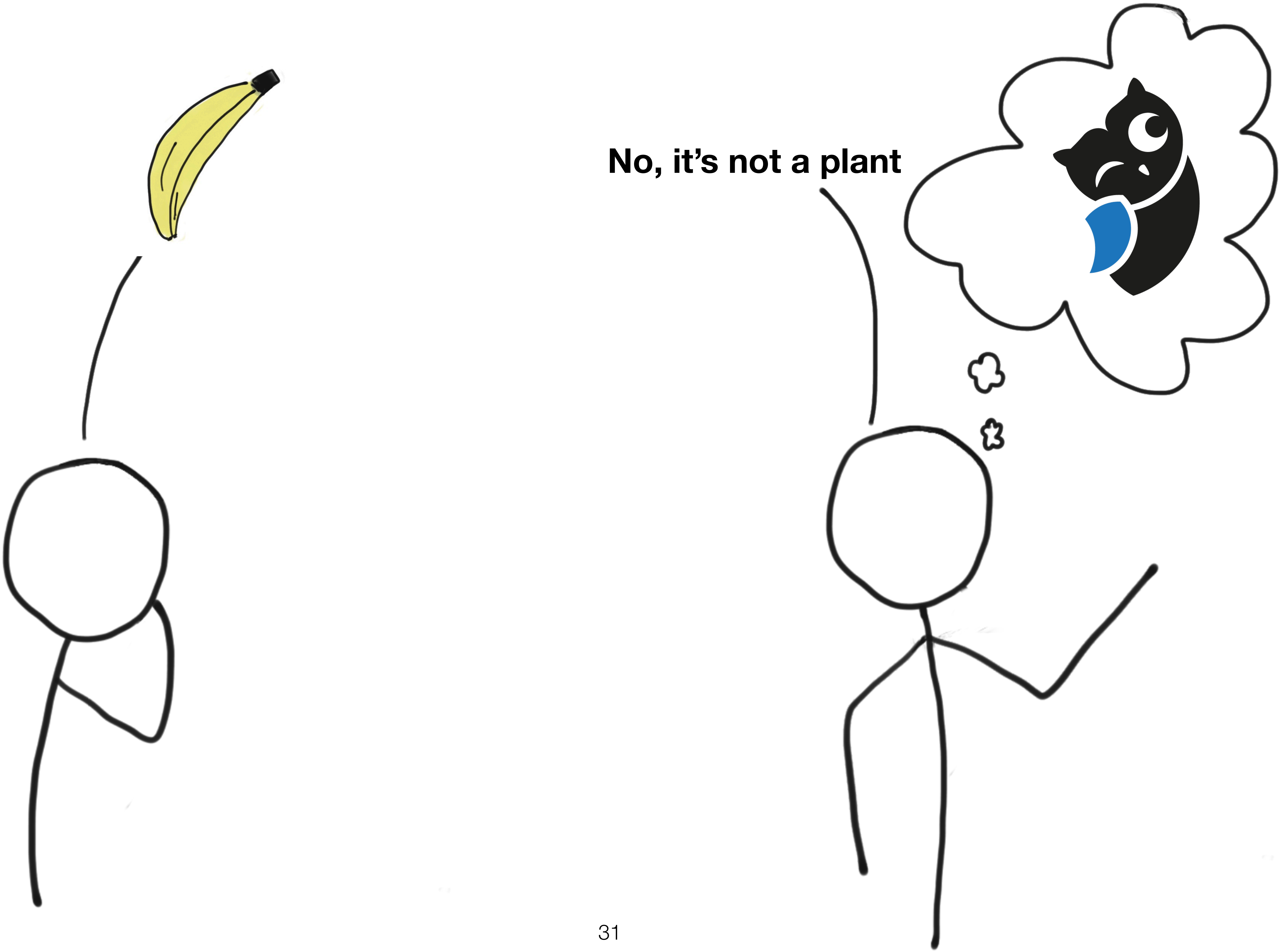


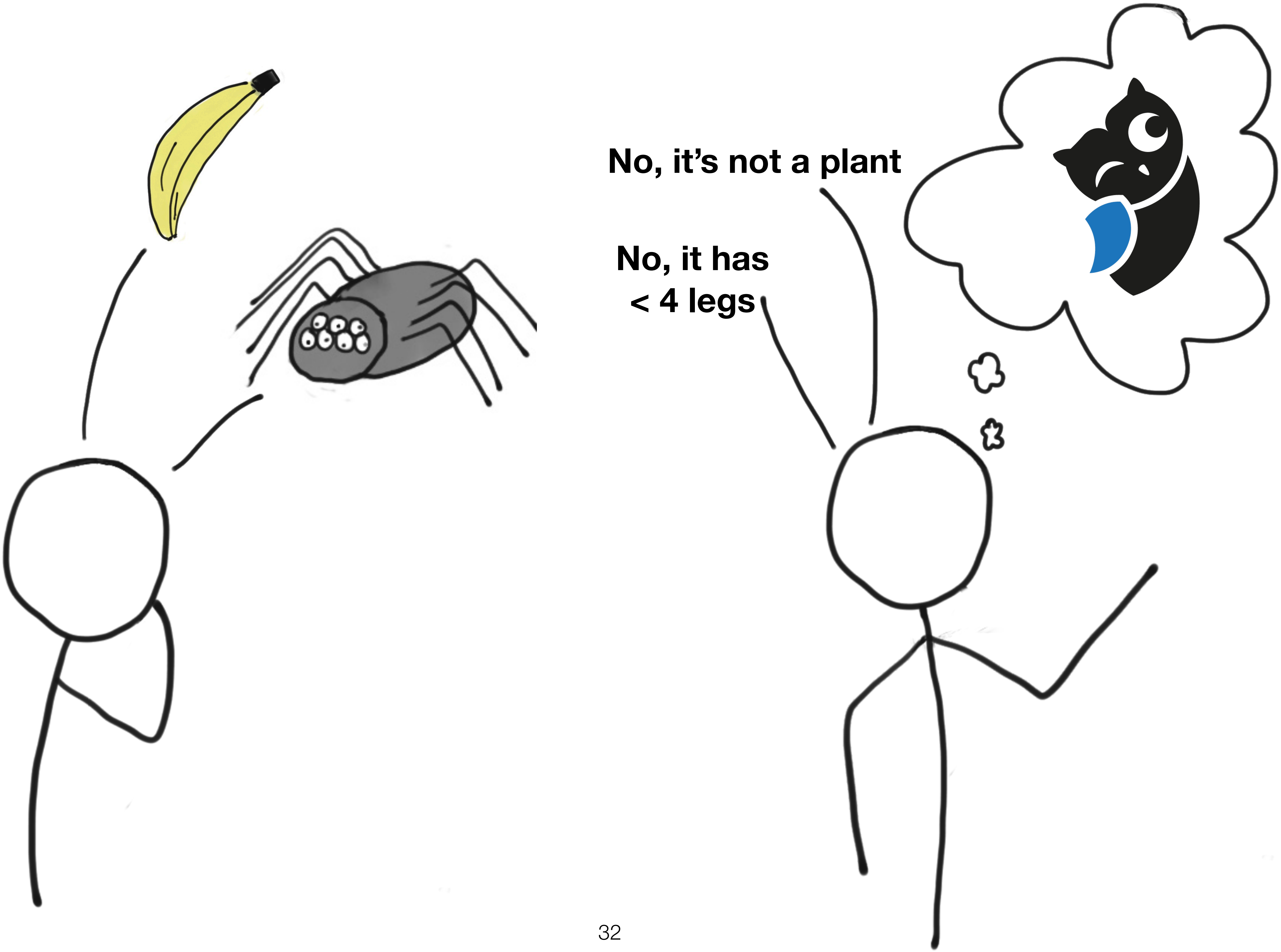
And so on ..

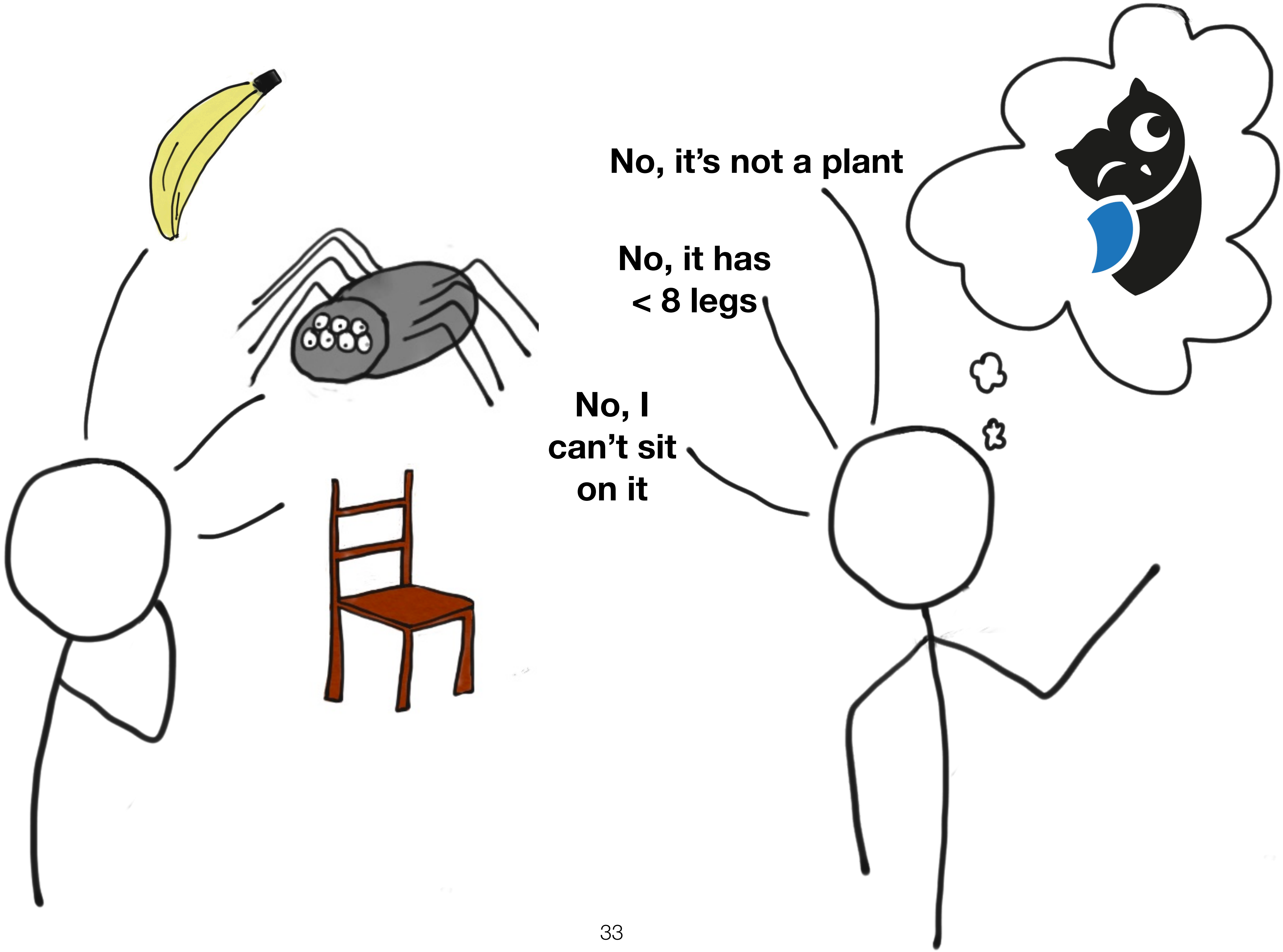
Can we ask more general questions?


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


SYNTHESIZE
VERIFY


No, it’s not a plant


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


No, it's not a plant
No, it has
< 8 legs
No, I
can't sit
on it

Can we give more general answers?

More general questions

+

More general answers

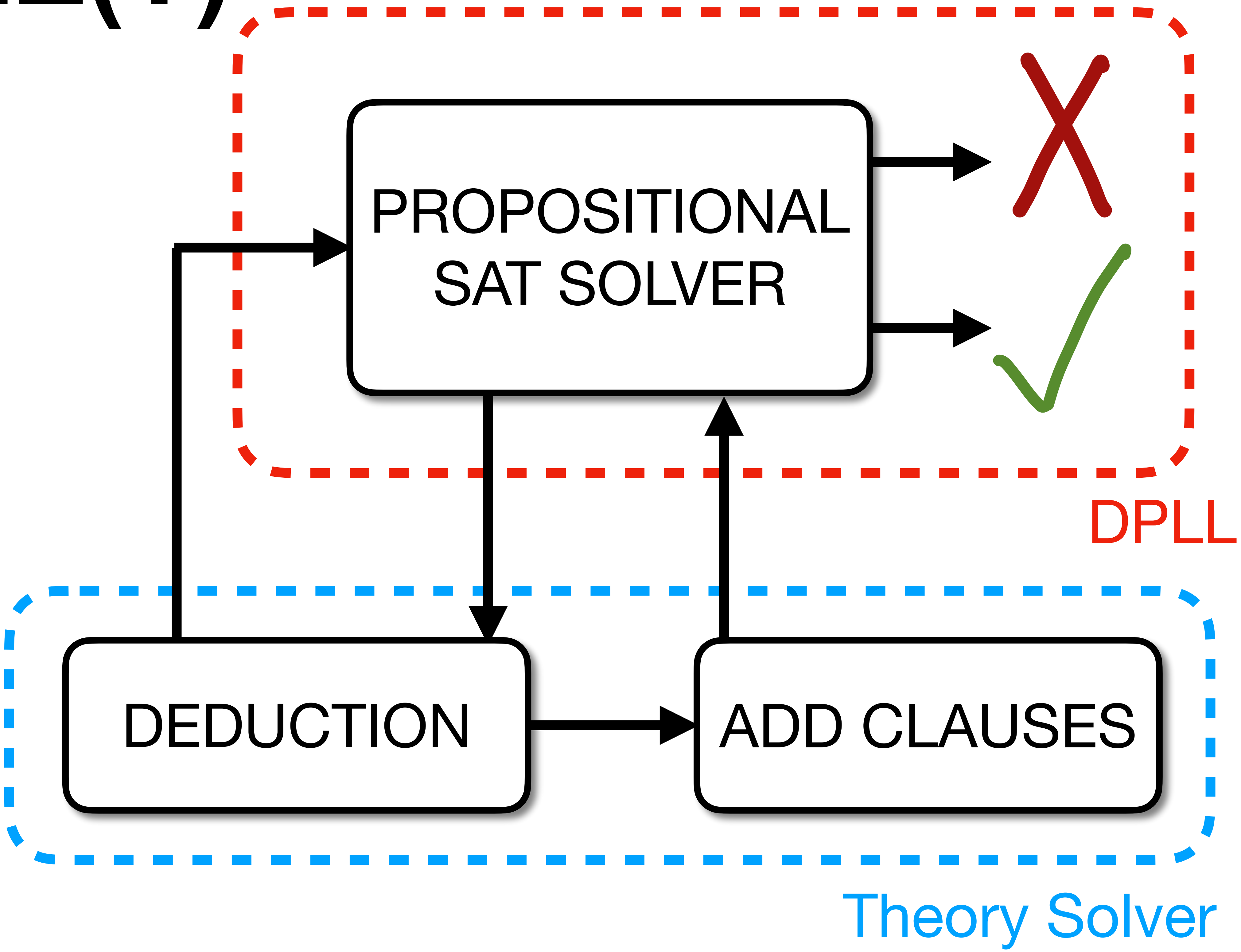
=

CEGIS(T)

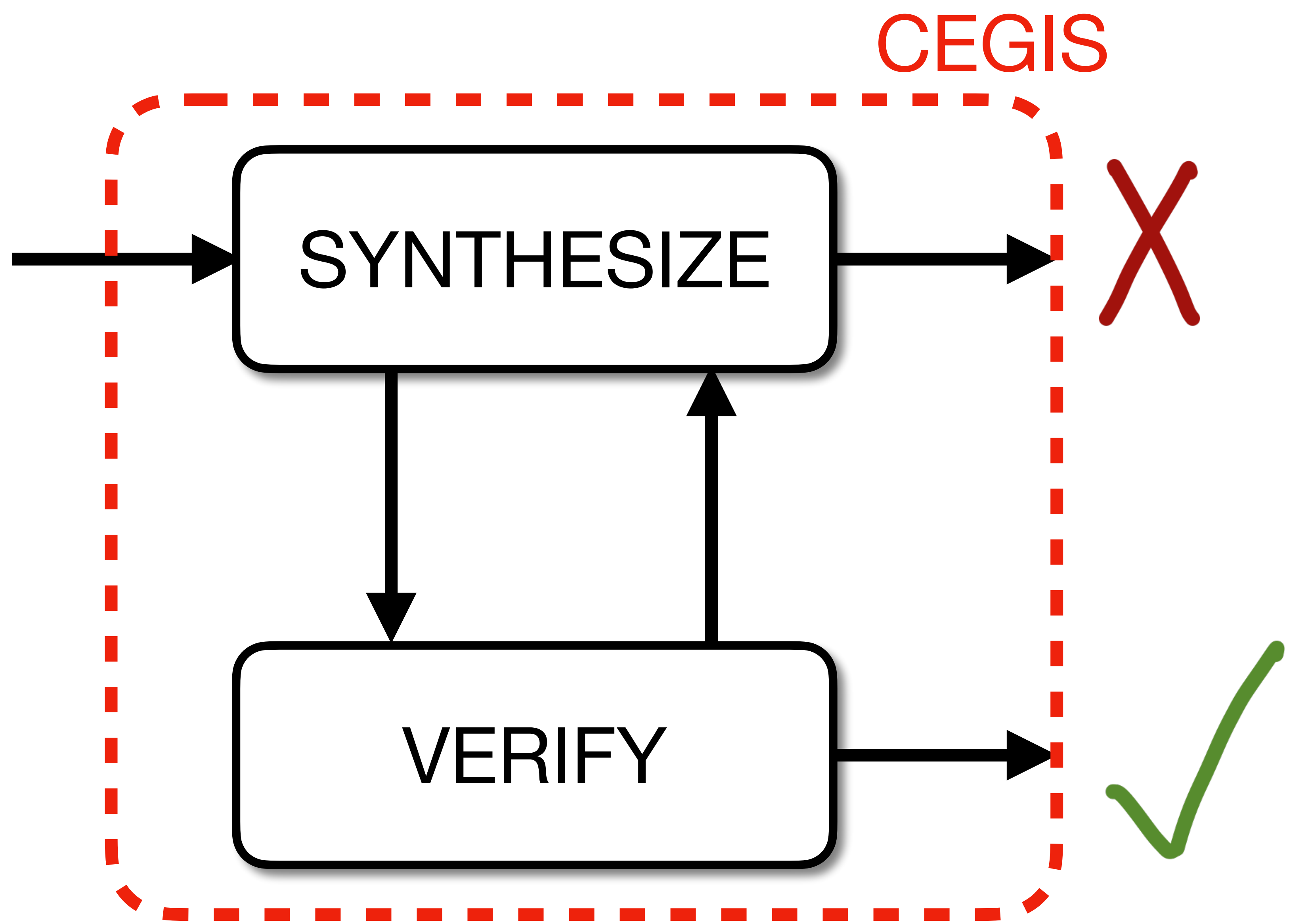
Outline

- Overview of CEGIS and motivation for CEGIS(T)
- **CEGIS(T): algorithm in detail**
- Evaluation
- CEGIS(T) in CVC4
- Ongoing work: beyond constants

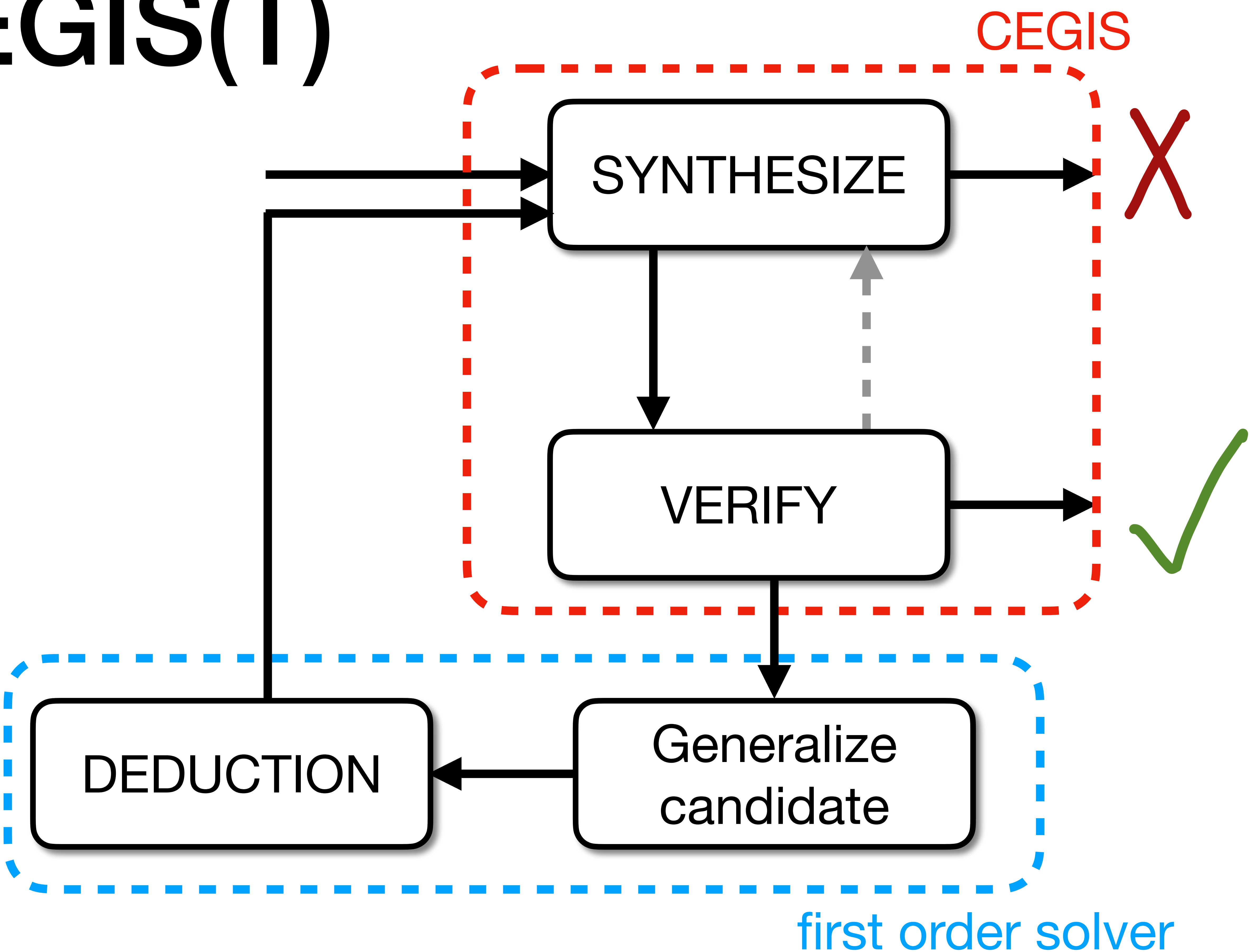
DPLL(T)


PROPOSITIONAL
SAT SOLVER
DPLL
DEDUCTION
ADD CLAUSES
Theory Solver

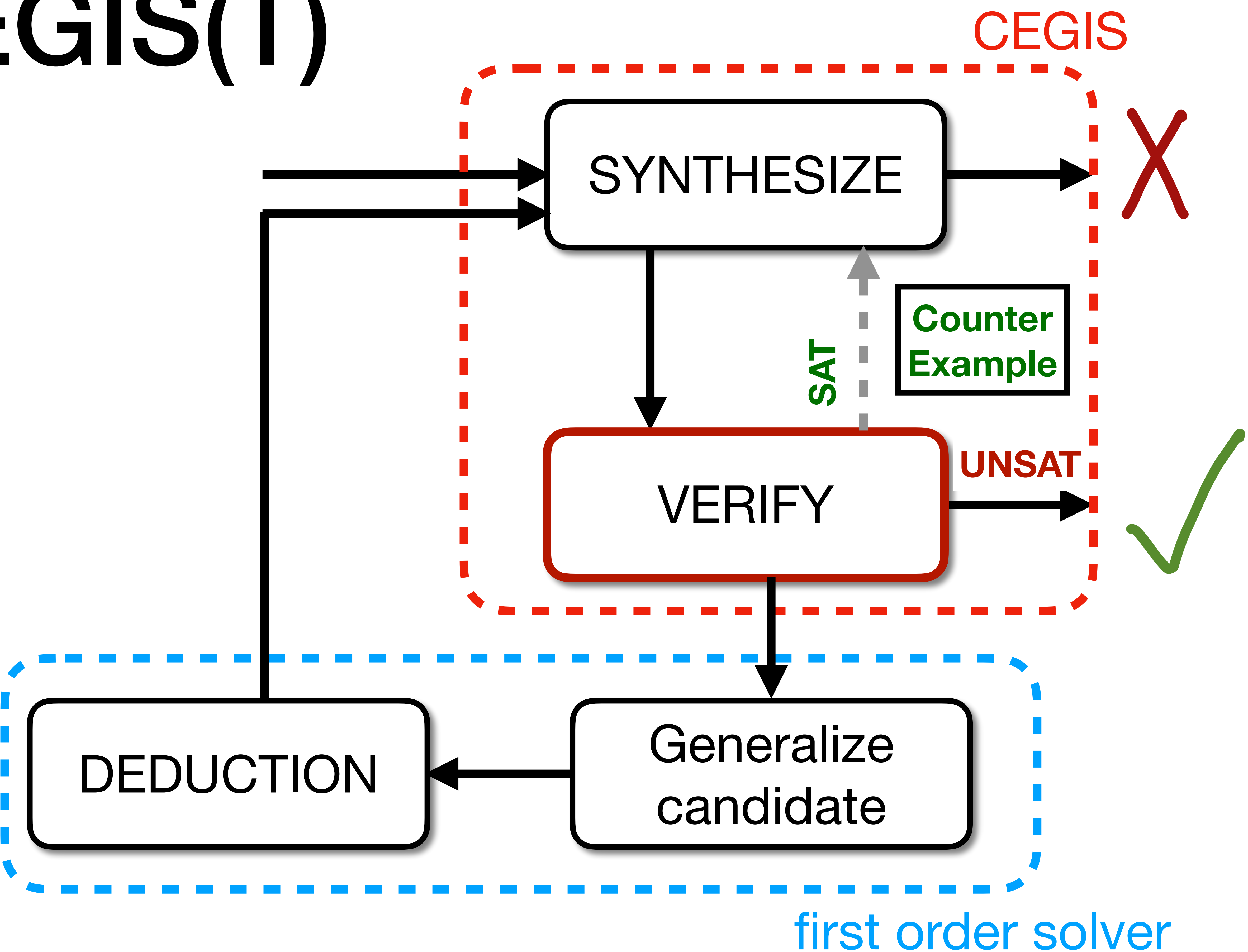
CEGIS(T)



CEGIS(T)

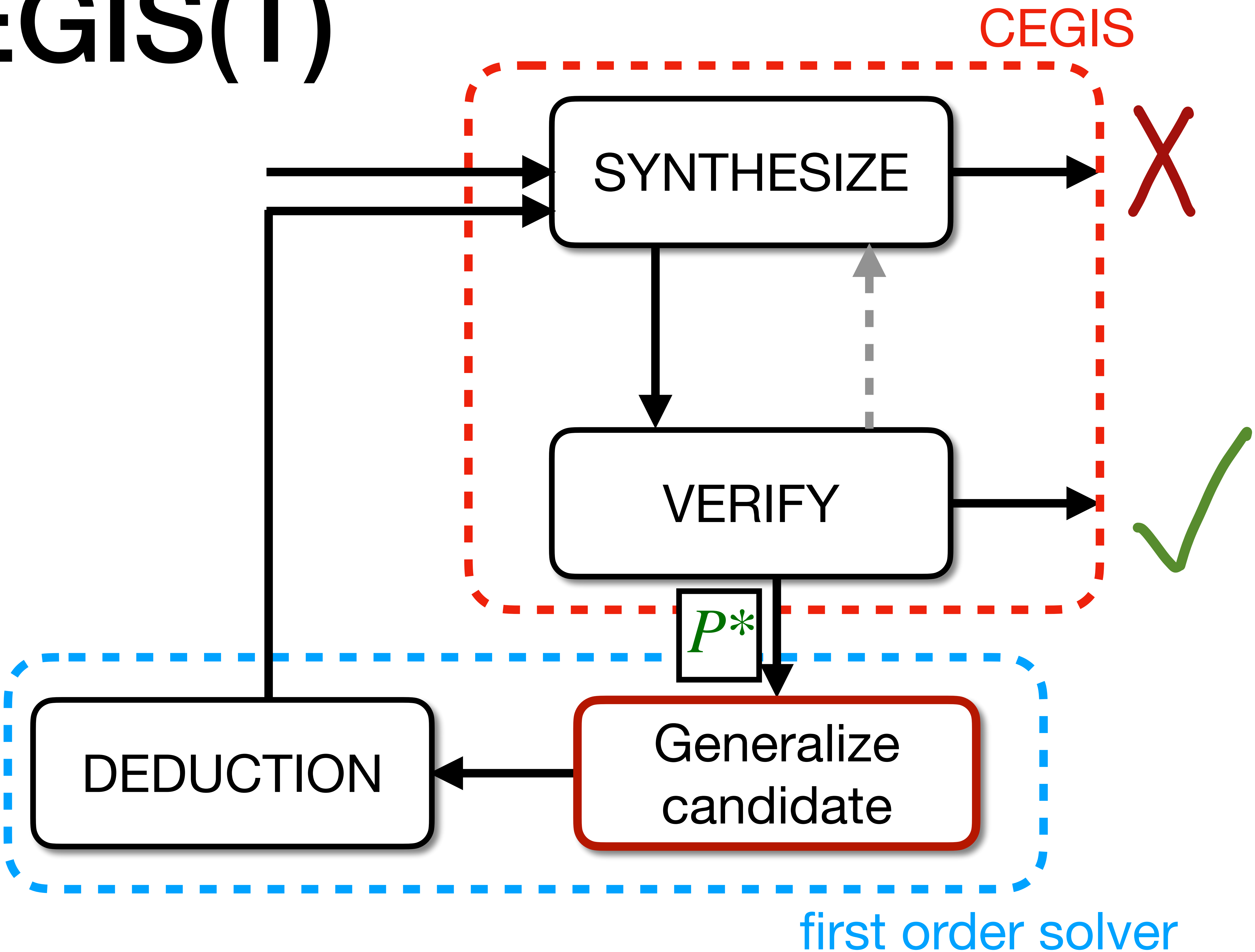

CEGIS
SYNTHESIZE
VERIFY
DEDUCTION
Generalize candidate
first order solver

CEGIS(T)



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

CEGIS(T)


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

Generalize

Candidate

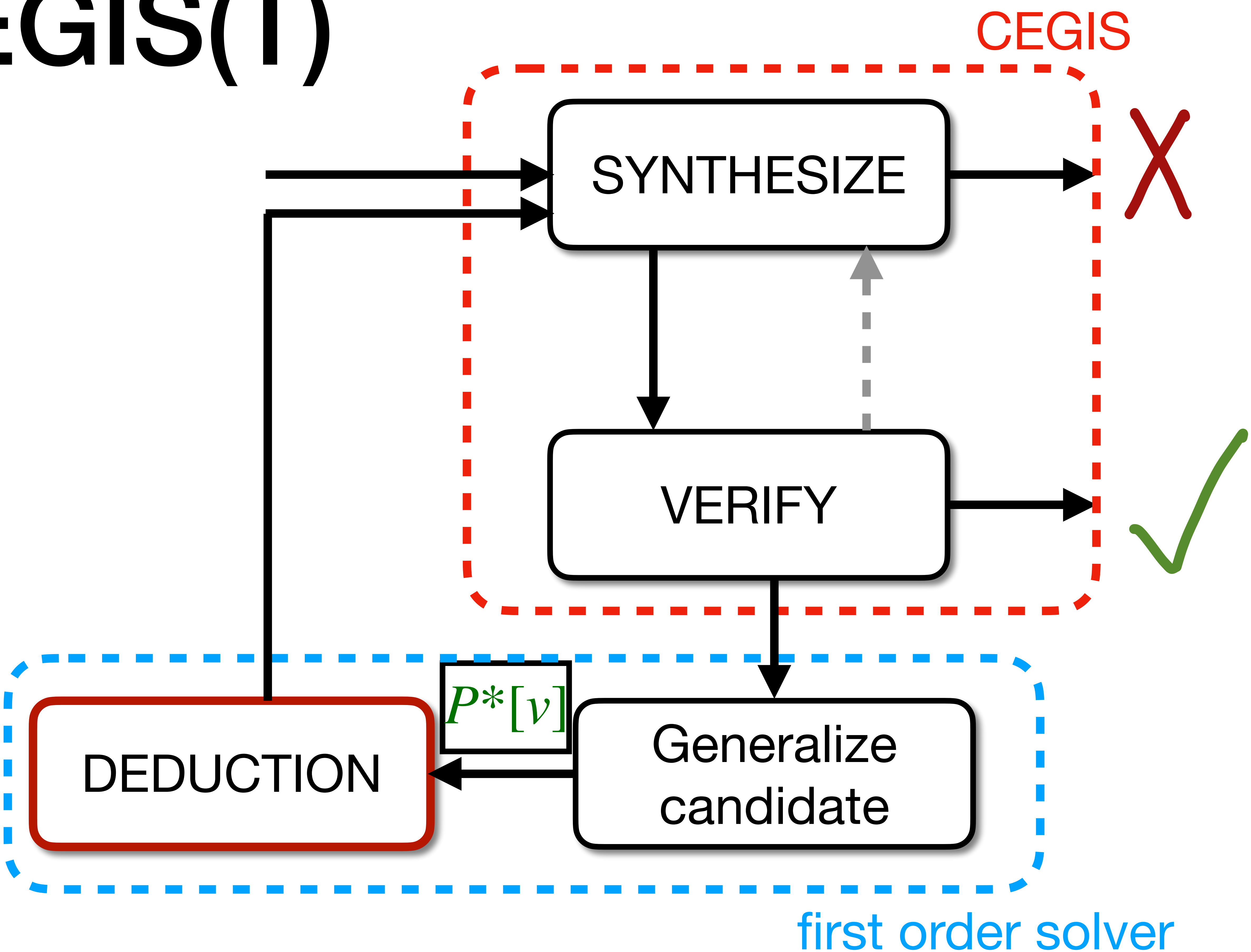
P^* $(x < \boxed{95})$

$\big\downarrow$

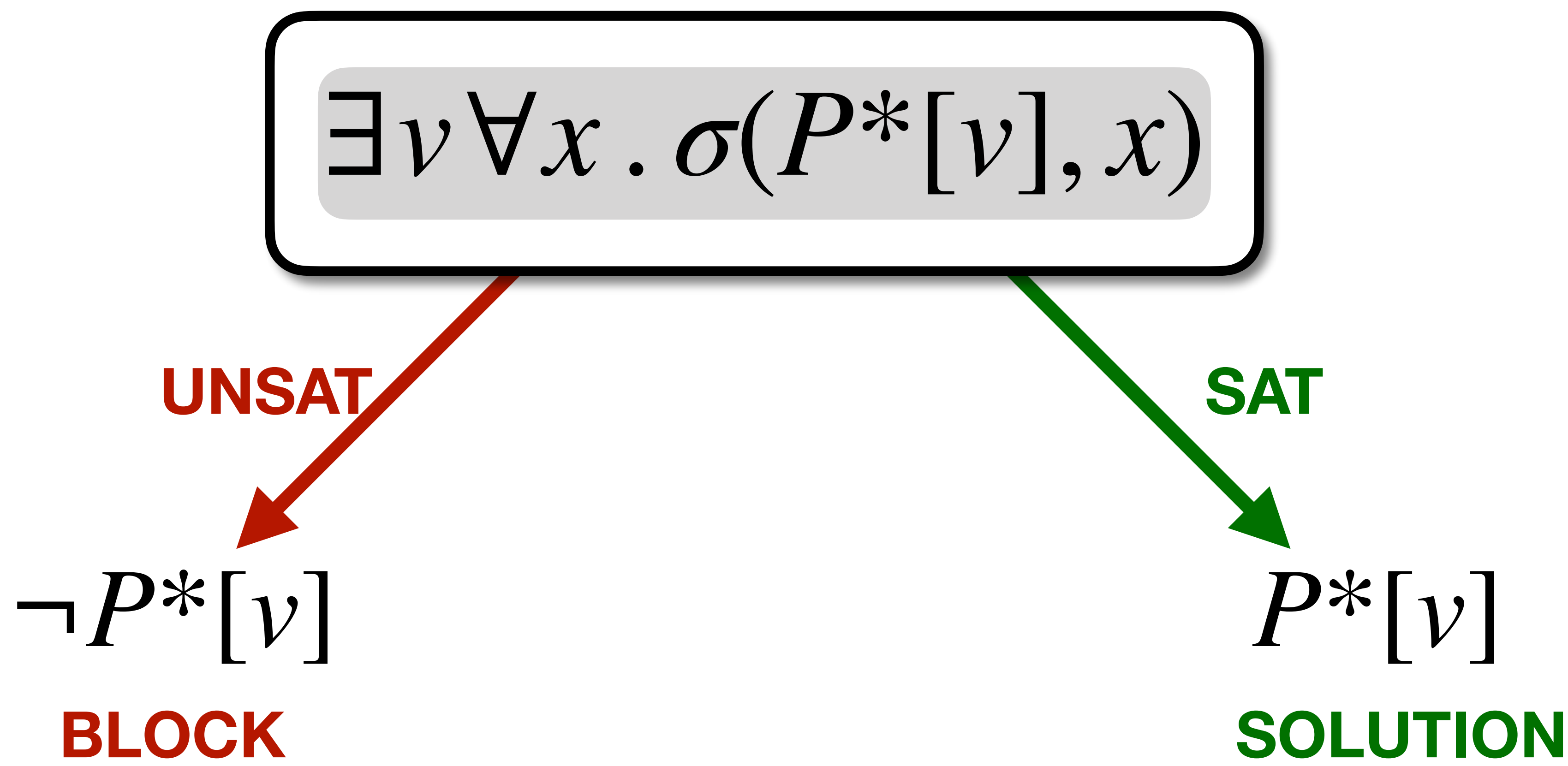
$P^*[v]$ $(x < \boxed{v})$

Generalized candidate

CEGIS(T)

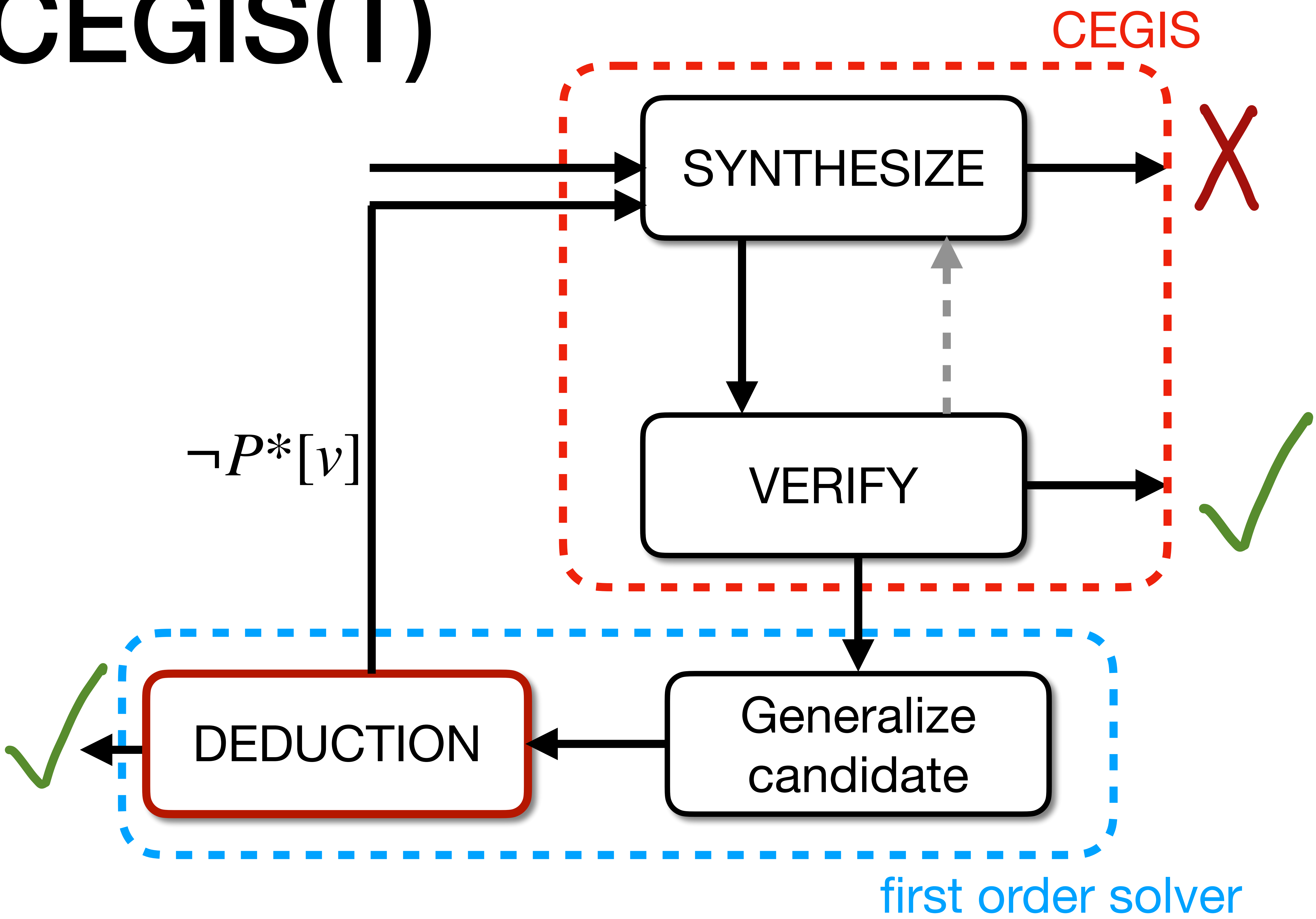

CEGIS
SYNTHESIZE
VERIFY
DEDUCTION
$P^*[v]$
Generalize candidate
first order solver

Deduction



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

First order solver

Solves 1st order formula with:
- Arbitrary propositional structure
- 1 quantifier alternation

Paper presents 2 versions:
- SMT
- FM

CEGIS(T) - SMT

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

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

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

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

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

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

$\neg P^*[v]$
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:

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

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

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:

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

SAT

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

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

$\neg P^*[v]$
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

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

$\neg P^*[v]$
BLOCK

$v > 95$
CONSTRAINT

$v < 95$
CONSTRAINT

$P^*[v]$
SOLUTION

Outline

- Overview of CEGIS and motivation for CEGIS(T)
- CEGIS(T): algorithm in detail
- **Evaluation**
- CEGIS(T) in CVC4
- Ongoing work: beyond constants

Experiments

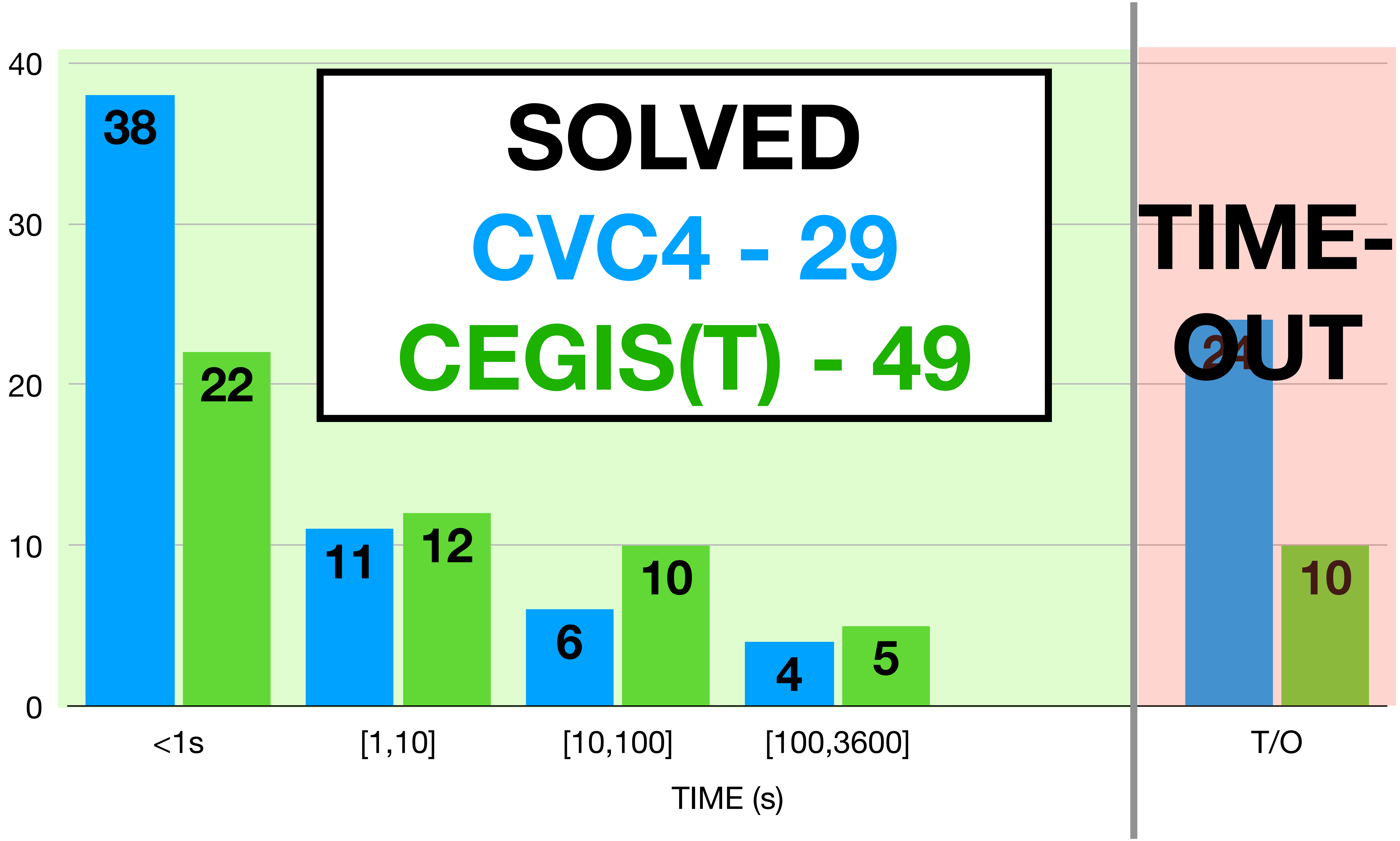
Benchmarks:

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

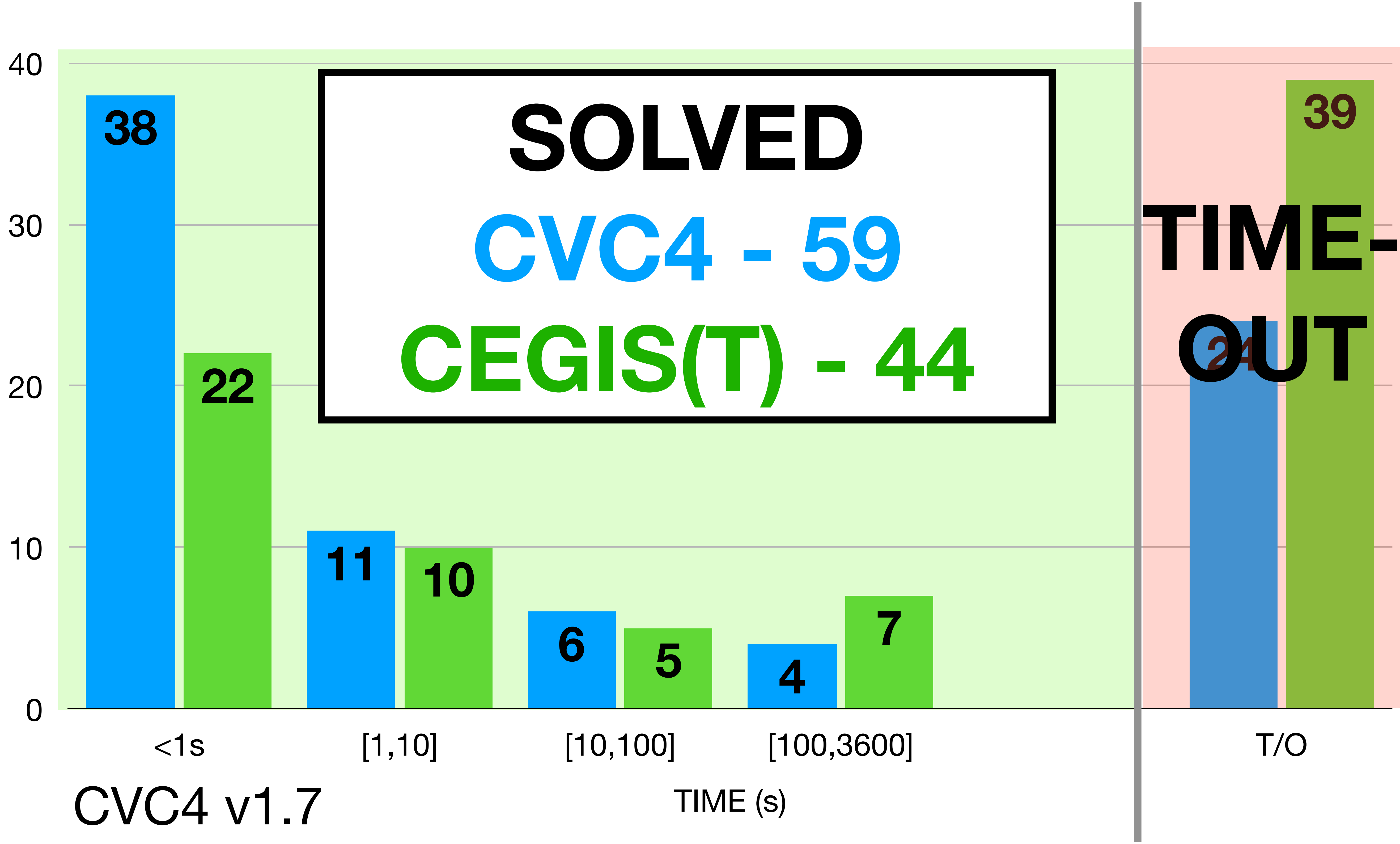
Solvers:

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

Experiments



Experiments - update



CVC4 v1.7

CEGIS(T) updated

more benchmarks

Outline

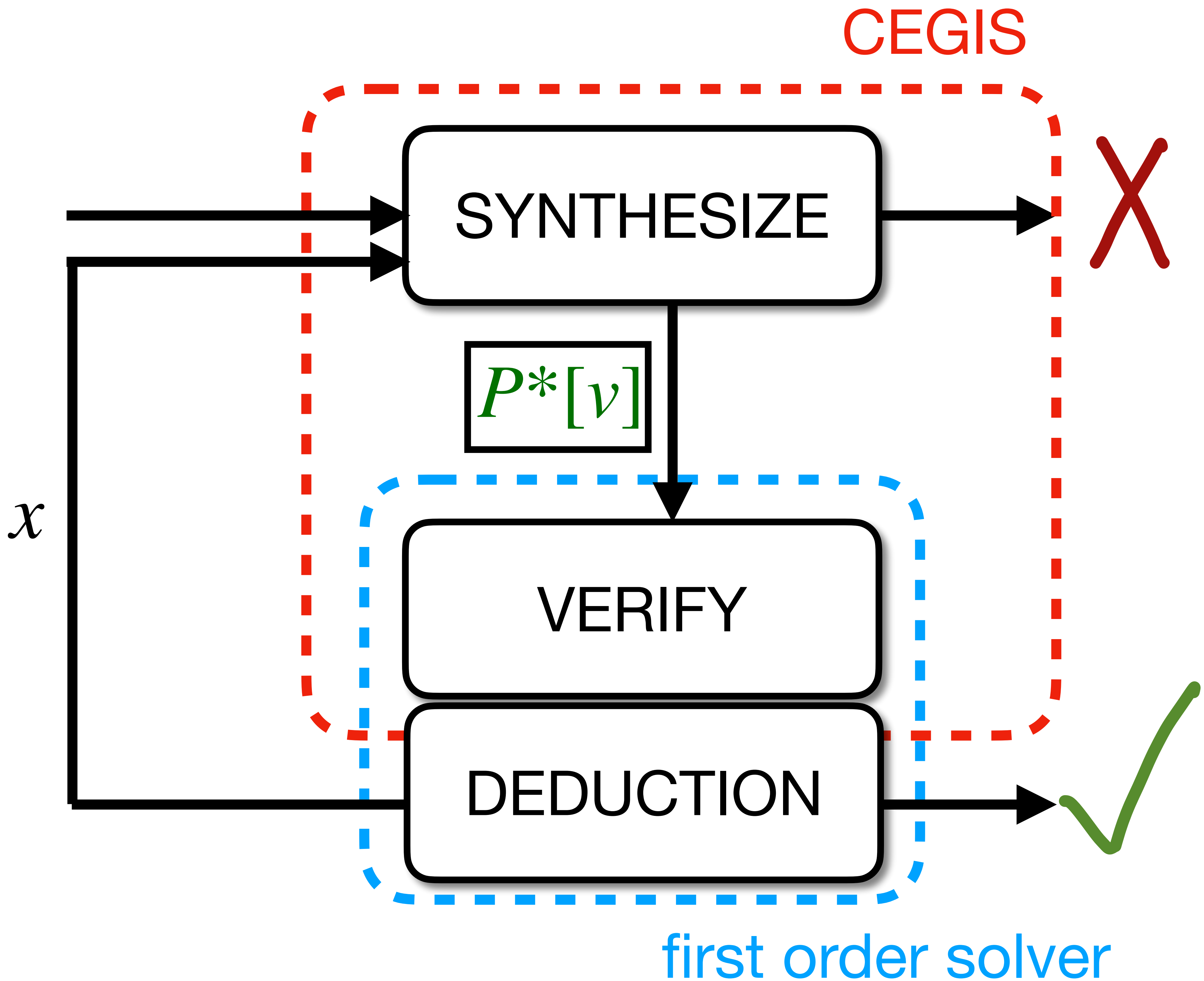
- Overview of CEGIS and motivation for CEGIS(T)
- CEGIS(T): algorithm in detail
- Evaluation
- **CEGIS(T) in CVC4**
- Ongoing work: beyond constants

CEGIS(T) in CVC4

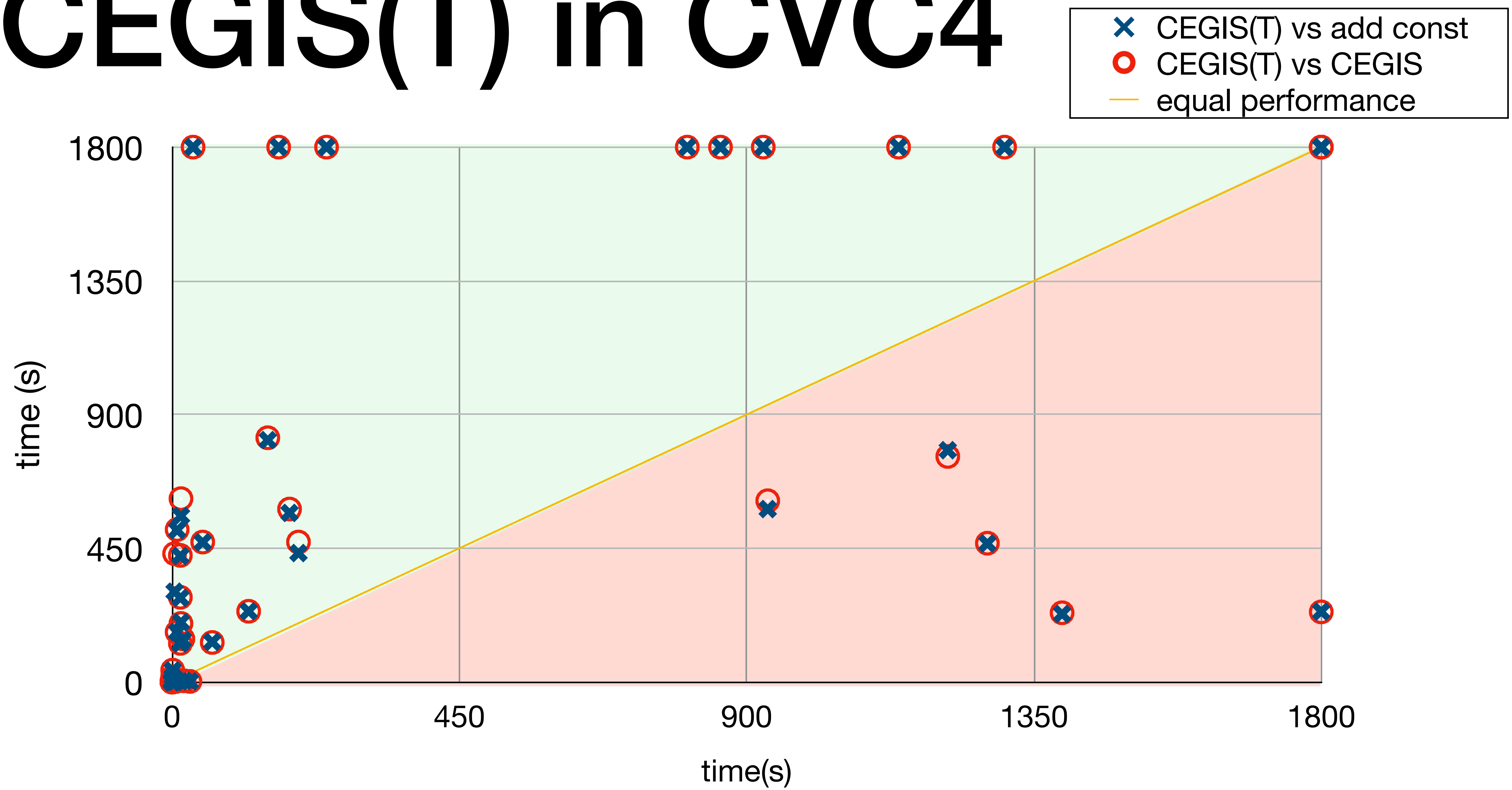
CVC4 implementation of CEGIS(T)

- CVC4 version 1.7
- Makes self-call to CVC4 SMT solver
- Supports CEGIS(T) with a syntactic template

CEGIS(T) in CVC4



CEGIS(T) in CVC4



- BV-invertibility benchmarks
- CEGIS(T) solves 8 unique benchmarks
- avg. ~20s faster on mutually solved benchmarks
- VBS solver gains avg. 48s on mutually solved benchmarks

CEGIS(T)

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

CEGIS(T)

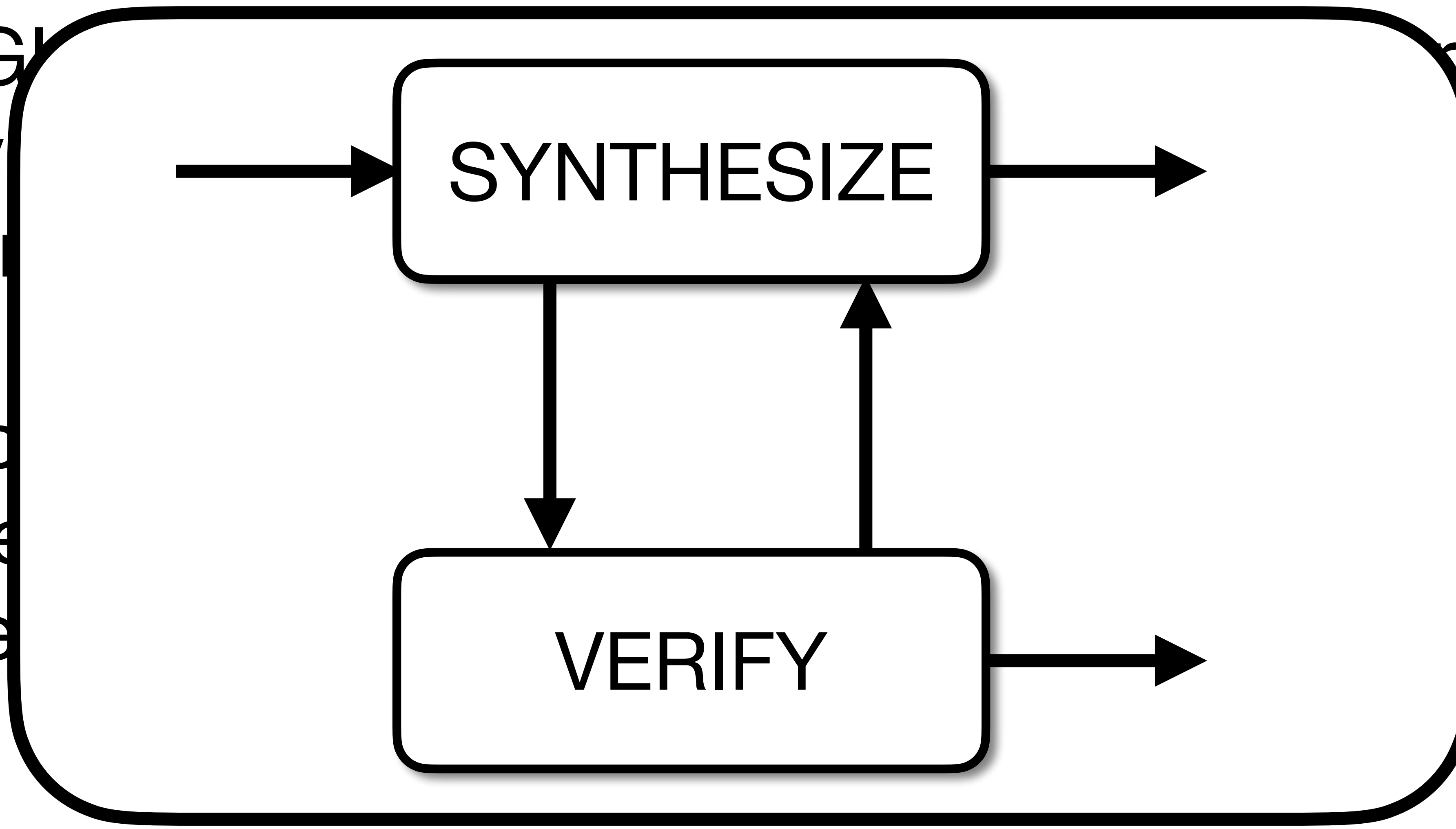
CEG rder

solv

- E

Algo

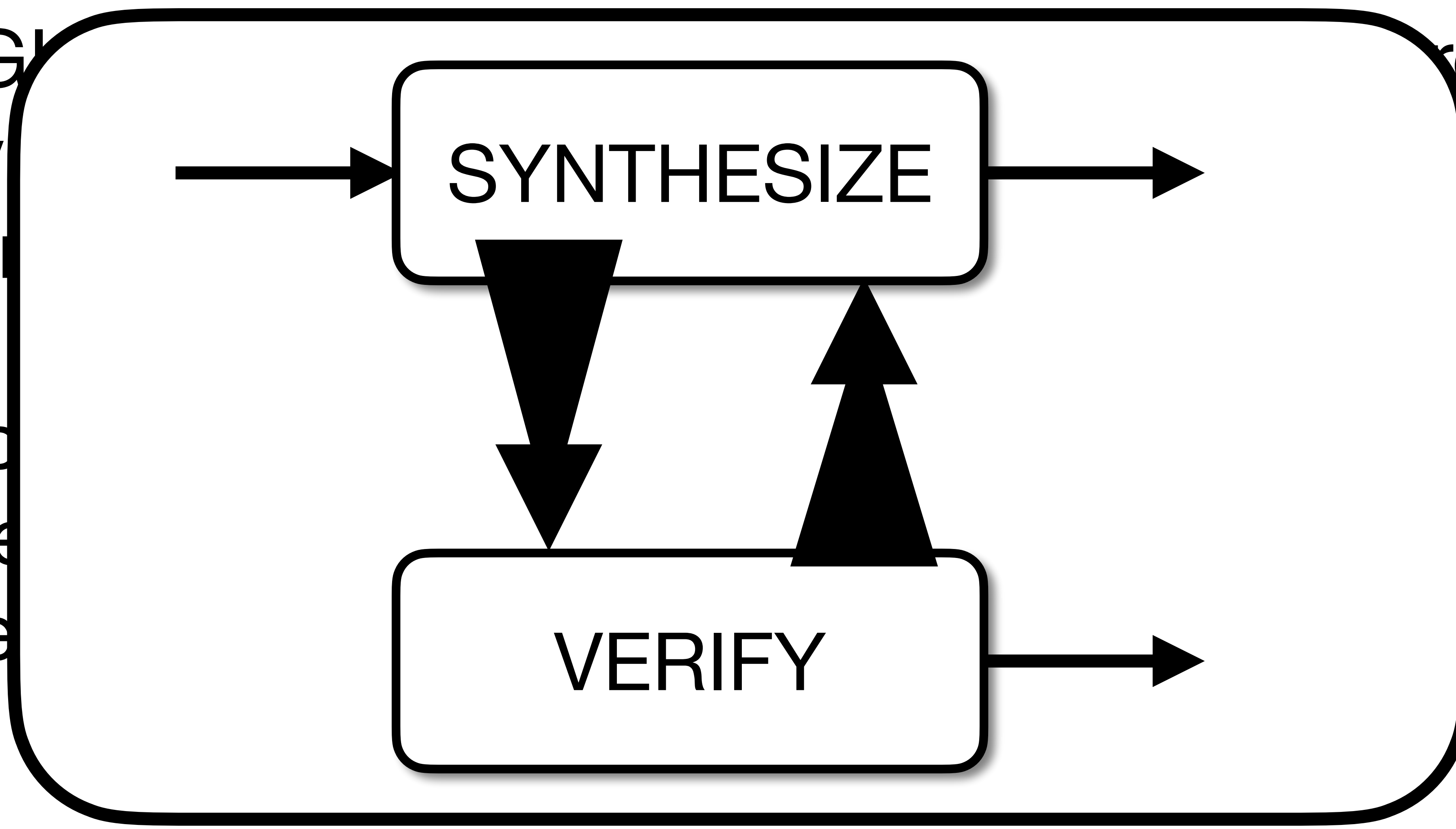
- ve
- re



CEGIS(T)

CEG rder
solv
- E

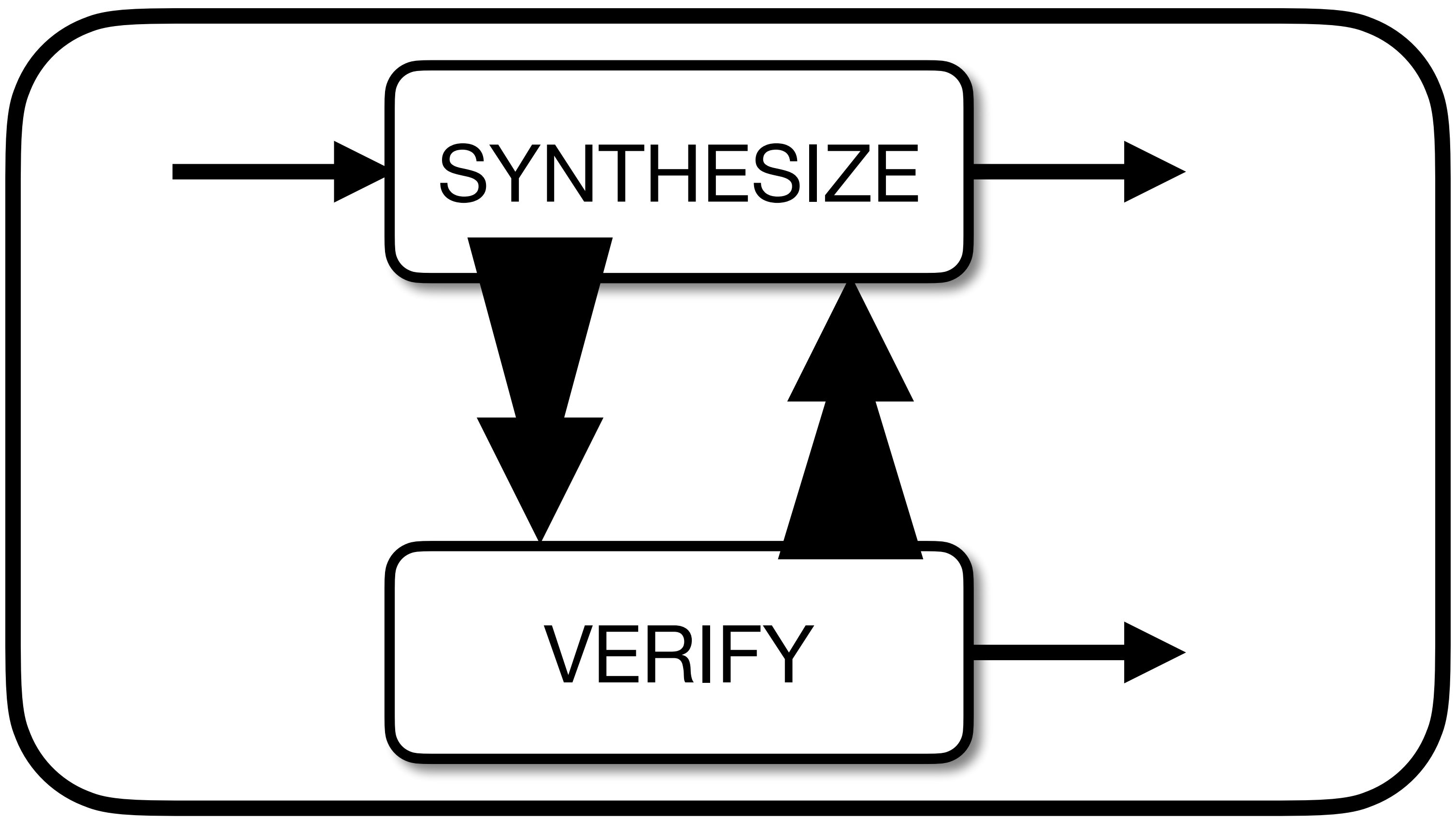
Algo
- ve
- re



Outline

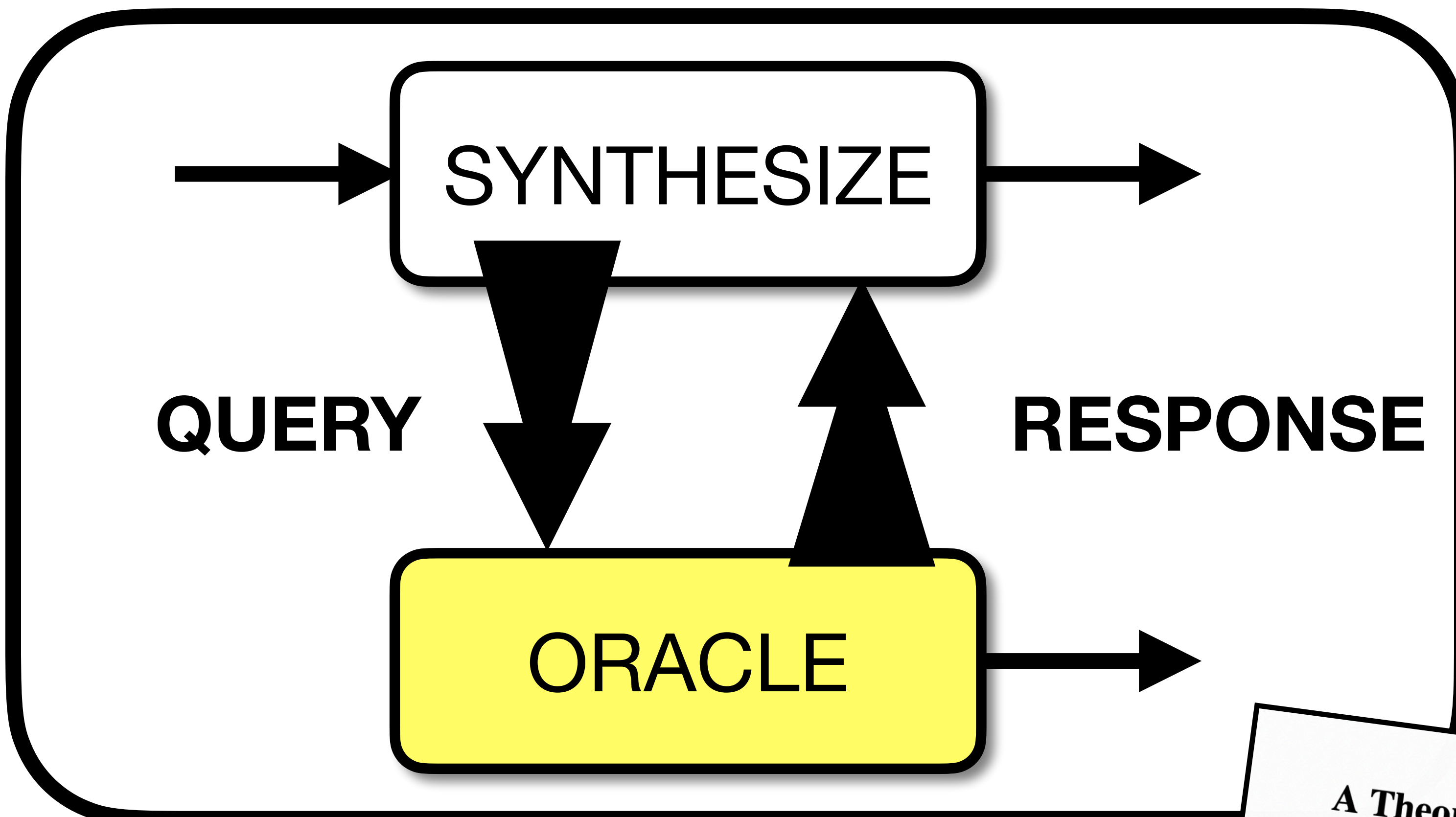
- Overview of CEGIS and motivation for CEGIS(T)
- CEGIS(T): algorithm in detail
- Evaluation
- CEGIS(T) in CVC4
- **Ongoing work: beyond constants**

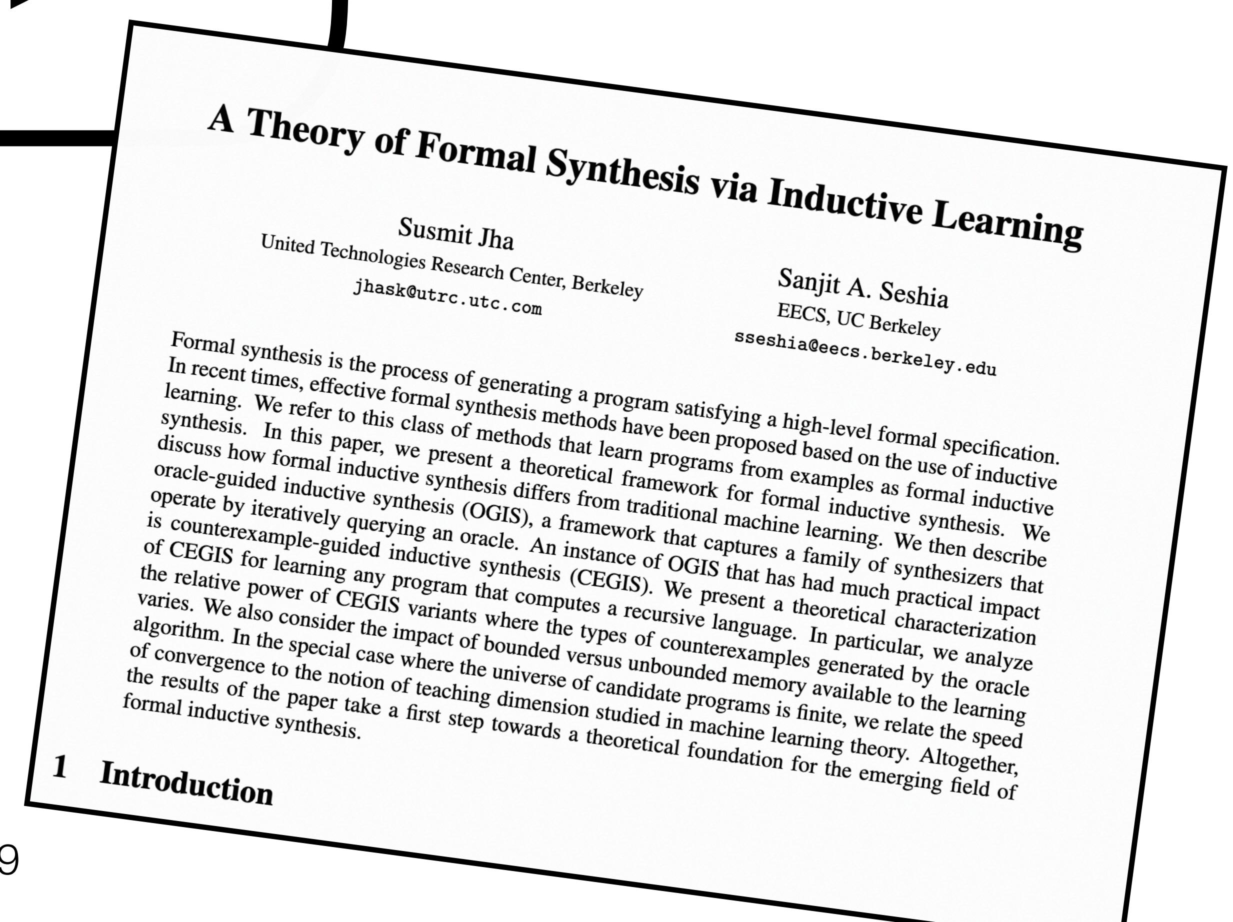
Beyond constants?



Beyond constants?

Via Oracle Guided Synthesis





A Theory of Formal Synthesis via Inductive Learning

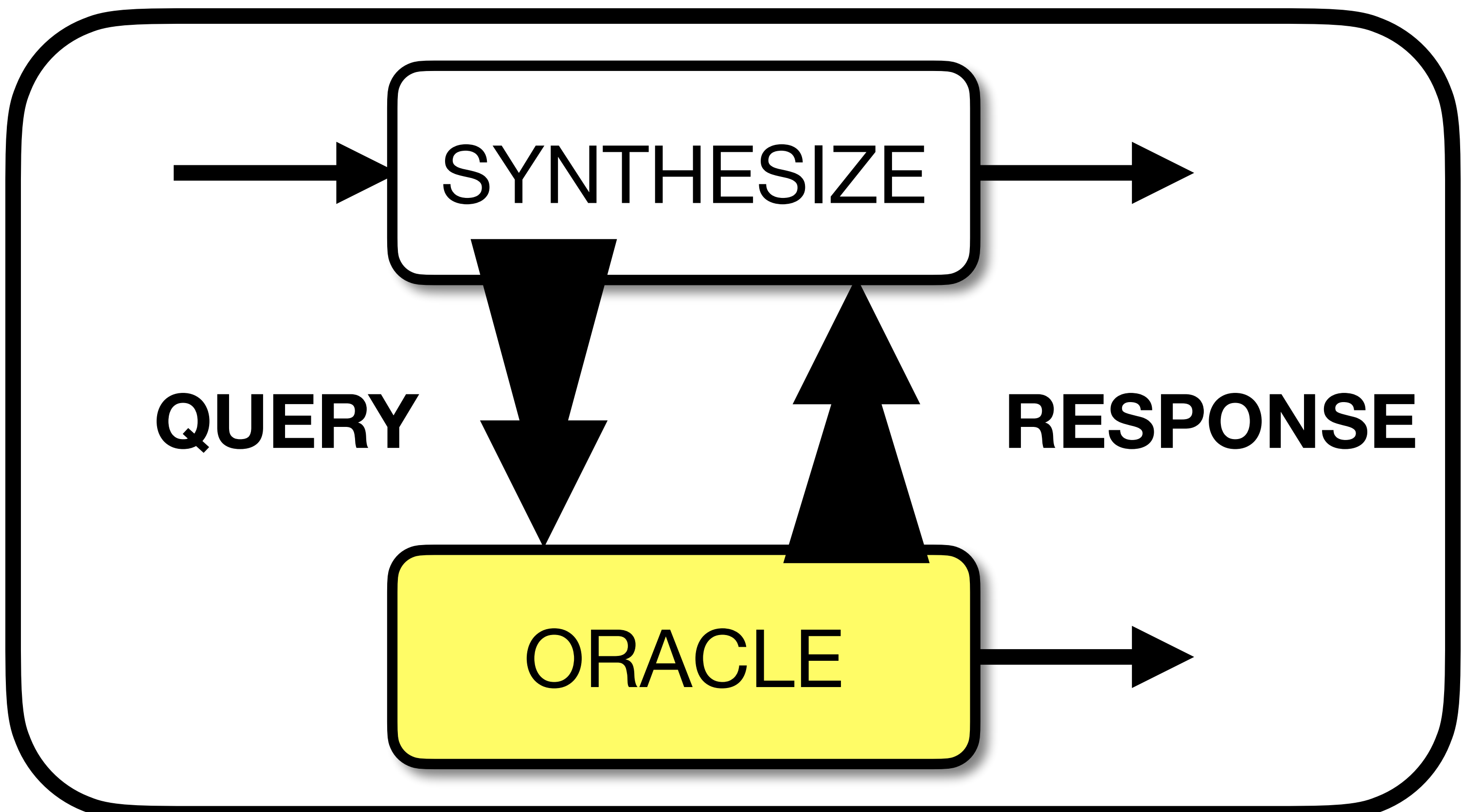
Susmit Jha
United Technologies Research Center, Berkeley
jhask@utrc.utc.com

Sanjit A. Seshia
EECS, UC Berkeley
sseshia@eecs.berkeley.edu

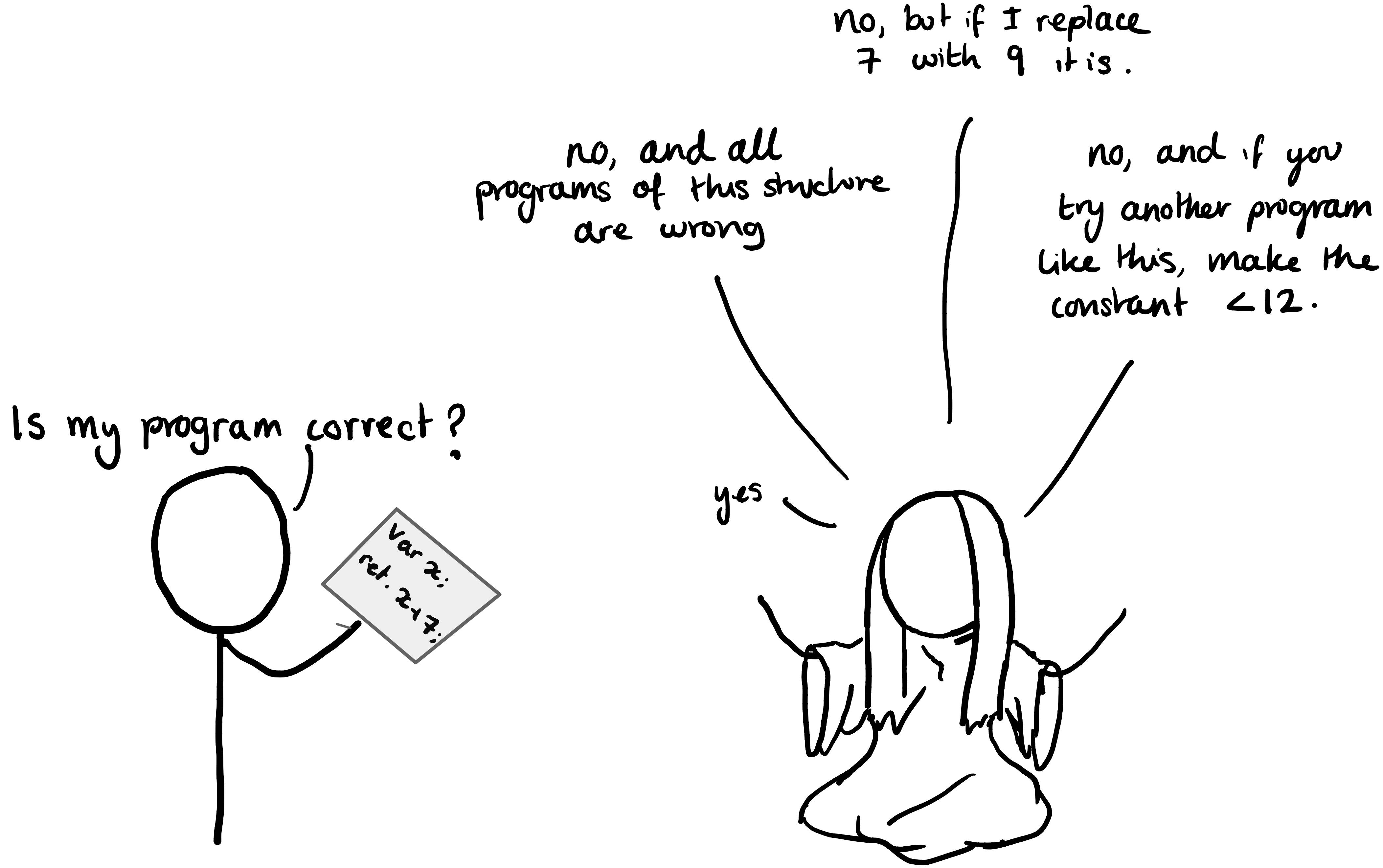
Formal synthesis is the process of generating a program satisfying a high-level formal specification. In recent times, effective formal synthesis methods have been proposed based on the use of inductive learning. We refer to this class of methods that learn programs from examples as formal inductive synthesis. In this paper, we present a theoretical framework for formal inductive synthesis. We discuss how formal inductive synthesis differs from traditional machine learning. We then describe oracle-guided inductive synthesis (OGIS), a framework that captures a family of synthesizers that operate by iteratively querying an oracle. An instance of OGIS that has had much practical impact is counterexample-guided inductive synthesis (CEGIS). We present a theoretical characterization of CEGIS for learning any program that computes a recursive language. In particular, we analyze the relative power of CEGIS variants where the types of counterexamples generated by the oracle varies. We also consider the impact of bounded versus unbounded memory available to the learning algorithm. In the special case where the universe of candidate programs is finite, we relate the speed of convergence to the notion of teaching dimension studied in machine learning theory. Altogether, the results of the paper take a first step towards a theoretical foundation for the emerging field of formal inductive synthesis.

1 Introduction

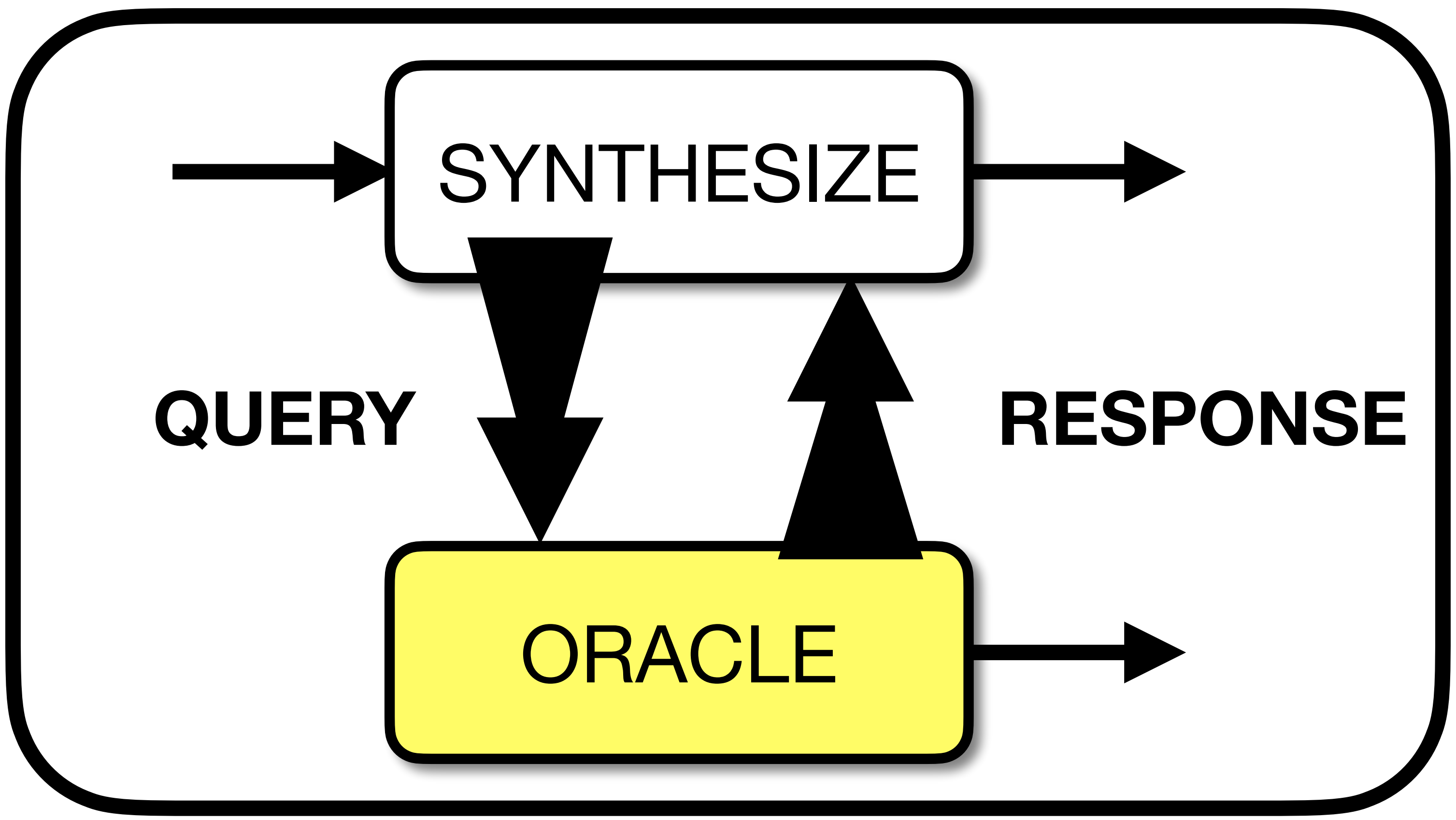
Beyond constants:
general queries and responses



CEGIS(T) as oracle guided synthesis



Beyond constants:
general queries and responses



- Future direction for SyGuS
- Syntax extension in SyGuS-IF
- What type of queries/responses? Any! (Provided the response can be expressed a logical constraint)

Conclusions

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

Algorithmic insights:

- verify generalized candidate solutions
- return generalized counterexamples

Broader communication is good!

Conclusions

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

Algorithmic insights:

- verify generalized candidate solutions
- return generalized counterexamples

Broader communication is good!