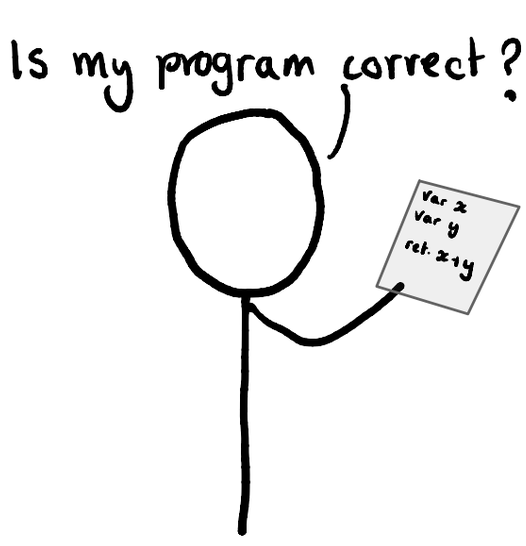
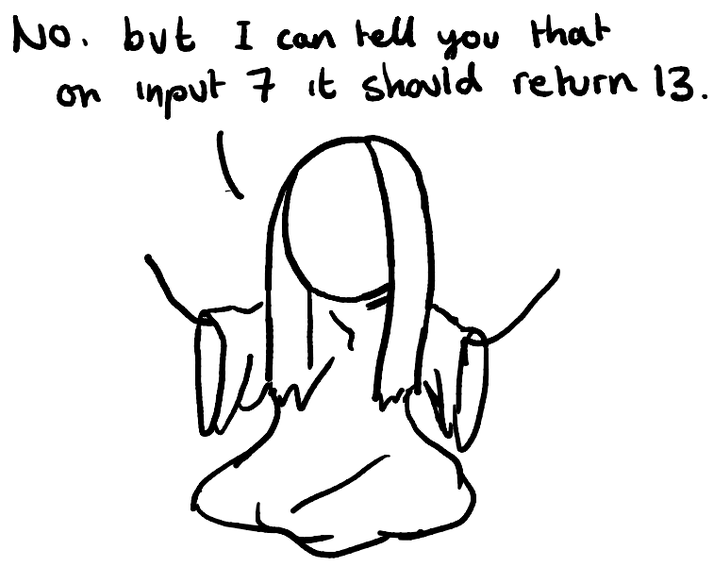
Satisfiability and Synthesis Modulo Oracles

Elizabeth Polgreen, Andrew Reynolds, Sanjit A. Seshia

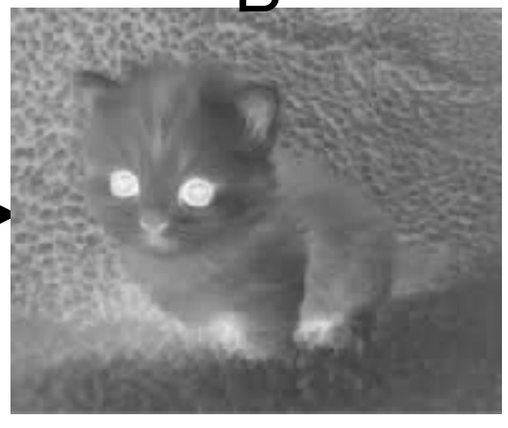




Find a function that transforms Cat A into Cat B?

A

B



Can I use program synthesis?

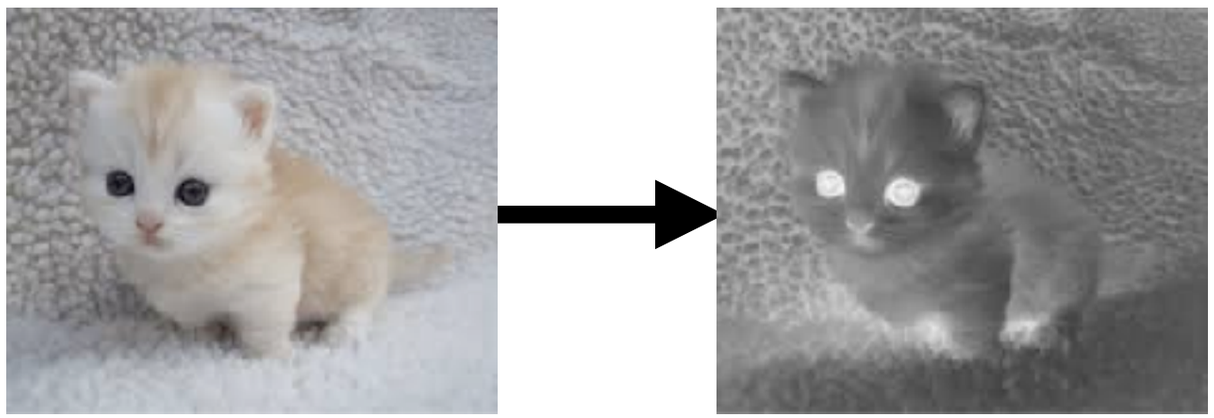
$$\exists f . f(CatA) = CatB$$

!! PROBLEM !!

f is complicated! Needs to:

- **load the JPG image**
- **apply transformation to all the pixels**
- **output a new image**

Find a function that transforms Cat A into Cat B?



Can I use program synthesis?

f(pixel value, location)

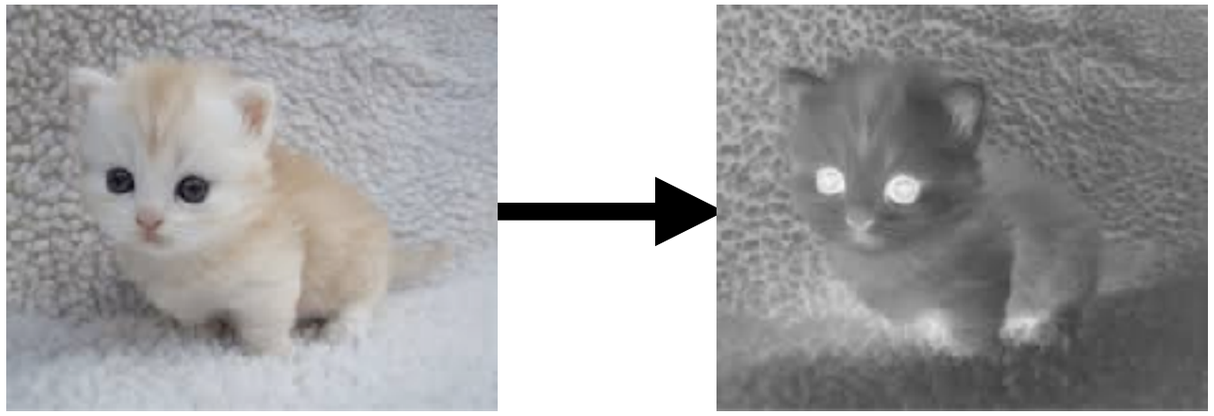
- Load image
- Compile f
- Apply f to all pixels
- Output image

$$\exists f\ \forall pixels, locs\ .\ f(CatA(loc), loc) = CatB(loc)$$

!! PROBLEM !!

Synthesiser needs to interpret CatA(loc)

Find a function that transforms Cat A into Cat B?



Can I use program synthesis?

Give all pixels input-output examples prior to solving?

$$\exists f . f(122,0) = 141 \wedge \ldots \wedge f(133,65536) = 144$$

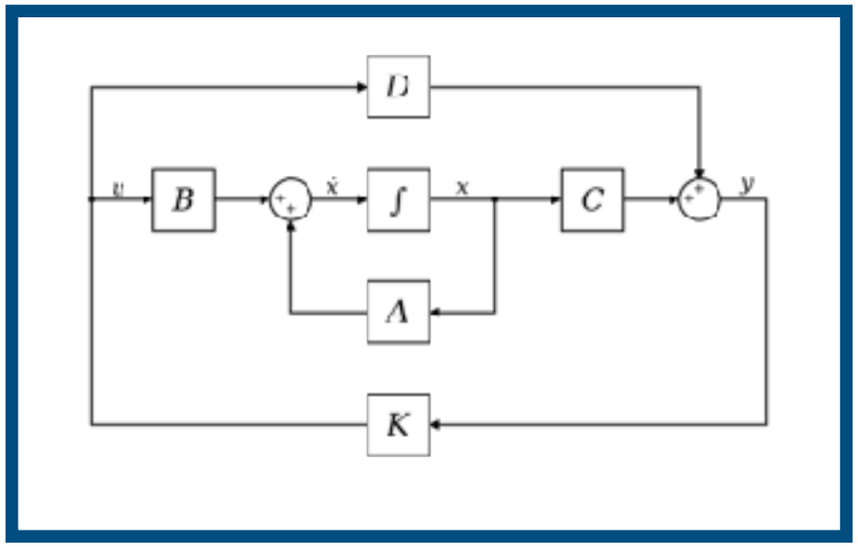
!! PROBLEM !!

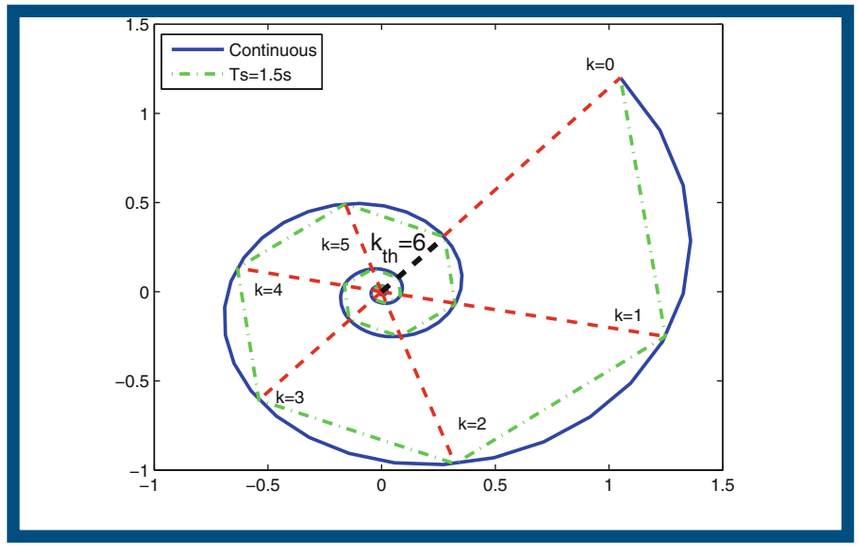
We tried this!

- **65536 input-output examples**
- **Assumes that compiler/image processing library is all bug-free.**

Do I need to build a new solver?

Find a digital controller K for this LTI system?





Can I use program synthesis?

$$\exists k \,.\, Stable(A - Bk) \wedge \forall x \,.\, Safe(x)$$

!! PROBLEM !!

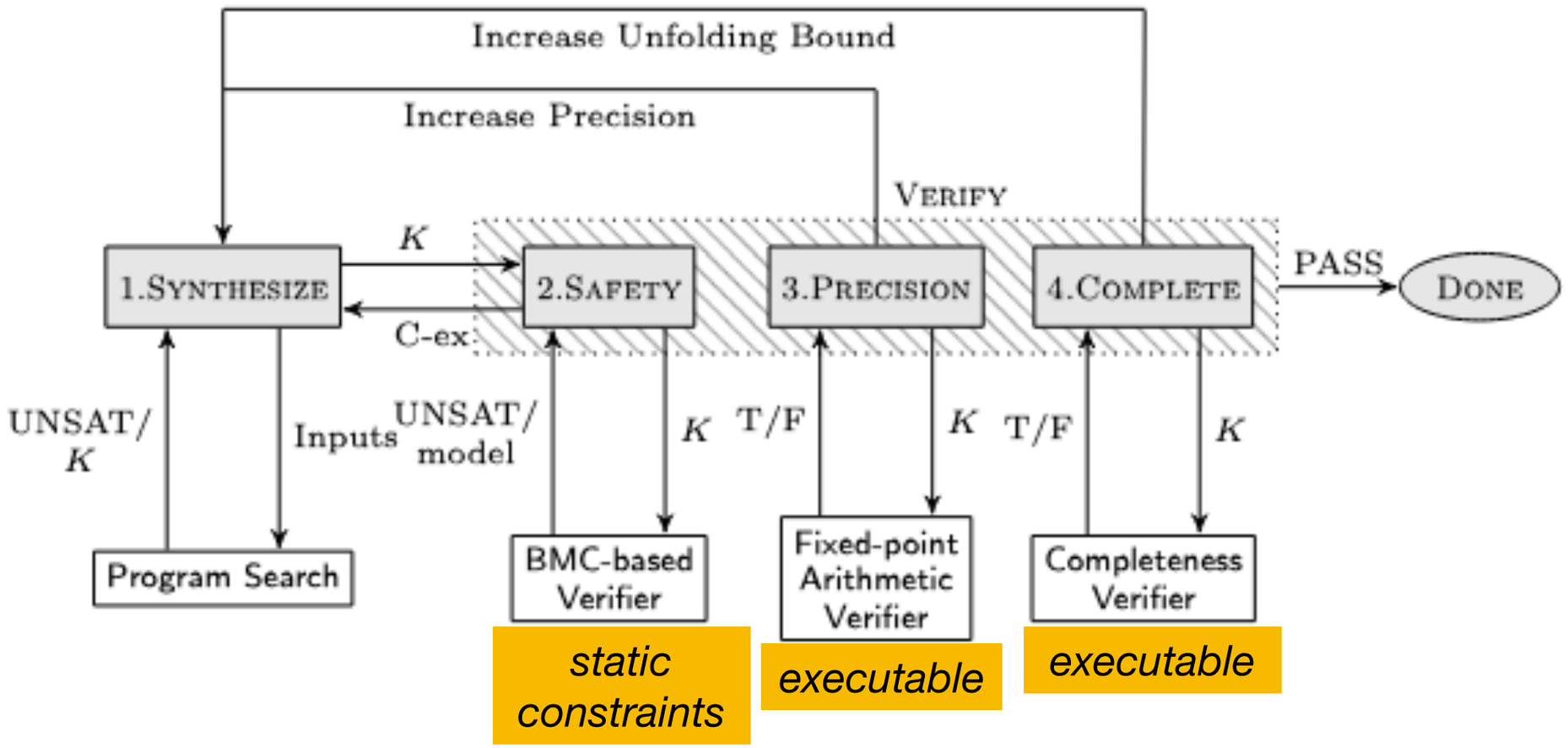
We tried this!

- **Verifier needs to find eigenvalues**
- **Extremely non-linear**

Do I need to build a new solver?

Automated formal synthesis of provably safe digital controllers for continuous plants

Alessandro Abate[2] · Iury Bessa[3] · Lucas Cordeiro[4] · Cristina David[5] · Pascal Kesseli[6] · Daniel Kroening[2] · Elizabeth Polgreen[1]



Find prime factors of a number?

Can I use SMT?

$$\exists f_1, f_2 \,.\, isPrime(f_1) \wedge isPrime(f_2) \wedge f_1 * f_2 = x$$

!! PROBLEM

isPrime is a recursive function!

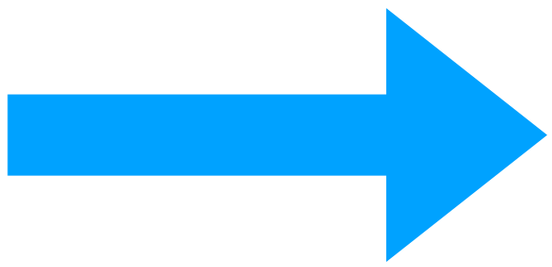
Do I need to build a new solver?

Why don’t off-the-shelf solvers work?

- Parts are hard to model with static constraints e.g., the image processing library.

- Parts are hard to reason about e.g., eigenvalues, primes.

- Too many constraints, which ones are important? e.g., which pixels matter?



Solution: Use executable “oracles”

Coming up

- Existing use of oracles

- Formal definition of oracle interfaces

- SMT with oracles (**SMTO**):
  - when is it satisfiable/unsatisfiable
  - algorithm

- Synthesis with oracles (**SyMO**):
  - when are solutions correct
  - unifying algorithm for solving

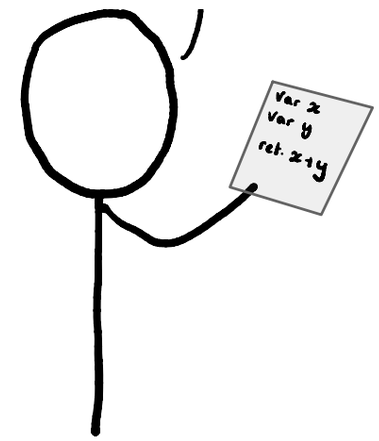
- More cat pictures

- Prototype evaluation

What is an oracle?

Query

Is this number prime?



Response

No, it is not prime.

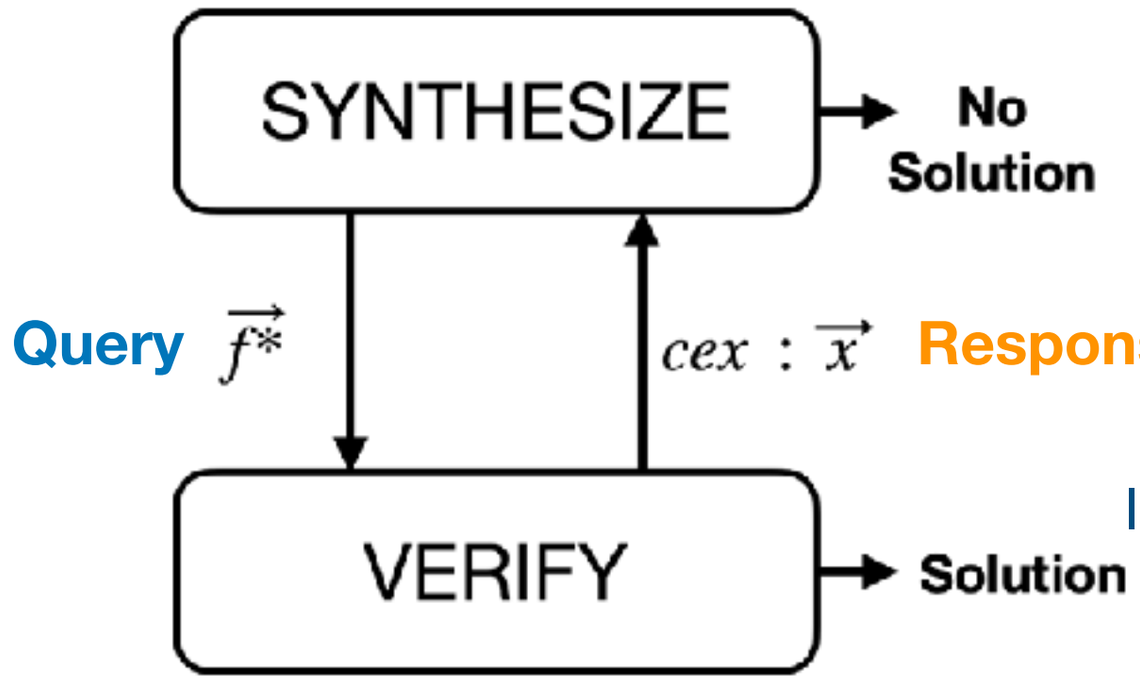
Existing use of oracles:

- Existing synthesis algorithms are using oracles!
- Jha and Seshia[1] set out the theory behind this.
- Our contribution: defining satisfiability/synthesis modulo oracles problem and proposing a unifying algorithm

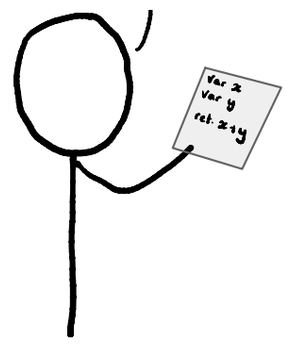
[1] A theory of formal synthesis via inductive learning - Jha and Seshia

Existing use of oracles:

- Counterexample Guided Inductive Synthesis [2]



Is my program “x + 5” correct?



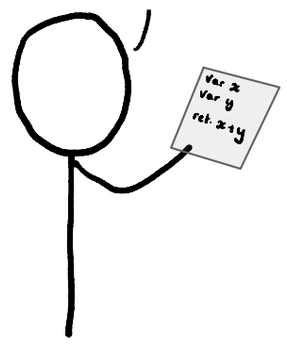
No, it’s wrong on input 7

[2] Combinatorial sketching for finite programs - Solar Lezama et al

Existing use of oracles:

- CEGIS(T) [3]

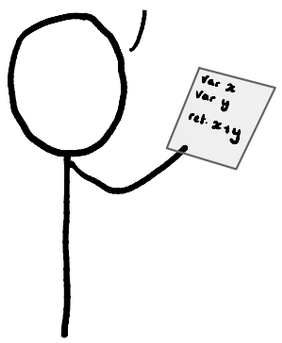
Is my program “x + 5” correct?



No, but if you replace 5 with 1, it would be

- ICE learning [4]

Is my invariant “x > 5 ” correct?



No, and $inv(6) \implies inv(5)$

No, and $inv(100) = false$

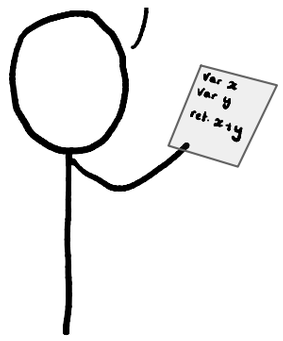
No, and $inv(0) = true$

[3] Counterexample Guided Inductive Synthesis modulo Theories - Abate et al
[4] ICE: A robust framework for learning invariants - Garg et al

Existing use of oracles:

- Program lifting [5]

Is my program correct?



No, it's wrong on this input

it should access these memory locations

the runtime complexity should be linear

[5] Program Lifting using Gray-Box Behavior - Collie et al

Examples

People use oracles, but they build their own custom solver. Why?

- Solver needs custom information about the oracles (what does the response from the oracle mean?)

What if we could communicate this information to an off-the-shelf SMT or SyGuS solver?

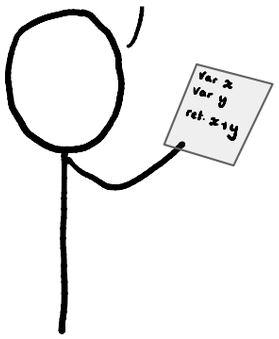
Coming up

- Existing use of oracles
- Formal definition of oracle interfaces
- SMT with oracles (**SMTO**):
  - when is it satisfiable/unsatisfiable
  - algorithm
- Synthesis with oracles (**SyMO**):
  - when are solutions correct
  - unifying algorithm for solving
- More cat pictures
- Prototype evaluation

What is an oracle?

- We define how the oracle is queried by defining an interface

Query



Response

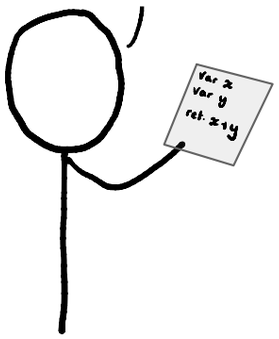
$\overrightarrow{y}$: query domain

$\vec{z}$: response co-domain

What is an oracle?

- We define how the oracle is queried by defining an interface

Query



Response

$\overrightarrow{y}$: query domain

$\vec{z}$: response co-domain

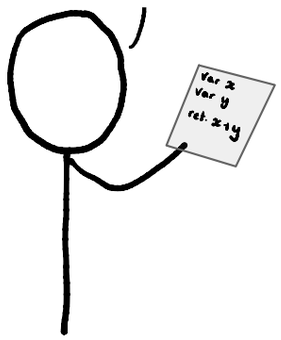
α_{gen} : assumption generator

β_{gen} : constraint generator

- and assumption and constraint generators, which generate:
  - assumptions the solver is allowed to make
  - and constraints the solver must abide by

Example oracle: $\mathcal{O}_{prime}$

Is this number y prime?



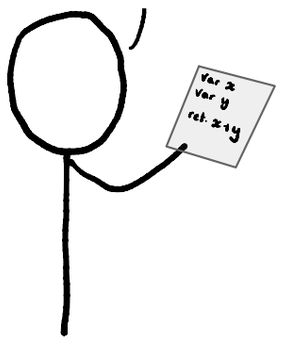
No, z=false, it is not prime.

$$\begin{aligned}
\overrightarrow{y} &: (y : integer) \\
\vec{z} &: (z : bool) \\
\alpha_{gen} &: prime(y) = z \\
\beta_{gen} &: \emptyset
\end{aligned}$$

Solve: $\exists x \,.\, prime(x) \wedge (x \,\%\, 2 = 1)$

Example oracle: $\mathcal{O}_{prime}$

Is this number y prime?

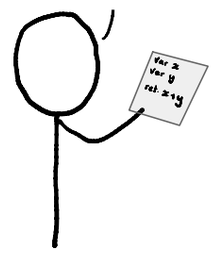


No, z=false, it is not prime.

$$\overrightarrow{y} \quad : (y : integer)$$
$$\vec{z} \quad : (z : bool)$$
$$\alpha_{gen} : prime(y) = z$$
$$\beta_{gen} : \emptyset$$

Solve: $\exists x \,.\, prime(x) \wedge (x \,\%\, 2 = 1)$

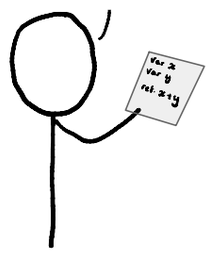
$y = 1?$



$z = false$

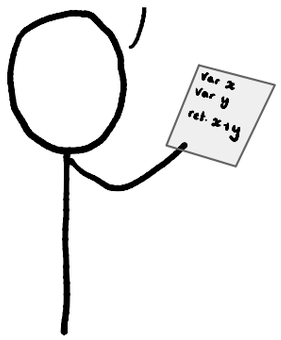
Okay, now I know I can assume

$prime(1) = false$



Example oracle: $\mathcal{O}_{positive-witness}$

Is my program f^* correct?

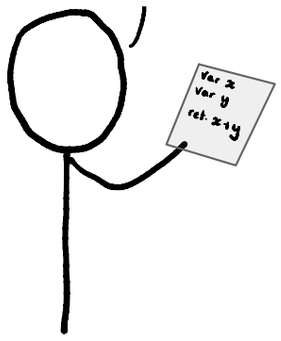


I can tell you that on input 7 it should return 13

$$
\begin{aligned}
\overrightarrow{y} &: (f^* : int \rightarrow int) \\
\vec{z} &: (z_1 : int\,, z_2 : int) \\
\alpha_{gen} &: \varnothing \\
\beta_{gen} &: f(z_1) = z_2
\end{aligned}
$$

Example oracle: $\mathcal{O}_{positive-witness}$

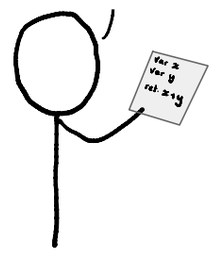
Is my program f^* correct?



I can tell you that on input 7 it should return 13

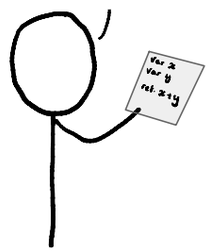
$$\begin{aligned}
\overrightarrow{y} &: (f^* : int \rightarrow int) \\
\vec{z} &: (z_1 : int\,, z_2 : int) \\
\alpha_{gen} &: \emptyset \\
\beta_{gen} &: f(z_2) = z_3
\end{aligned}$$

$f^* = x + 1$



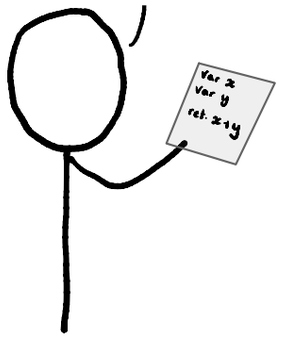
$z_1 = 7,\ z_2 = 13$

Okay, now I know that a valid program f must satisfy $f(7) = 13$



Oracle function symbols

Is this number y prime?



No, z=false, it is not prime.

$$\begin{aligned}
\overrightarrow{y} &: (y : integer) \\
\vec{z} &: (z : bool) \\
\alpha_{gen} &: prime(y) = z \\
\beta_{gen} &: \emptyset
\end{aligned}$$

An oracle function symbol is a symbol whose behaviour is defined to be the same as an external oracle.

Note: oracle must be functional

prime is an oracle function symbol

Coming up

- Existing use of oracles
- Formal definition of oracle interfaces
- SMT with oracles (**SMTO**):
  - when is it satisfiable/unsatisfiable
  - algorithm
- Synthesis with oracles (**SyMO**):
  - when are solutions correct
  - unifying algorithm for solving
- More cat pictures
- Prototype evaluation

Satisfiability Modulo Theories and Oracles (SMTO)

An SMTO problem is a tuple:

$\vec{f}$: a set of ordinary function symbols

$\overrightarrow{\theta}$: a set of oracle function symbols

ρ : a formula in a background theory

$\overrightarrow{\mathcal{O}}$: a set of oracle interfaces

$\vec{f} : \{f_1, f_2\}$

$\overrightarrow{\theta} : \{prime\}$

$\rho : prime(f_1) \wedge prime(f_2) \wedge (f_1 * f_2 = 24)$

$\overrightarrow{\mathcal{O}} : \{\mathcal{O}_{prime}\}$

$\mathcal{O}_{prime}$

$\overrightarrow{y} : (y : integer)$

$\vec{z} : (z : bool)$

$\alpha_{gen} : prime(y) = z$

$\beta_{gen} : \emptyset$

Is this satisfiable? What is a valid assignment to f_1 and f_2?

Satisfiability Modulo Theories and Oracles (SMTO)

SAT?

$$prime(f_1) \wedge prime(f_2) \wedge (f_1 * f_2 = 24)$$

$\mathcal{O}_{prime}$

$\overrightarrow{y}$: $(y : integer)$

$\vec{z}$: $(z : bool)$

α_{gen} : $prime(y) = z$

β_{gen} : $\emptyset$

- If $prime$ does what we expect, then yes! But we don't know that
- If we haven't called the oracle, an assignment must work for ALL possible behaviours of prime
- When we call $\mathcal{O}_{prime}$, we get assumptions about prime
- If the assumptions rule out all possible assignments: UNSAT

Satisfiability Modulo Theories and Oracles (SMTO)

SAT?

$$prime(f_1) \wedge prime(f_2) \wedge (f_1 * f_2 = 24)$$

$\mathcal{O}_{prime}$

$$\begin{aligned}
\overrightarrow{y} &: (y : integer) \\
\vec{z} &: (z : bool) \\
\alpha_{gen} &: prime(y) = z \\
\beta_{gen} &: \varnothing
\end{aligned}$$

Conjunction of assumptions. True if no assumptions

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies \rho$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho$ is unsatisfiable

Unknown otherwise

Satisfiability Modulo Theories and Oracles (SMTO)

SAT?

$$prime(f_1) \wedge prime(f_2) \wedge (f_1 * f_2 = 24)$$

$\mathcal{O}_{prime}$

$\overrightarrow{y}$: $(y : integer)$

$\vec{z}$: $(z_1 : bool, z_2 : integer)$

α_{gen} : $prime(y) = z$

β_{gen} : $f_1 < z_2$

Constraints must be obeyed by the solver:

Conjunction of constraints. True if no constraints.

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies (\rho \wedge B)$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho \wedge B$ is unsatisfiable

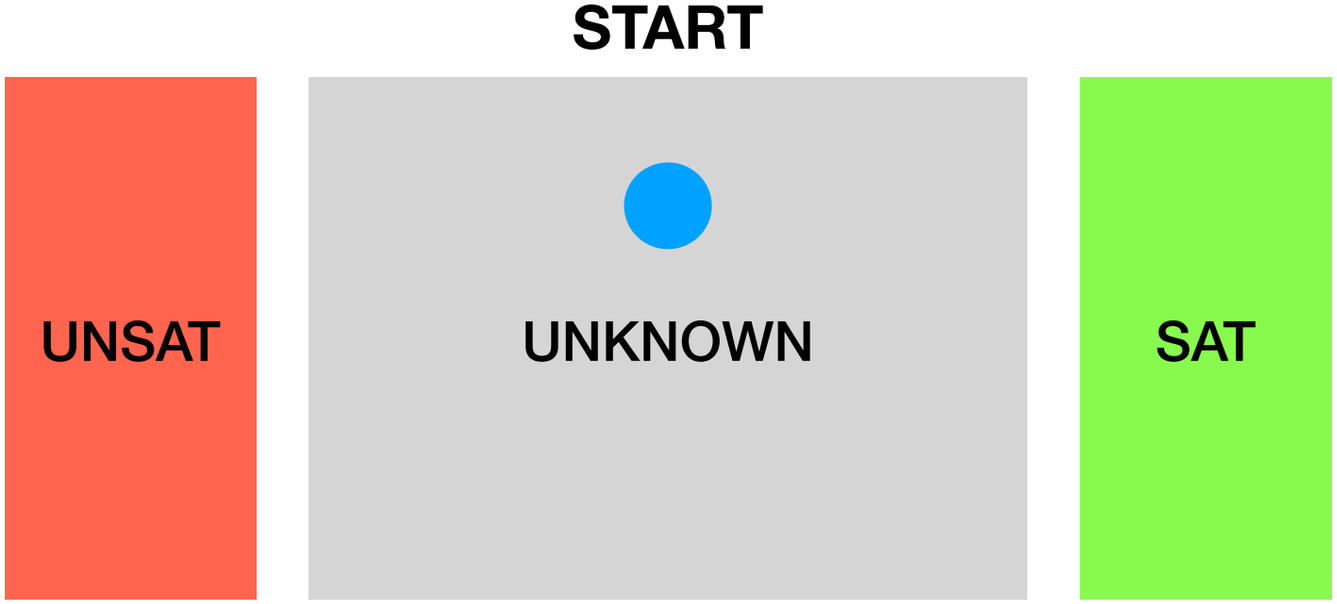
Unknown otherwise

Satisfiability Modulo Theories and Oracles (SMTO)

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies (\rho \wedge B)$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho \wedge B$ is unsatisfiable

Unknown otherwise

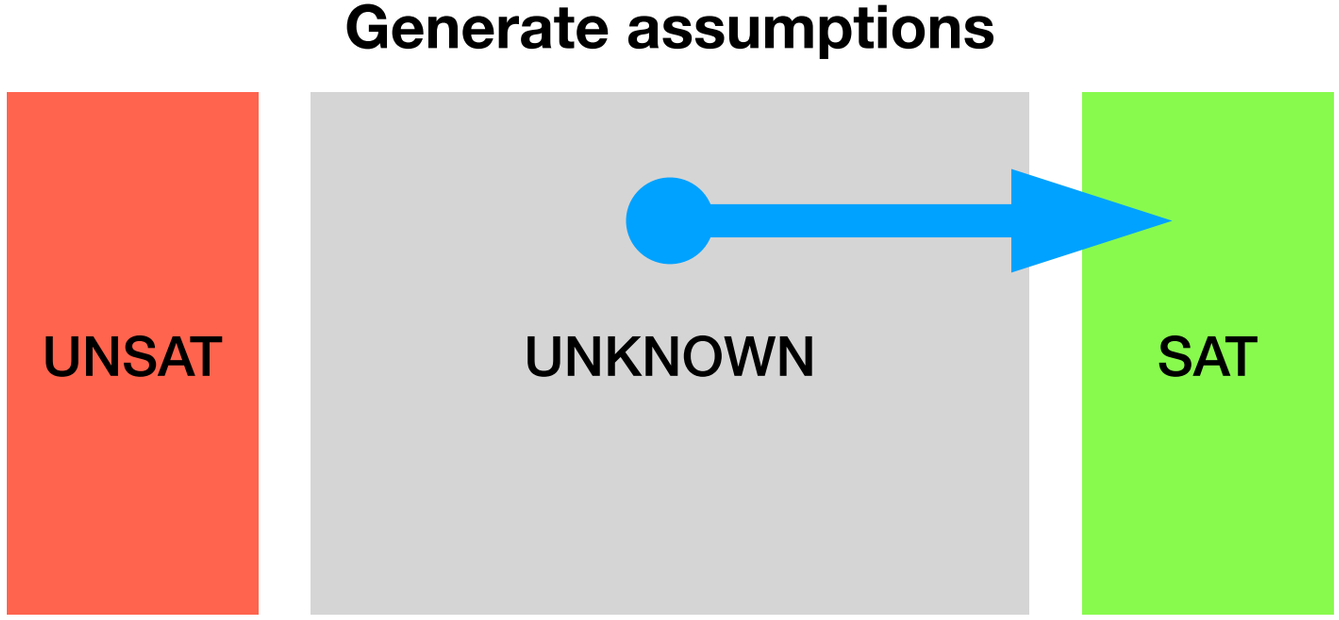


Satisfiability Modulo Theories and Oracles (SMTO)

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies (\rho \wedge B)$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho \wedge B$ is unsatisfiable

Unknown otherwise

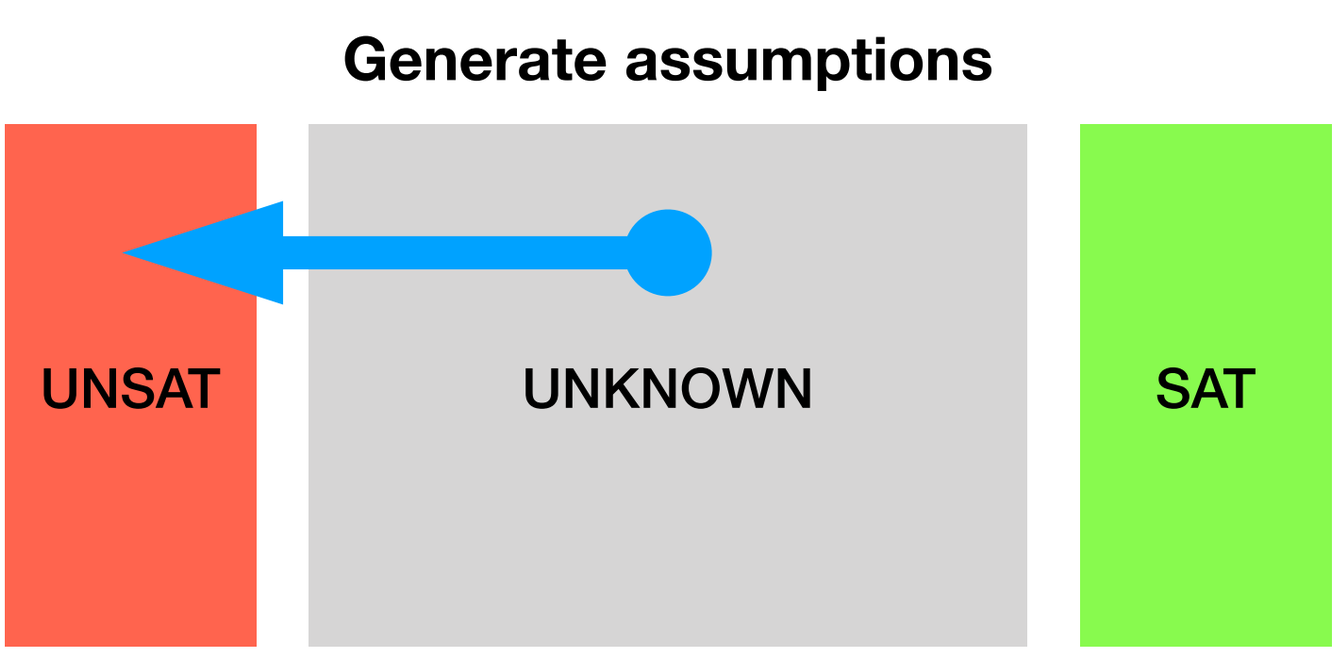


Satisfiability Modulo Theories and Oracles (SMTO)

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies (\rho \wedge B)$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho \wedge B$ is unsatisfiable

Unknown otherwise

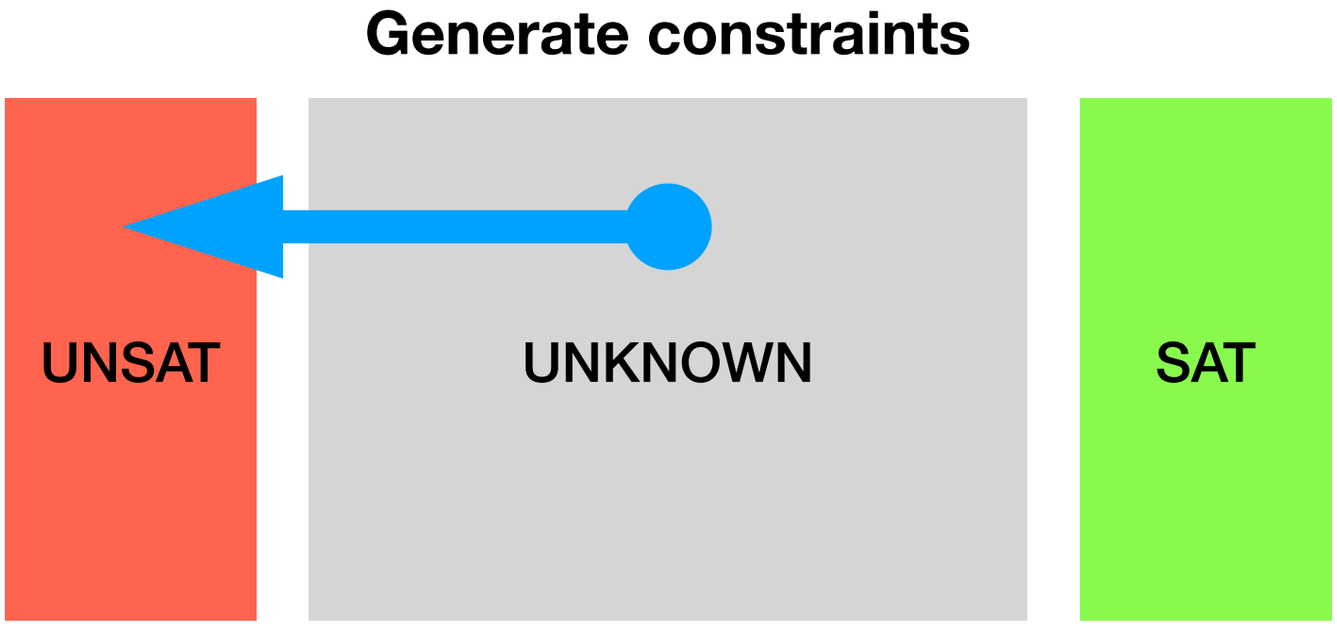


Satisfiability Modulo Theories and Oracles (SMTO)

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies (\rho \wedge B)$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho \wedge B$ is unsatisfiable

Unknown otherwise

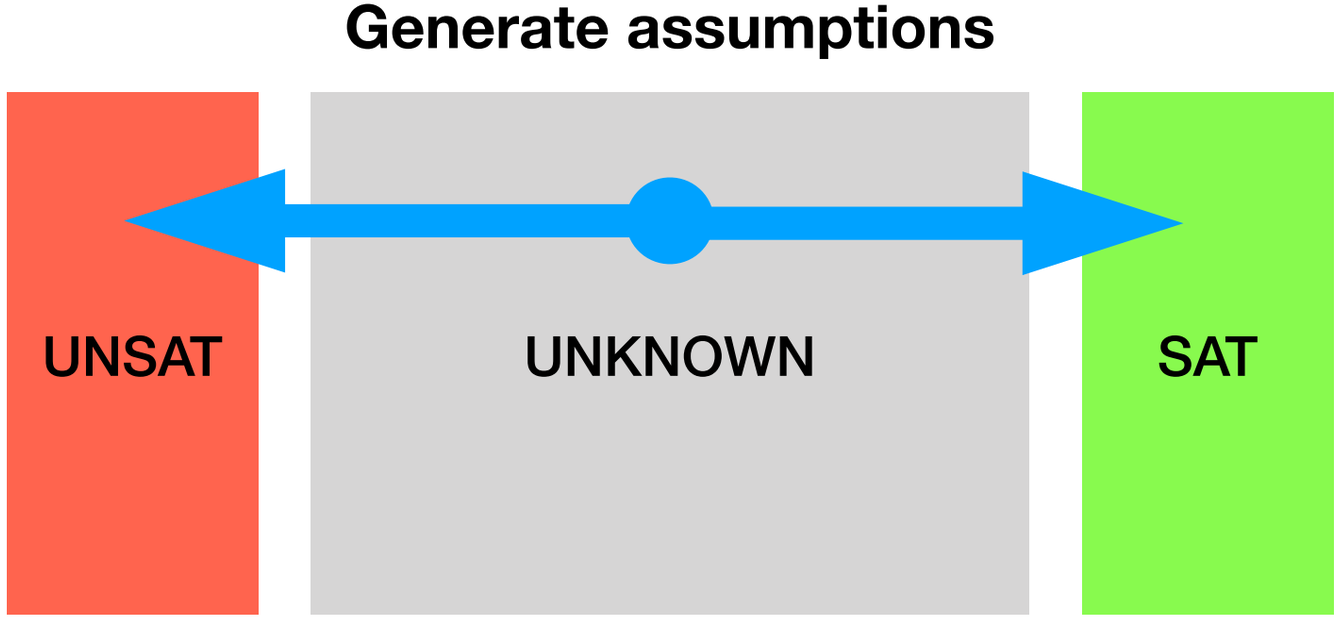


Satisfiability Modulo Theories and Oracles (SMTO)

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies (\rho \wedge B)$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho \wedge B$ is unsatisfiable

Unknown otherwise

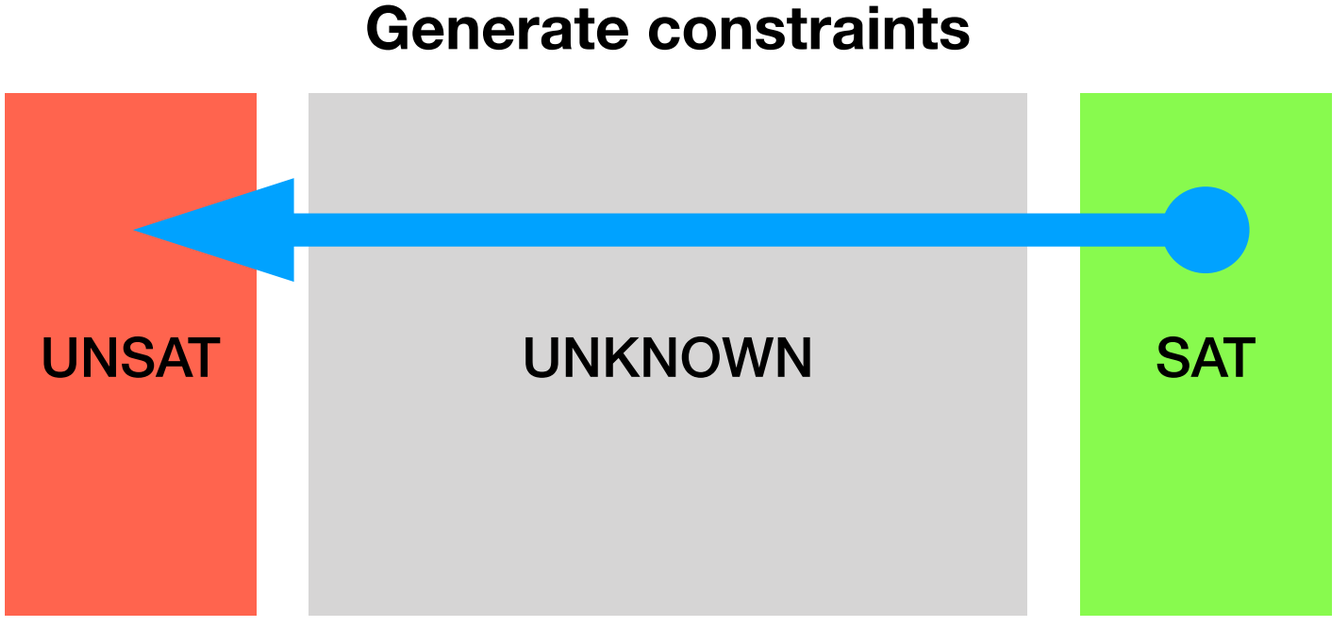


Satisfiability Modulo Theories and Oracles (SMTO)

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies (\rho \wedge B)$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho \wedge B$ is unsatisfiable

Unknown otherwise



Definitional

Satisfiability Modulo Theories and Oracles (SMTO)

A **definitional** SMTO problem is a tuple:

$\vec{f}$: a set of ordinary function symbols

$\overrightarrow{\theta}$: a set of oracle function symbols

ρ : a formula in a background theory

$\overrightarrow{\mathcal{O}}$: a set of oracle interfaces

And:

- All oracle interfaces define oracle functions
- There are no constraint generators

$\Longrightarrow$ No conflicting assumptions (only functional oracles)

$\Longrightarrow$ No changeable results (no constraints)

Coming up

- Existing use of oracles

- Formal definition of oracle interfaces

- SMT with oracles (**SMTO**):
  - when is it satisfiable/unsatisfiable
  - algorithm

- Synthesis with oracles (**SyMO**):
  - when are solutions correct
  - unifying algorithm for solving

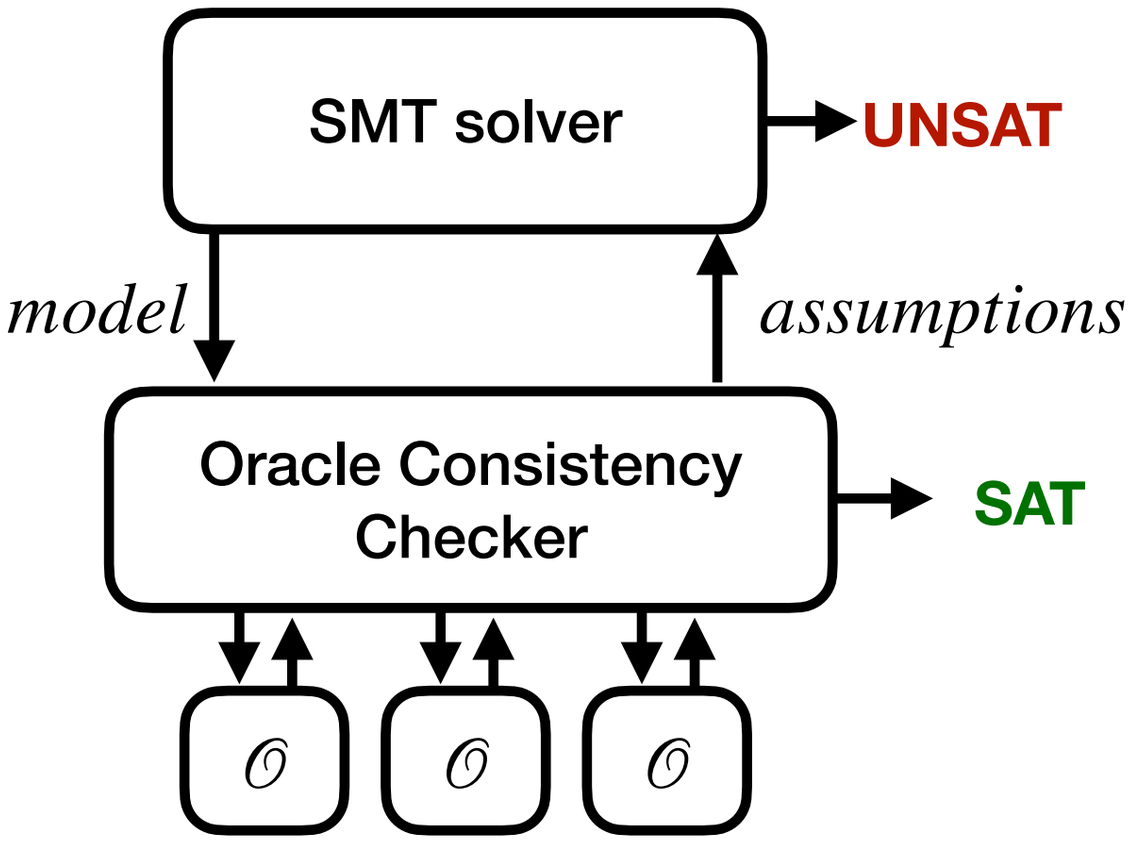
- More cat pictures

- Prototype evaluation

Solving (definitional) SMTO

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies \rho$ is satisfiable

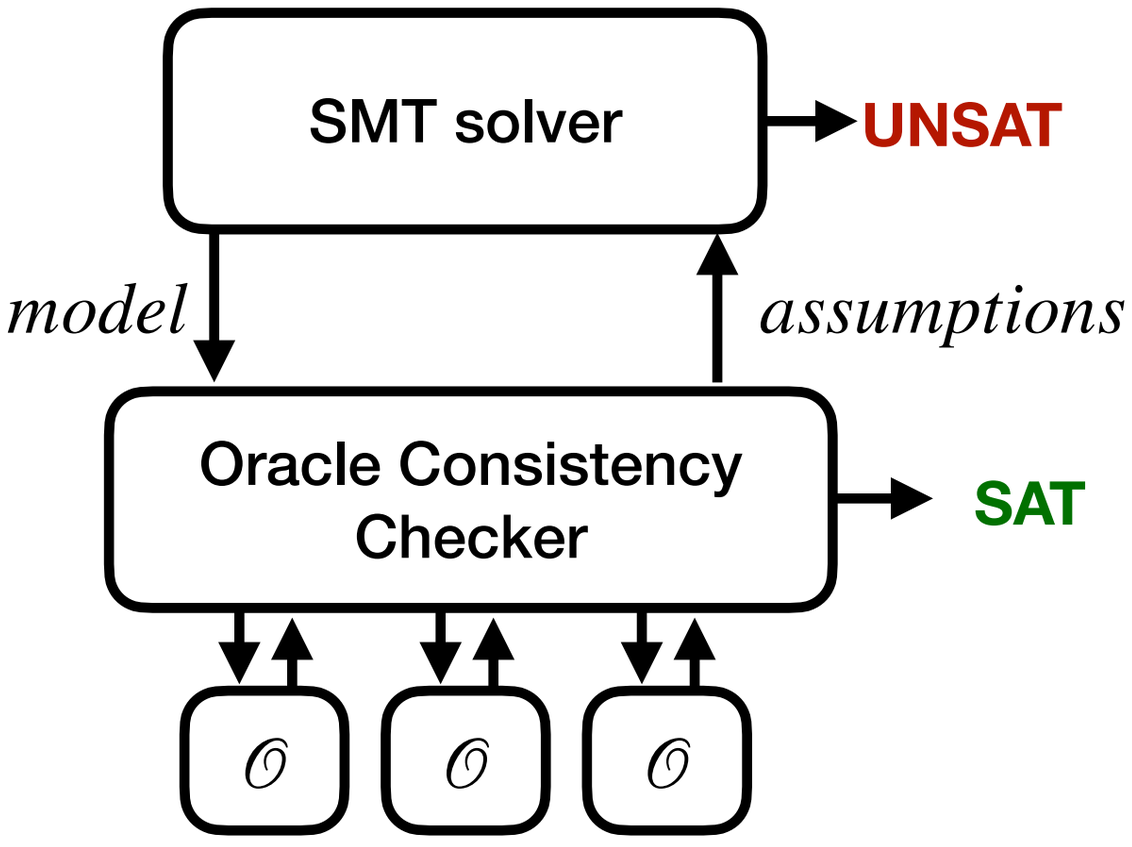
Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho$ is unsatisfiable



Solving (definitional) SMTO

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies \rho$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho$ is unsatisfiable

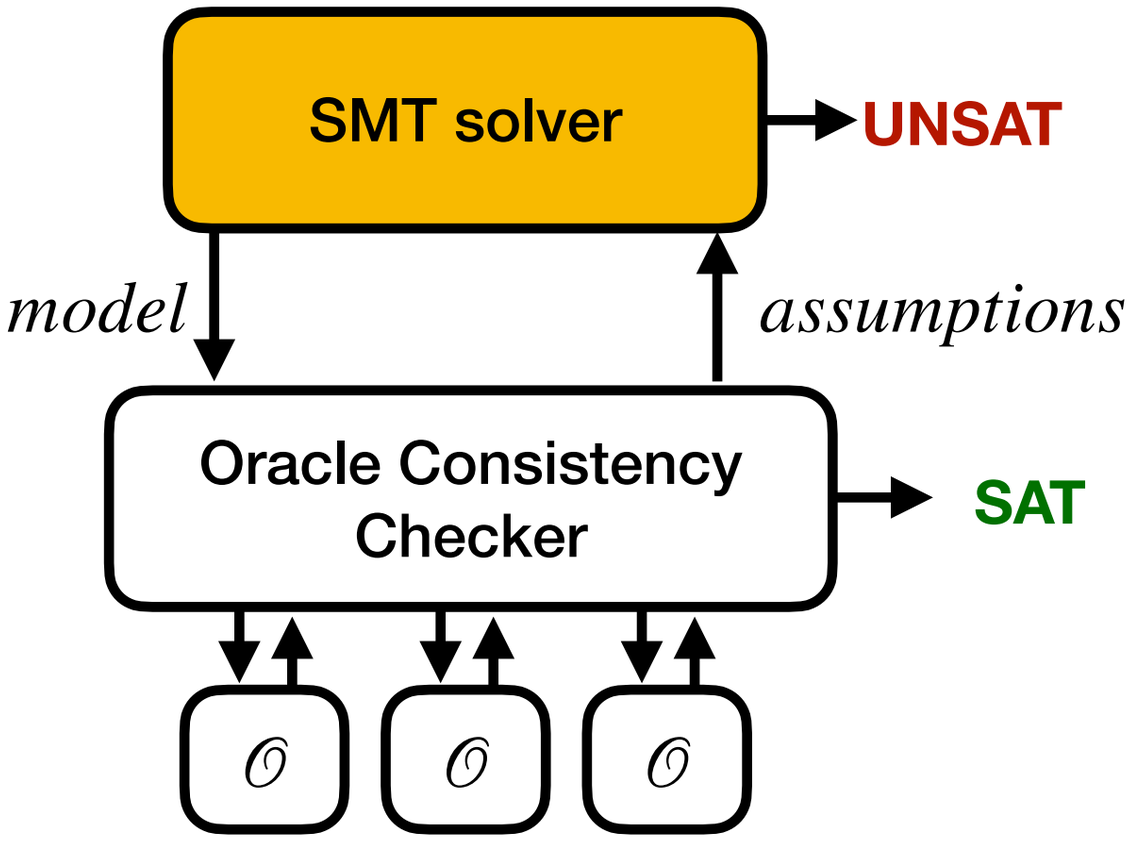



```
A ← true
while(true)
{
 if (ρ ∧ A) is UNSAT.
   return UNSAT
 else
   if(model consistent with O⃗)
     return SAT
   else
     conjoin assumptions to A
}
```

Solving (definitional) SMTO

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies \rho$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho$ is unsatisfiable

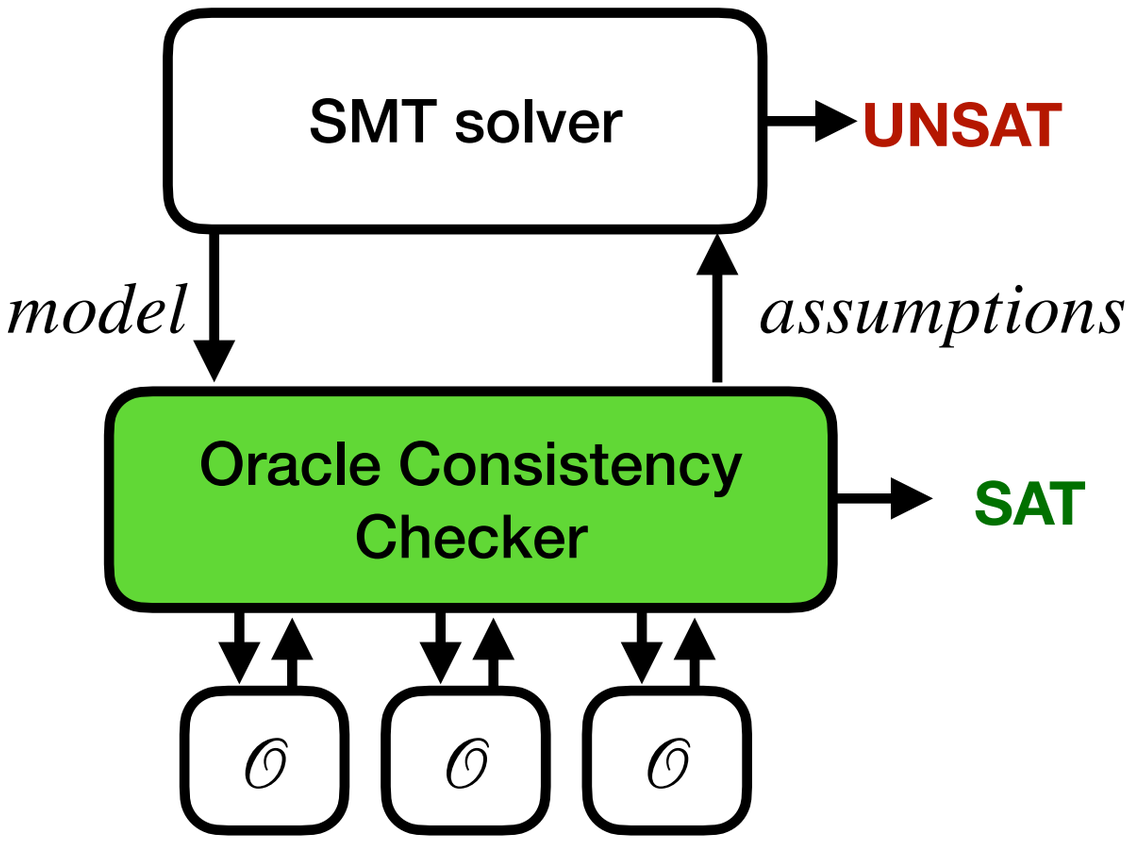



```
A ← true
while(true)
{
 if (ρ ∧ A) is UNSAT.
   return UNSAT
 else
    if(model consistent with O⃗)
      return SAT
    else
      conjoin assumptions to A
}
```

Solving (definitional) SMTO

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies \rho$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho$ is unsatisfiable




```
A ← true
while(true)
{
 if (ρ ∧ A) is UNSAT.
   return UNSAT
 else
   if(model consistent with O⃗)
     return SAT
   else
     conjoin assumptions to A
}
```

Coming up

- Existing use of oracles

- Formal definition of oracle interfaces

- SMT with oracles (**SMTO**):
  - when is it satisfiable/unsatisfiable
  - algorithm

- Synthesis with oracles (**SyMO**):
  - when are solutions correct
  - unifying algorithm for solving

- More cat pictures

- Prototype evaluation

Synthesis Modulo Oracles (SyMO)

A SyMO problem is a tuple:

$\vec{f}$: a tuple of functions to synthesise

$\overrightarrow{\theta}$: a set of oracle function symbols

$\forall \overrightarrow{x} . \phi$: a formula in a background theory, where ϕ is quantifier-free

$\overrightarrow{\mathcal{O}}$: a set of oracle interfaces

$\vec{f} : \{f\}$

$\overrightarrow{\theta} : \{corr\}$

$\forall \overrightarrow{x} . \phi : \forall x . corr(f) \wedge f(x) < 256$

$\overrightarrow{\mathcal{O}} : \{\mathcal{O}_{corr}\}$

$\mathcal{O}_{corr}$

$\overrightarrow{y} : (f* : int \rightarrow int)$

$\vec{z} : (z_1 : Bool , z_2 : int , z_3 : int)$

$\alpha_{gen} : corr(f*) = z_1$

$\beta_{gen} : f(z_2) = z_3$

What is a valid f ?

Synthesis Modulo Oracles (SyMO)

What is a valid f ?

$$\forall x \,.\, \phi : \forall x \,.\, corr(f) \wedge f(x) < 256$$

f is VALID if **there is no** x such that ϕ is false

i.e., if the SMTO problem $(\overrightarrow{x}, \overrightarrow{\theta}, \neg\phi\{\vec{f} \rightarrow \overrightarrow{f^*}\}, \overrightarrow{\mathcal{O}})$ is UNSAT

f is INVALID if **there is an** x such that ϕ is false

i.e., if the SMTO problem $(\overrightarrow{x}, \overrightarrow{\theta}, \neg\phi\{\vec{f} \rightarrow \overrightarrow{f^*}\}, \overrightarrow{\mathcal{O}})$ is SAT

Synthesis Modulo Oracles (SyMO)

A SyMO problem is a tuple:

$\vec{f}$: a tuple of functions to synthesise

$\overrightarrow{\theta}$: a set of oracle function symbols

$\forall \overrightarrow{x} . \phi$: a formula in a background theory, where ϕ is quantifier-free

$\overrightarrow{\mathcal{O}}$: a set of oracle interfaces

And:

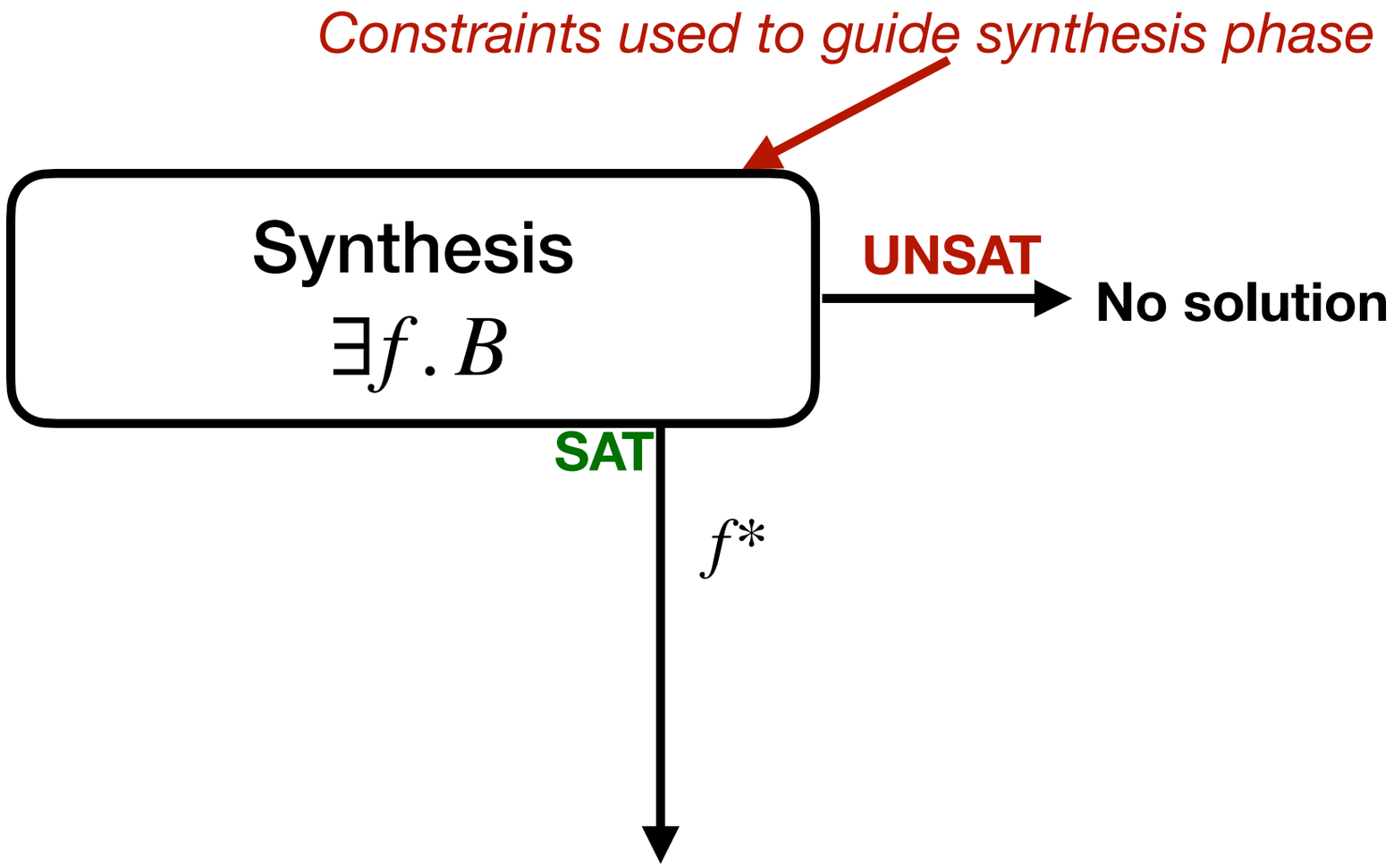
- All **assumption** generators define **oracle function symbols**
- All oracles are **functional**

$\Longrightarrow$ checking $\vec{f}$ is valid is now **definitional SMTO**

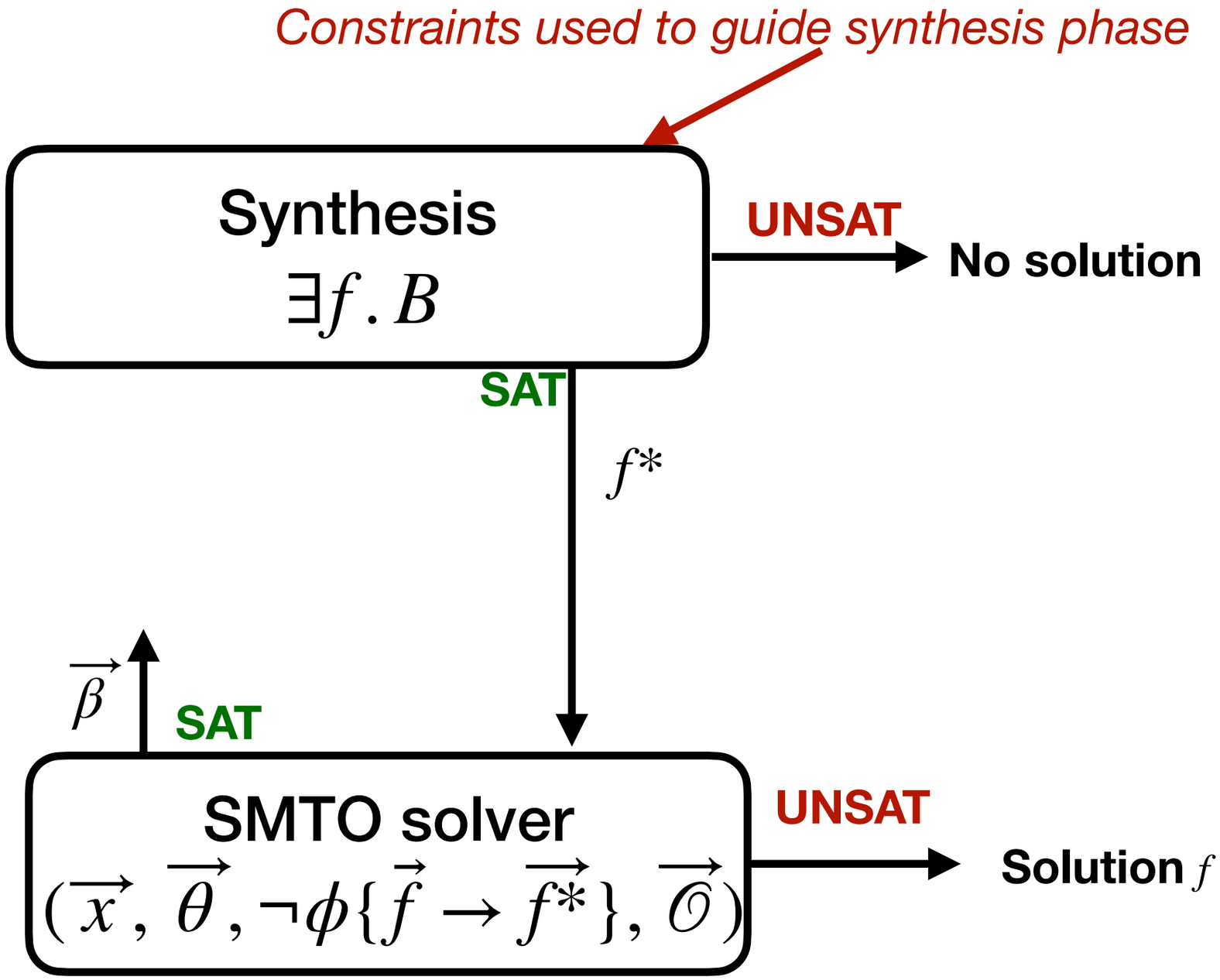
Coming up

- Existing use of oracles
- Formal definition of oracle interfaces
- SMT with oracles (**SMTO**):
  - when is it satisfiable/unsatisfiable
  - algorithm
- Synthesis with oracles (**SyMO**):
  - when are solutions correct
  - unifying algorithm for solving
- More cat pictures
- Prototype evaluation

Solving SyMO

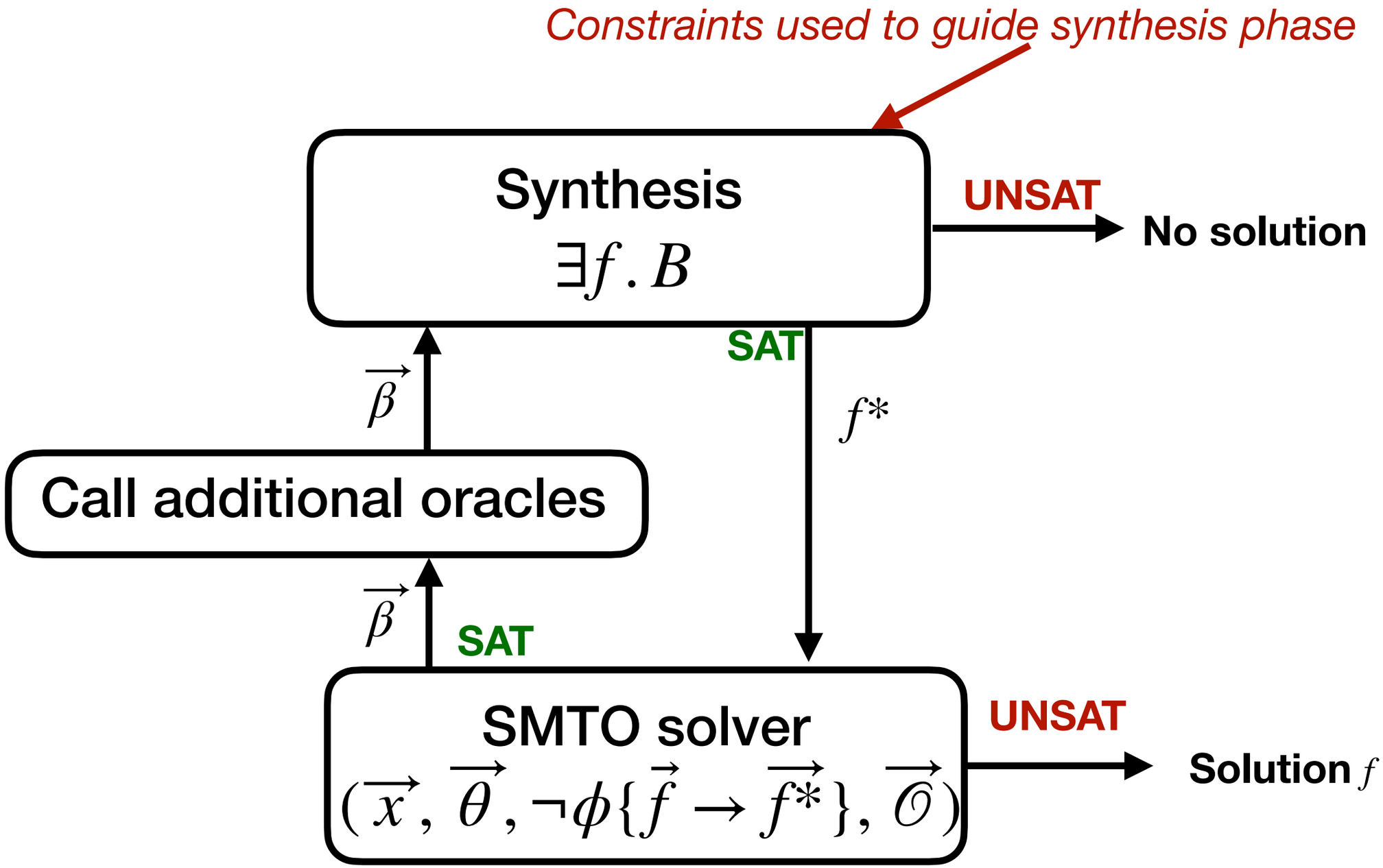


Solving SyMO

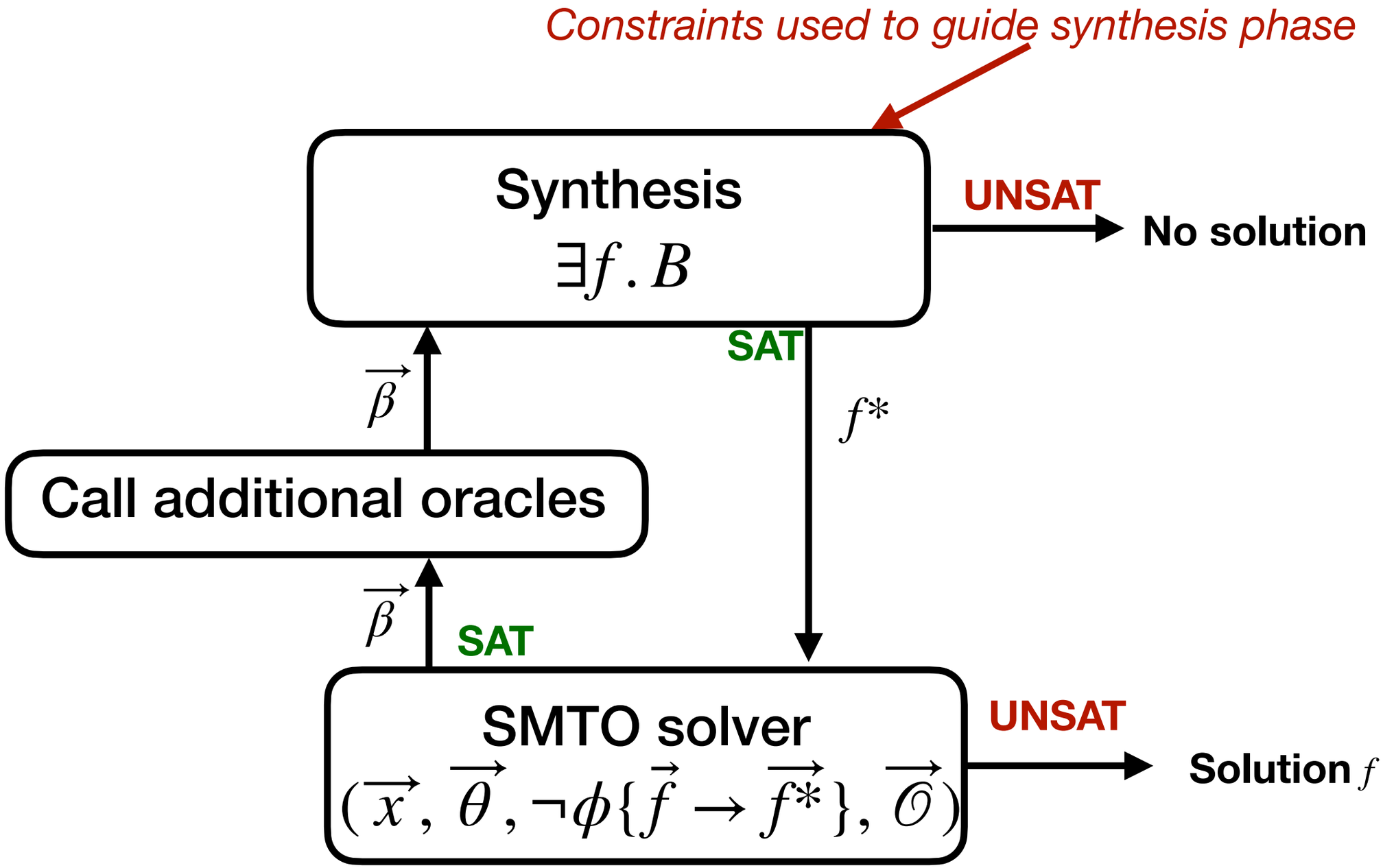


We also pass the assignment model back from the SMTO solver as a constraint

Solving SyMO



Solving SyMO



Note: $f*$ is guaranteed to satisfy B so the SMTO solver doesn't need to consider B

Thus definitional SMTO!

Coming up

- Existing use of oracles

- Formal definition of oracle interfaces

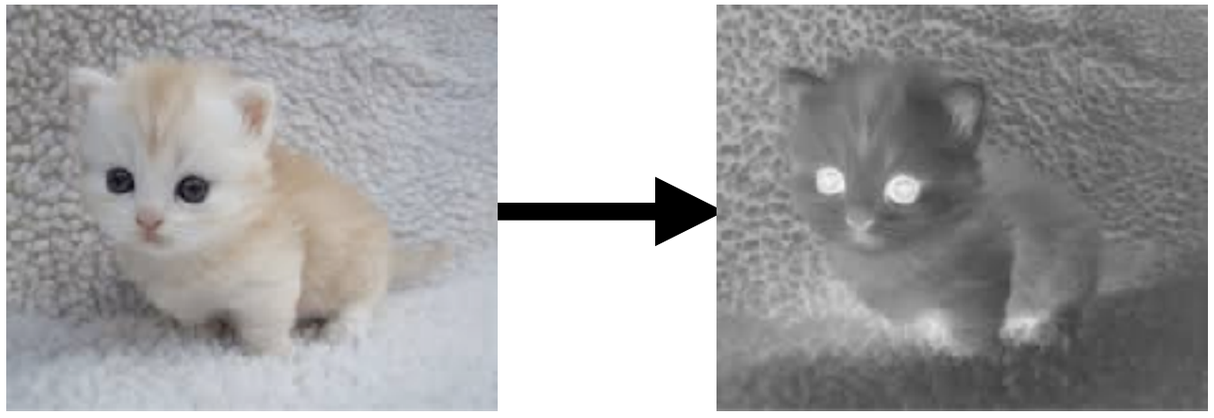
- SMT with oracles (**SMTO**):
  - when is it satisfiable/unsatisfiable
  - algorithm

- Synthesis with oracles (**SyMO**):
  - when are solutions correct
  - unifying algorithm for solving

- More cat pictures

- Prototype evaluation

Find a function that transforms Cat A into Cat B?



$$\exists f . \forall x . corr(f)$$

Oracle:

$f^* : int \rightarrow int$

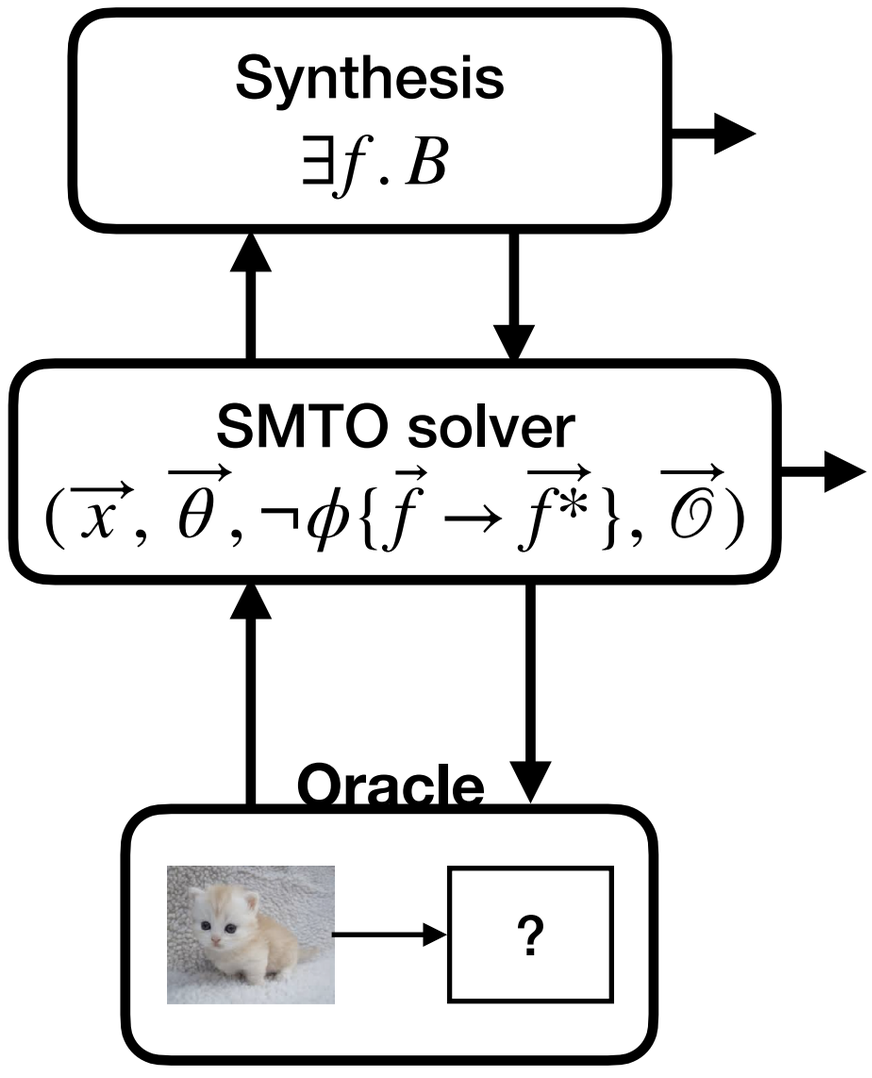
- Load image
- Compile f
- Apply f to all pixels
- Output image

$(z_1 : int,\ z_2 : int,\ z_3 : int)$

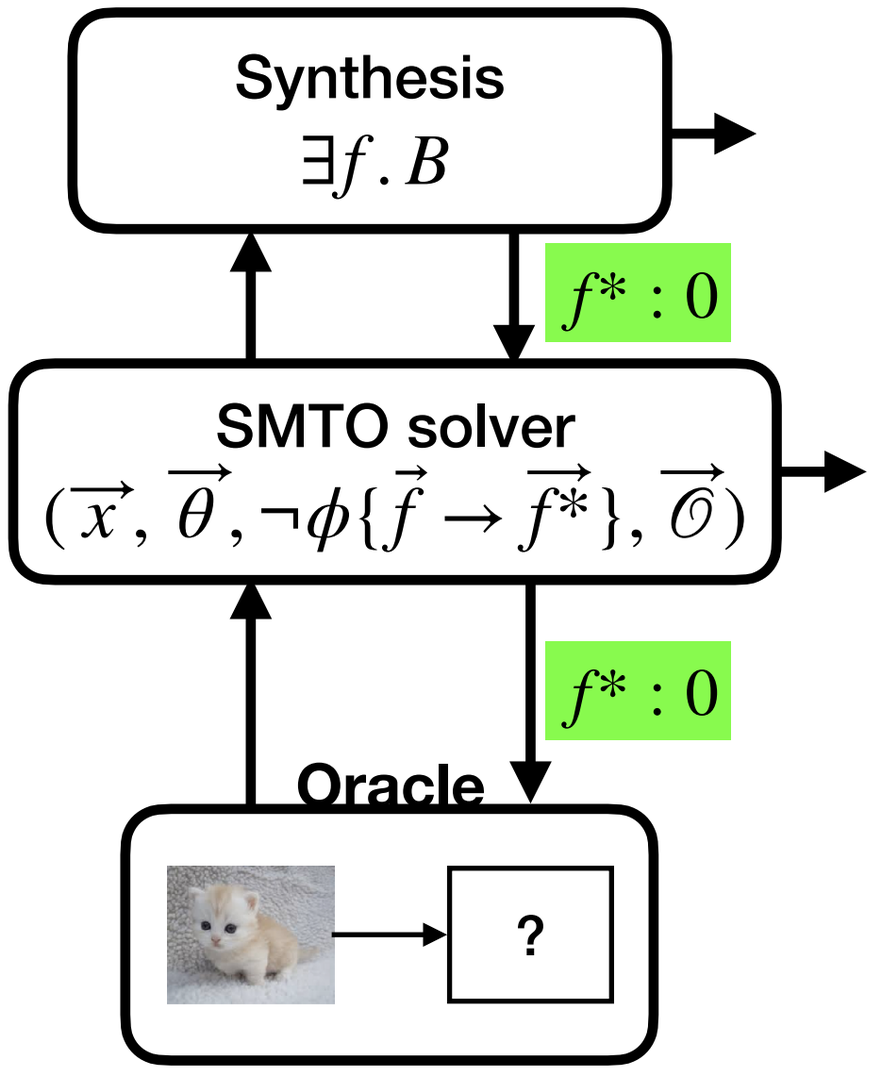
$\beta_{gen} : f(z_1) = z_2$

$\alpha_{gen} : corr(f^*) = z_3$

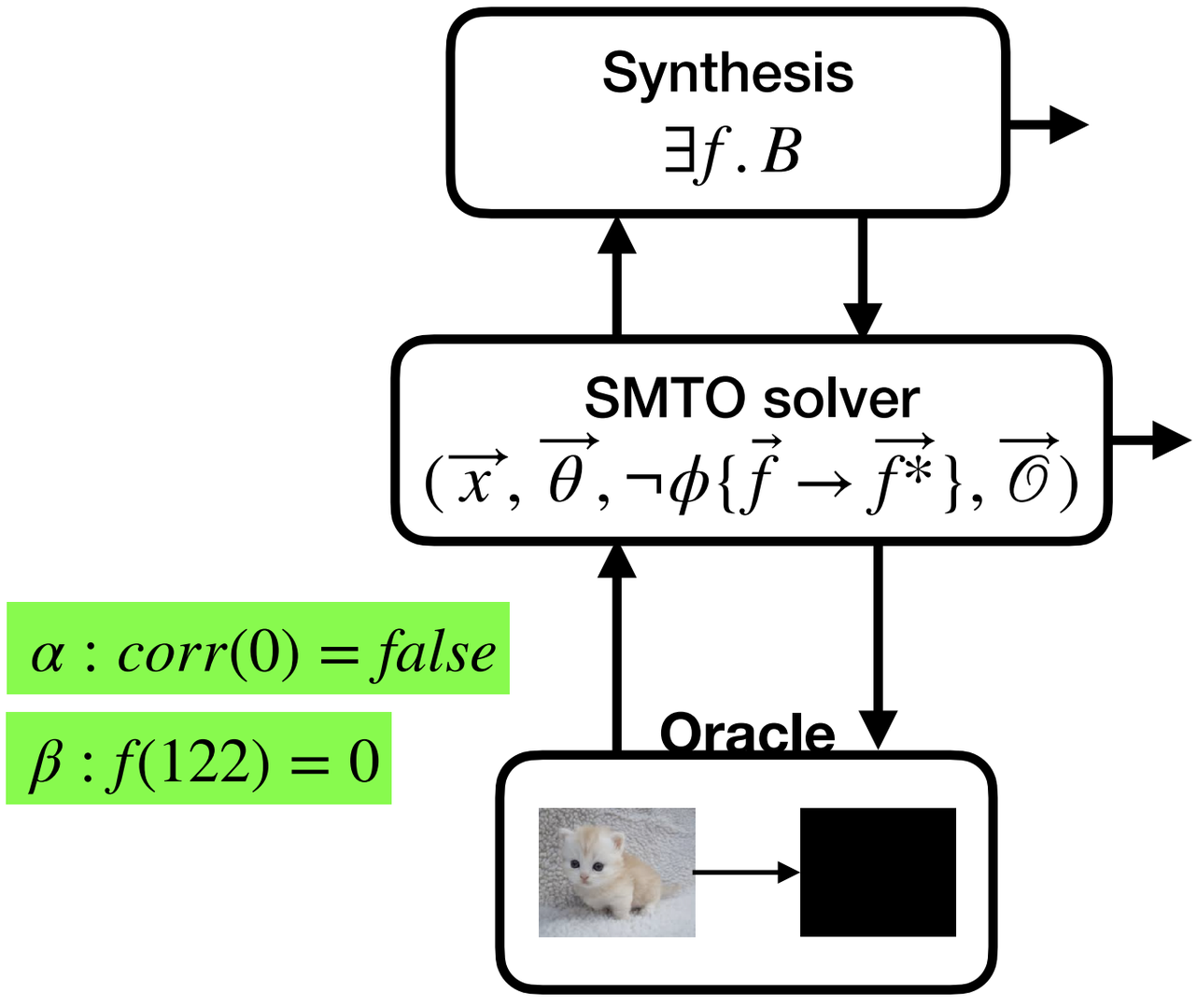
$\phi : \ corr(f)$
$A : \top$
$B : \top$



$\phi : corr(f)$
$A : \top$
$B : \top$



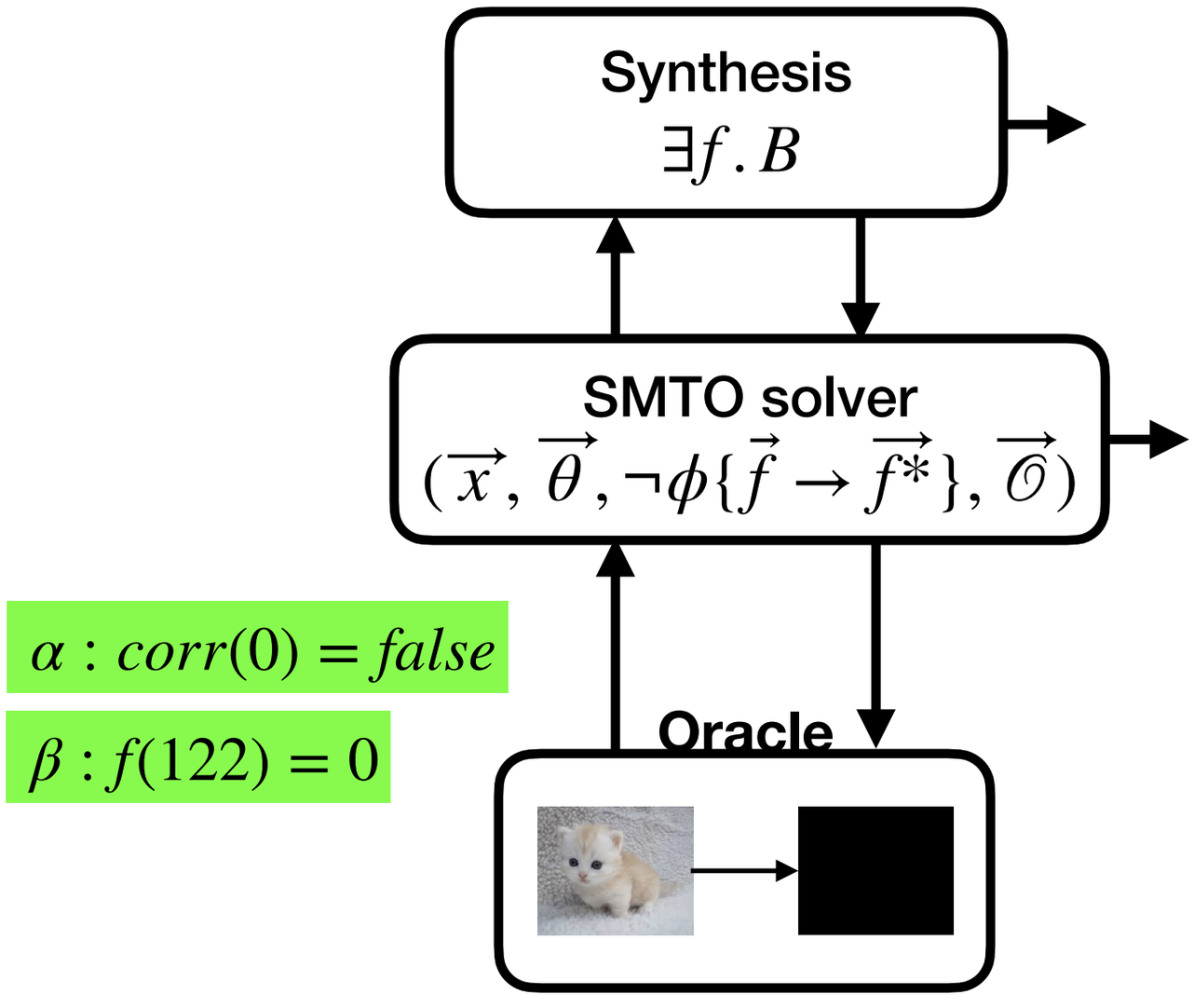
$\phi : corr(f)$
$A : \top$
$B : \top$



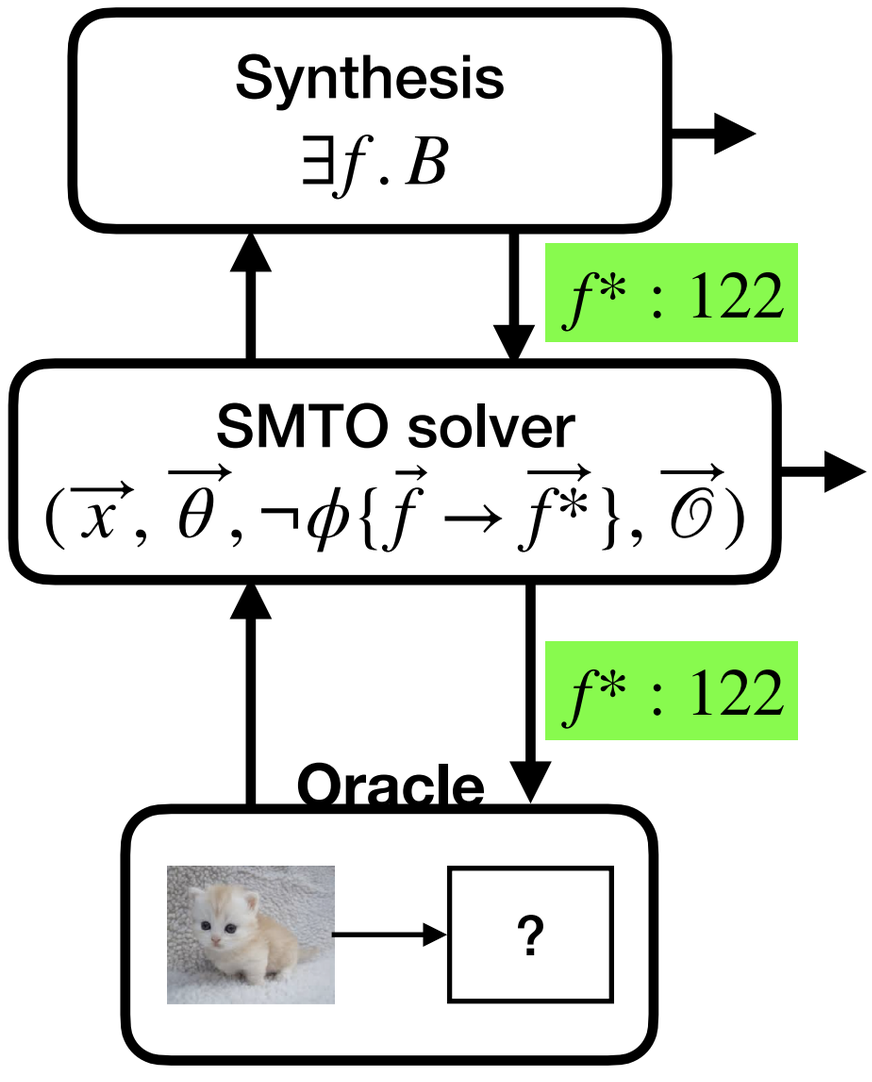
$\alpha : corr(0) = false$

$\beta : f(122) = 0$

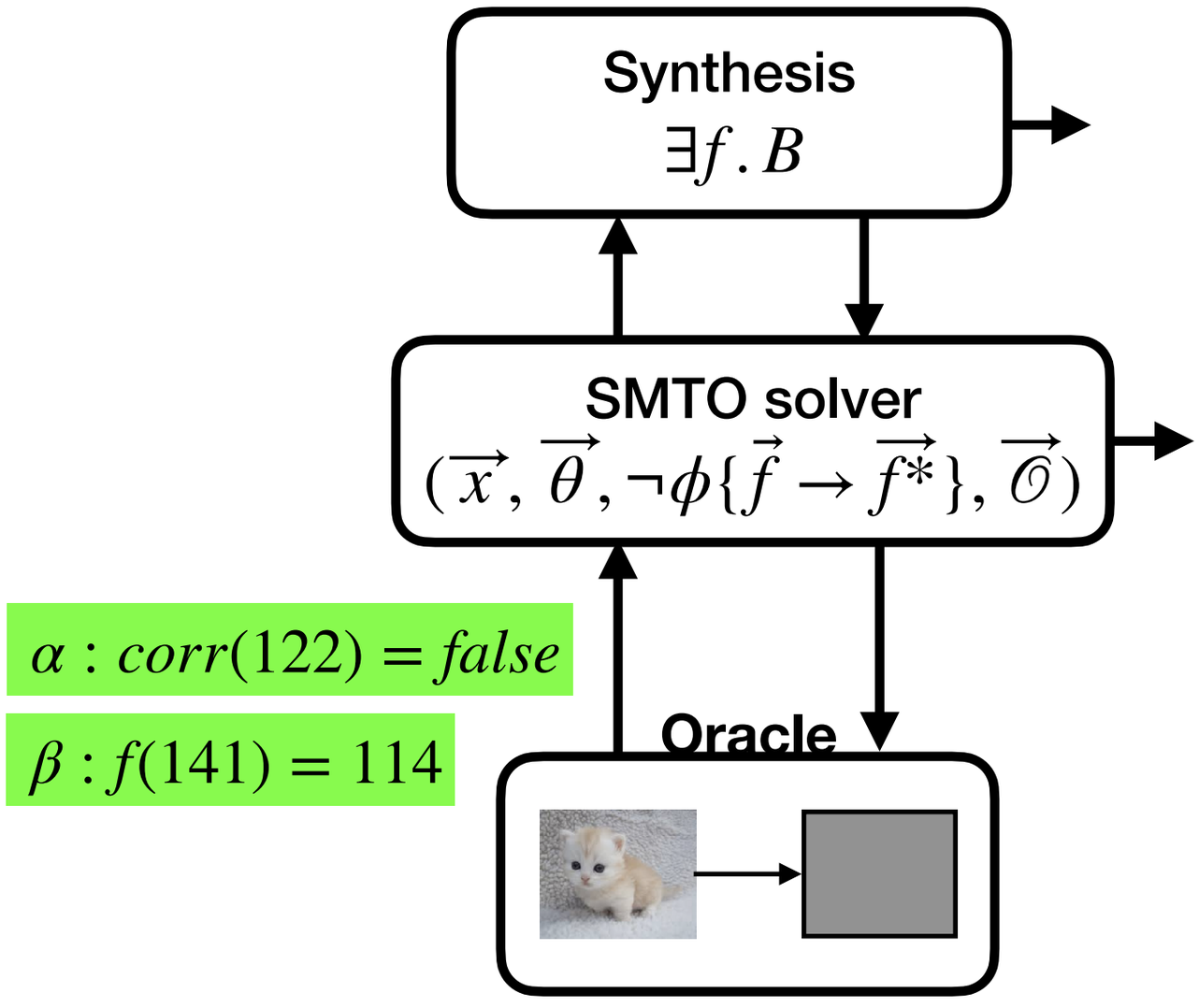
$\phi : \ corr(f)$
$A : \ \top \wedge corr(0) = false$
$B : \ \top \wedge f(122) = 0$



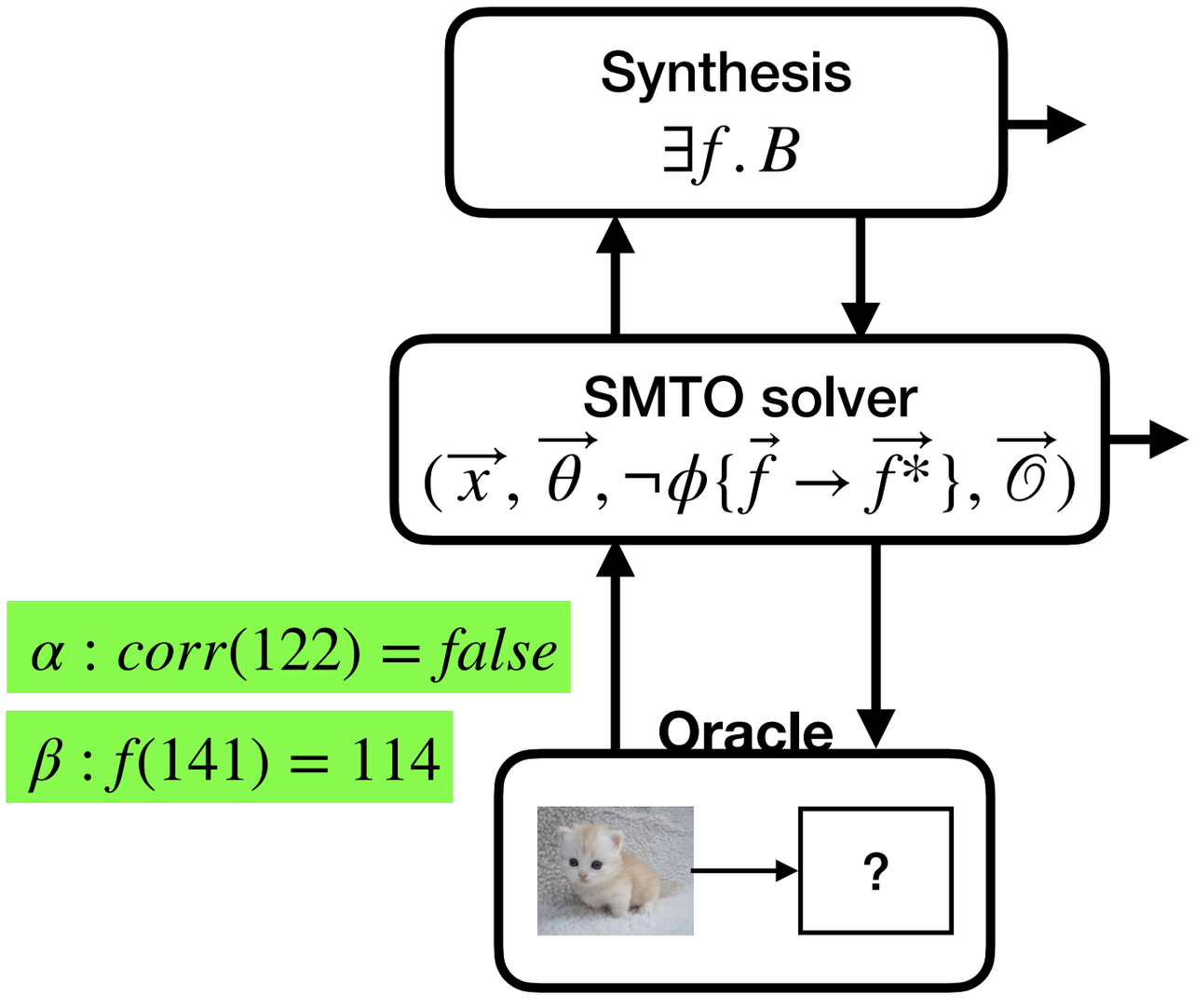
$\phi : \ corr(f)$
$A : \ \top \wedge corr(0) = false$
$B : \ \top \wedge f(122) = 0$



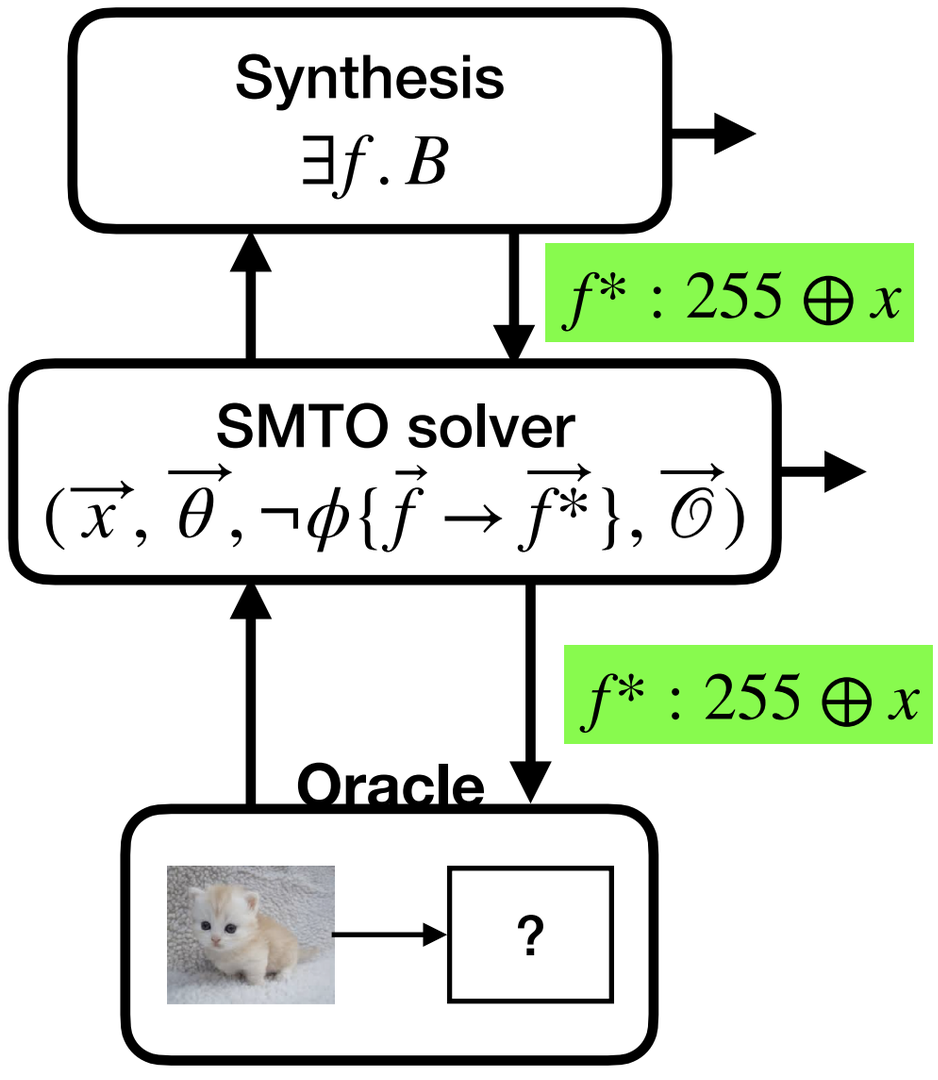
$\phi : \ corr(f)$
$A : \ \top \wedge corr(0) = false$
$B : \ \top \wedge f(122) = 0$



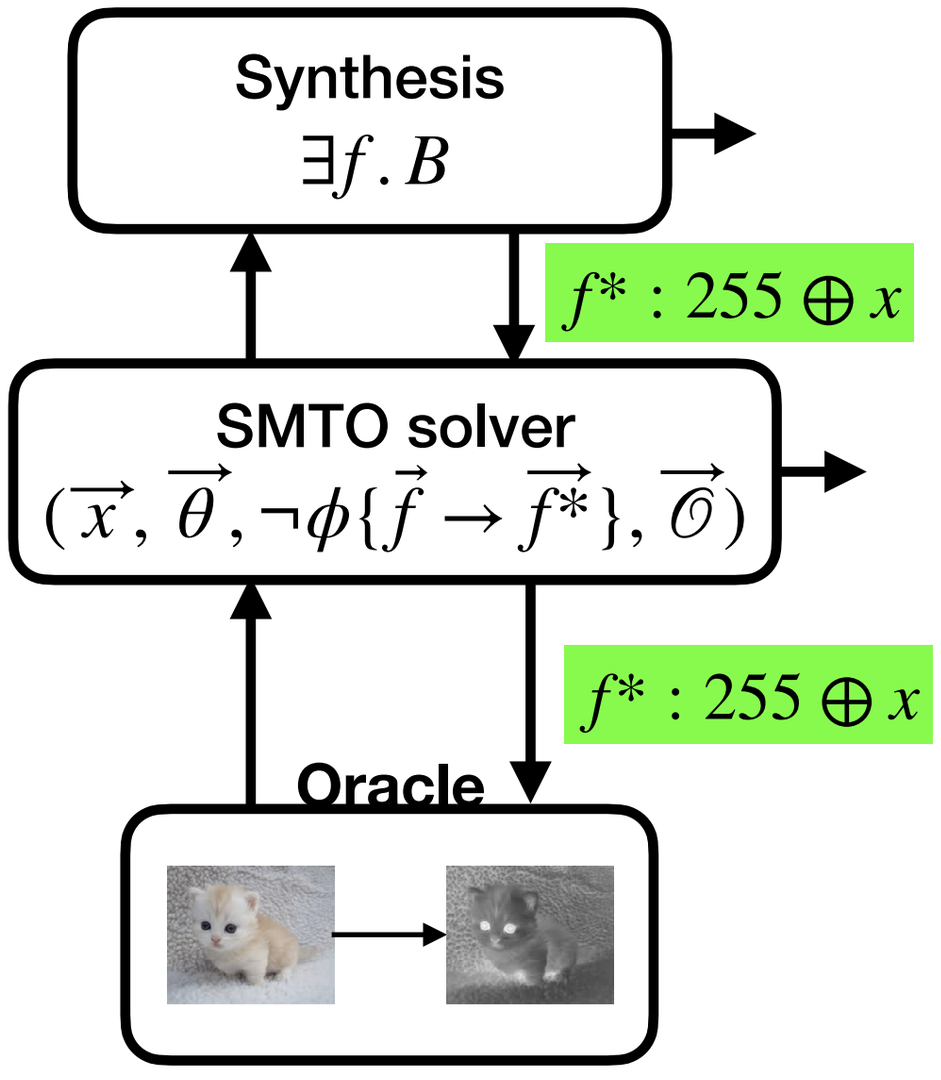
$\phi : \ corr(f)$
$A : \ \top \wedge corr(0) = false \wedge corr(122) = false$
$B : \ \top \wedge f(122) = 0 \wedge f(141) = 114$



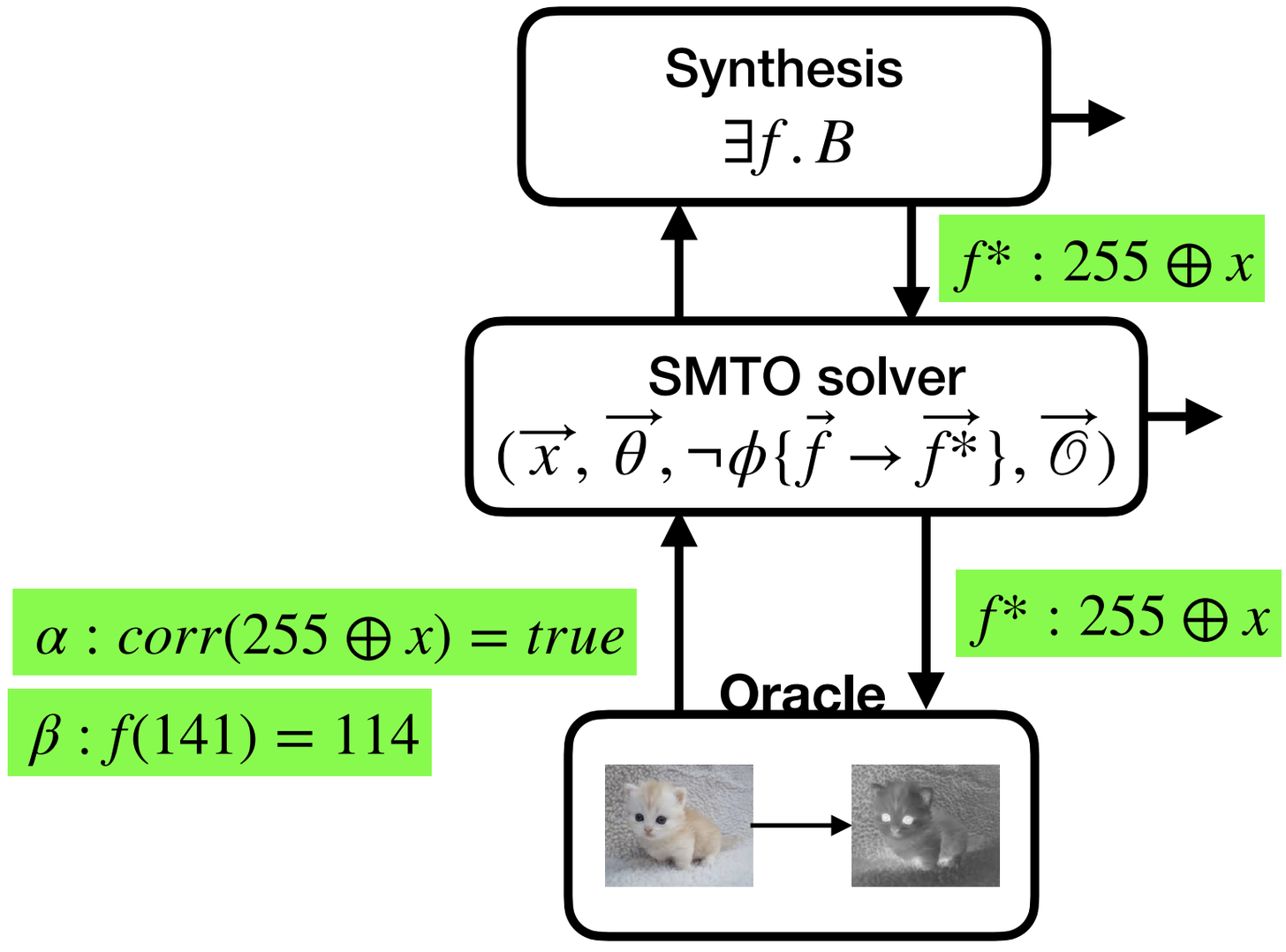
$$\phi : \ corr(f)$$
$$A : \ \top \wedge corr(0) = false \wedge corr(122) = false$$
$$B : \ \top \wedge f(122) = 0 \wedge f(141) = 114$$



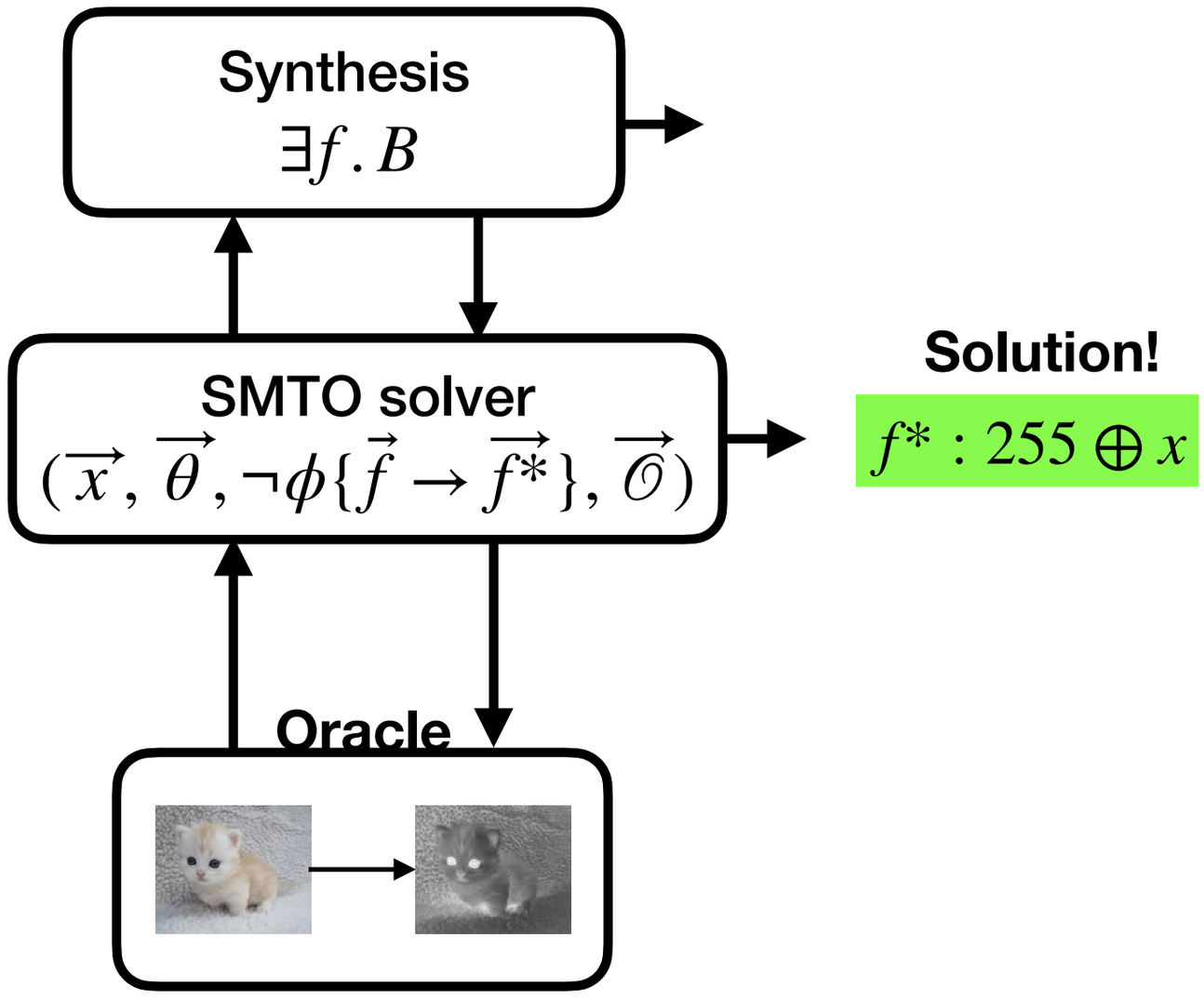
$$\phi : \ corr(f)$$
$$A : \ \top \wedge corr(0) = false \wedge corr(122) = false$$
$$B : \ \top \wedge f(122) = 0 \wedge f(141) = 114$$



$$\phi : corr(f)$$
$$A : \top \wedge corr(0) = false \wedge corr(122) = false$$
$$B : \top \wedge f(122) = 0 \wedge f(141) = 114$$



$$\phi : \ corr(f)$$
$$A : \ \top \wedge corr(0) = false \wedge corr(122) = false$$
$$B : \ \top \wedge f(122) = 0 \wedge f(141) = 114$$



Coming up

- Existing use of oracles
- Formal definition of oracle interfaces
- SMT with oracles (**SMTO**):
  - when is it satisfiable/unsatisfiable
  - algorithm
- Synthesis with oracles (**SyMO**):
  - when are solutions correct
  - unifying algorithm for solving
- More cat pictures
- Prototype evaluation

Case studies

Approximate model

	Problem	#	Delphi (oracles) #	Delphi (oracles) s	CVC5 (no oracles) #	CVC5 (no oracles) s
SyMO	Images	10	9	21.6s	0	
	Control stability	112	104	29.3s	16	19.4s
	Control safety	112	31	59.9s	0	
	PBE	150	148	0.5s	150	<0.5s
SMTO	Math	12	9	<0.5s	5	2.2s

https://github.com/polgreen/delphi

https://github.com/polgreen/oracles

SyGuS-IF extension

- Syntax for declaring oracle constraints, oracle assumptions and oracle functions

$$
\begin{array}{rcl}
\langle OracleCmd \rangle & ::= & \texttt{(oracle-assume (} \langle SortedVar \rangle^* \texttt{) (} \langle SortedVar \rangle^* \texttt{)} \ \langle Term \rangle \ \langle Symbol \rangle \ \texttt{)} \\
& | & \texttt{(oracle-constraint (} \langle SortedVar \rangle^* \texttt{) (} \langle SortedVar \rangle^* \texttt{)} \ \langle Term \rangle \ \langle Symbol \rangle \ \texttt{)} \\
& | & \texttt{(declare-oracle-fun} \ \langle Symbol \rangle \ \texttt{(} \langle Sort \rangle^* \texttt{)} \ \langle Sort \rangle \ \langle Symbol \rangle \ \texttt{)} \\
& | & \texttt{(oracle-constraint-io} \ \langle Symbol \rangle \ \langle Symbol \rangle \ \texttt{)} \\
& | & \texttt{(oracle-constraint-cex} \ \langle Symbol \rangle \ \langle Symbol \rangle \ \texttt{)} \\
& | & \texttt{(oracle-constraint-membership} \ \langle Symbol \rangle \ \langle Symbol \rangle \ \texttt{)} \\
& | & \texttt{(oracle-constraint-poswitness} \ \langle Symbol \rangle \ \langle Symbol \rangle \ \texttt{)} \\
& | & \texttt{(oracle-constraint-negwitness} \ \langle Symbol \rangle \ \langle Symbol \rangle \ \texttt{)} \\
& | & \texttt{(declare-correctness-oracle} \ \langle Symbol \rangle \ \langle Symbol \rangle \ \texttt{)} \\
& | & \texttt{(declare-correctness-cex-oracle} \ \langle Symbol \rangle \ \langle Symbol \rangle \ \texttt{)}
\end{array}
$$

https://github.com/SyGuS-Org/docs

Conclusion

- Defined oracle interfaces, grouping responses into assumptions and constraints
- Presented algorithms for SMTO and SyMO
- Performs synthesis with complex oracles without building new solvers

Conclusion

Oracles can be (almost) anything!

- Must be able to say “yes” or “no” for correctness
- Must provide semantic guidance to the learner
- Can be used with traditional SMT constraints

Do you have oracles? Talk to us!

elizabeth.polgreen@ed.ac.uk

ajreynolds@gmail.com

sseshia@berkeley.edu