Satisfiability and Synthesis Modulo Oracles

Elizabeth Polgreen, Andrew Reynolds, Sanjit A. Seshia

Why Satisfiability Modulo Oracles (SMTO)

Find a prime number between 15 and 18

Easy?

Not for state-of-the-art SMT Solvers like CVC5 ☹

Why?

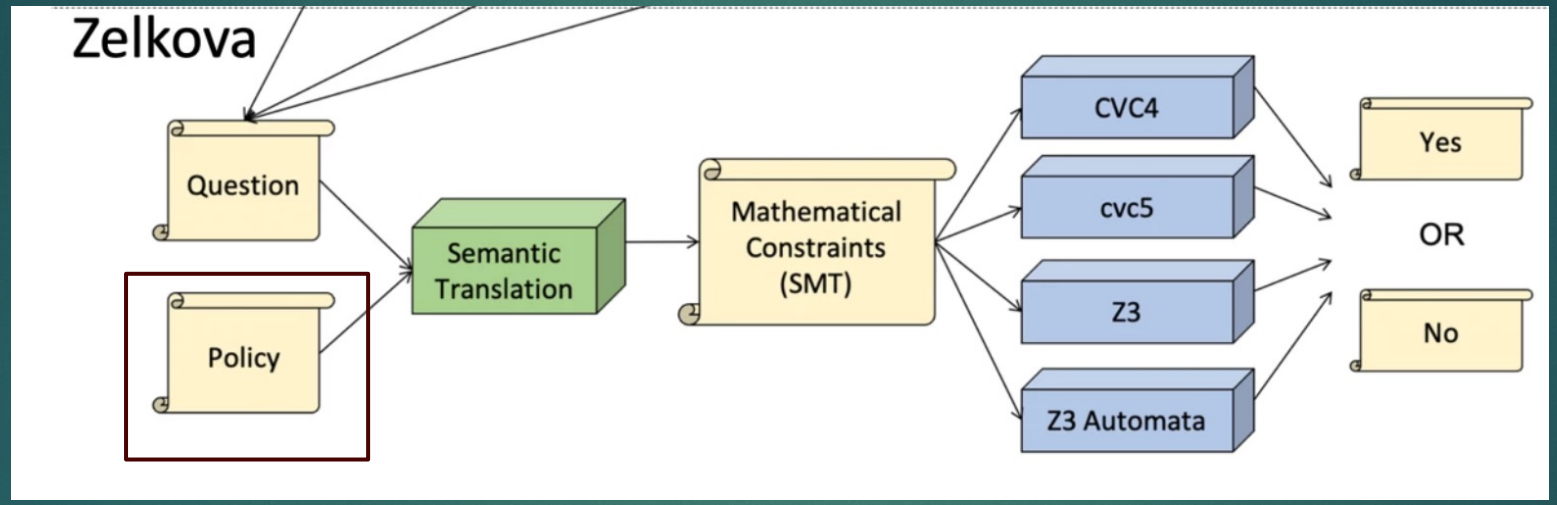
```
(define-fun-rec isPrimeRec ((a Int) (b Int)) Bool
(ite (> b (div a 2)) true
(ite (= (mod a b) 0)
false
(isPrimeRec a (+ b 1)))))

(define-fun isPrime ((a Int)) Bool
(ite (<= a 1)
false
(isPrimeRec a 2)))

(declare-const p Int)
(assert (isPrime p))
(assert (and (>= 15 p) (<= p 18)))
```

Too complex to reason about

Why Satisfiability Modulo Oracles (SMTO)

A more practical example!

AWS uses a tool called Zelkova to reason about Access Control Policies



"Allows access to a resource if a string S in the request context is numerically less than 42"

How to encode $realValueOfStr(s) < 42$ in SMT

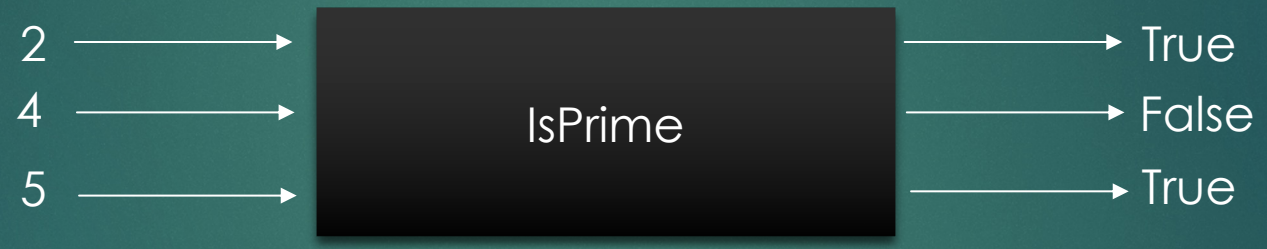
Why don't off-the-shelf solvers work?

1. Some parts are hard to reason about
   - isPrime
2. Some parts are hard to model in SMT
   - Type-casting semantics

Solution: Use "executable" oracles!!

What are oracles?

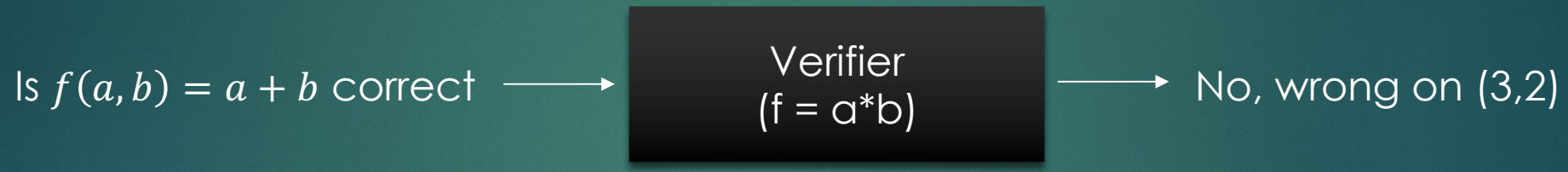
A component (code, binary, anything) that can be queried!



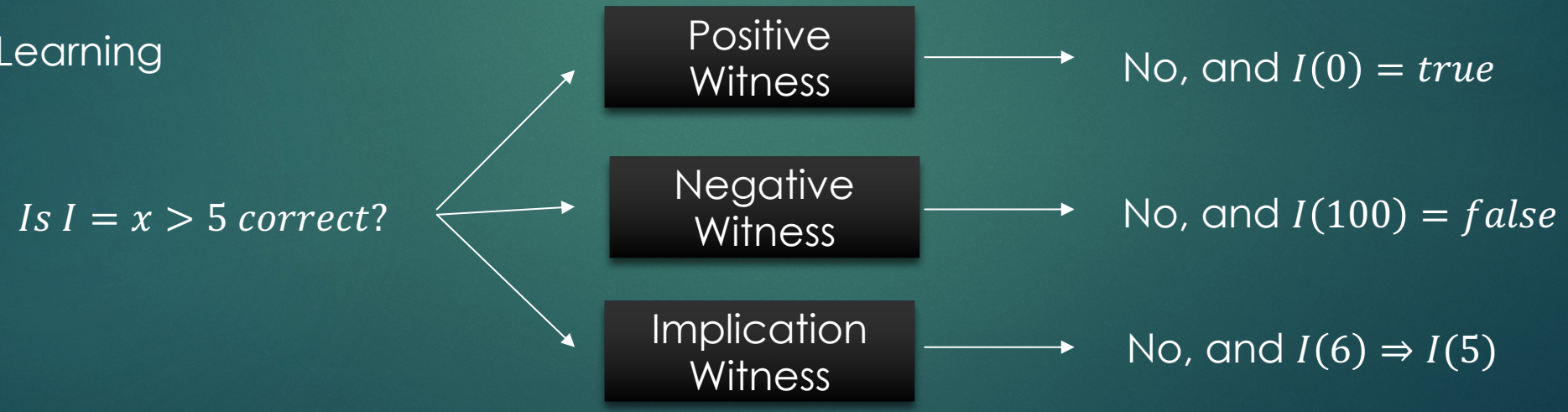
Existing use of oracles

Many algorithms already use oracles!

1. Counter-example guided inductive synthesis



2. ICE Learning



[1] Combinatorial sketching for finite programs. Solar-Lezama et. al [2] ICE: A robust framework for learning invariants Garg et. al

What does this work really do?

People use oracles but build their custom solvers

As the solver needs custom information about the oracle
Like what does the response from oracle mean

This work proposes a unifying way to:

- Define oracles and how they interact with the solvers
- Give a unifying algorithm to solve SMT and Synth problems in presence of oracles

What are oracle interfaces?

Oracle interface defines how an oracle and the solver interact!

1. **Query Domain** → Defines oracle inputs (y)
2. **Response Co-Domain** → Defines oracle outputs (z)
3. **Assumption Generator** → $\alpha_{gen}(y, z)$

   Generates assumptions $\boldsymbol{A}$ that the solver can make!

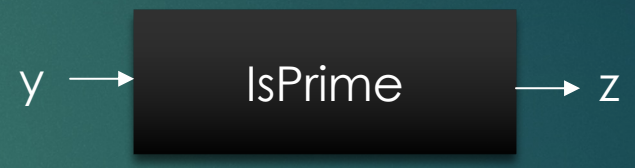
   $$\boldsymbol{A} \Rightarrow \rho$$

4. **Constraints Generator** → β_{gen}(y, z)

   Generates constraints $\boldsymbol{B}$ that the solver must satisfy!

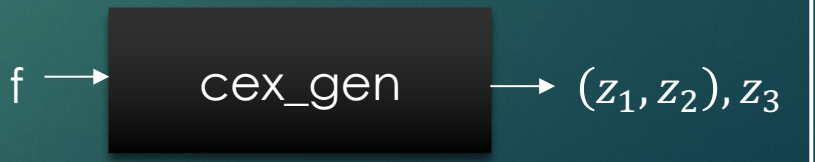
   $$\boldsymbol{A} \Rightarrow \rho \wedge \boldsymbol{B}$$

$isPrime(y) = z$



$isPrime(2) = True$
$isPrime(4) = False$
$isPrime(5) = True$
…

$f(z_1, z_2) = z_3$



$f(2,2) = 4$
$f(3,2) = 6$
…

SMTO Formulation

- $\vec{f}$ Ordinary (non-oracle) function symbols
- $\vec{\theta}$ Oracle function symbols (that are modelled using oracles)
- $\vec{I}$ Oracle interfaces modelling the oracle symbols
- A formula ρ

SATISFIABLE: $\exists \vec{f} \cdot \forall \vec{\theta} \cdot A \Rightarrow (\rho \wedge B)$ is satisfiable.

where A is the set of assumptions generated by the oracle interfaces,
and B is the set of constraints generated by the oracle interfaces

UNSATISFIABLE: $\exists \vec{f} \cdot \exists \vec{\theta} \cdot A \wedge \rho \wedge B$ is unsatisfiable.

Instability of Unrestricted SMTO

SATISFIABLE: $\exists \vec{f} \cdot \forall \vec{\theta} \cdot A \Rightarrow (\rho \wedge B)$ is satisfiable.

UNSATISFIABLE: $\exists \vec{f} \cdot \exists \vec{\theta} \cdot A \wedge \rho \wedge B$ is unsatisfiable.

Conflicting Oracle Results!

Say after i calls to the oracle, the assumption set A becomes unsatisfiable.

The problem becomes both SAT and UNSAT

A similar problem can occur if the constraints set B is satisfiable and then becomes unsat.

Definitional SMTO

A fragment of SMTO is definitional if:

- All oracles are *functional!*
- Generated assumptions $\boldsymbol{A}$ are of the form $\theta(y_1, y_2, \cdots) = z$

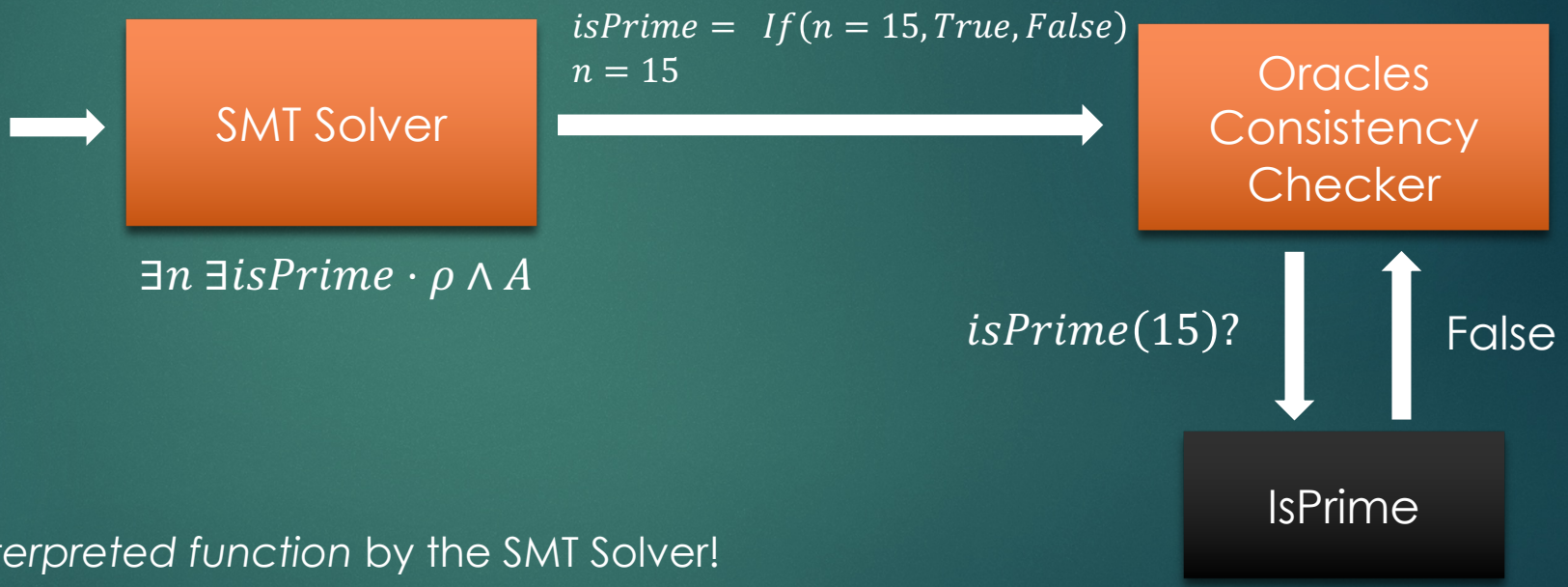
$$isPrime(2) = True$$
$$isPrime(4) = False$$
$$isPrime(5) = True$$
$$\ldots$$

- Generated set of constraints $\boldsymbol{B}$ is empty.

Working of SMTO

$\rho = isPrime(n) \wedge n \geq 15 \wedge n \leq 18$

Oracle Assumptions (**A**):
{}

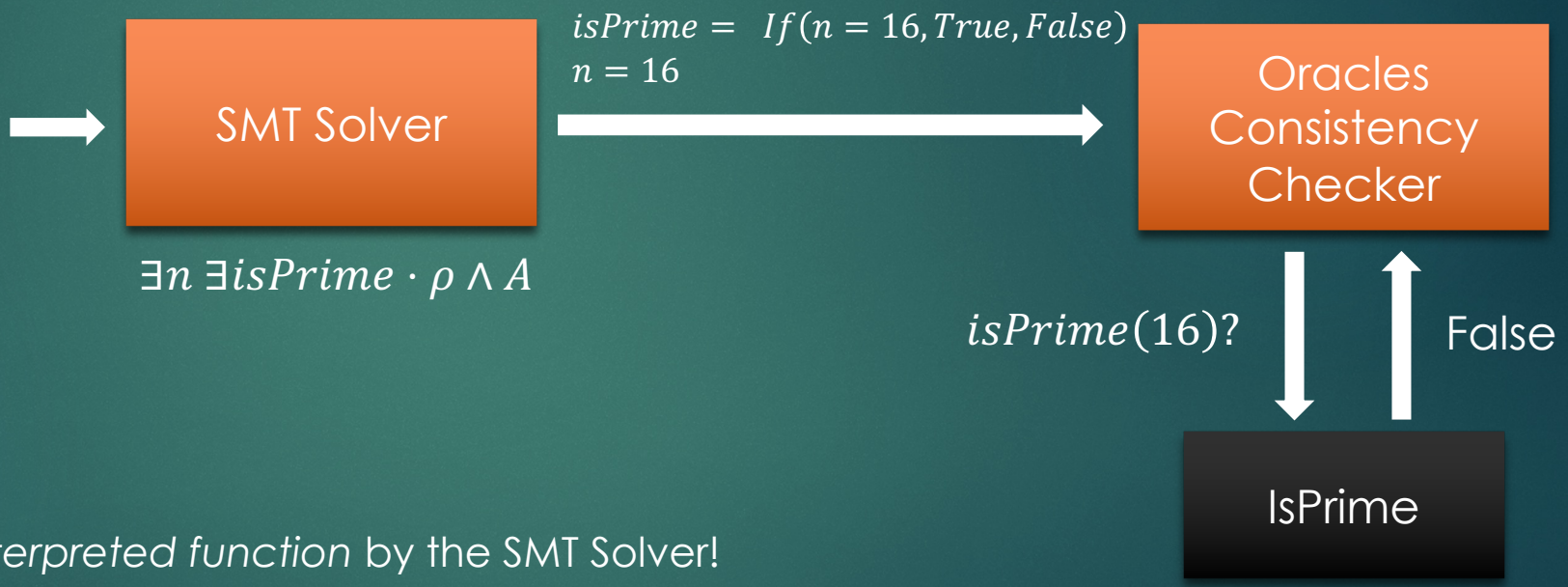


* is$Prime$ treated as a *normal uninterpreted function* by the SMT Solver!

Working of SMTO

$\rho = isPrime(n) \wedge n \geq 15 \wedge n \leq 18$

Oracle Assumptions (**A**):
$\{isPrime(15) = false\}$



* i$sPrime$ treated as a *normal uninterpreted function* by the SMT Solver!

Working of SMTO

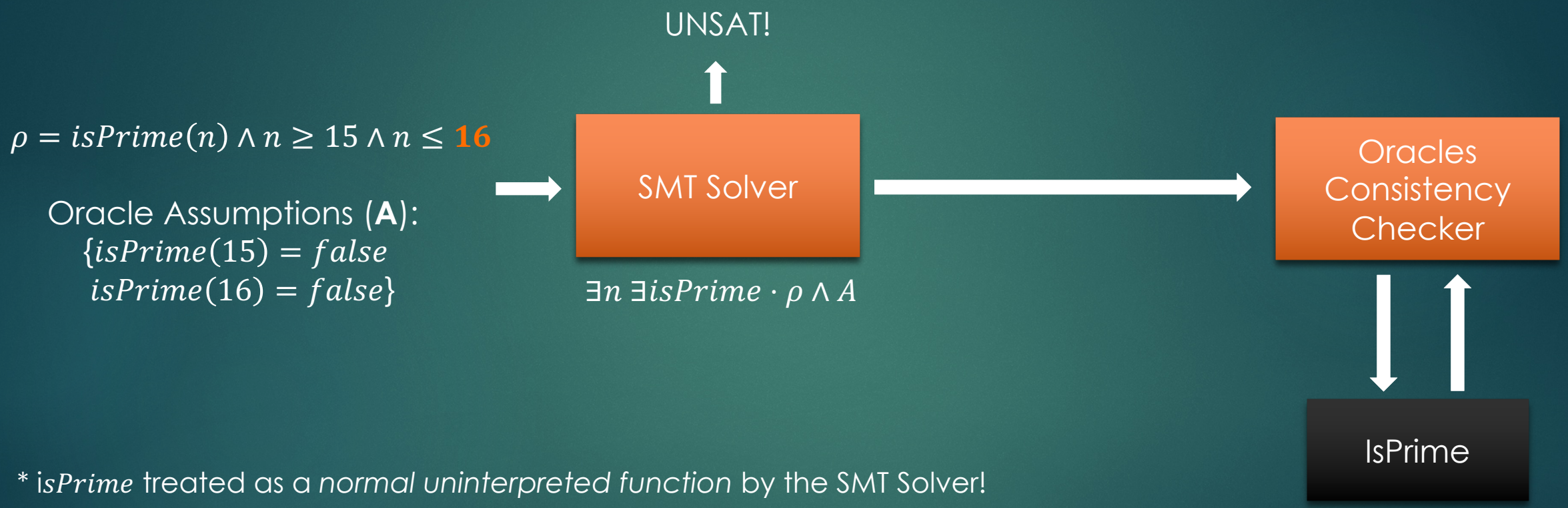
$\rho = isPrime(n) \wedge n \geq 15 \wedge n \leq 18$

Oracle Assumptions (**A**):
$\{isPrime(15) = false$
$isPrime(16) = false\}$

* is$Prime$ treated as a *normal uninterpreted function* by the SMT Solver!

Working of SMTO

* Say our problem was to find a n ≤ 16



* is$Prime$ treated as a *normal uninterpreted function* by the SMT Solver!

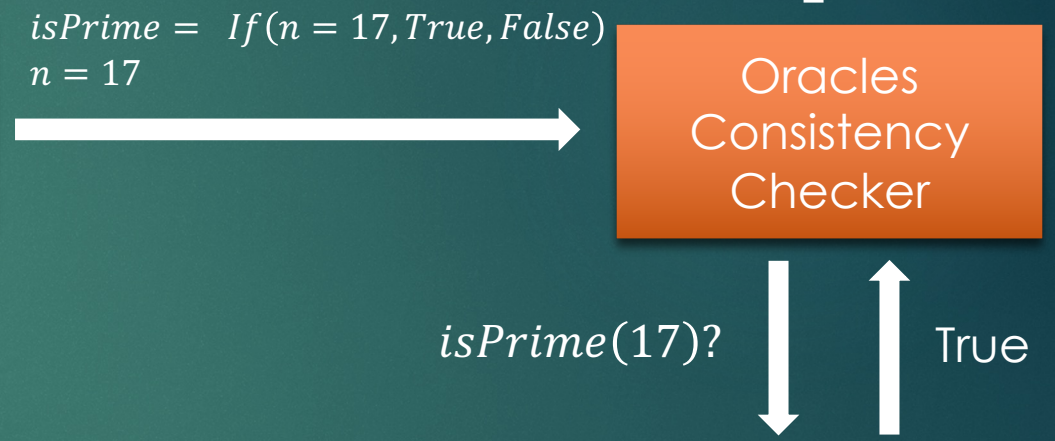
Working of SMTO

$\rho = isPrime(n) \wedge n \geq 15 \wedge n \leq 18$

Oracle Assumptions (**A**):
$\{isPrime(15) = false$
$isPrime(16) = false\}$

$\exists n \; \exists isPrime \cdot \rho \wedge A$

$isPrime = If(n = 17, True, False)$
$n = 17$



Assertions consistent with oracles!

* is$Prime$ treated as a *normal uninterpreted function* by the SMT Solver!

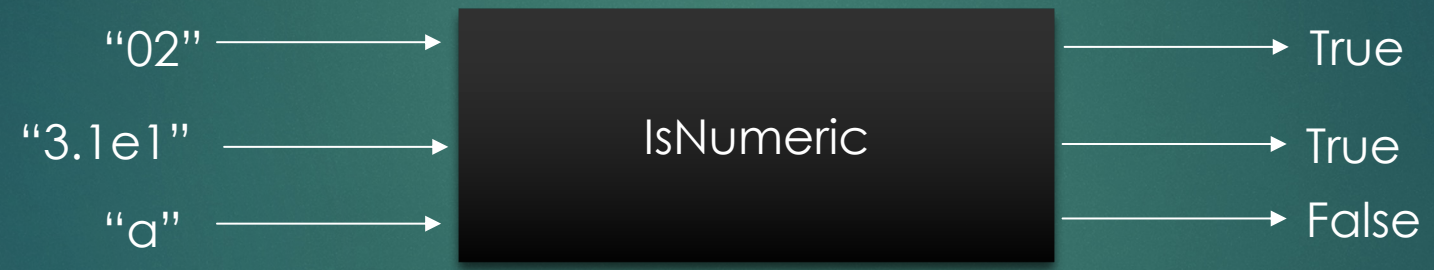
Type-casting Handling

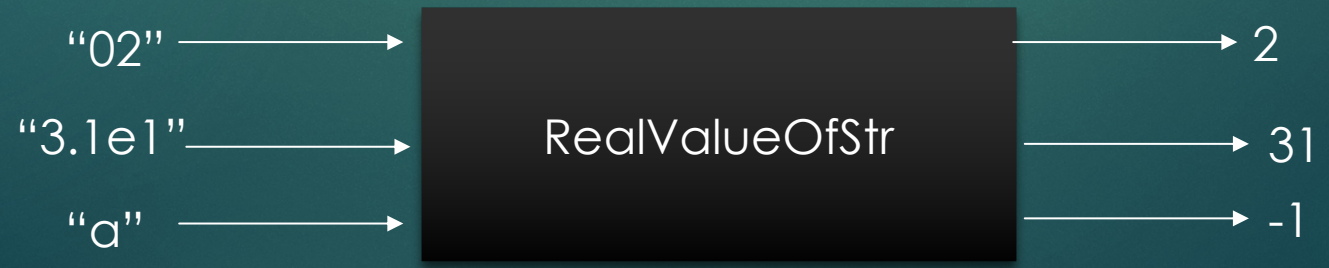
Model semantics of “string s is numerically less than 42”

Now possible through oracles!

$$\phi = IsNumeric(s) \wedge (RealValueOfStr(s) < 42)$$

```
[/Users/sgomber/Desktop >> ./IsStringNumeric "01"
1
[/Users/sgomber/Desktop >> ./IsStringNumeric "1.1e1"
1
[/Users/sgomber/Desktop >> ./IsStringNumeric "1.1f1"
0
```





Things to consider while using SMTO

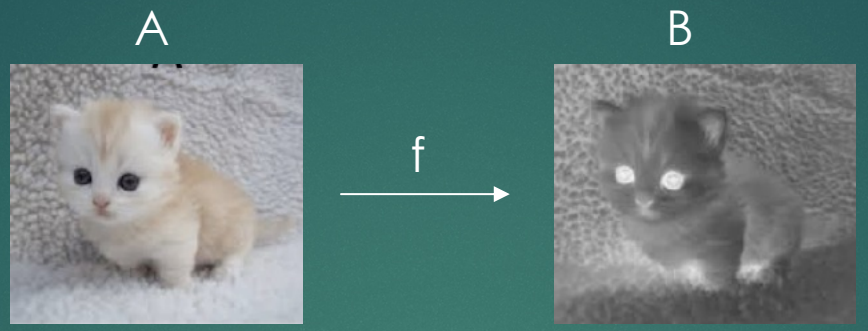
Slow oracles can lead to slower query-solving time

It is assumed that the oracles give correct answers

What if oracle gives isPrime(4) = true

Calling external oracles can be a security threat?

Why Synthesis Modulo Oracles (SyMO)



Can we use Program Synthesis to find f?

Checking if f is correct will need calling image processing libraries

Can now be done in an executable oracle

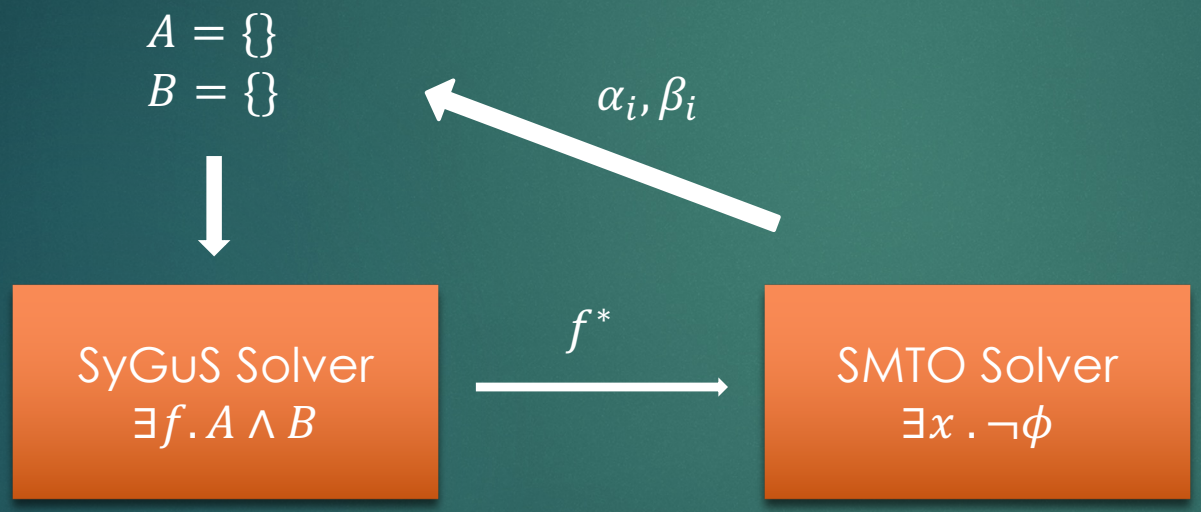
SYMO Formulation

- $\vec{f}$ Ordinary (non-oracle) function symbols to be synthesized
- $\vec{\theta}$ Oracle function symbols (that are modelled using oracles)
- $\vec{I}$ Oracle interfaces modelling the oracle symbols
- A formula $\forall x \cdot \phi$

$\vec{f}^{*}$ *is a solution if the SMTO problem* $\left(\vec{x}, \vec{\theta}, \neg\phi\{\vec{f} \rightarrow \vec{f}^{*}\}\right)$ is UNSAT

SYMO Solving

- Assumptions Generated by the Oracles $\boldsymbol{A}$
- Constraints Generated by the Oracles $\boldsymbol{B}$
- A formula $\forall x \cdot \phi$



Requirements:

- All assumptions of the form $\theta(y) = z$
- Oracles can generate constraints:
  - Not used by SMTO
  - Passed to the synthesis phase

SMTO Solver gives SAT!

f^* is not correct, need to synthesize again

β_i can be something like $f(3,2) = 6$

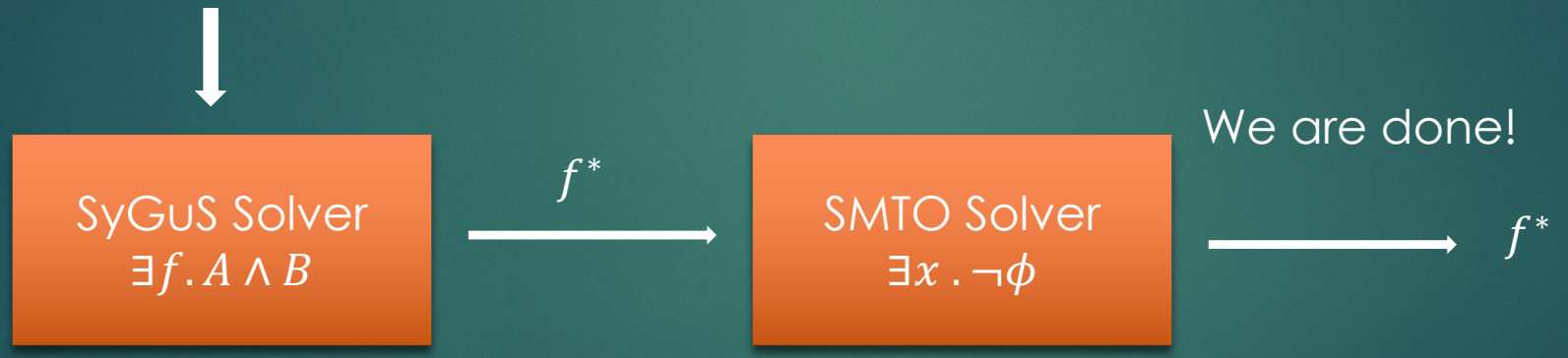
SYMO Solving

- Assumptions Generated by the Oracles $\boldsymbol{A}$
- Constraints Generated by the Oracles $\boldsymbol{B}$
- A formula $\forall x \cdot \phi$

Requirements:

- All assumptions of the form $\theta(y) = z$
- Oracles can generate constraints:
  - Not used by SMTO
  - Passed to the synthesis phase

$A = \{\alpha_i\}$
$B = \{\beta_i\}$



SMTO Solver gives UNSAT!

SYMO's Flexibility

Query Type	Oracle Interface	Example algorithms
	Constraint generating oracles	
Membership	$\mathcal{I}_{mem}(\{y_1, y_2, y\}, z_b, \top, z_b \Leftrightarrow f(y_1, y_2) \approx y)$	Angluin's L^* [7]
Input-Output	$\mathcal{I}_{io}(\{y_1, y_2\}, z, \top, z \approx f(y_1, y_2))$	Classic PBE
Negative witness	$\mathcal{I}_{neg}(\emptyset, \{z_1, z_2, z\}, \top, f(z_1, z_2) \not\approx z)$	ICE-learning [19]
Positive witness	$\mathcal{I}_{pos}(\emptyset, \{z_1, z_2, z\}, \top, f(z_1, z_2) \approx z)$	ICE-learning [19]
Implication	$\mathcal{I}_{imp}(f^*, \{z_1, z_2, z_1', z_2'\}, \top, f(z_1, z_2) \Rightarrow f(z_1', z_2')$	ICE-learning [19]
Counterexample	$\mathcal{I}_{cex}(f^*, \vec{z}, \top, \phi\{\vec{x} \rightarrow \vec{z}\})$	Synthesis with validators [23]
Distinguishing-input	$\mathcal{I}_{di}(f^*, \{z_1, z_2, z\}, \top, f(z_1, z_2) \approx z)$	Synthesis with distinguishing inputs [20]
	Constraint and assumption generating oracles	
Correctness	$\mathcal{J}_{corr}(f^*, z_b, \theta(f^*) \approx z_b, \top)$	ICE-learning [19]
Correctness with cex	$\mathcal{J}_{ccex}(f^*, z_b, \vec{z}, \theta(f^*) \approx z_b, \phi\{\vec{x} \rightarrow \vec{z}\})$	classic CEGIS [29]

Table 1: Common oracle interfaces, illustrated for synthesizing a single function which takes two inputs $f(x_1, x_2)$. y indicates query variables, except where they are the candidate function, in which case we use f^*, and z indicates response variables, where z_b is a Boolean.

Thanks!