Computer-Aided Reasoning for Software

CSE507

Program Synthesis

courses.cs.washington.edu/courses/cse507/17wi/

Emina Torlak
emina@cs.washington.edu

Today

topics

Last lecture

- Solvers as angelic runtime oracle

Today

- Program synthesis: computers programming computers

Reminders

- HW3 is due.
- Please fill out the course evaluation (March 5-12)!

Computers programming computers?

“Information technology has been praised as a labor saver and cursed as a destroyer of obsolete jobs. But the entire edifice of modern computing rests on a fundamental irony: **the software that makes it all possible is, in a very real sense, handmade.** Every miraculous thing computers can accomplish begins with a human programmer entering lines of code by hand, character by character.”

Interview with Moshe Vardi

Computers programming computers?

“Information technology has been praised as a labor saver and cursed as a destroyer of obsolete jobs. But the entire edifice of modern computing rests on a fundamental irony: **the software that makes it all possible is, in a very real sense, handmade.** Every miraculous thing computers can accomplish begins with a human programmer entering lines of code by hand, character by character.”

Interview with Moshe Vardi

> Program synthesis aims to automate (tedious parts of) programming.

The program synthesis problem

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

Find a program P that satisfies the specification φ on all inputs.

The program synthesis problem

φ may be a formula, a reference implementation, input/output pairs, traces, demonstrations, etc.

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

Find a program P that satisfies the specification φ on all inputs.

The program synthesis problem

φ may be a formula, a reference implementation, input/output pairs, traces, demonstrations, etc.

Synthesis improves

- Productivity (when writing φ is easier than writing P).
- Correctness (when verifying φ is easier than verifying P).

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

Find a program P that satisfies the specification φ on all inputs.

Two kinds of program synthesis

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

Deductive (classic) synthesis

Inductive (syntax-guided) synthesis

Two kinds of program synthesis

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

Deductive (classic) synthesis

Derive the program P from the constructive proof of the theorem $\forall\, x.\ \exists\, y.\ \varphi(y, x)$.

Inductive (syntax-guided) synthesis

Two kinds of program synthesis

$\exists$ P. $\forall$ x. φ(x, P(x))

Deductive (classic) synthesis

Derive the program P from the constructive proof of the theorem $\forall$ x. $\exists$ y. φ(y, x).

Inductive (syntax-guided) synthesis

Discover the program P by searching a restricted space of candidate programs for one that satisfies φ on all inputs.

Two kinds of program synthesis

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

Synthesis as a problem in deductive theorem proving.

Synthesis as a search problem.

Two kinds of program synthesis

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

SPIRAL

Synthesis as a problem in deductive theorem proving.

FlashFill

Synthesis as a search problem.

Deductive synthesis with axioms and E-graphs

Denali Superoptimizer
[Joshi, Nelson, Randall, PLDI'02]

Deductive synthesis with axioms and E-graphs

Specification φ, given as a reference implementation.

`reg6 * 4 + 1`

Denali Superoptimizer [Joshi, Nelson, Randall, PLDI'02]

Deductive synthesis with axioms and E-graphs

Specification φ, given as a reference implementation.

Optimal (lowest cost) program P that is equivalent to φ on all inputs (values of reg6).

`reg6 * 4 + 1` → Denali Superoptimizer [Joshi, Nelson, Randall, PLDI’02] → `s4addl(reg6, 1)`

Deductive synthesis with axioms and E-graphs

Specification φ, given as a reference implementation.

`reg6 * 4 + 1`

Denali Superoptimizer [Joshi, Nelson, Randall, PLDI'02]

`s4addl(reg6, 1)`

Optimal (lowest cost) program P that is equivalent to φ on all inputs (values of reg6).

$\forall$ k, n. 2^n = 2**n

$\forall$ k, n. k*2^n = k << n

$\forall$ k, n. k*4 + n = **`s4addl`**(k, n)

…

Two kinds of axioms:

- Instruction semantics.
- Algebraic properties of functions and relations used for specifying instruction semantics.

Deductive synthesis with axioms and E-graphs

Specification φ, given as a reference implementation.

1. Construct an E-graph.
2. Use a SAT solver to search the E-graph for a K-cycle program.

Optimal (lowest cost) program P that is equivalent to φ on all inputs (values of reg6).

`reg6 * 4 + 1`

Denali Superoptimizer [Joshi, Nelson, Randall, PLDI'02]

`s4addl(reg6, 1)`

$\forall$ k, n. 2^n = 2**n

$\forall$ k, n. k*2^n = k << n

$\forall$ k, n. k*4 + n = **`s4addl`**(k, n)

…

Two kinds of axioms:

- Instruction semantics.
- Algebraic properties of functions and relations used for specifying instruction semantics.

Denali by example

`reg6 * 4 + 1`

$\forall k, n.\ 2^n = 2{*}{*}n$

$\forall k, n.\ k{*}2^n = k \ll n$

$\forall k, n.\ k{*}4 + n = \mathbf{s4addl}(k, n)$

…

E-graph matching

SAT

`s4addl(reg6, 1)`

Denali by example

`reg6 * 4 + 1`

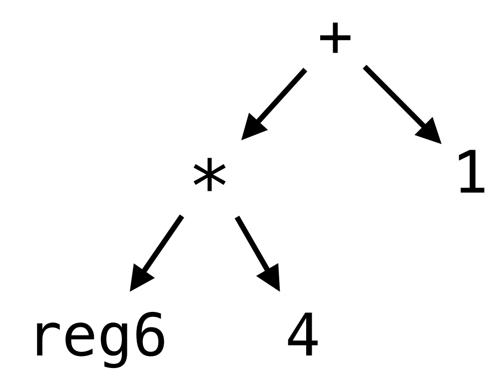
$\forall$ k, n. 2^n = 2**n

$\forall$ k, n. k*2^n = k **<<** n

$\forall$ k, n. k*4 + n = **`s4addl`**(k, n)

…

E-graph matching



SAT

`s``4addl(reg6, 1)`

Denali by example

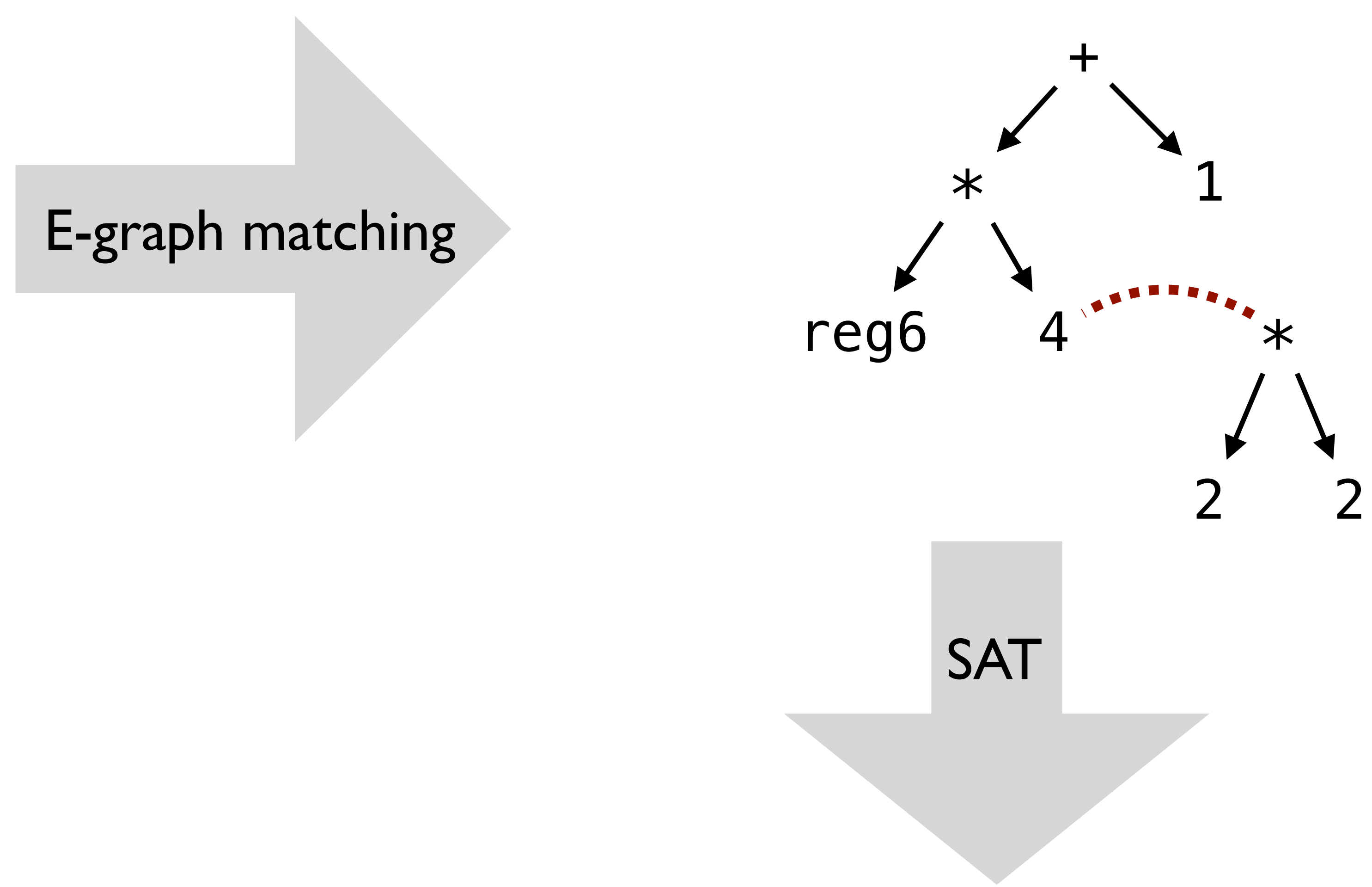
```
reg6 * 4 + 1
```

∀ k, n. 2^n = 2**n

∀ k, n. k*2^n = k **<<** n

∀ k, n. k*4 + n = **`s4addl`**(k, n)

…



`s``4addl(reg6, 1)`

Denali by example

```
reg6 * 4 + 1
```

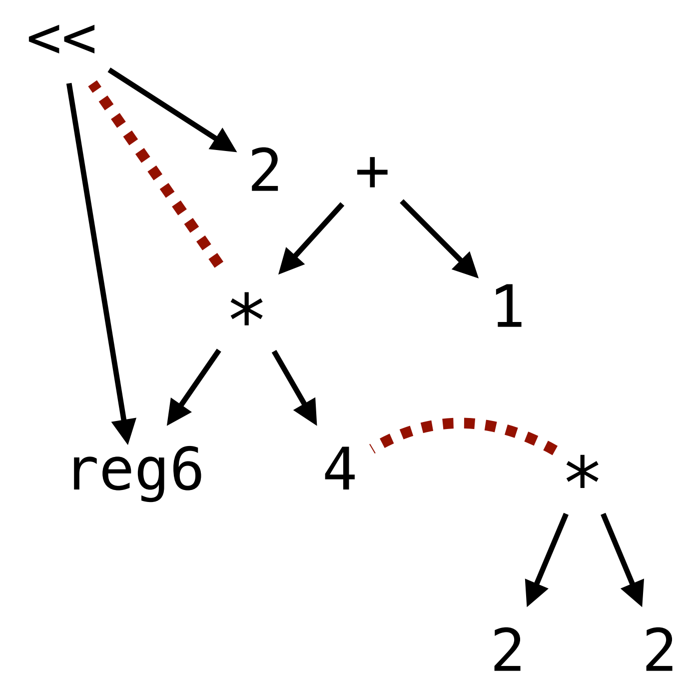
$\forall$ k, n. 2^n = 2**n

$\forall$ k, n. k*2^n = k **<<** n

$\forall$ k, n. k*4 + n = **`s4addl`**(k, n)

…

E-graph matching



SAT

`s``4addl(reg6, 1)`

Denali by example

`reg6 * 4 + 1`

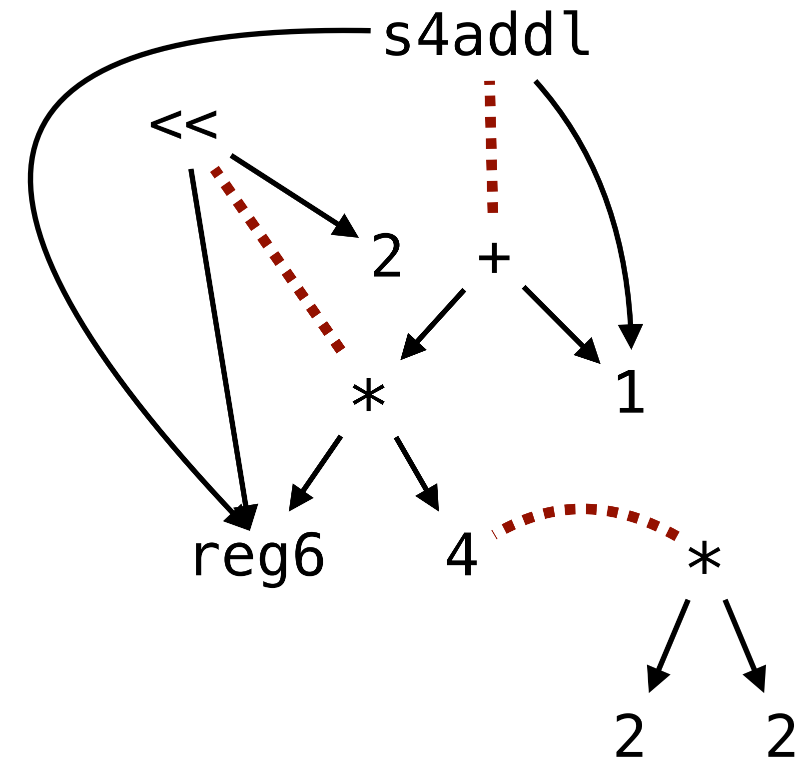
$\forall$ k, n. 2^n = 2**n

$\forall$ k, n. k*2^n = k **<<** n

$\forall$ k, n. k*4 + n = **`s4addl`**(k, n)

…

E-graph matching



SAT

`s4addl``(reg6, 1)`

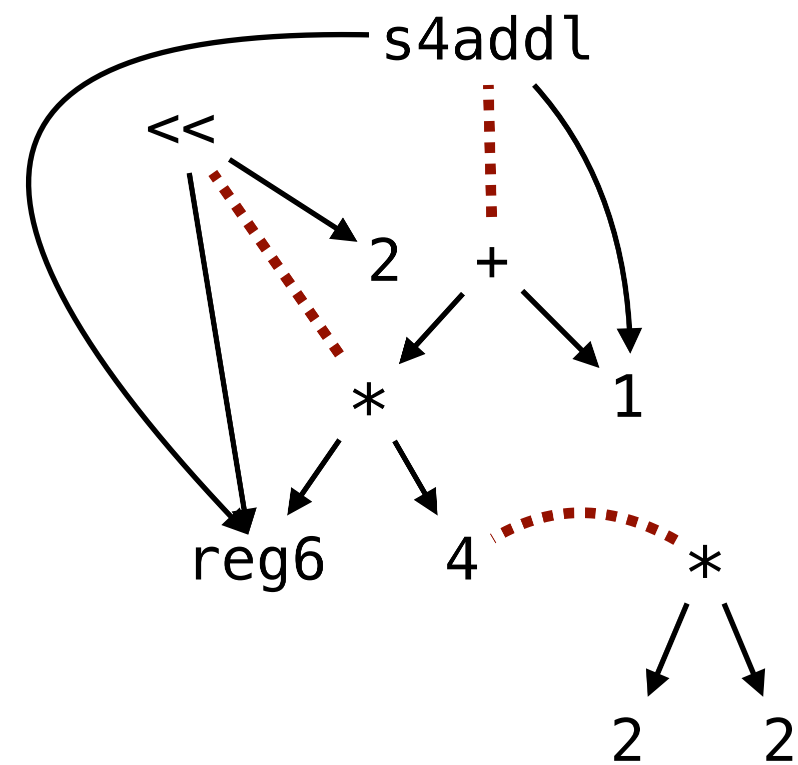
Deductive synthesis versus compilation

Deductive synthesizer

- Non-deterministic.
- Searches *all correct rewrites* for one that is optimal.

Compiler

- Deterministic.
- Lowers a source program into a target program using a *fixed sequence of rewrite steps*.




```
reg6 * 4 + 1
reg6 << 2 + 1
```

Deductive synthesis versus inductive synthesis

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

Deductive synthesis

- Efficient and provably correct: thanks to the semantics-preserving rules, only correct programs are explored.
- Requires *sufficient axiomatization* of the domain.
- Requires *complete* specifications to seed the derivation.

Deductive synthesis versus inductive synthesis

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

Deductive synthesis

- Efficient and provably correct: thanks to the semantics-preserving rules, only correct programs are explored.
- Requires *sufficient axiomatization* of the domain.
- Requires *complete* specifications to seed the derivation.

Inductive synthesis

- Works with *multi-modal and partial* specifications.
- Requires *no axioms*.
- But often at the cost of *lower efficiency* and *weaker (bounded) guarantees* on the correctness/optimality of synthesized code.

Inductive syntax-guided synthesis

CEGIS:
Counterexample-Guided
Inductive Synthesis
[**Solar-Lezama et al, ASPLOS'06**]

Inductive syntax-guided synthesis

A partial or multimodal specification φ of the desired program (e.g., assertions, i/o pairs).

`reg6 * 4 + 1`

CEGIS:
Counterexample-Guided Inductive Synthesis
[**Solar-Lezama et al, ASPLOS'06**]

Inductive syntax-guided synthesis

A partial or multimodal specification φ of the desired program (e.g., assertions, i/o pairs).

```
reg6 * 4 + 1
```

CEGIS: Counterexample-Guided Inductive Synthesis [Solar-Lezama et al, ASPLOS'06]

```
expr :=
 const | reg6 |
 s4addl(expr, expr) |
 …
```

A syntactic *sketch* (e.g., a grammar) describing the shape of the desired program P.

This defines the space of candidate programs to search. Can be fine-tuned for better performance.

Inductive syntax-guided synthesis

A partial or multimodal specification φ of the desired program (e.g., assertions, i/o pairs).

```
reg6 * 4 + 1
```

CEGIS: Counterexample-Guided Inductive Synthesis [Solar-Lezama et al, ASPLOS'06]

A program P from the given space of candidates that satisfies φ on all (usually bounded) inputs.

```
s4addl(reg6, 1)
```

```
expr :=
 const | reg6 |
 s4addl(expr, expr) |
 …
```

A syntactic *sketch* (e.g., a grammar) describing the shape of the desired program P.

This defines the space of candidate programs to search. Can be fine-tuned for better performance.

Inductive syntax-guided synthesis

A partial or multimodal specification φ of the desired program (e.g., assertions, i/o pairs).

Guess a program that works on a finite set of inputs, verify it, and learn from bad guesses.

A program P from the given space of candidates that satisfies φ on all (usually bounded) inputs.

`reg6 * 4 + 1`

CEGIS: Counterexample-Guided Inductive Synthesis [Solar-Lezama et al, ASPLOS'06]

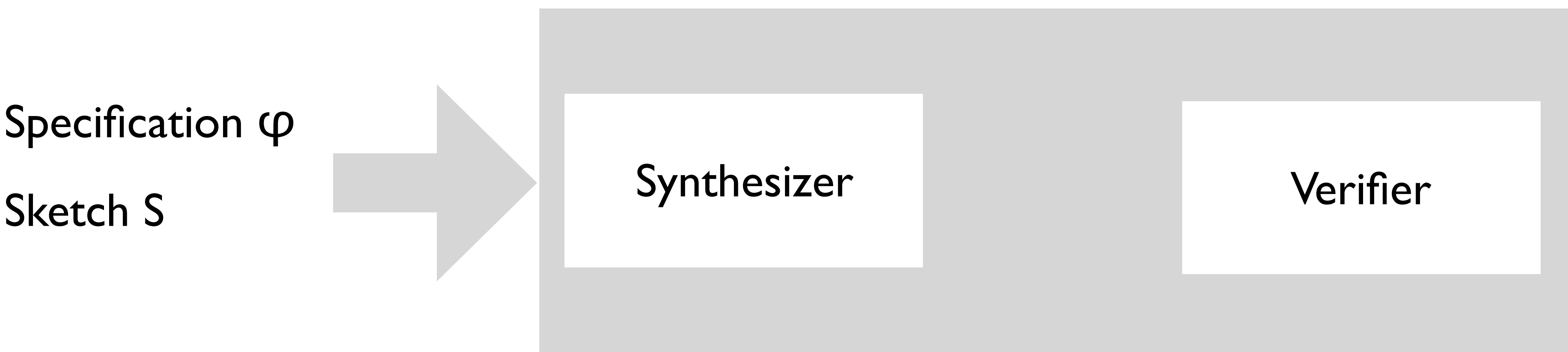
`s4addl(reg6, 1)`

```
expr :=
 const | reg6 |
 s4addl(expr, expr) |
 …
```

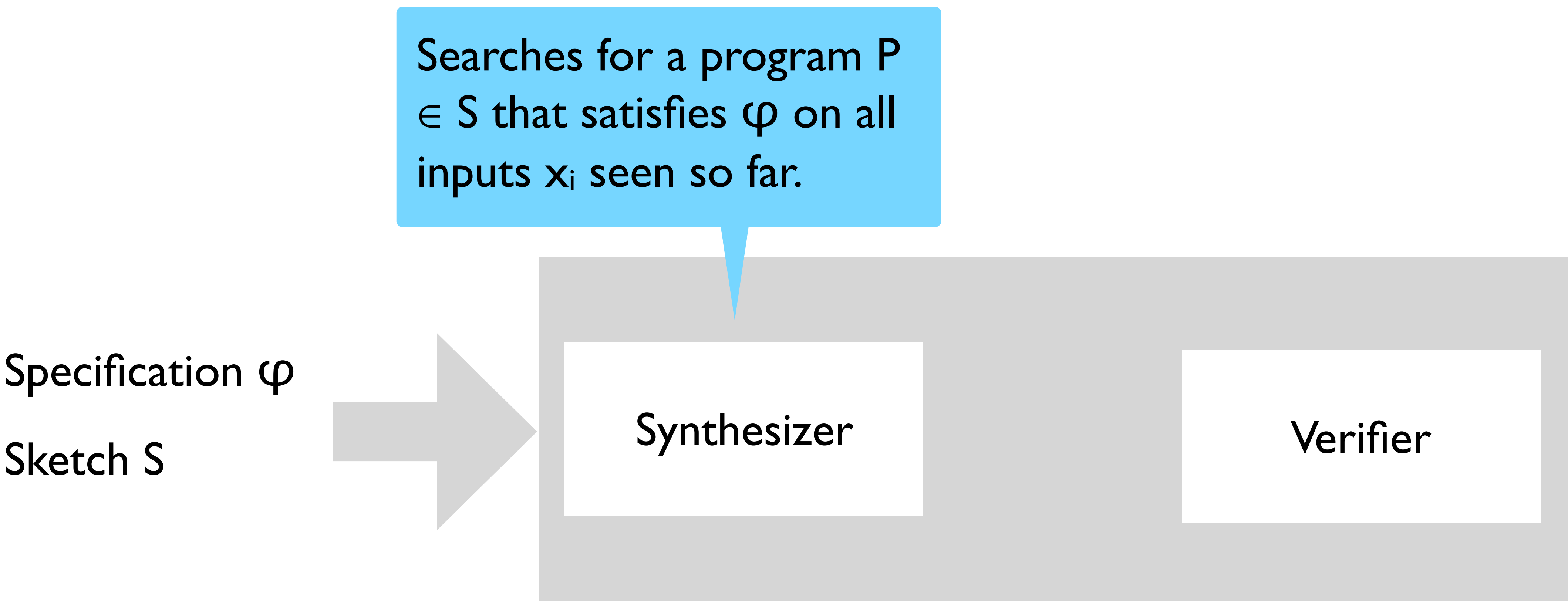
A syntactic *sketch* (e.g., a grammar) describing the shape of the desired program P.

This defines the space of candidate programs to search. Can be fine-tuned for better performance.

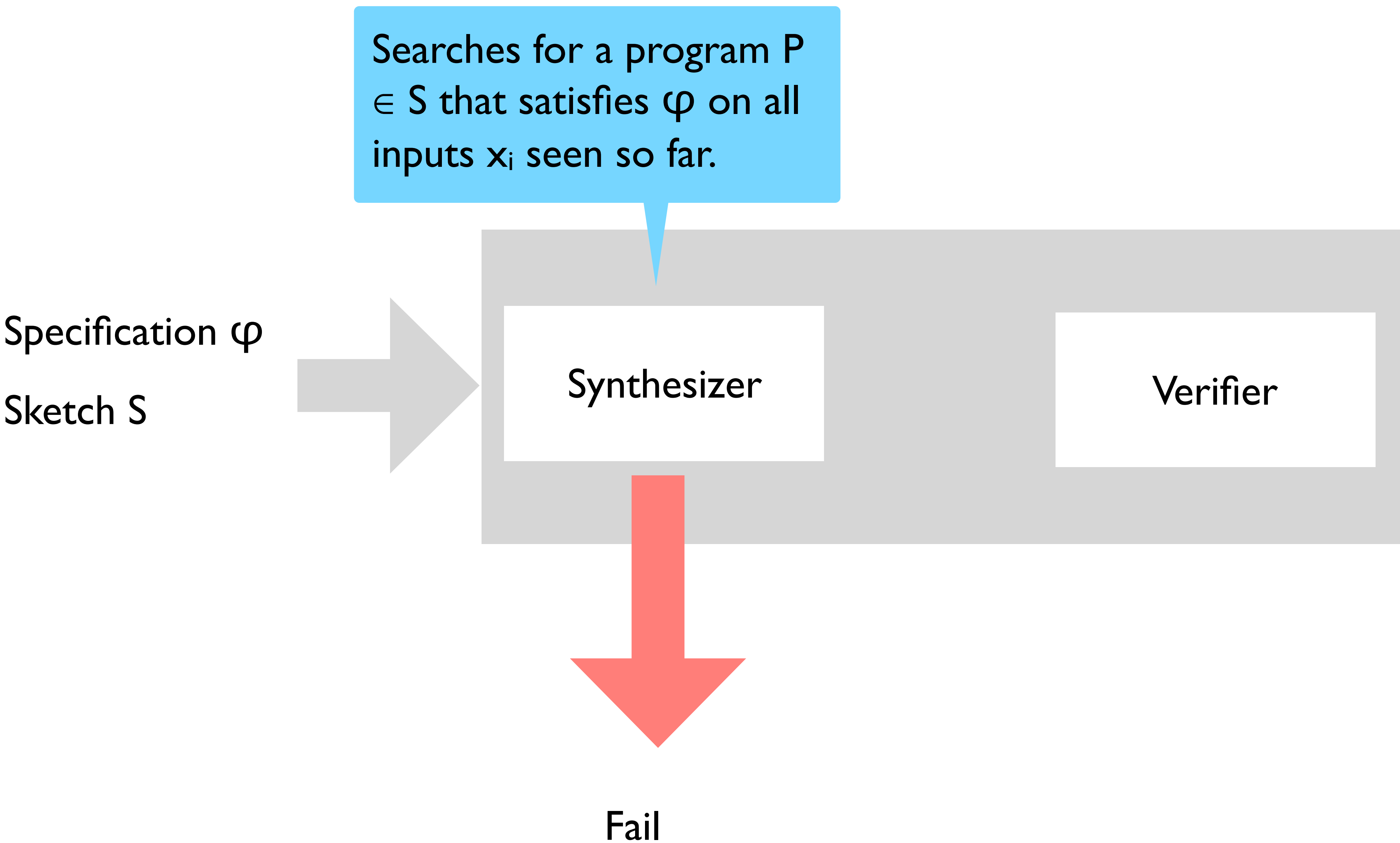
Overview of CEGIS



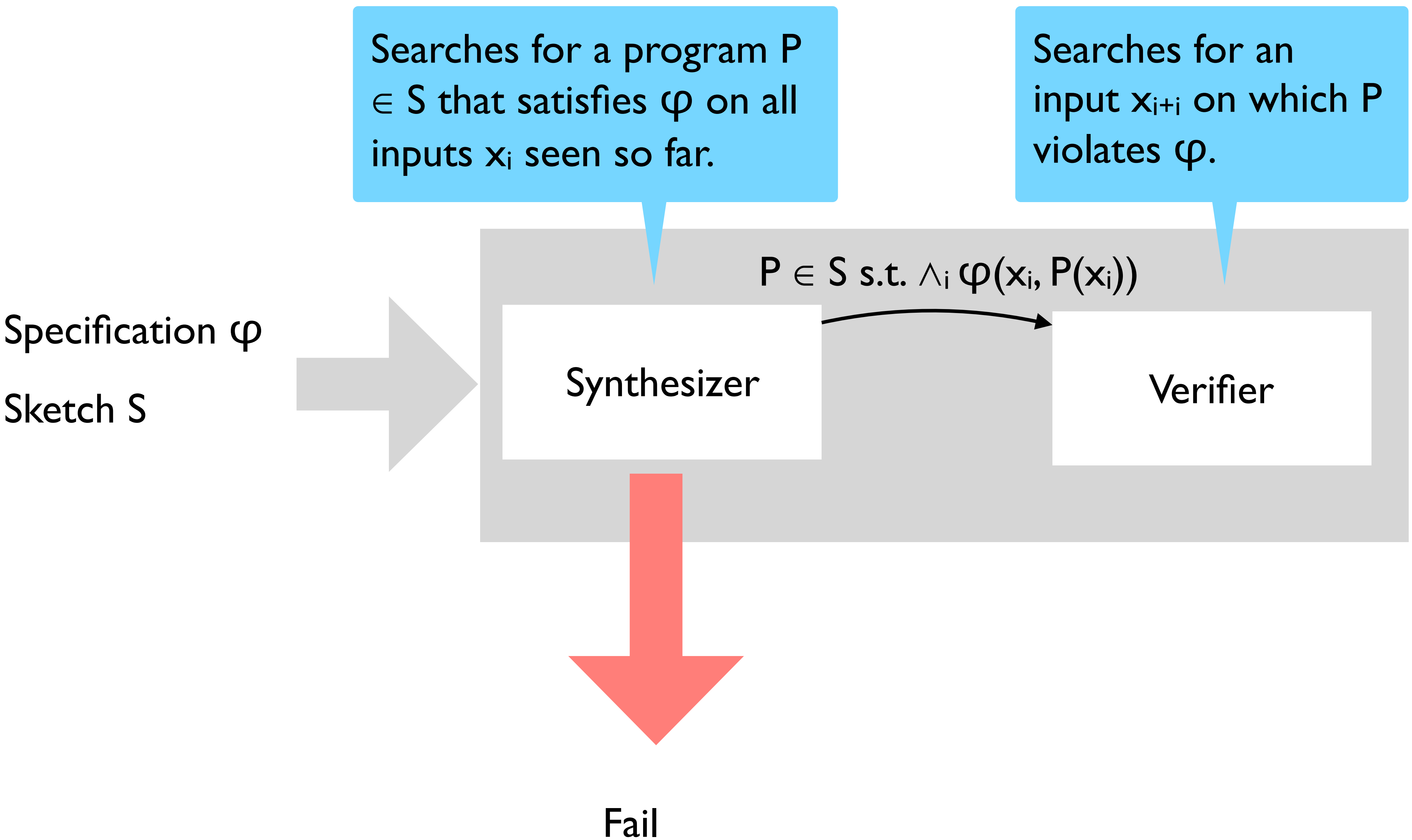
Overview of CEGIS



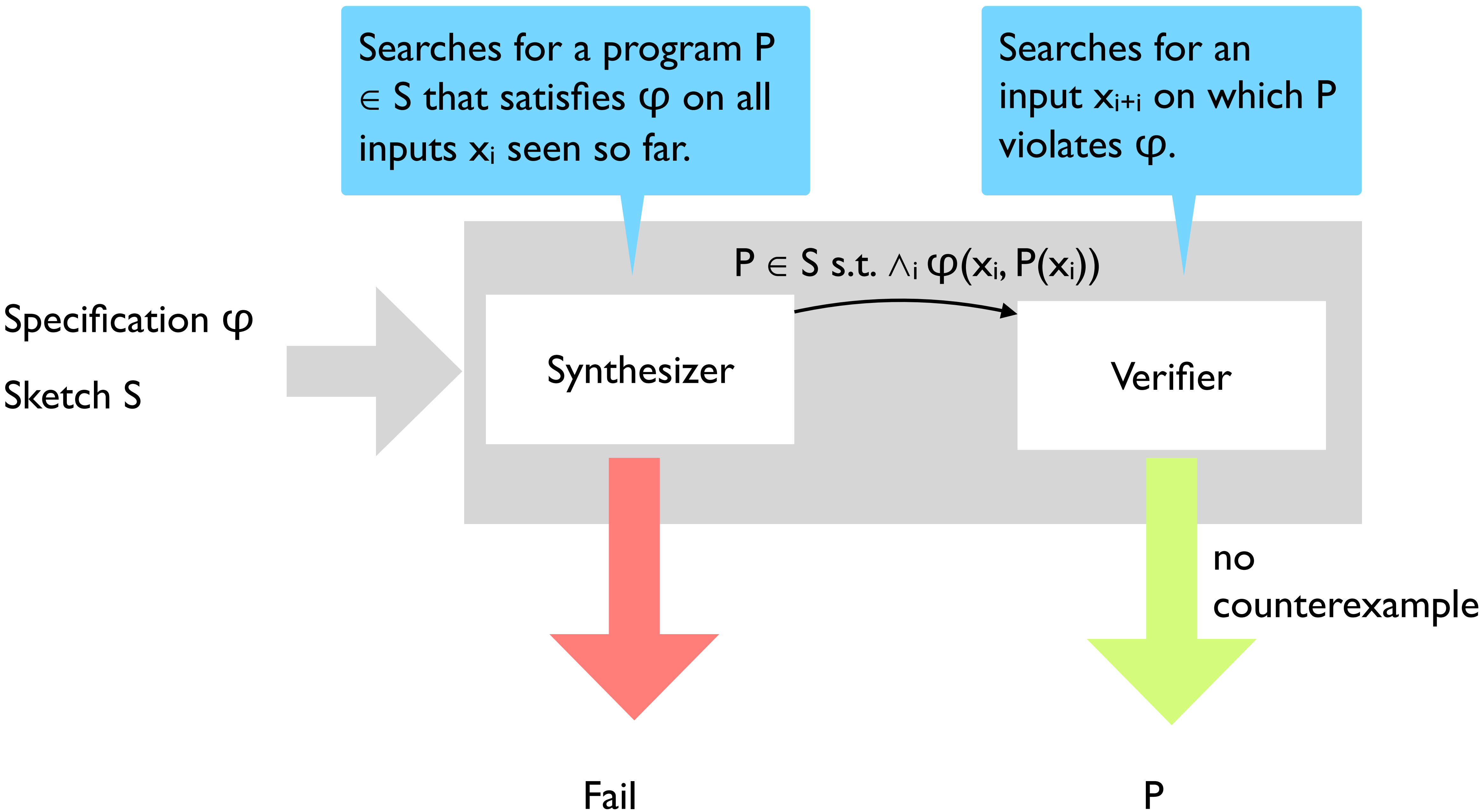
Overview of CEGIS



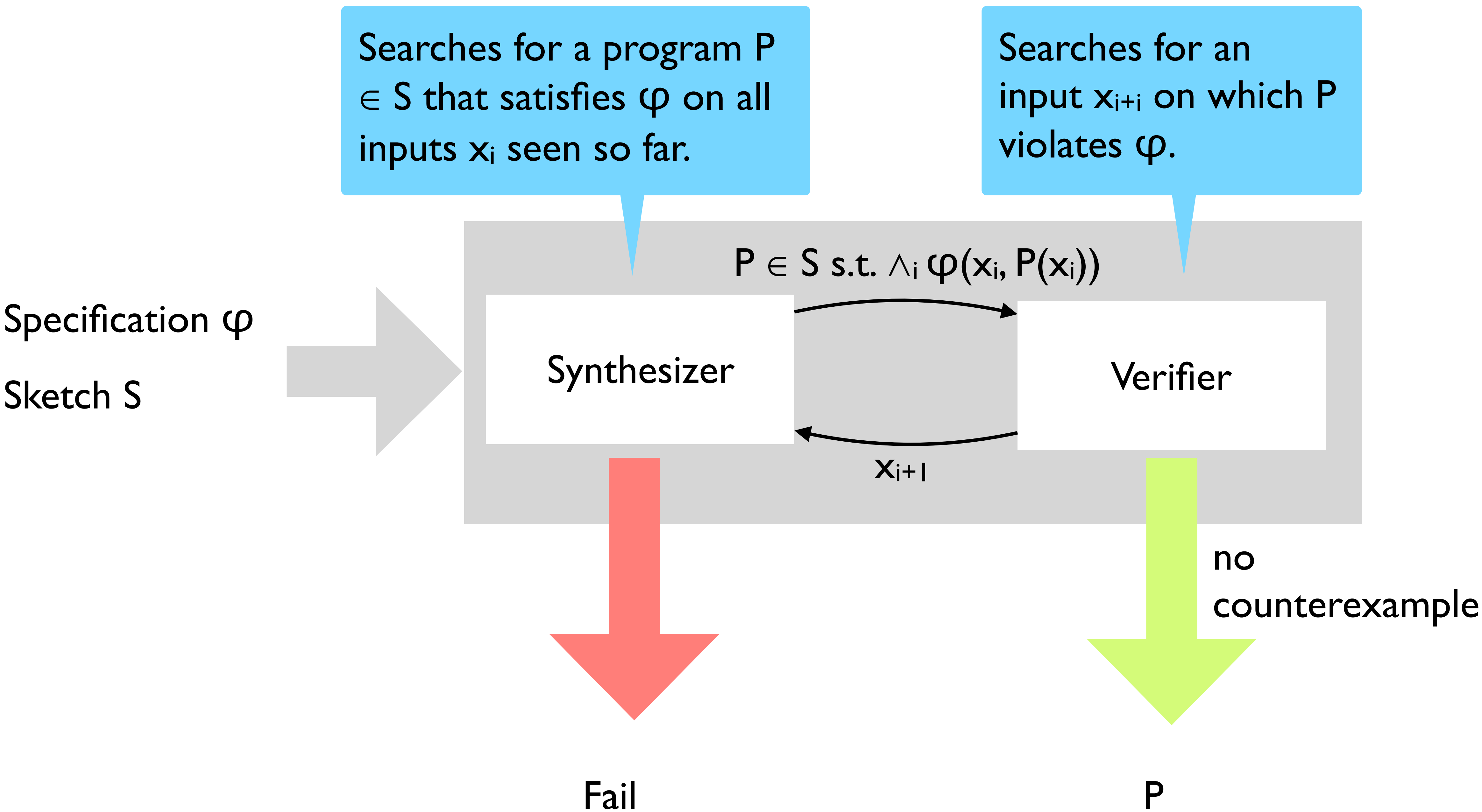
Overview of CEGIS



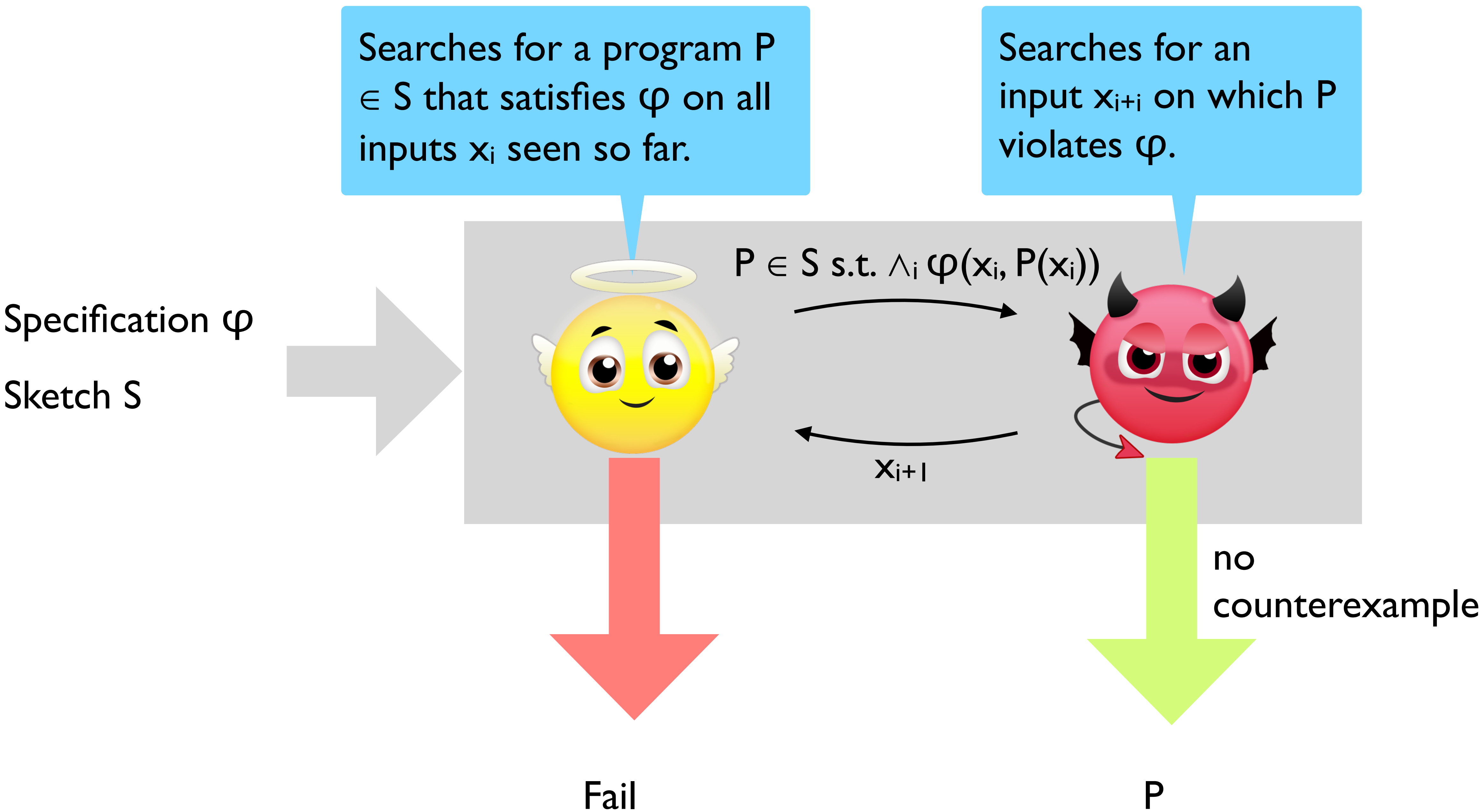
Overview of CEGIS



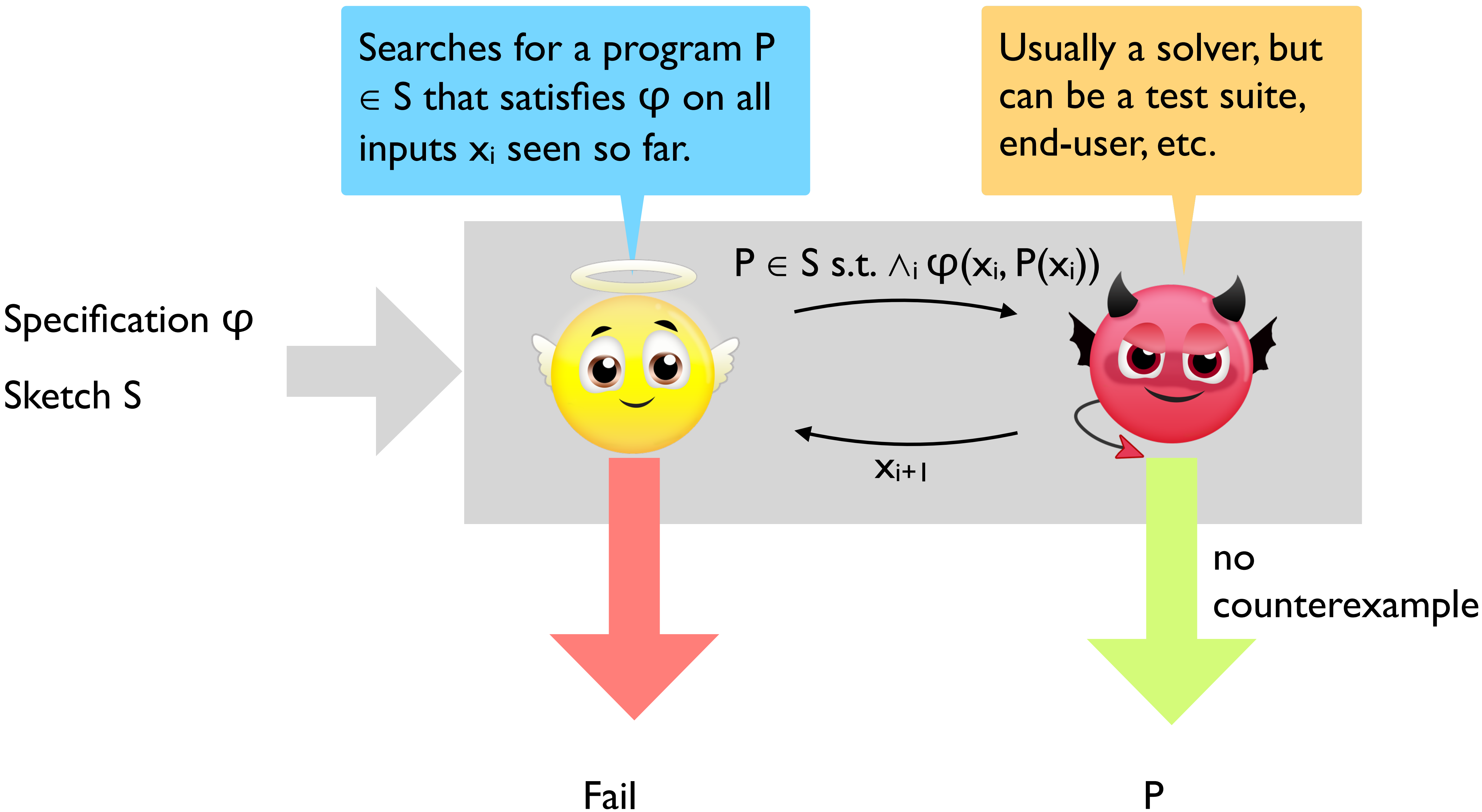
Overview of CEGIS



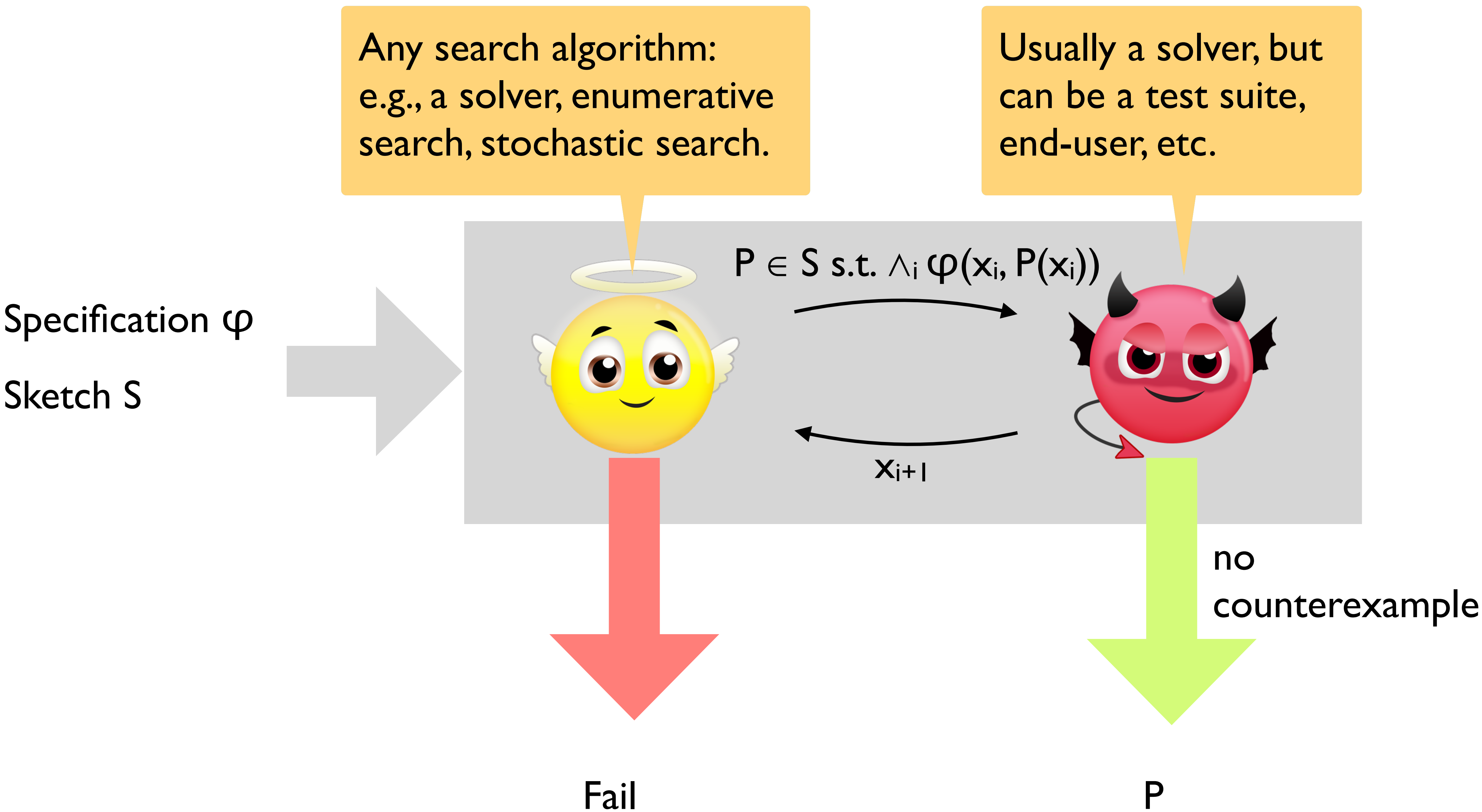
Overview of CEGIS



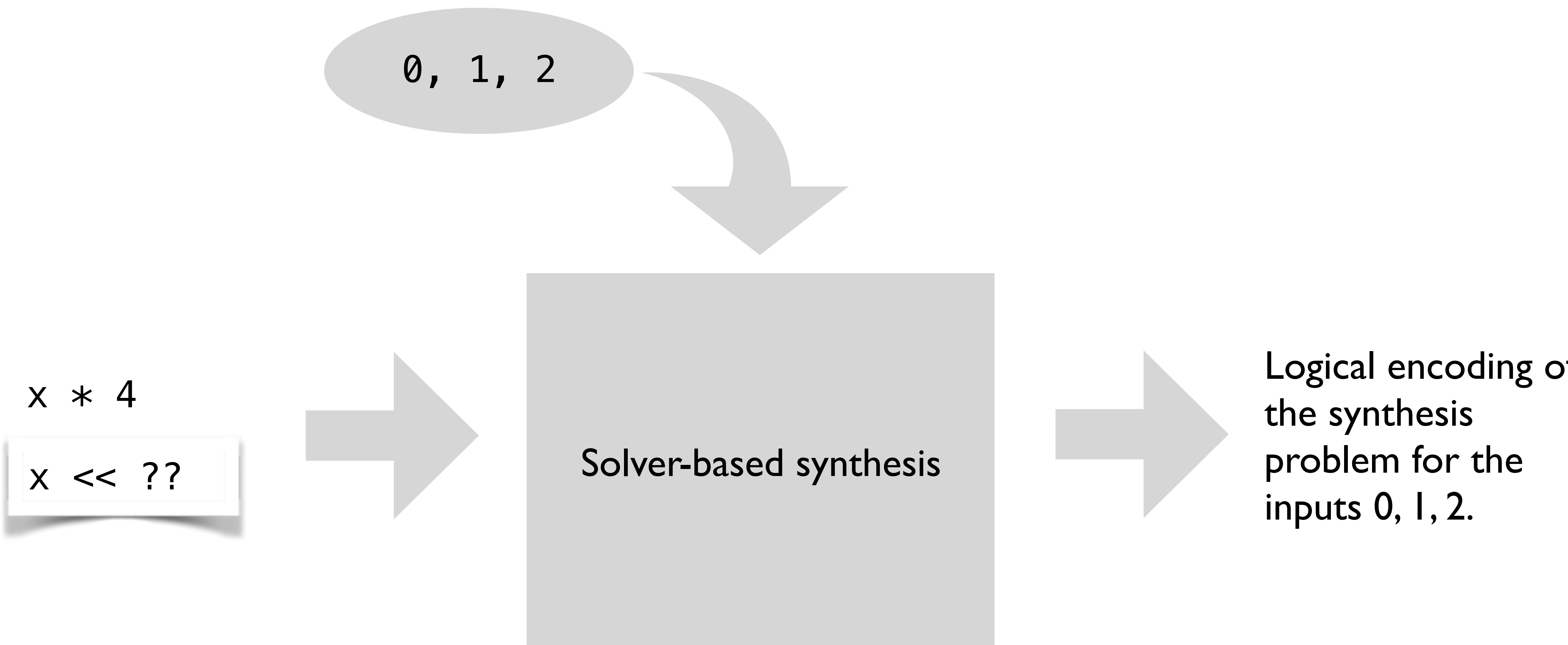
Overview of CEGIS



Overview of CEGIS

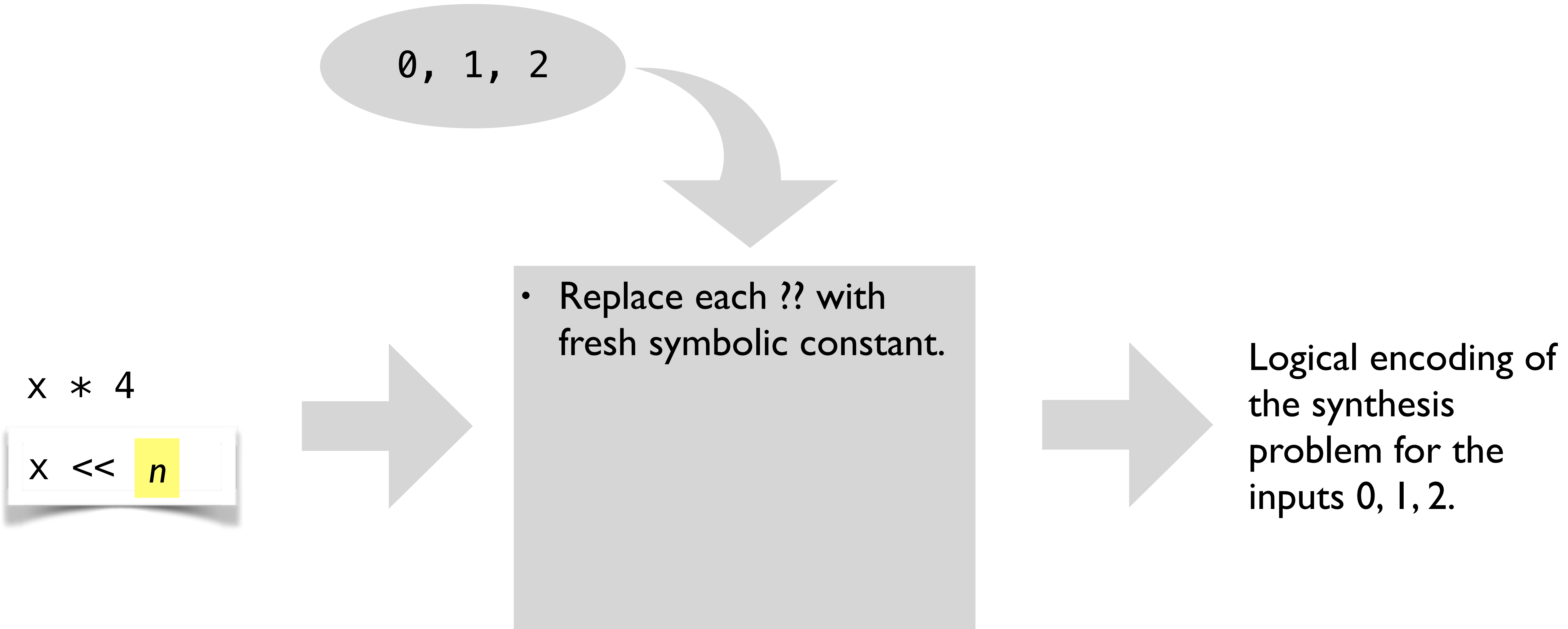


Synthesizing programs with a solver



[Solar-Lezama et al, ASPLOS'06]

Synthesizing programs with a solver



[Solar-Lezama et al, ASPLOS'06]

Synthesizing programs with a solver

`0, 1, 2`

`x * 4`

`x << n`

- Replace each ?? with fresh symbolic constant.
- Translate the resulting problem to constraints w.r.t. the current inputs.

$(0 << n = 0) \wedge$

$(1 << n = 4) \wedge$

$(2 << n = 8)$

[Solar-Lezama et al, ASPLOS'06]

Synthesizing programs with a solver

`0, 1, 2`

`x * 4`

`x << n`

- Replace each ?? with fresh symbolic constant.
- Translate the resulting problem to constraints w.r.t. the current inputs.
- If SAT, convert the model to a program P.

$(0 \ll n = 0) \wedge$

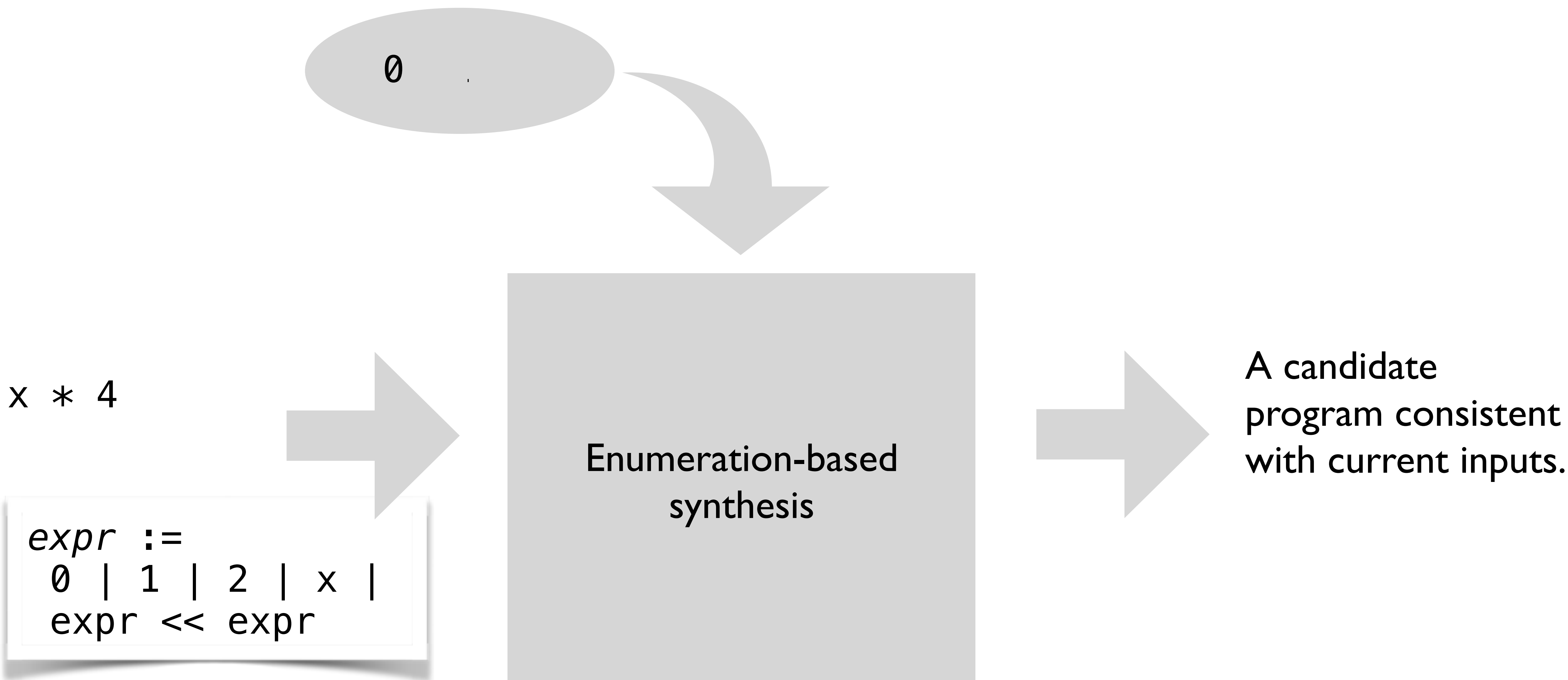
$(1 \ll n = 4) \wedge$

$(2 \ll n = 8)$

`x << 2`

[Solar-Lezama et al, ASPLOS'06]

Synthesizing programs with enumerative search



[Udupa et al, PLDI'13]

Synthesizing programs with enumerative search

0

x * 4

```
expr :=
 0 | 1 | 2 | x |
 expr << expr
```

- Iteratively construct all programs of size *K* until one is consistent with the current inputs.
- If two programs produce the same output on all current inputs, keep just one of the two.

A candidate program consistent with current inputs.

[Udupa et al, PLDI'13]

Synthesizing programs with enumerative search

`0`

`x * 4`

```
expr :=
 0 | 1 | 2 | x |
 expr << expr
```

- Iteratively construct all programs of size *K* until one is consistent with the current inputs.
- If two programs produce the same output on all current inputs, keep just one of the two.

K=*1*: 0

[Udupa et al, PLDI'13]

Synthesizing programs with enumerative search

```
0, 1
```

```
x * 4
```

```
expr :=
 0 | 1 | 2 | x |
 expr << expr
```

- Iteratively construct all programs of size *K* until one is consistent with the current inputs.
- If two programs produce the same output on all current inputs, keep just one of the two.

[Udupa et al, PLDI'13]

Synthesizing programs with enumerative search

`0, 1`

`x * 4`

```
expr :=
 0 | 1 | 2 | x |
 expr << expr
```

- Iteratively construct all programs of size *K* until one is consistent with the current inputs.
- If two programs produce the same output on all current inputs, keep just one of the two.

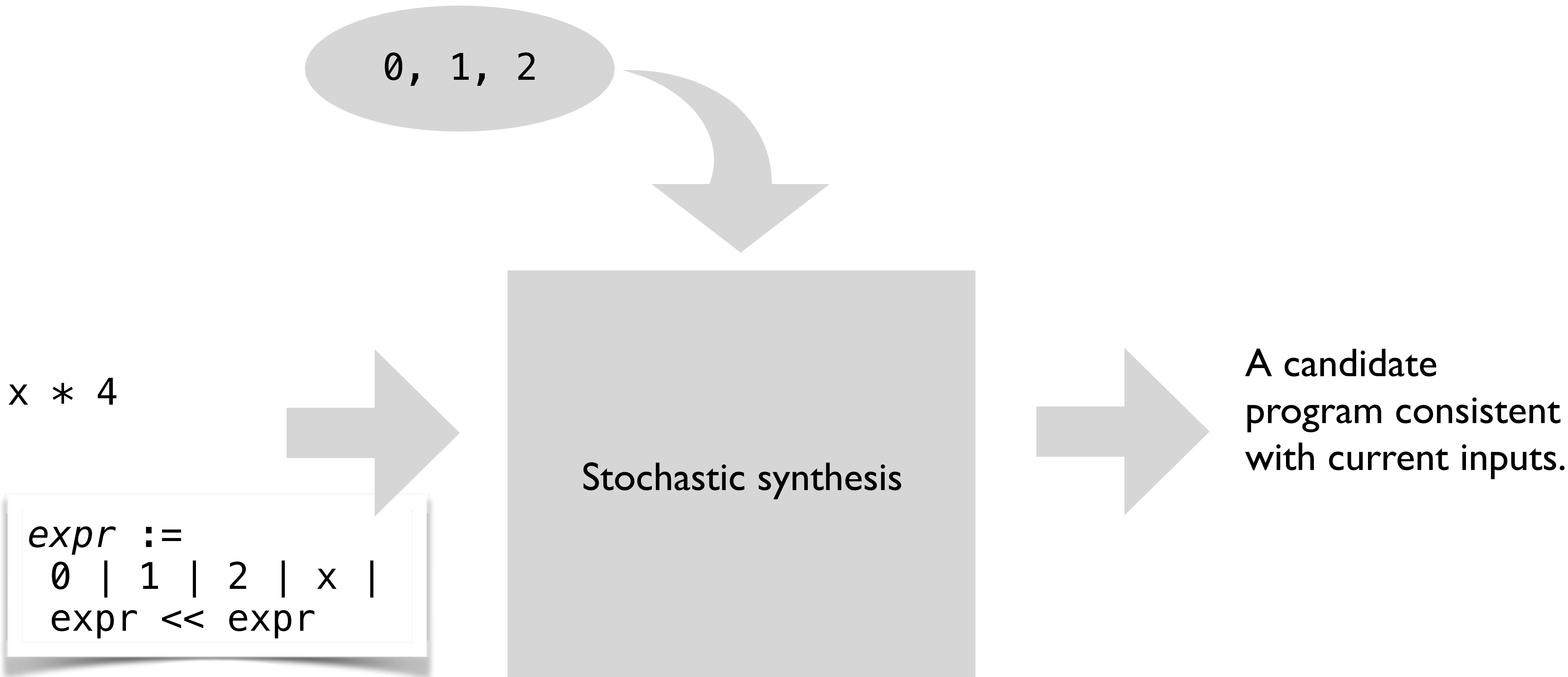
K=*1*: 0, 1, 2, x

K=2: 1 << 2, 2 << 2,

x << 1, **x << 2**

[Udupa et al, PLDI'13]

Synthesizing programs with stochastic search



[Schkufza et al, ASPLOS'13]

Synthesizing programs with stochastic search

```
0, 1, 2
```

```
x * 4
```

```
expr :=
 0 | 1 | 2 | x |
 expr << expr
```

- Use Metropolis-Hastings to sample expressions.
- Mutate the current candidate program and keep the mutation with probability proportional to its correctness w.r.t. the current inputs.

A candidate program consistent with current inputs.

[Schkufza et al, ASPLOS'13]

Summary

Today

- Deductive synthesis with axioms and E-graphs
- Inductive synthesis with solvers, enumeration, and stochastic search

Next lecture

- Solver-aided languages