# Building a State-of-The-Art Model Checker

— Sebastian Wolff —

*Technische Universität Kaiserslautern*

# What is Model Checking?

- proving correctness
  - vs. testing
- w.r.t. a specification
- automated

# Automated Proofs

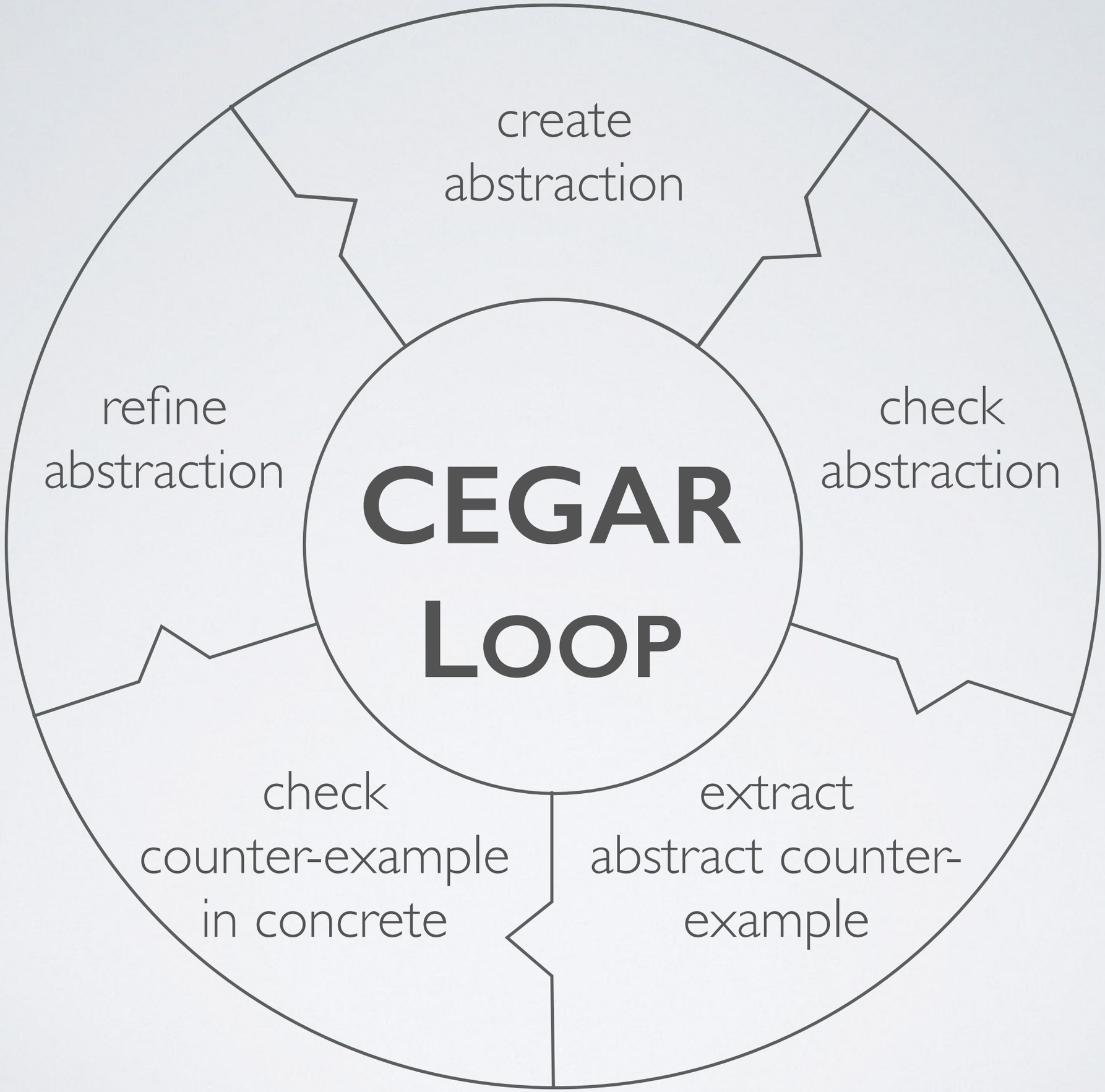
- state space exploration
  - control locations, variable values, …
  - is there a bad state?
- multiple challenges

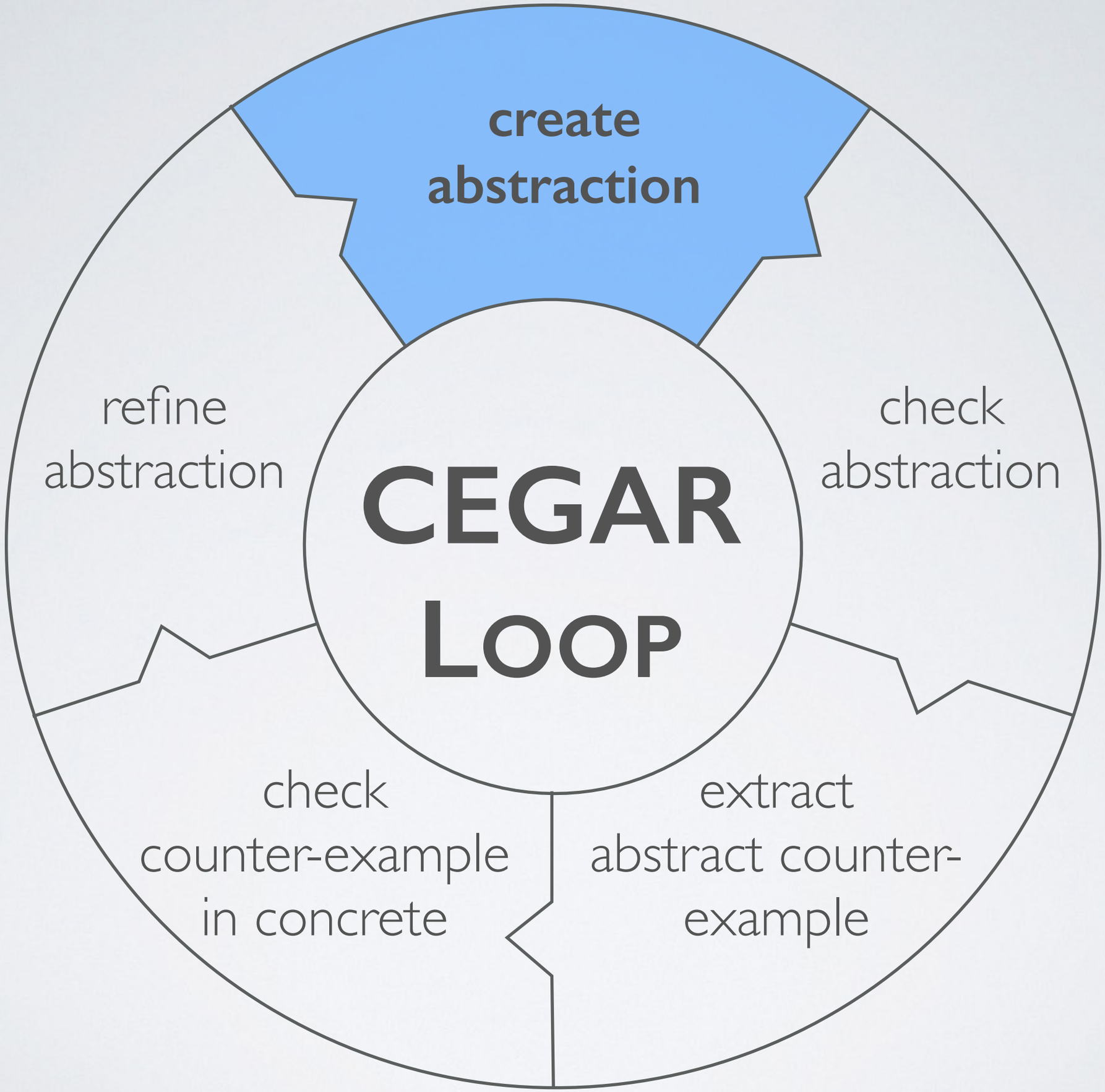
# Challenges

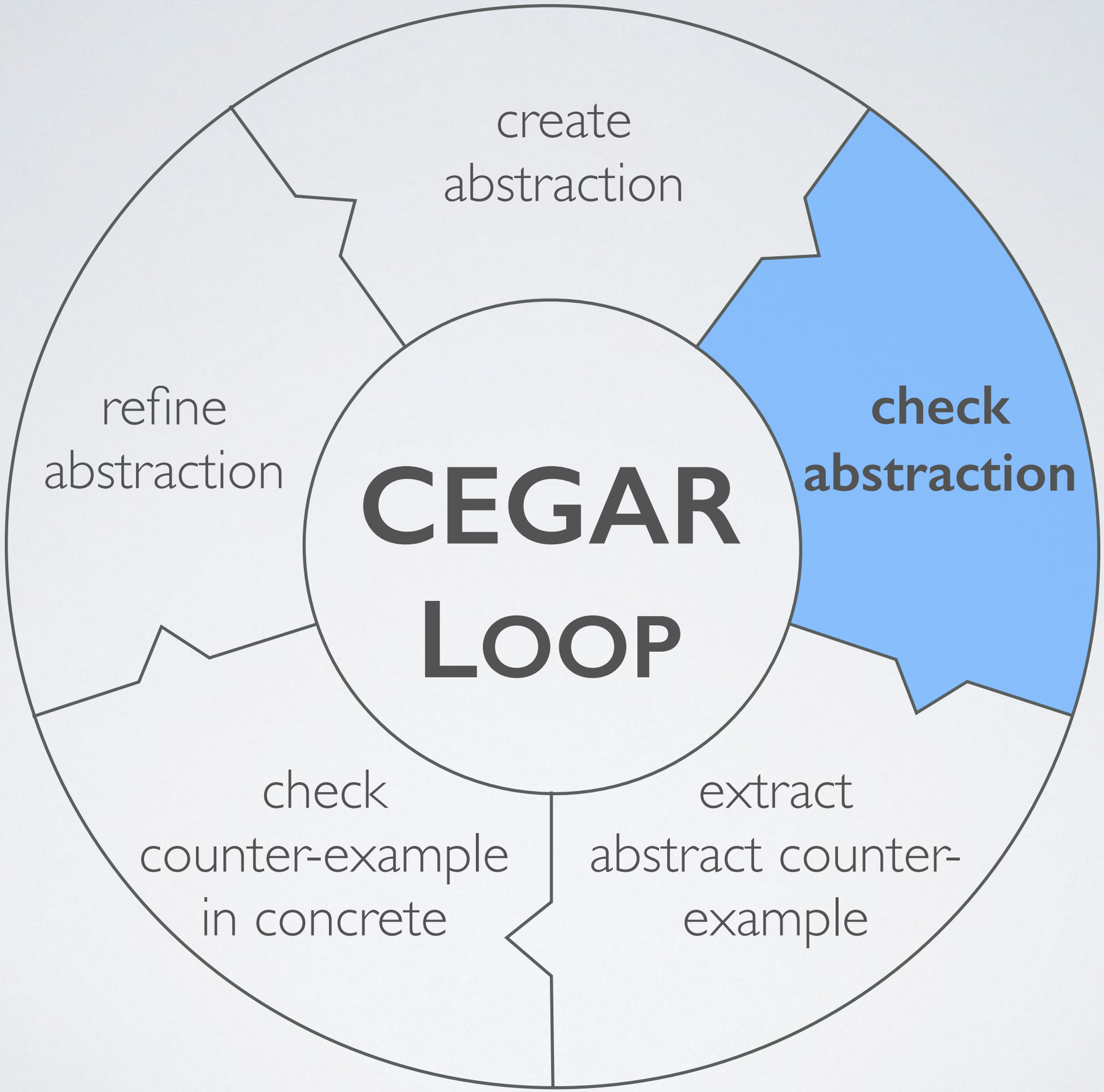
- unbounded recursion
- infinite data types
- unbounded heap
- specification
- …

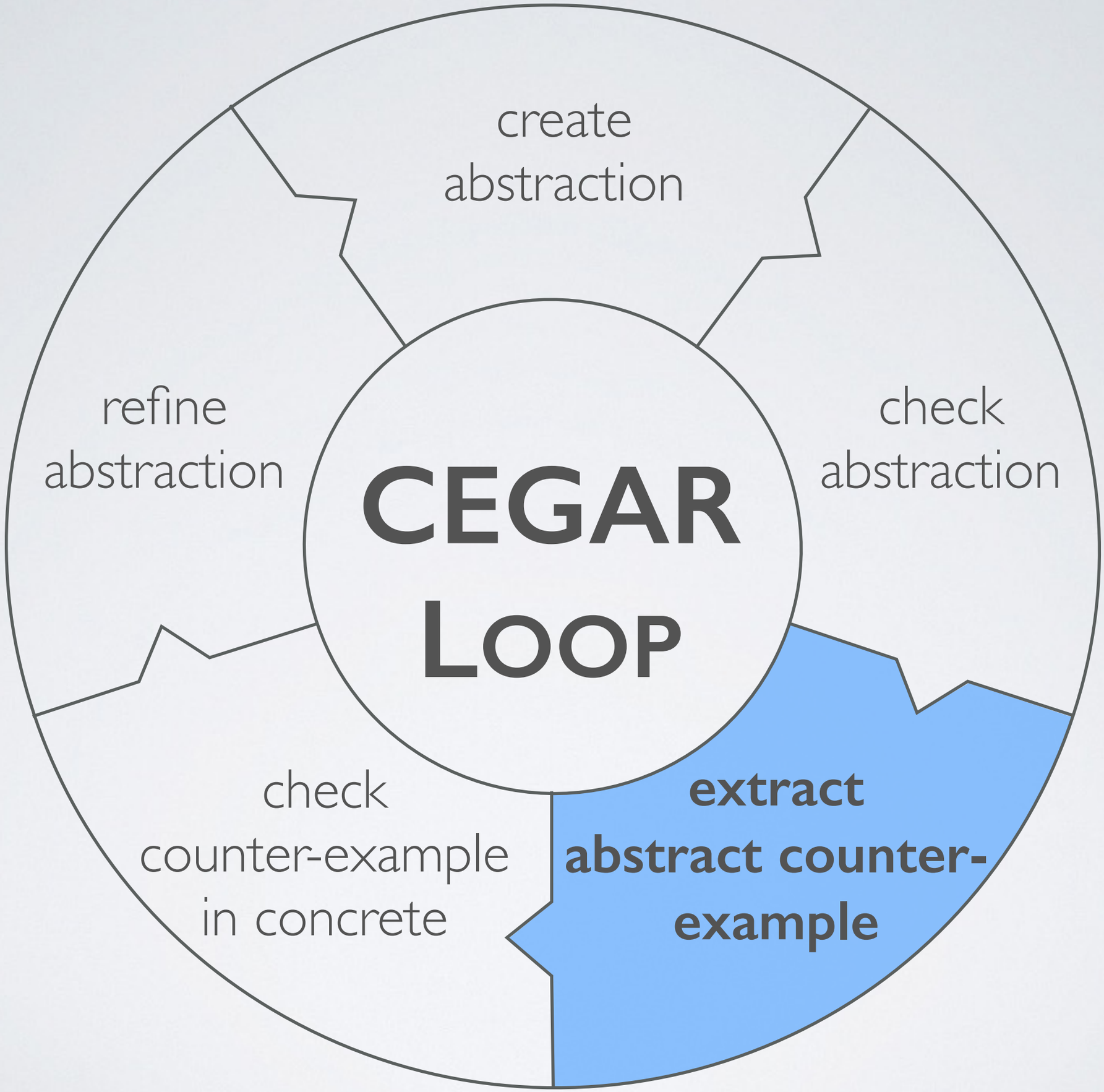
# Challenges

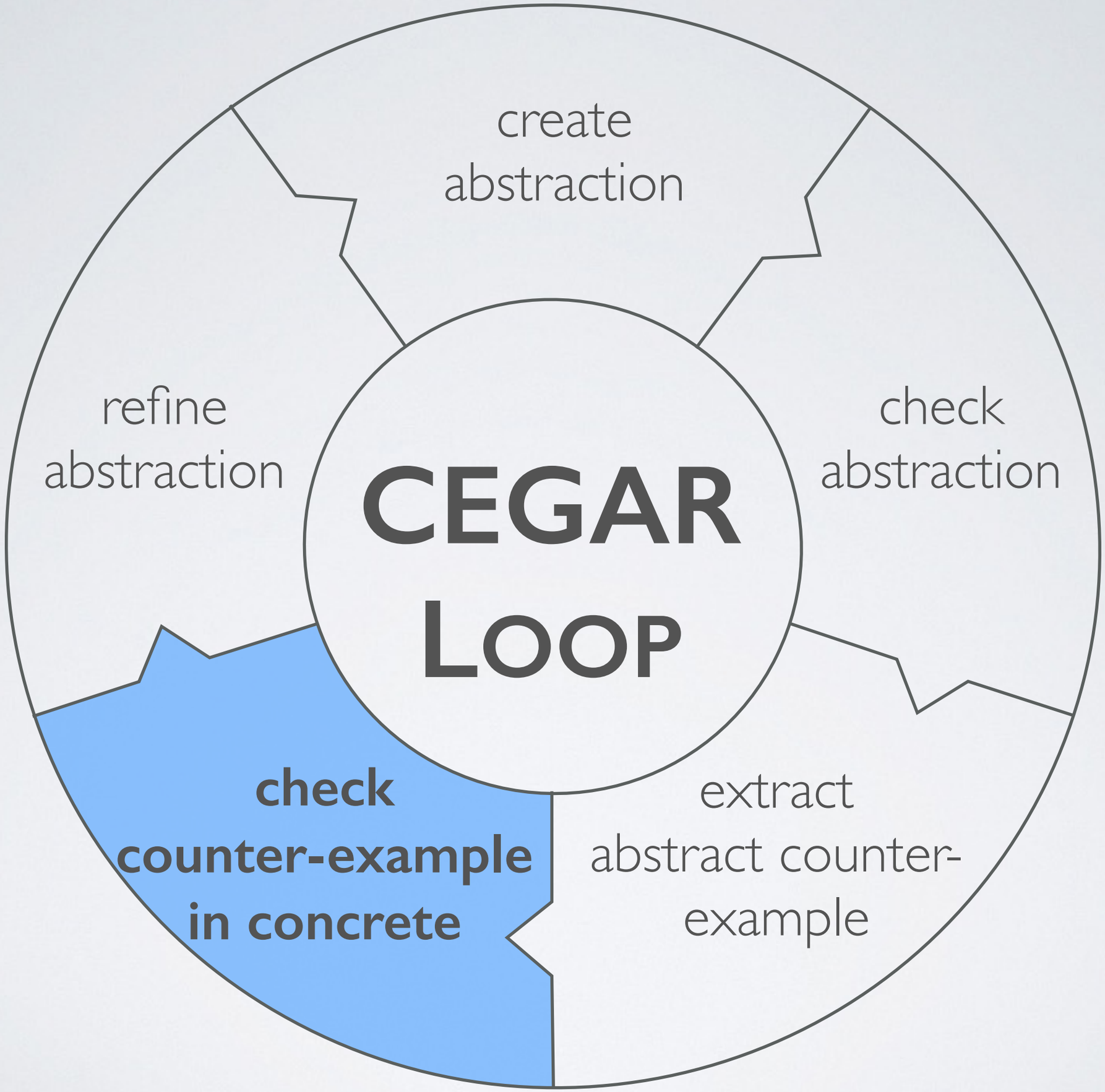
- **unbounded recursion**
- **infinite data types**
- unbounded heap
- **specification → by assertions**
- …

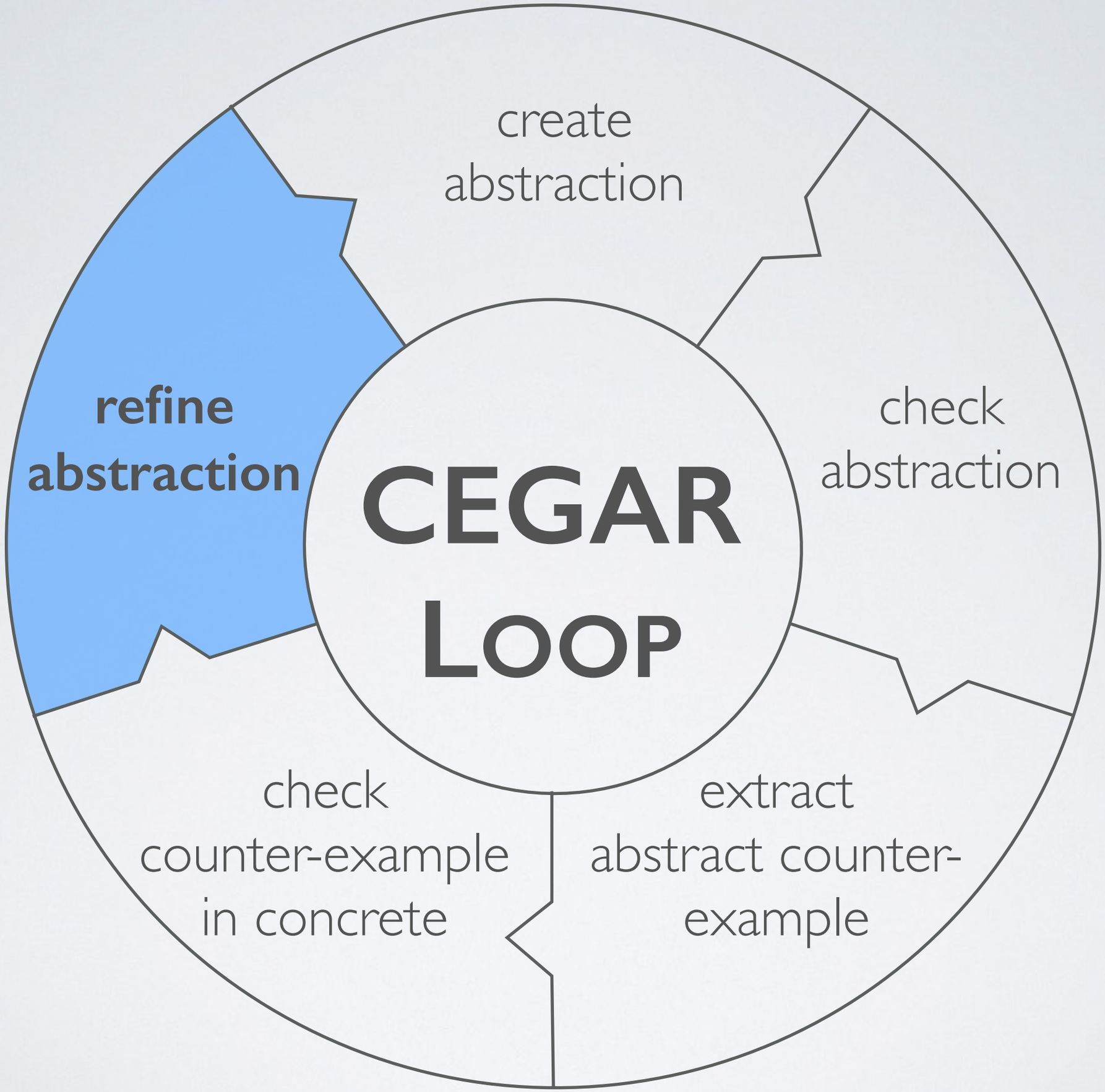

create
abstraction
check
abstraction
extract
abstract counter-
example
check
counter-example
in concrete
refine
abstraction
CEGAR
Loop


create
abstraction
check
abstraction
extract
abstract counter-
example
check
counter-example
in concrete
refine
abstraction
CEGAR
Loop


create
abstraction
check
abstraction
refine
abstraction
CEGAR
LOOP
check
counter-example
in concrete
extract
abstract counter-
example


create
abstraction
refine
abstraction
CEGAR
Loop
check
abstraction
check
counter-example
in concrete
extract
abstract counter-
example


create
abstraction
refine
abstraction
CEGAR
Loop
check
abstraction
check
counter-example
in concrete
extract
abstract counter-
example


create
abstraction
check
abstraction
extract
abstract counter-
example
check
counter-example
in concrete
refine
abstraction
CEGAR
LOOP

# Predicate Abstraction

- aim: from infinite to finite
- don't track values
- track *useful* relations among values
- predicates evaluate to *true* or *false*

# PREDICATE ABSTRACTION

```
int x, y;

void main() {
    // ...
    x = x - 1;
    if (x <= y)
        swap();
    assert(x >= y);
}

void swap() { ... }
```

```
bool p1; // x <= y

void main() {
    // ...
    p1 = p1 ? p1 : *;
    if (p1)
        swap();
    assert(!p1);
}

void swap() { ... }
```

# PREDICATE ABSTRACTION

```
int x, y;

void main() {
    // ...
    x = x - 1;
    if (x <= y)
        swap();
    assert(x >= y);
}

void swap() { ... }
```

```
bool p1; // x <= y

void main() {
    // ...
    p1 = p1 ? p1 : *;
    if (p1)
        swap();
    assert(!p1);
}

void swap() { ... }
```

# PREDICATE ABSTRACTION

```
int x, y;

void main() {
    // ...
    x = x - 1;
    if (x <= y)
        swap();
    assert(x >= y);
}

void swap() { ... }
```

```
bool p1; // x <= y

void main() {
    // ...
    p1 = p1 ? p1 : *;
    if (p1)
        swap();
    assert(!p1);
}

void swap() { ... }
```

# PREDICATE ABSTRACTION

```
int x, y;

void main() {
    // ...
    x = x - 1;
    if (x <= y)
        swap();
    assert(x >= y);
}

void swap() { ... }
```

```
bool p1; // x <= y

void main() {
    // ...
    p1 = p1 ? p1 : *;
    if (p1)
        swap();
    assert(!p1);
}

void swap() { ... }
```

# PREDICATE ABSTRACTION

```
int x, y;

void main() {
    // ...
    x = x - 1;
    if (x <= y)
        swap();
    assert(x >= y);
}

void swap() { ... }
```

```
bool p1; // x <= y

void main() {
    // ...
    p1 = p1 ? p1 : *;
    if (p1)
        swap();
    assert(!p1);
}

void swap() { ... }
```

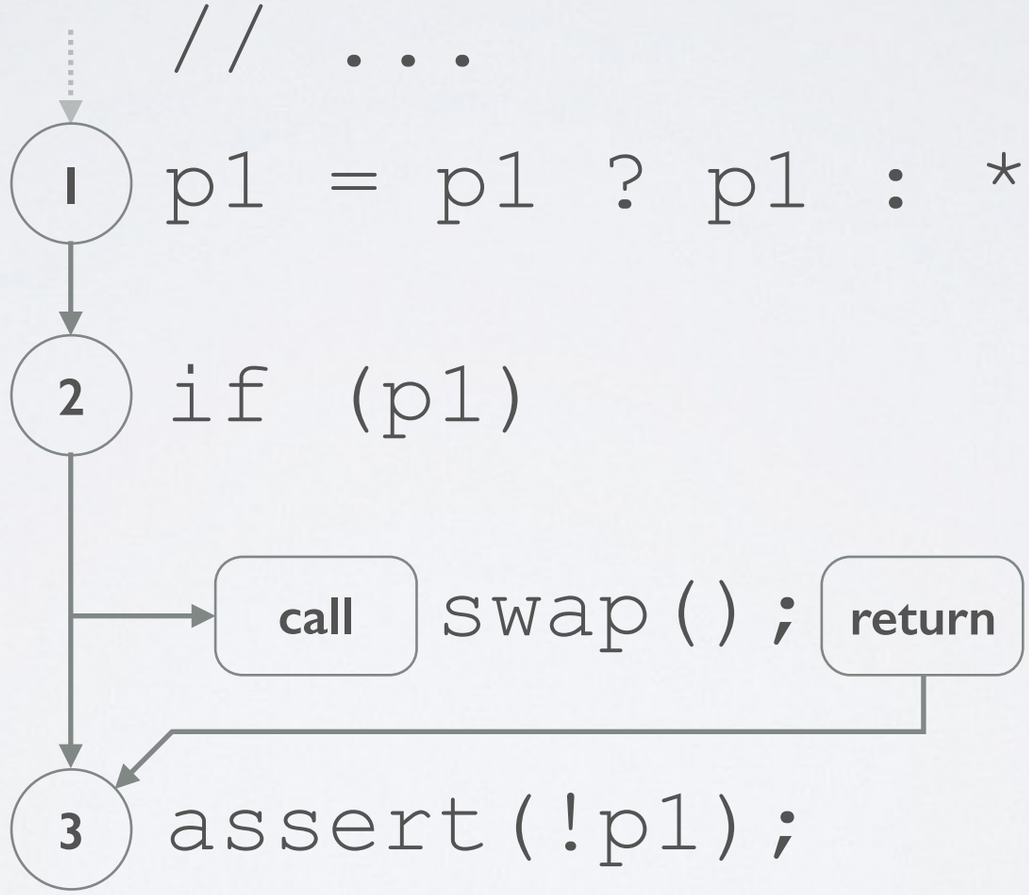
# Check for Correctness

- reduce to reachability check
  - translate to control flow graph
  - bad state reachable?
- procedure summaries (skipped)

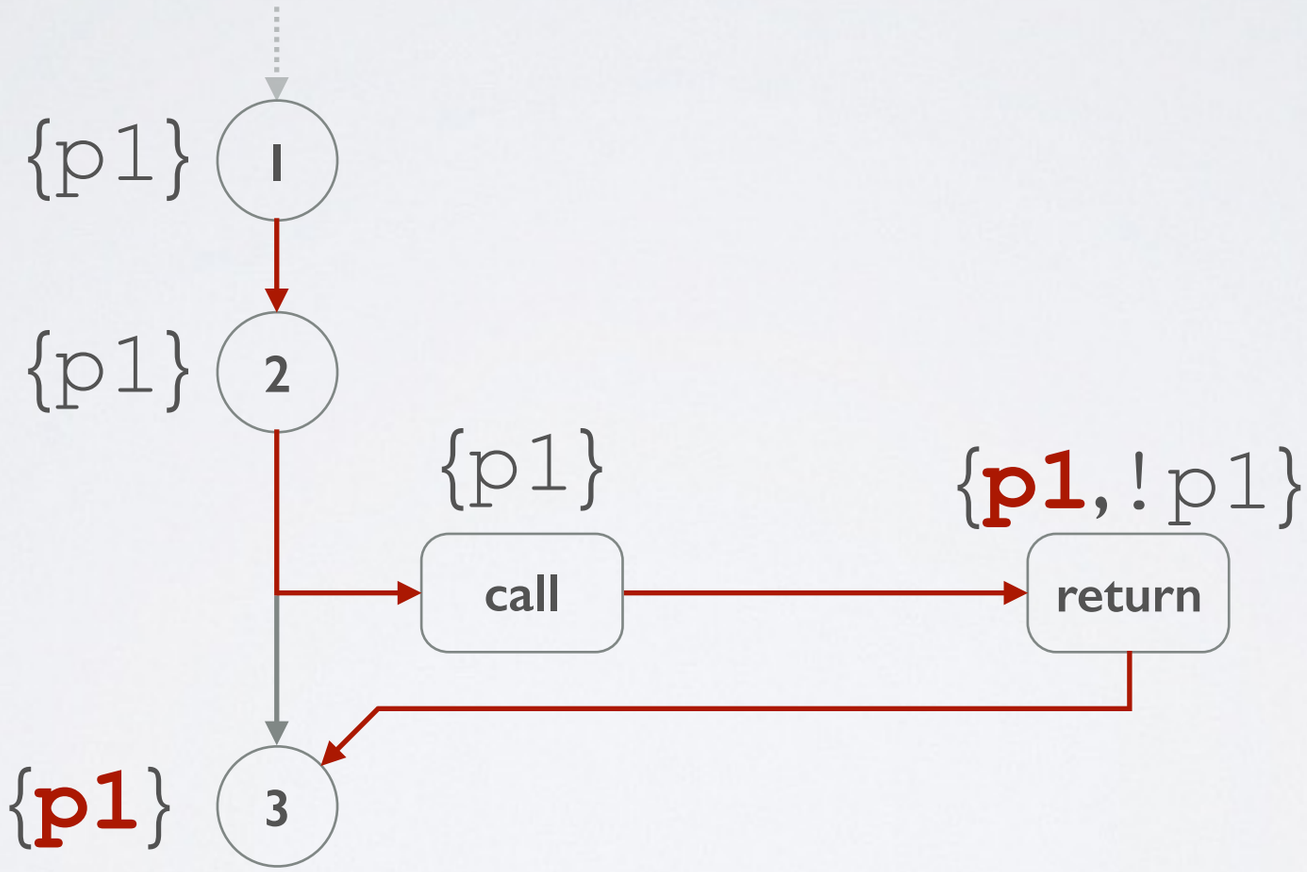
# EXAMPLE

```
void main() {
    // ...
    p1 = p1 ? p1 : *;
    if (p1)
        swap();
    assert(!p1);
}

void swap() { ... }
```




# Extract Counter-Example




# Check Feasibility

- counter-example trace possible?
  - judge in original program
- weakest preconditions
  - {*true*} trace {*false*} valid Hoare triple?

# Example

```
x = x - 1;

if (x <= y)

        swap();

assert(x >= y);
```

fails in counter-example

# Example

```
x = x - 1;

if (x <= y)
    // x > y
    swap();
// x < y
assert(x >= y);
```

fails in counter-example

# Example

```
x = x - 1;

if (x <= y)
    // x > y
    swap();
// x < y
assert(x >= y);
```

fails in counter-example

# Refine Abstraction

- abstraction too imprecise
- need new/better predicates
- remove spurious counter-example
- Craig interpolation

# There's a Prototype

- ~5500 line C++ code
  - CUDD BDD library
  - Z3 SMT solver
- >90% time spend on generating abstractions

— FIN —

# BACKUP SLIDES

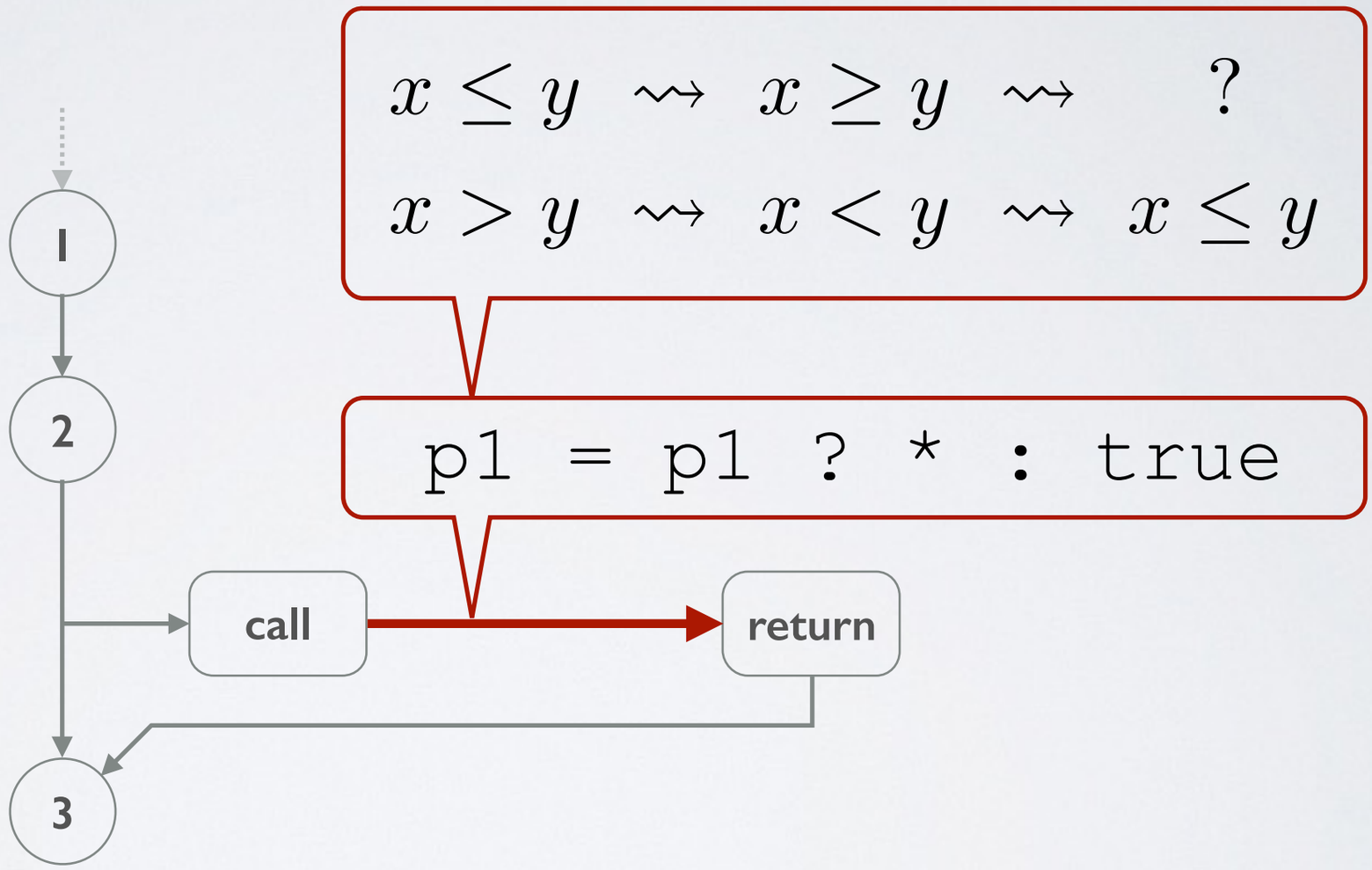
# Full WP-Example

```
{true}
x = x - 1;
{x <= y ∨ x > y}
assume(x <= y)
{x <= y}
swap();
{false ∨ x >= y}
assume(x < y);
{false}
assert(false);
{false}
```

# Procedure Summaries

- sub-analysis for function calls
- extend CFG with summary edges
  - impact of function to global variables
  - on demand
- termination check (!)

# Example


$x \leq y \rightsquigarrow x \geq y \rightsquigarrow ?$
$x > y \rightsquigarrow x < y \rightsquigarrow x \leq y$
p1 = p1 ? * : true
1
2
call
return
3