ETH zürich

HYPRA: A DEDUCTIVE PROGRAM VERIFIER FOR HYPER HOARE LOGIC

Thibault Dardinier*, **Anqi Li***, Peter Müller

Oct 25, 2024

MOTIVATION

- Hyperproperties: properties over multiple executions of the same program

MOTIVATION

- Hyperproperties: properties over multiple executions of the same program

$\forall^+$

Determinism

MOTIVATION

- Hyperproperties: properties over multiple executions of the same program

$\forall^+$

Determinism

$\exists^+$

Existence of bugs
e.g. violation of determinism

MOTIVATION

- Hyperproperties: properties over multiple executions of the same program

$\forall^+$

$\exists^+$

$\forall^+\exists^+$

Determinism

Existence of bugs
e.g. violation of determinism

Generalized non-interference (GNI)

MOTIVATION

➤ Hyperproperties: properties over multiple executions of the same program

$\forall^+$ — Determinism

$\exists^+$ — Existence of bugs e.g. violation of determinism

$\forall^+\exists^+$ — Generalized non-interference (GNI)

$\exists^+\forall^+$ — Negation of GNI; Existence of a minimum

MOTIVATION

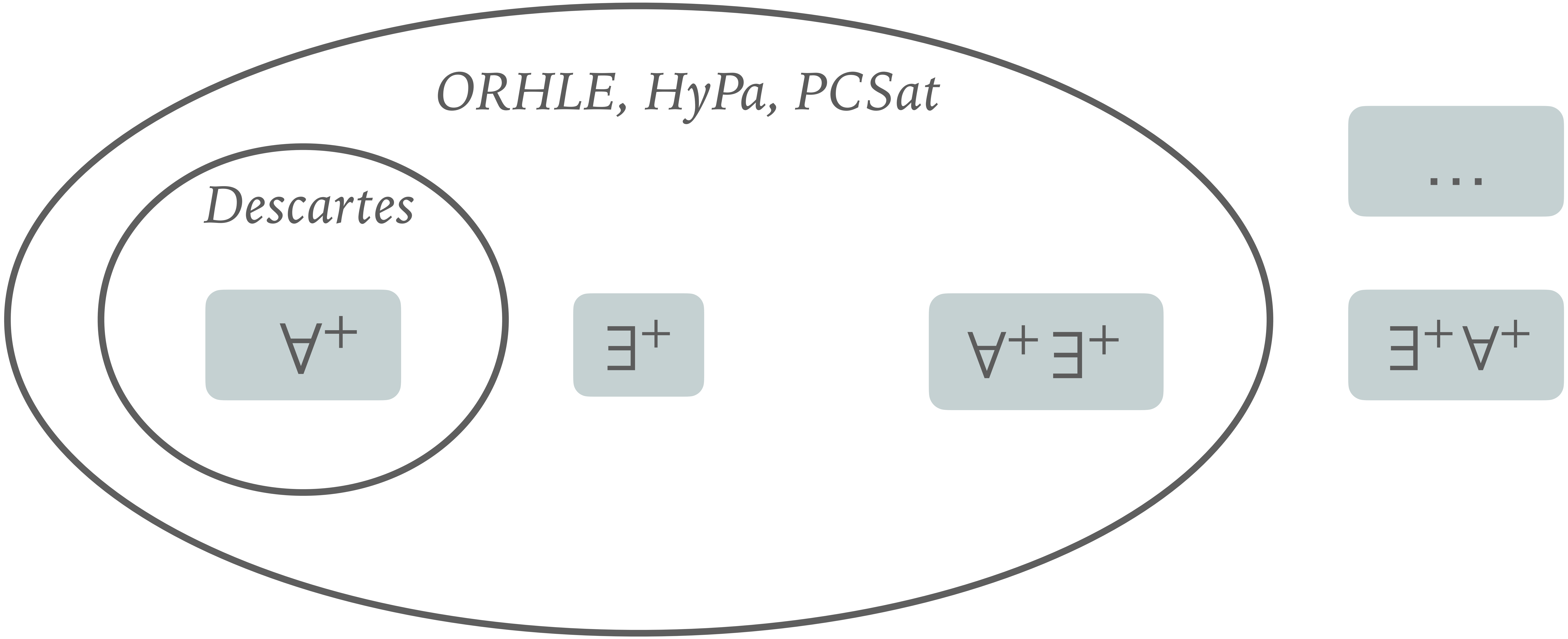
- Hyperproperties: properties over multiple executions of the same program

…

$\forall^+$ $\exists^+$ $\forall^+\exists^+$ $\exists^+\forall^+$

MOTIVATION

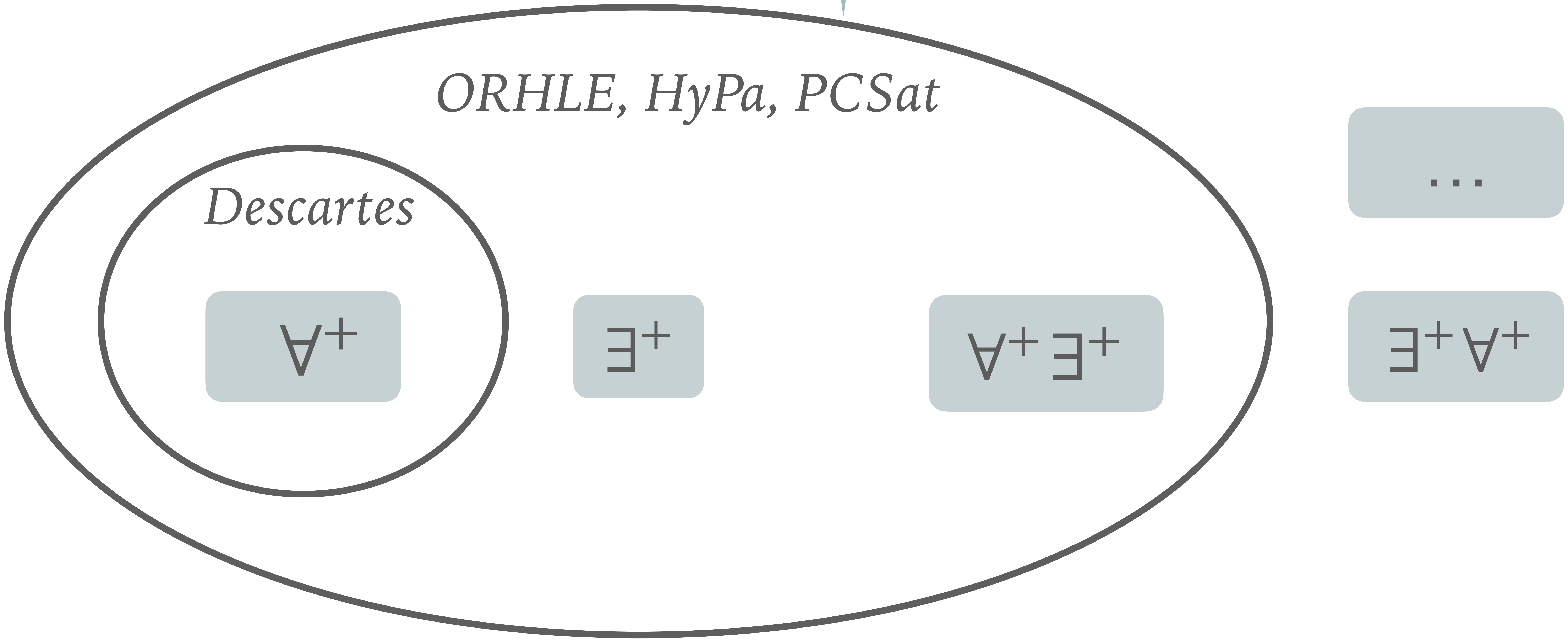
- Hyperproperties: properties over multiple executions of the same program



MOTIVATION

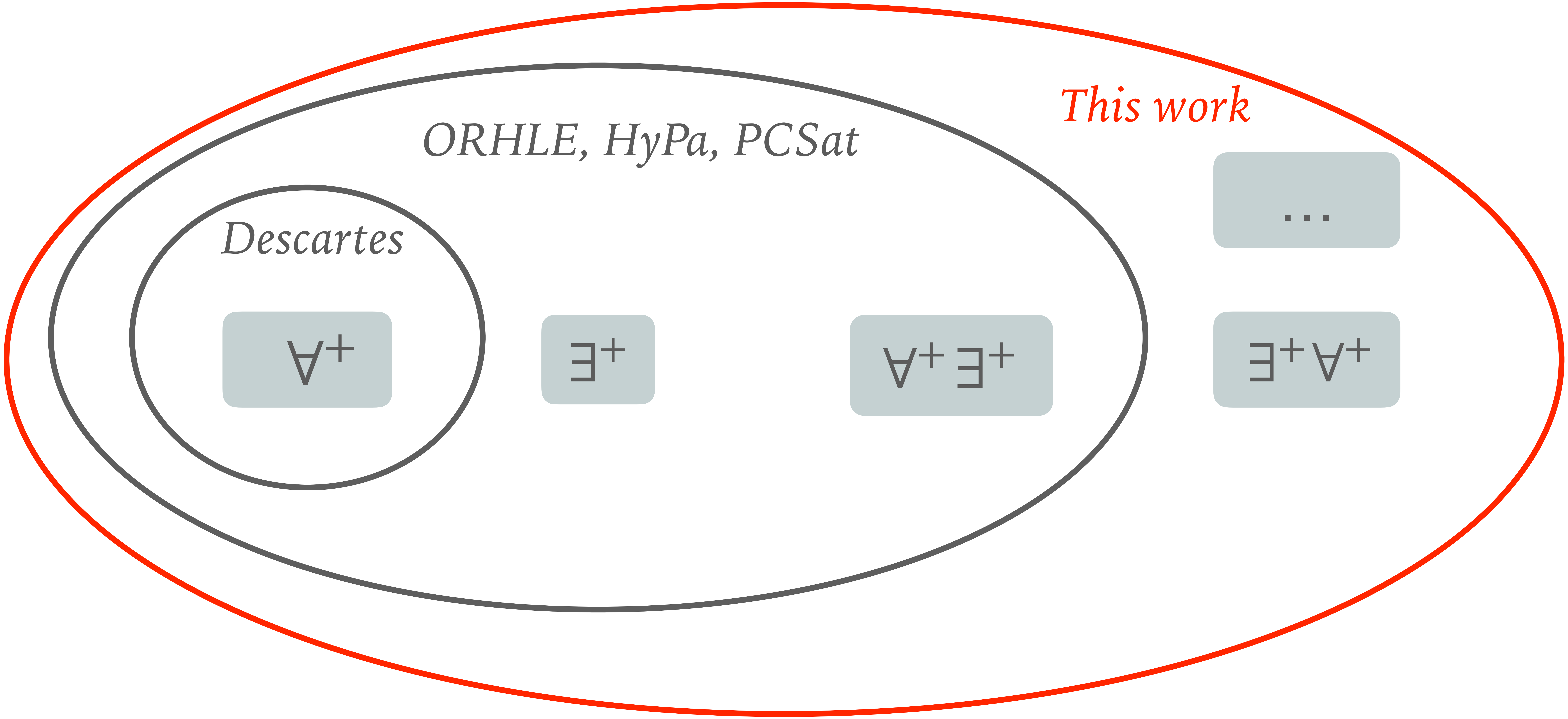
Limited to a fixed quantification scheme

- Hyperproperties: properties over … e same program



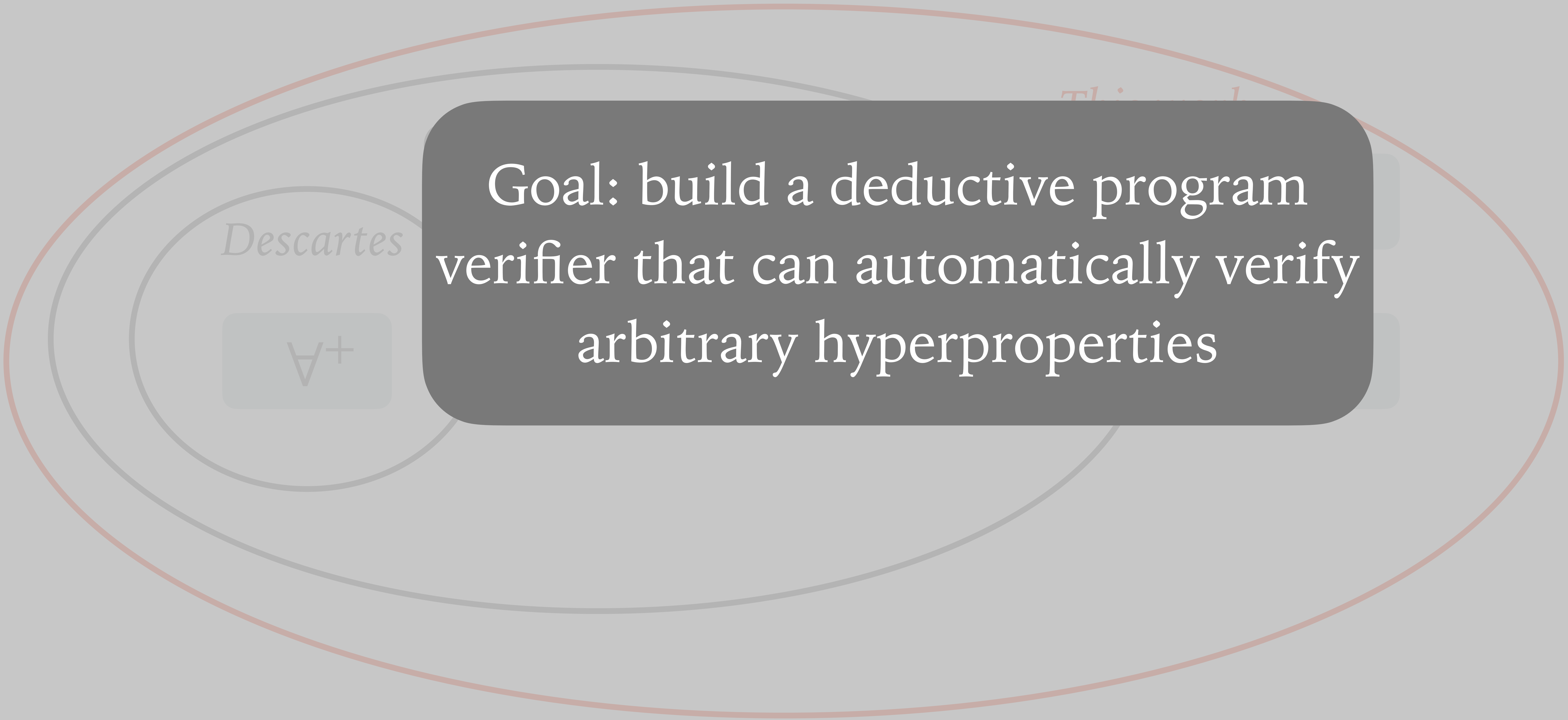
MOTIVATION

- Hyperproperties: properties over multiple executions of the same program



MOTIVATION

- Hyperproperties: properties over multiple executions of the same program



Goal: build a deductive program verifier that can automatically verify arbitrary hyperproperties

BACKGROUND: HYPER HOARE LOGIC (HHL)

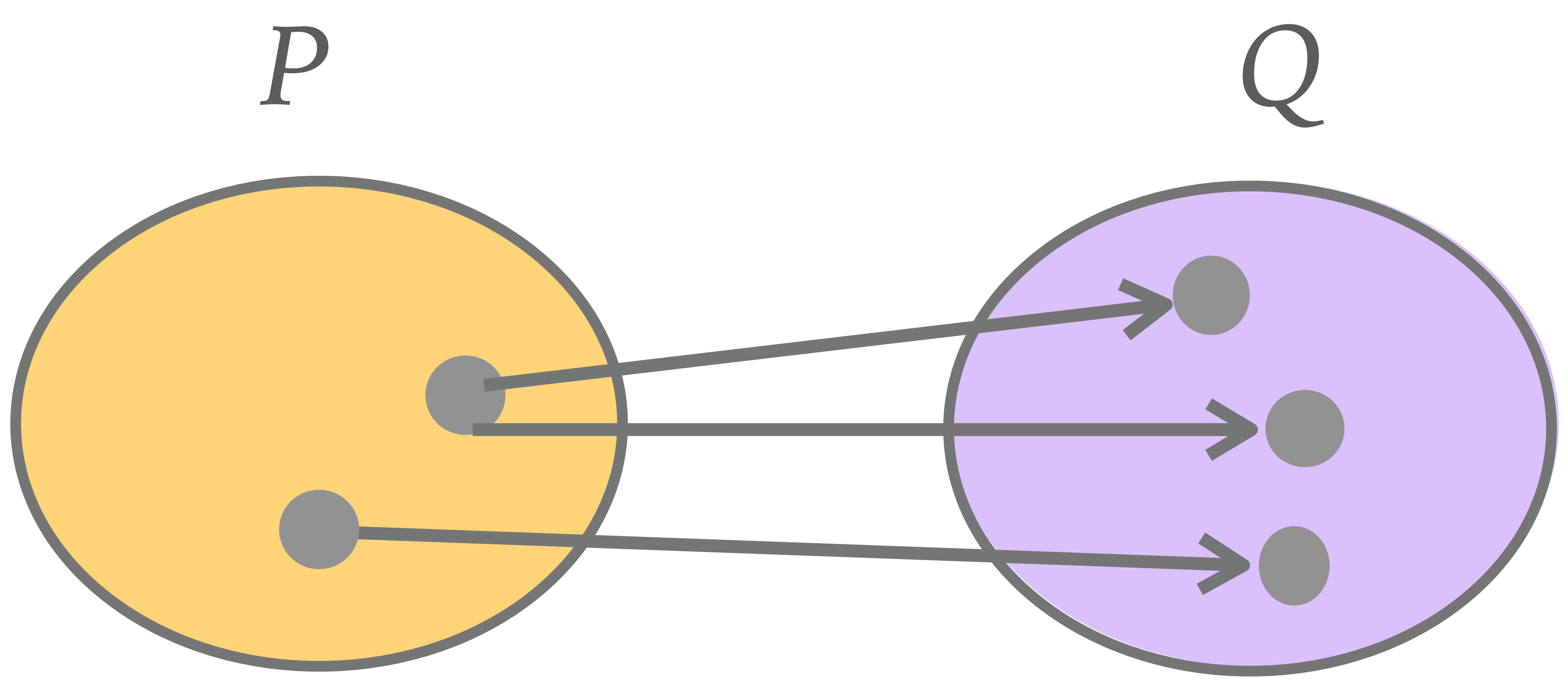
Hoare Logic

Hyper Hoare Logic

BACKGROUND: HYPER HOARE LOGIC (HHL)

Hoare Logic

Hyper Hoare Logic

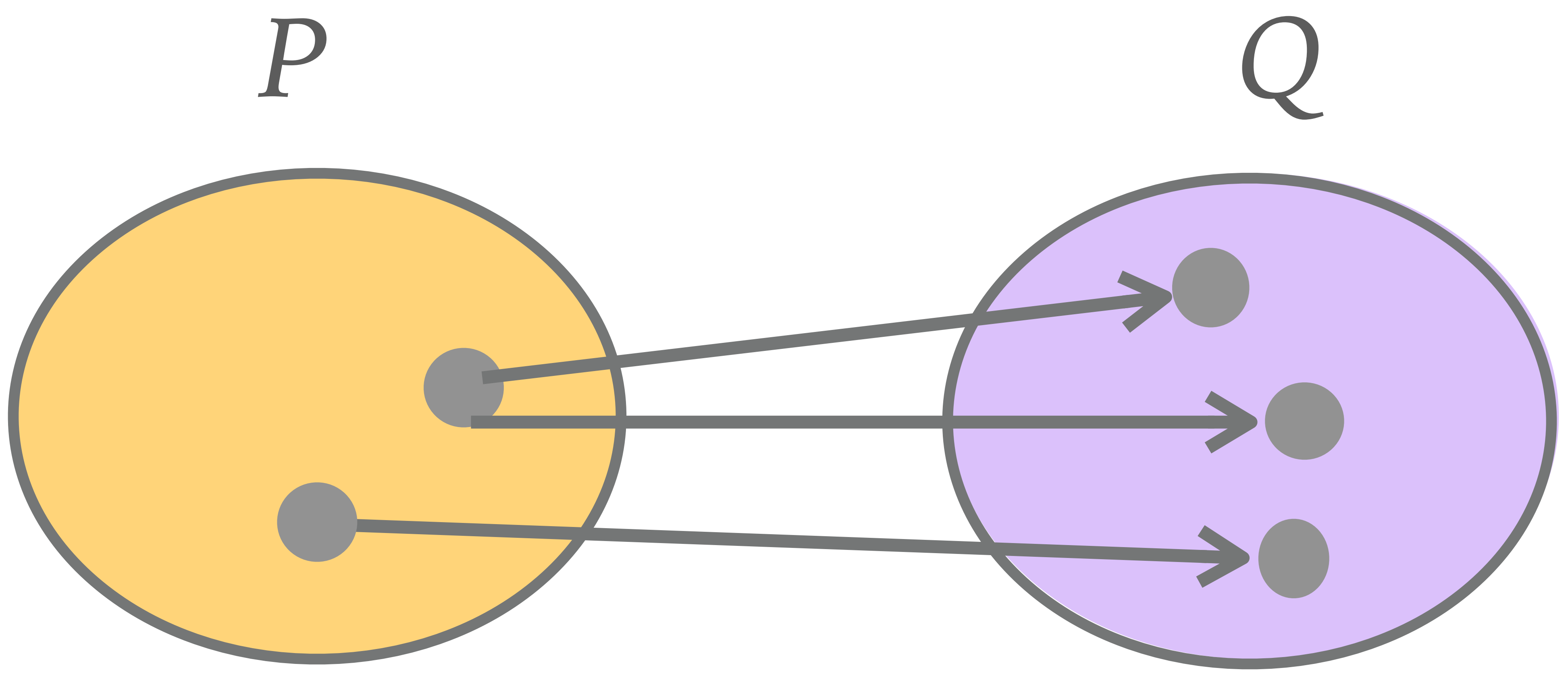


Hoare triple $\vDash \{P\}C\{Q\}$

P and Q are predicates over states

BACKGROUND: HYPER HOARE LOGIC (HHL)

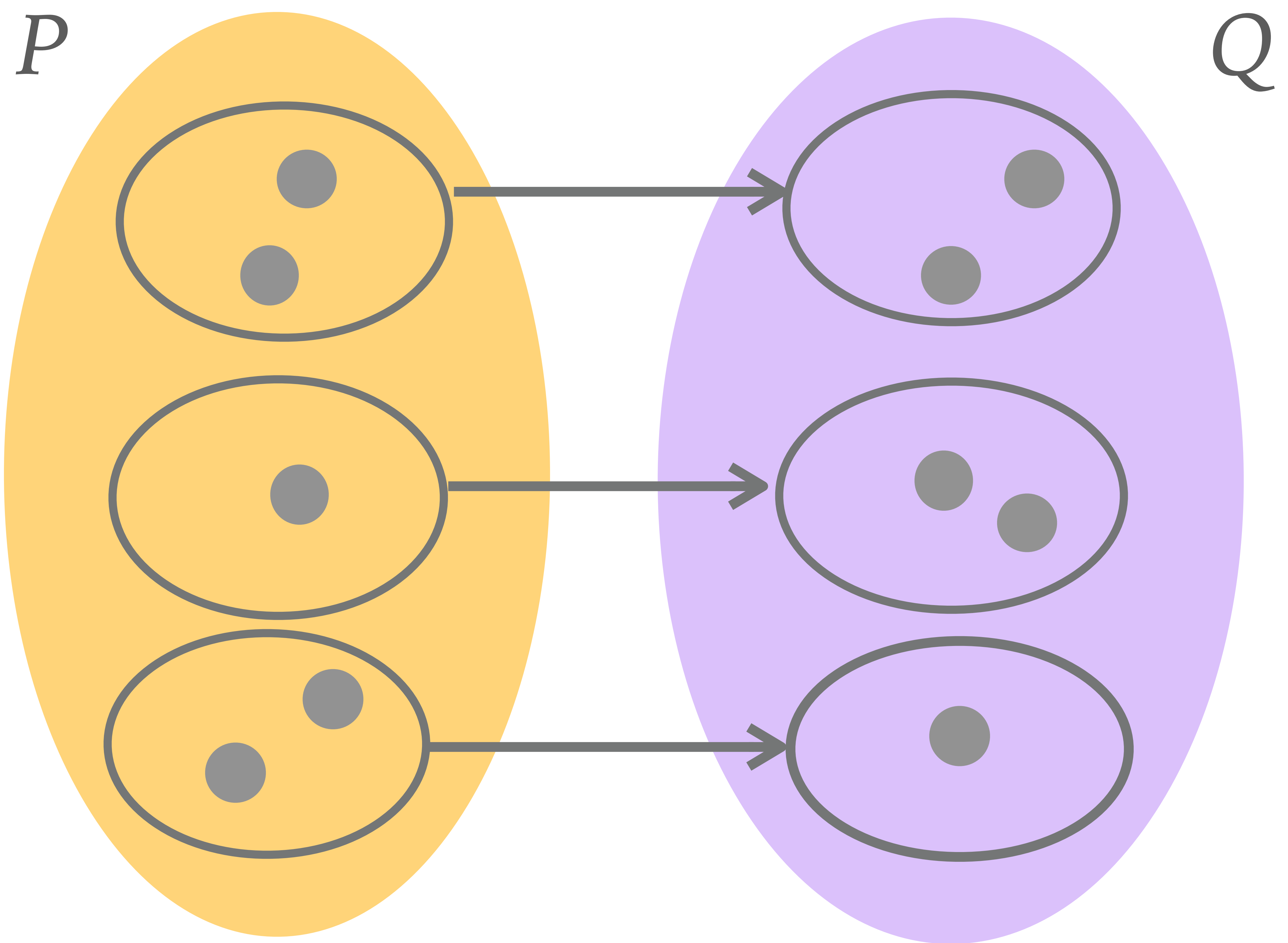
Hoare Logic



Hoare triple $\models \{P\}C\{Q\}$

P and Q are predicates over ***states***

Hyper Hoare Logic

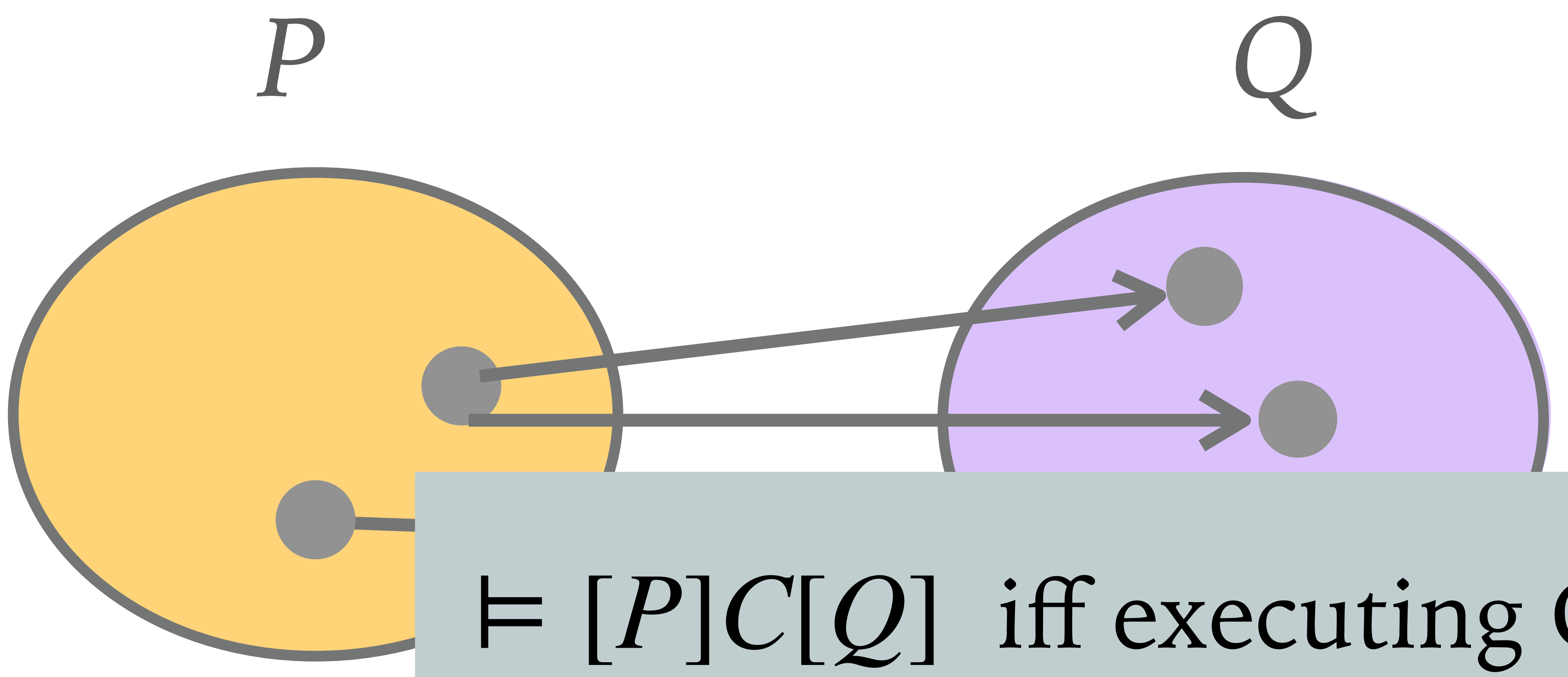


Hyper triple $\models [P]C[Q]$

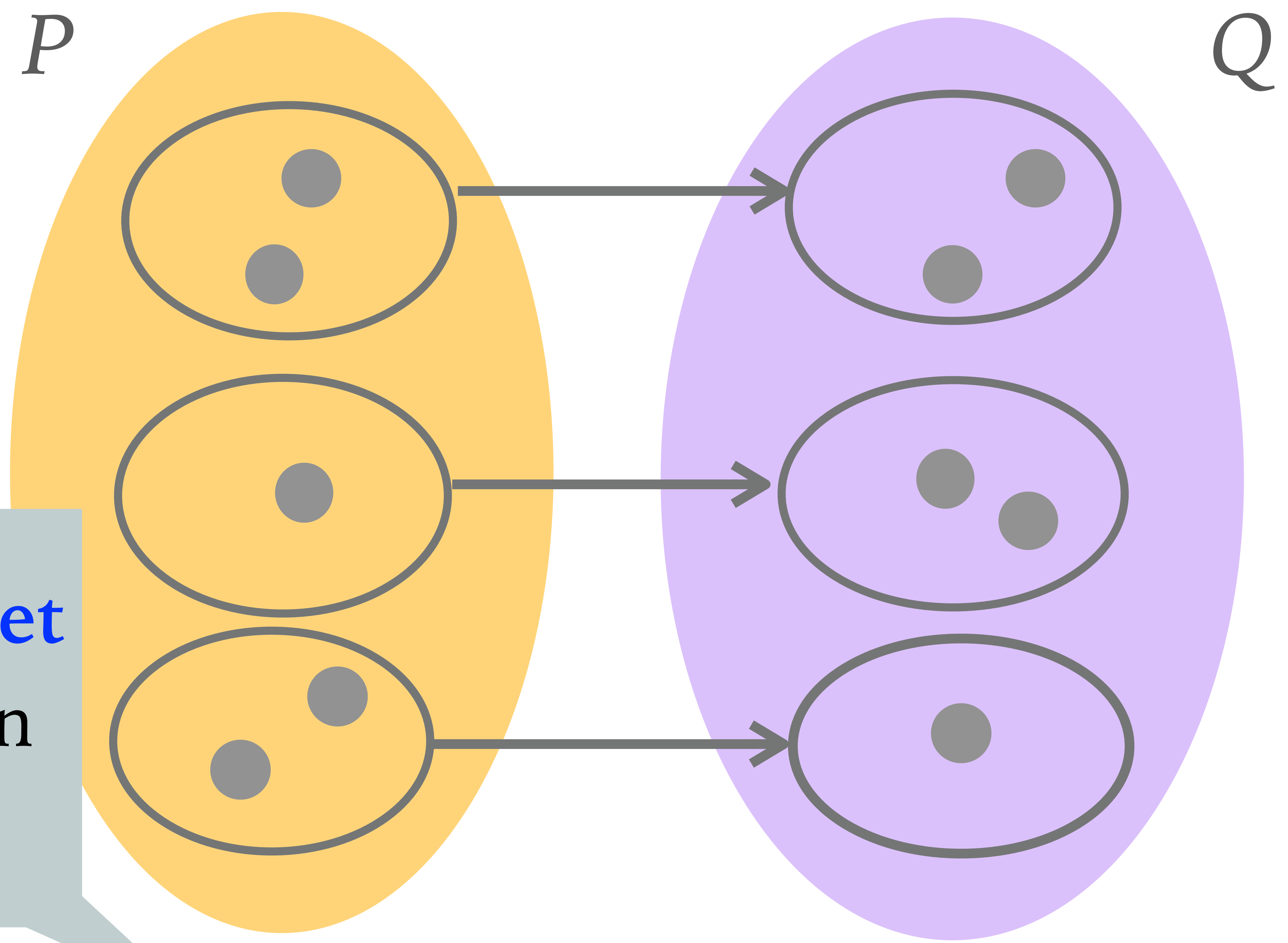
P and Q are predicates over ***sets of states***

BACKGROUND: HYPER HOARE LOGIC (HHL)

Hoare Logic



Hyper Hoare Logic



$\vDash [P]C[Q]$ iff executing C in any **set of initial states** satisfying P results in a **set of final states** satisfying Q

Hoare triple $\vDash \{P\}C\{Q\}$
*P and Q are predicates over **states***

Hyper triple $\vDash [P]C[Q]$
*P and Q are predicates over **sets of states***

BACKGROUND: HYPER HOARE LOGIC (HHL)

- Hyper-assertions: predicates over sets of states
  - Can explicitly quantify over the states with ∀ and ∃ quantifiers

BACKGROUND: HYPER HOARE LOGIC (HHL)

- Hyper-assertions: predicates over sets of states
  - Can explicitly quantify over the states with ∀ and ∃ quantifiers

$$[\lambda S \,.\, \forall \sigma_1, \sigma_2 \in S \,.\, \sigma_1(in) = \sigma_2(in)]$$

```
out := in
```

$$[\lambda S' \,.\, \forall \sigma_1', \sigma_2' \in S' \,.\, \sigma_1'(out) = \sigma_2'(out)]$$

THIS WORK

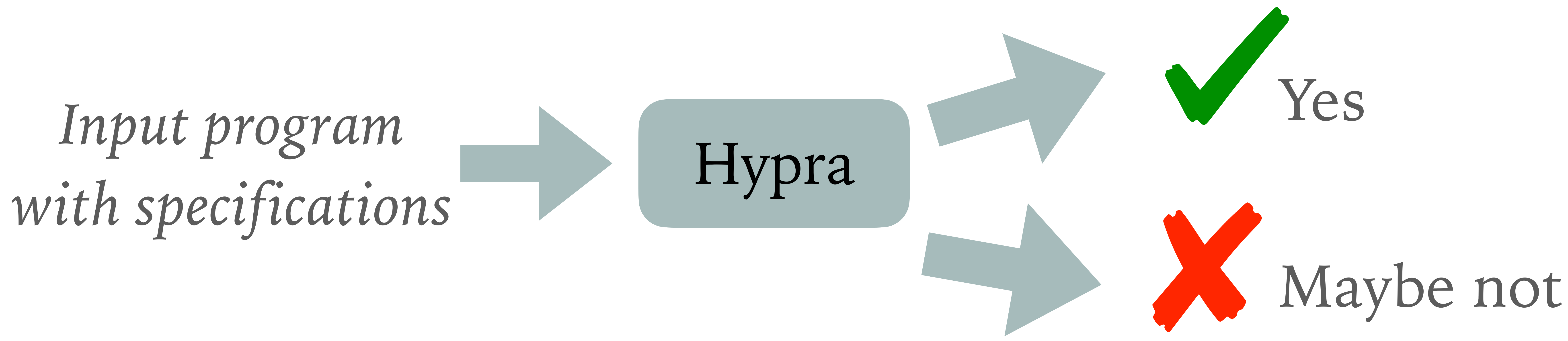
- An automated deductive program verifier for HHL

THIS WORK

- An automated deductive program verifier for HHL

THIS WORK

- An automated deductive program verifier for HHL



THIS WORK

- An automated deductive program verifier for HHL

THIS WORK

- An automated deductive program verifier for HHL

- Challenges

  1. Design an encoding that tracks an unbounded number of executions

  2. Make the encoding work with SMT solvers in practice

HIGH-LEVEL ENCODING

Input program
with specifications

```
method simple(x: Int)
returns (y: Int)
requires P
ensures Q
{
  C
}
```

HIGH-LEVEL ENCODING


```
method simple(x: Int)
returns (y: Int)
requires P
ensures Q
{
  C
}
```

```
var S: Set[State]
assume $S \vDash P$
var S’: Set[State]
// Constrain S’ based on S and C
. . .
assert $S' \vDash Q$
```

EXAMPLE

- $C \triangleq x := 5$

EXAMPLE

- $C \triangleq x := 5$

$$\texttt{assume}\ \forall \sigma \in S\,.\,\sigma[x := 5] \in S'$$

EXAMPLE

- C $\triangleq$ x := 5

$$\texttt{assume}\ \forall \sigma \in S\,.\,\sigma[x := 5] \in S'$$

$$\texttt{assume}\ \forall \sigma' \in S'\,.\,\exists \sigma \in S\,.\,\sigma' = \sigma[x := 5]$$

EXAMPLE

- $C \triangleq x := 5$

`// Precondition`

`. . .`

`assume` $\forall \sigma \in S \, . \, \sigma[x := 5] \in S'$

`assume` $\forall \sigma' \in S' . \, \exists \sigma \in S \, . \, \sigma' = \sigma[x := 5]$

`// Postcondition`

`assert` $(\forall \sigma' \in S' . \ldots) \land (\exists \sigma' \in S' . \ldots)$

EXAMPLE

- $C \triangleq x := 5$

```
// Precondition
...
```

$\texttt{assume}\ \forall \sigma \in S \,.\, \sigma[x := 5] \in S'$

Useful for verifying $\forall^{+}$-properties

$\texttt{assume}\ \forall \sigma' \in S' \,.\, \exists \sigma \in S \,.\, \sigma' = \sigma[x := 5]$

```
// Postcondition
```

$\texttt{assert}\ (\forall \sigma' \in S' \,.\, \ldots) \land (\exists \sigma' \in S' \,.\, \ldots)$

EXAMPLE

- $C \triangleq x := 5$

```
// Precondition
. . .
assume ∀σ ∈ S . σ[x := 5] ∈ S′
assume ∀σ′ ∈ S′ . ∃σ ∈ S . σ′ = σ[x := 5]
// Postcondition
assert (∀σ′ ∈ S′ . ...) ∧ (∃σ′ ∈ S′ . ...)
```

Useful for verifying $\exists^+$-properties

Useful for verifying $\forall^+$-properties

EXAMPLE

- $C \triangleq x := 5$

```
// Precondition
...
assume ∀σ ∈ S . σ[x := 5] ∈ S′
assume ∀σ′ ∈ S′ . ∃σ ∈ S . σ′ = σ[x := 5]
// Postcondition
assert (∀σ′ ∈ S′ . ...) ∧ (∃σ′ ∈ S′ . ...)
```

Useful for verifying $\exists^+$-properties

Useful for verifying $\forall^+$-properties

Challenge 1 solved

? *Challenge 2*

E-MATCHING

`assume` $\forall x \, . \, f(x) = 2x$

`assert` $f(10) = 20$

E-MATCHING

Trigger

$f(x)$

`assume` $\forall x \,.\, f(x) = 2x$

`assert` $f(10) = 20$

E-MATCHING

Trigger

$f(x)$

`assume` $\forall x \,.\, f(x) = 2x$

`assert` $f(10) = 20$

Matches the trigger

$x = 10$

E-MATCHING

Trigger

$f(x)$ *20*

`assume` $\forall x \,.\, f(x) = 2x$

`assert` $f(10) = 20$

Matches the trigger
$x = 10$

E-MATCHING

Trigger

$f(x)$

`assume` $\forall x \,.\, f(x) = 2x$

`assert` $f(10) = 20$

Trigger

$f(x)$

`assume` $\forall x \,.\, f(x) = f(2x)$

`assert` $f(10) = 20$

E-MATCHING

Trigger

$f(x)$

`assume` $\forall x\,.\,f(x) = 2x$

`assert` $f(10) = 20$

Trigger

$f(x)$

`assume` $\forall x\,.\,f(x) = f(2x)$

`assert` $f(10) = 20$

Matches the trigger
$x = 10$

E-MATCHING

Trigger

$f(x)$

`assume` $\forall x \,.\, f(x) = 2x$

`assert` $f(10) = 20$

Trigger

$f(x)$

$f(20)$

`assume` $\forall x \,.\, f(x) = f(2x)$

`assert` $f(10) = 20$

Matches the trigger
$x=10$

E-MATCHING

Trigger

$f(x)$

`assume` $\forall x\,.\, f(x) = 2x$

`assert` $f(10) = 20$

Trigger

$f(x)$

Matches the trigger again
$x\!=\!20$

$f(20)$

`assume` $\forall x\,.\, f(x) = f(2x)$

`assert` $f(10) = 20$

Matches the trigger
$x\!=\!10$

E-MATCHING

Trigger

$f(x)$

`assume` $\forall x \,.\, f(x) = 2x$

`assert` $f(10) = 20$

Trigger

$f(x)$

Matches the trigger again
$x = 20$

$f(20)$

`assume` $\forall x \,.\, f(x) = f(2x)$

`assert` $f(10) = 20$

Matches the trigger
$x = 10$

E-MATCHING

Trigger

$f(x)$

assume $\forall x \,.\, f(x) = 2x$

assert $f(10) = 20$

Trigger

$f(x)$

Matches the trigger again
x=20

$f(20)$

assume $\forall x \,.\, f(x) = f(2x)$

Matching loop

assert $f(10) = 20$

Matches the trigger
x=10

EXAMPLE REVISITED

- $C \triangleq x := 5$

EXAMPLE REVISITED

- $C \triangleq x := 5$

$\sigma \in S$

Trigger

`assume` $\forall \sigma \in S \,.\, \sigma[x := 5] \in S'$

`assume` $\forall \sigma' \in S' \,.\, \exists \sigma \in S \,.\, \sigma' = \sigma[x := 5]$

$\sigma' \in S'$

EXAMPLE REVISITED

- $C \triangleq x := 5$

$\sigma \in S$

Trigger

`assume` $\forall \sigma \in S \,.\, \sigma[x := 5] \in S'$

`assume` $\forall \sigma' \in S' \,.\, \exists \sigma \in S \,.\, \sigma' = \sigma[x := 5]$

$\sigma' \in S'$

EXAMPLE REVISITED

- $C \triangleq x := 5$

$\sigma \in S$

Trigger

`assume` $\forall \sigma \in S \, . \, \sigma[x := 5] \in S'$

`assume` $\forall \sigma' \in S' \, . \, \exists \sigma \in S \, . \, \sigma' = \sigma[x := 5]$

$\sigma' \in S'$

EXAMPLE REVISITED

- $C \triangleq x := 5$

$\sigma \in S$

Trigger

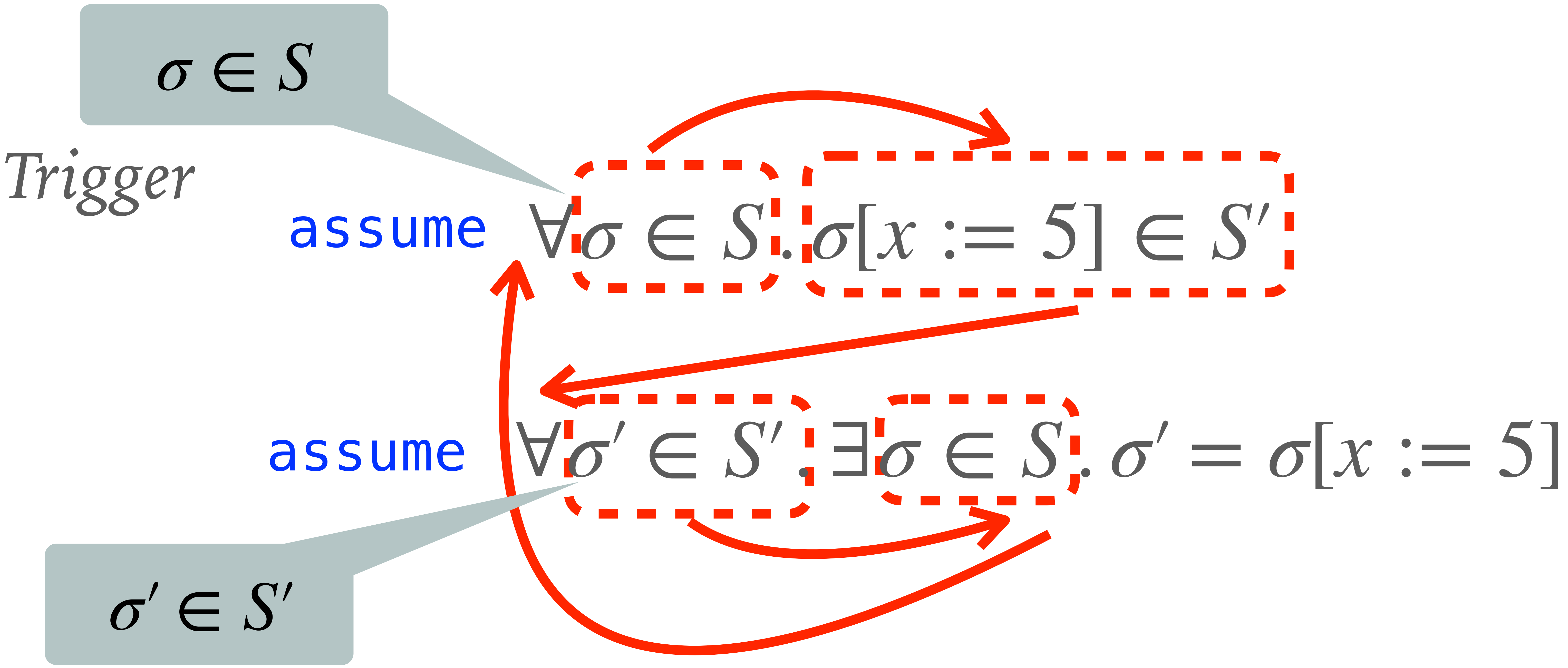
`assume` $\forall \sigma \in S.\ \sigma[x := 5] \in S'$

`assume` $\forall \sigma' \in S'.\ \exists \sigma \in S.\ \sigma' = \sigma[x := 5]$

$\sigma' \in S'$

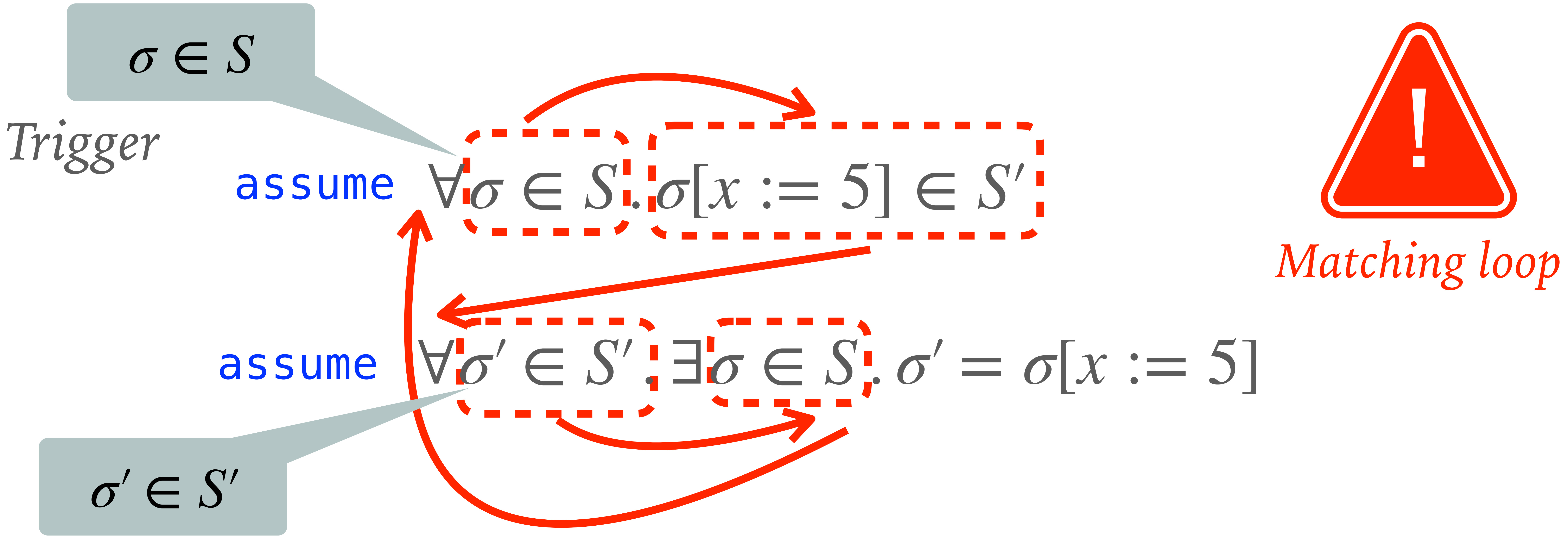
EXAMPLE REVISITED

- $C \triangleq x := 5$



EXAMPLE REVISITED

- $C \triangleq x := 5$



KEY IDEA OF THE ENCODING

- Track an upper bound and a lower bound of the sets of reachable states separately

KEY IDEA OF THE ENCODING

- Track an upper bound and a lower bound of the sets of reachable states separately

Trigger

$\sigma \in S$

`assume` $\forall \sigma \in S\,.\, \sigma[x := 5] \in S'$

$\sigma' \in S'$

`assume` $\forall \sigma' \in S'\,.\, \exists \sigma \in S\,.\, \sigma' = \sigma[x := 5]$

KEY IDEA OF THE ENCODING

➤ Track an upper bound and a lower bound of the sets of reachable states separately

Trigger

$\sigma \in S$

$\sigma' \in S'_\forall$

`assume` $\forall \sigma \in S \,.\, \sigma[x := 5] \in S'_\exists$

`assume` $\forall \sigma' \in S'_\forall \,.\, \exists \sigma \in S \,.\, \sigma' = \sigma[x := 5]$

KEY IDEA OF THE ENCODING

- Track an upper bound and a lower bound of the sets of reachable states separately

Trigger

$\sigma \in S$

$\sigma' \in S'_\forall$

`assume` $\forall \sigma \in S \,.\, \sigma[x := 5] \in S'_\exists$

`assume` $\forall \sigma' \in S'_\forall \,.\, \exists \sigma \in S \,.\, \sigma' = \sigma[x := 5]$

KEY IDEA OF THE ENCODING

- Track an upper bound and a lower bound of the sets of reachable states separately

Trigger

$\sigma \in S$

$\sigma' \in S'_\forall$

`assume` $\forall \sigma \in S \,.\, \sigma[x := 5] \in S'_\exists$

Does NOT match the trigger

`assume` $\forall \sigma' \in S'_\forall \,.\, \exists \sigma \in S \,.\, \sigma' = \sigma[x := 5]$

KEY IDEA OF THE ENCODING

- Track an upper bound and a lower bound of the sets of reachable states separately

Trigger

$\sigma \in S$

$\sigma' \in S'_\forall$

assume $\forall \sigma \in S \,.\, \sigma[x := 5] \in S'_\exists$

assume $\forall \sigma' \in S'_\forall \,.\, \exists \sigma \in S \,.\, \sigma' = \sigma[x := 5]$

```
// Postcondition
```

assert $(\forall \sigma' \in S' \,.\, \ldots) \wedge (\exists \sigma' \in S' \,.\, \ldots)$

KEY IDEA OF THE ENCODING

➤ Track an upper bound and a lower bound of the sets of reachable states separately

Trigger

$\sigma \in S$

$\sigma' \in S'_\forall$

```
assume
```
$\forall \sigma \in S \,.\, \sigma[x := 5] \in S'_\exists$

```
assume
```
$\forall \sigma' \in S'_\forall \,.\, \exists \sigma \in S \,.\, \sigma' = \sigma[x := 5]$

```
// Postcondition
assert
```
$(\forall \sigma' \in S'_\forall \,.\, \ldots) \land (\exists \sigma' \in S'_\exists \,.\, \ldots)$

KEY IDEA OF THE ENCODING

- Track an upper bound and a lower bound of the sets of reachable states separately

Trigger

$\sigma \in S$

$\sigma' \in S'_\forall$

```
assume
```
$\forall \sigma \in S \,.\, \sigma[x := 5] \in S'_\exists$

```
assume
```
$\forall \sigma' \in S'_\forall \,.\, \exists \sigma \in S \,.\, \sigma' = \sigma[x := 5]$

```
// Postcondition
assert
```
$(\forall \sigma' \in S'_\forall \,.\, \ldots) \wedge (\exists \sigma' \in S'_\exists \,.\, \ldots)$

Challenge 2 solved

EVALUATION

- 84 benchmarks adapted from the benchmarks of state-of-the-art verifiers (i.e. Descartes, HyPa, ORHLE, and PCSat)

EVALUATION

- 84 benchmarks adapted from the benchmarks of state-of-the-art verifiers (i.e. Descartes, HyPa, ORHLE, and PCSat)

Type of Hyperproperties	Files	Verification Time mean (s)	File LoC Mean	Lines of Annotations per LoC
$\forall^+$	18	2.1	111	0.004
$\exists^+$	14	8.7	59	0.056
$\forall^+\exists^+$	37	2.0	19	0.075
$\exists^+\forall^+$	15	1.6	25	0.067

EVALUATION

- 84 benchmarks adapted from the benchmarks of state-of-the-art verifiers (i.e. Descartes, HyPa, ORHLE, and PCSat)
- For 93% of the benchmarks, verification finished within 5s

Type of Hyperproperties	Files	Verification Time mean (s)	File LoC Mean	Lines of Annotations per LoC
$\forall^+$	18	2.1	111	0.004
$\exists^+$	14	8.7	59	0.056
$\forall^+\exists^+$	37	2.0	19	0.075
$\exists^+\forall^+$	15	1.6	25	0.067

EVALUATION

- 84 benchmarks adapted from the benchmarks of state-of-the-art verifiers (i.e. Descartes, HyPa, ORHLE, and PCSat)
- For 93% of the benchmarks, verification finished within 5s
- In general, a modest amount of proof annotations is needed

Type of Hyperproperties	Files	Verification Time mean (s)	File LoC Mean	Lines of Annotations per LoC
$\forall^+$	18	2.1	111	0.004
$\exists^+$	14	8.7	59	0.056
$\forall^+\exists^+$	37	2.0	19	0.075
$\exists^+\forall^+$	15	1.6	25	0.067

SUMMARY

- Hyper Hoare Logic: predicates over sets of states

SUMMARY

- Hyper Hoare Logic: predicates over sets of states
- This work: an automated verifier for Hyper Hoare Logic
  - By tracking sets of states via Viper encodings
  - By tracking an upper bound and a lower bound of the set of reachable states separately

SUMMARY

- Hyper Hoare Logic: predicates over sets of states
- This work: an automated verifier for Hyper Hoare Logic
  - By tracking sets of states via Viper encodings
  - By tracking an upper bound and a lower bound of the set of reachable states separately
- What else is in the paper:
  - Reasoning about errors
  - Reasoning about loops