Scaling Automatic Modular Verification

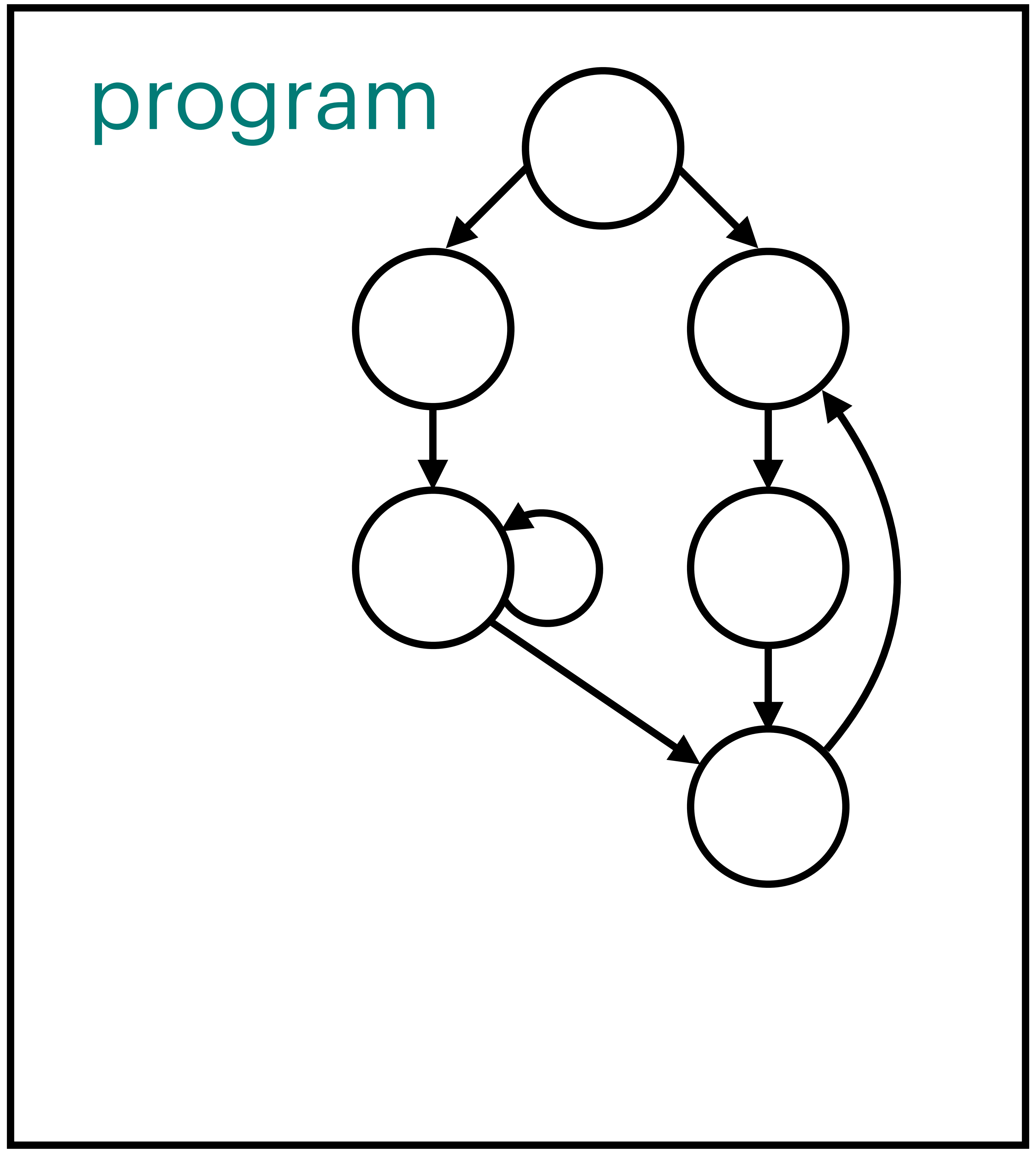
Lauren Pick

Automated Software Verification

verification problem

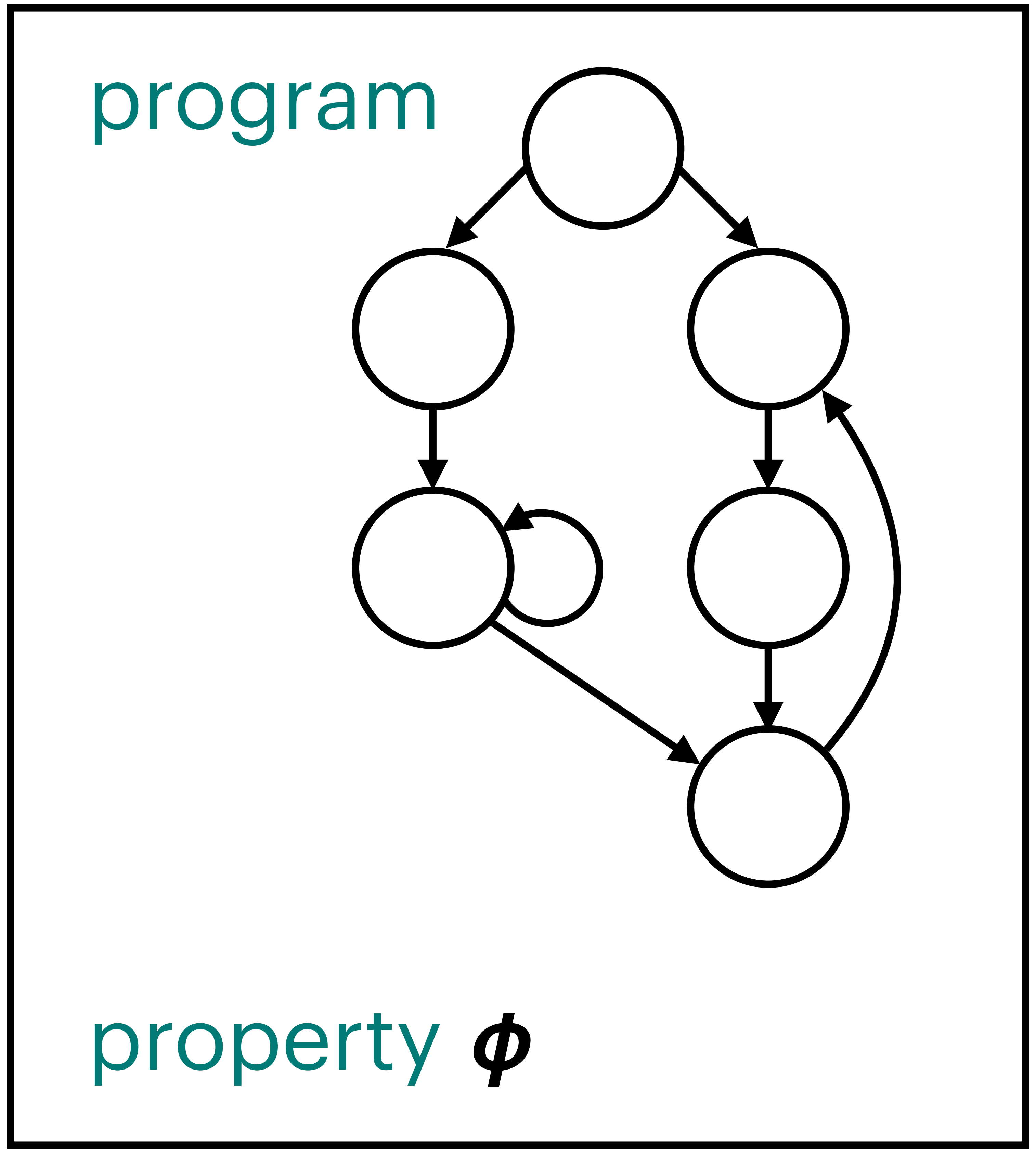
Automated Software Verification

verification problem

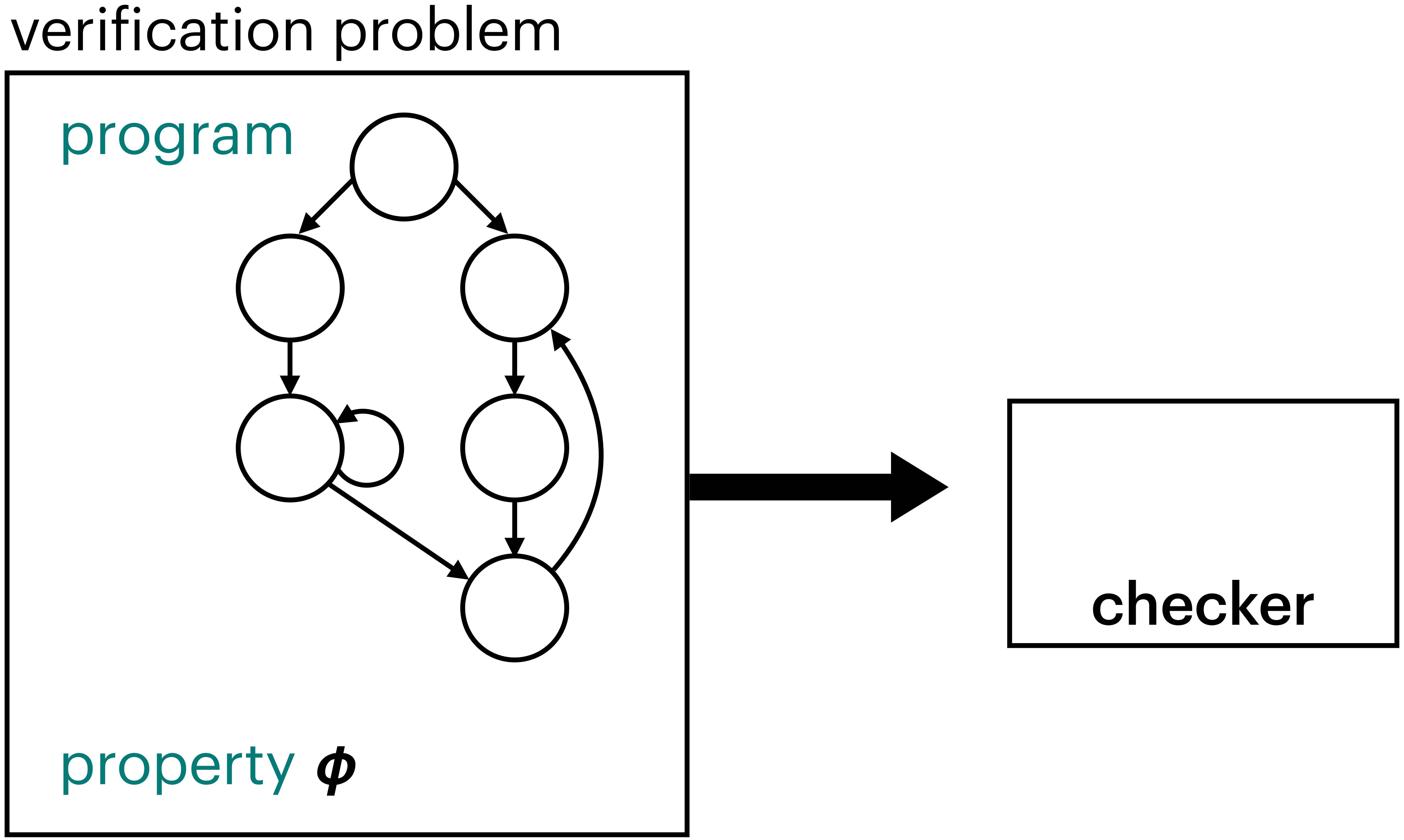


Automated Software Verification

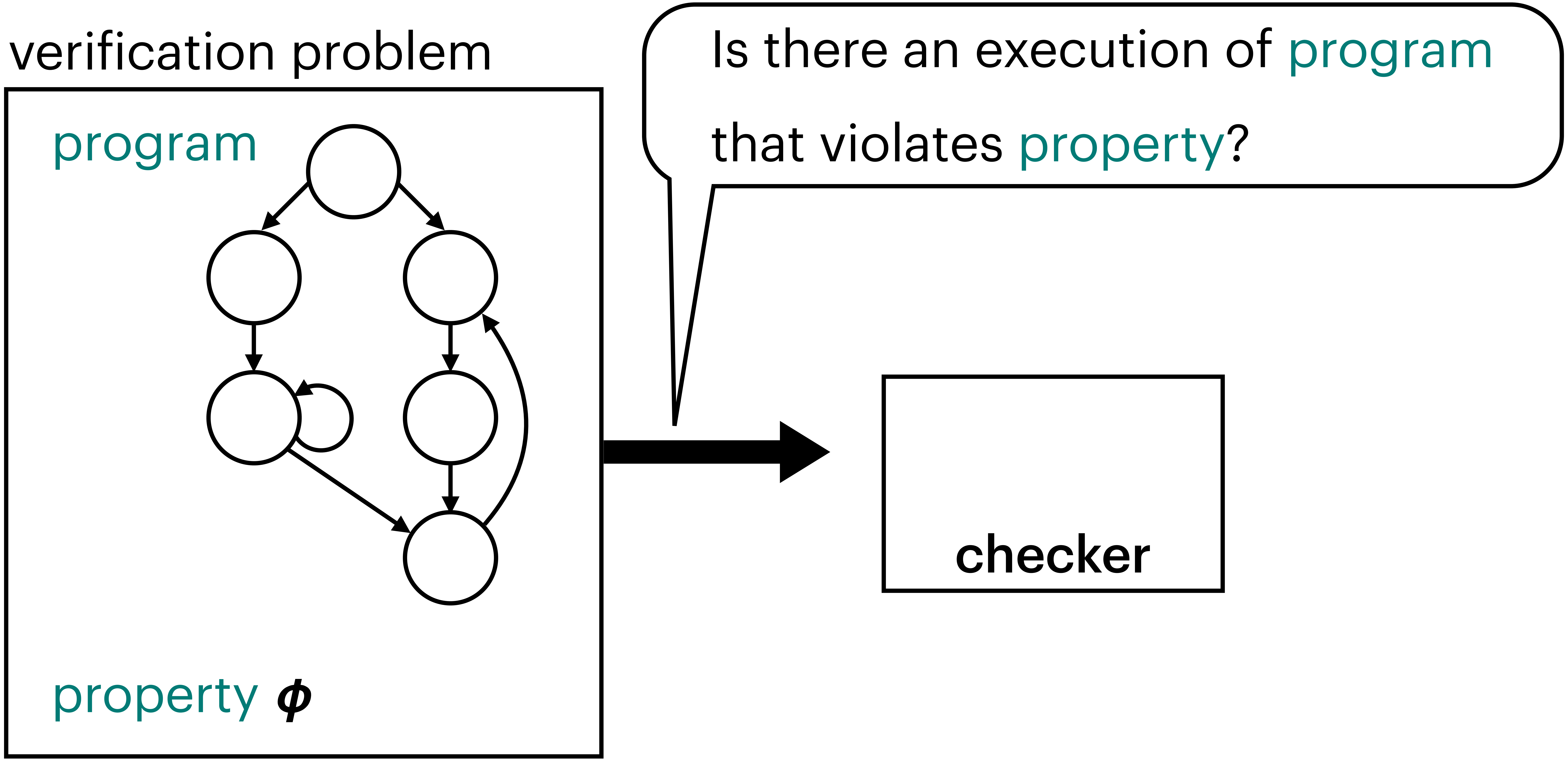
verification problem



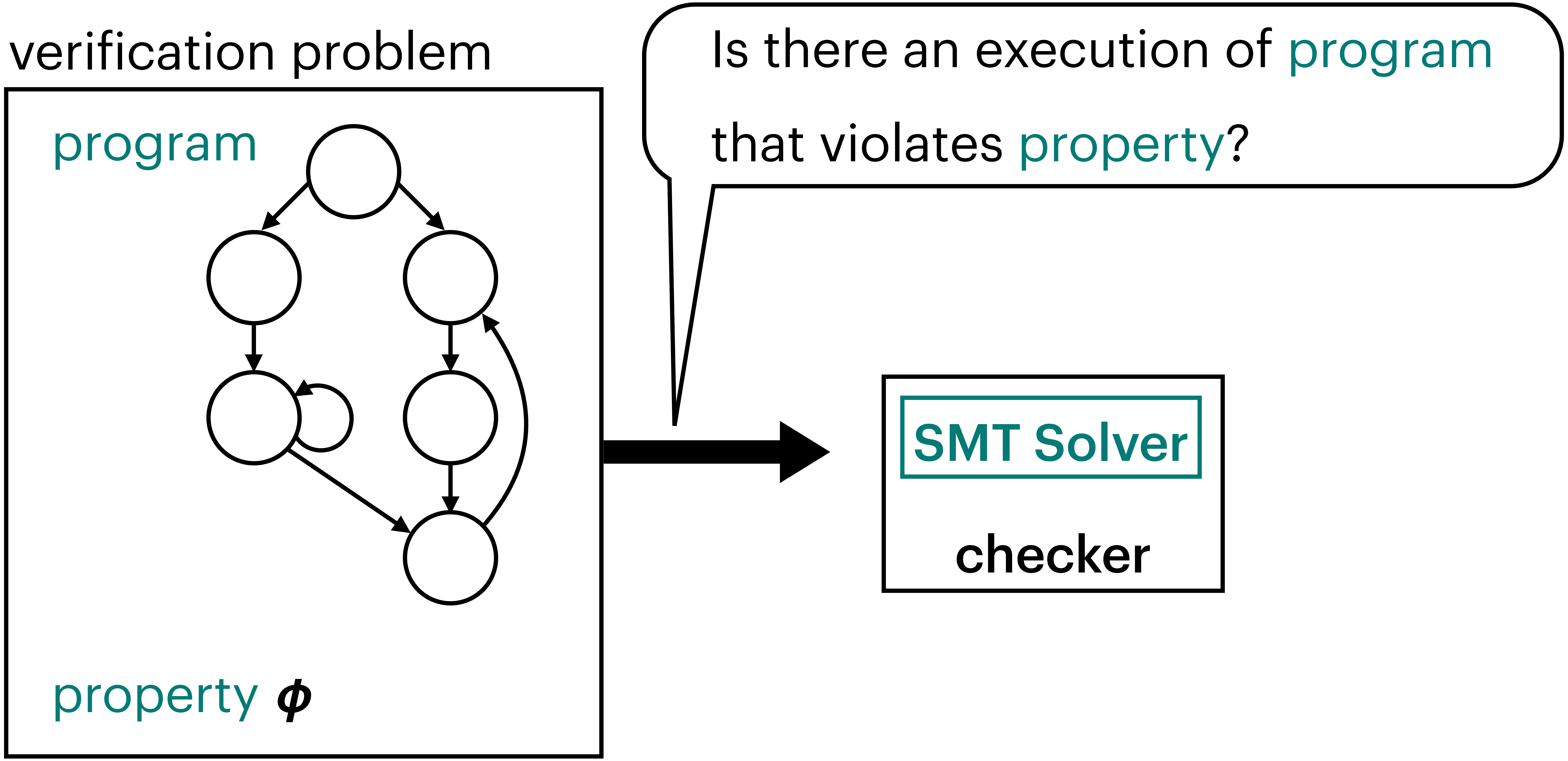
Automated Software Verification



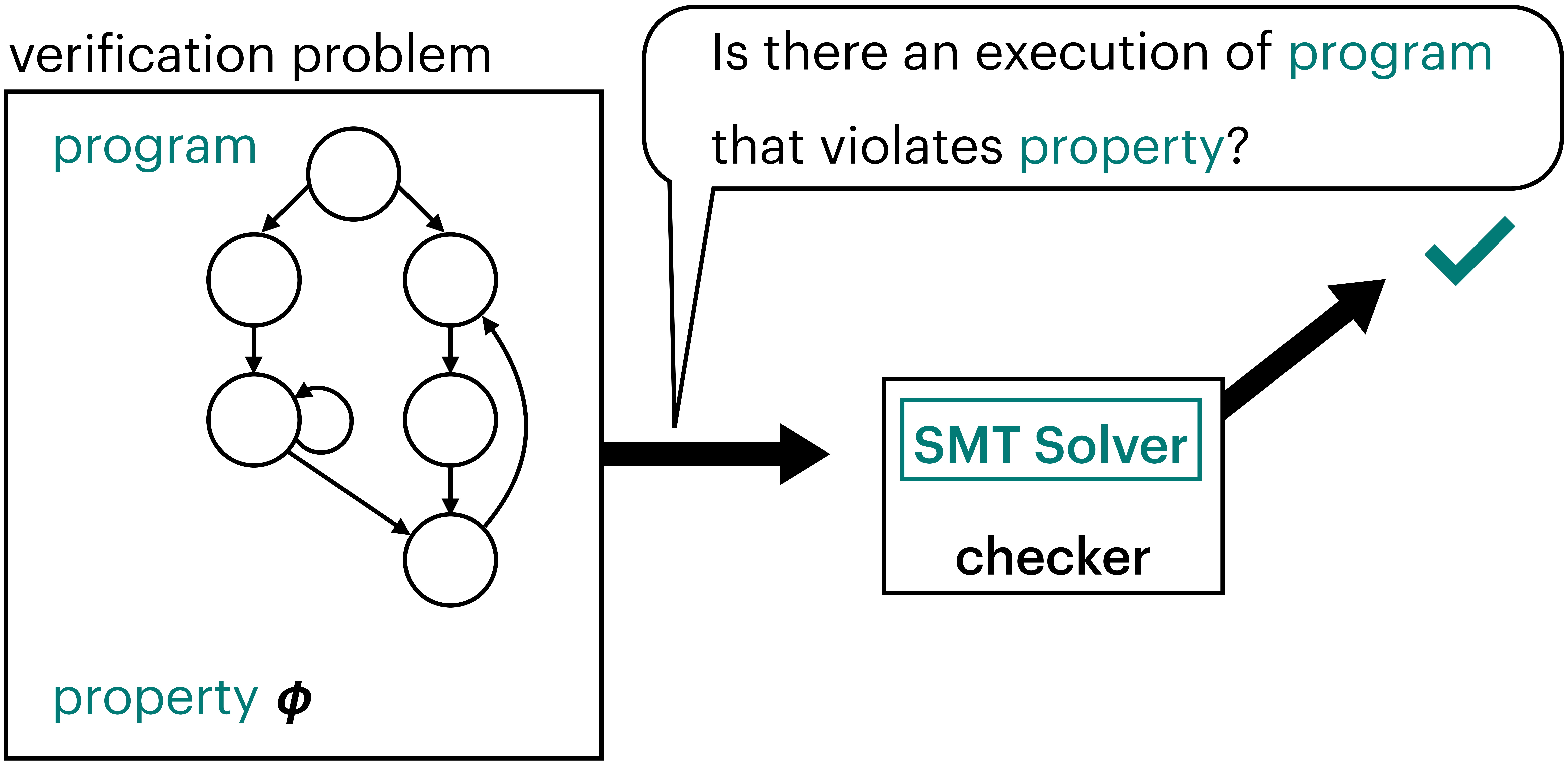
Automated Software Verification



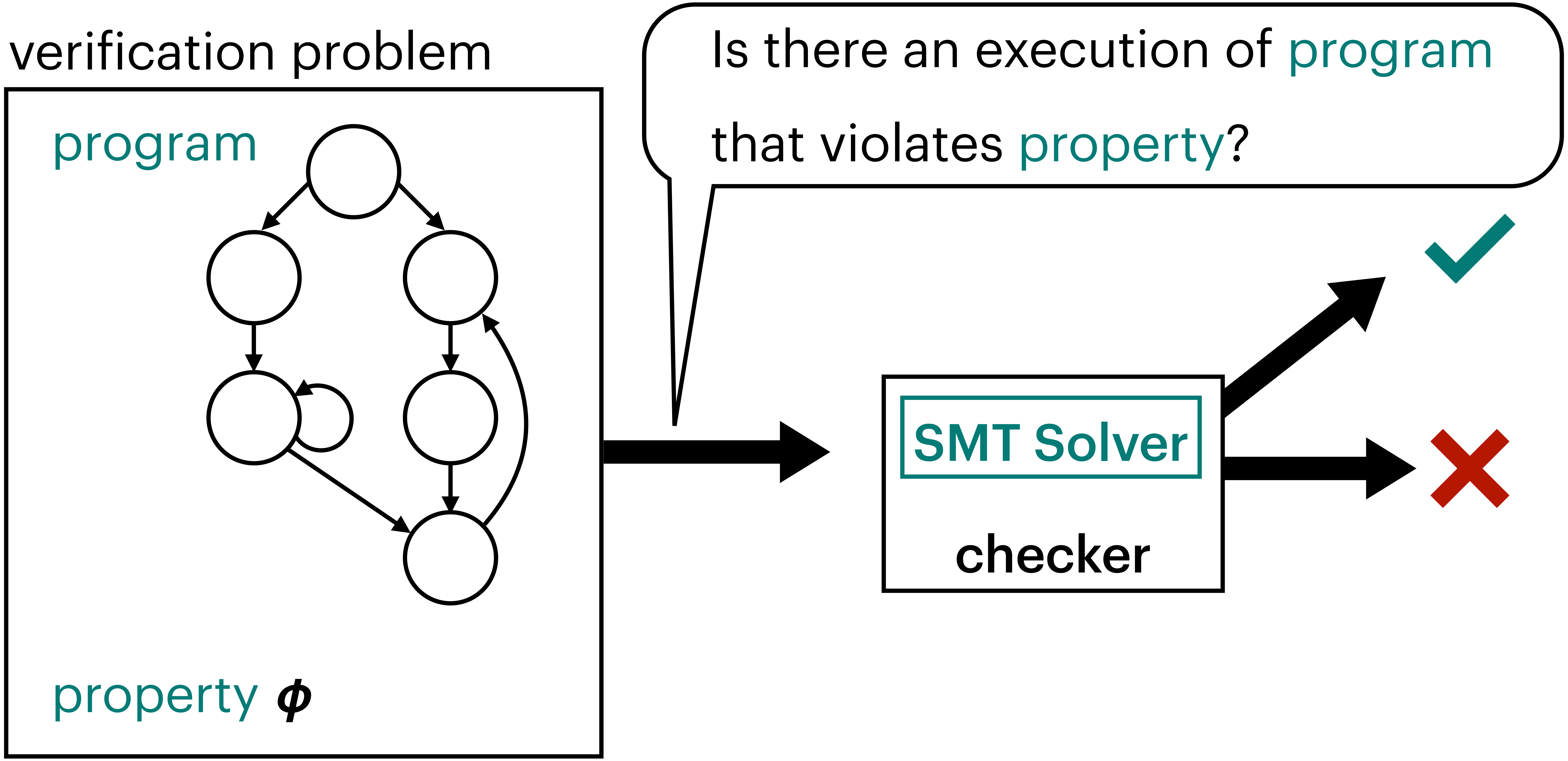
Automated Software Verification



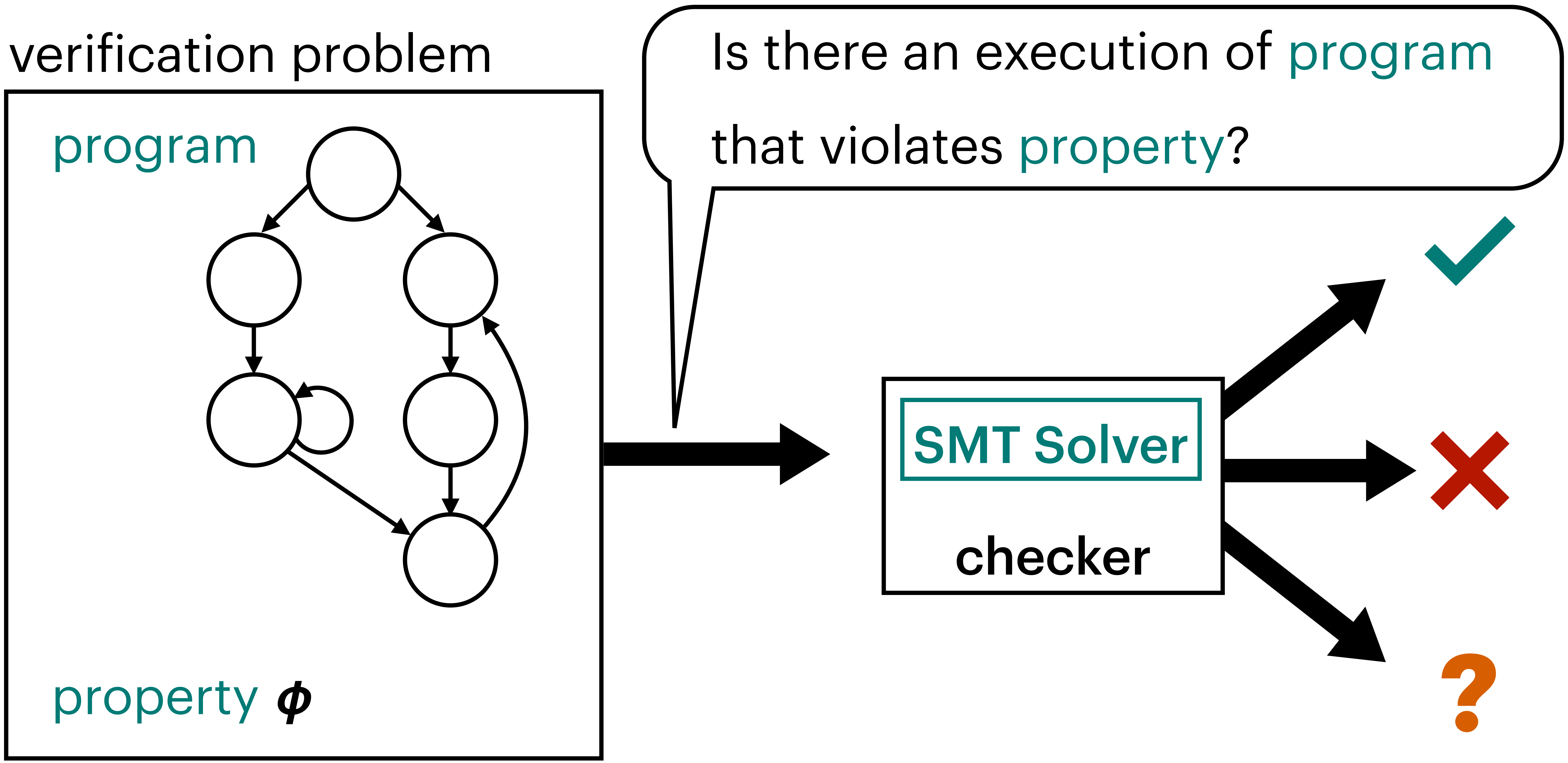
Automated Software Verification



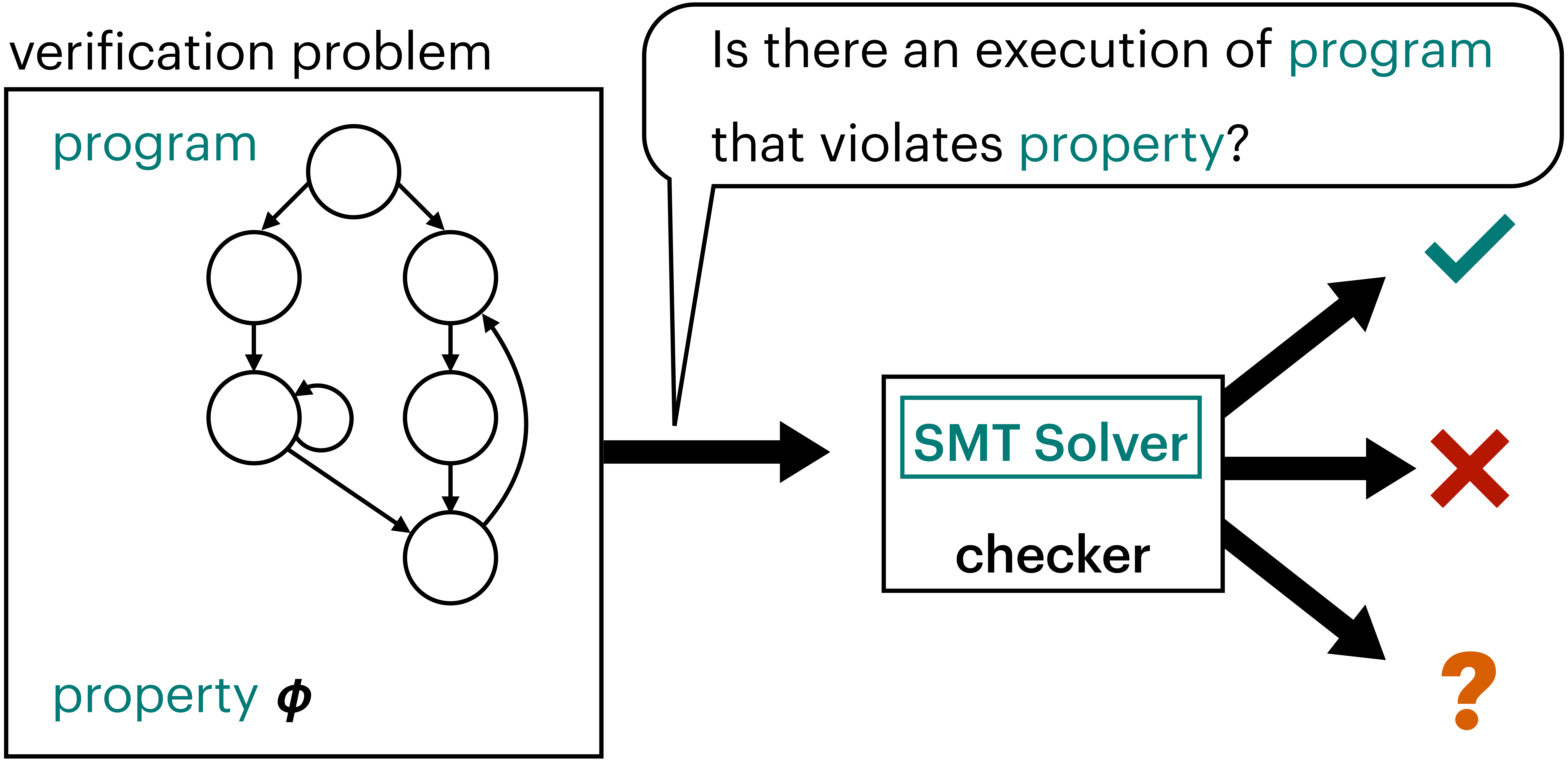
Automated Software Verification



Automated Software Verification



Automated Software Verification



Undecidable in general.

Satisfiability Modulo Theories (SMT) Solvers

SMT Solver

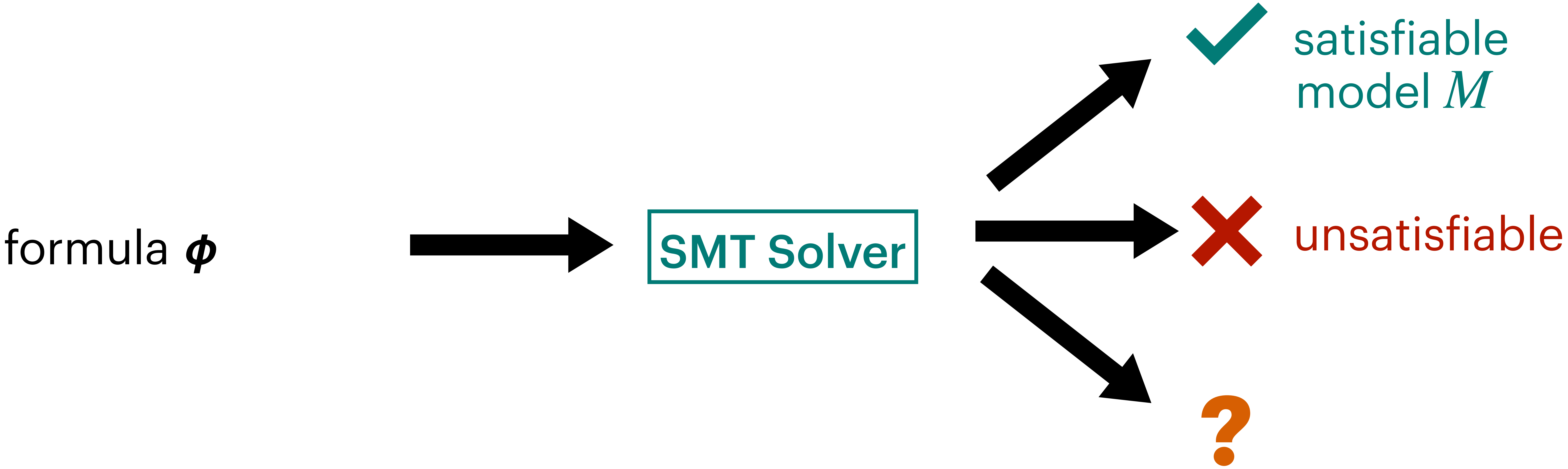
Satisfiability Modulo Theories (SMT) Solvers

formula $\boldsymbol{\phi}$ → **SMT Solver**

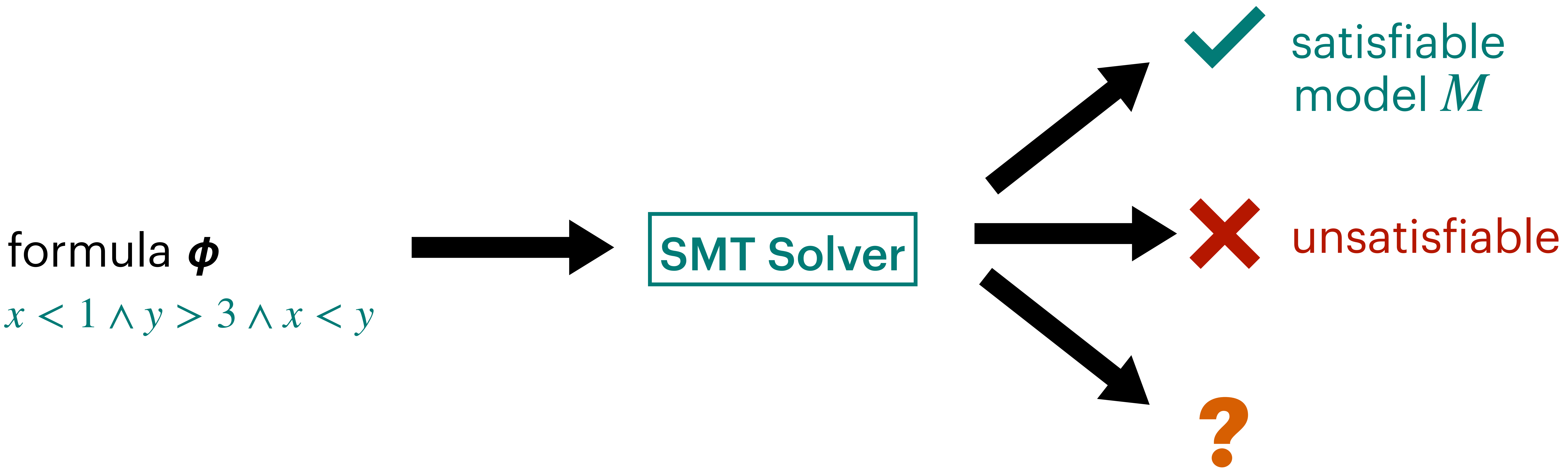
Satisfiability Modulo Theories (SMT) Solvers

Satisfiability Modulo Theories (SMT) Solvers

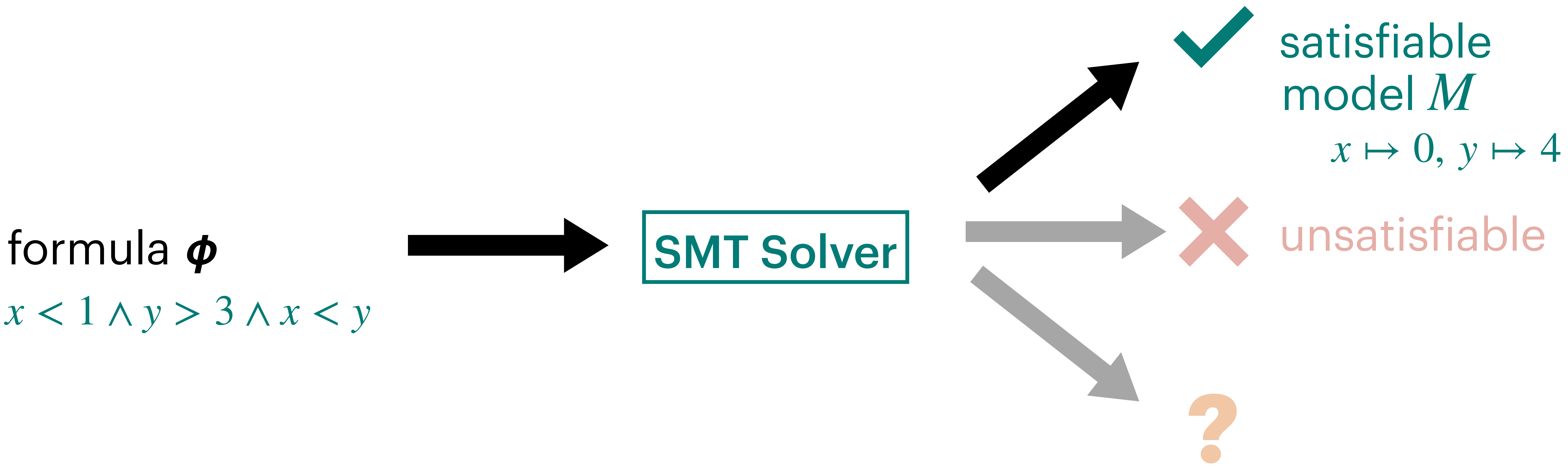
Satisfiability Modulo Theories (SMT) Solvers



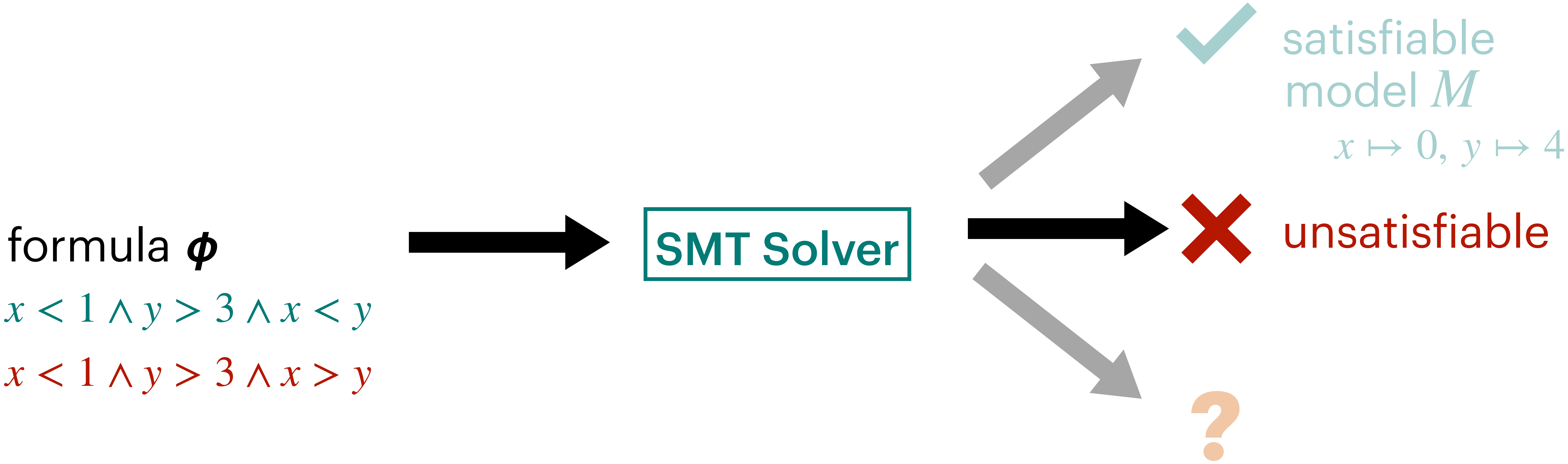
Satisfiability Modulo Theories (SMT) Solvers



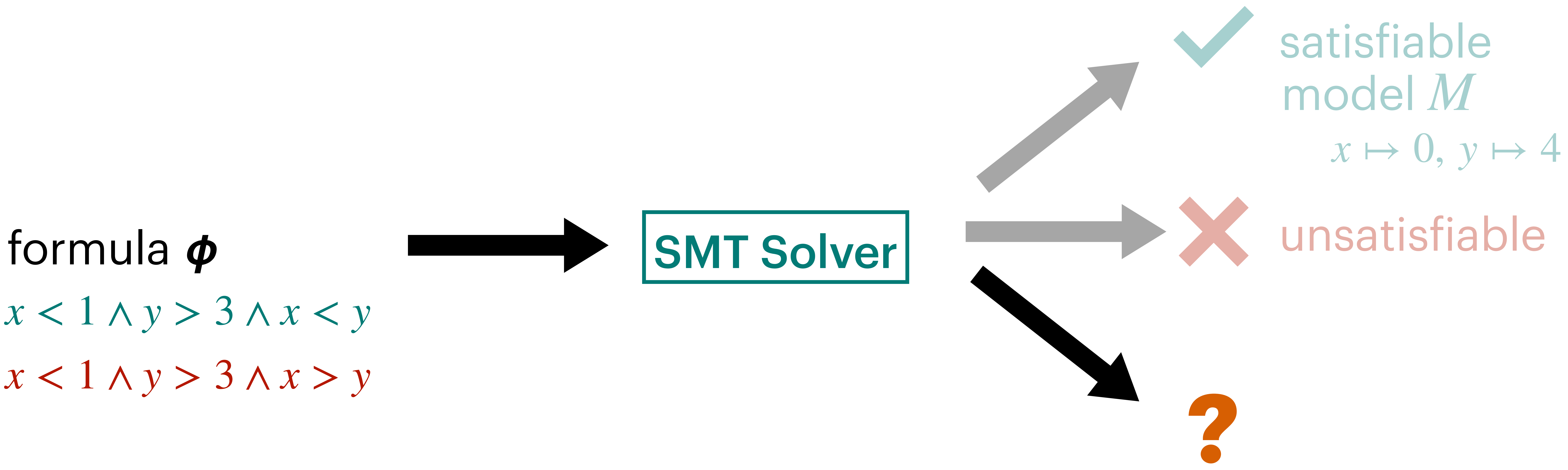
Satisfiability Modulo Theories (SMT) Solvers



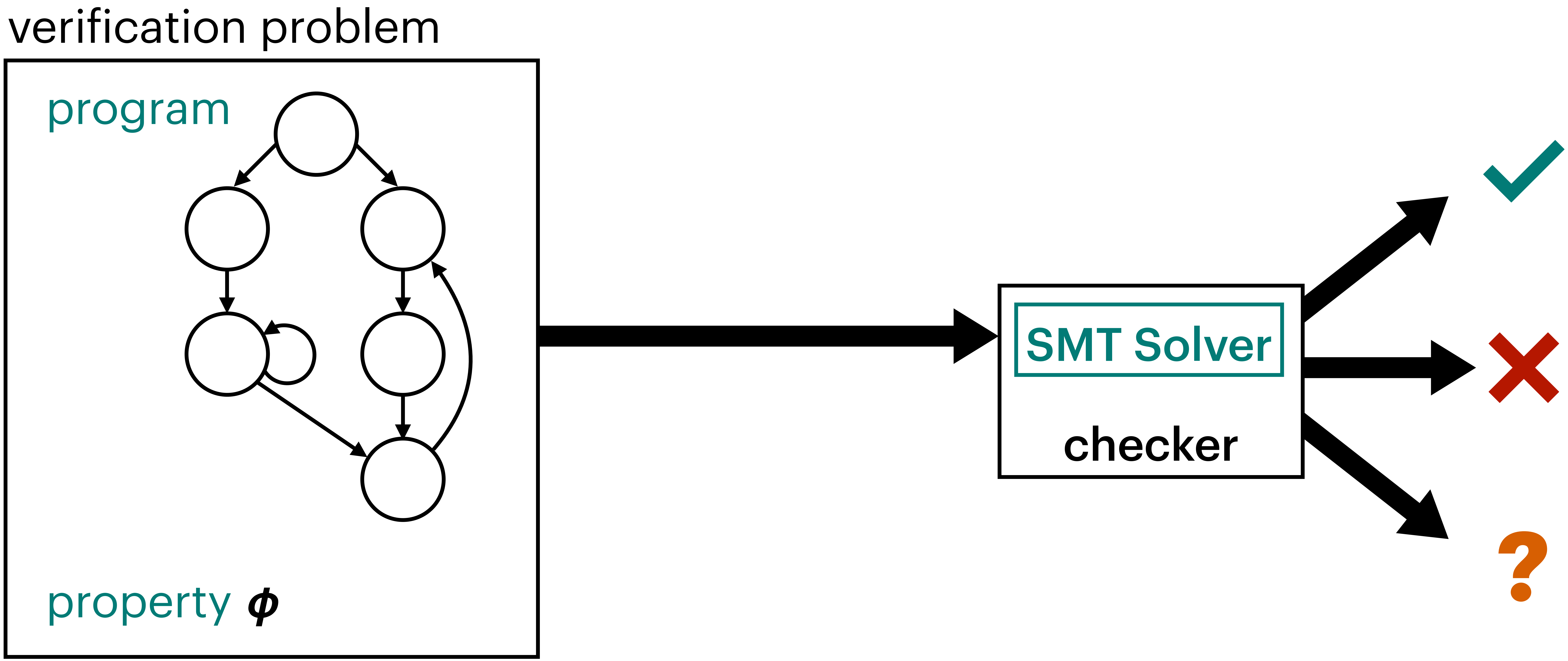
Satisfiability Modulo Theories (SMT) Solvers



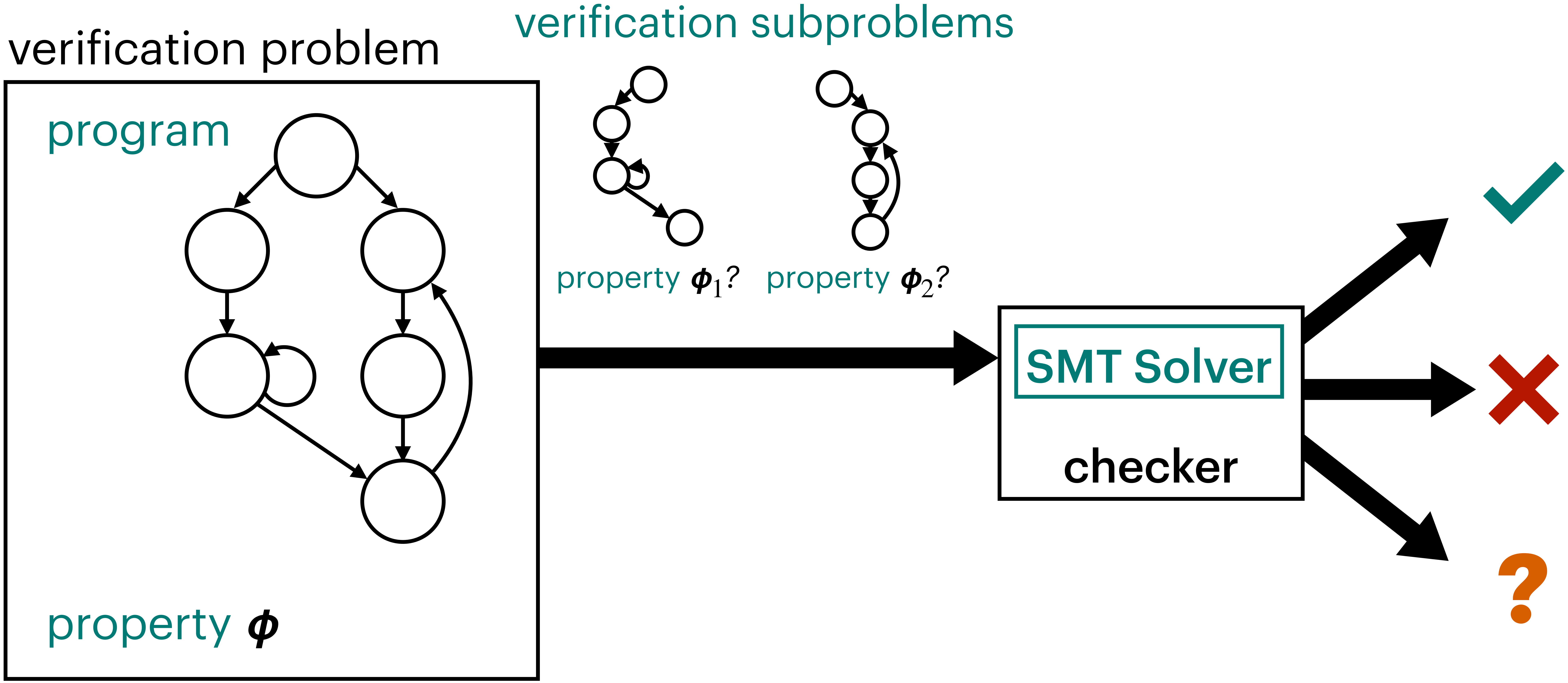
Satisfiability Modulo Theories (SMT) Solvers



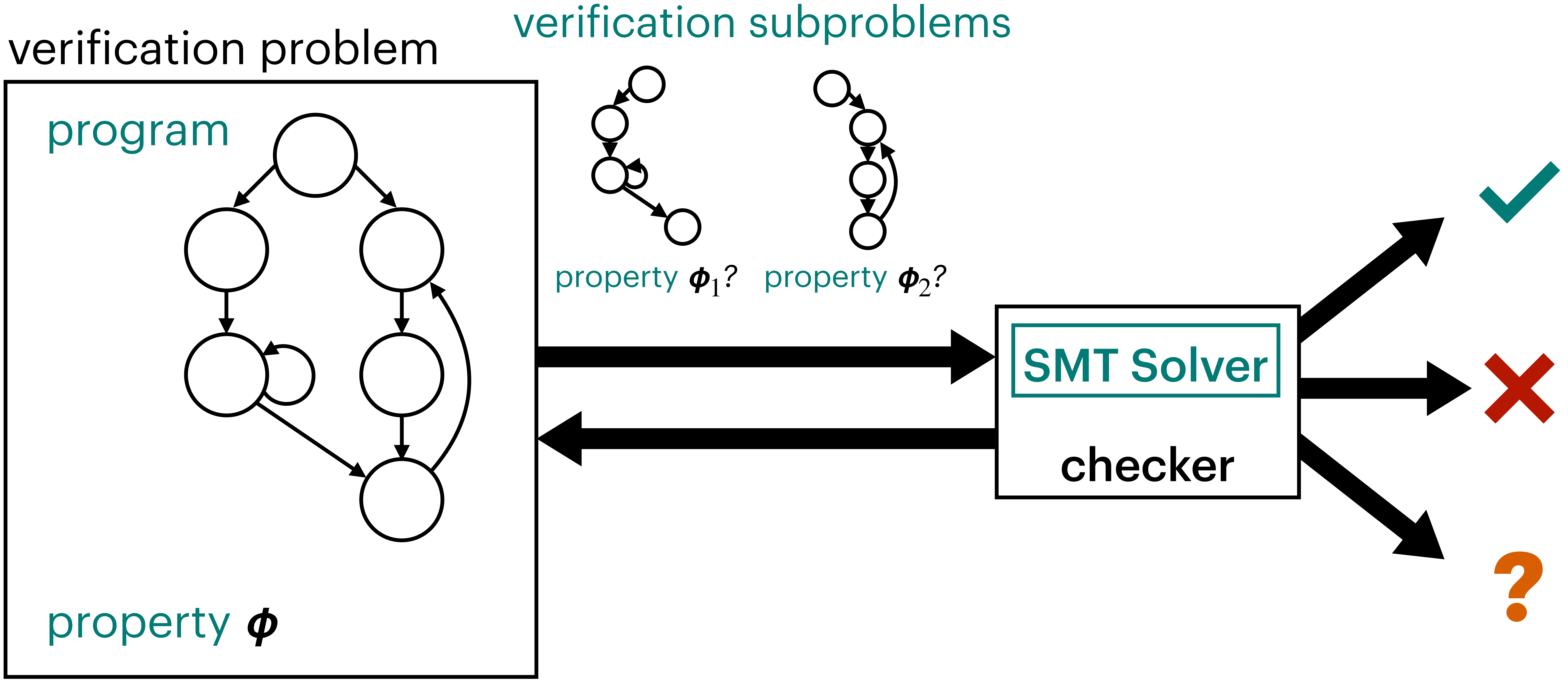
Automated Modular Verification



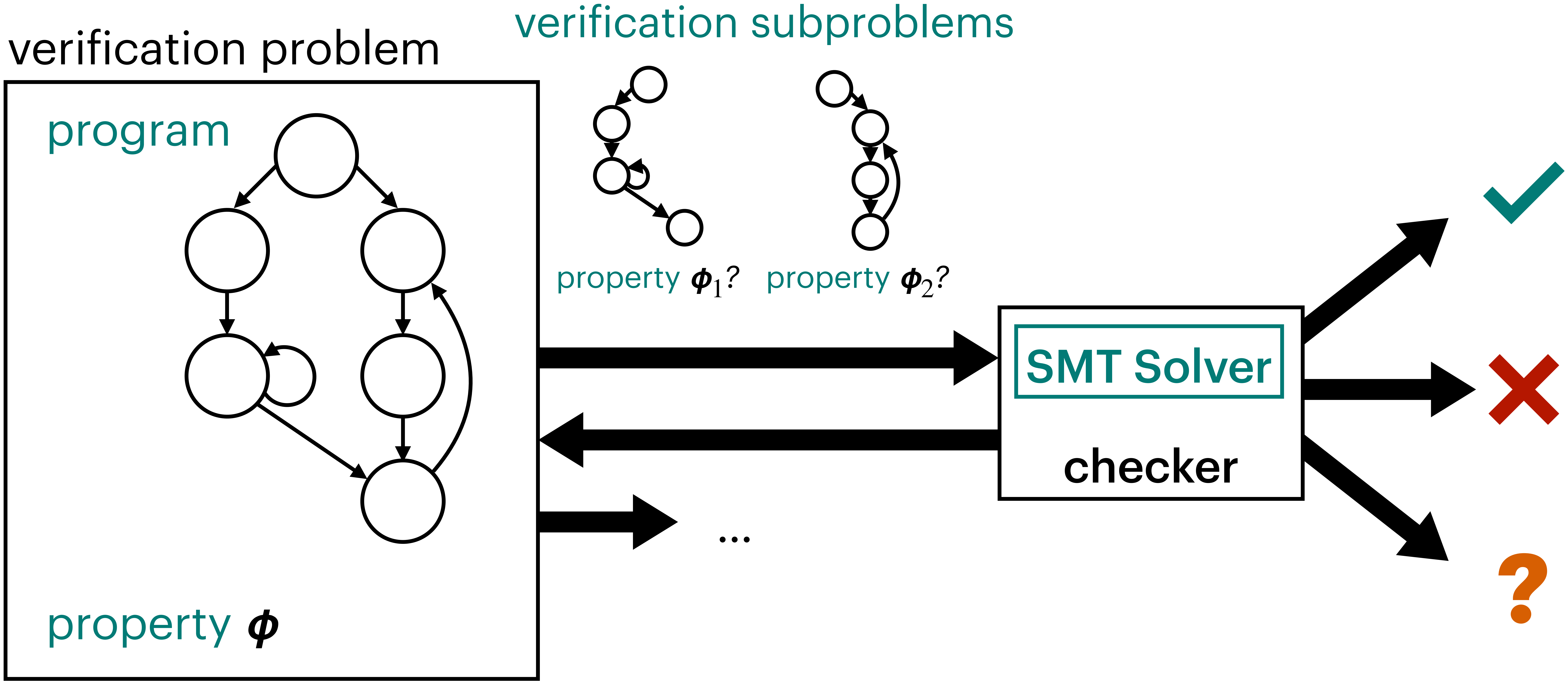
Automated Modular Verification



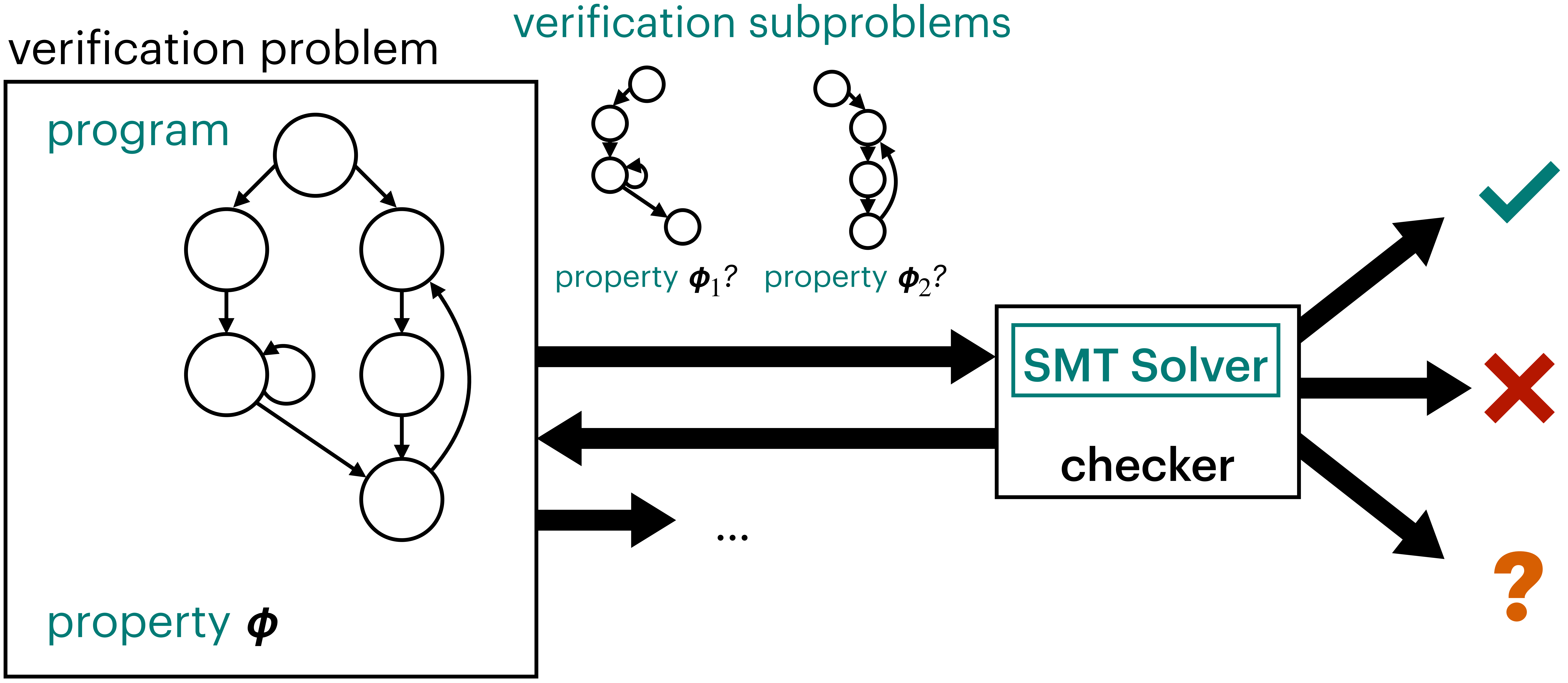
Automated Modular Verification



Automated Modular Verification



Automated Modular Verification



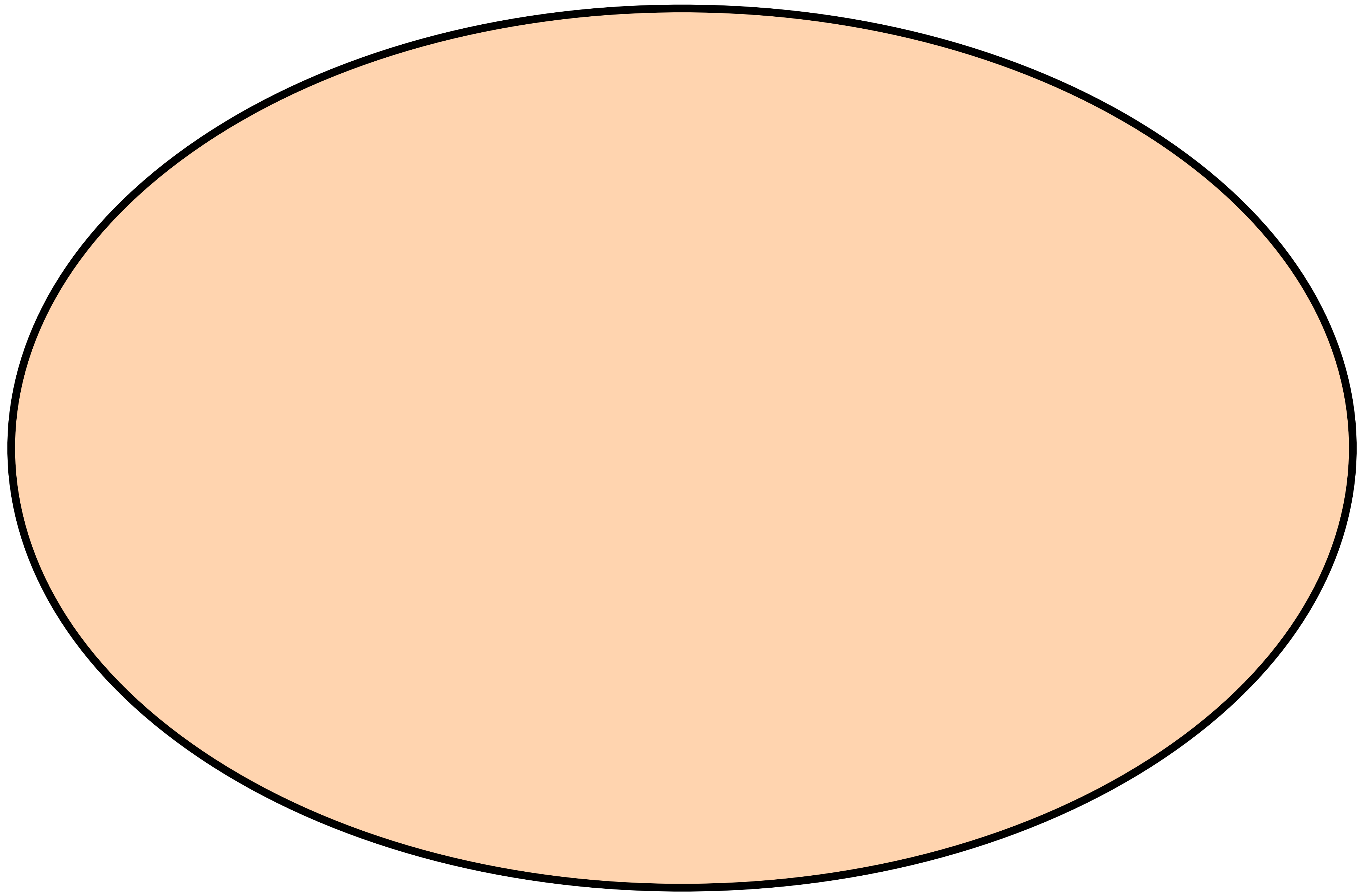
Verification subproblems can involve discovery of inductive invariants

Verification of Transition Systems

For a transition system $(S, T, Init)$:

Verification of Transition Systems

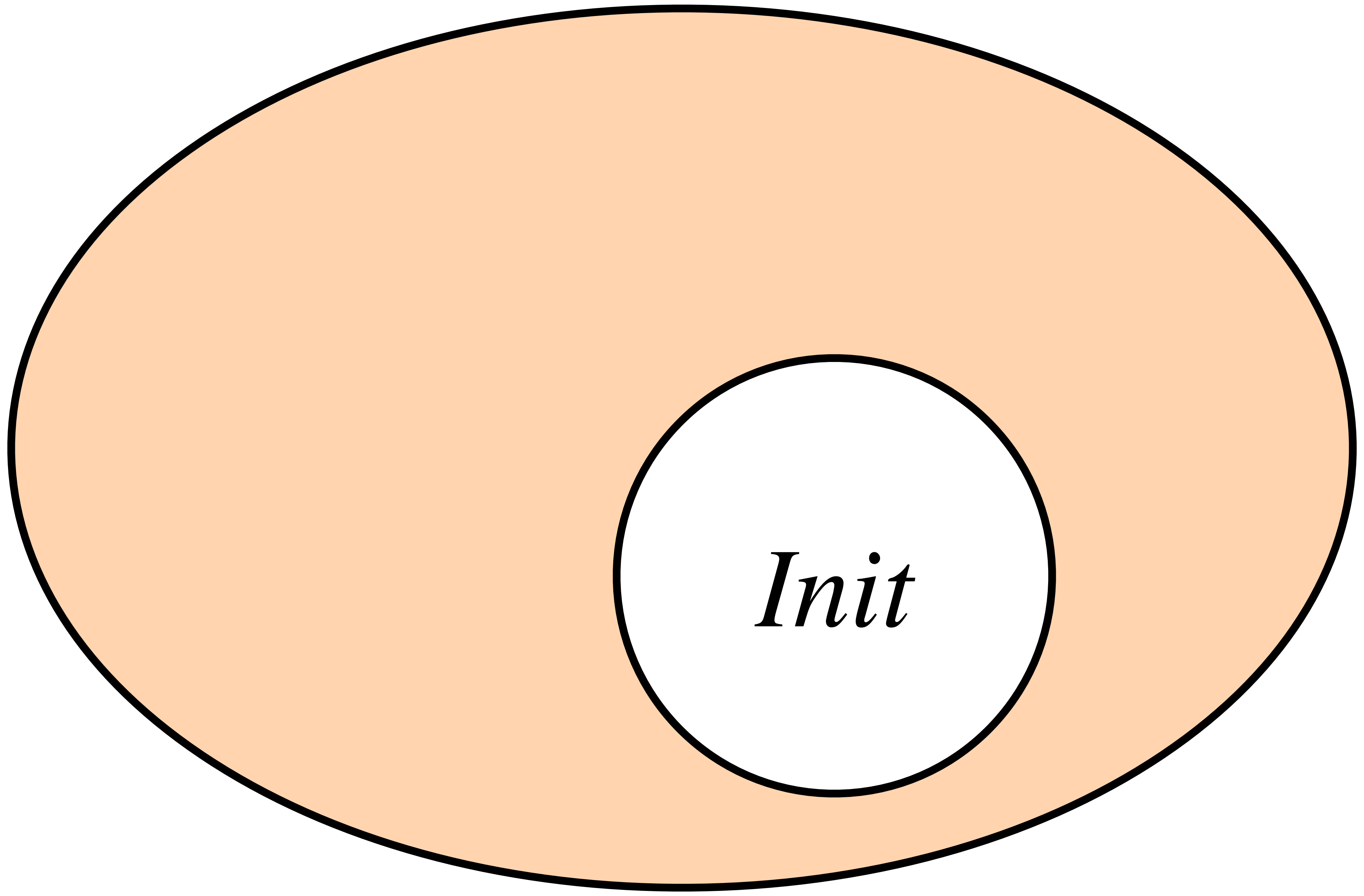
For a transition system $(S, T, Init)$:



States S

Verification of Transition Systems

For a transition system $(S, T, Init)$:

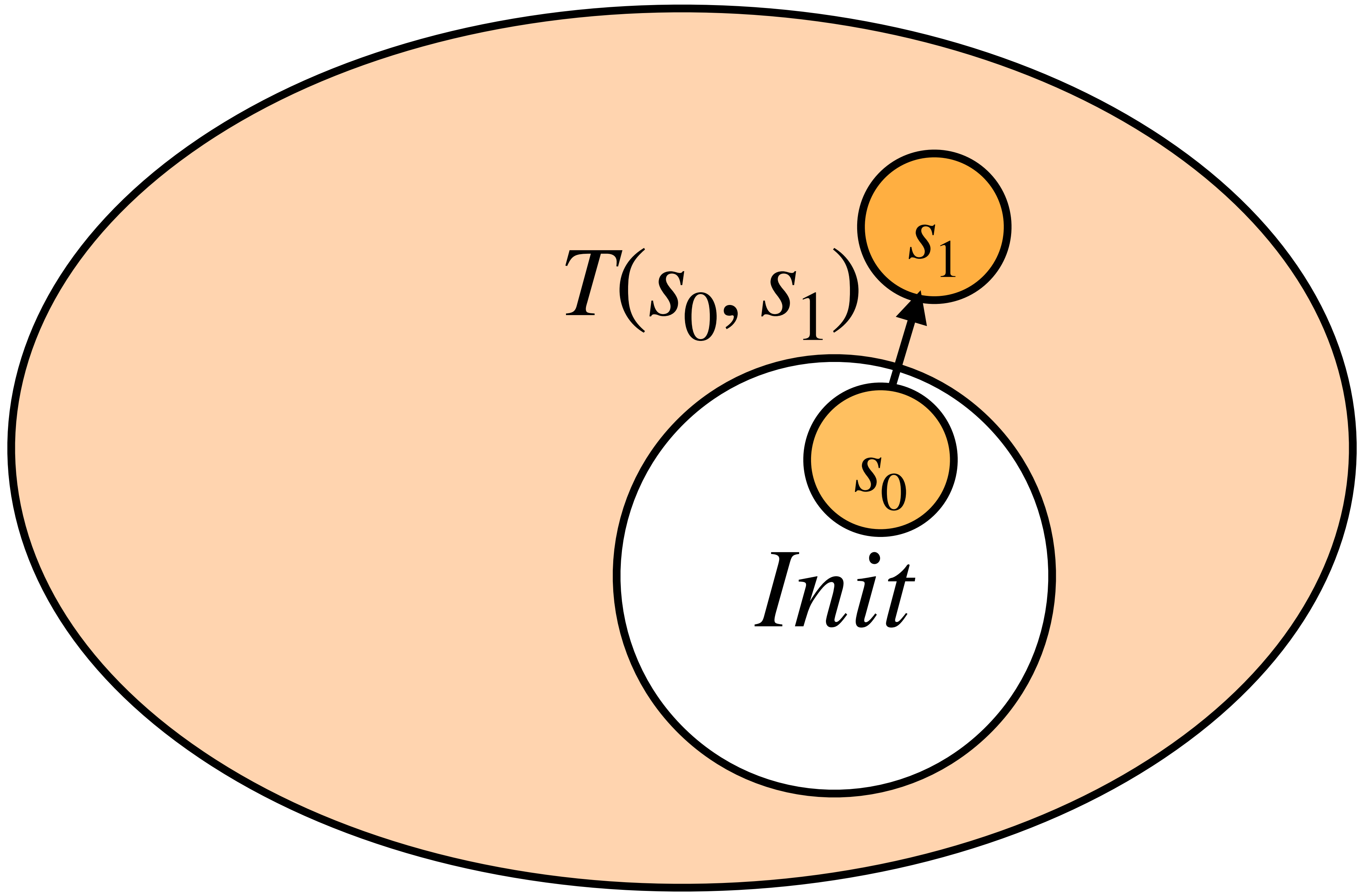


States S

Initial states $Init \subseteq S$

Verification of Transition Systems

For a transition system $(S, T, Init)$:



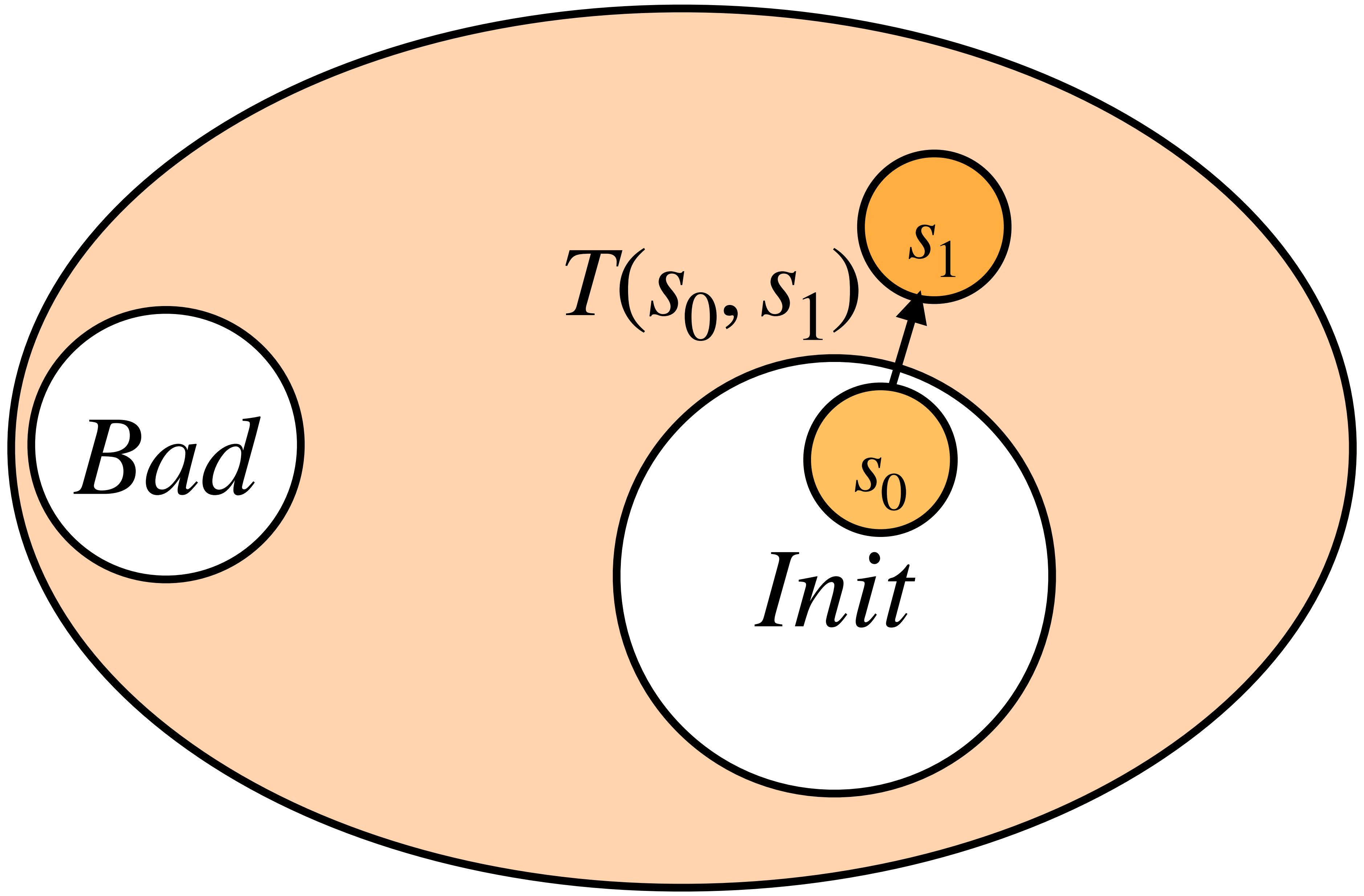
States S

Initial states $Init \subseteq S$

Transition relation T

Verification of Transition Systems

For a transition system $(S, T, Init)$:



States S

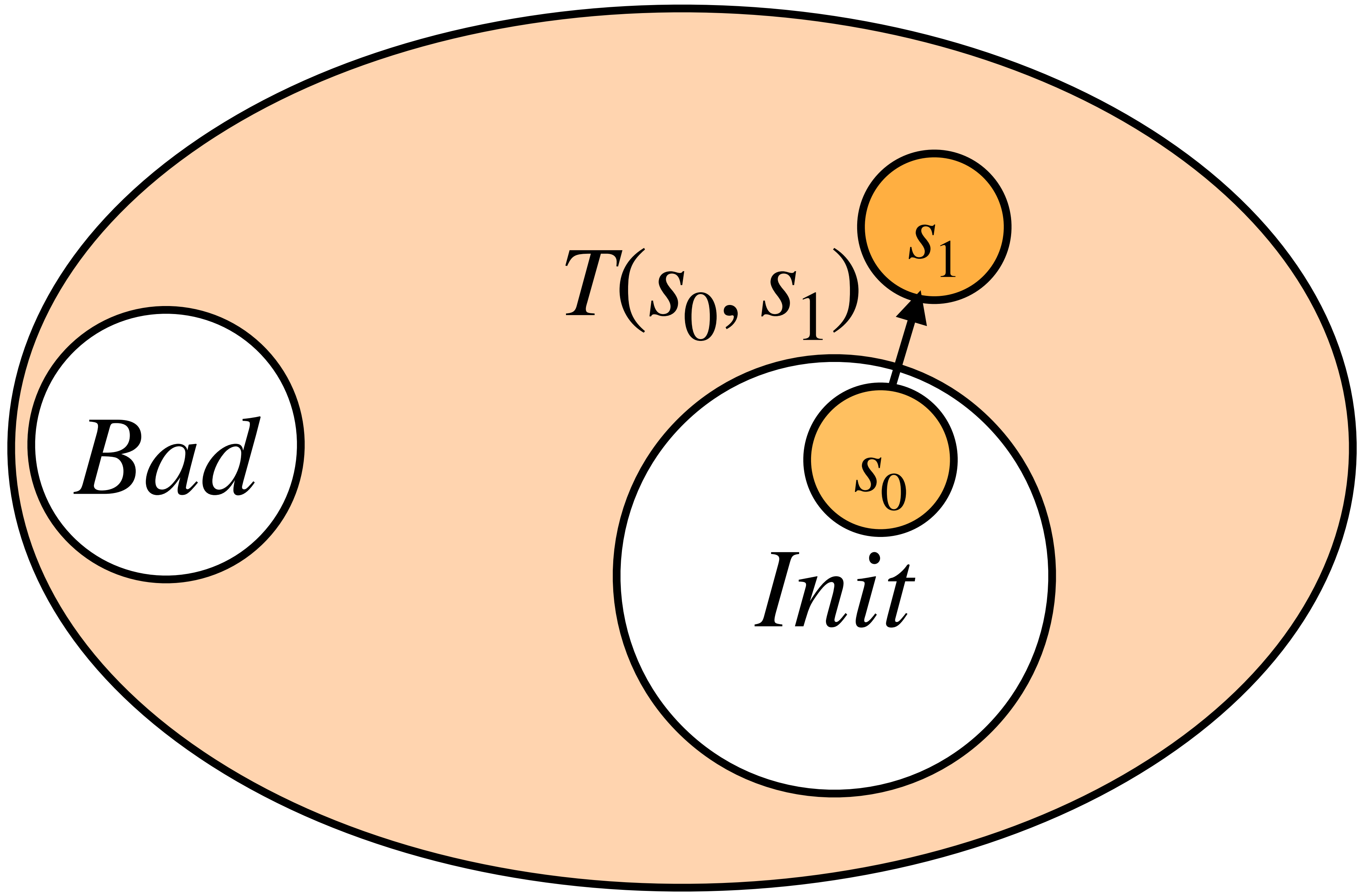
Initial states $Init \subseteq S$

Transition relation T

Bad states $Bad \subseteq S$

Verification of Transition Systems

For a transition system $(S, T, Init)$:



States S

Initial states $Init \subseteq S$

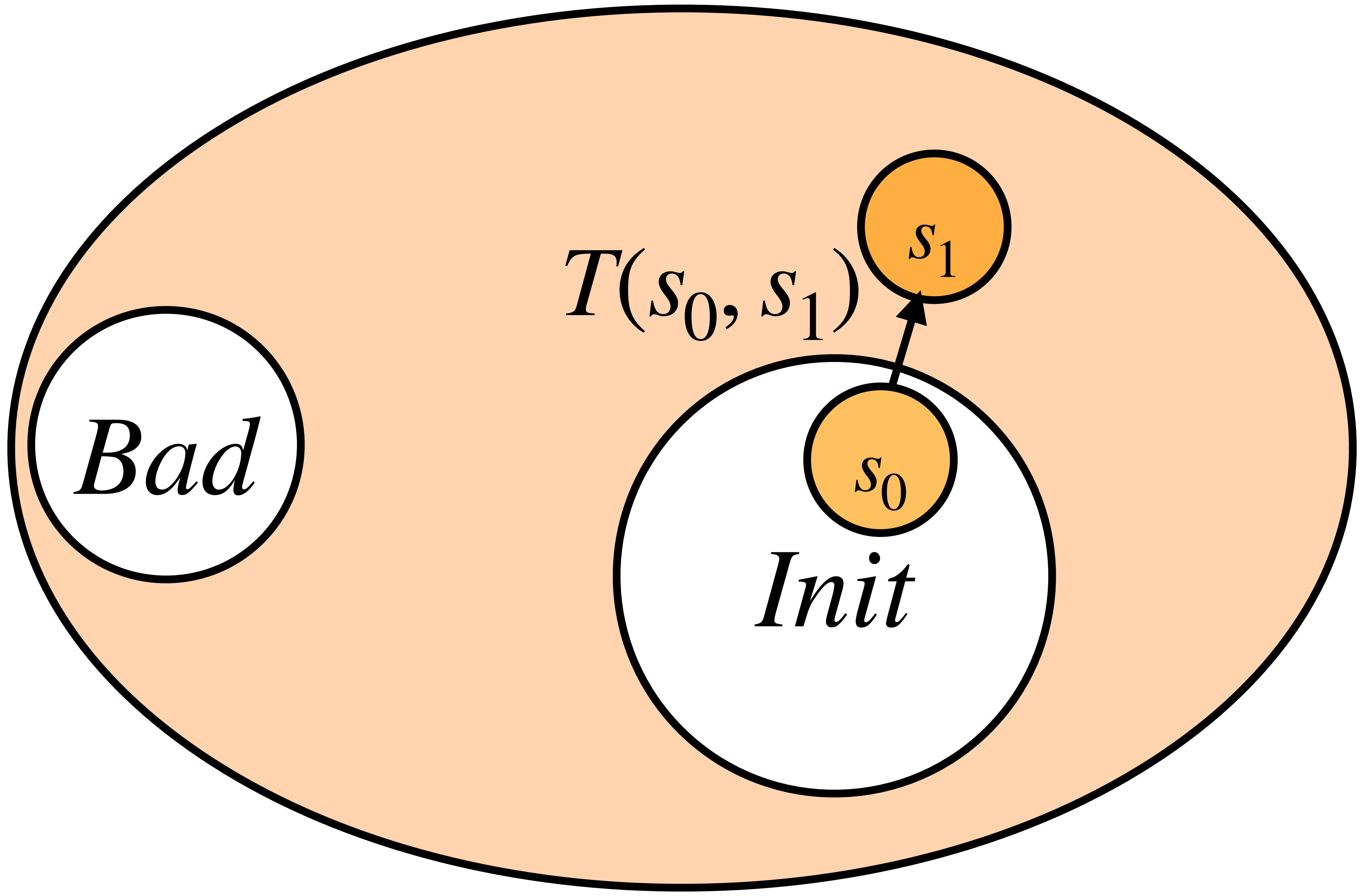
Transition relation T

Bad states $Bad \subseteq S$

Want to prove *safety property* that no Bad states are reachable from $Init$ states

Inductive Invariants for Transition Systems

For a transition system $(S, T, Init)$:



States S

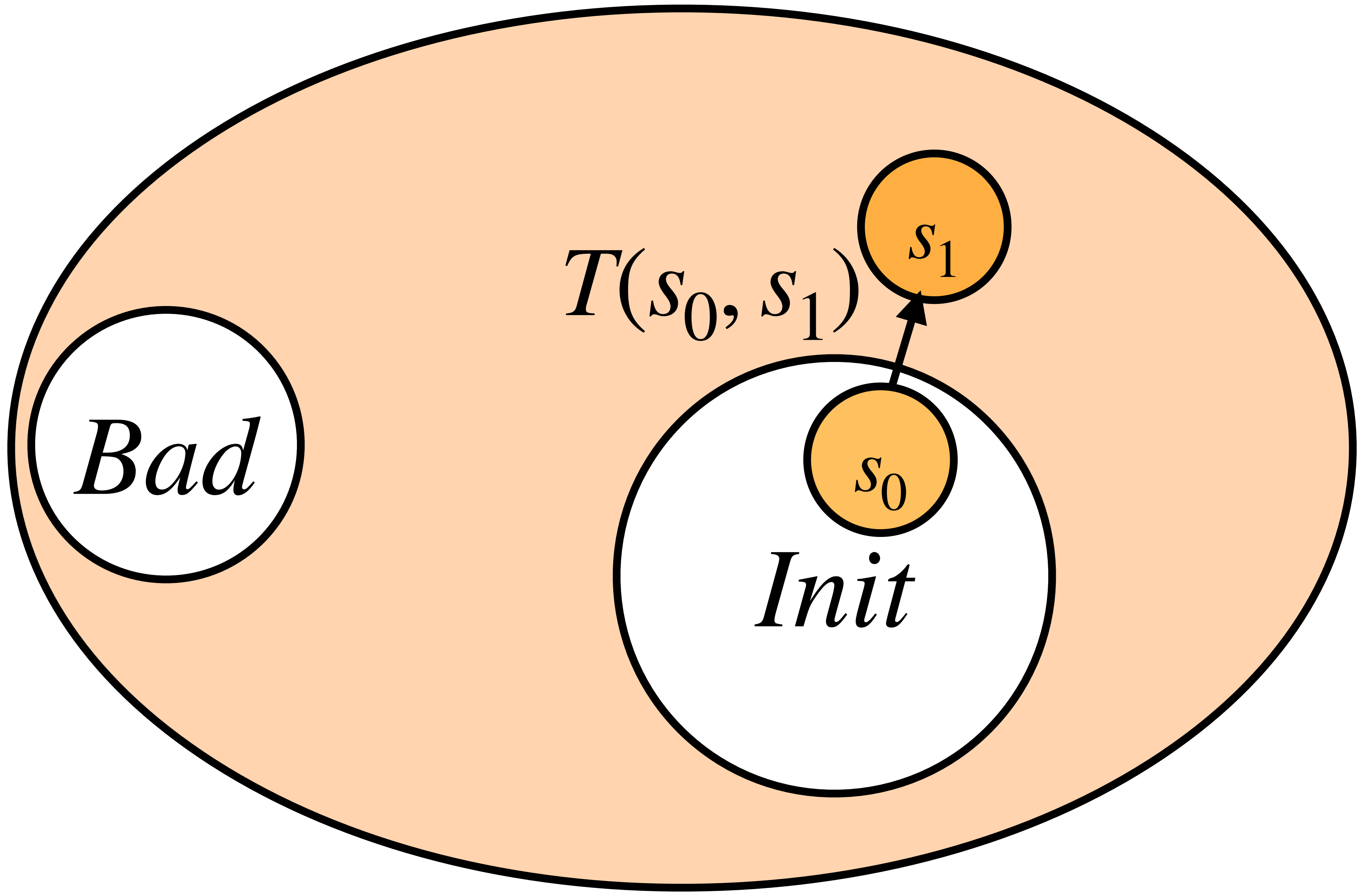
Initial states $Init \subseteq S$

Transition relation T

Bad states $Bad \subseteq S$

Inductive Invariants for Transition Systems

For a transition system $(S, T, Init)$:



States S

Initial states $Init \subseteq S$

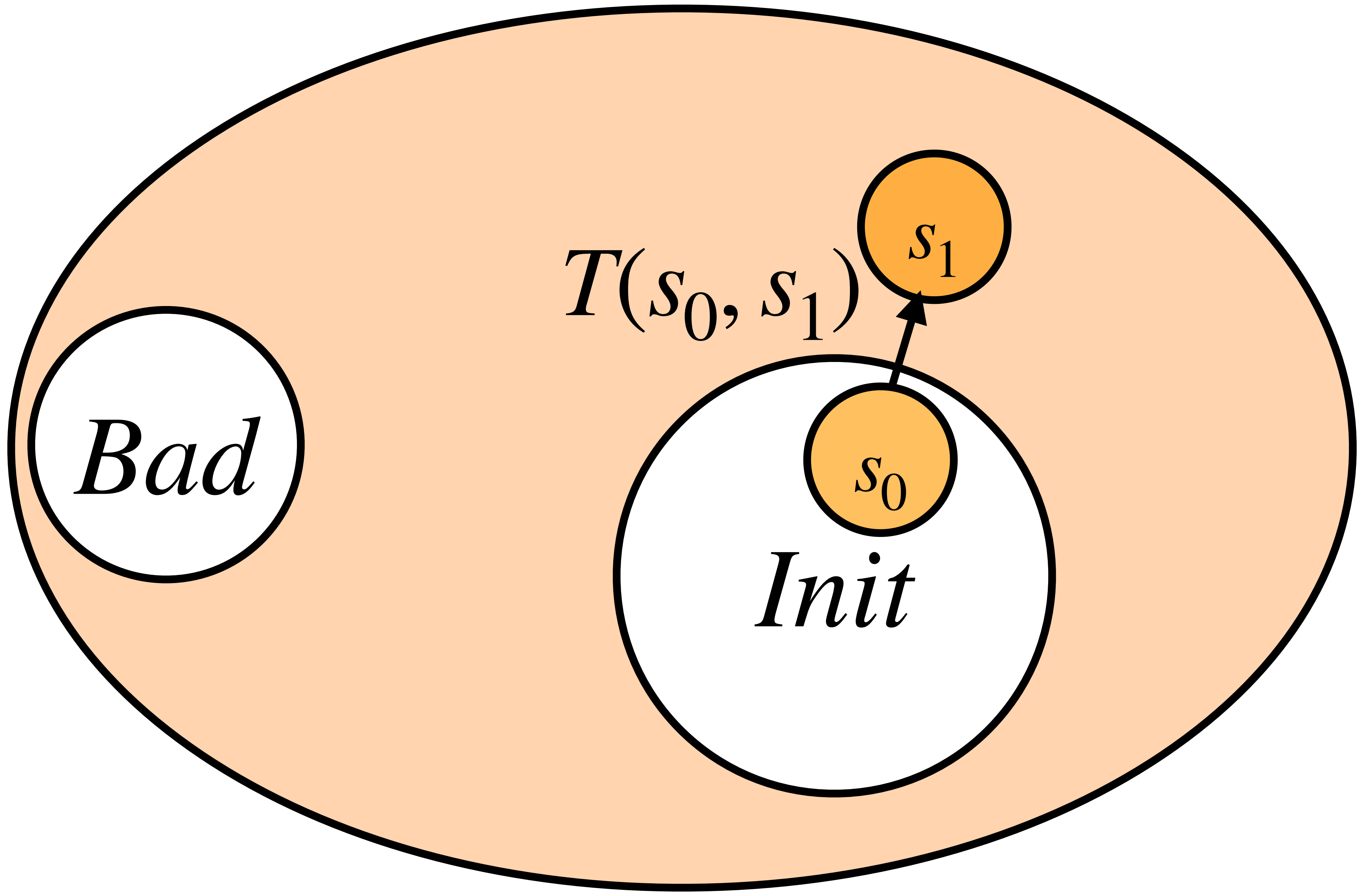
Transition relation T

Bad states $Bad \subseteq S$

Formula I is an inductive invariant for the system if the following hold:

Inductive Invariants for Transition Systems

For a transition system $(S, T, Init)$:



States S

Initial states $Init \subseteq S$

Transition relation T

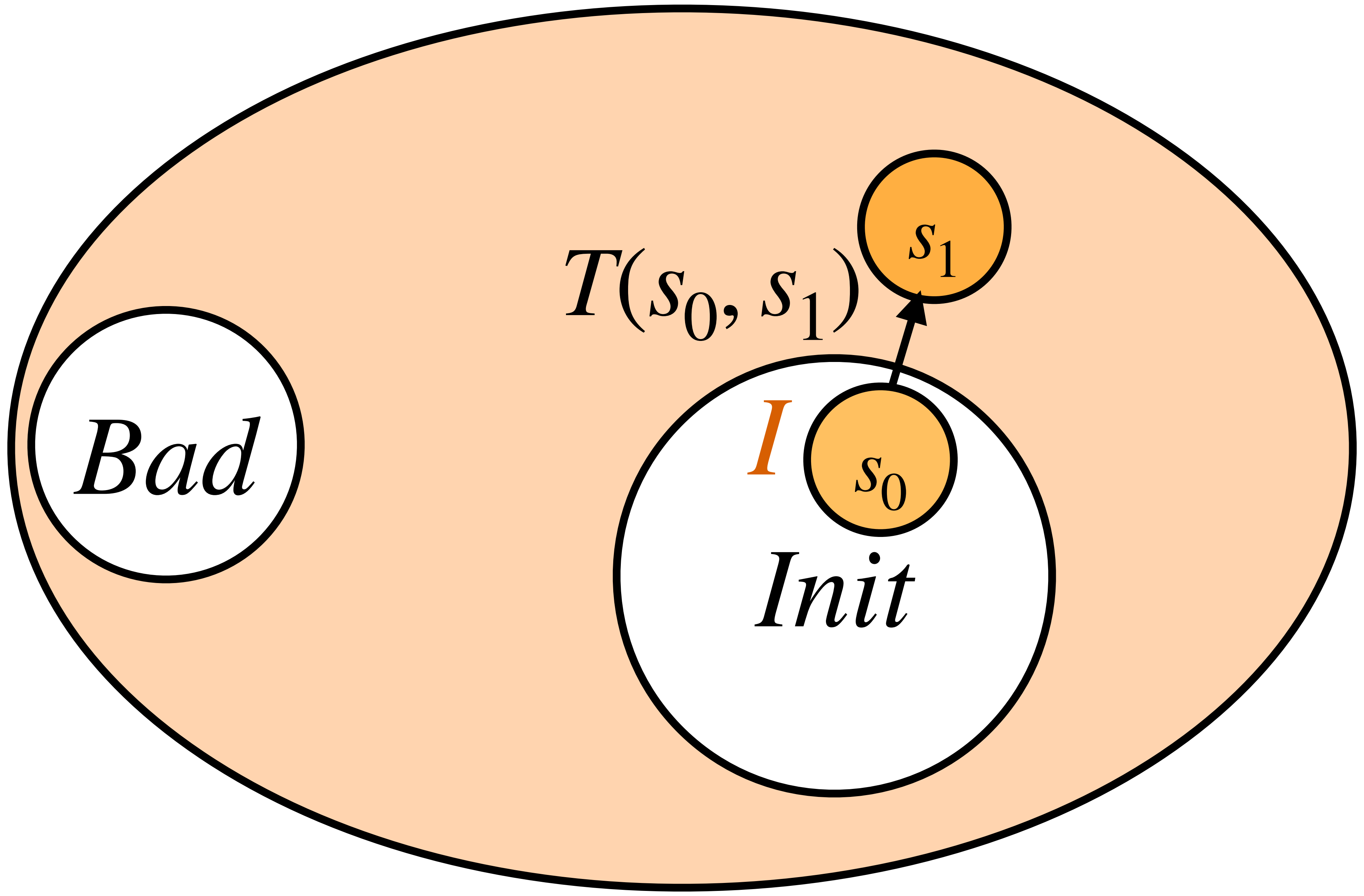
Bad states $Bad \subseteq S$

Formula I is an inductive invariant for the system if the following hold:

Initiation: $\forall s \in Init \,.\, I(s)$

Inductive Invariants for Transition Systems

For a transition system $(S, T, Init)$:



States S

Initial states $Init \subseteq S$

Transition relation T

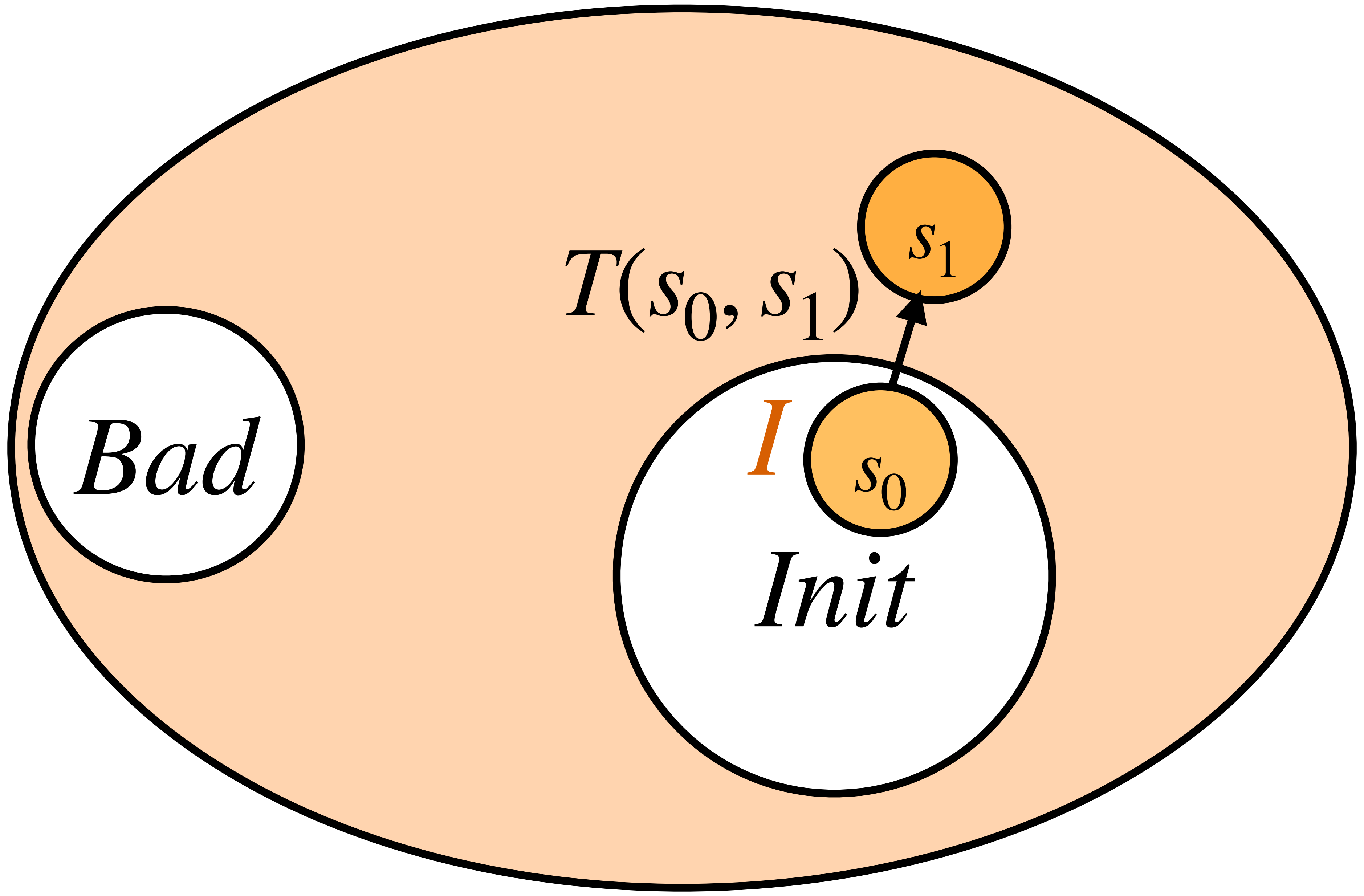
Bad states $Bad \subseteq S$

Formula I is an inductive invariant for the system if the following hold:

Initiation: $\forall s \in Init \,.\, I(s)$

Inductive Invariants for Transition Systems

For a transition system $(S, T, Init)$:



States S

Initial states $Init \subseteq S$

Transition relation T

Bad states $Bad \subseteq S$

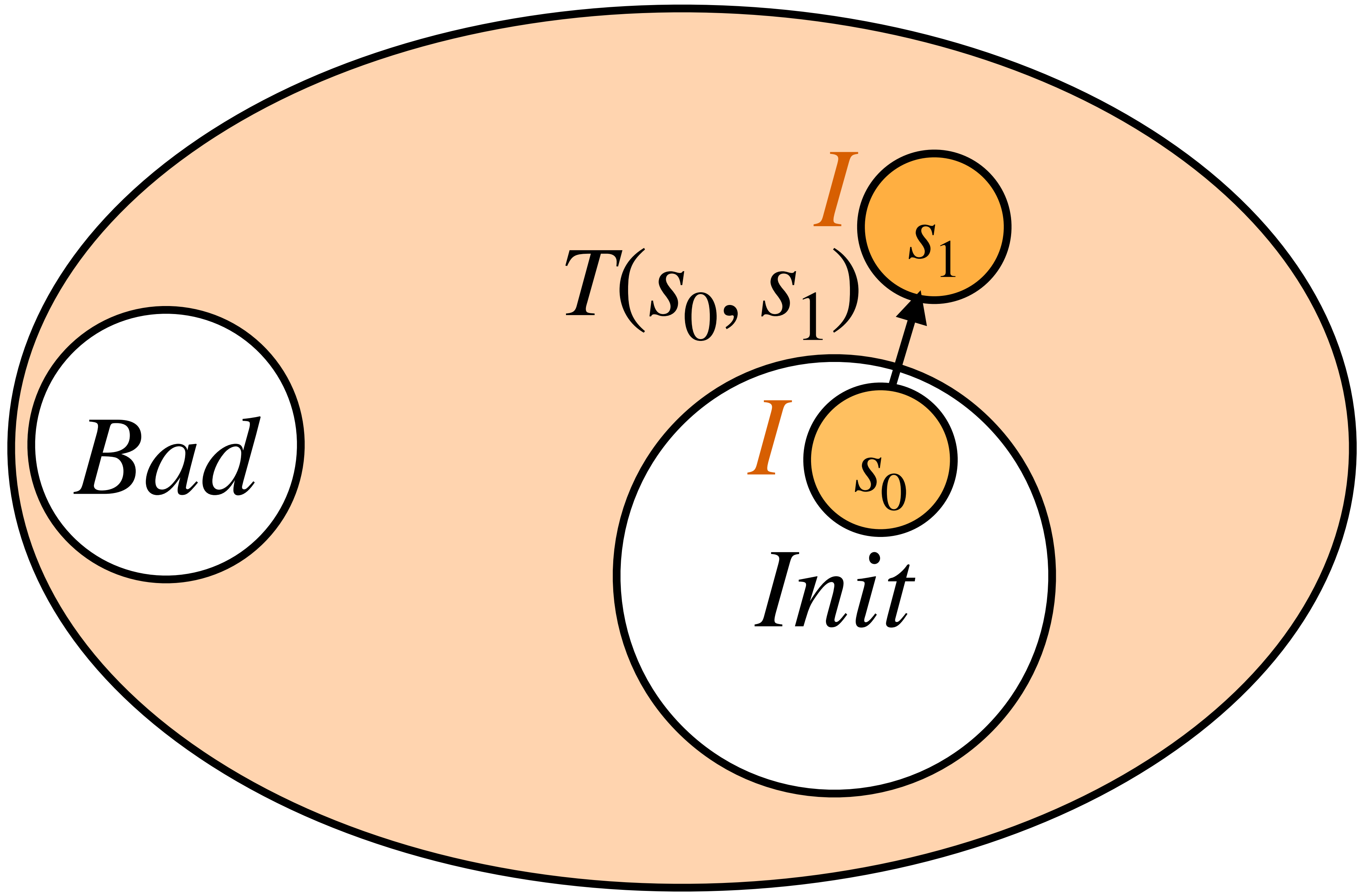
Formula I is an inductive invariant for the system if the following hold:

Initiation: $\forall s \in Init \,.\, I(s)$

Consecution: $\forall s, s' \in S \,.\, I(s) \wedge T(s, s') \Rightarrow I(s')$

Inductive Invariants for Transition Systems

For a transition system $(S, T, Init)$:



States S

Initial states $Init \subseteq S$

Transition relation T

Bad states $Bad \subseteq S$

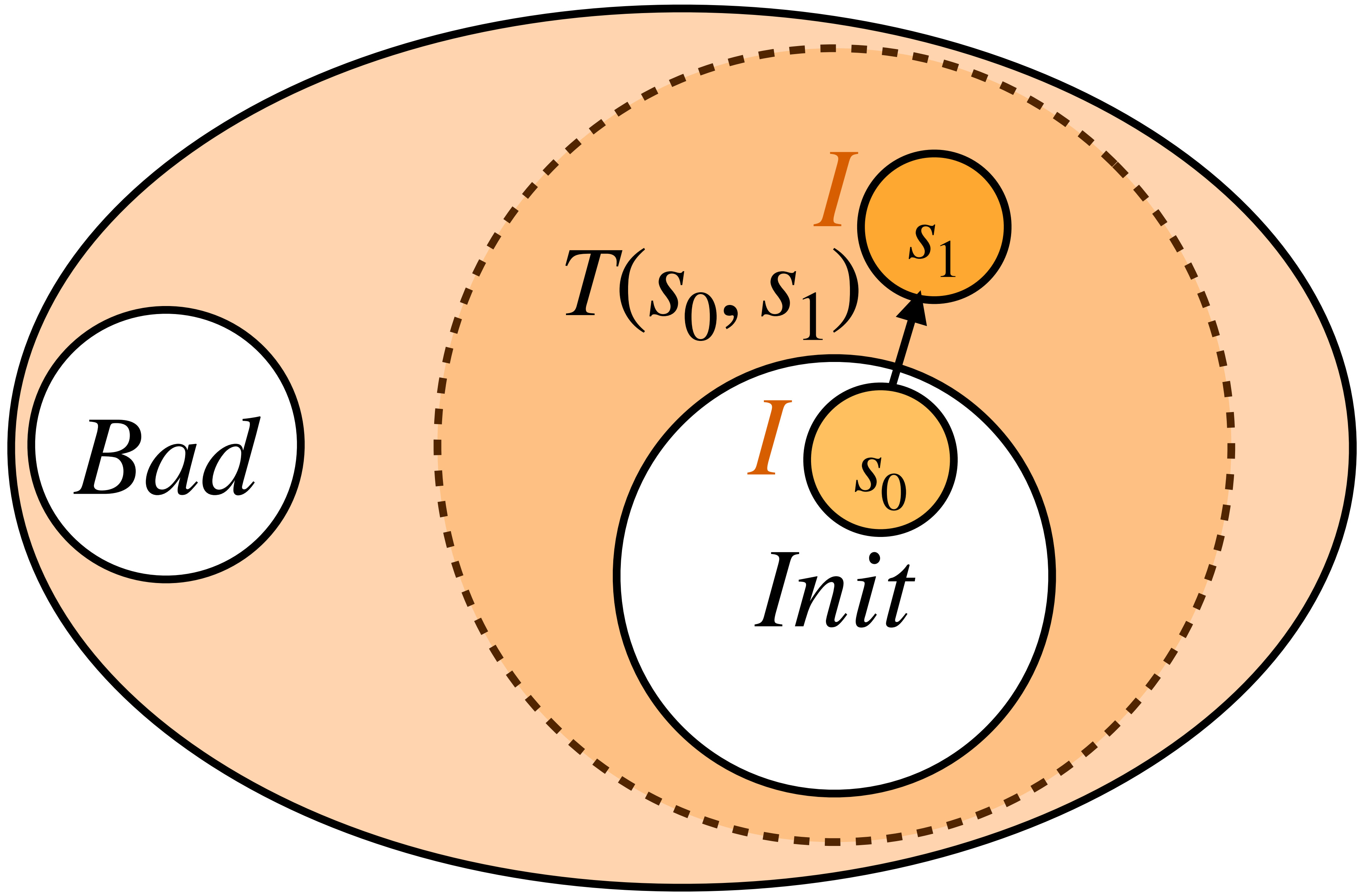
Formula I is an inductive invariant for the system if the following hold:

Initiation: $\forall s \in Init \,.\, I(s)$

Consecution: $\forall s, s' \in S \,.\, I(s) \land T(s, s') \Rightarrow I(s')$

Inductive Invariants for Transition Systems

For a transition system $(S, T, Init)$:



States S

Initial states $Init \subseteq S$

Transition relation T

Bad states $Bad \subseteq S$

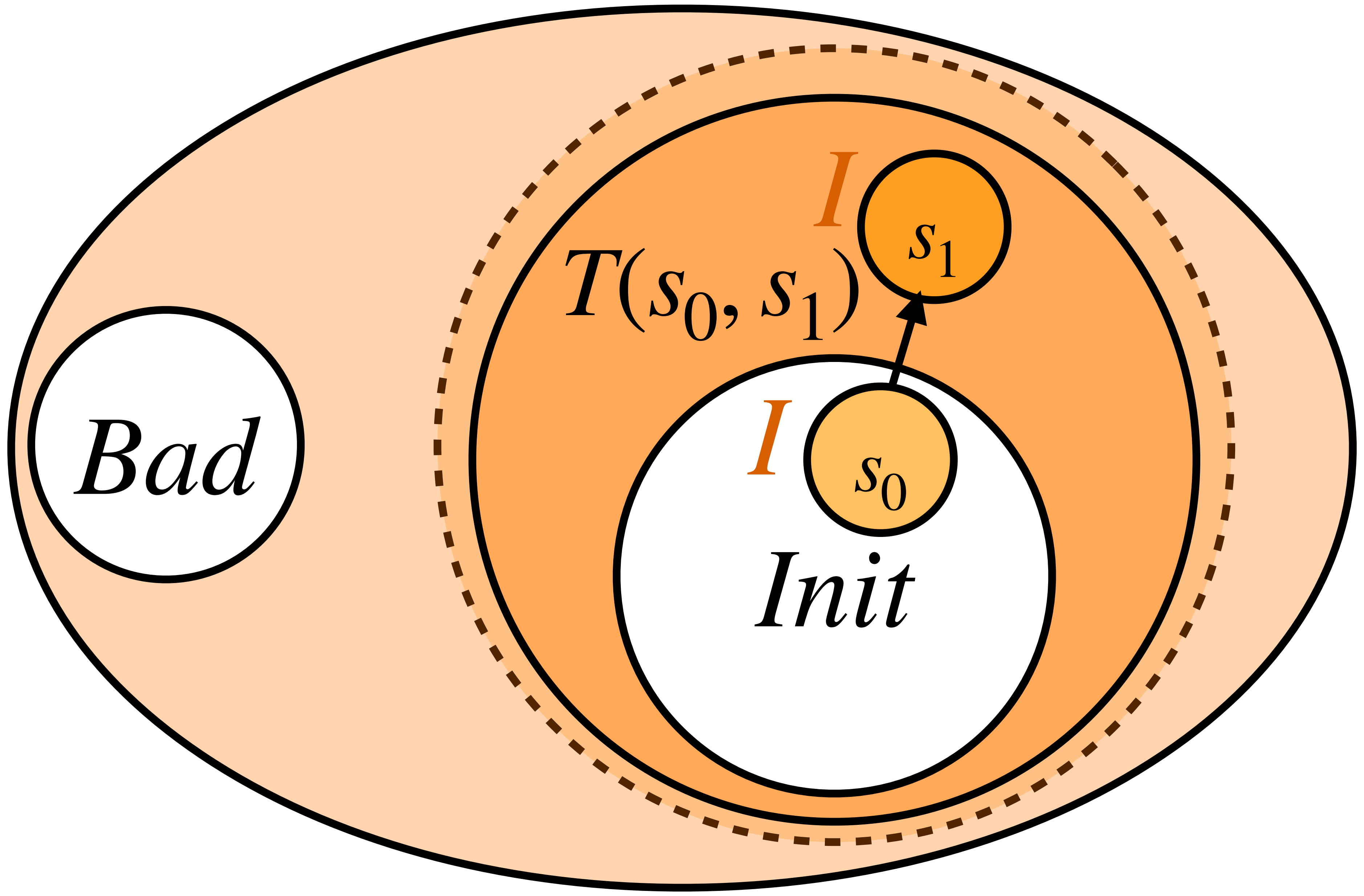
Formula I is an inductive invariant for the system if the following hold:

Initiation: $\forall s \in Init \,.\, I(s)$

Consecution: $\forall s, s' \in S \,.\, I(s) \wedge T(s, s') \Rightarrow I(s')$

Inductive Invariants for Transition Systems

For a transition system $(S, T, Init)$:



States S

Initial states $Init \subseteq S$

Transition relation T

Bad states $Bad \subseteq S$

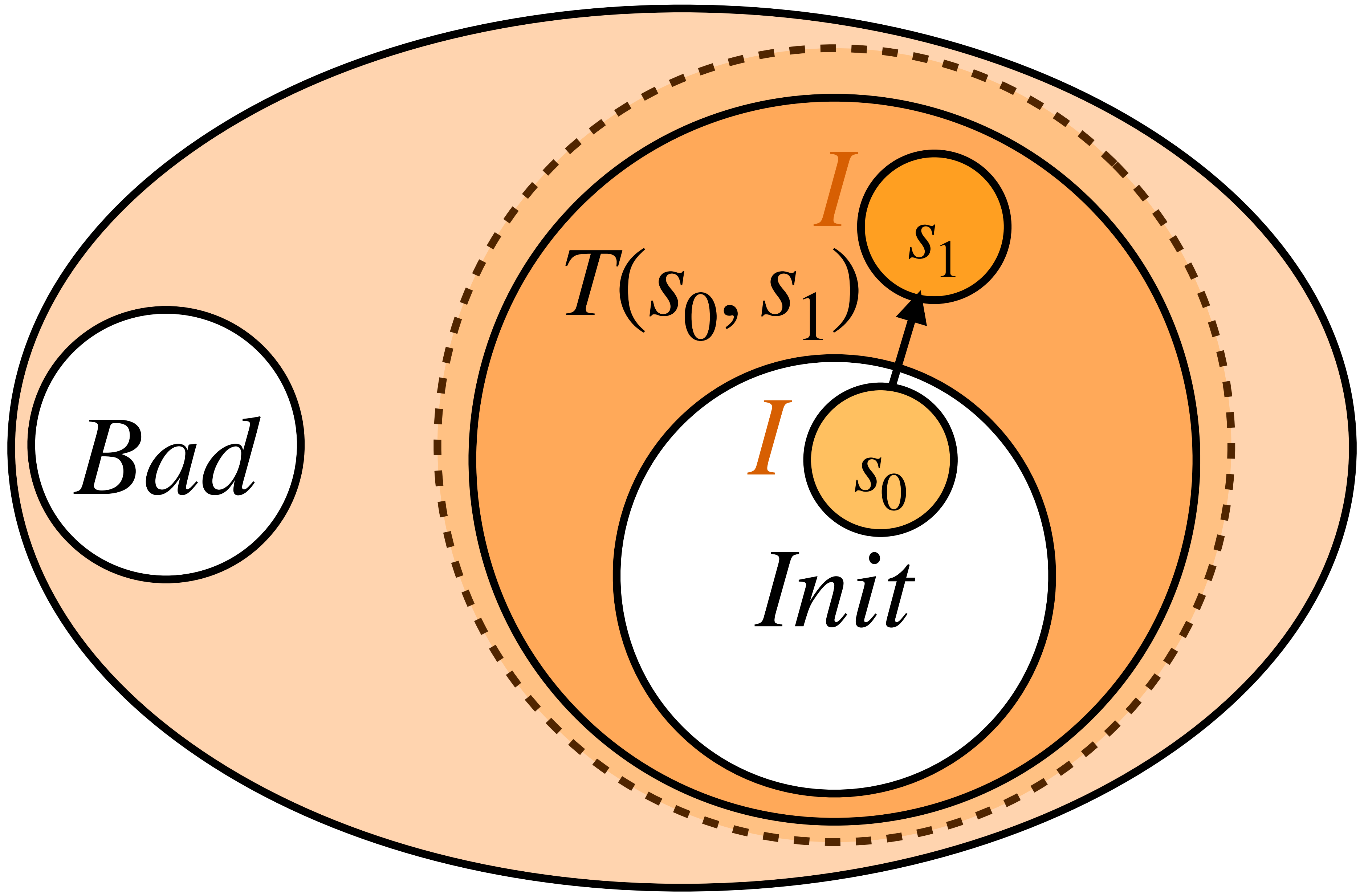
Formula I is an inductive invariant for the system if the following hold:

Initiation: $\forall s \in Init \,.\, I(s)$

Consecution: $\forall s, s' \in S \,.\, I(s) \wedge T(s, s') \Rightarrow I(s')$

Inductive Invariants for Transition Systems

For a transition system $(S, T, Init)$:



States S

Initial states $Init \subseteq S$

Transition relation T

Bad states $Bad \subseteq S$

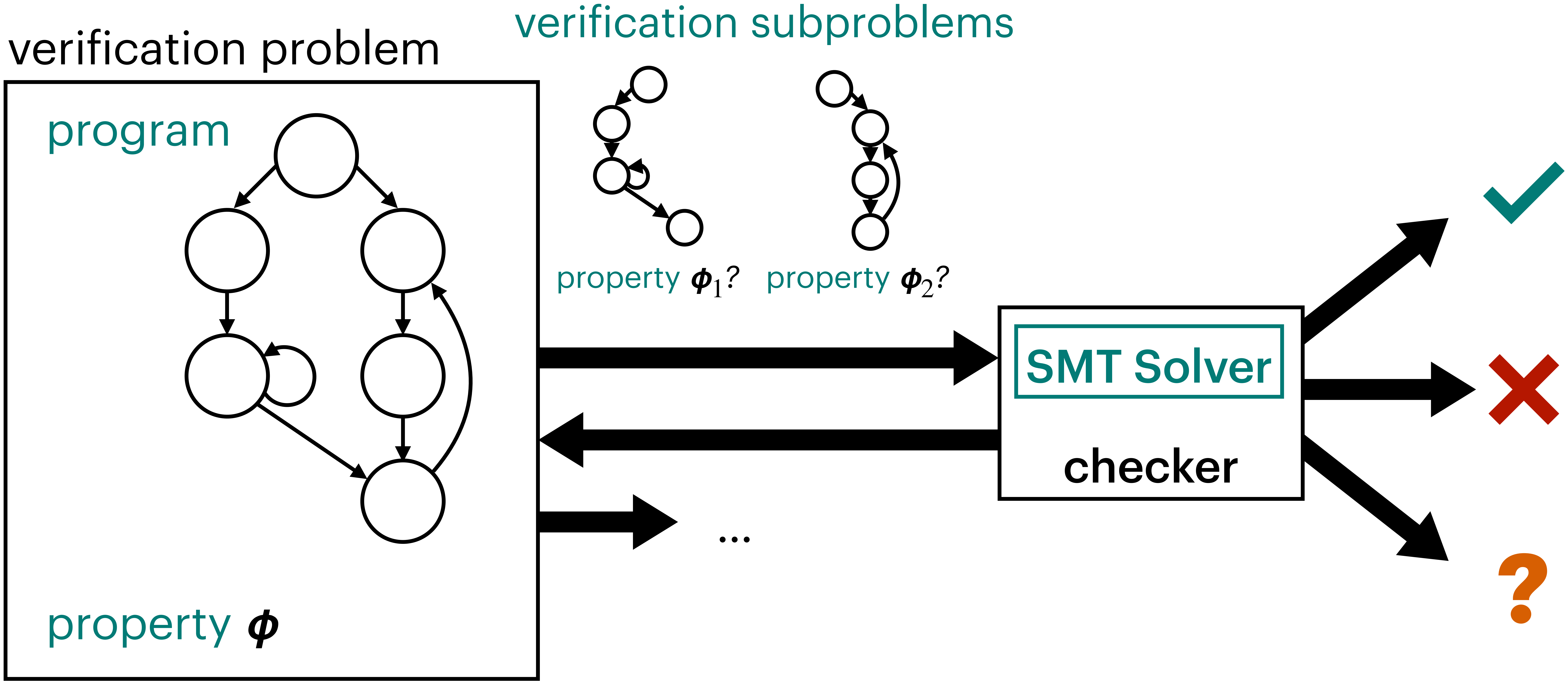
Formula I is an inductive invariant for the system if the following hold:

Initiation: $\forall s \in Init \,.\, I(s)$

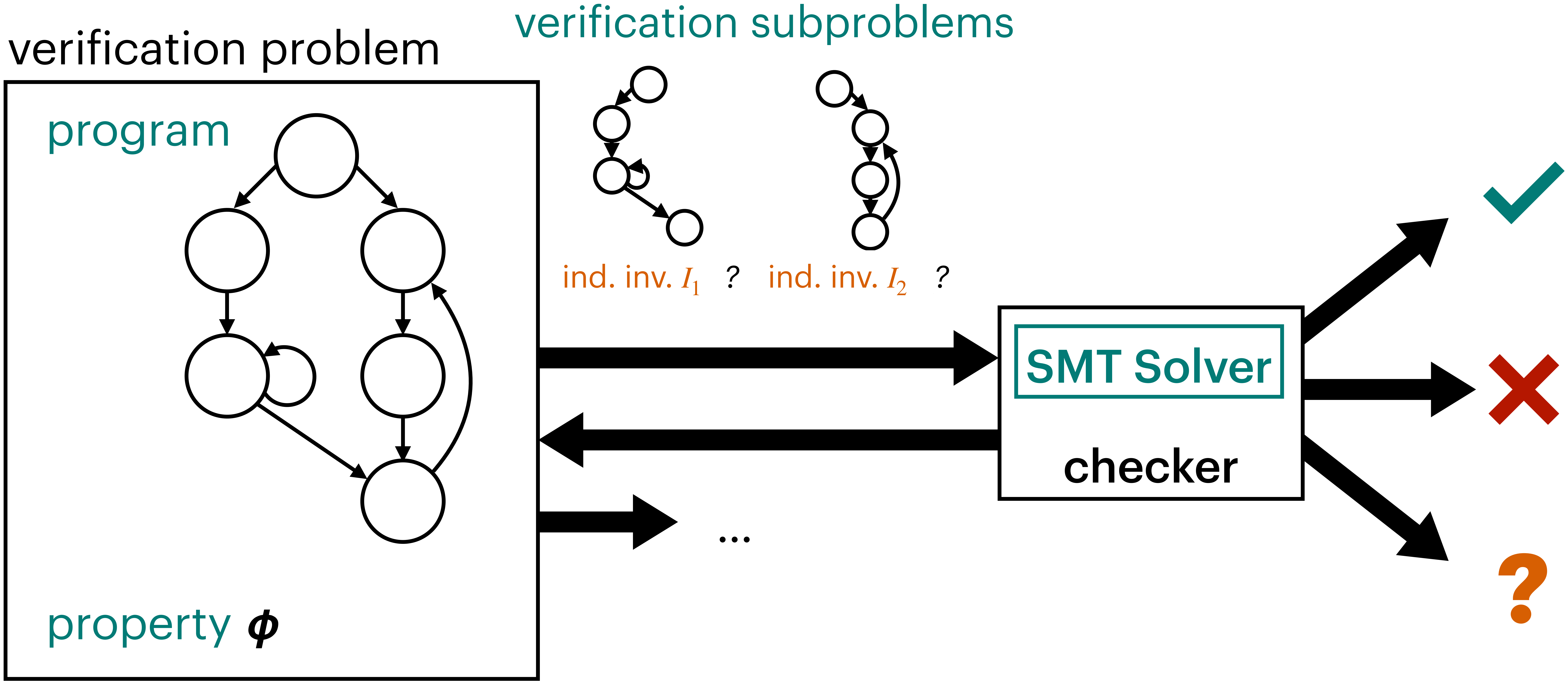
Consecution: $\forall s, s' \in S \,.\, I(s) \wedge T(s, s') \Rightarrow I(s')$

Can use invariants to help prove safety properties: $\forall s \in S \,.\, I(s) \Rightarrow \neg Bad(s)$

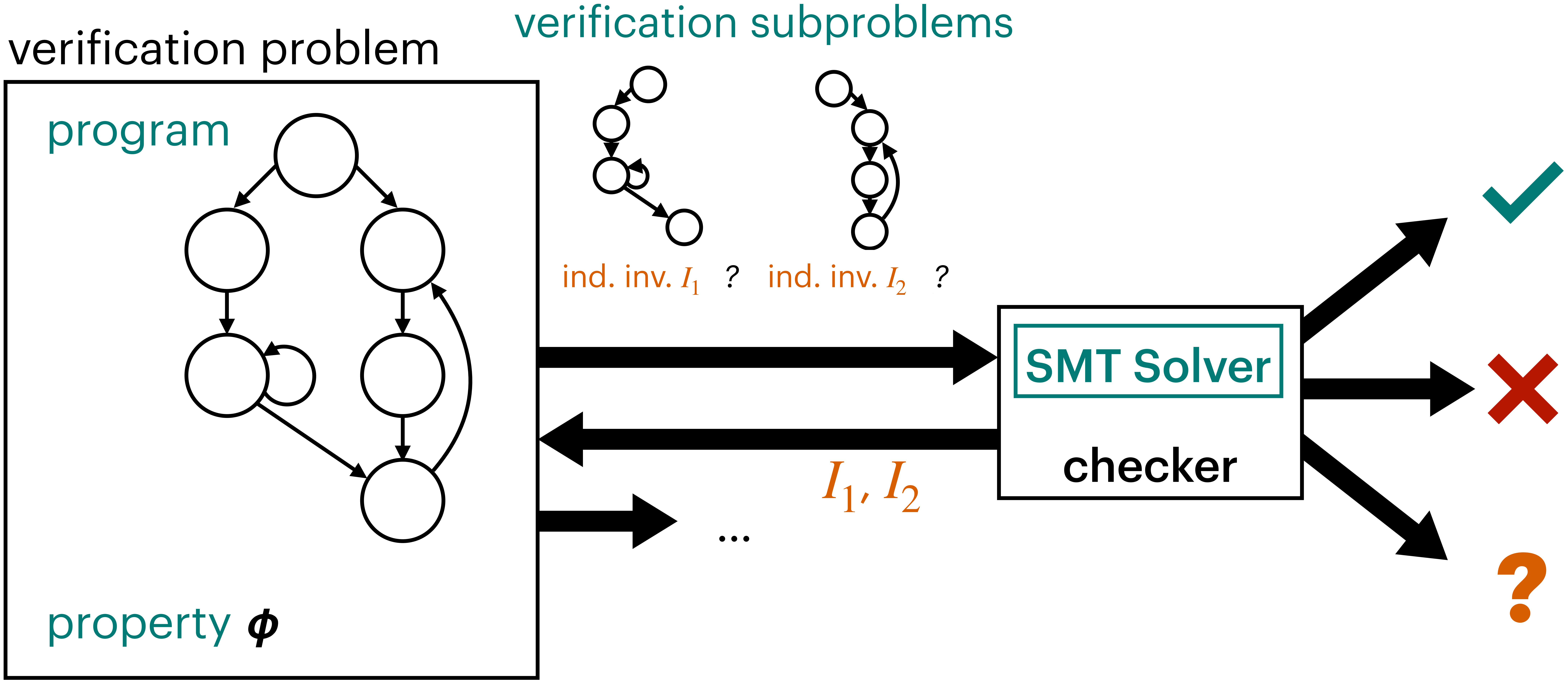
Automated Modular Verification



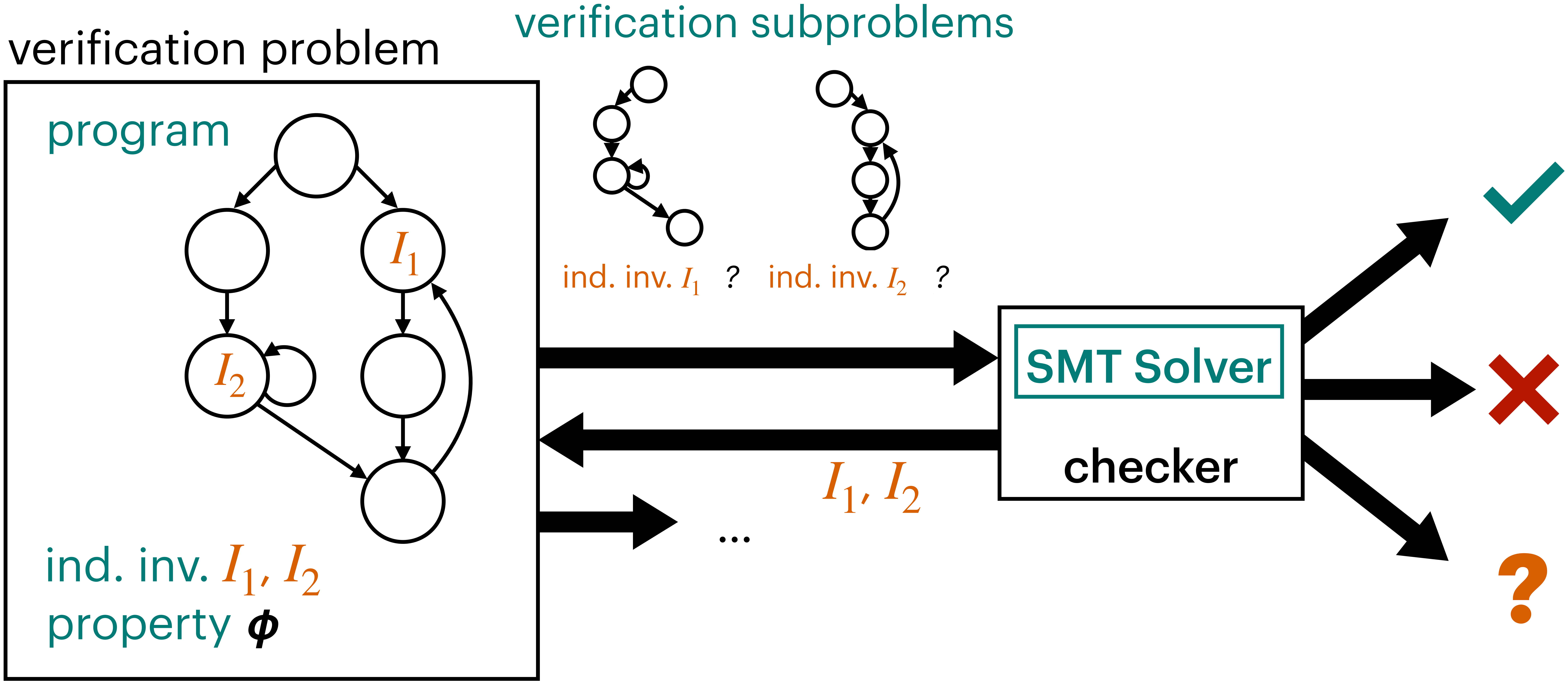
Automated Modular Verification



Automated Modular Verification

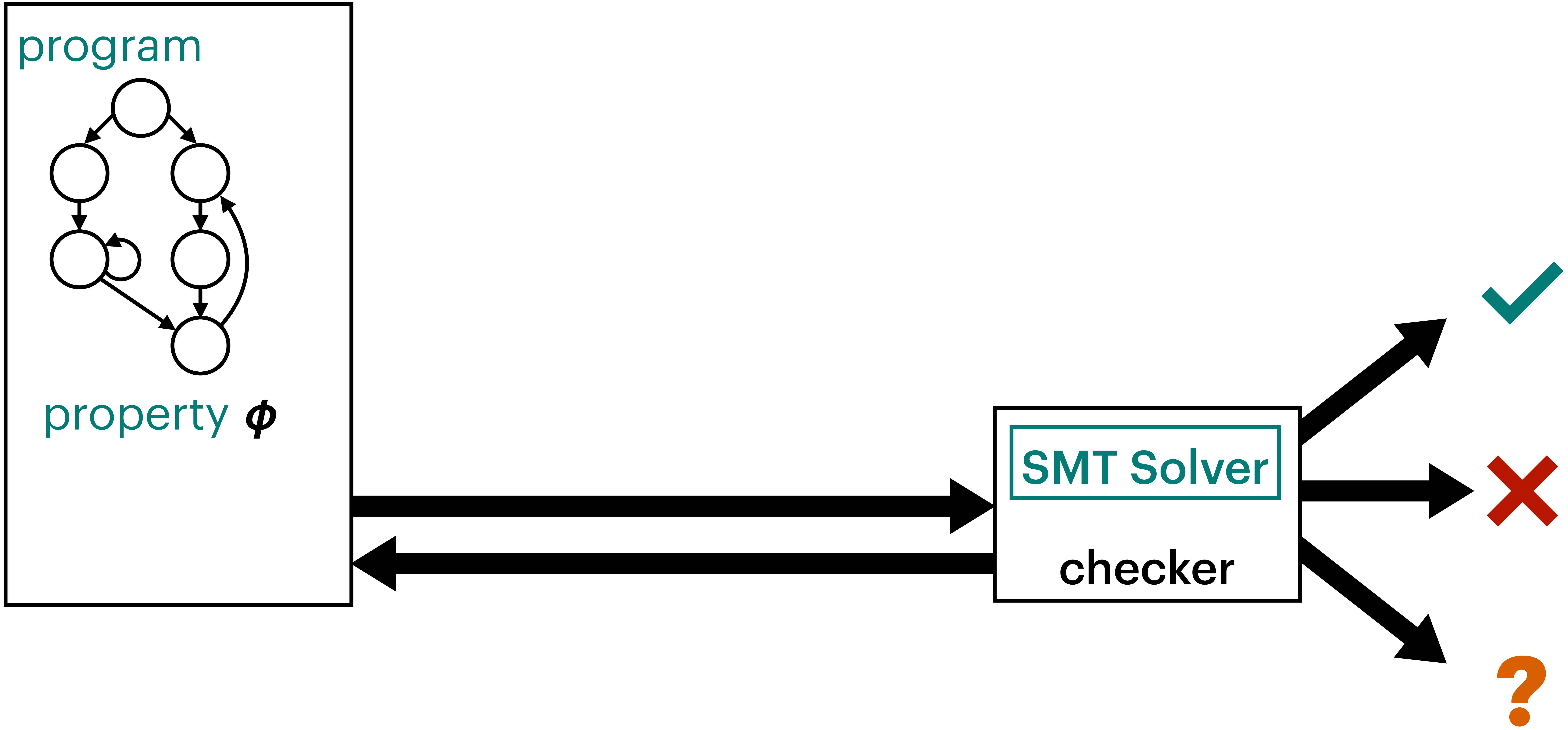


Automated Modular Verification



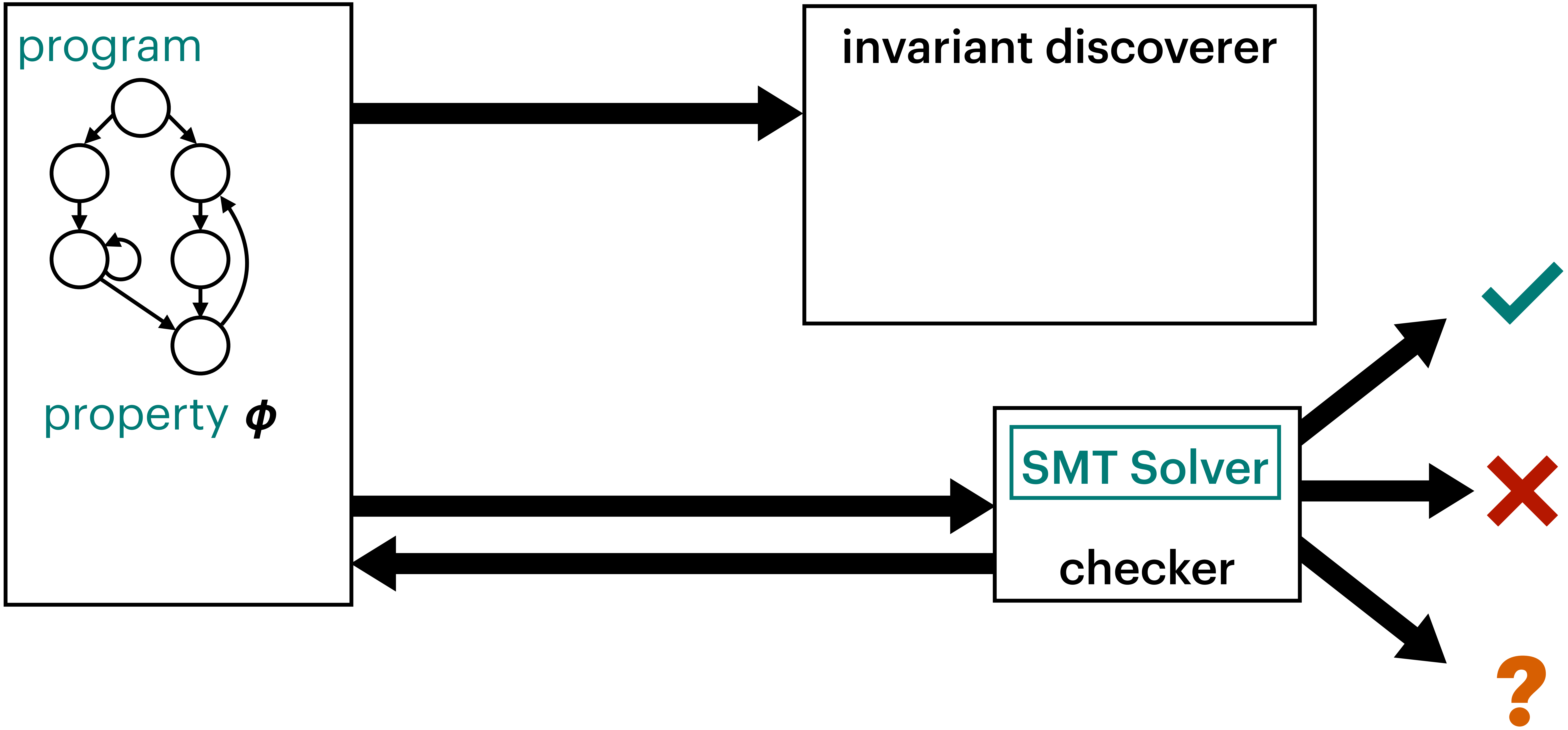
Invariant Discovery

Consider how to discover invariants



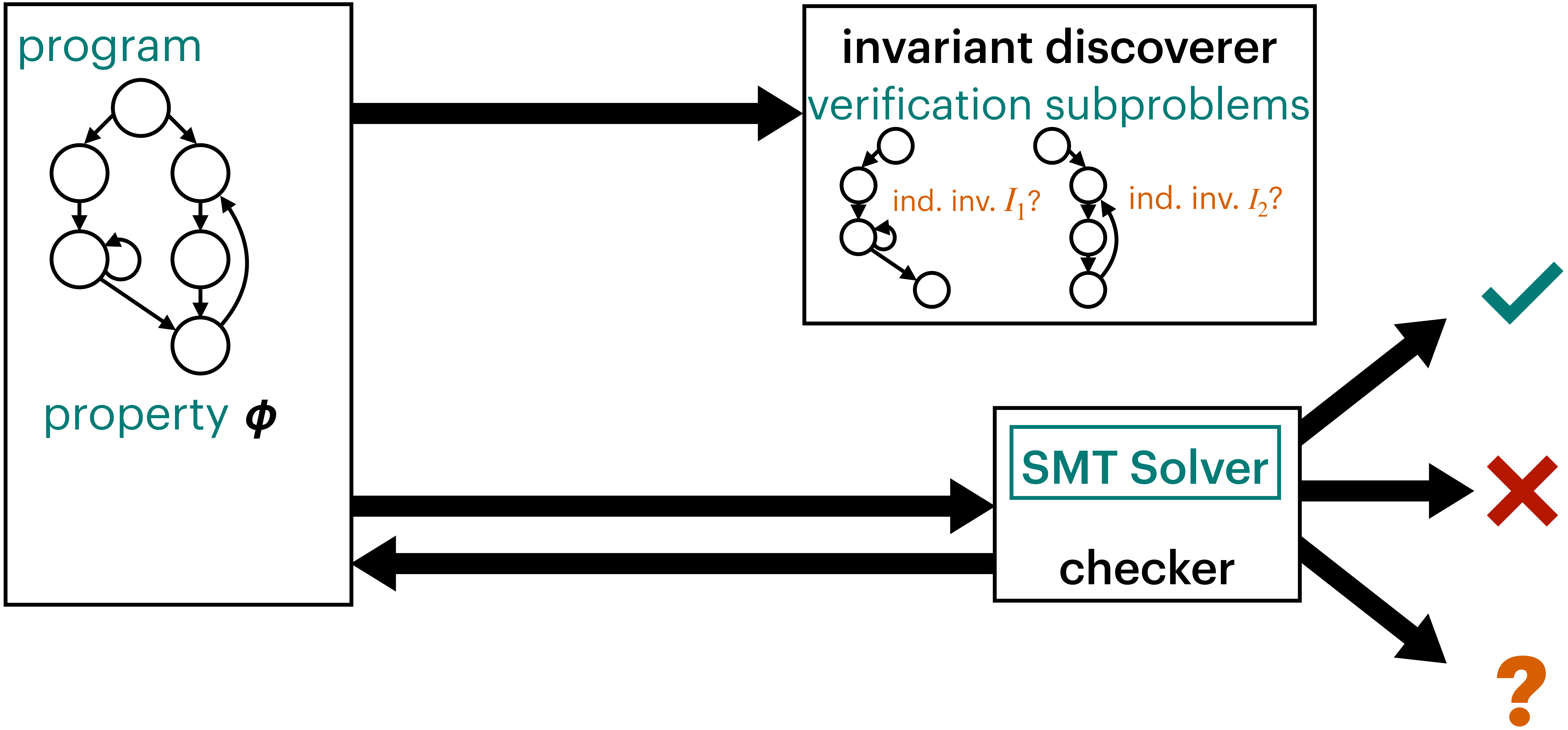
Invariant Discovery

Consider how to discover invariants



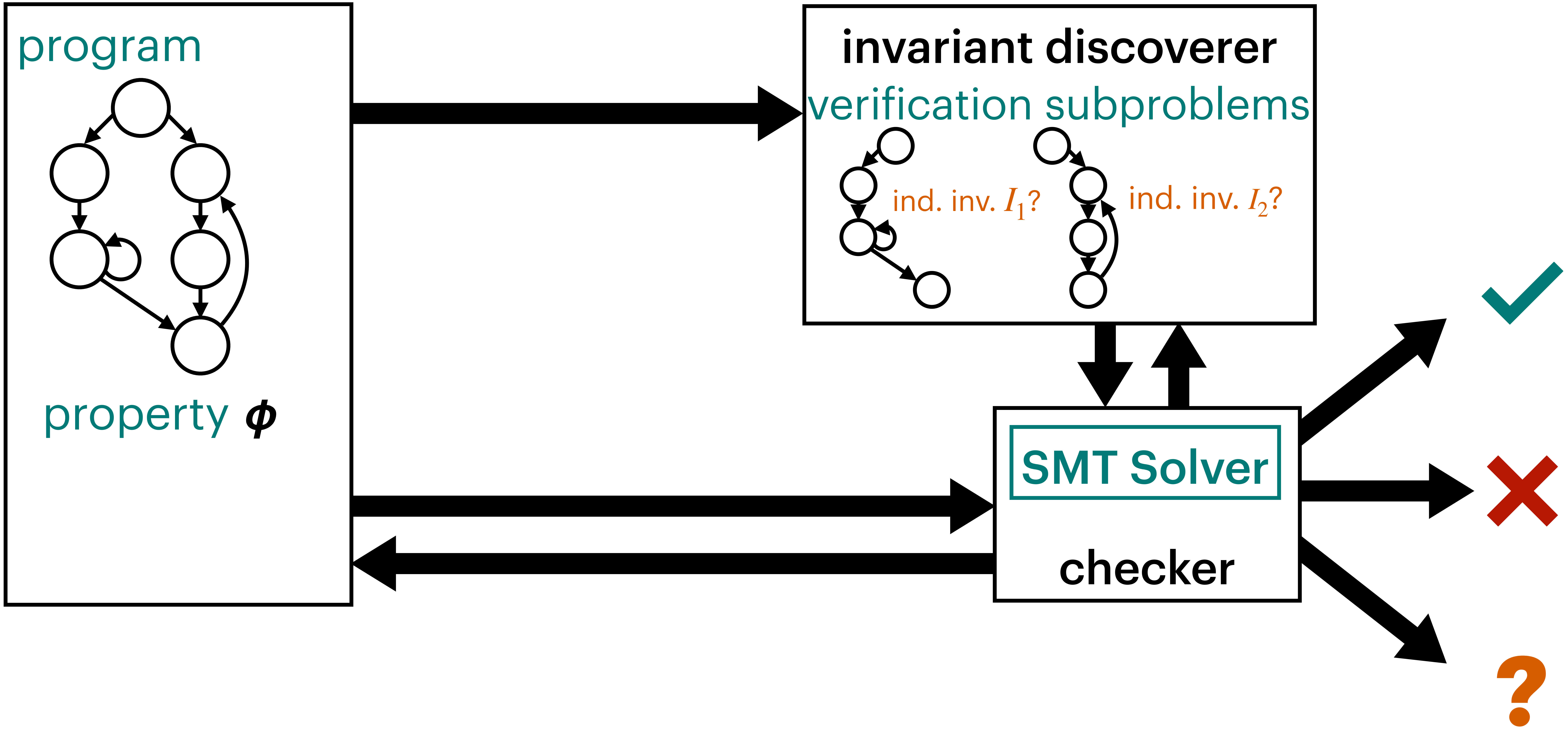
Invariant Discovery

Consider how to discover invariants



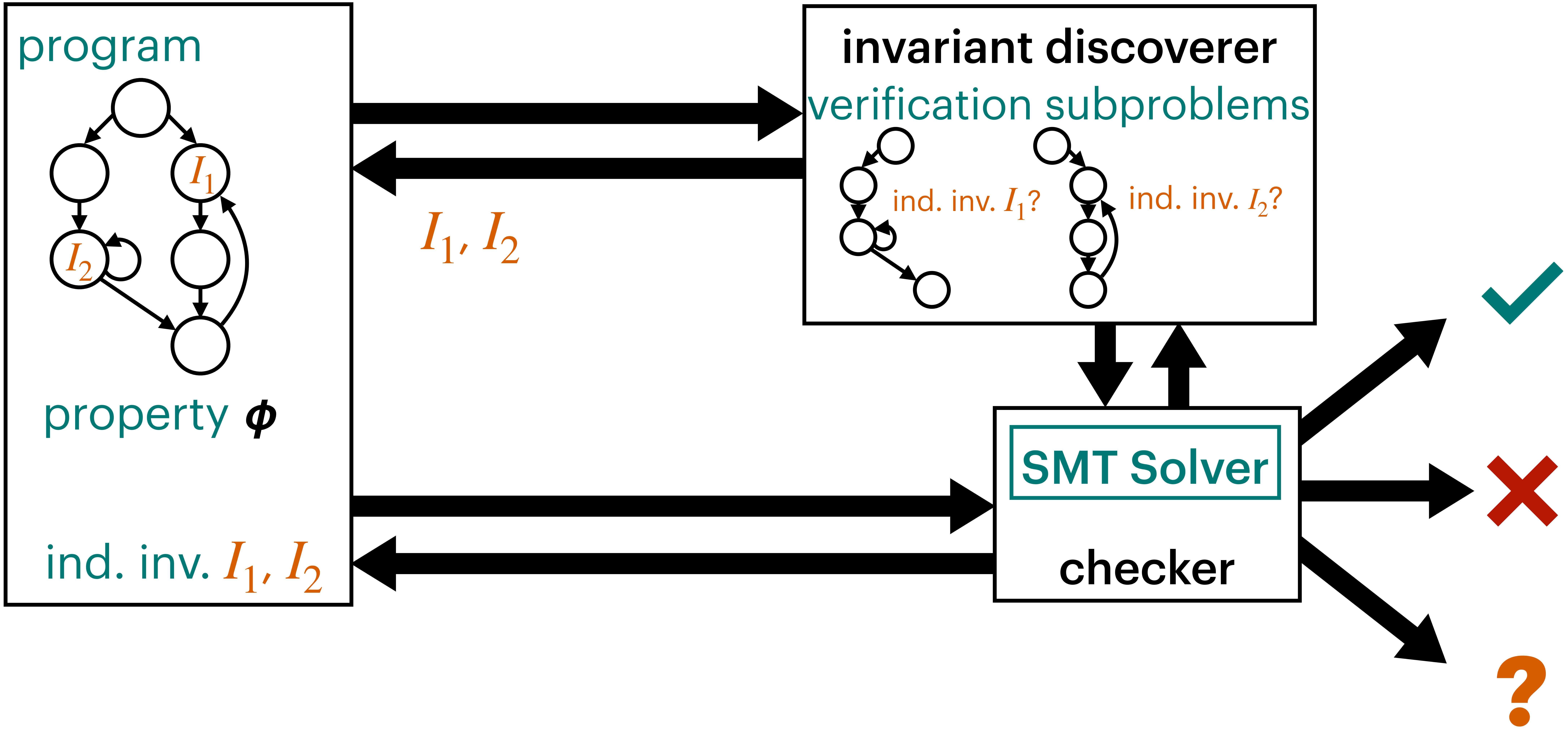
Invariant Discovery

Consider how to discover invariants



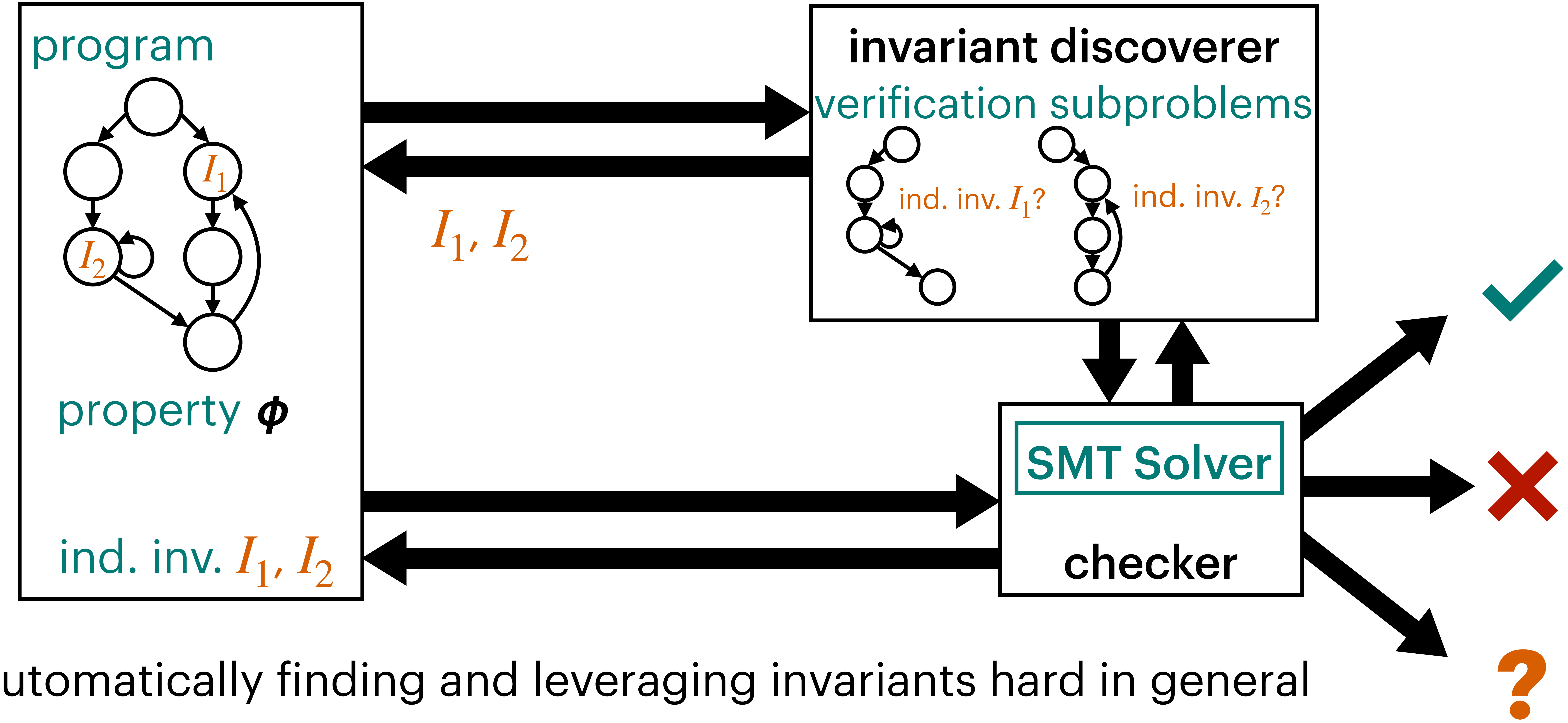
Invariant Discovery

Consider how to discover invariants



Invariant Discovery

Consider how to discover invariants

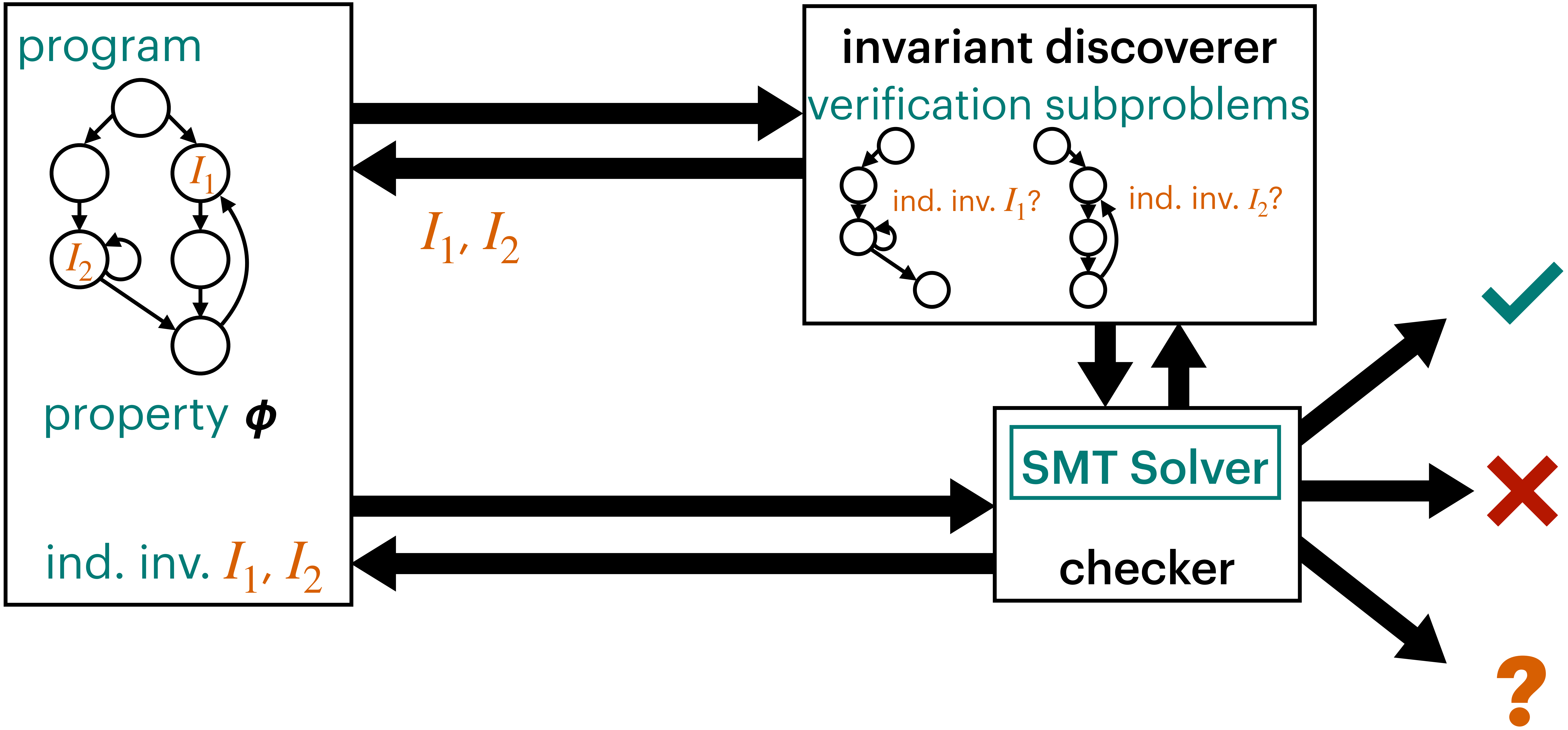


Automatically finding and leveraging invariants hard in general

Structure and Syntax

Structural info about programs and properties can help with:

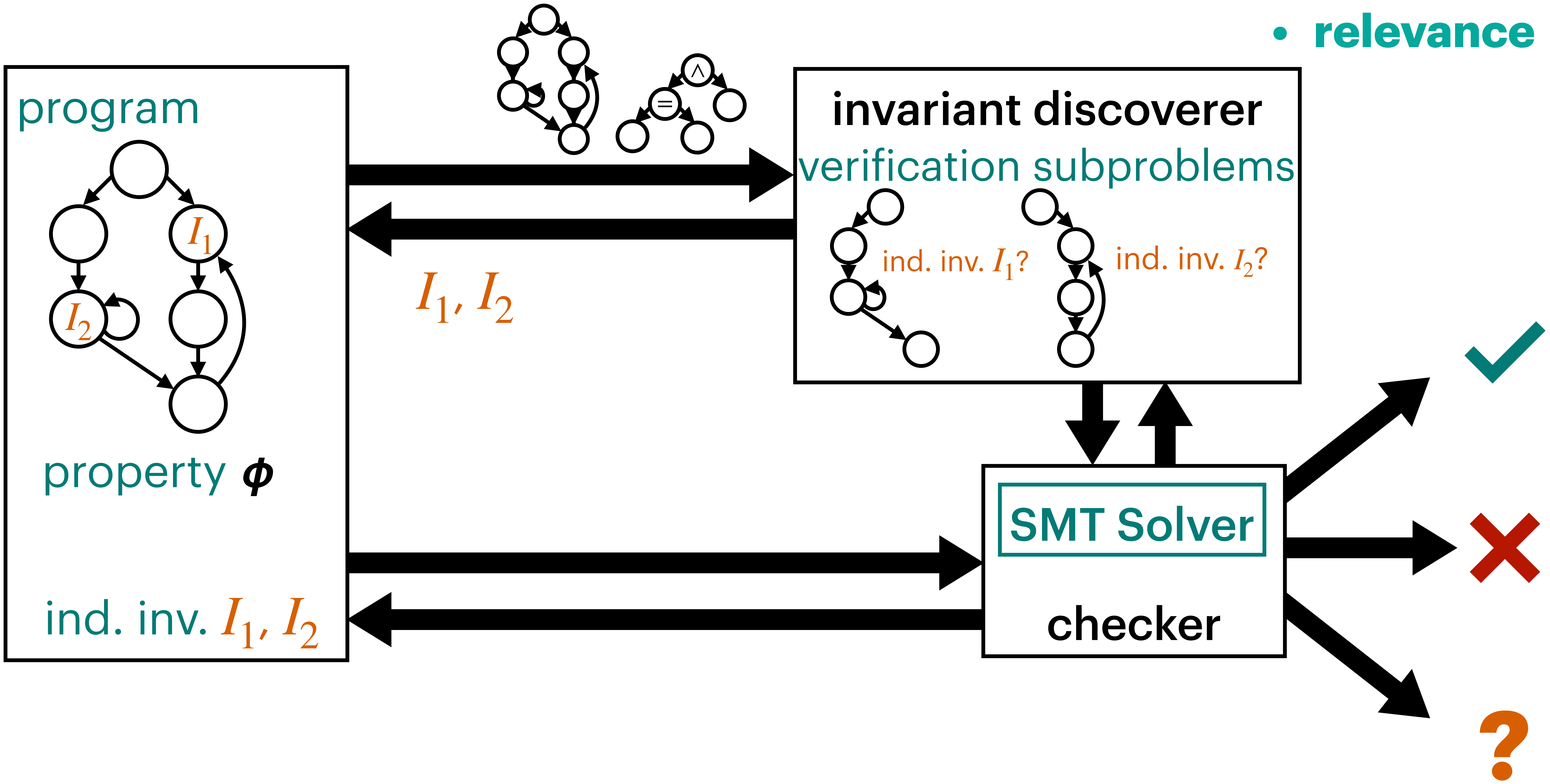
- **performance**
- **scalability**
- **relevance**



Structure and Syntax

Structural info about programs and properties can help with:

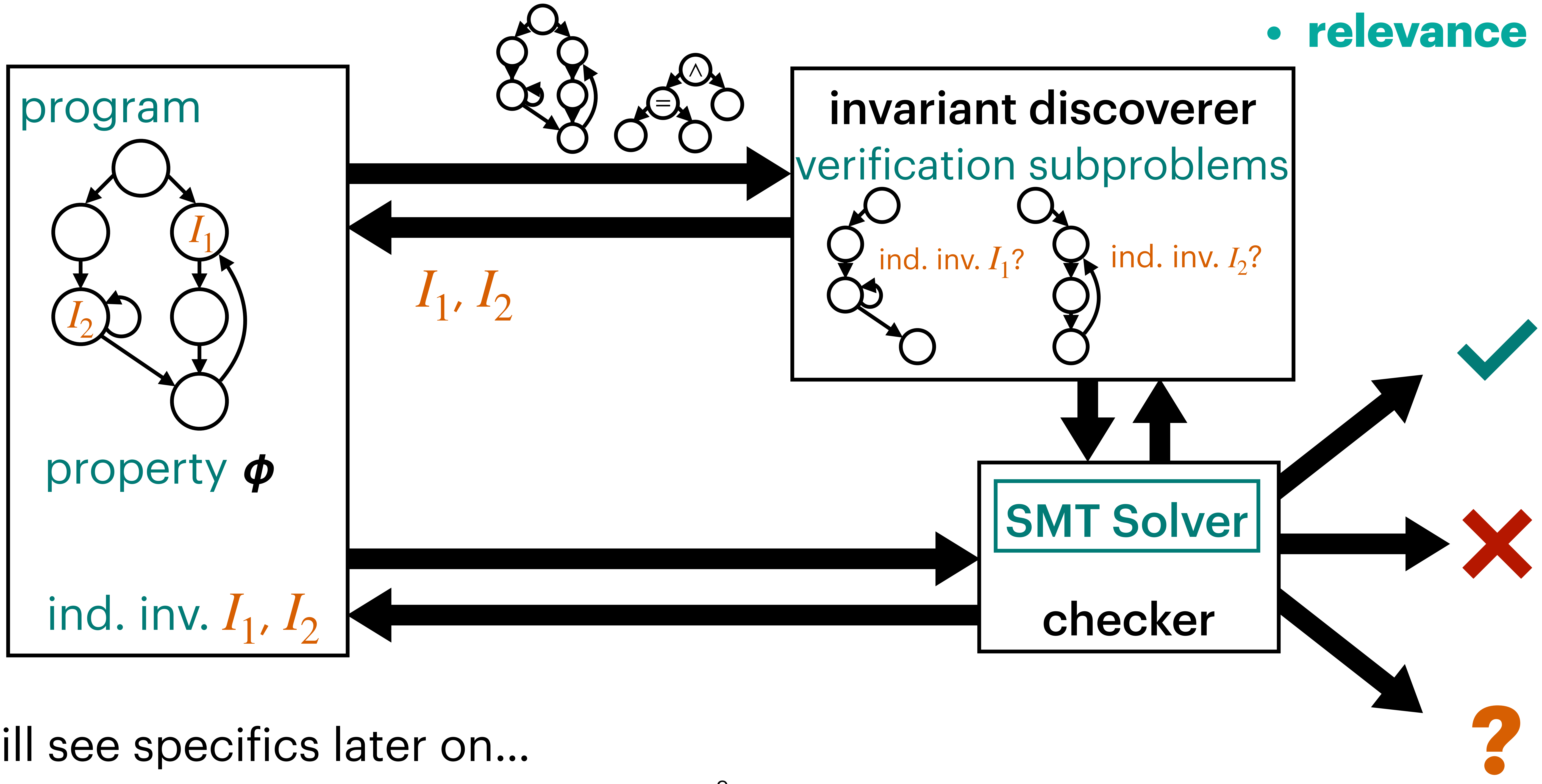
- **performance**
- **scalability**
- **relevance**



Structure and Syntax

Structural info about programs and properties can help with:

- **performance**
- **scalability**
- **relevance**



Will see specifics later on...

Contributions

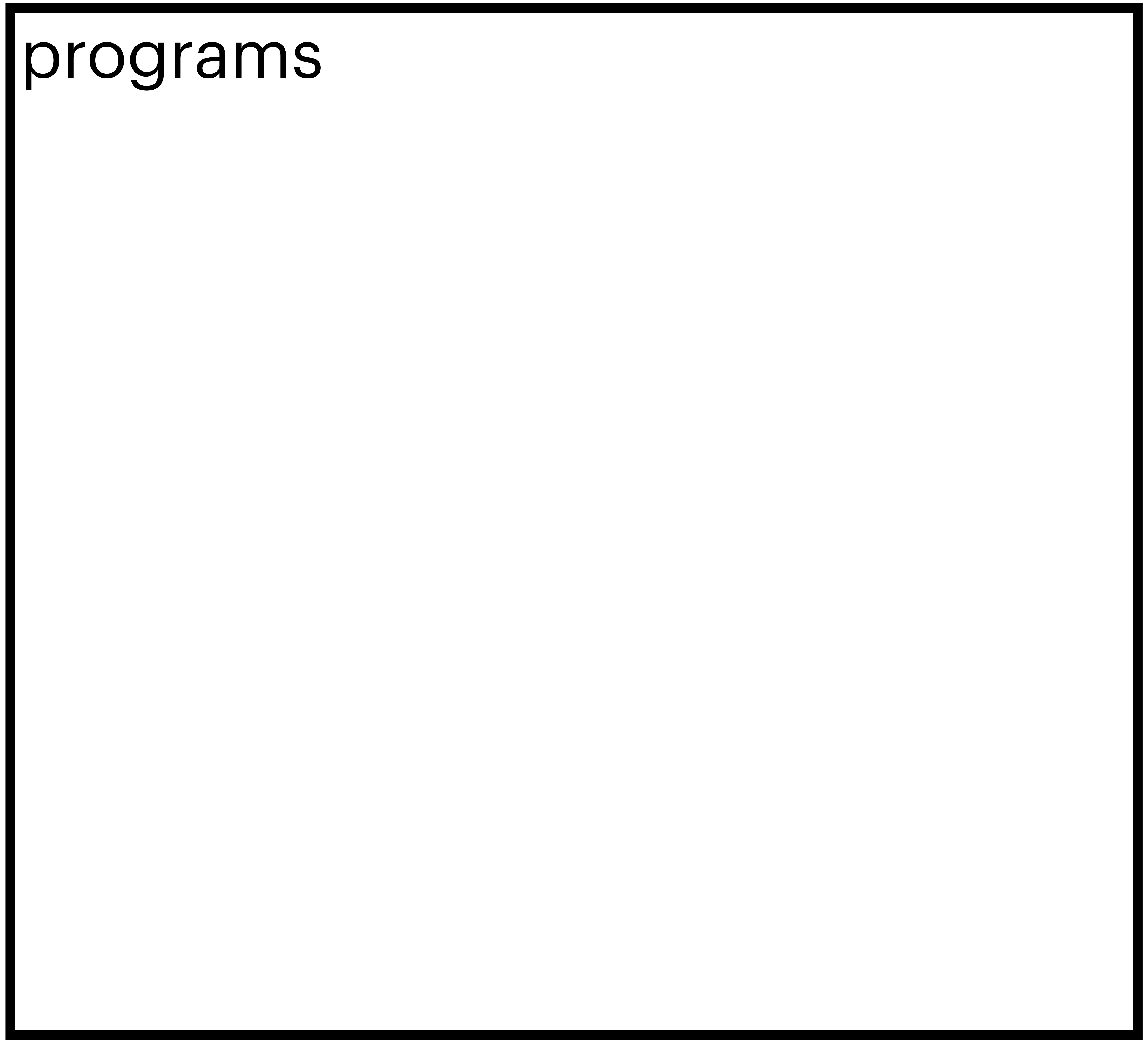
How to exploit **structure** of both **programs** and **properties**
to infer and leverage **invariants** that improve **scalability** and **performance**
in SMT-based automated verification.

Programs and Properties

Consider **certain kinds** of programs + properties rather than **general** ones

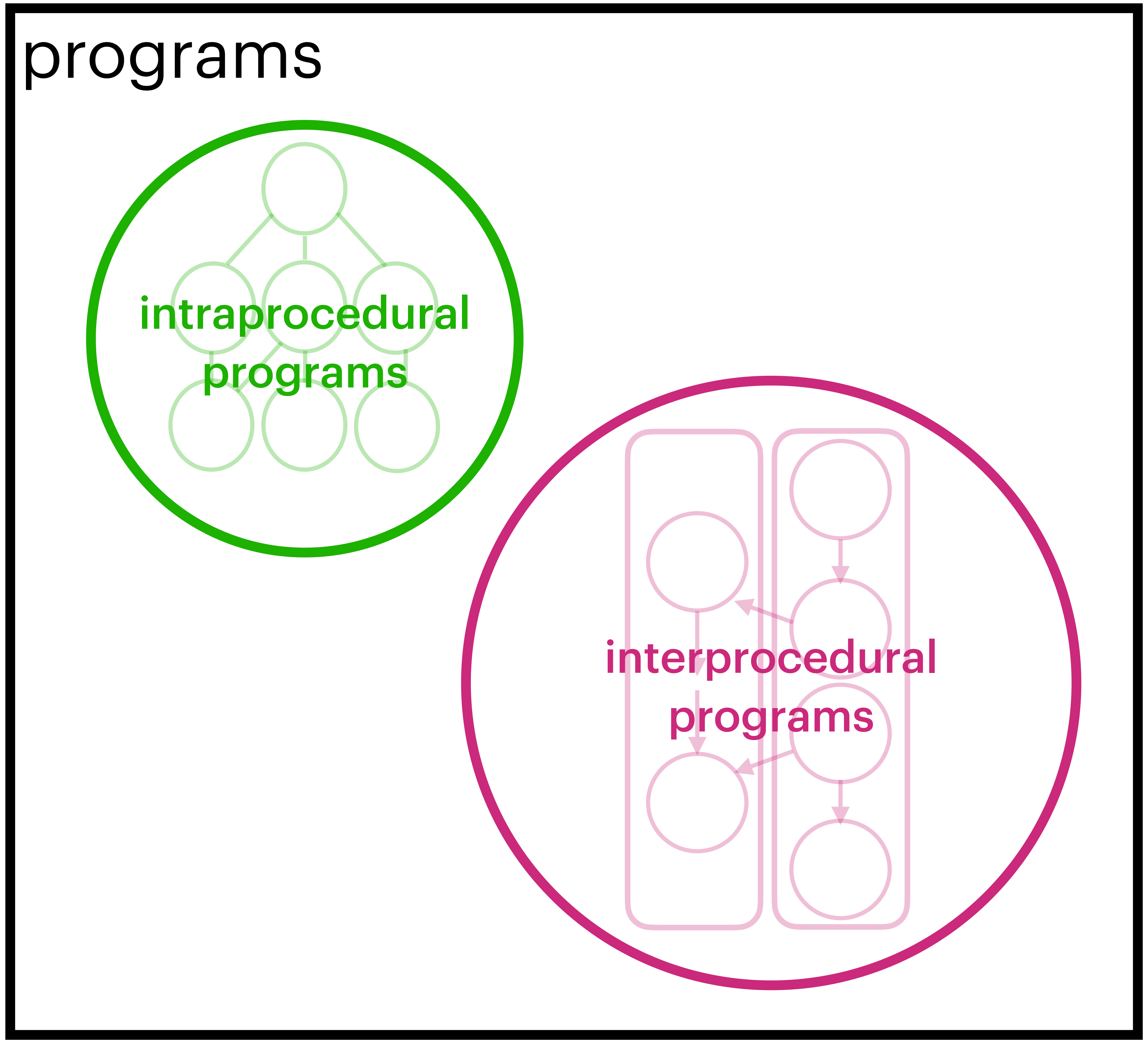
Programs and Properties

Consider **certain kinds** of programs + properties rather than **general** ones



Programs and Properties

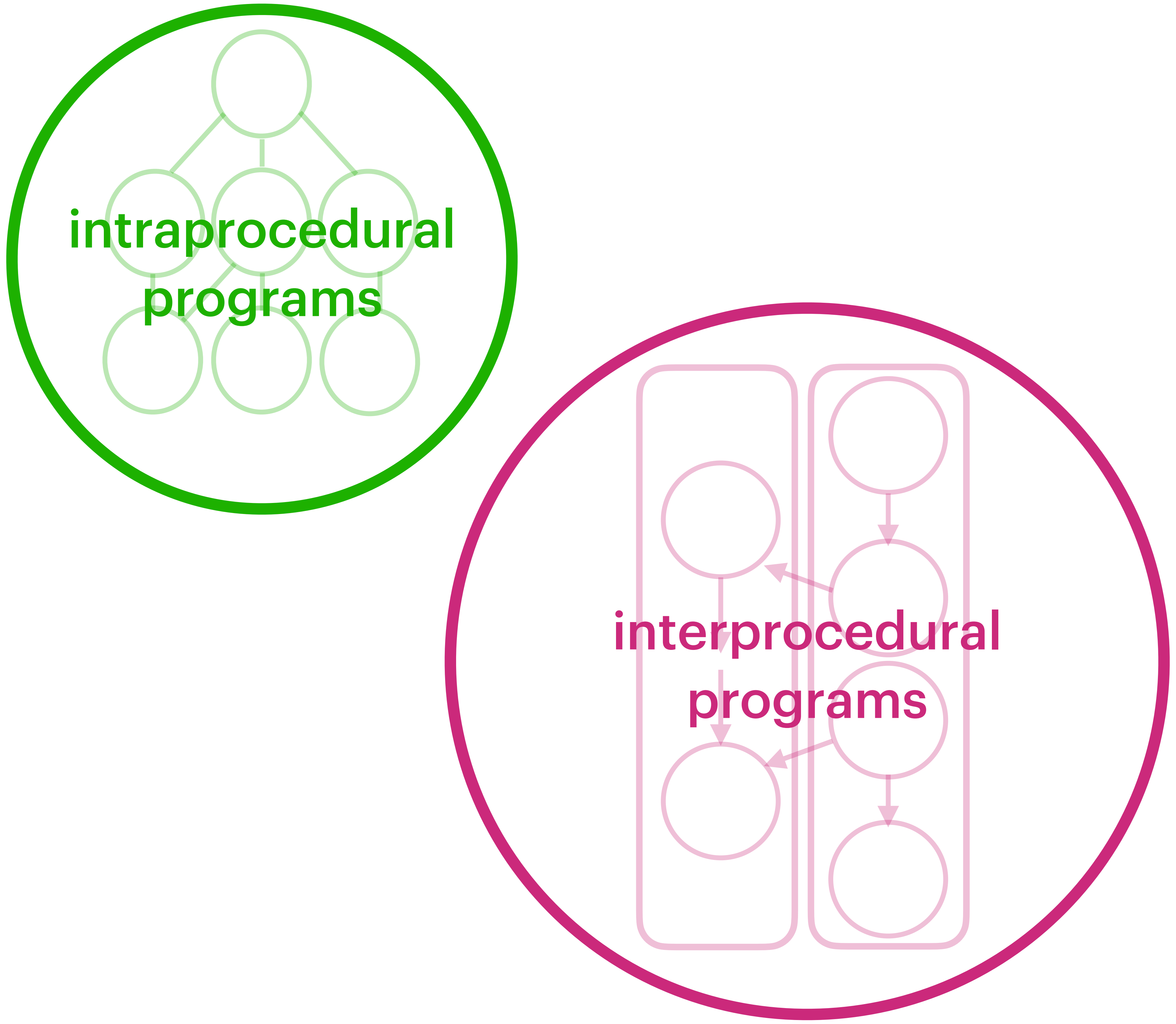
Consider **certain kinds** of programs + properties rather than **general** ones



Programs and Properties

Consider **certain kinds** of programs + properties rather than **general** ones

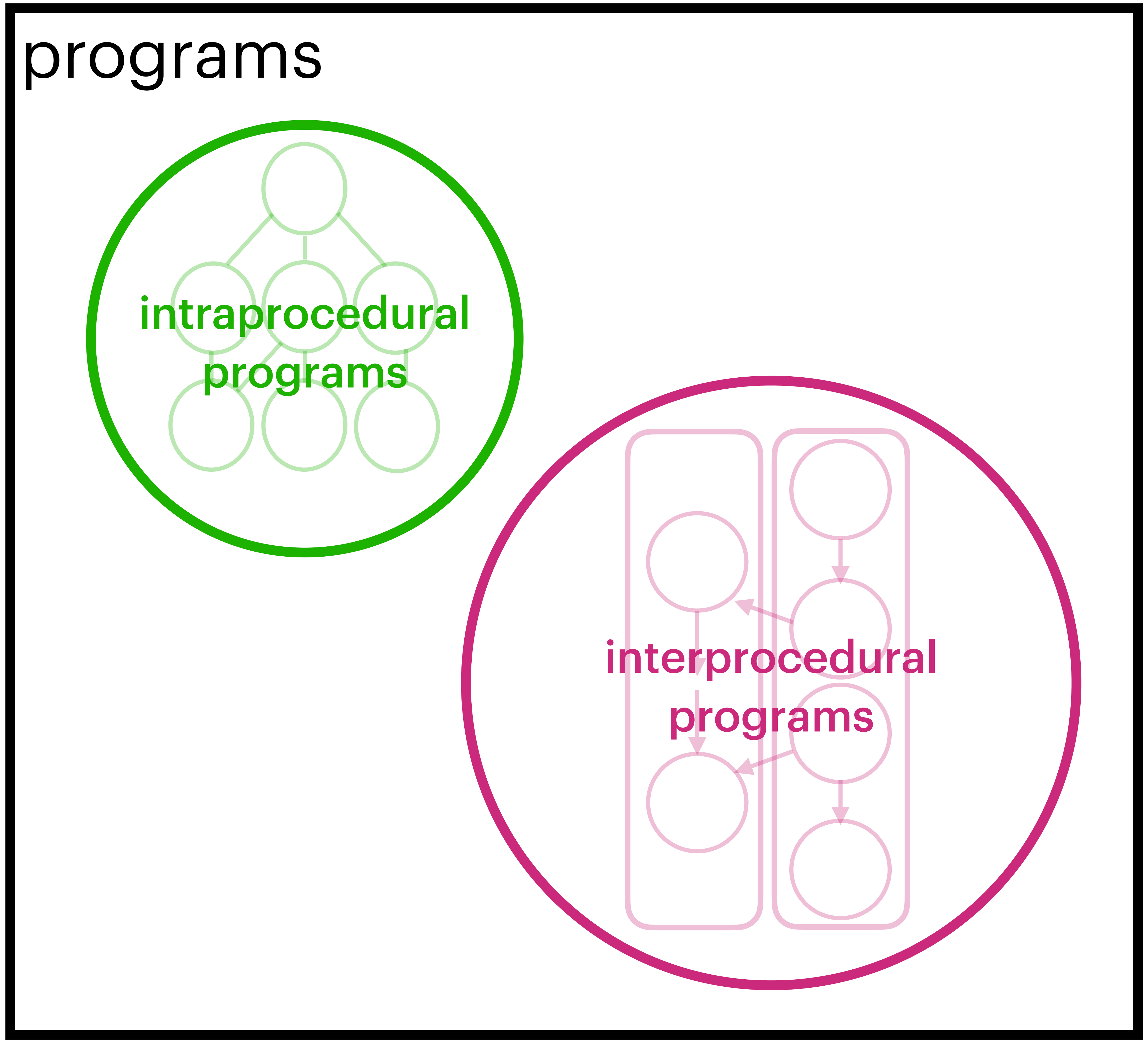
programs

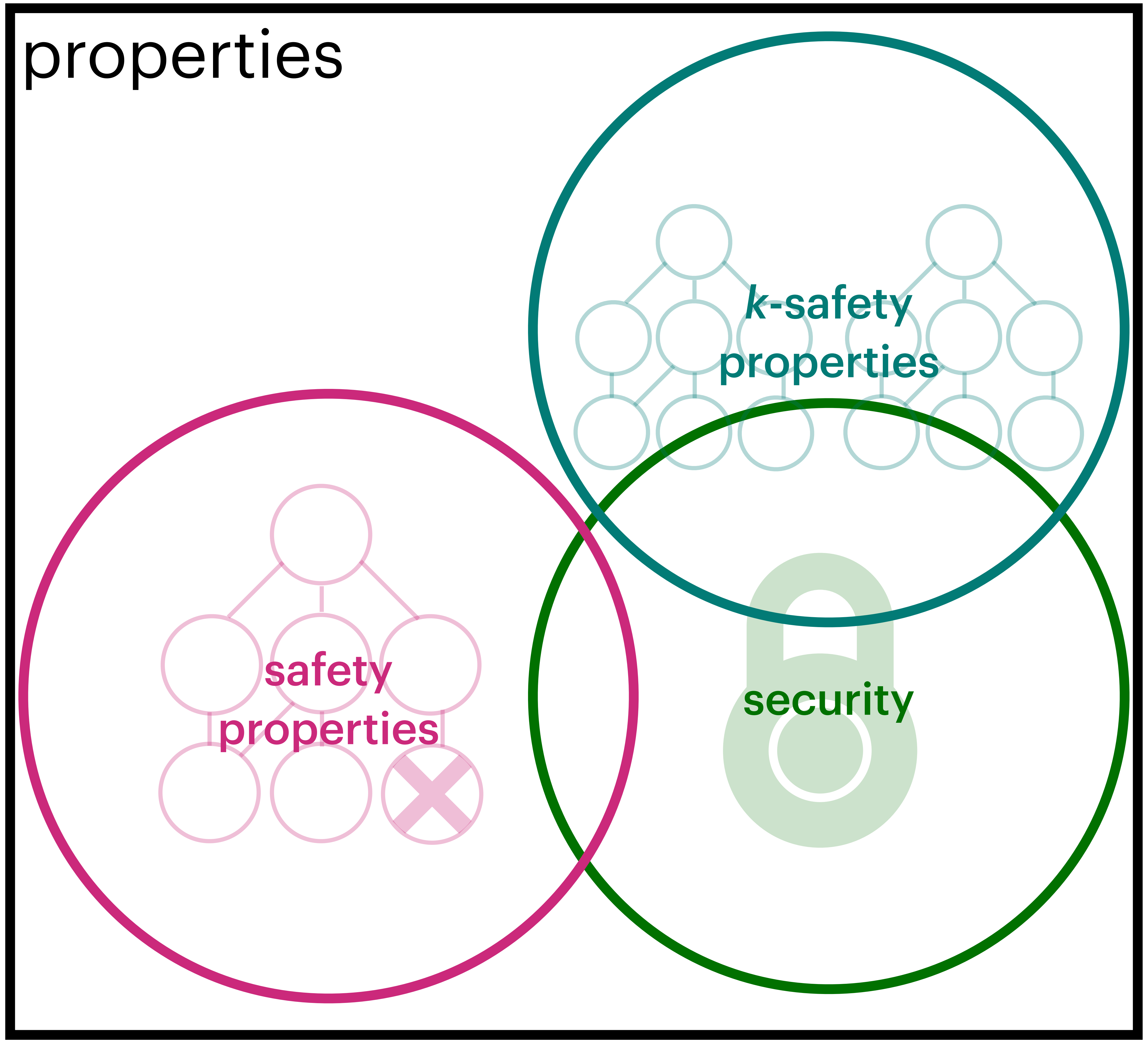


properties

Programs and Properties

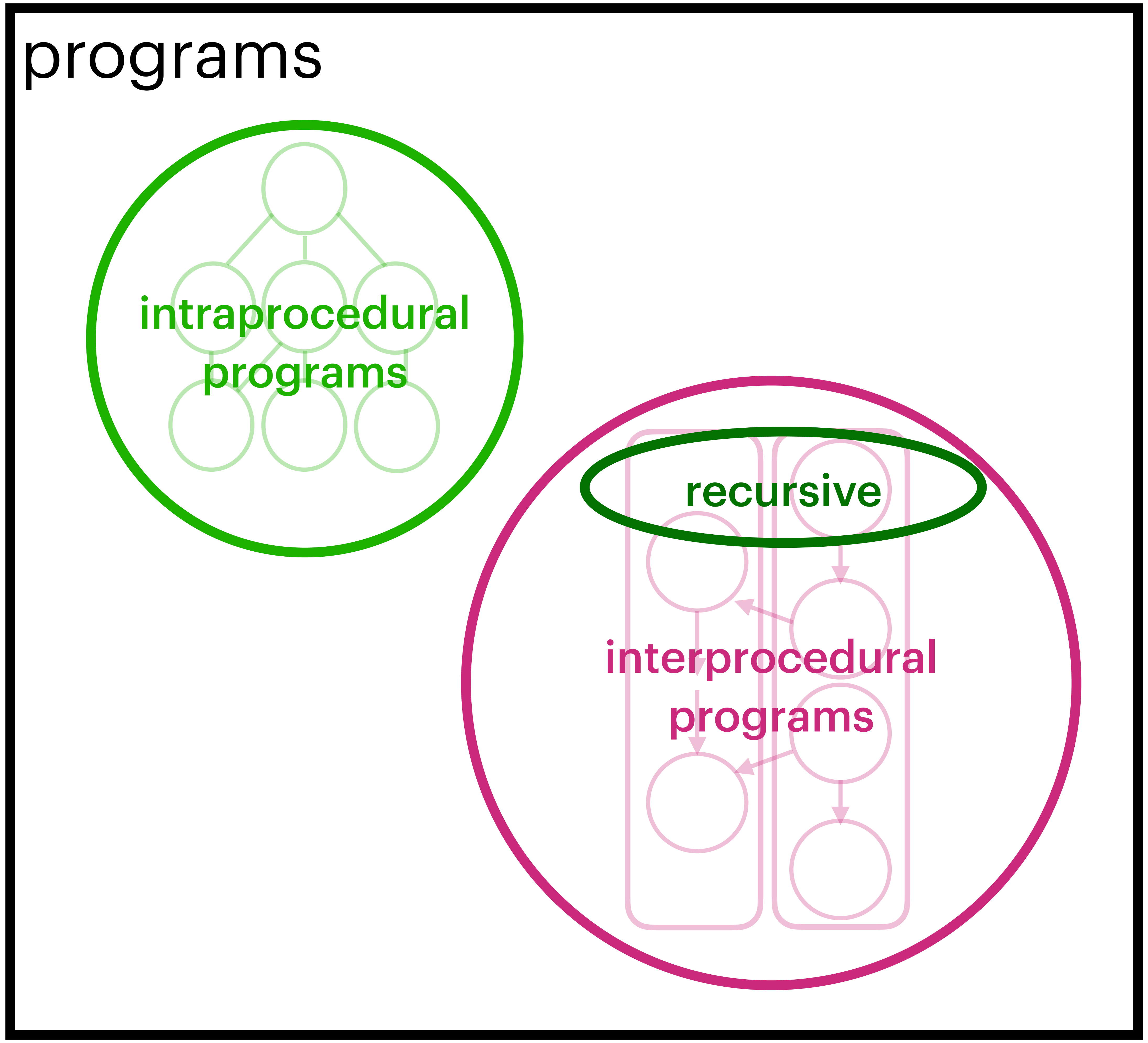
Consider **certain kinds** of programs + properties rather than **general** ones

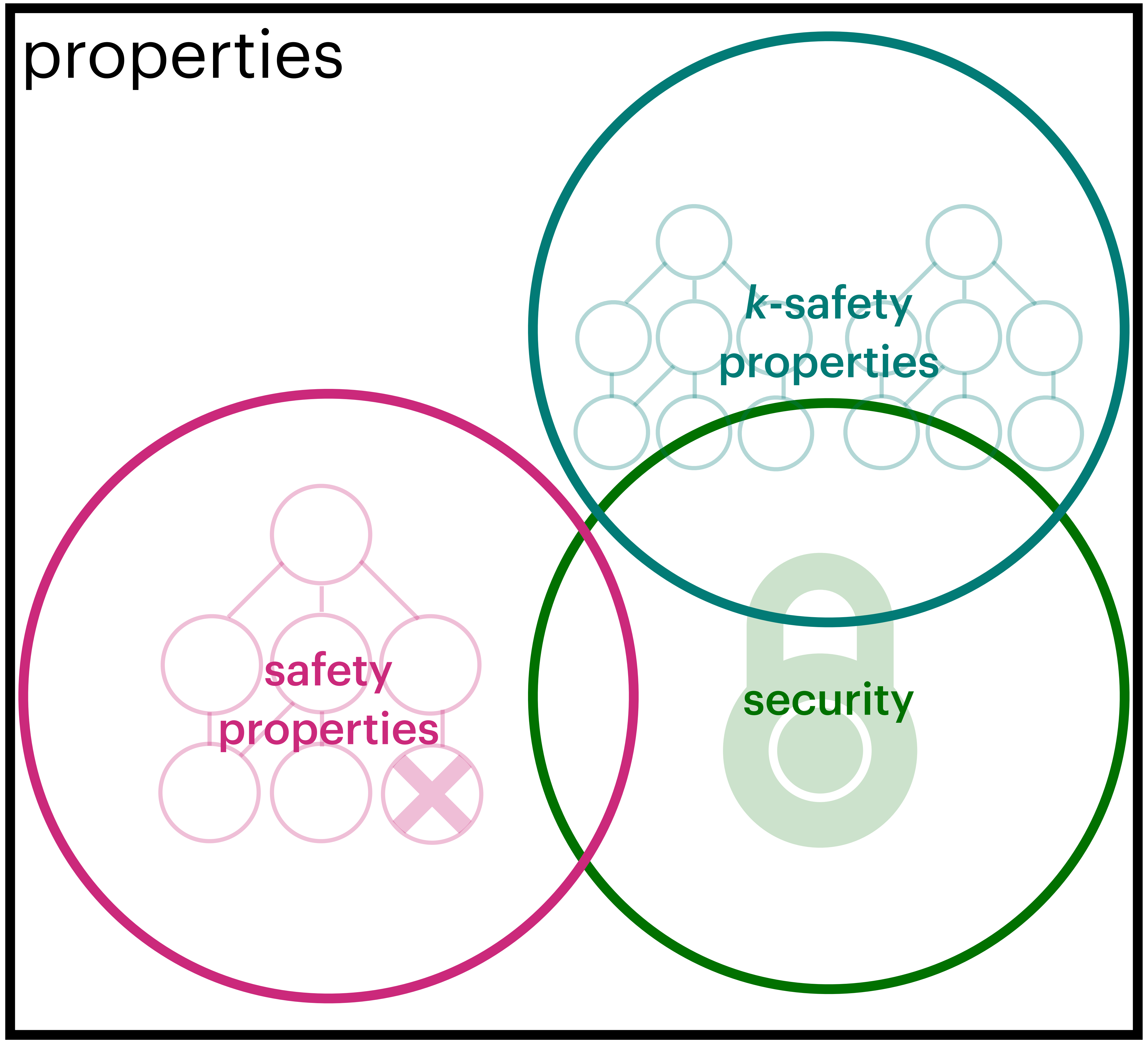




Programs and Properties

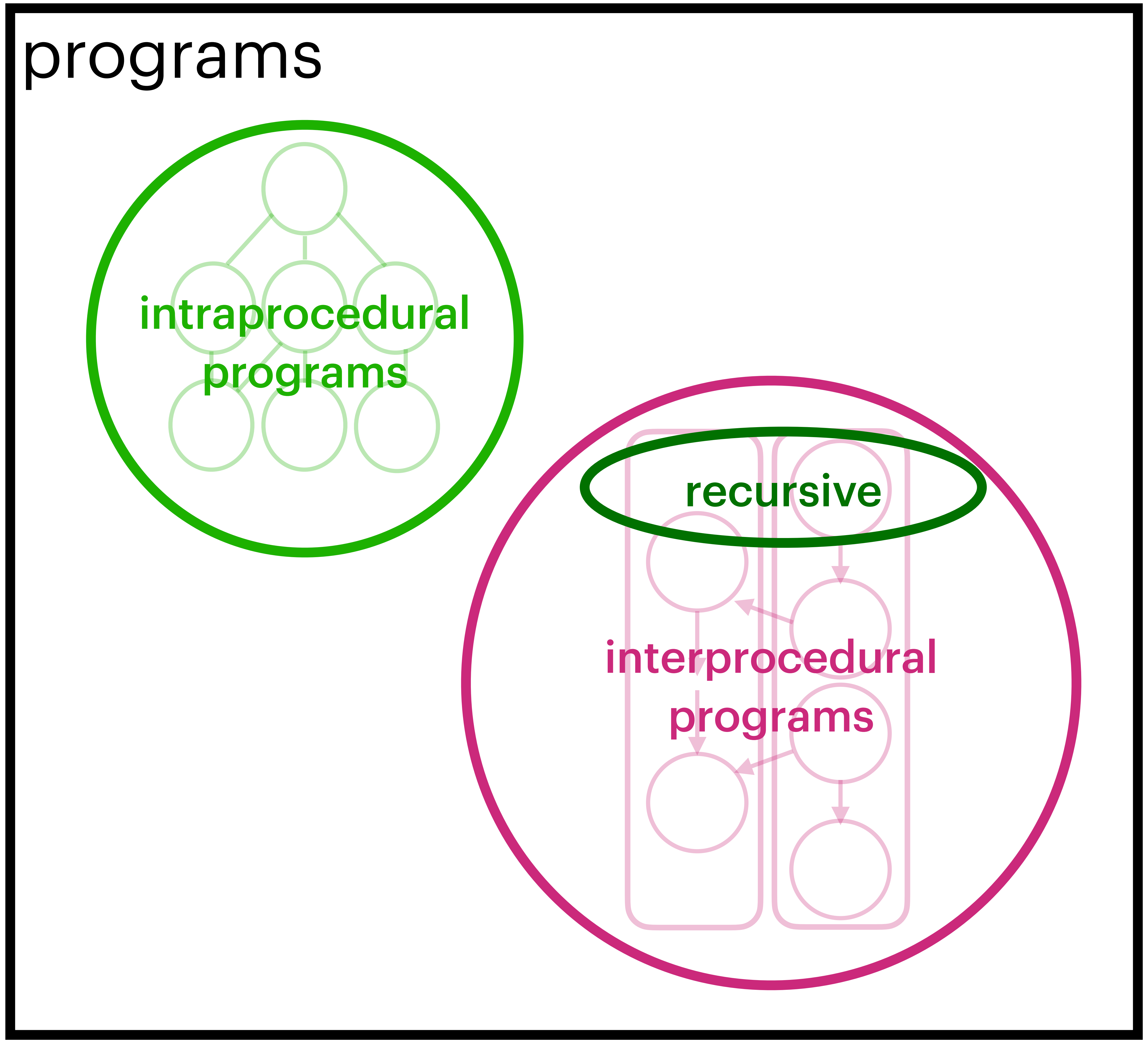
Consider **certain kinds** of programs + properties rather than **general** ones

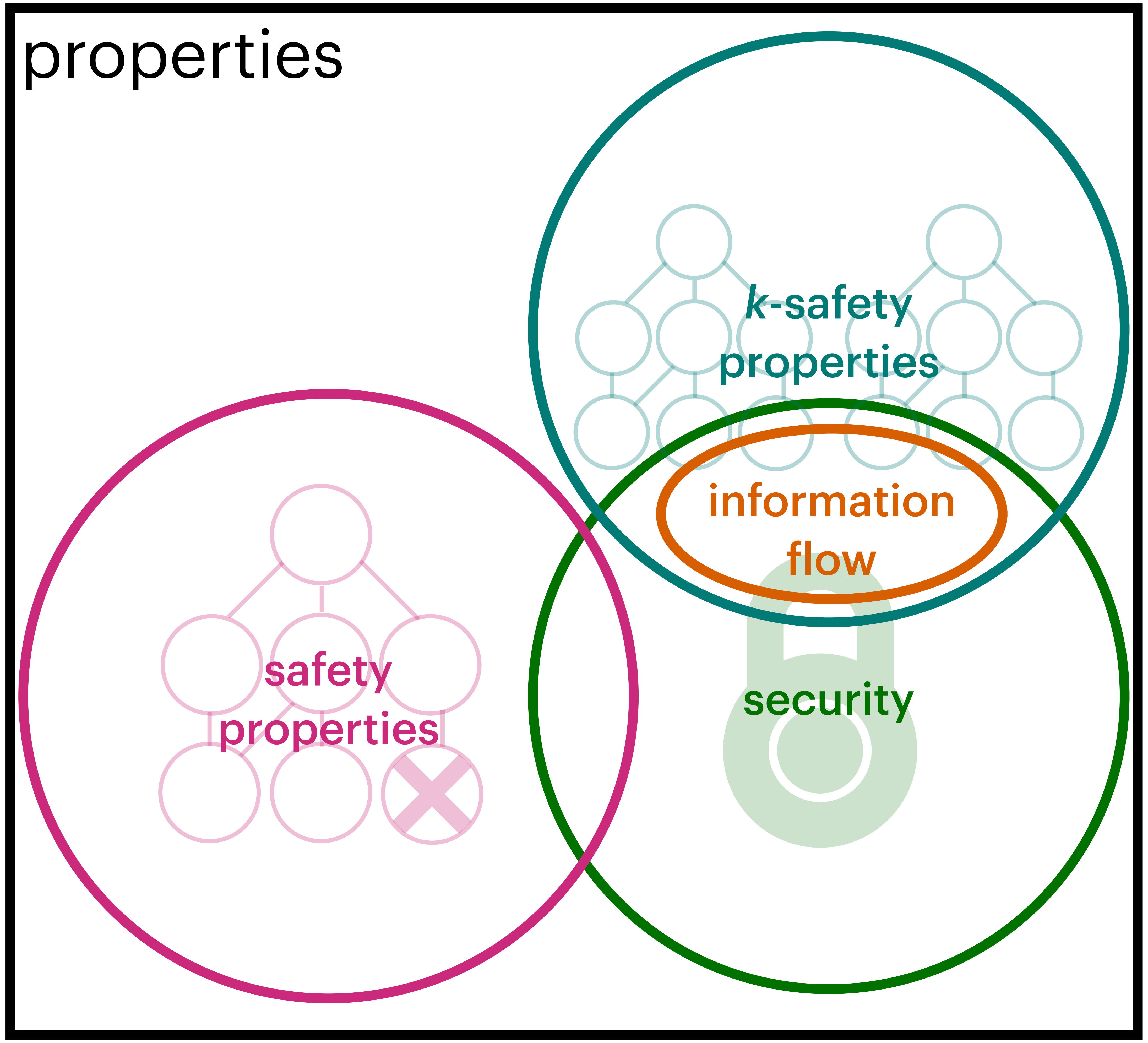




Programs and Properties

Consider **certain kinds** of programs + properties rather than **general** ones





Classes of Verification Problems

I. *k*-safety Verification

II. Interprocedural Program Verification

III. Information-Flow Verification

Classes of Verification Problems

Will talk about each of these in turn

I. *k*-safety Verification

II. Interprocedural Program Verification

III. Information-Flow Verification

Classes of Verification Problems

Will talk about each of these in turn

I. *k*-safety Verification

II. Interprocedural Program Verification

III. Information-Flow Verification

Will talk about the third most detail
(Extra slides on the second)

Classes of Verification Problems

I. *k*-safety Verification

II. Interprocedural Program Verification

III. Information-Flow Verification

Classes of Verification Problems

Brief note about formalisms used to model each class of problems

I. *k*-safety Verification

II. Interprocedural Program Verification

III. Information-Flow Verification

Classes of Verification Problems

Brief note about formalisms used to model each class of problems

I. *k*-safety Verification
Cartesian Hoare Logic

II. Interprocedural Program Verification

III. Information-Flow Verification

Classes of Verification Problems

Brief note about formalisms used to model each class of problems

I. *k*-safety Verification
Cartesian Hoare Logic

II. Interprocedural Program Verification
Constrained Horn Clauses

III. Information-Flow Verification

Classes of Verification Problems

Brief note about formalisms used to model each class of problems

I. *k*-safety Verification
Cartesian Hoare Logic

II. Interprocedural Program Verification
Constrained Horn Clauses

III. Information-Flow Verification
Constrained Horn Clauses

Classes of Verification Problems

Brief note about formalisms used to model each class of problems

I. *k*-safety Verification
Cartesian Hoare Logic

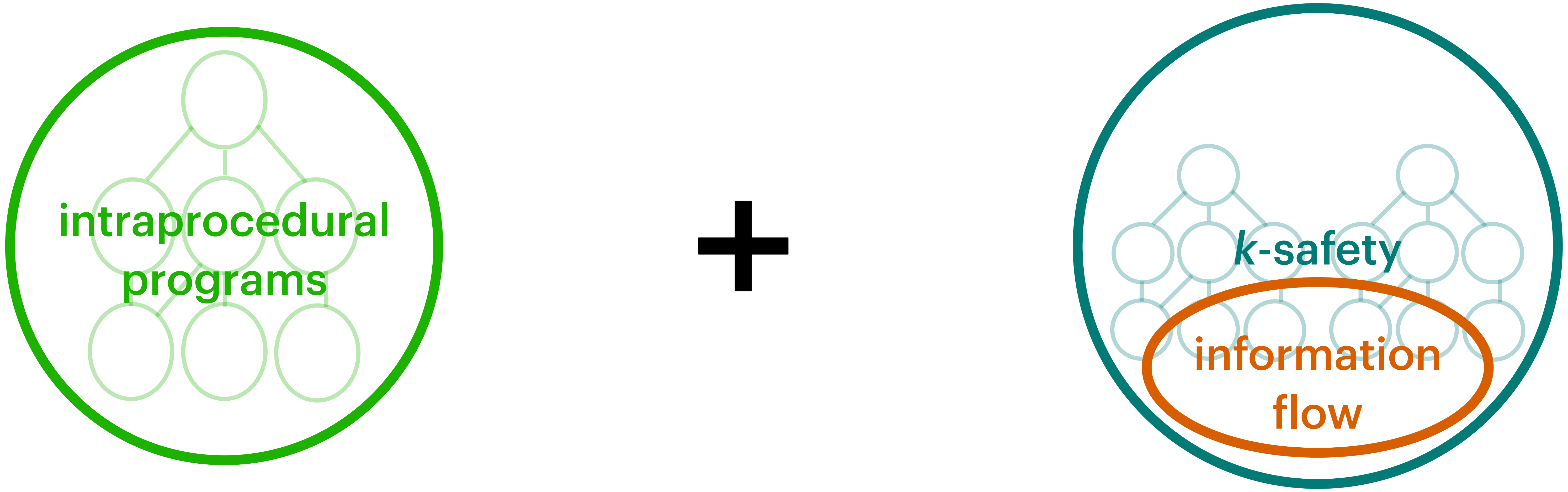
II. Interprocedural Program Verification
Constrained Horn Clauses

III. Information-Flow Verification
Constrained Horn Clauses

- No (specialized) heap modeling
- No higher-order functions
- Static call graph

Cartesian Hoare Logic, Sousa and Dillig, PLDI'16

I. *k*-safety Verification

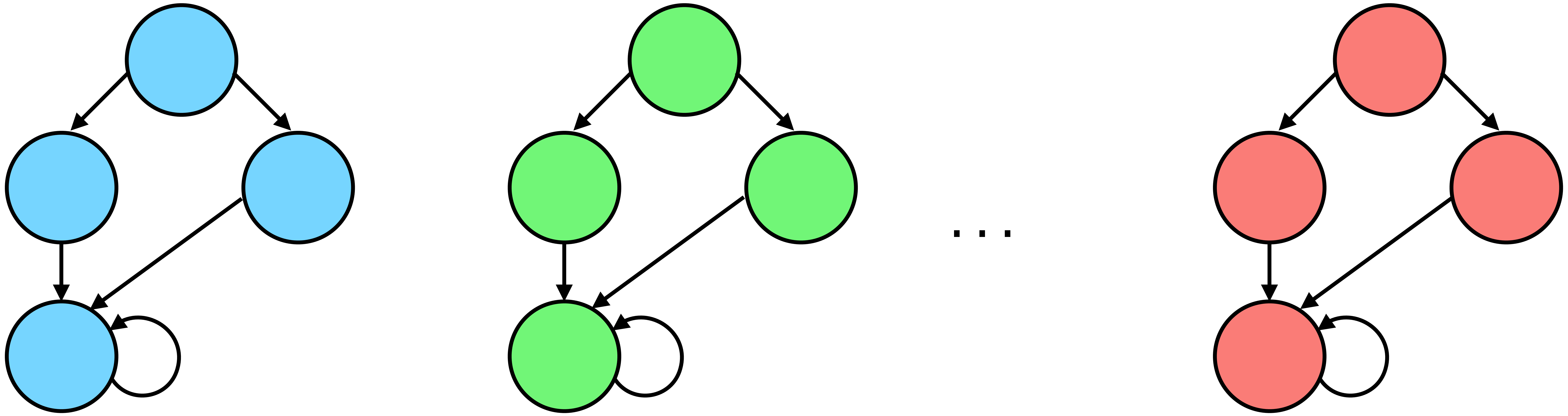


Single-procedure programs (may contain loops)

Properties over k copies of the same program

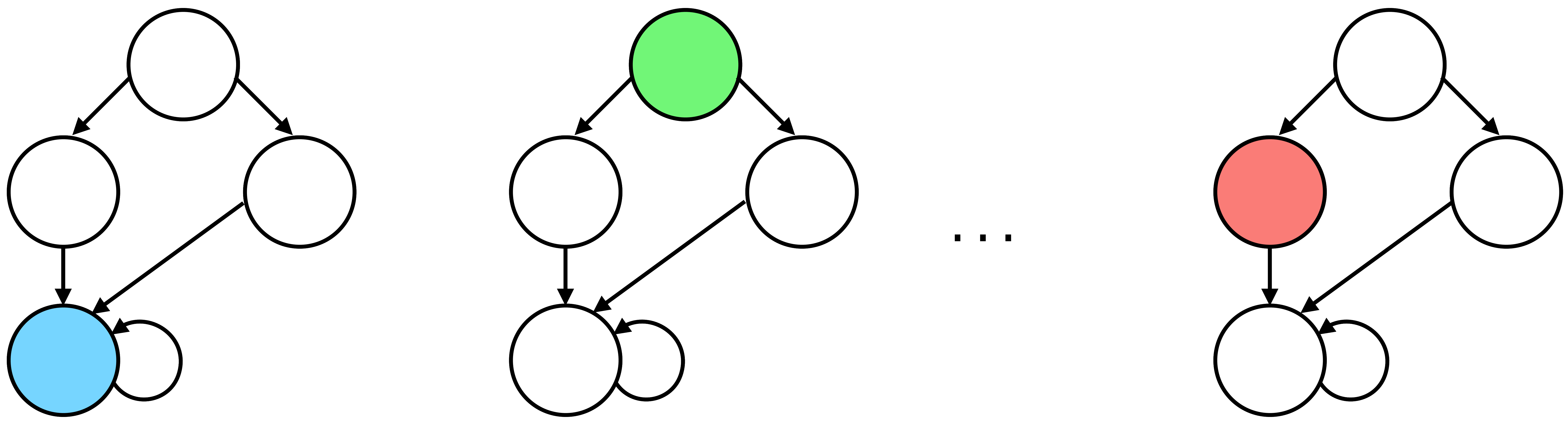
Relational Invariants

Relate the *k* program copies at intermediate points



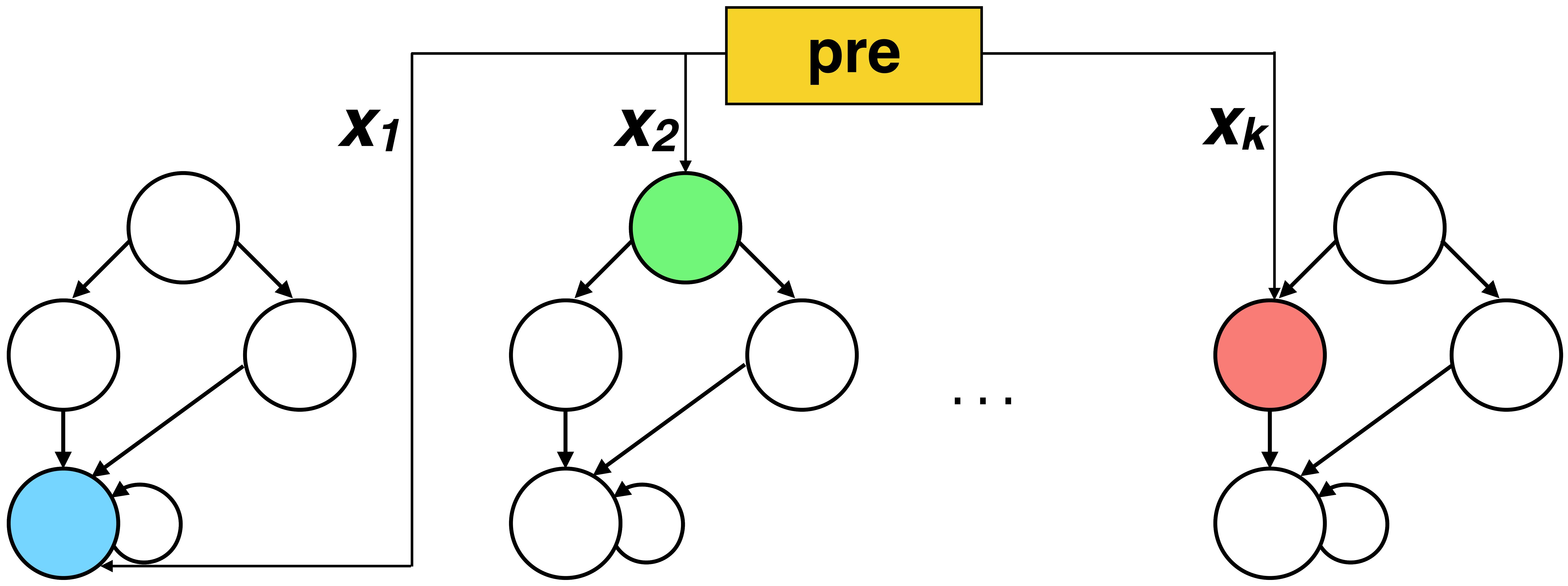
Relational Invariants

Relate the *k* program copies at intermediate points



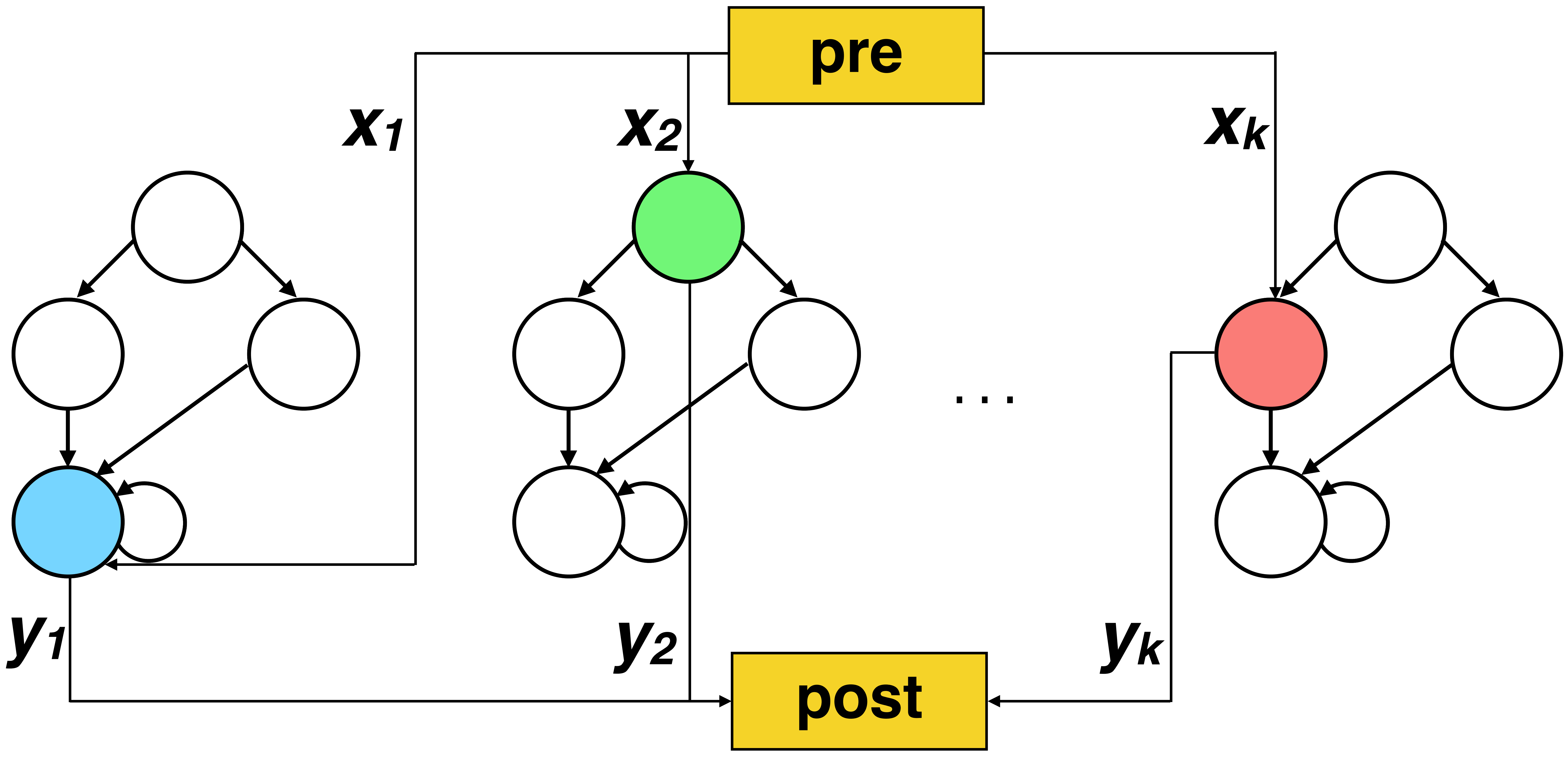
Relational Invariants

Relate the *k* program copies at intermediate points



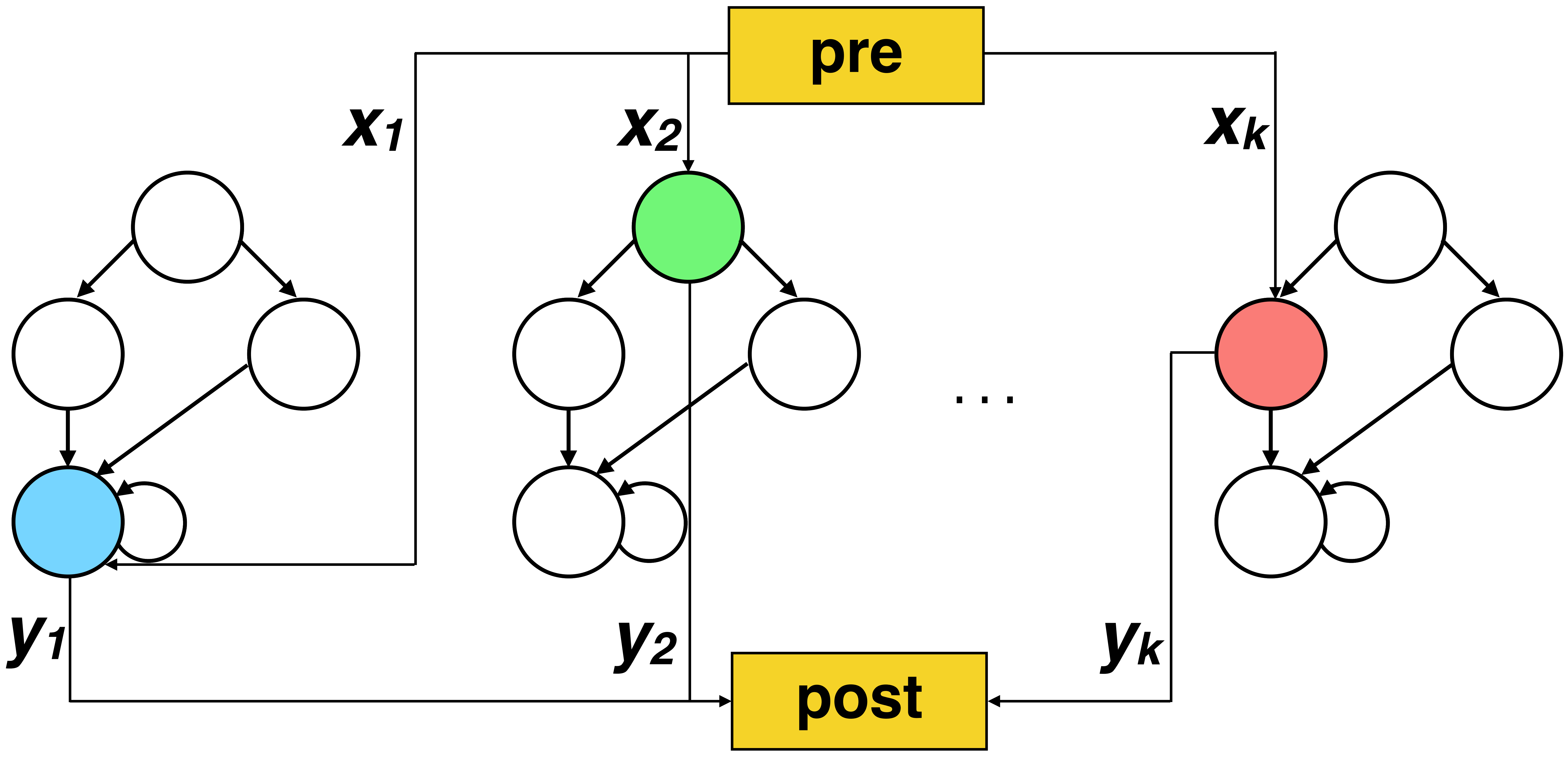
Relational Invariants

Relate the k program copies at intermediate points



Relational Invariants

Relate the *k* program copies at intermediate points



How to **leverage** and how (where) to **infer** them for scalable verification?

Symmetry and Synchrony

L. Pick, G. Fedyukovich, A. Gupta. Exploiting Synchrony and Symmetry in Relational Verification, CAV'18

Symmetry and Synchrony

How to **leverage** relational properties?

L. Pick, G. Fedyukovich, A. Gupta. Exploiting Synchrony and Symmetry in Relational Verification, CAV'18

Symmetry and Synchrony

How to **leverage** relational properties?

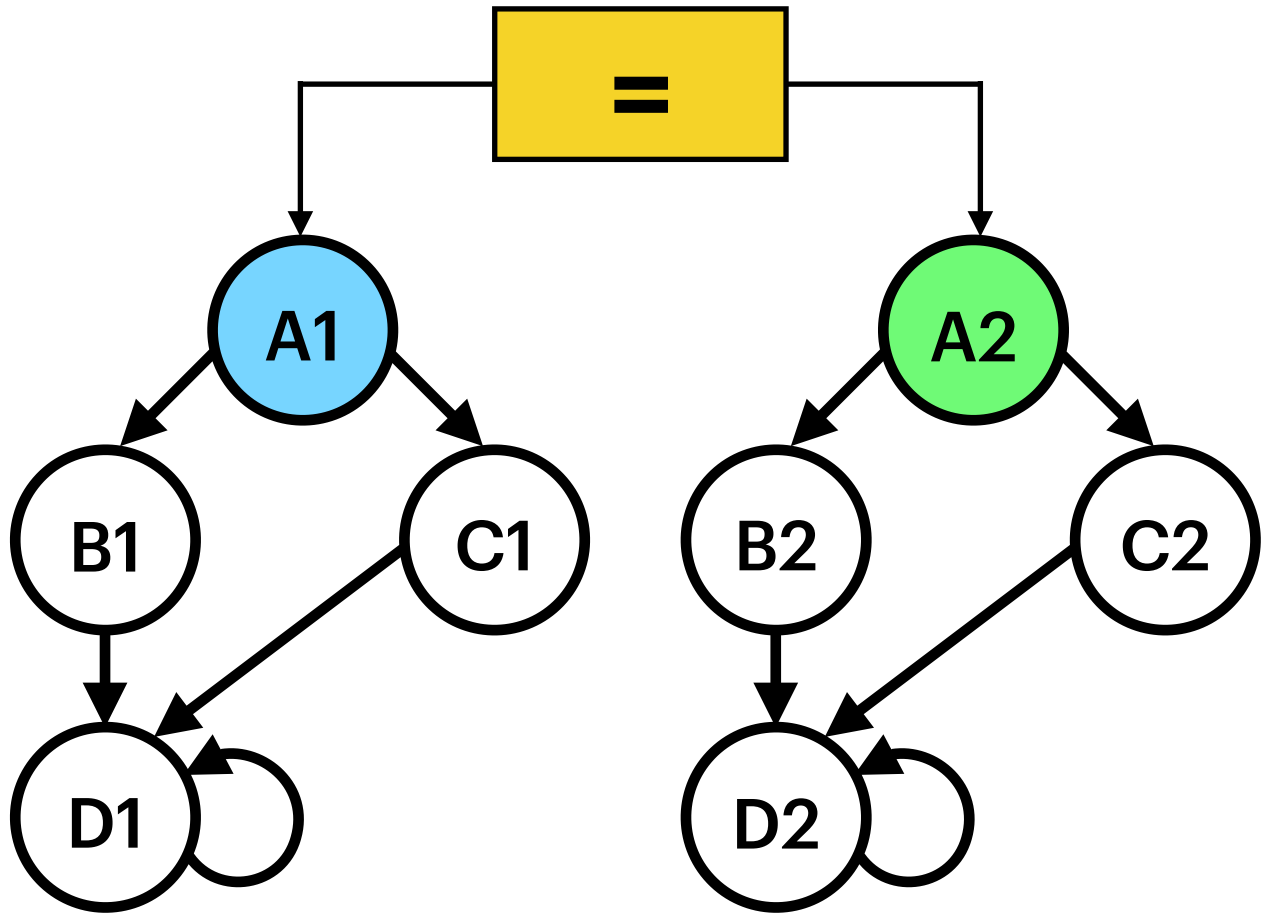
Symmetries in properties lead to redundant subtasks, so **prune** them

L. Pick, G. Fedyukovich, A. Gupta. Exploiting Synchrony and Symmetry in Relational Verification, CAV'18

Symmetry and Synchrony

How to **leverage** relational properties?

Symmetries in properties lead to redundant subtasks, so **prune** them

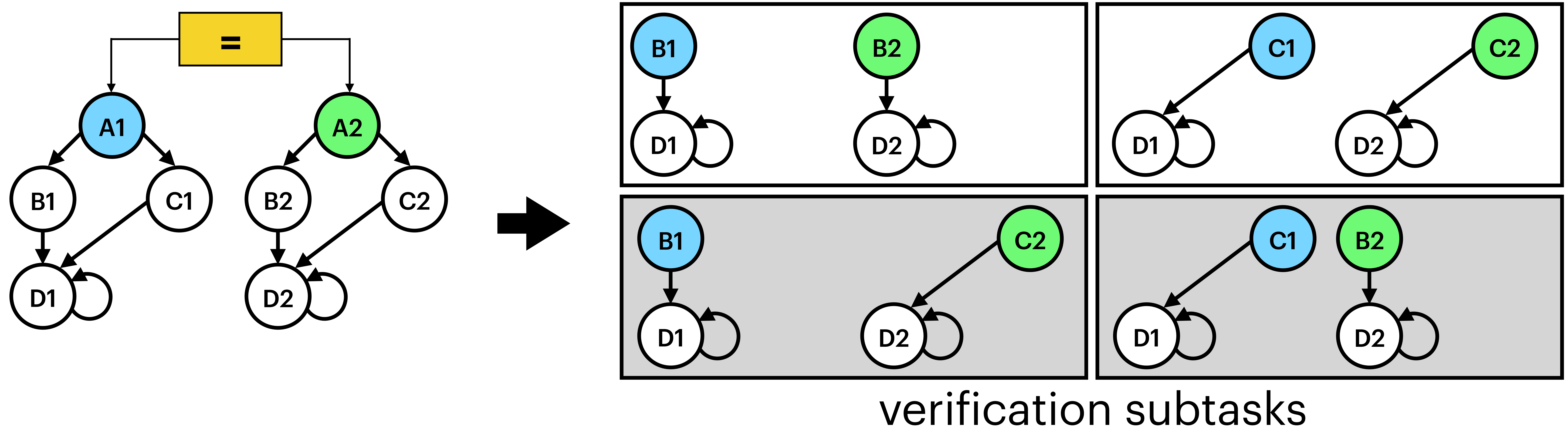


L. Pick, G. Fedyukovich, A. Gupta. Exploiting Synchrony and Symmetry in Relational Verification, CAV'18

Symmetry and Synchrony

How to **leverage** relational properties?

Symmetries in properties lead to redundant subtasks, so **prune** them

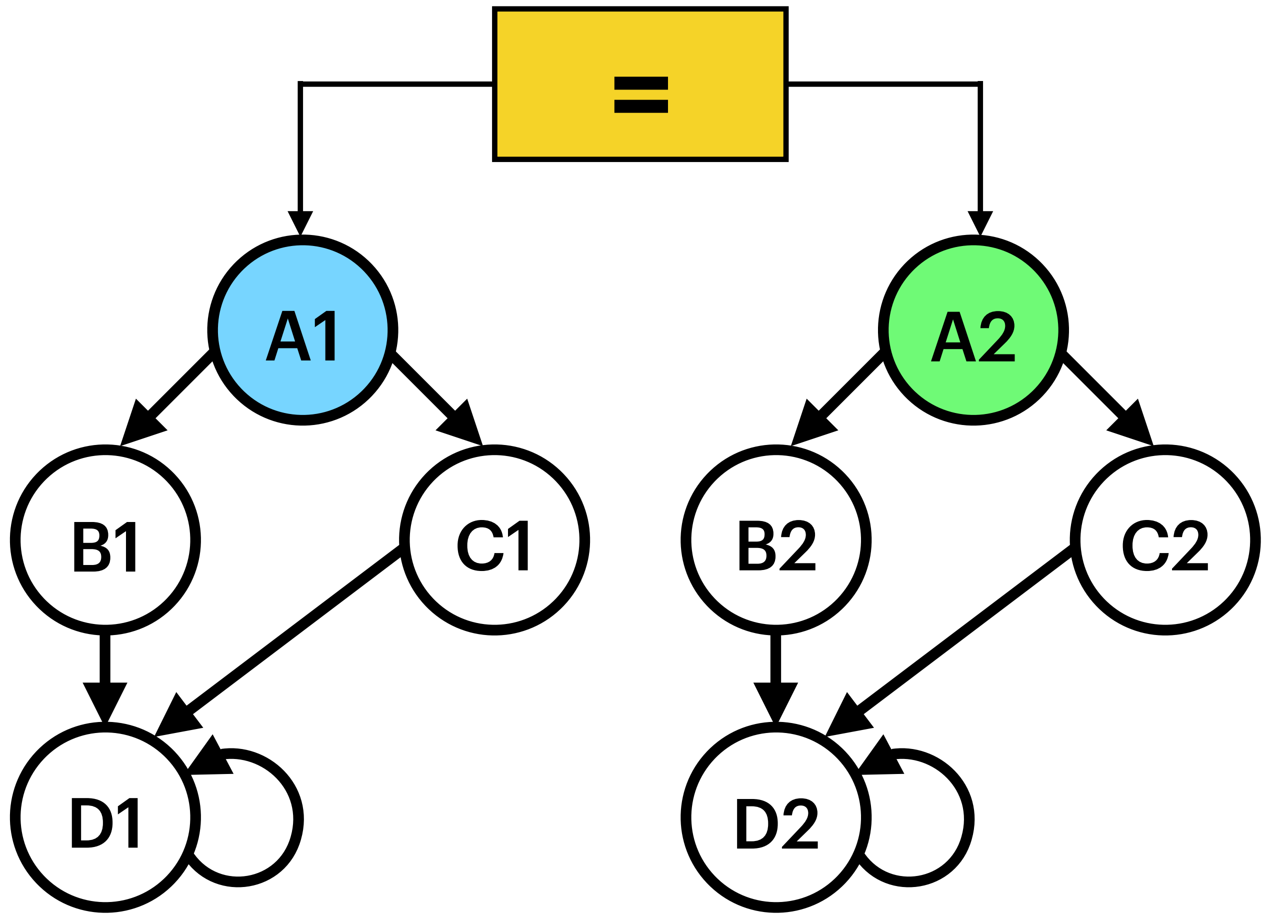


L. Pick, G. Fedyukovich, A. Gupta. Exploiting Synchrony and Symmetry in Relational Verification, CAV'18

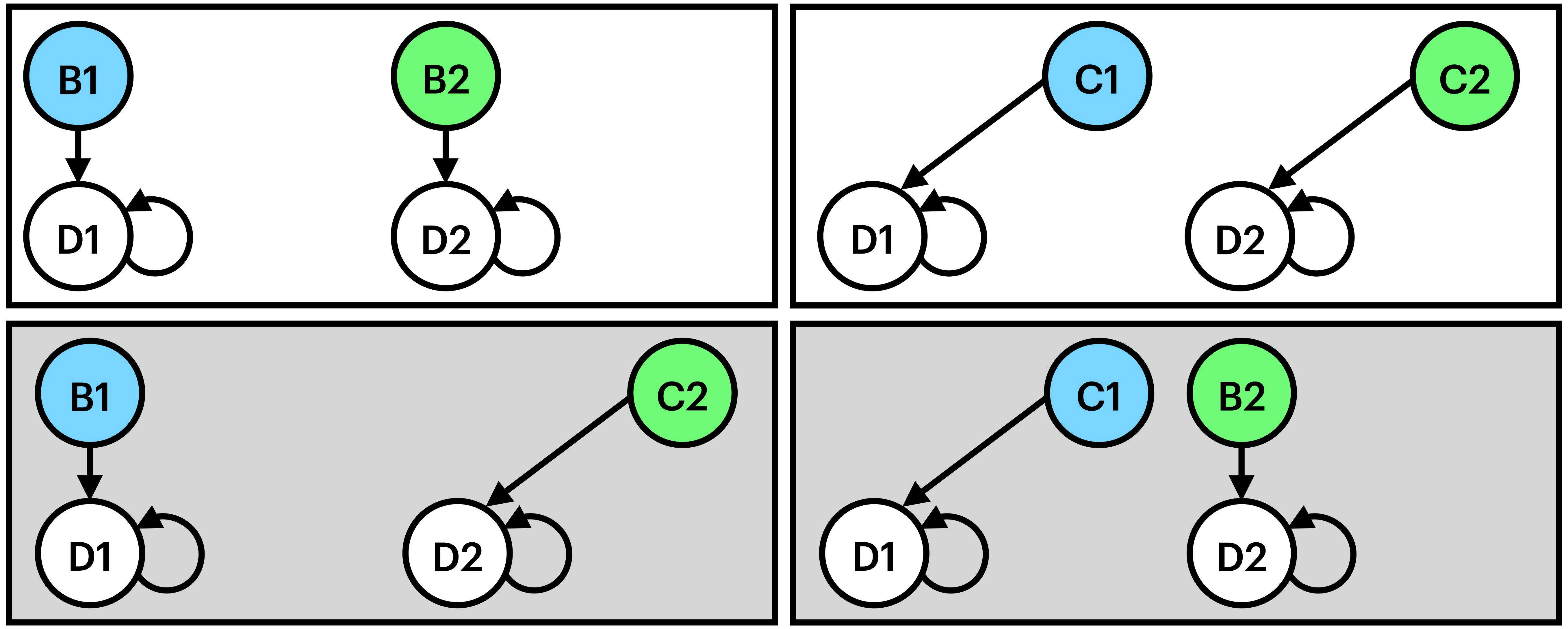
Symmetry and Synchrony

How to **leverage** relational properties?

Symmetries in properties lead to redundant subtasks, so **prune** them



Property symmetry 1↔2



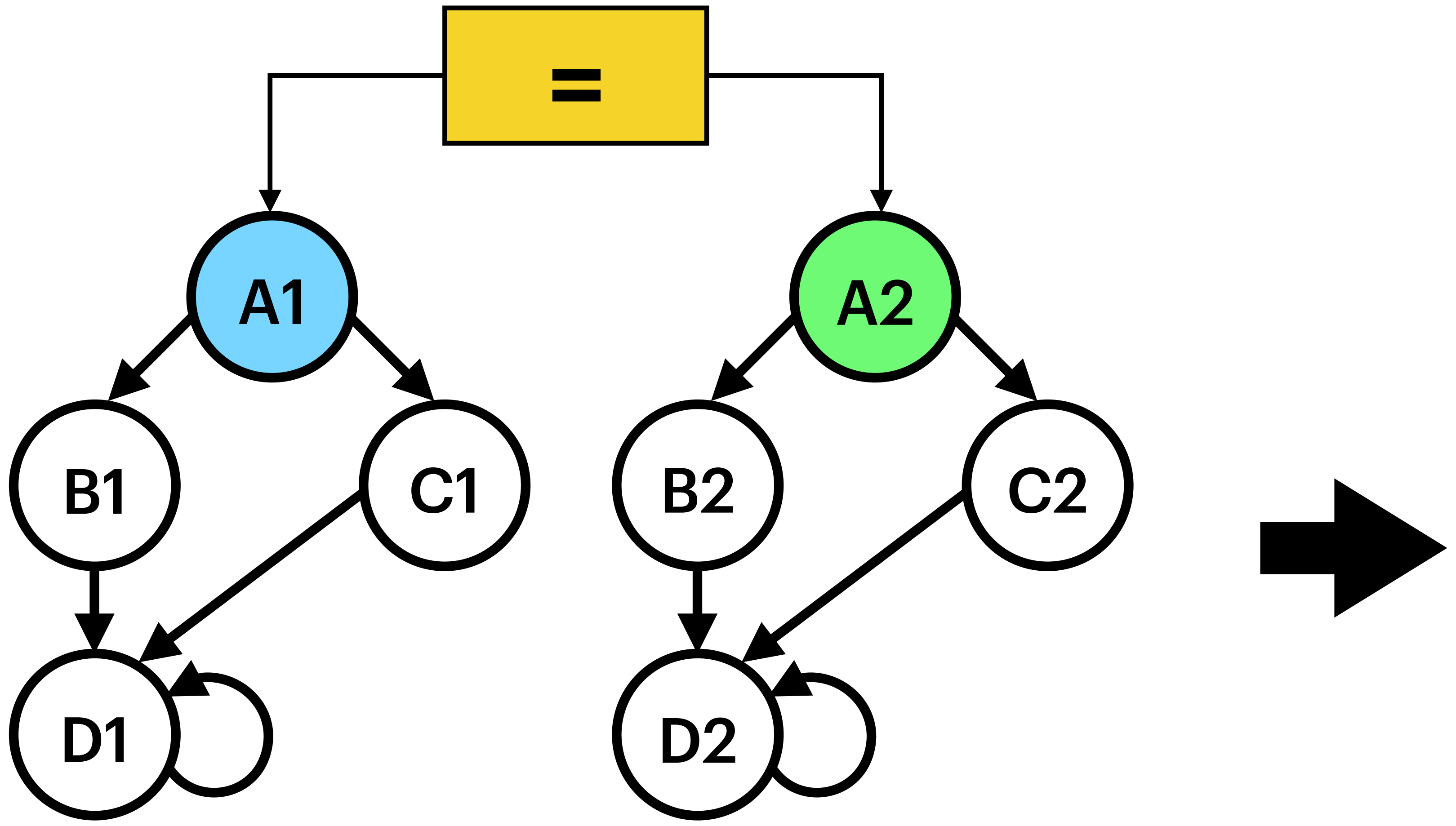
verification subtasks

L. Pick, G. Fedyukovich, A. Gupta. Exploiting Synchrony and Symmetry in Relational Verification, CAV'18

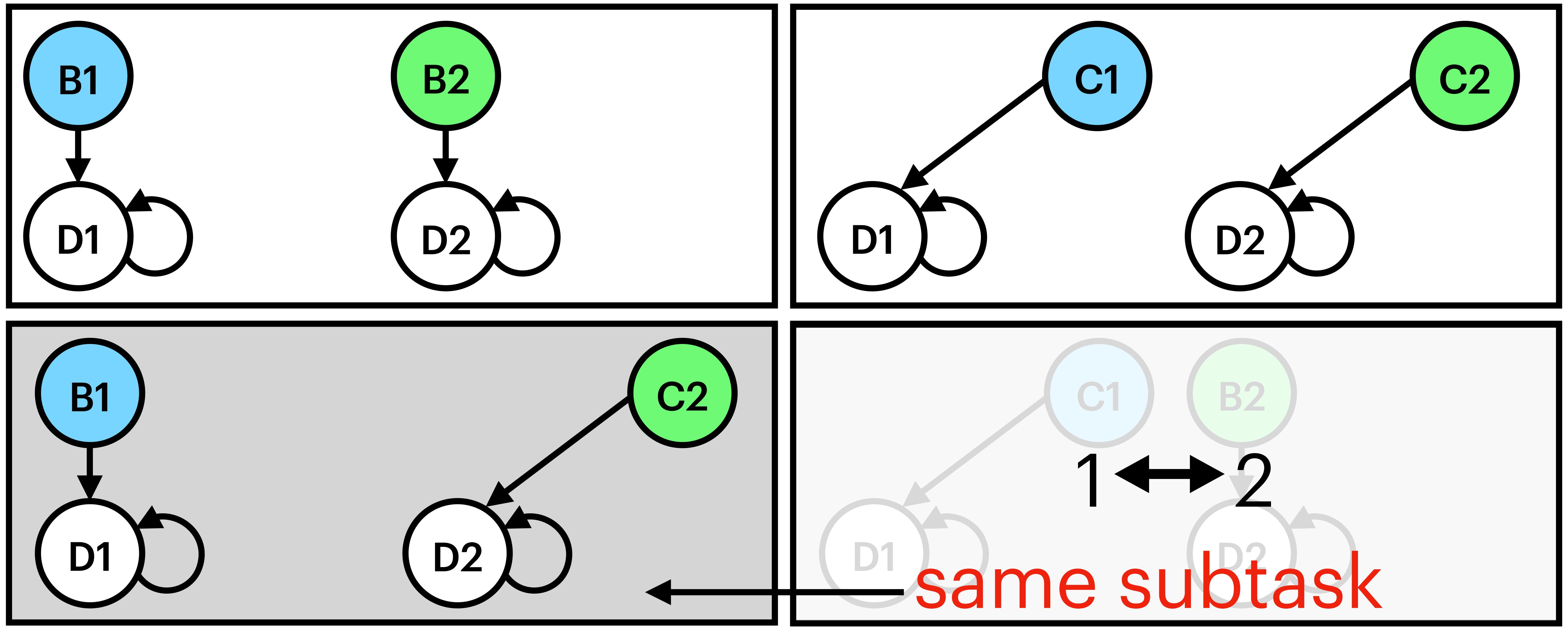
Symmetry and Synchrony

How to **leverage** relational properties?

Symmetries in properties lead to redundant subtasks, so **prune** them



Property symmetry 1↔2



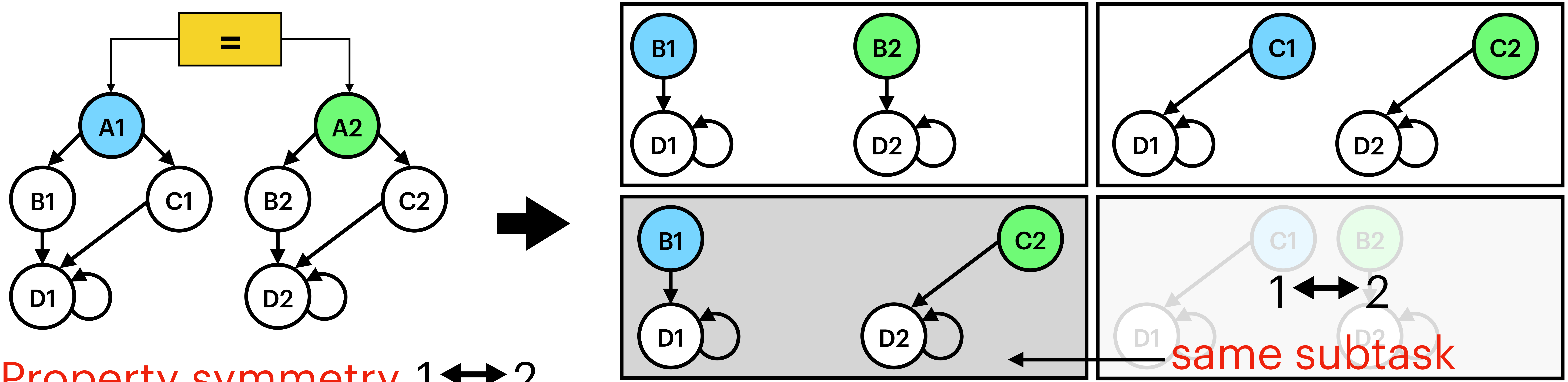
verification subtasks

L. Pick, G. Fedyukovich, A. Gupta. Exploiting Synchrony and Symmetry in Relational Verification, CAV'18

Symmetry and Synchrony

How to **leverage** relational properties?

Symmetries in properties lead to redundant subtasks, so **prune** them



Property symmetry 1↔2

verification subtasks

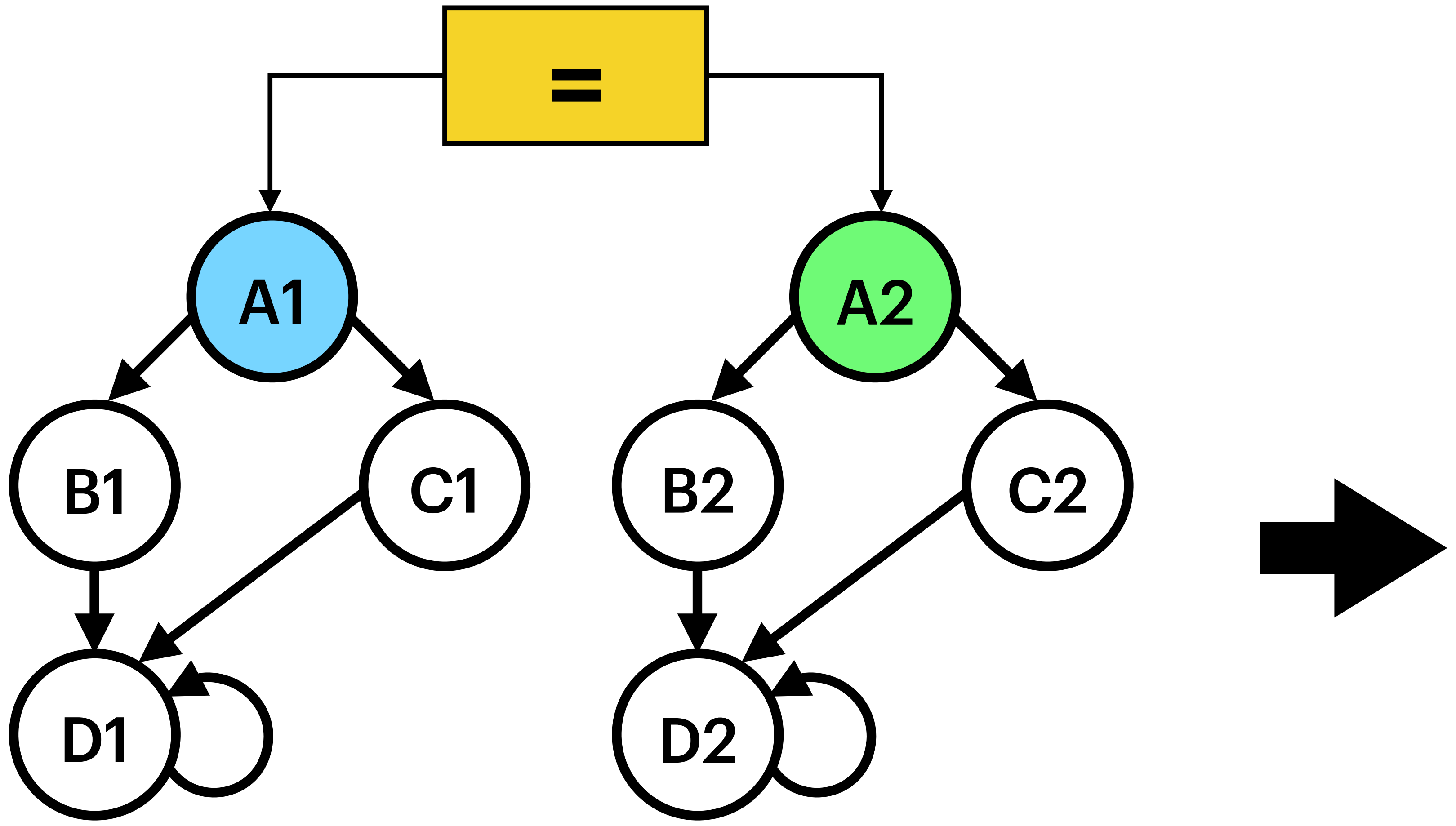
How to **infer** relational properties?

L. Pick, G. Fedyukovich, A. Gupta. Exploiting Synchrony and Symmetry in Relational Verification, CAV'18

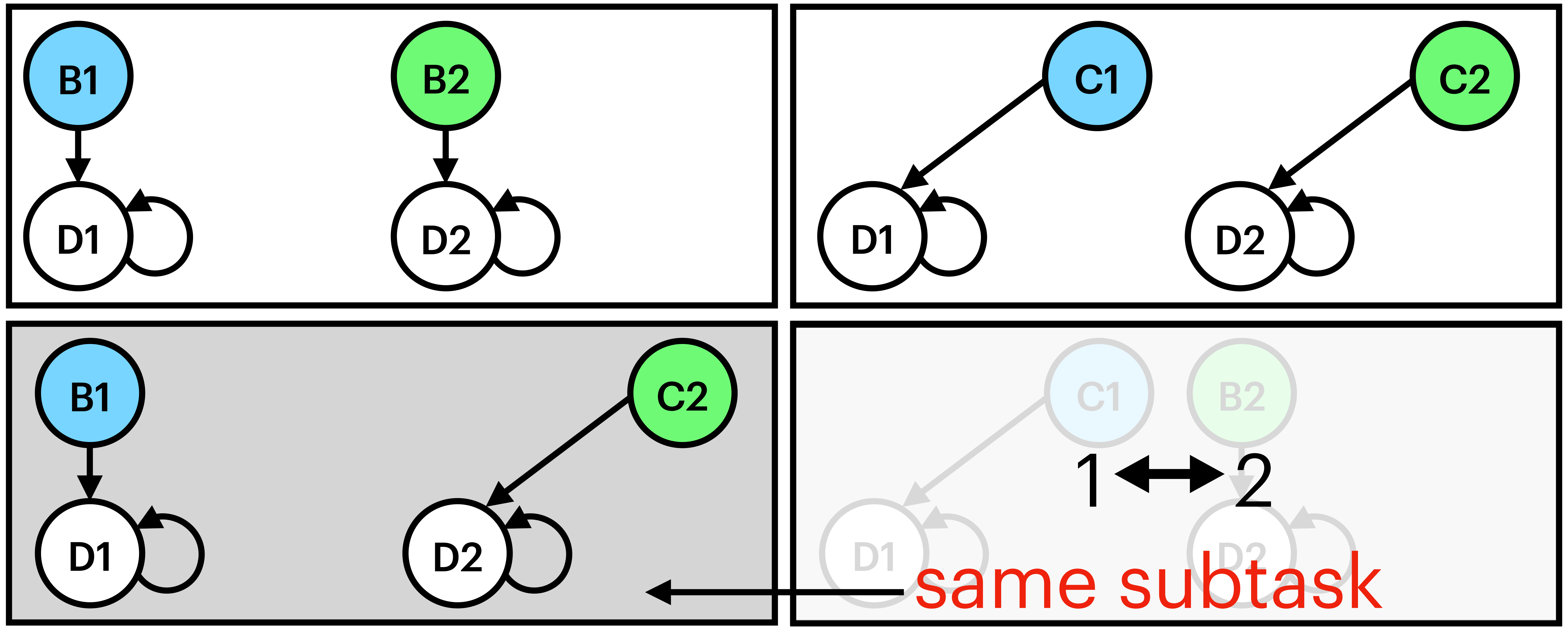
Symmetry and Synchrony

How to **leverage** relational properties?

Symmetries in properties lead to redundant subtasks, so **prune** them



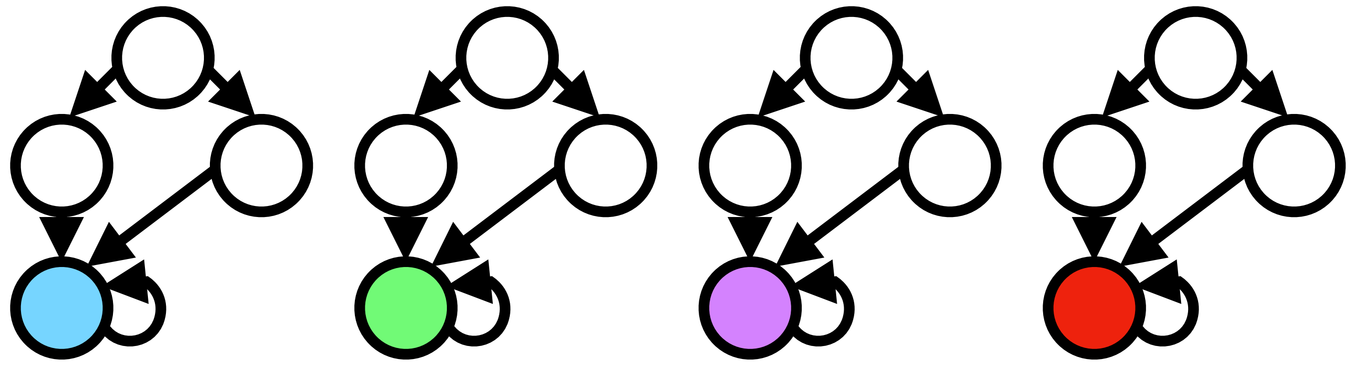
Property symmetry 1⟷2



verification subtasks

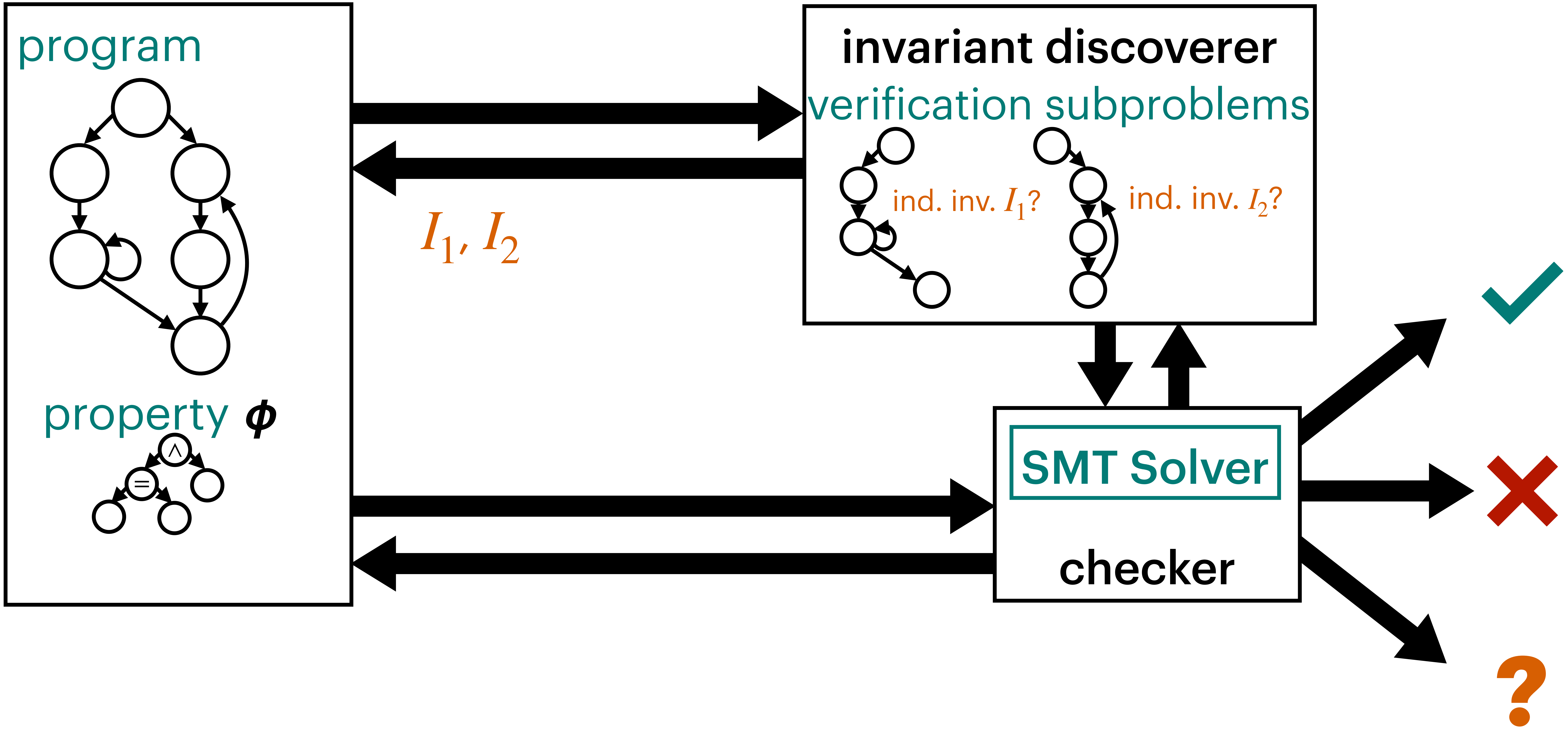
How to **infer** relational properties?

Use **synchrony** technique for loops for fewer and simpler invariants

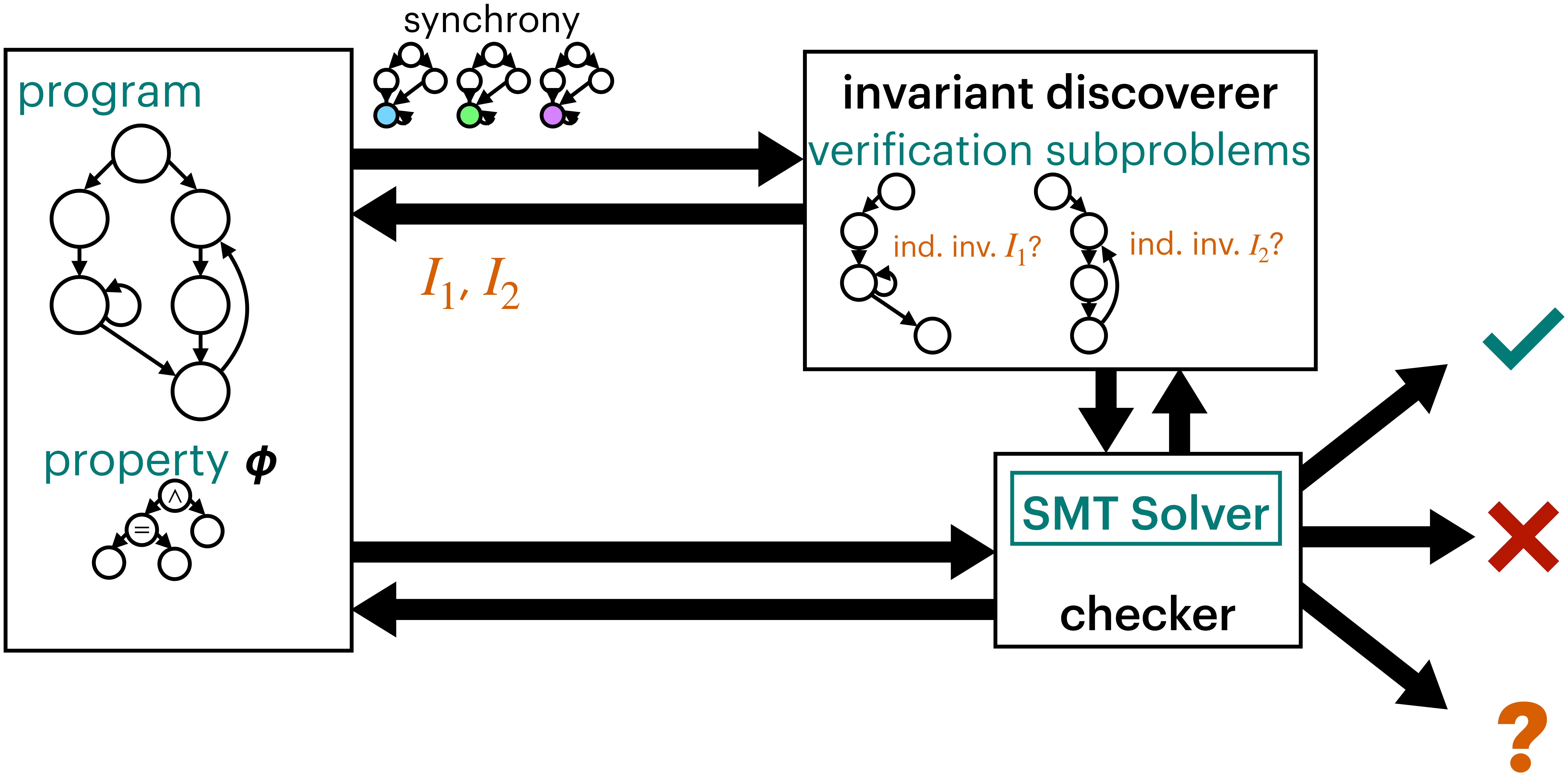


L. Pick, G. Fedyukovich, A. Gupta. Exploiting Synchrony and Symmetry in Relational Verification, CAV'18

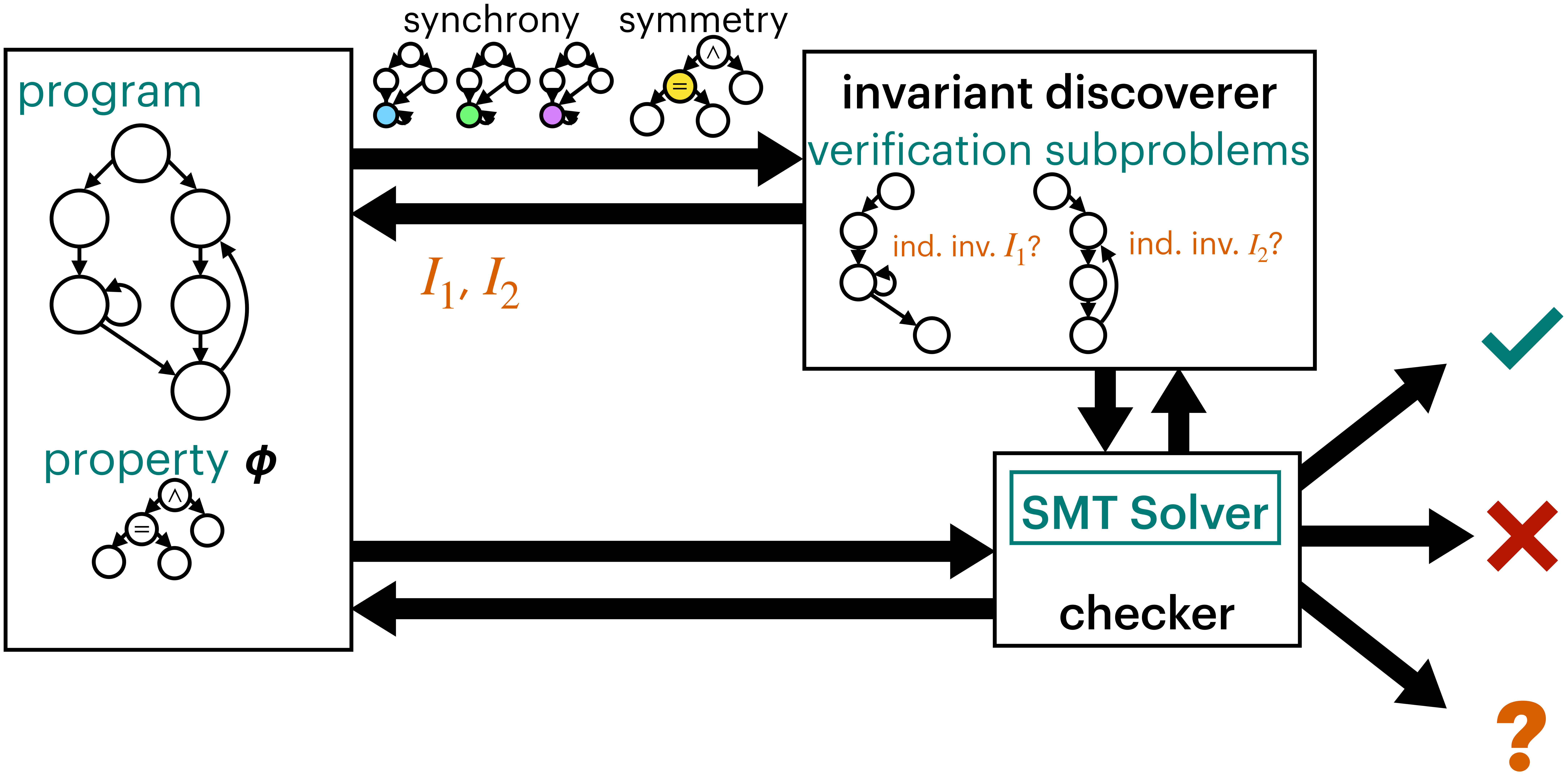
k-safety Verification



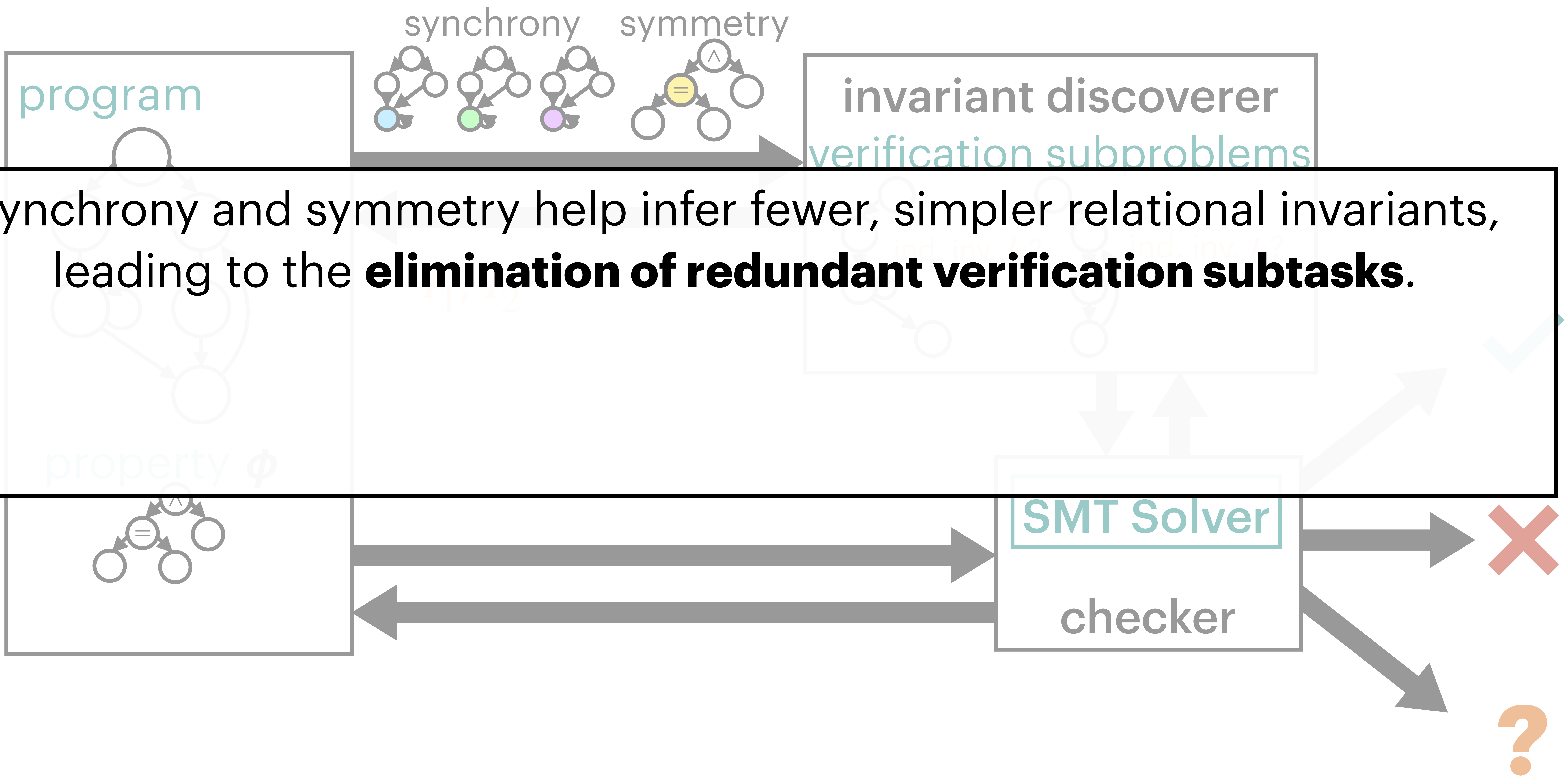
k-safety Verification



k-safety Verification

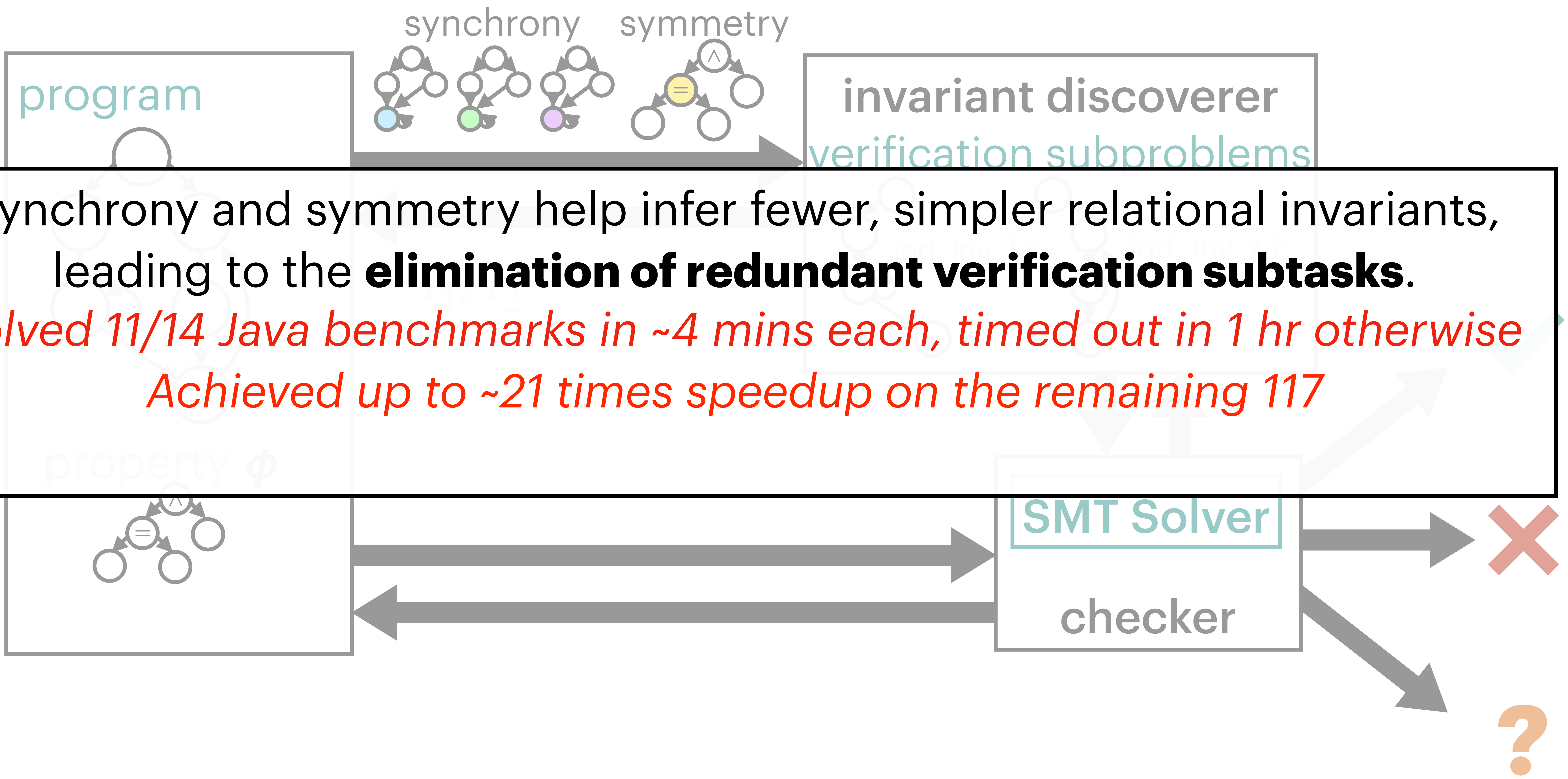


k-safety Verification



Synchrony and symmetry help infer fewer, simpler relational invariants, leading to the **elimination of redundant verification subtasks**.

k-safety Verification

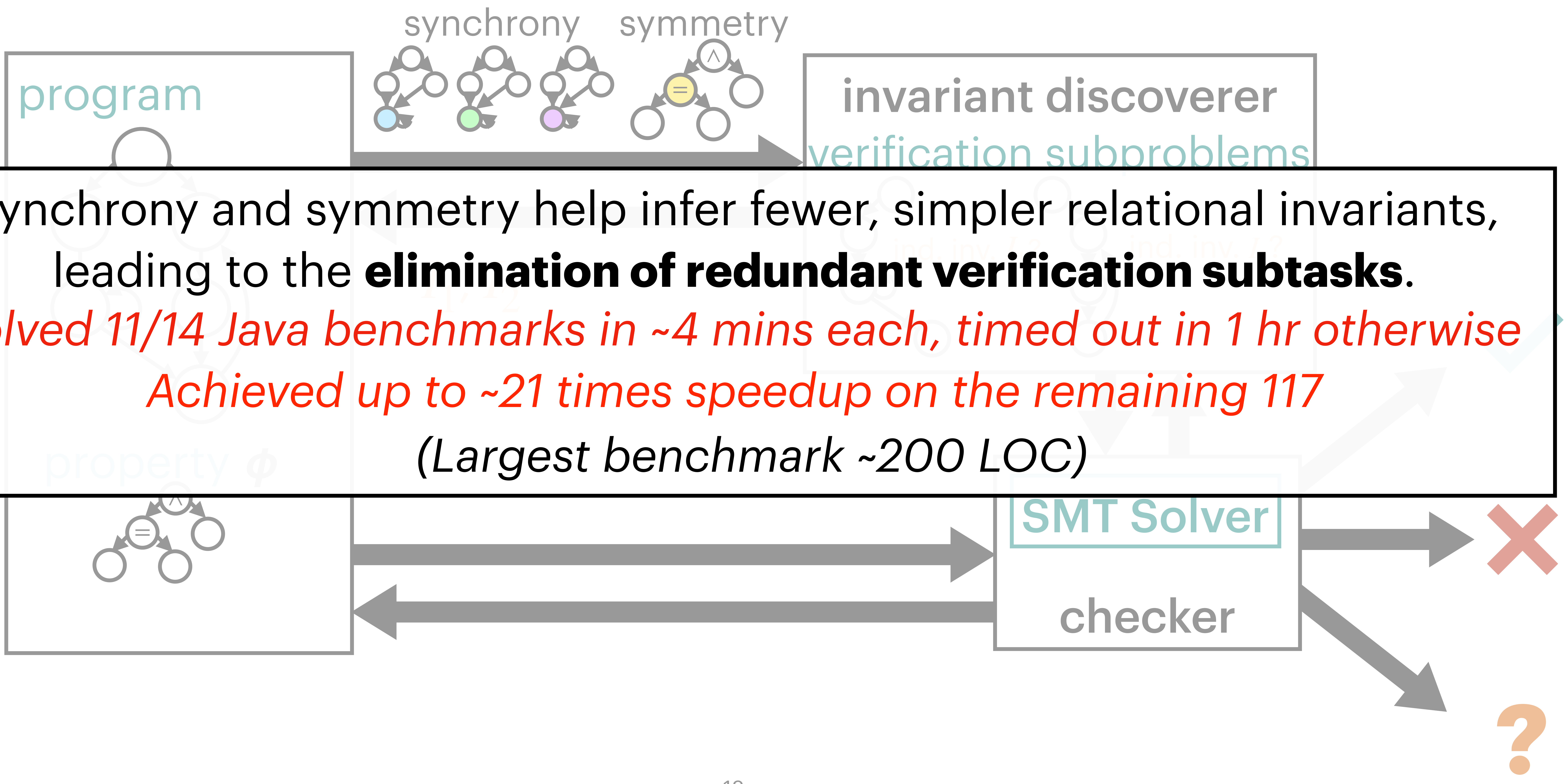


Synchrony and symmetry help infer fewer, simpler relational invariants, leading to the **elimination of redundant verification subtasks**.

Solved 11/14 Java benchmarks in ~4 mins each, timed out in 1 hr otherwise

Achieved up to ~21 times speedup on the remaining 117

k-safety Verification



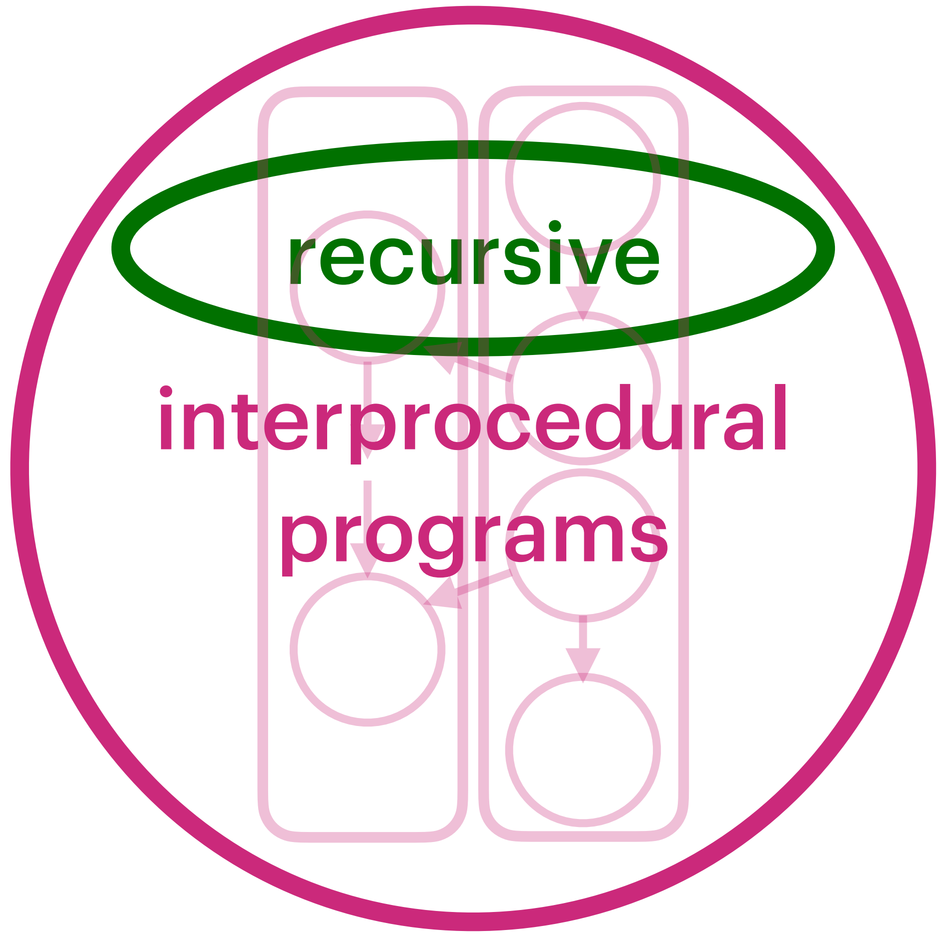
Synchrony and symmetry help infer fewer, simpler relational invariants, leading to the **elimination of redundant verification subtasks**.

Solved 11/14 Java benchmarks in ~4 mins each, timed out in 1 hr otherwise

Achieved up to ~21 times speedup on the remaining 117

(Largest benchmark ~200 LOC)

II. Interprocedural Program Verification

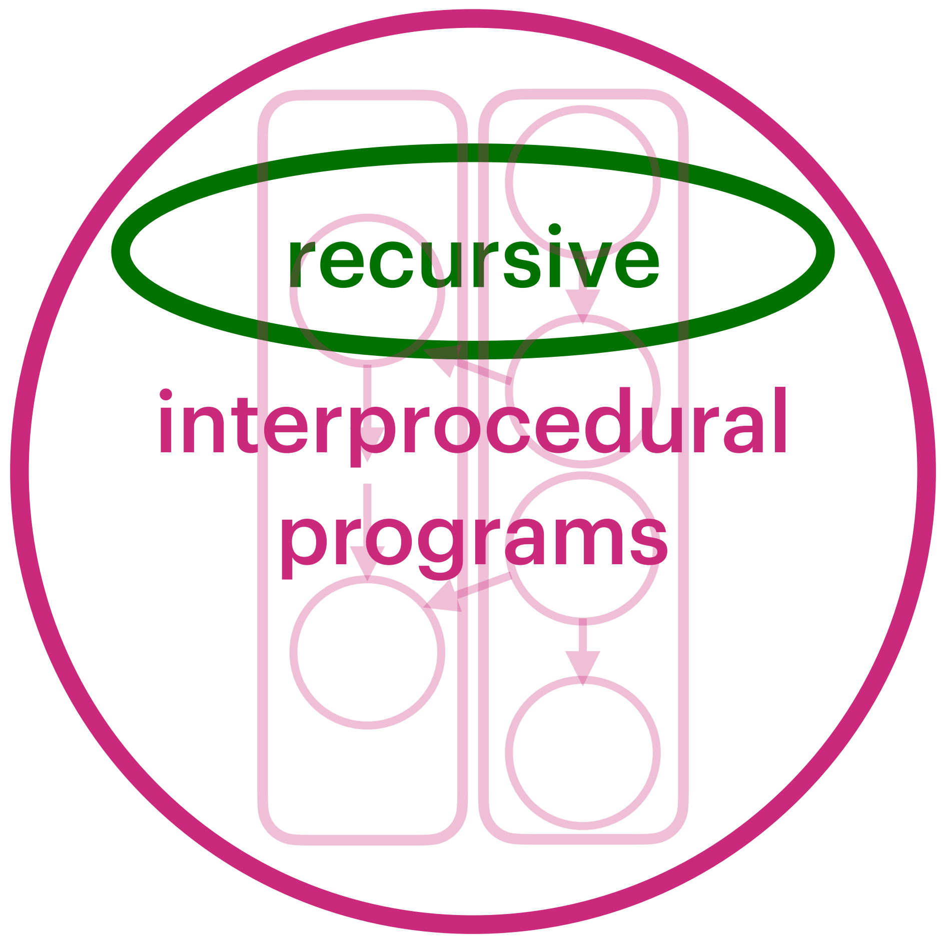


+

Multiple-procedure programs
(may contain recursion)

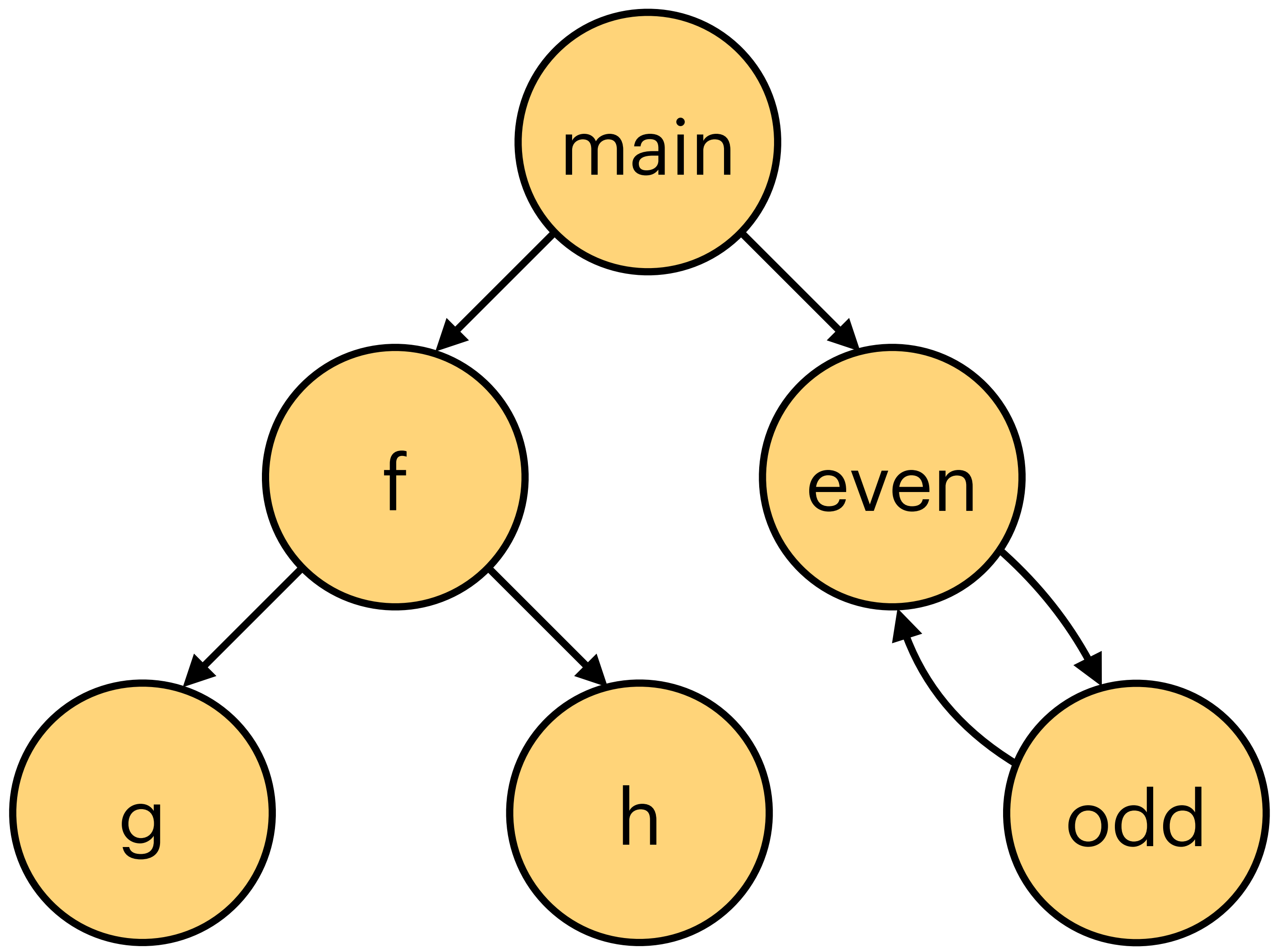
General safety properties
(hoisted to entry procedure)

Interprocedural Programs

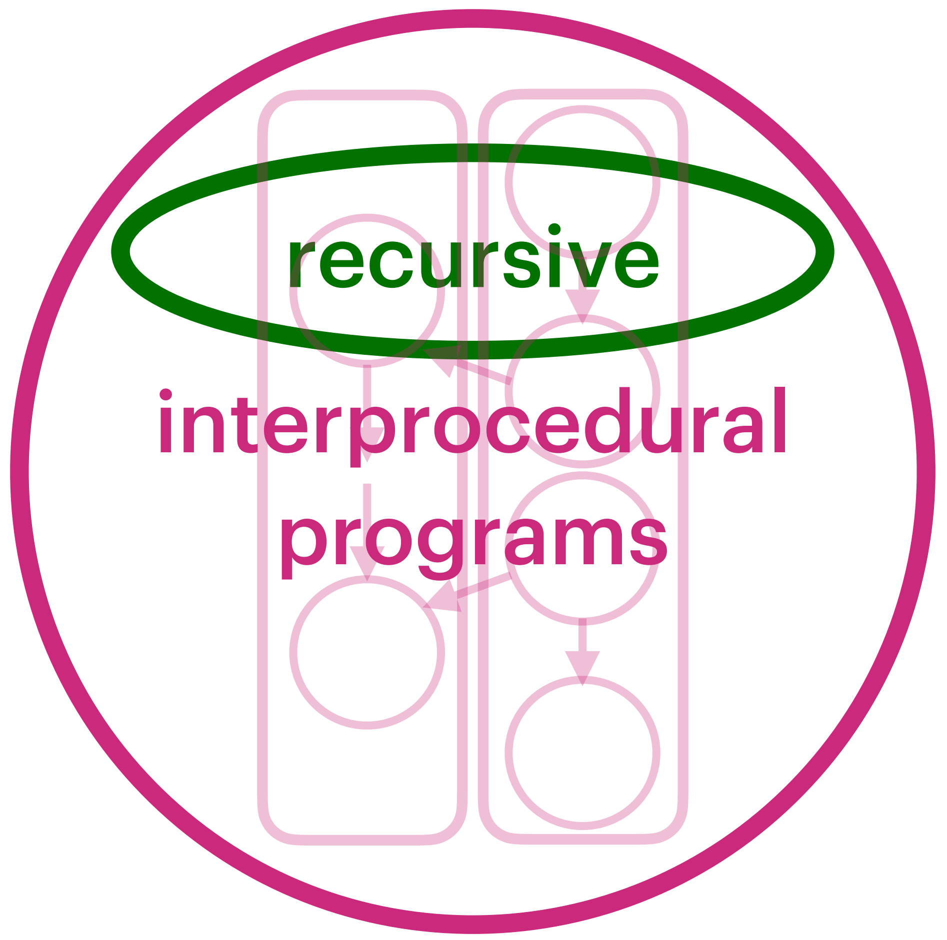


Have call graphs

Example call graph

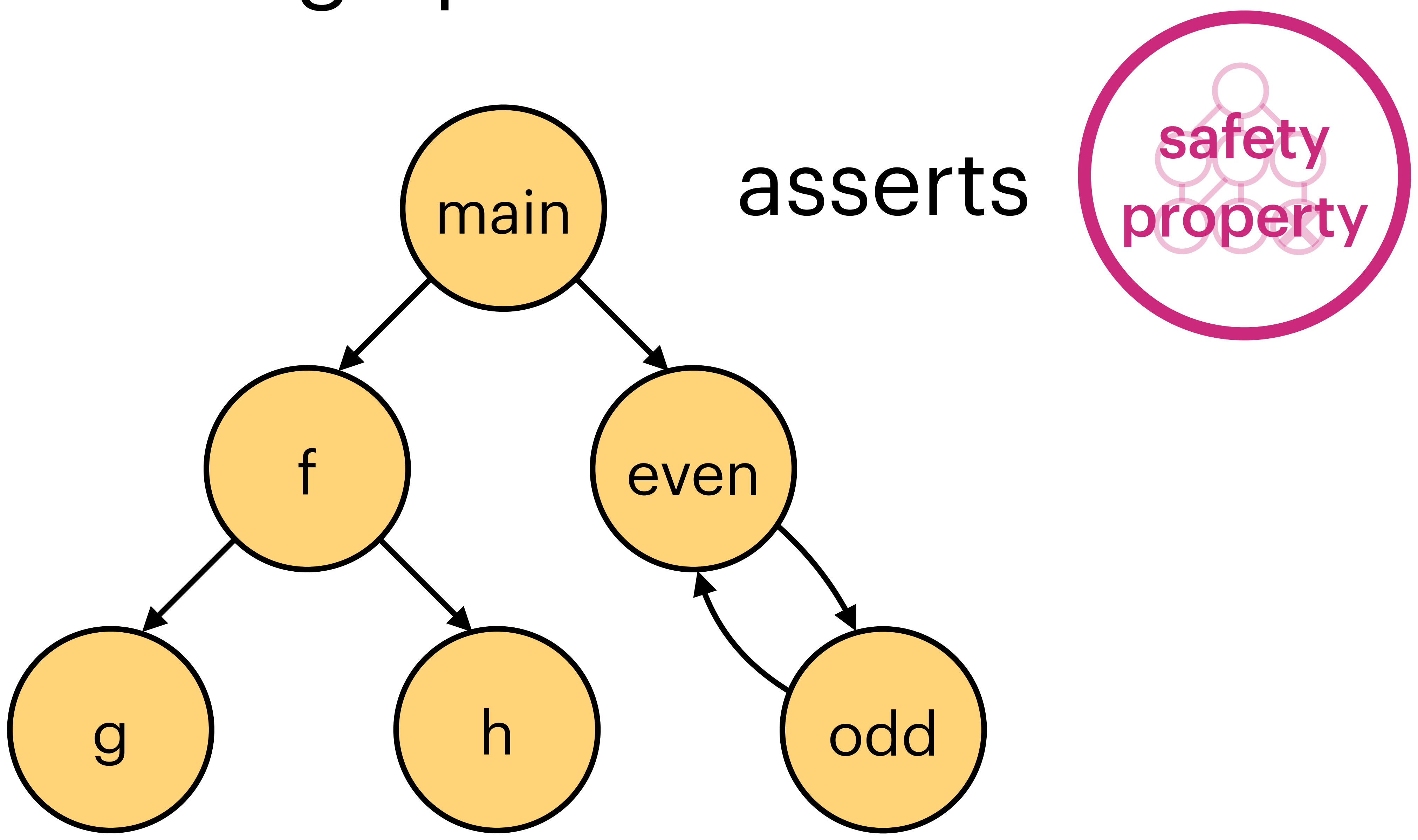


Interprocedural Programs

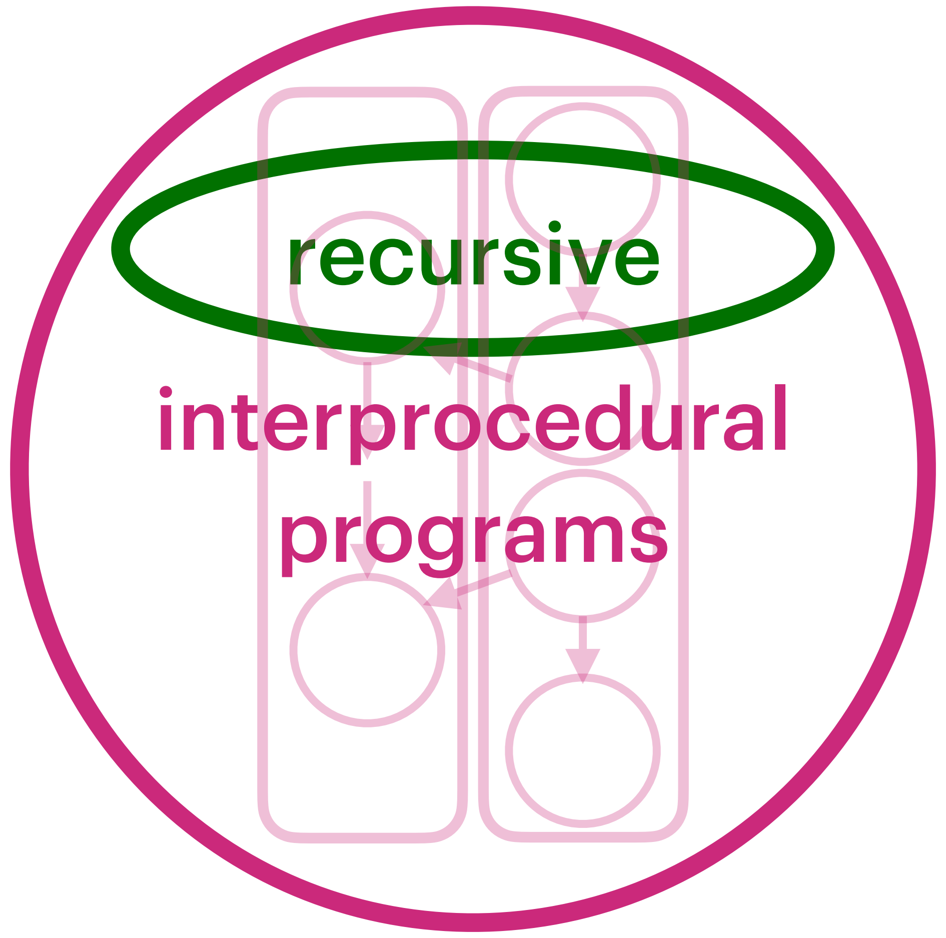


Have call graphs

Example call graph

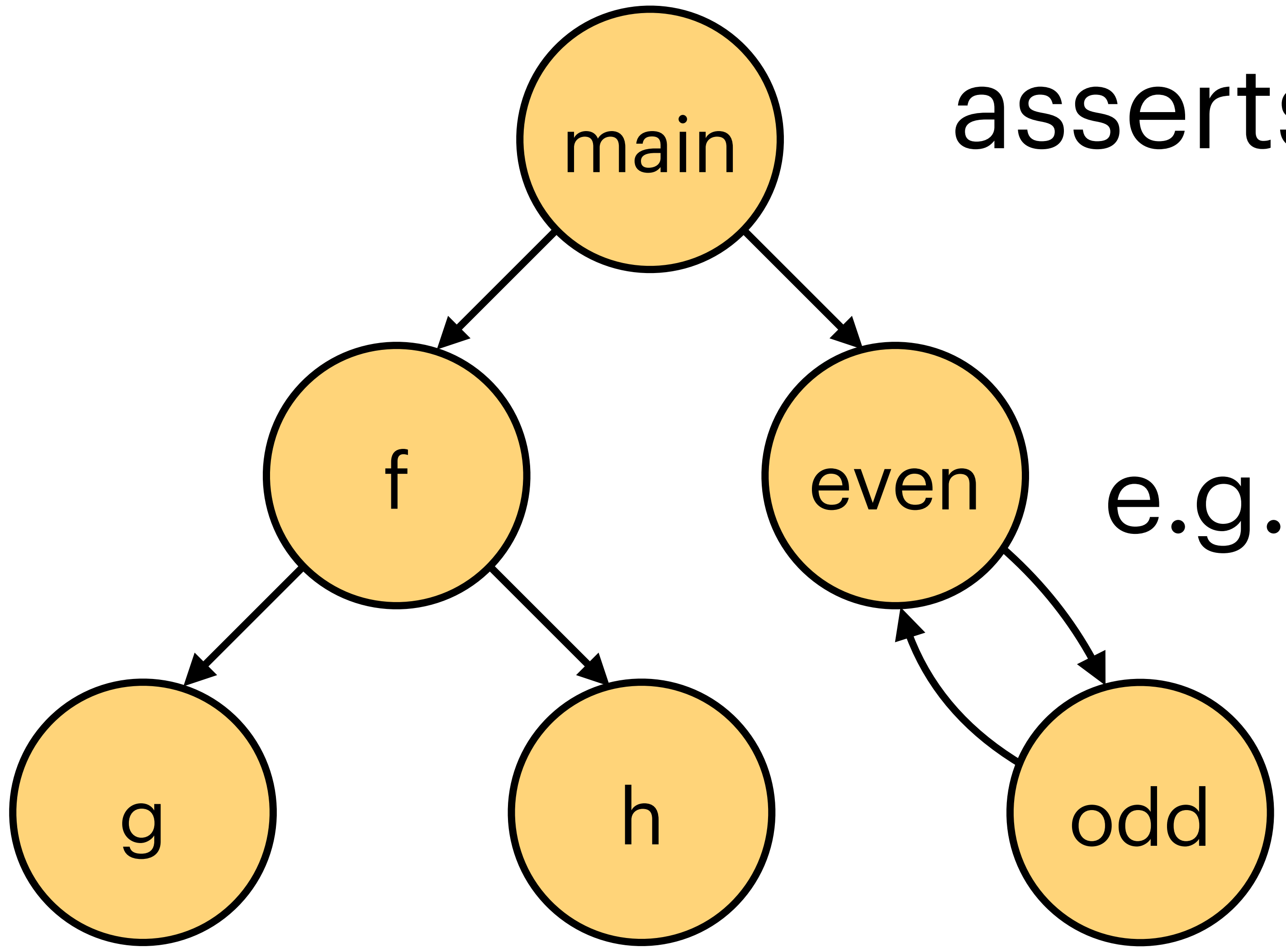


Interprocedural Programs



Have call graphs

Example call graph

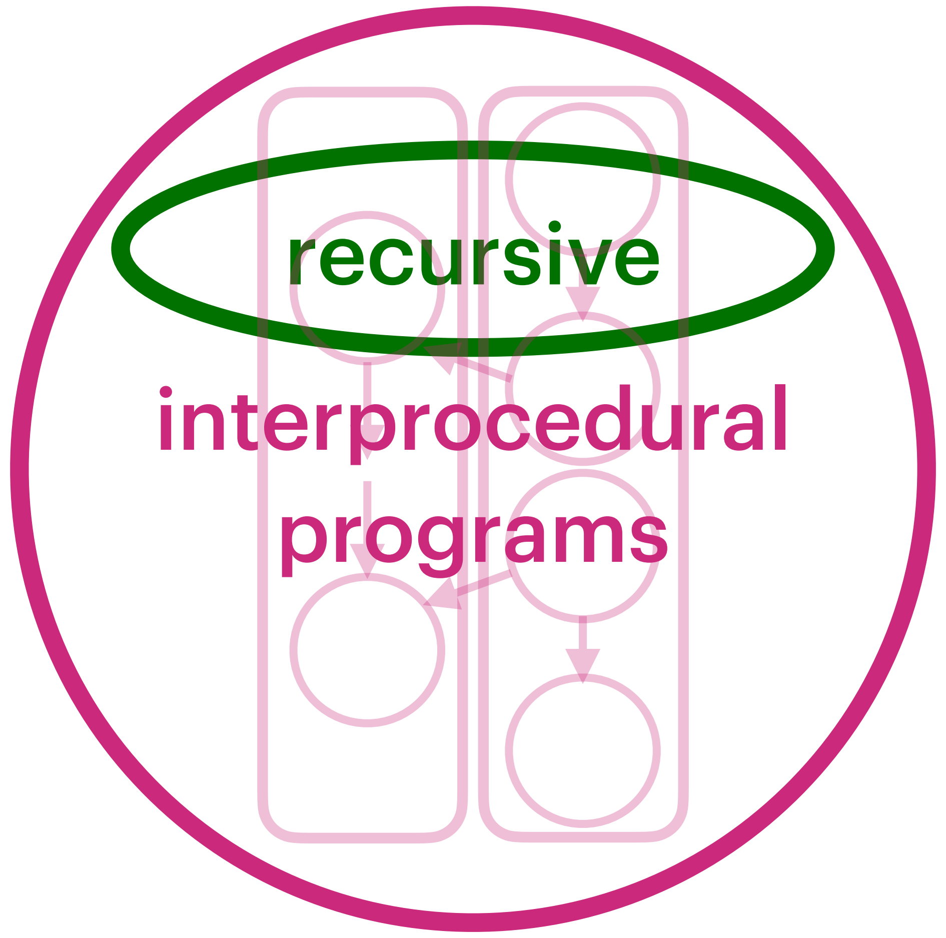


asserts

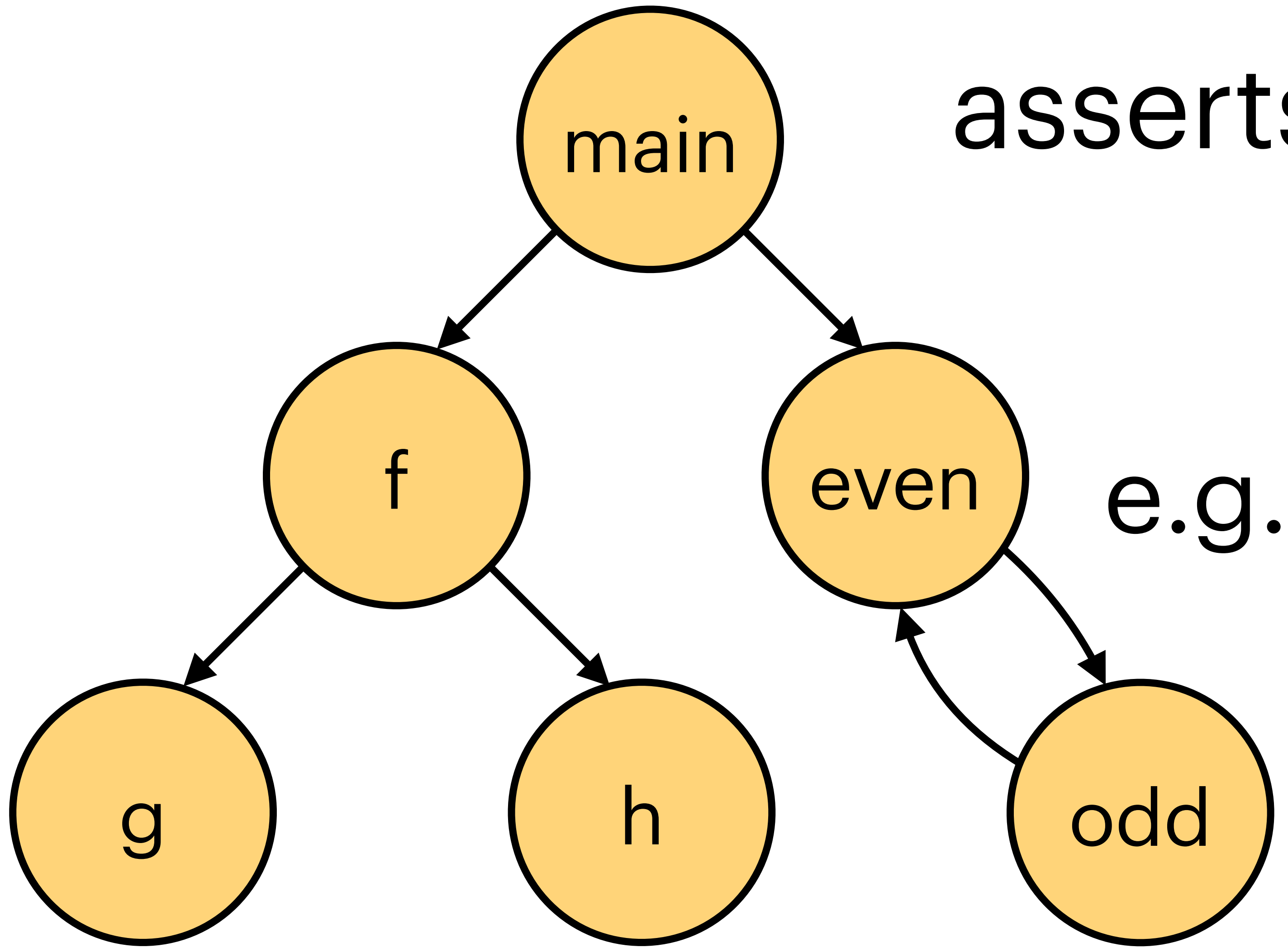
e.g., $even(f(x)) \bmod 2 \equiv 0$

Interprocedural Programs

Example call graph



Have call graphs



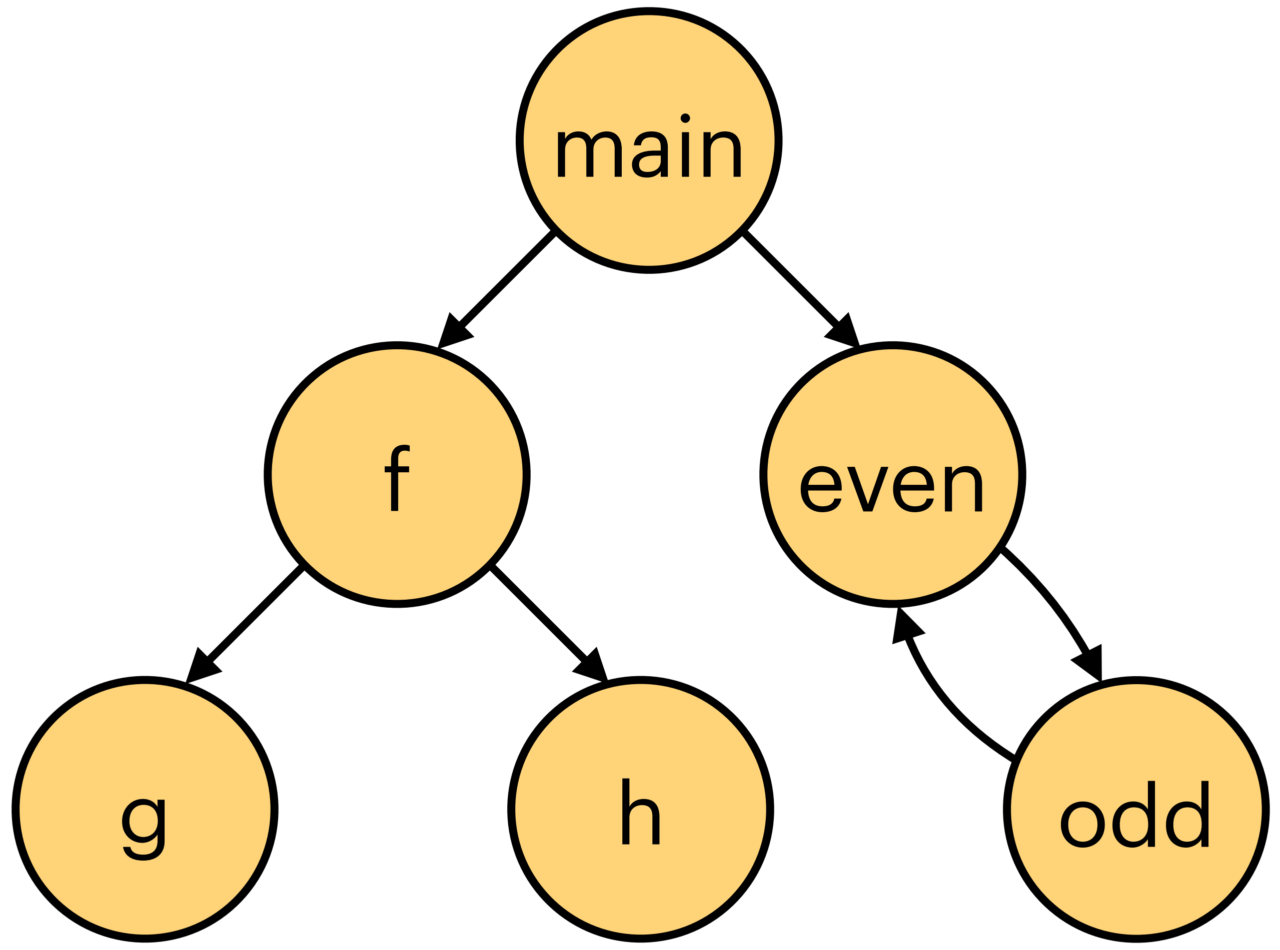
asserts

e.g., $even(f(x)) \bmod 2 \equiv 0$

Will derive and use over- and under-approximate procedure summaries

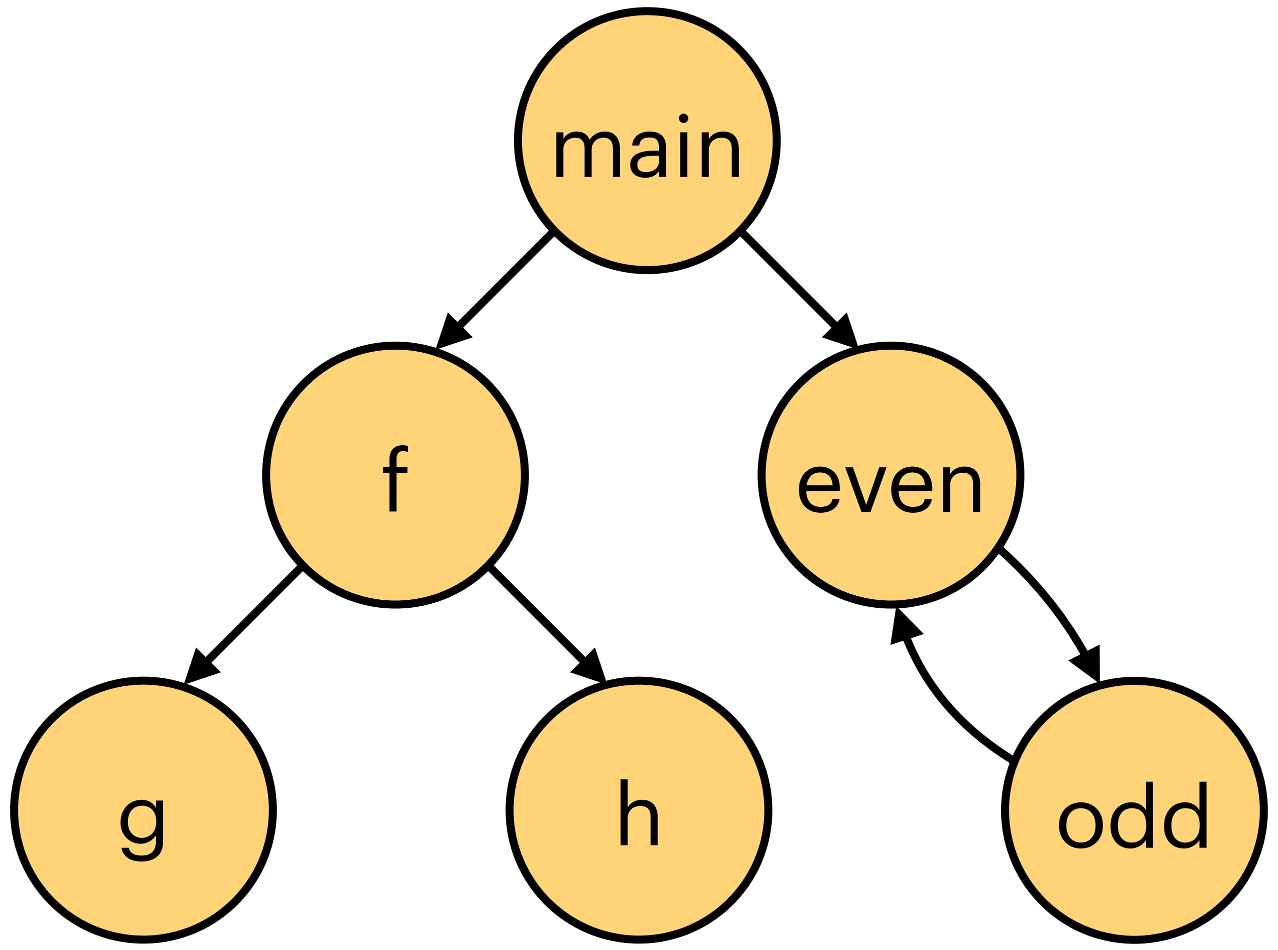
Modular Verification of Interprocedural Programs

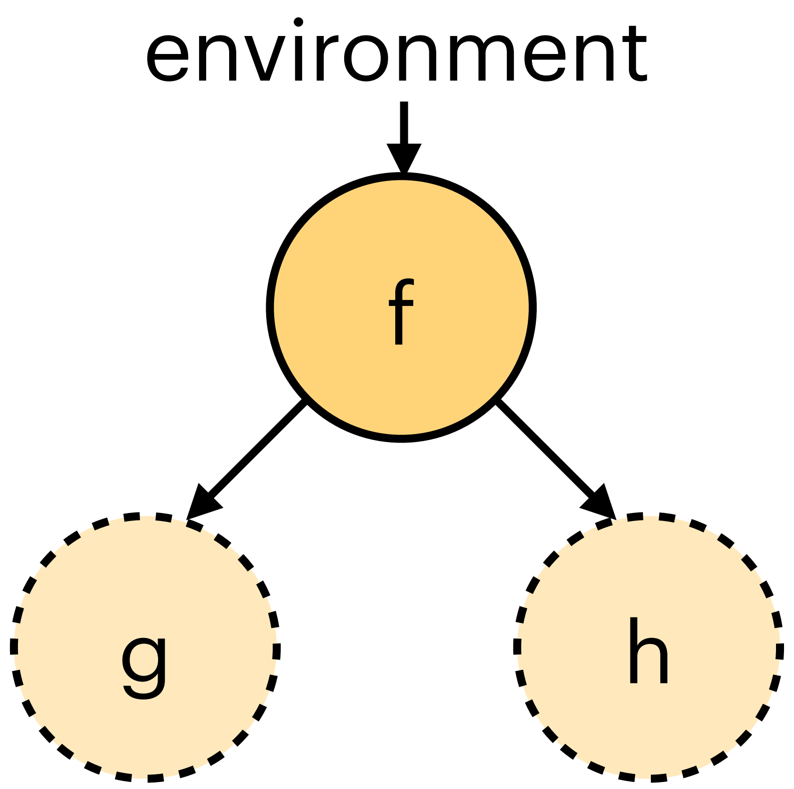
Infer and use procedure summaries (invariants)



Modular Verification of Interprocedural Programs

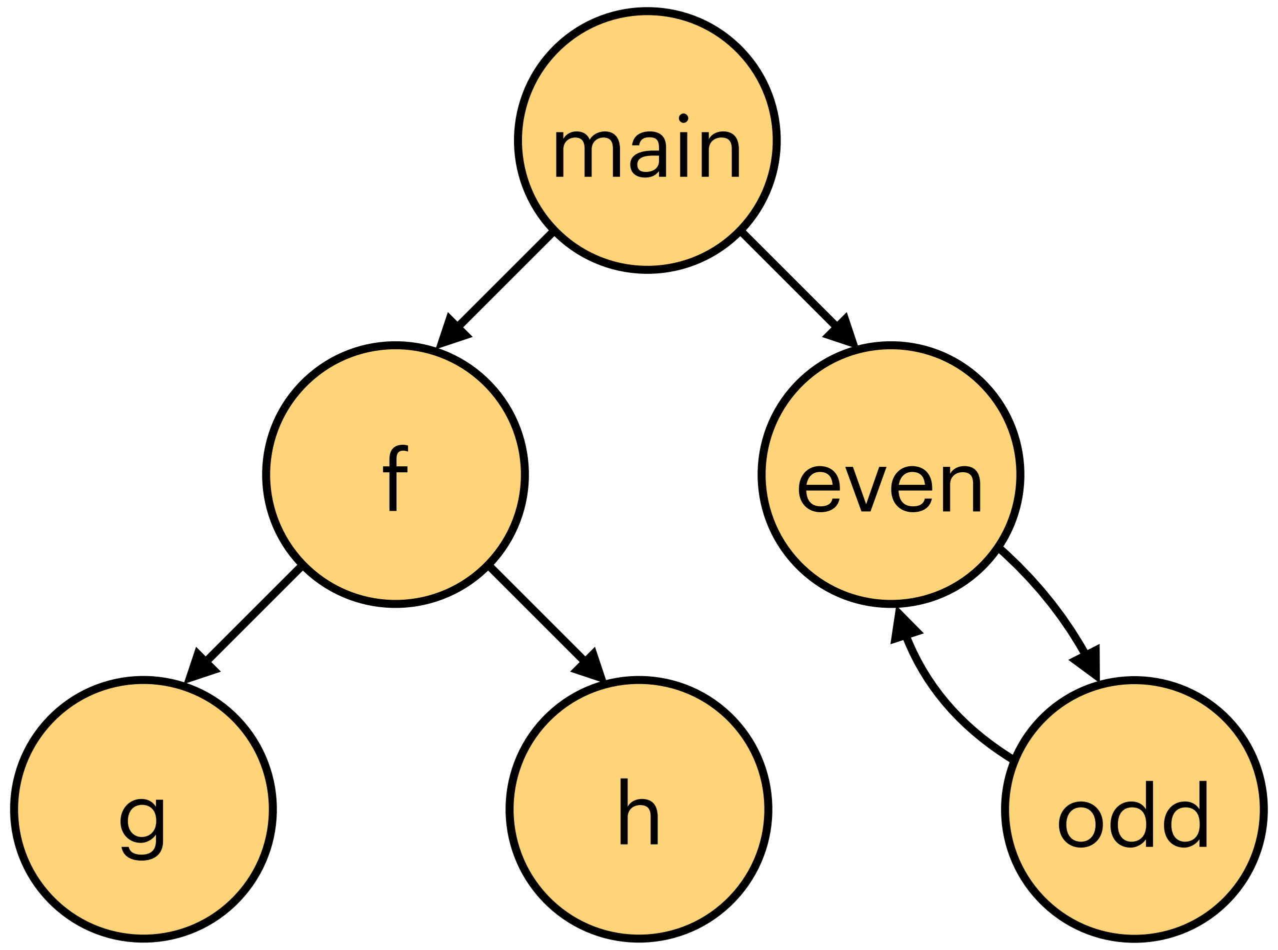
Infer and use procedure summaries (invariants)

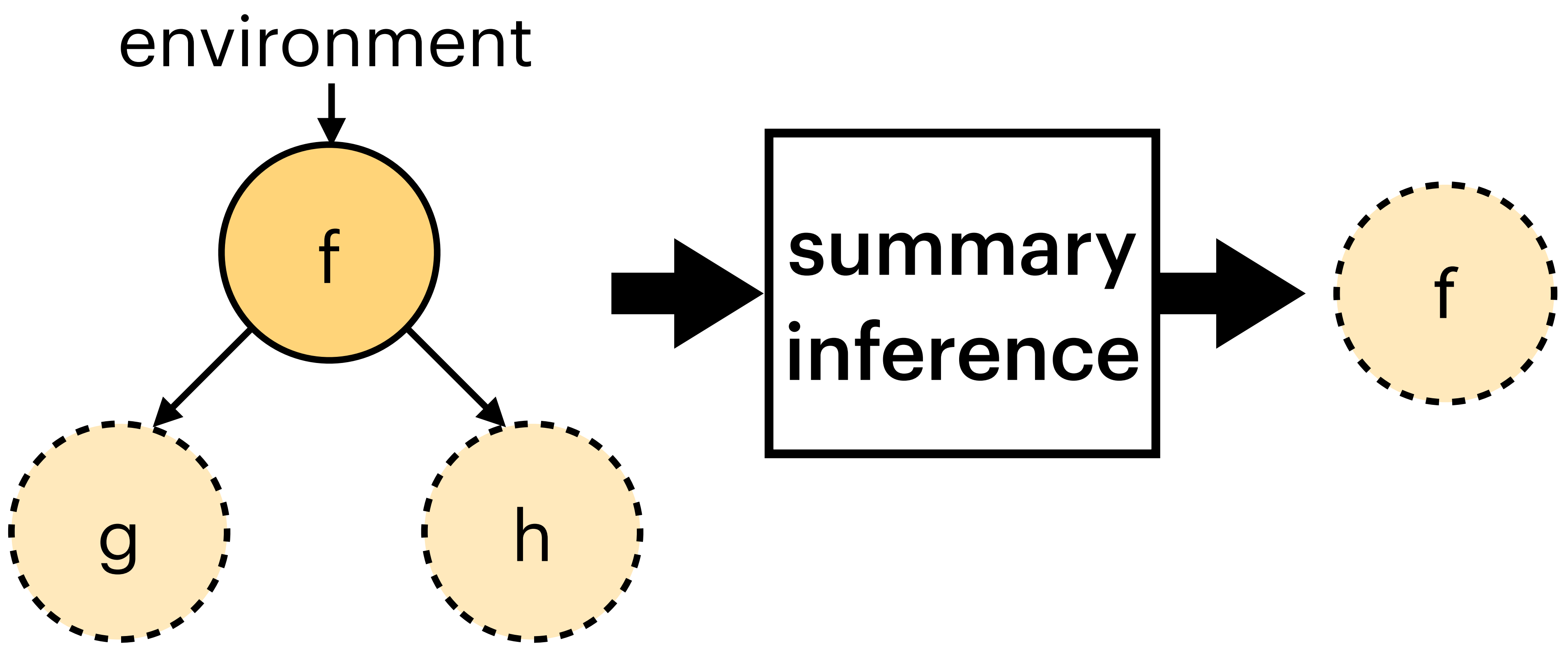




Modular Verification of Interprocedural Programs

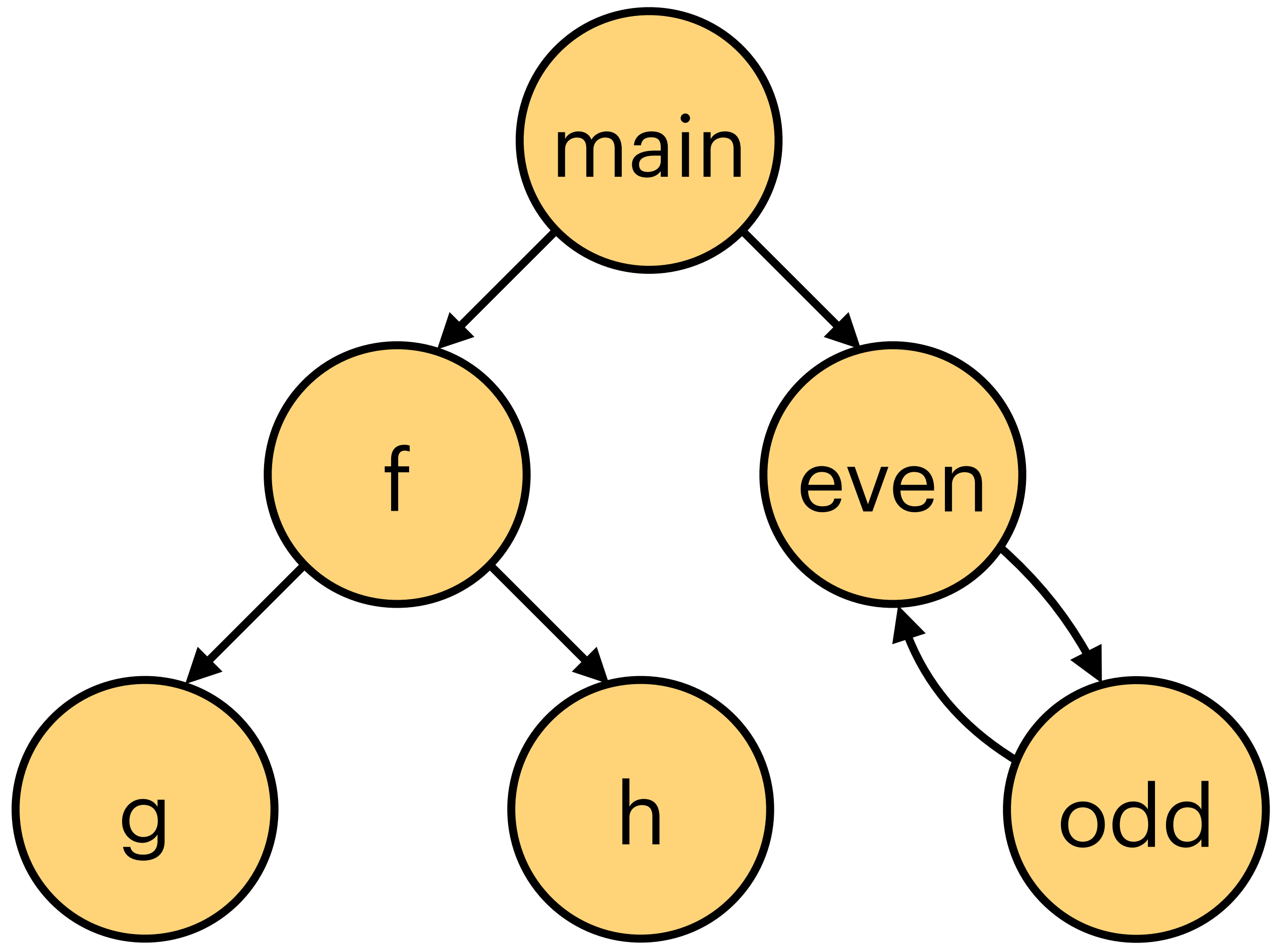
Infer and use procedure summaries (invariants)

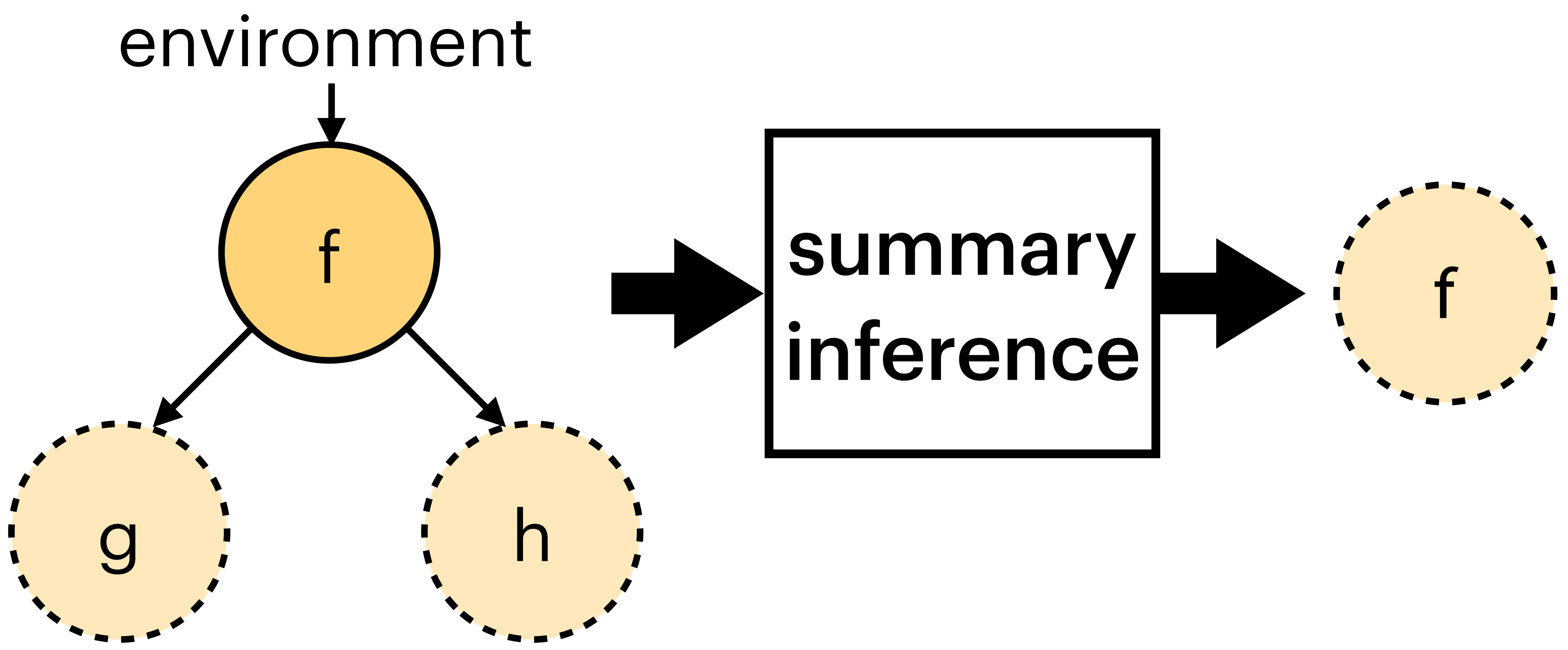




Modular Verification of Interprocedural Programs

Infer and use procedure summaries (invariants)

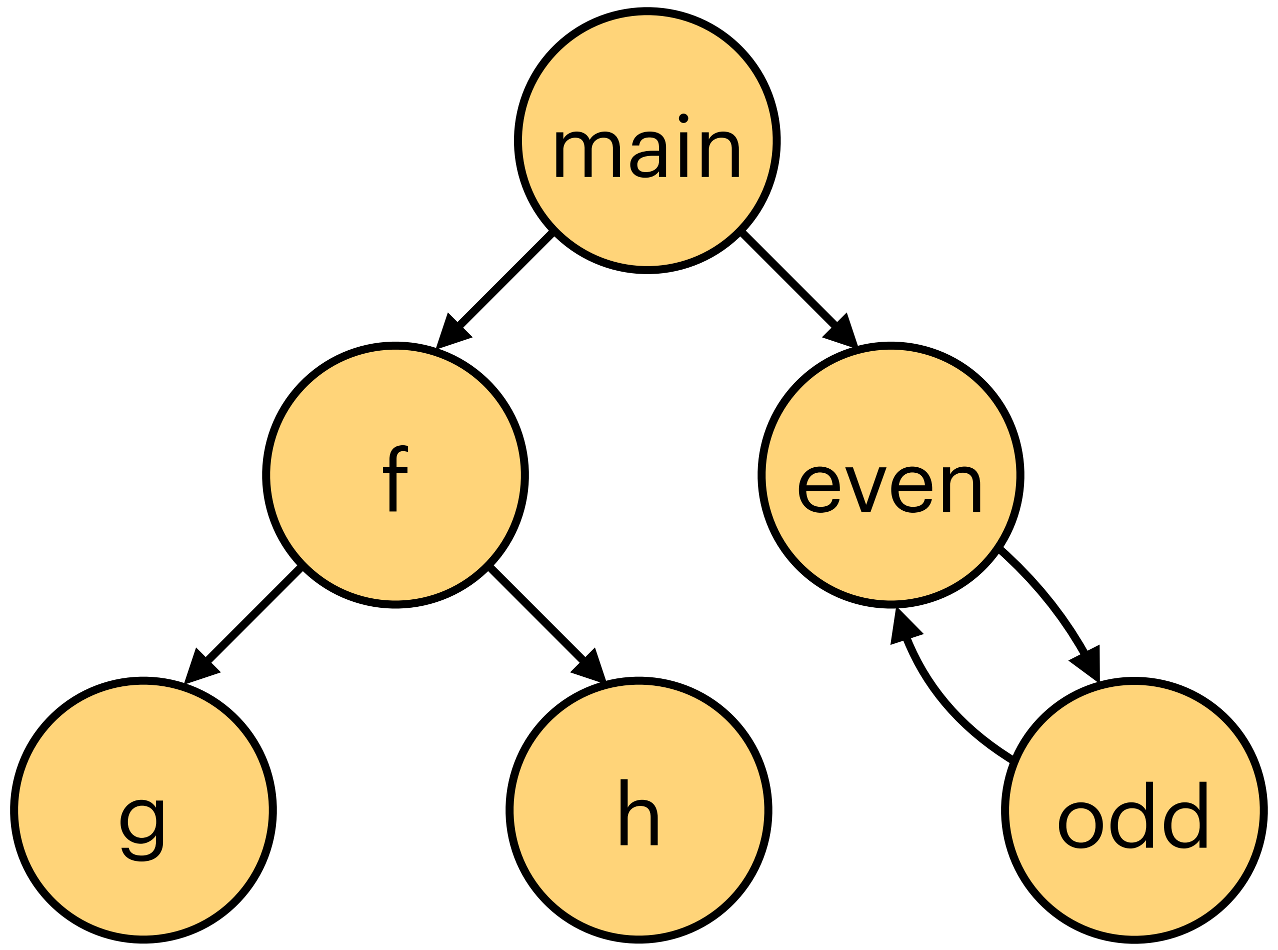


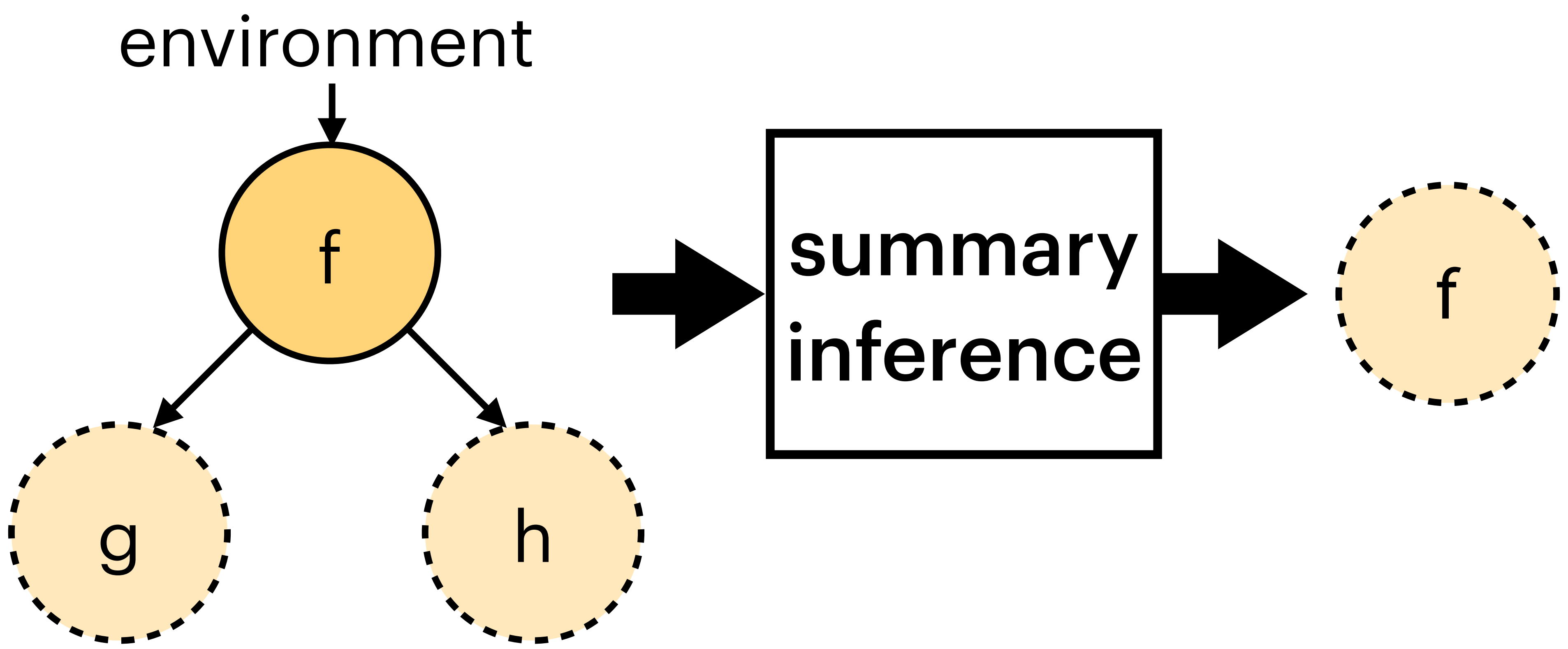


to handle mutual recursion

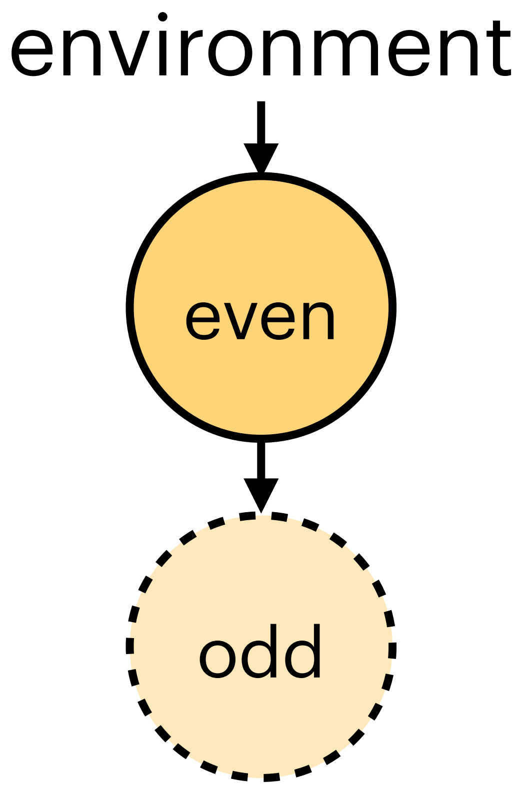
Modular Verification of Interprocedural Programs

Infer and use procedure summaries (invariants)



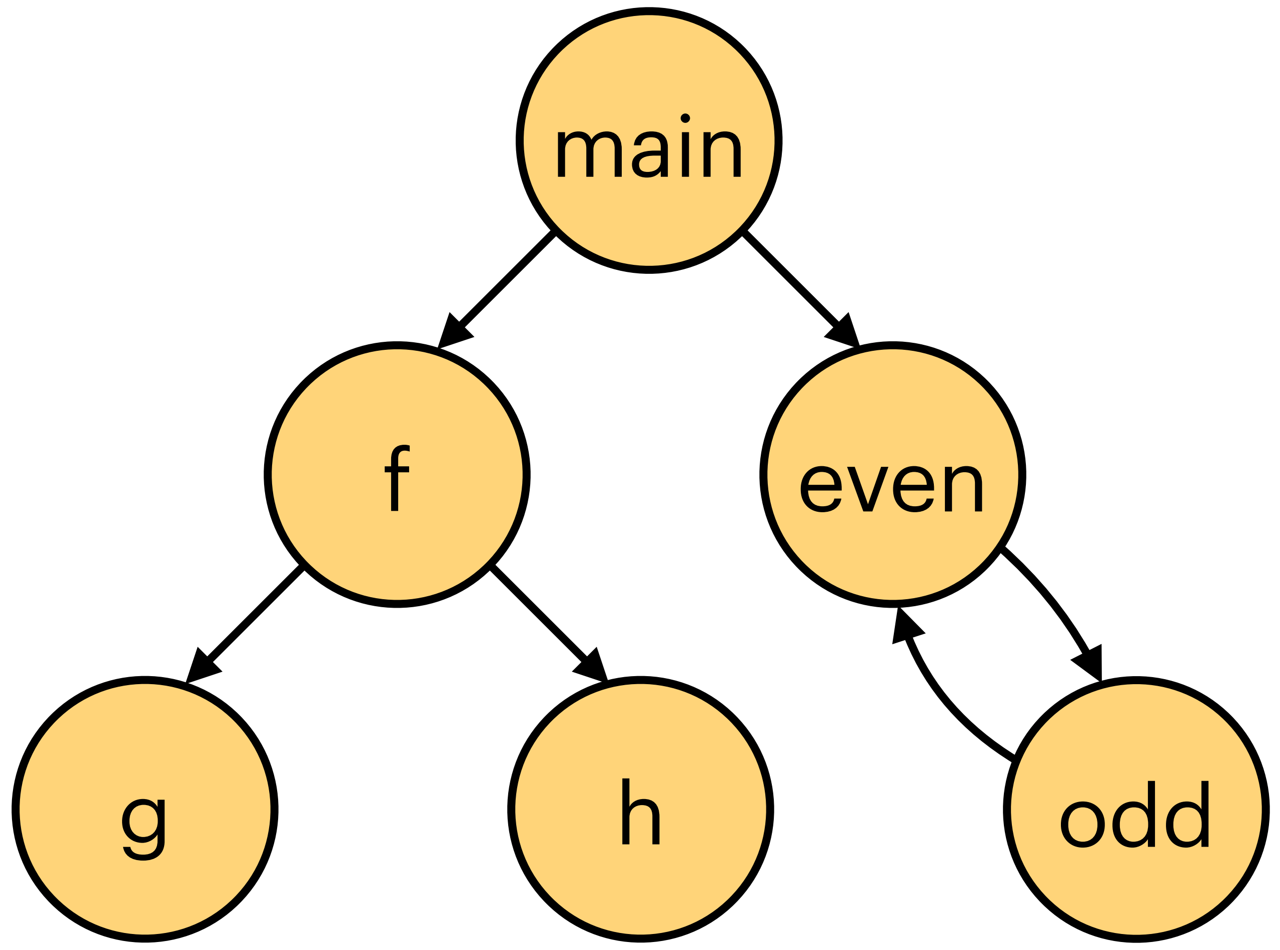


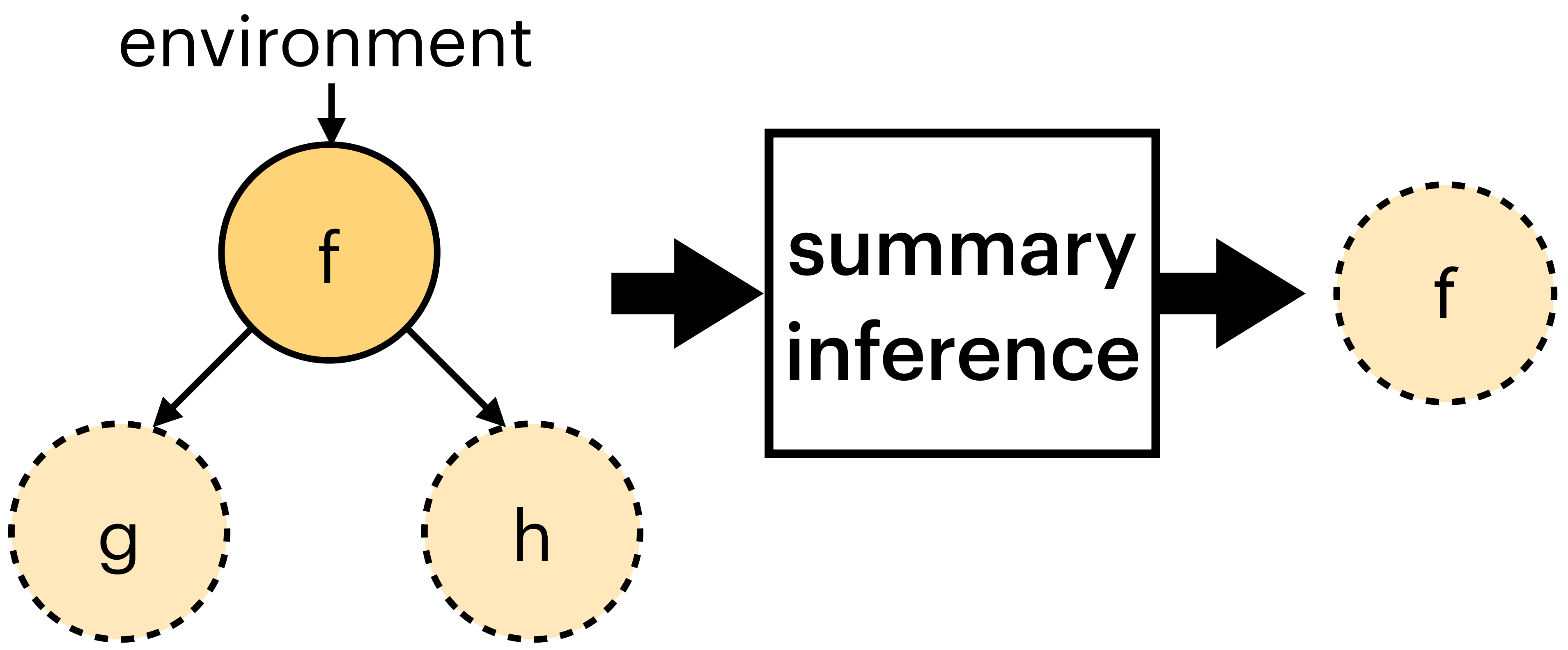
to handle mutual recursion



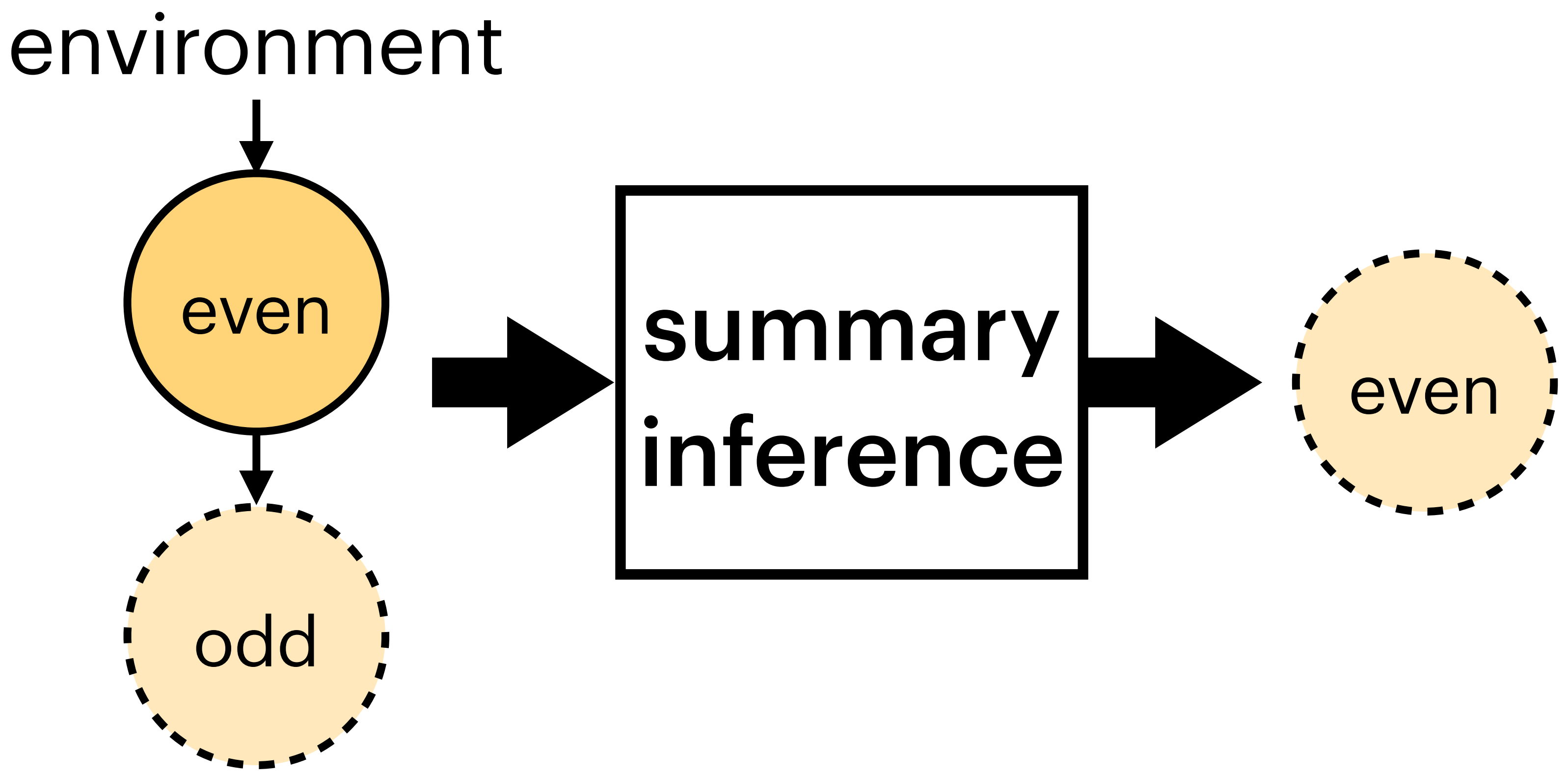
Modular Verification of Interprocedural Programs

Infer and use procedure summaries (invariants)



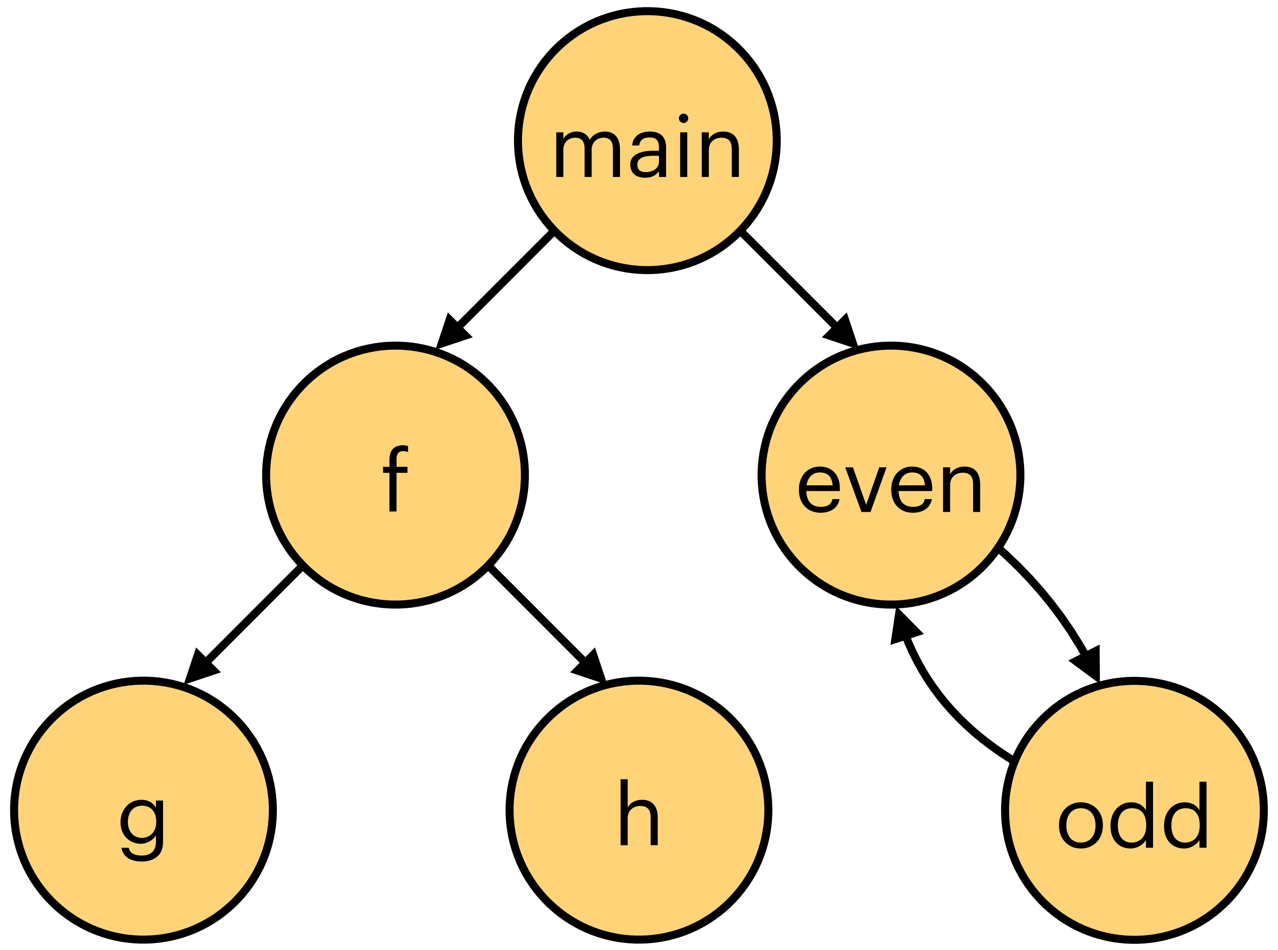


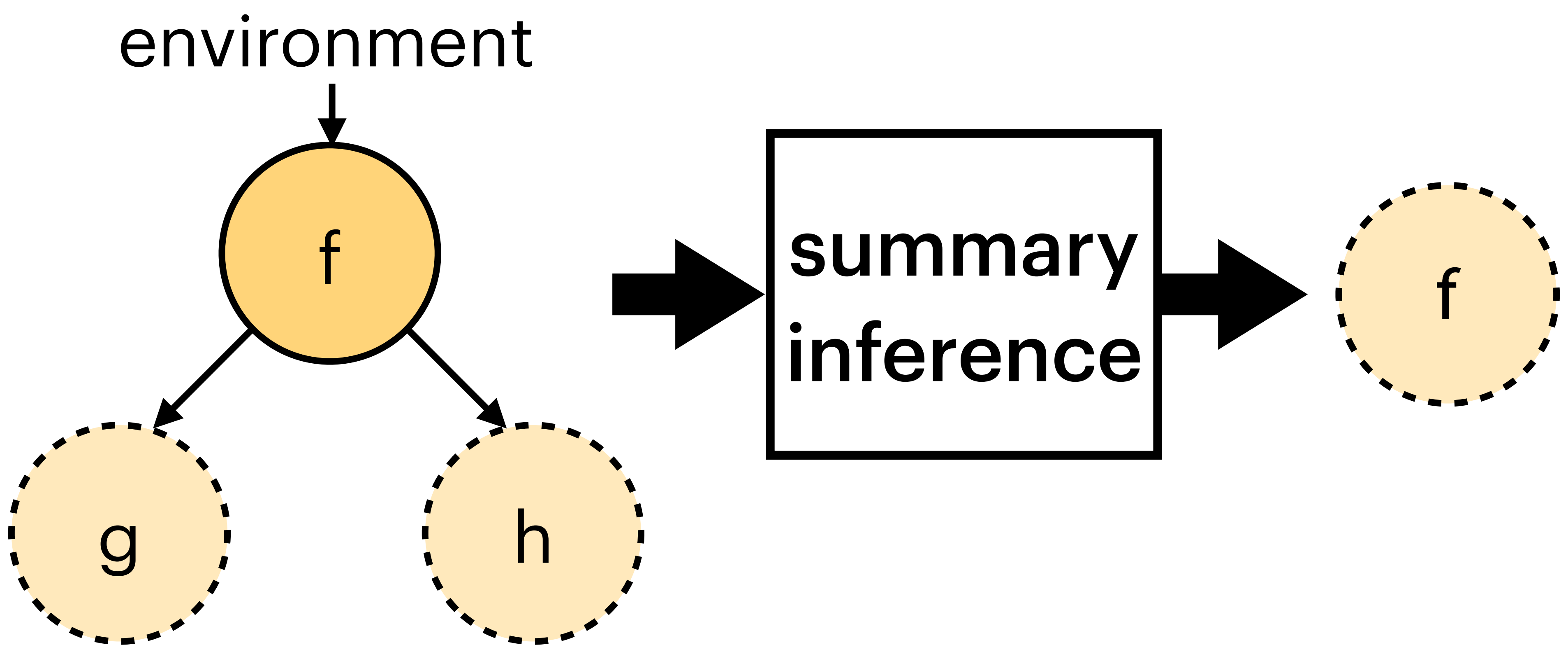
to handle mutual recursion



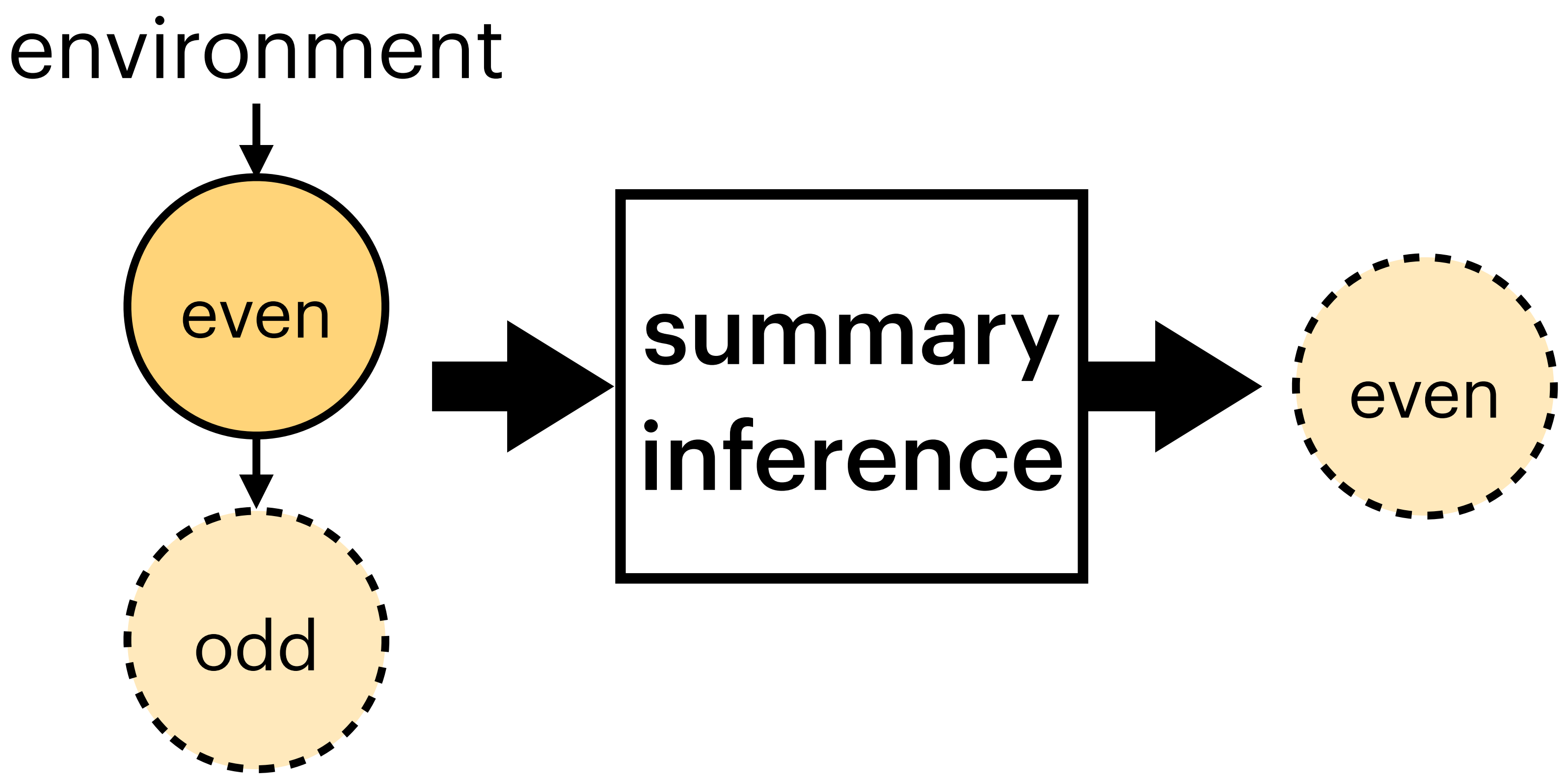
Modular Verification of Interprocedural Programs

Infer and use procedure summaries (invariants)



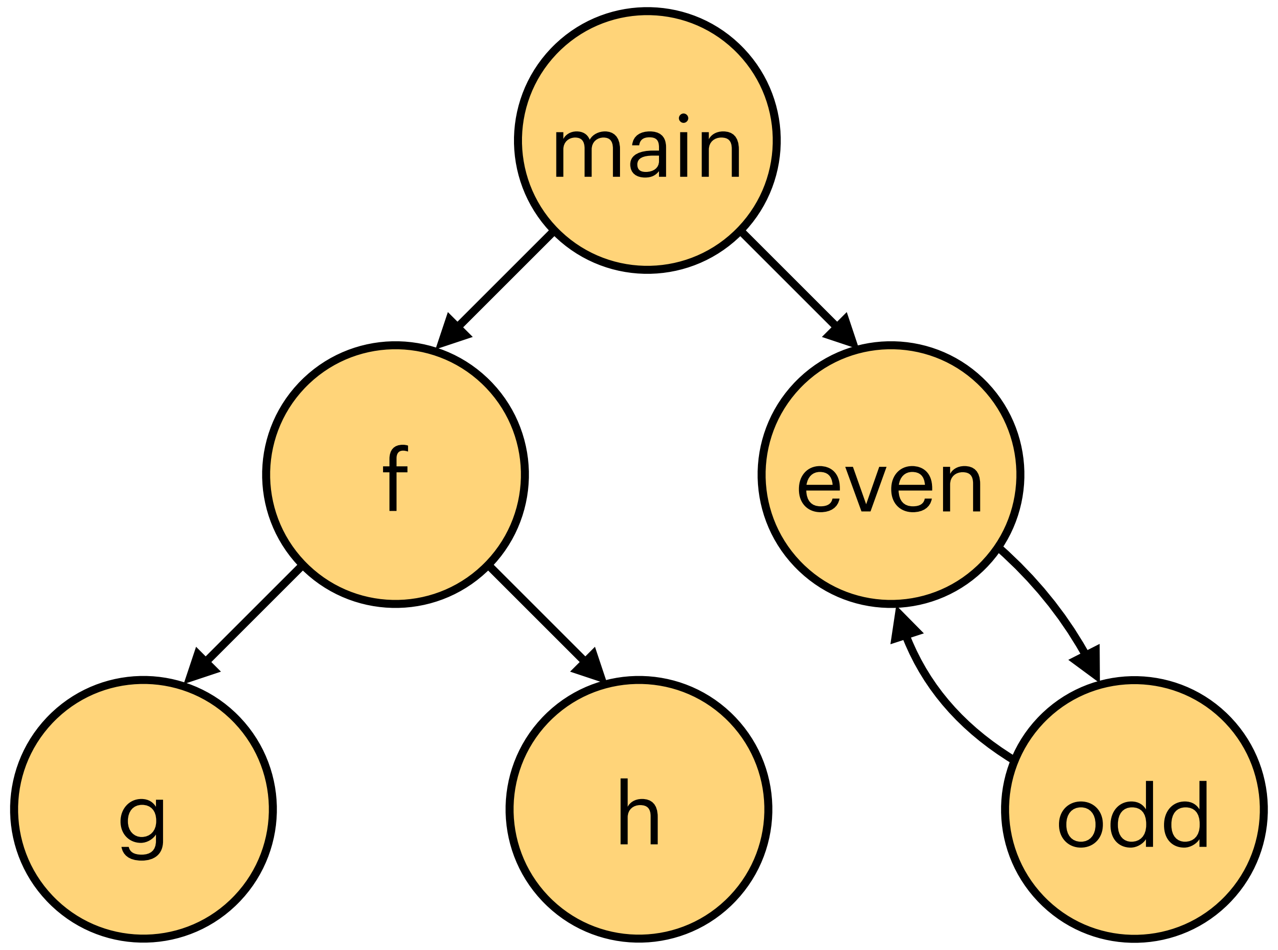


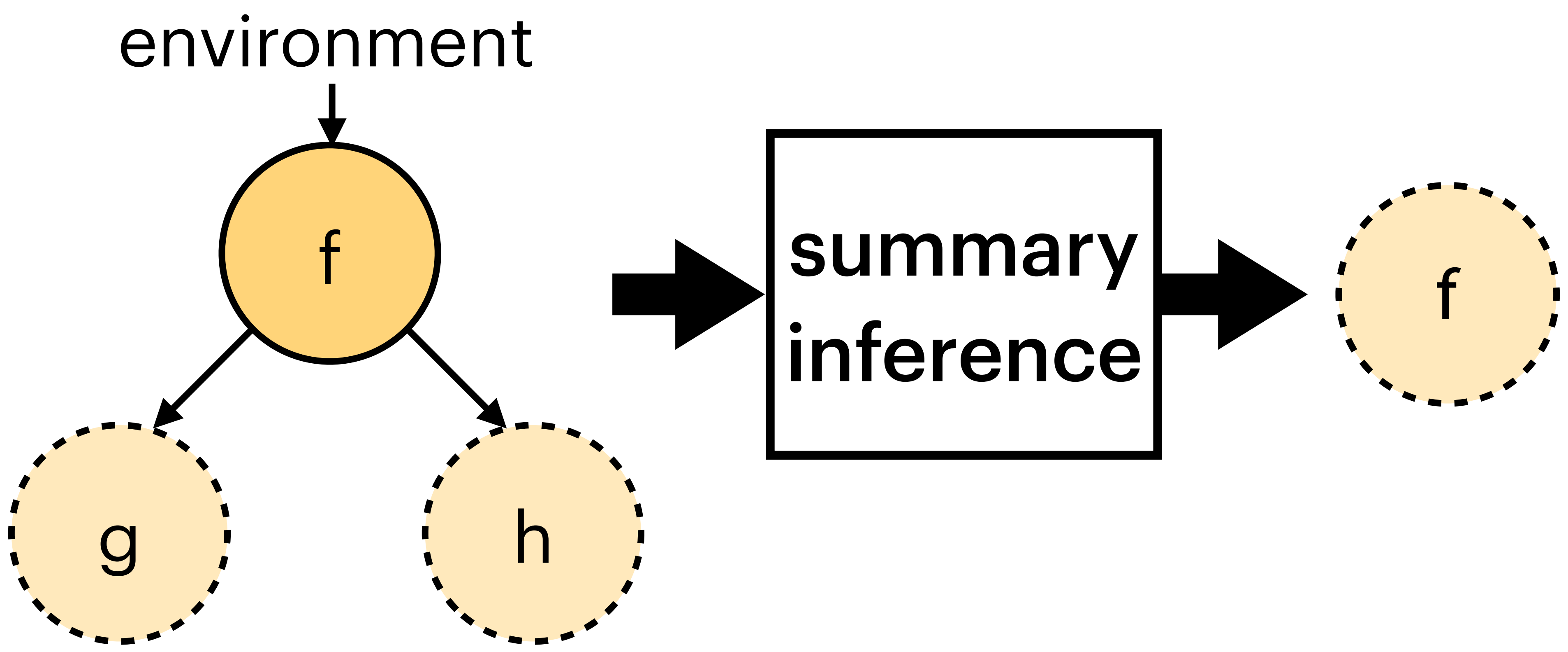
to handle mutual recursion and scale verification



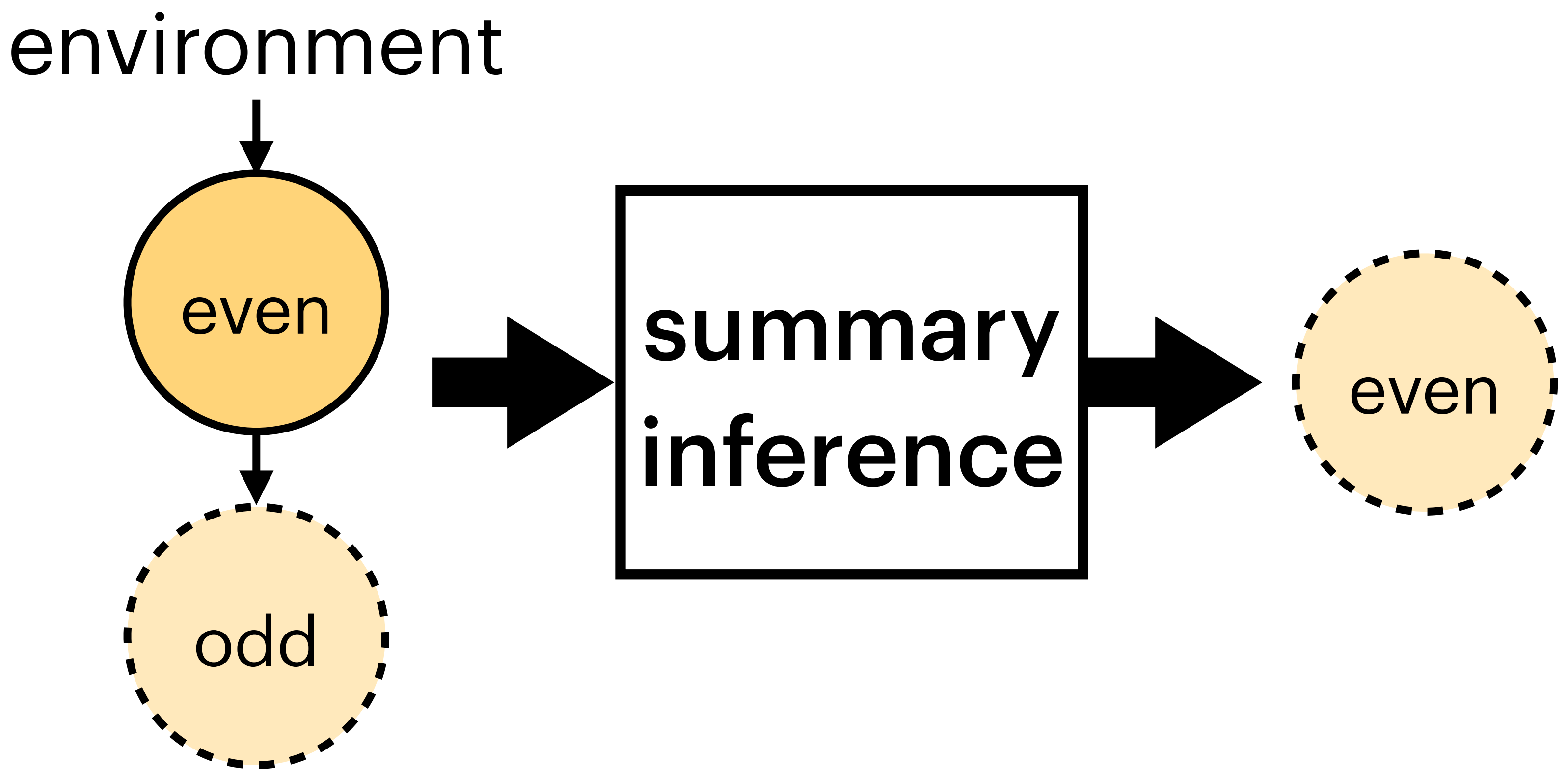
Modular Verification of Interprocedural Programs

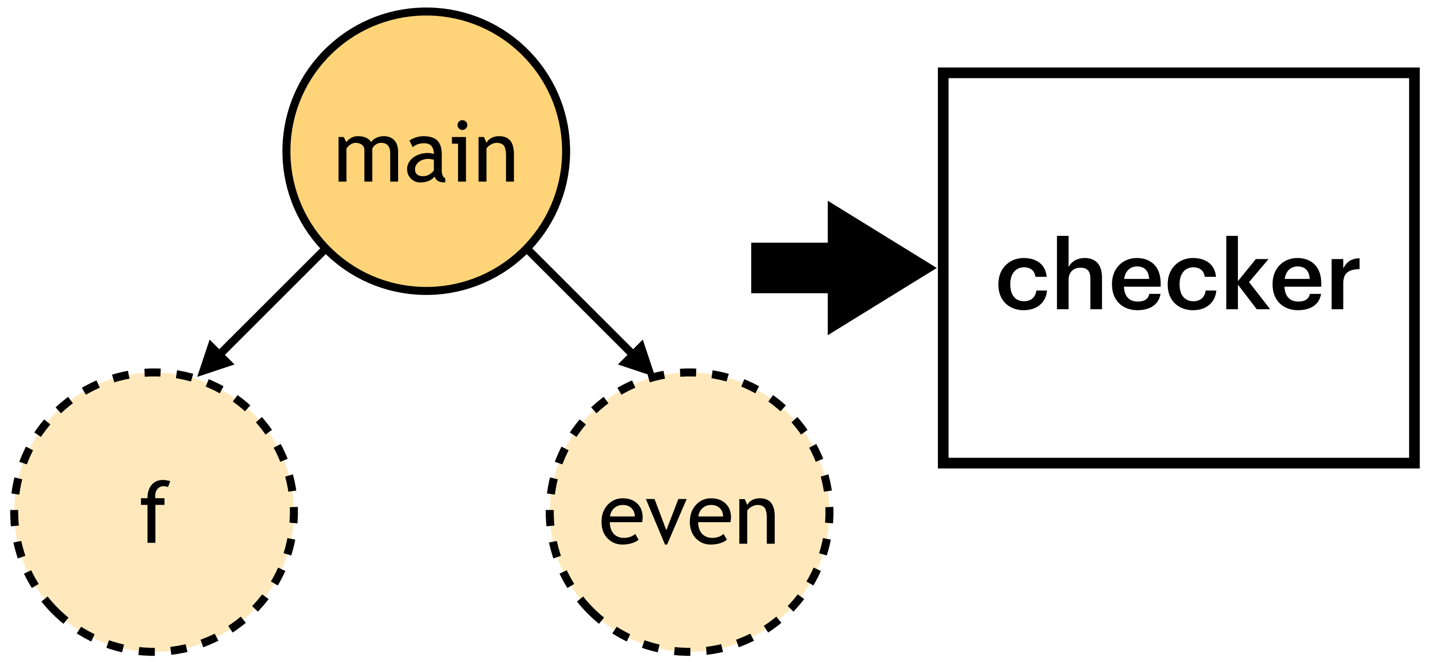
Infer and use procedure summaries (invariants)



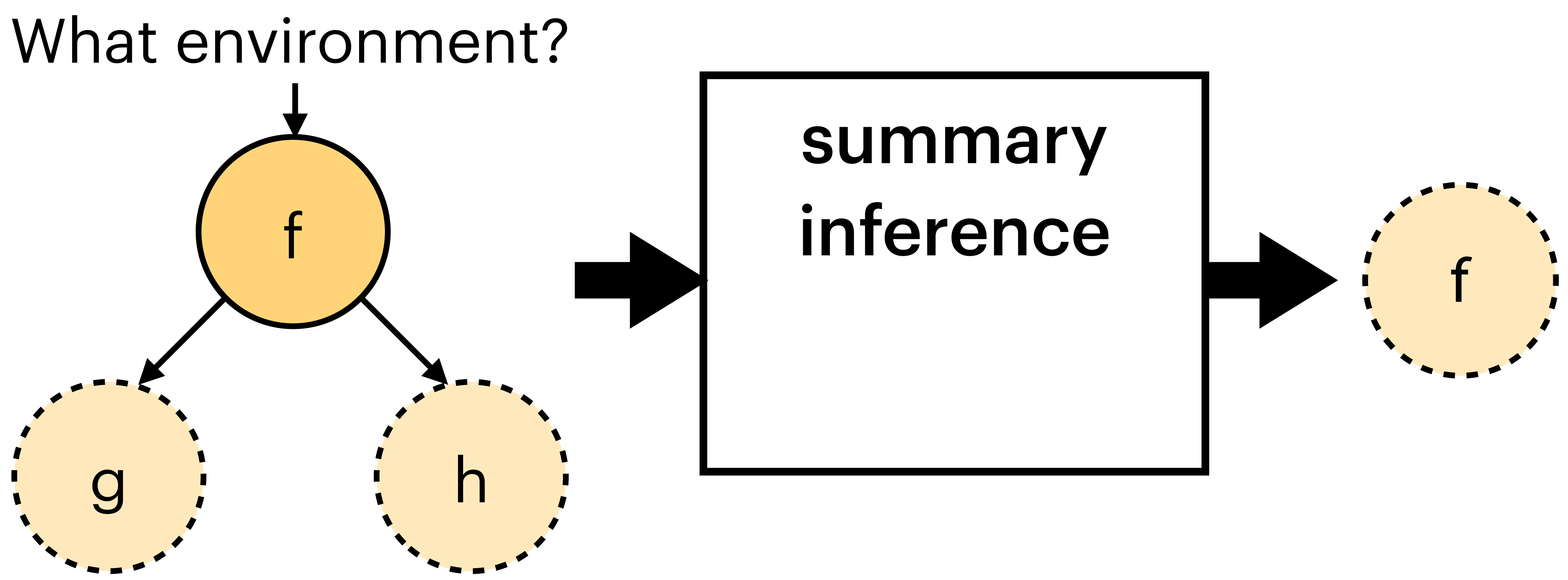


to handle mutual recursion and scale verification

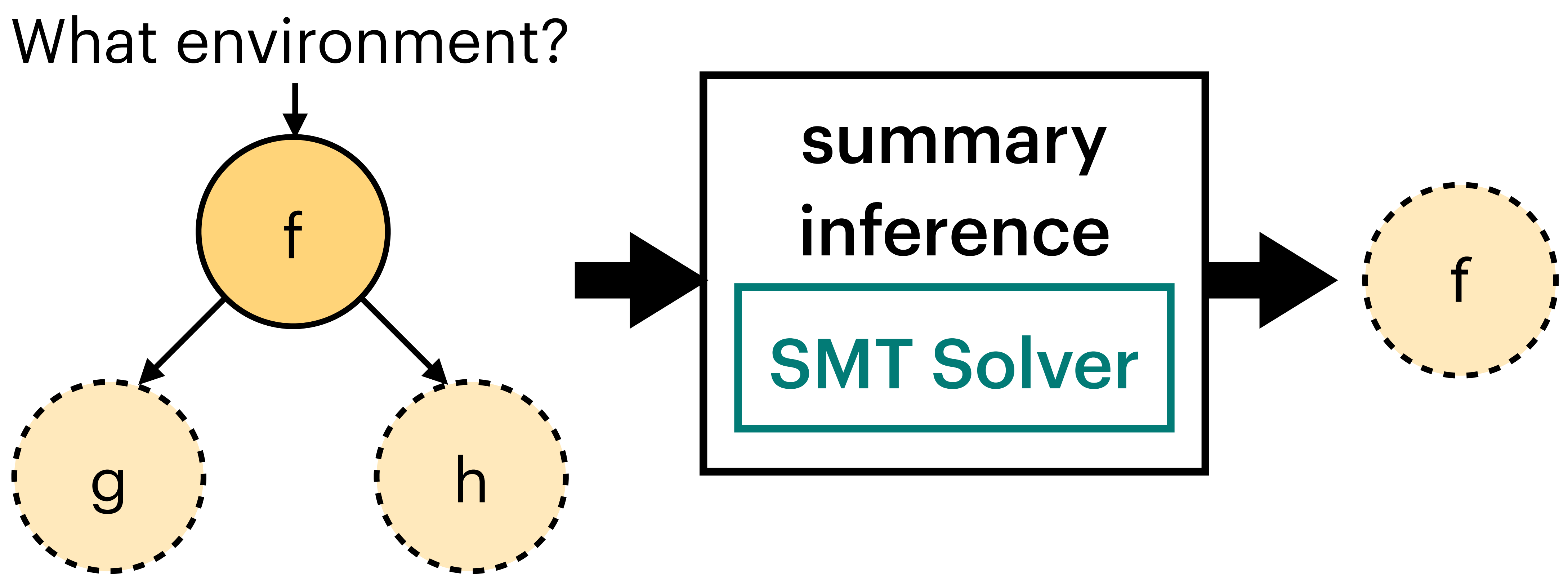




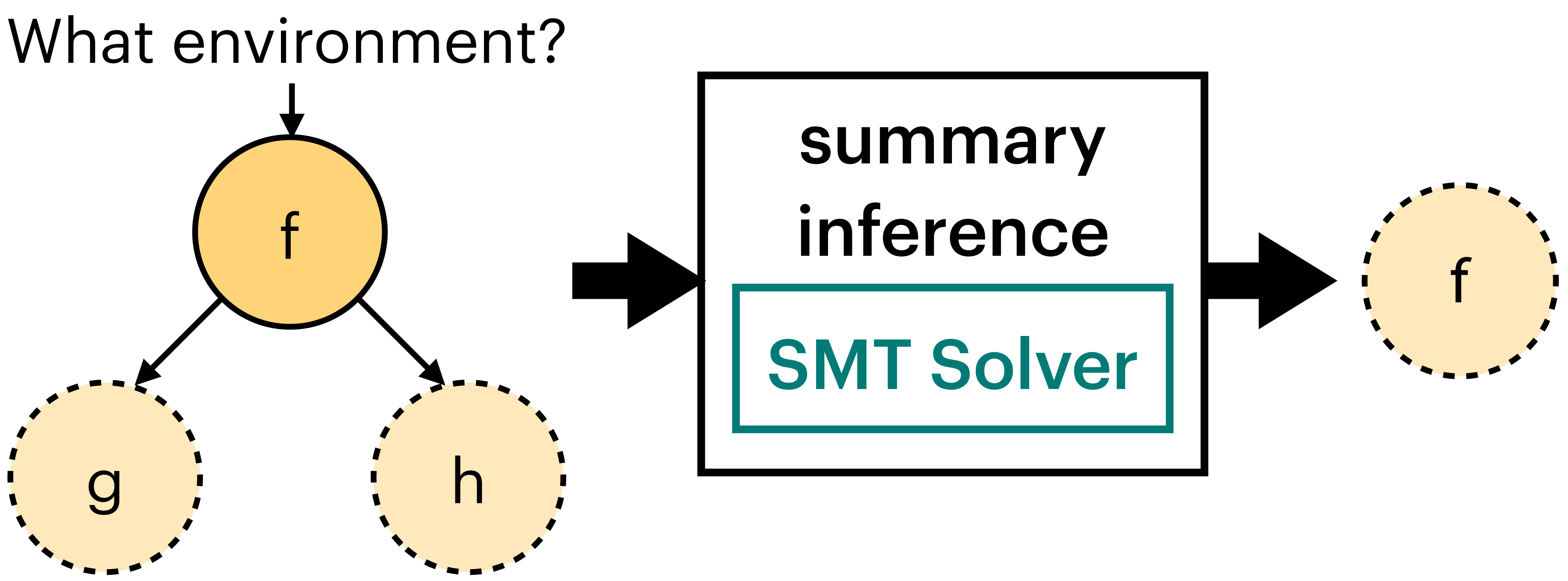
Scalable Inference vs. Relevance of Invariants



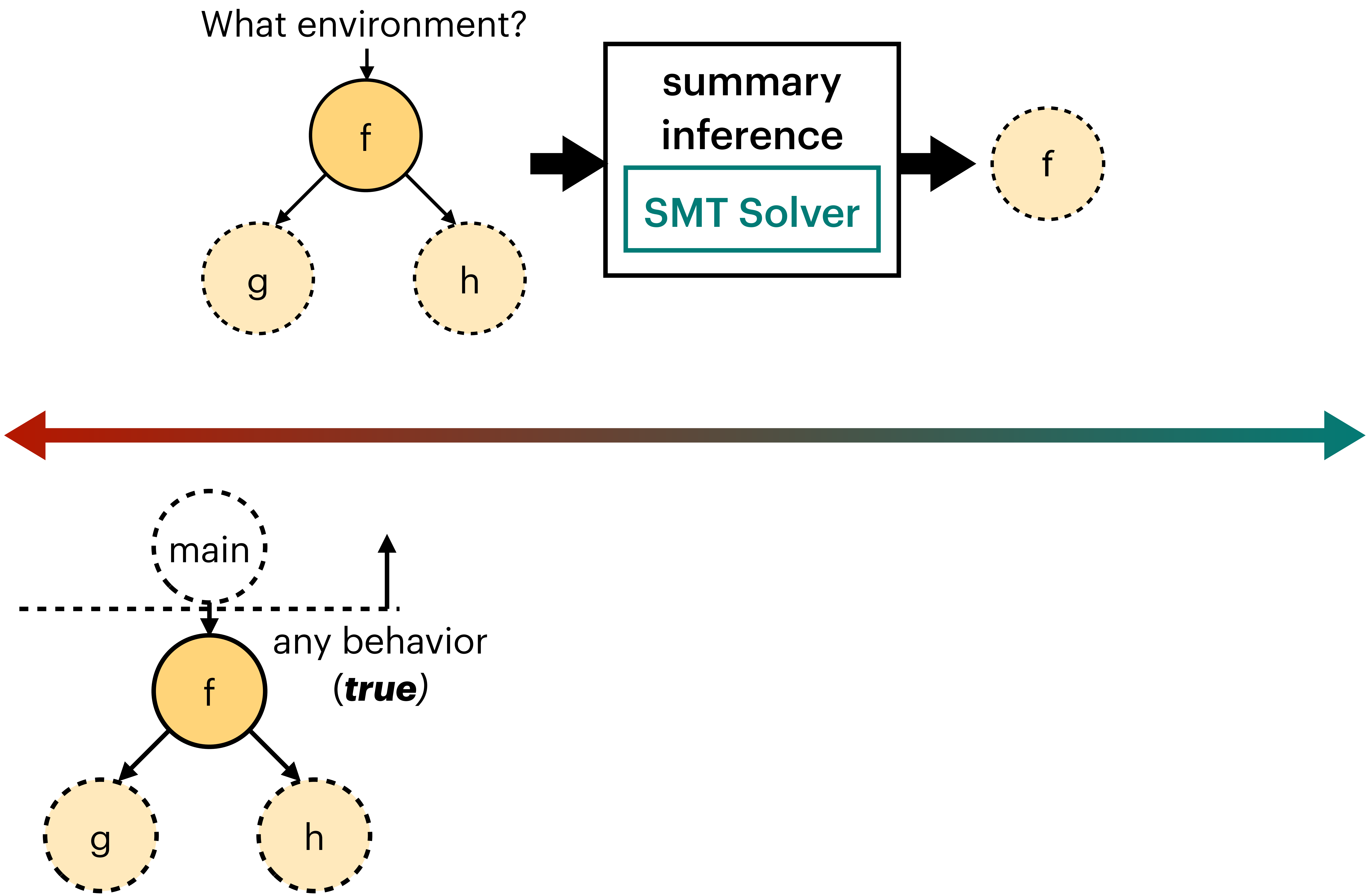
Scalable Inference vs. Relevance of Invariants



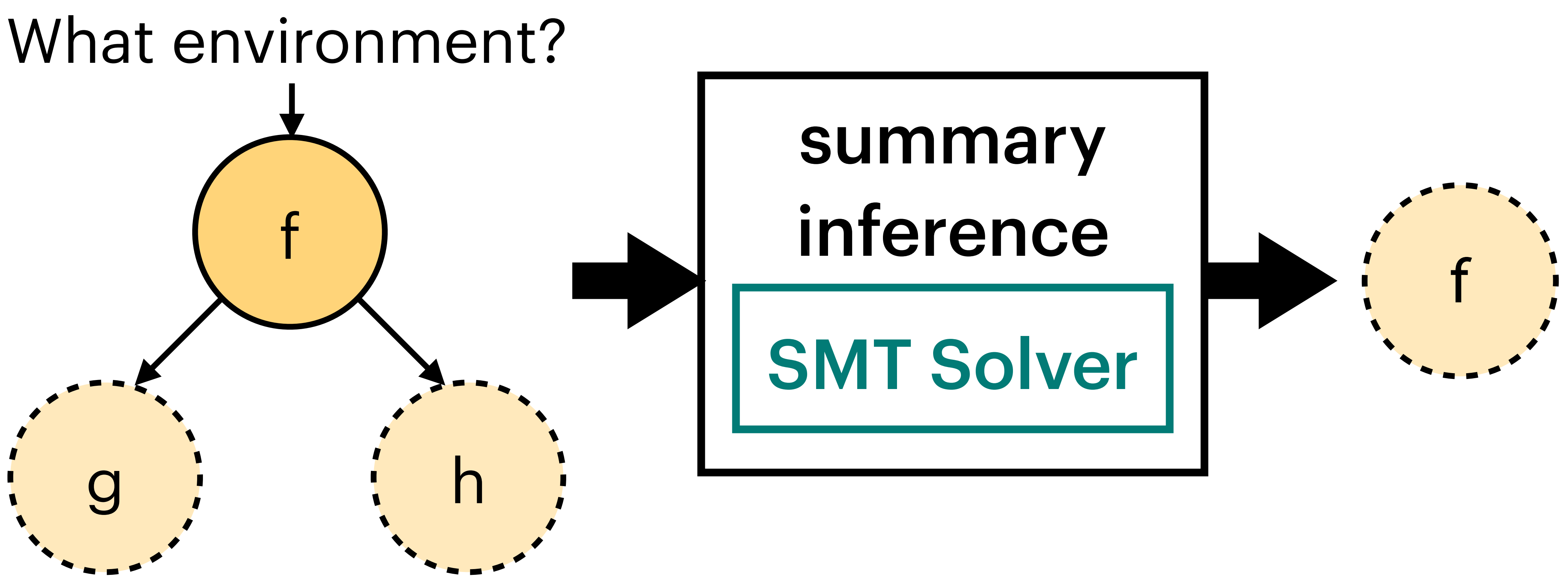
Scalable Inference vs. Relevance of Invariants



Scalable Inference vs. Relevance of Invariants

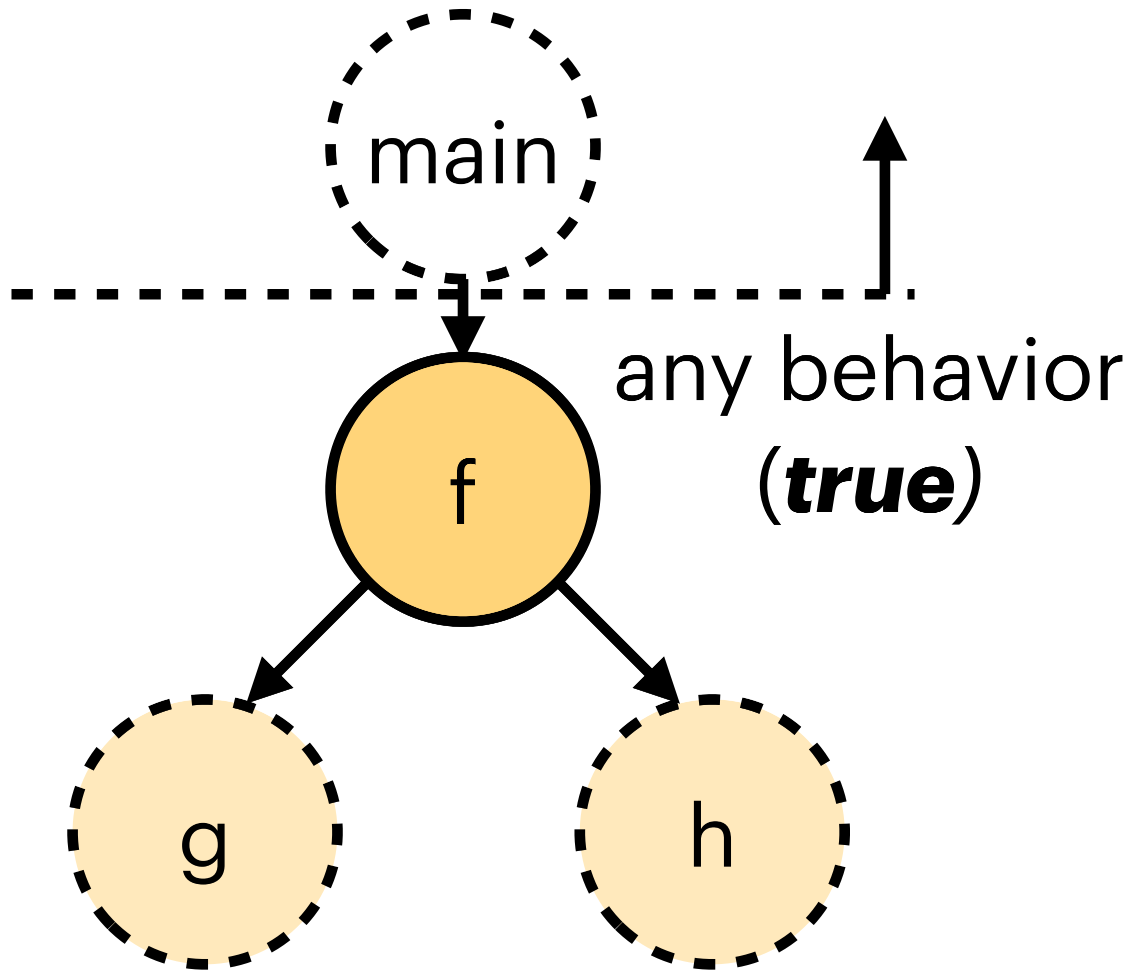


Scalable Inference vs. Relevance of Invariants

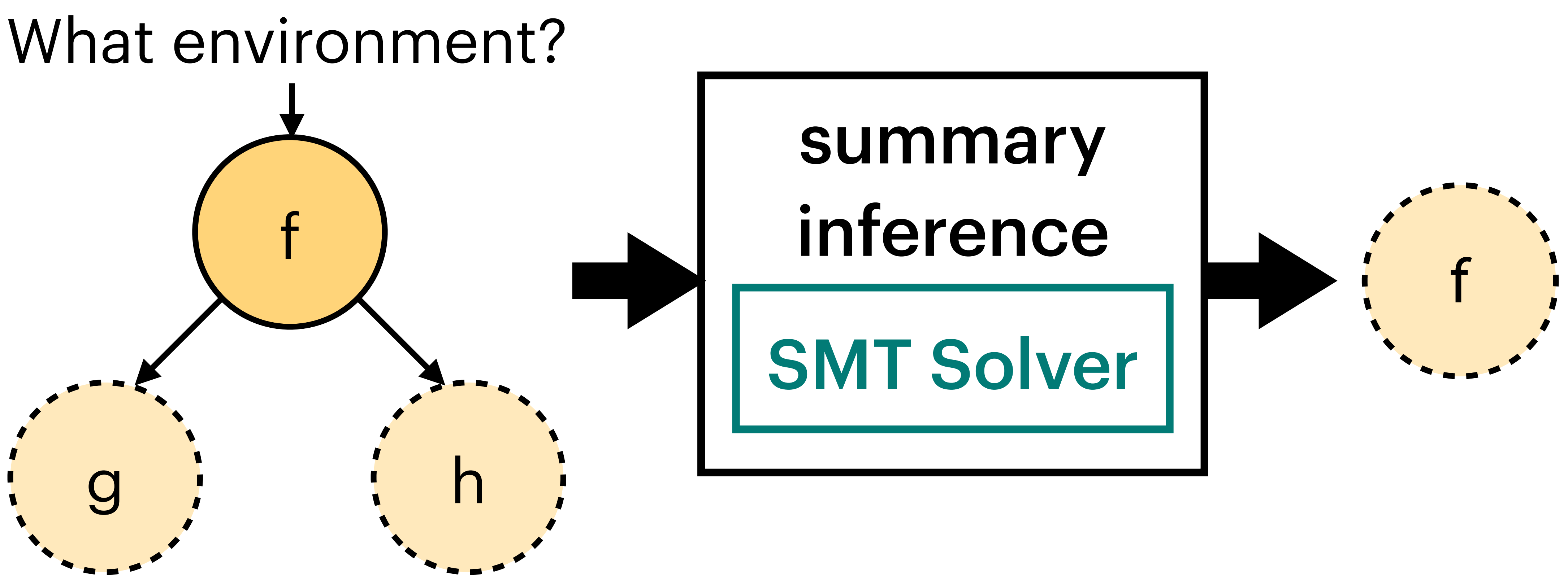


most scalable

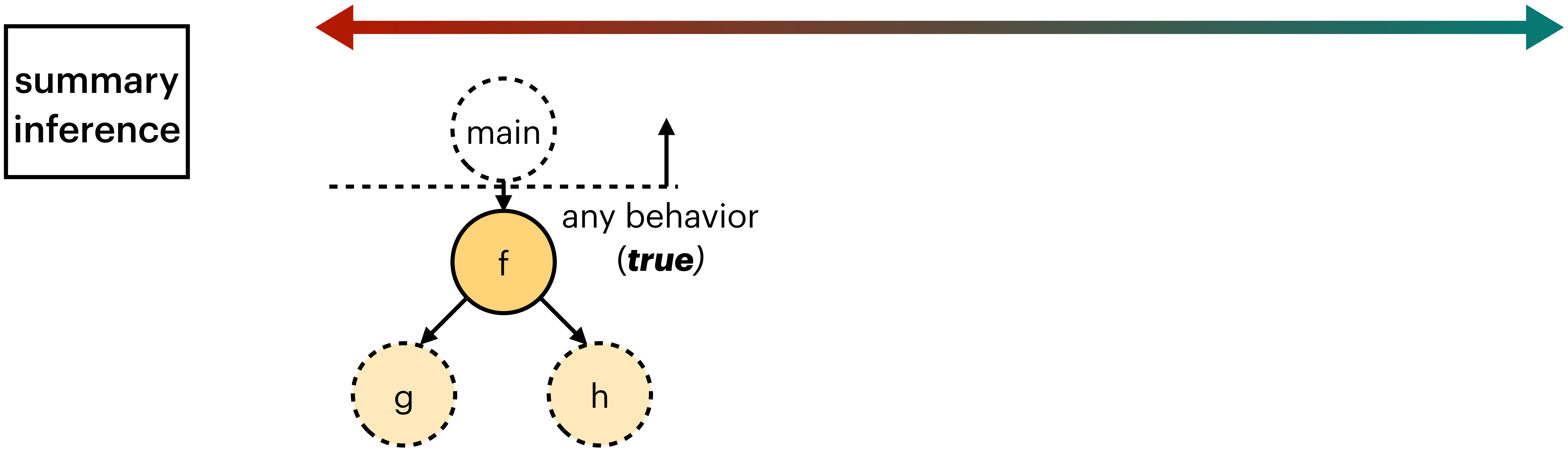
summary inference



Scalable Inference vs. Relevance of Invariants

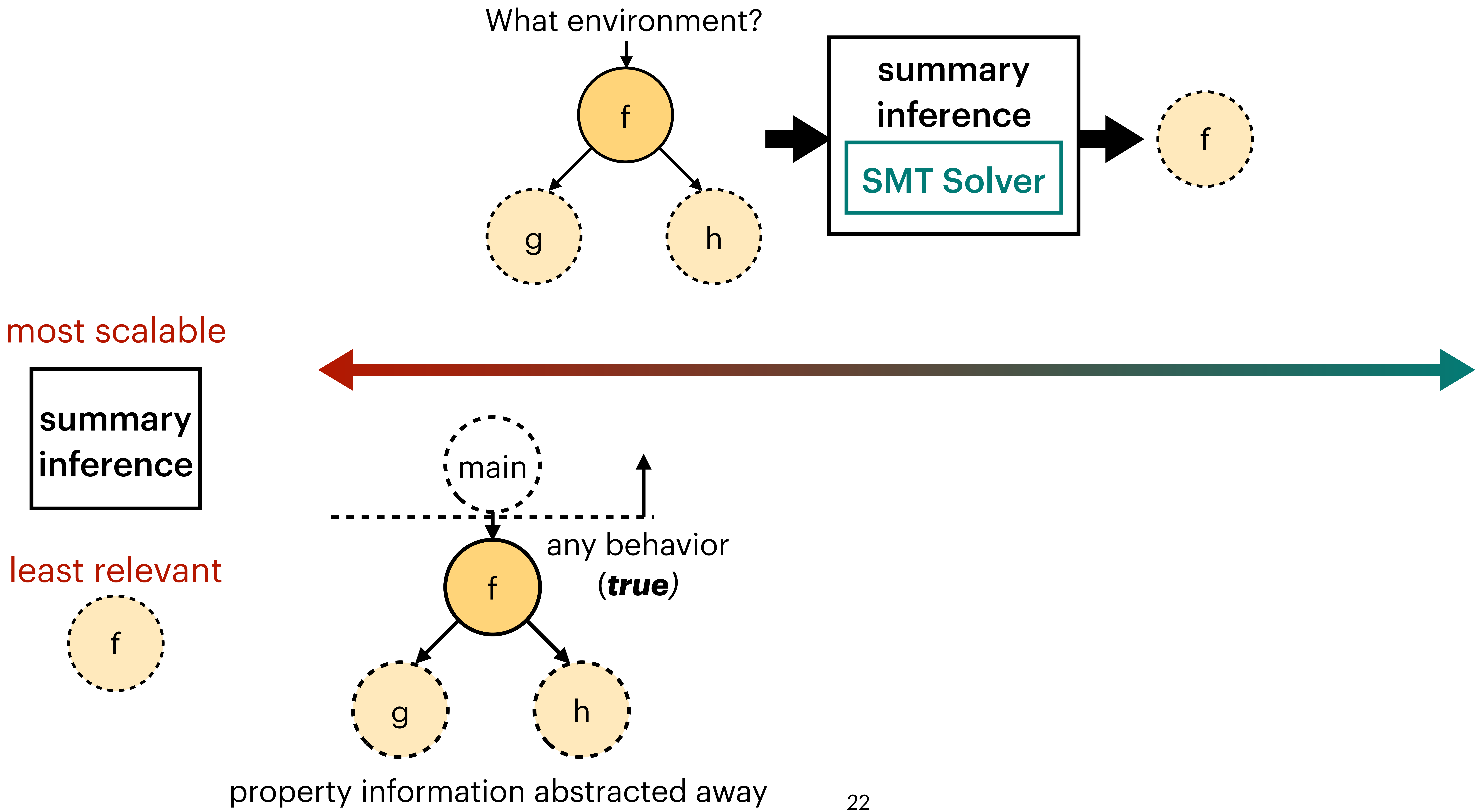


most scalable

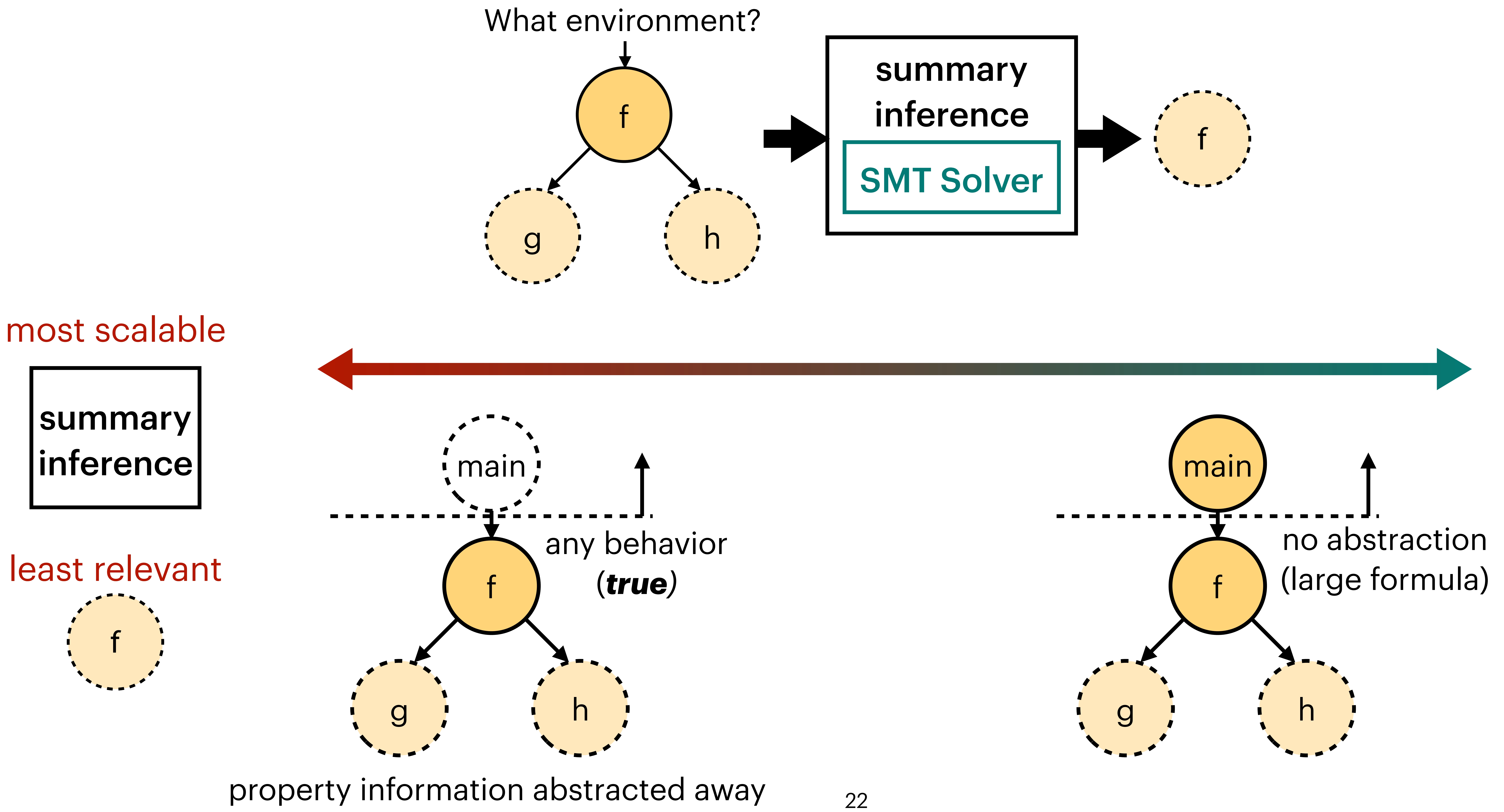


property information abstracted away

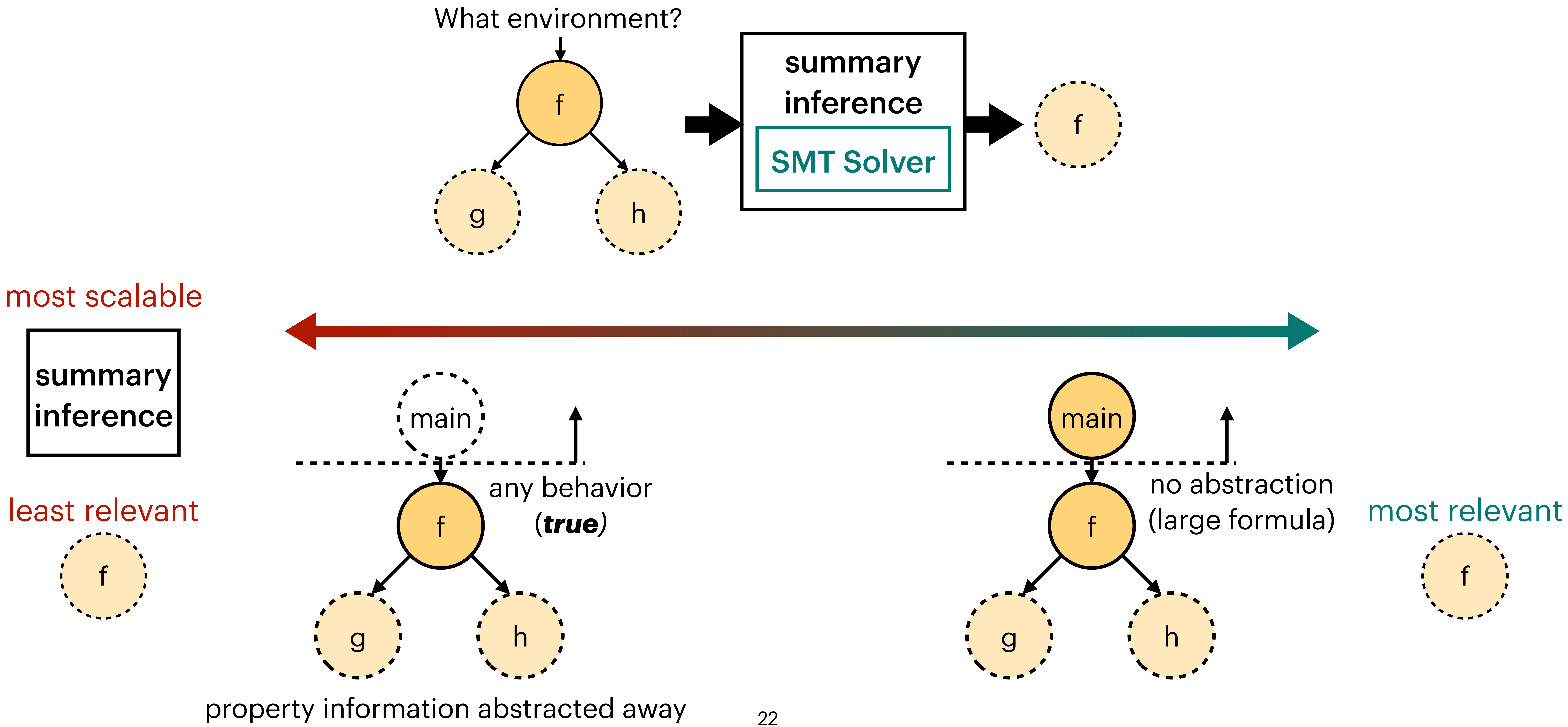
Scalable Inference vs. Relevance of Invariants



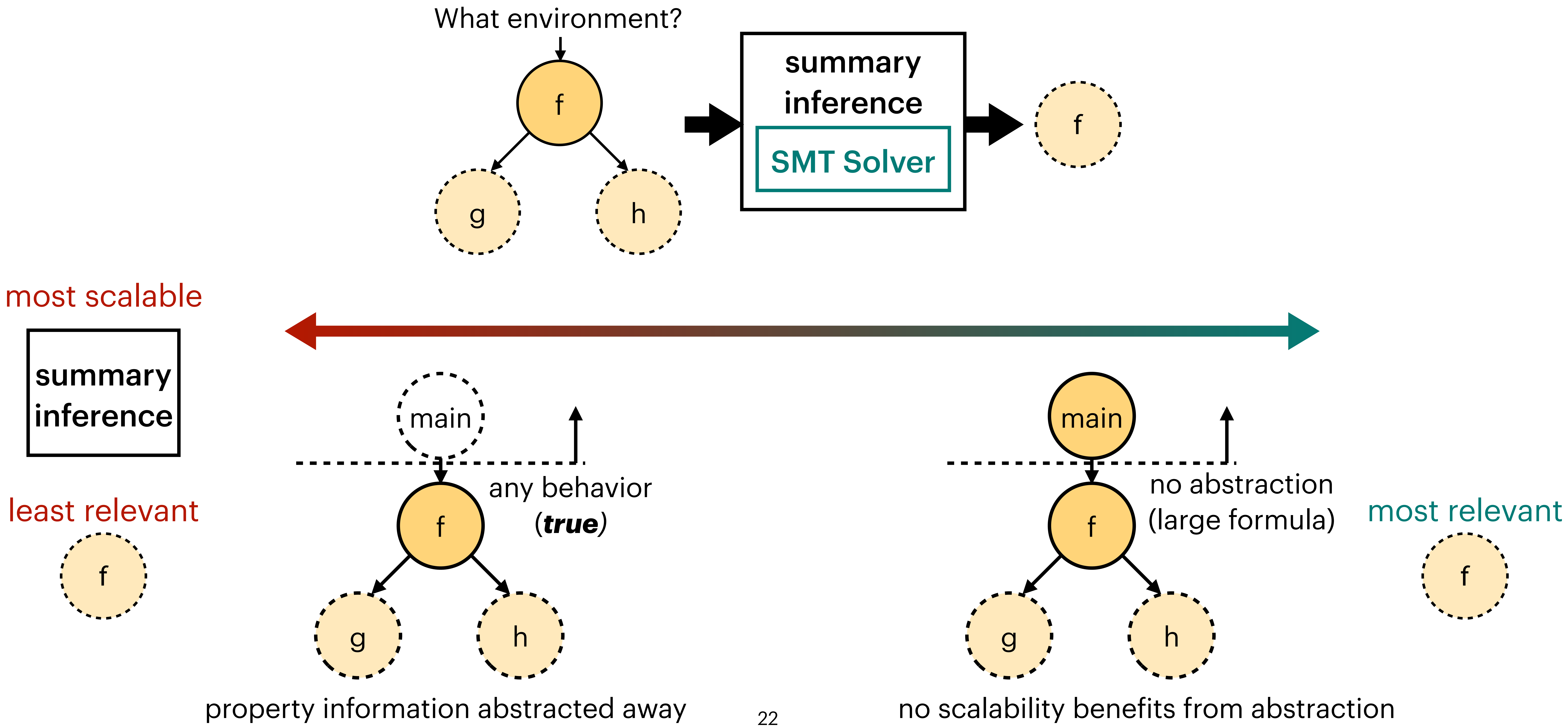
Scalable Inference vs. Relevance of Invariants



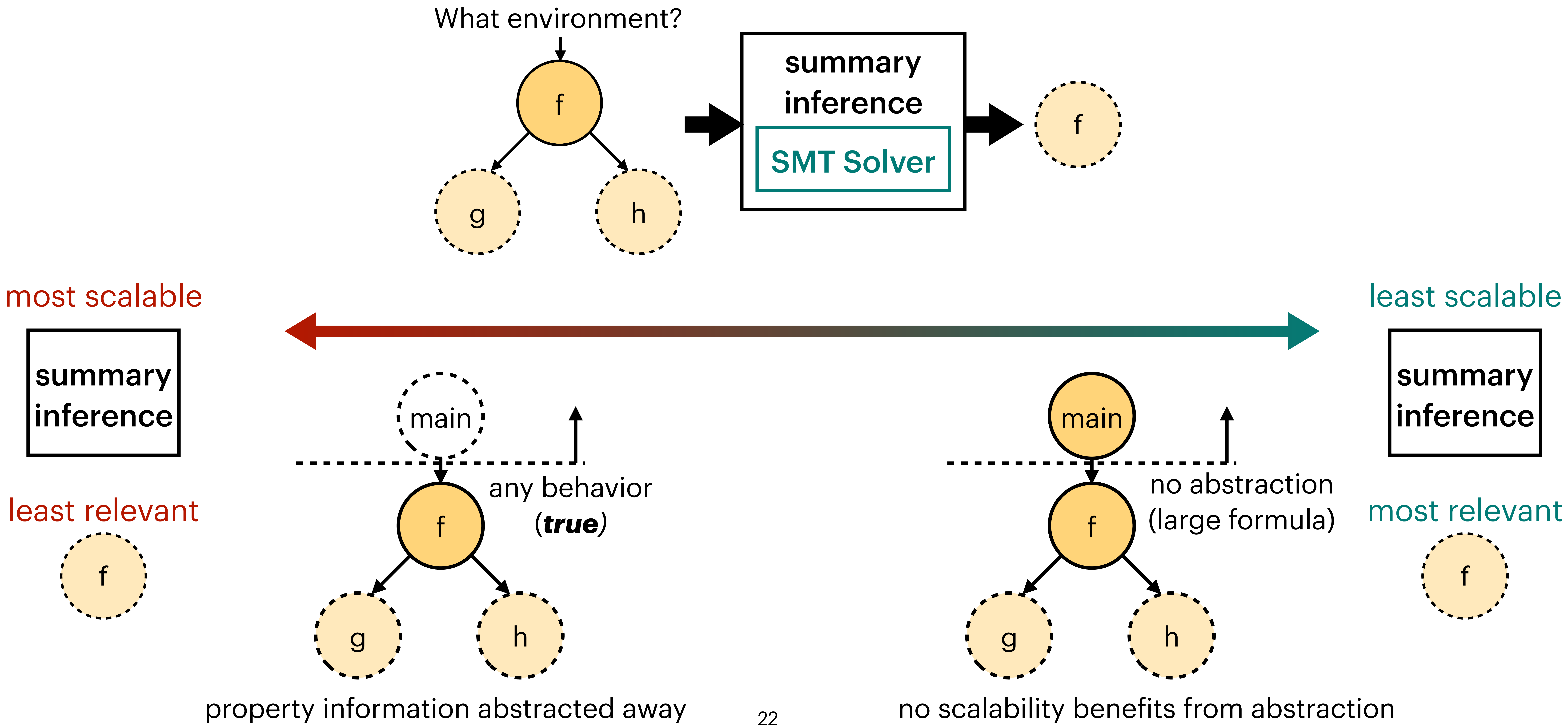
Scalable Inference vs. Relevance of Invariants



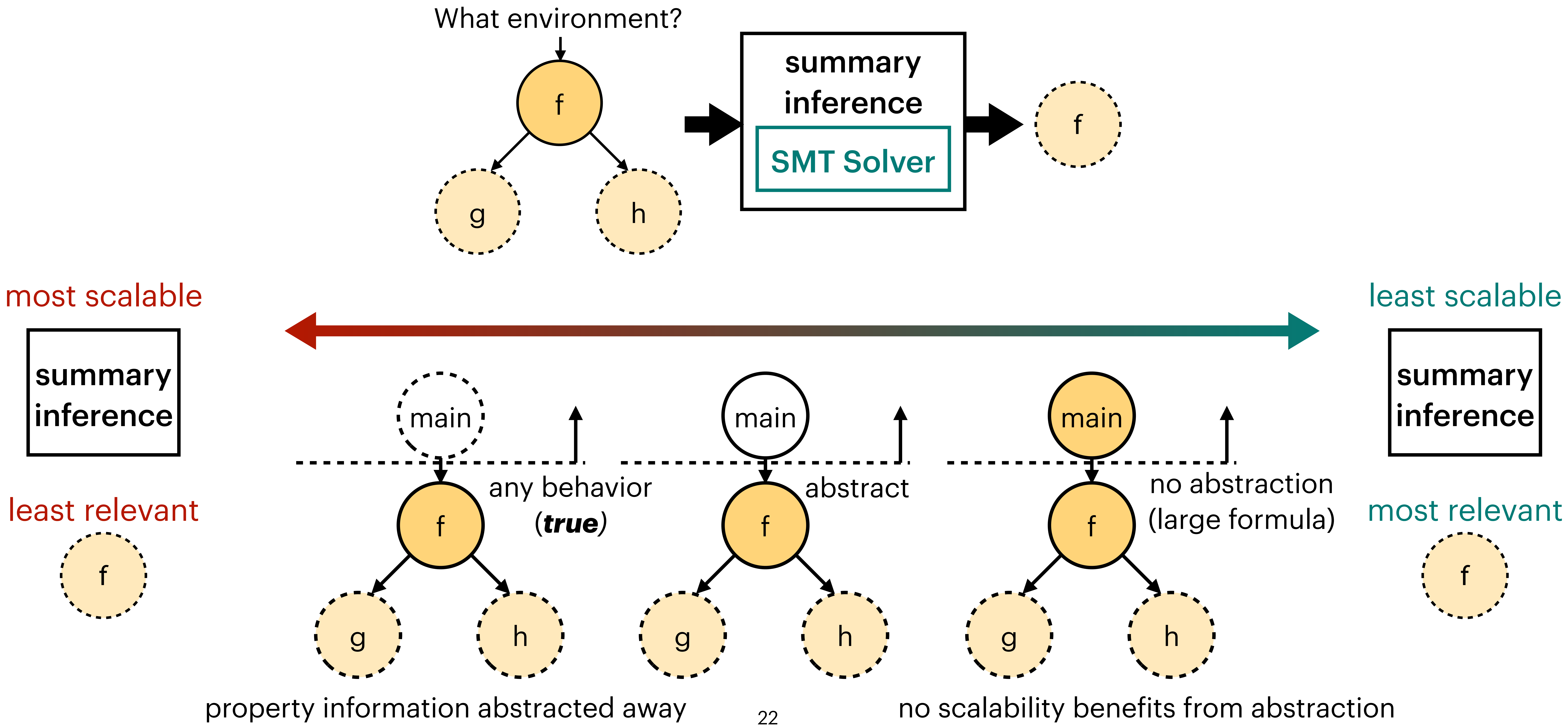
Scalable Inference vs. Relevance of Invariants



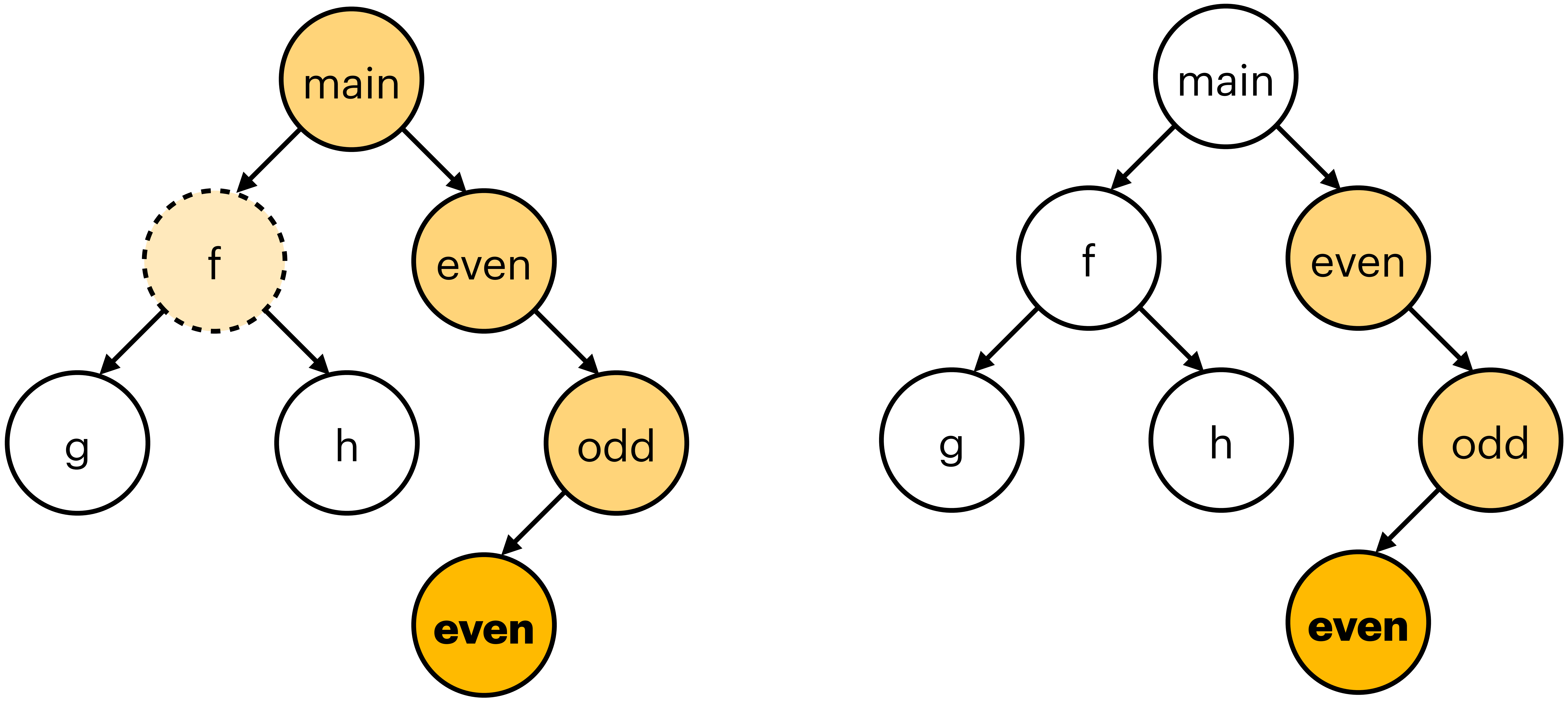
Scalable Inference vs. Relevance of Invariants



Scalable Inference vs. Relevance of Invariants

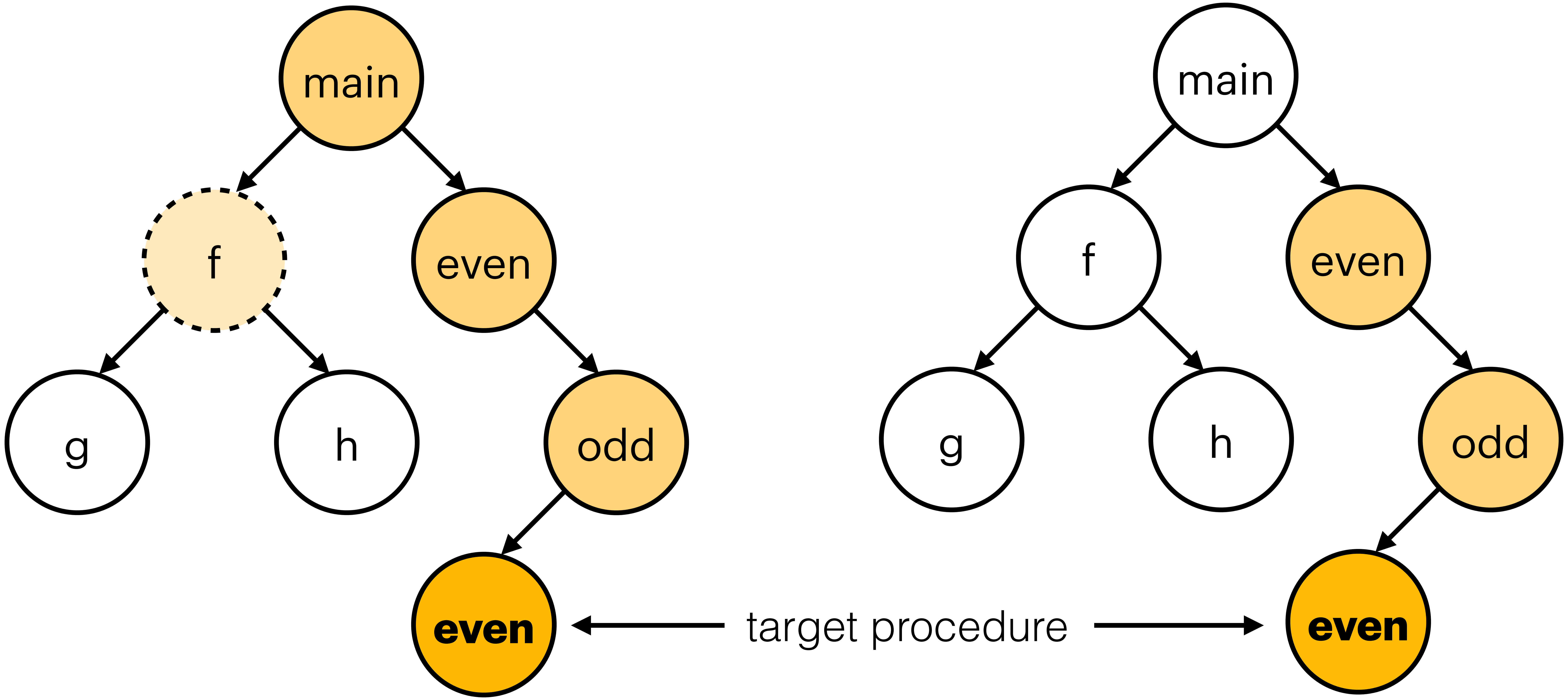


Bounded Environments



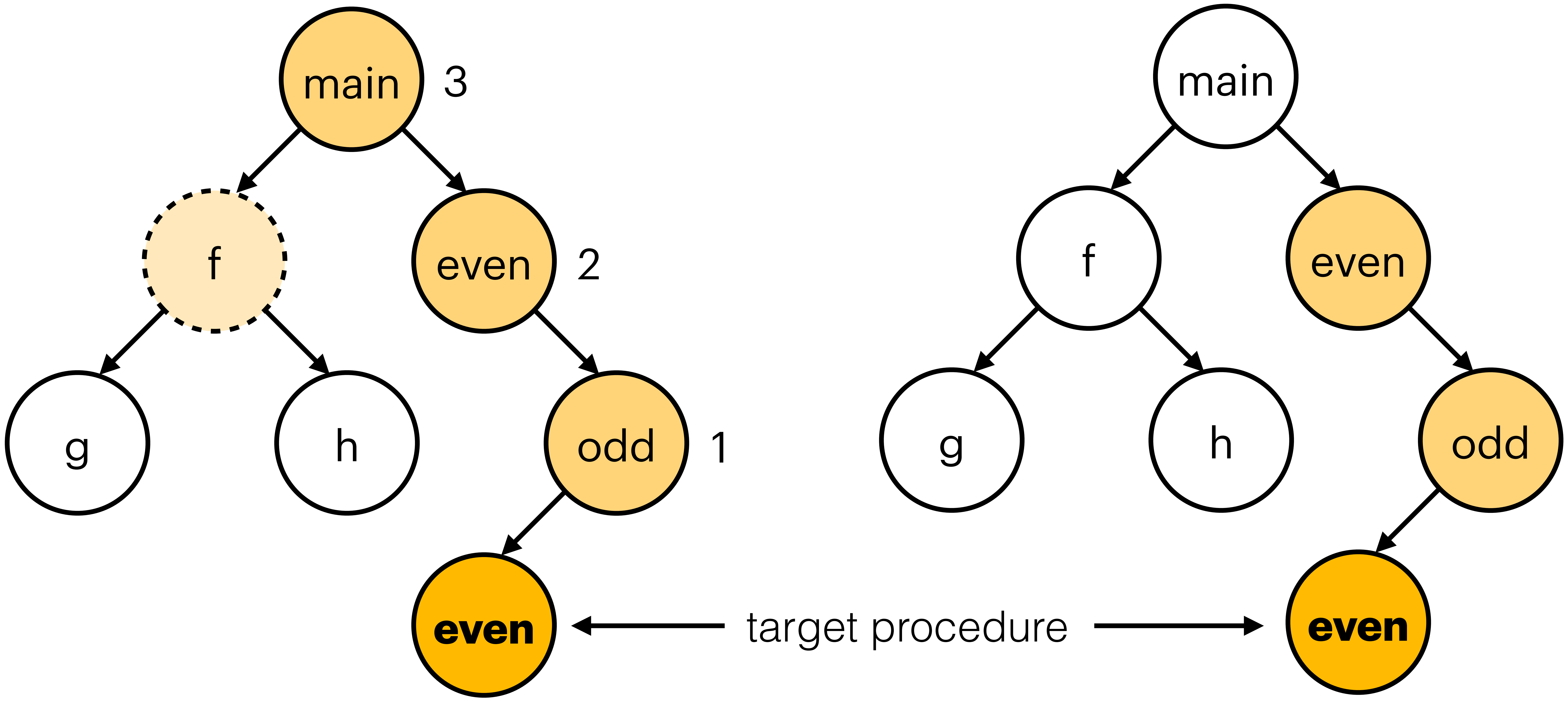
Unbounded Procedure Summaries from Bounded Environments, Pick et al., VMCAI'21

Bounded Environments



Unbounded Procedure Summaries from Bounded Environments, Pick et al., VMCAI'21

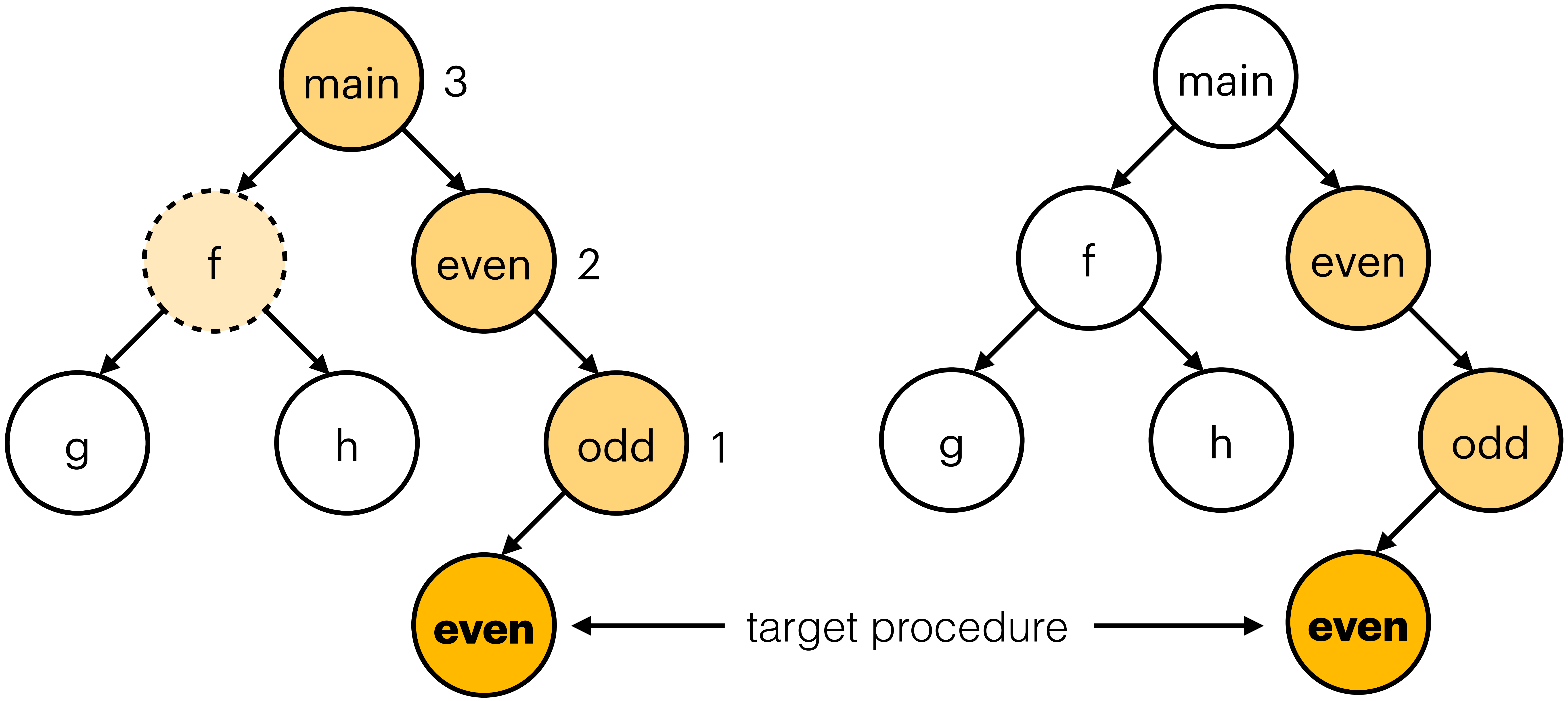
Bounded Environments



Unbounded Procedure Summaries from
Bounded Environments, Pick et al., VMCAI'21

Bounded Environments

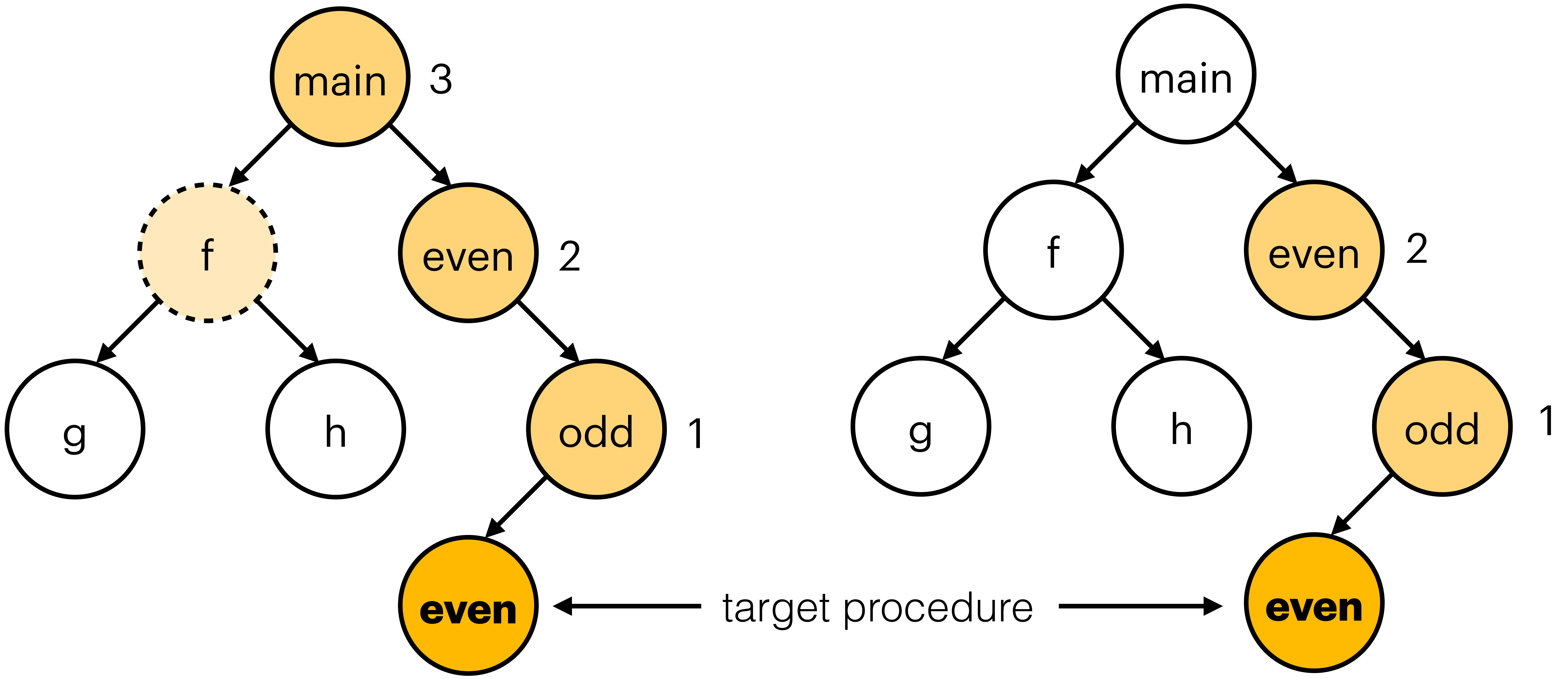
3-bounded environment



Unbounded Procedure Summaries from Bounded Environments, Pick et al., VMCAI'21

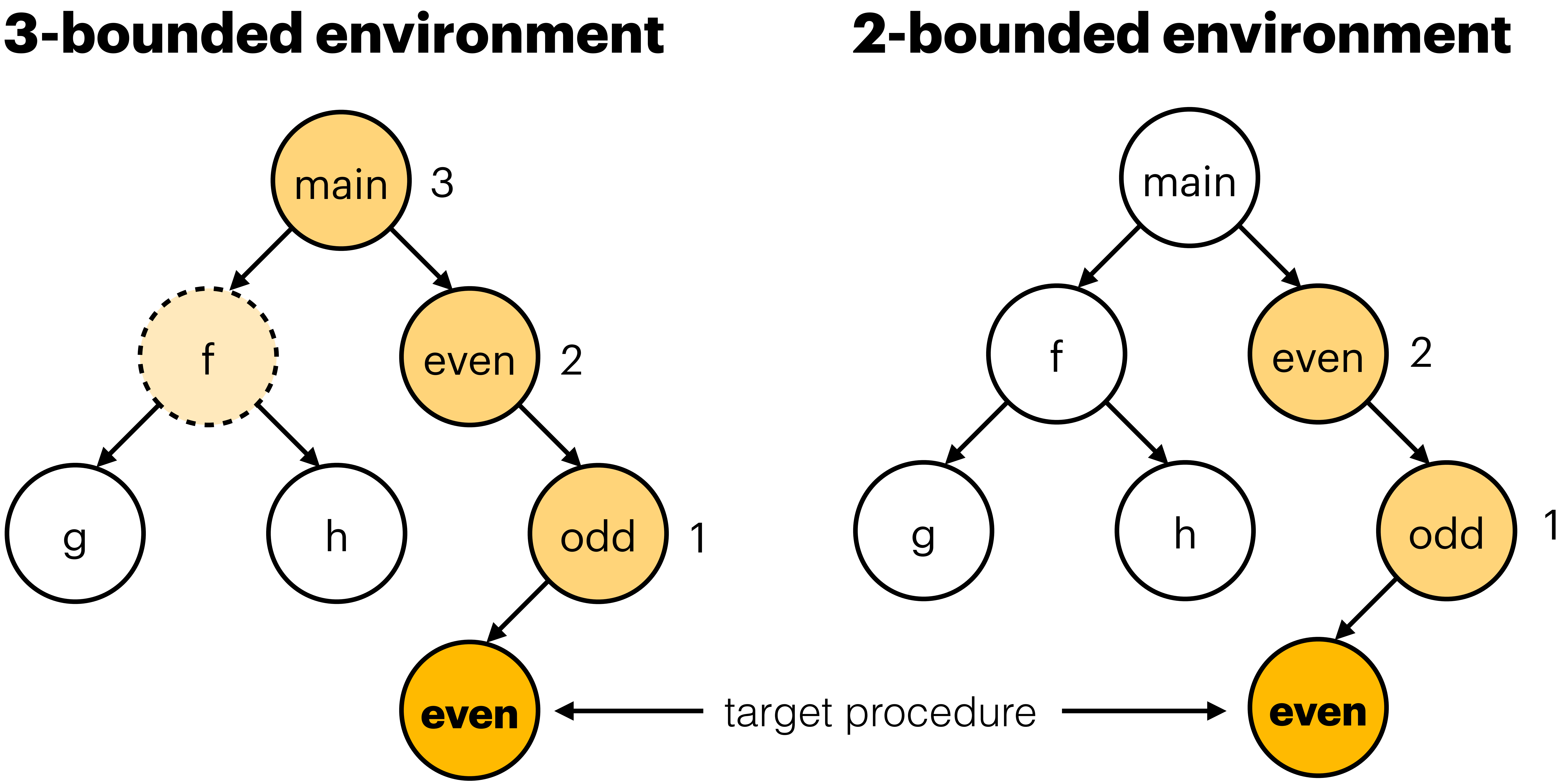
Bounded Environments

3-bounded environment



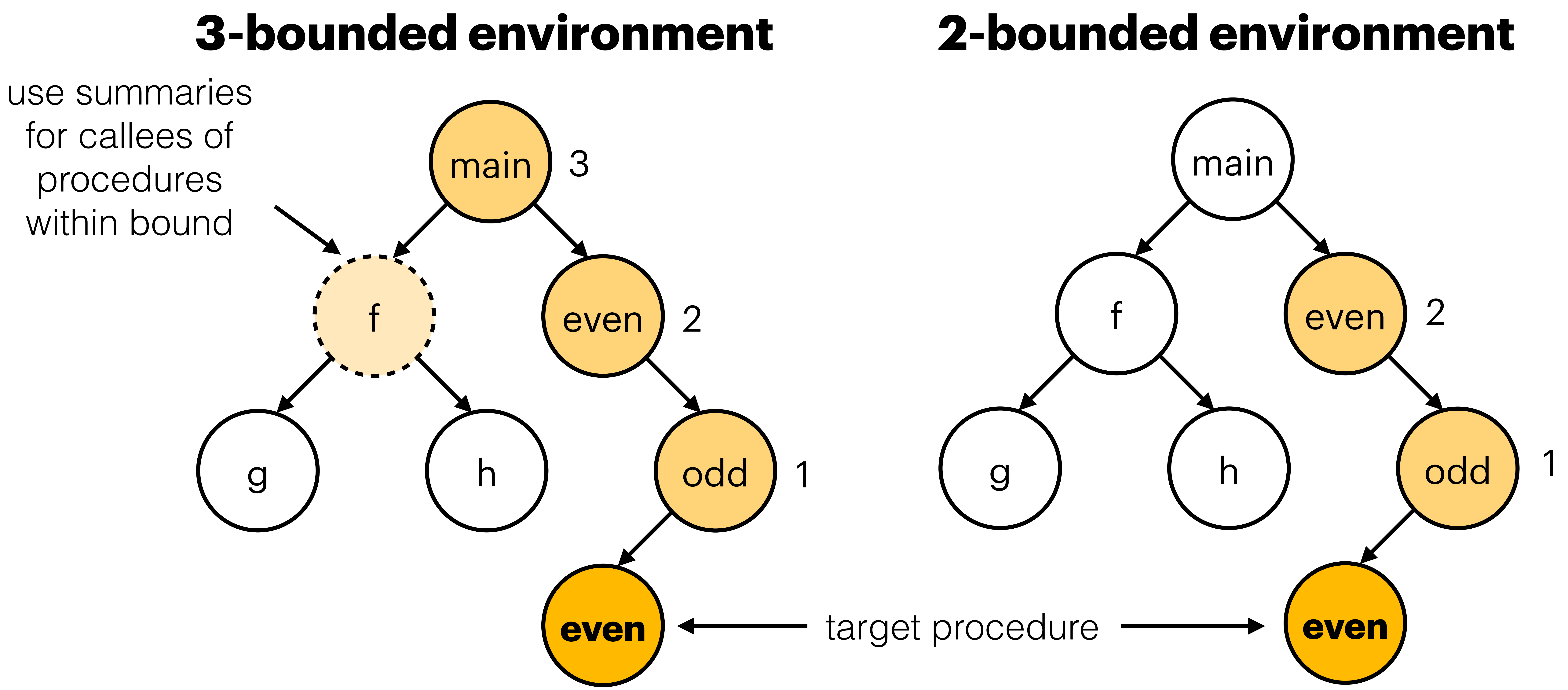
Unbounded Procedure Summaries from Bounded Environments, Pick et al., VMCAI’21

Bounded Environments



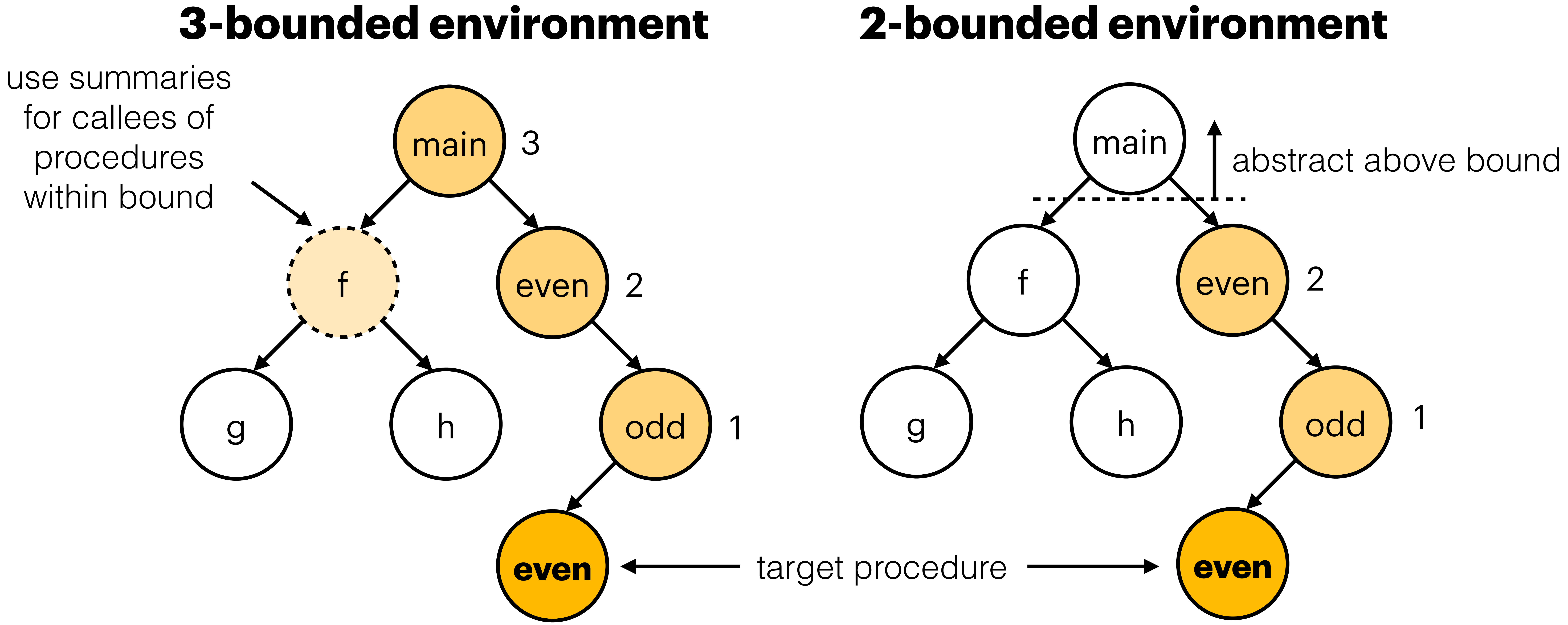
Unbounded Procedure Summaries from Bounded Environments, Pick et al., VMCAI'21

Bounded Environments



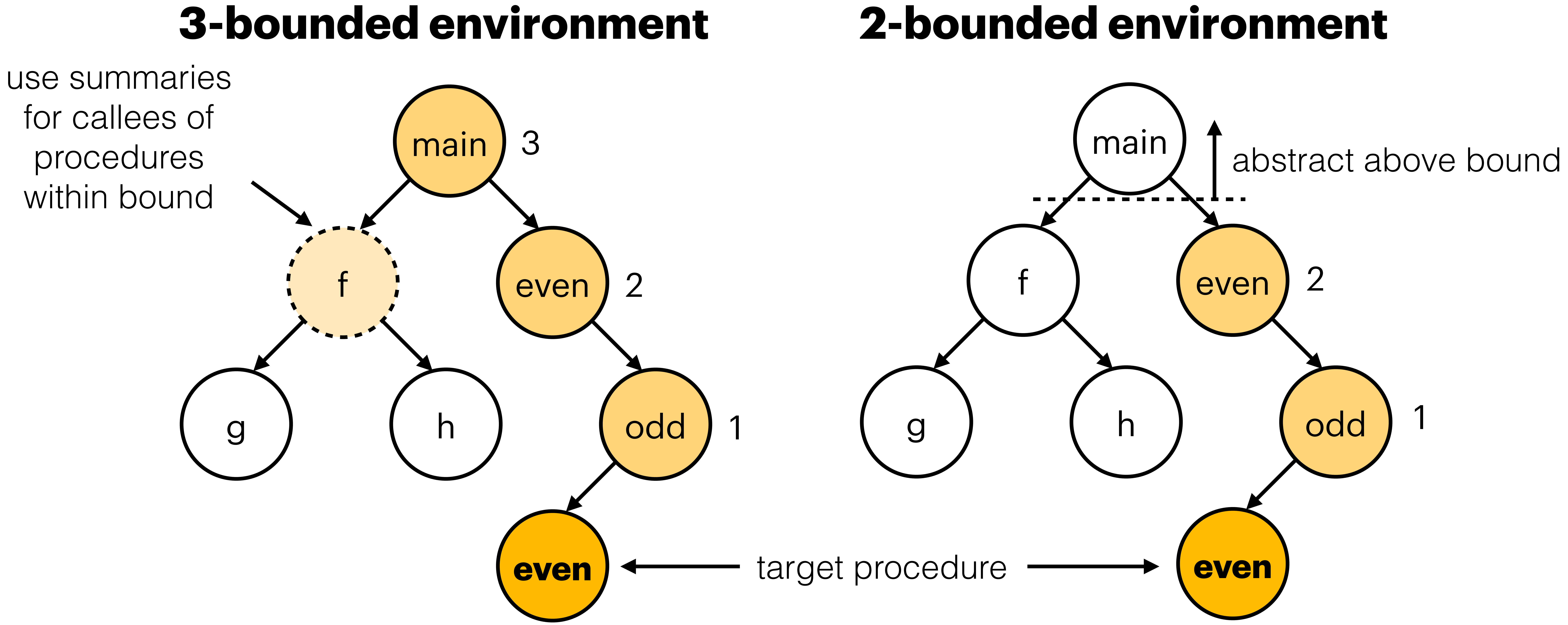
Unbounded Procedure Summaries from Bounded Environments, Pick et al., VMCAI'21

Bounded Environments



Unbounded Procedure Summaries from Bounded Environments, Pick et al., VMCAI'21

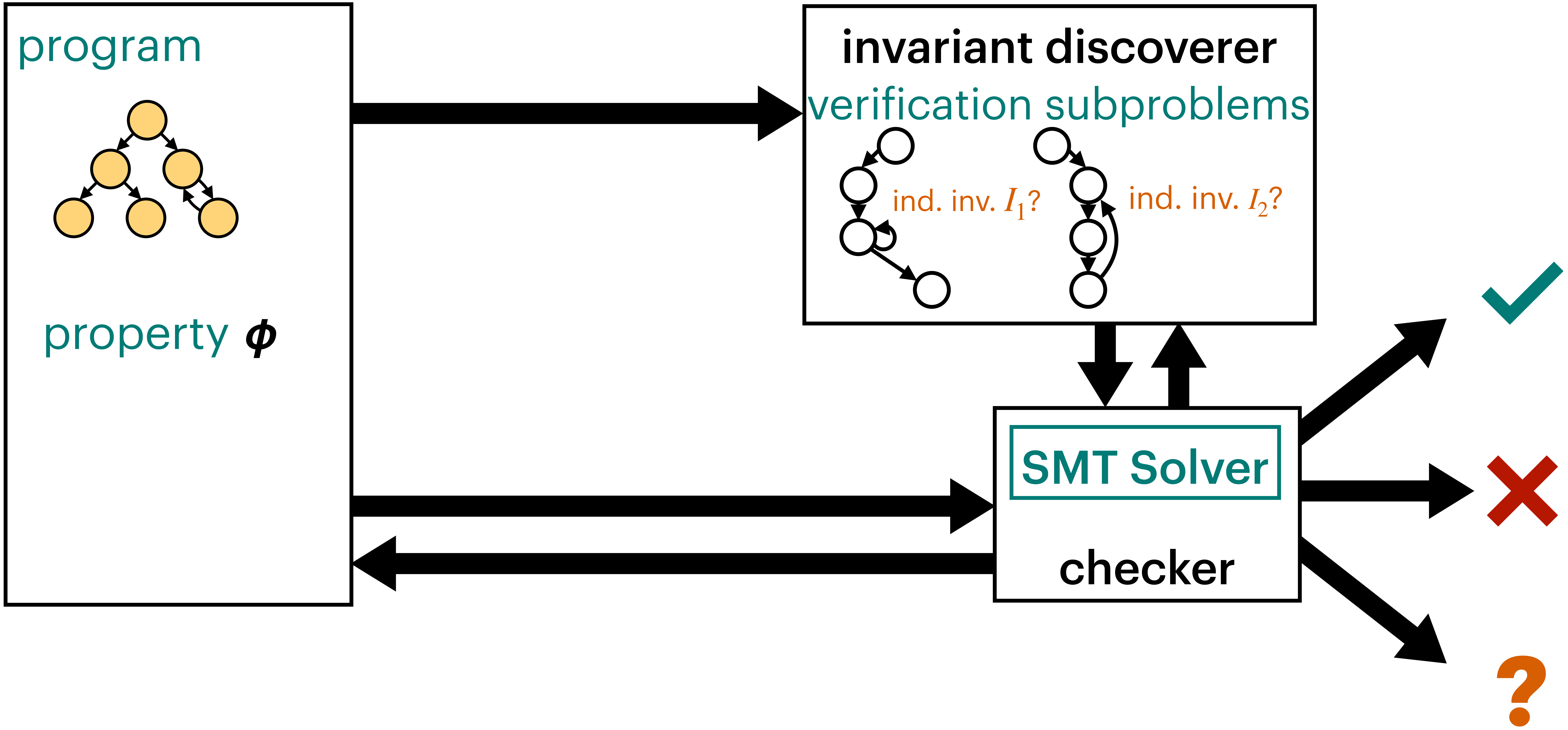
Bounded Environments



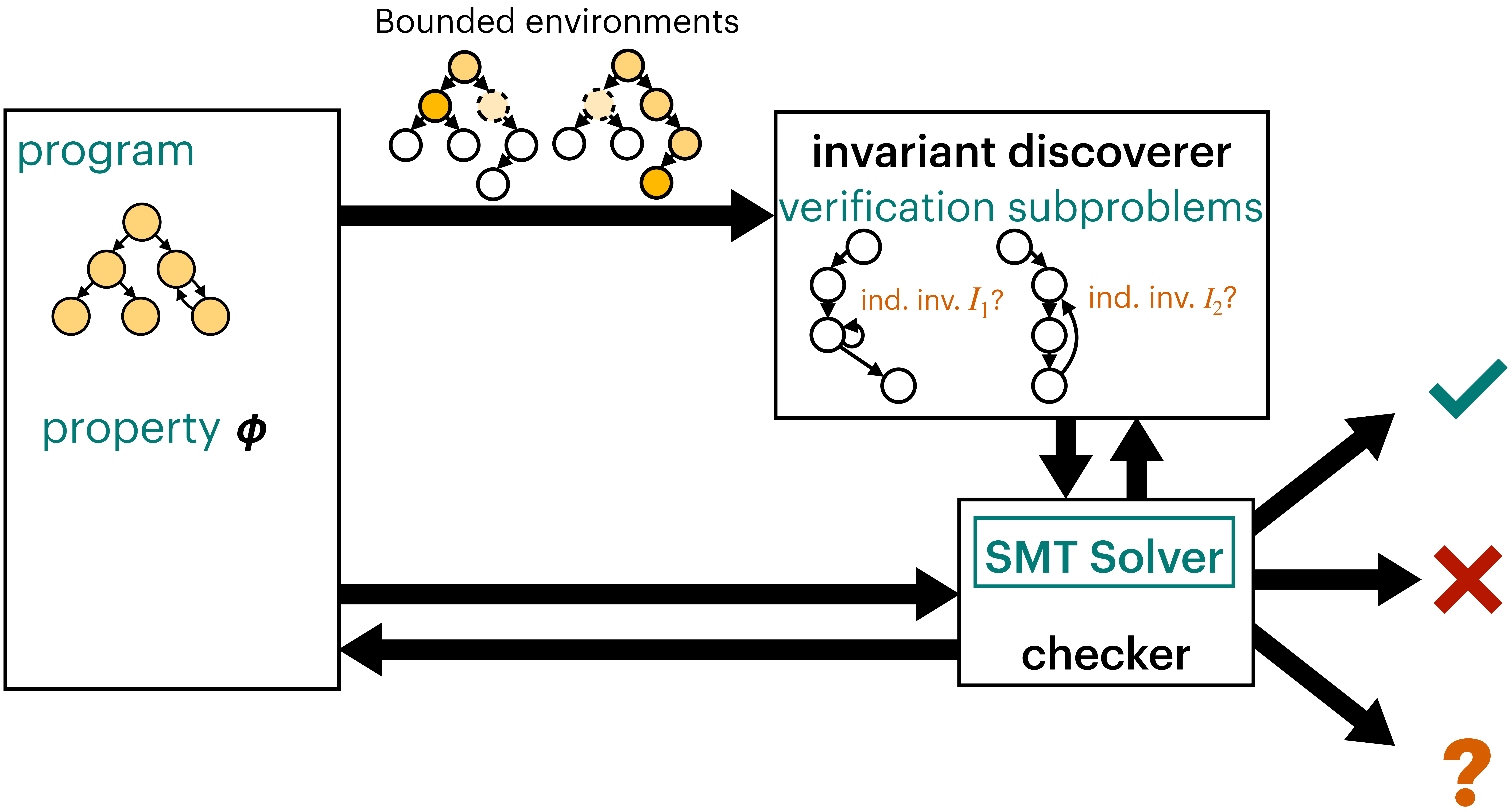
Larger bound, more relevant/less scalable

Unbounded Procedure Summaries from Bounded Environments, Pick et al., VMCAI'21

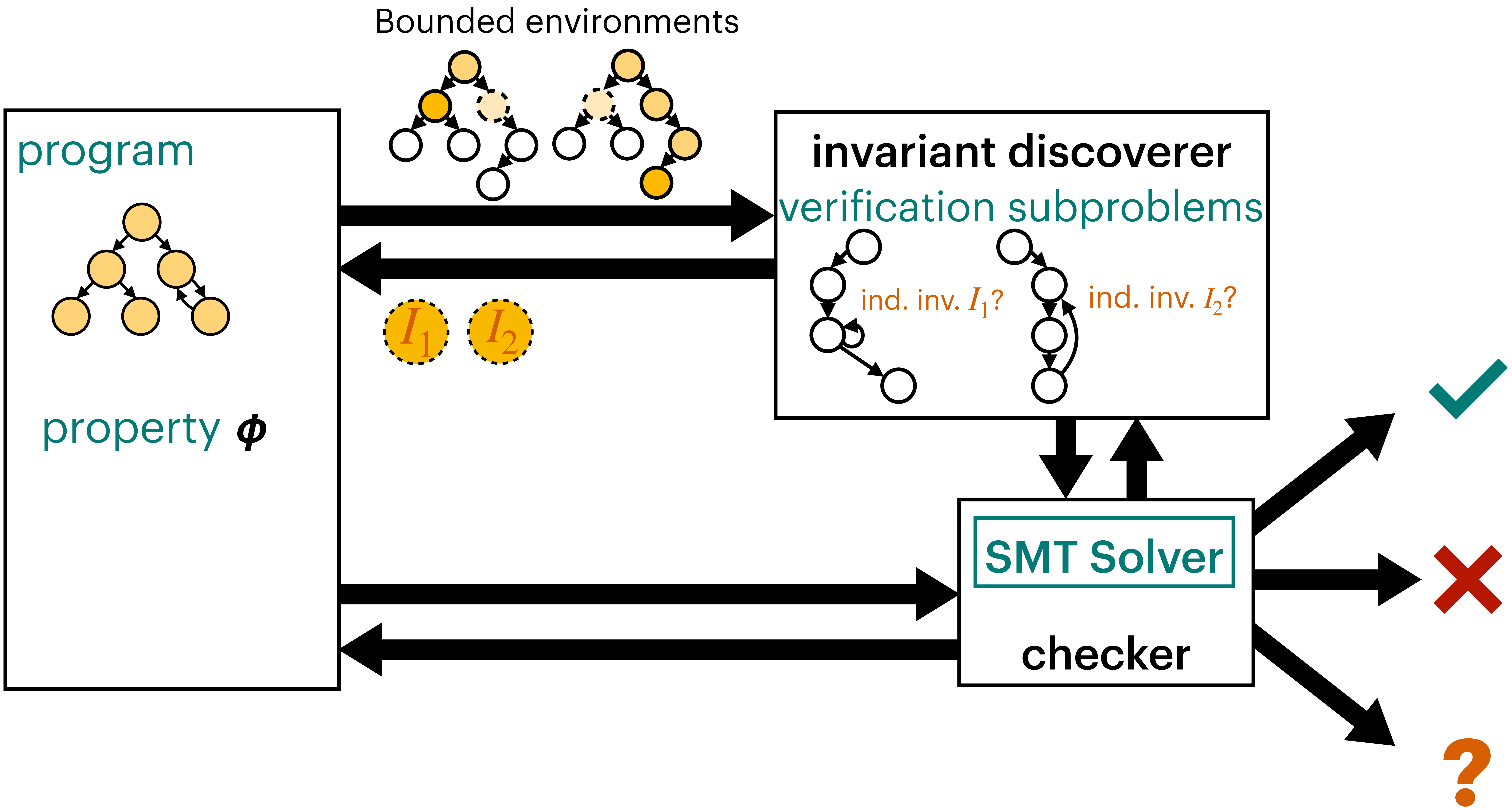
Interprocedural Program Verification



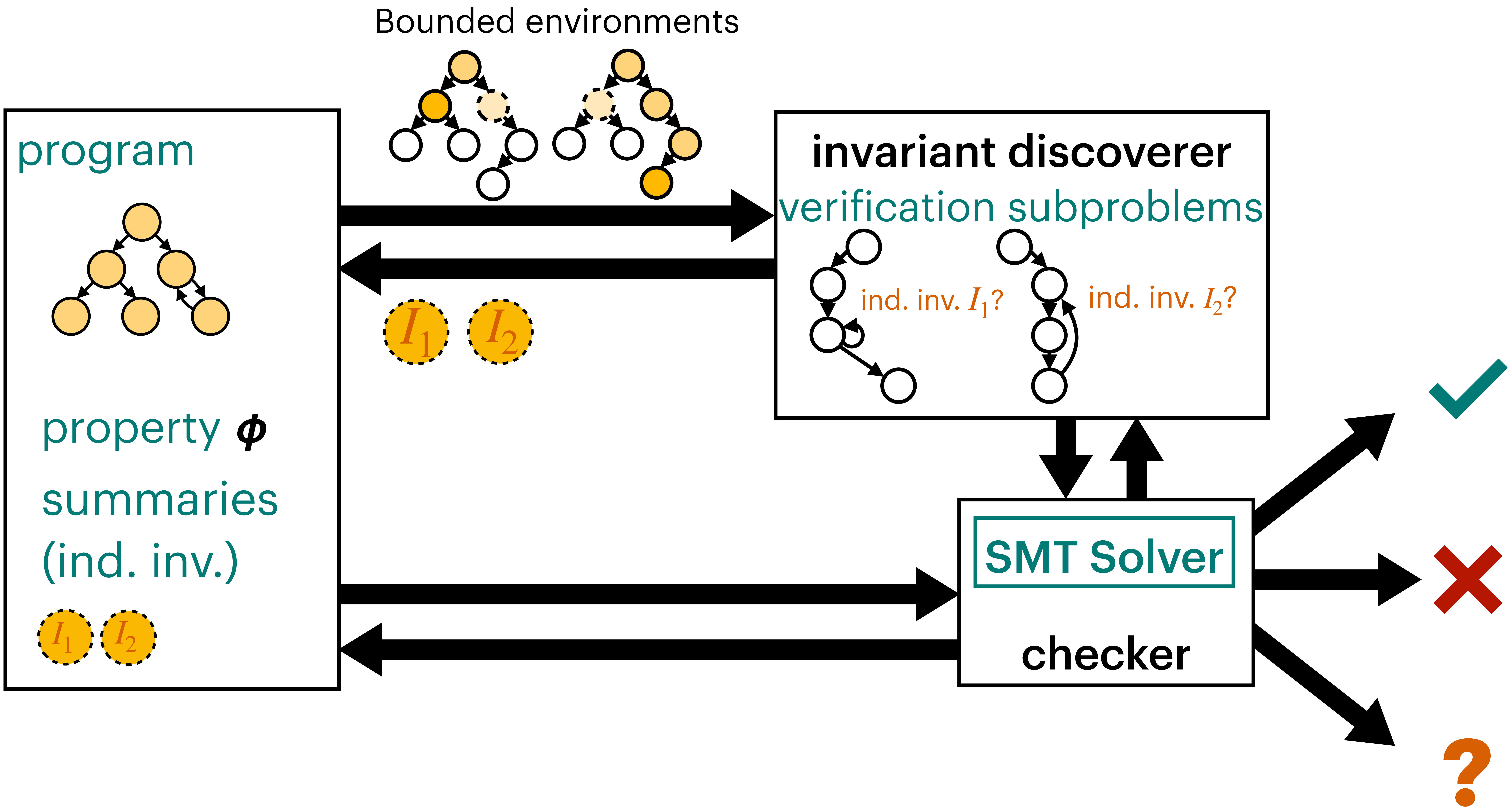
Interprocedural Program Verification



Interprocedural Program Verification



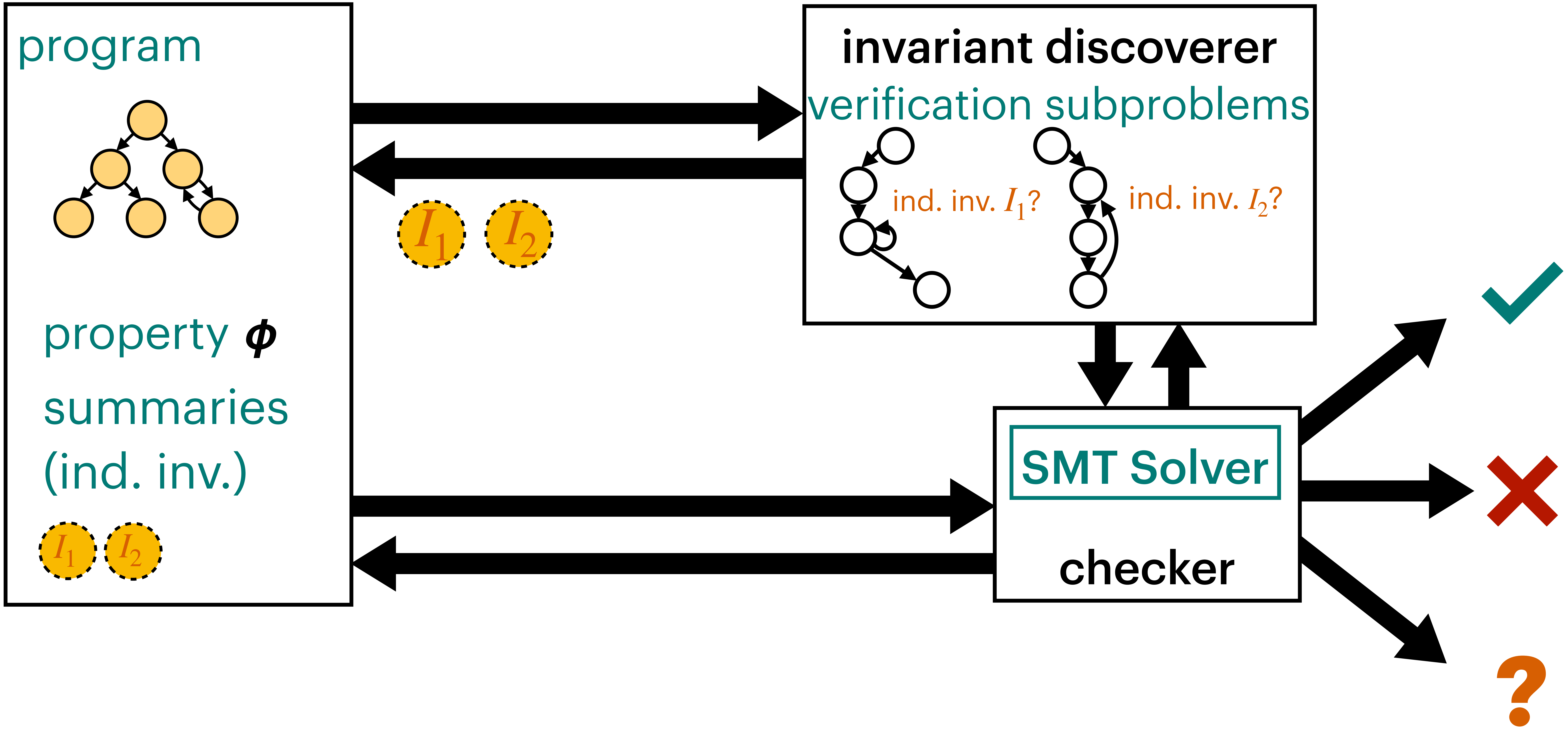
Interprocedural Program Verification



Interprocedural Program Verification

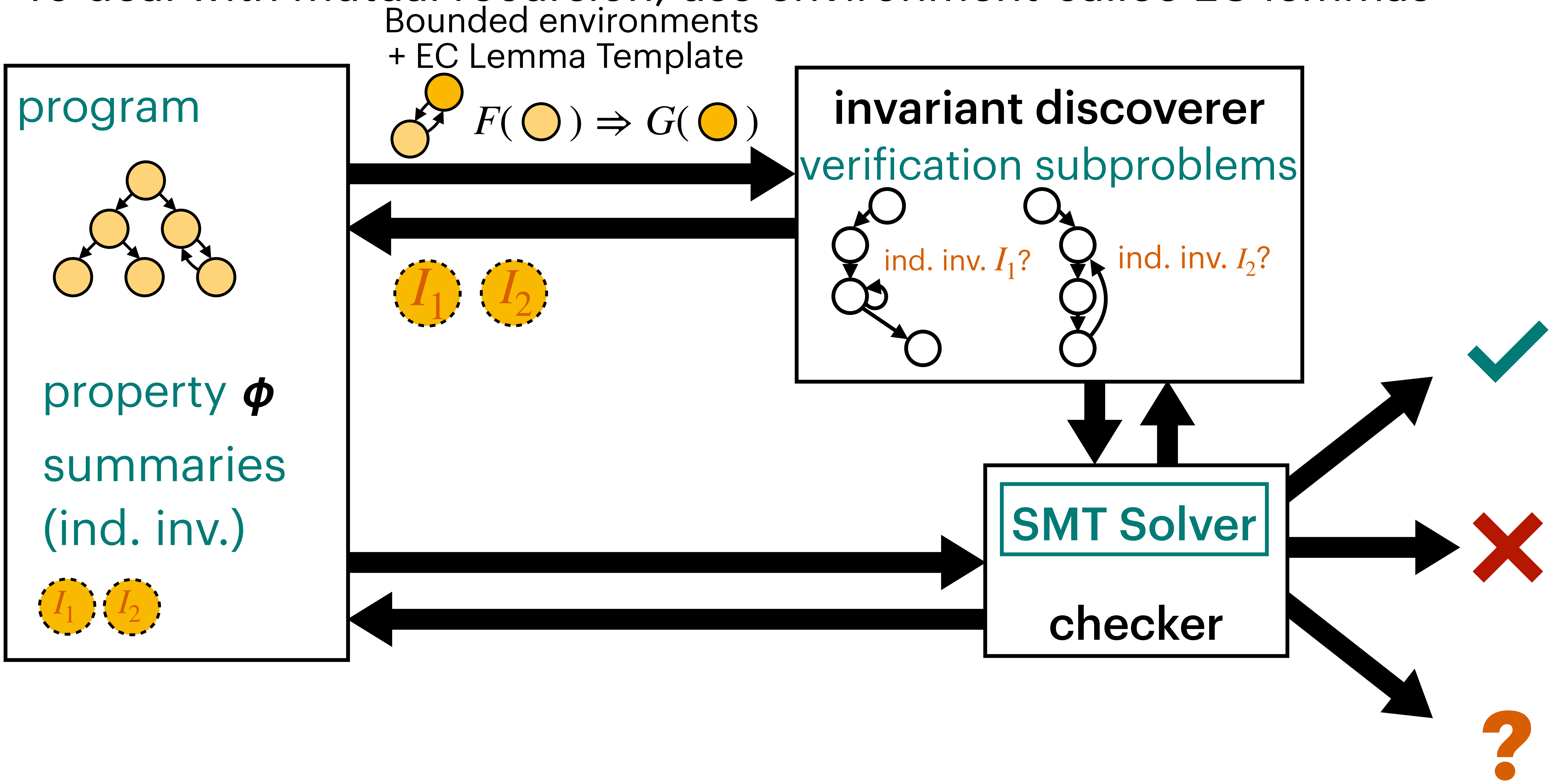
To deal with mutual recursion, use environment-callee EC lemmas

Bounded environments



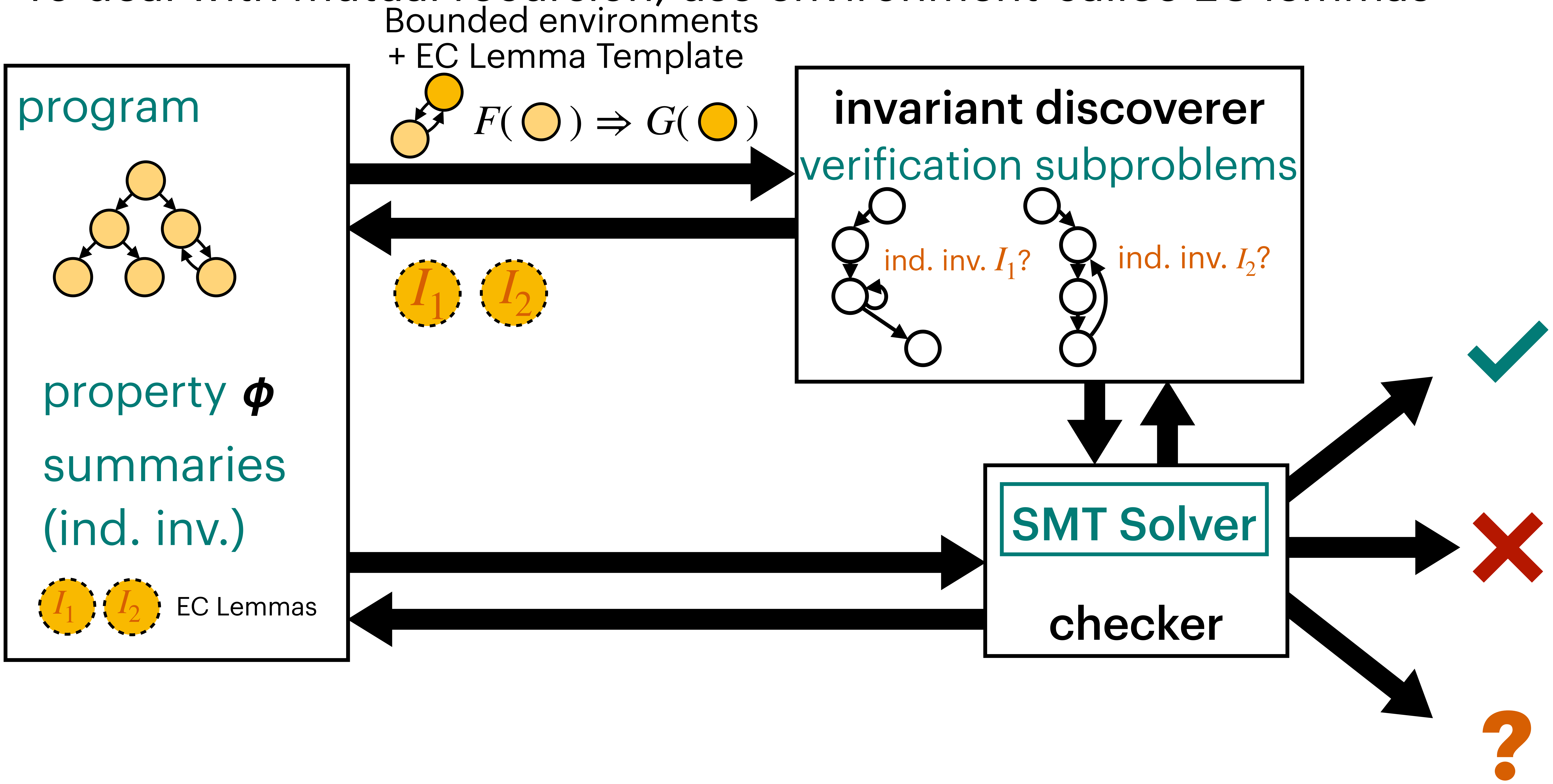
Interprocedural Program Verification

To deal with mutual recursion, use environment-callee EC lemmas



Interprocedural Program Verification

To deal with mutual recursion, use environment-callee EC lemmas



Experimental Results

Implemented in tool called Clover built on top of FreqHorn constrained Horn clause solver [Fedyukovich et al., 2017]

[1] Komuravelli et al., Formal Methods in Sys. Des.'16
[2] Hojjat and Rümer, FMCAD'18
[3] Champion et al., APLAS'18
[4] Satake et al., 2019
[5] Dietsch et al., HCVS/PERR'18

Experimental Results

Implemented in tool called Clover built on top of FreqHorn constrained Horn clause solver [Fedyukovich et al., 2017]

	Clover (b=10)	**Spacer** [1]	**Eldarica** [2]	**HoIce** [3]	**PCSat** [4]	**Ultimate** [5]
CHC-Comp (101)	77	93	**94**	92	81	76
Real World (16)	**16**	8	12	14	3	15
Mutual Recursion (46)	**45**	13	4	14	5	0
Total (163)	**138**	114	110	120	89	91

[1] Komuravelli et al., Formal Methods in Sys. Des.'16
[2] Hojjat and Rümer, FMCAD'18
[3] Champion et al., APLAS'18
[4] Satake et al., 2019
[5] Dietsch et al., HCVS/PERR'18

Experimental Results

Implemented in tool called Clover built on top of FreqHorn constrained Horn clause solver [Fedyukovich et al., 2017]

	Clover (b=10)	**Spacer** [1]	**Eldarica** [2]	**HoIce** [3]	**PCSat** [4]	**Ultimate** [5]
CHC-Comp (101)	77	93	**94**	92	81	76
Real World (16)	**16**	8	12	14	3	15
Mutual Recursion (46)	**45**	13	4	14	5	0
Total (163)	**138**	114	110	120	89	91

Comparable to other tools in general (timeout 10 min)

[1] Komuravelli et al., Formal Methods in Sys. Des.'16
[2] Hojjat and Rümer, FMCAD'18
[3] Champion et al., APLAS'18
[4] Satake et al., 2019
[5] Dietsch et al., HCVS/PERR'18

Experimental Results

Implemented in tool called Clover built on top of FreqHorn constrained Horn clause solver [Fedyukovich et al., 2017]

	Clover (b=10)	**Spacer** [1]	**Eldarica** [2]	**HoIce** [3]	**PCSat** [4]	**Ultimate** [5]
CHC-Comp (101)	77	93	**94**	92	81	76
Real World (16)	**16**	8	12	14	3	15
Mutual Recursion (46)	**45**	13	4	14	5	0
Total (163)	**138**	114	110	120	89	91

Comparable to other tools in general (timeout 10 min), excels at mutual recursion

[1] Komuravelli et al., Formal Methods in Sys. Des.'16

[2] Hojjat and Rümer, FMCAD'18

[3] Champion et al., APLAS'18

[4] Satake et al., 2019

[5] Dietsch et al., HCVS/PERR'18

Experimental Results

Experimental Results

EC Lemmas are useful!

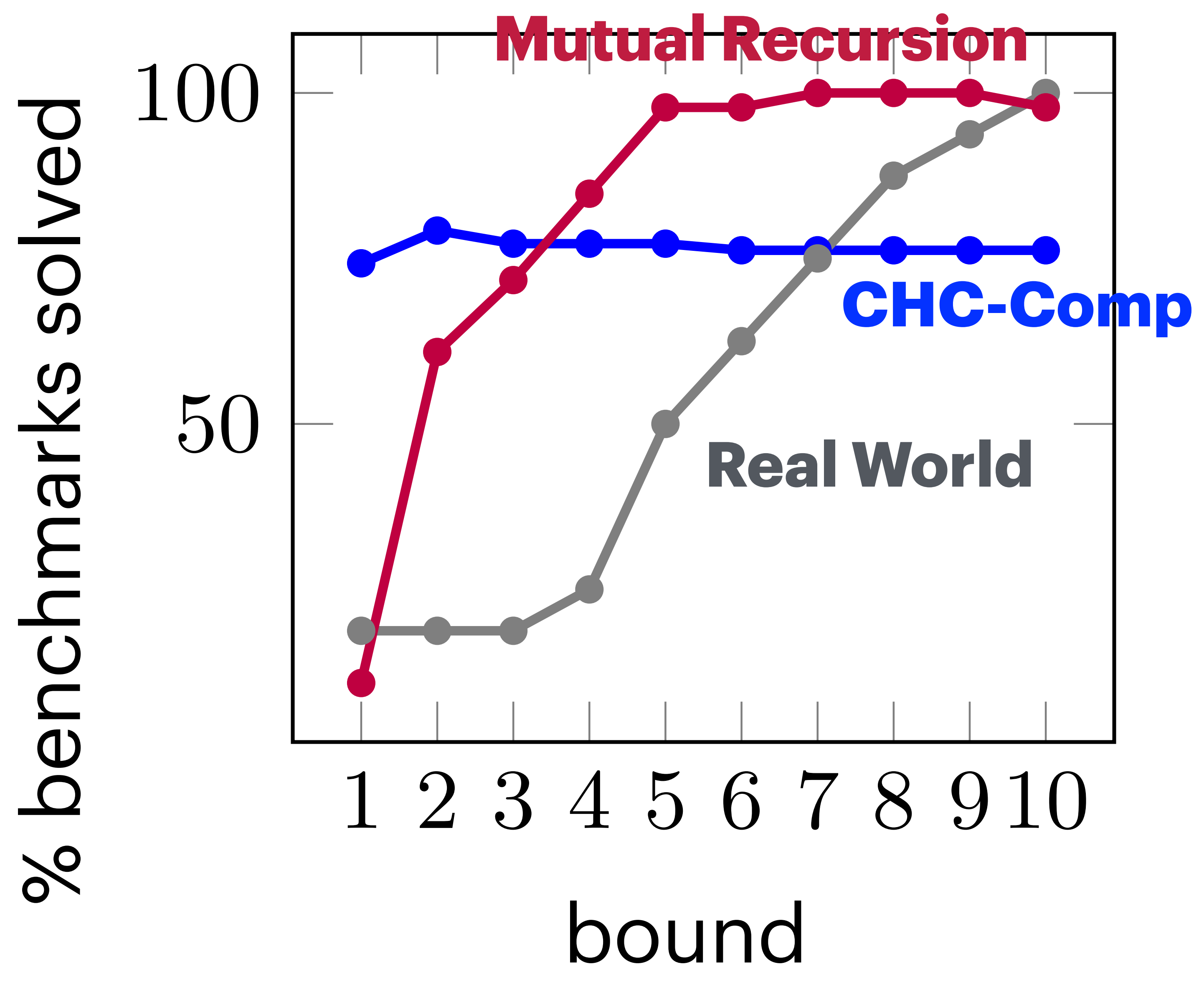
	Clover (b=10)	Clover (b=10), no EC lemmas
CHC-Comp	**77**	72
Real World	16	16
Mutual Recursion	**45**	5
Total	**138**	93

Experimental Results

EC Lemmas are useful!

	Clover (b=10)	Clover (b=10), no EC lemmas
CHC-Comp	**77**	72
Real World	16	16
Mutual Recursion	**45**	5
Total	**138**	93

Different bounds help for different benchmark sets

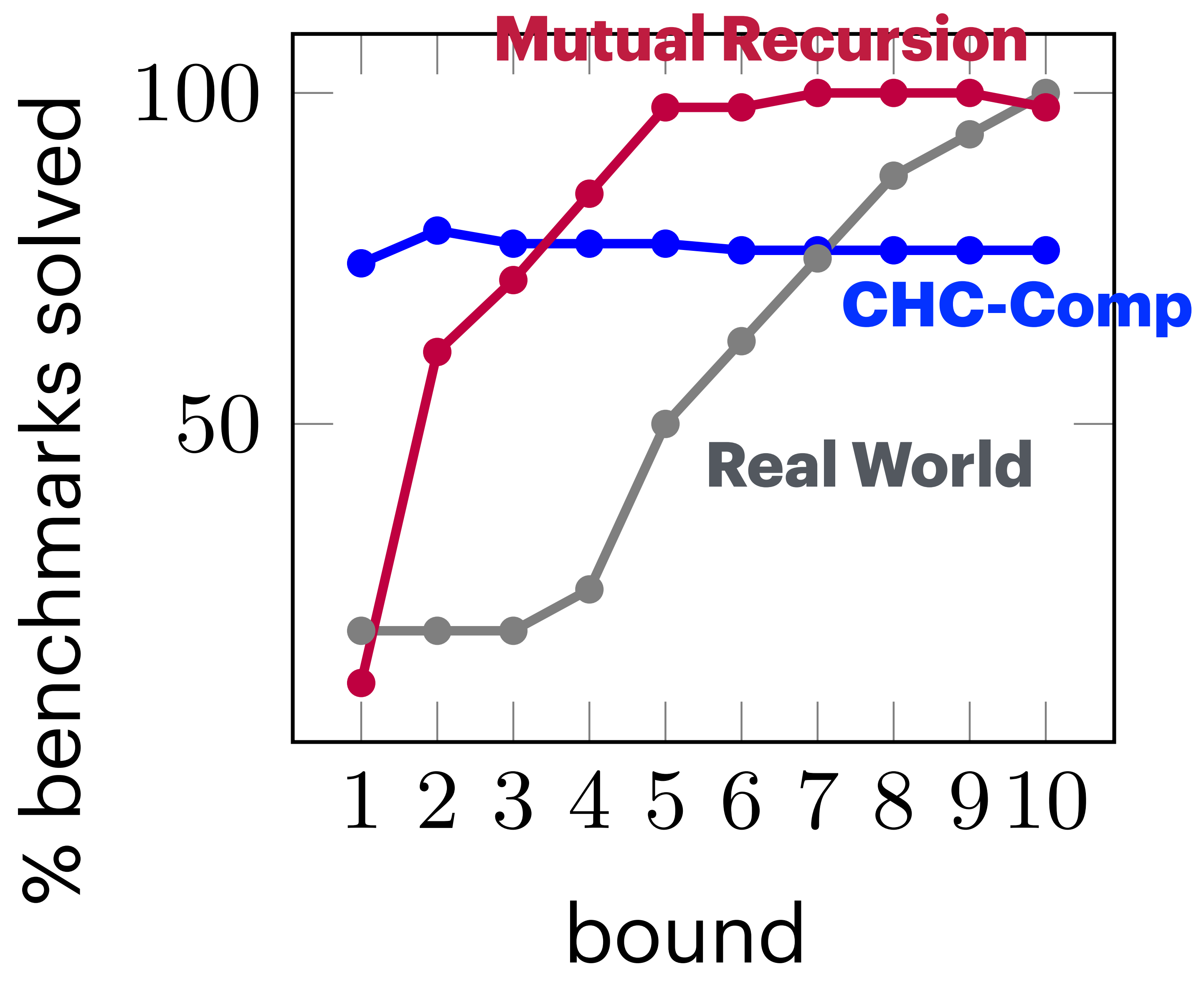


Experimental Results

EC Lemmas are useful!

	Clover (b=10)	Clover (b=10), no EC lemmas
CHC-Comp	**77**	72
Real World	16	16
Mutual Recursion	**45**	5
Total	**138**	93

Different bounds help for different benchmark sets



e.g., bounds 7-9 were best for Mutual Recursion

Related Work

Related Work

Constrained-Horn-Clause-Based Program Verification

[Komuravelli et al., Formal Methods in Sys. Des.'16]
[McMillan, CAV'14]
[Hojjat and Rümer, FMCAD'18]
[Champion et al., APLAS'18]
[Dietsch et al., EPTCS'19]
[Grebenshchikov et al., PLDI'12]
[McMillan and Rybalchenko, 2013]

Related Work

Constrained-Horn-Clause-Based Program Verification

[Komuravelli et al., Formal Methods in Sys. Des.'16]
[McMillan, CAV'14]
[Hojjat and Rümer, FMCAD'18]
[Champion et al., APLAS'18]
[Dietsch et al., EPTCS'19]
[Grebenshchikov et al., PLDI'12]
[McMillan and Rybalchenko, 2013]

Program Analysis and Verification

Related Work

Constrained-Horn-Clause-Based Program Verification

[Komuravelli et al., Formal Methods in Sys. Des.'16]
[McMillan, CAV'14]
[Hojjat and Rümer, FMCAD'18]
[Champion et al., APLAS'18]
[Dietsch et al., EPTCS'19]
[Grebenshchikov et al., PLDI'12]
[McMillan and Rybalchenko, 2013]

Program Analysis and Verification

Abstract Interpretation

[Cousot and Cousot, IFIP'77]
[Cousot and Cousot, VMCAI'13]
[Fähndrich et al., FoVeOOS'10]

Related Work

Constrained-Horn-Clause-Based Program Verification

[Komuravelli et al., Formal Methods in Sys. Des.'16]
[McMillan, CAV'14]
[Hojjat and Rümer, FMCAD'18]
[Champion et al., APLAS'18]
[Dietsch et al., EPTCS'19]
[Grebenshchikov et al., PLDI'12]
[McMillan and Rybalchenko, 2013]

Program Analysis and Verification

Abstract Interpretation

[Cousot and Cousot, IFIP'77]
[Cousot and Cousot, VMCAI'13]
[Fähndrich et al., FoVeOOS'10]

Interprocedural Dataflow Analysis

[Reps et al., POPL'95]
[Ball and Rajamani, PASTE'01]

Related Work

Constrained-Horn-Clause-Based Program Verification

[Komuravelli et al., Formal Methods in Sys. Des.'16]
[McMillan, CAV'14]
[Hojjat and Rümer, FMCAD'18]
[Champion et al., APLAS'18]
[Dietsch et al., EPTCS'19]
[Grebenshchikov et al., PLDI'12]
[McMillan and Rybalchenko, 2013]

Program Analysis and Verification

Abstract Interpretation

[Cousot and Cousot, IFIP'77]
[Cousot and Cousot, VMCAI'13]
[Fähndrich et al., FoVeOOS'10]

Interprocedural Dataflow Analysis

[Reps et al., POPL'95]
[Ball and Rajamani, PASTE'01]

Summary Usage

[Godefroid et al., POPL'10]

Related Work

Constrained-Horn-Clause-Based Program Verification

[Komuravelli et al., Formal Methods in Sys. Des.'16]
[McMillan, CAV'14]
[Hojjat and Rümer, FMCAD'18]
[Champion et al., APLAS'18]
[Dietsch et al., EPTCS'19]
[Grebenshchikov et al., PLDI'12]
[McMillan and Rybalchenko, 2013]

Specification Inference

[Albargouthi et al., POPL'16]
[Alur et al., POPL'05]
[Ammons et al., POPL'02]

Program Analysis and Verification

Abstract Interpretation

[Cousot and Cousot, IFIP'77]
[Cousot and Cousot, VMCAI'13]
[Fähndrich et al., FoVeOOS'10]

Interprocedural Dataflow Analysis

[Reps et al., POPL'95]
[Ball and Rajamani, PASTE'01]

Summary Usage

[Godefroid et al., POPL'10]

Related Work

Constrained-Horn-Clause-Based Program Verification

[Komuravelli et al., Formal Methods in Sys. Des.'16]
[McMillan, CAV'14]
[Hojjat and Rümer, FMCAD'18]
[Champion et al., APLAS'18]
[Dietsch et al., EPTCS'19]
[Grebenshchikov et al., PLDI'12]
[McMillan and Rybalchenko, 2013]

Specification Inference

[Albargouthi et al., POPL'16]
[Alur et al., POPL'05]
[Ammons et al., POPL'02]

Program Analysis and Verification

Abstract Interpretation

[Cousot and Cousot, IFIP'77]
[Cousot and Cousot, VMCAI'13]
[Fähndrich et al., FoVeOOS'10]

Interprocedural Dataflow Analysis

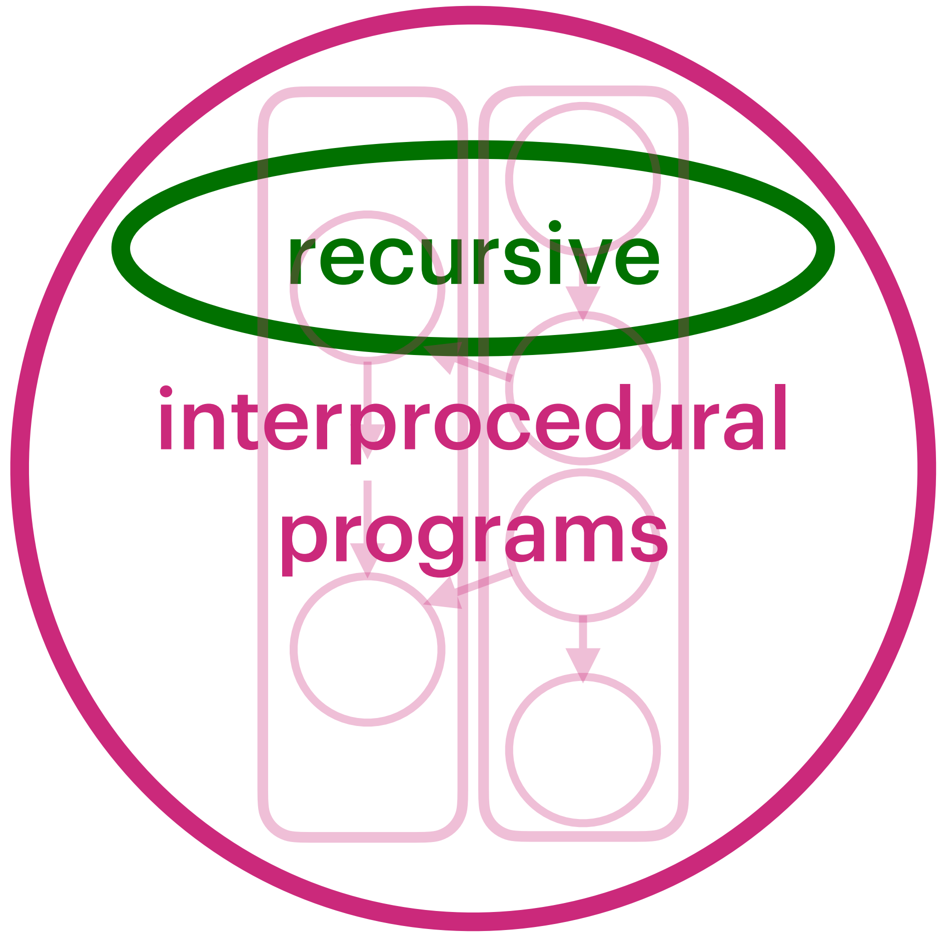
[Reps et al., POPL'95]
[Ball and Rajamani, PASTE'01]

Summary Usage

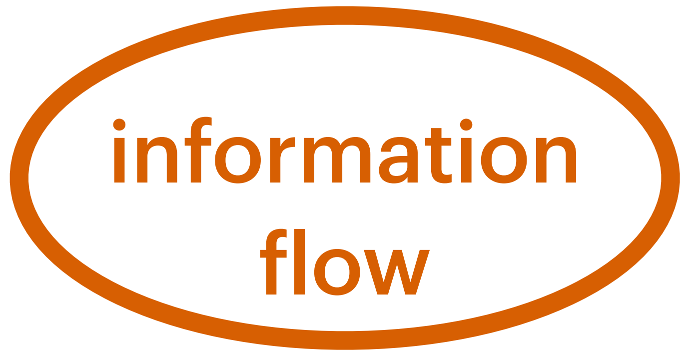
[Godefroid et al., POPL'10]

No bounded environments or EC lemmas

III. Information Flow Checking for Interprocedural Programs



+



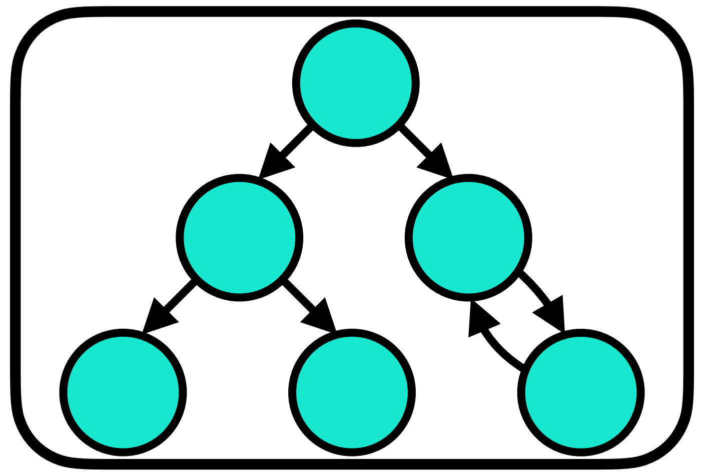
Multiple-procedure programs (may contain recursion)

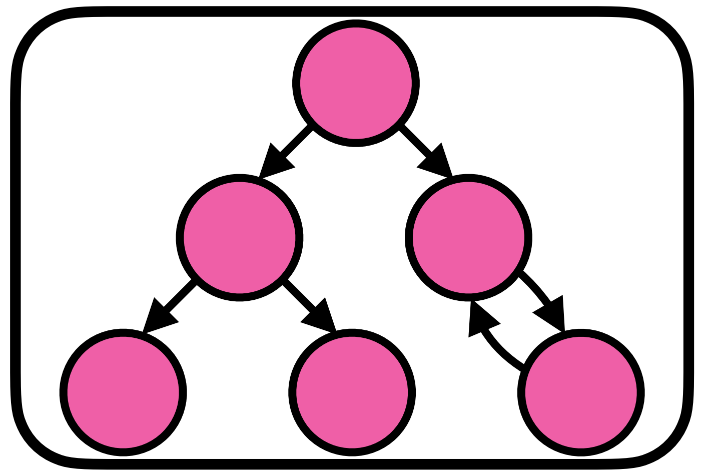
Information-flow security properties

Information-flow properties

Information-flow properties

2-safety property ***relating*** 2 copies of the same program with **equalities** on subsets of corresponding components, e.g., noninterference:





Information-flow properties

2-safety property ***relating*** 2 copies of the same program with **equalities** on subsets of corresponding components, e.g., noninterference:
"High-security inputs do not leak information to low-security outputs."

Information-flow properties

2-safety property ***relating*** 2 copies of the same program with **equalities** on subsets of corresponding components, e.g., noninterference:

"High-security inputs do not leak information to low-security outputs."

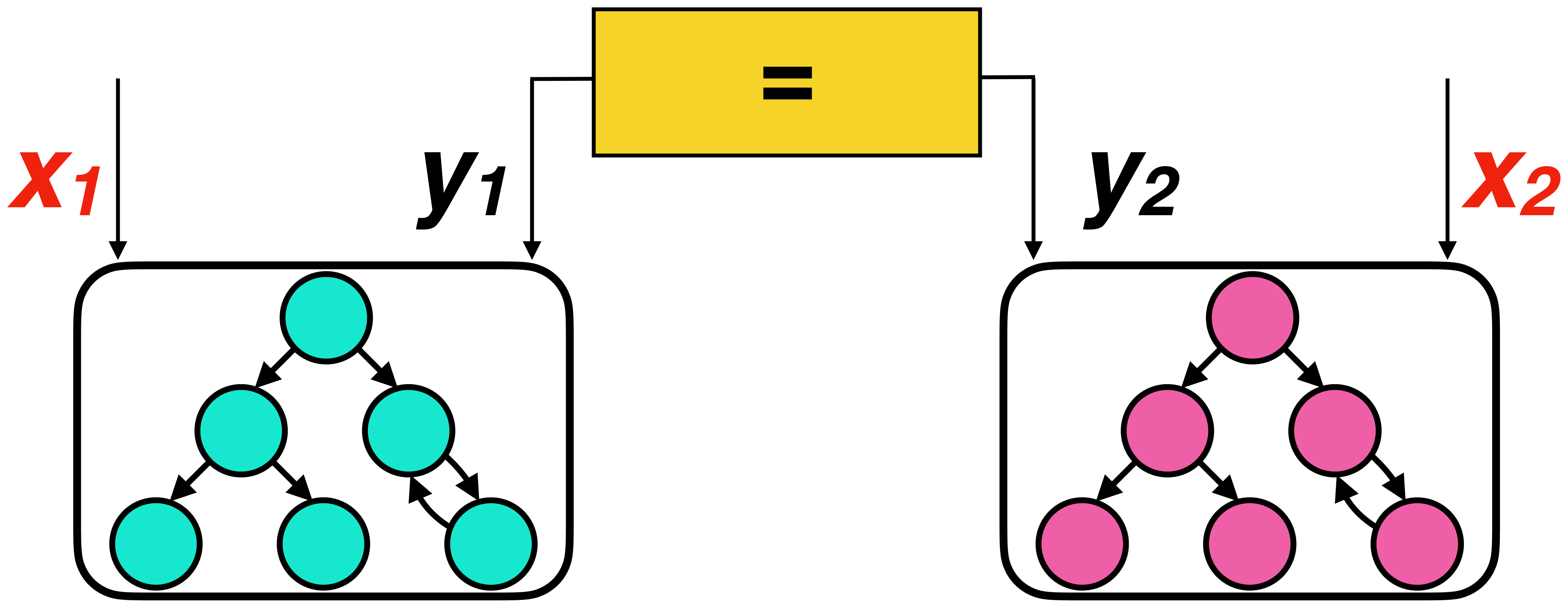
High-security inputs shown in red

Information-flow properties

2-safety property ***relating*** 2 copies of the same program with **equalities** on subsets of corresponding components, e.g., noninterference:

"High-security inputs do not leak information to low-security outputs."

High-security inputs shown in red

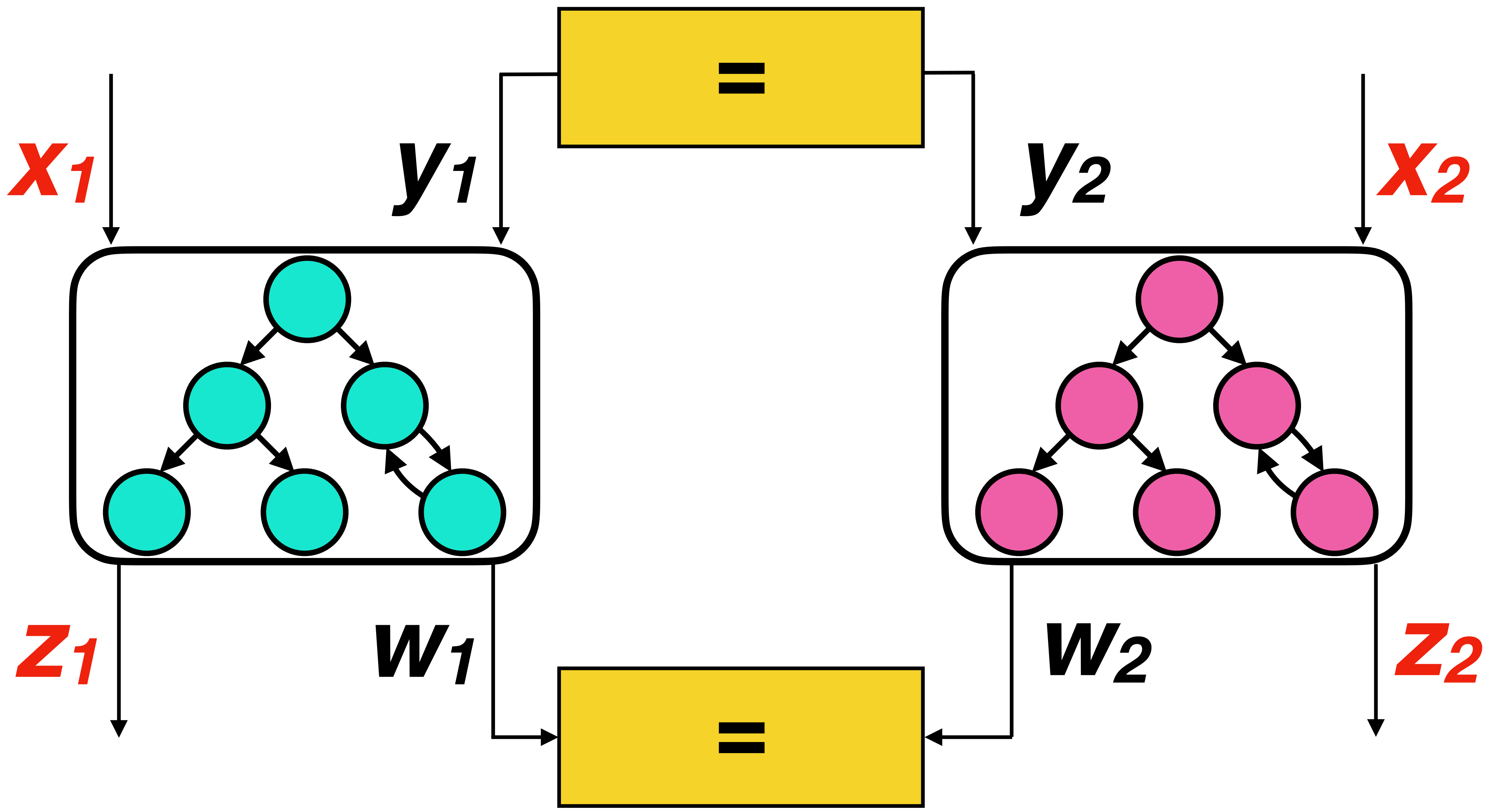


Information-flow properties

2-safety property ***relating*** 2 copies of the same program with **equalities** on subsets of corresponding components, e.g., noninterference:

"High-security inputs do not leak information to low-security outputs."

High-security inputs shown in red

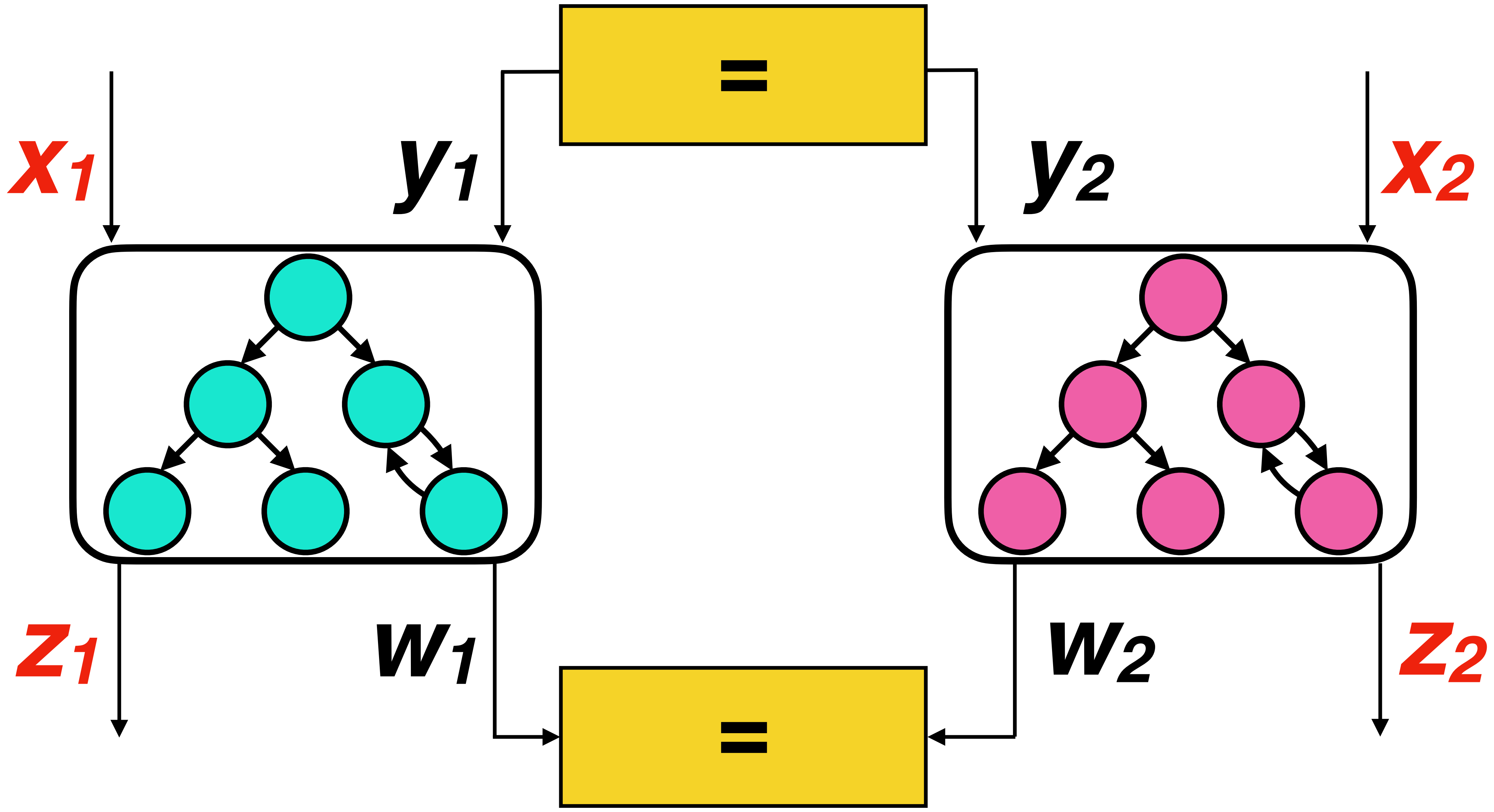


Product Programs

Can turn 

into 

by constructing a product program



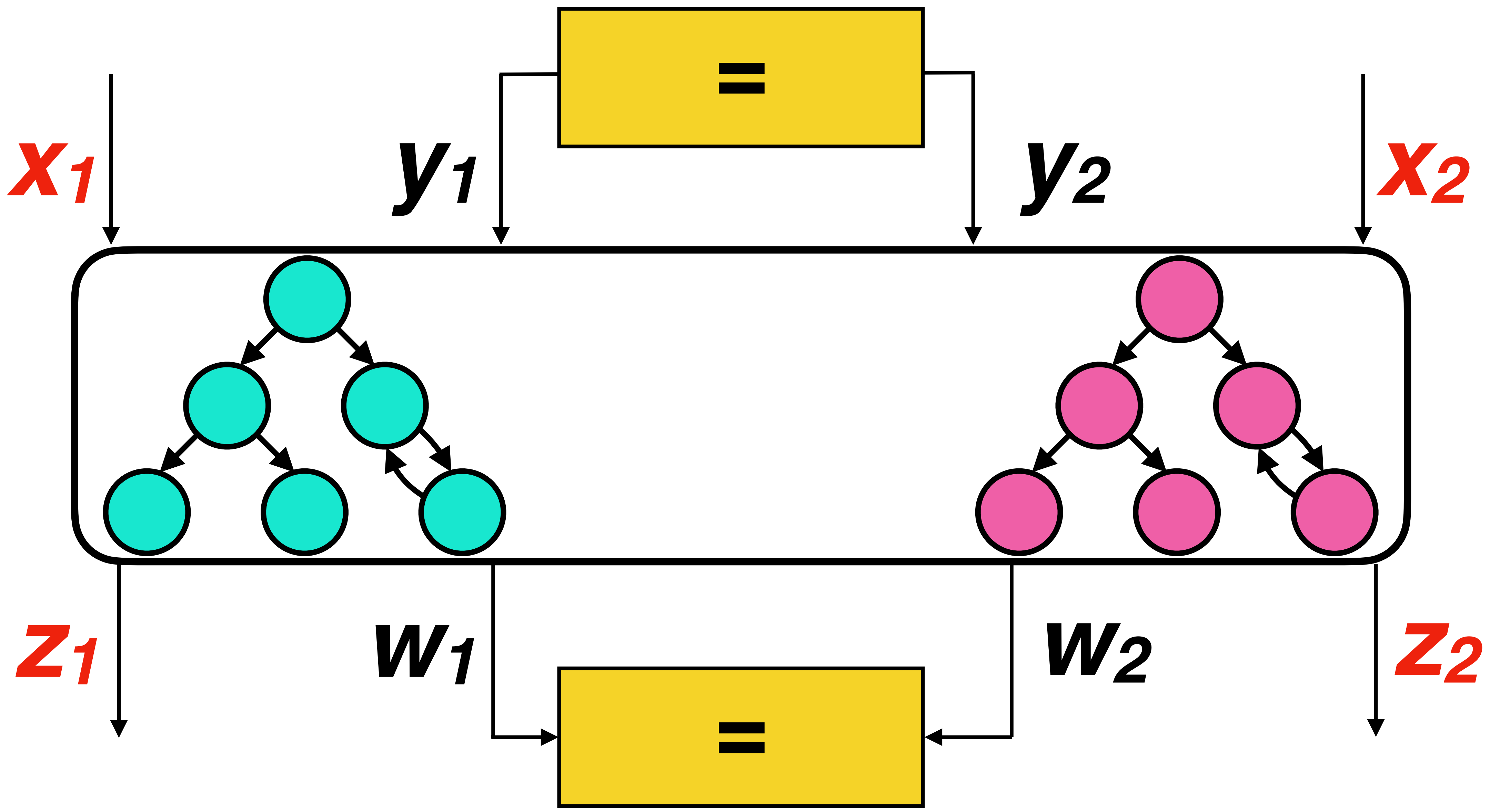
Secure information flow by self-composition, Barthe et al., CSFW'04
Relational verification using product programs, Barthe et al., FM'11

Product Programs

Can turn 

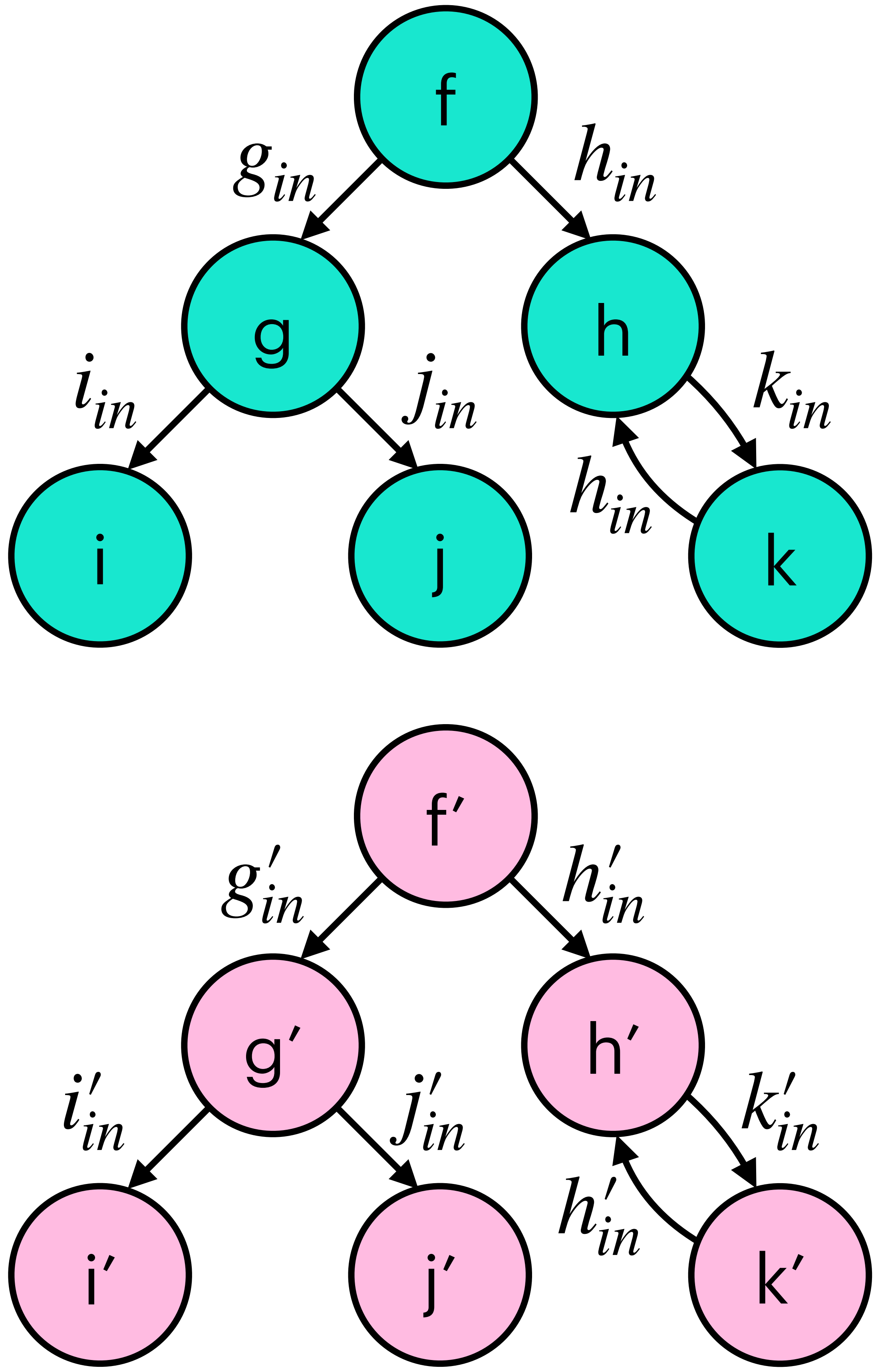
into 

by constructing a product program



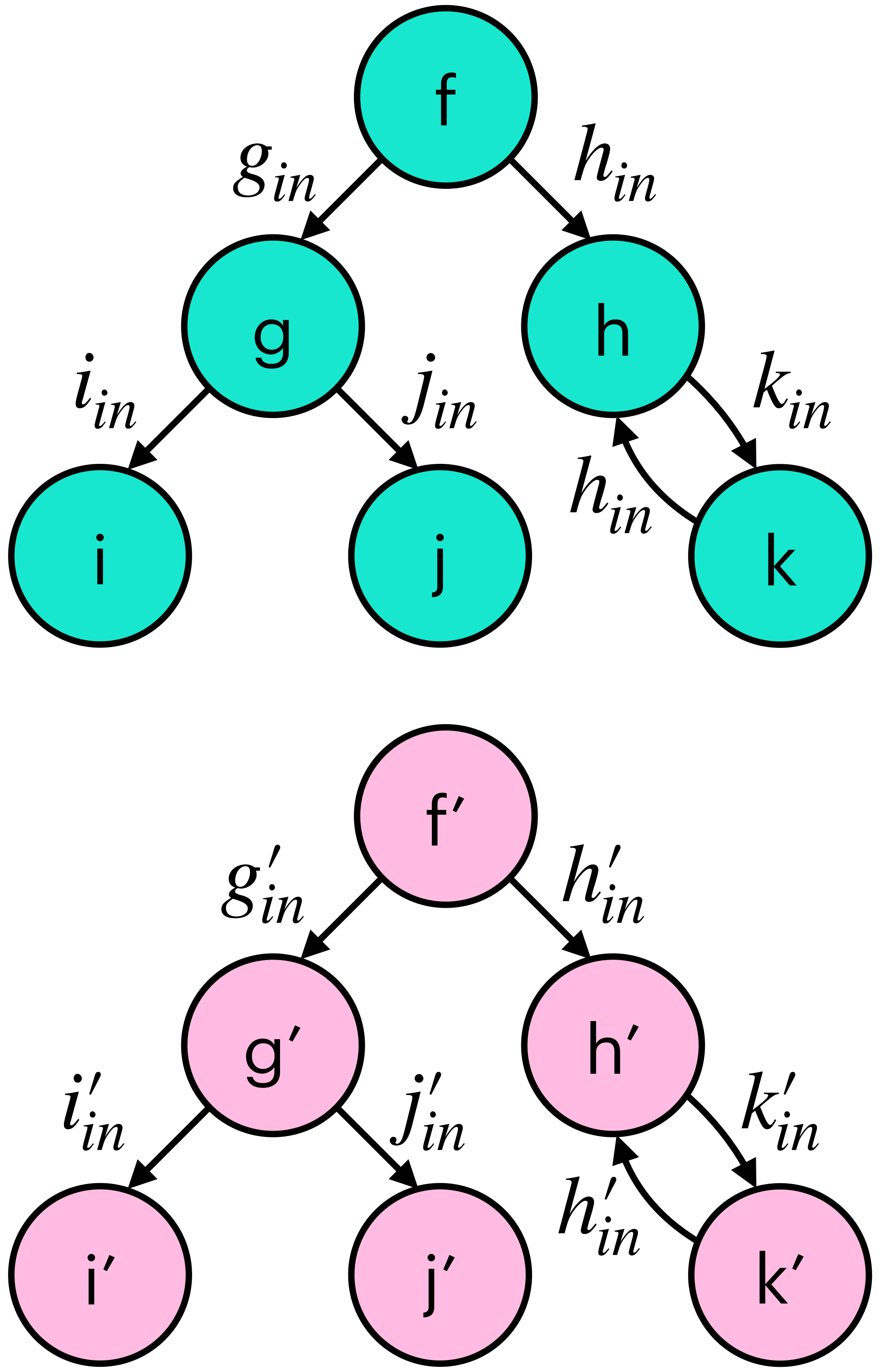
Secure information flow by self-composition, Barthe et al., CSFW'04
Relational verification using product programs, Barthe et al., FM'11

Modular Product Programs



Modular product programs, Eilers et al., ESOP'18

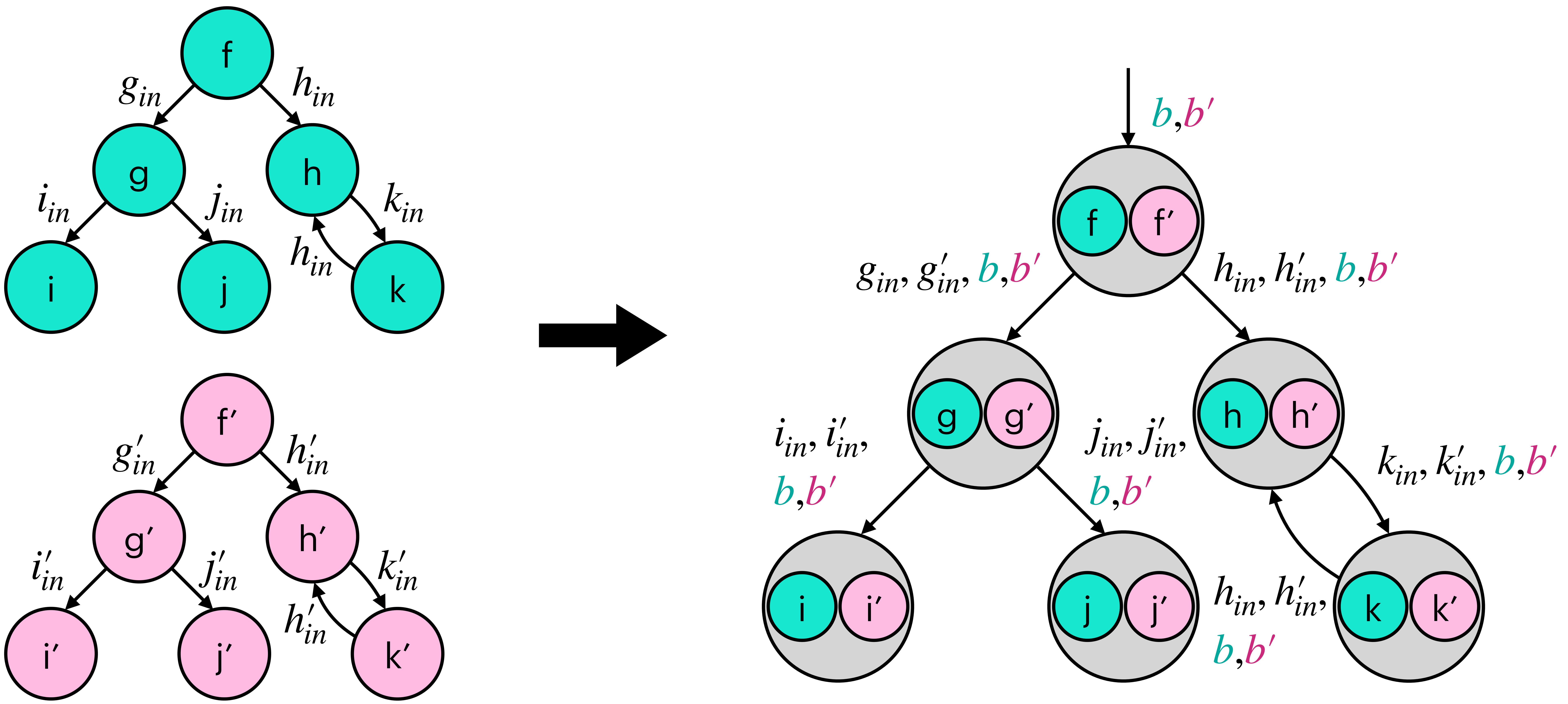
Modular Product Programs



Labels denote input variables

Modular product programs, Eilers et al., ESOP'18

Modular Product Programs

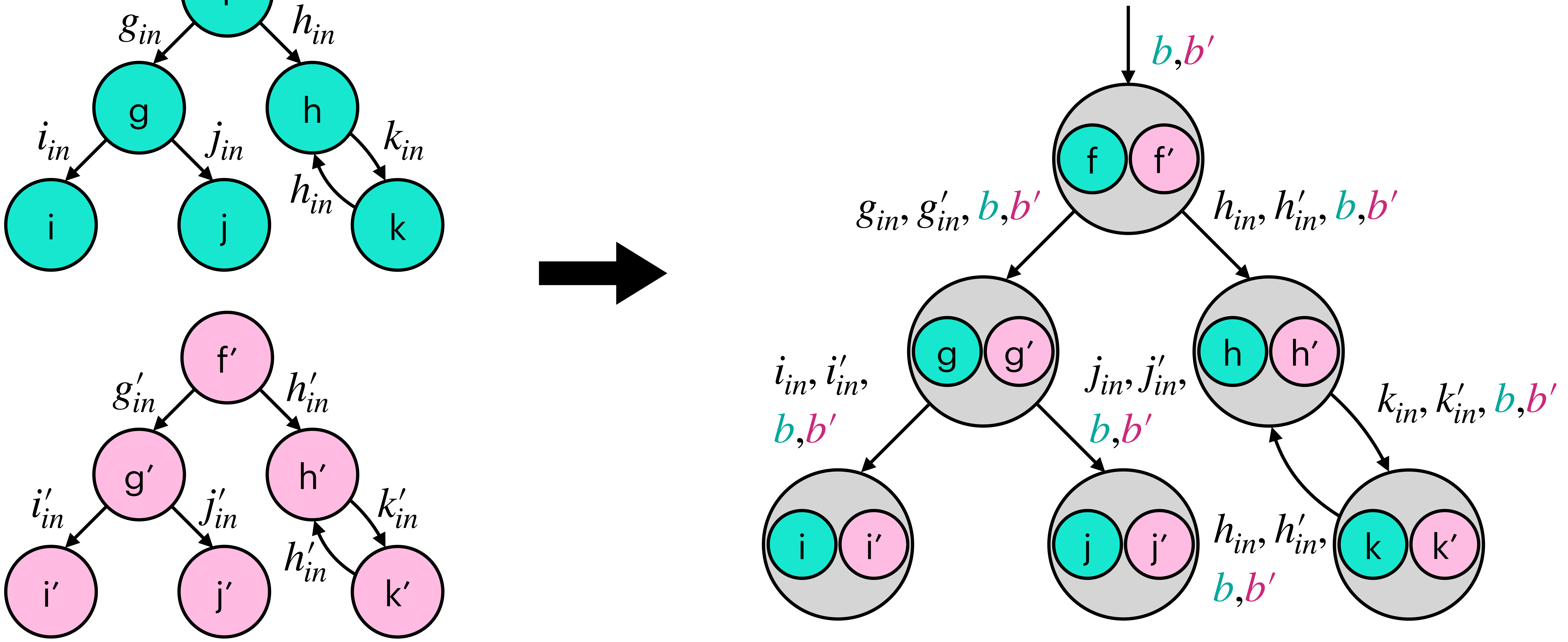


Labels denote input variables

Modular product programs, Eilers et al., ESOP'18

Modular Product Programs

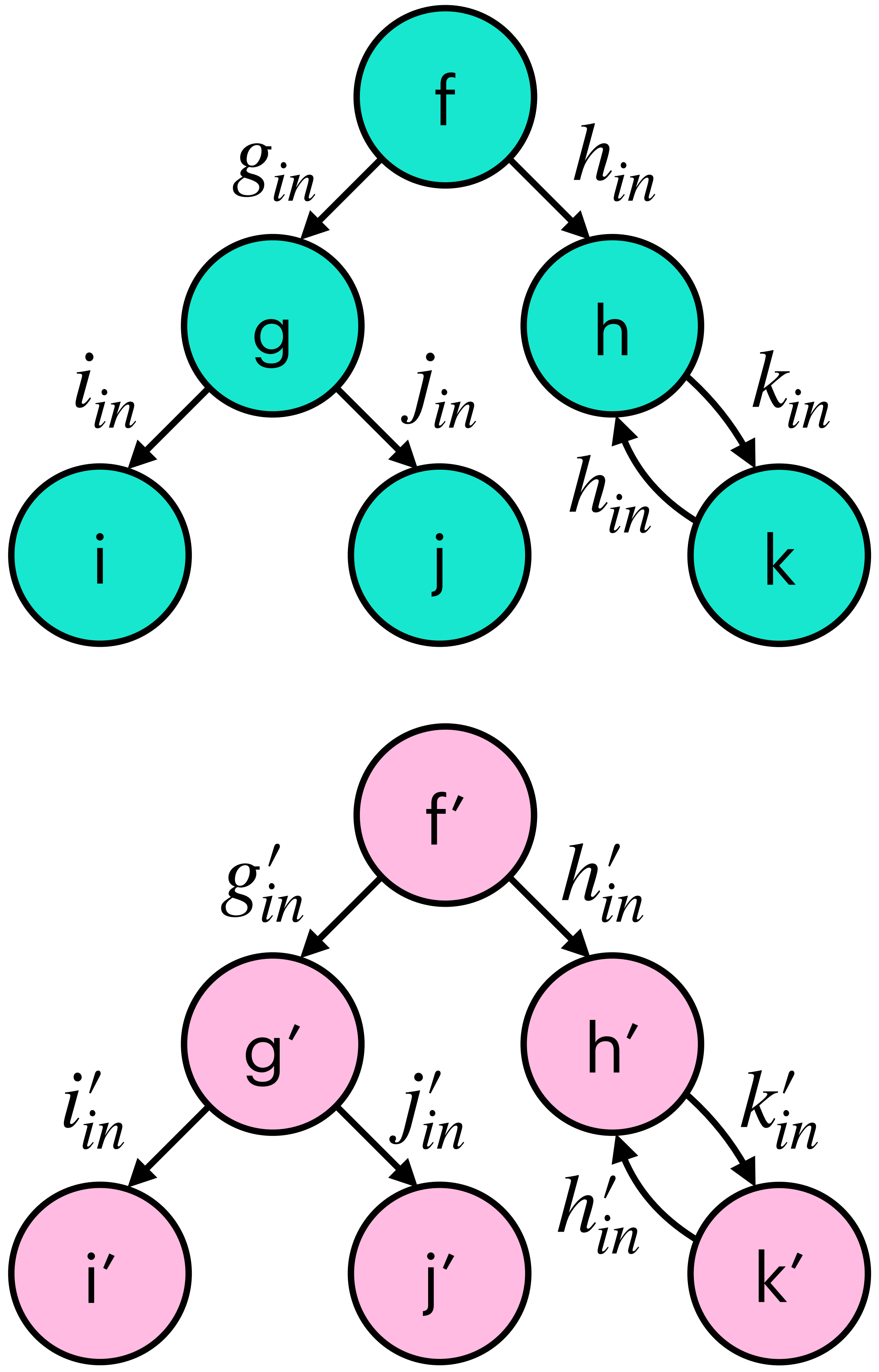
Activation variables b, b' specify if copy is active



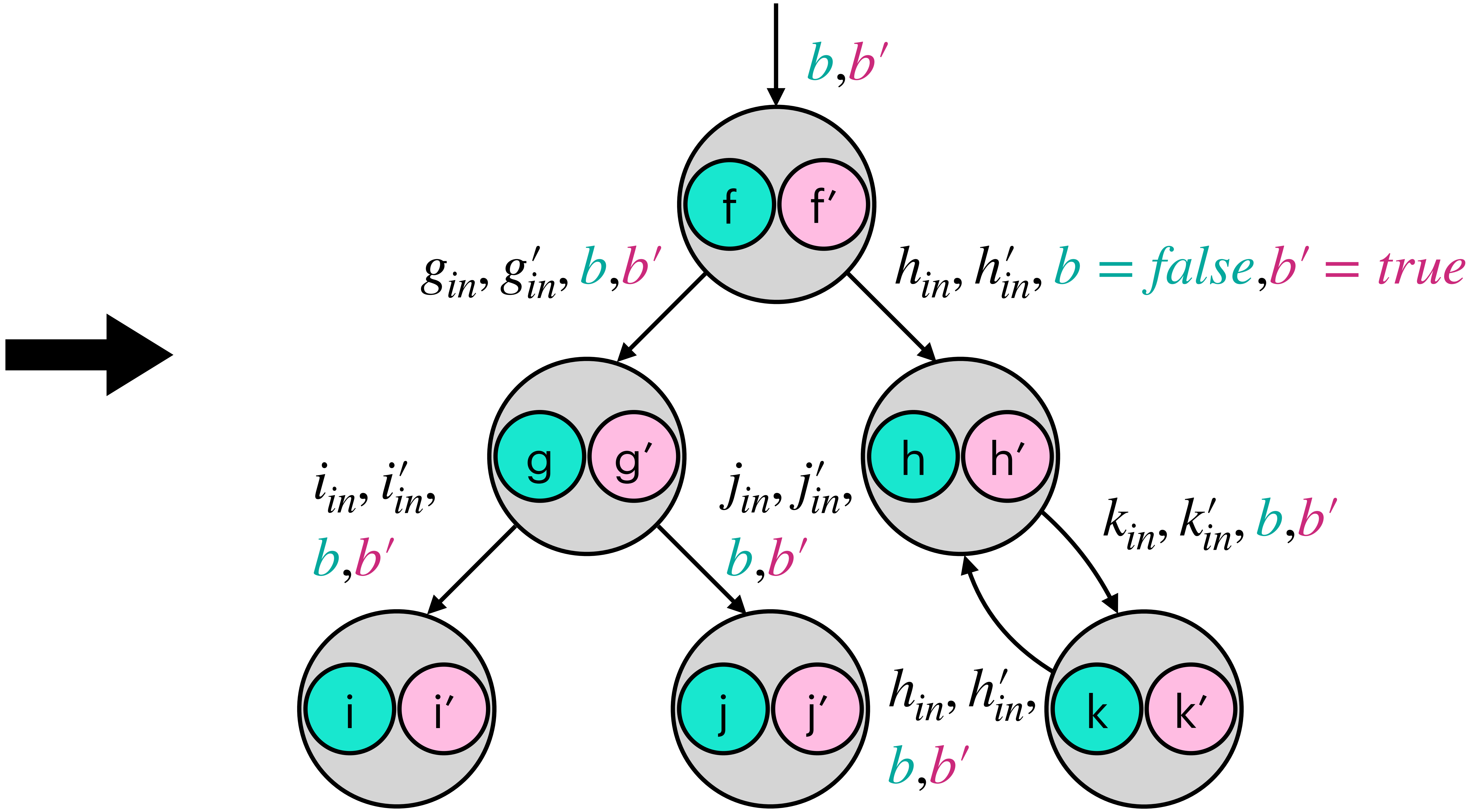
Labels denote input variables

Modular product programs, Eilers et al., ESOP’18

Modular Product Programs

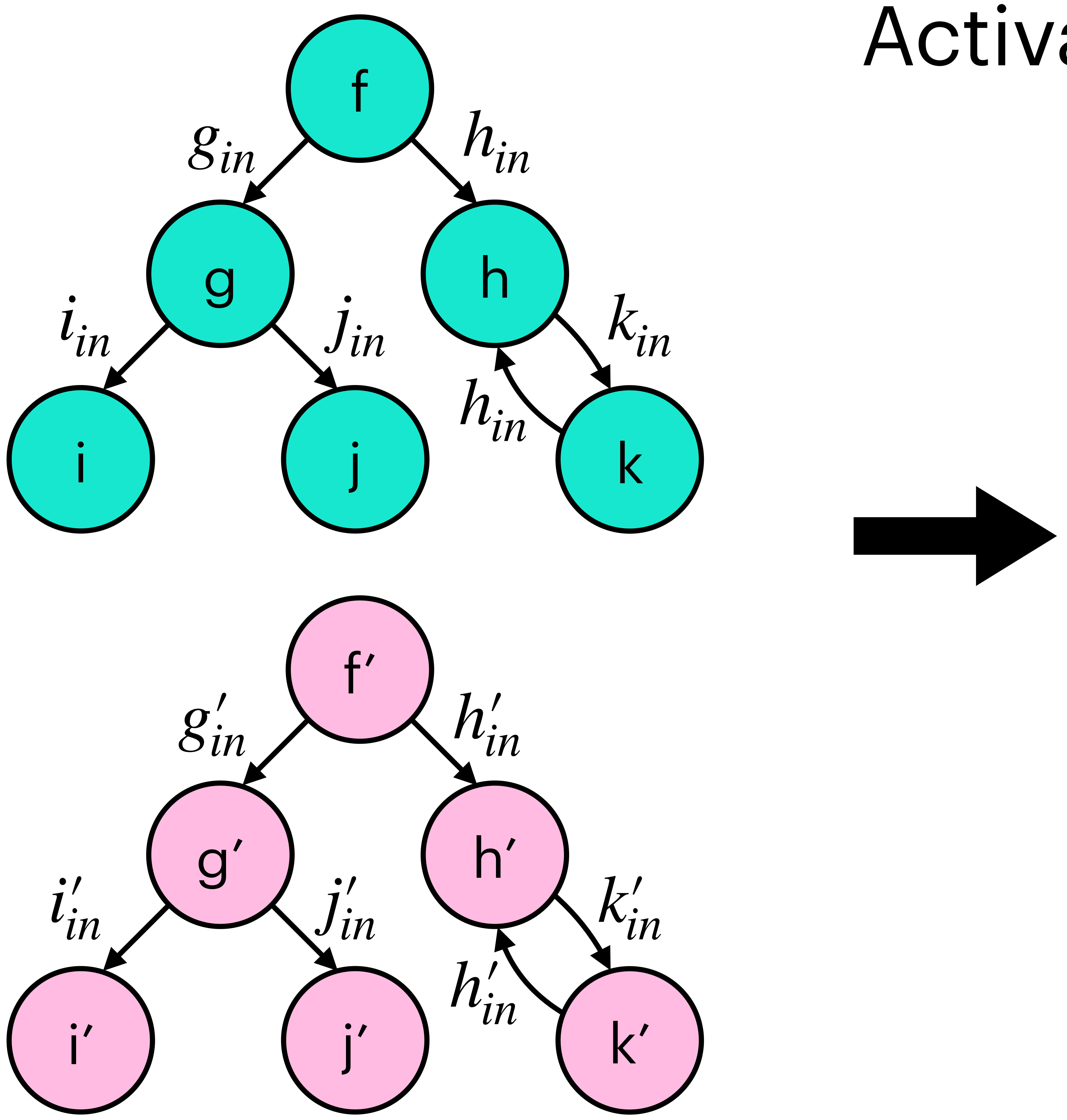


Activation variables b, b' specify if copy is active

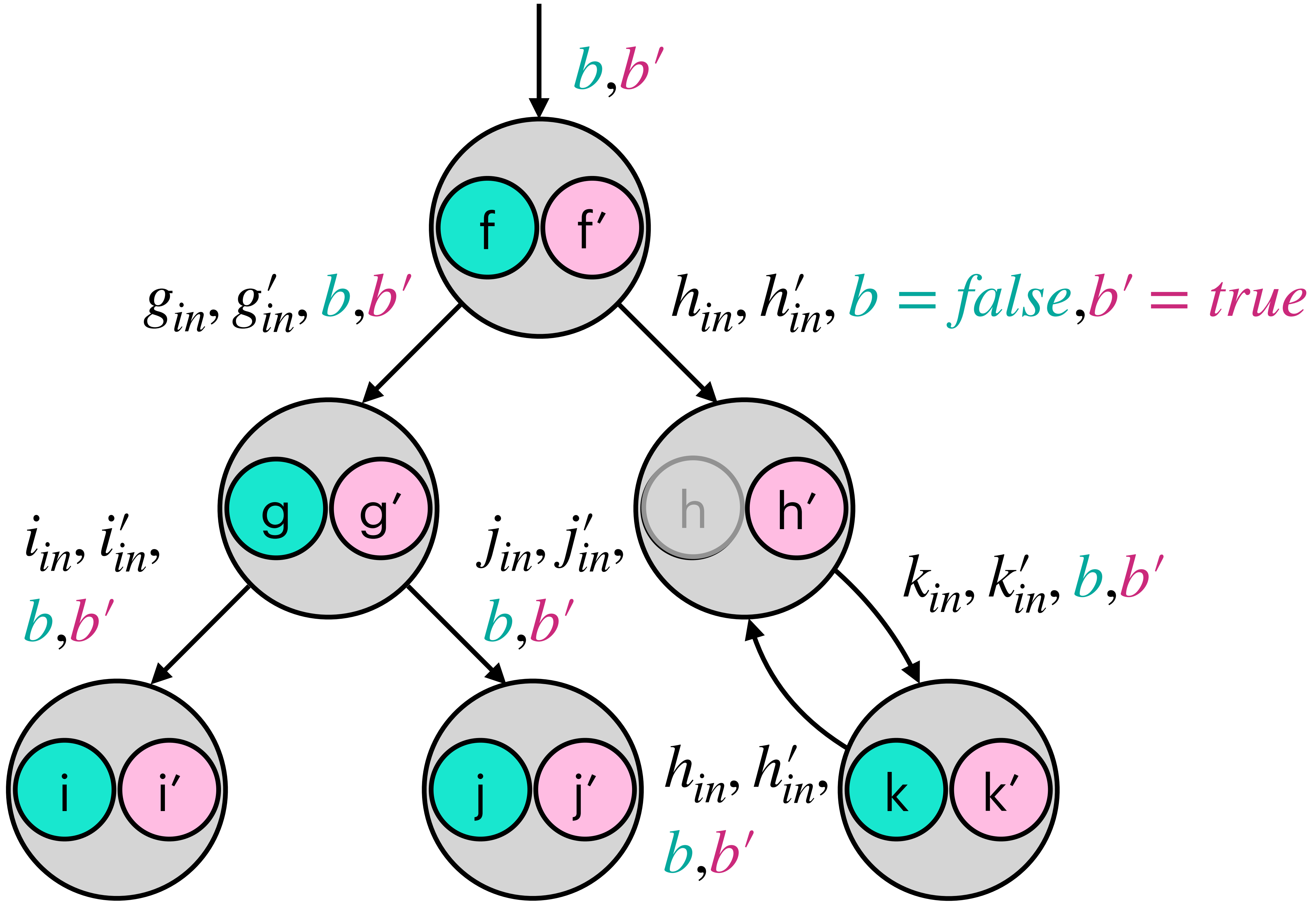


Modular product programs, Eilers et al., ESOP’18

Modular Product Programs

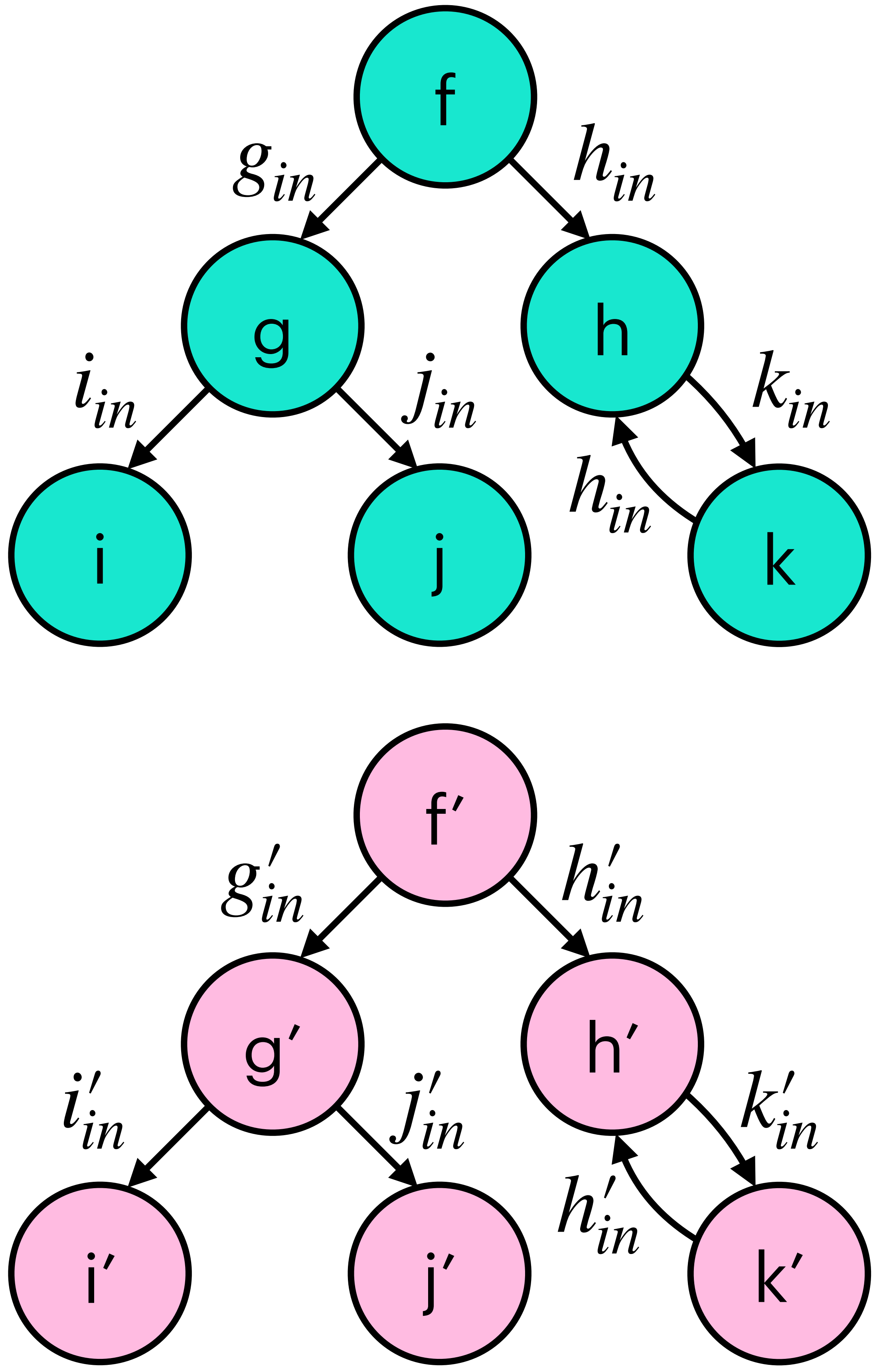


Activation variables b, b' specify if copy is active

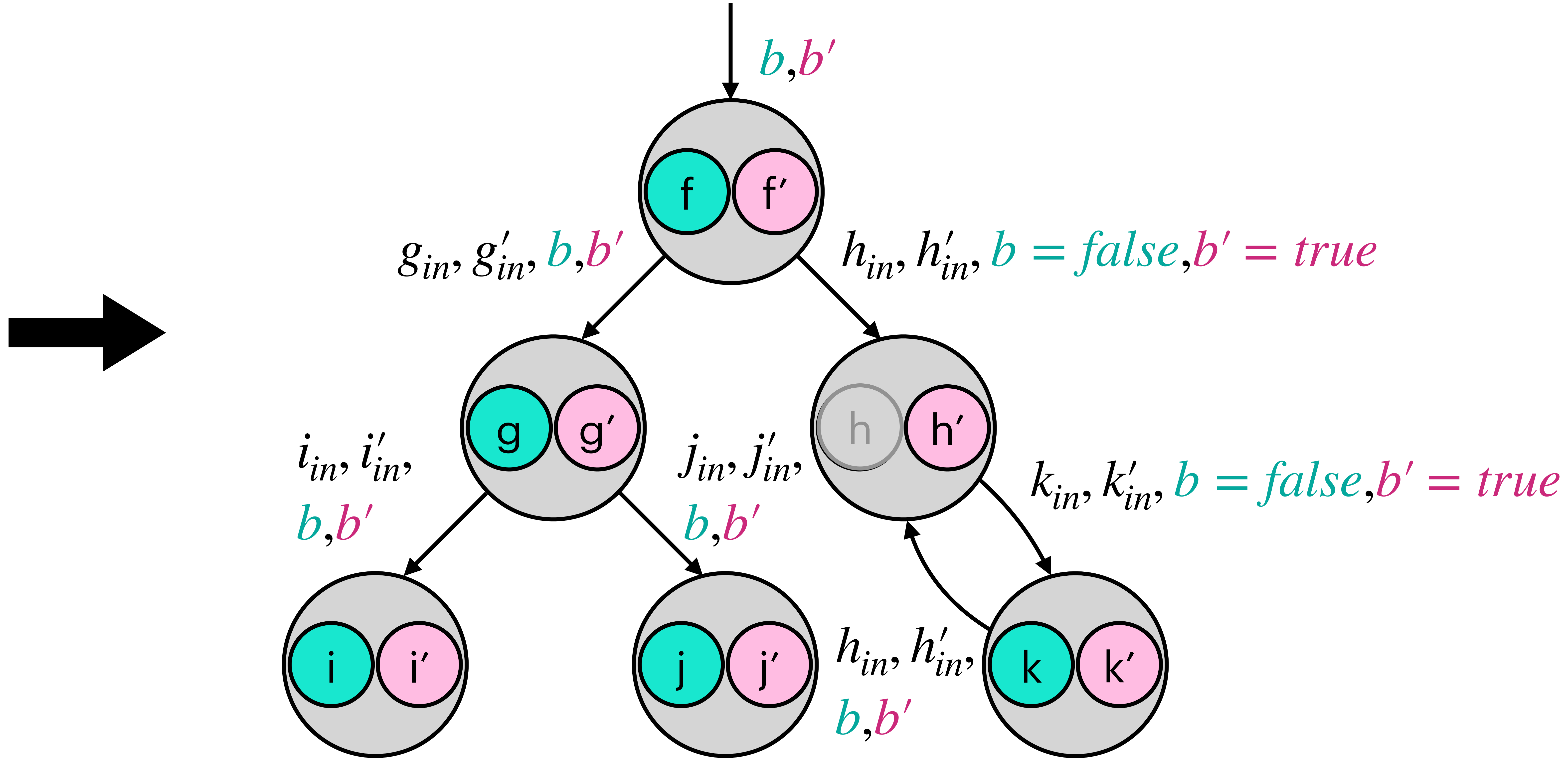


Modular product programs, Eilers et al., ESOP'18

Modular Product Programs

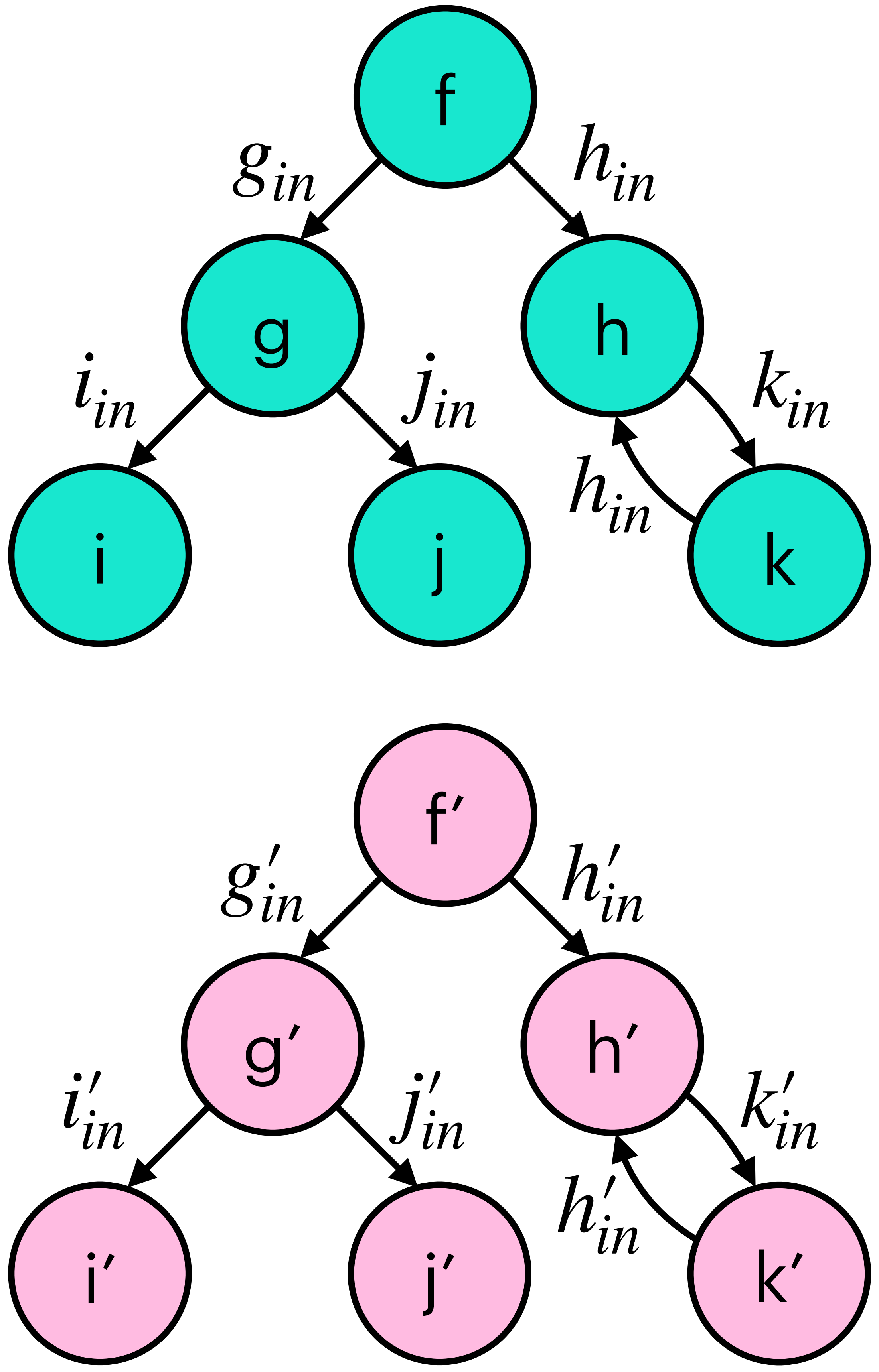


Activation variables b, b' specify if copy is active

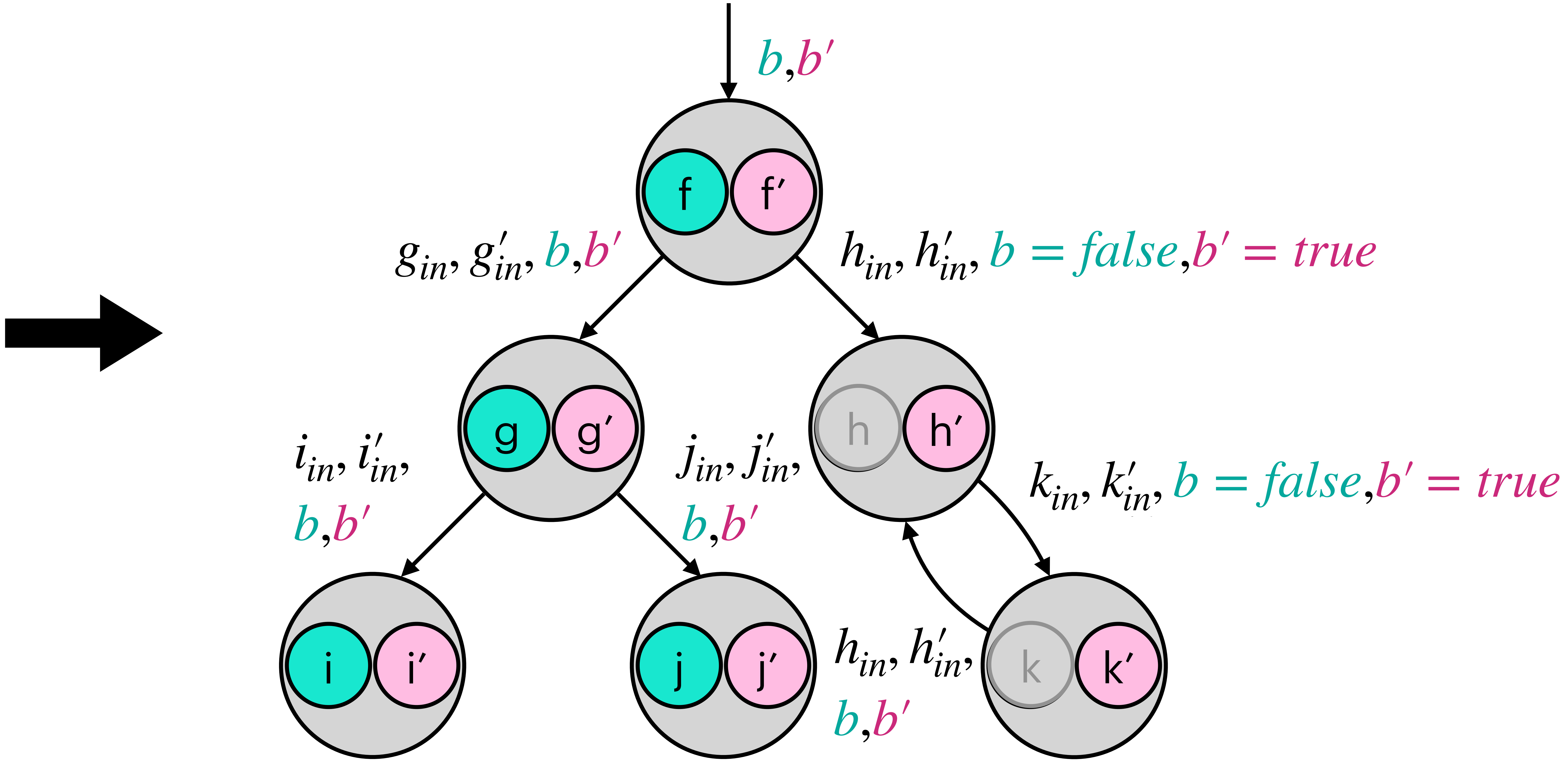


Modular product programs, Eilers et al., ESOP’18

Modular Product Programs

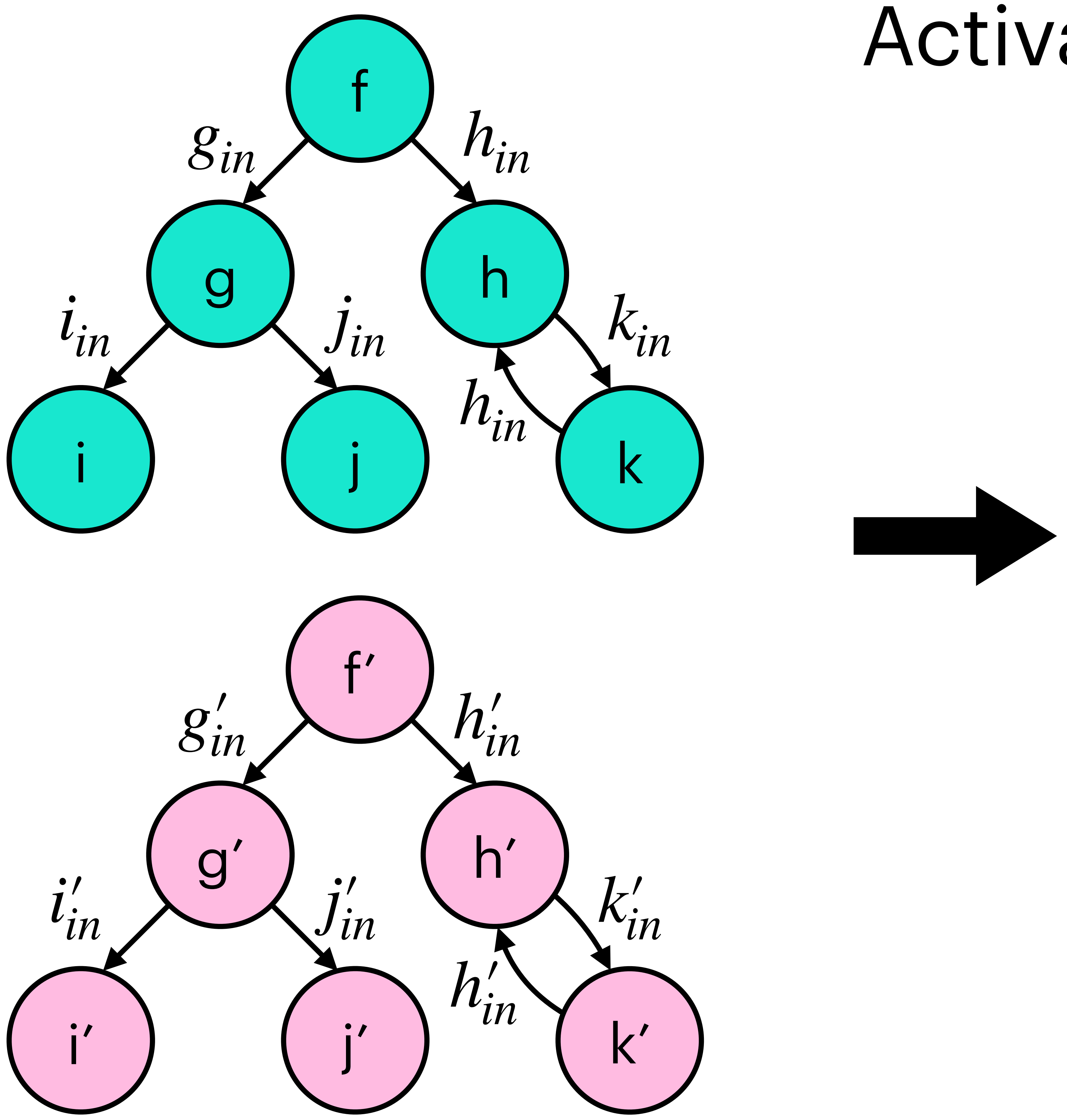


Activation variables b, b' specify if copy is active

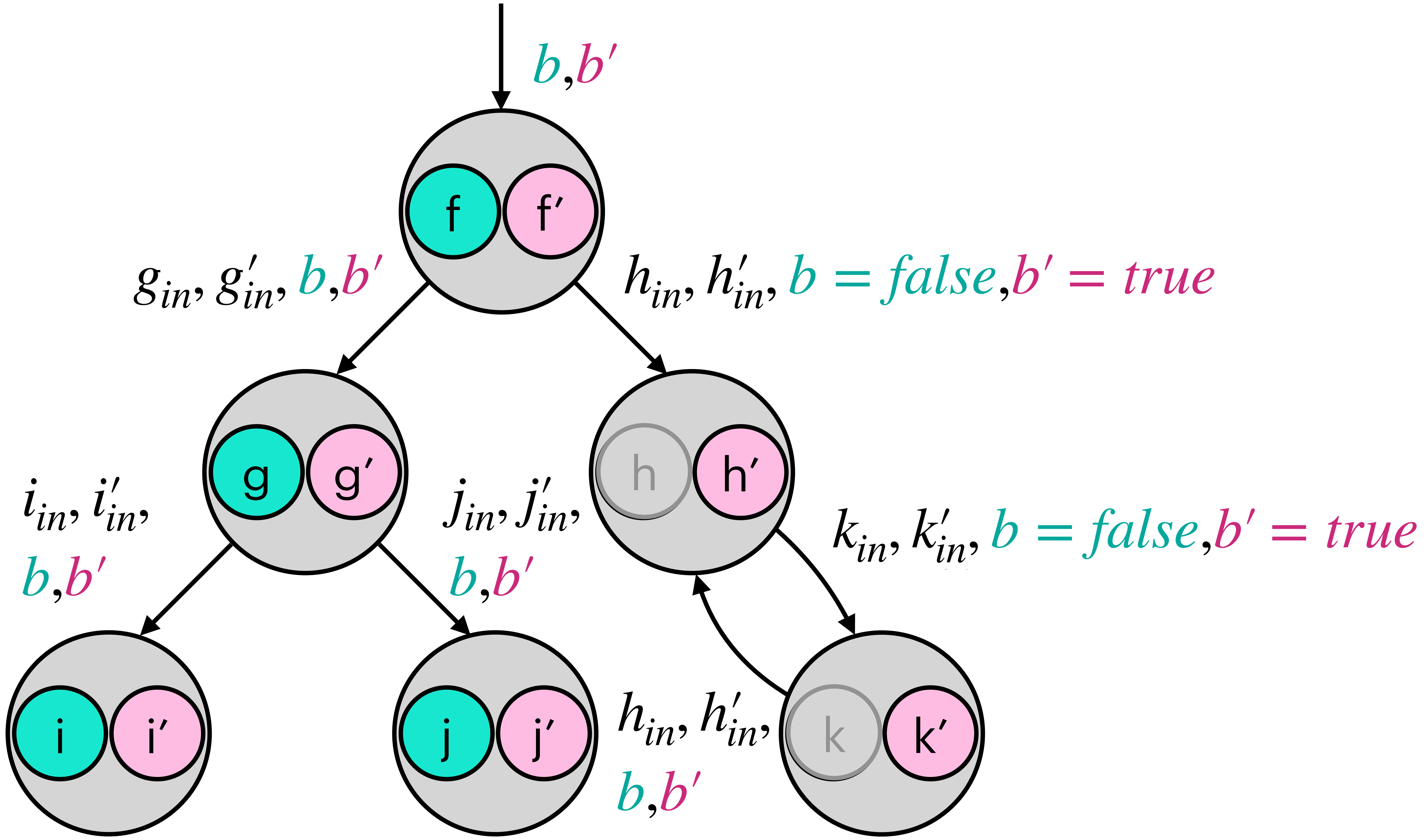


Modular product programs, Eilers et al., ESOP’18

Modular Product Programs



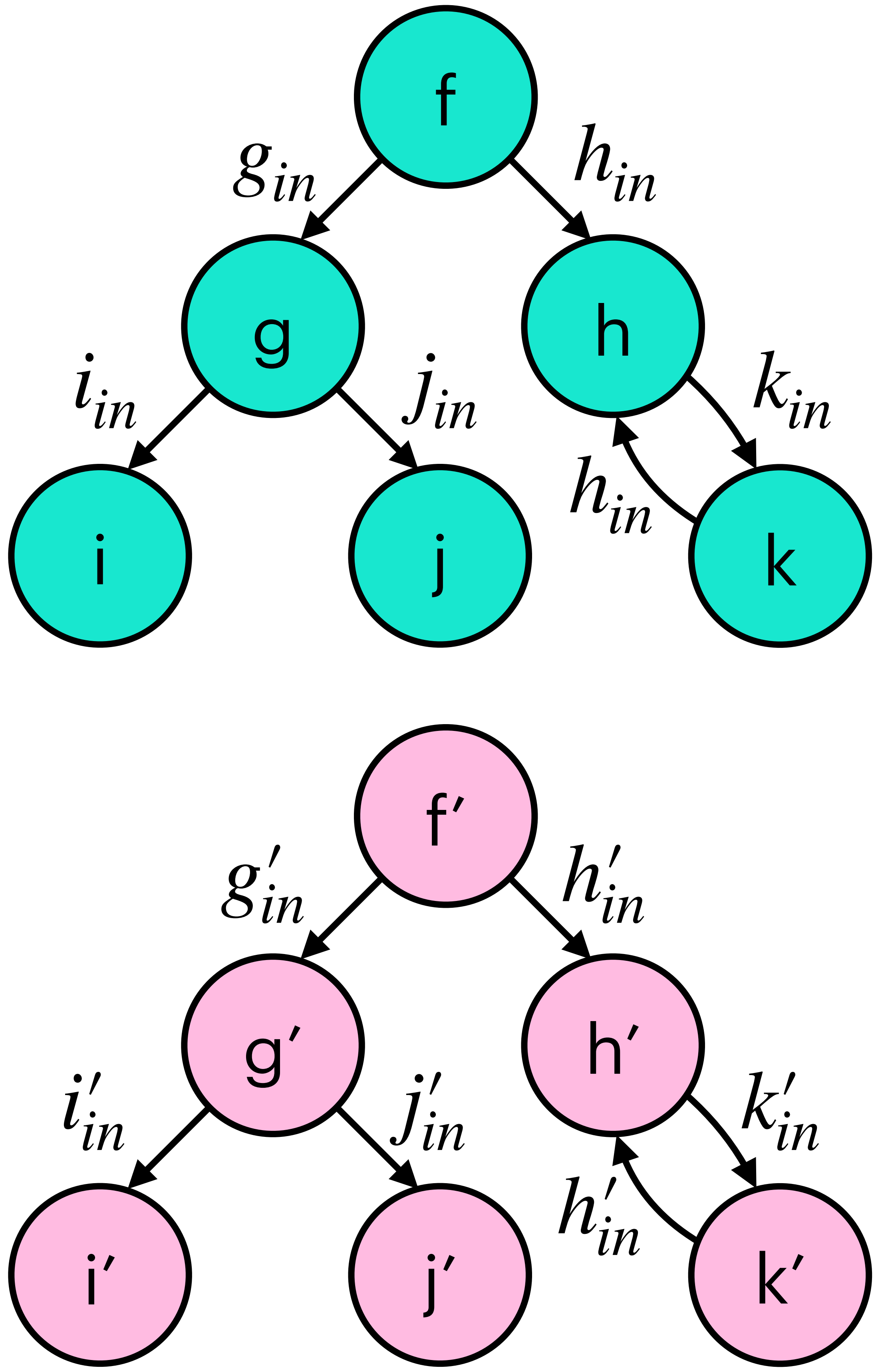
Activation variables b, b' specify if copy is active



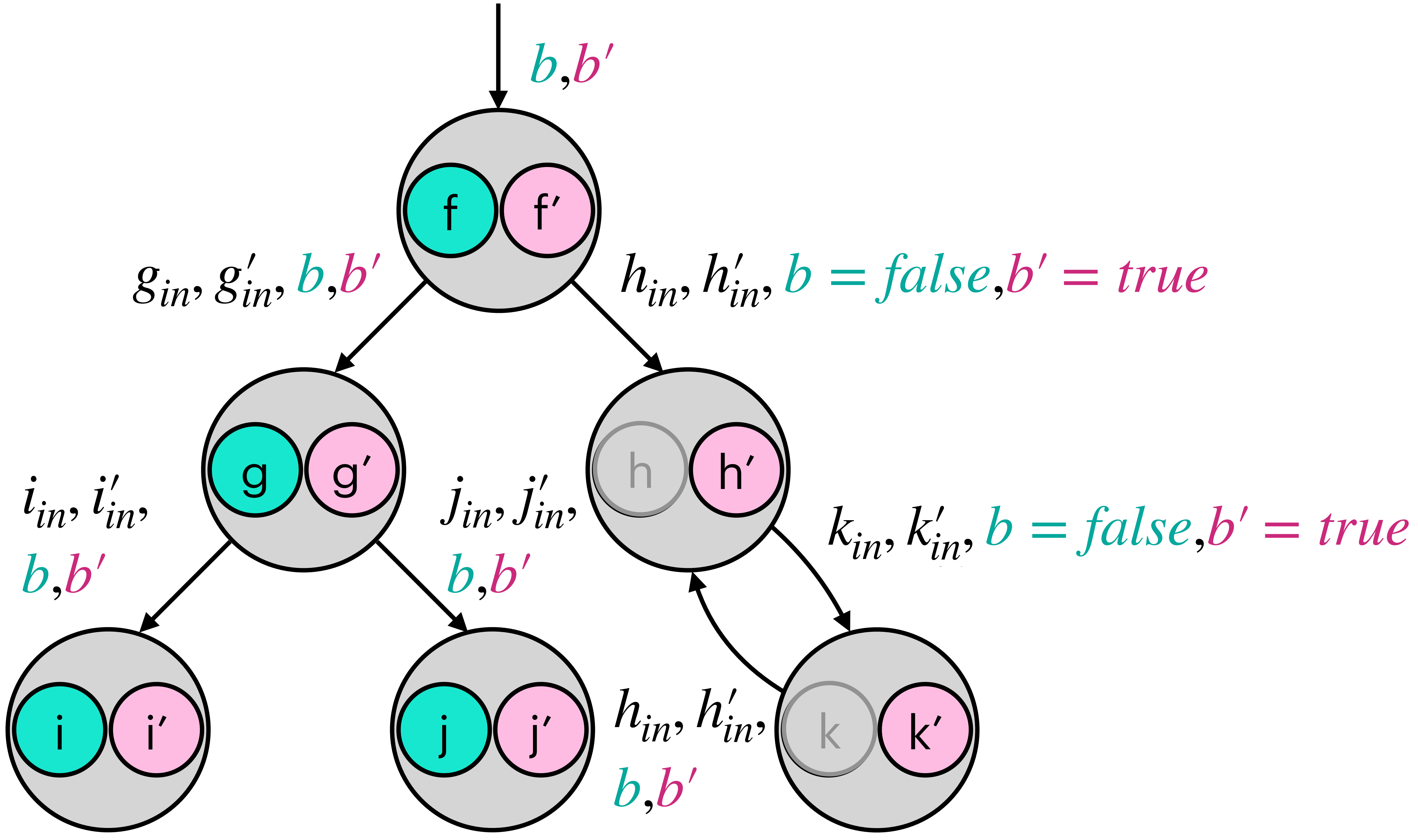
Required user-provided annotations (which variables are high-/low-security?)

Modular product programs, Eilers et al., ESOP'18

Modular Product Programs



Activation variables b, b' specify if copy is active

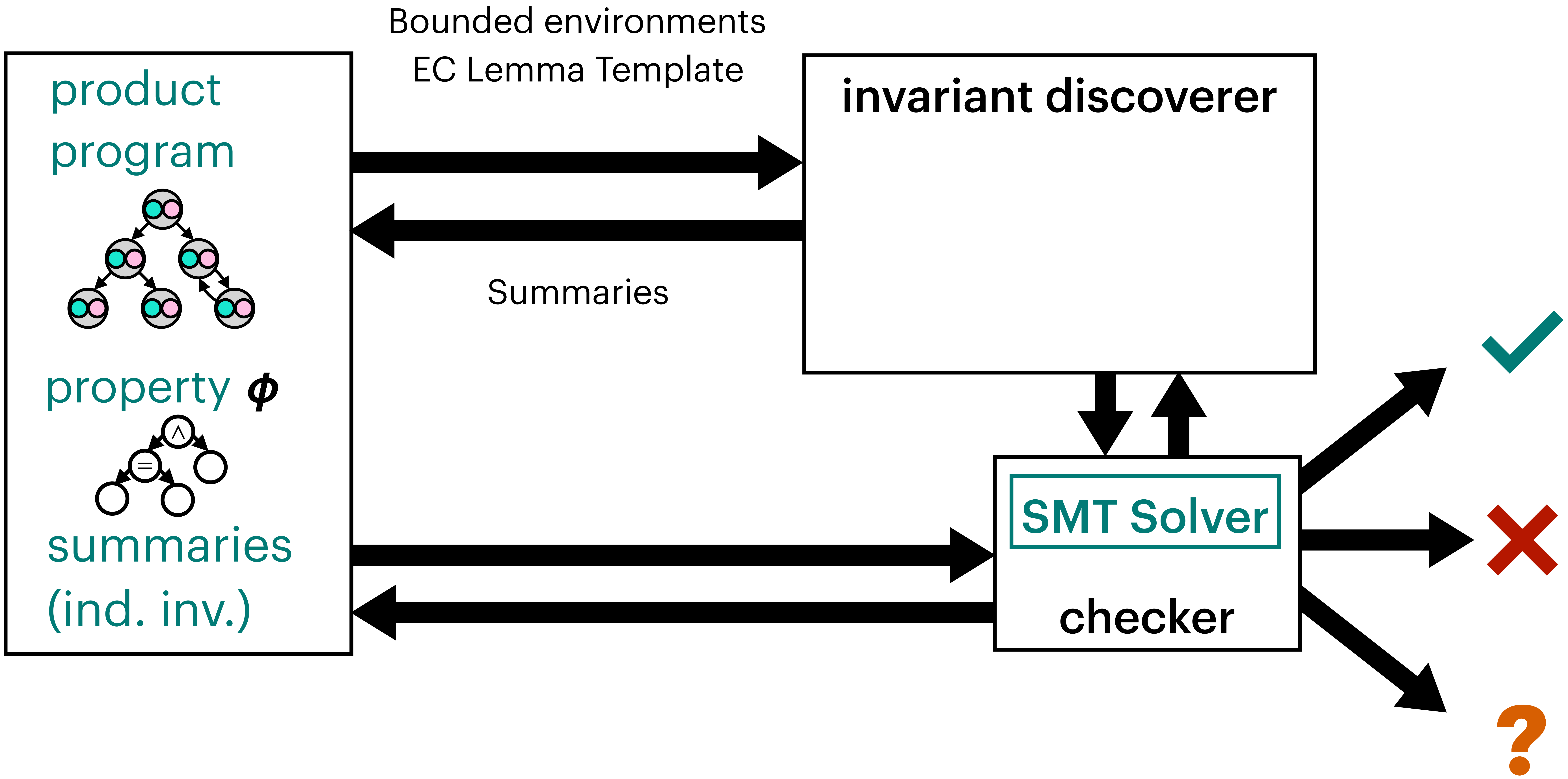


Required user-provided annotations (which variables are high-/low-security?)

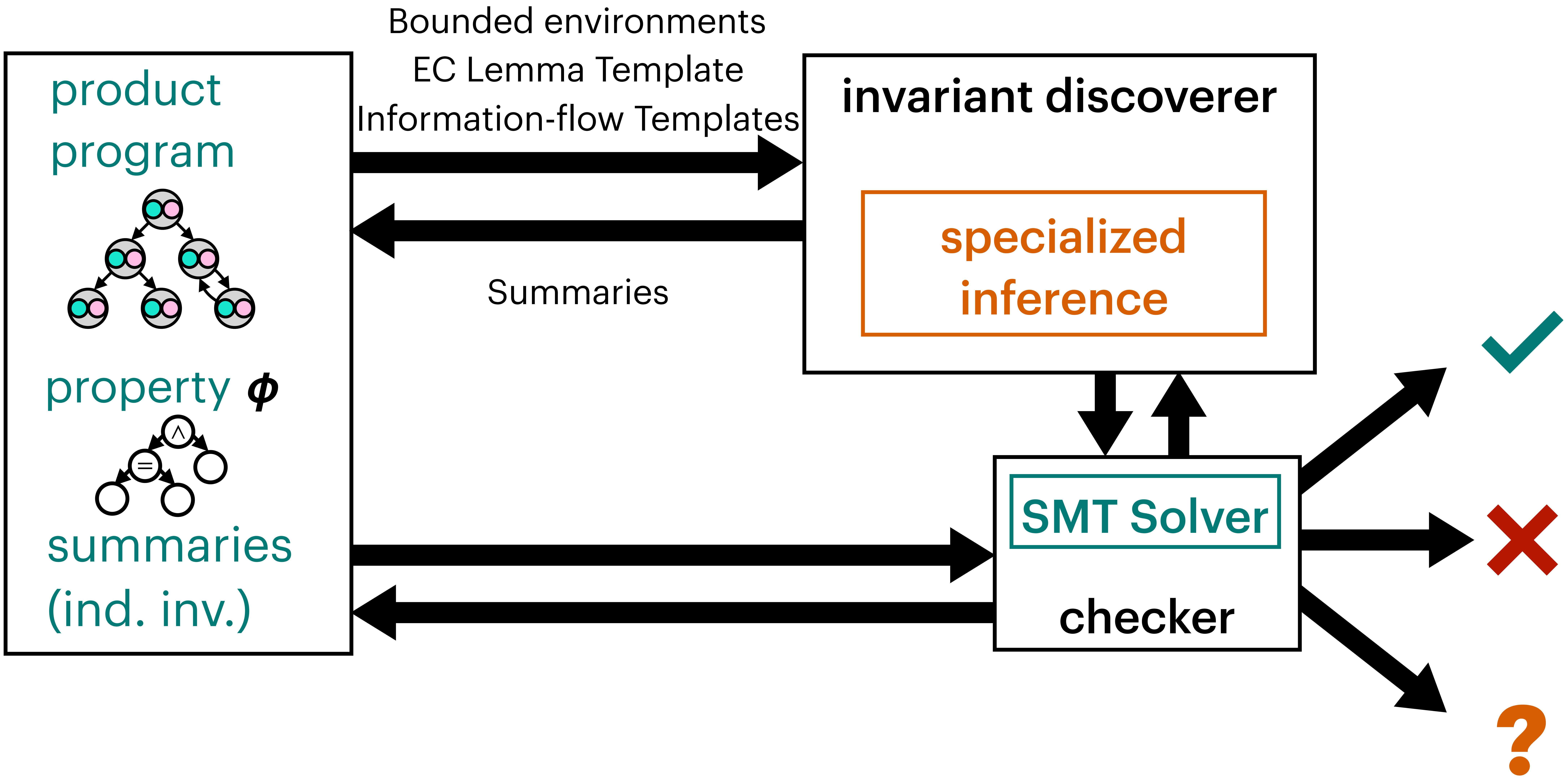
Can we infer these invariants?

Modular product programs, Eilers et al., ESOP’18

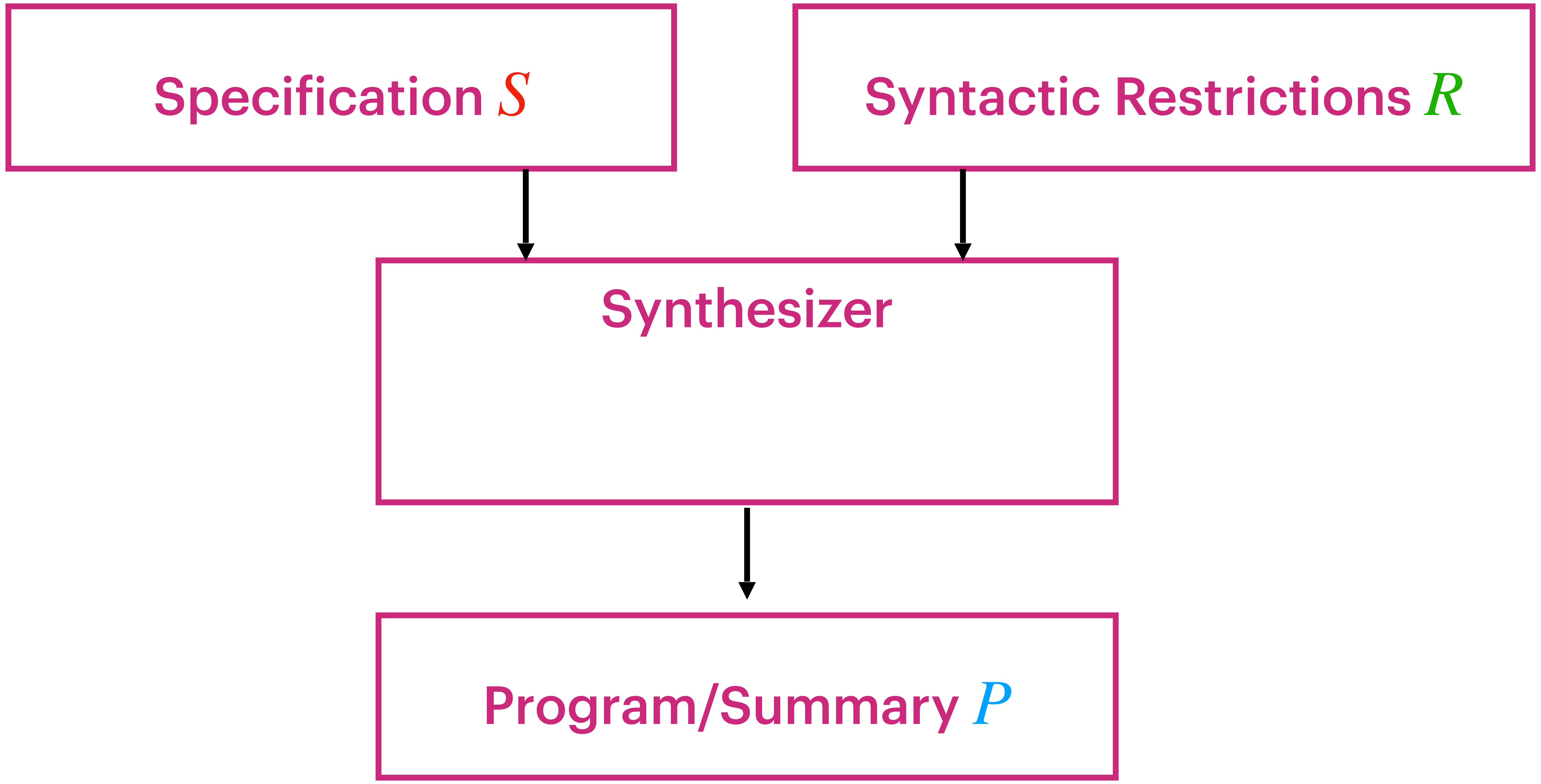
Adapting Interprocedural Program Verification



Adapting Interprocedural Program Verification



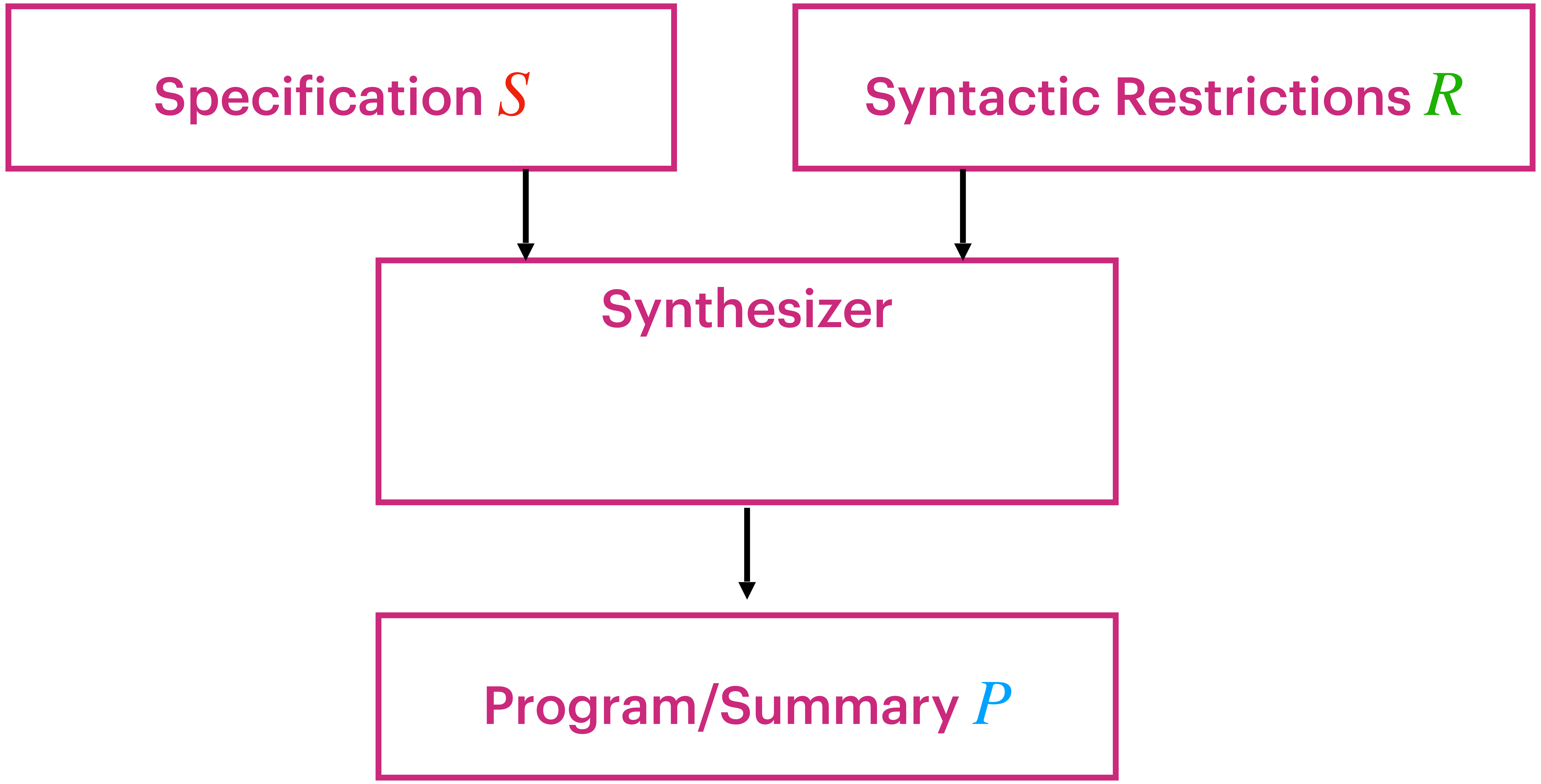
Syntax-Guided Synthesis (SyGuS)



Syntax-Guided Synthesis, Alur et al., FMCAD'13

Syntax-Guided Synthesis (SyGuS)

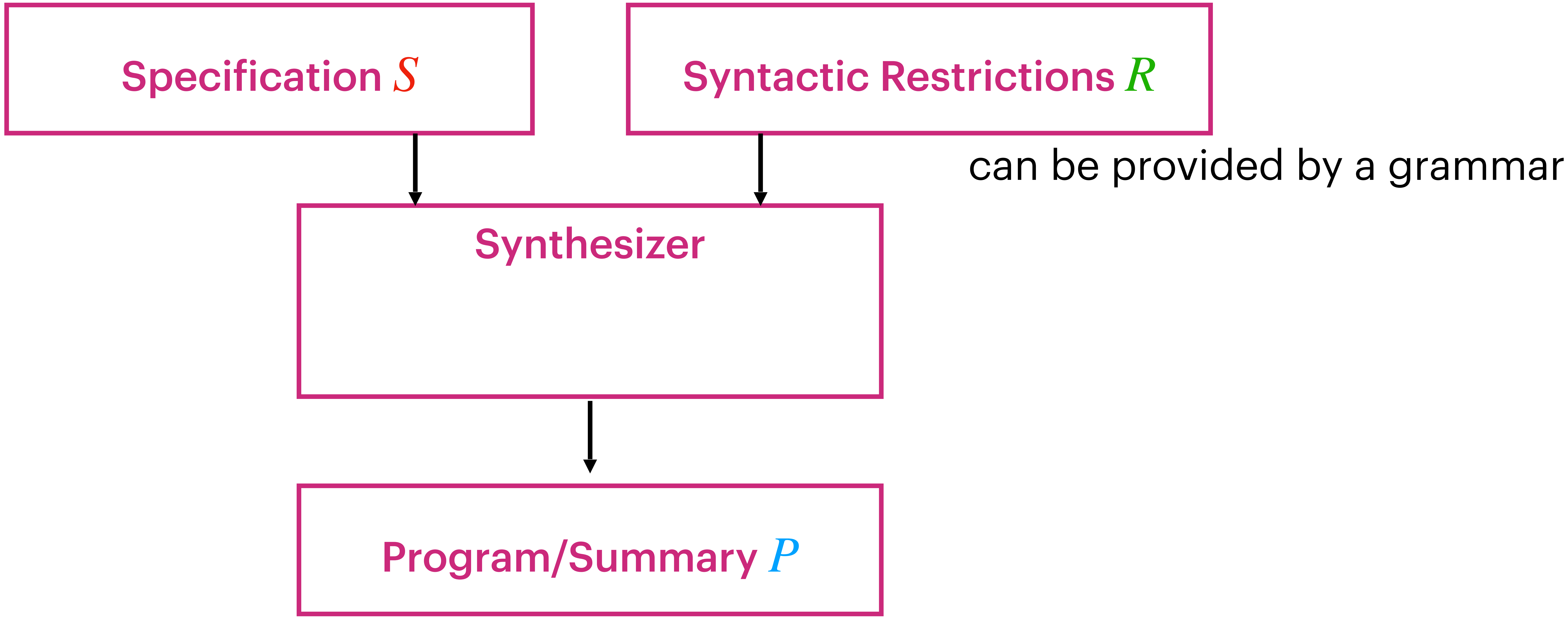
$$\exists P \in [\![R]\!] . \forall i . P(i) \vDash S(i)$$



Syntax-Guided Synthesis, Alur et al., FMCAD'13

Syntax-Guided Synthesis (SyGuS)

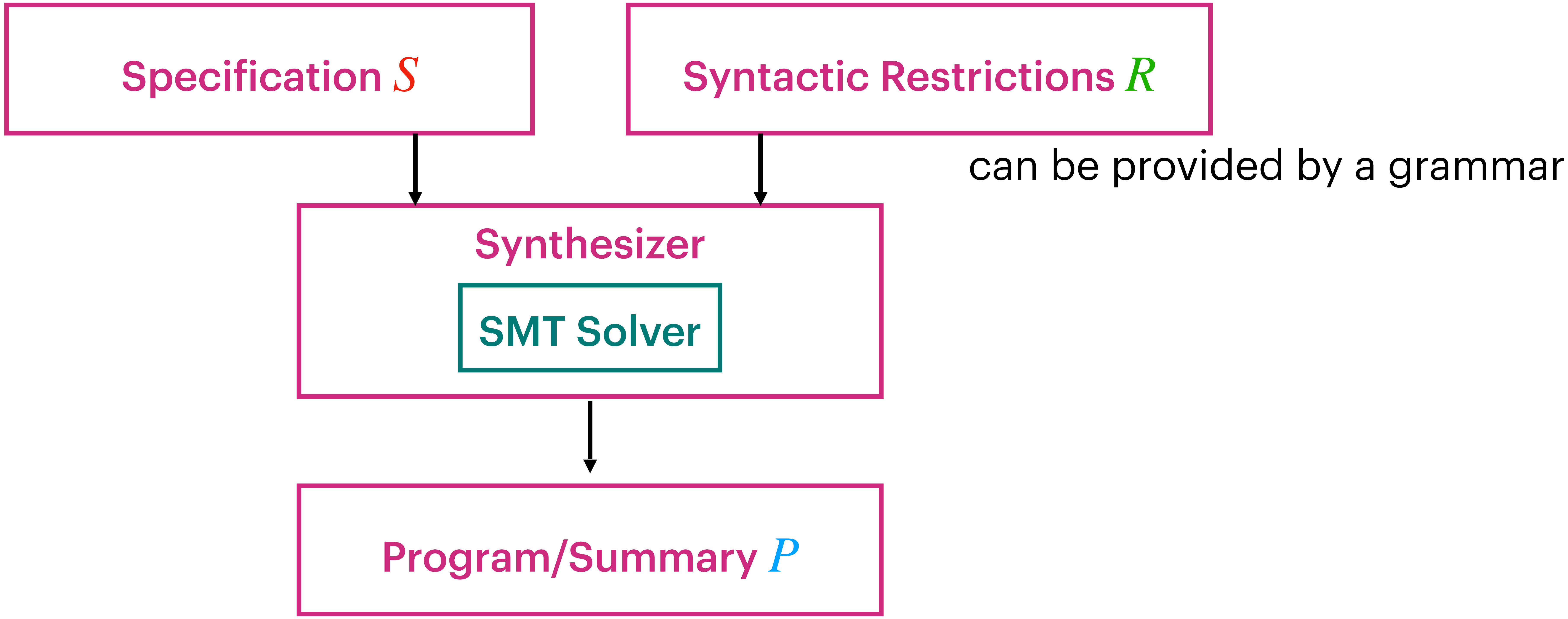
$$\exists\, P \in [\![R]\!] \,.\, \forall i \,.\, P(i) \vDash S(i)$$



Syntax-Guided Synthesis, Alur et al., FMCAD'13

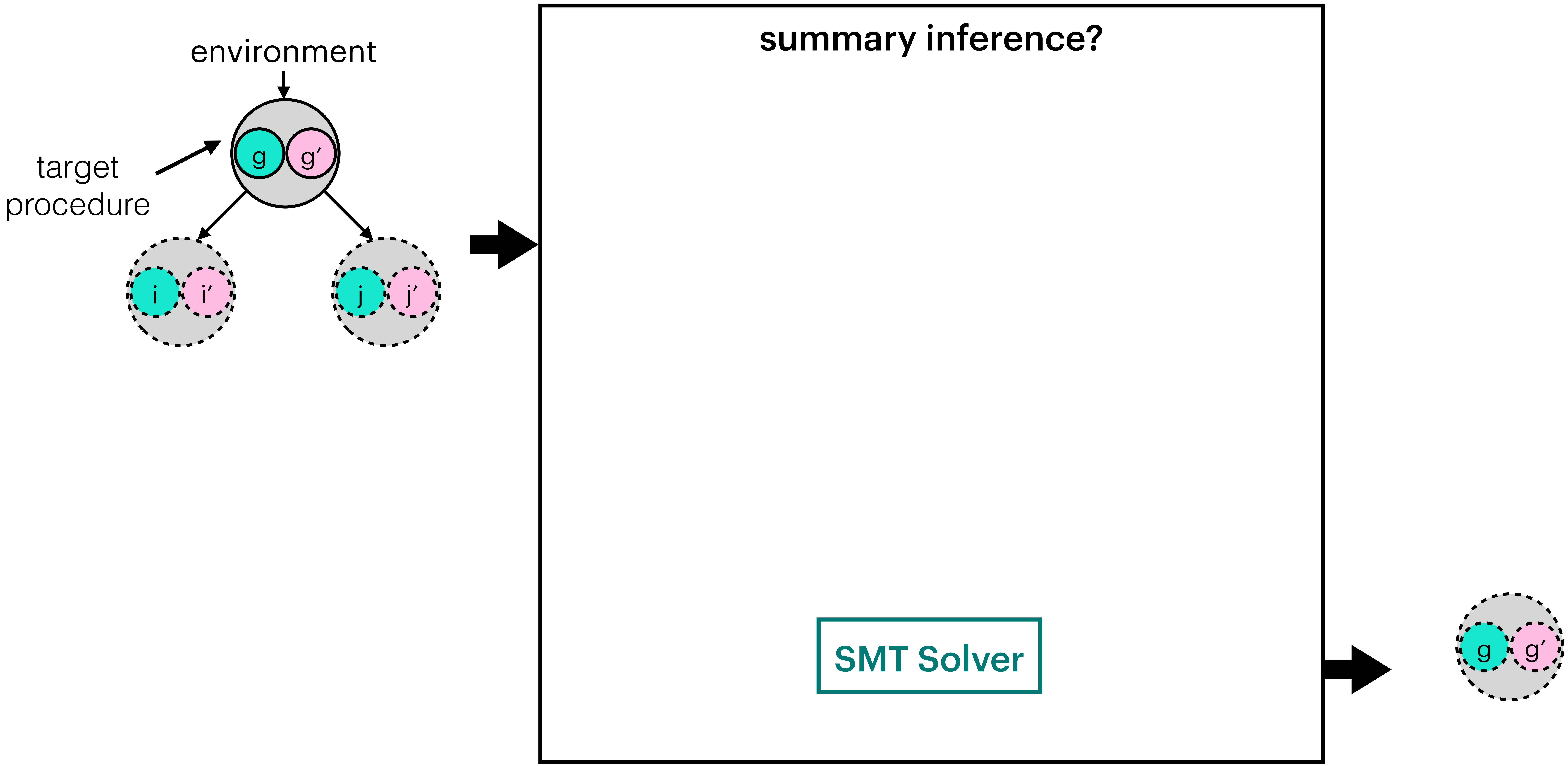
Syntax-Guided Synthesis (SyGuS)

$$\exists P \in [\![R]\!] . \forall i . P(i) \vDash S(i)$$

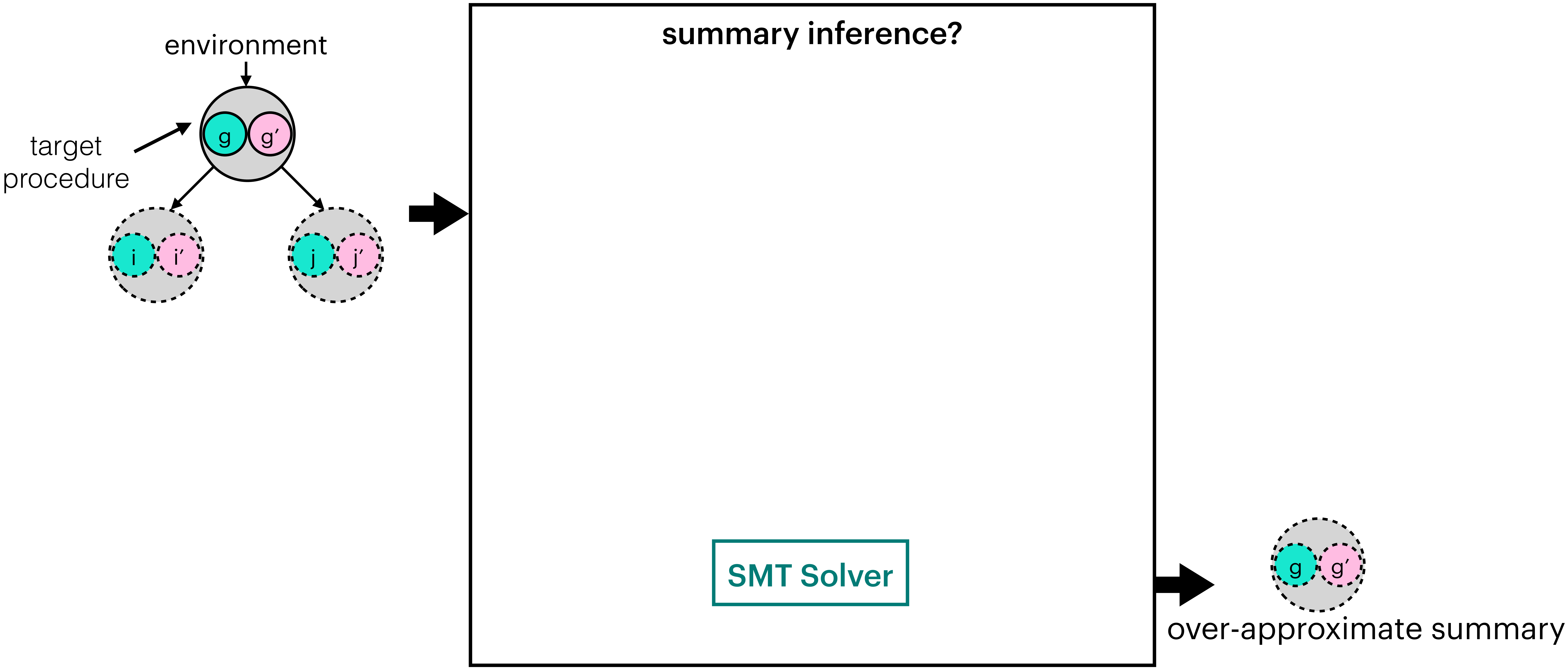


Syntax-Guided Synthesis, Alur et al., FMCAD'13

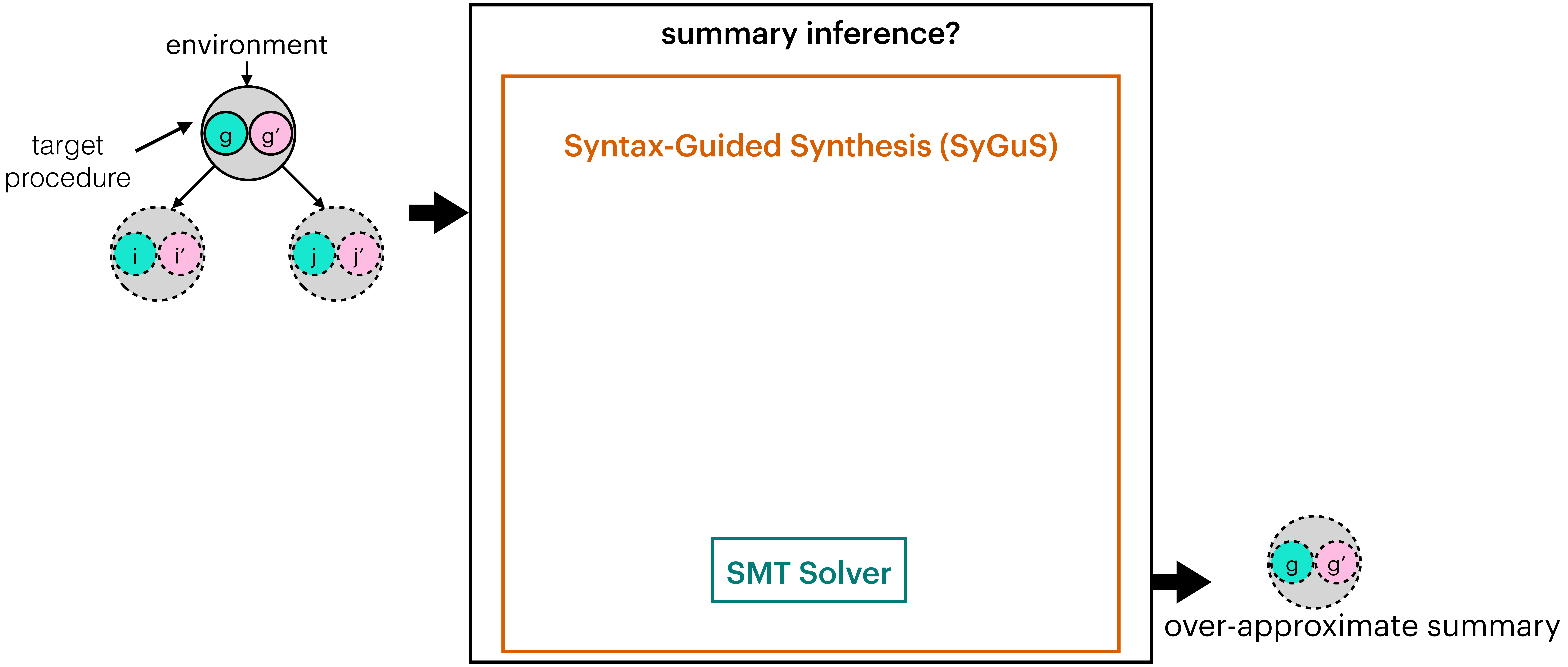
Information-Flow Summary Inference



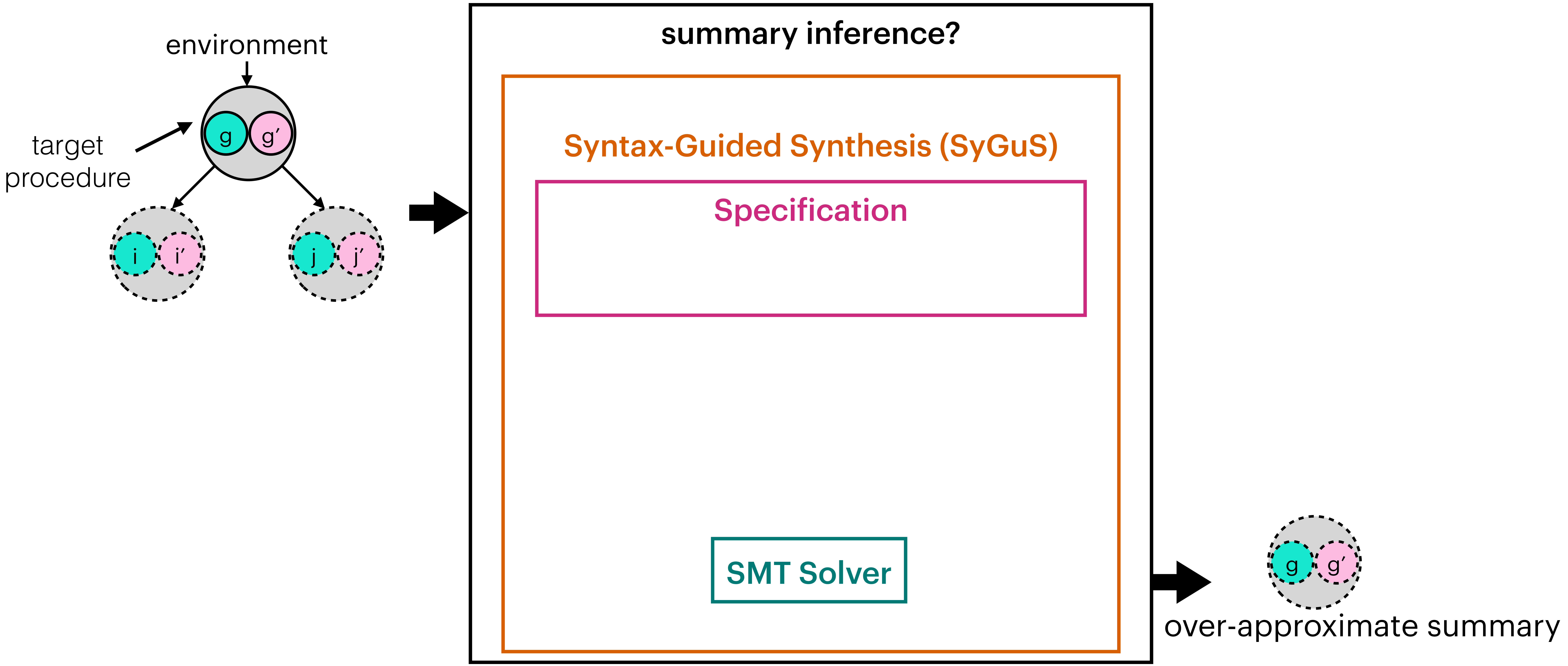
Information-Flow Summary Inference



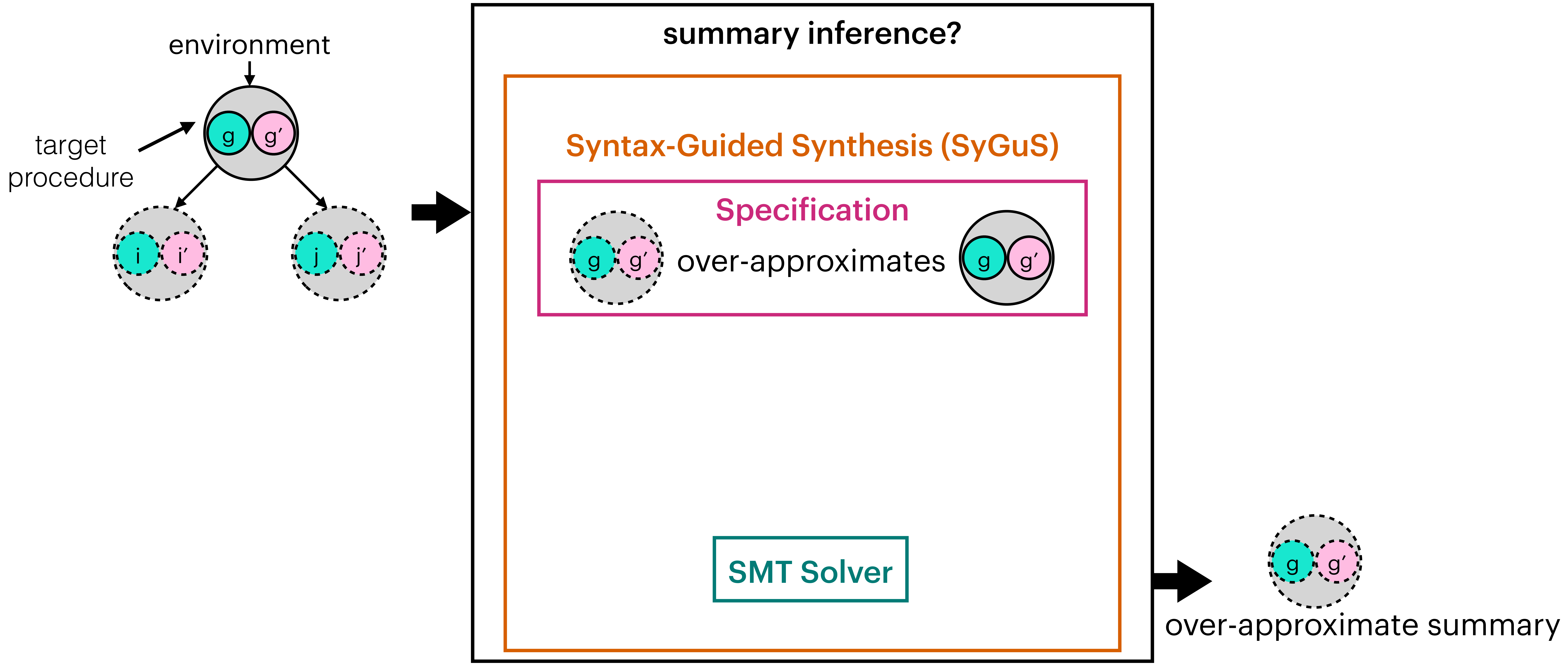
Information-Flow Summary Inference



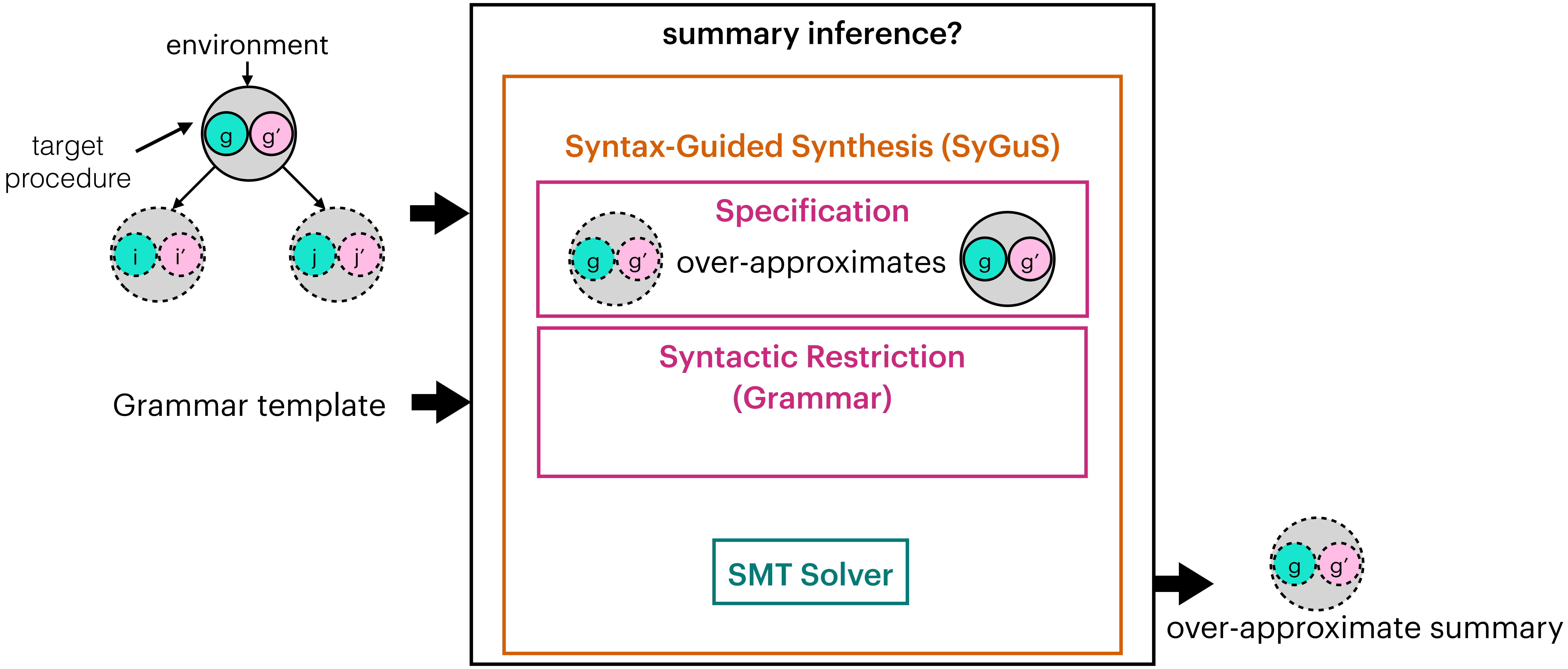
Information-Flow Summary Inference



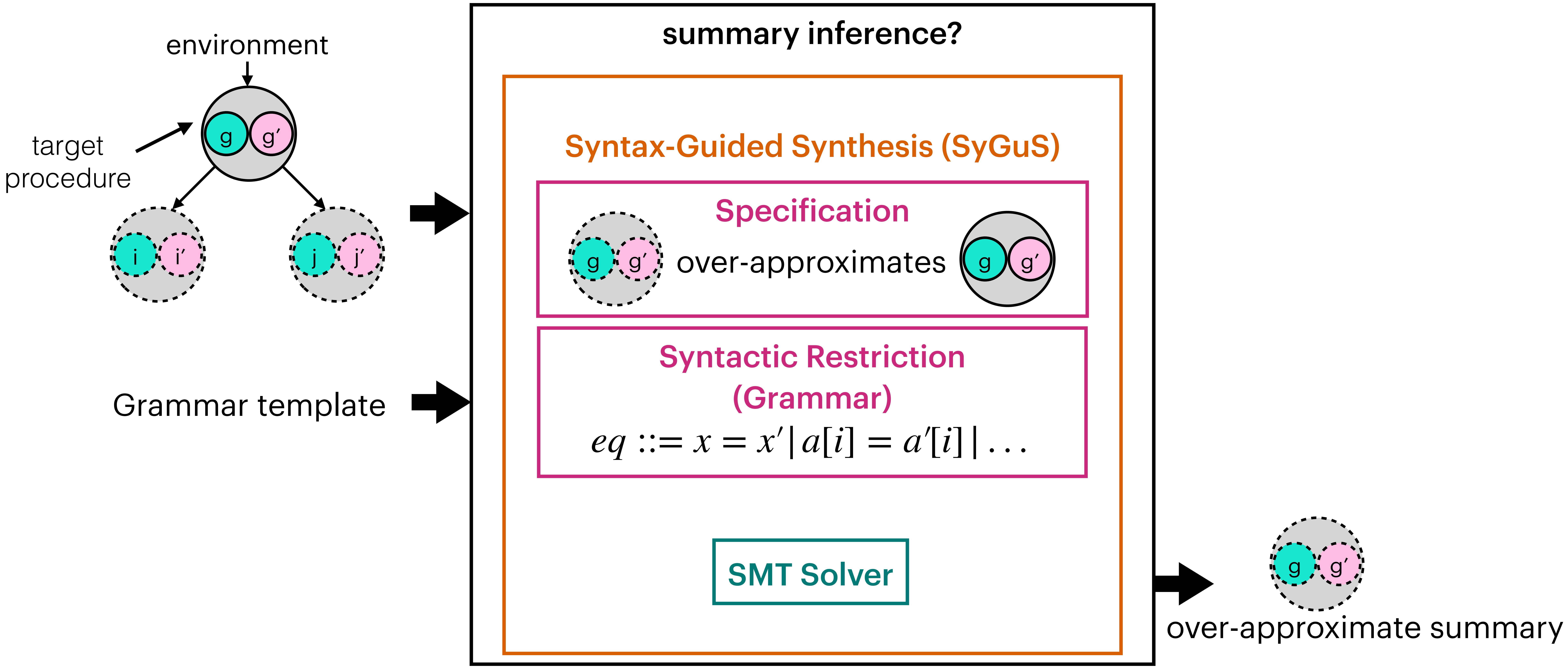
Information-Flow Summary Inference



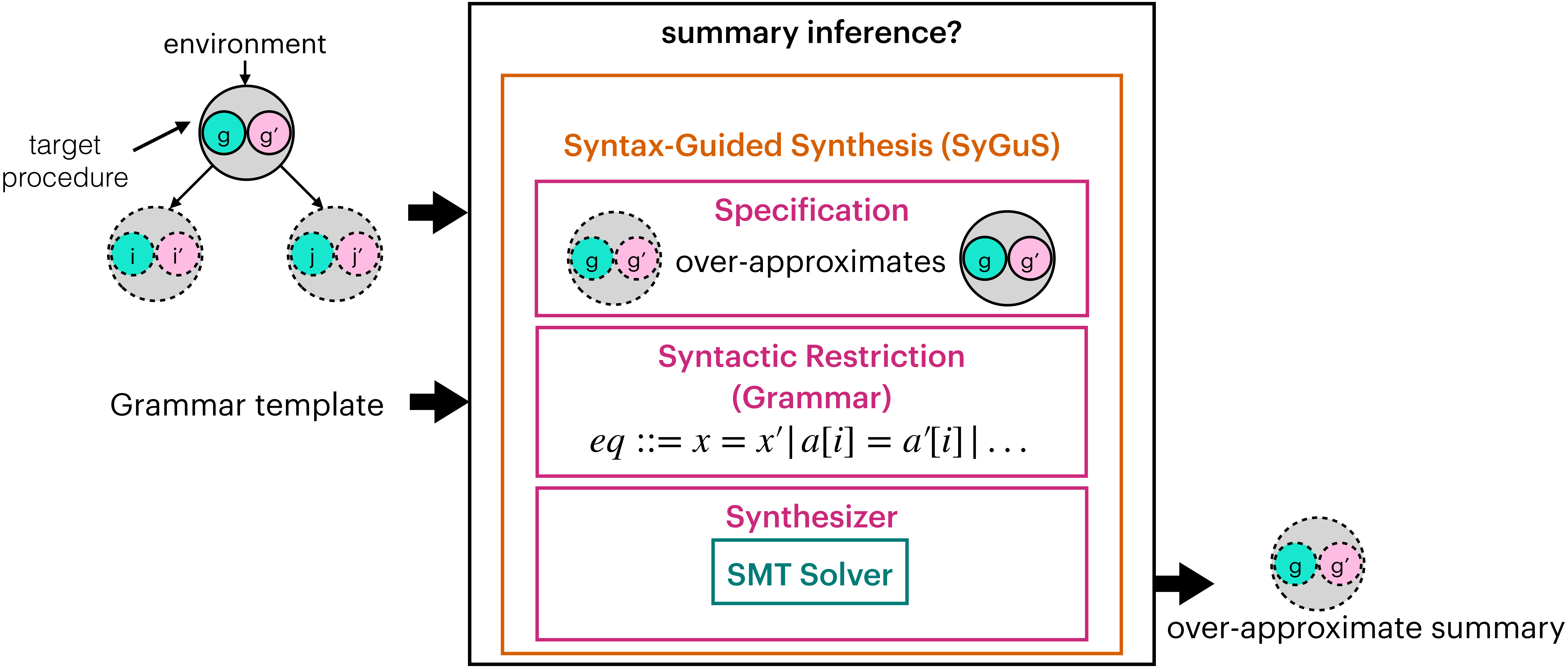
Information-Flow Summary Inference



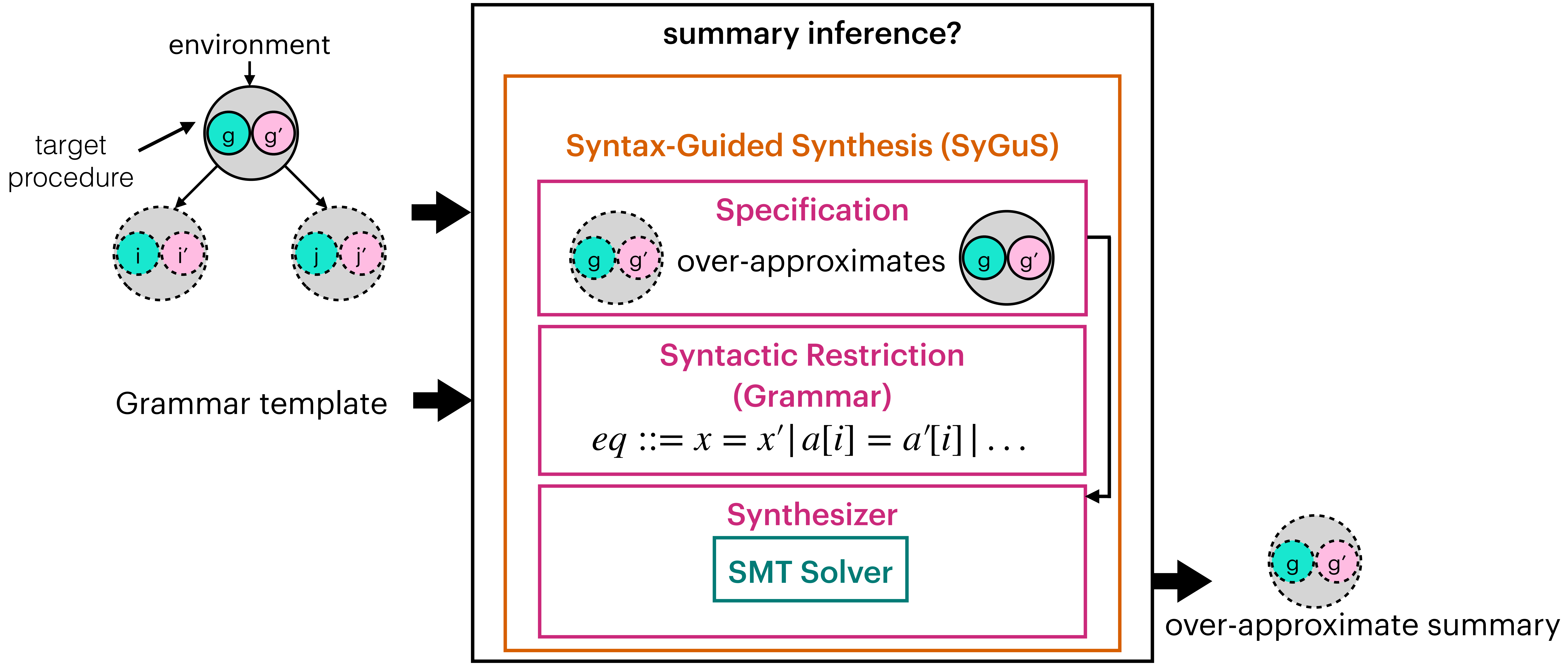
Information-Flow Summary Inference



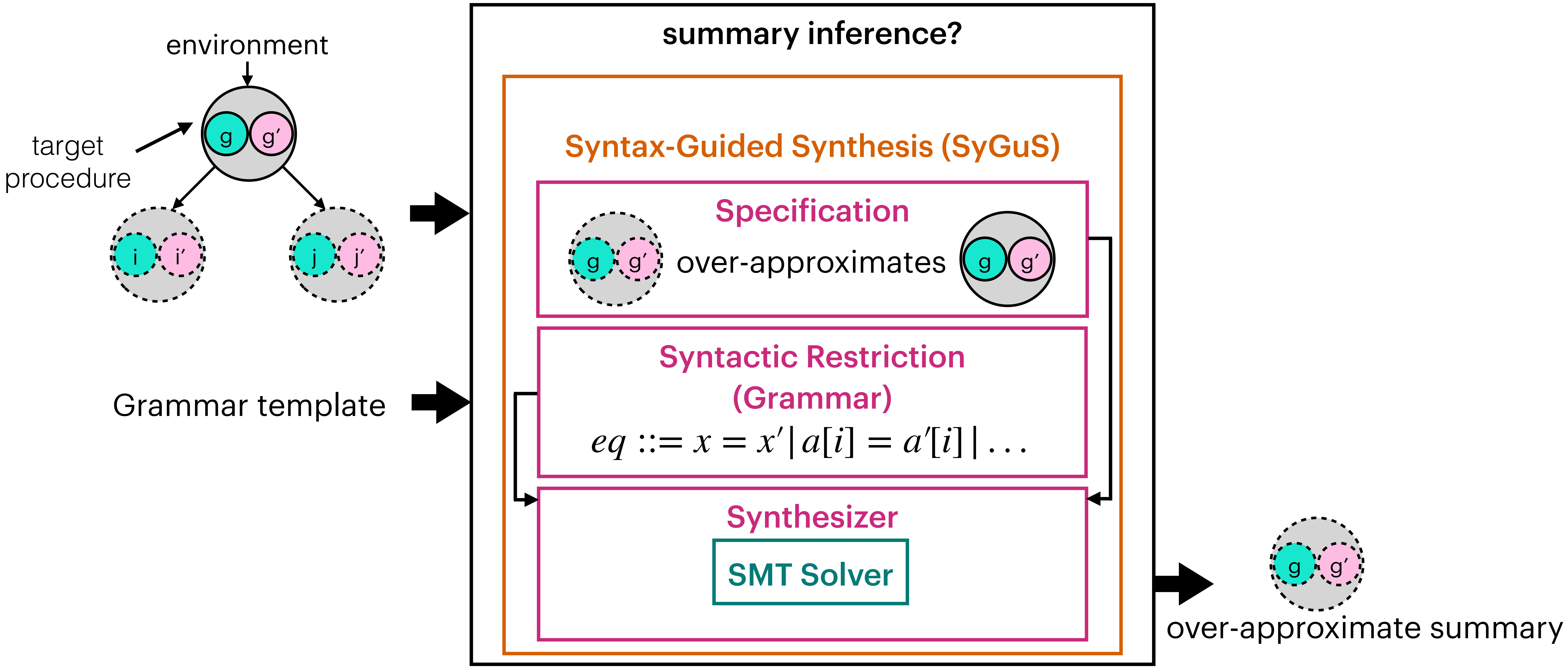
Information-Flow Summary Inference



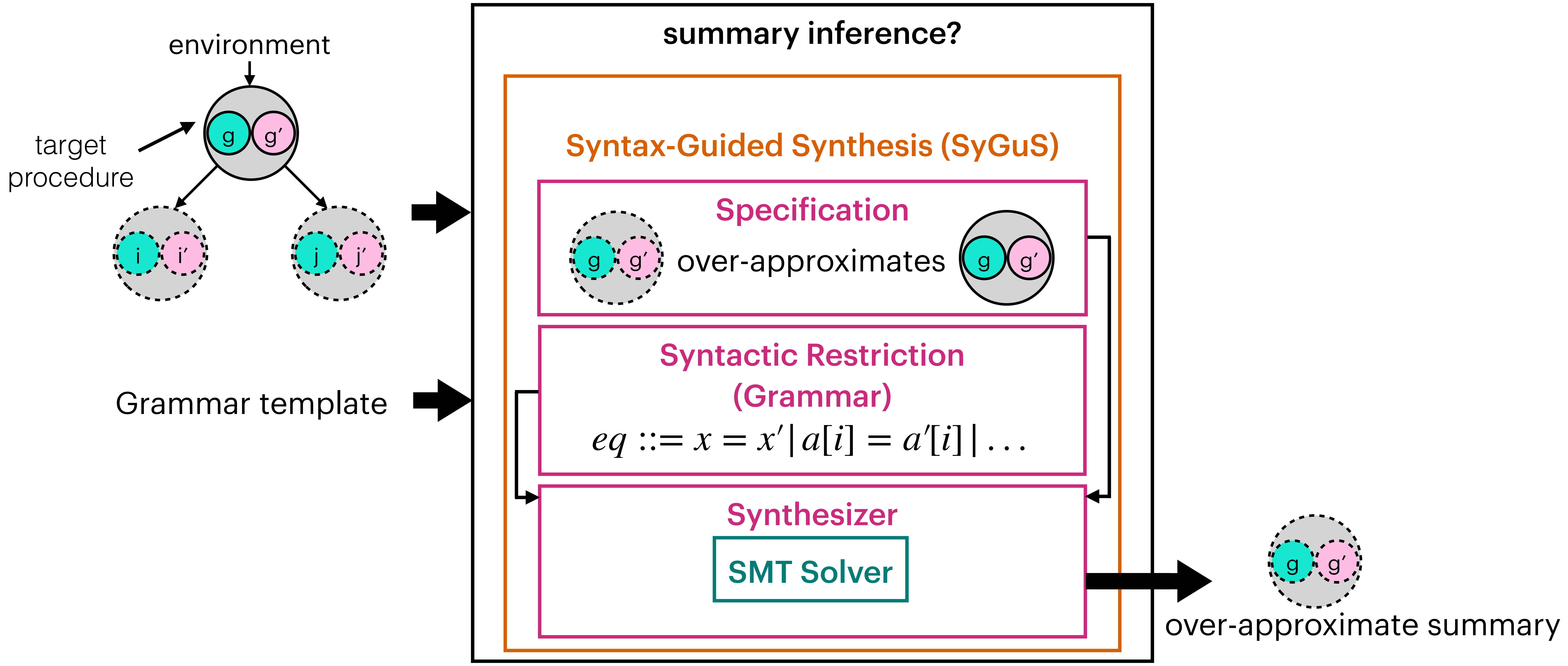
Information-Flow Summary Inference



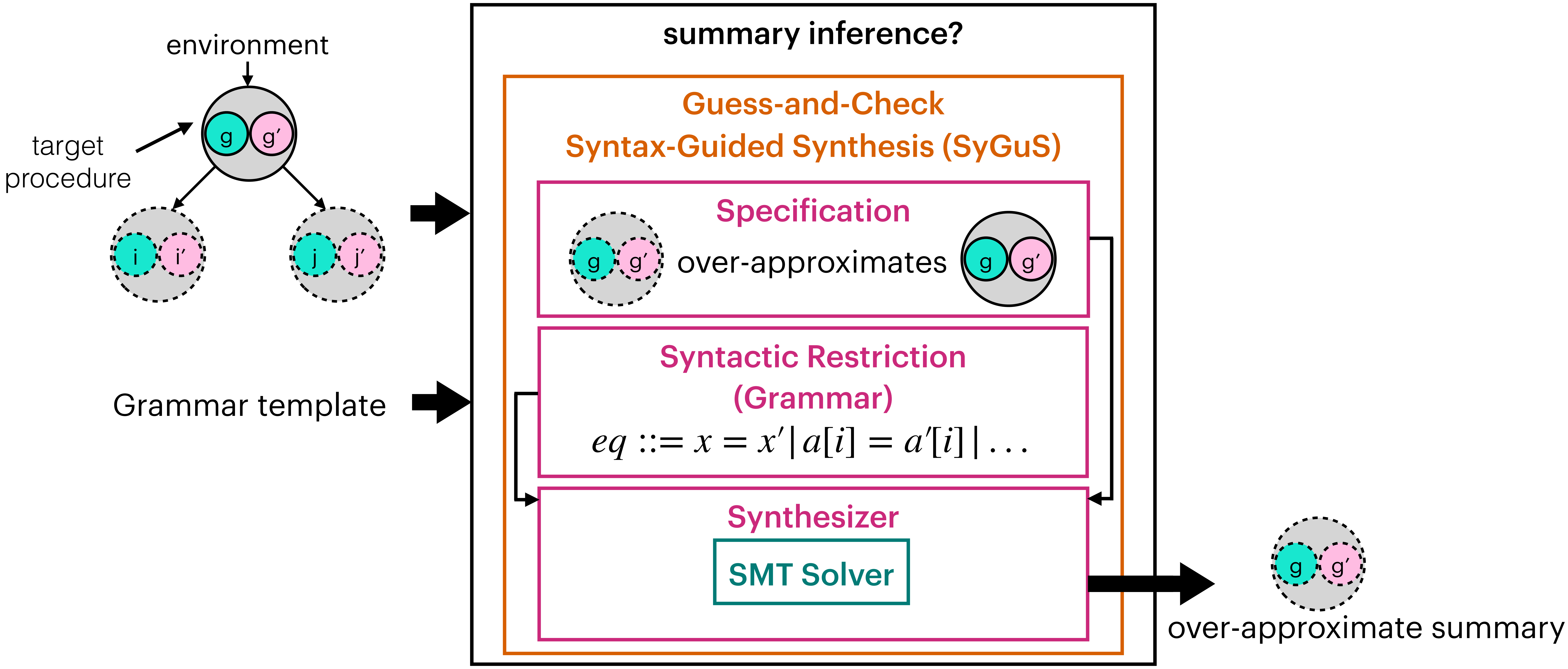
Information-Flow Summary Inference



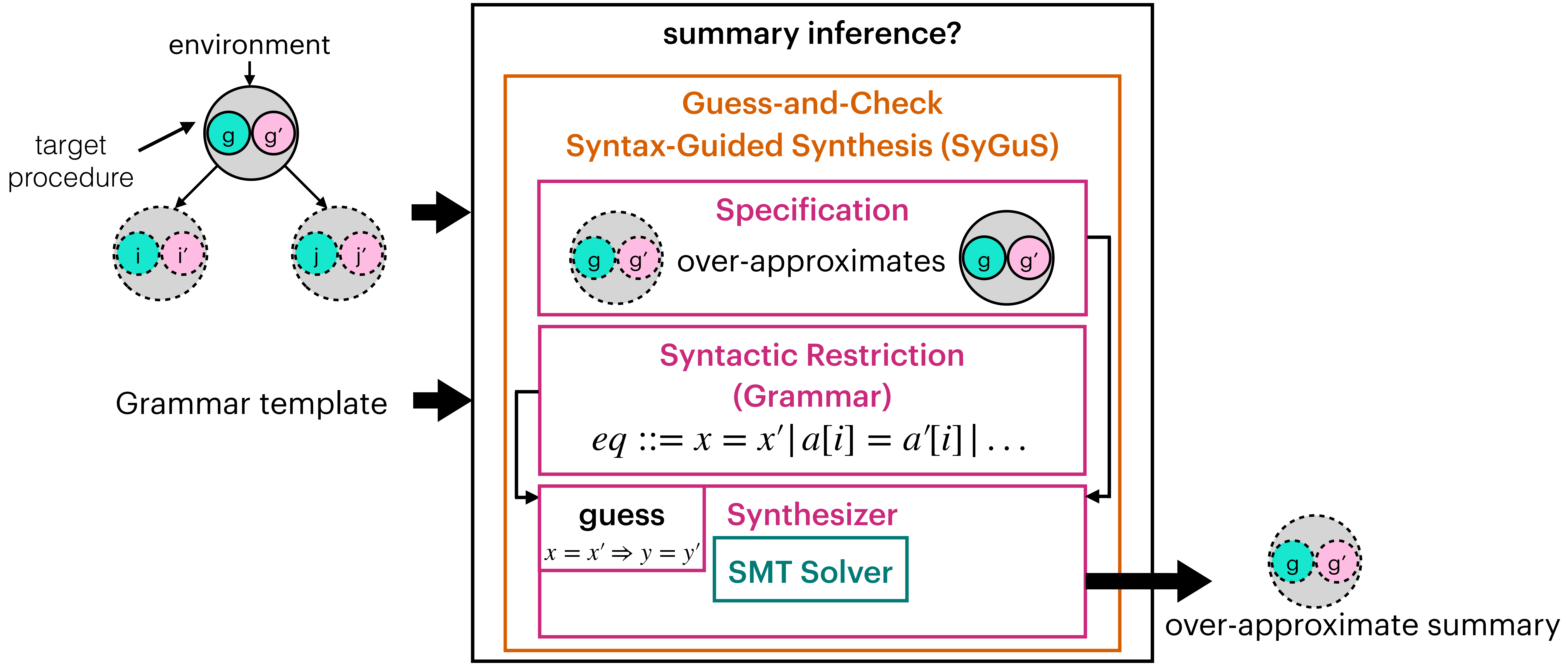
Information-Flow Summary Inference



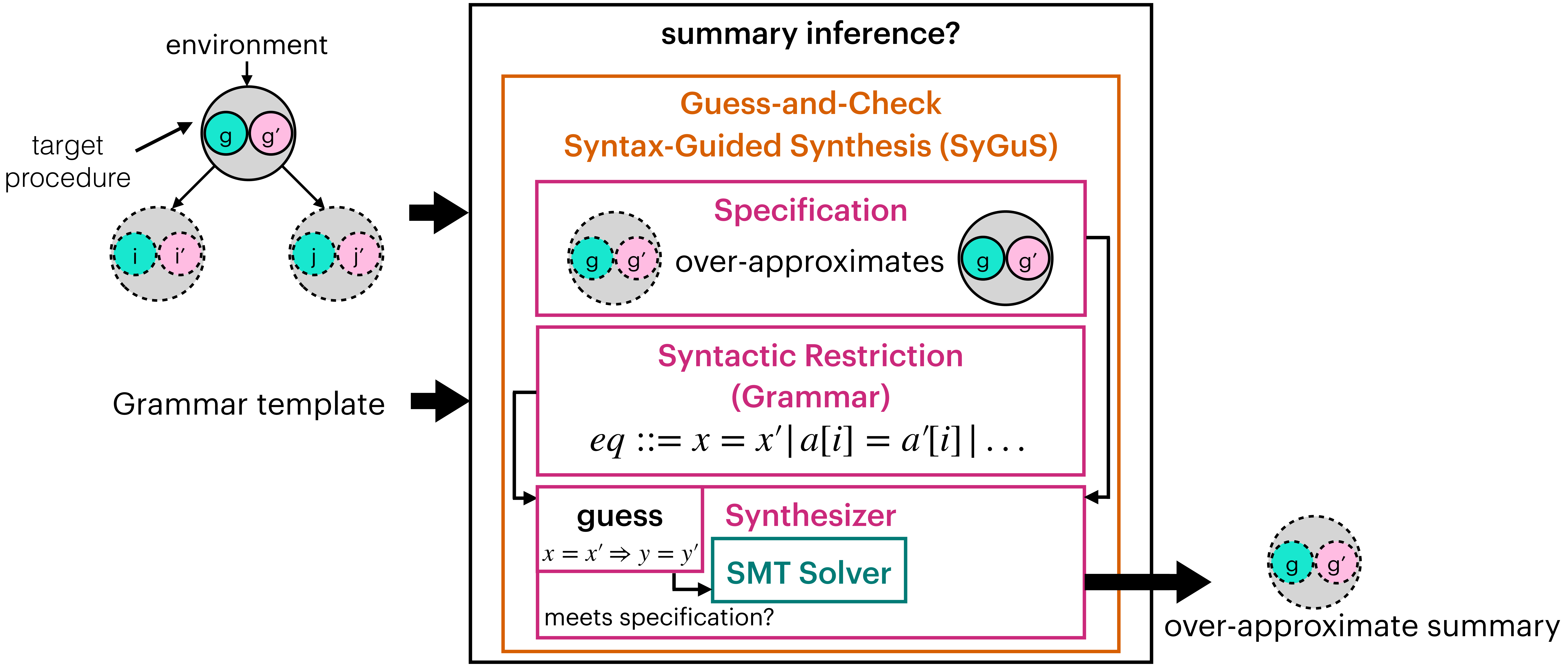
Information-Flow Summary Inference



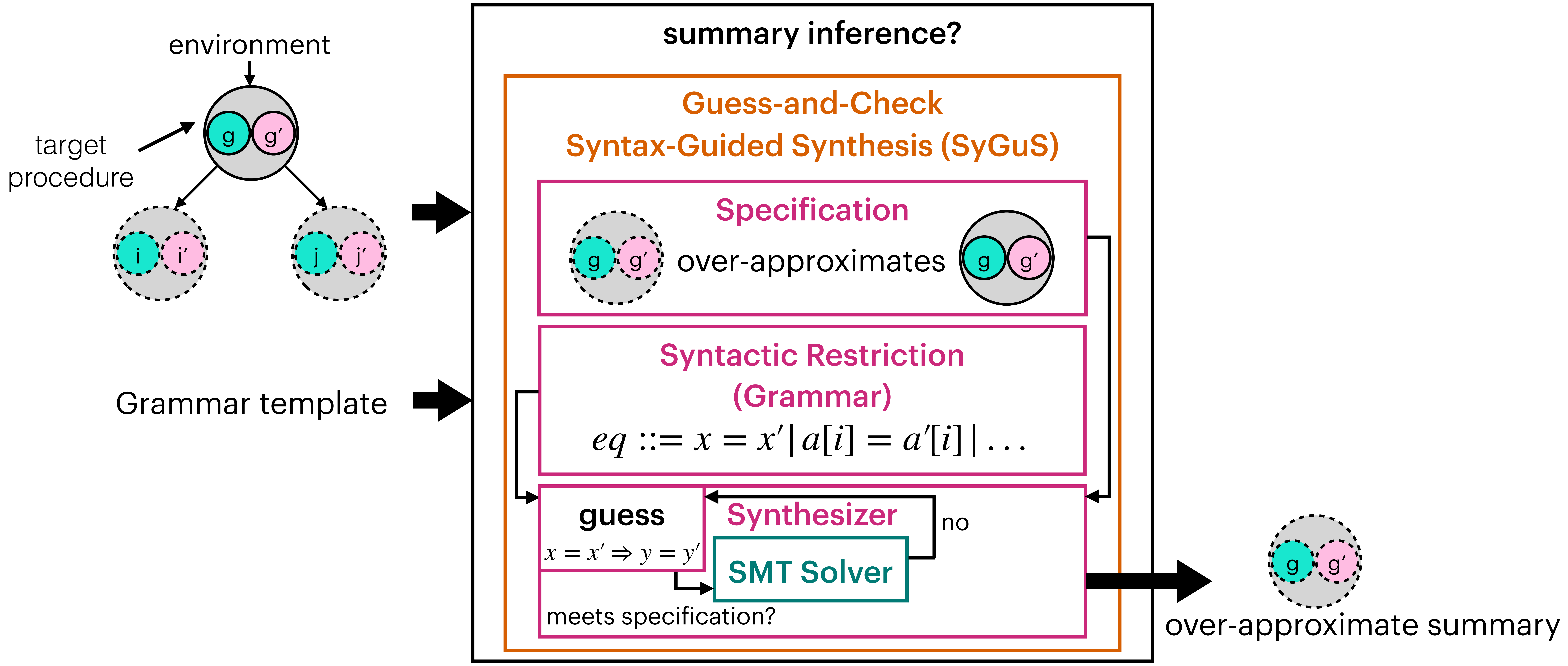
Information-Flow Summary Inference



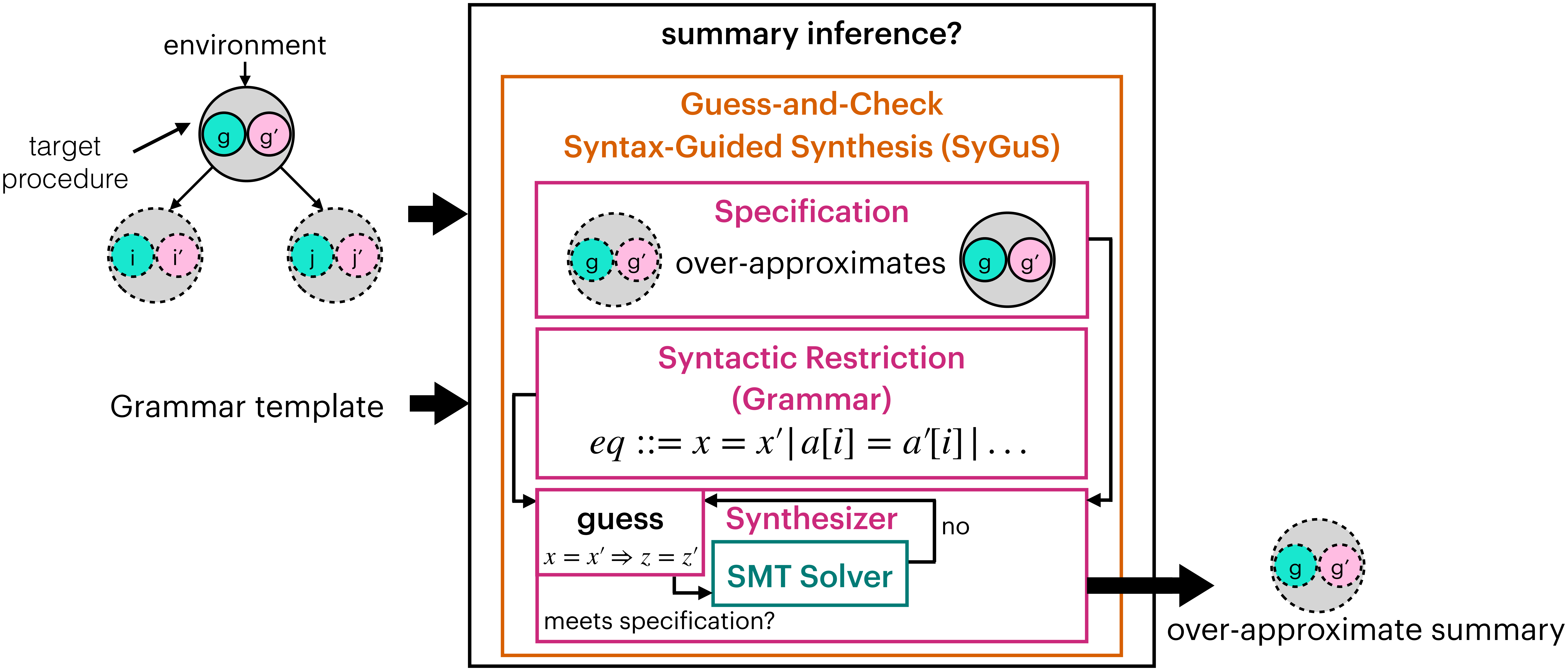
Information-Flow Summary Inference



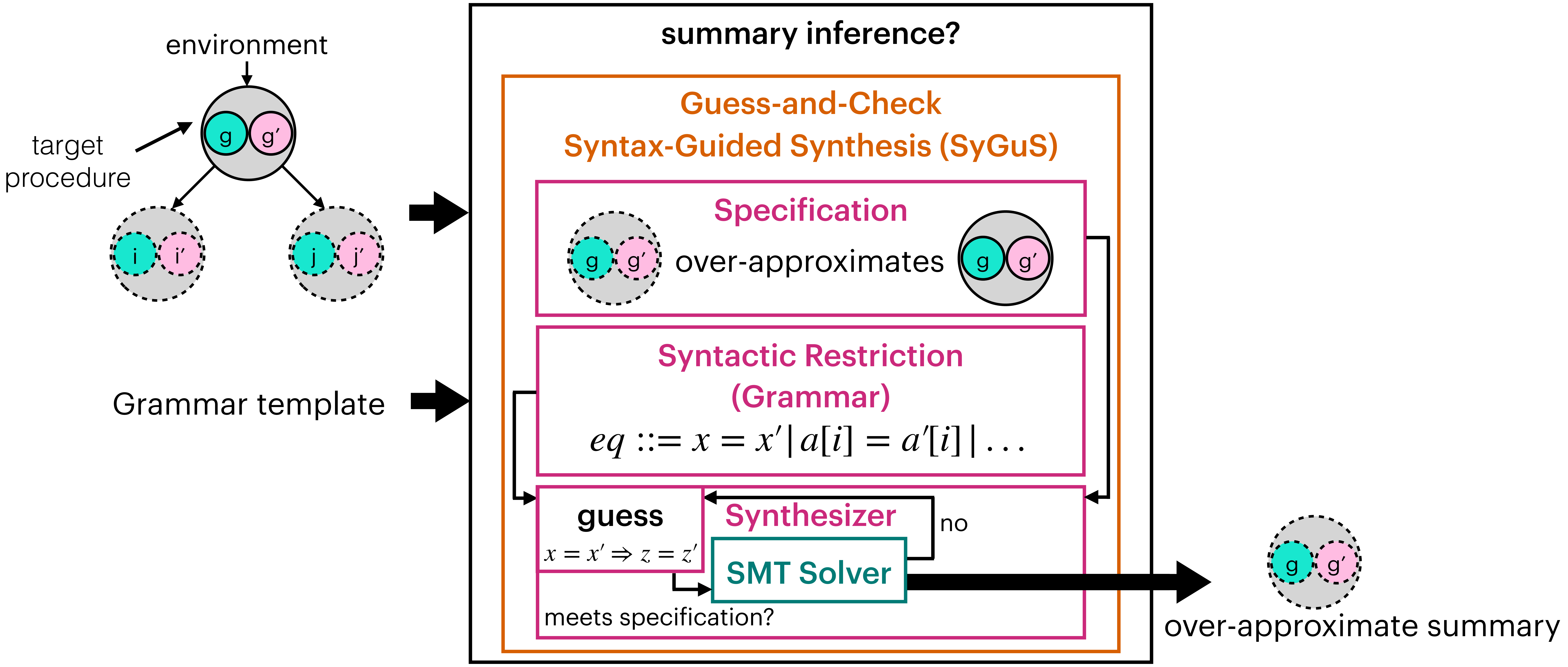
Information-Flow Summary Inference



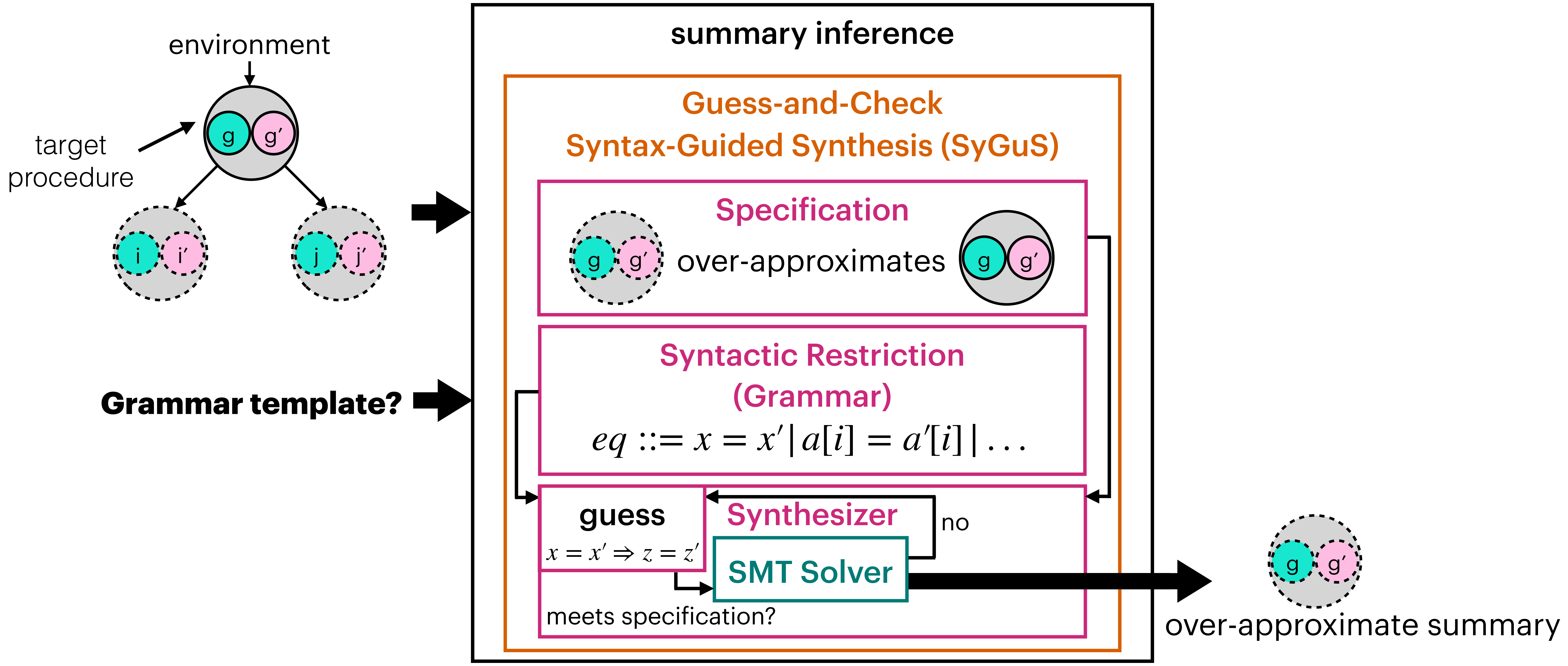
Information-Flow Summary Inference



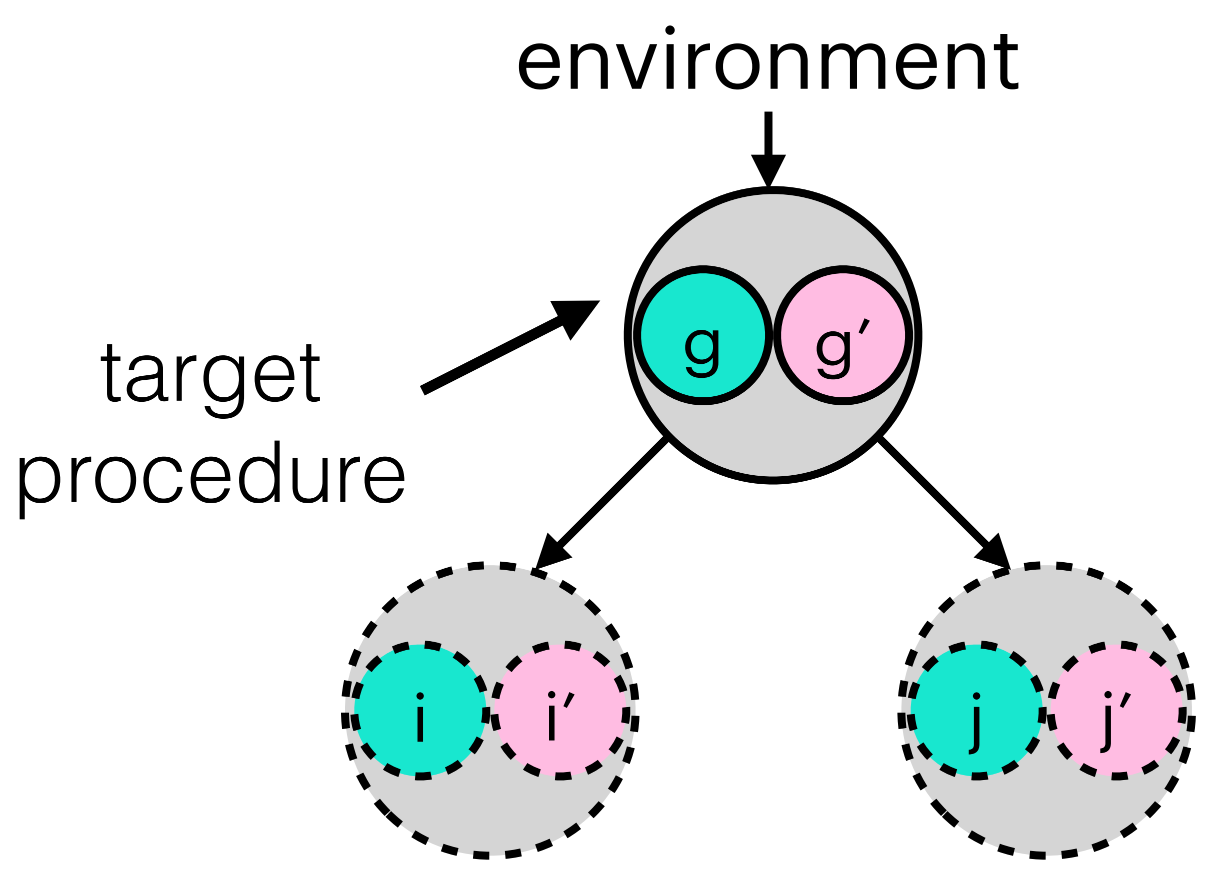
Information-Flow Summary Inference



Inferring Summaries with SyGuS

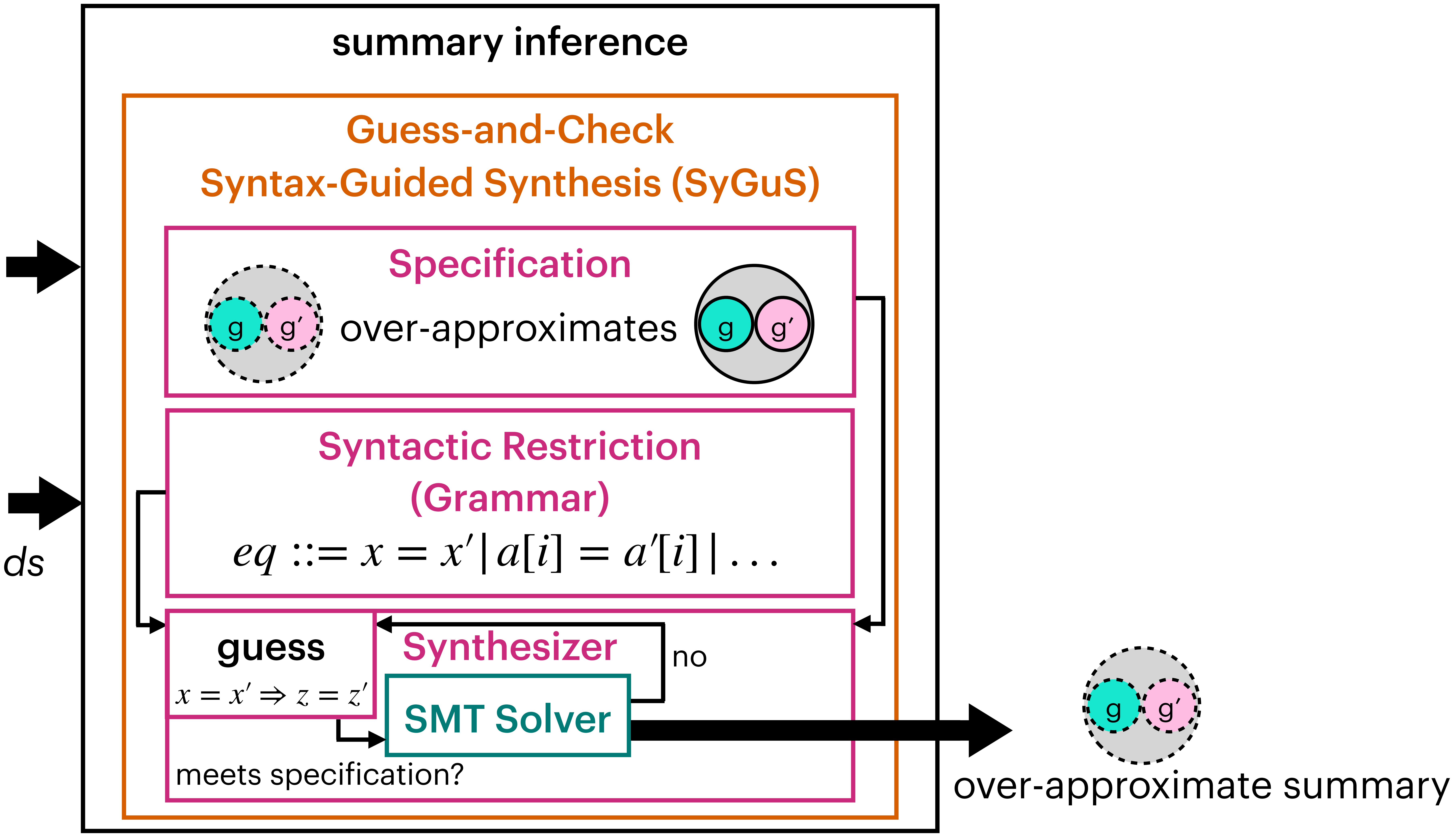


Inferring Summaries with SyGuS

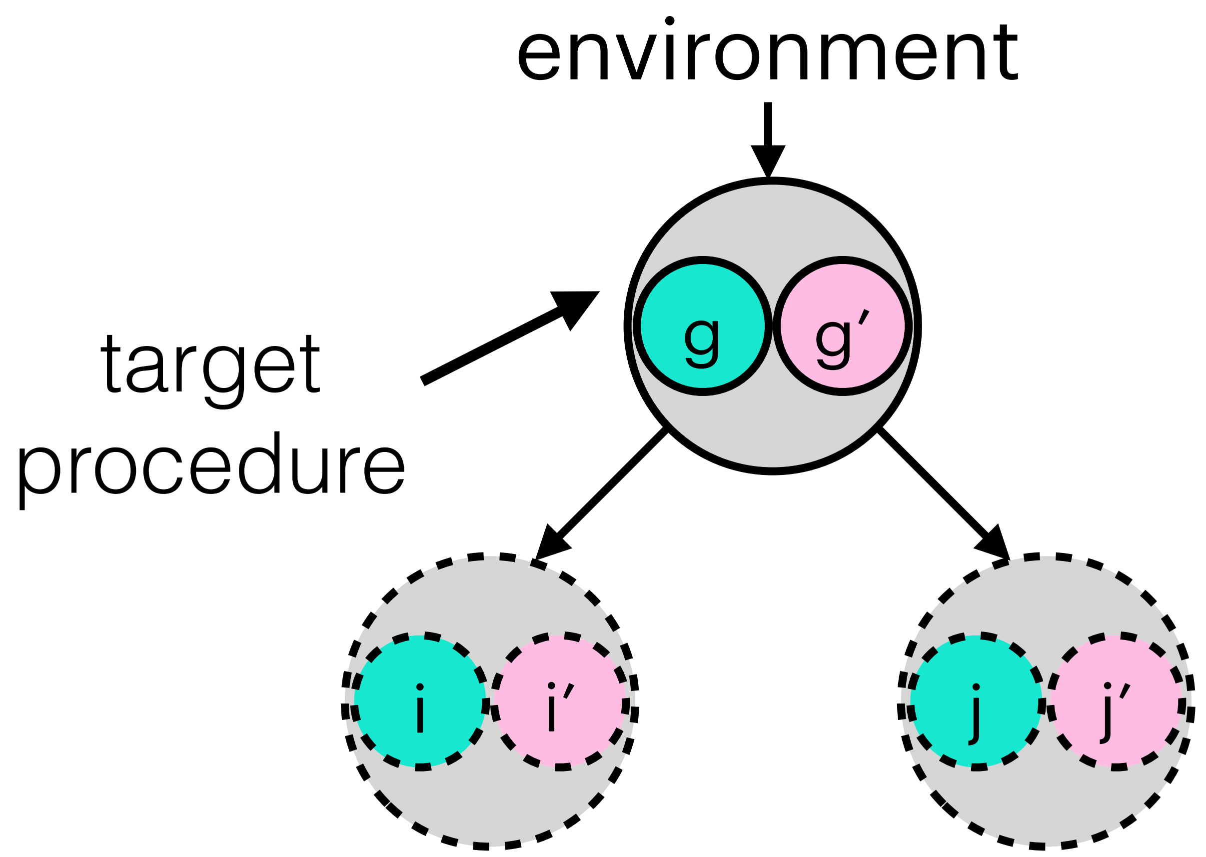


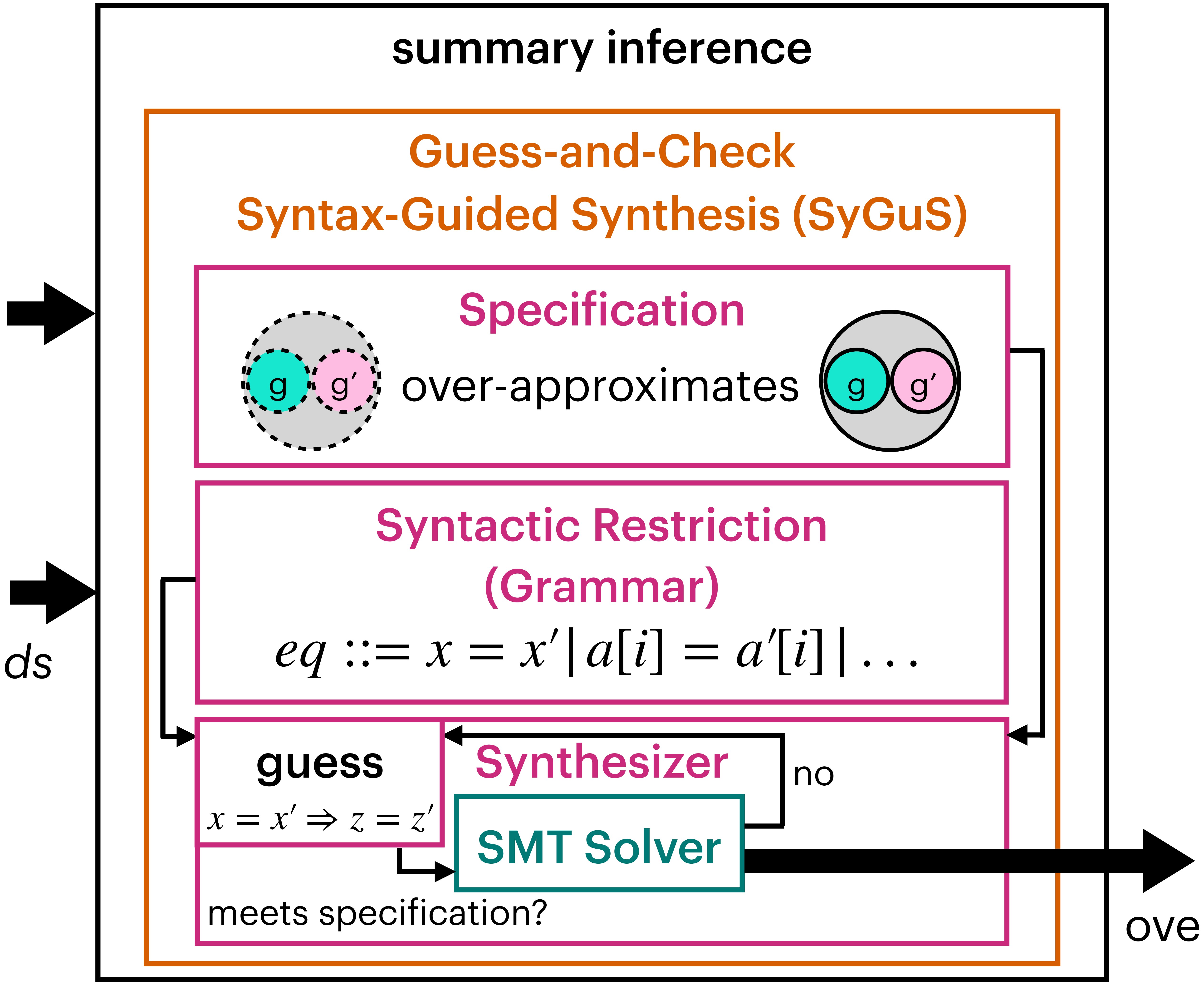
Grammar template?

Will talk about *three kinds*



Inferring Summaries with SyGuS





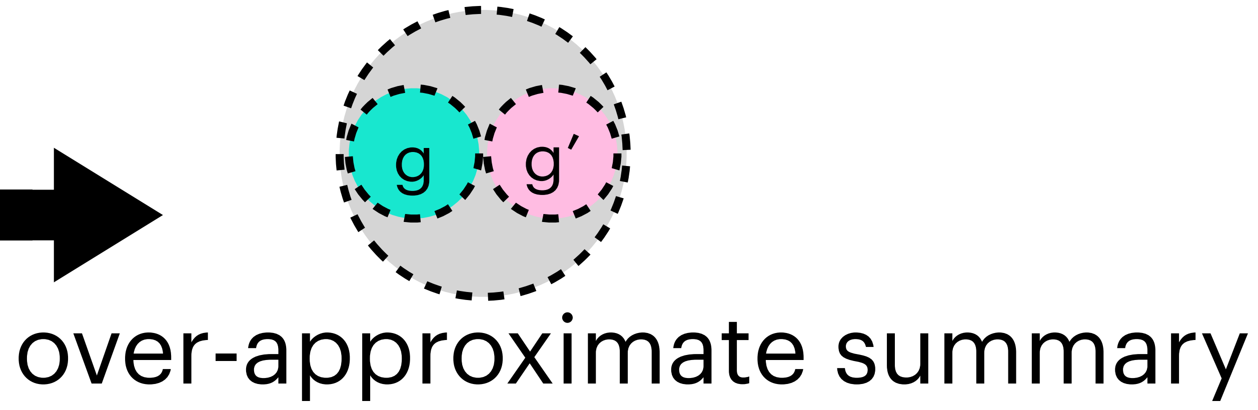
Grammar template?

Will talk about *three kinds*

Quantifier-Free

Quantified Array

Property-Directed



Grammar Templates

Grammar Templates

Insight:
information flow involves **equalities** on subsets of corresponding components

Grammar Templates

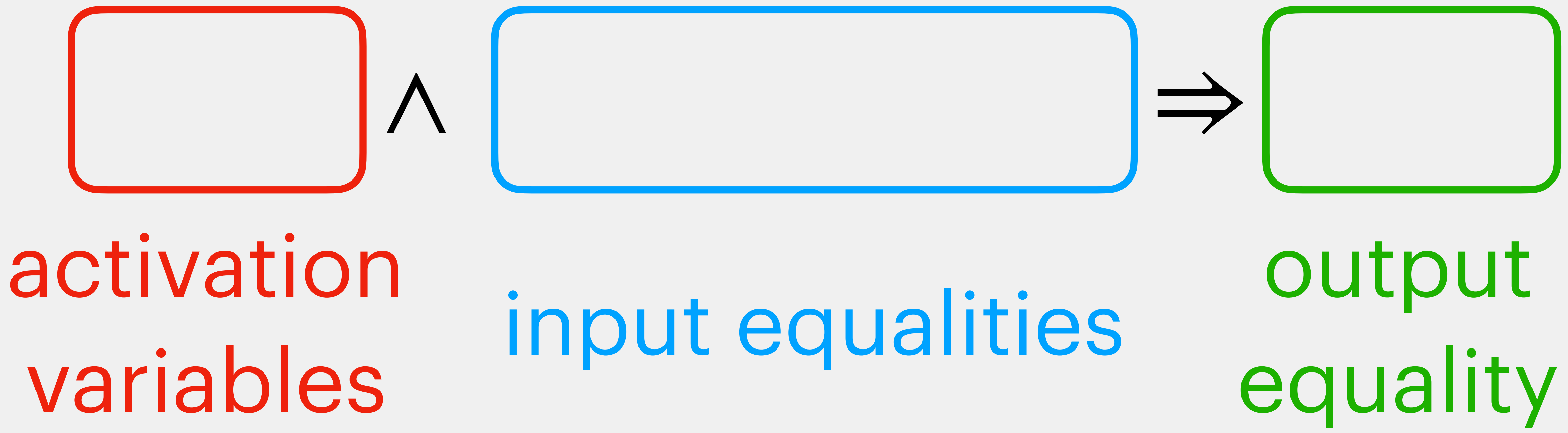
Insight:
information flow involves **equalities** on subsets of corresponding components

Quantifier-free

Grammar Templates

Insight:
information flow involves **equalities** on subsets of corresponding components

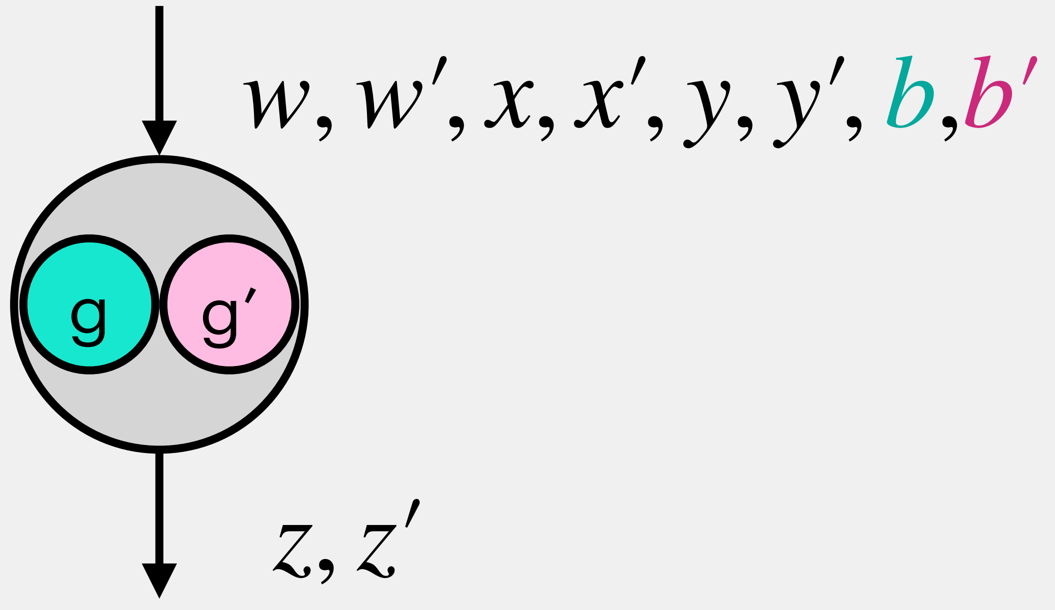
Quantifier-free

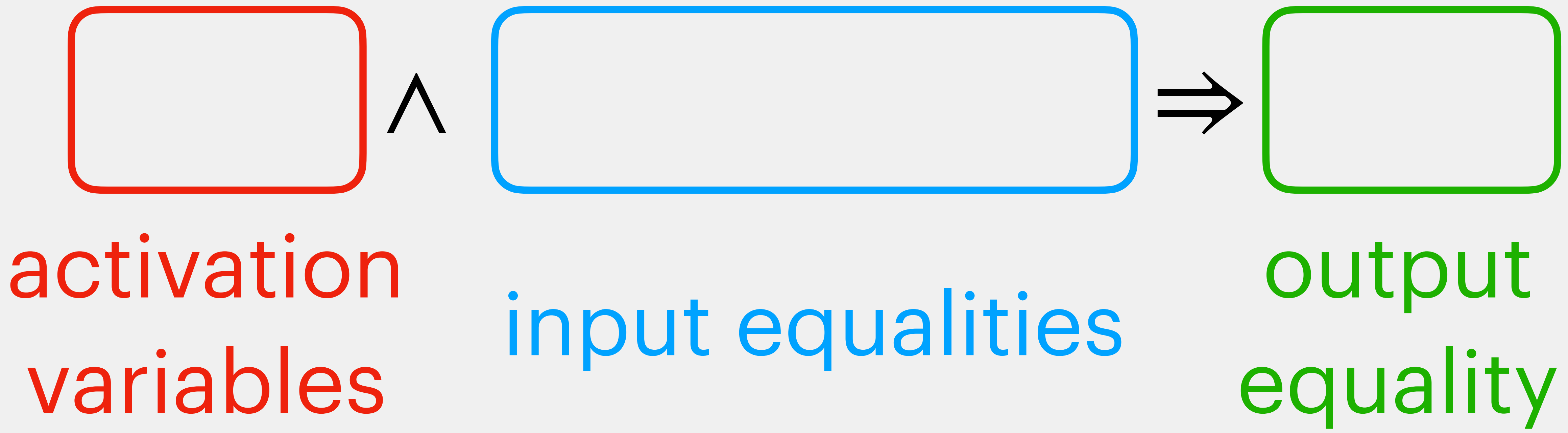


Grammar Templates

Insight:
information flow involves **equalities** on subsets of corresponding components

Quantifier-free



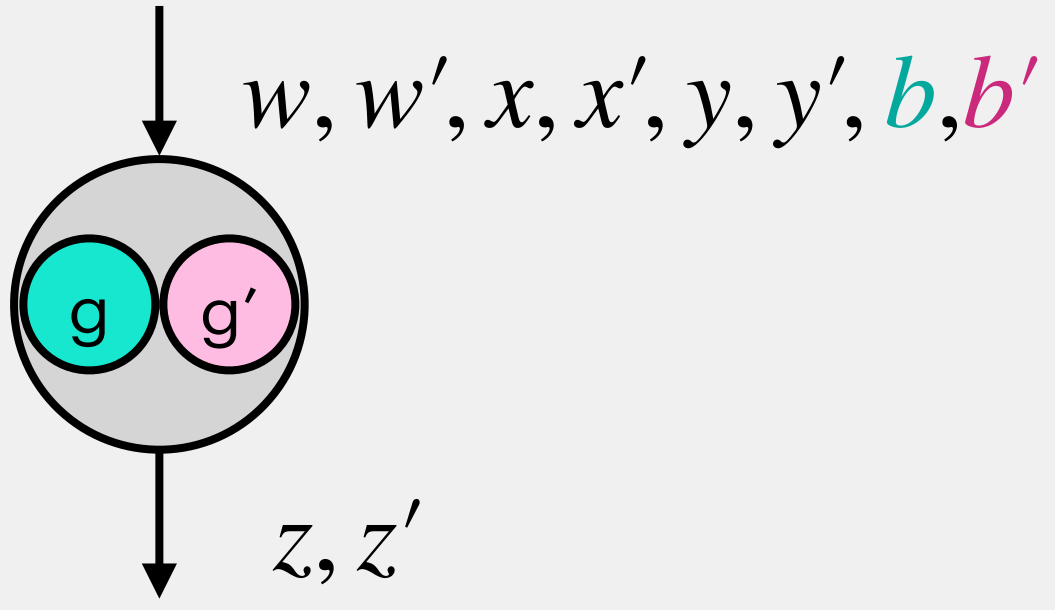


Grammar Templates

Insight:

information flow involves **equalities** on subsets of corresponding components

Quantifier-free



$b \wedge b' \wedge \square \Rightarrow \square$

activation variables

input equalities

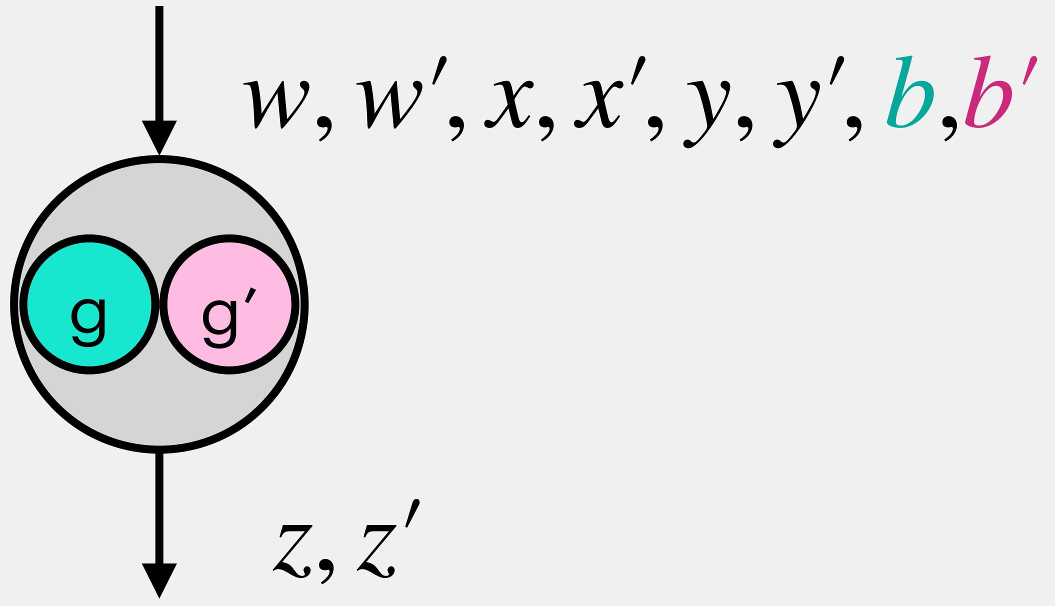
output equality

Grammar Templates

Insight:

information flow involves **equalities** on subsets of corresponding components

Quantifier-free



$$b \wedge b' \wedge x = x' \wedge y = y' \Rightarrow \square$$

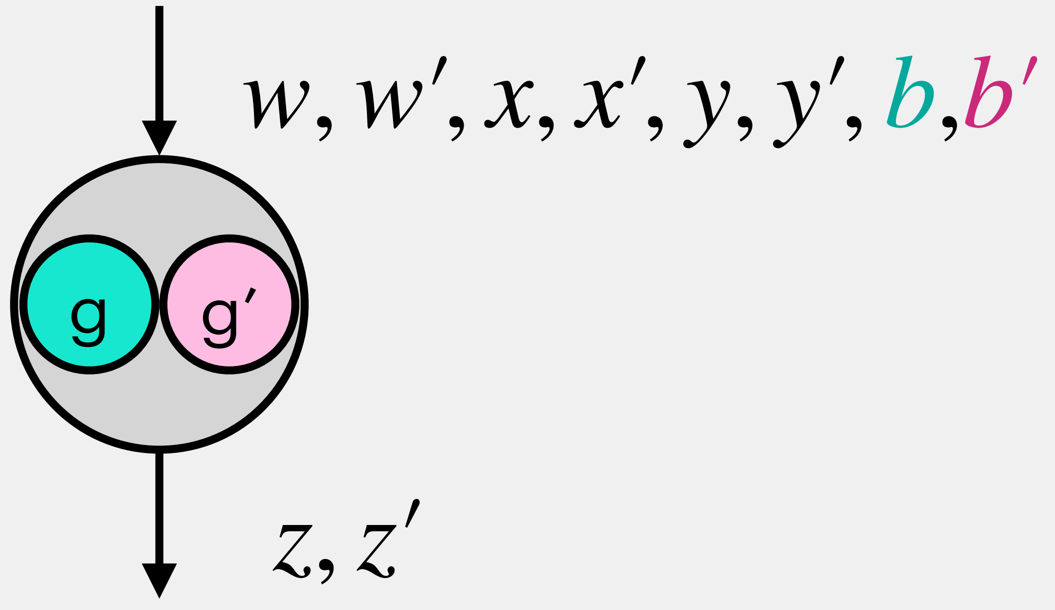
activation variables | input equalities | output equality

Grammar Templates

Insight:

information flow involves **equalities** on subsets of corresponding components

Quantifier-free



$$\boxed{b \land b'} \land \boxed{x = x' \land y = y'} \Rightarrow \boxed{z = z'}$$

activation variables

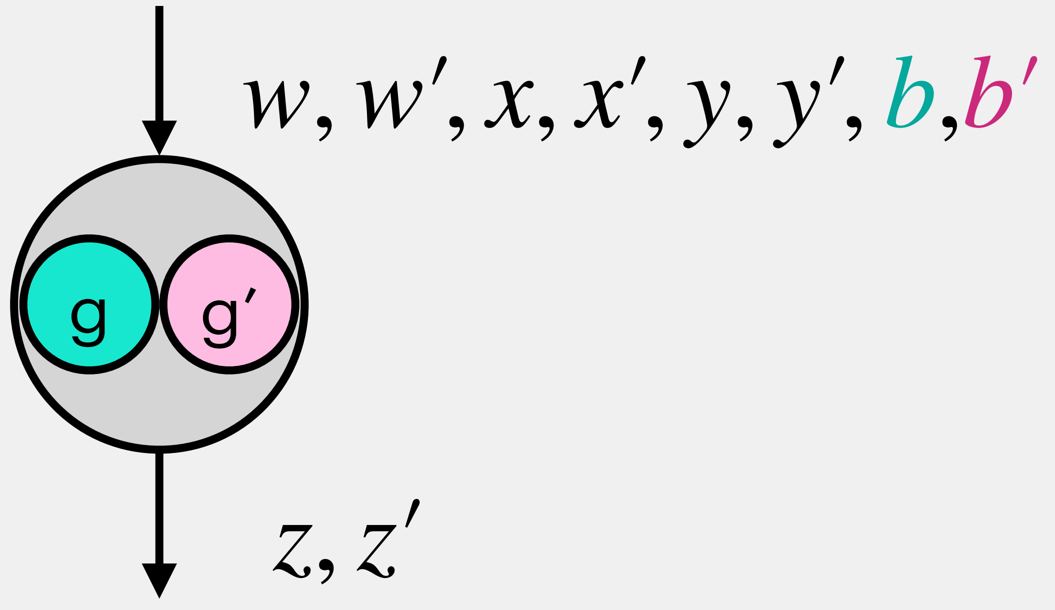
input equalities

output equality

Grammar Templates

Insight:
information flow involves **equalities** on subsets of corresponding components

Quantifier-free



$$b \wedge b' \wedge x = x' \wedge y = y' \Rightarrow z = z'$$

activation variables ($b \wedge b'$)

input equalities ($x = x' \wedge y = y'$)

output equality ($z = z'$)

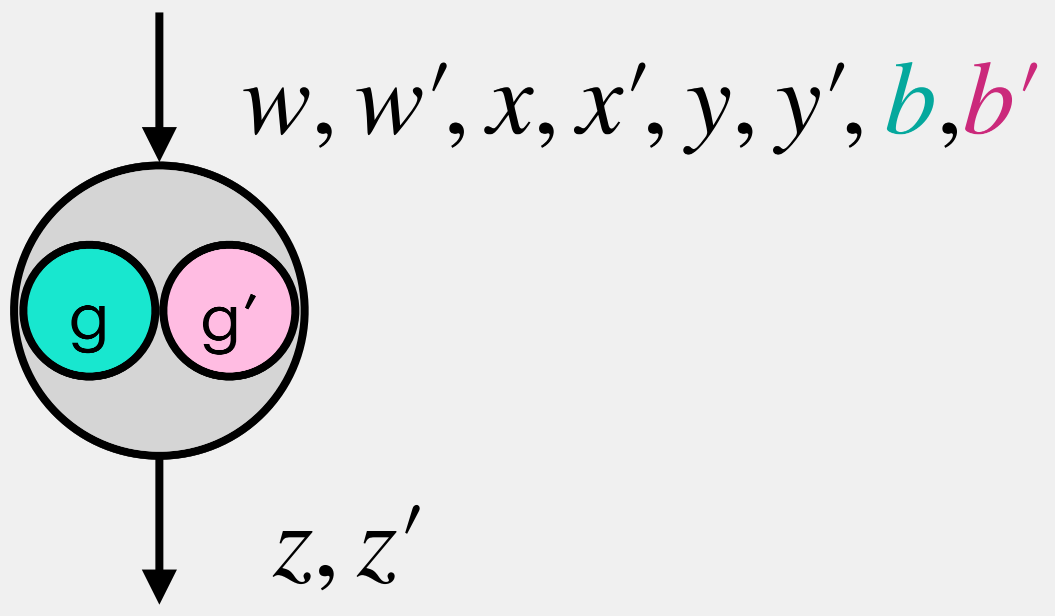
Quantified Array

Grammar Templates

Insight:

information flow involves **equalities** on subsets of corresponding components

Quantifier-free



$$\boxed{b \wedge b'} \wedge \boxed{x = x' \wedge y = y'} \Rightarrow \boxed{z = z'}$$

activation variables | input equalities | output equality

Quantified Array

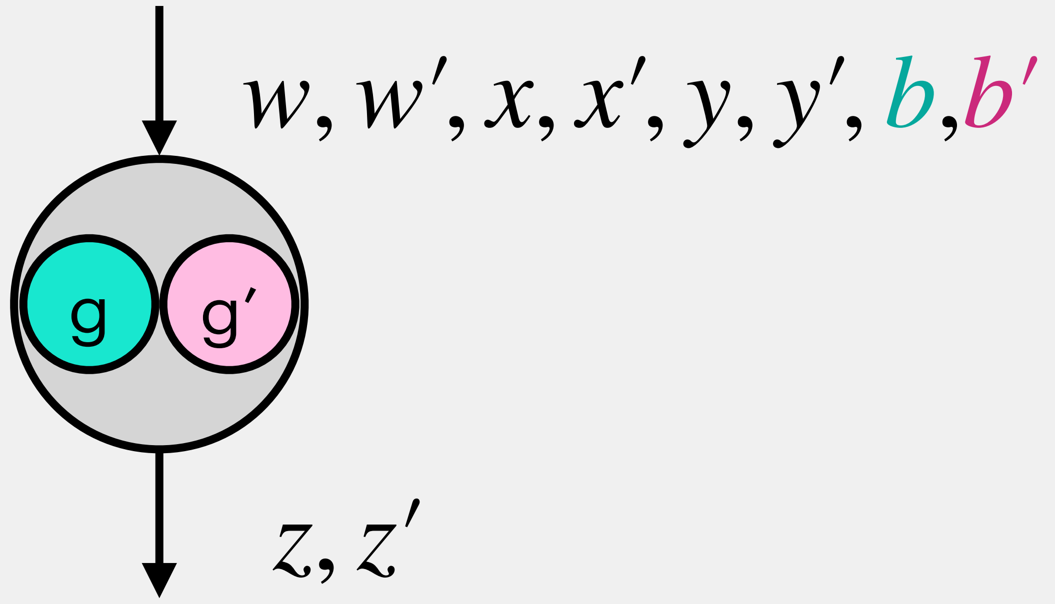
$$\forall \boxed{} . \boxed{} \wedge \boxed{} \wedge \boxed{} \Rightarrow \boxed{}$$

quantified indices | range | activation variables | equalities | cell property

Grammar Templates

Insight:
information flow involves **equalities** on subsets of corresponding components

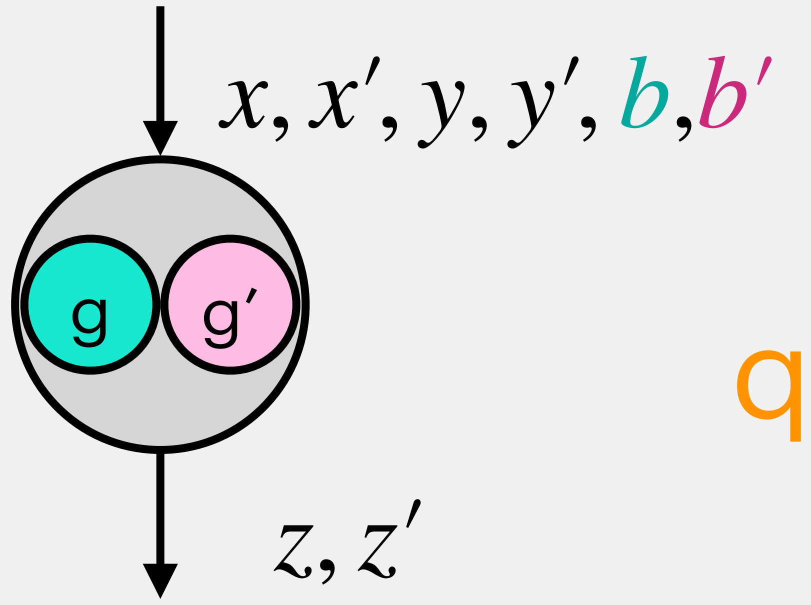
Quantifier-free



$$\boxed{b \wedge b'} \wedge \boxed{x = x' \wedge y = y'} \Rightarrow \boxed{z = z'}$$

activation variables | input equalities | output equality

Quantified Array



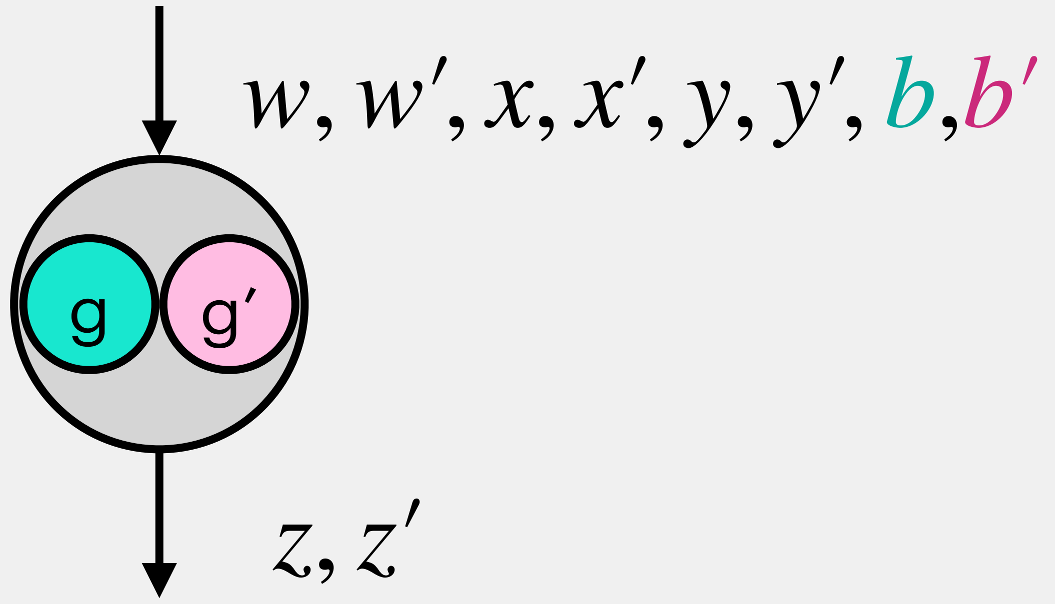
$$\forall \boxed{} . \boxed{} \wedge \boxed{} \wedge \boxed{} \Rightarrow \boxed{}$$

quantified indices | range | activation variables | equalities | cell property

Grammar Templates

Insight:
information flow involves **equalities** on subsets of corresponding components

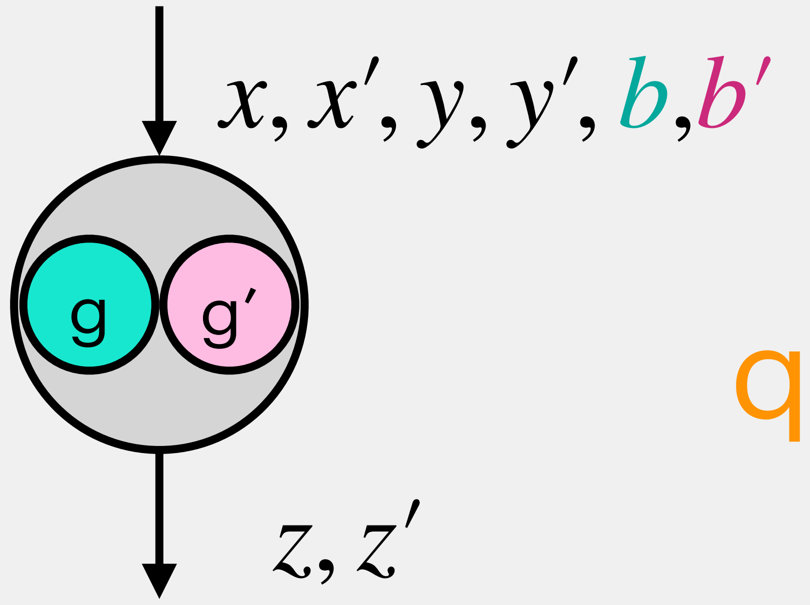
Quantifier-free



$$b \wedge b' \wedge x = x' \wedge y = y' \Rightarrow z = z'$$

activation variables ($b \wedge b'$) — input equalities ($x = x' \wedge y = y'$) — output equality ($z = z'$)

Quantified Array



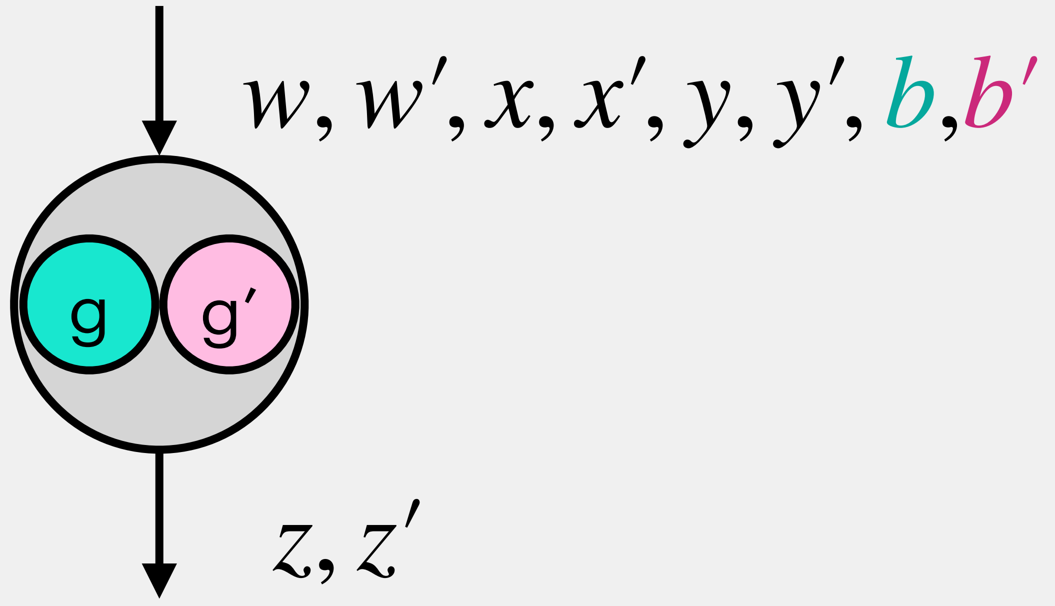
$$\forall\ i, i'.\ [\ \ ] \wedge [\ \ ] \wedge [\ \ ] \Rightarrow [\ \ ]$$

quantified indices (i, i') — range — activation variables — equalities — cell property

Grammar Templates

Insight:
information flow involves **equalities** on subsets of corresponding components

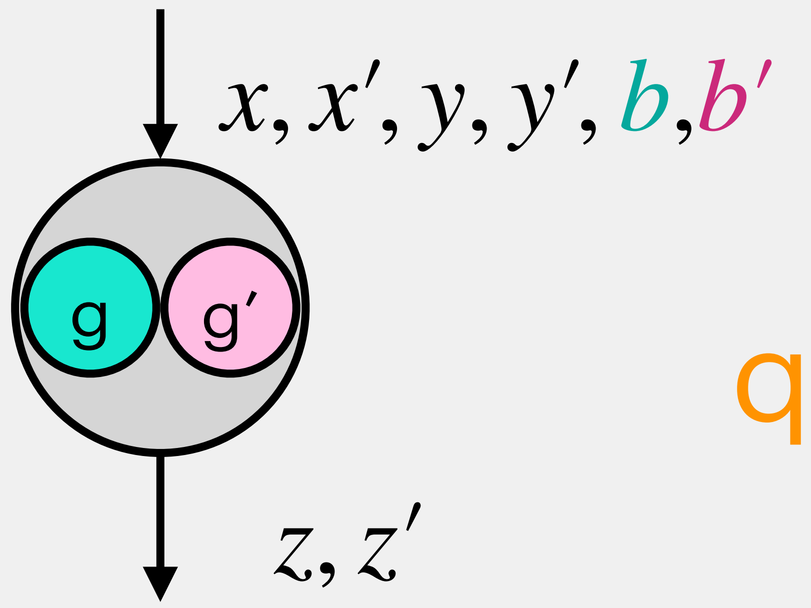
Quantifier-free



$$b \wedge b' \;\wedge\; x = x' \wedge y = y' \Rightarrow z = z'$$

activation variables | input equalities | output equality

Quantified Array



$$\forall\; i, i'.\; 0 \leq i < y \;\wedge\; \square \;\wedge\; \square \Rightarrow \square$$

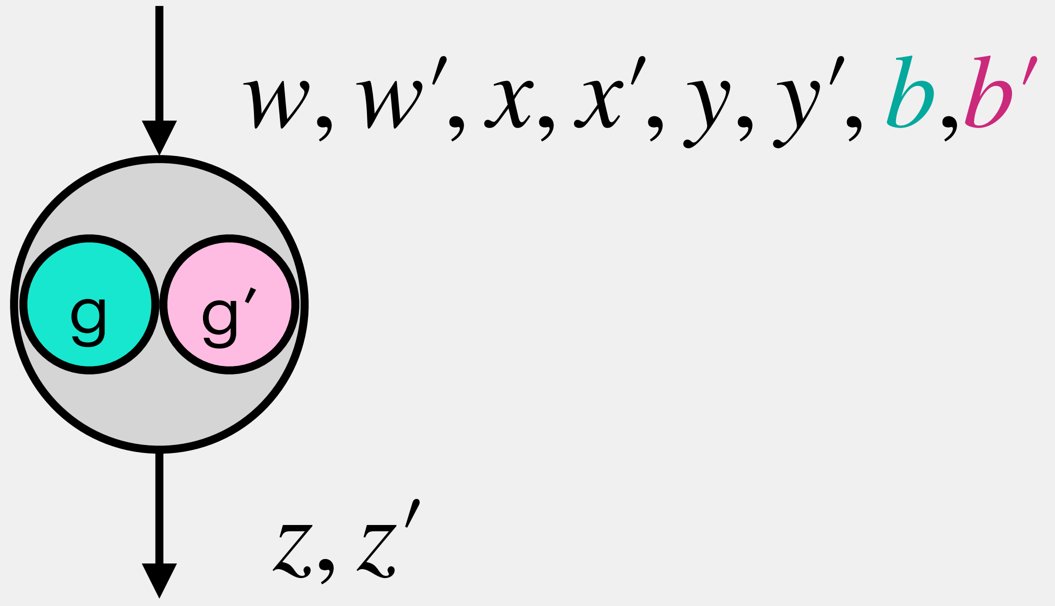
quantified indices | range | activation variables | equalities | cell property

Grammar Templates

Insight:

information flow involves **equalities** on subsets of corresponding components

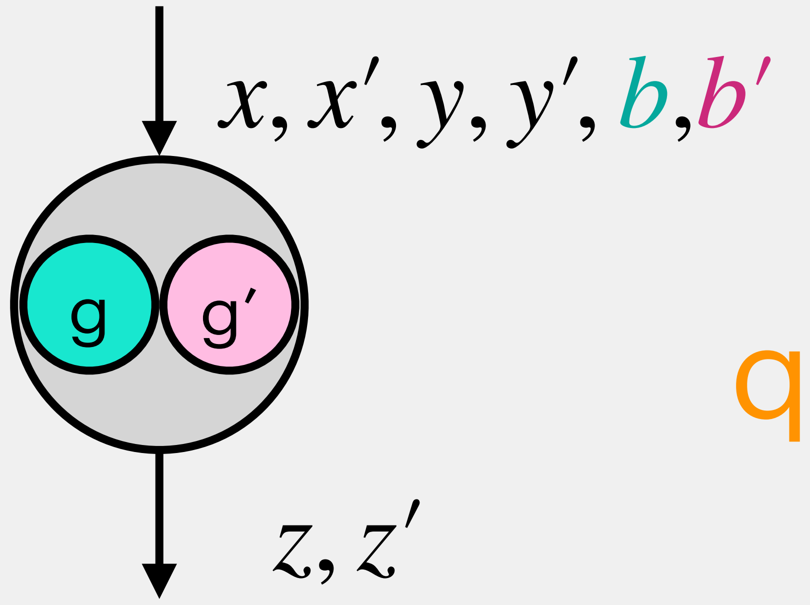
Quantifier-free



$$\boxed{b \wedge b'} \wedge \boxed{x = x' \wedge y = y'} \Rightarrow \boxed{z = z'}$$

activation variables — input equalities — output equality

Quantified Array



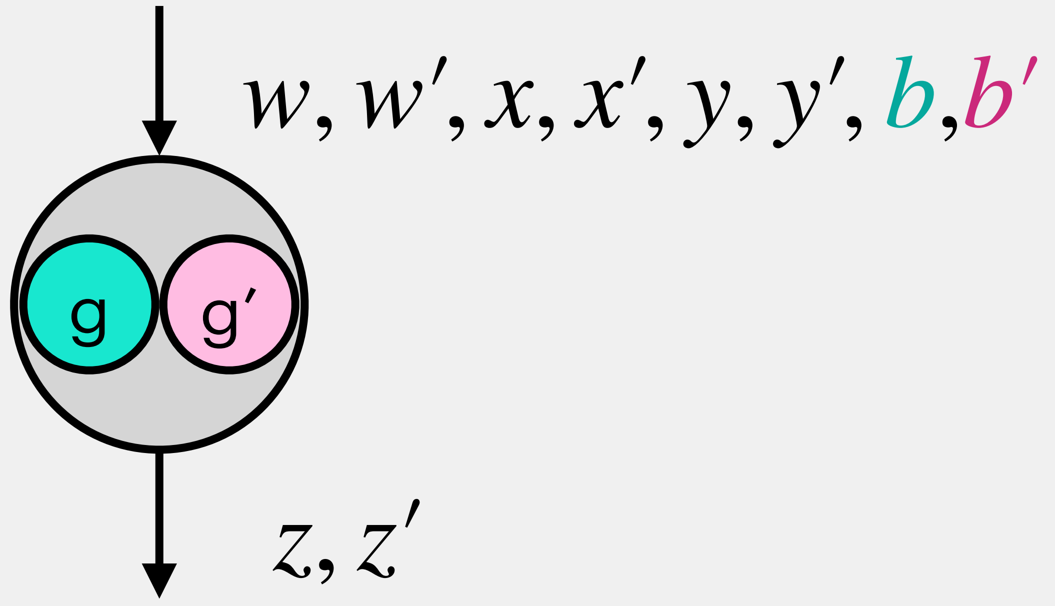
$$\forall\ \boxed{i, i'}\ .\ \boxed{0 \leq i < y} \wedge \boxed{b \wedge b'} \wedge \boxed{} \Rightarrow \boxed{}$$

quantified indices — range — activation variables — equalities — cell property

Grammar Templates

Insight:
information flow involves **equalities** on subsets of corresponding components

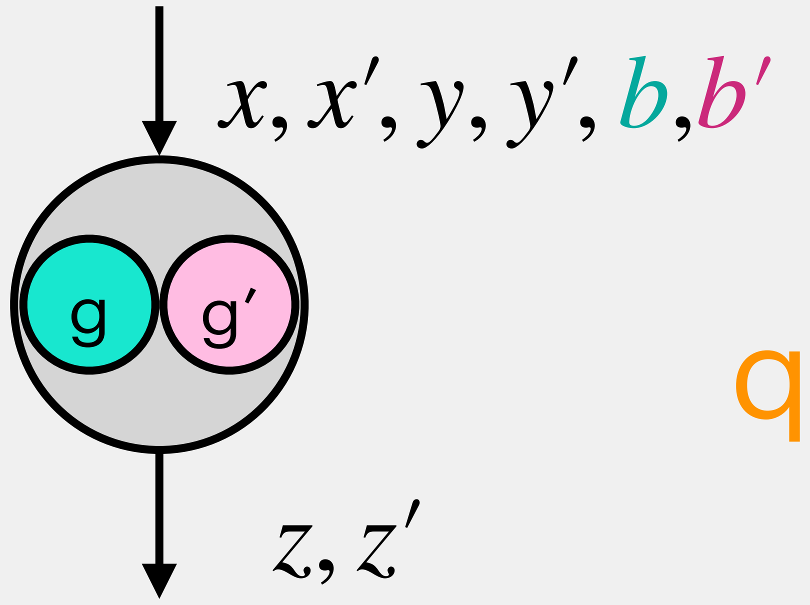
Quantifier-free



$$b \wedge b' \wedge x = x' \wedge y = y' \Rightarrow z = z'$$

activation variables ($b \wedge b'$) · input equalities ($x = x' \wedge y = y'$) · output equality ($z = z'$)

Quantified Array



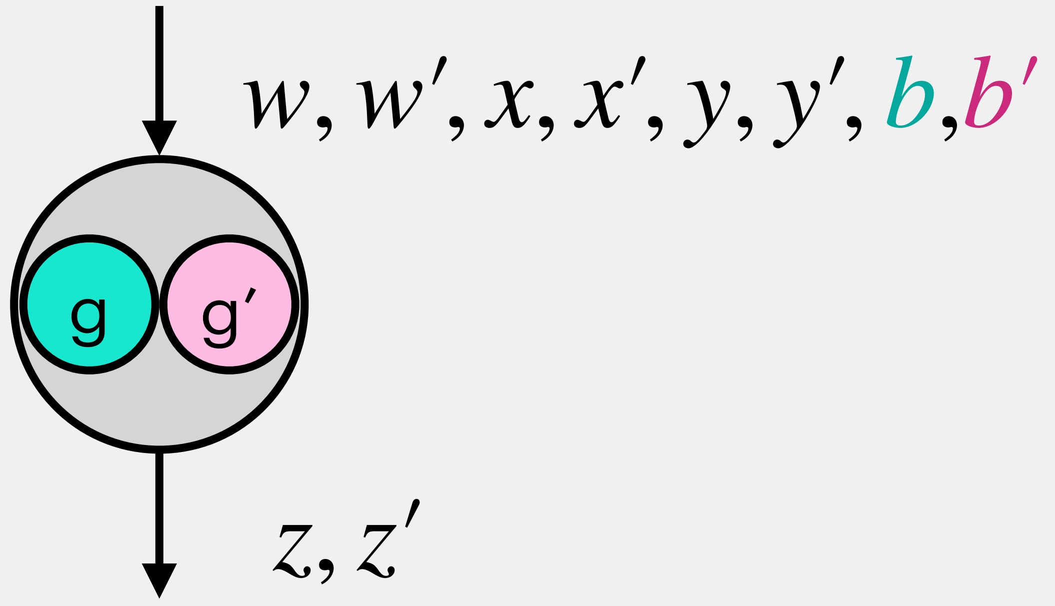
$$\forall\, i, i'.\; 0 \leq i < y \wedge b \wedge b' \wedge i = i' \wedge y = y' \Rightarrow \square$$

quantified indices (i, i') · range ($0 \leq i < y$) · activation variables ($b \wedge b'$) · equalities ($i = i' \wedge y = y'$) · cell property (empty box)

Grammar Templates

Insight:
information flow involves **equalities** on subsets of corresponding components

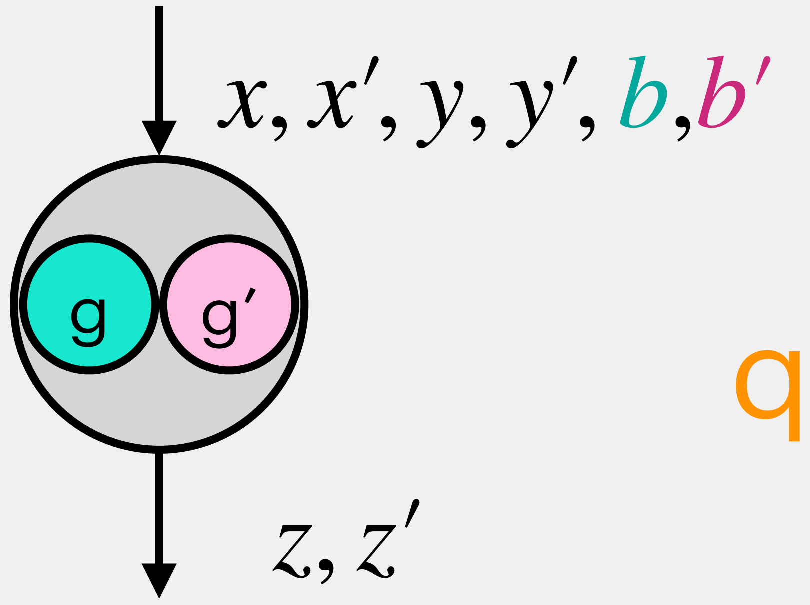
Quantifier-free



$$b \wedge b' \wedge x = x' \wedge y = y' \Rightarrow z = z'$$

activation variables ($b \wedge b'$) — input equalities ($x = x' \wedge y = y'$) — output equality ($z = z'$)

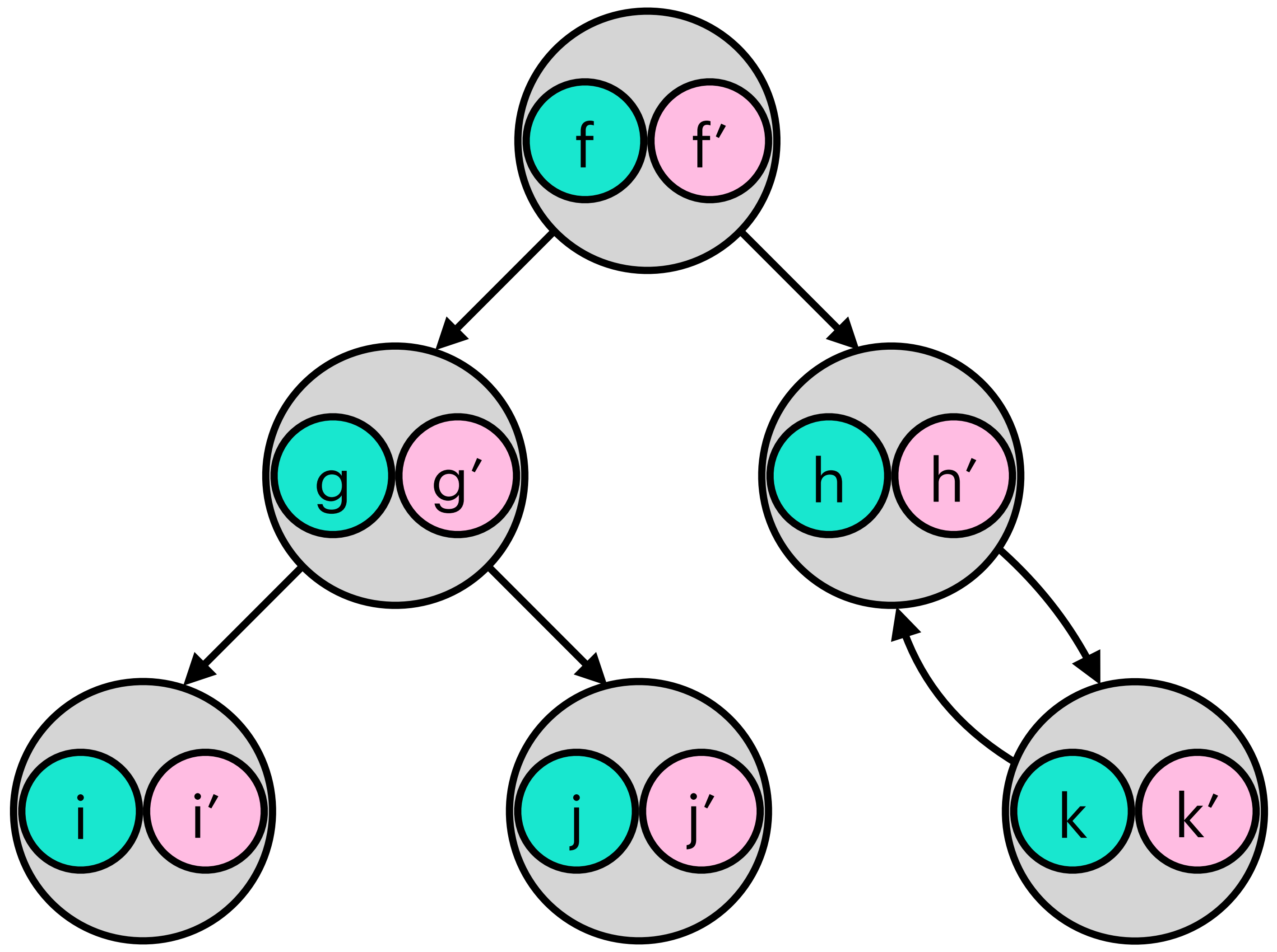
Quantified Array



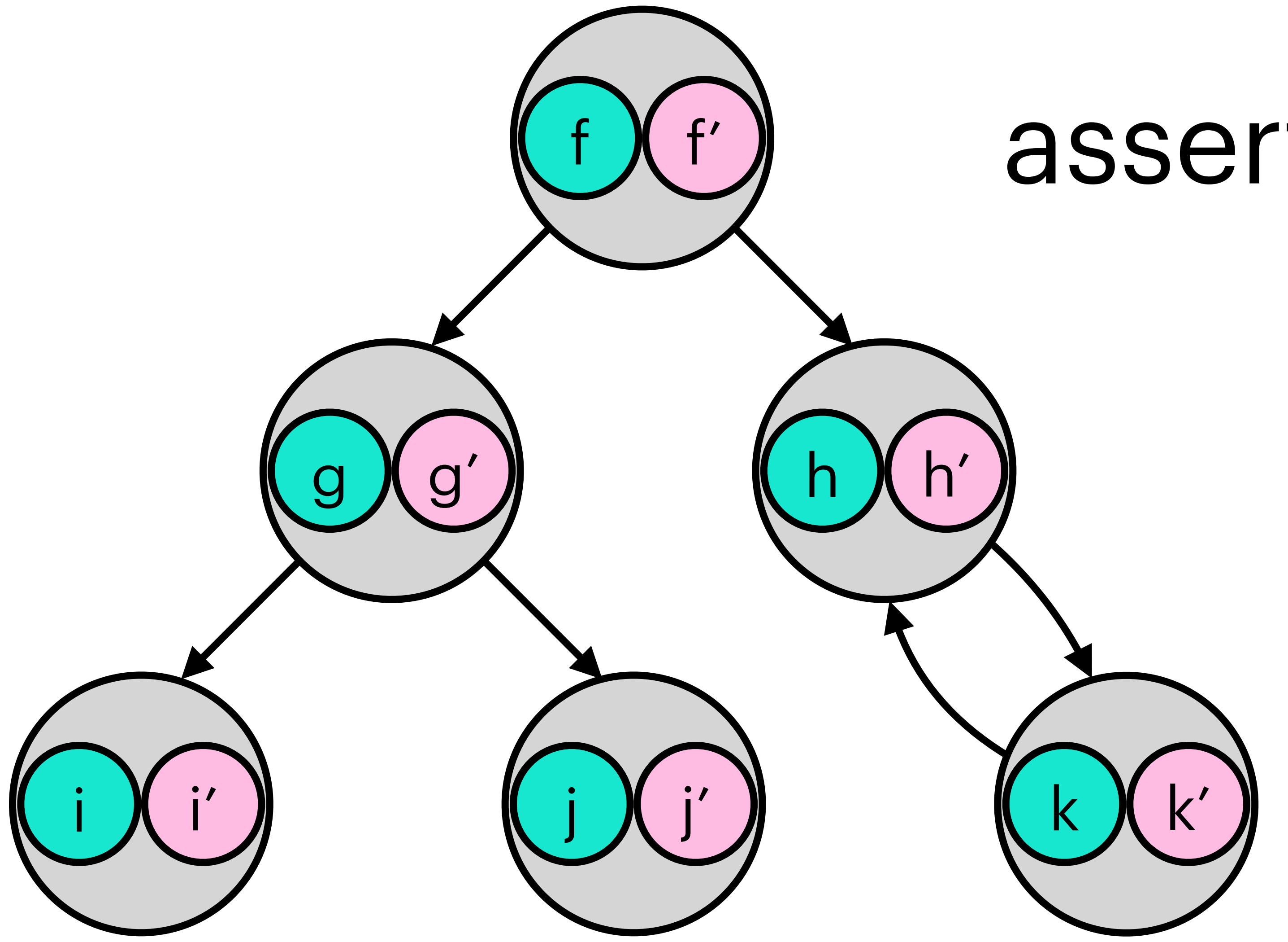
$$\forall\, i, i'.\ 0 \leq i < y \wedge b \wedge b' \wedge i = i' \wedge y = y' \Rightarrow z[i] = z'[i']$$

quantified indices (i, i') — range ($0 \leq i < y$) — activation variables ($b \wedge b'$) — equalities ($i = i' \wedge y = y'$) — cell property ($z[i] = z'[i']$)

Property-Directed Summaries



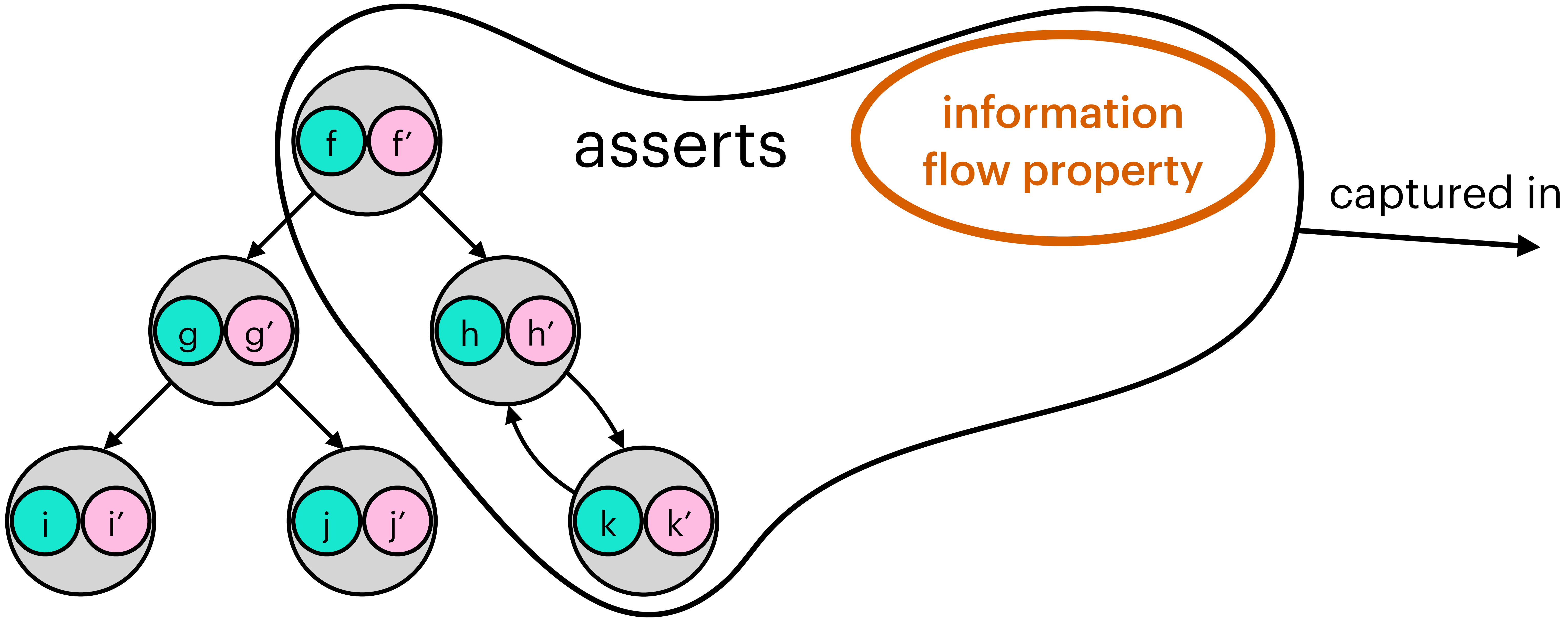
Property-Directed Summaries



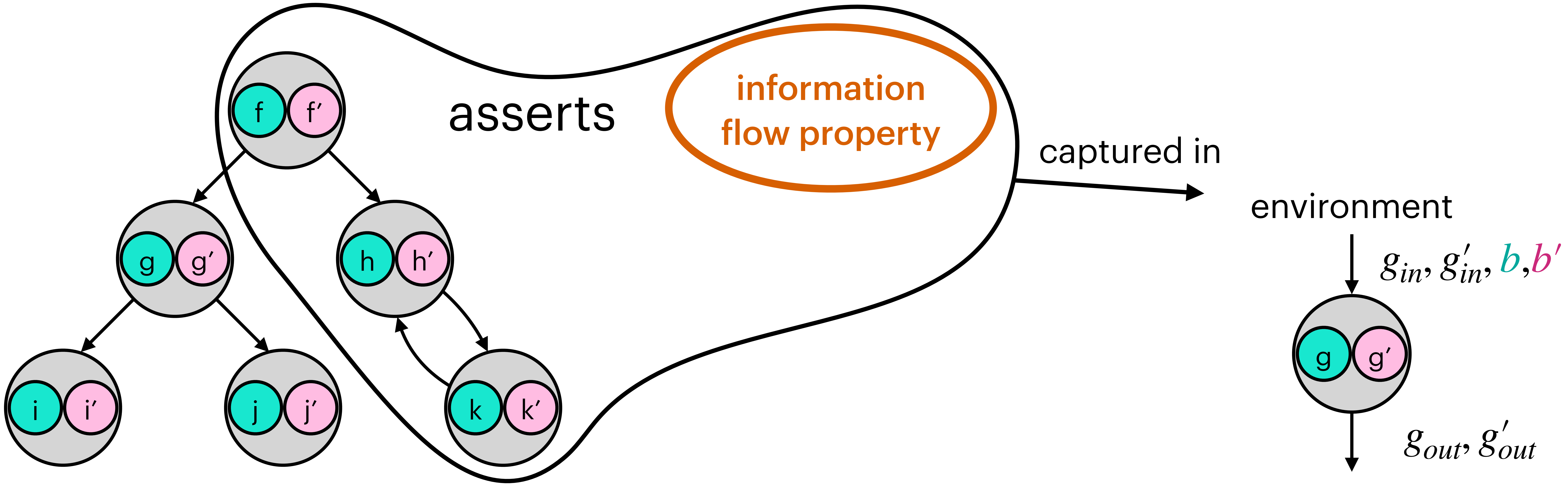
asserts

information
flow property

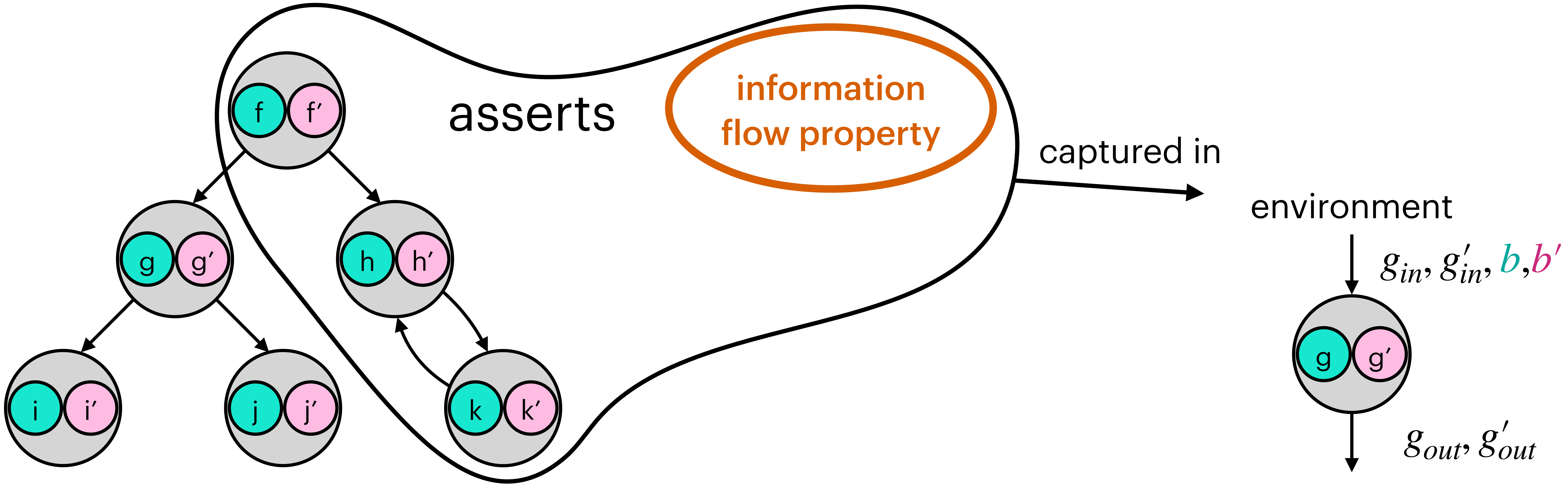
Property-Directed Summaries



Property-Directed Summaries



Property-Directed Summaries



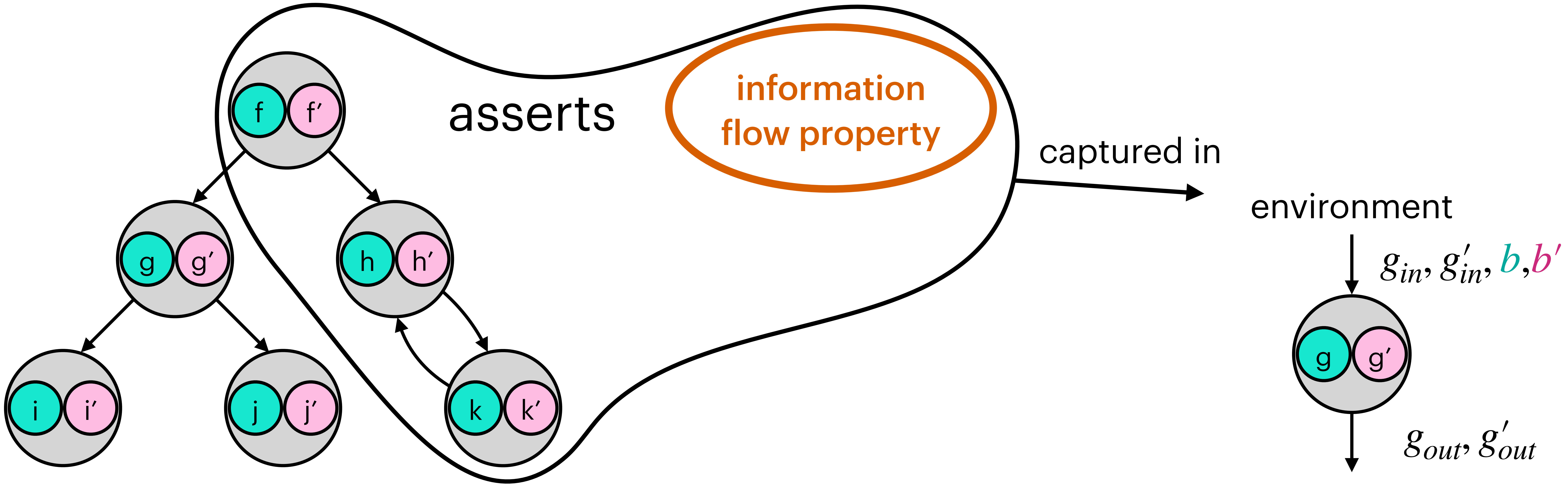
$$\boxed{b \wedge b'} \wedge \boxed{x = x' \wedge y = y'} \Rightarrow \boxed{z = z'}$$

activation variables

input equalities

output equalities

Property-Directed Summaries



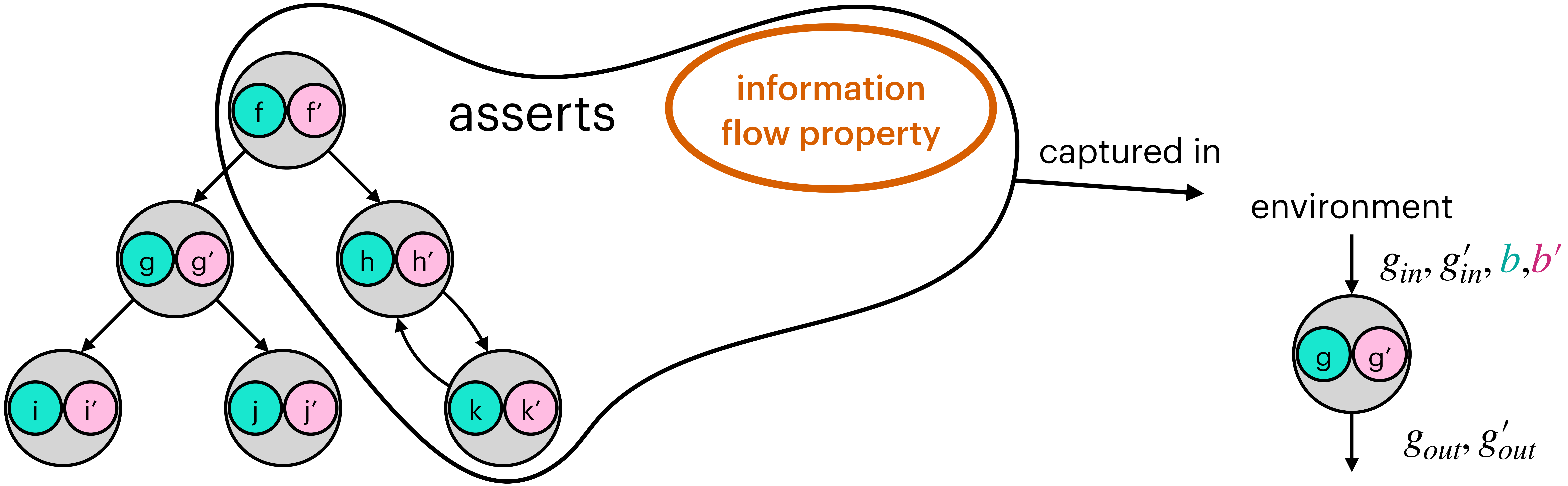
Conjuncts in environment $\wedge$ $\boxed{b \wedge b'}$ $\wedge$ $\boxed{x = x' \wedge y = y'}$ $\Rightarrow$ $\boxed{z = z'}$

activation variables

input equalities

output equalities

Property-Directed Summaries



Conjuncts in environment $\wedge$ $b \wedge b'$ $\wedge$ $x = x' \wedge y = y'$ $\Rightarrow$ $z = z'$

activation variables

input equalities

output equalities

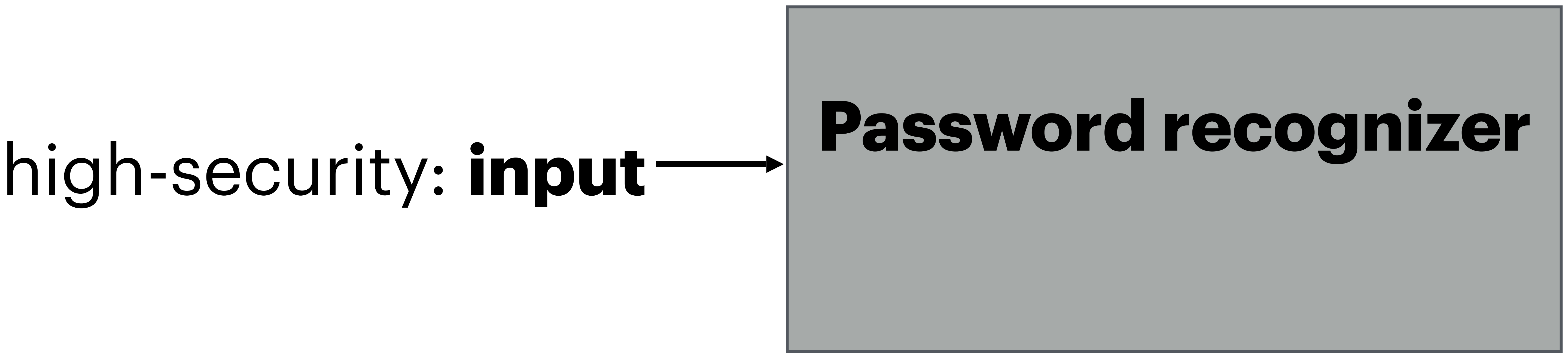
Useful for handling declassification

Declassification

Non-interference alone can be too restrictive

Declassification

Non-interference alone can be too restrictive



Declassification

Non-interference alone can be too restrictive

Declassification

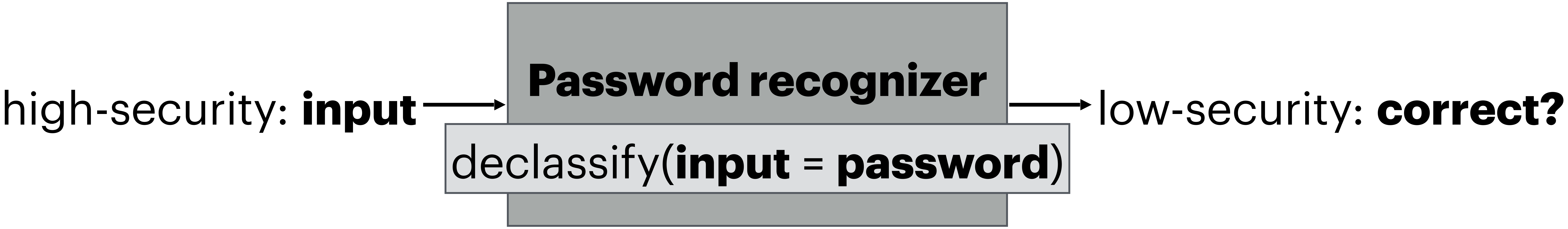
Non-interference alone can be too restrictive

Can *declassify* to allow *some* leakage

Declassification

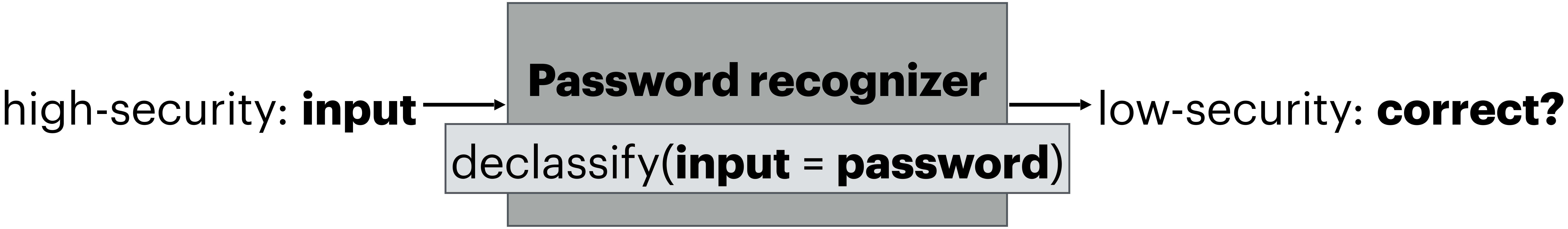
Non-interference alone can be too restrictive

Can *declassify* to allow *some* leakage



Declassification

Non-interference alone can be too restrictive
Can *declassify* to allow *some* leakage



Declassification can be captured in the environment

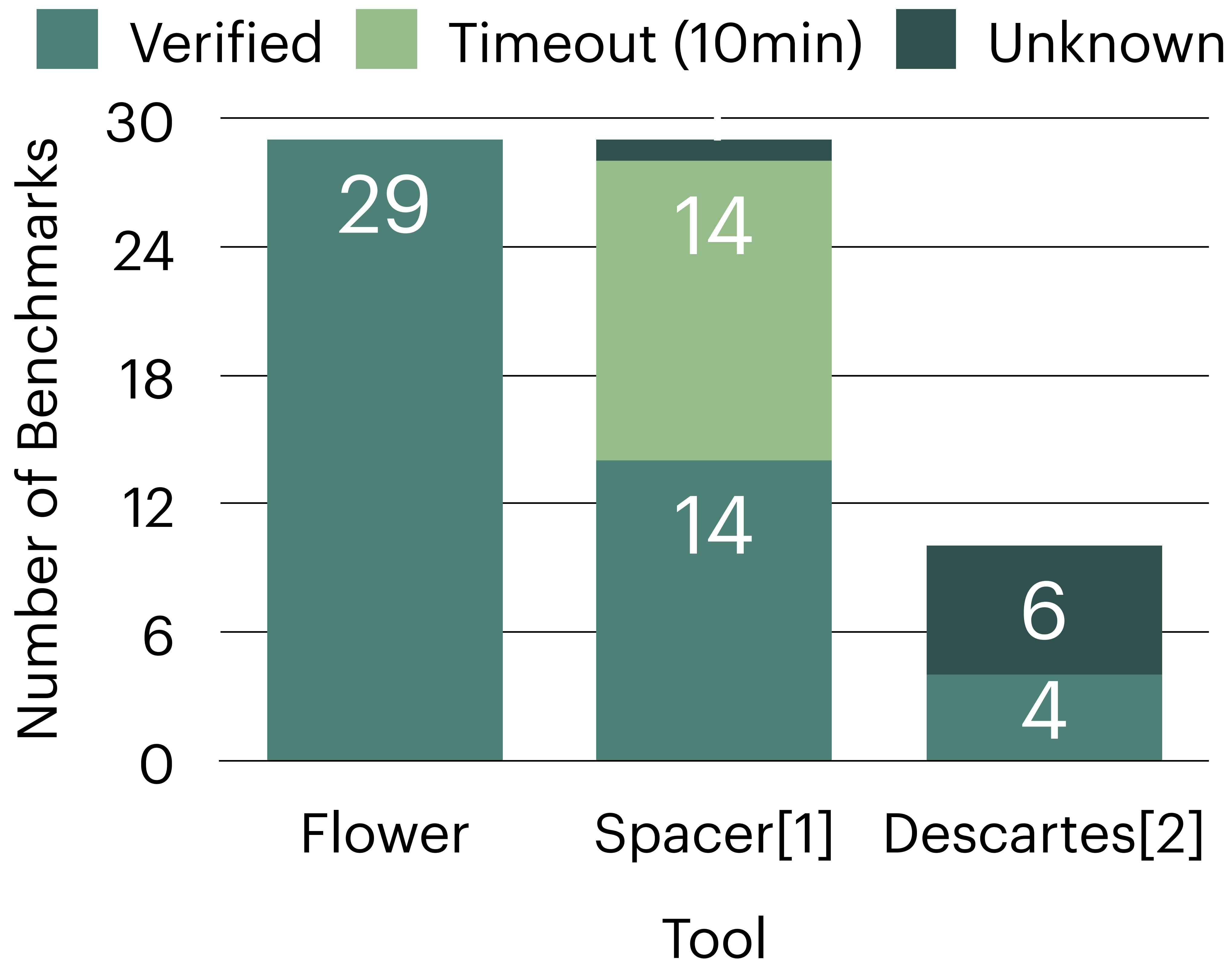
Experimental Results

Implemented in tool called Flower built on top of Clover

[1] SMT-based model-checking for recursive programs, Komuravelli et al. FMSD.'16
[2] Cartesian Hoare Logic, Sousa and Dillig, PLDI'16

Experimental Results

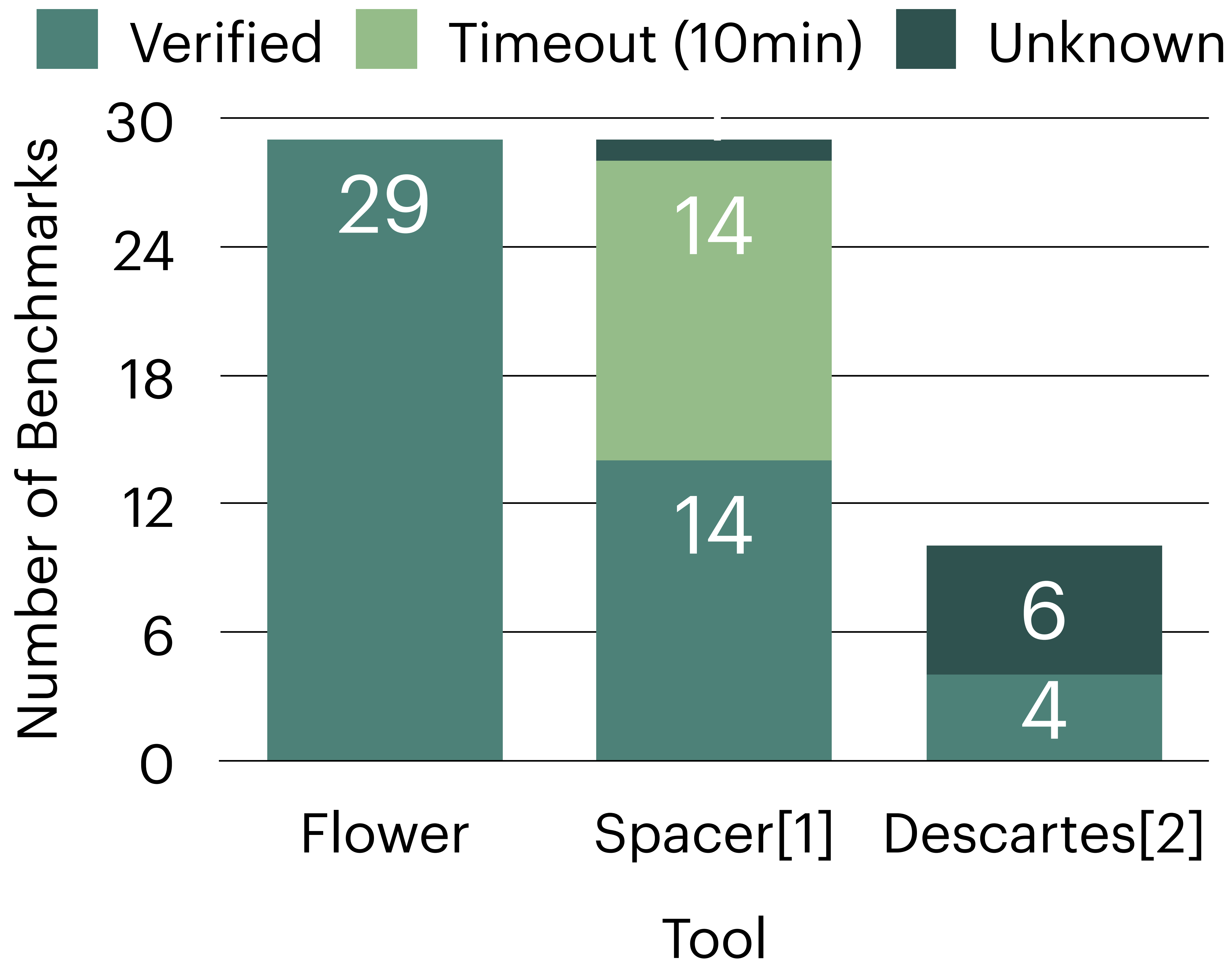
Implemented in tool called Flower built on top of Clover



[1] SMT-based model-checking for recursive programs, Komuravelli et al. FMSD.'16
[2] Cartesian Hoare Logic, Sousa and Dillig, PLDI'16

Experimental Results

Implemented in tool called Flower built on top of Clover

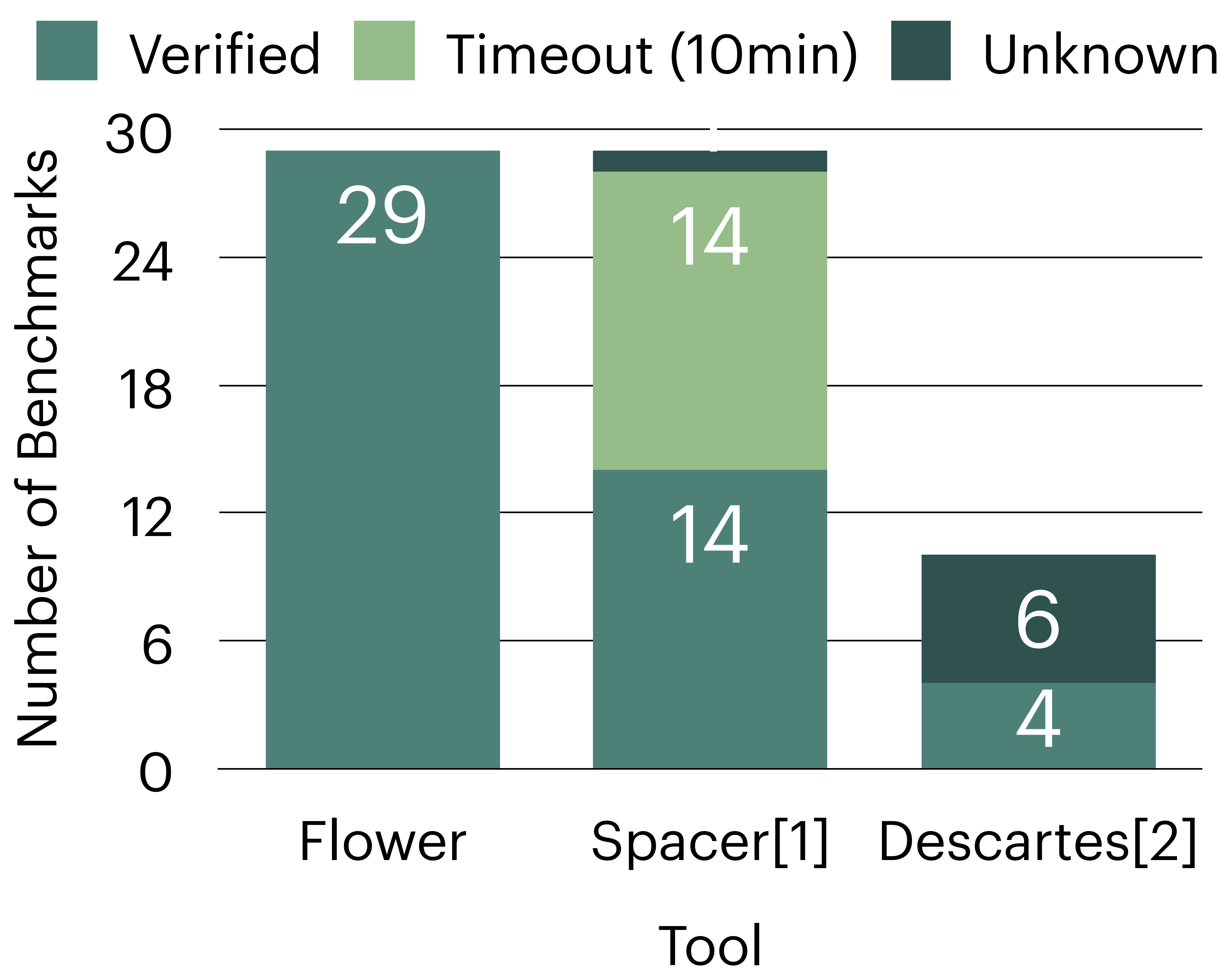


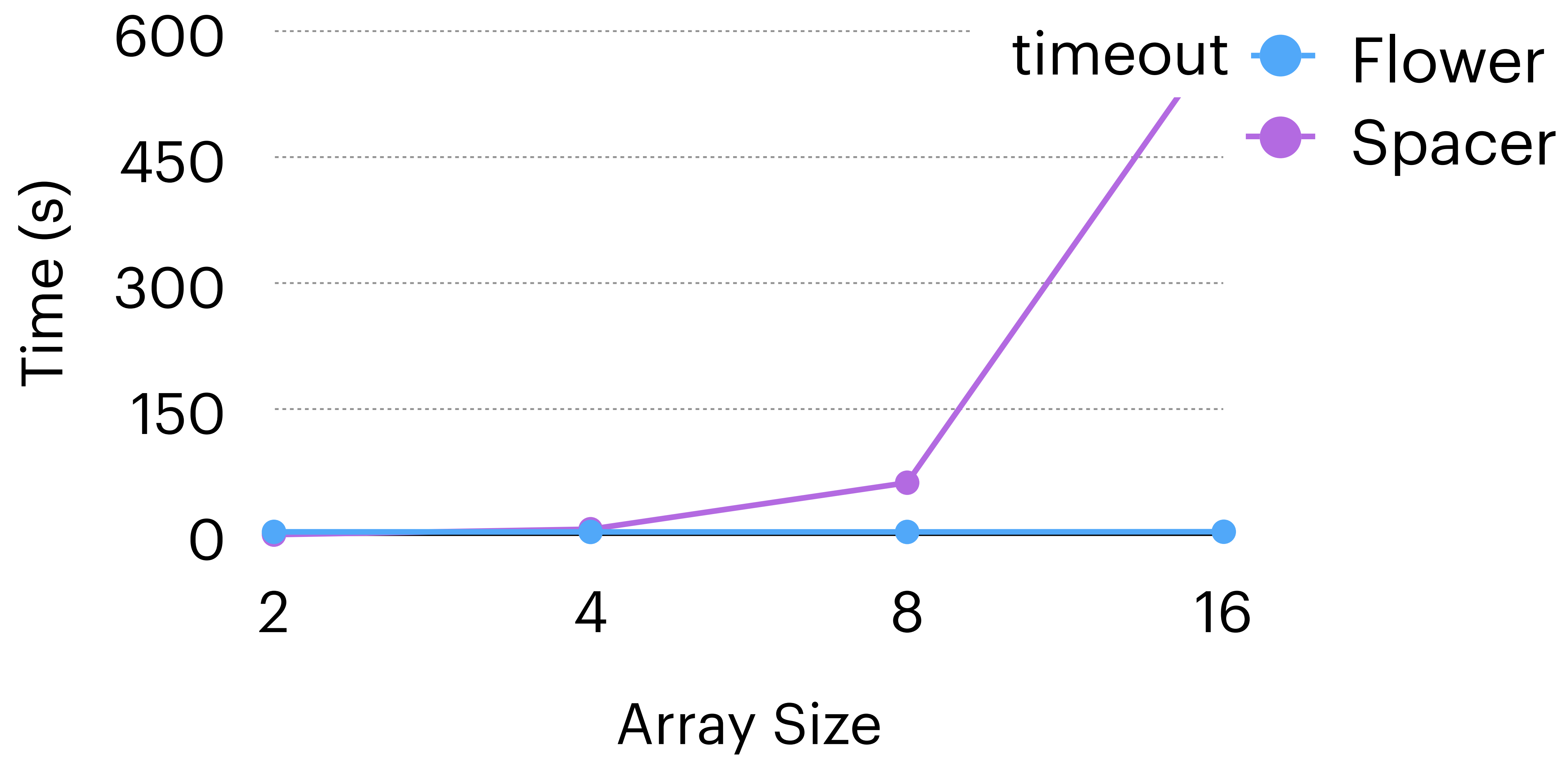
Unknown indicates inferred invariants too weak

[1] SMT-based model-checking for recursive programs, Komuravelli et al. FMSD.'16
[2] Cartesian Hoare Logic, Sousa and Dillig, PLDI'16

Experimental Results

Implemented in tool called Flower built on top of Clover



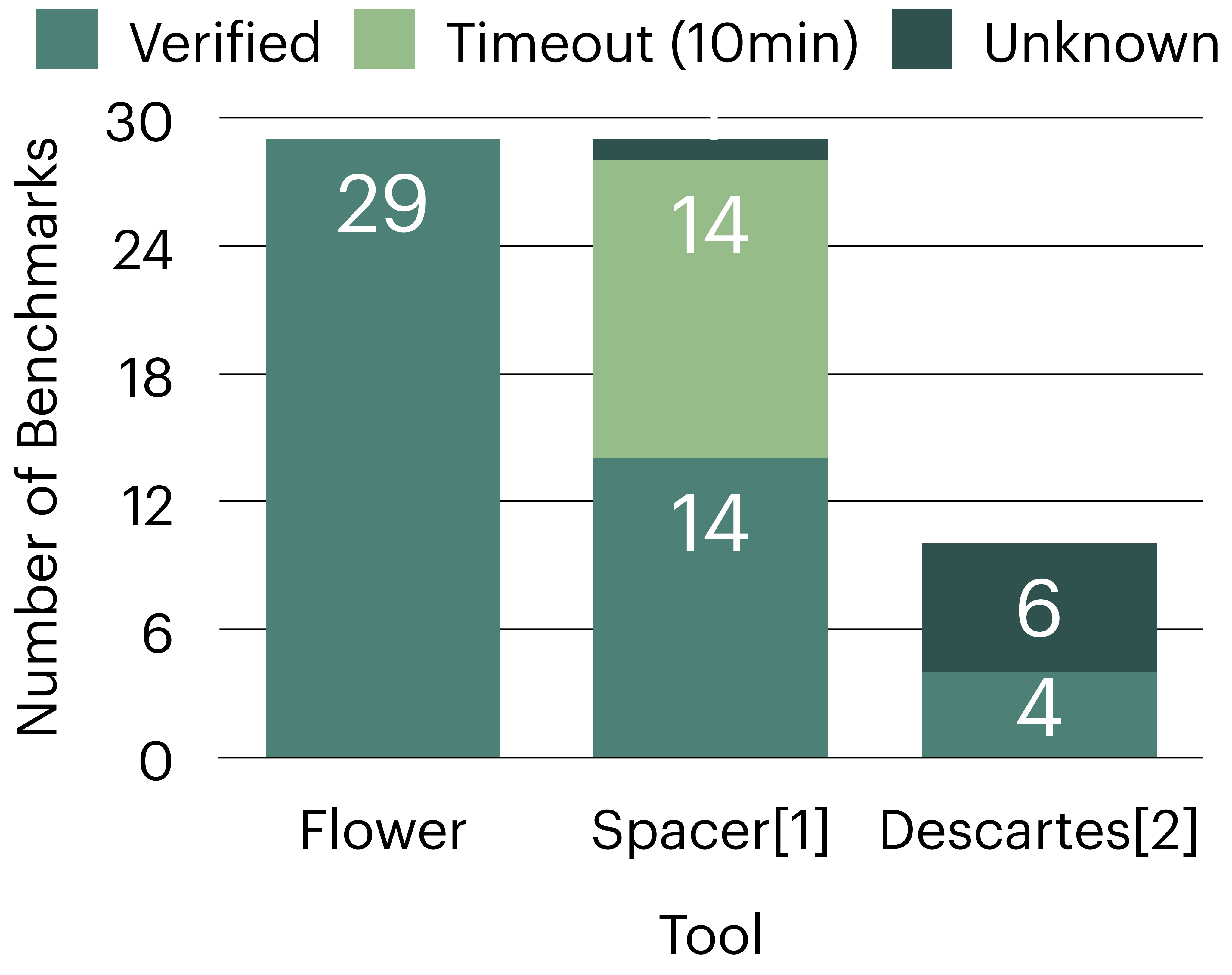


Unknown indicates inferred invariants too weak

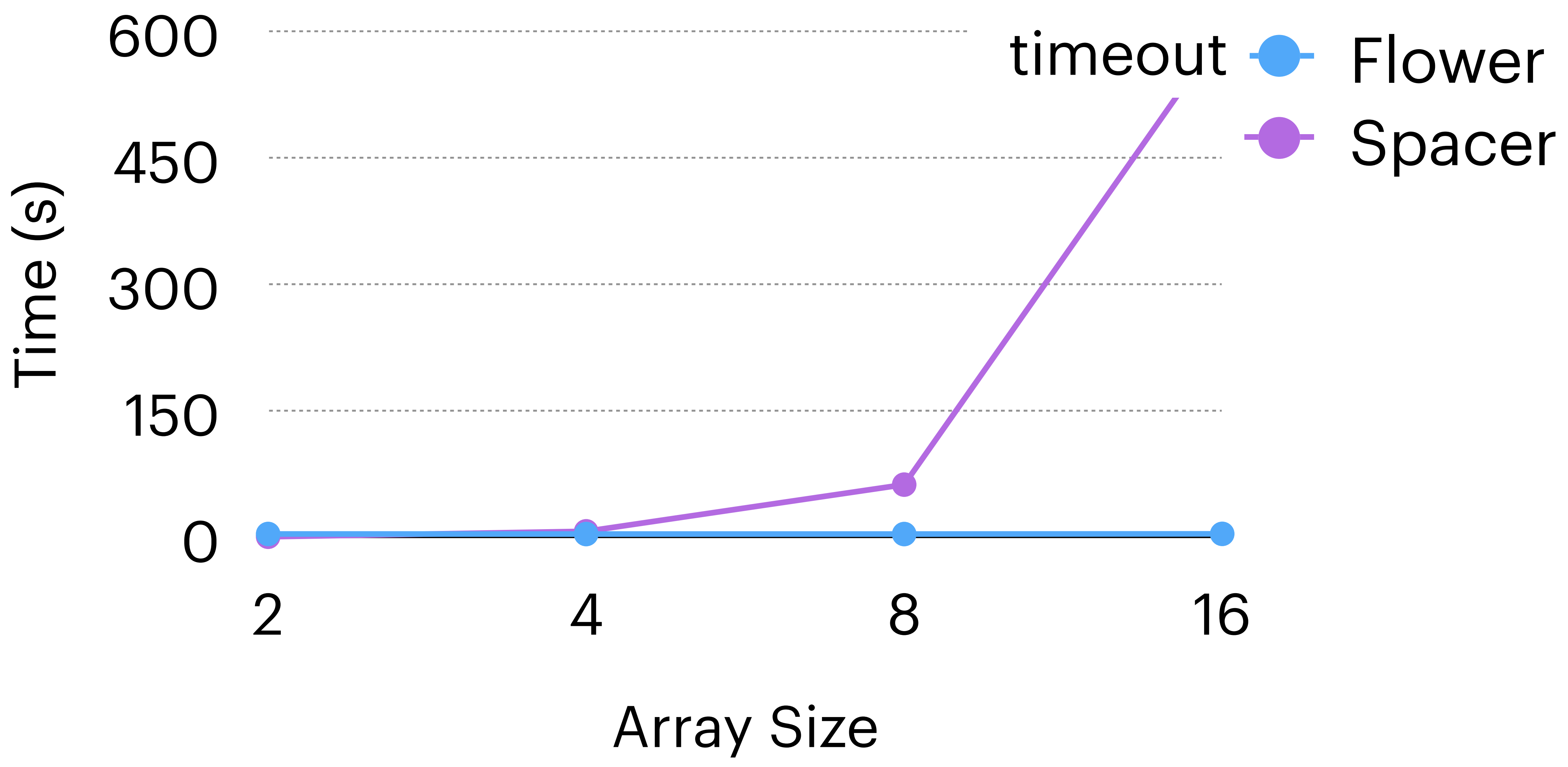
[1] SMT-based model-checking for recursive programs, Komuravelli et al. FMSD.'16
[2] Cartesian Hoare Logic, Sousa and Dillig, PLDI'16

Experimental Results

Implemented in tool called Flower built on top of Clover



Unknown indicates inferred invariants too weak



Parametrizable benchmark shows array size does not affect Flower's performance because of quantified template

[1] SMT-based model-checking for recursive programs, Komuravelli et al. FMSD.'16
[2] Cartesian Hoare Logic, Sousa and Dillig, PLDI'16

Related Work

Relational Program Verification	Information-Flow Checking

Related Work

Relational Program Verification

Information-Flow Checking

Non-modular approaches

[Barthe et al., CSFW'04] [Terauchi and Aiken, SAS'05] [Banerjee et al., FSTTCS'16]

[Barthe et al., FM'11] [Sousa and Dillig, PLDI'16] [Beringer, ITP'11]

Related Work

Relational Program Verification

Information-Flow Checking

Non-modular approaches

[Barthe et al., CSFW'04] [Terauchi and Aiken, SAS'05] [Banerjee et al., FSTTCS'16]

[Barthe et al., FM'11] [Sousa and Dillig, PLDI'16] [Beringer, ITP'11]

Modular, non-automated

[Eilers et al., ESOP'18]

Related Work

Relational Program Verification

Information-Flow Checking

Non-modular approaches

[Barthe et al., CSFW'04] [Terauchi and Aiken, SAS'05] [Banerjee et al., FSTTCS'16]

[Barthe et al., FM'11] [Sousa and Dillig, PLDI'16] [Beringer, ITP'11]

Modular, non-automated

[Eilers et al., ESOP'18]

Security-Type Systems

[Denning and Denning, Commun. ACM, 1977]
[Volpano et al., 1996]

Related Work

Relational Program Verification

Information-Flow Checking

Non-modular approaches

[Barthe et al., CSFW'04] [Terauchi and Aiken, SAS'05] [Banerjee et al., FSTTCS'16]

[Barthe et al., FM'11] [Sousa and Dillig, PLDI'16] [Beringer, ITP'11]

Modular, non-automated

[Eilers et al., ESOP'18]

Security-Type Systems

[Denning and Denning, Commun. ACM, 1977]
[Volpano et al., 1996]

Dynamic Taint Analysis

[Sarwar et al., SECRYPT'13]

Related Work

Relational Program Verification

Information-Flow Checking

Non-modular approaches

[Barthe et al., CSFW'04] [Terauchi and Aiken, SAS'05] [Banerjee et al., FSTTCS'16]

[Barthe et al., FM'11] [Sousa and Dillig, PLDI'16] [Beringer, ITP'11]

Modular, non-automated

[Eilers et al., ESOP'18]

Security-Type Systems

[Denning and Denning, Commun. ACM, 1977]

[Volpano et al., 1996]

Dynamic Taint Analysis

[Sarwar et al., SECRYPT'13]

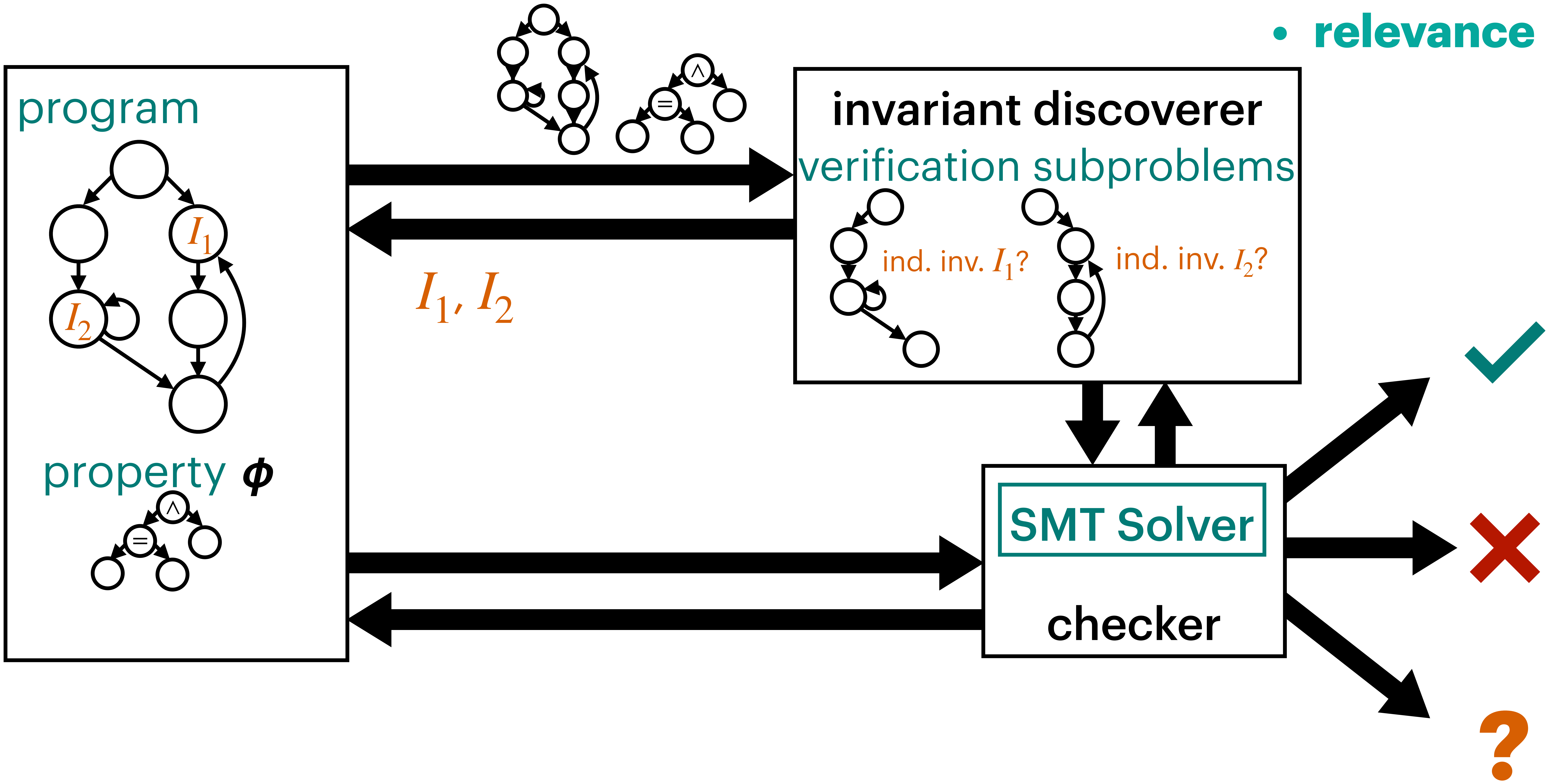
Syntax-Guided Synthesis for Quantified Array Invariants

[Fedyukovich et al., CAV'19]

Structure and Syntax

Structural info about programs and properties can help with:

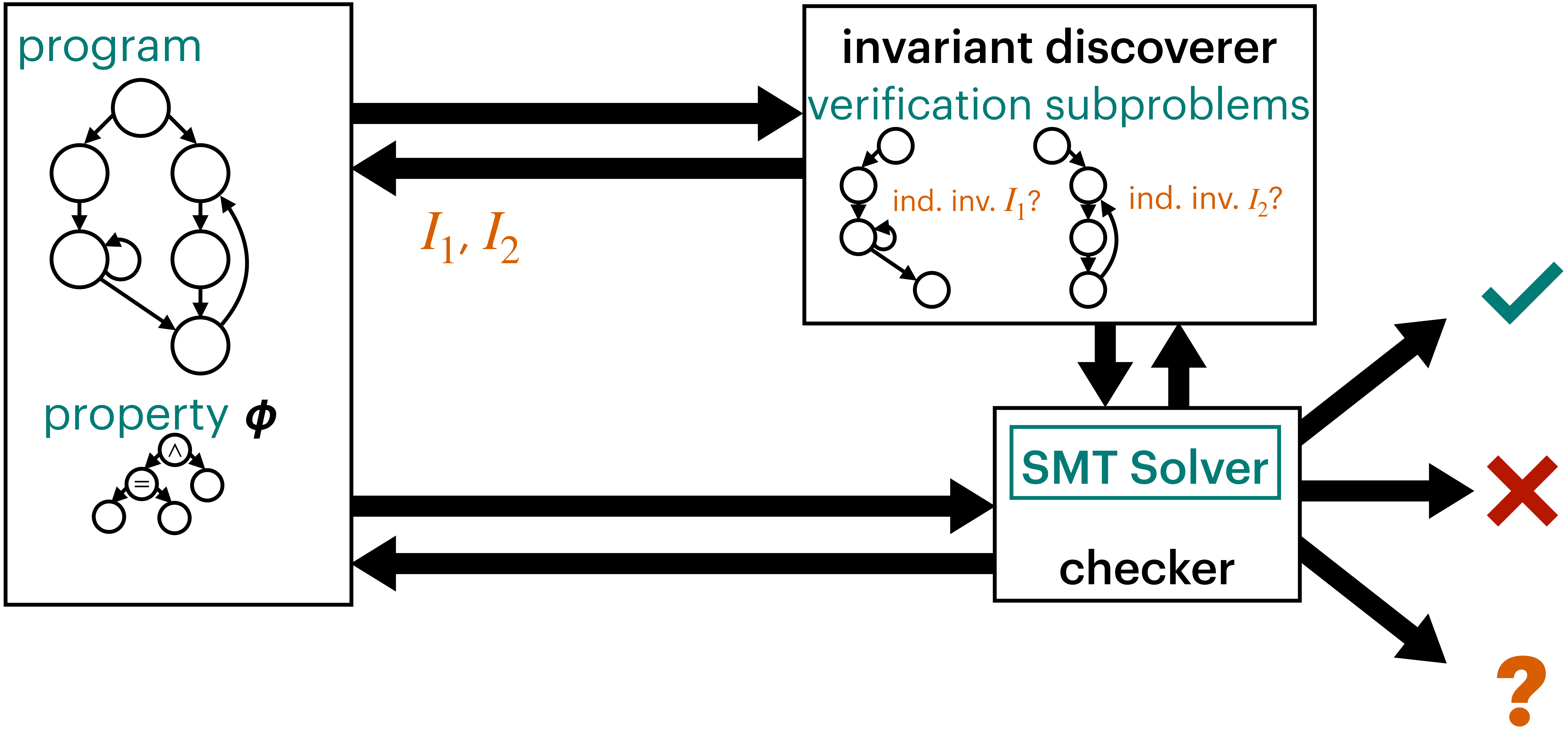
- **performance**
- **scalability**
- **relevance**



Structure and Syntax

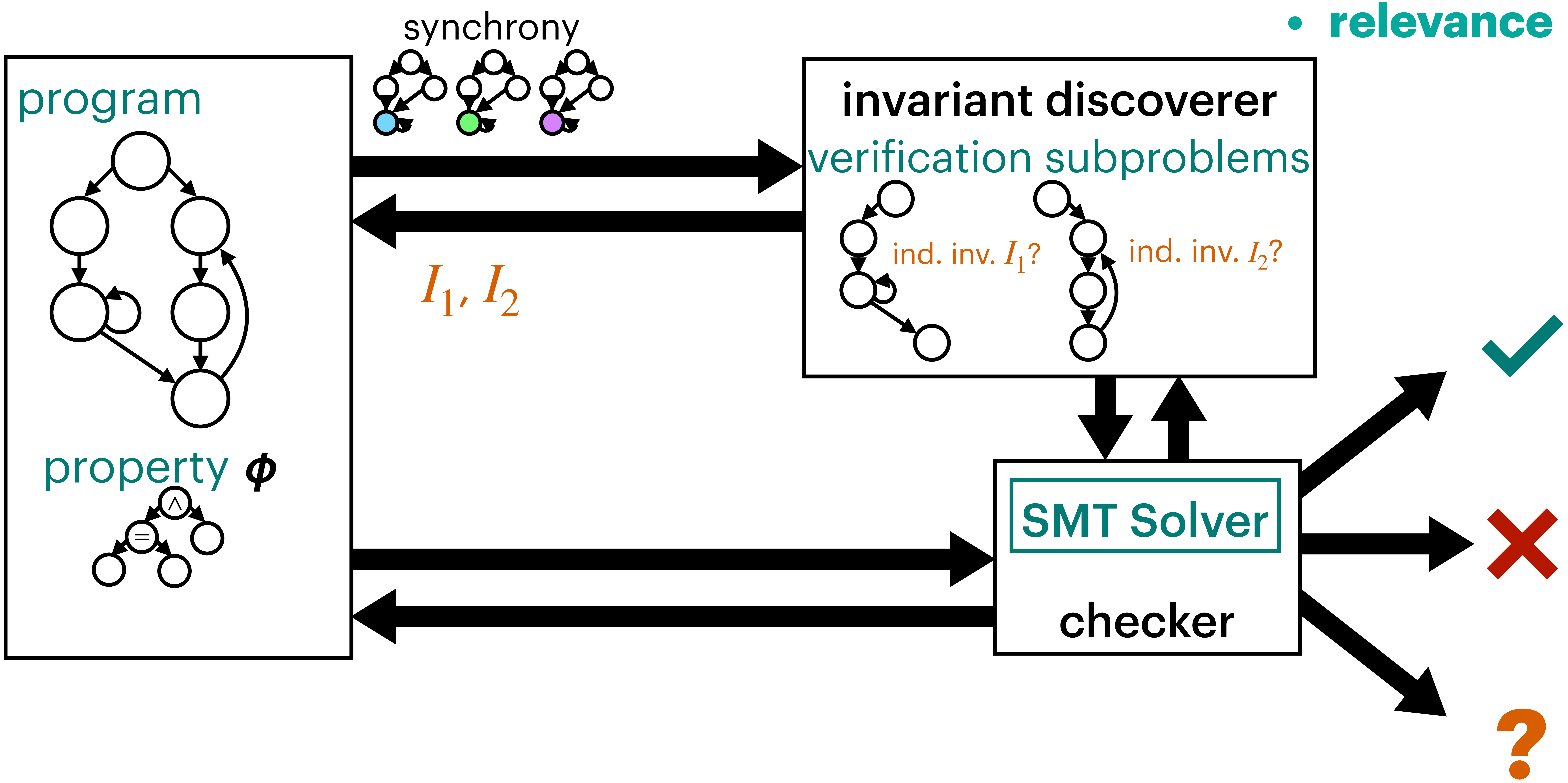
Structural info about programs and properties can help with:

- performance
- scalability
- relevance



Structure and Syntax

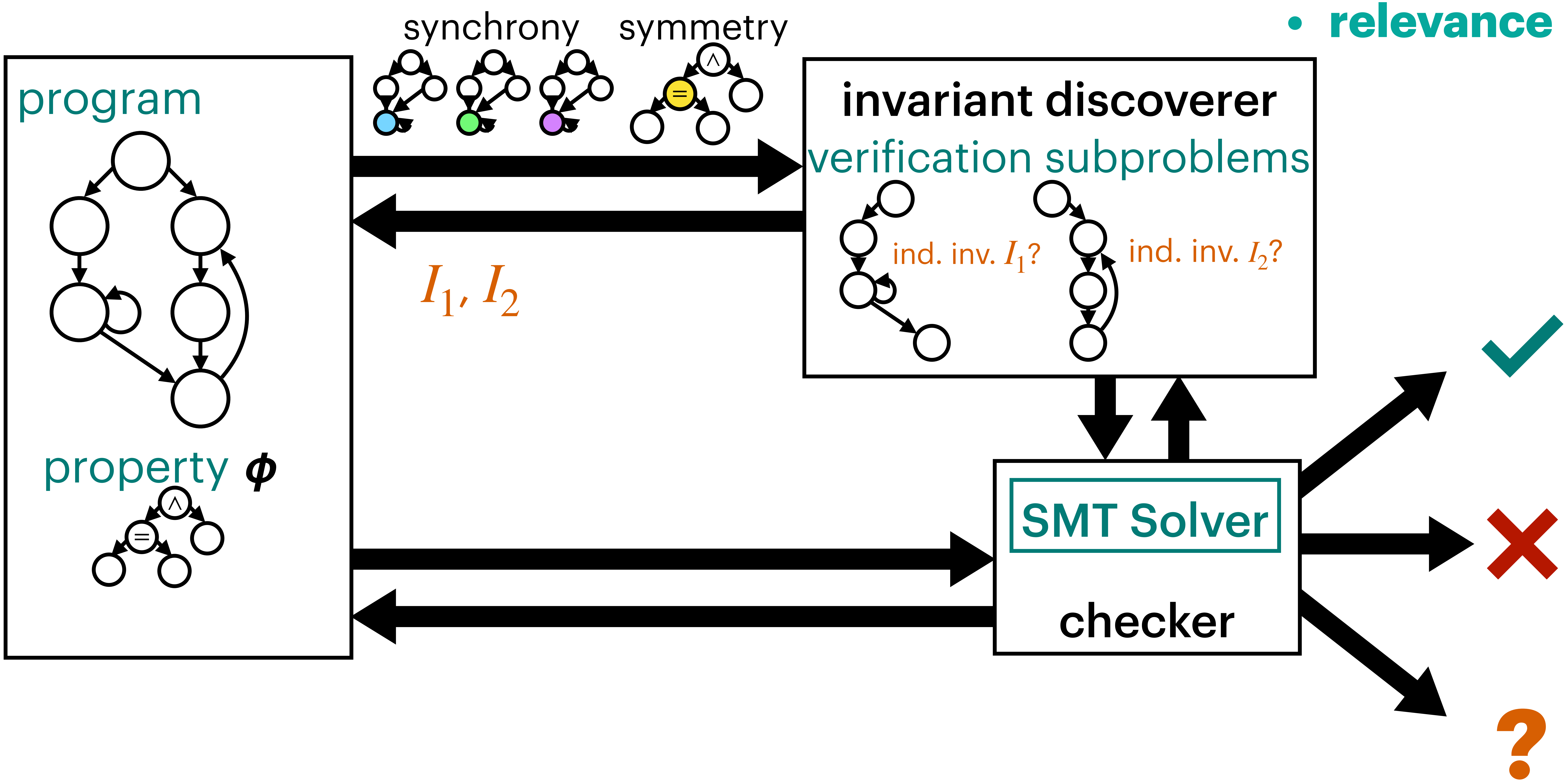
Structural info about programs and properties can help with:
- **performance**
- **scalability**
- **relevance**



Structure and Syntax

Structural info about programs and properties can help with:

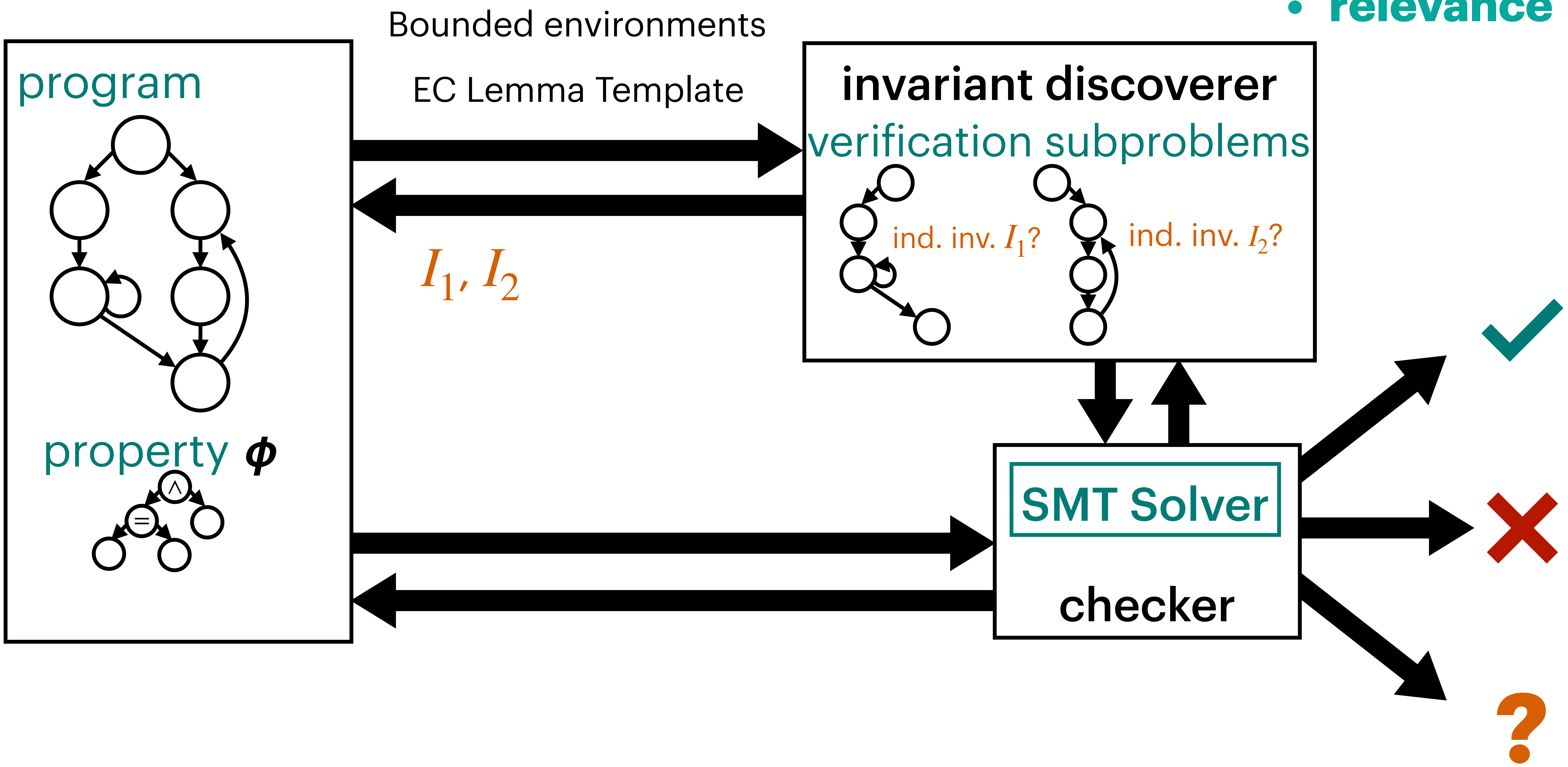
- **performance**
- **scalability**
- **relevance**



Structure and Syntax

Structural info about programs and properties can help with:

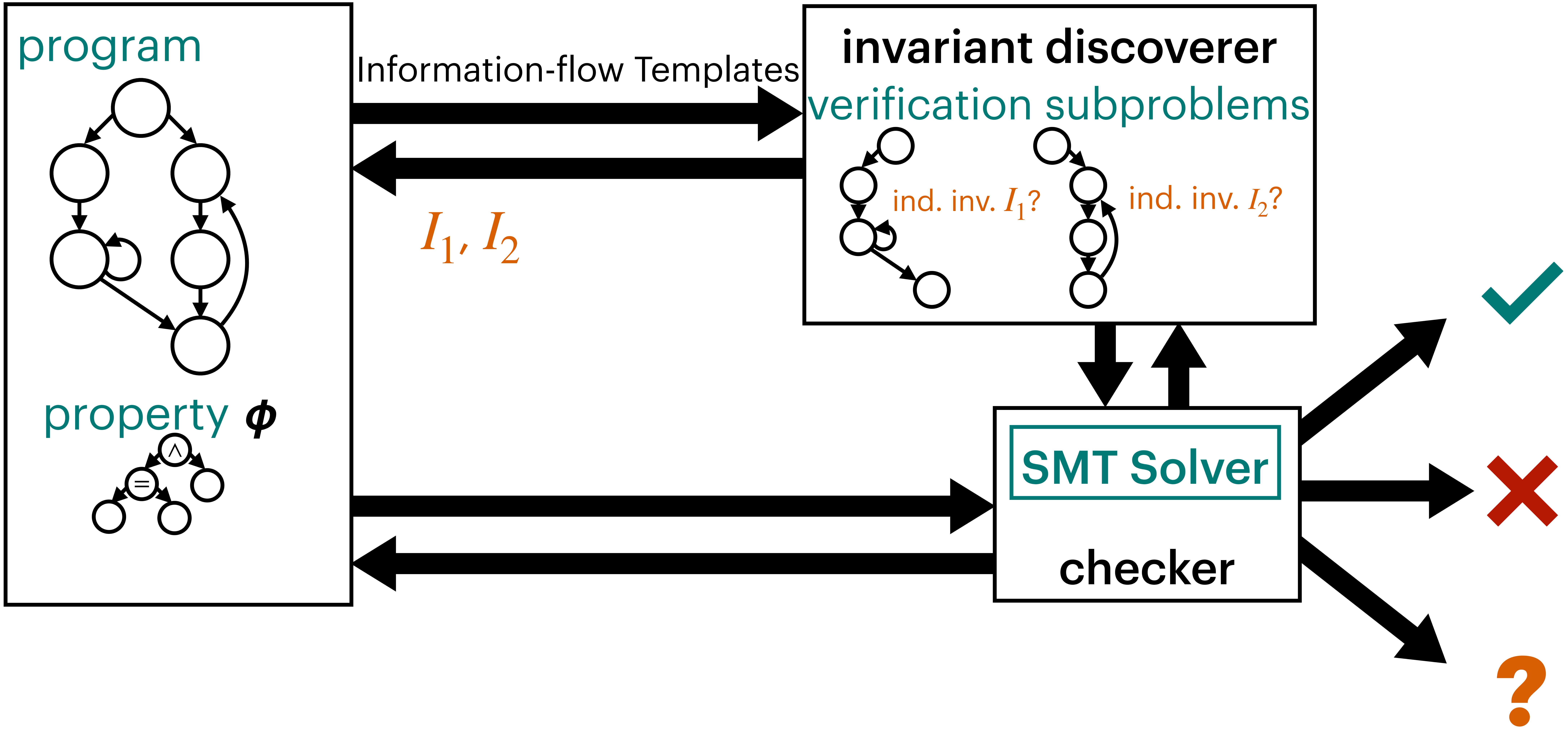
- **performance**
- **scalability**
- **relevance**



Structure and Syntax

Structural info about programs and properties can help with:

- **performance**
- **scalability**
- **relevance**



Contributions

How to exploit **structure** of both **programs** and **properties**
to infer and leverage **invariants** that improve **scalability** and **performance**
in SMT-based automated verification.

Future Work

I. *k*-safety Verification
Cartesian Hoare Logic [2]

Symmetry-breaking for Constrained Horn Clauses

II. Interprocedural Program Verification
Constrained Horn Clauses

III. Information-Flow Verification
Constrained Horn Clauses

Handle heaps: Constrained Horn Clauses + heaps [1]

[1] Towards an SMT-Lib Theory of Heap, Esen and Rümmer, IJCAR'20
[2] Cartesian Hoare Logic, Sousa and Dillig, PLDI'16

Extra slides

Invariants

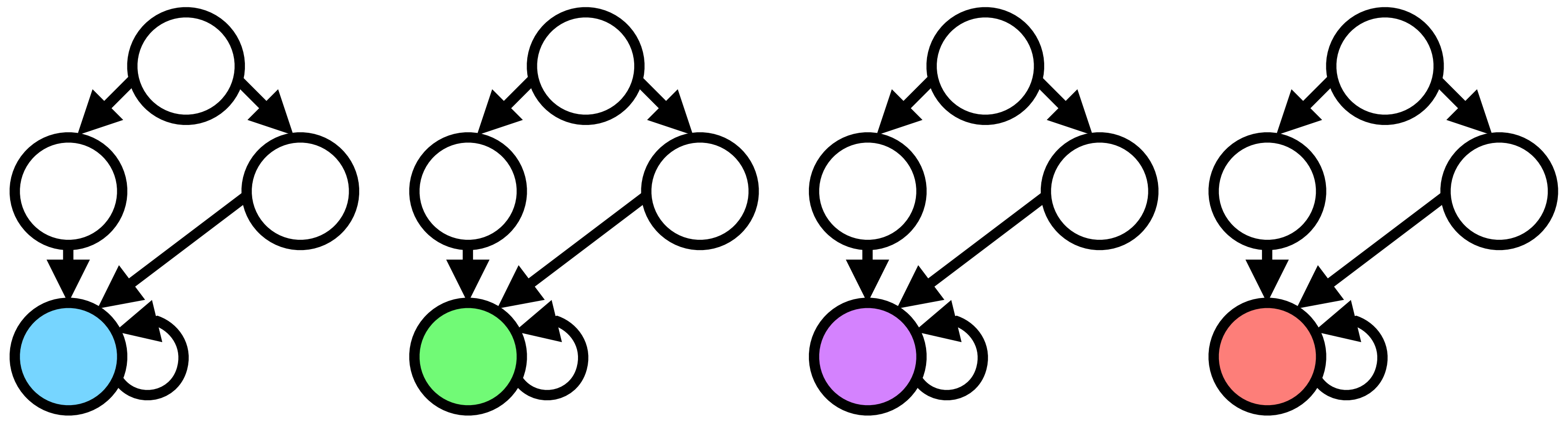
How to make it easy to **infer** relational properties with symmetries?

invariant synthesizer

Invariants

How to make it easy to **infer** relational properties with symmetries?

synchronize (align) structurally similar parts
(e.g., control-flow graph nodes)

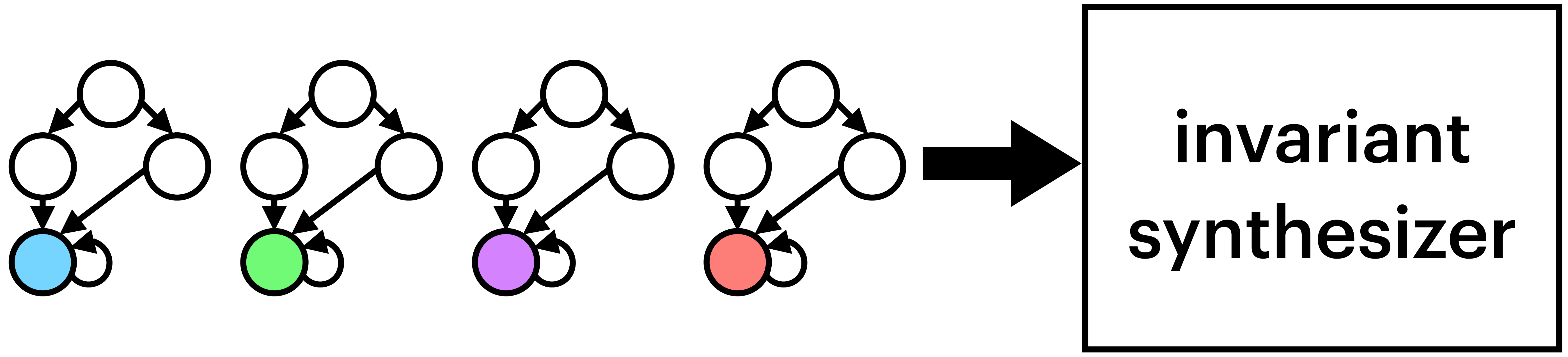


invariant synthesizer

Invariants

How to make it easy to **infer** relational properties with symmetries?

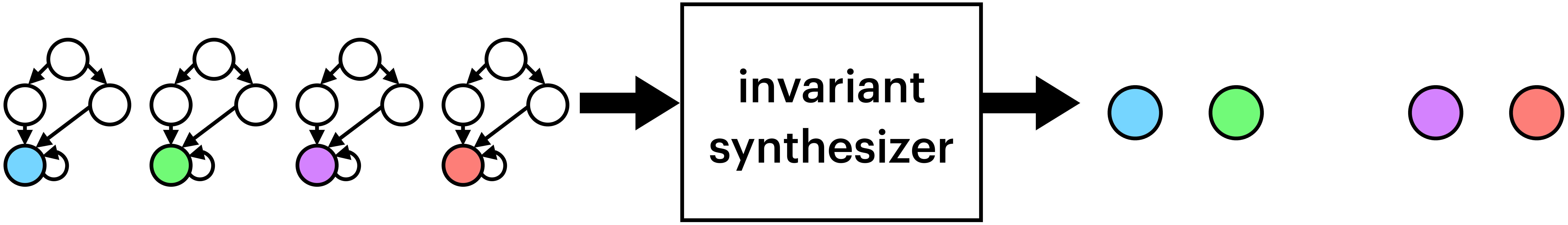
synchronize (align) structurally similar parts
(e.g., control-flow graph nodes)



Invariants

How to make it easy to **infer** relational properties with symmetries?

synchronize (align) structurally similar parts
(e.g., control-flow graph nodes)

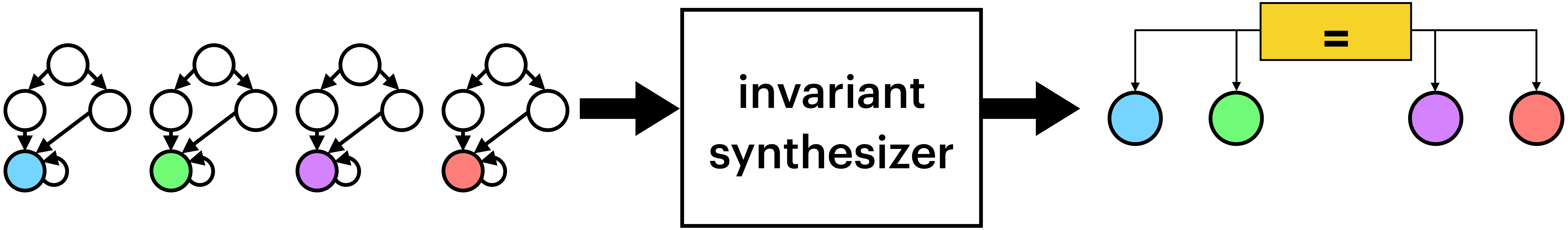


infer simpler relational invariants
that are more likely to have symmetries

Invariants

How to make it easy to **infer** relational properties with symmetries?

synchronize (align) structurally similar parts
(e.g., control-flow graph nodes)

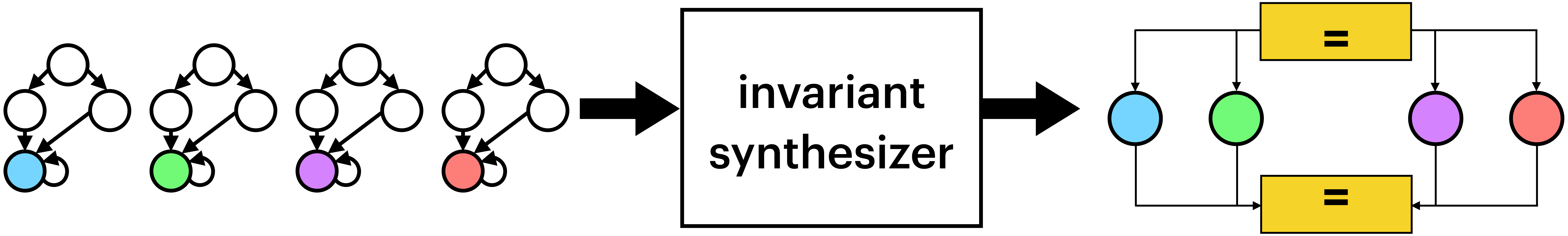


infer simpler relational invariants
that are more likely to have symmetries

Invariants

How to make it easy to **infer** relational properties with symmetries?

synchronize (align) structurally similar parts
(e.g., control-flow graph nodes)



infer simpler relational invariants
that are more likely to have symmetries

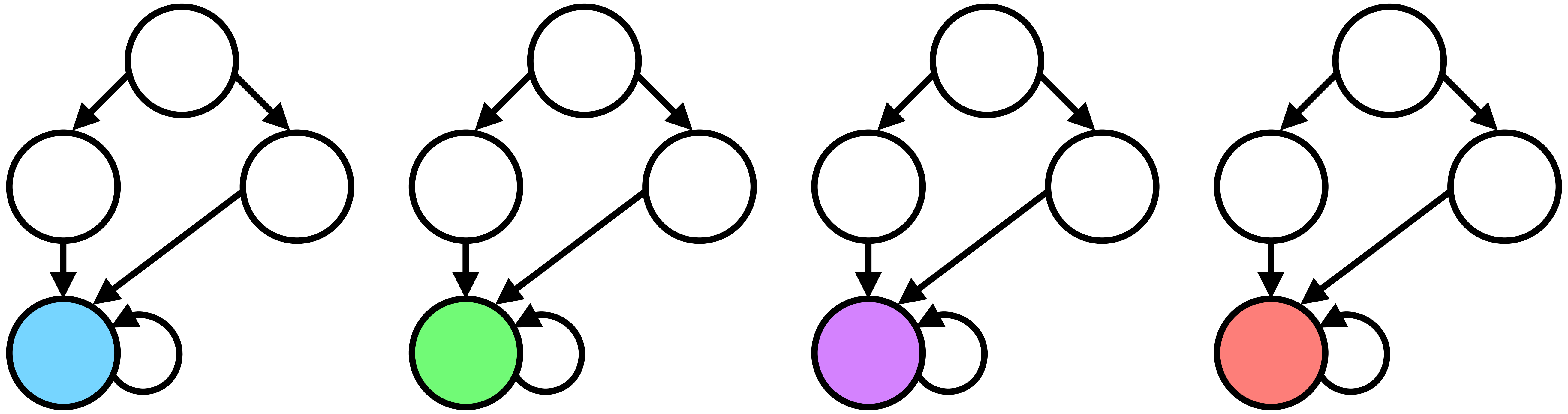
Synchrony for Loops

How to make it easy to infer relational loop invariants?

[Barthe et al., 2011]
[Sousa and Dillig, 2016]

Synchrony for Loops

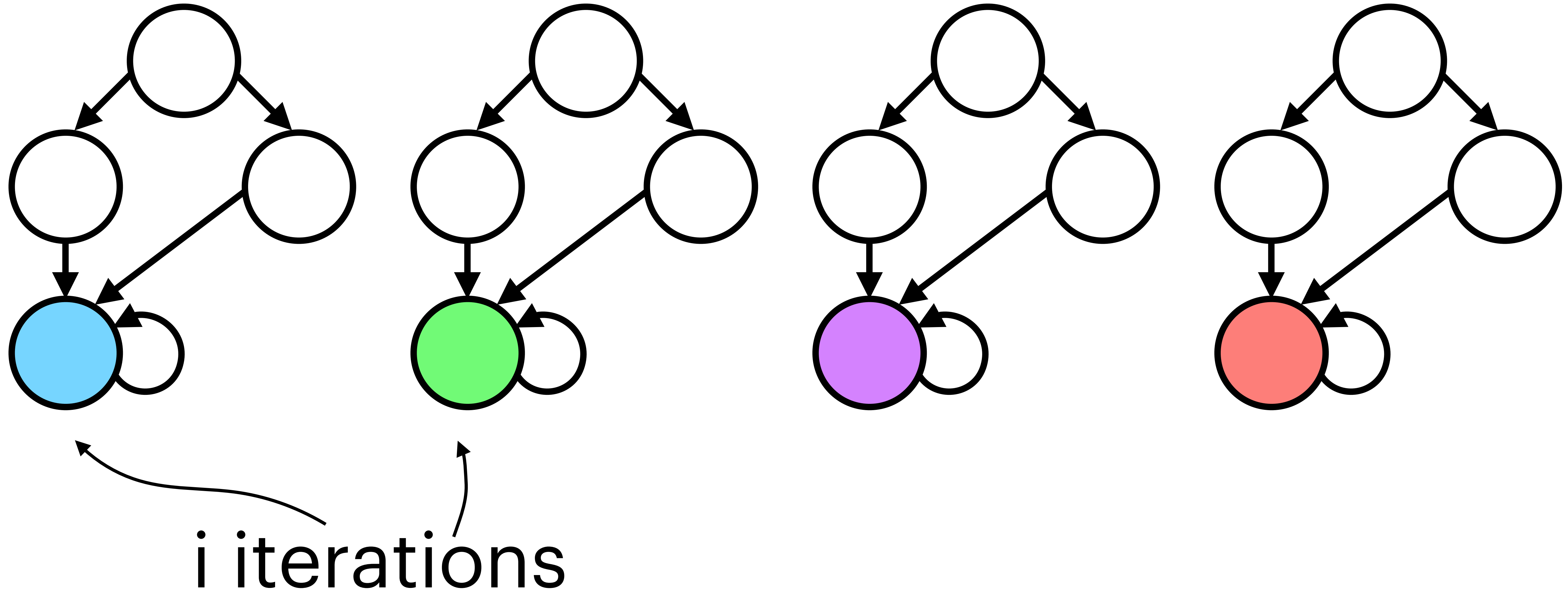
How to make it easy to infer relational loop invariants?



[Barthe et al., 2011]
[Sousa and Dillig, 2016]

Synchrony for Loops

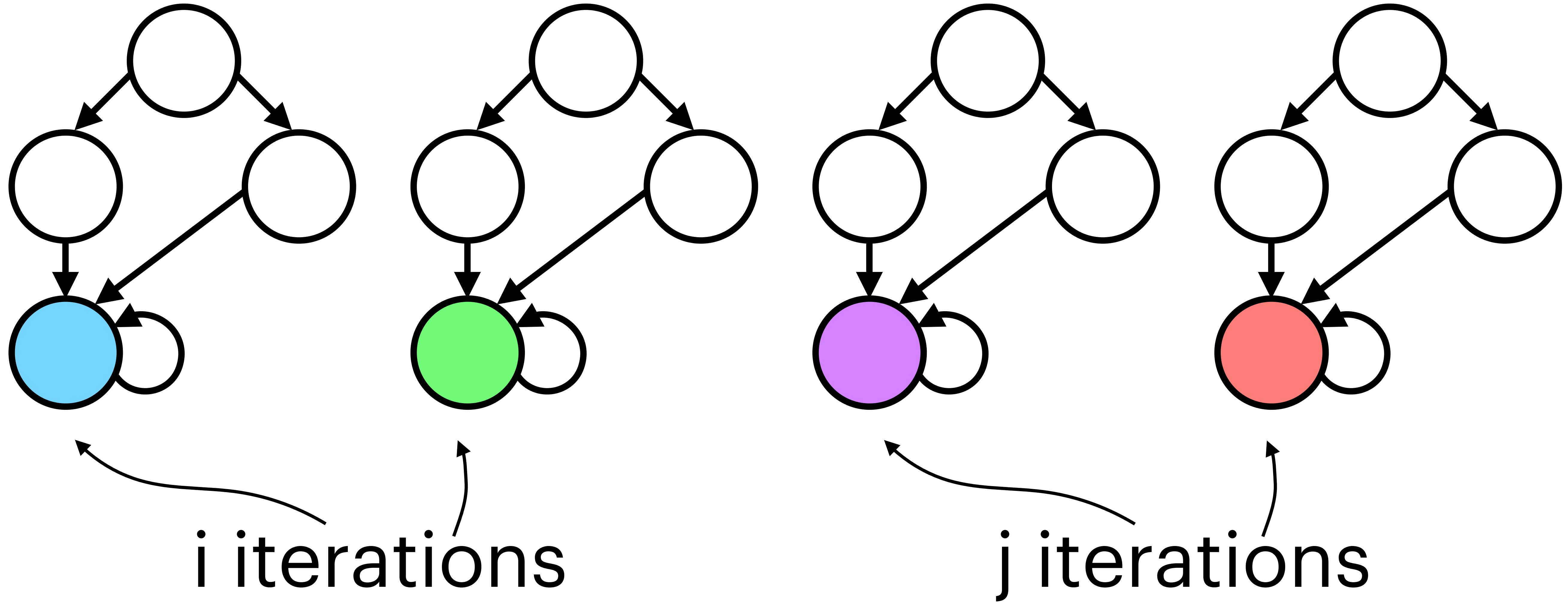
How to make it easy to infer relational loop invariants?



[Barthe et al., 2011]
[Sousa and Dillig, 2016]

Synchrony for Loops

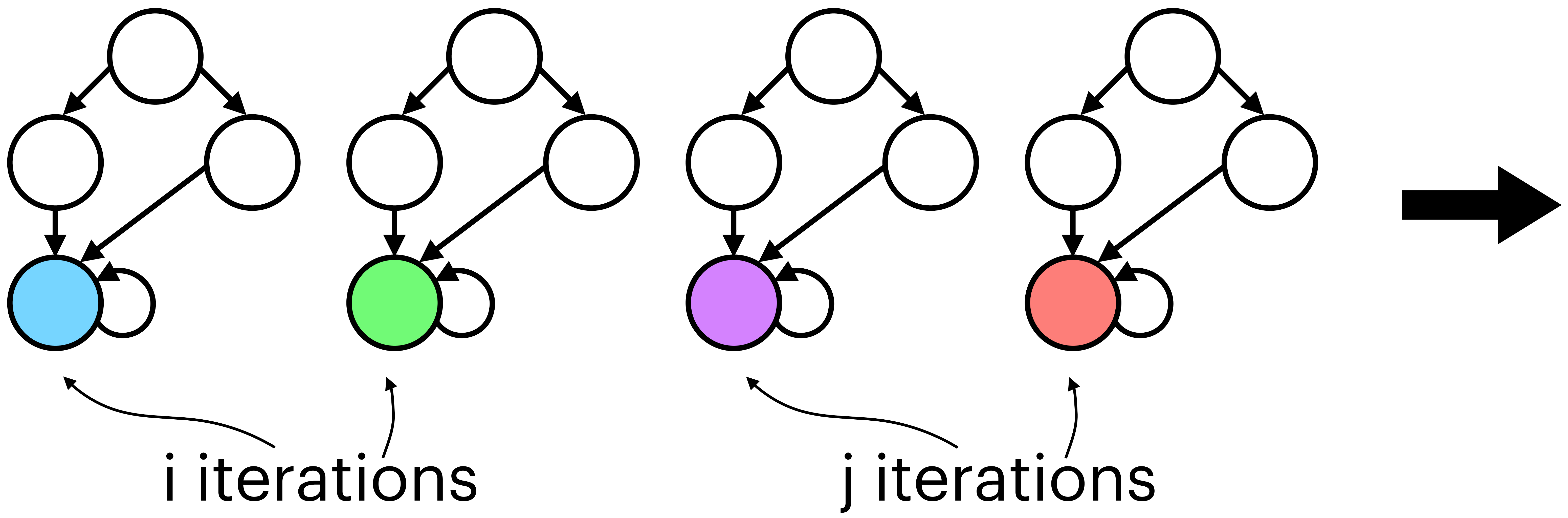
How to make it easy to infer relational loop invariants?



[Barthe et al., 2011]
[Sousa and Dillig, 2016]

Synchrony for Loops

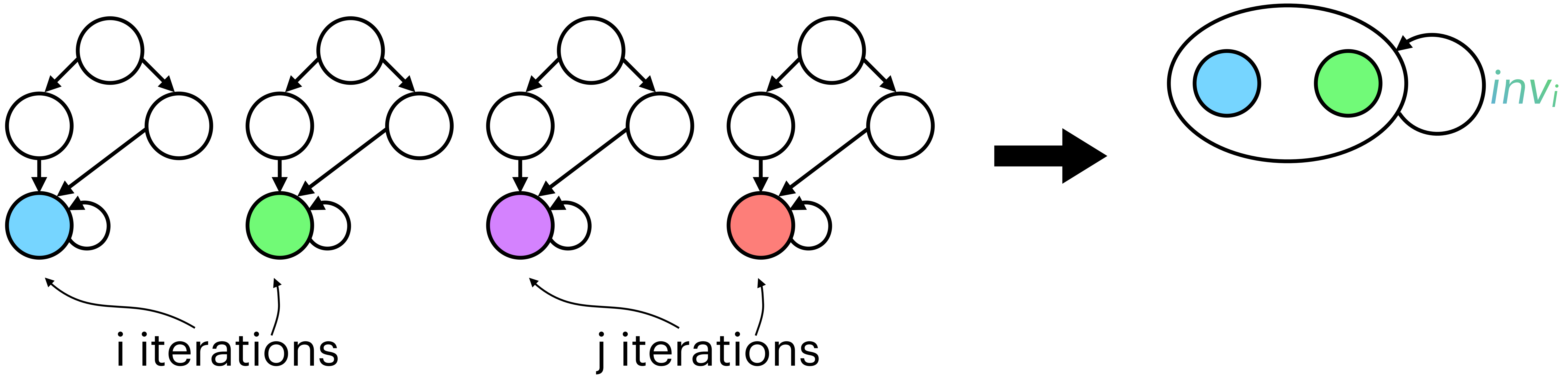
How to make it easy to infer relational loop invariants?



[Barthe et al., 2011]
[Sousa and Dillig, 2016]

Synchrony for Loops

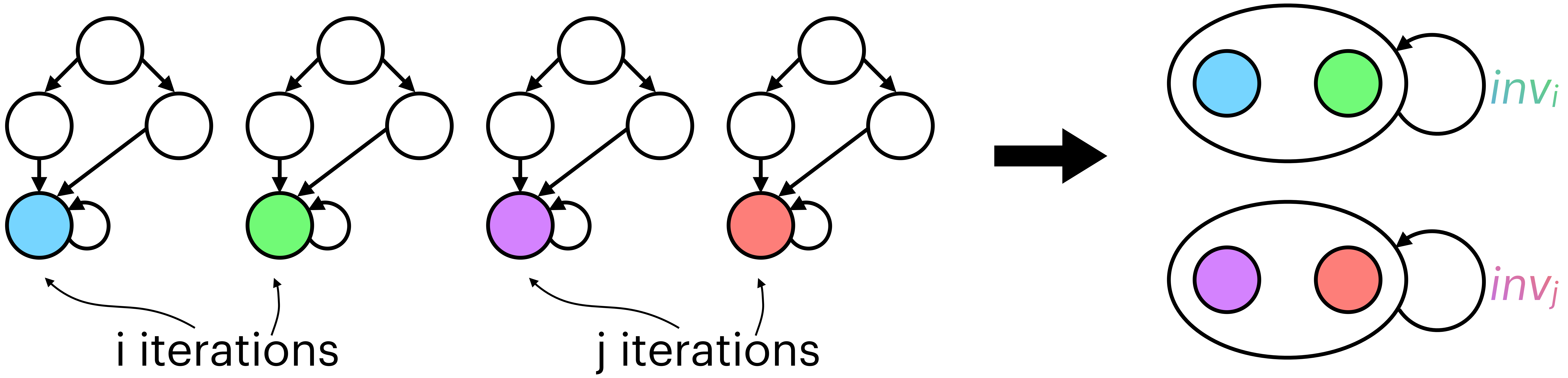
How to make it easy to infer relational loop invariants?



[Barthe et al., 2011]
[Sousa and Dillig, 2016]

Synchrony for Loops

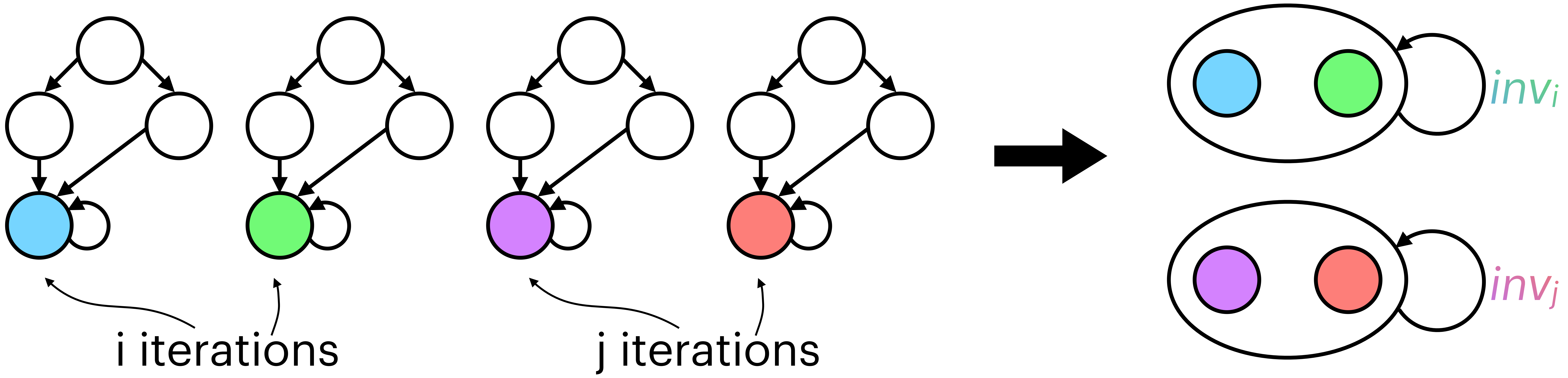
How to make it easy to infer relational loop invariants?



[Barthe et al., 2011]
[Sousa and Dillig, 2016]

Synchrony for Loops

How to make it easy to infer relational loop invariants?



Use one simple relational loop invariant per set of “lockstep” loops.

[Barthe et al., 2011]
[Sousa and Dillig, 2016]

Maximal Lockstep Loop Detection

Synthesize simple relational invariant I, then do partition-refinement:

At each step, ask:

[Pick et al., 2018]

Maximal Lockstep Loop Detection

Synthesize simple relational invariant I, then do partition-refinement:

At each step, ask:

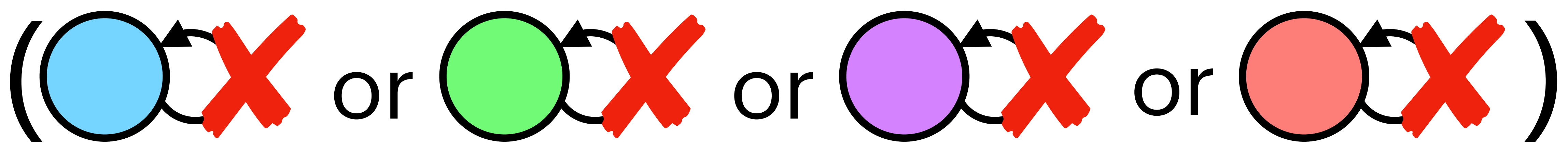
I and

[Pick et al., 2018]

Maximal Lockstep Loop Detection

Synthesize simple relational invariant I, then do partition-refinement:

At each step, ask:

I and 

One loop terminated

[Pick et al., 2018]

Maximal Lockstep Loop Detection

Synthesize simple relational invariant I, then do partition-refinement:

At each step, ask:

I and 

One loop terminated

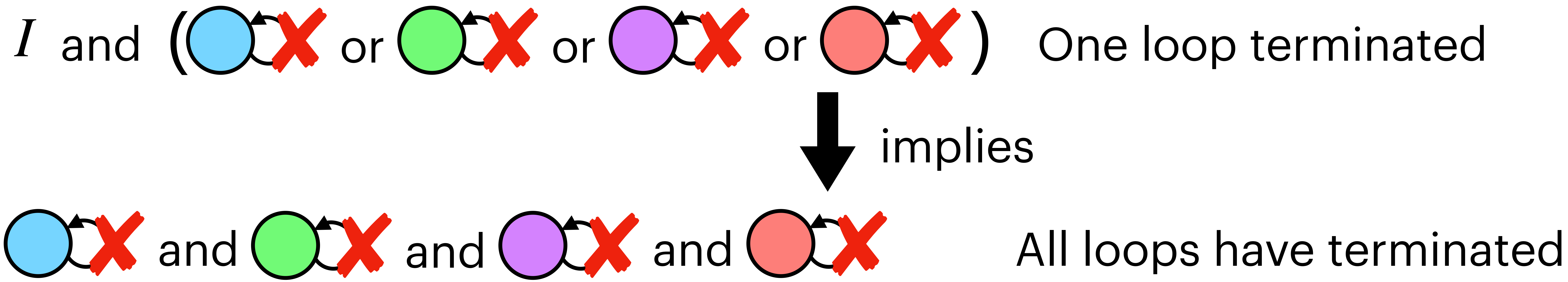
implies

[Pick et al., 2018]

Maximal Lockstep Loop Detection

Synthesize simple relational invariant I, then do partition-refinement:

At each step, ask:

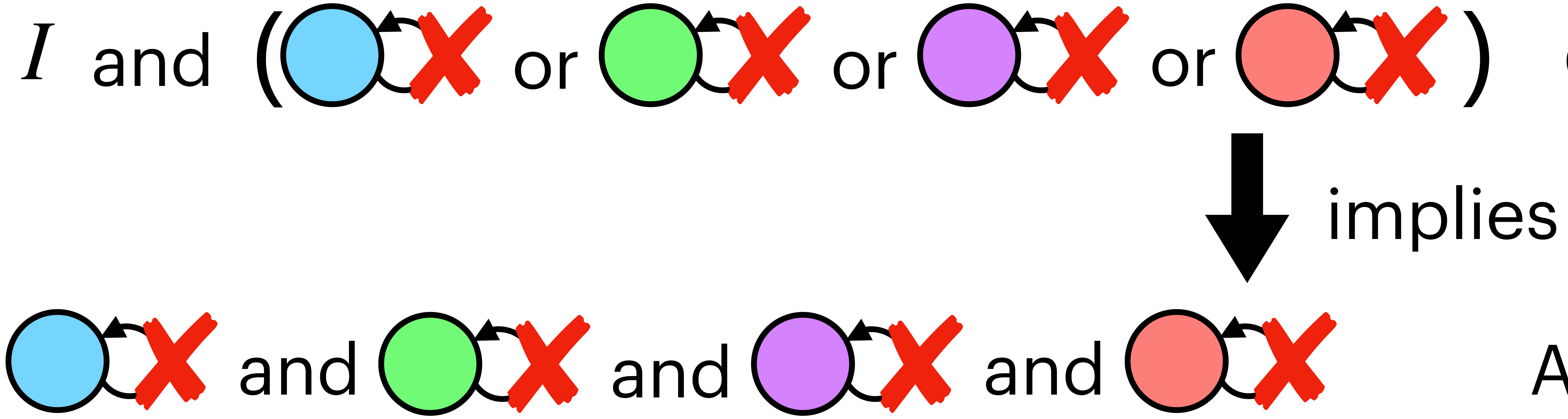


[Pick et al., 2018]

Maximal Lockstep Loop Detection

Synthesize simple relational invariant I, then do partition-refinement:

At each step, ask:



One loop terminated

All loops have terminated

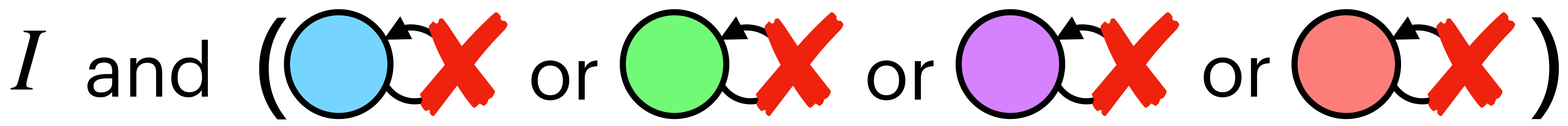
Ask as SMT query, and use model to partition

[Pick et al., 2018]

Maximal Lockstep Loop Detection

Synthesize simple relational invariant I, then do partition-refinement:

At each step, ask:



One loop terminated

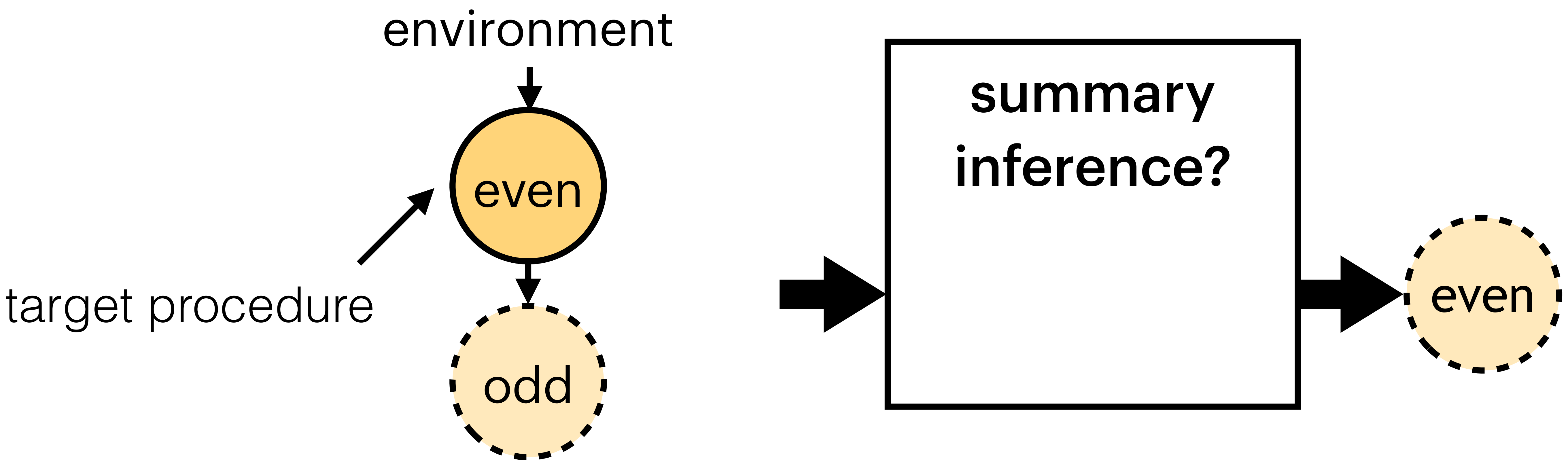
implies

All loops have terminated

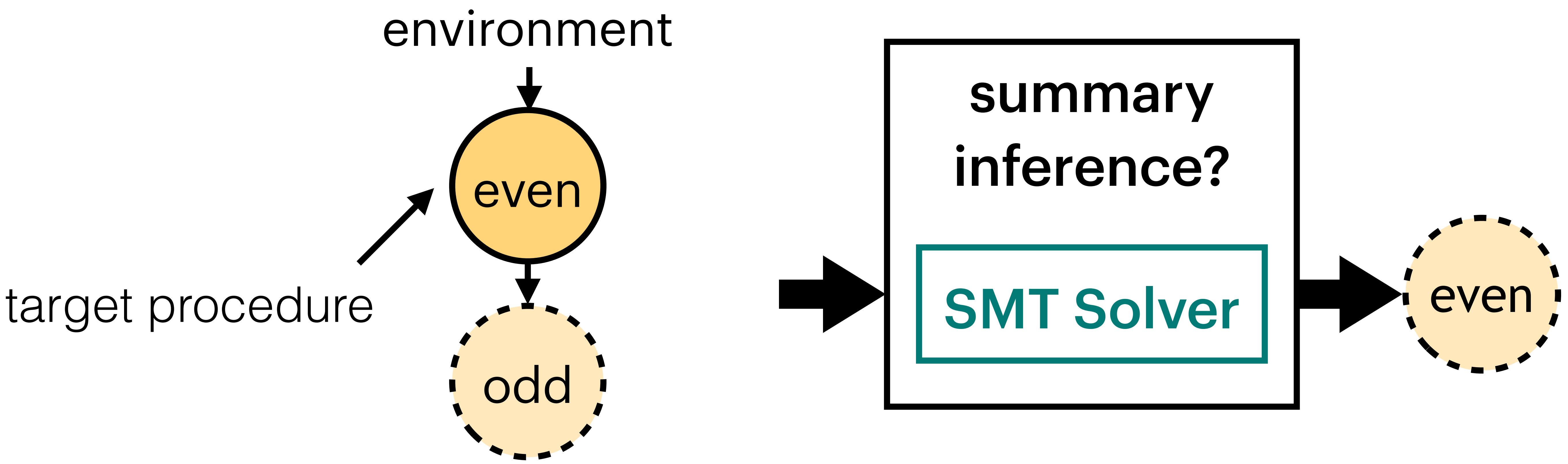
Ask as SMT query, and use model to partition

[Pick et al., 2018]

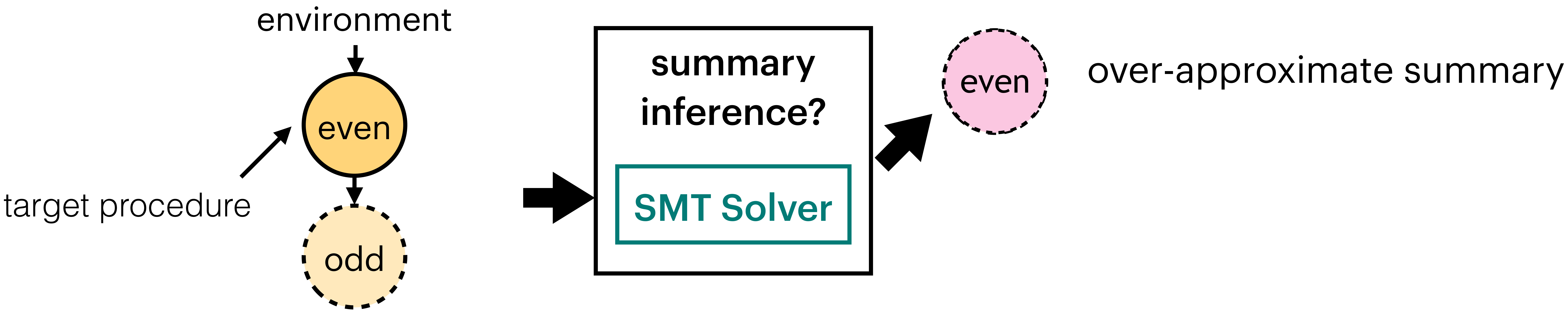
Summary Inference



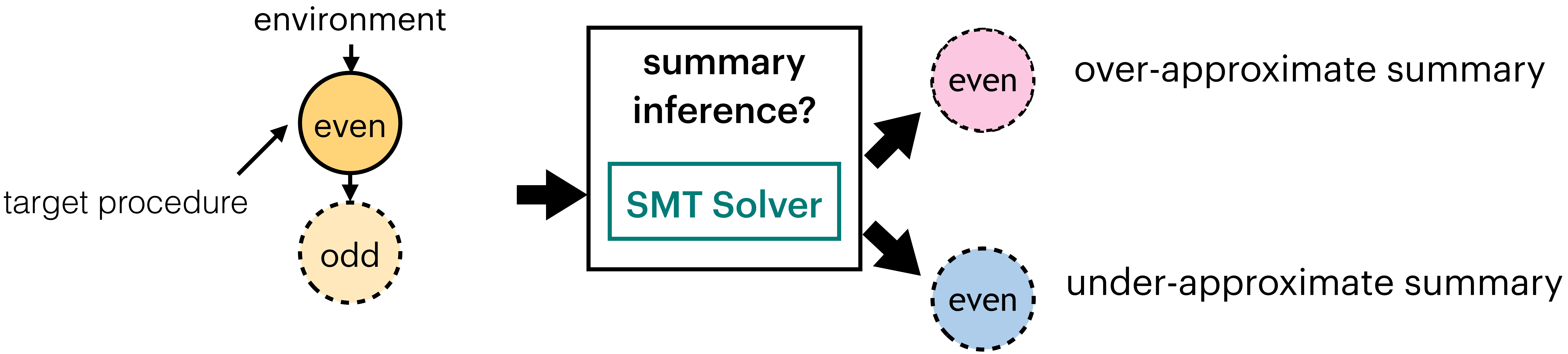
Summary Inference



Summary Inference

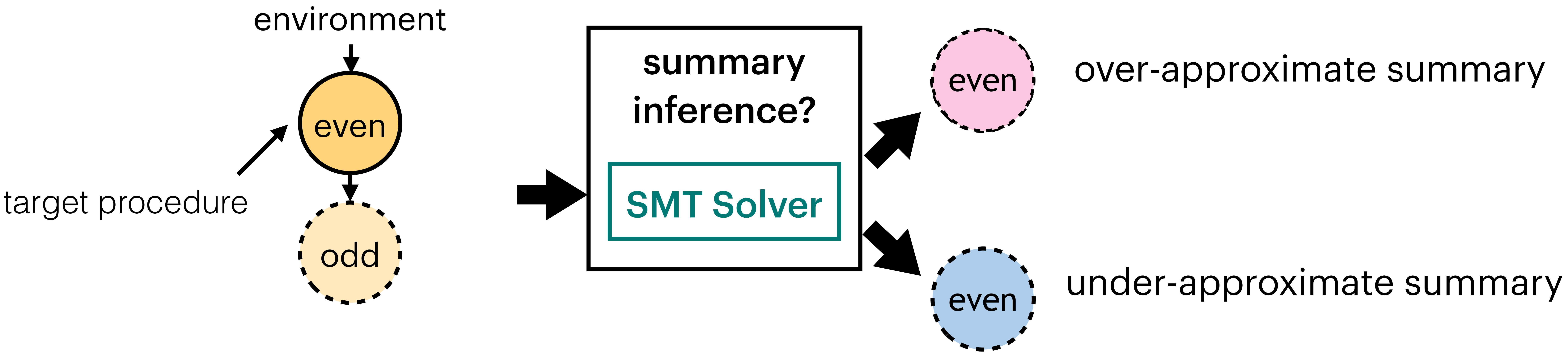


Summary Inference



Summary Inference

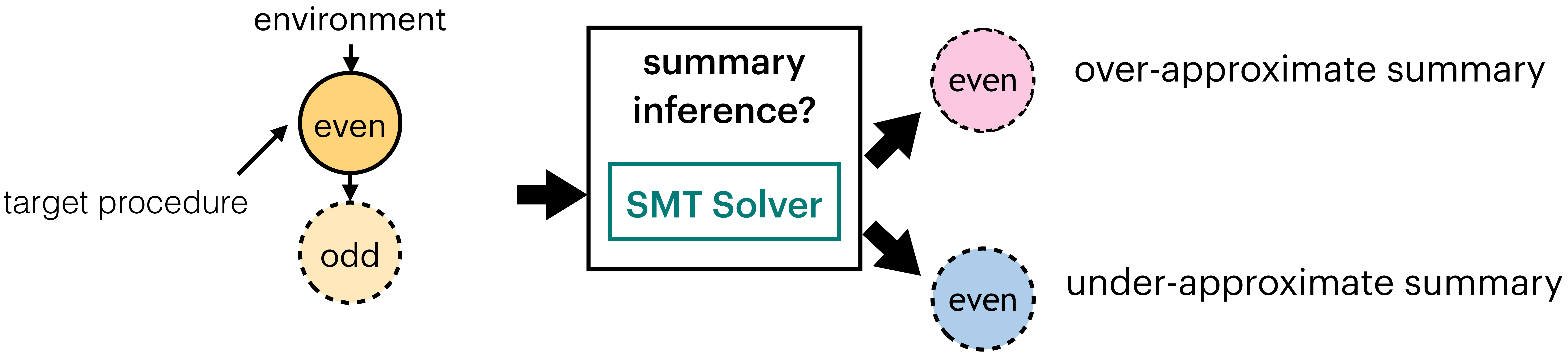
example: $y \leftarrow 2x + 2$



Summary Inference

example: $y \leftarrow 2x + 2$

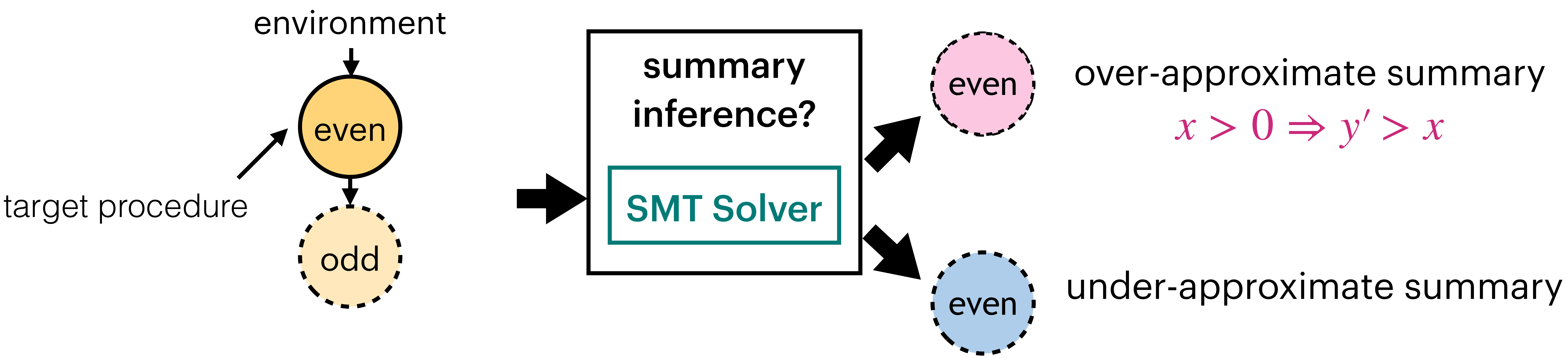
$$y' = 2x + 2$$



Summary Inference

example: $y \leftarrow 2x + 2$

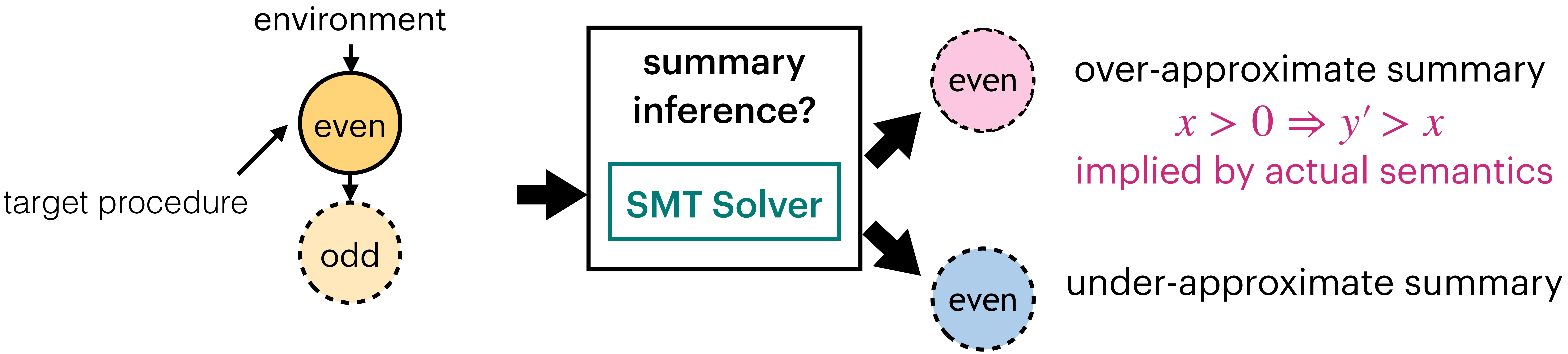
$$y' = 2x + 2$$



Summary Inference

example: $y \leftarrow 2x + 2$

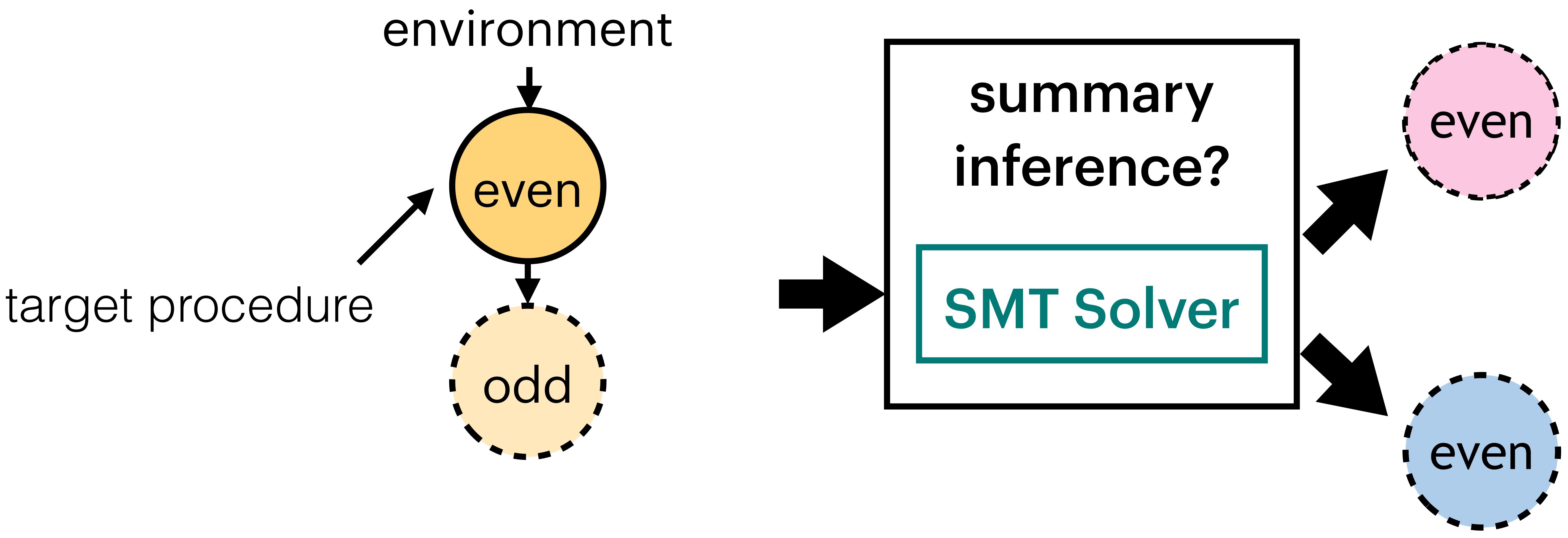
$$y' = 2x + 2$$



Summary Inference

example: $\mathrm{y} \leftarrow 2\mathrm{x} + 2$

$$y' = 2x + 2$$



over-approximate summary

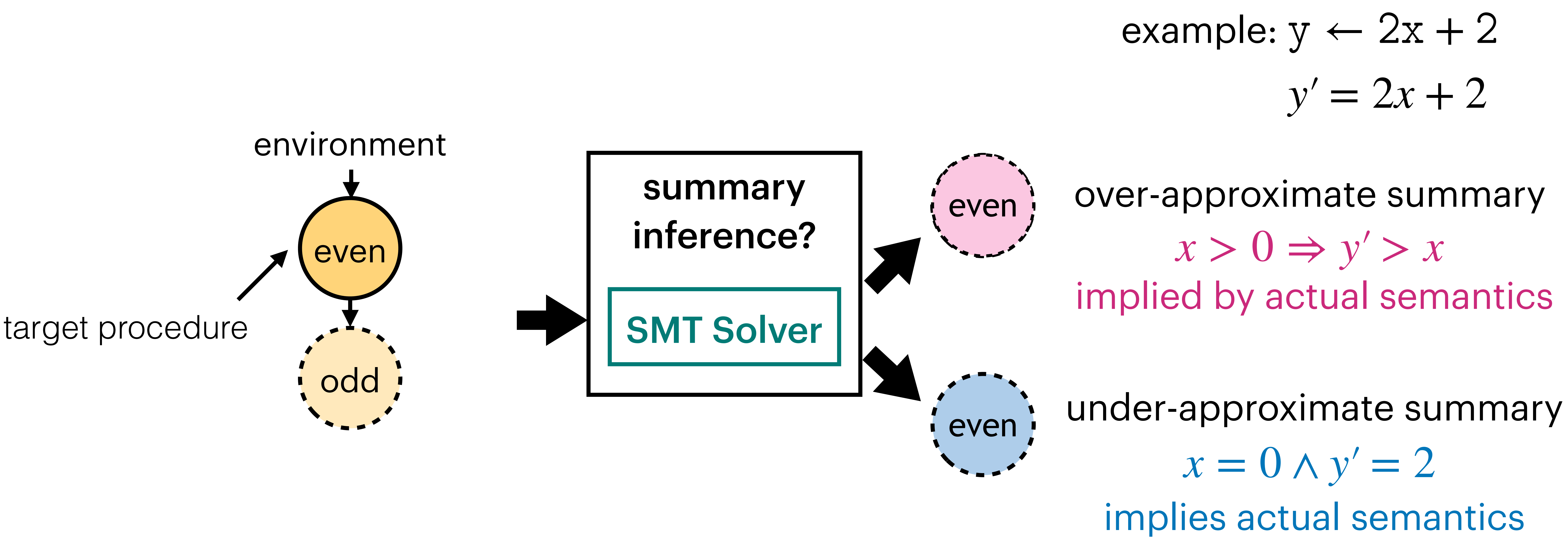
$x > 0 \Rightarrow y' > x$

implied by actual semantics

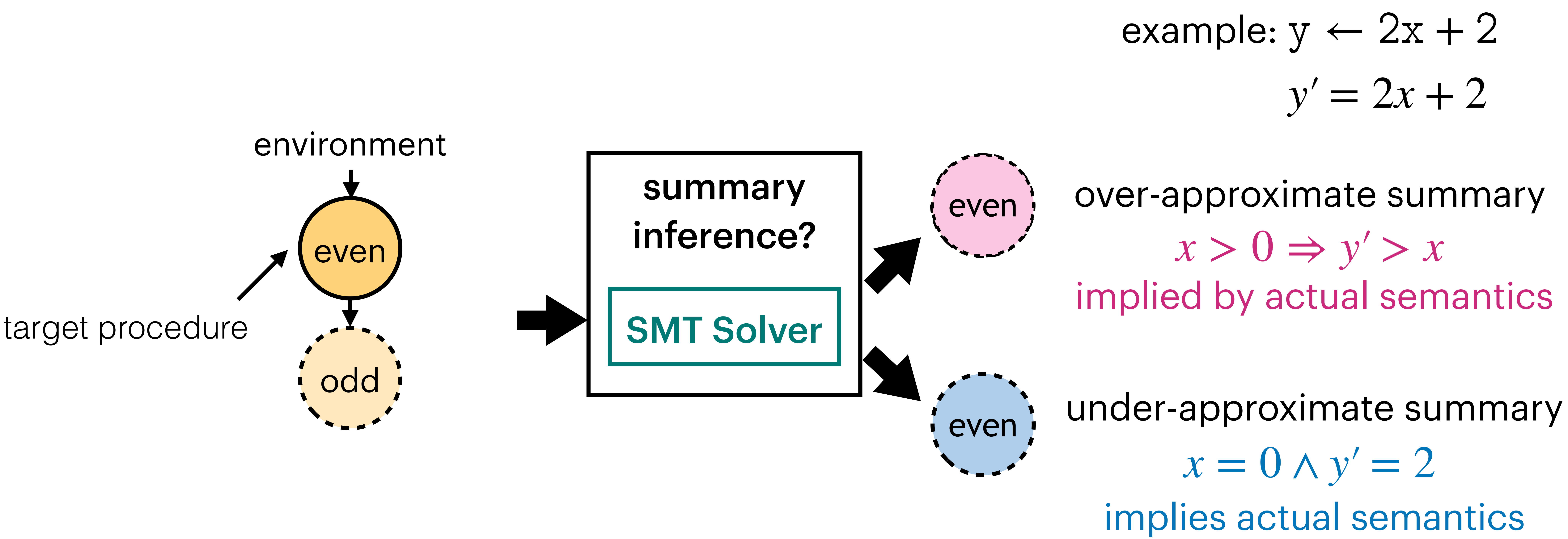
under-approximate summary

$x = 0 \wedge y' = 2$

Summary Inference

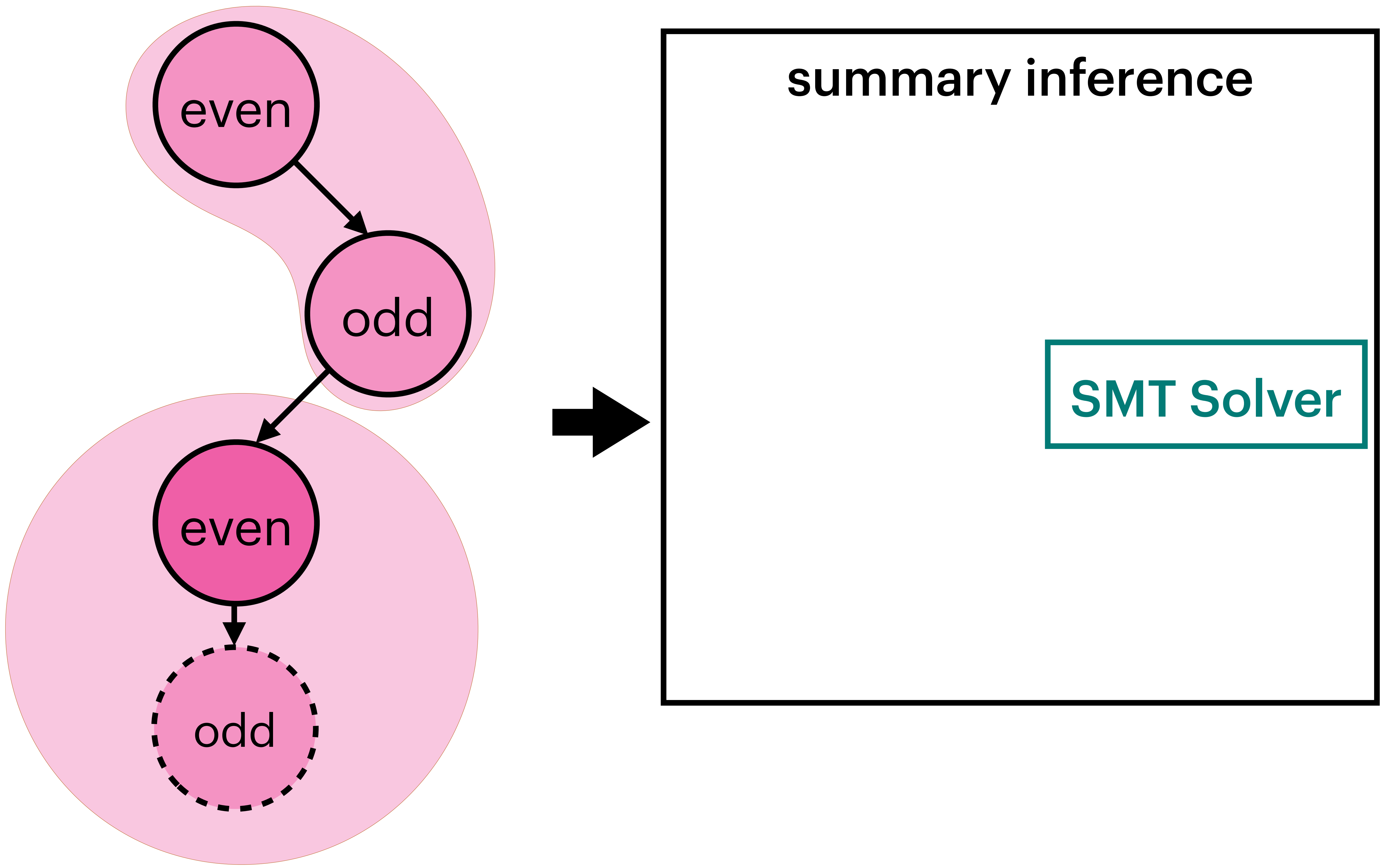


Summary Inference

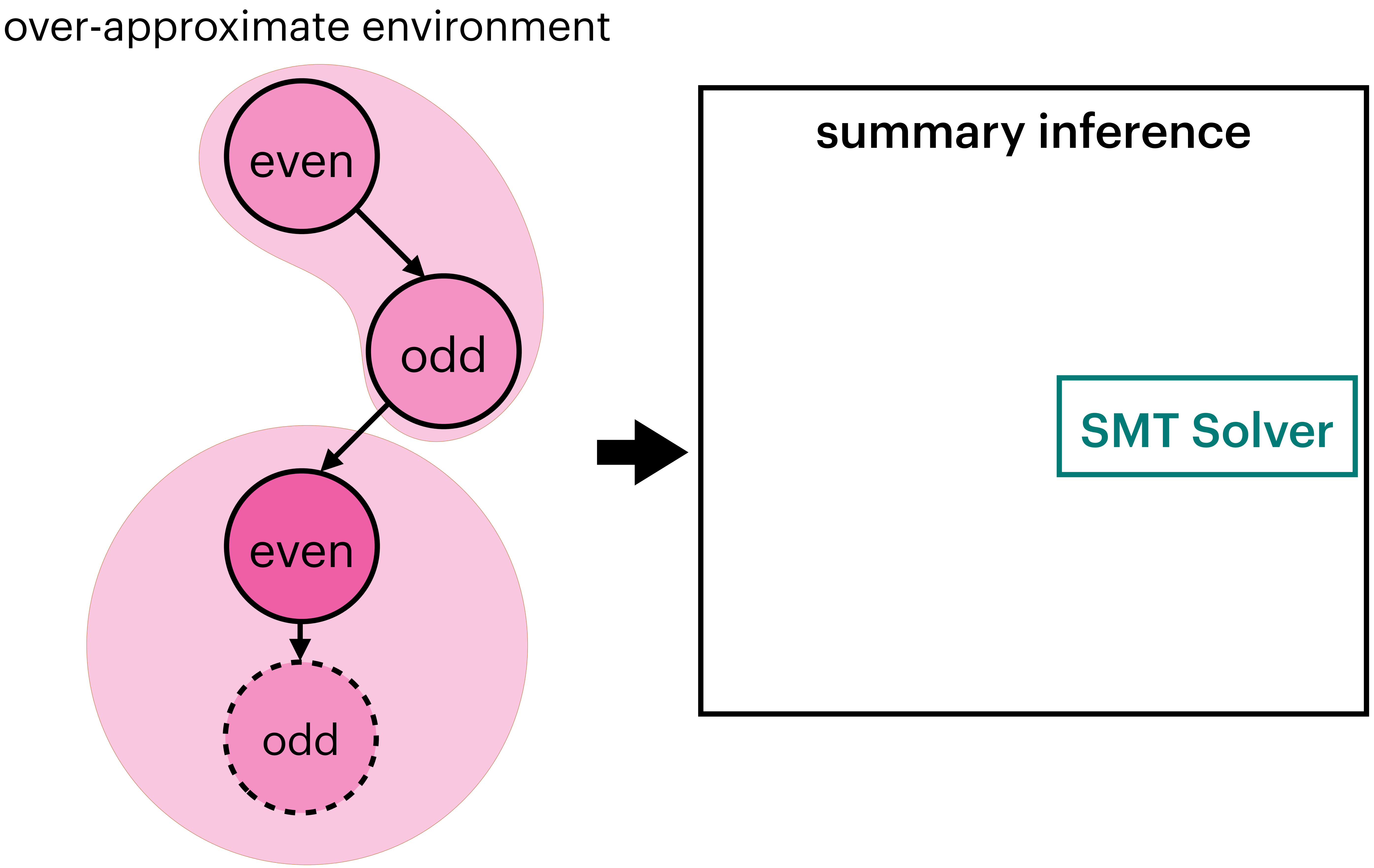


Will make four SMT queries, over- and under-approximating both environment and target procedure

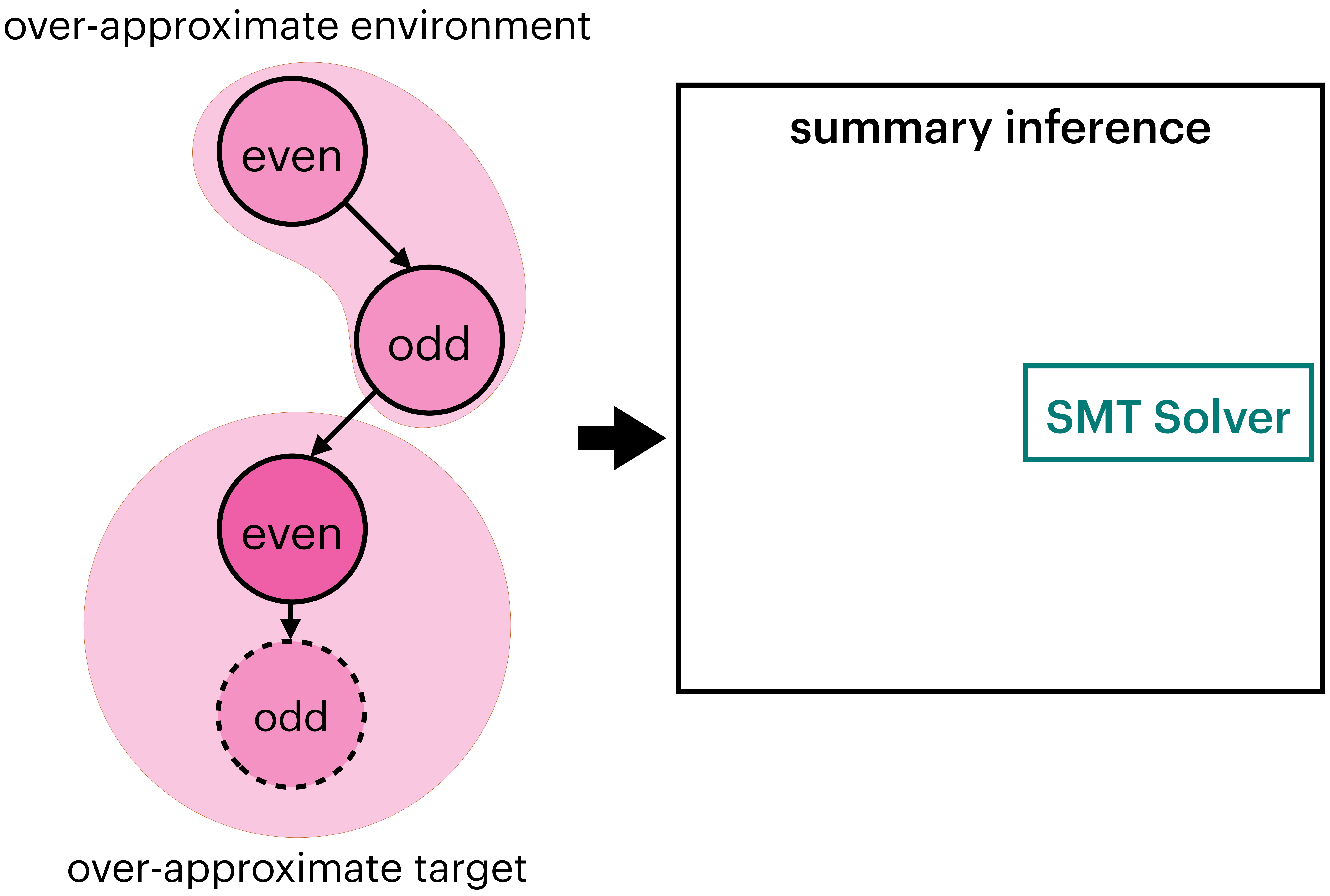
Over-Approximate Summary Inference



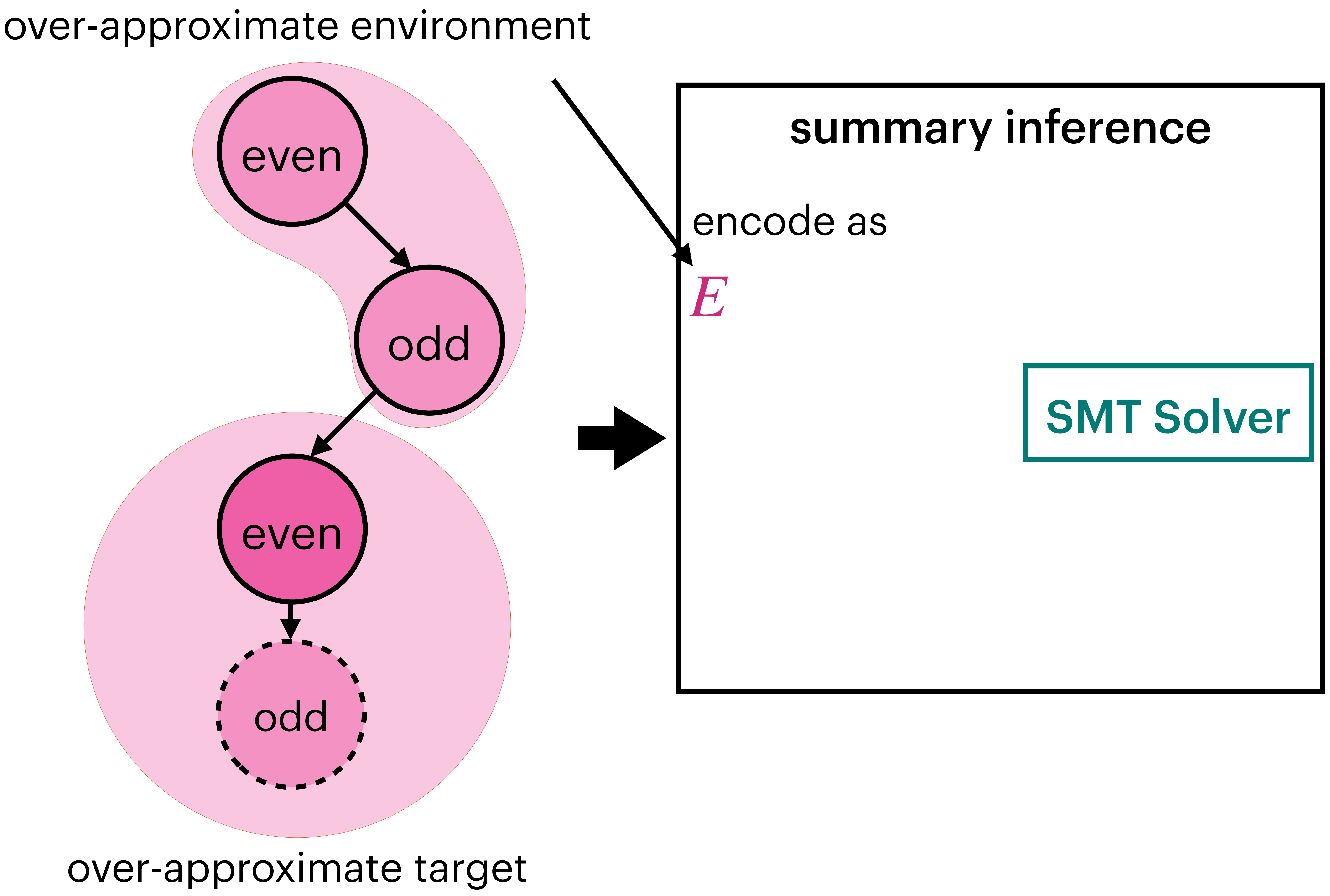
Over-Approximate Summary Inference



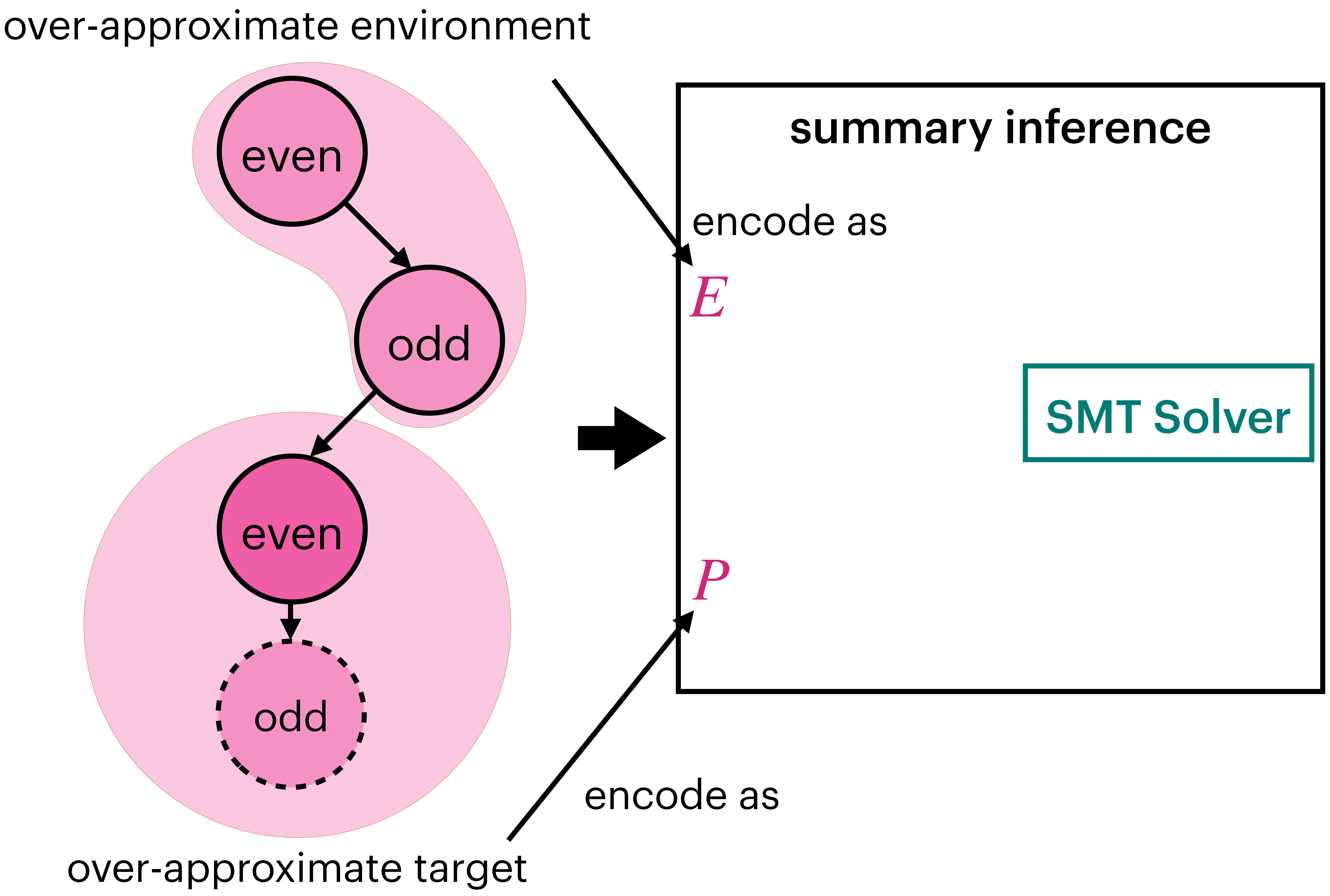
Over-Approximate Summary Inference



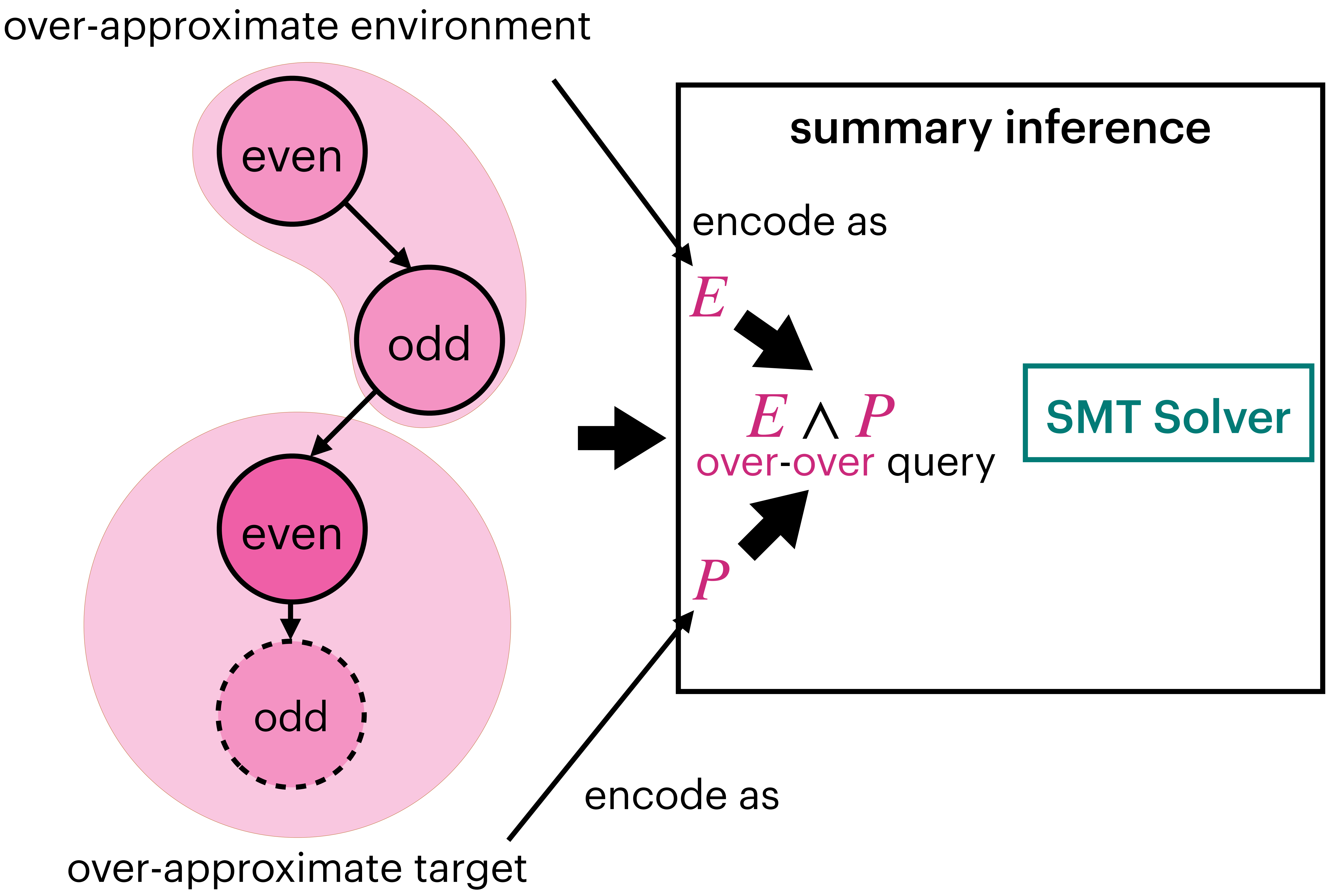
Over-Approximate Summary Inference



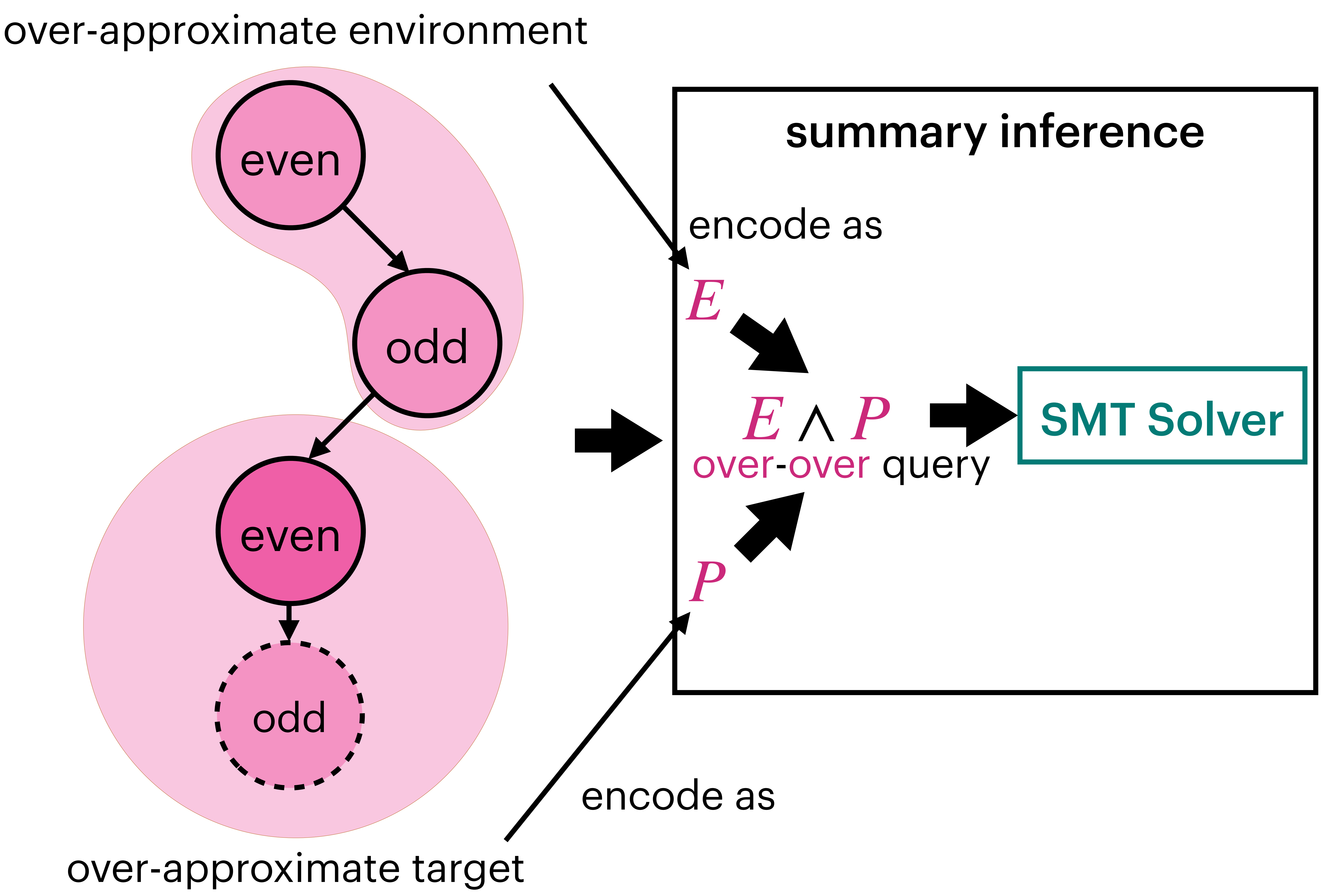
Over-Approximate Summary Inference



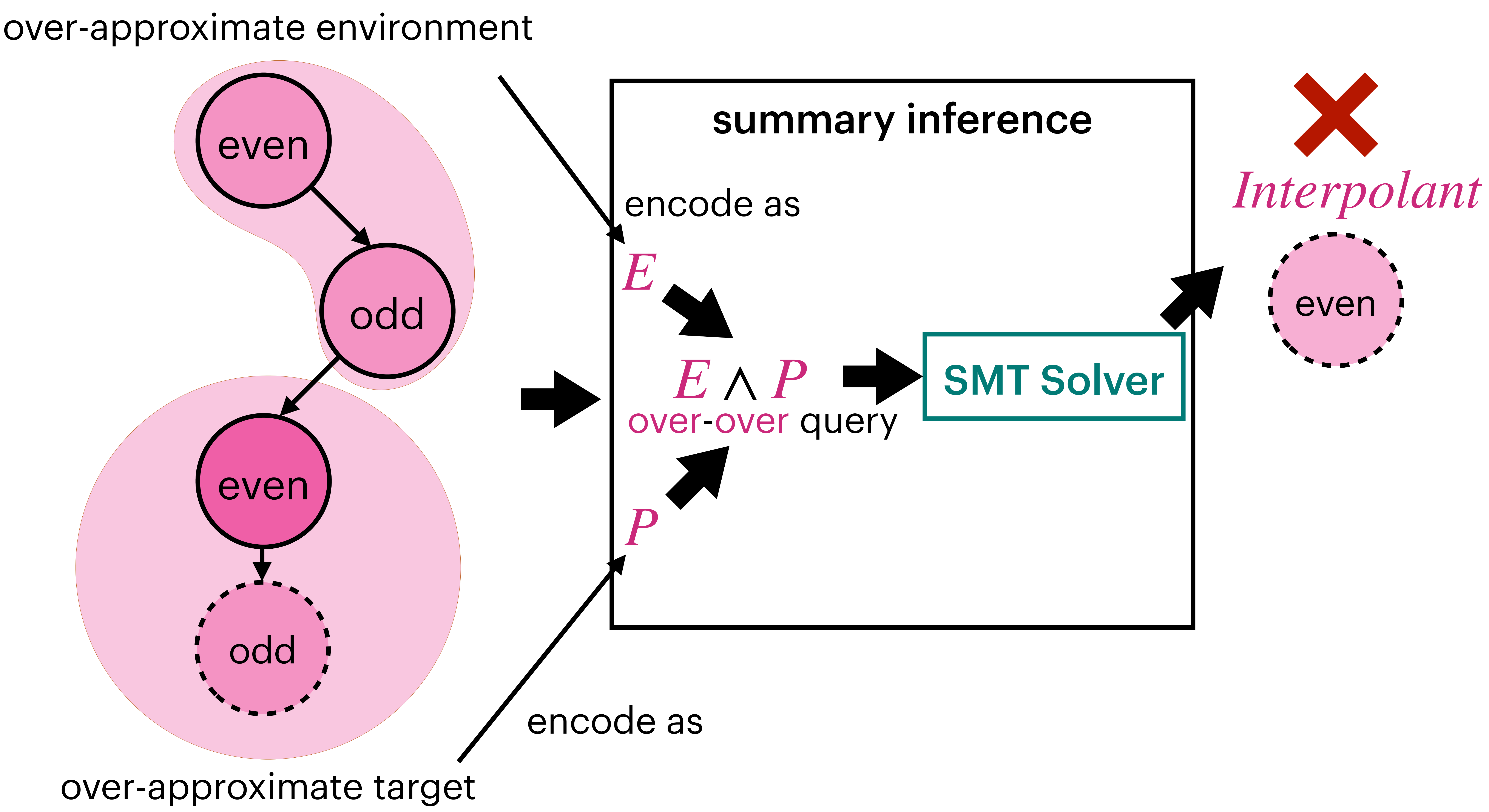
Over-Approximate Summary Inference



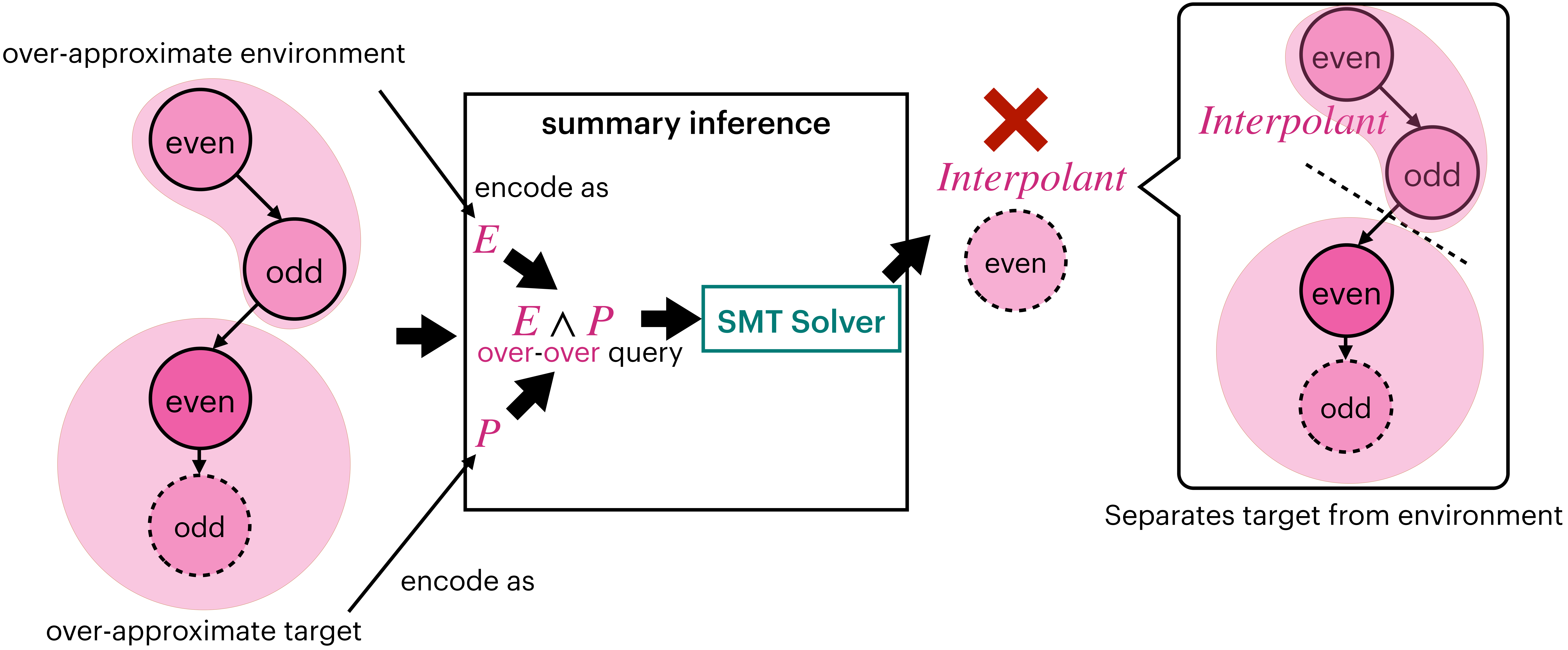
Over-Approximate Summary Inference



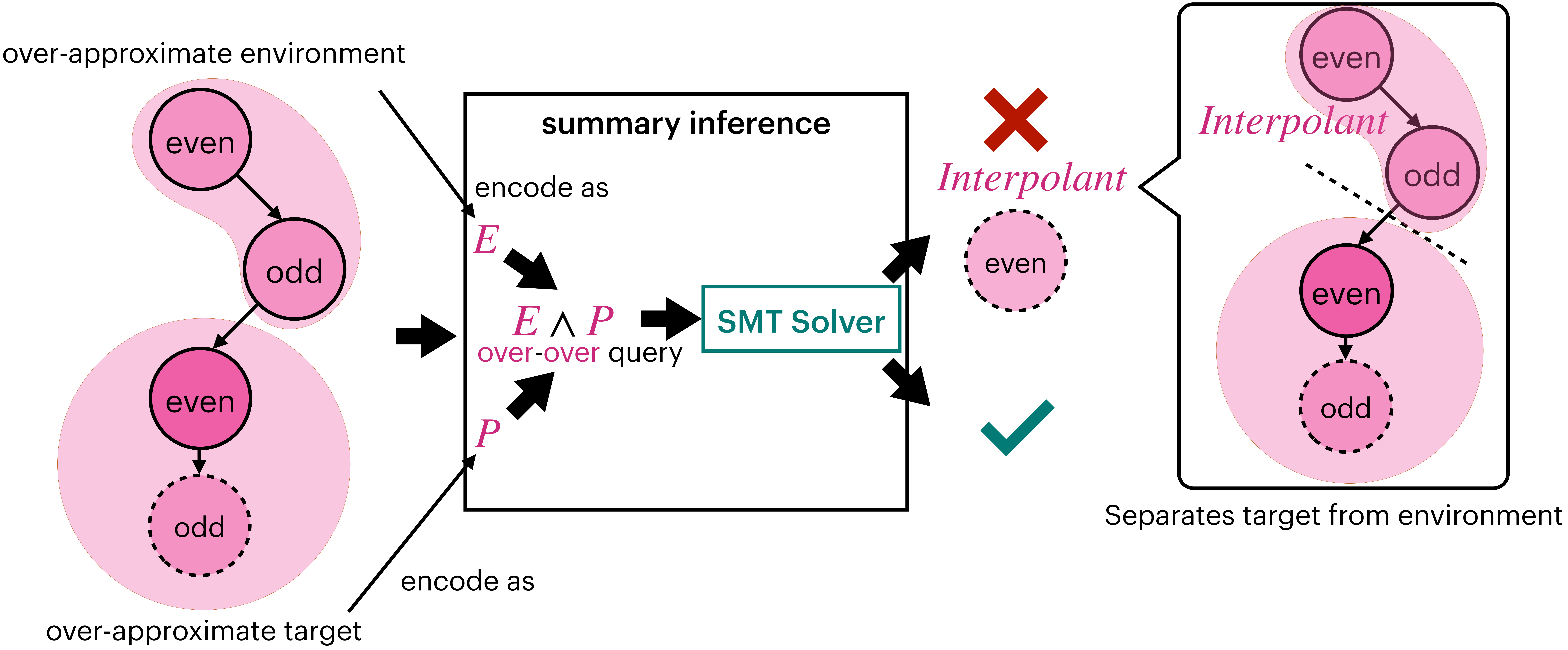
Over-Approximate Summary Inference



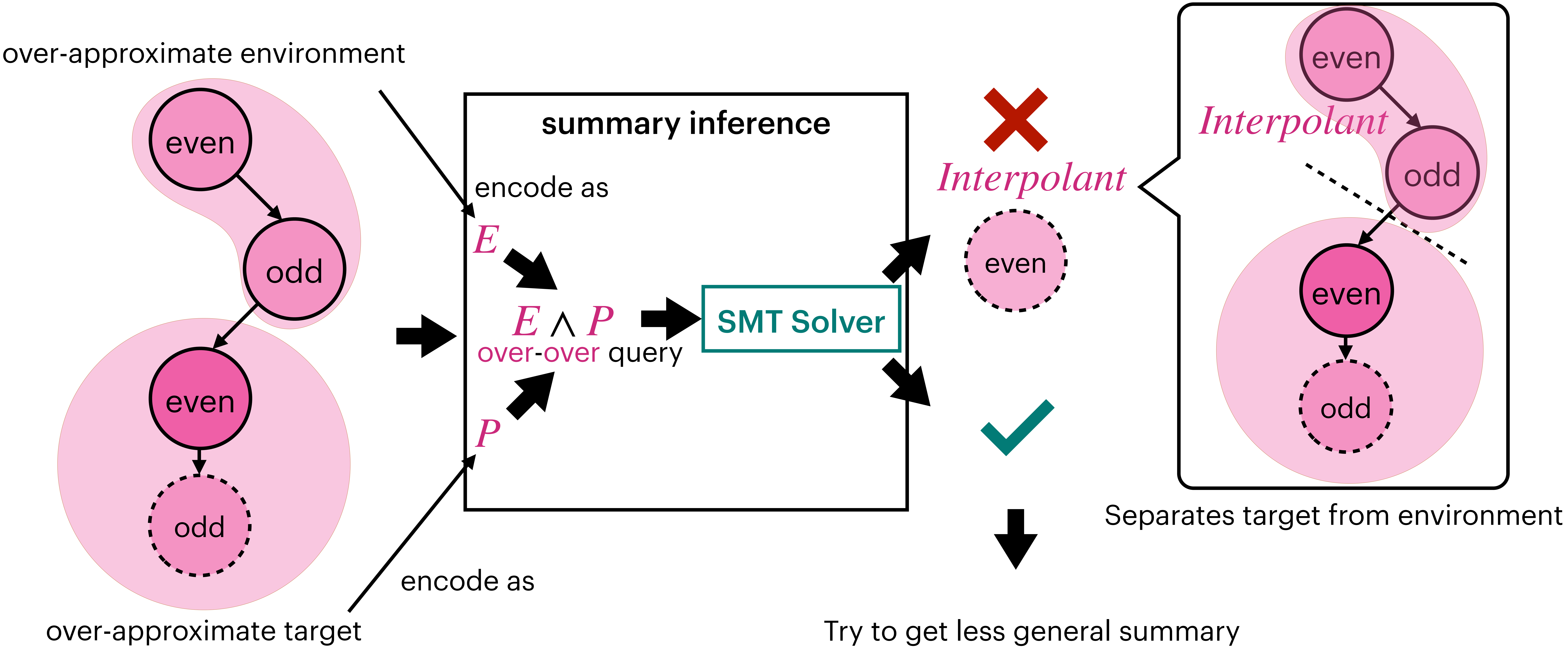
Over-Approximate Summary Inference



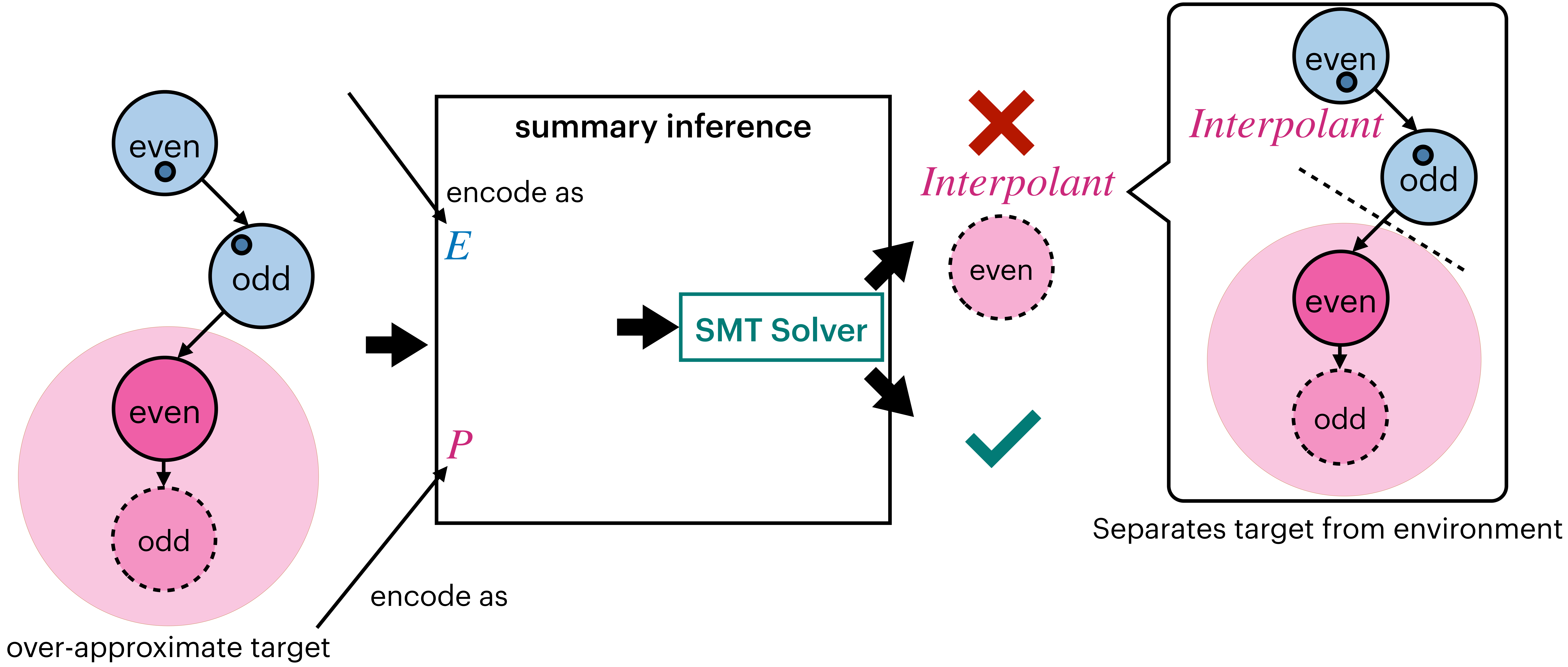
Over-Approximate Summary Inference



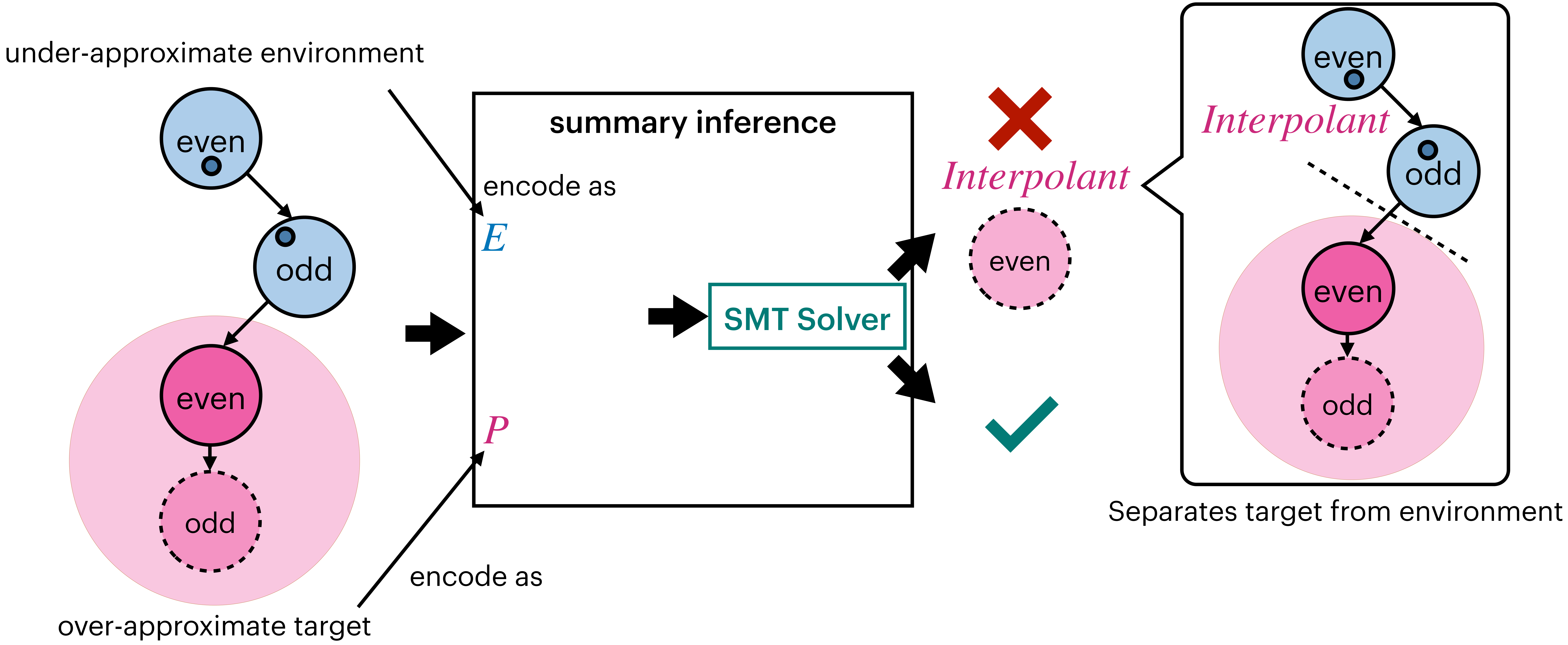
Over-Approximate Summary Inference



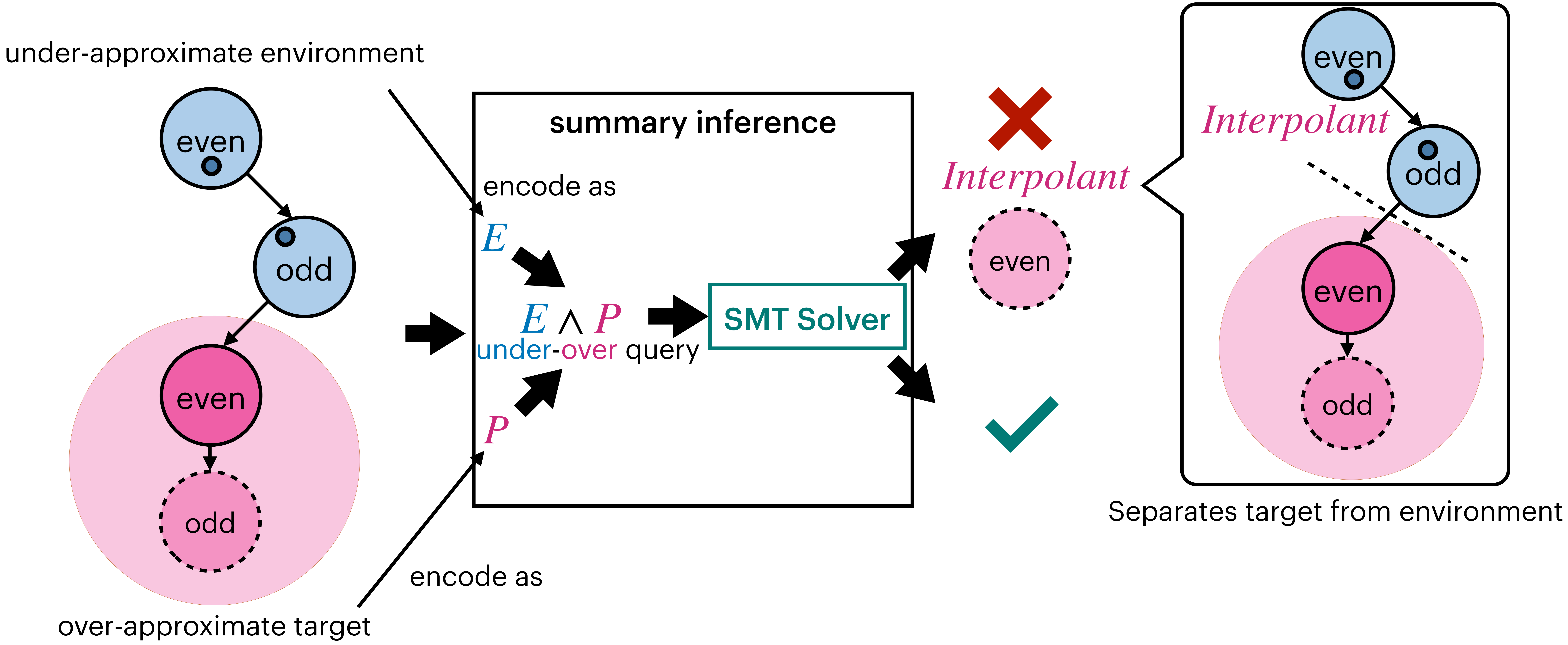
Over-Approximate Summary Inference



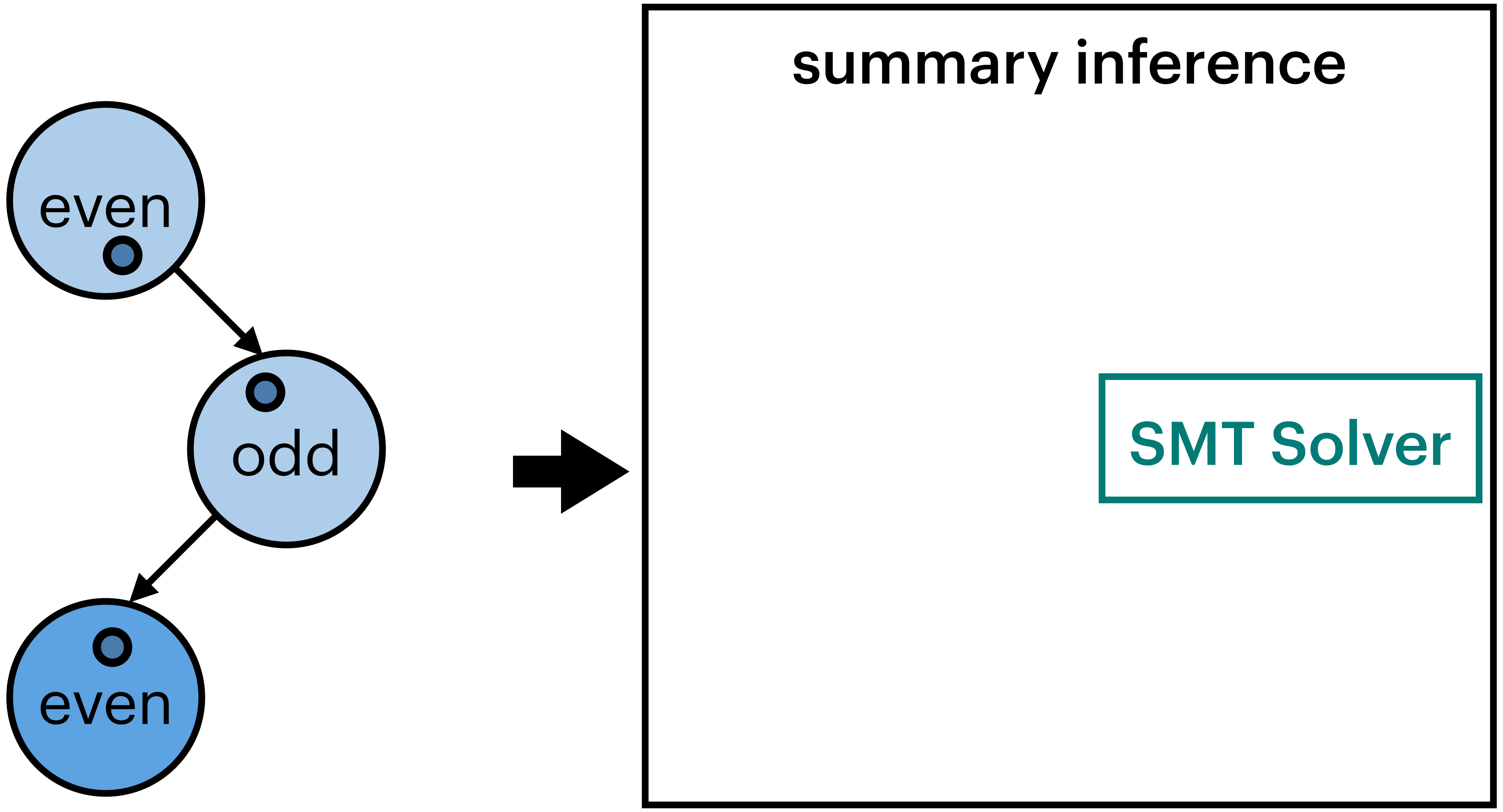
Over-Approximate Summary Inference



Over-Approximate Summary Inference

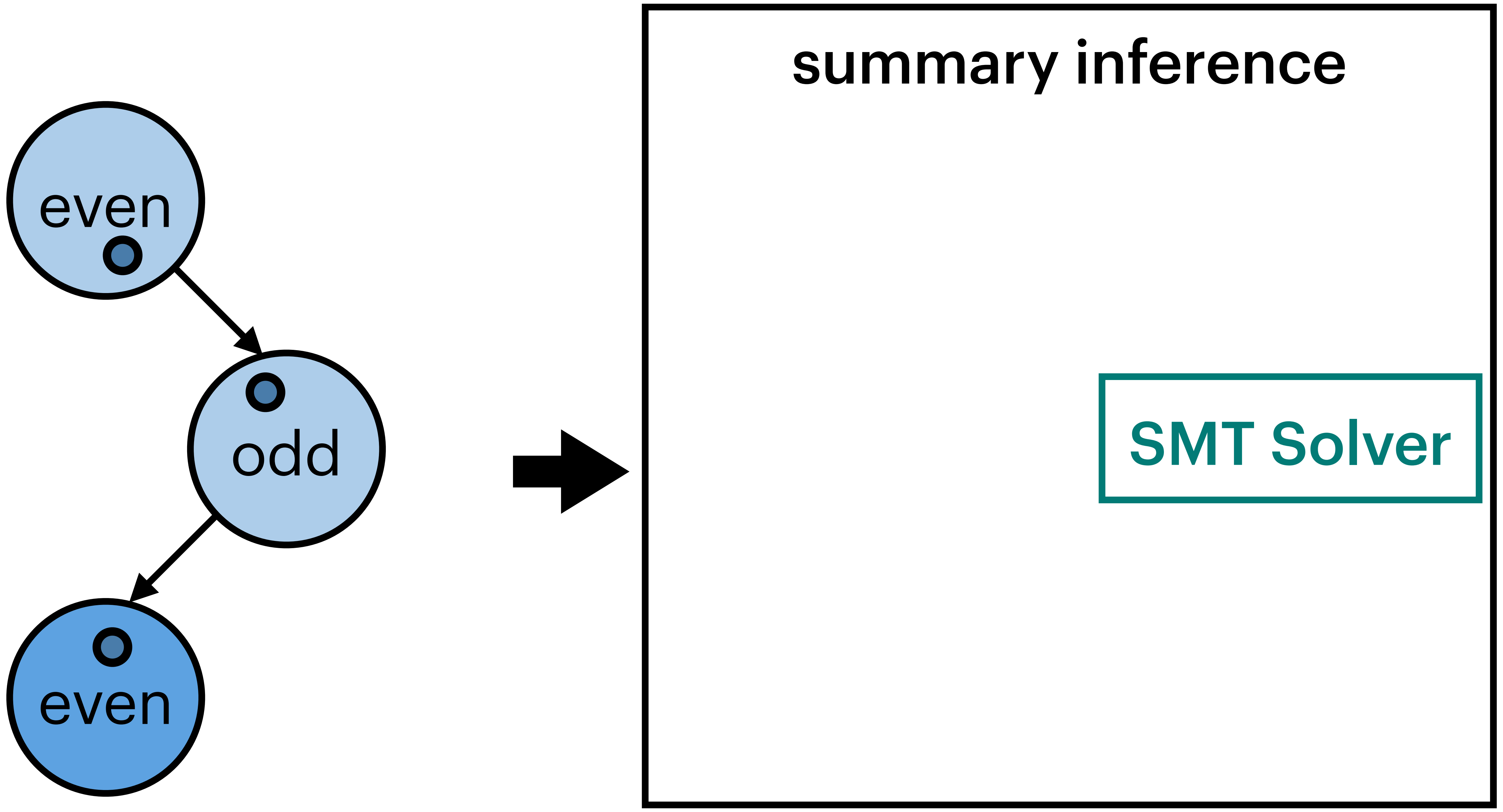


Under-Approximate Summary Inference



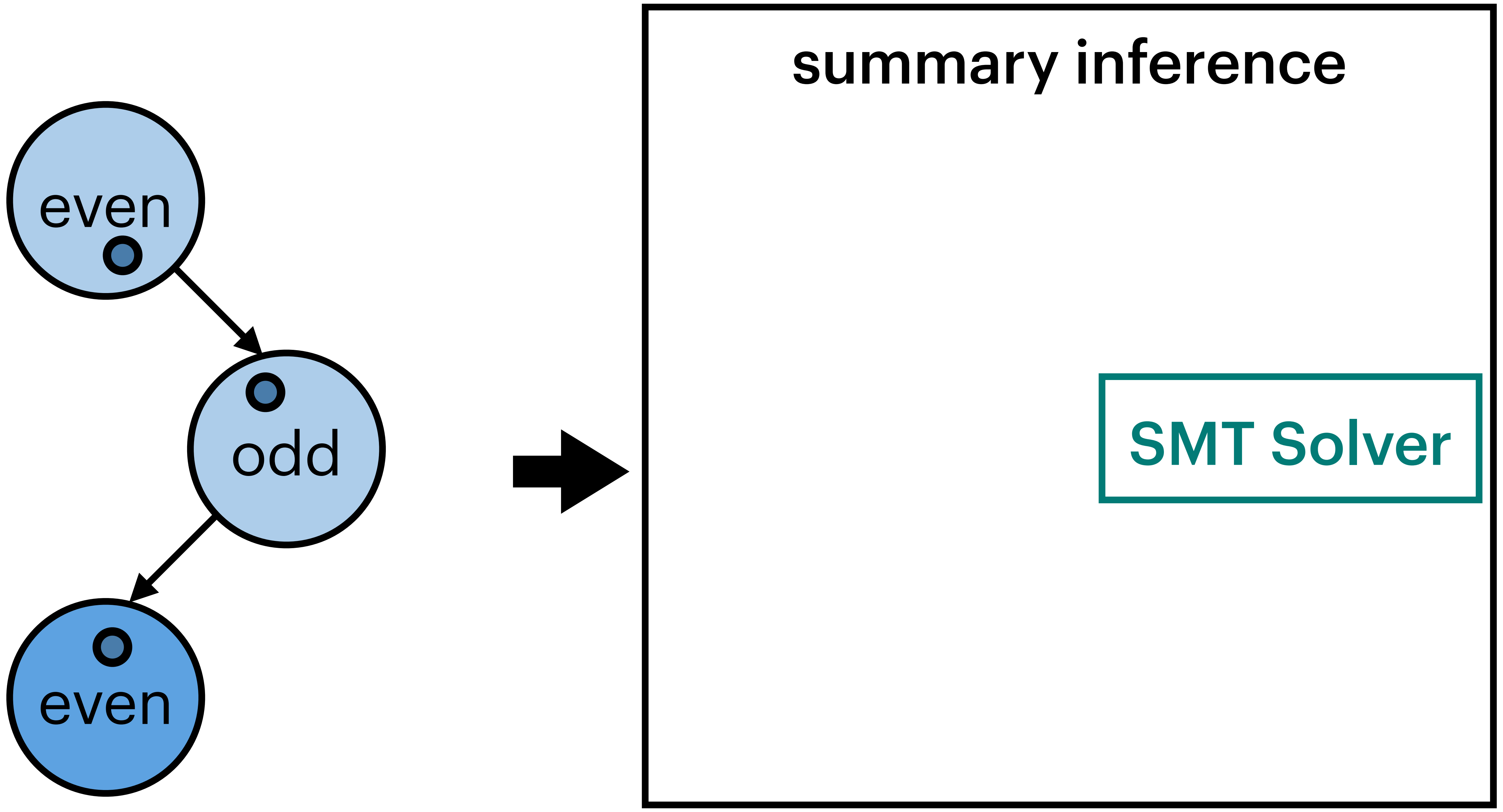
Under-Approximate Summary Inference

under-approximate environment



Under-Approximate Summary Inference

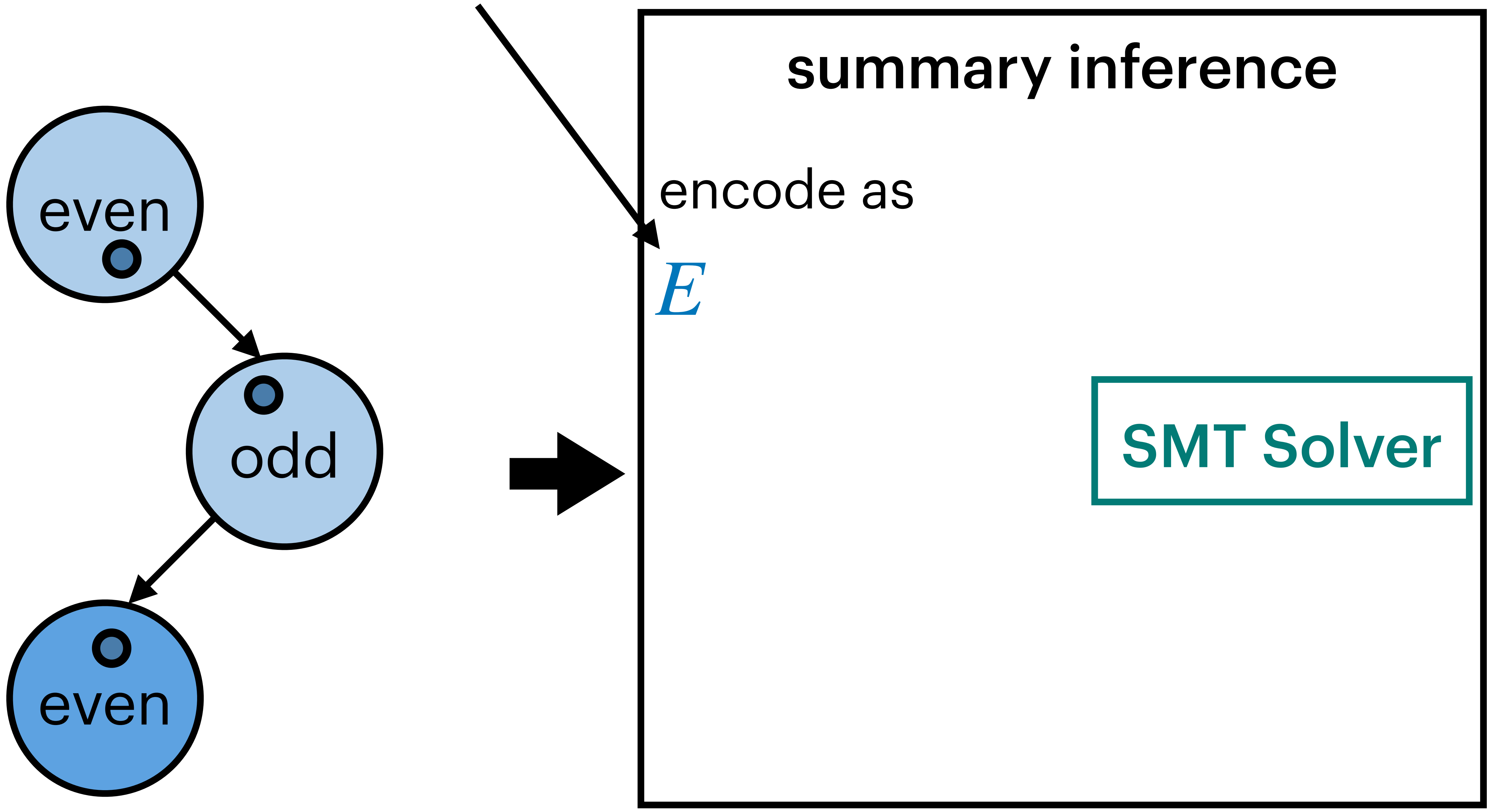
under-approximate environment



under-approximate target

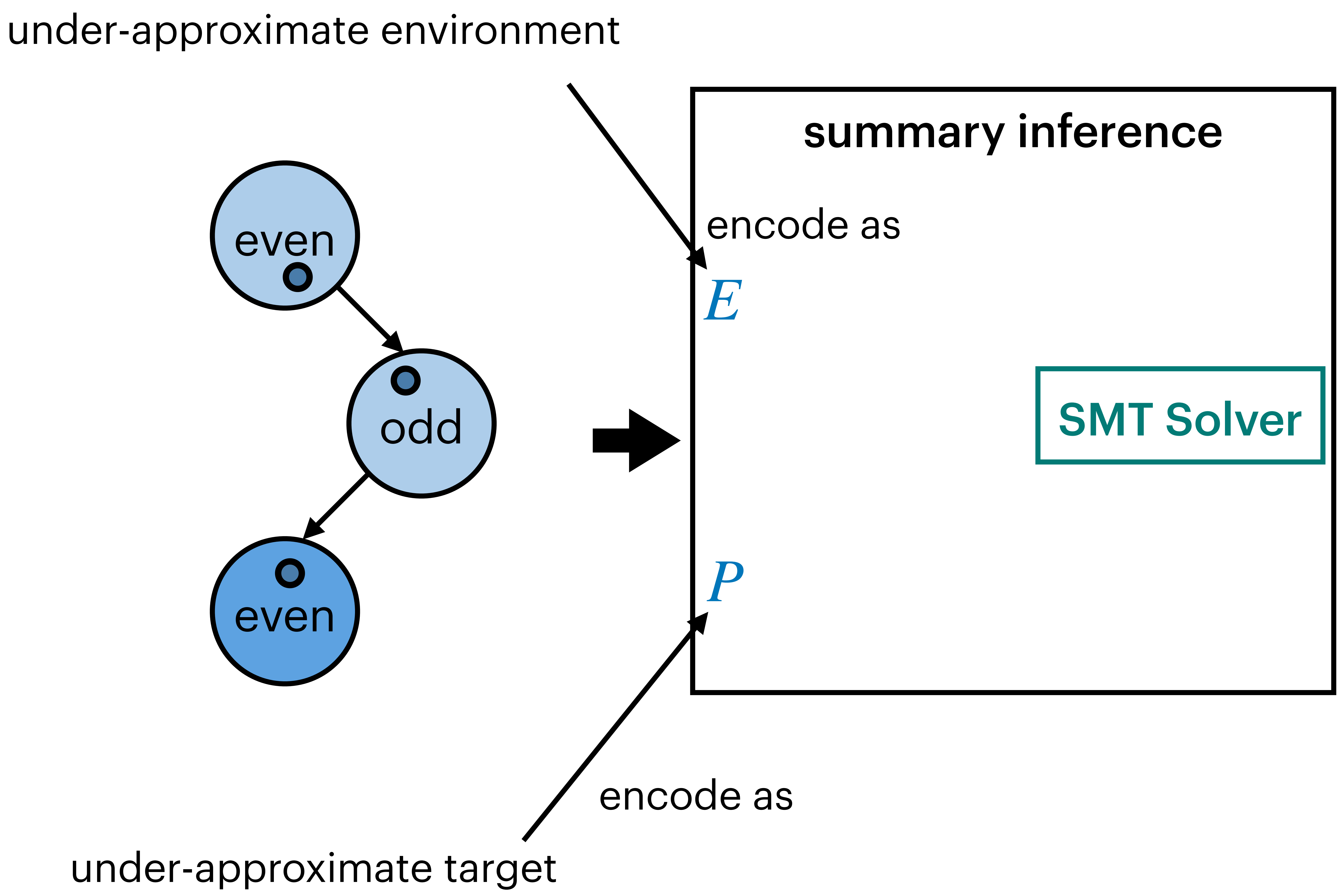
Under-Approximate Summary Inference

under-approximate environment

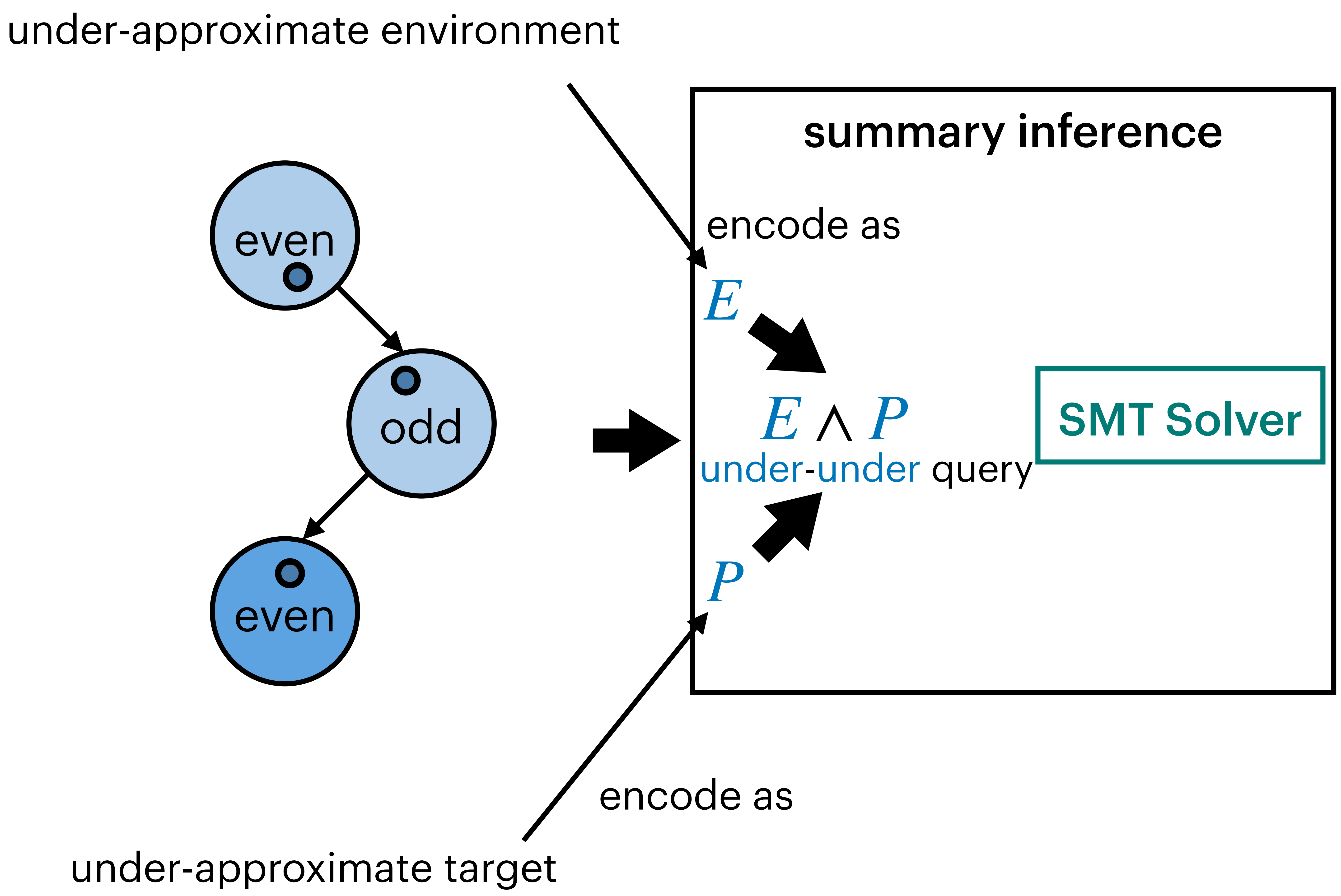


under-approximate target

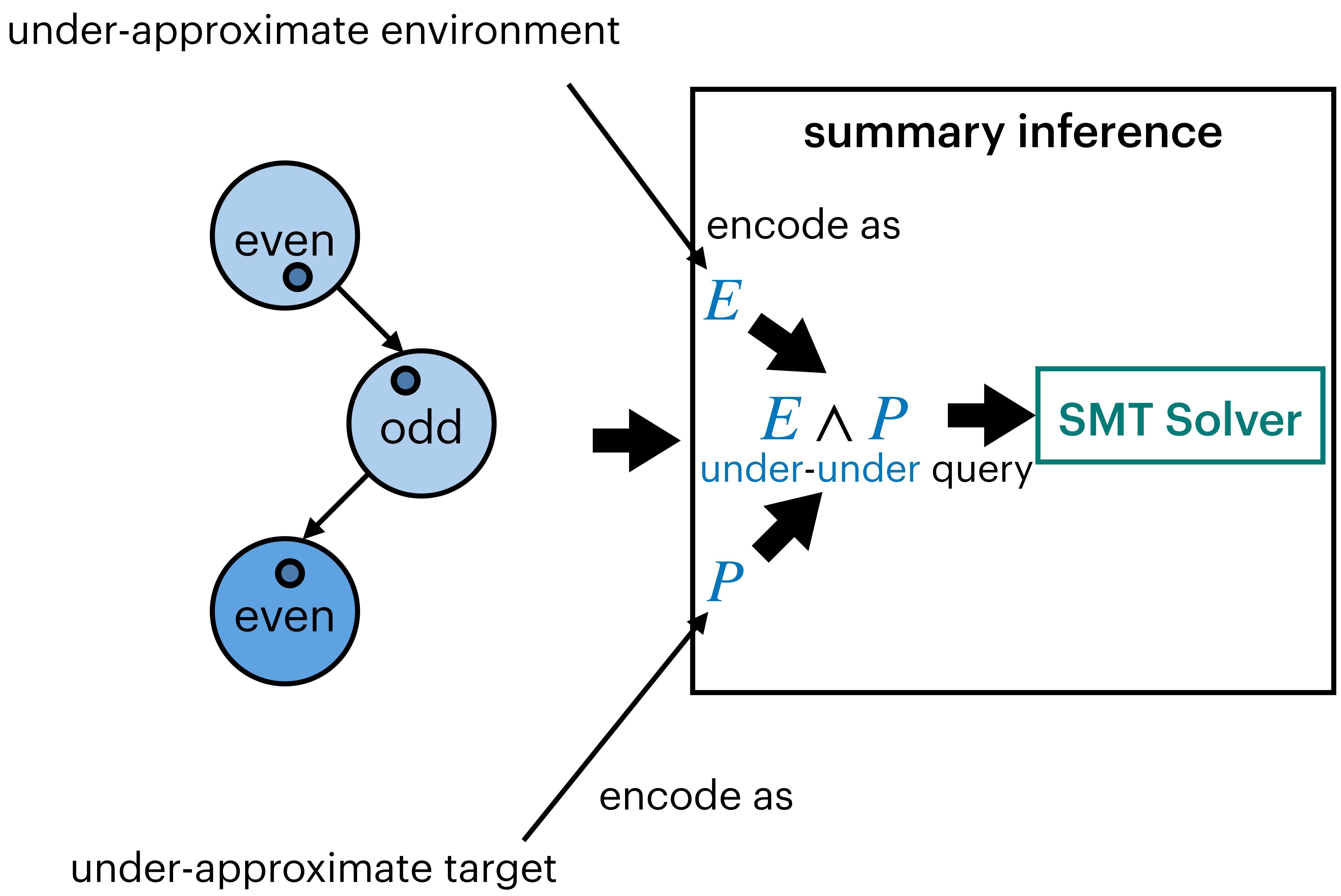
Under-Approximate Summary Inference



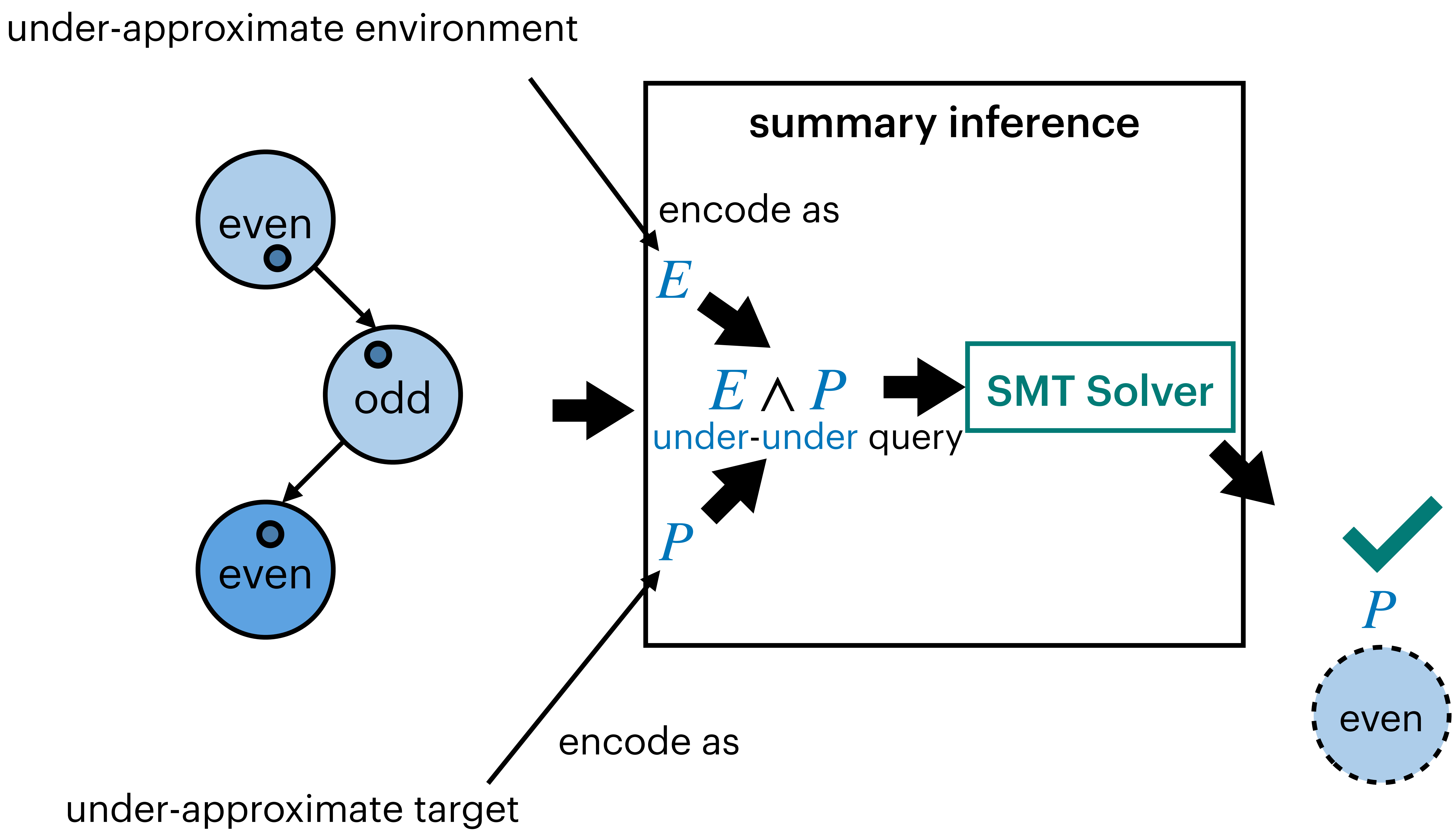
Under-Approximate Summary Inference



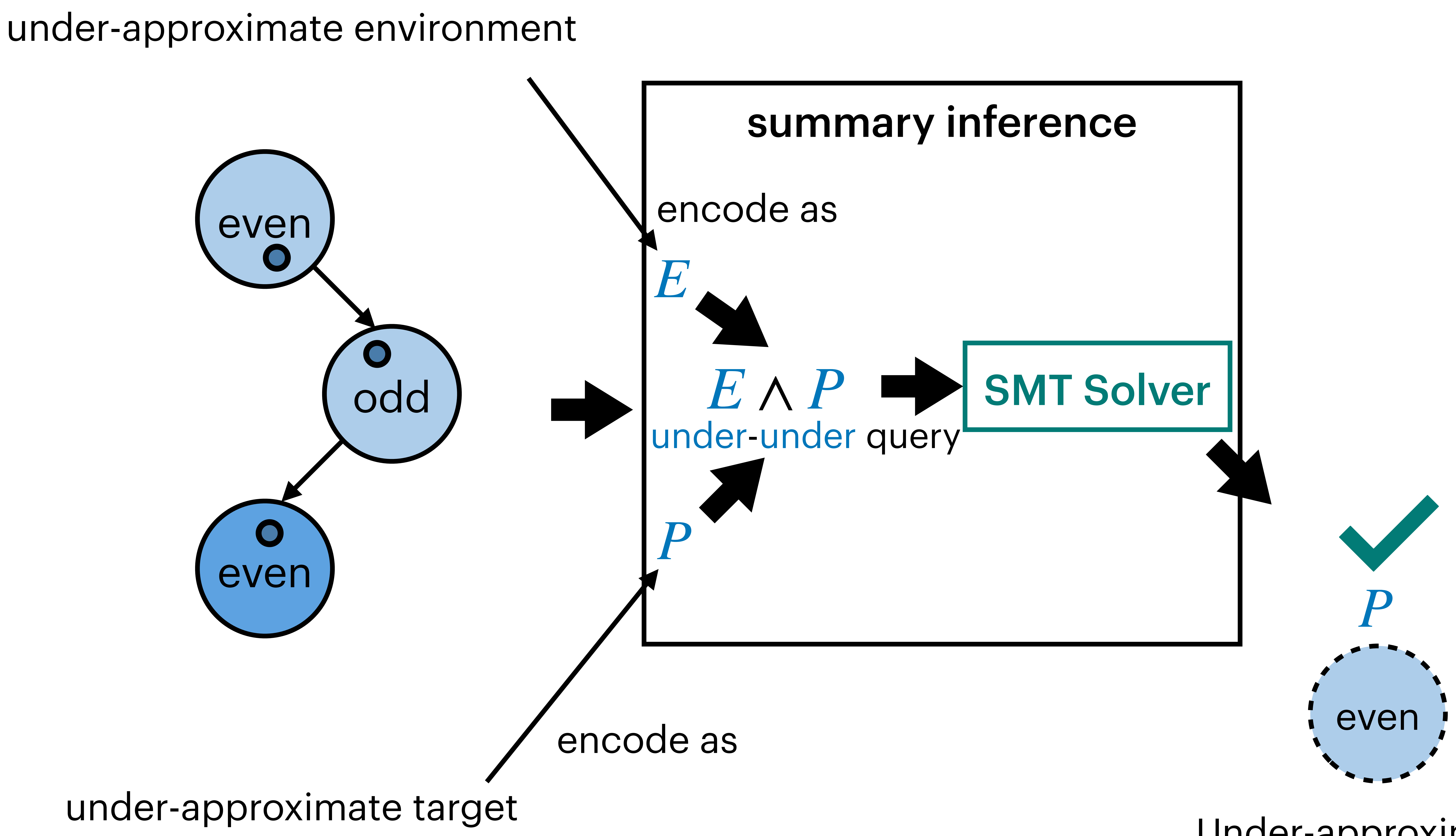
Under-Approximate Summary Inference



Under-Approximate Summary Inference

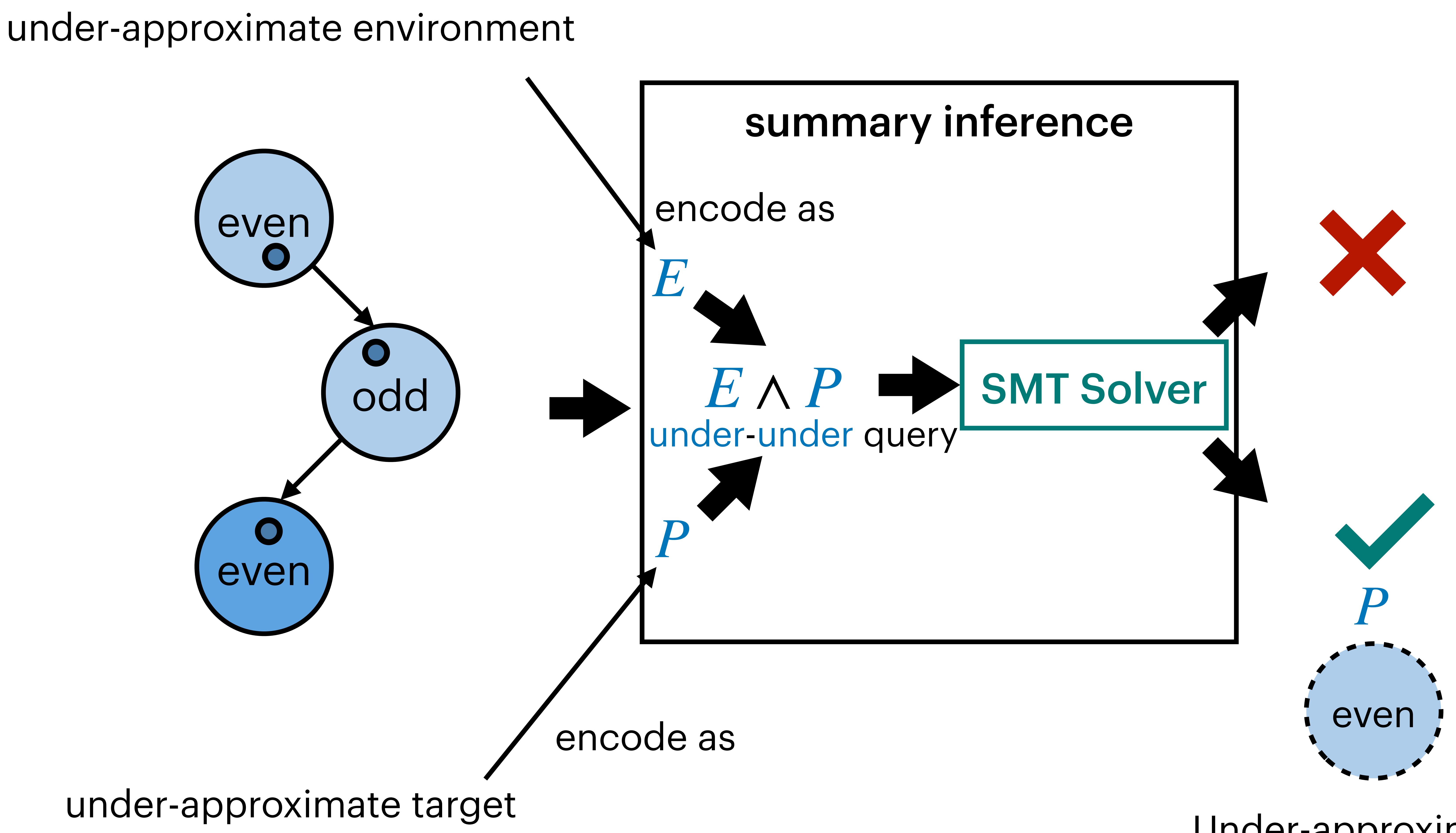


Under-Approximate Summary Inference



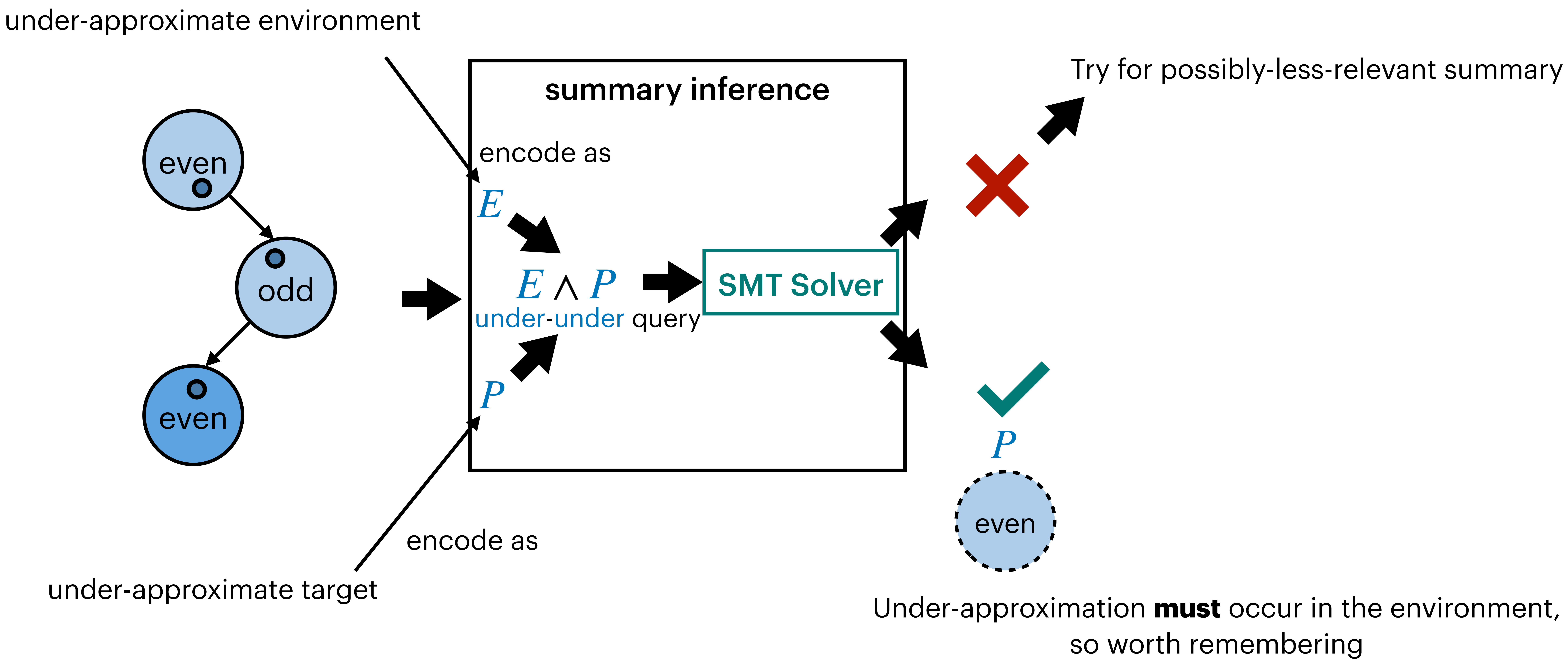
Under-approximation **must** occur in the environment, so worth remembering

Under-Approximate Summary Inference

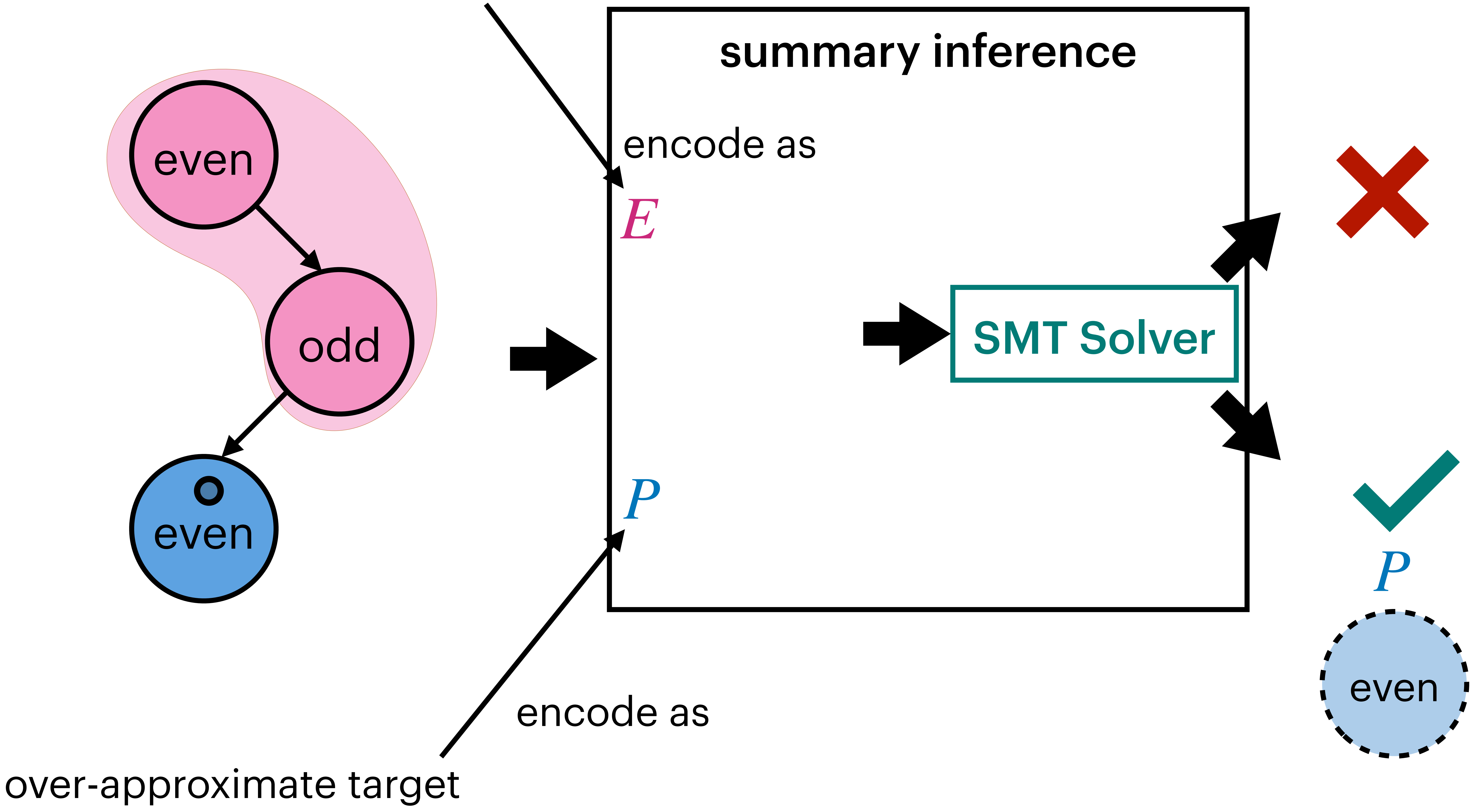


Under-approximation **must** occur in the environment, so worth remembering

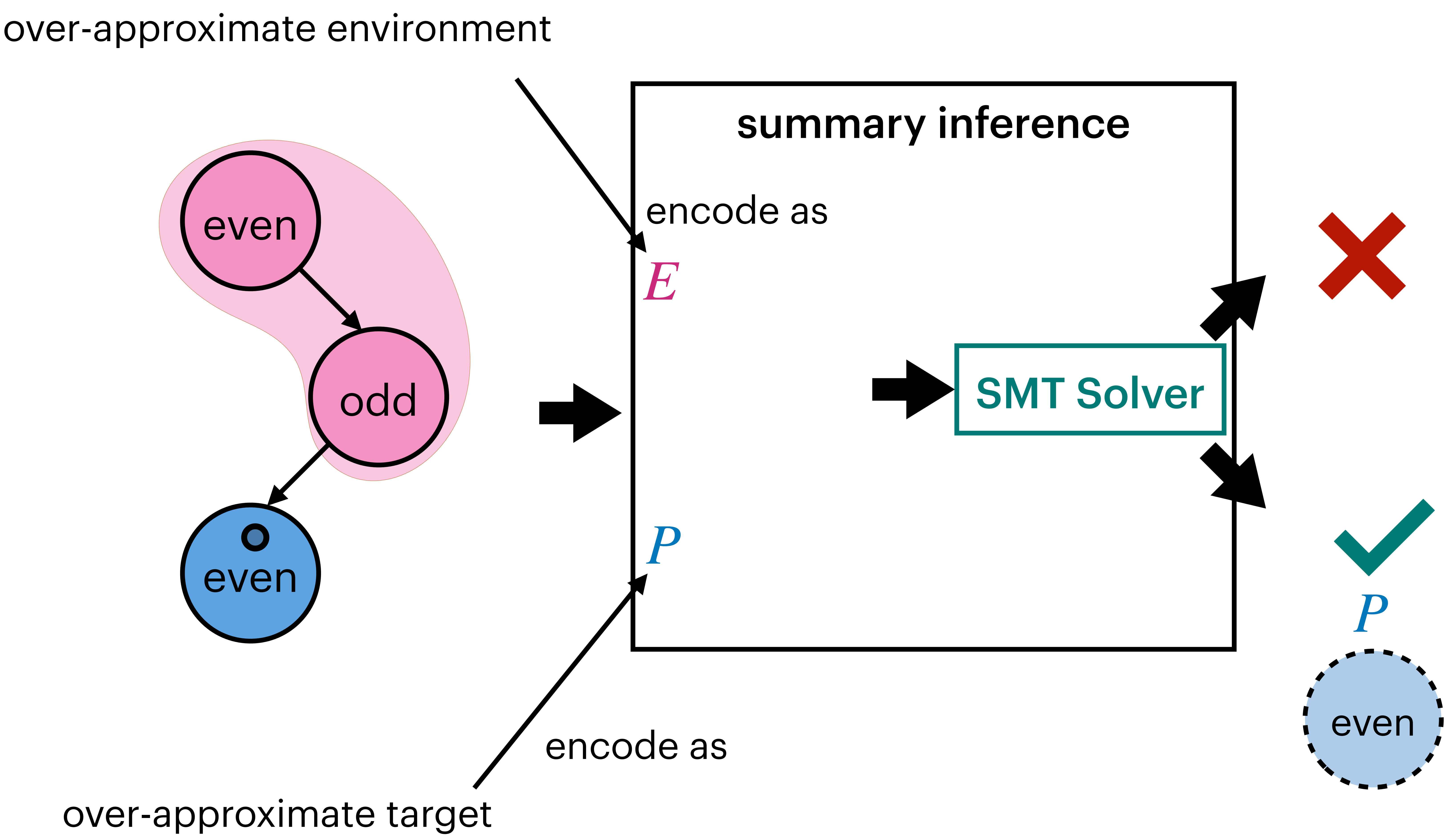
Under-Approximate Summary Inference



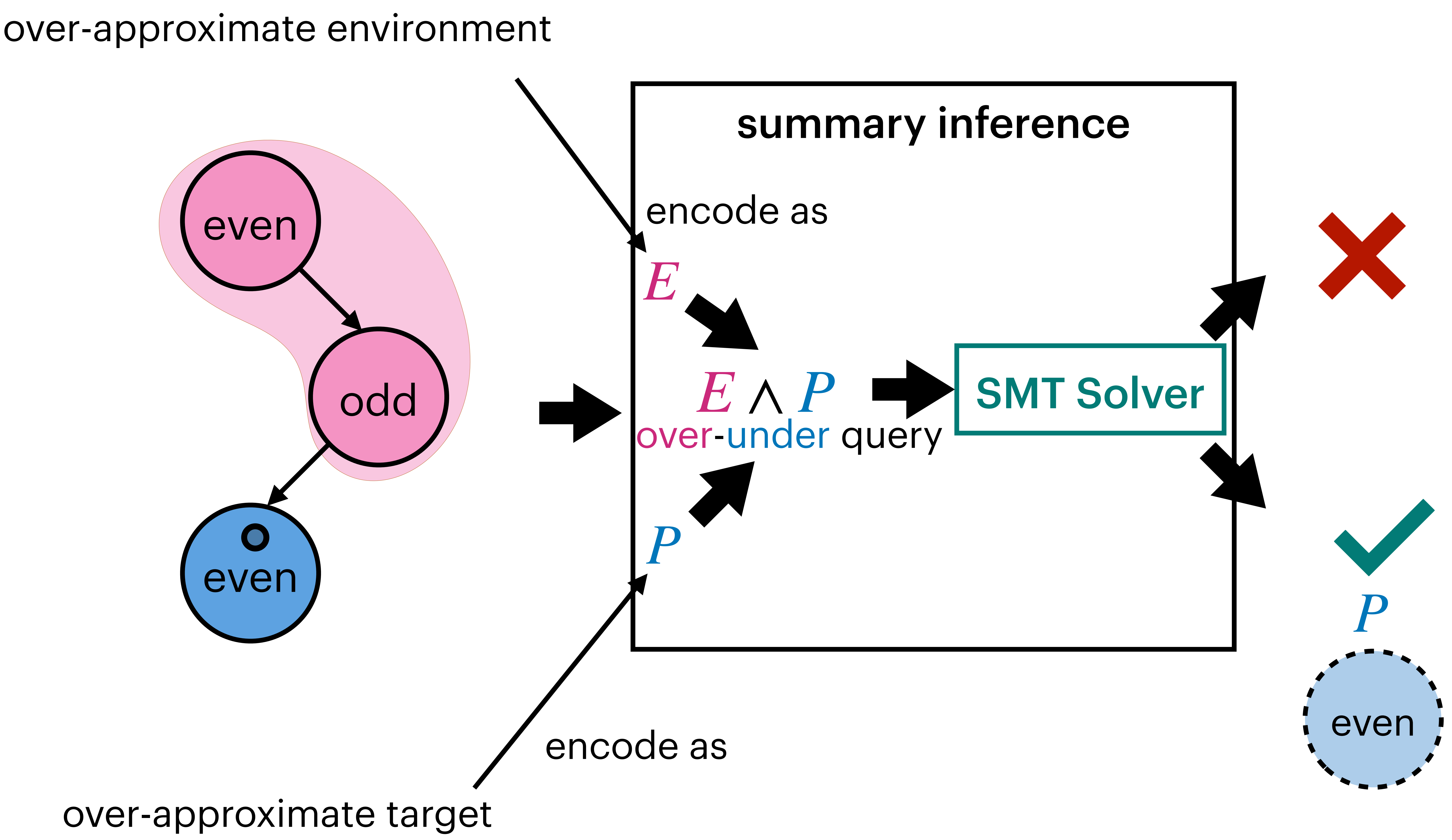
Under-Approximate Summary Inference



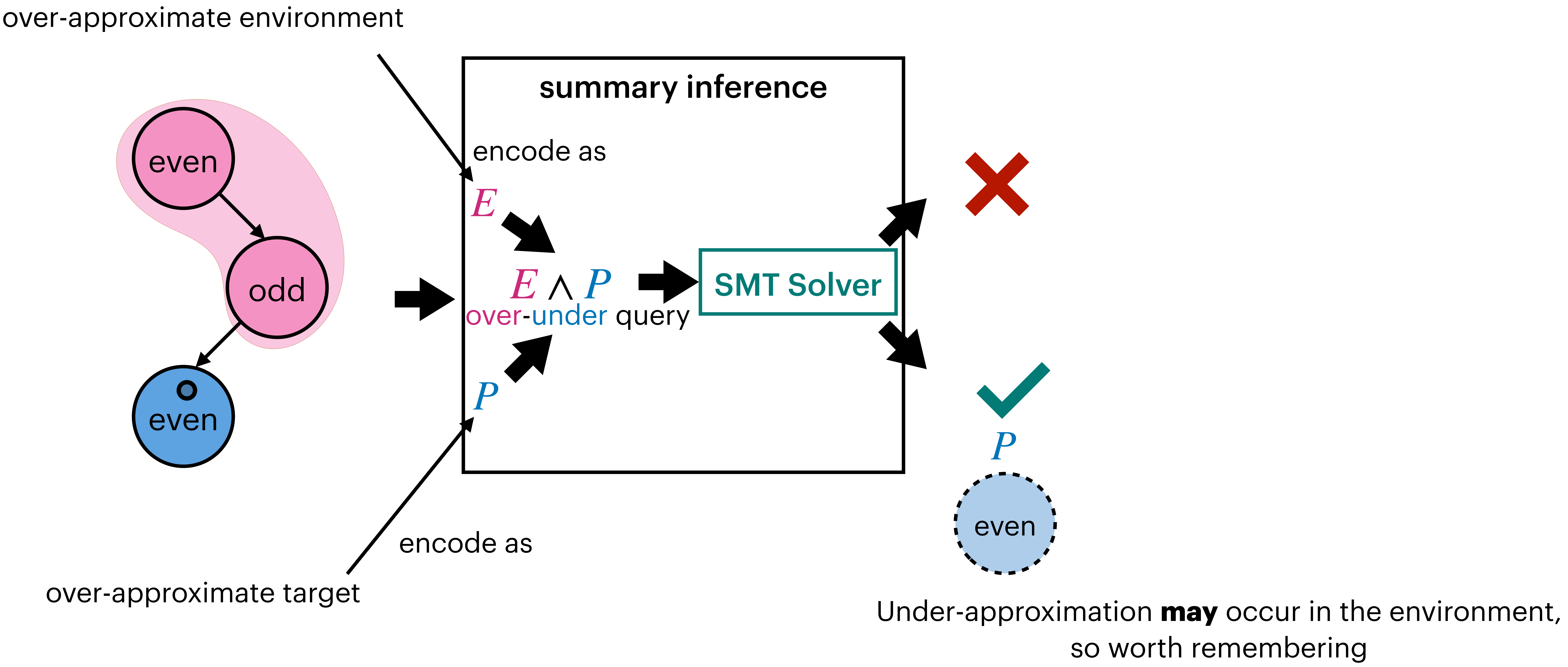
Under-Approximate Summary Inference



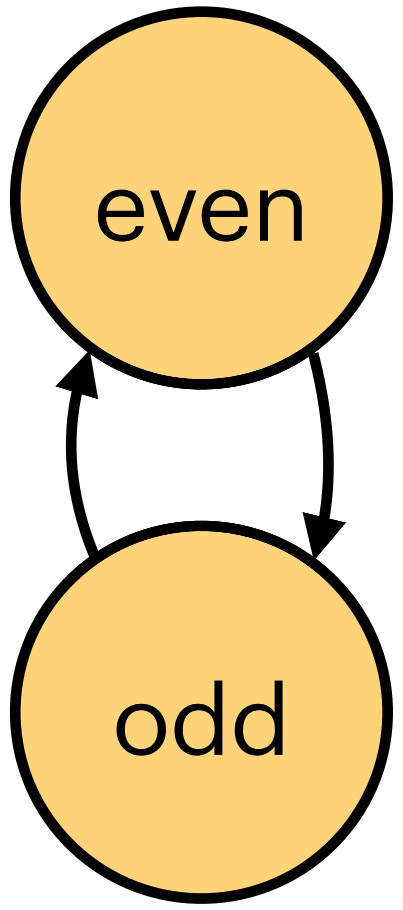
Under-Approximate Summary Inference



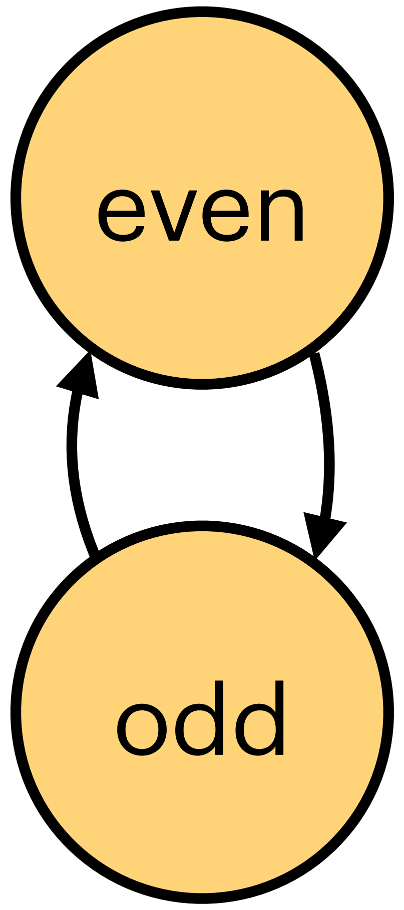
Under-Approximate Summary Inference



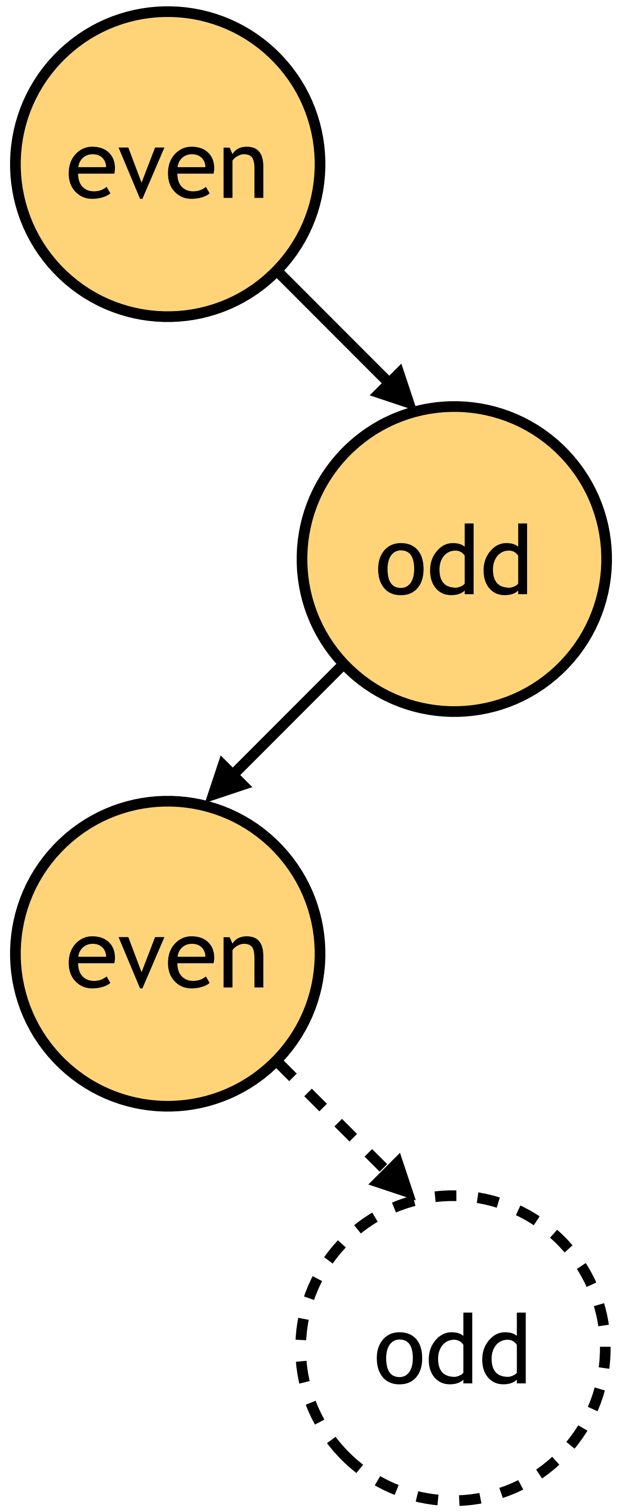
Mutual Recursion



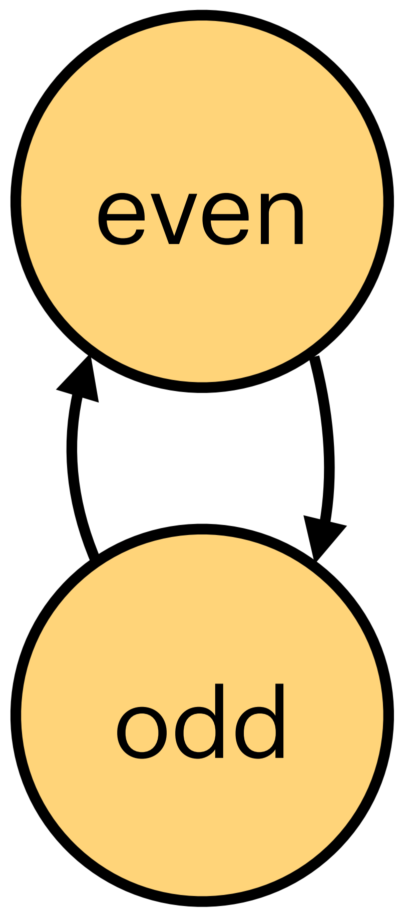
Mutual Recursion



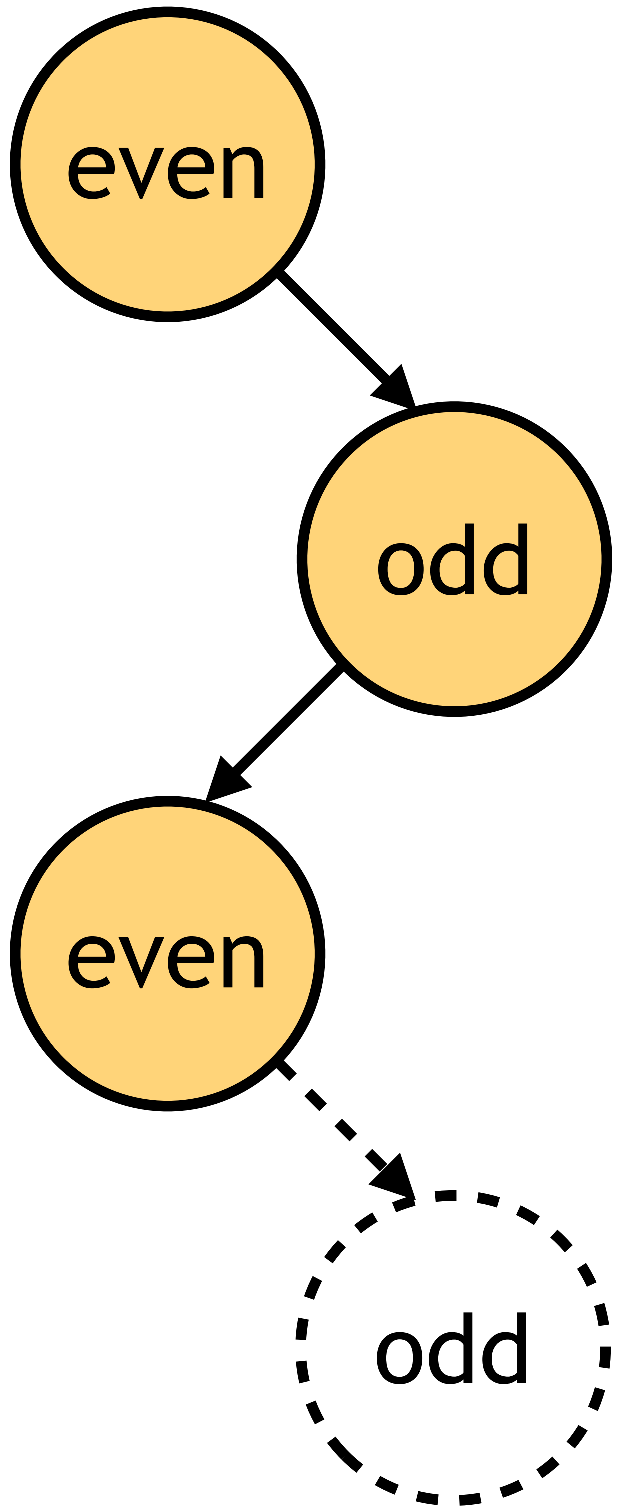
Unfolding:



Mutual Recursion

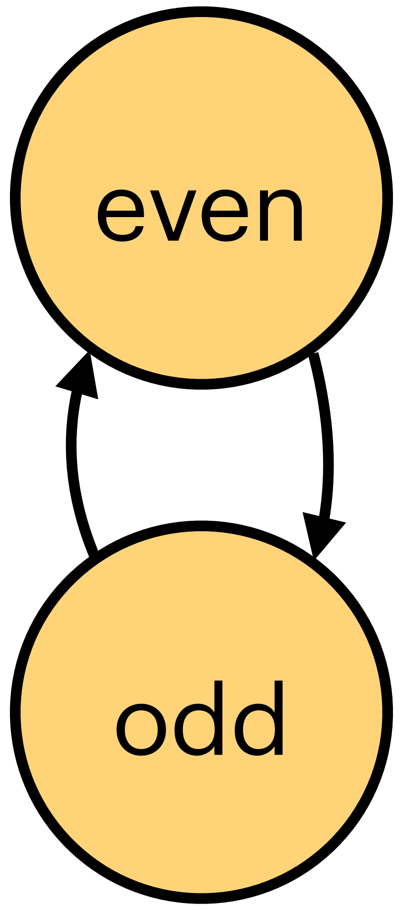


Unfolding:

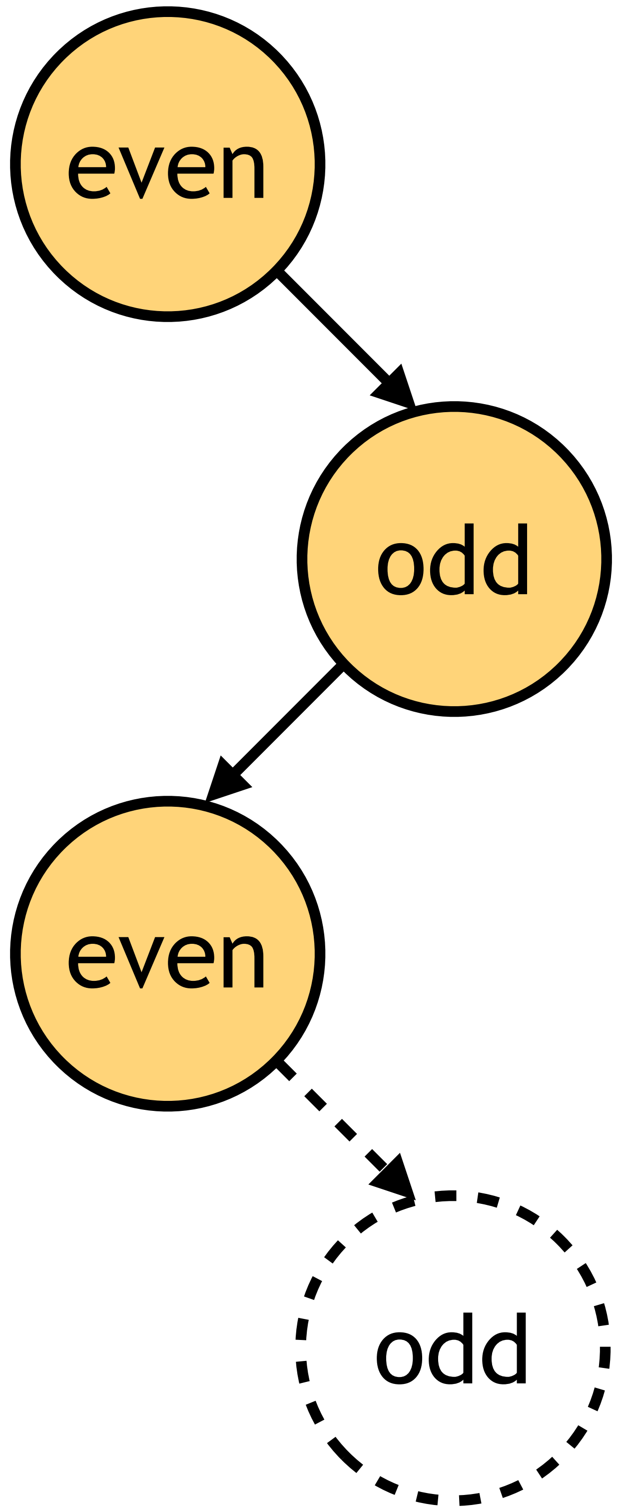


How much to unfold?

Mutual Recursion



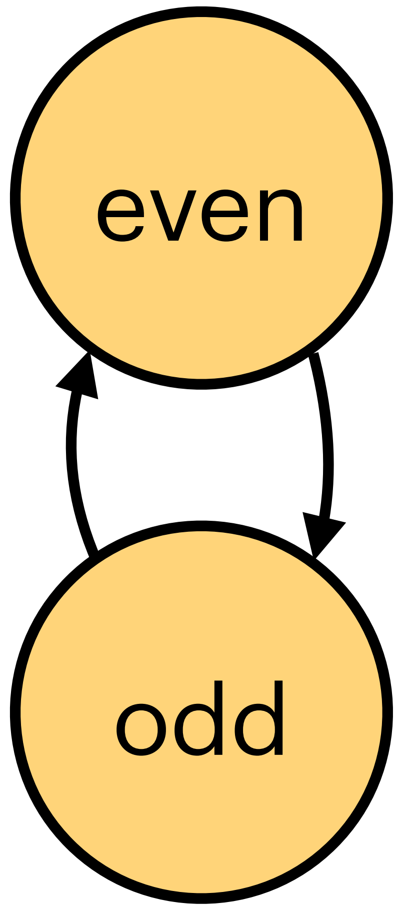
Unfolding:



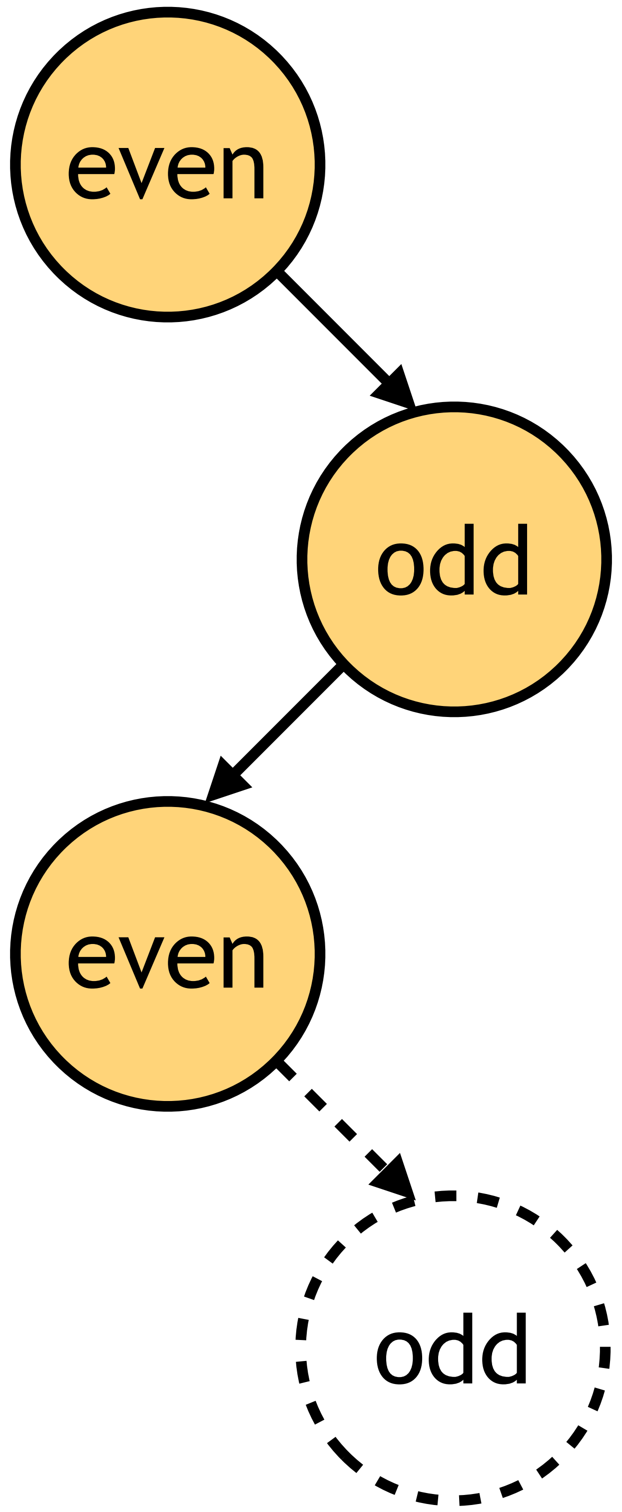
How much to unfold?

Can’t do induction directly on even

Mutual Recursion



Unfolding:

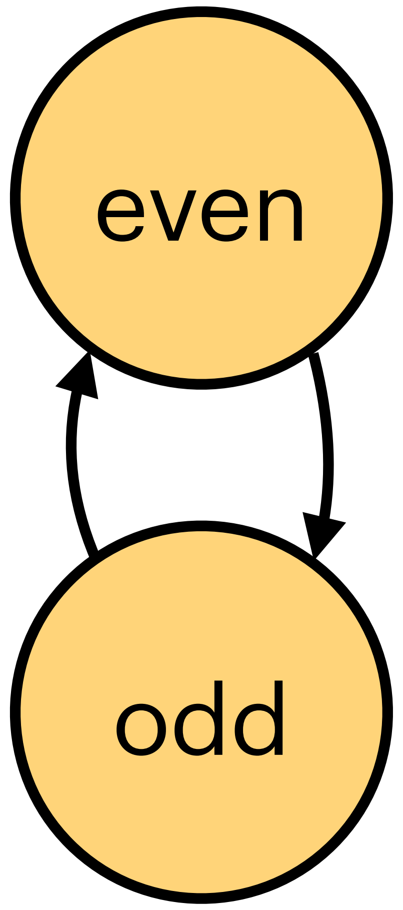


Inlining:

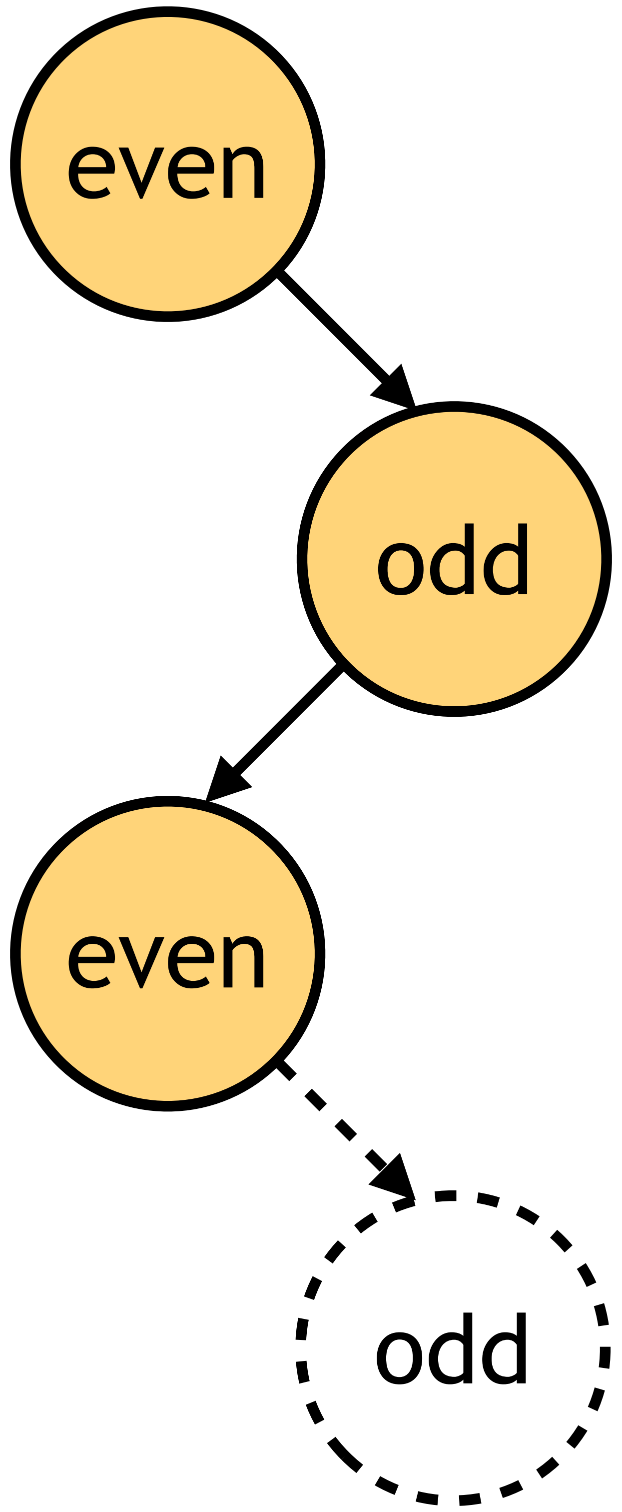
How much to unfold?

Can't do induction directly on even

Mutual Recursion



Unfolding:



How much to unfold?

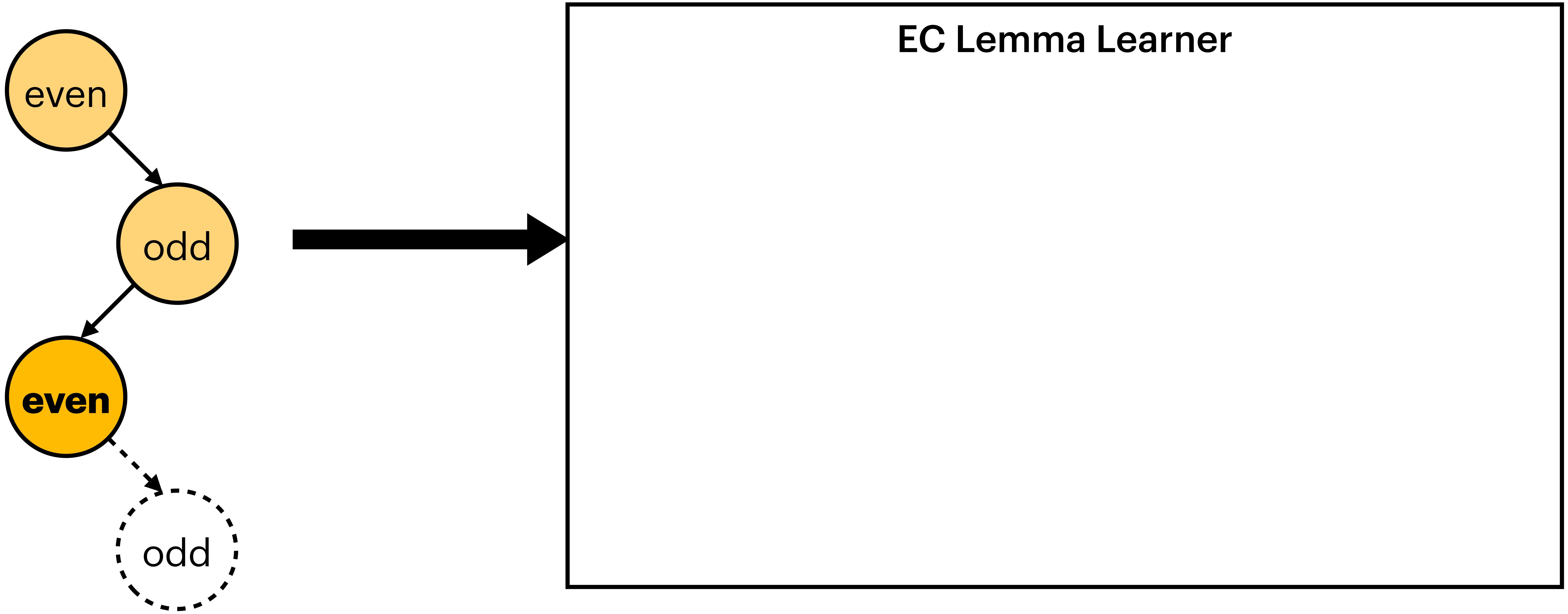
Can’t do induction directly on even

Inlining:

No summary for odd

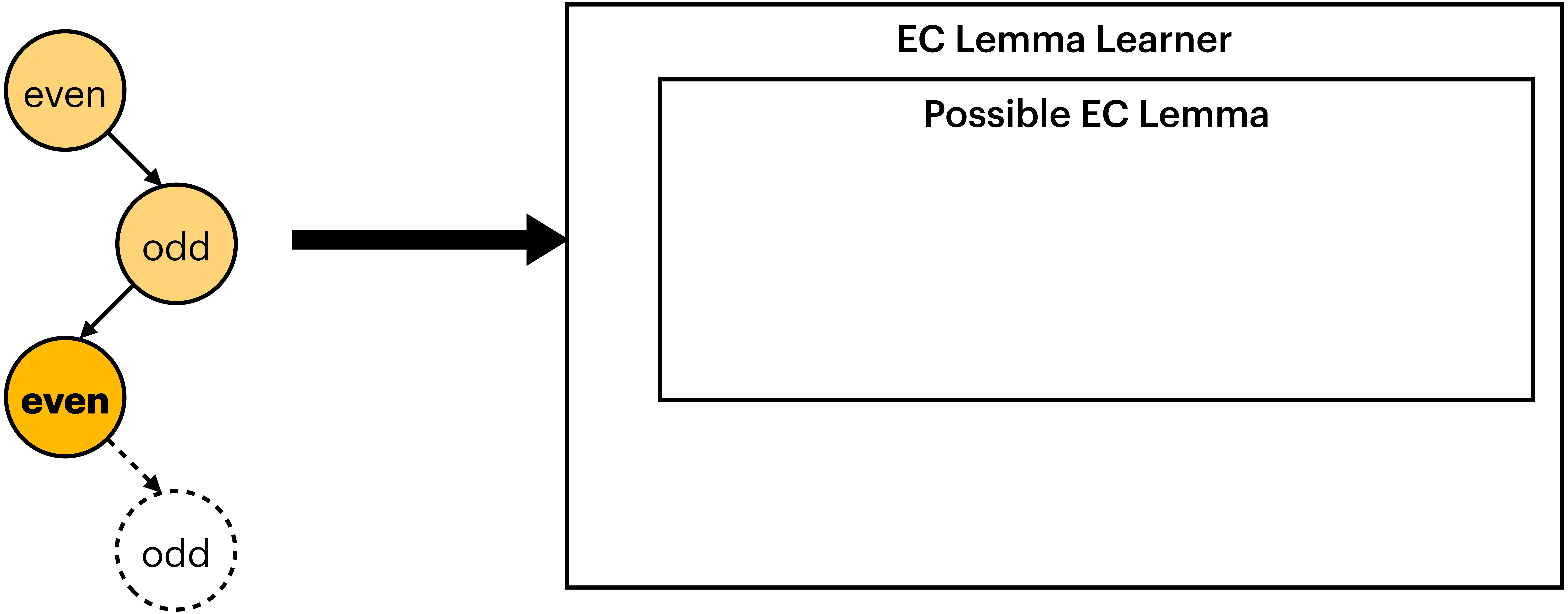
Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



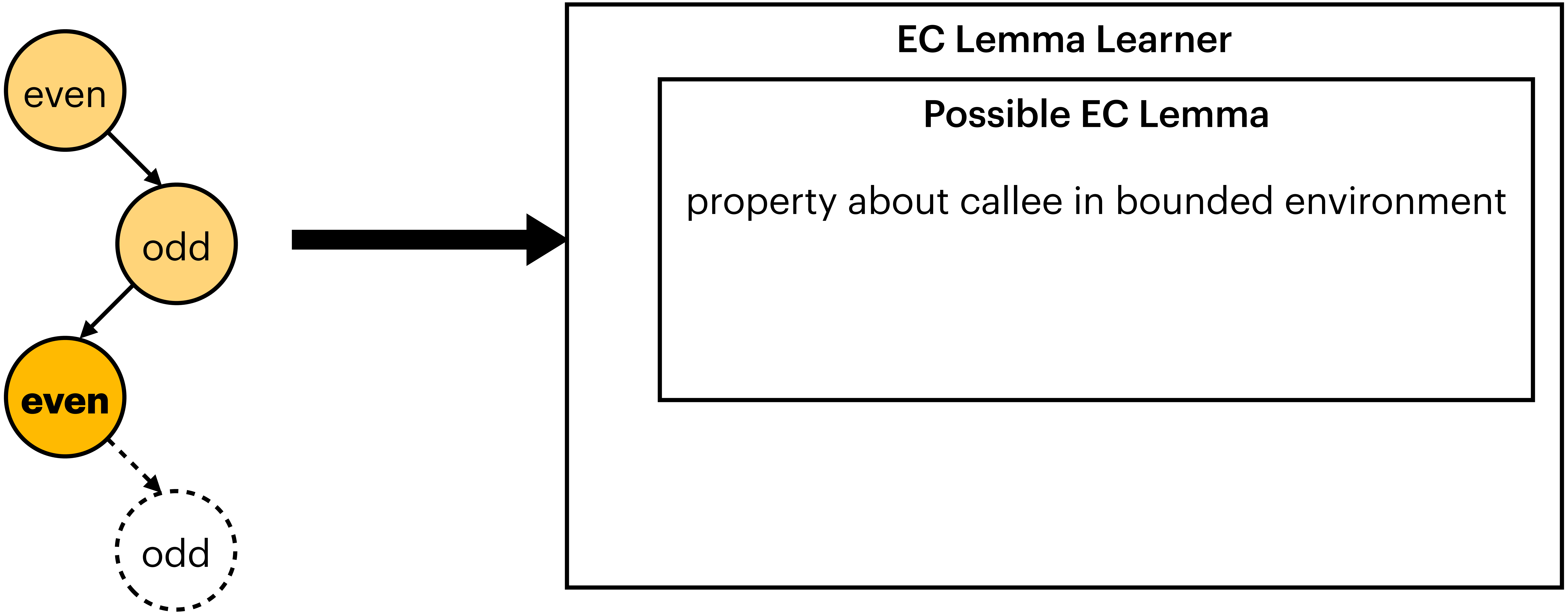
Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



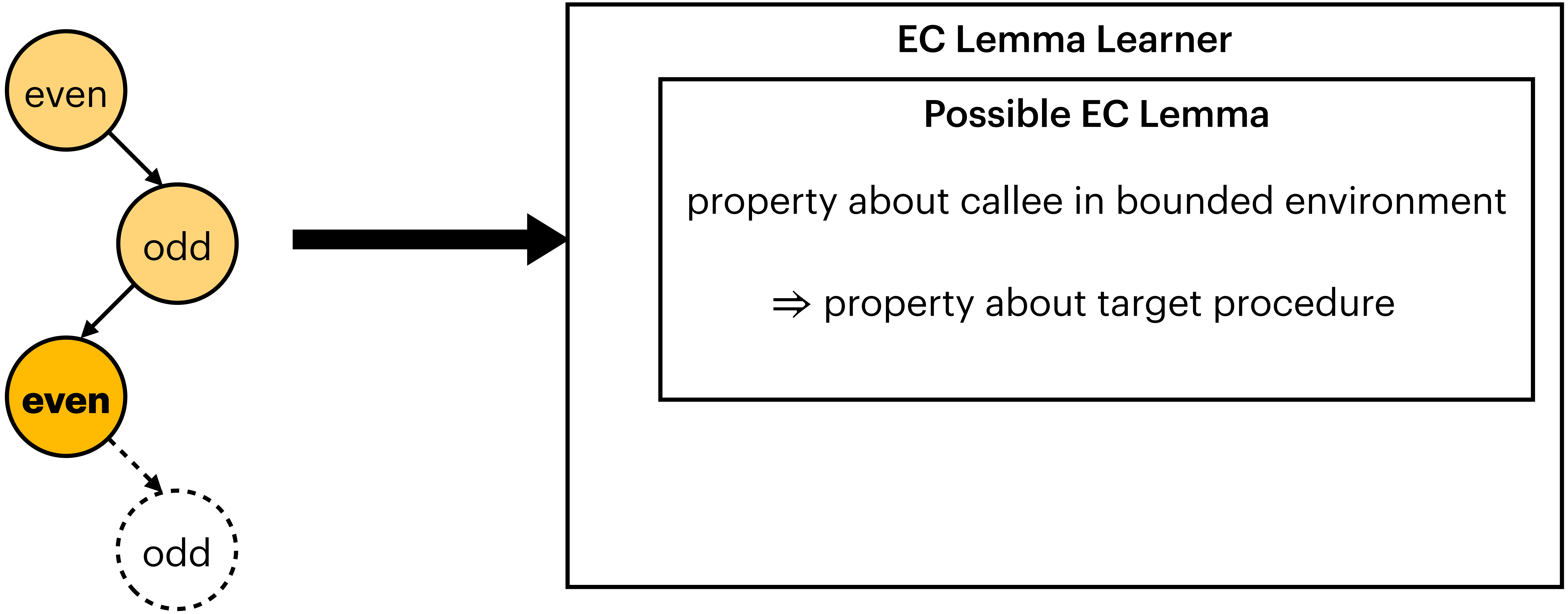
Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



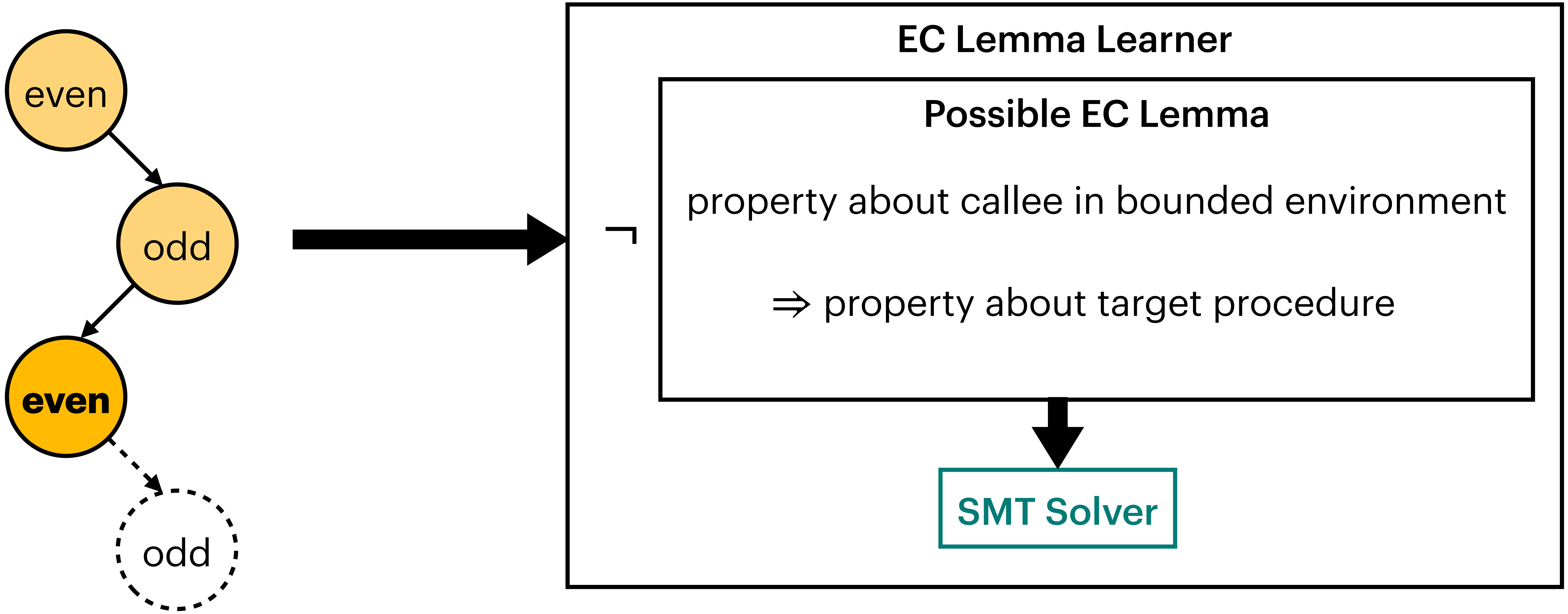
Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



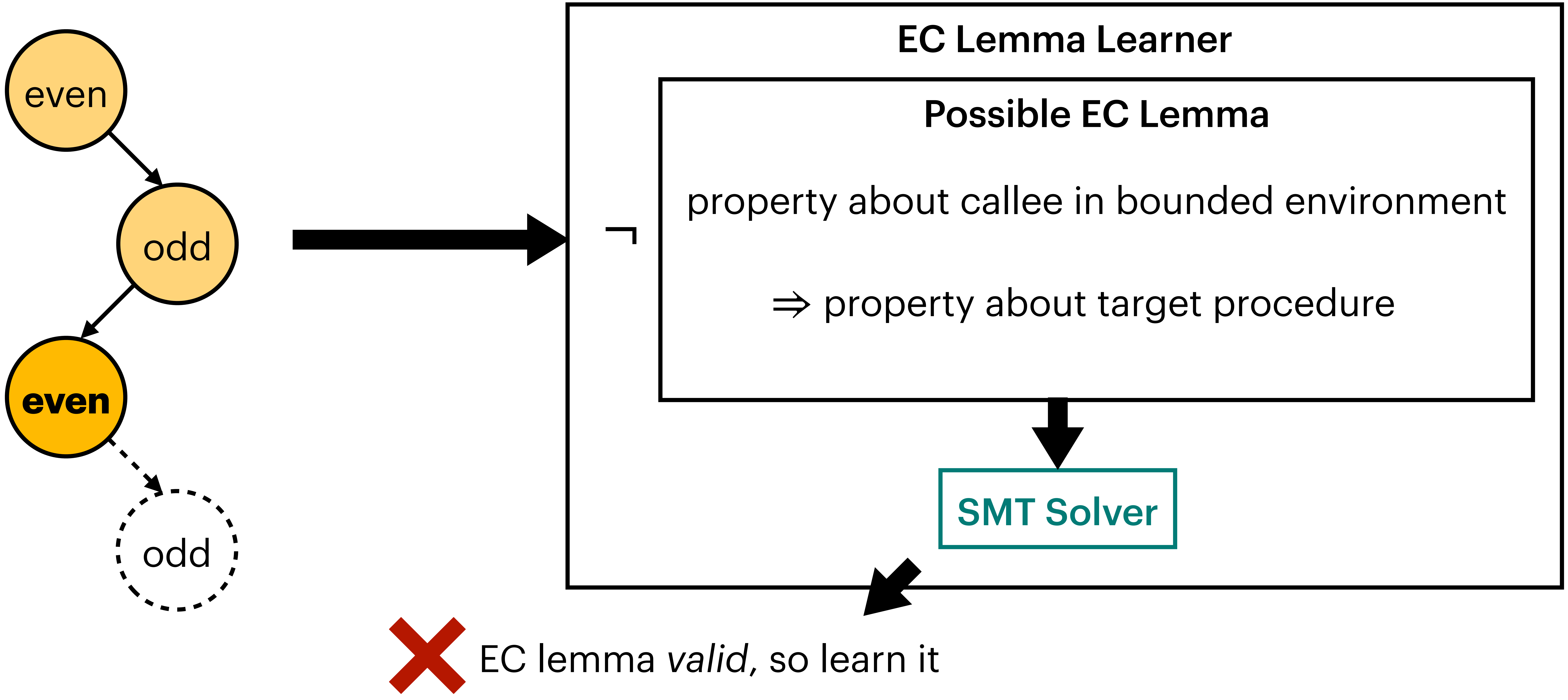
Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



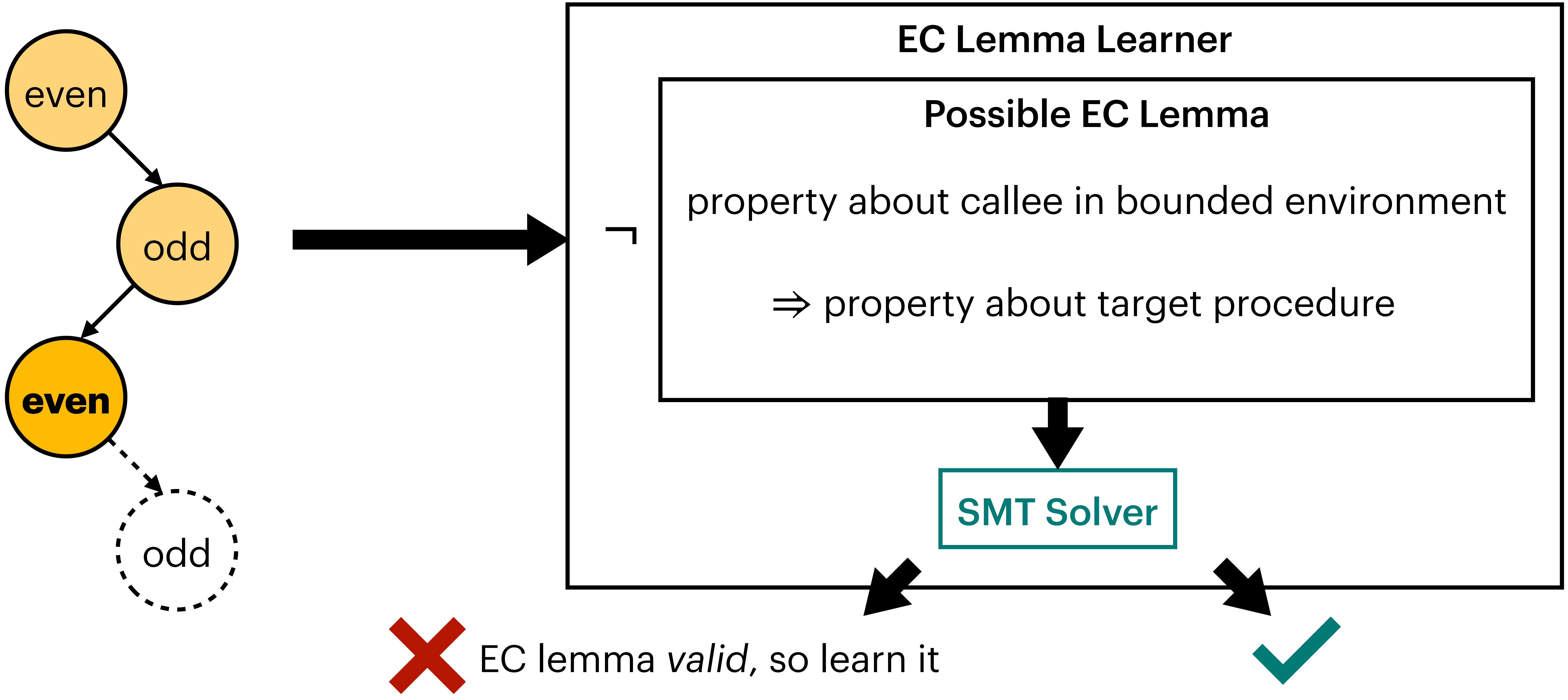
Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



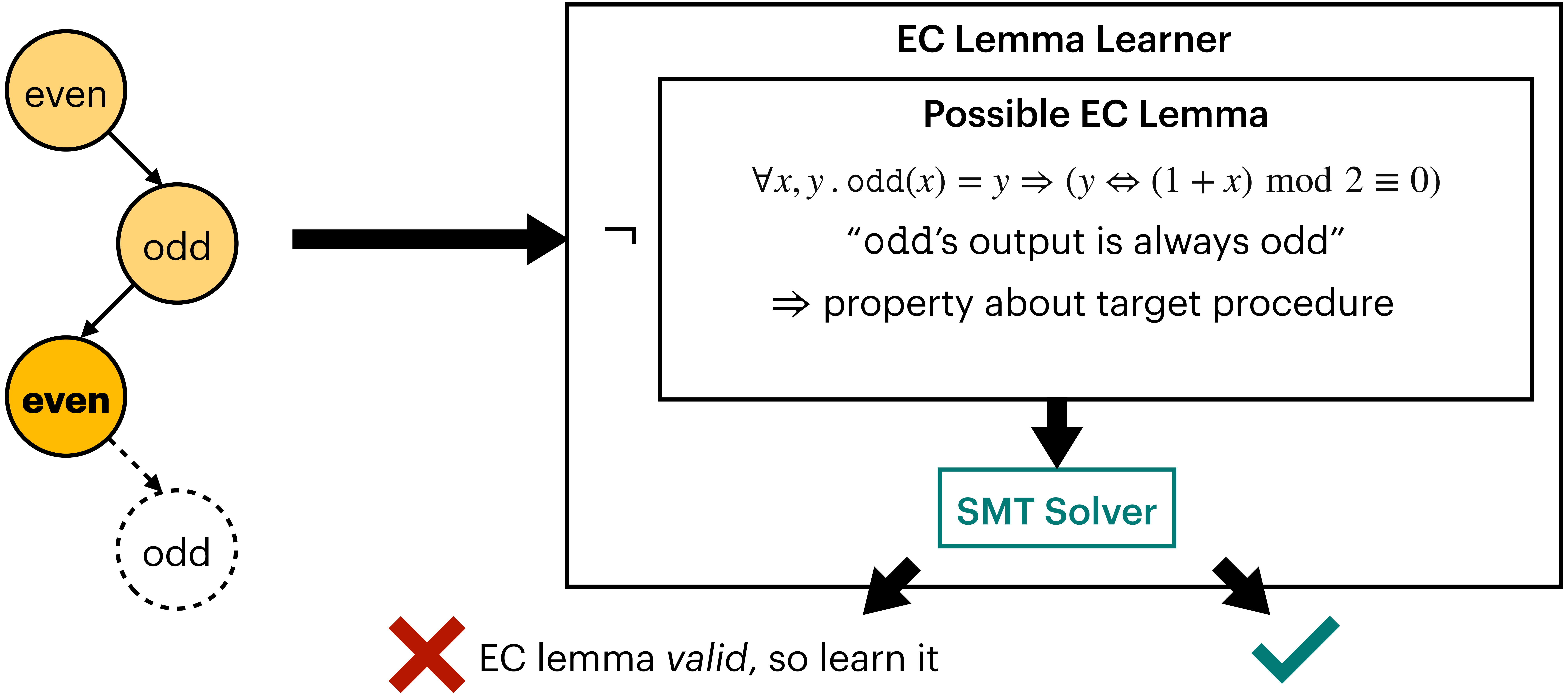
Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



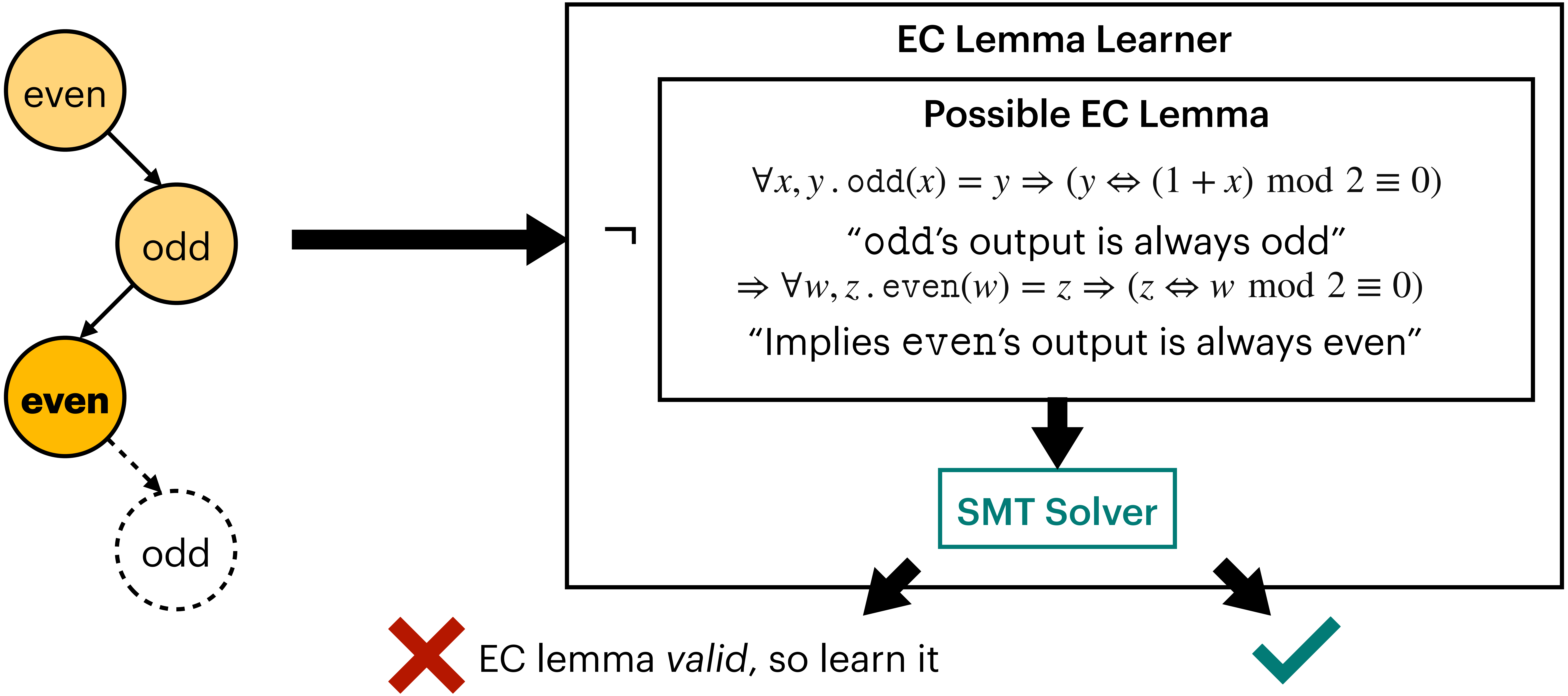
Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



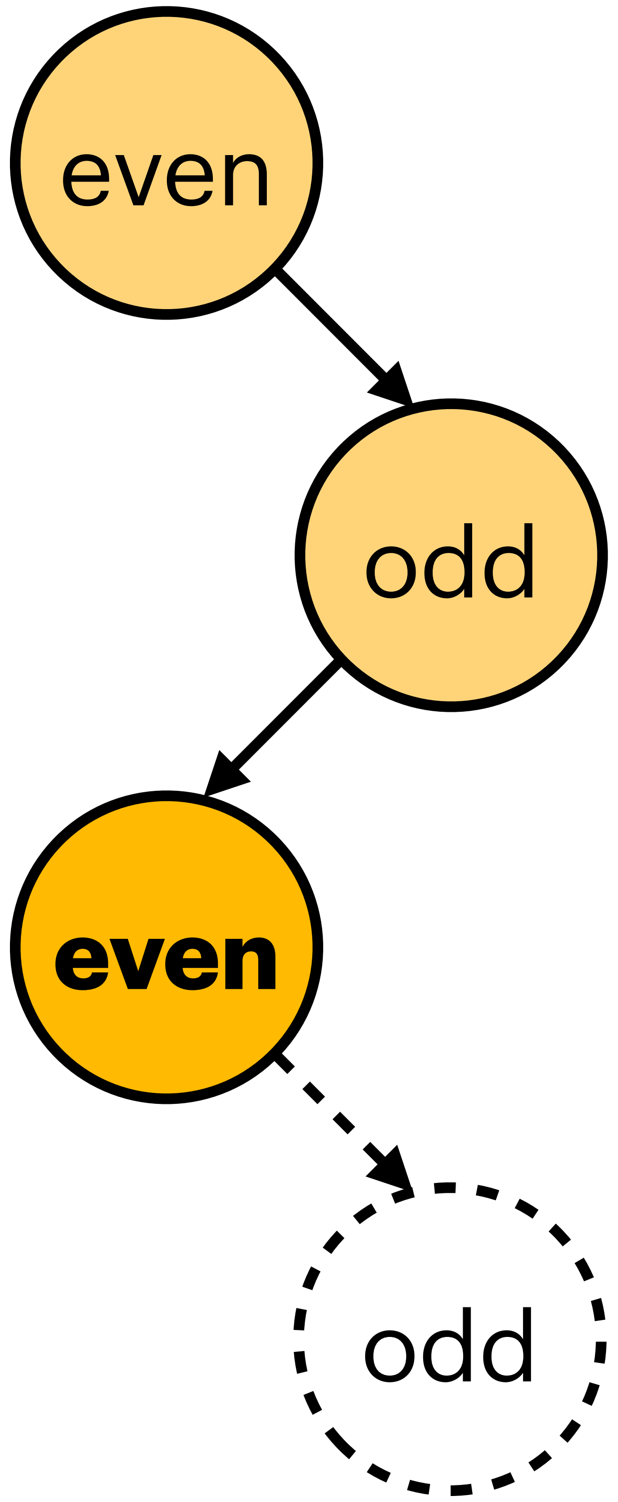
Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



Environment-Callee (EC) Lemmas

Express relationships between summaries of procedures on the same call path in a program



EC Lemma Learner

Possible EC Lemma

$$\neg \quad \forall x, y\,.\,\texttt{odd}(x) = y \Rightarrow (y \Leftrightarrow (1 + x) \bmod 2 \equiv 0)$$

"odd's output is always odd"

$$\Rightarrow \forall w, z\,.\,\texttt{even}(w) = z \Rightarrow (z \Leftrightarrow w \bmod 2 \equiv 0)$$

"Implies even's output is always even"

SMT Solver

✗ ✓

EC lemma *valid*, so learn it

learn: "odd's output always being odd implies that even's output is always even"