Property Directed Inference of Relational Invariants

Dmitry Mordvinov[1] **Grigory Fedyukovich**[2]

[1] JetBrains Research,
Saint Petersburg State University,
Russia

[2] Florida State University,
USA

FMCAD, Oct 25, 2019, San Jose, CA

Motivation

- Verification of program safety
- Constrained Horn Clauses (CHC):
  - Representation of a wide range of programs and assertions
- Model checker = CHC-translation + CHC-solving
  - E.g., *SeaHorn* + *Spacer*, or *JayHorn* + *Eldarica*
- Invariant generation for CHC-solving:
  - Construct proofs iteratively using SMT solvers
- Relational verification
  - Program equivalence,
  - Non-interference, ...

Constrained Horn Clauses

Constrained Horn Clause (CHC):

$$\varphi \wedge f_1(\overline{x}_1) \wedge \ldots \wedge f_n(\overline{x}_n) \rightarrow H$$

- φ — constraint (quantifier-free formula in first-order constraints language)
- f_i — uninterpreted predicate symbol
- either $H = \bot$ or $H = f(\overline{x})$ for some f
- $\overline{x_i}, \overline{x}$ — vectors of (distinct) variables
- H is a head of clause ($head(C)$)
- the premise of the implication is the body of the clause

Systems of Constrained Horn Clauses

- CHC system S is an arbitrary set of clauses
- Clauses with heads $\perp$ are called queries, otherwise rules
  - Rules define the "semantics" of uninterpreted symbols
  - Queries define the specification
  - Queries can be rewritten to $f_1(\overline{x}_1) \wedge \ldots \wedge f_n(\overline{x}_n) \rightarrow P$
- A set of rules for uninterpreted symbol f is a set of clauses with applications of P in the head
- $body(f)$ is the disjunction of bodies of rules for f

Linear and Nonlinear CHCs

- CHC is called linear, if it has at most one application of uninterpreted symbol in its body
- Otherwise it is nonlinear

Linear:

$\varphi \wedge f(\overline{x}) \rightarrow h(\overline{x})$

Nonlinear:

$\varphi \wedge f(\overline{x}) \wedge g(\overline{x}) \rightarrow h(\overline{x}, \overline{y})$

Linear Queries

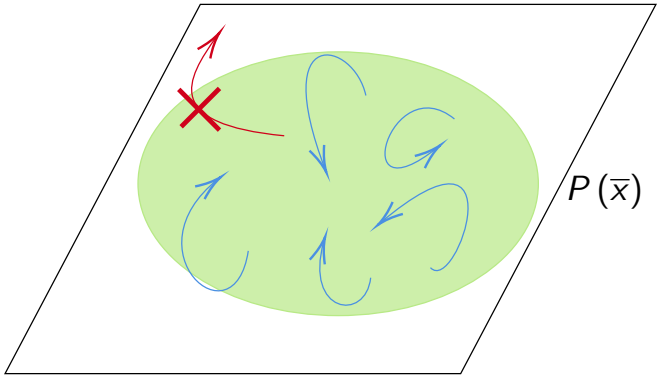
Linear query is a clause of the form

$$f(\overline{x}) \rightarrow P(\overline{x})$$

Uninterpreted symbol

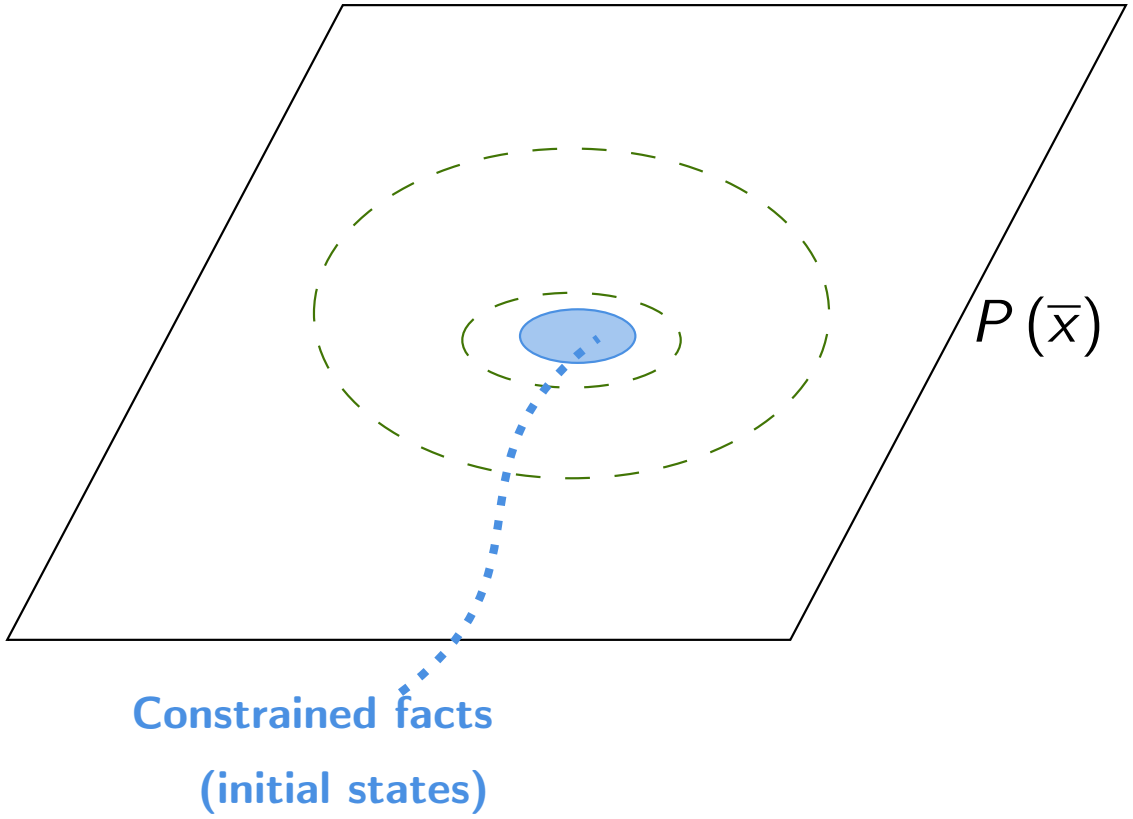
Safety property

To answer the query, we should find the subset of $P(\overline{x})$, which is inductive for f



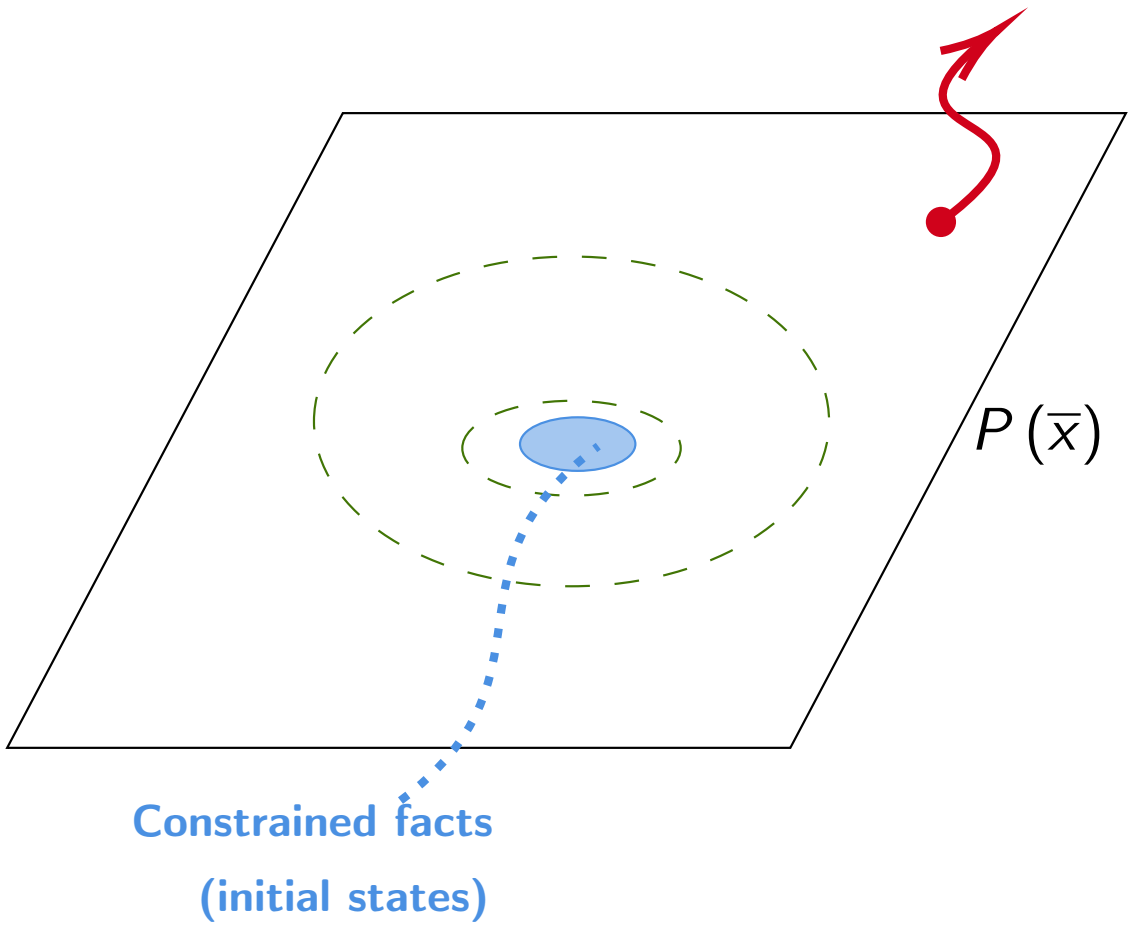
Property Directed Reachability

"SAT-based model checking without unrolling" (Bradley, 2011)
"Efficient Implementation of Property Directed Reachability" (Eén et al., 2011)



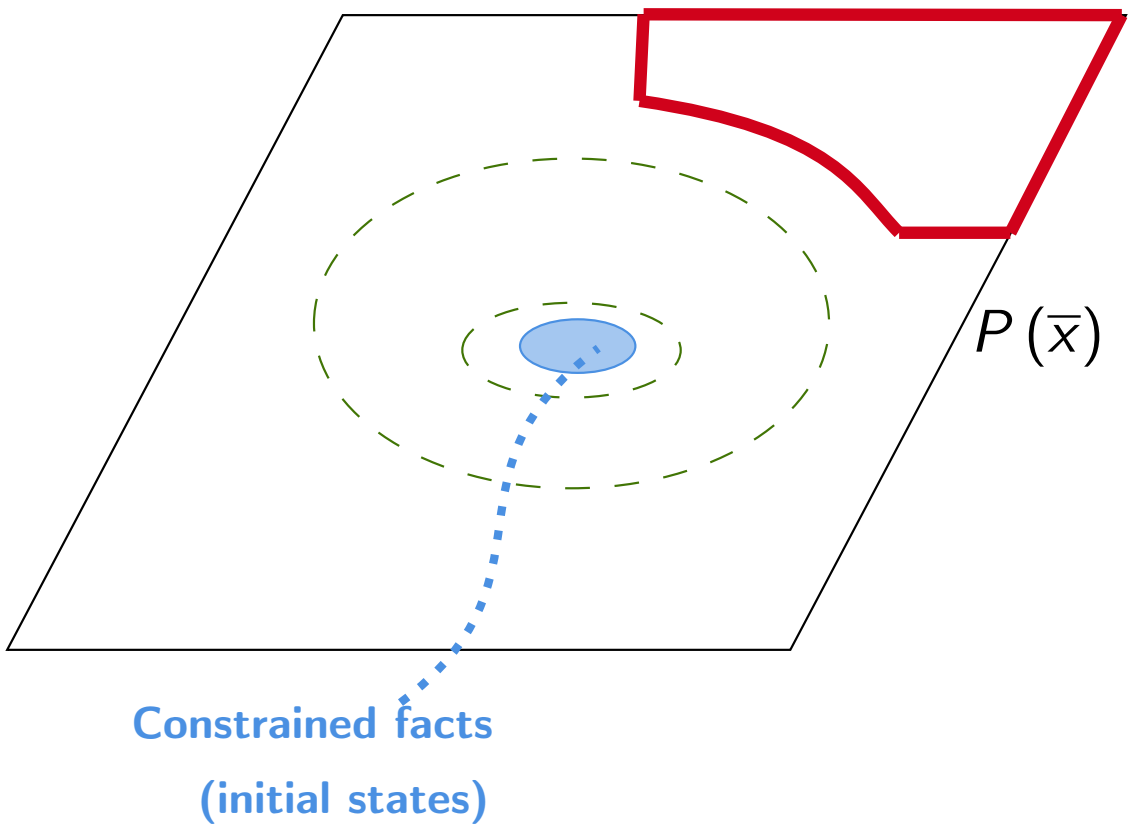
Property Directed Reachability

"SAT-based model checking without unrolling" (Bradley, 2011)
"Efficient Implementation of Property Directed Reachability" (Eén et al., 2011)



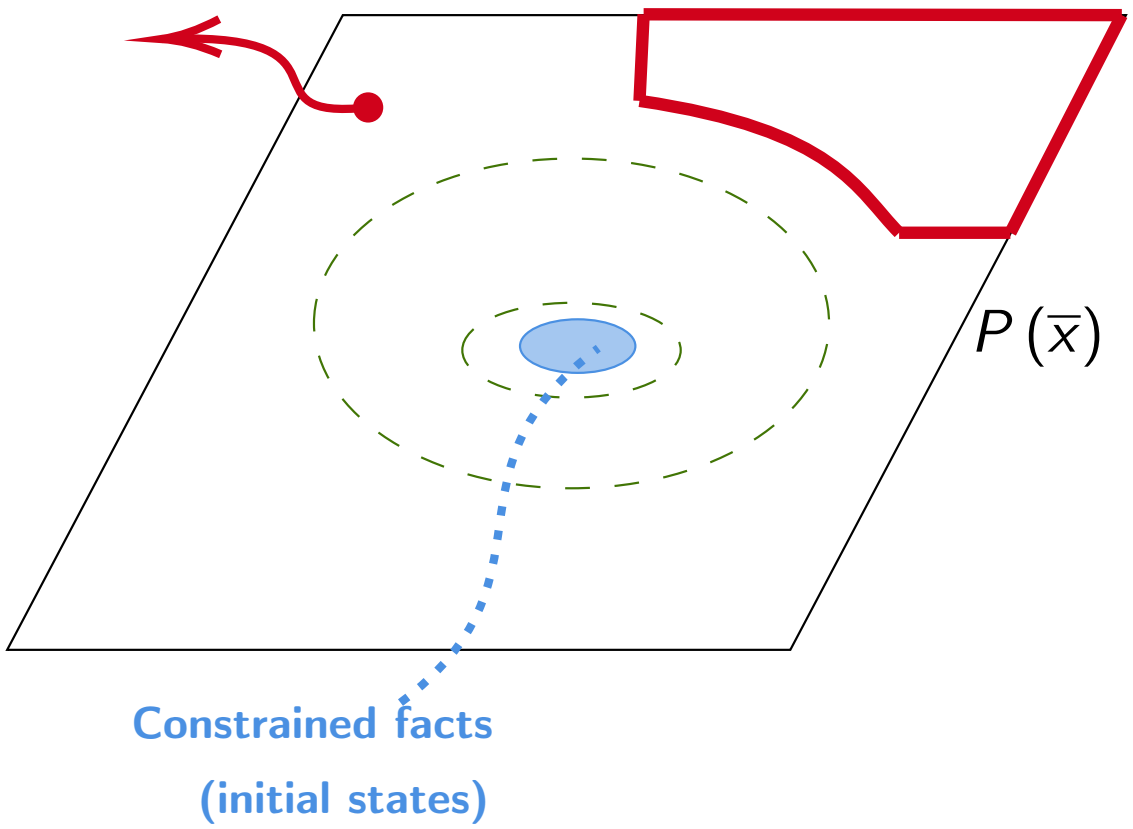
Property Directed Reachability

"SAT-based model checking without unrolling" (Bradley, 2011)
"Efficient Implementation of Property Directed Reachability" (Eén et al., 2011)



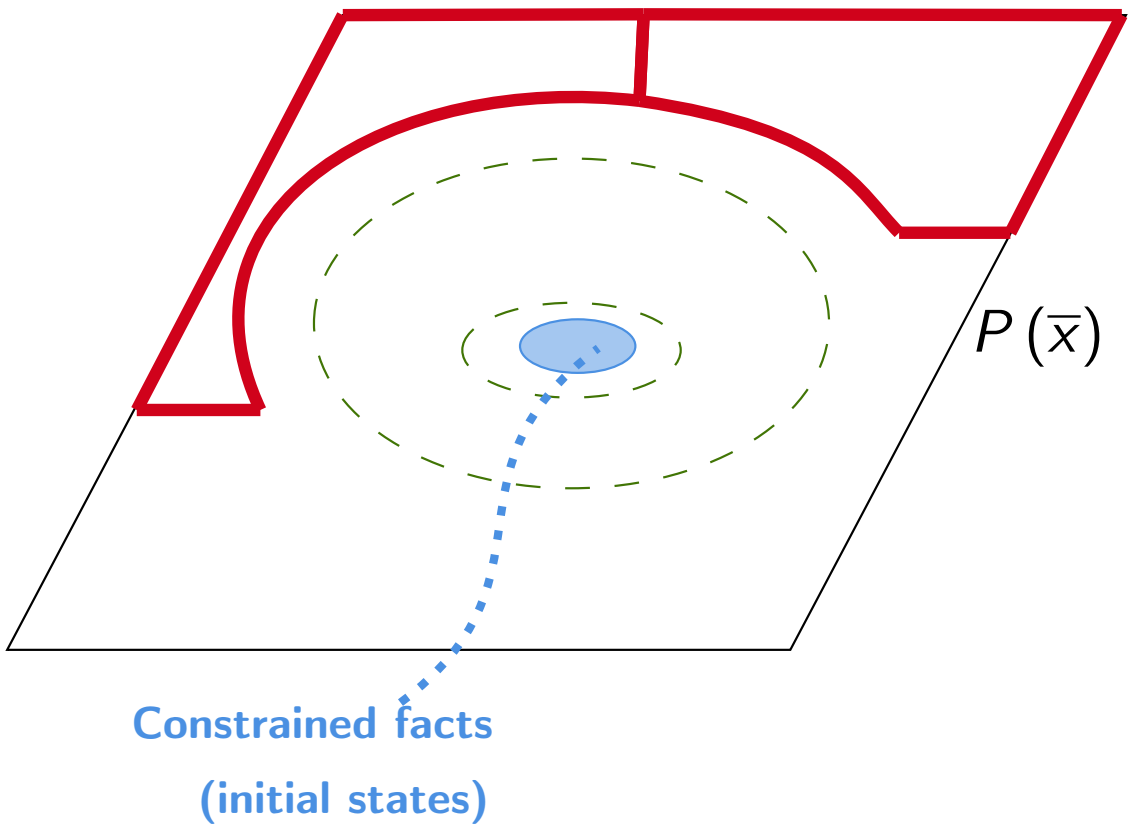
Property Directed Reachability

"SAT-based model checking without unrolling" (Bradley, 2011)
"Efficient Implementation of Property Directed Reachability" (Eén et al., 2011)



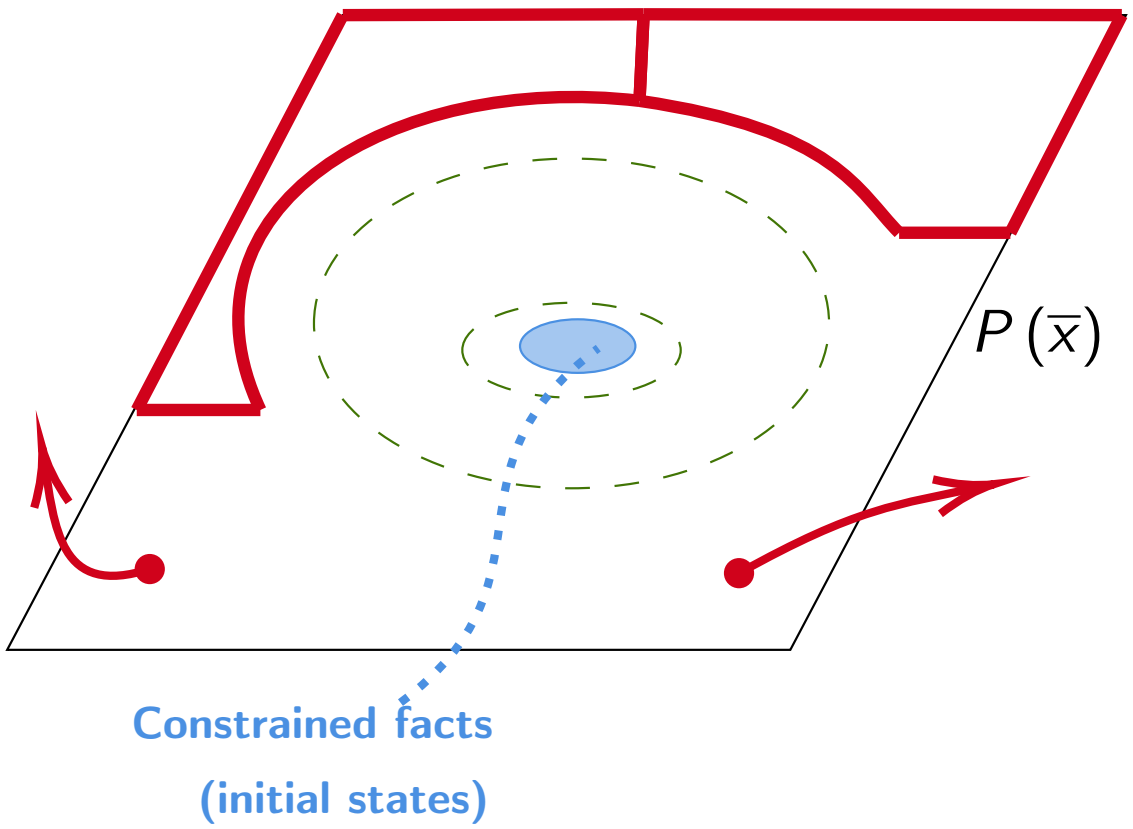
Property Directed Reachability

"SAT-based model checking without unrolling" (Bradley, 2011)
"Efficient Implementation of Property Directed Reachability" (Eén et al., 2011)



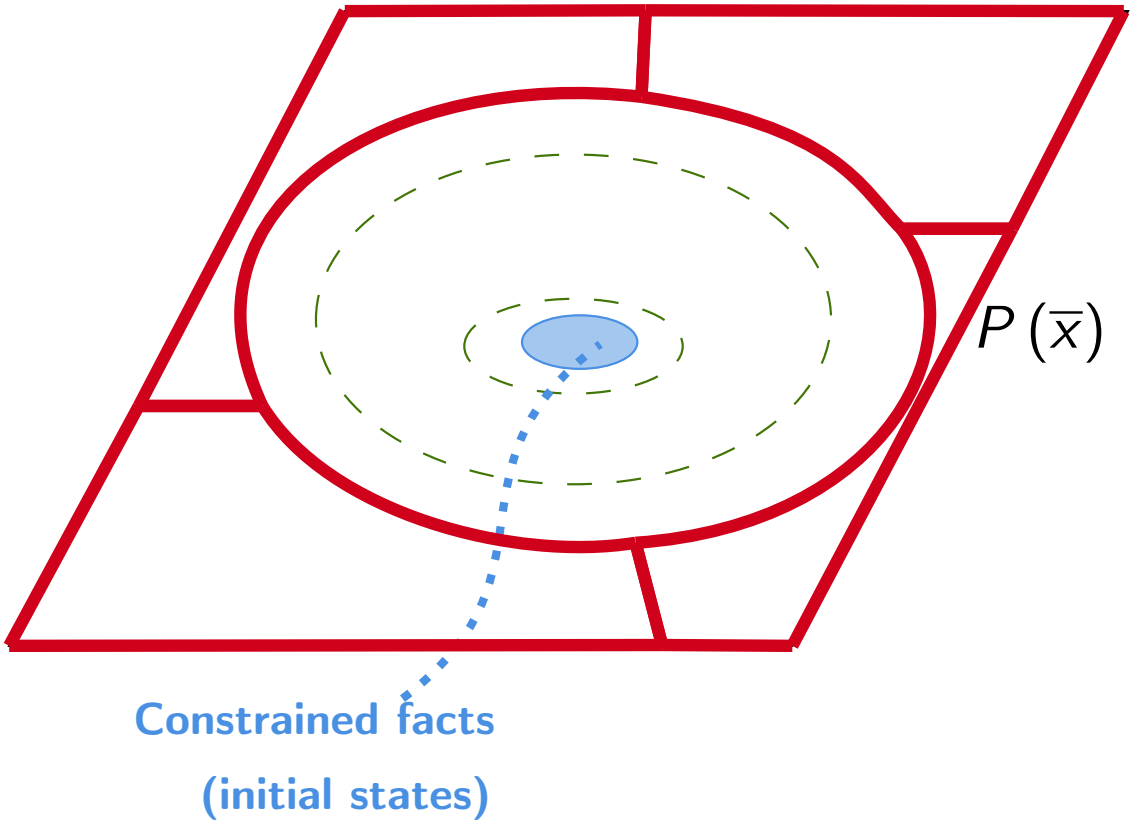
Property Directed Reachability

"SAT-based model checking without unrolling" (Bradley, 2011)
"Efficient Implementation of Property Directed Reachability" (Eén et al., 2011)



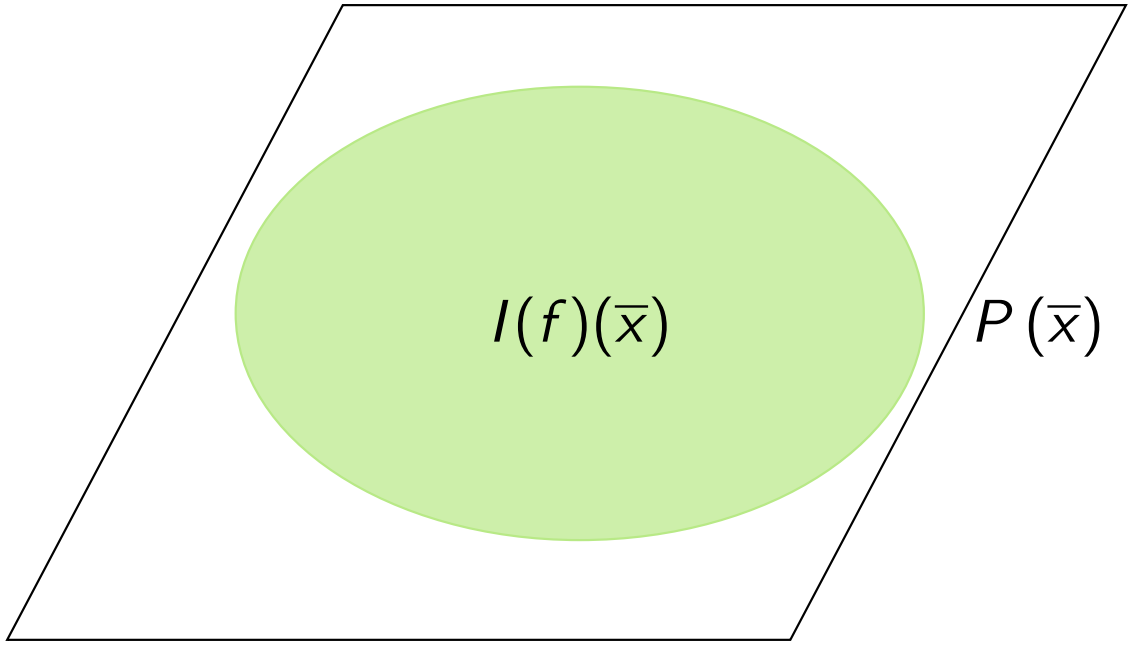
Property Directed Reachability

"SAT-based model checking without unrolling" (Bradley, 2011)
"Efficient Implementation of Property Directed Reachability" (Eén et al., 2011)



Property Directed Reachability

"SAT-based model checking without unrolling" (Bradley, 2011)
"Efficient Implementation of Property Directed Reachability" (Eén et al., 2011)



PDR for Linear CHC Systems

- Start with root query $f(\overline{x}) \rightarrow P$
- Get counterexample to inductiveness (CTI) s
- Detect a prime implicant of $body(f)$ satisfied by s
- As the system is linear, the prime implicant is linear as well!
  - Let it be $\psi \wedge g(\overline{y})$
- Use s form a new query P'
  - Project away all variables except $\overline{y}$ from $\psi \rightarrow P$

PDR for Linear CHC Systems

- Start with root query $f(\overline{x}) \to P$
- Get counterexample to inductiveness (CTI) s
- Detect a prime implicant of $body(f)$ satisfied by s
- As the system is linear, the prime implicant is linear as well!
  - Let it be $\psi \wedge g(\overline{y})$
- Use s form a new query P'
  - Project away all variables except $\overline{y}$ from $\psi \to P$

Thus we (recursively) turn the linear safety problem

$$f(\overline{x}) \to P$$

to a (simpler) linear subproblem

$$g(\overline{y}) \to P'$$

At the previous level

PDR for Linear CHC Systems

Given a CHC system, PDR terminates, if

- CTI satisfies some constrained fact
  - then the system is unsafe
- Summary facts at two subsequent levels are identical
  - then the system is safe
  - summary facts form a safety proof $\mathcal{I}$
  - $\mathcal{I}$ maps each **uninterpreted symbol** f to a first-order formula

PDR for Linear CHC Systems

Given a CHC system, PDR terminates, if

- CTI satisfies some constrained fact
  - then the system is unsafe
- Summary facts at two subsequent levels are identical
  - then the system is safe
  - summary facts form a safety proof $\mathcal{I}$
  - $\mathcal{I}$ maps each **uninterpreted symbol** f to a first-order formula

Linear problems are **easier** for modern PDR implementations (and other CHC solvers) than nonlinear

Results: **LIA-Lin**

Solver	Score	#SAT	#UNSAT	Avg time
Spacer	**279**	194	85	28.90
Rebus	**267**	188	79	41.85
Eldarica	**209**	129	80	24.55
Ultimate Unihorn Automizer	**133**	63	70	23.05
Hoice	**129**	65	64	7.09
Ultimate Tree Automizer	**107**	42	65	29.15
PCSat	**45**	33	12	23.74

* **325** instances total

Nonlinear Problems

Nonlinear CHCs naturally appear in various verification problems

- Verification of programs with multiple procedures

Program:

```
int x = f(n);
x = g(x);
assert(P(n,x));
```

Verification conditions:

$$\ldots \rightarrow f(n,x)$$

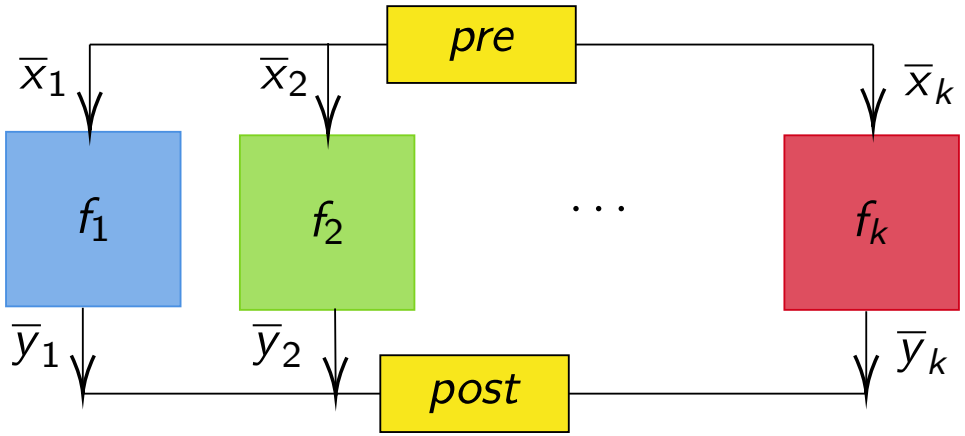
$$\ldots \rightarrow g(x,x')$$

$$f(n,x) \wedge g(x,x') \rightarrow P(n,x')$$

Nonlinear Problems

Nonlinear CHCs naturally appear in various verification problems

- Relational verification
  "Relational Verification Using Product Programs" (Barthe et al., 2011)



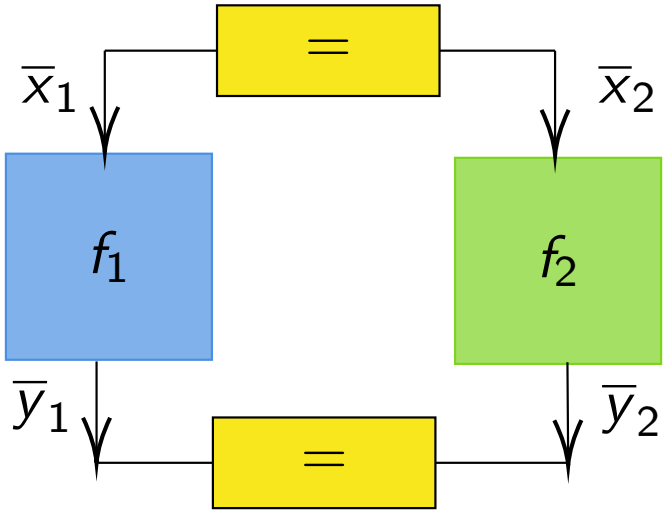
Goal: prove that a relational specification $(pre, post)$ holds for k programs

Nonlinear Problems

Nonlinear CHCs naturally appear in various verification problems

- Relational verification
  "Relational Verification Using Product Programs" (Barthe et al., 2011)

Given programs f_1 and f_2, that respectively have inputs $\overline{x}_1, \overline{x}_2$ and outputs $\overline{y}_1, \overline{y}_2$, prove $\overline{x}_1 = \overline{x}_2 \Rightarrow \overline{y}_1 = \overline{y}_2$



Example: program equivalence

Nonlinear Problems: Undefinability

Consider the following nonlinear CHC system in Linear Integer Arithmetic (LIA) ($\mathbb{Z} \cup \{\leq, +, -\}$):

$$mul(x, y, z) \leftarrow x = 0 \wedge z = 0$$

$$mul(x, y, z) \leftarrow x > 0 \wedge x' = x - 1 \wedge z = z' + y \wedge mul(x', y, z')$$

$$\bot \leftarrow x = x' \wedge y = y' \wedge mul(x, y, z) \wedge mul(x', y', z') \wedge z \neq z'$$

Nonlinear Problems: Undefinability

Consider the following nonlinear CHC system in Linear Integer Arithmetic (LIA) ($\mathbb{Z} \cup \{\leq, +, -\}$):

$$mul(x, y, z) \leftarrow x = 0 \wedge z = 0$$
$$mul(x, y, z) \leftarrow x > 0 \wedge x' = x - 1 \wedge z = z' + y \wedge mul(x', y, z')$$
$$\bot \leftarrow x = x' \wedge y = y' \wedge mul(x, y, z) \wedge mul(x', y', z') \wedge z \neq z'$$

Although the system is safe, no safety proof is definable in linear integer arithmetic!

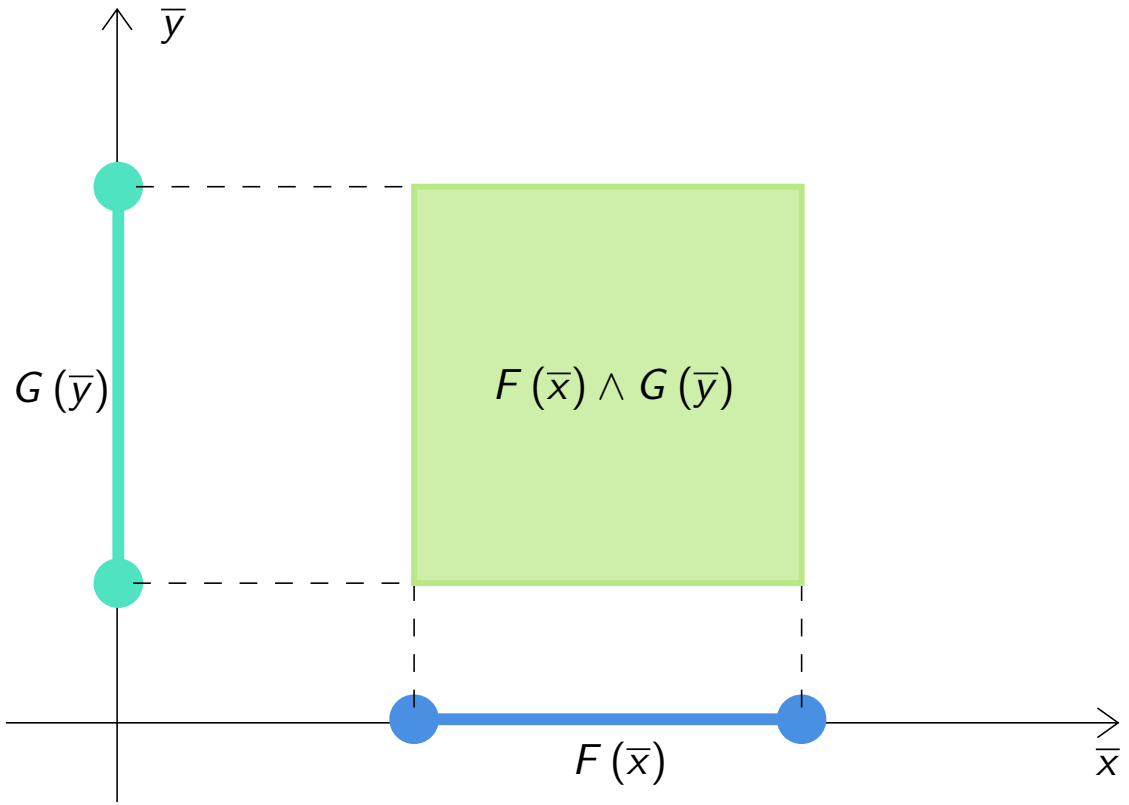
- The obvious invariant

$$\mathcal{I} = \{mul \mapsto (x \geq 0 \wedge y \geq 0 \rightarrow z = x \cdot y)\}$$

  is undefinable in LIA!

Traditional Safety Proofs

The problem is that "classical" safety proofs are not expressive enough for nonlinear systems:



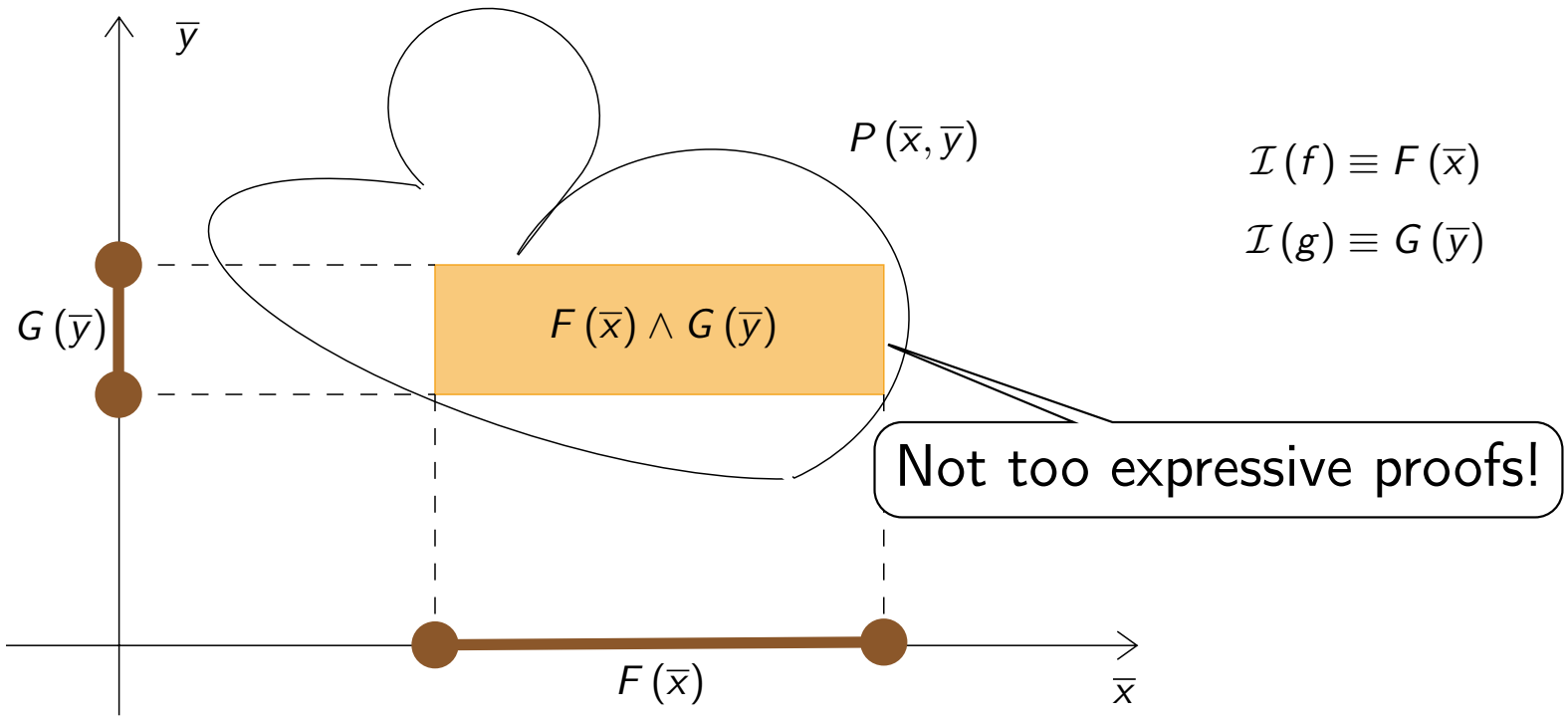
$\mathcal{I}(f) \equiv F(\overline{x})$

$\mathcal{I}(g) \equiv G(\overline{y})$

For nonlinear bodies like $f(\overline{x}) \wedge g(\overline{y}) \rightarrow H$, traditional interpretations define only Cartesian relations

Traditional vs Relational Proofs

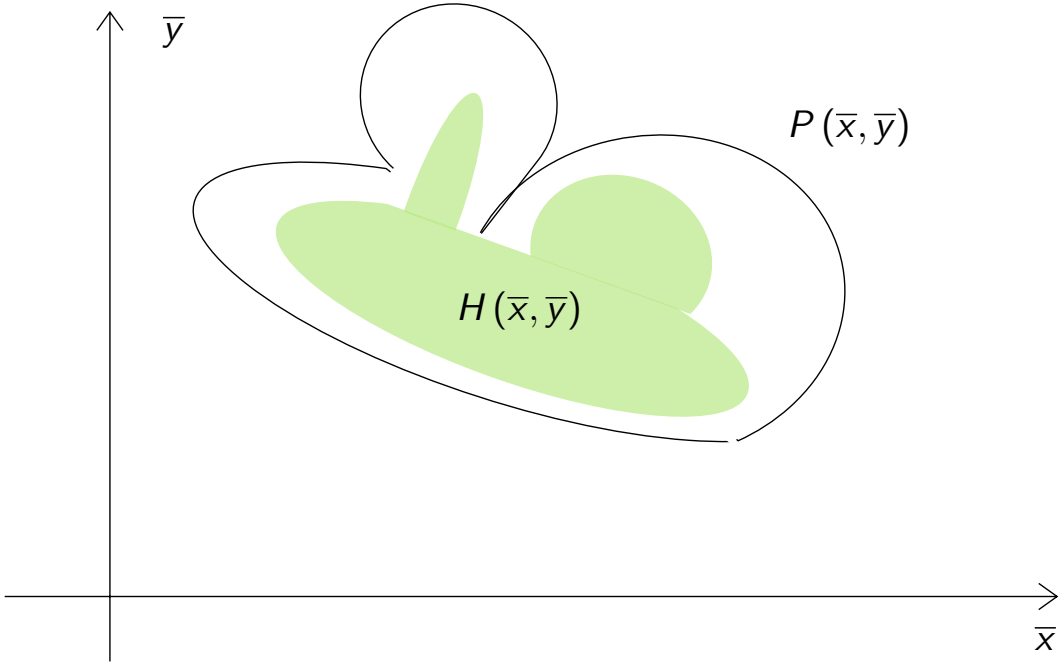
Problem: $f(\overline{x}) \wedge g(\overline{y}) \rightarrow P(\overline{x}, \overline{y})$



- $P(x, y)$ — safety property
- $F(\overline{x})$ is the invariant for f
- $G(\overline{y})$ is the invariant for g

Traditional vs Relational Proofs

Problem: $f(\overline{x}) \wedge g(\overline{y}) \rightarrow P(\overline{x}, \overline{y})$



$R(\langle f, g \rangle) \equiv H(\overline{x}, \overline{y})$

- $P(x, y)$ — safety property
- $H(x, y)$ is the invariant for product program $f \times g$

Relational Proofs

- Symbolic relational proofs interpret multisets of uninterpreted symbols instead of singleton symbols
- But they should be an invariant for product programs!

Relational Proofs

- Symbolic relational proofs interpret multisets of uninterpreted symbols instead of singleton symbols
- But they should be an invariant for product programs!

$$mul(x, y, z) \leftarrow x = 0 \wedge z = 0$$
$$mul(x, y, z) \leftarrow x > 0 \wedge x' = x - 1 \wedge z = z' + y \wedge mul(x', y, z')$$
$$\bot \leftarrow x = x' \wedge y = y' \wedge mul(x, y, z) \wedge mul(x', y', z') \wedge z \neq z'$$

Relational Proofs

- Symbolic relational proofs interpret multisets of uninterpreted symbols instead of singleton symbols
- But they should be an invariant for product programs!

$$mul(x, y, z) \leftarrow x = 0 \wedge z = 0$$
$$mul(x, y, z) \leftarrow x > 0 \wedge x' = x - 1 \wedge z = z' + y \wedge mul(x', y, z')$$
$$\bot \leftarrow x = x' \wedge y = y' \wedge mul(x, y, z) \wedge mul(x', y', z') \wedge z \neq z'$$

This system does not have "traditional" definable in LIA, but has a relational one:

$$\mathcal{J} = \left\{ \begin{array}{ll} mul & \mapsto \top \\ \langle mul, mul \rangle & \mapsto (x_1 = x_2 \wedge y_1 = y_2 \rightarrow z_1 = z_2) \end{array} \right\}$$

Relational Proofs

Let S be a CHC system

- If S has a relational proof, then it is safe
- If S has a definable "traditional" safety proof, then it has a relational safety proof as well
- The converse is not true!
- Both forms of proofs evaluate linear systems identically
  - The relational invariants are useful only for nonlinear systems!

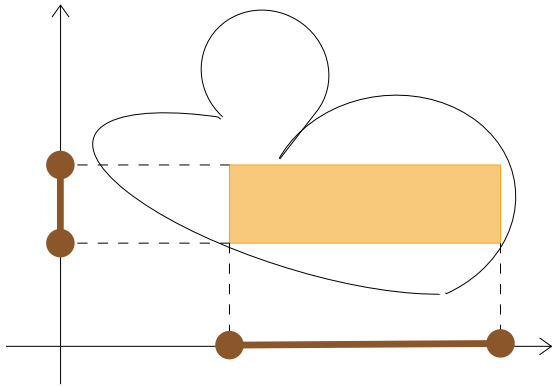
PDR vs Relational PDR

Problem : $f(\overline{x}) \wedge g(\overline{y}) \rightarrow P(\overline{x}, \overline{y})$

"SMT-based model checking for recursive programs" (Komuravelli et al., 2016)

use lemmas from previous levels to **split** the safety property

- Spawns **two** proof obligations (f, level, P_1) and (g, level, P_2)
  - $P_1(\overline{x}) \equiv \exists \overline{y}.\sigma(g, \mathit{level} - 1)(\overline{y}) \rightarrow P(\overline{x}, \overline{y})$
  - $P_2(\overline{y}) \equiv \exists \overline{x}.\sigma(f, \mathit{level} - 1)(\overline{x}) \rightarrow P(\overline{x}, \overline{y})$
- If no model of a safe system is definable, never terminates



PDR vs Relational PDR

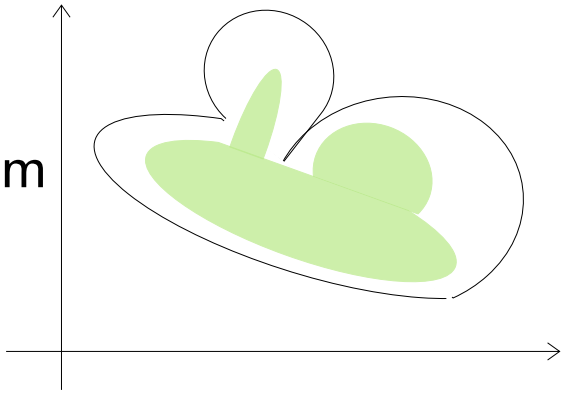
$$\textbf{Problem} : f(\overline{x}) \wedge g(\overline{y}) \rightarrow P(\overline{x}, \overline{y})$$

Our algorithm partitions the predicates into the groups, tending to obtain the linear problems, but for the relational specifications

- Tries to answer the relational proof obligation

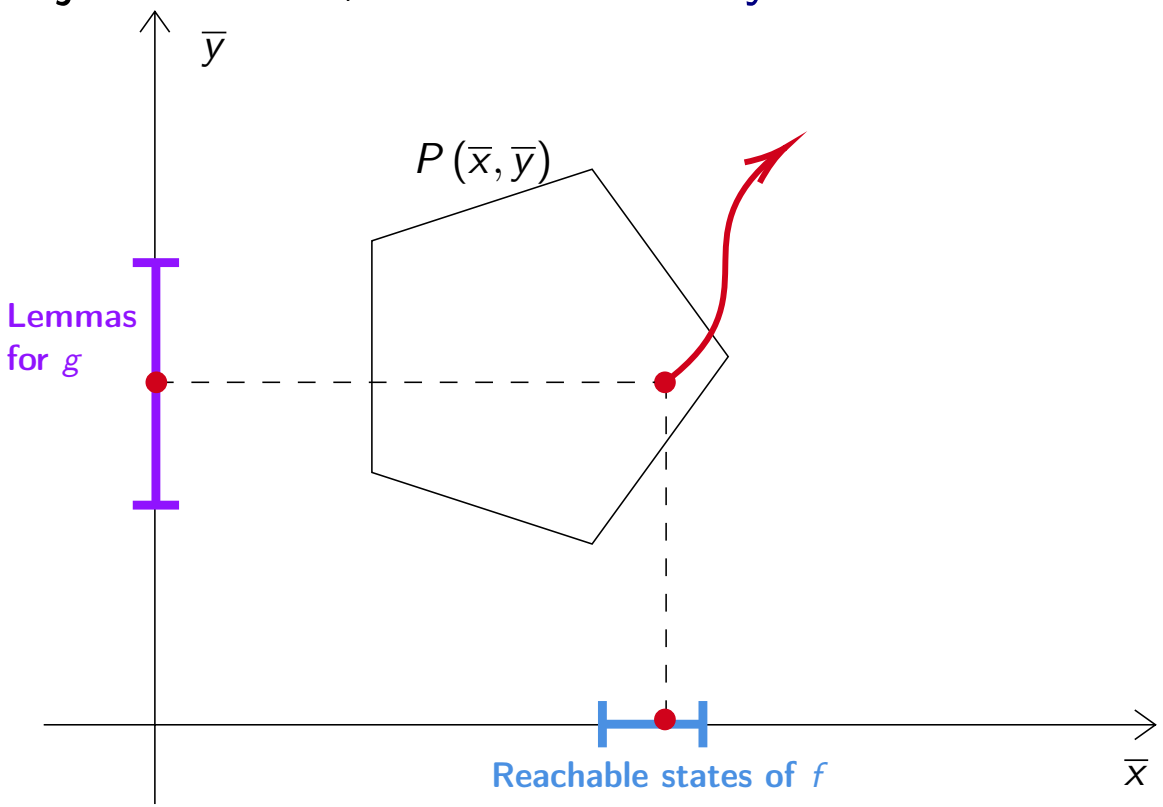
$$\left(\langle f, g \rangle, \mathit{level}, P(\overline{x}, \overline{y}) \right)$$

- Detects and blocks CTIs in product program $f \times g$
- Doesn't explicitly construct a product program
- Obtains relational lemmas by interpolation



To Merge or Not to Merge?

- Let s be a CTI at some level for a nonlinear problem
- Unlike the linear case, some predicates may reach the projection of s, while some may not:



To Merge or Not to Merge?

- If some predicate reaches the projection of CTI s, then there is no need to strengthen its lemmas
  - We can't exclude the point anyway
- Thus all predicates witnessing s are not "merged"
  - Instead, we use the reachability information to weaken the safety property
- Different CTI may result in different groups of predicates
  - But larger groups "subsume" lemmas of smaller groups

Implementation

- The procedure was implemented in SPACER kernel
  - The relational generalization of PDR
  - Given constraints satisfiability oracle, the algorithm is a co-decision procedure for safety and complete for the systems with finite state space

Experiments: HoIce benchmarks

"HoIce: An ICE-Based Non-linear Horn Clause Solver" (Champion et al., 2018)

840 tests, different sources (mostly generated by MoCHi)

The amount of tests	HoIce	Spacer	RelSpacer
840	808	788	807

- Small timeout (30 sec.)
- Spacer has more timeouts than HoIce on nonlinear benchmarks
- Our implementation is more competitive, outperforming (timewise) HoIce on solved benchmarks

Experiments: Relational Benchmarks

"Synchronizing Constrained Horn Clauses" (Mordvinov et al., 2017)

- 37 relational problems
- A lot of problems with undefinable «classical» invariants
- New competitor: CHCProduct implemented as a syntactical preprocessing of clauses in Z3, then solving with Spacer
- Timeout: 5 min.

Total	HoIce	Spacer	CHCProduct	RelSpacer
37	11	11	24	32

Experiments: Relational Benchmarks

"Synchronizing Constrained Horn Clauses" (Mordvinov et al., 2017)

- 37 relational problems
- A lot of problems with undefinable «classical» invariants
- New competitor: CHCProduct implemented as a syntactical preprocessing of clauses in Z3, then solving with Spacer
- Timeout: 5 min.

Total	HoIce	Spacer	CHCProduct	RelSpacer
37	11	11	24	32

For some large unsafe problems syntactical transformation fails to deliver a counterexample within the timeout!

- Generates exponential amount of rules
- RelSpacer does the job in seconds

Conclusion

- Traditional safety proofs are not expressive enough for nonlinear CHC systems
- Relational safety proofs is a better form of their solutions
- Traditional PDR works well for linear systems
- It can be easily generalized to infer relational proofs!
  - Generate relational proof obligations instead of splitting safety property
  - Block CTI for product programs
- Relational PDR is a better choice than syntactical transformations!