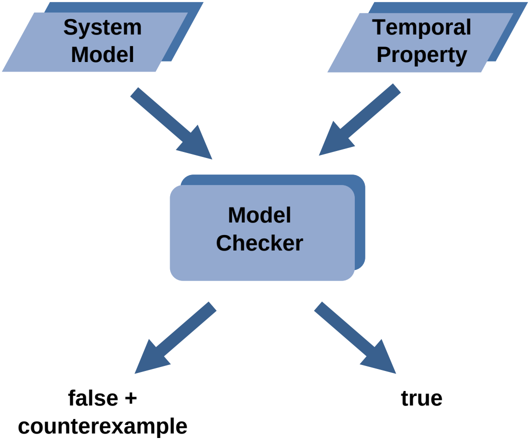
Exploiting Resolution Proofs to Speed Up LTL Vacuity Detection for BMC

Jocelyn Simmonds   Jessica Davies   Marsha Chechik
Department of Computer Science, University of Toronto

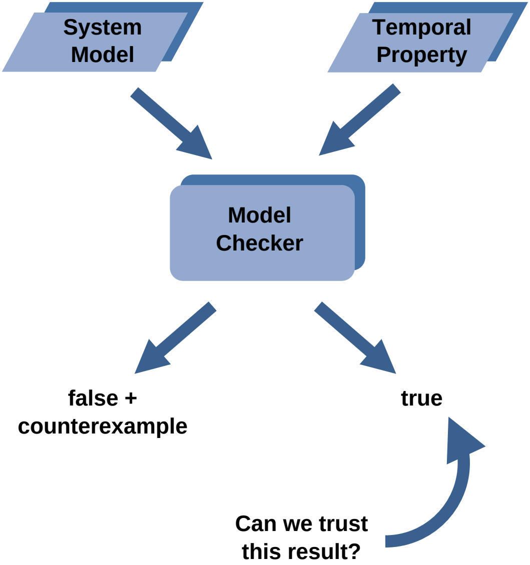
Arie Gurfinkel
Software Engineering Institute at Carnegie Mellon University

Nov 12, 2007 - FMCAD '07

Model Checking



Model Checking



Sanity Checks

Errors in Model	**Errors in Environment**	**Errors in Property**
Debugging Overconstrained Declarative Models	Finding Environmental Guarantees	Vacuity Detection
[Shlyakhter et al. ’03]	[Chechik et al. ’07]	[Beer et al. ’99]
		[Kupferman, Vardi ’99] . . .

Sanity Checks

Errors in Model	**Errors in Environment**	**Errors in Property**
Debugging Overconstrained Declarative Models	Finding Environmental Guarantees	Vacuity Detection
[Shlyakhter et al. '03]	[Chechik et al. '07]	[Beer et al. '99]
		[Kupferman, Vardi '99] . . .

Vacuity Dectection

Sanity Checks

Errors in Model	**Errors in Environment**	**Errors in Property**
Debugging Overconstrained Declarative Models	Finding Environmental Guarantees	Vacuity Detection
[Shlyakhter et al. '03]	[Chechik et al. '07]	[Beer et al. '99]
		[Kupferman, Vardi '99] . . .

Vacuity Dectection

GOAL: determine what parts of a property are not relevant

Sanity Checks

Errors in Model	**Errors in Environment**	**Errors in Property**
Debugging Overconstrained Declarative Models	Finding Environmental Guarantees	Vacuity Detection
[Shlyakhter et al. ’03]	[Chechik et al. ’07]	[Beer et al. ’99]
		[Kupferman, Vardi ’99] . . .

Vacuity Dectection

GOAL: determine what parts of a property are not relevant

- . . . anything that can be substituted without changing the value of the property

Sanity Checks

Errors in Model	Errors in Environment	Errors in Property
Debugging Overconstrained Declarative Models	Finding Environmental Guarantees	Vacuity Detection
[Shlyakhter et al. '03]	[Chechik et al. '07]	[Beer et al. '99]
		[Kupferman, Vardi '99] . . .

Vacuity Dectection

GOAL: determine what parts of a property are not relevant

- . . . anything that can be substituted without changing the value of the property
- Example: "all requests are eventually serviced"

Sanity Checks

Errors in Model	Errors in Environment	Errors in Property
Debugging Overconstrained Declarative Models	Finding Environmental Guarantees	Vacuity Detection
[Shlyakhter et al. '03]	[Chechik et al. '07]	[Beer et al. '99]
		[Kupferman, Vardi '99] ...

Vacuity Dectection

GOAL: determine what parts of a property are not relevant

- ...anything that can be substituted without changing the value of the property
- Example: "all requests are eventually serviced"
  LTL: $p = G(\mathit{request} \Rightarrow F\ \mathit{serviced})$

Sanity Checks

Errors in Model	**Errors in Environment**	**Errors in Property**
Debugging Overconstrained Declarative Models	Finding Environmental Guarantees	Vacuity Detection
[Shlyakhter et al. '03]	[Chechik et al. '07]	[Beer et al. '99]
		[Kupferman, Vardi '99] ...

Vacuity Dectection

GOAL: determine what parts of a property are not relevant

- ... anything that can be substituted without changing the value of the property
- Example: "all requests are eventually serviced"
  LTL: $p = G(\mathit{request} \Rightarrow F\ \mathit{serviced})$
  holds in a model that does not produce any requests!

Naive Vacuity Detection [Beer et al. ’97, Kupferman and Vardi ’99]

Test by substituting each subformula to check which ones are vacuous

EXAMPLE: “all requests are eventually serviced”

formalized as $p = G\,(\text{ request } \Rightarrow F \text{ serviced })$

SOLUTION: four model checking runs

- $p_1 = G\,(\text{ true } \Rightarrow F \text{ serviced })$
- $p_2 = G\,(\text{ false } \Rightarrow F \text{ serviced })$

Naive Vacuity Detection [Beer et al. '97, Kupferman and Vardi '99]

Test by substituting each subformula to check which ones are vacuous

EXAMPLE: "all requests are eventually serviced"

formalized as $p = G$ (request $\Rightarrow$ F serviced)

SOLUTION: four model checking runs

- $p_1 = G$ (true $\Rightarrow$ F serviced)
- $p_2 = G$ (false $\Rightarrow$ F serviced)

p is vacuous w.r.t. "request" iff $M \models p_1 = M \models p_2$

Naive Vacuity Detection [Beer et al. '97, Kupferman and Vardi '99]

Test by substituting each subformula to check which ones are vacuous

EXAMPLE: "all requests are eventually serviced"

formalized as $p = G\,(\text{ request } \Rightarrow F \text{ serviced })$

SOLUTION: four model checking runs

- $p_1 = G\,(\text{ true } \Rightarrow F \text{ serviced })$
- $p_2 = G\,(\text{ false } \Rightarrow F \text{ serviced })$

  p is vacuous w.r.t. "request" iff $M \models p_1 = M \models p_2$
- $p_3 = G\,(\text{ request } \Rightarrow F \text{ true })$
- $p_4 = G\,(\text{ request } \Rightarrow F \text{ false })$

Naive Vacuity Detection [Beer et al. '97, Kupferman and Vardi '99]

Test by substituting each subformula to check which ones are vacuous

EXAMPLE: "all requests are eventually serviced"

formalized as $p = G\,(\text{request} \Rightarrow F\,\text{serviced})$

SOLUTION: four model checking runs

- $p_1 = G\,(\text{true} \Rightarrow F\,\text{serviced})$
- $p_2 = G\,(\text{false} \Rightarrow F\,\text{serviced})$

  p is vacuous w.r.t. "request" iff $M \models p_1 = M \models p_2$
- $p_3 = G\,(\text{request} \Rightarrow F\,\text{true})$
- $p_4 = G\,(\text{request} \Rightarrow F\,\text{false})$

  p is vacuous w.r.t. "serviced" iff $M \models p_3 = M \models p_4$

Naive Vacuity Detection [Beer et al. '97, Kupferman and Vardi '99]

Test by substituting each subformula to check which ones are vacuous

EXAMPLE: "all requests are eventually serviced"

formalized as $p = G\ (\ \text{request} \Rightarrow F\ \text{serviced}\ )$

SOLUTION: four model checking runs

- $p_1 = G\ (\ \text{true} \Rightarrow F\ \text{serviced}\ )$
- $p_2 = G\ (\ \text{false} \Rightarrow F\ \text{serviced}\ )$

  p is vacuous w.r.t. "request" iff $M \models p_1 = M \models p_2$
- $p_3 = G\ (\ \text{request} \Rightarrow F\ \text{true}\ )$
- $p_4 = G\ (\ \text{request} \Rightarrow F\ \text{false}\ )$

  p is vacuous w.r.t. "serviced" iff $M \models p_3 = M \models p_4$

Complete

Naive Vacuity Detection [Beer et al. '97, Kupferman and Vardi '99]

Test by substituting each subformula to check which ones are vacuous

EXAMPLE: "all requests are eventually serviced"

formalized as $p = G\ (\text{request} \Rightarrow F\ \text{serviced})$

SOLUTION: four model checking runs

- $p_1 = G\ (\text{true} \Rightarrow F\ \text{serviced})$
- $p_2 = G\ (\text{false} \Rightarrow F\ \text{serviced})$

  p is vacuous w.r.t. "request" iff $M \models p_1 \mathbf{=} M \models p_2$
- $p_3 = G\ (\text{request} \Rightarrow F\ \text{true})$
- $p_4 = G\ (\text{request} \Rightarrow F\ \text{false})$

  p is vacuous w.r.t. "serviced" iff $M \models p_3 \mathbf{=} M \models p_4$

Complete

Can be done without any special purpose tools

Naive Vacuity Detection [Beer et al. '97, Kupferman and Vardi '99]

Test by substituting each subformula to check which ones are vacuous

EXAMPLE: "all requests are eventually serviced"

formalized as $p = G \text{ (request } \Rightarrow F \text{ serviced)}$

SOLUTION: four model checking runs

- $p_1 = G \text{ (true } \Rightarrow F \text{ serviced)}$
- $p_2 = G \text{ (false } \Rightarrow F \text{ serviced)}$

  p is vacuous w.r.t. "request" iff $M \models p_1 = M \models p_2$
- $p_3 = G \text{ (request } \Rightarrow F \text{ true)}$
- $p_4 = G \text{ (request } \Rightarrow F \text{ false)}$

  p is vacuous w.r.t. "serviced" iff $M \models p_3 = M \models p_4$

Complete

Can be done without any special purpose tools

# of extra model checking runs grows with size of property

Brief Overview of Vacuity Detection

	Main Idea	Logic	Tool
[Beer et al. '97]	Replace single occurrence of a subformula with true, false	w-ACTL	RuleBase
[Kupferman and Vardi '99]	Generalized Beer's definition	CTL*	–
[Purandare and Somenzi '02]	Parse tree analysis to speed up vacuity detection	CTL	VIS
[Armoni et al. '03]	Introduced trace vacuity	LTL	Forecast Thunder
[Gurfinkel and Chechik '04]	Extended trace vacuity to CTL*	CTL*	Any model checker
[Gheorghiu and Gurfinkel '06]	Introduced concept of "vacuity" lattice	CTL	VaqUoT

Brief Overview of Vacuity Detection

	Main Idea	Logic	Tool
[Beer et al. '97]	Replace single occurrence of a subformula with true, false	w-ACTL	RuleBase
[Kupferman and Vardi '99]	Generalized Beer's definition	CTL*	–
[Purandare and Somenzi '02]	Parse tree analysis to speed up vacuity detection	CTL	VIS
[Armoni et al. '03]	Introduced trace vacuity	LTL	Forecast Thunder
[Gurfinkel and Chechik '04]	Extended trace vacuity to CTL*	CTL*	Any model checker
[Gheorghiu and Gurfinkel '06]	Introduced concept of "vacuity" lattice	CTL	VaqUoT

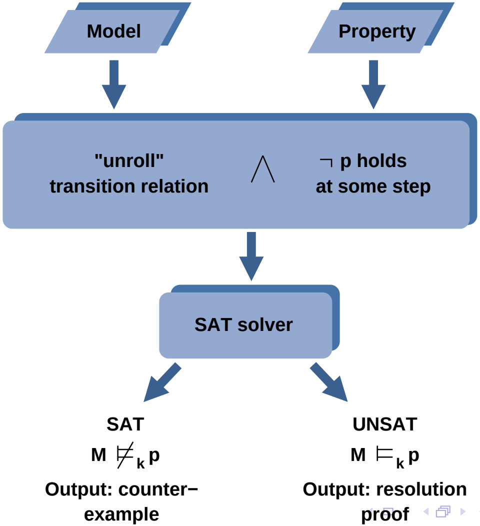
Definition of vacuity used in this work [Gurfinkel and Chechik '04]

Property p is vacuous w.r.t. variable v iff $M \models p[v \leftarrow x]$, where x is an unconstrained model variable

Bounded Model Checking (BMC)

Check if property p holds up to k steps on model M: $M \models_k p$

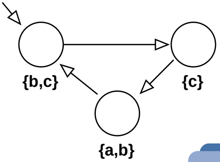
- i.e., can we reach a state in k steps that satisfies $\neg$ p?

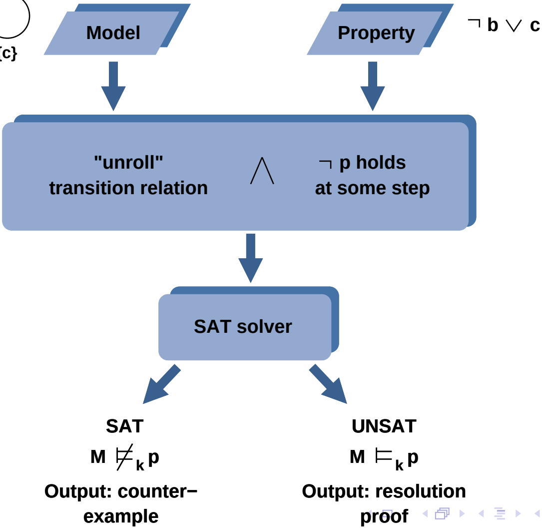


Bounded Model Checking (BMC)

Check if property p holds up to k steps on model M: $M \models_k p$

- i.e., can we reach a state in k steps that satisfies $\neg$ p?



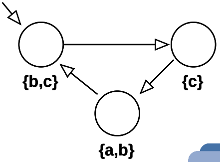


k = 0

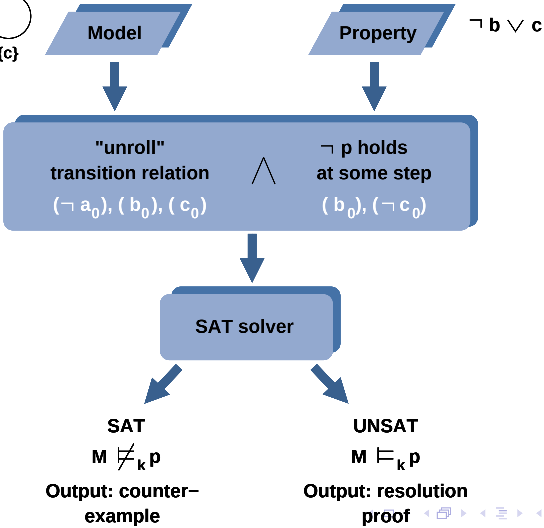
Bounded Model Checking (BMC)

Check if property p holds up to k steps on model M: $M \models_k p$

- i.e., can we reach a state in k steps that satisfies $\neg p$?



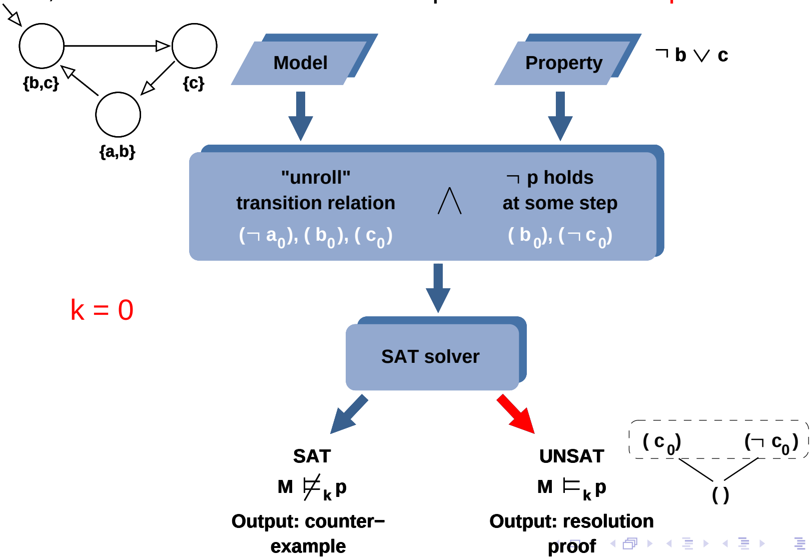
k = 0



Bounded Model Checking (BMC)

Check if property p holds up to k steps on model M: $M \models_k p$

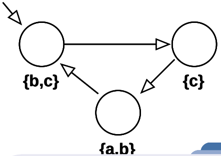
- i.e., can we reach a state in k steps that satisfies $\neg$ p?



Bounded Model Checking (BMC)

Check if property p holds up to k steps on model M: $M \models_k p$

- i.e., can we reach a state in k steps that satisfies $\neg$ p?



$\neg$ b $\vee$ c

GOAL: use resolution proof for vacuity detection

- focus on variable vacuity
- use naive detection as baseline for comparison

SAT

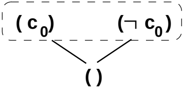
$M \not\models_k p$

Output: counter-example

UNSAT

$M \models_k p$

Output: resolution proof



Outline

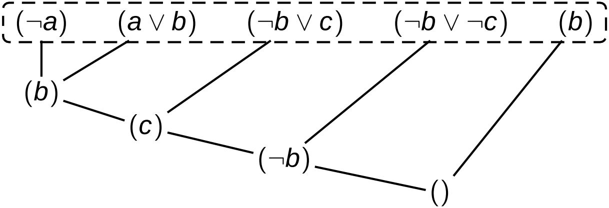
- Model Checking
- Sanity Checks
- Naive Vacuity Detection
- Brief Overview of Vacuity Detection
- Bounded Model Checking
- New methods:
  - Irrelevance
  - Local Irrelevance
  - Peripherality
- Implementation: VAQTREE
- Experiments
- Conclusions and Future Work

Algorithm 1 - Irrelevance

Model	$(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$
Property	$(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

Algorithm 1 - Irrelevance

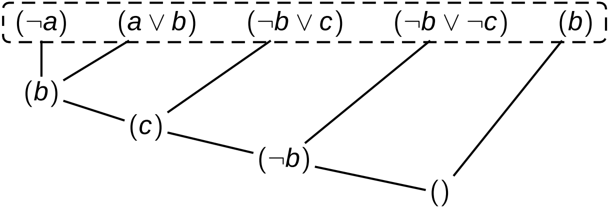
Model	$(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$
Property	$(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$



Algorithm 1 - Irrelevance

Model $(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$

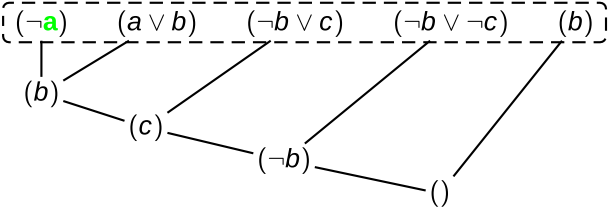
Property $(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$



Variables in the property but not in the UNSAT core are irrelevant

Algorithm 1 - Irrelevance

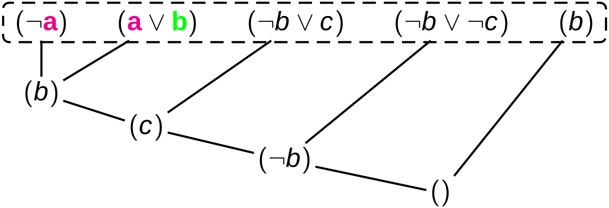
Model	$(\neg b \lor \neg c), (b), (\neg e), (d \lor f)$
Property	$(\neg \mathbf{a}), (\mathbf{a} \lor b), (\neg b \lor c), (d \lor e \lor f), (\mathbf{a} \lor \neg c \lor d)$



Variables in the property but not in the UNSAT core are irrelevant

Algorithm 1 - Irrelevance

Model	$(\neg \mathbf{b} \vee \neg c), (\mathbf{b}), (\neg e), (d \vee f)$
Property	$(\neg \mathbf{a}), (\mathbf{a} \vee \mathbf{b}), (\neg \mathbf{b} \vee c), (d \vee e \vee f), (\mathbf{a} \vee \neg c \vee d)$

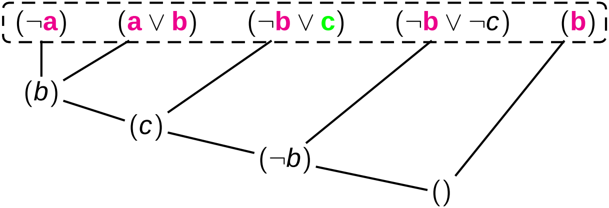


Variables in the property but not in the UNSAT core are irrelevant

Algorithm 1 - Irrelevance

Model $(\neg \mathbf{b} \lor \neg \mathbf{c}), (\mathbf{b}), (\neg e), (d \lor f)$

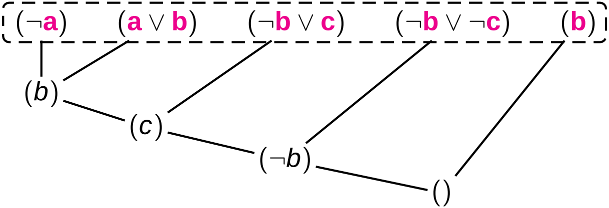
Property $(\neg \mathbf{a}), (\mathbf{a} \lor \mathbf{b}), (\neg \mathbf{b} \lor \mathbf{c}), (d \lor e \lor f), (\mathbf{a} \lor \neg \mathbf{c} \lor d)$



Variables in the property but not in the UNSAT core are irrelevant

Algorithm 1 - Irrelevance

Model	$(\neg\mathbf{b} \lor \neg\mathbf{c}), (\mathbf{b}), (\neg e), (d \lor f)$
Property	$(\neg\mathbf{a}), (\mathbf{a} \lor \mathbf{b}), (\neg\mathbf{b} \lor \mathbf{c}), (d \lor e \lor f), (\mathbf{a} \lor \neg\mathbf{c} \lor d)$

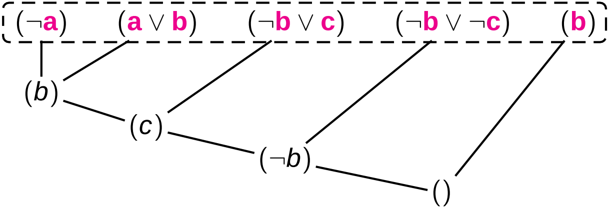


Variables in the property but not in the UNSAT core are irrelevant

Algorithm 1 - Irrelevance

Model $(\neg \mathbf{b} \vee \neg \mathbf{c}), (\mathbf{b}), (\neg e), (d \vee f)$

Property $(\neg \mathbf{a}), (\mathbf{a} \vee \mathbf{b}), (\neg \mathbf{b} \vee \mathbf{c}), (d \vee e \vee f), (\mathbf{a} \vee \neg \mathbf{c} \vee d)$



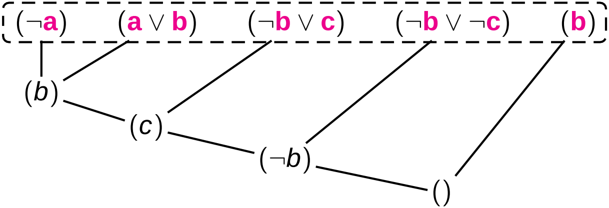
Variables in the property but not in the UNSAT core are irrelevant

VACUITY: d, e, f not in UNSAT core ⇒ irrelevant ⇒ vacuous

Algorithm 1 - Irrelevance

Model $(\neg \mathbf{b} \vee \neg \mathbf{c}), (\mathbf{b}), (\neg e), (d \vee f)$

Property $(\neg \mathbf{a}), (\mathbf{a} \vee \mathbf{b}), (\neg \mathbf{b} \vee \mathbf{c}), (d \vee e \vee f), (\mathbf{a} \vee \neg \mathbf{c} \vee d)$



Variables in the property but not in the UNSAT core are irrelevant

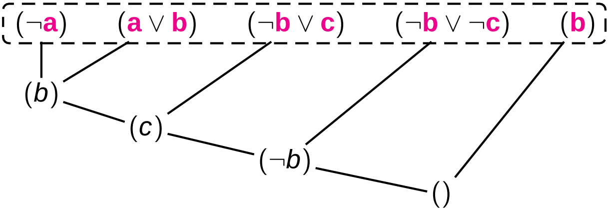
VACUITY: d, e, f not in UNSAT core $\Rightarrow$ irrelevant $\Rightarrow$ vacuous

Linear in size of UNSAT core

Algorithm 1 - Irrelevance

Model $(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$

Property $(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

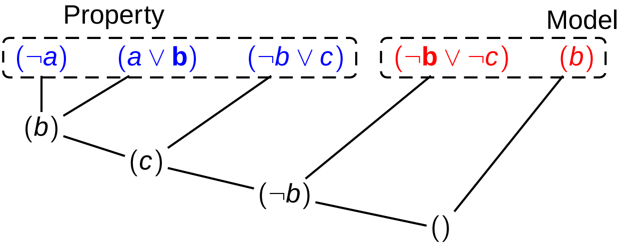


Variables in the property but not in the UNSAT core are irrelevant

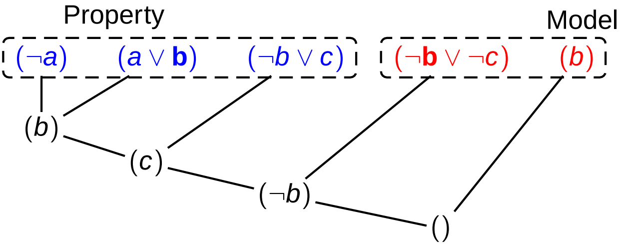
VACUITY: d, e, f not in UNSAT core ⇒ irrelevant ⇒ vacuous

- Linear in size of UNSAT core
- Very incomplete

Algorithm 2 - Local Irrelevance

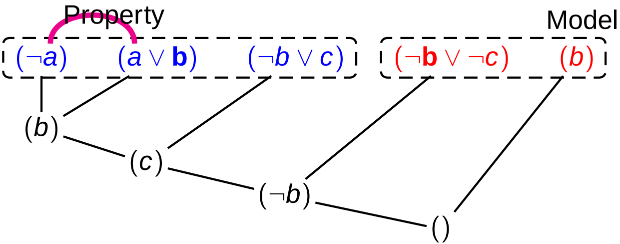


Algorithm 2 - Local Irrelevance



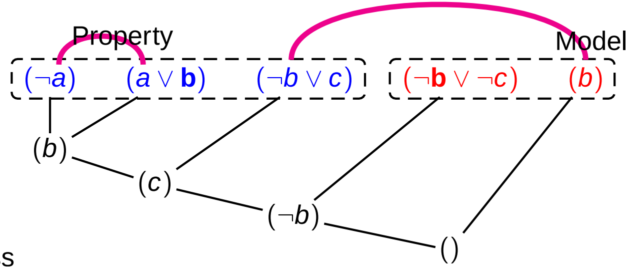
Variables that only appear in the property part of the UNSAT core are locally irrelevant

Algorithm 2 - Local Irrelevance



s

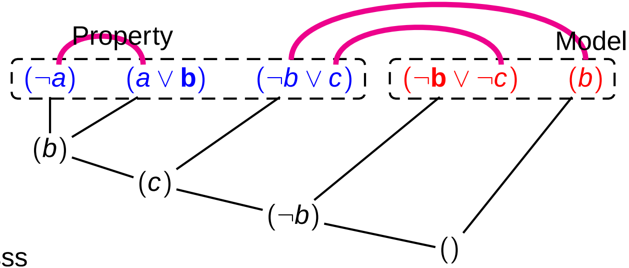
Variables that only appear in the property part of the UNSAT core are locally irrelevant



ss

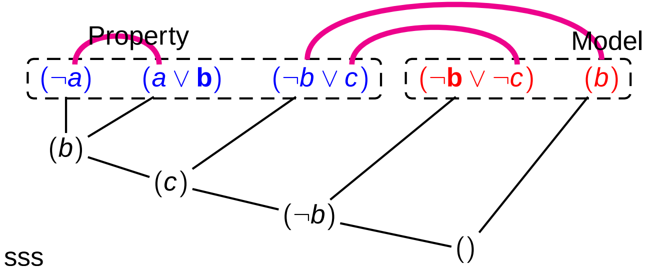
Variables that only appear in the property part of the UNSAT core are locally irrelevant

Algorithm 2 - Local Irrelevance



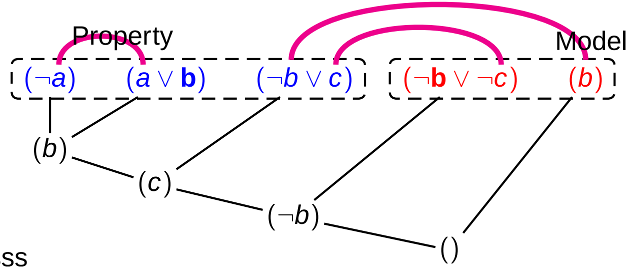
sss

Variables that only appear in the property part of the UNSAT core are locally irrelevant



Variables that only appear in the property part of the UNSAT core are locally irrelevant

VACUITY: a only in Property part of the UNSAT core
$\Rightarrow$ locally irrelevant $\Rightarrow$ vacuous



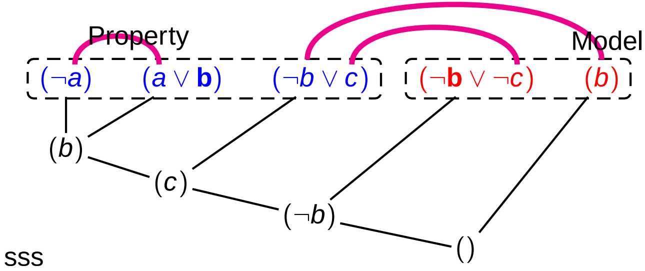
sss

Variables that only appear in the property part of the UNSAT core are locally irrelevant

VACUITY: a only in Property part of the UNSAT core

$\Rightarrow$ locally irrelevant $\Rightarrow$ vacuous

 Linear in size of UNSAT core



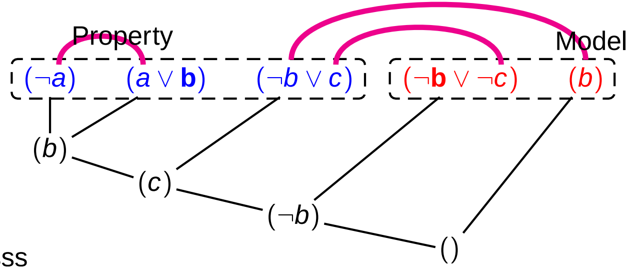
Variables that only appear in the property part of the UNSAT core are locally irrelevant

VACUITY: a only in Property part of the UNSAT core

$\Rightarrow$ locally irrelevant $\Rightarrow$ vacuous

- Linear in size of UNSAT core
- More precise than Irrelevance

Algorithm 2 - Local Irrelevance



sss

Variables that only appear in the property part of the UNSAT core are locally irrelevant

VACUITY: a only in Property part of the UNSAT core

$\Rightarrow$ locally irrelevant $\Rightarrow$ vacuous

- Linear in size of UNSAT core
- More precise than Irrelevance
- Still very incomplete

Algorithm 3 - Peripherality

Property

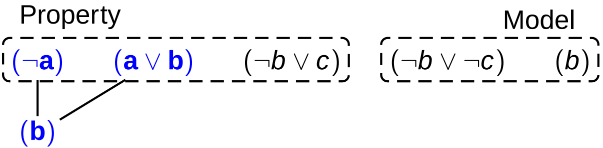
$(\neg \mathbf{a}) \quad (\mathbf{a} \lor \mathbf{b}) \quad (\neg b \lor c)$

Model

$(\neg b \lor \neg c) \quad (b)$

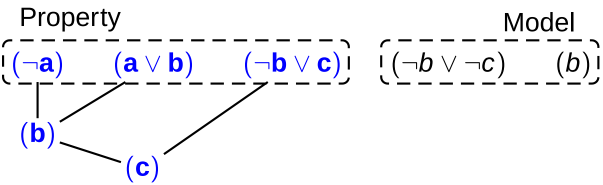
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality



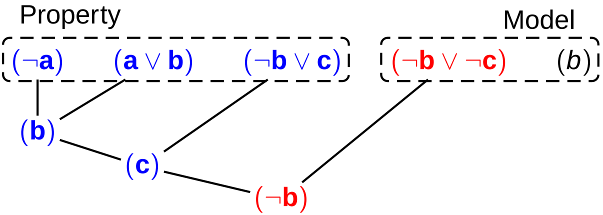
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality



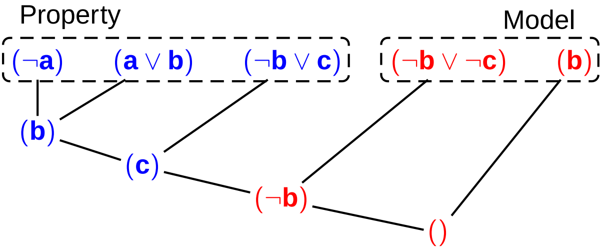
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality



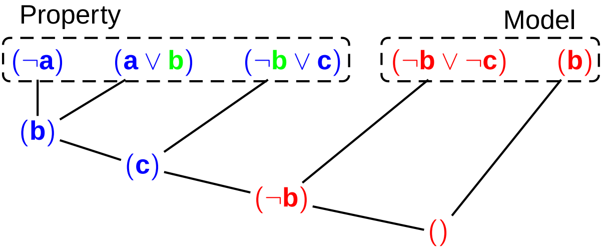
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality



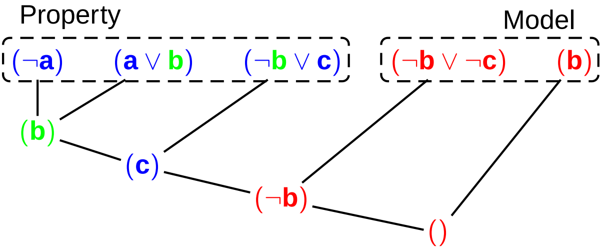
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality



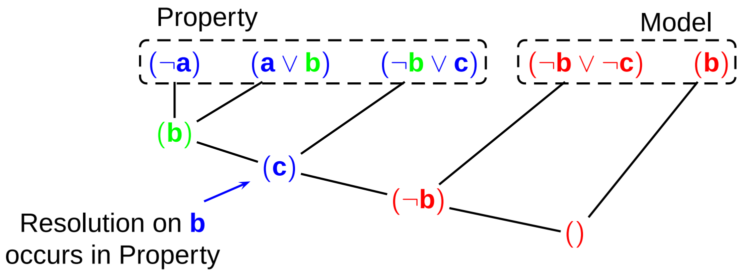
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality



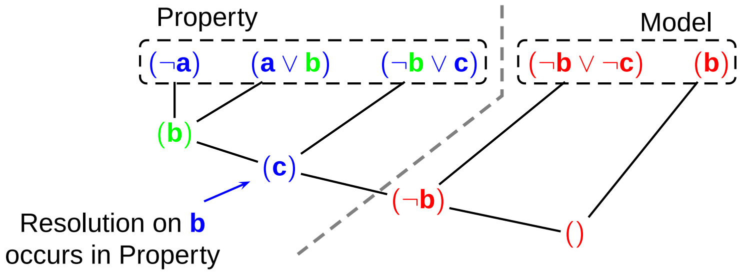
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality



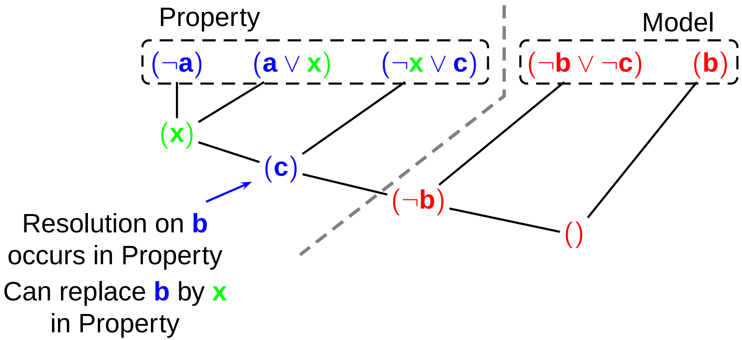
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality



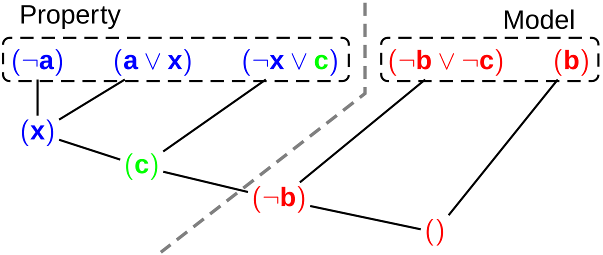
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality

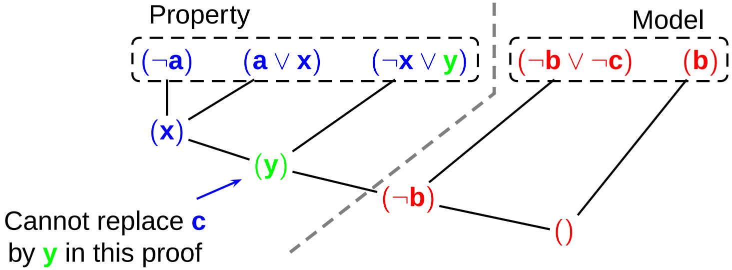


Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality

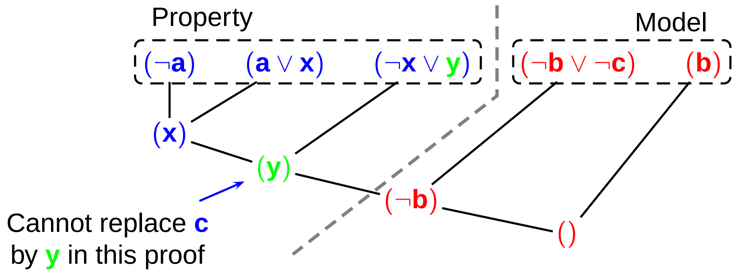


Variables that are not central to the proof are peripheral



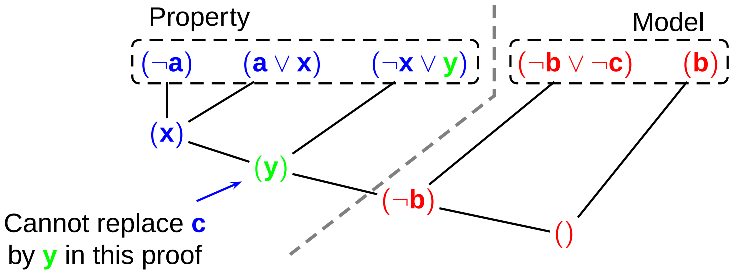
Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality



Variables that are not central to the proof are peripheral

Algorithm 3 - Peripherality

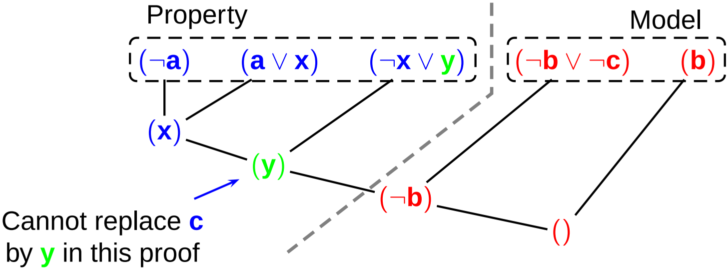


Variables that are not central to the proof are peripheral

VACUITY: replaced b by x in Property without changing proof

$\Rightarrow$ peripheral $\Rightarrow$ vacuous

Algorithm 3 - Peripherality



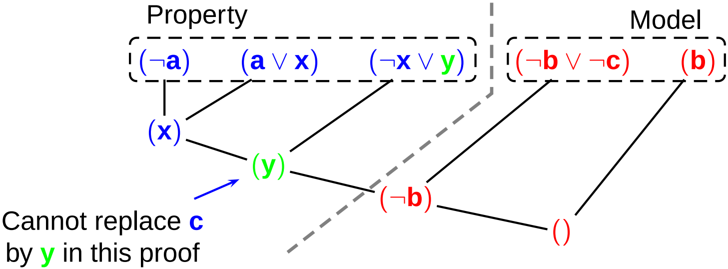
Variables that are not central to the proof are peripheral

VACUITY: replaced b by x in Property without changing proof

$\Rightarrow$ peripheral $\Rightarrow$ vacuous

Linear in size of resolution proof

Algorithm 3 - Peripherality



Variables that are not central to the proof are peripheral

VACUITY: replaced b by x in Property without changing proof

$\Rightarrow$ peripheral $\Rightarrow$ vacuous

- Linear in size of resolution proof
- If p is vacuous, there exists a resolution proof s.t. p is peripheral

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model	$(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$
Property	$(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model $(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$

Property $(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

IRRELEVANCE ALGORITHM

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model	$(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$
Property	$(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

IRRELEVANCE ALGORITHM

d,e,f are vacuous

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model	$(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$
Property	$(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

IRRELEVANCE ALGORITHM

d,e,f are vacuous

COMPLETING STEP

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model $(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$

Property $(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

IRRELEVANCE ALGORITHM

d,e,f are vacuous

COMPLETING STEP

6 extra model checking runs

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model $(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$

Property $(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

IRRELEVANCE ALGORITHM

d,e,f are vacuous

COMPLETING STEP

6 extra model checking runs

$M \models p[a \leftarrow \text{true}]?$

$M \models p[a \leftarrow \text{false}]?$

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model $(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$

Property $(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

IRRELEVANCE ALGORITHM

d,e,f are vacuous

COMPLETING STEP

6 extra model checking runs

$M \models p[a \leftarrow \text{true}]$?

$M \models p[a \leftarrow \text{false}]$?

p is vacuous w.r.t. a iff

$M \models p[a \leftarrow \text{true}] = M \models p[a \leftarrow \text{false}]$

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model $(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$

Property $(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

IRRELEVANCE ALGORITHM

d,e,f are vacuous

COMPLETING STEP

6 extra model checking runs

$M \models p[a \leftarrow \text{true}]$?

$M \models p[a \leftarrow \text{false}]$?

p is vacuous w.r.t. a iff

$M \models p[a \leftarrow \text{true}] = M \models p[a \leftarrow \text{false}]$

Similar for b,c

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model $(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$

Property $(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

IRRELEVANCE ALGORITHM

d,e,f are vacuous

COMPLETING STEP

6 extra model checking runs

$M \models p[a \leftarrow \text{true}]$?

$M \models p[a \leftarrow \text{false}]$?

p is vacuous w.r.t. a iff

$M \models p[a \leftarrow \text{true}] = M \models p[a \leftarrow \text{false}]$

Similar for b,c

IRRELEVANCE METHOD: Irrelevance algorithm + completing step

Complete Analysis

GOAL: complete analysis using Naive Detection for leftover variables

EXAMPLE:

Model $(\neg b \vee \neg c), (b), (\neg e), (d \vee f)$

Property $(\neg a), (a \vee b), (\neg b \vee c), (d \vee e \vee f), (a \vee \neg c \vee d)$

IRRELEVANCE ALGORITHM

d,e,f are vacuous

COMPLETING STEP

6 extra model checking runs

$M \models p[a \leftarrow \text{true}]$?

$M \models p[a \leftarrow \text{false}]$?

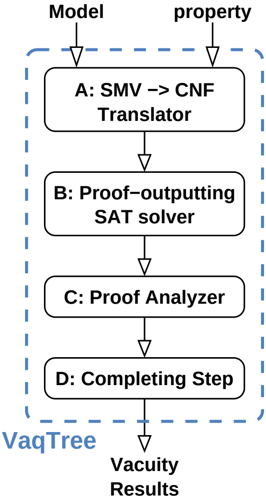
p is vacuous w.r.t. a iff $M \models p[a \leftarrow \text{true}] = M \models p[a \leftarrow \text{false}]$

Similar for b,c

IRRELEVANCE METHOD: Irrelevance algorithm + completing step

Local Irrelevance and Peripherality are also extended in this manner

VAQTREE: Vacuity Detection Framework



To our knowledge, VAQTREE is the first vacuity detection tool for BMC

[A] NUSMV v. 2.3.1, modified to identify model/property clauses

[B] MINISAT-p v. 1.14, modified to output XML proof

[C] New component (Java)

- proof analysis done in memory
- 700 MB of RAM $\approx$ 2.5 million resolutions

[D] New component (Perl)

Evaluation

GOALS:

- Compare effectiveness of the three algorithms
  - how many vacuous variables can each algorithm detect?
- Evaluate the performance of the three methods, using Naive Detection as a baseline
  - are any of our methods faster than Naive Detection?

BENCHMARKS:

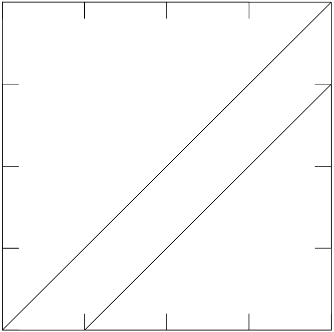
- Models and properties from the NuSMV distribution
- Models and properties from the IBM Formal Verification Benchmarks Library

Benchmark 1

SETUP

- Models and properties: NUSMV distribution
- 121 properties:
  - 99 present vacuity
  - 2 - 4 temporal operators per property, from {G, F, U, X}
  - 6 variables on average, 26 max., 1 min.
- Largest proof: 2.5 million resolutions

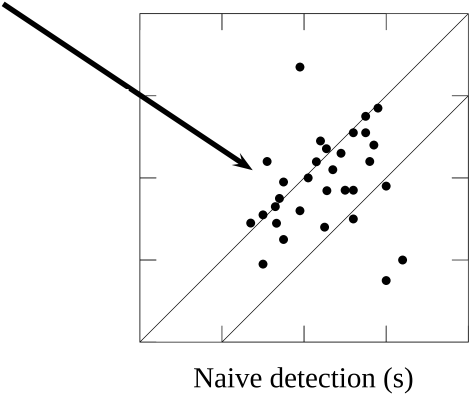
Interpreting Performance Graphs



Naive detection (s)

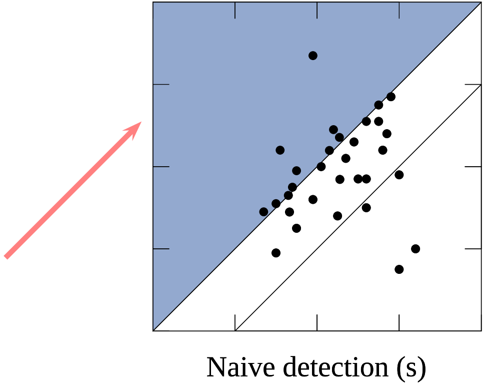
Interpreting Performance Graphs

Plotting execution times

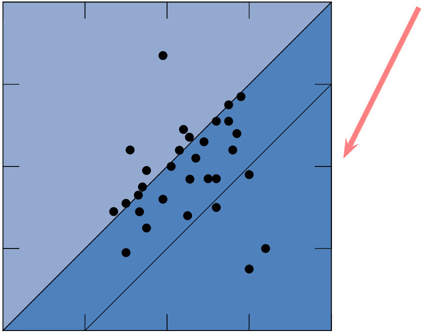


Interpreting Performance Graphs

Naive Detection is faster here



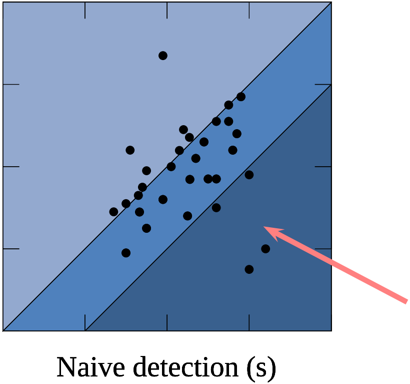
Interpreting Performance Graphs



Naive detection (s)

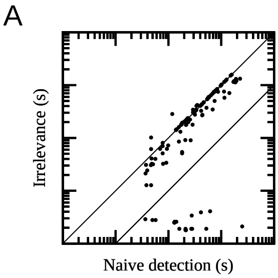
“Method” is faster here

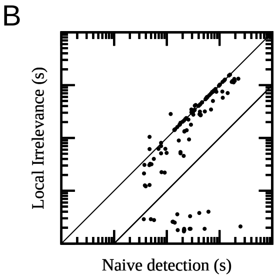
Interpreting Performance Graphs



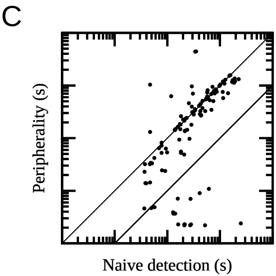
"Method" is faster by an order of magnitude here

Benchmark 1: Performance

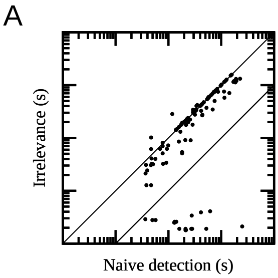


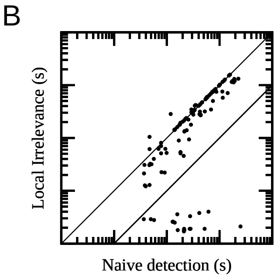


Execution times measured for complete methods



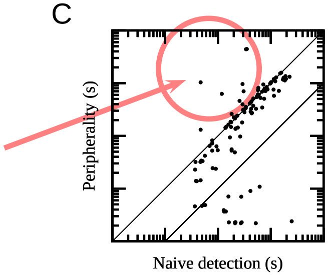
Benchmark 1: Performance





Execution times measured for complete methods

Peripherality is much slower in some cases



Why is Peripherality much slower in some cases?

Naive Detection

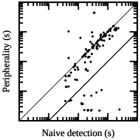
$$\Phi_1 = M \models p_1$$
$$\Phi_2 = M \models p_2$$
$$\vdots$$
$$\Phi_n = M \models p_n$$

Peripherality

$$\Phi = M \models p$$

Why is Peripherality much slower in some cases?

- Low clause/variable ratio
- No vacuous variables
- Large resolution proofs



Naive Detection

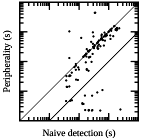
$$\Phi_1 = M \models p_1$$
$$\Phi_2 = M \models p_2$$
$$\vdots$$
$$\Phi_n = M \models p_n$$

Peripherality

$$\Phi = M \models p$$

Why is Peripherality much slower in some cases?

- Low clause/variable ratio
- No vacuous variables
- Large resolution proofs



Naive Detection

SAT →

$\Phi_1 = M \models p_1$

$\Phi_2 = M \models p_2$

$\vdots$

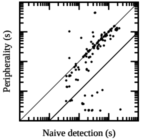
$\Phi_n = M \models p_n$

Peripherality

$\Phi = M \models p \longleftarrow$ UNSAT

Why is Peripherality much slower in some cases?

- Low clause/variable ratio
- No vacuous variables
- Large resolution proofs



Naive Detection

SAT → $\Phi_1 = M \models p_1$

SAT → $\Phi_2 = M \models p_2$

$\vdots$

SAT → $\Phi_n = M \models p_n$

time: $\tau_1, \tau_2, \ldots \tau_n$

to find sat. assignment

Peripherality

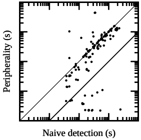
$\Phi = M \models p \longleftarrow$ UNSAT

T

periph. analysis

Why is Peripherality much slower in some cases?

- Low clause/variable ratio
- No vacuous variables
- Large resolution proofs



Naive Detection

SAT →

$$\Phi_1 = M \models p_1$$
$$\Phi_2 = M \models p_2$$
$$\vdots$$
$$\Phi_n = M \models p_n$$

Peripherality

$$\Phi = M \models p \longleftarrow \text{UNSAT}$$

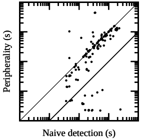
time: $\tau_1, \tau_2, \ldots \tau_n$ to find sat. assignment

T periph. analysis

$$\tau_i <<< T$$

Why is Peripherality much slower in some cases?

- Low clause/variable ratio
- No vacuous variables
- Large resolution proofs



Naive Detection

SAT →

$$\Phi_1 = M \models p_1$$
$$\Phi_2 = M \models p_2$$
$$\vdots$$
$$\Phi_n = M \models p_n$$

time: $\tau_1, \tau_2, \ldots \tau_n$ to find sat. assignment

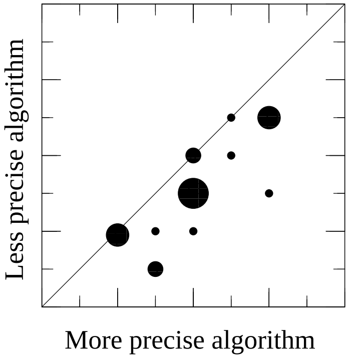
Peripherality

$$\Phi = M \models p \longleftarrow \text{UNSAT}$$

T periph. analysis

$$\tau_i <<< T$$
$$\sum \tau_i <<< T$$

Interpreting Effectiveness Graphs

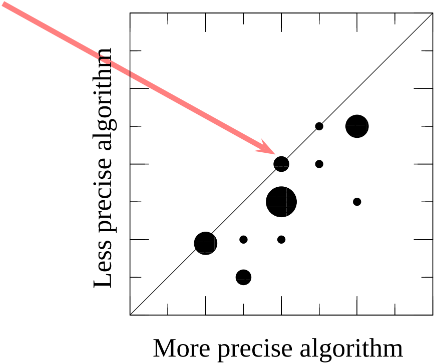


Interpreting Effectiveness Graphs

vacuous variables found: (x, y)

x = found by X-axis algorithm

y = found by Y-axis algorithm



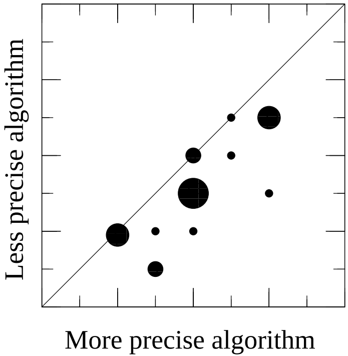
Interpreting Effectiveness Graphs

vacuous variables found: (x, y)

x = found by X-axis algorithm

y = found by Y-axis algorithm

X-axis algorithm is more precise, so $x \geq y$ always

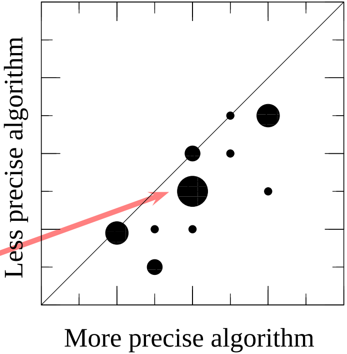


Interpreting Effectiveness Graphs

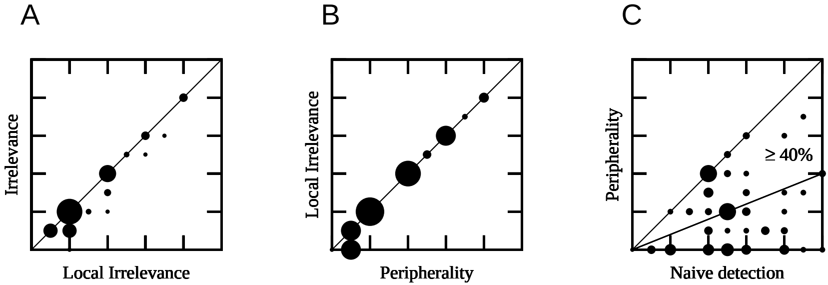
# vacuous variables found: (x, y)
x = found by X-axis algorithm
y = found by Y-axis algorithm

X-axis algorithm is more precise,
so $x \geq y$ always

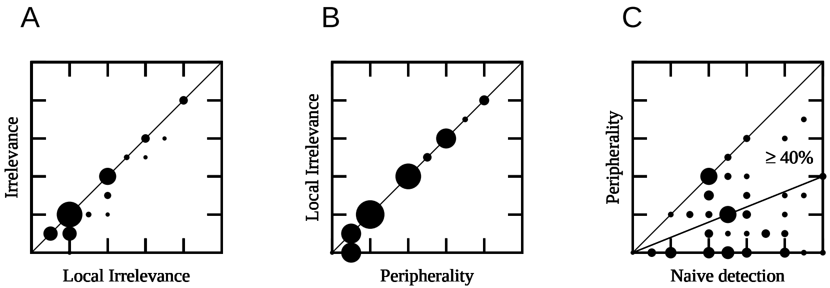
Larger point = more test cases



Benchmark 1: Effectiveness



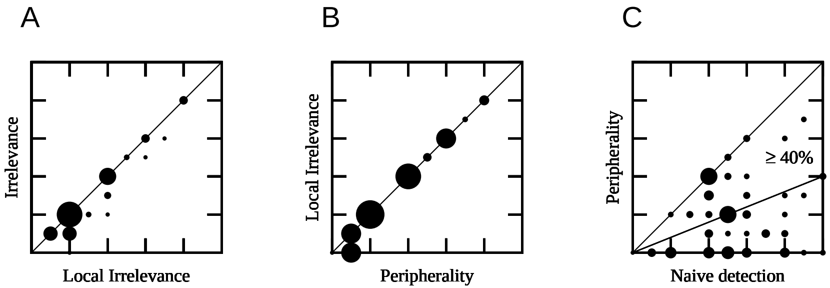
Benchmark 1: Effectiveness



Reduced # of extra model checking runs:

- $\geq$ 40% reduction in 54% of cases with vacuity

Benchmark 1: Effectiveness

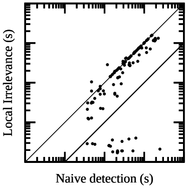


Reduced # of extra model checking runs:

- $\geq$ 40% reduction in 54% of cases with vacuity

Local Irrelevance is faster than Naive Detection in 70 cases (59%):

- Twice as fast in 40% of these cases
- Order of magnitud faster in 30% of these cases



Benchmark 2

GOAL: evaluate scalability of our tool to industrial models

SETUP

- Models and properties: IBM Formal Verification Benchmarks Library
- 18 properties:
  - 12 present vacuity
  - 1 temporal operator, from $\{G, F\}$
  - 4 variables on average, 17 max., 1 min.
- Picked k-depth in line with bounds used in Benchmark 1
- Largest proof: 500k resolutions

Benchmark 2

GOAL: evaluate scalability of our tool to industrial models

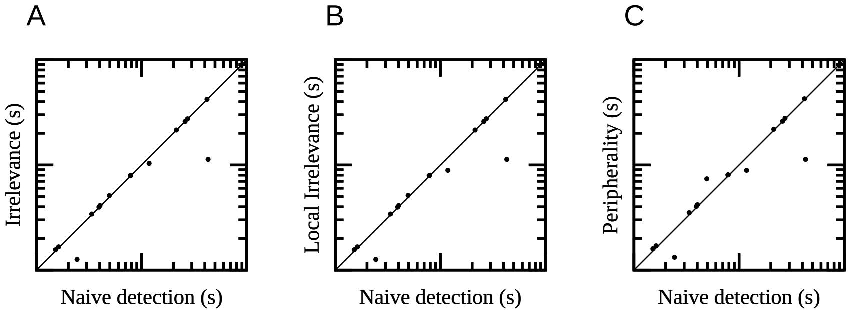
SETUP

- Models and properties: IBM Formal Verification Benchmarks Library
- 18 properties:
  - 12 present vacuity
  - 1 temporal operator, from $\{G, F\}$
  - 4 variables on average, 17 max., 1 min.
- Picked k-depth in line with bounds used in Benchmark 1
- Largest proof: 500k resolutions

Proof sizes are in same range as those for Benchmark 1

- new models are more complex
- but properties are simpler

Benchmark 2: Scalability



- Reasonable execution times
- No noticeable spike in peripherality execution times
  - models with low clause/variable ratio present vacuity
  - proofs for these models are medium-sized
- Little vacuity in this suite, yet algorithms detect some vacuity

Experimental Conclusions

	Benchmark 1	Benchmark 2
Models	Simple	Complex
Properties	Complex	Simple
Irrelevance	Very fast	Very fast
Local Irrelevance	Fastest	Fastest
Peripherality	Slow in certain cases	Very fast

Our algorithms:

- discover vacuous variables
- . . . via relatively inexpensive analyses of BMC artifacts

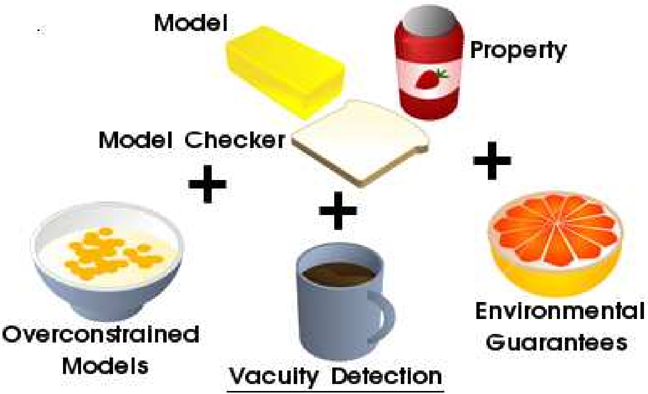
Our methods are complete and generally faster than Naive Detection

Summary

- Vacuity detection for BMC
  - we analyze BMC artifacts like UNSAT cores and resolution proofs
- Proposed and implemented a vacuity detection tool, VAQTREE

Summary

- Vacuity detection for BMC
  - we analyze BMC artifacts like UNSAT cores and resolution proofs
- Proposed and implemented a vacuity detection tool, VAQTREE
- Step towards making vacuity detection part of complete process



Future Work

- When do our algorithms apply?
  - heuristics based on clause/variable ratio and proof size
- Increase scalability of our tool
  - implement on-the-fly proof analysis
- Use interpolants for vacuity detection
- Use results of previous depths for vacuity detection

Thanks for your attention
Questions?