Getting readable proofs for replicated systems from automated provers

Michael Raskin

LaBRI

2024-01-16

Joint work with Javier Esparza and Christoph Welzel (TU Munich)

Setting: replicated systems

- Systems with many identical components
- Arbitrary number of components
- Safety conditions: never go into bad states
- Unbounded state space, risk of undecidability

Examples

- Dining philosophers
- Dijkstra's mutual exclusion algorithm
- Cache coherence
- ...

Setting: replicated systems

- Systems with many identical components
- Arbitrary number of components
- Safety conditions: never go into bad states
- Unbounded state space, risk of undecidability

Examples

- Dining philosophers
- Dijkstra's mutual exclusion algorithm
- Cache coherence
- ...

Is decidable model checking enough?

Some models are undecidable

… some are even undecidable for good reasons!

Unclear what parts of decision procedures are reusable

The answer «yes, all OK» is not always **all** you want to know

… and neither is 10MB of quickly-verifiable certificate

Can we get a simple explanation?

Is decidable model checking enough?

Some models are undecidable
… some are even undecidable for good reasons!

Unclear what parts of decision procedures are reusable

The answer «yes, all OK» is not always **all** you want to know
… and neither is 10MB of quickly-verifiable certificate

Can we get a simple explanation?

Is decidable model checking enough?

Some models are undecidable
… some are even undecidable for good reasons!

Unclear what parts of decision procedures are reusable

The answer «yes, all OK» is not always **all** you want to know
… and neither is 10MB of quickly-verifiable certificate

Can we get a simple explanation?

Is decidable model checking enough?

Some models are undecidable

… some are even undecidable for good reasons!

Unclear what parts of decision procedures are reusable

The answer «yes, all OK» is not always **all** you want to know

… and neither is 10MB of quickly-verifiable certificate

Can we get a simple explanation?

Is decidable model checking enough?

Some models are undecidable
… some are even undecidable for good reasons!

Unclear what parts of decision procedures are reusable

The answer «yes, all OK» is not always **all** you want to know
… and neither is 10MB of quickly-verifiable certificate

Can we get a simple explanation?

Is decidable model checking enough?

Some models are undecidable
… some are even undecidable for good reasons!

Unclear what parts of decision procedures are reusable

The answer «yes, all OK» is not always **all** you want to know
… and neither is 10MB of quickly-verifiable certificate

Can we get a simple explanation?

Dijkstra's mutual exclusion algorithm

(Just to pick *which* of Dijkstra's mutex...)

- n agents
- Each can execute a **critical section**
- To enter critical section an agent...
  - Sets an intent flag in its own shared memory block
  - Checks if any other agent has the intent flag set
  - Proceeds if no
- Some extra work to ensure progress
- Progress features cannot violate correctness

Dijkstra's mutual exclusion algorithm

(Just to pick *which* of Dijkstra's mutex...)

- n agents
- Each can execute a **critical section**
- To enter critical section an agent...
  - Sets an intent flag in its own shared memory block
  - Checks if any other agent has the intent flag set
  - Proceeds if no
- Some extra work to ensure progress
- Progress features cannot violate correctness

Dijkstra's mutual exclusion algorithm

(Just to pick *which* of Dijkstra's mutex...)

- n agents
- Each can execute a **critical section**
- To enter critical section an agent...
  - Sets an intent flag in its own shared memory block
  - Checks if any other agent has the intent flag set
  - Proceeds if no
- Some extra work to ensure progress
- Progress features cannot violate correctness

Dijkstra's mutual exclusion algorithm

(Just to pick *which* of Dijkstra's mutex...)

- n agents
- Each can execute a **critical section**
- To enter critical section an agent...
  - Sets an intent flag in its own shared memory block
  - Checks if any other agent has the intent flag set
  - Proceeds if no
- Some extra work to ensure progress
- Progress features cannot violate correctness

Implications of Dijkstra's mutex

We want to model DM and prove it's safe
What a good model of DM implies?

- Leader election
  What *else* is a mutex?
- Leader can find a follower in a specific state
- Minsky counter machine
- Undecidability

Oh well, maybe just use heuristics

Implications of Dijkstra's mutex

We want to model DM and prove it's safe
What a good model of DM implies?

- Leader election
  What *else* is a mutex?
- Leader can find a follower in a specific state
- Minsky counter machine
- Undecidability

Oh well, maybe just use heuristics

Implications of Dijkstra's mutex

We want to model DM and prove it's safe
What a good model of DM implies?

- Leader election
  What *else* is a mutex?
- Leader can find a follower in a specific state
- Minsky counter machine
- Undecidability

Oh well, maybe just use heuristics

Implications of Dijkstra's mutex

We want to model DM and prove it's safe
What a good model of DM implies?

- Leader election
  What *else* is a mutex?
- Leader can find a follower in a specific state
- Minsky counter machine
- Undecidability

Oh well, maybe just use heuristics

Generic heuristic search tools

Many tools for undecidable or just intractable manage to be good enough

- SAT-solvers
- SMT-solvers
- First-order logic automated theorem provers
- …

Generic heuristic search tools

Many tools for undecidable or just intractable manage to be good enough

- SAT-solvers
- SMT-solvers
- First-order logic automated theorem provers
- …

Generic heuristic search tools

Many tools for undecidable or just intractable manage to be good enough

- SAT-solvers
- SMT-solvers
- First-order logic automated theorem provers
- ...

Heuristic search in finite case: CEGAR

Finite state systems behaviours are decidable, but how explainable they are?

CEGAR (CounterExample-Guided Abstraction Refinement)
Recall the CEGAR loop:

- Is there a bad state compatible with current invariants?
  SMT-solver
- Can this bad state be reached?
  (optional)
- Any nice invariant separating reachable states from this bad one?
  SMT-solver
- If yes, add this invariant

Heuristic search in finite case: CEGAR

Finite state systems behaviours are decidable, but how explainable they are?

CEGAR (CounterExample-Guided Abstraction Refinement)
Recall the CEGAR loop:

- Is there a bad state compatible with current invariants?
  SMT-solver
- Can this bad state be reached?
  (optional)
- Any nice invariant separating reachable states from this bad one?
  SMT-solver
- If yes, add this invariant

Heuristic search in finite case: CEGAR

Finite state systems behaviours are decidable, but how explainable they are?

CEGAR (CounterExample-Guided Abstraction Refinement)
Recall the CEGAR loop:

- Is there a bad state compatible with current invariants?
  SMT-solver
- Can this bad state be reached?
  (optional)
- Any nice invariant separating reachable states from this bad one?
  SMT-solver
- If yes, add this invariant

Heuristic search in finite case: CEGAR

Finite state systems behaviours are decidable, but how explainable they are?

CEGAR (CounterExample-Guided Abstraction Refinement)
Recall the CEGAR loop:

- Is there a bad state compatible with current invariants?
  SMT-solver
- Can this bad state be reached?
  (optional)
- Any nice invariant separating reachable states from this bad one?
  SMT-solver
- If yes, add this invariant

Heuristic search in finite case: CEGAR

Finite state systems behaviours are decidable, but how explainable they are?

CEGAR (CounterExample-Guided Abstraction Refinement)
Recall the CEGAR loop:

- Is there a bad state compatible with current invariants?
  SMT-solver
- Can this bad state be reached?
  (optional)
- Any nice invariant separating reachable states from this bad one?
  SMT-solver
- If yes, add this invariant

CEGAR loop

Heuristic loop; possible outcomes:

- Reachable bad state
- Set of invariants excluding all bad states
- Unreachable bad state not preventable by nice invariants

Invariant description is simpler in finite-state systems
We either finish, or exhaust states
… or hang the SMT-solver … or run out of RAM
… or run out of patience

CEGAR loop

Heuristic loop; possible outcomes:

- Reachable bad state
- Set of invariants excluding all bad states
- Unreachable bad state not preventable by nice invariants

Invariant description is simpler in finite-state systems
We either finish, or exhaust states
… or hang the SMT-solver … or run out of RAM
… or run out of patience

CEGAR loop

Heuristic loop; possible outcomes:

- Reachable bad state
- Set of invariants excluding all bad states
- Unreachable bad state not preventable by nice invariants

Invariant description is simpler in finite-state systems
We either finish, or exhaust states
… or hang the SMT-solver … or run out of RAM
… or run out of patience

CEGAR loop

Heuristic loop; possible outcomes:

- Reachable bad state
- Set of invariants excluding all bad states
- Unreachable bad state not preventable by nice invariants

Invariant description is simpler in finite-state systems
We either finish, or exhaust states
… or hang the SMT-solver … or run out of RAM
… or run out of patience

Replicated systems

- Some special language for describing invariants
- Even when decidable (WS1S), we still want explanations!

Strategy: generalise CEGAR invariants from specific size

Replicated systems

- Some special language for describing invariants
- Even when decidable (WS1S), we still want explanations!

Strategy: generalise CEGAR invariants from specific size

Our work

- Make fixed-size instances
- Run CEGAR there
- Encode the invariants in WS1S or just first-order logic
- Encode the system in WS1S or FOL
- Generalise

  … heuristically

  … or by proving sound abstraction rules

Our work

- Make fixed-size instances
- Run CEGAR there
- Encode the invariants in WS1S or just first-order logic
- Encode the system in WS1S or FOL
- Generalise

  … heuristically

  … or by proving sound abstraction rules

Sound abstraction rule: example

Where sound abstraction rules come from? Symmetry

Let system definition be stable under component permutation

Let interactions be pairwise

CEGAR finds inductive invariant:
«at size 5,
component 1 has state ‘a’ or component 2 has state ‘b’»

Then this holds for every size above four and every two components,
by symmetry

Sound abstraction rule: example

Where sound abstraction rules come from? Symmetry

Let system definition be stable under component permutation

Let interactions be pairwise

CEGAR finds inductive invariant:
«at size 5,
component 1 has state ‘a’ or component 2 has state ‘b’»

Then this holds for every size above four and every two components,
by symmetry

Sound abstraction rule: example

Where sound abstraction rules come from? Symmetry

Let system definition be stable under component permutation

Let interactions be pairwise

CEGAR finds inductive invariant:
«at size 5,
component 1 has state ‘a’ or component 2 has state ‘b’»

Then this holds for every size above four and every two components,
by symmetry

Sound abstraction rule: example

Where sound abstraction rules come from? Symmetry

Let system definition be stable under component permutation

Let interactions be pairwise

CEGAR finds inductive invariant:
«at size 5,
component 1 has state ‘a’ or component 2 has state ‘b’»

Then this holds for every size above four and every two components,
by symmetry

Heuristic abstraction

Either component 2 is in state ‘a’,
or component 7 is in state ‘b’,
or one of the components 3,4,5,6 is in state ‘c’.

Maybe…
for each $i < j$: ‘a’ at i or ‘b’ at j or ‘c’ between?

Heuristic abstraction

Either component 2 is in state ‘a’,
or component 7 is in state ‘b’,
or one of the components 3,4,5,6 is in state ‘c’.

Maybe…
for each $i < j$: ‘a’ at i or ‘b’ at j or ‘c’ between?

Using heuristic abstraction

We have a guess, so what

- Encode it in the first order logic
- Encode the system in first order logic
- Ask an automated theorem prover if the guess is invariant

How many invariants do we need anyway?

- Try a new size
- Instantiate all generic invariants
- If not enough for one size, try to get more
- Ask FOL ATP if invariants imply safety

Using heuristic abstraction

We have a guess, so what

- Encode it in the first order logic
- Encode the system in first order logic
- Ask an automated theorem prover if the guess is invariant

How many invariants do we need anyway?

- Try a new size
- Instantiate all generic invariants
- If not enough for one size, try to get more
- Ask FOL ATP if invariants imply safety

Using heuristic abstraction

We have a guess, so what

- Encode it in the first order logic
- Encode the system in first order logic
- Ask an automated theorem prover if the guess is invariant

How many invariants do we need anyway?

- Try a new size
- Instantiate all generic invariants
- If not enough for one size, try to get more
- Ask FOL ATP if invariants imply safety

Readability

For Dijkstra's mutex, it's a handful

One of them:
«if processes p_0 and p_1 are both actively iterating,
p_0's pointer has not yet reached p_1 or vice versa»

(Quite annoying to formalise just right by hand the first time)

Readability

For Dijkstra's mutex, it's a handful

One of them:
«if processes p_0 and p_1 are both actively iterating,
p_0's pointer has not yet reached p_1 or vice versa»

(Quite annoying to formalise just right by hand the first time)

Technicalities

- Things get encoded: WS1S, Petri nets, Regular transition systems
- Solvers: SAT-solver, SMT-solver, ATP
  Z3, Z3, mainly Vampire + Eprover + CVC4
- Invariants: traps ($\Sigma \geq 1$), balanced sets ($\Sigma = 1$)
- Generalisations: logic of order, regular languages of traps

Applicability

How specific are parts?

Choices

- Model for describing the original system
- Model for studying finite instances
- Logic for general case

Things to have

- Translation to finite instances
- Translation to parametric definition
- CEGAR loop
- Generalisation heuristics

The most «creative» parts are easiest to reuse between areas

Applicability

How specific are parts?

Choices

- Model for describing the original system
- Model for studying finite instances
- Logic for general case

Things to have

- Translation to finite instances
- Translation to parametric definition
- CEGAR loop
- Generalisation heuristics

The most «creative» parts are easiest to reuse between areas

Applicability

How specific are parts?

Choices

- Model for describing the original system
- Model for studying finite instances
- Logic for general case

Things to have

- Translation to finite instances
- Translation to parametric definition
- CEGAR loop
- Generalisation heuristics

The most «creative» parts are easiest to reuse between areas

Performance observations

So we have...
... CEGAR loop, using solvers of NP-complete SAT
... and FOL ATP, striving at an undecidable problem

Good cases: ATP takes much *less* time than SAT-solver
Bad cases: CEGAR fails for a specific size

Easy not to be undecidability-limited here!
Future work:

- Better invariant classes
- Better models to run CEGAR loop in?

Performance observations

So we have...
... CEGAR loop, using solvers of NP-complete SAT
... and FOL ATP, striving at an undecidable problem

Good cases: ATP takes much *less* time than SAT-solver
Bad cases: CEGAR fails for a specific size

Easy not to be undecidability-limited here!
Future work:

- Better invariant classes
- Better models to run CEGAR loop in?

Performance observations

So we have...
... CEGAR loop, using solvers of NP-complete SAT
... and FOL ATP, striving at an undecidable problem

Good cases: ATP takes much *less* time than SAT-solver
Bad cases: CEGAR fails for a specific size

Easy not to be undecidability-limited here!
Future work:

- Better invariant classes
- Better models to run CEGAR loop in?

Performance observations

So we have...
... CEGAR loop, using solvers of NP-complete SAT
... and FOL ATP, striving at an undecidable problem

Good cases: ATP takes much *less* time than SAT-solver
Bad cases: CEGAR fails for a specific size

Easy not to be undecidability-limited here!
Future work:

- Better invariant classes
- Better models to run CEGAR loop in?

Thanks for your attention!

Questions?

Conclusions

- Some parametric systems need heuristic approach
- First-order logic theorem provers can be useful
- Proofs have readable summaries
- Undecidability is not the limiting factor in practice
- We need more and better invariants!