STRUCTURAL REDUCTIONS REVISITED

Yann Thierry-Mieg
LIP6, Sorbonne Université, CNRS
VERIFYING PROPERTIES OF PETRI NETS

Properties of interest

**Deadlock Detection**

- AirplaneLandingGear

**Safety Properties**

- EGFr receptor

Can a deadlock state be reached?

Is « m(P1) < m(P2) OR m(p3) <= 2 » an invariant?
EXPLORING THE STATE SPACE

*Petri net vs. State space (marking graph)*

- Do reachable and « bad » states intersect?
VERIFICATION OF AN INVARIANT

Petri net vs. State space (marking graph)

• Does my invariant hold in all reachable states of the net?

Empty intersection
We **cannot** reach a bad state
Invariant is TRUE

Non-empty intersection
We can reach a bad state
Invariant is FALSE
OUR APPROACH

Three complementary strategies

1. Over-approximation
   Can formally prove **TRUE** invariants
   SMT based constraints to approximate reachable states

2. Under-approximation
   Can **contradict** **FALSE** invariants if it can produce a counter-example
   Sampling using a pseudo-random walk

3. Property preserving reduction
   Produce a smaller net that preserves existence of reachable bad states
   Property specific structural reduction rules
1. OVER-APPROXIMATE WITH SMT

*Leveraging SAT Modulo Theory SMT*

- Describe constraints on reachable states: an envelope

- The envelope is a much simpler object than the actual state space.
1. OVER-APPROXIMATE WITH SMT

Can we find an bad state in the envelope?

NO INTERSECTION (UNSAT)

Over-approximation => Invariant holds.

WITH INTERSECTION (SAT)

False Positive

OR

Over-approximation => INCONCLUSIVE

but we can provide a candidate solution (SAT model).
SMT CONSTRAINTS

Highlights

• Places = variables
  • \( P_1 \geq 0, P_2 \geq 0 \ldots \)

• Generalized flows
  • \( P_1 + 2P_2 - P_3 = 1 \)

• Trap constraints
  • \( P_1 > 0 \) OR \( P_2 > 0 \)
  • Compute *useful constraints* as separate SMT problem

• State Equation
  • Add a positive variable for firing count of transitions
  • \( P_1 = T_1 - T_2 + 1 \)

• Read => Feed
  • \( T_1 \) reads \( P \); \( m_0(P)=0 \); \( T_2 \) and \( T_3 \) feed \( P \)
  • \( T_1 > 0 \) => \( T_2 > 0 \) OR \( T_3 > 0 \)

• Causal constraints (*precedes* is a strict partial order)
  • \( T_1 \) consumes from \( P \); \( m_0(P)=0 \); \( T_2 \) and \( T_3 \) feed \( P \)
  • \( T_1 > 0 \) => \( (T_2>0 \text{ AND } T_2 \text{ precedes } T_1) \) OR \( (T_3 > 0 \text{ AND } T_3 \text{ precedes } T_1) \)
  • Is inconsistent (UNSAT) if we also have « \( T_1 \text{ precedes } T_2 \) » and « \( T_1 \text{ precedes } T_3 \) »
TRAP CONSTRAINTS

An initially marked trap cannot be emptied

• A trap is a set of places of the net
  • Any transition consuming from the trap must also feed the trap

• As noted by Esparza et al. in 2000, good complement to state equation
  • Complex mutex protocols e.g. Peterson, Lamport
  • But worst case exponential number of traps

• Iterative process:
  • When main SMT procedure is SAT: examine candidate solution
  • We use a separate SMT solver to find relevant traps:
    • Can we find an initially marked trap that is unmarked in the candidate?
      • SAT => add the trap constraint to main engine and try again
      • UNSAT => no trap constraints that contradict the candidate exist
Constraining the transition firing count

- The state equation ignores read arcs
  ⇒ spurious solutions, $t_1$ and $t_2$ are not correlated in the state equation constraints

Reason on first occurrence of each transition:
- If a transition has positive firing count and reads in place « p » initially empty, it must be the case that a transition feeding « p » also has positive firing count.
  \[ t_1 > 0 \implies t_2 > 0 \]
CAUSAL CONSTRAINTS (UNSAT)

A partial order on first occurrence of each transition

The state equation can borrow non-existing tokens

⇒ t1=1 and t2=1 is a solution to the state equation to mark « p »

We assert that:

• t1 > 0 ⇒ t2 > 0 and t2 precedes t1
• t2 > 0 ⇒ t1 > 0 and t1 precedes t2

Obtaining a contradiction (UNSAT) as soon as t1 or t2 positive in the solution
CAUSAL CONSTRAINTS (SAT)

A partial order on first occurrence of each transition

The state equation can borrow non existing tokens
⇒ t1=1 and t2=1 is a solution to the state equation to mark « p »

We assert that:
• t1 > 0 ⇒ t2 > 0 and t2 precedes t1
• t2 > 0 ⇒ (t1 > 0 and t1 precedes t2) OR (t3 > 0 and t3 precedes t2)

Obtaining a solution (SAT) : t3 precedes t2 precedes t1
2. UNDER-APPROXIMATE WITH SAMPLING

*Memory-less random exploration of the state space*

- Execute the net, trying to find a reachable bad state
2. UNDER-APPROXIMATE WITH SAMPLING

Memory-less pseudo-random walk of the state space

- Execute the net, trying to find a reachable bad state (counter-example)

If an bad state is met => Invariant DOES NOT hold.
Otherwise INCONCLUSIVE :
- we might have been unlucky and not found the bug,
- or the bug might genuinely not exist.
RANDOM WALKS

Highlights

• Fast sparse implementation
  • Avoid TxT or PxP matrices

• Some states exponentially unlikely to be met by pure random walk
  • Use multiple heuristics each with a strong bias

• Guiding the walk:
  • Pure random walk with resets
  • Guided by a firing count coming from an SMT « SAT » result
  • Guided by the property (choose « best » successor w/ heuristic)
  • Recently enabled / Not recently used
  • ...

• Random walk is fast and scales well
  • Always first try to disprove with random walk before trying to prove with SMT.

+Fast results in many FALSE cases
+Disprove by counter-example
+Complements SMT TRUE proofs
+Guided by SMT inconclusive SAT
3. PROPERTY SPECIFIC STRUCTURAL REDUCTIONS

Incrementally build a smaller net using **structural reduction rules**

Each transformation **rule** produces a net $N'$ that satisfies the property **if and only if** original net $N$ satisfies it.

Reduction of the Petri net structure typically induces an **exponential** state space reduction.
PROPERTY SPECIFIC?

Properties of interest

Deadlock Detection

Safety Properties

AirplaneLandingGear

Can a deadlock state be reached?

=> Existence of at least one finite trace.

Specific rules preserving only unavoidable loops.

Is « m(P1) < m(P2) OR m(P3) <= 2 » an invariant?

Focus on a projection of reachable states over the places in the support.
No unavoidable SCC => Deadlock unavoidable!

AirplaneLandingGear

Can a deadlock state be reached?

=> Existence of at least one finite trace.

Specific rules preserving only unavoidable loops.

Is « m(P1) < m(P2) OR m(p3) <= 2 » an invariant?

Focus on a projection of reachable states over the places in the support.
PROPERTY SPECIFIC?

Properties of interest

**Deadlock Detection**

AirplaneLandingGear

Can a deadlock state be reached?

=> Existence of **at least one** finite trace.

*Specific rules preserving only unavoidable loops.*

**Safety Properties**

EGFr receptor

Is « m(P1) < m(P2) OR m(p3) <= 2 » an invariant?

Focus on a **projection** of reachable states over the places in the **support.**
Properties of interest

Deadlock Detection

AirplaneLandingGear

Blue cannot influence red!
Discard!

Can a deadlock state be reached?

=> Existence of at least one finite trace.

Specific rules preserving deadlock freedom.
Graph Based Rules

Reason on an abstraction of the net structure

- Compute the prefix in the influence graph of places in the support of the property
- **Brutally discard** places and transitions outside this prefix
- Two variants of the rule
  - For Deadlocks focus on SCC of the graph and their prefix:
    - Side effect: if there are no SCC, the net contains deadlocks.
  - For Safety, focus on places in the support
    - Asymmetric effect of read arcs: Places that are controlled by the places of interest are not interesting.
**FREE AGGLOMERATION**

*Safety preserving agglomeration*

- Two cases:
  - If \( t_2 \) was actually fireable originally, \( t_1.t_2 \) is still fireable, no behavior is lost
  - If \( t_2 \) was not fireable, now \( t_1.t_2 \) is not fireable, so we lost the possibility of firing \( t_1 \); but
    - \( t_1 \) stutters
    - \( t_1 \) can only feed \( p \), so firing \( t_1 \) is *weakening* the rest of the net

- Free-agglomeration preserves safety but not deadlocks
  - Firing \( t_1 \) and then being unable to fire \( t_2 \) can lead to a deadlock.
A place is *structurally implicit* iff. it never prevents any transition from firing

- In any marking, if a transition \( t \) consuming from \( p \) is enabled without considering \( p \), then \( p \) *always* contains enough tokens to fire \( t \)
- Build an SMT problem, asserting this invariant
- Discard \( p \) if the invariant can be proved

- Can help start another round of reductions
  - Powerful test though more costly than most rules
  - Covers variants of "redundant place" rules in e.g. Berthelot.
STRUCTURAL REDUCTION RULES

Highlights

• Total of 22 rules presented in the paper

• Basic rules:
  • equal places, constant place, sink place, …
  • neutral transition, dominated transition…

• Advanced rules:
  • Unmarked Syphon, Future equivalent places, token movement

• Agglomeration based rules:
  • pre and post-agglomeration,
  • new « free » agglomeration

• Graph based rules:
  • Compute SCC or a prefix of nodes in an abstraction of the net structure
  • Notion of « Prefix of interest » for deadlock and invariants
  • Fusion of « free » SCC

• Structural reductions supported by SMT over-approximation
  • Structurally dead transitions
  • Structurally implicit places

+preserves properties of interest
+memory and time efficient
+simplifies the net for any analysis
+synergy with over/under approximations
+leverage SMT component for more reduction power
EVALUATION

Validation with Model-Checking Contest 2019 nets and formulas

• Examination = (model + 16 invariants) or (model + deadlock)
  • Select all examinations with known results in 2019 (produced by any tool):
  • 90 model families, 2680 examinations, 28900 properties
  • Max runtime 12 minutes, 8GB RAM
    • 21/2680 : 0.008 % timeout
    • On average 31 seconds per examination

• Deadlocks:
  • 902 / 932 : 96.8 % solved

• Invariants:
  • 1634 / 1748 : 93.5 % fully solved all 16 invariants
  • 27594 / 27968 : 98.6 % of formulas solved

• Resulting nets when not fully solved are much smaller
CONCLUSION

Structural Reductions Revisited

- Combine three complementary strategies
- Competing as a «filter» within the model-checking contest in «its-tools» and «its-lola»
- Full graphical examples used in this presentation

https://lip6.github.io/ITSTools-web/structural