NOKIA Bell Labs

DARPA NSF

Kedar Namjoshi
Bell Labs, Nokia

VMCAI 2019 Winter School
11 Jan 2019

NOKIA Bell Labs

A talk of many things…

- What is compositional verification?
- What makes it important? Difficult? Fascinating?
- Compositional Symmetry Reduction
- Parameterized Compositional Verification

“The time has come,” the Walrus said,
“To talk of many things:
Of shoes--and ships--and sealing-wax--
Of cabbages--and kings--
And why the sea is boiling hot--
And whether pigs have wings.”

from *Through the Looking-Glass and What Alice Found There*, Lewis Carroll, 1872.

Compositional Verification

What is Compositional Verification?

Key Idea: Break up a program proof into local questions of limited scope.

> How can one check a routine in the sense of making sure that it is right?
>
> In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

(*Checking a Large Routine*, Alan Turing, 1949)

For sequential programs, to prove {P} S1;S2 {Q}, invent "midpoint" assertion R and establish {P} S1 {R} and {R} S2 {Q} instead.

Why Compositional Verification?

Key: Break up a program proof into local questions of limited scope.

We'll focus on compositional verification for concurrent, shared-memory programs.

- State Explosion

  A program with N concurrent components can have a global state space of size 2^N
  (In theory, verification is PSPACE-hard in N.)

  Compositional verification is one way to ameliorate this "State Explosion."

- Loose Coupling

  Large, scalable programs are expected to have "loosely coupled" components.

  Compositional methods aim to exploit loose coupling.

What is a Compositional Verification Method?

Key: Break up a program proof into local questions of limited scope.

A (de)compositional verification method should have

1. A sound strategy for breaking up a proof of goal φ for program $P = P_1 \mid\mid \ldots \mid\mid P_N$ into independent proofs of subgoals θ_i for each P_i
2. A solution for possible incompleteness in this abstraction, and
3. A mechanical method to achieve #1 and #2.

We'll focus on program invariants.

Running Example: Dining Philosophers

Philosophers $P_0 \ldots P_{N-1}$ sit around a circular table, with forks f_0 … f_{N-1} between them.

1. Each philosopher cycles though states Thinking, Hungry, and Eating.
2. Initially, all philosophers are Thinking, and every fork is available.
3. A philosopher may eat only if it has picked up its left and right forks.
4. To avoid deadlock, a hungry philosopher may put down a fork it has picked up earlier (this ensures mutual exclusion but not starvation-freedom).
5. After eating, a philosopher makes both its forks available.

The goal is to show that no pair of neighboring philosophers may eat at the same time.

Dining Philosophers: Global Invariance

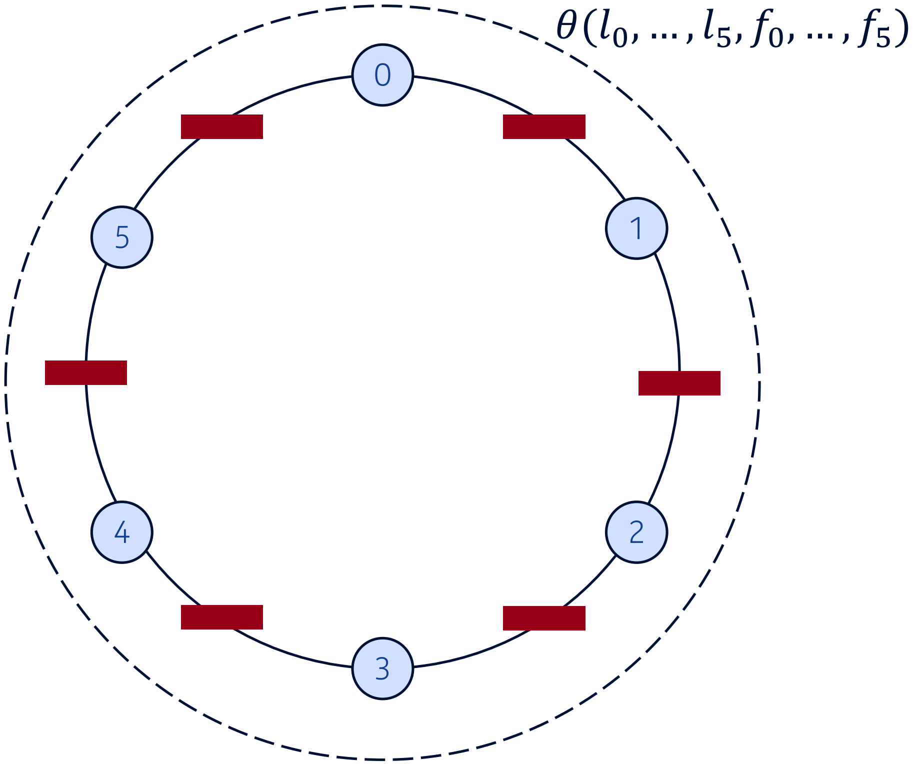
A global invariant is an assertion, $\theta(l_0, \ldots, l_{N-1}, f_0, \ldots, f_{N-1})$, over the global state space formed by local variables $\{l_i\}$ and shared variables $\{f_i\}$.

From an experiment with SPIN:

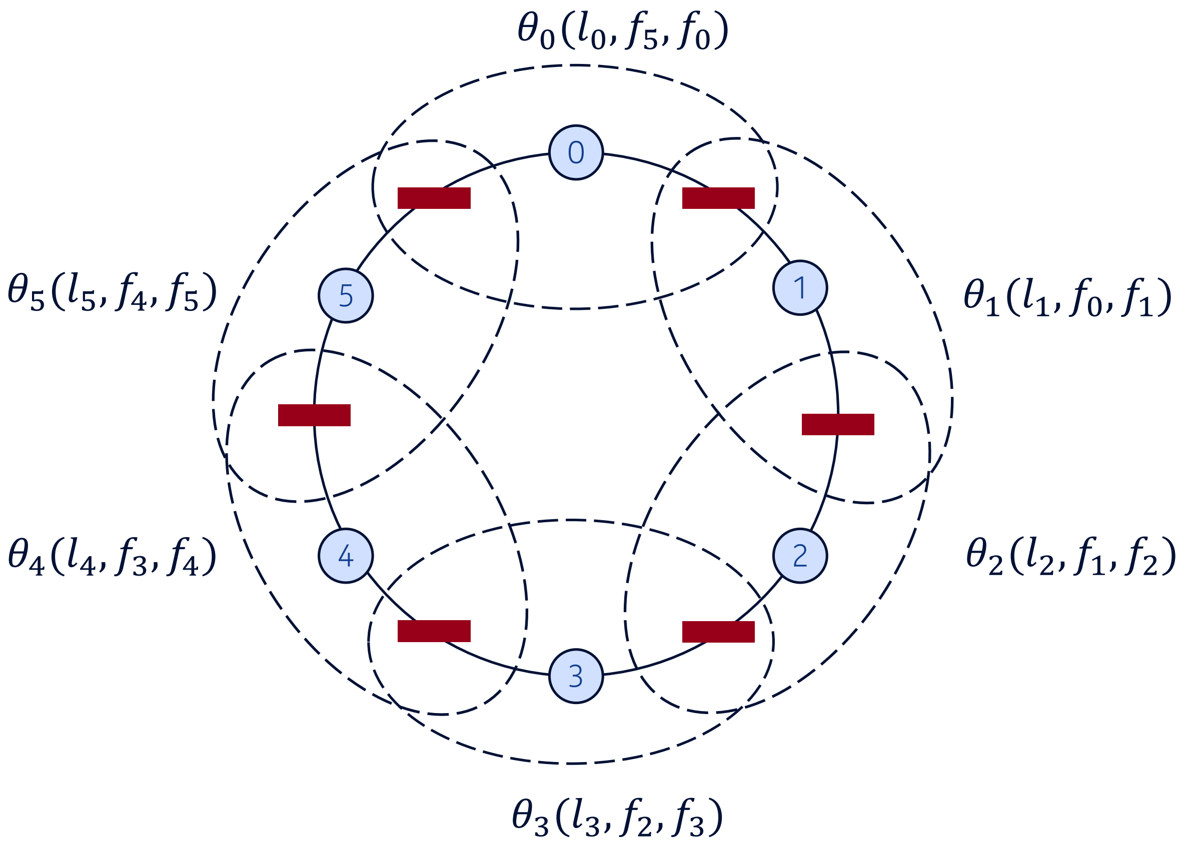
Num. Proc.	Time (sec.)	Num. Reachable States
5	0.12	91808
6	2.4	902800
7	34	8874944
8	(killed)	(memory > 2.5GB)

Compositional analysis takes less than 1 second, for over 3000 processes.

Dining Philosophers: Global Invariance in a Picture



Dining Philosophers: Compositional Invariance in a Picture



“Loosely Coupled”
Local Invariants

NOKIA Bell Labs

Compositional Invariance

A compositional invariant is a collection of per-process assertions $\{\theta_i\}$ such that for each i:

1. The assertion θ_i is limited to the "neighborhood" of process P_i: i.e., the state space induced by the local variables of this process and its adjacent shared variables.
2. θ_i is an inductive invariant for process P_i
3. The assertion θ_i is immune to "interference" (via shared variables) from actions of adjacent processes. I.e., $[\theta_i \wedge \theta_j \wedge T_j \Rightarrow \theta_i']$ is valid.

(These are just the Owicki-Gries proof rules, freed from program syntax.)

Theorem: For a compositional invariant $\{\theta_i\}$, the conjunction $(\wedge\, i\colon \theta_i)$ is a global inductive invariant.

Calculating a Compositional Invariant

A process P_i is defined by its initial states I_i and transition relation T_i .

The conditions for compositional invariance are:

1. (Inductiveness) $[I_i \Rightarrow \theta_i]$ and $[\theta_i \wedge T_i \Rightarrow \theta_i']$
2. (Non-Interference) $[\theta_i \wedge \theta_j \wedge T_j \Rightarrow \theta_i']$

Collecting constraints for each i, we get a set of simultaneous implications

$$\{[F_i(\theta) \Rightarrow \theta_i']\}$$

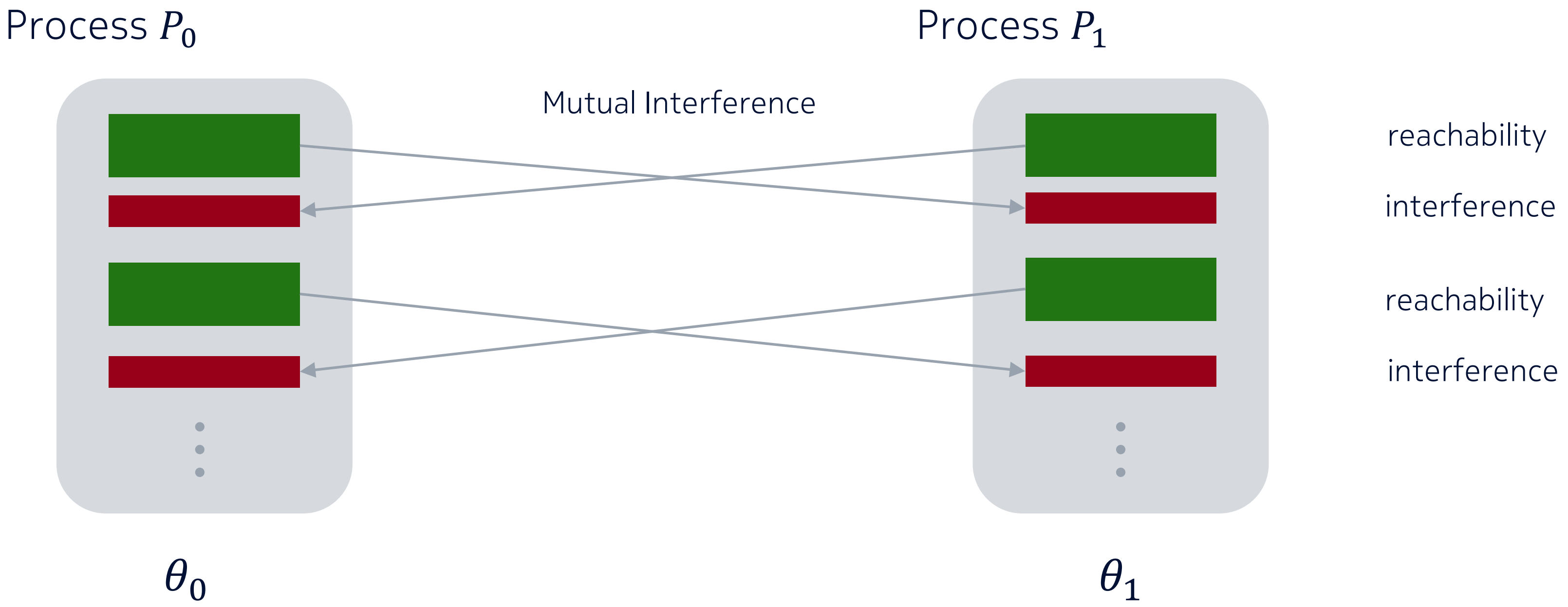
where each F_i is a monotone operator on the θ's. E.g., for the 6-node ring:

$$F_0(\theta) = I_0 \vee next_0(T_0, \theta_0) \vee next_0(T_5, \theta_0 \wedge \theta_5) \vee next_0(T_1, \theta_0 \wedge \theta_1)$$

Theorem: The strongest compositional invariant $\{\theta_i^*\}$ is the least simultaneous fixpoint of the operators $\{F_i\}$. Computational complexity is **polynomial** in N.

Calculating a Compositional Invariant: in a Picture

The fixpoint may be calculated by alternating reachability with interference-closure.



NOKIA Bell Labs

Calculating a Compositional Invariant-II

The conditions for compositional invariance are in Horn-clause form with unknowns $\{\theta_i\}$:

1. (Inductiveness) $[I_i \Rightarrow \theta_i]$ and $[\theta_i \wedge T_i \Rightarrow \theta_i']$
2. (Non-Interference) $[\theta_i \wedge \theta_j \wedge T_j \Rightarrow \theta_i']$

A Horn-clause solver may be also used to find a solution
(which need not be the strongest compositional invariant).

Incompleteness

A compositional invariant is generally weaker than the set of reachable states. Thus, some properties may be true but unprovable compositionally.

Example:

```
var lock : {0 ,1} , initially lock =1

process P(i) {
   while ( true ) {
       Thinking:
       Hungry: atomic {if ( lock =1) then lock := 0}
       Eating: lock := 1
   }
}
```

The strongest compositional invariant is $\theta_i = true$!

(In)completeness

This can be remedied by adding **auxiliary** shared variables to “couple” the local state spaces together more tightly.

Example:

```
var lock : {0 ,1} , initially lock =1
var last : 0..N, initially 0                    // the last process to Eat

process P(i) {
   while ( true ) {
       Thinking:
       Hungry: atomic { if ( lock =1) then lock := 0; last := i }
       Eating: lock := 1
   }
}
```

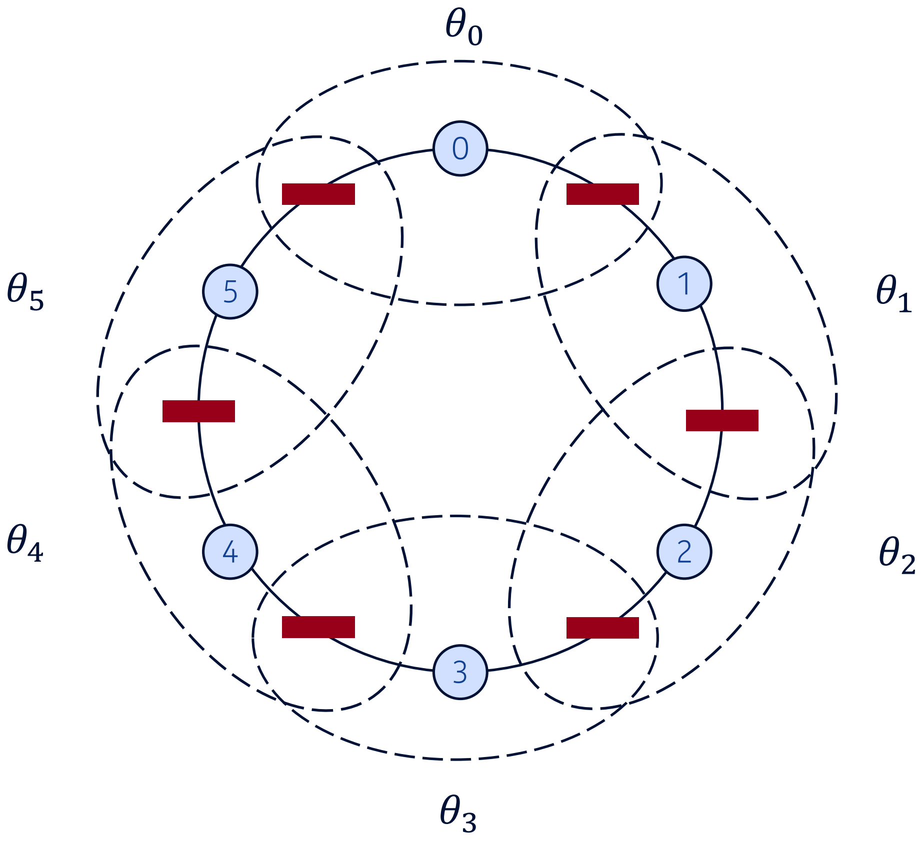
Now the strongest compositional invariant is $\theta_i = (Eating(i) \Leftrightarrow (lock = 0) \wedge (last = i))$

Open Questions

- What sort of auxiliary information is needed for typical proofs? Are scalable programs truly loosely coupled? I.e., is the auxiliary information simple and minimal?
- Can the right auxiliary variables be discovered automatically?
  [for an initial attempt, see Cohen & Namjoshi, CAV 2008]
- What hints could a programmer give to simplify such inference?
- How should one design (perhaps, synthesize) programs with easy modular proofs? Are new kinds of type systems/interface specifications needed?

Neighborhood Symmetry

Dining Philosophers: Compositional Symmetry



Which symmetries are appropriate for compositional invariants?

Neighborhood Symmetry

- Nodes m and n have symmetric neighborhoods if there is a bijection between the edges adjacent to the nodes.

  In the ring, the edges of node m are the forks $\{f_{m-1}, f_m\}$

- **Conjecture**: if nodes m and n have symmetric neighborhoods and the processes placed on them, P_m and P_n, are isomorphic, then in any compositional invariant, θ_m and θ_n are isomorphic.

  This is not quite true.

  One needs a **recursive** notion of neighborhood symmetry, called "balance."

  (cf. Martin Golubitsky and Ian Stewart: *Nonlinear dynamics of networks: the groupoid formalism*, Bull. AMS, 2006.)

Balance Relations

- A **local symmetry** is a triple (m, β, n) where β is a bijection on the edges of m and n. (E.g., for a ring, the bijection $\beta_{\{m,n\}}$ maps $f_{m-1} \mapsto f_{n-1}$ and $f_m \mapsto f_n$)

- A **balance relation** is a set of local symmetries satisfying a bisimulation-like constraint

If (m, β, n) is in the relation and node k is adjacent to m
there is a node l adjacent to n such that for some γ
(k, γ, l) is in the relation
and β and γ agree on all common edges.

Theorem: If (m, β, n) is in a balance relation and the processes P_m and P_n are isomorphic, the <u>strongest</u> compositional invariants θ_m^* and θ_n^* are also isomorphic up to β.

Discovering Balance

- Many networks with regular structure are fully balanced (e.g., ring, mesh, torus, hypercube). Note these have limited global symmetry.

- Every global symmetry group induces a balance relation
  - Star and complete networks are fully balanced as well
  - In fact, any transitive global symmetry group induces a full balance relation

- Even irregular networks can be balanced, after a bit of neighborhood abstraction

Using Balance for Symmetry Reduction

- Consider a ring with N nodes: any two nodes are balanced.
  - Hence, in the strongest compositional invariant, θ_m^* and θ_n^* are isomorphic
  - So it suffices to compute just <u>one</u> component, say θ_0^*, and derive the others by symmetry
  - The complexity thus reduces from $Poly(N)$ to a **constant**!

- More generally, the computation is limited to the representatives for each balance class.

Parameterized Compositional Verification

- Scalable programs are typically parametric in the number of processes
- The parameterized model checking problem (PMCP) is to **automatically** validate all instances of a parametric program
  - This is undecidable in general [Apt-Kozen 1986]
  - Only a handful of decidable cases are known
- What if we weaken the requirement to the construction of a **modular** parametric proof? This is the parameterized compositional model checking problem (PCMCP)
  - It is also undecidable in general [Namjoshi-Trefler 2016]
  - However, **many undecidable cases of PMCP become decidable for PCMCP**

Parameterized Compositional Verification – Decidability

- The key to decidability is exploiting balance across arbitrary-sized configurations.

- E.g., consider Dining Philosophers on rings.
  - By balance reduction, it suffices to compute θ_0^* for a ring of size N
  - This computation is **independent** of the value of N
  - Hence, θ_0^* computed on a ring of size 2 produces a compositional invariant for all N!
  - Thus, the PCMCP is decidable for rings – note that the PMCP is undecidable for rings (quite easily, too)

- A similar argument shows the decidability of PCMCP for other constant-degree networks such as mesh and tori.

- PCMCP is also decidable for non-constant degree networks (e.g., one control and many user processes) under restrictions.

Balancing Irregular Networks with Abstraction

- Example: Dining Philosophers on an arbitrary graph.
  A philosopher must gather forks on all adjacent edges in order to eat.
- Abstract philosopher behavior by ignoring the number of edges, retaining only the predicate “all forks are acquired”. Under this abstraction, any two nodes are balanced.
- By previous arguments, the PCMCP is decidable for Dining Philosophers over arbitrary graphs.
- A similar argument applies to the PCMCP over **dynamic** graphs as well.

Open Questions

- What types of abstractions are most useful for compositional reasoning?
- Can the right abstractions be inferred automatically?
- How can one add auxiliary variables (i.e., ensure completeness) while preserving balance?
- Could one use the symmetry results to design (perhaps, synthesize) parametric programs with easy modular proofs?

What I haven’t talked about

- Behavior-based compositional verification rules. E.g., infer $M_1 || M_2 \vDash \varphi$ from

$$M_1 || A_2 \vDash A_1$$
$$M_2 || A_1 \vDash A_2$$
and $A_1 || A_2 \vDash \varphi$

- Automated learning-based algorithms to infer adequate A_1 and A_2.

- Compositional rules for liveness properties and their delicate soundness proofs.

- And anything remotely practical ☺

To Sum Up

- Modular reasoning is absolutely essential to understanding and designing complex programs.

- Fundamental, fascinating questions about modular verification remain open.

- The ideal is a program design method that uses modular assertions to ease verification.

Thanks to

- Richard Trefler (U. Waterloo)
- Ariel Cohen (NYU), Lenore Zuck (UIC), Yaniv Sa’ar (Weizmann)
- Dimitra Giannakopoulou and Corina Pasareanu (NASA Ames)
- DARPA and the NSF, for supporting this research

Background Reading List – I

- **Books and Surveys**
  - de Roever et al: Concurrency Verification: **Introduction to Compositional and Noncompositional Methods**, 2001
  - Giannakopoulou, D., Namjoshi, K.S., Pasareanu, C. Compositional Reasoning, **Handbook of Model Checking**, 2018
- **Seminal Work**
  - Owicki, S.S., Gries, D.: Verifying properties of parallel programs: an axiomatic approach. Commun. ACM 1976
  - Lamport, L.: Proving the correctness of multiprocess programs. Trans. Softw. Eng., 1977
  - Misra, J., Chandy, K.M.: Proofs of networks of processes. Trans. Softw. Eng. 1981
  - Jones, C.B.: Tentative steps toward a development method for interfering programs, TOPLAS 1983
- **Behavior-based Rules**
  - Grumberg, O., Long, D.E.: Model checking and modular verification, TOPLAS 1994
  - Alur, R., Henzinger, T.A.: Reactive modules, FMSD 1999
  - McMillan, K.L.: Circular compositional reasoning about liveness, CHARME 1999
- **Fixpoint Formulations of Compositional Invariance**
  - Cousot, P., Cousot, R.: Invariance proof methods and analysis techniques for parallel programs, Automatic Program Construction Techniques, 1984
  - Flanagan, C., Qadeer, S.: Thread-modular model checking, SPIN 2003
  - Namjoshi, K.S.: Symmetry and completeness in the analysis of parameterized systems, VMCAI 2007

Background Reading List – II

- **Neighborhood Symmetry**
  - Golubitsky, M., Stewart, I.: Nonlinear dynamics of networks: the groupoid formalism. Bull. AMS, 2006
  - Namjoshi K.S., Trefler, R.J.: Local symmetry and compositional verification, VMCAI 2012
- **Learning auxiliary assertions**
  - Giannakopoulou, D., Pasareanu, C.S., Barringer, H.: Component verification with automatically generated assumptions, ASE 2005
  - Cohen, A., Namjoshi, K.S.: Local proofs for global safety properties, CAV 2007
  - Gupta, A., Popeea, C., Rybalchenko, A.: Predicate abstraction and refinement for verifying multi-threaded programs. POPL 2011
- **Parameterized Compositional Verification**
  - Pnueli, A., Ruah, S., Zuck, L.: Automatic deductive verification with invisible invariants, TACAS 2001
  - Abdulla, P.A., Haziza, F., Holík, L.: All for the price of few, VMCAI 2013
  - Namjoshi, K.S., Trefler, R.J.: Parameterized Compositional Model Checking, TACAS 2016

There are many, many other excellent papers on compositional reasoning theory and practice!

Formal Acknowledgements

This work was supported, in part, by DARPA under agreement number FA8750-12-C-0166. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

This material is based upon work supported, in part, by the National Science Foundation under Grant No. (NSF CCF-1563393). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

