Model Checking for Software Architectures

– position paper –

Radu Mateescu

INRIA Rhône-Alpes / VASY
655, avenue de l'Europe
F-38330 Montbonnot Saint Martin
http://www.inrialpes.fr/vasy

Outline

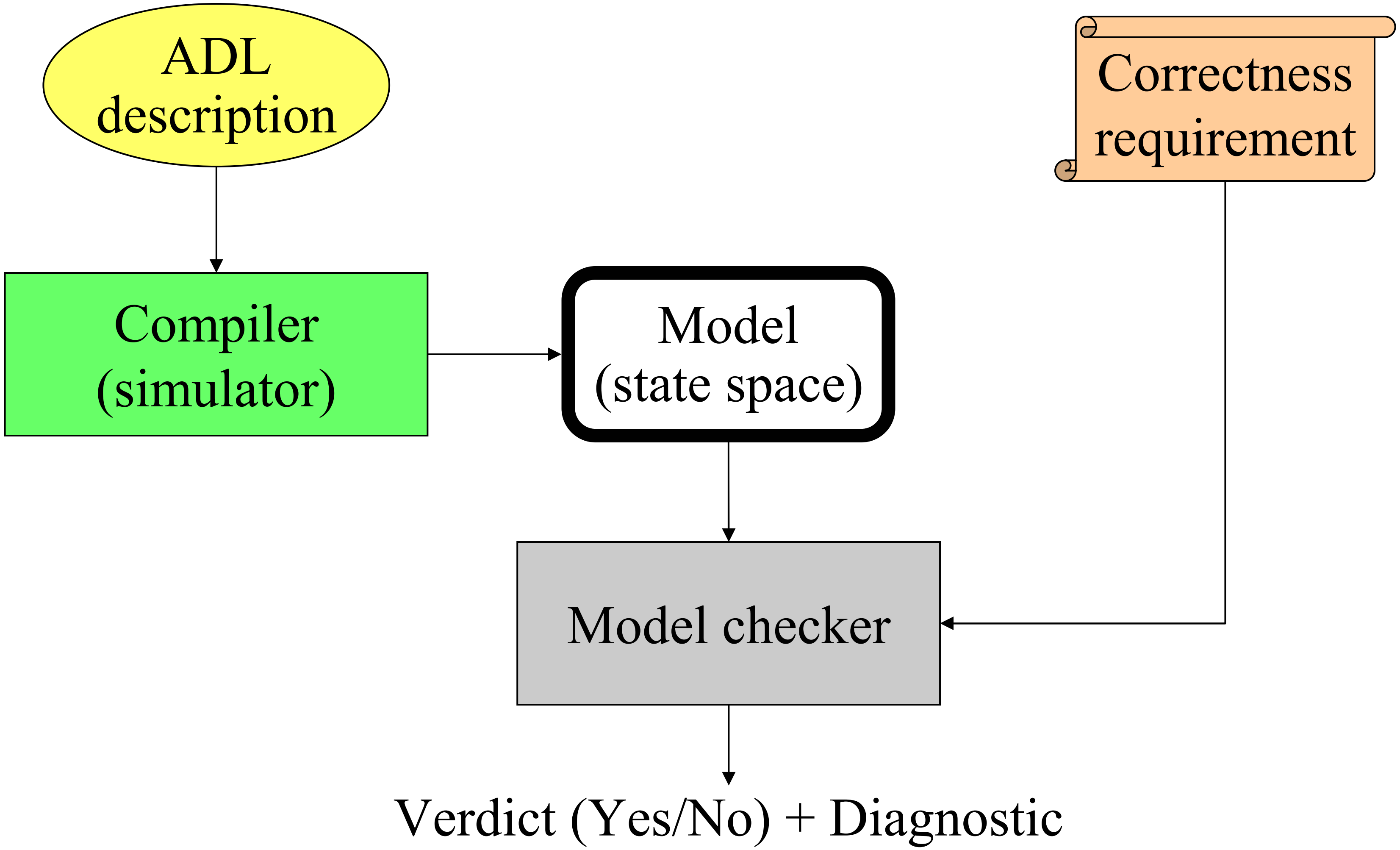
- Introduction
- Constructing state spaces
- Checking correctness requirements
- Handling large systems
- Conclusion

Introduction

- Software Architectures (SA) [Shaw-Garlan-96]
  – Gross organization of a system into elements
  – Protocols for communication and data access
  – Functionality of design elements
- Architecture Description Languages (ADLs)
  – Specify a SA formally
  – Analyze its structure and behaviour
- Complex, industrial-scale systems

  need computer-assisted analysis methodologies

Model checking methodology



Model checking for ADLs (1/2)

- WRIGHT [Allen-97]
  - CSP
  - FDR (deadlock detection, refinement)
- Dynamic WRIGHT [Allen-Douence-Garlan-98]
  - Reconfiguration + steady-state behaviour
  - Configuror process
- DARWIN [Magee-Dulay-Eisenbach-Kramer-95]
  - π-calculus + FSP
  - LTSA (LTL properties)

Model checking for ADLs (2/2)

- ΔPADL [Abate-Bernardo-02]
  - Finite replication + transparent routers
  - TwoTowers (equivalence and model checking)
- Publish-subscribe [Garlan-Khersonsky-Kim-03]

- π-Space [Chaudet-Oquendo-00]
- ArchWare ADL [Oquendo-Alloui-Cîmpan-Verjus-02]

{
- Dynamicity
- Mobility
- Evolution

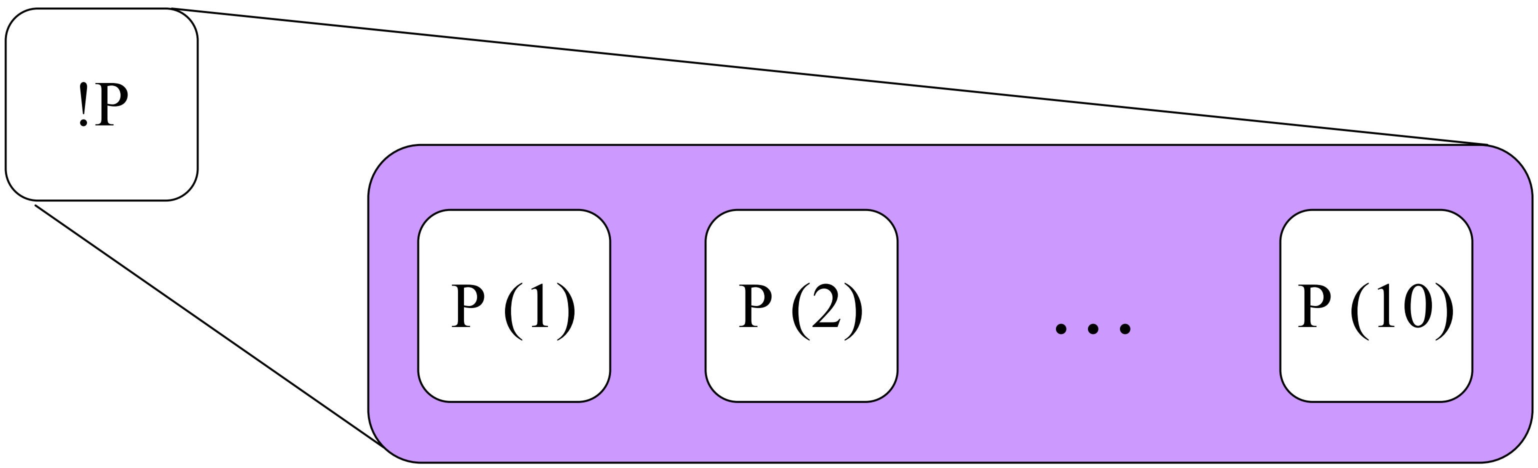
higher-order polyadic π-calculus

Constructing state spaces

- Develop from scratch an ADL simulator
  - ☺ Accurate w.r.t. the operational semantics
  - ☹ Complex to develop
- Translate the ADL into a target language
  - ☺ Easier to develop
  - ☺ Reuse the existing tools
- Target languages
  - LOTOS [ISO-88] and E-LOTOS [ISO-01]
  - CADP toolbox (http://www.inrialpes.fr/vasy/cadp)

Dynamic process creation

- Finite-state models
  → bound the number of SA element replicas

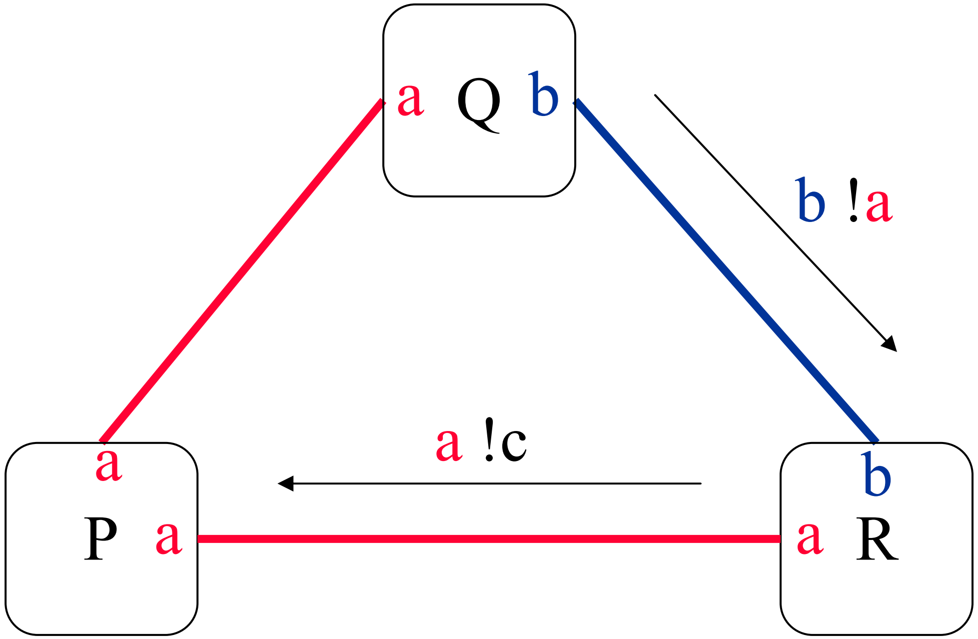


- LOTOS: **process** Q **:=** P **|||** Q **endproc**
- E-LOTOS: **par** n:1..10 **in** P (n) **endpar**

Channel mobility (1/2)

- Dynamic topology (π-calculus)

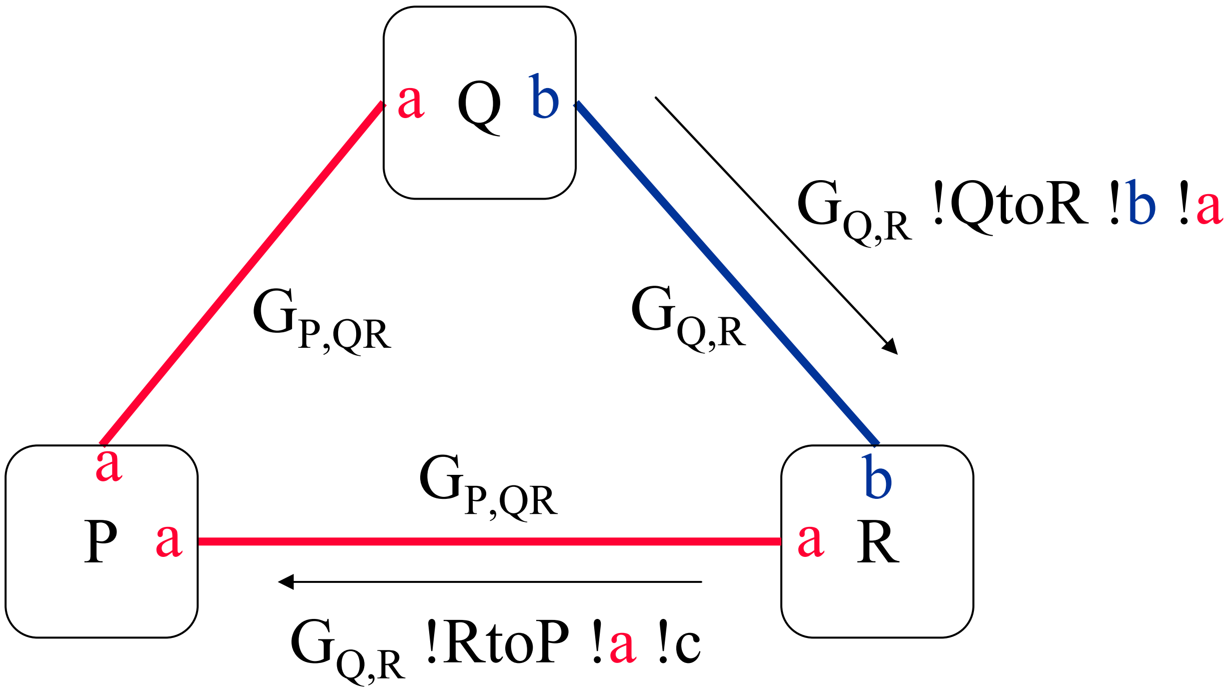
  P | Q | R



Channel mobility (2/2)

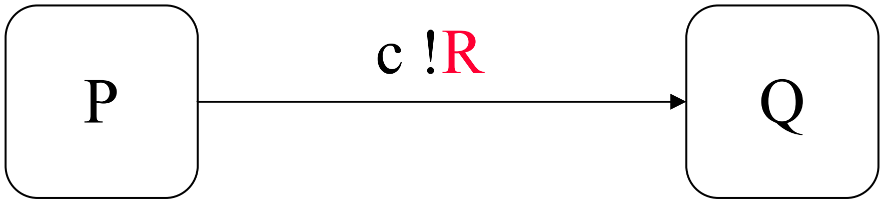
- Static topology (LOTOS, E-LOTOS)

  $P \;|[G_{P,QR}]|\; (Q \;|[G_{Q,R}]|\; R)$

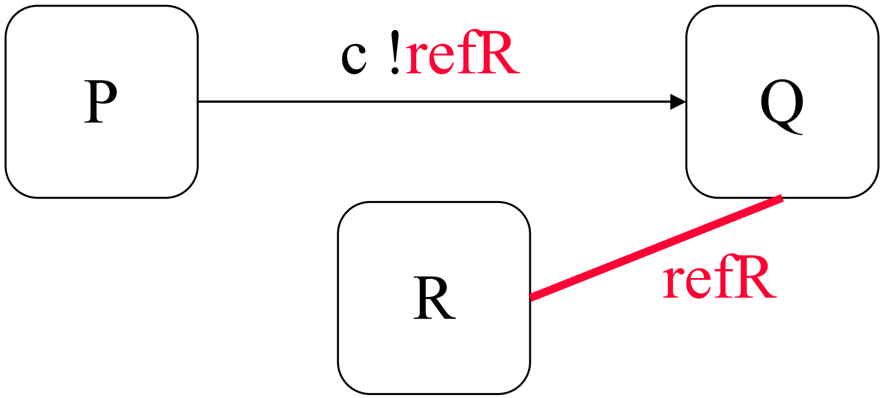


Higher-order process handling

- Evolvability → higher-order constructs



- Translation from higher-order to first-order π-calculus [Sangiorgi-01]



Checking correctness requirements

- Temporal logics + mu-calculi
  - Well-developed theory
  - Robust model checkers
- Optimisation of model checking algorithms
  - Memory-efficient algorithms (e.g., on traces)
- Improvement of user interfaces
  - Extension of TLs with higher-level constructs
  - Identification of domain-specific properties
  - Interpretation of diagnostics w.r.t. application

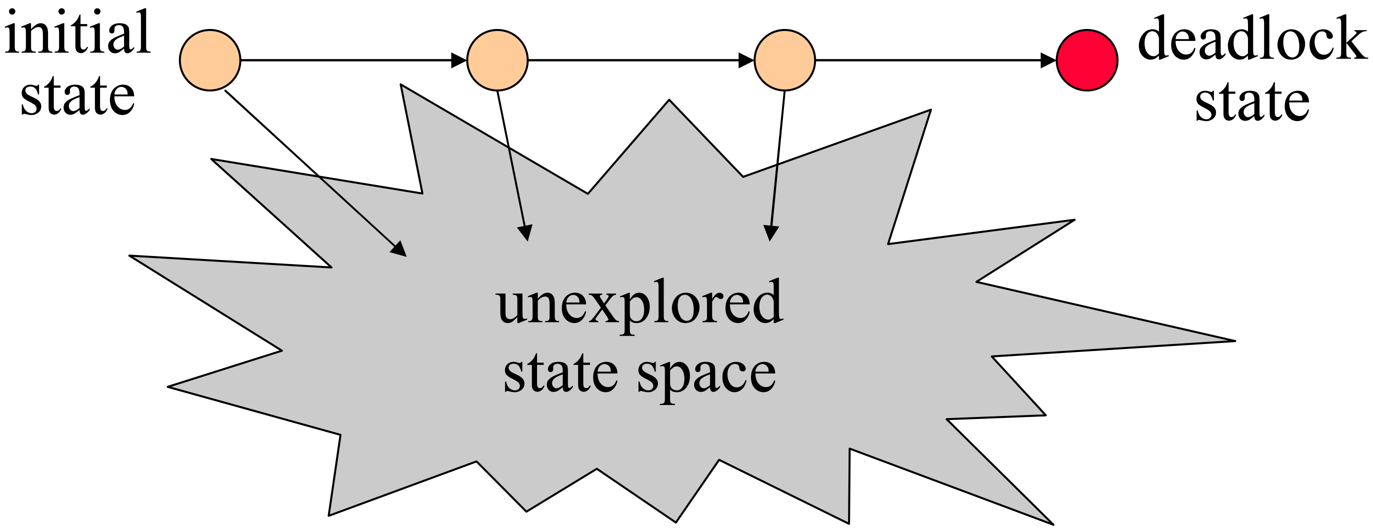
Handling large systems

- Industrial-scale systems
  - Many SA elements (parallel processes)
  - Complex data types

  → State explosion problem
- Techniques to combat state explosion
  - On-the-fly verification
  - Partial order reduction
  - Compositional verification
  - Sufficient locality conditions

On-the-fly verification

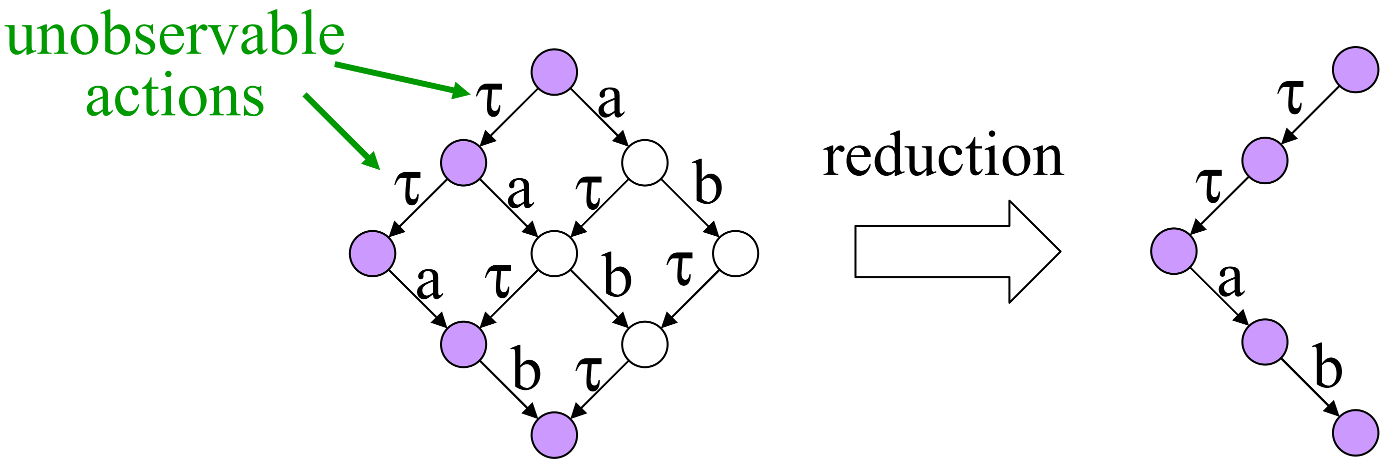
- Incremental construction of the state space



- OPEN/CAESAR environment [Garavel-98]
  - Generic API for state space exploration
  - Powerful libraries for graph manipulation

Partial order reduction

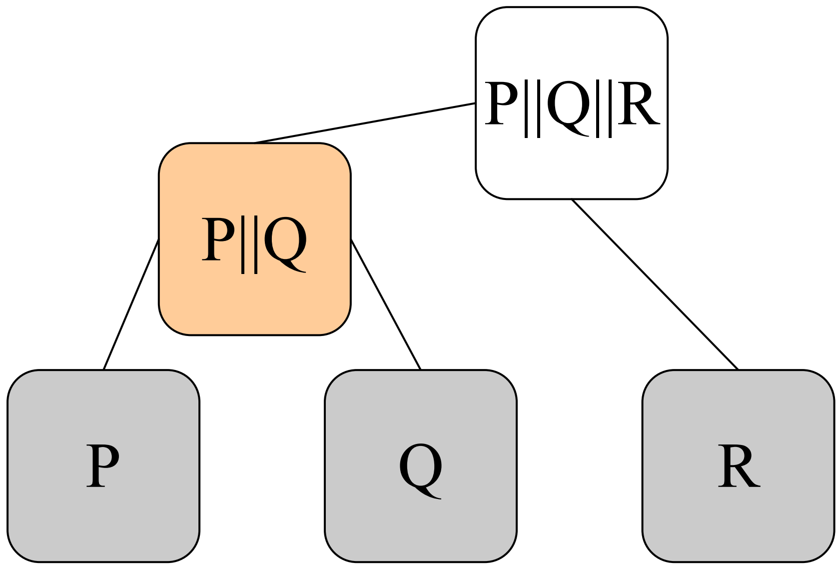
- Independent (parallel) components
  → redundant interleavings of actions



- Tau-confluence reduction [Groote-Pol-00]
  - Preserves branching equivalence

Compositional verification

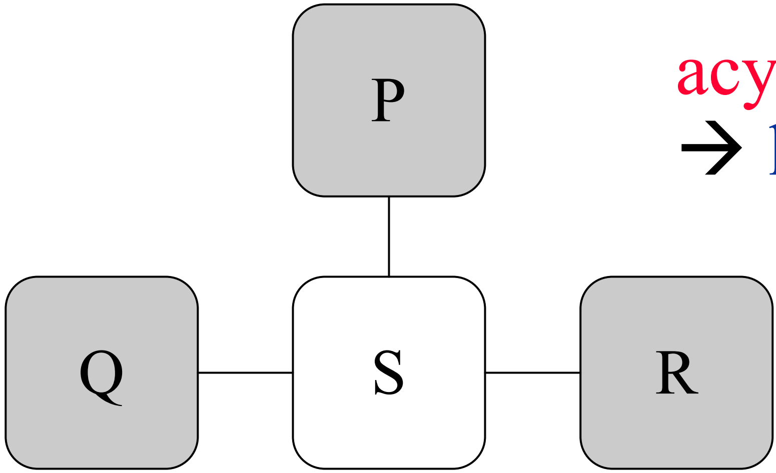
- Exploit the hierarchical structure of the SA
  - Construct the state spaces of SA elements
  - Reduce them modulo a suitable equivalence relation
  - Compose them to obtain the global state space



- SVL environment [Garavel-Lang-01]

Sufficient locality conditions

- Particular requirements (e.g., no deadlock)
  - Check requirements locally on SA elements
  - Ensure they hold on the whole SA



acyclic interconnection topology
→ local deadlock check

- Topology-related sufficient conditions
  [Bernardo-Ciancarini-Donatiello-01]

Conclusion

- Important aspects of model checking SA
  - Constructing state spaces
  - Checking requirements
  - Handling large systems

  } combine different techniques

- Claim

  effective way to proceed: reuse, enhance, and adapt the existing model checking technologies in the framework of software architectures