Verification of Adaptive Embedded Systems

Tobias Schüle

Reactive Systems Group
Department of Computer Science
University of Kaiserslautern

VerAS Workshop

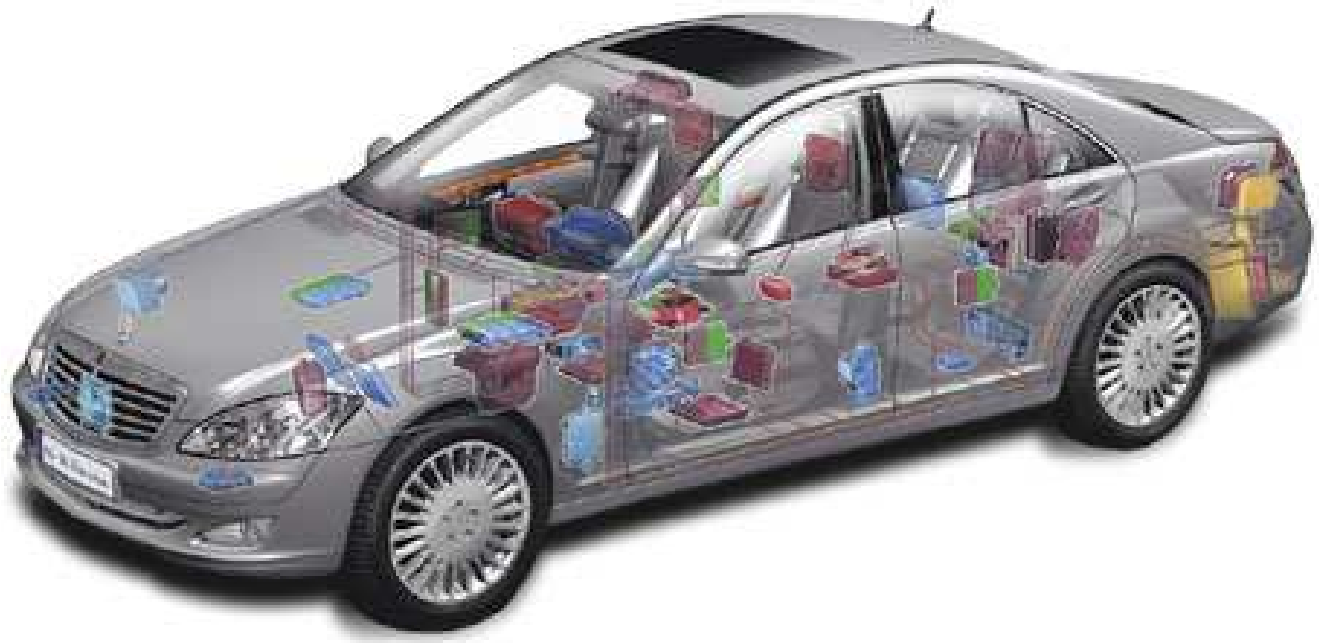
14 September 2007

Overview

Why Adaptation in Embedded Systems?

Improve quality and functionality

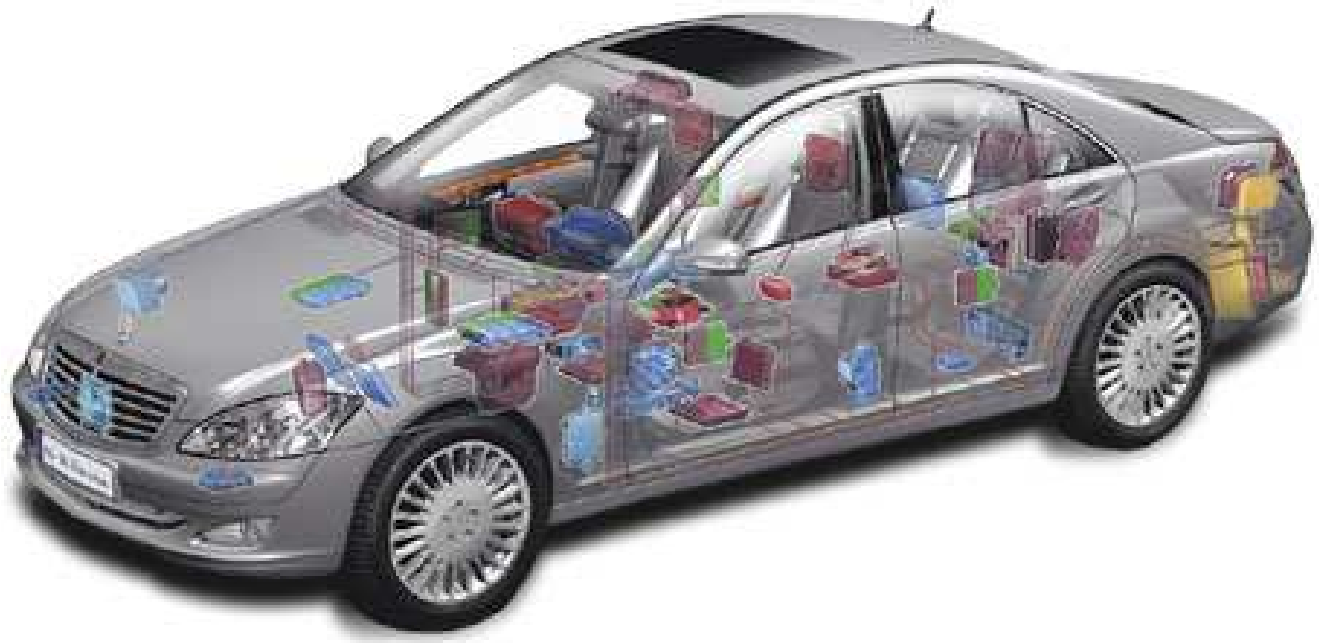
- changing environmental conditions (tunnel, failure of sensors)
- dependability and fault–tolerance (get safely to the next garage)
- customization (adapt to different drivers and their specific needs)



Why Adaptation in Embedded Systems?

Reduce costs

- depending on the situation not all parts are active at the same time
- share common parts of the system that are not used simultaneously
- dynamically adapt the components according to the current situation



Adaptation in Embedded Systems

Challenges regarding verification

- embedded systems are reactive real–time systems
  - verification of functional and temporal behavior
- hybrid systems (interacting analog and digital parts)
  - verification requires abstraction to discrete domains
- safety–critical systems (aviation, automotive industry)
  - legal aspects (“it wasn’t me who pushed the brakes”)

What’s so special about adaptation?

- adaptation has become increasingly complex part
- may trigger further adaptations in other components
- chain reaction of adaptations (up to 80% affected)
- can cause inconsistent and unstable configurations

Adaptation in Embedded Systems

Challenges regarding verification

- embedded systems are reactive real–time systems
  - verification of functional and temporal behavior
- hybrid systems (interacting analog and digital parts)
  - verification requires abstraction to discrete domains
- safety–critical systems (aviation, automotive industry)
  - legal aspects (“it wasn’t me who pushed the brakes”)

What’s so special about adaptation?

- adaptation has become increasingly complex part
- may trigger further adaptations in other components
- chain reaction of adaptations (up to 80% affected)
- can cause inconsistent and unstable configurations

Synchronous Languages

Programming model

- perfect synchrony: partitioning into micro and macro steps
- consumption of time explicitly specified by the programmer
- deterministic form of concurrency (processes run in lockstep)

Basic Quartz statements

pause;	consumption of time
nothing;	empty statement
$x = \tau$; / **next**$(x) = \tau$;	immediate/delayed assignment
emit x; / **emit next**(x);	immediate/delayed emission
while (φ) S	loop
if (φ) S_1 **else** S_2	conditional
$S_1\, S_2$	sequential execution
$S_1 \mid S_2$ / $S_1 \parallel S_2$ / $S_1 \mid\mid\mid S_2$	interl./sync./async. parallel execution
[**weak**] **abort** S **when** [**immediate**] (φ)	process abortion
[**weak**] **suspend** S **when** [**immediate**] (φ)	process suspension

Synchronous Languages

Programming model

- perfect synchrony: partitioning into micro and macro steps
- consumption of time explicitly specified by the programmer
- deterministic form of concurrency (processes run in lockstep)

Basic Quartz statements

pause;	consumption of time
nothing;	empty statement
$x = \tau$; / **next**$(x) = \tau$;	immediate/delayed assignment
emit x; / **emit next**(x);	immediate/delayed emission
while (φ) S	loop
if (φ) S_1 **else** S_2	conditional
$S_1\, S_2$	sequential execution
$S_1 \mid S_2$ / $S_1 \parallel S_2$ / $S_1 \mid\mid\mid S_2$	interl./sync./async. parallel execution
[**weak**] **abort** S **when** [**immediate**] (φ)	process abortion
[**weak**] **suspend** S **when** [**immediate**] (φ)	process suspension

Synchronous Languages

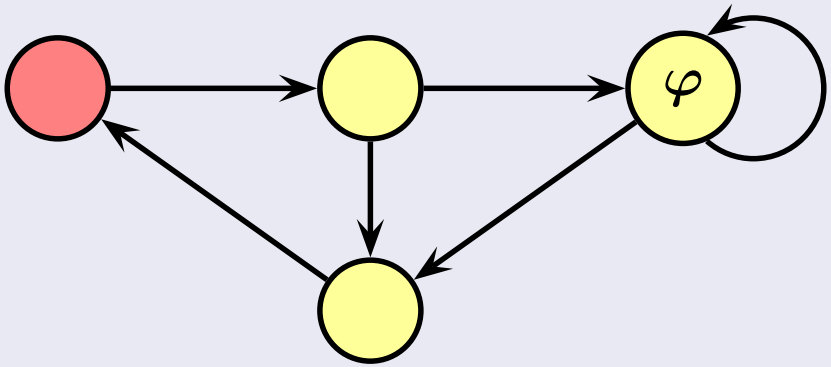
Example

module ABRO(**event** $a, b, r, \&o$) **implements** ABRO_Spec(a, b, r, o) {
  loop {
    abort {
      await(a); || **await**(b);
      emit o;
    } **when**(r);
  }
}

spec ABRO_Spec(**event** a, b, r, o) {
  S1 : A G ($o \rightarrow a \vee b$);
}

Model Checking

System (Kripke structure)



- directed graph
- nodes: configurations
- edges: adaptations
- paths: adaptation sequences

Specification (temporal logic, e.g. CTL or LTL)

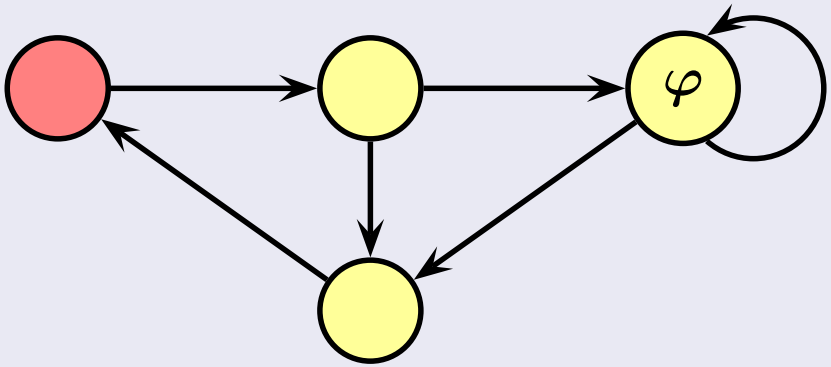
- EF φ "there exists a path such that φ eventually holds" (least FP)
- AG φ "φ invariantly holds on all paths" (greatest FP)

Model checking

Does the system satisfy the specification? $\Rightarrow$ Fixpoint computation

Model Checking

System (Kripke structure)



- directed graph
- nodes: configurations
- edges: adaptations
- paths: adaptation sequences

Specification (temporal logic, e.g. CTL or LTL)

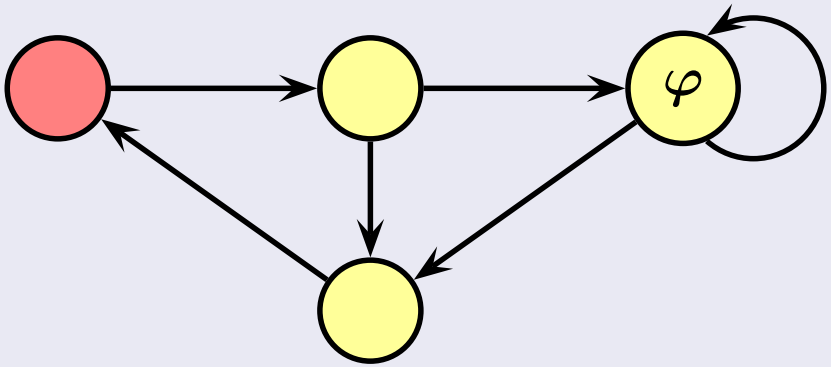
- EF φ "there exists a path such that φ eventually holds" (least FP)
- AG φ "φ invariantly holds on all paths" (greatest FP)

Model checking

Does the system satisfy the specification? $\Rightarrow$ Fixpoint computation

Model Checking

System (Kripke structure)



- directed graph
- nodes: configurations
- edges: adaptations
- paths: adaptation sequences

Specification (temporal logic, e.g. CTL or LTL)

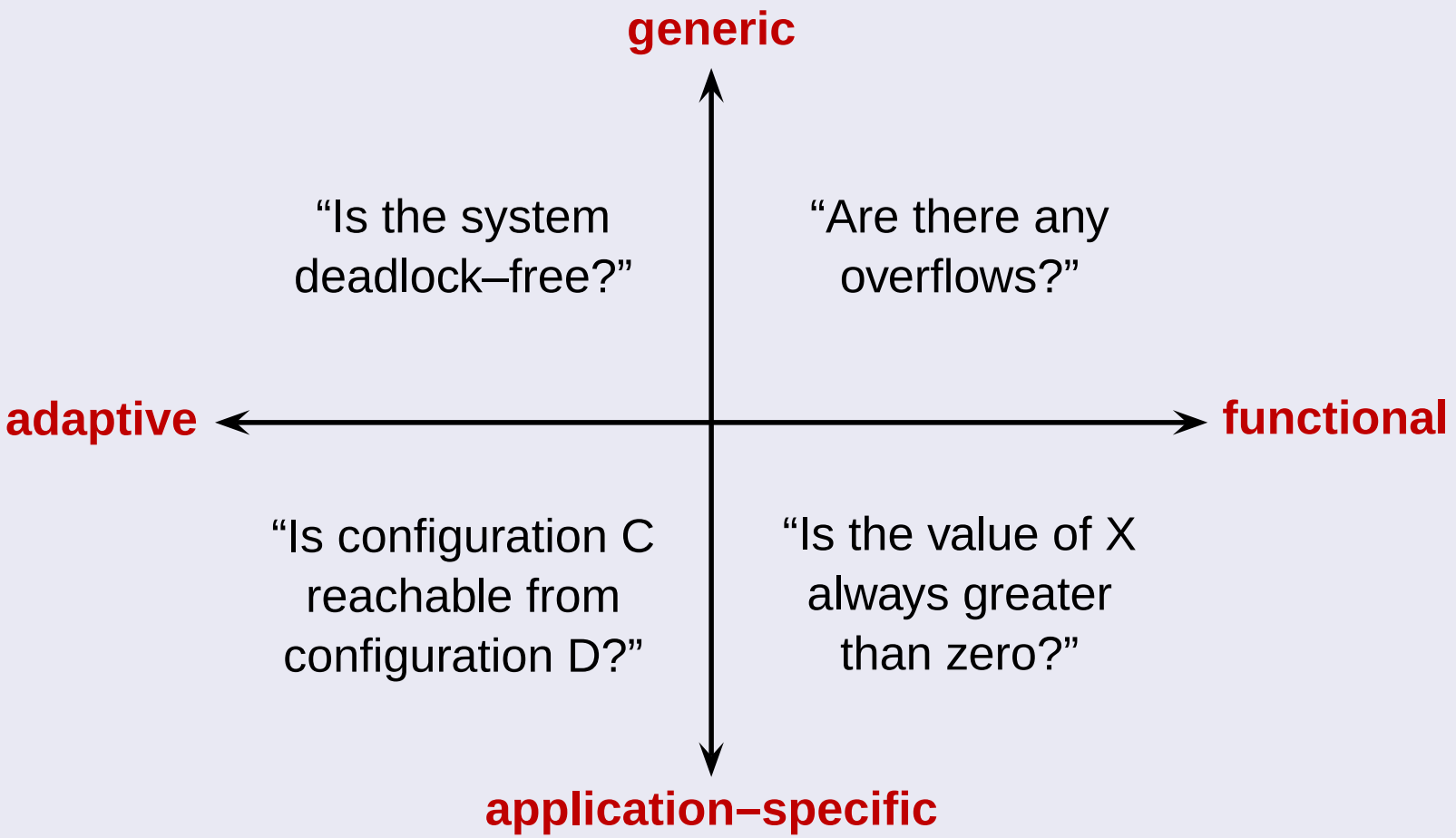
- EF φ "there exists a path such that φ eventually holds" (least FP)
- AG φ "φ invariantly holds on all paths" (greatest FP)

Model checking

Does the system satisfy the specification? $\Rightarrow$ Fixpoint computation

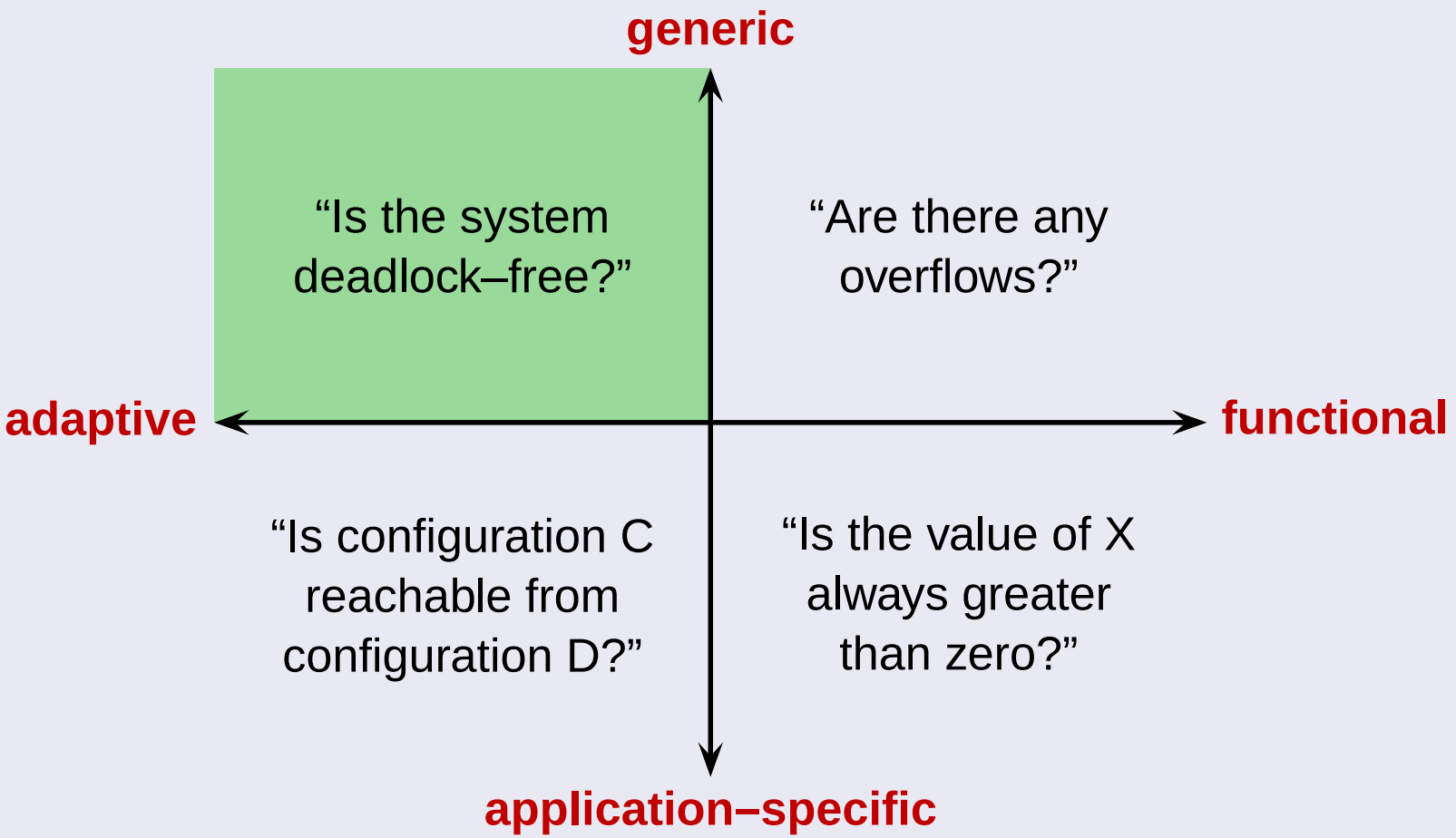
Properties of Adaptive Systems

Classification



Properties of Adaptive Systems

Classification



Properties of Adaptive Systems

Generic properties of adaptation behavior (CTL)

1. No module gets stuck in the default configuration ‘off’:

$$\mathrm{AG}(c = \mathit{off} \rightarrow \mathrm{EF}\, c \neq \mathit{off})$$

2. Every module can reach all configurations at all times:

$$\mathrm{AG}(\bigwedge_{i=1}^{n} \mathrm{EF}\, c = \mathit{config}_i)$$

3. No inconsistent states can be reached:

$$\mathrm{AG}(\bigvee_{i=1}^{n} c = \mathit{config}_i)$$

4. No configuration is always only transient:

$$\bigwedge_{i=1}^{n} \mathrm{EFEG}\, c = \mathit{config}_i$$

Stability of Adaptation Behavior

Finite vs. infinite adaptation sequences

- no central authority $\Rightarrow$ chain reaction of adaptations
- finite adaptation sequences usually intended
- infinite adaptation sequences lead to unstable system
- verify that the outputs stabilize if the inputs do not change

Stability checking by LTL model checking

- $\varphi_{\text{in}} \Leftrightarrow$ inputs stable for one unit of time
- $\varphi_{\text{out}} \Leftrightarrow$ outputs stable for one unit of time
- $\mathsf{AG}(\mathsf{G}\varphi_{\text{in}} \rightarrow \mathsf{FG}\varphi_{\text{out}})$
- requires computation of interdependent fixpoints

Stability of Adaptation Behavior

Finite vs. infinite adaptation sequences

- no central authority $\Rightarrow$ chain reaction of adaptations
- finite adaptation sequences usually intended
- infinite adaptation sequences lead to unstable system
- verify that the outputs stabilize if the inputs do not change

Stability checking by LTL model checking

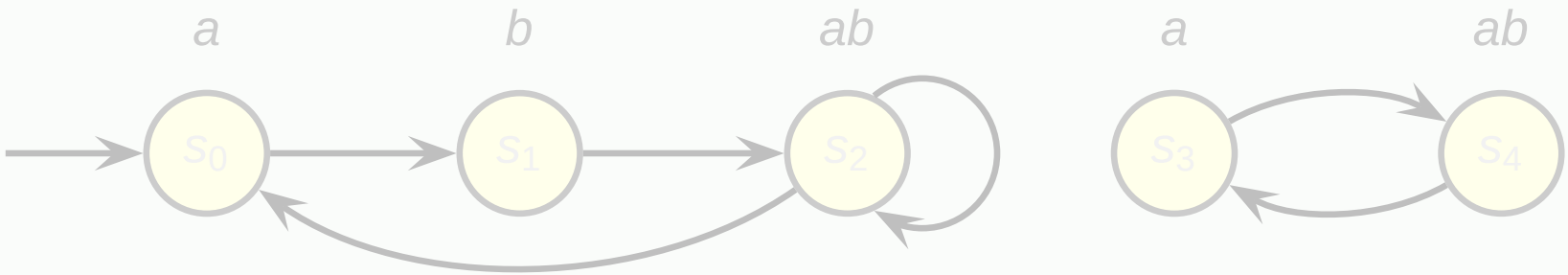
- $\varphi_{\text{in}} \Leftrightarrow$ inputs stable for one unit of time
- $\varphi_{\text{out}} \Leftrightarrow$ outputs stable for one unit of time
- $\mathsf{AG}(\mathsf{G}\varphi_{\text{in}} \rightarrow \mathsf{FG}\varphi_{\text{out}})$
- requires computation of interdependent fixpoints

Stability of Adaptation Behavior

Specialized verification procedures for stability

- a path is steady iff the inputs/outputs do not change from time $t \geq 0$
- a state is stable iff for all paths input steadiness $\Rightarrow$ output steadiness
- a Kripke structure is stable iff all reachable states are stable

Example



Theorem

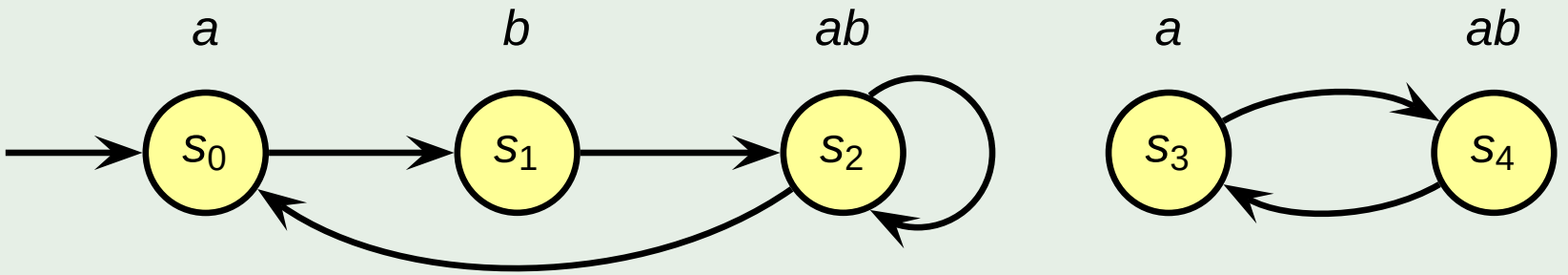
Stability can be checked by evaluating two alternation–free formulas.

Stability of Adaptation Behavior

Specialized verification procedures for stability

- a path is steady iff the inputs/outputs do not change from time $t \geq 0$
- a state is stable iff for all paths input steadiness $\Rightarrow$ output steadiness
- a Kripke structure is stable iff all reachable states are stable

Example



Theorem

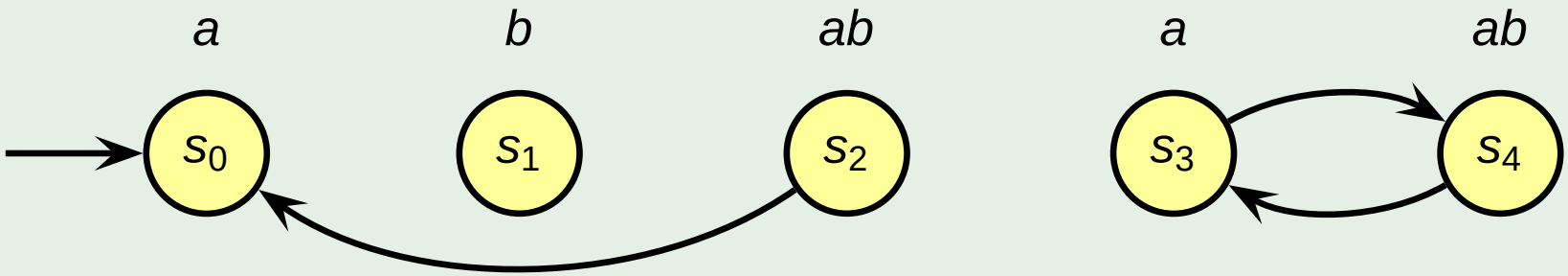
Stability can be checked by evaluating two alternation–free formulas.

Stability of Adaptation Behavior

Specialized verification procedures for stability

- a path is steady iff the inputs/outputs do not change from time $t \geq 0$
- a state is stable iff for all paths input steadiness $\Rightarrow$ output steadiness
- a Kripke structure is stable iff all reachable states are stable

Example



Theorem

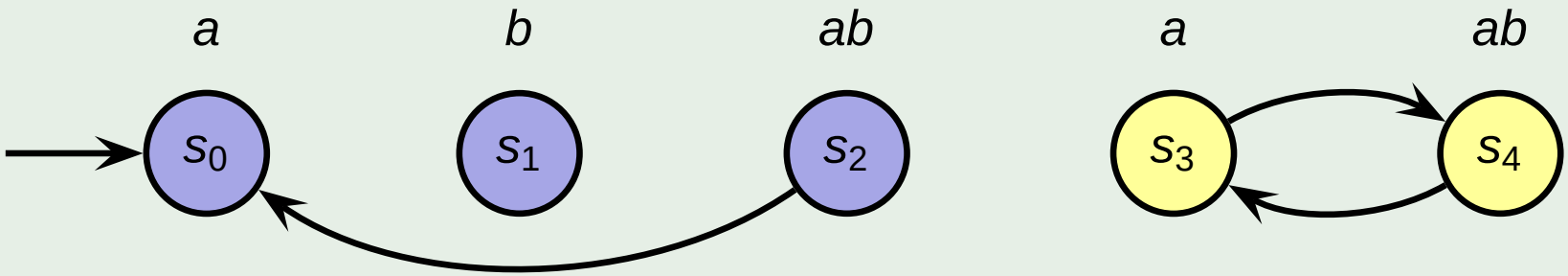
Stability can be checked by evaluating two alternation–free formulas.

Stability of Adaptation Behavior

Specialized verification procedures for stability

- a path is steady iff the inputs/outputs do not change from time $t \geq 0$
- a state is stable iff for all paths input steadiness $\Rightarrow$ output steadiness
- a Kripke structure is stable iff all reachable states are stable

Example



Theorem

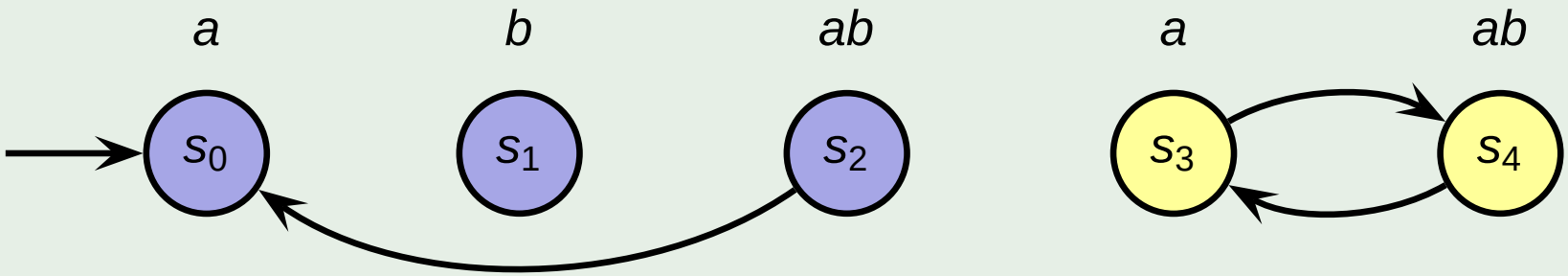
Stability can be checked by evaluating two alternation–free formulas.

Stability of Adaptation Behavior

Specialized verification procedures for stability

- a path is steady iff the inputs/outputs do not change from time $t \geq 0$
- a state is stable iff for all paths input steadiness $\Rightarrow$ output steadiness
- a Kripke structure is stable iff all reachable states are stable

Example



Theorem

Stability can be checked by evaluating two alternation–free formulas.

Experimental Results

Traction control system

- Fraunhofer Institute for Experimental Software Engineering (IESE)
- Software Technology Group, University of Kaiserslautern
- 28 modules, 70 configurations, $>$ 2000 lines of Quartz code
- 151 generic specifications concerning adaptation behavior

Generic specifications of adaptation behavior

event variables	min. [s]	avg. [s]	max. [s]	total [s]
no	< 1	5	222	782
yes	< 1	2	143	298

Stability of adaptation behavior

LTL model checking: $>$ 1h

specialized procedure: 1723s

Experimental Results

Traction control system

- Fraunhofer Institute for Experimental Software Engineering (IESE)
- Software Technology Group, University of Kaiserslautern
- 28 modules, 70 configurations, $>$ 2000 lines of Quartz code
- 151 generic specifications concerning adaptation behavior

Generic specifications of adaptation behavior

event variables	min. [s]	avg. [s]	max. [s]	total [s]
no	< 1	5	222	782
yes	< 1	2	143	298

Stability of adaptation behavior

LTL model checking: $>$ 1h specialized procedure: 1723s

Experimental Results

Traction control system

- Fraunhofer Institute for Experimental Software Engineering (IESE)
- Software Technology Group, University of Kaiserslautern
- 28 modules, 70 configurations, $>$ 2000 lines of Quartz code
- 151 generic specifications concerning adaptation behavior

Generic specifications of adaptation behavior

event variables	min. [s]	avg. [s]	max. [s]	total [s]
no	< 1	5	222	782
yes	< 1	2	143	298

Stability of adaptation behavior

LTL model checking: $>$ 1h specialized procedure: 1723s

Summary and Conclusion

Adaptive embedded systems

- cost–efficient way to increase dependability of embedded systems
- verification particularly important in safety–critical areas, e.g. cars
- translate high–level model via formal rep. to synchronous program
- standard and specialized algorithms for verifying adaptation behavior

Directions for future work

- support for hybrid systems (real numbers)
- timing analysis (worst case execution time)
- predicate abstraction and refinement
- software synthesis for code generation

Summary and Conclusion

Adaptive embedded systems

- cost–efficient way to increase dependability of embedded systems
- verification particularly important in safety–critical areas, e.g. cars
- translate high–level model via formal rep. to synchronous program
- standard and specialized algorithms for verifying adaptation behavior

Directions for future work

- support for hybrid systems (real numbers)
- timing analysis (worst case execution time)
- predicate abstraction and refinement
- software synthesis for code generation