Comparison of methods for supervisory control and submodule construction

Gregor v. Bochmann and Bassel Daou

School of Information Technology and Engineering (SITE)
University of Ottawa

ACSD conference, Hamilton, June 2004

Abstract

Over the last 25 years, methods for supervisory control of discrete event systems and methods for submodule construction based on state machine specifications have been developed quite independently by different research communities. The purpose of this paper is to give a summary of the results in these two areas and to point out the many similarities and certain differences between the approaches taken by these two communities. The basic problem, in both cases, is to find the behavior of a single submodule X such that combined with a given submodule C, this composition exhibits a behavior that conforms to a given specification S. In the case of supervisory control, the submodule C is an existing system that is to be controlled by the controller X in such a manner that a behavior compatible with S is obtained. We discuss the main issues that must be addressed for solving this problem, review certain conditions for the existence of a solution, and present the major solution algorithms. We also discuss the different treatment of allowed and required behavior, and the difficulties that arise in the context of different communication paradigms (for instance, distinguishing controllability, observability, input/output, synchronous and asynchronous communication) and different specification formalisms.

Equation solving: Integer division

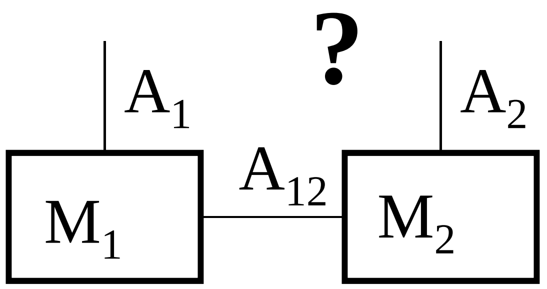
- Multiplication: $R_1 * R_2 = ?$
- Equation solving: $R_1 * X = R_3$
  - What is the value of X ?
- Solution: definition of the division operation
  - Written " $X = R_3 / R_1$ "
  - What does it mean ?
    - X = biggest Y such that $R_1 * X \leq R_3$
    - Note: in many cases, there is no *exact* solution, that is, there is no X such that $R_1 * X = R_3$
      - For instance: $7 / 3 = 2$, and $3 * 2 = 6 \leq 7$

Submodule construction

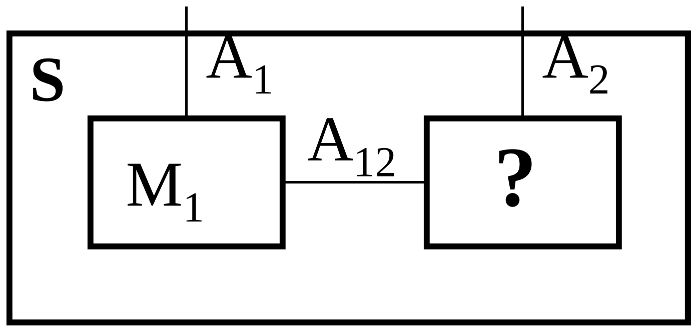
Multiplication → Machine composition

Division → Submodule construction *("equation solving")*

Example:

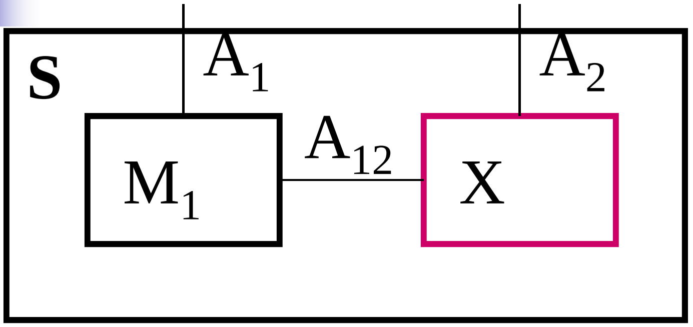


Multiplication



Division

Equation solving for machines

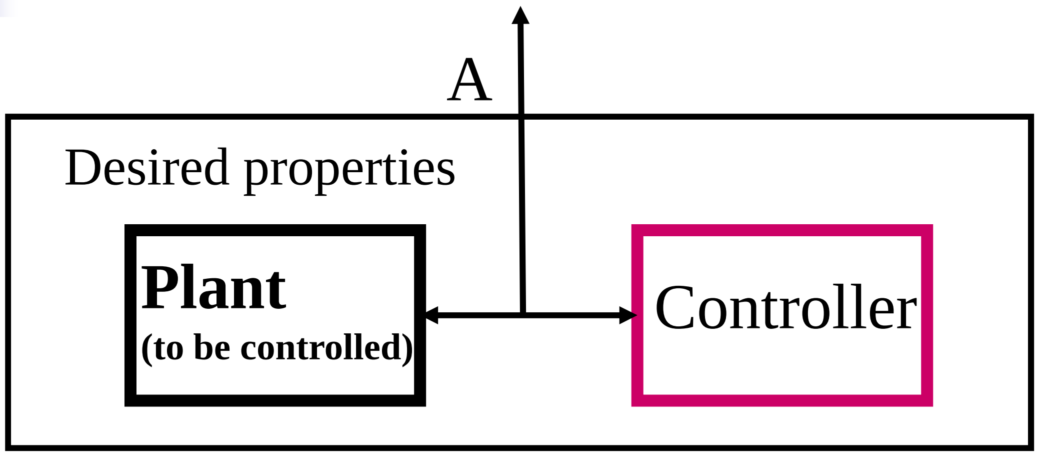


Given machine $\mathbf{M_1}$ and specification **S** for the behavior of the composition of $\mathbf{M_1}$ with **X**, find a behavior of machine **X** such that

$$\textbf{hide } \mathbf{A_{12}} \textbf{ in } (\mathbf{M_1} \infty \mathbf{X}) \leq \mathbf{S}$$

Meaning of ≤ : set inclusion of possible execution sequences ("traces", i.e. sequences of interactions), also called *trace inclusion*

Controller design



Applications in process control, robotics, etc.

- In the context of so-called “Discrete event systems” *[Ramage-Wonham, 1989]*
- Distinction between non-controllable and controllable interactions *(like input/output)*

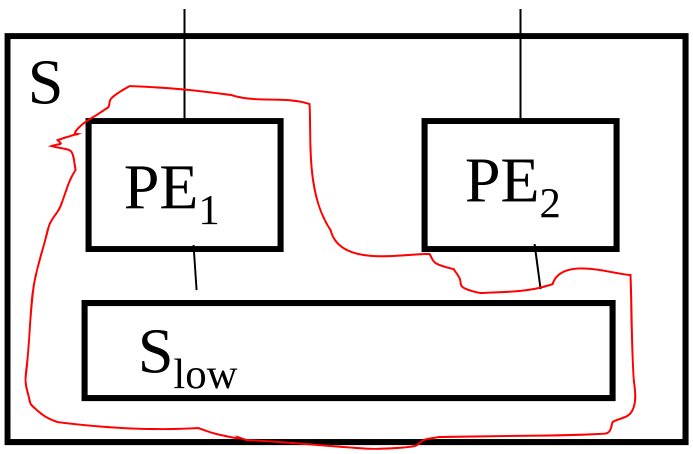
Overview

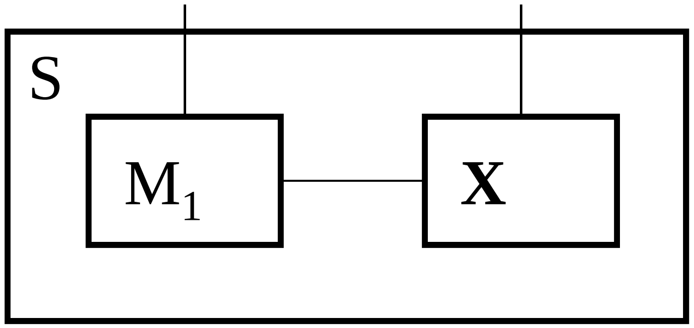
- Introduction
- Application areas
- Overview of differences between CD and SC
- SC solution formulas for LTS trace semantics
- Other differences: progress requirements etc.
- IO Automata and partial specifications
- Other conformance relations and issues
- Conclusions

Application Areas

- Controller design for discrete event systems
- Communication protocols
  - Protocol design (Merlin-Bochmann, 1980)
  - Design of communication gateways
- Component reuse, e.g. in software engineering
- Embedded testing

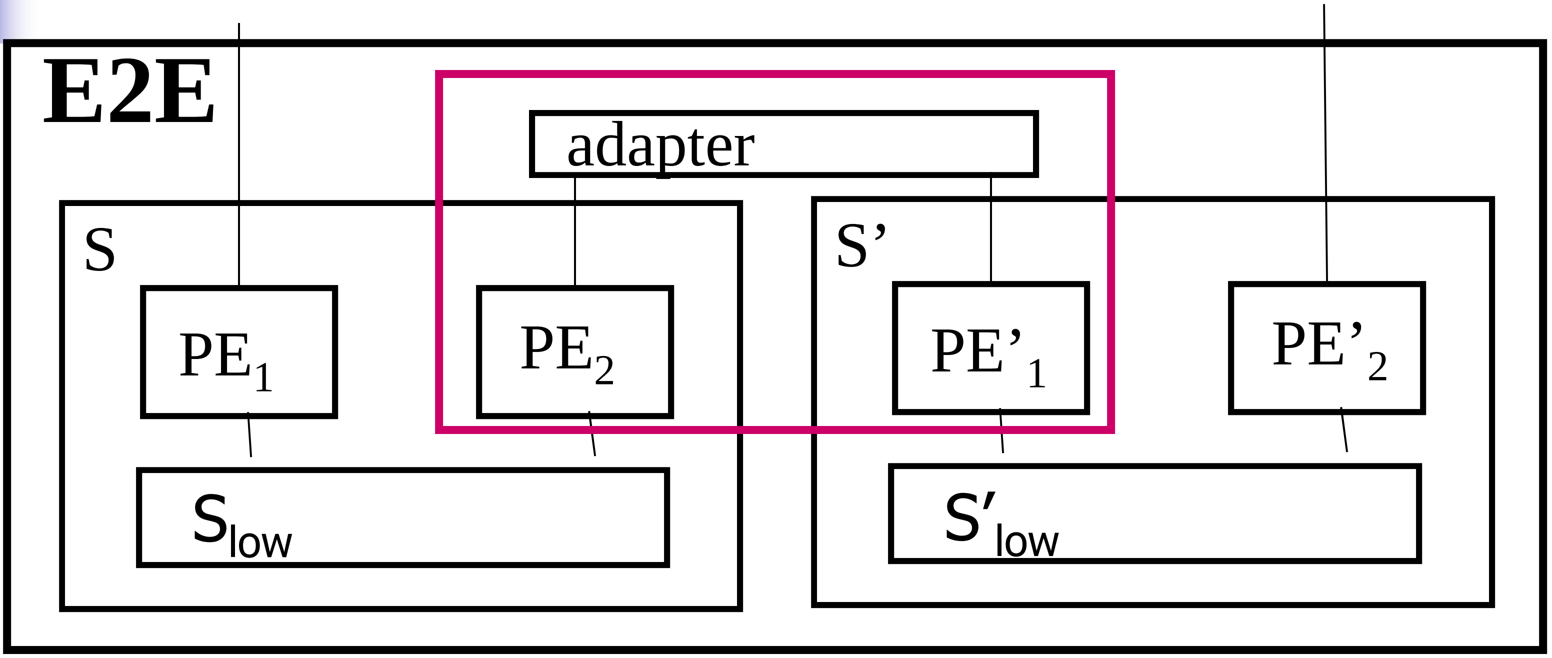
Communication protocol design





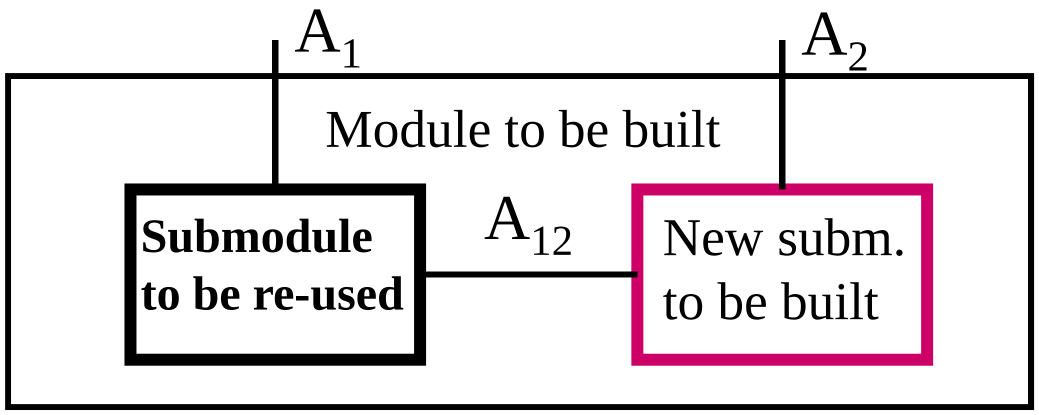
- Protocol entities PE_1 and PE_2 use the underlying service S_{low} and provide the service **S** to the users of the protocol
  - PE_1 and S_{low} are given
  - PE_2 = **X** is to be found
  - M_1 corresponds to ($PE_1 \infty S_{low}$)

Communication gateways



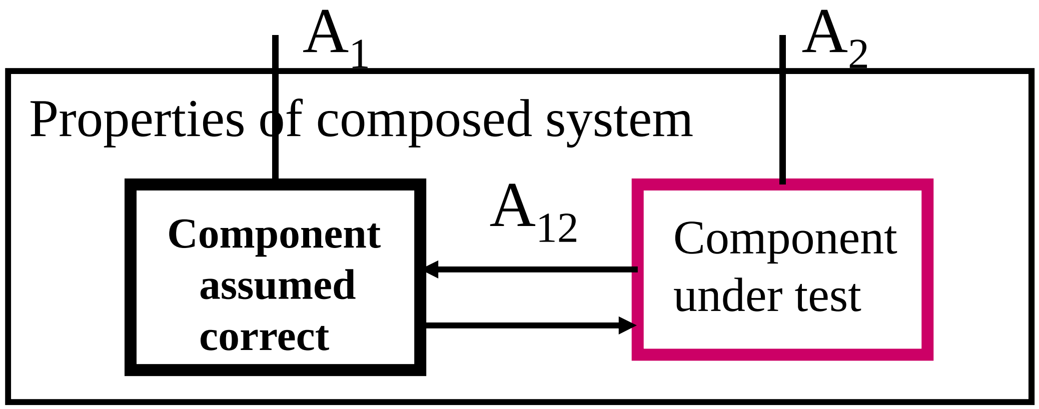
- Given
  - desired end-to-end communication service *E2E*
  - Protocols in the two networks (different)
- To be found: gateway behavior (shown by red box)

Component reuse



- A given submodule does not completely correspond to the specification of the system to be built
- An additional submodule to be built *(and designed throught equation solving)* makes up the “difference”

Embedded testing



- If internal interactions (i.e. A_{12}) are not visible, only the properties of the composed system can be observed
- The most general behavior of the SUT that leads to conforming behavior for the composed system, is the solution of submodule construction.
  - This behavior is often more general than the specification for the SUT; the difference can not be observed.

Overview

- Introduction
- Application areas
- Overview of differences between CD and SC
- SC solution formulas for LTS trace semantics
- Other differences: progress requirements etc.
- IO Automata and partial specifications
- Other conformance relations and issues
- Conclusions

Overview of differences

[M-B-1980] : SC for trace semantics

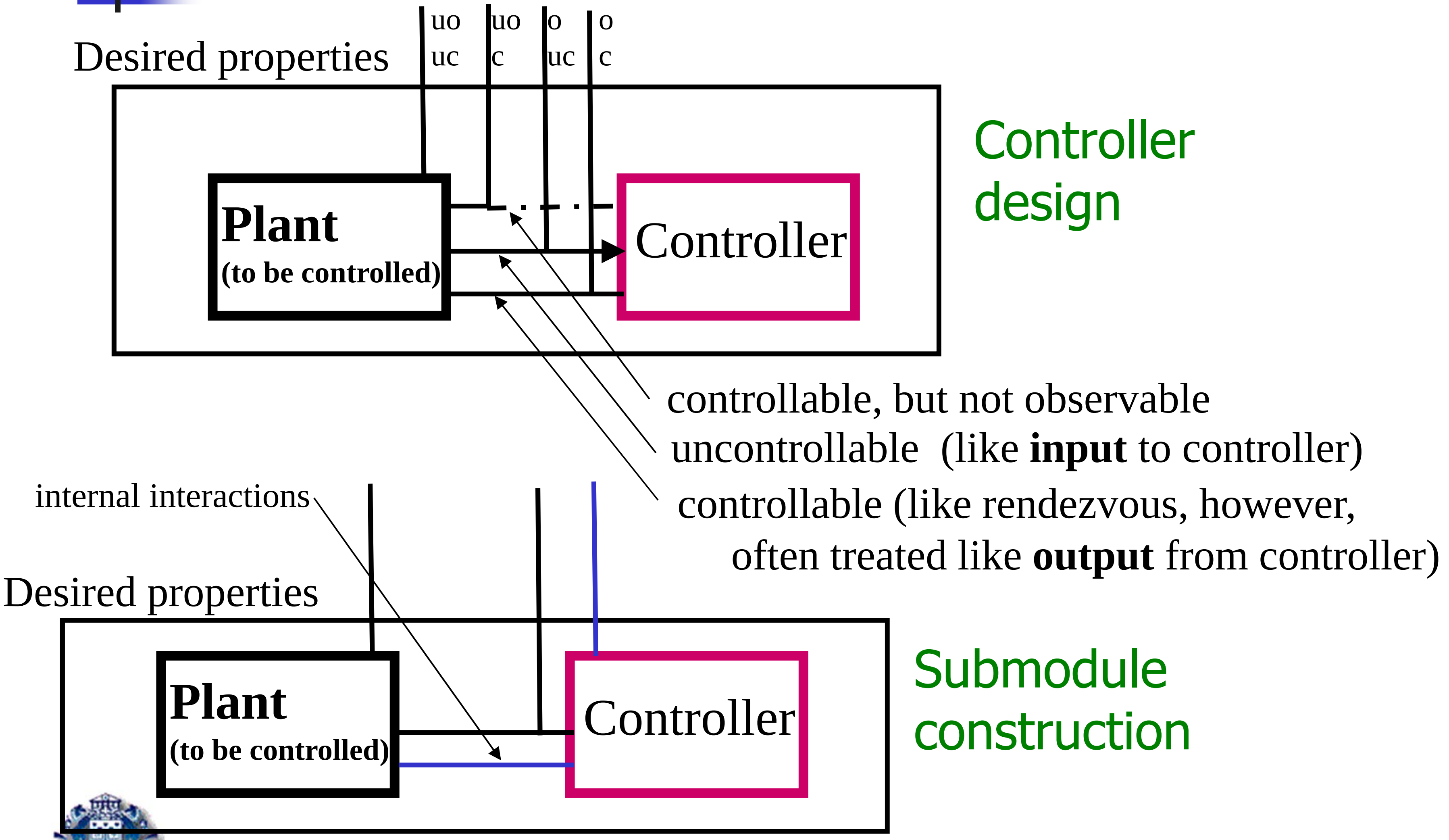
- rendezvous interactions
- partial observability by controller
- internal interactions (not visible at service level)
- construction algorithm for regular languages

[R-W-1989] + follow-on papers: CD for trace semantics

Like above, except the following

- no internal interactions
- distinction of controllability of interactions
- pruning algorithm to avoid deadlocks

Architectural overview


Desired properties
uo uc
uo c
o uc
o c
Plant
(to be controlled)
Controller
controllable, but not observable
uncontrollable (like **input** to controller)
controllable (like rendezvous, however, often treated like **output** from controller)
internal interactions
Desired properties
Plant
(to be controlled)
Controller

Controller design

Submodule construction

Modeling controller interactions

Questions: Can the different types of controller interactions be modelled with rendezvous interactions ?

- unobservable – uncontrollable
  - controller is not involved
- observable – controllable
  - normal rendezvous interaction (if the controller state has no corresponding transition, the interaction is not possible)
- observable – uncontrollable
  - "input" to controller: each state of the controller must have a corresponding transition
- unobservable – controllable
  - If a state of the controller has a corresponding transition, it must be a self-loop (controller goes back to the same state, no visibility)

Overview

- Introduction
- Application areas
- Overview of differences between CD and SC
- SC solution formulas for LTS trace semantics
- Other differences: progress requirements etc.
- IO Automata and partial specifications
- Other conformance relations and issues
- Conclusions

The SC problem and its solution

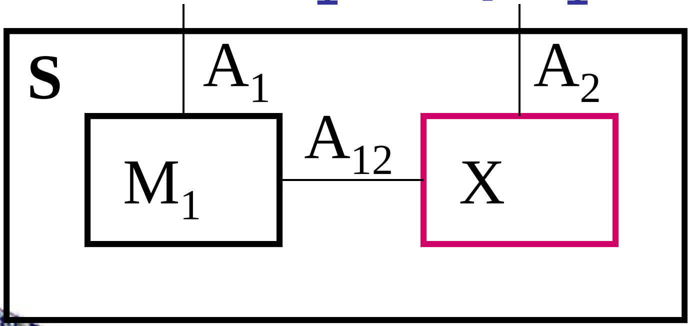
Problem: Find largest set X (of execution sequences) over the alphabet $(A_2 \cup A_{12})$ such that

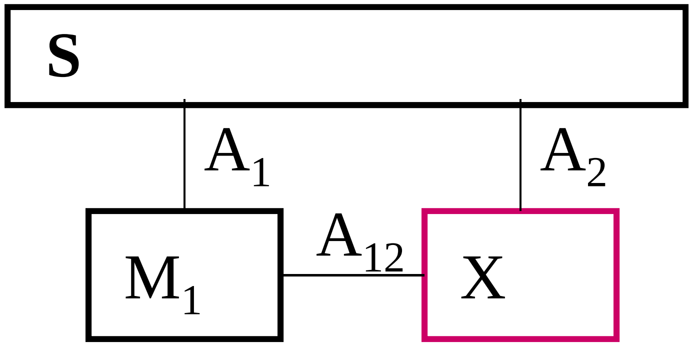
$$\text{hide } A_{12} \text{ in } (M_1 \infty X) \quad \leq \quad S$$

Solution: $X = (A_2 \cup A_{12})^* \quad \backslash$ *(minus)*

any sequence that could lead to an observable execution sequence not in S , i.e.

$$\text{hide } A_1 \text{ in } (M_1 \infty ((A_1 \cup A_2)^* \backslash S))$$



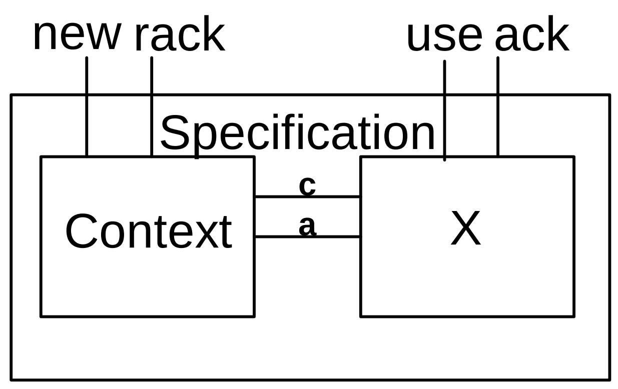


A comment about the largest solution

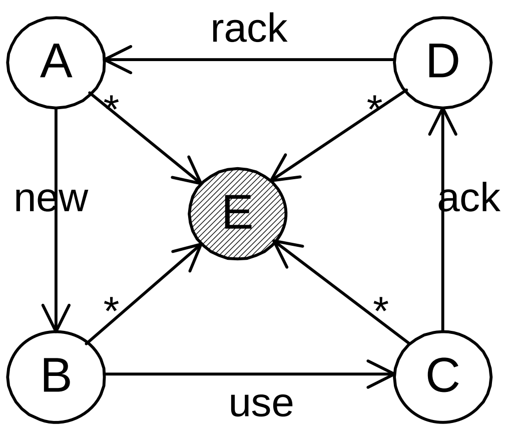
- Since all execution sequences of X must go in interaction with M_1 and S, we may replace $(A_2 U A_{12})^*$ (the chaos for X) by hide A_1 in $(M_1 \infty S)$
  - The obtained "reduced" solution is as good as the largest one, since the sequences in the difference between the two will block in the interaction with M_1

An example: one-place queue

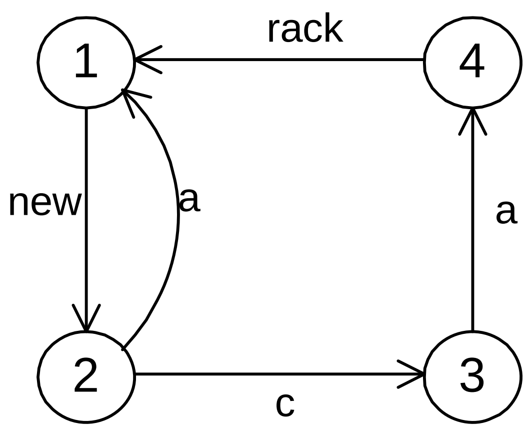
Architecture



Specification

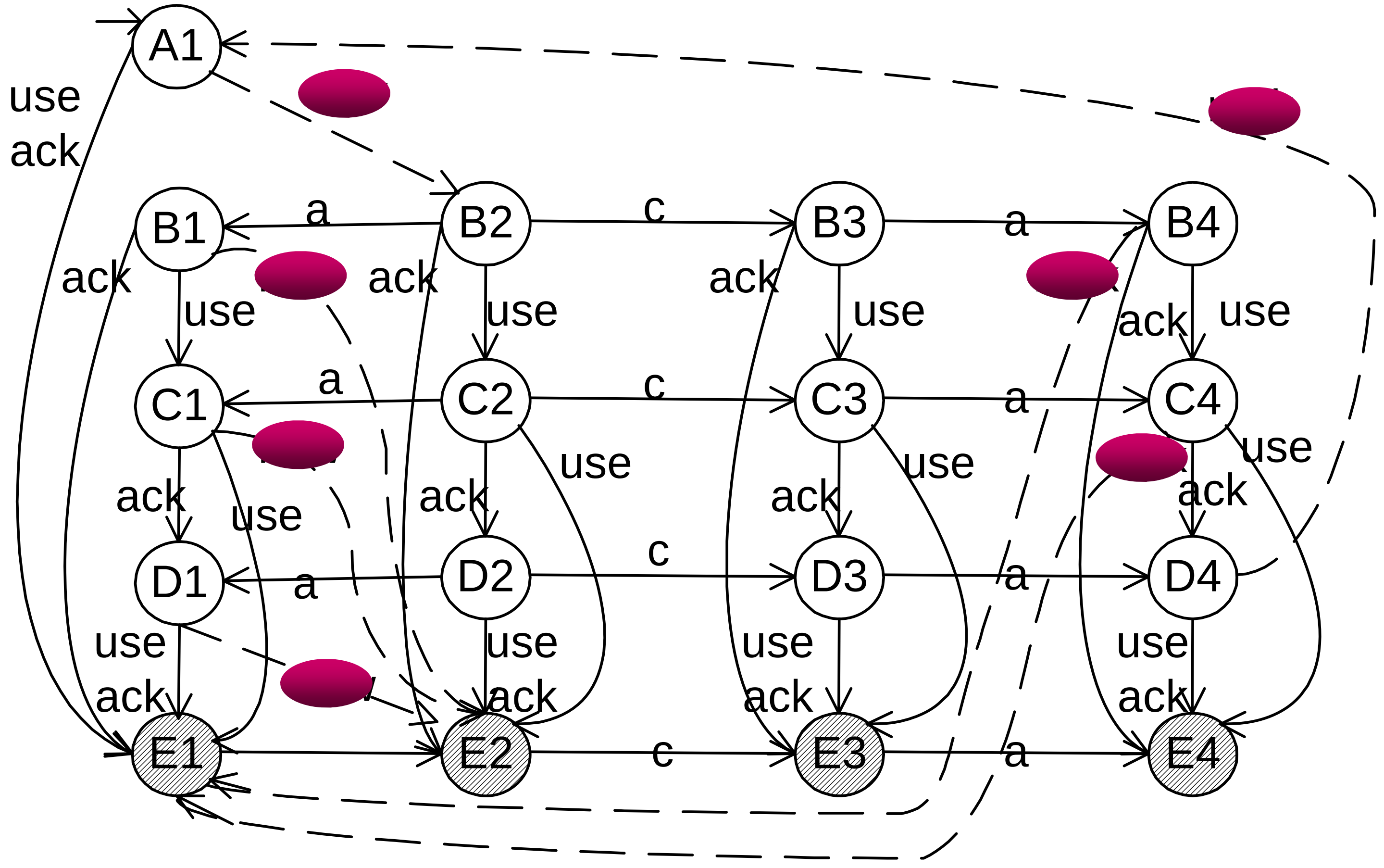


Context



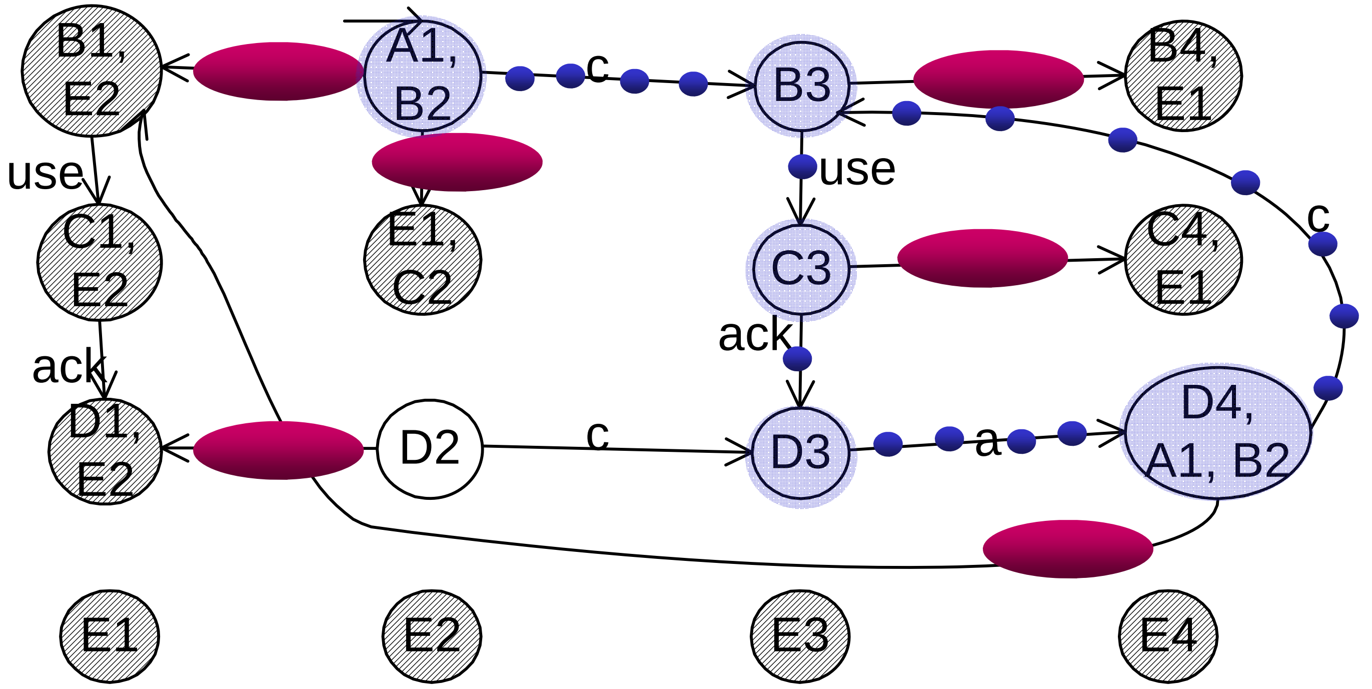
Note: * means any other interaction

. . . Product: Spec ∞ Context



- shaded states are non-accepting
- ● not visible by X (controller)

. . . after determination



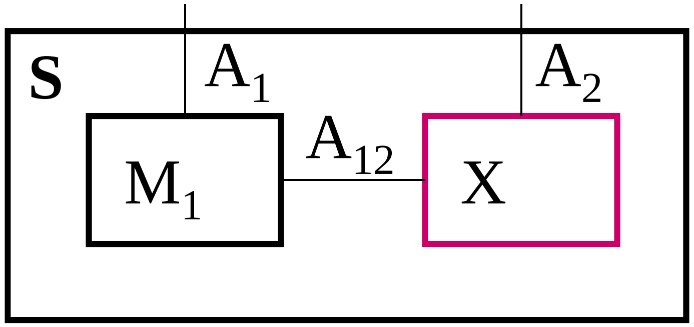
⬬ transitions to be eliminated

Other specification domains

Problem: hide A_{12} in $(M_1 \infty X) \quad \leq \quad S$

Sol: $X = (A_2 \cup A_{12})^* \setminus$ hide A_1 in $(M_1 \infty ((A_1 \cup A_2)^* \setminus S))$

- Observation: Structurally equivalent solution formula hold for different specification domains, as follows:
- Synchronous automata *[Yevtushenko]*



- Relational databases
  - "hide A_{12} " ⇔ "$\text{proj}_{\{a1,\, a2\}}$"
  - " ∞ "⇔ join *(between relations)*
- Predicate logic: Variables A_1, A_2, and A_{12} represent interaction sequences
  - Problem: **M1**(A_1, A_{12}) **and X**(A_2, A_{12}) implies **S**(A_1, A_2)
  - Solution: X(A_2, A_{12}) = **not exists** A_1' **: (M1**(A_1', A_{12}) **and not S**(A_1', A_2) **)**

Algorithms for equation solving

Sol: $X = (A_2 \cup A_{12})^* \setminus \text{hide } A_1 \text{ in } (M_1 \infty ((A_1 \cup A_2)^* \setminus S))$

- Algorithms for operations ∞ , \ , hide
  - In general not decidable (infinite sets of arbitrary sequences)
  - For finite state models (regular languages) :
    - Polynomial complexity for **∞** , **hide**
    - **hide** introduces non-determinism *(in case of non-observable interactions)*
    - \ requires conversion to deterministic models, which has exponential complexity

Overview

- Introduction
- Application areas
- Overview of differences between CD and SC
- SC solution formulas for LTS trace semantics
- Other differences: progress requirements etc.
- IO Automata and partial specifications
- Other conformance relations and issues
- Conclusions

Minimum service requirements

- Above problem definition – Safeness: S = "allowed behavior"
  - Any possible interaction sequence is included in S
- Need for some form of liveness definition
  - minimum set of sequences that must be realized (sometimes called "required behavior" in CD)
    - The above algorithms find the largest solution which may be less than S. Check that this behavior includes the minimum required.
  - Required and optional transitions *[Larson, Drissi]*
  - Progress *[Kumar, El-Fakih]*
    - For any reachable state of the system and the corresponding externally visible trace t, if the specification of S admits i as next interaction after t, then the system must be able to produce the interaction i, possibly after a certain number of internal interactions.
    - This means required behavior *(which is deterministic)* must be realized exactly *(without any possible blocking)*

Language properties for CD

Properties for sublanguages of the Plant language (for given subsets of controllable and observable interactions) :

- Controllability (e.g. maximal controllable sublanguage)
- Normality and Observability
  - normality implies observability
  - observability implies normality if controllable events are also observable

Overview

- Introduction
- Application areas
- Overview of differences between CD and SC
- SC solution formulas for LTS trace semantics
- Other differences: progress requirements etc.
- IO Automata and partial specifications
- Other conformance relations and issues
- Conclusions

Systems with input and output

Nature of input/output *(non-rendezvous)*

- Output: time and parameters of an interaction are determined by the system component producing the output
- Input: The component receiving the interaction cannot influence the time nor parameter values

Specification of component behavior

- Output: The specification gives guarantees about timing and parameter values
- Input: The specification may make assumptions about timing of inputs and the received parameter values

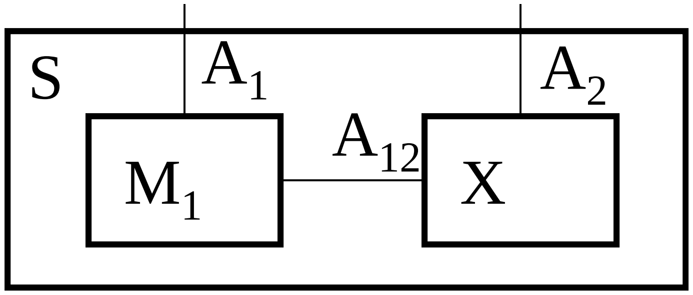
Specification paradigms with hypothesis and guarantees

- Software
  - Pre- and postconditions of a procedure call
    - They define hypotheses on input parameters, and guarantees on output parameters, respectively
- Finite state machines *(state-deterministic)*
  - Unspecified input: hypothesis about the behavior of the environment: *this input will not occur when the machine is in this state*

Conformance to specifications
based on IO sequences

- Given a specification S and a sequence T
    - Either T ε S (we say T conforms to S) or …
    - T has wrong input: all prefixes of T up some time t conform to S, but there is wrong input at time (t+1)
    - T has wrong output: similarly
    - T has wrong input and output at the same time instant
- A component conforms to a specification S iff no sequence T in which the component participates has wrong output in respect to S
    - Note: if a sequence has wrong input, nothing can be assumed about wrong output at a later time

Equation solving for specifications based on IO sequences



- Find most general specification X such that any sequence T of the composition of M_1 and X has the following properties:
  - $\text{proj}_{\{A1, A2\}}$ (T) conforms to S
  - If $\text{proj}_{\{a1, a2\}}$ (T) has no wrong input in respect to S then $\text{proj}_{\{a1, a12\}}$ (T) has no wrong input in resp. to M_1

Solution: *see [Drissi] and [Bochmann]*

Overview

- Introduction
- Application areas
- Overview of differences between CD and SC
- SC solution formulas for LTS trace semantics
- Other differences: progress requirements etc.
- IO Automata and partial specifications
- Other conformance relations and issues
- Conclusions

Different conformance relations

- What are the requirements for the behavior of the controlled system ? (in case of CD: behavior of the composition of the Context and the new component X)
- Answer (in many cases): conformance to a specification S
  - Conformance relations:
    - Equal traces (and no internal blocking) [controllability property of S indicates whether this is possible]
    - Equal traces with progress
    - Trace inclusion (and no internal blocking)
    - quasi-equivalence for IO automata
    - Additional properties: refusal semantics, state-simulations, real-time properties *[Sifakis, Grenoble],* liveness properties *[Thistle]*

Considering several specifications

- Another answer (in some cases): consideration of more than one specification. In CD, the following specifications have been considered:
  - Plant behavior (this corresponds to the behavior of the Context M_1 in SC)
  - The “allowed” behavior (subset of Plant behavior, corresponding to S in SC). Typically, trace inclusion would be required here.
  - The “required” behavior (minimum behavior as mentioned earlier, subset of “allowed” behavior). Typically, trace equivalence with progress would be required here.

Other issues

- Characterizing all solutions
  - Easy for SC with trace inclusion conformance
    - All submachines of largest solution (which is found by construction algorithm)
  - Complex for conformance with progress
    - *See [Drissi], [El-Khatib]*
- Hierarchical and distributed system models
  - E.g. distributed plant with local and global controllers
- Difficulty of the hiding operator
  - In case of unobservable events (alphabet A_1)
  - In case of internal events (alphabet A_{12})
    - e.g. for timed automata, no timer should be set on hidden transitions

Conclusions

- Application areas of SC/CD
    - Controller design
    - Protocol design (Merlin-Bochmann, 1980)
    - Design of communication gateways
    - Component reuse, e.g. in software engineering
    - Embedded testing
- Very similar concepts are used in SC and CD
- These two fields can profit from cross-fertilization
- Future directions
    - More powerful specification paradigms
        - e.g. interaction parameters and variables
    - More powerful tools
    - Practical design methodology based on formal methods