fortiss
innovation in software and systems

# Distributed Priority Synthesis using Knowledge

[AGERE!'12@SPLASH, 20121022]

**Chih-Hong Cheng**

**Computer-Aided Synthesis and Verification (CASV) Group**

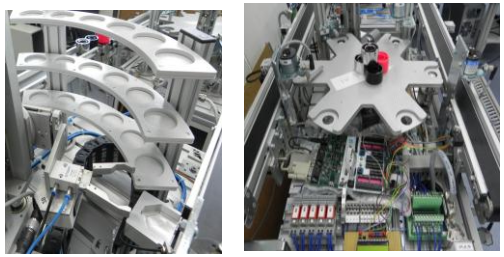
**Fortiss - An-Institut der Technischen Universität München**

**Joint work with R.-J. Yan (ISCAS), S. Bensalem (Verimag), and H. Ruess (Fortiss)**

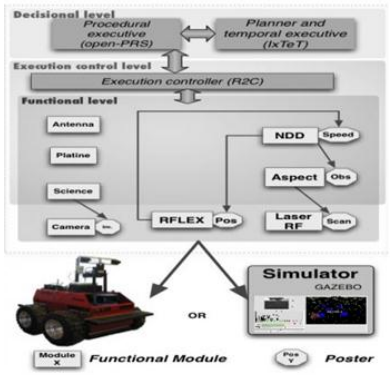
# Synthesis – making programming implicit
## with applications in embedded systems

- Goal: automatic generation of correct software controllers for embedded systems

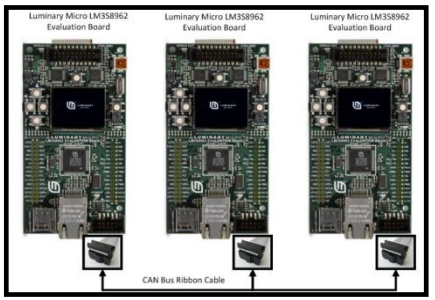
- Applications:



*Automation systems*




*Robotics*

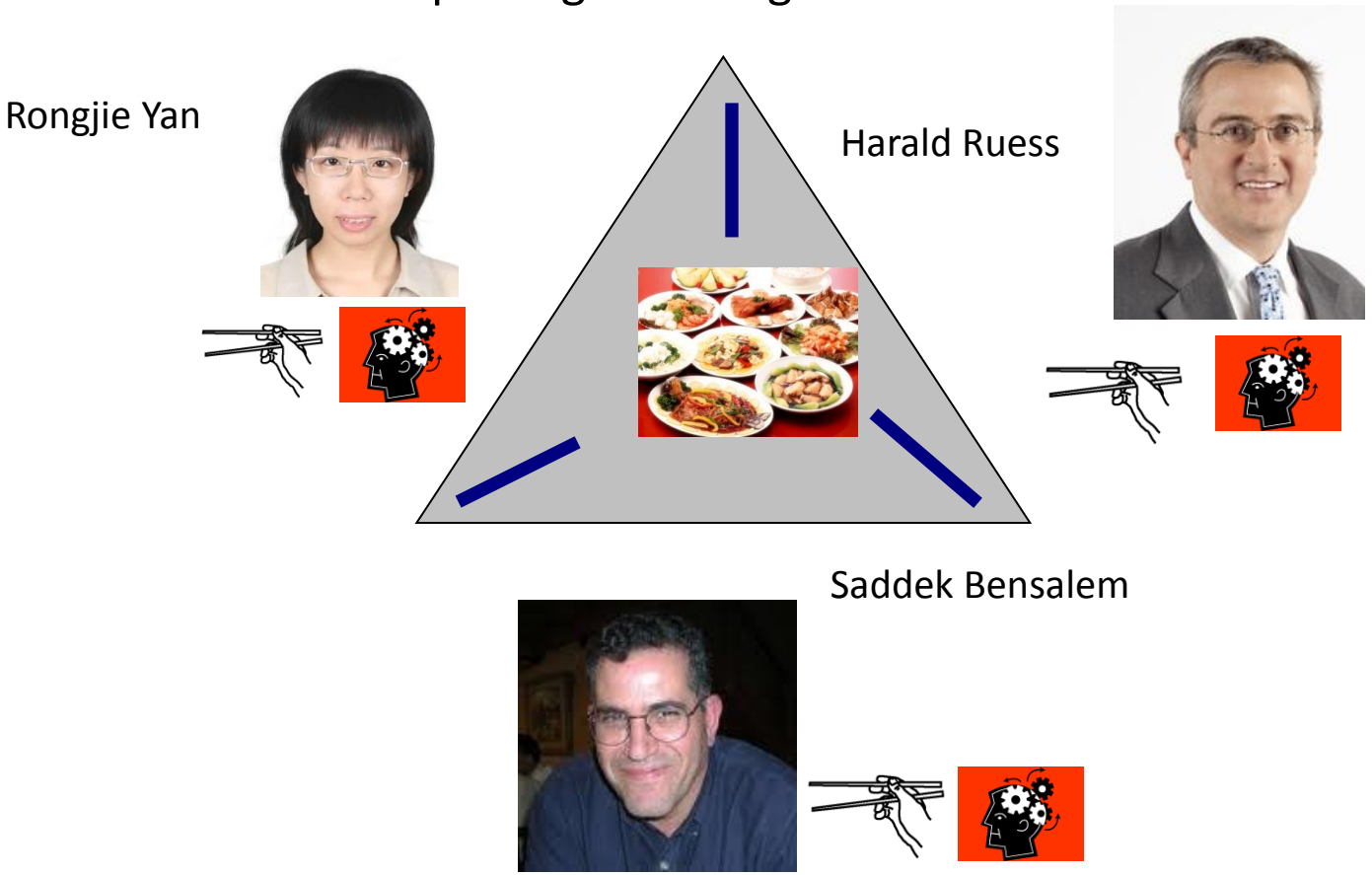



*Education*

- This talk: **automatic orchestration among multiple components**
  - See other aspects that study games in theoretical computer science (GAVS+ [TACAS'11]), synthesize single components (G4LTL [under review]), or apply synthesis in industrial automation (MGSyn [CAV'12])

# Dining Philosophers Problem

- Two types of components: philosophers, chopsticks
  - Problem occurs when putting them together



# Resolving Deadlock using "Politeness Rules"

## philosopher is aware of others and constrains himself

Politeness rule:

Harald_Ruess.take_**Left** ≺ Saddek_Bensalem.take_**Right**

Rongjie Yan

Harald Ruess

STOP

Harald is polite and does not use his left chopstick

Saddek Bensalem

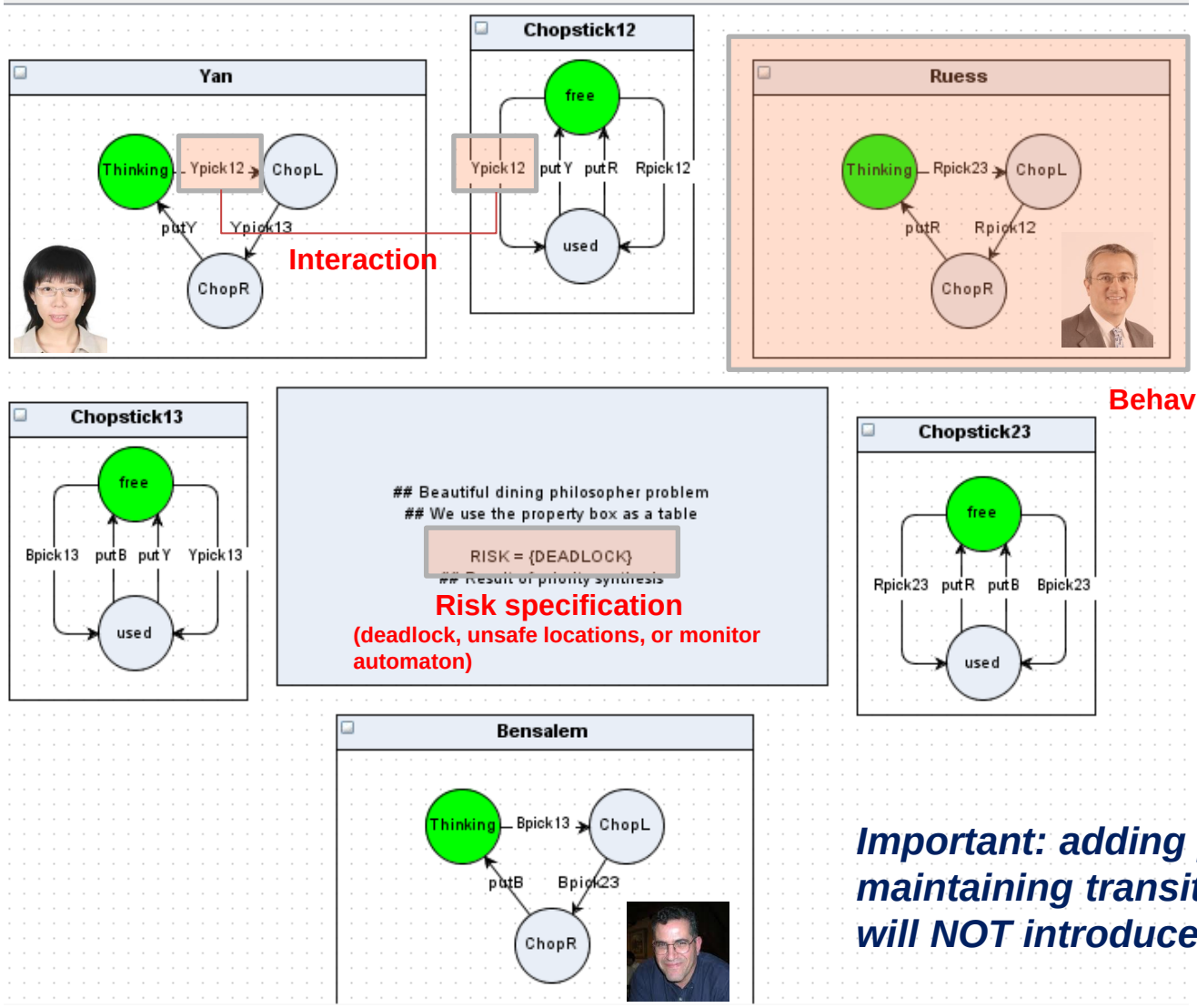
**Interaction Priority =**

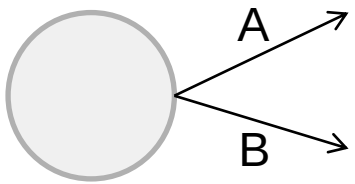
**(conditional precedence between interactions)**

**Politeness Rules**

# VissBIP for component-based synthesis

## Remove deadlock for dining philosophers by synthesizing politeness rules




A≺B: disable A when A and B are both available

(A will not be influenced when B is not available)

***Important: adding priorities (while maintaining transitivity and irreflexibility) will NOT introduce new deadlocks***

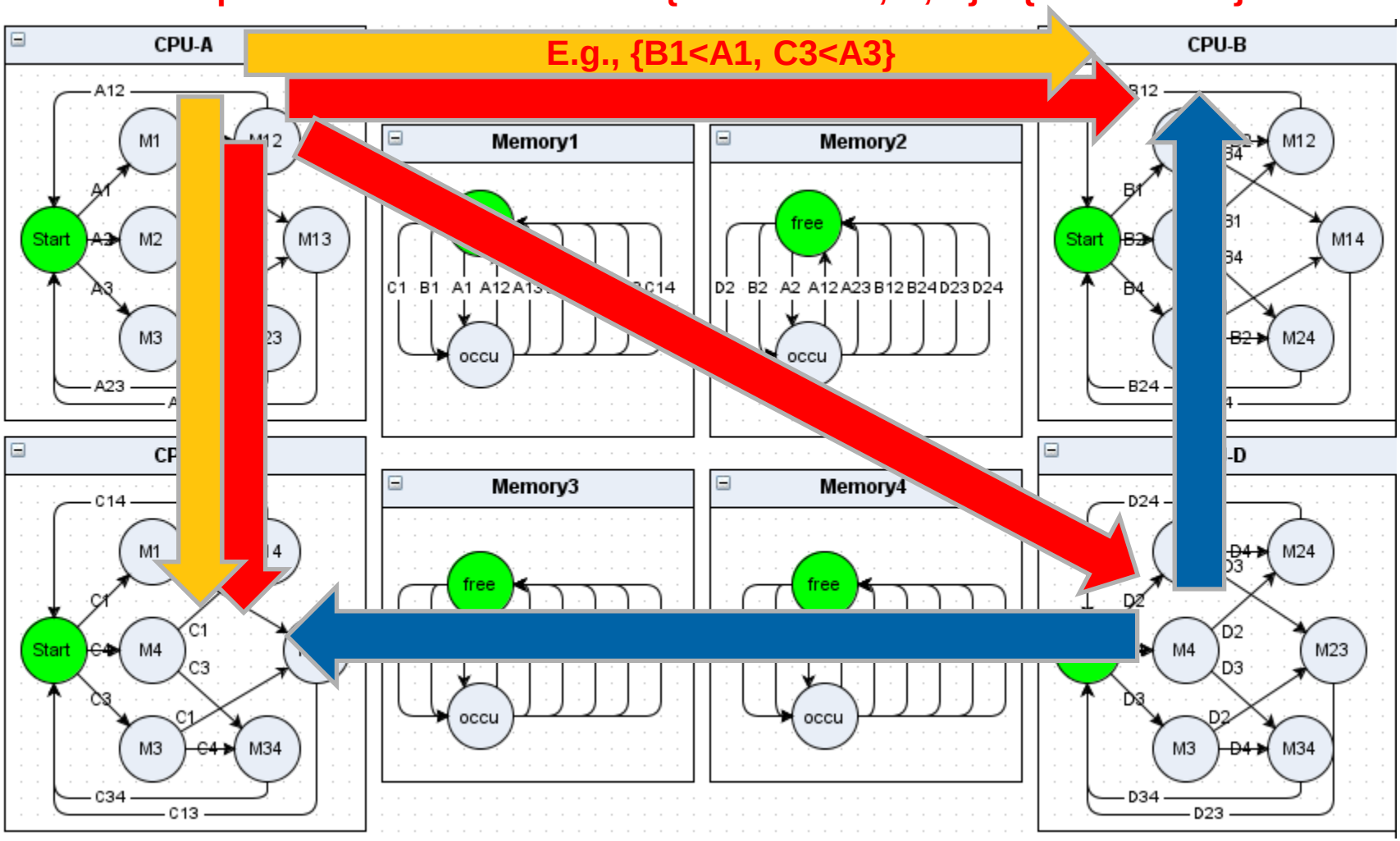
# Demonstration

- Dining philosophers (use the problem that matches the number of attendees)
- Erroneous implementation for Peterson's algorithm
- DPU signal processing unit

# Communication poses additional constraints

## Distributed priority synthesis: Resolve deadlock in multi-processor scenario

**Impose constraints on rules: {actions of B, C, D} < {actions of A}**



# Previous talk

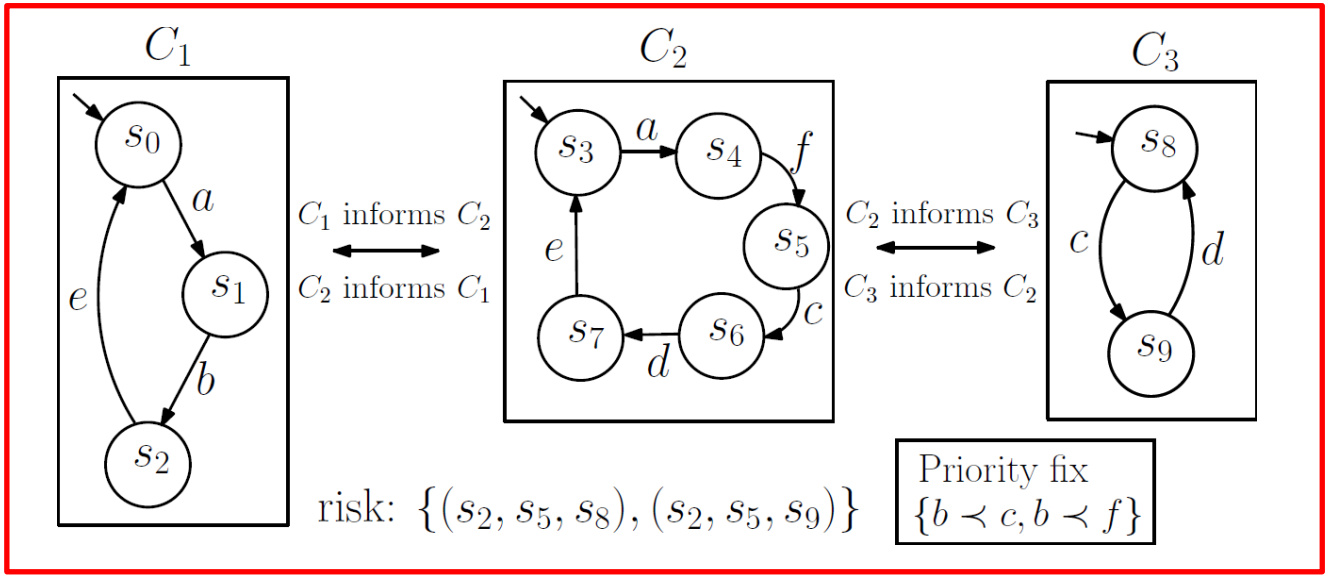
- *Optimized Distributed Implementation of Multiparty Interactions with Observation*
  - Saddek Bensalem, Marius Bozga, Jean Quilbeuf and Joseph Sifakis

- *Observation*: conditions to perform execution for multiparty interactions to ensure correctness
  - Generalized formulation that subsumes ***priorities*** and ***knowledge*** (global information of the system state from partial view)

- Focus of this paper - “synthesis”
  - How to automatically generate correct/safe distributed controllers (i.e., generate observations) using knowledge?
  - The distribution problem is handled by the previous talk

# OK, back to the simplest example

- $C_1$ and $C_3$ can't see each others' intention
  - Priority $b \prec c$ is disallowed to be used – as $C_1$ can't make decision whether or not to execute interaction $b$.
  - [Distributed priority synthesis] no feasible solution
- Knowledge: infer global information from local view
  - [Problem] Can things go better when we utilize knowledge?



# Knowledge for distributed systems
## (difference between execution and synthesis)

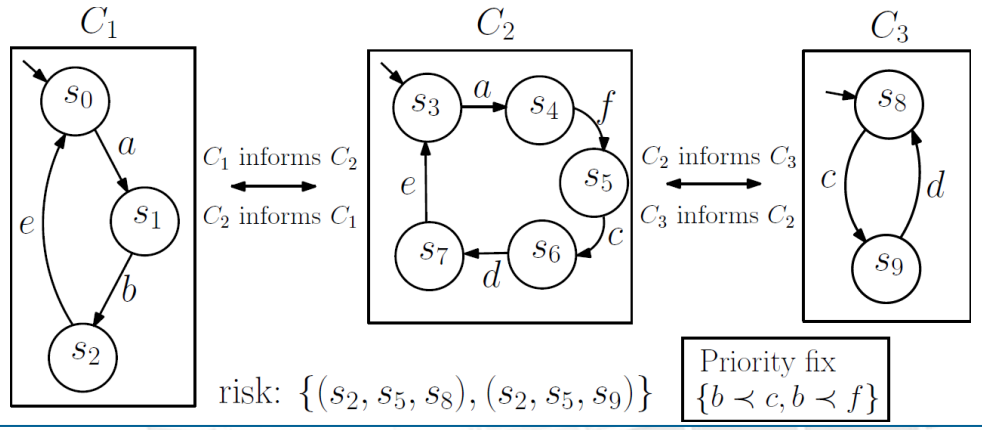
- System execution
  - $b \prec c$: components executing $b$ can safely **execute** $b$ if they can infer (by knowledge) that $c$ **is not enabled**
  - Reduce communication overhead
  - Knowledge is used at **run-time** in **negated (absence)** form

- System synthesis
  - $b \prec c$: used as a gadget to constrain some execution paths
    - » We want to introduce $b \prec c$ to restrict $b$, on knowing that $c$ **is enabled**
  - **[Observation]** Knowing something is not enabled is not useful when we want to introduce priority-based controllers
  - Knowledge is used at **compile/synthesis-time** in **positive (presence)** form

# Visibility, fact and knowledge

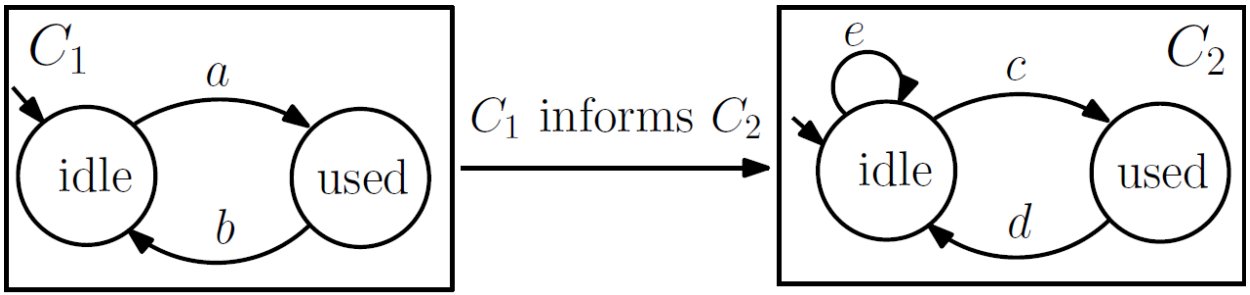
- **[Visibility]** $C_1$ is visible to $C_2$ if $C_1$ can pass its intention to $C_2$
- **[Fact]** We define a local component can understand
  - **[From itself]** Its own local state, its locally enabled interactions
  - **[From visible components]** Their locally enabled interactions
  - **[From non-visible components]** Nothing
- **[Knowledge]** We define a local component can infer
  - whether an (originally not visible) interaction is globally enabled

$$\mathcal{K}_1^{\mathrm{STR}}(b, s_1, \{C_1.b\}, \{C_2.d\}, \emptyset)$$



# Strong knowledge and weak knowledge

- Strong knowledge
  - Every interaction in the set is guaranteed to be globally enabled
- Weak knowledge
  - At least one interaction in the set is guaranteed to be globally enabled
  - In Fig. 3, $C_1$ knows that at least one element in {c, d, e} is enabled

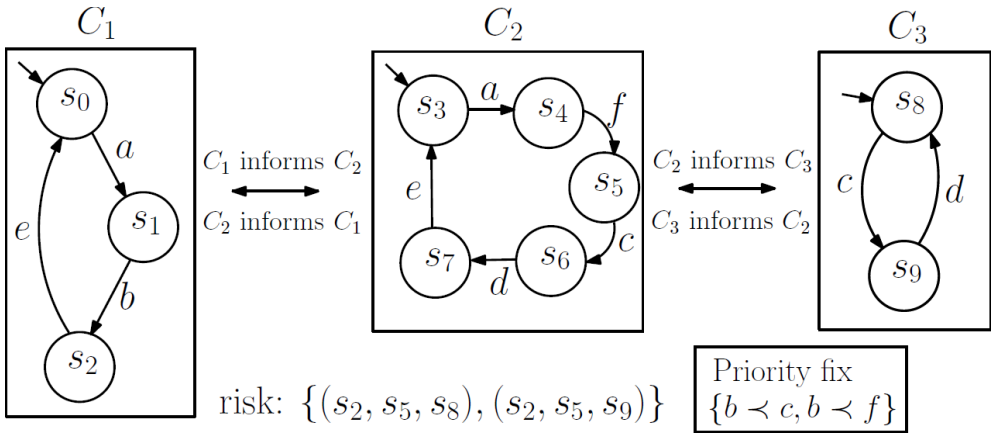


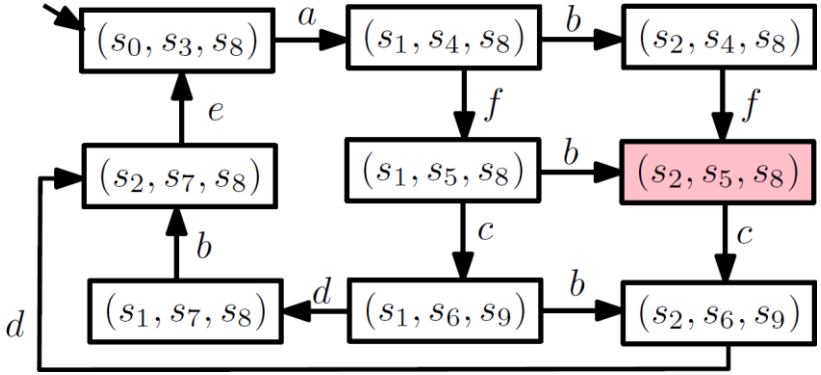

- Use weak knowledge in synthesis => introduce $\{a \prec e, a \prec c, a \prec d\}$ with the guarantee that one of them blocks interaction $a$

# Knowledge computation
## (can be computed symbolically)

$$\mathcal{K}_1^{\text{STR}}(b, s_1, \{C_1.b\}, \{C_2.d\}, \emptyset)$$

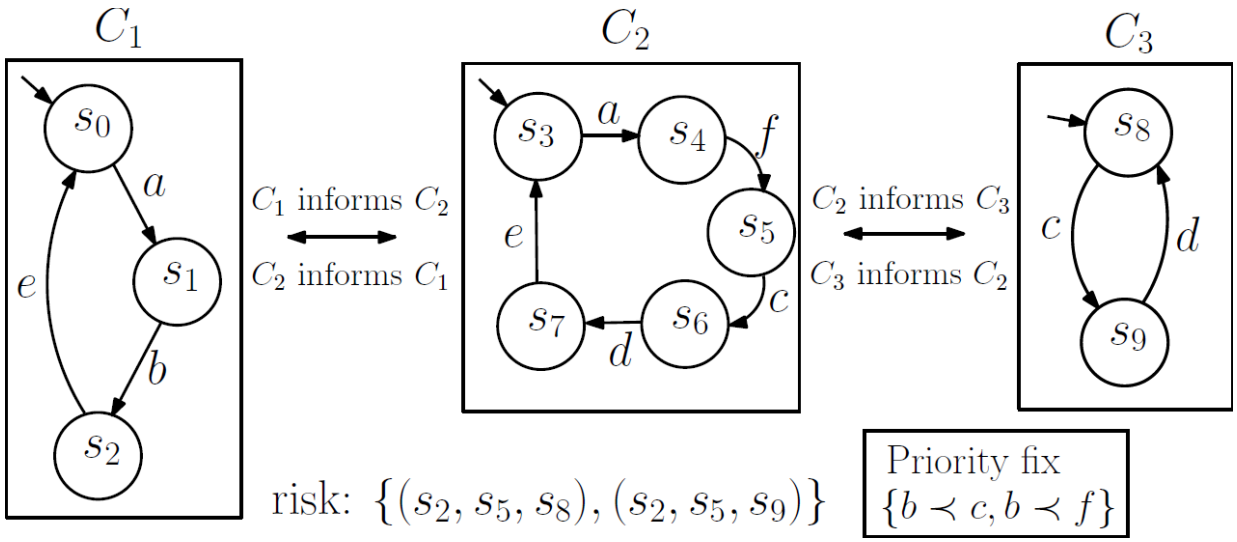
- Step 1: Derive locations when $d$ is locally enabled at $C_2$.
  - Generate $\{s_6\}$
- Step 2: Compute the set of reachable states with $C_1.state = s_1$ and $C_2.state \in \{s_6\}$
  - Generate S = $\{(s_1, s_6, s_9)\}$
- Step 3: Check if there's any interaction that is enabled in all states of S (strong knowledge) and is not a fact (i.e., $b$).
  - Return $\{d\}$ as strong knowledge



# Knowledge-based distributed priority synthesis (KDPS)

- Strictly more powerful than distributed priority synthesis (DPS)
  - For example below, no feasible solution for DPS
  - The fix is realizable via KDPS



# Putting everything altogether (work-in-progress)

## An algorithm of synthesizing controllers using 2QBF

- Formulating priority synthesis via 2QBF:

  $\exists\ priorities, Invariants\ \forall\ s, s' \in States$:

  1. $initial\ state \in Invariants$
  2. $risk\ states \notin Invariants$
  3. $\left(s \in Invariants \wedge (s, s') \in Transition_{priorities}\right) \rightarrow s' \in Invariants$
  4. $Created\ priorities\ are\ irreflexive\ and\ transitive$

$Invariants$ = An under-approximation of the set of all safe states in template form

- Extend the concept to distributed synthesis
  - Add additional constraints due to component visibility
- Extend the concept to knowledge-based priority synthesis
  - Add additional existential variables to indicate the triggering of knowledge
  - When using weak knowledge, one priority in the knowledge enabled => all priorities enabled. E.g., $\{a \prec e, a \prec c, a \prec d\}$

# In this talk

- Synthesis for component-based systems
  - Regulate dining philosophers via priorities
  - Distributed priority synthesis = priority synthesis + architectural constraints
- Knowledge-based priority synthesis
  - Assist synthesis of priorities in distributed systems
  - Strictly more powerful then pure distributed priority synthesis
  - Need to use it positively in compile/synthesis time
  - Notion of strong and weak knowledge
  - We can compute it statically
  - Once computed, integration into 2QBF-based solvers (work-in-progress)
  - Efficient deployment achieved by the paper in previous talk
  - Synthesis methodology applicable for general MoCs
    - » e.g., BPEL (in progress in Fortiss), behavioral programming (above-above talk)

# Contact

**Dr. Chih-Hong Cheng**

Research Group Leader: Computer-Aided Synthesis & Verification
fortiss – An-Institut der Technischen Universität München
Guerickestr. 25 | 80805 München | Germany
Tel. +49 89 360 35 22 – 511 | Fax +49 89 360 35 2250
cheng@fortiss.org | www.fortiss.org