CAS 745
Supervisory Control of Discrete-Event Systems

Slides 4: Decentralized Supervision

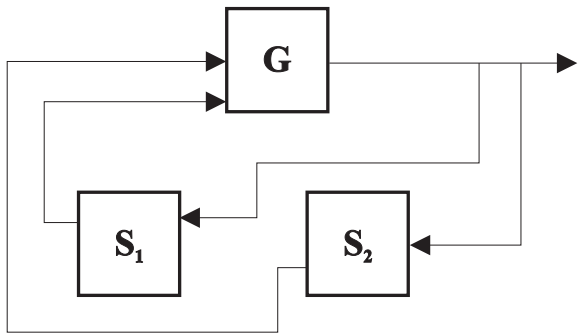
Dr. Ryan Leduc

Department of Computing and Software
McMaster University

Material based on W. M. Wonham, Supervisory Control of Discrete-Event Systems, Department of Electrical and Computer Engineering, University of Toronto, July 2004. Lecture notes of Professor Wonham also used.

Modular Supervision Intro - §4.1

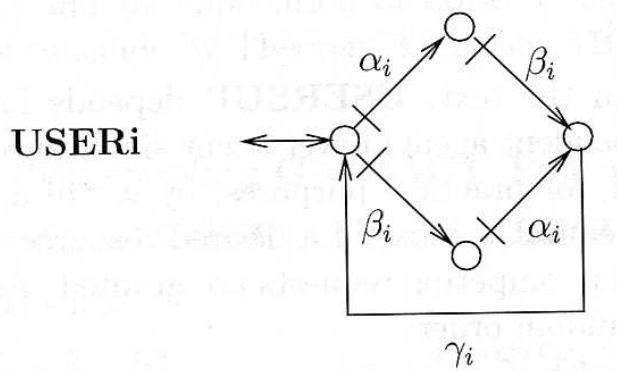
- So far we have talked about a single centralized supervisor.
- We can also design our supervisor as two or more specialized supervisors, each designed to perform a specific subtask.



- **Advantage:** Each supervisor, say $\mathbf{S}_1$ and $\mathbf{S}_2$, would be simpler to design and implement than a single "monolithic" supervisor.
- **Disadvantage:** Supervisors $\mathbf{S}_1$ and $\mathbf{S}_2$ may be individually controllable and nonblocking for plant, but acting together, they cause the plant to block.

"Deadly Embrace" Example - §4.3

- This example shows how two supervisors can be fine separately, but block working together.
- We have two shared resources (say a notepad and a pen), and two users who need to acquire both to complete their task. Also, the resources can be acquired in any order.
- For example, α_1 could mean user1 acquires the pen (resourceA), and β_1 could mean user1 acquires the notepad (resourceB). User1 would then be able to complete his/her task, say, jot down an idea (γ_1).
- This idea is captured in DES USERi, $i = 1, 2..$

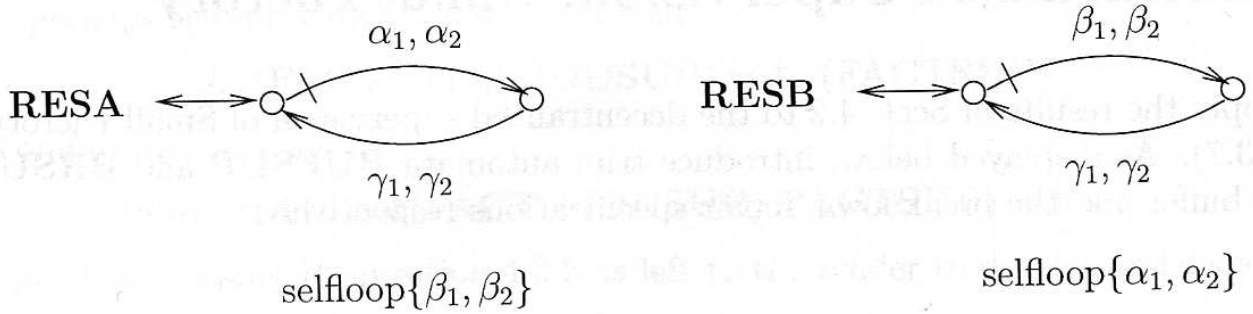


"Deadly Embrace" Example - II

- Our plant is thus:

$$\textbf{USER} = \textbf{sync}(\textbf{USER1}, \textbf{USER2})$$

- We then proceed to design two naive modular supervisors (**RESA** and **RESB**), to control access to each resource.



- Separately, both **RESA** and **RESB** are controllable and nonblocking for **USER** (can be verified with **condat** and **nonconflict**).

- Let **RES** = **meet**(**RESA**, **RESB**).

- Using **nonconflict**, we can see:

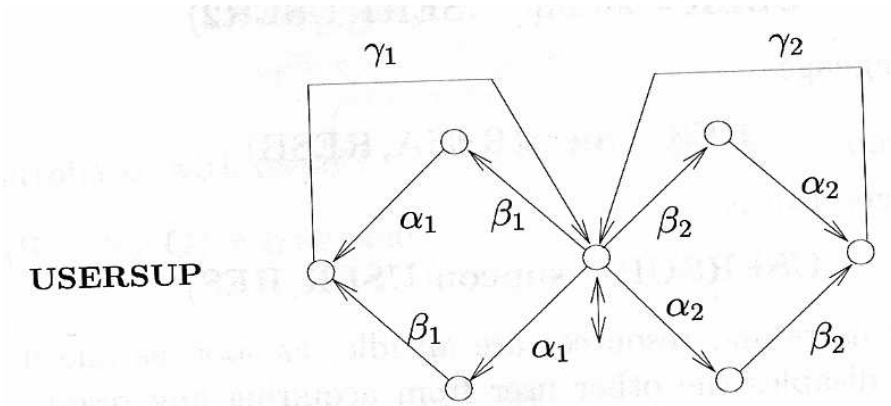
$$\textbf{nonconflict}(\textbf{USER}, \textbf{RES}) = \text{FALSE}$$

"Deadly Embrace" Example - III

- This is because nothing prevents one user to acquire resource A, and then the other user to acquire resource B (say string $\alpha_1\beta_2$ or $\alpha_2\beta_1$). Neither can release their resource, or gain the other resource needed to complete their task.

- The correct centralized supervisor can be constructed using supcon:

$$\textbf{USERSUP} = \textbf{supcon}(\textbf{USER}, \textbf{RES})$$



Conjunction of Supervisors - §4.2

- Let $\mathbf{S}_1 = (X_1, \Sigma, \xi_1, x_{o_1}, X_{m_1})$ and $\mathbf{S}_2 = (X_2, \Sigma, \xi_2, x_{o_2}, X_{m_2})$ be proper supervisors for $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$.

- That is to say, $\mathbf{S}_i$ $(i = 1, 2)$ are each trim, controllable for $\mathbf{G}$, and $\mathbf{S}_i/\mathbf{G}$ are each nonblocking.

- **Defn:** The conjunction of supervisors $\mathbf{S}_1$ and $\mathbf{S}_2$, written $\mathbf{S}_1 \wedge \mathbf{S}_2$, is defined to be:

$$\mathbf{S}_1 \wedge \mathbf{S}_2 = \mathbf{meet}(\mathbf{S}_1, \mathbf{S}_2)$$

- Behavior is to enable $\sigma \in \Sigma$ in plant $\mathbf{G}$ only when both $\mathbf{S}_1$ and $\mathbf{S}_2$ both enable σ. ie. a logical and of their enablement output.

Design Steps

1. Model plant as components $\mathbf{G}_i$, $i = 1, 2, \ldots, m$.

2. Combine plant components into "monolithic" model $\mathbf{G} = \mathbf{G}_1 || \ldots || \mathbf{G}_m$.

3. Design supervisors $\mathbf{S}_i$, $i = 1, 2, \ldots, n$, individually.

4. Construct conjunction of supervisors $\mathbf{S} = \textbf{meet}(\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n)$.

5. Use **condat** to check that $\mathbf{S}$ is controllable for plant $\mathbf{G}$.

6. Use **nonconflict** to verify that $\mathbf{S}/\mathbf{G}$ is nonblocking.

NOTE: If condat shows that each $\mathbf{S}_i$ is individually controllable for $\mathbf{G}$, then $\mathbf{S}$ is controllable for plant $\mathbf{G}$. Why? Intersection of controllable closed languages $L(\mathbf{S}_i)$ is also controllable.

Modular Supervision: Results

- **Proposition 4.2.1** Let $K_1, K_2 \subseteq \Sigma^*$ be controllable with respect to $\mathbf{G}$. If K_1 and K_2 are nonconflicting then $K_1 \cap K_2$ is controllable with respect to $\mathbf{G}$. $\square$

- Consider applying to $K_1 = L_m(\mathbf{S}_1)$ and $K_2 = L_m(\mathbf{S}_2)$.
  NOTE: two closed languages are always nonconflicting.

- **Proposition 4.2.2** Let $E_1, E_2 \subseteq \Sigma^*$. If $\mathrm{sup}\mathcal{C}(E_1)$, $\mathrm{sup}\mathcal{C}(E_2)$ are nonconflicting then

  $$\mathrm{sup}\mathcal{C}(E_1 \cap E_2) = \mathrm{sup}\mathcal{C}(E_1) \cap \mathrm{sup}\mathcal{C}(E_2) \qquad \square$$

- In practice, construct DES $\mathbf{E} = \mathbf{meet}(\mathbf{E}_1, \mathbf{E}_2)$ then $\mathbf{Sup} = \mathbf{supcon}(\mathbf{G}, \mathbf{E})$, as usually involves less computation.

Modular Supervision: Results - II

- **Defn:** For DES $\mathbf{G}_1$ and $\mathbf{G}_2$, we say that "$\mathbf{G}_1$ and $\mathbf{G}_2$ are behaviorally equivalent," written $\mathbf{G}_1 \approx \mathbf{G}_2$, if:

$$L_m(\mathbf{G}_1) = L_m(\mathbf{G}_2), \quad L(\mathbf{G}_1) = L(\mathbf{G}_2)$$

- **Theorem 4.2.2** Assume:

  (i) $\mathbf{S}_1$ and $\mathbf{S}_2$ are controllable with respect to $\mathbf{G}$ [as confirmed, say, by **condat**],

  (ii) **nonconflict**$(\mathbf{S}_1 \wedge \mathbf{S}_2, \mathbf{G}) =$ TRUE, and

  (iii) $\mathbf{S}_1 \wedge \mathbf{S}_2$ is trim

  then $\mathbf{S}_1 \wedge \mathbf{S}_2$ is a proper supervisor for $\mathbf{G}$, with

$$(\mathbf{S}_1 \wedge \mathbf{S}_2) \wedge \mathbf{G} \approx (\mathbf{S}_1 \wedge \mathbf{G}) \wedge (\mathbf{S}_2 \wedge \mathbf{G}) \qquad \square$$

- For $\mathbf{G}_1 \wedge \mathbf{G}_2$, read **meet**$(\mathbf{G}_1, \mathbf{G}_2)$

Modular Supervision: Results - III

- **Corollary 4.2.2** Let $\mathbf{E}_1$ and $\mathbf{E}_2$ be arbitrary DES and $\mathbf{S}_i = \textbf{supcon}(\mathbf{G}, \mathbf{E}_i)$, $i = 1, 2$.

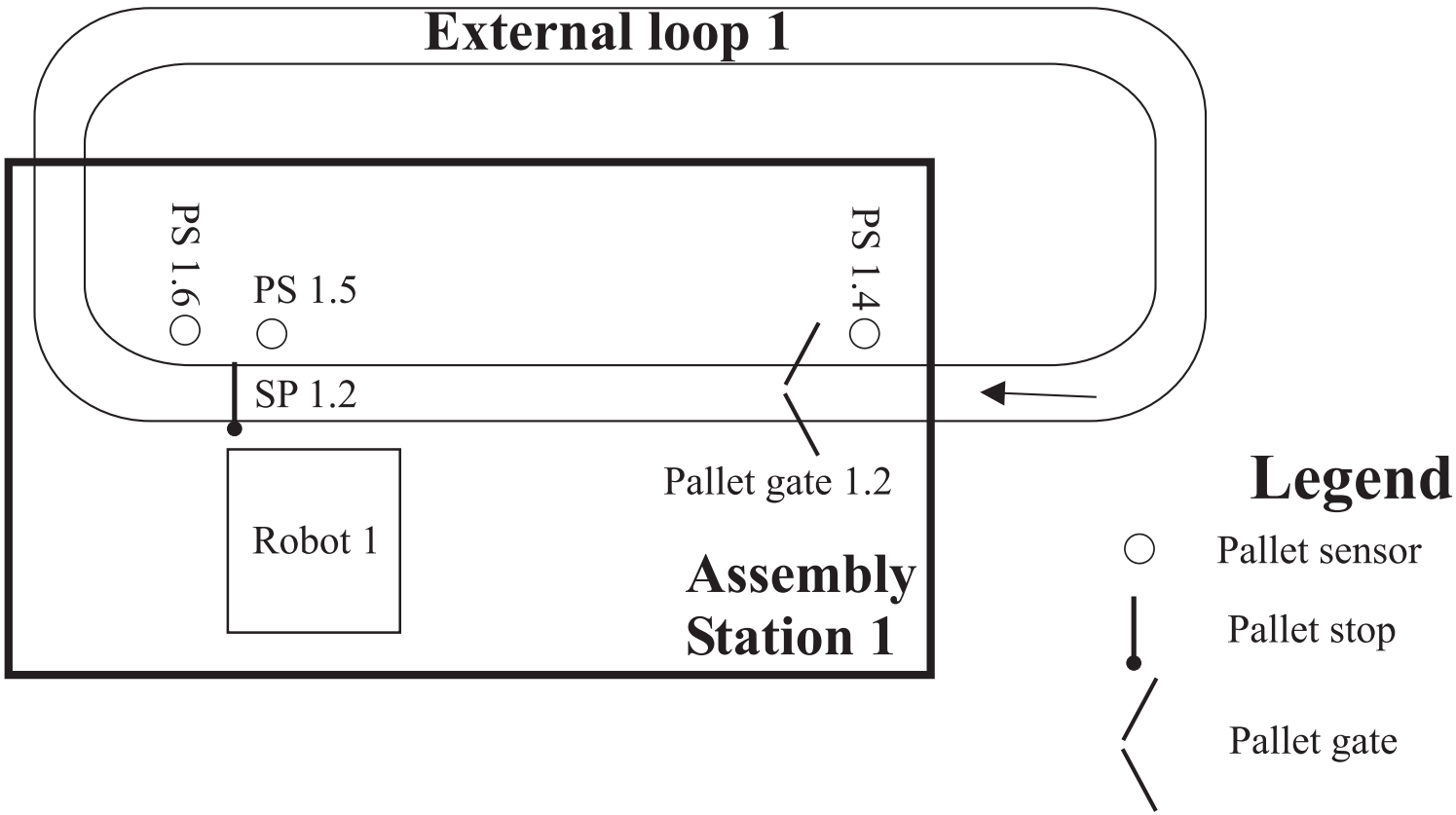
- If $\textbf{nonconflict}(\mathbf{S}_1, \mathbf{S}_2) =$ TRUE, then

$$\mathbf{S}_1 \wedge \mathbf{S}_2 \approx \textbf{supcon}(\mathbf{G}, \mathbf{E}_1 \wedge \mathbf{E}_2) \qquad \square$$

- Read examples in Sections 4.4 and 4.5 on your own.

Modular Supervisors for Assembly Station 1

- In the following slides, we return to the assembly station.
- We will now design modular supervisors for it.
- Refer to earlier slides to refresh your memory re: the plant model.



Control Specifications: AS1

1. To avoid collisions, only one pallet is allowed in assembly station at a given time.

2. When a pallet arrives at the pallet stop, the robot will first perform task1a on it, and then task1b.

3. If there is an assembly error or the robot breaks down, then the pallet will be released from the station without finishing the remaining assembly.

4. Except in the above case, a pallet is not released from the station until the assembly operation has been performed.

5. When robot breaks down, it should be repaired right away.

Expansion Events

- Sometimes the events that are physically a part of a plant do not allow for easy communication between supervisors.
- Often we will want to separate the control of the operation of particular components from the control of when the component should perform a task.
- For example, the supervisor in Fig K controls the pallet stop to allow a pallet to leave system, but has no information about when it should release the pallet.
- This allows us to have several small supervisors for each component, and then another supervisor that handles the higher level details of when the components should perform these tasks. For our example, this was the supervisor in Fig I.

Expansion Events - II

- We can add expansion events as a new plant with these controllable events selflooped (see Fig H).
- The component supervisor then waits for this event to occur to perform its task, and the other supervisor disables the event until it wishes the task to be performed.
- There may also be one or more expansion events that are also used by the component supervisor to signal that the task is completed as well as information about the results (error, successful).

Plant to add Expansion Events

- Plant ASNewEvents added to introduce the needed expansion events to the system.

Fig H: ASNewEvents

Supervisor: HndlPallet

- This is the primary supervisor that controls the operation of the assembly. It directs the other supervisors, as needed.
- For this and every other supervisor in example, assume that they are self looped at each state by every event in Σ for which they don't have a transition. The self loops are simply not shown to keep the diagrams simple.

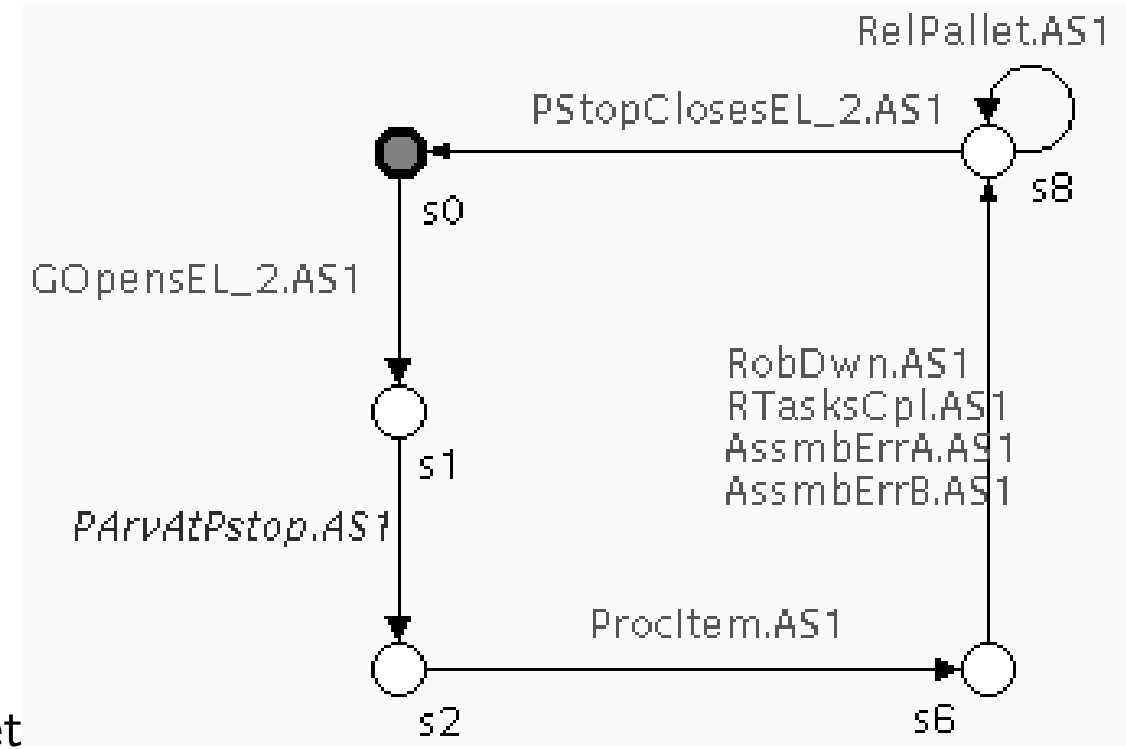


Fig I: HndlPallet

Supervisor: OperateGateEL_2

- Controls operation of gate.
- Ensures gate doesn't open till pallet present.

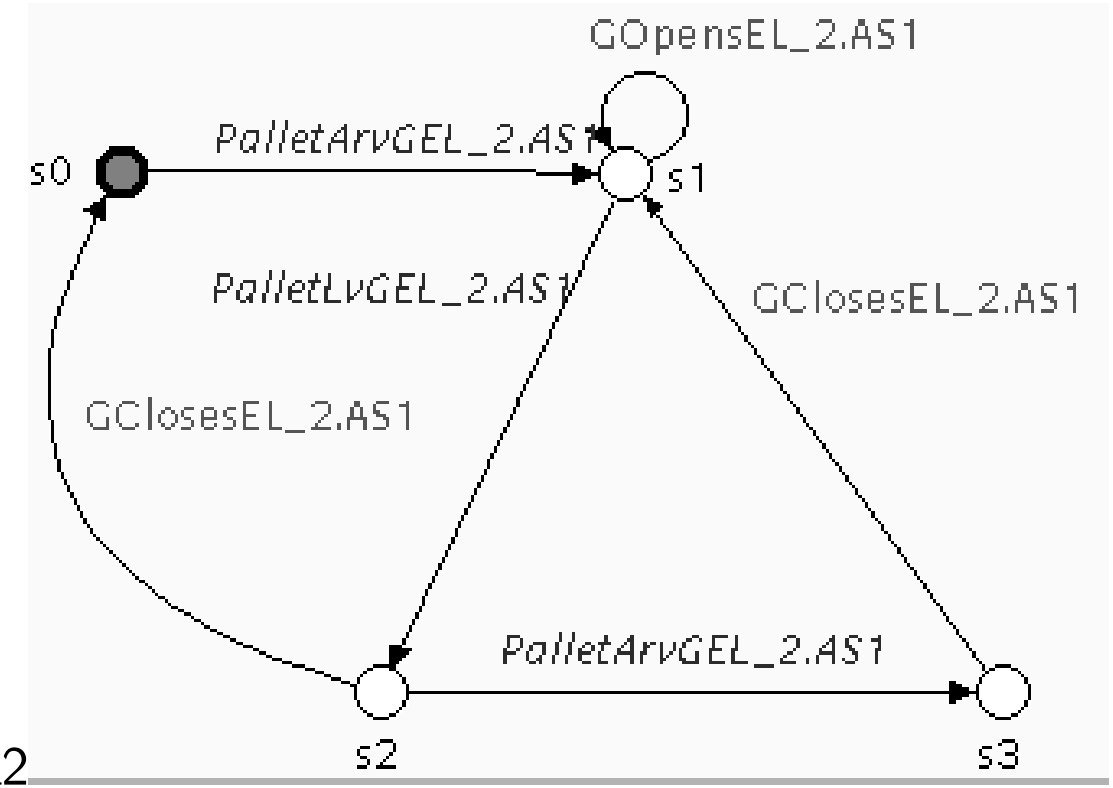


Fig J: OperateGateEL_2

Supervisor: HndlPalletLvAS

- Controls operation of pallet stop to allow pallet to leave station.

- Waits for expansion event *RelPallet.AS1* to occur to tell it when to release the pallet.

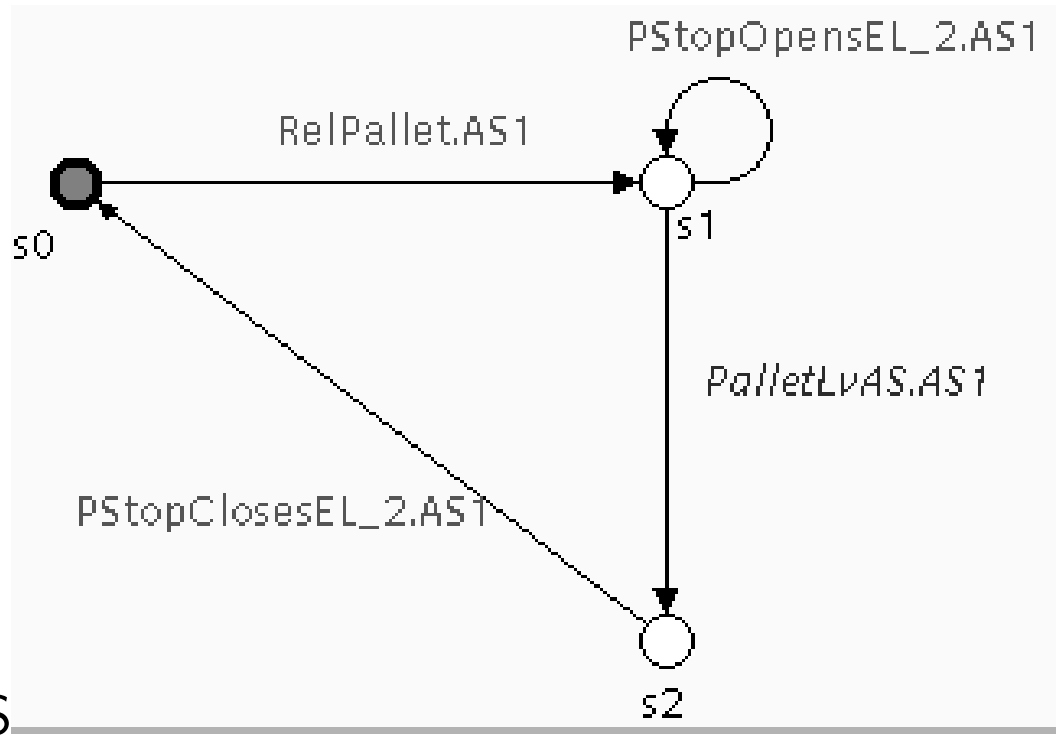
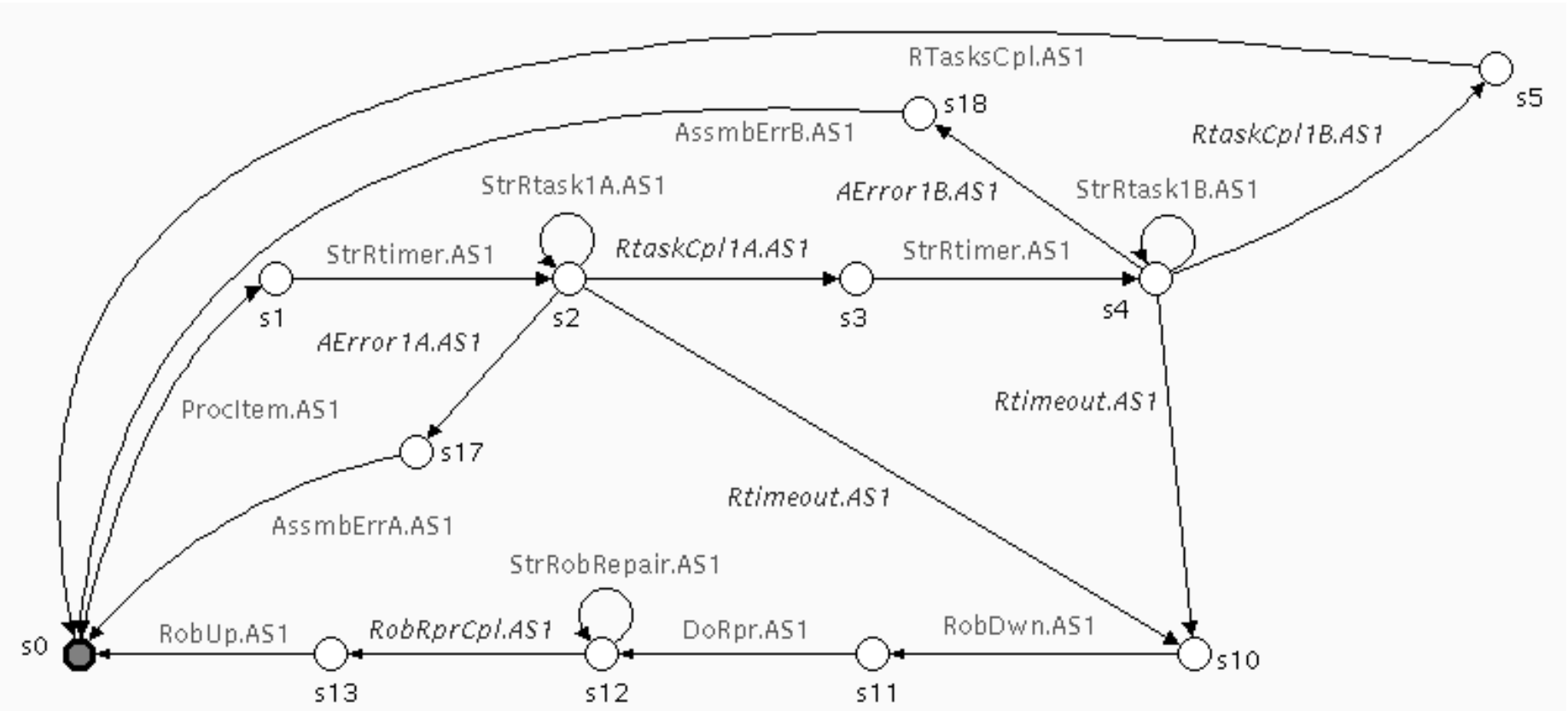


Fig K: HndlPalletLvAS

Supervisor: DoRobotTasks1

- Shown in Fig. L below, this supervisor controls the operation of the robot.
- When event *ProcItem.AS1* occurs, it has robot first perform task1a and then task 1B, then it signals back the results.
- If Robot breaks down, it initiates repair.



Final Result

- Our new plant is

$$\textbf{PLANT} = \text{sync}(B, C, D, E, F, G, H)$$

  where B refers to the DES in Fig. B etc.
- Our supervisor is

$$\textbf{SUP} = \text{meet}(I, J, K, L)$$

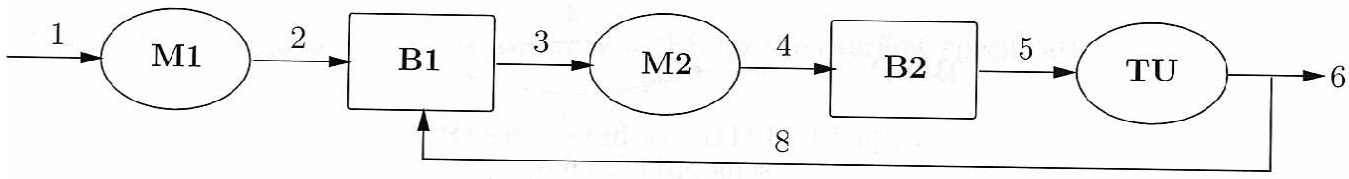
- Our closed loop behavior is

$$\textbf{CL} = \text{meet}(\textbf{PLANT}, \textbf{SUP})$$

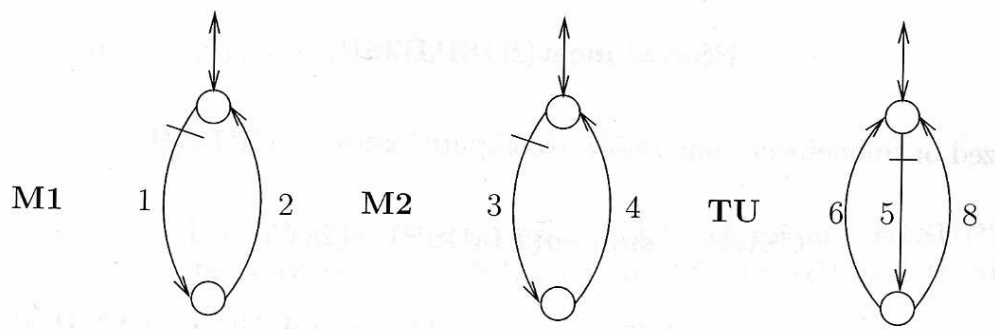
- Using TCT, we can easily show that **SUP** is controllable for **PLANT** and that **CL** is nonblocking.

Transfer Line Example - §4.6

- For this example, we have two machines **M1**, **M2**, followed by a test unit **TU**, which are linked by buffers **B1** and **B2**, as shown below.



- A work piece tested by **TU** can be accepted (6), or rejected (8).
- If accepted, the piece is allowed to leave the system. If rejected, it is placed back into **B1**.
- The models of the plant components are shown below.

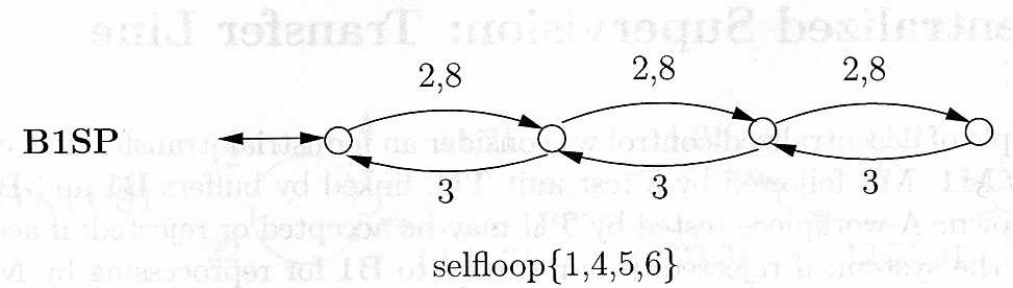


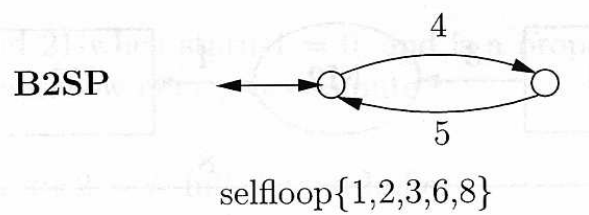
Transfer Line Example - II

- Our plant is thus:

$$\mathbf{TL} = \mathbf{sync}(\mathbf{M1}, \mathbf{M2}, \mathbf{TU})$$

- The capacity of **B1** is 3, and of **B2** is 1.
- **Specification:** Buffers **B1** and **B2** must not underflow or overflow.
- These specifications can be modelled as DES **B1SP** and **B2SP**.





Transfer Line Example - III

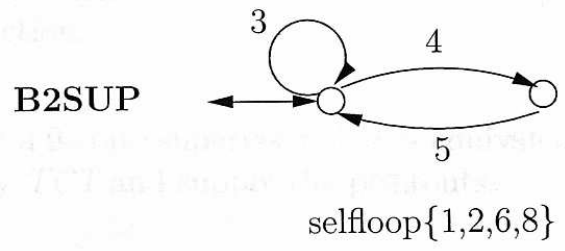
- Our complete spec is:

$$\textbf{BSP} = \textbf{meet}(\textbf{B1SP}, \textbf{B2SP})$$

- The centralized ("monolithic") supervisor is:

$$\textbf{CSUP} = \textbf{supcon}(\textbf{TL}, \textbf{BSP})\ (28, 65)$$

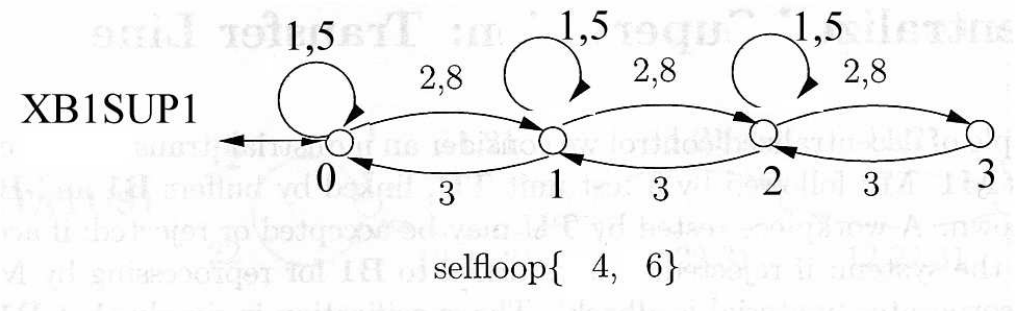
- A modular supervisor for **B2** is easy to design as it is similar to the small factory buffer. See **B2SUP** below:



- For **B1**, we first design a supervisor to prevent underflow. We simply take as supervisor **B1SUP2**, the spec **B1SP**.

Transfer Line Example - IV

- We now need to design supervisor **B1SUP1** to make sure that **B1** doesn't overflow. We take as our first naive attempt, supervisor **XB1SUP1** below.



- We thus take as our supervisor:

$$\textbf{XBSUP} = \textbf{meet}(\textbf{XB1SUP1}, \textbf{B1SUP2}, \textbf{B2SUP})$$

- We can then use $\textbf{condat}(\textbf{TL}, \textbf{XBSUP})$ to determine that **XBSUP** is indeed controllable.

- Unfortunately,

$$\textbf{nonconflict}(\textbf{TL}, \textbf{XBSUP}) = \text{FALSE}$$

Transfer Line Example - V

- Why? **XB1SUP1** doesn't ensure enough space for a failed piece to be put into **B1**.
- Each DES will admit the string:

$$s = 1\,2\,1\,2\,1\,2\,3\,4\,1\,2$$

- String leaves **B1** and **B2** full. After s, **B2SUP** disables **M2**, while **XB1SUP1** disables **M1** and **TU**.
- When we consider the overall system, we see that any workpiece removed from **B1** could potentially be returned by **TU**. We must ensure space for it.
- We must ensure the following doesn't happen:

$$\textbf{Num\_Wpieces\_In}\{\textbf{M2}, \textbf{B2}, \textbf{TU}\} > \textbf{Num\_Empty\_Slots\_B1}$$

- We can ensure this by disabling **M1** when:

$$\textbf{Num\_Wpieces\_In}\{\textbf{M2}, \textbf{B2}, \textbf{TU}\} \geq \textbf{Num\_Empty\_Slots\_B1}$$

Transfer Line Example - VI

- Noting that **B1** has three slots, gives:

$$\begin{aligned} \textbf{Num\_Empty\_Slots\_B1} &= 3 + \#3 - \#2 - \#8 \\ \textbf{Num\_Wpieces\_In}\{\textbf{M2}, \textbf{B2}, \textbf{TU}\} &= \#3 - \#6 - \#8 \end{aligned}$$

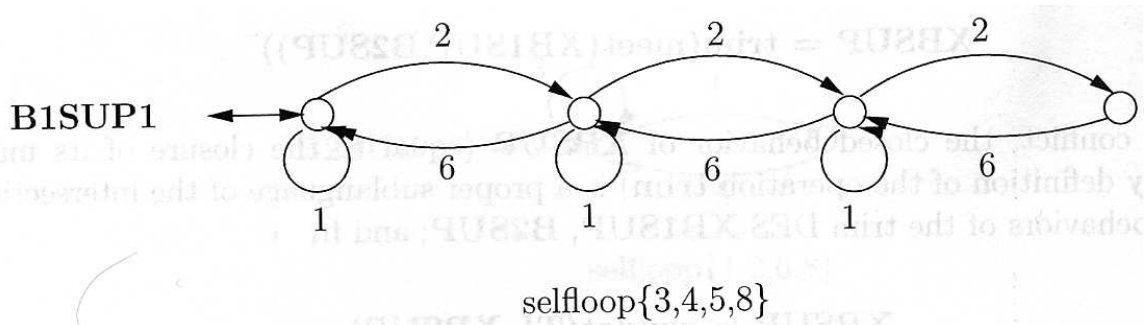
- Substituting into our inequality gives:

$$\#3 - \#6 - \#8 \geq 3 + \#3 - \#2 - \#8$$

- After algebraic simplification, we see we must disable **M1** when:

$$\#2 - \#6 \geq 3$$

- We can represent this as **B1SUP1** below:



Transfer Line Example - VII

- We thus take as our new supervisor:

$$\textbf{BSUP} = \textbf{meet}(\textbf{B1SUP1}, \textbf{B1SUP2}, \textbf{B2SUP})$$

- We can quickly check that **BSUP** is controllable for **TL**, and that the closed loop behavior, **MSUP**, is nonblocking.

$$\textbf{MSUP} = \textbf{meet}(\textbf{TL}, \textbf{BSUP})$$

- We then check and find that **MSUP** is isomorphic with **CSUP**, thus our modular supervisors are optimal.

- Browse through Sections 4.7, and 4.9 on own.