Resource-bound process algebras for Schedulability and Performance Analysis of Real-Time and Embedded Systems

Insup Lee¹, Oleg Sokolsky¹, Anna Philippou²

¹ RTG (Real-Time Systems Group)
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA

² Department of Computer Science
University of Cyprus
Nicosia, CY

Outline

• Real-Time and Embedded systems
• Resource-bound computation
• Resource-bound formalisms
  - ACSR (Algebra of communicating shared resources)
  - Schedulability Analysis Problem
  - PACSR (Probabilistic ACSR)
  - Schedulability analysis for soft real-time systems
  - Design framework for embedded systems
  - P²ACSR (Probabilistic ACSR with power consumption)
  - Scheduling synthesis and parametric schedulability analysis
  - ACSR-VP (ACSR with Value-Passing)

• Conclusions
Real-time, Embedded Systems

• Difficulties
  - Increasing complexity
  - Decentralized
  - Safety critical
  - End-to-end timing constraints
  - Resource constrained
    • Non-functional: power, size, etc.
• Development of trustworthy (i.e., reliable, robust, safe, secure, etc.) embedded software

Properties of embedded systems

• Adherence to safety-critical properties
• Meeting timing constraints
• Satisfaction of resource constraints
• Confinement of resource accesses
• Supporting fault tolerance
• Domain specific requirements
  - Mobility
  - Software configuration
Real-time Behaviors

- Correctness and reliability of real-time systems depend on
  - Functional correctness
  - Temporal correctness
  - End-to-end temporal constraints
- Factors that affect temporal behavior are
  - Synchronization and communication
  - Resource limitations and availability/failures
  - Scheduling algorithms
  - Interaction with physical world
- An integrated framework to bridge the gap between concurrency theory and real-time scheduling

Scheduling Problems

- Priority Assignment Problem
- Schedulability Analysis Problem
  - Compositional analysis
  - Hierarchical system
- Soft timing/performance analysis (Probabilistic Performance Analysis)
- End-to-end Design Problem
  - Parametric Analysis
  - End-to-end constraints, intermediate timing constraints
  - Execution Synchronization Problem
  - Start-time Assignment Problem with Inter-job Temporal Constraints
- Fault tolerance: dealing with failures, overloads
Scheduling Factors

- Static priority vs dynamic priority
  - Cyclic executive, RM (Rate Monotonic)
  - EDF (Earliest Deadline First)
- Priority inversion problem
- Independent tasks vs. dependent tasks
- Single processor vs. multiple processors
- Communication delays
- Uncertainty in execution times
- Resource use tradeoffs
- End-to-end timing requirements

Example: Simple Scheduling Problem

- (period, [e-, e+]), where e- and e+ are the lower and upper bound of execution time, respectively.
- Goal is to find the priority of each job so that jobs are schedulable
- Considering only worst case leads to scheduling anomaly
Let $J_{1,1} > J_{2,1}$ and $J_{2,2} > J_{3,1}$
Consider worst case execution time for all jobs, i.e.,
Execution time $E_{1,1} = 2$, $E_{2,1} = 3$, $E_{2,2} = 2$, $E_{3,1} = 3$

With same priorities, $J_{1,1} > J_{2,1}$ and $J_{2,2} > J_{3,1}$
Let execution time $E_{1,1} = 1$, $E_{3,1} = 1$, $E_{2,2} = 2$, $E_{3,1} = 3$

So with the priority assignment of $J_{1,1} > J_{2,1}$ and $J_{2,2} > J_{3,1}$,
jobs cannot be scheduled and scheduling problems are in general NP-hard
End-to-end Design Problem

- Given a task set with end-to-end constraints on inputs and outputs
  - Freshness from input X to output Y (\(F(Y|X)\)) constraints: bound time from input X to output Y
  - Correlation between input X1 and X2 (\(C(Y|X1,X2)\)) constraints: max time-skew between inputs to output
  - Separation between output Y (\(u(Y)\) and \(l(Y)\)) constraints: separation between consecutive values on a single output Y
- Derive scheduling for every task
  - Periods, offsets, deadlines
  - Priorities
- Meet the end-to-end requirements
- Subject to
  - Resource limitations, e.g., memory, power, weight, bandwidth

Example: Start-time Problem

Start-time Assignment Problem with Inter-job Temporal Constraints

Goal is to statically determine the range of start times for each job so that jobs are schedulable and all inter-job temporal constraints are satisfied.
Example: power-aware RT scheduling

• Dynamic Voltage Scaling allows tradeoffs between performance and power consumption
• Problem is how to minimize power consumption while meeting timing constraints.
• Example: three tasks with probabilistic execution time distribution

<table>
<thead>
<tr>
<th>Task</th>
<th>Worst-case execution time</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>14</td>
</tr>
</tbody>
</table>

Our approach and objectives

• Design formalisms for real-time and embedded systems
  - Resource-bound real-time process algebras
  - Executable specifications
  - Logic for specifying properties
• Design analysis techniques
  - Automated verification techniques
  - Parameterized end-to-end schedulability analysis
• Toolset implementation
Resource-bound computation

- Computational systems are always constrained in their behaviors
- Resources capture physical constraints
- Resources should be supported as a first-class notion in modeling and analysis
- Resource-bound computation is a general framework of wide applicability

Resources

- Resources capture constraints on executions
- Resources can be
  - Serially reusable:
    - processors, memory, communication channels
  - Consumable
    - power
- Resource capacities
  - Single-capacity resources
  - Multiple-capacity resources
  - Time-sliced, etc.
Process Algebras

- Process algebras are abstract and compositional methodologies for concurrent-system specification and analysis.

- "Design methodology which systematically allows to build complex systems from smaller ones" [Milner]

Process Algebras

- A process algebra consists of
  - a set of operators and syntactic rules for constructing processes
  - a semantic mapping which assigns meaning or interpretation to every process
  - a notion of equivalence or partial order between processes
  - a set of algebraic laws that allow syntactic manipulation of processes.

- Ancestors
  - CCS, CSP, ACP,...
  - focus on communication and concurrency
Advantages of Process Algebra

A large system can be broken into simpler subsystems and then proved correct in a modular fashion.

1. A hiding or restriction operator allows one to abstract away unnecessary details.
2. Equality for the process algebra is also a congruence relation; and thus, allows the substitution of one component with another equal component in large systems.
ACSR

- ACSR (Algebra of Communicating Shared Resource)
  - A real-time process algebra which features discrete time, resources, and priorities
  - Timeouts, interrupts, and exception handling
  - Two types of actions:
    - Instantaneous events
    - Timed actions

Events

- Events represent non-time consuming activities
  - events are instantaneous: crash
  - point-to-point synchronization
Events

- Events
  - have priorities: \((\text{job}, 10^{10})\)
  - have input and output capabilities
    or
    \[(e, p_1) (\bar{e}, p_2)
       (e? , p_1) (e! , p_2)\]

Actions

- **Actions** represent activities that
  - take time
  - require access to resources
  - each resource usage has priority of access
    \[A = \{ (r_1, p_1), (r_2, p_2) \}\]
  - each resource can be used at most once
  - resources of action \(A\): \(\rho(A)\)
  - idling action: \(\emptyset\)

- Examples:
  \[\{(\text{cpu}, 2)\}, \{(\text{cpu}_1, 3), (\text{cpu}_2, 4)\}, \{(\text{semaphore}, 5)\}\]
Syntax for ACSR processes

- **Process terms**
  
  \[ P ::= \]
  
  - \( NIL \)
  - \( A : P \)
  - \( (a,n).P \)
  - \( P + P \)
  - \( P \parallel P \)
  - \( P\Delta_i(Q,R,S) \)
  - \([P]_i\)
  - \( P \setminus F \)
  - \( b \rightarrow P \)
  - \( C \)

- **Process names**

\[ C = P \]

---

Constant and Nil

\[ C = P \]

C is a constant that represents the process algebra expression P

\[ P = NIL \]

P does nothing
Prefix Operators

\[ P = A : Q \]  
- P performs timed action A and then behaves as Q

\[ P = (a,n).Q \]  
- P performs event (a,n) and then behaves as Q

**EXAMPLE**

```text
\texttt{def}
Operator = (\texttt{ring},1).(\texttt{pickup},1).\texttt{Talk}
Talk = \{(\texttt{phone},2)\} : (\texttt{hangup},1).\texttt{Operator}
```

Choice

\[ P = Q + R \]  
- P can choose nondeterministically to behave like Q or R

**EXAMPLE**

```
def
CAR = (\texttt{golef} \downarrow).CAR' + (\texttt{goright},1).CAR''
```
Parallel Composition

\[ P = Q \parallel R \]

P is composed by Q and R that may synchronize on events and must synchronize on timed actions.

**EXAMPLE**

\[
\begin{align*}
\text{def} & \quad \text{Operator} = (\text{ring}, \text{?}, 1).\{(\text{phone}, 2)\} \\
& \quad \quad \quad \quad : (\text{hangup}, \text{?}, 1).\text{Operator} \\
\text{def} & \quad \text{Caller} = (\text{ring}, \text{!}, 2).\{(\text{phone}, 3)\} \\
& \quad \quad \quad \quad : (\text{hangup}, \text{!}, 1).\text{Caller} \\
\text{def} & \quad \text{Converse} = \text{Operator} \parallel \text{Caller}
\end{align*}
\]

Scope

\[ P = Q \leftarrow (R, S, T) \]

Q may execute for at most t time units. If message a is produced, control is delegated to R, else control is delegated to S. At any time T may interrupt.

**EXAMPLE**

\[
\begin{align*}
\text{def} & \quad \text{Runner} = \text{Run} \leftarrow \text{finish} (\text{GoForCoffee}, \\
& \quad \quad \quad \quad \text{GoToWork,} \\
& \quad \quad \quad \quad \text{BeepedToWork}) \\
\text{def} & \quad \text{Run} = \{(\text{run}, 1)\} : \text{Run} + \text{finish}.\text{NIL}
\end{align*}
\]
**Hiding/Restriction**

- $P = [Q]_I$  
  P behaves just as Q but resources in I are no longer visible to the environment

- $P = Q \setminus F$  
  P behaves just as Q but labels in F are no longer visible to the environment

**EXAMPLE**

```
Caller || PayPhone || [Home]_phone
```

---

**ACSR semantics**

- *Gives an unambiguous meaning to language expressions.*
- *Semantics is operational, given by a set of semantic rules.*

- Example of a labeled transition system:

  $P_0 \xrightarrow{\emptyset} P_1 \xrightarrow{NC} P_2 \xrightarrow{(gate, train)} P_3 \xrightarrow{(gate, train)} P_4 \xrightarrow{IC} ...$
ACSR semantics

- Two-level semantics:
  - A collection of inference rules gives the unprioritized transition relation:
    \[ P \xrightarrow{\alpha} P' \]
  - A preemption relation on actions and events disables some of the transitions, giving a prioritized transition relation:
    \[ P \xrightarrow{\alpha, \pi} P' \]

Unprioritized transition relation

- Prefix operators
  - \textbf{ActT}:
    \[ A: P \xrightarrow{\delta} P \]
  - \textbf{ActI}:
    \[ (a, p): P \xrightarrow{(a, p)} P \]
- Choice
  - \textbf{ChoiceL}:
    \[ P \xrightarrow{\alpha} P' \]
    \[ P + Q \xrightarrow{\alpha} P' \]
- Parallel
  - \textbf{ParIL}:
    \[ P \xrightarrow{(a, p)} P' \]
    \[ P \parallel Q \xrightarrow{(a, p)} P' \parallel Q \]
Unprioritized transition relation (II)

- Resource-constrained execution

\[ \text{ParT} \quad P \xrightarrow{\Delta} P' \quad Q \xrightarrow{\Delta} Q' \quad P \parallel Q \xrightarrow{\Delta \cup \Delta} P' \parallel Q' \quad \rho(A_1) \cap \rho(A_2) = \emptyset \]

- Priority-based communication

\[ \text{ParCom} \quad P \xrightarrow{(r,r_1)} P' \quad Q \xrightarrow{(r,r_2)} Q' \quad P \parallel Q \xrightarrow{(r,r_1 + r_2)} P' \parallel Q' \]

- Resource closure

\[ \text{CloseT} \quad P \xrightarrow{\Delta} P' \quad [P]_I \xrightarrow{\Delta \cup \Delta} [P']_I \quad A_2 = \{(r,0) \mid r \in I - A_1\} \]

Examples

- Resource conflict

\[ P = \{(r,1)\} : P' \quad Q = \{(r,2)\} : Q' \quad P \parallel Q \sim N:\emptyset \]

- Processes must provide for preemption

\[ P = \{(r,1)\} : P' + \emptyset : P \quad Q = \{(r,2)\} : Q' + \emptyset : Q \]

- Unprioritized transitions:

\[ \emptyset \xrightarrow{\{(r,1)\}} P \parallel Q \xrightarrow{\{(r,2)\}} P' \parallel Q' \]

\[ P \parallel Q \xrightarrow{\{(r,1)\}} P' \parallel Q' \]

\[ P \parallel Q \xrightarrow{\{(r,2)\}} P' \parallel Q' \]
Unprioritized transition relation (III)

\[
\begin{align*}
\text{ScopeCT} & : \quad P \xrightarrow{A} P' \\
& \quad PΔ_i^a(Q, R, S) \xrightarrow{A} P'Δ_i^a(Q, R, S) \\
& \quad (t > 0) \\
\text{ScopeCI} & : \quad P \xrightarrow{e} P' \\
& \quad PΔ_i^e(Q, R, S) \xrightarrow{e} P'Δ_i^e(Q, R, S) \\
& \quad (l(e) \neq a, t > 0) \\
\text{ScopeE} & : \quad P \xrightarrow{(a, \rho)} P' \\
& \quad PΔ_i^e(Q, R, S) \xrightarrow{(a, \rho)} Q \\
& \quad (t > 0) \\
\text{ScopeT} & : \quad R \xrightarrow{\alpha} R' \\
& \quad PΔ_i^\alpha(Q, R, S) \xrightarrow{\alpha} R' \\
& \quad (t = 0) \\
\text{ScopeI} & : \quad S \xrightarrow{\alpha} S' \\
& \quad PΔ_i^\alpha(Q, R, S) \xrightarrow{\alpha} S' \\
& \quad (t > 0)
\end{align*}
\]

Example

- A Scheduler

\[
\text{Sched} = \phi : \text{Sched} + (tc, l)\phi^\infty : \Delta_{\text{max}}^\infty(NIL \text{ kill } \text{Sched}, rc \text{Sched})
\]
Preemption relation

• To take priorities into account in the semantics we define the relation $\alpha$ is preempted by $\beta$: $\alpha < \beta$

• An action $\beta$ preempts an action $\alpha$ iff
  - no lower priorities: $\forall r \in \rho(\alpha), \pi_r(\alpha) \leq \pi_r(\beta)$
  - some higher priorities: $\exists r \in \rho(\beta), \pi_r(\alpha) < \pi_r(\beta)$
  - it contains fewer resources $\rho(\beta) \subseteq \rho(\alpha)$
  e.g. $\{(r_1,3),(r_2,5)\} < \{(r_1,7),(r_2,5)\}$

• An event preempts another event iff
  - same label, higher priority e.g. $(a!,1) < (a!,3)$

• An event preempts an action iff
  - $\tau$ with non-zero priority preempts all actions
  e.g. $\{(r,4)\} < (\tau,1)$

Prioritized transition relation

• We define

$$P \xrightarrow{\alpha}^\pi P'$$

when
- there is an unprioritized transition

$$P \xrightarrow{\alpha} P'$$

- there is no $P \xrightarrow{\beta} P''$ such that $\alpha < \beta$

• Compositional
Example

• Unprioritized and prioritized transitions:

\[ P = \{(r,1)\} : P' + \emptyset : P \quad Q = \{(r,2)\} : Q' + \emptyset : Q \]


Example (cont.)

• Resource closure enforces progress

\[ \{(r,0)\} \]

\[ \{(r,1)\} \]

\[ \{(r,2)\} \]

\[ \{(r,0)\} \]

\[ \{(r,1)\} \]

\[ \{(r,2)\} \]

\[ \{(r,0)\} \]

\[ \{(r,1)\} \]
**Compositionality of preemption relation**

- Given

\[
P_1 = (a,2).S_1 + (b,1).S_2
\]

\[
P_2 = (a,2).S_1
\]

\[
Q_1 = (\overline{a},3).T_1 + (\overline{b},5).T_2
\]

\[
Q_2 = (\overline{a},3).T_1 + (\overline{b},2).T_2
\]

\[
R_1 = (b,2).S_1 + (b,1).S_2
\]

\[
R_2 = (b,2).S_1
\]

- Given \(P_1\) and \(P_2\), can they be treated as equivalent?

That is, for all \(Q\), \(P_1 \parallel Q = P_2 \parallel Q\)?

- How about \(R_1\) and \(R_2\)?

**Bisimulation**

- Observational equivalence is based on the idea that two equivalent systems exhibit the same behavior at their interfaces with the environment.

- This requirement was captured formally through the notion of bisimulation, a binary relation on the states of systems.

- Two states are bisimilar if for each single computational step of the one there exists an appropriate matching (multiple) step of the other, leading to bisimilar states.
Prioritized strong equivalence

- An equivalence relation is congruence when it is preserved by all the operators of the language.
- This implies that replacement of equivalent components in any complex system leads to equivalent behavior.

\[ P \xrightarrow{\alpha} P' \] is a congruence relation with respect to the ACSR operators.

Equational Laws

- Equational laws are a set of axioms on the syntactic level of the language that characterize the equivalence relation.
- They may be used for manipulating complex systems at the level of their syntactic (ACSR) description.
- There is a set of laws that is complete for finite state ACSR processes:

\[
\begin{align*}
P + \text{NIL} &= P \\
P + P &= P \\
P + Q &= Q + P \\
(P \parallel Q) \parallel R &= P \parallel (Q \parallel R)
\end{align*}
\]

...
Equational Laws

- **ACSR-specific laws for scope and resource closure:**

\[
A : P \Delta^e_t (Q,R,S) = A \left( P \Delta^e_t (Q,R,S) \right) + S \quad \text{if } t > 0
\]

\[
e.P \Delta^e_t (Q,R,S) = e \left( P \Delta^e_t (Q,R,S) \right) + S \quad \text{if } t > 0 \land \overline{I(e)} \neq a
\]

\[
e.P \Delta^u_t (Q,R,S) = \left( r, \pi(e) \right) Q + S \quad \text{if } t > 0 \land \overline{I(e)} = a
\]

\[
P \Delta^u_0 (Q,R,S) = \begin{cases} 
R & A_1 : P \\
[ e.P ] & e.P
\end{cases} 
\]

\[
A_2 = \{ (r,0) \mid r \in I - \rho(A_i) \}
\]

---

Laws (1)

- **Choice(1)** \( P + \text{NIL} = P \)
- **Choice(2)** \( P + P = P \)
- **Choice(3)** \( P + Q = Q + P \)
- **Choice(4)** \( (P + Q) + R = P + (Q + R) \)
- **Choice(5)** \( \alpha P + \beta Q = \beta Q \) if \( \alpha < \beta \)
- **Part(1)** \( P \parallel Q = Q \parallel P \)
- **Part(2)** \( (P \parallel Q) \parallel R = P \parallel (Q \parallel R) \)
- **Part(3)** \[
\sum_{A : P, Q : S} (A : B_k : (P_k \parallel R_k)) + \sum_{e_j : Q_j} \left( \sum_{k \in \text{e}_j} B_k : R_k + \sum_{k \in \text{f}_j} S_j \right)
\]

\[
= \sum_{e_j : Q_j} \left( \sum_{k \in \text{e}_j} A_k : P_k + \sum_{k \in \text{f}_j} Q_j \right) + \sum_{f_j : S_j} (r, \pi(e_j) + \pi(f_j)) (Q_j \parallel S_j)
\]
Laws (2)

Scope(1) \( A : R^n_t(Q,R,S) = A : (R^n_{t+1}(Q,R,S)) + S \) if \( t > 0 \)

Scope(2) \( e.R^n_t(Q,R,S) = e.(R^n_{t+1}(Q,R,S)) + S \) if \( t > 0 \) and \( \overline{\tau(e)} \neq b \)

Scope(3) \( e.R^n_t(Q,R,S) = (r,\pi(e)) Q + S \) if \( t > 0 \) and \( \overline{\tau(e)} = b \)

Scope(4) \( R^n_0(Q,R,S) = R \)

Scope(5) \( P_1 + P_2)_t(Q,R,S) = P_1 R^n_t(Q,R,S) + P_2 R^n_t(Q,R,S) \)

Scope(6) \( NIL R^n_t(Q,R,S) = S \) if \( t > 0 \)

Res(1) \( NIL \setminus F = NIL \)

Res(2) \( (P + Q) \setminus F = (P \setminus F) + (Q \setminus F) \)

Res(3) \( (A : P) \setminus F = A : (P \setminus F) \)

Res(4) \( ((a,n),P) \setminus F = (a,n), (P \setminus F) \) if \( a \notin F \)

Res(5) \( ((a,n),P) \setminus F = NIL \) if \( a \notin F \)

Res(6) \( P \setminus E \setminus F = P \setminus E \setminus F \)

Res(7) \( P \setminus \emptyset = P \)

Laws (3)

Close(1) \([NIL]_i = NIL\]

Close(2) \([P + Q]_i = [P]_i + [Q]_i\]

Close(3) \([A_i : P]_i = (A_i \cup A_i) : [P]_i\) where \( A_i = \{(r,0) \mid r \in I - \rho(A_i)\}\)

Close(4) \([e.P]_i = e[P]_i\]

Close(5) \([i[P_i]_i = [P]_{0,i}\]

Close(6) \([P]_g = P\]

Close(7) \([P \setminus E]_i = [P]_i \setminus E\]

Rec(1) \( rec X.P = P[rec X.P / X]\)

Rec(2) If \( P = Q[P / X]\) and \( X \) is guarded in \( Q \) then \( P = rec X.Q\)

Rec(3) \( rec X.(P + \sum_{i \in I} [X \setminus E_i]_{i}) = rec X.(\sum_{i \in I} [P \setminus E_i]_{i})\)

where \( E_j = \bigcup_{i \in I} E_i, U_j = \bigcup_{i \in I} U_i, I \) is finite and \( X \) is guarded in \( P\)
Soundness of the laws

• Theorem:

\[ \text{if } P = Q \text{ then } P \sim_\pi Q \]

• Proof approach:
  - Construct the set of prioritized derivations for each \( P \)
  - Prove that if \( P = Q \), then the sets of derivations are the same

Completeness of the laws

• Theorem:

\[ \text{if } P \text{ and } Q \text{ are finite-state processes and } P \sim_\pi Q \text{ then } P = Q \]
Schedulability Analysis

- Can all real-time tasks meet their deadlines?
- Factors include
  - Delay caused by synchronization between tasks
  - Delay caused by precedence between tasks
  - Delay caused by resource constraints
  - Scheduling disciplines and synchronization protocols
Outline

- ACSR-VP: ACSR with value-passing and dynamic priorities
- Specifying real-time systems using ACSR-VP
  - Specifying task models
  - Specifying scheduling disciplines
- Analyzing real-time systems using bisimulation
  - Specification correctness
  - Schedulability analysis
- Schedulability analysis using VERSA (ACSR Toolkit)

ACSR (Algebra of Communicating Shared Resources)

- A timed process algebra based on CCS with notions of time, resources and priorities
- Discrete time and dense time
- Static priorities
- Actions: Instantaneous Events + Timed Actions
  - Timed action: a set of (resource, priority) pairs
    \( \{(cpu, 4), (data, 3)\}, \{(cpu_1, 2), (cpu_2, 3)\}, \emptyset \)
  - Instantaneous event: (event, priority) pair
    \( (signal, 2), (chan, 2) (\tau, 3) \)
- Real-time operators for timeout, interrupt, exception
- Graphical specification language (GCSR)
- Toolkit (VERSA)
- No value passing communication, no variables for priorities
ACSR-VP (ACSR with Value Passing)

- Extends ACSR with variables and value passing communications
- Values can be specified using expressions
  - Timed Actions:
    \{ (cpu, x), (data, y + 1) \}
  - Instantaneous events:
    \{ signal ! 8, x \} - output
    \{ chan ? y, 2 \} - input
- Dynamic priorities
- Exchange priority information without global variables

ACSR-VP Syntax

\[
P ::= \text{NIL} \quad \text{process that does nothing} \\
| \quad A : P \quad \text{timed action prefix} \\
| \quad e.P \quad \text{instantaneous event prefix} \\
| \quad be \rightarrow P \quad \text{conditional process} \quad (be : \text{boolean expression}) \\
| \quad P_1 + P_2 \quad \text{choice} \\
| \quad P_1 \parallel P_2 \quad \text{parallel composition} \\
| \quad [P]_t \quad \text{resource close} \\
| \quad P \setminus F \quad \text{event restriction} \\
| \quad P \setminus I \quad \text{resource hiding} \\
| \quad C(x) \quad \text{process name defined to be a process} \\
\]

\[ C(x) = P \]
ACSR-VP Example

Preemptable and Non-preemptable Jobs

- Both jobs execute $c$ time units on $cpu$ with priority $\pi$
- Non-preemptable job: once it acquires $cpu$, it executes to completion

$$\begin{align*}
\text{Job}_1 & \overset{\text{def}}{=} \emptyset : \text{Job}_1 + \text{Exec}_1(0) \\
\text{Exec}_1(s) & = (s < c) \rightarrow \{(cpu, \pi)\} : \text{Exec}_1(s + 1)
\end{align*}$$

- Preemptable job: its execution can be preempted by actions on $cpu$ of other jobs with higher priorities

$$\begin{align*}
\text{Job}_2 & \overset{\text{def}}{=} \emptyset : \text{Job}_2 + \text{Exec}_2(0) \\
\text{Exec}_2(s) & = (s < c) \rightarrow \{(cpu, \pi)\} : \text{Exec}_2(s + 1) \\
& + \emptyset : \text{Exec}_2(s)
\end{align*}$$

Unprioritized Operational Semantics

<table>
<thead>
<tr>
<th>Act</th>
<th>A : $P \xrightarrow{\Delta} P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Act1</td>
<td>$(P {x \leftarrow v}, v).P \xrightarrow{(x[v/x])} \mathcal{P} n / x$</td>
</tr>
<tr>
<td>Act2</td>
<td>$(P {v \leftarrow \emptyset}, v).P \xrightarrow{(v[v/\emptyset])} \mathcal{P}$</td>
</tr>
<tr>
<td>Act3</td>
<td>$(\tau, v).P \xrightarrow{\tau[v/\emptyset]} P$</td>
</tr>
<tr>
<td>ParT</td>
<td>$P \xrightarrow{\Delta} P', Q \xrightarrow{\delta} Q' (\rho(A_1) \cap \rho(A_2) = \emptyset)$</td>
</tr>
<tr>
<td>ParC2</td>
<td>$P \xrightarrow{(x[v/\emptyset])} \mathcal{P}, Q \xrightarrow{(x[v/\emptyset])} \mathcal{Q}$</td>
</tr>
</tbody>
</table>
Unprioritized Operational Semantics

\[
\begin{align*}
\text{CloseT} &: P \xrightarrow{\delta \cup \delta_s} P' \\
\text{CloseI} &: P \xrightarrow{\epsilon} P' \\
\text{HideT} &: P \xrightarrow{\delta} P' \\
\text{HideI} &: P \xrightarrow{\epsilon} P'
\end{align*}
\]

\[
\begin{align*}
(A_2 = \{(r,0) \mid r \in I - \rho(A_i)\}) \\
(P \xrightarrow{\delta} P') \quad (\{(r,p) \in A \mid r \notin I\})
\end{align*}
\]

Preemption

A preemption relation is defined for two any actions \( \alpha \) and \( \beta \), denoted \( \alpha \prec \beta \), read \( \beta \) preempts \( \alpha \).

Examples:
- \( \{(r_1,2), (r_2,5)\} \prec \{(r_1,7), (r_2,5)\} \)
- \( \{(r_1,2), (r_2,5)\} \not\prec \{(r_1,7), (r_2,3)\} \)
- \( \{(r_1,2), (r_2,0)\} \prec \{(r_1,7)\} \)
- \( \{(r_1,2), (r_2,1)\} \not\prec \{(r_1,7)\} \)
- \( (a,2) \prec (a,5) \)
- \( (a,1) \not\prec (b,2) \)
- \( (\tau,1) \prec (\tau,2) \)
- \( \{(r_1,2), (r_2,5)\} \prec (\tau,2) \)
Prioritized Operational Semantics

The operational semantics of ACSR-VP, the prioritized transition relation $\xrightarrow{\pi,\alpha}$, is defined as follows:

$P \xrightarrow{\alpha} P'$ iff

1. $P \xrightarrow{\alpha} P''$
2. there is no $P \xrightarrow{\beta} P'''$ such that $\alpha < \beta$

\[\text{Example: } P = \{(cpu,2)\} : P_1 + \{(cpu,3)\} : P_2 \]

- Unprioritized transition:
  \[\begin{align*}
P & \xrightarrow{\{(cpu,2)\}} P_1 \\
P & \xrightarrow{\{(cpu,3)\}} P_2
\end{align*}\]

- Prioritized transition:
  \[\begin{align*}
P & \xrightarrow{\{(cpu,3)\}\pi} P_2
\end{align*}\]

Modeling a Real-Time System

- A real-time system consists of a set of tasks running in parallel under a specific scheduling discipline.
- A task is a process composed of a sequence of jobs executed serially.
  - A task can be
    - Independent or dependent
    - Preemptable or non-preemptable
    - Periodic or aperiodic
- Possible timing constraints of a task are:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Starting time</td>
</tr>
<tr>
<td>c, d</td>
<td>Execution time and deadline</td>
</tr>
<tr>
<td>p</td>
<td>Period for periodic task</td>
</tr>
<tr>
<td>$p_1, p_2$</td>
<td>Minimum and maximum inter - arrival times for aperiodic task</td>
</tr>
</tbody>
</table>
Specification of a real-Time System

A real-time system is specified by the process $\text{RTS}$:

$$\text{RTS} = [\text{Job}_1 \parallel \text{Activator}_1 \parallel \cdots \parallel \text{Job}_n]$$

Tasks are specified by the processes $\text{Job}_i$:

$$\text{Job}_i = (\text{Activator}_i \parallel \text{Activator}_i) \setminus \{\text{start, end}\}$$

- Process $\text{Job}_i$: internal characteristics, e.g.,
  - resource requirements
  - synchronization
- Process $\text{Activator}_i$: external timing attributes, e.g.,
  - periodic or aperiodic
  - period and deadline
- Events $\text{start, end}$ are synchronization events:
  - $\text{start}$: activate jobs
  - $\text{end}$: mark deadlines of jobs – deadlock if unsuccessful

Sample Activators

Activator 1. A periodic task with $(b, d, p)$

$$\text{Activator} = \emptyset^b : \text{Activator}'$$
$$\text{Activator}' = (\text{start}!, 1)\emptyset^d : (\text{end}! , 2).$$
$$\emptyset^{p-d} : \text{Activator}'$$

Activator 2. An aperiodic task with $(b, d, p_1, p_2)$

$$\text{Activator} = \emptyset^b : \text{Activator}'$$
$$\text{Activator}' = (\text{start}!, 1)\emptyset^d : (\text{end}! , 2).$$
$$\emptyset^{p_1-d, p_2-d} : \text{Activator}'$$

where

$$\emptyset^n = \emptyset : \cdots : \emptyset$$  \hspace{1cm} (idling for $n$ time units)

$$\emptyset^{m-n} = \emptyset^m + \emptyset^{m+1} + \cdots + \emptyset^n$$
Sample Jobs

Job 1
- preemptable, independent jobs running on cpu
  priority $\pi$ and execution time $c$:
  \[
  \text{Job} \overset{\text{def}}{=} \emptyset : \text{Job} + (\text{start},1)\cdot\text{Exec}(0,0) \\
  \text{Exec}(s,t) = \begin{cases} 
  (s < c) \rightarrow ((c\text{pu},\pi)) : \text{Exec}(s+1,t+1) \\
  (s = c) \rightarrow \text{Wait}
  \end{cases} \\
  \text{Wait} \overset{\text{def}}{=} \emptyset : \text{Wait} + (\text{end},1)\cdot\text{Job}
  \]
  - $s$ for accumulated execution time
  - $t$ for the elapsed time
  - Job can response to end event only when its current execution is finished

Sample Jobs

Job 2
- nonpreemptable, independent jobs on multiprocessors $c\text{pu}_1, \ldots, c\text{pu}_k$
  with priorities $\pi_1, \ldots, \pi_k$ and execution time $c$:
  \[
  \text{Job} \overset{\text{def}}{=} \emptyset : \text{Job} + (\text{start},1)\cdot\text{Exec} \\
  \text{Exec} = \sum_{1 \leq i \leq k} \{(c\text{pu}_i,\pi_i)^c : \text{Wait}) \\
  \text{Wait} \overset{\text{def}}{=} \emptyset : \text{Wait} + (\text{end},1)\cdot\text{Job}
  \]
  - A job can be executed on any of the processors
  - Once a processor is assigned to a job, the job executes on that processor until completion
Sample Jobs

Job 3

• dependent jobs on processor cpu with priority \( \pi \) and execution time \( c \) on resource data (with priority \( \pi' \) after at \( c' \) time units execution:

<table>
<thead>
<tr>
<th>Event</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>( \preceq ) : Job + ((start),1).Exec(0,0)</td>
</tr>
</tbody>
</table>
| Exec | \( s < c \wedge (s+c') \rightarrow (\text{cpu,},\pi) \) : Exec(s+1,t+1) + \( s+c' \rightarrow (p,0).CS(x,t) \)
| Wait | \( s = c \rightarrow \text{Wait} \) |
| CS(x,t) | \( s < c+cs \rightarrow ((\text{cpu,},\pi) : \text{CS}(s+1,t+1) + \text{Wait} : \text{CS}(s,t+1)) \)
| P | \( \text{P?} \) V \( \text{?} \) V \( \text{?} \) V |
| V | \( \text{V?} \) P \( \text{?} \) V \( \text{?} \) V |

- P and V operations are modeled by the processes P and V with events (\(p?,0\)) and (\(v?,0\))
- When \( s \) equals \( c' \), Exec waits for (\(p?,0\)) to enter the critical section CS(s,t)

Scheduling Disciplines

Earliest Deadline First

• Tasks \( T_i \) = Job 1 + Activator 1
• Priority \( \pi_i = d_{max} - (d_i - t) \)

where \( d_{max} = (1 \max (d_1, \ldots, d_n)) \)

<table>
<thead>
<tr>
<th>Event</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDFSys</td>
<td>( T_1 \parallel T_2 \parallel \cdots \parallel T_{\text{num}} )</td>
</tr>
<tr>
<td>Job_i</td>
<td>( \varnothing : \text{Job}_i + (\text{start},1).\text{Exec}(0,0) )</td>
</tr>
</tbody>
</table>
| Exec(s,t) | \( s < c \rightarrow ((\text{cpu,},d_{max} - (d_i - t)) : \text{Exec}(s,t+1) + \text{Wait} : \text{Exec}(s,t+1)) \)
| Wait_i | \( \varnothing : \text{Wait}_i + (\text{end},1).\text{Job}_i \) |
| Activator | \( (\text{start},1) \parallel \varnothing : (\text{end},2) \parallel \varnothing : \text{Activator} \) |
Other Time-Driven Scheduling Disciplines

<table>
<thead>
<tr>
<th>Discipline</th>
<th>( \pi_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline Monotonic</td>
<td>( \pi_i = d_{\text{max}} - d_i )</td>
</tr>
<tr>
<td>Shortest Remaining Time First</td>
<td>( \pi_i = c_{\text{max}} - (c_i - s) )</td>
</tr>
<tr>
<td>Least Laxity First</td>
<td>( \pi_i = d_{\text{max}} - (d_i - r) - (c_i - s) )</td>
</tr>
</tbody>
</table>

where \( c_{\text{max}} = 1 + \max \{c_1, \ldots, c_n\} \)

The Priority Inversion Problem

Without priority inheritance

With priority inheritance

5/27/08 Korea University
**Task parameters**

<table>
<thead>
<tr>
<th>Resources:</th>
<th>cpu</th>
<th>processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ready time</td>
<td>$r_1 = 5$</td>
<td>$r_2 = 10$</td>
</tr>
<tr>
<td>comp. time</td>
<td>$c_1 = 6$</td>
<td>$c_2 = 8$</td>
</tr>
<tr>
<td>deadline</td>
<td>$d_1 = 30$</td>
<td>$d_2 = 30$</td>
</tr>
<tr>
<td>start time of CS</td>
<td>$cs_1 = 3$</td>
<td>$cs_2 = 5$</td>
</tr>
<tr>
<td>length of CS</td>
<td>$c'_1 = 2$</td>
<td>$c'_2 = 2$</td>
</tr>
<tr>
<td>priority</td>
<td>$\pi_1 = 3$</td>
<td>$\pi_2 = 2$</td>
</tr>
<tr>
<td>max priority</td>
<td>$\pi_{max} = 4$</td>
<td></td>
</tr>
</tbody>
</table>

**Priority Inheritance Protocol**

$$T_c = \text{Job} \equiv \text{Activator1 + Priority - Passing Events}$$

$$\text{PISys} = \{\text{Job} \equiv \text{Activator0, } (\text{start, end})\}$$

$$\text{Job}_0 = \emptyset : \text{Job}_0 = (\text{start, end}) : \text{Job}_0$$

$$\text{Exec}_p(s) = (s < c, s < cs) \rightarrow ((\text{cpu, run}) : \text{Exec}(s + 1))$$

$$\text{Wait} = \emptyset : \text{Wait} = (\text{stop, start}) : \text{Job}_0$$

$$\text{Req}_p(s) = (p < c, s < cs) \rightarrow ((\text{cpu, run}) : \text{Req}(s + 1))$$

$$\text{CS}(s, c) = (s < c, s < cs) \rightarrow ((\text{cpu, run}) : \text{CS}(s + 1))$$

$$\text{Activator} = \emptyset : (\text{start}) : \emptyset : (\text{end, stop}) : \emptyset$$

$$\text{P} = (p < c, s, \text{max}) \rightarrow (p < c, s, \text{max})$$

$$\text{V}(\text{max}) = (c, s) : \text{P} : \text{V}(\text{max})$$

Parameters of $T_c$
- Priority $c$
- Execution time of a job $c_s$
- Time for entering critical section $c'_s$
- Execution time in critical section $c'_s$
Traces of tasks

<table>
<thead>
<tr>
<th>Time</th>
<th>Process 1</th>
<th>Process 2</th>
<th>Process 3</th>
<th>Process P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>P</td>
</tr>
</tbody>
</table>

| 0    | [x]       | [y]       | [z]       | [P]       |
| 1    | [y]       | [z]       | [x]       | [P]       |
| 2    | [z]       | [x]       | [y]       | [P]       |
| 3    | [x]       | [y]       | [z]       | [P]       |
| 4    | [y]       | [z]       | [x]       | [P]       |
| 5    | [z]       | [x]       | [y]       | [P]       |
| 6    | [x]       | [y]       | [z]       | [P]       |
| 7    | [y]       | [z]       | [x]       | [P]       |
| 8    | [z]       | [x]       | [y]       | [P]       |
| 9    | [x]       | [y]       | [z]       | [P]       |
| 10   | [y]       | [z]       | [x]       | [P]       |
| 11   | [z]       | [x]       | [y]       | [P]       |
| 12   | [x]       | [y]       | [z]       | [P]       |
| 13   | [y]       | [z]       | [x]       | [P]       |
| 14   | [z]       | [x]       | [y]       | [P]       |
| 15   | [x]       | [y]       | [z]       | [P]       |
| 16   | [y]       | [z]       | [x]       | [P]       |
| 17   | [z]       | [x]       | [y]       | [P]       |
| 18   | [x]       | [y]       | [z]       | [P]       |
| 19   | [y]       | [z]       | [x]       | [P]       |
| 20   | [z]       | [x]       | [y]       | [P]       |
| 21   | [x]       | [y]       | [z]       | [P]       |
| 22   | [y]       | [z]       | [x]       | [P]       |
| 23   | [z]       | [x]       | [y]       | [P]       |
| 24   | [x]       | [y]       | [z]       | [P]       |
| 25   | [y]       | [z]       | [x]       | [P]       |
| 26   | [z]       | [x]       | [y]       | [P]       |

*: in critical section

Weak Bisimulation

**Def.** If $t \in D^*$, then $\bar{t} \in (D - \{\tau\})^*$ is the sequence derived by deleting all occurrences of $\tau$ from $t$.

**Def.** If $t = \alpha_1 \ldots \alpha_n \in D^*$, then $E \bar{t} E'$ if

$$P(\bar{t}) \rightarrow^* \alpha_1 \rightarrow^*(\bar{t}) \rightarrow^* \ldots \rightarrow^* (\bar{t}) \rightarrow^* \alpha_n \rightarrow^*(\bar{t}) \rightarrow^* P',$$

where "_" in $(\bar{t}, \_)$ represents arbitrary integer.

**Def.** For a given transition system $\rightarrow_s$, any binary relation $r$ is a weak bisimulation if, for $(P, Q) \in r$ and for any action $\alpha \in D$,

1. if $P \rightarrow_s^+ P'$, then, for some $Q', Q \rightarrow_s Q'$ and $(P', Q') \in r$, and

2. if $Q \rightarrow_s Q'$, then, for some $P', P \rightarrow_s P'$ and $(P', Q') \in r$.

**Def.** $\rightarrow_s$ is the largest weak bisimulation over $\rightarrow_s$. It is an equivalence relation (though not a congruence) for ACSR.
Analyzing Real-Time Systems in ACSR-VP

• Two types of analyses
  – Validation
  – Schedulability analysis

• Basic idea
  – Checking weak bisimulation \( \approx \)
  – Searching deadlocked states

• Practical Issues
  – Ensure that the EDFSys and PIPSys processes are finite state
  – Translate ACSR-VP processes to ACSR processes and use VERSA, the toolkit for ACSR

Validating the EDFSys Specification

Construct a correctness specification, EDFSpec, that is sequential and easy to inspect

Verify that \( \text{EDFSys} \approx \pi \text{EDFSpec} \)

\[
\text{EDFSpec} = [S(0, \ldots, 0, 0)]_{c}^{def}
\]

\[
S(s_1, t_1, \ldots, s_n) = \sum_{\text{trans}} \left\{ \begin{array}{l}
  (s_i = c_i \land t_i = p_i) \\
  \rightarrow (r, 1).S(\ldots, s_{i-1}, t_{i-1}, 0, 0, s_{i+1}, t_{i+1}, \ldots) \\
  +
  \ (s_i < c_i \land t_i = d_i) \\
  \rightarrow (r, 1).\text{NIL} \\
  +
  \ (s_i = c_i \land t_i < p_i) \\
  \rightarrow \emptyset : S(\ldots, s_{i-1}, t_{i-1} + 1, s_i, t_i + 1, s_{i+1}, t_{i+1} + 1, \ldots) \\
  +
  \ (s_i < c_i \land t_i < d_i) \\
  \rightarrow \{(cpu, d_{\max} - (d_i - t)) \} : S(\ldots, s_{i-1}, t_{i-1} + 1, s_i + 1, t_i + 1, s_{i+1}, t_{i+1} + 1, \ldots)
\end{array} \right.
\]
Schedulability Analysis

Lemma 1 If $\textbf{EDFSys}$ is deadlock free, then it is schedulable.

Lemma 2 If

$\textbf{EDFSys} \setminus \{cpu\} = \emptyset$,  

then $\textbf{EDFSys}$ is deadlock free.

Lemma 3 If $\textbf{PIPSys}$ is deadlock free, then it is schedulable.

Lemma 4 If

$\textbf{PIPSys} \setminus \{cpu\} = \emptyset$,  

then $\textbf{PIPSys}$ is deadlock free.

Example 1

• Consider an instance $\textbf{EDFSys}_1$ of $\textbf{EDFSys}$ where:
  
  Task $T_1$: $c_1 = 1, d_1 = 2, p_1 = 3$
  Task $T_2$: $c_2 = 2, d_2 = 3, p_2 = 3$

• The following sufficient condition for schedulability from [Liu and Lay 73] is not satisfied:

$\frac{c_1}{d_1} + \frac{c_2}{d_2} \leq 1$

• The following equation

$\textbf{EDFSys} \setminus \{cpu\} = \emptyset$,  

is satisfied, i.e., the task system is schedulable.

More specifically, we have

$\textbf{EDFSys}_1 \xrightarrow{(r,2)} \pi \xrightarrow{(r,2)} \pi \xrightarrow{((cpu,2))} \pi \xrightarrow{((cpu,3))} \pi \xrightarrow{(r,3)} \pi \xrightarrow{((cpu,3))} \pi \xrightarrow{(r,3)} \pi \xrightarrow{\pi} \textbf{EDFSys}_1$
Example 2

Consider another instance EDFSys\textsubscript{2} of EDFSys where:

- Task T\textsubscript{1}: c\textsubscript{1} = 2, d\textsubscript{1} = 2, p\textsubscript{1} = 3
- Task T\textsubscript{2}: c\textsubscript{2} = 2, d\textsubscript{2} = 3, p\textsubscript{2} = 3

The equivalence

\[
\text{EDFSys}_{2} \setminus \{\text{cpu}\} =_{\pi} \emptyset^{\omega},
\]

is false and the task system is therefore not schedulable.

More specifically, we have

\[
\begin{align*}
\text{EDFSys}_{2} &\xrightarrow{(\tau, 2)} \pi \xrightarrow{(\tau, 2)} \pi \xrightarrow{(\text{cpu}, 2)} \pi \xrightarrow{(\text{cpu}, 2)} \pi \xrightarrow{(\tau, 3)} \pi \\
&\xrightarrow{(\text{cpu}, 3)} \pi \text{NIL}
\end{align*}
\]

Summary

- The ACSR paradigm:
  - Formalism for modular specification of real-time systems along with scheduling disciplines
  - Formal characterization of the schedulability analysis in process algebra
- Automated schedulability analysis
  - Provide techniques for detecting timing anomalies before an implementation is developed
  - Integrate into a methodology for engineering reliable real-time systems
- Tools:
  - GCSR (Graphical ACSR)
  - XVERSA: VERSA and GCSR