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

Insup Lee¹, Oleg Sokolsky¹, Anna Philippou²

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

²Department 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 reliable and robust 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 depends on
  - Functional correctness
  - Temporal correctness
- Factors that affect temporal behavior are
  - Synchronization and communication
  - Resource limitations and availability/failures
  - Scheduling algorithms
  - End-to-end temporal constraints
- An integrated framework to bridge the gap between concurrency theory and real-time scheduling

Scheduling Problems

- Priority Assignment Problem
- Schedulability Analysis Problem
- 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

- \((\text{period, } [e-, e+])\), 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
Example (2)

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 $E_{1,1} = 2$, $E_{2,1} = 3$, $E_{2,2} = 2$, $E_{3,1} = 3$

Example (3)

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

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 \textit{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]

- 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 \((e, p_1) \ (e, p_2)\)
    \((e, p_1) \ (e, p_2)\)

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}, 3), (\text{cpu}, 4)),
  ((\text{semaphore}, 5))\}
Syntax for ACSR processes

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

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**

\[
def \text{Operator} = (\text{ring},1), (\text{pickup},1), \text{Talk} \\
\text{Talk} = \{(\text{phone},2) \}: (\text{hangup},1), \text{Operator}
\]

Choice

P = Q+R
P can choose nondeterministically to behave like Q or R

**EXAMPLE**

\[
def \text{CAR} = (\text{goleft},1), \text{CAR}' + (\text{goright},1), \text{CAR}''
\]
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**

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

Scope

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

\( Q \) 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**

\[
\begin{align*}
\text{Runner} & = \text{Run} \left( \text{Delta}^{\text{finish}}_{10} \right) \left( \text{GoForCoffee, GoToWork, BeepedToWork} \right) \\
\text{Run} & = \{(\text{run},1)\} : \text{Run} + \text{finish!}.\text{NIL}
\end{align*}
\]
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**

\[ \text{Caller} \parallel \text{PayPhone} \parallel [\text{Home}]_{\text{phone}} \]

ACSR semantics

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

\begin{align*}
\text{ACSR specification} & \rightarrow \text{Semantic rules} & \rightarrow \text{Labeled transition system}
\end{align*}

- **Example of a labeled transition system:**

\[ P_0 \xrightarrow{\emptyset} P_1 \xrightarrow{NC} P_2 \xrightarrow{\{\text{gate, train}\}} P_3 \xrightarrow{\{\text{gate, train}\}} P_4 \xrightarrow{IC} \ldots \]
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**
  \[
  \text{ActT} \quad A: P \xrightarrow{\alpha} P \\
  \text{ActI} \quad (a, p): P \xrightarrow{(a, p)} P
  \]

- **Choice**
  \[
  \text{ChoiceL} \quad P \xrightarrow{\alpha} P' \\
  \]
  \[
  P + Q \xrightarrow{\alpha} P' \\
  \]

- **Parallel**
  \[
  \text{ParIL} \quad P \xrightarrow{(a, p)} P' \\
  \]
  \[
  P \parallel Q \xrightarrow{(a, p)} P' \parallel Q
  \]
Unprioritized transition relation (II)

- Resource-constrained execution
  \[ \text{ParT} \quad P \xrightarrow[A_i \cup A_j]{} P' \quad Q \xrightarrow[A_i \cup A_j]{} Q' \quad \rho(A_i) \cap \rho(A_j) = \emptyset \]

- Priority-based communication
  \[ \text{ParCom} \quad P \xrightarrow[(p, r)]{} P' \quad Q \xrightarrow[(p, r)]{} Q' \]

- Resource closure
  \[ \text{CloseT} \quad P \xrightarrow[A_i \cup A_j]{} P' \quad \text{for } r = \{ \{r, 0\} | r \in I - A_i \} \]

Examples

- Resource conflict
  \[ P = \{(r, 1)\} : P' \quad Q = \{(r, 2)\} : Q' \quad P \parallel Q \sim \text{NIL} \]

- Processes must provide for preemption
  \[ P = \{(r, 1)\} : P + \emptyset : P \quad Q = \{(r, 2)\} : Q + \emptyset : Q \]

- Unprioritized transitions:
  \[
  \begin{array}{c}
  \emptyset \\
  \{(r, 1)\} \\
  P' \parallel Q \\
  \{(r, 2)\} \\
  P \parallel Q \\
  \end{array}
  \]
Unprioritized transition relation (III)

\[
\begin{align*}
\text{ScopeCT} & \quad \frac{P \xrightarrow{A} P'}{P' \Delta_{r-1}^e(Q, R, S)} \quad (t > 0) \\
\text{ScopeCI} & \quad \frac{P \xrightarrow{c} P'}{P' \Delta_{r}^e(Q, R, S)} \quad (l(e) \neq a, t > 0) \\
\text{ScopeE} & \quad \frac{P \xrightarrow{(a, a)} P'}{P' \Delta_{r}^e(Q, R, S)} \quad (t > 0) \\
\text{ScopeT} & \quad \frac{R \xrightarrow{\alpha} R'}{P \Delta_{r}^e(Q, R, S) \xrightarrow{\alpha} R'} \quad (t = 0) \\
\text{ScopeI} & \quad \frac{S \xrightarrow{\alpha} S'}{P \Delta_{r}^e(Q, R, S) \xrightarrow{\alpha} S'} \quad (t > 0)
\end{align*}
\]

Example

• A Scheduler

\[
Sched = \phi : Sched + (tc, l).\phi^e \Delta^v_{t_{\text{max}}} (NIL, kill .Sched, rc.Sched)
\]

\[
\begin{align*}
Sched & \rightarrow \phi^e : \Delta^v_{t_{\text{max}}} (\ldots) \rightarrow \phi^e : \Delta^v_{t_{\text{max}}-1} (\ldots) \rightarrow \ldots \rightarrow \phi^e : \Delta^v_{0} (\ldots) \\
\emptyset & \xrightarrow{rc} Sched \xrightarrow{rc} Sched \xrightarrow{\text{kill}} Sched
\end{align*}
\]
Preemption relation

- To take priorities into account in the semantics we define the relation \( \alpha \text{ is preempted by } \beta : \alpha < \beta \).
- An action \( \alpha \) preempts action \( \beta \) 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
  \[ P \xrightarrow{\alpha} P' \]
  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
Compositionality of preemption relation

Given

\[ P_1 = (a,2).S_1 + (b,1).S_2 \]
\[ P_2 = (a,2).S_1 \]
\[ Q_1 = (\overline{a},3).T_1 + (\overline{b},5).T_2 \]
\[ Q_2 = (\overline{a},3).T_1 + (\overline{b},2).T_2 \]
\[ R_1 = (a,2).S_1 + (a,1).S_2 \]
\[ R_2 = (a,2).S_1 \]

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 \( R_2 \)?

\begin{itemize}
  \item This requirement was captured formally through the notion of bisimulation, a binary relation on the states of systems.
  \item Observational equivalence is based on the idea that two equivalent systems exhibit the same behavior at their interfaces with the environment.
  \item 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.
\end{itemize}
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.
- Strong bisimulation over $P \xrightarrow{\alpha} P'$ 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:

$$
P + NIL = P \quad P + P = P \quad (P \parallel Q) \parallel R = P \parallel (Q \parallel R)
$$

$$
P + Q = Q + P \quad P + P = P
$$

...
Equational Laws

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

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

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

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

\[
P^t \Delta^t_0 (Q, R, S) = \left[ \left( A : P \right) Inc \right] = \left( A \cup A_r : P \right) \quad \text{if } A_r = \{ (r, 0) \mid r \in I - \rho(A_r) \}
\]

3 October 2003

---

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)**

\[
\begin{align*}
& \left( \sum_{i} A_i : P + \sum_{i} e_i(Q_i) \right) \parallel \left( \sum_{i} B_i : R_i + \sum_{i} f_i : S_i \right) \\
= & \left[ \sum_{i} (A_i : B_i) : \left( P_i \parallel R_i \right) \right] \\
& + \sum_{i} e_i(Q_i) \parallel \left( \sum_{i} B_i : R_i + \sum_{i} f_i : S_i \right) \\
& + \sum_{i} f_i \parallel \left( \sum_{i} A_i : P_i + \sum_{i} e_i(Q_i) \right) \parallel S_i \\
& + \sum_{i} (\tau, \pi(e_i) + \pi(f_i))(Q_i) \parallel S_i
\end{align*}
\]

3 October 2003
Laws (2)

Scope(1) \[ A : PA^*_t(Q,R,S) = A : (PA^*_{t-1}(Q,R,S)) + S \quad \text{if } t > 0 \]
Scope(2) \[ e.PA^*_t(Q,R,S) = e : (PA^*_{t-1}(Q,R,S)) + S \quad \text{if } t > 0 \wedge \{e\} \neq b \]
Scope(3) \[ e.PA^*_t(Q,R,S) = (r, \pi(e))Q + S \quad \text{if } t > 0 \wedge \{e\} = b \]
Scope(4) \[ PA^*_t(Q,R,S) = R \]
Scope(5) \[ P_1 + P_2 \Delta_t(Q,R,S) = P_1 \Delta_t(Q,R,S) + P_2 \Delta_t(Q,R,S) \]
Scope(6) \[ \text{NIL}^*_t(Q,R,S) = S \quad \text{if } t > 0 \]
Rest(1) \[ \text{NIL} \setminus F = \text{NIL} \]
Rest(2) \[ (P + Q) \setminus F = (P \setminus F) + (Q \setminus F) \]
Rest(3) \[ (A : P) \setminus F = A : (P \setminus F) \]
Rest(4) \[ ((a,n)P) \setminus F = (a,n)(P \setminus F) \quad \text{if } a,\overline{a} \notin F \]
Rest(5) \[ ((a,n)P) \setminus F = \text{NIL} \quad \text{if } a,\overline{a} \in F \]
Rest(6) \[ P \setminus E \setminus F = P \setminus (E \cup F) \]
Rest(7) \[ P \setminus \emptyset = P \]

3 October 2003

ESSES 2003

49

Laws (3)

Close(1) \[ [\text{NIL}]_i = \text{NIL} \]
Close(2) \[ [P + Q]_i = [P]_i + [Q]_i \]
Close(3) \[ [(A : P)]_i = (A \cup A_i) : [P]_i \quad \text{where } A_i = \{(r,0) \mid r \in I - \rho(A)\} \]
Close(4) \[ [eP]_i = e [P]_i \]
Close(5) \[ [[P]_i]_j = [P]_{i,j} \]
Close(6) \[ [P]_b = P \]
Close(7) \[ [P \setminus E]_i = [P]_i \setminus E \]
Rec(1) \[ rec X.P = P \{rec X.P / X\} \]
Rec(2) \[ \text{If } P = Q(P / X) \text{ and } X \text{ is guarded in } Q \text{ then } P = rec X.Q \]
Rec(3) \[ rec X.(P + \sum_{t \in T} X \setminus E_t)_{i,j} = rec X.(\sum_{t \in T} [P \setminus E_t]_{i,j}) \]
\[ \text{where } E_j = \bigcup_{i \in I} E_{i,j} = \bigcup_{j \in J} U_{i,j}, I \text{ is finite and } X \text{ is guarded in } P \]

3 October 2003

ESSES 2003

50
Soundness of the laws

• Theorem:
  \[ \text{if } P = Q \text{ then } P \sim_x Q \]

• 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:
  \[ \text{if } P \text{ and } Q \text{ are finite-state processes and } P \sim_x Q \text{ then } P = Q \]
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

\[
P ::= \begin{align*}
  \text{NIL} & : \quad \text{process that does nothing} \\
  A : P & : \quad \text{timed action prefix} \\
  e.P & : \quad \text{instantaneous event prefix} \\
  be \rightarrow P & : \quad \text{conditional process} \\
  P_1 + P_2 & : \quad \text{choice} \\
  P_1 | P_2 & : \quad \text{parallel composition} \\
  [P]_r & : \quad \text{resource close} \\
  P \setminus F & : \quad \text{event restriction} \\
  P \setminus I & : \quad \text{resource hiding} \\
  C(x) & : \quad \text{process name defined to be a process} \\
  C(x) = P & 
\end{align*}
\]
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
  \[ Job_1 \overset{\text{def}}{=} \varnothing : Job_1 + \text{Exec}_1(0) \]
  \[ \text{Exec}_1(s) = (s < c) \rightarrow \{(cpu, \pi)\} : \text{Exec}_1(s + 1) \]

- Preemptable job: its execution can be preempted by actions on $cpu$ of other jobs with higher priorities
  \[ Job_2 \overset{\text{def}}{=} \varnothing : Job_2 + \text{Exec}_2(0) \]
  \[ \text{Exec}_2(s) = (s < c) \rightarrow \{(cpu, \pi)\} : \text{Exec}_2(s + 1) \]
  + $\varnothing : \text{Exec}_2(s)$

Unprioritized Operational Semantics

- $A : P \xrightarrow{\Delta} P$
- $\text{Act}1: (l)(x), v \rightarrow P \xrightarrow{l[x/v]} P[n/x]$
- $\text{Act}2: (l)(v_2), v_1 \rightarrow P \xrightarrow{l[v_1/v_2]} P$
- $\text{Act}3: (\tau, v) \rightarrow P \xrightarrow{\tau[v]} P$
- $\text{ParT}: P \xrightarrow{\Delta} P', Q \xrightarrow{\Delta} Q' \rightarrow P \parallel Q' \rightarrow P \parallel Q'$
  $(\rho(A_1) \cap \rho(A_2) = \varnothing)$
- $\text{ParC2}: P \xrightarrow{(l)(x)/v} P', Q \xrightarrow{(l)(x)/v} Q' \rightarrow P \parallel Q' \xrightarrow{(l)(x)/v} P' \parallel Q'$

3 October 2003  ESSES 2003  59

3 October 2003  ESSES 2003  60
Preemption

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

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

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

\[ P \overset{\alpha}{\rightarrow}_\pi P' \text{ iff } (1) \ P \overset{\alpha}{\rightarrow} P', \ \text{ and } (2) \text{ there is no } P \overset{\beta}{\rightarrow} P'' \text{ such that } \alpha < \beta \]

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

- Unprioritized transition: \[
\begin{align*}
P & \overset{(\text{cpu},2)}{\rightarrow} P_1 \\
P & \overset{(\text{cpu},3)}{\rightarrow} P_2
\end{align*}
\]
- Prioritized transition: \[
\begin{align*}
P & \overset{(\text{cpu},2)}{\rightarrow}_\pi P_1 \\
P & \overset{(\text{cpu},3)}{\rightarrow}_\pi P_2
\end{align*}
\]

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>b, c, d, p</th>
<th>Starting time, Execution time and deadline, Period for periodic task, Minimum and maximum inter-arrival times for aperiodic task</th>
</tr>
</thead>
</table>

3 October 2003 ESSES 2003 63
Specification of a real-Time System

A real-time system is specified by the process RTS:

\[ \text{RTS} \stackrel{\text{def}}{=} [T_1 \parallel T_2 \parallel \cdots \parallel T_n] \]

Tasks are specified by the processes \( T_i \):

\[ \text{T}_i \stackrel{\text{def}}{=} (\text{Job}_i \parallel \text{Activator}_i) \setminus \{\text{start}, \text{end}\} \]

- 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 \( \text{start}, \text{end} \) are synchronization events:
  - \( \text{start} \): activate jobs
  - \( \text{end} \): mark deadlines of jobs – deadlock if unsuccessful

Sample Activators

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

\[
\text{Activator} \overset{\text{def}}{=} \varnothing^b : \text{Activator}' \\
\text{Activator}' \overset{\text{def}}{=} (\text{start}!,1)\varnothing^d : (\text{end}!,2).
\]

\(\varnothing^p : \text{Activator}'\)

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

\[
\text{Activator} \overset{\text{def}}{=} \varnothing^b : \text{Activator}' \\
\text{Activator}' \overset{\text{def}}{=} (\text{start}!,1)\varnothing^d : (\text{end}!,2).
\]

\(\varnothing^{p_1 \cdot d \cdot p_2 \cdot d} : \text{Activator}'\)

where

\[
\varnothing^n = \varnothing : \cdots : \varnothing \quad \text{(idling for } n \text{ time units)}
\]

\[
\varnothing^{m \cdot n} = \varnothing^m + \varnothing^{m+1} + \cdots + \varnothing^n
\]
Sample Jobs

Job 1

- preemptable, independent jobs running on cpu
- priority $\pi$ and execution time $c$

\begin{align*}
\text{Job} & \overset{\text{def}}{=} \emptyset : \text{Job} + (\text{start} \; ?, \; 1).\text{Exec}(0,0) \\
\text{Exec}(s,t) & = (s < c) \rightarrow ((\text{cpu}_i, \pi) \; : \; \text{Exec}(s + 1, t + 1)) \\
& \quad + (s = c) \rightarrow \text{Wait} \\
\text{Wait} & = \emptyset : \text{Wait} + (\text{end} \; ?, \; 1).\text{Job}
\end{align*}

- $s$ 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 $\text{cpu}_1, \ldots, \text{cpu}_k$
- priorities $\pi_1, \ldots, \pi_k$ and execution time $c$

\begin{align*}
\text{def} & \quad \text{Job} = \emptyset : \text{Job} + (\text{start} \; ?, \; 1).\text{Exec} \\
\text{def} & \quad \text{Exec} = \sum_{i=1}^k \{(\text{cpu}_i, \pi_i)\}^c : \text{Wait} \\
\text{def} & \quad \text{Wait} = \emptyset : \text{Wait} + (\text{end} \; ?, \; 1).\text{Job}
\end{align*}

- 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 \( c \)
- a single preemptable critical section of length \( cs \) on resource \( data \) (with priority \( \pi \)) after at \( c' \) time units execution:

\[
\begin{align*}
\text{Job} & \triangleq \text{Job} + (\text{start},1).\text{Exec}(0,0) \\
\text{Exec} & \triangleq (s < c \wedge s \neq c') \rightarrow ((\text{cpu},\pi)) : \text{Exec}(s+1,t+1) \wedge \Box \langle \text{Exec}(s,t+1) \rangle \\
& + (s = c') \rightarrow ((p\gamma,0)\text{CS}(s,t)) \wedge \Box \langle \text{Exec}(s,t+1) \rangle \\
& + (s = c) \rightarrow \text{Wait} \\
\text{Wait} & \triangleq \text{Wait} + (\text{end},1).\text{Job} \\
\text{CS}(s,t) & \triangleq (s < c' + cs) \rightarrow ((\text{cpu},\pi)) : \text{CS}(s+1,t+1) \\
& + (s = c' + cs) \rightarrow ((0,0)\text{Exec}(c,s,t)) \\
\end{align*}
\]

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

3 October 2003

Scheduling Disciplines

Earliest Deadline First
- Tasks \( T_i = \text{Job}_1 + \text{Activator}_i \)
- Priority \( \pi_i = d_{\text{max}} - (d_i - t) \)
  where \( d_{\text{max}} = (1 + \max\{d_1, \ldots, d_n\}) \)

\[
\begin{align*}
\text{EDFSys} & \triangleq \bigcup_i T_i \\
T_i & \triangledown = (\text{Job}_1 \text{Activator}_i) \setminus \{\text{start, end}\} \\
\text{Job}_j & \triangleq \emptyset : \text{Job}_j + (\text{start},1).\text{Exec}(0,0) \\
\text{Exec}(s,t) & \triangleq (s < c) \rightarrow ((\text{cpu},d_{\text{max}} - (d_i - t))) \\
& \wedge \Box \langle \text{Exec}(s,t+1) \rangle \\
& + (s = c) \rightarrow \text{Wait} \\
\text{Wait}_i & \triangleq \emptyset : \text{Wait} + (\text{end},1).\text{Job}_i \\
\text{Activator}_i & \triangledown = (\text{start},1) \triangledown^{(1)} : (\text{end},1) \triangledown^{(n)} : \text{Activator}
\end{align*}
\]
Other Time-Driven Scheduling Disciplines

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

where $c_{\text{max}} = (1 + \max\{c_1, \cdots, c_n\})$
## Priority Inheritance Protocol

\[ T_0 = \text{Job} \cup \text{Activator} \cup \text{Priority - Passing Events} \]

### Resources

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu</td>
<td>processor</td>
</tr>
<tr>
<td>ready time</td>
<td>( r_1 = 5 ) ( r_2 = 10 ) ( r_3 = 0 )</td>
</tr>
<tr>
<td>comp. time</td>
<td>( c_1 = 6 ) ( c_2 = 8 ) ( c_3 = 13 )</td>
</tr>
<tr>
<td>deadline</td>
<td>( d_1 = 30 ) ( d_2 = 30 ) ( d_3 = 30 )</td>
</tr>
<tr>
<td>start time of CS</td>
<td>( cs_1 = 3 ) ( cs_2 = 5 ) ( cs_3 = 1 )</td>
</tr>
<tr>
<td>length of CS</td>
<td>( c'_1 = 2 ) ( c'_2 = 2 ) ( c'_3 = 10 )</td>
</tr>
<tr>
<td>priority</td>
<td>( \pi_1 = 3 ) ( \pi_2 = 2 ) ( \pi_3 = 1 )</td>
</tr>
<tr>
<td>max priority</td>
<td>( \pi_{\text{max}} = 4 )</td>
</tr>
</tbody>
</table>

### Priority Inheritance Protocol

\[ \text{Execution} = (\text{Job} \cup \text{Activator}) \cup \text{Priority - Passing Events} \]

\[ \text{CS}(s, x) = (s < c_x \cup c'_{cs}) \cup (c_x \cup c'_{cs}) \cup \text{Req}(s) \]

\[ \text{CS}(s, x) = (s < c_x + c'_{cs}) \cup (c_x + c'_{cs}) \cup \text{CS}(s, x, \text{new}) \cup \text{Req}(s) \]

\[ \text{Activator} = (\text{Job} \cup \text{Activator}) \cup \text{Priority - Passing Events} \]

Parameters of \( T_0 \):
- \( c_x \): Execution time of a job
- \( c'_{cs} \): Time for entering critical section
- \( c'_{cs} \): Execution time in critical section
Traces of tasks

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>req, h, (cpu, 3)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
</tr>
<tr>
<td>2</td>
<td>(cpu, 3)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
</tr>
<tr>
<td>3</td>
<td>(cpu, 3)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
<td>V(1)</td>
</tr>
</tbody>
</table>

*: in critical section

3 October 2003

ESSES 2003

75

Weak Bisimulation

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

Def. If $t = \alpha_1 \ldots \alpha_n \in D^\tau$, then $E \vdash E'$ if $P(t) \xrightarrow{\alpha_1} P(t) \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} P(t)$, where "_" in $(\tau, _) \text{ represents arbitrary integer.}

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

1. if $P \xrightarrow{\alpha} P'$, then, for some $Q' \in Q$ and $(P', Q') \in r$, and
2. if $Q \xrightarrow{\alpha} Q'$, then, for some $P', P \xrightarrow{\alpha} P'$ and $(P', Q') \in r$.

Def. $\sim$ is the largest weak bisimulation over "$\rightarrow"$. It is an equivalence relation (though not a congruence) for ACSR.

3 October 2003

ESSES 2003

76
Analyzing Real-Time Systems in ACSR-VP

- Two types of analyses
  - Validation
  - Schedulability analysis
- Basic idea
  - Checking weak bisimulation $\approx_{\pi}$
  - 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 $\text{EDFSys} \approx_{\pi} \text{EDFSpec}$

\[
\text{EDFSpec} = [S(0, \cdots, 0, 0)]_{\text{cpu}}
\]

\[
S(s_1, t_1, \cdots, s_n, t_n) =
\begin{cases}
\begin{align*}
(s_i = c_j & \land t_i = p_j) \\
+ (s_i < c_j & \land t_i = d_j)
\end{align*}
\rightarrow (\tau, 1).S(\cdots, s_{i-1}, t_{i-1}, 0, 0, s_{i+1}, t_{i+1}, \cdots)
\end{cases}
\]

\[
\sum_{1 \leq i < n}
\begin{cases}
\begin{align*}
(s_i = c_j & \land t_i < p_j) \\
+ (s_i < c_j & \land t_i < d_j)
\end{align*}
\rightarrow \emptyset: S(\cdots, s_{i-1}, t_{i-1} + 1, s_i, t_i + 1, s_{i+1}, t_{i+1} + 1, \cdots)
\end{cases}
\]

\[
\rightarrow \{(\text{cpu}, d_{\text{max}} - (d_j - t))
\rightarrow S(\cdots, s_{i-1}, t_{i-1} + 1, s_i + 1, t_i + 1, s_{i+1}, t_{i+1} + 1, \cdots)
\]
Schedulability Analysis

Lemma 1 If \( \text{EDFSys} \) is deadlock free, then it is schedulable.

Lemma 2 If \( \text{EDFSys} \setminus \{ \text{cpu} \} \simeq_{\pi} \emptyset^{\infty} \),
then \( \text{EDFSys} \) is deadlock free.

Lemma 3 If \( \text{PIPSys} \) is deadlock free, then it is schedulable.

Lemma 4 If \( \text{PIPSys} \setminus \{ \text{cpu} \} \simeq_{\pi} \emptyset^{\infty} \),
then \( \text{PIPSys} \) is deadlock free.

Example 1

- Consider an instance \( \text{EDFSys}_1 \) of \( \text{EDFSys} \) 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 \( \text{EDFSys} \setminus \{ \text{cpu} \} \simeq_{\pi} \emptyset^{\infty} \),
is satisfied, i.e., the task system is schedulable.
More specifically, we have

\[
\text{EDFSys}_1 \xrightarrow{(r,2)} \text{EDFSys}_1 \xrightarrow{(2,2)} \text{EDFSys}_1 \xrightarrow{\{(\text{cpu},2)\}} \text{EDFSys}_1 \xrightarrow{\{(\text{cpu},3)\}} \text{EDFSys}_1 \xrightarrow{(r,3)} \text{EDFSys}_1
\]

3 October 2003  ESSES 2003  79
Example 2

- Consider another instance $\text{EDFSys}_2$ of $\text{EDFSys}$ where:
  - Task $T_1$: $c_1 = 2, d_1 = 2, p_1 = 3$
  - Task $T_2$: $c_2 = 2, d_2 = 3, p_2 = 3$
- The equivalence
  $$\text{EDFSys}_2 \setminus \{cpu\} \equiv^\pi \emptyset,$$
  is false and the task system is therefore not schedulable.
- More specifically, we have
  $$\text{EDFSys}_2 \xrightarrow{(r,2)} \pi \xrightarrow{(r,2)} \pi \xrightarrow{((\text{cpu},2))} \pi \xrightarrow{((\text{cpu},2))} \pi \xrightarrow{(r,3)} \pi \xrightarrow{((\text{cpu},3))} \pi \xrightarrow{\pi} \text{NIL}$$

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
Probabilistic ACSR for soft real-time scheduling analysis

PACSR (Probabilistic ACSR)

- ACSR extension for probabilistic behaviors.
- Objective:
  - formally describe behavioral variations in systems that arise due to failures in physical devices.
  - Since failing devices are modeled by resources we associate a failure probability $p(\tau)$ with every resource $\tau$
    - at any time unit, $\tau$ is down with probability $p(\tau)$ or up with probability $1-p(\tau)$
    - failures are assumed to be independent
Syntax for PACSR processes

• Similar to ACSR

• Process terms

\[ P := \text{NIL} \mid A : P \mid (a, n). P \mid P + P \mid P \parallel P \mid P \triangleright (Q, R, S) \mid [P]_i \mid P \setminus F \mid b \rightarrow P \mid C \]

• Process names

\[ \text{def} \quad C = P \]

• Distinction: For all resources \( r \) we write \( \tau \) for the failed occurrence of resource \( r \). Thus, an action can specify access to failed resources.

Resource failures and recoveries

• An action containing resource \( r \) cannot be taken when \( r \) is failed, i.e.,

\[ r \text{ is failed, } r \in \rho(A) \Rightarrow A : P = \text{NIL} \]

• Failed resources: \( \bar{r} \), \( \text{pr}(\bar{r}) = 1 - \text{pr}(r) \)

• Recoveries are modeled by using failed resources in actions

\[
\begin{align*}
\{(\text{phone,1})\} : \text{PlaceCall} \\
+ \{(\text{phone,1})\} : \text{UsePayPhone}
\end{align*}
\]
PACSR Semantics

• Semantics of a PACSR process is given in terms of probabilistic transition systems: some transitions are labeled with probabilities and others with actions/events.

• Labeled Concurrent Markov Chain (LCMC)

\[
\begin{align*}
\tau & \quad 1/2 \quad a \\
\tau & \quad 1/2 \quad b \\
\tau & \quad 2/3 \quad c \\
\tau & \quad 1/3 \quad d
\end{align*}
\]

PACSR Semantics

• Configurations are pairs of the form \((P,W)\), where
  - \(P\) is a PACSR process, and
  - \(W\) is a world capturing the state of resources as follows
    \[
    \forall r, r \in W \Rightarrow \neg r \in W \quad \text{and} \quad \forall r, \neg r \in W \Rightarrow r \in W
    \]

• A configuration \((P,W)\) is characterized as
  - Probabilistic, if \(P\) requires resources whose state is not in \(W\).
    Example: \((\{r_1,1\}:Q, \{r_2\})\)
  - Nondeterministic, if all resource information required by \(P\) is in \(W\).
    Example: \((a,1):\text{NIL}, \emptyset\)
PACSR semantics (II)

- The semantics is given via a pair of transition relations:
  - Probabilistic transition relation,
    \[(P,W) \xrightarrow{pr} \rho(P,W')\]
  - Nondeterministic transition relation,
    \[(P,W) \xrightarrow{a} (Q,W)\]

- Let \( \text{imr}(P) \) be resources that can be used in the first step:
  \[\{ r \mid P \xrightarrow{A} P', r \in \rho(A) \}\]

Operational semantics

- The probabilistic transition relation is as follows:
  \[P \in S_p, Z_1 = \text{imr}(P) - (W \cup \overline{W}), Z_2 \in W(Z_1)\]
  \[(P,W) \xrightarrow{pr(Z_2)} \rho(P,W \cup Z_2)\]

  \( W(Z) \) is a set of all possible scenarios of resources: e.g.,
  \[W(\{r_1, r_2\}) = \{(r_1, r_2), (r_1, r_2), (r_1, r_2), (r_1, r_2)\}\]

- The nondeterministic transition relation is taken from ACSR, with one exception:
  \[\text{ActT} \xrightarrow{\rho(A) \subseteq W} \rho(A) \subseteq W\]
Example

• Let \( P = \{(r_1,2), (r_2,3)\} : Q \), \( \text{pr}(r_1) = \frac{1}{3} \) and \( \text{pr}(r_2) = 1/3 \).

Then \( \text{imr}(P) = \{r_1, r_2\} \) and \( \text{W}(\{(r_1, r_2)\}) = \{(r_1, \bar{r}_2), (\bar{r}_1, r_2), (\bar{r}_1, \bar{r}_2)\} \)

• Thus by the probabilistic transition relation

\[
\begin{align*}
(P, \phi) & \xrightarrow{1/6} \rho(P, \{r_1, r_2\}) & (P, \phi) & \xrightarrow{1/6} \rho(P, \{\bar{r}_1, r_2\}) \\
(P, \phi) & \xrightarrow{1/3} \rho(P, \{r_1, r_2\}) & (P, \phi) & \xrightarrow{1/3} \rho(P, \{\bar{r}_1, r_2\}) \\
\end{align*}
\]

• and by the nondeterministic transition relation

\[
\begin{align*}
(P, \{r_1, r_2\}) & \xrightarrow{\tau} \\
(P, \{r_1, r_2\}) & \xrightarrow{\tau} (P, \{\bar{r}_1, r_2\}) \\
(P, \{\bar{r}_1, r_2\}) & \xrightarrow{\tau} (Q, \phi) \\
\end{align*}
\]

Example: A faulty channel

\[ \text{FCh} = \phi : \text{FCh} \]

\[ + \text{in.}(\{\text{ch}\} : \text{out}! \cdot \text{FCh} \]

\[ + \{\bar{\text{ch}}\} \cdot \text{FCh} \setminus \{\text{ch}\} \]

\[ \text{where } \text{pr}(\text{ch}) = 0.99 \]
Probabilistic HML with until

- In order to analyze PACSR specifications we may want to check whether a specification satisfies a property written as a logical formula.
- We use a probabilistic HML with an 'until' operator.
- The 'until' operator is parameterized with regular expressions over event names.

Syntact
\[ f ::= tt \mid \neg f \mid f \land f' \mid f\langle \Phi \rangle_{\omega} f' \mid f\langle \Phi \rangle_{\omega} \]

where \( \Phi \) is a regular expression over actions and \( \omega \in \{\leq, \geq\} \)

The until operator

There is some execution with probability \( \leq q \) for which \( f \) holds until \( f' \) becomes true within time \( t \) and observable behavior from \( \Phi \)

EXAMPLE

\[ \text{true}\langle \{\text{talk, wait}\}^{20} \text{hangup}\rangle_{0.01} \text{true} \]

= the probability that within 20 time units after any number of talk and wait actions action hangup arises is \( \leq 0.01 \)
Semantics for *until*

• \( s \models f_1 \langle \Phi \rangle \succ_\pi f_2 \)
  
  if there exists a scheduler \( \sigma \) such that the set of computations that
  - start at \( s \)
  - contain only states (except the last) satisfying \( f_1 \)
  - have observable content \( \Phi \)
  - end in a state satisfying \( f_2 \)

  have probability greater than \( \pi \)

---

**Resolving non-determinism**

• Analysis involves computing the probability of reaching a set of desired states (within a time period) via an acceptable set of behaviors.

• Example:
  
  ![Diagram](image-url)

  - What is the probability that event *head* takes place?
  - Such probability depends on how the nondeterminism of \( s \) is resolved.
Model Checking

- **Schedulers** are used for resolving non-determinism. These are functions that given a computation ending in a nondeterministic state choose the next transition to take place.

- Given a scheduler $\sigma$ of a system $P$, sets of states $A$ and $B$, and a regular expression $\Phi$, we may compute probabilities
  - $\Pr_{\sigma}(P \rightarrow B, \Phi, t, \sigma)$, the probability of reaching a state in $B$, passing only via states in $A$, via paths with observable content in $\Phi$, and within $t$ time units

- So for example: 
  \[
  P \models f_1(\Phi) \Rightarrow f_2 \iff \text{there is scheduler } \sigma \text{ such that } q \geq \Pr_{\sigma}(P \rightarrow B, \Phi, t, \sigma)
  \]
  where $A = \{ P' \mid P' \models f \}$, $B = \{ P' \mid P' \models f' \}$

---

Model checking *until*

- To check $s \models f_1(\Phi) \Rightarrow f_2$
  - Compute the least solution to the set of equations:
    \[
    X_{f_1(\Phi)\ f_2}^s = \begin{cases} 
      \sum_{s' \in S} \pi \cdot X_{f_1(\Phi)\ f_2}^{s'} & s \in S \setminus P \\
      \max_{s' \in S} (X_{f_1(\Phi)\ f_2}^{s'}) & s \in S \setminus N \\
      1 & s = s_1, s \models f_1 \\
      0 & s \in S \setminus N, s \models f_2, e \in \Phi, p \geq 0 \\
      \text{otherwise} & 
    \end{cases}
    \]
  - Return true if $X_{f_1(\Phi)\ f_2}^s \geq \pi$
Equivalence Relations

- New notions of equivalence for the LCMC model taking account both action types and probabilities.

- In particular two LCMCs are strongly bisimilar if
  1. they reach sets of bisimilar states with the same probability, and
  2. for each nondeterministic step of one there exists a step of the other leading to bisimilar states.

Equivalence Relations

- There is a set of laws that completely axiomatizes strong bisimulation for PACSR processes.

- Other equivalence notions include *weak bisimulation* which relates systems that have the same observable behavior, that is, it ignores $\tau$ actions.
Two Examples

- EDF with probabilistic execution time
- Telecommunication application

EDF task scheduling

- Periodic process Job: Period $p_i$, computation time $c_i$
  - At each step, total time $i$ increases, active time $e$ increases only if resource cpu is available; complete when $e = c_i$
- Resource cpu: scheduling
  - Priority of a task dynamically increases closer to the deadline
- Process Actuator keeps timing deadlines
  - Every $p_i$ seconds, signal start is sent to the task, which can accept it only if it has finished execution
EDF task with probabilistic completion

- The task may decide to become inactive after completing a computation step.
- Resource `cont` controls probabilistic completion:
  - failure means "terminate early"

A Telecommunication Application

- Based on the specification of a switching system considered in AJK97.
- The system consists of a number of concurrent processes with real-time constraints.
- Probabilistic behavior is present in the form of:
  - probabilistic arrival of alarms, and
  - uncertain execution times of processes.
Example: A Telecommunication Application

The system in its initial state: a parallel composition of all the components

PACSR Specification

- The System

\[ S_{sys} = (Env \parallel B_0 \parallel \phi : Sched \parallel AS \parallel AH \parallel BP) \backslash F \backslash I \]

- The environment

\[ Env = \Pi_{i \in \text{sys}} P_i \]
\[ P_i = \{r_i\} : P_i + \{r_i\} : (P_i \parallel Q_i) \]
\[ Q_i = a : NIL + \phi : Q_i \]

The environment provides probabilistic alarms: at the failure of any of resources \( r_i \), an alarm is sent via channel \( a \)
PACSR Specification

• Background Process

\[ BP = (tc, 0).BP \Delta (NIL, NIL, kill.BP) + \phi : BP \]
\[ BP' = (\{r\} : BP' + \{r\} : rc.BP) \setminus \{r\} \]

The background process competes for processor time managed by the scheduler. Its duration is geometrically distributed.

• The Scheduler

\[ Sched = \phi : Sched + (tc, 1) \phi^* \Delta_{run}^i (NIL, kill.Sched, rc.Sched) \]

PACSR Specification

• The buffer

\[ B_0 = in.B_i + \phi : B_0 \]
\[ B_i = in.B_{i+1} + \sum_{j \geq i} d_j.B_{i-j} + \phi : B_i + out_i.B_i \]
\[ B_n = in.overflow.NIL + \sum_{i \geq n} d_j.B_{i-j} + \phi : B_n + out_n.B_n \]

• The Alarm Sampler and the Alarm Handler

\[ AS = AS' || (\phi^p : AS) \]
\[ AS' = (ic, 2).AS'' + \phi : AS' \]
\[ AS'' = a.in.AS'' + \phi : rc.NIL \]
\[ AH = \Sigma.out_i.AH_{n(i)} + \phi : AH \]
\[ AH_i = (ic, 2).AH_i^A + \phi : AH_i \]
\[ AH_i^A = \phi^{pet}(i) : \overline{d_i}.rc.AH \]
Two configurations

- Consider two versions of the system:
  \( S_1 \) with
  - Possibility of 1 alarm per time unit,
  - Buffer size of 3
  - Capability of processing 2 alarms per time unit, and
  \( S_2 \) with
  - Possibility of 2 alarms per time unit
  - Buffer size of 6
  - Capability of processing 4 alarms per time unit
- Comparison criterion: What is the probability of overflow in the alarm buffer?

Checking \( f = tt(overflow)^t \leq q tt \)

<table>
<thead>
<tr>
<th>T(time units)</th>
<th>( S_1 )</th>
<th>( S_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2x10(^{-6})</td>
<td>3x10(^{-10})</td>
</tr>
<tr>
<td>20</td>
<td>5x10(^{-6})</td>
<td>6x10(^{-10})</td>
</tr>
<tr>
<td>30</td>
<td>9x10(^{-6})</td>
<td>1.0x10(^{-2})</td>
</tr>
<tr>
<td>40</td>
<td>1.2x10(^{-5})</td>
<td>1.3x10(^{-9})</td>
</tr>
<tr>
<td>50</td>
<td>1.5x10(^{-5})</td>
<td>1.6x10(^{-9})</td>
</tr>
<tr>
<td>60</td>
<td>1.9x10(^{-5})</td>
<td>2.1x10(^{-9})</td>
</tr>
<tr>
<td>70</td>
<td>2.2x10(^{-5})</td>
<td>2.4x10(^{-9})</td>
</tr>
<tr>
<td>80</td>
<td>2.5x10(^{-5})</td>
<td>2.8x10(^{-9})</td>
</tr>
<tr>
<td>90</td>
<td>2.9x10(^{-5})</td>
<td>3.1x10(^{-9})</td>
</tr>
<tr>
<td>100</td>
<td>3.2x10(^{-5})</td>
<td>3.5x10(^{-9})</td>
</tr>
</tbody>
</table>

The table shows for various values of \( t \), the probability \( q \) that makes property \( f \) true for each of the systems.
Design of Embedded Systems in a Resource-oriented Framework

Embedded systems design process

- High-level model
- Code generation
- Task set
- Timing estimation
- Task set with timing
- Resource modeling
- Resource tradeoffs
- Task allocation
- Platform parameters
- Schedulability analysis
- Resource-aware model
Modeling and code generation

- High-level model captures functionality of the system and assumptions about the environment
- Code generation breaks the functional behavior into a set of tasks

Timing parameter estimation

- Estimate the execution time for task on a given platform
- Assign task periods based on end-to-end timing constraints
Resource modeling

- Resource is a critical notion in embedded and real-time system design, yet lacks systematic formal treatment
- Key idea: resource attributes capture tradeoffs

Formal schedulability analysis

- Resource conflicts introduce execution delays
- Violations of timing constraints lead to deadlocks in the model behavior
- Discovered by state-space exploration
Modeling and Analysis of Power-Aware Systems

Motivation

- Features of mobile embedded systems:
  - Resource constraints
    - Limited battery life
  - Uncertainty
    - changing communication delays, failures

- Solution:
  - a unified formal framework for designing and reasoning about power-constrained, timed systems with probabilistic behavior
$P^2\text{ACSR}$ – A power-aware extension of PACSR

- A unified framework for modeling and analyzing power-aware real-time systems.
- We associate a further attribute to resource usage, that of power consumption.
- The syntax remains the same, except that actions are tuples of the form $(r,p,c)$, where $r$ is the resource, $p$ is the priority level and $c$ the power consumption of the resource usage.

**EXAMPLE**

$(\text{phone},1,0) : \text{Call}_1$

$+$

$(\text{cellphone},1,3) : \text{Call}_2$

$P^2\text{ACSR}$

- Semantics is given similarly to PACSR, as a LCMC.
- We can use various techniques to perform various analyses on $P^2\text{ACSR}$ models including:
  - **Model checking**
    We may express temporal logic properties involving power consumption bounds and check that they are satisfied by $P^2\text{ACSR}$ processes.
  - **Probabilistic bounds on power consumption**
    We may compute the probability that power consumption exceeds certain limits.
  - **Average power consumption**
    We may compute the average power consumption during intervals of interest.
P²ACSR

- P²ACSR is an extension of PACSR, a probabilistic real-time process algebra.

- In P²ACSR:
  - system is a collection of concurrent processes
  - communication among processes is instantaneous
  - access to serially-reusable resources consumes time and power

Resources

- Resources capture constraints on executions

- Features of resources:
  - Serially reusable
    - processors, memory, communication channels
  - Unreliable
    - Fail with a fixed probability in each step
  - Require time and power
    - May allow different levels of power consumption
Actions

- **Actions** represent computation
  - actions take one unit of time
  - require access to resources
    - each resource \( r \) has priority of access \( p_r \)
    - each resource \( r \) has power use level \( c_r \)
    - \( A = \{ (r_1, p_{r_1}, c_{r_1}), (r_2, p_{r_2}, c_{r_2}) \} \)
  - each resource can be used at most once
  - resources of action \( A \): \( \rho(A) \)
  - power consumption of action \( A \): \( pc(A) = \sum_{r \in \rho(A)} c_r \)

Power constraints

- Resource classes \( \mathcal{R}_1, \ldots, \mathcal{R}_n \)
  - correspond to different power sources

- Attributes of resource class \( \mathcal{R}_i \):
  - capacity \( C_i \) - maximum amount of power in one step
  - charge \( p_i \) - total amount of power

- Valid actions satisfy capacity constraints:
  - for each \( \mathcal{R}_i \), \( pc_i(A) = \sum_{r \in \rho(A), r \in \mathcal{R}_i} c_r < C_i \)
Processes

- Event and action steps
- Choice $P_1 + P_2$
- Parallel composition $P_1 \parallel P_2$
- Temporal scope, time-outs, exceptions, ...

- Structural operational semantic rules build behaviors of complex processes from behaviors of component processes

P^2ACSR semantics

- Before steps of a process can be computed, status of relevant resources has to be determined
- Resource status is kept in a world
- Non-deterministic configurations $S_n$
  - world has complete knowledge of resources
- Probabilistic configurations $S_p$
  - incomplete knowledge
- Probabilistic steps: $S_p \rightarrow S_n$
  - acquire missing knowledge
Non-deterministic rules

• Action can happen if all resources are available and power constraints are obeyed:

\[ (A : P, W) \rightarrow^A (P, \emptyset), \quad \rho(A) \subseteq W, valid(A) \]

• Parallel processes can proceed if their actions do not conflict and the joint step does not violate constraints

\[
\begin{align*}
&P \rightarrow^A P' \quad Q \rightarrow^A Q' \\
&P \parallel Q \rightarrow^{A \cup A_2} P' \parallel Q' \\
&\rho(A_1) \cap \rho(A_2) = \emptyset, valid(A_1 \cup A_2)
\end{align*}
\]

• Model: Labeled Concurrent Markov Chains

Example

• \( C \) is a process that reliably translates messages from \( \text{in} \) to \( \text{out} \) in 1 time unit using 2 units of power per message

\[ \pi(\text{cpu}) = 1 \]
Example

• FC (fault-tolerant C) accommodates for cpu failures
  – π(cpu) = 0.99
• If cpu fails, the message is not delivered, but less power is consumed
• Message is delivered with probability 1
  - What is the expected power consumption per message?

A logic for power constraints

• $L_{PHMLu}^{pc}$: Power-aware probabilistic HML with until
  - Propositional operators  $tt, ¬f, f_1 \land f_2$
  - until operators specify probabilistic bounds on power consumption along a set of paths
    • Basic variant: $f_1\langle \Phi \rangle_{>\pi}^{\leq p} f_2$
    • With time constraints: $f_1\langle \Phi \rangle_{>\pi,t}^{\leq p} f_2$
    • With resource class constraints: $f_1\langle \Phi \rangle_{>\pi}^{\leq p,\mathcal{R}} f_2$
Semantics for $\mathcal{L}_{PHMLu}^{pc} : until$

- $s \models f_1 \langle \Phi \rangle_{>\pi}^{\leq p} f_2$
  - if there exists a scheduler $\sigma$ such that the set of computations that
    - start at $s$
    - contain only states (except the last) satisfying $f_1$
    - have observable content $\Phi$
    - consume no more power than $p$
    - end in a state satisfying $f_2$
  - have probability greater than $\pi$

Model checking $until$

- To check $s \models f_1 \langle \Phi \rangle_{>\pi}^{\leq p} f_2$
  - Compute the least solution to the set of equations:

$$X^{s,z}_{f_1,\Phi} = \begin{cases} \sum_{s' \in S} \pi \cdot X^{s',z}_{f_1,\Phi} X^{s'}_{f_1,\Phi} f_2 & s \in S_p \\ \max_{z' \in S_n} \langle X^{s',z'}_{f_1,\Phi} X^{s' - pow(c)} f_2 \rangle & s \in S_n, s \models f_1 \\ 1 & s \in S_n, s \models f_2, \epsilon \in \Phi, p \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

  - Additional annotation $\leq p$ in the variable set
  - Return true if $X^{s,z}_{f_1,\Phi} > \pi$
Example

• Power consumption per message:

\[ FC, \emptyset | = \text{tt} \left( \text{in}^{\text{cpu,cpu}} \text{out} \right)^{\geq 3} \text{tt} \]

\[ FC, \emptyset | = \text{tt} \left( \text{in}^{\text{cpu,cpu}} \text{out} \right)^{\geq 0.999} \text{tt} \]
Power-aware scheduling

- Trade-off: power vs. execution time
- CMOS-based processors can operate at reduced voltage levels
  - Power dissipation is proportional to $V^2$
  - StrongARM SA2:
    - 600 MHz / 500 mJ or 150 MHz / 160 mJ
- Tasks can take less than worst-case time
  - Adjust frequency dynamically to utilize “time slack”

Dynamic Voltage Scaling

- Dynamic voltage scaling is a technique proposed for making energy savings by dynamically altering the power consumed by a processor.
- Lower frequency execution implies longer processing of tasks.
- This may lead to violation of real-time constraints.
- [Pillai and Shin 01] propose extensions to real-time scheduling algorithms to make use of dynamic voltage scaling.
Case study: two kinds of resources

- Power-aware resources:
  - Attributes:
    - Priority (dynamic) - schedulability analysis
    - Power consumption (dynamic) - power calculations

- "abstract" resources:
  - Attributes:
    - Availability (static) - probabilistic completion
    - No power consumption, same priority

Power-Aware Real-Time Scheduling

- Let $I$ be a set of tasks with periods $p_i$ and worst-case execution times $c_i$, sharing the same CPU.

- In reality tasks often take much less time to execute.

- This probabilistic execution time may be modeled in PACSR as follows:
  \[
  \text{Task}_i = (\text{start}?,0) \cdot \text{Exec}_{i,0,0} + \emptyset : \text{Task}_i \quad i = \{1, \ldots, I\}
  \]
  \[
  \text{Exec}_{i,e,t} = e < c_i \rightarrow (\emptyset : \text{Exec}_{i,e+1,t} + \{(\text{cpu}, d_{\max} - (p_i - t)) (\text{cont},1)\} : \text{Exec}_{i,e+1,t+1} + \{(\text{cpu}, d_{\max} - (p_i - t)) (\text{cont},1)\} : \text{Task}_i)
  \]
  \[
  + e = c_i \rightarrow \text{Task}_i \quad i = \{1, \ldots, I\}
  \]
  \[
  e = \{0, \ldots, c_i\}
  \]
  \[
  t = \{0, \ldots, c_i\}
  \]
  
  (potential for early termination (geometric distribution))
Power-Aware Real-Time Scheduling

• The algorithm of [Pillai and Shin] takes advantage of the possibility of early termination of a task by then executing the next task at the lowest possible frequency.

• Specifically, on every release or completion of a task it re-computes the sum

\[ \alpha = \frac{c_{last}^1}{p_1} + \ldots + \frac{c_{last}^n}{p_n} \]

where \( c_{last}^i \) is the computation time of the last execution of task \( i \) or \( c_i \) if task \( i \) has just been released.

• Based on this value it decides the lowest frequency that is consistent with the current effective utilization.

Power-Aware Real-Time Scheduling

• First we extend the model of a task with the ability of executing slower or faster. It responds to messages fast and slow. In the slow mode a computation step takes twice as long, i.e. two time units. It also signals its release when execution commences and its completion time when it completes.
Speed-sensitive task

- If operating frequency is fast, take one time unit per scheduling step
- If operating frequency is slow, take two time units per scheduling step

Power-Aware Real-Time Scheduling

- The DVS algorithm is represented as the $P^2ACSR$ process:

$$DVS = (Scale_{e_1,e_2,e_3} \parallel Proc_{fast}) \setminus \{f_{up}, f_{down}\}$$

- Scale responds to release and completion signals and triggers the re-computation of $\alpha$

$$Scale_{e_1,e_2,e_3} = (release_{i},o).SetNew_{e_1,e_2,e_3} + (release_{2},o).SetNew_{e_2,e_3,e_1} + (release_{3},o).SetNew_{e_3,e_1,e_2} + \ldots + (end_{1,o},?).SetNew_{e_3,e_2,e_1} + (end_{2,o},?).SetNew_{e_2,e_3,e_1} + \ldots$$
Power-Aware Real-Time Scheduling

- SetNew decides the lowest frequency to the current effective utilization and sends the appropriate signal

\[ \text{SetNew}_{e_1,e_2,e_3} = \frac{e_1}{p_1} + \frac{e_2}{p_2} + \frac{e_3}{p_3} < \frac{1}{2} \rightarrow (f_{\text{down}}, 4) \]

\[ \text{Scale}_{e_1,e_2,e_3} + \frac{e_1}{p_1} + \frac{e_2}{p_2} + \frac{e_3}{p_3} \geq \frac{1}{2} \rightarrow (f_{\text{up}}, 4) \]

- DVS_{fast} and DVS_{slow} describe the processor operating in the high and low frequency, respectively

\[ \text{DVS}_{\text{fast}} = \{(\text{power}, \text{pw}_{\text{fast}})\}: \text{DVS}_{\text{fast}} + (\text{fast}, 1) \]

\[ \text{DVS}_{\text{slow}} = \{(\text{power}, \text{pw}_{\text{slow}})\}: \text{DVS}_{\text{slow}} + (\text{slow}, 1) \]

Operating frequency manipulation

- Recompute frequency each time a task is released or completed
- Consume \( \text{pw}_{\text{fast}} \) in fast mode and \( \text{pw}_{\text{slow}} \) in slow mode
Analysis of DVS

• We considered the following set of tasks:

<table>
<thead>
<tr>
<th>Task</th>
<th>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>1</td>
<td>14</td>
</tr>
</tbody>
</table>

• The algorithm guarantees the task set remains schedulable.

• We computed the expected power consumption for one major frame \((t=p_1 \cdot p_2 \cdot p_3)\) for \(pr(cont)=1/3\) and \(pw_{fast}=2\), \(pw_{slow}=1\).
  - With DVS minimum power consumption = 1906.66 and maximum power consumption = 1922.65
  - Without DVS power consumption = 2240
  - Thus expected savings between 14% and 14.8%.

Conclusions

• We have developed a timed, probabilistic, process algebra that allows modeling the power consumption of system resources

• Various techniques for quantitative analysis of power properties have been developed and implemented in the PARAGON toolset
  - Probabilistic bounds computation
  - Model checking

• Research direction:
  - Uniform resource attribute model
Example: A Start-time Assignment Problem

- Start-time Assignment Problem with Inter-job Temporal
  Constraints
- The order of execution of job is not known
- 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.
ACSR-VP (ACSR With Value-passing)

- Extends ACSR with
  - variables: $(a?x,1).(c!x,1)$...
  - value passing communications: $(c!x,1) || (c?x,1)$...
  - parameterized processes: $P(x) = (x > 1) \rightarrow (a!x,1).nil$
- Priorities can be specified using expressions
  - timed actions: `{(data, y+1)}`
  - instantaneous events: `(signal!8, x+3)`

Syntax

\[ P ::= \text{NIL} \mid a . P \mid A : P \mid P + P \mid P \parallel P \]
\[ b \rightarrow P \mid P \setminus F \mid P \mid t \mid C \]
\[ a ::= (r, e) \mid (c?x, e) \mid (c!e_1, e_2) \]
\[ A ::= \emptyset \mid \{ S \} \]
\[ S ::= (r, e) \mid (r, e), S \]
\[ C ::= X \mid X(\tilde{v}) \]

Symbolic Graph With Assignment (SGA)

SGA is a directed graph with edges labeled with $b, \alpha$, and $\theta$, where $b$ is a Boolean condition, $\alpha$ is an action, and $\theta$ is an assignment.

We use SGA to capture the semantics of ACSR-VP

\[ P(x) = (a!x,1).Q(x) \]
\[ Q(y) = (y \leq 0) \rightarrow (b!y,1).Q(y+1) \]
\[ + (y > 0) \rightarrow (a!y-1,1).Q(y-1) \]

\[ P(0) \Rightarrow (a!0,1),(b!0,1),(a!0,1) \ldots \]
Symbolic Bisimulation (Informal Description)

\[
P(x) = (x < 0) \rightarrow (b!x,1).nil + (x \geq 0) \rightarrow (a!x+1,1).nil
\]

\[
Q(y) = (a!y,1).nil
\]

\[
P(x) \sim Q(y) \iff (x < 0 \Rightarrow false) \wedge (x \geq 0 \Rightarrow (true \wedge x+1 = y)) \wedge (true \Rightarrow (x \geq 0 \wedge y = x+1))
\]

Schedulability Analysis Using Symbolic Bisimulation

Suppose we have an ACSR-VP term System \((0,s_1,s_2)\) that model a real-time system or a scheduling problem. We generate the Symbolic Graph with Assignment for System \((0,s_1,s_2)\)

Given two SGAs, we can apply the symbolic weak bisimulation algorithm to check the equivalence of System \((0,s_1,s_2)\) and this idle process \(\varnothing^\infty\), which never deadlocks.

That is, finding a condition that makes a system schedulable is equivalent to finding a symbolic bisimulation relation with a non-blocking process.
ACSR-VP approach

- Provides a formal framework for modeling real-time systems, especially for real-time scheduling problems such as:
  - Priority Assignment Problem
  - Execution Synchronization Problem
    - Start-time assignment problem
    - Period assignment problem
- Deals with unknown parameters in the problems rather than "yes/no" answer (i.e., parametric approach)
- Provides a fully automatic method for the analysis of real-time scheduling problems
- Takes advantages of existing techniques such as integer programming and BDD

Overview of General Approach

System Described in ACSR-VP

Non-blocking Process in ACSR-VP

SGA

SGA

Symbolic Weak Bisimulation

Predicate Equations with Free Variables

Constraint Logic Programming or Theorem Prover

Solution Space (Ranges of Free Variables)
Example: Start-time Assignment 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.

```
Job_i(t,s) = (t < s) → ∅: Job_i(t+1,s)  
+ (t = s) → (Start!,1).Job_i'(0,t,s)

Job'_i(e,t,s) = (e < e_i) → ([cpu,1]): Job'_i(e+1,t+1,s) 
+ (e = e_i) → Job''_i(e,t,s)

Job''_i(e,t,s) = (e < e_i+1) → ([cpu,1]): Job''_i(e+1,t+1,s) 
+ (e ≥ e_i+1) → (Finished!,1).Idle
```

```
Constraint(t) = (start?,1).Constraint(t) + ∅: Constraint(t+1)
Constraint_1 (t) = (Finished?,1).Constraint_1 (t) + ∅: Constraint_1 (t+1)
Constraint_2 (t) = (t ≤ 12) → Constraint_2 (t,0)
Constraint_3 (t) = ...
```

```
System(s_1,...,s_n) = (Job_1(0,s_1)|...|| Job_n(0,s_n)|Constraint(0))|Start,Finished)
```

Modeling With ACSR-VP
Predicate Equations

- The following fragments of predicate equations are generated from the symbolic weak bisimulation algorithm with the infinite idle process:

\[
X_0(t, s_1, s_2) = (t \leq 5 \land t < s_2) \longrightarrow X_1(t+1, s_1, s_2) \\
\land (t \leq 5 \land t = s_1) \longrightarrow X_2(0, t+5, s_2) \\
\land (t \leq 5 \land t < s_1 \land X_1(t+1, s_1, s_2)) \\
\lor (t < 5 \land t = s_1 \land X_2(0, t+5, s_2)) \\
\]

To get the values of \( s_1 \) and \( s_2 \), we can ask a query \( X_0(0, s_1, s_2) \).

Solution Space

- The solutions to the predicate equations can be obtained using linear/integer programming techniques, constraint logic programming techniques, or a theorem prover.

- The solutions for the previous example are:

<table>
<thead>
<tr>
<th>Start time ( S_1 )</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start time ( S_2 )</td>
<td>14</td>
<td>14</td>
<td>15</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>
An Automatic Approach

- The disadvantage of symbolic weak bisimulation is that it requires to add new $\tau$ edges into SGA. This will increase the size of predicate equations.
- The disadvantage of CLP is that there is no guarantee that it terminates.

- Reachability Analysis: Finding a condition that makes a system schedulable is equivalent to finding a condition that guarantees there is always a cycle in an SGA regardless of a path taken.
  - No need to add new $\tau$ edges.
- Restricted ACSR-VP
  - Give syntactic restriction to identify a decidable subset of ACSR-VP:
    - Control Variable: in finite range; Values can be changed
    - Data Variable: could be in infinite range; Values cannot be changed
    - $P(x:0..100,y) = (x<0 \land x+y>10) \rightarrow \emptyset:Q(x+3,y)$
  - Generate a boolean expression or boolean equations (i.e., no need to use CLP).

Conclusions: resources

- We have presented a family of resource-bound process-algebraic formalisms:
  - the notion of a resource plays central role:
    - Abstractions of physical resources
    - Resource sharing: coordination and synchronization
    - Resource consumption takes time: real-time behavior
    - Resource failures: probabilistic behavior
  - Sample application domain: analysis of scheduling problems:
    - Other domains: protocol analysis, rapid prototyping.
Conclusions: analysis techniques

- Analysis of safety properties by means of deadlock detection
- Conformance analysis by means of equivalence and preorder checking
- Probabilistic analysis techniques:
  - Model checking
  - Resource utilization
- Parametric analysis in ACSR-VP

Extensions

- Presented: serially reusable resources with access constraints
- Other types of resources:
  - Consumable resources: each resource use depletes resource stock
  - Multi-capacity resources: allow simultaneous access by a limited number of processes
- Other kinds of resource constraints:
  - non-functional constraints such as memory, power consumption, weight, etc.
References


- These papers are also available on-line from www.cis.upenn.edu/~rtg/papers.php3.

3 October 2003

Thanks

- ... for invitation to ESSES 2003
- ... for fundamental work done by my former Ph.D. students:
  - Amy Zwarico
  - Rich Gerber
  - Patrice Brémont-Gregoire
  - Hanène Ben-Abdallah
  - Duncan Clark
  - Hee Hwan Kwak
- ... for support from ARO, NSF, ONR over a number of years

3 October 2003
Q & A