Automata on Multisets of Communicating Objects

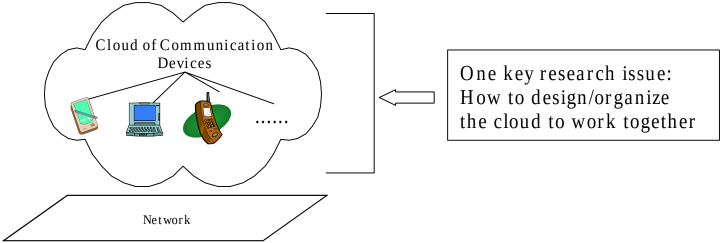
Linmin Yang[1] Yong Wang[2] Zhe Dang[1]

[1]School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA

[2]Google Inc.
Mountain View, CA 94043, USA

Motivation: Network Services

Network services: programs running on top of a (possibly large) number of devices, such as cellular phones, laptops, PDAs and sensors.

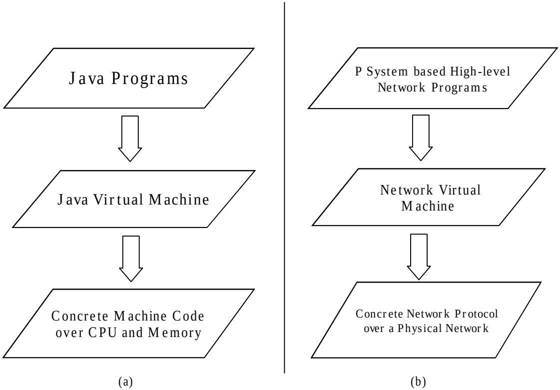


Related Work

How to design and implement such programs?

- Colony algorithms
  - inspired from ant colonies
- Input/Output (I/O) automata
  - models and reasons a concurrent and distributed discrete event system based on the broadcasting communication
- Linda
  - another model of communications among processes
- P systems
  - a biologically inspired abstract computing model running on multisets of symbol or string objects

A comparison of Java and P systems based high level network programs



Modeling of Network Services

We introduce a formal computation model called *service automata*.

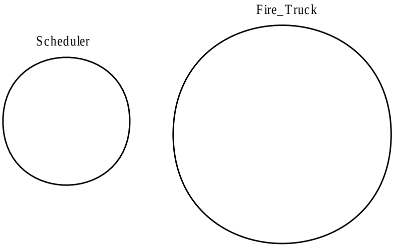
- Network devices are abstracted as communicating objects, which are typed but **addressless**.
- A communicating object is modeled as a finite automaton (FA).
- Objects of the same type have the same type of FA.
- The total number of objects is not specified.

Modeling of Network Services (contd.)

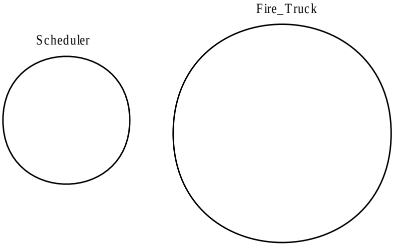
- A network service program is a service automaton running over a multiset of objects (finite automata).
- An *active* object is one holding a token.
- A *process* of the service automaton is a sequence of transitions that the token is passing through.

Definitions of Service Automata

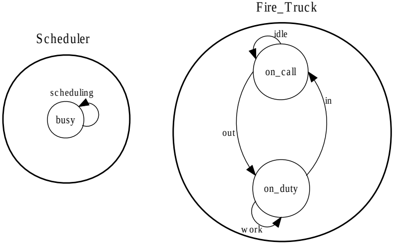
- $\Sigma = \{A_1, ..., A_k\}$ is an alphabet of symbols; each A_i is called a *type*.
  - e.g. $\Sigma = \{Scheduler, Fire\_Truck\}$ means there are two types of objects in that system: *Scheduler* and *Fire_Truck*.



- Each type A_i is associated with a finite automaton $A_i = (\mathcal{S}_i, \delta_i, q_{i0})$.
  - $\mathcal{S}_i = \{S_{i1}, ..., S_{il}\}$ is a finite set of *internal states*,
  $\delta_i \subseteq \mathcal{S}_i \times \mathcal{S}_i$ is the set of the *internal* state transitions, and
  $q_{i0} \in \mathcal{S}_i$ is the initial state of the automaton A_i.

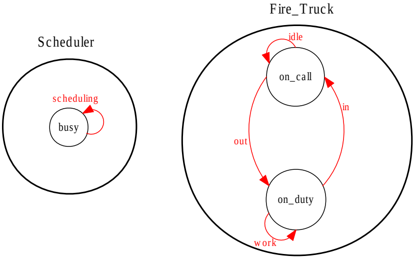


- Each typeA_i is associated with a finite automaton $A_i = (\mathcal{S}_i, \delta_i, q_{i0})$.
  \- $\mathcal{S}_i = \{S_{i1}, ..., S_{il}\}$ is a finite set of *internal states*,
  $\delta_i \subseteq \mathcal{S}_i \times \mathcal{S}_i$ is the set of the *internal* state transitions, and
  $q_{i0} \in \mathcal{S}_i$ is the initial state of the automaton A_i.



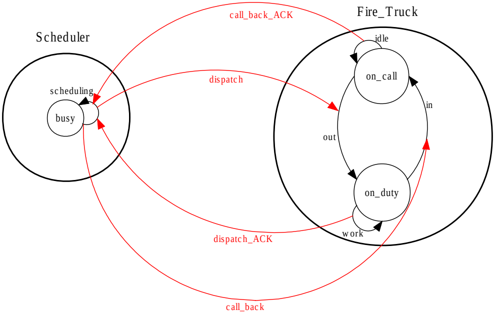
Transitions

- Internal state transition (inside an automaton)
  $t_i : \; S_{iu} \rightarrow S_{iv}$



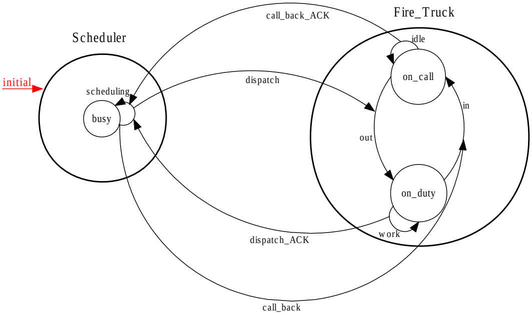
Transitions (contd.)

- External transition (to connect two internal state transitions)
  $r: \ (A_i, t_i) \rightarrow (A_j, t_j)$
  - $t_i \in \delta_i$ and $t_j \in \delta_j$ are internal state transitions.



Initial Type/Active Object

- Initial type: we designate some type as the initial type.
- Active object: initially, we nondeterministically pick an object of the initial type as the active object.

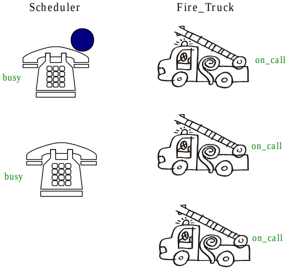


Semantics of Service Automata

Semantics is defined by reachability between configurations.

A configuration is specified by a *collection*, which is a pair $(\mathcal{C}, O)$, where

- $\mathcal{C}$ is a multiset of objects.
- O is the active object, which is a member in $\mathcal{C}$.

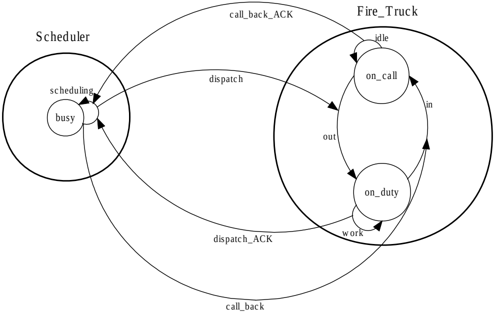


Semantics of Service Automata (contd.)

- One step reachability:
  $(\mathcal{C}, O) \xrightarrow{r} (\mathcal{C}', O')$
  - r is an internal state transition or external transition
  - the collection $(\mathcal{C}, O)$ changes to $(\mathcal{C}', O')$ by firing the transition r
- Transitive closure of one step reachability
  $(\mathcal{C}, O) \rightsquigarrow (\mathcal{C}', O')$
  - $(\mathcal{C}, O)$ can reach $(\mathcal{C}', O')$ by one or more steps.

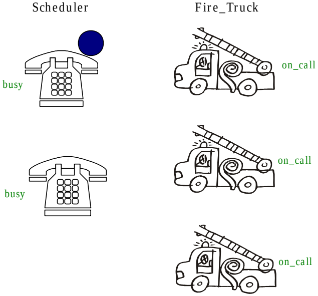
Example of firing transitions

A fire truck scheduling system:
$\Sigma = \{Scheduler, Fire\_Truck\}$ ($Scheduler$ is the initial type)



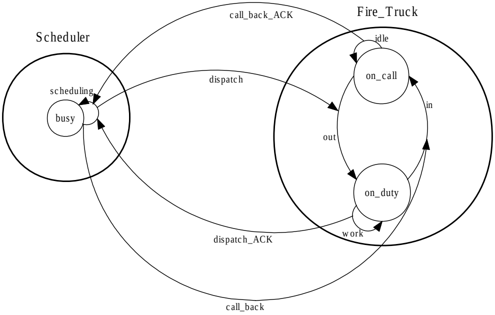
Example of firing transitions

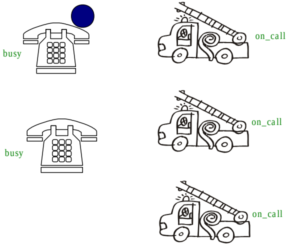
The service automaton is running on a collection $(\mathcal{C}, O)$



Example of firing transitions

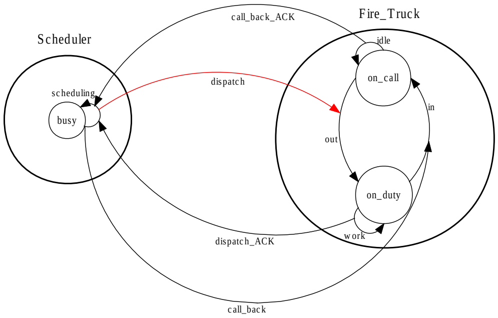
A fire truck scheduling system:

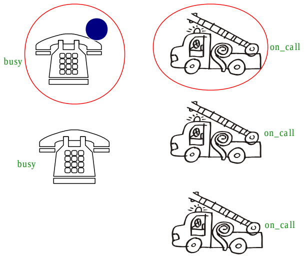




Example of firing transitions

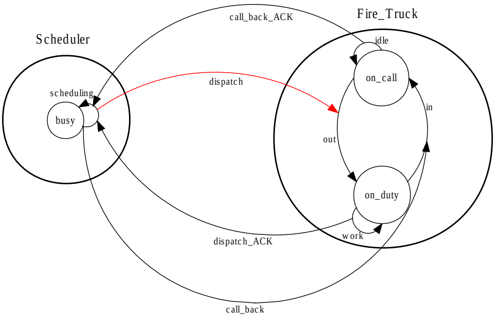
A fire truck scheduling system:

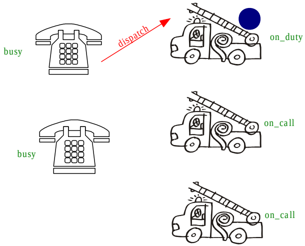




Example of firing transitions

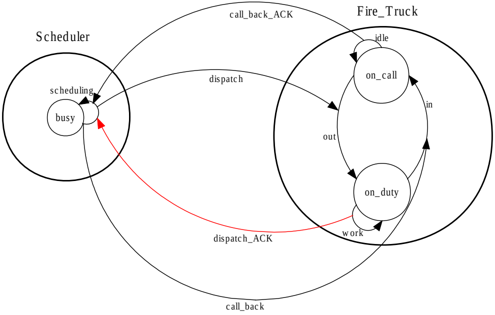
A fire truck scheduling system:

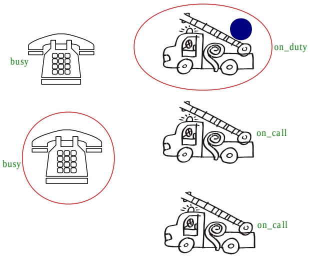




Example of firing transitions

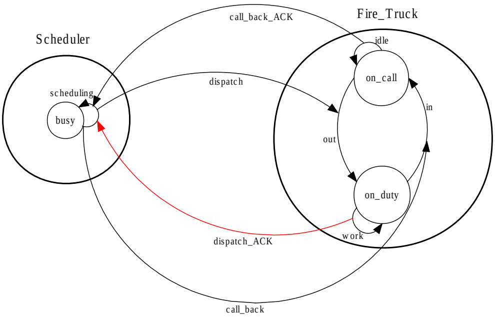
A fire truck scheduling system:

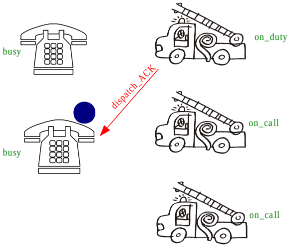




Example of firing transitions

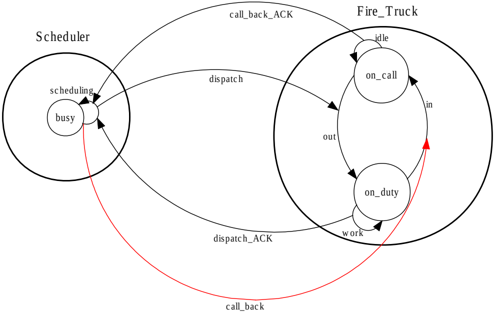
A fire truck scheduling system:

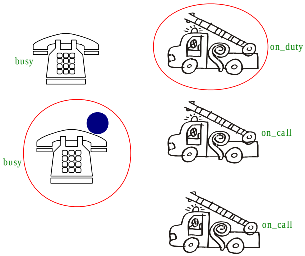




Example of firing transitions

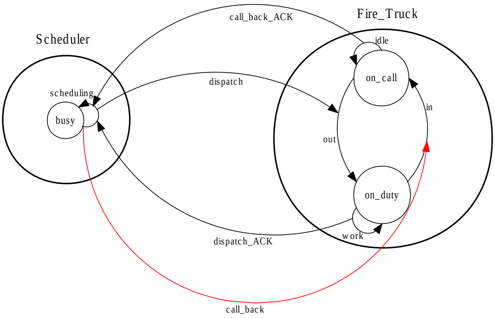
A fire truck scheduling system:

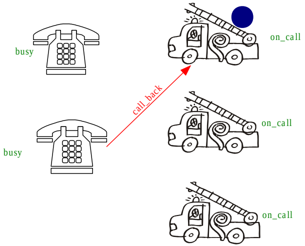




Example of firing transitions

A fire truck scheduling system:





Why are reachability problems important?

- In software engineering, verification problems address issues on whether a system design meets some desired properties.
- A simple but important class of verification queries is reachability, which is about whether a configuration (or a set of configurations) can reach another.
- There are also classes of verification queries beyond reachability, e.g., temporal logics (LTL, CTL).

Presburger Reachability Problem: Presburger Formula

Here, we focus on the Presburger Reachability Problem.

A *Presburger formula P* is a Boolean combination of atomic linear relations and linear congruences.

$$((3x_1 + 2x_2) < 5) \wedge ((x_1 + 3x_2) \equiv_4 2)$$

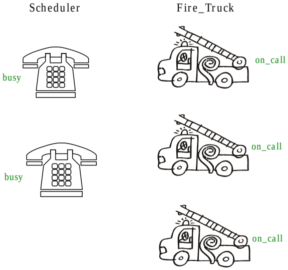
x_1 and x_2 are integer variables.

Presburger Reachability Problem

A multiset of objects $\mathcal{C}$ can actually be viewed as a vector.

Example:

$$(\#_{(\mathit{Schduler},\mathit{busy})}, \#_{(\mathit{Fire\_Truck},\mathit{on\_call})}, \#_{(\mathit{Fire\_Truck},\mathit{on\_duty})}) = (2, 3, 0)$$



Presburger Reachability Problem

Intuitive Version:

Given: a service automaton
a "bad" property

Question: Can the automaton reach a "bad" configuration?

Why is Presburger Reachability Problem important?

A “bad” property can be specified by a Presburger formula P; i.e., $\neg P$ is intended to be true, and hence is a safety property.

Presburger Reachability Problem

A vector $(a_1, \cdots, a_n)$ satisfies a Presburger formula $P(x_1, \cdots, x_n)$ if and only if $P(a_1, \cdots, a_n)$ holds.

Formal Version:

Given: a service automaton G, and a Presburger formula P.

Question: Is there any initial collection that can reach another collection satisfying P?

Presburger Reachability Problem

Vector Addition Systems with States (VASS) = Petri Nets

Theorem

Service automata are equivalent to VASS, and therefore the Presburger reachability problem of service automata is decidable.

This theorem implies that there is an algorithm to automatically check whether a network application specified by a service automaton satisfies a given “bad” property specified by a Presburger formula.

Process

- We assign each external transition with a symbol (could be empty).
- For each run of the service automaton G, we collect all the symbols, and get a sequence of labels for transitions. That sequence is a *process* of G.
- The set $L(G)$ of all processes of G is called *the service* defined by the service automaton G.

Process (contd.)

- Since service automata are equivalent to VASS, and hence services defined by service automata are nonregular.
- Open problem: we currently can not identify a nontrivial subclass of service automata that exactly define regular services.

1-type service automata

1-type service automaton: a service automaton that only has objects of one type; i.e., Σ is of size 1

Theorem

1-type service automata can simulate any service automata, and, therefore, services defined by 1-type service automata are equivalent to those defined by service automata.

Internal-free service automata

Internal-free service automaton: a service automaton without purely internal state transitions; i.e., all internal state transitions are associated with some external transition(s)

Theorem

Any service automaton can be simulated by an internal-free service automaton.

Multiprocess Service Automata

Multiprocess service automata are exactly the same as the (single-process) service automata, except that initially there are multiple active objects, and each active object can simultaneously initiate a process.

Rules

- There are finitely many transition rules in a multiprocess service automaton.
- Each transition rule is in the form of

$$R = \{r_1^{n_1}, \cdots, r_m^{n_m}\},$$

where $n_i \in \mathbb{N} \cup \{*\}$ is the multiplicity of transition r_i.

$$R = \{r_1^*, r_5^2\}$$

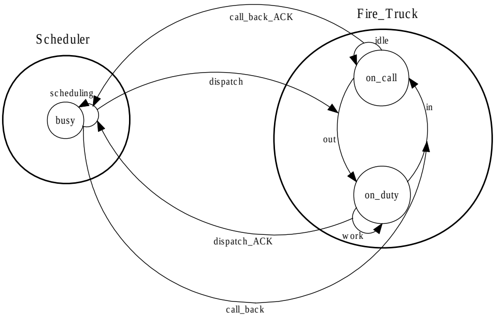
Example

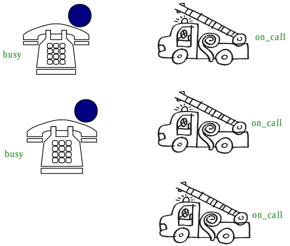
Back to the fire truck scheduling system:

- initially, there are two (the number is nondeterministically chosen) *Scheduler*'s being active.
- $R_1 = \{dispatch^*\}$,
  $R_2 = \{dispatch\_ACK^1\}$,
  $R_3 = \{dispatch^1, dispatch\_ACK^1\}$

Example of firing transitions

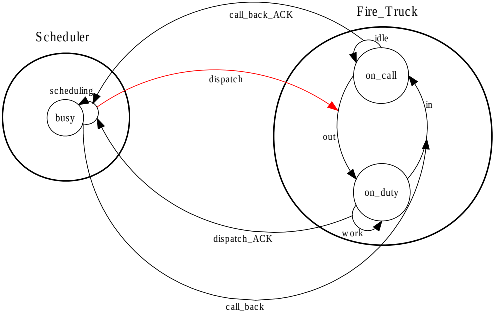
A fire truck scheduling system (Multiprocess):

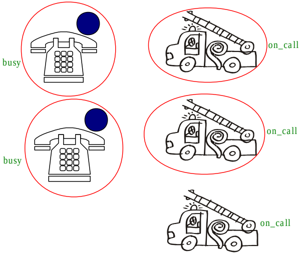




Example of firing transitions

A fire truck scheduling system (Multiprocess):

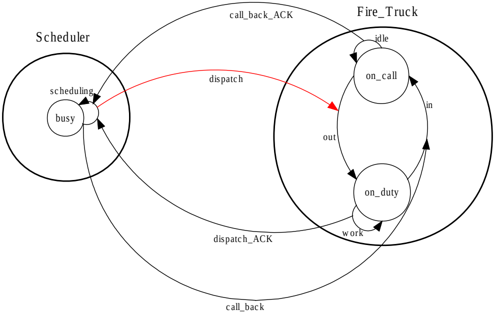


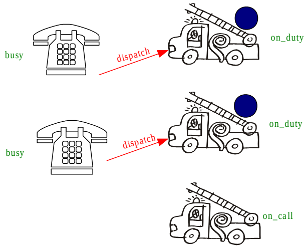


$$R_1 = \{\text{dispatch}^*\}$$

Example of firing transitions

A fire truck scheduling system (Multiprocess):

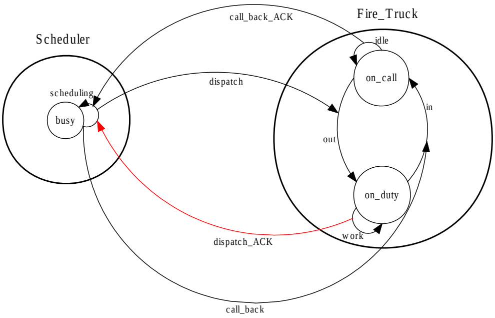


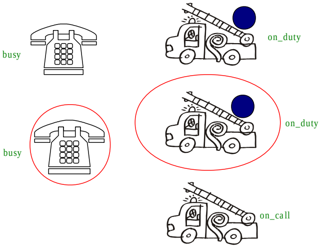


$$R_1 = \{dispatch^*\}$$

Example of firing transitions

A fire truck scheduling system (Multiprocess):

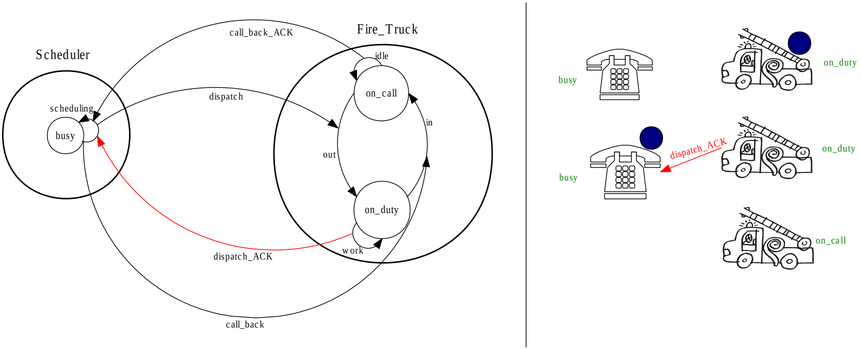




$$R_2 = \{dispatch\_ACK^1\}$$

Example of firing transitions

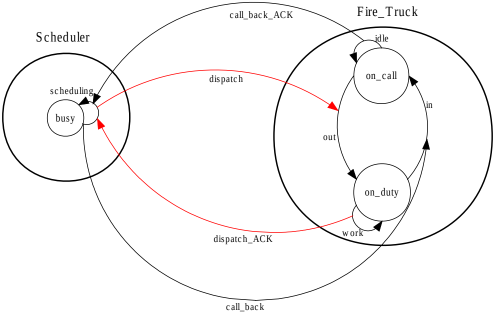
A fire truck scheduling system (Multiprocess):

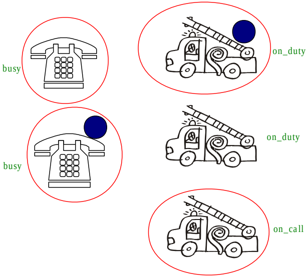


$$R_2 = \{dispatch\_ACK^1\}$$

Example of firing transitions

A fire truck scheduling system (Multiprocess):

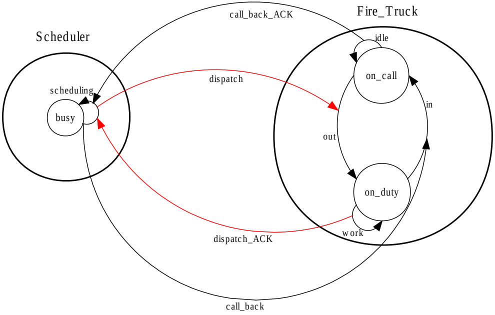


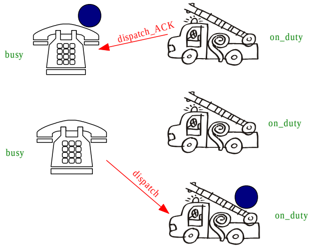


$$R_3 = \{dispatch^1, dispatch\_ACK^1\}$$

Example of firing transitions

A fire truck scheduling system (Multiprocess):





$$R_3 = \{dispatch^1, dispatch\_ACK^1\}$$

Theorem

The Presburger reachability problem for multiprocess service automata is undecidable.

Proof idea: by reduction to a class of quadratic Diophantine equation system, whose solvability is undecidable (G. Xie, Z. Dang, and O. H. Ibarra, *ICALP'03*).

Open problem: we currently can not identify a nontrivial subclass of multiprocess service automata whose Presburger reachability problem is decidable.

P systems

Service automata can be treated as a variation of P systems. P systems were initiated by Gheorghe Paun eight years ago. A P system is:

- an unconventional computing model motivated from natural phenomena of cell evolutions and chemical reactions
- multisets of objects are placed in regions of the membrane structure
- membranes are organized as a Venn diagram or a tree structure

Service automata and P systems

Service automata can be translated to P systems.
An external transition that connects from the internal transition $q \longrightarrow q'$ in an automaton of type A to the internal transition $p \longrightarrow p'$ in an automaton of type B can be depicted in a P system rule in the following form:

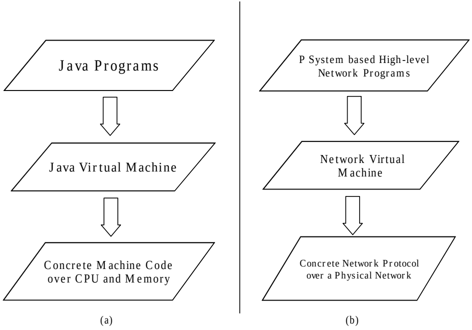
$$\bar{A}_q B_p \longrightarrow A_{q'} \bar{B}_{p'}$$

where the symbol objects $\bar{A}_q$ and $\bar{B}_{p'}$ indicate the active objects.

Service automata and P systems (contd.)

We introduce the notion of “processes”, which is extremely important in analyzing behaviors of network service applications. Such a notion of processes may also be applicable to other classes of P systems.

A comparison of Java and P systems based high level network programs



A detailed mechanism of part (b) can be referred to Yong Wang's PhD dissertation:
Clustering, grouping, and process over networks, Washington State University, 2007.

Thanks!

Questions?