Dipartimento di
Informatica e Sistemistica
"Antonio Ruberti"
SAPIENZA
Università di Roma

# Transition Systems and Service Composition

Giuseppe De Giacomo

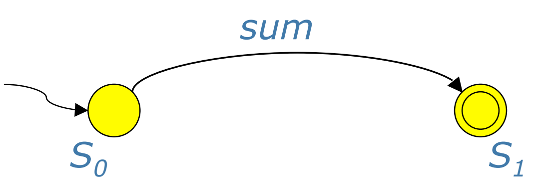
Seminari di Ingegneria del Software
A.A. 2006/2007

Dipartimento di
Informatica e Sistemistica
"Antonio Ruberti"
SAPIENZA
Università di Roma

## *Transition Systems*

# Concentrating on behaviors: SUM two integers

- Consider a program for computing the sum of two integers.
- Such a program has essentially two states
  - the state $S_0$ of the memory before the computation: including the two number to sum
  - the state $S_1$ of the memory after the computation: including the result of the computation
- Only one action, i.e. "*sum*", can be performed



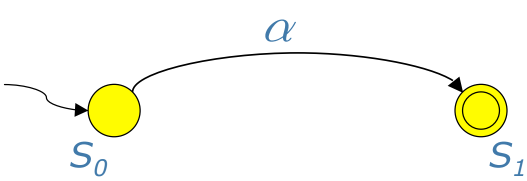



# Concentrating on behaviors: CheckValidity

- Consider a program for computing the validity of a FOL formula:
- Also such a program has essentially two states
  - the state $S_1$ of the memory before the computation: including the formula to be checked
  - the state $S_2$ of the memory after the computation: including "yes", "no", "time-out"
- Only one action, i.e. "*checkValidity*", can be performed

# Concentrating on behaviors

- The programs SUM and CheckValidity are very different from a computational point of view.
  - SUM is trivial
  - CheckValidity is a theorem prover hence very complex
- However they are equally trivial from a behavioral point of view:
  - two states $S_1$ and $S_2$
  - a single action $\alpha$ causing the transition

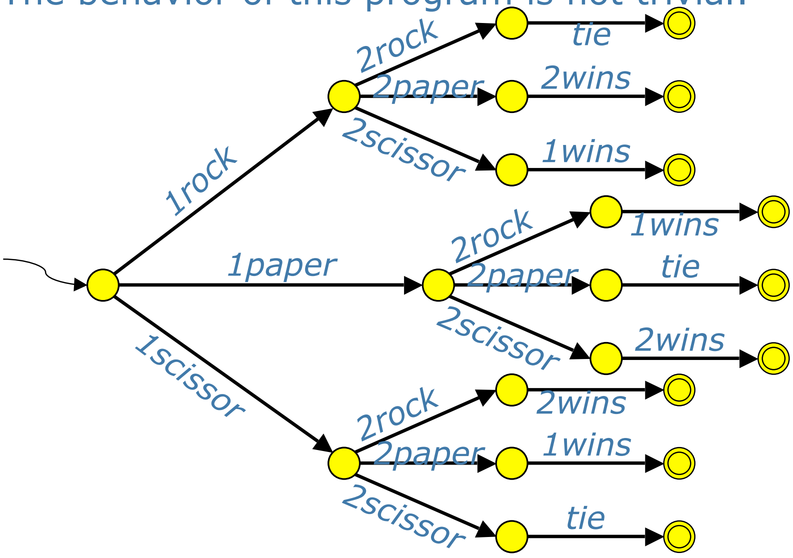





Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

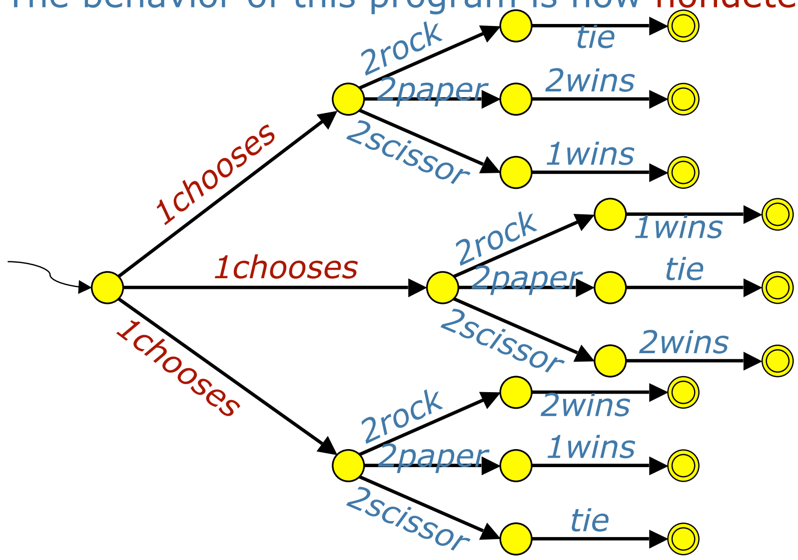
# Concentrating on behaviors: RockPaperScissor

- Consider the program RockPaperScissor that allows to play two players the the well-known game.
- The behavior of this program is not trivial:



# Concentrating on behaviors: RockPaperScissor (automatic)

- Consider a variant of the program RockPaperScissor that allows one players to play against the computer.
- The behavior of this program is now nondeterministic:



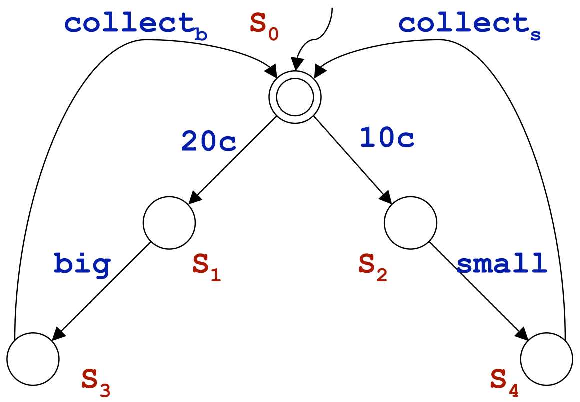



# Concentrating on behaviors: WebPage

http://www.informatik.uni-trier.de/~ley/db/

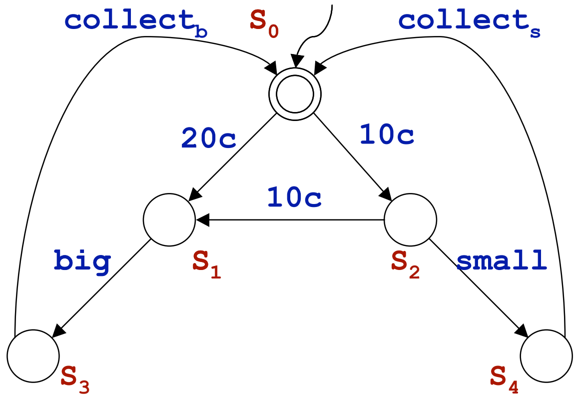
A web page can have a complex behavior!

# Concentrating on behaviors: Vending Machine

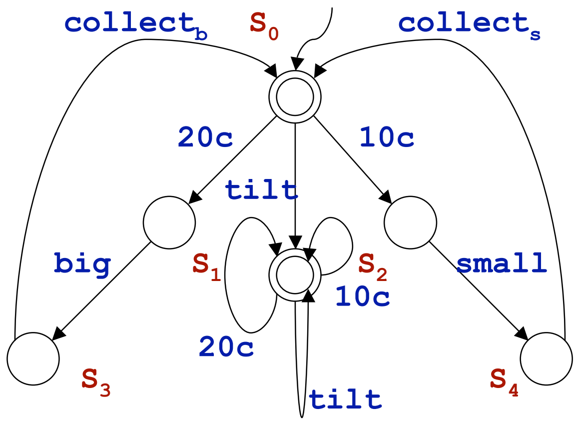





# Concentrating on behaviors: Another Vending Machine



# Concentrating on behaviors: Vending Machine with Tilt






# Transition Systems

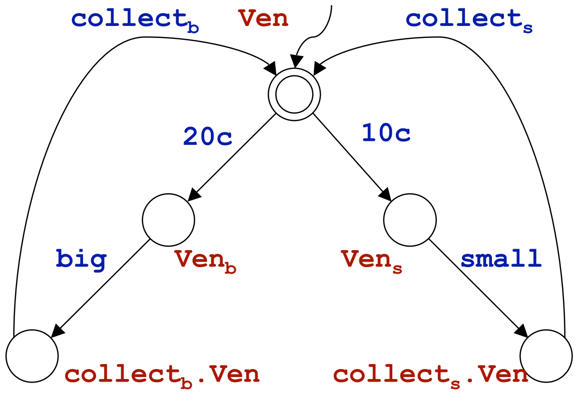
- A transition system TS is a tuple $T = < A, S, S^0, \delta, F>$ where:
  - A is the set of actions
  - S is the set of states
  - $S^0 \subseteq S$ is the set of initial states
  - $\delta \subseteq S \times A \times S$ is the transition relation
  - $F \subseteq S$ is the set of final states

- Variants:
  - No initial states
  - Single initial state
  - Deterministic actions
  - States labeled by propositions other than Final/¬Final

*(c.f. Kripke Structure)*

# Process Algebras are Formalisms for Describing TS

- Trans (a la CCS)
  - $Ven = 20c.Ven_b + 10c.Ven_s$
  - $Ven_b = big.collect_b.Ven$
  - $Ven_l = small.collect_s.Ven$
- Final
  - √ Ven

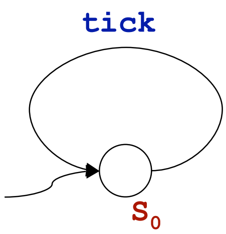



- *TS may have infinite states - e.g., this happens when generated by process algebras involving iterated concurrency*
- *However we have good formal tools to deal only with finite states TS*



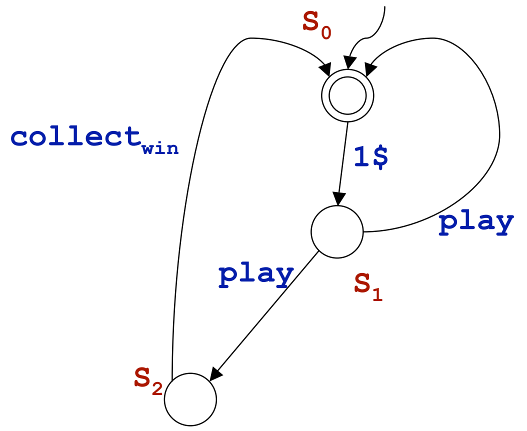
# Example (Clock)

TS may describe (legal) nonterminating processes



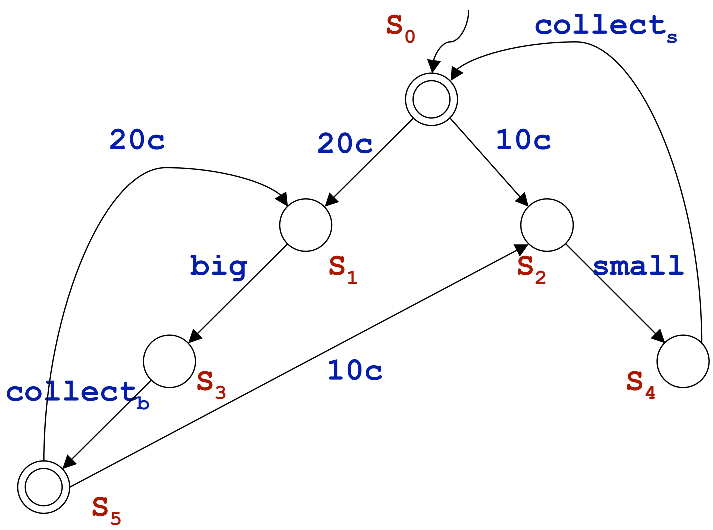
# Example (Slot Machine)

Nondereminisic transitions express
choice that is not under the control of clients

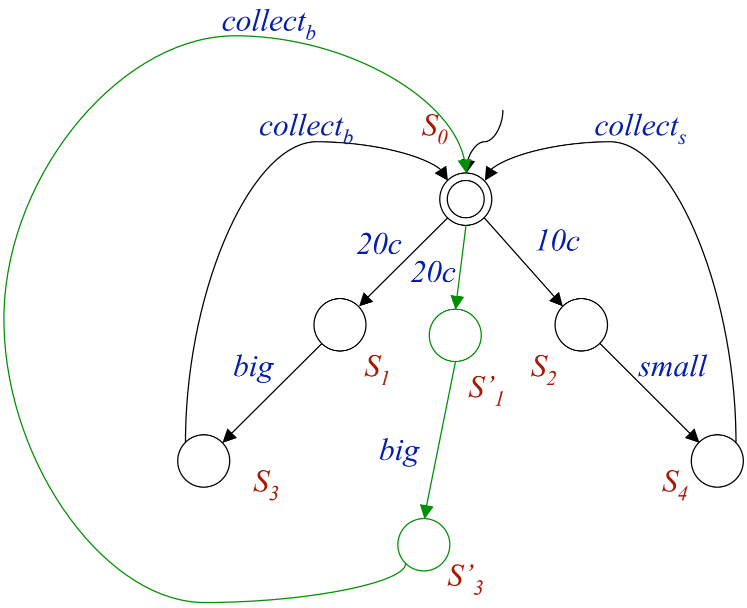





# Example (Vending Machine - Variant 1)



# Example
# (Vending Machine - Variant 2)






# Inductive vs Coinductive Definitions: Reachability, Bisimilarity, …

# Reachability

- A binary relation R is a **reachability-like relation** iff:
  - $(s,s) \in R$
  - if $\exists$ a. s'. $s \rightarrow_a s' \wedge (s',s'') \in R$ then $(s,s'') \in R$
- A state s_0 of transition system S is **reachable-from** a state $s_f$ iff for **all** a **reachability-like relations** R we have $(s_0, s_f) \in R$.
- Notably that
  - **reachable-from** is a reachability-like relation itself
  - **reachable-from** is the smallest reachability-like relation

*Note it is a inductive definition!*



# Computing Reachability on Finite Transition Systems

**Algorithm** ComputingReachability

**Input:** transition system TS
**Output:** the **reachable-from** relation (the smallest reachability-like relation)

**Body**

```
    R = ∅
    R' = {(s,s) | s ∈ S}
    while (R ≠ R') {
            R := R'
            R' := R' ∪ {(s,s'') | ∃ s',a. s →a s' ∧ (s',s'')∈ R }
    }
    return R'
```

**YdoB**

# Bisimulation

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- A binary relation *R* is a **bisimulation** iff:

  (s,t) $\in$ *R* implies that
  - s is *final* iff t is *final*
  - for all actions a
    - if s $\to_a$ s' then $\exists$ t' . t $\to_a$ t' and (s',t')$\in$ *R*
    - if t $\to_a$ t' then $\exists$ s' . s $\to_a$ s' and (s',t')$\in$ *R*

- A state $s_0$ of transition system S is **bisimilar**, or simply **equivalent,** to a state $t_0$ of transition system T iff there **exists** a **bisimulation** between the initial states $s_0$ and $t_0$.

- Notably
  - **bisimilarity** is a bisimulation
  - **bisimilarity** is the largest bisimulation

*Note it is a co-inductive definition!*



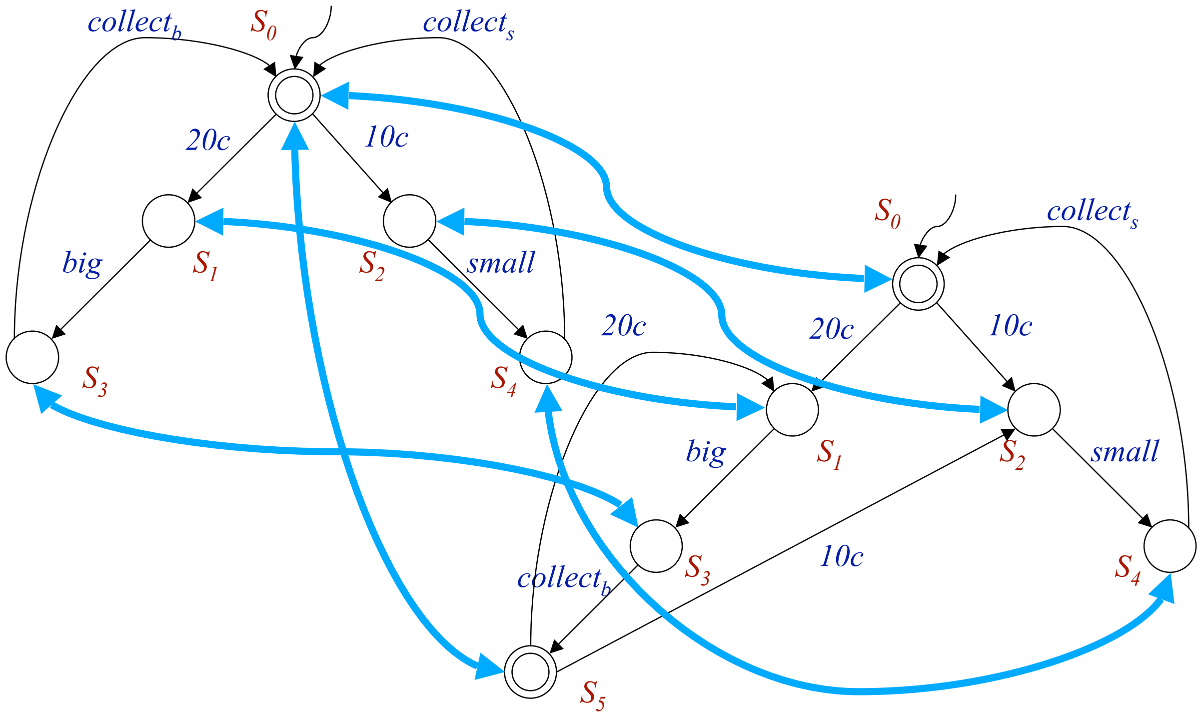
# Computing Bisimilarity on Finite Transition Systems

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

**Algorithm** ComputingBisimulation
**Input:** transition system $TS_S$ = < A, S, $S^0$, $\delta_S$, $F_S$> and
transition system $TS_T$ = < A, T, $T^0$, $\delta_T$, $F_T$>
**Output:** the **bisimilarity** relation (the largest bisimulation)

**Body**
```
R = ∅
R' = S × T - {(s,t) | ¬(s ∈ F_S ≡ t ∈ F_T)}
while (R ≠ R') {
        R := R'
        R' := R' - ({(s,t) | ∃ s',a. s →_a s' ∧ ¬∃ t' . t →_a t' ∧ (s',t') ∈ R' }
                    {(s,t) | ∃ t',a. t →_a t' ∧ ¬∃ s' . s →_a s' ∧ (s',t') ∈ R' })
}
return R'
```
**Ydob**

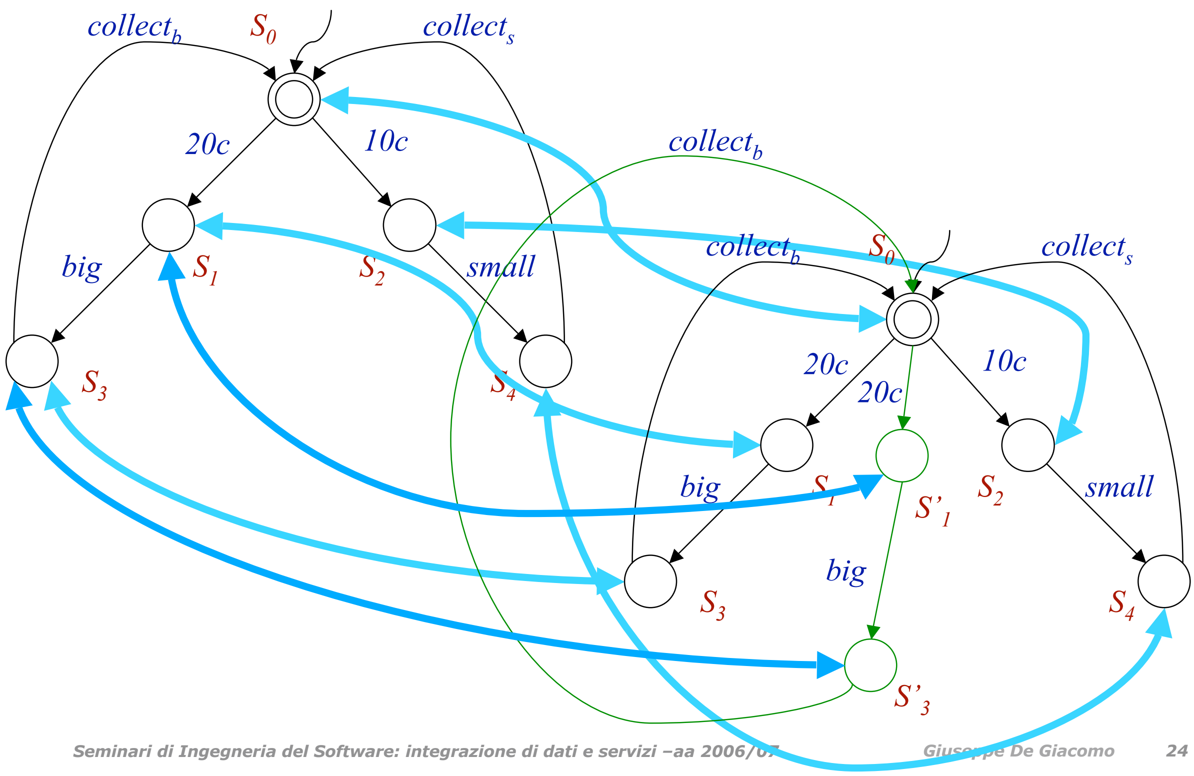
# Example of Bisimulation






Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

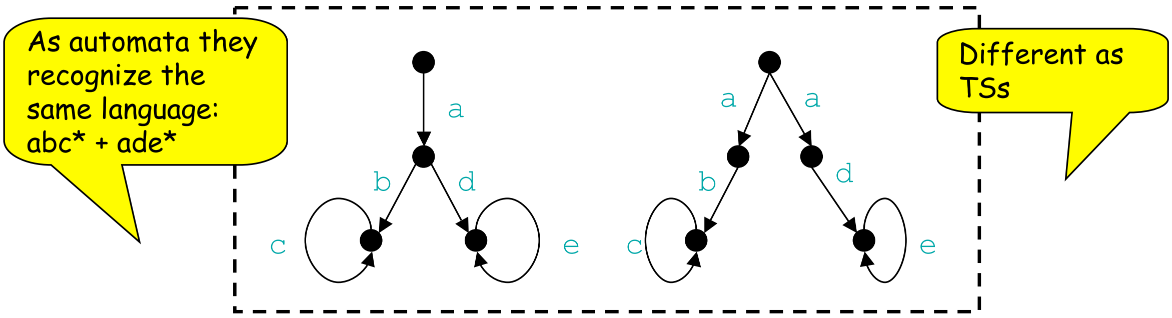
# Example of Bisimulation



## Automata vs.Transition Systems

Dipartimento di Informatica e Sistemistica "Antonio Ruberti" SAPIENZA Università di Roma

- Automata
  - define sets of runs (or traces or strings): (finite) length sequences of actions
- TSs
  - … but I can be interested also in the alternatives "encountered" during runs, as they represent client's "choice points"






Dipartimento di Informatica e Sistemistica "Antonio Ruberti" SAPIENZA Università di Roma

# Logics of Programs

## Logics of Programs

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- Are modal logics that allow to describe properties of transition systems
- Examples:
  - HennesyMilner Logic
  - Propositional Dynamic Logics
  - Modal (Propositional) Mu-calculus
- Perfectly suited for describing transition systems: they can tell apart transition systems modulo bisimulation



## HennessyMilner Logic

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- $\Phi$ := P | *(atomic propositions)*
  $\neg \Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid$ *(closed under boolean operators)*
  $[a]\Phi \mid \langle a \rangle\Phi$ *(modal operators)*
- Propositions are used to denote final states
- $\langle a \rangle\Phi$ means there exists an a-transition that leads to a state where $\Phi$ holds; i.e., expresses the capability of executing action a bringing about $\Phi$
- $[a]\Phi$ means that all a-transitions lead to states where $\Phi$ holds; i.e., express that executing action a brings about $\Phi$

# Logics of Programs: Examples

- Usefull abbreviation:
  - $\langle \text{any} \rangle\, \Phi$ stands for $\langle a_1 \rangle \Phi \vee \cdots \vee \langle a_n \rangle \Phi$
  - $[\text{any}]\, \Phi$ stands for $[a_1]\Phi \wedge \cdots \wedge [a_n]\Phi$
  - $\langle \text{any} - a_1 \rangle\, \Phi$ stands for $\langle a_2 \rangle \Phi \vee \cdots \vee \langle a_v \rangle \Phi$
  - $[\text{any} - a_1]\, \Phi$ stands for $[a_2]\Phi \wedge \cdots \wedge [a_v]\Phi$

- Examples:
  - $\langle a \rangle \text{true}$ *cabability of performing action a*
  - $[a]\text{false}$ *inability of performing action a*
  - $\neg\text{Final} \wedge \langle \text{any} \rangle \text{true} \wedge [\text{any-}a]\text{false}$
    *necessity/inevitability of performing action a*
    *(i.e., action a is the only action possible)*
  - $\neg\text{Final} \wedge [\text{any}]\text{false}$ *deadlock!*



# Propositional Dynamic Logic

- $\Phi := P \mid$ *(atomic propositions)*
  $\neg\, \Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid$ *(closed under boolean operators)*
  $[r]\Phi \mid \langle r \rangle \Phi$ *(modal operators)*

  $r := a \mid r_1 + r_2 \mid r_1 ; r_2 \mid r^* \mid P?$ *(complex actions as regular expressions)*

- Essentially add the capability of expressing partial correctness assertions via formulas of the form
  - $\Phi_1 \rightarrow [r]\Phi_2$ *under the conditions $\Phi_1$ all possible executions of r that terminate reach a state of the TS where $\Phi_2$ holds*

- Also add the ability of asserting that a property holds in all nodes of the transition system
  - $[(a_1 + \cdots + a_v)^*]\Phi$ *in every reachable state of the TS $\Phi$ holds*

- Useful abbereviations:
  - any stands for $(a_1 + \cdots + a_v)$ *Note that + can be expressed also in HM Logic*
  - u stands for any* *This is the so called master/universal modality*

# Modal Mu-Calculus

- $\Phi := P \mid$ *(atomic propositions)*
  $\neg \Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid$ *(closed under boolean operators)*
  $[r]\Phi \mid \langle r \rangle \Phi$ *(modal operators)*

  $\mu X.\Phi(X) \mid \nu X.\Phi(X)$ *(fixpoint operators)*

- It is the most expressive logic of the family of logics of programs.
- It subsumes
  - PDL (modalities involving complex actions are translated into fomulas involving fixpoints)
  - LTL (linear time temporal logic),
  - CTS, CTS* (branching time temporal logics)

- Examples:
- $[any^*]\Phi$ can be expressed as $\nu X.\ \Phi \wedge [any]X$

- $\mu X.\ \Phi \vee [any]X$ *along all runs eventually $\Phi$*
- $\mu X.\ \Phi \vee \langle any \rangle X$ *along some run eventually $\Phi$*
- $\nu X.\ [a](\mu Y.\ \langle any \rangle true \wedge [any\text{-}b]Y) \wedge X$
  *every run that that contains a contains later b*



# Model Checking

- Model checking is polynomial in the size of the TS for
  - HennessyMilner Logic
  - PDL
  - Mu-Calculus
- Also model checking is wrt the formula
  - Polynomial for HennessyMiner Logic
  - Polynomial for PDL
  - Polynomial for Mu-Calculus with bounded alternation of fixpoints and NP∩coNP in general

## Model Checking

- Given a TS T, one of its states s, and a formula $\Phi$ verify whether the formula holds in s. Formally:

$$T,s \models \Phi$$

- Examples (TS is our vending machine):
  - $S_0 \models$ Final
  - $S_0 \models$ <10c>true — *capability of performing action 10c*
  - $S_2 \models$ [big]false — *inability of performing action big*
  - $S_0 \models$ [10c][big]false — *after 10c cannot execute big*
  - $S_i \models \mu X.\ Final \vee [any]\ X$ — *eventually a final state is reached*
  - $S_0 \models \nu Z.\ (\mu X.\ Final \vee [any]\ X) \wedge [any]\ Z$ — *or equivalently*
    $S_0 \models [any^*](\mu X.\ Final \vee [any]\ X)$
    *from everywhere eventually final*



## AI Planning as Model Checking

- **Build the TS of the domain:**
  - Consider the set of states formed all possible truth value of the propositions (this works only for propositional setting).
  - Use Pre's and Post of actions for determining the transitions

  *Note: the TS is exponential in the size od the description.*

- **Write the goal in a logic of program**
  - typically a single least fixpoint formula of Mu-Calculus (compute ***reachable*** states intersection states where goal true)

- **Planning:**
  - model check the formula on the TS starting from the given initial state.
  - use the path (paths) used in the above model checking for returning the plan.

- *This basic technique works only when we have complete information (or at least total observability on state):*
  - *Sequential plans if initial state known and actions are deterministic*
  - *Conditional plans if many possible initial states and/or actions are nondeterministic*

# Example

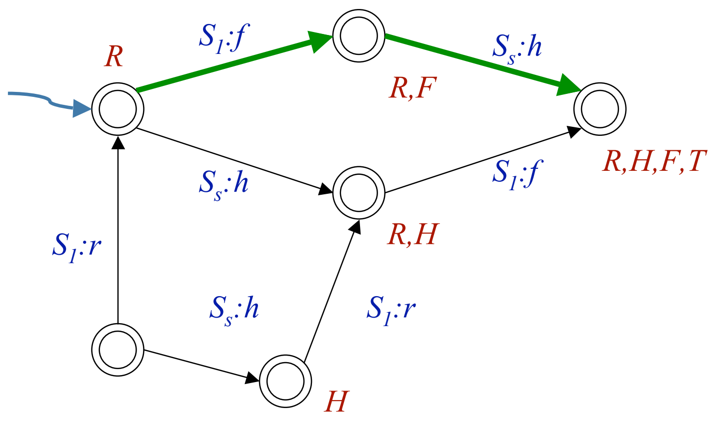
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- Operators (Services + Mappings)
  - Registered ∧ ¬FlightBooked → [$S_1$:bookFlight] FlightBooked
  - ¬Registered → [$S_1$:register] Registered
  - ¬HotelBooked → [$S_2$:bookHotel] HotelBooked
- Additional constraints (Community Ontology):
  - TravelSettledUp ≡
    FlightBooked ∧ HotelBooked ∧ EventBooked
- Goals (Client Service Requests):
  - Starting from state
    Registered ∧ ¬FlightBooked ∧ ¬ HotelBooked ∧ ¬EventBooked
    check <any*>TravelSettedUp
  - Starting from all states such that
    ¬FlightBooked ∧ ¬ HotelBooked ∧ ¬EventBooked
    check <any*>TravelSettledUp



# Example

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

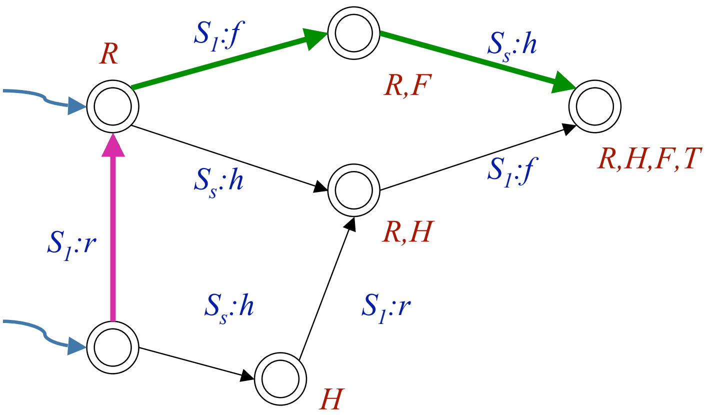



Plan:
S_1:bookFlight;
S_2:bookHotel

Starting from state
Registered ∧ ¬ FlightBooked ∧ ¬ HotelBooked ∧ ¬ EventBooked
check
<any*>TravelSettledUp

## Example






Plan:
if(¬Registered) {
$S_1$:register;
}
$S_1$:bookFlight;
$S_2$:bookHotel

Starting from states where
¬ FlightBooked ∧ ¬ HotelBooked ∧ ¬ EventBooked
check
<any*>TravelSettledUp



## Satisfiability



- Observe that a formula $\Phi$ may be used to select among all TS T those such that for a given state s we have that T,s $\models \Phi$
- SATISFIABILITY: Given a formula $\Phi$ verify whether there exists a TS T and a state s such that. Formally:

  check whether exists T, s such that T,s $\models \Phi$

- Satisfiability is:
  - PSPACE for HennesyMilner Logic
  - EXPTIME for PDL
  - EXPTIME for Mu-Calculus

# References


[Stirling Banff96] C. Stirling: Modal and temporal logics for processes. Banff Higher Order Workshop LNCS 1043, 149-237, Springer 1996

[Bradfield&Stirling HPA01] J. Bradfield, C. Stirling: Modal logics and mu-calculi. Handbook of Process Algebra, 293-332, Elsevier, 2001.

[Stirling 2001] C. Stirling: Modal and Temporal Properties of Processes. Texts in Computer Science, Springer 2001

[Kozen&Tiuryn HTCS90] D. Kozen, J. Tiuryn: Logics of programs. Handbook of Theoretical Computer Science, Vol. B, 789–840. North Holland, 1990.

[HKT2000] D. Harel, D. Kozen, J. Tiuryn: Dynamic Logic. MIT Press, 2000.

[Clarke& Schlingloff HAR01] E. M. Clarke, B. Schlingloff: Model Checking. Handbook of Automated Reasoning 2001: 1635-1790

[CGP 2000] E.M. Clarke, O. Grumberg, D. Peled: Model Checking. MIT Press, 2000.

[Emerson HTCS90] E. A. Emerson. Temporal and Modal Logic. Handbook of Theoretical Computer Science, Vol B: 995-1072. North Holland, 1990.

[Emerson Banff96] E. A. Emerson. Automated Temporal Reasoning about Reactive Systems. Banff Higher Order Workshop, LNCS 1043, 111-120, Springer 1996

[Vardi CST] M. Vardi: Alternating automata and program verification. Computer Science Today -Recent Trends and Developments, LNCS Vol. 1000, Springer, 1995.

[Vardi etal CAV94] M. Vardi, O. Kupferman and P. Wolper: An Automata-Theoretic Approach to Branching-Time Model Checking (full version of CAV'94 paper).

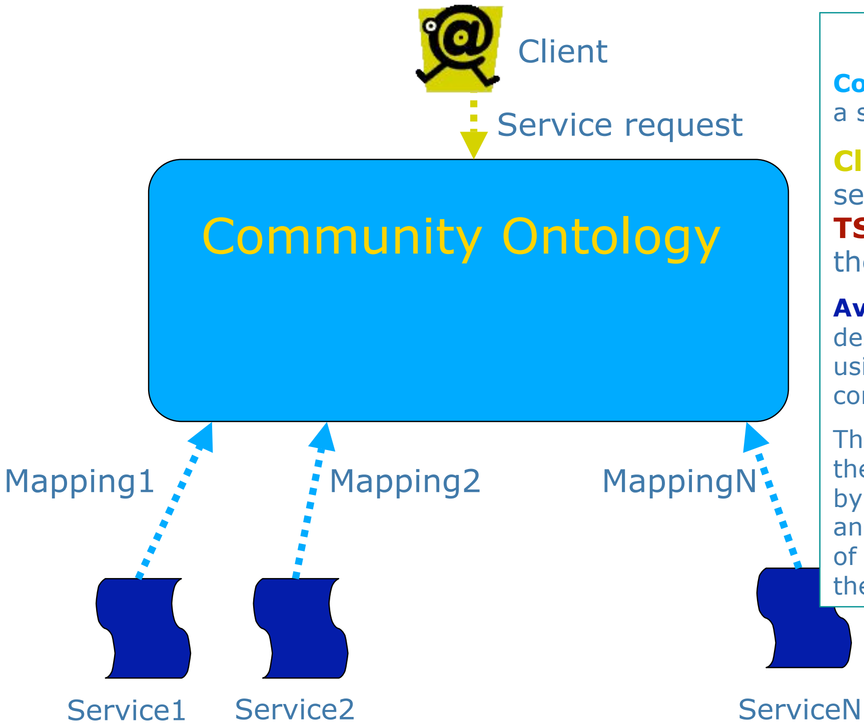
[Schneider 2004] K. Schenider: Verification of Reactive Systems, Springer 2004.




Name by **Rick Hull**

# Composition: the "Roman" Approach

## The Roman Approach




**Client-tailored!**

**Community ontology:** just a set of **actions**

**Client** formulates the service it requires as a **TS** using the **actions** of the common ontology

**Available services:** described in terms of a **TS** using **actions** of the community ontology

The **community** realizes the **client's target service** by "reversing" the mapping and hence using **fragments** of the computation of the the **available services**



## Community of Services

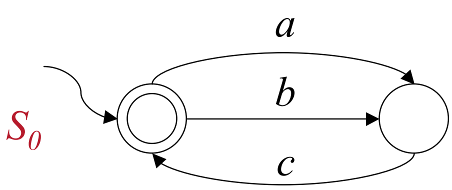
- A community of Services is
  - a set of services …
  - … that share implicitly a *common understanding* on a common set of actions (common ontology limited to the alphabet of actions)…
  - … and export their behavior using (finite) TS over this common set of actions
- A client specifies needs as a service behavior, i.e, a (finite) TS using the common set of actions of the community

# (Target & Available) Service TS

- We model services as finite TS  T = (Σ, S, $s^0$, δ, F) with
  - single initial state *($s^0$ )*
  - deterministic transitions *(i.e., δ is a partial function from S×Σ to S)*

*Note: In this way the client entirely controls/chooses the transition to execute*

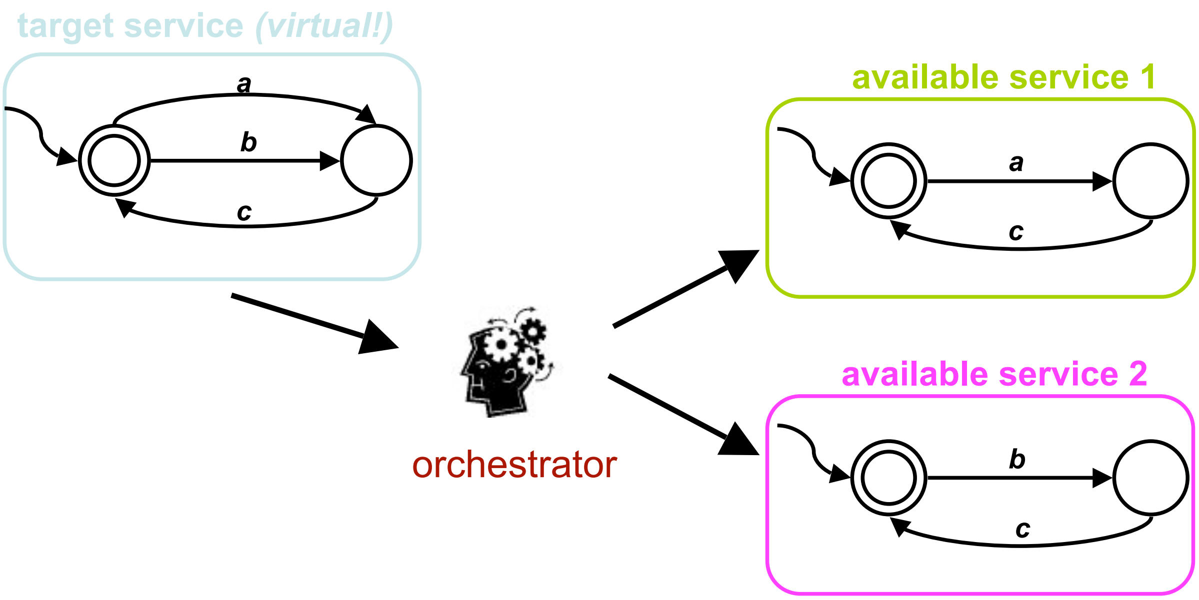
*Example:*




*a: "search by author (and select)"*
*b: "search by title (and select)"*
*c: "listen (the selected song)"*

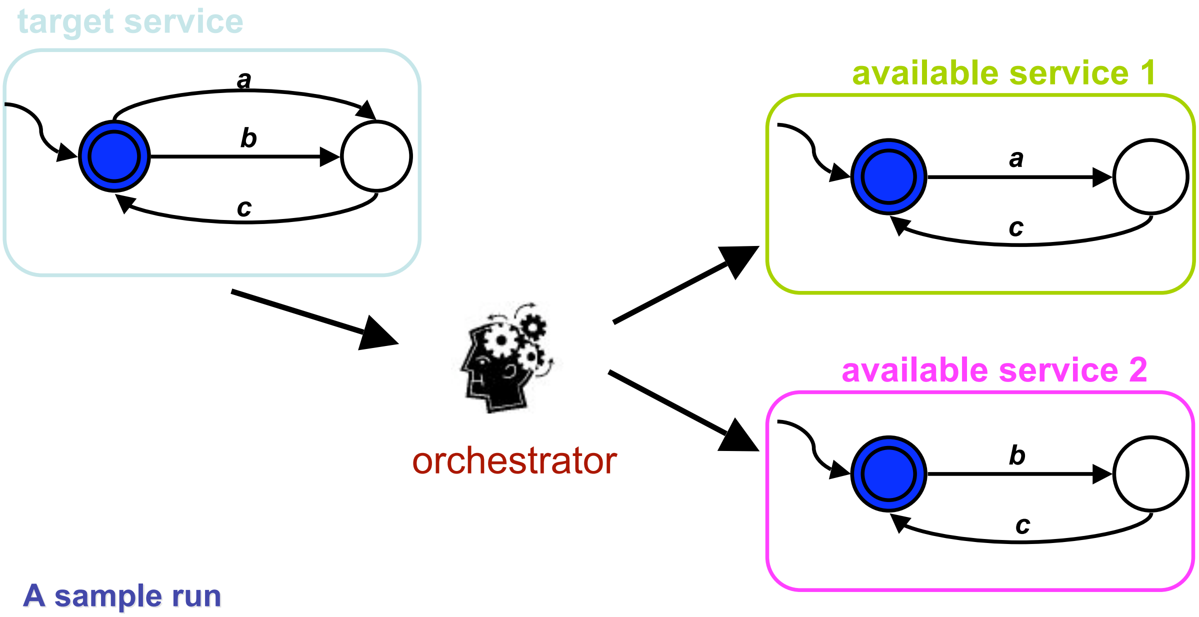


# Composition: an Example




*Lets get some intuition of what a composition is through an example*

# Composition: an Example



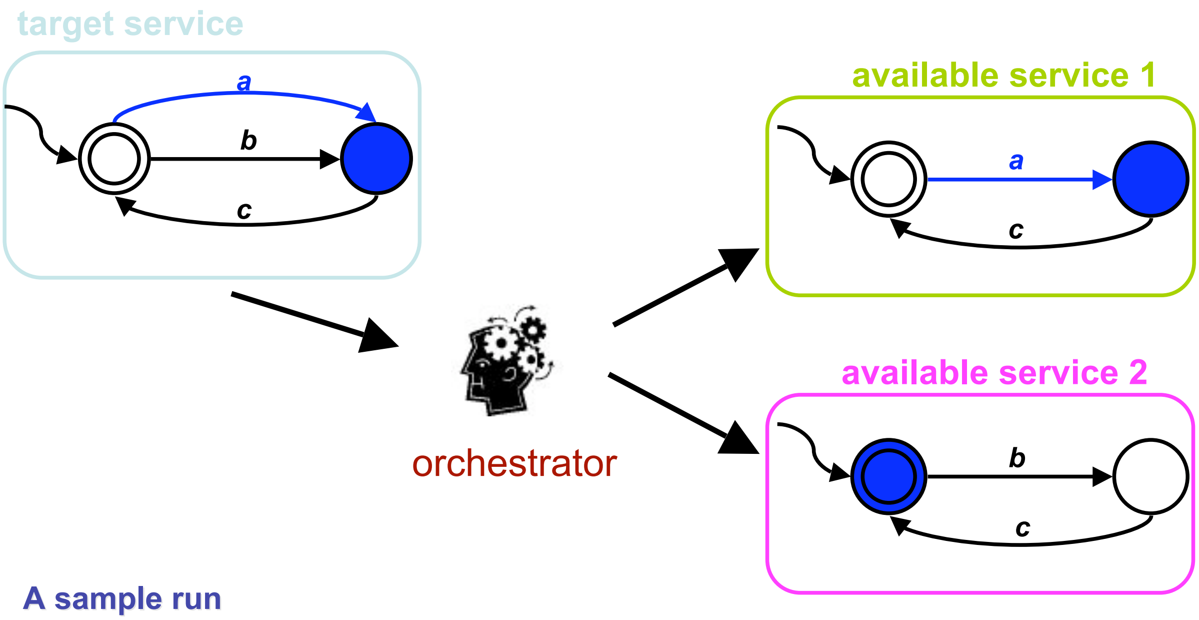

**A sample run**

action request:

orchestrator response:



# Composition: an Example

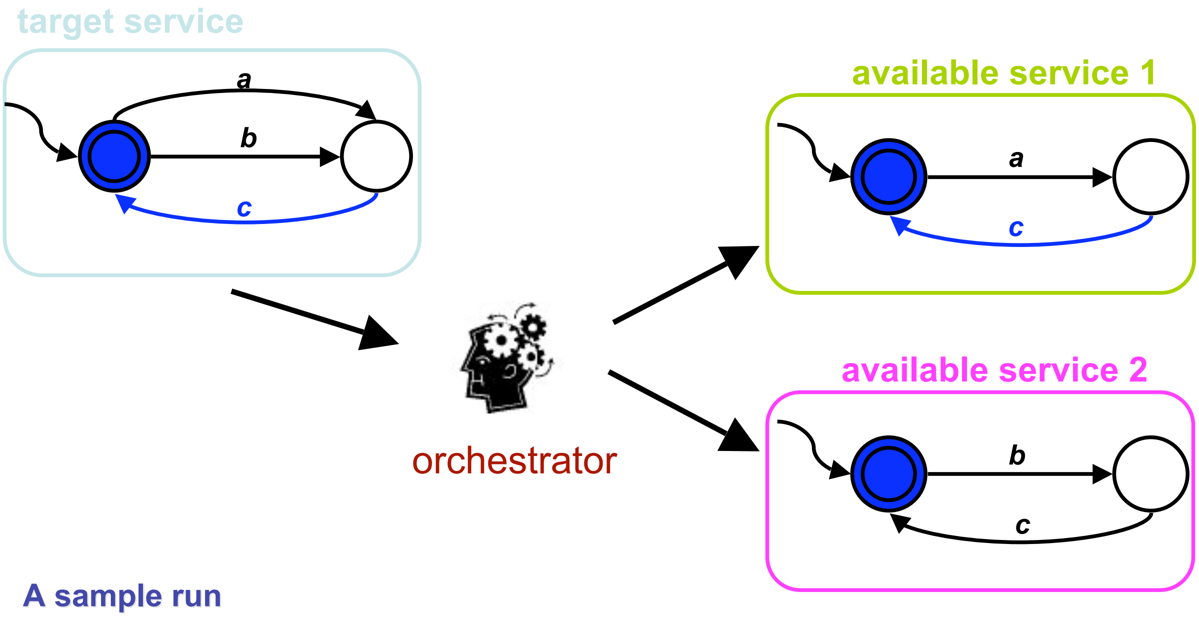



**A sample run**

action request: *a*

orchestrator response: *a,1*

# Composition: an Example



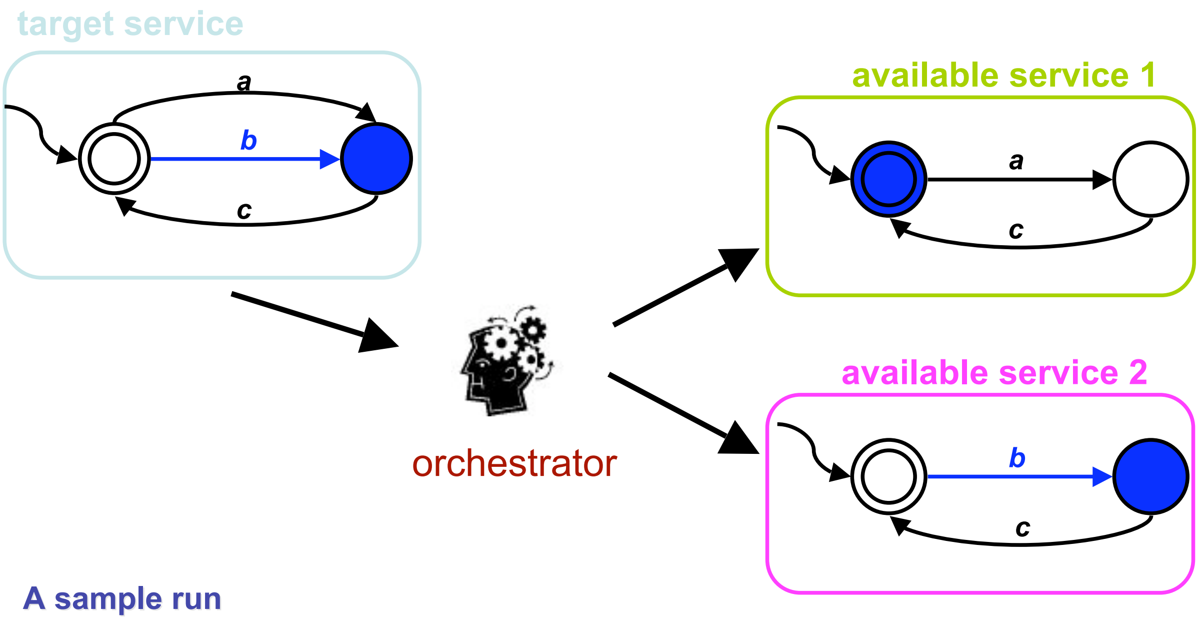

**A sample run**

| | | |
|---|---|---|
| **action request:** | *a* | *c* |
| **orchestrator response:** | ***a,1*** | ***c,1*** |





# Composition: an Example

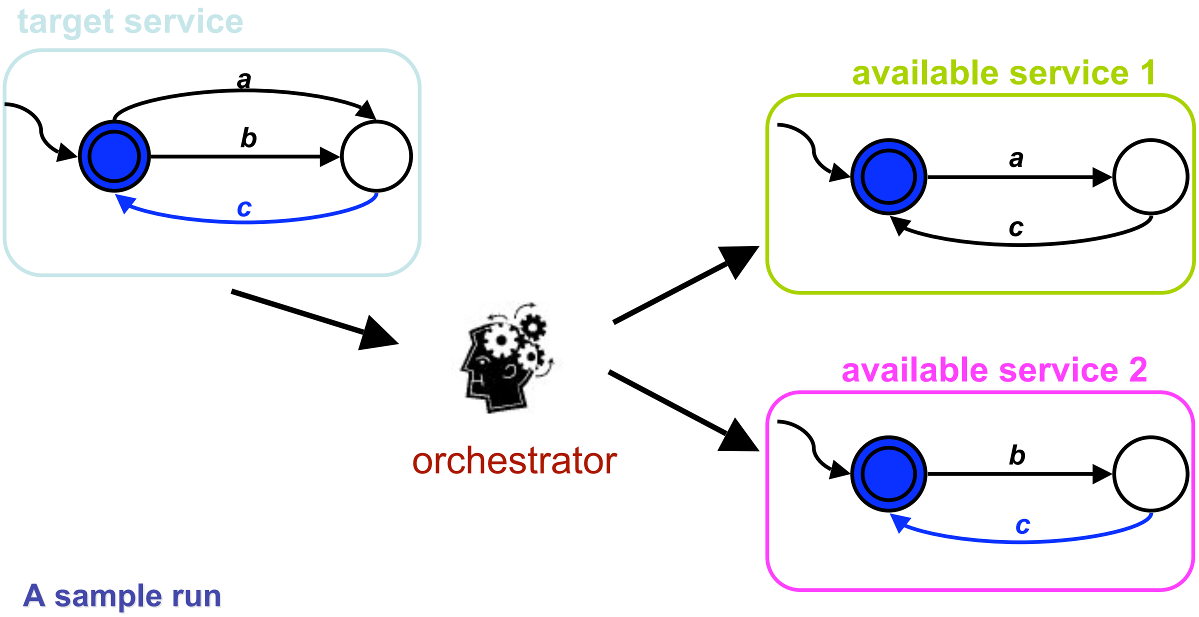



**A sample run**

| | | | |
|---|---|---|---|
| **action request:** | *a* | *c* | *b* |
| **orchestrator response:** | ***a,1*** | ***c,1*** | ***b,2*** |

# Composition: an Example

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma



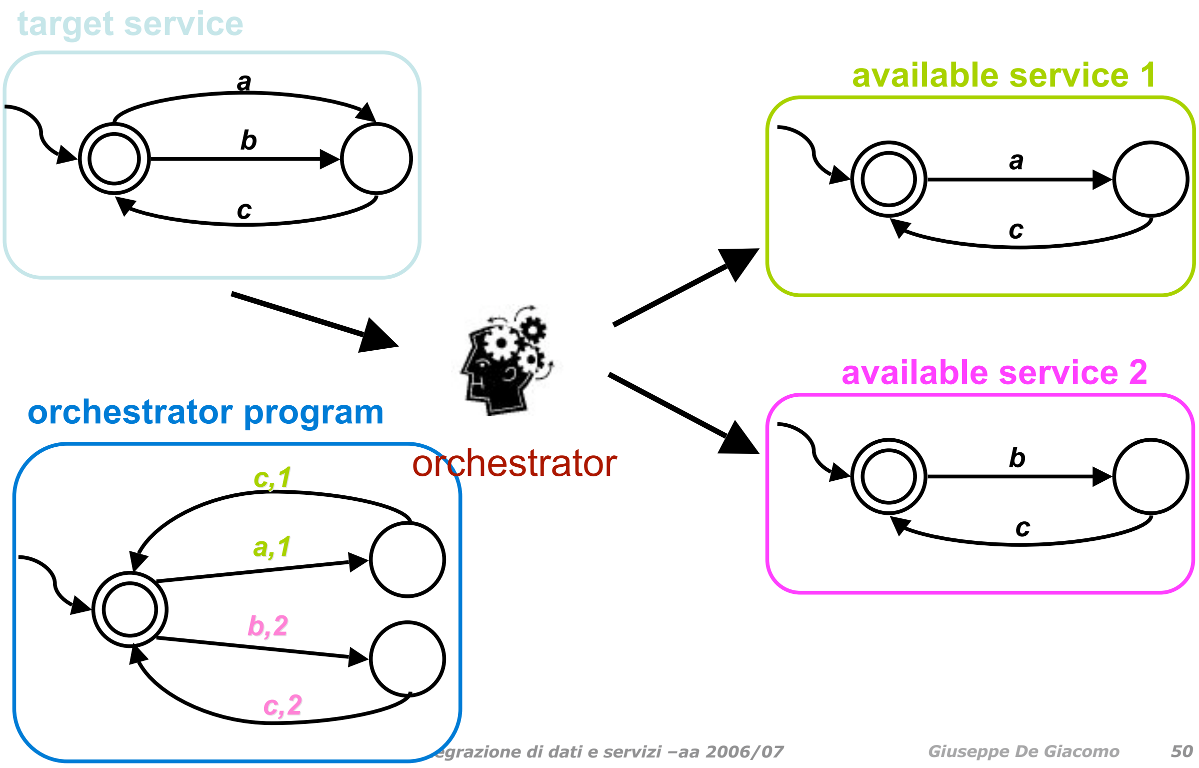

**A sample run**

| | | | | | |
|---|---|---|---|---|---|
| action request: | a | c | b | c | ... |
| orchestrator response: | a,1 | c,1 | b,2 | c,2 | |



# A orchestrator program realizing the target behavior

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma



# Orchestrator programs

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- **Orchestrator program** is *any function* $P(h,a) = i$ that takes a **history** *h* and an **action** *a* to execute and **delegates** *a* to one of the available services *i*
- A **history** is the sequence of actions done so far:

  $h = a_1 a_2 \dots a_k$

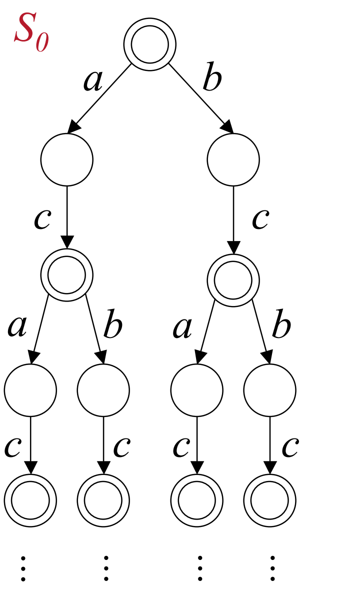
- Observe that to take a decision *P* has **full access to the past**, but no access to the future
  - *Note given an history* $h = a_1 a_2 \dots a_k$ *an the function* $P$ *we can reconstruct the state of the target service and of each available service*
    - $a_1 a_2 \dots a_k$ *determines the state of the target service*
    - $(a_1, P([\,], a_k))(a_2, P([a_1], a_2)) \dots (a_k, P([a_1 a_2 \dots a_{k-1}], a_k))$ *determines the state of of each available service*
- ***Problem: synthesize a orchestrator program P that realizes the target service making use of the available services***



# Service Execution Tree

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

By "unfolding" a (finite) TS one gets an (infinite) execution tree
*-- yet another (infinite) TS which bisimilar to the original one)*




- *Nodes: history i.e., sequence of actions executed so far*
- *Root: no action yet performed*
- *Successor node x·a of x: action a can be executed after the sequence of action x*
- *Final nodes: the service can terminate*

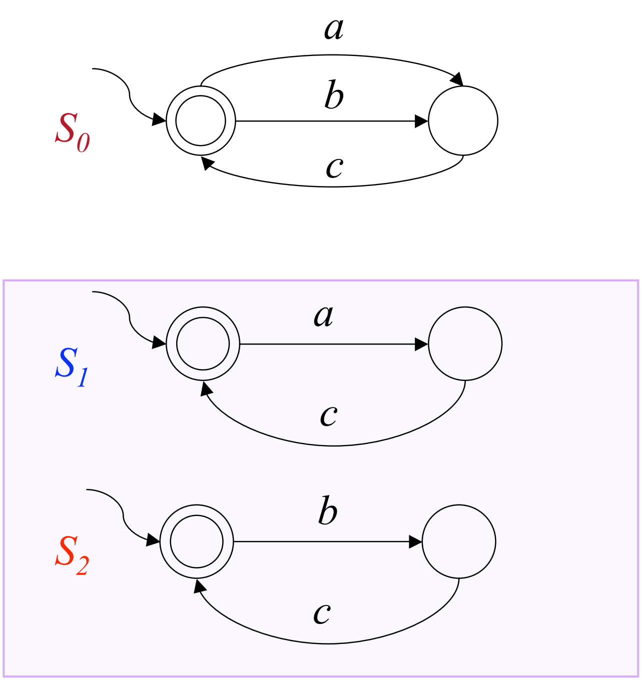
# Alternative (but Equivalent) Definition of Service Composition

Composition:
- coordinating program …
- … that realizes the target service …
- … by suitably coordinating available services

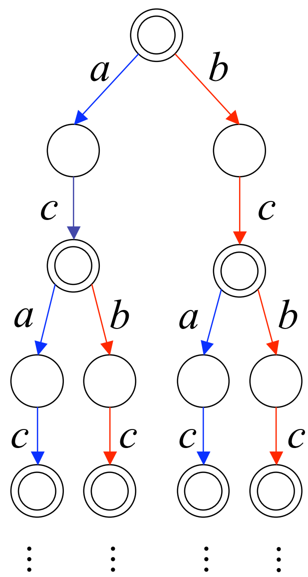
⇒ Composition can be seen as:
- a labeling of the execution tree of the **target service** such that …
- … each action in the execution tree is labeled by the available service that executes it …
- … and each possible sequence of actions on the target service execution tree corresponds to possible sequences of actions on the available service execution trees, suitably interleaved



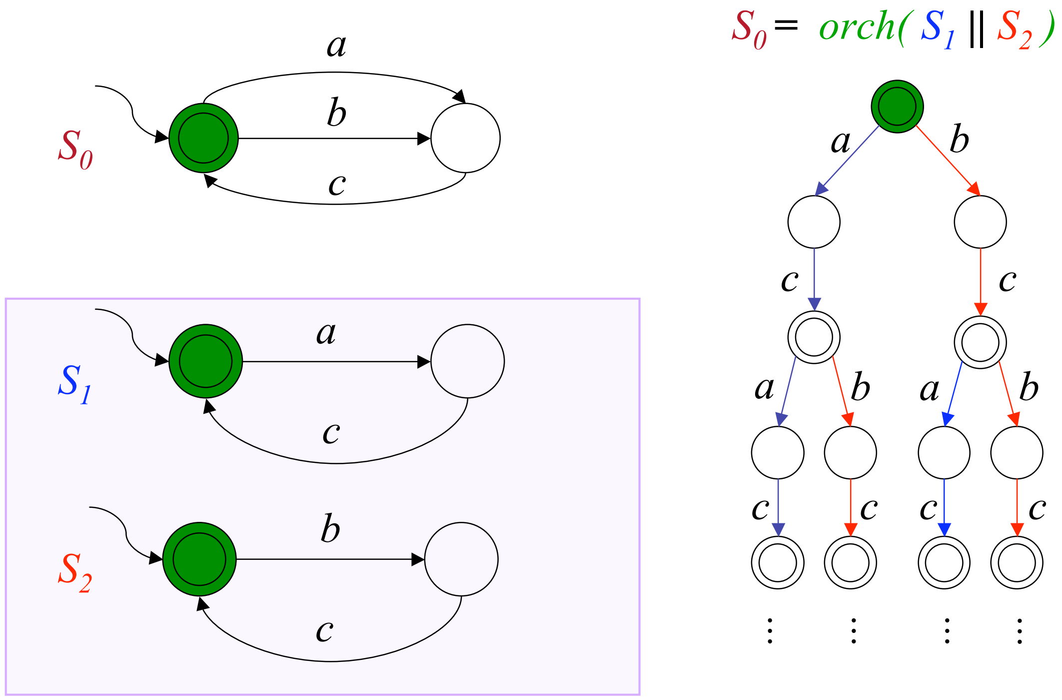
# Example of Composition




$S_0 = orch( S_1 \| S_2 )$



# Example of Composition

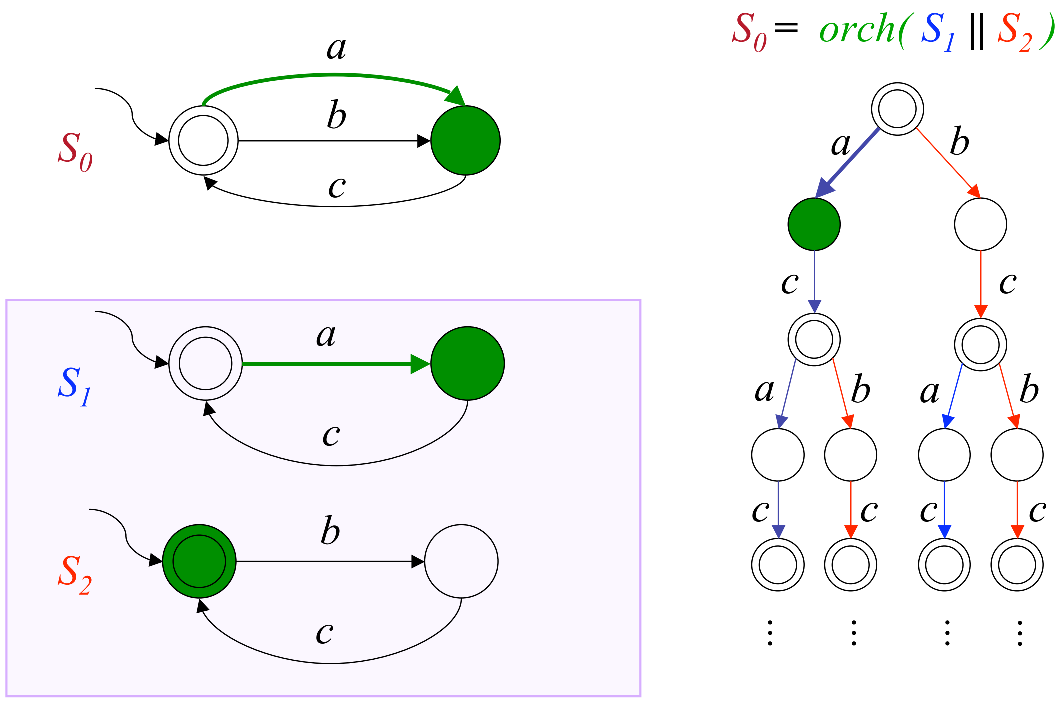



*All services start from their starting state*



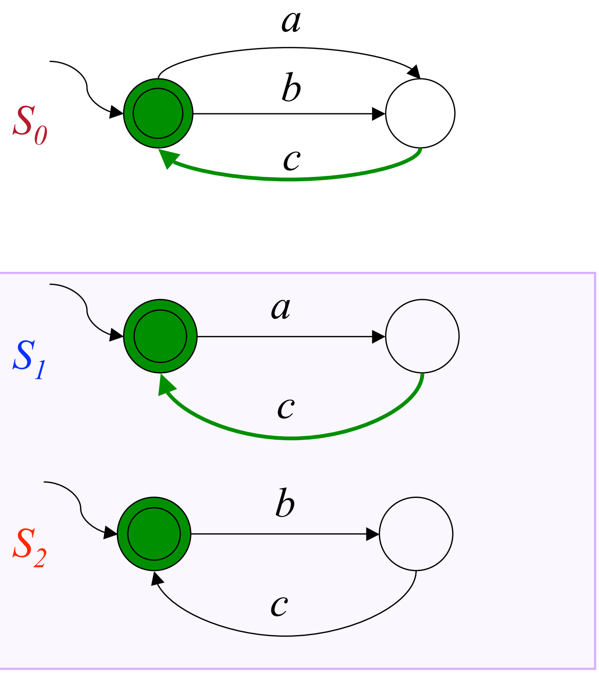
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

# Example of Composition (5)

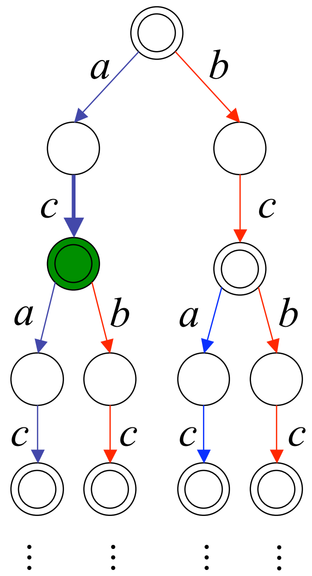



*Each action of the target service is executed by at least one of the component services*

# Example of composition (6)




$S_0 = orch( S_1 \parallel S_2 )$

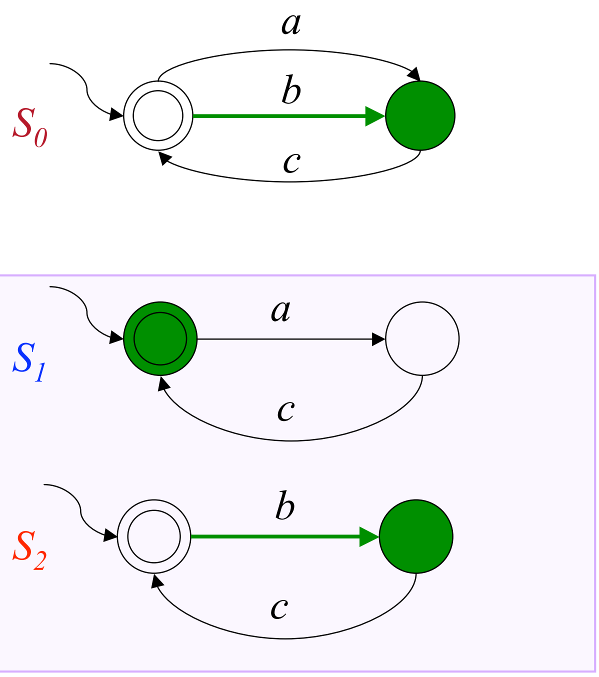



When the target service can be left, then all component services must be in a final state

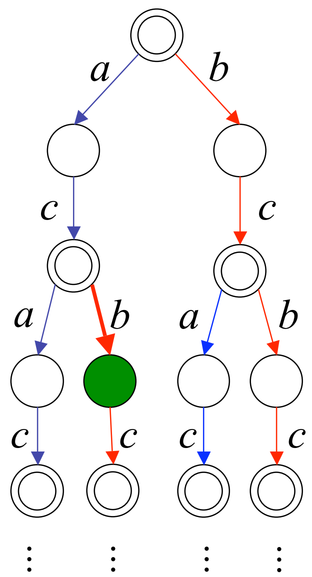


Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

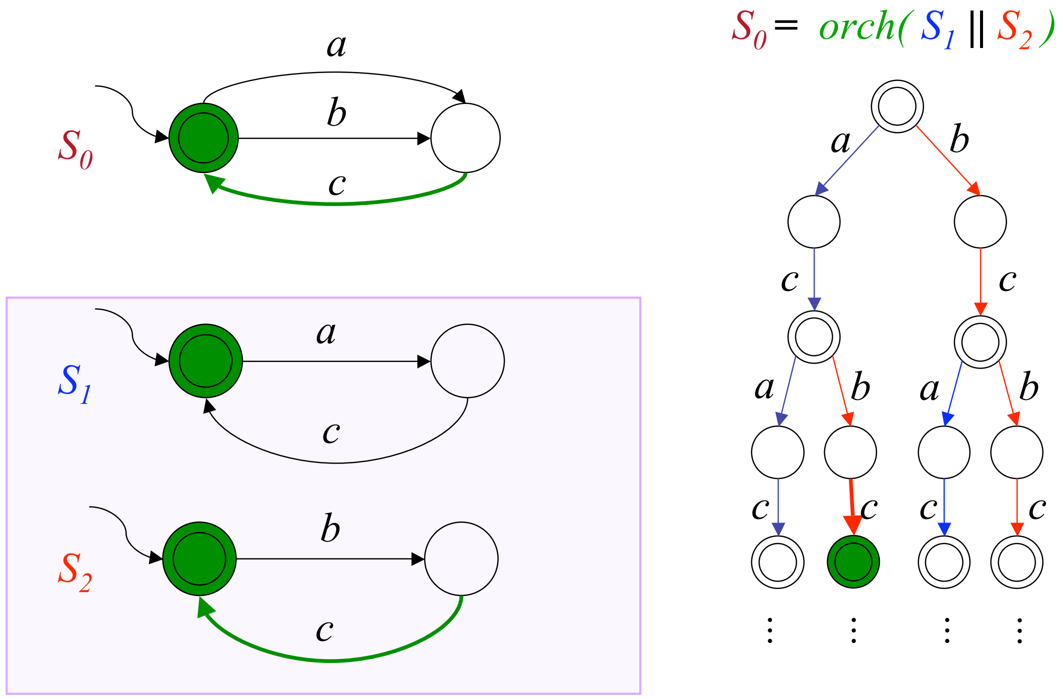
# Example of composition (7)




$S_0 = orch( S_1 \parallel S_2 )$



# Example of composition (8)

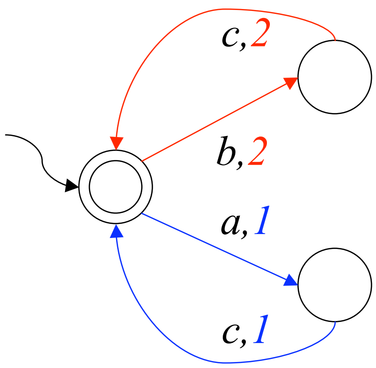







# Observation

- This labeled execution tree has a finite representation as a finite TS …
- …with transitions labeled by an action and the service performing the action




*Is this always the case when we deal with services expressible as finite TS? See later…*

# Questions

Assume services of community and target service are finite TSs

- Can we always check composition existence?
- If a composition exists there exists one which is a finite TS?
- If yes, how can a finite TS composition by computed?

*To answer ICSOC'03 exploits PDL SAT*



# Answers

Reduce service composition synthesis to satisfability in (deterministic) PDL

- Can we always check composition existence?
  *Yes, SAT in PDL is decidable in EXPTIME*
- If a composition exists there exists one which is a finite TS?
  *Yes, by the small model property of PDL*
- How can a finite TS composition be computed?
  *From a (small) model of the corresponding PDL formula*

# Encoding in PDL

Basic idea:

- A orchestrator program *P* realizes the target service *T* iff at each point:
  - ∀ transition labeled *a* of the target service *T* …
  - … ∃ an available service $B_i$ (the one chosen by *P*) that can make an *a*-transition, realizing the *a*-transition of *T*
- Encoding in PDL:
  - ∀ transition labeled *a* …
    use **branching**
  - ∃ an available service $B_i$ that can make an *a*-transition …
    use underspecified predicates **assigned through SAT**



# Structure of the PDL Encoding

$$\Phi = \mathbf{Init} \wedge [u](\Phi_0 \wedge \bigwedge_{i=1,\ldots,n} \Phi_i \wedge \Phi_{aux})$$

Init: Initial states of all services

$\Phi_0$: PDL encoding of target service

$\Phi_i$: PDL encoding of *i*-th component service

$\Phi_{aux}$: PDL additional domain-independent conditions

*PDL encoding is polynomial in the size of the service TSs*

# PDL Encoding

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- Target service $S_0 = (\Sigma, S_0, s^0_0, \delta_0, F_0)$ in PDL we define $\Phi_0$ as the conjunction of:
  - $s \rightarrow \neg s'$ for all pairs of distinct states in $S_0$
    *service states are pair-wise disjoint*
  - $s \rightarrow \langle a \rangle T \wedge [a]s'$ for each $s'=\delta_0(s,a)$
    *target service can do an a-transition going to state s'*
  - $s \rightarrow [a] \perp$ for each $\delta_0(s,a)$ undef.
    *target service cannot do an a-transition*
  - $F_0 \equiv \vee_{s \in F0}\, s$
    *denotes target service final states*
- …



# PDL Encoding (cont.d)

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- available services $S_i = (\Sigma, S_i, s^0_i, \delta_i, F_i)$ in PDL we define $\Phi_i$ as the conjunction of:
  - $s \rightarrow \neg s'$ for all pairs of distinct states in $S_i$
    *Service states are pair-wise disjoint*
  - $s \rightarrow [a](moved_i \wedge s' \vee \neg\, moved_i \wedge s)$ for each $s'=\delta_i(s,a)$
    *if service moved then new state, otherwise old state*
  - $s \rightarrow [a](\neg\, moved_i \wedge s)$ for each $\delta_i(s,a)$ undef.
    *if service cannot do a, and a is performed then it did not move*
  - $F_i \equiv \vee_{s \in Fi}\, s$
    *denotes available service final states*
- …

## PDL Encoding (cont.d)

- Additional assertions $\Phi_{aux}$
  - $\langle a \rangle T \rightarrow [a] \vee_{i=1,\ldots,n} moved_i$ for each action a

    *at least one of the available services must move at each step*
  - $F_0 \rightarrow \wedge_{i=1,\ldots,n} F_i$

    *when target service is final all comm. services are final*
  - $Init \equiv s^0_0 \wedge_{i=1,\ldots,n} s^0_i$

    *Initially all services are in their initial state*

**PDL encoding:** $\Phi = Init \wedge [u](\Phi_0 \wedge_{i=1,\ldots,n} \Phi_i \wedge \Phi_{aux})$



## Results

**Thm[ICSOC'03,IJCIS'05]:**
Composition exists iff PDL formula $\Phi$ SAT

*From composition labeling of the target service one can build a tree model of the PDL formula and viceversa*

*Information on the labeling is encoded in predicates $moved_i$*

**Corollary [ICSOC'03,IJCIS'05]:**
Checking composition existence is decidable in **EXPTIME**

**Thm[Muscholl&Walukiewicz FoSSaCS'07]:**
Checking composition existence is **EXPTIME-hard**

# Results on TS Composition

**Thm[ICSOC'03,IJCIS'05]**:

If composition exists then finite TS composition exists.

*From a small model of the PDL formula Φ, one can build a finite TS machine*

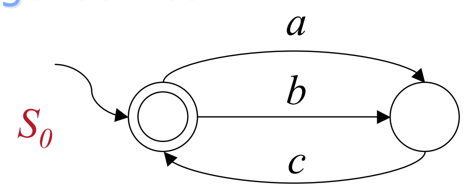
*Information on the output function of the machine is encoded in predicates $moved_i$*

⇒ finite TS composition existence of services expressible as finite TS is EXPTIME-complete

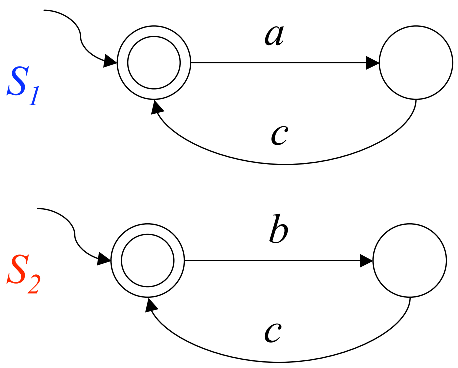


# Example (1)

Target service




Available services




PDL

...

...

...

$s_0^0 \wedge s_1^0 \wedge s_2^0$

$\langle a \rangle T \rightarrow [a] (moved_1 \vee moved_2)$

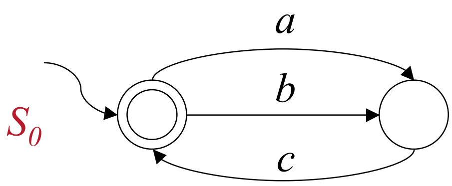
$\langle b \rangle T \rightarrow [b] (moved_1 \vee moved_2)$

$\langle c \rangle T \rightarrow [c] (moved_1 \vee moved_2)$

$F_0 \rightarrow F_1 \wedge F_2$

# Example (2)

## Target service




$s_0^0 \rightarrow \neg\, s_0^1$

$s_0^0 \rightarrow \langle a \rangle T \wedge [a]\, s_0^1$

$s_0^0 \rightarrow \langle b \rangle T \wedge [b]\, s_0^1$

$s_0^1 \rightarrow \langle c \rangle T \wedge [c]\, s_0^0$

$s_0^0 \rightarrow [c] \perp$

$s_0^1 \rightarrow [a] \perp$

$s_0^1 \rightarrow [b] \perp$

$F_0 \equiv s_0^0$

...

...

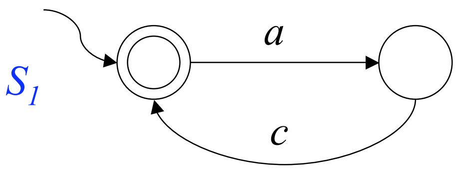
...

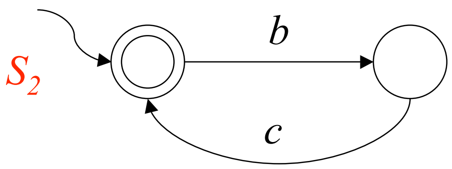


Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

# Example (3)

## Available services




...

$s_1^0 \rightarrow \neg\, s_1^1$

$s_1^0 \rightarrow [a]\, (moved_1 \wedge s_1^1 \vee \neg moved_1 \wedge s_1^0)$

$s_1^0 \rightarrow [c]\, \neg moved_1 \wedge s_1^0$

$s_1^0 \rightarrow [b]\, \neg moved_1 \wedge s_1^0$

$s_1^1 \rightarrow [a]\, \neg moved_1 \wedge s_1^1$

$s_1^1 \rightarrow [b]\, \neg moved_1 \wedge s_1^1$

$s_1^1 \rightarrow [c]\, (moved_1 \wedge s_1^0 \vee \neg moved_1 \wedge s_1^0)$

$F_1 \equiv s_1^0$

$s_2^0 \rightarrow \neg\, s_2^1$

$s_2^0 \rightarrow [b]\, (moved_2 \wedge s_2^1 \vee \neg moved_2 \wedge s_2^0)$

$s_2^0 \rightarrow [c]\, \neg moved_2 \wedge s_2^0$

$s_2^0 \rightarrow [a]\, \neg moved_2 \wedge s_2^0$

$s_2^1 \rightarrow [b]\, \neg moved_2 \wedge s_2^1$

$s_2^1 \rightarrow [a]\, \neg moved_2 \wedge s_2^1$

$s_2^1 \rightarrow [c]\, (moved_2 \wedge s_2^0 \vee \neg moved_2 \wedge s_2^0)$

$F_2 \equiv s_2^0$

...

# Example (4)

Check: run SAT on PDL formula $\Phi$

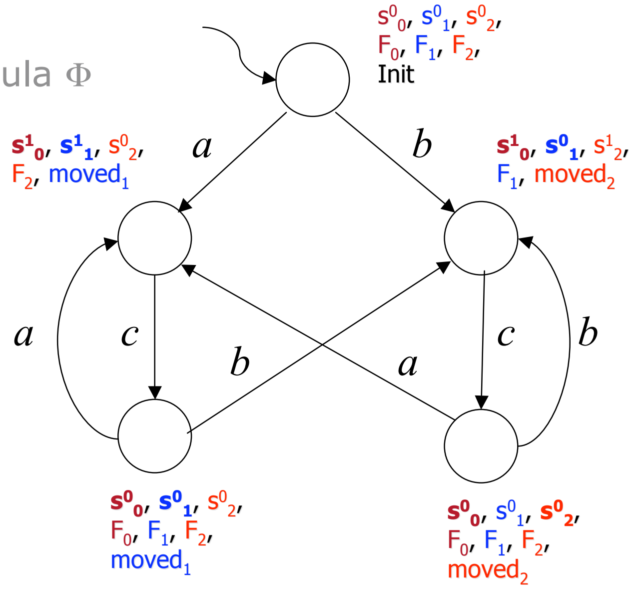




# Example

Check: run SAT on PDL formula $\Phi$

Yes $\Rightarrow$ (small) model

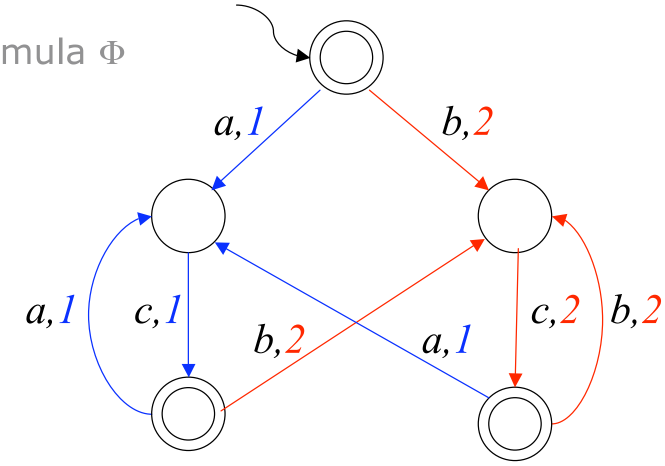


# Example

Check: run SAT on PDL formula $\Phi$

Yes $\Rightarrow$ (small) model

$\Rightarrow$ extract finite TS






# Example

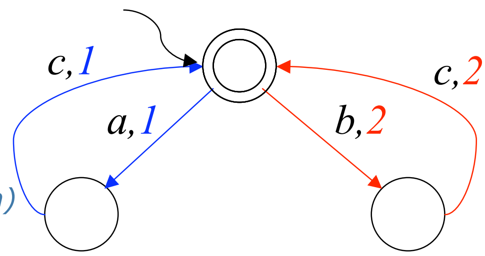
Check: run SAT on PDL formula $\Phi$

Yes $\Rightarrow$ (small) model

$\Rightarrow$ extract finite TS

$\Rightarrow$ minimize finite TS

*(similar to Mealy machine minimization)*



## Results on Synthesizing Composition

- Using PDL reasoning algorithms based on model construction (cf. tableaux), build a (small) model

  *Exponential in the size of the PDL encoding/services finite TS*

  *Note: SitCalc, etc. can compactly represent finite TS,*
  *PDL encoding can preserve compactness of representation*

- From this model extract a corresponding finite TS

  *Polynomial in the size of the model*

- Minimize such a finite TS using standard techniques (opt.)

  *Polynomial in the size of the TS*

  *Note: finite TS extracted from the model is not minimal*
  *because encodes output in properties of individuals/states*



## Tools for Synthesizing Composition

- In fact we use only a fragment of PDL in particular we use fixpoint (transitive closure) only to get the universal modality …
- … thanks to a tight correspondence between PDLs and Description Logics (DLs), we can use current highly optimized DL reasoning systems to do synthesis …
- … when the ability or returning models will be added …

  ***Pellet already has this ability, and we are exploring its use***

- … meanwhile we have developed a prototype tool on this idea (see last Massimo’s lecture)

# Composition via Simulation

## Bisimulation

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- A binary relation *R* is a **bisimulation** iff:

  (s,t) $\in$ *R* implies that
  - s is *final* iff t is *final*
  - for all actions a
    - if s $\rightarrow_a$ s' then $\exists$ t' . t $\rightarrow_a$ t' and (s',t')$\in$ *R*
    - if t $\rightarrow_a$ t' then $\exists$ s' . s $\rightarrow_a$ s' and (s',t')$\in$ *R*

- A state $s_0$ of transition system S is **bisimilar**, or simply **equivalent,** to a state $t_0$ of transition system T iff there **exists** a **bisimulation** between the initial states $s_0$ and $t_0$.

- Notably
  - **bisimilarity** is a bisimulation
  - **bisimilarity** is the largest bisimulation

*Note it is a co-inductive definition!*

# Computing Bisimilarity on Finite Transition Systems

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

**Algorithm** ComputingBisimulation
**Input:** transition system $TS_S$ = < A, S, $S^0$, $\delta_S$, $F_S$> and
transition system $TS_T$ = < A, T, $T^0$, $\delta_T$, $F_T$>
**Output:** the **bisimilarity** relation (the largest bisimulation)

**Body**

R = ∅
R' = S × T - {(s,t) | ¬(s ∈ $F_S$ ≡ t ∈ $F_T$)}
while (R ≠ R') {
R := R'
R' := R' - ({(s,t) | ∃ s',a. s $\rightarrow_a$ s' ∧ ¬∃ t' . t $\rightarrow_a$ t' ∧ (s',t') ∈ R' }
{(s,t) | ∃ t',a. t $\rightarrow_a$ t' ∧ ¬∃ s' . s $\rightarrow_a$ s' ∧ (s',t') ∈ R' })
}
return R'

**Ydob**



# Simulation

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- A binary relation *R* is a **simulation** iff:

  (s,t) ∈ *R* implies that
  - s is *final* implies that t is *final*
  - for all actions a
    - if s $\rightarrow_a$ s' then ∃ t' . t $\rightarrow_a$ t' and (s',t')∈ *R*

- A state $s_0$ of transition system S is **simulated by** a state $t_0$ of transition system T iff there **exists** a **simulation** between the initial states $s_0$ and $t_0$.

- Notably
  - **simulated-by** is a simulation
  - **simulated-by** is the largest simulation

*Note it is a co-inductive definition!*

- NB: A simulation is just one of the two directions of a bisimulation

## Computing Simulation on Finite Transition Systems

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

**Algorithm** ComputingSimulation
**Input:** transition system $TS_S = < A, S, S^0, \delta_S, F_S>$ and
transition system $TS_T = < A, T, T^0, \delta_T, F_T>$
**Output:** the **simulated-by** relation (the largest simulation)

**Body**

```
R = ∅
R' = S × T - {(s,t) | s ∈ F_S ∧ ¬(t ∈ F_T)}
while (R ≠ R') {
        R := R'
        R' := R' - {(s,t) | ∃ s',a. s →_a s' ∧ ¬∃ t' . t →_a t' ∧ (s',t') ∈ R' }
}
return R'
```

**Ydob**



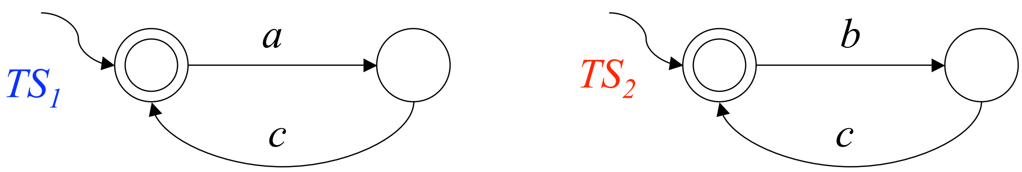
## Potential Behavior of the Whole Community

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

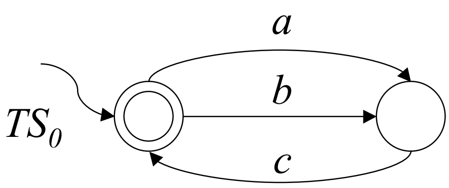
- Let $TS_1, \cdots ,TS_n$ be the TSs of the component services.
- The **Community TS** is defined as the **asynchronous product** of $TS_1, \cdots ,TS_n$, namely:
  $TS_c = < A, S_c, S_c^0, \delta_c, F_c>$ where:
  - A is the set of actions
  - $S_c = S_1 \times \cdots \times S_n$
  - $S_c^0 = \{(s^0_1, \cdots, s^0_m)\}$
  - $F \subseteq F_1 \times \cdots \times F_n$
  - $\delta_c \subseteq S_c \times A \times S_c$ is defined as follows:
    $(s_1 \times \cdots \times s_n) \rightarrow_a (s'_1 \times \cdots \times s'_n)$ iff
    1. $\exists$ i. $s_i \rightarrow_a s'_i \in \delta_i$
    2. $\forall$ $j \neq i$. $s'_j = s_j$

# Example of Composition

- Available Services



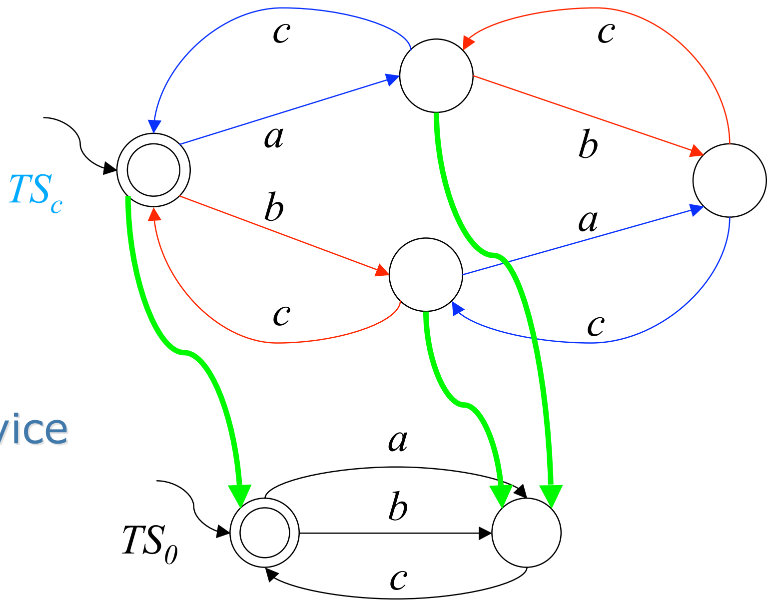

- Target Service






# Example of Composition

Community TS




Target Service

**Composition exists!**

# Composition via Simulation

- **Thm[Subm07]**
  A composition realizing a target service TS $TS_t$ exists if there **exists** a simulation relation between the initial state $s_t^0$ of $TS_t$ and the initial state $(s_1^0, .., s_n^0)$ of the community TS $TS_c$.

- Notice if we take the union of all simulation relations then we get the largest simulation relation ***S***, still satisfying the above condition.

- **Corollary[Subm07]**
  A composition realizing a target service TS $TS_t$ exists iff $(s_t^0 , (s_1^0, .., s_n^0)) \in$ ***S***.

- **Thm[Subm07]**
  Computing the largest simulation ***S*** is polynomial in the size of the target service TS and the size of the community TS…
- ... hence it is EXPTIME in the size of the available services.



# Composition via Simulation

- Given the largest simulation ***S*** form $TS_t$ to $TS_c$(which include the initial states), we can build the ***orchestrator generator***.

- This is an orchestrator program that can change its behavior reacting to the information acquired at run-time.

- Def: OG = $< A, [1,…,n], S_r, s_r^0, \omega_r, \delta_r, F_r>$ with
  - A : the **actions** shared by the community
  - [1,…,n]: the **identifiers** of the available services in the community
  - $S_r = S_t \times S_1 \times \cdots \times S_n$ : the **states** of the orchestrator program
  - $s_r^0 = (s^0_t, s^0_1, …, s^0_m)$ : the **initial state** of the orchestrator program
  - $F_r \subseteq \{ (s_t , s_1 , …, s_n) \mid s_t \in F_t$ : the **final states** of the orchestrator program
  - $\omega_r : S_r \times A_r \to [1,…,n]$ : the **service selection function**, defined as follows:
    - If $s_t \to_a s'_t$ then
      *chose* k s.t. $\exists s_k'. s_k \to_a s_k' \wedge (s_t', (s_1 , …, s'_k , …, s_n) )\in$ ***S***
  - $\delta_r \subseteq S_r \times A_r \times [1,…,n] \to S_r$ : the **state transition function**, defined as follows:
    - Let $\omega_r(s_t, s_1 , …, s_k , …, s_n, a) = k$ then
      $(s_t, s_1 , …, s_k , …, s_n) \to_{a,k} (s_t', s_1 , …, s'_k , …, s_n)$ where $s_k \to_a s'_k$

## *Composition via Simulation*

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- For generating OG we need only to compute ***S*** and then apply the template above
- For running an orchestrator from the OG we need to store and access ***S*** *(polynomial time, exponential space)* …
- … and compute $\omega_r$ and $\delta_r$ at each step *(polynomial time and space)*



Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

## *Extension to the Roman Model*

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

# Extensions

- **Nondeterministic** (angelic) **target** specification
  - Loose specification in client request
  - **Angelic (don't care)** vs devilish (don't know) nondeterminism
  - See [ICSOC'04]
- **Nondeterministic** (devilish) available **services**
  - Incomplete specification in available services
  - **Devilish (don't know)** vs angelic (don't care) nondeterminism
  - See below & [IJCAI'07]

***See later***

- **Distributing** the orchestration
  - Often a centralized orchestration is unrealistic: eg. services deployed on mobile devices
    - too tight coordination
    - too much communication
    - orchestrator cannot be embodied anywhere
  - Drop centralized orchestrator in favor of **independent controllers** on single available services (exchanging messages)
  - Under suitable conditions: a distributed orchestrator exists iff a centralized one does
  - Still decidable (EXPTIME-complete)
  - See [AAAI'07]
- Dealing with **data**
  - This is the single most difficult issue to tackle
    - First results: actions as DB updates, see [VLDB'05]
    - Literature on Abstraction in Verification
  - From finite to **infinite transition** systems!
- **Security and trust** aware composition [SWS'06]
- Automatic **Workflows** Composition of Mobile Services [ICWS'07]



Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

# *Nondeterministic Available Services*

# Nondeterminism in Available Services

*Devilish (don't know)!*

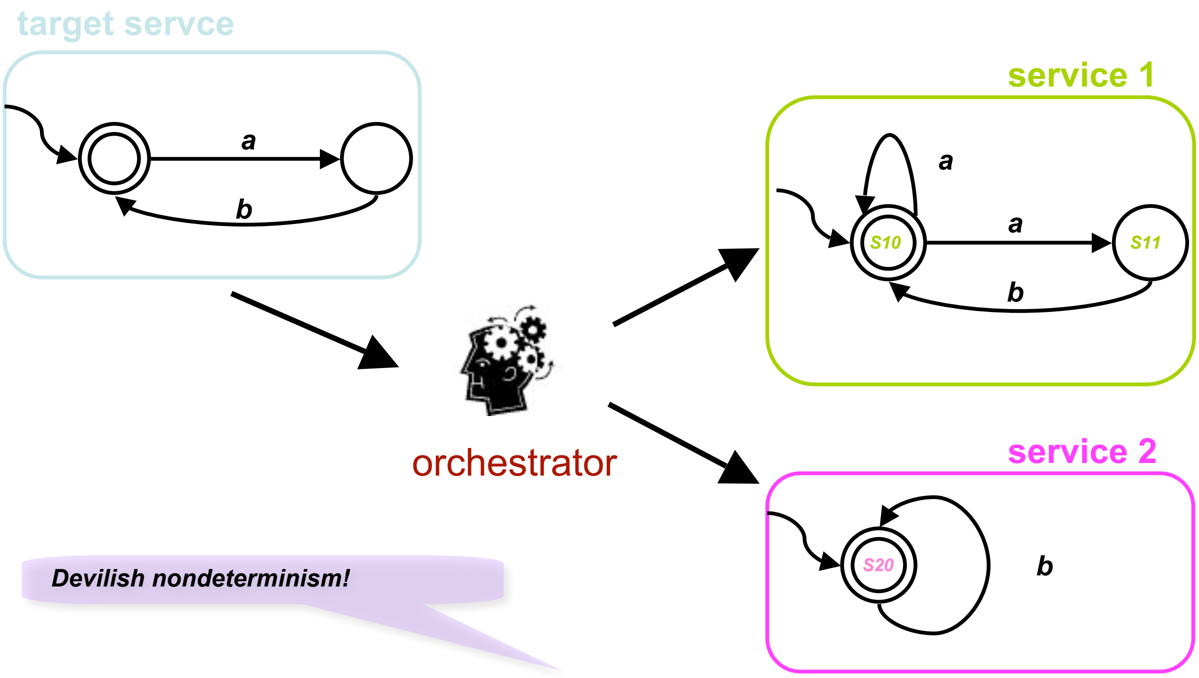
- Nondeterministic available services
  - **Incomplete information** on the actual **behavior**
  - **Mismatch between behavior description** (which is in terms of the environment actions) and **actual behavior** of the agents/devices

- Deterministic target service
  - it's a spec of a desired service: (devilish) nondeterminism is banned

*In general, devilish nondeterminism difficult to cope with*
*eg. nondeterminism moves AI Planning from PSPACE (classical planning) to EXPTIME (contingent planning with full observability [Rintanen04])*



# Example: Nondeterministic Available Services

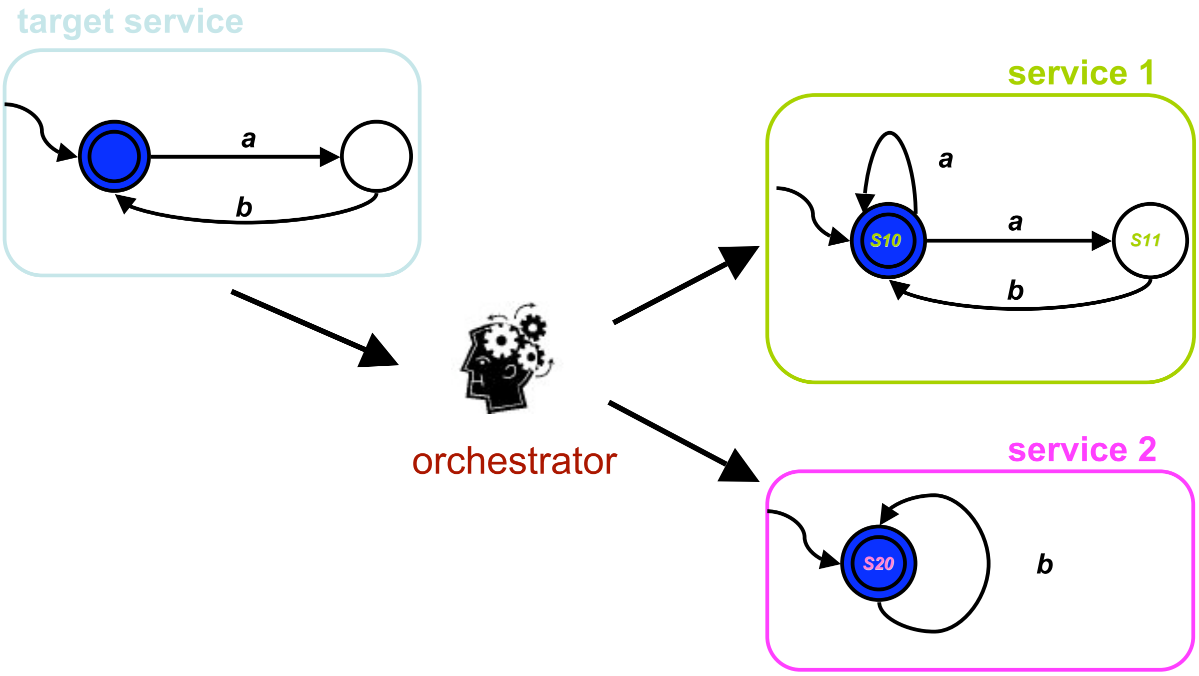



*Devilish nondeterminism!*

*Available services represented as nondeterministic transition systems*

# Example: Nondeterministic Available Services

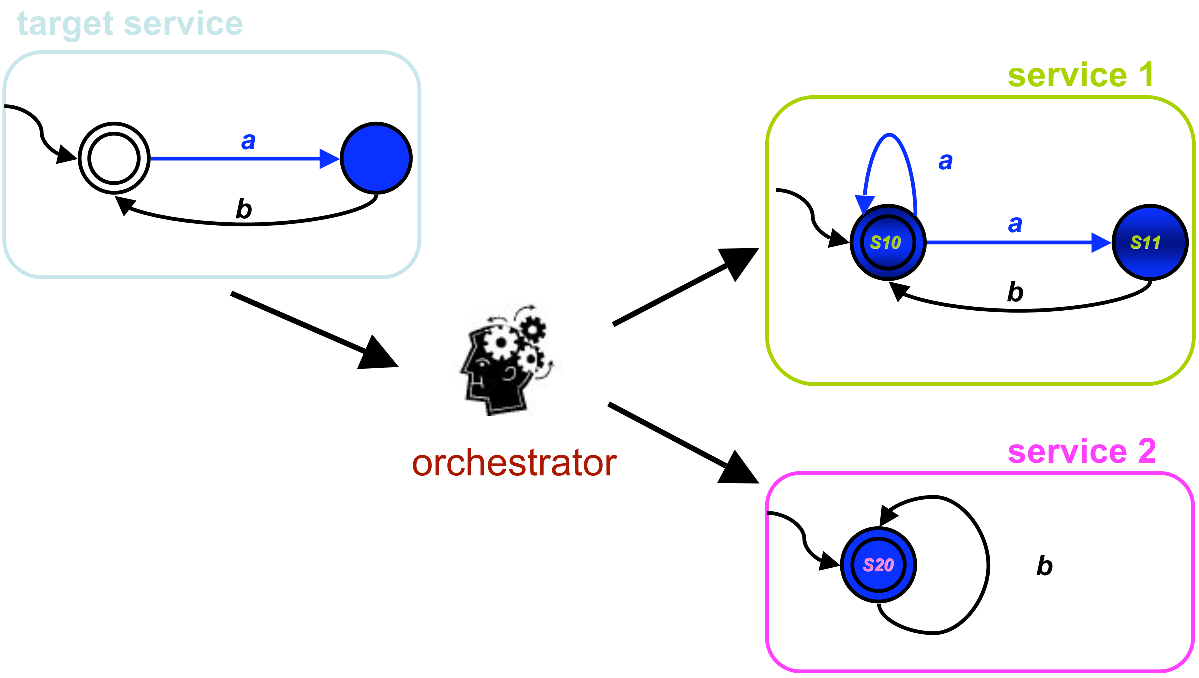
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma





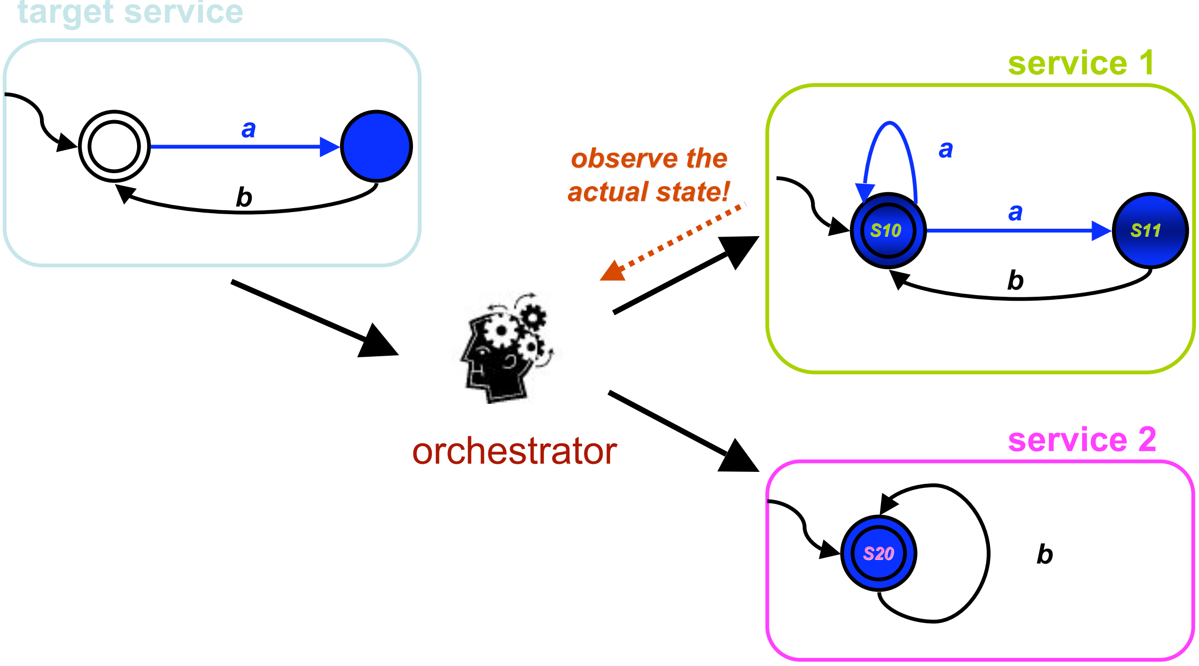

# Example: Nondeterministic Available Services

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma



# Example: Nondeterministic Available Services

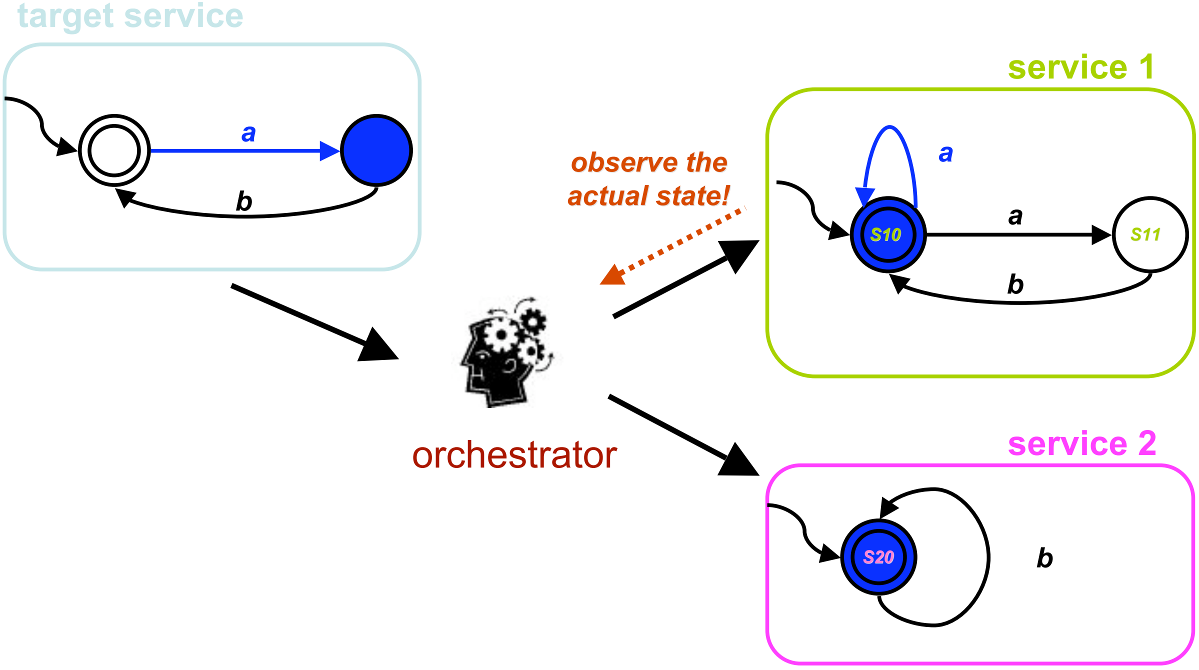




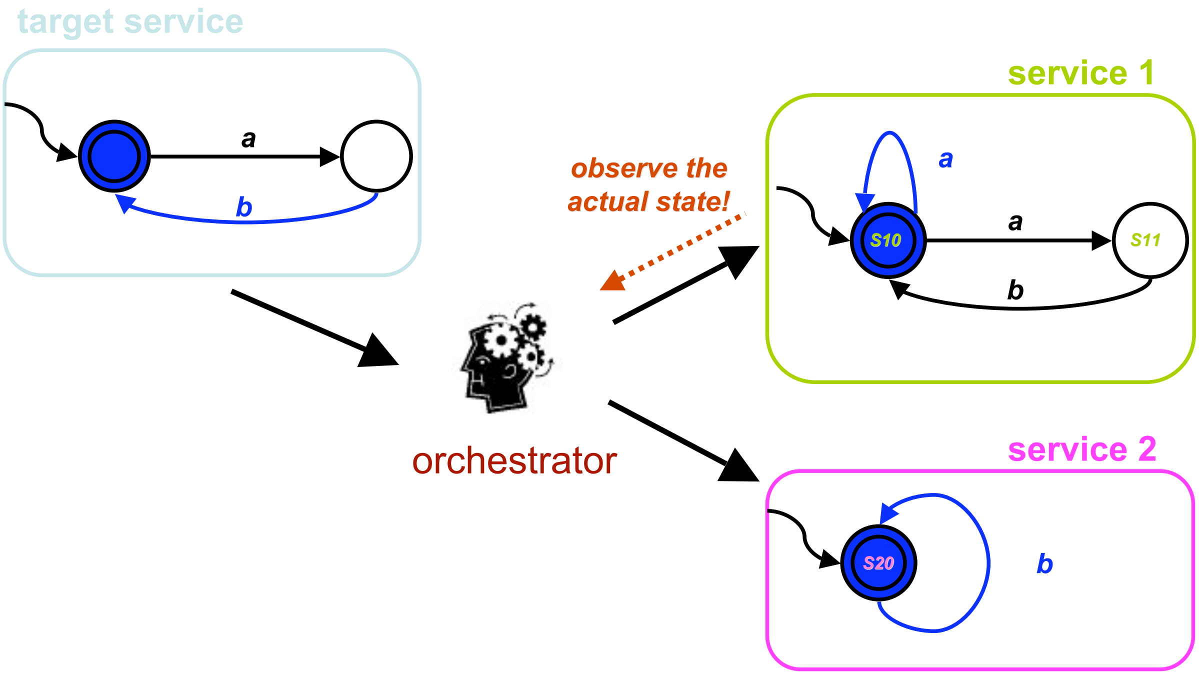



# Example: Nondeterministic Available Services



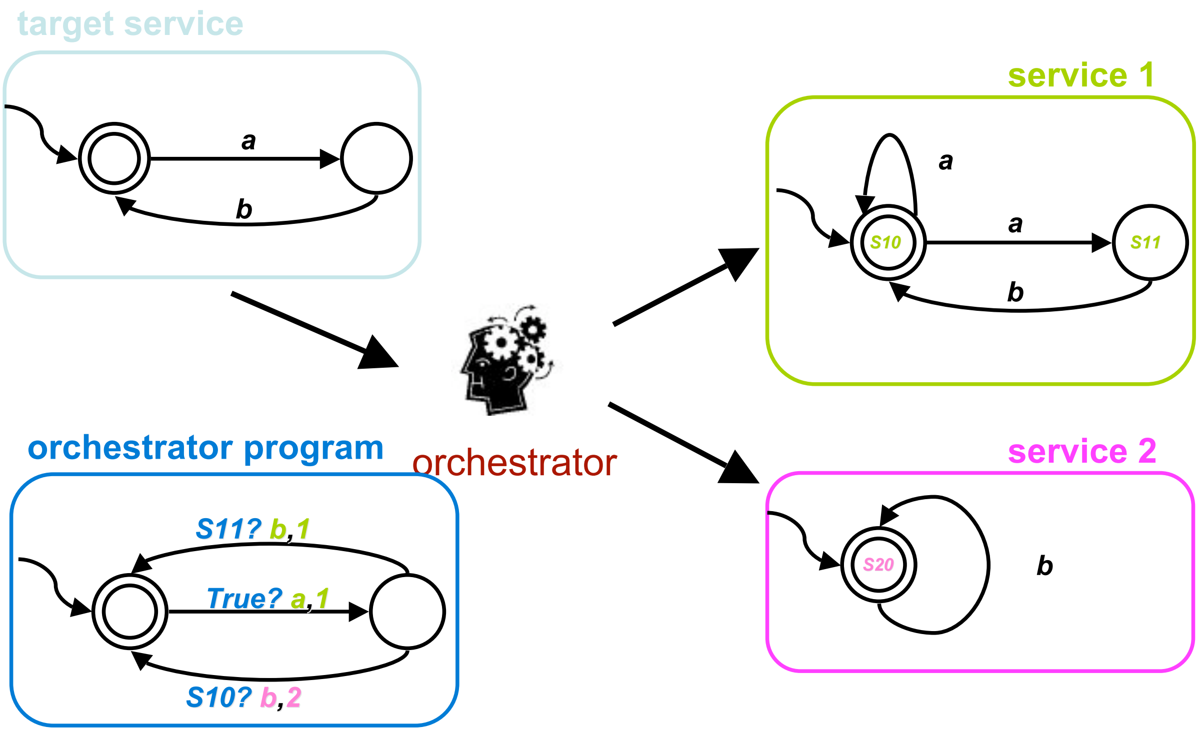


# Example: Nondeterministic Available Services






# An Orchestrator Program Realizing the Target Service



# Orchestrator Programs

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

*contains all the observable information up the current situation*

- **Orchestrator program** is *any function P(h,a) = i* that takes a **history** *h* and an **action** *a* to execute and **delegates** *a* to one of the available services *i*

- A **history** is a sequence of the form:

  $(s_1^0,s_2^0,\ldots,s_n^0,e^0)\ a_1\ (s_1^1,s_2^1,\ldots,s_n^1,e^1)\ \ldots\ a_k\ (s_k^1,s_2^k,\ldots,s_n^k,e^k)$

- Observe that to take a decision *P* has **full access to the past**, but no access to the future

- *Problem: synthesize a orchestrator program P that realizes the target service making use of the available services*



# Technique: Reduction to PDL

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

Basic idea:

- A orchestrator program *P* realizes the target service *T* iff at each point:
  - ∀ transition labeled *a* of the target service *T* …
  - … ∃ an available service $B_i$ (the one chosen by *P*) which can make an *a*-transition …
  - … and ∀ *a*-transition of $B_i$ realize the *a*-transition of *T*

- Encoding in PDL:
  - ∀ transition labeled *a* …
    use **branching**
  - ∃ an available service $B_i$ …
    use underspecified predicates **assigned through SAT**
  - ∀ *a*-transition of $B_i$ … :
    use **branching** again

# Technical Results: Theoretical

**Thm[IJCAI'07]** Checking the existence of orchestrator program realizing the target service is **EXPTIME-complete**.

***EXPTIME-hardness due to Muscholl&Walukiewicz07 for deterministic services***

**Thm [IJCAI'07]** If a **orchestrator program exists** there exists one that is **finite state**.

***Exploits the finite model property of PDL***

***Note: same results as for deterministic services!***



# Technical Results: Practical

*Reduction to PDL provides also a practical sound and complete technique to compute the orchestrator program also in this case*

***eg, PELLET @ Univ. Maryland***

- Use state-of-the-art tableaux systems for OWL-DL for checking SAT of PDL formula $\Phi$ coding the composition existence
- *If SAT, the tableau returns a finite model of* $\Phi$

***exponential in the size of the behaviors***

- Project away irrelevant predicates from such model, and possibly minimize
- *The resulting structure is a finite orchestrator program that realizes the target behavior*

***polynomial in the size of the model***

# Nondeterministic Available Services: Composition à la Simulation

# Composition à la Simulation

- We consider binary relations *R* satisfying the following co-inductive condition:

  $(s,(q_1, .., q_n)) \in R$ implies that
  - if s is *final* then $q_i$, with i=1, .., n, is *final*
  - for **all** actions a
    - if $s \rightarrow_a s'$ then $\exists\, k \in 1..n$.
      - $\exists\, q_k' .\ q_k \rightarrow_a q_k'$
      - $\forall\, q_k'.\ q_k \rightarrow_a q_k' \supset (s',(q_1,..,q_k', .., q_n)) \in R$

  *Note similar in the spirit to simulation relation!*
  *But more involved, since it deals with*

  - *the existential choice (as the simulation) of the service, and*
  - *the universal condition on the nondeterministic branches!*

- A composition realizing a target service TS $TS_t$ exists if there **exists** a relation *R* satisfying the above condition between the initial state $s_t^0$ of $TS_t$ and the initial state $(s_1^0, .., s_n^0)$ of the community big TS $TS_c$.

- Notice if we take the union of all such relation *R* then we get the largest relation ***RR*** satisfying the above condition.

- A composition realizing a target service TS T exists iff $(s_t^0 , (s_1^0, .., s_n^0)) \in$ ***RR***.

# Composition à la Simulation

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

- Given ***RR*** form $TS_t$ to $TS_c$(which include the initial states), we can build the ***orchestrator generator***.
- This is an orchestrator program that can change its behavior reacting to the information acquired at run-time.
- Def: OG = < A, [1,…,n], $S_r$, $s_r^0$, $\omega_r$, $\delta_r$, $F_r$> with
  - A : the **actions** shared by the community
  - [1,…,n]: the **identifiers** of the available services in the community
  - $S_r = S_t \times S_1 \times \cdots \times S_n$ : the **states** of the orchestrator program
  - $s_r^0 = (s^0_t, s^0_1, \ldots, s^0_m)$ : the **initial state** of the orchestrator program
  - $F_r \subseteq \{ (s_t , s_1 , \ldots, s_n) \mid s_t \in F_t$ : the **final states** of the orchestrator program
  - $\omega_r : S_r \times A_r \to [1,\ldots,n]$ : the **service selection function**, defined as follows:
    - If $s_t \to_a s'_t$ then
      *chose* k s.t. $\exists\, s_k'.\ s_k \to_a s_k' \wedge \forall\, s_k'.\ s_k \to_a s_k' \supset (s_t', (s_1 , \ldots, s'_k , \ldots, s_n) ) \in$ ***RR***
  - $\delta_r \subseteq S_r \times A_r \times [1,\ldots,n] \times S_r$ : the **state transition relation**, defined as follows:
    - Let $\omega_r(s_t, s_1 , \ldots, s_k , \ldots, s_n, a) = k$ then
      $(s_t, s_1 , \ldots, s_k , \ldots, s_n) \to_{a,k} (s_t', s_1 , \ldots, s'_k , \ldots, s_n)$ for each $s_k \to_a s'_k$



# Composition à la Simulation

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

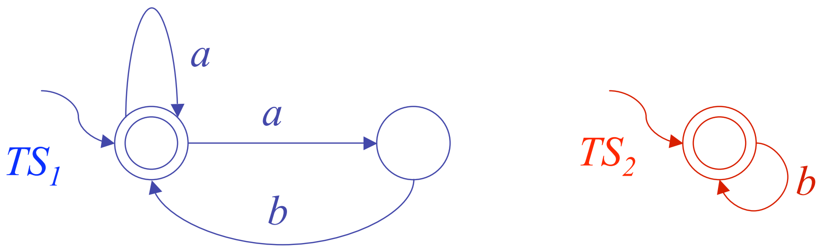
- Computing ***RR*** is polynomial in the size of the target service TS and the size of the community TS…
- … composition can be done in EXPTIME in the size of the available services

- For generating OG we need only to compute ***RR*** and then apply the template above

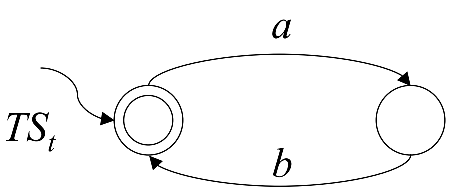
- For running the OG we need to store and access ***RR*** *(polynomial time, exponential space)* …
- … and compute $\omega_r$ and $\delta_r$ at each step *(polynomial time and space)*

# Example of Composition

## Available Services




## Target Service



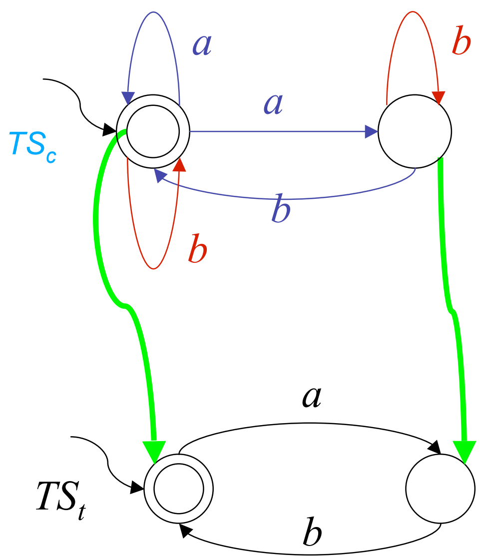



Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

# Example of Composition

## Community TS

## Target Service




**Composition exists!**

# References

[ICSOC'03] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: Automatic Composition of E-services That Export Their Behavior. ICSOC 2003: 43-58

[WES'03] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: A Foundational Vision of e-Services. WES 2003: 28-40

[TES'04] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: : A Tool for Automatic Composition ofServices Based on Logics of Programs. TES 2004: 80-94

[ICSOC'04] Daniela Berardi, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella, Diego Calvanese: Synthesis of underspecified composite e-services based on automated reasoning. ICSOC 2004: 105-114

[IJCIS'05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: Automatic Service Composition Based on Behavioral Descriptions. Int. J. Cooperative Inf. Syst. 14(4): 333-376 (2005)

[VLDB'05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, Massimo Mecella: Automatic Composition of Transition-based Semantic Web Services with Messaging. VLDB 2005: 613-624

[ICSOC'05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Massimo Mecella: Composition of Services with Nondeterministic Observable Behavior. ICSOC 2005: 520-526

[SWS'06] Fahima Cheikh, Giuseppe De Giacomo, Massimo Mecella: Automatic web services composition in trustaware communities. Proceedings of the 3rd ACM workshop on Secure web services 2006. Pages: 43 - 52.

[AISC'06] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Massimo Mecella. Automatic Web Service Composition: Service-tailored vs. Client-tailored Approaches. In Proc. AISC 2006, International Workshop jointly with ECAI 2006.

[FOSSACS'07] Anca Muscholl, Igor Walukiewicz: A lower bound on web services composition. Proceedings FOSSACS, LNCS, Springer, Volume 4423, page 274--287 - 2007.

[IJCAI'07] Giuseppe De Giacomo, Sebastian Sardiña: Automatic Synthesis of New Behaviors from a Library of Available Behaviors. IJCAI 2007: 1866-1871

[AAAI'07] Sebastian Sardiña, Fabio Patrizi, Giuseppe De Giacomo: Automatic synthesis of a global behavior from multiple distributed behaviors. In Proceedings of the National Conference on Artificial Intelligence (AAAI), Vancouver, Canada, July 2007.

[Subm07] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, Fabio Patrizi: Automatic Service Composition via Simulation. Submitted.