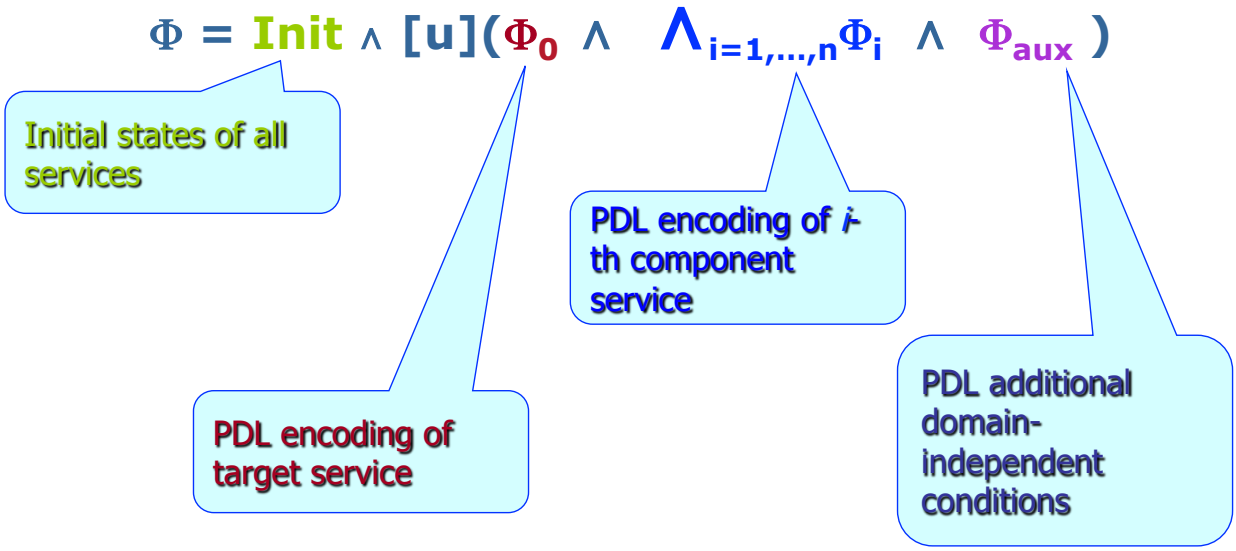
Composition: the "Roman" Approach Reduction to SAT in PDL

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



PDL encoding is polynomial in the size of the service TSs

PDL Encoding

- Target service $\mathcal{S}_0 = (\Sigma, S_0, s^0_0, \delta_0, F_0)$ in PDL we define Φ_0 as the conjunction of:
  - $s \rightarrow \neg s'$ for all pairs of distinct states in S_0
    service states are pair-wise disjoint
  - $s \rightarrow \langle a \rangle \top \wedge [a]s'$ for each $s'=\delta_0(s,a)$
    target service can do an a-transition going to state s'
  - $s \rightarrow [a] \bot$ 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)

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

PDL Encoding (cont.d)

- Additional assertions Φ_{aux}
  - $\langle a \rangle T \rightarrow [a] \vee_{i=1,\ldots,n} \text{moved}_i$ 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
  - $\text{Init} \equiv s^0_0 \wedge_{i=1\ldots n} s^0_i$
    Initially all services are in their initial state

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

Results

Thm[ICSOC'03,IJCIS'05]:
Composition exists iff PDL formula Φ 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

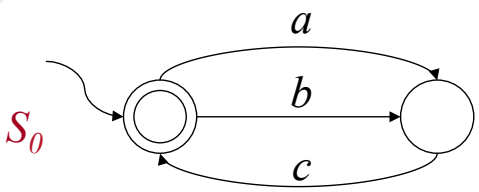
Information on the output function of the machine is encoded in predicates moved$_i$

$\Rightarrow$ finite TS composition existence of services expressible as finite TS is EXPTIME-complete

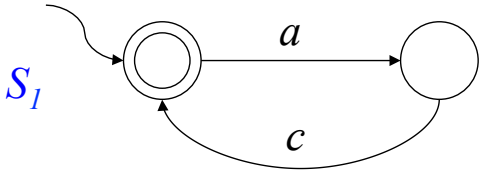
Example (1)

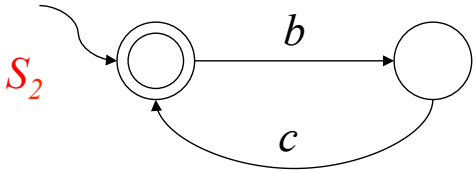
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA UNIVERSITÀ DI ROMA

Target service



Available services





PDL

...

...

...

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

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

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

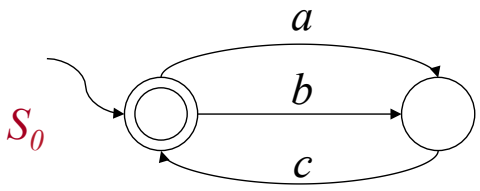
$\langle c \rangle T \rightarrow [c] (moved_1 \vee moved_2)$

$F_0 \rightarrow F_1 \wedge F_2$

Example (2)

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA UNIVERSITÀ DI ROMA

Target service



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

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

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

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

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

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

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

$F_0 \equiv s_0^0$

...

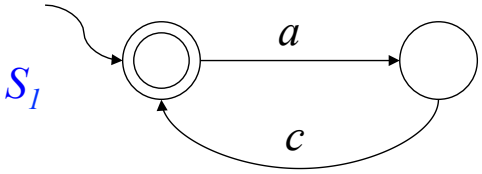
...

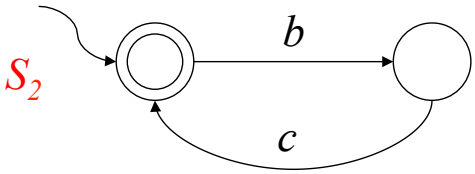
...

Example (3)

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA UNIVERSITÀ DI ROMA

Available services





...

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

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

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

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

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

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

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

$F_1 \equiv s_1^0$

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

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

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

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

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

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

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

$F_2 \equiv s_2^0$

...

Example (4)

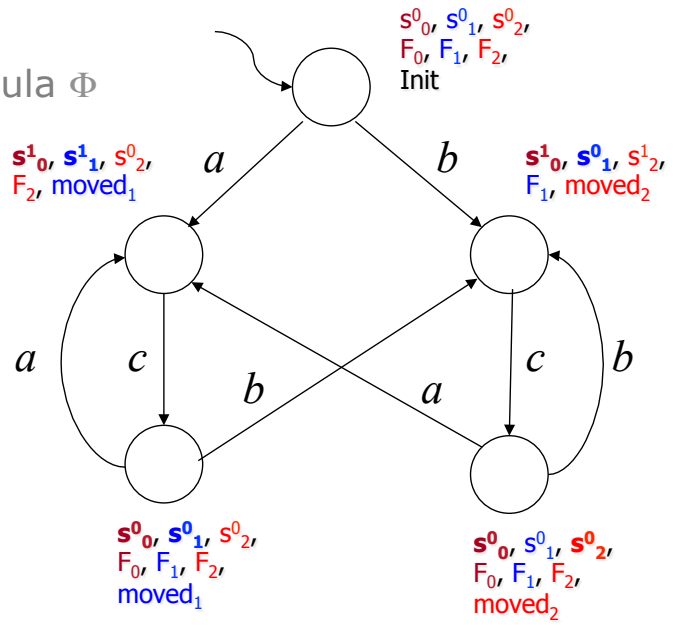
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA UNIVERSITÀ DI ROMA

Check: run SAT on PDL formula Φ

Example

Check: run SAT on PDL formula Φ

Yes $\Rightarrow$ (small) model

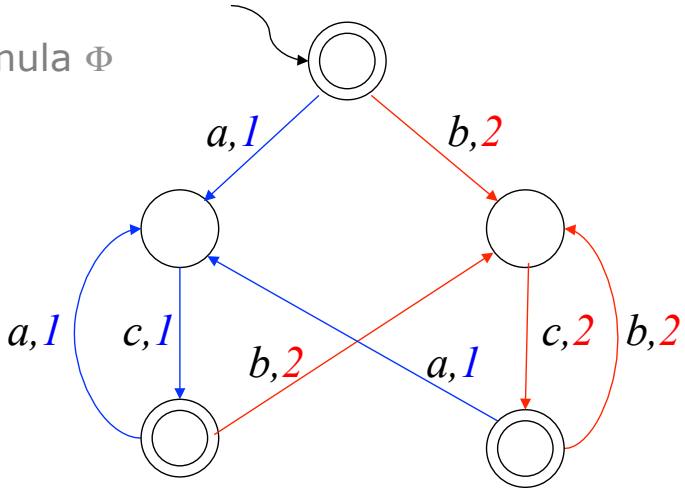


Example

Check: run SAT on PDL formula Φ

Yes $\Rightarrow$ (small) model

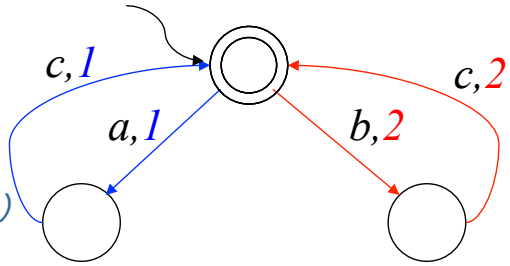
$\Rightarrow$ extract finite TS



Example

Check: run SAT on PDL formula Φ
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), lately highly optimized tableaux based reasoning systems are available to:
  - check for composition existence
  - do composition synthesis *(if the ability or returning models is present)*

- Among them we recall:
  - Racer (http://www.racer-systems.com/) based on DLs
  - Pellet (http://clarkparsia.com/pellet) based on DLs
  - Fact++ (http://owl.man.ac.uk/factplusplus/) based on DLs
  - PDL Tableaux (http://www.cs.manchester.ac.uk/~schmidt/pdl-tableau/) based on PDL
  - Tableaux Workbench (http://twb.rsise.anu.edu.au/) based on PDL
  - Lotrec (http://www.irit.fr/Lotrec/) based on PDL

Reduction to PDL SAT works also for nondeteministic available services

Technique1: Reduction to PDL

Basic idea:

- A orchestrator program P realizes the target service T iff at each point:
  - $\forall$ transition labeled a of the target service T …
  - … $\exists$ an available service B_i (the one chosen by P) which can make an a-transition …
  - … and $\forall$ a-transition of B_i realize the a-transition of T
- Encoding in PDL:
  - $\forall$ transition labeled a …
    use **branching**
  - $\exists$ an available service B_i …
    use underspecified predicates **assigned through SAT**
  - $\forall$ a-transition of B_i … :
    use **branching** again

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 Φ coding the composition existence
- *If SAT, the tableau returns a finite model of* Φ

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