

# Realizability of Concurrent Recursive Programs

Benedikt Bollig<sup>1</sup>

Manuela-Lidia Grindei<sup>1</sup>

Peter Habermehl<sup>2</sup>

<sup>1</sup>LSV, ENS Cachan, CNRS  
France

<sup>2</sup>LIAFA, CNRS, Paris 7  
France

IRISA, Rennes, France  
April 2009

# Concurrent recursive programs

## Verification

- amounts to analyzing multi-stack pushdown systems
- abstraction of unrestricted systems
  - ▶ overapproximation [BET'03]
  - ▶ underapproximation [QR'05]
- restricting the degree of synchronization and parallelism [SV'06]

## Synthesis

- language and automata theoretic framework needed
- generalization of asynchronous automata and Mazurkiewicz traces

---

[BET'03] Bouajjani & Esparza & Touili. [A generic approach to the static analysis of concurrent programs with procedures](#). 2003.

[QR'05] Qadeer & Rehof. [Context-bounded model checking of concurrent software](#). 2005.

[SV'06] Sen & Viswanathan. [Model checking multithreaded programs with asynchronous atomic methods](#). 2006.

# Concurrent recursive programs

## Verification

- amounts to analyzing multi-stack pushdown systems
- abstraction of unrestricted systems
  - ▶ overapproximation [BET'03]
  - ▶ underapproximation [QR'05]
- restricting the degree of synchronization and parallelism [SV'06]

## Synthesis

- language and automata theoretic framework needed
- generalization of asynchronous automata and Mazurkiewicz traces

---

[BET'03] Bouajjani & Esparza & Touili. [A generic approach to the static analysis of concurrent programs with procedures](#). 2003.

[QR'05] Qadeer & Rehof. [Context-bounded model checking of concurrent software](#). 2005.

[SV'06] Sen & Viswanathan. [Model checking multithreaded programs with asynchronous atomic methods](#). 2006.

# Outline

## Model of concurrent recursive programs

### Concurrent visibly pushdown automata (CVPA)

## Global specification

- Multi-stack visibly pushdown automata (MVPA)
- Monadic Second-Order Logic (MSO)

## Synthesis

- $MVPA \rightarrow CVPA$
- $MSO \rightarrow CVPA$

# Outline

## Model of concurrent recursive programs

Concurrent visibly pushdown automata ( $\text{CVPA}$ )

## Global specification

- Multi-stack visibly pushdown automata ( $\text{MVPA}$ )
- Monadic Second-Order Logic ( $\text{MSO}$ )

## Synthesis

- $\text{MVPA} \rightarrow \text{CVPA}$
- $\text{MSO} \rightarrow \text{CVPA}$

# Outline

## Model of concurrent recursive programs

Concurrent visibly pushdown automata ( $CVPVA$ )

## Global specification

- Multi-stack visibly pushdown automata ( $MVPA$ )
- Monadic Second-Order Logic ( $MSO$ )

## Synthesis

- $MVPA \rightarrow CVPVA$
- $MSO \rightarrow CVPVA$

## Example of a concurrent recursive program

```
p1(int x)
```

```
1 wait(turn=0); turn := 1;  
2 if x > 0 then return f(x,p1(x-1));  
3     else return x;
```

```
p2(int y)
```

```
1 wait(turn=1); turn := 0;  
2 if y > 0 then return g(y,p2(y-1));  
3     else return y;
```

## Example of a concurrent recursive program

```
p1(int x)
```

```
1 wait(turn=0); turn := 1;  
2 if x > 0 then return f(x,p1(x-1));  
3     else return x;
```

```
p2(int y)
```

```
1 wait(turn=1); turn := 0;  
2 if y > 0 then return g(y,p2(y-1));  
3     else return y;
```

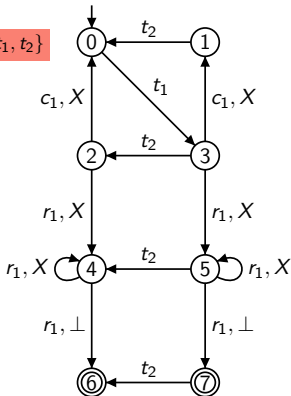
$$\Sigma_{p1} = \{c_1, r_1, t_1, t_2\}$$
$$\Sigma_{p2} = \{c_2, r_2, t_1, t_2\}$$

# Example of a concurrent recursive program

```
p1(int x)
```

```
1 wait(turn=0); turn := 1;
2 if x > 0 then return f(x,p1(x-1));
3     else return x;
```

$\Sigma_{p1} = \{c_1, r_1, t_1, t_2\}$

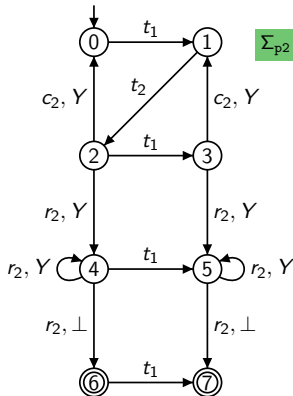


turn = 0      turn = 1

```
p2(int y)
```

```
1 wait(turn=1); turn := 0;
2 if y > 0 then return g(y,p2(y-1));
3     else return y;
```

$\Sigma_{p2} = \{c_2, r_2, t_1, t_2\}$



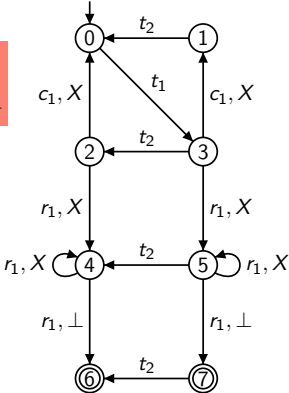
turn = 0      turn = 1

# Example of a concurrent recursive program

```

p1(int x)
1 wait(turn=0); turn :=1;
2 if x>0 then return f(x,p1(x-1));
3     else return x;
    
```

$Call_{p1} = \{c_1\}$   
 $Ret_{p1} = \{r_1\}$   
 $Int_{p1} = \{t_1, t_2\}$

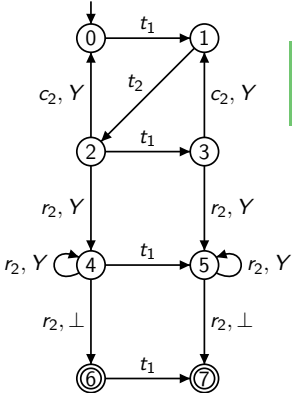


turn = 0      turn = 1

```

p2(int y)
1 wait(turn=1); turn :=0;
2 if y>0 then return g(y,p2(y-1));
3     else return y;
    
```

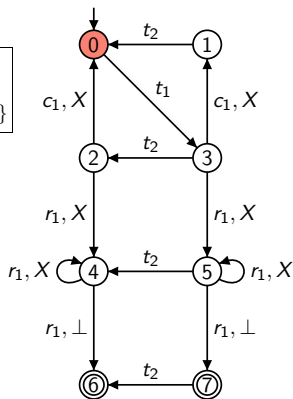
$Call_{p2} = \{c_2\}$   
 $Ret_{p2} = \{r_2\}$   
 $Int_{p2} = \{t_1, t_2\}$



turn = 0      turn = 1

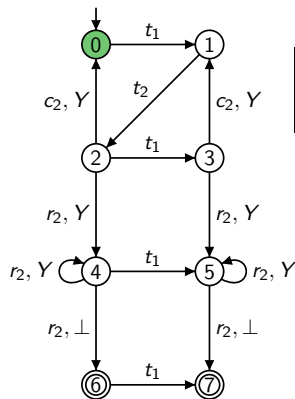
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



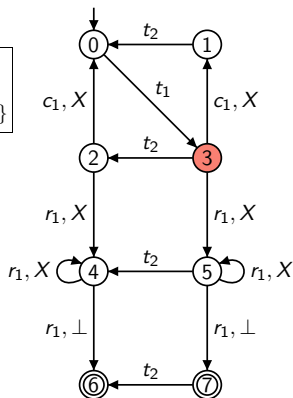
$t_1$

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



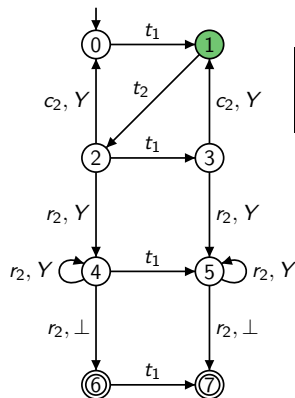
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



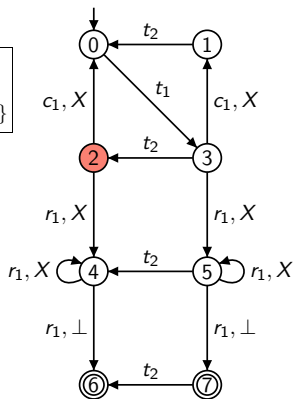
$t_1$   $t_2$

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



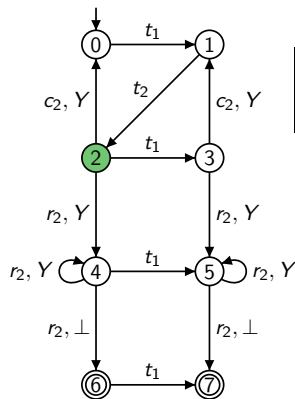
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



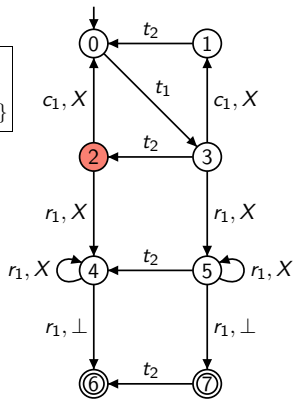
$t_1$   $t_2$   $c_2$

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



# Concurrent visibly pushdown automaton (CVPA)

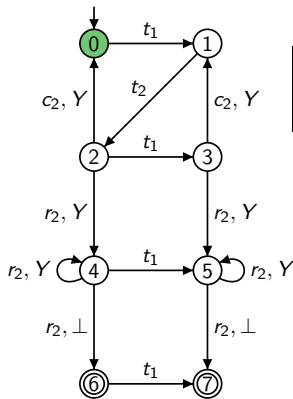
$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



⊥

$t_1$   $t_2$   $c_2$   $c_1$

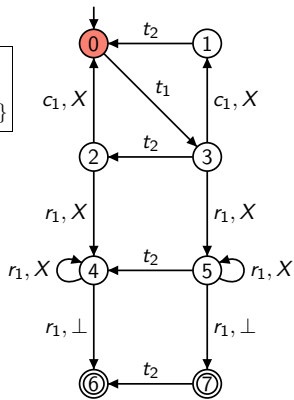
$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



$Y$   
 $\perp$

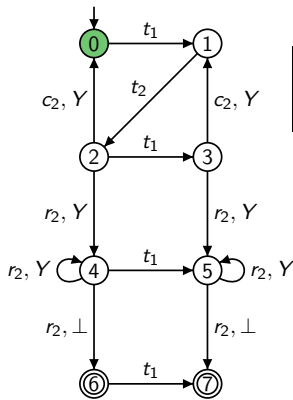
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



$X$   
 $\perp$

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$

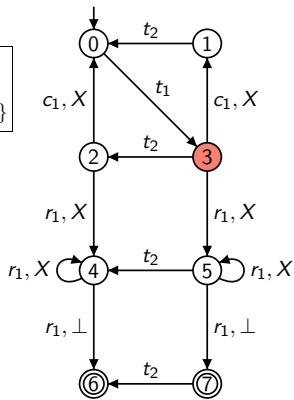


$Y$   
 $\perp$

$t_1$   $t_2$   $c_2$   $c_1$   $t_1$

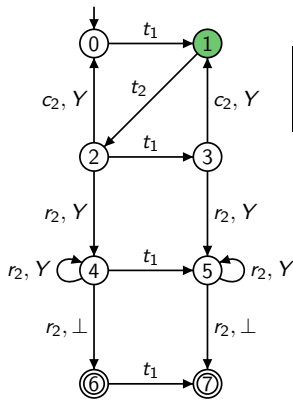
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



$X$   
 $\perp$

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$

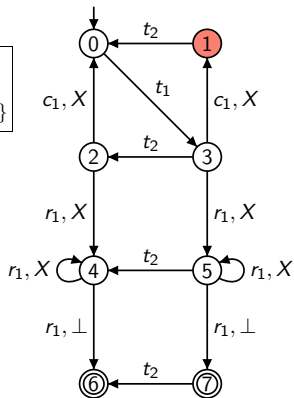


$Y$   
 $\perp$

$t_1$   $t_2$   $c_2$   $c_1$   $t_1$   $c_1$

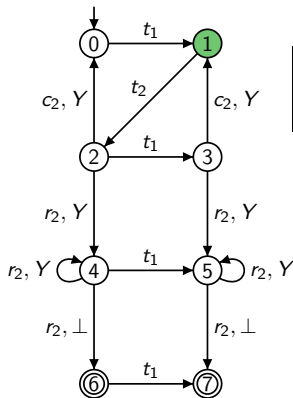
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



$X$   
 $X$   
 $\perp$

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$

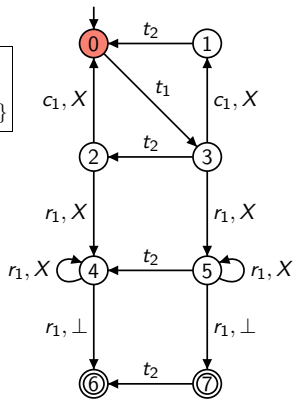


$Y$   
 $\perp$

$t_1$   $t_2$   $c_2$   $c_1$   $t_1$   $c_1$   $t_2$

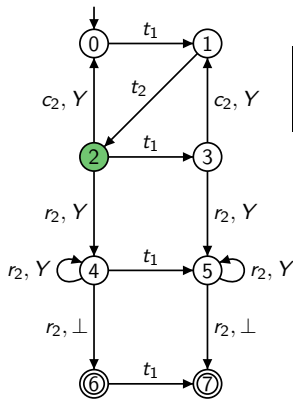
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



$X$   
 $X$   
 $\perp$

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$

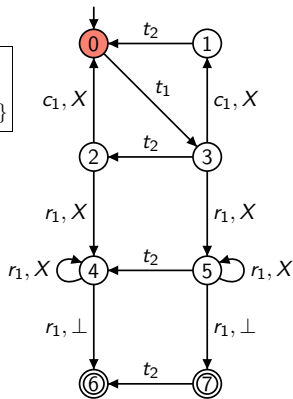


$Y$   
 $\perp$

$t_1$   $t_2$   $c_2$   $c_1$   $t_1$   $c_1$   $t_2$   $r_2$

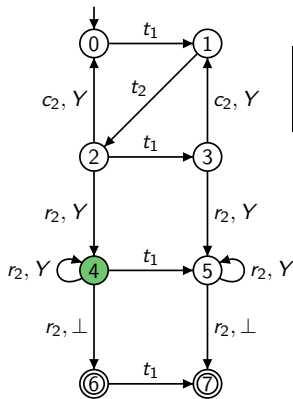
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



$X$   
 $X$   
 $\perp$

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$

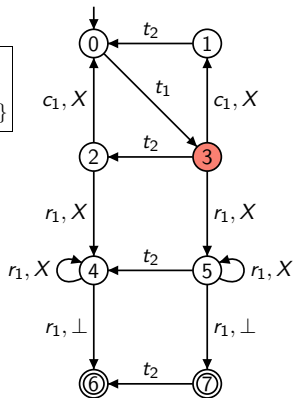


$\perp$

$t_1$   $t_2$   $c_2$   $c_1$   $t_1$   $c_1$   $t_2$   $r_2$   $t_1$

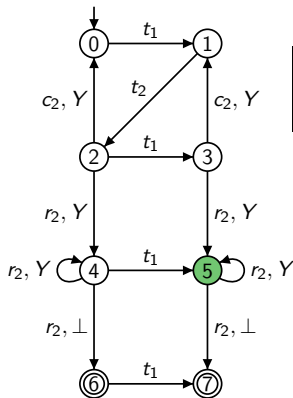
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



$\times$   
 $\times$   
 $\perp$

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$

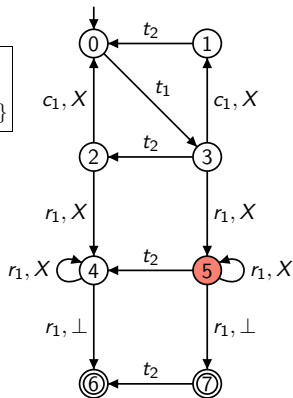


$\perp$

$t_1$   $t_2$   $c_2$   $c_1$   $t_1$   $c_1$   $t_2$   $r_2$   $t_1$   $r_1$

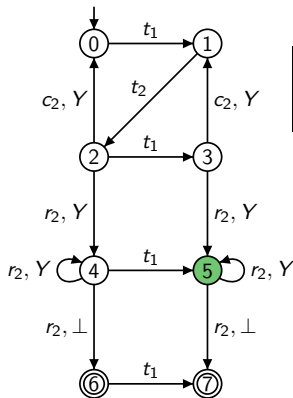
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



X  
⊥

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$

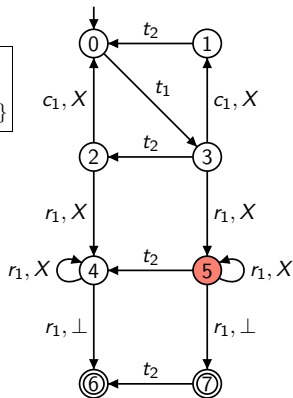


⊥

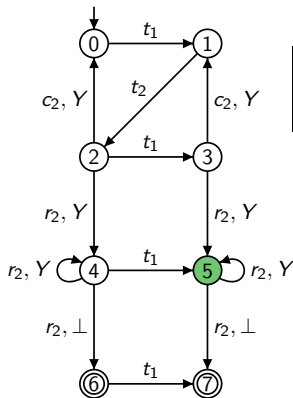
$t_1$   $t_2$  c<sub>2</sub> c<sub>1</sub>  $t_1$  c<sub>1</sub>  $t_2$  r<sub>2</sub>  $t_1$  r<sub>1</sub>  $r_1$

# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



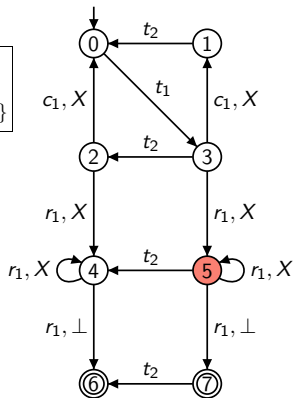
$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



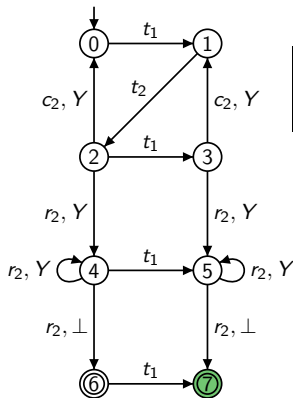
$t_1$   $t_2$   $c_2$   $c_1$   $t_1$   $c_1$   $t_2$   $r_2$   $t_1$   $r_1$   $r_1$   $r_2$

# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



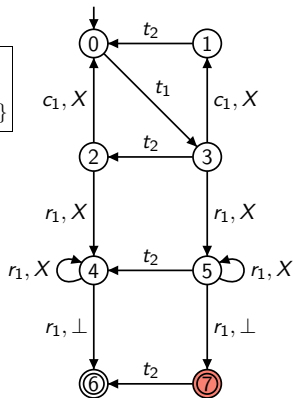
$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



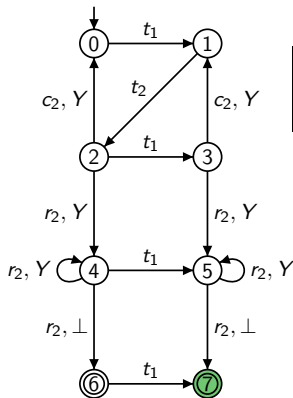
$t_1$   $t_2$   $c_2$   $c_1$   $t_1$   $c_1$   $t_2$   $r_2$   $t_1$   $r_1$   $r_1$   $r_2$   $r_1$

# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



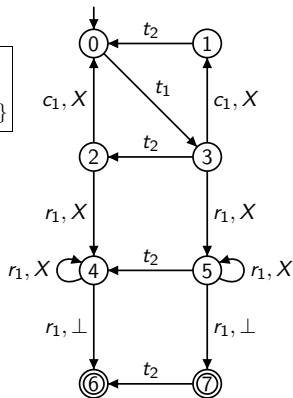
$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



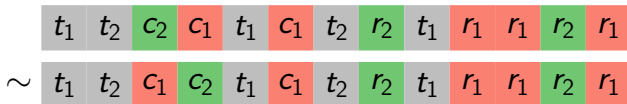
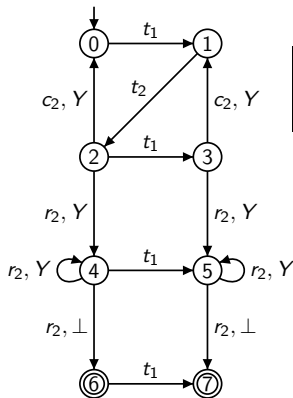
$t_1$   $t_2$   $c_2$   $c_1$   $t_1$   $c_1$   $t_2$   $r_2$   $t_1$   $r_1$   $r_1$   $r_2$   $r_1$

# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$

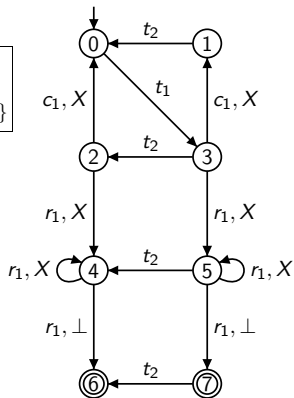


$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



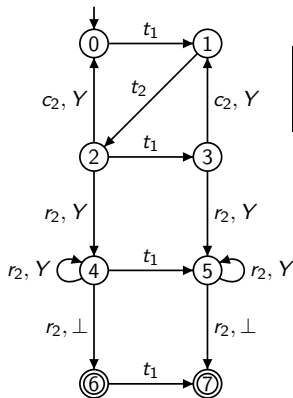
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$

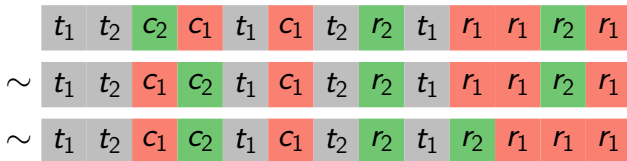


⊥

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$

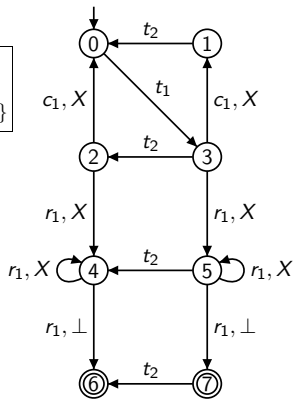


⊥



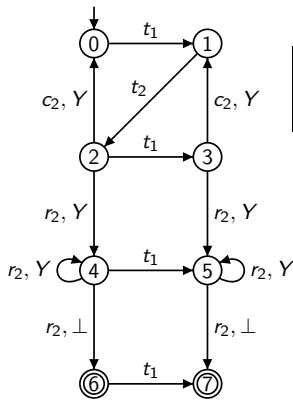
# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



⊥

$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



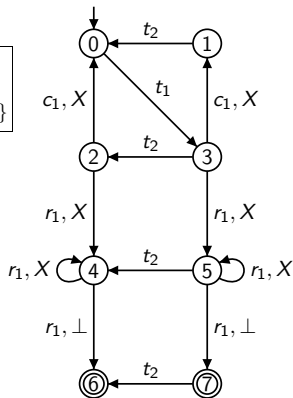
⊥

$$L(\mathcal{A}_p) = \{ t_1 ((c_1 t_2 + t_2 c_1) t_1)^n r_1^i (t_2 + \varepsilon) r_1^j \mid n, i, j \in \mathbb{N}, i + j = n + 1 \geq 2 \}$$

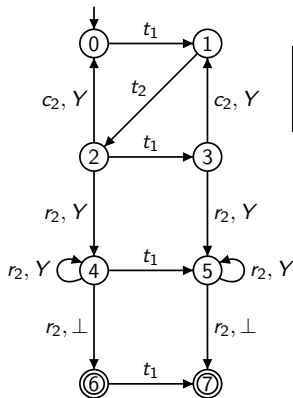
$$L(\mathcal{A}_q) = \{ t_1 t_2 ((c_2 t_1 + t_1 c_2) t_2)^n r_2^i (t_1 + \varepsilon) r_2^j \mid n, i, j \in \mathbb{N}, i + j = n + 1 \geq 2 \}$$

# Concurrent visibly pushdown automaton (CVPA)

$\text{Call}_{p_1} = \{c_1\}$   
 $\text{Ret}_{p_1} = \{r_1\}$   
 $\text{Int}_{p_1} = \{t_1, t_2\}$



$\text{Call}_{p_2} = \{c_2\}$   
 $\text{Ret}_{p_2} = \{r_2\}$   
 $\text{Int}_{p_2} = \{t_1, t_2\}$



$$L(\mathcal{A}_p) = \{ t_1 ((c_1 t_2 + t_2 c_1) t_1)^n r_1^i (t_2 + \varepsilon) r_1^j \mid n, i, j \in \mathbb{N}, i + j = n + 1 \geq 2 \}$$

$$L(\mathcal{A}_q) = \{ t_1 t_2 ((c_2 t_1 + t_1 c_2) t_2)^n r_2^i (t_1 + \varepsilon) r_2^j \mid n, i, j \in \mathbb{N}, i + j = n + 1 \geq 2 \}$$

$$L(\mathcal{A}) = \{ w \in \Sigma^* \mid w \upharpoonright \Sigma_p \in L(\mathcal{A}_p) \text{ and } w \upharpoonright \Sigma_q \in L(\mathcal{A}_q) \}$$

# The architecture of a concurrent recursive program

## Definition

We fix the following parameters:

- $\mathcal{P}$  a finite set of **processes**
- With  $p \in \mathcal{P}$ , we associate pairwise disjoint alphabets

$$(\text{Call}_p, \text{Ret}_p, \text{Int}_p)$$

such that  $(\text{Call}_p \cup \text{Ret}_p) \cap (\text{Call}_q \cup \text{Ret}_q) = \emptyset$  whenever  $p \neq q$ .

## Notation

- $\text{Call} = \bigcup_{p \in \mathcal{P}} \text{Call}_p$  call actions
- $\text{Ret} = \bigcup_{p \in \mathcal{P}} \text{Ret}_p$  return actions
- $\text{Int} = \bigcup_{p \in \mathcal{P}} \text{Int}_p \setminus (\text{Call} \cup \text{Ret})$  internal actions
- $\Sigma = \text{Call} \cup \text{Ret} \cup \text{Int}$

# The architecture of a concurrent recursive program

## Definition

We fix the following parameters:

- $\mathcal{P}$  a finite set of **processes**
- With  $p \in \mathcal{P}$ , we associate pairwise disjoint alphabets

$$(\text{Call}_p, \text{Ret}_p, \text{Int}_p)$$

such that  $(\text{Call}_p \cup \text{Ret}_p) \cap (\text{Call}_q \cup \text{Ret}_q) = \emptyset$  whenever  $p \neq q$ .

## Notation

- $\text{Call} = \bigcup_{p \in \mathcal{P}} \text{Call}_p$  call actions
- $\text{Ret} = \bigcup_{p \in \mathcal{P}} \text{Ret}_p$  return actions
- $\text{Int} = \bigcup_{p \in \mathcal{P}} \text{Int}_p \setminus (\text{Call} \cup \text{Ret})$  internal actions
- $\Sigma = \text{Call} \cup \text{Ret} \cup \text{Int}$

# Definition of concurrent visibly pushdown automaton

## Definition

A **concurrent visibly pushdown automaton** (CVPA) consists of:

- $S_p$  a finite set of local states for every  $p \in \mathcal{P}$
- $\Gamma$  a set of stack symbols
- a set  $\delta_a$  of transitions for every action  $a$ :

$$\delta_a \subseteq S_a \times \Gamma \times S_a \quad \text{if } a \in \text{Call}$$

$$\delta_a \subseteq S_a \times (\Gamma \cup \{\perp\}) \times S_a \quad \text{if } a \in \text{Ret}$$

$$\delta_a \subseteq S_a \times S_a \quad \text{if } a \in \text{Int}$$

where  $s \in S_a$  contains a local state for every process  $p$  with

$$a \in \text{Call}_p \cup \text{Ret}_p \cup \text{Int}_p$$

- global initial and global final states

- set of configurations:  $\prod_{p \in \mathcal{P}} S_p \times \prod_{p \in \mathcal{P}} \Gamma^* \{\perp\}$

# Definition of concurrent visibly pushdown automaton

## Definition

A **concurrent visibly pushdown automaton** (CVPA) consists of:

- $S_p$  a finite set of local states for every  $p \in \mathcal{P}$
- $\Gamma$  a set of stack symbols
- a set  $\delta_a$  of transitions for every action  $a$ :

$$\delta_a \subseteq S_a \times \Gamma \times S_a \quad \text{if } a \in \text{Call}$$

$$\delta_a \subseteq S_a \times (\Gamma \cup \{\perp\}) \times S_a \quad \text{if } a \in \text{Ret}$$

$$\delta_a \subseteq S_a \times S_a \quad \text{if } a \in \text{Int}$$

where  $s \in S_a$  contains a local state for every process  $p$  with

$$a \in \text{Call}_p \cup \text{Ret}_p \cup \text{Int}_p$$

- global initial and global final states

- set of configurations:  $\prod_{p \in \mathcal{P}} S_p \times \prod_{p \in \mathcal{P}} \Gamma^* \{\perp\}$

# Definition of concurrent visibly pushdown automaton

## Definition

A **concurrent visibly pushdown automaton** (CVPA) consists of:

- $S_p$  a finite set of local states for every  $p \in \mathcal{P}$
- $\Gamma$  a set of stack symbols
- a set  $\delta_a$  of transitions for every action  $a$ :

$$\delta_a \subseteq S_a \times \Gamma \times S_a \quad \text{if } a \in \text{Call}$$

$$\delta_a \subseteq S_a \times (\Gamma \cup \{\perp\}) \times S_a \quad \text{if } a \in \text{Ret}$$

$$\delta_a \subseteq S_a \times S_a \quad \text{if } a \in \text{Int}$$

where  $s \in S_a$  contains a local state for every process  $p$  with

$$a \in \text{Call}_p \cup \text{Ret}_p \cup \text{Int}_p$$

- global initial and global final states

- set of configurations:  $\prod_{p \in \mathcal{P}} S_p \times \prod_{p \in \mathcal{P}} \Gamma^* \{\perp\}$

# Definition of concurrent visibly pushdown automaton

## Definition

A **concurrent visibly pushdown automaton** (CVPA) consists of:

- $S_p$  a finite set of local states for every  $p \in \mathcal{P}$
- $\Gamma$  a set of stack symbols
- a set  $\delta_a$  of transitions for every action  $a$ :

$$\delta_a \subseteq S_a \times \Gamma \times S_a \quad \text{if } a \in \text{Call}$$

$$\delta_a \subseteq S_a \times (\Gamma \cup \{\perp\}) \times S_a \quad \text{if } a \in \text{Ret}$$

$$\delta_a \subseteq S_a \times S_a \quad \text{if } a \in \text{Int}$$

where  $s \in S_a$  contains a local state for every process  $p$  with

$$a \in \text{Call}_p \cup \text{Ret}_p \cup \text{Int}_p$$

- global initial and global final states

- set of configurations:  $\prod_{p \in \mathcal{P}} S_p \times \prod_{p \in \mathcal{P}} \Gamma^* \{\perp\}$

# Definition of concurrent visibly pushdown automaton

## Definition

A **concurrent visibly pushdown automaton** (CVPA) consists of:

- $S_p$  a finite set of local states for every  $p \in \mathcal{P}$
- $\Gamma$  a set of stack symbols
- a set  $\delta_a$  of transitions for every action  $a$ :

$$\delta_a \subseteq S_a \times \Gamma \times S_a \quad \text{if } a \in \text{Call}$$

$$\delta_a \subseteq S_a \times (\Gamma \cup \{\perp\}) \times S_a \quad \text{if } a \in \text{Ret}$$

$$\delta_a \subseteq S_a \times S_a \quad \text{if } a \in \text{Int}$$

where  $s \in S_a$  contains a local state for every process  $p$  with

$$a \in \text{Call}_p \cup \text{Ret}_p \cup \text{Int}_p$$

- global initial and global final states

- set of configurations:  $\prod_{p \in \mathcal{P}} S_p \times \prod_{p \in \mathcal{P}} \Gamma^* \{\perp\}$

## Semantics of CVPA ...

... can also be described in terms of MVPA:

### Definition ([LMP'07])

A **multi-stack visibly pushdown automaton** (MVPA) is a tuple

$$(S, \Gamma, \Delta, \iota, F)$$

- $\Delta \subseteq$ 
  - $S \times \text{Call} \times \Gamma \times S$
  - $\cup S \times \text{Ret} \times (\Gamma \cup \{\perp\}) \times S$
  - $\cup S \times \text{Int} \times S$

- set of configurations:  $S \times \prod_{p \in \mathcal{P}} \Gamma^* \{\perp\}$
- notion of a 'process' meaningless
- processes only determine number of stacks

---

[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

## Semantics of CVPA ...

... can also be described in terms of MVPA:

### Definition ([LMP'07])

A **multi-stack visibly pushdown automaton** (MVPA) is a tuple

$$(S, \Gamma, \Delta, \iota, F)$$

$$\begin{aligned} \bullet \Delta \subseteq & S \times \text{Call} \times \Gamma \times S \\ & \cup S \times \text{Ret} \times (\Gamma \cup \{\perp\}) \times S \\ & \cup S \times \text{Int} \times S \end{aligned}$$

- set of configurations:  $S \times \prod_{p \in \mathcal{P}} \Gamma^* \{\perp\}$
- notion of a 'process' meaningless
- processes only determine number of stacks

---

[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

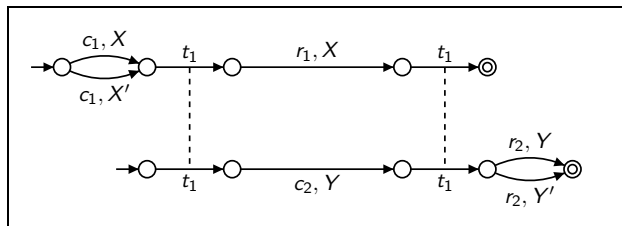
# CVPA and MVPA

Architecture     $\mathcal{P} = \{1, 2\}$      $\text{Call}_p = \{c_p\}$      $\text{Ret}_p = \{r_p\}$      $\text{Int}_p = \{t_1, t_2\}$

MVPA



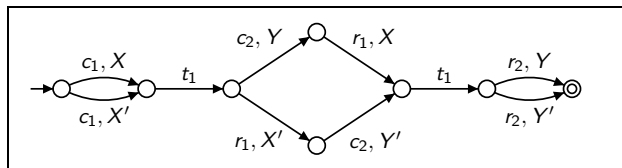
CVPA



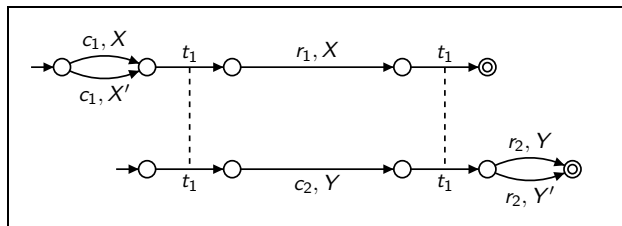
# CVPA and MVPA

Architecture  $\mathcal{P} = \{1, 2\}$   $\text{Call}_p = \{c_p\}$   $\text{Ret}_p = \{r_p\}$   $\text{Int}_p = \{t_1, t_2\}$

MVPA  
[LMP'07]



CVPA



[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

# Specification formalisms for distributed systems

## Specification

finite automata

rational expressions

temporal logics (LTL, CTL, LTrL, ...)

MVPA

monadic second-order logic

...

*Synthesis*

## Implementation

asynchronous automata

message-passing automata

CVPA

...

# Specification formalisms for distributed systems

## Specification

finite automata  
rational expressions  
temporal logics (LTL, CTL, LTrL, ...)  
MVPA  
monadic second-order logic  
...

*Synthesis*

## Implementation

asynchronous automata  
message-passing automata  
CVPA  
...

# Specification formalisms for distributed systems

## Specification

finite automata  
rational expressions  
temporal logics (LTL, CTL, LTrL, ...)  
**MVPA**  
monadic second-order logic  
...

*Synthesis*

## Implementation

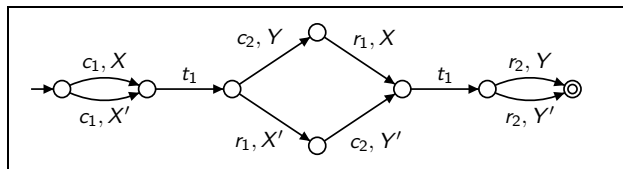
asynchronous automata  
message-passing automata  
**CVPA**  
...

# From MVPA to CVPA

Architecture  $\mathcal{P} = \{1, 2\}$   $\text{Call}_p = \{c_p\}$   $\text{Ret}_p = \{r_p\}$   $\text{Int}_p = \{t_1, t_2\}$

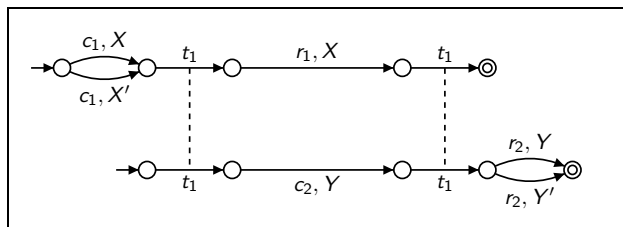
+

MVPA



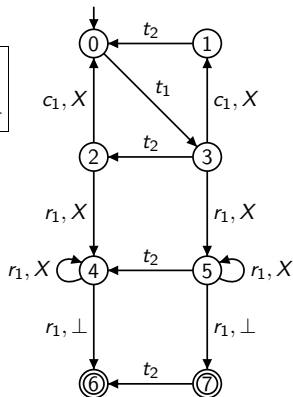
$\Downarrow$

CVPA

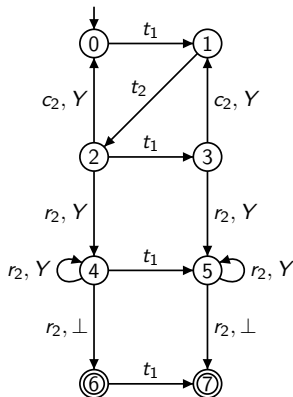


# Closure property of CVPA

$\text{Call}_p = \{c_1\}$   
 $\text{Ret}_p = \{r_1\}$   
 $\text{Int}_p = \{t_1, t_2\}$



$\text{Call}_q = \{c_2\}$   
 $\text{Ret}_q = \{r_2\}$   
 $\text{Int}_q = \{t_1, t_2\}$



# Zielonka's Theorem

## Remark

If  $\Sigma = \text{Int}$  (i.e., every action is internal), then

- an MVPA is a finite automaton.
- a CVPA an asynchronous automaton.

## Theorem ([Zie'87])

Let  $L \subseteq \Sigma^*$  be a  $\sim$ -closed regular language.

There is an asynchronous automaton  $\mathcal{C}$  such that  $L(\mathcal{C}) = L$ .

---

[Zie'87] Zielonka. [Notes on finite asynchronous automata](#). 1987.

# From MVPA to CVPA

## Theorem

Let  $\mathcal{A}$  be an MVPA over  $\tilde{\Sigma}$  such that  $L(\mathcal{A})$  is  $\sim_{\tilde{\Sigma}}$ -closed.

There is a CVPA  $\mathcal{C}$  over  $\tilde{\Sigma}$  such that  $L(\mathcal{C}) = L(\mathcal{A})$ . The size of  $\mathcal{C}$  is

- doubly exponential in  $|\mathcal{A}|$
- triply exponential in  $|\Sigma|$

## Proof

- Interpret  $\mathcal{A} = (S, \Gamma, \Delta, \iota, F)$  as finite automaton over  $\Sigma \times \Gamma$ .
- Apply Zielonka's Theorem to obtain CVPA  $\mathcal{C}$  over  $((\emptyset, \emptyset, \Sigma_p \times \Gamma))_{p \in \mathcal{P}}$ .
- Interpret  $\mathcal{C}$  as CVPA over  $\tilde{\Sigma}$ .

# From MVPA to CVPA

## Theorem

Let  $\mathcal{A}$  be an MVPA over  $\tilde{\Sigma}$  such that  $L(\mathcal{A})$  is  $\sim_{\tilde{\Sigma}}$ -closed.

There is a CVPA  $\mathcal{C}$  over  $\tilde{\Sigma}$  such that  $L(\mathcal{C}) = L(\mathcal{A})$ . The size of  $\mathcal{C}$  is

- doubly exponential in  $|\mathcal{A}|$
- triply exponential in  $|\Sigma|$

## Proof

- Interpret  $\mathcal{A} = (S, \Gamma, \Delta, \iota, F)$  as finite automaton over  $\Sigma \times \Gamma$ .
- Apply Zielonka's Theorem to obtain CVPA  $\mathcal{C}$  over  $((\emptyset, \emptyset, \Sigma_p \times \Gamma))_{p \in \mathcal{P}}$ .
- Interpret  $\mathcal{C}$  as CVPA over  $\tilde{\Sigma}$ .

# Is a specification implementable?

## Theorem ([Mus'94,PWW'96])

*The following problem is PSPACE-complete:*

INPUT: Architecture with  $\Sigma = \text{Int} + \text{MVPA } \mathcal{A}$

QUESTION: Is  $L(\mathcal{A}) \sim$ -closed?

## Theorem

*The following problem is undecidable:*

INPUT: Architecture + MVPA  $\mathcal{A}$

QUESTION: Is  $L(\mathcal{A}) \sim$ -closed?

---

[Mus'94] Muscholl. *Über die Erkennbarkeit unendlicher Spuren*. 1994

[PWW'96] Peled & Wilke & Wolper. *An algorithmic approach for checking closure properties of temporal logic specifications and  $\omega$ -regular languages*. 1996.

# Is a specification implementable?

## Theorem ([Mus'94,PWW'96])

*The following problem is PSPACE-complete:*

INPUT: Architecture with  $\Sigma = \text{Int} + \text{MVPA } \mathcal{A}$

QUESTION: Is  $L(\mathcal{A}) \sim$ -closed?

## Theorem

*The following problem is undecidable:*

INPUT: Architecture + MVPA  $\mathcal{A}$

QUESTION: Is  $L(\mathcal{A}) \sim$ -closed?

---

[Mus'94] Muscholl *Über die Erkennbarkeit unendlicher Spuren*. 1994

[PWW'96] Peled & Wilke & Wolper. *An algorithmic approach for checking closure properties of temporal logic specifications and  $\omega$ -regular languages*. 1996.

## Restriction to $k$ -phase words

### Definition ([LMP'07])

Let  $k \in \mathbb{N}$ . A word  $w \in \Sigma^*$  is a  **$k$ -phase word** if it can be written as  $w_1 \cdot \dots \cdot w_k$  where  $w_i \in (\text{Call} \cup \text{Int} \cup \text{Ret}_p)^*$  for some  $p \in \mathcal{P}$ .

---

[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

# Restriction to $k$ -phase words

## Definition ([LMP'07])

Let  $k \in \mathbb{N}$ . A word  $w \in \Sigma^*$  is a  **$k$ -phase word** if it can be written as  $w_1 \cdot \dots \cdot w_k$  where  $w_i \in (\text{Call} \cup \text{Int} \cup \text{Ret}_p)^*$  for some  $p \in \mathcal{P}$ .

$t_1$   $t_2$   $c_2$   $c_1$   $t_1$   $c_1$   $t_2$   $r_2$   $t_1$   $r_1$   $r_1$   $r_2$   $r_1$

$\sim$   $t_1$   $t_2$   $c_1$   $c_2$   $t_1$   $c_1$   $t_2$   $r_2$   $t_1$   $r_1$   $r_1$   $r_1$   $r_2$

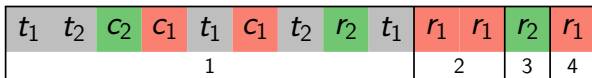
$\sim$   $t_1$   $t_2$   $c_1$   $c_2$   $t_1$   $c_1$   $t_2$   $r_2$   $t_1$   $r_2$   $r_1$   $r_1$   $r_1$

[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

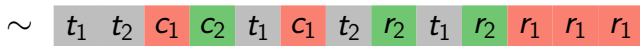
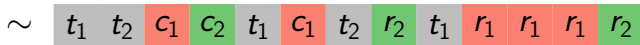
# Restriction to $k$ -phase words

## Definition ([LMP'07])

Let  $k \in \mathbb{N}$ . A word  $w \in \Sigma^*$  is a  **$k$ -phase word** if it can be written as  $w_1 \cdot \dots \cdot w_k$  where  $w_i \in (\text{Call} \cup \text{Int} \cup \text{Ret}_p)^*$  for some  $p \in \mathcal{P}$ .



4-phase word

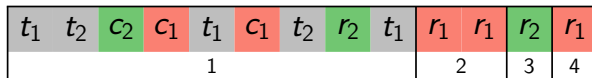


[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

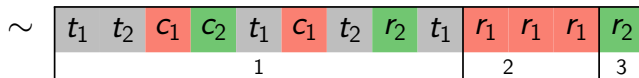
# Restriction to $k$ -phase words

## Definition ([LMP'07])

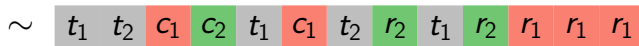
Let  $k \in \mathbb{N}$ . A word  $w \in \Sigma^*$  is a  **$k$ -phase word** if it can be written as  $w_1 \cdot \dots \cdot w_k$  where  $w_i \in (\text{Call} \cup \text{Int} \cup \text{Ret}_p)^*$  for some  $p \in \mathcal{P}$ .



4-phase word



3-phase word

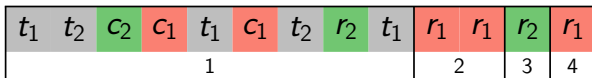


[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

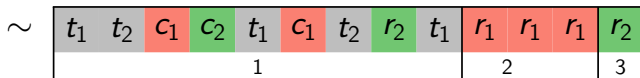
# Restriction to $k$ -phase words

## Definition ([LMP'07])

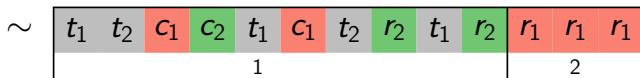
Let  $k \in \mathbb{N}$ . A word  $w \in \Sigma^*$  is a  **$k$ -phase word** if it can be written as  $w_1 \cdot \dots \cdot w_k$  where  $w_i \in (\text{Call} \cup \text{Int} \cup \text{Ret}_p)^*$  for some  $p \in \mathcal{P}$ .



4-phase word



3-phase word



2-phase word

[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

# MVPA and $k$ -phase words

## Notation

- Let  $W_k$  denote the set of  $k$ -phase words.
- For an MVPA  $\mathcal{A}$ , let  $L_k(\mathcal{A}) = L(\mathcal{A}) \cap W_k$ .

## Theorem ([LMP'07])

*The following problem is decidable:*

INPUT: Architecture + MVPA  $\mathcal{A}$  +  $k \in \mathbb{N}$

QUESTION: Is  $L_k(\mathcal{A})$  empty?

*The time complexity is doubly exponential wrt.  $k$ .*

## Theorem ([LMP'07])

*Let  $k \in \mathbb{N}$  and let  $\mathcal{A}$  be an MVPA. One can effectively construct an MVPA  $\mathcal{A}'$  such that  $L(\mathcal{A}') = \Sigma^* \setminus L_k(\mathcal{A})$ .*

---

[LMP'07] La Torre & Madhusudan & Parlato. A Robust Class of Context-Sensitive Languages. 2007.

# MVPA and $k$ -phase words

## Notation

- Let  $W_k$  denote the set of  $k$ -phase words.
- For an MVPA  $\mathcal{A}$ , let  $L_k(\mathcal{A}) = L(\mathcal{A}) \cap W_k$ .

## Theorem ([LMP'07])

*The following problem is decidable:*

INPUT: Architecture + MVPA  $\mathcal{A}$  +  $k \in \mathbb{N}$

QUESTION: Is  $L_k(\mathcal{A})$  empty?

*The time complexity is doubly exponential wrt.  $k$ .*

## Theorem ([LMP'07])

*Let  $k \in \mathbb{N}$  and let  $\mathcal{A}$  be an MVPA. One can effectively construct an MVPA  $\mathcal{A}'$  such that  $L(\mathcal{A}') = \Sigma^* \setminus L_k(\mathcal{A})$ .*

---

[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

# MVPA and $k$ -phase words

## Notation

- Let  $W_k$  denote the set of  $k$ -phase words.
- For an MVPA  $\mathcal{A}$ , let  $L_k(\mathcal{A}) = L(\mathcal{A}) \cap W_k$ .

## Theorem ([LMP'07])

*The following problem is decidable:*

INPUT: Architecture + MVPA  $\mathcal{A}$  +  $k \in \mathbb{N}$

QUESTION: Is  $L_k(\mathcal{A})$  empty?

*The time complexity is doubly exponential wrt.  $k$ .*

## Theorem ([LMP'07])

Let  $k \in \mathbb{N}$  and let  $\mathcal{A}$  be an MVPA. One can effectively construct an MVPA  $\mathcal{A}'$  such that  $L(\mathcal{A}') = \Sigma^* \setminus L_k(\mathcal{A})$ .

---

[LMP'07] La Torre & Madhusudan & Parlato. *A Robust Class of Context-Sensitive Languages*. 2007.

# A decidable sufficient criterion for implementability

## Definition

A set  $L \subseteq W_k$  is a  **$k$ -phase representation** if, for all equivalent  $uabv, ubav \in W_k$ , we have  $uabv \in L$  iff  $ubav \in L$ .

# A decidable sufficient criterion for implementability

## Definition

A set  $L \subseteq W_k$  is a  **$k$ -phase representation** if, for all equivalent  $uabv, ubav \in W_k$ , we have  $uabv \in L$  iff  $ubav \in L$ .

$\left\{ u \text{ } t_1 \text{ } r_1^m \text{ } r_2^n \mid m, n \geq 2, u \in \{c_1, c_2\}^*, |u|_{c_1} = m, |u|_{c_2} = n \right\}$   
is a 2-phase representation.

# A decidable sufficient criterion for implementability

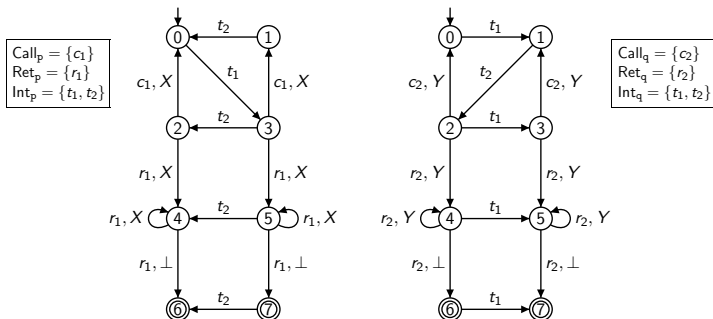
## Definition

A set  $L \subseteq W_k$  is a  **$k$ -phase representation** if, for all equivalent  $uabv, ubav \in W_k$ , we have  $uabv \in L$  iff  $ubav \in L$ .

$\left\{ u \text{ } t_1 \text{ } r_1^m \text{ } r_2^n \mid m, n \geq 2, u \in \{c_1, c_2\}^*, |u|_{c_1} = m, |u|_{c_2} = n \right\}$   
is a 2-phase representation.

Any  $\sim$ -closed subset of  $W_k$  is a  $k$ -phase representation.

# The closure of a $k$ -phase representation

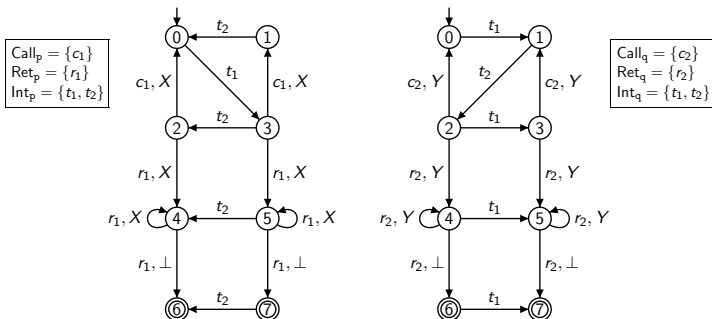


$$L(\mathcal{A}_p) = \{ t_1 ((c_1 t_2 + t_2 c_1) t_1)^n r_1^i (t_2 + \varepsilon) r_1^j \mid n, i, j \in \mathbb{N}, i + j = n + 1 \geq 2 \}$$

$$L(\mathcal{A}_q) = \{ t_1 t_2 ((c_2 t_1 + t_1 c_2) t_2)^n r_2^i (t_1 + \varepsilon) r_2^j \mid n, i, j \in \mathbb{N}, i + j = n + 1 \geq 2 \}$$

$$L(\mathcal{A}) = \{ w \in \Sigma^* \mid w \upharpoonright_{\Sigma_p} \in L(\mathcal{A}_p) \text{ and } w \upharpoonright_{\Sigma_q} \in L(\mathcal{A}_q) \}$$

# The closure of a $k$ -phase representation



$$L(\mathcal{A}_p) = \{ t_1 ((c_1 t_2 + t_2 c_1) t_1)^n r_1^i (t_2 + \varepsilon) r_1^j \mid n, i, j \in \mathbb{N}, i + j = n + 1 \geq 2 \}$$

$$L(\mathcal{A}_q) = \{ t_1 t_2 ((c_2 t_1 + t_1 c_2) t_2)^n r_2^i (t_1 + \varepsilon) r_2^j \mid n, i, j \in \mathbb{N}, i + j = n + 1 \geq 2 \}$$

$$L(\mathcal{A}) = \{ w \in \Sigma^* \mid w \upharpoonright_{\Sigma_p} \in L(\mathcal{A}_p) \text{ and } w \upharpoonright_{\Sigma_q} \in L(\mathcal{A}_q) \}$$

$L_2(\mathcal{A})$  is a 2-phase representation and we have  $[L_2(\mathcal{A})]_{\sim} = L(\mathcal{A})$ .

# From MVPA to CVPA

## Theorem

Let  $\mathcal{A}$  be an MVPA such that  $L_k(\mathcal{A})$  is a  $k$ -phase representation.

There is a CVPA  $\mathcal{C}$  such that  $L(\mathcal{C}) = [L_k(\mathcal{A})]_{\sim}$ . The size of  $\mathcal{C}$  is

- doubly exponential in  $|\mathcal{A}|$  and  $k$
- triply exponential in  $|\Sigma|$

# Decidability of sufficient criterion

## Theorem

*The following problems are decidable in elementary time:*

INPUT:            *Architecture + MVPA  $\mathcal{A}$  +  $k \in \mathbb{N}$*

QUESTION 1:    *Is  $L_k(\mathcal{A})$   $\sim$ -closed?*

QUESTION 2:    *Is  $L_k(\mathcal{A})$  a  $k$ -phase representation?*

# Decidability of sufficient criterion

## Theorem

*The following problems are decidable in elementary time:*

INPUT:            *Architecture + MVPA  $\mathcal{A}$  +  $k \in \mathbb{N}$*

QUESTION 1:    *Is  $L_k(\mathcal{A})$   $\sim$ -closed?*

QUESTION 2:    *Is  $L_k(\mathcal{A})$  a  $k$ -phase representation?*

## Proof (Question 1)

- From  $\mathcal{A}$  construct MVPA  $\mathcal{A}'$  such that  $L(\mathcal{A}') = \Sigma^* \setminus L_k(\mathcal{A})$ .
- Build MVPA  $\mathcal{B}$  recognizing words of the form  $uabv$  with  $(a, b)$  independent,  $uabv \in L(\mathcal{A})$ , and  $ubav \in L(\mathcal{A}')$ .
- $L_k(\mathcal{A})$  is not  $\sim$ -closed iff  $L_k(\mathcal{B}) \neq \emptyset$ .
- For Question 2, build  $\mathcal{A}'$  such that  $L(\mathcal{A}') = (\Sigma^* \setminus L_k(\mathcal{A})) \cap W_k$ .

# Decidability of sufficient criterion

## Theorem

*The following problems are decidable in elementary time:*

INPUT:            *Architecture + MVPA  $\mathcal{A}$  +  $k \in \mathbb{N}$*

QUESTION 1:    *Is  $L_k(\mathcal{A})$   $\sim$ -closed?*

QUESTION 2:    *Is  $L_k(\mathcal{A})$  a  $k$ -phase representation?*

## Proof (Question 1)

- From  $\mathcal{A}$  construct MVPA  $\mathcal{A}'$  such that  $L(\mathcal{A}') = \Sigma^* \setminus L_k(\mathcal{A})$ .
- Build MVPA  $\mathcal{B}$  recognizing words of the form  $uabv$  with  $(a, b)$  independent,  $uabv \in L(\mathcal{A})$ , and  $ubav \in L(\mathcal{A}')$ .
- $L_k(\mathcal{A})$  is not  $\sim$ -closed iff  $L_k(\mathcal{B}) \neq \emptyset$ .
- For Question 2, build  $\mathcal{A}'$  such that  $L(\mathcal{A}') = (\Sigma^* \setminus L_k(\mathcal{A})) \cap W_k$ .

# Decidability of sufficient criterion

## Theorem

*The following problems are decidable in elementary time:*

INPUT:            *Architecture + MVPA  $\mathcal{A}$  +  $k \in \mathbb{N}$*

QUESTION 1:    *Is  $L_k(\mathcal{A})$   $\sim$ -closed?*

QUESTION 2:    *Is  $L_k(\mathcal{A})$  a  $k$ -phase representation?*

## Proof (Question 1)

- From  $\mathcal{A}$  construct MVPA  $\mathcal{A}'$  such that  $L(\mathcal{A}') = \Sigma^* \setminus L_k(\mathcal{A})$ .
- Build MVPA  $\mathcal{B}$  recognizing words of the form  $uabv$  with  $(a, b)$  independent,  $uabv \in L(\mathcal{A})$ , and  $ubav \in L(\mathcal{A}')$ .
- $L_k(\mathcal{A})$  is not  $\sim$ -closed iff  $L_k(\mathcal{B}) \neq \emptyset$ .
- For Question 2, build  $\mathcal{A}'$  such that  $L(\mathcal{A}') = (\Sigma^* \setminus L_k(\mathcal{A})) \cap W_k$ .

# Decidability of sufficient criterion

## Theorem

*The following problems are decidable in elementary time:*

INPUT:            *Architecture + MVPA  $\mathcal{A} + k \in \mathbb{N}$*

QUESTION 1:    *Is  $L_k(\mathcal{A})$   $\sim$ -closed?*

QUESTION 2:    *Is  $L_k(\mathcal{A})$  a  $k$ -phase representation?*

## Proof (Question 1)

- From  $\mathcal{A}$  construct MVPA  $\mathcal{A}'$  such that  $L(\mathcal{A}') = \Sigma^* \setminus L_k(\mathcal{A})$ .
- Build MVPA  $\mathcal{B}$  recognizing words of the form  $uabv$  with  $(a, b)$  independent,  $uabv \in L(\mathcal{A})$ , and  $ubav \in L(\mathcal{A}')$ .
- $L_k(\mathcal{A})$  is not  $\sim$ -closed iff  $L_k(\mathcal{B}) \neq \emptyset$ .
- For Question 2, build  $\mathcal{A}'$  such that  $L(\mathcal{A}') = (\Sigma^* \setminus L_k(\mathcal{A})) \cap W_k$ .

# Specification formalisms for distributed systems

## Specification

finite automata  
rational expressions  
temporal logics (LTL, CTL, LTrL, ...)  
MVPA  
monadic second-order logic  
...

*Synthesis*

## Implementation

asynchronous automata  
message-passing automata  
CVPA  
...

# Specification formalisms for distributed systems

## Specification

finite automata  
rational expressions  
temporal logics (LTL, CTL, LTrL, ...)  
MVPA  
monadic second-order logic  
...

*Synthesis*

## Implementation

asynchronous automata  
message-passing automata  
CVPA  
...

# From words to nested Mazurkiewicz traces

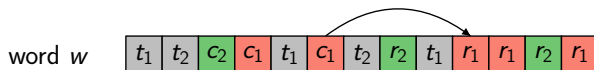
word  $w$ 

$t_1$	$t_2$	$c_2$	$c_1$	$t_1$	$c_1$	$t_2$	$r_2$	$t_1$	$r_1$	$r_1$	$r_2$	$r_1$
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

---

[AM'06] Alur & Madhusudan. Adding nesting structure to words. 2006.

## From words to nested Mazurkiewicz traces

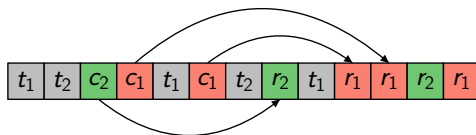


---

[AM'06] Alur & Madhusudan. Adding nesting structure to words. 2006.

# From words to nested Mazurkiewicz traces

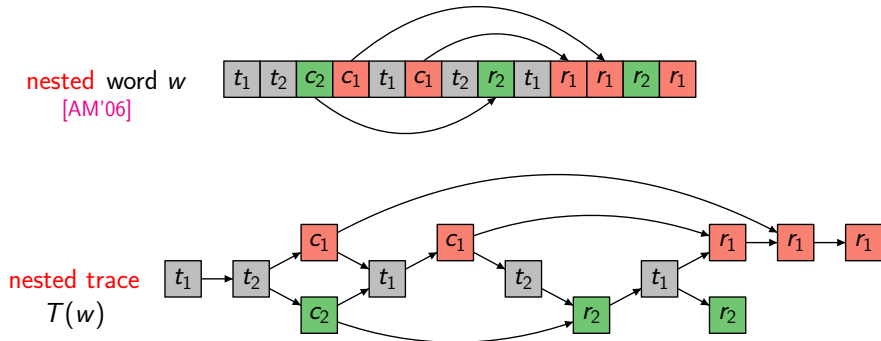
nested word  $w$   
[AM'06]



---

[AM'06] Alur & Madhusudan. Adding nesting structure to words. 2006.

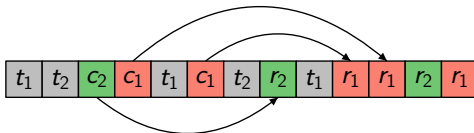
# From words to nested Mazurkiewicz traces



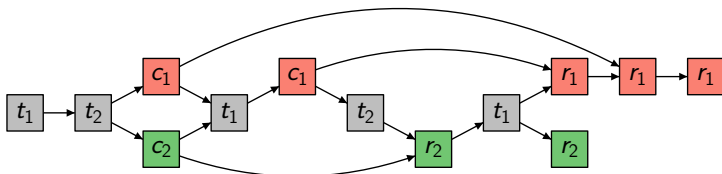
[AM'06] Alur & Madhusudan. Adding nesting structure to words. 2006.

# From words to nested Mazurkiewicz traces

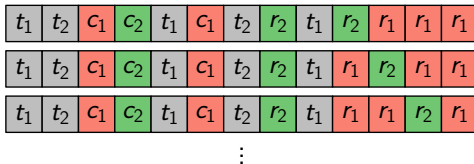
nested word  $w$   
[AM'06]



nested trace  
 $T(w)$

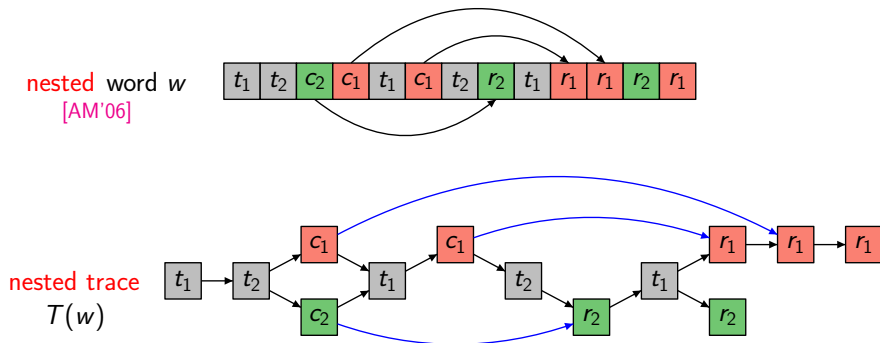


linearizations



[AM'06] Alur & Madhusudan. Adding nesting structure to words. 2006.

# From words to nested Mazurkiewicz traces

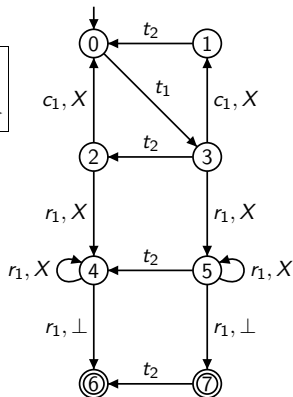


$$T(w) = (E, \leq, \mu, \lambda) \text{ where } \leq, \mu \subseteq E \times E \text{ and } \lambda : E \rightarrow \Sigma$$

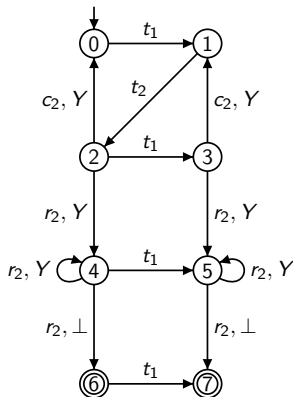
[AM'06] Alur & Madhusudan. Adding nesting structure to words. 2006.

# The nested-trace language of a CVPA $\mathcal{C}$

$\text{Call}_p = \{c_1\}$   
 $\text{Ret}_p = \{r_1\}$   
 $\text{Int}_p = \{t_1, t_2\}$



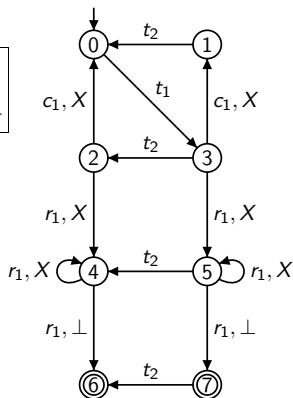
$\text{Call}_q = \{c_2\}$   
 $\text{Ret}_q = \{r_2\}$   
 $\text{Int}_q = \{t_1, t_2\}$



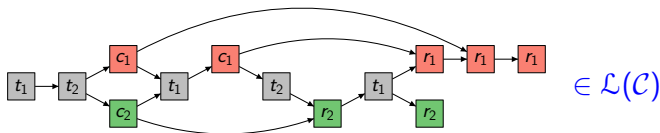
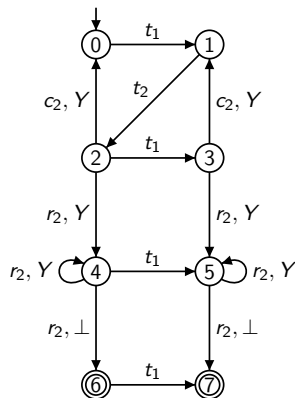
$$\mathcal{L}(\mathcal{C}) = \{T(w) \mid w \in L(\mathcal{C})\}$$

# The nested-trace language of a CVPA $\mathcal{C}$

$\text{Call}_p = \{c_1\}$   
 $\text{Ret}_p = \{r_1\}$   
 $\text{Int}_p = \{t_1, t_2\}$



$\text{Call}_q = \{c_2\}$   
 $\text{Ret}_q = \{r_2\}$   
 $\text{Int}_q = \{t_1, t_2\}$



# MSO logic for nested traces

## Definition

Monadic second-order logic MSO:

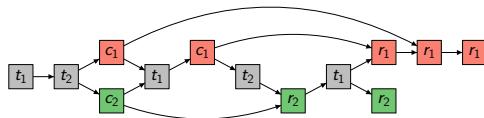
$$\begin{aligned} \varphi ::= & x \leq y \mid (x, y) \in \mu \mid \lambda(x) = a \mid \\ & x \in X \mid \neg \varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \end{aligned}$$

# MSO logic for nested traces

## Definition

Monadic second-order logic MSO:

$$\begin{aligned} \varphi ::= & x \leq y \mid (x, y) \in \mu \mid \lambda(x) = a \mid \\ & x \in X \mid \neg \varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \end{aligned}$$



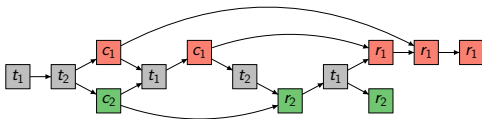
$$\models \forall x \forall y [(\lambda(x) \in \{c_1, c_2\} \wedge \lambda(y) \in \{r_1, r_2\}) \rightarrow x \leq y]$$

# MSO logic for nested traces

## Definition

Monadic second-order logic MSO:

$$\begin{aligned} \varphi ::= & x \leq y \mid (x, y) \in \mu \mid \lambda(x) = a \mid \\ & x \in X \mid \neg \varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \end{aligned}$$



$$\models \forall x \forall y [(\lambda(x) \in \{c_1, c_2\} \wedge \lambda(y) \in \{r_1, r_2\}) \rightarrow x \leq y]$$

$$\not\models \forall y [\lambda(y) \in \{r_1, r_2\} \rightarrow \exists x (x, y) \in \mu]$$

# CVPA cannot be complemented

## Theorem ([B'08])

- MVPA/CVPA *can in general not be complemented.*
- MSO *is strictly more expressive than MVPA/CVPA.*

---

[B'08] B. On the Expressive Power of 2-Stack Visibly Pushdown Automata. 2008.

# CVPA cannot be complemented

## Theorem ([B'08])

- *MVPA/CVPA can in general not be complemented.*
- *MSO is strictly more expressive than MVPA/CVPA.*

↪ restrict to *k-phase traces*

---

[B'08] B. On the Expressive Power of 2-Stack Visibly Pushdown Automata. 2008.

## $k$ -phase traces

### Definition

Let  $k \in \mathbb{N}$ . A  $k$ -phase trace is a nested trace  $T$  such that there is a  $k$ -phase word  $w \in \Sigma^*$  with  $T(w) = T$ .

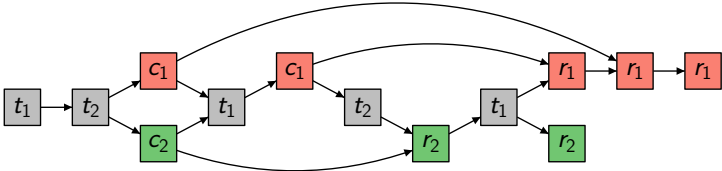
Let  $\text{Tr}_k$  denote the set of  $k$ -phase traces.

# k-phase traces

## Definition

Let  $k \in \mathbb{N}$ . A **k-phase trace** is a nested trace  $T$  such that there is a **k-phase word**  $w \in \Sigma^*$  with  $T(w) = T$ .

Let  $\text{Tr}_k$  denote the set of **k-phase traces**.



is a 2-phase trace

# MSO characterization of CVPA wrt. $k$ -phase traces

## Theorem

For every CVPA  $\mathcal{C}$ , there is  $\varphi \in \text{MSO}$  such that  $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{C})$ .

## Theorem

Let  $k \in \mathbb{N}$ . For every  $\varphi \in \text{MSO}$ , there is a CVPA  $\mathcal{C}$  such that

$$\mathcal{L}(\mathcal{C}) = \mathcal{L}(\varphi) \cap \text{Tr}_k$$

## Proof

- By induction.
- Lemma: If  $\mathcal{L} \subseteq \text{Tr}_k$  is recognizable, then so is  $\overline{\mathcal{L}} \cap \text{Tr}_k$ .
- $\text{Tr}_k$  is recognizable.
- Atomic formulas standard.
- CVPA are closed under union, intersection, and projection.

# MSO characterization of CVPA wrt. $k$ -phase traces

## Theorem

For every CVPA  $\mathcal{C}$ , there is  $\varphi \in \text{MSO}$  such that  $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{C})$ .

## Theorem

Let  $k \in \mathbb{N}$ . For every  $\varphi \in \text{MSO}$ , there is a CVPA  $\mathcal{C}$  such that

$$\mathcal{L}(\mathcal{C}) = \mathcal{L}(\varphi) \cap \text{Tr}_k$$

## Proof

- By induction.
- Lemma: If  $\mathcal{L} \subseteq \text{Tr}_k$  is recognizable, then so is  $\overline{\mathcal{L}} \cap \text{Tr}_k$ .
- $\text{Tr}_k$  is recognizable.
- Atomic formulas standard.
- CVPA are closed under union, intersection, and projection.

# MSO characterization of CVPA wrt. $k$ -phase traces

## Theorem

For every CVPA  $\mathcal{C}$ , there is  $\varphi \in \text{MSO}$  such that  $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{C})$ .

## Theorem

Let  $k \in \mathbb{N}$ . For every  $\varphi \in \text{MSO}$ , there is a CVPA  $\mathcal{C}$  such that

$$\mathcal{L}(\mathcal{C}) = \mathcal{L}(\varphi) \cap \text{Tr}_k$$

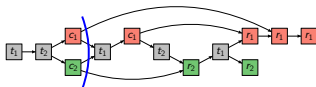
## Proof

- By induction.
- Lemma: If  $\mathcal{L} \subseteq \text{Tr}_k$  is recognizable, then so is  $\overline{\mathcal{L}} \cap \text{Tr}_k$ .
- $\text{Tr}_k$  is recognizable.
- Atomic formulas standard.
- CVPA are closed under union, intersection, and projection.

# Future work: Temporal logic for nested traces

Combine works on

- temporal logic for nested words [AAB<sup>+</sup>'08]
- temporal logic for traces
  - ▶ global [DG'02], interpreted over configurations:



- ▶ local [GK'07], interpreted over events:



---

[AAB<sup>+</sup>'08] Alur & Arenas & Barcelo & Etessami & Immerman & Libkin.

First-Order and Temporal Logics for Nested Words. 2008.

[DG'02] Diekert & Gastin. LTL is expressively complete for Mazurkiewicz traces. 2002.

[GK'07] Gastin & Kuske. Uniform satisfiability in PSPACE for local temporal logics over Mazurkiewicz traces. 2007.

# Future work: Nested MSCs

Extend Zielonka-like theorems and logical characterizations of

- unbounded message-passing automata [BL'06]
- existentially bounded message passing automata [GKM'06]
- universally bounded message-passing automata [HMN<sup>+</sup>'05]

by visibly pushdown stacks.

---

[AAB<sup>+</sup>'08] Alur & Arenas & Barcelo & Etesami & Immerman & Libkin.

First-Order and Temporal Logics for Nested Words. 2008.

[BL'06] B. & Leucker. Message-Passing Automata are expressively equivalent to EMSO Logic. 2006.

[GKM'06] Genest & Kuske & Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. 2006.

[HMNST'05] Henriksen & Mukund & Narayan Kumar & Sohoni & Thiagarjan.

A Theory of Regular MSC Languages. 2005.