Candidature au concours CNRS 06/01

Radu IOSIF (VERIMAG)
CNRS / Université Grenoble Alpes

Vérification de programmes

```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```

Vérification de programmes

{x>0}

```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```

Vérification de programmes

{x>0}

```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```

{x=1}

Vérification de programmes

{x>0}

```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```

sûreté

{x=1}

Vérification de programmes

{x>0}

```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```

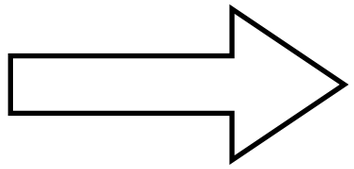
sûreté

terminaison

{x=1}

Vérification de programmes

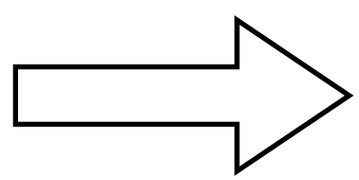
```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```



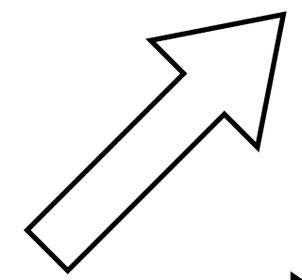
modèle
+
spécification

Vérification de programmes

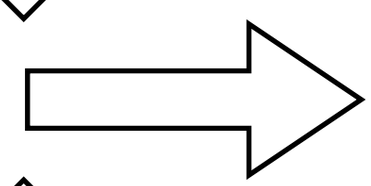
```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```



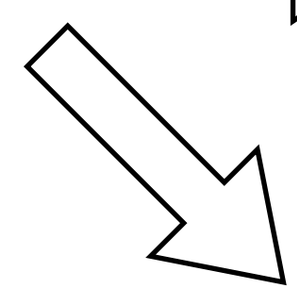
modèle
+
spécification



ok



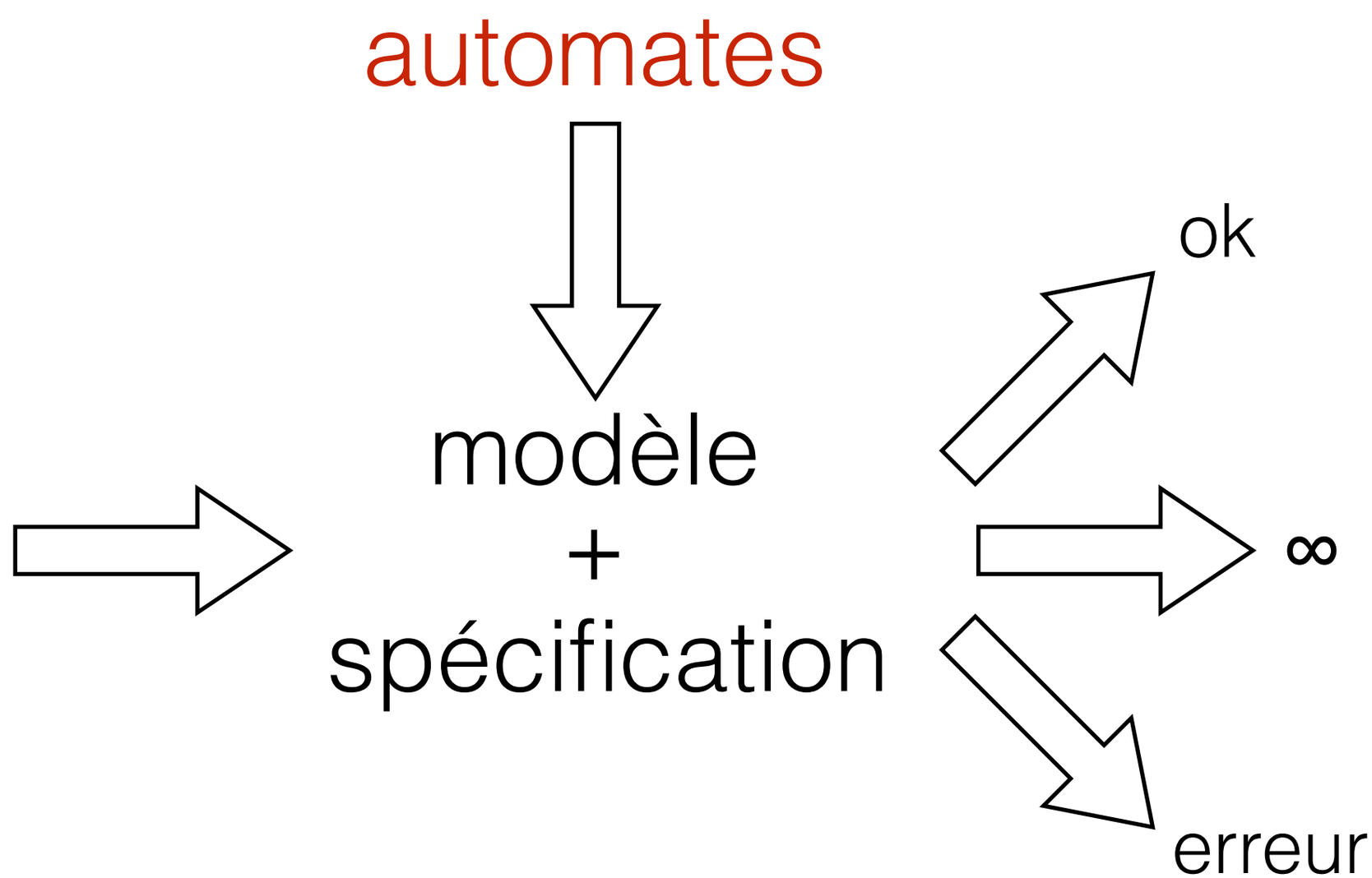
∞



erreur

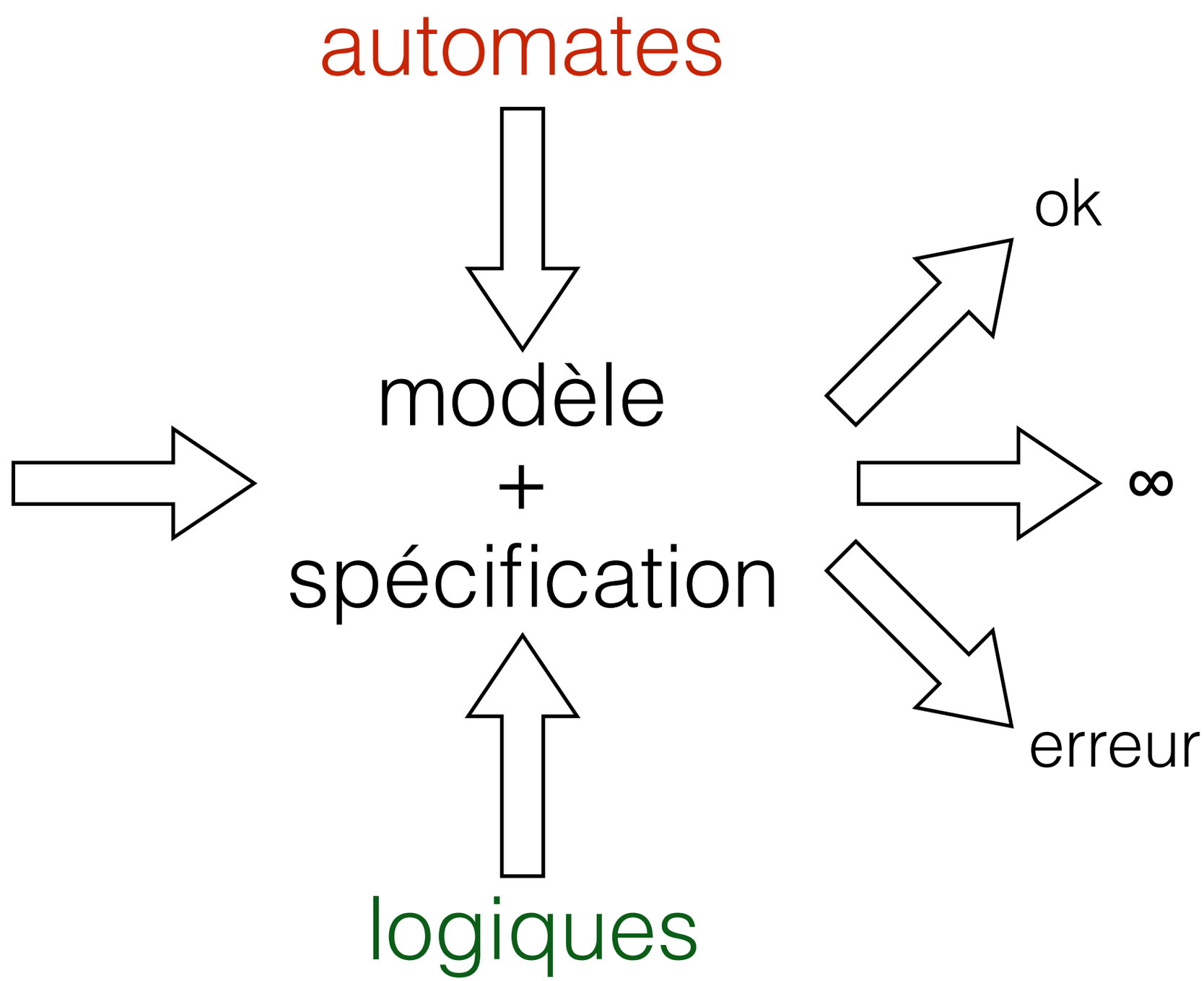
Vérification de programmes

```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```



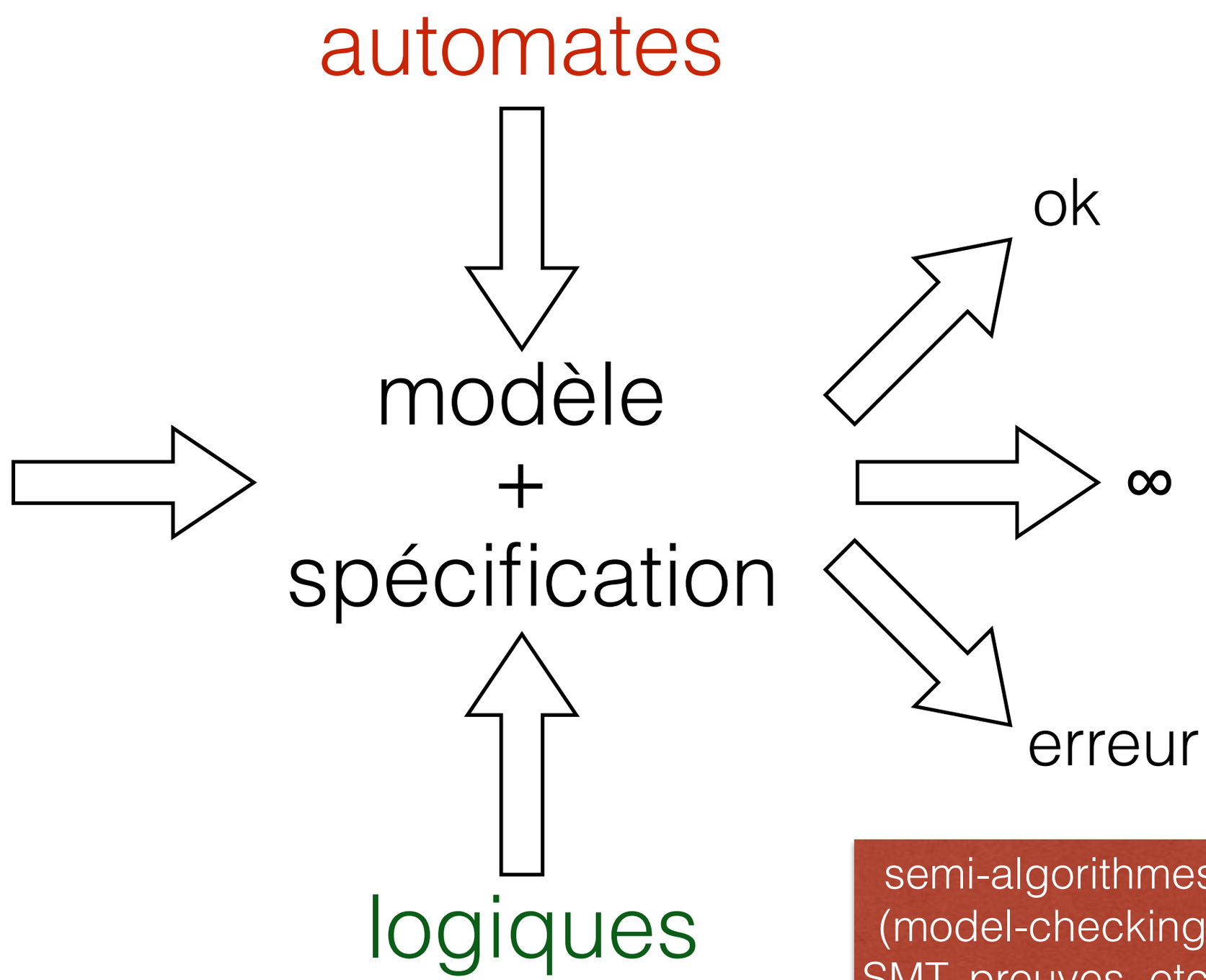
Vérification de programmes

```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```



Vérification de programmes

```
while (x > 1) {
  if (!(x % 2))
    x = 3*x + 1;
  else
    x = x / 2;
}
```



semi-algorithmes (model-checking, SMT, preuves, etc.)

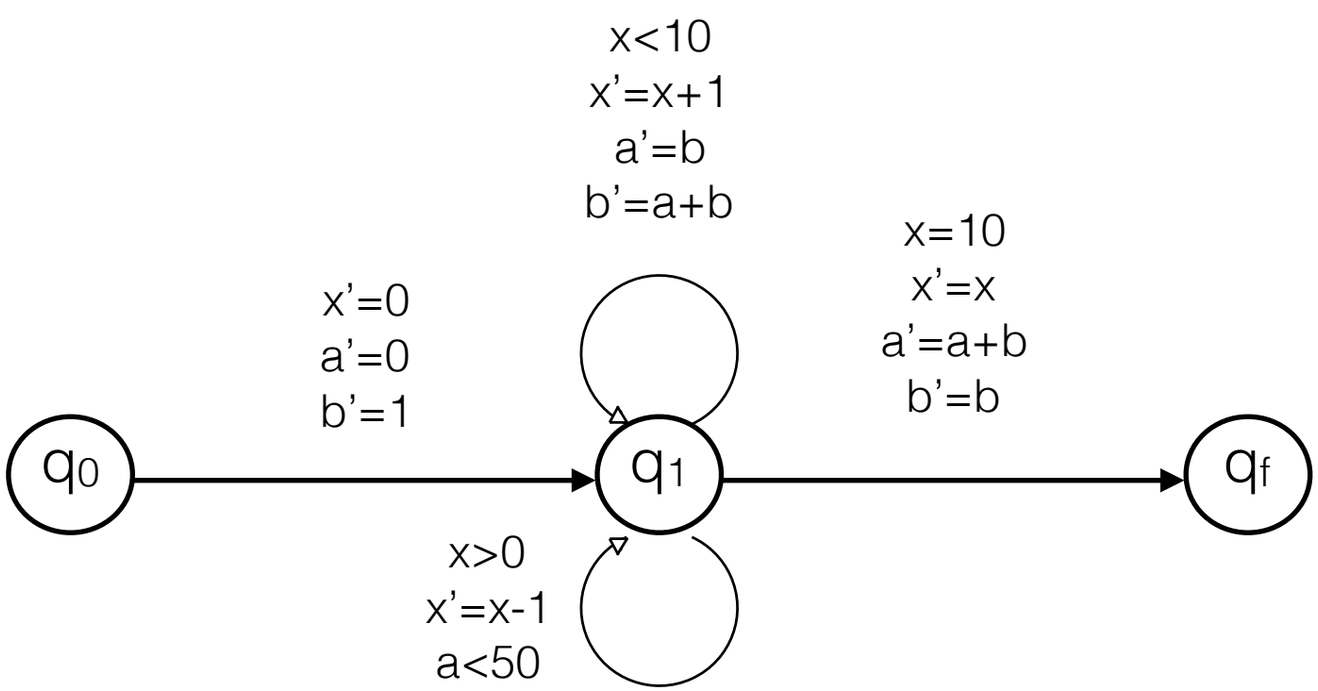
Travaux précédents (2002-2017)

Modèles

Travaux précédents (2002-2017)

Modèles

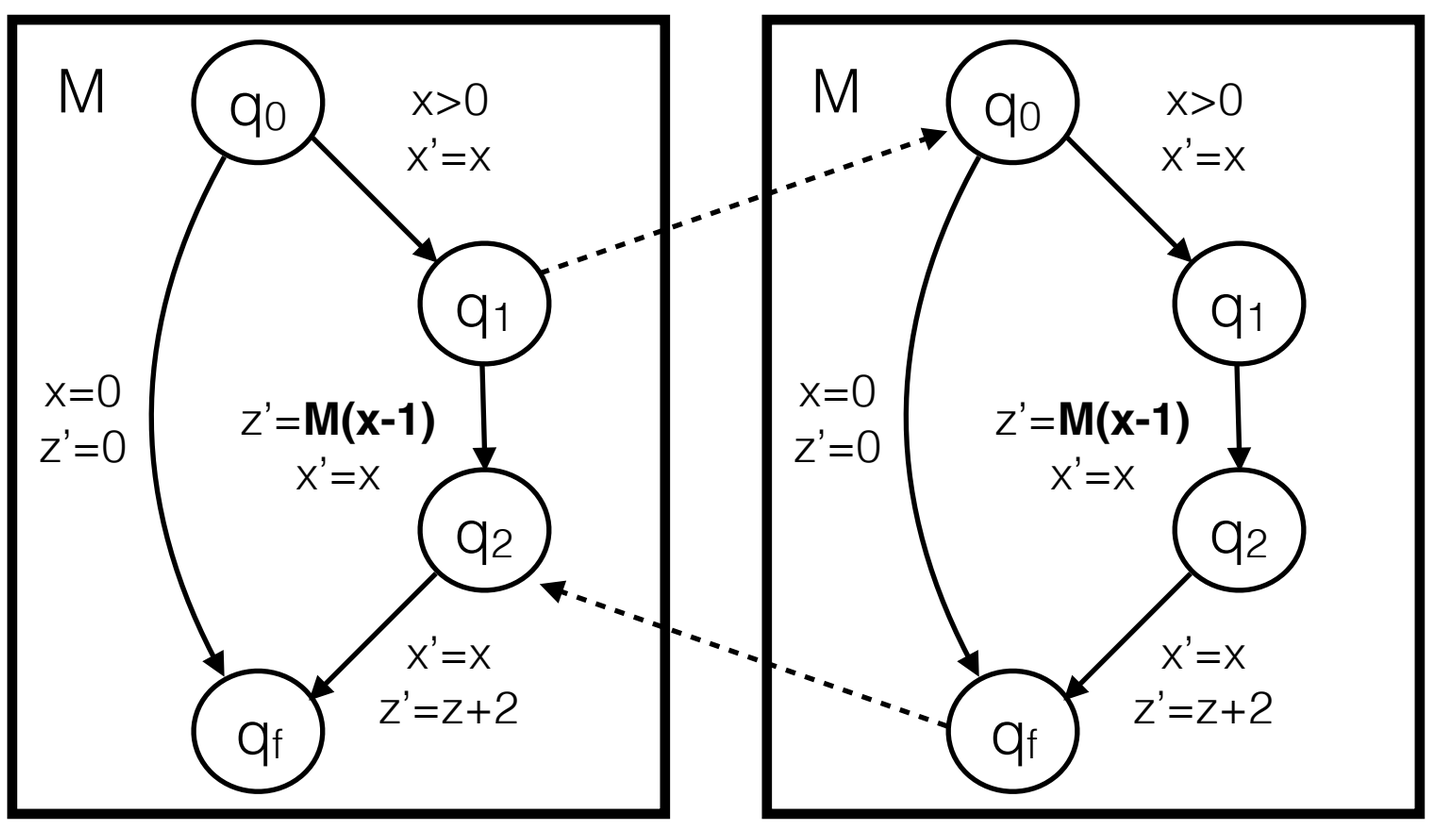
machines à compteurs



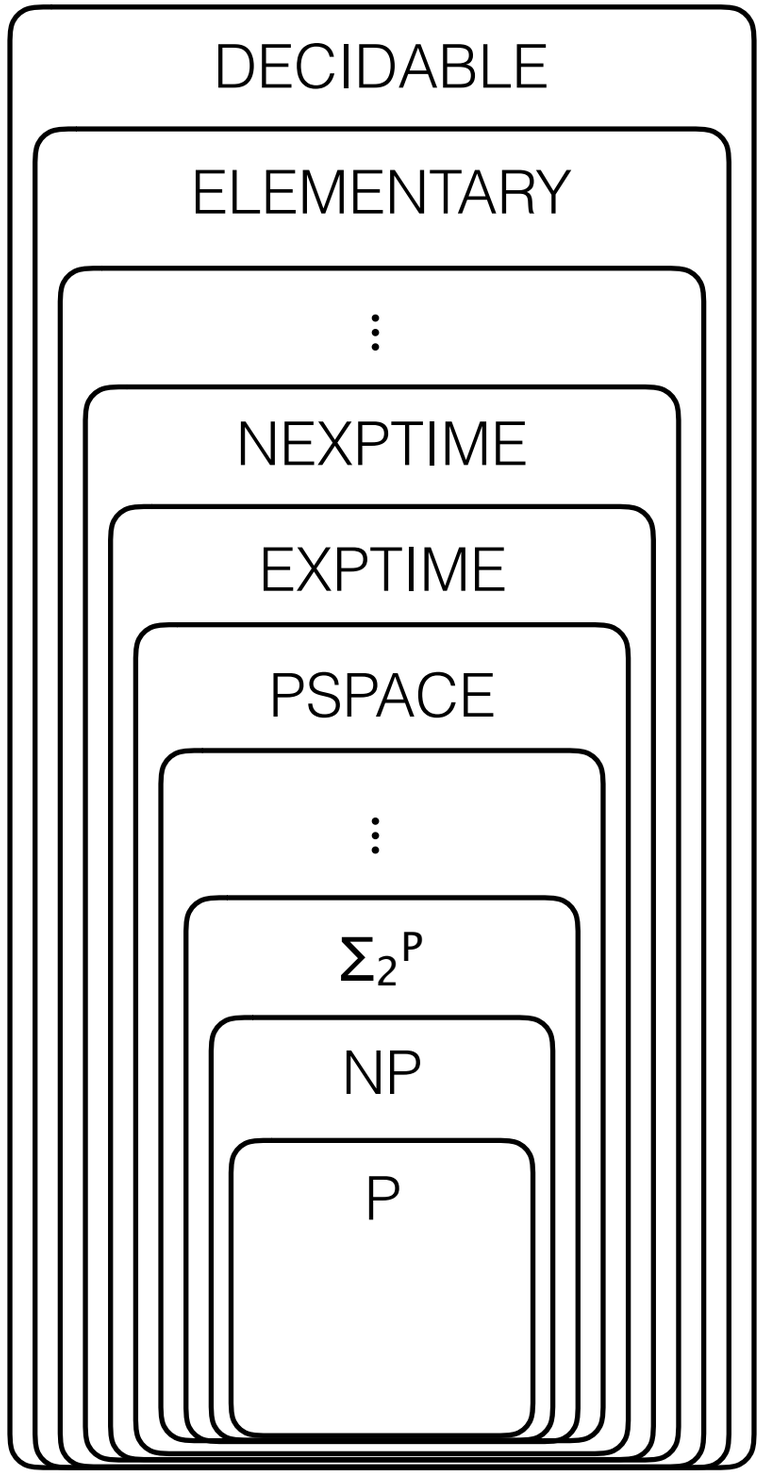
Travaux précédents (2002-2017)

Modèles

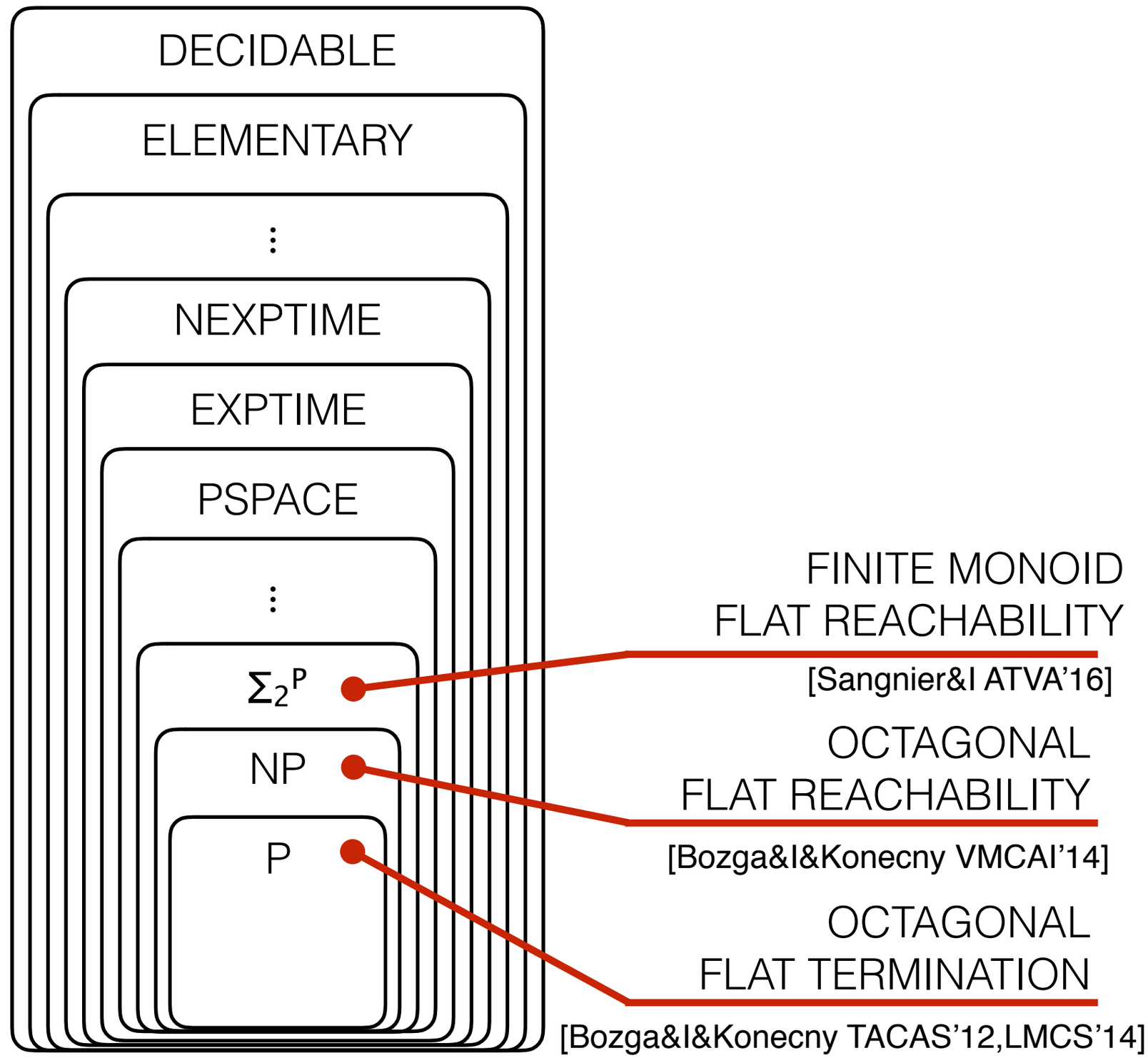
machines à compteurs récursives



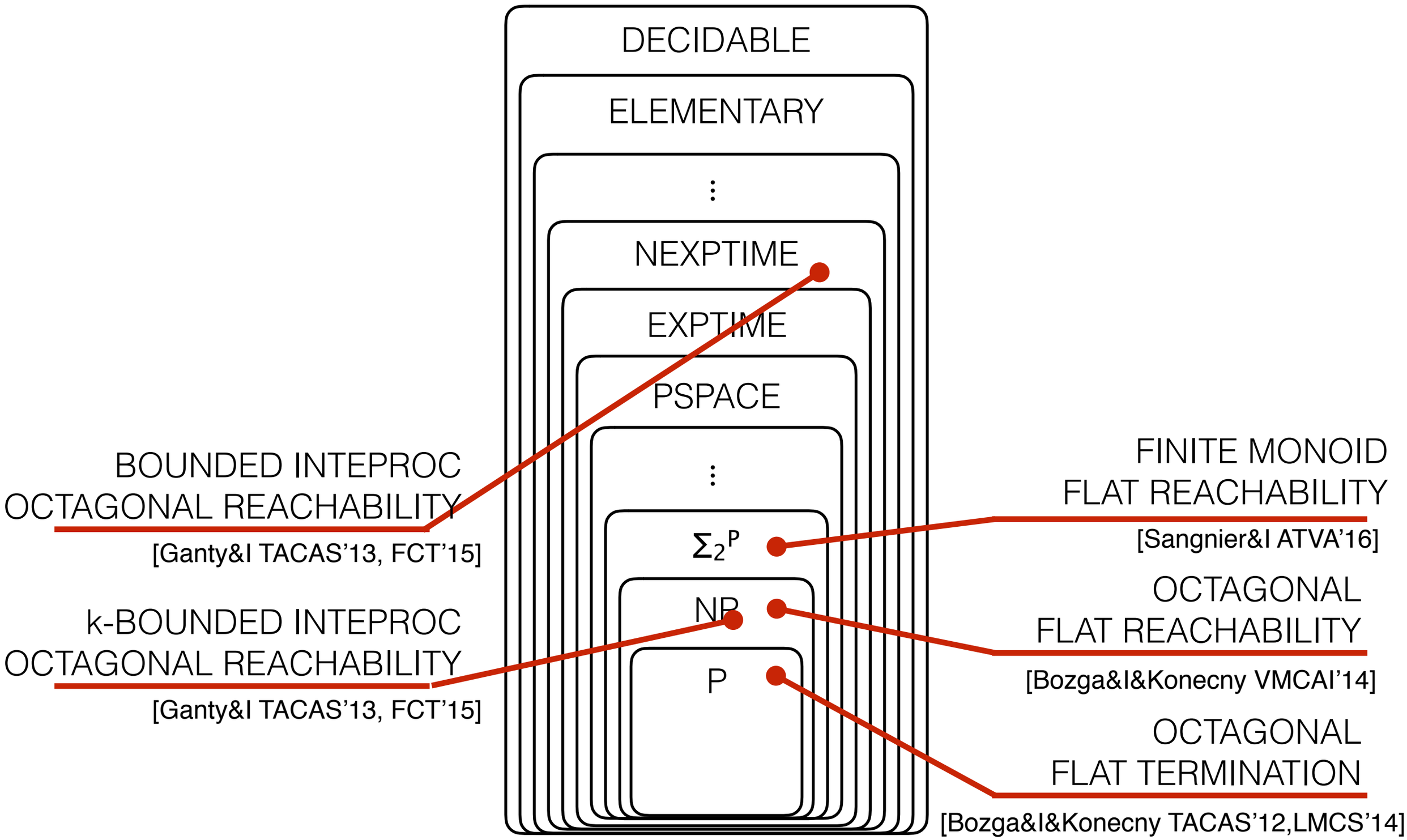
Algorithmes de vérification



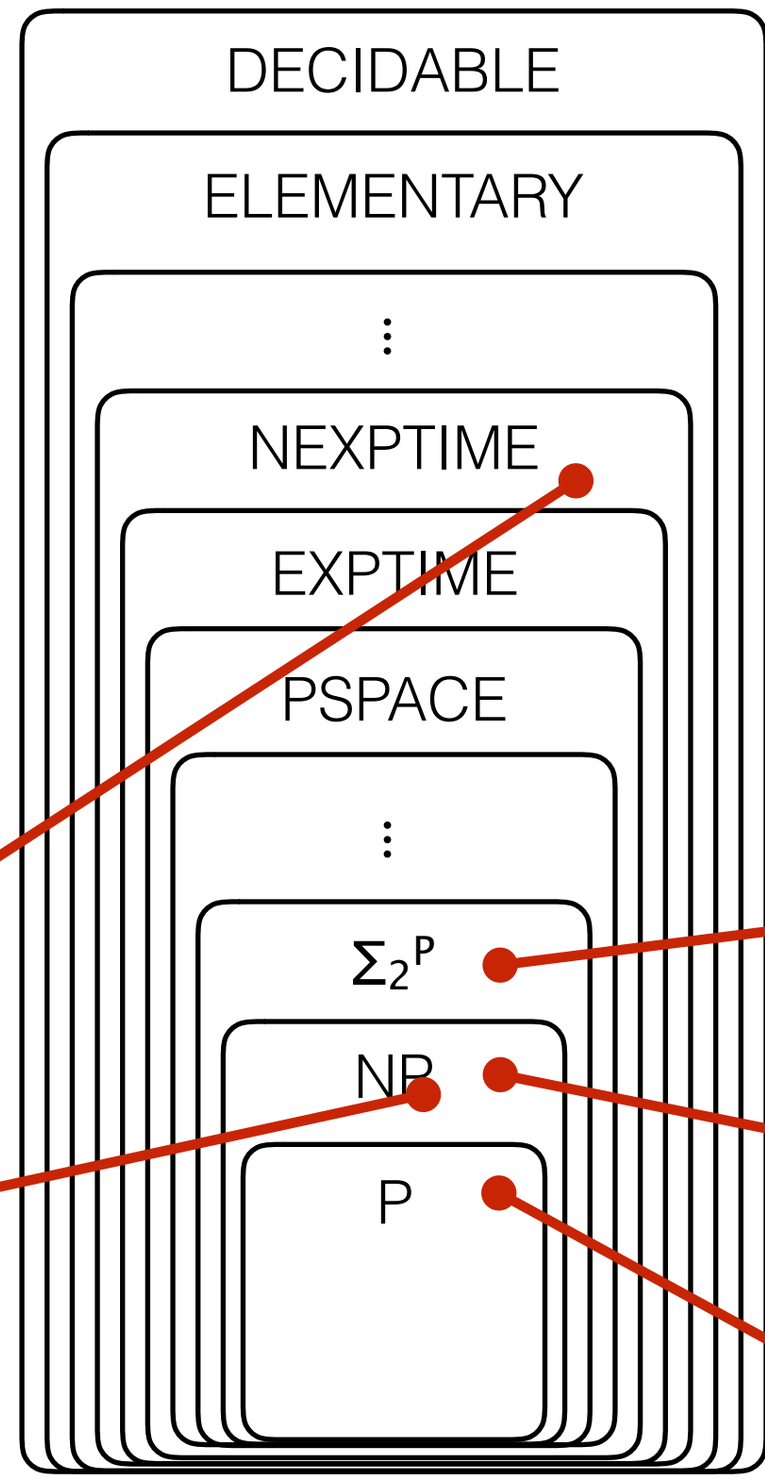
Algorithmes de vérification



Algorithmes de vérification



Algorithmes de vérification



FLATA (F. Konecny, VERIMAG)
https://github.com/filipkonecny/flata

ELDARICA (P. Rümmer, Uppsala)
https://github.com/uuverifiers/eldarica

BOUNDED INTEPROC
OCTAGONAL REACHABILITY
[Ganty&I TACAS'13, FCT'15]

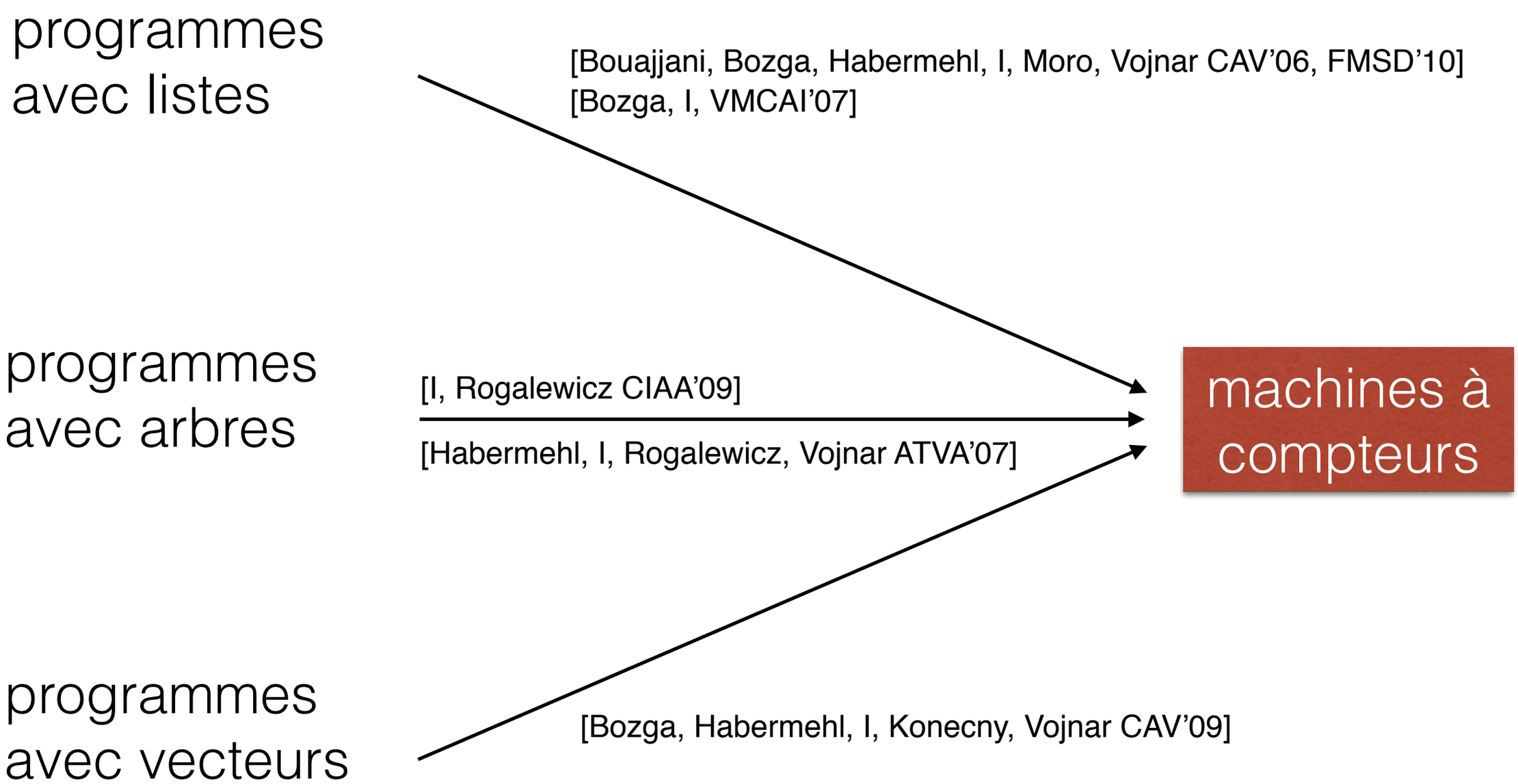
k-BOUNDED INTEPROC
OCTAGONAL REACHABILITY
[Ganty&I TACAS'13, FCT'15]

FINITE MONOID
FLAT REACHABILITY
[Sangnier&I ATVA'16]

OCTAGONAL
FLAT REACHABILITY
[Bozga&I&Konecny VMCAI'14]

OCTAGONAL
FLAT TERMINATION
[Bozga&I&Konecny TACAS'12,LMCS'14]

Travaux précédents (2002-2017)



Travaux précédents (2002-2017)

Logiques pour la spécification de propriétés

Logiques de pointeurs

[Bozga, I, Lakhnech, SAS'04]
[Bozga, I, FOSSACS'05]

Logiques de vecteurs d'entiers

[Habermehl, I, Vojnar, FOSSACS'08, LPAR'08]
[Bozga, Habermehl, I, Konecny, Vojnar, CAV'09]

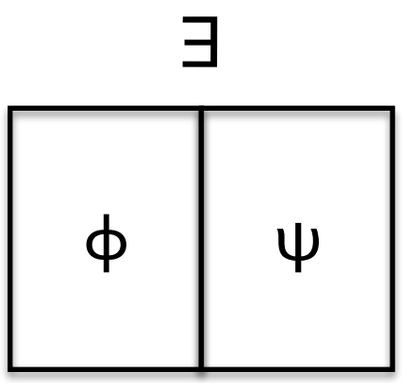
Logique de séparation

[Bozga, I, Perarnau, IJCAR'08, JAR'10]
[I, Rogalewicz, Simacek, CADE'13]
[I, Rogalewicz, Vojnar, ATVA'14]
[I, Reynolds, Serban, King, ATVA'16]
[I, Reynolds, Serban, VMCAI'17]

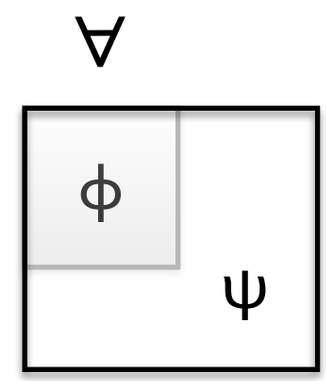
Logique de séparation

Logique de premier ordre pour décrire la mémoire [Ishtiaq,O'Hearn'99] [Reynolds'99]

- opérateurs adjoints booléens ($\wedge$,$\rightarrow$) et **spatiaux** ($*$,$-*$)



$\phi * \psi$

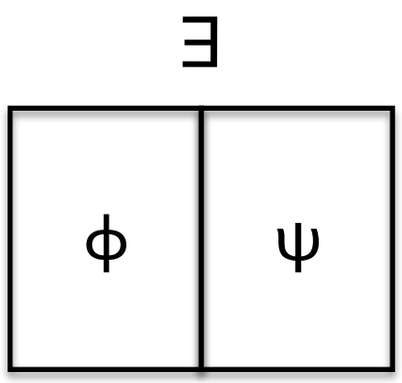


$\phi -* \psi$

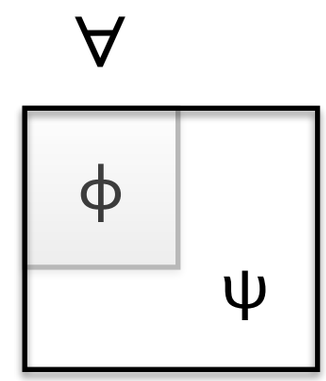
Logique de séparation

Logique de premier ordre pour décrire la mémoire [Ishtiaq,O'Hearn'99] [Reynolds'99]

- opérateurs adjoints booléens ($\wedge$,$\rightarrow$) et **spatiaux** ($*$,$-*$)



$\phi * \psi$



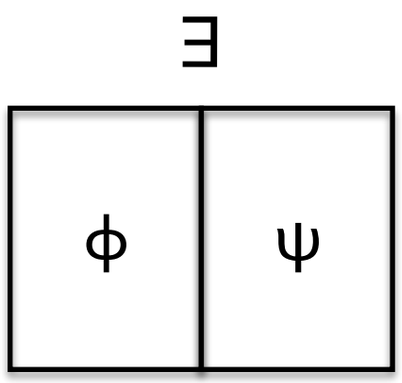
$\phi -* \psi$

Règle de **preuve compositionelle** dans la logique de Hoare (frame rule)

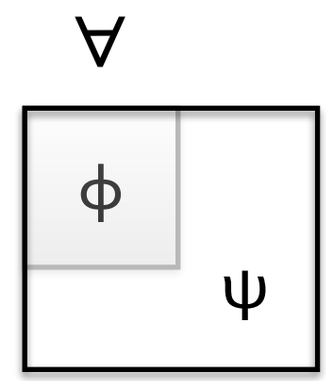
Logique de séparation

Logique de premier ordre pour décrire la mémoire [Ishtiaq,O'Hearn'99] [Reynolds'99]

- opérateurs adjoints booléens ($\wedge$,$\rightarrow$) et **spatiaux** ($*$,$-*$)



$\phi * \psi$

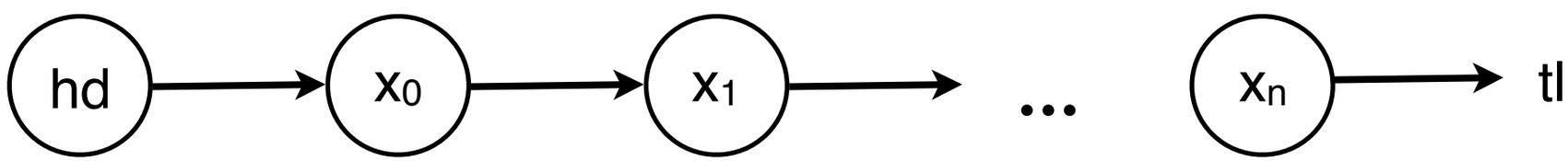


$\phi -* \psi$

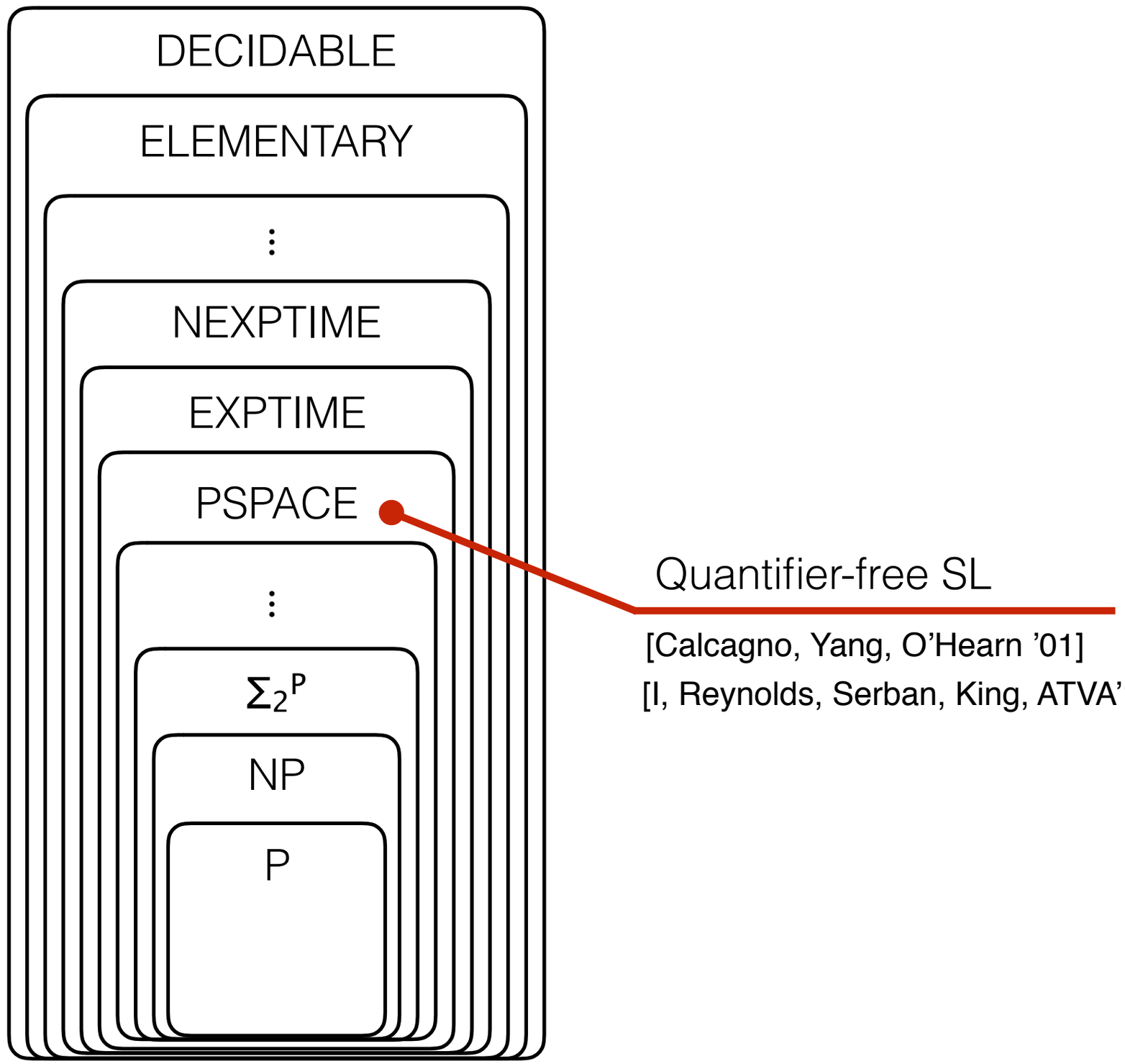
Règle de **preuve compositionelle** dans la logique de Hoare (frame rule)

Définitions inductives pour décrire des structures de données récursives

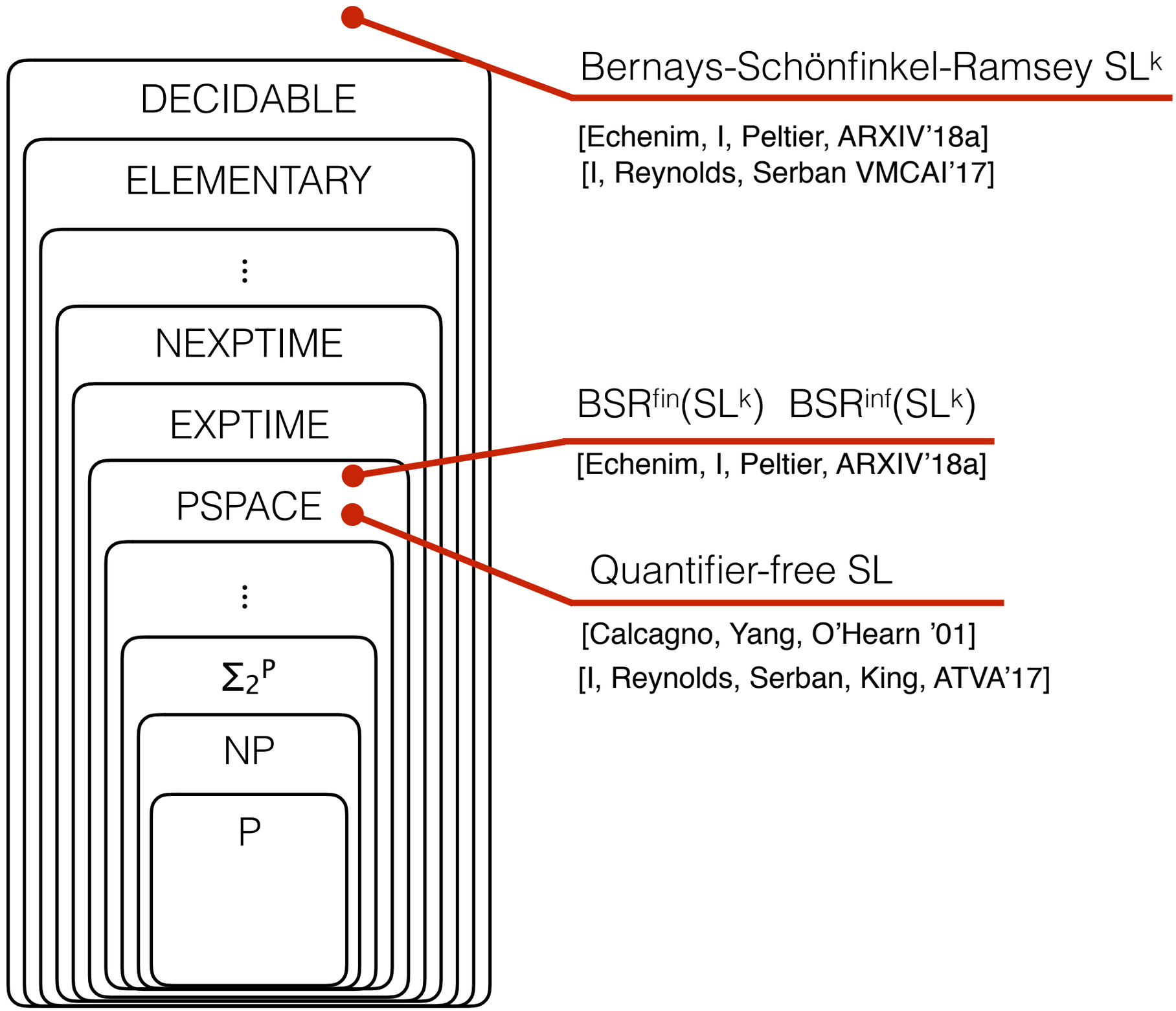
$$\mathrm{lseg}(hd,tl) \equiv emp \wedge hd=tl \vee \exists x \,.\, hd \rightarrow x * \mathrm{lseg}(x,tl)$$



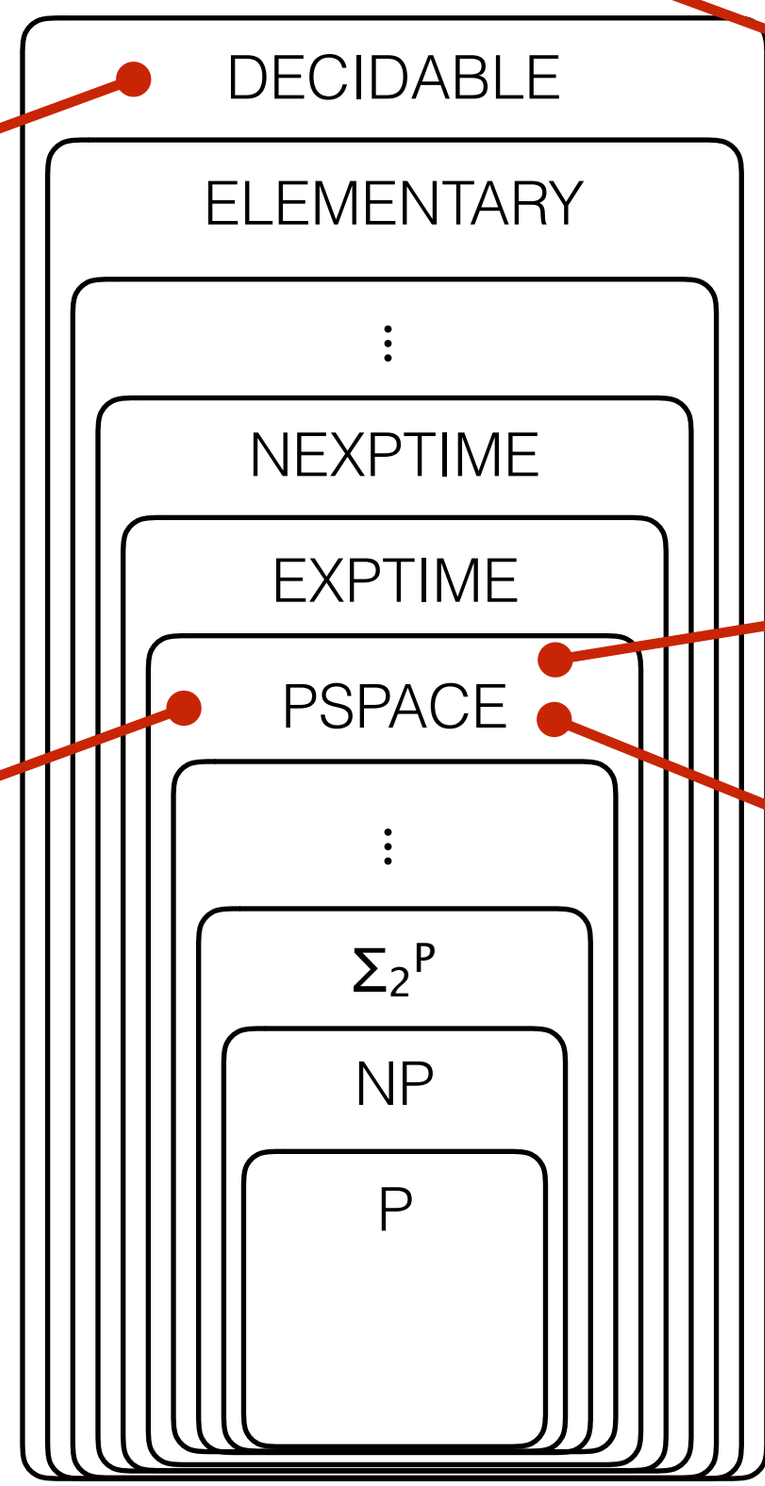
Algorithmes (premier ordre)



Algorithmes (premier ordre)



Algorithmes (premier ordre)



Prenex SL^1

[Echenim, I, Peltier, ARXIV'18b]

Bernays-Schönfinkel-Ramsey SL^k

[Echenim, I, Peltier, ARXIV'18a]
[I, Reynolds, Serban VMCAI'17]

$BSR^{fin}(SL^k)$ $BSR^{inf}(SL^k)$

[Echenim, I, Peltier, ARXIV'18a]

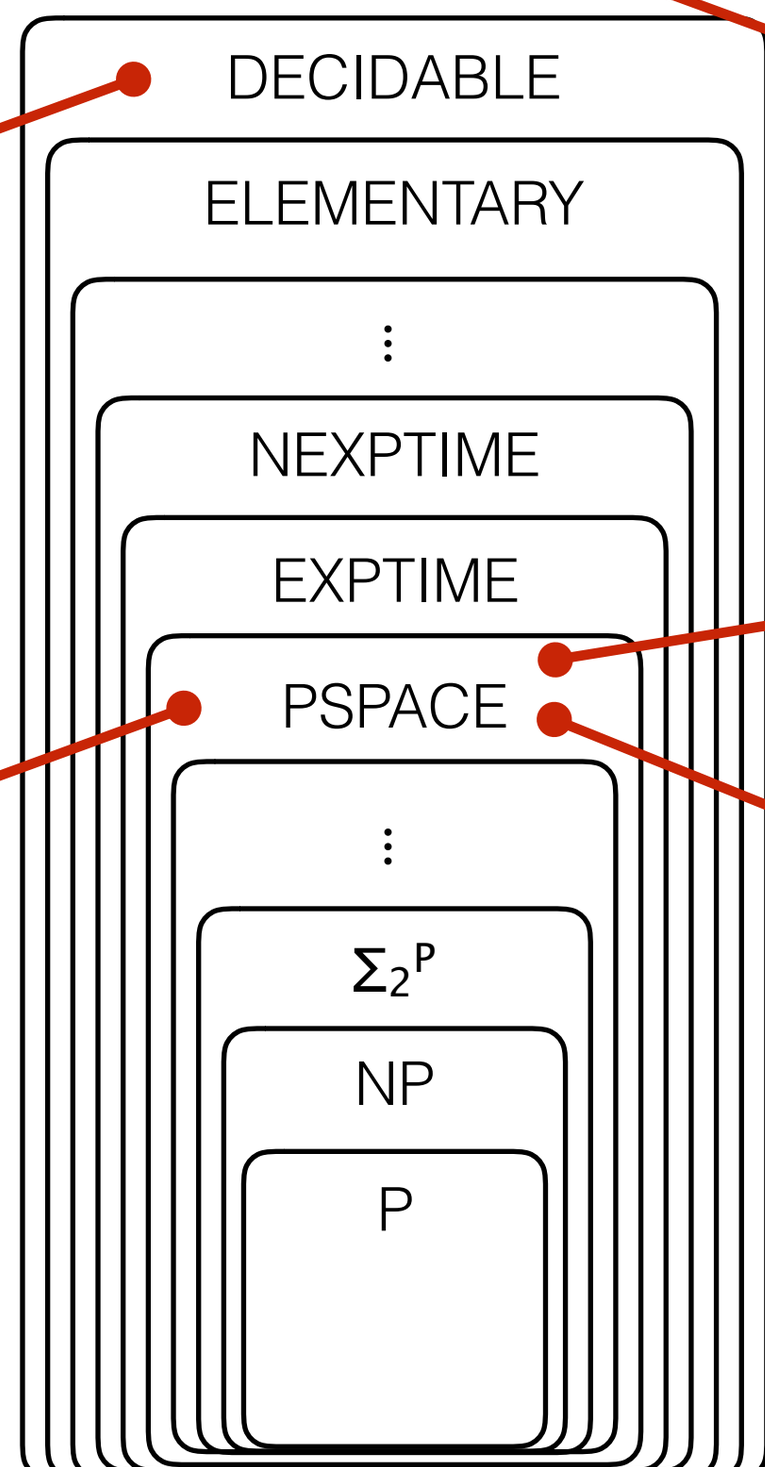
$BSR(SL^1)$

[Echenim, I, Peltier, ARXIV'18b]

Quantifier-free SL

[Calcagno, Yang, O'Hearn '01]
[I, Reynolds, Serban, King, ATVA'17]

Algorithmes (premier ordre)



Prenex SL^1

[Echenim, I, Peltier, ARXIV'18b]

Bernays-Schönfinkel-Ramsey SL^k

[Echenim, I, Peltier, ARXIV'18a]
[I, Reynolds, Serban VMCAI'17]

$BSR^{fin}(SL^k)$ $BSR^{inf}(SL^k)$

[Echenim, I, Peltier, ARXIV'18a]

$BSR(SL^1)$

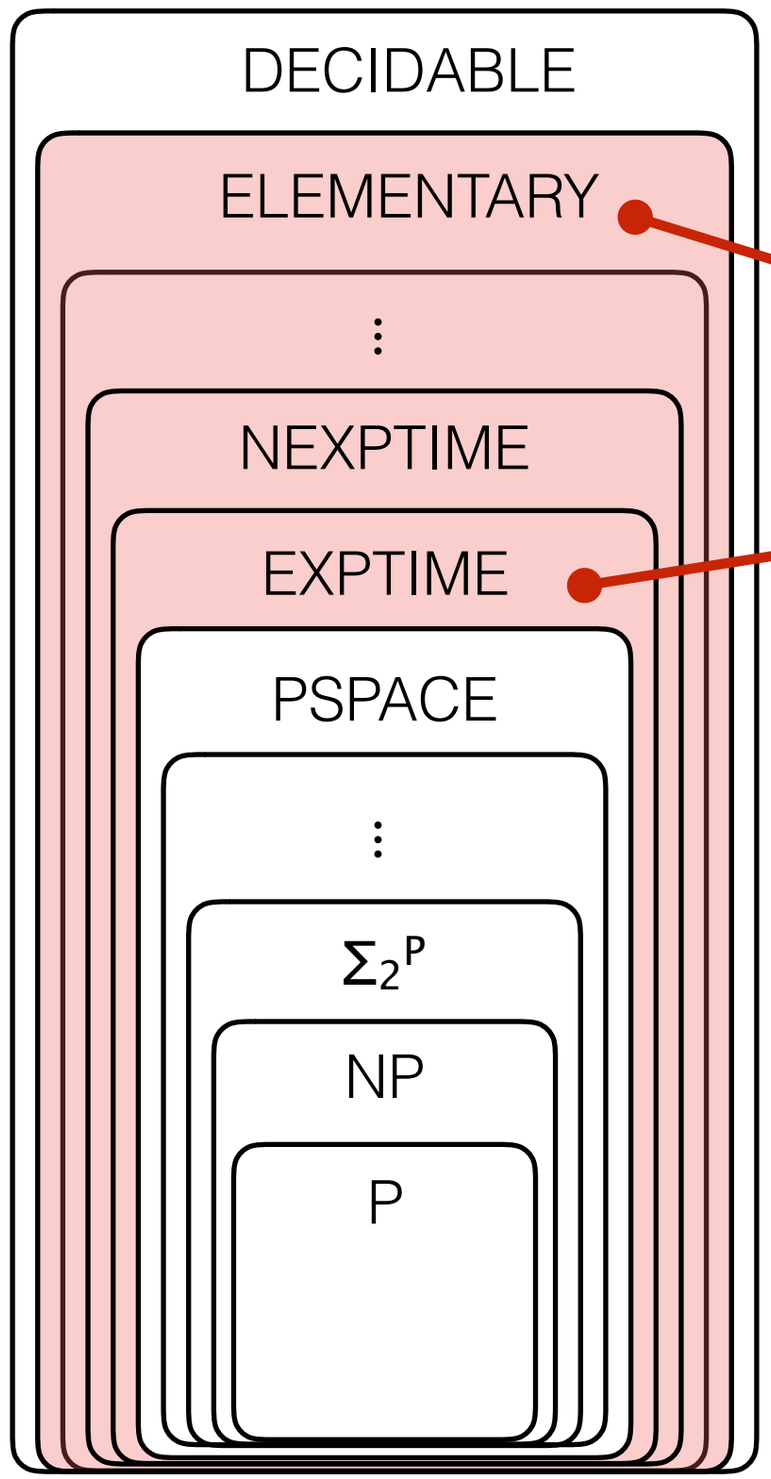
[Echenim, I, Peltier, ARXIV'18b]

Quantifier-free SL

[Calcagno, Yang, O'Hearn '01]
[I, Reynolds, Serban, King, ATVA'17]

CVC4 (A. Reynolds, Iowa)
https://github.com/CVC4

Algorithmes (ordre supérieur)



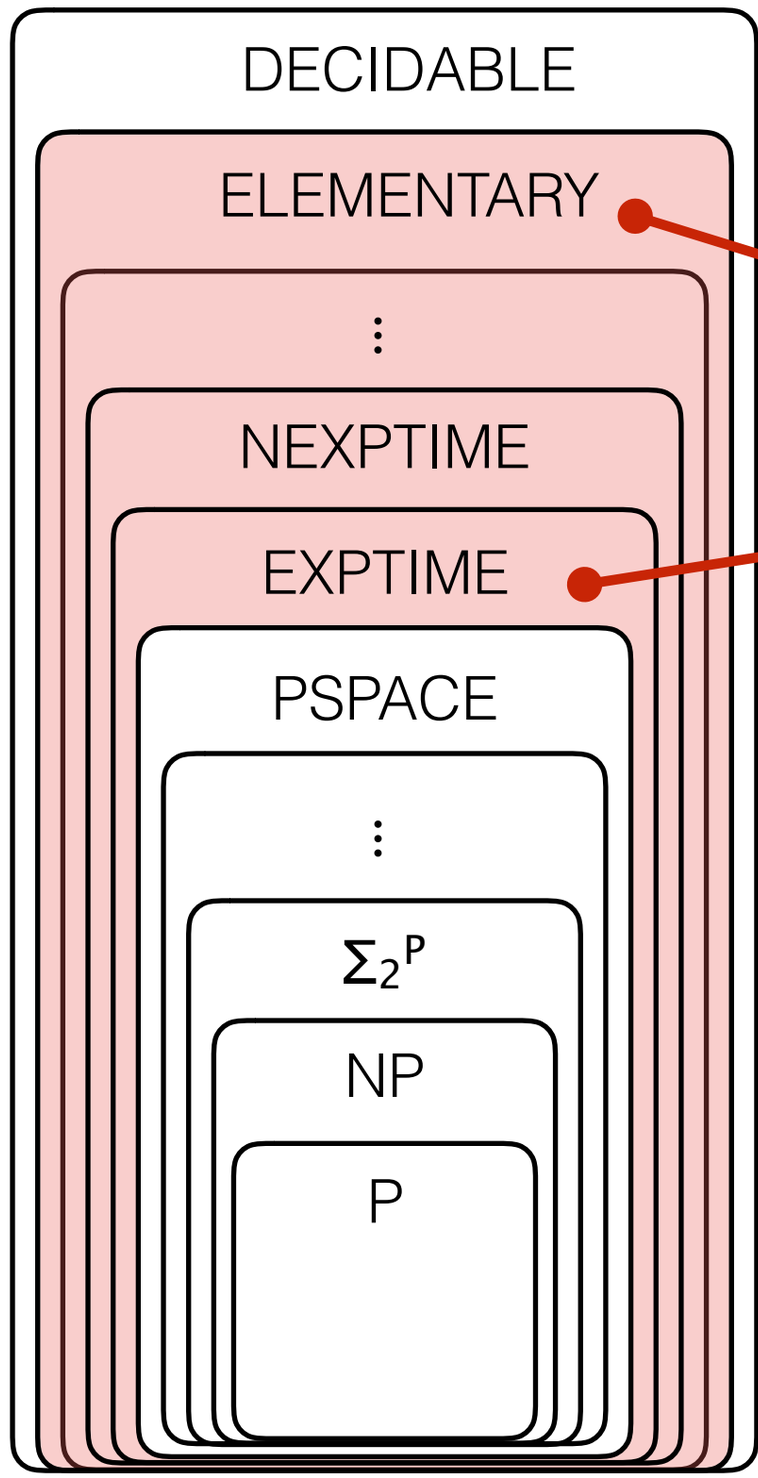
ENTAILMENT SL_{BTW}

[I, Rogalewicz, Simacek' CADE13]

ENTAILMENT SL_{LOC}

[I, Rogalewicz, Vojnar ATVA'14]

Algorithmes (ordre supérieur)



ENTAILMENT SL_{BTW}

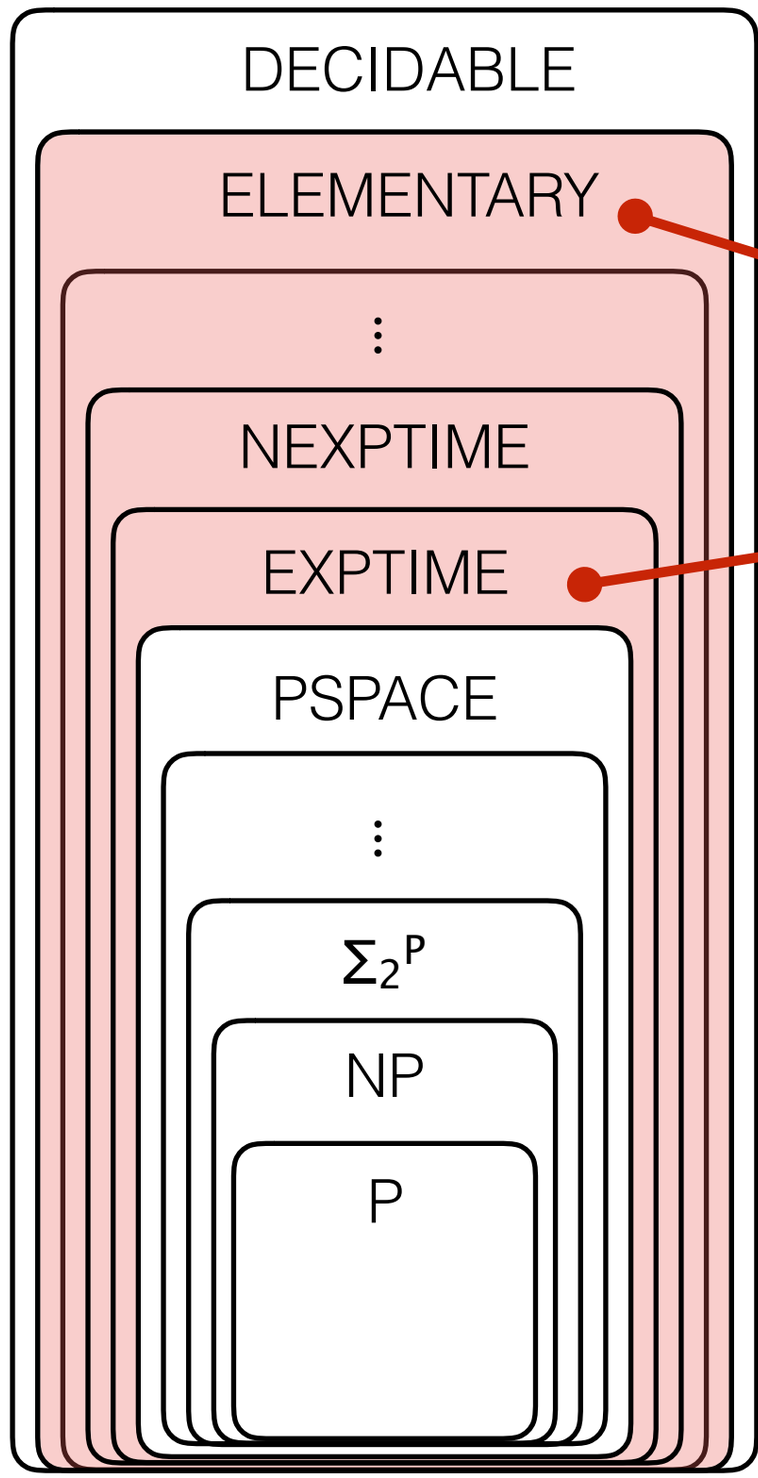
[I, Rogalewicz, Simacek' CADE13]

ENTAILMENT SL_{LOC}

[I, Rogalewicz, Vojnar ATVA'14]

SLIDE (A. Rogalewicz, Brno)
http://www.fit.vutbr.cz/research/groups/verifit/tools/slide/

Algorithmes (ordre supérieur)



ENTAILMENT SL_{BTW}

[I, Rogalewicz, Simacek' CADE13]

ENTAILMENT SL_{LOC}

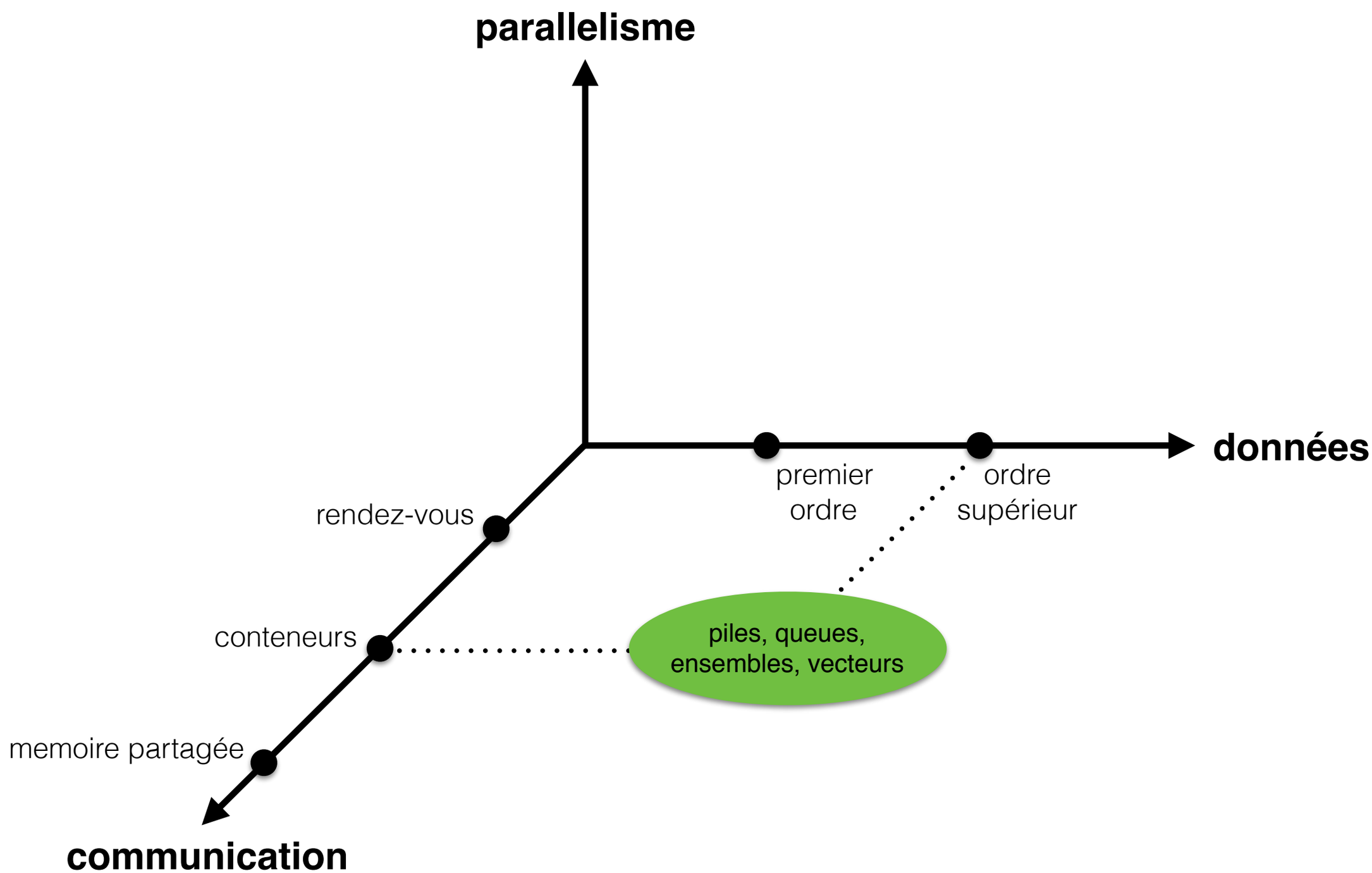
[I, Rogalewicz, Vojnar ATVA'14]

SLIDE (A. Rogalewicz, Brno)
http://www.fit.vutbr.cz/research/groups/verifit/tools/slide/

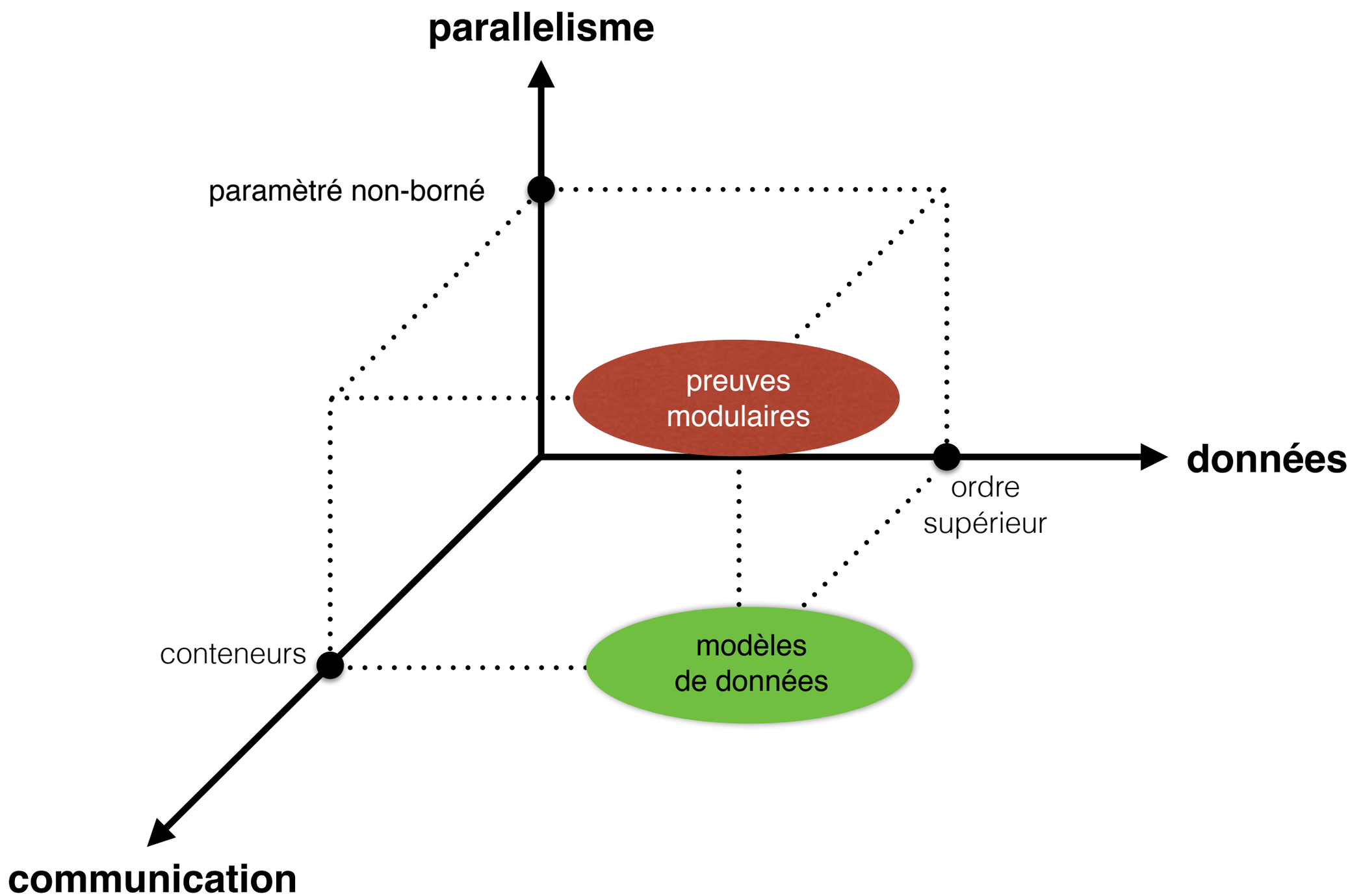
INDUCTOR (C. Serban, VERIMAG)
https://github.com/cristina-serban/inductor

Travaux en cours et futurs

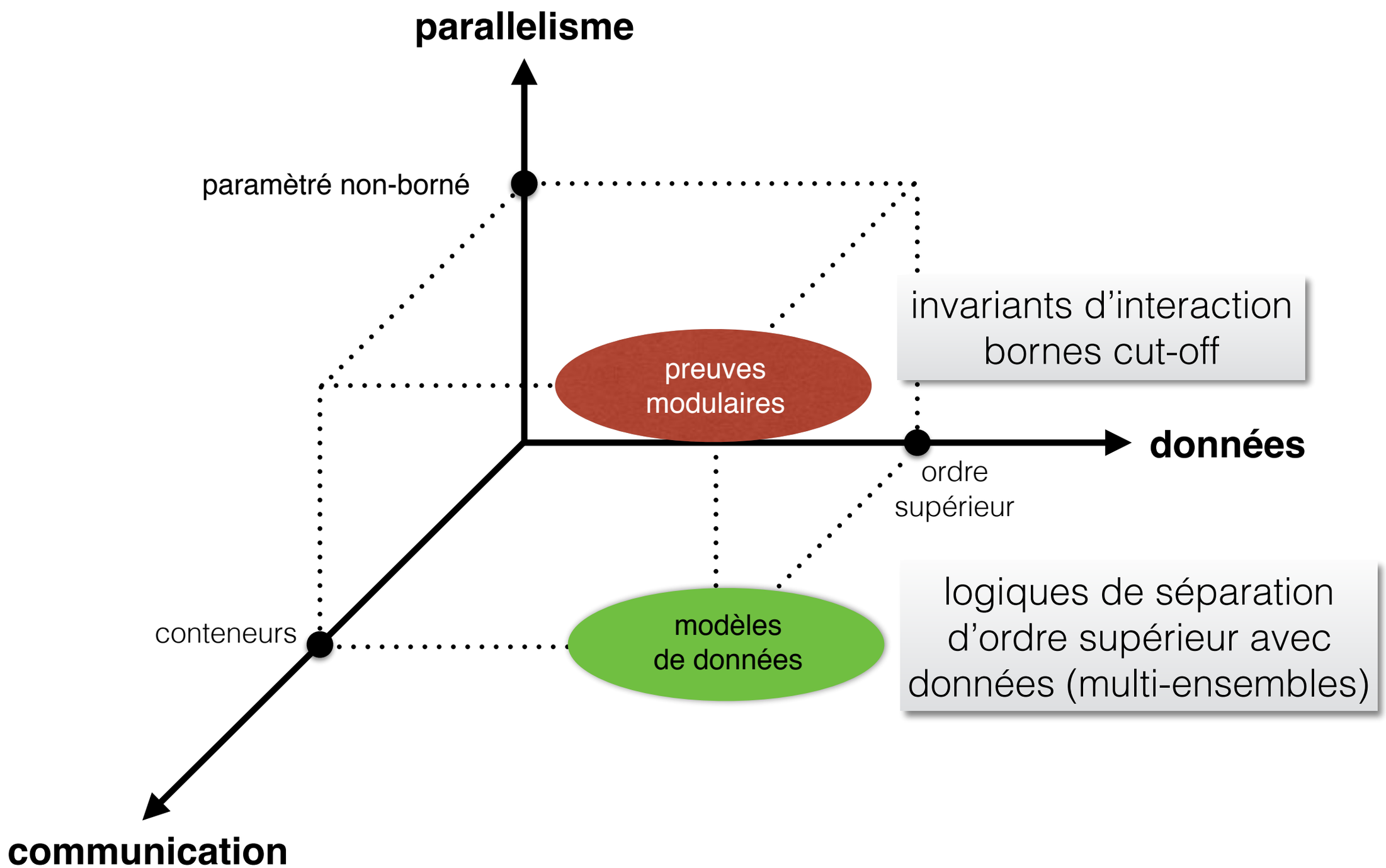
Parallelisme et communication



Preuves modulaires de programmes concurrents

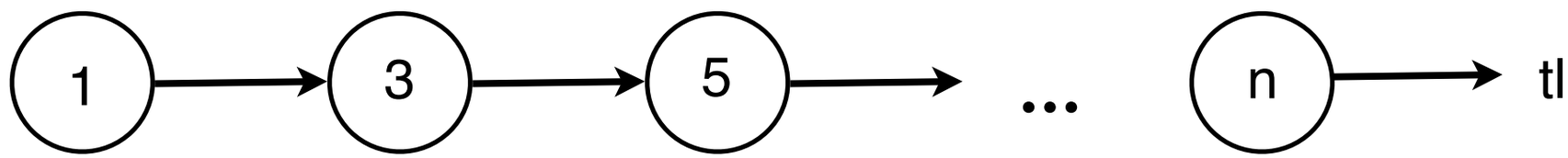


Preuves modulaires de programmes concurrents



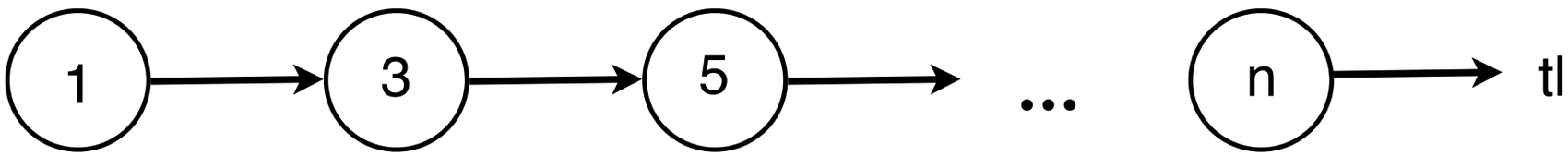
Logiques de séparation avec données

$\mathsf{lseg}(\mathsf{hd},d,\mathsf{tl}) \equiv \mathsf{emp} \wedge \mathsf{hd}=\mathsf{tl} \vee \exists x\ \exists e\ .\ \mathsf{hd} \rightarrow (x,d) * \mathsf{lseg}(x,e,\mathsf{tl}) \wedge d \leq e$



Logiques de séparation avec données

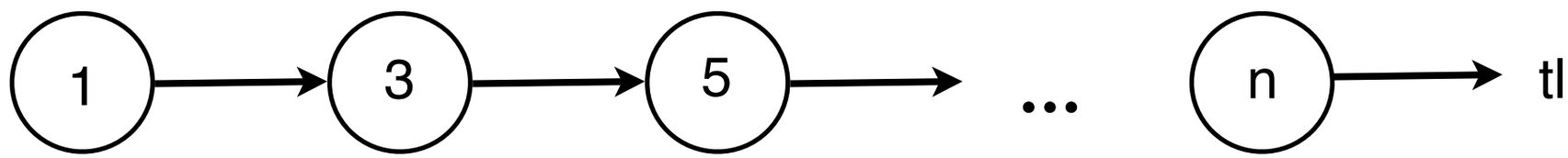
$\mathsf{lseg}(hd,d,tl) \equiv emp \wedge hd=tl \vee \exists x\ \exists e\ .\ hd \rightarrow (x,d) * \mathsf{lseg}(x,e,tl) \wedge d \leq e$



preuves pour
la validité
des implications

Logiques de séparation avec données

$\mathsf{lseg}(hd,d,tl) \equiv \mathsf{emp} \wedge hd=tl \vee \exists x\ \exists e\ .\ hd \rightarrow (x,d) * \mathsf{lseg}(x,e,tl) \wedge d \leq e$



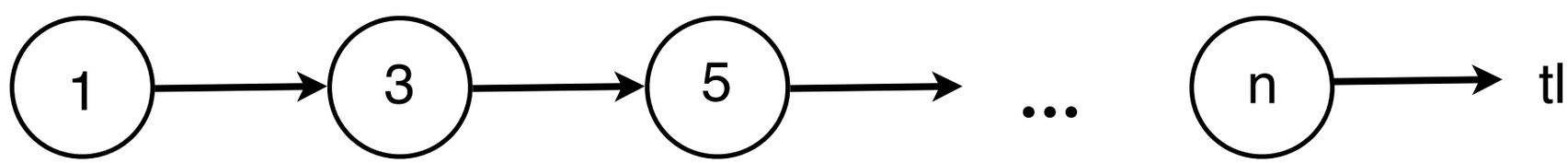
invariants sur
les données

preuves pour
la validité
des implications

règles d'inférence pour
raisonnement inductif

Logiques de séparation avec données

$\mathsf{lseg}(hd,d,tl) \equiv emp \wedge hd=tl \vee \exists x\ \exists e\ .\ hd \rightarrow (x,d) * \mathsf{lseg}(x,e,tl) \wedge d \leq e$



inclusion des langages de données
[I, Rogalewicz, Vojnar, TACAS'16]

automates alternants avec données
[I,Xu, TACAS'18]

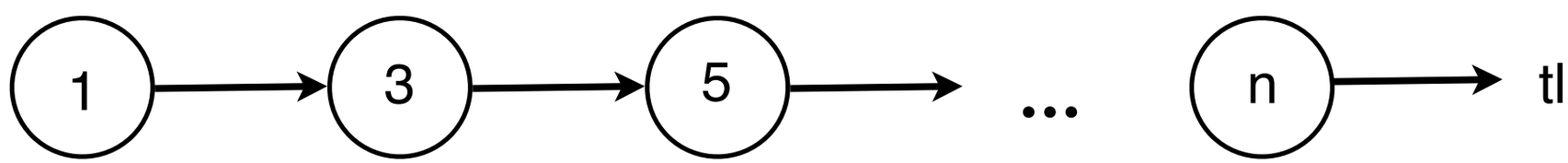
invariants sur
les données

preuves pour
la validité
des implications

règles d'inférence pour
raisonnement inductif

Logiques de séparation avec données

$\mathsf{lseg}(hd,d,tl) \equiv emp \wedge hd=tl \vee \exists x\ \exists e\ .\ hd \rightarrow (x,d) * \mathsf{lseg}(x,e,tl) \wedge d \leq e$



inclusion des langages de données
[I, Rogalewicz, Vojnar, TACAS'16]

automates alternants avec données
[I,Xu, TACAS'18]

invariants sur
les données

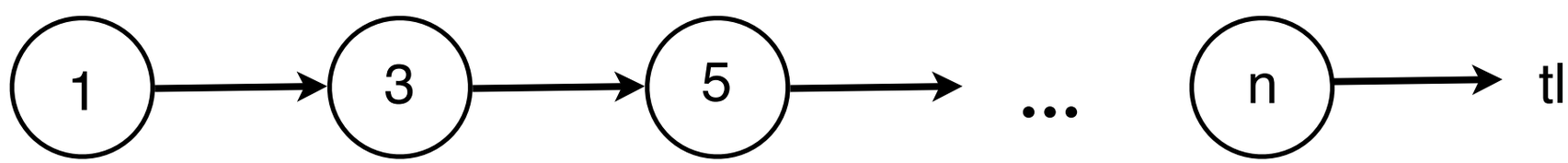
preuves pour
la validité
des implications

règles d'inférence pour
raisonnement inductif

systèmes de preuve par descente infinie
[I,Serban,ARXIV'17]

Logiques de séparation avec données

$\mathsf{lseg}(hd,d,tl) \equiv emp \wedge hd=tl \vee \exists x\ \exists e\ .\ hd \rightarrow (x,d) * \mathsf{lseg}(x,e,tl) \wedge d \leq e$



inclusion des langages de données
[I, Rogalewicz, Vojnar, TACAS'16]

automates alternants avec données
[I,Xu, TACAS'18]

invariants sur
les données

preuves pour
la validité
des implications

règles d'inférence pour
raisonnement inductif

intégration des résultats

systèmes de preuve par descente infinie
[I,Serban,ARXIV'17]

Conteneurs concurrents

Défis relevés

Etat de l’art

Conteneurs concurrents

Défis relevés	**Etat de l'art**
identification automatique des points de linéarisation	annotations fournies par le développeur

Conteneurs concurrents

Défis relevés	**Etat de l'art**
identification automatique des points de linéarisation	annotations fournies par le développeur
implementations utilisant des structures recursive chaînées	implementations utilisant des vecteurs ou des listes simples

Conteneurs concurrents

Défis relevés	**Etat de l'art**
identification automatique des points de linéarisation	annotations fournies par le développeur
implementations utilisant des structures recursive chaînées	implementations utilisant des vecteurs ou des listes simples
spécifications axiomatiques fournies par l'utilisateur	politiques d'accès pré-codées

Conteneurs concurrents

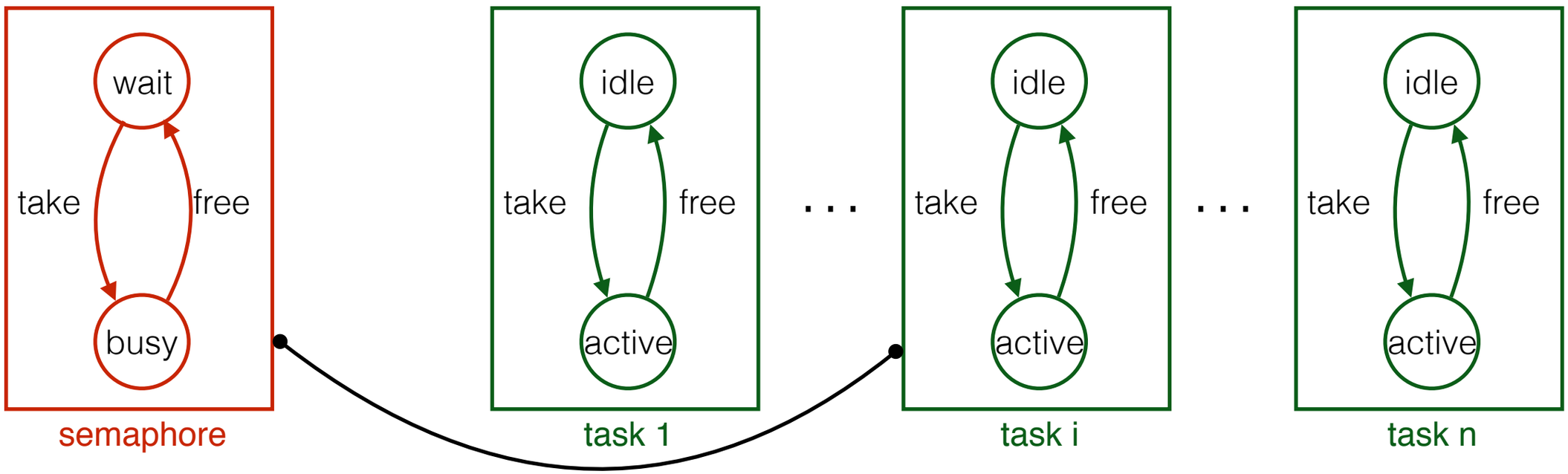
Défis relevés	**Etat de l'art**
identification automatique des points de linéarisation	annotations fournies par le développeur
implementations utilisant des structures recursive chaînées	implementations utilisant des vecteurs ou des listes simples
spécifications axiomatiques fournies par l'utilisateur	politiques d'accès pré-codées

Directions de travail

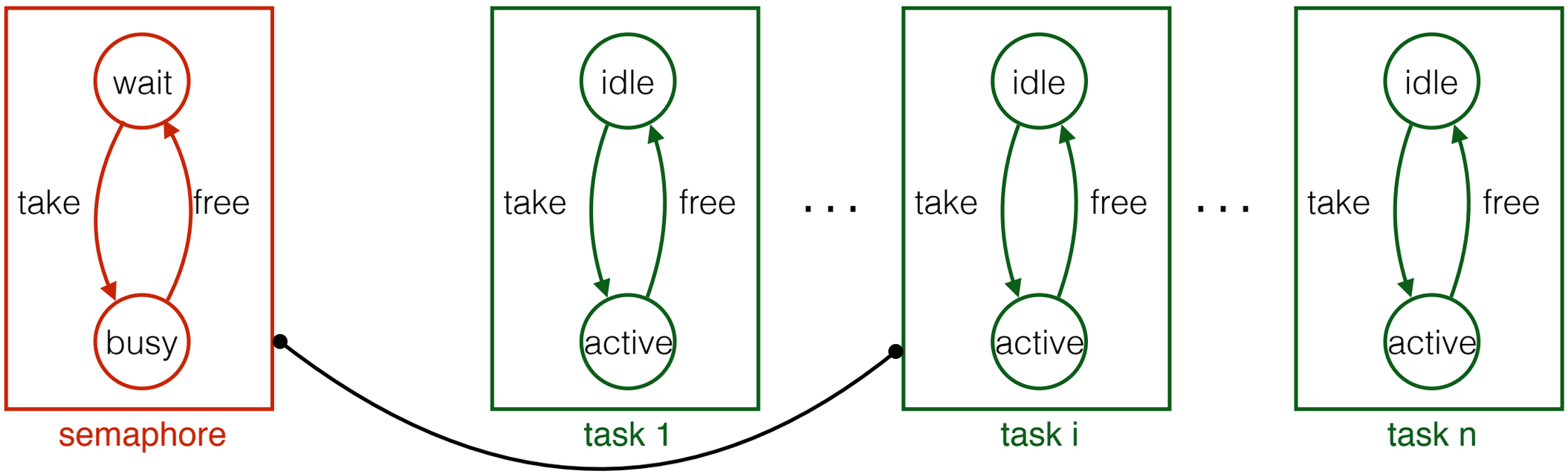
synthèse de relations de raffinement

type de données algébrique ↔ implementation

Systèmes parallels paramètrés



Systèmes parallels paramètrés

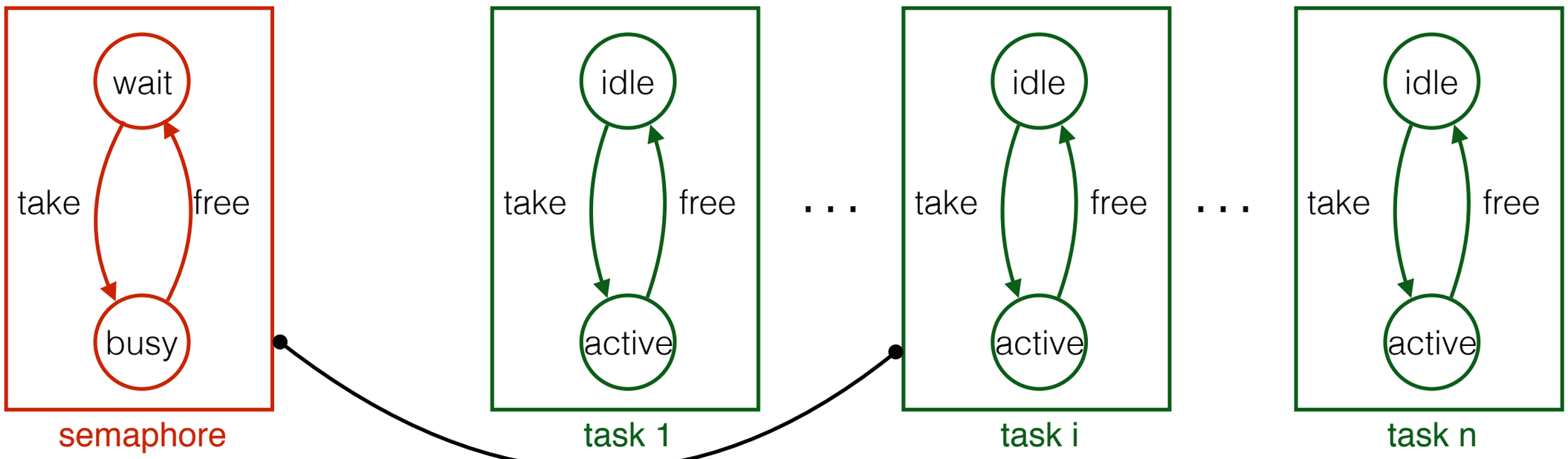


absence de blocages

exclusion mutuelle

⇦ invariants d'interaction

Systèmes parallels paramètrés

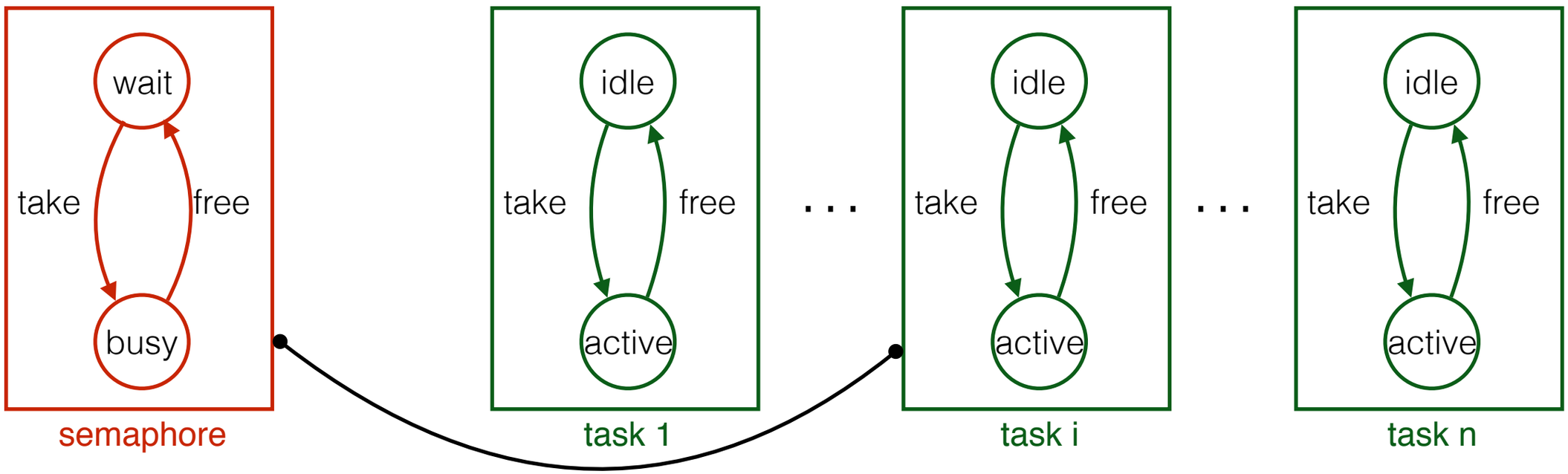


topologies symétriques
[Bozga, I, Sifakis, en cours]

absence de blocages
exclusion mutuelle

⇦ invariants d'interaction

Systèmes parallels paramètrés



absence de blocages

exclusion mutuelle

⇦ invariants d'interaction

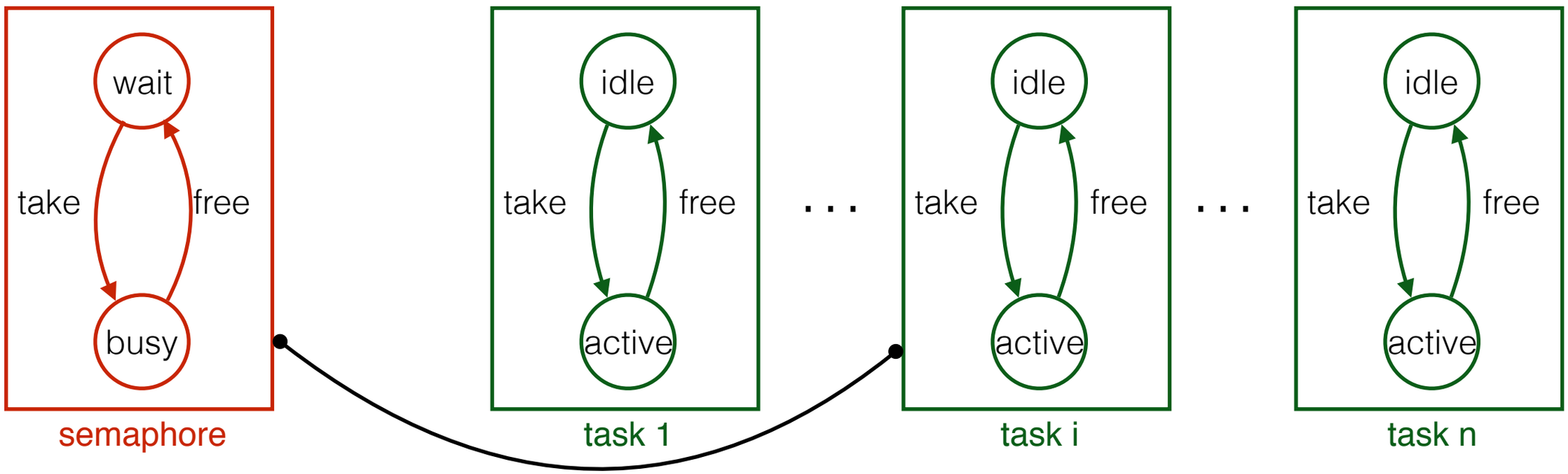
topologies symétriques
[Bozga, I, Sifakis, en cours]

topologies linéaires et arborescentes

architectures dynamiques

communication par conteneurs
(linéarisation)

Systèmes parallels paramètrés



absence de blocages
exclusion mutuelle

⇦ invariants d'interaction

topologies symétriques
[Bozga, I, Sifakis, en cours]

topologies linéaires et arborescentes

architectures dynamiques

communication par conteneurs
(linéarisation)

terminaison
fairness

⇦ fonctions de terminaison
décompositions à la Ramsey

Conclusions

- Travaux précédents 2002-2017
  - ➡ machines à compteurs
  - ➡ traductions de programmes vers machines à compteurs
  - ➡ logiques pour la mémoire (logique de séparation)

- Travaux en cours et futurs
  - ➡ logiques pour décrire à la fois les topologies et les données
  - ➡ conteneurs concurrents (linéarisation)
  - ➡ systèmes paramétrés