Vérification de systèmes asynchrones avec CADP

Radu Mateescu

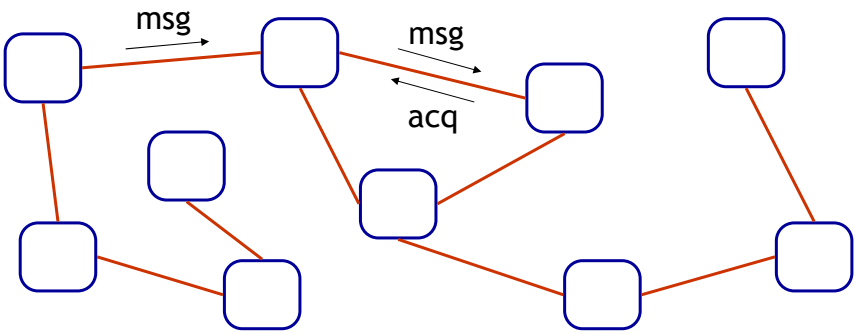
INRIA Rhône-Alpes / VASY

http://www.inrialpes.fr/vasy

Plan

- Introduction
- La boîte à outils CADP
  - Architecture et principes
  - Quelques nouveaux outils
- Illustration sur un système industriel
  - Modélisation formelle
  - Vérification par équivalence et logique temporelle
- Conclusion

Systèmes parallèles asynchrones



Caractéristiques :
- Ensemble d'agents distribués
- Communication par messages
- Non déterminisme

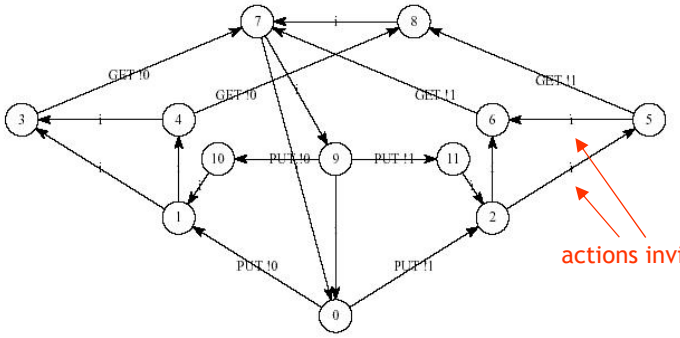
Applications :
- Matériel
- Logiciel
- Télécommunications

La boîte à outils CADP

http://www.inrialpes.fr/vasy/cadp

- Langages de description :
  - Normes ISO (LOTOS, E-LOTOS)
  - Réseaux d'automates communicants
- Fonctionnalités :
  - Compilation et prototypage rapide
  - Simulation interactive et guidée
  - Vérification par équivalences et logiques temporelles
  - Génération de tests
- Etudes de cas et applications :
  - 74 études de cas industrielles
  - 17 outils dérivés
- Diffusion : 340 sites (juin 2005)

Systèmes de transitions étiquetées

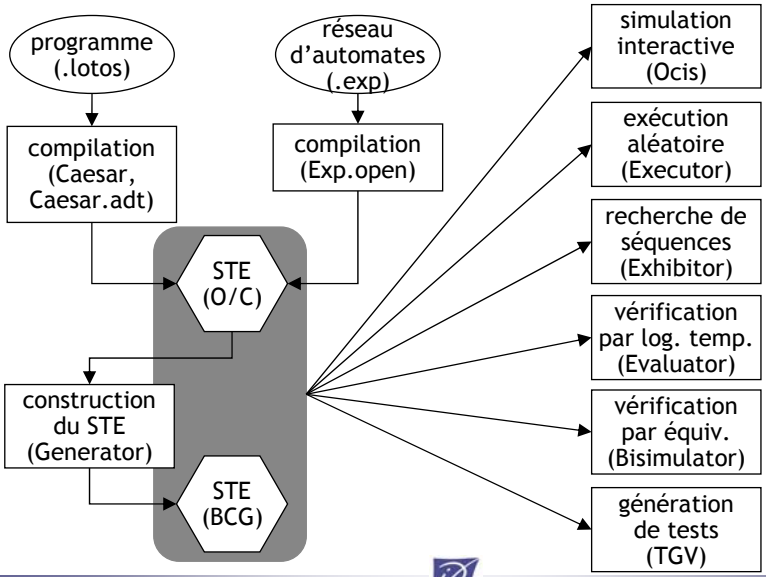


Deux représentations des STEs dans CADP :

- Explicite
  (fonction succ/pred)
  - BCG (Binary Coded Graphs)
- Vérification énumérative

- Implicite
  (fonction successeur)
  - OPEN/CAESAR [Garavel-98]
- Vérification à la volée

Architecture

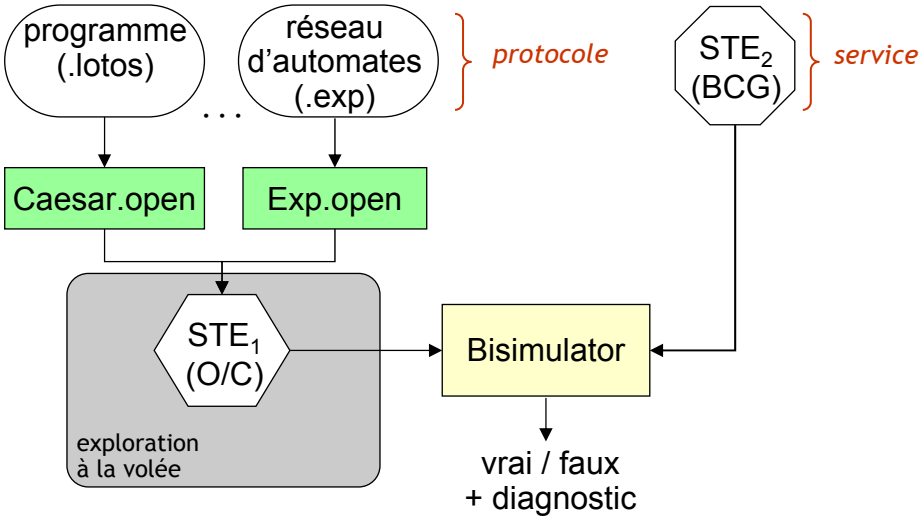


Caesar_Solve

(http://www.inrialpes.fr/vasy/cadp/man/html/caesar_solve_1.html)

- Bibliothèque de résolution à la volée de systèmes d'équations booléennes (SEBs) d'alternance 1
- Représentation de SEBs comme graphes booléens
- Quatre algorithmes de résolution [Mateescu-05] :
  - Parcours DFS, BFS (SEBs généraux)
  - Parcours DFS optimisés en mémoire (SEBs acycliques, disjonctifs/conjonctifs)
  - Complexité linéaire en taille du SEB
- Génération de diagnostics [Mateescu-00]
  - Exemples et contre-exemples (sous-graphes booléens)

Vérification par équivalences

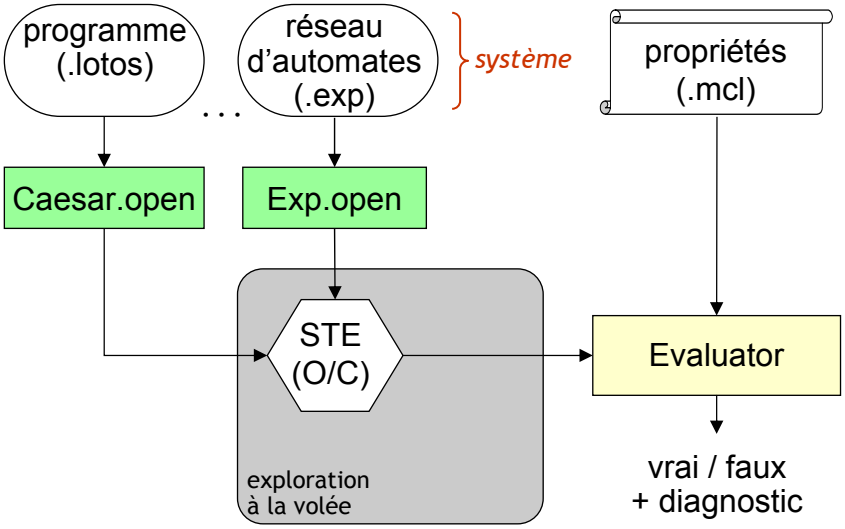


Bisimulator

(http://www.inrialpes.fr/vasy/cadp/man/html/bisimulator.html)

- Outil de vérification à la volée par équivalences (comparaison de STEs)
- Sept relations d'équivalence implémentées :
  - Bisimulations (forte, faible, branchement, tau*.a)
  - Équivalences de simulation (sûreté, trace – forte, faible)
  - Préordres (pour toutes les équivalences)
- Méthode de vérification [Mateescu-03] :
  - Traduction du problème vers une résolution de SEB
  - Utilisation de Caesar_Solve (stratégies DFS, BFS, …)
  - Génération de contre-exemples (sous-graphes acycliques)

Vérification par logiques temporelles

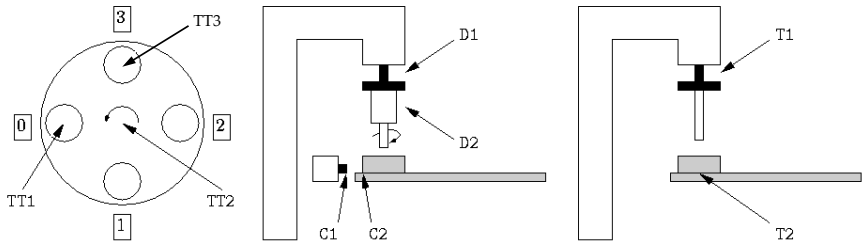


Evaluator 3.5

(http://www.inrialpes.fr/vasy/cadp/man/html/evaluator.html)

- Outil de vérification à la volée par logiques temporelles (évaluation de propriétés sur un STE)
- Mu-calcul régulier d'alternance 1 [Mateescu-Sighireanu-02] :
  - Modalités (possibilité/nécessité), opérateurs de point fixe
  - Expressions régulières sur les séquences d'actions
- Méthode de vérification [Mateescu-03] :
  - Traduction du problème vers une résolution de SEB
  - Utilisation de Caesar_Solve (stratégies DFS, BFS, …)
  - Génération d'exemples et contre-exemples
- Bibliothèques d'opérateurs dérivés
  - ACTL [deNicola-Vaandrager-90], schémas[Dwyer-et-al-99]

Illustration : unité de perçage des pièces métalliques [Bos-Kleijn-02]



(a) Table rotative (b) Perceuse (c) Testeur

Contrôleurs locaux

(dialogue avec les capteurs/actionneurs)

Portes pour recevoir les signaux des capteurs		
Dispositif	Porte	Fonction
Table	TT1	Pièce présente en position 0
	TT2	Rotation de 90° accomplie
	TT3	Pièce absente en position 3
Verrou	C1	Verrou relâché
	C2	Verrou bloqué
Perceuse	D1	Perceuse en position haute
	D2	Perceuse en position basse
Testeur	T1	Testeur en position haute
	T2	Testeur en position basse

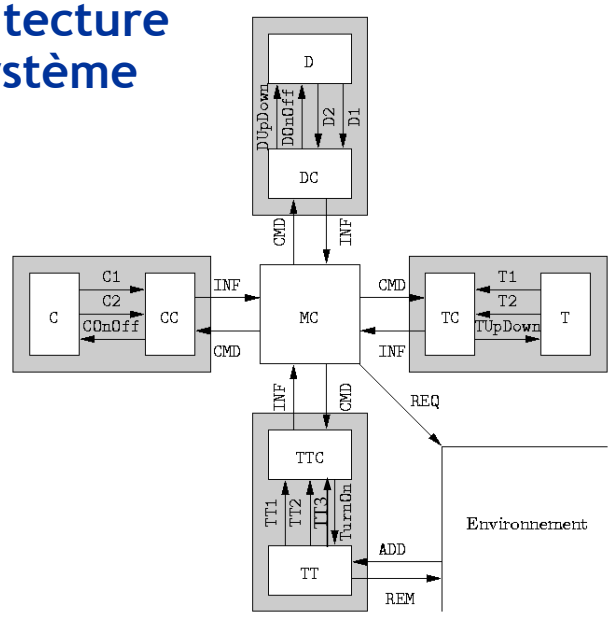
Portes pour envoyer les commandes aux actionneurs		
Dispositif	Porte	Fonction
Table	TurnOn	Démarre une rotation de 90°
Verrou	COnOff	Bloque ou relâche le verrou
Perceuse	DOnOff	Démarre ou arrête le moteur de la perceuse
	DUpDown	Démarre le mouvement ascendant ou descendant
Testeur	TUpDown	Démarre le mouvement ascendant ou descendant

Contrôleur principal

(dialogue avec les contrôleurs locaux et l'environnement)

Porte	Signal	Signification
CMD	Turn	Rotation de 90° de la table
	Drill	Perçage de la pièce en position 1
	Lock	Blocage du verrou
	Unlock	Relâchement du verrou
	Test	Test de la pièce en position 2
INF	Turned	Rotation de 90° de la table accomplie
	Present	Pièce présente en position 0
	Drilled	Perçage de la pièce en position 1 accompli
	Locked	Verrou bloqué
	Unlocked	Verrou relâché
	Tested	Pièce en position 2 testée
	Absent	Pièce en position 3 absente
REQ	Add	Chargement d'une pièce en position 0
	Remove	Récupération de la pièce en position 3

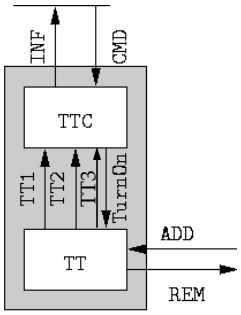
Architecture du système



Modélisation en LOTOS

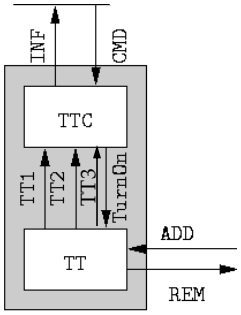
```
hide TT1, …, D1, …, C1, …, T1, … in
  (   ( TT [TT1, TT2, TT3, TurnOn, ADD, REM] (false, true, true, false)
          |[TT1, TT2, TT3, TurnOn]|
        TTC [TT1, TT2, TT3, TurnOn, INF, CMD] )
      |||
      ( D [D1, D2, DUpDown, DOnOff] (false, true)
          |[D1, D2, DUpDown, DOnOff]|
        DC [D1, D2, DUpDown, DOnOff, INF, CMD] )
      |||
      (C [C1,C2,COnOff] (false) |[C1, C2, COnOff]| CC [C1, C2, COnOff, INF, CMD])
      |||
      (T [T1,T2,TUpDown] (true) |[T1,T2,TUpDown]| TC [T1,T2,TUpDown,INF,CMD])
  )
  |[INF, CMD]|
      MC [REQ, INF, CMD] (false, true, true, false, false)
```

Processus TT
(table rotative)



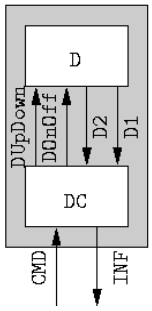

```
process TT [TT1, TT2, TT3, TurnOn, ADD, REM]
            (p0, p1, p2, p3:Bool) : noexit :=
   TurnOn;
      TT2;
         TT [TT1, TT2, TT3, TurnOn, ADD, REM]
            (p3, p0, p1, p2)
   []
   [not (p0)] -> ADD;
      TT1;
         TT [TT1, TT2, TT3, TurnOn, ADD, REM] (true, p1, p2, p3)
   []
   [p3] -> REM;
      TT3;
         TT [TT1, TT2, TT3, TurnOn, ADD, REM] (p0, p1, p2, false)
endproc
```


Processus TTC
(table rotative - contrôleur local)



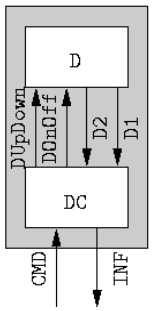

```
process TTC [TT1, TT2, TT3, TurnOn, INF, CMD] :
         noexit :=
   TT1;
      INF !Present;
         TTC [TT1, TT2, TT3, TurnOn, INF, CMD]
   []
   CMD !Turn;
      TurnOn;
         TT2;
            INF !Turned;
               TTC [TT1, TT2, TT3, TurnOn, INF, CMD]
   []
   TT3;
      INF !Absent;
         TTC [TT1, TT2, TT3, TurnOn, INF, CMD]
endproc
```

Processus D
(perceuse)



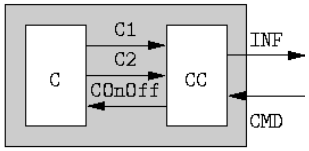

```
process D [D1, D2, DUpDown, DOnOff]
          (mode, pos:Bool) : noexit :=
   DOnOff;
      D [D1, D2, DUpDown, DOnOff]
        (not (mode), pos)
   []
   [mode] -> (
      DUpDown;
         ( [pos] ->       D2; D [D1, D2, DUpDown, DOnOff] (mode, not (pos))
           []
           [not (pos)] -> D1; D [D1, D2, DUpDown, DOnOff] (mode, not (pos))
         )
   )
endproc
```


Processus DC
(perceuse - contrôleur local)



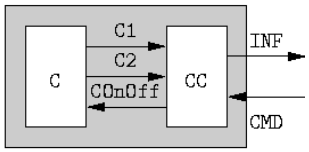

```
process DC [D1, D2, DUpDown, DOnOff, INF, CMD]
         : noexit :=
   CMD !Drill;
      DOnOff;
         DUpDown;
            D2;
               DUpDown;
                  D1;
                     DOnOff;
                        INF !Drilled;
                           DC [D1, D2, DUpDown, DOnOff, INF, CMD]
endproc
```

Processus C
(verrou)



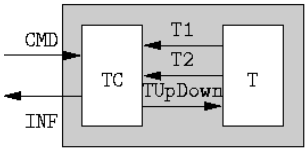

```
process C [C1, C2, COnOff] (locked:Bool) : noexit :=
   COnOff;
   (
   [locked] ->
      C1;
         C [C1, C2, COnOff] (not (locked))
   []
   [not (locked)] ->
      C2;
         C [C1, C2, COnOff] (not (locked))
   )
endproc
```


Processus CC
(verrou - contrôleur local)



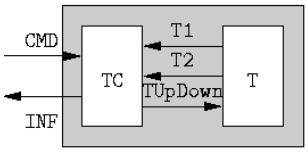

```
process CC [C1, C2, COnOff, INF, CMD] : noexit :=
   CMD !Lock;
      COnOff;
         C2;
            INF !Locked;
               CMD !Unlock;
                  COnOff;
                     C1;
                        INF !Unlocked;
                           CC [C1, C2, COnOff, INF, CMD]
endproc
```

Processus T
(testeur)




```
process T [T1, T2, TUpDown] (up:Bool) : noexit :=
   TUpDown;
   ( [up] -> (
        T2; T [T1, T2, TUpDown] (not (up))      (* produit OK *)
        []
        T [T1, T2, TUpDown] (not (up))          (* produit KO *)
     )
     []
     [not (up)] ->
        T1;
           T [T1, T2, TUpDown] (not (up))
   )
endproc
```


Processus TC
(testeur - contrôleur local)

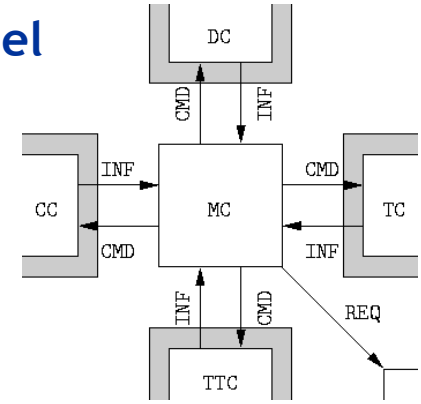



```
process TC [T1, T2, TUpDown, INF, CMD] : noexit :=
   CMD !Test; TUpDown;
      (  T2;
           TUpDown;
              T1;
                 INF !Tested !true;
                    TC [T1, T2, TUpDown, INF, CMD]
        []
        TUpDown;
           T1;
              INF !Tested !false;
                 TC [T1, T2, TUpDown, INF, CMD]
      )
endproc
```

Processus MC séquentiel
(contrôleur principal – phase 1)

```
process MC_SEQ [REQ, INF, CMD]
        (p0, p1, p2, p3, tr:Bool) : noexit :=
  (
    [not (p0)] ->
      REQ !Add;
        INF !Present;
          exit (true)
    []
    [p0] ->
      exit (p0)
  )
  >>
  accept new_p0:Bool in
  ...
```

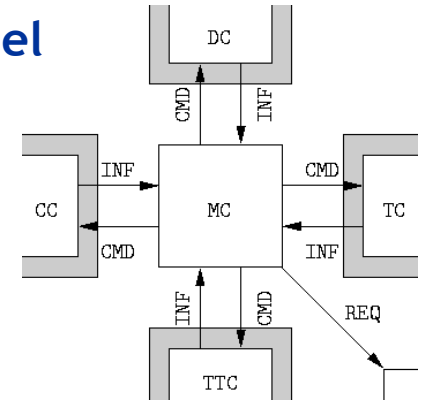
Phase1



Processus MC séquentiel
(contrôleur principal – phase 2)

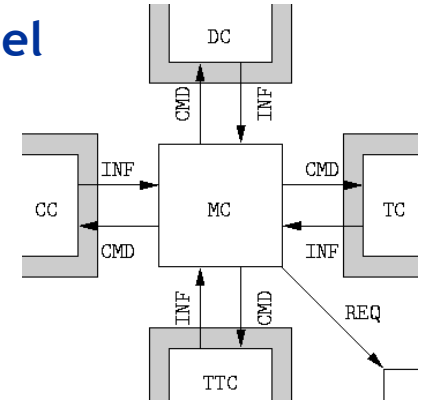
```
process MC_SEQ [REQ, INF, CMD]
        (p0, p1, p2, p3, tr:Bool) : noexit :=
  ...
  accept new_p0:Bool in
  (
    (
      [p1] -> CMD !Lock; INF !Locked;
                CMD !Drill; INF !Drilled;
                  CMD !Unlock; INF !Unlocked;
                    exit
      []
      [not (p1)] -> exit
    )
    >>
    ...
```

Phase2



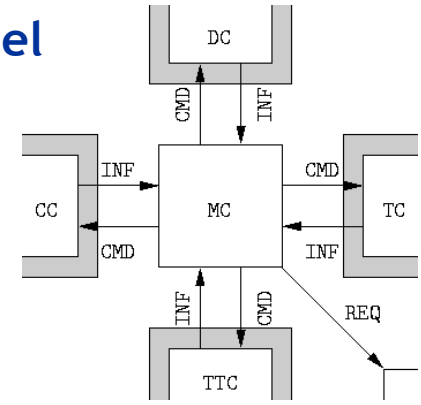
Processus MC séquentiel
(contrôleur principal – phase 3)

```
process MC_SEQ [REQ, INF, CMD]
        (p0, p1, p2, p3, tr:Bool) : noexit :=
  ...
    (
      [p2] ->
        CMD !Test;
          INF !Tested ?r:Bool;
            exit (r)
      []
      [not (p2)] ->
        exit (tr)                          Phase3
    )
    >>
    accept new_tr:Bool in
    ...
```

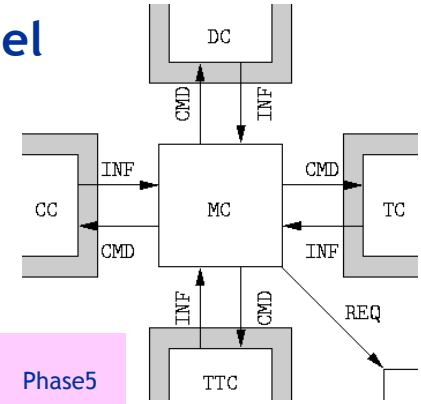


Processus MC séquentiel
(contrôleur principal – phase 4)

```
process MC_SEQ [REQ, INF, CMD]
        (p0, p1, p2, p3, tr:Bool) : noexit :=
  ...
    accept new_tr:Bool in
    (
      (  [p3] -> REQ !Remove !tr;
                   INF !Absent;
                     exit (false)
        []
        [not (p3)] ->
          exit (p3)                        Phase4
      )
      >>
      accept new_p3:Bool in
        ...
```



Processus MC séquentiel
(contrôleur principal – phase 5)



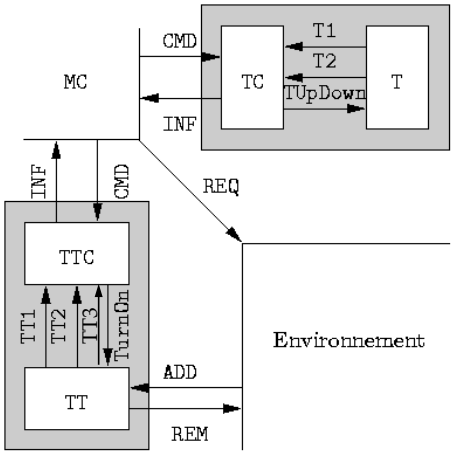

```
process MC_SEQ [REQ, INF, CMD]
        (p0, p1, p2, p3, tr:Bool) : noexit :=
  ...
      >>
      accept new_p3:Bool in
         CMD !Turn;                                  Phase5
            INF !Turned;
               MC_SEQ [REQ, INF, CMD]
                        (new_p3, new_p0, p1, p2, new_tr)
    )
  )
endproc
```

Processus MC parallèle

```
process MC_PAR [REQ, INF, CMD] (p0, p1, p2, p3, tr:Bool) : noexit :=
   (
      Phase1   |||   Phase2   |||   Phase3   |||   Phase4
   )
   >>
      accept new_p0, new_p1, new_p2, new_p3, new_tr:Bool in
         CMD !Turn;
            INF !Turned;                                 Phase5
               MC_PAR [REQ, INF, CMD]
                        (new_p3, new_p0, new_p1, new_p2, new_tr)
endproc
```

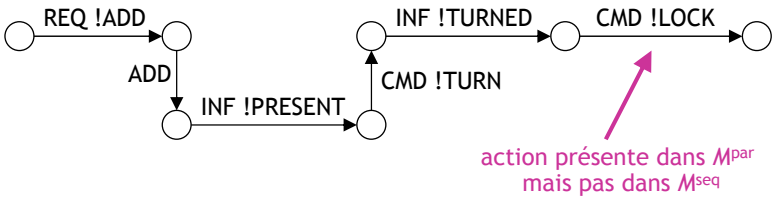
Processus Env
(environnement)

```
process Env [REQ, ADD, REM, ERR]
            : noexit :=
   REQ !Add;
     ADD;
        Env [REQ, ADD, REM, ERR]
   []
   REQ !Remove ?r:Bool;
   (  [r] ->
        REM;
           Env [REQ, ADD, REM, ERR]
     []
     [not (r)] ->
        ERR;
           REM;
              Env [REQ, ADD, REM, ERR]
   )
endproc
```



Vérification par équivalence
(Bisimulator)

- M^{seq}, M^{par} : LTS du système avec contrôleur séquentiel (resp. parallèle)
  - $M^{seq} \leq M^{par}$ modulo la bisimulation de branchement
    ➔ le contrôleur séquentiel est simulé par celui parallèle
  - $M^{seq} \neq M^{par}$ modulo toutes les équivalences
    Contre-exemple :

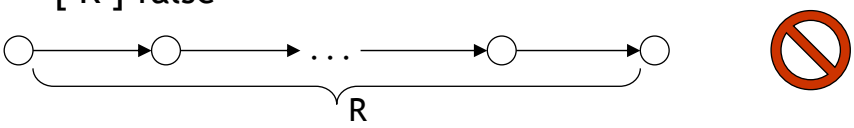


Vérification par logique temporelle

(Evaluator 3.5)

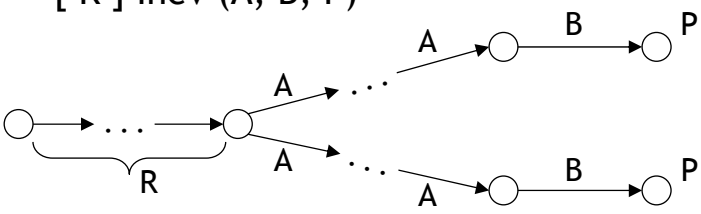
- Propriétés de *sûreté* (« rien de mal n'arrivera »)

  [R] false



- Propriétés de *vivacité* (« qqe chose de bien arrivera »)

  [R] Inev (A, B, P)



Propriétés de sûreté (1/3)

À chaque cycle de traitement, après qu'une pièce ait été chargée et que la table ait effectué une rotation, le contrôleur principal ne peut pas envoyer une commande de rotation avant que la pièce n'ait été percée.

```
[   (nil | (true* . "INF !TURNED")) .
    (not "INF !TURNED")* .
    "INF !PRESENT" .
    (not "INF !TURNED")* .
    "INF !TURNED" .
    (not "INF !DRILLED")* .
    "CMD !TURN"
]   false
```

Propriétés de sûreté (2/3)

À chaque cycle de traitement, le contrôleur principal ne peut pas envoyer une commande de perçage avant que le verrou n'ait été bloqué.

```
[   (nil | (true* . "INF !TURNED")) .
    (not "INF !LOCKED")* .
    "CMD !DRILL"
] false
```

Propriétés de sûreté (3/3)

Quand des pièces ont été chargées dans les positions de perçage et de test, le contrôleur principal ne peut pas envoyer, pendant le même cycle, une commande de test avant d'envoyer une commande de perçage.

```
[   true* . "INF !PRESENT" .
    true* . "INF !PRESENT" .
    true* . "INF !PRESENT" .
    true* . "CMD !TEST" .
    (not "INF !TURNED")* .
    "CMD !DRILL"
] false
```

Propriétés de vivacité (1/3)

Le système ne comporte pas de blocage.

[true*] < true > true

Propriétés de vivacité (2/3)

Chaque fois qu'une pièce est chargée, elle sera inévitablement percée après la prochaine rotation de la table.

```
[ true* . "INF !PRESENT" ]
   lnev (
      not "INF !TURNED",
      "INF !TURNED",
      lnev (
         not "INF !TURNED",
         "INF !DRILLED",
         true
      )
   )
```

Propriétés de vivacité (3/3)

Chaque commande envoyée par le contrôleur principal aux actionneurs sera inévitablement suivie d'un acquittement, avant la prochaine rotation de la table.

```
[ true* ] (
 [ "CMD !LOCK" ] lnev (not "INF !TURNED", "INF !LOCKED", true)
   and
 [ "CMD !DRILL" ] lnev (not "INF !TURNED", "INF !DRILLED", true)
   . . .
   and
 [ "CMD !TEST" ] lnev (not "INF !TURNED", 'INF !TESTED.*', true)
   and
 [ "CMD !TURN" ] lnev (not "INF !TURNED", "INF !TURNED", true)
)
```

Conclusion

- Autres fonctionnalités d'analyse de CADP :
  - Vérification distribuée [Joubert-Mateescu-04,05]
  - Vérification semi-compositionnelle [Lang-05]
  - Réductions à la volée [Mateescu-05]
  - Evaluation de performances [Garavel-Hermanns-02]
- Pour en savoir plus :
  - http://www.inrialpes.fr/vasy/Publications
  - http://www.inrialpes.fr/vasy/cadp/man
  - http://www.inrialpes.fr/vasy/cadp/case-studies
  - http://www.inrialpes.fr/vasy/cadp/software