Conceptions Formelles

Alain Griffault

Alain.Griffault@labri.fr

LaBRI-MVTsi

Plan

- Quelques faits.
- Les techniques formelles.
- Conceptions formelles
  - La vérification de modèles.
  - Le rafinement de modèles.
  - La synthèse de contrôleurs.
- Un exemple : le contrôle du niveau d'une cuve.

Quelques faits

La banque de New York

Le 21 novembre 1985, pour une variable entière passant *malencontreusement* de 32768 à 0 :

- La *crise* des bons du Trésor.
- Un emprunt de 20 milliards de dollars de la banque au gouvernement.
- Les intérêts de l'emprunt . . .

Le crash d'AT&T

Le 15 janvier 1990, un patch

```
switch(i){
   case 1 :  function1(); break;
   case 2 :  function2(); // break;
   default:  defaultfunction();
}
```

devant réduire des temps de réponse.

Le crash d'AT&T

Le 15 janvier 1990, un patch

```
switch(i){
   case 1 :  function1(); break;
   case 2 :  function2(); // break;
   default:  defaultfunction();
}
```

devant réduire des temps de réponse.

Pour un `break;` oublié et une utilisation pendant 9 heures :

- 6000 communications interrompues
- 70 000 000 de coups de fil qui n'ont pu être passés.
- la réputation d'AT&T entachée.

Le Pentium

Juin 1994 un chercheur en théorie des nombres trouvent des résultats expérimentaux en contradiction avec des résultats antérieurs.

Juin-Octobre 1994 Il isole le problème, puis prévient Intel.

2 novembre Intel a corrigé le bug (un algo plus efficace, mais pas toujours correct !).

Pour Intel :

- Coût de l'échange des processeurs.
- Psychose du Pentium.
- Adopte depuis des méthodes formelles pour la conception des processeurs.

Ariane 5-01 (4 juin 1996)

- Le 23 juillet, la commission d'enquête remet son rapport : La fusée a eu un comportement nominal jusqu'à la 36ème seconde de vol. Puis les systèmes de référence inertielle (SRI) ont été simultanément déclarés défaillants. Le SRI n'a pas transmis de données correctes parce qu'il était victime d'une erreur d'opérande trop élevée du "biais horizontal" . . . Or, sur Ariane 5, l'accélération au décollage est beaucoup plus forte que sur Ariane 4, . . . La variable (l'accélération au décollage) a dépassé la plage des valeurs autorisée par le logiciel de vol. La fonction d'alignement qui reste active environ 40 secondes est une exigence d'Ariane 4 mais n'a aucune utilité sur Ariane 5. Elle a cependant été maintenue pour des raisons de commodité. . .
- Perte financière énorme.

Le Therac-25

Le Therac-25 est un appareil médical doté d'un interface *convivial* pour le traitement des cancers. Il émet un faisceau soit d'électrons (tissus superficiels), soit de photons (très hautes énergies pour les tissus internes).

1976 : premier prototype utilisé en milieu hospitalier.

1982 : première vente.

Le Therac-25

Le Therac-25 est un appareil médical doté d'un interface *convivial* pour le traitement des cancers. Il émet un faisceau soit d'électrons (tissus superficiels), soit de photons (très hautes énergies pour les tissus internes).

juin 1985 : Pour une tumeur à l'épaule, une patiente ressent une très forte chaleur. Elle reçoit entre 15000 à 20000 rads au lieu de 200. Impossible de reproduire l'incident $\rightarrow$ affaire classée.

Le Therac-25

Le Therac-25 est un appareil médical doté d'un interface *convivial* pour le traitement des cancers. Il émet un faisceau soit d'électrons (tissus superficiels), soit de photons (très hautes énergies pour les tissus internes).

juillet 1985 : Idem pour une tumeur au bassin, le patient décède 5 mois plus tard. L'analyse conclue par une défaillance de micro-commutateurs $\rightarrow$ redondance corrective.

Le Therac-25

Le Therac-25 est un appareil médical doté d'un interface *convivial* pour le traitement des cancers. Il émet un faisceau soit d'électrons (tissus superficiels), soit de photons (très hautes énergies pour les tissus internes).

décembre 1985, mars 1986 : deux nouveaux incidents, 1 mort.
L'analyse conclue par la présence de chocs électriques dus à un court-circuit.

Le Therac-25

Le Therac-25 est un appareil médical doté d'un interface *convivial* pour le traitement des cancers. Il émet un faisceau soit d'électrons (tissus superficiels), soit de photons (très hautes énergies pour les tissus internes).

avril 1986 : un nouvel incident, 1 nouveau mort, mais le second pour un opérateur qui peut alors reproduire l'erreur. La société reconnaît l'erreur logicielle. L'erreur est *corrigée*.

Le Therac-25

Le Therac-25 est un appareil médical doté d'un interface *convivial* pour le traitement des cancers. Il émet un faisceau soit d'électrons (tissus superficiels), soit de photons (très hautes énergies pour les tissus internes).

janvier 1987 : un dernier incident, 1 dernier mort. De nouvelles erreurs logicielles sont découvertes $\rightarrow$ arrêt d'exploitation du Therac-25.

Limites des approches actuelles

- Coût des tests.
- Interprétation(s) des termes usuels $\rightarrow$ UML.
- Ambiguïté des méthodes semi-formelles (# sémantiques UML).
- Bonne maîtrise des applications ayant des objets *statiques*.
- Maîtrise difficile de la programmation événementielle.
- UML (RdP, ST, MSC), mais aucune indication sur la composition d'un RdP et d'un MSC.
- Maintenance évolutive difficile.

Tendances et préconisations

- Dans le schéma de certification Critère Commun ou ITSEC, il est nécessaire à partir d'un certain niveau de certification d'utiliser des techniques formelles.
- EAL (Evaluation Assurance Level) > 4 rend obligatoire l'utilisation de méthodes formelles.
- Obligatoire pour les grandes entreprises. . . qui l'imposent de plus en plus à leurs sous-traitants. $\rightarrow$ effet boule de neige depuis 3 ans.

Les techniques formelles

Définitions

Objectif Pouvoir raisonner sur les logiciels et les systèmes afin de connaître leurs comportements et les contrôler.

Moyen Les systèmes sont des objets mathématiques.

Processus :

1. Obtenir un modèle formel du logiciel ou du système.
2. L'analyser par une technique formelle.
3. Transposer les résultats obtenus sur les modèles aux logiciels et systèmes réels.

Problèmes de l'approche :

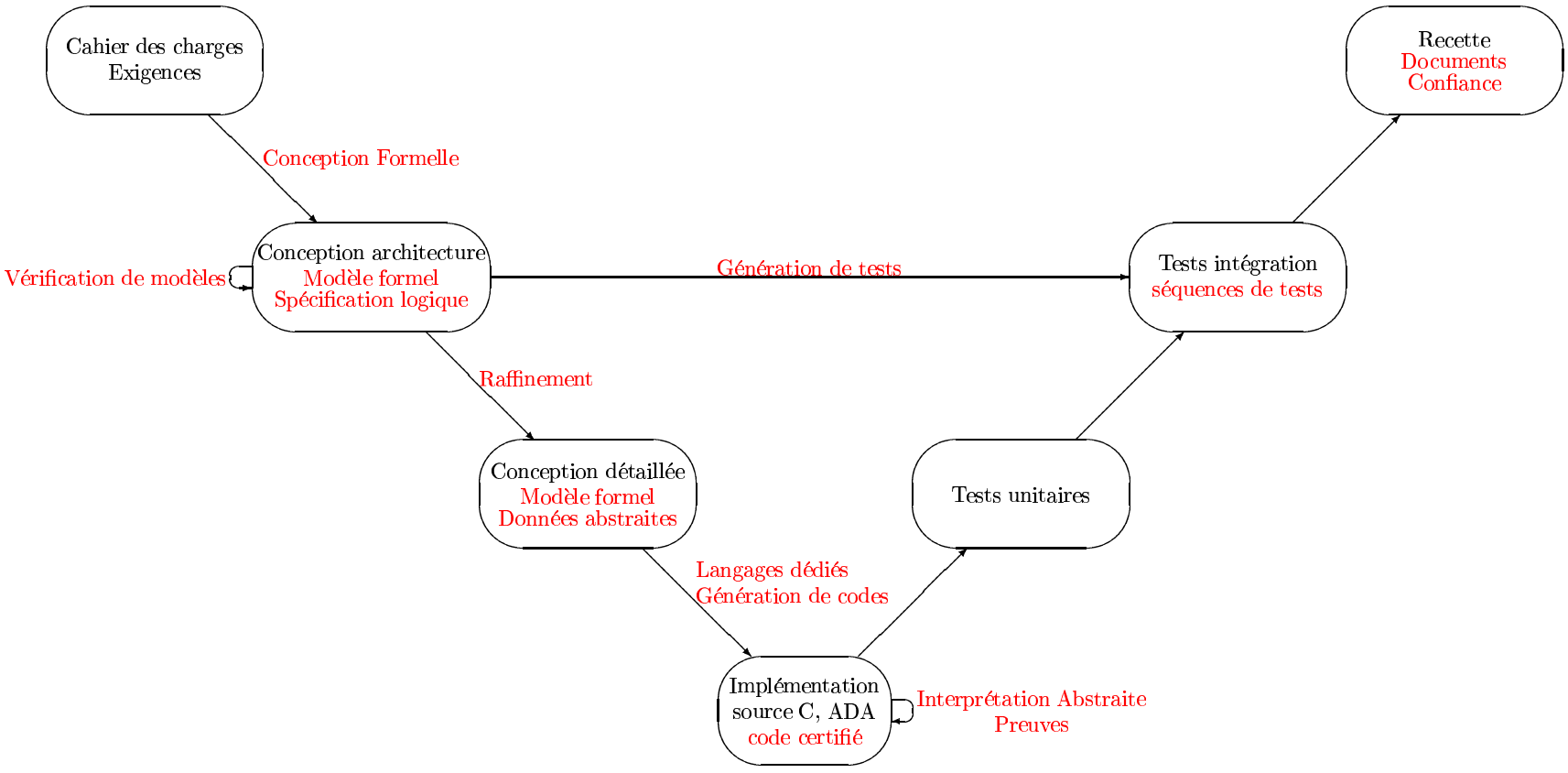
- Le modèle est-il fidèle ? *validation*.
- Peux t-on tout vérifier ? *décidabilité*.
- La transposition des résultats est-elle toujours possible ? *abstraction*

Méthodes formelles : Pour qui ? Pour quoi

Pour des raisons économiques, le surcroît de travail fait que seuls les concepteurs et/ou réalisateurs de systèmes complexes et/ou critiques s'intéressent aujourd'hui aux techniques formelles. Un système est généralement considéré comme tel lorsque :

- la vie de personnes est lié à son bon fonctionnement. C'est le cas des logiciels embarqués pour le transport (avion, train, voiture, navette...), des contrôleurs de systèmes (centrales nucléaires, matériels médicaux...).
- le coût économique d'un dysfonctionnement est catastrophique. C'est le cas des logiciels mis dans le silicium pour être produit en très grande quantité (électroménager, téléphonie...)

Panorama des techniques (I)



Panorama des techniques (II)

Les démonstrateurs de théorèmes sont des outils qui aident à fabriquer des preuves **justes.** Un programme est vu comme une fonction de calcul, et on va chercher à montrer que d'une part la fonction termine; d'autre part que le résultat calculé est bien celui attendu.
Cette technique est bien adaptée pour des programmes séquentiels complexes, mais moins bien pour les systèmes de types réactifs.

Panorama des techniques (II)

L'interprétation abstraite consiste à transformer un problème concret dans un modèle abstrait plus simple, sur lequel les *preuves* seront plus faciles à effectuer. Un exemple d'interprétation abstraite est la preuve par 9.
Cette technique est bien adaptée pour les programmes manipulant beaucoup de données, mais moins bien pour les systèmes de types réactifs.

Panorama des techniques (II)

La génération automatique de codes exécutables transforme une description, dans un langage de très haut niveau et très spécialisé, en un programme dans un langage *classique* (C ou ADA).
Cette technique est bien adaptée pour les systèmes temps réels.

Panorama des techniques (II)

Les vérificateurs de modèles sont des outils qui permettent de décider si un modèle d'un système satisfait ou non des propriétés logiques. Dans cette démarche, le modèle est une représentation du système, et les propriétés représentent les fonctionnalités attendues du système. Cette technique est très bien adaptée pour la conception des systèmes réactifs, mais manipule assez mal les données. En pratique, les utilisateurs appliquent *manuellement* des techniques d'interprétation abstraite lorsque le système possèdent de trop nombreuses données.

Panorama des techniques (II)

Le raffinement permet, en un nombre d'étapes plus ou moins important de passer d'une spécification à une implantation en préservant à chaque étape les propriétés essentielles du système.
Cette technique est bien adaptée pour des programmes séquentiels complexes, mais moins bien pour les systèmes de types réactifs.

Panorama des techniques (II)

Les générateurs de tests sont des outils qui permettent de générer des séquences de tests qui ont la particularité d'être d'une part conforme aux spécifications; d'autre part conforme à des objectifs de tests spécifiques. Ils permettent ainsi de contrôler que le système réel est plus ou moins conforme avec le cahier des charges initial.

Cette technique est très bien adaptée pour les systèmes répartis et notamment pour les protocoles. Elle se démarque des précédentes du fait que c'est la seule à faire intervenir le système réel.

Panorama des techniques (II)

La simulation stochastique Les processus stochastiques ajoutent dans la description des systèmes des grandeurs probabilistes. Il est alors possible de calculer des statistiques relatives à des propriétés. La sûreté de fonctionnement qui cherche à maîtriser les systèmes en présence de dysfonctionnements est un des domaines applicatifs des processus stochastiques.

Conceptions formelles

La vérification formelle

Le principe est simple :

- Le système est représenté par un modèle M,
- Les exigences sont décrites par une propriété ϕ,
- Un vérificateur de modèles prend en entrée d'une part M, d'autre part ϕ et répond **automatiquement** à la question $M \models \phi$? De plus, il peut fournir une explication lorsque la réponse est négative.

$\rightarrow$ nombreux outils académiques disponibles.

La vérification formelle

Les limites théoriques sont données par les couples (classe de modèles, logique) pour lesquels le problème est décidable.

- Systèmes finis : toutes les logiques comportementales sont décidables.
- Systèmes infinis : presque rien n'est décidable $\rightarrow$ études de systèmes infinis particuliers (avec compteurs, FIFO, temporisés) pour lesquels des fragments de logiques sont décidables.

$\rightarrow$ nombreux travaux de recherches théoriques.

La vérification formelle

Les limites pratiques sont fixées par :

- Les machines et par les techniques utilisées pour coder le graphe en mémoire (explicite, BDD, ordre partiel, DBM . . .). Actuellement de l'ordre de 100 variables booléennes.
- La validité du modèle M.
- La difficulté de l'écriture de ϕ.

$\rightarrow$ nombreux travaux de recherches algorithmiques et méthodologiques.

Le raffinement

Le principe est simple :

- Le système est représenté par un modèle M_i,
- Un modèle plus concret M_{i+1} est proposé.
- Un outil vérifie (et/ou) montre que M_{i+1} est un raffinement de M_i. Le raffinement est une relation de simulation (d'implantation) :
  - $e_1^1 e_1^{m_1} \dots e_n^1 e_n^{m_n} \in M_{i+1} \implies e_1 e_2 \dots e_n \in M_i$

  qui enlève des comportements (non déterminisme, pile vide, ...). Le raffinement peut porter sur les données et/ou sur les comportements.

Le processus est itéré jusqu'au code. Celui-ci est donc une implantation certifiée correcte du système M_0 de départ. $\rightarrow$ quelques outils académiques disponibles.

Le raffinement

Les limites théoriques sont données par les classes de décidabilité et les logiques utilisées. $\rightarrow$ nombreux travaux de recherches théoriques.

Le raffinement

Les limites pratiques sont fixées par :

- La validité du modèle M_0.
- Le nombre important de lemmes à établir plus on se rapproche du code. C'est pourquoi dans certains outils, il est possible de définir des postulats.

$\rightarrow$ Le langage B pour le métro Météor.

La synthèse de contrôleur

Le principe est simple :

- Le système est représenté par un modèle M,
- Les exigences sont décrites par une propriété ϕ,
- Un outil de synthèse de contrôleur prend en entrée d'une part M, d'autre part ϕ et retourne un modèle C tel que $(M \times C) \models \phi$.

$\rightarrow$ très peu d'outils académiques disponibles.

La synthèse de contrôleur

Les limites théoriques sont données par les couples (classe de modèles, logique) pour lesquels le problème est décidable.

- événements contrôlables ou non,
- événements observables ou non.

$\rightarrow$ nombreux travaux de recherches théoriques.

La synthèse de contrôleur

Les limites pratiques sont fixées par :

- Les machines et par les techniques utilisées pour coder le graphe en mémoire (explicite, BDD, ordre partiel, DBM . . .). Actuellement de l'ordre de 10 variables booléennes, donc non industrialisable.
- La difficulté de l'écriture de ϕ.

Un exemple : le contrôle du niveau d'une cuve

Le cahier des charges

Le système que l'on souhaite concevoir est composé :

- d'un réservoir contenant **toujours** suffisamment d'eau pour alimenter l'exploitation,
- d'une cuve,
- de deux canalisations amont reliant le réservoir à la cuve, et permettant d'amener l'eau à la cuve,
- d'une canalisation aval permettant de vider l'eau de la cuve,
- chaque canalisation est équipée d'une vanne contrôlable, afin de réguler l'alimentation et la vidange de la cuve,
- d'un contrôleur.

Le cahier des charges

La vanne

Les vannes sont toutes de même type, elles possèdent trois niveaux de débits correspondant à trois diamètres d'ouverture : 0 correspond à la vanne fermée, 1 au diamètre intermédiaire et 2 à la vanne complètement ouverte. Les vannes sont contrôlables par les deux instructions `inc` et `dec` qui respectivement augmente et diminue l'ouverture. Malheureusement, la vanne est sujet à défaillance, auquel cas le système de commande devient inopérant, la vanne est désormais pour toujours avec la même ouverture.

Le cahier des charges

Les canalisations

Elles sont supposées parfaites.

Le cahier des charges

La Cuve

Elles est munie de quatre capteurs situés à quatre hauteurs qui permettent de délimiter cinq zones. La zone 0 est comprise entre le niveau 0 et le niveau du capteur le plus bas; la zone 1 est comprise entre ce premier capteur et le second, et ainsi de suite.

Le cahier des charges

Les débits

Les règles d'évolution du niveau de l'eau dans la cuve :

- si (V1amont + V2amont $>$ Vaval) alors au temps suivant, le niveau aura augmenté d'une unité.
- si (V1amont + V2amont $<$ Vaval) alors au temps suivant, le niveau aura baissé d'une unité.
- si (V1amont + V2amont = Vaval = 0) alors au temps suivant, le niveau n'aura pas changé.
- si (V1amont + V2amont = Vaval) et (Vaval $>$ 0) alors au temps suivant, le niveau pourra :
  - avoir augmenté d'une unité,
  - avoir baissé d'une unité,
  - être resté le même.

Le cahier des charges

Les spécifications

Le système devra vérifier les propriétés suivantes :

Property 0 *En l'absence de panne sur les vannes, et après une éventuelle période initiale la plus courte possible, le niveau de la cuve doit rester entre les niveaux 1 et 3 inclus, et le débit de la vanne aval doit rester maximal.*

Property 0 *En présence d'une seule panne, le niveau de la cuve doit rester entre les niveaux 1 et 3 inclus, et le débit de la vanne aval doit rester le plus souvent possible maximal, et ne jamais devenir nul.*

Property 0 *En présence d'au moins deux pannes, le niveau de la cuve doit rester entre les niveaux 1 et 3 inclus, même s'il faut pour cela arrêter définitivement la vanne aval.*

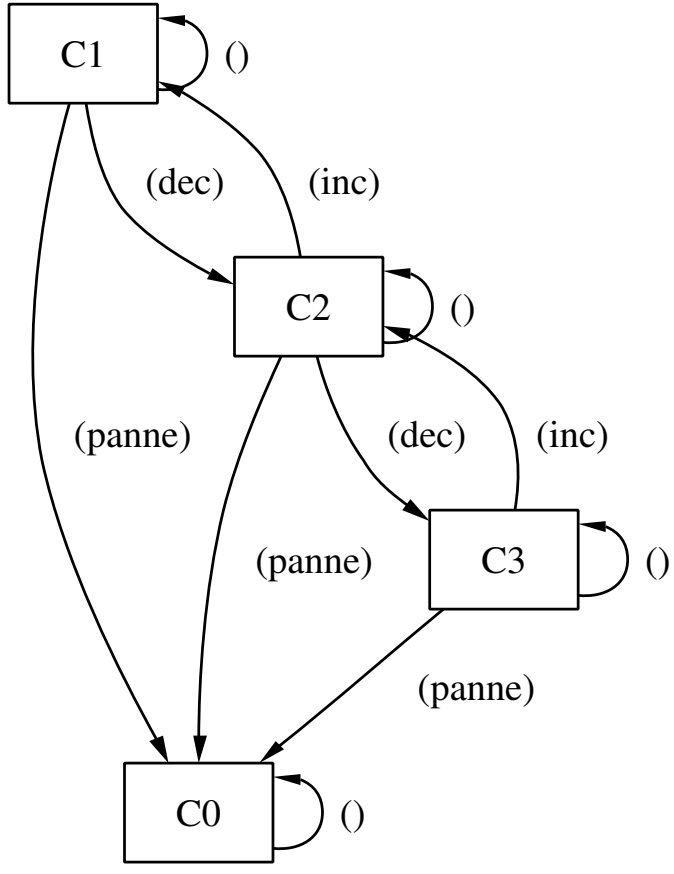
Le modèle ALTARICA

Le nœud Vanne

```
node Vanne
  state
    debit : [0,2] : public;
    on : bool : public;
  event
    inc, dec, panne;
  trans
    on |- inc -> debit := debit+1;
    on |- dec -> debit := debit-1;
    on |- panne -> on := false;
  init
    on := true, debit := 0;
edon
```

Le modèle ALTARICA

Le nœud Vanne



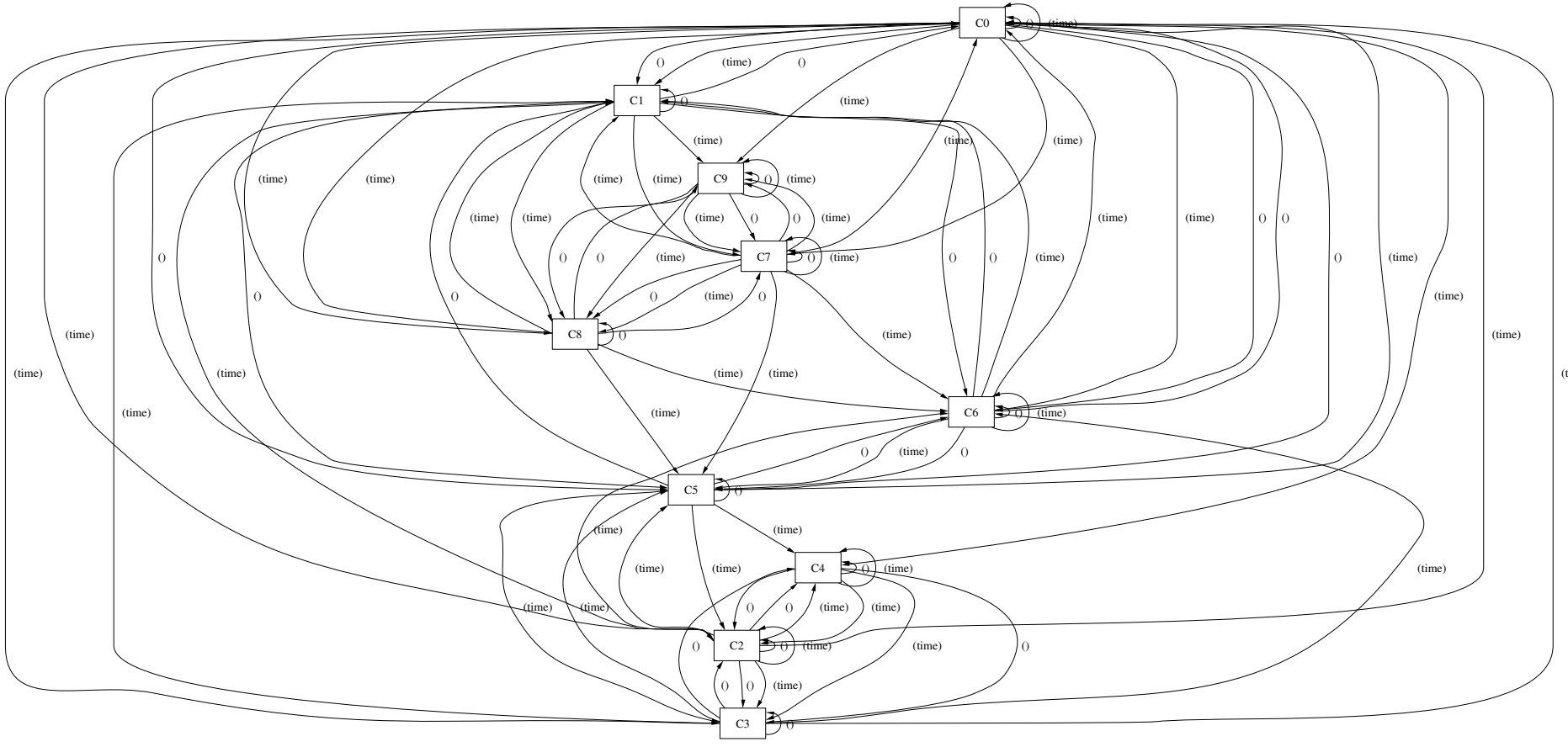
Le modèle ALTARICA

Le nœud Cuve

```
node Cuve
  flow  f1, f2, f3 : [0,2];
  state niveau : [0,4] : public;
  event time;
  trans
    (f1+f2=f3) |- time -> ;
    (f1+f2=f3) & (f3>0) |- time ->
                 niveau := niveau+1;
    (f1+f2=f3) & (f3>0) |- time ->
                 niveau := niveau-1;
    (f1+f2>f3) |- time -> niveau := niveau+1;
    (f1+f2<f3) |- time -> niveau := niveau-1;
  init  niveau := 2;
edon
```

Le modèle ALTARICA

Le nœud Cuve



Le modèle ALTARICA

Le système

```
node System
  sub     V1, V2, V3 : Vanne;
          Cu : Cuve;
  state   ctrl : bool;
          nbPanne : [0,3];
  event   commande, panne, time;
  trans   ctrl  |- commande -> ctrl := false;
          ~ctrl |- panne    -> ctrl := true,
                               nbPanne := nbPanne+1;
          ~ctrl |- time     -> ctrl := true;
  assert  Cu.f1 = V1.debit;
    ...
  sync    <commande, V1.inc>;
    ...
```

La spécification

```
with System, SystemP0, SystemP0Raf do
 dead      := any_s - src(any_t - self_epsilon);
 ER        := [Cu.niveau = 0] | [Cu.niveau = 4]
 P0         := ER & [nbPanne < 1];
 P1         := ER & [nbPanne < 2];
 P2         := ER & [nbPanne < 3];
 P3         := ER & [nbPanne < 4];
 test(any_s,0) >  '$NODENAME.res';
 test(any_t,0) >> '$NODENAME.res';
 test(P0,0)    >> '$NODENAME.res';
 test(P1,0)    >> '$NODENAME.res';
 test(P2,0)    >> '$NODENAME.res';
 test(P3,0)    >> '$NODENAME.res';
done
```

La vérification

```
TEST(any_s=0) [FAILED] actual size = 2134
TEST(any_t=0) [FAILED] actual size = 9697
TEST(P0=0) [FAILED] actual size = 86
TEST(P1=0) [FAILED] actual size = 407
TEST(P2=0) [FAILED] actual size = 731
TEST(P3=0) [FAILED] actual size = 839
```

L'itération

```
node SystemV1
  sub     V1, V2, V3 : Vanne;
          Cu : Cuve;
  state   ctrl : bool;
          nbPanne : [0,3];
  event   commande, panne, time;
  trans   ~ctrl |- panne     -> ctrl := true,
                                nbPanne := nbPanne+1;
          ~ctrl |- time      -> ctrl := true;
    n < 2                    |- inc1inc2, inc1dec3, i
    n = 2 & (d1+d2<d3) |- inc1, inc2, dec3 -> ;
    n = 2 & (d1+d2=d3) & (d3<2) |- inc1inc3, in
    n = 2 & (d1+d2>d3) |- inc3, dec1, dec2 -> ;
    n > 2                    |- dec1dec2, inc3dec1, i
  assert Cu.f1 = V1.debit;
    ...
```

La synthèse (le contrôleur initial)

```
node ControleurPermissif
  event nop, inc1, inc2, inc3, dec1, dec2, dec3
    inc1inc2, inc1inc3, inc1dec2, inc1dec3, inc
    inc2dec3, inc3dec1, inc3dec2, dec1dec2, dec
    inc1inc2inc3, inc1inc2dec3, inc1dec2inc3, i
    dec1inc2inc3, dec1inc2dec3, dec1dec2inc3, d
  flow  d1, d2, d3 : [0,2];
        n : [0,4];
  trans true  |- nop, inc1, inc2, inc3, dec1, d
              inc1inc2, inc1inc3, inc1dec2, inc1
              inc2dec3, inc3dec1, inc3dec2, dec1
              inc1inc2inc3, inc1inc2dec3, inc1de
              dec1inc2inc3, dec1inc2dec3, dec1de
          -> ;
edon
```

La synthèse (le systeme contrôlé)

```
node SystemControle
  sub     V1, V2, V3 : Vanne;
          Cu : Cuve;
          Co : ControleurPermissif;
  state   ctrl : bool;
          nbPanne : [0,3];
  event   commande, panne, time;
  trans   ctrl  |- commande -> ctrl := false;
          ~ctrl |- panne    -> ctrl := true,
                               nbPanne := nbPanne+1;
          ~ctrl |- time     -> ctrl := true;
  assert Cu.f1 = V1.debit;
    ...
  sync    <commande, Co.inc1, V1.inc>;
    ...
```

La synthèse (la spécification)

```
with SystemControle do
 dead      := any_s - src(any_t - self_epsilon);
 ER        := [Cu.niveau = 0] | [Cu.niveau = 4]
 nonControle := label panne | label time;
 controle    := rsrc([controle]) & rtgt([~contr

/* Les equations classiques de stratégies gagna
 Gagnant     = Gagne |& \\ | pour pppf et & pou
               src(CoupGagnant);
 CoupGagnant = coup & rtgt(Perdant);
 Perdant     = Perdu |& \\ | pour pppf et & pou
               (src(CoupPerdant) - src(coup - C
 CoupPerdant = coup & rtgt(Gagnant);
*/
```

La synthèse (la spécification)

```
// les actions de controle
Ctrl -= controle &
   rtgt(src(nonControle &
            rtgt((any_s - ER) & src(Ctrl))) -
        src(nonControle -
            rtgt((any_s - ER) & src(Ctrl))));
// Le système est-il controlable
Controlable := initial & src(Ctrl);
// les erreurs a ne pas faire
ErreurControle  := controle - Ctrl;
// enumeration des actions de controle
// afin de générer le controleur
wts(src(Ctrl), Ctrl) > '$NODENAME_Controleur';
```

La synthèse (la spécification)

```
  // sortie des resultats
  show(all)                        >  '$NODENAME.prop';
  test(ER,0)                       >  '$NODENAME.res';
  test(Controlable,1)              >> '$NODENAME.res';
  test(ErreurControle,0)           >> '$NODENAME.res';
done
```

La synthèse (le résultat)

```
TEST(ER=0) [FAILED] actual size = 839
TEST(Controlable=1) [PASSED]
TEST(ErreurControle=0) [FAILED] actual size = 4
```

La synthèse (le contrôleur P0)

```
/* this file generated by mec2alta.awk on jeu m

node Controleur_Out2_dddn_P0
  event nop, inc1, inc2, inc3, dec1, dec2, dec3
    inc1inc2, inc1inc3, inc1dec2, inc1dec3, inc
    inc2dec3, inc3dec1, inc3dec2, dec1dec2, dec
    inc1inc2inc3, inc1inc2dec3, inc1dec2inc3, i
    dec1inc2inc3, dec1inc2dec3, dec1dec2inc3, d
  flow
    d1 : [0,2];
    d2 : [0,2];
    d3 : [0,2];
    n : [0,4];
```

La synthèse (le contrôleur P0)

```
trans // Projection du controleur de :  Syste
  d1=0 & d2=0 & d3=0 & n=2 |-
    inc1inc2inc3,
    inc2inc3, inc1inc3 -> ;
  d1=1 & d2=0 & d3=1 & n=2 |-
    dec1inc2inc3, inc1inc2inc3,
    inc2inc3, inc1inc3,
    inc3 -> ;
  d1=1 & d2=0 & d3=1 & n=3 |-
    dec1inc2inc3,
    inc3dec1,
    inc3 -> ;
  d1=1 & d2=0 & d3=1 & n=1 |-
    inc1inc2inc3 -> ;
  ...
```

La synthèse (vérification)

```
TEST(any_s=0) [FAILED] actual size = 43
TEST(any_t=0) [FAILED] actual size = 148
TEST(P0=0) [PASSED]
TEST(P1=0) [PASSED]
TEST(P2=0) [PASSED]
TEST(P3=0) [PASSED]
```

Le rafinement

```
node CuveRaf
  flow  f1, f2, f3 : [0,2];
  state niveau : [0,4] : public;
  event time;
  trans
    (f1+f2=f3) |- time -> ;
    (f1+f2=f3) & (f3>0) |- time ->
                 niveau := niveau+1;
    //(f1+f2=f3) & (f3>0) |- time ->
      //           niveau := niveau-1;
    (f1+f2>f3) |- time -> niveau := niveau+1;
    (f1+f2<f3) |- time -> niveau := niveau-1;
  init  niveau := 2;
edon
```

La vérification du raffinement

```
TEST(any_s=0) [FAILED] actual size = 36
TEST(any_t=0) [FAILED] actual size = 125
TEST(P0=0) [PASSED]
TEST(P1=0) [PASSED]
TEST(P2=0) [PASSED]
TEST(P3=0) [PASSED]
```