Modélisation et vérification d'un évaporateur en UPPAAL *

Mohamed El Mongi BEN GAID[1], Béatrice BÉRARD[2], and Olivier DE SMET[3]

[1] COSI – ESIEE
Cité Descartes - BP 99 - 2 Bd Blaise Pascal, F-93162 Noisy-Le-Grand Cedex, France
`bengaidm@esiee.fr`
[2] LSV – CNRS UMR 8643 & ENS de Cachan,
61 av. du Prés. Wilson, F-94235 Cachan Cedex, France
`berard@lsv.ens-cachan.fr`
[3] LURPA – EA 1385 – ENS de Cachan,
61 av. du Prés. Wilson, F-94235 Cachan Cedex, France
`olivier.de_smet@lurpa.ens-cachan.fr`

Résumé Dans ce travail, nous nous intéressons aux aspects temporisés de la programmation des automates programmables industriels (API). La vérification des programmes d'API est un enjeu important pour les constructeurs, ce qui a déjà suscité de nombreux travaux. L'étude de propriétés quantitatives faisant intervenir le temps a été abordée plus récemment, avec des difficultés supplémentaires liées à l'introduction de temporisateurs et d'horloges. En utilisant l'exemple d'un mécanisme d'évaporation, nous proposons une modélisation du système contrôlé ainsi que du programme de contrôle écrit en *Ladder Diagram*, sous la forme d'un réseau d'automates temporisés, avec une hypothèse forte d'atomicité pour l'ensemble des instructions du programme. Nous montrons, sur cet exemple, comment cette hypothèse permet de réduire la taille du modèle et d'accélérer les calculs d'accessibilité. Les performances sont illustrées par la vérification de plusieurs propriétés de l'évaporateur avec l'outil UPPAAL.

Mots clés : Automates programmables industriels, automates temporisés, vérification par model-checking.

1 Introduction

Dans certaines applications critiques où se produisent des interactions avec l'environnement physique, des erreurs d'un module de contrôle-commande peuvent engendrer des défaillances graves du système. Ainsi, la vérification de propriétés de sûreté pour des programmes d'API (Automates Programmables Industriels) est maintenant reconnue comme un enjeu important pour les constructeurs et les utilisateurs. Le programme d'un API peut être écrit dans plusieurs langages spécifiés par la norme IEC-61131-3 [IEC93], ce qui ne facilite pas la phase de validation, mais de nombreux travaux ont déjà été consacrés à cet objectif dans le cadre non temporisé, par des méthodes à base d'automates [RdSLC$^+$00,DCR$^+$00], [CCL$^+$00,FL00].
Lorsqu'on considère des aspects quantitatifs liés au temps, la modélisation doit faire intervenir des mécanismes additionnels. Un des modèles qui a été largement étudié et utilisé avec succès depuis les années 1990 est celui des automates temporisés, introduits par [AD90,AD94]. La validation devient alors plus difficile, en raison de l'explosion combinatoire supplémentaire due au temps. L'étude approfondie de ces modèles a cependant conduit à des résultats positifs de décidabilité et a été suivie du développement d'outils de vérification comme HYTECH [HHW95], KRONOS [DOTY96] ou UPPAAL [LPY97]. Ces outils ont montré leur efficacité pour la vérification de très nombreuses études de cas en

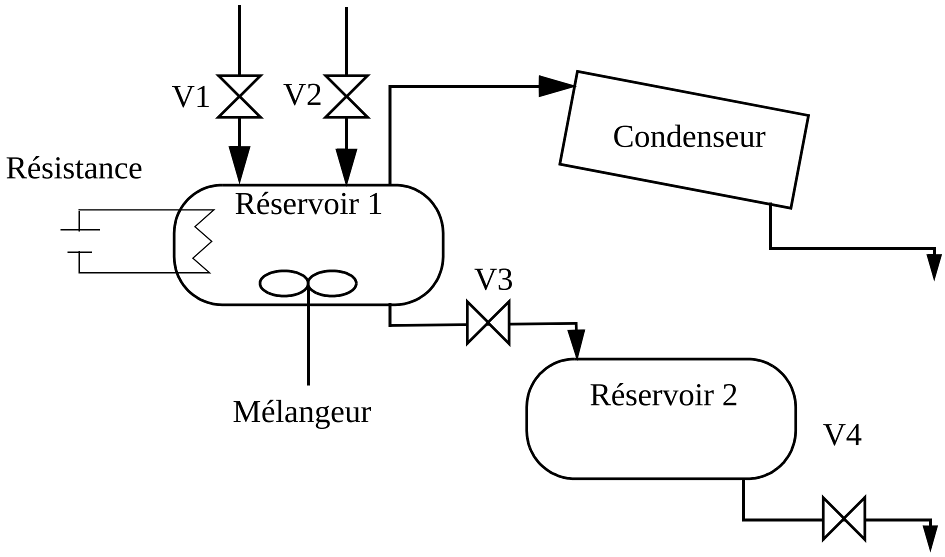
---

* Ce travail est issu du stage de DEA de M. Ben Gaid et a été soutenu par le Plan Pluri Formation VSMT de l'ENS de Cachan.

milieu industriels. Plus récemment, l'intérêt s'est porté sur les aspects temporisés des langages de programmation des API [Old99,MW99,Die00], par exemple avec l'utilisation de blocs fonctionnels permettant des temporisations [RS00].
L'exemple traité ici est une version modifiée de l'étude classique [KSB01,HLL01] de l'évaporateur. Un des aspects originaux de ce travail, par rapport à l'étude [HLL01] par exemple, est de modéliser, en plus du dispositif lui même, un programme écrit en Ladder Diagram pour le contrôle, comportant plusieurs instances de temporisateurs. Après une description du système (section 2), nous montrons comment les différents éléments de l'évaporateur, le programme et les temporisateurs sont modélisés par des automates temporisés au format d'Uppaal (section 3), de façon à en vérifier certaines propriétés (section 4). Les résultats obtenus sont performants, grâce aux hypothèses fortes d'atomicité concernant l'exécution du programme de contrôle.

2 Spécification de l'évaporateur

L'installation considérée se compose de deux réservoirs et d'un condenseur reliés par des conduites dont certaines sont munies de valves unidirectionnelles.



Les valves V1 et V2 assurent le remplissage du réservoir 1. La valve V3 permet de transférer le contenu du réservoir 1 dans le réservoir 2 tandis que la valve V4 permet de vider le réservoir 2. Le réservoir 1 est muni d'une résistance chauffante et d'un mélangeur. La résistance chauffante est utilisée afin de chauffer la solution présente dans le réservoir 1 jusqu'à l'ébullition. Le mélangeur homogénéise les produits versés dans le réservoir 1 par l'intermédiaire des valves V1 et V2. Le condenseur permet de condenser la vapeur dégagée lors de l'ébullition de la solution présente dans le réservoir 1. L'eau ainsi produite est évacuée du condenseur. Les réservoirs 1 et 2 sont munis de capteurs qui assurent la détection du niveau haut (réservoir plein) et du niveau bas (réservoir vide). L'API qui contrôle l'évaporateur comporte les entrées-sorties indiquées ci-dessous, où le suffixe *e* (comme environnement) indique que les signaux correspondants sont relatifs aux ports de l'API, et donc reliés respectivement aux capteurs et aux pré-actionneurs. Notons que les sorties sont considérées comme des commandes vers le dispositif physique.

Entrées		**Sorties**	
R1Pe	Réservoir 1 plein	V1e	Ouverture valve 1
R1Ve	Réservoir 1 vide	V2e	Ouverture valve 2
R2Pe	Réservoir 2 plein	V3e	Ouverture valve 3
R2Ve	Réservoir 2 vide	V4e	Ouverture valve 4
Malfe	dysfonctionnement condenseur	Rese	démarrage résistance
		MRe	démarrage mélangeur
		Alarme	actionnement alarme

Fonctionnement normal. Le réservoir 1 est d'abord rempli par l'intermédiaire des valves V1 et V2 par deux solutions de faible concentration. Quand le niveau haut du réservoir 1 est atteint, alors les valves V1 et V2 sont fermées et le mélangeur MR est actionné. Après 2 unités de temps, la résistance chauffante est mise en marche. Au bout d'un temps égal à 20 unités de temps, la concentration désirée est atteinte. La résistance chauffante est arrêtée et la valve V3 est ouverte afin de transférer le contenu du réservoir 1 dans le réservoir 2. Quand le réservoir 1 est vidé, le moteur MR est arrêté. Le mélange subit un traitement dans le réservoir 2. Ce traitement dure 32 unités de temps. Au bout de cette période, la valve V4 est ouverte afin de transférer le contenu du réservoir 2 à l'étape postérieure du processus de fabrication.

Traitement d'un dysfonctionnement du condenseur. Durant l'évaporation de la solution dans le réservoir 1, un dysfonctionnement du condenseur peut survenir. La vapeur ne pouvant plus être refroidie, la pression à l'intérieur du condenseur commence à augmenter. Il est ainsi nécessaire d'arrêter toute production de vapeur afin de ne pas dépasser une certaine pression limite qui causerait une explosion du condenseur. D'autre part, l'arrêt de la production de vapeur engendre un autre problème : si la température dans le réservoir 1 descend en dessous d'une certaine valeur (suite à l'arrêt de la production de vapeur), alors la solution présente dans le réservoir 1 se solidifie et ne peut plus s'écouler dans le réservoir 2. Il est donc nécessaire d'ouvrir la valve V3 suffisamment tôt afin de transférer la solution du réservoir 1 au réservoir 2 où le traitement subi l'empêche de se solidifier. Cependant, avant d'ouvrir la valve V3, il faut s'assurer que le réservoir 2 est vide, afin d'éviter son débordement. La valve V4 doit donc être ouverte en premier pour vider ce réservoir.

Propriétés à vérifier. Afin de garantir la sécurité de l'installation, le programme de contrôle-commande doit respecter certaines propriétés. Nous mentionnons d'abord des propriétés non temporisées.

P1 Afin d'éviter l'endommagement du réservoir 1 par la chaleur dégagée par la résistance chauffante, il faut vérifier que, chaque fois que la résistance chauffante est en marche, le réservoir 1 est plein.

P2 Durant l'étape de vaporisation, la vapeur doit uniquement se dégager vers le condenseur : chaque fois que la résistance chauffante est en marche, les valves V1, V2 et V3 sont fermées.

P3 Afin d'éviter d'avoir un flux de liquide incontrôlé, les valves d'entrée et de sortie d'un réservoir ne doivent jamais être ouvertes simultanément.

Dans le cas d'un dysfonctionnement du condenseur, des propriétés faisant intervenir des grandeurs quantitatives sur les temps de réponse du programme doivent être satisfaites. Ces propriétés sont déduites des caractéristiques physiques suivantes de l'installation :

- Dans le cas où un dysfonctionnement du condenseur se produit, alors le condenseur risque d'exploser si la production de vapeur se poursuit pendant une durée supérieure à 22 unités de temps depuis l'occurrence du dysfonctionnement.
- Si la résistance chauffante est mise à l'arrêt, alors la production de vapeur cesse après exactement 12 unités de temps.
- Si la production de vapeur s'arrête dans le réservoir 1, alors la solution risque de devenir solide après 19 unités de temps.
- Vider le réservoir 2 prend entre 0 et 26 unités de temps.
- Le vidage du réservoir 1 est très rapide relativement aux durées mises en jeu, il est ainsi supposé instantané.
- Remplir le réservoir 1 prend au plus 6 unités de temps.

On considère donc les propriétés négatives suivantes, qui doivent être évitées :

P4 Le condensateur explose
P5 La solution se solidifie dans le réservoir 1
P6 Le réservoir 2 déborde

Programme de contrôle. Il s'agit d'un programme écrit en Ladder Diagram [RD02]. Ce langage graphique est très utilisé en milieu industriel et s'inspire de la représentation des circuits de commande, basée sur la logique électrique. La norme IEC 61131-3 a défini le langage Ladder par un ensemble de règles mais n'a pas proposé une sémantique formelle. Un programme Ladder est décrit par une séquence d'échelons, comportant des symboles qui sont soit des contacts soit des bobines. Un contact permet la lecture d'une variable booléenne tandis qu'une bobine correspond à l'affectation d'une telle variable. Par exemple :

le programme

```
|    a     b     ST    |
+---| |---| |---(  )---+
|    c           MA    |
+---|/|---+-----(  )---+
|    d    |            |
+---| |---+            |
|    a            L    |
+---|/|----------(S)---|
|    b            P    |
+---|/|----------(R)---|
|                      |
```

réalise les opérations

```
ST := a and b;
MA := not(c) or d;
IF (a=0) THEN L:=1;
IF (b=0) THEN P:=0;
```

Pour le programme de contrôle de l'évaporateur, présenté sur la figure 1, la variable interne PrEx doit être initialisée à 1, les autres variables internes à zéro.

Notons d'autre part que ce programme comporte des temporisations, sous forme de blocs fonctionnels TON (Timer On-delay). Ce mécanisme, appelé également temporisateur d'enclenchement, possède

- deux entrées : une variable booléenne IN, qui permet de lancer et d'annuler la temporisation, et un paramètre de temps PT (Preset Time) qui spécifie la durée de temporisation,
- et deux sorties : une variable booléenne Q, qui vaut 1 si le délai de temporisation a expiré et une variable de temps ET (Elapsed Time) qui indique le temps écoulé depuis le lancement de la temporisation.

La norme décrit un comportement possible du bloc TON par le chronogramme de la figure 2. La temporisation est initialisée lorsque IN passe de 0 à 1. La sortie ET augmente alors jusqu'à atteindre la valeur de PT. À ce moment, la sortie Q est mise à 1. Quand IN passe à 0, les sorties ET et Q sont immédiatement mises à 0. Si la temporisation est abandonnée (IN passe de 1 à 0 avant que le délai PT ne soit atteint), alors ET est immédiatement remise à 0.

```
|   V1    V2    V3    V4    R1V   R2V    INIT_OK  |
+---|/|---|/|---|/|---|/|---| |---| |------( )----+
|   PrEx INIT_OK     Alrm                         |
+---| |---|/|--------->>                          |
|   malf              Malfun                       |
+---| |--------------->>                          |
|   malf  PrEx INIT_OK                     V1     |
+---|/|---| |---| |-------------------+----(S)----+
|                                     |    V2     |
|                                     +----(S)----+
|                                     |    PrEx   |
|                                     +----(R)----+
|   V1    V2    R1P                        V1     |
+---| |---| |---| |-------------------+----(R)----+
|                                     |    V2     |
|                                     +----(R)----+
|                                     |   ITEMP1  |
|                                     +----(S)----+
|                                     |     MR    |
|                                     +----(S)----+
|                                                 |
|                           +-------+             |
|  ITEMP1                   |       |      Res    |
+---| |---------------------+ TON1  +------(S)----+
|                           |       |             |
|                           +-------+             |
|                                                 |
|                           +-------+             |
|  ITEMP1                   |       |      Res    |
+---| |---------------------+ TON2  +-+----(R)----+
|                           |       | |   ITEMP1  |
|                           +-------+ +----(R)----+
|                                     |   FinTR1  |
|                                     +----(S)----+
|  FinTR1                                  V3     |
+---| |-------------------------------+----(S)----+
|                                     |    MR     |
|                                     +----(R)----+
| FinTR1  V3    R1V                      FinTR1   |
+---| |---| |---| |-------------------+----(R)----+
|                                     |    V3     |
|                                     +----(R)----+
|                                     |   ITEMP2  |
|                                     +----(S)----+
|                                                 |
```

```
|                           +-------+             |
|  ITEMP2                   |       |      V4     |
+---| |---------------------+ TON3  +------(S)----+
|                           |       |             |
|                           +-------+             |
|                                                 |
|                                                 |
|                                                 |
|                                                 |
|   V4    R2V                              V4     |
+---| |---| |-------------------------+----(R)----+
|                                     |   ITEMP2  |
|                                     +----(R)----+
|                                     |    PrEx   |
|                                     +----(S)----+
|                                                 |
+-----------------------<RETURN>                  |
|                                                 |
Malfun                                            |
|                  V1    V2    V3    V4    MR     |
+------------------(R)---(R)---(R)---(S)---(R)----+
|                                                 |
|                           +-------+             |
|                           |       |      Res    |
+---------------------------+ TON4  +------(S)----+
|                           |       |             |
|                           +-------+             |
|   R2V   V4                               V3     |
+---| |---| |-------------------------+----(S)----+
|                                     |    V4     |
|                                     +----(R)----+
|   R1V                                    V3     |
+---| |------------------------------------(R)----+
|                                                 |
Alrm                                      Alarm   |
+------------------------------------------( )----+
|                                                 |
```

Fig. 1. Programme de contrôle

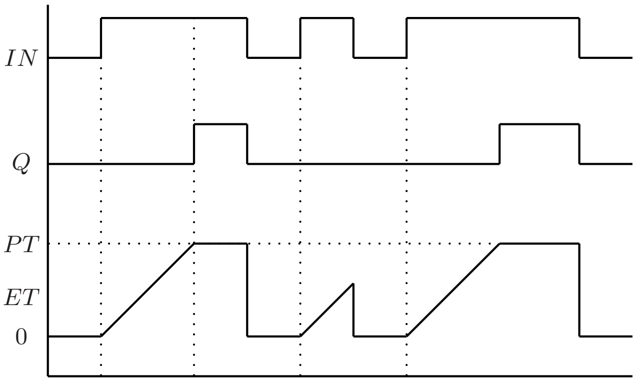


Fig. 2. Chronogramme d'un bloc TON

Notons enfin que seules les entrées et les sorties booléennes des temporisateurs TON ont été représentées dans le programme de la figure 1. En effet, la norme IEC 61131-3 n'oblige pas à connecter toutes les entrées et les sorties des blocs fonctionnels. L'affectation des paramètres *PT* des différentes instances du bloc fonctionnel TON sont réalisées dans la partie "déclarations" du programme (non représentée sur la figure 1).

3 Modélisation avec UPPAAL

3.1 L'outil UPPAAL

Nous donnons rapidement les principales caractéristiques d'UPPAAL (voir [DBLY03a] pour une présentation des plus récents développements), qui seront illustrées dans les paragraphes suivants.

Pour UPPAAL, un modèle consiste en un ensemble d'automates temporisés, qui communiquent par une synchronisation binaire, utilisant des canaux et une syntaxe du type émission/réception. Ainsi, sur le canal `c`, un émetteur envoie le signal `c!` et un récepteur se synchronise avec lui par le signal complémentaire `c?`. Les automates temporisés d'UPPAAL sont une variante des modèles originaux de [AD94]. Ce sont des structures finies manipulant deux types de variables : des horloges, qui évoluent de manière synchrone avec le temps, et des variables entières discrètes bornées. Un état de l'automate peut comporter une condition sur les horloges, appelée *invariant*, qui doit être satisfaite pendant toute la durée passée dans cet état. Une *transition* de l'automate est étiquetée par :

- une *garde*, qui exprime une condition sur les valeurs des variables (`true` par défaut). Cette condition doit généralement être compatible avec l'invariant de l'état origine de la transition et elle doit être satisfaite pour franchir la transition,
- une *synchronisation* de la forme `c!` ou `c?`, l'absence de synchronisation indiquant une action interne de l'automate,
- une remise à zéro de certaines horloges et une mise à jour de certaines variables entières.

Une *configuration* (globale) du système est une paire (l, v) où l (pour *location*) est un vecteur indiquant l'état de chacun des automates et v est une valuation des variables. Une exécution est une suite de configuration qui part d'un état initial de chacun des automates avec toutes les valeurs des variables à zéro. La sémantique de ce modèle contient trois types de changements de configurations (ou mouvements) :

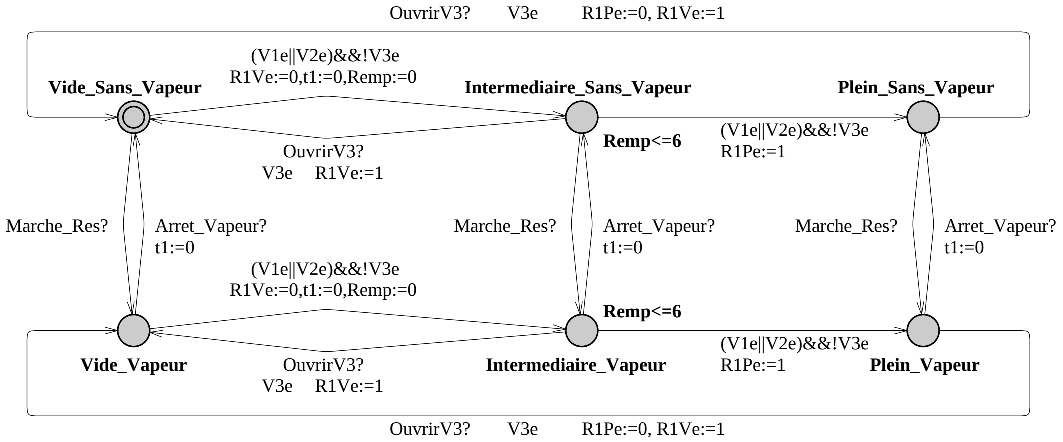
- **Délai :** le temps peut progresser dans les états courants d'une durée d, à condition que l'invariant de chacun de ces états reste satisfait. Les valeurs des horloges augmentent de d et les valeurs des variables discrètes ne changent pas.
- **Synchronisation :** Si deux actions complémentaires de deux composants sont possibles, et que les gardes associées aux transitions sont satisfaites, ces deux composants se synchronisent. Les états correspondants sont modifiés, et les valeurs des horloges et des variables discrètes sont modifiées selon les indications de remises à zéro et de mises à jour.
- **Action interne :** Si une action interne (sans synchronisation) d'un composant est possible (sa garde est satisfaite), cette action peut être exécutée indépendamment des autres composants : l'état et les variables du composant sont modifiés comme dans le cas précédent.

Pour notre étude, le modèle global résulte du produit d'automates temporisés de trois types : les automates de l'environnement, l'automate du programme lui-même et les automates représentant les temporisateurs. La communication entre ces automates est une synchronisation binaire, comme décrite ci-dessus. Par exemple, la synchronisation entre l'automate modélisant le réservoir 1, l'automate modélisant la résistance et l'automate du programme est réalisée au moyen des canaux de communication `Arret_Vapeur` et `Marche_Res`.

3.2 Modélisation de l'environnement

L'environnement comporte un automate temporisé pour chaque élément du dispositif.

Le réservoir 1. Il est modélisé par l'automate ci-dessous.

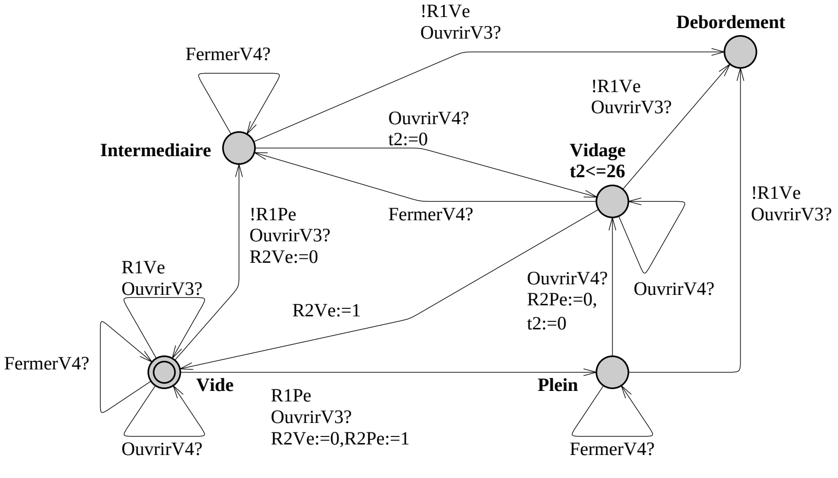


Les 6 états caractérisent :

- le niveau d'eau dans le réservoir : `Vide`, `Plein` ou `Intermediaire` (c'est-à-dire ni vide ni plein).
- la présence ou non de vapeur dans le réservoir (`Vapeur`, `Sans_Vapeur`). Une horloge *Remp* est utilisée afin de pouvoir mesurer la durée de remplissage du réservoir. L'invariant $Remp \leq 6$ appliqué aux états `Intermediaire_Vapeur` et `Intermediaire_Sans_Vapeur` modélise le fait que le temps de remplissage ne peut dépasser 6 unités de temps.

La vidange du réservoir est instantanée. Elle est exécutée instantanément lors de la réception de l'évènement `OuvrirV3`(cet évènement étant envoyé par l'automate du programme). L'horloge $t1$ est utilisée afin de mesurer la durée passée dans un état où le réservoir contient du liquide et où la vapeur est arrêtée (états `Intermediaire_Sans_Vapeur` et `Plein_Sans_Vapeur`). L'automate ne doit en aucun cas passer plus de 18 unités de temps dans l'un de ces deux états, sinon le liquide risque de se solidifier. L'automate permet enfin d'affecter les valeurs adéquates aux variables d'environnement R1Pe et R1Ve.

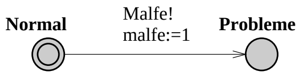
Le réservoir 2. Il est modélisé par un l'automate temporisé suivant à 5 états : `Vide`, `Plein`, `Intermediaire`, `Vidage` et `Debordement`.



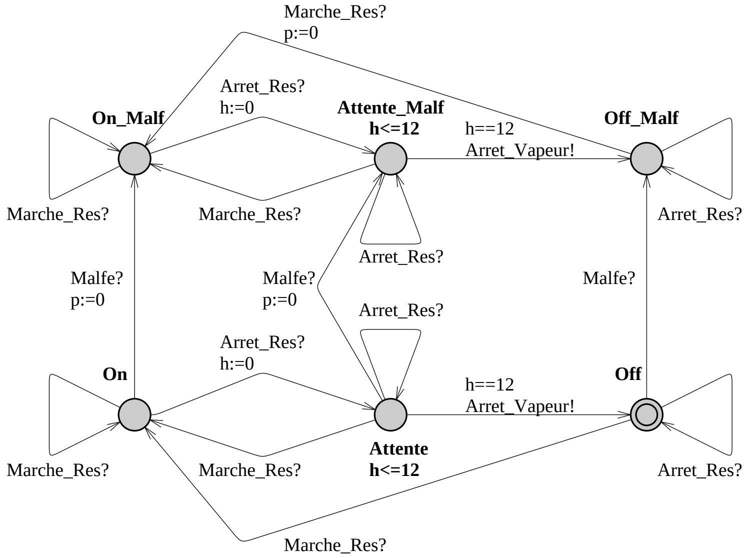
Le remplissage du réservoir 2 est instantané. Il est réalisé lors de la réception de l'évènement OuvrirV3 ! envoyé par l'automate du programme. Initialement, le réservoir 2 est dans l'état

`Vide`. L'ouverture de la valve V3 le fait passer à l'état `Plein` si le réservoir 1 est plein et à l'état `Intermediaire` si le réservoir 1 n'est ni vide, ni plein. Contrairement au remplissage, le vidage du réservoir 2 prend un temps au plus égal à 26 unités. L'ouverture de la valve V4 dans l'état `Plein` ou dans l'état `Intermediaire` fait passer instantanément l'automate à l'état `Vidage`. La fermeture de la valve V4 dans l'état `Vidage` fait passer instantanément l'automate à l'état `Intermediaire`. Si le réservoir 2 n'est pas vide, le réservoir 1 n'est pas vide et la valve V3 est ouverte, alors l'automate passe immédiatement à l'état `Debordement`.

Le condenseur. Un modèle très simple a été retenu pour le condenseur. Initialement dans l'état `Normal`, il passe à l'état `Probleme` si un défaut se produit, en envoyant le signal Malfe ! et en mettant à 1 la variable d'environnement malfe.



La résistance chauffante. La résistance est modélisée par l'automate ci-dessous.



Les états `On`, `Attente` et `Off` représentent l'état de la résistance chauffante lorsque le condenseur fonctionne normalement tandis que les états suffixés par `_Malf` représentent l'état de la résistance chauffante lorsqu'un dysfonctionnement se produit dans le condenseur. L'état de la résistance chauffante et la présence de vapeur dans le réservoir 1. Les états `On` et `On_Malf` représentent la résistance chauffante en état de marche, les états `Attente` et `Attente_Malf` représentent la résistance chauffante à l'arrêt, la vapeur continuant encore à se dégager. Enfin, les états `Off` et `Off_Malf` correspondent à un état d'arrêt de la résistance chauffante, avec une absence de dégagement de vapeur dans le réservoir 1. Les ordres de mise en marche et d'arrêt sont reçus grâce aux canaux de communication `Marche_Res` et `Arret_Res`. Ces ordres sont envoyés par l'automate du programme.

Pour la vérification de la propriété **P4**, une horloge p est utilisée afin de mesurer la durée passée dans un état où la vapeur se dégage encore et où le condenseur est dans un état de dysfonctionnement (états `On_Malf` et `Attente_Malf`).

3.3 Modélisation du programme

L'exécution d'un programme d'API est un cycle comprenant une phase d'acquisition des entrées, une phase de traitement et une phase d'affectation des sorties, avec une durée de cycle comprise entre deux paramètres ε_1 et ε_2.

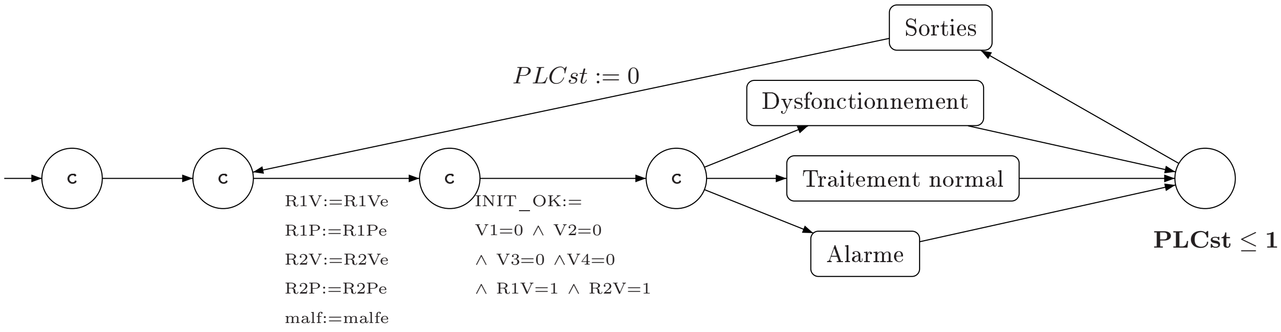


Fig. 3. Structure de l'automate du programme

```
|                           +-------+              |
|  ITEMP2                   |       |      V4      |
+---| |---------------------+  TON  +------(S)-----+
|                           |       |              |
|                           +-------+              |
|                                                  |
|                                                  |
|                                                  |
|                                                  |
|   V4    R2V                              V4      |
+---| |---| |-------------------------+----(R)-----+
|                                     |   ITEMP2   |
|                                     +----(R)-----+
|                                     |    PrEx    |
|                                     +----(S)-----+
|                                                  |
+-----------------------<RETURN>                   |
|                                                  |
```

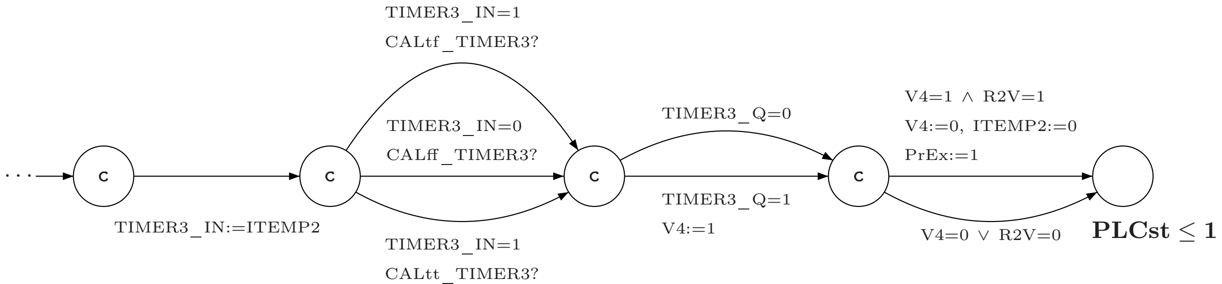


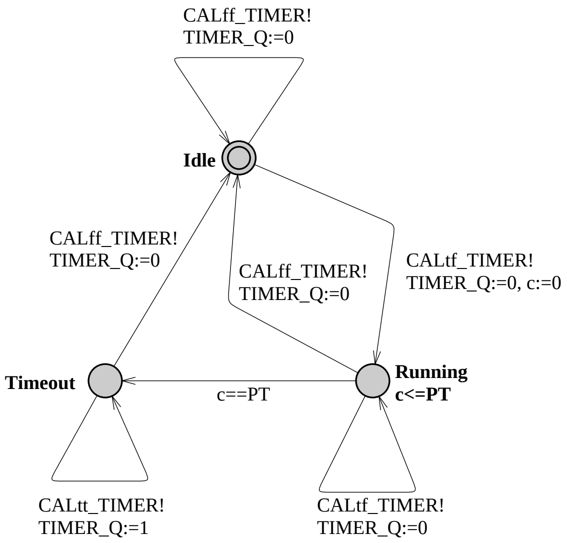
Fig. 4. Quelques échelons du programme

Dans notre modélisation (figure 3), la structure globale de l'automate du programme est une boucle, dans laquelle les échelons sont traduits successivement. L'acquisition des entrées et l'exécution du code Ladder Diagram sont instantanées. Ceci se traduit en Uppaal par le fait que tous les états de l'automate (sauf un) sont *urgents*, c'est-à-dire qu'aucun

écoulement du temps n'y est autorisé : ils sont étiquetés par `c` (pour *committed*). Les durées minimale et maximales du cycle sont prises ici respectivement à $\varepsilon_1 = 0$ et $\varepsilon_2 = 1$. Pour représenter la durée du cycle, nous utilisons une horloge $PLCsystime$ (abrégée en $PLCst$ dans les figures), initialisée à 0, et dont la valeur en fin de cycle doit être inférieure ou égale à 1. Le temps s'écoule donc dans le dernier état du cycle, qui est muni de l'invariant $PLCst \leq 1$. Les variables de sortie (suffixées par e) et leurs affectations sont explicitement modélisées, dans une phase qui suit l'exécution proprement dite. Suivant les valeurs de certaines variables, des canaux de communication (`OuvrirV3`, `OuvrirV4`, `FermerV4`, `Marche_Res` et `Arret_Res`) sont utilisés dans cette phase d'affectation des sorties afin de forcer l'environnement à réagir aussitôt que les variables d'environnement sont changées. La dernière transition (retour à l'état initial) remet enfin $PLCst$ à 0.
À titre d'exemple, nous montrons dans la figure 4 la partie de cet automate qui correspond aux deux derniers échelons du traitement normal. L'état le plus à droite est donc le dernier état du cycle. Notons dans cette figure la communication entre le programme et le temporisateur appelé TIMER3, qui est réalisée par trois canaux.

3.4 Modélisation d'un temporisateur

Nous montrons maintenant comment modéliser par un automate temporisé chacune des quatre instances du bloc fonctionnel TON dans le programme. Le modèle est l'automate TIMER ci-dessous, qui comporte 3 états : `Idle`, où le temporisateur est inactif, `Running` où le déclenchement a eu lieu, et `Timeout`, lorsque le délai est atteint.



Une horloge c permet de mesurer la durée de temporisation : l'automate doit passer de l'état `Running` à l'état `Timeout` quand c atteint la durée spécifiée $TIMER\_PT$. Les transitions sont étiquetées par des actions de la forme CALxy_TIMER où $x, y \in \{t, f\}$ désignent respectivement les valeurs (vrai ou faux) du paramètre TIMER_IN communiqué au bloc fonctionnel lors de l'appel et la valeur du paramètre TIMER_Q mise à jour après l'appel. Ces deux variables sont donc ajoutées à celles du programme, avec des valeurs initiales $f =$ `false`. La sémantique des transitions est la suivante :

- CALff_TIMER représente l'abandon (ou la fin) de la temporisation. On a donc une telle transition depuis chaque état vers l'état `Idle`.

- CALtf_TIMER a pour état d'arrivée `Running`, puisqu'elle correspond à une temporisation délenchée, pour laquelle de délai n'est pas atteint.
- CALtt_TIMER est l'étiquette d'une transition de l'état `Timeout` dans lui-même, qui correspond à la valeur 1 de la sortie TIMER_Q.

Notons enfin que la sortie $TIMER\_ET$ n'est pas représentée dans ce modèle car elle n'est pas utilisée par le programme. Cependant, quand le temporisateur est dans l'état `Running`, sa valeur est égale à celle de l'horloge c.

4 Vérification avec UPPAAL

Les propriétés énoncées dans la spécification sont formalisées ci-dessous dans la syntaxe d'UPPAAL. Dans les formules, `A` est le quantificateur universel de chemin, qui se lit *sur tout chemin...*, et `[ ]` est la modalité *partout*. La combinaison `A[ ]` signifie donc *pour tout état dans le futur*. De plus, `M.s` désigne l'état `s` de l'automate `M`.

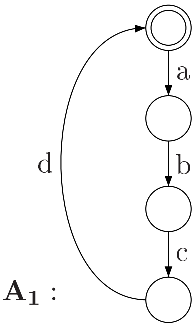
P1 : `A[ ]` (Rese imply R1Pe)
P2 : `A[ ]` (Rese imply (not(V1e) and not(V2e) and not(V3e))
P3 : `A[ ]` not(((V1e or V2e) and V3e) or (V3e and V4e))
P4 : `A[ ]` not((`Resistance.On_Malf` or `Resistance.Attente_Malf`) and p>21)
P5 : `A[ ]` not((`Reservoir1.Plein_sans_Vapeur` or `Reservoir1.Intermediaire_sans_Vapeur`) and t1>18)
P6 : `A[ ]` not(`Reservoir2.Debordement`)

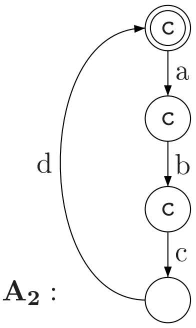
Le modèle traité ici comporte 37 variables booléennes, 19 canaux de communication et 10 horloges, avec 9 processus (cycliques) en parallèle et un nombre total d'états voisin d'un million. À titre de comparaison, la version de [HLL01] produit environ 750 états, avec un programme de contrôle non cyclique et aucun bloc fonctionnel de temporisations.
Les mesures ont été réalisées en utilisant la version 3.3.36 d'UPPAAL de juin 2003. Les meilleurs résultats expérimentaux ont été obtenus avec une recherche en largeur d'abord en utilisant une représentation par DBM (Difference Bounded Matrices). Ils sont résumés dans le tableau suivant (et obtenus avec l'utilitaire `memtime`) :

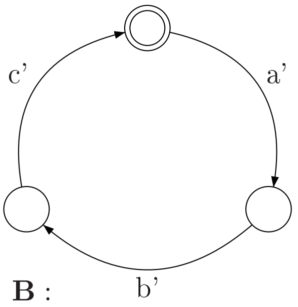
Propriété	Résultat	Temps (sec)	Mémoire (Mo)
P1	oui	4.56	4.02
P2	oui	4.58	4.02
P3	oui	4.56	4.02
P4	oui	16.37	4.71
P5	oui	13.16	4.71
P6	oui	4.48	4.02

Les performances sont dues à deux principaux facteurs : l'hypothèse d'atomicité et le choix de $\varepsilon_1 = 0$.

Hypothèse d'atomicité. La traduction du programme suivant cette hypothèse fait en sorte que la séquence de la phase d'affectation des sorties du cycle $n-1$ et des phases d'acquisition des entrées et d'exécution du programme du cycle n s'exécutent de façon atomique, ce qui est raisonnable dans un environnement d'exécution monotâche. Rappelons que l'écoulement du temps n'est autorisé que dans un seul état (l'état qui ne contient pas l'étiquette c dans l'automate de la figure 3), situé entre la phase d'exécution du programme et la phase d'affectation des sorties du cycle n. La présence d'au moins un état où le temps peut s'écouler est nécessaire pour n'importe quel modèle temporisé d'API. En effet, sans un tel état, le programme s'exécuterait une infinité de fois sans progression du temps (modèle Zénon), et les propriétés temporisées ne pourraient plus être vérifiées. Afin d'illustrer le gain apporté par cette hypothèse en termes de performances de vérification, considérons les automates A_1, A_2 et B suivants, A_1 et A_2 étant des modèles simplifiés du programme et B un modèle simplifié de l'environnement. Dans A_2 les instructions a, b et c s'exécutent de façon atomique.







Les produits synchronisés (nécessaires pour l'analyse d'accessibilité) $A_1 \otimes B$ et $A_2 \otimes B$ sont illustrés à la figure 5.

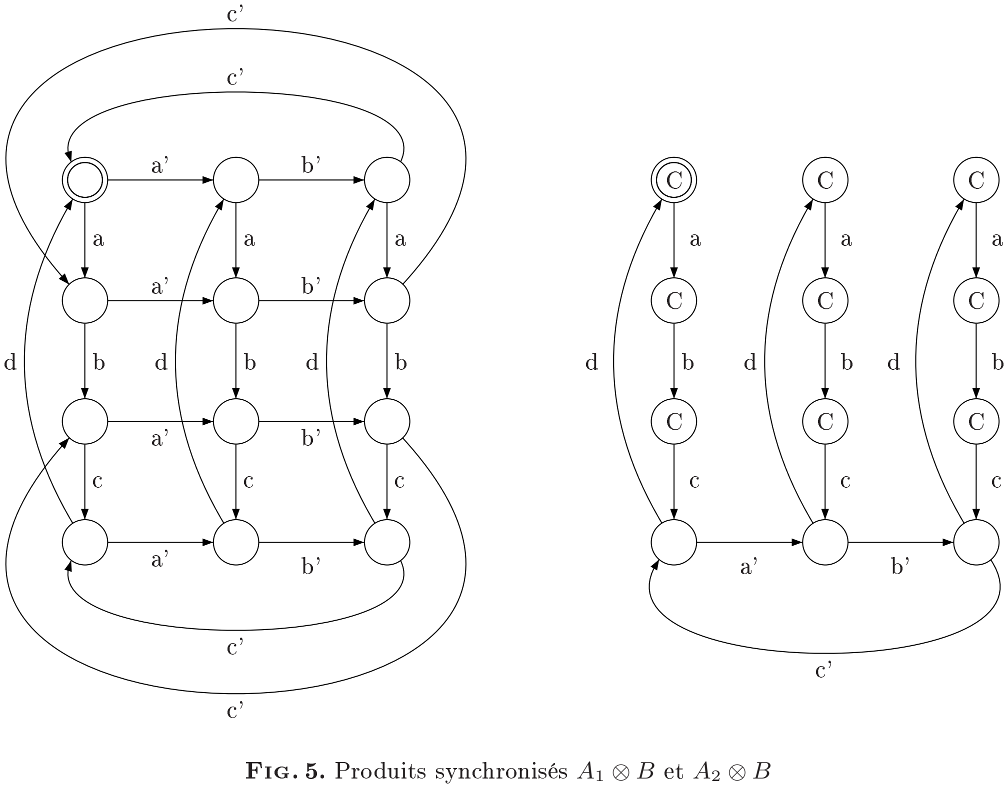
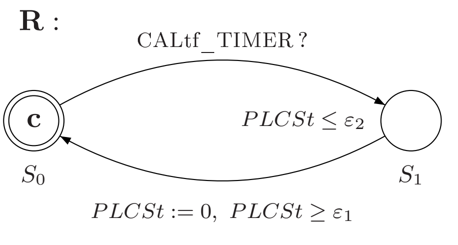


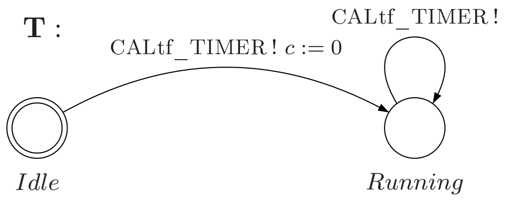
Fig. 5. Produits synchronisés $A_1 \otimes B$ et $A_2 \otimes B$

La taille du produit $A_2 \otimes B$ est sensiblement réduite par rapport à $A_1 \otimes B$ car l'hypothèse d'atomicité élimine toutes les séquences d'actions obtenues par entrelacement des actions représentant les instructions du programme avec les actions des autres automates (changement d'une variable d'environnement par exemple). L'automate sur lequel la vérification est réalisée est donc considérablement réduit, car dans la pratique, l'automate du programme a une taille nettement plus grande que les autres automates. D'autre part, l'atomicité implique le non écoulement du temps dans les états de contrôle urgents (étiquetés par `c`). Ainsi, les calculs sur les contraintes d'horloge ne sont pas effectués dans ces états, qui représentent la partie la pus importante du produit synchronisé d'où le gain en termes de temps de calcul et d'espace mémoire.

***Hypothèse* $\varepsilon_1 = 0$.** Cette hypothèse permet de borner l'espace mémoire engendré par la prise en compte de façon explicite du mécanisme de temporisation. En effet, Uppaal (comme les autres outils de model-checking temporisé) utilise un algorithme d'exploration de l'espace d'état basé sur la notion de zone (voir [CGP99] par exemple). Une zone est un ensemble de valuations définie par une contrainte sur les horloges de la forme $\phi ::= x \sim c | x - y \sim c | \phi \wedge \phi$ avec $\sim \in \{<, >, =, \leq, \geq\}$. Les trois principales opérations réalisées sur les zones sont l'intersection, la projection (remise à zéro d'horloges) et le futur (écoulement du temps).

Dans Uppaal, l'espace d'état est représenté par des triplets (l, v, Z) appelés états symboliques où l est un vecteur indiquant l'état de chacun des automates, v est une valuation des variables discrètes et Z une zone. Durant l'exploration de l'espace d'état, il est nécessaire (pour des raison d'efficacité et de terminaison) de garder en mémoire les états symboliques qui ont été explorés ainsi que ceux qui doivent être explorés. Dans les premières versions [BBD$^+$02] d'Uppaal, deux structures de données ont été utilisées pour mémoriser ces états, la **passed list** (qui contient les états symboliques déjà explorés) et la **waiting list** (qui contient les états symboliques encore à explorer). À partir de la version 3.3.24, une nouvelle approche est proposée, qui unifie ces listes dans une même structure : la **unified list** [DBLY03b], avec ajout d'un marquage pour distinguer les états qui doivent être explorés de ceux déjà explorés. Les apports de cette nouvelle approche se basent sur la notion de couverture. On dit que l'état symbolique (l, v, Z) couvre l'état symbolique (l, v, Z') si $Z' \subseteq Z$. Une optimisation très intéressante apportée par cette nouvelle approche fait en sorte qu'en ajoutant un état dans la **unified list** tous les états qu'il couvre sont supprimés. Le fait de mettre ε_1 à 0 permet d'exploiter cette dernière optimisation et de borner la taille de l'espace mémoire quand $PT/\varepsilon_2 \to +\infty$ (pour le paramètre PT d'un temporisateur). Pour illustrer ce dernier point, considérons les automates R et T suivants :

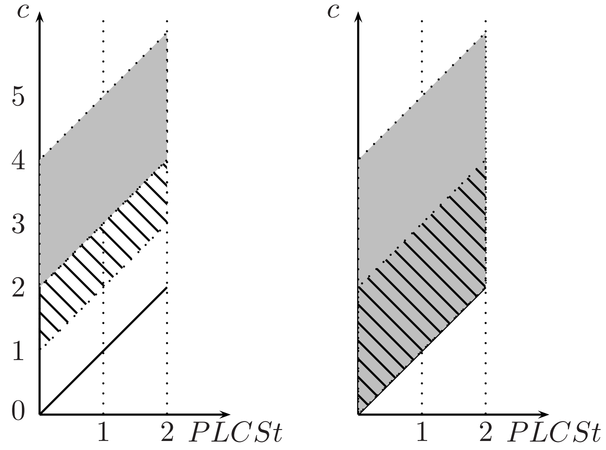




Le calcul des premières contraintes d'horloges avec $\varepsilon_1 \neq 0$ et avec $\varepsilon_1 = 0$ est illustré ci-dessous. Dans les deux cas (ε_1 nul ou non), les deux premières contraintes définissent les zones suivantes :

$Z_0 = Z'_0$ est le point défini par $PLCSt = 0 \ \wedge \ c = 0$

$Z_1 = Z'_1$ est le segment (diagonal) défini par $PLCSt = c \ \wedge \ 0 \leq PLCSt \leq 2$



Ensuite, dans la partie gauche (pour $\varepsilon_1 \neq 0$), on obtient :
pour Z_{2i} le segment défini par $PLCSt = 0 \ \wedge \ i \leq c \leq 2i$
pour Z_{2i+1} le segment défini par $0 \leq PLCSt \leq 2 \ \wedge \ i \leq c - PLCSt \leq 2i$
ainsi, Z_3 est la partie hachurée tandis que Z_5 est la partie grisée
tandis que dans la partie droite (pour $\varepsilon_1 = 0$), on obtient :
pour Z'_{2i} le segment défini par $PLCSt = 0 \ \wedge \ 0 \leq c \leq 2i$
pour Z'_{2i+1} le segment défini par $0 \leq PLCSt \leq 2 \ \wedge \ 0 \leq c - PLCSt \leq 2i$
ainsi, Z'_3 est la partie hachurée tandis que Z'_5 est la partie grisée
Quand $\varepsilon_1 \neq 0$, tous les états symboliques $((S_1, Running), v, Z_{2i+1})$ et $((S_0, Running), v, Z_{2i})$ avec $i \geq 1$ doivent être sauvegardés car les conditions de couverture ne sont pas vérifiées ce qui entraine l'explosion de la taille mémoire. Par contre, quand $\varepsilon_1 = 0$, puisque $Z'_{2k} \subseteq Z'_{2i}$ et $Z'_{2k+1} \subseteq Z'_{2i+1}$ pour $i \geq 1$ et $0 \leq k \leq i$, il n'est nécessaire de garder en mémoire que les derniers états $((S_0, Running), v, Z'_{2i})$ et $((S_1, Running), v, Z'_{2i+1})$ explorés.

5 Conclusion

Nous présentons un travail de modélisation et de vérification d'un système d'évaporateur avec l'outil Uppaal. L'originalité de cette étude réside dans le choix d'un programme de contrôle écrit en Ladder Diagram, un des langages de la norme IEC-61131-3, qui introduit explicitement des temporisations avec plusieurs instances du bloc fonctionnel TON. La modélisation utilise des automates temporisés pour les éléments physiques du dispositif, ainsi que pour le contrôle et les temporisateurs. Ces automates communiquent entre eux par des canaux. Pour la vérification, nous avons fait plusieurs hypothèses qui simplifient le déroulement de l'algorithme de calcul des états accessibles dans Uppaal, ce qui permet d'obtenir les propriétés souhaitées avec des performances raisonnables, compte tenu de la taille du modèle. Bien que la traduction du programme ait été réalisée manuellement, la traduction automatique semble être parfaitement faisable (une traduction automatique du Ladder vers SMV a été déjà réalisée dans [dSR02]).

Références

[AD90] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proc. 17th Int. Coll. Automata, Languages, and Programming (ICALP'90), Warwick University, England, July 1990*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[AD94] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science (TCS)*, 126(2) :183–235, 1994.

[BBD$^+$02] Gerd Behrmann, Johan Bengtsson, Alexandre David, Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL implementation secrets. In *Proc. of 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002.

[CCL$^+$00] G. Canet, S. Couffin, J.-J. Lesage, A. Petit, and Ph. Schnoebelen. Towards the automatic verification of PLC programs written in Instruction List. In *Proc. IEEE Int. Conf. Systems, Man and Cybernetics (SMC'2000), Nashville, TN, USA, Oct. 2000*, pages 2449–2454, 2000.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[DBLY03a] Alexandre David, Gerd Behrmann, Kim Guldstrand Larsen, and Wang Yi. A Tool Architecture for the Next Generation of UPPAAL. Technical Report 2003-011, Department of Information Technology, Uppsala University, Feb. 2003. 20 pages.

[DBLY03b] Alexandre David, Gerd Behrmann, Kim Guldstrand Larsen, and Wang Yi. Unification & sharing in timed automata verification. In *SPIN Workshop 03*, LNCS, 2003.

[DCR$^+$00] O. De Smet, S. Couffin, O. Rossi, G. Canet, J.-J. Lesage, Ph. Schnoebelen, and H. Papini. Safe programming of PLC using formal verification methods. In *Proc. 4th Int. PLCopen Conf. on Industrial Control Programming (ICP'2000), Utrecht, The Netherlands, Oct. 2000*, pages 73–78. PLCOpen, Zaltbommel, The Netherlands, 2000.

[Die00] H. Dierks. PLC-Automata : A New Class of Implementable Real-Time Automata. *Theoretical Computer Science*, 253(1) :61–93, 2000.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Proc. Workshop Hybrid Systems III : Verification and Control, New Brunswick, NJ, USA, Oct. 1995*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1996.

[dSR02] O. de Smet and O. Rossi. Verification of a controller for a flexible manufacturing line written in ladder diagram via model-checking. In *American Control Conference, ACC'02*, pages 4147–4152, May 2002.

[FL00] G. Frey and L. Litz. Formal methods in PLC-programming. In *Proc. IEEE Int. Conf. Systems, Man and Cybernetics (SMC'2000), Nashville, TN, USA, Oct. 2000*, pages 2431–2436, 2000.

[HHW95] T. A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. A user guide to HyTech. In *Proc. 1st Int. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95), Aarhus, DK, May 1995*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer, 1995.

[HLL01] R. Huuck, B. Lukoschus, and Y. Lakhnech. Verifying Untimed and Timed Aspects of the Experimental Batch Plant. *European Journal of Control*, 7(4) :400–415, 2001.

[IEC93] IEC (International Electrotechnical Commission). *IEC Standard 61131-3 : Programmable controllers - Part 3*, 1993.

[KSB01] S. Kowalewski, O. Stursberg, and N. Bauer. An Experimental Batch Plant as a Test Case for the Verification of Hybrid Systems. *European Journal of Control*, 7(4), 2001.

[LPY97] K. G. Larsen, P. Pettersson, and Wang Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2) :134–152, 1997.

[MW99] A. Mader and H. Wupper. Timed automaton models for simple programmable logic controllers. In *Proc. 11th Euromicro Conference on Real-Time Systems (ECRTS'99), York, UK, June 1999*, pages 114–122. IEEE Comp. Soc. Press, 1999.

[Old99] E.R. Olderog. Correct real-time software for programmable logic controllers. In *Correct System Design. Recent Insights and Advances*, volume 1710 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 1999.

[RD02] J.-M. Roussel and B. Denis. Safety properties verification of ladder diagram programs. *Journal Européen des Systèmes Automatisés*, 7(36) :905–917, 2002.

[RdSLC$^+$00] O. Rossi, O. de Smet, S. Lampérière-Couffin, J.-J. Lesage, H. Papini, and H. Guennec. Formal verification : a tool to improve the safety of control systems. In *4th Symposium on Fault Detection, Supervision and Safety for Technical Processes (IFAC Safeprocess 2000), Budapest, Hungary*, pages 885–890, 2000.

[RS00] O. Rossi and Ph. Schnoebelen. Formal modeling of timed function blocks for the automatic verification of Ladder Diagram programs. In *Proc. 4th Int. Conf. Automation of Mixed Processes : Hybrid Dynamic Systems (ADPM'2000), Dortmund, Germany, Sept. 2000*, pages 177–182. Shaker Verlag, Aachen, Germany, 2000.