0.1 Le Résolveur

Le résolveur est le module du simulateur qui assure la résolution des équations sur horloges et signaux booléens. Il accepte des équations sous forme partiellement résolue et guide leur résolution à travers un dialogue itératif avec l'utilisateur. Pendant ce dialogue la solution est choisie variable par variable. L'instanciation d'une variable entraine de nouvelles contraintes sur les variables restantes. Un mode automatique est également prévu, basé sur le même algorithme où le choix de l'utilisateur est remplacé par un tirage aléatoire pour chaque variable à déterminer.

0.1.1 Les équations

Une équation partiellement résolue est une liste de fonctions polynomiales

$$\begin{aligned} x_k &= f_1(x_{k+1}, ..., x_n, p_1, ..., p_m) \\ &... \\ x_1 &= f_1(x_2, ..., x_n, p_1, ..., p_m) \end{aligned}$$

représentant la partie triangularisée d'une équation et une équation résiduelle:

$$Q(x_{k+1}, ..., x_n, p_1, ..., p_m) = 0$$

Les x_i sont les inconnues et les p_j représentent les paramêtres. Une équation résiduelle ne peut être résolue que si tous ses paramêtres sont instanciés.

Le résolveur peut gérer plusieurs équations de ce type.

0.1.2 Les variables

Le résolveur gère un certain nombre de variables représentant des paramètres d'équations, des arguments de fonctions, des inconnues ou des états. Certaines variables peuvent changer de statut en fonction de l'équation partiellement résolue qui est en cours de résolution.

Aux variables peuvent être associées une ou plusieurs valeurs dans $Z/3Z$. La gestion de ces valeurs est différente suivant le status de la variable. Une variable non rémanente peut être dans un état **indeterminé** s'il n'y a pas de valeur associée; **pré-instanciée** si plusieurs valeurs sont associées et enfin **instanciée** si une seule valeur est associée. Une variable non-rémanente, dans l'état instancié, ne peut changer de valeur ni même d'état en dehors d'un **raz** de toutes les variables non-rémanentes. Une tentative de réinstanciation d'une variable instanciée amène une erreur fatale.

Par contre, une variable rémanente est toujours **instanciée** mais sa valeur peut être modifiée.

Les variables sont globales et donc visibles par toutes les configurations. Leur état peut changer de manière interne lors de résolutions ou par appel à une méthode publique permettant de les instancier de l'extérieur. Afin d'assurer la liaison entre variables symboliques et valeurs, des ensembles standards de variables sont définis: les états, les incontrolables ou mesures, les conditions et les inconnues.

Les méthodes *set_ etats*, *set_ conditions* , *set_ mesures* permettent d'instancier globalement les variables correspondantes. La méthode *set_ inconnue* permet d'instantier une par une les sorties de la configuration courante.

La méthode *get_ valperm* donne un ensemble de valeurs possibles pour les variables de sortie de la configuration courante alors que *get_ solution* donne une solution de l'équation.

Enfin la méthode *raz* désinstancie toutes les variables non rémanentes. Cette méthode doit obligatoirement être invoquée au début d'un nouveau cycle de résolution. Les paramètres de l'équation résiduelle doivent également être instanciés. Ceci est assuré par la méthode *init_ resolution*. Cette méthode évalue les fonctions dans l'ordre où elles ont été déclarées puis instancie les paramétres de l'équation. Cette stratégie permet de résoudre des systèmes partiellement triangularisés et d'utiliser les évaluations des fonctions pour instancier les paramétres de l'équation résiduelle. Combinée avec l'utilisation de variables rémanentes elle permet d'implémenter certains reconstructeurs d'états.

0.1.3 Les configurations

Une configuration est l'association d'une équation non résolue, d'une liste de fonctions et d'une liste de variables. Les deux premiers éléments peuvent constituer une équation partiellement triangularisée mais la partie fonctionnelle peut également servir au calcul de paramêtres ou contenir des équations d'êvolution d'état. La liste de variables représente les sorties de la configuration. Les variables exprimées fonctionnellement peuvent en faire partie. Les inconnues de l'équation résolue y figurent obligatoirement. Cette liste peut comprendre des variables qui n'appartiennent à aucune des catégories que nous venons d'énumérer: ce sont des variables non contraintes par la configuration mais qui devrons être instanciées lors de la résolution.

Plusieurs configurations peuvent être gérées simultanément. Elle sont désignées par leur numéro d'ordre dans le fichier de description *xxx.res*. La méthode *set_ config* permet de positionner la configuration courante.

Une résolution se fait toujours sur une configuration. Elle commence par l'évaluation des fonctions dans l'ordre de la liste. Les arguments des fonctions doivent avoir été instanciés préalablement au calcul de la fonction. Un manquement à cette règle entraine une erreur fatale. La méthode

init_ resolution assure cette phase initiale sur la configuration courante.

La détermination des autres variables de sortie se poursuit alors par résolution interactive ou automatique de l'équation. Dans les deux cas la valeur d'une variable est choisie parmis celles autorisées par l'équation. Cette valeur est substituée dans l'équation et un nouveau choix est proposé pour les inconnues restant dans la nouvelle équation. Le mode automatique simule, par un tirage aléatoire, les choix sucessifs d'un utilisateur. Si une variable de sortie ne figure pas comme résultat d'une évaluation de fonction ou comme inconnue, elle est considérée comme totalement libre.

0.1.4 Les fichiers de description

La mise en oeuvre du résolveur doit résoudre le problème des liaisons entre les variables symboliques des polynomes utilisés pour représenter les équations et les valeurs des variables du programme C généré par le compilateur SIGNAL. La solution retenue est d'ordonner les variables symboliques et de mettre le même ordre sur les variables C correspondantes. La correspondance symbolique/valeurs se fait de manière analogue à la correspondance paramètres formels/paramètres effectifs dans les appels de procédures usuels. Ces correspondances sont décrites dans deux fichiers générés par SIGALI: *xxx.sim* et *xxx.res*. La génération en parallèle de ces deux fichiers par SIGALI assure les liaisons. Le fichier *xxx.sim* est utilisé par le programme de génération de simulateurs pour engendrer les interfaces entre le programme utilisateur et le résolveur. Le fichier *xxx.res* contient la liste des configurations ainsi que les variables **etats, mesures, conditions** qui seront instanciées à partir de cette interface. Par exemple les états seront énumérés dans *xxx.res* sous la forme (les variables symboliques sont codées par des entiers):

```
$E 1 3 4 6 9 13 15 17 19 22 35 36 37 38 39 40 41 42 43 44 45 46
```

et dans *xxx.sim*:

```
$E Z_Souris_Piece_1  Z_Souris_Porte_3  Z_Souris_Piece_3  Z_Souris_Porte_6
   Z_Souis_Piece_0  Z_Souris_Porte_1  Z_Souris_Porte_4 Z_Chat_Piece_2
   Z_Chat_Porte_3 Z_Chat_Piece_4  Z_Chat_Porte_6 Z_Chat_Piece_0
   Z_Chat_Porte_1 Z_Chat_Porte_4 Z_Souris_Piece_2  Z_Souris_Porte_2
   Z_Chat_Piece_1 Z_Chat_Porte_2  Z_Chat_Piece_3 Z_Souris_Piece_4
   Z_Souris_Porte_5 Z_Chat_Porte_5
```

Le fichier *xxx.res* contient également les TDD implémentant fonctions et équations sous un format externe compacté. Le résolveur comprend un package TDD permettant la lecture et le stockage des équations ainsi que

des fonctions de substitution et résolution. Le chargement du fichier *xxx.res* doit être fait en début de simulation en utilisant la méthode *chargement*.

0.1.5 Utilisation

Les descriptions des équations à résoudre ainsi que les états initiaux des éventuelles variables rémanentes ayant été chargés, un cycle de résolution a la forme:

- *raz*: toutes les variables non rémanentes sont mises dans l'état non-instancié.
- set_config: choix de l'équation partiellement résolue à résoudre.
- init_resolution: calcul des fonctions et instanciation des paramètres de l'équation.
- cycle *get_valperm/set_inconnue* proposant des valeurs autorisées et instanciant la variable choisie par l'utilisateur. La méthode *get_valperm* émet un message quand l'équation est résolue.
- A tout moment le cycle précédent peut être interrompu par un appel à *get_solution* qui complète la solution partielle pour les variables non instanciées.

0.2 Programmes Signal simulables

La synthèse de contrôleurs se faisant à partir de l'interprétation abstraite de la spécification en SIGNAL du système à contrôler, un contrôleur ne peut controler en fait que la partie système à évènements discrets d'un système hybride. Les assertions sur le modèle qui devront être vérifiées par le résolveur lors de la simulation ne pourrons l'être que si la perte d'information due à l'interprétation ne rend pas impossible le contrôle de cohérence sur les valeurs de signaux. Bien entendu il n'y a aucun problème si le système spécifié est purement logique. Les problèmes viennent des prédicats sur les signaux numériques.

Considérons par exemple un processus SIGNAL avec trois entrées `y1, y2, y3` de type *real* et comportant l'instruction `y3^= when(y1>y2)`. `y1` et `y2` sont alors nécessairement synchrones et l'horloge de `y3` est un sous échantillonnage de leur horloge commune.

Au niveau de l'interprétation abstraite on aura comme entrées l'horloge commune à `y1, y2` et l'horloge dont les instants correspondent à `y1>y2` vrai. L'introduction de cette horloge est rendue nécessaire par l'impossibilité de raisonner sur les valeurs numériques des signaux. Les équations d'horloge utilisées par le résolveur pour vérifier la cohérence des entrées impliqueront que l'horloge de `y1>y2` est un sous échantillonnage de l'horloge commune des signaux `y1` et `y2`. Par contre ces équations ne permettent pas de vérifier la cohérence des valeurs numériques de `y1` et `y2` avec l'horloge `when(y1>y2)`.

Par conséquent les programmes qui pourront être simulés de façon fiable sont les modèles purement logiques (ne comportant que des booléens et des horloges) et les programmes sans prédicats sur les valeurs numériques ou du moins tels que les horloges des entrées ne dépendent pas de prédicats numériques.

Un environnement de simulation acceptant une gamme plus large de programmes a été envisagé. Sa mise en oeuvre exigeant des études plus poussées et (en autre) une modification du générateur de code C, sa réalisation a été remise à plus tard.

Une deuxième limitation est liée à des problèmes techniques de compilation. Certains contrôleurs utilisent expressément les variables d'état. Il en est de même de certaines assertions: Dans l'exemple CHAT-SOURIS on doit exprimer l'impossibilité pour un animal d'être dans deux pièces au même moment, ce qui fait intervenir les états. Il est donc nécessaire de communiquer les valeurs des variables d'état au résolveur. Remarquons tout de suite que ce problème ne concerne que les *variables d'état booléennes.*

Or en SIGNAL, les variables d'état n'existent pas explicitement. Seul le retard ($) invoque implicitement une mémoire. Au niveau du source il est donc impossible de désigner explicitement un état. Il est envisageable de

récupérer les variables C créées au cours de la génération du code et de coupler les générateurs de code C et $Z/3Z$ afin d'assurer la liaison variable symbolique-valeur. Ceci demandant une intervention lourde sur le compilateur sans régler complètement le problème, nous avons préféré, pour cette première version, imposer une écriture des équations d'état que nous savons traiter en modifiant un peu le générateur de code $Z/3Z$. Cette écriture est d'ailleur naturelle dès que l'on veut décrire un automate en SIGNAL:

```
| X_ETAT := X_MODIF default ZX_ETAT
| ZX_ETAT := X_ETAT $ 1
```

où `X_ETAT` est une variable d'état. La première ligne est l'équation d'évolution où `X_MODIF` est le signal modifiant l'état. La deuxième ligne assure la fonction de mémorisation.

Le problème avec ce codage est de donner une horloge à l'équation d'évolution. En effet, les équations précédentes ne précisent pas à quelle horloge l'état est accessible et rafraichi. (En SIGNAL ce n'est pas un véritable état mais un signal, donc un couple (valeur, horloge)). On peut penser à deux solutions: soit la réunion de l'horloge de `X_MODIF` et de l'horloge d'utilisation de `X_ETAT` , soit l'horloge la plus rapide du programme. En fait tout suréchantillonnage de la première convient. Malheureusement aucune de ces horloges n'est disponible.

La solution provisoirement retenue est d'imposer une horloge la plus rapide `TICK` au modèle et de synchroniser les états booléens à cette horloge. Cette horloge sera l'horloge de simulation imposée par l'environnement de simulation généré automatiquement. Comme la notion d'horloge la plus rapide n'existe pas en SIGNAL nous serons obligés de synchroniser les entrées avec `TICK` et d'interdire tout autre utilisation de l'instruction retard.

La dernière contrainte sur les états est liée uniquement aux techniques de compilation. La compilation des processus SIGNAL se faisant par macro-expansion, le compilateur renomme les identificateurs des signaux de ces processus. Seules les entrées et les sorties d'un programme ne sont pas renommées. Afin d'assurer les liaisons entre les variables d'état et leurs valeurs, il est demandé à l'utilisateur de mettre les états en sortie du processus SIGNAL modélisant le système à simuler. Le shéma général [] du simulateur montre l'interfaçage avec l'environnement.