Graphe Hiérarchique d'Unités Fonctionnelles et l'Optimisation

Nikom Suvonvorn Nicolas Pernet Yves Sorel

INRIA - Domaine de Voluceau - Rocquencourt B.P.105
78153 Le Chesnay Cedex - France
nikomsai@hotmail.com

Résumé

Nous présentons une méthode de transformation des graphes flot de contrôle en un graphe flot de données en utilisant un graphe intermédiaire. L'idée d'utilisation de graphe intermédiaire pour la transformation est très employée dans l'analyse de données et la transformation de programme : "Static Single Assignment (SSA)" , "Dependence Flow Graph (DFG)" et "Sparse Evaluation Graphs (SEGs)". Dans notre transformation, nous proposons un type de graphe intermédiaire, appelé *graphe hiérarchique d'unités fonctionnelles*, ainsi que des méthodes d'optimisation.

Mots Clef : transformation des graphes, optimisation, système temps réel, graphe flot de données, graphe flot de contrôle.

Résumé

We present a method of graph transformation for transforming the control flow graphe into the data flow graph by using an intermediate graph. The idea of using the intermediate graph for the transformation is very employed in the domain of data analysis or program transformation : Static Single Assignment(SSA), Dependence Flow Graph (DFG) and Sparse Evaluation Graphs (SEGs). In our transformation, we propose a intermediate graph, called *hierarchical graph of functional units*, as well as the methods of optimization.

Mots Clef : graph transformation, optimization, real time system, data flow graph, control flow graph.

1 Introduction

Mon travail se situe dans le cadre du développement de SynDEx, logiciel d'aide à l a conception niveau système reposant sur la méthodologie AAA, développé par l'équipe OSTRE (Optimisation des Systèmes Temps Réel Embarqués) à l'INRIA Rocquencourt (Institut Nationale de Recherche en Informatique et Automatique) . SynDEx ne permet, jusqu'à présent, la spécification d'algorithme que sous forme de graphes flot de données. Ce formalisme est bien adapté à la représentation d'algorithme de traitement de données comme les lois de commande ou les filtres. Mais, bien que le principe de conditionnement ait été introduit et, avec lui une certaine manière d'inclure du contrôle dans les algorithmes, exprimer un algorithme de contrôle complexe avec ce formalisme reste fastidieux. Les graphes flot de contrôle (Réseaux de Petri, Statecharts) étant plus adaptés pour cela, il est intéressant de permettre le multi-formalisme pour la spécification d'algorithme : exprimer la partie traitement de données en graphe flot de données et la partie contrôlant les changements de modes, de paramètres, en graphe flot de contrôle. Le format flot de données permettant une distribution plus efficace que le format flot de contrôle, il faut unifier la spécification multi-formalisme en une spécification flot de données pour une implantation optimisée. Pour résoudre à ce problème, nous introduisons la méthode de transformation des graphes flot de contrôle en un graphe flot de données en utilisant un graphe intermédiaire [1] [2] [3] [4] [5], appelé *graphe hiérarchique d'unités fonctionnelles.*

La section 2 page suivante, nous présentons la notation du graphe hiérarchique d'unités fonctionnelles et la méthode d'optimisation du graphe sera présenté dans la section 3 page 6.

2 Graphe hiérarchique d'unités fonctionnelles

2.1 Définition de l'unité fonctionnelle

Une *unité fonctionnelle* est un graphe conditionné de dépendances de données. C'est un hypergraphe orienté. Elle se spécifie par un ensemble de sommets connectés, où chaque sommet est une fonction de calcul, et chaque arc est une dépendance de données entre *une fonction productrice* et une ou plusieurs *fonctions consommatrices*[6].

Définition 1 Soit $fu = [O, D]$ est une unité fonctionnelle où :

- $O = \{sw, f_0, f_1, \cdots, f_{n-1}, sl\}$ est en ensemble d'éléments, appelés des *sommets*. C'est un graphe de $(n+2)$ème ordre où :
  - sw est un sommet de type "switch",
  - f_i est un sommet de type "func$_i$",
  - sl est un sommet de type "selector".
- D est un ensemble d'arcs, appelés *dépendances de données intra-unité fonctionnelle*, définis par la matrice d'incidence sommets-arcs suivante :

$$D = \begin{pmatrix} +1 & +1 & \cdots & +1 & 0 & 0 & \cdots & 0 \\ -1 & 0 & \cdots & 0 & +1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 & 0 & +1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 0 & 0 & \cdots & +1 \\ 0 & 0 & \cdots & 0 & -1 & -1 & \cdots & -1 \end{pmatrix}$$

  Telle que chaque colonne correspond aux arcs $(s_0, s_1, \cdots, s_{n-1}, o_0, o_1, \cdots, o_{n-1})$ et chaque ligne aux sommets $(sw, f_0, f_1, \cdots, f_{n-1}, sl)$. Les éléments de la matrice peuvent prendre les valeurs $+1$, -1 et 0 où :
  - $+1$ origine de l'arc,
  - -1 extremité de l'arc,
  - 0 pas de sortie, ni d'entrée.
- le fonctionnement d'une unité fonctionnelle défini dans [2] par :
  - $s_0, s_1, \cdots, s_{n-1} = switch(s)$,
  - $o_i = func_i(s_i, v_i)$,
  - $c = selector(s, o_0, o_1, \cdots, o_{n-1})$.

Une unité fonctionnelle permet de représenter le comportement conditionné d'un ensemble de fonction "func$_i$". Elle se caractérise par ses entrées $(s, v_0, v_1, \cdots, v_{n-1})$ et sa sortie (c), et par les fonctions qui calculent la valeur de la sortie à partir de celles des entrées. Une unité fonctionnelle a une entrée spéciale (s), appelée *dépendance de conditionnement*. La dépendance de conditionnement porte une valeur, appelée *donnée de conditionnement*, qui permet au sommet "switch" de choisir parmi des fonctions alternatives, le sommet "func$_i$" qui sera exécuté.

Pour spécifier le comportement d'un système, un graphe comportant plusieurs unités fonctionnelles peut être utilisé. Le système spécifié est réactif [7], c'est à dire qu'il réagit à des stimuli provenant de l'environement extérieur en produisant des actions qui peuvent à leur tour modifier l'environnement. Les stimulis externes n'étant pas bornés en nombre dans le temps, le nombre de réactions du système ne peut pas l'être non plus. Ainsi le graphe spécifiant le système est implicitement répété infiniement. Le graphe ne représente que le comportement du système pour l'une de ces répétitions infinies. Si un sommet du graphe doit consommer lors de la $n^{ème}$ répétition du graphe une donnée produite lors de la $(n-1)^{ème}$, on utilise un sommet "delay" qui produit la donnée consommée lors de la répétition précédente.

Une unité fonctionnelle peut consommer des données produites par ses prédécesseurs. Ceux-ci peuvent être d'autres unités fonctionnelles, des sommets "sensor" ou des sommets "delay". La sortie d'une unité fonctionnelle peut être consommée par des successeurs pouvant être d'autres unités fonctionnelles, des sommets "actuator" ou des sommets "delay". La figure 1 page suivante représente une unité fonctionnelle.

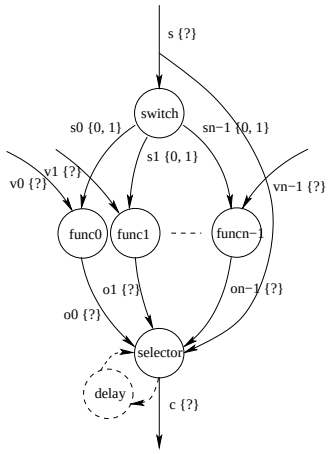


FIG. 1 – *L'unité fonctionnelle.*

2.1.1 Sommet "switch"

Le sommet "switch" consomme une donnée (s) et produit les données (s_0, $s_1, \cdots, s_{n-1}$) respectivement pour ses successeurs (f_0, $f_1, \cdots, f_{n-1}$). Chacune de ces données (s_i) est un booléen ($0 \rightarrow fausse$ et $1 \rightarrow vraie$). On a $s_i = 1$ si $i = s$, sinon $s_i = 0$. La donnée s_i sera utilisée par le sommet "$func_i$" pour s'exécuter si sa valeur est vraie.

2.1.2 Sommet "func$_i$"

Le sommet "func$_i$" consomme les données (s_i, v_i) et produit une donnée (o_i). Cette fonction ne s'exécute (consommation de v_i et production de o_i) que si la valeur de s_i est vraie. Dans ce cas, on a $o_i = f_i(v_i)$ et la valeur est significative. Et si la valeur de s_i est fausse, on produit la valeur *null*.

2.1.3 Sommet "selector"

Le sommet "selector" consomme les données (s, o_0, $o_1, \cdots, o_{n-1}$) et produit une donnée (c). Le but de ce sommet est de recopier la donnée (o_i) ayant une valeur significative sur sa sortie. Pour connaître l'entrée (o_i) qu'il doit prendre en compte, il utilise la valeur de la *donnée de conditionnement* (s). Ainsi on a $c = o_s$ sauf si $o_s = null$. Dans ce cas, la sortie c doit garder la valeur qu'elle avait lors de la précédente répétition infinie et pour cela, le sommet *selector* utilise un sommet delay. Par la suite le sommet "delay" utilisé par chaque sommet "selector" sera implicite et non représenté sur les graphes. Le fait qu'une donnée garde sa précédente valeur est appelé *rémanence*.

2.2 Graphe d'unités fonctionnelles

La spécification fonctionnelle d'une application peut être obtenue par la composition de plusieurs unités fonctionnelles. Cette composition consiste à combiner les unités fonctionnelles en un graphe, appelé *graphe d'unités fonctionnelles*. On appelle *dépendance de données inter-unités fonctionnelles conditionnante* un arc reliant le sommet "selector" d'une unité fonctionnelle au(x) sommet(s) "switch" d'une ou plusieurs autre unité(s) fonctionnelle(s). L'unité fonctionnelle productrice produit la *donnée de conditionnement* sur cet arc, qui est consommée par une ou plusieurs unité(s) fonctionnelle(s) consommatrice(s). Cet arc symbolise le fait que la sortie (c) peut conditionner les autres unités fonctionnelles.

Dans un graphe d'unités fonctionnelles, on peut avoir un autre type d'arc, appelée *dépendance de données inter-unités fonctionnelles non-conditionnante*. Cet arc peut représenter une dependance de données entre :

- un sommet "selector" d'une unité fonctionnelle et un sommet "func$_i$" d'une autre unité fonctionnelle,
- un sommet "selector" d'une unité fonctionnelle et un sommet "delay" ou "actuator",
- un sommet "sensor" et un sommet "func$_i$" d'une unité fonctionnelle,

– un sommet "delay" et un sommet "func$_i$" d'une unité fonctionnelle.

Cet arc symbolise le fait qu'une unité fonctionnelle peut échanger (consommation et production) des données avec l'extérieur, que ce soit avec une autre unité fonctionnelle ou avec des sommets "externes" tel que les "delay", "sensor" et "actuator".

Définition 2 Soit $G_{fu} = [FU_a, D_a]$ est un graphe d'unités fonctionnelles où :
– FU_a est l'ensemble des unités fonctionnelles du graphe, $FU_a = \{fu_i\}_{1 \leq i \leq n}$, Card $FU_a = n$ où n est le nombre d'unités fonctionnelles,
– D_a est l'ensemble des arcs de *dépendance de données inter-unités fonctionnelles conditionnante* et de *dépendance de données inter-unités fonctionnelles non-conditionnante* du graphe.

La figure 2 est un exemple du graphe d'unités fonctionnelles.

2.3 Graphe hiérarchique d'unités fonctionnelles

Le graphe d'unités fonctionnelles est composée d'un ensemble d'unités fonctionnelles où chaque unité fonctionnelle représente un comportement conditionné. Ce comportement est caractérisé par les sommets "func$_i$". Le sommet "func$_i$" peut être définie par la fonction "super" qui permet de spécifier un sous-graphe, sous forme de graphe d'unités fonctionnelles. Ainsi ce sommet introduit la notion de hiérarchie. Le graphe d'unités fonctionnelles utilisant des sommets "super" est appelé *graphe hiérarchique d'unités fonctionnelles.*

2.4 Fonctions spéciales

2.4.1 Fonction "sensor"

La fonction "sensor" est une fonction de calcul avec une sortie et sans entrée. Il s'agit de représenter une entrée d'application, interface avec l'environnement extérieur. Cette fonction ne peut produire qu'une donnée, consommée par une ou plusieurs unité(s) fonctionnelle(s).

2.4.2 Fonction "actuator"

La fonction "actuator" est une fonction de calcul avec une entrée et sans sortie. Il s'agit de représenter une sortie d'application, interface avec l'environnement extérieur. Elle consomme une donnée produite par une unité fonctionnelle.

2.4.3 Fonction "delay"

La fonction "delay" est une fonction possédant une entrée et une sortie. Cette fonction produit à sa sortie la donnée consommé en entrée lors de la précédente répétition infinie du graphe. Lorsqu'une unité fonctionnelle a besoin de consommer, lors de la $n^{ème}$ répétition infinie du graphe, une donnée produite lors de la $(n-1)^{ème}$ répétition infinie, il faut intercaler ce type de fonction.

2.4.4 Fonction "super"

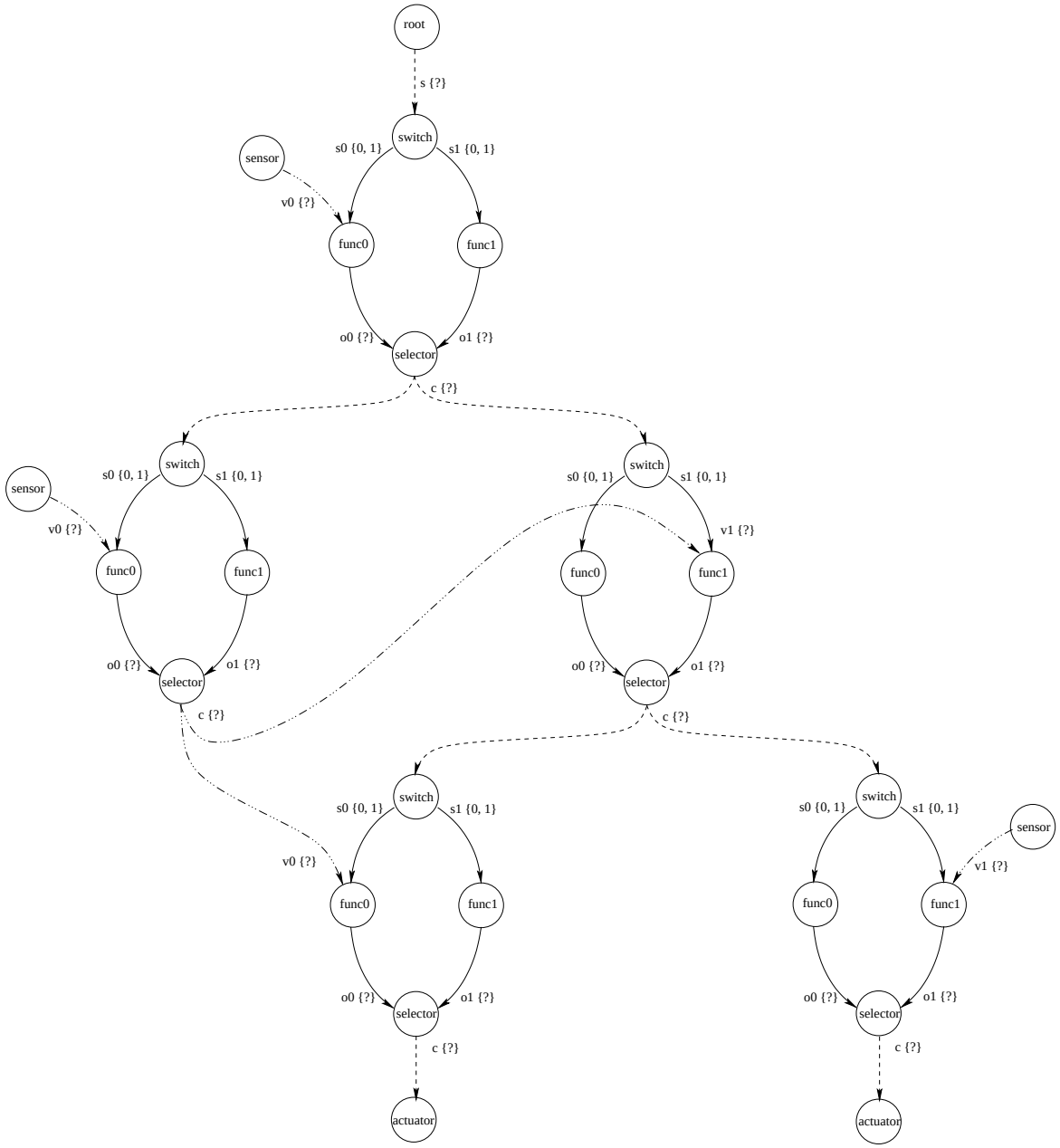
La fonction "super" introduit la notion de hiérarchie dans le graphe hiérarchique d'unités fonctionnelles. Permettant de spécifier un sous-graphe, elle est utilisée à la place de la fonction "func$_i$".

2.4.5 Fonction "dummy"

La fonction "dummy" est une non-fonction dans le sens où elle ne fait aucune action. Utilisée à la place de la fonction "func$_i$" dans l'unité fonctionnelle, elle consomme une donnée (s_i) sur son arc d'entrée et produit la valeur *null* sur sa sortie (o_i).

2.4.6 Fonction "root"

La fonction "root" est une fonction qui est utilisé pour définir l'horloge du graphe hiérarchique d'unités fonctionelles. Elle ne possède pas d'entrée et produit la valeur 1 sur sa sortie lors de chaque répétition infinie du graphe.



(a)

FIG. 2 – *Graphe d'unités fonctionnelles.*

3 Optimisation du graphe hiérarchique d'unités fonctionnelles

Dans le chapitre précédent, nous avons présenté le graphe hiérarchique d'unités fonctionnelles. Ce graphe nous permet de spécifier le fonctionnement d'une application. Cependant, ce graphe n'est pas optimal car certains sommets sont employés de manière rédondante. Afin de diminuer le nombre de sommet et ainsi la complexité du graphe, il convient de détecter et d'éliminer cette redondance.

3.1 Détection et élimination de la redondance des "switch"

Dans un graphe hiérarchique d'unités fonctionnelles, nous avons vu qu'une *dépendance de données inter-unités fonctionnelles conditionnante* permet à une unité fonctionnelle productrice fu_p de produire une *donnée de conditionnement* consommée par des unités fonctionnelles consommatrices fu_c. Dans ces unités fonctionnelles consommatrices, il peut y avoir rédondance de sommets "swtich" que nous proposons de détecter et éliminer de la manière suivante.

Définition 3 Soit fu_1, et fu_2 deux unités fonctionnelles tel que :
- $fu_1 = [O_1, D_1]$ et $O_1 = \{sw_1, f_{1_0}, f_{1_1}, \cdots, f_{1_{n-1}}, sl_1\}$,
- $fu_2 = [O_2, D_2]$ et $O_2 = \{sw_2, f_{2_0}, f_{2_1}, \cdots, f_{2_{n-1}}, sl_2\}$.

Définition 4 Les unités fonctionnelles fu_1 et fu_2 sont exclusives si et seulement si, $\forall i, 0 \leq i \leq n-1$, au moins l'un des deux sommets "func$_{1_i}$" et "func$_{2_i}$" est une fonction "dummy".

Règle 1 Les sommets "switch" sw_1 et sw_2 appartenant respectivement aux unités fonctionnelles fu_1 et fu_2 sont redondants si et seulement si les unités fonctionnelles fu_1 et fu_2 sont exclusives et consomment la même *donnée de conditionnement* produite par une unité fonctionnelle fu_p.

La règle 1 permet de détecter la redondance des sommets "switch" dans un graphe hiérarchique d'unités fonctionnelles afin de les éliminer et d'optimiser ainsi le graphe. Pour cela, on remplace les sommets "switch" sw_1 et sw_2 par un seul sommet "switch" sw_c dont l'entrée s_c est la même que sw_1 et sw_2, et les sorties $s_{c0}, s_{c1}, \cdots$, et s_{cn-1} sont définies par l'union des sorties des sommets "switch" sw_1 et sw_2, soit $s_{ci} = s_{1_i} \cup s_{2_i}$, $\forall i, 0 \leq i \leq n-1$. En outre, il faut également insérer sur chaque *dépendance de données inter-unités fonctionnelles non-conditionnante* entre fu_1 et fu_2, une fonction "delay" afin de préserver le comportement du graphe.

Cette technique permet de réduire le nombre des sommets "switch" utilisé dans le graphe et donc de diminuer le temps de calcul nécessaire à l'évaluation de celui-ci. En contre-partie, les sommets "delay" ajoutés augmentent l'espace mémoire nécessaire.

La figure 3 page suivante donne un exemple de détection et d'élimination de sommets "switch". La figure (a) est un graphe hiérarchique d'unités fonctionnelles composé de trois unités fonctionnelles. L'unité fonctionnelle qui se trouve en haut du graphe produit une donnée de conditionnement qui est consommée, via une *dépendance de données inter-unités fonctionnelles conditionnante*, par les deux autres unités fonctionnelles. De plus, en observant les sommets "func$_i$" de ces deux unités fonctionnelles, on en déduit qu'elles sont exclusives. Les sommets "switch" de ces deux unités fonctionnelles sont donc rédondants. On peut éliminer cette rédondance et obtenir le graphe qui apparait sur la figure (b). Il ne comporte plus que deux sommets "switch" au lieu de trois et un sommet "delay" a été insérer sur la *dépendance de données inter-unités fonctionnelles non-conditionnante* qui reliait les deux unités fonctionnelles exclusives de la figure (a).

3.2 Détection et élimination de la redondance des "selector"

La détection de la redondance de sommets "selector" utilise le même principe que celui de détection de redondance des sommets "switch".

Règle 2 Les sommets "selector" sl_1 d'unité fonctionnelle fu_1 et sl_2 d'unité fonctionnelle fu_2 sont redondant si et seulement si ces unités fonctionnelles sont exclusives et consomment la *donnée de conditionnement* produite par la même unité fonctionnelle fu_p.

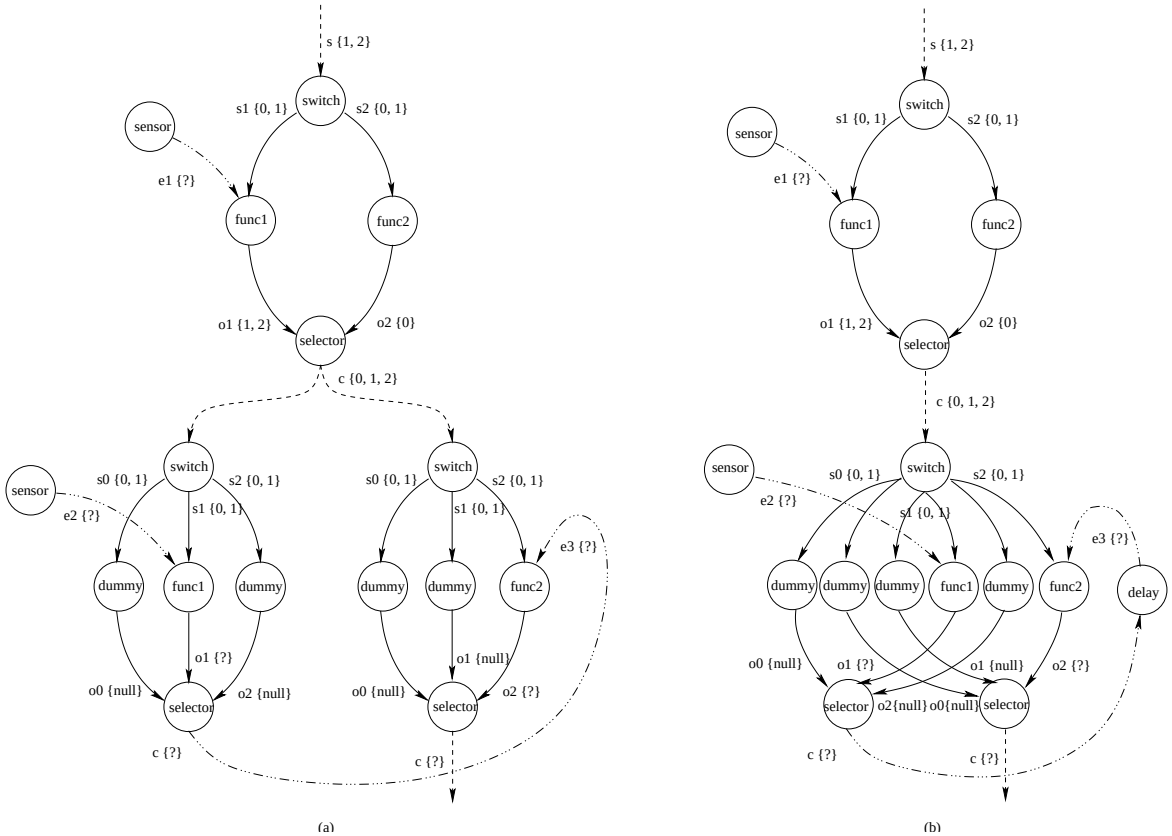


FIG. 3 – *(a) Le graphe hiérarchique d'unités fonctionnelles ; (b) le même graphe après élimination de la redondance du sommet "switch".*

On déduit des règles 1 et 2 que si les sommets "switch" de deux unités fonctionnelles sont redondants, alors leurs sommets "selector" le sont aussi. Bien que la règle 2 permette de détecter la redondance des sommets "selector", leur élimination pose un problème de comportement. En effet, il n'est possible de supprimer uniquement que les sommets "selector" redondants dont les sorties sont consommées par des fonctions "actuator". Dans ce cas, on remplace les sommets "selector" sl_1 et sl_2 par un seul sommet "selector" sl_c dont les entrées o_{c0}, $o_{c1}, \cdots$, et o_{cn-1} sont définies par l'union des entrées des sommets "selector" sl_1 et sl_2 soit $o_{ci} = o_{1i} \cup o_{2i}$, $\forall i$, $0 \leq i \leq n-1$, et sa sortie c_c est consommée par l'ensemble des consommateurs des sommets "selector" sl_1 ou sl_2. Lorsque l'on élimine les sommets "selector" de deux unités fonctionnelles, celles-ci ne comportent déjà plus qu'un seul sommet "switch", la rédondance de "switch" ayant été éliminée. Passer de deux sommets "selector" à un seul entraine de passer de de $2n$ sommets "func" à n. Pour cela, on supprime, pour i allant de 0 à $n-1$, le sommet "func$_{1i}$" si c'est un sommet "dummy", sinon le sommet "func$_{2i}$".

La figure 4 page suivante présente un exemple d'élimination de la redondance de sommets "selector". La figure (a) est un graphe hiérarchique d'unités fonctionnelles après élimination de la redondance de sommets "switch". Il est est composé de trois unités fonctionnelles, une unité fonctionnelle productrice et deux unités fonctionnelles consommatrices, ne comportant plus qu'un seul sommet switch pour deux. La figure (b) montre le graphe obtenu après élimination de la redondance de sommets "selector".

3.3 Détection et élimination des tests inutiles

Dans un graphe hiérarchique d'unités fonctionnelles, il est possible qu'une unité fonctionnelle fu_c soit la seule consommatrice de la *donnée de conditionnement* s_c qui est produite par une unité fonctionnelle productrice fu_p. La donnée s_c est consommée par le sommet "switch" sw_c de l'unité fonctionnelle consommatrice et est testée pour choisir le sommet "func$_{cs_c}$" qui doit être activé. Or, il arrive que certains de ces tests soient inutiles et nous pouvons les détecter de la manière suivante :

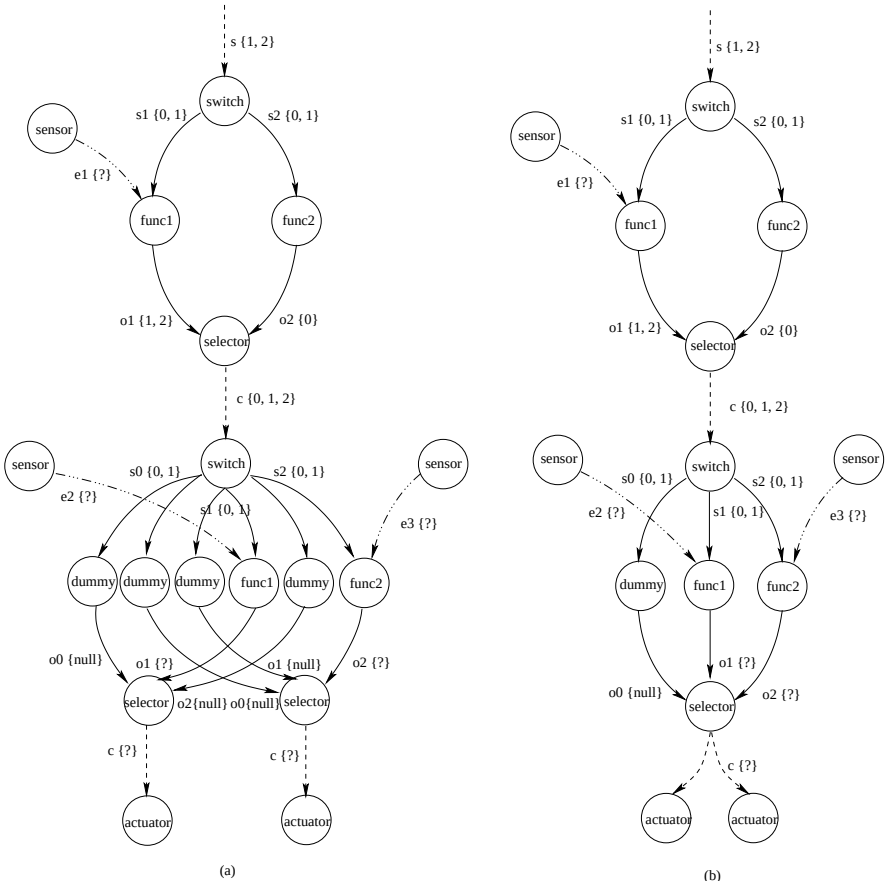


FIG. 4 – *(a) Le graphe hiérarchique d'unités fonctionnelles ; (b) le même graphe après élimination de la redondance des sommets "selector".*

Règle 3 Le test de la *donnée de conditionnement* s_c de valeur r par le sommet "switch" w_c est inutile si et seulement si les sommets "func$_{cr}$" sont tous les fonctions "dummy".

Définition 5 Soit r une des valeurs possibles de la donnée s_c dont le test est inutile au sens de la règle 3 :

- la fonction "func$_{p_r}$" est la fonction "func$_{p_i}$" (f_{p_i}) de l'unité fonctionnelle productrice qui produit la valeur r,
- la fonction "func$_{p_n}$" est la fonction "func$_{p_i}$" (f_{p_i}) de l'unité fonctionnelle productrice qui produit toute les valeurs possibles de la *donnée de conditionnement* c_p sauf r.

Pour éliminer la redondance de test de la donnée s_c de valeur r, nous procédons de la manière suivante :

- supprimer le test de la donnée s_c de valeur r du sommet "switch" sw_c,
- supprimer les sommets "func$_{cr}$",
- supprimer l'entrée o_{cr} du sommets "selector" sl_c
- remplacer le sommet "func$_{p_r}$" par une fonction "dummy",
- remplacer la fonction "func$_{p_n}$ par une fonction "super" contenant la fonction "func$_{p_n}$" et l'unité fonctionnelle consommatrice ainsi que toutes les unités fonctionnelles qu'elle conditionne par des *dépendances de données inter-unités fonctionnelles conditionnantes*. La donnée produite par le sommet "func$_{p_n}$" est consommée par l'unité fonctionnelle consommatrice.
- dupliquer le sommet "selector" de l'unité fonctionnelle productrice si le sommet super possède plus d'une sortie.

Les entrées du sommet "super" sont l'union des entrées de la fonction "func$_{p_n}$" d'unité fonctionnelle productrice (s_{p_i}, et v_{p_i}) et des entrées des sommets "func" de l'unité fonctionnelle consommatrice ($v_{c0}, v_{c1}, \cdots,$ et v_{cn-1}). Ses sorties sont celles de l'unité fonctionnelle consommatrice et des unités fonctionnelles qu'elle conditionne. Chaque sortie de la fonction "super" o_{p_i} est consommée par un sommet

"selector" différent, ce qui peut impliquer la dupplication de celui existant dans l'unité fonctionnelle productrice.

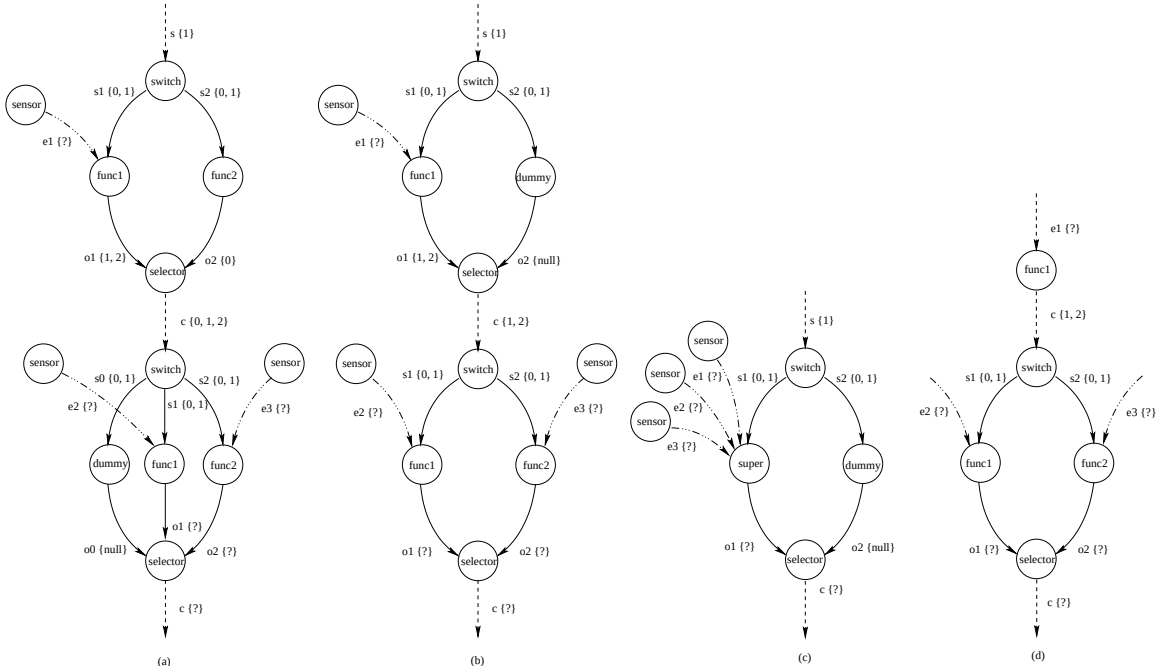


FIG. 5 – *(a) Un graphe hiérarchique d'unités fonctionnelles ; (b) le même graphe après suppression du test de la donnée c de valeur* 0 *et du sommet "dummy" de l'unité fonctionnelle consommatrice et remplacement du sommet "func$_2$" de l'unité fonctionnelle productrice par une fonction "dummy" ; (c) le graphe après inclusion de l'unité fonctionnelle consommatrice dans le sommet "super" de l'unité fonctionnelle productrice ; (d) le sous-graphe du sommet "super".*

3.4 Conclusion

Nous avons proposé le formalisme de *graphe hiérarchique d'unités fonctionnelles* comme type de graphe intermédiaire pour la transformation des graphes. Pour cela, nous avons décrit comment il permettait de décrire du contrôle tout en étant un formalisme flot de données. De plus, nous avons défini des règles pour minimiser le nombre de tests composant ce type de graphe, autrement dit d'optimiser du graphe.

Dans le cadre du développement de SynDEx, nous avons appliqué ce formalisme pour transformer le graphe Scicos en un graphe d'algorithme SynDEx et nous avons obtenu le résultat du graphe d'algorithme SynDEx optimisé, donc qui est bien adapté pour les systèmes temps réel. Nous espérons que ce formalise peut être appliqué pour la transformation de différents types du graphe.

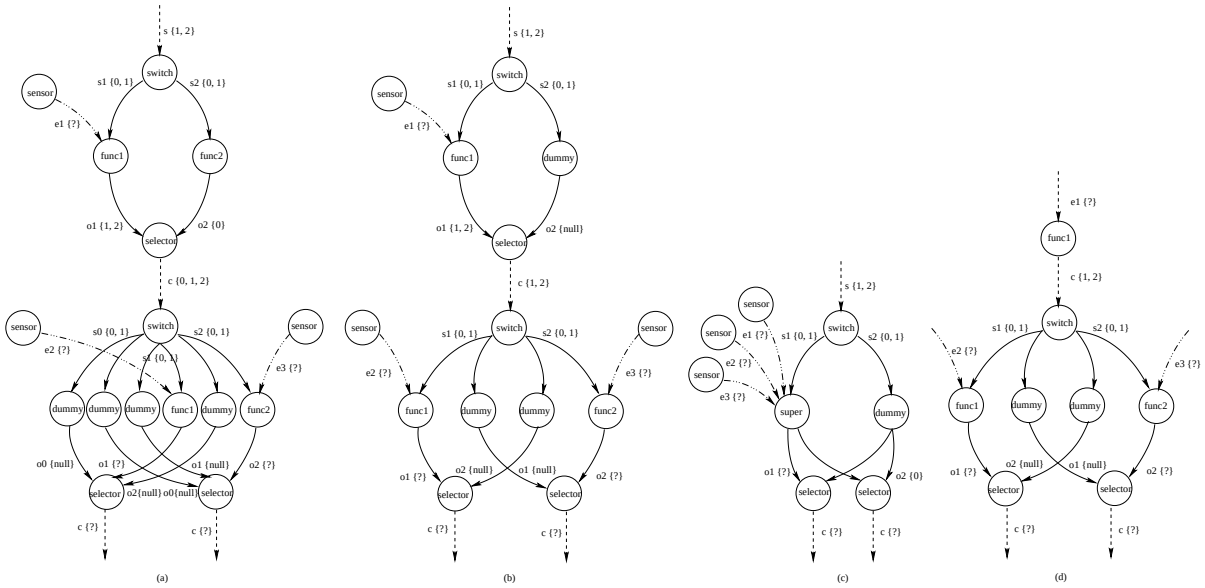


FIG. 6 – *(a) Un graphe hiérarchique d'unités fonctionnelles ; (b) le même graphe après suppression du test de la donnée c de valeur* 0 *et du sommet "dummy" de l'unité fonctionnelle consommatrice et remplacement du sommet "func$_2$" de l'unité fonctionnelle productrice par une fonction "dummy" ; (c) le graphe après inclusion de l'unité fonctionnelle consommatrice dans le sommet "super" de l'unité fonctionnelle productrice et duplication le sommet "selector" pour chacune des sorties du sommet "super" ; (d) le sous-graphe du sommet "super".*

Références

[1] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, October 1991.

[2] K. Pingali, M. Beck, R. Johnson, M. Moudgill, and P. Stodghill. Dependence Flow Graphs: an Algebraic Approach to Program Dependencies. In A. Nicolau, D. Gelernter, T. Gross, and D. Padua, editors, *Advances in Languages and Compilers for Parallel Processing*, pages 445–467. MIT Press, Cambridge, MA, 1991.

[3] Richard Johnson and Keshav Pingali. Dependence-based program analysis. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 78–89, 1993.

[4] Jong Deok Choi, Ron Cytron, and Jeanne Ferrante. Automatic construction of sparse data flow evaluation graphs. *Conferrence Record of the 18th Annual ACM Symposium on Principles of Programming Languages*, pages 55–66, January 1991.

[5] Nicolas Pernet. Spécification multi-formalisme d'algorithmes de contrôle-commande. Rapport de stage, INRIA, septembre 2002.

[6] Thierry Grandpierre. *Modélisation d'architectures parallèles hétérogènes pour la génération automatique d'exécutifs distribués temps réel optimisés.* PhD thesis, Université de Paris Sud, Spécialité électronique, 30/11/2000.

[7] David Harel and Amir Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*. Springer Verlag, New York, 1985.