RenPar'9,
Lausanne. Suisse, 20–23 mai 1997.

Sémantique dénotationnelle d'un langage parallèle pour modèles PRAM*

François Siewe [a], Maurice Tchuente [a], Jean-Louis Roch [b]

[a] Université de Yaoundé I, département Informatique BP 812 Yaoundé (Cameroun)

[b]LMC-IMAG, BP 53X, 38041 Grenoble Cedex 9

Mots clef : Sémantique dénotationnelle, modèle PRAM, modèle BSP.

Résumé

Nous proposons une sémantique dénotationnelle d'un langage élémentaire, $\mathrm{L}_{\mathrm{PRAM}}$, de calcul parallèle synchrone pour modèle PRAM en définissant la sémantique des accès synchrones à la mémoire partagée. Puis nous étendons ce langage pour construire la sémantique dénotationnelle d'un langage $\mathrm{L}_{\mathrm{BSP}}$ de type BSP, synchrone par super-pas, dans lequel les accès mémoire sont asynchrones.

1. Introduction

L'approche dénotationnelle a été utilisée avec succès pour spécifier la sémantique de la plupart des langages de programmation séquentielle [5]. Mais la sémantique dénotationnelle des langages de programmation parallèle est difficile à exprimer du fait de la présence de plusieurs processeurs interagissant concurremment d'une part et de la diversité des modèles parallèles d'autre part. Une autre difficulté importante provient du non-déterminisme, intrinsèque à la plupart des modèles parallèles. Cette situation a amené la plupart des auteurs à adopter la méthode opérationnelle pour la description de la sémantique des langages parallèles [1]. Nous proposons ici une sémantique dénotationnelle pour langages parallèles basés sur le modèle PRAM, modèle qui, bien que très utilisé en algorithmique n'a pas encore à notre connaissance fait l'objet d'une telle étude.
Nous définissons d'abord (§2) la syntaxe abstraite d'un langage élémentaire synchrone, $\mathbf{L}_{\mathrm{PRAM}}$: les communications sont exprimées par des accès en lecture/écriture à une mémoire globale. $\mathrm{L}_{\mathrm{PRAM}}$ étend le langage L [1] par l'introduction de branchements. Cette syntaxe sert de base pour définir $\mathbf{L}_{\mathrm{BSP}}$, langage inspiré du modèle BSP [3], qui permet un fonctionnement asynchrone (synchrone relaché) des processeurs. La synchronisation par phase (super-pas) est réalisée par l'introduction d'une barrière de synchronisation; les commandes de lecture et d'écriture en mémoire globale sont asynchrones. Puis, nous définissons la sémantique dénotationnelle des langages $\mathrm{L}_{\mathrm{PRAM}}$ (§3) et $\mathrm{L}_{\mathrm{BSP}}$ (§4).

2. Syntaxe abstraite des langages $\mathbf{L}_{\mathrm{PRAM}}$ et $\mathbf{L}_{\mathrm{BSP}}$

Notre modèle de base est du type PRAM SIMD [4] : tous les processeurs exécutent soit la même instruction soit aucune (`skip`) à chaque étape, possèdent une mémoire locale privée et accèdent en lecture/écriture à une mémoire globale. Le langage $\mathrm{L}_{\mathrm{PRAM}}$ est un langage élémentaire pour la programmation d'un tel modèle. Une particularité est que les processeurs peuvent exécuter des instructions de branchement à partir d'un test sur la valeur d'une variable globale : ainsi le flot d'instructions reste le même sur chaque processeur. Nous donnons ci-dessous la syntaxe abstraite de ce langage, en commentant les instructions de base du langage. Les domaines syntaxiques sont les suivants : PROGRAM (programme : P), EXP (expressions : E), DEC (déclarations : D), NUM (numéraux : N), CMDE (commandes : C), IDE (identificateurs : I), ETIQ (étiquettes de commandes, Φ).

* . Ce travail a été réalisé dans le cadre du projet Calcul Parallèle soutenu par l'agence Aire Développement.

E, D, N sont définis comme dans un langage séquentiel classique. Dans la suite, E_{global}(resp. E_{local}) dénote une adresse en mémoire globale (resp. locale). P est l'axiome de base de la grammaire des programmes, C celui des commandes et Ce celui des commandes élémentaires (i.e. qui ne comportent pas d'instructions de branchement).

P::=	`Program`(N) D C	N *désigne le nombre de processeurs qui vont exécuter le corps du programme,* D *les déclarations des variables partagées,* C *le corps du programme parallèle.*
C::=	Φ:C	*commande étiquetée*
|	`Goto` Φ	*Branchement inconditionnel*
|	`GlobalTest` I `ThenGoto` Φ	*Branchement conditionnel sur un test global (*I *variable globale).*
|	Ce `Where` E	*Instruction conditionnelle;* Ce *est une commande élémentaire (pas de branchement) et* E *une expression booléenne.*
|	`Begin` D C `end`	*Bloc de commandes.*
|	C; C	*Composition séquentielle de commandes.*
|	`halt`	*Commande de fin du programme.*
Ce::=	I:= E	*Affectation à variable locale.*
|	`Skip`	*Instruction vide.*
|	`GetProcId`(I)	*Accès au numéro du processeur.*
|	`GlobalRead`(E_{local}, E_{global})	*Lecture en mémoire partagée.*
|	`GlobalWrite`(E_{local}, E_{global})	*Écriture en mémoire partagée.*

Puis, nous étendons le langage L_{PRAM} pour définir un langage élémentaire L_{BSP} (inspiré de [3]) asynchrone de type BSP. Les processeurs fonctionnent de manière indépendante et sont synchronisés par phase (superpas) à l'aide d'une barrière de synchronisation explicite : instruction `sync`. Chaque superpas est exécuté de manière asynchrone. Les accès en mémoire globale (`GlobalRead` et `GlobalWrite`) sont alors asynchrones et deviennent effectifs après exécution de la première instruction `sync` qui leur succède lors de l'exécution. À une étape donnée, les processeurs peuvent exécuter indépendamment les uns des autres des instructions de branchement. Par rapport à L_{PRAM} , l'instruction «`GlobalTest` I `ThenGoto` Φ» est remplacée par une instruction de branchement sur un test local: `LocalTest` I `ThenGoto` Φ où I est une variable locale, l'instruction «Ce `where` E» par Ce. La syntaxe abstraite étend L_{PRAM} par ces deux commandes :

C::=	...	*Les commandes de* L_{PRAM}
|	`sync`	*Barrière de synchronisation.*
|	`LocalTest` I `ThenGoto` Φ	*Branchement conditionnel sur un test local.*

3. Une sémantique dénotationnelle du langage L_{PRAM}

Nous définissons la sémantique par utilisation des domaines utilisés en séquentiel et ajout de nouveaux domaines nécessités par la pre'sence de plusieurs processeurs. Les domaines et fonctions sémantiques hérités de la sémantique dénotationnelle du modèle séquentiel sont : INT (nombres), LOCN (locations), Sv = INT (valeurs stockables en mémoire), Ev = Sv+INT (valeurs expressives), Dv = LOCN+ARRAY (valeurs dénotables), ENV = IDE → Dv (environnement des déclarations), ECONT = ETIQ → CONT (environnement des étiquettes de commandes), CONT = (PSTATE → PSTATE)+{[]} (continuations des commandes: [] est la continuation vide et signifie qu'il n'y a plus rien à faire), STORE = LOCN → Sv (fonctions mémoires), STATE = LOCN×STORE (états d'une mémoire).
Pour la mise en oeuvre du parallélisme, nous introduisons les domaines :
• PROCID : domaine des identificateurs de processeur.
• PSTATE = STATE×(PROCID → STATE). Si (x,y) est un élément de PSTATE alors x représente l'état courant de la mémoire globale et y une fonction qui à chaque processeur associe l'état courant de sa mémoire locale.
• PENV = ENV×(PROCID → ENV)×ECONT. Si (x,y,z) est un élément de PENV, x représente l'environnement des déclatations globales, y une fonction qui à chaque processeur

associe son environnement local et z une fonction qui à chaque étiquette associe la continuation qu'elle dénote dans le programme.
• ACU = PROCID → {0,1}. C'est le domaine des fonctions qui indiquent l'état d'activité de chaque processeur. Un processeur est soit actif (dans ce cas son état d'activité vaut 1) soit passif (son état d'activité vaut 0). A chaque étape de l'exécution d'un programme, seuls les processeurs actifs à cette étape exécutent l'instruction courante.
Fonctions et règles sémantiques. Dans ce paragraphe, nous donnons les dénotations des principales constructions de L_{PRAM} .
• Programme : $\mathbb{P}$: PROGRAM → PENV → PSTATE → PSTATE
L'exécution d'un programme « `Program`(N) D C » consiste à faire exécuter les déclarations globales (D) par un processeur (par exemple le processeur 0) et à exécuter les commandes (C) en parallèle sur N processeurs.
$\mathbb{P}$ [`Program`(N) D C]$(\rho_1, \rho_2, \rho_3)((g, \pi), \Omega) = \mathbb{C}$ [C]$(\rho'_1, \rho_2, \rho_3)\omega[]((g', \pi'), \Omega)$
où $(\rho'_1, (g', \pi')) = \mathbf{D}$[D]$\rho_1(g, \pi)$ et $\omega = \lambda\mu$.if $\mu \in$ {0,1, ..., V[N]-1} then 1 else 0.
• Commandes : $\mathbb{C}$: CMDE → PENV → ACU → CONT → PSTATE → PSTATE
Nous nous limitons aux principales commandes en les commentant; l'utilisation des continuations permet la dénotation des instructions de branchement.
– $\mathbb{C}$ [`goto` Φ]$(\rho_1, \rho_2, \rho_3)\omega\theta((g, \pi), \Omega) = \rho_3[\Phi]((g, \pi), \Omega)$: *transfert du contrôle à la continuation* $\rho_3[\Phi]$.
– $\mathbb{C}$ [`GlobalTest` I `ThenGoto` Φ] $(\rho_1, \rho_2, \rho_3)\omega\theta((g, \pi), \Omega) = \theta'((g, \pi), \Omega)$ où θ' = if $\pi(\rho_1$[I])=1 then $\rho_3[\Phi]$ else θ. *Si I vaut 1 (vrai), le contrôle est passé à la commande étiquetée par* Φ *(continuation* $\rho_3[\Phi]$*), sinon on continue en séquence.*
– $\mathbb{C}$ [C where E] $\rho\omega\theta((g, \pi), \Omega) = \mathbb{C}$ [C]$\rho\omega'\theta((g, \pi), \Omega)$ où $\omega' = \lambda\mu$.if $\omega(\mu) = 1$ then E_{bool}[E]$\rho_2[\mu]\Omega[\mu]$ else 0. *Seuls les processeurs qui évaluent E à 1 (vrai) exécutent C.*
– $\mathbb{C}$ [`halt`]$\rho\omega\theta((g, \pi), \Omega) = ((g, \pi), \Omega)$. *Arrêt de l'exécution du programme.*
– $\mathbb{C}$ [`GlobalRead`(E_{local}, E_{global})] $(\rho_1, \rho_2, \rho_3)\omega\theta((g, \pi), \Omega) = \theta((g, \pi), \Omega')$
– $\mathbb{C}$ [`GlobalWrite`(E_{local}, E_{global})] $(\rho_1, \rho_2, \rho_3)\omega\theta((g, \pi), \Omega) = \theta((g, \pi'), \Omega)$
où Ω' *et* π' *sont définis suivant la classe de machine XRXW.* Nous ne donnons pas ici leur calcul qui est détaillé dans [6].
• Expressions : La fonction $\mathbb{E}$: EXP → (ENV×(PROCID → ENV)) → STATE → Ev est utilisée pour évaluer une dénotation d'adresse globale. Les fonctions E_{bool}, E_r, E_l, **V** et **D** défínnissent la sémantique des expressions booléennes, droites (R-values), gauches (L-values), des numéraux et des déclarations. Nous les utilisons telles qu'elles sont définies en séquentiel [5] pour modéliser les traitements faits localement sans interaction avec la mémoire globale.

4. Une sémantique dénotationnelle du langage L_{BSP}

Pour modéliser l'exécution en parallèle et asynchrone de plusieurs flots d'instructions, nous introduisons la notion intuitive de continuation parallèle. Nous désignons par continuation parallèle ce qui reste à faire pour terminer l'exécution d'un programme parallèle, *i.e.* la "somme" de ce qui reste à faire à chaque processeur pour terminer le programme. Dans le cas SIMD du langage L_{PRAM} , la continuation parallèle est identique à la continuation séquentielle i.e la continuation d'un programme exécuté sur un seul processeur. Nous introduisons de nouveaux domaines sémantiques pour décrire, pour chaque processeur, les accès asynchrones effectués en mémoire partagée (TENV et XSTATE) et les continuations locales (PCONT) :
• TENV = $(\text{LOCN}\times\text{LOCN})^\star$: *domaine des environnements temporaires,*
• XSTATE = TENV×TENV×STATE,
• CONT = (XSTATE → (CONT×XSTATE))+{[]} : *domaine des continuations séquentielles,*
• PCONT = $(\text{PROCID}\times\text{CONT})^\star$: *domaine des continuations parallèles.*
Fonctions et règles sémantiques. Dans la suite, $\mathbb{P}$ définit la fonction sémantique d'un programme parallèle, Δ celle d'une continuation parallèlle et $\mathbb{C}$ celle d'une commande qui s'exécute localement sur un processeur.
• Programme : $\mathbb{P}$: PROGRAM → ENV → PSTATE → PSTATE

$\mathbb{P}$[Program(N) D C]$\rho((g,\pi),\Omega) = \Delta((0,\theta),(1,\theta),...,($ V[N]-1, $\theta))((g',\pi'),\Omega)$
où $\theta = \mathcal{C}$ [C]$(\rho',\{\},\{\})[]$ et $(\rho',(g',\pi')) =$ D[D]$\rho(g,\pi)$. *Chaque processeur exécute localement une copie de* C.
• Continuations parallèles : Δ: PCONT $\rightarrow$ PSTATE $\rightarrow$ PSTATE. Δ lance en parallèle l'exécution des différentes continuations séquentielles sur un super-pas. Au point de synchronisation, les requètes de lecture (R) et d'écriture (W) sont effectuées. L'exécution reprend ensuite pour le super-pas suivant. Les processeurs dont la continuation est vide arrêtent l'exécution (fonction RED). Δ est définie recursivement par : $\Delta()((g,\pi),\Omega) = ((g,\pi),\Omega)$ et
$\Delta((\mu_0,\theta_0),\ldots,(\mu_{n-1},\theta_{n-1}))((g,\pi),\Omega) = \Delta\mathrm{RED}((\mu_0,\theta'_0),\ldots,(\mu_{n-1},\theta'_{n-1}))((g,\pi'),\Omega')$

$$\text{où} \left\{ \begin{array}{l} (\theta'_i, R_{\mu_i}, W_{\mu_i}, (f'_{\mu_i}, \sigma'_{\mu_i})) = \theta_i((),(),\Omega[\mu_i]), i = 0,1,\ldots,n-1 \\ \pi' = \text{if } \exists\mu \in \{\mu_0,\ldots,\mu_{n-1}\}, W_\mu \neq () \text{ then } \pi_{\text{cas}} \text{ else } \pi. \\ \Omega' = \lambda\mu. \text{ if } \mu \in \{\mu_0,\ldots,\mu_{n-1}\} \text{ et } R_\mu \neq () \text{ then } S_\mu \text{ else } \Omega[\mu] \end{array} \right. .$$

et RED: PCONT $\rightarrow$ PCONT est une fonction de prétraitement; RED $= \lambda t : PCONT.t'$ où t' est obtenu en supprimant dans t tous les couples de la forme $(\mu,[])$. Ainsi, on a :
$\mathrm{RED}((0,[])) = ()$ et $\mathrm{RED}((\mu_0,\theta_0),(\mu_1,[]),(\mu_2,\theta_2)) = ((\mu_0,\theta_0),(\mu_2,\theta_2))$ avec $\theta_0,\theta_2 \neq []$.
Dans le calcul de π_{cas} et S_μ plusieurs types de conflits sont à prendre en compte : les conflits d'accès en écriture, les conflits d'accès en lecture (traités comme pour $\mathrm{L_{PRAM}}$) et un nouveau type de conflits, lors d'accès asynchrones en lecture et écriture à une même variable globale, qui est résolu en réalisant d'abord toutes les lectures avant les écritures [2, 6].
• $\mathcal{C}$: CMDE $\rightarrow$ (ENV$\times$ENV$\times$ECONT) $\rightarrow$ CONT $\rightarrow$ XSTATE $\rightarrow$ (CONT$\times$XSTATE). Chaque processeur exécute localement sa continuation jusqu'à atteindre une commande `sync` ou `halt`. Lorsque tous les processeurs sont synchronisés, les accès mémoire (R et W : fonction Δ) sont effectués. Les processeurs qui ont exécuté `halt` terminent l'exécution du prgramme et les autres reprennent l'exécution à partir de leur continuation locale. Nous nous limitons ici aux principales commandes.
– $\mathcal{C}$ [`LocalTest` I `ThenGoto` Φ] $(\rho_1,\rho_2,\rho_3)\theta(R,W,(f,\sigma)) = \theta'(R,W,(f,\sigma))$ où
$\theta' =$ if $\sigma(\rho_2[\mathrm{I}])$=1 then $\rho_3[\Phi]$ else θ.
– $\mathcal{C}$ [`GlobalRead`($\mathrm{E_{local}}$, $\mathrm{E_{global}}$)] $(\rho_1,\rho_2,\rho_3)\theta(R,W,(f,\sigma)) = \theta((u,v).R,W,(f,\sigma))$ où
u $= \mathbf{E}_l[\mathrm{E_{local}}]\rho_2(f,\sigma)$ et v $= \mathbb{E}[\mathrm{E_{global}}](\rho_1,\rho_2)(f,\sigma)$. *Le couple (u, v) est concatené à R.*
– $\mathcal{C}$ [`GlobalWrite`($\mathrm{E_{local}}$, $\mathrm{E_{global}}$)] $(\rho_1,\rho_2,\rho_3)\theta(R,W,(f,\sigma)) = \theta(R,(u,v).W,(f,\sigma))$ où
u $= \mathbf{E}_l[\mathrm{E_{local}}]\rho_2(f,\sigma)$ et v $= \mathbb{E}[\mathrm{E_{global}}](\rho_1,\rho_2)(f,\sigma)$ *Le couple (u, v) est concatené à W.*
– $\mathcal{C}$ [`halt`]$\rho\theta(R,W,(f,\sigma)) = ([],R,W,(f,\sigma))$. `halt` *arrète l'exécution du proceseur.*
– $\mathcal{C}$ [`sync`]$\rho\theta(R,W,(f,\sigma)) = (\theta,R,W,(f,\sigma))$. `sync` *synchronise le processeur.*

5. Conclusion

Nous nous sommes intéressés à la spécification formelle d'un langage parallèle épuré. Cette définition de base peut être utilisée pour définir l'équivalence et la quasi-équivalence de programmes parallèles [6].

Bibliographie

1. L. Bougé and J.L. Levaire. Control structures for data-parallel simd languages: Semantics and implementation. In *Future Generation of Computer System*, 1992.
2. S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the 10th ACM Symposium on Theory of Computing*, pages 114–118. ACM Press, 1978.
3. M.W. Goudreau, J.M.D. Hill, K. Lang, and B. McColl. A Proposal for the BSP Worldwide Standard Library . http://www.bsp-worldwide.org/, Oxford University, april 1996.
4. J. Jájá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.
5. P.D. Mosses. Denotational semantics. In J. van Leuwen, editor, *Handbook of Theoretical Computer Science*, chapter 11. Elsevier, 1990.
6. F. Siewe. Sématique dénotationnelle d'un langage parallèle pour modèle PRAM. Research report, Université de Yaoundé I, Cameroun, décembre 1996.