SU/FS/master/info/MU4IN503 APS
Notes de cours

P. Manoury

Janvier 2022

Contents

4 APS2: tableaux

Il s'agit ici d'ajouter des structures de tableaux: séquences de valeurs stockées à des adresses consécutives en mémoire. Cet ajout va modifier le mécanisme d'allocation mémoire. Le langage *APS2* fournira une primitive explicite d'allocation d'un *bloc mémoire*. Ces blocs mémoire constitueront une nouvelle catégorie de *valeurs* manipulées par le langage. On trouvera ces valeurs dans les environnements, mais également en mémoire.

Les tableaux seront des structures modifiables (comme les variables). L'ajout de ces structures aura donc un impact sur l'instruction d'affectation. En particulier, *à gauche*[1] d'une affectation pourront figurer des expressions désignant un élément de tableau et non plus simplement un identificateur. Nous aurons donc une nouvelle catégorie syntaxique pour les membres gauches des affectations: les *lvalue*'s (voir par exemple, dans la grammaire de C).

Du point de vue des types, les tableaux sont des *types paramétrés*. Les fonctions primitives pour les tableaux que nous introduisons dans *APS2* sont applicables à des tableaux quelque soit le type des éléments qu'ils contiennent. En ce sens, ces opérateurs sont *polymorphes*.

4.1 Syntaxe

Le langage est étendu avec des symboles pour les opérations primitives sur les tableaux: création d'un tableau, désignation d'un élément, longueur d'un tableau et modification de la valeur d'un élément. Le langage des types est également étendu avec un symbole et une notation pour le *type* des tableaux.

4.1.1 Lexique

Opérateurs primitifs `alloc len nth vset`

Type `vec`

4.1.2 Grammaire

Le principal ajout à la grammaire est le non-terminal LVAL qui peut prendre place comme premier terme (*lvalue*) dans l'instruction d'affectation. On ajoute également une nouvelle construction de type pour les tableaux.

STAT	::=	...
	|	`SET` LVAL EXPR
LVAL	::=	ident
	|	`(nth` LVAL EXPR `)`
EXPR	::=	...
	|	`(alloc` EXPR `)`
	|	`(len` EXPR `)`
	|	`(nth` EXPR EXPR `)`
	|	`(vset` EXPR EXPR EXPR `)`
TYPE	::=	...
	|	`(vec` TYPE `)`

4.2 Typage

4.2.1 Expressions

Le polymorphisme des opérateurs primitifs est traité au niveau des règles de typage, comme l'était le polymorphisme de l'opérateur fonctionnel d'alternative (le `if`):

(ALLOC) pour tout $t \in$ TYPE, si $\Gamma \vdash_{\text{EXPR}} e$: `int` alors $\Gamma \vdash_{\text{EXPR}}$ `(alloc` e`)` : `(vec` t`)`

[1] L'expression *«à gauche»* est héritée de la notation usuelle de l'affectation par un symbole infixe: `x = x+1`, par exemple.

(NTH) pour tout $t \in$ TYPE, si $\Gamma \vdash_{\text{EXPR}} e_1$: `(vec` t`)` et si $\Gamma \vdash_{\text{EXPR}} e_2$: `int` alors $\Gamma \vdash_{\text{EXPR}}$ `(nth` e_1 e_2`)` : t

(LEN) pour tout $t \in$ TYPE, si $\Gamma \vdash_{\text{EXPR}} e$: `(vec` t`)` alors $\Gamma \vdash_{\text{EXPR}}$ `(len` e`)` : `int`

(VSET) pour tout $t \in$ TYPE, si $\Gamma \vdash_{\text{EXPR}} e_1$: `(vec` t`)`, si $\Gamma \vdash_{\text{EXPR}} e_1$: `int` et si $\Gamma \vdash_{\text{EXPR}} e_3 : t$ alors $\Gamma \vdash_{\text{EXPR}}$ `(vset` e_1 e_2 e_3`)` : `(vec` t`)`

lvalue Il est inutile de donner des règles de typage spécifiques pour les *lvalue*'s. De ce point de vue, ce sont des expressions comme les autres. Il n'en ira pas de même du point de vue sémantique.

4.2.2 Instructions

La seule instruction dont le typage est affecté par le passage de *APS1(a)* à *APS2* est l'affectation où le premier terme peut prendre la forme d'une expression, restreinte à celles autorisées par la clause LVAL de la grammaire.

(SET) si $\Gamma \vdash_{\text{EXPR}} lv : t$ et si $\Gamma \vdash_{\text{EXPR}} e : t$ alors $\Gamma \vdash_{\text{STAT}}$ (`SET` lv e) : `void`

4.3 Sémantique

Pour *APS2*, la sémantique va connaître des changements assez importants. Le premier est l'extension à une nouvelle catégorie de valeurs qui figureront dans les environnement et dans la mémoire: les blocs de valeurs. Le deuxième tient à ce qu'une expression peut maintenant avoir un effet sur la mémoire: application des opérateurs d'allocation et d'affectation. Un dernier impact est que le domaine des adresses ne peut plus rester complètement abstrait car nous y avons besoin de la notions d'adresses *consécutives* pour constituer des blocs.

4.3.1 Domaines et opérations sémantiques

Domaines Il faut choisir un domaine d'adresses où l'on possède la notion de succession afin de pouvoir obtenir des *intervalles* d'adresse. Un candidat naturel pour définir ce domaine est celui des entiers naturels. Un bloc mémoire peut alors simplement être représenté par l'adresse de départ et la taille du bloc. On pose donc:

Adresses $A = N$ (ordonnées avec incrément)

Blocs mémoires $B = A \times N$

Valeurs $V\oplus = B$

Mémoire $S = A \to (Z \oplus B)$

Opérations Il nous faut pour *APS2* un mécanisme d'allocation d'une plage mémoire (intervalle ou bloc d'adresses). On se donne pour cela la fonction *allocn*, de type $S \times N \to (A \times S)$ telle que $allocn(\sigma, n) = (a, \sigma')$ si et seulement si, pour tout $i \in [0, n[$, $a+i$ n'est pas dans le domaine de σ et $\sigma' = \sigma[a = any; \ldots; a+n-1 = any]$.

Hypothèse simplificatrice: la mémoire contient désormais deux catégories de valeurs: les *valeurs immédiates* entières (Z) et les blocs constitués d'une adresse et d'une taille (B), donc de deux «valeurs» entières. Notre *hypothèse simplificatrice* consistera à ne pas distinguer ces deux catégories de valeurs du point de vue de leur taille en mémoire. Les éléments de Z et ceux de B seront considérés comme occupant des espaces mémoire de même taille – ce qui est réalisable, mais entraîne des contraintes sur l'ensemble des valeurs et des adresses possibles.

4.3.2 Expressions

Avec l'introduction des primitives `alloc` et `vset`, l'évaluation d'une expression produit une valeur ainsi qu'un effet sur la mémoire. Ainsi la signature de $\vdash_{\text{Expr}}$ pour *APS2* est $E \times S \times \text{Expr} \times V \times S$.

On écrit $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (v, \sigma')$.

La primitive `alloc` donne un *nouveau* bloc mémoire. Son résultat est double: le bloc lui-même (adresse et taille); le nouvel état mémoire.

(ALLOC) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (inZ(n), \sigma')$ et si $allocn(\sigma', n) = (a, \sigma'')$
alors $\rho, \sigma \vdash_{\text{Expr}}$ `(alloc` e`)` $\rightsquigarrow (inB(a, n), \sigma'')$

La primitive `vset` est l'avatar fonctionnel de l'instruction d'affectation `SET` lorsque l'adresse affectée (*lvalue*) est celle d'un élément d'un bloc mémoire (valeur des tableaux). L'évaluation de son application a donc un effet sur la mémoire. La valeur de cette «affectation» est celle du bloc affecté associé au nouvel état mémoire.

(VSET) si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow (inB(a, n), \sigma')$, si $\rho, \sigma' \vdash_{\text{Expr}} e_2 \rightsquigarrow (inZ(i), \sigma'')$ et si $\rho, \sigma'' \vdash_{\text{Expr}} e_3 \rightsquigarrow (v, \sigma''')$
alors $\rho, \sigma \vdash_{\text{Expr}}$ `(vset` e_1 e_2 e_3`)` $\rightsquigarrow (inB(a, n), \sigma'''[a + i := v])$

La primitive `nth` donne la valeur de l'un des éléments d'un bloc mémoire:

(NTH) si $\rho, \sigma \vdash_{\text{Expr}} e_1 \rightsquigarrow (inB(a, n), \sigma')$ et si $\rho, \sigma' \vdash_{\text{Expr}} e_2 \rightsquigarrow (inZ(i), \sigma'')$
alors $\rho, \sigma \vdash_{\text{Expr}}$ `(nth` e_1 e_2`)` $\rightsquigarrow (\sigma''(a + i), \sigma'')$

La primitive `len` donne la longueur associée à un bloc mémoire:

(LEN) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (inB(a, n), \sigma')$ alors $\rho, \sigma \vdash_{\text{Expr}}$ `(len` e`)` $\rightsquigarrow (inZ(n), \sigma')$

Les autres règles de $\vdash_{\text{Expr}}$ doivent simplement être amendées pour tenir compte de la nouvelle signature de la relation. Elles sont données dans le formulaire. Les seules règles qui demandent un léger commentaire sont celles concernant l'évaluation des identificateurs:

(ID1) si $x \in \mathsf{ident}$, si $\rho(x) = inA(a)$ et si $\sigma(a) = v$ alors $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (v, \sigma)$

(ID2) si $x \in \mathsf{ident}$, si $\rho(x) = v$ et si $v \neq inA(a)$ alors $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (v, \sigma)$

Notez que dans (ID2), si la valeur associée à x dans l'environnement est un bloc, celui-ci est la valeur donnée par la relation.

Paramètres d'appel Les régles de la relation $\vdash_{\text{Expar}}$ sont simplement amendées pour tenir compte de l'éventuel effet sur la mémoire (voir formulaire).

4.3.3 Déclarations

Le changement de $\vdash_{\text{Expr}}$ induit un changement dans la déclaration des constantes puisque, en particulier, lorsque l'on définit un tableau, on a un effet sur la mémoire. Pour *APS2*, la signature de la relation $\vdash_{\text{Def}}$ est $E \times S \times \text{Dec} \times E \times S$.

(CONST) si $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow (v, \sigma')$ alors $\rho, \sigma \vdash_{\text{Def}}$ (`CONST` x t e) $\rightsquigarrow (\rho[x = v], \sigma')$

Les autres règles sont simplement ammendées pour tenir compte des éventuels effets mémoire.

4.3.4 Valeurs gauches

Le premier terme d'une affectation est soit un identificateur simple, soit la désignation d'un élément de tableau avec la primitive `nth`. Dans ces deux cas, il faut obtenir l'adresse visée par l'affectation. Le premier cas sera similaire à ce que nous avions pour *APS1*; dans le second cas, nous devons tenir compte du fait que le calcul de l'indice dans le tableau fait appel à l'évaluation d'une expression et peut donc avoir un effet sur la mémoire.

Nous appelons $\vdash_{\text{LVAL}}$ la relation sémantique pour les «valeurs-gauches», elle a pour domaine $E \times S \times \text{LVAL} \times A \times S$.

On écrit $\rho, \sigma \vdash_{\text{LVAL}} lv \rightsquigarrow (a, \sigma')$

Pour un identificateur, l'adresse cherchée doit être donnée par l'environnement et il n'y a alors pas d'effet sur la mémoire.

(LID) si $x \in \mathsf{ident}$ et si $\rho(x) = inA(a)$ alors $\rho, \sigma \vdash_{\text{LVAL}} x \rightsquigarrow (a, \sigma)$

On a, dans *APS2*, une situation analogue à celle des tableaux *statique* du langage C: les variables de type tableau ne sont pas modifiables; comme *lvalue*, un identificateur ne doit pas avoir une valeur de la forme $inB(a, n)$.

Dans le cas de l'usage de `nth`, il faut calculer la valeur de l'expression désignant l'indice visé, ce qui peut avoir un effet sur la mémoire. De plus, on distingue le cas d'un usage imbriqué de `nth` de son usage simple (avec un identificateur). Dans le cas d'un usage imbriqué, il faut suivre l'indirection mémoire:

(LNTH1) si $x \in \mathsf{ident}$, si $\rho(x) = inB(a, n)$ et si $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow (inZ(i), \sigma')$
alors $\rho, \sigma \vdash_{\text{LVAL}} \texttt{(nth } x\ e\texttt{)} \rightsquigarrow (a + i, \sigma')$

(LNTH2) si $\rho, \sigma \vdash_{\text{LVAL}} lv \rightsquigarrow (a_1, \sigma')$ avec $\sigma'(a_1) = inB(a_2, \_)$ et si $\rho, \sigma' \vdash_{\text{EXPR}} e \rightsquigarrow (inZ(i), \sigma'')$
alors $\rho, \sigma \vdash_{\text{LVAL}} \texttt{(nth } lv\ e\texttt{)} \rightsquigarrow (a_2 + i, \sigma'')$

Notez ici l'ordre dans lequel nous avons évalué les composants de (`nth` lv e). Il eût pu être inverse.

4.3.5 Instructions

Le principal changement dans la définition de $\vdash_{\text{STAT}}$ est l'usage de $\vdash_{\text{LVAL}}$ pour l'évaluation de l'affectation.

(SET) $\rho, \sigma \vdash_{\text{EXPR}} e \rightsquigarrow (v, \sigma')$ et si $\rho, \sigma' \vdash_{\text{LVAL}} lv \rightsquigarrow (a, \sigma'')$ alors $\rho, \sigma, \omega \vdash_{\text{STAT}} (\texttt{SET } lv\ e) \rightsquigarrow (\sigma''[a := v], \omega)$

Pour les autres cas, il faut simplement tenir compte du changement de signature de $\vdash_{\text{EXPR}}$.