Chapitre 5

Définition formelle des ASM

Nous avons vu, dans le chapitre un, l'intérêt d'un langage algorithmiquement complet. Nous avons ensuite présenté des exemples de programmes en AsmL, implémentation partielle d'un langage dont nous avons annoncé qu'il était algorithmiquement complet (sans justification pour l'instant). Nous allons maintenant donner une définition formelle des ASM dans ce chapitre.

Les différences essentielles entre le langage ASM et les autres langages de programmation (impératifs) sont au nombre de trois, à savoir :

- Le *relâchement des primitives* : dans un langage tel que le langage C, il existe un nombre fini (très petit) de types prédéfinis, avec un nombre (également très petit) d'opérations, de fonctions et de relations définies pour ces types ; ceci est également le cas en AsmL ; en langage ASM théorique, par contre, ces primitives seront définies pour chaque algorithme et feront parties de l'environnement de l'algorithme.
- La *simultanéïté* comme structure de contrôle par défaut pour les blocs, au lieu du séquencement pour tous les autres modèles impératifs.
- L'*itération du programme* pour effectuer les étapes du calcul.

5.1 Le relâchement des primitives et les structures du premier ordre

5.1.1 Étude d'un exemple

Reprenons un algorithme très simple, celui qui consiste à convertir une température exprimée en degré Celsius en la température exprimée en degré Farenheit.

En langage vernaculaire, cet algorithme se traduit par la formule :

$$F = \frac{9}{5}C + 32$$

Remarquons cependant que si cette formule est l'essence de l'algorithme, il ne s'agit pas de l'algorithme complètement explicité : on a besoin d'un **environnement** pour exprimer clairement l'algorithme, par exemple pour dire qu'il faut exprimer F en fonction de C et non le contraire.

5.1.1.1 Cas du langage C

Commençons par utiliser le langage C comme langage formel d'expression d'algorithmes. Commençons par écrire un programme irréaliste, car sans entrée-sortie. Un tel programme minimum est :

```
void main(void)
     {
     float C, F;

     F = 9/5*C + 32;
     }
```

Commentaires.- 1^o) On remarque, sur cet exemple, que l'**environnement de contrôle** (ou **syntaxique**) minimum en langage C consiste à entourer notre formule de :

```
void main(void)
     {

     }
```

ce qui permet de donner quelques indications au compilateur. Ces indications sont obligatoires pour un programme en langage C ou en n'importe quel langage (y compris en AsmL, où elles sont moins nombreuses cependant).

2^o) La **déclaration des types** :

```
     float C, F;
```

permet elle aussi de préciser notre algorithme. Elle fait partie de celui-ci, bien qu'elle apparaisse (si on peut dire) souvent sous forme de non-dit lorsqu'on exprime l'algorithme en langage vernaculaire. La preuve qu'il est indispensable de préciser ces types est que ce n'est pas le seul choix possible ; nous aurions pu aussi bien écrire :

```
     double C, F;
```

(dont certains diront que ce changement de type est lié au langage C) mais aussi :

```
     int C, F;
```

qui change vraiment la nature de l'algorithme. On peut parler d'**environnement sémantique** à propos de la déclaration des types.

3^o) Remarquons que, dans le cas du type entier, on a un problème de parenthésage pour la formule, qu'il faut absolument réécrire :

```
F = (9*C)/5 + 32;
```

si on veut traduire l'algorithme informel au plus proche. Nous retrouvons ici une nouvelle forme de l'*environnement syntaxique*.

4^o) L'*environnement sémantique* permet également d'interpréter '/' comme la division (dans le cas des types `float` et `double`) et comme le quotient exact dans la divison euclidienne dans le cas du type `int`. Il permet aussi d'interpréter '$*$' comme la multiplication, '+' comme l'addition et '32' comme une constante bien définie de l'univers du discours (lui aussi défini grâce à la déclaration des types).

5^o) Le point-virgule ';' à la fin de la formule a une importance fondamentale en langage C. Il indique la fin d'une instruction. Il fait partie de l'*environnement syntaxique*.

6^o) Remarquons que nous avons peu de façons d'exprimer notre formule en langage C : les seules variantes portent sur le choix des noms des variables et sur le parenthésage.

7^o) Un tel programme n'a cependant aucun intérêt parce qu'il n'a aucune interaction avec l'extérieur. L'utilisation d'un langage formel tel que le langage C nous force à compléter l'algorithme en ajoutant explicitement des entrées-sorties. Un premier programme C complet est :

```
#include <stdio.h>

void main(void)
     {
     float C, F;

     printf("C = ");
     scanf("%f", &C);
     F = 9/5*C + 32;
     printf("F = %f\n", F);
     }
```

On a ajouté l'**environnement d'interactivité**, qui est toujours un non-dit en langage vernaculaire.

Une autre façon de présenter l'algorithme est d'écrire une fonction :

```
float conv(float C)
     {
     return 9/5*C + 32;
     }
```

Conclusion.- Un programme (en langage C) comporte trois types d'environnements :

- l'*environnement syntaxique*, qui permet de donner des indications au compilateur ;
- l'*environnement sémantique*, qui permet d'interpréter les opérations ;
- l'*environnement d'interactivité*, qui permet une interaction avec le monde extérieur.

5.1.1.2 Cas de ASM

Programme brut.- Le programme brut peut également être traduit par la formule :

```
F := (9*C)/5 + 32
```

en AsmL. On peut également le traduire par l'une des formules suivantes, parmi beaucoup d'autres :

```
F := (9.C)/5 + 32
```

```
F := (9xC)/5 + 32
```

si on veut jouer, par exemple, sur la façon d'exprimer la multiplication, qui est une opération sans symbole standard dans la vie courante.

Notion d'algèbre statique.- Les ASM veulent tenir compte de cette flexibilité de la notation de la multiplication. Cette flexibilité entraîne la nécessité d'une explicitation de l'environnement sémantique de l'algorithme. Celui-ci se fait à travers de ce qui est appelé l'**algèbre statique** dans le cas de ASM.

Lorsqu'on présente une formule telle que la formule ci-dessus, on sous-entend un **univers du discours** sur lequel seront effectués nos opérations. Il nous vient immédiatement à l'esprit, dans notre cas, l'ensemble $\mathbb{R}$ des nombres réels. Pour des raisons de calculabilité, on peut lui préférer l'ensemble $\mathbb{Q}$ des nombres rationnels ou un autre ensemble de nombres (comme, par exemple, celui représenté par `float` ou `double` en langage C). Cependant l'univers du discours que l'on sous-entend pour notre exemple n'a pas une grande importance pour l'instant : nous voulons juste pointer sur la notion d'algèbre statique et donc, pour commencer, sur celle d'univers du discours.

On a également besoin d'interpréter les opérations qui interviennent dans la formule : la multiplication, la division et l'addition dans notre exemple, ou tout au moins les opérations représentées par les symboles. Il faut également interpréter les constantes 9, 5 et 32. Autrement dit, on a besoin d'un univers du discours muni de quelques opérations. Il s'agit de ce qu'on appelle une *algèbre* dans la discipline mathématique appelée *algèbre universelle* ou d'une *structure du premier ordre* en logique mathématique.

La structure qui nous vient immédiatement à l'esprit dans notre cas est :

$$(\mathbb{R}, +,\ .,\ /,\ 5,\ 9,\ 32).$$

Ce n'est pas la seule possible, puisqu'il peut aussi s'agir de :

$$(\mathbb{Q}, +,\ .,\ /,\ 5,\ 9,\ 32)$$

ou de :

$$(\mathbb{R}, +32,\ /5,\ .9)$$

dans la mesure où on n'a pas besoin de l'addition, de la division et de la multiplication dans toute leurs généralités.

Ceci nous montre bien l'intérêt de spécifier l'algèbre statique pour spécifier (mieux) notre algorithme.

Notion d'éléments dynamiques.- Nous n'avons pas parlé des *variables* `F` et `C` pour l'instant. En ASM, on préfère ne pas avoir de variables proprement dites (mais qu'est-ce qu'une variable ?) mais considérer ces entités `F` et `C` comme des éléments de l'algèbre, ici des éléments distingués (ou fonctions 0-aires).

L'algèbre que l'on considère sera donc :

$$(\mathbb{R}, +, ., /, 5, 9, 32, C, F).$$

Dans notre premier exemple, on peut considérer qu'au départ tout est fixé, sauf `F`. On dit, par exemple, que `C` est un **élément statique** (fixé) et que `F` est un **élément dynamique**, parce que sa valeur variera au cours du calcul.

Notion de calcul.- En ASM, un calcul sera une suite de structures du premier ordre de même signature et de même univers du discours. Au cours d'une étape de calcul les valeurs des éléments dynamiques (constante, fonction ou relation) auront variées de façon finie.

Dans notre cas, il n'y aura qu'une seule étape de calcul. Initialement l'élément distingué F n'aura pas de valeur définie et l'élément distingué C aura par exemple pour valeur 19. Après l'étape de calcul, l'élément distingué F aura 66 pour valeur.

5.1.2 Exemples de structures du premier ordre

Nous venons de voir l'intérêt des structures logiques du premier ordre pour les ASM. Avant d'en donner une définition formelle, commençons par en donner des exemples. Nous supposons connues pour cela les notions d'ensemble, de relations, d'applications et quelques structures algébriques simples comme celle de groupe.

Structures algébriques.- Un *groupe* est un couple $(G, +)$, avec G un ensemble non vide, $+$ une loi de composition interne sur G, vérifiant certaines conditions, à savoir que la loi est associative, admet un élément neutre et que chaque élément admet un opposé. Il existe beaucoup de groupes.

Un tel couple $(G, +)$ peut avoir d'autres propriétés. On parle de *magma* lorsqu'on n'impose aucune propriété, de *monoïde* lorsque la loi est associative et possède un élément neutre ...

Un groupe est quelquefois plutôt considéré comme un triplet $(G, +, 0)$, où 0 est l'élément neutre, donc un élément distingué de G. Ceci pourrait d'ailleurs sembler montrer que notre modélisation laisse à désirer, puisqu'à une notion apparemment bien définie nous associons deux structures ; mais en fait la seconde structure est associée à la première (techniquement comme *expansion par définition*).

Un *groupe ordonné* est un triplet $(G, +, \leq)$ tel que $(G, +)$ soit un groupe et que $\leq$ soit une relation d'ordre compatible avec la loi de composition.

Les ensembles de nombres.- On note $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{D}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ les ensembles des entiers naturels (à savoir 0, 1, 2, 3, ...), des entiers relatifs (à savoir ..., -2, -1, 0, 1, 2, ...), des décimaux, des rationnels (1/2, 2/3, 1, 3/4, ...), des réels et des complexes.

$(\mathbb{N}, +)$, $(\mathbb{N}, \leq, +)$ et $(\mathbb{N}, S, +, 0)$ sont des structures d'ensemble de base l'ensemble $\mathbb{N}$ des entiers naturels.

Le *corps réel* est la structure $(\mathbb{R}, +, .)$. Le *corps réel exponentiel* est la structure $(\mathbb{R}, +, ., exp)$, où exp est la fonction exponentielle.

Structures multi-bases.- Un *espace vectoriel* est un sextuplet $(K, V, \oplus, \otimes, +, \times)$, où $(K, \oplus, \otimes)$ est un corps commutatif, $+$ une loi de composition interne sur V et $\times$ une application de $K \times V$ dans V, ces deux dernières opérations vérifiant sept axiomes que nous ne rappelerons pas ici.

Structures du second ordre.- Toutes les structures que nous avons vues jusqu'ici sont des structures du premier ordre. Il existe des structures du second ordre dans lesquelles interviennent à la fois un ensemble de base E et un sous-ensemble de

l'ensemble $\mathcal{P}(E)$ des parties de E, par exemple les *espaces topologiques* dont nous ne rappellerons pas la définition ici.

5.1.3 Définition

5.1.3.1 Structure

De façon générale une structure est un ensemble non vide muni d'un certain nombre de lois de composition internes (non nécessairement binaires), d'un certain nombre de relations et d'un certain nombre d'éléments de G mis en exergue pour une raison ou une autre.

Définition 1.- *On appelle* **structure du premier ordre** *tout uplet :*

$$\mathcal{M} = (M,\ R_1, \ldots,\ R_h,\ f_1, \ldots,\ f_k,\ c_l, \ldots,\ c_m),$$

avec :

- *M un ensemble non vide, appelé l'***ensemble de base** *ou le* **domaine** *de la structure $\mathcal{M}$;*
- *pour $i \in [1, h]$, R_i est une relation sur M, donc $R_i \subseteq M^{\alpha_i}$, l'élément α_i de $\mathbb{N}^*$ étant l'***arité** *de R_i ;*
- *pour $i \in [1, k]$, f_i est une application à plusieurs variables à arguments et à valeurs dans M, donc $f_i : M^{\beta_i} \longrightarrow M$, l'élément β_i de $\mathbb{N}^*$ étant l'***arité** *de f_i ;*
- *pour $i \in [1, m]$, c_i est un élément de M ; les c_i sont appelés les* **éléments distingués**.

Remarques.-

1. Rien n'oblige à avoir des objets de base de chacune des sortes indiquées, on peut avoir $h = 0$ ou $k = 0$ ou $m = 0$, ou même les trois (mais ce dernier cas ne donne rien de vraiment intéressant).
2. On considère en général que la relation d'égalité $=$ est présente (on parle alors de **structure égalitaire**), ce que l'on ne mentionne pas nécessairement de façon explicite.
3. On considère aussi quelquefois des structures du premier ordre ayant un nombre infini de relations, de fonctions ou d'éléments distingués mais ce ne sera pas le cas ici (puisque ce n'est pas compatible avec la calculabilité).

5.1.3.2 Signature d'une structure

Les deux structures $(\mathbb{N}, +,\ .)$ et $(\mathbb{R}, +,\ .)$ ont quelque chose en commun, le fait qu'elles soient toutes les deux munies de deux lois de composition interne, bien que les propriétés ne soient pas toutes les mêmes. Ceci nous conduit à la notion de *signature* ou de *langage* d'une structure.

Notation.- Pour n entier naturel, nous noterons I_n l'ensemble des entiers naturels de 1 à n, soit $I_n = \{1, 2, ..., n\}$. En particulier $I_0 = \emptyset$.

Définition 2.- *On appelle* **signature de la structure** *du premier ordre :*

$$\mathcal{M} = (M,\ R_1, \ldots,\ R_h,\ f_1, \ldots,\ f_k,\ c_l, \ldots,\ c_m),$$

le triplet $((\alpha_1, \ldots, \alpha_h), (\beta_1, \ldots, \beta_k), m)$.

On appelle **signature** $\mathcal{L}$ *tout triplet $(r,\ f,\ m)$, avec m un entier naturel, r et f des applications de I_h dans $\mathbb{N}$ et de I_k dans $\mathbb{N}$, avec h et k des entiers naturels.*

Pour une signature $\mathcal{L}$ donnée, on appelle **$\mathcal{L}$-structure** *toute structure dont la signature est $\mathcal{L}$.*

Notation.- En pratique les applications finies r et f se noteront sous forme de listes $(\alpha_1, \ldots, \alpha_h)$ et $(\beta_1, \ldots, \beta_k)$.

Exemple.- Les deux structures de l'introduction sont des $((), (2, 2), 0)$-structures.
Les magmas, les monoïdes et les groupes sont des $((), (2), 0)$-structures.

Remarque.- Les structures du premier ordre d'ensemble de base dénombrable (ou plus exactement un langage formel) sont les *structures abstraites de données* de l'informatique.

5.1.3.3 Langage logique

Il faut bien avouer que la signature n'est pas psychologiquement très parlante, surtout lorsqu'on n'étudie que des structures de signatures très proches. On préfère alors donner des noms aux constantes R_i, f_j et c_n. Par exemple les lois de composition binaires sont la plupart du temps notées : $+$, . , *, T, $\perp$, $\oplus$, $\otimes$, et autres notations standard.

Définition 3.- *On appelle* **alphabet propre d'un langage logique du premier ordre** *tout ensemble fini non vide $\mathcal{L}$. On indique en plus à propos de chaque élément de $\mathcal{L}$ qu'il est :*
- *soit un* **prédicat d'arité** *n, pour un certain entier naturel non nul n ;*
- *soit un* **symbole de fonction d'arité** *n, avec la même condition sur n ;*
- *soit un* **symbole de constante**.

Exemples.-
1. L'alphabet des magmas et des groupes est $\{+\}$, où $+$ est un symbole de fonction binaire.
2. L'alphabet du corps exponentiel ordonné est $\{+, ., exp, \leq\}$, où $+$ et . sont des symboles de fonctions binaires, exp est un symbole de fonction unaire et $\leq$ un symbole de relation binaire.

Remarque.- Il ne faut pas confondre les *langages logiques* (dont nous ne connaissons pour l'instant que la notion d'alphabet propre) et les *langages formels* d'alphabet A (c'est-à-dire des ensembles de mots sur A).

Définition 4.- *Si $\mathcal{L}$ est un alphabet logique alors une* **$\mathcal{L}$-structure** *est une structure logique ayant autant d'éléments distingués que $\mathcal{L}$ a de symboles de constante, autant de relation n-aires que $\mathcal{L}$ a de prédicats n-aires et autant de fonctions n-aires que $\mathcal{L}$ a de symboles de fonctions n-aires, et ceci pour tout entier naturel non nul n.*

Exemple.- Un magma est une $\{+\}$-structure, si on a spécifié que $+$ est un symbole de fonction binaire.

5.1.3.4 Termes d'un langage logique

Lorsqu'on veut parler des propriétés d'une structure, on fait abstraction de tout l'environnement extérieur. Nous ne voulons parler que des propriétés ne faisant intervenir que les éléments spécifiés dans son langage. Cette notion intuitive se formalise de la façon que nous allons voir maintenant.

Quels sont les objets de M, le domaine de la structure $\mathcal{M}$, dont on peut parler immédiatement ? On dit aussi dont on peut parler explicitement.

Dans $(\mathbb{R}, +, ., 0, 1)$ on pourra parler de $1 + 1$ (soit 2), ou de $(1 + 1).(1 + 1 + 1)$. Mais on a aussi souvent envie de prendre un *élément générique* ("Soit x un réel quelconque") et en faire quelque chose au sein de cette structure. Ceci conduit à la notion de *variable* et des éléments que l'on peut distinguer avec cette ou ces variables et la structure. On pourra par exemple parler de $x.x+(1+1+1).x+(1+1)$, noté plus simplement $x^2 + 3.x + 2$, mais pas de $sin(x)$ par exemple.

La notion intuitive se formalise naturellement en celle de *terme* et de *terme clos* définis précisément de la façon suivante.

Définition 5.- *On introduit un ensemble dénombrable de symboles nouveaux, notés en général x_n, pour n entier naturel, et appelés* **variables**.

Remarques.-

1. Si on veut que les mots auxquels nous nous intéressons soient des éléments d'un langage formel, il faut que ce soient des mots sur un alphabet fini, ce qui n'est pas le cas si l'on considère l'ensemble des variables. Mais, en fait, une variable peut être considérée comme un mot sur l'alphabet {X, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, par exemple, c'est-à-dire un X suivi de la représentation décimale de n.
2. Ces variables, comme nous l'avons vu dans l'introduction, joueront le rôle d'élément générique de l'ensemble de base, et non par exemple de relation sur cet ensemble de base. C'est dans ce sens que l'on parle de *langage du premier ordre*, pour le distinguer des langages qui permettent de parler, par exemple, des sous-ensembles de l'ensemble de base (indispensable pour parler de sous-groupes). Les *langages d'ordre supérieur* sont importants par ailleurs mais ne nous intéresseront pas ici.

Définition 6.- *On définit l'ensemble des* **termes** *d'un langage logique du premier ordre $\mathcal{L}$ de la façon suivante :*

1. *Toute variable est un terme ;*
2. *Toute constante d'individu de $\mathcal{L}$ est un terme ;*
3. *Si f est un symbole fonctionnel n-aire et $t_1, \ldots, t_n$ des termes de $\mathcal{L}$ alors il en est de même de $f(t_1, \ldots, t_n)$;*
4. *Tous les termes sont obtenus à partir de 1), 2) et 3).*

Un **terme** *est* **clos** *(*ground term *en anglais) si, et seulement si, il ne contient aucune occurrence de variable.*

Exemple.- Si on considère le langage $\mathcal{L} = \{+, ., 0, 1\}$ avec $+$ et $.$ des symboles fonctionnels binaires, 0 et 1 des constantes d'individus, alors les mots suivants sont des termes :

$$+(1, 1), .(1, +(0, 1)), +(x_3, 1).$$

Les deux premiers termes sont clos.

Par contre $+(2, 0)$ et $+(0, 1, 1)$ ne sont pas des termes. Le premier mot car 2 n'est pas un symbole permis, le second mot car la condition sur l'arité n'est pas respectée.

Notation.- En pratique, lorsque τ est un symbole fonctionnel binaire, on notera plutôt $(a\tau b)$ (on parle de **notation infixe**) que $\tau(a, b)$ (on parle de **notation fonctionnelle**).

Exemple.- Les termes ci-dessus seront plutôt notés en pratique de la façon suivante, en enlevant la paire de parenthèses la plus externe :

$$1+1,\ 1.(0+1),\ x_3+1.$$

5.1.3.5 Interprétation d'un terme

Un terme est un mot d'un certain langage, mais nous voulons qu'il représente quelque chose de plus concret pour la structure, ce que nous appellerons son *interprétation dans la structure*. Celle-ci est intuitivement immédiate, mais encore faut-il la définir de façon mathématique.

Une alternative se pose d'ailleurs : faut-il interpréter les variables pour qu'un terme représente un élément de l'ensemble de base, ou faut-il les laisser non interprétées pour qu'elles reprsentent une fonction ? La premire solution s'est révélée plus intéressante.

Définition 7.- *Soient $\mathcal{L}$ un langage logique et $\mathcal{M}$ une $\mathcal{L}$-structure. On appelle* **$\mathcal{L}$-interprétation** *de $\mathcal{L}$ dans $\mathcal{M}$ toute application de $\mathcal{L}$ dans l'ensemble des constantes de $\mathcal{M}$ telle que :*

- *l'image d'un prédicat n-aire soit une relation n-aire ;*
- *l'image d'un symbole fonctionnel n-aire soit une application n-aire ;*
- *l'image d'une constante d'individu soit un élément distingué.*

Pour tout symbole c de $\mathcal{L}$, on notera $\overline{c}^{\mathcal{M}}$ son image par l'interprétation.

Exemple.- Si $\mathcal{L} = (S, +, .)$, avec S un symbole fonctionnel unaire, $+$ et $.$ des symboles fonctionnels binaires, alors $\mathcal{N} = (\mathbb{N},\ S, +,\ .)$, avec S la fonction successeur, $+$ et $.$ l'addition et la multiplication sur les entiers naturels, est une $\mathcal{L}$-structure. Remarquons que les symboles S, $+$ et $.$ jouent chacun deux rôles différents : un nom pour un symbole du langage et un nom pour un élément de la structure. Sans indication contraire, on prend en général : $\overline{+}^{\mathcal{M}} = +$ et $\overline{.}^{\mathcal{M}} = .$; mais rien n'empêche *a priori* de faire le contraire.

Remarque.- En général lorsque nous parlerons d'une $\mathcal{L}$-structure $\mathcal{M}$, nous supposerons que nous avons choisi une interprétation sans que nous ne le précisions explicitement.

Définition 8.- *Étant donnée une $\mathcal{L}$-structure $\mathcal{M}$, on appelle* **assignation** *(ou* **affectation***) dans $\mathcal{M}$ toute application α de l'ensemble V des variables dans le domaine M de $\mathcal{M}$.*

Exemple.- Si $\mathbb{N}$ est le domaine, une première assignation α est définie par $\alpha(x_n) = n$. Une autre est définie par $\alpha(x_n) = n!$.

Remarque.- L'application α n'a aucune raison d'être explicite, ni même d'être récursive. Nous utilisons cette notion surtout en théorie, en raisonnant sur l'ensemble des assignations et sans avoir à en donner une explicitement.

Proposition 1.- *Étant données une $\mathcal{L}$-structure $\mathcal{M}$, une interprétation de $\mathcal{L}$ dans $\mathcal{M}$ et une assignation α dans $\mathcal{M}$, il existe un et un seul prolongement $\hat{\alpha}$ de α qui soit une application de l'ensemble $\tau_{\mathcal{M}}$ des termes de $\mathcal{L}$ dans M, le domaine de $\mathcal{M}$, tel que :*

1. *$\hat{\alpha}(x) = \alpha(x)$ pour toute variable x ;*
2. *$\hat{\alpha}(c) = \overline{c}^{\mathcal{M}}$ pour toute constante d'individu c de $\mathcal{L}$;*

3. $\hat{\alpha}(t) = \overline{f}^{\mathcal{M}}(\hat{\alpha}(t_1), \ldots, \hat{\alpha}(t_n))$ *si* t *est un terme de la forme* $f(t_1, \ldots, t_n)$.

Exercice.- Démontrer cette proposition.

Définition 9.- *Cette application* $\hat{\alpha}$ *s'appelle l'***affectation des termes**, *l'assignation* α *étant donnée.*

Proposition 2..- *Soient* $\mathcal{M}$ *une* $\mathcal{L}$*-structure,* t *un terme, les variables ayant une occurrence dans* t *étant parmi* $x_0, x_l, \ldots, x_n$*, et* α *et* β *des assignations dans* $\mathcal{M}$ *prenant les mêmes valeurs pour les variables* $x_0, x_l, \ldots, x_n$*. Alors* $\hat{\alpha}(t) = \hat{\beta}(t)$.

Exercice.- Démontrer cette proposition.

Définition 10.- *Soit* t *un terme non clos du langage* $\mathcal{L}$ *dont la variable libre d'indice le plus élevé est* x_n*. L'***interprétation du terme** t *pour une* $\mathcal{L}$*-structure* $\mathcal{M}$ *est l'application* $(n+l)$*-aire qui à* $(a_0, \ldots, a_n) \in M^{n+1}$ *associe* $\hat{\alpha}(t)$*, pour* α *assignation telle que :*

$$\alpha(x_0) = a_0, \ldots, \alpha(x_n) = a_n.$$

Exemple.- Pour la structure $\mathcal{N} = (\mathbb{N}, S, +, .)$, le terme $x_0 + x_1$ s'interprète par l'addition.

Remarque.- Il n'est pas difficile de démontrer que l'ensemble des interprétations des termes de $(\mathbb{R}, +, ., 0, 1)$ est l'ensemble des fonctions polynomiales à plusieurs variables à coefficients entiers naturels. On n'obtient donc pas ainsi toutes les fonctions réelles (à plusieurs) variables réelles.

Ceci montre bien que la notion de structure nous permet de *cibler une partie du monde*, d'où son intérêt.

5.1.3.6 Structure fonctionnelle

Définition 11.- *Une* **structure** *du premier ordre est* **fonctionnelle** *si elle n'a pas de prédicat.*

Exemple.- Les structures de groupe, d'anneau, de corps sont des structures fonctionnelles.

Contre-exemple.- La structure de corps réel ordonné $(\mathbb{R}, +, ., \leq)$ n'est pas une structure fonctionnelle.

Remarques.-

1. La propriété "être fonctionnelle" pour une structure logique du premier ordre ne porte que sur la signature de celle-ci.
2. On peut toujours s'arranger pour qu'une structure soit fonctionnelle. Il suffit de considérer deux éléments distingués, appelés en général `true` et `false` ou 0 et 1. Une relation est alors une fonction qui ne prend que ces deux valeurs. À toute relation n-aire R on associe alors la fonction f_R définie par :

   $f_R(a_1, \ldots, a_n) = true$ si, et seulement si, $R(a_1, \ldots, a\_n)$ est vrai.

   Une telle **fonction** est quelquefois dite **relationnelle** ou **booléenne**.
3. Dans le cas extrême, on ne considère que des fonctions. Un élément distingué sera alors considéré comme une fonction 0-aire.

5.1.3.7 Isomorphisme

Le lecteur a déjà rencontré la notion d'isomophisme de groupe, d'anneaux et pour d'autres structures algébriques.

Définition 12.- *Soient*

$$\mathcal{A} = (A,\ R_1, \ldots,\ R_h,\ f_1, \ldots,\ f_k,\ c_1, \ldots,\ c_m)$$

et

$$\mathcal{B} = (B,\ S_1, \ldots,\ S_h,\ g_1, \ldots,\ g_k,\ d_1, \ldots,\ d_m)$$

des structures logiques du premier ordre de même signature $\mathcal{L}$.

Une bijection ϕ de A sur B est un **$\mathcal{L}$-isomorphime** *de $\mathcal{A}$ sur $\mathcal{B}$ si, et seulement si :*

- $\phi(c_i) = d_i$ *pour* $i \in [1, m]$;
- $\phi(f_i(\overline{x})) = g_i(\overline{\phi}(\overline{x}))$ *pour* $i \in [1, k]$ *et tout uplet adapté à l'arité de* f_i ;
- $R_i(\overline{x})$ *est vrai si, et seulement si,* $S_i(\overline{\phi}(\overline{c}))$ *pour* $i \in [1, h]$ *et tout uplet adapté à l'arité de* R_i.

5.1.4 Structures effectivement calculables

Nous venons de présenter la notion générale de structure du premier ordre. Comment faire des calculs dans de telles structures ? En fait, on ne pourra jamais faire de calculs effectifs que dans les *structures effectivement calculables*, ou *structures récursives* au sens des logiciens.

Rien ne spécifie en ASM que les structures considérées soient récursives. La présentation des ASM avec d'autres structures correspond aux *calculs relatifs* (les machines de Turing avec oracles).

5.1.5 Historique

La notion générale de structure a été dégagée par Alfred Tarski en 1933 [Tar33] après une longue évolution.

On peut considérer, rétrospectivement, que les mathématiciens ont travaillé sur des *structures concrètes* données sans expliciter la notion proprement dite. C'est une façon de réécrire l'histoire.

La notion de *structure abstraite* apparaît en algèbre à propos des grandes structures algébriques. Ce n'est pas le moment ici d'en faire l'historique[1].

Prenons, à titre d'exemple, la première structure algébrique à être dégagée, celle de groupe. La notion de groupe abstrait a été introduite par Augustin Cauchy en 1821. Cependant l'application qui l'a fait émergée est due à Évariste Galois : rappelons, qu'après avoir résolu les équations du troisième et du quatrième degré au XVI^e siècle, les mathématiciens se demandaient comment résoudre les autres équations et, pour commencer, celles du cinquième degré ; Ruffini démontre en 1799 qu'on ne peut pas résoudre l'équation du cinquième degré par radicaux (c'est-à-dire que les racines ne peuvent pas être exprimées à l'aide des coefficients, de l'addition, de la soustraction, de la multiplication, de la division et des racines k-ièmes [pour k inférieur à cinq, cette dernière restriction ne jouant pas de rôle essentiel]) ; sa démonstration n'est pas compréhensible mais Abel la précise en 1824. Évariste Galois caractérise en 1832 les équations qui peuvent être résolues, qu'elles soient du cinquième degré ou d'un degré plus élevé : il associe un groupe à

[1] Il n'existe pas, malheureusement, d'historique des notions des structures algébriques en tant que structures.

tout polynôme ; le groupe doit être résoluble pour que l'équation le soit. Qu'importe ici la façon dont on associe ce groupe, ce qu'est un groupe *résoluble* (en un sens bien défini) et comment savoir si le groupe est résoluble ou non. Nous en retenons que GALOIS a besoin de la notion générale de *groupe*, en fait de la notion de sous-groupe des groupes de permutation. Il ne peut pas travailler sur un nombre fini de structures concrètes données *a priori*. Il a besoin de la notion de structure en tant que variable.

Pour GALOIS, un *groupe* est un sous-ensemble de l'ensemble des permutations clos par composition.

D'autres instances de groupes apparaissent, en particulier les groupes de symétrie des figures du plan et de l'espace, en fait sous-groupes de l'ensemble des isométries.

La définition abstraite de structure de groupe (et non plus comme sous-groupe de quelque chose) apparaît à la fin du dix-neuvième siècle.

De même d'autres structures algébriques sont dégagées : celles de corps, d'anneau, d'algèbre, d'espace vectoriel...

Les notions de structures abstraites particulières se dégagent petit à petit, dont un exposé apparaît dans le livre *Modern Algebra* de VAN DER WAERDEN paru en 1930, mais pas celle de *structure*. Une discipline, appelée *algèbre universelle* s'y consacre pour les structures algébriques uniquement.

Alfred TARSKI a besoin de donner la définition générale de ce qu'est une structure du premier ordre en 1933 pour préciser la notion de *vérité* d'une théorie logique (du premier ordre).

Le groupe de mathématicien BOURBAKI popularisa cette notion générale de structure, en distinguant les structures algébriques, les structures d'ordre et les structures topologiques, à partir de la fin des années 1930.

On considèrera surtout des **structures concrètes** à propos des ASM, c'est-à-dire des structures dont l'ensemble des base, les fonctions et les relations sont déterminés et bien connus : BOURBAKI parle de **structure monomorphe**, par exemple $(\mathbb{N}, +, .)$ ne devant avoir qu'une seule interprétation, par opposition aux **structures polymorphes**, par exemple celle de groupe commutatif, ayant plusieurs interprétations possibles. Nous avons déjà dit que seules des structures concrètes étaient considérées jusqu'à l'apparition des structures algébriques. En fait les travaux du dix-neuvième siècle sur les géométries non euclidiennes ont jetés un doute sur l'existence de structures vraiment monomorphes. Pire même, les travaux du vingtième siècle ont jeté un doute sur le fait que la structure sur les entiers naturels par excellence, $(\mathbb{N}, +, .)$, puisse être monomorphe, mais ceci est une autre histoire.

5.2 Définition d'un programme ASM

Nous allons préciser la syntaxe des ASM, c'est-à-dire la notion de programme.

5.2.1 Vocabulaire d'ASM

Définition 13.- *Un* **vocabulaire**, *ou* **signature**, **d'ASM** *est une signature $\mathcal{L}$ de structure du premier ordre fonctionnelle pure (c'est-à-dire sans symbole de relation ni symbole d'éléments distingués), finie, égalitaire et booléenne, c'est-à-dire ayant les symboles (ou fonctions 0-aires)* **true**, **false** *et* **null**, *la fonction unaire relationnelle* **Boolean** *et les fonctions booléennes pour les connecteurs logiques (*__not__, **and** *et* **or**).

Commentaires.-

1. Le fait que la signature soit fonctionnelle n'est pas très important. C'est juste pour simplifier la définition d'un pas de calcul comme nous le verrons ci-dessous.
2. Dans la mesure où la signature est fonctionnelle, on comprend l'intérêt qu'elle soit booléenne pour pouvoir manipuler les relations.
3. La constante `null` (encore appelée **undef**) sert à désigner les éléments dont la valeur n'est pas (encore) définie.

5.2.2 Mise à jour

Nous avons vu que l'instruction la plus élémentaire des ASM est l'affectation ou mise à jour. Formalisons la syntaxe de celle-ci. On parle de **règle** plutôt que d'instruction en ASM.

Définition 14.- *Soit $\mathcal{L}$ un vocabulaire d'ASM. Une* **règle de mise à jour** *(*update rule *en anglais) est une expression de la forme :*

$$f(t_1, \ldots, t_n) := t_0$$

où f est un symbole fonctionnel n-aire et $t_0, t_1, \ldots, t_n$ des termes clos de $\mathcal{L}$.

Commentaires.- Cela correspond à la notion d'affectation que nous avons vue avec AsmL. On pourra peut-être s'étonner du fait qu'on n'a que des termes clos alors que l'on parle de variable en AsmL. Dans un programme donné, les variables sont en nombre fini. On a vu, lors de l'introduction au début de ce chapitre, que l'on distingue entre les éléments statiques et les éléments dynamiques, autrement dit les constantes et les variables. Une constante ou une variable sera un élément distingué (une fonction 0-aire pour des raisons techniques). Les expressions sont donc des termes ne faisant intervenir que des éléments distingués, c'est-à-dire des termes clos au sens des logiciens.

5.2.3 Simultanéïté

Nous avons vu que la structure de contrôle par défaut, qui correspond à un bloc d'instructions, est la simultanéïté. Formalisons la syntaxe des blocs d'instructions. Rappelons qu'en AsmL, on jouait sur l'indentation. Ceci n'est pas très pratique pour une définition formelle. On va donc introduire les délimiteurs de bloc **par** et **endpar**. À l'origine *par* rappelle le début du mot *parallèle* ; nous lui préférons celui de *parenthèse*.

Définition 15.- *Soit $\mathcal{L}$ un vocabulaire d'ASM. Si $R_1, \ldots, R_k$ sont des règles du vocabulaire $\mathcal{L}$, alors il en est de même de l'expression :*

par
R_1
.
.
.
R_k
endpar

appelée **bloc** *ou* **règle de simultanéïté**.

Si $k = 0$, il s'agit de l'instruction vide, quelquefois notée **skip**.

5.2.4 Test et alternative

Nous avons vu que le test et l'alternative sont des structures de contrôle utilisées en AsmL comme dans beaucoup d'autres langages. Comme d'habitude, il suffit du test ou de l'alternative.

Définition 16.- *Un* **terme** *est dit* **booléen** *s'il a la valeur* `true` *ou* `false`.

Définition 17.- *Soit $\mathcal{L}$ un vocabulaire d'ASM. Si ϕ est un terme booléen et R_1 et R_2 des règles du vocabulaire $\mathcal{L}$, alors l'expression :*
if ϕ then R_1
else R_2
endif
est également une règle du vocabulaire $\mathcal{L}$.

Notations.- On omet les mots clés `endpar` et `endif` lorsque le contexte est clair. De même on omet le mot clé `else` si R_2 est `skip`.

5.2.5 Pas d'autre structure de contrôle

Dans les langages structurés, on a une autre structure de contrôle, l'itération ou répétition ou boucle. En AsmL nous avons vu plusieurs formes de la structure de contrôle *step*, dont la forme fondamentale *step until fixpoint*. On n'en a pas besoin explicitement en ASM. La notion de *calcul* jouera ce rôle.

Définition 18.- *Soit $\mathcal{L}$ un vocabulaire d'ASM. Un* **programme** *de vocabulaire $\mathcal{L}$ est une règle (composée) de ce vocabulaire.*

5.3 Exécution d'un programme

Précisons maintenant la sémantique des ASM, c'est-à-dire la notion d'exécution d'un programme.

5.3.1 Étude d'un exemple

Considérons l'exemple du calcul du pgcd :

```
if b = 0 then d := a
         else par
              a := b;
              b := a mod b;
              endpar
endif
```

Quel est l'effet de l'exécution d'un tel programme ? On peut considérer qu'il s'agit d'une suite d'algèbres de même signature :

$$(\mathcal{A}_n)_{n<\kappa}$$

indexée par les entiers naturels, κ étant soit un entier naturel, soit ∞ suivant que l'exécution est finie ou infinie.

L'**algèbre initiale** $\mathcal{A}_0$ précise les valeurs initiales des entités dynamiques (ainsi, bien sûr, que les valeurs des entités statiques). On passe d'une algèbre $\mathcal{A}_n$ à l'algèbre $\mathcal{A}_{n+1}$ en appliquant les règles édictées par le programme.

Pour revenir à notre exemple, considérons l'algèbre initiale :

$$\mathcal{A}_0 = (\mathbb{N},\ mod,\ a,\ b,\ d),$$

où $\mathbb{N}$ est l'ensemble des entiers naturels, *mod* est l'opération binaire infixe "reste dans la division euclidienne du premier opérande par le second", a, b et c des constantes dynamiques dont les valeurs sont 4, 6 et sans importance.

Pour le programme ci-dessus, dans l'algèbre $\mathcal{A}_1$, les constantes a, b et d ont pour valeurs 6, 4 et d initial.

Dans l'algèbre $\mathcal{A}_2$, les constantes a, b et d ont pour valeurs 4, 2 et le d initial.

Dans l'algèbre $\mathcal{A}_3$, les constantes a, b et d ont pour valeurs 2, 0 et le d initial.

Dans les algèbres suivantes, les constantes a, b et d ont pour valeurs 2, 0 et 2. La valeur du pgcd est donc 2.

Rien ne dit, pour l'instant, à quel moment on doit s'arrêter.

Formalisons maintenant ceci.

5.3.2 Définition formelle

Définition 19.- *Si $\mathcal{L}$ est une signature d'ASM, une $\mathcal{L}$-structure est aussi appelée un* **état abstrait** *d'ASM, plus spécifiquement un $\mathcal{L}$-état.*

Commentaire.- Nous avons vu à la section 5.1.1.2 en quoi cela correspond à notre notion intuitive d'état lors d'un calcul.

Définition 20.- *Soient $\mathcal{L}$ un vocabulaire d'ASM et A un ensemble non vide. On appelle* **ensemble de modifications** *tout ensemble fini de triplets :*

$$(f,\ \overline{a},\ a),$$

où f est un élément de $\mathcal{L}$, $\overline{a}$ un n-uplet de A, avec n l'arité de f, et a un élément de A.

Définition 21.- *Soient $\mathcal{L}$ un vocabulaire d'ASM, $\mathcal{A}$ un $\mathcal{L}$-état et Π un $\mathcal{L}$-programme. On note $\Delta_\Pi(\mathcal{A})$ l'ensemble de modifications défini de la façon suivante par récurrence sur la structure de Π :*

1. *Si Π est une règle de mise à jour :*

   $$f(t_1, \ldots,\ t_n) := t_0$$

   alors, en notant $a_1 = \overline{t_1}^{\mathcal{A}}, \ldots,\ a_n = \overline{t_n}^{\mathcal{A}}$ et $a = \overline{t_0}^{\mathcal{A}}$, l'ensemble de modifications est le singleton :

   $$\{(f,\ (a_1, \ldots,\ a_n),\ a)\}$$

2. *Si Π est un bloc :*

   $$par\ R_1 \ldots\ R_n\ endpar$$

   alors l'ensemble de modifications est la réunion :

   $$\Delta_{R_1}(\mathcal{A}) \cup \ldots \cup \Delta_{R_n}(\mathcal{A})$$

   pour n non nul et l'ensemble vide sinon.

3. *Si Π est un test :*

   $$if\ \phi\ then\ R$$

   on commence par évaluer l'expression $\overline{\phi}^{\mathcal{A}}$. Si elle est fausse alors l'ensemble de modifications est vide, sinon il est égal à :

   $$\Delta_R(\mathcal{A}).$$

Remarques.-

1. C'est à propos de cette définition qu'on voit l'intérêt technique de ne considérer que des structures fonctionnelles permettant les fonctions 0-aires. Ceci nous évite à distinguer moult cas.
2. Nous avons donné la définition dans le cas du test. On a une définition analogue pour l'alternative, si on préfère celle-ci comme structure de contrôle primitive.
3. On peut montrer facilement que $\Delta_\Pi(\mathcal{A})$ est un ensemble fini.

Définition 22.- *Un ensemble de modifications est* **incohérent** *s'il comprend deux éléments* $(f,\ \overline{a},\ a)$ *et* $(f,\ \overline{a},\ b)$ *avec* $a \neq b$.

Remarque.- L'incohérence ne peut provenir que d'un bloc.

Définition 23.- *Soient* $\mathcal{L}$ *un vocabulaire d'ASM,* Π *un* $\mathcal{L}$*-programme et* $\mathcal{A}$ *un* $\mathcal{L}$*-état.*

Si $\Delta_\Pi(\mathcal{A})$ *est cohérent, le* **transformé** $\tau_\Pi(\mathcal{A})$ *de* $\mathcal{A}$ *par* Π *est la* $\mathcal{L}$*-structure* $\mathcal{B}$ *définie de la façon suivante :*

- *l'ensemble de base de* $\mathcal{B}$ *est l'ensemble de base* A *de* $\mathcal{A}$ *;*
- *pour tout élément* f *de* $\mathcal{L}$ *et tout élément* $\overline{a} = (a_1, \ldots,\ a_n)$ *de* A^n*, si* n *est l'arité de* f *:*
  - *si* $(f,\ \overline{a},\ a) \in \Delta_\Pi(\mathcal{A})$ *pour un certain* $a \in A$*, alors :*

$$\overline{f}^{\mathcal{B}}(\overline{a}) = a\ ;$$

  - *sinon :*

$$\overline{f}^{\mathcal{B}}(\overline{a}) = \overline{f}^{\mathcal{A}}(\overline{a}).$$

Si $\Delta_\Pi(\mathcal{A})$ *est incohérent, on considère qu'on a un point fixe :* $\tau_\Pi(\mathcal{A}) = \mathcal{A}$.

Remarque.- Si $\Delta_\Pi(\mathcal{A})$ est incohérent, le transformé n'est pas défini et l'exécution du programme s'arrête. Nous avons vu que dans le cas de AsmL, ceci conduit à une exception.

Définition 24.- *Soient* $\mathcal{L}$ *un vocabulaire d'ASM,* Π *un* $\mathcal{L}$*-programme et* $\mathcal{A}$ *un* $\mathcal{L}$*-état. On appelle* **calcul** *la suite*

$$(\mathcal{A}_n)_{n<\kappa}$$

définie par :

- $\mathcal{A}_0 = \mathcal{A}$ *(appelée* **algèbre initiale** *du calcul) ;*
- $\mathcal{A}_{n+1} = \tau_\Pi(\mathcal{A}_n)$ *pour* $n \in \mathbb{N}$.

Remarque.- Si, dans un calcul, on a $\mathcal{A}_{n+1} = \mathcal{A}_n$ pour un certain n alors $\mathcal{A}_p = \mathcal{A}_n$ pour $p \geq n$.

Définition 25.- *Un calcul* **s'arrête** *s'il existe un point fixe* $\mathcal{A}_{n+1} = \mathcal{A}_n$.

5.3.3 Cas de AsmL : forme normale

Introduction.- Dans le cas de AsmL nous avons vu que l'on pouvait utiliser l'itération plusieurs fois. Émuler un calcul d'ASM tel que nous venons d'en donner la définition revient à une **mise sous forme normale** :

```
methode1
...
methoden

Main()
  initially ...
  ...
  initially
  step until fixpoint
    programme ASM
```

Les méthodes (ou la classe) servent à préciser l'algèbre statique. Les déclarations `initially` permettent de spécifier l'algèbre initiale.

Exemple.- Reprenons le cas du calcul du pgcd et mettons-le sous forme normale :

```
// pgcdFN.asml

Main()
  initially a as Integer = 0
  initially b as Integer = 0
  initially d as Integer = 0
  initially s as String = ""
  initially mode = "initial"
  step until fixpoint
    if mode = "initial"
      Write("a = ")
      mode := "lireA"
    if mode = "lireA"
      s := ReadLine()
      mode := "promptB"
    if mode = "promptB"
      a := ToInteger(s)
      Write("b = ")
      mode := "lireB"
    if mode = "lireB"
      s := ReadLine()
      mode := "entierB"
    if mode = "entierB"
      b := ToInteger(s)
      mode := "calcul"
    if mode = "calcul"
      if b = 0 then
                d := a
                mode := "affiche"
              else
                a := b
                b := a mod b
    if mode = "affiche"
```

```
WriteLine("pgcd(a,b) = " + d)
```

Exercice.- *Mettre sous forme normale tous les programmes AsmL que vous avez écrit jusqu'ici.*

5.3.4 ASM séquentielle

Nous allons enfin pouvoir donner la définition formelle de ce qu'est une ASM.

Définition 26.- *Une* **machine à états abstraits séquentielle** *(*sequential abstract state machine *en anglais), ou* **ASM séquentielle**, *A de langage $\mathcal{L}$ est un quadruplet* $(\Pi, \mathcal{S}(A), \mathcal{I}(A), \tau_A)$, *où :*

- Π *est un programme de vocabulaire* $\mathcal{L}$ *;*
- $\mathcal{S}(A)$ *est un ensemble de* $\mathcal{L}$*-structures, l'***ensemble des états**, *clos par isomorphime et par l'opération de transformation* τ_Π *;*
- $\mathcal{I}(A)$ *est un sous-ensemble de* $\mathcal{S}(A)$ *clos par isomorphime, appelé l'***ensemble des états initiaux** *;*
- τ_A *est la restriction de* τ_Π *à* $\mathcal{S}(A)$.