Rapport de stage
DEA Algorithmique 2001/2002

Langages de données

Manuel BACLET

Sous la direction d'Antoine PETIT

Laboratoire Spécification et Vérification, ENS Cachan

Table des matières

Introduction

Les langages réguliers, pierre de voûte de l'informatique théorique, peuvent être définis de différentes manières : automates finis, expressions rationnelles, logique monadique du second ordre, logiques temporelles, monoïdes finis... (voir, par exemple, [RS97]). On peut de plus étendre ces équivalences à des fragments de la classe des langages réguliers. Il est ainsi possible de décider si un langage reconnaissable est définissable par une formule de logique temporelle linéaire (LTL, [Pnu77]), en combinant deux résultats : une équivalence entre langages définissables par une formule de LTL et langages définissables par une fornule logique du premier ordre [Kam68] et une équivalence entre langages définissables par une formule du premier ordre et langages reconnus par un monoïde apériodique [Sch65].

Dans le cadre des langages temporisés, le modèle des automates temporisés, proposé par Alur et Dill en 1990 dans [AD90], s'est révélé être un modèle adapté à l'étude des systèmes temporisés. Il a depuis été largement étudié. Néanmoins, la théorie des langages temporisés n'a pas l'élégance de celle des langages réguliers ; en particulier, il n'existe pas de caractérisation algébrique des langages reconnus par un automate temporisé.

Afin de combler ces lacunes, P. Bouyer, A. Petit et D. Therien ont proposé dans [BPT01] une notion plus générale que les langages temporisés : les *langages de données* ainsi que deux systèmes de reconnaissance dont les puissances sont équivalentes : les automates de données et les mécanismes à monoïde fini. Ces deux mécanismes sont munis d'une mémoire : un ensemble fini de registres dans lesquels il est possible d'écrire des données. Une *mise à jour* des registres est définie comme un sous-ensemble des registres ; étant sous-entendu qu'on écrit la donnée courante sur dans les registres de ce sous-ensemble.

Pour définir les langages de données, on utilise un alphabet fini Σ et un ensemble de données $\mathcal{D}$ (qui peut être un domaine de temps mais plus généralement n'importe quel ensemble). Un *mot de données* est une suite de paires (a, d) où $a \in \Sigma$ et $d \in \mathcal{D}$.

Nous proposons dans une première partie de définir les notions présentées cidessus et d'énoncer les principaux résultats existant sur les langages de données et de les comparer aux résultats similaires existant sur les langages réguliers. Ces rappels sont entièrement extraits de [BPT01]. Nous étudions ensuite la robustesse du modèle en regardant si des mises à jour plus générales modifient le pouvoir de reconnaissance du modèle. Ainsi, nous montrons qu'autoriser des permutations des valeurs des registres, qui éventuellement dépendent des valeurs courantes des registres et de la donnée courante, ne modifie pas le pouvoir d'expression du modèle. Nous présentons ensuite un modèle de mises à jour qui rend le modèle plus expressif. Finalement, après avoir rappelé des notions de la théorie des réseaux de Petri, nous montrons que les automates de données permettent de simuler ces réseaux.

Dans une seconde partie, nous nous intéressons à un langage logique, proposé par P. Bouyer dans [Bou02], qui capture la puissance des automates de données. Il s'agit d'une logique proche de l'esprit de MSO. Comme dans le cas des langages réguliers, nous montrons qu'il est possible de caractériser logiquement les langages reconnus par un mécanisme à monoïde apériodique. La preuve est en deux parties : on commence par caractériser logiquement les langages reconnus par un mécanisme non déterministe à monoïde apériodique dont les mises à jour sont de la forme $up_{m,a} = \mathcal{P}(1..k)$ où k est le nombre de registres du mécanisme ; le cas général est étudié ensuite.

Finalement, nous démontrons un théorème de Kleene/Büchi sur les automates de données en généralisant un résultat similaire pour les automates temporisés proposé par P. Bouyer et A. Petit dans [BP02]. Pour cela, on définit les *langages de registres* qui correspondent aux langages d'horloges pour les langages temporisés. On présente ensuite les opérations sur les langages de registres : concaténation, itérations finie et infinie. On définit ensuite les expressions rationnelles ; contrairement au cas des langages réguliers, on a besoin, ici, d'un nombre infini d'objets de base qu'on combine à l'aide des opérations d'union, de concaténation, d'itérations finie et infinie. Cependant, une expression rationnelle ne contient qu'un nombre fini d'objets de base, comme pour les langages réguliers. La notion de langage de registres reconnaissable est définie à l'aide des automates de données. Pour démontrer que tout langage rationnel est reconnaissable, on prouve que la classe des langages reconnaissables est close pour les opérations de concaténation, d'union, d'itérations finie et infinie et que les objets de base dénotent des langages reconnaissables, comme pour les langages réguliers. Pour prouver la réciproque (i.e. tout langage reconnaissable est rationnel), on généralise une preuve du théorème de Kleene utilisant une technique de résolution d'équation sur les mots, voir, par exemple [Brz62]. On montre enfin un théorème de Kleene sur les langages de registres de mots finis ou infinis, ce qui nous permet d'obtenir un «théorème de Kleene» pour les langages de données à l'aide d'une projection des langages de registres vers les langages de données. On étend ensuite ces résultats à un modèle d'automate plus général qui englobe les cas particuliers des automates temporisés et les automates de données.

Dans les deux tableaux suivants, on présente les similitudes entre les théories des langages de données et des langages de mots réguliers. Le premier compare les théories générales et le second compare les équivalences connus sur les fragments des langages du «premier ordre».

Langages réguliers	**Langages de données**
Automates finis	Automates de données [BPT01]
Expressions rationnelles	*Théorème de Kleene*
MSO	Logique du second ordre [Bou02]
Monoïdes	Monoïdes [BPT01]

Langages réguliers	**Langages de données**
Automates sans compteur	? ? ?
Langages sans étoile	? ? ?
FO/LTL	? ? ?
Monoïdes apériodiques	? ? ?

Chapitre 1

Langages de données

Dans toute la suite, Σ est un alphabet fini et $\mathcal{D}$ est un ensemble (l'ensemble des données). Parmi les éléments de $\mathcal{D}$, on distingue une valeur particulière qu'on note $\perp$.
Les définitions et les résultats rappelés dans ce chapitre (sections 1.1,1.2,1.3,1.4,1.5 et 1.6.1) sont tous issus de [BPT01].

1.1 Premières définitions

Si Z est un ensemble, Z^* désigne les suites finies d'éléments de Z, Z^ω désigne les suites infinies d'éléments de Z et Z^∞, les suites finies ou infinies d'éléments de Z.
Un *mot de données* sur Σ et $\mathcal{D}$ est une suite finie $(a_1, d_1)...(a_p, d_p)$ de $(\Sigma \times \mathcal{D})^*$.
Un *langage de données* est un ensemble de mots de données.
Les automates qu'on va considérer seront munis d'une mémoire bornée : un ensemble fini de registres dans lesquels il sera possible d'écrire des données.
Si $k \geqslant 0$ est le nombre de registres, une *mise à jour des registres*, $\overline{up}$, est une partie de $\{1..k\}$. L'ensemble des mises à jour (i.e. l'ensemble des parties de $\{1..k\}$) sera noté $\mathcal{U}$.
Une mise à jour des registres induit une application up de $\mathcal{D}^k \times \mathcal{D}$ dans $\mathcal{D}^k$ qui à $((d_i)_{i=1..k}, d)$ associe $((d'_i)_{i=1..k})$ où $d'_i = d$ si $i \in \overline{up}$ et $d'_i = d_i$ sinon.
Dans la suite, on écrira $((d'_i)_{i=1..k}) = up((d_i)_{i=1..k}, d)$.

1.2 Reconnaissance par monoïde

Dans le cadre des langages réguliers, on dispose d'un mécanisme algébrique de reconnaissance des langages : si $L \subset \Sigma^*$ est un langage, L est régulier si il existe un monoïde fini M, une partie $F \subset M$ et un morphisme de monoïde $f : \Sigma^* \to M$ tels que $L = f^{-1}(F)$.
On propose ici un mécanisme de reconnaissance de langage de données par monoïde, similaire au mécanisme décrit ci-dessus, mais nécessairement modifié pour tenir compte des données.

Définition 1.2.1 *Un mécanisme à k registres sur un monoïde fini M est un triplet $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \phi]$ où :*

– *pour tout* $(m, a) \in M \times \Sigma$*,* $up_{m,a}$ *est une mise à jour des registres.*
– $\sim$ *est une relation d'équivalence d'index fini sur* $\mathcal{D}^k$*.*
– ϕ *est un morphisme de monoïde de* $(\Sigma \times \mathcal{D}^k / \sim)^*$ *dans* M*.*

Calcul à l'aide du mécanisme

On se donne $u = (a_1, d_1)...(a_n, d_n)$ un mot de données, un mécanisme à registres $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \phi]$ et une partie $F \subset M$ du monoïde. Le calcul du mécanisme sur le mot u se réalise algorithmiquement de la façon suivante, où $\overline{\theta}$ désigne la classe d´équivalence de θ modulo $\sim$:

% initialisation
$m := 1$
$\forall 1 \leqslant j \leqslant k, \theta[j] := \perp$
%Calcul
For $i := 1$ to n do
$\theta := up_{m,a}(\theta, d_i)$
$m := m.\phi(a_i, \overline{\theta})$
Endfor
% Résultat
Si $m \in F$ retourner oui
Sinon, retourner non

On dit que le mot u est *accepté* par le mécanisme si et seulement si le calcul renvoie «oui». La dernière valeur de m calculée par le mécanisme ρ sur le mot u sera notée $\rho(u)$.

Définition 1.2.2 *Soit* L *un langage de données sur* Σ *et* $\mathcal{D}$ *et soit* M *un monoïde fini. On dit que* M *reconnait* L *s'il existe un sous-ensemble* F *de* M *et un mécanisme à* k *registres* $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \phi]$ *tels que :*

$$L = \rho^{-1}(F).$$

Définition 1.2.3 *Un langage de données est dit* reconnaissable par monoïde *s'il existe un mécanisme à monoïde qui le reconnaît.*

Exemple 1 : Le langage de données $L = \{(a, d)(a, d') | d \neq \perp, d \neq d'\}$ sur l'alphabet $\{a\}$ et sur l'ensemble de données $\mathcal{D}$ est reconnu par le monoïde fini $M = \{1, y, y^2, 0\}$ où $y^3 = 0$ et $0.x = x.0 = 0$ pour tout $x \in M$. On définit un mécanisme à deux registres $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \phi]$:
– Les mises à jour sont : $up_{1,a} = \{1\}$ et $up_{z,a} = \{2\}$ si $z \neq 1$.
– $\sim$ est formée de deux classes d'équivalence, $\overline{\theta_{\neq}} = \{(d, d') | d \neq d'\}$ et $\overline{\theta_{=}} = \mathcal{D}^2 \backslash \overline{\theta_{\neq}}$.
– Le morphisme $\phi : (\{a\} \times \{\overline{\theta_{\neq}}, \overline{\theta_{=}}\})^* \longrightarrow M$ est défini par $\phi(a, \overline{\theta_{\neq}}) = y$ et $\phi(a, \overline{\theta_{=}}) = 0$.

Ainsi défini, le mécanisme accepte L en utilisant la partie $F = \{y^2\}$. Donnons à présent un exemple de calcul. On considère un mot de données $(a, d)(a, d')$ où $d \neq \perp$ et $d \neq d'$.

Dans le monoïde M	1_M	$\xrightarrow[d]{a}$	y	$\xrightarrow[d']{a}$	y^2
Valeurs des registres	$\begin{pmatrix} \perp \\ \perp \end{pmatrix}$		$\begin{pmatrix} d \\ \perp \end{pmatrix}$		$\begin{pmatrix} d \\ d' \end{pmatrix}$
Classe d'équivalence	$\overline{\theta_{=}}$		$\overline{\theta_{\neq}}$		$\overline{\theta_{\neq}}$

Exemple 2 : On montre facilement que le langage de données

$$\{(a, d_1)...(a, d_n)(a, d) | d \notin \{d_1, ..., d_n\}\}$$

sur l'alphabet $\{a\}$ et sur $\mathcal{D}$, supposé infini, n'est reconnu par aucun monoïde fini. (Intuitivement, il faudrait un nombre infini de registres.)

Si M est un monoïde fini et k un entier naturel, l'ensemble des langages de données sur Σ et $\mathcal{D}$ reconnus par M en utilisant k registres est noté $\mathcal{L}_{M,k}(\Sigma, \mathcal{D})$. On pose également : $\mathcal{L}_M = \bigcup_k \mathcal{L}_{M,k}$ et $\mathcal{L}_k = \bigcup_M \mathcal{L}_{M,k}$.

Propriétés de clôture

On s'intéresse à présent aux propriétés de clôture des langages reconnus par un mécanisme à monoïde :

Proposition 1.2.1 *L'ensemble $\mathcal{L}_{M,k}$ est stable par complémentation. Si $L_1 \in \mathcal{L}_{M_1,k_1}$ et $L_2 \in \mathcal{L}_{M_2,k_2}$, alors, $L_1 \cup L_2$ et $L_1 \cap L_2$ sont dans $\mathcal{L}_{M_1 \times M_2, k_1+k_2}$.*

Rôles relatifs des registres et du monoïde

La proposition suivante précise les rôles du monoïde et des registres dans le pouvoir de reconnaissance d'un mécanisme. Ainsi, si on s'autorise à utiliser n'importe quel monoïde fini, augmenter le nombre de registres accroit la puissance de reconnaissance alors que si on choisit un monoïde, on peut borner le nombre de registres "utiles".

Proposition 1.2.2
1. *La suite $(\mathcal{L}_k(\Sigma, \mathcal{D}))_k$ est strictement croissante.*
2. *Si M est un monoïde fini, la suite $(\mathcal{L}_{M,k}(\Sigma, \mathcal{D}))_k$ est stationnaire :*

$$\forall k \geqslant 2^{|\Sigma \times M|} - 1, \text{ on a } \mathcal{L}_{M, 2^{|\Sigma \times M|}-1} = \mathcal{L}_{M,k}.$$

Remarque :
Pour démontrer le premier point, on montre que le langage de données :

$$L_k = \{(a, d_1)...(a, d_n) | i \equiv j \bmod (k-1) \Longrightarrow d_i = d_j\}$$

sur l'alphabet $\{a\}$ et sur l'ensemble de données $\mathcal{D}$, supposé infini, n'est reconnu par aucun mécanisme utilisant moins de k registres.

1.3 Automates de données

Comme dans le cas des langages réguliers de mots, on dispose d'une notion d'automate dont la puissance est équivalente à celle des mécanismes à monoïde. On rappelle que $\mathcal{U}$ désigne l'ensemble des mises à jour des registres.

Définition 1.3.1 *Un* automate de données *sur Σ et $\mathcal{D}$ est un $6-uplet$ $\mathcal{A} = (Q, q_0, F, k, \sim, T)$ où :*
- *Q est un ensemble fini d'états,*
- *$q_0 \in Q$ est l'état initial,*
- *$F \subset Q$ est l'ensemble des états finaux,*

- *k est un entier positif,*
- *$\sim$ est une relation d'équivalence d'index fini sur $\mathcal{D}^k$, et*
- *$T \subset (Q \times \mathcal{D}^k/\sim \times \Sigma \times \mathcal{U} \times \mathcal{D}^k/\sim \times Q)$ est un ensemble fini de transitions.*

tel que la condition déterministe suivante est vérifiée :
pour tout $(q, g, a) \in Q \times \mathcal{D}^k/\sim \times\Sigma$, il existe une unique mise à jour up telle que pour toute transition $(q, g, a, up', g', q') \in T$, on ait $up = up'$ et si (q, g, a, up, g', q'_1) et (q, g, a, up, g', q'_2) sont dans T, alors $q'_1 = q'_2$.

Remarque :
Au lieu d'imposer un unique état initial, on aurait pu autoriser un ensemble d'états initiaux. Le pouvoir de reconnaissance du modèle resterait identique, comme le montre la proposition 1.2.1 et le théorème 1.3.1.

La condition déterministe de la définition revient à imposer la condition suivante : l'automate peut réaliser au plus un calcul sur un mot donné.

Un mot de données $(a_1, d_1)...(a_n, d_n)$ est accepté par l'automate de données $\mathcal{A}$ si il existe un chemin dans $\mathcal{A}$

$$q_0 \xrightarrow[d_1]{g_1,a_1,up_1,g'_1} q_1 \xrightarrow[d_2]{g_2,a_2,up_2,g'_2} q_2...q_{n-1} \xrightarrow[d_n]{g_n,a_n,up_n,g'_n} q_n$$

tel que la suite $(\theta_i)_{i=0..n}$ définie par

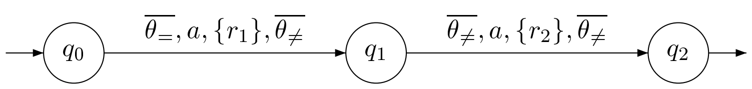
$$\theta_0 = \perp^k \text{ et } \theta_{i+1} = up_{i+1}(\theta_i, d_{i+1})$$

vérifie $\overline{\theta_{i-1}} = g_i$ pour $1 \leqslant i \leqslant n, \overline{\theta_i} = g'_i$ pour $1 \leqslant i \leqslant n$ et $q_n \in F$.
L'ensemble des mots de données acceptés par $\mathcal{A}$ est noté $L(\mathcal{A})$.

Exemple 3 :
Le langage de données de l'exemple 1,

$$L = \{(a, d)(a, d')|d \neq \perp, d \neq d'\}$$

est reconnu par l'automate de données suivant :



Théorème 1.3.1 *Soit L un langage de données sur Σ et $\mathcal{D}$. Alors L est reconnaissable par un automate de données si et seulement si L est reconnaissable par un mécanisme à monoïde.*

1.4 Comparaison avec les automates temporisés

Comme annoncé dans l'introduction, les langages temporisés reconnaissables forment une sous-classe des langages de données reconnaissables. Les automates de données non déterministes sont définis à la section 1.6.1.

Proposition 1.4.1 *Soit $\mathcal{A}$ un automate temporisé (resp. déterministe) à n horloges sur un domaine de temps $\mathcal{D}$. Il existe un automate de données (resp. déterministe) à $2n+2$ registres qui reconnaît le même langage.*

1.5 Décidabilité du vide

La classe des langages de données reconnaissables est indécidable : on peut simuler une machine à deux compteurs [Min67] par un automate de données. Voici une condition qui définit une classe d'automates de données pour lesquels le problème du vide est décidable :

$$\text{CONDITION } (\dagger) : \overline{\theta} \xrightarrow{up} \overline{\theta'} \text{ ssi } \exists v \in \overline{\theta}, \exists d \in \mathcal{D}, up(v,d) \in \overline{\theta'}$$

Si $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \phi]$ est un mécanisme à registres, on dit que ρ vérifie la condition ($\dagger$) si la condition ($\dagger$) est vérifiée pour toute classe de $\sim$ et pour chaque mise à jour $up_{m,a}$. On a alors le résultat de décidabilité suivant :

Théorème 1.5.1 *Soit L un langage de données reconnaissable sur l'alphabet Σ et sur l'ensemble de données $\mathcal{D}$. On suppose que L est reconnu par un monoïde fini M en utilisant un mécanisme à registres ρ vérifiant la condition ($\dagger$). Alors le problème du vide pour L est décidable et appartient à la classe de complexité $PSPACE$.*

1.6 Extensions du modèle

1.6.1 Modèles non déterministes

Jusqu'à présent, les différents mécanismes de reconnaissance de langages de données étaient déterministes, dans le sens où un unique calcul était possible sur un mot donné. On définit donc les automates de données non déterministes de la même manière que précédemment, mais en oubliant l'hypothèse déterministe de la définition. On peut par ailleurs autoriser plusieurs états initiaux, comme dans la remarque qui suit la définition 1.3.1. La condition d'acceptation est identique à celle définie pour les automates de données déterministes.
Un mécanisme à k-registres non déterministe est un triplet

$$\rho = [(Up_{m,a})_{m \in M, a \in \Sigma}, \sim, \phi]$$

où la seule différence avec la définition 1.2.1 est qu'à présent $Up_{m,a}$ désigne un ensemble de mises à jour. Un calcul du mécanisme sur le mot $u = (a_1, d_1)...(a_n, d_n)$

se réalise de la façon suivante :

% initialisation
$m := 1$
$\forall 1 \leqslant j \leqslant k, \theta[j] := \perp$
%Calcul
For $i := 1$ to n do
Choisir $up \in Up_{m,a_i}$
$\theta := up(\theta, d_i)$
$m := m.\phi(a_i, \overline{\theta})$
Endfor
% Résultat
Si $m \in F$ retourner oui
Sinon, retourner non

Le mot u est accepté par le mécanisme si et seulement si il existe un calcul qui renvoie «oui».
On dit qu'un monoïde fini M reconnaît de manière non déterministe un langage de données L si il existe un mécanisme à k registres qui reconnaît L, de la même manière qu'à la définition 1.2.2.

Proposition 1.6.1 *Soit L un langage de données.*

- *L est reconnu de manière non déterministe par un automate de données si et seulement si il est reconnu de manière non déterministe par un monoïde. Dans ce cas, on dit que L est* nd-reconnaissable*.*
- *Si L est reconnu par un automate de données non déterministe qui satisfait la condition (†), on peut décider si L est le langage vide.*
- *La classe des langages de données nd-reconnaissables contient strictement la classe des langages de données reconnaissables.*

La situation est donc différente de celle des langages de mots réguliers puisqu'à partir d'un automate fini non déterministe, on peut construire un automate fini déterministe qui reconnaît le même langage.

On prouve le dernier point de la proposition 1.6.1 en démontrant que le langage
$$L = \{(a, d_1)...(a, d_n) | \exists 1 \leqslant i < j < n, d_i = d_j \neq \perp\}$$
est nd-reconnaissable mais pas reconnaissable. L étant la concaténation des langages $\{(a, d_1)...(a, d_p) | d_i \in \mathcal{D}\}$ et $\{(a, d_0)...(a, d_n) | \exists 1 \leqslant j < n, d_j = d_0\}$ qui sont reconnaissables, on déduit la proposition suivante :

Proposition 1.6.2 *La classe des langages de données reconnaissables n'est pas stable par concaténation.*

La proposition suivante résume les propriétés de clôture de la classe des langages nd-reconnaissables :

Proposition 1.6.3 *La classe des langages de données nd-reconnaissable est stable pour les opérations d'union, d'intersection, de concaténation et d'itération finie. Elle n'est pas stable par complémentation.*

1.6.2 Le pouvoir des mises à jour

On s'intéresse à présent à la classe des langages reconnaissables lorsqu'on autorise des mises à jour plus générales. On commence par montrer qu'utiliser des mises à jour un peu moins restrictives ne modifie pas la classe des langages reconnaissables.

On va utiliser, pour commencer, des mises à jour de la forme suivante : si l'automate considéré a k registres, une mise à jour sera une fonction de $\{1..k\}$ dans $\{1..k\} \cup \{\perp\}$. Si up est une telle mise à jour, up induit une application de $\mathcal{D}^k \times \mathcal{D}$ dans $\mathcal{D}$, en notant $R' = up(R, d)$:

$$r'_i = \begin{cases} r_{up(i)} \text{ si } up(i) \in \{1..k\} \\ d \text{ sinon.} \end{cases}$$

On note $\mathcal{H}$ l'ensemble des fonctions de $\{1..k\}$ dans $\{1..k\} \cup \{\perp\}$.

Proposition 1.6.4 *Soit $\mathcal{A} = (Q, I, F, k, \sim, T)$ un automate de données (resp. déterministe) dont les mises à jour sont de la forme décrite ci-dessus. Alors, il existe un automate de données «classique» (resp. déterministe) qui reconnaît le même langage.*

Preuve :

On construit un automate $\mathcal{B}$:

$$\mathcal{B} = (Q', I', F', k, \sim', T')$$

où, si $\mathcal{F} = \{f : \{1..k\} \to \{1..k\}\}$

- $Q' = Q \times \mathcal{F}$
- $I' = I \times \{id\}$
- $F' = F \times \mathcal{F}$
- Afin de construire l'ensemble T' des transitions, on considère une transition τ de l'automate $\mathcal{A}$:

$$\begin{array}{ccc} q & \xrightarrow[d]{g,a,up,g'} & q' \\ R & & R' \end{array}$$

  où R et R' désignent les vecteurs des registres avant et après la transition. On se donne une fonction $f \in \mathcal{F}$. On va construire une transition τ' :

$$\begin{array}{ccc} (q, f) & \xrightarrow[d]{\widetilde{g},a,\widetilde{up},\widetilde{g'}} & (q', f') \\ \widetilde{R} & & \widetilde{R'} \end{array}$$

  de l'automate $\mathcal{B}$, $\widetilde{R}$ et $\widetilde{R'}$ désignant les vecteurs de registres avant et après le passage de la transition.
  On note, pour $g \in \mathcal{F}$, $R.g$ le vecteur de registre $(r_{g(i)})_{i=1..k}$.
  Nous allons contruire une mise à jour «classique» et une fonction $f' \in \mathcal{F}$, de sorte que si, au cours d'un calcul, on a $R = \widetilde{R}.f$ et qu'on lit la lettre (a, d) alors :

– la transition τ est franchissable dans $\mathcal{A}$ si et seulement si la transition τ' est franchissable dans $\mathcal{B}$,
– dans ce cas, après passage des transitions, on a $R' = \widetilde{R'}.f'$.

Pour cela, on pose $\tilde{g} = \{\tilde{R} | f.\tilde{R} \in g\}$, de sorte que :

$$\tilde{R} \in \tilde{g} \Longleftrightarrow R \in g$$

dès que $R = f.\tilde{R}$.

De même, on pose $\tilde{g'} = \{\tilde{R'} | f'.\tilde{R'} \in g'\}$.
Pour choisir la fonction f' et la mise à jour $\tilde{up}$, on va distinguer plusieurs cas :

- Si la fonction up ne prend pas la valeur $\perp$ (i.e. $up \in \mathcal{F}$) :
  on pose $\tilde{up} = \emptyset$ et $f' = f \circ up$.
  Ainsi, si $R = \tilde{R'}.f$, on a : $R' = R.up$, $\tilde{R'} = \tilde{R}$ et

$$R' = R.up = (\tilde{R}.f).up = \tilde{R}.(f \circ up) = \tilde{R'}.(f \circ up).$$

- Sinon, il existe un entier j tel que la fonction up ne prenne pas la valeur j i.e la valeur du registre r_j n'est ni recopiée dans un autre registre, ni conservée.
  – s'il existe l tel que f ne prenne pas la valeur l, on pose $\tilde{up} = \{l\}$ et on définit f' de la façon suivante :

$$\begin{array}{rccc} f' : & \{1..k\} & \longrightarrow & \{1..k\} \\ & i & \longmapsto & \begin{array}{cc} f \circ up(i) & \text{si } up(i) \neq \perp \\ l & \text{sinon.} \end{array} \end{array}$$

  Montrons que f' et $\tilde{up}$ ainsi définies conviennent ; si $i \in \{1..k\}$,
  si $up(i) = \perp$, $r'_i = d = \tilde{r'}_l = \tilde{r'}_{f'(i)}$,
  si $up(i) \neq \perp$, $r'_i = r_{up(i)} = \tilde{r}_{f \circ up(i)} = \tilde{r'}_{f'(i)}$, car $f \circ up(i) \neq l$.
  – si f est surjective (i.e. bijective), on pose $\tilde{up} = \{f(j)\}$ et

$$\begin{array}{rccc} f' : & \{1..k\} & \longrightarrow & \{1..k\} \\ & i & \longmapsto & \begin{array}{cc} f \circ up(i) & \text{si } up(i) \neq \perp \\ f(j) & \text{sinon.} \end{array} \end{array}$$

  En effet, si $i \in \{1..k\}$,
  si $up(i) = \perp$, $r'_i = d = \tilde{r'}_{f(j)} = \tilde{r'}_{f'(i)}$,
  si $up(i) \neq \perp$, $r'_i = r_{up(i)} = \tilde{r}_{f \circ up(i)} = \tilde{r'}_{f'(i)}$, car $f \circ up(i) \neq f(j)$. La transition ainsi construite convient.
  Remarque :
  Il n'y a pas, en général, unicité pour les choix de $\tilde{up}$ et de f'.

  On définit l'ensemble T' des transitions de la façon suivante : pour tout couple $(\tau, f) \in T \times \mathcal{F}$, on construit, de la même manière que ci-dessus, une transition τ'_f.
  On pose $T' = \bigcup_{(\tau,f) \in (T,\mathcal{F})} \{\tau'_f\}$.

– $\sim'$ est la relation d'équivalence engendrée par les gardes des transitions.
Montrons que $L(\mathcal{A}) = L(\mathcal{B})$.

Soit $u = (a_1, d_1)...(a_n, d_n) \in (\Sigma \times \mathcal{D})^*$, un mot de données.

Si $u \in L(\mathcal{A})$, il existe un chemin acceptant dans $\mathcal{A}$ pour le mot u :

$$q_0 \xrightarrow[d_1]{g_1,a_1,up_1,g_1'} q_1 ... q_{n-1} \xrightarrow[d_n]{g_n,a_n,up_n,g_n'} q_n$$

On considère le chemin suivant dans $\mathcal{B}$:

$$(q_0, Id) \xrightarrow[d_1]{t_{(q_0,q_1,Id)}} (q_2, f_2)...(q_{n-1}, f_{n-1}) \xrightarrow[d_n]{t_{(q_{n-1},q_n,f_{n-1})}} (q_n, f_n)$$

Par construction de T', ce chemin est acceptant pour le mot u et $u \in L(\mathcal{B})$.
Réciproquement, si $u \in \mathcal{B}$, il existe un chemin acceptant pour u dans $\mathcal{B}$:

$$(q_0, Id) \xrightarrow[d_1]{t_{(q_0,q_1,Id)}} (q_2, f_2)...(q_{n-1}, f_{n-1}) \xrightarrow[d_n]{t_{(q_{n-1},q_n,f_{n-1})}} (q_n, f_n)$$

et toujours par construction de T', le chemin :

$$q_0 \xrightarrow[d_1]{g_1,a_1,up_1,g_1'} q_1 ... q_{n-1} \xrightarrow[d_n]{g_n,a_n,up_n,g_n'} q_n$$

est un chemin acceptant pour u dans $\mathcal{A}$. Ainsi, $L(\mathcal{A}) = L(\mathcal{B})$ et $\mathcal{A}$ est déterministe si et seulement si $\mathcal{B}$ l'est, vue la méthode de construction des transitions de l'automate $\mathcal{B}$. $\square$

Le preuve montre également le résultat suivant :

Proposition 1.6.5 *Dans les définitions d'automate de données non déterministe et de mécanisme à monoïde non déterministe, on peut imposer la condition* $|up| \leqslant 1$ *sans modifier le pouvoir d'expression du modèle.*

On généralise à présent la proposition 1.6.4, en autorisant des mises à jour de la même forme que précedemment mais dépendant des valeurs des registres : une mise à jour up est à présent une application de $\mathcal{D}^k$ dans $\mathcal{H}$. On notera up_R pour $up(R)$.

Proposition 1.6.6 *Soit* $\mathcal{A} = (Q, I, F, k, \sim, T)$ *un automate de données (resp. non déterministe) dont les mises à jour sont de la forme décrite ci-dessus. Alors, il existe un automate «classique» (resp. non déterministe) qui reconnaît le même langage.*

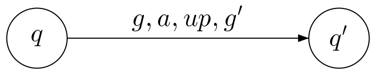
Preuve :
On va se ramener aux hypothèses de la proposition 1.6.4.
On construit un automate de données $\mathcal{B} = (Q, I, F, k, \sim', T')$.
Pour toute transition $\tau = (q, g, a, up, g', q') \in T$, on considère la relation d'équivalence $\approx_\tau$ sur $\mathcal{D}^k$ définie par :

$$R \approx_\tau R' \iff up_R = up_{R'}$$

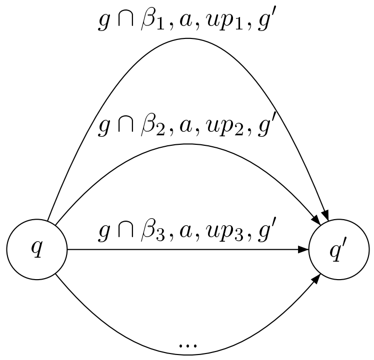
Le nombre de classes d'équivalence est fini car $|\mathcal{H}|$ est fini. Notons $\beta_1, ..., \beta_n$ ces classes d'équivalences. A chaque classe β_i, on associe la fonction $up_i \in \mathcal{H}$ telle que :

$$\forall R \in \beta_i, up_R = up_i$$

On pose alors $T_{tau} = \bigcup_{i=1..n}\{(q, g \cap \beta_i, a, up_i, g', q')\}$ et $T' = \bigcup_{\tau \in T} T_{tau}$.
Une transition dans l'automate $\mathcal{A}$:



devient un ensemble de transitions :



$\sim'$ est la relation d'équivalence engendrée par les gardes des transitions. Il est alors clair que $L(\mathcal{A}) = L(\mathcal{B})$ et que $\mathcal{B}$ est déterministe si et seulement si $\mathcal{A}$ l'est et la proposition 1.6.4 permet de conclure la preuve. $\square$

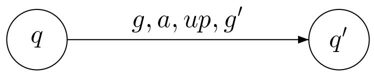
On généralise encore un peu le résultat précédent en considérant des mises à jour qui dépendent à la fois des valeurs courantes des registres et de la nouvelle donnée. A présent, une mise à jour up est une application de $\mathcal{D}^k \times \mathcal{D}$ dans $\mathcal{H}$. On notera $up_{(R,d)}$ pour $up(R, d)$.

Proposition 1.6.7 *Soit $\mathcal{A} = (Q, I, F, k, \sim, T)$ un automate de données (resp. non déterministe) dont les mises à jour sont de la forme décrite ci-dessus. Alors, il existe un automate «classique» (resp. non déterministe) qui reconnaît le même langage.*

Preuve :
On va, comme précédemment, se ramener aux hypothèses de la proposition 1.6.4. On construit un automate $\mathcal{B} = (Q', I', F', 2 * k + 2, \sim', T')$ où :
- $Q' = Q \times \{0, 1\}$,
- $I' = I \times \{0\}$,
- $F' = F \times \{0, 1\}$,

Les registres seront notés $r_1, .., r_{k+1}, r'_1, ..., r'_{k+1}$: on écrira alternativement sur les registres primés et non primés.

Considérons une transition $\tau \in T$ de l'automate $\mathcal{A}$:



On définit la relation d'équivalence $\approx_\tau$, sur $\mathcal{D}^{k+1}$:

$$(R,d) \approx_\tau (R',d') \Longleftrightarrow up_{(R,d)} = up_{(R',d')}$$

L'ensemble $\mathcal{H}$ étant fini, la relation $\approx_\tau$ est d'index fini. Notons $\beta_1, ..., \beta_n$ les classes d´équivalence. A chaque classe β_i, on associe un élément up_i de $\mathcal{H}$ de sorte que :

$$\forall (R,d) \in \beta_i, up_{(R,d)} = up_i.$$

On définit les mises à jour :

$$up_{i,0} : j \mapsto \begin{array}{lll} up_i(j) & \text{si} & j < k+1 \\ \perp & \text{si} & j = k+1 \\ k' & \text{si} & j = k' \end{array}$$

et

$$up_{i,1} : j \mapsto \begin{array}{lll} up_i(j')' & \text{si} & j < k+1 \\ \perp & \text{si} & j = (k+1)' \\ k & \text{si} & j = k \end{array}$$

et les parties de $\mathcal{D}^{(2k+2)}$:

$$g_{i,0} = \{r_1, .., r_{k+1}, r'_1, ..., r'_{k+1} \in \mathcal{D}^{2k+2} | (r'_1, .., r'_k, r_{k+1}) \in \beta_i\}$$

et

$$g_{i,1} = \{r_1, .., r_{k+1}, r'_1, ..., r'_{k+1} \in \mathcal{D}^{2k+2} | (r_1, .., r_k, r'_{k+1}) \in \beta_i\}.$$

On pose alors :

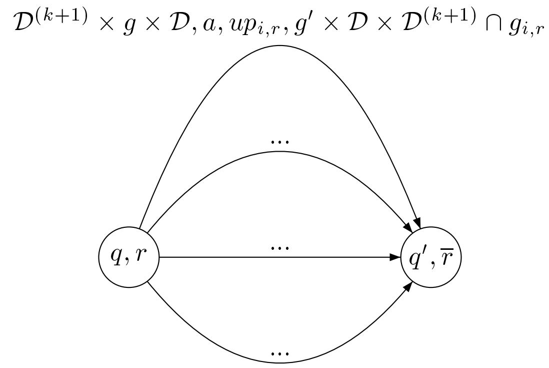
$$T_\tau = \bigcup_{i=1..n} \{((q,0), \mathcal{D}^{(k+1)} \times g \times \mathcal{D}, a, up_{i,0}, g' \times \mathcal{D} \times \mathcal{D}^{(k+1)} \cap g_{i,0}, (q',1))\}$$

$$\cup \bigcup_{i=1..n} \{((q,1), g \times \mathcal{D} \times \mathcal{D}^{k+1}, a, up_{i,1}, \mathcal{D}^{k+1} \times g' \times \mathcal{D} \cap g_{i,1}, (q',0))\}$$

et

$$T' = \bigcup_{\tau \in T} T_\tau.$$

La transition τ de l'automate $\mathcal{A}$ est transformée en un ensemble de transition dans l'automate $\mathcal{B}$:



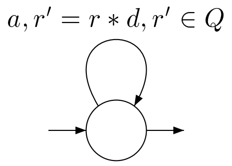
$\sim'$ est la relation d'équivalence engendrée par les gardes des transitions.

Alors, l'automate $\mathcal{B}$ reconnaît exactement $L(\mathcal{A})$ et la proposition 1.6.4 permet de conclure la preuve. □

Remarque :
Les mises à jour définies avant la proposition 1.6.7 sont les plus puissantes qu'on peut autoriser sans changer la puissance des automates. En effet, si on autorise des fonctions quelconques, on peut, par exemple, avec un seul registre, reconnaître le langage

$$L = \{(a_1, d_1)(a_2, d_2)...(a_n, d_n) \in \Sigma \times \mathbb{N} | d_1.d_2...d_n \text{ est sans facteur carré}\}$$

grâce à l'automate suivant :



où $Q = \{n \in \mathbb{N} | n \text{ est sans facteur carré}\}$.

Pourtant, aucun automate de données (déterministe ou non) ne reconnaît L :

Proposition 1.6.8 *Soit :*

$$L = \{(a_1, d_1)...(a_n, d_n) \in \Sigma \times \mathbb{N} | d_1.d_2....d_n \textit{ est sans facteur carré}\}$$

Aucun automate de données, déterministe ou non, ne reconnait L.

Preuve :
Par l'absurde, soit $\mathcal{A} = (Q, I, F, k, \sim, T)$ un automate de données qui reconnaît L. Soit $\mathcal{C} = \{\text{chemins de longueur } k{+}1 \text{ initiaux et finaux}\}$. Pour $c \in \mathcal{C}$, on pose :

$$E(c) = \{\omega = (a, d_1)(a, d_2)...(a, d_{k+1}) | \omega \text{ est accepté par } c\}$$

Pour un tel chemin, il existe un entier $n(c)$ tel que la donnée indicée par $n(C)$ n'est pas gardée en mémoire à la fin du chemin(i.e. si R_{k+1} est le dernier vecteur de registre, R_{k+1} ne contient jamais la valeur $d_{n(c)}$). Soit à présent $D'_n = \{n \text{ premiers nombres premiers}\} \subset \mathcal{D}$. Nous choisirons l'entier n plus tard. Alors, il existe un chemin c_0, tel que :

$$|E(c_0)| \geqslant \frac{A_{k+1}^{|D'_n|}}{|\mathcal{C}|}$$

On choisit n assez grand, de sorte que $\dfrac{A_{k+1}^{|D_n|}}{|\mathcal{C}|} > A_k^{|D_n|}$.
Pour tout $1 \leqslant i \leqslant k+1$, on définit une partition P_i de

$$U = \{((a, d_1)(a, d_2)...(a, d_{k+1}) | \forall 1 \leqslant i \leqslant k{+}1, d_i \in D_n \text{ et } \forall i, j, i \neq j \Rightarrow d_i \neq d_j\},$$

par :

$$P_i = \sqcup_j H_i^{(j)}$$

où $\omega, \omega' \in U$ sont dans la même classe $H_i^{(j)}$ si et seulement si leurs ièmes caractères diffèrent. Pour tout i, le nombre de classes est : $A_k^{|D_n|}$. Puisque $|E(c_0)|$ est supérieur au nombre de classes, il existe deux mots de données $(a, d_1)...(a, d_{n(c_0)})..(a, d_{k+1})$ et $(a, d_1)...(a, d'_{n(c_0)})..(a, d_{k+1})$ dans $E(c_0)$ tels que :

$$\begin{array}{cccccccccc}
(a, d_1) & & ... & & (a, d_{n(c_0)}) & & ... & & (a, d_{k+1}) & \\
 & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
 & \theta_1 & & \theta_{n(c_0)-1} & & \theta_{n(c_0)} & & \theta_k & & \theta_{k+1} \\
 & = & & = & & \sim & & \sim & & = \\
 & \theta'_1 & & \theta'_{n(c_0)-1} & & \theta'_{n(c_0)} & & \theta'_k & & \theta'_{k+1} \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
(a, d_1) & & ... & & (a, d'_{n(c_0)}) & & ... & & (a, d_k) &
\end{array}$$

Comme $\theta_{k+1} = \theta'_{k+1}$, le mot de données $(a, d_1)...(a, d_{n(c_0)})..(a, d_{k+1})\omega$ est reconnu par $\mathcal{A}$ si et seulement si $(a, d_1)...(a, d'_{n(c_0)})..(a, d_{k+1})\omega$ est reconnu par $\mathcal{A}$. Ce qui est absurde car $\mathcal{A}$ reconnaîtrait le mot

$$(a, d_1)...(a, d_{n(c_0)})..(a, d_{k+1})(a, d_{n(c_0)}).$$

Ainsi aucun automate de données n'accepte le langage L. $\square$

Remarque :
La preuve de la première partie de la proposition 1.2.2 est similaire à la preuve ci-dessus.

1.7 Simulation de réseaux de Petri

Nous avons vu que les automates de données permettent de simuler les automates temporisés. Nous nous intéressons à présent à un autre modèle de système temporisé : les réseaux de Petri. Nous commençons par les présenter et ensuite nous montrons que les automates de données peuvent simuler le comportement de ces réseaux.

1.7.1 Rappels sur les réseaux de Petri

Structure Commençons par rappeler la définition formelle des réseaux de Petri. Nous la détaillerons ensuite.

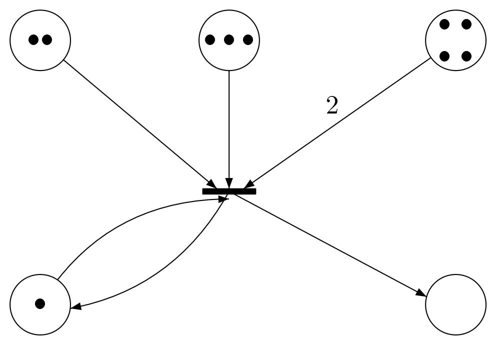
Définition 1.7.1 *Un* réseau de Petri R *est un triplet* $R = (P, T, W)$*, où :*
- $P = \{p_1, ..., p_m\}$ *est un ensemble fini de* places*,*
- $T = \{t_1, ..., t_n\}$ *est un ensemble fini de* transitions*,*
- $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ *est la fonction de* valuation*.*
- *Un* marquage M *est une fonction de* P *dans* $\mathbb{N}$*.*
- *Un* réseau de Petri marqué *est un couple* $Rm = (R, M_0)$*, où* R *est un réseau de Petri, et* M_0 *est un marquage, appelé* marquage initial*.*

On définit également la fonction Pre, restriction de la fonction W à $P \times T$ et la fonction $Post$, restriction de la fonction W à $T \times P$.

Intuitivement, les places et leur marquage correspondent respectivement aux différentes ressources et aux nombres de ressources disponibles, les transitions, aux actions et la fonction de transitions donne une condition pour qu'une action (une transition) puisse se réaliser ainsi que le résultat de l'action sur les places. Plus précisément, la fonction Pre donne les conditions pour qu'une action soit possible et la fonction $Post$ indique le résultat d'une action sur le marquage des places : $W(p,t) = k$ signifie que la transition t utilise k ressources du site p et $W(t,p) = k$ signifie que l'action t crée k ressources dans le site p. Nous reviendrons sur l'aspect comportemental un peu plus loin. Pour l'instant, nous allons expliquer comment on représente graphiquement les réseaux :

Représentation des réseaux de Petri On représente les places par des ronds et les transitions par des rectangles. Soit $(u,v) \in T \times P \cup P \times T$. Si $W(u,v) > 0$, on dessine un arc de u vers v, étiqueté par $W(u,v)$. Si $W(u,v) = 1$, on n'écrit pas l'étiquette. Le marquage initial est représenté par des jetons ($\bullet$) disposés dans les places.

Exemple :



Comportement d'un réseau de Petri Nous allons à présent décrire l'évolution du réseau lors du franchissement d'une transition :

Définition 1.7.2 *Soient $R = (P,T,W)$ un réseau de Petri, t une transition. La transition t est dite* franchissable *pour le marquage M si :*

$$\forall p \in P, M(p) \geqslant W(p,t)$$

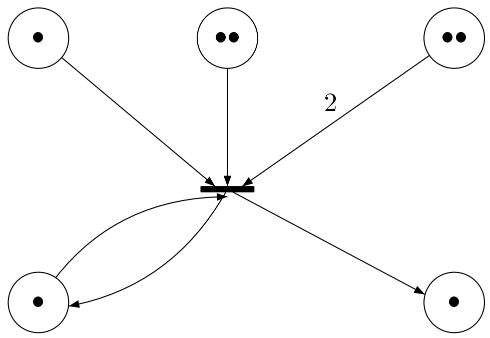
On note $M(t >$. Si t est franchissable pour M, on dit que le franchissement de t fait passer de M à M' *si :*

$$\forall p \in P, M'(p) = M(p) - W(p,t) + W(t,p)$$

On note $M(t > M'$.

Intuitivement, une transition est franchissable dès qu'il y a suffisamment de ressources disponibles dans les places concernées par cette transition. Lorsqu'on la franchit, on doit mettre à jour le marquage pour tenir compte des ressources utilisées et des ressources créées.

Exemple :
Quand on franchit l'unique transition de l'exemple précédent, le marquage du réseau devient :



On peut à présent définir le langage associé à un réseau de Petri :

Définition 1.7.3 *Soit R un réseau de Petri. Le langage associé à R, noté $L(R)$, est l'ensemble des suites finies de marquages $M_0, M_1, ..., M_k$ telles que :*

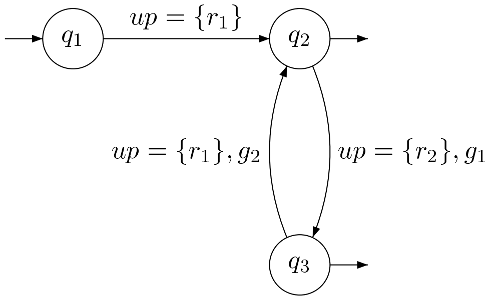
$$\forall 0 \leqslant i \leqslant (k-1), \exists t \in T, \text{ telle que } M_i(t > M_{i+1}.$$

1.7.2 Simulation d'un réseau de Petri par un automate de données

On se propose de simuler un réseau de Petri R par un automate de données $\mathcal{A}$ sur l'alphabet $\{a\}$ et sur l'ensemble de données $\mathcal{D} = \mathbb{N}^m$, si m est le nombre de places du réseau, dans le sens suivant :
le mot de données $u = (a, d_0)(a, d_2)...(a, d_k)$ est accepté par $\mathcal{A}$ si et seulement si la suite de marquage $d_0, ..., d_k \in L(R)$.
On a alors la proposition suivante :

Proposition 1.7.1 *Si R est un réseau de Petri, on peut construire un automate de données $\mathcal{A}$ qui simule R.*

Exemple :
L'automate de données ci-dessous, sur l'alphabet $\{a\}$ et l'ensemble de données $\mathcal{D} = \mathbb{N}^5$, reconnaît les comportements du réseau de l'exemple précédent, i.e. le mot de données $(a, d_1)...(a, d_k)$ est accepté par l'automate si et seulement si d_0 est le marquage initial du réseau et si, $\forall 0 \leqslant i < (k-1)$, on peut passer du marquage d_i à d_{i+1} en franchissant une transition. Il est muni de deux registres r_1, r_2. Intuitivement, la première transition correspond à l'écriture du marquage initial sur le premier registre. On écrit ensuite alternativement sur le premier et sur le second registre afin de tester si la condition de franchissement d'une transition est vérifiée.



où :
$g_1 = (r_{1,1} = r_{2,1}) \wedge (r_{1,2} - r_{2,2} = 1) \wedge (r_{1,3} - r_{2,3} = 1) \wedge (r_{1,4} - r_{2,4} = 2)$
$\wedge (r_{2,5} - r_{1,5} = 1)$ et
$g_2 = (r_{1,1} = r_{2,1}) \wedge (r_{2,2} - r_{1,2} = 1) \wedge (r_{2,3} - r_{1,3} = 1) \wedge (r_{2,4} - r_{1,4} = 2)$
$\wedge (r_{2,5} - r_{1,5} = 1)$.

Démonstration de la proposition 1.7.1 :

On construit un automate de données à trois états :

$$\mathcal{A} = (Q, I, F, 2, \sim, T)$$

où :
- $Q = \{q_1, q_2, q_3\}$.
- $I = \{q_0\}$.
- $F = \{q_1, q_2\}$.
- Les registres sont notés r_1 et r_2.
- On pose $\tau_{init} = (q_1, up = \{r_1\}, q_2)$ et pour toute transition t_i du réseau, on pose

$$g_i = \bigwedge_{k=1}^{m} ((r_{1,k} - r_{2,k}) = W(p_k, t_i)) \wedge \bigwedge_{k=1}^{m} ((r_{2,k} - r_{1,k}) = W(t_i, p_k)),$$

$$g = \bigvee_i g_i,$$

$$g'_i = \bigwedge_{k=1}^{m} ((r_{2,k} - r_{1,k}) = W(p_k, t_i)) \wedge \bigwedge_{k=1}^{m} ((r_{1,k} - r_{2,k}) = W(t_i, p_k)), \text{ et}$$

$$g' = \bigvee_i g'_i.$$

  On pose $\tau = (q_2, up = \{r_2\}, g, q_3)$, $\tau' = (q_3, up = \{r_1\}, , g', q_2)$ et, finalement, $T = \{\tau_{init}, \tau, \tau'\}$.
- $\sim$ est la relation d'équivalence engendrée par g et g'.

Il est alors clair que $\mathcal{A}$ reconnaît exactement le langage $L(R)$. $\square$

Conclusion

Après avoir rappeler les définitions et les principaux résultats concernant les langages de données, nous avons étudié plusieurs extensions du modèle en autorisant des mises à jour plus générales. Pour résumer, tant que ces extensions étaient «raisonnables», le modèle conservait le même pouvoir d'expression, ce qui témoigne de la robustesse du modèle.
Finalement, nous avons montré que les automates de données permettent de simuler les comportements des réseaux de Petri, résultat qu'on peut rapprocher de la proposition 1.4 qui assure qu'on peut simuler un automate temporisé par un automate de données.

Chapitre 2

Logique et langages de données

Les formalismes logiques sur les langages permettent de spécifier simplement (c'est à dire dans une langue proche des langues naturelles) des propriétés. Par exemple, la formule $\forall x.\big(R_{panne}x \Rightarrow (\exists y.(x < y) \wedge R_{alarme}y)\big)$, sur un alphabet contenant, au moins, les lettres *panne* et *alarme*, désigne l'ensemble des mots qui vérifie la propriété : si le mot contient la lettre *panne*, il contient plus loin la lettre *alarme*.

A des fins de vérification, il est naturel de se demander si le système étudié possède telle ou telle propriété. La technique du model-checking consiste à se donner un modèle du système, par exemple sous la forme d'un automate $\mathcal{A}$, et une formule logique ϕ qui exprime une propriété et à répondre à la question : l'automate $\mathcal{A}$ vérifie-t-il la propriété ϕ ? On peut se demander, en reprenant l'exemple précédent, si, dans le modèle de notre système, toute panne est suivi d'une alarme.

Nous commençons donc ce chapitre en décrivant un langage logique, proposé par P.Bouyer dans [Bou02], qui caractérise les langages de données reconnaissables. Dans la seconde partie du chapitre, nous exhibons un fragment de la logique qui correspond aux langages de données reconnus par des mécanismes non déterministes à monoïdes apériodiques, comme dans le cas des langages réguliers où les langages reconnus par un monoïde apériodique sont exactement les langages définissables par une formule logique du premier ordre.

Pour arriver à ce résultat, on commence par caractériser logiquement les langages de données reconnus par un mécanisme à monoïde non déterministe dont les mises à jour sont de la forme suivante, si le mécanisme, sur le monoïde M, est muni de k registres :

$$Up_{m,a} = \mathcal{P}([1..k]), \forall m \in M, \forall a \in \Sigma.$$

Dans ce chapitre, comme dans le chapitre suivant, le terme reconnaissable sera utilisé à la place de nd-reconnaissable : on parlera de mécanismes à monoïde en sous-entendant qu'il s'agit de mécanismes non déterministe.

2.1 Sur les traces de MSO...

2.1.1 Une première tentative

On se propose de décrire un langage logique dont le pouvoir d'expression est identique à celui des automates de données. On considère, pour commencer, le langage logique $\mathcal{L}_c$, constitué de toutes les formules closes $\phi ::= \exists X_1 ... \exists X_k.\psi$, où les $(X_i)_{1 \leqslant i \leqslant k}$ sont des variables du second ordre et où ψ est une formule de la grammaire suivante :

$$\psi ::= x < y | x = y | \exists x.\psi | \forall x.\psi | R_a x | Xx | g(x_1, ..., x_l) | \neg\psi | \psi \wedge \psi | \psi \vee \psi$$

où g est un sous-ensemble de $\mathcal{D}^k$, $a \in \Sigma$ et $x, y, x_1, ..., x_l$ sont des variables du premier ordre. Ce langage logique est une logique du second ordre à laquelle on a ajouté des prédicats de données $g(x_1, ..., x_l)$. Intuitivement, ces prédicats nous permettent de savoir si le vecteur de données représenté par les variables $x_1, ..., x_l$ vérifie la contrainte g.
Ces formules s'interprètent sur des mots de données et des assignations pour les variables libres de la formule. Dans la suite, μ (resp. ν) est une assignation pour les variables du premier ordre (resp. du second ordre). On suppose que $u = (a_1, d_1)...(a_n, d_n)$ est un mot de données, que μ et ν sont des assignations pour les variables libres qui apparaissent dans la formule qu'on considére. La relation de satisfaction est alors définie inductivement de la façon suivante :

$$\begin{array}{rcl}
(u; \mu.\nu) \vDash x < y & \Longleftrightarrow & \mu(x) < \mu(y) \\
(u; \mu.\nu) \vDash x = y & \Longleftrightarrow & \mu(x) = \mu(y) \\
(u; \mu.\nu) \vDash \exists x.\psi & \Longleftrightarrow & \text{il existe } 1 \leqslant i \leqslant n, (u; \mu[x/i].\nu) \vDash \psi \\
(u; \mu.\nu) \vDash \forall x.\psi & \Longleftrightarrow & \text{pour tout } 1 \leqslant i \leqslant n, (u; \mu[x/i].\nu) \vDash \psi \\
(u; \mu.\nu) \vDash R_a x & \Longleftrightarrow & a_{\mu(x)} = a \\
(u; \mu.\nu) \vDash Xx & \Longleftrightarrow & \mu(x) \in \nu(X) \\
(u; \mu.\nu) \vDash g(x_1, ..., x_l) & \Longleftrightarrow & (d; \mu(x_1), ..., d_{\mu(x_l)}) \in g \\
(u; \mu.\nu) \vDash \neg\psi & \Longleftrightarrow & (u; \mu.\nu) \nvDash \psi \\
(u; \mu.\nu) \vDash \psi_1 \wedge \psi_2 & \Longleftrightarrow & (u; \mu.\nu) \vDash \psi_1 \text{ et } (u; \mu.\nu) \vDash \psi_2 \\
(u; \mu.\nu) \vDash \psi_1 \vee \psi_2 & \Longleftrightarrow & (u; \mu.\nu) \vDash \psi_1 \text{ ou } (u; \mu.\nu) \vDash \psi_2 \\
(u; \mu.\nu) \vDash \exists X.\psi & \Longleftrightarrow & \text{il existe } \alpha \subset \{1, .., n\} \text{ tel que } (u; \mu.\nu[X/\alpha]) \vDash \psi
\end{array}$$

$\mu[x/i]$ signifie qu'on étend l'assignation μ à la variable du premier ordre x et qu'on a alors $\mu(x) = i$. On définit $\nu[X/\alpha]$ de manière similaire. Si ϕ est une formule close, i.e. sans variable libre, la relation de satisfaction est définie pour des assignations vides. Dans ce cas, on écrit $u \vDash \phi$ pour $(u, \emptyset.\emptyset) \vDash \phi$. Un langage de données L est l'ensemble des *modèles* d'une formule close ϕ si :

$$L = \{u \in (\Sigma \times D)^* | u \vDash \phi\}.$$

On note, dans ce cas, $L = L(\phi)$.

Exemple 1 : Le langage de données

$$\{(a_1, d_1)..(a_n, d_n) | n \in \mathbb{N} \text{ et } \exists 1 \leqslant i < j \leqslant n, (a_i, d_i) = (a_j, d_j)\}$$

est l'ensemble des modèles de la formule $\exists x.\exists y.(x < y \wedge \bigvee_{a \in \Sigma}(R_a x \wedge R_a y) \wedge g(x, y))$ où $g = \{(d, d') | d = d'\}$.

Exemple 2 : Le complémentaire du langage précédent, le langage

$$\{(a_1, d_1)...(a_n, d_n)|n \in \mathbb{N} \text{ et } \forall 1 \leqslant i < j \leqslant n, (a_i, d_i) = (a_j, d_j)\}$$

peut être décrit par la formule $\forall x.\forall y.(x < y \Rightarrow g_{\neq}(x, y) \vee \bigwedge_{a\in\Sigma}(\neg R_a x \vee \neg R_a y))$ où $g_{\neq} = \{(d, d')|d \neq d'\}$.

Néanmoins, ce langage n'est reconnu par aucun automate de données.

La logique $\mathcal{L}_c$ ne capture donc pas la puissance des automates de données. C'est à un fragment de cette logique qu'on va s'intéresser à présent.

2.1.2 Caractérisation logique des langages de données reconnaissables

On définit le langage logique $\mathcal{L}$ comme l'ensemble des formules closes de la forme $\exists X_1...\exists X_k.\psi$ où les $(X_i)_{1\leqslant i\leqslant k}$ sont des variables du second ordre et ψ est une formule de la grammaire $\mathcal{G}$, définie ci-dessous :

$$\mathcal{G} : \psi ::= x < y|x = y|\exists x.\psi|\forall x.\psi|R_a x|Xx|g(X_1, .., X_l, x)|\neg\psi|\psi \wedge \psi|\psi \vee \psi$$

où g est un sous-ensemble de $\mathcal{D}^k$ et $a \in \Sigma$. La seule différence avec la grammaire $\mathcal{L}_c$ est la forme des prédicats de données $g(X_1, ..., X_l, x)$ dont voici la sémantique. Si $u = (a_1, d_1)...(a_n, d_n)$ est un mot de données et si μ et ν sont des assignations pour les variables du premier et du second ordre, on a :

$$(u, \mu.\nu) \vDash g(X_1, ..., X_l, x) \Longleftrightarrow (d^{(\mu,\nu)}_{max(X_1\leqslant x)}, ..., d^{\mu,\nu)}_{max(X_l\leqslant x)}) \in g$$

où pour tout $1 \leqslant i \leqslant l$,

$$d^{(\mu,\nu)}_{max(X_i\leqslant x)} = \begin{cases} d_\lambda \text{ si } \lambda \in \nu(X_i) \text{ et } \forall j, (j \in \nu(X_i) \text{ et } j \leqslant \mu(x) \text{ implique } j \leqslant \lambda) \\ \perp \text{ et } \forall j, j \in \mu(X_i) \text{ implique } \mu(x) < j \end{cases}$$

Intuitivement, les variables du second ordre X_i représentent l'ensemble des dates où le registre r_i est mis à jour. $max(X_i \leqslant x)$ représente donc la dernière date où a été mis à jour le registre r_i. En suivant les mises à jour définies par les X_i, $g(X_1, ..., X_l, x)$ teste si à la date x les registres satisfont la contrainte g.

Lemme 2.1.1 *Le langage logique $\mathcal{L}$ n'est pas plus expressif que le langage $\mathcal{L}_c$,i.e. pour toute formule ϕ dans $\mathcal{L}$, il existe une formule ψ dans $\mathcal{L}_c$ logiquement équivalente à ϕ.*

Théorème 2.1.1 (Caractérisation logique) *Un langage de données est reconnaissable par un automate de données si et seulement si il constitue l'ensemble des modèles d'une formule close de $\mathcal{L}$.*

Nous ne rapportons pas la preuve ici, mais nous aurons besoin, pour la suite, d'un résultat de réduction sur les formules.

2.1.3 Une réduction des formules de $\mathcal{L}$

Une *formule atomique* est une formule de la forme $x < y, x = y, R_a x, Xx$ ou $g(Y_1, ..., Y_l, x)$. Une formule atomique du type $g(Y_1, ..., Y_l, x)$ est un *prédicat atomique de données.* Toute formule close de $\mathcal{L}$ peut être mise sous forme prenex :

$$\phi = \exists X_1 ... \exists X_k . Q_1 x_1 ... Q_p x_p \left(\bigvee_i \bigwedge_j p_{i,j} \right)$$

où pour tout i, $Q_i \in \{\exists, \forall\}$ et pour tout $i, j, p_{i,j}$ est une formule atomique ou sa négation. Soit ϕ une formule sous forme prenex. Pour toute variable du second ordre apparaissant dans ϕ, il existe un entier $1 \leqslant i_Y \leqslant k$ tel que $Y = X_{i_Y}$. On définit une relation d'équivalence $\sim_\phi$ sur $\mathcal{D}^k$ de la façon suivante :

$$(\theta_i)_{i=1..k} \sim_\phi (\theta'_i)_{i=1..k} \Longleftrightarrow \text{ pour tout prédicat atomique de données}$$

$$g(Y_1, .., Y_l, x) \text{ de } \phi,\ ((\theta_i)_{i=i_{Y_1}..i_{Y_l}} \in g \Longleftrightarrow (\theta'_i)_{i=i_{Y_1}..i_{Y_l}} \in g)$$

Cette relation d'équivalence est d'index fini car une formule ne contient qu'un nombre fini de prédicats atomiques de données.

Si g est une classe d'équivalence pour $\sim_\phi$, un prédicat atomique de données de la forme $g(X_1, .., X_k, x)$ est dit *$\phi-$primitif.* Il est immédiat de vérifier qu'un prédicat atomique de données peut être transformé en une disjonction de prédicats atomiques de données $\phi-$primitif.

Lemme 2.1.2 *Toute formule close ϕ de $\mathcal{L}$ peut être transformée en une formule équivalente de la forme suivante :*

$$\exists X_1 ... \exists X_k . Q_1 x_1 ... Q_p x_p \left(\bigvee_i \bigwedge_j p_{i,j} \right)$$

où les $p_{i,j}$ sont une formule atomique ou sa négation, ou un prédicat de données $\phi-$primitif.

2.2 Vers FO=apériodique...

Dans le cadre des langages réguliers, on dispose d'une caractérisation logique des langages reconnus par un monoïde fini apériodique (i.e. pour tout élément x du monoïde considéré, $\exists n \in \mathbb{N}, x^n = x^{n+1}$) :

Théorème 2.2.1 (Kamp, Schützenberger) *Un langage $L \subset \Sigma^*$ est reconnu par un monoïde fini apériodique si et seulement si il est définissable par une formule logique du premier ordre.*

Pour une description des logiques et une preuve du théorème, voir [Str94].

Dans le reste de ce chapitre, on démontre un résultat similaire en caractérisant logiquement les langages de données reconnus par un monoïde apériodique.

2.2.1 Pour commencer...

Théorème 2.2.2 *Soit L un langage de données, les deux énoncés suivants sont équivalents :*

i. L est définissable dans $\mathcal{L}$ par une formule ne contenant pas de formule atomique du type Xx.

ii. L est reconnu par un mécanisme à monoïde apériodique avec $Up_{m,a} = \mathcal{P}(1..k)$.

Preuve :
Soit L un langage de données.
Supposons qu'il existe une formule ϕ de $\mathcal{L}$ définissant L de la forme :

$$\exists X_1 ... \exists X_k . Q_1 x_1 ... Q_p x_p \bigvee_i \bigwedge_j p_{i,j}$$

où $p_{i,j}$ est une formule atomique, sa négation ou un prédicat de données ϕ-primitif. On suppose que ϕ ne contient pas de terme de la forme Xx.
Soit $\psi = \bigvee_i \bigwedge_j p_{i,j}$. On transforme ψ en $\bar{\psi}$ suivant les règles suivantes :

$$\begin{array}{cc} \overline{x<y} = x<y & \overline{g(X_1..X_k,x)} = R_g x \\ \overline{x=y} = x=y & \overline{\psi_1 \wedge \psi_2} = \overline{\psi_1} \wedge \overline{\psi_2} \\ \overline{R_a x} = R_a x & \overline{\neg\psi} = \neg\overline{\psi} \end{array}$$

Ces formules s'interprètent sur des mots de $(\Sigma \times \mathcal{D}^k/\sim)^*$ en utilisant des assignations pour les variables libres. μ et ν sont des des assignations pour les variables du premier ordre et pour les variables du second ordre, respectivement.

$$((a_1,g_1)...(a_n,g_n);\mu.\nu) \vDash R_g x \iff g_{\mu(X)} = g$$

$$((a_1,g_1)...(a_n,g_n);\mu.\nu) \vDash R_a x \iff a_{\mu(X)} = a$$

Soit $u = (a_1,d_1)...(a_n,d_n) \in (\Sigma \times D)^*$ un mot de données et soit ν une assignation pour les variables du second ordre $X_1,..,X_k$. On associe à u les mots u_ν et u'_ν définis de la façon suivante :

$$u'_\nu = (a_1,d_1,\theta_1,\chi_1)..(a_n,d_n,\theta_n,\chi_n) \in (\Sigma \times D \times D^k \times \mathcal{P}(\{1..k\})^*$$

avec $\theta_0 = \perp^k, \theta_{i+1}[j] = \begin{cases} \theta_i[j] & \text{si } i \notin \nu(X_j) \\ d_i & \text{si } i \in \nu(X_j) \end{cases}$ et $\chi_i = \{j | i \in \nu(X_j)\}$,

$$u_\nu = (a_1,\overline{\theta_1})...(a_n,\overline{\theta_n}) \in (\Sigma \times D^k/\sim)$$

Lemme 2.2.1 $(u;\nu) \vDash Q_1 x_1 ... Q_p x_p \psi \iff (u_\nu;\nu) \vDash Q_1 x_1 .. Q_p x_p \overline{\psi}$

Preuve : Soit μ une valuation pour les variables du premier ordre. Nous allons démontrer que pour toute formule atomique ρ, $(u,\mu.\nu) \vDash \rho$ si et seulement si $(u_\nu,\mu.\nu) \vDash \overline{\rho}$. On en déduira l'équivalence annoncée puisque ψ est une combinaison booléenne de formules atomiques.

- si $\rho = R_a x$, alors $\overline{\rho} = R_a x$ et

$$(u;\mu.\nu) \vDash R_a x \iff a_{\mu(x)} = a \iff (u_\nu;\mu.\nu) \vDash R_a x.$$

- si $\rho = g(X_1, ..., X_k, x)$, alors $\overline{\rho} = R_g x$ et

$$(u; \mu.\nu) \models g(X_1, ..., X_k, x) \iff (d^{(\mu,\nu)}_{max(X_1 \leqslant x)}, ..., d^{(\mu,\nu)}_{max(X_k \leqslant x)}) \in g.$$

Mais $\theta_{\mu(x)} = (d^{(\mu,\nu)}_{max(X_1 \leqslant x)}, ..., d^{(\mu,\nu)}_{max(X_k \leqslant x)})$. On a donc :

$$(u_\nu; \mu.\nu) \models R_g x \iff \overline{\theta_{\mu(x)}} = g.$$

□

Puisque $Q_1x_1..Q_px_p\overline{\psi}$ est une formule logique du premier ordre, il existe un monoïde apériodique M qui reconnait les modèles de $Q_1x_1..Q_px_p\overline{\psi}$, en utilisant le théorème 2.2.1. Notons f le morphisme de monoïde de $(\Sigma \times \mathcal{D}^k/\sim)^*$ dans M et $F \subset M$ les éléments acceptants de M.

Lemme 2.2.2 *Le langage de données L est reconnu par le monoïde M et le mécanisme à k registres suivant :*

$$\rho = (Up_{m,a}, \sim, f)$$

où $Up_{m,a} = \mathcal{P}(\{1..k\})$, en utilisant $F \subset M$ comme partie acceptante.

Preuve :
Soit $u = (a_1, d_1)...(a_n, d_n) \in (\Sigma \times \mathcal{D})^*$.
Si $u \in L$, il existe une assignation ν des variables du second ordre telle que $(u; \nu) \models Q_1x_1...Q_px_p\psi$. On sait que :

$$(u_\nu) = (a_1, \overline{\theta_1})...(a_n, \overline{\theta_n}) \models Q_1x_1...Q_px_p\overline{\psi}.$$

On définit une suite de mises à jour de la façon suivante :

$$\forall 1 \leqslant i \leqslant n, up_i = \{j | i \in \nu(X_j)\}$$

On a alors : $f(a_1, \overline{\theta_1})...f(a_n, \overline{\theta_n}) \in F$. Donc u est reconnu par le mécanisme.

Réciproquement, si u est reconnu par le mécanisme, il existe une suite de mises à jour $up_i, 1 \leqslant i \leqslant n$ tel que la suite :

$$\theta_0 = \perp^k, \theta_{i+1} = up_i(\theta_i, d_i), i \leqslant n$$

vérifie :

$$f(a_1, \overline{\theta_1})...f(a_n, \overline{\theta_n}) \in F.$$

Soit ν la valuation des variables du second ordre définie par :

$$\nu(X_i) = \{j | i \in up_j\}$$

On a : $u_\nu = (a_1, \overline{\theta_1})...(a_n, \overline{\theta_n})$ et comme $u_\nu \models Q_1x_1...Q_px_p\overline{\psi}$,
$(u; \nu) \models Q_1x_1...Q_px_p\psi$ et $u \models \phi$. □

Ainsi, L est reconnu par un mécanisme à monoïde apériodique dont les mises à jour sont de la forme $Up_{m,a} = \mathcal{P}(1..k)$.

Réciproquement, supposons que L soit reconnu par un mécanisme

$$\rho = (Up_{m,a}, k, f)$$

sur le monoïde apériodique M, dont les mises à jour sont de la forme

$$Up_{m,a} = \mathcal{P}(1..k),$$

en utilisant la partie acceptante $F \subset M$.
Le morphisme f avec la partie F reconnaît un langage $\tilde{L} \subset (\Sigma \times \mathcal{D}^k / \sim)^*$ qui est définissable par une formule logique du premier ordre sous forme prenex $Q_1x_1..Q_px_p\overline{\psi}$. On transforme $\overline{\psi}$ en inversant les règles du paragraphe précédent pour obtenir ψ.

Lemme 2.2.3 *La formule $\exists X_1..\exists X_k Q_1x_1...Q_px_p\psi$ a comme ensemble de modèles les mots de L.*

Preuve :
Soit $u = (a_1, d_1)...(a_n, d_n) \in (\Sigma \times \mathcal{D})^*$.
Si $u \vDash \exists X_1...\exists X_k Q_1x_1...Q_px_p\psi$, il existe une valuation ν de $X_1, ..., X_k$ telle que $(u; \nu) \vDash Q_1x_1...Q_px_p\psi$. On a alors que :

$$u_\nu = (a_1, \overline{\theta_1})...(a_n, \overline{\theta_n}) \vDash Q_1x_1..Q_px_p\overline{\psi}.$$

On définit une suite de mises à jour :

$$\forall 1 \leqslant i \leqslant n, up_i = \{j | i \in \nu(X_j)\}$$

Le mécanisme, en utilisant la suite up_i, reconnaît le mot u.
Réciproquement, si le mécanisme reconnaît le mot u, il existe une suite de mises à jour $up_i, 1 \leqslant i \leqslant n$, telle que la suite :

$$\theta_0 = \perp^k, \theta_{i+1} = up_i(\theta_i, d_i)$$

vérifie $f(a_1, \overline{\theta_1})...f(a_n, \overline{\theta_n}) \in F$.

Si ν est la valuation définie par : $\mu(X_i) = \{j | i \in up_j\}$,
$u_\nu = (a_1, \overline{\theta_1})...(a_n, \overline{\theta_n})$ et $u_\nu \vDash Q_1x_1..Q_px_p\overline{\psi}$.
On en déduit que : $u \vDash \exists X_1..\exists X_k Q_1x_1...Q_px_p\psi$. □

Remarque :
Il existe des langages de données reconnus par des mécanismes à monoïdes apériodiques mais qui ne sont pas reconnus par des mécanismes à monoïdes apériodiques où $Up_{m,a} = \mathcal{P}(1..k)$. Ainsi, il est facile de montrer que le langage $L = \{u = (a_1, d_1)...(a_n, d_n) | \exists i < j, a_i = a_j = a \text{ et } d_i = d_j\}$ sur l'alphabet $\Sigma = \{a, b\}$ et sur un ensemble de données $\mathcal{D}$ infini n'est reconnu par aucun mécanisme où $Up_{m,a} = \mathcal{P}(1..k)$.
Il existe également des langages de données reconnaissables qui ne sont reconnus par aucun mécanisme à monoïde apériodique. Ainsi, le langage $L = \{u = (a, d_1)...(a, d_n) \in (\{a\} \times \mathcal{D})^* | n \text{ est pair}\}$ n'est reconnu par aucun monoïde apériodique.

2.2.2 Caractérisation logique des langages reconnus par un monoïde apériodique

On commence par montrer un résultat général sur les formules logiques ne contenant pas de terme de la forme Xx.

Proposition 2.2.1 *Soit $\phi \in \mathcal{G}$ une formule logique ne contenant pas de terme de la forme Xx. On se donne une variable du premier ordre x qui n'apprait pas dans ϕ. Il existe une formule $\phi_x \in \mathcal{G}$ telle que si $(a_1, d_1)...(a_n, d_n)$ est un mot de données et $i \in [1, n]$, on a équivalence entre :*
i. $((a_1, d_1)...(a_i, d_i); \mu.\mu) \vDash \phi$
ii. $((a_1, d_1)...(a_n, d_n); \mu[x/i].\nu) \vDash \phi_x$

Preuve :
Pour définir ϕ_x, on va procéder par induction sur la structure des formules :
i. si $\phi = (y < z)$, on pose $\phi_x = (y < z) \wedge (z \leqslant x)$,
ii. si $\phi = (y = z)$, on pose $\phi_x = (y = z) \wedge (z \leqslant x)$,
iii. si $\phi = \exists y.\psi$, on pose $\phi_x = \exists y.(\psi_x \wedge (y \leqslant x))$,
iv. si $\phi = R_a y$, on pose $\phi_x = R_a y \wedge (y \leqslant x)$,
v. si $\phi = g(X_1, ..., X_l, y)$, on pose $\phi_x = g(X_1, ..., X_l, y) \wedge (y \leqslant x)$,
vi. si $\phi = \neg\psi$, on pose $\phi_x = \neg\psi_x$ et
vii. si $\phi = \psi \wedge \psi'$, on pose $\phi = \psi_x \wedge \psi'_x$.
Il reste à vérifier que l´équivalence de la proposition est vraie pour chacune de ces transformations. Celle-ci est claire pour i., ii., iv.,vi. et vii..
Traitons en détail la preuve pour iii. :
Soient donc $u = (a_1, d_1)...(a_n, d_n)$ un mot de données et $i \in [1..n]$.

Si $((a_1, d_1)...(a_i, d_i); \mu.\nu) \vDash \exists y.\psi$, il existe une valuation de y, $\mu(y) = k$, avec $k \leqslant i$, telle que $((a_1, d_1)...(a_i, d_i); \mu[y/k].\nu) \vDash \psi$. Par hypothèse d'induction, $((a_1, d_1)...(a_n, d_n); \mu[y/k][x/i].\nu) \vDash \psi_x$ et donc $(u; \mu[x/i].\nu) \vDash \exists y.(\psi_x \wedge (y \leqslant x))$.

Réciproquement, si $(u; \mu[x/i].\nu) \vDash \exists y.(\psi_x \wedge (y \leqslant x))$, il existe une valuation de y, $\mu(y) = k$, où $k \leqslant i$, telle que :

$$(u; \mu[x/i][y/k].\nu) \vDash \psi_x.$$

Par hypothèse d'induction, $((a_1, d_1)...(a_i, d_i); \mu[y/k].\nu) \vDash \psi$.
Soit, $((a_1, d_1)...(a_i, d_i); \mu.\nu) \vDash \exists y.\psi$. Ce qui achève la preuve. □

Nous noterons $\mathcal{L}_a$ l'ensemble des formules de la forme suivante :

$$\exists X_1...\exists X_k \psi \wedge \bigwedge_i \left(\forall x.\tilde{\psi_{i,x}} \Rightarrow (\bigwedge_{a \in \Sigma} R_a(x+1) \Rightarrow f_{a,i}((x+1)X_1, ..., (x+1)X_k)) \right)$$

où $f_{a,i}$ est une formule booléenne, ψ et les ψ_i sont des formules du premier ordre.

Théorème 2.2.3 *Soit L un langage de données, les deux énoncés suivants sont équivalents :*
i. L est définissable par une formule de $\mathcal{L}_a$.

ii. L est reconnu par un mécanisme à monoïde apériodique.

Preuve :
Soit L un langage de données.
Supposons que L soit définissable par une formule de $\mathcal{L}_a$.
Soit ϕ une formule de $\mathcal{L}_a$ telle que $L = L(\phi)$:

$$\phi = \exists X_1 ... \exists X_k \psi \wedge \bigwedge_{i=1..r} \left(\forall x.\psi_{i,x} \Rightarrow (\bigwedge_{a \in \Sigma} R_a(x+1) \Rightarrow f_{a,i}(X_1(x+1), ..., X_k(x+1))) \right)$$

où $f_{a,i}$ est une formule booléenne, ψ et les ψ_i sont des formules du premier ordre.
Les formules ψ et $(\psi_i)_{i=1..r}$ définissent donc des langages de données reconnus par des mécanismes à monoïde apériodique qu'on notera

$$\rho = [(Up_{n,a})_{n \in N, a \in \Sigma}, k, \sim, f],$$

de partie acceptante $G \subset N$ et

$$\rho_i = [(Up_{i_{n,a}})_{n \in N_i, a \in \Sigma}, k, \sim_i, f_i],$$

de partie acceptante $G_i \subset N_i$.
On pose $M = N \times N_1 \times ... \times N_r$, qui est apériodique car un produit de monoïdes apériodiques est un monoïde apériodique, et $F = G \times N_1 \times ... \times N_r$.
Les mises à jour vont être définies à l'aide des formules booléennes $f_{a,i}$. En effet, une formule booléenne f à k variables représente un ensemble de parties, qu'on note P_f, de $[1..k]$: une partie $I \subset [1..k]$ appartient à P_f si et seulement si la valuation v, définie par $v(x_i) = 1$ si et seulement si $i \in I$, satisfait f.

Ainsi, les formules booléennes $f_{a,i}$ définissent des ensembles de mises à jour. Si $m = (n, n_1, ..., n_r) \in M$, on pose :

$$\overline{Up_{m,a}} = \{I \subset [1..k] | \forall i \in [1..r], n_i \in G_i \Rightarrow I \vDash f_{a,i}\}.$$

On définit le mécanisme $\overline{\rho} = [\overline{(Up_{m,a})}_{m \in M, a \in \Sigma}, k, \approx, \overline{f}]$ sur le monoïde M, de partie acceptante $F \subset M$:

- $\approx$ est la relation d´équivalence engendrée par $\sim$ et la famille $(\sim_i)_{i=1..r}$:

$$R \approx R' \text{ ssi } R \sim R' \text{ et } R \sim_i R', \forall i = 1..r,$$

- $\overline{f}$ est définie ainsi :

$$\overline{f}(a, \overline{\theta}^{\approx}) = (f(a, \overline{\theta}^{\sim}), f_1(a, \overline{\theta}^1), ..., f_r(a, \overline{\theta}^r))$$

Alors, $\overline{\rho}$, en utilisant la partie acceptante F, reconnaît exactement le langage L.
Soit $u = ((a_1, d_1)...(a_n, d_n))$ un mot de données.
Si u est reconnu par $\overline{\rho}$, il existe une suite de mises à jour $(up_i)_{i=1..n}$ telle que :

- la suite $\theta_0 = \perp^k, \theta_{i+1} = up_{i+1}(\theta_i, d_{i+1}), \forall i = 0..(n-1)$ vérifie :

$$\overline{f}(a_1, \overline{\theta_1}^{\approx}) ... \overline{f}(a_n, \overline{\theta_n}^{\approx}) \in F$$

- $\forall i = 0..n-1$, si $m_i = \overline{f}(a_1, \overline{\theta_1}^{\approx}) ... \overline{f}(a_i, \overline{\theta_i}^{\approx})$, $up_{i+1} \in \overline{Up_{m_i, a_{i+1}}}$.

On définit une valuation pour les variables du second ordre $(X_i)_{i=1..k}$:

$$\nu(X_i) = \{j | i \in up_j\}.$$

Montrons que :

$$(u;\emptyset.\nu) \models \psi \wedge \bigwedge_{i=1..r} \Big(\forall x.\psi_{i,x} \Rightarrow (\bigwedge_{a\in\Sigma} R_a(x{+}1) \Rightarrow f_{a,i}(X_1(x{+}1),...,X_k(x{+}1)))\Big).$$

Il est clair que $(u;\emptyset.\nu) \models \psi$. Soient donc $i \in [1..r]$ et μ une valuation pour la variable du premier ordre x : $\mu(x) = l$ tels que $(u;\mu.\nu) \models \psi_{i,x}$.
On va montrer que $(u;\mu.\nu) \models \bigwedge_{a\in\Sigma} R_a(x+1) \Rightarrow f_{a,i}((x+1)X_1,...,(x+1)X_k)$, ce qui revient à prouver que $up_{l+1} \models f_{a_{l+1},i}$. Or d'après la proposition 2.2.1, $((a_1,d_1)...(a_l,d_l);\emptyset.\nu) \models \psi_i$ et donc, par définition des mises à jour, $up_{l+1} \models f_{a_{l+1},i}$.

Réciproquement, si $u \models \phi$, il existe une valuation ν des variables du second ordre telle que :

$$(u;\emptyset.\nu) \models \psi \wedge \bigwedge_{i=1..r} \Big(\forall x.\psi_{i,x} \Rightarrow (\bigwedge_{a\in\Sigma} R_a(x{+}1) \Rightarrow f_{a,i}((x{+}1)X_1,...,(x{+}1)X_k))\Big).$$

On définit une suite de mises à jour par :

$$\forall 1 \leqslant i \leqslant n, up_i = \{j | i \in \nu(X_j)\}.$$

La suite : $\theta_0 = \perp^k, \theta_{i+1} = up_{i+1}(\theta_i, d_{i+1}), \forall i = 0..(n-1)$ vérifie :

$$\overline{f}(a_1, \overline{\theta_1}^{\approx})...\overline{f}(a_n, \overline{\theta_n}^{\approx}) \in F$$

et, si $m_i = \overline{f}(a_1, \overline{\theta_1}^{\approx})...\overline{f}(a_{i-1}, \overline{\theta_i}^{\approx})$, on a que $up_{i+1} \in \overline{Up}_{m_i,a_{i+1}}$ par définition des ensembles $\overline{Up}_{m,a}$. On en déduit que le mot u est reconnu par le mécanisme $\overline{\rho}$.
Ainsi, L est reconnu par un mécanisme à monoïde apériodique.

Réciproquement, supposons que le langage L est reconnu par le mécanisme

$$\rho = [(Up_{m,a})_{m\in M, a\in\Sigma}, k, \sim, f]$$

où M est apériodique, en utilisant la partie acceptante $F \subset M$.
Pour tout élément m du monoïde, on note L_m le langage reconnu par ρ en prenant comme ensemble acceptant $\{m\}$. Comme le monoïde M est apériodique, il existe, pour tout $m \in M$, une formule $\exists X_1...\exists X_k.\psi_m$ de $\mathcal{L}$ ne contenant pas de formule atomique de type Xx telle que ψ_m définisse L_m. On pose $\psi = \bigvee_{m\in F} \psi_m$.
Chaque mise à jour du mécanisme $Up_{m,a}$ peut être décrite par une formule booléenne $f_{m,a}$, comme on vient de le voir ci-dessus. On pose alors :

$$\phi = \exists X_1...\exists X_k \psi \wedge \bigwedge_{m\in M} \Big(\forall x.\psi_{m_x} \Rightarrow (\bigwedge_{a\in\Sigma} R_a(x{+}1) \Rightarrow f_{m,a}((x{+}1)X_1,...,(x{+}1)X_k))\Big)$$

Montrons que $L = L(\phi)$.
Soit $u = ((a_1,d_1)...(a_n,d_n))$ un mot de données.
Si u est reconnu par ρ, il existe une suite de mises à jour $(up_i)_{i=1..n}$ telle que :

– la suite $\theta_0 = \perp^k, \theta_{i+1} = up_{i+1}(\theta_i, d_{i+1}), \forall i = 0..(n-1)$ vérifie :

$$f(a_1,\theta_1)...f(a_n,\theta_n) \in F,$$

– si $m_i = (a_1,\theta_1)...f(a_i,\theta_i), \forall i = 0..n-1$, $up_{i+1} \in Up_{m_i,a_{i+1}}$.

On définit une valuation ν des variables du second ordre :

$$\nu(X_i) = \{j | i \in up_j\}.$$

Il est clair que $(u; \emptyset.\nu) \vDash \psi$ puisque u est reconnu par ρ. Soit $m \in M$. On se donne une valuation μ de la variable du premier ordre x telle que $(u;\mu.\nu) \vDash \psi_{m,x}$. Il nous reste à démontrer que si $a \in \Sigma$ et si $a_{\mu(x)+1} = a$, alors

$$(u;\mu.\nu) \vDash f_{m,a}(X_1(x+1),...,X_k(x+1)).$$

Or d'après la proposition 2.2.1, on sait que :

$$((a_1,d_1)...(a_{\mu(x)},d_{\mu(x)});\emptyset.\nu) \vDash \psi_m.$$

Ainsi, on a : $((a_1,d_1)...(a_\mu(x),d_{\mu(x)}) \in L_m$ et $up_{\mu(x)+1} \in Up_{m,a}$. Par définition des formules boolénnes $f_{m,a}$, $(u;\mu.\nu) \vDash f_{m,a}(X_1(x+1),...,X_k(x+1))$.

Réciproquement, si $u \vDash \phi$, il existe une valuation ν des variables du second ordre telle que : $(u;\emptyset.\nu) \vDash \psi$ et

$$(u;\emptyset.\nu) \vDash \bigwedge_{m\in M}\left(\forall x.\psi_{m\,x} \Rightarrow (\bigwedge_{a\in\Sigma} R_a(x+1) \Rightarrow f_{m,a}(X_1(x+1),...,X_k(x+1)))\right).$$

On définit une suite de mises à jour $(up_i)_{i=1..n}$:

$$up_i = \{j | i \in \nu(X_j)\}.$$

Montrons que le mécanisme $\overline{\rho}$, en utilisant la suite up_i reconnaît le mot u.

La suite $\theta_0 = \perp^k, \theta_{i+1} = up_{i+1}(\theta_i, d_{i+1}), \forall i = 0..(n-1)$ vérifie

$$f(a_1,\theta_1)...f(a_n,\theta_n) \in F$$

car $(u,\nu,\emptyset) \vDash \psi$.

Il reste à vérifier que si $m_i = f(a_1,\theta_1)...f(a_i,\theta_i)$, $up_{i+1} \in Up_{m_i,a_{i+1}}$, pour $i \in [0..n-1]$. Soit $i \in [0..n-1]$.

D'après la proposition 2.2.1, $((a_1,d_1)...(a_n,d_n);\mu[x/i].\nu) \vDash \psi_{m_i,x}$ et, par suite, $up_{i+1} \in Up_{m_i,a_{i+1}}$. Le mécanisme $\overline{\rho}$ reconnaît u. Le langage L est donc définissable par une formule de $\mathcal{L}_a$. □

Conclusion

Après avoir décrit la logique proposée dans [Bou02], nous avons caractérisé logiquement les langages de données reconnus par un monoïde apériodique. Même si cette caractérisation n'a pas l'élégance de celle existant pour les langages de mots réguliers, elle souligne les ressemblances entre les théories des langages réguliers et des langages de données et donc l'intérêt du modèle étudié ici.

Chapitre 3

Un théorème de Kleene

Les théorèmes de Kleene [Kle56] et de Büchi [Büc62] montrent une équivalence entre la classe des langages reconnus par automate fini, d'une part, et la classe des langages dénotés par une *expression rationnelle*, d'autre part.
Par exemple, l'expression rationnelle $\Sigma^* ab \Sigma^*$, sur l'alphabet $\Sigma = \{a, b\}$, désigne l'ensemble des mots qui contiennent le facteur ab. Ce formalisme est adapté pour décrire des langages et donc pour écrire des *spécifications*.

Nous proposons dans ce chapitre un théorème de Kleene/Büchi pour les langages de données, similaire à celui existant pour les langages de mots réguliers. La démonstration est une adaptation d'un théorème de Kleene pour les langages temporisés proposé par P.Bouyer et A.Petit dans [BP02].
Nous étendons, ensuite, ce résultat à un modèle plus général qui englobe les cas des langages réguliers de mots, des langages de données et des langages temporisés.

On rappelle que dans cette partie, reconnaissable signifie nd-reconnaissable ; c'est à dire qu'on utilisera les termes automate de données à la place d'automate de données non déterministe.

3.1 Langages de registres

L'union de deux ensembles A et B sera notée $A \cup B$ ou $A + B$, suivant le contexte.

Définition 3.1.1 *Un mot à n-*registres *est une suite finie ou infinie de la forme :*

$$(R_0)(a_1, d_1, R_1)...(a_k, d_k, R_k)...$$

où : $a_i \in \Sigma$, $d_i \in \mathcal{D}$ et $R_i \in \mathcal{D}^n$.

Le mot à n-registres w est dit de longueur finie p (notée $|w|$) si $w \in \mathcal{D}^n \times (\Sigma \times \mathcal{D} \times \mathcal{D}^n)^p$, infini si $w \in \mathcal{D}^n \times (\Sigma \times \mathcal{D} \times \mathcal{D}^n)^\omega$.
L'ensemble des mots vides à n-registres est, par définition, $\Gamma_n = \mathcal{D}^n$.
L'ensemble de tous les langages à n-registres est noté $RW_n(\Sigma, \mathcal{D})$.

L'ensemble des langages à registres est : $RW(\Sigma,\mathcal{D}) = \bigcup_{n\geqslant 0} RW_n(\Sigma,\mathcal{D})$.

Si ω est un mot à n-registres fini et si ω' est un mot à n-registres,

$$\omega = (R_0)(a_1,d_1,R_1)...(a_p,d_p,R_p)$$

$$\omega' = (R'_0)(a'_1,d'_1,R'_1)...$$

alors, ω et ω' sont dit *compatibles* si $R_p = R'_0$. Dans ce cas, on définit la *concaténation* de ω et ω', notée $\omega\,.\,\omega'$, par :

$$\omega\,.\,\omega' = (R''_0)(a''_i,d''_i,R''_i)_{i\geqslant 1},$$

où :
- $R''_0 = R_0$,
- pour $1 \leqslant i \leqslant p$, $a''_i = a_i$ et $(d''_i,R''_i) = (d_i,R_i)$
- pour $i > p$, $a''_i = a'_i$ et $(d''_i,R''_i) = (d'_{i-p},R'_{i-p})$.

Lorsqu'elle est définie, la concaténation est associative dans le sens suivant : si ω, ω' et ω'' sont des mots à n-registres et si ω est compatible avec ω' et ω' est compatible avec ω'', alors :
- $\omega\,.\,\omega'$ est compatible avec ω'',
- ω est compatible avec $\omega'\,.\,\omega''$
- $(\omega\,.\,\omega')\,.\,\omega'' = \omega\,.\,(\omega'\,.\,\omega'')$.

3.2 Opérations sur les langages de registres

On définit ensuite naturellement la concaténation de deux langages.
Si L et L' sont deux langages à n-registres, on pose :

$L.L' = \{\omega.\omega' | \omega \in L, \omega' \in L' \text{ et } \omega \text{ est compatible avec } \omega'\} \cup \{\omega | \omega \text{ est infini et } \omega \in L\}$

La concaténation de langages est associative et Γ_n est élément neutre, à droite et à gauche.
On peut, à présent définir les itérations finie et infinie d'un langage de la façon suivante :
$L^* = \Gamma_n \cup \{\omega_1\,.\,\omega_2\,.\,...\,.\,\omega_h|$

$$\forall 1 \leqslant i \leqslant h, \omega_i \in L \text{ et}$$

$$\forall 1 \leqslant i \leqslant h-1, \omega_i \text{ est compatible avec } \omega_{i+1}\}$$

$L^\omega = \{\omega_1\,.\,\omega_2\,.\,...\,.\,\omega_h|$

$$\forall 1 \leqslant i \leqslant h, \omega_i \in L \text{ et}$$

$$\forall 1 \leqslant i \leqslant h-1, \omega_i \text{ est compatible avec } \omega_{i+1} \text{ et } \omega_h \text{ est un mot infini}\}$$

$\cup\{\omega_1\,.\,\omega_2\,.\,...|$

$$\forall 1 \leqslant i, \omega_i \in L\backslash\Gamma_n \text{ et}$$

$$\forall 1 \leqslant i, \omega_i \text{ est compatible avec } \omega_{i+1}\}$$

Proposition 3.2.1 *Soient L, L' et L'' des langages à n-registres. Alors :*
- $L \centerdot (L' + L'') = (L \centerdot L') + (L \centerdot L'')$
- $(L + L') \centerdot L'' = (L \centerdot L'') + (L' \centerdot L'')$
- $(L \centerdot L') \centerdot L'' = L \centerdot (L' \centerdot L'')$
- $L^* = \bigcup_{i \geqslant 0} L^i$, *où* $L^0 = \Gamma_n$ *et* $\forall i \geqslant 1, L^{i+1} = L^i \centerdot L$
- $L^* \centerdot L' = \sum_{i \geqslant 0}(L^i \centerdot L')$

La preuve de la proposition 3.2.1 est élémentaire et n'est pas rapportée ici.

3.3 Une équation sur les langages

Lemme 3.3.1 *Soit L_1 et L_2 deux langages de mots finis à n-registres tels que $L_1 \cap \Gamma_n = \emptyset$. Alors, l'équation :*

$$L = L_1 \centerdot L + L_2$$

où L est un langage de mots finis a pour unique solution le langage $L_1^* \centerdot L_2$

Preuve.
Montrons tout d'abord que $K = L_1^* \centerdot L_2$ est bien solution. En utilisant la proposition 3.2.1, il vient :

$$\begin{aligned} L_1 \centerdot K + L_2 &= L_1 \centerdot (L_1^* \centerdot L_2) + L_2 \\ &= (L_1 \centerdot L_1^*) \centerdot L_2 + L_2 \\ &= (L_1 \centerdot L_1^* + \Gamma_n) \centerdot L_2 \\ &= L_1^* \centerdot L_2 = K \end{aligned}$$

Soit maintenant L une solution, alors $L_2 \subset L$ et donc $L_1 \centerdot L_2 \subset L$. On a alors, par récurrence que $L_1^* \centerdot L_2 \subset L$. Ainsi, toute solution contient K. Supposons que L contienne strictement K et soit ω un mot dans $L \backslash K$ de longueur minimale. Alors $\omega \in L_2$ ou $\omega \in L_1 \centerdot L$. Le premier cas est impossible car $L_2 \subset K$. Dans le second cas, il existe $\omega_1 \in L_1$ et $\omega' \in L$ tels que $\omega = \omega_1 \centerdot \omega'$. Si $\omega' \in K$, alors, $\omega \in L_1 \centerdot K \subset K$ ce qui est absurde. Sinon, $\omega' \in L \backslash K$, et comme $L_1 \cap \Gamma_n = \emptyset$, $|\omega'| < |\omega|$ ce qui est contredit la minimalité de ω. Ainsi, l'équation a une unique solution. □

3.4 Expressions rationnelles

3.4.1 Syntaxe

Pour tout $n \in \mathbb{N}$, on définit la grammaire :

$$\begin{aligned} E_n ::= &= \emptyset | \epsilon_n | < A, a, up, B > \text{ où } A \subset \mathcal{D}^n, a \in \Sigma, up \subset \{1, ..., n\}, B \subset \mathcal{D}^n \\ & | E + E' | E \centerdot E' | E^* \end{aligned}$$

L'ensemble des expressions rationnelles est alors l'ensemble E, défini par :

$$E = \bigcup_{i \in \mathbb{N}} E_i$$

3.4.2 Sémantique

La sémantique d'une expression rationnelle est un langage de registres qui est défini inductivement, sur la structure des formules, de la façon suivante :

Les expressions triviales :

$$[\![\emptyset]\!] = \emptyset$$

$$[\![\epsilon_n]\!] = \Gamma_n = \mathcal{D}^n$$

Les expressions de registres de base :

$$[\![< A, a, up, B >]\!] = \{(R)(a, d, R') | R \in A, R' \in B, R' = up(R, d)\}$$

où $R' = up(R, d)$ signifie que $r'_i = \begin{cases} r_i \text{ si } i \notin up \\ d \text{ sinon} \end{cases}$

Union de deux expressions rationnelles :

$$[\![E + E']\!] = [\![E]\!] + [\![E']\!]$$

Concaténation de deux expressions rationnelles :

$$[\![E \centerdot E']\!] = [\![E]\!] \centerdot [\![E']\!]$$

Itération finie d'une expression rationnelle :

$$[\![E^*]\!] = [\![E]\!]^*$$

Itération infinie d'une expression rationnelle :

$$[\![E^\omega]\!] = [\![E]\!]^\omega$$

3.4.3 Langages rationnels

Un langage à n-registres L est dit *rationnel* s'il existe une expression rationnelle $F \in E_n$ telle que $[\![F]\!] = L$. L'ensemble des langages rationnels à n-registres sera noté $Rat[RW_n(\Sigma, \mathcal{D})]$. On définit également l'ensemble des langages rationnels par :

$$Rat[RW(\Sigma, \mathcal{D})] = \bigcup_{n \geqslant 0} Rat[RW_n(\Sigma, \mathcal{D})]$$

3.5 Langages reconnaissables

3.5.1 Automates de données

On redéfinit ici les automates de données. On rajoute un ensemble d'états : les états répétés, afin de reconnaître des mots de données infinis (condition à la Büchi). Ensuite, nous présenterons une nouvelle sémantique pour ces automates qui nous permettra de relier langages de registres et langages de données.

Définition 3.5.1 *Un* automate de données *sur Σ et $\mathcal{D}$ est un 7-uplet $\mathcal{A} = (Q, I, F, k, \sim, T, B)$ où Q est un ensemble fini d'états, $I \subset Q$ est l'ensemble des états initiaux, $F \subset Q$ est l'ensemble des états finaux, k est un entier, $\sim$ est une relation d'équivalence sur $\mathcal{D}^k$ d'index fini, $T \subset (Q \times \mathcal{D}^k / \sim \times \Sigma \times \mathcal{U} \times \mathcal{D}^k / \sim \times Q)$ est un ensemble fini de transitions ($\mathcal{U}$ est l'ensemble des mises à jour) et $B \subset Q$ est l'ensemble des états répétés.*

L'ensemble des automates de données à n registres sera noté $RA_n(\Sigma, D)$ et on définit également l'ensemble des automates de données :

$$RA(\Sigma, \mathcal{D}) = \bigcup_{n \geqslant 0} RA_n(\Sigma, \mathcal{D})$$

On définit la sémantique des automates :

Un *chemin* dans l'automate $\mathcal{A}$ est une suite de transitions consécutives. Un chemin fini est *acceptant* s'il aboutit dans un état final et un chemin infini est *acceptant* s'il passe une infinité de fois par des états de l'ensemble $B \subset Q$.

Un mot à n-registres $\omega = (R_0)(a_1, d_1, R_1)...$ est accepté par l'automate $\mathcal{A}$ si il existe un chemin acceptant dans $\mathcal{A}$

$$P = \ q_0 \xrightarrow{g_1, a_1, up_1, g'_1} q_1 \xrightarrow{g_2, a_2, up_2, g'_2} q_2 ...$$

tel que la suite θ_i définie par :

$$\theta_0 = R_0 \text{ et } \theta_{i+1} = up_{i+1}(\theta_i, d_{i+1})$$

vérifie $\overline{\theta_{i-1}} = g_i$ et $\overline{\theta_i} = g'_i \ \forall i \geqslant 1$.
Un tel calcul X est une *exécution* dans l'automate $\mathcal{A}$, le long du chemin P. Le mot à n-registres ω est l'étiquette de l'exécution, ce qu'on note $\omega = l(X)$.

L'ensemble des mots de registres reconnus par $\mathcal{A}$ est noté $L_R(\mathcal{A})$.

Un langage à n-registres L est *reconnaissable* si il existe un automate de données $\mathcal{A} \in RA_n(\Sigma, \mathcal{D})$ tel que $L = L_R(\mathcal{A})$. L'ensemble des langages à n-registres reconnaissable est noté $Rec[RW_n(\Sigma, \mathcal{D})]$ et on définit l'ensemble des langages reconnaissable à registres :

$$Rec[RW(\Sigma, \mathcal{D})] = \bigcup_{n \geqslant 0} Rec[RW_n(\Sigma, \mathcal{D})]$$

3.5.2 Propriétés de clôture

Proposition 3.5.1 *Soient $\mathcal{A}$ et $\mathcal{B}$ des automates de données de $RA_n(\Sigma, \mathcal{D})$. Alors,*
-il existe un automate de données $\mathcal{A} + \mathcal{B} \in RA_n(\Sigma, \mathcal{D})$ tel que :

$$L_R(\mathcal{A} + \mathcal{B}) = L_R(\mathcal{A}) + L_R(\mathcal{B})$$

-il existe un automate de données $\mathcal{A} \,.\, \mathcal{B} \in RA_n(\Sigma, \mathcal{D})$ tel que :

$$L_R(\mathcal{A} \,.\, \mathcal{B}) = L_R(\mathcal{A}) \,.\, L_R(\mathcal{B})$$

-il existe un automate $\mathcal{A}^* \in RA_n(\Sigma, \mathcal{D})$ *tel que :*

$$L_R(\mathcal{A}^*) = L_R(\mathcal{A})^*$$

-il existe un automate $\mathcal{A}^\omega \in RA_n(\Sigma, \mathcal{D})$ *tel que :*

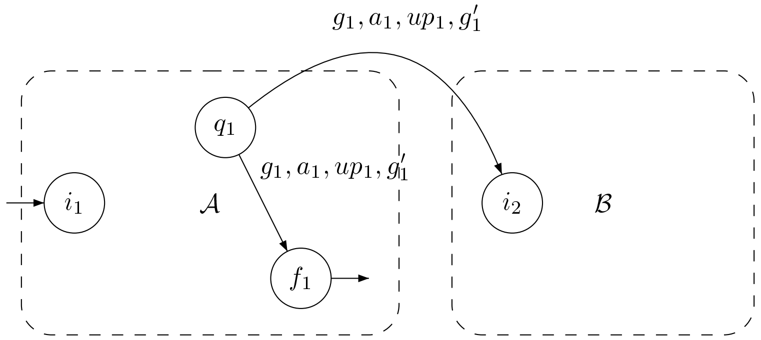
$$L_R(\mathcal{A}^\omega) = L_R(\mathcal{A})^\omega$$

Preuve :
Nous traitons en détails le cas de la concaténation :
Si $\mathcal{A} = (Q_1, I_1, F_1, n, \sim_1, T_1, B_1)$ et $\mathcal{B} = (Q_2, I_2, F_2, n, \sim_2, T_2, B_2)$, on pose, en supposant les ensembles Q_1 et Q_2 disjoints :

$$\mathcal{A} \centerdot \mathcal{B} = (Q_1 \cup Q_2, I, F_2, n, \sim, T_1 \cup T_2 \cup T, B)$$

où :

$$\begin{cases} T = \{(q_1, g_1, a_1, up_1, g_1', i_2) | i_2 \in I_2 \text{ et } \exists f_1 \in F_1 \text{ tel que } (q_1, g_1, a_1, up_1, g_1', f_1) \in T_1\} \\ I = \begin{cases} I_1 & \text{si } I_1 \cap F_1 = \emptyset \\ I_1 \cup I_2 & \text{sinon.} \end{cases} \\ \sim \ \text{ est la relation d'équivalence engendrée par } \ \sim_1 \ \text{ et } \ \sim_2 \end{cases}$$



Nous allons démontrer les inclusions réciproques entre $L_R(\mathcal{A}.\mathcal{B})$ et $L_R(\mathcal{A}).L_R(\mathcal{B})$.

Commençons par montrer que $L_R(\mathcal{A}).L_R(\mathcal{B}) \subset L_R(\mathcal{A}.\mathcal{B})$:
Un mot infini à n-registres ω de $L_R(\mathcal{A})$ est dans $L_R(\mathcal{A}).L_R(\mathcal{B})$ par définition de la concaténation. Par définition de $\mathcal{A}.\mathcal{B}$, $\omega \in L_R(\mathcal{A}.\mathcal{B})$.

Soient $\omega = (R_0)(a_1, d_1, R_1)...(a_p, d_p, R_p) \in L(\mathcal{A})$ un mot à n-registres *finis* et $\omega' = (R_0')(a_1', d_1', R_1')... \in L_R(\mathcal{B})$ tels que ω et ω' soient compatibles (i.e. $R_p = R_0'$). Il existe dans $\mathcal{A}$ un chemin acceptant P pour ω, de condition initiale R_0 :

$$P = q_0 \xrightarrow{g_1, a_1, up_1, \overline{g_1}} q_1 ... \xrightarrow{g_p, a_p, up_p, \overline{g_p}} q_p$$

et une exécution X le long de P à partir de la condition initiale R_0 telle que $\omega = l(X)$ et dans $\mathcal{B}$, un chemin acceptant P' pour ω' :

$$P' = q'_0 \xrightarrow{g'_1, a'_1, up'_1, \overline{g'_1}} q'_1 \xrightarrow{g'_2, a'_2, up'_2, \overline{g'_2}} q'_2 ...$$

et une exécution X' le long de P' de condition initiale R'_0 telle que $\omega' = l(X')$. On distingue deux cas :

Premier cas : Si $\omega \notin \Gamma_n$
Par construction de l'automate $\mathcal{A}.\mathcal{B}$, le chemin :

$$P' = q_o \xrightarrow{g_1, a_1, up_1, \overline{g_1}} q_1 ... q_{p-1} \xrightarrow{g_p, a_p, up_p, \overline{g_p}} q'_0 \xrightarrow{g'_1, a'_1, up'_1, \overline{g'_1}} ...$$

est un chemin acceptant dans $\mathcal{A}.\mathcal{B}$. Par ailleurs, ω et ω' étant compatibles, on a : $R_p = R'_0$. Ainsi,

$$X'' =< q_0, R_0 > \xrightarrow{g_1, a_1, up_1, \overline{g_1}} ... < q_{p-1}, R_{p-1} > \xrightarrow{g_p, a_p, up_p, \overline{g_p}} < q'_0, R'_0 > \xrightarrow{g'_1, a'_1, up'_1 \overline{g'_1}} ...$$

est une exécution acceptante dans l'automate $\mathcal{A}.\mathcal{B}$ sur le mot $\omega \centerdot \omega'$ et le chemin P'', de condition initiale R_0. Ainsi, $l(X'') = \omega \centerdot \omega' \in L_R(\mathcal{A}.\mathcal{B})$.

Second cas : $\omega \in \Gamma_n$
Dans ce cas, on a $I_1 \cap F_1 \neq \emptyset$, et P' est un chemin acceptant dans $\mathcal{A}.\mathcal{B}$. Ainsi, $l(X) = \omega' = \omega \centerdot \omega' \in L_R(\mathcal{A}.\mathcal{B})$.

Montrons à présent que $L_R(\mathcal{A}.\mathcal{B}) \subset L_R(\mathcal{A}).L_R(\mathcal{B})$:
Soit $\omega'' \in L_R(\mathcal{A}.\mathcal{B})$,

$$\omega'' = (R_0)(a''_1, d''_1, R''_1)(A''_2, d''_2, R''_2)...$$

Il existe, pour ω'', un chemin acceptant P'' dans $\mathcal{A}.\mathcal{B}$:

$$P'' = q''_0 \xrightarrow{g''_1, a''_1, up''_1, \overline{g''_1}} q''_1 \xrightarrow{g''_2, a''_2, up''_2, \overline{g''_2}} q''_2 ...$$

et une exécution X'' le long de P' à partir de la condition initiale R''_0 telle que $\omega'' = l(X'')$.
On distingue à présent trois cas :
Premier cas : Le chemin P'' ne passe que par des états de Q_1.
Dans ce cas, P'' est un chemin acceptant de $\mathcal{A}$ et $\omega'' \in L_R(\mathcal{A})$. Puisque $F \cap F_1 = \emptyset$, ω'' est infini et donc $\omega'' \in L_R(\mathcal{A}).L_R(\mathcal{B})$, par définition de la concaténation.

Second cas : Le chemin P'' ne passe que par des états de Q_2.
P'' est un chemin acceptant de $\mathcal{B}$ et $I_1 \cup F_1 \neq \emptyset$. On a donc que $R''_0 \in \Gamma_n \subset L_R(\mathcal{A})$ et $\omega'' = (R''_0) \centerdot \omega'' \in L_R(\mathcal{A}).L_R(\mathcal{B})$.

Troisième cas : Le chemin P'' passe par un état de Q_1 et par un état de Q_2. La définition de T implique qu'il existe un entier p tel que :

$$P'' = q_0'' \xrightarrow{g_1'',a_1'',up_1'',\overline{g_1''}} q_1'' ... \xrightarrow{g_p'',a_p'',up_p'',\overline{g_p''}} q_p'' \xrightarrow{g_{p+1}'',a_{p+1}'',up_{p+1}'',\overline{g_{p+1}''}} q_{p+1}'' ...$$

où

$$P' = q_p'' \xrightarrow{g_{p+1}'',a_{p+1}'',up_{p+1}'',\overline{g_{p+1}''}} q_{p+1}'' ...$$

est un chemin acceptant(fini ou infini) pour $\mathcal{A}$ et il existe un état $f_1 \in F_1$ tel que :

$$P = q_0'' \xrightarrow{g_1'',a_1'',up_1'',\overline{g_1''}} q_1'' ... \xrightarrow{g_p'',a_p'',up_p'',\overline{g_p''}} f_1$$

soit un chemin acceptant fini de $\mathcal{A}$. Puisque X'' est une exécution de $\mathcal{A}.\mathcal{B}$ le long de P'', de condition initiale R_0'', il est immédiat que :

$$X =< q_0'', R_0'' > \xrightarrow{g_1'',a_1'',up_1'',\overline{g_1''}} ... \xrightarrow{g_p'',a_p'',up_p'',\overline{g_p''}} < f_1, R_p'' >$$

est une exécution de $\mathcal{A}$ le long de P de condition initiale R_0''. De façon similaire,

$$X' =< q_p'', R_p'' > \xrightarrow{g_{p+1}'',a_{p+1}'',up_{p+1}'',\overline{g_{p+1}''}} < q_{p+1}'', R_{p+1}'' > ...$$

est une exécution de $\mathcal{B}$ le long de P' de condition initiale R_p''. Ainsi,

$$\omega = l(X) \in L_R(\mathcal{A}) \text{ et } \omega' = l(X') \in L_R(\mathcal{B}).$$

Finalement, $\omega = \omega' \,.\, \omega'' \in L_R(\mathcal{A}).L_R(\mathcal{B})$, ce qui termine la preuve.

Afin de construire un automate qui reconnaît l'itération finie d'un langage reconnu, on pose :
$\mathcal{A}^* = (Q \cup \{q_0\}, \{q_0\}, \{q_0\}, n, \sim, T \cup T' \cup T'', B)$ où :

$$\begin{cases} q_0 \text{ est un nouvel état qui n'est pas dans } Q, \\ T' = \{(q_0, g, a, up, g', q) | q \in Q \text{ et } \exists i \in I \text{ tel que } (i, g, a, up, g', q) \in T\} \\ T'' = \{(q, g, a, up, g', q_0) | q \in Q \text{ et } \exists f \in F \text{ tel que } (q, g, a, up, g', f) \in T\} \end{cases}$$

et pour la construction correspondant au cas de l'itération infinie, on définit :
$\mathcal{A}^\omega = (q \cup \{q_0\}, \{q_0\}, n, \sim, T \cup T' \cup T'', B \cup \{q_0\})$ où :

$$\begin{cases} q_0 \text{ est un nouvel état qui n'est pas dans } Q, \\ T' = \{(q_0, g, a, up, g', q) | q \in Q \text{ et } \exists i \in I \text{ tel que } (i, g, a, up, g', q) \in T\} \\ T'' = \{(q, g, a, up, g', q_0) | q \in Q \text{ et } \exists f \in F \text{ tel que } (q, g, a, up, g', f) \in T\}. \end{cases}$$

La démonstration du fait que ces constructions conviennent est similaire à la preuve précédente pour la concaténation. □

3.6 Un théorème de Kleene/Büchi

Proposition 3.6.1 *Soit $n \in \mathbb{N}$, on a l'égalité suivante :*

$$Rat[RW_n(\Sigma, \mathcal{D})] = Rec[RW_n(\Sigma, \mathcal{D})].$$

On en déduit immédiatement un théorème de Kleene/Büchi pour les langages de registres :

Théorème 3.6.1 *Un langage de registres est rationnel si et seulement si il est reconnaissable :*

$$Rat[RW(\Sigma,\mathcal{D})] = Rec[RW(\Sigma,\mathcal{D})]$$

3.6.1 Les rationnels sont reconnaissables

Nous avons vu que les langages reconnaissables à n-registres sont stables par les opérations d'union, de concaténation, itérations finie et infinie. Il reste à établir que les langages suivants sont reconnaissables :

- $[\![\emptyset]\!]$
- $[\![\epsilon_n]\!]$
- $[\![< A, a, up, B >]\!]$

Ce qui est immédiat.

3.6.2 Les reconnaissables sont rationnels

Nous allons utiliser l'équation de la première partie. Soit L un langage reconnaissable. Il existe un automate de données $\mathcal{A} = (Q, I, F, n, \sim, T, R)$ tel que $L = L(\mathcal{A})$. Pour tout couple d'états (q, q'), on pose :

$$\mathcal{A}_{q,q'} = (Q, \{q\}, \{q\}', n, \sim, T, \emptyset)$$

et on considère le langage à n-registres associé $L_{q,q'} = L(\mathcal{A}_{q,q'})$. On a alors l'équation suivante :

$$L(\mathcal{A}) = \sum_{(i,f)\in I\times F} L_{i,f} + \sum_{(i,r)\in I\times R} L_{i,r} \centerdot L_{r,r}^{\omega}$$

On aura prouvé que $L(\mathcal{A})$ est rationnel dès qu'on aura montré que $L_{q,q'}$ est rationnel pour tout $q, q' \in Q$. Soit $f \in Q$ fixé, alors la famille de langages de registres $(L_{q,f})_{q\in Q}$ vérifie le système d'équations suivant :

$$\begin{cases} L_{q,f} = \sum_{(q,g,a,up,g',q')\in T} L(\mathcal{A}_{g,a,up,g'}) \centerdot L_{q',f} + \Gamma_n & \text{si } q = f \\ L_{q,f} = \sum_{(q,g,a,up,g',q')\in T} L(\mathcal{A}_{g,a,up,g'}) \centerdot L_{q',f} & \text{sinon.} \end{cases}$$

On ordonne à présent les élément de Q :

$$q_1 < q_2 < \dots < q_n$$

En utilisant le lemme 3.3.1, on résoud l'équation dont le membre de gauche est $L_{q_n,f}$:

$$L_{q_n,f} = \left(\sum_{(q_n,g,a,up,g',q_n)\in T} [\![< g, a, up, g' >]\!]\right)^{*} \centerdot \left(\sum_{(q_n,g,a,up,g',q')\in T, q'\neq q_n} [\![< g, a, up, g' >]\!] \centerdot L_{q',f}(+\Gamma_n)\right)$$

On peut donc remplacer $L_{q_n,f}$ par cette formule dans les $n-1$ autres équations et continuer ainsi, récursivement, jusqu'à q_1. On obtient alors une expression rationnelle pour $L(\mathcal{A})$. □

3.7 Langages de données

On considère la fonction suivante :

$$\Pi : \bigcup_{n \geqslant 0} [\mathcal{D}^n \times (\Sigma \times \mathcal{D} \times \mathcal{D}^n)^\infty] \longrightarrow (\Sigma \times \mathcal{D})^\infty$$

$$(R_0)(a_i, d_i, R_i)_{i \geqslant 1} \longmapsto \begin{cases} (a_i, d_i)_{i \geqslant 1} & \text{si } R_0 = \perp^n \\ \boxplus & \text{sinon.} \end{cases}$$

où $\boxplus$ signifie que la fonction n'est pas définie.

On dit qu'un langage de données est rationnel s'il est l'image par Π d'un langage de registres. On a alors la proposition :

Proposition 3.7.1 *Un langage de données est rationnel si et seulement si il est reconnaissable.*

Remarque :
Si on avait autorisé des mises à jour plus générales, qui rendent le modèle plus puissant, dans les définitions d'automate de données et d'expression rationnelle, la proposition 3.7.1 resterait vraie.

3.8 Une infinité de théorèmes de Kleene

Nous étendons, à présent, le résultat précédent à un modèle plus général d'automate. Le théorème de Kleene présenté dans cette section englobe les cas des langages de mots réguliers, des langages temporisés et des langages de données. La trame de cette section est similaire à celle du début du chapitre.

3.8.1 Premières définitions

Dans la suite, $\mathcal{E}$ est un ensemble (l'ensemble des *étiquettes des transitions*) et Γ un alphabet (eventuellement infini).
On note Γ^∞ l'ensemble des mots finis ou infinis sur l'alphabet Γ.
Un *langage* est un ensemble de mots.
On se donne :

a. une fonction taille sur Γ^∞ :

$$|\ | : \quad \Gamma^\infty \quad \to \quad \mathbb{N} \cup \{\infty\}$$

telle que :
i. $|u| = 0$ ssi $u = \epsilon$.
ii. $|u| = \infty$ ssi $u \in \Gamma^\omega$,

b. une relation R sur $\Gamma^* \times \Gamma^\infty$

c. une application $\centerdot$ partiellement définie, associative (dans le même sens qu'à la fin de la section 3.1) et d'élément neutre ϵ :

$$\begin{array}{ccc} (\Gamma^* \times \Gamma^\infty) & \to & \Gamma^\infty \\ (u, v) & \mapsto & \begin{array}{ll} u \centerdot v & \text{si } (u,v) \in R \\ \boxplus & \text{sinon} \end{array} \end{array}$$

(où $\boxplus$ signifie que l'application n'est pas définie)
telle que $|u| + |v| < \infty \Rightarrow |u \centerdot v| < \infty$ et $|u \centerdot v| \geqslant |u| + |v|$.

d. une application ϕ :

$$\phi : \mathcal{E} \to \mathcal{P}(\Gamma^*)$$

3.8.2 Opérations sur les langages

On définit les itérations finie et infinie d'un langage L de la façon suivante :

$$L^* = \{\epsilon\} \cup \{\omega_1 \centerdot \omega_2 \centerdot \ldots \centerdot \omega_h | \forall 1 \leqslant i \leqslant h, \omega_i \in L\}$$

et

$$L^\omega = \{\omega_1 \centerdot \omega_2 \centerdot \ldots \centerdot \omega_h | \forall 1 \leqslant i \leqslant h, \omega_i \in L \text{ et } \omega_h \text{ est un mot infini}\}$$
$$\cup \{\omega_1 \centerdot \omega_2 \centerdot \ldots | \forall 1 \leqslant i, \omega_i \in L \backslash \Gamma_0\}$$

Proposition 3.8.1 *Soient L, L' et L'' des langages de Γ^*. Alors :*
- $L \centerdot (L' + L'') = (L \centerdot L') + (L \centerdot L'')$
- $(L + L') \centerdot L'' = (L \centerdot L'') + (L' \centerdot L'')$
- $(L \centerdot L') \centerdot L'' = L \centerdot (L' \centerdot L'')$
- $L^* = \bigcup_{i \geqslant 0} L^i$, *où* $L^0 = \{\epsilon\}$ *et* $\forall i \geqslant 1, L^{i+1} = L^i \centerdot L$
- $L^* \centerdot L' = \sum_{i \geqslant 0} (L^i \centerdot L')$

Le preuve de cette proposition est facile et n'est pas rapportée ici.

3.8.3 Une équation sur les langages

Lemme 3.8.1 *Soit L_1 et $L_2 \subset \Gamma^*$ deux langages de mots finis tels que $\epsilon \notin L_1$. Alors, l'équation :*

$$L = L_1 \centerdot L + L_2$$

où L est un langage de mots finis a pour unique solution le langage $L_1^* \centerdot L_2$

La preuve du lemme est identique à celle du lemme 3.3.1.

3.8.4 Expression $\mathcal{E}$-rationnelles

Syntaxe On définit la grammaire suivante :

$$E :== \emptyset | \epsilon_{\mathcal{E}} | e, e \in \mathcal{E} | E + E' | E \centerdot E' | E^* | E^\omega$$

Sémantique On définit la sémantique de cette grammaire inductivement sur la structure des formules :

Les expressions triviales :

$$[\![\emptyset]\!] = \emptyset$$

$$[\![\epsilon_{\mathcal{E}}]\!] = \{\epsilon\}$$

Les expressions de base :

$$[\![e]\!] = \phi(e)$$

Union de deux expressions rationnelles :

$$\llbracket E + E' \rrbracket = \llbracket E \rrbracket \cup \llbracket E' \rrbracket$$

Concaténation de deux expressions rationnelles :

$$\llbracket E \cdot E' \rrbracket = \llbracket E \rrbracket \cdot \llbracket E' \rrbracket$$

Itération finie d'une expression rationnelle :

$$\llbracket E^* \rrbracket = \llbracket E \rrbracket^*$$

Itération infinie d'une expression rationnelle :

$$\llbracket E^\omega \rrbracket = \llbracket E \rrbracket^\omega$$

Langages rationnels Un langage $L \subset \Gamma^\infty$ est dit *rationnel* s'il existe une expression rationnelle $F \in E$ tel que $\llbracket F \rrbracket = L$. L'ensemble des langages rationnels est noté $Rat_{\mathcal{E}}(\Gamma)$.

3.8.5 Langages reconnaissables

$\mathcal{E}$-automates

Définition 3.8.1 *Un* $\mathcal{E}$-automate $\mathcal{A}$ *est un* 5-*uplet :*

$$\mathcal{A} = (Q, I, F, T, R)$$

où :

i. *Q est un ensemble fini d'états.*

ii. *$I \subset Q$ est l'ensemble des états initiaux.*

iii. *$F \subset Q$ est l'ensemble des états finaux.*

iv. *$T \subset (Q \times \mathcal{E} \times Q)$ est un ensemble fini de transitions.*

v. *$R \subset Q$ est l'ensemble des états répétés.*

On définit la sémantique des automates de la façon suivante. Un mot $\gamma = \gamma_1 \cdot \gamma_2 \cdot \ldots$ est reconnu par l'automate $\mathcal{A}$ si il existe un chemin acceptant dans l'automate :

$$q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \ldots$$

tel que pour tout $i \geqslant 1$, $\gamma_i \in \phi(e_i)$.

L'ensemble des mots reconnus par $\mathcal{A}$ est noté $L_{\mathcal{E}}(\mathcal{A})$.
Un langage $L \subset \Gamma^*$ est *reconnaissable* si il existe un $\mathcal{E}$-automate qui reconnaît L.
L'ensemble des langages reconnaissables est noté $Rec_{\mathcal{E}}(\Gamma)$.

Propriétés de clôture :

Proposition 3.8.2 *Soient $\mathcal{A}$ et $\mathcal{B}$ deux $\mathcal{E}$-automates. Alors,*
-il existe un $\mathcal{E}$-automate $\mathcal{A}+\mathcal{B}$ tel que :

$$L_{\mathcal{E}}(\mathcal{A}+\mathcal{B}) = L_{\mathcal{E}}(\mathcal{A}) + L_{\mathcal{E}}(\mathcal{B})$$

-il existe un $\mathcal{E}$-automate $\mathcal{A}\,.\,\mathcal{B}$ tel que :

$$L_{\mathcal{E}}(\mathcal{A}\,.\,\mathcal{B}) = L_{\mathcal{E}}(\mathcal{A})\,.\,L_{\mathcal{E}}(\mathcal{B})$$

-il existe un $\mathcal{E}$-automate $\mathcal{A}^*$ tel que :

$$L_{\mathcal{E}}(\mathcal{A}^*) = L_{\mathcal{E}}(\mathcal{A})^*$$

-il existe un $\mathcal{E}$-automate $\mathcal{A}^\omega$ tel que :

$$L_{\mathcal{E}}(\mathcal{A}^\omega) = L_{\mathcal{E}}(\mathcal{A})^\omega$$

La démonstration est similaire à la preuve de la proposition 3.5.1.

3.8.6 Théorème de Kleene

Théorème 3.8.1 *Un langage est $\mathcal{E}$-rationnel si et seulement si il est $\mathcal{E}$-reconnaissable.*

Les rationnels sont reconnaissables Nous avons vu que les langages $\mathcal{E}$-reconnaissables sont stables par les opérations d'union, de concaténation, itérations finie et infinie. Il reste à établir que les langages suivants sont reconnaissables :
- $[\![\emptyset]\!]$
- $[\![\epsilon_{\mathcal{E}}]\!]$
- $[\![e]\!]$

Ce qui est immédiat.

Les reconnaissables sont rationnels Le preuve est similaire à celle de la proposition 3.6.1.

On déduit immédiatement de ce résultat des théorèmes de Kleene pour les langages de mots réguliers, pour les langages d'horloges et pour les langages de registres. En utilisant une projection, comme dans la section 3.7, on obtient des théorèmes de Kleene pour les langages temporisés et pour les langages de données.

Conclusion

Nous disposons à présent d'un théorème de Kleene/Büchi pour les langages de données. La preuve proposée dans la première partie du chapitre étant similaire aux preuves des théorèmes de Kleene présentées dans [BP02] et [Brz62], nous avons ensuite recherché un théorème de Kleene plus général dont les théorèmes de Kleene pour les langages de mots réguliers, les langages temporisés et les langages de données sont des corollaires.

Encore une fois, les théorie des langages de mots réguliers et des langages de données sont semblables. En effet le théorème de Kleene pour les langages de données que nous avons présenté est similaire à celui des langages réguliers, la différence majeure étant l'utilisation d'une projection pour passer des langages de registres aux langages de données.

Conclusion

Nous avons vu que beaucoup de résultats de la théorie des langages de mots réguliers s'étendent à la théorie des langages de données. Ainsi, l'équivalence entre automate fini, expressions rationnelles, logique monadique du second ordre et mécanisme à monoïde pour les langages de mots réguliers se transpose pour la théorie des langages de données. Dans le troisième chapitre, nous avons étendu l'équivalence, démontrée dans [BPT01], entre mécanisme à monoïde et automate de données et celle démontrée dans [Bou02] entre automate de données et la logique présentée dans le second chapitre, à une équivalence entre automate de données et une certaine forme d'expressions rationnelles.
Le tableau suivant récapitule les points communs et les différences entre les deux théories. Les termes en italique désignent les résultats démontrés dans ce mémoire :

Langages réguliers	**Langages de données**
Automates finis	Automates de données [BPT01]
Expressions rationnelles	*Théorème de Kleene*
MSO	Logique du second ordre [Bou02]
Monoïdes	Monoïdes [BPT01]

Nous avons proposé, dans le second chapitre, une caractérisation logique des langages de données reconnus par un mécanisme à monoïde apériodique. Ici encore, les théories des langages de mots réguliers et des langages de données sont semblables. Le tableau suivant résume la situation actuelle :

Langages réguliers	**Langages de données**
Automates sans compteur	? ? ?
Langages sans étoile	? ? ?
FO/LTL	*Logique du «premier ordre»*
Monoïdes apériodiques	*Monoïdes apériodiques*

L'étude du modèle du premier chapitre montre qu'il s'agit d'un modèle robuste puisqu'autoriser des mises à jour plus générales (mais raisonnables) ne modifie pas le pouvoir de reconnaissance du modèle.
Finalement, nous avons montré que les mécanismes de reconnaissances des langages de données permettent de simuler les réseaux de Petri.
L'étude du modèle peut se poursuivre en étudiant, par exemple, les langages reconnus par certaines classes de monoïdes, ou encore en recherchant des résultats de déterminisation ou de minimisation. Il serait également intéressant de proposer des définitions de langage sans étoile et d'automate sans compteur

dans la théorie des langages de données afin d'obtenir une équivalence semblable à celle existant pour les langages de mots réguliers.

Bibliographie

[AD90] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages, and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. 1960 International Congress of Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[Bou02] P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, vol.84(2) :75–85, August 2002.

[BP02] P. Bouyer and A. Petit. A Kleene/Büchi-like theorem for clock languages. *Journal of Automata, Languages and Combinatorics*, 2002. To appear. Available as Research Report LSV-01-12 on http ://www.lsv.ens-cachan.fr/Publis/PAPERS/BP-JALC2001.ps.

[BPT01] P. Bouyer, A. Petit, and D. Thérien. An algebraic characterization of data and timed languages. In *Proc. 12th International Conference on Concurrency Theory (CONCUR'2001)*, volume 2154 of *Lecture Notes in Computer Science*, pages 248–261. Springer-Verlag, 2001.

[Brz62] J.A. Brzozowski. A survey of regular expressions and their applications. In *Proc. IRE transactions on Electronic Computers*, volume 3, pages 324–335, 1962.

[Kam68] H. Kamp. *On tense logic and the theory of linear order.* PhD thesis, UCLA, 1968.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–40. Princeton University Press, 1956.

[Min67] M. L. Minsky. *Computation : Finite and Infinite Machines.* Prentice Hall, 1967.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57. IEEE Computer Society Press, 1977.

[RS97] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages.* Springer, 1997.

[Sch65] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, vol.8 :190–194, April 1965.

[Str94] H. Straubing. *Finite automata, formal logic, and circuit complexity.* Birkhäuser, 1994.