Devoir maison
Informatique classique

Luc Lapointe
luc.lapointe@ens-paris-saclay.fr
home.lmf.cnrs.fr/LucLapointe/

Logique de Presburger

La **logique de Presburger** permet de décrire des formules mathématiques avec des variables quantifiées et des additions. L'objet de ce devoir est de concevoir une méthode basée sur des automates qui permet de décrire l'ensemble des solutions d'une formule de la logique de Presburger.

1. Définition de la logique

1.1. Syntaxe

Une **formule de la logique de Presburger** est décrite par la grammaire suivante :

$$f \longrightarrow x = 0 \mid x = 1 \mid z = x + y \mid \neg f \mid f \wedge f \mid f \vee f \mid \exists x.f \mid \forall x.f$$

où x, y, z sont n'importe quel élément provenant d'un ensemble de variables infini[1]. Pour rappel, le sens des symboles mathématiques décrits est le suivant :

$\exists$: il existe	$\neg$: non	$\wedge$: et
$\forall$: pour tout		$\vee$: ou

Exemple La formule

$$x = x + y \wedge \exists x.x = x + z \wedge \forall u.(u = 0 \vee x = u)$$

est une formule de la logique de Presburger.

Question 1 Pour les trois formules suivantes, dire si c'est une formule de la logique de Presburger ou pas. Si non, justifier, et si oui, dessiner tous ses arbres de dérivation.

$$\forall x.x = 0 \vee x > 0 \qquad \forall x.\exists n.x = 2 * n \vee x = 2 * (n + 1)$$

$$\exists x.x = x + z \wedge \forall u.u = 0 \vee x = u + y$$

Dans une formule, une **variable libre** est une variable qui n'est pas quantifiée, c'est à dire sur laquelle ne porte aucun quantificateur existentiel $\exists$ ou universel $\forall$.

[1] Ce n'est donc pas *techniquement* une grammaire identique à celles vues en cours, qui nécessitent un alphabet fini. Mais ce détail n'a pas d'importance ici.

L'ensemble des variables libres d'une formule f, noté $\text{libre}(f)$ est défini récursivement :

$$\text{libre}(x = 0) = \text{libre}(x = 1) = \{x\} \qquad \text{libre}(z = x + y) = \{x, y, z\}$$

$$\text{libre}(\neg f) = \text{libre}(f) \qquad \text{libre}(f_1 \vee f_2) = \text{libre}(f_1 \wedge f_2) = \text{libre}(f_1) \cup \text{libre}(f_2)$$

$$\text{libre}(\exists x.f) = \text{libre}(\forall x.f) = \text{libre}(f) \setminus \{x\}$$

Question 2 Donner les variables libres de chacune des trois formules suivantes.

$$(\neg \forall x.x = x + y) \vee y = 0 \qquad (\neg \forall x.x = x + y) \vee x = 0$$

$$(\forall p.\exists n.p = n + n) \vee (\forall i.\exists n.\exists p.(p = n + n) \wedge (i = p + x) \wedge (x = 1))$$

1.2. Sémantique

Une **valuation**, souvent notée σ, est une fonction qui à des variables associe des entiers positifs. L'ensemble des variables étant infini, on se contente en général de décrire une valuation seulement sur un nombre fini de variables qui nous intéressent, notées entre crochet. Ainsi, la notation

$$[x \mapsto 1, y \mapsto 2]$$

désigne l'ensemble contenant les valuations valant 1 en x, 2 en y et n'importe quoi ailleurs. L'ensemble de toutes les valuations possibles est noté Val.

La **sémantique** d'une formule f, notée $[\![f]\!]$, est l'ensemble des valuations qui la satisfont. Elle est définie récursivement :

$$[\![x = 0]\!] = \{[x \mapsto 0]\} \qquad [\![x = 1]\!] = \{[x \mapsto 1]\}$$

$$[\![z = x + y]\!] = \{[z \mapsto n_1, x \mapsto n_2, y \mapsto n_3] \text{ tels que } n_1 = n_2 + n_3\}$$

$$[\![\neg f]\!] = \text{Val} \setminus [\![f]\!]$$

$$[\![f_1 \vee f_2]\!] = [\![f_1]\!] \cup [\![f_2]\!] \qquad [\![f_1 \wedge f_2]\!] = [\![f_1]\!] \cap [\![f_2]\!]$$

$[\![\exists x.f]\!]$ est l'ensemble des valuations σ telles qu'il existe un entier n tel que la valuation σ' définie par $\sigma'(x) = n$ et $\sigma'(y) = \sigma(y)$ ailleurs appartient à $[\![f]\!]$.

$[\![\forall x.f]\!]$ est l'ensemble des valuations σ telles que pour tout entier n la valuation σ' définie par $\sigma'(x) = n$ et $\sigma'(y) = \sigma(y)$ ailleurs appartient à $[\![f]\!]$.

Exemple $[\![\exists y.x = y + z]\!]$ est l'ensemble des valuations qui à x associe un entier plus grand que l'entier associé à z. $[\![\exists n.p = n + n]\!]$ est l'ensemble des valuations qui à p associe un entier pair.

1.3. Problème de décision

On cherche à résoudre le problème de décision suivant, dit de *satisfiabilité* de la logique de Presburger :

Entrée: Une formule f de la logique de Presburger.
Question: Est-ce que $[\![f]\!]$ est non vide ?

Autrement dit, on se demande si la formule prise en entrée admet une solution.

Exemple L'algorithme recherché doit répondre *Oui* sur l'entrée $[\![\exists y.x = y + z]\!]$ et *Non* sur l'entrée $[\![x = 0 \wedge x = 1]\!]$.

2. Encodage et automates

Un algorithme de résolution ne peut pas directement manipuler l'objet mathématique qu'est une fonction définie sur une infinité d'entiers. Il doit en manipuler un encodage.

En vue de décrire une valuation σ sur les variables $x_1, ..., x_n$, on introduit l'alphabet $\Sigma_k = \{0,1\}^k$, qui a vocation à décrire des tuples d'entiers écrits en binaire avec bit de poids fort à gauche. Plutôt que d'écrire un mot de Σ_k^* comme une suite de tuples de bits de taille k, on se permet, par confort, de l'écrire comme un mot de $(\{0,1\}^*)^k$. Autrement dit, on remplace une suite de tuples de bits par un tuple de suites de bits.

Ainsi, le mot $(1,0)(1,1)(0,0)(0,1)(0,0)$ de Σ_2^* peut se réécrire en $(11000, 01010)$, et dénote la paire d'entiers $(24, 10)$.

L'objectif va être, pour une formule f, de construire un automate, *non nécessairement déterministe*, qui reconnaît les suites de tuples de bits encodant les solutions de f - incluant ceux avec des zéros non significatifs. La taille des tuples doit être égale au nombre de variables libres dans f.

Exemple L'automate $\mathcal{A}_{\times 2}$ ci-contre reconnaît le langage des mots solutions de la formule

$$p = n + n$$

où la première composante des tuples est associée à p et la seconde à n.

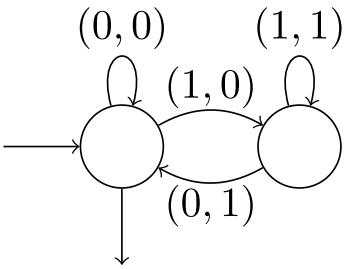

Figure 1. – Un automate $\mathcal{A}_{\times 2}$.

2.1. Cas de base

Question 3 La formule $x = 0$ a une seule variable libre, donc les tuples solution sont de taille 1. Décrire avec une expression régulière le langage des mots sur Σ_1 qui encodent une solution de cette formule, puis dessiner un automate qui reconnaît ce langage.

Lorsqu'un automate est dessiné pour une formule avec deux ou plus variables libres, il n'y a *dans le dessin* pas de correspondance explicite entre les composantes du tuple et les variables libres d'une formule. Il faut donc l'expliciter avec une phrase en plus du dessin, ressemblant par exemple à « La première composante du tuple correspond à la variable x, la seconde à la variable α, la troisième à la variable θ ».

Question 4 En s'inspirant des méthodes vues à l'école primaire d'addition posée, mais adaptées de l'écriture décimale à l'écriture binaire, dessiner un automate qui reconnaît le langage des solutions de $z = x + y$. On suppose que la première (resp. deuxième, troisième) composante des tuples de bits lus par l'automate décrivent l'entier associé à la variable x (resp. y, z). *Pour rappel, les bits de poids fort sont ceux à gauche du mot, donc les premiers lus par l'automate.*

2.2. Cas récursifs

Il est possible de réécrire une formule logique sans utiliser les symboles $\vee$ ou $\forall$ grâce aux formules de De Morgan:

$$[\![f_1 \vee f_2]\!] = [\![\neg(\neg f_1 \wedge \neg f_2)]\!] \qquad [\![\forall x.f]\!] = [\![\neg(\exists x.\neg f)]\!]$$

Nous nous contentons donc de traiter les cas $\neg$, $\wedge$, $\exists$.

Question 5 En supposant connu l'automate $\mathcal{A}$ décrivant les solutions d'une formule f, expliquer comment obtenir l'automate décrivant les solutions de la formule $\neg f$. Utiliser votre méthode pour dessiner l'automate décrivant les solutions de $\neg x = 0$ à partir de celui décrivant les solutions de $x = 0$.

Question 6 En supposant connus les automates $\mathcal{A}_1$ et $\mathcal{A}_2$ des formules f_1 et f_2, expliquer comment obtenir l'automate décrivant les solutions de la formule $f_1 \wedge f_2$. Utiliser votre méthode pour dessiner l'automate décrivant les solutions de la formule $x = 0 \wedge y = 1$ à partir de ceux décrivant les solutions de $x = 0$ et de $y = 1$.

Le cas du quantificateur existentiel est plus délicat. Une première idée, pour construire l'automate associé à $\exists x.f$ à partir de celui associé à f, est d'effacer la composante associée à x dans l'automate décrivant les solutions de f, comme en Figure 2.

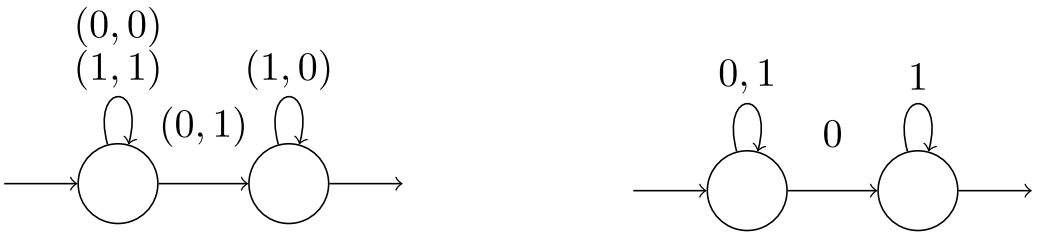

Figure 2. – À gauche, un automate décrivant les solutions de la formule $x_2 = x_1 + 1$, avec x_i en i-ème composante du tuple. À droite, le même automate en effaçant la composante liée à x_2.

Question 7 Montrer que l'automate de droite de la Figure 2 reconnaît un autre langage que celui des solutions de $\exists x_2.x_2 = x_1 + 1$.

Le problème de cet effacement de composante est qu'il rend significatifs dans l'automate à composante effacée certains zéros de x_1 qui ne le sont pas dans le premier automate.

Question 8 En supposant connu l'automate $\mathcal{A}$ décrivant les solutions d'une formule f dont une des variables libres est x, expliquer comment obtenir l'automate décrivant les solutions de $\exists x.f$. Pour ce faire, proposer une amélioration de l'idée de l'effacement de la composante liée à x, qui permet de rerendre non significatifs dans l'automate à composante effacée les zéros qui étaient non significatifs dans le premier automate. Aucune preuve que votre construction fonctionne n'est attendue.

3. Méthode de décision

Nous disposons maintenant des briques essentielles pour concevoir un algorithme qui résout le problème de satisfiabilité de la logique de Presburger avec des automates.

Question 9 Concevoir un algorithme récursif qui prend en entrée une formule de Presburger f et renvoie un automate dont le langage est celui des solutions de f. Pour ce faire, utilisez en boîte noire les résultats des questions précédentes, en indiquant clairement quelle question est utilisée où.

Une fois cet automate conçu, il est possible d'exécuter un algorithme dit de *parcours de graphe* sur l'automate obtenu pour tester s'il existe un chemin d'un état initial à un état final. C'est le cas si et seulement si le langage de l'automate est non vide, si et seulement si la sémantique de f est non vide.

En comparaison, la même logique mais avec la multiplication en plus de l'addition est indécidable.