Simulation d'algorithmes répartis

TER Licence

21 mars 2001

Table des matières

1 Le problème

1.1 Les schtroumpfs, mythe ou réalité ?

La première apparition publique des schtroumpfs date de 1958. Petit lutin à peau bleue et à bonnet blanc, le schtroumpf est un animal grégaire nichant le plus souvent dans un champignon.

Leur organisation sociale présente très peu de défauts. Chaque schtroumpf a un rôle clairement défini qu'il est heureux de remplir. Tout au plus arrive-t-il épisodiquement que l'un d'entre eux modifie son statut en changeant de spécialité, mais cela ne prête en général pas à conséquence.

Le problème. Deux grains de sable viennent perturber ce système politique presque parfait. Le premier fait son apparition dans [Pey76] sous forme d'une entité particulièrement attractive qui provoque la convergence des pensées de tous les schtroumpfs vers un objet unique : elle-même. Par malheur, cette entité (surnommée "la schtroumpfette") n'existe qu'en un seul exemplaire. Malgré la pénurie, les schtroumpfs sont trop civilisés pour se livrer à une guerre ouverte. Mais ils ne s'en trouvent pas moins confrontés à un sérieux problème : une schtroumpfette, cent schtroumpfs...

Dans [Pey78], il est fait mention qu'une des activités préférées des schtroumpfs est d'assister à un coucher de soleil. Dès lors, assister à un coucher de soleil *avec* la schtroumpfette devient une consécration. Reste à cette dernière la difficile tâche de choisir son accompagnateur.

Première solution. Pour lui permettre de faire un choix juste et équitable, le grand schtroumpf (une sorte de dictateur éclairé auquel tous obéissent) établit une tradition. Les soirs de lune bleue, tous les schtroumpfs font une grande ronde autour de la schtroumpfette. A l'un d'entre eux, il est remis la "baguette de l'orateur". L'heureux possesseur de cette baguette dispose d'une minute pour lire un poème, chanter une chanson ou faire une quelconque déclaration susceptible d'émouvoir la schtroumpfette. Ensuite, l'orateur / chanteur / déclarateur passe la baguette à son voisin de gauche qui peut également tenter sa chance pendant une minute. Et ainsi de suite jusqu'à ce que la schtroumpfette choisisse. Cette dernière étant par nature assez indécise, il n'est pas rare que le même schtroumpf puisse s'exprimer plusieurs fois. Mais les schtroumpfs sont patients...

Le vainqueur gagne tous les couchers de soleil jusqu'à la prochaine lune bleue.

Le deuxième grain de sable. Surnommé “schtroumpf farceur”, il est omniprésent dans toute l'histoire des schtroumpfs. Sous son air débonnaire et guilleret, ce psychoparanoïaque maniacodépressif dissimule une tendance compulsive et irrépressible au terrorisme systématique. Ses innombrables bombes font presque partie intégrante de la vie des autres schtroumpfs.

Cela ne serait que demi mal s'il s'en tenait là. Mais sa nature perverse le pousse à régulièrement modifier son comportement. Un jour il pose une bombe, un autre jour il offre un gâteau, un troisième il tombe lui-même dans son propre piège... Bref, le schtroumpf farceur est problématique parce que complètement imprévisible.

Même lors du cérémonial de la lune bleue, moment considéré par beaucoup de schtroumpfs comme étant sacré, il se livre à ses continuelles farces. Régulièrement, il fabrique des imitations de la baguette de l'orateur et les distribue mesquinement avant que la cérémonie ne commence. Plusieurs schtroumpfs, tous se croyant orateurs, se mettent alors à parler simultanément. Naturellement, aucun ne veut se taire de peur de perdre son temps de parole. La schtroumpfette, centre de la cacophonie, ne comprend plus ce qu'on lui dit et ne peut plus choisir.

Solution auto-stabilisante. Comme toujours, le grand schtroumpf trouva une solution. Il eut l'idée d'introduire deux types de baguettes : des rouges et des vertes. “La rouge, déclara-t-il, donne un temps de parole d'une minute, alors que la verte ne laisse que trente secondes. Et si un schtroumpf reçoit une baguette alors qu'il en a déjà une, il doit en jeter une et ne conserver que l'autre”. Ainsi fut-il fait. Les schtroumpfs acceptèrent ce nouveau règlement avec complaisance.

De son côté, le grand schtroumpf décida qu'indépendamment de la couleur des baguettes qu'il recevrait, il transmettrait une fois sur deux une baguette rouge, une fois sur deux une baguette verte.

Curieusement, en appliquant ces nouvelles règles, on constata une nette amélioration du déroulement de la cérémonie. Quand le schtroumpf farceur mettait en jeu plusieurs baguettes, les baguettes vertes circulant plus vite que les rouges, les schtroumpfs orateurs recevaient de temps en temps une deuxième baguette. Ils en jetaient alors une et progressivement, les baguettes excédentaires étaient éliminées. Bien sûr, le retour à la normale pouvait être long, mais les schtroumpfs sont patients...

1.2 Informatique et schtroumpferie

En langage informatique, chaque schtroumpf est assimilable à un processeur. La ronde des schtroumpfs est un réseau. Les instructions qu'ils exécutent sont les codes programmes des processeurs. L'ensemble des schtroumpfs exécutant leur code illustre le fonctionnement d'un algorithme réparti.

Le problème qu'ils doivent résoudre est connu sous le nom “d'exclusion mutuelle” : plusieurs processeurs veulent accéder à une même ressource (l'oreille de la schtroumpfette) et ne peuvent le faire que un par un.

La première solution proposé est une solution répartie classique : pour le résoudre, un protocole est mis en place (la première solution). Il fonctionne assez bien. Si tous les processeurs suivent leurs instructions et qu'aucune perturbation n'a lieu, il assure que tous les processeurs pourront avoir accès à la ressource.

La “mesquinerie” des ordinateurs n'a pas encore été établie de manière formelle et les célèbres lois de Murphy ne sont que des conjonctures. Néanmoins, il arrive fréquemment que les réseaux informatiques soient sujets à des perturbations. Elles sont ici représentées par la distribution de fausses baguettes, œuvre du schtroumpf farceur [1].

Qu'elles frappent le réseau des schtroumpfs ou celui de systèmes informatiques plus classiques, les pannes ont bien souvent un effet fatal sur la bonne marche des opérations. Ainsi, la schtroumpfette n'entend plus les messages qui lui sont adressés.

L'auto-stabilisation est une propriété des algorithmes répartis leur permettant de corriger automatiquement un certain type de panne. Dans notre exemple, la solution auto-stabilisante permet au système de revenir progressivement vers une situation où un seule baguette subsiste. Ce faisant, elle permet a l'algorithme de retrouver un comportement correct moyennant une petite phase transitoire où des perturbations sont encore présentes.

2 Cahier des charges

Le grand schtroumpf s'est récemment mis a Internet. Il est en train de se faire une page web et souhaiterai y inclure une simulation de la cérémonie de la lune bleue. L'idée est de permettre à un

1. Schroumpf farceur a qui nous présentons toutes nos excuses pour les qualificatifs péjoratifs dont nous l'avons affublé...

internaute visitant sa page de faire les choses suivantes :

1. Créer un graphe de processeurs : un processeur se modélise par un nombre fini de variables entières. Un graphe de processeur est un ensemble de processeurs dont certains sont reliés par des liens de communication.
2. Initialiser le graphe : pour chaque processeur, l'utilisateur doit pouvoir définir l'état initial des variables.
3. Spécifier un code : l'utilisateur doit également pouvoir munir chaque processeur d'un code à exécuter.
4. Exécuter l'algorithme réparti : quand tout est prêt, l'utilisateur doit avoir la possibilité de simuler l'exécution de l'algorithme réparti qu'il vient de définir.

2.1 Créer un graphe de processeurs

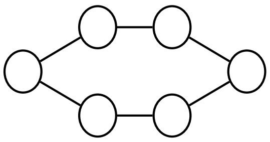
Un processeur se modélise par un nombre fini de **variables entières ou décimales**. Naturellement, chaque variable à un nom.

Un graphe de processeur est un ensemble de processeurs dont certains sont reliés par des liens de communication. Les processeurs d'un graphe peuvent être :

- Anonyme : tous les processeurs sont rigoureusement identiques ET ont le même code.
- Semi-anonyme : tous les processeurs sont rigoureusement identiques et ont le même code SAUF un qui peut avoir des variables et un code différant des autres.
- Non-anonyme : les processeurs peuvent avoir des variables et un code différant les uns des autres.

Pour un processeur P donné, l'ensemble des processeurs avec lesquels il est relié est appelé ensemble des voisins de P. Cet ensemble peut éventuellement être ordonné (P a un premier voisin, un deuxième, ...) **mais l'utilisateur ne doit pas en avoir connaissance et ne doit pas pouvoir utiliser cette information dans la définition des prédicats, des actions et des règles.**

Exemple :

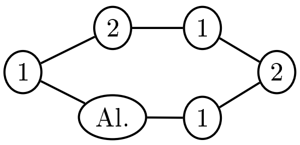


Certains graphes ont des propriétés particulières. Par exemple, sur un anneau, il est possible de définir une orientation. On peut donc envisager de proposer un choix à l'utilisateur : soit un graphe prédéfini (comme un anneau orienté, ou un arbre binaire), soit un graphe quelconque qu'il devra lui-même définir.

2.2 Initialiser le graphe

Chaque variable d'un processeur doit disposer d'une valeur à prendre lorsque l'exécution de l'algorithme commence. Cette valeur doit pouvoir être fixée par l'utilisateur OU être tirée aléatoirement en début d'exécution (c'est à dire, l'utilisateur décide qu'elle doit être tirée aléatoirement).

Exemple :



2.3 Spécifier le code

Le grand schtroumpf est sympa, il ne vous demande pas de programmer un analyseur syntaxique. Il se contentera d'utiliser des programmes définissables grâce à des menus déroulants **ou toute autre méthode qui vous semblera adéquat.**

Formellement, le code programme d'un processeur P est constitué d'un groupe de prédicat, d'un groupe d'actions possible et d'un ensemble de "règles gardée".

Un prédicat est une condition portant sur les variables d'un processeur ou sur les variables de ses voisins. Pour simplifier, les seuls prédicats considérés seront de la forme :

$$Predicat \Leftrightarrow V_1 \ Bin \ V_2$$

$$Predicat \Leftrightarrow (V_1 \ Op \ x) \ Bin \ (V_2 \ Op \ x')$$

avec V_1 et V_2 variables de P ou d'un de ses voisins, Op opérateur arithmétique $(+, - , \times ,/)$, Bin opérateur binaire $(= , \neq , < , \leq , > , \geq)$ et x,x' deux nombres (entiers ou décimaux).

Une action est simplement une opération d'affectation de la forme :

$$Action \Leftrightarrow V_1 \longleftarrow x$$

$$Action \Leftrightarrow V_1 \longleftarrow V_2 \ Op \ x$$

avec V_1 variable de P, V_2 variable de P ou d'un de ses voisins, Op opérateur arithmétique $(+, - , \times ,/)$ et x un nombre (entier ou décimal).

Une règle gardée est alors une implication $Predicat \Longrightarrow Action$. **Quand plusieurs règles sont définies, elles sont numérotées.**

Exemple : on suppose que chaque processeur dispose d'une unique variable *Orateur* et que le processeur P a deux voisins P_1 et P_2.

Prédicat
$Privilege(P) \quad \Leftrightarrow Orateur(P) = Orateur(P_1)$

Action
$Transmettre(P) \quad \Leftrightarrow Orateur(P) \longleftarrow Orateur(P_1) + 1$

Règle gardée
$R1 \quad Privilege(P) \quad \Longrightarrow Transmettre(P)$

De la même manière que pour les graphes, on peut également prédéfinir un certain nombre de programmes que l'utilisateur pourra selectionner sans avoir besoin de spécifier lui-même un code. Des exemples de programmes possibles vous seront fournis ultèrieurement.

2.4 Exécuter l'algorithme réparti

Quand tout est prêt, l'utilisateur doit pouvoir simuler l'exécution de l'algorithme réparti qu'il vient de définir. Il existe plusieurs manières de modéliser ces exécutions. L'utilisateur en choisit une, puis il "clique". A chaque "clic", selon le mode choisi, il se passe quelque chose :

- Mode synchrone : après un clic, TOUS les processeurs évaluent simultanément leur prédicats, puis TOUS les processeurs exécutent simultanément leurs règles gardées ayant leur prédicat à vrai.
- Mode asynchrone : un unique processeur choisi par l'utilisateur (celui sur lequel il clique) évalue ses prédicats, **puis il exécute dans l'ordre, dans l'ordre ou elles sont numéroté, toutes ses règles gardées ayant leur prédicat à vrai.**
- Mode aléatoire : un unique processeur choisi aléatoirement évalue ses prédicats, puis il exécute simultanément ses règles gardées ayant leur prédicat à vrai.

L'utilisateur peut donc suivre à l'écran l'exécution simulée de son algorithme réparti.

3 C'est clair ?

Alors à vos stylos...

4 Erratum

Suite à vos nombreux commentaires, le commanditaire du programme s'est rendu compte d'un certain nombre de problèmes[2] :

- Les actions sont des affectations. Mais un processeur ne peut modifier que ses variables propres, et non pas les variables de ses voisins. Donc, les actions doivent être de la forme :

$$Action \Leftrightarrow V_1 \longleftarrow x$$

$$Action \Leftrightarrow V_1 \longleftarrow V_2\ Op\ x$$

avec V_1 variable de P, V_2 variable de P ou d'un de ses voisins, Op opérateur arithmétique $(+,-,\times,/)$ et x un nombre (entier ou décimal).
- Il est dit paragraphe 2.1 que P a un voisin, un deuxième,... Ce n'est pas obligatoire. P peut tout aussi bien avoir un ensemble de voisins (et non une liste). Dans tous les cas, l'utilisateur ne doit pas en avoir connaissance. En particulier, il ne doit pas pouvoir utiliser le fait qu'un voisin de P aie un "numéro d'ordre" plus petit (ou plus grand) qu'un autre des voisins de P.
- En mode asynchrone, lorsqu'un processeur exécute ses règles, il est dit qu'il "évalue ses prédicats, puis il exécute simultanément ses règles gardées ayant leur prédicat à vrai". Concrètement, il évalue tous ses prédicats, puis exécute une à une toutes les règles dans l'ordre défini par l'utilisateur. En particulier, si une même variable est modifiée plusieurs fois, la dernière modification est la bonne.
- Première ligne du paragraphe 2.1, il est dit qu'un processeur est un ensemble de variables entières. Or il peut être un ensemble de variables, chacune pouvant être entières OU décimales (comme ça l'est précisé par la suite).

Il a aussi constaté que ses souhaits n'étaient pas toujours très bien formulés. Aussi a-t-il décidé de préciser sa demande. En particulier :

- Pour permettre à un processeur d'acceder aux variables de ses voisins, il est nécéssaire d'introduire des "quantificateurs" ou des "rôles"[3]. La structure d'un prédicat devient donc :

$$Predicat \Leftrightarrow Quant_1\ V_1\ Bin\ Quant_2\ V_2$$

$$Predicat \Leftrightarrow (Quant_1\ V_1\ Op\ x)\ Bin\ (Quant_2\ V_2\ Op\ x')$$

avec $Quant_1$ et $Quant_2$ deux quantificateurs, V_1 et V_2 variables de P ou d'un de ses voisins, Op opérateur arithmétique $(+,-,\times,/)$, Bin opérateur binaire $(=\ ,\ \neq\ ,\ <\ ,\ \leq\ ,\ >\ ,\ \geq)$ et x,x' deux nombres (entiers ou décimaux). On note que "Moi" (ou "Moi-même", ou "Self", ou " ") doit obligatoirement faire parti des quantificateurs.

De même, la structure d'une action devient :

$$Action \Leftrightarrow Moi.V_1 \longleftarrow x$$

$$Action \Leftrightarrow Moi.V_1 \longleftarrow Quant\ V_2\ Op\ x$$

On note que les quantificateurs peuvent être de différentes formes, le commanditaire n'étant pas encore vraiment fixé[4].

Références

[Pey76] Peyo. "La Schtroumpfette". Ed. Dupuis.

[Pey78] Peyo. "52 Histoires de Schtroumpfs". Ed. Dupuis.

2. Les trois modifications suivantes ont déjà été intégrées dans le présent document.
3. Que chacun garde l'appellation qu'il a utilisé en TP.
4. En clair, vous pouvez garder la structure que vous avez présenté en TP ou, si vous avez d'autres idées, l'améliorer.