PROJET IA02

Semestre P18

Thibault METAIS et Pierre ROMON

Les prédicats

Nous avons commencé tout d'abord par établir 9 listes de 9 nombres initialisés à 0. En effet, à ce moment-là, nous n'étions pas bien renseignés sur les listes de listes car il était encore tôt au vu de l'avancement du semestre. Dans une seconde version, nous avons réalisé une liste contenant elle-même 9 listes pour plus de propreté et moins de redondances. Enfin, la dernière version de notre programme accepte la création de grilles de divers formats.

Nous allons ici parler de la dernière version du programme, tout en faisant néanmoins un point à la fin de ce rapport sur l'intérêt de l'utilisation de listes de listes d'après notre expérience au fil des versions.

Nous avons choisi de réaliser un code le plus modulaire et concis possible : nous avons donc commencé par définir ce qu'était un **nombre** avec le prédicat du même nom, ainsi qu'une **taille** valable avec le prédicat du même nom.

Nous avons aussi défini ce qu'était un nombre **valide** selon la taille passée en paramètre. Nous avons également défini afficher ligne, prédicat appelé récursivement au sein de **afficher sodoku** qui a pour but d'afficher la grille.

Pour vérifier la grille, nous avons choisi de créer un prédicat pour tester si les nombres d'une liste sont **tous différents**. Nous avons créé des prédicats capables d'extraire une liste à partir d'une ligne, d'une colonne ou d'un bloc selon la taille de la grille. On peut ensuite appliquer **tous différents** à chacun d'entre eux afin de tester la validité de la grille dans son ensemble. Nous avons donc un prédicat **valide grille** qui contient **valide lignes**, **valide colonnes** et **valide blocs** qui appellent chacun récursivement **tous différents** sur les différentes listes à tester après les avoir extraites de la grille.
Afin d'extraire des éléments de la grille de différentes manières pour les besoins des prédicats précédents, on a défini les prédicats **extraire ligne**, **extraire partie ligne**, **trouve élément ligne** et **trouve élément grille** (qui fait appel à trouve élément ligne récursivement). Et pour constituer des listes à partir de ces éléments extraits, on a défini les prédicats **concat** et **concat listes** qui concatène un élément à une liste pour le premier et qui concatène deux listes pour le second.

Tous différents utilise récursivement le prédicat **élément** récursivement pour savoir si l'élément de tête est contenu dans la queue. On continue ensuite avec la queue pour faire de même.

Nous avons défini **modifier ligne** qui prend une liste en paramètre, une position et un nombre et qui renvoie dans son dernier paramètre la liste modifiée.
Modifier grille se base sur ce même prédicat récursivement afin de modifier un élément dans une grille avec cette fois les coordonnées en X et Y en tant que paramètres.
Ces deux prédicats renvoient faux si jamais le nombre à modifier est différent de 0.

Pour tester si un changement est valable, et donc potentiellement pour tester toutes ou une partie des changements possible, on a créé le prédicat **changement valide** qui teste si X, Y et

N sont valides, si la grille est modifiable, et enfin si celle-ci est valide (via les prédicats expliqués précédemment). Cette structure permet donc d'explorer potentiellement toutes les possibilités.

Avant de commencer le prédicat de résolution, nous avons dû tester si une case était vide. Pour cela, on a défini le prédicat **vide** qui teste si un élément à une certaine position dans une liste est égal à 0 (il utilise pour cela **trouve élément ligne** à la même position).
Vide grille possède la même fonction mais à l'échelle de la grille. Il utilise pour cela **extraire ligne** pour y appliquer le prédicat **vide** vu juste avant.

Le prédicat **résolution** peut donc maintenant être réalisé. Il utilise pour cela le prédicat **valide** successivement sur X et Y passés en paramètre pour tester les diverses positions, puis le prédicat **vide grille** avec ces mêmes coordonnées pour tester si la case voulue est vide. Puis, on utilise le prédicat **valide** avec le nombre N à placer dans la grille qui explorera également les diverses possibilités. Enfin, on fait suivre les prédicats **modifier grille** et **valide grille** à la manière de **changement valide** afin de modifier la grille passée en paramètre. On appelle enfin **résolution** avec la nouvelle grille afin d'instaurer la récursivité.
Un point essentiel ici est la présence d'une coupure juste après le prédicat **vide grille.**
Nous avons réalisé cette nécessité après les premiers tests pauvres en performances de ce prédicat : en effet, si la case située en X et Y ne comporte aucune solution N valable, le prédicat se doit de renvoyer faux car aucune case ne peut rester vide.

On a également défini juste avant un autre prédicat **résolution** qui teste si la grille est pleine avant d'afficher cette grille : cela fait office de condition d'arrêt de la récursivité.
Le prédicat **grille pleine** utilise le prédicat **élément grille** qui utilise récursivement **extraire ligne** et **élément** pour avoir si la grille ne comporte plus de 0.

Le prédicat **jouer** permet de remplir le sudoku via le programme.
Il en existe 4 définitions : la première partie commune aux 4 prédicats est l'appel du prédicat **valide** successivement sur les coordonnées X et Y ainsi que sur le nombre à placer N.
Ensuite ils se distinguent :
Le premier modifie la grille, la valide, teste si elle est pleine et l'affiche avec un message de félicitations. Le second la modifie, la valide, l'affiche, puis lit les nouvelles valeurs X, Y et N pour le nouveau tour de jeu avant d'appeler jouer récursivement. Le troisième la modifie, ne la valide pas, affiche un message d'erreur à l'utilisateur, puis lit les nouvelles valeurs X, Y et N pour le nouveau tour de jeu avant d'appeler jouer récursivement. Le dernier quant à lui, ne peut pas la modifier (la case est déjà remplie) et lit donc les nouvelles valeurs X, Y et N pour le nouveau tour de jeu avant d'appeler jouer récursivement.

Le prédicat **génère** prend en paramètre un nombre de nombres à placer dans une grille en paramètre. Il accepte aussi un nombre entre 1 et 81 correspondant à une position dans la grille, en déduit les coordonnées X et Y grâce à une formule mathématique. Il teste ensuite un nombre valide N, tente de modifier la grille et si celle-ci est valide. Enfin, il génère un nouveau nombre aléatoire avant d'effectuer un appel récursif.
Deux autres définitions de ce prédicat sont rédigées afin de gérer une modification impossible (case vide) ou d'une modification non valide (non respectueuse des règles) qui ne décrémentent pas le compteur de nombres à placer contrairement au précédent.

Une dernière placée en première stoppe le prédicat quand le compteur tombe à zéro pour appeler le prédicat **test**.

Le prédicat **test** essaye de résoudre la grille afin de voir si elle est résolvable, et sinon crée une nouvelle grille vide avant d'appeler à nouveau **génère**.
Si elle est résolvable, il appelle le prédicat **préparer**.

Le prédicat **préparer** prend en paramètre un nombre de cases à vider et un nombre aléatoire entre 1 et 81 correspondant à une position dans la grille, essaye de modifier la grille passée en paramètre pour y placer un zéro (grâce à un prédicat **vider grille**) avant de décrémenter le compteur (si succès) et de faire un appel récursif.
La condition d'arrêt appelle le prédicat **play** quand le compteur tombe à zéro.

Le prédicat **play** prend en compte le choix de l'utilisateur (jouer ou demander la résolution) et l'oriente en fonction.

Le prédicat **sudoku** prend en paramètre une difficulté D et une taille S, crée une **grille vide** et appelle le prédicat **génère**.

Critique du programme

L'un des plus grands enjeux pour nous était la modularité du programme, d'où notre décision au fil des versions de modifier la structure de données de notre grille de sudoku.

Les 9 listes de départs multipliaient de manière inutile les paramètres des prédicats ou alors multipliaient les versions des prédicats par 9, rendant le programme très peu modulable et moins viable du point de vue logique.

Avec la configuration actuelle, le programme est fonctionnel du point de vue logique pour diverses tailles de grille car nous avons pensé les prédicats en conséquence.

Néanmoins, du point de vue performances, le sudoku ne peut excéder une grille de 9 par 9. En effet, la génération d'une grille demande trop de calculs pour que cela réussisse la plupart du temps. Elle peut néanmoins résoudre une grille dont il manque un petit nombre de chiffres (autour de 10).

Ceci est très certainement lié à la stratégie de génération de grille que nous avons adoptée, mais aussi de la rédaction sûrement imparfaite de certains prédicats (absence de coupures) qui multiplient le nombre de combinaisons à tester et donc de temps de calcul.

On aurait donc peut-être pu avec plus de temps réfléchir à de nouvelles manières de résoudre ce problème et de générer des grilles différentes et valables de manières efficace, mais aussi réfléchir au placement de nouvelles coupures qui auraient pu limiter l'arborescence des solutions.