Rapport du Groupe 4

Calculabilité

Florian JACQUOT
Sandra JAGODZINSKA
Valentina OSETROV
Agathe WALLET

17 Janvier 2023

1 Générateur de code MTdV à partir de la MTdV+

Dans le cadre du projet collectif, le groupe 4 a été responsable de la génération d'un script MTdV, conforme à la syntaxe classique, à partir d'un script MTdV+ incluant des variables.

1.1 Les éléments nécessaires

Afin d'assurer le bon fonctionnement du projet, nous avons eu pour objectif de récupérer plusieurs éléments essentiels, notamment :

- Les variables présentes dans le script en MTdV+ et leurs valeurs au fur et à mesure du script (groupe 2)
- Les éléments hors variables (groupe 1)
- L'adresse mémoire des variables, des constantes et de la mémoire vive (groupe 3)

1.2 Les nouveaux éléments introduits

Pour asssurer une exécution fluide, nous avons introduit les éléments suivants :

- Une variable 'position' qui définit la position actuelle de la tête sur la bande pour pouvoir se déplacer soit vers l'adresse des variables ou constantes que l'on souhaite modifier ou utiliser, soit vers l'adresse de la mémoire vive dans laquelle les calculs seront effectués
- Une fonction pour copier une variable/constante à une adresse donnée
- Une fonction pour comparer deux variables/constantes
- Une fonction pour générer le script en MTdV

1.3 Répartition du travail

Nous nous sommes rencontrés pour faire un point sur ce que nous devions faire pour pouvoir générer les scripts en MTdV à partir des sorties que nous devrions récupérer des autres groupes et avons fait une attribution des tâches que nous avons identifiées :

- Utiliser les sorties des autres groupes → Valentina
- Créer une fonction 'copie' pour pouvoir copier le contenu d'une adresse à une autre adresse → Agathe
- Créer une fonction 'comparaison' pour pouvoir comparer deux variables entre elles (directement au niveau des adresses de chacune) → Sandra

- Comprendre l'utilisation d'‘addition.TS‘ et ‘multiplication.TS‘ pour les adapter à notre script → Florian
- Gestion des triviaux → Valentina
- L'appel des fonctions dans la structure principale (main) → Commun
- Adaptation de la présentation (version 2) → Commun
- Rédaction du rapport → Commun

1.3.1 Copie

La fonctionnalité de copie du contenu d'une adresse à une autre adresse a nécessité la génération des trois fonctions suivantes :

La première fonction **"copie_variable"** calcule la différence de position entre l'adresse de destination et la position actuelle de la tête de bande, nous donnant la distance d entre les deux adresses. En fonction de la position actuelle de la tête de bande, on la fait se déplace vers la gauche ou la droite pour atteindre l'adresse de destination et on place un "0" au premier "bit" de l'adresse de destination. Ensuite, la fonction initialise une boucle de copie. Elle se positionne au niveau du deuxième bit de l'adresse source, le premier étant un "0" déjà copié. Tant que la tête de bande rencontre un "1" elle se déplace et copie les bâtons à l'adresse de destination puis se déplace d'un cran sur la droite, permettant de faire avancer la tête de boucle pour lire et écrire la valeur que l'on copie. Lorsque la tête de bande atteint un "0", indiquant la fin de la variable, elle écrit un "0" à l'adresse de destination et met fin à la boucle de copie. Cette fonction gère les déplacements vers la gauche ou la droite en fonction de la comparaison entre les adresses de destination et de provenance et elle adapte la position de la tête de bande en conséquence. À la fin de la copie, ne sachant pas où nous sommes dans l'adresse de destination puisque la valeur écrite est inconnue, nous devons revenir au début de l'adresse de destination. Cette fonction ne renvoie le script de Turing correspondant.

La deuxième fonction **"revenir_debut_adresse"** a donc pour objectif de déplacer la tête de bande au début de l'adresse de la variable actuelle. Elle fonctionne en initialisant d'abord une boucle. Elle effectue un déplacement vers la gauche, plaçant la tête de bande au niveau du "dernier" "1" de l'adresse variable actuelle. Elle continue de se déplacer vers la gauche tant qu'elle ne rencontre pas le "0" d'initialisation. Une fois que le "0" est détecté, la boucle se termine, et la position actuelle de la tête de bande correspond à l'emplacement du "0" d'initialisation, c'est-à-dire le début de l'adresse de la variable. Enfin cette fonction renvoie le script généré ainsi que la nouvelle position de la tête de bande.

La troisième fonction **"copie_deux_variables_memoire_vive"** copie deux variables dans la mémoire vive. Cela commence par la génération du script de la copie de la première variable (adresse_1) vers la mémoire vive (adresse_mv). Pour cela, elle utilise la fonction "copie_variable", et met à jour la position actuelle de la tête de bande. Ensuite, la fonction génère le script pour revenir au début de l'adresse de

la variable dans la mémoire vive en utilisant la fonction "revenir_debut_adresse". Le script généré jusqu'alors est concaténé avec celui de la copie de la deuxième variable (adresse_2) vers une nouvelle adresse dans la mémoire vive (adresse_mv_2), se trouvant 32 bits plus loin que l'adresse de la mémoire vive, permettant de copier la plus grande valeur possible dans la première partie de la mémoire vive. La position de la tête de bande est alors donc mise à jour. Un autre script est généré pour revenir au début de l'adresse de la deuxième variable dans la mémoire vive. Enfin, le script final est obtenu en concaténant l'ensemble des scripts générés pour les différentes étapes et est retourné avec la position actuelle de la tête de bande.

1.3.2 Comparaison

Cette fonction permet de comparer deux éléments sans interprétation, en naviguant entre les adresses dans la mémoire des éléments. Les adresses d'éléments à un état donné sont fournis par la groupe 3. Une fois connus, il est nécessaire de calculer la distance entre les deux adresses dans la mémoire et leurs positionnement relatif. Cela permet de déterminer combien de fois la tête de lecture doit être déplacée et dans quelle direction pour naviguer entre les deux variables.

Au début de la boucle, la tête de lecture est déplace vers la droite, permettant ainsi la comparaison des valeurs une par une. Ensuite, la tête de lecture se positionne sur le premier bâton de la variable 1 et vérifie s'il s'agit effectivement d'un bâton. Par la suite, elle se déplace vers la position équivalente dans la variable 2. Si la valeur à cette position diffère de celle de la variable 1, la tête retourne au début de la variable 1. Sinon, la tête revient à l'endroit équivalent dans la variable 1. Ce processus se répète jusqu'à la fin de la variable 1.

Une fois que la tête atteint le 0 final de la variable 1, il est nécessaire de vérifier si la variable 2 contient également un 0 au même endroit. Sinon, les variables ne sont pas égales.

À la toute fin, le segment comparaison se termine avec un `si(1)` pour renvoyer `true` ou `false` dans le programme général. La résultat de la condition dépend de la position sur la bande.

1.3.3 Les opérations : addition et multiplication

Nous traitons deux types d'opérations : l'addition et la multiplication.

La fonction **"addition"** commence par générer un script d'opération d'addition, marqué par le commentaire "% ADDITION". Ensuite, elle appelle la fonction appelée copie_deux_variables_memoire_vive pour copier les deux variables adresse_var1 et adresse_var2 dans la mémoire vive. La position de la tête de bande (pos_bande) et la position dans la mémoire vive (pos_memoire_vive) sont mises à jour en conséquence. La fonction adapte la position de la tête de bande pour se positionner sur le premier "bâton" (élément) de la première valeur stockée dans la mémoire vive. Ensuite, elle lit le contenu du fichier "TS/addition.TS" (un fichier contenant des instructions spécifiques à l'opération d'addition en MTdV classique) et l'ajoute au script. Enfin,

la fonction renvoie le script généré ainsi que la position mise à jour de la tête de bande.

La fonction **"multiplication"** a été conçue de la même manière que la fonction "addition" à l'excéption du fait que'elle effectue l'opération de multiplication sur deux variables et qu'elle lit le contenu du fichier "TS/multiplication.TS". De plus le script a été modifié par rapport au script fournit pour que le résultat fournit par celui-ci soit placé au début de la mémoire vive, permettant ainsi de connaître facilement la position finale.

1.3.4 Le nettoyage de la mémoire

Nous avons procédé au nettoyage systématique de la mémoire vive après chaque opération effectuée pour qu'il n'y ait pas de corruption lors des prochaines opérations. Pour cela, nous avons créer une fonction **nettoyage_mv()**. Une fois que l'opération a été effectuée et que la valeur a été copiée à l'adresse voulue, il ne reste qu'une valeur dans la mémoire vive, dont le "0" initialisateur se situe à l'adresse de la mémoire vive. Nous nous déplaçons donc au niveau de la mémoire vive pour nous nous déplaçons jusqu'au "0" fermant la valeur. Ensuite, on retourne en arrière en remplaçant chaque "1" par un "0" jusqu'à atteindre le "0" qui initialisait la variable et donc le premier bit de la mémoire vive. Cette fonction retourne le script correspondant et la position de la tête de la bande.

1.3.5 Les triviaux

Les caractères dits "triviaux" sont ceux qui ne changent pas de sens entre MTdV+ et MTdV. En particulier, il s'agit de : "#", "I", "P", "boucle", "fin" et "}". Afin de les identifier, nous avons créé une fonction **"affichage_triviaux"** qui récupère tous les triviaux depuis le fichier tabulaire (tsv) pour les stocker dans un dictionnaire de la façon suivante : {n° d'instruction : {'-' : token trivial}} . Le '-' représentant la position d'instruction qui, dans ce cas, sera toujours la même, car ils ne se trouvent ni à gauche (G), ni à droite (D) ni au milieu (M). Enfin, au moment de la génération, ces tokens seront simplement réécrits.

1.4 Défis rencontés

Du fait de l'interdépendance totale de ce projet, progresser dans la gestion du générateur, en tant que dernier groupe, s'est révélé être un défi non négligeable. Il a fallu comprendre comment fonctionnait le script dans sa globalité, comment les autres modules interagissaient les uns avec les autres au sein du script principal. Néanmoins, la réactivité de chaque membre du projet, que ce soit à nos questions ou à nos requêtes, nous a permis de nous approprier le script rapidement pour y intégrer nos outils de génération du script final.

2 Le management du projet

Dans le cadre de ce projet collectif, le groupe 4 a pris la responsabilité du management du projet, en supervisant les tâches assignées à chaque équipe pour s'assurer de la cohérence et de la progression du projet. Essentiellement, nous devions anticiper le moment où chaque équipe achèverait sa partie, car nos tâches étaient interdépendantes les unes des autres. De plus, il était crucial de maintenir une communication constante, entre les différents groupes, sur les axes d'amélioration repérées en cours.

Ce rôle nous a notamment été facilité par le choix collectif de l'utilisation d'un répertoire collaboratif GitHub[1]. Cela nous a permis de suivre l'état de progression et les modifications effectuées par les différents groupes (organisés par branches). Les tâches de chaque groupe étant bien réparties et Discord nous permettant de communiquer facilement, les problèmes de compatibilité ou autre que nous avons pu rencontrer ont été réglés efficacement.

[1] https://github.com/Araule/Projet_MTdV2023/