Projet 2021

Tom Redon

May 2021

Contents

1 Introduction

(A remplir)

2 Notions d'Architecture nécessaires

(Repréciser les pipelines et les prédicteurs, faire des rappels d'Architecture ✓)
(Préciser la raison des if dans le pipeline (il faut attendre le résultat du test pour enchaîner) ✓)

Aujourd'hui, la plupart des processeurs sont pipelinés. Cela signifie que chaque instruction est séparée en plusieurs étapes et le processeur peut ainsi exécuter plusieurs instructions en parallèle, sans avoir nécessairement besoin d'attendre la fin de l'exécution de la précédente.
Le processeur peut donc recevoir différents types d'instructions et fera en sorte d'optimiser l'exécution. On s'intéressera principalement à l'exécution des instructions conditionnelles ("if"). Ces instructions conditionnelles permettent de pouvoir "sauter" à une certaine partie de code en fonction de la valeur de la condition lue dans l'instruction. Le saut effectué est une redirection de la lecture du code vers une ligne du code assembleur, précisée par l'instruction conditionnelle. Elles sont aussi particulières car elle demande au pipeline d'attendre la fin de leur exécution avant de pouvoir continuer. On peut donc facilement se retrouver avec un pipeline en attente à cause d'instructions conditionnelles trop longues à résoudre.
Pour essayer d'optimiser l'exécution de ces instructions, le processeur essaye de réaliser des prédictions pour savoir s'il doit prendre le saut ou non. Pour pouvoir faire ces prédictions, des prédicteurs de branchements ont été développés. Il existe de nombreux types de prédicteurs de branchements. Il existe par exemple les prédicteurs statiques ou bien encore les prédicteurs dynamiques.
Les prédicteurs statiques ne se servent pas des informations récoltées pendant l'exécution du code précédent. Il aura donc des prédictions pré-établies. Par exemple, répondre à tous les branchements qu'ils sont pris. De ce fait, leur exécution est rapide, mais leur taux d'erreur de prédiction est relativement élevé.
Le modèle de prédicteur dynamique a été établi pour essayer d'obtenir un taux d'erreur de prédiction plus restreint. Il se sert des évaluations des instructions précédentes du code pour prendre ses décisions et sera souvent plus précis. Il existe différents niveaux de prédicteurs dynamiques.
On va régulièrement parler de prédicteurs à k-bits, $k \in \mathbb{N}$. Ainsi, on pourra voir des modèles assez simple n'utilisant que la prédiction précédente pour établir la nouvelle et qui seront considérés comme des prédicteurs à 1-bit. Les modèles les plus répandus sont les prédicteurs à 2-bits, et plus précisément le compteur saturé qui permet de répondre très efficacement quand le cas étudié prend des conditions dont les réponses sont identiques pendant un certain nombre de prédictions. On peut aussi parler du prédicteur nommé le flip-on-consecutive

qui permet d'effectuer une transition de prédiction plus simplement lorsque les résultats des prédictions précédentes ne sont pas obligatoirement identiques entre elles lorsqu'elles se suivent. Vous pouvez observer un exemple de ces deux prédicteurs à 2-bits dans le schéma suivant.

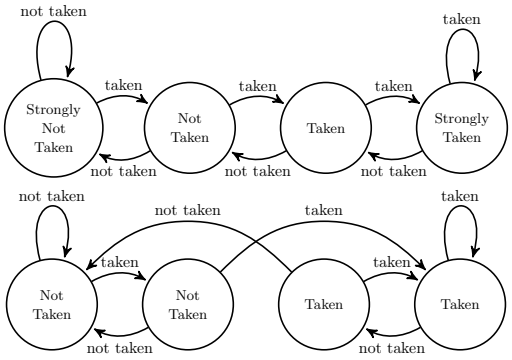


Figure 1: Deux prédicteurs à 2-bits différents (gauche : compteur saturé, droite: flip-on-consecutive).

Pour représenter ces différents prédicteurs dynamiques, on souhaite étudier et donc générer les graphes à k-états, $k \in \mathbb{N}$, fortement connexes et non-isomorphes entre eux.

3 Génération des graphes

Pour produire ces graphes, plusieurs méthodes ont été appliquées pendant l'avancée du projet. La première n'a pas été sélectionnée car elle était largement moins performante que la seconde. Vous pouvez donc continuer votre lecture après l'exemple.

(Repréciser pourquoi une des méthodes est moins bonne que l'autre et bien le préciser en premier lieu ✓)

3.1 Génération générale des graphes et algorithme de Tarjan

En premier lieu, tous les graphes à k-états, $k \in \mathbb{N}$, étaient générés à partir de combinaisons établies pour représenter les différentes destinations de chaque arête.
Ainsi, pour un graphe à k-états, on a $2 * k$ arêtes (Une arête "Taken" et une arête "Not Taken" par état).
On doit déterminer, pour chacune, l'état vers lequel elle se dirige.
Pour représenter cela, on générait des combinaisons de taille $2*k$ et dont chaque valeur pouvait appartenir entre 0 et $k-1$.
Ces combinaisons étaient ensuite permutées de toutes les façons possibles pour pouvoir générer tous les graphes à k-états.

Exemple 1 *Pour un graphe à 4-états, on pouvait par exemple avoir la combinaison suivante :*

$$Combinaison = \begin{pmatrix} 0 & 3 & 3 & 2 & 1 & 2 & 0 & 1 \end{pmatrix}$$

On peut donc le lire de la façon suivante :
$\forall n \in [0, k-1], k \in \mathbb{N}, Combinaison[2*n]$ *désigne l'état-destination de la branche "Not Taken".*
$\forall n \in [0, k-1], k \in \mathbb{N}, Combinaison[2*n+1]$ *désigne l'état-destination de la branche "Taken".*
Cette combinaison produira le graphe suivant :

(Ajouter un ref et mettre l'image dans un objet flottant ✓)

On pourra ensuite permuter la combinaison pour obtenir d'autres graphes.

Cette solution n'étant pas optimale en terme de performance, il a été décidé de générer toutes les combinaisons dont les graphes découlants seraient fortement connexes et n'ayant pas déjà été générés précédemment.
On transforme ensuite ces combinaisons en graphes, puis on vérifie grâce à l'algorithme de Tarjan que les graphes ne forment bien qu'une seule composante fortement connexe chacun.

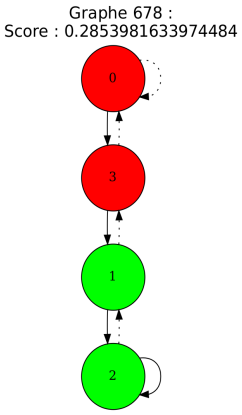


Figure 2: Premier graphe

(A modifier ✓)
L'algorithme de Tarjan étant un algorithme connu, il ne sera pas détaillé de nouveau ici. Vous pouvez cependant retrouver une description exhaustive de son but et de son implémentation ici : [1].
(A modifier ✓)

3.2 Parcours en profondeur et signature des graphes

(Donner un exemple pour le parcours en profondeur ✓)
(Définir les signatures pour les graphes ✓)
(Préciser les complexités)
(Faire un ensemble plus général, et en profiter pour repréciser l'exemple ✓)
(Préciser l'ancienne version et séparer toutes les méthodes dans des sous-parties)
On effectue par la suite un parcours en profondeur avec étiquetage des chemins, pour vérifier efficacement l'isomorphisme entre les graphes générés.
La signature d'un graphe est une étiquette représentant la disposition de ses états et de ses arêtes, qui aura une formulation bien précise afin de pouvoir identifier deux graphes qui auraient le même type de disposition des états, sans pour autant avoir exactement la même numérotation d'état.
Nous verrons dans l'exemple suivant comment déterminer les signatures d'un graphe.
Le principe du parcours en profondeur est donc le suivant.
Pour chaque graphe ayant été vérifié par l'algorithme de Tarjan :
- on établit une étiquette pour le graphe courant en partant du premier état (généralement l'état 0).
- on compare cette étiquette à celles précédemment enregistrées dans un ensemble, si elle n'est pas dans l'ensemble, on enregistre les k signatures existantes

pour le graphe (les autres signatures sont déterminées en commençant à partir des autres états). Si elle est dans l'ensemble, alors on passe à un autre graphe.

Exemple 2 *En prenant donc en considération le graphe représenté précédemment (2), on pourra facilement observer la génération de la signature du graphe. On va tout d'abord observer l'état 0.*

On créé plusieurs listes qui nous seront utiles pour la suite :
En premier lieu, on réalise une première exploration du graphe pour marquer l'ordre de rencontre du graphe.
Pour cela, on va tout d'abord marquer l'état comme visité, puis on va ajouter ce dernier dans une liste qui référencera les états dans leur ordre de découverte, que l'on nommera la liste "order". Par la suite, on continuera l'exploration dans les états voisins qui n'ont pas encore été visités. Quand tous les états sont identifiés comme visités, on arrête l'exploration.
En partant de l'état 0, on obtiendra donc la liste "order" suivante :

$$[0, 3, 1, 2]$$

On effectue un second parcours, en utilisant cette liste "order" pour établir l'étiquette.
Pour chaque état dans la liste "order", on va regarder ses voisins dans l'ordre suivant :
En premier lieu, on observera le voisin auquel on peut accéder par l'arête "Not Taken", puis celui lié par l'arête "Taken". On construit ensuite la signature, en ajoutant à chaque fois la position dans la liste "order" de l'état voisin que l'on observe actuellement.
Ici avec la liste précédemment établie, on obtiendrait la signature suivante :

"01021323"

Cette chaîne de caractères est donc l'une des quatres signatures possibles pour le graphe étudié.

Une fois que les graphes sont validés comme étant fortement connexes et non-isomorphes entre eux, on peut les considérer comme des graphes de chaînes de Markov.

(Rajouter exemple de comparaison entre 2 graphes ✓)

Exemple 3 *On peut exposer un second exemple de graphe :*

On peut donc établir toutes les signatures des graphes tels que :

$$Graphe1 : \text{"01021323"}, \text{"12100323"}, \text{"13202103"}, \text{"10203132"}$$
$$Graphe2 : \text{"01021323"}, \text{"12100323"}, \text{"13202103"}, \text{"10203132"}$$

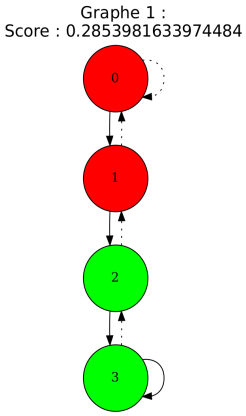


Figure 3: Second graphe

On peut donc voir que la signatures des graphes, il existe au moins une signature identique. Le second graphe est donc rejeté par l'algorithme et ne sera pas utilisé.

4 Chaînes de Markov

(Changer la lettre petit p dans $p_{i,j} en \Pi$ ✓)
(Rajouter la définition (pas dans une boîte) de ce qu'est une chaîne de Markov : (In simpler terms ...) ✓)
(Enlever la première définition)
(Faire une seule définition propre : parler du graphe de transition et de la matrice)
(Changer l'ordre des paragraphes, tout retravailler)

Après avoir généré tous les graphes, nous associons des probabilités aux arêtes pour les transformer en chaînes de Markov. Tout d'abord nous devons établir quelques définitions avant de continuer. On peut tout d'abord définir une chaîne de Markov comme étant un processus qui peut effectuer chaque prédiction en ne se basant que sur l'état courant. Ces prédictions sont aussi bonnes que des prédictions réalisées en connaissant tout l'historique. [4]

Définition 1 *Etant donné une chaîne de Markov défini sur l'ensemble d'états E, alors le nombre* $\mathbb{P}(X_1 = j \mid X_0 = i)$ *est appelé probabilité de transition de l'état i à l'état j en un pas, ou bien probabilité de transition de l'état i à l'état j, s'il n'y a pas d'ambiguïté.*
On note souvent ce nombre $\pi_{i,j}$ *:* $\pi_{i,j} = \mathbb{P}(X_1 = j \mid X_0 = i)$.
La famille de nombres $P = (\pi_{i,j})_{(i,j)\in E^2}$ *est appelée matrice de transition.*
Le graphe G d'une chaîne de Markov est un graphe orienté défini à partir de l'espace d'états E et de la matrice de transition $P = (\Pi_{i,j})$ *avec* $(i,j) \in E^2$ *de cette chaîne de Markov :*
- les sommets de G sont les éléments de E,
- les arêtes de G sont les couples $(i,j) \in E^2$ *vérifiant* $\Pi_{i,j} > 0$.
On peut donc dire que les matrices de transitions ont leurs lignes et leurs colonnes correspondantes dans l'ordre aux états représentés sur le graphe.
On peut définir la variable p, $p \in [0,1]$ *qui représente la probabilité réelle d'aller d'un état au suivant en empruntant la branche "Taken".* $\pi_{i,j}$ *pourra donc avoir comme valeur :*
- $\pi_{i,j} = p$*, pour les arêtes "Taken".*
- $\pi_{i,j} = 1 - p$*, pour les arêtes "Not Taken".*

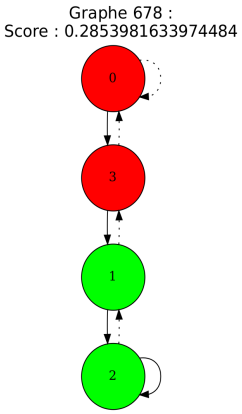
(Faire apparaître p (les probabilités) ✓)
(Compléter les définitions pour préciser les matrices spécifiques utilisées ✓)

(Changer l'ordre avec la proposition ✓)
On peut en profiter pour établir la proposition suivante qui nous sera utile lors de l'utilisation de la matrice de transition pour les calculs futurs :

Proposition 1 *La matrice de transition* $P = (p_{i,j})_{(i,j)\in E^2}$ *est stochastique : la somme des termes de n'importe quelle ligne de P donne toujours 1.*

On considérera donc que ces graphes de chaînes de Markov sont représentés par des matrices de transition à l'état initial 0.

En ayant établi ces matrices et les probabilités de transitions possibles, on pourra calculer les probabilités stationnaires π_i pour chaque état $Q \in E$, lié au graphe G.



Exemple 4 *Servons nous donc du graphe ci-dessus, pour donner un exemple de la matrice d'adjacence que l'on obtiendrait :*

$$P = \begin{pmatrix} 1-p & 0 & 0 & p \\ 0 & 0 & p & 1-p \\ 0 & 1-p & p & 0 \\ 1-p & p & 0 & 0 \end{pmatrix}$$

5 Calcul de probabilités stationnaires

Il existe deux façons de calculer des probabilités stationnaires.
On peut soit étudier la probabilité de se retrouver dans chaque état après chaque changement :
En considérant X, l'avancement sur la chaîne de Markov en fonction du temps (représenté par n, $n \in \mathbb{N}$), et k une valeur de temps précédente, soit $k \in [0, n]$:
(Quantifier les X, k et n ✓)
(Revoir la présentation pour la simplifier ✓)

$$X^{(n)} = X^{(n-k)} P^{(k)}$$

Exemple 5 *En reprenant l'exemple ci-dessus, on peut choisir X, telle que :*

$$X^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$$

On pourra donc calculer les probabilités d'être dans les prochains états à la prochaine prise de branche :

$$\begin{aligned} X^{(1)} &= X^{(0)} P = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1-p & 0 & 0 & p \\ 0 & 0 & p & 1-p \\ 0 & 1-p & p & 0 \\ 1-p & p & 0 & 0 \end{pmatrix} \\ \Leftrightarrow \quad X^{(1)} &= \begin{pmatrix} 1-p & 0 & 0 & p \end{pmatrix} \end{aligned}$$

Ou alors, on peut considérer avec la loi des grands nombres que l'on peut établir, tel qu'en considérant les variables X et n établies précédemment :

$$q = \lim_{n \to \infty} X^{(n)}$$

Considérons donc la matrice P (établie dans la section précédente), tel qu'il existe une matrice q qui peut vérifier l'égalité suivante :

$$qP = q$$

Introduisons la matrice identité I, qui nous permettra d'établir l'égalité suivante :

$$\begin{aligned} &\Leftrightarrow \quad qP = qI \\ &\Leftrightarrow \quad q(I - P) = 0 \end{aligned}$$

On peut aussi introduire la variable θ_i qui désignera les probabilités stationnaires de chaque état, i représentant le numéro des états présents dans le graphe.

Exemple 6 *En reprenant l'exemple ci-dessus, on peut établir la matrice q, telle que :*

$$q = \begin{pmatrix} \theta_0 & \theta_1 & \theta_2 & \theta_3 \end{pmatrix}$$

On aura donc :

$$\begin{aligned} q(I-P) &= \begin{pmatrix}\theta_0 & \theta_1 & \theta_2 & \theta_3\end{pmatrix}\left(\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}-\begin{bmatrix}1-p&0&0&p\\0&0&p&1-p\\0&1-p&p&0\\1-p&p&0&0\end{bmatrix}\right) \\ &\Leftrightarrow \begin{pmatrix}\theta_0 & \theta_1 & \theta_2 & \theta_3\end{pmatrix}\begin{pmatrix}p&0&0&-p\\0&1&-p&p-1\\0&p-1&1-p&0\\p-1&-p&0&1\end{pmatrix} \\ &\Leftrightarrow \begin{pmatrix}(-1+p)\theta_3+p\theta_0 & -p\theta_3+\theta_1+(-1+p)\theta_2 & -p\theta_1+(1-p)\theta_2 & \theta_3-p\theta_0+(-1+p)\theta_1\end{pmatrix} \end{aligned}$$

On obtiendra donc le système d'équation suivant :

$$\begin{cases}(-1+p)\theta_3+p\theta_0=0\\-p\theta_3+\theta_1+(-1+p)\theta_2=0\\-p\theta_1+(1-p)\theta_2=0\\\theta_3-p\theta_0+(-1+p)\theta_1=0\\\theta_0+\theta_1+\theta_2+\theta_3=1\end{cases}$$

Ce qui établira les prochaines égalités :

$$\begin{cases}\theta_0=\frac{1-3p+3p^2-p^3}{1-2p+2p^2}\\\theta_1=\frac{p^2-p^3}{1-2p+2p^2}\\\theta_2=\frac{p^3}{1-2p+2p^2}\\\theta_3=\frac{(-1+p)^2p}{1-2p+2p^2}\end{cases}$$

(Faire une transition et repréciser le but des calculs ✓)
On a donc obtenu des valeurs pour chaque état de leur probabilité stationnaire. Cela nous permet donc de savoir la probabilité que j'ai de me retrouver dans l'état i lors des n premiers pas dans la chaîne de Markov.
On pourra donc, en fonction de ces valeurs, facilement déduire si les états doivent être "Taken" ou "Not Taken".
Plus précisément, pour déterminer le statut des états, on appliquera les calculs d'intégrales suivants, pour tous les états π_i du graphe donné :

$$\begin{aligned} x &= \int_0^1 (\theta_i * (1-p))\, dp \\ y &= \int_0^1 (\theta_i * p)\, dp \end{aligned}$$

On choisira la valeur inférieure entre les deux intégrales calculées pour déterminer si l'état est "Taken" ou "Not Taken".

Exemple 7 *En reprenant les calculs effectués précédemment, on obtient :*
Les calculs d'intégrales de π_0 *:*

$$\begin{aligned} x &= \int_0^1 (\frac{1-3p+3p^2-p^3}{1-2p+2p^2} * (1-p))\, dp = 0.27400 \\ y &= \int_0^1 (\frac{1-3p+3p^2-p^3}{1-2p+2p^2} * p)\, dp = 0.08333 \end{aligned}$$

Comme $x \geq y$*, alors l'état est "Not Taken".*
Les calculs d'intégrales de π_1 *:*

$$\begin{aligned} x &= \int_0^1 (\frac{p^2-p^3}{1-2p+2p^2} * (1-p))\, dp = 0.05937 \\ y &= \int_0^1 (\frac{p^2-p^3}{1-2p+2p^2} * p)\, dp = 0.08333 \end{aligned}$$

Comme $x \leq y$*, alors l'état est "Taken".*
Les calculs d'intégrales de π_2 *:*

$$\begin{aligned} x &= \int_0^1 (\frac{p^3}{1-2p+2p^2} * (1-p))\, dp = 0.08333 \\ y &= \int_0^1 (\frac{p^3}{1-2p+2p^2} * p)\, dp = 0.27400 \end{aligned}$$

Comme $x \leq y$*, alors l'état est "Taken".*
Les calculs d'intégrales de π_3 *:*

$$\begin{aligned} x &= \int_0^1 (\frac{(-1+p)^2 p}{1-2p+2p^2} * (1-p))\, dp = 0.08333 \\ y &= \int_0^1 (\frac{(-1+p)^2 p}{1-2p+2p^2} * p)\, dp = 0.05937 \end{aligned}$$

Comme $x \geq y$*, alors l'état est "Not Taken".*

Pour calculer le "score" des graphes, on additionne juste les résultats obtenus.

Exemple 8 *On aura donc, pour ce graphe :*

$$score = 0.08333 + 0.05937 + 0.08333 + 0.05937 = 0.2854$$

References

[1] *Algorithme de Tarjan*. URL: https://fr.wikipedia.org/wiki/Algorithme_de_Tarjan. (Visité: 03.07.2021).

[2] *Chaîne de Markov*. URL: https://fr.wikipedia.org/wiki/Cha%C3%AEne_de_Markov. (Visité: 30.06.2021).

[3] *Graphe d'une chaîne de Markov*. URL: https://fr.wikipedia.org/wiki/Graphe_d%27une_cha%C3%AEne_de_Markov_et_classification_des_%C3%A9tats. (Visité: 30.06.2021).

[4] *Markov Chain*. URL: https://en.wikipedia.org/wiki/Markov_chain. (Visité: 03.07.2021).