

Projet Info LDD2 – TD 12

Renaud Vilmar -- vilmar@lsv.fr

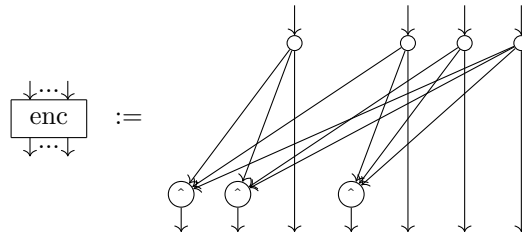
Objectifs du TD : Réécriture : vérification d'un code de Hamming.

Rendu final : Attendu pour le **9 mai**, à déposer sur eCampus. Peaufinez bien l'architecture du projet, les docstrings et les commentaires, et les quelques tests qui ont été demandés. N'hésitez pas à me poser des questions si vous avez besoin d'indications.

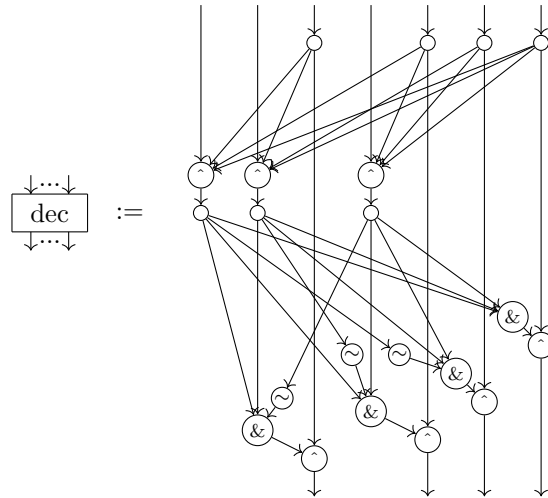
Lorsqu'on transmet de l'information sur un canal (fibre optique, wifi, signal radio, ...) du bruit peut intervenir, i.e. des bits d'information peuvent changer de valeur entre l'émetteur et le récepteur. Des codes correcteurs peuvent alors être utilisés pour pallier ce problème. L'idée conceptuellement est simple : au message qu'on veut transmettre, on va rajouter des bits d'information, qui vont servir à ajouter de la redondance.

Les codes de Hamming sont un des tous premiers exemples de codes correcteurs. Pour tout entier $n > 1$, il permet d'encoder un message de $2^n - n - 1$ bits en utilisant $2^n - 1$ bits, et peut corriger 1 erreur. On va s'intéresser dans la suite au cas $n = 3$ où on encode donc 4 bits avec 7.

Les blocs qui vont nous intéresser sont l'encodeur (qui est utilisé par l'émetteur pour ajouter les bits de redondance), et le décodeur (utilisé par le récepteur pour récupérer le message originel à partir de celui – potentiellement bruité – reçu). L'encodeur peut-être implémenté par le circuit booléen suivant :

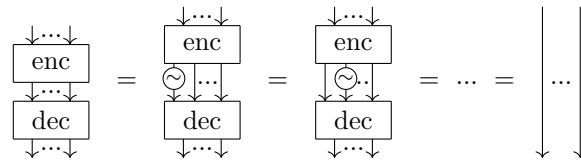


et le décodeur par le suivant :



▮ **Exercice 1 :** Donnez ces deux circuits comme des méthodes de classe de `bool_circ`. (On peut s'aider de la méthode `parse_parentheses`.) ▮

On va dans la suite vérifier la propriété principale du code : s'il y a au plus une erreur (un bit qui change de valeur) entre l'encodeur et le décodeur, on devrait retrouver exactement le message d'origine. Graphiquement, on veut montrer :



(en effet, une erreur sur un bit va être représentée par une porte NON).
On va le faire en réécrivant nos circuits, d'une manière similaire au TD précédent, mais avec les règles supplémentaires, que voici :

Associativité de XOR	Associativité de la copie	
“involution” de XOR	Effacement	
NON à travers XOR	NON à travers la copie	involution de NON

Notes : dans la règle d’effacement, “op” peut être n’importe quel opérateur, binaire ou unaire, ou même une constante. Dans l’“involution” de XOR, s’il y a plus que deux arêtes entre les deux noeuds, la règle nous dit qu’on peut les retirer deux par deux. On peut donc plus directement laisser une arête si on en avait un nombre impair, et 0 si on en avait un nombre pair.

▮ **Exercice 2 :** Implémenter ces règles de réécriture. Implémenter également une méthode qui applique ces règles (ainsi que celles du précédent TD) tant qu’on peut. ▮

▮ **Exercice 3 :** Au choix dans la worksheet ou dans le fichier de tests, vérifier la propriété principale du code de Hamming : que la composition de l’encodeur et du décodeur réduit à l’identité, et que cela reste vrai si une unique erreur survient sur n’importe quel bit. Enfin, montrez qu’avec 2 erreurs on peut ne pas retrouver le message d’origine. ▮

Bonus : Question ouverte

On peut aussi se servir de réécritures pour simplifier un circuit (par exemple en réduisant le nombre de portes).

▮ **Exercice 4 :** Proposer de telles réécritures, et les implémenter (en les distinguant bien des autres dans la docstring). Sur un grand ensemble de circuits générés aléatoirement, quel est le ratio moyen de portes supprimées par cette procédure ? ▮