**Compilation HipHop**

hiphop.js est un point d’entrée qui ne fait rien de compliqué.

On crée à partir de **machine.js** (ReactiveMachine) l’AST (ast.js, lang.js) en utilisant le fichier source (dans *setSignals* d’orchestrationHH.js avec le code ).

1. **orchestrationHH.js** -> setSignals -> machine = new hh.*ReactiveMachine*( orchestration, {sweep:true}); La référence à **hh** n’est qu’un lien vers l’export *ReactiveMachine* de **machine.js**.

*ReactiveMachine* est un objet (class). Le constructeur de *ReactiveMachine* a comme paramètre une variable *ast\_node* qui est le programme généré par Blockly. Ce programme doit être de type Module. Cet objet comprend toute une série de paramètre dont des map de signaux in et out, et une copie de l’AST. La compilation est lancée par le constructeur en ligne 123.

Puis on appelle le compilateur (**compiler.js**) qui fait appel à net.js

On compile une machine avec un AST.

Ce que génère Blockly est un bien l’AST dans la class ReactiveMachine. lang.js ne sert pas à grand-chose avec Blockly ?

Ceci crée la machine qui va gérer l’exécution (machine.js). L’exécution consiste à appeler la machine avec react() (Il peut y avoir une recompilation si l’AST a été modifié). La machine possède un clone de l’AST. Elle balaie les nets pour faire avancer la machine. Les nets constituent la machine.

Node => nœud de l’AST, va stocker le circuit et se connecter aux autres circuits après compilation. Comprend des nœuds action, expression, count expression et les éléments du langage, emit, sustain…

Le compilateur construit les circuits à partir de l’AST et les connecte, ça donne une machine.

Net => circuits composant une machine, il y des fan out et in?

Circuit => mise en œuvre en circuits logiques de HH.

Accessor ?

# Net.js (500 lignes)

Construction et propagation des nets HH.

Il y a des nets logiques, des nets registres et des nets action ( ?).

Class Net ----> LogicalNet

------> RegisterNet

------> ActionNet ----> TestExpressionNet

------> SignalExpressionNet

Hormis les classes il n’y a que deux fonctions : makeOr et makeAnd.

FAN IN et OUT => entrée sortie d’un net, composé de : net, polarity, dependency et antagonist.

Une classe Net dans un nœud AST. On push un net dans un nœud (node) AST.

connectTo(netExterieur, type) : connecte le net de la classe à un net extérieur. Les fanout.antagonist sont les fanin de la classe. Les fanin.antagonist sont les fanouts du net extérieur.

Un LogicalNet neutre est un AND si true un OR si false. ( ?!)

**Questions** :

C’est quoi les « control gates » ?

Dans net.js, que signifient STD, NEG, DEP (dependency) dans les FAN ?

C’est quoi la « polarity » d’un fanout ?

« Accessor » a quelque chose à voir avec les signaux ?