Travaux dirigés

Introduction à la Compilation

5-Analyseur contextuel

Jean-Christophe Le Lann

Mars 2023

Objectifs : Ce TP a deux objectifs. Il vise tout d'abord à réaliser un **analyseur contextuel** ou *sémantique* simplifié. Le sujet de départ vous permettra par ailleurs de vous assurer que vous êtes en mesure de développer votre propre compilateur complet.

Exercice

On s'intéresse à un langage (appelé *Synchrony*) permettant de décrire des circuits numériques élémentaires, à l'aide d'équations booléennes. Il ressemble à VHDL ou Verilog, mais reste beaucoup plus simple. On donne ici un exemple, qu'il s'agira d'extrapôler. [1]

Demi-additionneur : C'est le "Hello World" du numérique !

```
circuit half_adder
  input a,b
  output sum,carry

  sum   = a xor b
  carry = a and b
end
```

Question 1 : Ecrire l'ensemble des classes de l'AST modélisant le langage.

Question 2 : Décrire l'exemple précédent à l'aide de ces classes.

Le but du TP étant de réaliser un analyseur sémantique, **vous pouvez passer directement à la question 7**, ou choisir de réaliser le parser complet (question 2 à 4), ainsi que les visiteurs classiques (visiteur simple et pretty printer, question 5 et 6)).

Question 2 : Réaliser le lexer.

1. On pourra notamment réutiliser une partie de la grammaire des expressions rencontrées dans `Mini-C`.

Question 3 : Réaliser le parser, sans instancier l'AST.

Question 4 : Compléter le parser de manière à émettre un AST.

Question 5 : Ecrire un visiteur simple.

Question 6 : Ecrire un pretty printer.

Question 7 : Enoncer les règles de bon sens qui obligeraient à rejeter une description de circuits erronée.

Question 8 : Réaliser un visiteur appelé *Checker* qui vérifie ces règles.

Exercice 2 : Pour aller plus loin...

Question 1 : Compléter le compilateur de manière à permettre la réutilisation de composants. Par exemple, l'additionneur 1 bit complet réutilise deux demni-additionneurs :

```
require "ha"
circuit full_adder
  input a,b
  output sum,carry

  wire s1,c1,c2

  s1,c1 = ha(b,ci) # component instanciation
  s ,c2 = ha(a,s1) # that appears as function call
  carry = c1 or c2
end
```

Notez l'introduction de deux nouveaux mots clés :

- `require` qui permet de faire appel à des circuits décrits dans l'autres fichiers.
- `wire` qui permet de connecter des signaux *internes* et non plus seulement des entrées ou des sorties.

Question 2 : Réaliser un visiteur appelé *Drawer* qui génère un fichier *Graphviz* permettant de visualiser le circuit.

```
require "ha"

circuit Example
  input a,b,c,d,e,f
  output o1,o2,o3

  sig s1,s2,s3

  o1=(s1 and c) or (d and !d) and (e xor f)
```

```
  s1,s2=ha(a,b)
  o2=(a and b) or (c and d) or reg(s2)
  s3=reg(a and s3)
  o3=s3
end
```

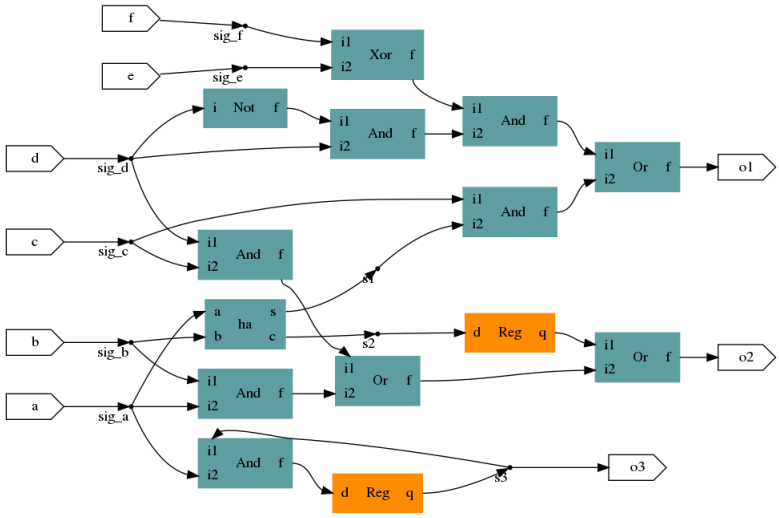

FIG. 1 – ***Netlist*** *du circuit décrit (on remarquera le rajout du mot clé* **`reg`** *décrivant une bascule D*