PERARD Lionel
VANBELLE Edouard
PARENTY Gaëtan
DUMORTIER Daniel

Rapport du mini-projet de PROLOG:

Diagnostic d'un circuit à base de logique combinatoire

Isen 4
Cours d'intelligence artificielle

Référent:

TAILLIBERT Patrick . . . . . Patrick.Taillibert@fr.thalesgroup.com

Auteurs:

PERARD Lionel . . . . . Lionel.Perard@isen.fr
VANBELLE Edouard . . . . . evanbelle@ifrance.com
PARENTY Gaëtan . . . . . Gaetan.Parenty@isen.fr
DUMORTIER Daniel . . . . . Daniel.Dumortier@isen.fr

Présentation

Afin de diagnostiquer des circuits simples de logique combinatoire, nous avons réalisé dans le cadre des cours d'intelligence artificielle, un programme en Prolog. Ce dernier est basé sur la méthode des diagnostics (avec génération de candidats).

Un format de fichier à été défini pour pouvoir décrire le circuit ainsi que ses observations.

Notre programme peut être utilisé de deux manières différentes. La première utilisation consiste à l'exécuter sous la console Prolog. Cependant l'utilisation reste difficile dans ce mode. Nous avons donc décidé d'ajouter un interface graphique pour alléger son fonctionnement.

Format de fichier

Nous avons décidé d'utiliser l'extension "`.cir`" comme circuit, mais cela reste arbitraire.
Les fichiers peuvent être édités dans un éditeur de texte. La syntaxe est proche de celle du Prolog, en effet, ce dernier compile le fichier pour que l'interpréteur puisse charger les caractéristiques du circuit.

Ainsi des commentaires peuvent être mis en plaçant % en début de ligne.
Pour décrire le circuit ajouter le mot clé: `cir` et y placer une liste de portes
Les portes sont définies par: gate [option: valeur, option: valeur, ...]
Il n'y a pas d'ordre pour les options, il faut simplement qu'elles soient séparées d'une virgule.
Chaque porte est séparée par une virgule.

Les options possibles sont:

- ✓ `name` (nom de la porte)
- ✓ `type` (type de circuit, choix possibles: `not, and, or, xor, nand, nor, xnor`)
- ✓ `in` (liste des entrées)
- ✓ `out` (liste des sorties) (pour l'instant, il ne peut y avoir qu'une seule sortie)

Pour décrire les observations, il suffit d'ajouter le mot clé `input`/`output` pour les entrées/sorties et d'insérer la liste des noeuds associés à leur valeur.

Exemple de fichier (`circuits/demo.cir`):

```
cir [
  gate [ name: et1, type: or, in: [a, b], out: [s_et1] ],
  gate [ name: comp, type: or, in: [c, s_or2], out: [y] ],
  gate [ name: et3, type: and, in: [a, b], out: [s_et3] ],
  gate [ name: et2, type: and, in: [a, b, c], out: [s_et2] ],
  gate [ name: or2, type: and, in: [s_et2, s_et3], out: [s_or2] ],
  gate [ name: or1, type: or, in: [s_et1, s_et2], out: [x] ]
].
input [
  a=1,
  b=1,
  c=1
].
output [
  x=0,
  y=0
].
```

Attention: Ne pas oublier de terminer les déclarations par des "."

Console

Pour utiliser le mode console, charger dans Prolog le fichier `console.pl` (ou lancer le fichier `console.bat` si votre Prolog est dans le même répertoire que celui effectué lors de nos tests).

Une fois dans la console, deux options sont disponibles: Vous pouvez lancer le diagnostic soit en utilisant tous les nogoods, soit en n'utilisant que le nogoods minimaux.

Remarque: les nogoods minimaux sont déterminés à partir de tous le nogoods, donc le gain de temps n'est pas optimal. L'idéal eu été de déterminer directement les nogoods minimaux, chose que nous n'avons pas faite, faute de temps.

L'utilisation est la suivante:

`diag('circuits/demo.cir').` (avec les nogoods minimaux)

`diag_all('circuit/demo.cir').` (avec tous les nogoods)

Exemple d'utilisation:

```
chargement terminé, pour lancer un diagnostic, utilisez:
"diag(nom_de_fichier.cir)." (utilise les nogoods minimaux)
"diag_all(nom_de_fichier.cir)." (utilise tous les nogoods)

{c:/mes documents/edouard/cours/tp/n4/ia/diagnostic/console.pl compiled, 110 msec 163052
bytes}
| ?- diag('circuits/demo.cir').
{compiling c:/mes documents/edouard/cours/tp/n4/ia/diagnostic/circuits/demo.cir...}
{c:/mes documents/edouard/cours/tp/n4/ia/diagnostic/circuits/demo.cir compiled, 60 msec
1888 bytes}
Lecture du circuit...

Circuit remodelé:
[[et1,or,[a,b],[s_et1]],[comp,or,[c,s_or2],[y]],[et3,and,[a,b],[s_et3]],[et2,and,[a,b,c]
,[s_et2]],[or2,and,[s_et2,s_et3],[s_or2]],[or1,or,[s_et1,s_et2],[x]]]

Observations: [a=1,b=1,s_et1=_1321,c=1,s_or2=_1335,y=0,s_et3=_1349,s_et2=_1354,x=0]

Equations:
clpb:sat(_1080=<(1+1=:=_1321)),clpb:sat(_1085=<(1+_1335=:=0)),clpb:sat(_1090=<(1*1=:=_13
49)),clpb:sat(_1095=<(1*1*1=:=_1354)),clpb:sat(_1100=<(_1354*_1349=:=_1335)),clpb:sat(_1
105=<(_1321+_1354=:=0))

Mode minimal:

   Liste des NOGOODS minimaux:
[[et2,or1],[et2,or2,or1],[et3,et2,or1],[et3,et2,or2,or1],[comp],[et1,or1],[et1,or2,or1],
[et1,comp]]

   Liste des CANDIDATS: [[comp,et1,et2],[comp,or1]]

yes
| ?-
```

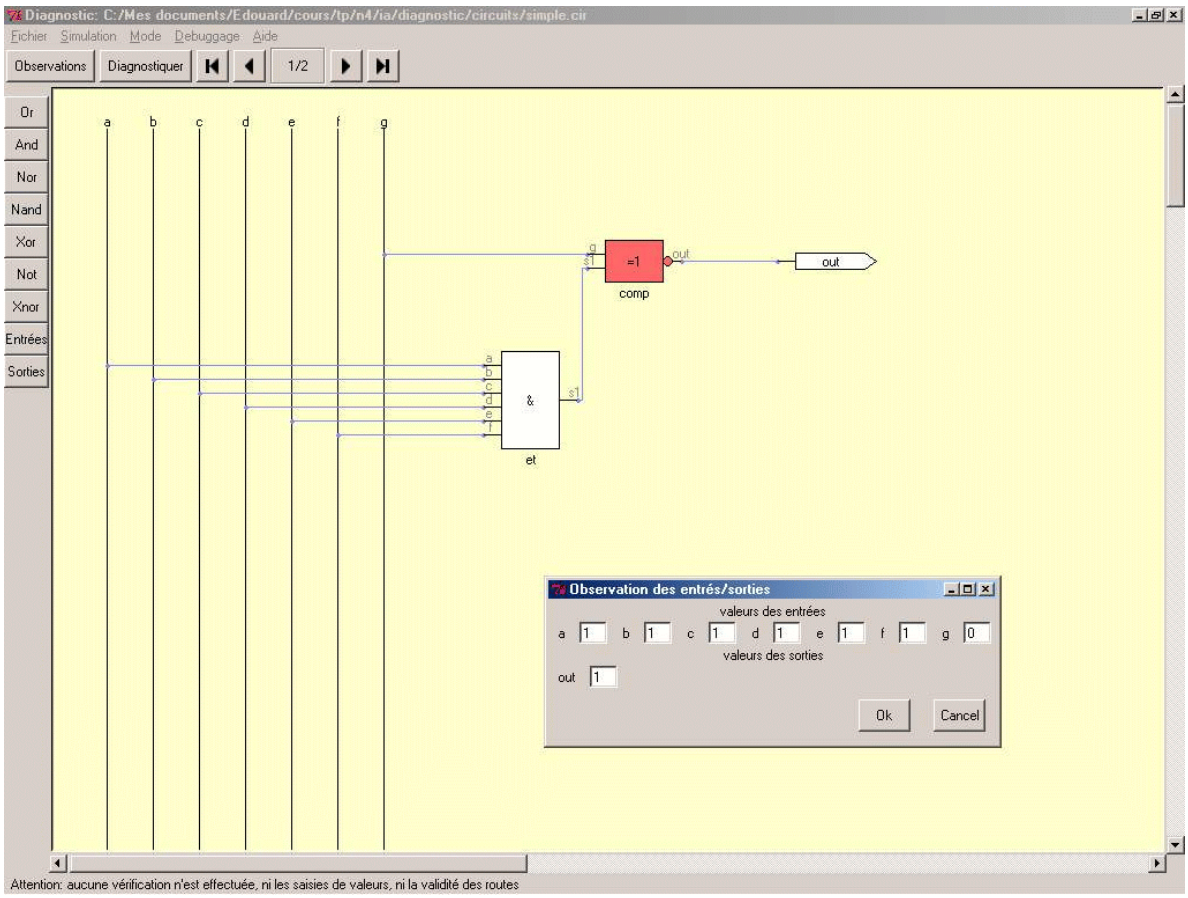
Le code tourne sous la version 3.8 de l'Isen et 3.9 (version limitée à 30 jours)

Interface Graphique

Pour lancer la console graphique, il suffit d'exécuter `graphique.bat`. S'il ne fonctionne pas, exécutez `graphique.pl` depuis Prolog.

Grâce au mode graphique, vous pouvez non seulement tester les circuits mais aussi les créer. Ainsi l'édition en est facilité.

Voici une capture d'écran du même exemple: `circuits/demo.cir`



Vous pouvez charger (dans ce cas, Prolog transmet à l'interface une estimation du positionnement des portes), sauvegarder un circuit (les sauvegardes ne mémorisent pas la position des portes).

Pour créer un circuit, utilisez les boutons de gauche. Si vous avez fait une erreur, vous pouvez supprimer ou corriger (encore en développement) un objet par un simple clic droit (ou double-clic gauche). Pour déplacer le plan de travail, utilisez les barres de navigations ou un simple Contrôle+clic gauche.

La barre de haut permet de visualiser les diagnostics. Le bouton “observations” permet de définir les observation. Pour diagnostiquer cliquez sur “diagnostiquer”. Dès lors, Prolog prend la main et effectue les calculs. Il renvoie les résultats à l’interface tcl/tk qui colorie en rouge les portes défectueuses et en vert les portes certifiées correctes. Pour naviguer parmi les différentes solutions, utilisez les flèches de la barre du haut.

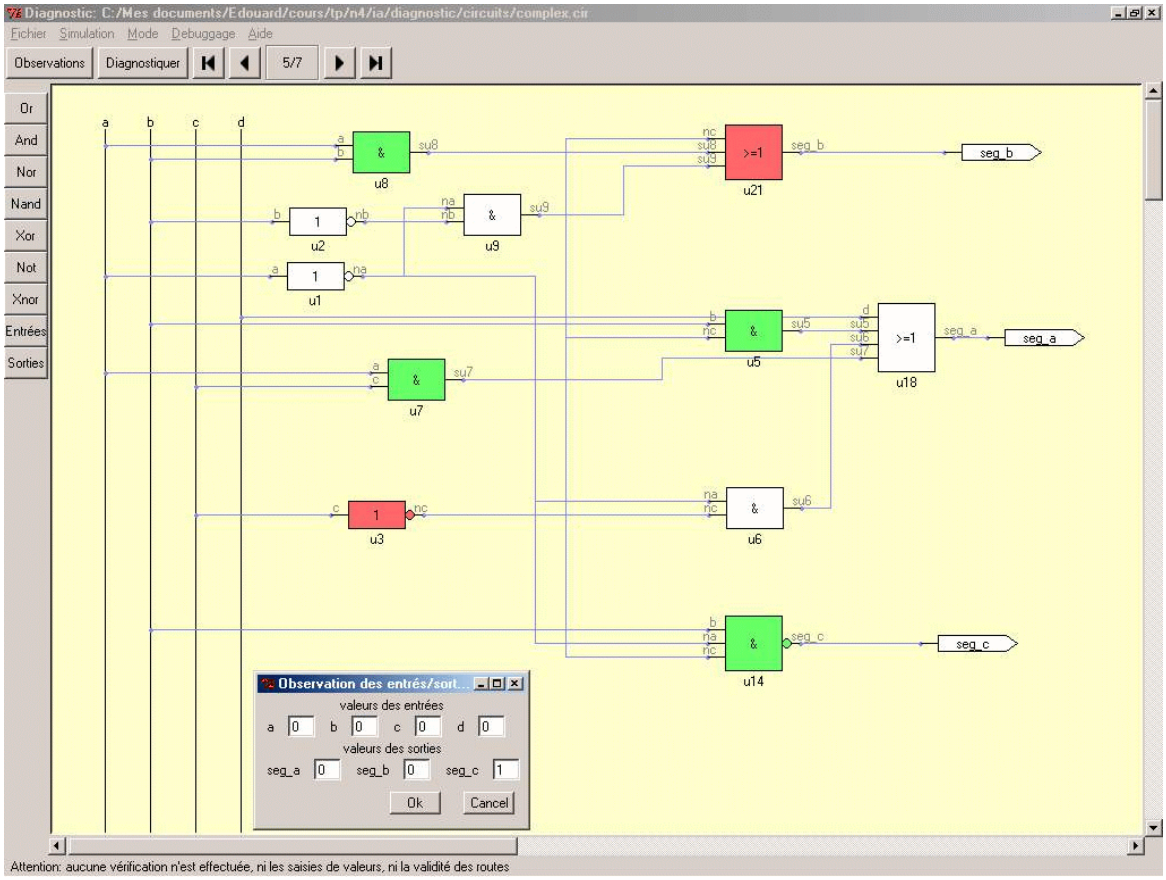
Le menu “simulation” reprend les boutons de la barre du haut. Cependant, vous y trouverez “diagnostics multiples...” qui permet d’effectuer une batterie de diagnostics en fonction de différentes observations.

Alors, l’interface mémorise les portes non défectueuses et les affiche en vert. Pour garantir l’efficacité de cette option, il faut utiliser un cas réel: en effet si vous utilisez des valeurs aléatoires pour les diagnostics, il risque d’y avoir des conflits. Si une porte est valide lors d’un diagnostic et se retrouve candidate lors d’un autre diagnostic, il y a incohérence: cela veut dire que la porte est à la fois défectueuse et à la fois correcte. Ce qui n’est pas possible. Il faut donc veiller à utiliser les observations réelles.

Ce dernier mode, mémorise les portes qui ne sont pas candidates et les cumule à chaque diagnostic. Ainsi on travaille par élimination. A la fin, les portes restantes en blanc sont celles qui peuvent être défectueuses. Donc si le nombre d’observation est important, il est possible que le champ de recherche se restreigne.

Le menu “mode” permet de choisir si Prolog travaillera avec tous les nogoods ou seulement les nogoods minimaux. Vous pouvez aussi effacer les routes si vous juger que cela encombre le plan de travail.

Le mode graphique a été testé avec Prolog 3.9 et Tcl/Tk 8.3.



Autre exemple: Dans ce cas 7 cas d'erreurs sont possible (on affiche le cas 5). On remarque que 4 portes sont correctes.

Remarque:

- ✓ à chaque diagnostic, l'interface enregistre le circuit dans le répertoire du programme sous le nom "`current.cir`" pour le transférer à Prolog. Donc si vous quittez, vous pouvez toujours récupérer le dernier schéma contrôlé.
- ✓ le décodeur 7 segments, est long à tester, il faut plus d'une heure sur un Pentium 850 Mhz. L'idéal est de séparer chaque sortie pour accélérer les calculs (car cela diminue considérablement le nombre de nogoods).