

# Deffects - een heuristisch expertsysteem voor het debuggen van elektronische schakelingen

Solvi Arnold - 0340561  
Jan-Olav Hajduk - 0342602  
Matthijs Melissen - 0423165  
Erik Schager - 0477559

7 juli 2006

## 1 Inleiding

Dit verslag hoort bij Deffects, een expertsysteem geschreven in CLIPS om problemen in elektronische schakelingen op te lossen. De naam is een samentrekking van deffects, een construct in CLIPS om data te definiëren, en het woord defect.

## 2 Programmaverloop

Ons programma maakt gebruik van een feitendatabase die gegeven een bepaald type poort, invoerwaarden en uitvoerwaarden, per mogelijke storing bijhoudt met welke a priori zekerheid (uitgedrukt als certainty factor, een waarde tussen -1 en 1) deze storing het geval is. De gekozen waarden in ons programma zijn slechts bedoeld als voorbeeld en op geen enkele wijze in de praktijk getest. Wanneer dit programma in de praktijk gebruikt gaat worden, zullen deze waarden experimenteel vastgesteld moeten worden.

Daarnaast houdt het systeem per poort per mogelijke storing een certainty factor bij, die aangeeft hoe waarschijnlijk deze storing is. Initieel zijn deze certainty factors 0.

Ons programma bestaat uit de volgende - door middel van een goal-variabele gerealiseerde - fasen:

- askCircuit
- linkCircuit
- initialize
- findGate
- selectInputs
- applyTest
- updateCF
- finishTest

## 2.1 askCircuit

In deze fase vraagt het systeem de gebruiker zijn schakeling te specificeren, door poort voor poort het type en de inputs van elke interne poort op te geven, of voor de schakeling uit de opgave te kiezen. Opgegeven poorten worden met behulp van het template "gate" vastgelegd in de database (zie ook: "Eigen invoer" onder "Uitbreidingen").

## 2.2 linkCircuit

Wanneer alle interne poorten bekend zijn, worden input-poorten automatisch toegevoegd waar nodig, en wordt aan de gebruiker gevraagd waar de output-poorten moeten komen. Deze externe poorten worden ook opgeslagen in het "gate"-deftemplate. Tevens worden enkele checks uitgevoerd om fouten bij het opgeven tijdig te ondervangen (zie ook "Eigen invoer" onder "Uitbreidingen").

## 2.3 initialize

In deze stap stellen we voor elke poort de certainty factor voor elke storing op 0.

## 2.4 findGate

De globale opzet is als volgt: we werken van inputs naar outputs het hele systeem door. We beginnen dus met de poorten die rechtstreeks op de invoer aangesloten zijn. Vervolgens testen we deze net zo lang tot we met grote zekerheid weten dat ze correct zijn, en nemen aan dat deze poorten geen storingen meer veroorzaken. De poorten die getest kunnen worden zijn alle poorten waarvan de poorten aan de ingangen ook al getest zijn of van type input zijn.

Soms zijn er meerdere poorten mogelijk, en op elke poort zijn meerdere inputs mogelijk. Met behulp van de certainty factors kiezen we de test waarvan we verwachten dat we het meeste aan de uitkomsten hebben. Dat doen we door voor alle mogelijk invoercombinaties op deze poorten de prioriteit  $P(g, i)$  te berekenen, die we als volgt hebben gedefinieerd:

$$P(g, i) = \sum_{s \in S} \sum_{o \in O} ((h(g, s) + 1) * |e(g, s, i, o)|) \quad (1)$$

Hierin is  $h(g, s)$  de huidige zekerheid dat poort  $g$  de storing  $s$  heeft, en  $e(g, s, i, o)$  de verwachte a priori zekerheid dat poort  $g$  de storing  $s$  heeft als we met invoer  $i$  de uitvoer  $o$  meten.

De factor  $|e(g, s, i, o)|$  zorgt ervoor dat we tests die óf heel hoge óf heel lage certainty factors opleveren, dat wil zeggen tests die een grote verandering in onze huidige overtuiging teweeg kunnen brengen, hogere prioriteit geven dan andere tests.

De factor  $h(g, s) + 1$  geeft aan dat we vooral geïnteresseerd zijn in grote veranderingen van certainty factors in de storingen waarvan we denken dat ze waarschijnlijk het geval zijn. De +1 is nodig om te voorkomen dat in het geval  $h(g, s) = 0$  de a priori zekerheid geen invloed heeft.

Vervolgens kiezen we de combinatie van inputs en poort die de prioriteit maximaliseert. Wel wordt er voor gezorgd dat er niet vaker dan één keer dezelfde test op een bepaalde poort

gedaan kan worden zolang nog niet alle mogelijke tests op die poort gedaan zijn. Omdat we uitgaan van een meetonzekerheid, kan het desondanks toch nodig zijn om dezelfde test meerdere keren uit te voeren.

## 2.5 selectInputs

We hebben tot nu toe alleen gekeken welke invoer we rechstreeks op de te testen poort plaatsen. Om de tester moeite te besparen, gaan we nu kijken welke invoer we op de inputpoorten moeten zetten om de gewenste invoer op de te testen poort te krijgen. Die invoer op de inputpoorten bepalen we door vanuit de benodigde inputs op een interne poort paden naar de inputpoorten te traceren. Uit de mogelijke invoer configuraties wordt er vervolgens eentje geselecteerd (op basis van eenvoud), en toegevoegd aan de voorgenomen test.

Een stukje CLIPS-uitvoer uit deze fase als men feiten en vuren van regels watcht; hier wordt gezocht naar een invoerconfiguratie die ertoe zal leiden dat zowel poort 5 als poort 7 een 0 als output geven:

```
FIRE 1100 traceBack: f-1335,f-1337,f-675
<== f-1337 (traceInput (gateNrs 5 7) (values 0 0))
==> f-1339 (traceInput (gateNrs 2 7) (values 0 0))
==> f-1340 (traceInput (gateNrs 3 7) (values 0 0))
FIRE 1101 traceBack: f-1335,f-1340,f-1235
<== f-1340 (traceInput (gateNrs 3 7) (values 0 0))
==> f-1341 (traceInput (gateNrs 3 1) (values 0 0))
==> f-1342 (traceInput (gateNrs 3 4) (values 0 0))
FIRE 1102 traceBack: f-1335,f-1342,f-789
<== f-1342 (traceInput (gateNrs 3 4) (values 0 0))
==> f-1343 (traceInput (gateNrs 3 2 3) (values 0 0 0))
==> f-1344 (traceInput (gateNrs 3 2 3) (values 0 1 1))
FIRE 1103 traceBack: f-1335,f-1339,f-1235
<== f-1339 (traceInput (gateNrs 2 7) (values 0 0))
==> f-1345 (traceInput (gateNrs 2 1) (values 0 0))
==> f-1346 (traceInput (gateNrs 2 4) (values 0 0))
FIRE 1104 traceBack: f-1335,f-1346,f-789
<== f-1346 (traceInput (gateNrs 2 4) (values 0 0))
==> f-1347 (traceInput (gateNrs 2 2 3) (values 0 0 0))
==> f-1348 (traceInput (gateNrs 2 2 3) (values 0 1 1))
FIRE 1105 pruneTraces: f-1335,f-1348
<== f-1348 (traceInput (gateNrs 2 2 3) (values 0 1 1))
FIRE 1106 pruneTraces: f-1335,f-1347
<== f-1347 (traceInput (gateNrs 2 2 3) (values 0 0 0))
==> f-1349 (traceInput (gateNrs 2 3) (values 0 0))
FIRE 1107 pruneTraces: f-1335,f-1344
<== f-1344 (traceInput (gateNrs 3 2 3) (values 0 1 1))
FIRE 1108 pruneTraces: f-1335,f-1343
<== f-1343 (traceInput (gateNrs 3 2 3) (values 0 0 0))
==> f-1350 (traceInput (gateNrs 3 2) (values 0 0))
FIRE 1109 singleOut: f-1335,f-1350,f-1349
```

```
<== f-1349 (traceInput (gateNrs 2 3) (values 0 0))
FIRE 1110 singleOut: f-1335,f-1350,f-1345
<== f-1345 (traceInput (gateNrs 2 1) (values 0 0))
FIRE 1111 singleOut: f-1335,f-1350,f-1341
<== f-1341 (traceInput (gateNrs 3 1) (values 0 0))
resterend traceInput feit:
f-1350 (traceInput (gateNrs 3 2) (values 0 0))
```

Dus door een 0 op zowel inputpoort 2 als 3 te zetten zal de output van poorten 5 en 7 ook 0 zijn. Wat er op poort 1 staat is niet van belang, dus dat wordt ook niet voorgeschreven (zie ook "Eigen invoer" onder "Uitbreidingen" voor extra toelichting).

## 2.6 applyTest

Er is nu een test die de gebruiker kan uitvoeren. Het programma zegt op welk invoerkanaal welke invoerwaarde moet worden gezet. De invoerkanalen worden hier op volgorde genoemd. Vervolgens wordt de gebruiker gevraagd de uitvoerwaarde bij de poort (die getest wordt) te geven. Na een geldig antwoord, kan het programma naar de volgende fase gaan.

## 2.7 updateCF

Als een nieuwe meting is gedaan, worden per storing de nieuwe certainty factors gelijk gemaakt aan de uitkomst van de co-conclusieregel [1] als deze wordt toegepast op de betreffende a priori-zekerheid en de oude certainty factor. Merk op dat de certainty factors initiëel 0 zijn. Omdat de co-conclusie-regel voor certainty factors associatief is, is dit equivalent aan de CF's van alle metingen bijhouden en deze achteraf pas samen nemen. Onze methode is echter geheugenefficiënter.

## 2.8 finishTest

Zodra 'correct' met een CF groter dan 0.9 geconcludeerd kan worden, meldt het programma dat dit onderdeel correct is, en gaat hier in het vervolg vanuit. Als een storing met een CF groter dan 0.9 geconcludeerd wordt, wordt de gebruiker gevraagd dit component te vervangen en wordt het component opnieuw getest. Bij het vervangen van een component kan de gebruiker immers ook een storing in het nieuwe component veroorzaken.

## 3 Taakverdeling

De module voor het invoeren van eigen schakelingen is door Solvi geschreven. Ook heeft zij het algoritme dat de benodigde invoer op de input-poorten vindt gegeven de invoer op de interne poort. Matthijs heeft de fase geschreven die bepaalt welke test als eerste uitgevoerd moet worden. Jan-Olav heeft zich bezig gehouden met de meldingen van correcte en incorrecte poorten, en het gedeelte dat tests uitvoert en uitkomsten die de gebruiker opgeeft verwerkt is door Erik geschreven.

#### 4 Voorbeeld van interactie

\* Put the value 0 on the input channel with number 2.  
\* Put the value 1 on the input channel with number 3.  
\* Give the output at gate number 4 of type 'xor'.

1

\* Put the value 0 on the input channel with number 2.  
\* Put the value 0 on the input channel with number 3.  
\* Give the output at gate number 5 of type 'and'.

0

Gate number 5 of type 'and' is likely to be correct.

\* Put the value 0 on the input channel with number 2.  
\* Put the value 0 on the input channel with number 3.  
\* Give the output at gate number 4 of type 'xor'.

0

Gate number 4 of type 'xor' is likely to be correct.

\* Put the value 1 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.  
\* Put the value 0 on the input channel with number 3.  
\* Give the output at gate number 6 of type 'xor'.

0

\* Put the value 1 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.  
\* Put the value 0 on the input channel with number 3.  
\* Give the output at gate number 7 of type 'and'.

1

\* Put the value 1 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.  
\* Put the value 1 on the input channel with number 3.  
\* Give the output at gate number 7 of type 'and'.

0

\* Put the value 0 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.  
\* Put the value 0 on the input channel with number 3.  
\* Give the output at gate number 7 of type 'and'.

0

\* Put the value 1 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.

\* Put the value 1 on the input channel with number 3.  
\* Give the output at gate number 6 of type 'xor'.

1

\* Put the value 0 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.  
\* Put the value 0 on the input channel with number 3.  
\* Give the output at gate number 6 of type 'xor'.

1

\* Put the value 0 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.  
\* Put the value 1 on the input channel with number 3.  
\* Give the output at gate number 7 of type 'and'.

0

Gate number 7 of type 'and' is likely to be correct.

\* Put the value 1 on the input channel with number 2.  
\* Put the value 1 on the input channel with number 3.  
\* Give the output at gate number 8 of type 'or'.

1

\* Put the value 1 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.  
\* Put the value 0 on the input channel with number 3.  
\* Give the output at gate number 8 of type 'or'.

1

\* Put the value 0 on the input channel with number 2.  
\* Put the value 0 on the input channel with number 3.  
\* Give the output at gate number 8 of type 'or'.

0

\* Put the value 1 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.  
\* Put the value 0 on the input channel with number 3.  
\* Give the output at gate number 8 of type 'or'.

1

Gate number 8 of type 'or' is likely to be correct.

\* Put the value 0 on the input channel with number 1.  
\* Put the value 1 on the input channel with number 2.  
\* Put the value 1 on the input channel with number 3.  
\* Give the output at gate number 6 of type 'xor'.

0

Gate number 6 of type 'xor' is likely to be correct.  
\* All problems are fixed by now.

## 5 Gedane uitbreidingen

We hebben de minimale oplossing uitgebreid met een paar extra dingen.

### 5.1 Meetonnauwkeurigheid

We gaan ervan uit dat metingen nooit 100% nauwkeurig zijn. Daarom vragen we de gebruiker soms om een meting opnieuw te doen. Er zal echter eerst geprobeerd worden de fout te vinden door metingen aan andere poorten te doen.

### 5.2 Eigen invoer

Gebruikers kunnen zelf makkelijk eigen netwerken invoeren. De invoermodule legt de gebruiker uit hoe hij zijn schakeling moet nummeren en kan hem vervolgens zelf bouwen op basis van enkel de typen en invoerpoorten van de interne poorten, plus een lijstje van poortnummers waar outputpoorten aan moeten. Inputpoorten worden volledig automatisch toegevoegd. Dit houdt het invoeren overzichtelijk en het typewerk minimaal. Ook biedt de module de mogelijkheid om met het keyword "default" bij de eerste prompt direct het voorbeeldnetwerk uit de opgave in de database te zetten. Ook houdt deze module in de gaten of er geen gaten in de nummering van de user zitten. Na succesvolle invoer wordt een overzicht van het opgegeven netwerk geprint zodat de gebruiker kan checken of het overeenkomt met zijn schakeling. Loops zijn in principe niet supported, omdat niet geheel duidelijk was in hoeverre het diagnostiseren van defecten daarmee overweg kan. Kleine loops (n of twee poorten) in de opgegeven schakeling worden door de invoermodule gedetecteerd en als error gerapporteerd. Het systeem volledig compatibel maken met schakelingen waar loops in voorkomen zou een leuke uitbreidingsmogelijkheid zijn.

### 5.3 Enkel invoer op input-poorten

Om het meten eenvoudig te houden voor de gebruiker, vraagt het systeem nooit om invoer op interne poorten. Dit hebben we bereikt door inputs die we op een interne poort nodig hebben, terug te traceren naar de input poorten. Hiervoor gebruiken we lijstjes van poorten en invoerwaarden, die telkens herschreven worden tot ze enkel inputpoorten bevatten. Herschrijven wil hier zeggen: het vervangen van een interne poort in de lijst door een of twee van zijn directe invoerpoorten, en het vervangen van de bijbehorende invoerwaarde door de benodigde uitvoerwaarden van zijn directe invoerpoorten. Zijn er meerdere mogelijkheden, dan splitst de search zich.

Een klein voorbeeldje (uitgaand van de default schakeling) om het process conceptueel te illustreren:

we willen waarden 0 en 1 op poort 7 zetten. Poort 7 heeft Poorten 1 en 4 als invoer. Dan maken we het volgende lijstje:

((1 4) (0 1))

Poort 4 is een interne poort, dus die herschrijven we. Poort 4 heeft poorten 2 en 3 als invoer. Aangezien het een xor-poort is en we er een 1 uit willen krijgen, moeten we hem (0 1) of (1 0) voeren. de search splitst dus:

((1 2 3) (0 0 1)) & ((1 2 3) (0 1 0))

Poorten 1, 2 & 3 zijn allen inputpoorten, dus nu zijn we klaar met zoeken.

Wanneer de search splitst en later zichzelf kruist, kan n en dezelfde poort meerdere keren in de lijst voorkomen. In zulke gevallen wordt gecheckt of er geen strijdige waarden in staan (je kan immers niet 0 op input 2 en 1 op input 2 zetten). Strijdige exemplaren worden gelimineerd, en bij niet-strijdige exemplaren wordt de redundante informatie verwijderd. Als er dan meerdere uitkomsten overblijven, wordt n van de eenvoudigsten gekozen.

#### 5.4 Vervangen van defecte poort

Als een gate zeer waarschijnlijk defect is, dan wordt de gebruiker verzocht deze te vervangen. Vervolgens zal de vervanging ook getest worden, want we gaan er niet zomaar van uit dat deze correct is. Daarna kan het programma doorgaan met het testen van de verdere poorten.

## 6 Andere mogelijke uitbreidingen

Het kan handig zijn om bepaalde componenten door de gebruiker correct te laten verklaren (als die b.v. nieuw zijn). De CF van correct moet bij die poorten dan 1 worden. Ook zouden bepaalde poorten verdacht kunnen worden gemaakt door de CFs van defecten te vergroten. De drempelwaarde van  $CF > 0.9$  zou instelbaar gemaakt kunnen worden, het hoeft niet per se die door ons bedachte waarde te zijn.

Het is op zich ook mogelijk om invoer direct op intern gelegen poorten te zetten. Dat is echter minder wenselijk. Toch zou het nuttig zijn om bepaalde (door de gebruiker als verdacht aangemerkte) poorten meteen al te kunnen testen, zonder eerst alle ervoor te moeten doen.

De a priori CFs zouden wat beter onderbouwd kunnen worden, bijvoorbeeld door gegevens over (de reparatie van) defecte circuits te verzamelen en te analyseren. Dit om erachter te komen welk soort defect het waarschijnlijkst is bij een bepaalde gate-invoer-uitvoer combinatie.

## Referenties

- [1] E. H. Shortlife and B. G. Buchanan. A model of inexact reasoning in medicine. *Math. Biosci.*, 23:351-379, 1975.