Project Theoretische Informatica 2016-2017: NFA Adventures

Groepswerk - 2 personen

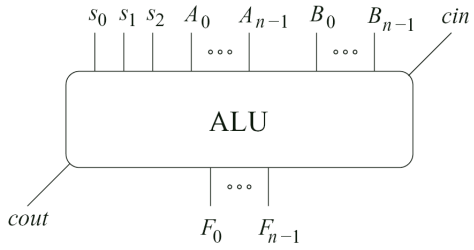
1 Motivatie: Automatische verificatie

In de herfst van 1994 introduceert Intel zijn Pentium microprocessor. Twee maanden later echter ontdekt de wiskundige Thomas R. Nicely per toeval een bug:

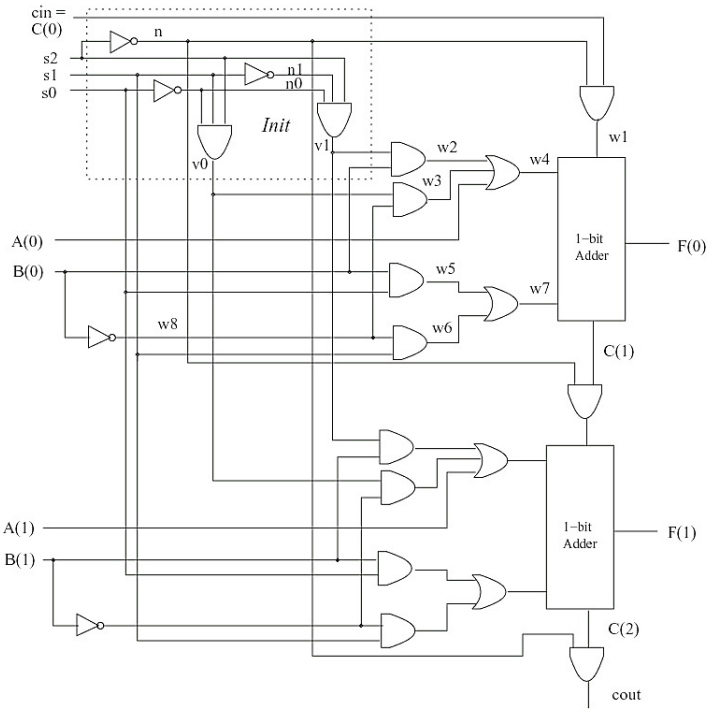
```
FROM:  Dr. Thomas R. Nicely
       Professor of Mathematics
       Lynchburg College
TO:    Whom it may concern
RE:    Bug in the Pentium FPU
DATE:  30 October 1994

It appears that there is a bug in the floating point unit
(numeric coprocessor) of many, and perhaps all, Pentium processors.

In short, the Pentium FPU is returning erroneous values for
certain division operations. For example,
     0001/824633702441.0
is calculated incorrectly (all digits beyond the eighth
significant digit are in error). This can be verified in
compiled code, an ordinary spreadsheet such as Quattro Pro
or Excel, or even the Windows calculator (use the scientific
mode), by computing
     00(824633702441.0)*(1/824633702441.0),
which should equal 1 exactly (within some extremely small
rounding error; in general, coprocessor results should contain
19 significant decimal digits).  However, the Pentiums
tested return
     0000.999999996274709702
for this calculation.  A similar erroneous value is obtained
for x*(1/x) for most values of x in the interval
     00824633702418 <= x <= 824633702449,
and throughout any interval obtained by multiplying or
dividing the above interval by an integer power of 2 (there
are yet other intervals which also produce division errors).

The bug can also be observed by calculating 1/(1/x) for the
above values of x.  The Pentium FPU will fail to return the
original x (in fact, it will often return a value exactly
3072 = 6*0x200 larger).
```

```
The bug has been observed on all Pentiums I have tested or
had tested to date, including a Dell P90, a Gateway P90, a
Micron P60, an Insight P60, and a Packard-Bell P60.  It has
not been observed on any 486 or earlier system, even those
with a PCI bus.  If the FPU is locked out (not always
possible), the error disappears; but then the Pentium becomes
a "586SX", and floating point must run in emulation, slowing
down computations by a factor of roughly ten.

I encountered erroneous results which were related to this
bug as long ago as June, 1994, but it was not until 19 October
1994 that I felt I had eliminated all other likely sources of
error (software logic, compiler, chipset, etc.).  I contacted
Intel Tech Support regarding this bug on Monday 24 October
(call reference number 51270).  The contact person later
reported that the bug was observed on a 66-MHz system at
Intel, but had no further information or explanation, other
than the fact that no such bug  had been previously reported
or observed.

I would be interested in hearing of test results from other
Pentiums, and also from 486-DX4s and (if anybody has one yet)
the AMD, Cyrix, and NexGen clones of the Pentium.

You may use this information freely as long as you give me
attribution by name and employer.
```

Zij $x = 4195835$, $y = 3145727$, en $z = x - (x/y) * y$, bijvoorbeeld. Dan zou z nul moeten zijn. De processor berekende echter 256. Uit onderzoek bleek dat 1 op 9 miljard delingen een fout resultaat gaf. In het design van de FPU (floating point unit) was dus een menselijke ontwerpfout geslopen die door testen zeer moeilijk te vinden is. Om dergelijke fiasco's in de toekomst te vermijden is men technieken voor automatische verificatie gaan ontwikkelen. Hiervoor heeft men algoritmen ontwikkeld die kunnen nagaan of een ontwerp voldoet aan een specificatie. Een voorbeeld van een specificatie en een implementatie voor een ALU wordt hieronder gegeven:



s_2	s_1	s_0	cin	output	function
0	0	0	0	$F = A$	transfer A
0	0	0	1	$F = A + 1$	increment A
0	0	1	0	$F = A + B$	addition no carry
0	0	1	1	$F = A + B + 1$	addition with carry
0	1	0	0	$F = A - B - 1$	substract with borrow
0	1	0	1	$F = A - B$	substract
0	1	1	1	$F = A - 1$	decrement A
0	1	1	1	$F = B$	transfer B
1	0	0	X	$F = A \vee B$	or
1	0	1	X	$F = A \oplus B$	xor
1	1	0	X	$F = A \wedge B$	and
1	1	1	X	$F = \overline{A}$	not A



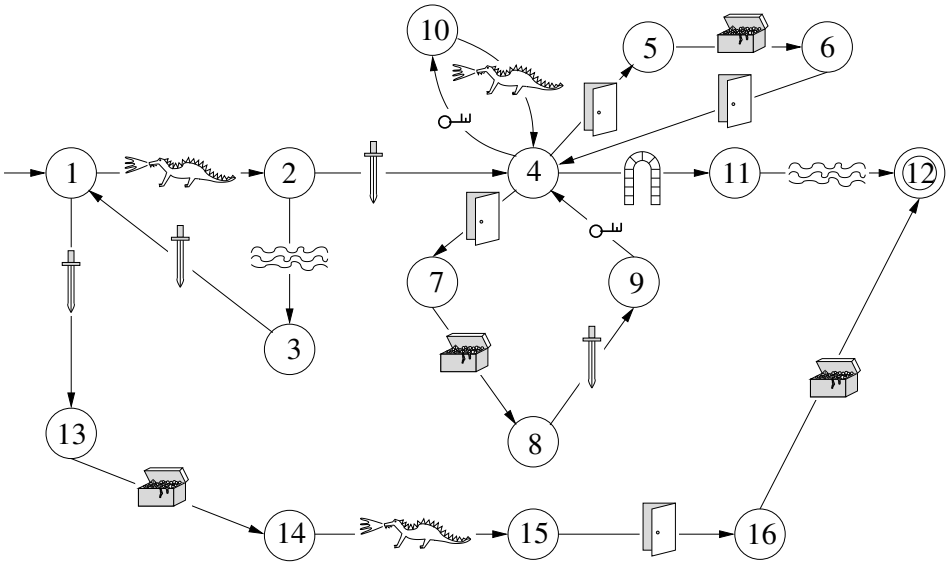
De specificatie definieert formeel aan welke eigenschappen de ALU moet voldoen. Het ontwerp implementeert deze. Zelfs voor eenvoudige eenheden is het zeer moeilijk om na te gaan of de ontwerpen aan de specificatie voldoen.

Verrassend genoeg maken veel van de ontwikkelde technieken gebruik van automatentheorie. Een volledige beschrijving van deze technieken valt buiten het bestek van deze cursus. Wel willen we de student introduceren in dit onderwerp aan de hand van een ludiek voorbeeld. In de volgende paragraaf bespreken we daarom NFA adventures.

2 NFA adventures

De krachtigste toepassing van automatentheorie is automatische verificatie (cf. Pentium bug). In dit project willen we hiervan een voorsmaakje geven aan de hand van een ludiek voor-

beeld: de NFA adventures naar het idee van Brauer, Holzer, König, Schwoon (Bulletin of the EATCS). In Figuur 1 zie je een voorbeeld van zulk een NFA adventure graaf M. Wanneer je M als een NFA bekijkt, dan accepteert deze precies alle paden om van de ingang naar de uitgang te geraken. We zijn niet geïnteresseerd in willekeurige paden, maar wel in deze die aan een aantal voorwaarden voldoen. Bijvoorbeeld, er moeten minstens twee schatten gevonden worden. Deze constraint kan worden gedefinieerd met behulp van een automaat. Inderdaad, de automaat N in Figuur 2 definieert precies alle paden die minstens twee schatten bevatten. In de context van automatische verificatie staat M dus voor het ontwerp en is N de specificatie of de constraint waaraan M moet voldoen. Uiteraard willen we dat er minstens één pad in M is dat aan N voldoet (d.i., minstens twee schatten bevat). Dit kunnen we nu heel eenvoudig nagaan door te testen of de taal aanvaard door $M \cap N$ leeg is. Inderdaad, deze taal is leeg als en slechts als er geen pad in M is dat minstens twee schatten bevat. Wanneer we meerdere constraints $N_1, \ldots, N_k$ hebben, dan moeten we simpelweg testen of $M \cap N_1 \cap \cdots \cap N_k$ leeg is. Testen of een automaat de lege taal aanvaardt, is besproken in Opgave 4.13.



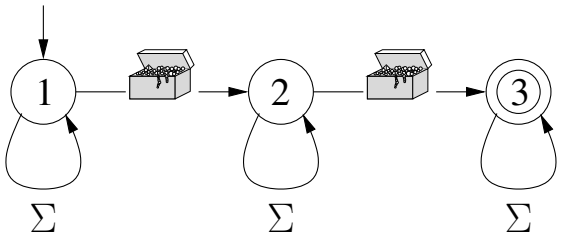
Figuur 1: Een NFA adventure graph.

3 Deel 1: Eindig automaten

In het eerste deel gaan we een eindige automaat uit een tekstbestand inlezen en vervolgens kunnen we hierop een aantal operaties toepassen.

Een `.aut` bestand is een tekstuele beschrijving van een eindige automaat. Zo'n bestand heeft een structuur zoals weergegeven in Figuur 3. Meer bepaald geven `(START) |- 1` en `3 -| (FINAL)` aan dat toestand 1 een begintoestand is en dat 3 een eindtoestand is; `2 a 3` geeft aan dat er een transitie is van `2` naar `3` gelabeld met `a`. ε-transities worden voorgesteld als `2 $ 3`. In een adventure gebruiken we `D`, `S`, `A`, `R`, `K`, `G`, `T`, voor draak, zwaard (sword), boog (arc), rivier, sleutel (key), deur (gate), schat (treasure), respectievelijk.

Opdracht 1. FUNCTIONALITEIT
Alles dient in Java te worden geschreven. Enkel klassen uit de JDK worden gebruikt. *Aange-*



Figuur 2: De NFA N die alle paden definieert die minstens twee schatten bevatten.

```
(START) |- 1
3 -| (FINAL)
1 A 2
2 A 3
2 S 3
3 D 3
```

Figuur 3: `vb.aut`

zien de code gedeeltelijk via scripts zal worden geëvalueerd, dien je je strict aan de volgende klasse- en methodespecificaties te houden.

Voorzie de Java-klasse `AutomatonParser` van de volgende methodes:

`public AutomatonParser(String filename)`
De constructor leest een `.aut` bestand `filename` in.

`public void parse() throws Exception`
Deze methode construeert de eindige automaat behorende bij `filename` (die is doorgegeven in de constructor). Indien blijkt dat het bestand niet van het juiste formaat is (welke zal resulteren in een fout tijdens het parsen), dan wordt een `Exception` geworpen.

`public Automaton automaton()`
Deze methode geeft de eindige automaat berekend met `parse()` terug.

Voorzie de Java-klasse `Automaton` van de volgende methodes:

`public Automaton intersection(Automaton aut)`
Deze methode geeft een eindige automaat terug waarvan de taal gelijk is aan de intersectie van de taal L_1 van de automaat van de klasse en taal L_2 van de automaat `aut`.
Hint: gebruik de productconstructie. Merk echter op dat de productconstructie in het boek van Sipser zich beperkt tot DFA's. Bekijk zelf hoe je het idee van de productconstructie kunt toepassen op NFA's. (Een alternatieve methode van NFA's eerst omzetten naar DFA's en dan de productconstructie van Sipser uitvoeren is waarschijnlijk te traag voor de voorbeeldadventures.)

`public String getShortestExample(Boolean accept)`
Deze methode schrijft een kortste string uit die door de automaat wel (niet, respectievelijk)

wordt aanvaard, als `accept` gelijk is aan `true` (`false`, respectievelijk). Indien er zo geen string is, dan wordt `null` uitgeschreven. De string die door `getShortestExample` wordt teruggegeven dient geen `$`-tekens te bevatten. Bijvoorbeeld de lege string is dus altijd de string `""` en niet `"$"`.

De code in Figuur 4 dient dus probleemloos te werken! Test je programma op de adventures die je kan vinden op BlackBoard.

Je bent vrij om extra klassen/methoden e.d. aan te maken om deze methods te implementeren. Denk na over de manier van werken en kies voor efficiënte methoden. Structureer je code en voorzie commentaar waar nodig.

De code van Figuur 4 zal dus `A` uitprinten als de inhoud van zowel `a.aut` als `b.aut` gelijk is aan hetvolgende:

```
(START) |- 1
1 A 2
2 -| (FINAL)
```

Mocht jouw code niet de juiste uitvoer geven bij aan dit minimale voorbeeld, dan zal jouw inzending niet (verder) worden beoordeeld.

4 Deel 2: NFA Levels

De functionaliteit van Deel 1 gaan we nu toepassen op de NFA adventures.

Opdracht 2. Level 0: God Mode
Schrijf een Java programma `Level0.java` dat, gebruikmakende van de methoden uit Deel 1, nagaat of in de adventure `adventure.aut` de uitgang kan worden gevonden, m.a.w. of een eindtoestand kan worden bereikt vanuit een begintoestand. Meer precies: het programma schrijft op stdout `null` als er geen string is die voldoet aan Level 0, en anders een korste string die voldoet aan Level 0. Je moet geen rekening houden met labels; in dit level ben je onsterfelijk.

Hint. Elke string die wordt aanvaard door de adventure-automaat representeert een pad van de begintoestand naar de eindtoestand. Schrijf een Java-programma dat test of de automaat een string aanvaardt.

Opdracht 3. Level 1: Rincewind Level
In dit level, kan een adventure slechts succesvol worden afgelegd indien aan de volgende voorwaarden voldaan is:

1. Er moeten minstens twee schatten gevonden worden. We gaan er vanuit dat schatten onmiddellijk worden teruggelegd: tweemaal een transitie met een schat passeren volstaat dus.
2. We kunnen slechts voorbij een deur wanneer we eerst een sleutel gevonden hebben. Eén sleutel opent echter alle volgende deuren.
3. Wanneer we een draak tegenkomen moeten we onmiddellijk in de rivier springen (we staan immers in brand[1]), behalve wanneer we een zwaard hebben.

Schrijf een Java programma `Level1.java` met `.aut` bestanden dat, gebruikmakende van de methoden in Figuur 4, nagaat of in de adventure `adventure.aut` een uitgang kan worden gevonden onder de bovenvermelde voorwaarden. Het programma schrijft op stdout `null` als er geen string is die voldoet aan Level 1, en anders een korste string die voldoet aan Level 1. Test je programma op de adventures die je kan vinden op BlackBoard.

[1] Probeert u zich een beetje in de situatie in te leven.

```
import java.io.*;

public class AdventureTest {

    public static void main(String[] args){
        try{
            // leest de automaat a.aut in
            AutomatonParser parse1 = new AutomatonParser("a.aut");
            // leest de automaat b.aut in
            AutomatonParser parse2 = new AutomatonParser("b.aut");
            parse1.parse();
            parse2.parse();
            Automaton aut1 = parse1.automaton();
            Automaton aut2 = parse2.automaton();
            // berekent de intersectie van aut1 en aut2
            Automaton result = aut1.intersection(aut2);

            // schrijft een string uit die wordt aanvaard
            // (true staat voor aanvaarden)
            // indien er zo geen string is wordt null
            // uitgeschreven
            System.out.println(result.getShortestExample(true));
        }
        catch(Exception e){
            System.out.print("Error:  ");
            System.out.println(e.toString());
            System.out.println(e.getMessage());
        }
    }

}
```

Figuur 4: AdventureTest.java

Hint. Elke string die wordt aanvaard door de adventure-automaat representeert een pad van de begintoestand naar de eindtoestand. Construeer voor elke beperking een automaat, die precies die strings aanvaardt die voldoen aan de beperking. Test dan of de doorsnede van de adventure-automaat met al de beperkingautomaten leeg is.

Opdracht 4. Level 2: Cohan Level
Alle regels van het vorige level blijven gelden. Daarenboven geldt dat we alle gevonden schatten verliezen zodra we een boog passeren. Schrijf opnieuw een Java-programma `Level2.java` met (`.aut` bestanden) dat nagaat of in de adventure `adventure.aut` de uitgang kan gevonden worden onder de gegeven voorwaarden. Het programma schrijft op stdout `null` als er geen string is die voldoet aan Level 2, en anders een korste string die voldoet aan Level 2.

Vanwege automatische tests, mag bij ieder van deze levels niets anders worden uitgeprint op stdout.

Hint. Test ieder van je programma's `LevelX.java` op de adventures die je kan vinden op BlackBoard. Echter, test ook op zelfgemaakte adventures. De ingeleverde programma's zullen namelijk ook op adventures worden getest die niet op BlackBoard staan.

Dit project toont aan hoe automaten kunnen worden gebruikt voor verificatie. Uiteraard worden normaal geen adventures geverifieerd maar wel ontwerpen van elektronische schakelingen, communicatie protocollen, reactieve systemen, ... De grootte van zulke systemen varieert van tien toestanden tot enkele tienduizenden.

5 Verslag

Voorzie een kort inleidend verslag voor jullie project waarin de volgende onderdelen terug te vinden zijn:

- korte inleiding;
- structuur van jullie implementatie met bondige omschrijving per klasse;
- taakverdeling indien van toepassing;
- korte uitleg hoe de levels zijn geïmplementeerd;
- figuren voor alle constraints;
- indien je een probleem bent tegengekomen, vermeld dit dan incl. de gekozen oplossing. Beschrijf bijvoorbeeld als (en waarom) een van je levels niet correct werkt op een adventure op BlackBoard.

Hou het verslag gestructureerd en verzorg je taalgebruik. Hou het verslag kort en bondig, het is een informatief overzicht van jullie werkwijze, geen roman!

6 Afspraken

Het project wordt gemaakt in groepjes van twee, de groepssamenstelling is vrij. Iedere groep maakt uiterlijk 10 november 2016 in BlackBoard een Group aan bestaande uit de twee personen. Laat de groepsnaam "`Project 1617 [achternaam1],[stud.nr1] [achternaam2],[stud.nr2]`" zijn. Degenen die geen partner kunnen vinden, kunnen een email sturen naar `robert.brijder@uhasselt.be`. Deze personen worden dan willekeurig in groepjes ingedeeld.

Dit project telt mee voor 20% van het totale cijfer van dit vak. De beoordeling is gebaseerd op zowel functionaliteit als de kwaliteit van het verslag en de code (incl. commentaar).

Lever een zip-bestand in die alle java code, `.aut` bestanden, en het verslag bevat. Omwille van automatische tests is het *niet* toegestaan je code onder te brengen in packages; gebruik dus een platte folderstructuur. Het zip-bestand bevat *geen* folders.

Inlevering via BlackBoard, onder Assignments. Deadline: **dinsdag 29 november 2016, 9u00** (strict).

6.1 Plagiaat

Het project zal intensief met de hand en met speciaal daarvoor ontworpen software gecontroleerd worden op plagiaat. Niet alleen het overnemen van andermans code, maar ook het geven van je eigen code aan iemand anders wordt beschouwd als fraude. Hetzelfde geldt voor expliciet samenwerken. Wanneer we fraude vaststellen moeten we, conform het onderwijs- en examenreglement (OER), de examencommissie hiervan onmiddellijk op de hoogte stellen. Er zal dan een fraudeprocedure opgestart worden; zie het OER.