Welkom vrienden, familie, collega's en anderen. Leuk dat jullie allemaal zijn gekomen.

Ons leven wordt steeds **afhankelijker** van software. Dit werd vorige week **pijnlijk** duidelijk toen ons landelijk alarmnummer niet meer bereikbaar was. ***** Ondanks de **backupsystemen**, lag 112 een paar uur plat. De oorzaak was waarschijnlijk een **softwarebug**.

Hoe kunnen we software veiliger en **robuuster** maken? Dit is een erg **lastig** probleem. Een van de aspecten is, denk ik, **begrijpen** wat de software eigenlijk doet. Dit klinkt misschien als een **trivialiteit**, maar veel bedrijven gebruiken grote stukken software die **jaren** geleden geschreven zijn. Heb je dan nog een goed **beeld** van wat de software doet?

*****

Het leren van automaten is een techniek waarmee we **automatisch een model** kunnen maken van het gedrag van software. Dit model kan dan worden gebruikt om de software beter te **begrijpen** en te analyseren op fouten. Het leren van automaten is een redelijk **nieuwe** techniek. En mijn onderzoek gaat dan ook over het **uitbreiden** van deze techniek.

De modellen die we leren zijn vaak **toestandsdiagrammen**. ***** Hierin worden toestanden van het systeem aangeduid met **rondjes**. En acties die kunnen gebeuren, worden aangeduid met **transities** tussen de toestanden. Hier is het systeem bijvoorbeeld in een toestand waar iemand **112 kan bellen**, en als iemand belt, zijn we in een andere toestand. Dan moet er worden **doorgeschakeld**. Natuurlijk zijn er ook nog **andere transities**.

Dit voorbeeld is onrealistisch **klein**. Echte modellen zijn veel **groter**. ***** Hier zien we een model van software van een **Océ** printer. Elk **puntje** is een toestand, en elke lijntje is een transitie. Laten we nu even **indenken** dat we nog niet weten hoe dit model precies eruit ziet. Hoe ziet het er dan uit als we het **gaan leren**? Op de volgende slide zien we zo een **filmpje** waarin we dit model leren. Het **zwarte gedeelte** is dan het geleerde model.

*****

Automatisch **ontdekt** het algoritme de verschillende toestanden van de software. Dit ziet er **misschien makkelijk** uit in dit filmpje. Maar dat is het **niet**. **Aanvankelijk** kan het leeralgoritme de verschillende toestanden namelijk nog

niet goed **onderscheiden**. Dit komt omdat ze erg op **elkaar lijken**. Twee toestanden kunnen bijvoorbeeld **dezelfde** uitvoer geven en pas na meerdere invoeren een verschil laten zien. De toestanden zijn dan anders, maar dat zien we **niet meteen**.

Tijdens het leren, zal het algoritme **communiceren** met de software. Gaandeweg leert het dan de toestanden te **onderscheiden** en het leert ook de toestanden te **bereiken**. Vooral dit laatste zien we duidelijk in dit filmpje.

Ik wil een stukje hier **uitlichten**. ***** Hier heeft het algoritme **bepaalde transities** geleerd (laser). En als-ie dan **doorleert**, zien we dat die transities **verdwijnen**. Ze verdwijnen niet echt, wat hier gebeurd is dat het algoritme ziet dat deze transities **incorrect** waren. Ze zijn **vervangen** door transities naar nieuw ontdekte toestanden. Hoe **weet** het leeralgoritme dat-ie moet corrigeren? Dit gebeurt door middel van **Black Box Testing**.

*****

Dit is het **eerste** thema in mijn proefschrift. Het **genereren** van tests, waarmee we het leeralgoritme een **handje helpen**. We nemen hierbij een van de hypotheses van het leeralgoritme en genereren een **verzameling tests** om uit te voeren op de software. Het gaat hierbij om **efficiëntie en volledigheid**. ***** Dit zijn **conflicterende** belangen. Testen kost **tijd**, dus we willen zo min mogelijk testen, dat is efficiënt. Aan de andere kant willen we, als de hypothesis nog niet goed is een verschil kunnen **aanwijzen** door middel van een test, dat is volledigheid.

In mijn proefschrift heb bestaande testgeneratie methodes uit de literatuur in **kaart** gebracht. Aan de hand van deze kaart, heb ik ook **nieuwe** algoritmes ontwikkeld, die bij een gegeven volledigheid efficiënt zijn.

Het **tweede** thema van mijn proefschrift wil ik ook toelichten aan de hand van het filmpje. * Er zijn hier twee **driehoeken** (tekenen). En als het algoritme doorleert, zien we het **volgende** gebeuren. Alles wat in de linker driehoek geleerd wordt, wordt **ook** rechts geleerd. Alles gebeurt in **tweevoud**. Er is hier sprake van een bepaalde **symmetrie** in dit stukje software.

Ik heb die symmetrieën **niet verder gebruikt** bij het leren van deze software. Maar ik heb het gebruikt bij een **ingewikkelder** probleem. Namelijk het leren van automaten over **oneindige** alfabetten. * Dit klinkt misschien **esoterisch**, oneindige alfabetten. Maar de motivatie komt echt uit de **praktijk**. Bij digitale

communicatie **protocollen** worden vaak getallen gebruikt om kanalen aan te duiden. Zo zit bijvoorbeeld Henk op **kanaal 1** en Ingrid op kanaal 2. We kunnen dit net zo goed **om wisselen**, Henk op 2 en Ingrid op 1. En dat is een **symmetrie**. In plaats van maar twee kanalen, zijn er **heel veel kanalen**. En de theorie wordt eenvoudiger als we aannemen dat we **oneindig** veel kanalen tot onze beschikking hebben. Deze combinatie van oneindigheid van symmetriën is de theorie van **nominale** technieken. ***** Met deze theorie kunnen we bestaande algoritmes gemakkelijk **vertalen** naar een situaties met oneindige alfabetten.

*****

Dit zijn de twee thema's in mijn proefschrift, beiden in de **context** van het leren van automaten. Ik hoop dat jullie nu een beeld hebben van de **vaktermen** in de titel van mijn proefschrift.

Het leren van automaten is een **fascinerend** onderwerp. Het vakgebied staat misschien nog in de **kinderschoenen**, of misschien tienerschoenen. Het zal niet direct de problemen rondom 112 gaan oplossen. Maar ik geloof dat het een interessante toevoeging is voor de **gereedschapsdoos** van software ontwikkelaars en testers. En dat we hiermee software beter begrijpen en kunnen **verbeteren**.

Dank voor uw aandacht.