Processen, Modellen en Spin

Deel 1: Promela programma is een executeerbaar model van gedistribueerde processen

Software, en met name control, bestaat dikwijls uit verschillende processen of threads die met elkaar moeten samenwerken. De non-deterministische afwisseling van executies in die processen kan leiden tot zeldzame en moeilijk reproduceerbare fouten. In de 'process meta language' (Promela) kunnen procesmodellen worden beschreven, die met de 'Spin model checker' kunnen worden 'doorgemeten' op door de ontwerper gespecificeerde systeemtoestanden en executiepaden. Dit eerste artikel van twee beschrijft het modeleren in Promela; het tweede behandelt de verificatie van dergelijke modellen met de Spin-tools.

HANS VAN THIEL

Een klassiek computerprogramma leest een bepaalde invoer, voert daarop berekeningen uit, schrijft het resultaat naar een opslagmedium en stopt dan. De software op een moderne PC bestaat al uit vele verschillende programma's (processen) die schijnbaar tegelijkertijd (concurrent) lopen, in werkelijkheid steeds afgewisseld op dezelfde CPU en gecoördineerd door een OS. Een netwerk, in een auto bijvoorbeeld, verbindt verschillende systemen die onafhankelijk van elkaar programma's uitvoeren, maar ook dikwijls met elkaar moeten communiceren om hun taken te vervullen. Kenmerken van dergelijke gedistribueerde programma's zijn dat zij nooit stoppen, alleen soms wachten op andere processen, en dat hun snelheden en de verschillen daarin niet van te voren vast liggen.

Omdat elke executiestap in een bepaald proces op elk moment afgewisseld kan worden door een executiestap in een van de andere processen (interleaving) wordt al gauw een gemeenschappelijke

'resource', bijvoorbeeld een globale variabele, op het verkeerde moment veranderd of afgelezen. Verder kunnen processen in een toestand komen dat de ene op de andere wacht en de andere op de ene (deadlock), of dat een bepaald proces steeds verdrongen wordt door de anderen (starvation).

Het ergste is dat sommige combinaties die tot een fout leiden zo zeldzaam zijn dat ze niet voorkomen tijdens het testen, of, als ze al voorkomen, niet gereproduceerd kunnen worden. Dergelijke onwaarschijnlijke fouten manifesteren zich weliswaar soms nooit, ook niet als het betreffende product in bedrijf is, maar soms ook wel, en dan met desastreuze gevolgen.

Modellen

Het is echter mogelijk om vereenvoudigde modellen op te stellen van de uiteindelijke software, waarin van de (meeste) berekeningen wordt geabstraheerd, maar juist de communicatie tussen en coördinatie van software processen wordt benadrukt. Dit modeleren voegt een extra stap toe aan het ontwerptraject, maar aan het begin, daar waar fouten het gemakkelijkst hersteld kunnen worden en de kosten ervan nog laag zijn.

Extra aandacht is wel nodig voor validatie van het model, de toetsing of dat inderdaad op de wezenlijke punten overeenkomt met het gedachte systeem. Ten tweede moet de uiteindelijke 'echte' software op de wezenlijke punten natuurlijk overeenstemmen met het model daarvan.

De tussenstap zelf, het model van de gedistribueerde processen, kan echter in principe volledig worden geverifieerd op alle mogelijke interacties die in het model kunnen voorkomen (1). Dit kan als het model bestaat uit een toestandsdiagram met mogelijk oneindige (door cyclussen) executiepaden, maar een eindig aantal toestanden. De moeilijkheid is dan om de complexiteit te beheersen door toepassing van de juiste zoektechnieken en -algoritmen. Dit nu wordt in hoge mate gerealiseerd door de Spin model checker (2, 3).

Spin is een open source tool dat is ontwikkeld door Gerard Holzmann van het NASA JPL 'laboratory for reliable software' (zie 4). Holzmann schreef ook het standaardwerk over Spin, maar er zijn inmiddels diverse boeken over gepubliceerd (zie 3). Spin is in de praktijk voor diverse projecten gebruikt. In Nederland is het toegepast bij het ontwerp van het 'beslis- en ondersteuningssysteem' van de stormvloedkering in de Nieuwe Waterweg.

De modellen, die met Spin kunnen worden gesimuleerd en geverifieerd, zijn geschreven in de 'process meta language'. Een Promela programma is dus een executeerbare (door simulatie) specificatie van de uiteindelijke software.

Non-determinisme

Promela lijkt veel op de C programmeertaal, zoals in Listing 1 is te zien, maar het abstraheert van de meeste berekeningen die in de uiteindelijke software zullen voorkomen. Zo heeft Promela geen functies (wel inline macro's) en geen schuivende-komma variabelen. Ook control structuren zoals 'while', 'for' en 'if... else' ontbreken. Een reeks vertakkingen met tests wordt aangegeven met if ... fi voor een eenmalige executie en do ... od voor een lus. Een test begint met twee dubbele punten en de statements na de test komen na een pijl (of een punt-komma, naar keuze van de programmeur).

Spin is een open source tool dat is ontwikkeld door Gerard Holzmann van het NASA JPL 'laboratory for reliable software'

Het belangrijkste in Promela is dat de programma-statements na een test alleen worden uitgevoerd als de test slaagt. Tot die tijd is het betreffende proces geblokkeerd. Verder is de volgorde van de tests non-deterministisch, en kunnen er meer tests 'tegelijkertijd' tot 'true' evalueren.

Statements binnen een proces zijn sequentieel en dus deterministisch binnen dat proces. Elk statement in een bepaald proces kan echter afgewisseld worden door een statement van een ander proces. Dat geldt ook voor een statement direct na een test. Als dus bijvoorbeeld een globale variabele x in het ene proces getest wordt op gelijkheid met 5, kan die variabele in een ander proces op 0 gezet worden. Pas dan wordt in het eerste proces de volgende expressie geevalueerd op basis van de, nu foutieve, waarde 5.

Het doel is om een softwaremodel te ontwerpen dat vrij is van zulke fouten, en de functie van de model checker is om ze te vinden als ze er toch in voorkomen.

Besturing

Listing 1 is een model van de besturingssoftware van een spoorwegovergang. Het proces 'Trein' gebruikt hiervoor een lokale variabele die aangeeft of de trein ver weg is, de overgang nadert, of op de overgang is.

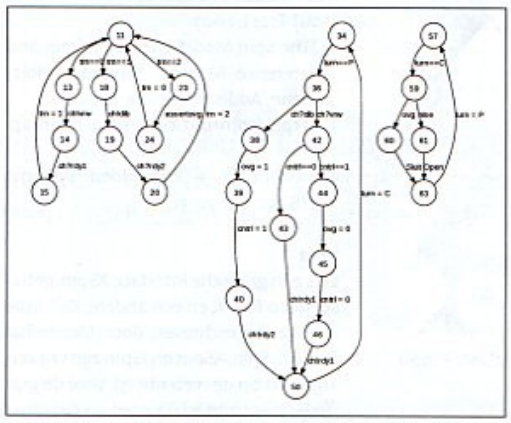
Het proces 'Controller' onderscheidt tussen de mogelijkheden dat de trein ver weg is of dichtbij, met een extra bit dat aangeeft of de trein nadert of juist vertrekt. De overgang wordt gerepresenteerd door een globale variabele die door een derde proces, 'Log' wordt afgelezen. Alle drie processen zijn non-terminerend; als het eind van de lus is bereikt begint het proces weer. (De goto statement en de do ... od lus zijn hier gelijkwaardig.)

Alle drie processen zijn gedistribueerd. Zonder coördinatie zou het kunnen dat de trein dichtbij is, de controller aangeeft dat hij ver weg is, de overgang open staat, en dat de logger niet de actuele maar een vorige toestand noteert.

Misschien is deze weergave van de gang van zaken enigszins misleidend, omdat die suggereert dat de listing de trein en de overgang simuleert. Het Promela programma simuleert echter de besturingssoftware, en slechts indirect de daadwerkelijke trein met overgang.

Figuur 1 is een door JSpin (zie noot) gegenereerde grafische representatie van alle drie processen uit listing 1. Elke toestand wordt in de figuur voorgesteld door een cirkel met het betreffende regelnummer in die listing. Elke evaluatie van een expressie, dus ook een test, is een toestandsovergang.

Processen in Promela kunnen communiceren via message channels, of via globale variabelen. Het proces 'Log' is geco-



Figuur 1: Een grafische representatie, gegenereerd door JSpin, van de drie processen uit listing 1. Cirkels representeren toestanden; de cijfers corresponderen met de regelnummers in de listing. In een Promela model is een test geen toestand, maar een toestandsovergang. Het model van het gehele systeem bestaat uit alle mogelijke combinaties van toestanden en overgangen.

ordineerd met de controller via het producer/consumer mechanisme; elk proces wacht op zijn beurt terwijl de ander iets doet, en elk proces geeft zelf aan het eind de beurt aan de ander. De trein en de controller processen communiceren via een message channel.

In de declaratie wordt aangegeven dat het betreffende kanaal ongebufferd is (door het getal 0) en dat het type van de boodschap 'mtype' is.

Een message channel kan zenden, aangegeven door een uitroepteken achter de naam, of ontvangen, aangegeven door een vraagteken. Achter het uitroep- of vraagteken staat de inhoud. Als een kanaal niet gelezen kan worden wordt het geblokkeerd tot dit wel het geval is.

Omdat kanaal chan in listing 1 ongebufferd is werkt het, met twee verschillende messages en twee responsboodschappen, als rendez-vous tussen de twee processen.

De drie processen kunnen nu met Spin en desgewenst de grafische tools XSpin en JSpin worden gesimuleerd en geverifieerd. De assert in het trein proces is analoog aan een assert statement in C; als de conditie niet geldt wordt dat als een software fout beschouwd. Anders dan in een C programma (en in Spin simulatie modus) geldt de conditie in verificatiemodus niet alleen voor de uitgevoerde executies, maar voor alle executies die maar mogelijk zijn. Deze volledige verificatie van Promela modellen is het onderwerp van het tweede artikel.

Literatuur

1] "Principles of Model Checking", Christel Baier and Joost-Pieter Katoen, MIT Press, 2007.

2] "The Spin Model Checker, Primer and Reference Manual", Gerard J. Holzmann, Addison-Wesley, 2004

3] http://spinroot.com/spin/sitemap.html

4] Elektronica + embedded systems 7/8-2008, p 30-31

Noot

Er is een grafische interface XSpin, gebaseerd op Tcl/Tk, en een andere, JSpin, die in Java is geschreven door Mordechai Ben-Ari. Spin, XSpin en JSpin zijn vrij verkrijgbaar op de web site (3). Voor de grafische tools zijn Tcl/Tk, Java en Graphviz (visualisatie diagrammen) vereist, en voor verificatie (niet simulatie) een C compiler. De huidige versie 5.2.4 van Spin is van december 2009. ■

```
mtype = {vrw, dib, rdy1,rdy2 };  /* signalen voor message channel ch */
chan ch = [0] of { mtype };      /* ongebufferd channel, message is 1 mtype veld */
bool ovg= false;                 /* globale variabele voor overgang */
mtype = {P, C };                 /* producer/consumer waarden */
mtype turn = P;                  /* globale variabele voor Log synchronisatie */

active proctype Trein() {
     byte trn;      /* default altijd 0, krijgt hier waarden 0, 1 of 2 */

 again :
     if
     :: trn == 0 ->          /* trein is ver weg */
                ch!vrw;   /* stuurt message dat trein ver weg is */
                ch?rdy1;  /* controller is klaar */
                trn = 1;  /* volgende toestand trein */

     :: trn == 1 ->          /* trein is dichtbij */
                ch!dib;   /* stuurt message dat trein dichtbij is */
                ch?rdy2;  /* controller is klaar */
                trn = 2;  /* volgende toestand trein */

     :: trn == 2 ->          /* trein is op overweg */
                assert(ovg); /* test dat overweg dicht is */
                trn = 0;     /* volgende toestand trein */
     fi;

     goto again
}

active proctype Controller() {

     bit cntrl;   /* verre trein kan komen (0) of vertrekken (1)  */

     do
     :: turn == P ->
     if
     :: ch?dib ->            /* ontvang message dat trein dichtbij is */
               ovg = true; /* zet spoorboom sein op dicht */
               cntrl = 1;  /* cntrl toestand wordt dat trein vertrekt */
               ch!rdy2;    /* stuur ready message 2 */

     :: ch?vrw -> if
               :: cntrl == 0 -> ch!rdy1;     /* stuur ready message 1 */
               :: cntrl == 1 -> ovg = false;  /* spoorboom sein op open */
                                cntrl = 0;    /* (nieuwe) trein is ver */
                                ch!rdy1;      /* stuur ready message 1 */
               fi;
     fi;

     turn = C;
     od
}

active proctype Log() {

     do
     :: turn == C ->
     if
     :: ovg  -> printf("Sluit\n");
     :: else -> printf("Open\n");
     fi;
     turn = P;
     od
}
```

Listing 1: Een specificatie van de besturingssoftware van een trein en een spoorwegovergang in Promela. De trein communiceert met de controller via rendez-vous messages. De controller is gesynchroniseerd met een logger via een globale variabele. De 'assert' in het trein proces verifieert dat, wanneer de trein over de overgang rijdt, de overgang is gesloten.