Modelleren van complexe systemen: verslag practicum 2

Door Francis Duvivier en Yuri Passchyn

1. Inleiding

Dit is het verslag van het tweede practicum van Modelleren van Complexe systemen. In dit verslag bespreken we de design-beslissingen bij het maken van het idp-gedeelte in het 2e punt en het NuSMV-gedeelte in het 3e punt van het verslag. De tijd gespendeerd aan dit practicum is per persoon gemiddeld 13 uur aan het idp-gedeelte en 17 uur aan het NuSMV-gedeelte, in het totaal is dit dus 60 uur.

2. Deel IDP

2.1. Linear time calculus

We maken gebruik van linear time calculus om het dynamische aspect van het systeem te modelleren. Bij de causale regels hebben we ervoor gekozen om geen cause_gold te hebben, omdat het aantal goudstukken alleen maar omlaag kan gaan. Buiten gold Hebben we enkel nog het Position-predicaat dat gebruik maakt van een causaal predicaat. Hierbij is er ook een predicaat Cn_Position dat aangeeft dat het Position-predicaat niet waar zal zijn voor een bepaalde agent, positie en plaats in het volgende tijdstip.

2.2. Fluents & Actions

We maken onderscheid tussen de twee. Beide zijn dynamische symbolen. Fluents zijn gevolgen van actions. Actions veranderen zonder oorzaak van waarde van het ene naar het volgen tijdstip. Ze zijn de drijfveer voor het veranderen van waarde (van het ene naar het volgende tijdstip) van de andere predicaten die een tijd-parameter hebben.
Uitleg over elk predicaat vindt u terug in de commentaar in de broncode:

- We hebben maar één action:
  - Move(time,agent,dir).
- Onze fluents zijn
  - Gold(time, xCo, yCo)
  - Cn_Gold(time, xCo, yCo)
  - Position(time, agent, xCo, yCo)
  - C_Position(time, agent, xCo, yCo)
  - Cn_Position(time, agent, xCo, yCo)
  - GameLost(time)
  - GameWon(time)
  - PreviousMove(time ,agent,dir)
  - StupidGhost(time)
  - PCrossedAG(time)
  - NextTo(time,agent,agent)

2.3. Randen van de doolhof

We kiezen ervoor om te werken met bounds aan de randen van het doolhof. We gaan dan na of een positie al dan niet OOB(out of bounds) is om ervoor te zorgen dat agents binnen de rand blijven. Een andere mogelijkheid is om aan de randen muren te definiëren zodat agents er niet door kunnen bewegen, dit zou zich vertalen in OOB(x,y,dir) → FoundWall(x,y,dir) maar wij zagen er geen voordeel aan dit toe te voegen. Door dit niet te doen wordt het onderscheid nog gemaakt waardoor het programma makkelijker aanpasbaar zou zijn voor een verandering die dit onderscheid nodig heeft, zoals bijvoorbeeld de mogelijkheid om van de ene rand in de andere uit te komen.

2.4. Regels van de modellering

2.4.1. Een aantal hulppredicaten:

- Neighbor modelleert voor twee vakjes en een richting of de vakjes elkaars buur zijn. Dit houdt geen rekening met muren, het gaat hier puur over de coördinaten. In het predicaat wordt ook gedefinieerd in welke richting de tweede coördinaat t.o.v de eerste ligt.
- Steppable maakt gebruik van Neighbor. Het gaat na voor 2 vakjes of vanuit het ene vakje naar het andere kan worden gestapt, dit houdt wel rekening met de aanwezigheid van muren.
- NeighborNoPos gaat na voor een vakje en een richting er naast het vakje in die richting een NoPos is.
- OOB gaat na of vanuit een vakje naar een bepaalde richting er out of bounds zou worden bewogen.
- FoundWall gaat na of vanuit een vakje naar een bepaalde richting door een muur bewogen zou worden.
- Reach modelleert of een vakje bereikbaar is vanuit een ander vakje, dit maakt gebruik van Steppable.

3. Deel NuSMV

3.1. Structuur

De structuur van de NuSMV modellering was al gedeeltelijk gegeven en we hebben uiteraard dit skelet gevolgd. Dit wil zeggen dat er 3 modules zijn.

De eerste is de main-module die 2 agent-modules en 10 square-modules initialiseert. In de main-module wordt ook de gameState en gameover variabele gedeclareerd. Wij hebben toegevoegd in deze module dat de waarde van deze variabelen gespecificeerd wordt in de initiële en de volgende staten aan de hand van de instanties van de andere modules. Verder hebben we in deze module nog de stillCoins boolean gedefinieerd.

Als laatste hebben we ook nog bijna alle gevraagde verificaties in deze module gezet.

De tweede gegeven module is de agent-module. Deze module houdt de staat van een spookje of pacman bij. Dit houdt in dat er een x en y variabele voor de huidige positie gedeclareerd wordt en ook nog vier booleans, de richting-variabelen: u(up), d(down), l(left), r(right). Degene die waar is geeft aan in welke richting pacman zal gaan bij de overschakeling naar de volgende staat. De initiële waarden van x en y worden ingesteld en ook wordt er bepaald wat de volgende waarden van x en y zal zijn. Als laatste wordt er ook nog een beperking op de waarden van u, d, l en f opgelegd met een TRANS-

specificatie.

De derde module is de square-module waarvan de instanties een bewandelbare tile voorstellen tezamen met de muren die er eventueel rond staan. Ook wordt er nog een variabele hasCoin gedeclareerd en ingesteld die aangeeft of er al dan niet een goudstukje op de square staat in de huidige staat. Verder wordt er in deze module met TRANS-specificaties de volgende waarde van u, d, l en r van pacman en ghost bepaald. Als laatste wordt hebben we hier de specificatie gezet die zegt dat de hoeveelheid goud niet groter mag worden.

3.1. Design-beslissingen

3.1.1. Overgangen positie agent

De positie van een agent wordt zoals gezegd voorgesteld met de x en y in de agent-module. Het is ook de agent-module die de initiële en volgende waarde van deze variabelen bepaalt. De initiële waarde wordt gegeven door de parameters startX en startY. De volgende waarden van x en y worden bepaald door de huidige waarde van u, d, l, en r. Op elk moment is er maximum 1 van deze booleans waar en deze bepalen dus of de agent zal bewegen en naar welke richting. De waarde van de richting-variabelen wordt echter ingesteld door de square-module. Dit is omdat de square-module de informatie heeft om te kijken of een agent op zijn plaats staat, en zo ja, weet welke richtingen mogelijk zijn. De square-module specificeert met TRANS-specificaties dat als pacman of ghost dezelfde positie heeft als de square zelf, dat dan de richting van pacman of ghost niet naar een muur toe mag zijn. We gebruiken dus hetzelfde systeem om muren aan te geven bij een vakje als om de richting van een agent aan te geven. Op deze manier wordt het opleggen van restricties op de richting aan de hand van muren op een vakje heel simpel en duidelijk. Om te zorgen dat pacman steeds maar 1 richting gaat en niet beweegt als het spel gedaan is gebruiken we een TRANS specificatie in de agent-module. Deze maakt gebruik van de gameover-parameter om dit uit te drukken.

3.1.2. Overgangen goudstuk op square

Of er al dan niet goud staat op een square wordt bepaald door de hasCoin-boolean in de square-module. Deze boolean wordt geïnitialiseerd met FALSE als pacman er op staat in de initiële staat en anders random. De reden waarom hij op FALSE wordt geïnitialiseerd als pacman er op staat komt later aan bod en staat ook in de commentaar vermeld. De square-module gebruikt de pacman-parameter om te bepalen of dat goud al dan niet moet blijven staan. We kijken op pacman in de huidige staat op deze square staat door na te gaan of de x en y gelijk zijn aan de x en y van deze square. Als pacman op deze square staat dan zal hasCoin FALSE moeten worden in de volgende staat.

3.1.3. Globale hoeveelheid goud

Om dan in de main-module te checken of er globaal gezien nog goud is gebruiken we de stillCoins-boolean die voor alle squares die in de main-module geïnstantieerd zijn kijkt of er nog goud op staat of niet. De definitie van stillCoins moet dus wel aangepast worden als er een square met goud op toegevoegd wordt.

3.1.4. Overgangen gameState en gameover

De initialisatie en bepaling van volgende waarden van gameState gebeurt in de main-module. Deze wordt initieel steeds op playing gezet. Om de volgende waarden van gameState te bepalen wordt er gekeken naar de waarde van stillCoins en naar de posities van pacman en ghost. Eerst kijken of het spel nog wel bezig is door gameover te checken. Als het gameover TRUE is, dan blijft gameState onveranderd. Dan checken we of pacman op dezelfde plaats zal staan als ghost in de volgende staat,

vervolgens kijken we of ze zullen kruisen. Indien één van de vorige waar is wordt gameState LOSE in de volgende staat. Zo niet, dan kijken we of er nog goudstukken over zijn met hasCoin. Indien stillCoins FALSE is, dan zal gameState WIN worden. Als geen van bovenstaande condities voldaan is dan blijft de gameState onveranderd.

De waarde van gameover wordt bepaald door een TRANS-specificatie die uitdrukt dat gameover equivalent is met gameState!=playing.

3.1.5. Delayed start en initiële staat

In ons model mag pacman nog geen richting hebben in de initiële staat. Dit hebben wij zo geïmplementeerd om requirements te kunnen opstellen van de vorm: “het is mogelijk dat voor eender beginrichting van pacman...”. Het is zo dat CTL requirements in NuSMV enkel waar zijn als ze waar zijn voor elke mogelijk initiële staat. Maar als pacman dus in de initiële staat al zou beslissen in welke richting hij zou bewegen, dan kunnen we die bepaalde requirements niet maken. Aangezien we ook alle borden met initieel 1 goudstuk willen toelaten zou het dan bijvoorbeeld niet mogelijk zijn om een requirement te maken die zegt dat het steeds mogelijk is dat pacman verliest. Immers, als er een initiële staat is waarin er maar 1 goudstuk is , dat bovendien naast pacman staat en de initiële richting van pacman naar dat goudstuk toe gericht is, dan zou in de initiële staat al bepaald zijn dat het spel gewonnen zou worden als ghost ver genoeg weg is.

Dit wil ook zeggen dat pacman pas vanaf de 2e state begint te bewegen. En omdat we alle spellen met 1 goudstuk willen toelaten zonder dat het spel in de initiële staat al gedetermineerd zou zijn, hebben we gespecificeerd dat er op de plaats van pacman geen goudstuk mag staat in de initiële staat. Op die manier kunnen we om na te gaan dat een initieel bord zinvol is gewoon nagaan of stillCoins=FALSE in de initiële staat. In de code doen we dit door te specificeren dat gameover TRUE moet zijn als stillCoins FALSE is en dan nog te specificeren dat in de initiële staat gameState playing is wat equivalent is met gameover=FALSE.

3.1.6. Implementatie requirements.

Zoals vermeld kan men de requirements terugvinden met commentaar in de main-module en square module. Deze requirements slagen allemaal behalve degene die zegt dat het niet mogelijk mag zijn dat het spel in 10 stappen al gewonnen is. Dit komt omdat wij er niet van uitgaan dat er initieel op elke square goud ligt aangezien we dit nergens in de opgave gelezen hebben. Als er dus bijvoorbeeld maar 1 goudstuk is dat bovendien naast pacman ligt, dan kan het spel in de 2e stap voor pacman (=3e staat) al gewonnen zijn.