Project MCS: Logical Pacman
Iteraties 2& 3

Jessa Bekker s0215494
Karel Moens s0215430

11 januari 2014

1 Inleiding

In dit verslag worden kort de ontwerpbeslissingen bij het uitwerken van het tweede deel van het project toegelicht. De opdracht was ... Tot slot vermeldt de tekst ook de tijd die nodig was om dit te verwezenlijken.

2 IDP

2.1 Verbetering iteratie 1

Ondanks de correcte werking van onze oplossing voor iteratie 1, waren enkele verbeteringen op vlak van complexiteit en performantie mogelijk. Voor de performantie werd de voorwaarde dat alle vakjes elkaar moeten bereiken versoepeld naar dat één vakje alle andere moet kunnen bereiken, wat hetzelfde impliceert. Verder werd het predicaat `Edge(x1,y1,x2,y2,d)` toegevoegd dat verschillende voordelen geeft. Edge drukt de relatie uit tussen 2 aangrenzende vakjes op posities (x1,y1) en (x2,y2) waarvan vakje 2 in richting d ten opzichte van vakje 1 ligt. Dit wordt gebruikt om `Reach(x1,y1,x2,y2)` efficienter na te gaan door enkel naar buren te kijken en niet naar eender welk vakje. Het wordt ook gebruikt om mogelijke stappen te vinden (die niet door een muur, van het bord af of naar een NoPos gaan), wat gebruikt wordt als voorwaarde op `Move` en de 2 nieuwe regels: `DeadEnd` en `Crossed`. De implementatie wordt hierdoor minder complex.

2.2 GameWon en GameLost

`GameWon` en `GameLost` zijn niet als fluents geimplementeerd. De reden hiervoor is dat de natuurlijke wijze om regels over winnen/verliezen te verwoorden van het type zijn: "De huidige staat is ..., dus nu is het spel gewonnen/verloren". Bijvoorbeeld: "In de huidige staat zijn er geen muntstukken meer dus het spel is nu gewonnen." Gebruikmakende van een fluent zou de regel moeten verwoord worden als "In de huidige staat wordt een actie uitgevoerd waardoor het spel in de volgende staat gewonnen/verloren zal zijn", dus: "In de huidige staat is er nog maar 1 muntstuk, dat ligt op het vakje waar pacman op staat en pacman gaat een stap zetten dus het spel zal in de staat gewonnen zijn." Dit is duidelijk omslachtig. Er werd gekozen om de predicaten `GameWon` en `GameLost` eenduidig te definiëren op de staat. Om dit te kunnen doen heeft de staat de predicaten `PreviousMove`, `PrevLost` en `PrevWon` nodig. `PreviousMove` is nodig om te bepalen of pacman en een geest elkaar net gekruisd zijn, `PrevWon` en `PrevLost` worden gebruikt om op te dragen dat eenmaal een spel verloren/gewonnen is, dit niet meer veranderd.

2.3 Verificatie

De eerste twee verificaties zijn voorwaarden waar altijd, in alle mogelijke modellen, aan voldaan moet zijn. De manier om dit na te gaan is contradictie: de omgekeerde voorwaarde toevoegen. Als de theorie met deze bijkomende voorwaarde satisfieerbaar is, betekend dat dat aan de originele voorwaarde niet voldaan werd. Als er zo geen modellen bestaan dan is het wel correct.

De laatste twee verificaties stellen dat er een model moet bestaan waarin aan een voorwaarde voldaan is. Om dit te controleren wordt de voorwaarde toegevoegd en nagegaan of er een model voor bestaat. Het verschil tussen de eerste twee verificaties en de laatste twee is dat het bij de eerste in alle modellen voldaan moet zijn en in de laatste in minimum 1 model.

De eerste verificatie (het aantal muntstukken op het bord kan enkel omlaag gaan) gebeurt door de voorwaarde toe te voegen dat er op een tijdstip geen goud ligt op een vakje maar op een later tijdstip, op dat zelfde vakje, plots wel. Dit werd verkozen boven de letterlijke vertaling van de opgave omdat het strikter (muntstukken kunnen ook niet van vakje veranderen) en performanter is. Deze striktere voorwaarde impliceert wel dat het aantal muntstukken op het bord enkel omlaag kan gaan.

3 NuSMV

3.1 Modellering

3.1.1 Direction en bewegen

Pacman en ghost bevinden zich in een coördinaten systeem. Bij elke stap verandert of x of y met 1 maar ter beperking van het aantal mogelijke toestanden, is 4 de maximale waarde voor beide. Dit zou zonder problemen naar een willekeurig getal groter dan 4 verhoogd kunnen worden.

Om te voorkomen dat ghost rechtsomkeer maakt, en om muren ondoordringbaar te maken, werd aan `agent` instanties een variabele `direction` toegevoegd. In combinatie met de huidige coördinaten bepaalt deze volledig en deterministisch de volgende positie van de `agent`. Daarom kan de volgende `direction` van een `agent` die ghost is, niet het tegengestelde van de huidige richting zijn, vandaar de parameter `isGhost`.

Op een gelijkaardige manier werden in de module `square` invarianten toegevoegd die de volgende richting beperken tot deze waarin geen muren staan wanneer respectievelijk pacman of ghost zich op dat vakje bevinden. Ook dit verklaart waarom een `square` bij creatie verwijzingen krijgt naar pacman en ghost.

3.1.2 Coins

De variabele `contains_coin` bepaalt of er op een `square` nog goud ligt. Bij het begin van het spel ligt overal behalve waar pacman start goud. In elke volgende toestand verdwijnt het goud op een vakje als pacman tijdens die beurt daar zal staan. Daarnaast verandert de variabele `contains_coin` niet.

3.1.3 GameState

`gameState` Kan drie waarden aannemen: *playing*, WIN, $LOSE$. Vanaf de start is `gameState` *playing*. In de toestanden die daarop volgen wordt telkens eerst gecontroleerd of de speler zal verliezen. Zo niet, kijkt de implementatie of alle goudstukken zullen verdwenen zijn en hij zal winnen. Wanneer ook dit niet het geval is, blijft de huidige `gameState` behouden.

De speler verliest niet alleen wanneer pacman een vakje deelt met ghost, maar ook als pacman en ghost elkaar kruisen. Zo is het spel in de volgende toestand verloren wanneer pacman en ghost naast elkaar staan, enkel hun x of y coördinaat verschilt met 1, en ze van plan zijn naar elkaar te bewegen.

3.2 Verificatie

De eerste verificatie vraagt dat het spel vanuit elke begintoestand gewonnen of verloren kan worden. Omdat niet volledig duidelijk is wat de mogelijke begintoestanden zijn, worden deze geïnterpreteerd als alle toestanden. Er bestaat altijd een toekomstige toestand waarin de speler verliest. Bij het winnen moet de extra voorwaarde opgelegd worden dat hij niet in de volgende beurt verliest. Het is namelijk zo dat de keuze van `direction` al deterministisch bepaalt of dat zal gebeuren.

Wanneer op het eerste vakje geen muntje ligt, zal er daar in alle volgende toestanden daar geen munt liggen. Omdat vakjes steeds zeer algemeen beschreven werden, is dit voldoende om hetzelfde voor alle vakjes te garanderen.

Om oneindig vaak op hetzelfde vakje te komen, moet er een uitvoering van het spel zijn waarin er voor elke toestand nog een latere toestand volgt zodat pacman opnieuw dat vakje bereikt.

Op een gelijkaardige manier impliceren oneindig veel bezoeken aan twee verschillende vakjes, dat het spel in elke toestand niet gewonnen is.

Tot slot zijn er minstens 10 tijdstappen nodig als het spel in elke 9^{de} toestand nog niet gewonnen is.

4 Tijdsbesteding

Jessa:	IDP	$10\,u$
	NuSMV	$2\,u$
Karel:	IDP	$4\,u$
	NuSMV	$7,5\,u$

5 Besluit