Project Formele Systeemmodellering voor Software

Bruno Corijn, Jasper Vander Jeugt, Toon Willems

5 januari 2013

1 Deel 1: ACL2

Voor de priority-queue zijn er drie versies terug te vinden. Oorspronkelijk waren we begonnen met de queue te implementeren als een binomial heap. Dit heeft als voordeel dat het veel sneller is dan de andere alternatieven. Het grote nadeel was echter de moeilijke formele verificatie van deze datastructuur.

Hierna hebben we geprobeerd de queue te implementeren als eenvoudige linked list. De eigenschappen hiervan zijn min of meer tegenovergesteld aan die van de binomial heap: de correctheid was makkelijker te bewijzen maar het was echter ook zeer inefficiënt voor de gebruiker, waardoor we ook dit niet verder hebben uitgewerkt.

Uiteindelijk hebben we gekozen voor een binaire boom. In het slechtste geval zal ook deze datastructuur traag werken, gezien er niet wordt geherbalanceerd maar gemiddeld zal deze implementatie sneller zijn dan de linked list.

De finale implementatie kan teruggevonden worden in `binary-tree.lisp`. De twee (onafgewerkte implementaties) bevinden zich in `linked-list.lisp` en `binomial-heap.lisp`.

1.1 Formele Specificaties

De specificaties zijn terug te vinden als theorems, die gebruik maken van enkele hulp functies. Deze functies controleren de volgende zaken: of alle elementen in de queue een kleinere prioriteit hebben dan een gegeven x, of alle elementen in de queue een grotere of gelijke prioriteit hebben aan een gegeven x, of er een element met prioriteit k en value v terug te vinden is in een gegeven queue en of de ordening van de gehele boom correct is.

Bij de theoremaś specifiëren we het gedrag van de priority queue en controleren we dat de correctheid bewaard blijft. In de code is er bij elk theorema commentaar voorzien zodat het duidelijk zou moeten zijn waartoe het onderstaand theorema dient.

1.2 Implementatie

De belangrijkste functies uit de API zijn *queue-insert (k v queue)*, die recursief een nieuwe boom zal opbouwen waar de value v met prioriteit k aan toegevoegd is. Indien de queue leeg is, wordt er een nieuw singleton aangemaakt.

Queue-find-min(queue) zal het element met de kleinste prioriteit teruggeven, wat neerkomt op recursief het meest linkse kind in de boom te zoeken. De *queue-delete-min(queue)* functie zal dit element verwijderen uit de boom door een nieuwe queue op te bouwen waar dit element niet meer in te vinden is.

Queue-merge (q1 q2) zal zoals de naam laat vermoeden, 2 queues mergen. Hiervoor voegen we eerst het rechterkind van *q1* aan *q2* toe aan de hand van de *queue-insert* functie, vervolgens het linkerkind en als laatste het element zelf.

Als laatste functionaliteit hebben we *queue-change-priority(k v queue)*, dit zal een nieuwe lijst opbouwen waarbij alle elementen van *queue* gekopieerd worden buiten het element met value v, waar we de nieuwe prioriteit k aan zullen toekennen.

1.3 Verificatie

De output van de automatische verificatie van de implementatie als binaire boom is te vinden als bijlage aan ons verslag.

2 Deel2: TLA/TLC