De modellering van queries

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 25-05-18 | Versie: 1 |

# Introductie

Een Query wordt in Perspectives geschreven als een *point free* function. We onderscheiden Rol- en PropertyQueries. Hun definitie is gekoppeld aan respectievelijk een Context en Rol[[1]](#footnote-1). Deze types zijn tegelijkertijd het type van het argument van de Query.

De functie is ofwel elementair, ofwel samengesteld.

## Elementaire functies

We onderscheiden twee groepen elementaire functies:

1. dataTypeGetters
2. modelBasedGetters (Rol- en Property getters).

DataTypeGetters zijn selectoren op de primaire representatie van ContextInstanties en RolInstanties, zoals: binding, context, label, binnenRol, enzovoort.

ModelBasedGetters selecteren een gemodelleerde Rol of Property. ‘Gemodelleerd’ wil zeggen dat er een model is die een beschrijving van zo’n Rol of Property bevat.

## Samengestelde functies

We stellen functies samen uit andere functies met combinatoren. We onderscheiden:

1. Unaire combinatoren (zoals laatste en notEmpty)
2. nAire combinatoren (er zijn er twee: compose en concat)
3. een serie speciale gevallen, namelijk: filter, rolesOf, contains, constant, variable, en setVariable.

## Perspectives.QueryFunctionDescriptionCompiler

Deze module verwerkt een Abstract Syntax Tree (gerepresenteerd met de Purescript types QueryStep en ElementaryQueryStep) tot een beschrijving van een functionele expressie in termen van Contexten en Rollen.

Deze beschrijving wordt door de module Perspectives.QueryCompiler als input genomen om er een concrete Purescript functie van te maken.

De QueryFunctionDescriptionCompiler controleert of de verschillende stappen in de query wel goed op elkaar aansluiten.

# Modellen en definities

Een model is een verzameling definities (we gebruiken definitie als term voor de beschrijving van een type). Een model moet voldoen aan regels die gecontroleerd worden door de Type Definition Checker (Perspectives.TypeDefChecker).

Een ContextInstantie moet een type hebben. Een ContextInstantie moet minstens een BinnenRol en een BuitenRol hebben. De TypeDefChecker controleert onder meer of de ContextInstantie de verplichte interne en externe properties heeft. Daartoe raadpleegt hij de beschrijving van de BinnenRol en van de BuitenRol van het type van de instantie. Hieruit volgt, dat de beschrijving van een Context dus minstens die twee onderdelen moet hebben. Oftewel, we kunnen eenvoudig vaststellen of een ContextInstantie een Context beschrijft, of niet.

Op vergelijkbare manier kunnen we vaststellen of een ContextInstantie een Rol beschrijft: dan heeft hij namelijk een rol mogelijkeBinding.

Hoe weet de TypeDefChecker welke eisen gesteld moeten worden aan een definitie?

Welnu, die zijn in model:Perspectives vastgelegd. Dit model bevat beschrijvingen van types waarmee de modelleur vervolgens zelf zijn types mee kan beschrijven. Bovendien, model:Perspectives beschrijft óók de types waaruit model:Perspectives zèlf is opgebouwd! Het is een zichzelf beschrijvend model.

# Query functies

Het model QueryAst beschrijft types waarmee we query-expressies kunnen beschrijven. Het bevat bijvoorbeeld de types:

* DataTypeGetter
* PropertyGetter
* RolGetter
* UnaryCombinator
* nAryCombinator
* enz.

Het type DataTypeGetter heeft één (externe) Property, namelijk functionName. De QueryFunctionDescriptionCompiler instantieert DataTypeGetter met namen als “binding” en “context” om zo een ContextInstantie te maken die gecombineerd kan worden met andere functiebeschrijvingen.

DataTypeGetter en de andere bovenbeschreven types zijn ook (Purescript) functies. Als zodanig hebben ze een domein en range (meer algemeen: een typebeschrijving).

De QueryFunctionDescriptionCompiler controleert of de verschillende querystappen wel op elkaar aansluiten. Dat komt erop neer dat hij controleert of het type dat elke stap oplevert, wel als argument door de volgende stap verwerkt kan worden.

Men zou kunnen denken dat het daarbij gaat om de domeinen en ranges van DataTypeGetter, PropertyGetter, enzovoort. Maar dat is niet zo. Met elk van deze functies construeren we *functies op het gemodelleerde niveau*. Dat wil zeggen dat zo’n functie een domein heeft dat in een model is gedefinieerd, en een range heeft dat in een model is gedefinieerd. Deze types (domein en range) zijn gemodelleerde types.

De QueryFunctionDescriptionCompiler controleert op dat gemodelleerde niveau.

De consequentie daarvan is, dat DataTypeGetter & friends weliswaar functies representeren, maar dat het niet nodig is ze als functies met argumenten en een resultaat te modelleren!

1. In deze tekst bedoelen we met Context en Rol de gelijknamige *types*. Waar we instanties nodig hebben, benoemen we dat expliciet. [↑](#footnote-ref-1)