Representatie van queries

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 06-03-18 | Versie: 1 |

# Introductie

In Perspectives is een query een (Puresecript) functie die de identifier van een entiteit als argument neemt. Queries zijn vaak partiele functies over de verzameling entiteiten. Ze werken alleen voor contexten, of voor rollen. Partiele functies zijn lastig. We lossen dat probleem op, door een functie die bedoeld is voor een context, bij een rol een leeg array te laten opleveren (en idem voor een rol).

In Perspectives kennen we twee soorten queries:

* de rolquery, die als resultaat een array van identifiers van entiteiten oplevert
* de propertyquery, die als resultaat een array van simpleValues oplevert.

Er is één aantal basisoperator die op zichzelf een propertyquery is:

* label levert de displayname van een context

Verder kan voor elke rolproperty een ‘getter’ gegenereerd worden die een propertyquery is.

Er is ook een aantal basisoperatoren die op zichzelf rolqueries zijn:

1. neem de rol van een context (resultaat is alle instanties van die rol)
2. neem de binding van een rol
3. neem het type van een context
4. neem het type van een rol
5. neem alle rollen van een context.

Elk heeft een omgekeerde operator:

1. neem de context van een rolinstantie
2. neem de rollen die door een rol gevuld worden
3. neem de instanties van een contexttype
4. neem de instanties van een roltype
5. neem de context van een rolinstantie (is hetzelfde als 1).

Maar queries bestaan vooral uit composities van kleinere queries. In Perspectives ontstaan die door *functionele compositie*.

In de implementatie van Perspectives is het echter nodig om te kunnen reflecteren op queries. Meer in het bijzonder dienen we de volgende vragen te kunnen beantwoorden:

* wat is de eerste stap van een query?
* wat is de rest de stappen van een query?
* heeft de client volledige beschikking over informatie benodigd voor een query stap?
* wat is de omkering van een querystap?

Voor reflectie hebben we een andere representatie dan Purescript functies nodig. Immers, het is niet mogelijk om een functionele compositie te ontleden in zijn samenstellende delen.

# Representatie in termen van contexten en rollen

Elk type basisoperator representeren we met een context. Deze context is bijzonder eenvoudig, want hij heeft geen rollen. De simpelste query is dus zo’n context. Een samengestelde query representeren we met een context met twee rollen: één voor elk onderdeel. Deze context representeert dus de *compositie* van twee queries. Compositie is een combinator van twee queries die een query als resultaat geeft.

Onderstaande figuur geeft een compositie van driemaal de basisoperator binding.

>>>

binding

>>>

binding

binding

Een andere binaire combinator is *filter*. Een filter neemt een rolquery en een criterium en levert een query.

Een *criterium* is een propertyquery met een booleaans resultaat. Dat wil zeggen, een array met daarin de string “true” of “false”. Deze resultaten ontstaan door gebruik van een test of een vergelijking.

Een *test* is een combinator die één rol- of propertyquery als argument neemt. Het enige bekende voorbeeld is hasValue, die test of een queryresultaat leeg is.

Een *vergelijking* is een combinator die twee rol- of propertyqueries als argument neemt. Een voorbeeld van een vergelijking is isSubset, die test of het resultaat van de ene query een deelverzameling is van het resultaat van de andere query.

Er zijn ook unaire combinatoren. Een voorbeeld is closure, die een rolquery als argument neemt en als resultaat de transitieve closure van die rol oplevert. De vergelijkbare functie closure’ levert alleen het resultaat van de laatste transitieve toepassing. Zo kunnen we bijvoorbeeld de bodem van de roltelescoop vinden.

# De representatie van een query als AST

We kunnen de context-en-rol representatie van een query beschouwen als de Abstract Syntax Tree van een expressie die de query geeft. Er is op het moment van schrijven van deze tekst noch een syntax, noch een parser die deze AST als resultaat levert. Uiteindelijk zal zeker aan ARC een dergelijke querysyntax worden toegevoegd. Tot die tijd kunnen we de implementatie van Perspectives aan laten sluiten op de hier beschreven representatie.

Tevens is het mogelijk om in CRL een AST rechtstreeks op te schrijven als een aantal contexten en rollen.

# Queryconstructoren

Er is een aantal functies dat een query genereert uit een non-query argument. Een goed voorbeeld is de generator voor een rolgetter, d.w.z. een query die, gegeven een context, de instanties van een rol van die context oplevert. Dit is de functie constructRolGetter.

constructRolGetter :: forall e.

RolName ->

NamedFunction (TripleGetter e)

Merk op dat constructRolGetter een String (RolName) argument neemt. Het is dus geen query combinator, maar een query constructor!

De rolgetter representeren we met een context die een interne property heeft die het RolName argument representeert. We hebben zo ook constructPublicPropertyGetter en constructPrivatePropertyGetter.

# Combinatorconstructoren

Nog een andere categorie wordt gevormd door functies die een non-query argument nemen en dan een query combinator opleveren. contains is een voorbeeld. contains neemt een entiteit identifier of simpleValue en levert een combinator die een rol- of propertyquery als argument neemt, en een criterium oplevert.

De combinatorgenerator representeren we met een context met een interne property voor de entiteit identifier of simpleValue en een rol voor het query-argument.

Voor alle duidelijkheid:

* met de combinatorconstructor contains maken we een combinator;
* die combineren we bijvoorbeeld met een rolquery;
* en dan hebben we een criterium (een propertyquery), dat we kunnen toepassen op een entiteit identifier.