Ontwerp resources

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 05-10-17 | Versie: 1 |

# Probleem met PP 1.3

In Perspectives in Purescript versie 1.3 is een Resource altijd in precies één (en altijd dezelfde) Location opgeslagen. Hieruit ontstaat een probleem met updates. De queries:

p a

q b

waar de triples <a p b> en <b q c> aanwezig zijn, leiden tot het onderstaande netwerk:

p

q

Maar als we waren begonnen met de triples <a p d> en <b q c>, dan had het netwerk er zó uitgezien:

p

q

Als we beginnen met het eerste stel triples en een mutatie doorvoeren:

set a p d

waardoor het triple <a p b> wordt vervangen door <a p d> dan krijgen we het tweede stel triples en verwachten we natuurlijk ook het tweede netwerk.

De query p >-> q a geeft precies hetzelfde netwerk op dezelfde oorspronkelijke triples.

p

q

Maar het geeft een heel ander netwerk op het tweede stel.

p

q

Merk op dat de Locations met b en c en hun verbinding hier niet voorkomen. Er is immers geen query die daarmee moet worden gerepresenteerd; niemand heeft erom gevraagd. Maar het *triple* <b q c> bestaat natuurlijk nog wel. *Hoe bepalen we bij* een update *dat dit deel van het netwerk moet worden opgeruimd?*

De mutatie zou de Location gevuld met b vullen met c. Het *update mechanisme* zou de Location waarin oorspronkelijk c zat, vullen met Nothing. *Maar dat botst met het uitgangspunt dat een Resource onlosmakelijk verbonden is met één Location!*

Deze twee kwesties belichten hetzelfde probleem van twee kanten, namelijk *de vraag of het netwerk primair de queries representeert, of de triples* (d.w.z. de simpele verbindingen tussen resources). Dit komt omdat we in de huidige implementatie geen onderscheid maken tussen

* een Location als resultaat van een (functionele) query en
* een Location als representatie van een Resource.

In de eerste interpretatie is er geen bezwaar tegen als het update mechanisme een Location vult met een andere resource. Immers, een query wordt opnieuw berekend en krijgt een ander resultaat, dat we in die Location vastleggen. Geheel volgens het idee van dependency tracking, updates en herberekening.

In de tweede interpretatie kunnen we dat natuurlijk niet doen! Een Resource is vastgelegd in een Location en die is verbonden met andere Locations die propertywaarden van die Resource representeren (mits daar een keer om is gevraagd). Dat is onveranderlijk, tenzij een triple wordt veranderd of verwijderd.

Een *mutatie* verandert de triples, maar het *update mechanisme* mag dat natuurlijk niet doen.

# Scheiding Resources en queries

Ik los dit op door Resources en triples niet te representeren in het netwerk van locations. Een Resource wordt een datatype dat de triples representeert waarin die Resource als subject voorkomt. Ik zal dit zo inrichten dat de representatie *just in time* uitgebreid wordt, d.w.z. als een property van een Resource opgevraagd wordt, lezen we de PropDefs in en breiden de datastructuur uit. Aansluitend bouwen we het netwerk van Locations uit door de propertywaarde op te nemen in een Location en die te verbinden met andere.

Er is dan geen enkel bezwaar meer tegen het update mechanisme. Elke Location representeert een query resultaat in plaats van een (deel van een) onderliggend triple.

# StartLocations

Om een query uit te voeren op een Resource r, moet r in een Location worden gestopt. We bewaren een index van zulke Locations in het register StartLocations. Een StartLocation kan hergebruikt worden. Dat moet zelfs, willen we voorkomen dat we dezelfde query tweemaal uitvoeren!

In termen van StartLocations zien de voorbeelden waar we mee begonnen, er zo uit. De queries:

p a

q b

op de triples <a p b> en <b q c> leveren het volgende netwerk:

a

b

p

q

Merk op dat b tweemaal gerepresenteerd is: éénmaal als onderdeel van een query en éénmaal als start van queries. Als we nu de mutatie set a p d doorvoeren, zal in de query locatie b door d vervangen worden (rond weergegeven). Uiteraard blijft in de StartLocation b gewoon staan.

De query p >-> q a geeft, met het onderscheid tussen query- en StartLocations, het volgende netwerk:

p

q

a

q

b

copy

1

2

3

4

5

Het diagram geeft het netwerk uitgaande van de triples <a p b> en <b q c>. Merk op dat de locatie 5 gekopieerd is in de locatie 3. Dit is noodzakelijk in verband met updates, zoals het onderstaande diagram laat zien (de situatie nadat set b q e is uitgevoerd).

p

q

a

q

b

copy

1

2

3

4

5

Als echter de mutatie set a p d is uitgevoerd, krijgen we een ander soort update:

p

q

a

q

b

copy

1

2

3

4

5

Hier is de inhoud van locatie 3 overschreven met Nothing (omdat er geen triple <d q x> is) en bovendien is de copy-link verbroken. Immers, locatie 3 is niet langer afhankelijk van locatie 5. Bovendien, locatie 5 heeft geen dependencies en kan worden opgeruimd. Datzelfde geldt dan ook voor locatie 4.

Dit bijzondere gedrag vloeit voort uit de *dereferentie* van locatie 2 naar locatie 4. Uit de query-locatie (met inhoud b) wordt de startlocatie 4 gevonden, waar het tweede deel van de samengestelde query op wordt toegepast. Het resultaat daarvan kopiëren we naar locatie 3.

# Property getters

Onder een Property Getter verstaan we een functie die uit een subject voor een property een object teruggeeft, waarbij het triple <s p o> een basisfeit is (basisfeiten behoren tot de axiomas van de theorie en worden opgeslagen in de database).

Een samengestelde query (bijvoorbeeld p >-> q) heeft dezelfde signatuur als een Property Getter, maar levert niet het object dat bij het subject hoort als basisfeit.

Property Getters zijn zo geïmplementeerd, dat ze een locatie altijd derefereren.

# StartLocations voor combinatoren

De PluralToSingle compositie combinator verbindt het resultaat van een PluralGetter met dat van een SingleGetter. Het resultaat van een PluralGetter is Array Resource. Aansluitend passen we de SingleGetter toe op elke resource. Als die SingleGetter een Property Getter is, derefereert hij de resource naar zijn StartLocation. De resultaten van deze operaties worden verzameld in een Array in een Location, dat uiteindelijk gekopieerd wordt naar een Location die het resultaat is van de compositie combinator (het kopiëren is conform het kopiëren in de SingleToSingle combinator).

Deze manier van gebruik van een StartLocation maakt zo’n Location nooit afhankelijk van andere Locations. Er ontstaat dus nooit het probleem dat de inhoud van zo’n Location aangepast wordt.