De anatomie van het type systeem

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 13-04-18 | Versie: 1 |

# Deze tekst

Perspectives heeft een systeem van types en Aspecten. Deze tekst geeft de structuur van dat systeem. De tekst is op te vatten als een samenvatting van de tekst *Types from first principles*.

Ik geef geen definitie van type of van Aspect (voor de eerste, zie de tekst *Type Checking*; voor de laatste, zie de tekst *Aspecten*). In plaats daarvan ga ik uit van de Context Role Language en laat zien hoe de verschillende soorten expressies en de definities in model:Perspectives (ons basismodel) leiden tot het type systeem.

# Expressies

## Context declaratie

In CLR kan je een context declaratie maken. Die heeft deze vorm:

<typenaam> <instantienaam>

Zo'n declaratie creëert een Context die *geïdentificeerd* wordt met de <instantienaam> en waarvan het *type* <typenaam> is[[1]](#footnote-1).

## Rolbinding

In CLR kan je een rol binden. Dat heeft deze vorm:

<rolnaam> => <rolexpressie>

Zo'n rolbinding expressie moet in de context van een context declaratie staan:

<typenaam> <instantienaam>

<rolnaam> => <rolexpressie>

Zo'n expressie creëert een Rol (instantie bij de context instantie) die als psp:type <rolnaam> heeft. Hij wordt *geïdentificeerd* met de identifier <typenaam>$<rolnaam>\_<volgnummer>[[2]](#footnote-2). Hij heeft als *binding* de waarde van <rolexpressie>.

## Rolexpressie

<rolexpressie> kan vijf vormen hebben:

1. het kan een context declaratie zijn. In dat geval is de waarde van de binding de buitenRol van de gemaakte Context;
2. het kan de naam van een context zijn. Ook dan wordt de buitenRol genomen.
3. het kan een query-declaratie zijn. Die evalueert ook naar een context, zodat ook de buitenRol weer wordt genomen. Voorbeeld:

QUERY $berekendeRol = binding binding

1. het kan een query-toepassing zijn. Die kan evalueren naar een Rol of een Context. Voorbeeld:

usr:Systeem ~> gebruiker

1. het kan () zijn, waarmee we aangeven dat de rol geen binding heeft.

# Types zijn contexten

Er zijn twee plekken waar een RolNaam (dus de naam van een type) voorkomt:

* in een rolbinding expressie, als ie wordt geïntroduceerd;
* in een query expressie, als eraan wordt gerefereerd.

We kunnen in CRL dus niet een individuele rol (instantie) bij naam noemen. Dat is volkomen in lijn met het feit dat we een individuele rol geen naam kunnen geven (dat gebeurt automatich)!

De logische consequentie is dat in een context declaratie, de typenaam **altijd** de naam van een andere context is. Dat is geheel in overeenstemming met het feit dat een type beschreven wordt met een Context en dat de <typenaam> van de context declaratie als waarde van psp:type wordt gebruikt in de gedeclareerde context.

Dit is natuurlijk niet toevallig zo. We hebben de descriptieve rijkdom van een Context nodig om de structuur van een type te kunnen beschrijven. Een Rol zou gewoonweg niet voldoen.

# De structuur van een typebeschrijving

## Contextbeschrijving

Er zijn 3 Rollen van het type psp:Context die een speciale rol spelen in CRL:

* rolInContext, te binden aan een Rol
* externalProperty, te binden aan een Property
* internalProperty, te binden aan een Property

In een context X van het type psp:Context (of die psp:Context als Aspect heeft), kunnen we deze rollen gebruiken. Dat gebruik heeft consequenties:

1. De Rol A, die gebonden is aan een rolInContext instantie van X, kan in *instanties van X* als rol (met de naam A) gebonden worden.
2. De Property P, die gebonden is aan een externalProperty instantie van X, kan in *instanties* van X als externe property (met de naam P) een waarde krijgen.
3. Mutatis mutandis voor externalProperty.

Merk op dat in deze drie gevallen de instantie van X een Context moet zijn (want alleen Contexten hebben Rollen, Interne of Externe Properties).

Dit geeft ons een operationele definitie waarmee we kunnen herkennen dat een Context *een beschrijving van een type Context is*. Als, en alléén als, we in een Context X deze drie Rollen mogen gebruiken, beschrijft die Context een Context. En dat is zo als:

1. het type van X psp:Context is, of
2. X psp:Context als Aspect heeft.

## Rolbeschrijving

De volgende twee rollen van psp:Rol spelen een even belangrijke rol in CRL:

* rolProperty, te binden aan een Property
* mogelijkeBinding, te binden aan een Type

In een context X met het type psp:Rol (of die psp:Rol als Aspect heeft), kunnen we deze rollen gebruiken. Dat gebruik heeft consequenties:

1. De Property P, die gebonden is aan een rolProperty instantie van X, kan in *instanties* van X als rolproperty een waarde krijgen. X moet dus een (instantie van een) Rol zijn!
2. Een instantie van X kan als binding slechts een waarde krijgen waarvan het type T de binding is van de rol mogelijkeBinding van X[[3]](#footnote-3).

Dit geeft ons een operationele definitie waarmee we kunnen herkennen dat een Context *een beschrijving van een type Rol is*. Als, en alléén als, we in een Context X deze twee Rollen mogen gebruiken, beschrijft die Context een Rol. En dat is zo als:

1. het type van X psp:Rol is, of
2. X psp:Rol als Aspect heeft.

## Beschrijving van simple values

Strings, numbers, booleans en dates beschrijven we respectievelijk met de Aspecten psp:String, psp:Number, psp:Boolean en psp:Date. Deze Aspecten hebben geen properties en maar één Rol, namelijk psp:Aspect en die Rol is in alle gevallen gebonden aan psp:SimpleValue.

# Type

Kunnen we in zijn algemeenheid vaststellen of een context een type beschrijft? Hierboven zagen we dat we kunnen vaststellen of een context de beschrijving is van een Context, of van een Rol.

Kunnen we een willekeurige Context gebruiken als type? Zeker wel. Beschouw de definitie van de property volgnummer:

psp:Property psp:volgNummer

extern psp:isFunctioneel = true

extern psp:isVerplicht = true

psp:range => psp:Number

We kunnen best een instantie maken die psp:Volgnummer als type heeft:

psp:volgNummer usr:X

Maar we kunnen usr:X geen property geven en geen rol. Zulke contexten zijn dus gelijk aan de lege context, de unit van ons type systeem. psp:Volgnummer is dus, als type van een Context, letterlijk nietszeggend[[4]](#footnote-4).

Verwarrend is dat psp:volgNummer wel degelijk een type beschrijft. Maar geen context- of roltype, maar, wel, een ‘volgnummer’. Een instantie van een volgnummer zou een getal zijn. De semantiek van dat getal komt tot uitdrukking in de gedrag van Perspectives.

De conclusie moet zijn dat we drie gevallen kunnen onderscheiden. Voor elke context K die we in de type positie van een typedeclaratie gebruiken, geldt dat de resulterende context

1. een Context beschrijft (dus de representatie is van een type Context)
2. een Rol beschrijft (dus de representatie is van een type Rol)
3. of de lege context beschrijft (dus het lege Type is).

In het laatste geval is X meestal geen zinnig context- of roltype[[5]](#footnote-5).

## Kind

In Purescript is een ‘kind’ het ‘type van types’. In Perspectives zouden we psp:Context volgens die redenering de kind van Contexten kunnen noemen en psp:Rol de kind van Rollen. Ik weet niet of dat helemaal terecht is. ‘Zaak’ heeft psp:Context als Aspect. ‘Zaak’ is (daarom) óók een kind. Het is mij onduidelijk wat dat betekent voor psp:Context. psp:Context is, als Aspect, óók een type class.

1. In model:Perspectives is dit psp:type. ‘psp’ is de prefix voor model:Perspectives. [↑](#footnote-ref-1)
2. Maar ik zal laten zien dat deze identifiers nooit voorkomen in CRL teksten. [↑](#footnote-ref-2)
3. De binding moet als type T hebben, of T als Aspect hebben. [↑](#footnote-ref-3)
4. We kunnen geen rolinstantie maken die psp:Volgnummer als type heeft. CRL laat ons alleen een rolnaam binden als die in het type met psp:rolInContext is gedefinieerd. [↑](#footnote-ref-4)
5. We gebruiken wel psp:ElkType om Unit in Perspectives mee aan te duiden. psp:ElkType is een lege context. [↑](#footnote-ref-5)