Typen from first principles

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 28-11-17 | Versie: 1 |

# Introductie

Deze tekst heb ik geschreven om de manier van redeneren vast te leggen waarmee uit een klein aantal beginselen het gebouw van typen en metatypen opgetrokken kan worden.

# Uitgangspunt

We willen een Perspect model in tekst kunnen opschrijven. We hebben een minimale syntax gedefinieerd waarmee we alle aspecten van een context en een rol daarin, met properties voor beiden, kunnen beschrijven. Vertrekkend uit een expressie waarin al deze aspecten zijn verwerkt, kunnen we beredeneren welke types nodig zijn en hoe we die representeren.

Het uitgangspunt is:

:Aangifte :a1

public :urgentie = 7

private :aantekening = “Dit is een hopeloos geval.”

:aangever => :Jansen

:betrouwbaarheid = 6

Deze tekst telt vijf typen expressies:

1. type declaratie: :Aangifte :a1;
2. public context property assignment: public :urgentie = 7;
3. private context property assignment: private :aantekening = “…”;
4. rolbinding: :aangever => :Jansen;
5. rol property assignment: :betrouwbaarheid = 6.

Er zijn nog enkele typen expressies die we nodig hebben. Deze komen aan bod als we het voorbeeld uitbreiden.

# Afleidingsregels

Uit de type declaratie expressie volgt natuurlijk wat een type is en wat een instantie.

Een type declaratie impliceert een type (I)

Een binding-expressie is altijd ingebed in een expressie die een Context beschrijft. Hetzelfde geldt voor een public property assignment en een private property assignment.

private, public of => impliceert een context (II)

Een rolProperty (of een query, zoals we verderop in de tekst zullen zien) impliceert een rol:

een niet-gedecoreerde property assignment of het query sleutelwoord impliceert een rol (III)

De combinatie van een type declaratie en een binding-expressie impliceert dat het type een :rolInContext rol heeft die gevuld is met de rol die gebonden wordt:

:Type <perspectName>

:role => <perspectName>

impliceert dat :Type een rol :rolInContext heeft, gevuld met :Role (IV)

**Corollary**: :rolInContext => :Role legt op dat in instanties (van :Type) :role => <perspectName> geschreven moet worden.

Dezelfde redenering kunnen we toepassen op de drie property assignment expressies. Bij elkaar gezet:

|  |  |
| --- | --- |
| Bevat de instantie-expressie: | Dan geldt voor type: |
| :Type <perspectName> | :ContextType :Type |
| :role => :Filler | :rolInContext => :Role |
| :prop = value | :rolProperty => :Prop |
| private :prop = val | :privateProperty => :Prop |
| public :prop = val | :publicProperty => :Prop |
| query <operator>+ | :Query |

Een type dat rollen, private of public properties definieert voor zijn instanties, definieert een type context. Het type van zo’n type is per constructie ContextType. (V)

Een type dat (rol)properties of een query definieert voor zijn instanties, definieert een type rol. Het type van zo’n type is per constructie RolInContextType. (VI)

# Afleiding van :ContextType

Het uitgangspunt was:

:Aangifte :a1

public :urgentie = 7

private :aantekening = “Dit is een hopeloos geval.”

:aangever => :Jansen

:betrouwbaarheid = 6

Hieruit kunnen we nu met de afleidingsregels het type :Aangifte afleiden.

:ContextType :Aangifte

:publicProperty => :Urgentie

:privateProperty => :Aantekening

:rolInContext => :Aangever

Daaruit volgt op dezelfde manier de definitie van :ContextType:

:ContextType :ContextType

:rolInContext => :PublicProperty

:rolInContext => :PrivateProperty

:rolInContext => :RolInContext

Merk op dat deze definitie consistent is in die zin dat de regel :rolInContext => :RolInContext volgens regel IV impliceert dat het type van :ContextType een rol in context heeft die gevuld moet worden met :RolInContext. Het type van :ContextType is :ContextType en die heeft precies zo’n regel, namelijk diezelfde regel!

# Uitbreiding van het uitgangspunt

We willen ook nog een en ander zeggen over de diverse properties, namelijk:

* is een property functioneel?
* is een property verplicht?
* wat is de range van een property?

Nemen we de publieke contextproperty urgentie als voorbeeld:

:Property :Urgentie

public :isFunctioneel = true

public :isVerplicht = false

public :range = Number

Waarom :Property? Wel, :Urgentie moet een type hebben en dit is een redelijke naam. Waarom public properties? De alternatieven stuiten op problemen: private is onzinnig op dit niveau (private voor wie?) en als we een rolproperty zouden definiëren, volgt daaruit onmiddellijk dat :Property zelf een rol moet zijn – maar waar is het dan een rol van? Kortom, :Property moet een context zijn – en dan moeten zijn properties publiek zijn.

Moeten we :Urgentie niet als :PublicProperty definiëren? Nee. :Urgentie speelt de rol :publicProperty bij :Aangifte en daarmee is al vastgelegd om wat voor property het gaat.

# Afleiding van :Property, :IsFunctioneel, :IsVerplicht en :Range

Uit de uitbreiding van het uitgangspunt:

:Property :Urgentie

public :isFunctioneel = true

public :isVerplicht = false

public :range = Number

volgt één op één de definitie van :Property:

:ContextType :Property

:publicProperty => :IsFunctioneel

:publicProperty => :IsVerplicht

:publicProperty => :Range

Merk op dat deze definitie volledig consistent is met de definitie van :ContextType (want die heeft een rol in context gevuld met :PublicProperty).

:IsFunctioneel, :isVerplicht en :Range volgen nu. We willen namelijk ook in staat zijn om aan te geven dat het in Perspect verplicht is om deze drie properties toe te passen; dat ze alle drie functioneel zijn; en wat hun range is.

:Property :IsFunctioneel

public :isFunctioneel = true

public :isVerplicht = true

public :range = Boolean

:Property :IsVerplicht

public :isFunctioneel = true

public :isVerplicht = true

public :range = Boolean

Merk het gebruik van het sleutelwoord Boolean op. Dit is een terminal van de taal van Perspect. :Range heeft een interessante :range:

:Property :Range

public :isFunctioneel = true

public :isVerplicht = true

public :range = PerspectType

Ook het sleutelwoord PerspectType is een terminal. Het systeem weet hoe deze indicatie gebruikt kan worden om te controleren of de gebruiker wel een toegelaten waarde achter :range = invult.

# :RolInContext

Het voorbeeld laat structuur zien van de rol :Aangever:

:Aangifte :a1

…

:aangever => :Jansen

:betrouwbaarheid = 6

Bovendien willen we kunnen beperken waar :Aangever aan gebonden kan worden. Laten we zeggen dat we die bindingen willen beperken tot instanties van het type :Persoon, of rollen die daarmee gevuld zijn.

:RolInContext :Aangever

:rolProperty => :Betrouwbaarheid

:mogelijkeBinding => :Persoon

Bij de definitie van :ContextType ontstond :RolInContext. Dit is natuurlijk de aangewezen naam voor het type van :Aangever, al hebben we geen afleidingsregel waar dit uit volgt. We laten het type rol en het type van :Aangever dus samenvallen. :rolProperty volgt rechtstreeks uit de afleidingsregels.

Nu kunnen we :RolInContext afleiden:

:ContextType :RolInContext

:rolInContext => :RolProperty

:rolInContext => :MogelijkeBinding

Later zullen we :RolInContext verder uitbreiden.

Kunnen we iets zeggen over :MogelijkeBinding? Eigenlijk niets:

:ContextType :MogelijkeBinding

is voldoende, zoals bij :PublicProperty en :PrivateProperty.

# :PrivateProperty, :PublicProperty en :RolProperty

Ik heb de behandeling van deze drie uitgesteld. Deze types ontstonden bij de beschrijving van :ContextType en :RolInContext. Maar er valt eigenlijk niets over hen te zeggen. Bij verstek maak ik ze tot context typen, maar eigenlijk kan je alles kiezen:

:ContextType :PrivateProperty

:ContextType :PublicProperty

:ContextType :RolProperty

# Uitbreiding van het uitgangspunt

Bij een rol hoort een property-perspectief. Nu we een definitie hebben van :Aangever, kunnen we er een property-perspectief aan toevoegen:

:RolInContext :Aangever

:rolProperty => :Betrouwbaarheid

:mogelijkeBinding => :Persoon

:view => :AangeverBetrouwbaarheidsPerspectief

We moeten nu de definitie van :RolInContext uitbreiden:

:ContextType :RolInContext

:rolInContext => :RolProperty

:rolInContext => :MogelijkeBinding

:rolInContext => :view

# Afleiding van :view

Hoe beschrijven we het :AangeverBetrouwbaarheidsPerspectief? Als volgt:

:view :AangeverBetrouwbaarheidsPerspectief

:propertyReferentie => :Betrouwbaarheid

:volgNummer = 1

En hieruit volgt de definitie van :view zelf:

:ContextType :view

:rolInContext => :PropertyReferentie

en dan de rol :PropertyReferentie en :VolgNummer:

:RolInContext :PropertyReferentie

:rolProperty => :VolgNummer

:mogelijkeBinding => :Property

:Property :VolgNummer

public :isFunctioneel = true

public :isVerplicht = true

public :range = Number

# Cardinaliteit van rollen

Een aangifte kan door meerdere aangevers gedaan worden. Oftewel: net zoals bij een property willen we van een rol kunnen aangeven wat zijn cardinaliteit is. Breiden we het voorbeeld van :Aangever uit:

:RolInContext :Aangever

public :isVerplicht = true

public :isFunctioneel = false

:rolProperty => :Betrouwbaarheid

:mogelijkeBinding => :Persoon

Dienovereenkomstig breiden we :RolInContext nogmaals uit:

:ContextType :RolInContext

:rolInContext => :RolProperty

:rolInContext => :MogelijkeBinding

:rolInContext => :view

:publicProperty => :IsVerplicht

:publicProperty => :IsFunctioneel

:ContextType ondersteunt reeds :publicProperty, dus die hoeven we niet uit te breiden.

# Query

Een rol in een context kan ook berekend worden. De berekening noemen we een query. Een query introduceren we met het sleutelwoord query. Bijvoorbeeld:

:RolInContext :MijnBerekendeRol

query :aangever binding

Achter het sleutelwoord query volgt een reeks operatoren. Rol- en propertynamen (beginnend met een kleine letter) zijn operatoren. binding is een operator. Daarnaast is er een klein repertoire aan operatoren om operatoren te combineren, zoals filter. Queries worden elders uitgebreider beschreven. We moeten nu nogmaals :RolInContext uitbreiden:

:ContextType :RolInContext

:rolInContext => :RolProperty

:rolInContext => :MogelijkeBinding

:rolInContext => :view

:publicProperty => :IsVerplicht

:publicProperty => :IsFunctioneel

:Query

:Query zelf is een verder leeg :ContextType:

:ContextType :Query

# Inline type declaratie bij rolbinding

Een type declaratie kan opgenomen worden na de binding-pijl. Op die manier kunnen we rollen en properties, die immers precies in respectievelijk één context en één rol voorkomen, inline opnemen.

:ContextType :view

:rolInContext =>

:RolInContext :PropertyReferentie

:rolProperty => :VolgNummer

:mogelijkeBinding => :Property

# Syntactische ontleding en semantische constraints

Uit de definities (zie Appendix 2) volgen restricties. De rol :rolProperty mag bijvoorbeeld alleen gebonden worden (:rolProperty => :Betrouwbaarheid) in de beschrijving van een rol. Dit blijkt uit regel IV, die van toepassing is op deze situatie:

:Type <perspectName>

:role => <perspectName>

Regel IV stelt dat :Type een rol :rolInContext heeft, gevuld met :Role. :Role is hier :RolProperty en het enige type waarin de expressie :rolInContext :RolProperty voorkomt is :RolInContext. En alleen dat type beschrijft een rol.

We noemen dit een *semantische constraint*. Semantische constraints volgen dus uit de toepassing van de afleidingsregels op types. Welke types? Alle types! Er is wat dit betreft geen verschil tussen de types in Appendix 2 en types die de modelleur aanmaakt.

Semantisch constraints moeten we onderscheiden van syntactische constraints. Deze laatste volgen uit de grammatica voor correcte Perspect expressies. Dit zijn de enige correcte expressies:

1. type declaratie: :Aangifte :a1;
2. public context property assignment: public :urgentie = 7;
3. private context property assignment: private :aantekening = “…”;
4. rolbinding: :aangever => :Jansen;
5. rol property assignment: :betrouwbaarheid = 6.
6. query: query :aangever binding.
7. inline type declaration: :role => :Type <perspectName>....

Er is (syntactische) samenhang tussen deze expressies, die alleen blijkt uit de grammatica zelf (zie hieronder).

De ontleder hanteert alleen de grammatica en geeft aan of de tekst voldoet aan de syntactische constraints. Bovendien levert de ontleder een Abstract Syntax Tree (AST). Deze AST is louter opgesteld in termen van contexten en rollen.

De grammatica luidt:

expression = context | role

context = typeDeclaration

publicContextPropertyAssignment\*

privateContextPropertyAssignment\*

rolBinding\*

role = typeDeclaration

rolePropertyAssignment\*

query?

typeDeclaration = perspectName perspectName

publicContextPropertyAssignment = 'public' propertyName '=' simpleValue

privateContextPropertyAssignment = 'private' propertyName '=' simpleValue

rolBinding = roleName '=>' (perspectName | expression)

rolePropertyAssignment\*

rolePropertyAssignment = propertyName '=' simpleValue

query = 'query' operator+

roleName = lowerCaseChar anyChar\*

propertyName = lowerCaseChar anyChar\*

perspectName = upperCaseChar anyChar\*

Uit regel II volgt de definitie van context: iets is een context als het een publieke of private context property heeft, of als het een rolbinding heeft. Op dezelfde manier volgt de definitie van rol uit regel IV: iets is een rol, als het een rolproperty of een query heeft.

Contexten en rollen samen noemen we expressie[[1]](#footnote-1). Een rolBinding is op een perspectName, of op een expression (inline type declaration). De andere grammatica regels volgen direct uit het voorbeeld dat we als uitgangspunt namen.

De ontleder levert dus een serie contexten en rollen als de ontlede tekst grammaticaal correct is. In zo’n tekst zou een rolproperty als onderdeel van een context definitie gebonden kunnen worden (en daarmee de constraint geschonden worden die we in het begin van dit hoofdstuk beschreven), zonder dat de ontleder er bezwaar tegen maakt.

De semantische controle toont zo’n geschonden constraint aan (b.v. door in de editor de betreffende expressie te markeren). Of het systeem probeert te voorkomen dat de modelleur zo’n constraint schendt.

# Appendix 1. Integraal voorbeeld

:ContextType :Aangifte

:publicProperty =>

:Property :Urgentie

public :isFunctioneel = true

public :isVerplicht = false

public :range = Number

:privateProperty =>

:Property : Aantekening

public :isFunctioneel = true

public :isVerplicht = false

public :range = String

:rolInContext =>

:RolInContext :Aangever

public :isVerplicht = true

public :isFunctioneel = false

:rolProperty =>

:Property : Betrouwbaarheid

public :isFunctioneel = true

public :isVerplicht = false

public :range = Number

:mogelijkeBinding => :Persoon

:view =>

:view :AangeverBetrouw…Perspectief

:propertyReferentie => :Betrouwbaarheid

:volgNummer = 1

:rolInContext =>

:RolInContext :MijnBerekendeRol

query :aangever binding

# Appendix 2. Alle types

:ContextType :ContextType

:rolInContext => :PublicProperty

:rolInContext => :PrivateProperty

:rolInContext => :RolInContext

:ContextType :RolInContext

:rolInContext => :RolProperty

:rolInContext => :MogelijkeBinding

:rolInContext => :view

:publicProperty => :IsVerplicht

:publicProperty => :IsFunctioneel

:Query

:ContextType :Property

:publicProperty => :IsFunctioneel

:publicProperty => :IsVerplicht

:publicProperty => :Range

:Property :IsFunctioneel

public :isFunctioneel = true

public :isVerplicht = true

public :range = Boolean

:Property :IsVerplicht

public :isFunctioneel = true

public :isVerplicht = true

public :range = Boolean

:Property :Range

public :isFunctioneel = true

public :isVerplicht = true

public :range = PerspectType

:ContextType :PrivateProperty

:ContextType :PublicProperty

:ContextType :RolProperty

:ContextType :Query

:ContextType :MogelijkeBinding

:ContextType :view

:rolInContext => :PropertyReferentie

:RolInContext :PropertyReferentie

:rolProperty => :VolgNummer

:mogelijkeBinding => :Property

:Property :VolgNummer

public :isFunctioneel = true

public :isVerplicht = true

public :range = Number

1. NB. De purescript implementatie constateert een lus in de definities van rolBinding, expression en context. Daarom heb ik expression niet geïmplementeerd. [↑](#footnote-ref-1)