# **Proposal:**

**Summary**

* Editorial clean-up
* Remove quotes from context names
* Content of Figure 10.16 FEEL latice type
* Grammar change to support more types in instanceof

**Details**

The changes are baselined on DMN 1.2/Beta1 dtc-18-06-01.pdf.

**Position:**

**Page 116 Paragraph 10.3.2.9**

**Replace reference**

(see Figure <\*\*\*> )

**with**

(see Figure 10.16)

**Position:**

**Page 117 Table 39**

**Replace table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **e** | **type(e)** | | *123* | number | | *True* | boolean | | *"abc"* | string | | *date("2017-01-01")* | date | | *["a", "b", "c"]* | list<string> | | *["a", true, 123]* | list<Any> | | *{"name": "Peter", "age": 30}* | context<”age”:number, “name”:string> | | *function f(x: number, y: number) x + y* | (number, number) → number | | DecisionA  where the *typeRef* of *DecisionA* is  <itemDefinition name="Employee">  <itemComponent name="id">  <typeRef>number</typeRef>  </itemComponent>  <itemComponentname="name">  <typeRef>string</typeRef>  </itemComponent>  </itemDefinition> | context<”id”:number, “name”:string> | | BkmA  where the encapsulated logic is  <encapsulatedLogic>  <formalParameter name="x" typeRef="number"/>  <formalParameter name="y" typeRef="number"/>  <literalExpression typeRef="number">  <text>x + y</text>  </literalExpression>  </encapsulatedLogic> | (number, number) → number | |  |

**with table**

|  |  |
| --- | --- |
| **e** | **type(e)** |
| *123* | number |
| *true* | boolean |
| *"abc"* | string |
| *date("2017-01-01")* | date |
| *["a", "b", "c"]* | list<string> |
| *["a", true, 123]* | list<Any> |
| *{name: "Peter", age: 30}* | context<age:number, name:string> |
| *function f(x: number, y: number) x + y* | (number, number) → number |
| DecisionA  where the *typeRef* of *DecisionA* is  <itemDefinition name="Employee">  <itemComponent name="id">  <typeRef>number</typeRef>  </itemComponent>  <itemComponentname="name">  <typeRef>string</typeRef>  </itemComponent>  </itemDefinition> | context<id:number, name:string> |
| BkmA  where the encapsulated logic is  <encapsulatedLogic>  <formalParameter name="x" typeRef="number"/>  <formalParameter name="y" typeRef="number"/>  <literalExpression typeRef="number">  <text>x + y</text>  </literalExpression>  </encapsulatedLogic> | (number, number) → number |

|  |  |
| --- | --- |
|  |  |

**Position:**

**Page 118 Paragraph 10.3.2.9.1**

**Replace paragraph (remove quotes from names)**

* Two context types *context<”k1”: T1, …, ”kn”: Tn>* and *context<”l1”: S1, …, ”lm”: Sm>* are equivalent iff n = m and for every *ki* *:Ti* there is a unique *lj* *:Sj* such that *ki = lj* and *Ti* ≡ *Sj* for i = 1, n. Context types are the types defined via ItemDefinitions or the types associated to FEEL context literals such as { “name”: “John”, “age”: 25}.

**with**

* Two context types *context<k1: T1, …, kn: Tn>* and *context<l1: S1, …, lm: Sm>* are equivalent iff n = m and for every *ki* *:Ti* there is a unique *lj* *:Sj* such that *ki = lj* and *Ti* ≡ *Sj* for i = 1, n. Context types are the types defined via ItemDefinitions or the types associated to FEEL context literals such as { name: “John”, age: 25}.

**Position:**

**Page 118 Paragraph 10.3.2.9.2**

**Replace paragraph (remove quotes from names)**

* Two context types *context<”k1”: T1, …, ”kn”: Tn>* and *context<”l1”: S1, …, ”lm”: Sm>* are equivalent iff n = m and for every *ki* *:Ti* there is a unique *lj* *:Sj* such that *ki = lj* and *Ti* ≡ *Sj* for i = 1, n. Context types are the types defined via ItemDefinitions or the types associated to FEEL context literals such as { “name”: “John”, “age”: 25}.

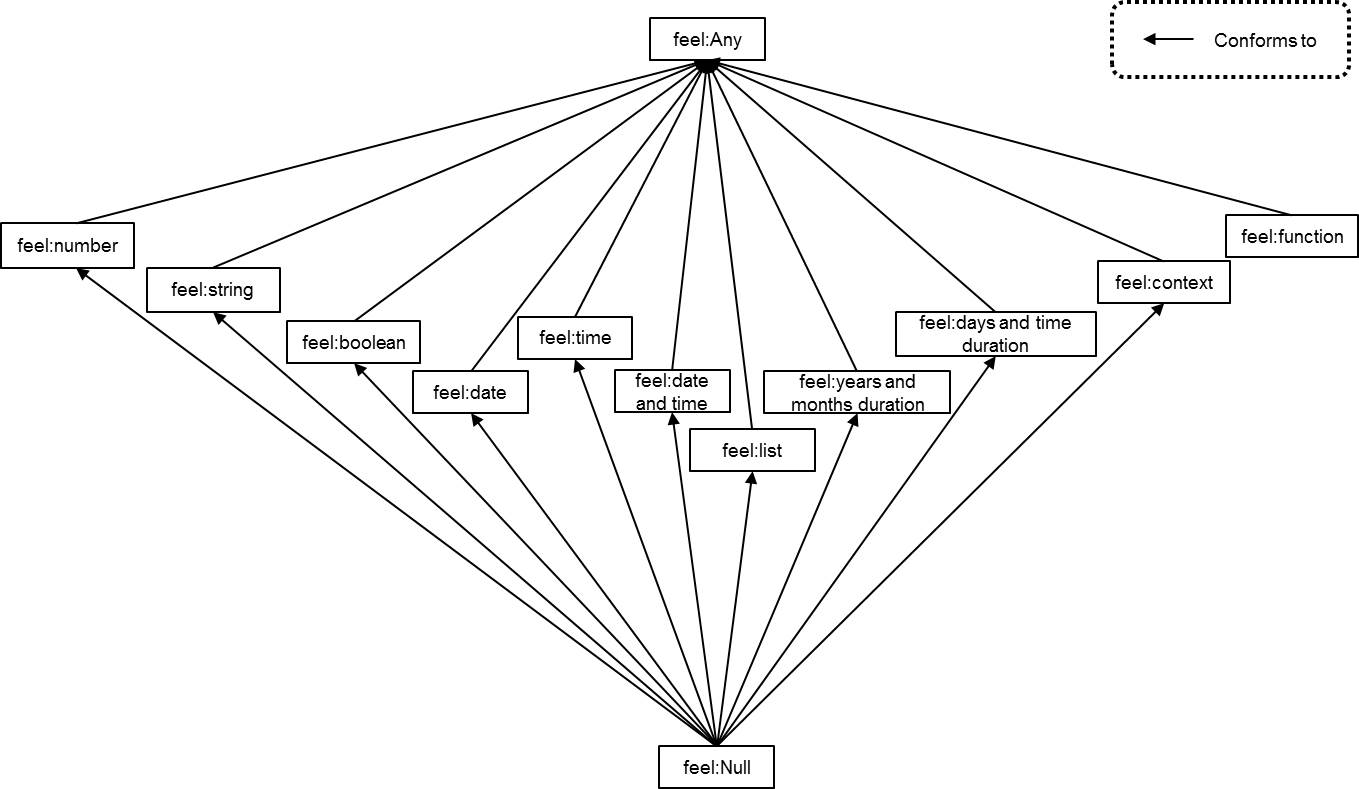
**with**

* Two context types *context<k1: T1, …, kn: Tn>* and *context<l1: S1, …, lm: Sm>* are equivalent iff n = m and for every *ki* *:Ti* there is a unique *lj* *:Sj* such that *ki = lj* and *Ti* ≡ *Sj* for i = 1, n. Context types are the types defined via ItemDefinitions or the types associated to FEEL context literals such as { name: “John”, age: 25}.

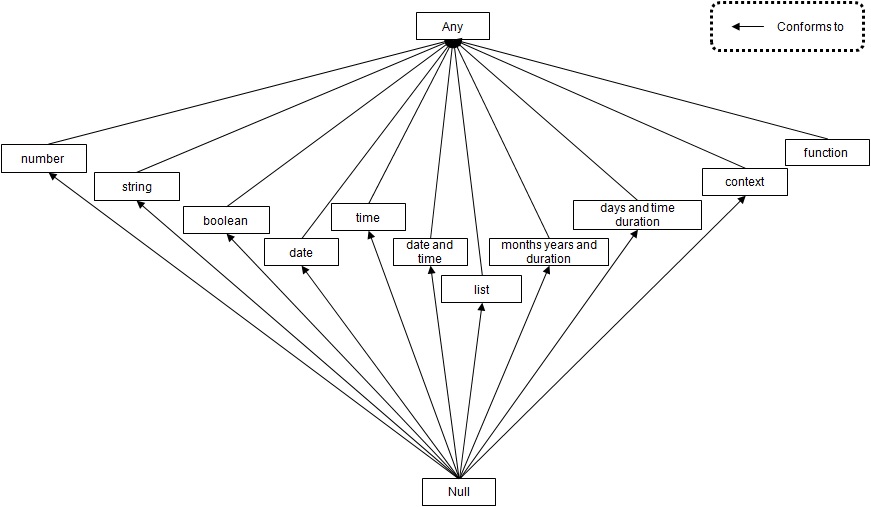
**Position:**

**Page 118 Figure 10.16**

**Replace content of figure**

****

**with**

****

**Position:**

**Page 134**

**Replace paragraph**

Type-checking is defined in Table 56. Note that *type* is not mapped to the domain, and *null* is not the name of a type, and **null** is not an instance of any type.

**with**

Type-checking is defined in Table 56. Note that *type* is not mapped to the domain, and **null** is the only value in the Null type (see 10.3.2.1).

**Position:**

**Page 111**

**Replace production**

54. type = qualified name ;

**with**

54. type =

qualified name |

'list' '<' type '>' |

'context' '<' name ':' type { ',' name ':' type } '>' |

'function' '<' [ type { ', ' type } ] '>' '->' type

;