Isabelle/DOF
A Framework for Proving Ontology-Relations and Runtime Testing
Ontology Instances

Idir Ait-Sadoune, Nicolas Meric and Burkhart Wolff
LMF, Université Paris-Saclay, France

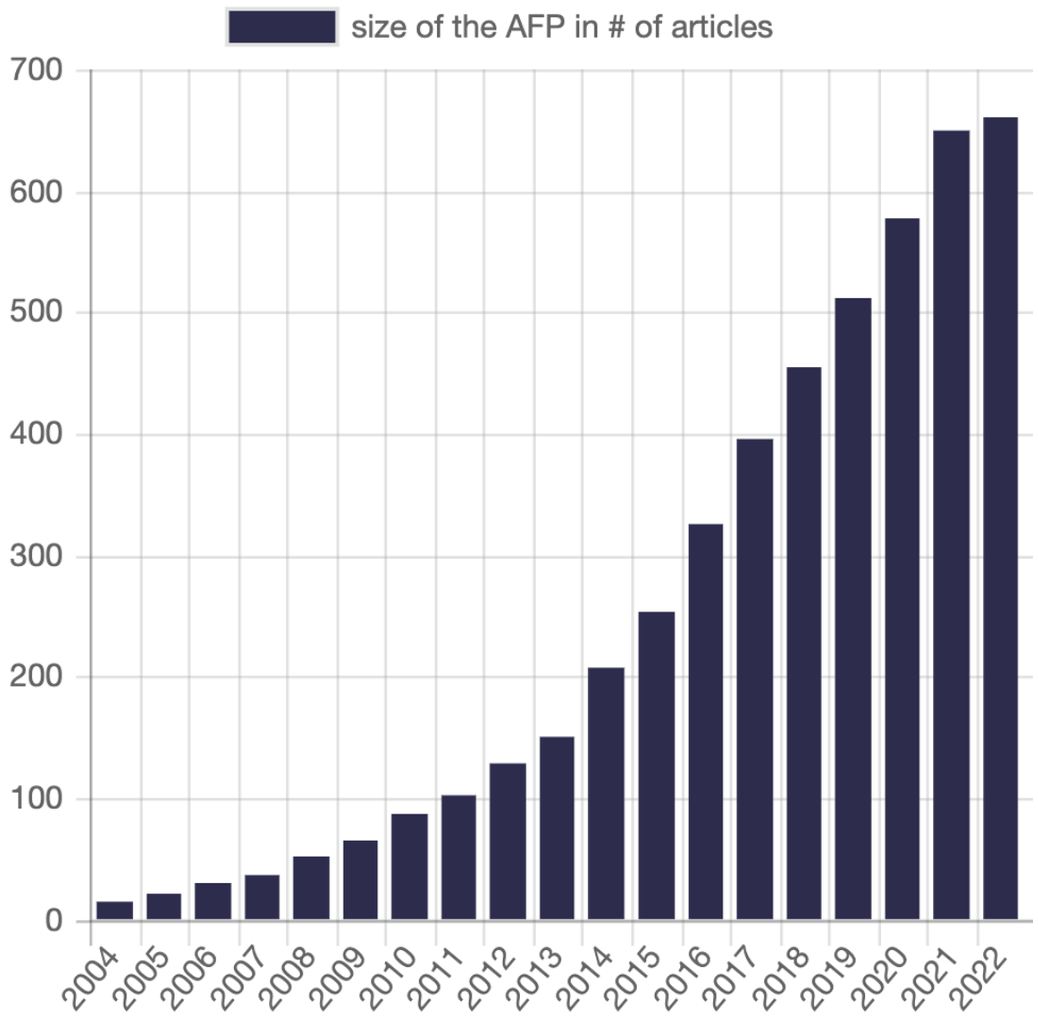
GT Deduction 17.2.2022

Overview

- Why (Document) Ontologies
- Ontologies and Formal Theories
- DOF Design
- Isabelle/DOF Implementation
- Some Application Scenarios

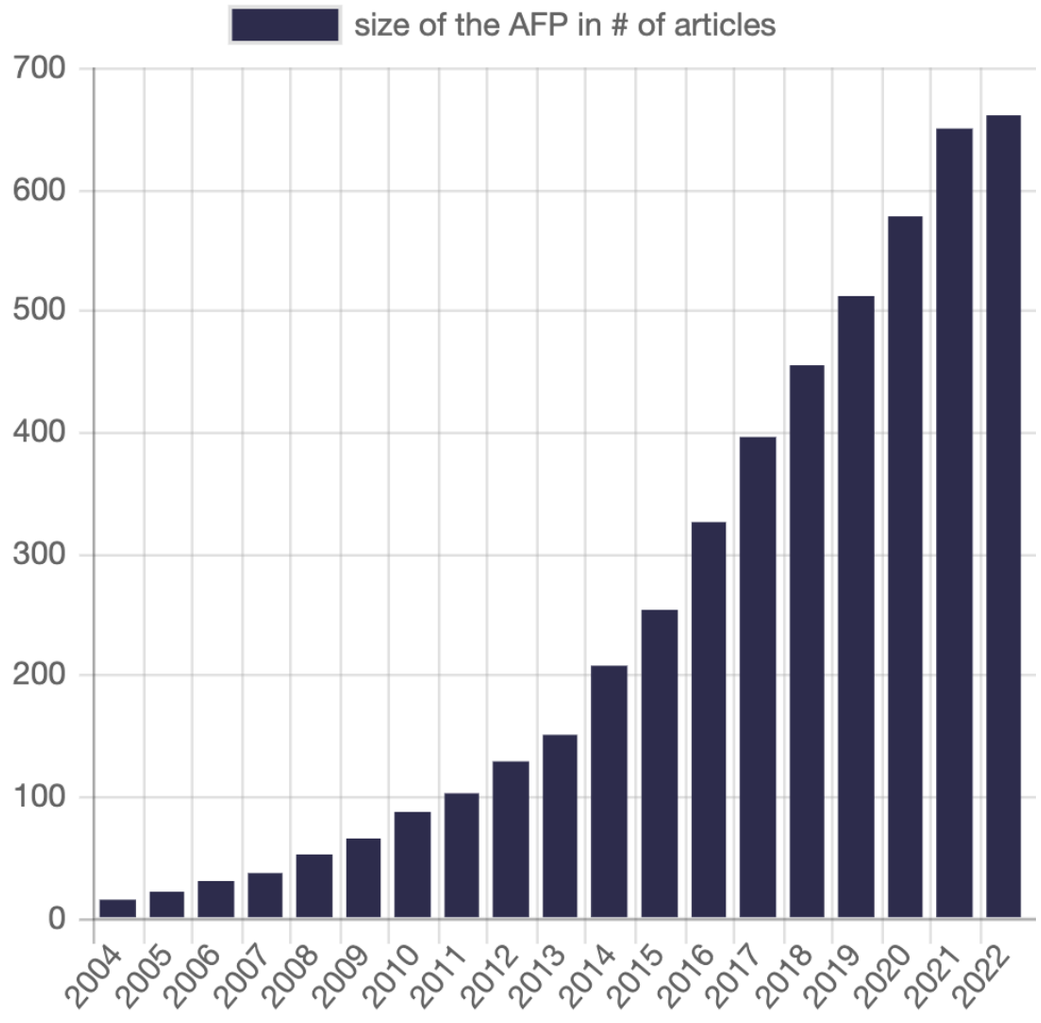
PUBLIC RELEASE:
http/10.5281/zenodo.3370483

Why Ontologies



Why Ontologies

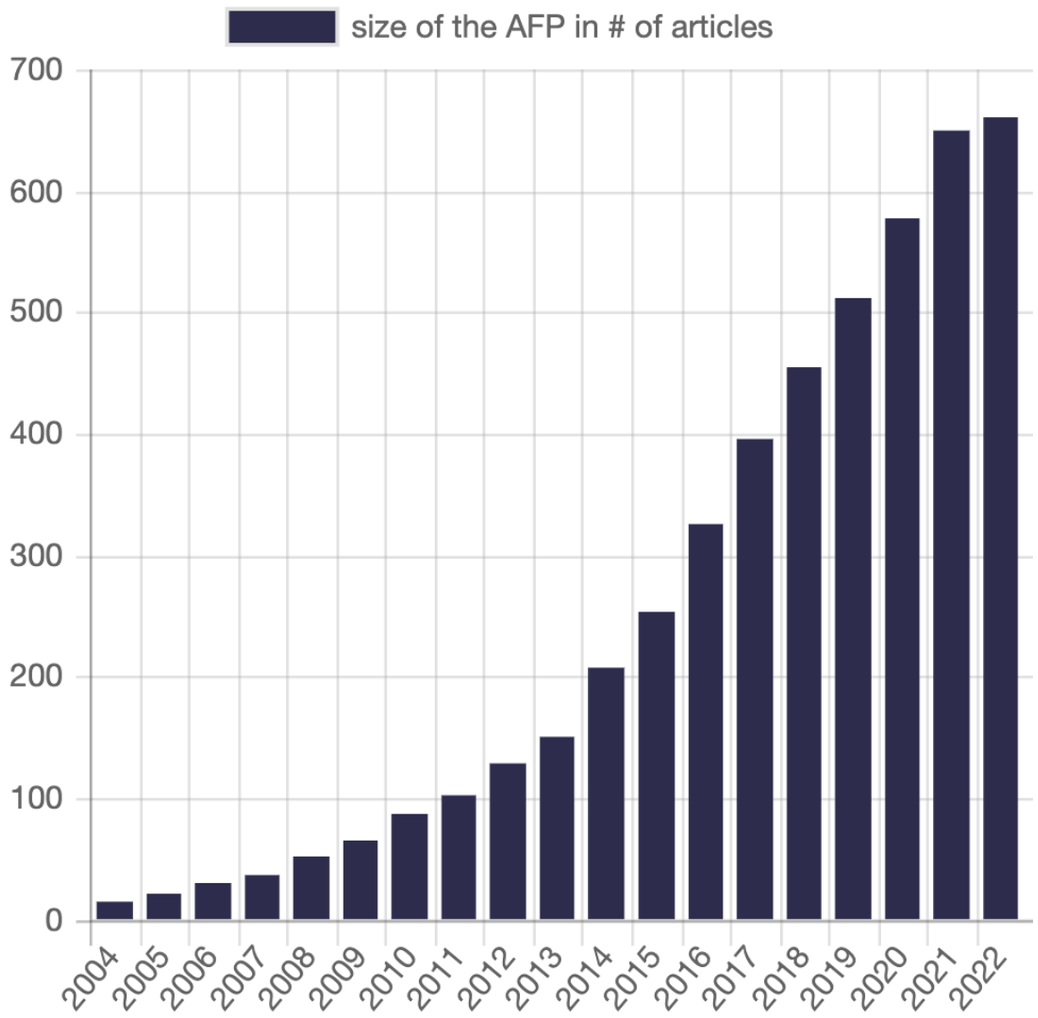
- More powerful ITP systems
  $\Longrightarrow$ growing body of formalised mathematics and engineering



Why Ontologies

- More powerful ITP systems
  $\Longrightarrow$ growing body of formalised mathematics and engineering

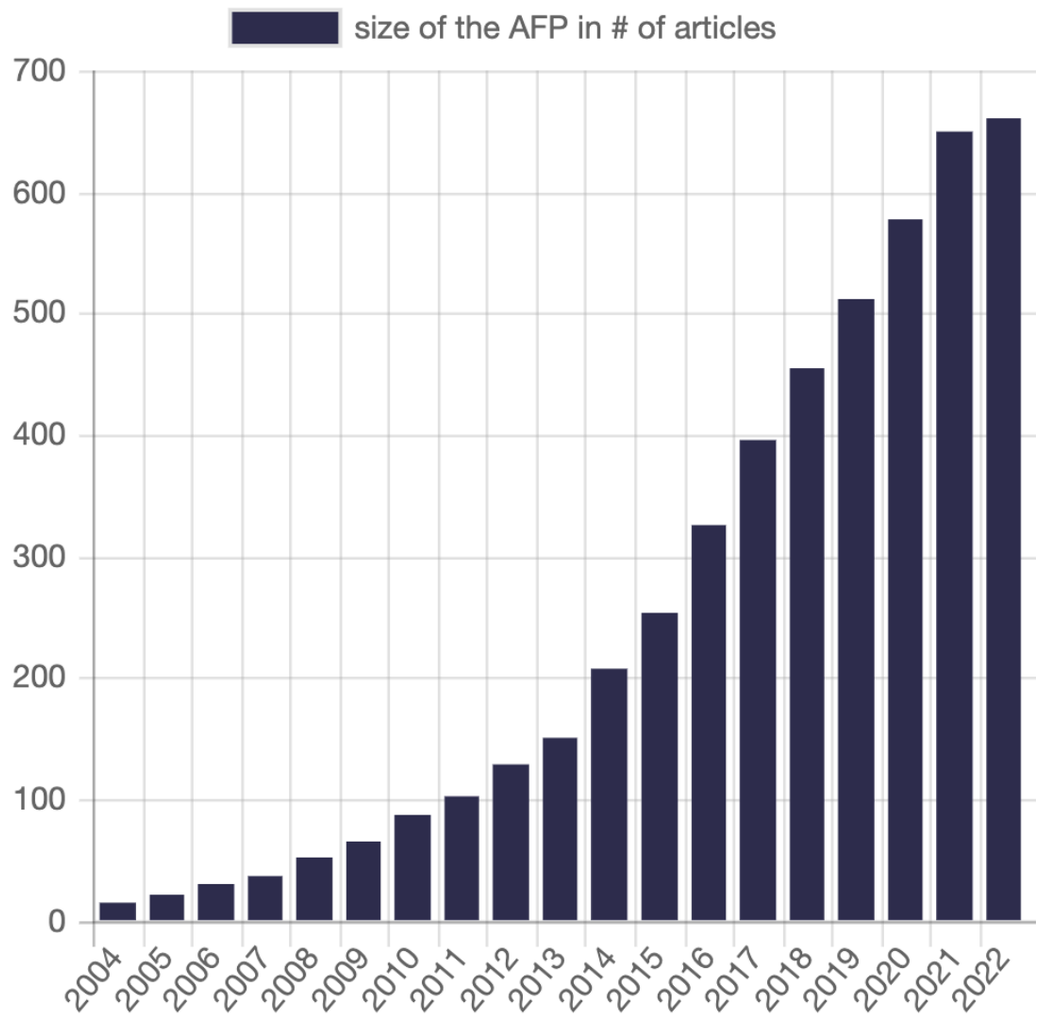
- The Isabelle AFP as example



Why Ontologies

- More powerful ITP systems
  ⟹ growing body of formalised mathematics and engineering

- The Isabelle AFP as example



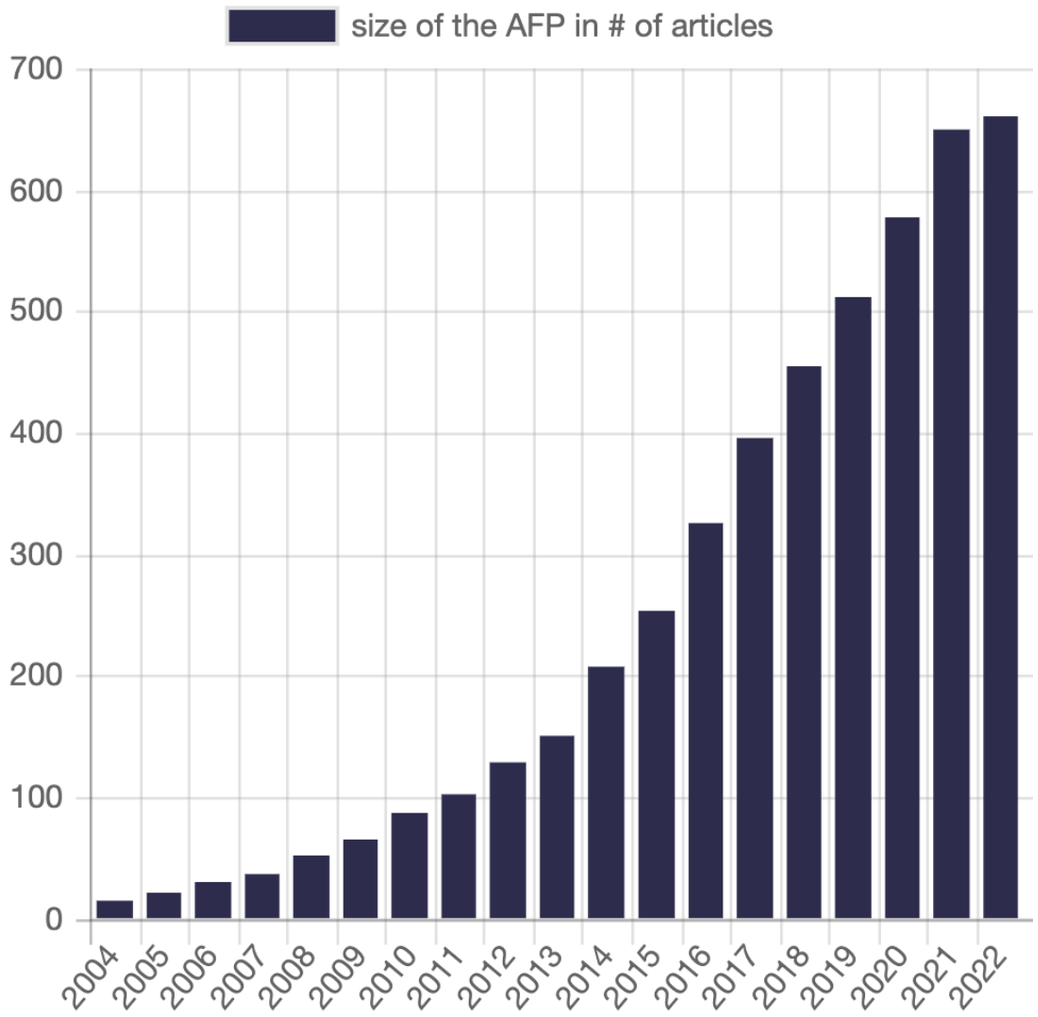
In 2022, the count stood at 661 articles, 420 authors and 3.3 mio loc !

Why Ontologies

- More powerful ITP systems
  ⟹ growing body of formalised mathematics and engineering

- The Isabelle AFP as example

- Rising need for



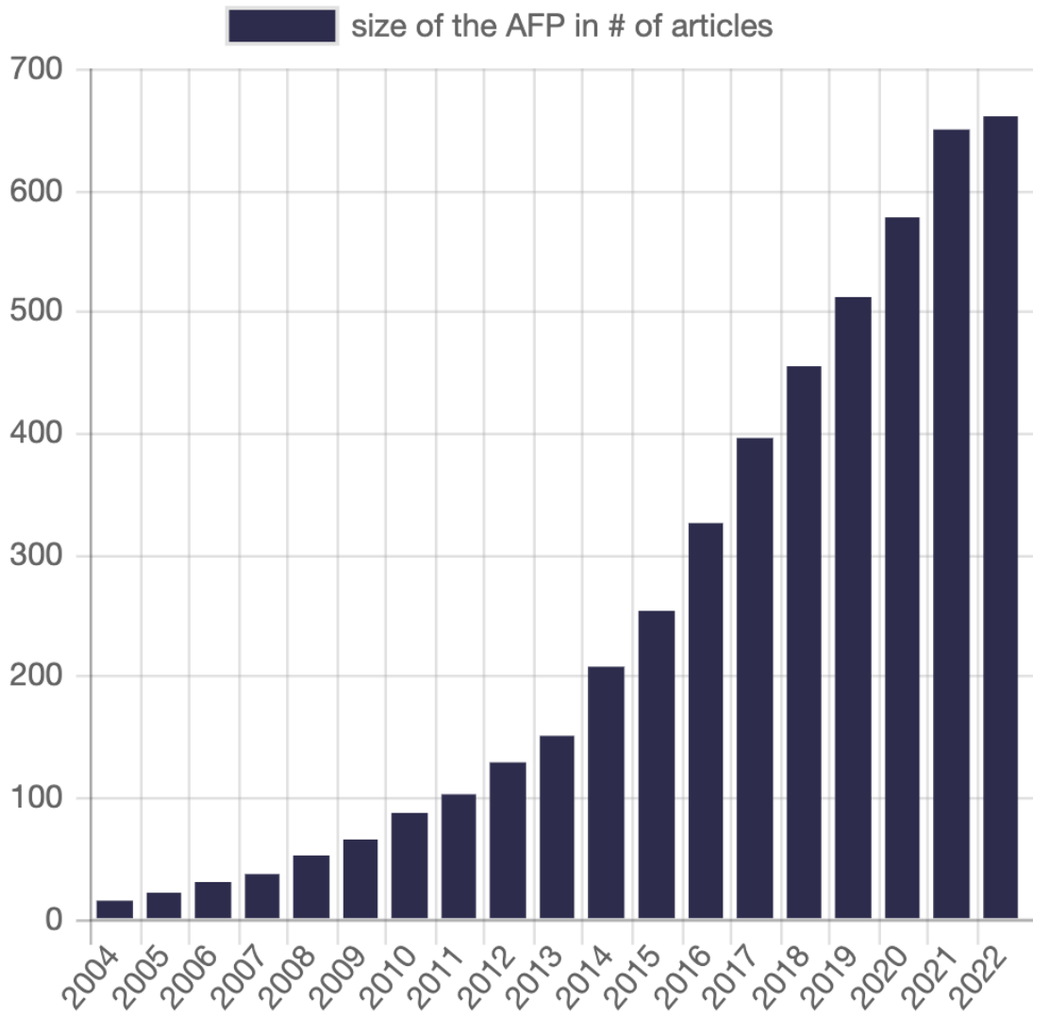
In 2022, the count stood at 661 articles, 420 authors and 3.3 mio loc !

Why Ontologies

- More powerful ITP systems
  ⟹ growing body of formalised mathematics and engineering

- The Isabelle AFP as example

- Rising need for
  - structuring and consistency,
  - advanced “semantic” search,
  - tool-interaction.



In 2022, the count stood at 661 articles, 420 authors and 3.3 mio loc !

Why Ontologies

Why Ontologies

- This requires more structured and typed meta-information for our application domain in theory developments

Why Ontologies

- This requires more structured and typed meta-information
  for our application domain in theory developments

- … and a better dependency-control of the different document elements,

Why Ontologies

- This requires more structured and typed meta-information
  for our application domain in theory developments

- … and a better dependency-control of the different document elements,
  - types, terms, theorems

Why Ontologies

- This requires more structured and typed meta-information
  for our application domain in theory developments

- … and a better dependency-control of the different document elements,
  - types, terms, theorems

  - code (proof-terms, proof generating programs, SML, LaTeX etc, but also Scala and C!)

Why Ontologies

- This requires more structured and typed meta-information
  for our application domain in theory developments

- … and a better dependency-control of the different document elements,
  - types, terms, theorems

  - code (proof-terms, proof generating programs, SML, LaTeX etc, but also Scala and C!)

  - text and diagrams (and perhaps animations, see Jupyter Notebooks https://jupyter.org/)

Why Ontologies

- This requires more structured and typed meta-information
  for our application domain in theory developments

- … and a better dependency-control of the different document elements,
  - types, terms, theorems
  - code (proof-terms, proof generating programs, SML, LaTeX etc, but also Scala and C!)
  - text and diagrams (and perhaps animations, see Jupyter Notebooks https://jupyter.org/)
  - … and the links between them, requiring notions of
    consistency and coherence for collaborative development

Why Ontologies

- This requires more structured and typed meta-information
  for our application domain in theory developments

- … and a better dependency-control of the different document elements,
  - types, terms, theorems
  - code (proof-terms, proof generating programs, SML, LaTeX etc, but also Scala and C!)
  - text and diagrams (and perhaps animations, see Jupyter Notebooks https://jupyter.org/)
  - … and the links between them, requiring notions of
    consistency and coherence for collaborative development

- The language in which such meta-information can be specified
  is called a *document ontology* (or *vocabulary*)

Linking the Formal and the Informal - Existing Approaches -

- Code Antiquotations as in LISP, MetaML, SML, …

```
-| val z = <f 4 5>;
val z = <%f 4 5> : <int>

-| let fun f x y = not x andalso y in run z end;
val it = 8 : int
```

- Document pragmas as in JavaDoc, Doxygen, et al

```
public class AddNum {
  /**
   * This method is used to add two integers. This is
   * a the simplest form of a class method, just to
   * show the usage of various javadoc Tags.
   * @param numA This is the first paramter to addNum method
   * @param numB  This is the second parameter to addNum method
   * @return int This returns sum of numA and numB.
   */
   public int addNum(int numA, int numB) {
     return numA + numB;
   }
```

- Compilation process allows for document generation and some consistency checks

  $\Longrightarrow$ batch mode consistency only.

Linking the Formal and the Informal
- Existing Approaches -

Linking the Formal and the Informal - Existing Approaches -

- The Isabelle Approach to “Text-Antiquotations” (heavily used to assure *coherence* and *traceability* in the technical documentations and papers)

Linking the Formal and the Informal - Existing Approaches -

- The Isabelle Approach to “Text-Antiquotations” (heavily used to assure *coherence* and *traceability* in the technical documentations and papers)
  - Definitions and proofs can be mixed with text elements

Linking the Formal and the Informal - Existing Approaches -

- The Isabelle Approach to “Text-Antiquotations” (heavily used to assure *coherence* and *traceability* in the technical documentations and papers)
  - Definitions and proofs can be mixed with text elements

```
text⟨This is a description.⟩
```

Linking the Formal and the Informal - Existing Approaches -

- The Isabelle Approach to “Text-Antiquotations” (heavily used to assure *coherence* and *traceability* in the technical documentations and papers)
  - Definitions and proofs can be mixed with text elements

```
text⟨This is a description.⟩
```

  - Text Elements may contain Antiquotations to Formal Content in the Logical Context, which are checked and animated in the IDE:

Linking the Formal and the Informal - Existing Approaches -

- The Isabelle Approach to “Text-Antiquotations” (heavily used to assure *coherence* and *traceability* in the technical documentations and papers)
  - Definitions and proofs can be mixed with text elements

```
text‹This is a description.›
```

- Text Elements may contain Antiquotations to Formal Content in the Logical Context, which are checked and animated in the IDE:

```
text‹According to the reflexivity axiom @{thm refl}, we obtain in Γ
      for @{term "fac 5"} the result @{value "fac 5"}.›
```

Linking the Formal and the Informal - Existing Approaches -

- The Isabelle Approach to “Text-Antiquotations” (heavily used to assure *coherence* and *traceability* in the technical documentations and papers)
  - Definitions and proofs can be mixed with text elements

```
text‹This is a description.›
```

- Text Elements may contain Antiquotations to Formal Content in the Logical Context, which are checked and animated in the IDE:

```
text‹According to the reflexivity axiom @{thm refl}, we obtain in Γ
      for @{term "fac 5"} the result @{value "fac 5"}.›
```

- The global doc-generation process yields a presentation in, e.g., .pdf :

Linking the Formal and the Informal
- Existing Approaches -

- The Isabelle Approach to “Text-Antiquotations” (heavily used to assure *coherence* and *traceability* in the technical documentations and papers)
  - Definitions and proofs can be mixed with text elements

```
text‹This is a description.›
```

- Text Elements may contain Antiquotations to Formal Content in the Logical Context, which are checked and animated in the IDE:

```
text‹According to the reflexivity axiom @{thm refl}, we obtain in Γ
      for @{term "fac 5"} the result @{value "fac 5"}.›
```

- The global doc-generation process yields a presentation in, e.g., .pdf :

> According to the reflexivity axiom $x = x$, we obtain in Γ for $\mathrm{fac}\,5$ the result 120.

Linking the Formal and the Informal
- Existing Approaches -

Linking the Formal and the Informal - Existing Approaches -

- Similarly, Isabelle Code uses heavily “SML-Antiquotations”

Linking the Formal and the Informal - Existing Approaches -

- Similarly, Isabelle Code uses heavily “SML-Antiquotations”
  - SML System Code can be mixed with antiquotations producing SML level representation of types and terms:

Linking the Formal and the Informal - Existing Approaches -

- Similarly, Isabelle Code uses heavily “SML-Antiquotations”
  - SML System Code can be mixed with antiquotations producing SML level representation of types and terms:

```
fun ltx_of_term _ _ (Const (@{const_name ‹Cons›},
                            @{typ "char ⇒ char list ⇒ char list"}) $ t1 $ t2)
        = HOLogic.dest_string (Const (@{const_name ‹Cons›},
                                      @{typ "char ⇒ char list ⇒ char list"}) $ t1 $  t2)
  | ltx_of_term _ _ (Const (@{const_name ‹Nil›}, _)) = ""
  | ltx_of_term _ _ (@{term "numeral :: _ ⇒ _"} $ t) = Int.toString(HOLogic.dest_numeral t)
  | ltx_of_term ctx encl ((Const (@{const_name ‹Cons›}, _) $ t1) $ t2) =
      let val inner = (case t2 of
                         Const (@{const_name ‹Nil›}, _) =>  (ltx_of_term ctx true t1)
                       | _ => ((ltx_of_term ctx false t1)^", " ^(ltx_of_term ctx false t2))
                      )
      in if encl then enclose "{" "}" inner else inner end
  | ltx_of_term _ _ (Const (@{const_name ‹None›}, _)) = ""
  | ltx_of_term ctxt _ (Const (@{const_name ‹Some›}, _)$t) = ltx_of_term ctxt true t
  | ltx_of_term ctxt _ t = ""^(Sledgehammer_Util.hackish_string_of_term ctxt t)
```

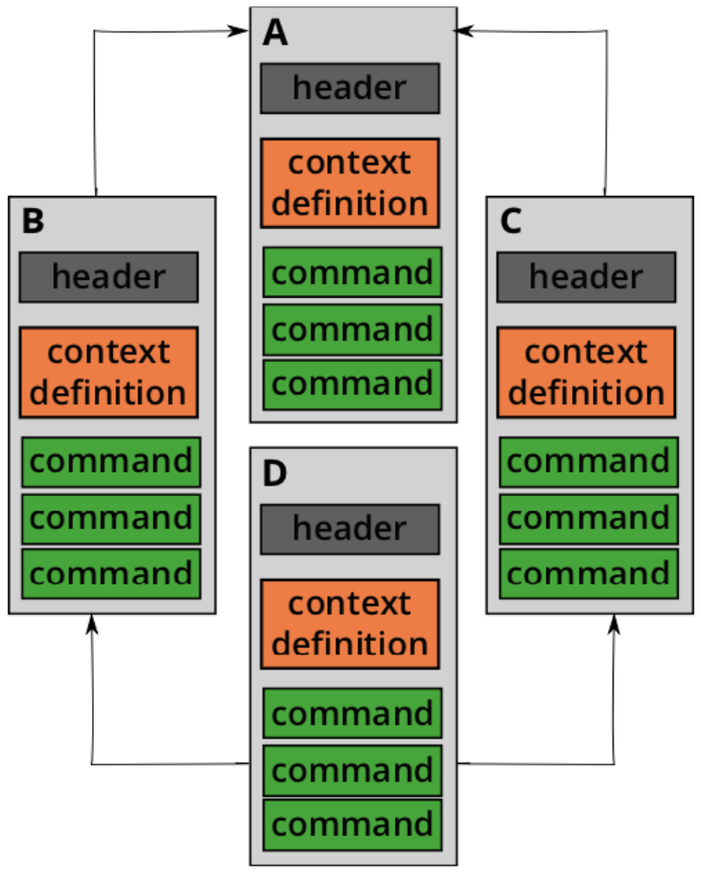
Isabelle’s Document-Centric View on Formal Development

Isabelle's Document-Centric View on Formal Development

- Primary document type: "XXX.thy"

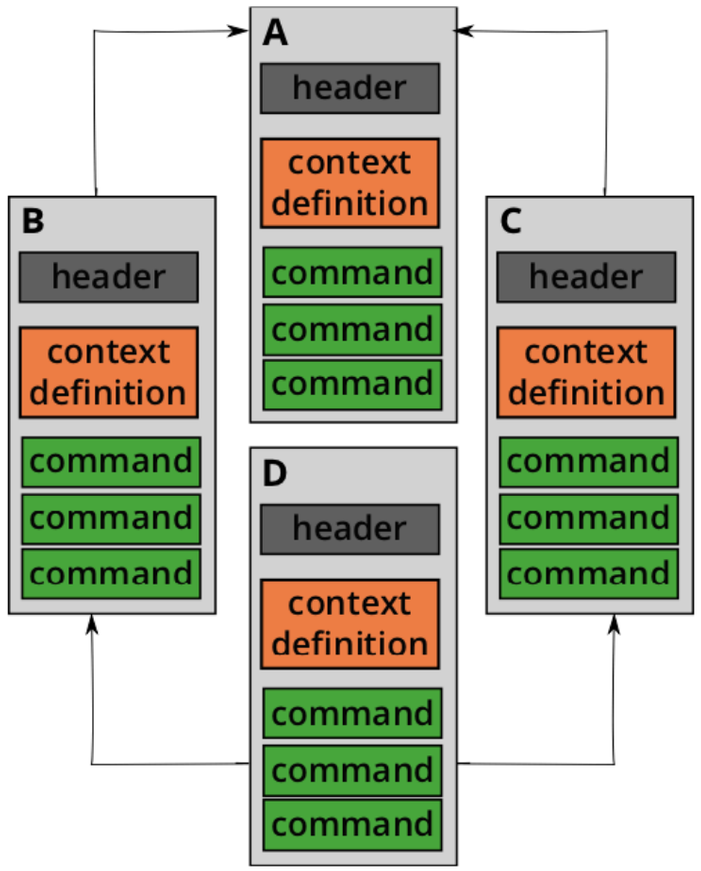
Isabelle's Document-Centric View on Formal Development

- Primary document type: "XXX.thy"
  - Acyclic Graph of units that consist of a sequence of *document elements* called "commands"



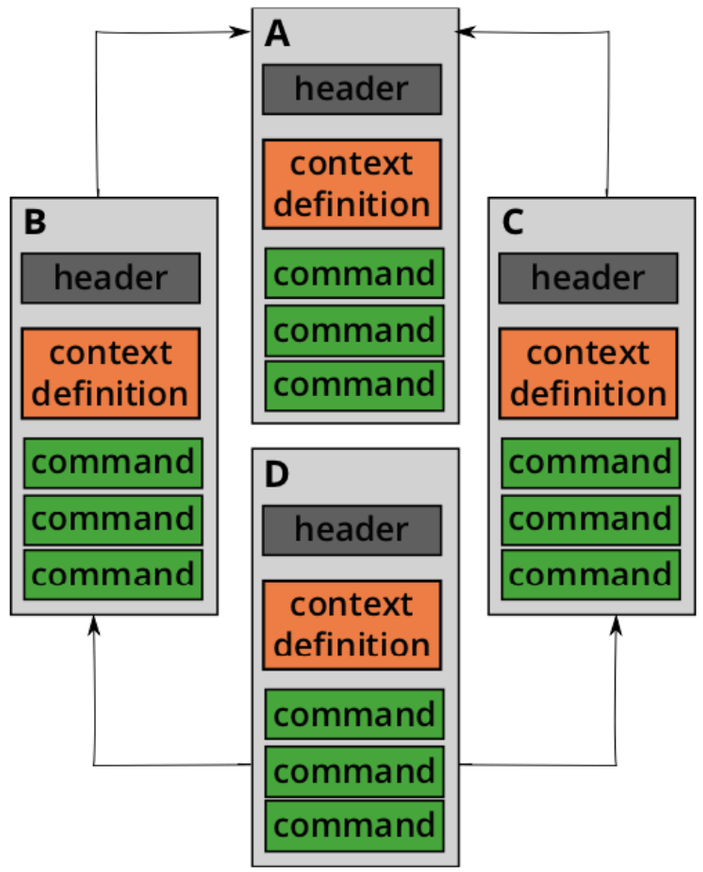
Isabelle's Document-Centric View on Formal Development

- Primary document type: "XXX.thy"
  - Acyclic Graph of units that consist of a sequence of *document elements* called "commands"
  - commands user-programmable in SML



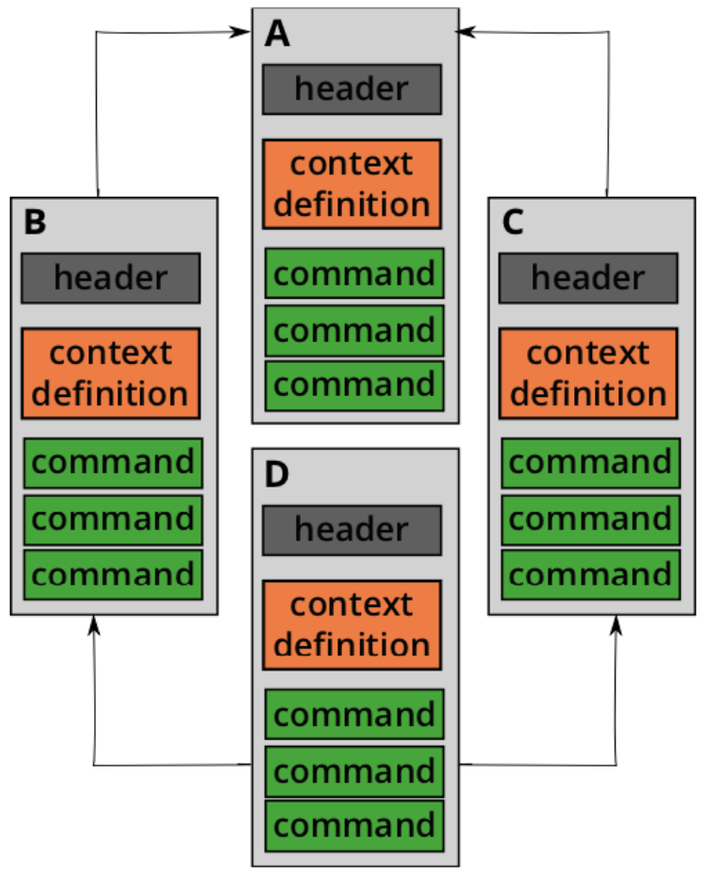
Isabelle's Document-Centric View on Formal Development

- Primary document type: "XXX.thy"
  - Acyclic Graph of units that consist of a sequence of *document elements* called "commands"
  - commands user-programmable in SML
  - Support of Cascade Syntax:
    @{SML ‹ … @{type ‹ …. ›} … ›}



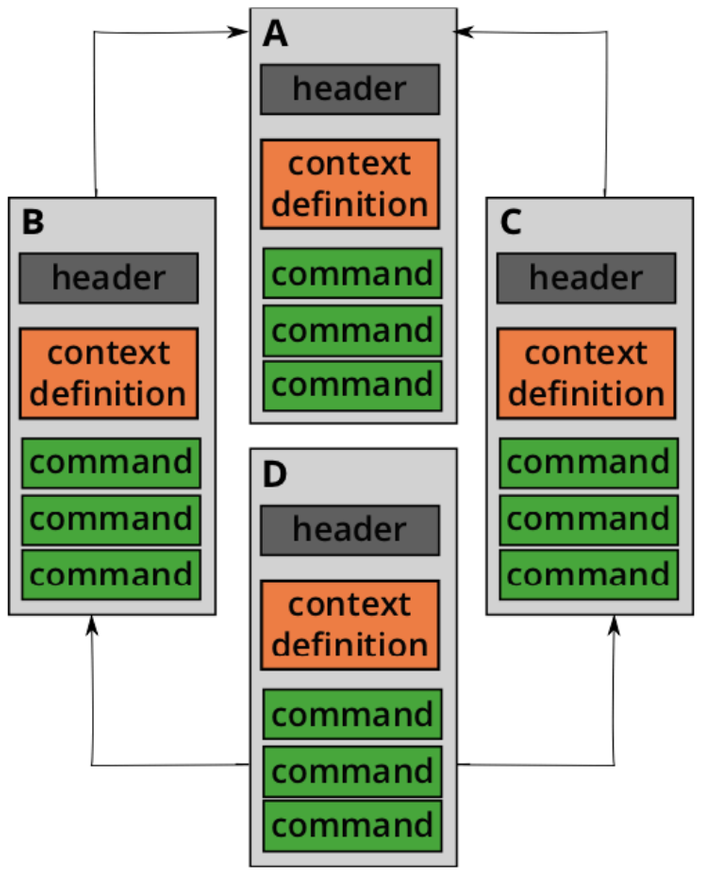
Isabelle's Document-Centric View on Formal Development

- Primary document type: "XXX.thy"
  - Acyclic Graph of units that consist of a sequence of *document elements* called "commands"
  - commands user-programmable in SML
  - Support of Cascade Syntax: @{SML ‹ … @{type ‹ …. ›} … ›}
  - Commands are semantically transformers of the logical context : $\Theta \Rightarrow \Theta$



Isabelle's Document-Centric View on Formal Development

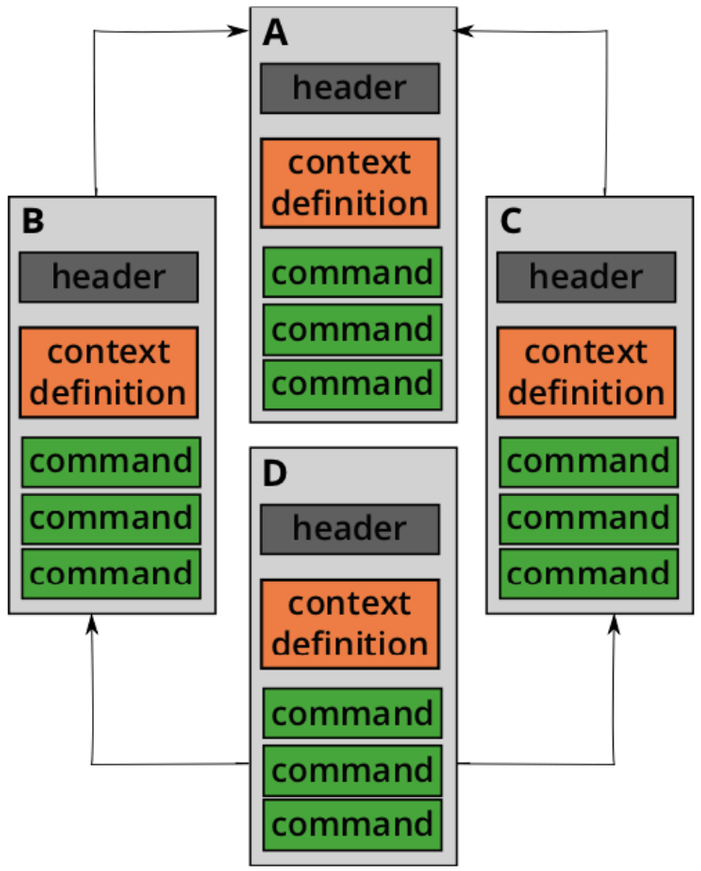
- Primary document type: "XXX.thy"
  - Acyclic Graph of units that consist of a sequence of *document elements* called "commands"
  - commands user-programmable in SML
  - Support of Cascade Syntax: @{SML ‹ … @{type ‹ …. ›} … ›}
  - Commands are semantically transformers of the logical context : $\Theta \Rightarrow \Theta$



- anti-quotations are "semantic macros" and as such partial) functions: $\Theta \Rightarrow text$, $\Theta \Rightarrow sml$, $\Theta \Rightarrow term$

Isabelle's Document-Centric View on Formal Development

- Primary document type: "XXX.thy"
  - Acyclic Graph of units that consist of a sequence of *document elements* called "commands"
  - commands user-programmable in SML
  - Support of Cascade Syntax: @{SML ‹ … @{type ‹ …. ›} … ›}
  - Commands are semantically transformers of the logical context : $\Theta \Rightarrow \Theta$



- anti-quotations are "semantic macros" and as such partial) functions: $\Theta \Rightarrow text$, $\Theta \Rightarrow sml$, $\Theta \Rightarrow term$
- *pervasive continuous build/check of Isabelle/PIDE supports anti-quotations.*

Isabelle/DOF

Isabelle/DOF

- DOF : The Document Ontology Framework

Isabelle/DOF

- DOF : The Document Ontology Framework
- Prior Versions of Isabelle/DOF support semantic annotations of text and code-contexts:

Isabelle/DOF

- DOF : The Document Ontology Framework

- Prior Versions of Isabelle/DOF support semantic annotations of text and code-contexts:

text*[label::classid, $attr_1 = E_1$, **...** $attr_n = E_n$]〈 some semi-formal text 〉

ML*[label::classid, $attr_1 = E_1$, **...** $attr_n = E_n$] 〈 some SML code 〉

Isabelle/DOF

- DOF : The Document Ontology Framework

- Prior Versions of Isabelle/DOF support semantic annotations of text and code-contexts:

text*[label::classid, $attr_1 = E_1$, **...** $attr_n = E_n$]〈 some semi-formal text 〉
ML*[label::classid, $attr_1 = E_1$, **...** $attr_n = E_n$] 〈 some SML code 〉

- Novelty in Isabelle/DOF: support of λ-term-contexts, e.g.:

Isabelle/DOF

- DOF : The Document Ontology Framework

- Prior Versions of Isabelle/DOF support semantic annotations of text and code-contexts:

text*[label::classid, $attr_1 = E_1$, **...** $attr_n = E_n$]〈 some semi-formal text 〉
ML*[label::classid, $attr_1 = E_1$, **...** $attr_n = E_n$] 〈 some SML code 〉

- Novelty in Isabelle/DOF: support of λ-term-contexts, e.g.:

value*[label::classid, $attr_1 = E_1$, **...** $attr_n = E_n$]〈 some annotated λ-term 〉

Isabelle/DOF Core : ODL

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email       :: "string" <= ''''''

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= "''''''"

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)

```
doc_class title   = short_title :: "string option" <= "None"
doc_class author  = email       :: "string" <= ''''''

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"       <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"        <= "{}"
    level       :: "int  option"       <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= "''''''"

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)

  - classes may have attributes with HOL type

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= "''''''"

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)

  - classes may have attributes with HOL type

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= ""
```

```
datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)

  - classes may have attributes with HOL type

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= "''''''"

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)

  - classes may have attributes with HOL type

  - class declarations can be interleaved with arbitrary HOL declarations

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= "''''''"

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)
  - classes may have attributes with HOL type
  - class declarations can be interleaved with arbitrary HOL declarations

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= "''''''"

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)
  - classes may have attributes with HOL type
  - class declarations can be interleaved with arbitrary HOL declarations
  - attributes of class-instances are mutable; (default) values can be denoted by HOL-terms

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= "''''"

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)
  - classes may have attributes with HOL type
  - class declarations can be interleaved with arbitrary HOL declarations
  - attributes of class-instances are mutable; (default) values can be denoted by HOL-terms

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= "''''"

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

- class declarations induce a HOL-type; this allows to establish “ontological links”

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:
  - classes (for the “concepts”)
  - classes may have attributes with HOL type
  - class declarations can be interleaved with arbitrary HOL declarations
  - attributes of class-instances are mutable; (default) values can be denoted by HOL-terms

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= ''''''

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

- class declarations induce a HOL-type; this allows to establish “ontological links”

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes (for the “concepts”)
  - classes may have attributes with HOL type
  - class declarations can be interleaved with arbitrary HOL declarations
  - attributes of class-instances are mutable; (default) values can be denoted by HOL-terms

```
doc_class title  = short_title :: "string option" <= "None"
doc_class author = email      :: "string" <= "''''"

datatype classification = SIL0 | SIL1 | SIL2 | SIL3 | SIL4

doc_class abstract =
    keywordlist :: "string list"      <= []
    safety_level :: "classification"  <= "SIL3"
doc_class text_section =
    authored_by :: "author set"       <= "{}"
    level       :: "int  option"      <= "None"
```

- class declarations induce a HOL-type; this allows to establish “ontological links”

Isabelle/DOF Core : ODL

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

```
type_synonym notion = string

doc_class introduction = text_section +
    authored_by :: "author set"       <= "UNIV"
    uses :: "notion set"
doc_class claim = introduction +
    based_on :: "notion list"
doc_class technical = text_section +
    formal_results  :: "thm list"
doc_class "definition" = technical +
    is_formal :: "bool"
    property  :: "term list"          <= "[]"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes have single inheritance (is a - relation)

```
type_synonym notion = string

doc_class introduction = text_section +
    authored_by :: "author set"       <= "UNIV"
    uses :: "notion set"
doc_class claim = introduction +
    based_on :: "notion list"
doc_class technical = text_section +
    formal_results  :: "thm list"
doc_class "definition" = technical +
    is_formal :: "bool"
    property  :: "term list"          <= "[]"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:
  - classes have single inheritance (is a - relation)

```
type_synonym notion = string

doc_class introduction = text_section +
    authored_by :: "author set"        <= "UNIV"
    uses :: "notion set"
doc_class claim = introduction +
    based_on :: "notion list"
doc_class technical = text_section +
    formal_results  :: "thm list"
doc_class "definition" = technical +
    is_formal :: "bool"
    property  :: "term list"           <= "[]"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes have single inheritance (is a - relation)

```
type_synonym notion = string

doc_class introduction = text_section +
    authored_by :: "author set"       <= "UNIV"
    uses :: "notion set"
doc_class claim = introduction +
    based_on :: "notion list"
doc_class technical = text_section +
    formal_results  :: "thm list"
doc_class "definition" = technical +
    is_formal :: "bool"
    property  :: "term list"          <= "[]"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes have single inheritance (is a - relation)

  - attribute overriding of attributes is possible

```
type_synonym notion = string

doc_class introduction = text_section +
    authored_by :: "author set"       <= "UNIV"
    uses :: "notion set"
doc_class claim = introduction +
    based_on :: "notion list"
doc_class technical = text_section +
    formal_results  :: "thm list"
doc_class "definition" = technical +
    is_formal :: "bool"
    property  :: "term list"          <= "[]"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes have single inheritance (is a - relation)

  - attribute overriding of attributes is possible

  - meta-level types of the ITP were included as abstract HOL types; their inhabitance is checked in the global context Θ

```
type_synonym notion = string

doc_class introduction = text_section +
    authored_by :: "author set"       <= "UNIV"
    uses :: "notion set"
doc_class claim = introduction +
    based_on :: "notion list"
doc_class technical = text_section +
    formal_results  :: "thm list"
doc_class "definition" = technical +
    is_formal :: "bool"
    property  :: "term list"          <= "[]"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes have single inheritance (is a - relation)

  - attribute overriding of attributes is possible

  - meta-level types of the ITP were included as abstract HOL types; their inhabitance is checked in the global context Θ

```
type_synonym notion = string

doc_class introduction = text_section +
    authored_by :: "author set"        <= "UNIV"
    uses :: "notion set"
doc_class claim = introduction +
    based_on :: "notion list"
doc_class technical = text_section +
    formal_results  :: "thm list"
doc_class "definition" = technical +
    is_formal :: "bool"
    property  :: "term list"           <= "[]"
```

Isabelle/DOF Core : ODL

- The Ontology Definition Language (ODL):
  The Mechanism to d*efine* Ontologies

- Features:

  - classes have single inheritance (is a - relation)

  - attribute overriding of attributes is possible

  - meta-level types of the ITP were included as abstract HOL types; their inhabitance is checked in the global context Θ

```
type_synonym notion = string

doc_class introduction = text_section +
    authored_by :: "author set"       <= "UNIV"
    uses :: "notion set"
doc_class claim = introduction +
    based_on :: "notion list"
doc_class technical = text_section +
    formal_results  :: "thm list"
doc_class "definition" = technical +
    is_formal :: "bool"
    property  :: "term list"          <= "[]"
```

DOF Example Document

DOF Example Document

- Defining the Ontological Context

DOF Example Document

- Defining the Ontological Context

```
theory Steam_Boiler
  imports
    tiny_cert  (* The ontology defined in Listing 1.1. *)
begin
```

DOF Example Document

- Defining the Ontological Context

```
theory Steam_Boiler
  imports
    tiny_cert  (* The ontology defined in Listing 1.1. *)
begin
```

- And there we go:

DOF Example Document

- Defining the Ontological Context

- And there we go:

```
title*[a] ⟨The Steam Boiler Controller⟩
abstract*[abs, safety_level="SIL4", keywordlist = "[''controller'']"]⟨
  We present a formalization of a program which serves to control the
  level of water in a steam boiler.
⟩

section*[intro::introduction]⟨Introduction⟩
text⟨We present ...⟩

section*[T1::technical]⟨Physical Environment⟩
text⟨
  The system comprises the following units
  • the steam-boiler
  • a device to measure the quantity of water in the steam-boiler
  • ...
⟩
```

DOF Example Document

- Defining the Ontological Context

- And there we go:

- … where title* and abstract* are macros for text*[a::title,…], etc…

```
title*[a] ⟨The Steam Boiler Controller⟩
abstract*[abs, safety_level="SIL4", keywordlist = "[''controller'']"]⟨
  We present a formalization of a program which serves to control the
  level of water in a steam boiler.
⟩

section*[intro::introduction]⟨Introduction⟩
text⟨We present ...⟩

section*[T1::technical]⟨Physical Environment⟩
text⟨
  The system comprises the following units
  • the steam-boiler
  • a device to measure the quantity of water in the steam-boiler
  • ...
⟩
```

DOF Example Document

- Defining the Ontological Context

- And there we go:

- … where title* and abstract* are macros for text*[a::title,…], etc…

- … and the meta-data instances are *a*, *abs*, *intro*, *T1*, attached to these doc elements …

```
title*[a] ⟨The Steam Boiler Controller⟩
abstract*[abs, safety_level="SIL4", keywordlist = "[''controller'']"]⟨
  We present a formalization of a program which serves to control the
  level of water in a steam boiler.
⟩

section*[intro::introduction]⟨Introduction⟩
text⟨We present ...⟩

section*[T1::technical]⟨Physical Environment⟩
text⟨
  The system comprises the following units
  • the steam-boiler
  • a device to measure the quantity of water in the steam-boiler
  • ...
⟩
```

DOF Example Document

- Defining the Ontological Context

- And there we go:

- … where title* and abstract* are macros for text*[a::title,…], etc…

- … and the meta-data instances are *a*, *abs*, *intro*, *T1*, attached to these doc elements …

```
title*[a]⟨The Steam Boiler Controller⟩
abstract*[abs  safety_level="SIL4", keywordlist = "[''controller'']"]⟨
  We present a formalization of a program which serves to control the
  level of water in a steam boiler.
⟩

section*[intro::introduction]⟨Introduction⟩
text⟨We present ...⟩

section*[T1::technical]⟨Physical Environment⟩
text⟨
  The system comprises the following units
  • the steam-boiler
  • a device to measure the quantity of water in the steam-boiler
  • ...
⟩
```

Isabelle/DOF Core : Class Invariants

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**
- Eg.: Invariants for

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*

- *New: ODL uses arbitrary λ-terms containing generated term-antiquo-tations in invariants, attribute definitions and commands like value**

- Eg.: Invariants for

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**
- Eg.: Invariants for
  - data-integrity constraints

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**
- Eg.: Invariants for
  - data-integrity constraints

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

$\forall\ \sigma \in \text{result}.\ \text{evidence}\ \sigma = \text{proof} \longleftrightarrow \text{property}\ \sigma \neq []$

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**
- Eg.: Invariants for
  - data-integrity constraints

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**
- Eg.: Invariants for
  - data-integrity constraints

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

invariant <lab> : evidence σ = proof ⟷ property σ ≠ []

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*

- *New: ODL uses arbitrary λ-terms containing generated term-antiquo-tations in invariants, attribute definitions and commands like value**

- Eg.: Invariants for

  - data-integrity constraints

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

—> definition <lab>_inv :: ‘a result_scheme ⇒ bool”

where “ <lab>_inv σ ≡ evidence σ = proof ⟷ property σ ≠ []

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**
- Eg.: Invariants for
  - data-integrity constraints

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*

- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**

- Eg.: Invariants for

  - data-integrity constraints

  - *… using "built-in" term antiquotations for "term", "typ", "thm*

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**
- Eg.: Invariants for
  - data-integrity constraints
  - *… using "built-in" term antiquotations for "term", "typ", "thm*

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

```
invariant <lab> : evidence σ=proof ⟷ @{thm"safe"}∈set(property σ)
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*

- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**

- Eg.: Invariants for

  - data-integrity constraints

  - *… using "built-in" term antiquotations for "term", "typ", "thm*

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*

- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and commands like value**

- Eg.: Invariants for

  - data-integrity constraints

  - *… using "built-in" term antiquotations for "term", "typ", "thm*

  - *may use DOF-generated term-antiquotations like @{result ''<some result instance>"} or @{introduction ''intro"} or @{instance-of ''result"}, etc.*

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"           <= "[]"
```

Isabelle/DOF Core : Class Invariants

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and some commands like value**

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and some commands like value**
- Invariants for

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and some commands like value**
- Invariants for

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"           <= "[]"
doc_class example = technical +
    referring_to :: "(notion + definition) set" <=  "{}"
doc_class "conclusion" = text_section +
    establish   :: "(claim × result) set"
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and some commands like value**
- Invariants for
  - *"a result* text element must provide evidence in form of a proven theorem …"

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"           <= "[]"
doc_class example = technical +
    referring_to :: "(notion + definition) set" <=  "{}"
doc_class "conclusion" = text_section +
    establish   :: "(claim × result) set"
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and some commands like value**
- Invariants for
  - *"a result* text element must provide evidence in form of a proven theorem …"

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
doc_class example = technical +
    referring_to :: "(notion + definition) set" <=  "{}"
doc_class "conclusion" = text_section +
    establish   :: "(claim × result) set"
```

$\forall\ \sigma \in \text{conclusion}.\ \forall\ x \in \text{Domain}(\text{establish}\ \sigma).$
$\exists\ y \in \text{Range}(\text{establish}\ \sigma).\ (x, y) \in \text{establish}\ \sigma$

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and some commands like value**
- Invariants for
  - *"a result* text element must provide evidence in form of a proven theorem …"

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
doc_class example = technical +
    referring_to :: "(notion + definition) set" <=  "{}"
doc_class "conclusion" = text_section +
    establish   :: "(claim × result) set"
```

~~∀ σ ∈ conclusion. ∀ x ∈ Domain(establish σ).~~
~~∃ y ∈ Range(establish σ). (x, y) ∈ establish σ~~

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and some commands like value**
- Invariants for
  - *"a result* text element must provide evidence in form of a proven theorem …"

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"           <= "[]"
doc_class example = technical +
    referring_to :: "(notion + definition) set" <=  "{}"
doc_class "conclusion" = text_section +
    establish   :: "(claim × result) set"
```

Isabelle/DOF Core : Class Invariants

- ODL used already *λ-(ground)-terms to denote values for attributes.*
- *New: ODL uses arbitrary λ-terms containing generated term-antiquotations in invariants, attribute definitions and some commands like value**
- Invariants for
  - *"a result* text element must provide evidence in form of a proven theorem …"

```
datatype kind = expert_opinion | argument | proof

doc_class result = technical +
    evidence  :: kind
    property  :: "thm list"            <= "[]"
doc_class example = technical +
    referring_to :: "(notion + definition) set" <=  "{}"
doc_class "conclusion" = text_section +
    establish   :: "(claim × result) set"
```

∀ σ ∈ conclusion. ∀ y ∈ @{instance_of ‘’claim”}.
∃ y ∈ Range(establish σ). (x,y) ∈ establish σ

Consequences

Consequences

- If an ODL ontology generates “intra-logical” representations, what’s the benefit ?

Consequences

- If an ODL ontology generates “intra-logical” representations, what’s the benefit ?
  - We don’t have to learn a new (meta)-language

Consequences

- If an ODL ontology generates “intra-logical” representations, what’s the benefit ?
  - We don’t have to learn a new (meta)-language
  - We can define new operations on them inside the logic and develop their theory …

Consequences

- If an ODL ontology generates “intra-logical” representations, what’s the benefit ?
    - We don’t have to learn a new (meta)-language
    - We can define new operations on them inside the logic and develop their theory …
    - … to develop a query language, for example:

Consequences

- If an ODL ontology generates “intra-logical” representations, what’s the benefit ?

  - We don’t have to learn a new (meta)-language

  - We can define new operations on them inside the logic and develop their theory …

  - … to develop a query language, for example:
    value*〈 *filter (is_interesting) @{instances-of ‘’result”}* 〉

Consequences

- If an ODL ontology generates “intra-logical” representations, what’s the benefit ?

  - We don’t have to learn a new (meta)-language

  - We can define new operations on them inside the logic and develop their theory …

  - … to develop a query language, for example:
    value*〈 *filter (is_interesting) @{instances-of ‘’result”}* 〉

- We can relate ontologies and ontology instances by formal proof (‘’ontology alignment, ontology mapping”)

Consequences: Example Proof of an Ontology Mapping

Consequences: Example Proof of an Ontology Mapping

- A “‘Generic’ Reference Ontology” vs. a “‘local’ Ontology”

Consequences: Example Proof of an Ontology Mapping

- A “‘Generic’ Reference Ontology” vs. a “‘local’ Ontology”

Isabelle code

```
definition sum
  where "sum S = (fold (+) S 0)"

datatype Hardware_Type =
  Motherboard |
  Expansion_Card |
  Storage_Device |
  Fixed_Media |
  Removable_Media |
  Input_Device |
  Output_Device

onto_class Resource =
  name :: string

onto_class Electronic = Resource +
  provider :: string
  manufacturer :: string

onto_class Component = Electronic +
  mass :: int
  dimensions :: "int list"

onto_class Simulation_Model = Electronic +
  type :: string

onto_class Informatic = Resource +
  description :: string

onto_class Hardware = Informatic +
  type :: Hardware_Type
  mass :: int
  composed_of :: "Component list"
  invariant c1 :: "mass σ = sum(map Component.mass (composed_of σ))"
```

Consequences: Example Proof of an Ontology Mapping

- A “‘Generic’ Reference Ontology” vs. a “‘local’ Ontology”

Isabelle code

```
definition sum
  where "sum S = (fold (+) S 0)"

datatype Hardware_Type =
  Motherboard |
  Expansion_Card |
  Storage_Device |
  Fixed_Media |
  Removable_Media |
  Input_Device |
  Output_Device

onto_class Resource =
  name :: string

onto_class Electronic = Resource +
  provider :: string
  manufacturer :: string

onto_class Component = Electronic +
  mass :: int
  dimensions :: "int list"

onto_class Simulation_Model = Electronic +
  type :: string

onto_class Informatic = Resource +
  description :: string

onto_class Hardware = Informatic +
  type :: Hardware_Type
  mass :: int
  composed_of :: "Component list"
  invariant c1 :: "mass σ = sum(map Component.mass (composed_of σ))"
```

Isabelle code

```
onto_class Item =
  name :: string

onto_class Product = Item +
  serial_number :: int
  provider :: string
  mass :: int

onto_class Computer_Software = Item +
  version :: int

onto_class Electronic_Component = Product +
  dimensions :: "int set"

onto_class Computer_Hardware = Product +
  type :: Hardware_Type
  composed_of :: "Product list"
  invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))"
```

Consequences: Example Proof of an Ontology Mapping

- A “‘Generic’ Reference Ontology” vs. a “‘local’ Ontology”

Isabelle code

```
definition sum
  where "sum S = (fold (+) S 0)"

datatype Hardware_Type =
  Motherboard |
  Expansion_Card |
  Storage_Device |
  Fixed_Media |
  Removable_Media |
  Input_Device |
  Output_Device

onto_class Resource =
  name :: string

onto_class Electronic = Resource +
  provider :: string
  manufacturer :: string

onto_class Component = Electronic +
  mass :: int
  dimensions :: "int list"

onto_class Simulation_Model = Electronic +
  type :: string

onto_class Informatic = Resource +
  description :: string

onto_class Hardware = Informatic +
  type :: Hardware_Type
  mass :: int
  composed_of :: "Component list"
  invariant c1 :: "mass σ = sum(map Component.mass (composed_of σ))"
```

Isabelle code

```
onto_class Item =
  name :: string

onto_class Product = Item +
  serial_number :: int
  provider :: string
  mass :: int

onto_class Computer_Software = Item +
  version :: int

onto_class Electronic_Component = Product +
  dimensions :: "int set"

onto_class Computer_Hardware = Product +
  type :: Hardware_Type
  composed_of :: "Product list"
  invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))"
```

Consequences: Example Proof of an Ontology Mapping

- A “‘Generic’ Reference Ontology” vs. a “‘local’ Ontology”

Isabelle code

```
definition sum
  where "sum S = (fold (+) S 0)"

datatype Hardware_Type =
  Motherboard |
  Expansion_Card |
  Storage_Device |
  Fixed_Media |
  Removable_Media |
  Input_Device |
  Output_Device

onto_class Resource =
  name :: string

onto_class Electronic = Resource +
  provider :: string
  manufacturer :: string

onto_class Component = Electronic +
  mass :: int
  dimensions :: "int list"

onto_class Simulation_Model = Electronic +
  type :: string

onto_class Informatic = Resource +
  description :: string

onto_class Hardware = Informatic +
  type :: Hardware_Type
  mass :: int
  composed_of :: "Component list"
  invariant c1 :: "mass σ = sum(map Component.mass (composed_of σ))"
```

Isabelle code

```
onto_class Item =
  name :: string

onto_class Product = Item +
  serial_number :: int
  provider :: string
  mass :: int

onto_class Computer_Software = Item +
  version :: int

onto_class Electronic_Component = Product +
  dimensions :: "int set"

onto_class Computer_Hardware = Product +
  type :: Hardware_Type
  composed_of :: "Product list"
  invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))"
```

```
definition Computer_Hardware_to_Hardware_morphism ::
            "'a Computer_Hardware_scheme ⇒ Hardware"
            ("_ ⟨Hardware⟩ComputerHardware" [1000]999)
          where "σ ⟨Hardware⟩ComputerHardware =
                (⦇ Resource.tag_attribute = 2::int ,
                   Resource.name = name σ ,
                   Informatic.description = ''no description'',
                   Hardware.type = Computer_Hardware.type σ ,
                   Hardware.mass = mass σ ,
                   Hardware.composed_of =
                                map Product_to_Component_morphism
                                    (Computer_Hardware.composed_of σ) ⦈"
```

Consequences: Example Proof of an Ontology Mapping

Consequences: Example Proof of an Ontology Mapping

- A “‘Generic’ Reference Ontology” vs. a “‘local’ Ontology”

Consequences: Example Proof of an Ontology Mapping

- A “‘Generic’ Reference Ontology” vs. a “‘local’ Ontology”

“The mapping is correct (preserves the invariants)”

Consequences: Example Proof of an Ontology Mapping

- A "'Generic' Reference Ontology" vs. a "'local' Ontology"

"The mapping is correct (preserves the invariants)"

Isabelle code

```
lemma inv_c2_preserved :
  "c2_inv σ ⟹ c1_inv (σ ⟨Hardware⟩_ComputerHardware)"
  unfolding c1_inv_def c2_inv_def
            Computer_Hardware_to_Hardware_morphism_def
            Product_to_Component_morphism_def
  using comp_def by (auto)

lemma Computer_Hardware_to_Hardware_morphism_total :
 "Computer_Hardware_to_Hardware_morphism ' ({X::Computer_Hardware. c2_inv X})
    ⊆ ({X::Hardware. c1_inv X})"
  using inv_c2_preserved by auto
```

But what “are” ontology-generated term antiquotations ???

But what “are” ontology-generated term antiquotations ???

- First of all: how are they processed:

But what “are” ontology-generated term antiquotations ???

- First of all: how are they processed:
  - parsing

But what “are” ontology-generated term antiquotations ???

- First of all: how are they processed:
  - parsing
  - type checking

But what “are” ontology-generated term antiquotations ???

- First of all: how are they processed:
  - parsing
  - type checking
  - validation (an argument is indeed a valid reference in the context)

But what “are” ontology-generated term antiquotations ???

- First of all: how are they processed:
  - parsing
  - type checking
  - validation (an argument is indeed a valid reference in the context)
  - expansion (replacement of a reference against logical terms)

But what “are” ontology-generated term antiquotations ???

- First of all: how are they processed:
  - parsing
  - type checking
  - validation (an argument is indeed a valid reference in the context)
  - expansion (replacement of a reference against logical terms)
  - evaluation (to SML code, or by nbe)

But what “are” ontology-generated term antiquotations ???

But what “are” ontology-generated term antiquotations ???

- Then “built-in” term-anti quotations can be:

But what “are” ontology-generated term antiquotations ???

- Then “built-in” term-anti quotations can be:
  - just uninterpreted constants (without expansion)

But what “are” ontology-generated term antiquotations ???

- Then “built-in” term-anti quotations can be:
  - just uninterpreted constants (without expansion)

```
sort “typ”
consts typ_anno :: ‘’string ⇒ typ”    (“@{typ _}” 100)
```

But what “are” ontology-generated term antiquotations ???

- Then “built-in” term-anti quotations can be:
  - just uninterpreted constants (without expansion)

```
sort “typ”
consts typ_anno :: ‘’string ⇒ typ”    (“@{typ _}” 100)
```

  - a ‘’shallow” data-type representation (without expansion)

```
datatype “typ” = typ_anno ‘’string” (“@{typ _}” 100)
```

But what “are” ontology-generated term antiquotations ???

- Then “built-in” term-anti quotations can be:
  - just uninterpreted constants (without expansion)

```
sort “typ”
consts typ_anno :: ‘’string ⇒ typ”    (“@{typ _}” 100)
```

  - a ‘’shallow” data-type representation (without expansion)

```
datatype “typ” = typ_anno ‘’string” (“@{typ _}” 100)
```

  - or a “deep” data-type representation into an Isabelle Meta-Model such as [Nipkow,Rosskopf 21] (with expansion)

But what “are” ontology-generated term antiquotations ???

- Then “built-in” term-anti quotations can be:
  - just uninterpreted constants (without expansion)

```
sort “typ”
consts typ_anno :: ‘’string ⇒ typ”    (“@{typ _}” 100)
```

  - a ‘’shallow” data-type representation (without expansion)

```
datatype “typ” = typ_anno ‘’string” (“@{typ _}” 100)
```

  - or a “deep” data-type representation into an Isabelle Meta-Model such as [Nipkow,Rosskopf 21] (with expansion)

```
definition  typ_anno :: “string ⇒ typ” where  “typ_anno S ≡ undefined”

datatype “typ”   =  is_Ty : Ty name “typ list”
                  |  is-Tv : Tv variable sort
datatype “term” = is_Ct : Ct name “typ”
                  |  is_Fv: Fv variable “typ”
                  | is_Bv: Bv nat
                  | is_Abs : Abs “typ” “term”
                  | is_App : App “term” “term” {infixl “$” 100)
```

Conclusion

Conclusion

- DOF provides a framework

Conclusion

- DOF provides a framework
  - for defining ontologies in the context of ITP systems

Conclusion

- DOF provides a framework
  - for defining ontologies in the context of ITP systems
  - its typed ! It has a logical interpretation !
    It is therefore an “in-between” between an ML and a logic.

Conclusion

- DOF provides a framework
  - for defining ontologies in the context of ITP systems
  - its typed ! It has a logical interpretation !
    It is therefore an “in-between” between an ML and a logic.
  - provides a generated infrastructure for meta-data of types, terms, thm’s and text and code elements

Conclusion

- DOF provides a framework
  - for defining ontologies in the context of ITP systems
  - its typed ! It has a logical interpretation !
    It is therefore an “in-between” between an ML and a logic.
  - provides a generated infrastructure for meta-data of types, terms, thm’s and text and code elements
- DOF provides a framework to enforce on-the-fly ontology-conform documentation checking

Conclusion

- DOF provides a framework
  - for defining ontologies in the context of ITP systems
  - its typed ! It has a logical interpretation !
    It is therefore an “in-between” between an ML and a logic.
  - provides a generated infrastructure for meta-data of types, terms, thm’s and text and code elements
- DOF provides a framework to enforce on-the-fly ontology-conform documentation checking
- DOF provides infrastructure for proofs over the logical representation of ontologies and meta-data …

Conclusion

- DOF provides a framework
  - for defining ontologies in the context of ITP systems
  - its typed ! It has a logical interpretation !
    It is therefore an “in-between” between an ML and a logic.
  - provides a generated infrastructure for meta-data of types, terms, thm’s and text and code elements
- DOF provides a framework to enforce on-the-fly ontology-conform documentation checking
- DOF provides infrastructure for proofs over the logical representation of ontologies and meta-data …
- Ontologies generating meta-data can be used for other forms of Tool Interaction via “deep interpretations” into a meta-model

Conclusion

Conclusion

- (P)IDE’s are more than just a technical asset

Conclusion

- (P)IDE’s are more than just a technical asset
- … it is a corner-stone for a revolution

Conclusion

- (P)IDE’s are more than just a technical asset
- … it is a corner-stone for a revolution

- 1970’ies TEXT

Conclusion

- (P)IDE’s are more than just a technical asset
- … it is a corner-stone for a revolution

- 1970’ies TEXT
- 1990’ies HYPERTEXT

Conclusion

- (P)IDE’s are more than just a technical asset
- … it is a corner-stone for a revolution

- 1970’ies TEXT
- 1990’ies HYPERTEXT
- 2010’ies REACTIVE DOCUMENTS

Conclusion

- (P)IDE’s are more than just a technical asset
- … it is a corner-stone for a revolution

- 1970’ies TEXT
- 1990’ies HYPERTEXT
- 2010’ies REACTIVE DOCUMENTS
- 2020’ies SEMANTIC DOCUMENTS (???)