



#### 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

ZECCCO Search	Q Upload Communities
August 18, 2019	Software Open Access
Isabelle/DOF	
💿 Brucker, Achim D.; Wolff, Burkhart	
Isabelle/DOF is a Document Ontology Framework (DOF formal developments with structured, typed meta-infor purposes (e.g., semantic queries, tool interaction, or do	-), on top of Isabelle/HOL, allowing to annotate text elements in rmation which can be defined by developers according to their ocument generation).
Files (3.2 MB)	×
Name	Size
Isabelle_DOF-1.0.0_Isabelle2019.tar.xz	3.2 MB
md5:c714698b973b7b212655705e9a976516 🚱	
Isabelle_DOF-1.0.0_Isabelle2019.tar.xz.asc	833 Bytes 🕹 Download
md5:cf2ba2d2a7c0ed98ee7c1c2e711c7366 🕖	
Set Citations 0	~
Show only: Literature (0) Dataset (0) So	ftware (0) Unknown (0) Search Q
	No citations.

PUBLIC RELEASE: http/10.5281/zenodo.3370483



More powerful ITP systems

 ⇒ growing body of formalised
 mathematics and engineering



- More powerful ITP systems
   ⇒ growing body of formalised
   mathematics and engineering
- The Isabelle AFP as example



- 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 !

- 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 !

- 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.





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

- 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,

- 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

- 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!)

- 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/)

- 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

- 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*)

• 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.

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

- 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

- 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.)
```

- 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:

- 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 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 \Gamma
for @{term "fac 5"} the result @{value "fac 5"}.)

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

- 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 \Gamma
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  $\Gamma$  for fac 5 the result 120.

• Similarly, Isabelle Code uses heavily "SML-Antiquotations"

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

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

• Primary document type: "XXX.thy"

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



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



- 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 < .... >} ... >}



- 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 :  $\Theta \Rightarrow \Theta$



- 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.



• DOF : The Document Ontology Framework

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

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

**text\***[label::classid, attr<sub>1</sub> =  $E_1$ , ... attr<sub>n</sub> =  $E_n$ ]  $\langle$  some semi-formal text  $\rangle$ **ML**\*[label::classid, attr<sub>1</sub> =  $E_1$ , ... attr<sub>n</sub> =  $E_n$ ]  $\langle$  some SML code  $\rangle$ 

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

**text\***[label::classid, attr<sub>1</sub> =  $E_1$ , ... attr<sub>n</sub> =  $E_n$ ]  $\langle$  some semi-formal text  $\rangle$ **ML**\*[label::classid, attr<sub>1</sub> =  $E_1$ , ... attr<sub>n</sub> =  $E_n$ ]  $\langle$  some SML code  $\rangle$ 

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

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

**text\***[label::classid, attr<sub>1</sub> =  $E_1$ , ... attr<sub>n</sub> =  $E_n$ ]  $\langle$  some semi-formal text  $\rangle$ **ML**\*[label::classid, attr<sub>1</sub> =  $E_1$ , ... attr<sub>n</sub> =  $E_n$ ]  $\langle$  some SML code  $\rangle$ 

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

**value**\*[*label::classid, attr*<sub>1</sub> =  $E_1$ , ... attr<sub>n</sub> =  $E_n$ ] < some annotated  $\lambda$ -term >

 The Ontology Definition Language (ODL): The Mechanism to define Ontologies

- The Ontology Definition Language (ODL): The Mechanism to define 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"</pre>

- The Ontology Definition Language (ODL): The Mechanism to define 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"</pre>

- The Ontology Definition Language (ODL): The Mechanism to define Ontologies
- Features:
  - classes (for the "concepts")



- The Ontology Definition Language (ODL): The Mechanism to define 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"</pre>

- The Ontology Definition Language (ODL): The Mechanism to define Ontologies
- Features:
  - classes (for the "concepts")
  - classes may have attributes with HOL type

0	I I	L] /	U	
doc_class	title = short_	title :: "st	ring option" <= "N	one"
doc_class	author = email	:: "str	ing" <= "''''	
datatype c	classification =	= SILO   SIL1	SIL2   SIL3   S	IL4
doc_class	abstract =			
keywor	dlist :: "strir	ng list"	<= []	
safety	/_level :: "clas	ssification"	<= "SIL3"	
doc_class	<pre>text_section =</pre>			
author	<pre>red_by :: "authorno"</pre>	or set"	<= "{}"	
level	:: "int	option"	<= "None"	

- The Ontology Definition Language (ODL): The Mechanism to define Ontologies
- Features:
  - classes (for the "concepts")
  - classes may have attributes with HOL type

	U	ТТ	L - 1			
d	oc_class	title = sh	ort_title ::	"string	option"	<= "None"
d	oc_class	author = em	ail ::	'string"	<= "''''	
d	atatype o	classificati	on = SIL0   S	SIL1   SI	L2   SIL	3   SIL4
d	oc_class	abstract =				
	keywo	rdlist :: "s	tring list"	<= [	]	
	safety	y_level :: "	classificatio	on" <= "	'SIL3"	
d	oc_class	text_sectio	n =			
	autho	red_by :: 🚜	uthor sot"	<= "	'{}"	
	level	:: <b>"</b> i	nt option"	<= "	'None"	

- The Ontology Definition Language (ODL): The Mechanism to define Ontologies
- Features:
  - classes (for the "concepts")
  - classes may have attributes with HOL type

0	I I	L] /	U	
doc_class	title = short_	title :: "st	ring option" <= "N	one"
doc_class	author = email	:: "str	ing" <= "''''	
datatype c	classification =	= SILO   SIL1	SIL2   SIL3   S	IL4
doc_class	abstract =			
keywor	dlist :: "strir	ng list"	<= []	
safety	/_level :: "clas	ssification"	<= "SIL3"	
doc_class	<pre>text_section =</pre>			
author	<pre>red_by :: "authorno"</pre>	or set"	<= "{}"	
level	:: "int	option"	<= "None"	

- The Ontology Definition Language (ODL): The Mechanism to define Ontologies
- Features:
  - classes (for the "concepts")
  - classes may have attributes with HOL type
  - class declarations can be interleaved with arbitrary HOL declarations

0	I I	LJ /	U	
doc_class doc_class	title = short author = email	_title :: "st :: "str	ring option" ing" <= "'''	<= "None" '"
datatype c	lassification	= SIL0   SIL1	SIL2   SII	L3   SIL4
doc_class keywor safety doc_class author level	abstract = dlist :: "stri _level :: cla text_section = ed_by :: "auth :: "int	ng list" ssification' or set" option"	<= [] <= "SIL3" <= "{}" <= "None"	

- The Ontology Definition Language (ODL): The Mechanism to define Ontologies
- Features:
  - classes (for the "concepts")
  - classes may have attributes with HOL type
  - class declarations can be interleaved with arbitrary HOL declarations

0	I I	L] /	U	
doc_class doc_class	title = short_ author = email	title :: "st :: "str	ring option" <= ing" <= "''''	"None"
datatype c	lassification =	= SILO   SIL1	.   SIL2   SIL3	SIL4
doc_class keywor safety doc_class	abstract = dlist :: "strin _level :: "clastertion =	ng list" ssification"	<= [] <= "SIL3"	
author level	red_by :: "autho :: "int	or set" option"	<= "{}" <= "None"	

- The Ontology Definition Language (ODL): The Mechanism to define 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

U	I I	L] /	U	
doc_class	title = short_	title :: "st	ring option"	<= "None"
doc_class	author = email	:: "stri	ing" <= "''''	п
<mark>datatype</mark> c	lassification =	SILO   SIL1	SIL2   SIL	.3   SIL4
doc_class	abstract =			
keywor	dlist :: "strin	g list"	<= []	
safety	_level :: "clas	sification"	<= "SIL3"	
doc_class	<pre>text_section =</pre>			
author	ed_by :: "autho	r set"	<= "{}"	
level	:: "int	option"	<= "None"	

- The Ontology Definition Language (ODL): The Mechanism to define 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

0	цТ	LJ /	U	
doc_class	title = short	_title :: "st	ring option	" <= "None"
doc_class	author = email	:: "str	ing" <= "''	
datatype	classification	= SILO   SIL1	SIL2   S	IL3   SIL4
doc_class	abstract =			
keywo	rdlist :: "stri	ng list"	<= []	
safet	y_level :: "cla	ssification"	<= "SIL3"	
doc_class	<pre>text_section =</pre>	:		
autho	<pre>red_by :: "auth</pre>	or set"	<= "{}"	
level	:: "int	option"	<= "None"	

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

- The Ontology Definition Language (ODL): The Mechanism to define 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



• class declarations induce a HOL-type; this allows to establish "ontological links"

- The Ontology Definition Language (ODL): The Mechanism to define 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

0	цТ	LJ /	U	
doc_class	title = short	_title :: "st	ring option	" <= "None"
doc_class	author = email	:: "str	ing" <= "''	
datatype	classification	= SILO   SIL1	SIL2   S	IL3   SIL4
doc_class	abstract =			
keywo	rdlist :: "stri	ng list"	<= []	
safet	y_level :: "cla	ssification"	<= "SIL3"	
doc_class	<pre>text_section =</pre>	:		
autho	<pre>red_by :: "auth</pre>	or set"	<= "{}"	
level	:: "int	option"	<= "None"	

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

 The Ontology Definition Language (ODL): The Mechanism to define Ontologies

- The Ontology Definition Language (ODL): The Mechanism to define Ontologies
- Features:

- The Ontology Definition Language (ODL): The Mechanism to define 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" <= "[]"</pre>
```

- The Ontology Definition Language (ODL): The Mechanism to define 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" <= "[]"</pre>
```

- The Ontology Definition Language (ODL): The Mechanism to define 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 tist" doc\_class technical = text\_section + formal\_results :: "thm list" doc\_class "definition" = technical + is\_formal :: "bool" property :: "term list" <= "[]"</pre>

- The Ontology Definition Language (ODL): The Mechanism to define 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" <= "[]"</pre>
```

- The Ontology Definition Language (ODL): The Mechanism to define 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" <= "[]"</pre>
```

- The Ontology Definition Language (ODL): The Mechanism to define 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 Ø

```
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" <= "[]"</pre>
```

- The Ontology Definition Language (ODL): The Mechanism to define 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" <= "[]"</pre>
```

- The Ontology Definition Language (ODL): The Mechanism to define 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 Ø

```
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" <= "[]"</pre>
```

#### 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
Defining the Ontological Context

theory Steam\_Boiler
 imports
 tiny\_cert (\* The ontology defined in Listing 1.1. \*)
begin

• And there we go:

 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
 ...
}

- 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

• ...

- 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
- . . .

- 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
<pre>abstract [abs] safety_level="SIL4", keywordlist = "[''controller'']"]</pre>
We present a formalization of a program which serves to control the
level of water in a steam boiler.
$\rangle$
<pre>section*[intro::introduction] (Introduction)</pre>
text(We present)
section [T1.] technicall (Physical Environment)
text(
The system comprises the following units
the steem beiler
• the steam-boller
<ul> <li>a device to measure the quantity of water in the steam-boiler</li> </ul>
•
$\rangle$

• ODL used already  $\lambda$ -(ground)-terms to denote values for attributes.

- ODL used already  $\lambda$ -(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\*

- ODL used already  $\lambda$ -(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

- ODL used already  $\lambda$ -(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 datatype kind = expert\_opinion | argument | proof
   doc\_class result = technical + evidence :: kind property :: "thm list" <= "[]"</li>

- ODL used already  $\lambda$ -(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	<pre>datatype kind = expert_opinion   a</pre>	argument   proof
	<ul> <li>data-integrity constraints</li> </ul>	<pre>doc_class result = technical +     evidence :: kind     property :: "thm list"</pre>	<= "[]"

- ODL used already  $\lambda$ -(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

   datatype kind = expert\_opinion | argument | proof
   doc\_class result = technical + evidence :: kind property :: "thm list" <= "[]"</li>

   ∀ σ ∈ result. evidence σ = proof ↔ property σ ≠ []

- ODL used already  $\lambda$ -(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	<pre>datatype kind = expert_opinion   a</pre>	argument   proof
	<ul> <li>data-integrity constraints</li> </ul>	<pre>doc_class result = technical +     evidence :: kind     property :: "thm list"</pre>	<= "[]"

- ODL used already  $\lambda$ -(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	<pre>for datatype kind = expert_opinion   argument   proof</pre>
<ul> <li>data-integrity constraints</li> </ul>	<pre>doc_class result = technical +     evidence :: kind     property :: "thm list" &lt;= "[]"</pre>

invariant <lab> : evidence  $\sigma$  = proof  $\leftrightarrow$  property  $\sigma \neq$  []

- ODL used already  $\lambda$ -(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

   datatype kind = expert\_opinion | argument | proof
   doc\_class result = technical +
   evidence :: kind
   property :: "thm list" <= "[]"</li>

   —> definition <lab>\_inv :: 'a result\_scheme ⇒ bool"

where " <lab>\_inv  $\sigma \equiv$  evidence  $\sigma =$  proof  $\leftrightarrow$  property  $\sigma \neq$  []

- ODL used already  $\lambda$ -(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	<pre>datatype kind = expert_opinion   a</pre>	argument   proof
	<ul> <li>data-integrity constraints</li> </ul>	<pre>doc_class result = technical +     evidence :: kind     property :: "thm list"</pre>	<= "[]"

- ODL used already  $\lambda$ -(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\*

<ul> <li>Eg.: Invariants for</li> </ul>	<pre>datatype kind = expert_opinion  </pre>	argument   proof
<ul> <li>data-integrity constraints</li> </ul>	<pre>doc_class result = technical +     evidence :: kind     property :: "thm list"</pre>	<= "[]"

 ... using "built-in" term antiquotations for "term", "typ", "thm

- ODL used already  $\lambda$ -(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\*

<ul> <li>Eg.: Invariants for</li> </ul>	<pre>datatype kind = expert_opinion  </pre>	argument   proof
<ul> <li>data-integrity constraints</li> </ul>	<pre>doc_class result = technical +     evidence :: kind     property :: "thm list"</pre>	<= "[]"

 ... using "built-in" term antiquotations for "term", "typ", "thm

invariant <lab> : evidence  $\sigma$ =proof  $\leftrightarrow @{\text{thm"safe"}} \in \text{set}(\text{property }\sigma)$ 

- ODL used already  $\lambda$ -(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\*

<ul> <li>Eg.: Invariants for</li> </ul>	<pre>datatype kind = expert_opinion  </pre>	argument   proof
<ul> <li>data-integrity constraints</li> </ul>	<pre>doc_class result = technical +     evidence :: kind     property :: "thm list"</pre>	<= "[]"

 ... using "built-in" term antiquotations for "term", "typ", "thm

- ODL used already  $\lambda$ -(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\*

<ul> <li>Eg.: Invariants for</li> </ul>	<pre>datatype kind = expert_opinion  </pre>	argument   proof
<ul> <li>data-integrity constraints</li> </ul>	<pre>doc_class result = technical +     evidence :: kind     property :: "thm list"</pre>	<= "[]"

- ... 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.

- ODL used already  $\lambda$ -(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\*

- ODL used already  $\lambda$ -(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

- ODL used already  $\lambda$ -(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
• 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"</pre>
```

- ODL used already  $\lambda$ -(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\*
- datatype kind = expert\_opinion | argument | proof Invariants for doc\_class result = technical + • "a result evidence :: kind text element property :: "thm list" <= "[]" must provide doc\_class example = technical + evidence in referring\_to :: "(notion + definition) set" <= "{}"</pre> form of a doc\_class "conclusion" = text\_section + proven theoestablish :: "(claim  $\times$  result) set" rem ...."

- ODL used already  $\lambda$ -(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).</pre>
```

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

- ODL used already  $\lambda$ -(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
```

- ODL used already  $\lambda$ -(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\*
- datatype kind = expert\_opinion | argument | proof Invariants for doc\_class result = technical + • "a result evidence :: kind text element property :: "thm list" <= "[]" must provide doc\_class example = technical + evidence in referring\_to :: "(notion + definition) set" <= "{}"</pre> form of a doc\_class "conclusion" = text\_section + proven theoestablish :: "(claim  $\times$  result) set" rem ...."

- ODL used already  $\lambda$ -(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"</pre>
```

 $\begin{array}{l} \forall \ \sigma \in \mbox{ conclusion. } \forall \ y \in @\{\mbox{instance\_of ''claim''\}.\\ \ \exists \ y \in \mbox{Range}(\mbox{establish } \sigma). \ (x,y) \in \mbox{establish } \sigma \end{array}$ 

• If an ODL ontology generates "intra-logical" representations, what's the benefit ?

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

- 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 ...

- 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:

- 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"} >

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

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

<pre>definition sum where "sum S = (fold (+) S 0)"</pre>	Isabelle c
<pre>datatype Hardware_Type =   Motherboard     Expansion_Card     Storage_Device     Fixed_Media     Removable_Media     Input_Device     Output_Device</pre>	
<pre>ontoclass Resource =     name :: string</pre>	
<pre>onto_class Electronic = Resource +    provider :: string    manufacturer :: string</pre>	
<pre>ontoclass Component = Electronic +   mass :: int   dimensions :: "int list"</pre>	
<pre>onto_class Simulation_Model = Electronic +   type :: string</pre>	
<pre>onto_class Informatic = Resource +   description :: string</pre>	
<pre>ontoclass Hardware = Informatic +   type :: Hardware_Type   mass :: int   composed_of :: "Component list"   invariant c1 :: "mass σ = sum(map Component.mass (composed_of σ))</pre>	u

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

Isabelle code

Isabelle code

where "sum S = (fold (+) S 0)"       name :: string         datatype Hardware_Type =       name :: string         Motherboard         expansion_Card           Expansion_Card         serial_number :: int         Fixed_Modia         name :: string         Not_class Product = Item +       serial_number :: int         Input_Device         name :: string         onto_class Resource =       name :: string         name :: string       onto_class Electronic = Resource +         provider :: string       onto_class Simulation_Model = Electronic +         mass :: int       dimensions :: "int list"         onto_class Informatic = Resource +       escource +         type :: string       onto_class Informatic = Resource +         description :: string       onto_class Informatic = Resource +	definition sum	onto_class Item =
datatype Hardware_Type =       onto_class Product = Item +         Motherboard         serial_number :: int         Expansion_Card         provider :: string         Storage_Device         provider :: string         Fixed_Media         onto_class Computer_Software = Item +         Newovable_Media         onto_class Computer_Software = Item +         onto_class Resource =       onto_class Electronic_Component = Product +         name :: string       onto_class Electronic = Resource +         provider :: string       onto_class Computer_Hardware = Product +         type :: Hardware_Type       composed_of :: "Product list"         onto_class Simulation_Model = Electronic +       type :: string         onto_class Informatic = Resource +       description :: string         onto_class Informatic = Resource +       description :: string	where "sum $S = (fold (+) S 0)$ "	name :: string
datatype Hardware_Type =Motherboard  Expansion_Card  Storage_Device  Fixed_Media  Removable_Media  Input_Device  Outo_class Resource =name :: stringonto_class Resource =name :: stringonto_class Electronic = Resource +provider :: stringonto_class Component = Electronic +mass :: intdimensions :: "int list"onto_class Simulation_Model = Electronic +type :: stringonto_class Informatic = Resource +type :: string		
Motherboard         serial_number :: int         Expansion_Card         serial_number :: int         Storage_Device         provider :: string         mamovable_Media         onto_class Computer_Software = Item +         Notput_Device         onto_class Resource =         name :: string       onto_class Electronic = Resource +         provider :: string       onto_class Electronic = Resource +         provider :: string       onto_class Component = Product +         dimensions :: "int list"       onto_class Simulation_Model = Electronic +         mass :: int       dimensions :: "int list"         onto_class Informatic = Resource +       description :: string         onto_class Informatic = Resource +       description :: string	datatype Hardware_Type =	<pre>onto_class Product = Item +</pre>
Expansion_Card           Storage_Device           Fixed.Media           Removable_Media           Input_Device           Output_Device         onto_class Resource =         name :: string         nonc_class Resource = string         manufacturer :: string         mass :: int         dimensions :: "int list"         onto_class Simulation_Model = Electronic +         type :: string         onto_class Informatic = Resource +         opto:_class Informatic = Resource +         description :: string	Motherboard	serial_number :: int
<pre>Storage_Device   Fixed_Media   Removable_Media   Input_Device Outo_class Resource = name :: string onto_class Electronic = Resource + provider :: string manufacturer :: string onto_class Simulation_Model = Electronic + mass :: int dimensions :: "int list" onto_class Informatic = Resource + description :: string onto_class Informatic = Resource + description :: string</pre>	Expansion_Card	provider :: string
<pre>Fixed_Media   Removable_Media   Removable_Media   Input_Device   Output_Device   Output_Device onto_class Resource = name :: string onto_class Electronic = Resource + provider :: string onto_class Component = Electronic + mass :: int dimensions :: "int list" onto_class Simulation_Model = Electronic + type :: string onto_class Informatic = Resource + description :: string</pre> onto_class Informatic = Resource + description :: string	Storage_Device	mass :: int
Removable_Media           Input_Device           Output_Device         onto_class Resource =         name :: string         onto_class Electronic = Resource +         provider :: string         manufacturer :: string         onto_class Somponent = Electronic +         mass :: int         dimensions :: "int list"         onto_class Simulation_Model = Electronic +         type :: string         onto_class Informatic = Resource +         description :: string	Fixed_Media	
Input_Device   Output_Device   Onto_class Resource = name :: string onto_class Electronic = Resource + provider :: string onto_class Component = Electronic + mass :: int dimensions :: "int list" onto_class Simulation_Model = Electronic + type :: string onto_class Simulation_Model = Electronic + type :: string onto_class Informatic = Resource + description :: string onto_class Informatic = Resource +	Removable_Media	onto class Computer_Software = Item +
Output_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	Input_Device	version :: int
<pre>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</pre> onto_class Informatic = Resource + description :: string	Output Device	
<pre>onto_class Resource = name :: string onto_class Electronic = Resource + provider :: string onto_class Component = Electronic + mass :: int dimensions :: "int list" onto_class Simulation_Model = Electronic + type :: string onto_class Informatic = Resource + description :: string</pre>		onto class Electronic Component = Product +
<pre>onto_class Electronic = Resource + provider :: string onto_class Component = Electronic + mass :: int dimensions :: "int list" onto_class Informatic = Resource + type :: string onto_class Informatic = Resource + description :: string</pre>	onto class Resource =	dimensions :: "int set"
<pre>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</pre>	name :: string	
<pre>onto_class Electronic = Resource + provider :: string onto_class Component = Electronic + mass :: int dimensions :: "int list" onto_class Simulation_Model = Electronic + type :: string onto_class Informatic = Resource + description :: string</pre>	nume in String	onto class Computer Hardware = Product +
<pre>conto_class Simulation_Model = Electronic +     type :: string onto_class Informatic = Resource +     description :: string onto_class Informatic = Resource +     description :: string </pre>	onto class Electronic = Resource +	type :: Hardware Type
<pre>invariant c2 :: "Product tist" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" invariant c2 :: "Product.mass σ = sum(</pre>	nrovider :: string	composed of :: "Product list"
<pre>invaluetation :: string invaluetation :: string i</pre>	manufacturer :: string	invariant $c_2$ :: "Product mass $\sigma = sum(man Product mass (composed of \sigma))"$
<pre>onto_class Component = Electronic +   mass :: int   dimensions :: "int list" onto_class Simulation_Model = Electronic +   type :: string onto_class Informatic = Resource +   description :: string</pre>	manaractarer scring	
<pre>mass :: int dimensions :: "int list" onto_class Simulation_Model = Electronic + type :: string onto_class Informatic = Resource + description :: string</pre>	onto class Component = Electronic +	
<pre>image if int dimensions :: "int list" onto_class Simulation_Model = Electronic + type :: string onto_class Informatic = Resource + description :: string</pre>	mass ·· int	
<pre>onto_class Simulation_Model = Electronic +   type :: string onto_class Informatic = Resource +   description :: string</pre>	dimensions :: "int list"	
<pre>onto_class Simulation_Model = Electronic +   type :: string onto_class Informatic = Resource +   description :: string</pre>		
<pre>onto_class Informatic = Resource +     description :: string</pre>	onto class Simulation Model = Electronic +	
onto_class Informatic = Resource + description :: string	type :: string	
onto_class Informatic = Resource + description :: string		
description :: string	onto class Informatic = Resource +	
	description :: string	
onto class Hardware = Informatic +	onto class Hardware = Informatic +	
type :: Hardware_Type	type :: Hardware_Type	
mass :: int	mass :: int	
composed_of :: "Component list"	<pre>composed_of :: "Component list"</pre>	
invariant c1 :: "mass $\sigma$ = sum(map Component.mass (composed_of $\sigma$ ))"	invariant c1 :: "mass $\sigma$ = sum(map Component.mass (composed_of $\sigma$ ))"	

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

definition sum where "sum $S = (fold (+) S \theta)$ "	belle code	onto_class Item =
where $Sum S = (10ta (+) S 0)$		nume II String
datatype Hardware_Type =		onto class Product = Item +
Motherboard		
Expansion_Card		provider :: string
Storage_Device		mass :: int
Fixed_Media		
Removable_Media		<pre>onto_class Computer_Software = Item +</pre>
Input_Device		version :: int
Output_Device		
		<pre>onto_class Electronic_Component = Product +</pre>
onto_class Resource =		dimensions :: "int set"
name :: string		
		<pre>onto_class Computer_Hardware = Product +</pre>
<pre>onto_class Electronic = Resource +</pre>		type :: Hardware_Type
provider :: string		composed_of :: "Product list"
manufacturer :: string		invariant c2 :: "Product.mass $\sigma$ = sum(map Product.mass (composed_of $\sigma$ ))"
<pre>onto_class Component = Electronic +</pre>		
mass :: int		
dimensions :: "int list"		
<pre>onto_class Simulation_Model = Electronic +</pre>		
type :: string		
<pre>onto_class Informatic = Resource +</pre>		
description :: string		
type :: Hardware_Type		
mass :: int		
composea_ot :: "Component List"		

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

<pre>datatype Hardware_Type = Motherboard   Expansion.Gard   Storage_Device   Fixed.Media   Removable.Media   Removable.Media   Input_Device   Output_Device   Output_Device onto_class Resource = name :: string onto_class Resource = rame :: string onto_class Startare = Product + type :: Hardware = Product + type :: Hardware = Product + type :: Hardware = Product + type :: Wroduct list* invariant c2 :: "Product list* definition Computer_Hardware = to_Hardware* ("- (Hardware)ComputerHardware* ("- (Hardware)C</pre>	definition sum where "sum S = (fold (+) S 0)"	<pre>onto_class Item =     name :: string</pre>
Motherboard         serial_number :: int         Expansion_Card         serial_number :: int         Storage_Device         mass :: int         Fixed_Media         mass :: int         Removable_Media         int         Input_Device         onto_class Computer_Software = Item +         output_Device       onto_class Electronic - Component = Product +         onto_class Electronic = Resource +       onto_class Computer_Hardware = Product +         provider :: string       onto_class Component = Flectronic +         mass :: int       dimensions :: "int list"         onto_class Simulation_Model = Electronic +       "a Computer_Hardware_to_Hardware"         type :: string       "a Computer_Hardware"         onto_class Informatic = Resource +       (!Gomputer_Hardware")         onto_cla	datatype Hardware_Type =	<pre>onto_class Product = Item +</pre>
Expansion_Card         provider :: string         Storage_Device         mass :: int         Fixed_Media         mass :: int         Removable_Media         onto_class Computer_Software = Item +         Output_Device         onto_class Resource =         name :: string       onto_class Electronic - Component = Product +         onto_class Electronic = Resource +       timesions :: "int set"         provider :: string       onto_class Computer_Hardware = Product +         onto_class Electronic = Resource +       type :: Hardware_Type         composed_of :: "Product List"       onto_class or = sum(map Product.mass (composed_of of))"         onto_class Simulation_Model = Electronic +       "'a Computer_Hardware_scheme => Hardware"         type :: string       "'a Computer_Hardware =         onto_class Informatic = Resource +       (" (Hardware)Computer_Hardware =         type :: string       "'a Computer_Hardware =         onto_class Informatic = Resource +       (" (Hardware)Computer_Hardware =         type :: string       "'a Computer_Hardware =         onto_class Informatic = Resource +       (" (Hardware)Computer_Hardware =         type :: string       "'a Computer_Hardware =         onto_class Informatic = Resource +       (" (Hardware) Computer_Hardware =         type :: string       "'a Computer_Hardware	Motherboard	serial_number :: int
Storage_Device           Fixed_Media           Removable_Media           Input_Device         onto_class Resource =         name :: string         onto_class Electronic = Resource +         provider :: string         onto_class Computer_Hardware = Product +         dimensions :: "int list"         onto_class Simulation_Model = Electronic +         type :: string         onto_class Informatic = Resource +         description :: string         onto_class Informatic = Resource +         description :: string	Expansion_Card	provider :: string
<pre>Fixed_Media   Removable_Media   Input_Device   Output_Device onto_class Resource = name :: string onto_class Resource = name :: string onto_class Electronic = Resource + provider :: string onto_class Computer_Hardware = Product + dimensions :: "int set" onto_class Computer_Hardware = Product + type :: Hardware_Type composed_of :: "Product list" invariant c2 :: "Product List" onto_class Simulation_Model = Electronic + type :: string onto_class Informatic = Resource + description :: string definition Computer_Hardware_to_Hardware_morphism ::     "'a Computer_Hardware [1000]99) where "or (Hardware)ComputerHardware [1000]99) where "or (Hardware)ComputerHardware =     ( Resource.tag_attribute = 2::int ,         Resource.tag_attribute</pre>	Storage_Device	mass :: int
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 +         onto_class Informatic = Resource +         definition Computer_Hardware_to_Hardware =         ("- (Hardware]ComputerMardware =         (" Resource.tag_attribute = 2::int ,         Resource.tag_attribute = 2::int ,         Resource.name = numer or ,         Informatic description :: "tromatic = resource +         (mather in the informatic = numer component = numer computer +         intromatic = numer computer +         intormatic = numer computer +	Fixed_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 +         definition Computer_Hardware_to_Hardware.string         inde_class Informatic = Resource +         description :: string         onto_class Informatic = Resource + <t< td=""><td>Removable_Media  </td><td><pre>onto_class Computer_Software = Item +</pre></td></t<>	Removable_Media	<pre>onto_class Computer_Software = Item +</pre>
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         otto_cl	Input_Device	version :: int
onto_class Resource = name :: string       onto_class Electronic_Component = Product + dimensions :: "int set"         onto_class Electronic = Resource + provider :: string manufacturer :: string       onto_class Computer_Hardware = Product + type :: Hardware = Type composed_of :: "Product list" invariant c2 :: "Product List" invariant c2 :: "Product List"         onto_class Simulation_Model = Electronic + type :: string       definition Computer_Hardware_to_Hardware_morphism :: "'a Computer_Hardware" [1000]999) where "σ (Hardware) <sub>ComputerHardware</sub> = ( Resource.tag_attribute = 2::int , Resource.name = name σ , Informatic_description = ''no description'', Hardware.type = Computer_Hardware.type σ ,	Output_Device	
onto_class Resource =       dimensions :: "int set"         name :: string       onto_class Computer_Hardware = Product +         provider :: string       composed_of :: "Product list"         onto_class Component = Electronic +       type :: Hardware_Type         mass :: int       dimensions :: "int list"         onto_class Simulation_Model = Electronic +       type :: string         onto_class Informatic = Resource +       definition Computer_Hardware_morphism ::         ("		<pre>onto_class Electronic_Component = Product +</pre>
<pre>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 ("</pre>	onto_class Resource =	dimensions :: "int set"
<pre>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 Informatic = Resource + description :: string onto_class Informatic = Informatic + type :: Hardware Type () Resource.tag_attribute = 2::int , Resource.name = name σ , Informatic.description = ''no description'', Hardware.type = Computer_Hardware.type σ , </pre>	name :: string	
onto_class flectronic = Resource +       type :: Hardware_lype         provider :: string       composed_of :: "Product list"         onto_class Component = Electronic +       mass :: int         dimensions :: "int list"       definition Computer_Hardware_to_Hardware_morphism ::         onto_class Simulation_Model = Electronic +       "'a Computer_Hardware_scheme ⇒ Hardware"         type :: string       where "σ (Hardware) <sub>ComputerHardware</sub> "[1000]999)         where "σ (Hardware) <sub>ComputerHardware</sub> =       ( Resource.tag_attribute = 2::int ,         Resource.tag_attribute = 0;       Informatic.description = ''no description'',         Hardware Tupe       Hardware Tupe		<pre>onto_class Computer_Hardware = Product +</pre>
<pre>provider :: string manufacturer :: 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 Informatic = Resource + description :: string onto_class Informatic = nmormatic + type :: hardware = n</pre>	onto_class Electronic = Resource +	type :: Hardware_Type
<pre>invariant c2 :: "Product.mass σ = sum(map Product.mass (composed_of σ))" onto_class Component = Electronic +     mass :: int     dimensions :: "int list" onto_class Simulation_Model = Electronic +     type :: string onto_class Informatic = Resource +     description :: string onto_class Informatic = Resource +     description :: string onto_class Informatic +     type :: Hardware Type </pre>	provider :: string	composed_of :: "Product list"
<pre>onto_class Component = Electronic +     mass :: int     dimensions :: "int list" onto_class Simulation_Model = Electronic +     type :: string onto_class Informatic = Resource +     description :: string onto_class Informatic = - Informatic +     type :: Hardware Type = Computer_Hardware to description '',     Hardware Type = Computer_Hardware to descripte o,     type :: Hardware Type = Computer_Hardware.type o,</pre>	manufacturer :: String	invariant c2 :: "Product.mass $\sigma = sum(map Product.mass (composed_of \sigma))"$
<pre>mass :: int dimensions :: "int list" onto_class Simulation_Model = Electronic + type :: string onto_class Informatic = Resource + description :: string onto_class Informatic = Informatic + type :: Hardware Type</pre>	<pre>ontoclass Component = Electronic +</pre>	
<pre>dimensions :: "int list" onto_class Simulation_Model = Electronic +   type :: string onto_class Informatic = Resource +   description :: string onto_class Informatic = Informatic +   type :: Hardware Type </pre> definition Computer_Hardware_morphism ::   "'a Computer_Hardware_scheme  Hardware" [1000]999) where "σ (Hardware) <sub>ComputerHardware</sub> " [1000]999) where "σ (Hardware) <sub>ComputerHardware</sub> =   ([Resource.tag_attribute = 2::int,    Resource.name = name σ,    Informatic.description = ''no description'',    Hardware Type = Computer_Hardware.type σ,    type :: Hardware Type = Computer_Hardware.type σ,	mass :: int	
<pre>definition Computer_Hardware_to_Hardware_morphism ::     onto_class Simulation_Model = Electronic +     type :: string     onto_class Informatic = Resource +     description :: string     onto_class Informatic = Informatic +</pre>	dimensions :: "int list"	
<pre>onto_class Simulation_Model = Electronic +     type :: string     "'a Computer_Hardware_scheme ⇒ Hardware"     ("_ (Hardware)<sub>ComputerHardware</sub>" [1000]999)     where "σ (Hardware)<sub>ComputerHardware</sub> =         ([ Resource.tag_attribute = 2::int,         Resource.name = name σ,         Informatic.description = ''no description'',         Hardware_type = Computer_Hardware.type σ,         Hardware_type = Computer_Hardware.type σ,</pre>		definition Computer_Hardware_to_Hardware_morphism ::
<pre>type :: string ("_ (Hardware)<sub>ComputerHardware</sub>" [1000]999) where "σ (Hardware)<sub>ComputerHardware</sub> =     () Resource.tag_attribute = 2::int ,     description :: string     Resource.name = name σ ,     Informatic.description = ''no description'',     Hardware Type = Computer_Hardware.type σ ,     ""</pre>	<pre>onto_class Simulation_Model = Electronic +</pre>	"'a Computer_Hardware_scheme $\Rightarrow$ Hardware"
<pre>onto_class Informatic = Resource +     description :: string onto_class Hardware - Informatic +     type :: Hardware Type</pre>	type :: string	("_ $\langle$ Hardware $ angle_{ComputerHardware}$ " [1000]999)
<pre>onto_class Informatic = Resource +     description :: string     () Resource.tag_attribute = 2::int ,     Resource.name = name σ ,     Informatic.description = ''no description'',     Hardware Type = Computer_Hardware.type σ ,     in i</pre>		where " $\sigma \ \langle Hardware \rangle_{ComputerHardware}$ =
description :: string Resource.name = name σ , Informatic.description = ''no description'', Hardware.type = Computer_Hardware.type σ , Hardware Type	onto_class Informatic = Resource +	<pre>( Resource.tag_attribute = 2::int ,</pre>
Informatic.description = ''no description'', Hardware_type = Computer_Hardware.type σ ,	description :: string	Resource.name = name $\sigma$ ,
$type :: Hardware Type$ = Computer_Hardware.type $\sigma$ ,		<pre>Informatic.description = ''no description'',</pre>
type :: Hardware Type		Hardware.type = Computer_Hardware.type $\sigma$ ,
Hardware_mass = mass $\sigma$ ,	type :: Hardware_Type	Hardware.mass = mass $\sigma$ ,
mass :: int Hardware.composed_of =	mass :: int	Hardware.composed_of =
composed_of :: "Component List" map Product_to_Component_morphism	composed_of :: "Component List"	<pre>map Product_to_Component_morphism</pre>
Invariant c1 :: "mass $\sigma$ = sum(map component.mass (composed_of $\sigma$ ))" (Computer_Hardware.composed_of $\sigma$ ) "	<b>invariant</b> c1 :: "mass $\sigma$ = sum(map component.mass (composed_of $\sigma$ ))"	(Computer_Hardware.composed_of $\sigma$ ) []"

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

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

"The mapping is correct (preserves the invariants)"

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

"The mapping is correct (preserves the invariants)"

<pre>lemma inv_c2_preserved :     "c2_inv <math>\sigma \implies c1_inv</math> (<math>\sigma \langle Hardware \rangle_{ComputerHardware}</math>)"     unfolding c1_inv_def c2_inv_def</pre>	Isabelle code
Computer_Hardware_to_Hardware_morphism_def	
<pre>Product_to_Component_morphism_def</pre>	
using comp_def by (auto)	
<pre>lemma Computer_Hardware_to_Hardware_morphism_total :     "Computer_Hardware_to_Hardware_morphism ' ({X::Computer_Hardware.</pre>	c2_inv X})

• First of all: how are they processed:

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

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

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

- 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)

- 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)

• Then "built-in" term-anti quotations can be:

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

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

**sort** "typ" **consts** *typ\_anno* :: '*'string* ⇒ *typ"* ("@{*typ\_*}" 100)

- 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)

- 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)

- 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)

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

- 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.

- 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

- 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

- 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 ...

- 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

• (P)IDE's are more than just a technical asset

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

- (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 (???)