Types for (Slow) AI

ITRS & WiL 2024

Viviana Bono
(*joint work with Lorenzo Bafunno, Davide Camino, Emanuele Rovaretto, Carmine Santella, Andrea Zito*)

Dipartimento di Informatica
University of Torino

July 9, 2024

Contents

1. The title, the abstract, the research path
2. A new type system
3. Using a functional language to manipulate knowledge
4. The subtyping issue
5. A possible case-study
6. Some thoughts

Contents

1. The title, the abstract, the research path
2. A new type system
3. Using a functional language to manipulate knowledge
4. The subtyping issue
5. A possible case-study
6. Some thoughts

The Slow AI movement

Main goal of the Slow AI movement[a]

[a] https://datalabnotes.com/slow-ai/

"**Slow AI** is about the responsible creation of data-driven systems through responsible, methodical, and well-documented processes. It is about prioritizing the voices of people impacted by AI systems and about building systems that promote a more fair and inclusive society."

Two important keywords: non-biased dataset, explainability.

Timnit Gebru and DAIR

Timnit Gebru, CEO of DAIR, fired by Google in 2020

Distributed AI Research Institute (DAIR)

DAIR is an interdisciplinary and globally distributed AI research institute rooted in the belief that AI is not inevitable, its harms are preventable, and when its production and deployment include diverse perspectives and deliberate processes it can be beneficial.

Some Semantic Web concepts before following our path into Slow AI

The Semantic Web

The Semantic Web[a] is an extension of the World Wide Web in which published documents (HTML pages, files, images, and so on) are associated with information and data (metadata) that specify their semantic context in a format suitable for querying and interpretation (e.g., via search engines) and, more generally, automatic processing.

[a] The term was coined by Tim Berners-Lee himself. Source: Wikipedia.

Resource Description Framework (RDF)[1]

Annotations describing data and their relationships (*resources*). They are in the form of triples:

$$(subject, predicate, object)$$

Each component is represented by an IRI (Internationalized Resource Identifier), deferential identifiers that point to the "real" resource (e.g. `http://example.org/#Pizza`).
A set of triples represents a directed graph.

$$\{(\text{Margherita}, \textit{type}, \text{Pizza}),$$
$$(\text{Vegetarian}, \textit{eats}, \text{Margherita}),\}$$

$$\equiv$$

[1] https://www.w3.org/RDF/

Ontology Web Language (OWL)[2]

OWL is a formal language that describes ontologies, based on descriptive logics (DL).

Ontology

Formal, shared and explicit representation of a conceptualization.

The DLs separate the representation into two parts, called Box:

- T-Box, defines the vocabulary of the domain (concepts and their characteristics);
  $\Longrightarrow$ subconcept relation: $\text{Student} \sqsubseteq \text{Person} \sqcap \exists \textit{attends}.\textit{Course}$
- A-Box, assertions about individuals, depend on the T-Box.
  $\Longrightarrow$ assertions of:
  1. concept: $\textit{Margherita} : \texttt{Pizza}$
  2. property: $(\textit{Gianni}, \textit{Margherita}) : \textit{eat}$

[2] https://www.w3.org/OWL/

Ontology Web Language (OWL)[2]

The main task of OWL ontologies is to allow structured reasoning and infer new relationships between data.

Example

From the previous example, we can:

- define the concepts of `Pizza`, `Vegetarian` and `Vegetarian Pizza`;
- impose that `Vegetarian Pizza` $\sqsubseteq$ `Pizza`;
- impose that a `Vegetarian Pizza` $\sqsubseteq \exists$*eat*.`Vegetarian`;
- declare that *Margherita* : `Pizza`;
- declare that *Gianni* : `Vegetarian`;
- declare that (*Gianni*, *Margherita*) : *eat*.

We can infer that *Margherita* : `Vegetarian Pizza`.

[2] https://www.w3.org/OWL/

"*AI is not (only) Machine Learning, indeed*", say colleagues in AI

- However, reasoning on knowledge bases is often done by machine learning models fed with RDF data rather than by applying formal inference techniques.
- Faster mechanism than any formal (run time) inference technique, BUT possible impoverishment of the expressivity of the properties proven.
- *Can statically typed languages help to attenuate the performance issue of some inference techniques?*

This is only the beginning of the story...

... and of a light, partial survey on proposals about how types (and functional languages) can be used for the Semantic Web and contribute to the Slow AI movement (at least from the point of view of *explainability*, even though also logical derivations might be obscure...).

Contents

1. The title, the abstract, the research path
2. A new type system
   - A type system for querying RDF graph
3. Using a functional language to manipulate knowledge
4. The subtyping issue
5. A possible case-study
6. Some thoughts

A type system for querying RDF graph: λ_{DL}

The problem

- In software that works on ontologies the reasoning process and error detection are most often both carried out at run-time.
- For a query it is a matter of checking whether it is inhabited, *i.e.*, whether it returns results, or not. Moreover, it is useful verifying whether results are well-typed or not.

A type system for querying RDF graph: λ_{DL}

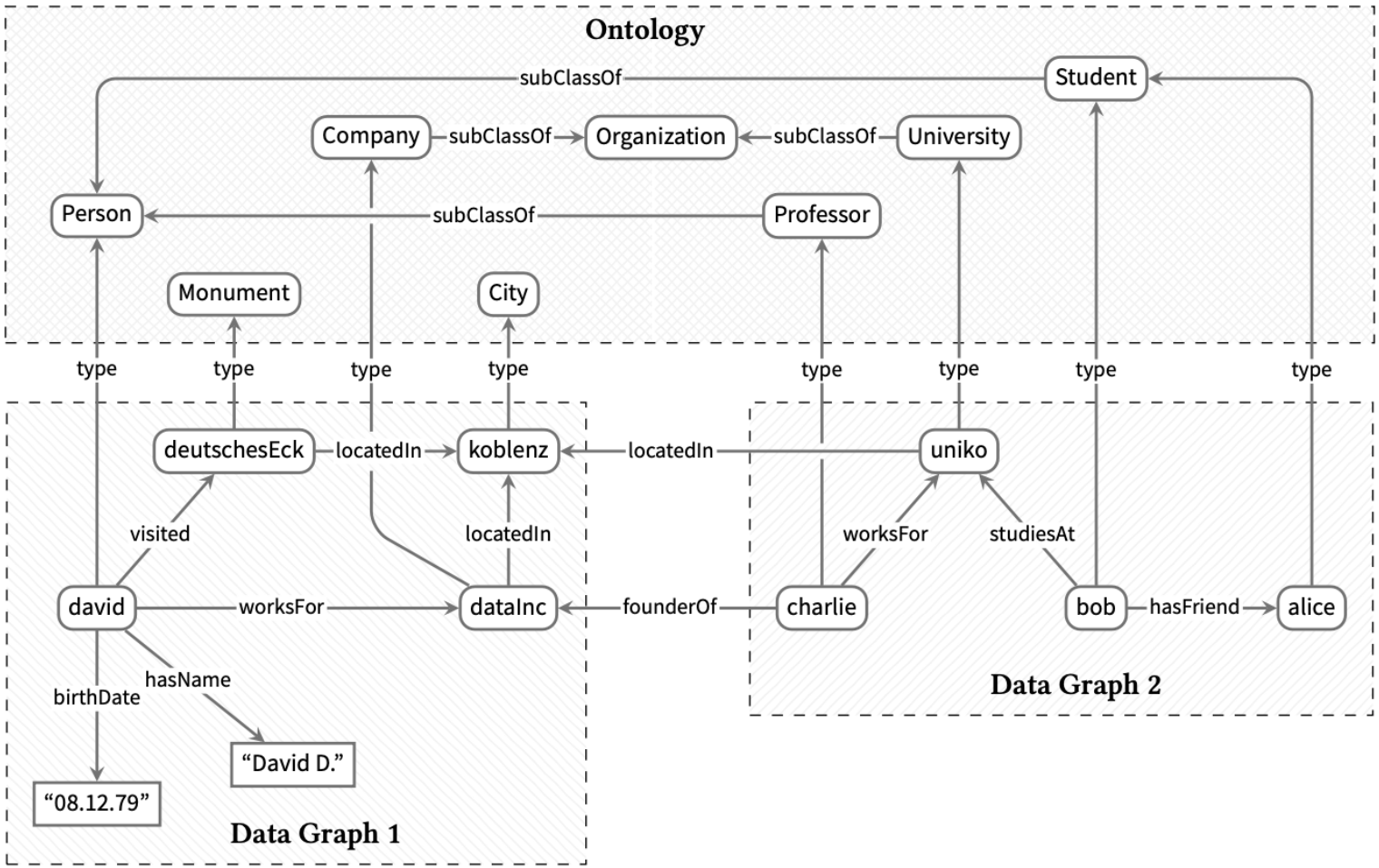
A solution

- Leinberger[1] introduced a typed lambda calculus extended with (1) SPARQL queries to RDF-based ontologies and (2) description logic (DL) constructs as types, called λ_{DL}
- λ_{DL} guarantees to execute SPARQL queries[2] that respect the A-Box and T-Box of the chosen ontology, performing the inhabitation control statically and typing the nodes returned from the queries using the concept expressions of the DL.

[1] [Leinberger, Martin Type-safe Programming for the Semantic Web. PhD thesis, University of Koblenz and Landau, Germany, 2021.]

[2] https://www.w3.org/TR/rdf-sparql-query/

The idea - an RDF-based ontology as example



The idea - some hints

- OWL comprise logical axioms. For example, the axiom $Person \sqsubseteq \exists hasName.T$, built with the two concept expressions *Person* and $\exists hasName.T$, defines that all instances of *Person* must have a name.
- However, OWL ontologies also allow for modeling incomplete knowledge. Even though alice, bob and charlie have no name, the graph as shown in figure is still valid with respect to the axiom.
- The aim is to use concept expressions like *Person* and $\exists hasName.T$ as types representing sets of values, to be assigned to variable in queries in order to avoid runtime errors such as for the SPARQL query:

```
for x in (query SELECT ?x WHERE {?x type Person})
    print x.hasName
    // alice, bob, charlie do not have names
```

A λ_{DL} rule: (T-QUERY)

$$\frac{q : K_q \quad head(q) = \{l_i^{i \in 1 \ldots m}\} \quad \forall x \in vars(q) : K \cup K_q \nvDash A_x \sqsubseteq \bot}{\Gamma, K \cup K_q \vdash query\ q : List\{l_i : A_{l_i}^{i \in 1 \ldots m}\}}$$

- Retrieve the set of axioms K_q containing the information about the atomic concepts A_x for each variable used in the query q.
- Check each variable for satisfiability: in case there is a variable for which $K \cup K_q \models A_x \sqsubseteq \bot$, then this variable is not satisfiable.
- If a query contains an unsatisfiable variable, there cannot be any answers to the query.
- If all variables are satisfiable, construct a record type $\{l_i : A_{l_i}\}$ where ${l_i}^{i \in 1 \ldots m}$ represent the output variables of the query.
- The final type that is assigned to the query term using $K \cup K_q$ is the list of records $\{l_i : A_{l_i}^{i \in 1 \ldots m}\}$.

Contents

1. The title, the abstract, the research path
2. A new type system
3. Using a functional language to manipulate knowledge
   - $\mathbb{C}$Duce: A full-fledged language to manipulate XML
   - $\mathbb{C}$Duce at work
4. The subtyping issue
5. A possible case-study
6. Some thoughts

ℂDuce in a nutshell

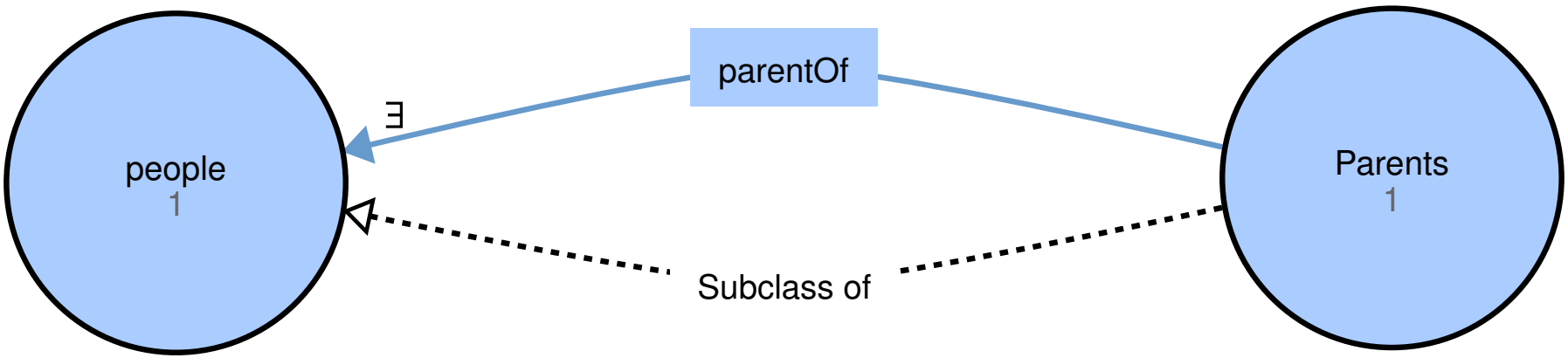
- ℂDuce was born from collaboration between the language research group of ENS[1] and the database research group of LRI[2].
- ℂDuce is a language with the capability to manipulate XML documents, by describing XML elements with accuracy.
- ℂDuce is a statically typed, functional, general purpose language oriented towards the development of applications for XML manipulation. Note: it exploit a form of intersection types, for instance, to type lately-bound overloaded functions.
- In ℂDuce it is possible to define XML elements as types. In ℂDuce an XML element has the general form

$$\langle (\texttt{tag})(\texttt{attr}) \rangle \texttt{content}$$

where `tag`, `attr` and `content` are expressions.

[1] `https://www.ens.psl.eu/`
[2] `https://www.lri.fr/`

An ontology describing people (by Davide - the ℂDuce Wizard) - graph



An XML ontology describing people built with Protégé[3] - part 1

```
<rdf:RDF xmlns="http://www.semanticweb.org/people/"
   xml:base="http://www.semanticweb.org/people/"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xml="http://www.w3.org/XML/1998/namespace"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:people="http://www.semanticweb.org/people#">
  <owl:Ontology rdf:about="http://www.semanticweb.org/people"/>
  <!-- http://www.semanticweb.org/people#parentOf -->
  <owl:ObjectProperty
   ↪  rdf:about="http://www.semanticweb.org/people#parentOf"/>
  <!-- http://www.semanticweb.org/people#Parent -->
  <owl:Class rdf:about="http://www.semanticweb.org/people#Parent">
    <owl:equivalentClass>
      <owl:Restriction>
        <owl:onProperty
         ↪  rdf:resource="http://www.semanticweb.org/people#parentOf"/>
        <owl:someValuesFrom
         ↪  rdf:resource="http://www.semanticweb.org/people#People"/>
      </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf
     ↪  rdf:resource="http://www.semanticweb.org/people#People"/>
  </owl:Class>
```

[3] A free, open-source ontology editor and framework for building intelligent systems, https://protege.stanford.edu/

An XML ontology describing people built with Protégé - part 2

```
23    <owl:Class rdf:about="http://www.semanticweb.org/people#People"/>
24    <!-- http://www.semanticweb.org/people#Bruto -->
25    <owl:NamedIndividual
      ↪  rdf:about="http://www.semanticweb.org/people#Bruto">
26      <rdf:type rdf:resource="http://www.semanticweb.org/people#People"/>
27      <rdfs:comment>Bruto</rdfs:comment>
28    </owl:NamedIndividual>
29    <!-- http://www.semanticweb.org/people#Cesare -->
30    <owl:NamedIndividual
      ↪  rdf:about="http://www.semanticweb.org/people#Cesare">
31      <rdf:type rdf:resource="http://www.semanticweb.org/people#Parent"/>
32      <people:parentOf
        ↪  rdf:resource="http://www.semanticweb.org/people#Bruto"/>
33      <rdfs:comment>Augusto</rdfs:comment>
34      <rdfs:comment>Caio</rdfs:comment>
35      <rdfs:comment>Cesare</rdfs:comment>
36      <rdfs:comment>Giulio</rdfs:comment>
37    </owl:NamedIndividual>
```

ℂDuce: the Ontology type - part 1

```
1 namespace owl="http://www.w3.org/2002/07/owl#"
2 namespace rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3 namespace www="http://www.people#"
4 namespace xml="http://www.w3.org/XML/1998/namespace"
5 namespace xsd="http://www.w3.org/2001/XMLSchema#"
6 namespace rdfs="http://www.w3.org/2000/01/rdf-schema#";;
```

ℂDuce: the Ontology type - part 2

```
8  type Ontology = <rdf:RDF xml:base=String> [ Thing * ]
9  type Thing = Ont | Property | Class | Individual
10
11 type Ont = <owl:Ontology rdf:about=String> []
12
13 type Property = <owl:ObjectProperty rdf:about=String> []
14
15 type Class = <owl:Class rdf:about=String> [ ClassAttr * ]
16 type ClassAttr = EqClass | SubClass
17 type EqClass = <owl:equivalentClass> [ <owl:Restriction> [ AnyXml * ] ]
18 type SubClass = <rdfs:subClassOf rdf:resource=String> []
19
20 type Individual = <owl:NamedIndividual rdf:about=String> [ IndAttr * ]
21 type IndAttr = TypeInd | PropInd | Name
22 type Name = <rdfs:comment> String
23 type TypeInd = <rdf:type rdf:resource=String> []
24 type PropInd = <_ rdf:resource=String> [];;
25
26 let ontology :? Ontology = load_xml "people.rdf";;
```

ℂDuce: how to extract the names of the individuals

```
let fun name (Individual -> [String *])
        <owl:NamedIndividual ..> [(n::Name|_)*] -> map n with
                <rdfs:comment> s -> s;;

let fun names (Ontology -> [String *])
        <rdf:RDF ..> x -> transform x with
                | (x & Individual) -> name x
                | _ -> [];;
```

N.B. The & is the intersection type constructor.

Do we really need ℂDuce?

With tools like Protégé, do we really need ℂDuce?

Our answer is that it can be a companion to such powerful tools. In particular, ℂDuce can help creating new knowledge bases.

Two ongoing experiments - n. 1

Restructuring ontologies of different nature, drawing inspirations from the NeOn's guidelines[4]:

- In ontology creation processes, the current trend is to reuse ontologies as much as possible rather than creating them from scratch.
- These operations are often subject to errors due to inconsistencies or heterogeneities between the information present in the knowledge sources.
- Currently, (more or less) formal tools already exist to facilitate this process, e.g., PROMPT in Protégé.
- Could compositionality of functional languages like $\mathbb{C}$Duce be of help? We think so.

[4] Suárez-Figueroa, M. C. et al, The NeOn Methodology framework: A scenario-based methodology for ontology development. Applied ontology 10, 107–145 (2015).

Two ongoing experiments - n. 2

Starting from very unstructured data such as:

```
<?xml version="1.0"?>
<information>
  <info>
    <subject>Leonardo da Vinci</subject>
    <verb>painted</verb>
    <complement>the Mona Lisa</complement>
    <gender>M</gender>
  </info>
  <info>
    <subject>Jane Austen</subject>
    <verb>wrote</verb>
    <complement>Pride and Prejudice</complement>
    <gender>F</gender>
  </info>
</information>
```

we can write $\mathbb{C}$Duce functions to transform them safely into structured data within formats such as RDF and OWL.

Contents

1. The title, the abstract, the research path
2. A new type system
3. Using a functional language to manipulate knowledge
4. The subtyping issue
5. A possible case-study
6. Some thoughts

Subtyping is a mathematical concept

- $A \to C <: B \to D$ iff $B <: A$ and $C <: D$
- Given $A <: B$, List$\langle A \rangle <:$ List$\langle B \rangle$? NO!
- Contravariance is mathematically correct, but counterintuitive
  $\to$ "Is a cow an animal"?

Is a cow an animal? (An old example - part 1)

```
public class Animal{
  AnyFood food;
  public void feed(AnyFood f){
    food = f;
  }
}
```

Now we introduce Cow as a subclass of Animal. We want to model real world, therefore we specialize co-variantly the food parameter (assuming `VegFood<:AnyFood`) against the contravariance rule.

```
public class Cow extends Animal{
  public void feed(VegFood f){
    food = f;
  }
}
```

Is a cow an animal? (An old example - part 2)

Given:

```
Cow <: Animal, VegFood <: AnyFood,
MeatFood <: AnyFood, VegFood and MeatFood not comparable
```

We can get:

```
Cow c = new Cow();
Animal a = c;
MeatFood m = ...;
a.feed(m); // IT COMPILES, BUT RUNTIME ERROR!
```

INHERITANCE IS NOT SUBTYPING![1] Subtyping is a relation on interfaces, inheritance is a relation on implementations.

[1] William R. Cook, Walter Hill, and Peter S. Canning, Inheritance is not subtyping. POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages December 1989

Roles and subtyping: study-in-progress

- An example of *role* is the student example in which the role Student is associated with players of type Person in the role specification under the context, i.e., of a university.
- *Roles* define *is-a* hierarchies.
- From: *F. Steimann, On the representation of roles in object-oriented and conceptual modelling. Data and Knowledge Engineering 35 (2000)*: *[...] Roles are no subtypes, but viewing them as supertypes poses problems, too. [...]* $\Rightarrow$ Student may be seen as a "supertype" of Person because it can have less attributes (more general state), but also as a "subtype" because it may offer more operations and/or redefine some (more specific behaviour).
- It seems there is no agreement yet in the literature on *consensual role characteristics which cover the fields of knowledge representation, conceptual modeling and object-oriented modeling.*[2]

[2] P. Barlatier, R. Dapoigny, A Type-Theoretical Approach for Ontologies: the Case of Roles. Applied Ontology 20 (2012) 1–3 1 IOS Press

Contents

1. The title, the abstract, the research path
2. A new type system
3. Using a functional language to manipulate knowledge
4. The subtyping issue
5. A possible case-study
   - Types for Functional Requirements for Bibliographic Records (FRBR)
6. Some thoughts

The Functional Requirements for Bibliographic Records (FRBR)

FRBR

The Functional Requirements for Bibliographic Records (FRBR) is a conceptual scheme developed by the International Federation of Library Associations and Institutions (IFLA), created through an entity-relationship model for the purpose of giving a *semi-formal* representation of bibliographic information.

FRBR describes:

- The works.
- The organizations that are responsible for the works.
- The subjects of the works.

Types for FRBR - works

The works are classified according to levels:

- work (work);
- expression (expression);
- manifestation(manifestation);
- item (concrete object).

Example

- Work: The Last of Us Part I (VideoGame), The Last of Us (TV Series).
- Expression: The Last of Us Part I Italian translation and original version.
- Manifestation: The Last Of Us Disc version and digital version.
- Item: Physical (or digital) copy of The Last of Us.

Retrieving duplicates of a work

Example

The two works recorded as "dark (TV)" and "Dark (TV)" are duplicates.

Partial think-to list:

- The levels of classification "work, expression, manifestation, item" do not form a hierarchy: what is/are the correct formal relationships among them?
- It is necessary to choose supporting tools and techniques, such as syntactic and natural-language analysis.
- ⚡ CAUTION ⚡ Retrieving duplicates reminds me of the intersection type constructor seen as proof-theoretic (same term/proof, different types/formulas)...

Contents

1. The title, the abstract, the research path
2. A new type system
3. Using a functional language to manipulate knowledge
4. The subtyping issue
5. A possible case-study
6. Some thoughts

Types: our contribution to Slow AI

- We can enforce forms of (static) correctness on certain procedures by designing new type systems (Re λ_{DL}).
- We can customize existing tools (Re $\mathbb{C}$Duce).
- We can rediscover the power of subtyping (and of sophisticated type theory, such as dependent types).
- ...
- We can talk with those fellows working in AI and not totally happy with the equation "AI = ML".

And a huge "thank you":

- To my (ex-UG) students Lorenzo, Davide, Emanuele, Carmine, Andrea!
- To the ITRS and WiL 2024 committees for having me here!
- And to all of you present at my talk!