An Axiomatic Approach to Deductive Object-Oriented Databases

Alvaro A.A. Fernandes

Department of Mathematical and Computing Sciences

Goldsmiths
UNIVERSITY
OF LONDON

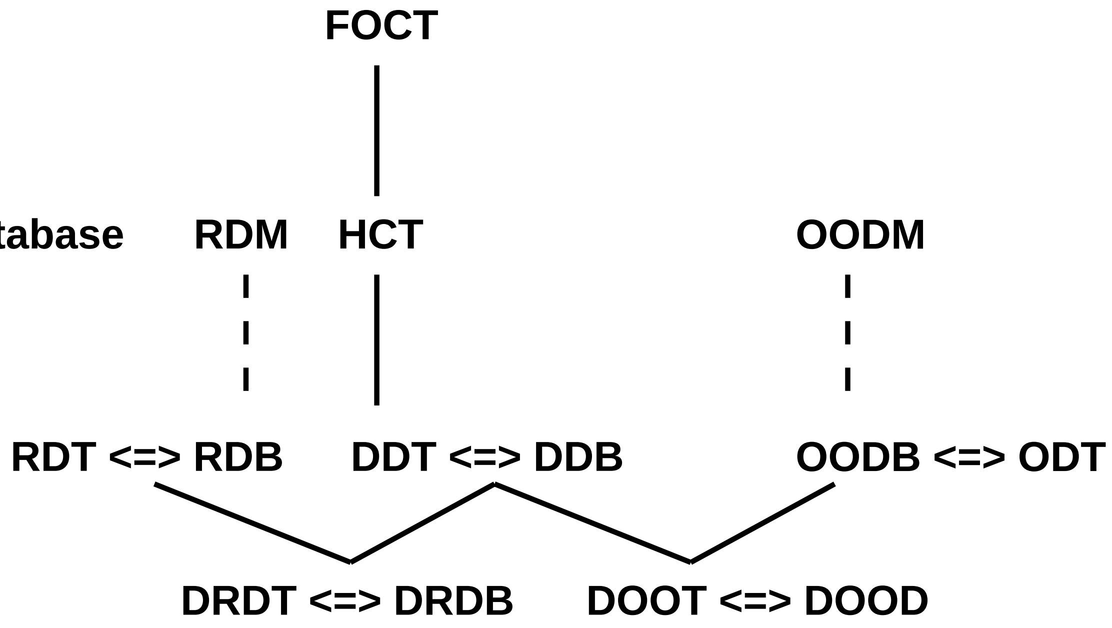
Agenda

1.	**Problem**	Shortcomings of RDBs
2.	**Partial Solutions**	DRDBs and OODBs
3.	**Motivation**	DOOD as DDB$\oplus$OODB
4.	**Technical Background**	Logic and (R)DBs
		Relating Logical Theories
5.	**Approaches to DOODs**	Restart!
		Map!
		Extend!
6.	**Axiomatize!**	Why, and How to
7.	**ROCK & ROLL**	A Practical, Implemented DOOD
8.	**Some Conclusions**	

Goldsmiths UNIVERSITY OF LONDON

Acronyms...

FO for first-order
C for clausal
T for theory
H for Horn
D for data or deductive or database
R for relational
OO for object-oriented
M for model
B for base
O for object



Goldsmiths
UNIVERSITY
OF LONDON

RDB Features (and Lack Thereof) (1)

1	**formal basis**	(+) RDBs are well defined mathematical constructs within a well understood logical system, viz., first-order logic..
2	**user empowering**	(+) RDBs can be queried using declarative languages whose expressions are optimizable and for which there exist tractable evaluation algorithms.
3	**implicit information**	(−) RDBs have limited means of expressing more complex kinds of implicit information (e.g., derived data) about application domains.

Goldsmiths
UNIVERSITY
OF LONDON

RDB Features (and Lack Thereof) (2)

4	**integrity constraints**	(−) RDBs have limited means of expressing more complex kinds of general information (e.g., business rules) about application domains.
5	**recursion**	(−) RDBs do not model structure or behaviour recursively.
6	**identity**	(−) RDBs do not model naturally the detachment of existence from state, i.e., RDBs are inherently value-based, as opposed to identity-based.

RDB Features (and Lack Thereof) (3)

7	**complex structures**	(−) RDBs do not model naturally properties that have nonatomic values.
8	**semantic content**	(−) RDBs do not, in the typical case, model relationships between types such as inclusion, composition, etc.
9	**behaviour modelling**	(−) RDBs have no means of assigning specific behaviour to particular types.

Shortcomings of RDBs

1. The means for expressing implicit information, recursion, and integrity constraints in the sub-languages of the RDM are poor or nonexistent.

2. The modelling constructs of the RDM are either not powerful or not natural enough to capture important kinds of semantic information about certain application domains which are likely to increase in importance (e.g., CAD/CAM/CIM, multimedia, etc.)

Partial Solution (1): DRDBs

DRDBs remedy RDBs' lack of **implicit information**, **integrity constraints** and **recursion**.

Datalog, the archetypical DRDB language, is capable of uniformly expressing implicit information, integrity constraints and recursive queries.

By leaving the underlying data model of RDBs untouched, DRDBs inherit the **formal basis** and **user empowering** qualities of RDBs, but also their lack of **identity**, **complex structures** and **semantic content**.

Partial Solution (2): OODBs

OODBs have **behaviour modelling** as their characteristic quality.

OODBs can also address the lack of **implicit information**, **integrity constraints** and **recursion**, albeit not in a declarative or in as uniform a way as can DDBs.

OODBs typically also major on **identity**, **complex structures** and **semantic content**.

However, OODBs lack a **formal basis** and, by not providing declarative languages, lack **user empowering** features.

Motivation for DOODs

Neither DRDBs alone nor OODBs alone provide a solution for *all* the perceived weaknesses of RDBs.

But, DRDB strengths are OODB weaknesses, and vice-versa.

Hence the best-of-both-worlds idea of DOODs as a class of databases that attempts to combine the complementary qualities of DRDBs and OODBs, i.e., DOOD $=$ DDB$\oplus$OODB.

A silver bullet, of sorts, then. *But*, how do we unify them? Where do we start from? What do we aim at? What are we prepared to trade-off? How do we define concretely what we have abstracted as $\oplus$?

Logic and (Relational) Databases (1)

Model-theoretic View

- A database state is an interpretation structure.
- Query evaluation is truth valuation against a database state.

Goldsmiths UNIVERSITY OF LONDON

Logic and (Relational) Databases (2)

Proof-theoretic View

- A database tuple is a ground, positive, function-free atom in the language induced by the schema.
- A database state is (the axiom set of) a theory.
- Assumptions that databases abide by are formalized as implicit axioms.
- Query evaluation, derivation of implicit information and integrity checking reduce to automated theorem-proving from a database state and the implicit axioms.

Goldsmiths UNIVERSITY OF LONDON

Relating Logical Theories

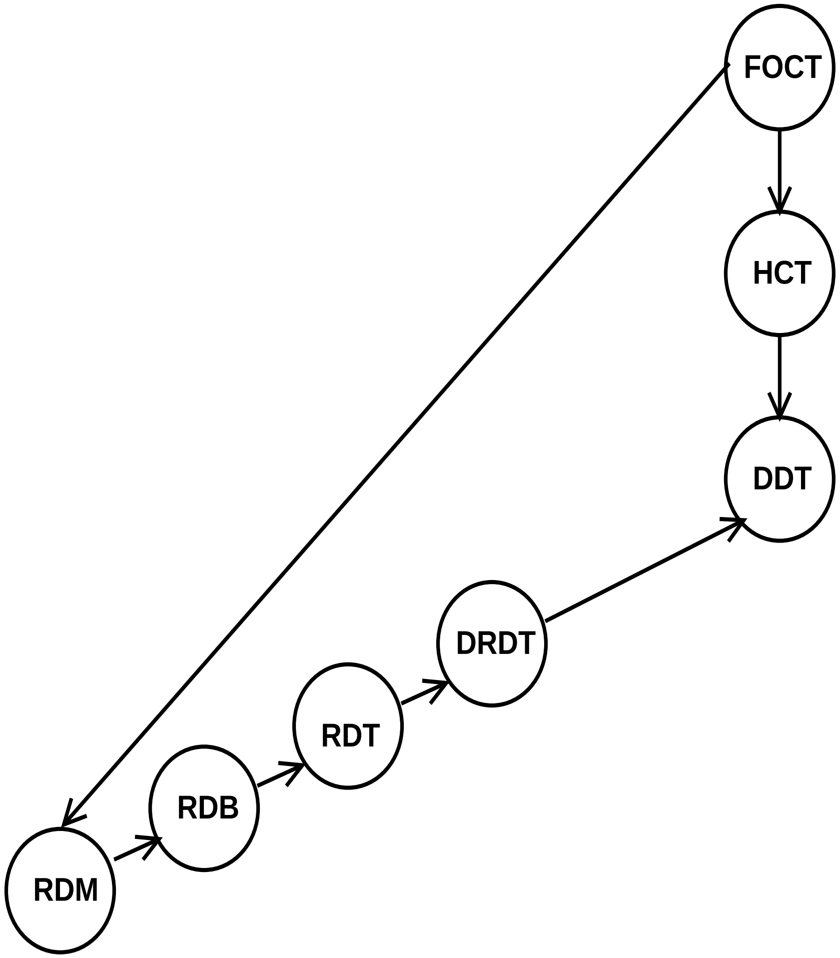
A logical theory is determined by its language, its axiom set and its inference rules.

We can define the map from one class of theories to the other by stating how the target differs from the source in its language, its axiom set or its inference rules.

We can then draw a space of related theories, each node a theory and each arc a map.

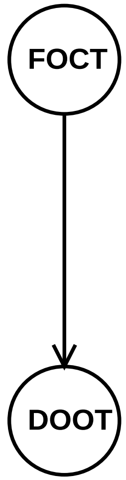
One such space of theories spans around 25 years of research contributions.

A Relational Space of Theories



Approaches to DOOD (1): Restart!

Idea: **Restart** from scratch, no existing automated theorem-proving formalisms can capture DOODs. $\oplus$ is logic invention.

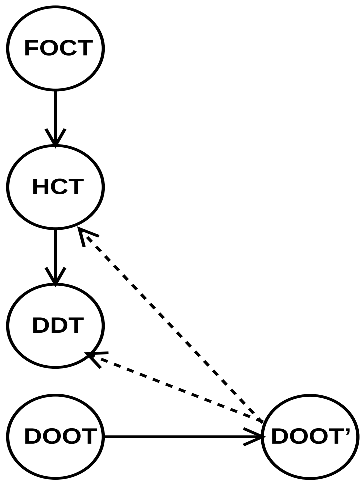


Consequence: Brave new metalogical world, out in the wilderness implementation-wise. The theory space is the sparsest possible.

Goldsmiths UNIVERSITY OF LONDON

Approaches to DOOD (2): Map!

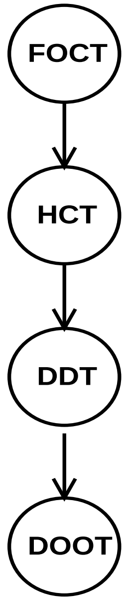
Idea: **Map** DOODs into a known theory. $\oplus$ is language translation.



Consequence: Explains away the metalogic of DOODs by translation to a target language with a known metalogic. However, the metalogic you get may not be the one you want (e.g., Prolog's). The theory space is still very sparse.

Approaches to DOOD (3): Extend!

Idea: **Extend DDBs** with object-oriented concepts. $\oplus$ is language extension.



Consequence: Drops a metalogical goodie (e.g., termination) at each extension step, but the theory space is denser.

Shortcomings, and an Alternative

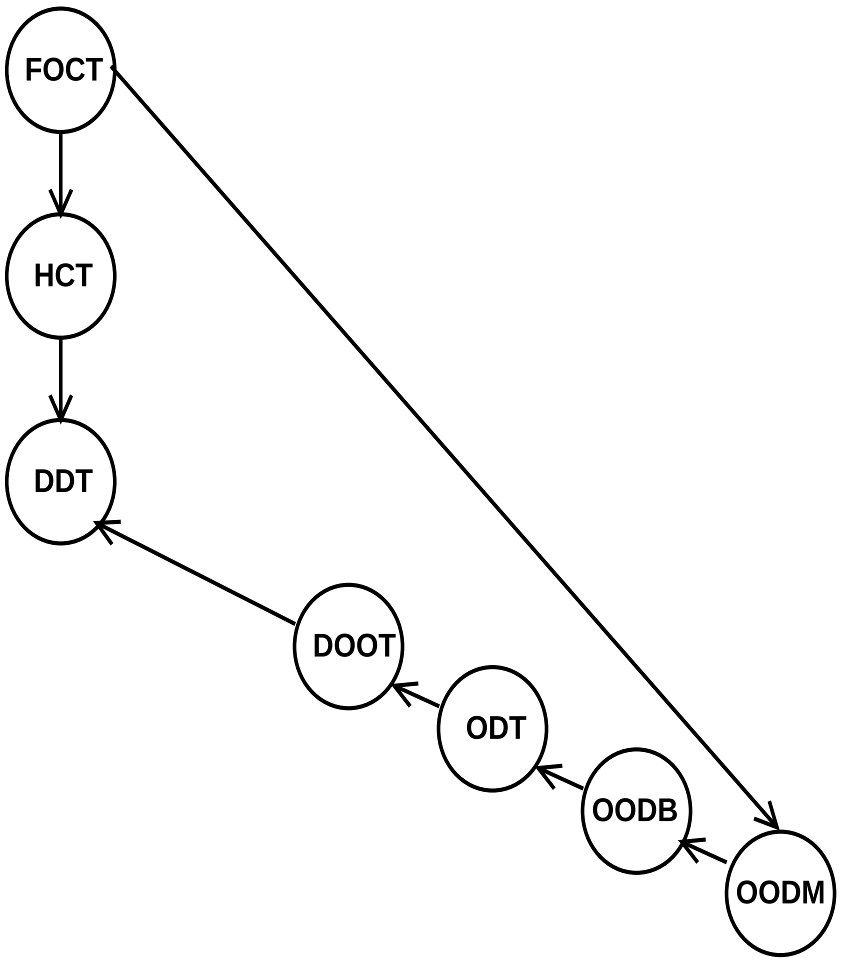
Tinkering with the language introduces intractability or undecidability; tinkering with the inference rules introduces a search space with unacceptable fan-outs from each node.

Idea: Leave the language and the inference rules alone, capture object-orientedness via axioms. $\oplus$ is axiomatization.

Justification: Tinkering with the axiom set leaves the meta-logic as undisturbed as possible.

We can build a theory space that is a mirror image of the relational one. Roughly, whether the underlying model is relational or object-oriented becomes orthogonal to the presence or not of deductive capabilities.

An Object-Oriented Space of Theories



Goldsmiths UNIVERSITY OF LONDON

OODBs as Theories

Start with the following axiom sets in function-free first-order-logic:

$$
\begin{aligned}
\textit{database} &= \textit{state} \cup \textit{object-modelling assumptions} \\
\textit{state} &= \textit{schema} \cup \textit{instance} \\
\textit{object-modelling assumptions} &= \textit{typing} \cup \textit{completion} \cup \textit{exception}
\end{aligned}
$$

typing axiomatizes the many-sorted nature of schemas that are partitioned into object classes; *completion* axiomatizes relationships like **a-kind-of**, **a-part-of**, **an-instance-of**, etc., and *exception* axiomatizes the domain-independent well-formedness conditions on *state*.

This yields a formal structural OODM which has as instances OODBs that are logical theories.

Supporting Side-Effect Free Behaviour

Translate to clauses so as to yield function-free Horn clauses only.

Then, use logic programming (hence recursion) to implement side-effect-free methods (hence declarative, optimizable)

$$\begin{array}{rcl} \mathit{schema} & \supset & \mathit{interface\text{-}declarations} \\ \mathit{instance} & \supset & \mathit{implementation\text{-}declarations} \cup \mathit{implementation\text{-}definitions} \end{array}$$

This extends the structural OODM with side-effect free behaviour modelling facilities.

Moving towards Pure DDBs

Rearrange to make possible classical recursive-query evaluation plus integrity enforcement, then extend with implicit information and application-specific integrity constraints:

$$\begin{aligned}
\textit{extensional-DB} &= \textit{schema} \cup (\textit{instance} \setminus \textit{implementation-definitions}) \\
\textit{intensional-DB} &= \textit{completion} \cup \textit{implementation-definitions} \\
\textit{constraint-DB} &= \textit{typing} \cup \textit{exception}
\end{aligned}$$

$$\begin{aligned}
\textit{extensional-DB'} &= \textit{extensional-DB} \\
\textit{intensional-DB'} &= \textit{intensional-DB} \cup \textit{intensionally-defined-data} \\
\textit{constraint-DB'} &= \textit{constraint-DB} \cup \textit{integrity-constraints}
\end{aligned}$$

This yields a Datalog-equivalent class of object-oriented DDBs.

Encapsulation and Late Binding

Define a language, of body literals only, with the alphabet of *interface-declarations* (hence encapsulation).

Define the selection function (required by SLD-resolution) to emulate late binding (e.g., select from *implementation-definitions* which head literal to resolve against depending on the type of the message recipient).

This yields a logic-based behavioural OODM with state transitions whose instances are DOODs.

ROCK & ROLL (1)

The OM object manager implements the OODM and plays the role of *extensional-DB*.

ROLL clause sets determine *intensionally-defined-data*. ROLL is syntactic-sugar for the DOOD language induced by the axiomatization.

ROLL is parsed into call graphs which are optimized and translated into relational-algebraic query execution plans for bottom-up evaluation over the state kept by OM.

ROCK & ROLL (2)

ROLL is not computationally complete: it has no side-effects and, with no function-symbols, cannot create temporary data structures. ROLL is for querying only.

ROCK is a fully-fledged persistent programming language that imports the same OM type system used by ROLL.

Because the type system is shared and the evaluation protocol is the same, data and control flow to and from ROCK and ROLL without impedance.

ROCK & ROLL approximates a platform for mixing imperative and declarative styles of programming over an object-oriented model of structure and behaviour.

Some Conclusions

- Retracing the steps in the construction of the relational space has led to the creation of its object-oriented mirror image.
- No need to invent a new logical system, recycle an existing one.
- Tinkering with the language and the inference rules is the way to metalogical heaven and implementational hell.
- Axioms might have more to them than people thought.
- Tried and tested technologies for implementation.
- Extensible in many directions suggested by logic programming theory.

For more information, please start from

`http://www.gold.ac.uk/~mas01aaf/`

Goldsmiths
UNIVERSITY
OF LONDON