The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

Object-Oriented Relations

- "Object-orientation" is about *instantiation*.
- Instantiation is needed for programming language "functions" with *state*.
- Such states are values of stored variables, or *attributes*.
- Database relations are stored values: a state could be a tuple.
- It could also be a tuple of a nested relation; or a subrelation.
- Thus, nesting is usually a part of OODB, called *complex objects*.
- Aldat has nesting, in ways efficient for secondary storage.
- Aldat also can instantiate, in bulk, computations with state.
- Instances form *classes*.
- Classes may contain each other: *inheritance* can save code.

Relations as Classes

relation *Couch*(*Id*, *Length*, *Width*);
relation *Chair*(*Id*, *Base*);
relation *Furniture*(*Id*, *Manuf*);
Couch **isa** *Furniture*;
Chair **isa** *Furniture*;

Now the projection,

[*Manuf*] **in** *Couch*

is syntactic sugar for

[*Manuf*] **in** (*Couch* **natjoin** *Furniture*)

and similarly for any other use of *Manuf* in *Couch* or *Chair*

Furniture		*Couch*			
(*Id*	*Manuf*)	(*Id*	*Length*	*Width*)	*Manuf*
1	Mobel	1	15	5	Mobel
2	Furn	2	17	5	Mobel
3	Mobel	3	18	6	Mobel
21	Mobel				
22	Furn	*Chair*	(*Id*	*Base*)	*Manuf*
			21	4	Mobel
			22	5	Furn

Inheritance as join

- Inheritance could also be *implemented* as a join,

- but most such O-O uses are *low activity* operations,

- and best implemented with pointer dereferencing instead of the full join.

- So the special case of O-O gets a special implementation:

- another example of syntactic sugar hiding specialized algorithms and data structures,

- although *defined* in terms of the general operator.

Inheritance as join

Here is a variant.

relation *Couch*(*Id*, *Length*, *Width*);
relation *Furniture*(*Fid*, *Manuf*);
Couch [*Id* **isa** *Fid*] *Furniture*;

[*Manuf*] **in** *Couch*
is syntactic sugar for
[*Manuf*] **in** (*Couch* [*Id* **natjoin** *Fid*] *Furniture*)

- So **isa** translates directly into a precise, if possibly complex, specification for natural join.
- Inclusion dependence,
  [*Id*] **in** *Couch* $\subseteq$ [*Fid*] **in** *Furniture*
  is not guaranteed: this would move **isa** from a purely syntactic specification to a *semantic* constraint.

Attaching Computations to Classes—— Polymorphism (a sketch)

comp *PolyArea*(*Area*) **is**
{ **comp** *Area*(A) **is**
  {A <– *Length*×*Width*;}; ≪ public variables
} **also**
{ **comp** *Area*(A) **is**
  {A <– *Base***2;}; ≪ —not hidden
};

FurnMethod <– *Furniture* **natjoin** *PolyArea*;
Couch **isa** *FurnMethod*;
Chair **isa** *FurnMethod*;
let *FootPrint* **be** *Area*{}
CouchPrint <– [*Id*, *FootPrint*] **in** *Couch*;
ChairPrint <– [*Id*, *FootPrint*] **in** *Chair*;

FurnMethod

(*Id*	*Manuf*	*Area*)
1	Mobel	:
2	Furn	:
3	Mobel	:
21	Mobel	:
22	Furn	:

CouchPrint

(*Id*	*FootPrint*)
1	75
2	85
3	108

ChairPrint

(*Id*	*FootPrint*)
21	16
22	25

