Strategic Term Rewriting and Its Application to a VDM-SL to SQL Convertion

T.L. Alves, P.F. Silva, J. Visser,
and J.N. Oliveira

Presentation: Nuno Luz

October 2010

Content

- **Aims**
- **Motivations**
- **Strategic Term Rewriting**
- **Database Design by Calculation**
- **The VooDooM Tool**
- **Conclusions and Future Work**

Aims

- Design a tool capable of **calculating database schemas** which infers SQL relational meta-data **from abstract data models** specified in VDM-SL formal modeling notation.

Motivations

- The need for splitting software design in two steps [1]:

  1. **Formal Specification** (mathematical representation of the software + analysis and validation);
  2. **Implementation** (production of machine code).

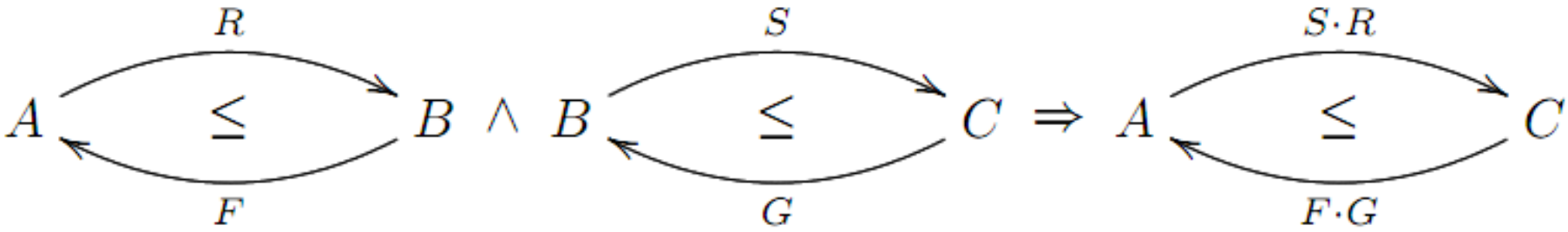
- "Improve Practice Through Theory" (HASLabs' lemma).

Strategic Term Rewriting

- **Term Rewriting** is the transformation of terms according to a set of rewrite rules of the form $x \Rightarrow y$
- **Strategic** means:
  - The specification of a rewriting strategy (e.g. leftmost-innermost) without affecting the rewriting rules
  - Separation of concerns, reusability, and understandability
- Used to **separate the individual conversion rules from the strategy** of applying them to the abstract syntax terms

Database Design by Calculation (1)

- *Data refinement (or reification) by calculation* strategy [2, 3] consisting of inequations of the form:



(read: "data type B implements, or refines data type A")

- ***R**: representation relation (injective and total);*
- ***F**: abstraction relation (surjective function).*

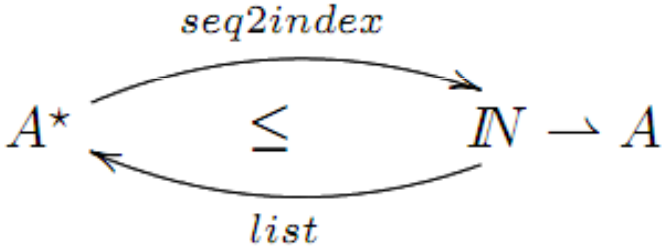
$$F \cdot R = id_A$$

Database Design by Calculation (2)

- This suggests that one may calculate **implementations from specifications:**

Spec = X ≤ X' ≤ X" ≤ ... ≤ Impl

- The inequations represent **convertion laws**, e.g.

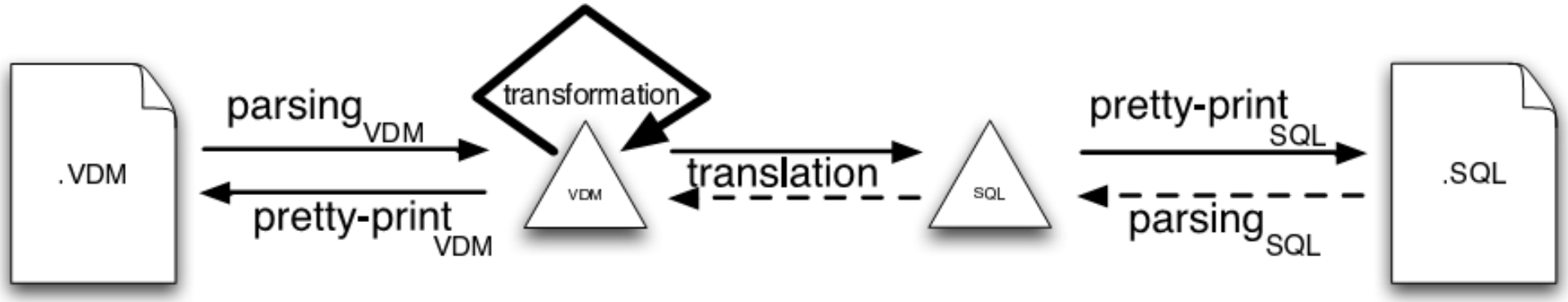


$$seq2index([a,b,a]) = \{1 \mapsto a, 2 \mapsto b, 3 \mapsto a\}$$
$$list(\{11 \mapsto a, 12 \mapsto b, 33 \mapsto a\}) = [a,b,a]$$

which are the basis for the **rewriting rules** applied by VooDooM using Strategic Term Rewriting.

The VooDooM Tool (1)

- Uses *Strategic Term Rewriting* to apply the *database design by calculation* refinement laws to VDM-SL source code
- Relies on *Strafunski* and *SDF* (Syntax Definition Formalism) software bundles, and the *DrIFT* tool



The VooDooM Tool (2)

1. **Recognize a VDM-SL specification file** and convert it to an *Abstract Syntax Tree* (AST)
2. **Transformation** of the AST into its relational equivalent, i.e. a *refinement of the original model using the following rewriting rules*:

Function	Rewrite rule
seq2index	$A^\star \Rightarrow \mathbb{N} \rightarrow A$
unconjoin	$A \rightarrow (B + C) \Rightarrow (A \rightarrow B) \times (A \rightarrow C)$
distr	$A \times (B + C) \Rightarrow (A \times B) + (A \times C)$
set2fm	$2^A \Rightarrow A \rightarrow 1$
opt-elim	$A + 1 \Rightarrow 1 \rightarrow A$
unpeither	$(B + C) \rightarrow A \Rightarrow (B \rightarrow A) \times (C \rightarrow A)$
unnjoin	$A \rightarrow (B \times (C \rightarrow D)) \Rightarrow (A \rightarrow B) \times (A \times C \rightarrow D)$
rec-elim	$\mu \mathsf{F} \Rightarrow (K \rightarrow \mathsf{F}\, K) \times K$

3. **Output** the transformed specification either as VDM-SL or SQL

The VooDooM Tool – Transformation (1)

Example:

```
types
  BAMS       = map AccId to Account;
  Account   :: H: set of AccHolder
               B: Amount;
  AccId      = seq of char;
  AccHolder = seq of char;
  Amount     = int
```

1. **Inlining and Recursion Removal** (uses rec-elim)

```
types
  BAMS = map compose AccId of seq of char end to
             compose Account of
           H: set of compose AccHolder of seq of char end
           B: compose Amount of int end
         end
```

The VooDooM Tool – Transformation (2)

2. **Desugaring** (uses seq2index, set2fm and opt-elim)

```
types
  BAMS = map compose AccId of seq of char end to
             compose Account of
           H: map compose AccHolder of seq of char end to NIL
           B: compose Amount of int end
         end
```

3. **Conversion to Relational** (uses unconjoin, unpeither and unnjoin)

```
types
  BAMS = compose mapAggr of
           map compose AccId of seq of char end
            to compose Amount of int end
           map compose tuple of
                    seq of char
                    seq of char
               end
            to NIL
         end
```

4. **Resugaring**

The VooDooM Tool – SQL Translation

```
types
  BAMS = compose mapAggr of
           map compose AccId of seq of char end
            to compose Amount of int end
           map compose tuple of
                    seq of char
                    seq of char
                end
            to NIL
         end
```

```
CREATE TABLE table1 (
 AccId VARCHAR (128) NOT NULL,
 Amount INT NOT NULL,
 PRIMARY KEY (AccId)
)
```

```
CREATE TABLE table2 (
 Attr1 VARCHAR (128) NOT NULL,
 Attr2 VARCHAR (128) NOT NULL,
 PRIMARY KEY (Attr1, Attr2)
)
```

Conclusions and Future Work

- Conclusions:
  - The VooDooM tool derives database SQL concrete implementations from arbitrarily complex (as far as VDM-SL data constructors are concerned) formal specifications, including recursive data types
  - Promotes the view of database design as a special case of data refinement

- Future Work:
  - Support the reverse process (relational to algebraic)
  - Offer better support for VDM-SL invariants

References

- 1. C. Necco. Polytypic data processing. Master's Thesis, Facultad de Cs. Físico Matemáticas y Naturales, University of San Luis, Argentina, 2005. (Submitted)
- [2] J.N. Oliveira. A reification calculus for model-oriented software specification. Formal Aspects of Computing, 2(1):1-23, April 1990.
- [3] J.N. Oliveira. Software reification using the SETS calculus. In Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK, pp. 140-171. Springer-Verlag, 8-10 January 1992. (Invited Paper).