Microsoft Research

Maintaining Database Integrity with Refinement Types

Johannes Borgström

Joint work with Ioannis Baltopoulos (Cambridge University)
and Andrew D. Gordon (MSR)

Revisiting Transaction Verification

- Can this transaction break a database invariant?
- Lots of prior art in this area
  - Gardarin & Melkanoff ‘79
  - Sheard & Stemple ‘89
  - Benzancen & Doucet ’95
  - …
- Can Refinement types and SMT solvers help?
  - Coverage & speed

Static Analysis Framework

- Ensure that user code performs enough checks
  - Database errors should not occur because of bad queries
- Guarantee preservation of database invariants
  - Row constraints
  - Integrity constraints
  - Other user-defined constraints
- Focus on insert, update, delete

A Table of Marriages

Symmetry, Antireflexivity, Monogamy

```
CREATE TABLE [Marriage](
    [Spouse1] [int] NOT NULL UNIQUE,
    [Spouse2] [int] NOT NULL,
  CONSTRAINT [PK_Marriage]
      PRIMARY KEY ([Spouse1],[Spouse2]),
  CONSTRAINT [FK_Marriage]
      FOREIGN KEY ([Spouse2], [Spouse1])
      REFERENCES [Marriage] ([Spouse1], [Spouse2]),
  CONSTRAINT
    [CK_Marriage] CHECK (NOT([Spouse1] = [Spouse2]))
)
```

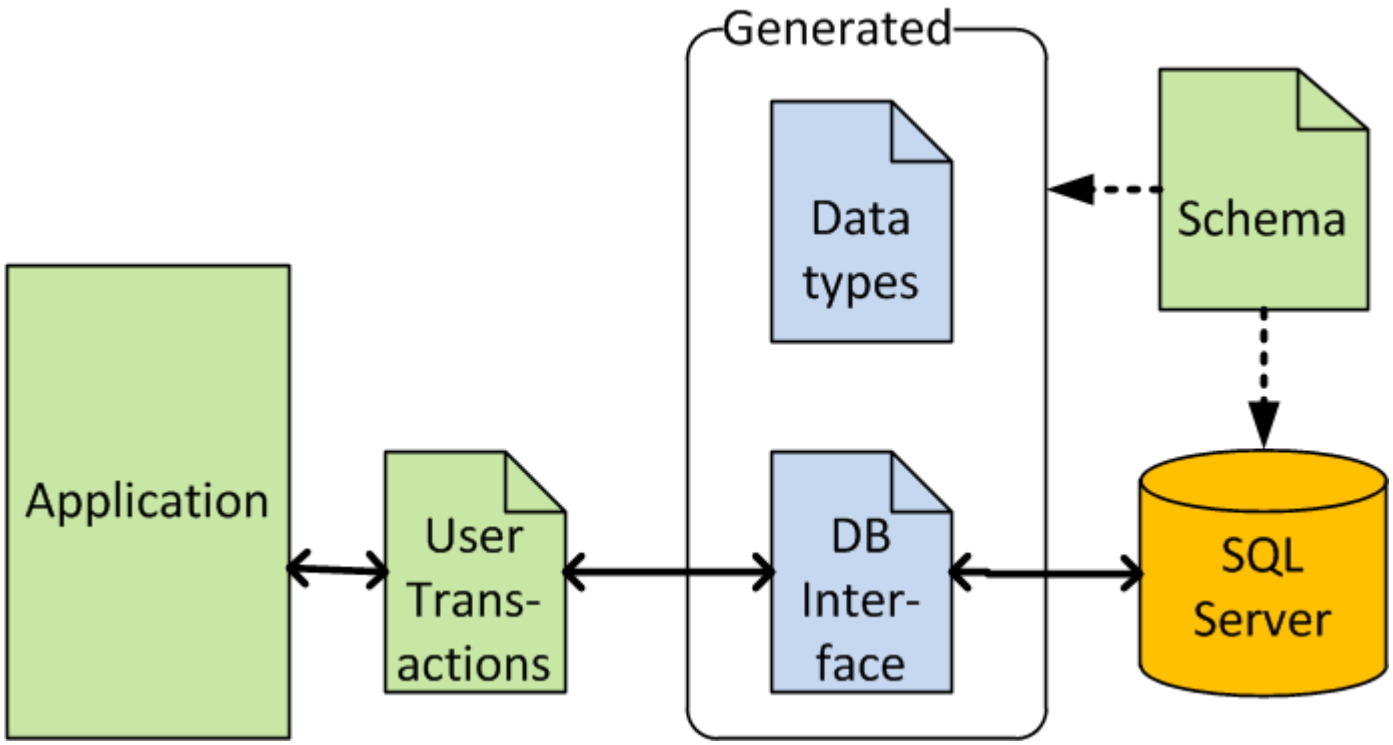
An Abstract Implementation

```
type marriage_row = { m_Spouse1:int; m_Spouse2:int; }
type db = { marriages: marriage_row list; }
```

- Rows = records
- Tables = lists of rows
- Database = record of tables

- Database operations = list processing
  - Cf. Wadler and Trinder 1989

System Overview



Generated DB Interface

- Create (insert) individual rows
- Read a row based on primary key
- Update row based on primary key
- Delete row based on primary key
- Helper functions
  - Generate fresh key, check if a key exists,…
- (Plus simple queries in stored procedures)

Two Transactions: Marry & Divorce

```
let marry_ref (spouse1,spouse2) =
   if hasKeyMarriage(spouse1, spouse2)
   then Some(false)
   else if spouse1 = spouse2
   then Some(false)
   else
      begin
         insertMarriageRowi
   {m_Spouse1=spouse1; m_Spouse2=spouse2;};
         insertMarriageRowi
   {m_Spouse1=spouse2; m_Spouse2=spouse1;};
         Some(true)
      end
let marry m = doTransact marry_ref m
```

```
let divorce_ref (spouse1,spouse2) =
   if hasKeyMarriage(spouse1, spouse2) then
      begin
         deleteMarriagePK(spouse1, spouse2);
         deleteMarriagePK(spouse2, spouse1);
         Some(true)
      end
   else Some(false)
let divorce m = doTransact divorce_ref m
```

Predicates

- Every SQL constraint becomes a predicate
- User-defined predicates possible
- Constraints at three different levels

```
assume (∀m. (CK_Marriage(m) ⇔ m.m_Spouse1 ≠ m.m_Spouse2))

assume (∀l. (Unique_Marriage_Spouse1(l) ⇔
  (∀x,y. (((Mem(x, l) ∧ Mem(y, l)) ∧
          x.m_Spouse1 = y.m_Spouse1) ⇒ x = y))))

assume (∀xs. PK_Marriages(xs) ⇔
          (∀x,m. Mem(x, xs) ∧ Mem(m, xs) ∧
                 x.m_Spouse1 = m.m_Spouse1 ∧
                 x.m_Spouse2 = m.m_Spouse2 ⇒ x = m))

assume (∀marriages', marriages.
  (FK_Marriages_Marriages(marriages, marriages') ⇔
    (∀x. (Mem(x, marriages') ⇒
      (∃u. (Mem(u, marriages) ∧
        (u.m_Spouse1, u.m_Spouse2) =
          (x.m_Spouse2, x.m_Spouse1)))))))

assume (∀d. (FK_Constraints(d) ⇔
  FK_Marriages_Marriages(d.marriages, d.marriages)))
```

Three Levels of Predicates

assume $(\forall m.\ (CK\_Marriage(m) \Leftrightarrow m.m\_Spouse1 \neq m.m\_Spouse2))$

assume $(\forall l.\ (Unique\_Marriage\_Spouse1(l) \Leftrightarrow$
$(\forall x,y.\ (((Mem(x, l) \wedge Mem(y, l)) \wedge$
$x.m\_Spouse1 = y.m\_Spouse1) \Rightarrow x = y))))$

assume $(\forall d.\ (FK\_Constraints(d) \Leftrightarrow$
$FK\_Marriages\_Marriages(d.marriages, d.marriages)))$

Invariant Specification

```
type marriage_row_ref = m:marriage_row {CK_Marriage(m)}
type marriages_ref = marriages:marriage_row_ref list
   {(PK_Marriages(marriages) ∧
     Unique_Marriage_Spouse1(marriages))}

type State = { marriages: marriages_ref }
type State_ref = d:State {FK_Constraints(d)}
```

- Rows carry check constraints
- Tables carry PK and uniqueness constraints
- The database carries foreign key constraints
  - These may be invalidated inside transactions (deferred checking)

Typing a transaction

- Computation types resembling Hoare triples.

```
val divorce_ref: (int ×int) →
    [(s) FK_Constraints(s) ] x:bool option
    [(t) x≠None ⇒ FK_Constraints(t) ]
```

- If the foreign key constraints hold on entry, the function terminates, and returns some value, then the foreign key constraints hold on exit.

(Translated into a state-passing refined function)

Transactions

```
type (α,β)transact = α → State_ref → (β × State_ref)
val doTransact: (α → [(s)FK_Constraints(s)] x:β option
      [(t)x ≠ None ⇒ FK_Constraints(t)]) → (α,β option)transact
```

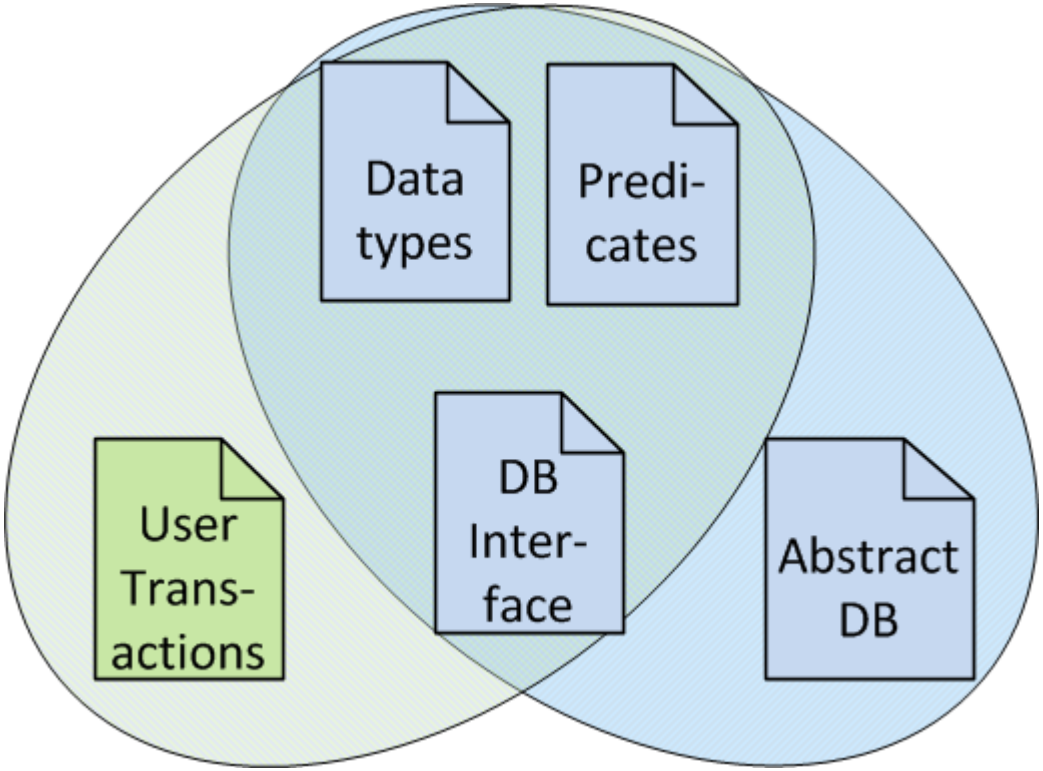
- Type **transact** is a transaction preserving the DB invariant.
- Function **doTransact** wraps another function
  in a database transaction (and asserts db invariant).

```
val divorce_ref: (int × int) →
      [(s) FK_Constraints(s) ] x:bool option
      [(t) x≠ None ⇒ FK_Constraints(t) ]
```

- Function **doTransact divorce_ref** can be safely called
  by untrusted code – guaranteed to preserve invariant.

Microsoft Research

Type-checking for Invariant Safety



Other Examples

- Simple web shopping cart
  - Ensures referential integrity
- Tree with a simple cross-row constraint
  - Field ordering compatible with topological ordering
- Tree with more complex cross-row constraint
  - Caching the sum of all values on the path to the root
  - (in progress)

The Refined Imperative Fixpoint Calculus

A STATEFUL LANGUAGE

Refined Imperative FPC

s, x, y, z	variable
$h ::= \mathsf{inl} \mid \mathsf{inr} \mid \mathsf{fold}$	value constructor
$M, N ::= x \mid () \mid (M, N) \mid h\, M \mid \mathbf{fun}\, x \to A$	value
$A, B ::=$	expression
$\quad M$	$\quad$ value
$\quad M\ N$	$\quad$ application
$\quad M = N$	$\quad$ syntactic equality
$\quad \mathbf{let}\ x = A\ \mathbf{in}\ B$	$\quad$ let
$\quad \mathbf{let}\ (x, y) = M\ \mathbf{in}\ A$	$\quad$ pair split
$\quad \mathbf{match}\ M\ \mathbf{with}\ h\, x \to A\ \mathbf{else}\ B$	$\quad$ constructor match
$\quad \mathbf{get}()$	$\quad$ get current state
$\quad \mathbf{set}(M)$	$\quad$ set current state
$\quad \mathbf{assume}\ (s)C$	$\quad$ assumption of formula C
$\quad \mathbf{assert}\ (s)C$	$\quad$ assertion of formula C

FPC

Operational Semantics

Configurations (E, N, S) where N is the state and S the assumed formulas.

$$
\begin{aligned}
&(\mathbf{get}(), N, S) \longrightarrow (N, N, S)\\
&(\mathbf{set}(M), N, S) \longrightarrow ((), M, S)\\
&(\mathbf{assume}\ (s)C, N, S) \longrightarrow ((), N, S \cup \{C\{^{N}/_{s}\}\})\\
&(\mathbf{assert}\ (s)C, N, S) \longrightarrow ((), N, S)
\end{aligned}
$$

(Plus standard functional operational semantics)

Types

$T, U, V ::=$	(value) type
α	type variable
unit	unit type
$\Pi x : T.\ F$	dependent function type
$\Sigma x : T.\ U$	dependent pair type
$T + U$	disjoint sum type
$\mu\alpha.T$	iso-recursive type

$F, G ::=$	computation type
$\{(s_0)C_0\}\, x{:}T\, \{(s_1)C_1\}$	

If E started in state s_0 satisfying C_0, and returned x, then x has type T and the resulting state s_1 satisfies C_1.

Safety Theorem

- Configuration (E, M, S) has failed iff

  $E = R[\text{assert}(s)C]$ and not $S \vdash C\{M/s\}$.

  (E is the program, M is the state, S is assumed formulas)

- (E, M, S) is safe iff no failed configuration is reachable from it.

- Theorem: If $\emptyset \vdash E : \{(s)\text{True}\}\ x{:}T\ \{(t)\text{True}\}$ and

  $\emptyset \vdash M : \text{state}$ then $(E, M, \emptyset)$ is safe.

Type Checking

- By compilation to RCF, using F7
- Logical queries handled by the SMT solver Z3
  - Theory of equality with uninterpreted functions
- Non-trivial type inference problem
  - Completeness vs. formula size
  - Polymorphism

Conclusions

- Refinement types are useful in checking standard SQL constraints
  - Structure of constraints conform well to refinements
  - Equality, function constraints, simple arithmetic
- Simple user-defined cross-row invariants can be checked
  - Problem with inductive properties b/c of 1st order logic
- More work needed…

Questions?

Thanks for your attention!