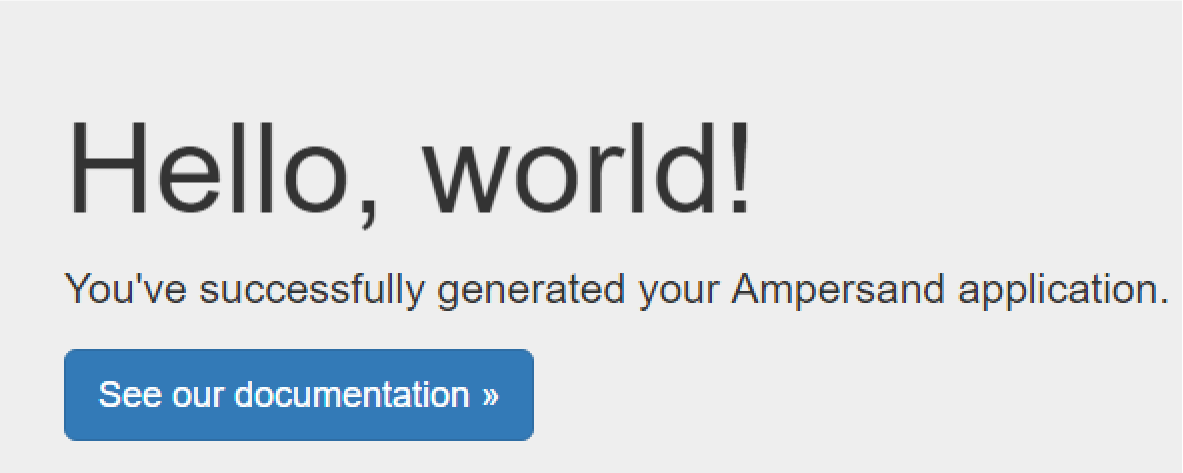
Ampersand with EFA

An automated way to restore system invariants

Presentation Summary

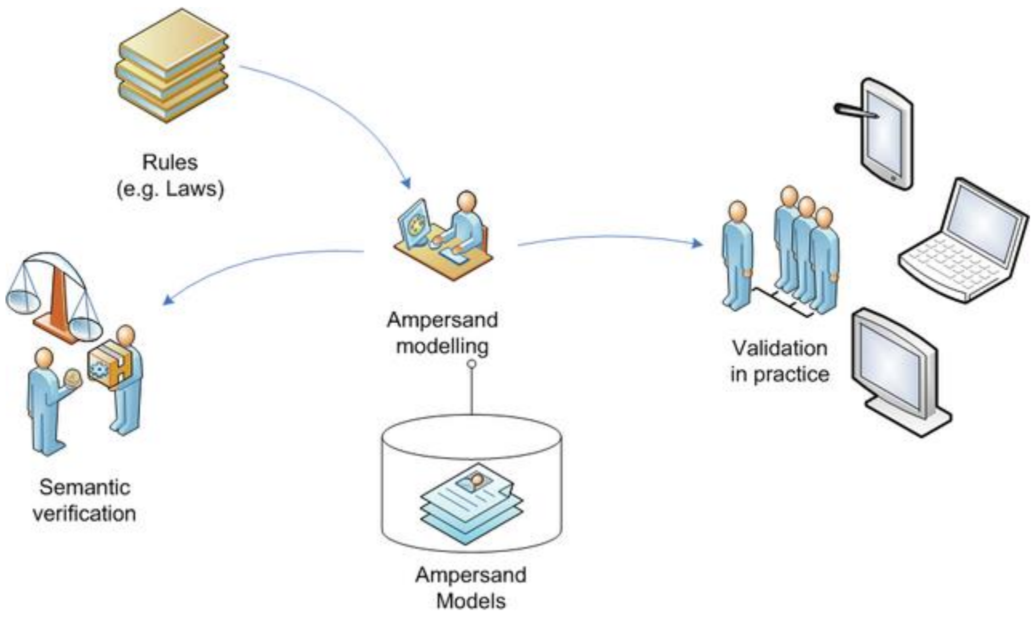
- INTRODUCTION
  - Ampersand System
  - EFA (ECA For Ampersand)
- SOFTWARE IMPLEMENTATION
  - Design Flow
  - Code Generation
  - Abstraction Barriers, Data types, Kinds
  - Testing for Properties
- TESTING
  - Module and System testing
  - MySQL Testing

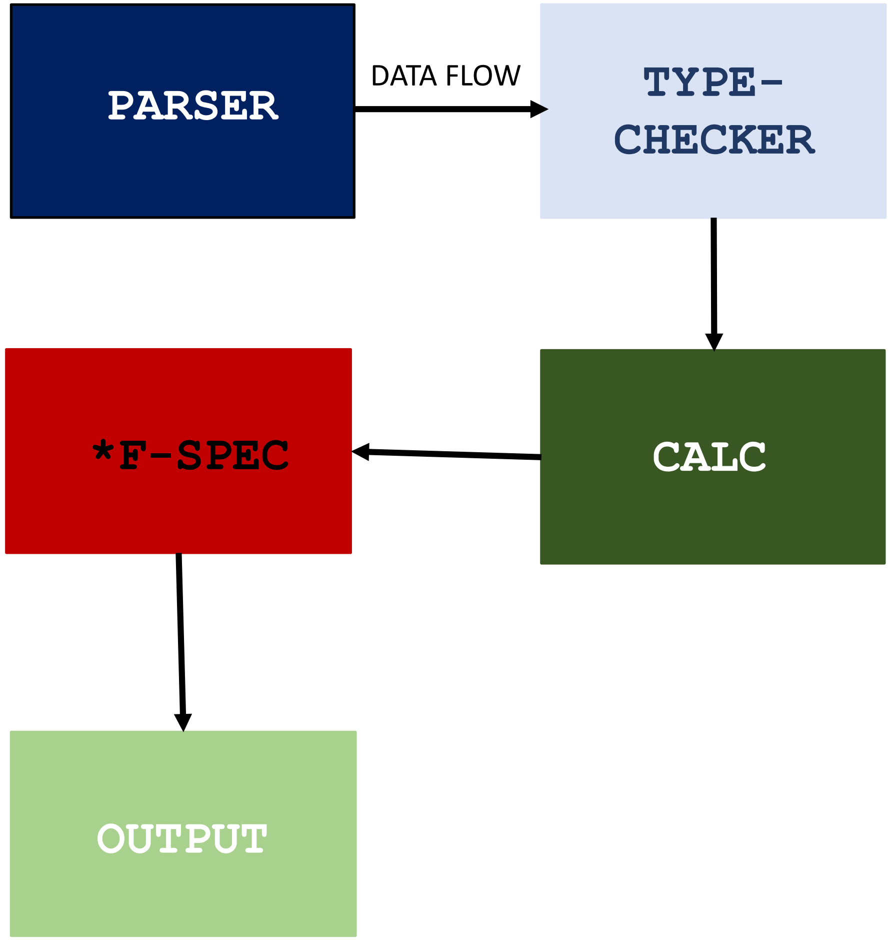


Hello, world!

You've successfully generated your Ampersand application.

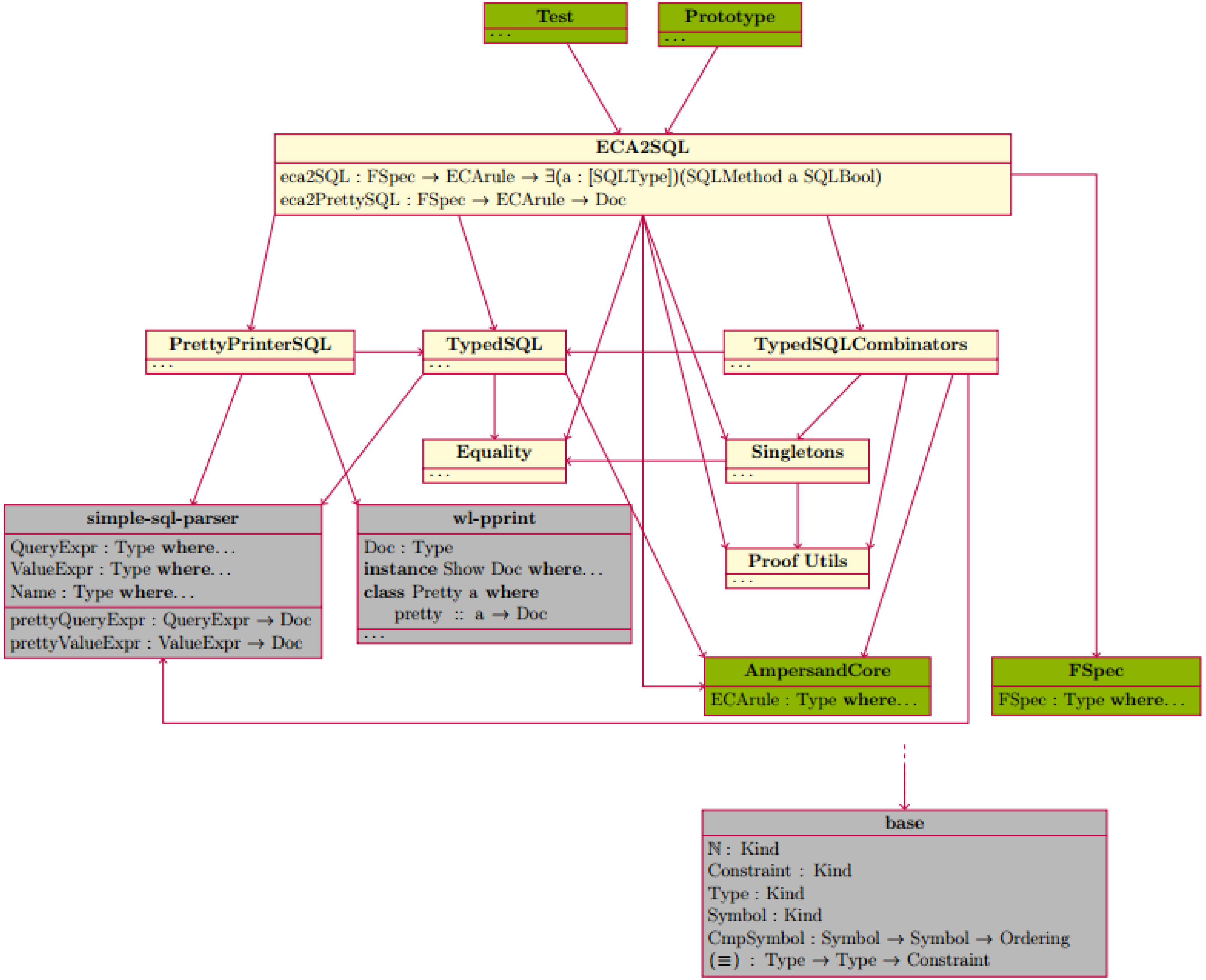
The Ampersand System



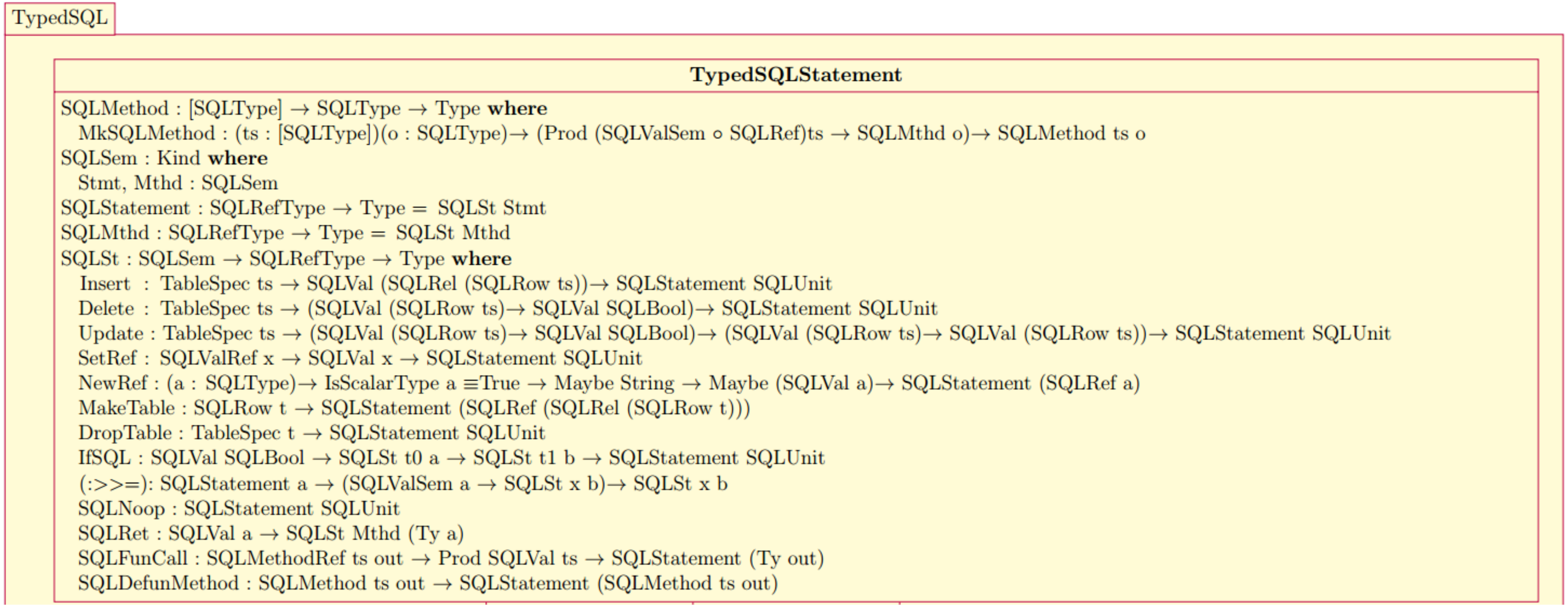


EFA (ECA For Ampersand)

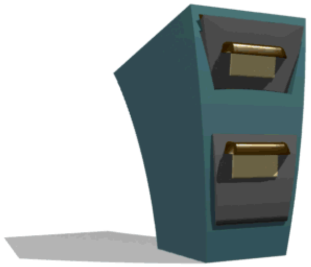
- ❖ Automatically correct system violations
- ❖ Provable correctness
- ❖ Testing for AMBRR algorithm (when AMBRR becomes complete)

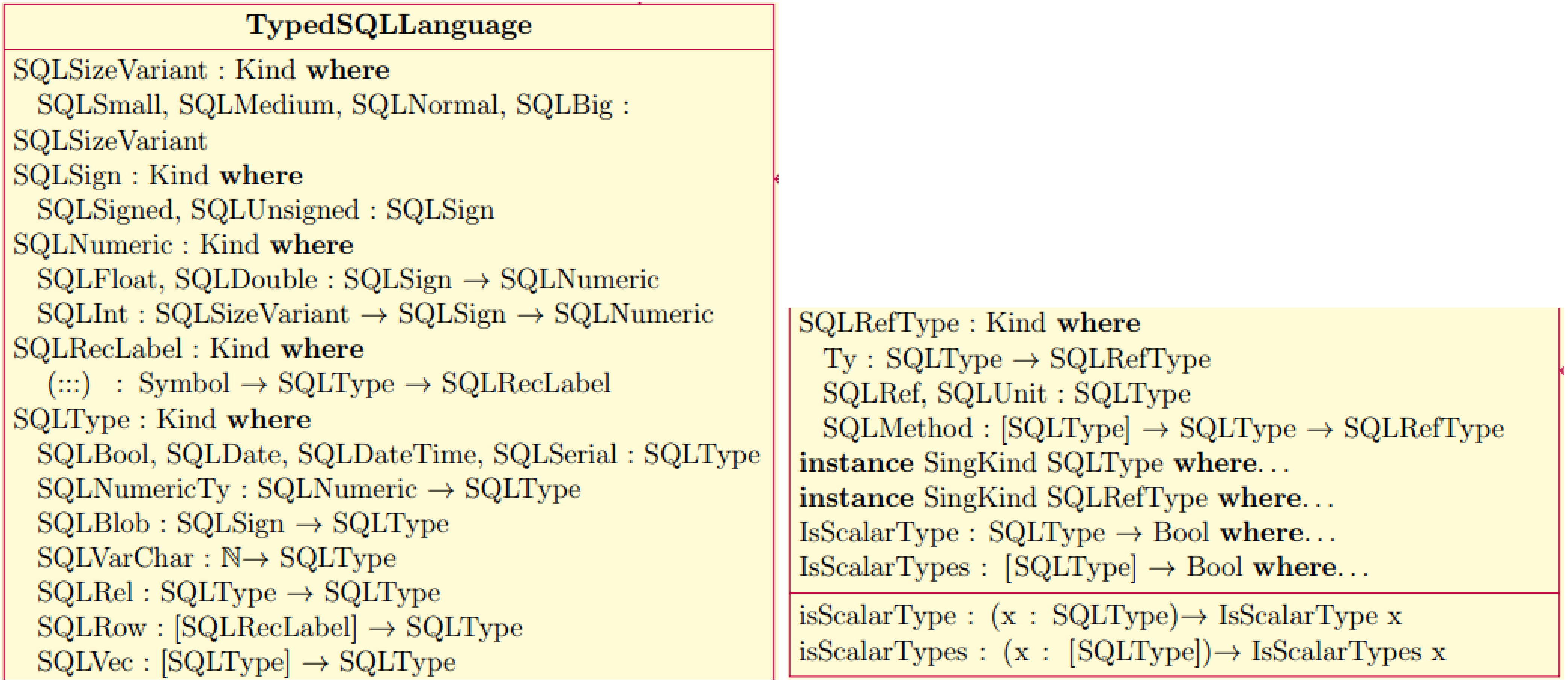

Test
. . .
Prototype
. . .
ECA2SQL
eca2SQL : FSpec → ECArule → ∃(a : [SQLType])(SQLMethod a SQLBool)
eca2PrettySQL : FSpec → ECArule → Doc
PrettyPrinterSQL
. . .
TypedSQL
. . .
TypedSQLCombinators
. . .
Equality
. . .
Singletons
. . .
Proof Utils
. . .
simple-sql-parser
QueryExpr : Type where. . .
ValueExpr : Type where. . .
Name : Type where. . .
prettyQueryExpr : QueryExpr → Doc
prettyValueExpr : ValueExpr → Doc
wl-pprint
Doc : Type
instance Show Doc where. . .
class Pretty a where
pretty :: a → Doc
. . .
AmpersandCore
ECArule : Type where. . .
FSpec
FSpec : Type where. . .
base
ℕ : Kind
Constraint : Kind
Type : Kind
Symbol : Kind
CmpSymbol : Symbol → Symbol → Ordering
(≡) : Type → Type → Constraint

Code Generation



```
TypedSQL

TypedSQLStatement

SQLMethod : [SQLType] → SQLType → Type where
  MkSQLMethod : (ts : [SQLType])(o : SQLType)→ (Prod (SQLValSem ∘ SQLRef)ts → SQLMthd o)→ SQLMethod ts o
SQLSem : Kind where
  Stmt, Mthd : SQLSem
SQLStatement : SQLRefType → Type = SQLSt Stmt
SQLMthd : SQLRefType → Type = SQLSt Mthd
SQLSt : SQLSem → SQLRefType → Type where
  Insert  : TableSpec ts → SQLVal (SQLRel (SQLRow ts))→ SQLStatement SQLUnit
  Delete  : TableSpec ts → (SQLVal (SQLRow ts)→ SQLVal SQLBool)→ SQLStatement SQLUnit
  Update : TableSpec ts → (SQLVal (SQLRow ts)→ SQLVal SQLBool)→ (SQLVal (SQLRow ts)→ SQLVal (SQLRow ts))→ SQLStatement SQLUnit
  SetRef : SQLValRef x → SQLVal x → SQLStatement SQLUnit
  NewRef : (a : SQLType)→ IsScalarType a ≡True → Maybe String → Maybe (SQLVal a)→ SQLStatement (SQLRef a)
  MakeTable : SQLRow t → SQLStatement (SQLRef (SQLRel (SQLRow t)))
  DropTable : TableSpec t → SQLStatement SQLUnit
  IfSQL : SQLVal SQLBool → SQLSt t0 a → SQLSt t1 b → SQLStatement SQLUnit
  (:>>=): SQLStatement a → (SQLValSem a → SQLSt x b)→ SQLSt x b
  SQLNoop : SQLStatement SQLUnit
  SQLRet : SQLVal a → SQLSt Mthd (Ty a)
  SQLFunCall : SQLMethodRef ts out → Prod SQLVal ts → SQLStatement (Ty out)
  SQLDefunMethod : SQLMethod ts out → SQLStatement (SQLMethod ts out)
```

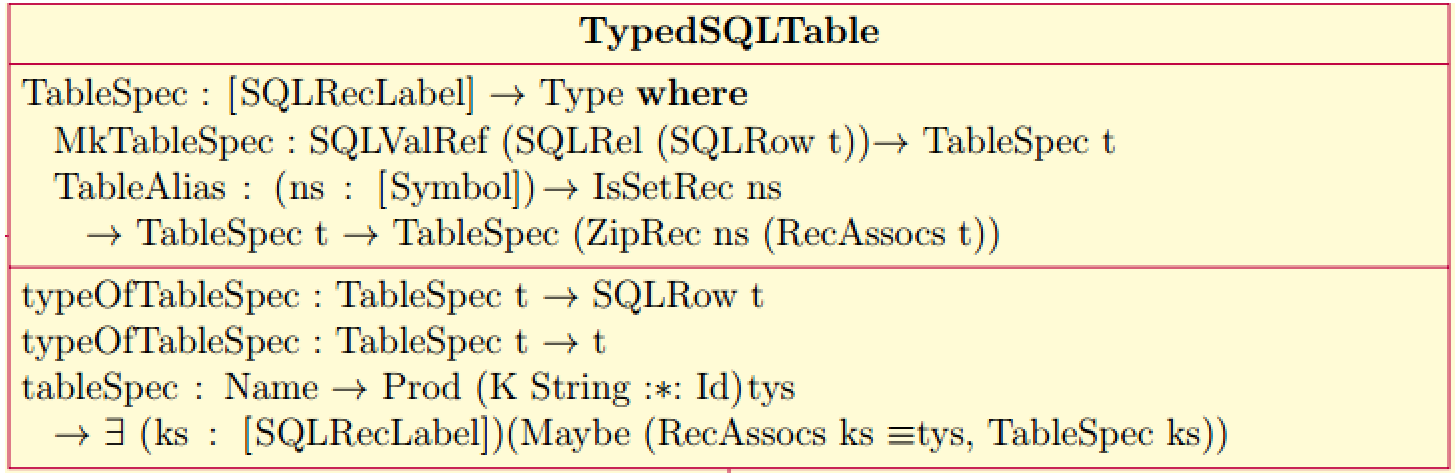




TypedSQLLanguage

```
SQLSizeVariant : Kind where
  SQLSmall, SQLMedium, SQLNormal, SQLBig :
SQLSizeVariant
SQLSign : Kind where
  SQLSigned, SQLUnsigned : SQLSign
SQLNumeric : Kind where
  SQLFloat, SQLDouble : SQLSign → SQLNumeric
  SQLInt : SQLSizeVariant → SQLSign → SQLNumeric
SQLRecLabel : Kind where
   (:::)  : Symbol → SQLType → SQLRecLabel
SQLType : Kind where
  SQLBool, SQLDate, SQLDateTime, SQLSerial : SQLType
  SQLNumericTy : SQLNumeric → SQLType
  SQLBlob : SQLSign → SQLType
  SQLVarChar : ℕ→ SQLType
  SQLRel : SQLType → SQLType
  SQLRow : [SQLRecLabel] → SQLType
  SQLVec : [SQLType] → SQLType
SQLRefType : Kind where
  Ty : SQLType → SQLRefType
  SQLRef, SQLUnit : SQLType
  SQLMethod : [SQLType] → SQLType → SQLRefType
instance SingKind SQLType where...
instance SingKind SQLRefType where...
IsScalarType : SQLType → Bool where...
IsScalarTypes : [SQLType] → Bool where...
```

```
isScalarType : (x : SQLType)→ IsScalarType x
isScalarTypes : (x : [SQLType])→ IsScalarTypes x
```



TypedSQLTable

```
TableSpec : [SQLRecLabel] → Type where
  MkTableSpec : SQLValRef (SQLRel (SQLRow t))→ TableSpec t
  TableAlias : (ns : [Symbol])→ IsSetRec ns
    → TableSpec t → TableSpec (ZipRec ns (RecAssocs t))
```

```
typeOfTableSpec : TableSpec t → SQLRow t
typeOfTableSpec : TableSpec t → t
tableSpec : Name → Prod (K String :∗: Id)tys
  → ∃ (ks : [SQLRecLabel])(Maybe (RecAssocs ks ≡tys, TableSpec ks))
```

TypedSQLExpr

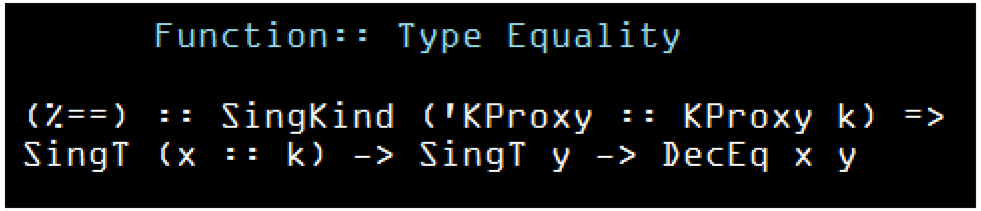
```
SQLVal : SQLType → Type where
  pattern SQLScalarVal : IsScalarType a ≡True → ValueExpr → SQLVal a
  pattern SQLQueryVal : IsScalarType a ≡False → QueryExpr → SQLVal a
SQLValSem : SQLRefType → Type where
  Unit : SQLValSem SQLUnit
  Val : (x : SQLType)→ SQLVal x → SQLValSem (Ty x)
  pattern Method : Name → SQLValSem (SQLMethod args out)
  pattern Ref : (x : SQLType)→ Name → SQLValSem (SQLRef x)
SQLVal : SQLType → Type = λx.SQLValSem (Ty x)
SQLValRef : SQLType → Type = λx.SQLValSem (SQLRef x)
```

```
typeOf : SQLVal a → a
argOfRel : SQLRel a → a
typeOfSem : f ∈ [SQLRef, Ty] → SQLValSem (f x)→ x
colsOf : SQLRow xs → xs
unsafeSQLValFromName : (x : SQLType)→ Name → SQLVal x
unsafeSQLValFromQuery : (xs : [SQLRecLabel])→ NonEmpty xs
  → IsSetRec xs → SQLVal (SQLRel (SQLRow xs))
unsafeRefFromName : (x : SQLType)→ Name → SQLValRef x
deref : SQLValRef x → SQLVal x
```

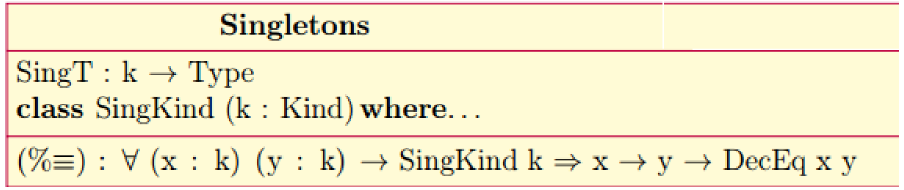
How it works.. And Why it is correct.

Source Code: Haskell

- Proposition as Types
- Type Level Modeling
- Dependent Types
- Abstraction Barriers



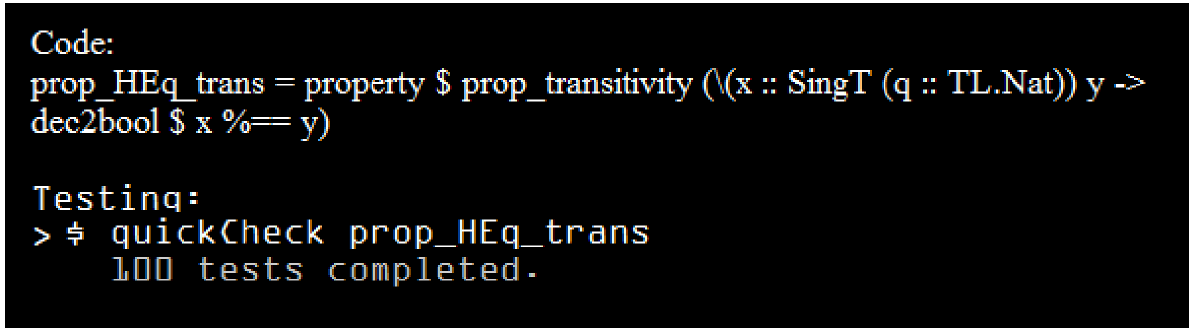
```
      Function:: Type Equality

(%==) :: SingKind ('KProxy :: KProxy k) =>
SingT (x :: k) -> SingT y -> DecEq x y
```



Singletons
SingT : k → Type **class** SingKind (k : Kind) **where**. . .
(%≡) : ∀ (x : k) (y : k) → SingKind k ⇒ x → y → DecEq x y

Testing for Properties

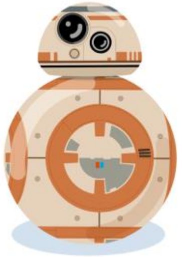
Transitive Property: if a=b, b=c, then a = c


Code:

```
prop_HEq_trans = property $ prop_transitivity (\(x :: SingT (q :: TL.Nat)) y ->
dec2bool $ x %== y)
```

Testing:

```
> $ quickCheck prop_HEq_trans
   100 tests completed.
```



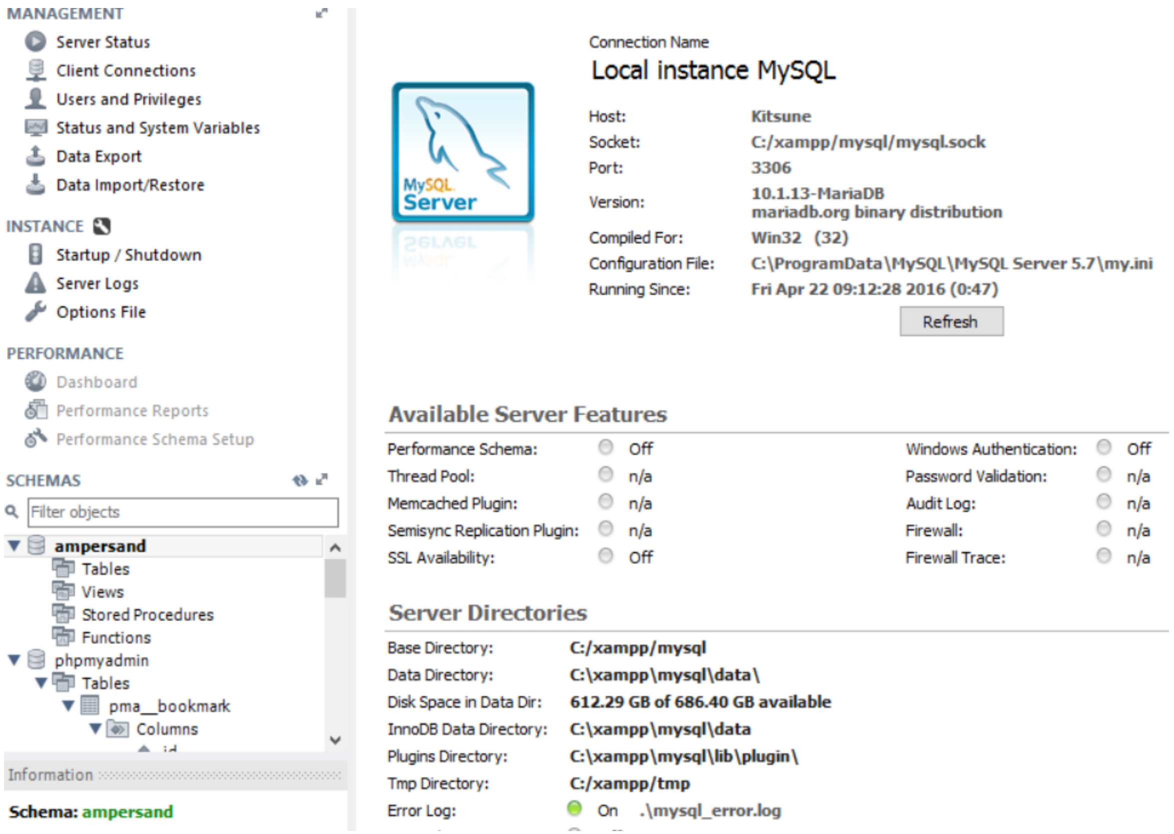
SQL Database Checking Using Workbench

Ampersand

ECA { ecaTriggr = On Ins
rel_assignmentStarted_Assignment_Assignment ,
ecaDelta = vio_Delta_Assignment_Assignment ,
ecaAction = Do Ins {<things that need to be inserted>})]
ecaNum = 29 }

EFA

ON On {eSrt = Ins, eDcl =
assignmentStarted[Assignment*Assignment]} INTO
Delta[Assignment*Assignment] DO
INSERT INTO Isn{detyp=Assignment}
SELECT FROM (Delta;Delta~ /\ I[Assignment]) -
I[Assignment] \/ (Delta~;Delta /\ I[Assignment]) -
I[Assignment]}



Schema: ampersand

Connection Name

Local instance MySQL

Host: Kitsune
Socket: C:/xampp/mysql/mysql.sock
Port: 3306
Version: 10.1.13-MariaDB
mariadb.org binary distribution
Compiled For: Win32 (32)
Configuration File: C:\ProgramData\MySQL\MySQL Server 5.7\my.ini
Running Since: Fri Apr 22 09:12:28 2016 (0:47)

Available Server Features

Performance Schema: ○ Off
Thread Pool: ○ n/a
Memcached Plugin: ○ n/a
Semisync Replication Plugin: ○ n/a
SSL Availability: ○ Off
Windows Authentication: ○ Off
Password Validation: ○ n/a
Audit Log: ○ n/a
Firewall: ○ n/a
Firewall Trace: ○ n/a

Server Directories

Base Directory: C:/xampp/mysql
Data Directory: C:\xampp\mysql\data\
Disk Space in Data Dir: 612.29 GB of 686.40 GB available
InnoDB Data Directory: C:\xampp\mysql\data
Plugins Directory: C:\xampp\mysql\lib\plugin\
Tmp Directory: C:/xampp/tmp
Error Log: ● On .\mysql_error.log

References

- Oracle Corporation. *MySQL 5.7 Reference Manual*.
- D.M. Weiss D.L. Parnas, P.C. Clements. The modular structure of complex systems. *Proceeding ICSE '84: Proceedings of the 7th international, conference on Software engineering*, 1984.
  Richard A. Eisenberg and Stephanie Weirich. Dependently typed programming with singletons. *SIGPLAN Not.*, 47(12):117–130, September 2012.
- Stef Joosten. Deriving Functional Specifications from Business Requirements with Ampersand. *CiteSeer*, 2007.
- Daan Leijen. wl-pprint package. https://hackage.haskell.org/package/wl-pprint-1.2. Accessed: 2016-02-28.
- Sam Lindley and Conor McBride. Hasochism: The pleasure and pain of dependently typed haskell programming. *SIGPLAN Not.*, 48(12):81–92, September 2013.
- A. Andreas Norell U., Danielsson N. A. The Agda Wiki. http://wiki.portal.chalmers.se/agda/pmwiki.php. Accessed: 2016-02-28.
- Oracle. Mysql reference manual. https://dev.mysql.com/doc/refman/5.7/en/. Accessed: 2016-02-28.
- The University Court of the University of Glasgow. *The Glorious Glasgow Haskell Compilation System User's Guide*. Accessed: 2016-02-29.
- D.L. Parnas. On the criteria to ne used in decomposing systems into modules. *Communications of the ACM*, 1972.
- Tim Sheard and Simon Peyton Jones. Template meta-programming for haskell. *SIGPLAN Not.*, 37(12):60–75, December 2002.
- Wouter Swierstra. Data types la carte. *Cambridge University Press*, 2008. 24
- Philip Wadler. Propositions as types. *Communications of the AC*, 58(12):7584, 2015.
- Jake Wheat. simple-sql-parser. https://hackage.haskell.org/package/simple-sql-parser-0.4.1. Accessed: 2016-02-28.
- Brent A. Yorgey, Stephanie Weirich, Julien Cretin, Simon Peyton Jones, Dimitrios Vytiniotis, and Jos´e Pedro Magalh˜aes. Giving haskell a promotion. In *Proceedings of the 8th ACM SIGPLAN Workshop on Types in Language Design and Implementation*, TLDI '12, pages 53–66, New York, NY, USA, 2012. ACM