Project: DRA FRONT END FILTER PROJECT

Title: Specification of Query Transformations in HOL (I)

Ref: DS/FMU/FEF/028 *Issue: Revision* : *2.1* *Date:* 5 June 2016

Status: Approved *Type:* Specification

Keywords:

Author:

Name	*Location*	*Signature*	*Date*
R.D. Arthan	WIN01		

Authorisation for Issue:

Name	*Function*	*Signature*	*Date*
R.B. Jones	HAT Manager		

Abstract: Preliminaries to a HOL specification of the SSQL Query Transformations for the DRA front end filter project RSRE 1C/6130.

Distribution: HAT FEF File
Simon Wiseman

0 DOCUMENT CONTROL

0.1 Contents List

0.2 Document Cross References

[1] L.Paulson. *ML for the Working Programmer.* Cambridge University Press, 1991.

[2] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* MIT Press, 1990.

[3] DS/FMU/FEF/003. *Formal Security Policy.* G.M. Prout, ICL Secure Systems, WIN01.

[4] DS/FMU/FEF/004. *Specification of SSQL Semantics I.* G.M. Prout, ICL Secure Systems, WIN01.

[5] DS/FMU/FEF/014. *Specification of SSQL Semantics II.* G.M. Prout, ICL Secure Systems, WIN01.

[6] DS/FMU/FEF/018. *Proposal for Phase 2.* G.M. Prout, ICL Secure Systems, WIN01.

[7] DS/FMU/FEF/019. *Specification of Query Transformations in SML (I).* G.M. Prout, ICL Secure Systems, WIN01.

[8] DS/FMU/FEF/020. *Specification of Query Transformations in SML (II).* G.M. Prout, ICL Secure Systems, WIN01.

[9] DS/FMU/FEF/029. *Specification of Query Transformations in HOL (II).* R.D. Arthan, ICL Secure Systems, WIN01.

[10] *SSQL Transformations.* Simon Wiseman, DRA, 14th January 1993.

0.3 Changes History

Issue 1.1 (19 May 1993) First draft.

Issue 1.6 (24 May 1993) SSQL and TSQL specs separated from transformations.

Issue *Revision* : *2.1* **(5 June 2016)** Final approved version.

Issue 2.2 Removed dependency on ICL logo font

0.4 Changes Forecast

None.

1 GENERAL

1.1 Scope

This document gives specifications in HOL ([2], [1]) of the SSQL and TSQL abstract syntax and preliminary specifications from [10] to support a formal specification in HOL of the SSQL query transformations of [10]. It is quite closely based on the Standard ML treatment of the same material in [7].

The present, draft, issue of this document constitutes part of deliverable D11 of work package 3, as given in the Proposal for Phase 2, [6].

1.2 Introduction

We proposed in [6] to formalise the SSQL query transformations of [10] in HOL. Here we provide HOL specifications of the SSQL and TSQL datatypes together with specifications of the transformation notation, generic functions, primitive routines, transformation types and symbol table from [10]. HOL specifications of the transformations may be found in [9].

In order to avoid overloading of identifiers, subscripts have been used to distinguish between constructors of SSQL types and TSQL types.

The structure of this document is largely based on that of the Standard ML treatment of [6]. The main differences between the two treatments are as follows:

1. The abstract syntax of SSQL is already defined in HOL in [5] and so need not be redefined here.
2. Some of the general purpose functions of [6] which represent various specification idioms used and defined in [10] are subsumed by library functions which are already defined for ProofPower-HOL and so do not need to be defined here.
3. Both [10] and its Standard ML description in [8] make much use of definition by cases over the alternatives for a data type. To facilitate the transcription to HOL, constants and type definitions which allow a systematic approach to defining such data types in a way which allows definition by cases using a style similar to that of [10].

1.3 ProofPower Preamble

The following commands initialise the ProofPower system to accept the specifications:

```
SML
open_theory"fef025";
(force_delete_theory "fef028" handle _ => ());
new_theory"fef028";
set_pc"hol";
```

2 UNIVERSAL TYPES

We will use the HOL types *BOOL*, *ONE*, *LIST* and *STRING* where appropriate.

The primitive types of [10] may mostly be borrowed from the SSQL semantics of [4]. However, *Enum* and *Fixed* are not given in [4] and *Integer* is called *Int* there. For the time being, we take *Enum*, *Integer* and *Fixed* to be the same as *Int*.

SML

```
declare_type_abbrev("Enum", [], ⌜:Int⌝);
declare_type_abbrev("Integer", [], ⌜:Int⌝);
declare_type_abbrev("Fixed", [], ⌜:Int⌝);
```

The values of type *Op* are taken from [4].

3 SSQL ABSTRACT SYNTAX

See [5]. Unfortunately, that document currently describes an out-of-date version of SSQL. As an interim solution to this version mismatch we introduce the *case* constructor here:

HOL Constant

```
        case : Value LIST → Value LIST → Value → Value
├──────────────────────────────────────────────────────
        true
```

4 TSQL ABSTRACT SYNTAX

For the time being this is being taken as the same as the SSQL abstract syntax.

5 SSQL TRANSFORMATIONS

5.1 Generic Functions and Primitive Functions

Many of the functions of [7] are handled directly by HOL library functions. For example, strings are just lists of characters in HOL and the list operators may be used to manipulate them. Many others are already defined in [5].

Functions which raise exceptions are modelled by functions which are loosely defined in the "error" cases. Thus, the functions which call them should not rely on them to report errors in any way. We will use the function *length* for the length of a list, (`#` in [10]) and *fold* to fold a list into a single value (& in [10]). *fold* with a lower case *f* is different from the HOL library *Fold* (which has an extra parameter indicating what to do with an empty list and turns out to have a slightly more general type).

SML

HOL Constant

$$\mathbf{fold} : ('a \times 'a \to 'a) \to 'a\ LIST \to 'a$$

$$\begin{array}{lll} \forall f\ h\ t \bullet & fold\ f\ (Cons\ h\ t) & \\ = & if & t = [] \\ & then & h \\ & else & f(h, fold\ f\ t) \end{array}$$

The library function *Combine* does for the function *combine2* of [7] and may easily be used for *combine3*, *combine4* etc. Some of the idioms used in [8] are most neatly expressed in HOL using a variant of *Combine* which truncates the longer of the two lists if necessary, we call this *splice* (to suggest rope, which can be cut):

HOL Constant

$$\mathbf{splice} : 'a\ LIST \to 'b\ LIST \to ('a \times 'b)\ LIST$$

$$\begin{array}{ll} \forall h1\ h2\ list1\ list2 \bullet & \\ & splice\ []\ []\ = [] \\ \wedge & splice\ []\ (Cons\ h2\ list2) = [] \\ \wedge & splice\ (Cons\ h1\ list1)\ []\ = [] \\ \wedge & splice\ (Cons\ h1\ list1)\ (Cons\ h2\ list2) = Cons\ (h1, h2)\ (splice\ list1\ list2) \end{array}$$

Now we define the function *at2* (and *at3* and *at4*) (@ in [10]) which turns a function taking a sequence of pairs (triples,..) into one taking a pair (triple,..) of sequences.

SML

HOL Constant

$$\mathbf{at2} : (('a \times 'b)\ LIST \to 'c) \to ('a\ LIST \times 'b\ LIST) \to 'c$$

$$\forall f\ as\ bs \bullet at2\ f\ (as, bs) = f(Combine\ as\ bs)$$

SML

HOL Constant

$$\mathbf{at3} : (('a \times 'b \times 'c)\ LIST \to 'd) \to ('a\ LIST \times 'b\ LIST \times 'c\ LIST) \to 'd$$

$$\forall f\ as\ bs\ cs \bullet at3\ f\ (as, bs, cs) = f(Combine\ as\ (Combine\ bs\ cs))$$

HOL Constant

$$\begin{array}{l} \mathbf{at4} : (('a \times 'b \times 'c \times 'd)\ LIST \to 'e) \\ \qquad \to ('a\ LIST \times 'b\ LIST \times 'c\ LIST \times 'd\ LIST) \to 'e \end{array}$$

$$\begin{array}{ll} \forall f\ as\ bs\ cs\ ds \bullet & \\ & at4\ f\ (as, bs, cs, ds) \\ = & f(Combine\ as\ (Combine\ bs\ (Combine\ cs\ ds))) \end{array}$$

The infix function *dom* of [7] is just the same as the function *dominates* of [3].

SML

```
declare_infix (150, "dom");
```

HOL Constant

$$\$\mathbf{dom} : Class \to Class \to BOOL$$

$$\$dom = \$dominates$$

For the function *invert* we use a somewhat different formulation from [7] to make the definition more evidently primitive recursive. N.b. the case when the argument is an empty list is left loose (it is an error case in the Standard ML version).

HOL Constant

$$\mathbf{invert} : 'a\ LIST\ LIST \to 'a\ LIST\ LIST$$

$$\begin{array}{lll} \forall xs\ more\bullet & & \\ & invert\ (Cons\ xs\ more) & \\ = & if \quad more = [] & \\ & then \quad Map\ (\lambda x\bullet\ Cons\ x\ [])\ xs & \\ & else \quad Map\ (\lambda(x,\ ys)\bullet\ Cons\ x\ ys)\ (Combine\ xs\ (invert\ more)) & \end{array}$$

HOL Constant

$$\mathbf{split3} : ('a \times 'b \times 'c)\ LIST \to 'a\ LIST \times 'b\ LIST \times 'c\ LIST$$

$$\forall abcs\bullet\ split3\ abcs = let\ (as,\ bcs) = Split\ abcs\ in\ (as,\ Split\ bcs)$$

HOL Constant

$$\begin{array}{l} \mathbf{split4} : ('a \times 'b \times 'c \times 'd)\ LIST \to \\ \qquad 'a\ LIST \times 'b\ LIST \times 'c\ LIST \times 'd\ LIST \end{array}$$

$$\forall abcds\bullet \quad split4\ abcds = let\ (as,\ bcds) = Split\ abcds\ in\ (as,\ split3\ bcds)$$

HOL Constant

$$\begin{array}{l} \mathbf{split5} : ('a \times 'b \times 'c \times 'd \times 'e)\ LIST \to \\ \qquad 'a\ LIST \times 'b\ LIST \times 'c\ LIST \times 'd\ LIST \times 'e\ LIST \end{array}$$

$$\forall abcdes\bullet \quad split5\ abcdes = let\ (as,\ bcdes) = Split\ abcdes\ in\ (as,\ split4\ bcdes)$$

5.2 Transformation Notation

The primitive types are discussed in section 2. We will use the HOL one-point type *ONE* in place of *Null*.

5.3 Data Types and Definition by Cases

The HOL disjoint union type constructor + will stand for the disjoint union symbol | used in the type definitions of [10]. Both [8] and [10] make frequent use of definition by cases over the alternatives for a disjoint union type. We mimic the form of these definitions using the following auxiliary definitions.

SML

```
declare_type_abbrev("OPT", ["'a"], ⌜:'a + ONE⌝);
```

HOL Constant

$$\textbf{CASE} : 'a \to ('a \to 'b\ OPT)\ LIST \to 'b$$

$$\forall a\ f\ fs \bullet\ CASE\ a\ (Cons\ f\ fs) = \ if\ IsL(f\ a)\ then\ OutL\ (f\ a)\ else\ CASE\ a\ fs$$

HOL Constant

$$\textbf{OTHERS} : 'b \to ('a \to 'b\ OPT)$$

$$\forall b\ a \bullet\ OTHERS\ b\ a = InL\ b$$

SML

```
declare_type_abbrev("MK_DEST", ["'rep", "'abs"], ⌜:('rep → 'abs) × ('abs → 'rep)⌝);
declare_type_abbrev("WHEN", ["'rep", "'abs", "'a"], ⌜:('rep → 'a) → ('abs → 'a OPT)⌝);
declare_type_abbrev("WHEN_CONST", ["'abs", "'a"], ⌜:'a → 'abs → 'a OPT⌝);
```

HOL Constant

$$\textbf{Lift} : ('rep, 'abs)\ MK\_DEST \to ('rep, 'abs, 'a)\ WHEN$$

$$\begin{array}{l}\forall\ mk\ dest\ f\ a \bullet \\ \quad Lift\ (mk, dest)\ f\ a \\ = \quad if \quad mk(dest\ a) = a \\ \quad then \quad InL\ (f\ (dest\ a)) \\ \quad else \quad InR\ One\end{array}$$

HOL Constant

$$\textbf{LiftConstant} : 'abs \to 'a \to 'abs \to 'a\ OPT$$

$$\begin{array}{l}\forall\ con\ a\ abs \bullet \\ \quad LiftConstant\ con\ a\ abs \\ = \quad if \quad abs = con \\ \quad then \quad InL\ a \\ \quad else \quad InR\ One\end{array}$$

The use of these will be illustrated and explained in subsequent sections.

The following material is used to represent the treatment of exceptions in [9]. It is a variant of the treatment of data types used in later sections, which gives more mnemonic names for the constructors etc.

SML

```
declare_type_abbrev("RESULT", ["'a"], ⌜:'a + Errors⌝);
```

HOL Constant

```
        (Ok, OkValue) :
                ('a, 'a RESULT) MK_DEST;
        (Exception, ExceptionValue) :
                (Errors, 'a RESULT) MK_DEST
├──────────────────────────────────────────────
        Ok              = InL
∧       Exception       = InR
∧       OkValue                 = OutL
∧       ExceptionValue          = OutR
```

SML

```
declare_binder"WHEN_ok";
declare_binder"WHEN_exception";
```

HOL Constant

```
        $WHEN_ok :
                ('a, 'a RESULT, 'b) WHEN;
        $WHEN_exception :
                (Errors, 'a RESULT, 'b) WHEN
├──────────────────────────────────────────────
        $WHEN_ok = Lift (Ok, OkValue)
∧       $WHEN_exception = Lift (Exception, ExceptionValue)
```

We also need some functions to assist in the propagation of errors:

HOL Constant

```
        $ListOk : ('a) RESULT LIST → ('a) LIST RESULT
├──────────────────────────────────────────────
∀resl•
        ListOk resl =
        if      Fold ($∧) (Map IsL resl) T
        then    InL(Map OutL resl)
        else    let folder res errs = if IsL res then errs else OutR res @ errs
                in InR (Fold folder resl [])
```

There is some overlap between the following and the $*$ functions of [5]; however, the following seem to be more convenient for transcribing the material from [9].

HOL Constant

$$\$\mathbf{Try} : ('a \to 'b\ RESULT) \to ('a\ RESULT \to 'b\ RESULT)$$

$$\forall f \bullet \quad Try\ f = \lambda a \bullet\ if\ isVal\ a\ then\ f(destVal\ a)\ else\ giveError(destError\ a)$$

5.4 Types

The definition of a (non-recursive) data type now follows a standard pattern. First of all we define the name of the data type to be an abbreviation for the underlying disjoint union type:

SML

```
declare_type_abbrev("TableSpecification", [],
    ⌜:(STRING LIST × STRING) + (ℕ × STRING LIST × STRING)⌝);
```

Next, we define constructor and destructor functions for the data type:

HOL Constant

$$(\mathbf{mk\_absolute}, \mathbf{dest\_absolute}) : ((STRING\ LIST \times STRING), TableSpecification)\ MK\_DEST;$$
$$(\mathbf{mk\_default}, \mathbf{dest\_default}) : ((\mathbb{N} \times STRING\ LIST \times STRING), TableSpecification)\ MK\_DEST$$

$$mk\_absolute = InL$$
$$\wedge\quad mk\_default = InR$$
$$\wedge\quad dest\_absolute = OutL$$
$$\wedge\quad dest\_default = OutR$$

Finally, we use *Lift* to define functions which act, in effect, as pattern-matching λ-abstractions for the various alternatives of the data type. We will use binder syntax for these functions. Thus for example, we can write $WHEN\_absolute\ (sl,\ s) \bullet f(sl,\ s)$, which is equivalent to applying the function *WHEN_absolute* to the λ-abstraction $(\lambda(sl,\ s) \bullet f(sl,\ s))$, and, in effect, denotes that function on *TableSpecification*s which returns the value $f(sl, s)$ for arguments of the form $mk\_absolute(sl,\ s)$ and returns an error indicate $(InR\ One)$ otherwise.

SML

```
declare_binder"WHEN_absolute";
declare_binder"WHEN_default";
```

HOL Constant

$$\$\mathbf{WHEN\_absolute} : (STRING\ LIST \times STRING,\ TableSpecification, 'b)\ WHEN;$$
$$\$\mathbf{WHEN\_default} : (\mathbb{N} \times STRING\ LIST \times STRING,\ TableSpecification, 'b)\ WHEN$$

$$\$WHEN\_absolute = Lift\ (mk\_absolute,\ dest\_absolute)$$
$$\wedge\quad \$WHEN\_default = Lift\ (mk\_default,\ dest\_default)$$

The declaration of the above functions with binder syntax allows operations on the data type to be defined in a style which is similar to the *CASE* constructs of [10]. For example, the operation of extracting the directory part of a *TableSpecification* called *ts* say may be written as:

```
        CASE ts [
                (WHEN_absolute (dir, tab) • dir)
        ;       (WHEN_default (up, dir, tab) • dir)
        ]
```

The working of the *CASE*-construct may be seen in action using the rewriting capability of the proof tool, for example, by executing:

ProofPower Input

```
rewrite_conv
(map get_spec [⌜CASE⌝, ⌜Lift⌝, ⌜$WHEN_absolute⌝, ⌜$mk_absolute⌝])
⌜CASE (mk_default(99, ["dir1"; "dir2"], "tab")) [
        (WHEN_absolute (dir, tab) • dir);
        (WHEN_default (up, dir, tab) • dir)
]⌝;
```

This returns the following theorem:

ProofPower Output

```
val it = ⊢ CASE
      (mk_default (99, ["dir1"; "dir2"], "tab"))
      [WHEN_absolute (dir, tab)• dir; WHEN_default (up, dir, tab)• dir]
    = ["dir1"; "dir2"] : THM
```

We treat the other data types for the transformations in a similar fashion. In cases such as the following, involving constructors with no argument, we just define the constant value and use *LiftConstant* to define the pattern-matching λ-abstraction.

SML

```
declare_type_abbrev("SwordType", [], ⌜:
        ONE                             (* nullType *)
+       ONE                             (* monoleanType *)
+       ONE                             (* booleanType *)
+       (Int × Int)                     (* stringType *)
+       (Int × Int)                     (* fixedType *)
+       (Int × TableSpecification)      (* enumType *)
+       STRING                          (* timeType *)
+       STRING                          (* intervalType *)
+       ONE                             (* classType *)
+       ONE                             (* codeType *)
+       ONE                             (* anyType *)
⌝);
```

HOL Constant

c_nullType : *SwordType*;
c_monoleanType : *SwordType*;
c_booleanType : *SwordType*;
(**mk_stringType** , **dest_stringType**) : (*Int* × *Int*, *SwordType*) *MK_DEST*;
(**mk_fixedType** , **dest_fixedType**) : (*Int* × *Int*, *SwordType*) *MK_DEST*;
(**mk_enumType** , **dest_enumType**)
: (*Int* × *TableSpecification*, *SwordType*) *MK_DEST*;
(**mk_timeType** , **dest_timeType**) : (*STRING*, *SwordType*) *MK_DEST*;
(**mk_intervalType** , **dest_intervalType**) : (*STRING*, *SwordType*) *MK_DEST*;
c_classType : *SwordType*;
c_codeType : *SwordType*;
c_anyType : *SwordType*

---

c_nullType = *InL One*
∧ *c_monoleanType* = (*InR o InL*) *One*
∧ *c_booleanType* = (*InR o InR o InL*) *One*
∧ *mk_stringType* = *InR o InR o InR o InL*
∧ *mk_fixedType* = *InR o InR o InR o InR o InL*
∧ *mk_enumType* = *InR o InR o InR o InR o InR o InL*
∧ *mk_timeType* = *InR o InR o InR o InR o InR o InR o InL*
∧ *mk_intervalType* = *InR o InR o InR o InR o InR o InR o InR o InL*
∧ *c_classType* = (*InR o InR o InR o InR o InR o InR o InR o InR o InL*) *One*
∧ *c_codeType* = (*InR o InR o InR o InR o InR o InR o InR o InR o InR o InL*) *One*
∧ *c_anyType* = (*InR o InR o InR o InR o InR o InR o InR o InR o InR o InR*) *One*
∧
dest_stringType = *OutL o OutR o OutR o OutR*
∧ *dest_fixedType* = *OutL o OutR o OutR o OutR o OutR*
∧ *dest_enumType* = *OutL o OutR o OutR o OutR o OutR o OutR*
∧ *dest_timeType* = *OutL o OutR o OutR o OutR o OutR o OutR o OutR*
∧ *dest_intervalType* = *OutL o OutR o OutR o OutR o OutR o OutR o OutR o OutR*

SML

declare_binder"*WHEN_stringType*";
declare_binder"*WHEN_fixedType*";
declare_binder"*WHEN_enumType*";
declare_binder"*WHEN_timeType*";
declare_binder"*WHEN_intervalType*";

HOL Constant

```
        $WHEN_nullType : (SwordType, 'a) WHEN_CONST;
        $WHEN_monoleanType : (SwordType, 'a) WHEN_CONST;
        $WHEN_booleanType : (SwordType, 'a) WHEN_CONST;
        $WHEN_stringType : (Int × Int, SwordType, 'a) WHEN;
        $WHEN_fixedType : (Int × Int, SwordType, 'a) WHEN;
        $WHEN_enumType : (Int × TableSpecification, SwordType, 'a) WHEN;
        $WHEN_timeType : (STRING, SwordType, 'a) WHEN;
        $WHEN_intervalType : (STRING, SwordType, 'a) WHEN;
        $WHEN_classType : (SwordType, 'a) WHEN_CONST;
        $WHEN_codeType : (SwordType, 'a) WHEN_CONST;
        $WHEN_anyType : (SwordType, 'a) WHEN_CONST
─────────────────────────────────────────────────────
        $WHEN_nullType = LiftConstant c_nullType
∧       $WHEN_monoleanType = LiftConstant c_monoleanType
∧       $WHEN_booleanType = LiftConstant c_booleanType
∧       $WHEN_stringType = Lift (mk_stringType, dest_stringType)
∧       $WHEN_fixedType = Lift (mk_fixedType, dest_fixedType)
∧       $WHEN_enumType = Lift (mk_enumType, dest_enumType)
∧       $WHEN_timeType = Lift (mk_timeType, dest_timeType)
∧       $WHEN_intervalType = Lift (mk_intervalType, dest_intervalType)
∧       $WHEN_classType = LiftConstant c_classType
∧       $WHEN_codeType = LiftConstant c_codeType
∧       $WHEN_anyType = LiftConstant c_anyType
```

SML

```
declare_type_abbrev("SsqlName", [], ⌜:
        ONE                 (* anon_s *)
+       STRING              (* name_s *)
⌝);
```

HOL Constant

```
        c_anon_s  : SsqlName;
        (mk_name_s , dest_name_s) : (STRING, SsqlName) MK_DEST
─────────────────────────────────────────────────────
        c_anon_s            = InL One
∧       mk_name_s           = InR
∧       dest_name_s         = OutR
```

SML

```
declare_binder"WHEN_name_s";
```

HOL Constant

```
        $WHEN_anon_s : (SsqlName, 'a) WHEN_CONST;
        $WHEN_name_s : (STRING, SsqlName, 'a) WHEN
├──────────────────────────────────────────────────
        $WHEN_anon_s = LiftConstant c_anon_s
∧       $WHEN_name_s = Lift (mk_name_s, dest_name_s)
```

SML

```
declare_type_abbrev("TsqlName", [], ⌜:
        ONE                     (* none_t *)
+       ONE                     (* anon_t *)
+       STRING                  (* name_t *)
⌝);
```

HOL Constant

```
        c_none_t  : TsqlName;
        c_anon_t  : TsqlName;
        (mk_name_t , dest_name_t) : (STRING, TsqlName) MK_DEST
├──────────────────────────────────────────────────
        c_none_t                = InL One
∧       c_anon_t                = (InR o InL) One
∧       mk_name_t               = InR o InR
∧       dest_name_t             = OutR o OutR
```

SML

```
declare_binder"WHEN_name_t";
```

HOL Constant

```
        $WHEN_none_t : (TsqlName, 'a) WHEN_CONST;
        $WHEN_anon_t : (TsqlName, 'a) WHEN_CONST;
        $WHEN_name_t : (STRING, TsqlName, 'a) WHEN
├──────────────────────────────────────────────────
        $WHEN_none_t = LiftConstant c_none_t
∧       $WHEN_anon_t = LiftConstant c_anon_t
∧       $WHEN_name_t = Lift (mk_name_t, dest_name_t)
```

SML

```
declare_type_abbrev("ColType", [], ⌜:SwordType × SwordType⌝);
```

SML

```
declare_type_abbrev("BoundInfo", [], ⌜:
        Class                   (* upb *)
+       Class                   (* constant *)
⌝);
```

HOL Constant

(**mk_upb** , **dest_upb**) : (*Class*, *BoundInfo*) *MK_DEST*;
(**mk_constant** , **dest_constant**) : (*Class*, *BoundInfo*) *MK_DEST*

---

mk_upb = *InL*
∧ *mk_constant* = *InR*
∧ *dest_upb* = *OutL*
∧ *dest_constant* = *OutR*

SML

```
declare_binder"WHEN_upb";
declare_binder"WHEN_constant";
```

HOL Constant

$**WHEN_upb** : (*Class*, *BoundInfo*, *'a*) *WHEN*;
$**WHEN_constant** : (*Class*, *BoundInfo*, *'a*) *WHEN*

$*WHEN_upb* = *Lift* (*mk_upb*, *dest_upb*)
∧ $*WHEN_constant* = *Lift* (*mk_constant*, *dest_constant*)

HOL Labelled Product

SsqlCol

sc_name	:	*SsqlName*;
sc_type_field	:	*ColType*;
sc_col_exist	:	*Class*;
sc_col_class	:	*BoundInfo*

SML

```
declare_type_abbrev("TsqlClassName", [], ⌜:
        ONE                 (* anon_tc *)
+       STRING              (* name_tc *)
+       Class               (* constant_tc *)
⌝);
```

HOL Constant

$$\mathbf{c\_anon_{tc}} : \mathit{TsqlClassName};$$
$$(\mathbf{mk\_name_{tc}}\ , \mathbf{dest\_name_{tc}}) : (\mathit{STRING},\ \mathit{TsqlClassName})\ \mathit{MK\_DEST};$$
$$(\mathbf{mk\_constant_{tc}}\ , \mathbf{dest\_constant_{tc}}) : (\mathit{Class},\ \mathit{TsqlClassName})\ \mathit{MK\_DEST}$$

$$\begin{aligned} & \mathit{c\_anon}_{tc} && = \mathit{InL}\ \mathit{One} \\ \wedge\ & \mathit{mk\_name}_{tc} && = \mathit{InR}\ o\ \mathit{InL} \\ \wedge\ & \mathit{mk\_constant}_{tc} && = \mathit{InR}\ o\ \mathit{InR} \\ \wedge\ & \mathit{dest\_name}_{tc} && = \mathit{OutL}\ o\ \mathit{OutR} \\ \wedge\ & \mathit{dest\_constant}_{tc} && = \mathit{OutR}\ o\ \mathit{OutR} \end{aligned}$$

SML

$$\mathit{declare\_binder}\texttt{"}\mathit{WHEN\_name}_{tc}\texttt{"};$$
$$\mathit{declare\_binder}\texttt{"}\mathit{WHEN\_constant}_{tc}\texttt{"};$$

HOL Constant

$$\$\mathbf{WHEN\_anon_{tc}} : (\mathit{TsqlClassName}, 'a)\ \mathit{WHEN\_CONST};$$
$$\$\mathbf{WHEN\_name_{tc}} : (\mathit{STRING},\ \mathit{TsqlClassName}, 'a)\ \mathit{WHEN};$$
$$\$\mathbf{WHEN\_constant_{tc}} : (\mathit{Class},\ \mathit{TsqlClassName}, 'a)\ \mathit{WHEN}$$

$$\begin{aligned} & \$\mathit{WHEN\_anon}_{tc} = \mathit{LiftConstant}\ \mathit{c\_anon}_{tc} \\ \wedge\ & \$\mathit{WHEN\_name}_{tc} = \mathit{Lift}\ (\mathit{mk\_name}_{tc},\ \mathit{dest\_name}_{tc}) \\ \wedge\ & \$\mathit{WHEN\_constant}_{tc} = \mathit{Lift}\ (\mathit{mk\_constant}_{tc},\ \mathit{dest\_constant}_{tc}) \end{aligned}$$

HOL Labelled Product

TsqlCol

tc_sterling_name	:	*TsqlName*;
tc_dinary_name	:	*TsqlName*;
tc_class_name	:	*TsqlClassName*

SML

HOL Labelled Product

TableInfo

ti_table_exist_class	:	*Class*;
ti_table_class	:	*Class*;
ti_row_class	:	*BoundInfo*

HOL Labelled Product

ConstraintInfo

ci_null_allowed	:	*BOOL LIST*;
ci_lwb	:	*Class LIST*;
ci_unique	:	$\mathbb{N}$ *LIST LIST*;
ci_uniform	:	$\mathbb{N}$ *LIST LIST*;
ci_index	:	$\mathbb{N}$ *LIST LIST*

SML

```
declare_type_abbrev("ColumnSpecification", [], ⌜:
        STRING                                  (* anonymous_column *)
+       (TableSpecification × STRING)           (* specific *)
⌝);
```

HOL Constant

```
        (mk_anonymous_column , dest_anonymous_column)
                : (STRING, ColumnSpecification) MK_DEST;
        (mk_specific , dest_specific)
                : (TableSpecification × STRING, ColumnSpecification) MK_DEST
```

```
        mk_anonymous_column     = InL
∧       mk_specific             = InR
∧       dest_anonymous_column   = OutL
∧       dest_specific           = OutR
```

SML

```
declare_binder"WHEN_anonymous_column";
declare_binder"WHEN_specific";
```

HOL Constant

```
        $WHEN_anonymous_column : (STRING, ColumnSpecification, 'a) WHEN;
        $WHEN_specific
                : (TableSpecification × STRING, ColumnSpecification, 'a) WHEN
```

```
        $WHEN_anonymous_column = Lift (mk_anonymous_column, dest_anonymous_column)
∧       $WHEN_specific = Lift (mk_specific, dest_specific)
```

SML

```
declare_type_abbrev("TsqlRepr", [], ⌜:
        STRING                                  (* local_identifier *)
+       (STRING × STRING)                       (* column *)
+       Class                                   (* constant_class *)
+       ONE                                     (* constant_null *)
⌝);
```

HOL Constant

```
        (mk_local_identifier , dest_local_identifier)
                : (STRING, TsqlRepr) MK_DEST;
        (mk_column , dest_column)
                : (STRING × STRING, TsqlRepr) MK_DEST;
        (mk_constant_class , dest_constant_class)
                : (Class, TsqlRepr) MK_DEST;
        c_constant_null : TsqlRepr
──────────────────────────────────────────────────
        mk_local_identifier    = InL
∧       mk_column              = InR o InL
∧       mk_constant_class      = InR o InR  o InL
∧       c_constant_null        = (InR o InR o InR) One
∧       dest_local_identifier  = OutL
∧       dest_column            = OutL o OutR
∧       dest_constant_class    = OutL o OutR o OutR
```

SML

```
declare_binder"WHEN_local_identifier";
declare_binder"WHEN_column";
declare_binder"WHEN_constant_class";
```

HOL Constant

```
        $WHEN_local_identifier : (STRING, TsqlRepr, 'a) WHEN;
        $WHEN_column : (STRING × STRING, TsqlRepr, 'a) WHEN;
        $WHEN_constant_class : (Class, TsqlRepr, 'a) WHEN;
        $WHEN_constant_null : (TsqlRepr, 'a) WHEN_CONST
──────────────────────────────────────────────────
        $WHEN_local_identifier = Lift (mk_local_identifier, dest_local_identifier)
∧       $WHEN_column = Lift (mk_column, dest_column)
∧       $WHEN_constant_class = Lift (mk_constant_class, dest_constant_class)
∧       $WHEN_constant_null = LiftConstant c_constant_null
```

SML

```
declare_type_abbrev( "ExpType", [], ⌜:SwordType × Worth⌝);
```

SML

```
declare_type_abbrev("ExpClass", [], ⌜:
        (Value × Class)                 (* variable *)
+       Class                           (* constant_ec *)
⌝);
```

HOL Constant

```
        (mk_variable , dest_variable)
                : (Value × Class, ExpClass) MK_DEST;
        (mk_constant_ec , dest_constant_ec)
                : (Class, ExpClass) MK_DEST
├──────────────────────────────────────────────
        mk_variable          = InL
∧       mk_constant_ec       = InR
∧       dest_variable        = OutL
∧       dest_constant_ec     = OutR
```

SML

```
declare_binder"WHEN_variable";
declare_binder"WHEN_constant_ec";
```

HOL Constant

```
        $WHEN_variable : (Value × Class, ExpClass, 'a) WHEN;
        $WHEN_constant_ec : (Class, ExpClass, 'a) WHEN
├──────────────────────────────────────────────
        $WHEN_variable = Lift (mk_variable, dest_variable)
∧       $WHEN_constant_ec = Lift (mk_constant_ec, dest_constant_ec)
```

SML

```
declare_type_abbrev("InternalExpClass", [], ⌜:
        (Value LIST × ExpClass LIST)          (* ands *)
+       (Value LIST × ExpClass LIST)          (* ors *)
+       ExpClass                              (* simple *)
⌝);
```

HOL Constant

```
        (mk_ands , dest_ands)
                : (Value LIST × ExpClass LIST, InternalExpClass) MK_DEST;
        (mk_ors , dest_ors)
                : (Value LIST × ExpClass LIST, InternalExpClass) MK_DEST;
        (mk_simple , dest_simple)
                : (ExpClass, InternalExpClass) MK_DEST
├──────────────────────────────────────────────
        mk_ands      = InL
∧       mk_ors       = InR o InL
∧       mk_simple    = InR o InR
∧       dest_ands    = OutL
∧       dest_ors     = OutL o OutR
∧       dest_simple  = OutR o OutR
```

SML

```
declare_binder"WHEN_ands";
declare_binder"WHEN_ors";
declare_binder"WHEN_simple";
```

HOL Constant

```
        $WHEN_ands : (Value LIST × ExpClass LIST, InternalExpClass, 'a) WHEN;
        $WHEN_ors : (Value LIST × ExpClass LIST, InternalExpClass, 'a) WHEN;
        $WHEN_simple : (ExpClass, InternalExpClass, 'a) WHEN
──────────────────────────────────────────────────────────
        $WHEN_ands = Lift (mk_ands, dest_ands)
∧       $WHEN_ors = Lift (mk_ors, dest_ors)
∧       $WHEN_simple = Lift (mk_simple, dest_simple)
```

SML

```
declare_type_abbrev("TableName", [], ⌜:
        ONE                     (* anon_tn *)
+       TableSpecification      (* name_tn *)
⌝);
```

HOL Constant

```
        c_anon_tn  : TableName;
        (mk_name_tn , dest_name_tn) : (TableSpecification, TableName) MK_DEST
──────────────────────────────────────────────────────────
        c_anon_tn               = InL One
∧       mk_name_tn              = InR
∧       dest_name_tn            = OutR
```

SML

```
declare_binder"WHEN_name_tn";
```

HOL Constant

```
        $WHEN_anon_tn : (TableName, 'a) WHEN_CONST;
        $WHEN_name_tn : (TableSpecification, TableName, 'a) WHEN
──────────────────────────────────────────────────────────
        $WHEN_anon_tn = LiftConstant c_anon_tn
∧       $WHEN_name_tn = Lift (mk_name_tn, dest_name_tn)
```

5.5 The Symbol Table

SML

HOL Labelled Product

TableDetail

td_tableName	:	*TableName*;
td_corrName	:	*SsqlName*;
td_genCorr	:	*STRING*;
td_info	:	*TableInfo*;
td_columns	:	*SsqlCol LIST*;
td_rowClass	:	*TsqlClassName*;
td_implementation	:	*TsqlCol LIST*;
td_constraints	:	*ConstraintInfo*

HOL Labelled Product

IdentDetail

id_identName	:	*STRING*;
id_info	:	*ExpType*;
$\mathbf{id\_lub_{id}}$	:	*Class*;
id_vName	:	*STRING*;
id_cName	:	*TsqlName*

HOL Labelled Product

Scope

s_tables	:	*TableDetail LIST*;
s_identifiers	:	*IdentDetail LIST*

In the following, for the time being *Value* (from [5] is used instead of *ConstantValue*.

HOL Labelled Product

ParamInfo

pi_name	:	*STRING*;
$\mathbf{pi\_val_p}$	:	*Value*;
pi_clasf	:	*Class*

The state accessed by the symbol table routines has the following two components. It is managed as a stack using the functions *new_scope* of [9] (corresponding to the "block-structured" use of *enter_scope* and *leave_scope* in [10]). The *enter_ldots* functions of [9] operate by updating the top of the stack.

HOL Labelled Product

ST_STACK

symbolTable	:	*Scope LIST*;
parameterTable	:	*ParamInfo LIST*

6 CLOSING DOWN

7 THE THEORY fef028

7.1 Parents

fef025

7.2 Children

fef029

7.3 Constants

case	$Value\ LIST \to Value\ LIST \to Value \to Value$
fold	$('a \times 'a \to 'a) \to 'a\ LIST \to 'a$
splice	$'a\ LIST \to 'b\ LIST \to ('a \times 'b)\ LIST$
at2	$(('a \times 'b)\ LIST \to 'c) \to 'a\ LIST \times 'b\ LIST \to 'c$
at3	$(('a \times 'b \times 'c)\ LIST \to 'd) \to 'a\ LIST \times 'b\ LIST \times 'c\ LIST \to 'd$
at4	$(('a \times 'b \times 'c \times 'd)\ LIST \to 'e) \to 'a\ LIST \times 'b\ LIST \times 'c\ LIST \times 'd\ LIST \to 'e$
$dom	$Class \to Class \to Bool$
invert	$'a\ LIST\ LIST \to 'a\ LIST\ LIST$
split3	$('a \times 'b \times 'c)\ LIST \to 'a\ LIST \times 'b\ LIST \times 'c\ LIST$
split4	$('a \times 'b \times 'c \times 'd)\ LIST \to 'a\ LIST \times 'b\ LIST \times 'c\ LIST \times 'd\ LIST$
split5	$('a \times 'b \times 'c \times 'd \times 'e)\ LIST \to 'a\ LIST \times 'b\ LIST \times 'c\ LIST \times 'd\ LIST \times 'e\ LIST$
CASE	$'a \to ('a \to 'b\ OPT)\ LIST \to 'b$
OTHERS	$('a, 'b)\ WHEN\_CONST$
Lift	$('rep, 'abs)\ MK\_DEST \to ('rep, 'abs, 'a)\ WHEN$
LiftConstant	$'abs \to ('abs, 'a)\ WHEN\_CONST$
ExceptionValue	$'a\ RESULT \to Errors$
Exception	$Errors \to 'a\ RESULT$
OkValue	$'a\ RESULT \to 'a$
Ok	$'a \to 'a\ RESULT$
$WHEN_exception	$(Errors, 'a\ RESULT, 'b)\ WHEN$
$WHEN_ok	$('a, 'a\ RESULT, 'b)\ WHEN$
ListOk	$'a\ RESULT\ LIST \to 'a\ LIST\ RESULT$
Try	$('a \to 'b\ RESULT) \to 'a\ RESULT \to 'b\ RESULT$
dest_default	$TableSpecification \to Worth \times Col \times Op$
mk_default	$Worth \times Col \times Op \to TableSpecification$
dest_absolute	$TableSpecification \to Col \times Op$

mk_absolute $Col \times Op \rightarrow TableSpecification$
$WHEN_default
$(Worth \times Col \times Op,\ TableSpecification,\ 'b)\ WHEN$
$WHEN_absolute
$(Col \times Op,\ TableSpecification,\ 'b)\ WHEN$
c_anyType $SwordType$
c_codeType $SwordType$
c_classType $SwordType$
dest_intervalType
$SwordType \rightarrow Op$
mk_intervalType
$Op \rightarrow SwordType$
dest_timeType
$SwordType \rightarrow Op$
mk_timeType $Op \rightarrow SwordType$
dest_enumType
$SwordType \rightarrow Fixed \times TableSpecification$
mk_enumType $Fixed \times TableSpecification \rightarrow SwordType$
dest_fixedType
$SwordType \rightarrow Fixed \times Fixed$
mk_fixedType $Fixed \times Fixed \rightarrow SwordType$
dest_stringType
$SwordType \rightarrow Fixed \times Fixed$
mk_stringType
$Fixed \times Fixed \rightarrow SwordType$
c_booleanType
$SwordType$
c_monoleanType
$SwordType$
c_nullType $SwordType$
WHEN_anyType $(SwordType,\ 'a)\ WHEN\_CONST$
WHEN_codeType
$(SwordType,\ 'a)\ WHEN\_CONST$
WHEN_classType
$(SwordType,\ 'a)\ WHEN\_CONST$
$WHEN_intervalType
$(Op,\ SwordType,\ 'a)\ WHEN$
$WHEN_timeType
$(Op,\ SwordType,\ 'a)\ WHEN$
$WHEN_enumType
$(Fixed \times TableSpecification,\ SwordType,\ 'a)\ WHEN$
$WHEN_fixedType
$(Fixed \times Fixed,\ SwordType,\ 'a)\ WHEN$
$WHEN_stringType
$(Fixed \times Fixed,\ SwordType,\ 'a)\ WHEN$
WHEN_booleanType
$(SwordType,\ 'a)\ WHEN\_CONST$
WHEN_monoleanType

(*SwordType*, *'a*) *WHEN_CONST*

WHEN_nullType
(*SwordType*, *'a*) *WHEN_CONST*

dest_name$_s$ *SsqlName* → *Op*

mk_name$_s$ *Op* → *SsqlName*

c_anon$_s$ *SsqlName*

$WHEN_name$_s$ (*Op*, *SsqlName*, *'a*) *WHEN*

WHEN_anon$_s$ (*SsqlName*, *'a*) *WHEN_CONST*

dest_name$_t$ *TsqlName* → *Op*

mk_name$_t$ *Op* → *TsqlName*

c_anon$_t$ *TsqlName*

c_none$_t$ *TsqlName*

$WHEN_name$_t$ (*Op*, *TsqlName*, *'a*) *WHEN*

WHEN_anon$_t$ (*TsqlName*, *'a*) *WHEN_CONST*

WHEN_none$_t$ (*TsqlName*, *'a*) *WHEN_CONST*

dest_constant
BoundInfo → *Class*

mk_constant *Class* → *BoundInfo*

dest_upb *BoundInfo* → *Class*

mk_upb *Class* → *BoundInfo*

$WHEN_constant
(*Class*, *BoundInfo*, *'a*) *WHEN*

$WHEN_upb (*Class*, *BoundInfo*, *'a*) *WHEN*

sc_col_class *SsqlCol* → *BoundInfo*

sc_col_exist *SsqlCol* → *Class*

sc_type_field
SsqlCol → *ColType*

sc_name *SsqlCol* → *SsqlName*

MkSsqlCol *SsqlName* → *ColType* → *Class* → *BoundInfo* → *SsqlCol*

dest_constant$_{tc}$
TsqlClassName → *Class*

mk_constant$_{tc}$
Class → *TsqlClassName*

dest_name$_{tc}$
TsqlClassName → *Op*

mk_name$_{tc}$ *Op* → *TsqlClassName*

c_anon$_{tc}$ *TsqlClassName*

$WHEN_constant$_{tc}$
(*Class*, *TsqlClassName*, *'a*) *WHEN*

$WHEN_name$_{tc}$
(*Op*, *TsqlClassName*, *'a*) *WHEN*

WHEN_anon$_{tc}$
(*TsqlClassName*, *'a*) *WHEN_CONST*

tc_class_name
TsqlCol → *TsqlClassName*

tc_dinary_name
TsqlCol → *TsqlName*

tc_sterling_name

$TsqlCol \to TsqlName$

MkTsqlCol $TsqlName \to TsqlName \to TsqlClassName \to TsqlCol$

ti_row_class $TableInfo \to BoundInfo$

ti_table_class
$TableInfo \to Class$

ti_table_exist_class
$TableInfo \to Class$

MkTableInfo $Class \to Class \to BoundInfo \to TableInfo$

ci_index $ConstraintInfo \to Errors\ LIST$

ci_uniform $ConstraintInfo \to Errors\ LIST$

ci_unique $ConstraintInfo \to Errors\ LIST$

ci_lwb $ConstraintInfo \to Class\ LIST$

ci_null_allowed
$ConstraintInfo \to Bool\ LIST$

MkConstraintInfo
$Bool\ LIST$
$\to Class\ LIST$
$\to Errors\ LIST$
$\to Errors\ LIST$
$\to Errors\ LIST$
$\to ConstraintInfo$

dest_specific
$ColumnSpecification \to TableSpecification \times Op$

mk_specific $TableSpecification \times Op \to ColumnSpecification$

dest_anonymous_column
$ColumnSpecification \to Op$

mk_anonymous_column
$Op \to ColumnSpecification$

$WHEN_specific
$(TableSpecification \times Op,\ ColumnSpecification,\ 'a)\ WHEN$

$WHEN_anonymous_column
$(Op,\ ColumnSpecification,\ 'a)\ WHEN$

c_constant_null
$TsqlRepr$

dest_constant_class
$TsqlRepr \to Class$

mk_constant_class
$Class \to TsqlRepr$

dest_column $TsqlRepr \to Op \times Op$

mk_column $Op \times Op \to TsqlRepr$

dest_local_identifier
$TsqlRepr \to Op$

mk_local_identifier
$Op \to TsqlRepr$

WHEN_constant_null
$(TsqlRepr,\ 'a)\ WHEN\_CONST$

$WHEN_constant_class
$(Class,\ TsqlRepr,\ 'a)\ WHEN$

$WHEN_column $(Op \times Op,\ TsqlRepr,\ 'a)\ WHEN$
$WHEN_local_identifier
$(Op,\ TsqlRepr,\ 'a)\ WHEN$
dest_constant$_{ec}$
$ExpClass \rightarrow Class$
mk_constant$_{ec}$
$Class \rightarrow ExpClass$
dest_variable
$ExpClass \rightarrow Value \times Class$
mk_variable $Value \times Class \rightarrow ExpClass$
$WHEN_constant$_{ec}$
$(Class,\ ExpClass,\ 'a)\ WHEN$
$WHEN_variable
$(Value \times Class,\ ExpClass,\ 'a)\ WHEN$
dest_simple $InternalExpClass \rightarrow ExpClass$
mk_simple $ExpClass \rightarrow InternalExpClass$
dest_ors $InternalExpClass \rightarrow Value\ LIST \times ExpClass\ LIST$
mk_ors $Value\ LIST \times ExpClass\ LIST \rightarrow InternalExpClass$
dest_ands $InternalExpClass \rightarrow Value\ LIST \times ExpClass\ LIST$
mk_ands $Value\ LIST \times ExpClass\ LIST \rightarrow InternalExpClass$
$WHEN_simple $(ExpClass,\ InternalExpClass,\ 'a)\ WHEN$
$WHEN_ors $(Value\ LIST \times ExpClass\ LIST,\ InternalExpClass,\ 'a)\ WHEN$
$WHEN_ands $(Value\ LIST \times ExpClass\ LIST,\ InternalExpClass,\ 'a)\ WHEN$
dest_name$_{tn}$
$TableName \rightarrow TableSpecification$
mk_name$_{tn}$ $TableSpecification \rightarrow TableName$
c_anon$_{tn}$ $TableName$
$WHEN_name$_{tn}$
$(TableSpecification,\ TableName,\ 'a)\ WHEN$
WHEN_anon$_{tn}$
$(TableName,\ 'a)\ WHEN\_CONST$
td_constraints
$TableDetail \rightarrow ConstraintInfo$
td_implementation
$TableDetail \rightarrow TsqlCol\ LIST$
td_rowClass $TableDetail \rightarrow TsqlClassName$
td_columns $TableDetail \rightarrow SsqlCol\ LIST$
td_info $TableDetail \rightarrow TableInfo$
td_genCorr $TableDetail \rightarrow Op$
td_corrName $TableDetail \rightarrow SsqlName$
td_tableName $TableDetail \rightarrow TableName$
MkTableDetail
$TableName$
$\rightarrow SsqlName$
$\rightarrow Op$
$\rightarrow TableInfo$
$\rightarrow SsqlCol\ LIST$
$\rightarrow TsqlClassName$

	$\rightarrow$ *TsqlCol LIST*
	$\rightarrow$ *ConstraintInfo*
	$\rightarrow$ *TableDetail*
id_cName	*IdentDetail* $\rightarrow$ *TsqlName*
id_vName	*IdentDetail* $\rightarrow$ *Op*
id_lub$_{id}$	*IdentDetail* $\rightarrow$ *Class*
id_info	*IdentDetail* $\rightarrow$ *ExpType*
id_identName	*IdentDetail* $\rightarrow$ *Op*
MkIdentDetail	
	Op $\rightarrow$ *ExpType* $\rightarrow$ *Class* $\rightarrow$ *Op* $\rightarrow$ *TsqlName* $\rightarrow$ *IdentDetail*
s_identifiers	
	Scope $\rightarrow$ *IdentDetail LIST*
s_tables	*Scope* $\rightarrow$ *TableDetail LIST*
MkScope	*TableDetail LIST* $\rightarrow$ *IdentDetail LIST* $\rightarrow$ *Scope*
pi_clasf	*ParamInfo* $\rightarrow$ *Class*
pi_val$_p$	*ParamInfo* $\rightarrow$ *Value*
pi_name	*ParamInfo* $\rightarrow$ *Op*
MkParamInfo	*Op* $\rightarrow$ *Value* $\rightarrow$ *Class* $\rightarrow$ *ParamInfo*
parameterTable	
	ST_STACK $\rightarrow$ *ParamInfo LIST*
symbolTable	*ST_STACK* $\rightarrow$ *Scope LIST*
MkST_STACK	*Scope LIST* $\rightarrow$ *ParamInfo LIST* $\rightarrow$ *ST_STACK*

7.4 Types

SsqlCol
TsqlCol
TableInfo
ConstraintInfo
TableDetail
IdentDetail
Scope
ParamInfo
ST_STACK

7.5 Type Abbreviations

Enum	*Fixed*
Integer	*Fixed*
Fixed	*Fixed*
$'a$ **OPT**	$'a$ *OPT*
$('rep, 'abs)$ **MK_DEST**	
	$('rep, 'abs)$ *MK_DEST*
$('rep, 'abs, 'a)$ **WHEN**	
	$('rep, 'abs, 'a)$ *WHEN*
$('abs, 'a)$ **WHEN_CONST**	
	$('abs, 'a)$ *WHEN_CONST*
$'a$ **RESULT**	$'a$ *RESULT*

TableSpecification
TableSpecification
SwordType *SwordType*
SsqlName *SsqlName*
TsqlName *TsqlName*
ColType *ColType*
BoundInfo *BoundInfo*
TsqlClassName
TsqlClassName
ColumnSpecification
ColumnSpecification
TsqlRepr *TsqlRepr*
ExpType *ExpType*
ExpClass *ExpClass*
InternalExpClass
InternalExpClass
TableName *TableName*

7.6 Fixity

Binder: **WHEN_absolute** **WHEN_local_identifier**
WHEN_andsWHEN_name$_s$
WHEN_anonymous_column **WHEN_name$_t$**
WHEN_column **WHEN_name$_{tc}$**
WHEN_constant **WHEN_name$_{tn}$**
WHEN_constant_class **WHEN_ok**
WHEN_constant$_{ec}$ **WHEN_ors**
WHEN_constant$_{tc}$ **WHEN_simple**
WHEN_default **WHEN_specific**
WHEN_enumType **WHEN_stringType**
WHEN_exception **WHEN_timeType**
WHEN_fixedType **WHEN_upb**
WHEN_intervalType **WHEN_variable**
Right Infix 150:
dom

7.7 Definitions

case $\vdash$ *true*
fold $\vdash \forall$ *f h t*
• *fold f* (*Cons h t*)
= (*if t* = [] *then h else f* (*h*, *fold f t*))
splice $\vdash \forall$ *h1 h2 list1 list2*
• *splice* [] [] = []
$\wedge$ *splice* [] (*Cons h2 list2*) = []
$\wedge$ *splice* (*Cons h1 list1*) [] = []
$\wedge$ *splice* (*Cons h1 list1*) (*Cons h2 list2*)
= *Cons* (*h1*, *h2*) (*splice list1 list2*)

at2 $\vdash \forall\ f\ as\ bs\bullet\ at2\ f\ (as,\ bs) = f\ (Combine\ as\ bs)$

at3 $\vdash \forall\ f\ as\ bs\ cs$
$\bullet\ at3\ f\ (as,\ bs,\ cs) = f\ (Combine\ as\ (Combine\ bs\ cs))$

at4 $\vdash \forall\ f\ as\ bs\ cs\ ds$
$\bullet\ at4\ f\ (as,\ bs,\ cs,\ ds)$
$= f\ (Combine\ as\ (Combine\ bs\ (Combine\ cs\ ds)))$

dom $\vdash \$dom = \$dominates$

invert $\vdash \forall\ xs\ more$
$\bullet\ invert\ (Cons\ xs\ more)$
$= (if\ more = []$
$then\ Map\ (\lambda\ x\bullet\ [x])\ xs$
$else$
Map
$(\lambda\ (x,\ ys)\bullet\ Cons\ x\ ys)$
$(Combine\ xs\ (invert\ more)))$

split3 $\vdash \forall\ abcs$
$\bullet\ split3\ abcs$
$= (let\ (as,\ bcs) = Split\ abcs\ in\ (as,\ Split\ bcs))$

split4 $\vdash \forall\ abcds$
$\bullet\ split4\ abcds$
$= (let\ (as,\ bcds) = Split\ abcds$
$in\ (as,\ split3\ bcds))$

split5 $\vdash \forall\ abcdes$
$\bullet\ split5\ abcdes$
$= (let\ (as,\ bcdes) = Split\ abcdes$
$in\ (as,\ split4\ bcdes))$

CASE $\vdash \forall\ a\ f\ fs$
$\bullet\ CASE\ a\ (Cons\ f\ fs)$
$= (if\ IsL\ (f\ a)\ then\ OutL\ (f\ a)\ else\ CASE\ a\ fs)$

OTHERS $\vdash \forall\ b\ a\bullet\ OTHERS\ b\ a = InL\ b$

Lift $\vdash \forall\ mk\ dest\ f\ a$
$\bullet\ Lift\ (mk,\ dest)\ f\ a$
$= (if\ mk\ (dest\ a) = a$
$then\ InL\ (f\ (dest\ a))$
$else\ InR\ maybe)$

LiftConstant $\vdash \forall\ con\ a\ abs$
$\bullet\ LiftConstant\ con\ a\ abs$
$= (if\ abs = con\ then\ InL\ a\ else\ InR\ maybe)$

OkValue
Ok
ExceptionValue
Exception $\vdash Ok = InL$
$\wedge\ Exception = InR$
$\wedge\ OkValue = OutL$
$\wedge\ ExceptionValue = OutR$

WHEN_ok
WHEN_exception
$\vdash \$WHEN\_ok = Lift\ (Ok,\ OkValue)$

```
                ∧ $WHEN_exception
                  = Lift (Exception, ExceptionValue)
ListOk          ⊢ ∀ resl
                  • ListOk resl
                    = (if Fold $∧ (Map IsL resl) true
                       then InL (Map OutL resl)
                       else
                         (let folder res errs
                                = (if IsL res
                                   then errs
                                   else OutR res @ errs)
                            in InR (Fold folder resl [])))
Try             ⊢ ∀ f
                  • Try f
                    = (λ a
                    • if isVal a
                      then f (destVal a)
                      else giveError (destError a))
dest_absolute
mk_absolute
dest_default
mk_default      ⊢ mk_absolute = InL
                ∧ mk_default = InR
                ∧ dest_absolute = OutL
                ∧ dest_default = OutR
WHEN_absolute
WHEN_default    ⊢ $WHEN_absolute = Lift (mk_absolute, dest_absolute)
                ∧ $WHEN_default = Lift (mk_default, dest_default)
c_nullType
c_monoleanType
c_booleanType
dest_stringType
mk_stringType
dest_fixedType
mk_fixedType
dest_enumType
mk_enumType
dest_timeType
mk_timeType
dest_intervalType
mk_intervalType
c_classType
c_codeType
c_anyType       ⊢ c_nullType = InL maybe
                ∧ c_monoleanType = (InR o InL) maybe
                ∧ c_booleanType = (InR o InR o InL) maybe
                ∧ mk_stringType = InR o InR o InR o InL
                ∧ mk_fixedType = InR o InR o InR o InR o InL
```

$$
\begin{aligned}
&\wedge\ mk\_enumType = InR\ o\ InR\ o\ InR\ o\ InR\ o\ InR\ o\ InL\\
&\wedge\ mk\_timeType\\
&\quad = InR\ o\ InR\ o\ InR\ o\ InR\ o\ InR\ o\ InR\ o\ InL\\
&\wedge\ mk\_intervalType\\
&\quad = InR\ o\ InR\ o\ InR\ o\ InR\ o\ InR\ o\ InR\ o\ InR\ o\ InL\\
&\wedge\ c\_classType\\
&\quad = (InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InL)\\
&\qquad maybe\\
&\wedge\ c\_codeType\\
&\quad = (InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InL)\\
&\qquad maybe\\
&\wedge\ c\_anyType\\
&\quad = (InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR\\
&\qquad o\ InR)\\
&\qquad maybe\\
&\wedge\ dest\_stringType = OutL\ o\ OutR\ o\ OutR\ o\ OutR\\
&\wedge\ dest\_fixedType = OutL\ o\ OutR\ o\ OutR\ o\ OutR\ o\ OutR\\
&\wedge\ dest\_enumType\\
&\quad = OutL\ o\ OutR\ o\ OutR\ o\ OutR\ o\ OutR\ o\ OutR\\
&\wedge\ dest\_timeType\\
&\quad = OutL\ o\ OutR\ o\ OutR\ o\ OutR\ o\ OutR\ o\ OutR\ o\ OutR\\
&\wedge\ dest\_intervalType\\
&\quad = OutL
\end{aligned}
$$

$o\ OutR$
$o\ OutR$
$o\ OutR$
$o\ OutR$
$o\ OutR$
$o\ OutR$
$o\ OutR$

WHEN_nullType
WHEN_monoleanType
WHEN_booleanType
WHEN_stringType
WHEN_fixedType
WHEN_enumType
WHEN_timeType
WHEN_intervalType
WHEN_classType
WHEN_codeType
WHEN_anyType $\vdash WHEN\_nullType = LiftConstant\ c\_nullType$
$\wedge\ WHEN\_monoleanType = LiftConstant\ c\_monoleanType$
$\wedge\ WHEN\_booleanType = LiftConstant\ c\_booleanType$
$\wedge\ \$WHEN\_stringType$
$= Lift\ (mk\_stringType,\ dest\_stringType)$
$\wedge\ \$WHEN\_fixedType$
$= Lift\ (mk\_fixedType,\ dest\_fixedType)$
$\wedge\ \$WHEN\_enumType$
$= Lift\ (mk\_enumType,\ dest\_enumType)$
$\wedge\ \$WHEN\_timeType$
$= Lift\ (mk\_timeType,\ dest\_timeType)$
$\wedge\ \$WHEN\_intervalType$
$= Lift\ (mk\_intervalType,\ dest\_intervalType)$
$\wedge\ WHEN\_classType = LiftConstant\ c\_classType$
$\wedge\ WHEN\_codeType = LiftConstant\ c\_codeType$
$\wedge\ WHEN\_anyType = LiftConstant\ c\_anyType$

c_anon_s
$dest\_name_s$
mk_name_s $\vdash c\_anon_s = InL\ maybe$
$\wedge\ mk\_name_s = InR$
$\wedge\ dest\_name_s = OutR$
$WHEN\_anon_s$
$WHEN\_name_s$ $\vdash WHEN\_anon_s = LiftConstant\ c\_anon_s$
$\wedge\ \$WHEN\_name_s = Lift\ (mk\_name_s,\ dest\_name_s)$
c_none_t
c_anon_t
$dest\_name_t$
mk_name_t $\vdash c\_none_t = InL\ maybe$
$\wedge\ c\_anon_t = (InR\ o\ InL)\ maybe$
$\wedge\ mk\_name_t = InR\ o\ InR$
$\wedge\ dest\_name_t = OutR\ o\ OutR$

WHEN_none$_t$
WHEN_anon$_t$
WHEN_name$_t$ $\vdash$ $WHEN\_none_t = LiftConstant\ c\_none_t$
$\wedge$ $WHEN\_anon_t = LiftConstant\ c\_anon_t$
$\wedge$ $\$WHEN\_name_t = Lift\ (mk\_name_t,\ dest\_name_t)$
dest_upb
mk_upb
dest_constant
mk_constant $\vdash$ $mk\_upb = InL$
$\wedge$ $mk\_constant = InR$
$\wedge$ $dest\_upb = OutL$
$\wedge$ $dest\_constant = OutR$
WHEN_upb
WHEN_constant
$\vdash$ $\$WHEN\_upb = Lift\ (mk\_upb,\ dest\_upb)$
$\wedge$ $\$WHEN\_constant$
$= Lift\ (mk\_constant,\ dest\_constant)$
SsqlCol $\vdash$ $\exists\ f\bullet\ TypeDefn\ (\lambda\ x\bullet\ true)\ f$
MkSsqlCol
sc_name
sc_type_field
sc_col_exist
sc_col_class $\vdash$ $\forall\ t\ x1\ x2\ x3\ x4$
$\bullet$ $sc\_name\ (MkSsqlCol\ x1\ x2\ x3\ x4) = x1$
$\wedge$ $sc\_type\_field\ (MkSsqlCol\ x1\ x2\ x3\ x4) = x2$
$\wedge$ $sc\_col\_exist\ (MkSsqlCol\ x1\ x2\ x3\ x4) = x3$
$\wedge$ $sc\_col\_class\ (MkSsqlCol\ x1\ x2\ x3\ x4) = x4$
$\wedge$ $MkSsqlCol$
$(sc\_name\ t)$
$(sc\_type\_field\ t)$
$(sc\_col\_exist\ t)$
$(sc\_col\_class\ t)$
$= t$
c_anon$_{tc}$
dest_name$_{tc}$
mk_name$_{tc}$
dest_constant$_{tc}$
mk_constant$_{tc}$
$\vdash$ $c\_anon_{tc} = InL\ maybe$
$\wedge$ $mk\_name_{tc} = InR\ o\ InL$
$\wedge$ $mk\_constant_{tc} = InR\ o\ InR$
$\wedge$ $dest\_name_{tc} = OutL\ o\ OutR$
$\wedge$ $dest\_constant_{tc} = OutR\ o\ OutR$
WHEN_anon$_{tc}$
WHEN_name$_{tc}$
WHEN_constant$_{tc}$
$\vdash$ $WHEN\_anon_{tc} = LiftConstant\ c\_anon_{tc}$
$\wedge$ $\$WHEN\_name_{tc}$

$$
\begin{array}{l}
\quad = \mathit{Lift}\ (\mathit{mk\_name}_{tc},\ \mathit{dest\_name}_{tc}) \\
\quad \wedge\ \$\mathit{WHEN\_constant}_{tc} \\
\quad = \mathit{Lift}\ (\mathit{mk\_constant}_{tc},\ \mathit{dest\_constant}_{tc}) \\
\textbf{TsqlCol} \quad \vdash \exists\ f \bullet\ \mathit{TypeDefn}\ (\lambda\ x \bullet\ \mathit{true})\ f \\
\textbf{MkTsqlCol} \\
\textbf{tc\_sterling\_name} \\
\textbf{tc\_dinary\_name} \\
\textbf{tc\_class\_name} \\
\qquad \vdash \forall\ t\ x1\ x2\ x3 \\
\qquad \bullet\ \mathit{tc\_sterling\_name}\ (\mathit{MkTsqlCol}\ x1\ x2\ x3) = x1 \\
\qquad\quad \wedge\ \mathit{tc\_dinary\_name}\ (\mathit{MkTsqlCol}\ x1\ x2\ x3) = x2 \\
\qquad\quad \wedge\ \mathit{tc\_class\_name}\ (\mathit{MkTsqlCol}\ x1\ x2\ x3) = x3 \\
\qquad\quad \wedge\ \mathit{MkTsqlCol} \\
\qquad\qquad (\mathit{tc\_sterling\_name}\ t) \\
\qquad\qquad (\mathit{tc\_dinary\_name}\ t) \\
\qquad\qquad (\mathit{tc\_class\_name}\ t) \\
\qquad\quad = t \\
\textbf{TableInfo} \quad \vdash \exists\ f \bullet\ \mathit{TypeDefn}\ (\lambda\ x \bullet\ \mathit{true})\ f \\
\textbf{MkTableInfo} \\
\textbf{ti\_table\_exist\_class} \\
\textbf{ti\_table\_class} \\
\textbf{ti\_row\_class} \quad \vdash \forall\ t\ x1\ x2\ x3 \\
\qquad \bullet\ \mathit{ti\_table\_exist\_class}\ (\mathit{MkTableInfo}\ x1\ x2\ x3) = x1 \\
\qquad\quad \wedge\ \mathit{ti\_table\_class}\ (\mathit{MkTableInfo}\ x1\ x2\ x3) = x2 \\
\qquad\quad \wedge\ \mathit{ti\_row\_class}\ (\mathit{MkTableInfo}\ x1\ x2\ x3) = x3 \\
\qquad\quad \wedge\ \mathit{MkTableInfo} \\
\qquad\qquad (\mathit{ti\_table\_exist\_class}\ t) \\
\qquad\qquad (\mathit{ti\_table\_class}\ t) \\
\qquad\qquad (\mathit{ti\_row\_class}\ t) \\
\qquad\quad = t \\
\textbf{ConstraintInfo} \\
\qquad \vdash \exists\ f \bullet\ \mathit{TypeDefn}\ (\lambda\ x \bullet\ \mathit{true})\ f \\
\textbf{MkConstraintInfo} \\
\textbf{ci\_null\_allowed} \\
\textbf{ci\_lwb} \\
\textbf{ci\_unique} \\
\textbf{ci\_uniform} \\
\textbf{ci\_index} \quad \vdash \forall\ t\ x1\ x2\ x3\ x4\ x5 \\
\qquad \bullet\ \mathit{ci\_null\_allowed}\ (\mathit{MkConstraintInfo}\ x1\ x2\ x3\ x4\ x5) \\
\qquad\qquad = x1 \\
\qquad\quad \wedge\ \mathit{ci\_lwb}\ (\mathit{MkConstraintInfo}\ x1\ x2\ x3\ x4\ x5) = x2 \\
\qquad\quad \wedge\ \mathit{ci\_unique}\ (\mathit{MkConstraintInfo}\ x1\ x2\ x3\ x4\ x5) \\
\qquad\qquad = x3 \\
\qquad\quad \wedge\ \mathit{ci\_uniform}\ (\mathit{MkConstraintInfo}\ x1\ x2\ x3\ x4\ x5) \\
\qquad\qquad = x4 \\
\qquad\quad \wedge\ \mathit{ci\_index}\ (\mathit{MkConstraintInfo}\ x1\ x2\ x3\ x4\ x5) = x5 \\
\qquad\quad \wedge\ \mathit{MkConstraintInfo} \\
\qquad\qquad (\mathit{ci\_null\_allowed}\ t)
\end{array}
$$

$$
\begin{array}{l}
\quad (ci\_lwb\ t) \\
\quad (ci\_unique\ t) \\
\quad (ci\_uniform\ t) \\
\quad (ci\_index\ t) \\
= t
\end{array}
$$

dest_anonymous_column
mk_anonymous_column
dest_specific
mk_specific $\vdash mk\_anonymous\_column = InL$
$\wedge\ mk\_specific = InR$
$\wedge\ dest\_anonymous\_column = OutL$
$\wedge\ dest\_specific = OutR$

WHEN_anonymous_column
WHEN_specific
$\vdash \$WHEN\_anonymous\_column = Lift\ (mk\_anonymous\_column,\ dest\_anonymous\_column)$
$\wedge\ \$WHEN\_specific = Lift\ (mk\_specific,\ dest\_specific)$

dest_local_identifier
mk_local_identifier
dest_column
mk_column
dest_constant_class
mk_constant_class
c_constant_null
$\vdash mk\_local\_identifier = InL$
$\wedge\ mk\_column = InR\ o\ InL$
$\wedge\ mk\_constant\_class = InR\ o\ InR\ o\ InL$
$\wedge\ c\_constant\_null = (InR\ o\ InR\ o\ InR)\ maybe$
$\wedge\ dest\_local\_identifier = OutL$
$\wedge\ dest\_column = OutL\ o\ OutR$
$\wedge\ dest\_constant\_class = OutL\ o\ OutR\ o\ OutR$

WHEN_local_identifier
WHEN_column
WHEN_constant_class
WHEN_constant_null
$\vdash \$WHEN\_local\_identifier = Lift\ (mk\_local\_identifier,\ dest\_local\_identifier)$
$\wedge\ \$WHEN\_column = Lift\ (mk\_column,\ dest\_column)$
$\wedge\ \$WHEN\_constant\_class = Lift\ (mk\_constant\_class,\ dest\_constant\_class)$
$\wedge\ WHEN\_constant\_null = LiftConstant\ c\_constant\_null$

dest_variable
mk_variable
dest_constant$_{ec}$
mk_constant$_{ec}$

$\vdash$ *mk_variable* = *InL*
$\wedge$ $mk\_constant_{ec}$ = *InR*
$\wedge$ *dest_variable* = *OutL*
$\wedge$ $dest\_constant_{ec}$ = *OutR*

WHEN_variable
WHEN_constant$_{ec}$

$\vdash$ \$*WHEN_variable* = *Lift* (*mk_variable*, *dest_variable*)
$\wedge$ \$$WHEN_constant_{ec}$
= *Lift* ($mk\_constant_{ec}$, $dest\_constant_{ec}$)

dest_ands
mk_ands
dest_ors
mk_ors
dest_simple
mk_simple $\vdash$ *mk_ands* = *InL*
$\wedge$ *mk_ors* = *InR o InL*
$\wedge$ *mk_simple* = *InR o InR*
$\wedge$ *dest_ands* = *OutL*
$\wedge$ *dest_ors* = *OutL o OutR*
$\wedge$ *dest_simple* = *OutR o OutR*

WHEN_ands
WHEN_ors
WHEN_simple $\vdash$ \$*WHEN_ands* = *Lift* (*mk_ands*, *dest_ands*)
$\wedge$ \$*WHEN_ors* = *Lift* (*mk_ors*, *dest_ors*)
$\wedge$ \$*WHEN_simple* = *Lift* (*mk_simple*, *dest_simple*)

c_anon$_{tn}$
dest_name$_{tn}$
mk_name$_{tn}$ $\vdash$ c_anon_{tn} = *InL maybe*
$\wedge$ mk_name_{tn} = *InR*
$\wedge$ $dest\_name_{tn}$ = *OutR*

WHEN_anon$_{tn}$
WHEN_name$_{tn}$

$\vdash$ $WHEN\_anon_{tn}$ = *LiftConstant* c_anon_{tn}
$\wedge$ \$$WHEN_name_{tn}$
= *Lift* (mk_name_{tn}, $dest\_name_{tn}$)

TableDetail $\vdash \exists f\bullet$ *TypeDefn* (λ *x*$\bullet$ *true*) *f*
MkTableDetail
td_tableName
td_corrName
td_genCorr
td_info
td_columns
td_rowClass
td_implementation
td_constraints

$\vdash \forall$ *t x1 x2 x3 x4 x5 x6 x7 x8*
$\bullet$ *td_tableName*
(*MkTableDetail x1 x2 x3 x4 x5 x6 x7 x8*)

$$
\begin{array}{l}
\quad = x1 \\
\wedge\ td\_corrName \\
\quad\quad (MkTableDetail\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8) \\
\quad = x2 \\
\wedge\ td\_genCorr \\
\quad\quad (MkTableDetail\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8) \\
\quad = x3 \\
\wedge\ td\_info\ (MkTableDetail\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8) \\
\quad = x4 \\
\wedge\ td\_columns \\
\quad\quad (MkTableDetail\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8) \\
\quad = x5 \\
\wedge\ td\_rowClass \\
\quad\quad (MkTableDetail\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8) \\
\quad = x6 \\
\wedge\ td\_implementation \\
\quad\quad (MkTableDetail\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8) \\
\quad = x7 \\
\wedge\ td\_constraints \\
\quad\quad (MkTableDetail\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8) \\
\quad = x8 \\
\wedge\ MkTableDetail \\
\quad\quad (td\_tableName\ t) \\
\quad\quad (td\_corrName\ t) \\
\quad\quad (td\_genCorr\ t) \\
\quad\quad (td\_info\ t) \\
\quad\quad (td\_columns\ t) \\
\quad\quad (td\_rowClass\ t) \\
\quad\quad (td\_implementation\ t) \\
\quad\quad (td\_constraints\ t) \\
\quad = t
\end{array}
$$

IdentDetail $\vdash \exists\, f \bullet\ TypeDefn\ (\lambda\ x \bullet\ true)\ f$

MkIdentDetail

id_identName

id_info

id_lub$_{id}$

id_vName

id_cName

$$
\begin{array}{l}
\vdash \forall\ t\ x1\ x2\ x3\ x4\ x5 \\
\quad \bullet\ id\_identName\ (MkIdentDetail\ x1\ x2\ x3\ x4\ x5) = x1 \\
\quad\quad \wedge\ id\_info\ (MkIdentDetail\ x1\ x2\ x3\ x4\ x5) = x2 \\
\quad\quad \wedge\ id\_lub_{id}\ (MkIdentDetail\ x1\ x2\ x3\ x4\ x5) = x3 \\
\quad\quad \wedge\ id\_vName\ (MkIdentDetail\ x1\ x2\ x3\ x4\ x5) = x4 \\
\quad\quad \wedge\ id\_cName\ (MkIdentDetail\ x1\ x2\ x3\ x4\ x5) = x5 \\
\quad\quad \wedge\ MkIdentDetail \\
\quad\quad\quad (id\_identName\ t) \\
\quad\quad\quad (id\_info\ t) \\
\quad\quad\quad (id\_lub_{id}\ t) \\
\quad\quad\quad (id\_vName\ t)
\end{array}
$$

$$
\begin{array}{ll}
 & \quad (id\_cName\ t) \\
 & \quad = t \\
\textbf{Scope} & \vdash \exists f \bullet\ TypeDefn\ (\lambda\ x \bullet\ true)\ f \\
\textbf{MkScope} & \\
\textbf{s\_tables} & \\
\textbf{s\_identifiers} & \\
 & \vdash \forall\ t\ x1\ x2 \\
 & \quad \bullet\ s\_tables\ (MkScope\ x1\ x2) = x1 \\
 & \qquad \wedge\ s\_identifiers\ (MkScope\ x1\ x2) = x2 \\
 & \qquad \wedge\ MkScope\ (s\_tables\ t)\ (s\_identifiers\ t) = t \\
\textbf{ParamInfo} & \vdash \exists f \bullet\ TypeDefn\ (\lambda\ x \bullet\ true)\ f \\
\textbf{MkParamInfo} & \\
\textbf{pi\_name} & \\
\textbf{pi\_val}_{\mathbf{p}} & \\
\textbf{pi\_clasf} & \vdash \forall\ t\ x1\ x2\ x3 \\
 & \quad \bullet\ pi\_name\ (MkParamInfo\ x1\ x2\ x3) = x1 \\
 & \qquad \wedge\ pi\_val_p\ (MkParamInfo\ x1\ x2\ x3) = x2 \\
 & \qquad \wedge\ pi\_clasf\ (MkParamInfo\ x1\ x2\ x3) = x3 \\
 & \qquad \wedge\ MkParamInfo \\
 & \qquad\quad (pi\_name\ t) \\
 & \qquad\quad (pi\_val_p\ t) \\
 & \qquad\quad (pi\_clasf\ t) \\
 & \qquad = t \\
\textbf{ST\_STACK} & \vdash \exists f \bullet\ TypeDefn\ (\lambda\ x \bullet\ true)\ f \\
\textbf{MkST\_STACK} & \\
\textbf{symbolTable} & \\
\textbf{parameterTable} & \\
 & \vdash \forall\ t\ x1\ x2 \\
 & \quad \bullet\ symbolTable\ (MkST\_STACK\ x1\ x2) = x1 \\
 & \qquad \wedge\ parameterTable\ (MkST\_STACK\ x1\ x2) = x2 \\
 & \qquad \wedge\ MkST\_STACK\ (symbolTable\ t)\ (parameterTable\ t) \\
 & \qquad = t
\end{array}
$$

8 INDEX