Project: DRA FRONT END FILTER PROJECT

Title: Specification of Query Transformations in HOL (II)

Ref: DS/FMU/FEF/029 *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: A specification of the SSQL Query Transformations in HOL 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/004. *Specification of SSQL Semantics I.* G.M. Prout, ICL Secure Systems, WIN01.

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

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

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

[7] DS/FMU/FEF/022. *SWORD Front End Architectural Model.* R.D. Arthan, ICL Secure Systems, WIN01.

[8] DS/FMU/FEF/028. *Specification of Query Transformations in HOL (I).* R.D. Arthan, ICL Secure Systems, WIN01.

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

[10] *The Specification of Secure SQL.* Simon Wiseman, DRA, 6th July 1992.

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

0.3 Changes History

Issue 1.1 (21 May 1993) First draft.

Issue 1.9 (23 August 1993) Minor correction and fix in light of comments received.

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 a formal specification in HOL ([2], [1]) of the SSQL query transformations of [11]. It constitutes part of deliverable D11 of work package 3, as given in the Proposal for Phase 2, [5].

The current version of this document is gives only a partial treatment to indicate the flavour of what can be done in this area. In particular, the main mutually recursive functions which perform the transformations are in vestigial form, since their structure is heavily dependent on the SSQL syntax and it was felt best to defer their formalisation in HOL until the SSQL syntax definitions [4] have been brought into line with the most recent specification of SSQL.

During discussions with DRA, it was agreed that this area of the specification work in HOL was adequately covered for the purposes of Phase 2, and that attention should be concentrated on the areas were proof work can be carried out within phase 2 resource limitations.

1.2 Introduction

We provide HOL specifications of the SSQL query transformations of [11]. Preliminary material needed to support these query transformation specifications may be found in [8].

1.3 ProofPower Preamble

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

SML

```
open_theory"fef028";
(force_delete_theory "fef029" handle _ => ());
new_theory"fef029";
set_pc"hol";
```

2 THE TRANSFORMATIONS

The following error values are required in addition to those introduced in [3].

HOL Constant

internalError	**wrongWorth**	**notTrigger**
onlyInTriggers	**notMonadic**	**notDyadic**
notTriadic	**notSetFunction**	**noSuchParameter**
noScope	**wrongScope**	**ambiguousName**
emptyUnionList	:	*Error*

---

true

2.1 Incompletely Specified Transformations

Some functions have not been specified in [11]. They are loosely defined here. Some of them depend on the state of the transformation process, this is modelled by the following labelled product type. Of the four components of the state, only the first, the symbol table stack, varies during the transformation, the other components simply record information supplied as parameters to the overall transformation process. The projection functions for these parameter components serve for the functions of the same name from [8, 11].

HOL Labelled Product

TRANS_STATE

st_stack	: *ST_STACK*;
query_class	: *Class*;
query_constants_class	: *Class*;
client_clearance	: *Class*

HOL Constant

check_enum : *Enum* × *Int* × *Table_spec* → *BOOL*;
check_fixed : *Fixed* × *Int* × *Int* → *BOOL*;
check_floating : *Float* × *Int* × *Int* × *Int* → *BOOL*;
check_interval : *Interval* × *STRING* → *BOOL*;
check_time : *Time* × *STRING* → *BOOL*;
timeFormatToInterval : *STRING* → *STRING*;
unique_name : *TRANS_STATE* → *STRING*;
contextual_data : *STRING* → *Value* × *Class*;
default_directory : *STRING LIST*

---

true

Notes:

1. *check_enum* is, presumably, parameterised by the structure *and content* of the database. The above parameterisation is therefore not very realistic, since it only permits dependence on the structure of the database (presumed fixed), not its contents.
2. *unique_name* parameterised as above could only produce names which were unique within the current symbol-table scope, not within the entire transformation process.

We need a function to set up an initial state based on the client clearance:

HOL Constant

init_trans_state: *Class* → *TRANS_STATE*

∀*c*•
init_trans_state c =
MkTRANS_STATE
(*MkST_STACK* [] [])
c
c
c

2.2 Symbol Table Model

HOL Constant

find$_{\mathbf{column}}$: *ColumnSpecification* × *TableDetail LIST* →
(*TableDetail* × *SsqlCol* × *TsqlCol*)*LIST*

∀*cs tdl*•
find$_{column}$(*cs*,*tdl*) =
let *look* (*n*, *td*, *scl*, *tcl*) =
let *sctcl* = *splice scl tcl*
in let *sctcl*′ = *sctcl* ↾ {(*sc*, *tc*) | *sc_name sc* = *mk_name*$_s$ *n*}
in Map (λ(*sc*, *tc*)• (*td*, *sc*, *tc*)) *sctcl*′
in let *do1td td* =
CASE cs [
WHEN_anonymous_column col•
look(*col*, *td*, *td_columns td*, *td_implementation td*);
WHEN_specific (*ts*, *col*)•
CASE (*td_tableName td*) [
WHEN_name$_{tn}$ *ts*•
look(*col*, *td*, *td_columns td*, *td_implementation td*);
OTHERS []
]
]
in *Flat*(*Map do1td tdl*)

HOL Constant

$$\mathbf{find_{ident}} : \mathit{ColumnSpecification} \times \mathit{IdentDetail\ LIST} \rightarrow \mathit{IdentDetail\ LIST}$$

$$\forall cs\ idl \bullet$$
$$\mathit{find}_{ident}\ (cs,\ idl) =$$
$$\mathit{CASE}\ cs\ [$$
$$\mathit{WHEN\_specific}\ (t,\ c) \bullet\ [];$$
$$\mathit{WHEN\_anonymous\_column}\ col \bullet$$
$$\quad idl \upharpoonright \{id \mid col = \mathit{id\_identName}\ id\}$$
$$]$$

We replace the function *look*, which is private to $\mathit{lookup}_{column\_info}$ in [9, 11] by a top-level function, *lookup_column_info_look* (since it is recursive and HOL local functions cannot be recursive).

HOL Constant

$$\mathbf{lookup\_column\_info\_look} :$$
$$\mathit{ColumnSpecification}\ \times\ \mathit{Scope\ LIST} \rightarrow (\mathit{TableInfo} \times \mathit{SsqlCol})\mathit{RESULT}$$

$$\forall cs\ ti\ outer \bullet$$
$$\mathit{lookup\_column\_info\_look}\ (cs,\ []) = \mathit{Exception}\ [\mathit{noSuchColumn}]$$
$$\wedge\quad \mathit{lookup\_column\_info\_look}\ (cs,\ outer \frown [ti]) =$$
$$\mathit{let}\quad t = \mathit{s\_tables}\ ti$$
$$\mathit{in\ let}\quad \mathit{tdsctcl} = \mathit{find}_{column}(cs,\ t)$$
$$\mathit{in}\quad \mathit{if}\quad \mathit{tdsctcl} = []$$
$$\quad \mathit{then}\quad \mathit{lookup\_column\_info\_look}\ (cs,\ outer)$$
$$\quad \mathit{else\ if}\quad \mathit{Tl}(\mathit{tdsctcl}) = []$$
$$\quad \mathit{then}\quad \mathit{let}\ (td,\ sc,\ tc) = \mathit{Hd}\ \mathit{tdsctcl}\ \mathit{in}\ \mathit{Ok}(\mathit{td\_info}\ td,\ sc)$$
$$\quad \mathit{else}\quad \mathit{Exception}\ [\mathit{ambiguousName}]$$

HOL Constant

$$\mathbf{lookup_{columninfo}} :$$
$$\mathit{TRANS\_STATE} \rightarrow \mathit{ColumnSpecification} \rightarrow (\mathit{TableInfo} \times \mathit{SsqlCol})\mathit{RESULT}$$

$$\forall st\ cs \bullet \mathit{lookup}_{columninfo}\ st\ cs =$$
$$\mathit{lookup\_column\_info\_look}\ (cs,\ \mathit{symbolTable}(\mathit{st\_stack}\ st))$$

HOL Constant

maxBound : *BoundInfo* → *Class*

$$\forall bi \bullet \quad maxBound\ bi =$$
$$CASE\ bi\ [$$
$$WHEN\_upb\ c \bullet\ c;$$
$$WHEN\_constant\ c \bullet\ c$$
$$]$$

HOL Constant

innermost : *Scope LIST* → *Scope LIST*

$$\forall outer\ inner \bullet$$
$$innermost\ [] = []$$
$$\wedge \quad innermost\ (outer \frown [inner]) =$$
$$let\ tds = s\_tables\ inner$$
$$in \quad if \quad tds = []$$
$$then \quad innermost\ outer \frown [inner]$$
$$else \quad [inner]$$

As with $lookup_{column\_info}$, we replace the recursive function *look* local to $lookup_{col\_info}$ by a top-level function *look_up_col_spec_class_look*. The comments before the *then*s below show the corresponding ML pattern match (cf. [6]).

HOL Constant

```
        lookup_col_spec_class_look
                : ColumnSpecification  × Scope LIST → (TsqlRepr × Class) RESULT
─────────────────────────────────────────────────────────────────────────────
∀ cs outer ti•
        lookup_col_spec_class_look(cs,[]) = Exception [noSuchColumn]
∧       lookup_col_spec_class_look(cs, outer ⌢ [ti]) =
        let     (t, i) = (s_tables ti, s_identifiers ti)
        in let  cds = find_column(cs,t)
        and     ids = find_ident(cs,i)
        in      if      ids = []
                then    if      cds = []
(* ([], []) *)          then    lookup_col_spec_class_look(cs, outer)
                        else    if      Tl cds = []
(* ([(td,sc,tc)], []) *) then   let     (td, sc, tc) = Hd cds
                                        in let  u = maxBound (sc_col_class sc)
                                        in      CASE (tc_class_name tc) [
                                                WHEN_anon_tc
                                                        (Exception[internalError]);
                                                WHEN_name_tc s•
                                                        Ok(mk_column(td_genCorr td, s), u);
                                                WHEN_constant_tc c•
                                                        Ok(mk_constant_class c,u)
                                                ]
(* (xs, []) *)                                  else    Exception [ambiguousName]
                else    if      cds = [] ∧ Tl ids = []
(* ([], [id]) *)        then    let     id = Hd ids
                                        in let  cl = id_lub_id id
                                        in      CASE (id_cName id) [
                                                WHEN_none_t (Ok(mk_constant_class cl, cl));
                                                WHEN_anon_t (Exception[internalError]);
                                                WHEN_name_t s•Ok(mk_local_identifier s, cl)
                                                ]
(* (xs, ys) *)          else    Exception [ambiguousName]
```

HOL Constant

$$\mathbf{lookup_{colspecclass}} : TRANS\_STATE \rightarrow (BOOL \times ColumnSpecification) \rightarrow (TsqlRepr \times Class)RESULT$$

$$\forall st\ flg\ cs \bullet$$
$$lookup_{colspecclass}\ st\ (flg,\ cs) =$$
$$\begin{array}{ll} if & flg \\ then & lookup\_col\_spec\_class\_look(cs,\ innermost(symbolTable(st\_stack\ st))) \\ else & lookup\_col\_spec\_class\_look(cs,\ (symbolTable(st\_stack\ st))) \end{array}$$

Mutatatis mutandis, the remarks about *lookup_col_spec_class_look* above apply both to to other functions with names of the form XXX_look below.

HOL Constant

lookup_col_spec_dinary_look
: *ColumnSpecification* × *Scope LIST* → *TsqlRepr RESULT*

```
∀ cs outer ti•
    lookup_col_spec_dinary_look(cs,[]) = Exception [noSuchColumn]
∧   lookup_col_spec_dinary_look(cs, outer ⌢ [ti]) =
    let     (t, i) = (s_tables ti, s_identifiers ti)
    in let  cds = find_column(cs,t)
    and     ids = find_ident(cs,i)
    in      if      ids = []
            then    if      cds = []
(* ([], []) *)     then    lookup_col_spec_dinary_look(cs, outer)
                    else    if      Tl cds = []
(* ([(td,sc,tc)], []) *)   then    let     (td, sc, tc) = Hd cds
                                    in      CASE (tc_dinary_name tc) [
                                            WHEN_none_t
                                                (Exception[internalError]);
                                            WHEN_anon_t
                                                (Exception[internalError]);
                                            WHEN_name_t s•
                                                Ok(mk_column(td_genCorr td, s))
                                            ]
(* (xs, []) *)                              else    Exception [ambiguousName]
            else    if      cds = [] ∧ Tl ids = []
(* ([], [id]) *)   then    let     id = Hd ids
                                    in let  (st, w) = id_info id
                                    in      if      w = dinary
                                            then    Ok(mk_local_identifier(id_vName id))
                                            else    Ok(c_constant_null)
(* (xs, ys) *)             else    Exception [ambiguousName]
```

HOL Constant

lookup$_{\text{colspecdinary}}$:
TRANS_STATE → (*BOOL* × *ColumnSpecification*) → *TsqlRepr RESULT*

```
∀st flg cs•
    lookup_colspecdinary st (flg, cs) =
    if      flg
    then    lookup_col_spec_dinary_look(cs, innermost(symbolTable(st_stack st)))
    else    lookup_col_spec_dinary_look(cs, (symbolTable(st_stack st)))
```

HOL Constant

lookup_col_spec_sterling_look
: *ColumnSpecification* × *Scope LIST* → *TsqlRepr RESULT*

```
∀ cs outer ti•
        lookup_col_spec_sterling_look(cs,[]) = Exception [noSuchColumn]
∧       lookup_col_spec_sterling_look(cs, outer ⌢ [ti]) =
        let     (t, i) = (s_tables ti, s_identifiers ti)
        in let  cds = find_column(cs,t)
        and     ids = find_ident(cs,i)
        in      if      ids = []
                then    if      cds = []
(* ([], []) *)          then    lookup_col_spec_sterling_look(cs, outer)
                        else    if      Tl cds = []
(* ([(td,sc,tc)], []) *) then   let     (td, sc, tc) = Hd cds
                                        in      CASE (tc_sterling_name tc) [
                                                WHEN_none_t
                                                        (Exception[internalError]);
                                                WHEN_anon_t
                                                        (Exception[internalError]);
                                                WHEN_name_t s•
                                                        Ok(mk_column(td_genCorr td, s))
                                                ]
(* (xs, []) *)                          else    Exception [ambiguousName]
                else    if      cds = [] ∧ Tl ids = []
(* ([], [id]) *)        then    let     id = Hd ids
                                in let  (st, w) = id_info id
                                in      if      w = sterling
                                        then    Ok(mk_local_identifier(id_vName id))
                                        else    Ok(c_constant_null)
(* (xs, ys) *)          else    Exception [ambiguousName]
```

HOL Constant

lookup$_{\text{colspecsterling}}$:
TRANS_STATE → (*BOOL* × *ColumnSpecification*) → *TsqlRepr RESULT*

```
∀st flg cs•
        lookup_colspecsterling st (flg, cs) =
        if      flg
        then    lookup_col_spec_sterling_look(cs, innermost(symbolTable(st_stack st)))
        else    lookup_col_spec_sterling_look(cs, (symbolTable(st_stack st)))
```

HOL Constant

lookup$_{\mathbf{localcolimplementation}}$
: $TRANS\_STATE \rightarrow TsqlCol\ LIST\ RESULT$

$\forall\ st \bullet\ lookup_{localcolimplementation}\ st =$
  let $extract\_implementation\ sc =$
    $Fold\ (\$^\frown)\ (Map\ td\_implementation\ (s\_tables\ sc))\ []$
  in let $trs =$
    $Fold\ (\$^\frown)$
    $(Map\ extract\_implementation\ (innermost(symbolTable(st\_stack\ st))))[]$
  in *if* $trs = []$
    then $Exception[noScope]$
    else $Ok\ trs$

HOL Constant

lookup$_{\mathbf{localcolinfo}}$
: $TRANS\_STATE \rightarrow SsqlCol\ LIST\ RESULT$

$\forall\ st \bullet\ lookup_{localcolinfo}\ st =$
  let $extract\_columns\ sc =$
    $Fold\ (\$^\frown)\ (Map\ td\_columns\ (s\_tables\ sc))\ []$
  in let $trs =$
    $Fold\ (\$^\frown)$
    $(Map\ extract\_columns\ (innermost(symbolTable(st\_stack\ st))))[]$
  in *if* $trs = []$
    then $Exception[noScope]$
    else $Ok\ trs$

HOL Constant

lookup$_{\mathbf{localcolspecclasses}}$
: $TRANS\_STATE \rightarrow (TsqlRepr \times Class)\ LIST\ RESULT$

$\forall\ st \bullet\ lookup_{localcolspecclasses}\ st =$
  let $look2(corr,sc,tc) =$
    let $u = maxBound(sc\_col\_class\ sc)$
    in $CASE\ (tc\_class\_name\ tc)\ [$
      $WHEN\_anon_{tc}\ (Exception[internalError]);$
      $WHEN\_name_{tc}\ s \bullet\ Ok(mk\_column(corr,\ s),\ u);$
      $WHEN\_constant_{tc}\ c \bullet\ Ok(mk\_constant\_class\ c,u)$
    $]$
  in let $look1\ td = at3\ (Map\ look2)$
    $(seq\ (Length\ (td\_columns\ td))\ (td\_genCorr\ td),$
      $td\_columns\ td,\ td\_implementation\ td)$
  in let $look\ sc = Fold\ (\$^\frown)\ (Map\ look1\ (s\_tables\ sc))\ []$
  in let $trs =$
    $Fold\ (\$^\frown)\ (Map\ look\ (innermost(symbolTable(st\_stack\ st))))[]$
  in *if* $trs = []$
    then $Exception[noScope]$
    else $ListOk\ trs$

HOL Constant

lookup$_{\mathbf{localcolspecsterlings}}$
: *TRANS_STATE* → *TsqlRepr LIST RESULT*

∀ *st*• *lookup*$_{localcolspecsterlings}$ *st* =
let *look2*(*corr*,*tc*) =
CASE (*tc_sterling_name tc*) [
WHEN_none$_t$ (*Ok c_constant_null*);
WHEN_anon$_t$ (*Exception*[*internalError*]);
WHEN_name$_t$ *s*• *Ok*(*mk_column*(*corr*, *s*))
]
in let *look1 td* = *at2* (*Map look2*)
(*seq* (*Length* (*td_columns td*)) (*td_genCorr td*),
td_implementation td)
in let *look sc* = *Fold* (\$⌢) (*Map look1* (*s_tables sc*)) []
in let *trs* =
Fold (\$⌢) (*Map look* (*innermost*(*symbolTable*(*st_stack st*))))[]
in *if* *trs* = []
then *Exception*[*noScope*]
else *ListOk trs*

HOL Constant

lookup$_{\mathbf{localrowclasses}}$
: *TRANS_STATE* → *TsqlRepr LIST RESULT*

∀ *st*• *lookup*$_{localrowclasses}$ *st* =
let *look1 td* =
CASE (*td_rowClass td*) [
WHEN_anon$_{tc}$ (*Exception*[*internalError*]);
WHEN_name$_{tc}$ *s*• *Ok*(*mk_column*(*td_genCorr td*, *s*));
WHEN_constant$_{tc}$ *c*• *Ok*(*mk_constant_class c*)
]
in let *look sc* = *Map look1* (*s_tables sc*)
in let *trs* =
Fold (\$⌢) (*Map look* (*innermost*(*symbolTable*(*st_stack st*))))[]
in *if* *trs* = []
then *Exception*[*noScope*]
else *ListOk trs*

Again *mutatatis mutandis*, the remarks about *lookup_col_spec_class_look* above apply to to the following.

HOL Constant

lookup_column_row_class_look
: *TRANS_STATE* → *ColumnSpecification* × *Scope LIST* → *TsqlRepr RESULT*

```
∀ st cs outer ti•
    lookup_column_row_class_look st (cs,[]) = Exception [noSuchColumn]
∧   lookup_column_row_class_look st (cs, outer ⌢ [ti]) =
    let     (t, i) = (s_tables ti, s_identifiers ti)
    in let  cds = find_column(cs,t)
    and     ids = find_ident(cs,i)
    in      if      ids = []
            then    if      cds = []
(* ([], []) *)     then    lookup_column_row_class_look st (cs, outer)
                    else    if      Tl cds = []
(* ([(td,sc,tc)], []) *)   then    let     (td, sc, tc) = Hd cds
                                    in      CASE (td_rowClass td) [
                                            WHEN_anon_tc
                                                (Exception[internalError]);
                                            WHEN_name_tc s•
                                                Ok(mk_column(td_genCorr td, s));
                                            WHEN_constant_tc c•
                                                Ok(mk_constant_class c)
                                           ]
(* (xs, []) *)                              else    Exception [ambiguousName]
            else    if      cds = [] ∧ Tl ids = []
(* ([], [id]) *)   then    let     id = Hd ids
                                    in      Ok(mk_constant_class(query_class st))
(* (xs, ys) *)             else    Exception [ambiguousName]
```

HOL Constant

lookup$_{\text{columnrowclass}}$
: *TRANS_STATE* → (*BOOL* × *ColumnSpecification*) → *TsqlRepr RESULT*

```
∀st flg cs•
    lookup_columnrowclass st (flg, cs) =
    if      flg
    then    lookup_column_row_class_look st (cs, innermost(symbolTable(st_stack st)))
    else    lookup_column_row_class_look st (cs, (symbolTable(st_stack st)))
```

Again *mutatatis mutandis*, the remarks about *lookup_col_spec_class_look* above apply to to the following.

HOL Constant

lookup_table_row_class_look
: *TableSpecification* × *Scope LIST* → *TsqlRepr RESULT*

```
∀ts outer ti•
    lookup_table_row_class_look (ts, []) = Exception [noSuchTable]
∧   lookup_table_row_class_look (ts, outer ⌢ [ti]) =
    let   look1 (ts, td) =
              CASE (td_tableName td) [
              WHEN_anon_tn [];
              WHEN_name_tn tn•
                  if    ts = tn
                  then  CASE (td_rowClass td) [
                        WHEN_anon_tc [Exception[internalError]];
                        WHEN_name_tc s•
                              [Ok(mk_column(td_genCorr td, s))];
                        WHEN_constant_tc c•
                              [Ok(mk_constant_class c)]
                        ]
                  else  []
              ]
    in let  (tds, ids) = (s_tables ti, s_identifiers ti)
    in let  trs = Fold ($⌢) (at2 (Map look1) (seq (Length tds) (ts), tds)) []
    in      if    trs = [] then lookup_table_row_class_look (ts, outer)
            else if Tl trs = []
            then  Hd trs
            else  Exception[ambiguousName]
```

HOL Constant

lookup$_{\text{tablerowclass}}$
: *TRANS_STATE* → (*BOOL* × *TableSpecification*) → *TsqlRepr RESULT*

```
∀st flg ts•
    lookup_tablerowclass st (flg, ts) =
    if    flg
    then  lookup_table_row_class_look (ts, innermost(symbolTable(st_stack st)))
    else  lookup_table_row_class_look (ts, (symbolTable(st_stack st)))
```

HOL Constant

lookup_table_detail_look
: $TableSpecification \times Scope\ LIST \rightarrow TableDetail\ RESULT$

$\forall ts\ outer\ ti \bullet$
$lookup\_table\_detail\_look\ (ts, []) = Exception\ [noSuchTable]$
$\wedge\ lookup\_table\_detail\_look\ (ts, outer \frown [ti]) =$
$let\ \ look1\ (ts, td) =$
$CASE\ (td\_tableName\ td)\ [$
$WHEN\_anon_{tn}\ [];$
$WHEN\_name_{tn}\ tn \bullet$
$if\ \ ts = tn$
$then\ \ [td]$
$else\ \ []$
$]$
$in\ let\ \ (tds, ids) = (s\_tables\ ti, s\_identifiers\ ti)$
$in\ let\ \ tis = Fold\ (\$\frown)\ (at2\ (Map\ look1)\ (seq\ (Length\ tds)\ (ts), tds))\ []$
$in\ \ if\ \ tis = []\ then\ lookup\_table\_detail\_look\ (ts, outer)$
$else\ if\ \ Tl\ tis = []$
$then\ \ Ok(Hd\ tis)$
$else\ \ Exception[ambiguousName]$

HOL Constant

lookup$_{\mathbf{tabledetail}}$
: $TRANS\_STATE \rightarrow TableSpecification \rightarrow TableDetail\ RESULT$

$\forall st\ ts \bullet$
$lookup_{tabledetail}\ st\ ts =$
$lookup\_table\_detail\_look\ (ts, (symbolTable(st\_stack\ st)))$

2.3 Symbol Table Operations

The following operation on state is a convenient short-hand in defining these operations. It replaces the top scope in the symbol table in a state with a given value.

HOL Constant

$$\mathbf{update\_top\_scope} : \mathit{TRANS\_STATE} \to \mathit{Scope} \to \mathit{TRANS\_STATE}$$

$$\begin{array}{l}
\forall st\ sc \bullet \\
\quad update_top_scope\ st\ sc = \\
\quad let \quad stk = st_stack\ st \\
\quad in\ let \quad symt = symbolTable\ stk \\
\quad in\ let \quad outer = Rev(Tl(Rev\ symt)) \\
\quad in\ let \quad symt' = outer \frown [sc] \\
\quad in \quad MkTRANS_STATE \\
\qquad\quad (MkST_STACK\ symt'\ (parameterTable\ stk)) \\
\qquad\quad (query_class\ st)\ (query_constants_class\ st)\ (client_clearance\ st)
\end{array}$$

The local function *find* used in several places in the following has been misspelt as *fynd* to avoid clashing with the constant *find* defined in [4].

HOL Constant

$$\mathbf{enter_{identifier}} : TRANS\_STATE \rightarrow (STRING \times ExpType \times Class) \rightarrow (TRANS\_STATE \times STRING \times STRING)RESULT$$

$$\begin{array}{l}
\forall st\ name\ et\ up \bullet \\
\quad enter_{identifier}\ st\ (name,et,up) = \\
\quad let\ \ fynd\ (n,id) = if\ n = id_identName\ id\ then\ \ [id]\ else\ [] \\
\quad in\ let\ \ sl = symbolTable(st_stack\ st) \\
\quad in\ \ \ if\ \ \ sl = [] \\
\qquad then\ \ Exception[noScope] \\
\qquad else\ \ let\ \ \ outer = Rev(Tl(Rev\ sl)) \\
\qquad\quad in\ let\ \ \ ti = Hd(Rev\ sl) \\
\qquad\quad in\ let\ \ (tds,\ ids) = (s_tables\ ti,\ s_identifiers\ ti) \\
\qquad\quad in\ \ \ if\ \ \ tds = [] \\
\qquad\qquad then\ \ let\ \ \ unv = unique_name\ st \\
\qquad\qquad\quad in\ let\ \ unc = unique_name\ st \\
\qquad\qquad\quad in\ let\ \ id = \\
\qquad\qquad\qquad\quad MkIdentDetail \\
\qquad\qquad\qquad\quad name\ et\ up\ unv\ (mk_name_t\ unc) \\
\qquad\qquad\quad in\ \ \ if\ \ \ \neg \\
\qquad\qquad\qquad\quad at2\ (Map\ fynd) \\
\qquad\qquad\qquad\quad (seq(Length\ ids)name,ids) \\
\qquad\qquad\qquad\quad = [] \\
\qquad\qquad\qquad then\ \ Exception[ambiguousName] \\
\qquad\qquad\qquad else\ \ let\ \ \ sc' = MkScope\ []\ (ids \frown [id]) \\
\qquad\qquad\qquad\quad in\ \ \ Ok(update_top_scope\ st\ sc',\ unv,unc) \\
\qquad\qquad then\ \ Exception[wrongScope]
\end{array}$$

HOL Constant

$$\mathbf{enter}_{\mathbf{identiferconstantclass}} : TRANS\_STATE \rightarrow (STRING \times ExpType \times Class) \rightarrow (TRANS\_STATE \times STRING)RESULT$$

$$\begin{array}{l}
\forall st\ name\ et\ clasf \bullet \\
\quad enter_{identiferconstantclass}\ st\ (name,et,clasf) = \\
\quad let\ \ fynd\ (n,id) = if\ n = id_identName\ id\ then\ [id]\ else\ [] \\
\quad in\ let\ \ sl = symbolTable(st_stack\ st) \\
\quad in\ \ if\ \ sl = [] \\
\quad\quad then\ \ Exception[noScope] \\
\quad\quad else\ \ let\ \ outer = Rev(Tl(Rev\ sl)) \\
\quad\quad\quad in\ let\ \ ti = Hd(Rev\ sl) \\
\quad\quad\quad in\ let\ \ (tds,\ ids) = (s_tables\ ti,\ s_identifiers\ ti) \\
\quad\quad\quad in\ \ if\ \ tds = [] \\
\quad\quad\quad\quad then\ \ let\ \ unv = unique_name\ st \\
\quad\quad\quad\quad\quad in\ let\ \ id = \\
\quad\quad\quad\quad\quad\quad MkIdentDetail \\
\quad\quad\quad\quad\quad\quad name\ et\ clasf\ unv\ c_anon_t \\
\quad\quad\quad\quad\quad in\ \ if\ \ \neg \\
\quad\quad\quad\quad\quad\quad at2\ (Map\ fynd) \\
\quad\quad\quad\quad\quad\quad (seq(Length\ ids)name,ids) \\
\quad\quad\quad\quad\quad\quad = [] \\
\quad\quad\quad\quad\quad then\ \ Exception[ambiguousName] \\
\quad\quad\quad\quad\quad else\ \ let\ \ sc' = MkScope\ []\ (ids \frown [id]) \\
\quad\quad\quad\quad\quad\quad in\ \ Ok\ (update_top_scope\ st\ sc',\ unv) \\
\quad\quad\quad\quad else\ \ Exception[wrongScope]
\end{array}$$

HOL Constant

$$\mathbf{extract}_{\mathbf{parameter}} : STRING \times ParamInfo\ LIST \rightarrow ParamInfo\ LIST$$

$$\forall name\ l \bullet \quad extract_{parameter}\ (name,l) = l \upharpoonright \{pi \mid pi\_name\ pi = name\}$$

HOL Constant

$$\mathbf{enter_{parameter}}$$
$$: TRANS\_STATE \rightarrow STRING \times Value \times Class \rightarrow TRANS\_STATE\ RESULT$$

$$\forall st\ name\ v\ clasf \bullet$$
$$enter_{parameter}\ st\ (name,v,clasf) =$$
$$\begin{array}{lll} let & sl = symbolTable(st\_stack\ st) & \\ in\ let & pt = parameterTable(st\_stack\ st) & \\ in & if & sl = [] \\ & then & Exception[noScope] \\ & else\ if & extract_{parameter}(name,\ pt) = [] \\ & then & Ok(MkTRANS\_STATE \\ & & (MkST\_STACK\ sl\ (pt \frown [MkParamInfo\ name\ v\ clasf])) \\ & & (query\_class\ st)\ (query\_constants\_class\ st) \\ & & (client\_clearance\ st)) \\ & else & Exception[ambiguousName] \end{array}$$

HOL Constant

$$\mathbf{enter_{corrtable}}$$
$$: TRANS\_STATE$$
$$\rightarrow (STRING \times TableName \times TableInfo \times$$
$$SsqlCol\ LIST \times TsqlClassName \times TsqlCol\ LIST)$$
$$\rightarrow (TRANS\_STATE \times STRING)RESULT$$

$$\forall st\ cn\ ts\ ti\ scs\ rcn\ tcs \bullet$$
$$enter_{corrtable}\ st\ (cn,\ ts,\ ti,\ scs,\ rcn,\ tcs) =$$
$$\begin{array}{lllll} let & sl = symbolTable(st\_stack\ st) & & & \\ in & if & sl = [] & & \\ & then & Exception[noScope] & & \\ & else & let & outer = Rev(Tl(Rev\ sl)) & \\ & & in\ let & t = Hd(Rev\ sl) & \\ & & in\ let & (tds,\ ids) = (s\_tables\ t,\ s\_identifiers\ t) & \\ & & in & if & ids = [] \\ & & & then & let \quad gc = unique\_name\ st \\ & & & & in\ let \quad td = \\ & & & & \quad MkTableDetail \\ & & & & \quad ts\ (mk\_name_s\ cn)\ gc\ ti\ scs\ rcn \\ & & & & \quad tcs\ (MkConstraintInfo[][][][][]) \\ & & & & in\ let \quad sc' = MkScope\ (tds \frown [td])\ [] \\ & & & & in \quad Ok\ (update\_top\_scope\ st\ sc',\ gc) \\ & & & else & Exception[wrongScope] \end{array}$$

HOL Constant

$$\begin{array}{l}
\quad \mathbf{enter_{table}} \\
\quad : \mathit{TRANS_STATE} \\
\quad \rightarrow (\mathit{TableName} \times \mathit{TableInfo} \times \mathit{SsqlCol\ LIST} \times \mathit{TsqlClassName} \times \mathit{TsqlCol\ LIST}) \\
\quad \rightarrow (\mathit{TRANS_STATE} \times \mathit{STRING})\mathit{RESULT} \\
\hline
\forall \mathit{st\ ts\ ti\ scs\ rcn\ tcs} \bullet \\
\quad \mathit{enter}_{\mathit{table}}\ \mathit{st}\ (\mathit{ts},\ \mathit{ti},\ \mathit{scs},\ \mathit{rcn},\ \mathit{tcs}) = \\
\quad \mathit{let} \quad \mathit{sl} = \mathit{symbolTable}(\mathit{st_stack\ st}) \\
\quad \mathit{in} \quad \mathit{if} \quad \mathit{sl} = [] \\
\qquad \mathit{then} \quad \mathit{Exception}[\mathit{noScope}] \\
\qquad \mathit{else} \quad \mathit{let} \quad \mathit{outer} = \mathit{Rev}(\mathit{Tl}(\mathit{Rev\ sl})) \\
\qquad\qquad \mathit{in\ let} \quad t = \mathit{Hd}(\mathit{Rev\ sl}) \\
\qquad\qquad \mathit{in\ let} \quad (\mathit{tds},\ \mathit{ids}) = (\mathit{s_tables}\ t,\ \mathit{s_identifiers}\ t) \\
\qquad\qquad \mathit{in} \quad \mathit{if} \quad \mathit{ids} = [] \\
\qquad\qquad\qquad \mathit{then} \quad \mathit{let} \quad \mathit{gc} = \mathit{unique_name\ st} \\
\qquad\qquad\qquad\qquad \mathit{in\ let} \quad \mathit{td} = \\
\qquad\qquad\qquad\qquad\qquad \mathit{MkTableDetail} \\
\qquad\qquad\qquad\qquad\qquad \mathit{ts}\ (\mathit{c_anon}_s)\ \mathit{gc\ ti\ scs\ rcn} \\
\qquad\qquad\qquad\qquad\qquad \mathit{tcs}\ (\mathit{MkConstraintInfo}[][][][][]) \\
\qquad\qquad\qquad\qquad \mathit{in\ let} \quad \mathit{sc}' = \mathit{MkScope}\ (\mathit{tds} \frown [\mathit{td}])\ [] \\
\qquad\qquad\qquad\qquad \mathit{in} \quad \mathit{Ok}\ (\mathit{update_top_scope\ st\ sc}',\ \mathit{gc}) \\
\qquad\qquad\qquad \mathit{else} \quad \mathit{Exception}[\mathit{wrongScope}]
\end{array}$$

HOL Constant

$$\begin{array}{l}
\quad \mathbf{enter_scope} : \mathit{TRANS_STATE} \rightarrow \mathit{TRANS_STATE} \\
\hline
\forall \mathit{st} \bullet \\
\quad \mathit{enter_scope\ st} = \\
\quad \mathit{let} \quad \mathit{stk} = \mathit{st_stack\ st} \\
\quad \mathit{in\ let} \quad \mathit{symt} = \mathit{symbolTable\ stk} \\
\quad \mathit{in\ let} \quad \mathit{outer} = \mathit{Rev}(\mathit{Tl}(\mathit{Rev\ symt})) \\
\quad \mathit{in\ let} \quad \mathit{symt}' = \mathit{outer} \frown [] \\
\quad \mathit{in} \quad \mathit{MkTRANS_STATE} \\
\qquad (\mathit{MkST_STACK\ symt}'\ (\mathit{parameterTable\ stk})) \\
\qquad (\mathit{query_class\ st})\ (\mathit{query_constants_class\ st})\ (\mathit{client_clearance\ st})
\end{array}$$

The block-structured approach we use means that *leave_scope* is not necessary.

HOL Constant

$$\mathbf{get_{tableinfo}}$$
$$: \mathit{TableSpecification} \rightarrow (\mathit{TableInfo} \times \mathit{ConstraintInfo} \times \mathit{SsqlCol\ LIST} \times \mathit{TsqlClassName} \times \mathit{TsqlCol\ LIST})$$

$$\mathit{true}$$

The function $lookup^{local\_table\_implementation}$ of [6, 11] is not used so we omit it.

HOL Constant

$$\mathbf{lookup_{localtableinfo}}$$
$$: \mathit{TRANS\_STATE} \rightarrow (\mathit{TableInfo\ LIST})\ \mathit{RESULT}$$

$$\forall st \bullet \quad lookup_{localtableinfo}\ st =$$
$$\mathit{let} \quad look\ sc = Map\ td\_info\ (s\_tables\ sc)$$
$$\mathit{in\ let} \quad trs = Fold(\$^\frown)(Map\ look\ (innermost(symbolTable(st\_stack\ st)))) []$$
$$\mathit{in} \quad \mathit{if}\ trs = []$$
$$\mathit{then} \quad Exception[noScope]$$
$$\mathit{else} \quad Ok\ trs$$

HOL Constant

$$\mathbf{lookup_{paramdata}}$$
$$: \mathit{TRANS\_STATE} \rightarrow \mathit{STRING} \rightarrow (\mathit{Value} \times \mathit{Class})\ \mathit{RESULT}$$

$$\forall st\ name \bullet$$
$$lookup_{paramdata}\ st\ name =$$
$$\mathit{let}\ infos = extract_{parameter}(name,\ parameterTable(st\_stack\ st))$$
$$\mathit{in} \quad \mathit{if} \quad infos = []$$
$$\mathit{then} \quad Exception[noSuchParameter]$$
$$\mathit{else\ if} \quad Tl\ infos = []$$
$$\mathit{then} \quad \mathit{let}\ info = Hd\ infos\ \mathit{in}\ Ok(pi\_val_p\ info,\ pi\_clasf\ info)$$
$$\mathit{else} \quad Exception[internalError]$$

HOL Constant

$$\mathbf{in\_new\_scope}$$
$$: (\mathit{TRANS\_STATE} \rightarrow 'a \rightarrow 'b) \rightarrow (\mathit{TRANS\_STATE} \rightarrow 'a \rightarrow 'b)$$

$$\forall what \bullet$$
$$in\_new\_scope\ what = (\lambda st \bullet\ (\lambda a \bullet\ what\ (enter\_scope\ st)\ a))$$

2.4 Transformations Proper

The following glosses over the difference between *ColSpec*s as defined in [4] and the more recent versions of [10].

We also do not spell out the details of various checking functions such as *monop_type* etc.

HOL Constant

repr_col : $TsqlRepr \rightarrow Col\_spec\ RESULT$

$\forall tr \bullet$

$repr\_col\ tr =$
$CASE\ tr\ [$
$WHEN\_local\_identifier\ name \bullet\ \ Ok(denote\_col\_spec\ [name]);$
$WHEN\_column(corr,col) \bullet\ Ok(denote\_col\_spec[corr;col]);$
$WHEN\_constant\_class\ c \bullet\ Exception[internalError];$
$WHEN\_constant\_null\ (Exception[internalError])$
$]$

HOL Constant

all_data_columns$_{\mathbf{local}}$: $TRANS\_STATE \rightarrow Col\_spec\ LIST\ RESULT$

$\forall st \bullet \quad all\_data\_columns_{local}\ st =$
Try
$(ListOk\ o\ Map\ repr\_col)$
$(lookup_{localcolspecsterlings}\ st)$

HOL Constant

binop_type : $Op \times SwordType \times SwordType \rightarrow SwordType\ RESULT$

$true$

HOL Constant

monop_type : $Op \times SwordType \rightarrow SwordType\ RESULT$

$true$

HOL Constant

triop_type : $Op \times SwordType \times SwordType \times SwordType \rightarrow SwordType\ RESULT$

$true$

HOL Constant

set_func_type : $Op \times SwordType \to SwordType\ RESULT$

$true$

HOL Constant

check_boolean : $ExpType \to ExpType\ RESULT$

$true$

HOL Constant

check_type_conversion : $SwordType \times SwordType \to ONE\ RESULT$

$true$

The following interprets the old-style SSQL column specification as formed from a hierarchical table directory name followed by a table name followed by a column name.

HOL Constant

convert$_{\mathbf{colspec}}$: $Col\_spec \to ColumnSpecification$

$\forall il \bullet \quad convert_{colspec}\ (denote\_col\_spec\ il) =$
$\quad let \quad col = Hd(Rev\ il)$
$\quad in\ let \quad dir = Hd(Tl(Rev\ il))$
$\quad in\ let \quad tab = Tl(Tl(Rev\ il))$
$\quad in \quad mk\_specific(mk\_absolute(tab, dir), col)$

HOL Constant

class_column : $TRANS\_STATE \to Col\_spec \to (Col\_spec + Class)\ RESULT$

$\forall st\ cs \bullet$
$\quad class\_column\ st\ cs =$
$\quad let \quad csc = lookup_{colspecclass}\ st\ (true, convert_{colspec}\ cs)$
$\quad in \quad if \quad isVal\ csc$
$\quad\quad then \quad let \quad (tr, lub\_cl) = destVal\ csc$
$\quad\quad\quad in \quad CASE\ tr\ [$
$\quad\quad\quad\quad WHEN\_local\_identifier\ name \bullet$
$\quad\quad\quad\quad\quad Ok(InL(denote\_col\_spec\ [name]));$
$\quad\quad\quad\quad WHEN\_column\ (gen\_corr, gen\_col) \bullet$
$\quad\quad\quad\quad\quad Ok(InL(denote\_col\_spec([gen\_corr; gen\_col])));$
$\quad\quad\quad\quad WHEN\_constant\_class\ cl \bullet\ Ok(InR\ cl);$
$\quad\quad\quad\quad WHEN\_constant\_null\ (Exception[internalError])$
$\quad\quad\quad\quad]$
$\quad\quad else \quad giveError(destError\ csc)$

SML

HOL Constant

denote_name : *TsqlRepr* → *Value*

```
∀tr•  denote_name tr =
      CASE tr [
      WHEN_local_identifier s• contents(denote_col_spec [s]);
      WHEN_column(cn,col)• contents(denote_col_spec [cn;col]);
      WHEN_constant_class c• denote_class c;
      WHEN_constant_null denote_null
      ]
```

HOL Constant

column_data_test : *TRANS_STATE* → *Col_spec* → *Value LIST RESULT*

```
∀st cs•
      column_data_test st cs =
      let   csc = lookup_colspecclass st (true, convert_colspec cs)
      in    if    isVal csc
            then  let   (tr, u) = destVal csc
                  in    if    client_clearance st dom u
                        then  Ok[]
                        else  let cc = denote_class(client_clearance st)
                              in    Ok[binop"dom"(cc, denote_name tr)]
            else  giveError(destError csc)
```

SML

HOL Constant

col_exp : *ExpType* → *ColType RESULT*

```
∀t w•
      col_exp (t, w) =
      if w = dinary then Ok(c_nullType, t)
      else if w = sterling then Ok(t, c_nullType)
      else if w = worthless then Ok(t, c_nullType)
      else Exception[wrongType]
```

SML

HOL Constant

col_target : *SsqlCol* → *TsqlCol RESULT*

∀ *sc*•
col_target sc =
let *bound bi* =
CASE bi [
WHEN_upb c• c_anon_{tc};
WHEN_constant c•$mk\_constant_{tc}$ *c*
]
in let *target* ((*s*, *d*), *c*) =
if *s* = *c_nullType* ∧ *d* = *c_nullType*
then *Exception*[*internalError*]
else if *d* = *c_nullType*
then *Ok*(*MkTsqlCol* c_anon_t c_none_t *c*)
else if *s* = *c_nullType*
then *Ok*(*MkTsqlCol* c_none_t c_anon_t *c*)
else *Ok*(*MkTsqlCol* c_anon_t c_anon_t *c*)
in *target*(*sc_type_field sc*, *bound*(*sc_col_class sc*))

The following interprets the old-style SSQL table specification as formed from a hierarchical table directory name followed by a table name followed by a column name.

HOL Constant

$\mathbf{convert_{tablespec}}$: *Table_spec* → *TableSpecification*

∀*il*• $convert_{tablespec}$ (*denote_table_spec il*) =
let *tab* = *Hd*(*Rev il*)
in let *dir* = *Tl*(*Rev il*)
in *mk_absolute*(*dir*, *tab*)

HOL Constant

$\mathbf{constant\_value_{type}}$: *Value* → *SwordType*

---

true

HOL Constant

$\mathbf{convert_{ssqltype}}$: *Type* → *SwordType*

---

true

HOL Constant

convert_tableSpecification_backup
: $STRING\ LIST \times \mathbb{N} \rightarrow STRING\ LIST\ RESULT$

$\forall d\ dirs\ n \bullet$
$convert\_tableSpecification\_backup\ ([],\ 0) = Ok\ []$
$\wedge\ convert\_tableSpecification\_backup\ (Cons\ d\ dirs,\ 0) = Ok\ (Cons\ d\ dirs)$
$\wedge\ convert\_tableSpecification\_backup\ ([],\ n+1) = Exception[noSuchDirectory]$
$\wedge\ convert\_tableSpecification\_backup\ (Cons\ d\ dirs,\ n+1) =$
$convert\_tableSpecification\_backup\ (dirs,\ n)$

HOL Constant

convert$_{\mathbf{tableSpecifcation}}$
: $TableSpecification \rightarrow Table\_spec\ RESULT$

$\forall ts \bullet\ convert_{tableSpecifcation}\ ts =$
$CASE\ ts\ [$
$WHEN\_absolute(directory,table) \bullet$
$\quad Ok(denote\_table\_spec(directory \frown [table]));$
$WHEN\_default(up,directory,table) \bullet$
$\quad let\ \ dir = convert\_tableSpecification\_backup$
$\quad\quad\quad\quad (default\_directory,\ up)$
$\quad in\ \ if\ \ isError\ dir$
$\quad\quad then\ \ giveError(destError\ dir)$
$\quad\quad else\ \ Ok(denote\_table\_spec(destVal\ dir \frown directory \frown [table]))$
$]$

HOL Constant

table_name
: *TableSpecification* → *STRING*

$\forall ts \bullet$ *table_name ts* =
let *dot s* = *s* ⌢ "."
in *CASE ts* [
WHEN_absolute(*dir*,*tab*)•
if *dir* = []
then *tab*
else *Fold* (\$⌢) (*Map dot dir*) [] ⌢ *tab*;
WHEN_default(*up*,*dir*,*tab*)•
if *dir* = []
then *Flat*(*seq up* "−") ⌢ *tab*
else *Flat*(*seq up* "−") ⌢ *Fold* (\$⌢) (*Map dot dir*) [] ⌢ *tab*
]

HOL Constant

$\mathbf{convert_{swordtype}}$: *SwordType* → *Type*

---

true

HOL Constant

$\mathbf{convert_{type}}$: *Type* → *Type*

$\forall t \bullet$ $convert_{type}$ *t* =
$convert_{swordtype}(convert_{ssqltype}\ t)$

HOL Constant

$\mathbf{denote_{classexp}}$: *ExpClass* → *Value*

$\forall ec \bullet$
$denote_{classexp}$ *ec* =
CASE ec [
WHEN_variable(*v*,*c*)• *v*;
$WHEN\_constant_{ec}$ *c*• *denote_class c*
]

HOL Constant

$$\mathbf{lub_{boundinfo}} : BoundInfo \times BoundInfo \to BoundInfo$$

$\forall bi1\ bi2 \bullet$
$lub_{boundinfo}\ (bi1,\ bi2) =$
$CASE\ bi1\ [$
$WHEN\_upb\ c1 \bullet$
$CASE\ bi2\ [$
$WHEN\_upb\ c2 \bullet mk\_upb(c1\ lub\ c2);$
$WHEN\_constant\ c2 \bullet\ mk\_upb(c1\ lub\ c2)$
$];$
$WHEN\_constant\ c1 \bullet$
$CASE\ bi2\ [$
$WHEN\_upb\ c2 \bullet mk\_upb(c1\ lub\ c2);$
$WHEN\_constant\ c2 \bullet\ if\ c1 = c2\ then\ bi1\ else\ mk\_upb(c1\ lub\ c2)$
$]$
$]$

HOL Constant

$$\mathbf{lub_{expclass}} : ExpClass \times ExpClass \to ExpClass$$

$\forall ec1\ ec2 \bullet$
$lub_{expclass}\ (ec1,\ ec2) =$
$CASE\ ec1\ [$
$WHEN\_variable\ (v1,\ c1) \bullet$
$CASE\ ec2\ [$
$WHEN\_variable\ (v2,\ c2) \bullet$
$mk\_variable(binop\ "lub"\ (v1,\ v2),\ c1\ lub\ c2);$
$WHEN\_constant_{ec}\ c2 \bullet$
$mk\_variable(binop\ "lub"\ (v1,\ denote\_class\ c2),\ c1\ lub\ c2)$
$];$
$WHEN\_constant_{ec}\ c1 \bullet$
$CASE\ ec2\ [$
$WHEN\_variable\ (v2,\ c2) \bullet$
$mk\_variable(binop\ "lub"\ (v2,\ denote\_class\ c1),\ c1\ lub\ c2);$
$WHEN\_constant_{ec}\ c2 \bullet$
$mk\_constant_{ec}\ (c1\ lub\ c2)$
$]$
$]$

SML

HOL Constant

$$\mathbf{lub_{type}} : SwordType \times SwordType \rightarrow SwordType$$

$$true$$

HOL Constant

$$\mathbf{lub_{coltype}} : ColType \times ColType \rightarrow ColType$$

$$\forall\ s1\ d1\ s2\ d2 \bullet$$
$$lub_{coltype}((s1,\ d1),\ (s2,\ d2)) =$$
$$(lub_{type}(s1,s2),lub_{type}(d1,d2))$$

HOL Constant

$$\mathbf{lub_{worth}} : Worth \times Worth \rightarrow Worth$$

$$\forall\ w1\ w2 \bullet$$
$$lub_{worth}\ (w1,\ w2) =$$
$$if\ w1 = w2\ then \quad w1$$
$$else\ if\ w2 = worthless\ then\ w1$$
$$else\ if\ w1 = worthless\ then\ w2$$
$$else\ priceless$$

HOL Constant

$$\mathbf{lub_{exp}} : ExpType \times ExpType \rightarrow ExpType$$

$$\forall\ t1\ w1\ t2\ w2 \bullet$$
$$lub_{exp}((t1,\ w1),\ (t2,\ w2)) =$$
$$(lub_{type}(t1,t2),lub_{worth}(w1,w2))$$

HOL Constant

$$\mathbf{lub_{ssqlname}} : SsqlName \times SsqlName \rightarrow SsqlName$$

$$\forall\ sn1\ sn2 \bullet$$
$$lub_{ssqlname}\ (sn1,\ sn2) =$$
$$CASE\ sn1\ [$$
$$WHEN\_name_s\ s1 \bullet$$
$$\quad CASE\ sn2\ [$$
$$\quad\quad WHEN\_name_s\ s2 \bullet\ if\ s1 = s2\ then\ sn1\ else\ c\_anon_s;$$
$$\quad\quad OTHERS\ c\_anon_s$$
$$\quad];$$
$$OTHERS\ c\_anon_s$$
$$]$$

HOL Constant

$$\mathbf{lub_{ssqlcol}} : SsqlCol \times SsqlCol \to SsqlCol$$

$\forall\ n1\ t1\ ce1\ cc1\ n2\ t2\ ce2\ cc2 \bullet$
$lub_{ssqlcol}\ (MkSsqlCol\ n1\ t1\ ce1\ cc1,\ MkSsqlCol\ n2\ t2\ ce2\ cc2) =$
$MkSsqlCol$
$(lub_{ssqlname}(n1,n2))\ (lub_{coltype}(t1,t2))$
$(ce1\ lub\ ce2)\ (lub_{boundinfo}(cc1,cc2))$

HOL Constant

$$\mathbf{lub_{tableinfo}} : TableInfo \times TableInfo \to TableInfo$$

$\forall\ tec1\ tc1\ rc1\ tec2\ tc2\ rc2 \bullet$
$lub_{tableinfo}\ (MkTableInfo\ tec1\ tc1\ rc1,\ MkTableInfo\ tec2\ tc2\ rc2) =$
$MkTableInfo$
$(tec1\ lub\ tec2)\ (tc1\ lub\ tc2)\ (lub_{boundinfo}(rc1,rc2))$

HOL Constant

$$\mathbf{lub_{tsqlclassname}} : TsqlClassName \times TsqlClassName \to TsqlClassName$$

$\forall\ tcn1\ tcn2 \bullet$
$lub_{tsqlclassname}\ (tcn1,\ tcn2) =$
$CASE\ tcn1\ [$
$WHEN\_name_{tc}\ s1 \bullet$
$\quad CASE\ tcn2\ [$
$\quad\quad WHEN\_name_{tc}\ s2 \bullet\ if\ s1 = s2\ then\ tcn1\ else\ c\_anon_{tc};$
$\quad\quad OTHERS\ c\_anon_{tc}$
$\quad];$
$OTHERS\ c\_anon_{tc}$
$]$

HOL Constant

$$\mathbf{lub_{tsqlname}} : TsqlName \times TsqlName \rightarrow TsqlName$$

$\forall\ tn1\ tn2 \bullet$

$lub_{tsqlname}\ (tn1,\ tn2) =$

$CASE\ tn1\ [$

$WHEN\_name_t\ s1 \bullet$

$CASE\ tn2\ [$

$WHEN\_name_t\ s2 \bullet\ if\ s1 = s2\ then\ tn1\ else\ c\_anon_t;$

$OTHERS\ c\_anon_t$

$];$

$WHEN\_none_t$

$(CASE\ tn2\ [$

$WHEN\_none_t\ c\_none_t;$

$OTHERS\ c\_anon_t$

$]);$

$OTHERS\ c\_anon_t$

$]$

HOL Constant

$$\mathbf{lub_{tsqlcol}} : TsqlCol \times TsqlCol \rightarrow TsqlCol$$

$\forall\ s1\ d1\ c1\ s2\ d2\ c2 \bullet$

$lub_{tsqlcol}\ (MkTsqlCol\ s1\ d1\ c1,\ MkTsqlCol\ s2\ d2\ c2) =$

$MkTsqlCol$

$(lub_{tsqlname}(s1,\ s2))$

$(lub_{tsqlname}(d1,\ d2))$

$(lub_{tsqlclassname}(c1,\ c2))$

The new *Select_value* category corresponds (at least in its *anonynomous* option with the old *Value*, which appears as the operand to *select_value*).

HOL Constant

$$\mathbf{make_{sv}} : TsqlCol \times TsqlCol \to Value\ LIST\ RESULT$$

$$\begin{array}{l}
\forall f\ t\bullet\ make_{sv}\ (f,\ t) = \\
\quad let\quad data_col\ (ftn,\ ttn) = CASE\ ftn\ [\\
\qquad\qquad WHEN_none_t \\
\qquad\qquad\qquad (CASE\ ttn\ [\\
\qquad\qquad\qquad WHEN_none_t\ (Ok\ []); \\
\qquad\qquad\qquad WHEN_anon_t\ (Ok[denote_null]); \\
\qquad\qquad\qquad WHEN_name_t\ ts\bullet\ Exception[internalError]]); \\
\qquad\qquad WHEN_anon_t \\
\qquad\qquad\qquad (CASE\ ttn\ [\\
\qquad\qquad\qquad WHEN_none_t\ (Ok[denote_null]); \\
\qquad\qquad\qquad OTHERS\ (Exception[internalError])]); \\
\qquad\qquad WHEN_name_t\ fs\bullet \\
\qquad\qquad\qquad (CASE\ ttn\ [\\
\qquad\qquad\qquad WHEN_anon_t\ (Ok[contents(denote_col_spec\ [fs])]); \\
\qquad\qquad\qquad WHEN_name_t\ ts\bullet\ Ok[contents(denote_col_spec\ [fs])]; \\
\qquad\qquad\qquad WHEN_none_t\ (Exception[internalError])])] \\
\quad in\ let\ class_col\ (fcn,\ tcn) = CASE\ fcn\ [\\
\qquad\qquad WHEN_constant_{tc}\ fc\bullet \\
\qquad\qquad\qquad CASE\ tcn\ [\\
\qquad\qquad\qquad WHEN_constant_{tc}\ tc\bullet \\
\qquad\qquad\qquad\qquad if\ fc = tc \\
\qquad\qquad\qquad\qquad then\ Ok\ [] \\
\qquad\qquad\qquad\qquad else\ Exception[internalError]; \\
\qquad\qquad\qquad WHEN_name_{tc}\ tn\bullet\ Ok[denote_class\ fc]; \\
\qquad\qquad\qquad WHEN_anon_{tc}\ (Exception[internalError])]; \\
\qquad\qquad WHEN_name_{tc}\ f\bullet \\
\qquad\qquad\qquad CASE\ tcn\ [\\
\qquad\qquad\qquad WHEN_constant_{tc}\ tc\bullet Exception[internalError]; \\
\qquad\qquad\qquad WHEN_name_{tc}\ tn\bullet\ Ok[contents(denote_col_spec\ [f])]; \\
\qquad\qquad\qquad WHEN_anon_{tc}\ (Ok[contents(denote_col_spec\ [f])])]; \\
\qquad\qquad WHEN_anon_{tc}\ (Exception[internalError])] \\
\quad in\ let\ \ ssv = data_col(tc_sterling_name\ f,\ tc_sterling_name\ t) \\
\quad in\ let\ \ dsv = data_col(tc_dinary_name\ f,\ tc_dinary_name\ t) \\
\quad in\ let\ \ csv = class_col(tc_class_name\ f,\ tc_class_name\ t) \\
\quad in\quad if\quad isError\ ssv \vee isError\ dsv \vee isError\ csv \\
\qquad\quad then\quad Exception[internalError] \\
\qquad\quad else\quad Ok(destVal\ ssv \frown destVal\ dsv \frown destVal\ csv)
\end{array}$$

HOL Constant

remove_constants : $(Col\_spec + Class)\ LIST \to Col\_spec\ LIST$

$\forall x\ s \bullet$
$\quad remove\_constants\ [] = []$
$\wedge \quad remove\_constants\ (Cons\ x\ s) =$
$\quad if\ IsL\ x$
$\quad then\ Cons\ (OutL\ x)\ (remove\_constants\ s)$
$\quad else\ remove\_constants\ s$

HOL Constant

remove_nulls : $TsqlRepr\ LIST \to TsqlRepr\ LIST$

$\forall x\ trs \bullet$
$\quad remove\_nulls\ [] = []$
$\wedge \quad remove\_nulls\ (Cons\ x\ trs) =$
$\quad CASE\ x\ [$
$\quad WHEN\_constant\_null\ (remove\_nulls\ trs);$
$\quad OTHERS\ (Cons\ x\ (remove\_nulls\ trs))]$

HOL Constant

upper : $ExpClass \to Class$

$\forall ec \bullet \quad upper\ ec =$
$\quad CASE\ ec\ [$
$\quad WHEN\_variable\ (c,\ u) \bullet\ u;$
$\quad WHEN\_constant_{ec}\ u \bullet\ u]$

HOL Constant

make_case : $Value \times ExpClass \to Value$

$\forall data\ ec \bullet$
$\quad make\_case\ (data,\ ec) =$
$\quad CASE\ ec\ [$
$\quad WHEN\_variable\ (c,\ u) \bullet\ case\ [data]\ [denote\_class(lattice\_top)]\ c;$
$\quad WHEN\_constant_{ec}\ c \bullet\ case\ [data]\ [denote\_class(lattice\_top)]\ (denote\_class\ c)]$

HOL Constant

simplify$_{\mathbf{ands}}$: *Value LIST* × *ExpClass LIST* → *Value* × *ExpClass*

∀*vs cs*•
simplify$_{ands}$ (*vs*, *cs*) =
let *v* = *fold* (*binop And*) *vs*
in let *c* = *case* [*v*]
[*fold*(*binop* "*lub*")(*Map denote*$_{classexp}$ *cs*)]
(*fold*(*binop* "*glb*")(*at2* (*Map make_case*) (*vs*,*cs*)))
in let *u* = *fold* (*Uncurry* $*lub*) (*Map upper cs*)
in (*v*, *mk_variable*(*c*, *u*))

HOL Constant

simplify$_{\mathbf{ors}}$: *Value LIST* × *ExpClass LIST* → *Value* × *ExpClass*

∀*vs cs*•
simplify$_{ors}$ (*vs*, *cs*) =
let *v* = *fold* (*binop Or*) *vs*
in let *c* = *case* [*v*]
[*fold*(*binop* "*glb*")(*Map denote*$_{classexp}$ *cs*)]
(*fold*(*binop* "*lub*")(*at2* (*Map make_case*) (*vs*,*cs*)))
in let *u* = *fold* (*Uncurry* $*lub*) (*Map upper cs*)
in (*v*, *mk_variable*(*c*, *u*))

HOL Constant

constant_value$_{\mathbf{data}}$: *Value* → *Value*

---

true

HOL Constant

dinary_columns : *TRANS_STATE* → *Col_spec LIST* → *Col_spec LIST RESULT*

∀*st css*•
dinary_columns st css =
let *look cs* = *lookup*$_{colspecdinary}$ *st* (*false*,*convert*$_{colspec}$ *cs*)
in *Try*
(*ListOk o Map repr_col o remove_nulls*)
(*ListOk* (*Map look css*))

HOL Constant

sterling_columns : *TRANS_STATE* → *Col_spec LIST* → *Col_spec LIST RESULT*

∀*st css*•
sterling_columns st css =
let look cs = $lookup_{colspecsterling}$ *st* (*false*,$convert_{colspec}$ *cs*)
in Try
(*ListOk o Map repr_col o remove_nulls*)
(*ListOk* (*Map look css*))

HOL Constant

tuple_list$_{\mathbf{maxrowclass}}$: *TRANS_STATE* → *Tuple_list* → *ExpClass*

∀*st t*• $tuple\_list_{maxrowclass}$ *st t* =
$mk\_constant_{ec}$(*client_clearance st*)

HOL Constant

upb_row_class : *TableInfo* → *Class*

∀*tec tc rowc*•
upb_row_class (*MkTableInfo tec tc rowc*) =
CASE rowc [
WHEN_upb rc• *rc*;
WHEN_constant rc• *rc*]

In the following, the specifications of $internal\_value_{class}$ and *value_type* are incomplete. (*all_binop*, *some_binop*, *all_binop_list* and *some_binop_list* are missing from $internal\_value_{class}$ and *caseVal* is missing from *value_type*.)

HOL Constant

value$_{\mathbf{data}}$: *TRANS_STATE* → *Value* → *Value*

---

true

HOL Constant

value$_{\mathbf{type}}$: *TRANS_STATE* → *Value* → *ExpType*

---

true

We now have to start breaking up the mutually recursive functions of [6]. We do this by passing the functions to be recalled recursively as parameters. For example:

HOL Constant

$$\mathbf{value_{class}}$$
$$: (TRANS\_STATE \rightarrow Value \rightarrow InternalExpClass\ RESULT)$$
$$\rightarrow TRANS\_STATE \rightarrow Value \rightarrow ExpClass\ RESULT$$

$$\forall st\ ivc\ v \bullet$$
$$value_{class}\ ivc\ st\ v =$$
$$let \quad x = ivc\ st\ v$$
$$in \quad if\ isError\ x$$
$$then\ giveError\ (destError\ x)$$
$$else \quad Ok(CASE\ (destVal\ x)\ [$$
$$WHEN\_ands(datas,\ classes) \bullet$$
$$let \quad (v,\ c) = simplify_{ands}(datas,classes)$$
$$in \quad c;$$
$$WHEN\_ors(datas,classes) \bullet$$
$$let \quad (v,\ c) = simplify_{ors}(datas,classes)$$
$$in\ c;$$
$$WHEN\_simple\ ec \bullet$$
$$CASE\ ec\ [$$
$$WHEN\_variable(exp,up) \bullet\ mk\_variable(exp,up);$$
$$WHEN\_constant_{ec}\ c \bullet\ mk\_constant_{ec}\ c]$$
$$])$$

HOL Constant

$$\mathbf{tuple\_list_{data}} : TRANS\_STATE \rightarrow Tuple\_list \rightarrow Tuple\_list\ RESULT$$

$$true$$

HOL Constant

$$\mathbf{tuple\_list_{type}} : TRANS\_STATE \rightarrow Tuple\_list \rightarrow Tuple\_list\ RESULT$$

$$true$$

HOL Constant

$$\mathbf{from\_spec_{enter}} : TRANS\_STATE \rightarrow From\_spec \rightarrow From\_spec\ RESULT$$

$$true$$

HOL Constant

$$\mathbf{select\_list_{type}} : TRANS\_STATE \rightarrow Select\_list \rightarrow ExpType\ RESULT$$

$$true$$

HOL Constant

select_value$_{\mathbf{type}}$: *TRANS_STATE* → *Value* → *ExpType RESULT*

---

true

HOL Constant

tuple_list$_{\mathbf{info}}$: *TRANS_STATE* → *Tuple_list*
→ (*TableName* × *TableInfo* × *SsqlCol LIST*) *RESULT*

---

true

HOL Constant

tuple_list$_{\mathbf{make}}$
: *TRANS_STATE* → *Tuple_list* × *TsqlClassName* × *TsqlCol LIST*
→ *Tuple_list RESULT*

---

true

HOL Constant

from_spec$_{\mathbf{info}}$
: *TRANS_STATE* → *From_spec*
→ (*TableInfo* × *SsqlCol LIST* × *TsqlClassName* × *TsqlCol LIST*) *RESULT*

---

true

HOL Constant

select_list$_{\mathbf{info}}$
: *TRANS_STATE* → *Select_list* → *SsqlCol LIST RESULT*

---

true

HOL Constant

select_value$_{\mathbf{info}}$
: *TRANS_STATE* → *Value* → *SsqlCol RESULT*

---

true

HOL Constant

value$_{\mathbf{info}}$
: *TRANS_STATE* → *Value* → *SsqlCol RESULT*

---

true

HOL Constant

$$\mathbf{internal\_value_{class}} : \mathit{TRANS\_STATE} \rightarrow \mathit{Value} \rightarrow \mathit{InternalExpClass\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{tuple\_list_{class}} : \mathit{TRANS\_STATE} \rightarrow \mathit{Tuple\_list} \rightarrow \mathit{Tuple\_list}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{select\_list_{class}} : \mathit{TRANS\_STATE} \rightarrow \mathit{Select\_list} \rightarrow \mathit{Select\_list\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{select\_value_{class}} : \mathit{TRANS\_STATE} \rightarrow \mathit{Value} \rightarrow \mathit{Value\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{select\_list_{make}} : \mathit{TRANS\_STATE} \rightarrow (\mathit{Select\_list} \times \mathit{TsqlClassName} \times \mathit{TsqlCol\ LIST}) \rightarrow \mathit{Value\ LIST\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{select\_value_{make}} : \mathit{TRANS\_STATE} \rightarrow (\mathit{Value} \times \mathit{TsqlCol\ LIST}) \rightarrow \mathit{Value\ LIST\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{make\_col} : \mathit{TRANS\_STATE} \rightarrow (\mathit{Value} \times \mathit{TsqlCol}) \rightarrow \mathit{Value\ LIST\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{make_{dinary}} : \mathit{TRANS\_STATE} \rightarrow (\mathit{Value} \times \mathit{ExpType}) \rightarrow \mathit{Value\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{make_{sterling}} : \mathit{TRANS\_STATE} \rightarrow (\mathit{Value} \times \mathit{ExpType}) \rightarrow \mathit{Value\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{select\_list_{data}} : \mathit{TRANS\_STATE} \rightarrow \mathit{Select\_list} \rightarrow \mathit{Select\_list\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{select\_value_{data}} : \mathit{TRANS\_STATE} \rightarrow \mathit{Value} \rightarrow \mathit{Value\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{tuple\_list_{makeouter}} : \mathit{TRANS\_STATE} \rightarrow (\mathit{Tuple\_list} \times \mathit{BOOL} \times \mathit{TsqlClassName} \times \mathit{TsqlCol\ LIST}) \rightarrow (\mathit{Tuple\_list} \times \mathit{Query\ LIST})\ \mathit{RESULT}$$

$$\mathit{true}$$

That is the last of the mutually recursive functions.

HOL Constant

$$\mathbf{tuple\_list_{outerinfo}} : \mathit{TRANS\_STATE} \rightarrow \mathit{Tuple\_list} \rightarrow \mathit{BOOL\ RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{transform_{selectquery}} : \mathit{TRANS\_STATE} \rightarrow \mathit{Query} \rightarrow (\mathit{SsqlCol\ LIST} \times \mathit{Query} \times \mathit{BOOL} \times \mathit{TsqlClassName} \times \mathit{TsqlCol\ LIST} \times \mathit{Query\ LIST})\ \mathit{RESULT}$$

$$\mathit{true}$$

HOL Constant

$$\mathbf{query_{selectquery}} : \mathit{TRANS\_STATE} \rightarrow \mathit{Query} \rightarrow \mathit{Query\ RESULT}$$

$$\mathit{true}$$

3 INTERFACE

We now wish to define the SSQL Transformation Processor as required by [7]. It is parameterised by the as yet undefined processing of the various updating queries.

The local function *dynamic* below computes the list of boolean flags needed by the output filter of [7] from the list of *TsqlCol*s produced by $transform_{selectquery}$.

Since the processing of updating queries is as yet unspecified, we treat it as an error if the select query processing fails (e.g, because the query is not a select query).

SML

HOL Constant

STP : (*Query*, *FILTER_PARS*) *STP_TYPE*

```
∀q c•  STP (q, c) =
       let      st = init_trans_state c
       in let   res = transform_selectquery st q
       in       if       isError res
                then     giveError(destError res)
                else     Ok
                         let      (scs, tq, scw, rc, tcs, chks) = destVal res
                         in let   dynamic tc =
                                  CASE tc [WHEN_constant_tc c• false; OTHERS true]
                         in let   cc = Map (dynamic o tc_class_name) tcs
                         in let   cr = if dynamic rc then InL c else InR One
                         in let   cq = if chks = [] then InR One else InL(Hd chks)
                         in       (tq, cq, (scw, cr, cc))
```

4 INDEX