Project: DRA FRONT END FILTER PROJECT

Title: Representation of an SSQL State as a TSQL State

Ref: DS/FMU/FEF/025 *Issue: Revision* : *2.1* *Date:* 5 June 2016

Status: Approved *Type:* Specification

Keywords:

Author:

Name	*Location*	*Signature*	*Date*
G. M. Prout	WIN01		

Authorisation for Issue:

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

Abstract: The formal specification of a mapping from an SSQL abstract machine state to the TSQL state which represents it. This is 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] DS/FMU/FEF/003. *Formal Security Policy.* G.M. Prout, ICL Secure Systems, WIN01.

[2] DS/FMU/FEF/006. *Security Conjecture for the SSQL Abstract Machine.* G.M. Prout, ICL Secure Systems, WIN01.

[3] DS/FMU/FEF/021. *Specification of TSQL.* G.M. Prout, ICL Secure Systems, WIN01.

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

0.3 Changes History

Issue 1.1 (18 February 1993) First draft.

Issue 1.4 (23 February 1993) $*$ now has correct fixity.

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

Issue 2.2 Removed dependency on ICL logo font

0.4 Changes Forecast

Changes may be necessary as a result of issues raised during Phase 2.

1 GENERAL

1.1 Scope

This document gives a formal specification of a mapping from an SSQL abstract machine state to the TSQL state which represents it.

1.2 Introduction

In [4] a description of the SWORD architectural model is given. The top level structure of the Front End implementation of SWORD is captured in its definition as the combination of appropriate subsystems to make a behavioural model in the sense of the formal security policy of [1]. This model requires the availability of a function which maps any state of the SSQL abstract machine of [2] to the TSQL state which represents it.

In [3] we defined the semantics of TSQL as a subset of SSQL with all classes in the state, except those that are actually stored as data, set at the lowest possible classification, *lattice_bottom*, and all worths set at *sterling*. Given an SSQL state with n columns, this will be represented as a TSQL state with between n and $3n + 1$ columns. The first column in the TSQL state will contain a value corresponding to the row existence class in the SSQL state, unless this is constant(i.e. when the maximum row existence class is the same as the table class). Then for each SSQL column there will be one, two or three TSQL columns. There will always be a column corresponding to the sterling value of the SSQL value. The second column corresponds to the dinary value of the SSQL value, except in the case where all values in this column would be null whence it is omitted. The next column corresponds to the classification of the SSQL value, except in the case where the maximum and minimum field classes for a column are the same whence, again, it is omitted. If the SSQL value is of type *NullItem* then the TSQL sterling and dinary values will be *null*. Otherwise, if the SSQL value is dinary then the TSQL sterling value will be *null* and the TSQL dinary value will be the SSQL value and if the the SSQL value is sterling then the TSQL sterling value will be the SSQL value and the TSQL dinary value will be *null*.

2 PRELIMINARIES

The following ProofPower instructions set up the new theory *fef025* and set the context for the proof tools.

```
SML
open_theory "fef021";
(force_delete_theory "fef025" handle _ => ());
new_theory"fef025";
push_pc "hol";
```

3 REPRESENTING AN SSQL STATE AS A TSQL STATE

First a function which takes a value and returns sterling data classified at bottom.

HOL Constant

$$\mathbf{MkData_t} : Val \rightarrow Data$$

$$\forall\ v \bullet MkData_t\ v = newData\ lattice\_bottom\ sterling\ v$$

$MkNullData_t$ is a sterling null value classified at bottom.

HOL Constant

$$\mathbf{MkNullData_t} : Data$$

$$MkNullData_t = MkData\ lattice\_bottom\ (NullItemItem\ null)$$

A function which makes a sterling value classified at bottom from a class.

HOL Constant

$$\mathbf{MkClassVal} : Class \rightarrow Data$$

$$\forall\ c : Class \bullet MkClassVal\ c = MkData_t(ClassVal\ c)$$

The function *repr_data* takes a value and returns the sterling value, the dinary value and the class value.

HOL Constant

$$\mathbf{repr\_data} : Data \rightarrow Data \times Data \times Data$$

$$\begin{array}{l}
\forall\ d : Data \bullet \\
repr_data\ d = \\
\quad let\ i = Dat_item\ d \\
\quad in\ let\ c = Dat_class\ d \\
\quad in \\
\quad if\ isNullItem\ i \\
\quad then\ (MkNullData_t, MkNullData_t, MkClassVal\ c) \\
\quad else \\
\quad let\ v = destValuedItem\ i \\
\quad in\ if\ VI_worth\ v = sterling \\
\quad then\ (MkData_t\ (VI_val\ v), MkNullData_t, MkClassVal\ c) \\
\quad else\ if\ VI_worth\ v = dinary \\
\quad then\ (MkNullData_t, MkData_t\ (VI_val\ v), MkClassVal\ c) \\
\quad else\ \ (MkNullData_t, MkNullData_t, MkClassVal\ c)
\end{array}$$

All column existence classes in the state will be at bottom.

HOL Constant

$$\mathbf{Repr\_colCon} : ColCon \rightarrow ColCon$$

$$\begin{aligned}
&\forall\ cc : ColCon \bullet \\
&Repr_colCon\ cc = MkColCon \\
&\qquad lattice_bottom \\
&\qquad (CC_uniform\ cc) \\
&\qquad (CC_unique\ cc) \\
&\qquad (CC_classLimited\ cc) \\
&\qquad (CC_primary\ cc) \\
&\qquad (CC_secondary\ cc) \\
&\qquad (CC_referential\ cc)
\end{aligned}$$

We need functions to generate new names for the new columns in the TSQL state.

HOL Constant

$$\begin{aligned}
&\mathbf{RowExistName} : Ide \rightarrow Ide; \\
&\mathbf{SterlingName} : Ide \rightarrow Ide; \\
&\mathbf{DinaryName} : Ide \rightarrow Ide; \\
&\mathbf{ClassName} : Ide \rightarrow Ide
\end{aligned}$$

$$\begin{aligned}
&OneOne\ RowExistName \\
\wedge\ &OneOne\ SterlingName \\
\wedge\ &OneOne\ DinaryName \\
\wedge\ &OneOne\ ClassName \\
\wedge\ &\forall\ f\ g\ x \bullet \{f;g\} \subseteq \{RowExistName;SterlingName;DinaryName;ClassName\} \\
&\qquad \wedge\ f\ x = g\ x \\
&\qquad \Rightarrow f = g
\end{aligned}$$

A new *ColSpec* for the row_existence column of a table may be required. In TSQL, all values will be sterling, so the type of dinary values in TSQL should be null. Since there is no null type in SSQL, we use type *monolean* to model the null type.

HOL Constant

$$\mathbf{rowExistCol} : \mathit{Ide} \rightarrow \mathit{ColSpec}$$

$$\begin{aligned}
&\forall\ \mathit{tab} : \mathit{Ide} \bullet \\
&\mathit{rowExistCol}\ \mathit{tab} = \mathit{MkColSpec} \\
&\qquad (\mathit{RowExistName}\ \mathit{tab}) \\
&\qquad \mathit{1} \\
&\qquad \mathit{monoleanType} \\
&\qquad \mathit{classType} \\
&\qquad \mathit{false} \\
&\qquad \mathit{MkNullData}_t \\
&\qquad \mathit{1} \\
&\qquad \mathit{lattice_bottom} \\
&\qquad \mathit{lattice_bottom}
\end{aligned}$$

A new set of constraints for the row_existence column, if required.

HOL Constant

$$\mathbf{rowExistColCon} : \mathit{ColCon}$$

$$\begin{aligned}
&\mathit{rowExistColCon} = \mathit{MkColCon} \\
&\qquad \mathit{lattice_bottom} \\
&\qquad \mathit{false} \\
&\qquad \mathit{false} \\
&\qquad \mathit{false} \\
&\qquad \mathit{false} \\
&\qquad \mathit{false} \\
&\qquad []
\end{aligned}$$

The function f_1 takes a table and a column position and if that column exists in the table returns its *ColSpec*.

HOL Constant

$$\mathbf{f_1} : \mathit{TableSpec} \rightarrow \mathit{Num} \rightarrow \mathit{ColSpec} + \mathit{ONE}$$

$$\begin{aligned}
&\forall\ t\ n \bullet f_1\ t\ n = \quad \mathit{if}\ \exists_1\ c \bullet c \in \mathit{TS_colspecs}\ t \wedge \mathit{CS_posn}\ c = n \\
&\qquad\qquad \mathit{then}\ \mathit{InL}\ (\epsilon\ c \bullet c \in \mathit{TS_colspecs}\ t \wedge \mathit{CS_posn}\ c = n) \\
&\qquad\qquad \mathit{else}\ \mathit{InR}\ \mathit{One}
\end{aligned}$$

ColNeeds determines the number of TSQL columns needed for a particular SSQL column by checking the relevant *ColSpec* components.

HOL Constant

$$\textbf{ColNeeds} : \mathit{TableSpec} \to \mathit{Num} \to \mathit{Num} + \mathit{ONE}$$

$$\begin{aligned}
&\forall\ t\ n \bullet \mathit{ColNeeds}\ \ t\ n = \\
&\quad \mathit{if}\ \mathit{IsL}\ (f_1\ t\ n)\ \mathit{then} \\
&\qquad \mathit{let}\ c = \mathit{OutL}\ (f_1\ t\ n) \\
&\qquad \mathit{in\ let}\ n' = \quad \mathit{if}\ \mathit{CS_dinaryType}\ c = \mathit{monoleanType} \\
&\qquad\qquad\qquad\qquad\ \ \mathit{then}\ 1 \\
&\qquad\qquad\qquad\qquad\ \ \mathit{else}\ 2 \\
&\qquad \mathit{in\ let}\ \mathit{ans} = \quad \mathit{if}\ \mathit{CS_min}\ c = \mathit{CS_max}\ c \\
&\qquad\qquad\qquad\qquad\ \ \mathit{then}\ n' \\
&\qquad\qquad\qquad\qquad\ \ \mathit{else}\ n' + 1 \\
&\qquad \mathit{in}\ \mathit{InL}\ \mathit{ans} \\
&\quad \mathit{else}\ \mathit{InR}(\mathit{OutR}(f_1\ t\ n))
\end{aligned}$$

Given an SSQL column position we need the first column position for the TSQL columns representing it.

HOL Constant

$$\textbf{NextNum} : \mathit{TableSpec} \to \mathit{Num} \to \mathit{Num}$$

$$\begin{aligned}
&\forall\ t\ n \bullet \mathit{NextNum}\ t\ n = (1 + \#\{i | i < n \wedge \mathit{IsL}(\mathit{ColNeeds}\ t\ i) \\
&\qquad\qquad \wedge\ \mathit{OutL}(\mathit{ColNeeds}\ t\ i) = 3\} * 3 \\
&\qquad\qquad + \#\{i | i < n \wedge \mathit{IsL}(\mathit{ColNeeds}\ t\ i) \\
&\qquad\qquad \wedge\ \mathit{OutL}(\mathit{ColNeeds}\ t\ i) = 2\} * 2 \\
&\qquad\qquad + \#\{i | i < n \wedge \mathit{IsL}(\mathit{ColNeeds}\ t\ i) \\
&\qquad\qquad \wedge\ \mathit{OutL}(\mathit{ColNeeds}\ t\ i) = 1\})
\end{aligned}$$

From each SSQL *ColSpec*, generate a list of TSQL *ColSpec*s. This list will always contain a sterling *ColSpec*, and may contain a dinary *ColSpec* and a class *ColSpec*. If an extra *ColSpec* is required for the row existence classifications, then we increment each column position by one.

HOL Constant

$$\mathbf{f_2} : \mathit{TableSpec} \to \mathit{ColSpec} \to \mathit{ColSpec\ LIST}$$

$$\begin{array}{l}
\forall\ t\ cs \bullet f_2\ t\ cs = \\
\quad \mathit{let}\ n = \mathit{CS_posn}\ cs \\
\quad \mathit{in\ let}\ rc = \mathit{if}\ \mathit{TS_class}\ t = \mathit{TS_maxRow}\ t\ \mathit{then}\ 0\ \mathit{else}\ 1 \\
\quad \mathit{in\ let}\ next = rc + \mathit{NextNum}\ t\ n \\
\quad \mathit{in\ let}\ cs_s = \mathit{MkColSpec} \\
\qquad (\mathit{SterlingName}\ (\mathit{CS_ide}\ cs)) \\
\qquad next \\
\qquad \mathit{monoleanType} \\
\qquad (\mathit{CS_sterlingType}\ cs) \\
\qquad (\mathit{CS_nullType}\ cs) \\
\qquad (\mathit{Fst}(\mathit{repr_data}(\mathit{CS_default}\ cs))) \\
\qquad (rc + \mathit{CS_consGroup}\ cs) \\
\qquad \mathit{lattice_bottom} \\
\qquad \mathit{lattice_bottom} \\
\quad \mathit{in\ let}\ (l_1, next_1) = \\
\qquad \mathit{if}\ \mathit{CS_dinaryType}\ cs = \mathit{monoleanType} \\
\qquad \mathit{then}\ ([cs_s], next + 1) \\
\qquad \mathit{else}\ (\mathit{Cons}\ cs_s[\mathit{MkColSpec} \\
\qquad\quad (\mathit{DinaryName}\ (\mathit{CS_ide}\ cs)) \\
\qquad\quad (next + 1) \\
\qquad\quad \mathit{monoleanType} \\
\qquad\quad (\mathit{CS_dinaryType}\ cs) \\
\qquad\quad (\mathit{CS_nullType}\ cs) \\
\qquad\quad (\mathit{Fst}(\mathit{Snd}(\mathit{repr_data}(\mathit{CS_default}\ cs)))) \\
\qquad\quad (rc + \mathit{CS_consGroup}\ cs) \\
\qquad\quad \mathit{lattice_bottom} \\
\qquad\quad \mathit{lattice_bottom}], next + 2) \\
\quad \mathit{in\ if}\ \mathit{CS_min}\ \ cs = \mathit{CS_max}\ cs \\
\qquad \mathit{then}\ l_1 \\
\qquad \mathit{else}\ l_1 \frown [\mathit{MkColSpec} \\
\qquad\quad (\mathit{ClassName}\ (\mathit{CS_ide}\ cs)) \\
\qquad\quad next_1 \\
\qquad\quad \mathit{monoleanType} \\
\qquad\quad \mathit{classType} \\
\qquad\quad (\mathit{CS_nullType}\ cs) \\
\qquad\quad (\mathit{Snd}(\mathit{Snd}(\mathit{repr_data}(\mathit{CS_default}\ cs)))) \\
\qquad\quad (rc + \mathit{CS_consGroup}\ cs) \\
\qquad\quad \mathit{lattice_bottom} \\
\qquad\quad \mathit{lattice_bottom}]
\end{array}$$

From each SSQL column position/data pair we generate a list of column position/data pairs containing one, two or three elements.

HOL Constant

$$\mathbf{f_3} : TableSpec \to Num \times Data \to (Num \times Data)\ LIST$$

$$\begin{aligned}
&\forall\ t\ n\ d \bullet f_3\ t\ (n,d) = \\
&\quad let\ rc = if\ TS_class\ t = TS_maxRow\ t\ then\ 0\ else\ 1 \\
&\quad in\ let\ next = rc + NextNum\ t\ n \\
&\quad in\ let\ cs = OutL(f_1\ t\ n) \\
&\quad in\ let\ d_s = Fst(repr_data\ d) \\
&\quad in\ let\ (l_1,next_1) = \\
&\qquad if\ CS_dinaryType\ cs = monoleanType \\
&\qquad then\ ([(next,d_s)],next + 1) \\
&\qquad else\ (Cons\ (next,d_s)[(next + 1,Fst(Snd\ (repr_data\ d)))],next + 2) \\
&\quad in\ if\ CS_min\ \ cs = CS_max\ cs \\
&\qquad then\ l_1 \\
&\qquad else\ l_1 \frown\ [(next_1,Snd(Snd\ (repr_data\ d)))]
\end{aligned}$$

Represent a row of SSQL data in a table as a row of TSQL data.

HOL Constant

$$\mathbf{repr\_row} : TableSpec \to Row \to Row$$

$$\begin{aligned}
&\forall\ t\ r \bullet repr_row\ t\ r = MkRow \\
&\qquad lattice_bottom \\
&\qquad (let\ prs = R_data\ r \\
&\qquad in\ let\ prs' = \{pr' | \exists\ pr \bullet pr \in prs \wedge pr' \in Elems(f_3\ t\ pr)\} \\
&\qquad in\ if\ TS_class\ t = TS_maxRow\ t \\
&\qquad then\ prs' \\
&\qquad else\ prs' \cup \{(1,MkClassVal\ (R_exist\ r))\})
\end{aligned}$$

Represent a table of SSQL data as a row of TSQL data.

HOL Constant

repr_table : $Ide \rightarrow TableSpec \rightarrow TableSpec$

$$\forall\ i\ t \bullet repr\_table\ i\ t =$$
$$let\ css = \{cs' | \exists\ cs \bullet cs \in TS\_colspecs\ t \wedge cs' \in Elems(f_2\ t\ cs)\}$$
$$in\ let\ ccs = \{(n,cc) | n \in Dom(TS\_cons\ t) \wedge$$
$$cc = Repr\_colCon\ ((TS\_cons\ t)\ @\ n)\}$$
$$in\ let\ (css',ccs') =$$
$$if\ TS\_class\ t = TS\_maxRow\ t$$
$$then\ (css,ccs)$$
$$else\ (css \cup \{rowExistCol\ i\},$$
$$\{1,rowExistColCon\} \cup$$
$$\{(n',cc') | \exists\ n\ cc \bullet (n,cc) \in ccs \wedge n' = n + 1 \wedge cc' = cc\})$$
$$in\ MkTableSpec$$
$$lattice\_bottom$$
$$lattice\_bottom$$
$$css'$$
$$ccs'$$
$$(Map\ (repr\_row\ t)\ (TS\_rows\ t))$$

Represent an SSQL directory data as a TSQL directory.

HOL Constant

repr_dir : $Directory \rightarrow Directory$

$$\forall\ d \bullet repr\_dir\ d =$$
$$MkDirectory$$
$$\{(id,tab) | id \in Dom(Dir\_tables\ d)\ \wedge\ tab = repr\_table\ id\ (Dir\_tables\ d\ @\ id)\}$$
$$lattice\_bottom$$
$$lattice\_bottom$$

Finally *repr* from an SSQL state to a TSQL state.

HOL Constant

reprState : $State \rightarrow State_t$

$$\forall\ s \bullet reprState\ s =$$
$$let\ s' = repState\ s$$
$$in\ absState_t(absState\{(i,dir) | i \in Dom\ s' \wedge dir = repr\_dir(s'\ @\ i)\})$$

4 INDEX