Project: DRA FRONT END FILTER PROJECT

Title: Specification of SSQL Semantics I

Ref: DS/FMU/FEF/004 *Issue: Revision* : *4.2* *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 first part of the formal specifications of the main functionality of the SSQL semantics 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/017. *Secure Database Technical Proposal.* High Assurance Team, ICL Secure Systems, WIN01, 21st January 1992.

[2] DS/FMU/FEF/005. *Specifications of hide and updateState.* G.M. Prout, ICL Secure Systems, WIN01.

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

[4] *Invitation to Tender RSRE 1c/6130.* DRA, December 1991.

0.3 Changes History

Issue *Revision* : *4.2* **(5 June 2016)** Latest approved version.

Issue 4.3 Removed dependency on ICL logo font

0.4 Changes Forecast

None.

1 GENERAL

1.1 Scope

This document gives a formal specification of part of the SSQL semantics. Together with [3] it constitutes deliverable D3 of work package 1a, as given in section 7 of the Secure Database Technical Proposal, [1].

1.2 Introduction

In the Secure Database Technical Proposal, [1], we stated that we propose to formalise the semantics of SSQL, supplied initially in Annex 2, *SSQL Abstract Machine*, to the ITT [4], in a manner suitable for the required security proof, as outlined in [1]. This involves the specific features of the semantics which ensure that query processing is *secure* be clearly separated from the purely functional aspects. This results in the formalisation of the semantics of SSQL being presented in three main parts:

1. A function *hide* on the state of the database which returns a sanitised state of the database. This version of the database does not include any information the existence of which the user is not cleared to know, nor does it contain the true value of any field which the user is not cleared to see.
2. The main functionality of SSQL, i.e. how the result of a query is computed from the state of the database.
3. A function *updateState* which applies the update determined in (2) to the database.This function is responsible for capturing the security constraints associated with updates.

This document provides the specification of enough of the functionality of the semantics of SSQL to enable the state of the abstract machine to be defined. The remainder of the main functionality of SSQL is specified in [3]. The functions *hide* and *updateState* are formalised in [2].

1.3 Terminology

Labelled products with names of the form $Xxxx$ will have labels prefixed by $X\_$; labelled products with names of the form $XxxxYyyy$ will have labels prefixed by $XY\_$. The only exceptions to this will be in the case where two labelled products would have identical labels, e.g. *Data* and $DirectoryT$ will have labels prefixed by $Dat\_$ and $Dir\_$ respectively. Constructors for labelled products are automatically generated by the ProofPower system; names of constructors for labelled products take the form of the name of the labelled product prefixed by Mk.

2 PRELIMINARIES

The following ProofPower instructions set up the new theory *fef004* and set the context for the proof tools.

SML

```
open_theory "fef003";
(force_delete_theory "fef004" handle _ => ());
new_theory"fef004";
push_pc "hol";
```

3 DATA TYPES IN THE STATE

In the following, as well as types we have introduced some constants, which are sets. This is in order to define invariants on the state of the database. Typically, these invariants are that certain relations are partial functions, and the partial function operator $\pfun$ is defined on sets. Where possible, the same name has been used for the type and the constant; otherwise the name of the set is the name of the type suffixed with S.

The types *CHAR*, *STRING*, $\mathbb{N}$ and *BOOL* are already defined in the system.

SML

```
declare_type_abbrev("Char",[],⌜: CHAR⌝);
```

SML

```
declare_type_abbrev("String",[],⌜: STRING⌝);
```

SML

```
declare_type_abbrev("Num",[],⌜: ℕ⌝);
```

The constant *Num* is the set of everything of type *Num*.

HOL Constant

$$\mathbf{Num} : Num\ \mathbb{P}$$

$$Num = Universe$$

SML

```
declare_type_abbrev("Bool",[],⌜: BOOL⌝);
```

SML

```
declare_type_abbrev("Ide",[],⌜: String⌝);
```

The constant *Ide* is the set of everything of type *Ide*.

HOL Constant

$$\mathbf{Ide} : Ide\ \mathbb{P}$$

$$Ide = Universe$$

HOL Labelled Product

Int

I_mantissa	: *Num*;
I_exponent	: *Num*

HOL Labelled Product

Float

F_mantissa	: *Num*;
F_exponent	: *Num*

We introduce new types *Time*, *Interval* and *Code*.

SML

```
new_type("Time",0);
```

SML

```
new_type("Interval",0);
```

SML

```
new_type("Code",0);
```

The different 'sorts' of errors are modelled as natural numbers.

HOL Constant

error notCleared tooWide tooTall wrongType
nullValue noSuchColumn ambiguousColumn
ambiguousUpdate mayNotBeComplete
underClassified downGrade classChange
noSuchTable noSuchDirectory
ambiguousEvaluate ambiguousHaving
nonUniformValues nonUniqueValues
noNulls fieldClassOutOfRange
rowClassTooLow accessDenied : *Num*

$$
\begin{array}{lll}
 & error & = 1 \\
\wedge & notCleared & = 2 \\
\wedge & tooWide & = 3 \\
\wedge & tooTall & = 4 \\
\wedge & wrongType & = 5 \\
\wedge & nullValue & = 6 \\
\wedge & noSuchColumn & = 7 \\
\wedge & ambiguousColumn & = 8 \\
\wedge & ambiguousUpdate & = 9 \\
\wedge & mayNotBeComplete & = 10 \\
\wedge & underClassified & = 11 \\
\wedge & downGrade & = 12 \\
\wedge & classChange & = 13 \\
\wedge & noSuchTable & = 14 \\
\wedge & noSuchDirectory & = 15 \\
\wedge & ambiguousEvaluate & = 16 \\
\wedge & ambiguousHaving & = 17 \\
\wedge & nonUniformValues & = 18 \\
\wedge & nonUniqueValues & = 19 \\
\wedge & noNulls & = 20 \\
\wedge & fieldClassOutOfRange & = 21 \\
\wedge & rowClassTooLow & = 22 \\
\wedge & accessDenied & = 23
\end{array}
$$

SML

```
declare_type_abbrev("Error",[],⌜: Num⌝);
```

The different 'sorts' of worth are also modelled as natural numbers.

HOL Constant

worthless dinary sterling priceless : *Num*

---

worthless = *1*
∧ *dinary* = *2*
∧ *sterling* = *3*
∧ *priceless* = *4*

SML

declare_type_abbrev("*Worth*",[],⌜: *Num*⌝);

Val is declared as a sum type (i.e. binary disjoint union). The 'types' *void* and *exception* are given as one-point types.

SML

declare_type_abbrev("*Val*",[],
⌜: *ONE* + *Bool* + *String* + *Int* + *Float* + *Time* + *Interval* + *Code* + *Class* + *ONE*⌝);

We give constructors for type *Val*.

HOL Constant

VoidVal : *Val*;
BoolVal : *Bool* → *Val*;
StringVal : *String* → *Val*;
IntVal : *Int* → *Val*;
FloatVal : *Float* → *Val*;
TimeVal : *Time* → *Val*;
IntervalVal : *Interval* → *Val*;
CodeVal : *Code* → *Val*;
ClassVal : *Class* → *Val*;
ExceptionVal : *Val*

∀ *b*:*Bool*; *s*:*String*; *i*:*Int*; *f*:*Float*; *t*:*Time*; *n*:*Interval*; *c*:*Code*; *cl*:*Class*
• *VoidVal* = *InL One*
∧ *BoolVal b* = *InR* (*InL b*)
∧ *StringVal s* = *InR* (*InR* (*InL s*))
∧ *IntVal i* = *InR* (*InR* (*InR* (*InL i*)))
∧ *FloatVal f* = *InR* (*InR* (*InR* (*InR* (*InL f*))))
∧ *TimeVal t* = *InR* (*InR* (*InR* (*InR* (*InR* (*InL t*)))))
∧ *IntervalVal n* = *InR* (*InR* (*InR* (*InR* (*InR* (*InR* (*InL n*))))))
∧ *CodeVal c* = *InR* (*InR* (*InR* (*InR* (*InR* (*InR* (*InR* (*InL c*)))))))
∧ *ClassVal cl* = *InR* (*InR* (*InR* (*InR* (*InR* (*InR* (*InR* (*InR* (*InL cl*))))))))
∧ *ExceptionVal* = *InR* (*InR* (*InR* (*InR* (*InR* (*InR* (*InR* (*InR One*)))))))

HOL Labelled Product

ValuedItem

VI_worth	: *Num*;
VI_val	: *Val*

The type *NullItem* is given as a one-point type.

SML

declare_type_abbrev("*NullItem*",[],⌜: *ONE*⌝);
declare_alias("*null*",⌜*One*⌝);

Item is declared as a sum type.

SML

declare_type_abbrev("*Item*",[],⌜: *ValuedItem* + *NullItem*⌝);

We give constructors for type *Item*.

HOL Constant

ValuedItemItem	: *ValuedItem* → *Item*;
NullItemItem	: *NullItem* → *Item*

∀ *v*:*ValuedItem*; *n*:*NullItem*
• *ValuedItemItem v* = *InL v*
∧ *NullItemItem n* = *InR n*

HOL Labelled Product

Data

Dat_class	: *Class*;
Dat_item	: *Item*

The constant *DataS* is the set of everything of type *Data*.

HOL Constant

DataS	: *Data* ℙ

DataS = *Universe*

SML

declare_type_abbrev("*Update*",[],⌜: *Item* + *Class* + *Data*⌝);

We need constructor, discriminator and destructor functions for *Update*.

HOL Constant

$$\begin{array}{l} \textbf{ItemUpdate} \quad : \mathit{Item} \rightarrow \mathit{Update}; \\ \textbf{ClassUpdate} \quad : \mathit{Class} \rightarrow \mathit{Update}; \\ \textbf{DataUpdate} \quad : \mathit{Data} \rightarrow \mathit{Update} \\ \hline \forall\ i{:}\mathit{Item};\ c{:}\mathit{Class};\ d{:}\mathit{Data} \\ \bullet \quad \mathit{ItemUpdate}\ i\ = \mathit{InL}\ i \\ \wedge \quad \mathit{ClassUpdate}\ c = \mathit{InR}\ (\mathit{InL}\ c) \\ \wedge \quad \mathit{DataUpdate}\ d = \mathit{InR}\ (\mathit{InR}\ d) \end{array}$$

HOL Constant

$$\begin{array}{l} \textbf{isItem} \quad : \mathit{Update} \rightarrow \mathit{Bool}; \\ \textbf{isClass} \quad : \mathit{Update} \rightarrow \mathit{Bool}; \\ \textbf{isData} \quad : \mathit{Update} \rightarrow \mathit{Bool} \\ \hline \forall \quad u{:}\mathit{Update} \\ \bullet \quad (\mathit{isItem}\ u \quad = \exists\ i{:}\mathit{Item} \quad \bullet\ u = \mathit{ItemUpdate}\ i) \\ \wedge \quad (\mathit{isClass}\ u \quad = \exists\ c{:}\mathit{Class} \quad \bullet\ u = \mathit{ClassUpdate}\ c) \\ \wedge \quad (\mathit{isData}\ u \quad = \exists\ d{:}\mathit{Data} \quad \bullet\ u = \mathit{DataUpdate}\ d) \end{array}$$

HOL Constant

$$\begin{array}{l} \textbf{destItem} \quad : \mathit{Update} \rightarrow \mathit{Item}; \\ \textbf{destClass} \quad : \mathit{Update} \rightarrow \mathit{Class}; \\ \textbf{destData} \quad : \mathit{Update} \rightarrow \mathit{Data} \\ \hline \forall \quad i{:}\mathit{Item};\ c{:}\mathit{Class};\ d{:}\mathit{Data} \\ \bullet \quad \mathit{destItem}\ (\mathit{ItemUpdate}\ i) \quad = i \\ \wedge \quad \mathit{destClass}(\mathit{ClassUpdate}\ c) \quad = c \\ \wedge \quad \mathit{destData}\ (\mathit{DataUpdate}\ d) \quad = d \end{array}$$

We give *monolean*, *boolean*, *chars*, *integer*, *floating*, *time*, *interval* and *class* as elements of one-point types.

SML

```
declare_alias("monolean",⌜One⌝);
declare_alias("boolean",⌜One⌝);
declare_alias("chars",⌜One⌝);
declare_alias("integer",⌜One⌝);
declare_alias("floating",⌜One⌝);
declare_alias("time",⌜One⌝);
declare_alias("interval",⌜One⌝);
declare_alias("class",⌜One⌝);
```

SML

```
declare_type_abbrev("Type",[],
        ⌜: ONE + ONE + ONE + ONE + ONE + ONE + ONE + ONE⌝);
```

SML

```
declare_type_abbrev("Errors",[],⌜: Error LIST⌝);
```

HOL Labelled Product

ColSpec

CS_ide	: *Ide*;
CS_posn	: *Num*;
CS_dinaryType	: *Type*;
CS_sterlingType	: *Type*;
CS_nullType	: *Bool*;
CS_default	: *Data*;
CS_consGroup	: *Num*;
CS_min	: *Class*;
CS_max	: *Class*

We have removed *Error* from the state of the data base and will provide access functions which return either a data type or an error.

SML

```
declare_type_abbrev("Tab",[],⌜: Ide LIST⌝);
```

HOL Labelled Product

Reference

R_table	: *Tab*;
R_group	: *Num*

HOL Labelled Product

ColCon

CC_exist	: *Class*;
CC_uniform	: *Bool*;
CC_unique	: *Bool*;
CC_classLimited	: *Bool*;
CC_primary	: *Bool*;
CC_secondary	: *Bool*;
CC_referential	: *Reference LIST*

The constant *ColConS* is the set of everything of type *ColCon*.

HOL Constant

$$\textbf{ColConS} : ColCon\ \mathbb{P}$$

$$ColConS = Universe$$

HOL Labelled Product

Row

$$\textbf{R\_exist} : Class;$$
$$\textbf{R\_data} : Num \leftrightarrow Data$$

The constant *RowS* is the set of everything of type *Row* where the data in the row is a partial function.

HOL Constant

$$\textbf{RowS} : Row\ \mathbb{P}$$

$$RowS = \{r \mid let\ data = R\_data\ r\ in\ \ data \in Num \nrightarrow DataS\}$$

HOL Labelled Product

TableSpec

$$\textbf{TS\_class} : Class;$$
$$\textbf{TS\_maxRow} : Class;$$
$$\textbf{TS\_colspecs} : ColSpec\ \mathbb{P};$$
$$\textbf{TS\_cons} : Num \leftrightarrow ColCon;$$
$$\textbf{TS\_rows} : Row\ LIST$$

The constant *TableSpecS* is the set of everything of type *TableSpec* where the column constraints is a partial function and the rows are members of *Rows*.

HOL Constant

$$\textbf{TableSpecS} : TableSpec\ \mathbb{P}$$

$$TableSpecS = \{ts \mid let\ \ cns = TS\_cons\ ts\ and\ rows = TS\_rows\ ts$$
$$in\ \ cns \in Num \nrightarrow ColConS \land \forall\ r \bullet r \in_l rows \Rightarrow r \in RowS\}$$

HOL Labelled Product

Directory

$$\textbf{Dir\_tables} : Ide \leftrightarrow TableSpec;$$
$$\textbf{Dir\_exist} : Class;$$
$$\textbf{Dir\_class} : Class$$

The constant *DirectoryS* is the set of everything of type *Directory* where the tables are a partial function from *Ide* to *TableSpecS*.

HOL Constant

$$\textbf{DirectoryS} : Directory\ \mathbb{P}$$

$$DirectoryS = \{d \mid Dir\_tables\ d \in Ide \pfun TableSpecS\}$$

The constant *IdeL* is the set of everything of type list of *Ide*.

HOL Constant

$$\textbf{IdeL} : (Ide\ LIST)\mathbb{P}$$

$$IdeL = Universe$$

The representation type for *State* : *Exp* is *StateR*.

SML

```
declare_type_abbrev("StateR",[],⌜: Ide LIST ↔ Directory⌝);
```

The constant *StateS* is the set of everything of type *StateR* which is a partial function from *IdeL* to *DirectoryS*.

HOL Constant

$$\textbf{StateS} : StateR\ \mathbb{P}$$

$$StateS = IdeL \pfun DirectoryS$$

We define the property required on the representation state.

HOL Constant

$$\textbf{isState} : StateR \to Bool$$

$$\forall\ s \bullet isState\ s \Leftrightarrow s \in StateS$$

We demonstrate that the new type will be non-empty.

SML

```
push_goal([],⌜∃ s : StateR • isState s⌝);
a(rewrite_tac[get_spec⌜isState⌝]);
a(∃_tac⌜{}⌝);
a(rewrite_tac[get_spec⌜StateS⌝,↦_def,functional_def,∩_def,↔_def]);
```

Now the new type *State* : *Exp* is defined.

SML

```
val state_type_def = new_type_defn(["state_type_def"],"State",[],pop_thm());
```

We have chosen not to model *Result*. A *Result* is either some data or just a classification (when the user is not cleared to observe the actual value). In effect, we always have something of type *Data*; in the case when the user is not cleared to see the value, the value will be *Hidden* anyway.

4 FUNCTIONALITY OF THE QUERY PROCESSING

4.1 Auxiliary Functions

General constructor, discriminator and destructor functions for a sum type where one component is an error or sequence of errors.

HOL Constant

$$
\begin{array}{ll}
\textbf{giveVal} & : 'a \to 'a + 'Error; \\
\textbf{giveError} & : 'Error \to 'a + 'Error; \\
\textbf{destVal} & : 'a + 'Error \to 'a; \\
\textbf{destError} & : 'a + 'Error \to 'Error; \\
\textbf{isVal} & : 'a + 'Error \to Bool; \\
\textbf{isError} & : 'a + 'Error \to Bool
\end{array}
$$

$$
\begin{array}{lll}
\forall\ v;\ e;\ ve & & \\
\bullet \quad giveVal\ v & & = InL\ v \\
\wedge \quad giveError\ e & & = InR\ e \\
\wedge \quad destVal\ (giveVal\ v) & & = v \\
\wedge \quad destError(giveError\ e) & & = e \\
\wedge \quad (isVal\ ve & & = \exists\ v_1 \bullet ve = giveVal\ v_1) \\
\wedge \quad (isError\ ve & & = \exists\ e_1 \bullet ve = giveError\ e_1)
\end{array}
$$

4.2 Type of *Effect*

The functionality of the semantics of SSQL will be captured by the function *processQuery* ,described in section 4.3. This function takes a query, a user's clearance and a state and returns the result of the query, i.e., the output to be seen by the user, an update to be applied to the data base and some error messages. The effect of a query depends on the type of the query. Insertions, deletions and updates either all succeed or all fail. i.e. any *mayNotbeComplete* on inner selects result in no change to the data base.

The 'effect' of an insert query is a sequence of relations between column number and data , together with a full table name. Each element in the sequence will give a new row whose existence classification is the same as the user's clearance. Missing columns (i.e. columns that were hidden from the user) will be supplied with the default data for that column.

SML

$$declare\_type\_abbrev(\texttt{"}Insert\texttt{"},[],\ulcorner: Tab \times (Num \leftrightarrow Data)\ LIST\urcorner);$$

The 'effect' of a delete query is a full table name and a set of non-negative integers corresponding to which rows in the table's sequence of rows are to be deleted. Note, these row positions will be relative to the hidden state of the database that the user sees, not its actual state.

SML

$$declare\_type\_abbrev(\texttt{"}Delete\texttt{"},[],\ulcorner: Tab \times Num\ \mathbb{P}\urcorner);$$

The 'effect' of an update query is a full table name and a relation from row number to a relation from column number to actual update. Again, the row positions will be relative to the hidden state of the database that the user sees, not its actual state. (We use the identifier *UpDate* because *Update* has already been used.)

SML

$$declare\_type\_abbrev(\texttt{"}UpDate\texttt{"},[],\ulcorner: Tab \times (Num \leftrightarrow (Num \leftrightarrow Update))\urcorner);$$

The 'effect' of a select query is some data to be returned to the user (in fact, a sequence of sequence of *Data*).

SML

$$declare\_type\_abbrev(\texttt{"}Select\texttt{"},[],\ulcorner: Data\ LIST\ LIST\urcorner);$$

This gives us the sum type *Effect*.

SML

$$declare\_type\_abbrev(\texttt{"}Effect\texttt{"},[],\ulcorner: Insert + Delete + UpDate + Select\urcorner);$$

We give constructor, discriminator and destructor functions for type *Effect*.

HOL Constant

$$\begin{array}{l} \quad \textbf{InsertEffect} \quad : Insert \to Effect; \\ \quad \textbf{DeleteEffect} \quad : Delete \to Effect; \\ \quad \textbf{UpdateEffect} \quad : UpDate \to Effect; \\ \quad \textbf{SelectEffect} \quad : Select \to Effect \\ \hline \quad \forall\ i{:}Insert;\ d{:}Delete;\ u{:}UpDate;\ s{:}Select \\ \quad \bullet \quad InsertEffect\ i = InL\ i \\ \quad \wedge \quad DeleteEffect\ d = InR\ (InL\ d) \\ \quad \wedge \quad UpdateEffect\ u = InR\ (InR\ (InL\ u)) \\ \quad \wedge \quad SelectEffect\ s = InR\ (InR\ (InR\ s)) \end{array}$$

HOL Constant

$$\begin{array}{l} \quad \textbf{isInsert} \quad : Effect \to Bool; \\ \quad \textbf{isDelete} \quad : Effect \to Bool; \\ \quad \textbf{isUpdate} \quad : Effect \to Bool; \\ \quad \textbf{isSelect} \quad : Effect \to Bool \\ \hline \quad \forall\ e : Effect \\ \quad \bullet \quad (isInsert\ e = \exists\ i{:}Insert \bullet e = InsertEffect\ i) \\ \quad \wedge \quad (isDelete\ e = \exists\ d{:}Delete \bullet e = DeleteEffect\ d) \\ \quad \wedge \quad (isUpdate\ e = \exists\ u{:}UpDate \bullet e = UpdateEffect\ u) \\ \quad \wedge \quad (isSelect\ e = \exists\ s{:}Select \bullet e = SelectEffect\ s) \end{array}$$

HOL Constant

$$\begin{array}{l} \textbf{destInsert} : \mathit{Effect} \rightarrow \mathit{Insert}; \\ \textbf{destDelete} : \mathit{Effect} \rightarrow \mathit{Delete}; \\ \textbf{destUpdate} : \mathit{Effect} \rightarrow \mathit{UpDate}; \\ \textbf{destSelect} : \mathit{Effect} \rightarrow \mathit{Select} \\ \hline \forall\ i{:}\mathit{Insert};\ d{:}\mathit{Delete};\ u{:}\mathit{UpDate};\ s{:}\mathit{Select} \\ \bullet\ \mathit{destInsert}(\mathit{InsertEffect}\ i) = i \\ \wedge\ \mathit{destDelete}(\mathit{DeleteEffect}\ d) = d \\ \wedge\ \mathit{destUpdate}(\mathit{UpdateEffect}\ u) = u \\ \wedge\ \mathit{destSelect}(\mathit{SelectEffect}\ s) = s \end{array}$$

We also provide a function that return the table name from an effect; this function will not be applied to a *Select* effect.

HOL Constant

$$\begin{array}{l} \textbf{tabFromEffect} : \mathit{Effect} \rightarrow \mathit{Tab} \\ \hline \forall\ i{:}\mathit{Insert};\ d{:}\mathit{Delete};\ u{:}\mathit{UpDate} \\ \bullet\ \mathit{tabFromEffect}\ (\mathit{InsertEffect}\ i) = \mathit{Fst}\ i \\ \wedge\ \mathit{tabFromEffect}\ (\mathit{DeleteEffect}\ d) = \mathit{Fst}\ d \\ \wedge\ \mathit{tabFromEffect}\ (\mathit{UpdateEffect}\ u) = \mathit{Fst}\ u \end{array}$$

4.3 The function *processQuery*

The functionality of SSQL is modelled with *processQuery*, specified in [3].

5 AUXILIARY PROOF WORK

For convenience, we provide some consistency proofs will be required by later proof documents.

5.1 Consistency Proof for *dominates*

We prove the consistency of *dominates*, *lattice_bottom*, *lattice_top* and *lub*.

SML

```
push_consistency_goal⌜$dominates⌝;
a(cases_tac⌜∀ top bot:Class • top = bot⌝);
(* *** Goal "1" *** *)
```

This is the case where *Class* has only one element:

```
SML
a(∃_tac⌜(λ x y • T),top,top,(λ x y • x),(λ x y • x)⌝);
a(asm_rewrite_tac[]);
(* *** Goal "2" *** *)
```

In this case, *top* and *bot* are given to be distinct elements of *Class*, we take the so-called flat lattice with *top* as top, *bot* as bottom and everything else incomparable (and in between).

```
SML
a(∃_tac⌜(
        (λ c1 c2 • c1 = top ∨ c2 = bot ∨ c1 = c2),
        bot,
        top,
        (λ c1 c2 •    if      c1 = top
                      then    top
                      else if c1 = bot
                      then    c2
                      else if c2 = c1 ∨ c2 = bot
                      then    c1
                      else    top),
        (λ c1 c2 •    if      c1 = bot
                      then    bot
                      else if c1 = top
                      then    c2
                      else if c2 = c1 ∨ c2 = top
                      then    c1
                      else    bot))

⌝);
a(rewrite_tac[] THEN REPEAT strip_tac
        THEN_TRY asm_rewrite_tac[]
        THEN_TRY SOLVED_T (PC_T1 "prop_eq" asm_prove_tac[])
        THEN_TRY all_var_elim_asm_tac);
```

This produces 11 subgoals but the proof is fairly mechanical:

SML

```
(* *** Goal "2.1" *** *)
a(all_asm_ante_tac THEN cases_tac⌜x = top⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜x = bot⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜y = x⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜y = bot⌝ THEN asm_rewrite_tac[]);
(* *** Goal "2.2" *** *)
a(all_asm_ante_tac THEN cases_tac⌜x = top⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜x = bot⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜y = x⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜y = bot⌝ THEN asm_rewrite_tac[]);
(* *** Goal "2.3" *** *)
a(GET_ASM_T ⌜¬ top = bot⌝ (strip_asm_tac o conv_rule (ONCE_MAP_C eq_sym_conv)));
a(DROP_NTH_ASM_T 2 ante_tac THEN asm_rewrite_tac[]);
(* *** Goal "2.4" *** *)
a(GET_ASM_T ⌜¬ top = bot⌝ (strip_asm_tac o conv_rule (ONCE_MAP_C eq_sym_conv)));
a(DROP_NTH_ASM_T 2 ante_tac THEN asm_rewrite_tac[]);
(* *** Goal "2.5" *** *)
a(cases_tac⌜x = top⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜x = bot⌝ THEN asm_rewrite_tac[]);
(* *** Goal "2.6" *** *)
a(cases_tac⌜y = top⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜y = bot⌝ THEN asm_rewrite_tac[]);
(* *** Goal "2.7" *** *)
a(POP_ASM_T ante_tac THEN asm_rewrite_tac[]);
a(cases_tac⌜x = bot⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜y = x⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜y = top⌝ THEN asm_rewrite_tac[]);
(* *** Goal "2.8" *** *)
a(POP_ASM_T ante_tac THEN asm_rewrite_tac[]);
a(cases_tac⌜x = bot⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜x = top⌝ THEN asm_rewrite_tac[]);
a(cases_tac⌜y = x⌝ THEN asm_rewrite_tac[]);
(* *** Goal "2.9" *** *)
a(DROP_NTH_ASM_T 2 ante_tac THEN asm_rewrite_tac[]);
(* *** Goal "2.10" *** *)
a(DROP_NTH_ASM_T 2 ante_tac THEN asm_rewrite_tac[]);
a(cases_tac⌜z = top⌝ THEN asm_rewrite_tac[]);
(* *** Goal "2.11" *** *)
a(DROP_NTH_ASM_T 2 ante_tac THEN asm_rewrite_tac[]);
a(cases_tac⌜z = top⌝ THEN asm_rewrite_tac[]);
save_consistency_thm⌜$dominates⌝(pop_thm());
```

Now we retrieve the definition of *dominates* with the consistency obligation satisfied.

SML
```
val dominates_def = prove_rule[get_spec⌜$dominates⌝]
        ⌜∀ x y z • x dominates x
        ∧ (x dominates y ∧ y dominates x ⇒ x = y)
        ∧ (x dominates y ∧ y dominates z ⇒ x dominates z)⌝;
```

6 CLOSING DOWN

The following ProofPower instruction restores the previous proof context.

SML
```
pop_pc();
```

7 THE THEORY fef004

7.1 Parents

cache′fef *fef003*

7.2 Children

fef005

7.3 Constants

Num $Worth\ \mathbb{P}$
Ide $Ide\ \mathbb{P}$
I_exponent $Int \rightarrow Worth$
I_mantissa $Int \rightarrow Worth$
MkInt $Worth \rightarrow Worth \rightarrow Int$
F_exponent $Float \rightarrow Worth$
F_mantissa $Float \rightarrow Worth$
MkFloat $Worth \rightarrow Worth \rightarrow Float$
accessDenied $Worth$
rowClassTooLow $Worth$
fieldClassOutOfRange $Worth$
noNulls $Worth$
nonUniqueValues $Worth$
nonUniformValues $Worth$
ambiguousHaving $Worth$
ambiguousEvaluate $Worth$
noSuchDirectory $Worth$
noSuchTable $Worth$
classChange $Worth$
downGrade $Worth$
underClassified $Worth$
mayNotBeComplete $Worth$
ambiguousUpdate $Worth$
ambiguousColumn $Worth$

noSuchColumn *Worth*
nullValue *Worth*
wrongType *Worth*
tooTall *Worth*
tooWide *Worth*
notCleared *Worth*
error *Worth*
priceless *Worth*
sterling *Worth*
dinary *Worth*
worthless *Worth*
ExceptionVal *Val*
ClassVal $Class \rightarrow Val$
CodeVal $Code \rightarrow Val$
IntervalVal $Interval \rightarrow Val$
TimeVal $Time \rightarrow Val$
FloatVal $Float \rightarrow Val$
IntVal $Int \rightarrow Val$
StringVal $Ide \rightarrow Val$
BoolVal $Bool \rightarrow Val$
VoidVal *Val*
VI_val $ValuedItem \rightarrow Val$
VI_worth $ValuedItem \rightarrow Worth$
MkValuedItem $Worth \rightarrow Val \rightarrow ValuedItem$
NullItemItem $NullItem \rightarrow Item$
ValuedItemItem
$ValuedItem \rightarrow Item$
Dat_item $Data \rightarrow Item$
Dat_class $Data \rightarrow Class$
MkData $Class \rightarrow Item \rightarrow Data$
DataS $Data\ \mathbb{P}$
DataUpdate $Data \rightarrow Update$
ClassUpdate $Class \rightarrow Update$
ItemUpdate $Item \rightarrow Update$
isData $Update \rightarrow Bool$
isClass $Update \rightarrow Bool$
isItem $Update \rightarrow Bool$
destData $Update \rightarrow Data$
destClass $Update \rightarrow Class$
destItem $Update \rightarrow Item$
CS_max $ColSpec \rightarrow Class$
CS_min $ColSpec \rightarrow Class$
CS_consGroup $ColSpec \rightarrow Worth$
CS_default $ColSpec \rightarrow Data$
CS_nullType $ColSpec \rightarrow Bool$
CS_sterlingType
$ColSpec \rightarrow Type$
CS_dinaryType

	$ColSpec \rightarrow Type$
CS_posn	$ColSpec \rightarrow Worth$
CS_ide	$ColSpec \rightarrow Ide$
MkColSpec	Ide
	$\rightarrow Worth$
	$\rightarrow Type$
	$\rightarrow Type$
	$\rightarrow Bool$
	$\rightarrow Data$
	$\rightarrow Worth$
	$\rightarrow Class$
	$\rightarrow Class$
	$\rightarrow ColSpec$
R_group	$Reference \rightarrow Worth$
R_table	$Reference \rightarrow Tab$
MkReference	$Tab \rightarrow Worth \rightarrow Reference$
CC_referential	
	$ColCon \rightarrow Reference\ LIST$
CC_secondary	$ColCon \rightarrow Bool$
CC_primary	$ColCon \rightarrow Bool$
CC_classLimited	
	$ColCon \rightarrow Bool$
CC_unique	$ColCon \rightarrow Bool$
CC_uniform	$ColCon \rightarrow Bool$
CC_exist	$ColCon \rightarrow Class$
MkColCon	$Class$
	$\rightarrow Bool$
	$\rightarrow Bool$
	$\rightarrow Bool$
	$\rightarrow Bool$
	$\rightarrow Bool$
	$\rightarrow Reference\ LIST$
	$\rightarrow ColCon$
ColConS	$ColCon\ \mathbb{P}$
R_data	$Row \rightarrow Worth \leftrightarrow Data$
R_exist	$Row \rightarrow Class$
MkRow	$Class \rightarrow Worth \leftrightarrow Data \rightarrow Row$
RowS	$Row\ \mathbb{P}$
TS_rows	$TableSpec \rightarrow Row\ LIST$
TS_cons	$TableSpec \rightarrow Worth \leftrightarrow ColCon$
TS_colspecs	$TableSpec \rightarrow ColSpec\ \mathbb{P}$
TS_maxRow	$TableSpec \rightarrow Class$
TS_class	$TableSpec \rightarrow Class$
MkTableSpec	$Class$
	$\rightarrow Class$
	$\rightarrow ColSpec\ \mathbb{P}$
	$\rightarrow Worth \leftrightarrow ColCon$
	$\rightarrow Row\ LIST$

	$\rightarrow$ *TableSpec*
TableSpecS	*TableSpec* $\mathbb{P}$
Dir_class	*Directory* $\rightarrow$ *Class*
Dir_exist	*Directory* $\rightarrow$ *Class*
Dir_tables	*Directory* $\rightarrow$ *Ide* $\leftrightarrow$ *TableSpec*
MkDirectory	*Ide* $\leftrightarrow$ *TableSpec* $\rightarrow$ *Class* $\rightarrow$ *Class* $\rightarrow$ *Directory*
DirectoryS	*Directory* $\mathbb{P}$
IdeL	*Tab* $\mathbb{P}$
StateS	*StateR* $\mathbb{P}$
isState	*StateR* $\rightarrow$ *Bool*
isError	$'a + 'Error \rightarrow Bool$
isVal	$'a + 'Error \rightarrow Bool$
destError	$'a + 'Error \rightarrow 'Error$
destVal	$'a + 'Error \rightarrow 'a$
giveError	$'Error \rightarrow 'a + 'Error$
giveVal	$'a \rightarrow 'a + 'Error$
SelectEffect	*Select* $\rightarrow$ *Effect*
UpdateEffect	*UpDate* $\rightarrow$ *Effect*
DeleteEffect	*Delete* $\rightarrow$ *Effect*
InsertEffect	*Insert* $\rightarrow$ *Effect*
isSelect	*Effect* $\rightarrow$ *Bool*
isUpdate	*Effect* $\rightarrow$ *Bool*
isDelete	*Effect* $\rightarrow$ *Bool*
isInsert	*Effect* $\rightarrow$ *Bool*
destSelect	*Effect* $\rightarrow$ *Select*
destUpdate	*Effect* $\rightarrow$ *UpDate*
destDelete	*Effect* $\rightarrow$ *Delete*
destInsert	*Effect* $\rightarrow$ *Insert*
tabFromEffect	
	Effect $\rightarrow$ *Tab*

7.4 Aliases

null	*One* : *NullItem*
monolean	*One* : *NullItem*
boolean	*One* : *NullItem*
chars	*One* : *NullItem*
integer	*One* : *NullItem*
floating	*One* : *NullItem*
time	*One* : *NullItem*
interval	*One* : *NullItem*
class	*One* : *NullItem*

7.5 Types

Int
Float
Time

Interval
Code
ValuedItem
Data
ColSpec
Reference
ColCon
Row
TableSpec
Directory
State

7.6 Type Abbreviations

Char	*Char*
String	*Ide*
Num	*Worth*
Bool	*Bool*
Ide	*Ide*
Error	*Worth*
Worth	*Worth*
Val	*Val*
NullItem	*NullItem*
Item	*Item*
Update	*Update*
Type	*Type*
Errors	*Errors*
Tab	*Tab*
StateR	*StateR*
Insert	*Insert*
Delete	*Delete*
UpDate	*UpDate*
Select	*Select*
Effect	*Effect*

7.7 Definitions

Num $\vdash Num = Universe$

Ide $\vdash Ide = Universe$

Int $\vdash \exists f \bullet TypeDefn\ (\lambda\ x \bullet T)\ f$

MkInt

I_mantissa

I_exponent
$$\vdash \forall\ t\ x1\ x2 \bullet I\_mantissa\ (MkInt\ x1\ x2) = x1 \wedge I\_exponent\ (MkInt\ x1\ x2) = x2 \wedge MkInt\ (I\_mantissa\ t)\ (I\_exponent\ t) = t$$

Float $\vdash \exists f \bullet TypeDefn\ (\lambda\ x \bullet T)\ f$

MkFloat

F_mantissa
F_exponent $\vdash \forall\ t\ x1\ x2$
$\bullet\ F\_mantissa\ (MkFloat\ x1\ x2) = x1$
$\wedge\ F\_exponent\ (MkFloat\ x1\ x2) = x2$
$\wedge\ MkFloat\ (F\_mantissa\ t)\ (F\_exponent\ t) = t$
error
notCleared
tooWide
tooTall
wrongType
nullValue
noSuchColumn
ambiguousColumn
ambiguousUpdate
mayNotBeComplete
underClassified
downGrade
classChange
noSuchTable
noSuchDirectory
ambiguousEvaluate
ambiguousHaving
nonUniformValues
nonUniqueValues
noNulls
fieldClassOutOfRange
rowClassTooLow
accessDenied $\vdash error = 1$
$\wedge\ notCleared = 2$
$\wedge\ tooWide = 3$
$\wedge\ tooTall = 4$
$\wedge\ wrongType = 5$
$\wedge\ nullValue = 6$
$\wedge\ noSuchColumn = 7$
$\wedge\ ambiguousColumn = 8$
$\wedge\ ambiguousUpdate = 9$
$\wedge\ mayNotBeComplete = 10$
$\wedge\ underClassified = 11$
$\wedge\ downGrade = 12$
$\wedge\ classChange = 13$
$\wedge\ noSuchTable = 14$
$\wedge\ noSuchDirectory = 15$
$\wedge\ ambiguousEvaluate = 16$
$\wedge\ ambiguousHaving = 17$
$\wedge\ nonUniformValues = 18$
$\wedge\ nonUniqueValues = 19$
$\wedge\ noNulls = 20$
$\wedge\ fieldClassOutOfRange = 21$

$\wedge$ *rowClassTooLow* = *22*
$\wedge$ *accessDenied* = *23*

worthless
dinary
sterling
priceless $\vdash$ *worthless* = *1*
$\wedge$ *dinary* = *2*
$\wedge$ *sterling* = *3*
$\wedge$ *priceless* = *4*

VoidVal
BoolVal
StringVal
IntVal
FloatVal
TimeVal
IntervalVal
CodeVal
ClassVal
ExceptionVal $\vdash \forall\ b\ s\ i\ f\ t\ n\ c\ cl$
• $VoidVal = InL\ class$
$\wedge\ BoolVal\ b = InR\ (InL\ b)$
$\wedge\ StringVal\ s = InR\ (InR\ (InL\ s))$
$\wedge\ IntVal\ i = InR\ (InR\ (InR\ (InL\ i)))$
$\wedge\ FloatVal\ f = InR\ (InR\ (InR\ (InR\ (InL\ f))))$
$\wedge\ TimeVal\ t = InR\ (InR\ (InR\ (InR\ (InR\ (InL\ t)))))$
$\wedge\ IntervalVal\ n$
$= InR\ (InR\ (InR\ (InR\ (InR\ (InR\ (InL\ n))))))$
$\wedge\ CodeVal\ c$
$= InR$
$(InR\ (InR\ (InR\ (InR\ (InR\ (InR\ (InL\ c)))))))$
$\wedge\ ClassVal\ cl$
$= InR$
$(InR$
$(InR$
$(InR$
$(InR$
$(InR$
$(InR\ (InR\ (InL\ cl))))))))$
$\wedge\ ExceptionVal$
$= InR$
$(InR$
$(InR$
$(InR$
$(InR$
$(InR$
$(InR$
$(InR$
$(InR$
$(InR$

$$class))))))))$$

ValuedItem $\vdash \exists f \bullet TypeDefn\ (\lambda\ x \bullet\ T)\ f$

MkValuedItem

VI_worth

VI_val $\vdash \forall\ t\ x1\ x2$
$\bullet\ VI\_worth\ (MkValuedItem\ x1\ x2) = x1$
$\wedge\ VI\_val\ (MkValuedItem\ x1\ x2) = x2$
$\wedge\ MkValuedItem\ (VI\_worth\ t)\ (VI\_val\ t) = t$

ValuedItemItem

NullItemItem $\vdash \forall\ v\ n$
$\bullet\ ValuedItemItem\ v = InL\ v \wedge NullItemItem\ n = InR\ n$

Data $\vdash \exists f \bullet TypeDefn\ (\lambda\ x \bullet\ T)\ f$

MkData

Dat_class

Dat_item $\vdash \forall\ t\ x1\ x2$
$\bullet\ Dat\_class\ (MkData\ x1\ x2) = x1$
$\wedge\ Dat\_item\ (MkData\ x1\ x2) = x2$
$\wedge\ MkData\ (Dat\_class\ t)\ (Dat\_item\ t) = t$

DataS $\vdash DataS = Universe$

ItemUpdate

ClassUpdate

DataUpdate $\vdash \forall\ i\ c\ d$
$\bullet\ ItemUpdate\ i = InL\ i$
$\wedge\ ClassUpdate\ c = InR\ (InL\ c)$
$\wedge\ DataUpdate\ d = InR\ (InR\ d)$

isItem

isClass

isData $\vdash \forall\ u$
$\bullet\ (isItem\ u \Leftrightarrow (\exists\ i \bullet\ u = ItemUpdate\ i))$
$\wedge\ (isClass\ u \Leftrightarrow (\exists\ c \bullet\ u = ClassUpdate\ c))$
$\wedge\ (isData\ u \Leftrightarrow (\exists\ d \bullet\ u = DataUpdate\ d))$

destItem

destClass

destData $\vdash ConstSpec$
$(\lambda\ (destItem', destClass', destData')$
$\bullet \forall\ i\ c\ d$
$\bullet\ destItem'\ (ItemUpdate\ i) = i$
$\wedge\ destClass'\ (ClassUpdate\ c) = c$
$\wedge\ destData'\ (DataUpdate\ d) = d)$
$(destItem, destClass, destData)$

ColSpec $\vdash \exists f \bullet TypeDefn\ (\lambda\ x \bullet\ T)\ f$

MkColSpec

CS_ide

CS_posn

CS_dinaryType

CS_sterlingType

CS_nullType

CS_default

CS_consGroup
CS_min
CS_max $\vdash \forall\ t\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9$
$\bullet\ CS\_ide\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x1$
$\wedge\ CS\_posn\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x2$
$\wedge\ CS\_dinaryType\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x3$
$\wedge\ CS\_sterlingType\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x4$
$\wedge\ (CS\_nullType\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) \Leftrightarrow x5)$
$\wedge\ CS\_default\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x6$
$\wedge\ CS\_consGroup\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x7$
$\wedge\ CS\_min\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x8$
$\wedge\ CS\_max\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x9$
$\wedge\ MkColSpec\ (CS\_ide\ t)\ (CS\_posn\ t)\ (CS\_dinaryType\ t)\ (CS\_sterlingType\ t)\ (CS\_nullType\ t)\ (CS\_default\ t)\ (CS\_consGroup\ t)\ (CS\_min\ t)\ (CS\_max\ t) = t$
Reference $\vdash \exists\ f\bullet\ TypeDefn\ (\lambda\ x\bullet\ T)\ f$
MkReference
R_table
R_group $\vdash \forall\ t\ x1\ x2$
$\bullet\ R\_table\ (MkReference\ x1\ x2) = x1$
$\wedge\ R\_group\ (MkReference\ x1\ x2) = x2$
$\wedge\ MkReference\ (R\_table\ t)\ (R\_group\ t) = t$
ColCon $\vdash \exists\ f\bullet\ TypeDefn\ (\lambda\ x\bullet\ T)\ f$
MkColCon
CC_exist
CC_uniform
CC_unique

CC_classLimited
CC_primary
CC_secondary
CC_referential

$$
\begin{array}{l}
\vdash \forall\ t\ x1\ x2\ x3\ x4\ x5\ x6\ x7 \\
\quad \bullet\ CC\_exist\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) = x1 \\
\qquad \wedge\ (CC\_uniform\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\
\qquad\quad \Leftrightarrow x2) \\
\qquad \wedge\ (CC\_unique\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\
\qquad\quad \Leftrightarrow x3) \\
\qquad \wedge\ (CC\_classLimited \\
\qquad\quad (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\
\qquad\quad \Leftrightarrow x4) \\
\qquad \wedge\ (CC\_primary\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\
\qquad\quad \Leftrightarrow x5) \\
\qquad \wedge\ (CC\_secondary\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\
\qquad\quad \Leftrightarrow x6) \\
\qquad \wedge\ CC\_referential\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\
\qquad\quad = x7 \\
\qquad \wedge\ MkColCon \\
\qquad\quad (CC\_exist\ t) \\
\qquad\quad (CC\_uniform\ t) \\
\qquad\quad (CC\_unique\ t) \\
\qquad\quad (CC\_classLimited\ t) \\
\qquad\quad (CC\_primary\ t) \\
\qquad\quad (CC\_secondary\ t) \\
\qquad\quad (CC\_referential\ t) \\
\qquad\quad = t
\end{array}
$$

ColConS $\vdash ColConS = Universe$

Row $\vdash \exists\ f \bullet\ TypeDefn\ (\lambda\ x \bullet\ T)\ f$

MkRow
R_exist

R_data

$$
\begin{array}{l}
\vdash \forall\ t\ x1\ x2 \\
\quad \bullet\ R\_exist\ (MkRow\ x1\ x2) = x1 \\
\qquad \wedge\ R\_data\ (MkRow\ x1\ x2) = x2 \\
\qquad \wedge\ MkRow\ (R\_exist\ t)\ (R\_data\ t) = t
\end{array}
$$

RowS $\vdash RowS = \{r | let\ data = R\_data\ r\ in\ data \in Num \nrightarrow DataS\}$

TableSpec $\vdash \exists\ f \bullet\ TypeDefn\ (\lambda\ x \bullet\ T)\ f$

MkTableSpec
TS_class
TS_maxRow
TS_colspecs
TS_cons

TS_rows

$$
\begin{array}{l}
\vdash \forall\ t\ x1\ x2\ x3\ x4\ x5 \\
\quad \bullet\ TS\_class\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x1 \\
\qquad \wedge\ TS\_maxRow\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x2 \\
\qquad \wedge\ TS\_colspecs\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x3 \\
\qquad \wedge\ TS\_cons\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x4
\end{array}
$$

$$
\begin{array}{l}
\quad \wedge\ TS\_rows\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x5 \\
\quad \wedge\ MkTableSpec \\
\qquad (TS\_class\ t) \\
\qquad (TS\_maxRow\ t) \\
\qquad (TS\_colspecs\ t) \\
\qquad (TS\_cons\ t) \\
\qquad (TS\_rows\ t) \\
\qquad = t
\end{array}
$$

TableSpecS $\vdash TableSpecS$

$$
\begin{array}{l}
\quad = \{ts \\
\quad | let\ cns = TS\_cons\ ts\ and\ rows = TS\_rows\ ts \\
\qquad in\ cns \in Num \pfun ColConS \\
\qquad\quad \wedge\ (\forall\ r \bullet\ r \in_l rows \Rightarrow r \in RowS)\}
\end{array}
$$

Directory $\vdash \exists\ f \bullet\ TypeDefn\ (\lambda\ x \bullet\ T)\ f$

MkDirectory

Dir_tables

Dir_exist

Dir_class

$$
\begin{array}{l}
\vdash \forall\ t\ x1\ x2\ x3 \\
\quad \bullet\ Dir\_tables\ (MkDirectory\ x1\ x2\ x3) = x1 \\
\qquad \wedge\ Dir\_exist\ (MkDirectory\ x1\ x2\ x3) = x2 \\
\qquad \wedge\ Dir\_class\ (MkDirectory\ x1\ x2\ x3) = x3 \\
\qquad \wedge\ MkDirectory \\
\qquad\quad (Dir\_tables\ t) \\
\qquad\quad (Dir\_exist\ t) \\
\qquad\quad (Dir\_class\ t) \\
\qquad\quad = t
\end{array}
$$

DirectoryS $\vdash DirectoryS = \{d | Dir\_tables\ d \in Ide \pfun TableSpecS\}$

IdeL $\vdash IdeL = Universe$

StateS $\vdash StateS = IdeL \pfun DirectoryS$

isState $\vdash \forall\ s \bullet\ isState\ s \Leftrightarrow s \in StateS$

state_type_def

$\vdash \exists\ f \bullet\ TypeDefn\ isState\ f$

giveVal

giveError

destVal

destError

isVal

isError

$$
\begin{array}{l}
\vdash ConstSpec \\
\quad (\lambda \\
\qquad (giveVal', giveError', destVal', destError', \\
\qquad\quad isVal', isError') \\
\qquad \bullet \forall\ v\ e\ ve \\
\qquad\quad \bullet\ giveVal'\ v = InL\ v \\
\qquad\qquad \wedge\ giveError'\ e = InR\ e \\
\qquad\qquad \wedge\ destVal'\ (giveVal'\ v) = v \\
\qquad\qquad \wedge\ destError'\ (giveError'\ e) = e \\
\qquad\qquad \wedge\ (isVal'\ ve \Leftrightarrow (\exists\ v_1 \bullet\ ve = giveVal'\ v_1)) \\
\qquad\qquad \wedge\ (isError'\ ve
\end{array}
$$

$$\Leftrightarrow (\exists\ e_1 \bullet\ ve = giveError'\ e_1)))$$
$$(giveVal,\ giveError,\ destVal,\ destError,\ isVal,\ isError)$$

InsertEffect
DeleteEffect
UpdateEffect
SelectEffect $\vdash \forall\ i\ d\ u\ s$
$\bullet\ InsertEffect\ i = InL\ i$
$\wedge\ DeleteEffect\ d = InR\ (InL\ d)$
$\wedge\ UpdateEffect\ u = InR\ (InR\ (InL\ u))$
$\wedge\ SelectEffect\ s = InR\ (InR\ (InR\ s))$

isInsert
isDelete
isUpdate
isSelect $\vdash \forall\ e$
$\bullet\ (isInsert\ e \Leftrightarrow (\exists\ i \bullet\ e = InsertEffect\ i))$
$\wedge\ (isDelete\ e \Leftrightarrow (\exists\ d \bullet\ e = DeleteEffect\ d))$
$\wedge\ (isUpdate\ e \Leftrightarrow (\exists\ u \bullet\ e = UpdateEffect\ u))$
$\wedge\ (isSelect\ e \Leftrightarrow (\exists\ s \bullet\ e = SelectEffect\ s))$

destInsert
destDelete
destUpdate
destSelect $\vdash ConstSpec$
$(\lambda$
$(destInsert',\ destDelete',\ destUpdate',\ destSelect')$
$\bullet \forall\ i\ d\ u\ s$
$\bullet\ destInsert'\ (InsertEffect\ i) = i$
$\wedge\ destDelete'\ (DeleteEffect\ d) = d$
$\wedge\ destUpdate'\ (UpdateEffect\ u) = u$
$\wedge\ destSelect'\ (SelectEffect\ s) = s)$
$(destInsert,\ destDelete,\ destUpdate,\ destSelect)$

tabFromEffect
$\vdash ConstSpec$
$(\lambda\ tabFromEffect'$
$\bullet \forall\ i\ d\ u$
$\bullet\ tabFromEffect'\ (InsertEffect\ i) = Fst\ i$
$\wedge\ tabFromEffect'\ (DeleteEffect\ d) = Fst\ d$
$\wedge\ tabFromEffect'\ (UpdateEffect\ u) = Fst\ u)$
$tabFromEffect$

7.8 Theorems

dominates_consistent
lattice_bottom_consistent
lattice_top_consistent
lub_consistent
glb_consistent

$$
\begin{array}{l}
\vdash \mathit{Consistent} \\
\quad (\lambda \\
\qquad (\mathit{dominates}', \mathit{lattice\_bottom}', \mathit{lattice\_top}', \\
\qquad\quad \mathit{lub}', \mathit{glb}') \\
\qquad \bullet \forall\ x\ y\ z \\
\qquad\quad \bullet\ \mathit{dominates}'\ x\ x \\
\qquad\qquad \wedge\ (\mathit{dominates}'\ x\ y \wedge \mathit{dominates}'\ y\ x \Rightarrow x = y) \\
\qquad\qquad \wedge\ (\mathit{dominates}'\ x\ y \wedge \mathit{dominates}'\ y\ z \\
\qquad\qquad\quad \Rightarrow \mathit{dominates}'\ x\ z) \\
\qquad\qquad \wedge\ (\forall\ x \bullet\ \mathit{dominates}'\ x\ \mathit{lattice\_bottom}') \\
\qquad\qquad \wedge\ (\forall\ x \bullet\ \mathit{dominates}'\ \mathit{lattice\_top}'\ x) \\
\qquad\qquad \wedge\ (\forall\ x\ y\ z \\
\qquad\qquad \bullet\ \mathit{dominates}'\ (\mathit{lub}'\ x\ y)\ x \\
\qquad\qquad\quad \wedge\ \mathit{dominates}'\ (\mathit{lub}'\ x\ y)\ y \\
\qquad\qquad\quad \wedge\ (\mathit{dominates}'\ z\ x \wedge \mathit{dominates}'\ z\ y \\
\qquad\qquad\qquad \Rightarrow \mathit{dominates}'\ z\ (\mathit{lub}'\ x\ y))) \\
\qquad\qquad \wedge\ (\forall\ x\ y\ z \\
\qquad\qquad \bullet\ \mathit{dominates}'\ x\ (\mathit{glb}'\ x\ y) \\
\qquad\qquad\quad \wedge\ \mathit{dominates}'\ y\ (\mathit{glb}'\ x\ y) \\
\qquad\qquad\quad \wedge\ (\mathit{dominates}'\ x\ z \wedge \mathit{dominates}'\ y\ z \\
\qquad\qquad\qquad \Rightarrow \mathit{dominates}'\ (\mathit{glb}'\ x\ y)\ z)))
\end{array}
$$

8 INDEX