

Project: DRA FRONT END FILTER PROJECT

Title: Specification of SSQL Semantics I

Ref: DS/FMU/FEF/004 *Issue: Revision :* 4.3 *Date:* 5 December 2009

Status: Approved *Type:* Specification

Keywords:

Author:

<i>Name</i>	<i>Location</i>	<i>Signature</i>	<i>Date</i>
G. M. Prout	WIN01		

Authorisation for Issue:

<i>Name</i>	<i>Function</i>	<i>Signature</i>	<i>Date</i>
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 DOCUMENT CONTROL	2
0.1 Contents List	2
0.2 Document Cross References	2
0.3 Changes History	3
0.4 Changes Forecast	3
1 GENERAL	4
1.1 Scope	4
1.2 Introduction	4
1.3 Terminology	4
2 PRELIMINARIES	4
3 DATA TYPES IN THE STATE	5
4 FUNCTIONALITY OF THE QUERY PROCESSING	14
4.1 Auxiliary Functions	14
4.2 Type of <i>Effect</i>	14
4.3 The function <i>processQuery</i>	16
5 AUXILIARY PROOF WORK	16
5.1 Consistency Proof for <i>dominates</i>	16
6 CLOSING DOWN	19
7 THE THEORY fef004	20
7.1 Parents	20
7.2 Children	20
7.3 Constants	20
7.4 Aliases	28
7.5 Types	28
7.6 Type Abbreviations	28
7.7 Definitions	29
7.8 Theorems	36
8 INDEX	38

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 : 4.3 (5 December 2009) 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 \leftrightarrow 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", [],  $\Gamma$ : CHAR $^\top$ );
```

SML

```
| declare_type_abbrev("String", [],  $\Gamma$ : STRING $^\top$ );
```

SML

```
| declare_type_abbrev("Num", [],  $\Gamma$ :  $\mathbb{N}^\top$ );
```

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

HOL Constant

Num	: <i>Num</i> \mathbb{P}
------------	---------------------------

<i>Num</i> = <i>Universe</i>

SML

```
| declare_type_abbrev("Bool", [],  $\Gamma$ : BOOL $^\top$ );
```

SML

```
| declare_type_abbrev("Ide", [],  $\Gamma$ : String $^\top$ );
```

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

HOL Constant

Ide	: <i>Ide</i> \mathbb{P}
------------	---------------------------

<i>Ide</i> = <i>Universe</i>

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*

	<i>error</i>	= 1
∧	<i>notCleared</i>	= 2
∧	<i>tooWide</i>	= 3
∧	<i>tooTall</i>	= 4
∧	<i>wrongType</i>	= 5
∧	<i>nullValue</i>	= 6
∧	<i>noSuchColumn</i>	= 7
∧	<i>ambiguousColumn</i>	= 8
∧	<i>ambiguousUpdate</i>	= 9
∧	<i>mayNotBeComplete</i>	= 10
∧	<i>underClassified</i>	= 11
∧	<i>downGrade</i>	= 12
∧	<i>classChange</i>	= 13
∧	<i>noSuchTable</i>	= 14
∧	<i>noSuchDirectory</i>	= 15
∧	<i>ambiguousEvaluate</i>	= 16
∧	<i>ambiguousHaving</i>	= 17
∧	<i>nonUniformValues</i>	= 18
∧	<i>nonUniqueValues</i>	= 19
∧	<i>noNulls</i>	= 20
∧	<i>fieldClassOutOfRange</i>	= 21
∧	<i>rowClassTooLow</i>	= 22
∧	<i>accessDenied</i>	= 23

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
 \wedge *dinary* = 2
 \wedge *sterling* = 3
 \wedge *priceless* = 4

SML

```
declare_type_abbrev("Worth",[], $\vdash$ : Num $\top$ );
```

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",[], $\vdash$ : ONE + Bool + String + Int + Float + Time + Interval + Code + Class + ONE $\top$ );
```

We give constructors for type *Val*.

HOL Constant

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

\forall *b:Bool; s:String; i:Int; f:Float; t:Time; n:Interval; c:Code; cl:Class*

- *VoidVal* = *InL One*
- \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 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",[],Γ: ONE⊥);
declare_alias("null",Γ One⊥);
```

Item is declared as a sum type.

SML

```
declare_type_abbrev("Item",[],Γ: 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* \mathbb{P}

DataS = *Universe*

SML

```
declare_type_abbrev("Update",[],Γ: Item + Class + Data⊥);
```

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

HOL Constant

ItemUpdate	:	$Item \rightarrow Update;$
ClassUpdate	:	$Class \rightarrow Update;$
DataUpdate	:	$Data \rightarrow Update$

 $\forall i:Item; c:Class; d:Data$

- $ItemUpdate\ i = InL\ i$
- \wedge $ClassUpdate\ c = InR\ (InL\ c)$
- \wedge $DataUpdate\ d = InR\ (InR\ d)$

HOL Constant

isItem	:	$Update \rightarrow Bool;$
isClass	:	$Update \rightarrow Bool;$
isData	:	$Update \rightarrow Bool$

 $\forall u:Update$

- $(isItem\ u = \exists i:Item \bullet u = ItemUpdate\ i)$
- \wedge $(isClass\ u = \exists c:Class \bullet u = ClassUpdate\ c)$
- \wedge $(isData\ u = \exists d:Data \bullet u = DataUpdate\ d)$

HOL Constant

destItem	:	$Update \rightarrow Item;$
destClass	:	$Update \rightarrow Class;$
destData	:	$Update \rightarrow Data$

 $\forall i:Item; c:Class; d:Data$

- $destItem\ (ItemUpdate\ i) = i$
- \wedge $destClass\ (ClassUpdate\ c) = c$
- \wedge $destData\ (DataUpdate\ d) = d$

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

SML

```
declare_alias("monolean", $\ulcorner One \urcorner$ );
declare_alias("boolean", $\ulcorner One \urcorner$ );
declare_alias("chars", $\ulcorner One \urcorner$ );
declare_alias("integer", $\ulcorner One \urcorner$ );
declare_alias("floating", $\ulcorner One \urcorner$ );
declare_alias("time", $\ulcorner One \urcorner$ );
declare_alias("interval", $\ulcorner One \urcorner$ );
declare_alias("class", $\ulcorner One \urcorner$ );
```

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	: <i>Ide</i> ;
CS_posn	: <i>Num</i> ;
CS_dinaryType	: <i>Type</i> ;
CS_sterlingType	: <i>Type</i> ;
CS_nullType	: <i>Bool</i> ;
CS_default	: <i>Data</i> ;
CS_consGroup	: <i>Num</i> ;
CS_min	: <i>Class</i> ;
CS_max	: <i>Class</i>

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	: <i>Tab</i> ;
R_group	: <i>Num</i>

HOL Labelled Product

ColCon

CC_exist	: <i>Class</i> ;
CC_uniform	: <i>Bool</i> ;
CC_unique	: <i>Bool</i> ;
CC_classLimited	: <i>Bool</i> ;
CC_primary	: <i>Bool</i> ;
CC_secondary	: <i>Bool</i> ;
CC_referential	: <i>Reference LIST</i>

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

HOL Constant

ColConS	: <i>ColCon</i> \mathbb{P}
----------------	------------------------------

<i>ColConS</i> = Universe

HOL Labelled Product

Row

R_exist	: <i>Class</i> ;
R_data	: <i>Num</i> \leftrightarrow <i>Data</i>

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

HOL Constant

RowS	: <i>Row</i> \mathbb{P}
-------------	---------------------------

<i>RowS</i> = { <i>r</i> let <i>data</i> = <i>R_data</i> <i>r</i> in <i>data</i> \in <i>Num</i> \leftrightarrow <i>DataS</i> }
--

HOL Labelled Product

TableSpec

TS_class	: <i>Class</i> ;
TS_maxRow	: <i>Class</i> ;
TS_colspecs	: <i>ColSpec</i> \mathbb{P} ;
TS_cons	: <i>Num</i> \leftrightarrow <i>ColCon</i> ;
TS_rows	: <i>Row</i> 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

TableSpecS	: <i>TableSpec</i> \mathbb{P}
-------------------	---------------------------------

<i>TableSpecS</i> = { <i>ts</i> let <i>cns</i> = <i>TS_cons</i> <i>ts</i> and <i>rows</i> = <i>TS_rows</i> <i>ts</i> in <i>cns</i> \in <i>Num</i> \leftrightarrow <i>ColConS</i> \wedge $\forall r \bullet r \in_l$ <i>rows</i> $\Rightarrow r \in$ <i>RowS</i> }
--

HOL Labelled Product

Directory

Dir_tables	: <i>Ide</i> \leftrightarrow <i>TableSpec</i> ;
Dir_exist	: <i>Class</i> ;
Dir_class	: <i>Class</i>

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

HOL Constant

DirectoryS : <i>Directory</i> \mathbb{P}
$DirectoryS = \{d \mid Dir_tables\ d \in Ide \mapsto TableSpecS\}$

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

HOL Constant

IdeL : (<i>Ide LIST</i>) \mathbb{P}
$IdeL = Universe$

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

SML

```
declare_type_abbrev("StateR", [],  $\ulcorner$ : Ide LIST  $\leftrightarrow$  Directory  $\urcorner$ );
```

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

HOL Constant

StateS : <i>StateR</i> \mathbb{P}
$StateS = IdeL \mapsto DirectoryS$

We define the property required on the representation state.

HOL Constant

isState : <i>StateR</i> \rightarrow <i>Bool</i>
$\forall s \bullet isState\ s \Leftrightarrow s \in StateS$

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

SML

```
push_goal([],  $\ulcorner$  $\exists s : StateR \bullet isState\ s\urcorner$ );
a(rewrite_tac[get_spec $\ulcorner$ isState $\urcorner$ ]);
a( $\exists$ -tac $\ulcorner$ { $\urcorner$ });
a(rewrite_tac[get_spec $\ulcorner$ StateS $\urcorner$ ,  $\mapsto$ -def, functional_def,  $\cap$ -def,  $\leftrightarrow$ -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

```

giveVal      : 'a → 'a + 'Error;
giveError   : 'Error → 'a + 'Error;
destVal     : 'a + 'Error → 'a;
destError   : 'a + 'Error → 'Error;
isVal       : 'a + 'Error → Bool;
isError     : 'a + 'Error → Bool

```

$\forall v; e; ve$

```

•   giveVal v           = InL v
∧   giveError e        = InR e
∧   destVal (giveVal v) = v
∧   destError(giveError e) = e
∧   (isVal ve          = ∃ v1 • ve = giveVal v1)
∧   (isError ve        = ∃ e1 • ve = giveError e1)

```

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("Insert",[],Γ: Tab × (Num ↔ Data) LISTΓ);

```

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("Delete",[],Γ: Tab × Num ℙΓ);

```

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("UpDate",[],⌈: Tab × (Num ↔ (Num ↔ Update))⌋);
```

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("Select",[],⌈: Data LIST LIST⌋);
```

This gives us the sum type *Effect*.

SML

```
|declare_type_abbrev("Effect",[],⌈: Insert + Delete + UpDate + Select⌋);
```

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

HOL Constant

InsertEffect	:	<i>Insert</i> → <i>Effect</i> ;
DeleteEffect	:	<i>Delete</i> → <i>Effect</i> ;
UpdateEffect	:	<i>UpDate</i> → <i>Effect</i> ;
SelectEffect	:	<i>Select</i> → <i>Effect</i>

∀ *i:Insert*; *d>Delete*; *u:UpDate*; *s>Select*

•	<i>InsertEffect</i> <i>i</i>	=	<i>InL</i> <i>i</i>
∧	<i>DeleteEffect</i> <i>d</i>	=	<i>InR</i> (<i>InL</i> <i>d</i>)
∧	<i>UpdateEffect</i> <i>u</i>	=	<i>InR</i> (<i>InR</i> (<i>InL</i> <i>u</i>))
∧	<i>SelectEffect</i> <i>s</i>	=	<i>InR</i> (<i>InR</i> (<i>InR</i> <i>s</i>))

HOL Constant

isInsert	:	<i>Effect</i> → <i>Bool</i> ;
isDelete	:	<i>Effect</i> → <i>Bool</i> ;
isUpdate	:	<i>Effect</i> → <i>Bool</i> ;
isSelect	:	<i>Effect</i> → <i>Bool</i>

∀ *e :Effect*

•	<i>(isInsert</i> <i>e</i>	=	∃ <i>i:Insert</i>	•	<i>e = InsertEffect</i> <i>i</i>)
∧	<i>(isDelete</i> <i>e</i>	=	∃ <i>d>Delete</i>	•	<i>e = DeleteEffect</i> <i>d</i>)
∧	<i>(isUpdate</i> <i>e</i>	=	∃ <i>u:UpDate</i>	•	<i>e = UpdateEffect</i> <i>u</i>)
∧	<i>(isSelect</i> <i>e</i>	=	∃ <i>s>Select</i>	•	<i>e = SelectEffect</i> <i>s</i>)

HOL Constant

```

destInsert   : Effect → Insert;
destDelete  : Effect → Delete;
destUpdate  : Effect → UpDate;
destSelect  : Effect → Select

```

```

∀      i:Insert; d:Delete; u:UpDate; s:Select
•      destInsert(InsertEffect i)    = i
∧      destDelete(DeleteEffect d)   = d
∧      destUpdate(UpdateEffect u)   = u
∧      destSelect(SelectEffect s)   = s

```

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

```

tabFromEffect   : Effect → Tab

```

```

∀      i:Insert; d:Delete; u:UpDate
•      tabFromEffect (InsertEffect i)    = Fst i
∧      tabFromEffect (DeleteEffect d)   = Fst d
∧      tabFromEffect (UpdateEffect u)   = Fst u

```

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( $\exists$ -tacΓ( $\lambda$  x y • T),top,top,( $\lambda$  x y • x),( $\lambda$  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( $\exists$ -tacΓ(
  ( $\lambda$  c1 c2 • c1 = top  $\vee$  c2 = bot  $\vee$  c1 = c2),
  bot,
  top,
  ( $\lambda$  c1 c2 •
    if c1 = top
    then top
    else if c1 = bot
    then c2
    else if c2 = c1  $\vee$  c2 = bot
    then c1
    else top),
  ( $\lambda$  c1 c2 •
    if c1 = bot
    then bot
    else if c1 = top
    then c2
    else if c2 = c1  $\vee$  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_tacΓx = top∇ THEN asm_rewrite_tac[]);
a(cases_tacΓx = bot∇ THEN asm_rewrite_tac[]);
a(cases_tacΓy = x∇ THEN asm_rewrite_tac[]);
a(cases_tacΓy = bot∇ THEN asm_rewrite_tac[]);
(* *** Goal "2.2" *** *)
a(all_asm_ante_tac THEN cases_tacΓx = top∇ THEN asm_rewrite_tac[]);
a(cases_tacΓx = bot∇ THEN asm_rewrite_tac[]);
a(cases_tacΓy = x∇ THEN asm_rewrite_tac[]);
a(cases_tacΓy = bot∇ THEN asm_rewrite_tac[]);
(* *** Goal "2.3" *** *)
a(GET_ASM_TΓ¬ 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_TΓ¬ 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_tacΓx = top∇ THEN asm_rewrite_tac[]);
a(cases_tacΓx = bot∇ THEN asm_rewrite_tac[]);
(* *** Goal "2.6" *** *)
a(cases_tacΓy = top∇ THEN asm_rewrite_tac[]);
a(cases_tacΓy = bot∇ THEN asm_rewrite_tac[]);
(* *** Goal "2.7" *** *)
a(POP_ASM_T ante_tac THEN asm_rewrite_tac[]);
a(cases_tacΓx = bot∇ THEN asm_rewrite_tac[]);
a(cases_tacΓy = x∇ THEN asm_rewrite_tac[]);
a(cases_tacΓy = top∇ THEN asm_rewrite_tac[]);
(* *** Goal "2.8" *** *)
a(POP_ASM_T ante_tac THEN asm_rewrite_tac[]);
a(cases_tacΓx = bot∇ THEN asm_rewrite_tac[]);
a(cases_tacΓx = top∇ THEN asm_rewrite_tac[]);
a(cases_tacΓy = 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_tacΓz = top∇ THEN asm_rewrite_tac[]);
(* *** Goal "2.11" *** *)
a(DROP_NTH_ASM_T 2 ante_tac THEN asm_rewrite_tac[]);
a(cases_tacΓz = top∇ THEN asm_rewrite_tac[]);
save_consistency_thmΓ$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	\mathbb{N} SET
Ide	CHAR LIST SET
I_exponent	$Int \rightarrow \mathbb{N}$
I_mantissa	$Int \rightarrow \mathbb{N}$
MkInt	$\mathbb{N} \rightarrow \mathbb{N} \rightarrow Int$
F_exponent	$Float \rightarrow \mathbb{N}$
F_mantissa	$Float \rightarrow \mathbb{N}$
MkFloat	$\mathbb{N} \rightarrow \mathbb{N} \rightarrow Float$
accessDenied	\mathbb{N}
rowClassTooLow	\mathbb{N}
fieldClassOutOfRange	\mathbb{N}
noNulls	\mathbb{N}
nonUniqueValues	\mathbb{N}
nonUniformValues	\mathbb{N}
ambiguousHaving	\mathbb{N}
ambiguousEvaluate	\mathbb{N}
noSuchDirectory	\mathbb{N}
noSuchTable	\mathbb{N}
classChange	\mathbb{N}
downGrade	\mathbb{N}
underClassified	\mathbb{N}
mayNotBeComplete	\mathbb{N}
ambiguousUpdate	\mathbb{N}
ambiguousColumn	\mathbb{N}

noSuchColumn	\mathbb{N}
nullValue	\mathbb{N}
wrongType	\mathbb{N}
tooTall	\mathbb{N}
tooWide	\mathbb{N}
notCleared	\mathbb{N}
error	\mathbb{N}
priceless	\mathbb{N}
sterling	\mathbb{N}
dinary	\mathbb{N}
worthless	\mathbb{N}
ExceptionVal	<i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
ClassVal	<i>Class</i> → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
CodeVal	<i>Code</i> → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
IntervalVal	<i>Interval</i> → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i>

	<ul style="list-style-type: none"> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
TimeVal	<ul style="list-style-type: none"> <i>Time</i> → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
FloatVal	<ul style="list-style-type: none"> <i>Float</i> → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
IntVal	<ul style="list-style-type: none"> <i>Int</i> → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
StringVal	<ul style="list-style-type: none"> <i>CHAR LIST</i> → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i>

	<ul style="list-style-type: none"> + <i>Class</i> + <i>ONE</i>
BoolVal	<ul style="list-style-type: none"> <i>BOOL</i> → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
VoidVal	<ul style="list-style-type: none"> <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
VI_val	<ul style="list-style-type: none"> <i>ValuedItem</i> → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i>
VI_worth	<i>ValuedItem</i> → \mathbb{N}
MkValuedItem	<ul style="list-style-type: none"> \mathbb{N} → <i>ONE</i> + <i>BOOL</i> + <i>CHAR LIST</i> + <i>Int</i> + <i>Float</i> + <i>Time</i> + <i>Interval</i> + <i>Code</i> + <i>Class</i> + <i>ONE</i> → <i>ValuedItem</i>
NullItemItem	<i>ONE</i> → <i>ValuedItem</i> + <i>ONE</i>

ValuedItemItem

	$ValuedItem \rightarrow ValuedItem + ONE$
Dat_item	$Data \rightarrow ValuedItem + ONE$
Dat_class	$Data \rightarrow Class$
MkData	$Class \rightarrow ValuedItem + ONE \rightarrow Data$
DataS	$Data SET$
DataUpdate	$Data \rightarrow (ValuedItem + ONE) + Class + Data$
ClassUpdate	$Class \rightarrow (ValuedItem + ONE) + Class + Data$
ItemUpdate	$ValuedItem + ONE \rightarrow (ValuedItem + ONE) + Class + Data$
isData	$(ValuedItem + ONE) + Class + Data \rightarrow BOOL$
isClass	$(ValuedItem + ONE) + Class + Data \rightarrow BOOL$
isItem	$(ValuedItem + ONE) + Class + Data \rightarrow BOOL$
destData	$(ValuedItem + ONE) + Class + Data \rightarrow Data$
destClass	$(ValuedItem + ONE) + Class + Data \rightarrow Class$
destItem	$(ValuedItem + ONE) + Class + Data \rightarrow ValuedItem + ONE$
CS_max	$ColSpec \rightarrow Class$
CS_min	$ColSpec \rightarrow Class$
CS_consGroup	$ColSpec \rightarrow \mathbb{N}$
CS_default	$ColSpec \rightarrow Data$
CS_nullType	$ColSpec \rightarrow BOOL$
CS_sterlingType	$ColSpec \rightarrow ONE + ONE + ONE + ONE + ONE + ONE + ONE + ONE$
CS_dinaryType	$ColSpec \rightarrow ONE + ONE + ONE + ONE + ONE + ONE + ONE + ONE$
CS_posn	$ColSpec \rightarrow \mathbb{N}$
CS_ide	$ColSpec \rightarrow CHAR LIST$
MkColSpec	$CHAR LIST$ $\rightarrow \mathbb{N}$ $\rightarrow ONE + ONE + ONE + ONE + ONE + ONE + ONE + ONE$ $\rightarrow ONE + ONE + ONE + ONE + ONE + ONE + ONE + ONE$ $\rightarrow BOOL$ $\rightarrow Data$ $\rightarrow \mathbb{N}$ $\rightarrow Class$ $\rightarrow Class$ $\rightarrow ColSpec$
R_group	$Reference \rightarrow \mathbb{N}$
R_table	$Reference \rightarrow CHAR LIST LIST$
MkReference	$CHAR LIST LIST \rightarrow \mathbb{N} \rightarrow Reference$
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	<i>Class</i> → <i>BOOL</i> → <i>BOOL</i> → <i>BOOL</i> → <i>BOOL</i> → <i>BOOL</i> → <i>Reference LIST</i> → <i>ColCon</i>
ColConS	<i>ColCon SET</i>
R_data	<i>Row</i> → ($\mathbb{N} \times \text{Data}$) <i>SET</i>
R_exist	<i>Row</i> → <i>Class</i>
MkRow	<i>Class</i> → ($\mathbb{N} \times \text{Data}$) <i>SET</i> → <i>Row</i>
RowS	<i>Row SET</i>
TS_rows	<i>TableSpec</i> → <i>Row LIST</i>
TS_cons	<i>TableSpec</i> → ($\mathbb{N} \times \text{ColCon}$) <i>SET</i>
TS_colspecs	<i>TableSpec</i> → <i>ColSpec SET</i>
TS_maxRow	<i>TableSpec</i> → <i>Class</i>
TS_class	<i>TableSpec</i> → <i>Class</i>
MkTableSpec	<i>Class</i> → <i>Class</i> → <i>ColSpec SET</i> → ($\mathbb{N} \times \text{ColCon}$) <i>SET</i> → <i>Row LIST</i> → <i>TableSpec</i>
TableSpecS	<i>TableSpec SET</i>
Dir_class	<i>Directory</i> → <i>Class</i>
Dir_exist	<i>Directory</i> → <i>Class</i>
Dir_tables	<i>Directory</i> → (<i>CHAR LIST</i> × <i>TableSpec</i>) <i>SET</i>
MkDirectory	(<i>CHAR LIST</i> × <i>TableSpec</i>) <i>SET</i> → <i>Class</i> → <i>Class</i> → <i>Directory</i>
DirectoryS	<i>Directory SET</i>
IdeL	<i>CHAR LIST LIST SET</i>
StateS	(<i>CHAR LIST LIST</i> × <i>Directory</i>) <i>SET SET</i>
isState	(<i>CHAR LIST LIST</i> × <i>Directory</i>) <i>SET</i> → <i>BOOL</i>
isError	'a + 'Error → <i>BOOL</i>
isVal	'a + 'Error → <i>BOOL</i>
destError	'a + 'Error → 'Error
destVal	'a + 'Error → 'a
giveError	'Error → 'a + 'Error
giveVal	'a → 'a + 'Error
SelectEffect	<i>Data LIST LIST</i> → (<i>CHAR LIST LIST</i> × ($\mathbb{N} \times \text{Data}$) <i>SET LIST</i>) + (<i>CHAR LIST LIST</i> × \mathbb{N} <i>SET</i>) + (<i>CHAR LIST LIST</i> × (\mathbb{N} × ($\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data}$) <i>SET</i>) <i>SET</i>) + <i>Data LIST LIST</i>
UpdateEffect	<i>CHAR LIST LIST</i>

	$\times (\mathbb{N} \times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data}) \text{SET})$ SET $\rightarrow (\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$
DeleteEffect	$\text{CHAR LIST LIST} \times \mathbb{N} \text{SET}$ $\rightarrow (\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$
InsertEffect	$\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST}$ $\rightarrow (\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$
isSelect	$(\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$ $\rightarrow \text{BOOL}$
isUpdate	$(\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$ $\rightarrow \text{BOOL}$
isDelete	$(\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$

	$\rightarrow \text{BOOL}$
isInsert	$(\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$
	$\rightarrow \text{BOOL}$
destSelect	$(\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$
	$\rightarrow \text{Data LIST LIST}$
destUpdate	$(\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$
	$\rightarrow \text{CHAR LIST LIST}$
	$\times (\mathbb{N} \times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data}) \text{SET})$
	SET
destDelete	$(\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$
	$\rightarrow \text{CHAR LIST LIST} \times \mathbb{N} \text{SET}$
destInsert	$(\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$ $\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$ $\text{SET}) \text{SET})$ $+ \text{Data LIST LIST}$
	$\rightarrow \text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST}$
tabFromEffect	$(\text{CHAR LIST LIST} \times (\mathbb{N} \times \text{Data}) \text{SET LIST})$ $+ (\text{CHAR LIST LIST} \times \mathbb{N} \text{SET})$ $+ (\text{CHAR LIST LIST}$ $\times (\mathbb{N}$

$$\times (\mathbb{N} \times (\text{ValuedItem} + \text{ONE}) + \text{Class} + \text{Data})$$

$$\text{SET) SET)}$$

$$+ \text{Data LIST LIST}$$

$$\rightarrow \text{CHAR LIST LIST}$$

7.4 Aliases

null	<i>One : ONE</i>
monolean	<i>One : ONE</i>
boolean	<i>One : ONE</i>
chars	<i>One : ONE</i>
integer	<i>One : ONE</i>
floating	<i>One : ONE</i>
time	<i>One : ONE</i>
interval	<i>One : ONE</i>
class	<i>One : ONE</i>

7.5 Types

Int
Float
Time
Interval
Code
ValuedItem
Data
ColSpec
Reference
ColCon
Row
TableSpec
Directory
State

7.6 Type Abbreviations

Char	<i>CHAR</i>
String	<i>CHAR LIST</i>
Num	<i>N</i>
Bool	<i>BOOL</i>
Ide	<i>CHAR LIST</i>
Error	<i>N</i>
Worth	<i>N</i>
Val	<i>ONE</i>
	<i>+ BOOL</i>
	<i>+ CHAR LIST</i>
	<i>+ Int</i>

	+ <i>Float</i>
	+ <i>Time</i>
	+ <i>Interval</i>
	+ <i>Code</i>
	+ <i>Class</i>
	+ <i>ONE</i>
NullItem	<i>ONE</i>
Item	<i>ValuedItem</i> + <i>ONE</i>
Update	(<i>ValuedItem</i> + <i>ONE</i>) + <i>Class</i> + <i>Data</i>
Type	<i>ONE</i> + <i>ONE</i> + <i>ONE</i> + <i>ONE</i> + <i>ONE</i> + <i>ONE</i> + <i>ONE</i> + <i>ONE</i>
Errors	\mathbb{N} <i>LIST</i>
Tab	<i>CHAR LIST LIST</i>
StateR	(<i>CHAR LIST LIST</i> \times <i>Directory</i>) <i>SET</i>
Insert	<i>CHAR LIST LIST</i> \times ($\mathbb{N} \times$ <i>Data</i>) <i>SET LIST</i>
Delete	<i>CHAR LIST LIST</i> \times \mathbb{N} <i>SET</i>
UpDate	<i>CHAR LIST LIST</i> \times ($\mathbb{N} \times$ ($\mathbb{N} \times$ (<i>ValuedItem</i> + <i>ONE</i>) + <i>Class</i> + <i>Data</i>) <i>SET</i>) <i>SET</i>
Select	<i>Data LIST LIST</i>
Effect	(<i>CHAR LIST LIST</i> \times ($\mathbb{N} \times$ <i>Data</i>) <i>SET LIST</i>) + (<i>CHAR LIST LIST</i> \times \mathbb{N} <i>SET</i>) + (<i>CHAR LIST LIST</i> \times ($\mathbb{N} \times$ ($\mathbb{N} \times$ (<i>ValuedItem</i> + <i>ONE</i>) + <i>Class</i> + <i>Data</i>) <i>SET</i>) <i>SET</i>) + <i>Data LIST LIST</i>

7.7 Definitions

Num	$\vdash \text{Num} = \text{Universe}$
Ide	$\vdash \text{Ide} = \text{Universe}$
Int	$\vdash \exists f \bullet \text{TypeDefn} (\lambda x \bullet T) f$
MkInt	
I_mantissa	
I_exponent	$\vdash \forall t \ x1 \ x2$ • $I_mantissa (MkInt \ x1 \ x2) = x1$ $\wedge I_exponent (MkInt \ x1 \ x2) = x2$ $\wedge MkInt (I_mantissa \ t) (I_exponent \ t) = t$
Float	$\vdash \exists f \bullet \text{TypeDefn} (\lambda x \bullet T) f$
MkFloat	
F_mantissa	
F_exponent	$\vdash \forall t \ x1 \ x2$ • $F_mantissa (MkFloat \ x1 \ x2) = x1$ $\wedge F_exponent (MkFloat \ x1 \ x2) = x2$ $\wedge MkFloat (F_mantissa \ t) (F_exponent \ t) = t$
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

	<ul style="list-style-type: none"> • $ValuedItem\ Item\ v = InL\ v \wedge NullItem\ Item\ n = InR\ n$
Data	$\vdash \exists f \bullet TypeDefn\ (\lambda x \bullet T)\ f$
MkData	
Dat_class	
Dat_item	$\vdash \forall t\ x1\ x2$ <ul style="list-style-type: none"> • $Dat_class\ (MkData\ x1\ x2) = x1$ • $\wedge\ Dat_item\ (MkData\ x1\ x2) = x2$ • $\wedge\ MkData\ (Dat_class\ t)\ (Dat_item\ t) = t$
DataS	$\vdash DataS = Universe$
ItemUpdate	
ClassUpdate	
DataUpdate	$\vdash \forall i\ c\ d$ <ul style="list-style-type: none"> • $ItemUpdate\ i = InL\ i$ • $\wedge\ ClassUpdate\ c = InR\ (InL\ c)$ • $\wedge\ DataUpdate\ d = InR\ (InR\ d)$
isItem	
isClass	
isData	$\vdash \forall u$ <ul style="list-style-type: none"> • $(isItem\ u \Leftrightarrow (\exists i \bullet u = ItemUpdate\ i))$ • $\wedge\ (isClass\ u \Leftrightarrow (\exists c \bullet u = ClassUpdate\ c))$ • $\wedge\ (isData\ u \Leftrightarrow (\exists d \bullet u = DataUpdate\ d))$
destItem	
destClass	
destData	$\vdash ConstSpec$ $(\lambda (destItem',\ destClass',\ destData')$ <ul style="list-style-type: none"> • $\forall i\ c\ d$ <ul style="list-style-type: none"> • $destItem'\ (ItemUpdate\ i) = i$ • $\wedge\ destClass'\ (ClassUpdate\ c) = c$ • $\wedge\ destData'\ (DataUpdate\ d) = d$ $(destItem,\ destClass,\ destData)$
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	$\vdash \forall t\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9$ <ul style="list-style-type: none"> • $CS_ide\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x1$ • $\wedge\ CS_posn\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x2$ • $\wedge\ CS_dinaryType\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) = x3$ • $\wedge\ CS_sterlingType$

	$ \begin{aligned} & (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) \\ & =\ x4 \\ & \wedge\ (CS_nullType \\ & \quad (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) \\ & \quad \Leftrightarrow\ x5) \\ & \wedge\ CS_default \\ & \quad (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) \\ & \quad =\ x6 \\ & \wedge\ CS_consGroup \\ & \quad (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) \\ & \quad =\ x7 \\ & \wedge\ CS_min\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) \\ & \quad =\ x8 \\ & \wedge\ CS_max\ (MkColSpec\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9) \\ & \quad =\ x9 \\ & \wedge\ MkColSpec \\ & \quad (CS_ide\ t) \\ & \quad (CS_posn\ t) \\ & \quad (CS_dinaryType\ t) \\ & \quad (CS_sterlingType\ t) \\ & \quad (CS_nullType\ t) \\ & \quad (CS_default\ t) \\ & \quad (CS_consGroup\ t) \\ & \quad (CS_min\ t) \\ & \quad (CS_max\ t) \\ & \quad =\ t \end{aligned} $
Reference	$\vdash \exists f \bullet TypeDefn\ (\lambda x \bullet T)\ f$
MkReference	
R_table	
R_group	$\vdash \forall t\ x1\ x2$ <ul style="list-style-type: none"> • $R_table\ (MkReference\ x1\ x2) = x1$ $\wedge\ R_group\ (MkReference\ x1\ x2) = x2$ $\wedge\ MkReference\ (R_table\ t)\ (R_group\ t) = t$
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	$\vdash \forall t\ x1\ x2\ x3\ x4\ x5\ x6\ x7$ <ul style="list-style-type: none"> • $CC_exist\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) = x1$ $\wedge\ (CC_uniform\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7)$ $\Leftrightarrow\ x2)$ $\wedge\ (CC_unique\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7)$ $\Leftrightarrow\ x3)$

	$\begin{aligned} & \wedge (CC_classLimited \\ & \quad (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\ & \quad \Leftrightarrow\ x4) \\ & \wedge (CC_primary\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\ & \quad \Leftrightarrow\ x5) \\ & \wedge (CC_secondary\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\ & \quad \Leftrightarrow\ x6) \\ & \wedge\ CC_referential\ (MkColCon\ x1\ x2\ x3\ x4\ x5\ x6\ x7) \\ & \quad =\ x7 \\ & \wedge\ MkColCon \\ & \quad (CC_exist\ t) \\ & \quad (CC_uniform\ t) \\ & \quad (CC_unique\ t) \\ & \quad (CC_classLimited\ t) \\ & \quad (CC_primary\ t) \\ & \quad (CC_secondary\ t) \\ & \quad (CC_referential\ t) \\ & \quad =\ t \end{aligned}$
ColConS	$\vdash\ ColConS = Universe$
Row	$\vdash\ \exists f \bullet TypeDefn\ (\lambda\ x \bullet T)\ f$
MkRow	
R_exist	
R_data	$\begin{aligned} & \vdash\ \forall\ t\ x1\ x2 \\ & \bullet\ R_exist\ (MkRow\ x1\ x2) = x1 \\ & \quad \wedge\ R_data\ (MkRow\ x1\ x2) = x2 \\ & \quad \wedge\ MkRow\ (R_exist\ t)\ (R_data\ t) = t \end{aligned}$
RowS	$\vdash\ RowS = \{r \mid let\ data = R_data\ r\ in\ data \in Num \leftrightarrow DataS\}$
TableSpec	$\vdash\ \exists f \bullet TypeDefn\ (\lambda\ x \bullet T)\ f$
MkTableSpec	
TS_class	
TS_maxRow	
TS_colspecs	
TS_cons	
TS_rows	$\begin{aligned} & \vdash\ \forall\ t\ x1\ x2\ x3\ x4\ x5 \\ & \bullet\ TS_class\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x1 \\ & \quad \wedge\ TS_maxRow\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x2 \\ & \quad \wedge\ TS_colspecs\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x3 \\ & \quad \wedge\ TS_cons\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x4 \\ & \quad \wedge\ TS_rows\ (MkTableSpec\ x1\ x2\ x3\ x4\ x5) = x5 \\ & \quad \wedge\ MkTableSpec \\ & \quad \quad (TS_class\ t) \\ & \quad \quad (TS_maxRow\ t) \\ & \quad \quad (TS_colspecs\ t) \\ & \quad \quad (TS_cons\ t) \\ & \quad \quad (TS_rows\ t) \\ & \quad =\ t \end{aligned}$
TableSpecS	$\vdash\ TableSpecS = \{ts$

	$ let\ cns = TS_cons\ ts\ and\ rows = TS_rows\ ts$ $in\ cns \in Num \mapsto ColConS$ $\wedge (\forall r \bullet r \in_l rows \Rightarrow r \in RowS)\}$
Directory	$\vdash \exists f \bullet TypeDefn (\lambda x \bullet T) f$
MkDirectory	
Dir_tables	
Dir_exist	
Dir_class	$\vdash \forall t\ x1\ x2\ x3$ <ul style="list-style-type: none"> • $Dir_tables (MkDirectory\ x1\ x2\ x3) = x1$ $\wedge Dir_exist (MkDirectory\ x1\ x2\ x3) = x2$ $\wedge Dir_class (MkDirectory\ x1\ x2\ x3) = x3$ $\wedge MkDirectory$ <ul style="list-style-type: none"> $(Dir_tables\ t)$ $(Dir_exist\ t)$ $(Dir_class\ t)$ $= t$
DirectoryS	$\vdash DirectoryS = \{d Dir_tables\ d \in Ide \mapsto TableSpecS\}$
IdeL	$\vdash IdeL = Universe$
StateS	$\vdash StateS = IdeL \mapsto DirectoryS$
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	$\vdash ConstSpec$ <ul style="list-style-type: none"> $(\lambda$ <ul style="list-style-type: none"> $(giveVal', giveError', destVal', destError',$ $isVal', isError')$ • $\forall v\ e\ ve$ <ul style="list-style-type: none"> • $giveVal'\ v = InL\ v$ $\wedge giveError'\ e = InR\ e$ $\wedge destVal'\ (giveVal'\ v) = v$ $\wedge destError'\ (giveError'\ e) = e$ $\wedge (isVal'\ ve \Leftrightarrow (\exists v_1 \bullet ve = giveVal'\ v_1))$ $\wedge (isError'\ ve$ <ul style="list-style-type: none"> $\Leftrightarrow (\exists e_1 \bullet ve = giveError'\ e_1))$ $(giveVal, giveError, destVal, destError, isVal,$ $isError)$
InsertEffect	
DeleteEffect	
UpdateEffect	
SelectEffect	$\vdash \forall i\ d\ u\ s$ <ul style="list-style-type: none"> • $InsertEffect\ i = InL\ i$ $\wedge DeleteEffect\ d = InR\ (InL\ d)$ $\wedge UpdateEffect\ u = InR\ (InR\ (InL\ u))$

$$\wedge \text{SelectEffect } s = \text{InR } (\text{InR } (\text{InR } s))$$

isInsert
isDelete
isUpdate
isSelect

$$\begin{aligned} &\vdash \forall e \\ &\bullet (\text{isInsert } e \Leftrightarrow (\exists i \bullet e = \text{InsertEffect } i)) \\ &\quad \wedge (\text{isDelete } e \Leftrightarrow (\exists d \bullet e = \text{DeleteEffect } d)) \\ &\quad \wedge (\text{isUpdate } e \Leftrightarrow (\exists u \bullet e = \text{UpdateEffect } u)) \\ &\quad \wedge (\text{isSelect } e \Leftrightarrow (\exists s \bullet e = \text{SelectEffect } s)) \end{aligned}$$

destInsert
destDelete
destUpdate
destSelect

$$\begin{aligned} &\vdash \text{ConstSpec} \\ &(\lambda \\ &\quad (\text{destInsert}', \text{destDelete}', \text{destUpdate}', \\ &\quad \quad \text{destSelect}') \\ &\bullet \forall i \ d \ u \ s \\ &\bullet \text{destInsert}' (\text{InsertEffect } i) = i \\ &\quad \wedge \text{destDelete}' (\text{DeleteEffect } d) = d \\ &\quad \wedge \text{destUpdate}' (\text{UpdateEffect } u) = u \\ &\quad \wedge \text{destSelect}' (\text{SelectEffect } s) = s) \\ &(\text{destInsert}, \text{destDelete}, \text{destUpdate}, \text{destSelect}) \end{aligned}$$

tabFromEffect

$$\begin{aligned} &\vdash \text{ConstSpec} \\ &(\lambda \text{tabFromEffect}' \\ &\bullet \forall i \ d \ u \\ &\bullet \text{tabFromEffect}' (\text{InsertEffect } i) = \text{Fst } i \\ &\quad \wedge \text{tabFromEffect}' (\text{DeleteEffect } d) = \text{Fst } d \\ &\quad \wedge \text{tabFromEffect}' (\text{UpdateEffect } u) = \text{Fst } u) \\ &\text{tabFromEffect} \end{aligned}$$

7.8 Theorems

dominates_consistent
lattice_bottom_consistent
lattice_top_consistent
lub_consistent
glb_consistent

$$\begin{aligned} &\vdash \text{Consistent} \\ &(\lambda \\ &\quad (\text{dominates}', \text{lattice_bottom}', \text{lattice_top}', \\ &\quad \quad \text{lub}', \text{glb}') \\ &\bullet \forall x \ y \ z \\ &\bullet \text{dominates}' x \ x \\ &\quad \wedge (\text{dominates}' x \ y \wedge \text{dominates}' y \ x \Rightarrow x = y) \\ &\quad \wedge (\text{dominates}' x \ y \wedge \text{dominates}' y \ z \\ &\quad \quad \Rightarrow \text{dominates}' x \ z) \\ &\quad \wedge (\forall x \bullet \text{dominates}' x \ \text{lattice_bottom}') \end{aligned}$$

$$\begin{aligned} & \wedge (\forall x \bullet \text{dominates}' \text{ lattice_top}' x) \\ & \wedge (\forall x y z \\ & \bullet \text{dominates}' (\text{lub}' x y) x \\ & \quad \wedge \text{dominates}' (\text{lub}' x y) y \\ & \quad \wedge (\text{dominates}' z x \wedge \text{dominates}' z y \\ & \quad \Rightarrow \text{dominates}' z (\text{lub}' x y))) \\ & \wedge (\forall x y z \\ & \bullet \text{dominates}' x (\text{glb}' x y) \\ & \quad \wedge \text{dominates}' y (\text{glb}' x y) \\ & \quad \wedge (\text{dominates}' x z \wedge \text{dominates}' y z \\ & \quad \Rightarrow \text{dominates}' (\text{glb}' x y) z))) \end{aligned}$$

8 INDEX

<i>accessDenied</i>	7	<i>dinary</i>	8
<i>ambiguousColumn</i>	7	<i>DirectoryS</i>	13
<i>ambiguousEvaluate</i>	7	<i>Directory</i>	12
<i>ambiguousHaving</i>	7	<i>Dir_class</i>	12
<i>ambiguousUpdate</i>	7	<i>Dir_exist</i>	12
<i>boolean</i>	10	<i>Dir_tables</i>	12
<i>BoolVal</i>	8	<i>dominates_def</i>	19
<i>Bool</i>	5	<i>downGrade</i>	7
<i>CC_classLimited</i>	11	<i>Effect</i>	15
<i>CC_exist</i>	11	<i>Errors</i>	11
<i>CC_primary</i>	11	<i>Error</i>	7
<i>CC_referential</i>	11	<i>error</i>	7
<i>CC_secondary</i>	11	<i>ExceptionVal</i>	8
<i>CC_uniform</i>	11	<i>fef004</i>	5
<i>CC_unique</i>	11	<i>fieldClassOutOfRange</i>	7
<i>chars</i>	10	<i>floating</i>	10
<i>Char</i>	5	<i>FloatVal</i>	8
<i>classChange</i>	7	<i>Float</i>	6
<i>ClassUpdate</i>	10	<i>F_exponent</i>	6
<i>ClassVal</i>	8	<i>F_mantissa</i>	6
<i>class</i>	10	<i>giveError</i>	14
<i>CodeVal</i>	8	<i>giveVal</i>	14
<i>Code</i>	6	<i>IdeL</i>	13
<i>ColConS</i>	12	<i>Ide</i>	5
<i>ColCon</i>	11	<i>InsertEffect</i>	15
<i>ColSpec</i>	11	<i>Insert</i>	14
<i>CS_consGroup</i>	11	<i>integer</i>	10
<i>CS_default</i>	11	<i>IntervalVal</i>	8
<i>CS_dinaryType</i>	11	<i>Interval</i>	6
<i>CS_ide</i>	11	<i>interval</i>	10
<i>CS_max</i>	11	<i>IntVal</i>	8
<i>CS_min</i>	11	<i>Int</i>	6
<i>CS_nullType</i>	11	<i>isClass</i>	10
<i>CS_posn</i>	11	<i>isData</i>	10
<i>CS_sterlingType</i>	11	<i>isDelete</i>	15
<i>DataS</i>	9	<i>isError</i>	14
<i>DataUpdate</i>	10	<i>isInsert</i>	15
<i>Data</i>	9	<i>isItem</i>	10
<i>Dat_class</i>	9	<i>isSelect</i>	15
<i>Dat_item</i>	9	<i>isState</i>	13
<i>DeleteEffect</i>	15	<i>isUpdate</i>	15
<i>Delete</i>	14	<i>isVal</i>	14
<i>destClass</i>	10	<i>ItemUpdate</i>	10
<i>destData</i>	10	<i>Item</i>	9
<i>destDelete</i>	16	<i>I_exponent</i>	6
<i>destError</i>	14	<i>I_mantissa</i>	6
<i>destInsert</i>	16	<i>mayNotBeComplete</i>	7
<i>destItem</i>	10	<i>monolean</i>	10
<i>destSelect</i>	16	<i>noNulls</i>	7
<i>destUpdate</i>	16	<i>nonUniformValues</i>	7
<i>destVal</i>	14	<i>nonUniqueValues</i>	7

noSuchColumn 7
noSuchDirectory 7
noSuchTable 7
notCleared 7
NullItemItem 9
NullItem 9
nullValue 7
null 9
Num 5
priceless 8
Reference 11
rowClassTooLow 7
RowS 12
Row 12
R_data 12
R_exist 12
R_group 11
R_table 11
SelectEffect 15
Select 15
StateR 13
StateS 13
sterling 8
StringVal 8
String 5
tabFromEffect 16
TableSpecS 12
TableSpec 12
Tab 11
TimeVal 8
Time 6
time 10
tooTall 7
tooWide 7
TS_class 12
TS_colspecs 12
TS_cons 12
TS_maxRow 12
TS_rows 12
Type 11
underClassified 7
UpdateEffect 15
Update 9
UpDate 15
ValuedItemItem 9
ValuedItem 9
Val 8
VI_val 9
VI_worth 9
VoidVal 8
worthless 8
Worth 8
wrongType 7