1. Descent Maps (II)

**Notation.** When \( J : T \leq T' \) is a theory inclusion, we denote a descent map \((J, \delta) : (T', G) \rightarrow (T, F)\) by \( T' \downarrow J \rightarrow T \).

In particular when \( T = T' \) and \( J = \text{id}_{\text{sig}(T)} \), we denote it by \((T, G) \leftrightarrow (T, F)\). Note that in this latter case \((1, \text{sig}(T))\) is both an ascent and a descent map.

We denote a general descent map \((H, \delta) : (T', G) \rightarrow (T, F)\) thus: \((T', G) \xrightarrow{\delta} (T, F)\).

Note that descent maps compose in the obvious way:

\[
(T', G) \xrightarrow{\delta} (T, F) \xrightarrow{\eta} (T'', K) \Rightarrow G \xrightarrow{H} \Rightarrow (T, G) = \delta \cdot \eta
\]

So they form a category \( \text{Descent} \).

1.1 Simple Descent Maps

A useful class of descent maps have the nice property of giving a list of information about \( \text{coalescing-stability} \) is achieved at the level of models. They are what I call
simple descent maps

**Definition.** A descent map \( (T', G) \xleftarrow{f} (T, \mathcal{F}) \) is called **simple** iff

1. \( \mathcal{H} \) is expansive or literally expansive
2. \( \forall \varphi \in \mathcal{G}, \forall M' \in \text{mod}(T') \)

\( \varphi \) satisfiable in \( M' \) \iff \( \exists \psi \in \mathcal{F}(\varphi) \) satisfiable in \( M'|_{M'} \).

Of course, that (1) - (2) always define a descent map but to be proved.

**Lemma.** Any \( (T', G) \xleftarrow{f} (T, \mathcal{F}) \) as above satisfying (1) - (2)

is a descent map. (for literally expansive: \( T' = (\Sigma', \mathcal{B}), T = (\Sigma, \mathcal{A}) \))

\( (m' \in \mathcal{B}) \)

**Proof.**

1. \( \forall \varphi \in \mathcal{G}, \forall \varphi \text{- satisfiable} \iff \exists M' \in \text{mod}(T') \) s.t. \( \varphi \) sat. in \( M' \)

\( \iff \exists (m' \in \mathcal{C}(T')) \) s.t. \( \varphi \) sat. in \( M'|_{M'} \)

2. \( \forall \varphi \in \mathcal{G}, \forall \varphi \text{- satisfiable} \iff \exists M' \in \text{mod}(T') \) s.t. \( \exists \psi \) sat. in \( M' \)

\( \iff (\text{by } \mathcal{H} \text{ expansive}) \exists (m' \in \mathcal{B}^\omega) \iff \varphi \) sat. in \( M' \)

\( \iff \exists M' \in \text{mod}(T') \) s.t. \( \varphi \) sat. in \( M' \iff \varphi \text{- satisfiable}. \Box \)

Note that (2) is enough to prove the lemma since it is a chain of equivalences, so (1) is not really needed and can be dropped.

**Examples (VarSat Revisted).** Note that many of the descent maps encountered in the VarSat paper are simple descent maps.

Let us begin by the variant satisfiability algorithm itself.

It is associated to a literally expansive theory extension

\[ (\Sigma, \{ T_{\Sigma/E_\mathcal{B}} \}) \geq (\Sigma, \{ T_{\Sigma/E_\mathcal{B} \cup \mathcal{D}} \}) \]
what \((\Sigma, EVB)\) has an FVP decomposition \((\Sigma, B, E)\)

provides a constructor decomposition \((\Sigma, B_0, E_0)\), which is OS-compact.

The descent map is the composition of simple descent maps:

\[
\text{df, conj, QF} \\
(\Sigma, \{\Sigma_{E/EVB}\}, QF) \overset{1_{\Sigma}}{\leftrightarrow} (\Sigma, \{\Sigma_{E/EVB}\}, \Lambda \cup (\Sigma)) \overset{\text{ctor-va-umf}}{\leftrightarrow} (\Sigma, \{\Sigma_{E/EVB}\}, \Lambda \cup \text{At}(\Sigma)).
\]

\[
\text{ctor-vay} \\
\supseteq (\Sigma, \{\Sigma_{E/EVB}\}, \Lambda \cup \text{At}(\Sigma)).
\]

For another example, we can consider the descent from \(Z_+\) to \(N_+\) in Appendix D of Varsit attached to the literally expansive extension:

\[
Z_+ = \left(\Sigma_{\text{INT}}, \{Z_{+, -}\}\right) \supseteq \left(\Sigma_{\text{NAT}}, \{N_0, +\}\right) = N_+
\]

which is accomplished by the chain of simple descent maps

\[
Z_+ \overset{\text{QF}}{\leftrightarrow} Z_+ \overset{\text{ctor-vay}}{\supseteq} N_+.
\]

where the classes of formulae involved in the (several steps) into which

\(\text{ctor-vay}\) and \(\text{QF}\) are decomposed is left implicit.

Note that in all these examples, these simple descent maps are effective.

For example in variant satisfiability, for \((\Sigma, \{\Sigma_{E/EVB}\})\)

\(\eta \in QF(\Sigma)\) was first mapped by df, conj to a set \(\{\forall C_i \land \forall D_i \mid i \in I\}\)

of conjunction of equations, \(C_i\) and disequations, \(D_i\), \(i \in I\), then

by cto-rva-umf each \(\forall C_i \land \forall D_i\) was mapped to a set of

congruences \(\{\forall D_i \land \exists \bar{x} \in \text{true}(C_i)\}\), and then each
\[ \text{A} \text{D; } \lambda \text{ was wapped to a set of ctn-variants} \]

\[ \mathcal{L} \text{A}_{\lambda d} \subseteq \mathcal{L} \text{E} \text{B} \text{. Then, choosing some } (\lambda D', \beta) \text{ in this} \]

\[ \text{and a grand substitution for its finite part, we checked that} \]

\[(\lambda D')! \text{ was } B_{\beta} \text{- consistent. From this we can effectively compute a grand substitution } \zeta \text{ such that} \]

\[ (\mathcal{L} x / \mathcal{E} B, \zeta) \models (\lambda D')! \]

\[ \text{But then we can construct the satisfying assignment} \]

\[ (\mathcal{L} x / \mathcal{E} B, \alpha \beta \gamma) \models \gamma \]

\[ \text{as desired.} \]

This can render part of process to provide a checkable certificate that the satisfiability of \( \gamma \) in \((\mathcal{L} x / \mathcal{E} B, \alpha \beta \gamma) \) inferred by this sequence of steps is correct.

1.2 Simple Descent Maps are Closed Under Unions

As done in §2 of Lecture 24 (4/6/17), we can analogously show that descent maps are not only closed under sequential composition in the category \text{Descent}, but also under a kind of "parallel composition" by union; provided they are simple and obey some extra conditions. Specifically, assume \( (T'_i, G'_i) \xrightarrow{H_i} (T_i, F_i) \) simple descent maps \((1 \leq i \leq 2)\) such that:
(1) \( \Sigma(T_2) \) and \( \Sigma(T_2') \) (resp. \( \Sigma(T_1) \) and \( \Sigma(T_1') \)) are optimally interdefinable in \( \Sigma_0 \) (resp. \( \Sigma_0' \)) where \( \Sigma_0 \) and \( \Sigma_0' \) consist only of sorts (no opens, or pred. symbol) and \( H_1 \) and \( H_2 \) agree on \( \Sigma_0 \) and induce a surjective restriction to \( H_0 : \Sigma_0 \rightarrow \Sigma_0' \). Then,

**Theorem** Under the above conditions,

\[
(T_{1'} \cup T_{2'}, G_1 \land G_2) \overset{S_1 \land S_2}{\rightarrow} (T_{1'} \cup T_{2'}, F_1 \land F_2)
\]

\[
\overset{H_1 \cup H_2}{\leftarrow}
\]

where \( S_1 \land S_2 (\psi_1 \land \psi_2) = \{ \psi_i \land \psi_i' \mid \psi_i \in \delta_i(\psi_i), 1 \leq i \leq 2 \} \)

is a simple descent map.

**Proof.** The proof that \( H_1 \cup H_2 \) is expansive is exactly as in §2, Lecture 24, proof for descent map with \( H_1 \cup H_2 \) expansive.

The rest of the proof goes as follows:

\[ A \psi_1 \land \psi_2 \in G_1 \land G_2 \quad A [m_1', m_2'] \in \text{mod}(T_{1'} \cup T_{2'}) \]

\[ \psi_1 \land \psi_2 \text{ sat. in } [m_1', m_2'] \iff \psi_1 \text{ sat. in } m_1' \text{ and } \psi_2 \text{ sat. in } m_2' \iff \exists \psi_1 \in \delta(m_1) \text{ s.t. } \psi_1 \text{ sat. in } m_1' \land \exists \psi_2 \in \delta(m_2) \text{ s.t. } \psi_2 \text{ sat. in } m_2' \]

\[ \iff \exists \psi_1 \land \psi_2 \in \delta(S_2(\psi_1 \land \psi_2)) \text{ s.t. } \psi_1 \land \psi_2 \text{ sat. in } [m_1', m_2'] \text{ mod. } (H_1 \cup H_2) \]

The practical interest of this theorem is that, as we shall see in a few lectures, under the assumption of \( T_1, T_2 \) key relatively infinite, we can combine the descent map \((H_1 \cup H_2, S_1 \land S_2)\) with the Nelson-Oppen procedure to obtain a decision procedure for \( G_1 \land G_2 - T_{1'} \cup T_{2'} \) satisfiability if \( F_i - T_i \) sat. is decidable, \( 1 \leq i \leq 2 \).
2. Non-Simple Descent Maps: A Decision Procedure for Arrays & Tables

The Problem. Arrays and Tables are of course finitary data structures extensively used in computer science practice. In particular this means that they are computable data types. Therefore, they are, as we shall see, equationally axiomatizable as elements of initial algebras defined by convergent equations in membership Equational logic.

The problem is that the models of arrays and, a fortiori, of tables used in decision procedures:

1. are not computable data types at all, and
2. do not correctly model the array and table computable algebras they are supposed to make decidable.

One could live with deficiencies (1) and (2) if at least the computable algebras of arrays and tables could be embedded in the non-computable models. Since then one could at least try to show that a satisfying assignment for the non-computable model yields somehow a conversion map one for real arrays and tables. But this fails to be the case rather badly because, for example:

(a) For N1 the set of indices, and 1 = 0, 3 the set of values, the model of arrays used, namely the functor set \([\mathbb{N} \times 1]\), has a single array.

Instead, since an array of this nature is exactly a finite partial function, i.e., a finite set of
the form \( A = \{(i_1, a_1), \ldots, (i_k, a_k)\} \) with \( i_j \in \mathbb{N}, \forall 1 \leq j \leq k \).

ij \neq i_j \text{ if } j \neq j'. \text{ There is no hope whatsoever that satisifying of a QF formula in } [\mathbb{N} \to 1] \text{ and in the algebra of such arrays could agree.}

In fact, \([\mathbb{N} \to 1]\) cannot satisfy any inequality \( A \neq A' \) for \( A, A' \) variables of sort \( Array \). That is, the sentence \((\forall A: Array, A': Array) A = A' \) is a theorem of \([\mathbb{N} \to 1]\), but this is of course false for the countably infinite \( \mathbb{N} \) different array with index set \( \mathbb{N} \) and values \([1]\).

(b) For \( [\mathbb{N}] \) the set of indices and \( 2 = \{0, 1\} \) the set of values, we have \([\mathbb{N} \to 2] \approx \mathcal{P}(\mathbb{N})\), a non-computable data type that has the power of the continuum.

But since there are all total functions, none of them correspond to real arrays, which are finite partial functions of the form:

\[ A = \{(i_1, b_1), \ldots, (i_k, b_k)\} \quad i_j \in \mathbb{N}, b_j \in 2, 1 \leq j \leq k, \]

and \( ij \neq i_j \text{ if } j \neq j' \). In particular, the fact that for such an array its value for an index \( i \) such that \( i \neq i_j, 1 \leq j \leq k \) is undefined cannot be modeled at all.

Therefore, the first order of business is to define a precise mathematical model of arrays and tables as computable algebraic data types using initial algebra semantics. The following handy specification closer so as parameterized data type:
*** theory of equality

fth EQ is pr BOOL.
sort Elt.
op _~_ : Elt Elt -> Bool. ** equality predicate

case x = y : Elt.

endfth

*** binary relations

fmod REL{X :: EQ, Y :: TRIV} is
  sorts Pair{X,Y} Rel{X,Y} Set{X}.
  subsort Pair{X,Y} < Rel{X,Y}.
  subsort X$Elt < Set{X}.

  op [~_~_] : X$Elt Y$Elt -> Pair{X,Y} [ctor]. ** ordered pairs
  op null : -> Rel{X,Y} [ctor]. ** empty relation
  op _~_ : Set{X} [ctor]. ** empty set
  op _~_ : Rel{X,Y} Rel{X,Y} -> Rel{X,Y} [assoc comm id: null]. ** union
  op _~_ : Set{X} Set{X} -> Set{X} [assoc comm id: mt]. ** union

  vars a a' : X$Elt. var b : Y$Elt. var S : Set{X}. var R : Rel{X,Y}.

  eq [a,b],[a,b] = [a,b]. ** idempotency
  eq a,a = a. ** idempotency

  op _in_ : X$Elt Set{X} -> Bool. ** set membership

  eq a in mt = false.
  eq a in (a,S) = true.
  eq a in (a',S) = (a ~ a') or (a in S).

  op dom : Rel{X,Y} -> Set{X}. ** domain of a relation

  eq dom(null) = mt.
  eq dom([a,b],R) = a,dom(R).
  endf

*** extensional arrays

fmod EXT-ARRAY{I :: EQ, V :: TRIV} is
  pr REL{I,V}.
  sorts Array{I,V} Val{V}.
  subsorts Pair{I,V} < Array{I,V} < Rel{I,V}.
  subsort V$Elt < Val{V}.

  op no-val : -> Val{V} [ctor]. ** error value
  op null : -> Array{I,V} [ctor]. ** empty array
var i : ISel.  vars v v' : VSel.  var A : Array[I,V].
cmb [(i,v),A] : Array[I,V] if i in dom(A) = false.

op _[.] : Array[I,V] ISel -> Val?{V}.  *** read
ceq ((i,v),A)[i] = v if ((i,v),A) : Array[I,V].
ceq A[i] = no-val if i in dom(A) = false.

op _[-=] : Array[I,V] ISel VSel -> Array[I,V].  *** write
ceq ((i,v),A)[i := v'] = ((i,v'),A) if ((i,v),A) : Array[I,V].
ceq A[i := v'] = ((i,v'),A) if i in dom(A) = false.
endfm

*** tables are arrays where an entry for an index/key can be deleted

fmod TABLE[I = EQ, V = TRIV] is
pr EXT-ARRAY[I,V]*(sort Array[I,V] to Table[I,V]).

vars i j : ISel.  vars v : VSel.  var T : Table[I,V].

op del : ISel Table[I,V] -> Table[I,V].  *** deletes a table entry
eq del(i,null) = null.
ceq del(i,[(i,v),T]) = T if ((i,v),T) : Table[I,V].
ceq del(i,[(i,v),T]) = [(i,v),del(i,T)]
   if (i ~ j) = false \ ([(j,v),T) : Table[I,V].
endfm

*** LOOKUP is a theory with no axioms from which a highly simplified view,
*** i.e., a reduct of TABLE, can be obtained by the parameterized view below.
*** Satisfiability of QF formulas in LOOKUP is decidable by many-sorted
*** Congruence Closure.

fth LOOKUP is
sorts Index Value? Readable.

op no-val : -> Value?.
op _[.] : Readable Index -> Value?.
endfth

*** (The following parameterized view is only definable in Full Maude
*** (in parentheses). Since operators have the same names and matching sorts,
*** they are mapped by default.

(view Table[I = EQ, V = TRIV] from LOOKUP to TABLE[I,V] is
  sort Index to ISel,
  sort Value? to Val?{V},
  sort Readable to Table[I,V].
endv)
)
Recall from Lecture 19, p. 11 that given a parameterized data type associated to a theory inclusion $J: P \leq B$ between the parameter theory $P$ and the "body" theory $B$, say with $\text{sig}(P) = \Sigma_P$ and $\text{sig}(B) = \Sigma_B$, where $P$ and $B$ are theories in ordered sorted Horn logic with the theory of the parameterized data type associated to $J$ is the semantic theory $(\Sigma_B, I F_J)$, where, by definition,

$$I F_J = \{ I F J(M) \mid M \in \text{mod}(P) \}$$

where $I F_J$ is the left adjoint to the reduct functor $-|_{\Sigma_P}: \text{mod}(B) \to \text{mod}(P)$.

For the above parameterized data types $\text{EXT-ARRAY}\{I::\text{EQ}, V::\text{TRIV}\}$ and $\text{TABLE}\{I::\text{EQ}, V::\text{TRIV}\}$ the parameter theory $P$ is the same, namely,

$$P = \text{EQ} \ast (\text{Elt} \to I \# \text{Elt}) \cup \text{TRIV} \ast (\text{Elt} \to V \# \text{Elt})$$

where $\ast (A \to B)$ is a sort renaming of sort name $A$ to sort name $B$. Then $B$ in each case is the theory defined above for $\text{EXT-ARRAY}$ or $\text{TABLE}$, plus the axioms of $\text{EQ}$ already contained in the subtheory $P$. Let $A$ denote $IF_J$ for the
inclusion $\mathcal{P} \subseteq \text{EXT-ARRAY}$, and $\mathcal{P} = \{F_{ij}\}$ for the inclusion $\mathcal{P} \subseteq \text{TABLE}$. Then the semantic theories

$$(\Sigma_A, \mathcal{A}) \quad \text{and} \quad (\Sigma_T, \mathcal{T})$$

where $\Sigma_A$ and $\Sigma_T$ are the signatures of $\text{EXT-ARRAY}$ and $\text{TABLE}$ are exactly the ones associated by free algebra semantics to the keywords found in the end for encoding the $\text{EXT-ARRAY}$ and $\text{TABLE}$ declarations.

In practice, however, we want to hide some of the functionality in $\Sigma_A$ and $\Sigma_T$ and give to users an interface to use these data types. For example, somebody can use an array or a table, for no need for the polymorphic function

$$\text{dom} : \text{Rel}\{I, V\} \rightarrow \text{Set}\{I\}$$

which belongs to both $\Sigma_A$ and $\Sigma_T$. The usual interfaces are:

$$\Sigma_{A_0} :$$

```
IBElk

noval

Val?\{V\}

V#$Elk

Array\{I, V\}

-[-:=-]
```

where $\mathcal{P} \subseteq \\text{EXT-ARRAY}$, and $\mathcal{P} = \{F_{ij}\}$ for the inclusion $\mathcal{P} \subseteq \text{TABLE}$. Then the semantic theories

$$(\Sigma_A, \mathcal{A}) \quad \text{and} \quad (\Sigma_T, \mathcal{T})$$

where $\Sigma_A$ and $\Sigma_T$ are the signatures of $\text{EXT-ARRAY}$ and $\text{TABLE}$ are exactly the ones associated by free algebra semantics to the keywords found in the end for encoding the $\text{EXT-ARRAY}$ and $\text{TABLE}$ declarations.

In practice, however, we want to hide some of the functionality in $\Sigma_A$ and $\Sigma_T$ and give to users an interface to use these data types. For example, somebody can use an array or a table, for no need for the polymorphic function

$$\text{dom} : \text{Rel}\{I, V\} \rightarrow \text{Set}\{I\}$$

which belongs to both $\Sigma_A$ and $\Sigma_T$. The usual interfaces are:

$$\Sigma_{A_0} :$$

```
IBElk

noval

Val\{V\}

V#$Elk

Array\{I, V\}

-[-:=-]
```
\[ \Sigma_{T_0} : \]

Therefore, the array and table data types in their common use are exactly described as the semantic theory:

\[ A = (\Sigma_{A_0}, A | A_{A_0}) \quad \text{and} \quad (\Sigma_{T_0}, T | T_{T_0}) = T \]

Note that both theories can be extended by adding to them a predicate:

\[ \text{def} : I \# \text{Elt} \quad \text{Array\{I, V\}} \]

resp. \[ \text{def} : I \# \text{Elt} \quad \text{Table\{I, V\}} \]

but such extensions are purely definitional extensions of the forms:

\[ \text{def}(x, A) \iff A[x] \neq \text{no-val} \]

\[ \text{def}(x, T) \iff T[x] \neq \text{no-val} \]

where \( x : I \# \text{Elt}, A : \text{Array\{I, V\}}, \) and \( T : \text{Table\{I, V\}} \)

therefore, there is no need to extend \( \Sigma_{A_0} \) or \( \Sigma_{T_0} \), since \( \text{def} \) is definable in their respective QF languages.
I will now show how we can use a non-simple descent map to obtain decision procedures for the theories $A$ and $T$. Specifically, for $(A, QF(\Sigma_{A_0}))$ and $(T, QF(\Sigma_{T_0}))$.

Since any such decision procedure for $(T, QF(\Sigma_{T_0}))$ will a fortiori give us a decision procedure for the more restricted language of $(A, QF(\Sigma_{A_0}))$, I only describe the descent map for $(T, QF(\Sigma_{T_0}))$. It is the composition:

\[
(T, QF(\Sigma_{T_0})) \xrightarrow{\text{load, conj, vals}} (T, \text{SLLit}(\Sigma_{T_0})) \xleftarrow{\text{Table}(I, V)} (\text{LOOKUP}, QF(\Sigma_{\text{LOOKUP}}))
\]

where \( \text{LOOKUP} \) in the theory \((\Sigma_{\text{LOOKUP}}, \emptyset)\) defined in 8.9, and \( \text{Table}(I, V) \) in the theory interpretation (view) also defined in 8.9, which obviously factors as:

\[
\text{LOOKUP} \xrightarrow{\text{Table}(I, V)} T \subseteq \text{TABLE}(I, V)
\]

Note that this descent map is not simple, since the reduct functor \(-\vert_{\text{Table}(I, V)} : \text{Mod}(T) \rightarrow \text{Mod}(\text{LOOKUP})\) is such that some models of \(\text{Mod}(\text{LOOKUP})\) cannot be expanded to models of \(\text{Mod}(T)\). For example the model \(M = (M, \neg m) \in \text{Mod}(\text{LOOKUP})\) with \(M_{\text{Index}} = \{a\}\),

\(M_{\text{value?}} = \{b, \text{no-val}\}\),

\(M_{\text{readable}} = \{x, z, z'\}\).
and no-val \( M = \text{no-val, } \) \( \phi \) \( \models \text{can not be expanded} \)

to a model \( M' \in \text{mod}(T) \) such that \( M' \models \text{Table[I,IV]} \)

Note that, since \( \text{LOOKUP} = (\Sigma, \emptyset) \), \( \text{LOOKUP-satisfiability} \)

of any \( \phi \in \text{QF}(\Sigma) \) is decidable by many-sorted

congruence closure. Therefore, all we need to do is to define

the function \( \text{a2l} : \text{SALit}(\Sigma_T) \rightarrow \text{QF}(\Sigma) \) and to

prove that \( \text{Table[I,IV], a2l} \) is indeed a decent map.

In this way we obtain a decision procedure for \( (T, \text{QF}(\Sigma_T)) \).

What does a formula \( \phi \in \text{SALit}(\Sigma_T) \) look like? It looks

exactly as follows, where I list negated atoms below for

readability:

\[
\phi = \\
\bigwedge x_i = x'_i \land y_{i,j} = y'_{i,j} \land z_{i,e} = z'_{i,e} \land \text{no-val} = y_p \land \text{no-val} = y_q \land z_q \models \xi_q \land \text{no-val} = y_x \land \text{no-val} = y_x' \land z_x \models z_x' \land x_i = y'_i
\]

where the \( x_i \) 's belong to a (finite) set \( X \) of variables of sort \( \text{Int} \), the \( y_{i,j} \) 's belong
to \( Y = \text{Int} \cup \text{Val(I,IV)} \), and the \( z_{i,e} \) 's to a set \( Z \) of variables of

sort \( \text{Table[I,IV]} \), and \( X, Y \) and \( Z \) only contain those variables

of sort \( \text{Table[I,IV]} \), and \( X, Y \) and \( Z \) only contain those variables

appearing in the above formula. The \( \text{a2l} \) function is defined as

\[
\text{a2l}(\phi) = \text{a2l}_1(\phi) \land \text{a2l}_2(\phi), \text{with } \text{a2l}_1 \text{ defined by:}
\]

\[
\text{a2l}_1(\pi \land \chi) = \text{a2l}_1(\pi) \land \text{a2l}_1(\chi)
\]

\[
\text{a2l}_1(\tau) = T
\]

where we assume \( \land \) associative and commutative with identity \( T \).
and where \( \text{lit} \) and \( C \) represent, respectively, a literal and a conjunction in \( \text{SAT}(\Sigma_{T_0}) \), and \( az_{\Sigma_{T_0}}(\text{lit}) \) is defined as follows:

1. \( \text{it is the identity on all literals of the forms} \)
   \( x = x', \ x \neq x', \ z = z', \ z \neq z', \ \text{no-val} = y, \ \text{no-val} \neq y, \)
   \( y = z[x], \) and \( y \neq z[x]. \)

   * \( \text{except that the sorts of the variables are renamed as follows:} \)
   \( x: \text{Int} \mapsto x: \text{Index}, \ y: \text{Val} \mapsto y: \text{Value}, \)
   \( y': \text{Val} \mapsto y': \text{Value}, \ z: \text{Table}[I,V] \mapsto z: \text{Readable} \)

   \( \text{Let} \ \tilde{x}, \tilde{y}, \tilde{y}', \text{and} \ \tilde{z} \ \text{denote the renamed versions of} \ x, y, y', \)
   \( y_v, y_{v'}, \text{and} \ z \ \text{according to the above renamings of variables.} \)

2. \( \text{We introduce a new set} \ X_z \ \text{of variables of sort Index}\)
   \( \text{such that} \ X \cap X_z = \emptyset, \) defined as follows:
   \[ X_z = \{ x_{z, z'} \mid (z, z') \in Z^2, z < z' \} \]

   where \( (Z, <) \) is a chosen total order on \( Z. \)

   \( \text{Intuitively} \ x_{z, z'}, \ \text{will be used as an extra index capable} \)
   \( \text{of distinguishing between interpretations of the variables} \ z \)
   \( \text{and} \ z' \ \text{where} \ z \text{and} \ z' \ \text{are interpreted as different elements} \)
   \( \text{of sort Readable.} \)

\( (*) \) To afford a simpler notation we assume without loss of generality that

the names of variables in \( y: \text{Val} \) and \( y': \text{Val} \) are disjoint, so for any

\( y: \text{Val} \) and \( y': \text{Val} \) we must have \( y: \text{Value} \neq y': \text{Value} \).
(3) $\text{azl}_4 (z = z' [x := y]) = (z [x] = y \land \bigwedge_{x \neq x'} z [x'] = z' [x'])$

$x' \in (\overline{x} - \{x\}) \cup X_z$

(3') $\text{azl}_4 (z \neq z' [x := y]) = (z [x] \neq y \lor \bigvee_{x' \in (\overline{x} - \{x\}) \cup X_z} z [x'] \neq z' [x'])$

(4) $\text{azl}_4 (z = \text{del} (x, z')) = (z [x] = \text{no-val} \land \bigwedge_{x \neq x'} z [x'] = z' [x'])$

$x' \in (\overline{x} - \{x\}) \cup X_z$

(4') $\text{azl}_4 (z \neq \text{del} (x, z')) = (z [x] \neq \text{no-val} \lor \bigvee_{x' \in (\overline{x} - \{x\}) \cup X_z} z [x'] \neq z' [x'])$

Finally, $\text{azl}_0 (\psi)$ is defined as follows:

\[
\text{azl}_0 (\psi) = \left( \bigwedge_{z, z' \in Z} z \neq z' \Rightarrow z [x, z'] \neq z' [x, z'] \right) \land \left( \bigwedge_{y \in Y, y \neq \text{no-val}} \right)
\]

That is, $\text{azl}_0 (\psi)$ intuitively can be understood as saying:

(a) if readble $z$ is different from readble $z'$, they can be distinguished by readly $x, z, z'$

(b) the interpretation of a variable $y$ of sort $Y$ can never be no-val.
Theorem \[ (T, S A L I x (\Sigma_{T_0})) \leftrightarrow (LOOKUP, QF(\Sigma_{LOOKUP})) \] can be descent.

Proof. For each \( \Psi \in S A L I x (\Sigma_{T_0}) \) we have to prove:

\[ \Psi \text{ T-satisfable } \iff a z l (\Psi) \text{ LOOKUP-satisfable} \]

\((\Rightarrow)\). It is enough to show that for any non-empty set \( I \) and \( V \) and any \( a \in [W \to T[I, V]] \) such that

\[ \Pi[I, V], a \models \Psi \text{ (where } W = vars(\Psi), \text{ and } \Pi[I, V] = (T[I, V], \Pi_{[I, V]}) \]

we have an \( \tilde{a} \in [\tilde{W} \to T[I, V]] \) such that

\[ \Pi[I, V], \tilde{a} \models a z l (\Psi), \text{ where } \tilde{W} = vars(azl(\Psi)). \]

We choose \( \tilde{a} \) as follows. Note that \( \tilde{W} = X \uplus Z \uplus Y_v \uplus \tilde{Y}_v \)

\( \uplus \tilde{Z} \), where the \( X, Y_v, \tilde{Y}_v \) and \( \tilde{Z} \) are in bijection correspondence with \( X, Y_v \in \text{Ev}, \tilde{Y}_v \in \text{Ev} \), and \( Z \), i.e., with \( W \).

\( \tilde{a} \) is then chosen on the renamed variables of \( W \) as the composition of the inverse renaming with \( a \). We just need to define \( \tilde{a} \) on the variables \( X_z \). We do so as follows:

(i) we choose some \( i_0 \in I \)

(ii) for each \( Z, Z' \in Z \) with \( Z < Z' \), we define

\[ \tilde{a}(X_{Z, Z'}) = \begin{cases} a(Z) & \text{if } a(Z) = a(Z') \text{ then } i_0 \text{ else } i_{Z, Z'} \end{cases} \]

where \( i_{Z, Z'} \in I \) is some chosen index such that

\[ a(Z)[i_{Z, Z'}] \neq a(Z')[i_{Z, Z'}] \]

\( \Pi[I, V] \), which always exists.
since $a(z)$ and $a(z')$ are finite partial functions from $I$ to $V$ such that $a(z) \neq a(z')$ so that there must be a pair $[i, z, z', v]$ in one of them but not in the other.

Since $a \in a\mathcal{L}(\mathcal{I})$ is a conjunction of formulas, to show that

$$\prod_{\mathcal{I}, \mathcal{V}} \models \exists \alpha \mathcal{L}(\mathcal{I})$$

we just need to show that each conjunct in $a \mathcal{L}(\mathcal{I})$ is satisfied. We reason by cases:

1. for literals in $\mathcal{I}$ that are mapped identically except for variable renaming this is trivially the case.

2. $a \mathcal{L}(\mathcal{I})$ is also trivially satisfied by the choice of the $i_{z, z'} = \alpha(x_{z, z'})$ and the fact that $a(y) \neq \alpha$-val for any $y \in \mathcal{V}$ get.

3. (we prove the case $(3')$ in the def. of $a \mathcal{L}^{-1}$, the case $(3)$ is similar)

   3.1 If $a(z) = a(z')$ and we reason by cases to see that $a \mathcal{L}_1 (z \neq z'[x := y])$ holds if $[i_{z, z'}]_{\prod_{\mathcal{I}, \mathcal{V}}} \neq a(i_{z, z'})$.

   3.2 If $a(z) \neq a(z')$ we must have $a(z)[i_{z, z'}]_{\prod_{\mathcal{I}, \mathcal{V}}} \neq a(i_{z, z'})$.

4. (we prove case $(4')$ and leave case $(4)$ to the reader.) We again reason by cases to see that $a \mathcal{L}_1 (z \neq \text{del}(x, z'))$ holds for $\alpha$ if $z \neq \text{del}(x, z')$ holds for $\alpha$. 
4.1 If \( a(z) = a(z') \) and \( a(z) \neq (\text{del}(x,z'))a \),

this means that \((z[x])a \in \mathcal{V}\), and therefore

\((z[x])a \neq \text{no-val}\).

4.2 If \( a(z) \neq a(z') \) we must have \( a(z)[i_{z,z'}] \neq a[i_{z,z'}] \).

\((\Leftarrow)\) Suppose that there is a \( \Sigma \)-algebra \( \mathcal{B} = (B, \ldots) \) and \( \tilde{a} \in [\tilde{W} \rightarrow \mathcal{B}] \) such that \( \mathcal{B}_1, \tilde{a} \models a(z) \).

Define the following sets:

\[ I = \{ \tilde{b} : \exists x. [\tilde{a}(x) \in \mathcal{B}_{\text{Index}} | x \in \tilde{x} \cup X_Z] \}, \]

where \( \tilde{b} \in \mathcal{B}_{\text{Index}} \), some chosen element.

\[ V = \{ * | \exists^* \tilde{a}(y) \mid y \in \tilde{Y}, \tilde{a}(y) \neq \text{no-val} \} \cup \{ (\tilde{a}(x), \tilde{b}) | x \in \tilde{x} \cup X_Z, (\tilde{a}(x), \tilde{b}) \neq \text{no-val} \} \]

where \( * \neq \text{B-value} \), is a fresh constant.

Then for each \( z \in Z \), define the following table:

\[ T_z \text{ in } \Pi[I, V] : \]

\[ T_z = \{ (\tilde{a}(x), \tilde{b}) | x \in \tilde{x} \cup X_Z \land \tilde{v} = (z[x]), \tilde{a}, \land \tilde{v} \neq \text{no-val} \} \]

Then define \( a \in [\tilde{W} \rightarrow \Pi[I, V]] \) as the function such that

\[ a(x) = \tilde{a}(x) \text{ if } x \in X, \quad a(y) = \tilde{a}(y) \text{ if } y \in Y, \quad \text{and} \]

\[ a(z) = T_z, \text{ if } z \in Z. \]

This is well-defined, since

\[ \text{if } y \in Y, \text{ then } \tilde{a}(y) \neq \text{no-val}, \text{ but } \tilde{a}(y) \models a(z), \text{ so } \tilde{a}(y) \in V. \]

\((*)\) Note that \( \tilde{a}(y) = \tilde{a}(y) \) means \( \tilde{a}(y) = \text{no-val} \) in case \( \tilde{a}(y) = \text{no-val} \).
We will be done if we prove that $\Pi [I, V], a \models \varphi$.

1. For conjuncts of the form $x_i = x_i', y = y'$, and their negations, this follows from $a$ and $\tilde{a}$ agreeing on such variables.

2. For conjuncts $z \in z'$, this follows from the definition of $T_z$ and $T_{z'}$.

2'. For conjuncts $z \neq z'$, we must have $\tilde{a}(z) \neq \tilde{a}(z')$ and therefore by (B), $a \models a \models \varphi$ (Q) we also must have

$$(\exists [x, z']) \tilde{a} \neq (\exists' [x, z']) \tilde{a}$$

which forces

$$T_z [\tilde{a}(x, z')]_{\Pi [I, V]} \neq T'_z [\tilde{a}(x, z')]_{\Pi [I, V]}$$

2''. For conjuncts $\text{no-val} = y_p$, by (B), $a \models a \models \varphi$ (Q) we must have $y_p \in \text{Val}[\{v\}]$ and $\tilde{a}(y_p) = \text{no-val}$, so that $a(y_p) = \text{no-val}$. Likewise, for $\text{no-val} \neq y_p$, we must have $\tilde{a}(y_p) \notin \text{V}$, so that $a(y_p) \neq \text{no-val}$.

2'''. For conjuncts $y_q = z_q [x_q]$, if $\tilde{a}(y_q) = \text{no-val}$, then $a(x_q) \notin \text{dom}(T_{z_q})$ and $a(y_q) = \text{no-val}$, so that $(z_q [x_q]) a = \text{no-val}$. And if $\tilde{a}(y_q) \neq \text{no-val}$, then $[a(x_q), a(y_q)] \in T_{z_q}$ and $(z_q [x_q]) a = a(y_q)$.
2°. For conjunctions \( y_q \neq z_q [x_q] \) if \( \tilde{a}(y_q) = \text{n}o.-\text{val} \),
then \( a(y_q) = \text{no.-val} \), and \( (z_q [x_q]) \tilde{a} \in V \).
Therefore, \( [a(x_q), \tilde{a}(z_q)[\tilde{a}(x_q)] ] \in T_{z_q} \), so that
\( a(y_q) \neq (z_q [x_q]) a \) holds. If \( \tilde{a}(y_q) \neq \text{n}o.-\text{val} \)
then \( \tilde{a}(y_q) = a(y_q) \in V \) and \( (z_q [x_q]) \tilde{a} \neq a(y_q) \),
which in either of the cases \( (z_q [x_q]) \tilde{a} = \text{n}o.-\text{val} \)
on \( \neq \text{n}o.-\text{val} \) forces \( a(y_q) \neq (z_q [x_q]) a \) by
the definition of \( T_{z_q} \).

(3) For conjunctions \( z_r = z'_r [x_r := y_r] \) we need to show
that \( T_{z_r} = T_{z'_r} [a(x_r) := a(y_r)]_{\Pi[I,V]} \). But since, by
construction, \( \text{dom}(T_{z_r}) \) and \( \text{dom}(T_{z'_r}) \) are subset
of \( \tilde{a}(X \cup X_z) \), this follows if we show that for
each \( x \in X \cup X_z \) we have \( T_{z_r} [\tilde{a}(x)]_{\Pi[I,V]} = T_{z'_r} [\tilde{a}(x) := a(y_r)]_{\Pi[I,V]} \).
We reason by cases.
If \( \tilde{a}(x) = \tilde{a}(x_r) \), then by (3) in the definition of \( a \) we
must have \( T_{(\text{}(z_r [x_r]) \tilde{a} = \tilde{a}(y_r) \text{)}_{\Pi}} [a(x_r), a(y_r)] \in T_{z, r} \), and therefore
Let \( a(y) = T_{z,r} \left[ \hat{a}(x) \right] \) for \( \pi_{[I,V]} \) and \( \pi_{[I,V]} \). 

Otherwise, if \( \hat{a}(x) \neq \hat{a}(x) \) by (3) we must have 

\[
T_{z,r} \left[ \hat{a}(x) \right] = T_{z,r} \left[ \hat{a}(x) \right] = T_{z,r} \left[ \hat{a}(x) := a(y) \right] \]

as desired.

(3') For conjuncts \( z \neq \pi_{[I,V]} \) we must show that there is an index \( \hat{a}(x) \in I \) such that \( T_{z} \left[ \hat{a}(x) \right] \neq T_{z} \left[ \hat{a}(x) \right] \).

But this follows from case (3') of the definition of \( \hat{a}_1 \), since \( \pi_{[I,V]} \hat{a} \neq \hat{a}(y) \) for any \( y \neq \hat{a}(x) \) for some \( \hat{a}(x) \in I \).

Otherwise, there is an \( x' \in (X - \hat{a}_1(x)) \) such that \( \hat{a}(x) \neq \hat{a}(x') \) and \( T_{z} \left[ \hat{a}(x) \right] = T_{z} \left[ \hat{a}(x') \right] \).

For conjuncts \( z = \pi_{[I,V]} \) we must show that for each \( x' \in X \) such that \( \hat{a}(x') = a(x) \) and \( \hat{a}(x') \neq \hat{a}(x) \) and \( \hat{a}(x') \neq \hat{a}(x) \), we have \( T_{z} \left[ \hat{a}(x') \right] = T_{z} \left[ \hat{a}(x') \right] \).
But this follows from the definition of $T_z$ and $T_{z'}$ and clause (4) in the definition of a 2l 4.

(4') For conjunct $z \neq \text{del}(x, z')$ we must show that there is an $x' \in X \cup X_z$ such that $T_z[\tilde{a}(x')]_{\pi[IIV]} \neq (\text{del}(x, z') a)[\tilde{a}(x')]_{\pi[IIV]}$. But this follows from clause (4) in the definition of a 2l 4, since either $T_z[\tilde{a}(x')]_{\pi[IIV]} \neq \text{no-val}$ and the result trivially follows, or otherwise, by clause (4) in the definition of a 2l 4 we must have an $x' \in X \cup X_z$ such that $\tilde{a}(x') \neq \tilde{a}(x)$ and $\tilde{a}(z)[\tilde{a}(x')]_{\pi[IIV]} \neq \tilde{a}(z)[a(x')]_{\pi[IIV]}$, which forces $T_z[\tilde{a}(x')]_{\pi[IIV]} \neq T_{z'}[\tilde{a}(x')]_{\pi[IIV]}$ and therefore the inequality $T_z[\tilde{a}(x')]_{\pi[IIV]} \neq (\text{del}(x, z') a)[\tilde{a}(x')]_{\pi[IIV]}$ as desired.

This completes the proof of the Theorem. □

Note, finally, that since $I$ and $V$ are finite sets, the construction mapping $M, \tilde{a} \models a 2l(\Phi)$ to $\pi[IIV], a \models \Phi$ is indeed effective, so $(\text{Table}[IIV], a 2l)$ is an effective descent map. In practice, it will be achieved with $M$ itself a computable finite model obtained by many-sorted congruence closure.
Remark. The ideas in the a2l descent map generalize and extend to actual arrays and tables as finite partial functions a similar formula transformation in:


However, the formula translation in the above-mentioned paper is used there to obtain a decision procedures just for arrays as total functions \( A : I \rightarrow V \), so the semantics given is completely different, suffers in their case from the anomalies pointed out of understandability arrays as total functions, cannot deal with undefinedness of arrays, and does not deal with tables at all.

The paper by Kapur and Zarba is worth reading for two reasons:

(i) it presents a notion of “reduction” that is a special case of that of descent maps, and

(ii) it defines other descent maps providing useful decision procedures for other theories.