CS 576 Lecture 25-26, 4/11-13/17

J. Meseguer

## 1. Descent Maps (II)

**Notation.** When $J : T \subseteq T'$ is a theory inclusion, we denote a descent map $(J, \delta) : (T', \mathcal{G}) \to (T, \mathcal{F})$ by $T' \supseteq^{\delta} T$.

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

We denote a general descent map $(H, \delta) : (T', \mathcal{G}) \to (T, \mathcal{F})$ thus: $(T', \mathcal{G}) \underset{H}{\overset{\delta}{\rightleftarrows}} (T, \mathcal{F})$.

Note that descent maps compose in the obvious way:

$$(T', \mathcal{G}) \underset{H}{\overset{\delta}{\rightleftarrows}} (T, \mathcal{F}) \underset{G}{\overset{\eta}{\rightleftarrows}} (T'', \mathcal{K})$$

$$= \delta ; \eta$$

$$G ; H$$

So they form a category **Descent**.

### 1.1 Simple Descent Maps

A useful class of descent maps have the nice property of giving a lot of information about how equisatisfiability is achieved at the level of **models**. They are what I call

simple descent maps

**Definition.** A descent map $(T', \mathcal{G}) \underset{H}{\overset{\delta}{\rightleftarrows}} (T, \mathcal{F})$ is called **simple** iff

1. $H$ is expansive or literally expansive
2. $\forall \varphi \in \mathcal{G}$, $\forall m' \in mod(T')$
   $\varphi$ satisfiable in $m' \Leftrightarrow^{\exists \psi \in \delta(\varphi)}$ satisfiable in $m'|_H$.

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

**Lemma.** Any $(T', \mathcal{G}) \underset{H}{\overset{\delta}{\rightleftarrows}} (T, \mathcal{F})$ as above satisfying (1)-(2) is a descent map. (for literally expansive: $T' = (\Sigma', \mathcal{B})$, $T = (\Sigma, \mathcal{A})$)

**Proof.**

1) $\forall \varphi \in \mathcal{G}$, ($\varphi \in \mathcal{B}$) $T'$-satisfiable $\Leftrightarrow \exists$ ($m' \in mod(T')$, $m' \in \mathcal{B}$) s.t. $\varphi$ sat. in $m'$
$\Leftrightarrow \exists$ ($m' \in mod(T')$ s.t. $\exists^{\psi \in \delta(\varphi)}$ ($\psi \in \mathcal{A}_\simeq$) sat. in $m'|_H$ $\Rightarrow \exists$ ($\psi \in \delta(\varphi)$) $T$-satisfiable

2) $\forall \varphi \in \mathcal{G}$, $\exists^{\psi \in \delta(\varphi)}$ ($\psi \in \mathcal{B}_\simeq$) $T$-satisfiable $\Leftrightarrow \exists$ ($m \in mod(T)$, $m \in \mathcal{A}$) s.t. $\exists^{\psi \in \delta(\varphi)}$ sat. in $m$
$\Leftrightarrow$ (by $H$ lit exp. expansive) $\exists$ ($m' \in mod(T')$, $m' \in \mathcal{B}_\simeq$) s.t. $\exists^{\psi \in \delta(\varphi)}$ sat. in $m'|_H$
$\Leftrightarrow \exists$ ($m' \in mod(T')$, $m' \in \mathcal{B}_\simeq$) s.t. $\varphi$ sat. in $m' \Leftrightarrow \varphi$ $T'$-satisfiable. □

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 Revisited). 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

$$\left(\Sigma, \{\mathcal{T}_{\Sigma/E \cup B}\}\right) \supseteq \left(\Omega, \{\mathcal{T}_{\Omega/E_\Omega \cup B_\Omega}\}\right)$$

where $(\Sigma, E \cup B)$ has an FVP decomposition $(\Sigma, B, \vec{E})$ protecting a constructor decomposition $(\Omega, B_\Omega, \vec{E}_\Omega)$ which is OS-compact. The descent map is the composition of simple descent maps:

$$(\Sigma, \{T_{\Sigma/E\cup B}\}), QF(\Sigma)) \overset{\text{dnf; conj}}{\underset{1_\Sigma}{\iff}} ((\Sigma, \{T_{\Sigma/E\cup B}\}), \wedge \mathrm{Lit}(\Sigma)) \overset{\text{ctor-var-unif}}{\underset{1_\Sigma}{\iff}} ((\Sigma, \{T_{\Sigma/E\cup B}\}), \wedge \neg \mathrm{At}(\Sigma))$$

$$\overset{\text{ctor-vars}}{\supseteq} ((\Omega, \{T_{\Omega/E_\Omega \cup B_\Omega}\}), \wedge \neg \mathrm{At}(\Omega)).$$

For another example, we can consider the descent from $\mathcal{Z}_+$ to $\mathcal{N}_+$ in Appendix D of VarSat associated to the literally expansive extension:

$$\mathcal{Z}_+ = (\Sigma_{\mathrm{INT}}, \{\mathbb{Z}_{+,-}\}) \supseteq (\Sigma_{\mathrm{NAT}}, \{\mathbb{N}_+\}) = \mathcal{N}_+$$

which is accomplished by the chain of simple descent maps

$$\mathcal{Z}_+ \underset{1_{\Sigma_{\mathrm{INT}}}}{\overset{\nu}{\rightleftarrows}} \mathcal{I}_+ \overset{-}{\supseteq} \mathcal{N}_+$$

where the classes of formulas involved in the (several steps) into which $\nu$ and $-$ are decomposed is left implicit. (QF)

Note that in all these examples these simple descent maps are effective. For example in variant satisfiability for $(\Sigma, \{T_{\Sigma/E\cup B}\})$ $\varphi \in QF(\Sigma)$ was first mapped by dnf; conj to a set $\{\wedge C_i \wedge \wedge D_i \mid i \in I\}$ of conjunctions of equalities $C_i$ and disequalities $D_i$, $i \in I$, then by ctor-var-unif each $\wedge C_i \wedge \wedge D_i$ was mapped to a set of conjunctions $\{\wedge D_i \alpha\}_{\alpha \in \mathrm{ctoralg}(\wedge C_i)}$, and then each

$\Lambda D_i \alpha$ was mapped to a set of ctor-variants

$[\![ \Lambda D_i \alpha ]\!]^{\hat{\Sigma}}_{\vec{E},B}$. Then, choosing some $(\Lambda D', \beta)$ in this and a ground substitution $\beta$ for its finite sorts, we checked that $(\Lambda D' \beta)!$ was $B_\Sigma$-consistent. From this we can effectively compute a ground substitution $\tau$ s.t.

$$T_{\Sigma/\vec{E}_\Sigma, B_\Sigma}, \tau \models \Lambda D' \beta$$

But then we can construct the satisfying assignment

$$T_{\Sigma/E \cup B}, \alpha \beta \beta \tau \models \varphi$$

as desired.

This can used as part of a process to provide a checkable certificate that the satisfiability of $\varphi$ in $T_{\Sigma/E \cup B}$ 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 Descent, but also under a kind of "parallel composition" by union, provided they are simple and obey some extra conditions. Specifically, assume $(T'_i, G_i) \underset{H_i}{\overset{\delta_i}{\rightleftarrows}} (T_i, F_i)$ simple descent maps $(1 \le i \le 2)$

such that:

(1) $sig(T_1)$ and $sig(T_2)$ (resp. $sig(T_1')$ and $sig(T_2')$ are optimally intersectable in $\Sigma_0$ (resp. $\Sigma_0'$) where $\Sigma_0$ and $\Sigma_0'$ consist only of sorts (no opers. or pred. symbols) and $H_1$ and $H_2$ agree on $\Sigma_0$ and induce a surjective restriction to $H_0 : \Sigma_0 \longrightarrow \Sigma_0'$. Then,

Theorem Under the above conditions,

$$(T_1' \cup T_2', \mathcal{G}_1 \wedge \mathcal{G}_2) \underset{H_1 \cup H_2}{\overset{\delta_1 \wedge \delta_2}{\rightleftarrows}} (T_1 \cup T_2, \mathcal{F}_1 \wedge \mathcal{F}_2)$$

where $\delta_1 \wedge \delta_1 (\varphi_1 \wedge \varphi_2) = \{ \psi_1 \wedge \psi_2 \mid \psi_i \in \delta_i(\varphi_i), 1 \leq i \leq 2 \}$ 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 ascent map with $H_1 \cup H_2$ expansive. The rest of the proof goes as follows:

$\forall \varphi_1 \wedge \varphi_2 \in \mathcal{G}_1 \wedge \mathcal{G}_2 \quad \forall [\mathfrak{M}_1', \mathfrak{M}_2'] \in mod(T_1' \cup T_2')$

$\varphi_1 \wedge \varphi_2$ sat. in $[\mathfrak{M}_1', \mathfrak{M}_2'] \iff \varphi_1$ sat. in $\mathfrak{M}_1'$ and $\varphi_2$ sat. in $\mathfrak{M}_2' \iff$

$\exists \psi_1 \in \delta(\varphi_1)$ s.t. $\psi_1$ sat. in $\mathfrak{M}_1'|_{H_1}$ and $\exists \psi_2 \in \delta(\varphi_2)$ s.t. $\psi_2$ sat. in $\mathfrak{M}_2'|_{H_2}$

$\iff \exists \psi_1 \wedge \psi \in \delta_1 \wedge \delta_2 (\varphi_1 \wedge \varphi_2)$ s.t. $[\psi_1, \psi_2]$ sat. in $[\mathfrak{M}_1', \mathfrak{M}_2']|_{H_1 \cup H_2}$. □

---

The practical interest of this theorem is that as we shall see in a few lectures, under the assumption of $T_1, T_2$ being stably infinite, we can combine the descent map $(H_1 \cup H_2, \delta_1 \wedge \delta_2)$ with the Nelson-Oppen procedure to obtain a decision procedure for $\mathcal{G}_1 \wedge \mathcal{G}_2 - T_1 \cup T_2$ satisfiability if $\mathcal{F}_i - T_i$ 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 corresponding one for real arrays and tables. But this fails to be the case rather badly because, for example:

(a) For $\mathbb{N}$ the set of Indices, and $1 = \{0\}$ the set of values, the model of arrays used, namely the function set $[\mathbb{N} \to 1]$ 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 $A = \{(i_1, 1), \ldots, (i_k, 1)\}$ with $i_j \in \mathbb{N}$, $1 \leq j \leq k$, $i_j \neq i_{j'}$ if $j \neq j'$. There is no hope whatsoever that satisfiability of a QF formula in $[\mathbb{N} \to 1]$ 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 $(\forall A: \text{Array}, A': \text{Array})\ A = A'$ is a theorem of $[\mathbb{N} \to 1]$; but this is of course false for the countably infinite different array with index set $\mathbb{N}$ and values in $\{1\}$.

(6) For $\mathbb{N}$ the set of Indices and $2 = \{0, 1\}$ the set of values, we have $[\mathbb{N} \to 2] \cong \mathcal{P}(\mathbb{N})$, 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 $i_j \neq i_{j'}$ 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 Maude specification does so as parameterized data types:

```
*** theory of equality

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

vars x y : Elt .

eq x ~ x = true .
ceq x = y if x ~ y = true [nonexec] .
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 mt : -> 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) .
endfm

*** 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 : I$Elt .  vars v v' : V$Elt .  var A : Array{I,V} .

cmb ([i,v],A) : Array{I,V} if i in dom(A) = false .

op _[_] : Array{I,V} I$Elt -> 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} I$Elt V$Elt -> 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 : I$Elt . vars v : V$Elt . var T : Table{I,V} .

op del : I$Elt 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,([j,v],T)) = [j,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 I$Elt .
  sort Value? to Val?{V} .
  sort Readable to Table{I,V} .
endv)

)
```

Recall from Lecture 19, pg. 11 that given a parameterized data type associated to a theory inclusion $J : P \subseteq B$ between the parameter theory $P$ and the "body" theory $B$, say with $sig(P) = \Sigma_P$ and $sig(B) = \Sigma_B$, where $P$ and $B$ are theories in order-sorted Horn logic with the theory of the parameterized data type associated to $J$ is the semantic theory $(\Sigma_B, \mathbb{F}_J)$, where, by definition,

$$\mathbb{F}_J = \{ \mathbb{F}_J(M) \mid M \in mod(P) \}$$

where $\mathbb{F}_J$ is the left adjoint to the reduct functor

$-|_{\Sigma_P} : Mod(B) \longrightarrow Mod(P)$.

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

$$P = EQ*(Elt\ to\ I\$Elt) \cup TRIV*(Elt\ to\ V\$Elt)$$

where $*(A\ to\ B)$ is a sort-renaming of sort name $A$ to sort name $B$. Then $B$ in each case is the theory defined above for EXT-ARRAY or TABLE, plus the axioms of EQ already contained in the subtheory $P$. Let $\mathbb{A}$ denote $\mathbb{F}_J$ for the

inclusion $P \overset{J}{\subseteq}$ EXT-ARRAY, and $\Pi = \mathbb{F}_{J'}$ for the inclusion $P \overset{J'}{\subseteq}$ TABLE. Then the semantic theories

$$(\Sigma_A, \mathbb{A}) \quad \text{and} \quad (\Sigma_T, \Pi)$$

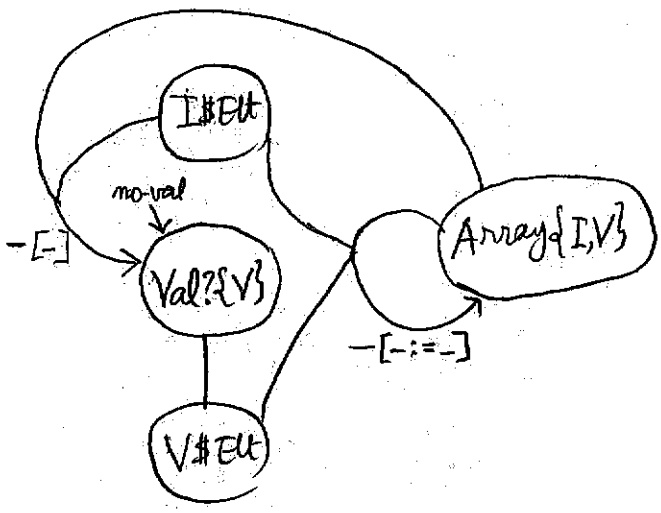
where $\Sigma_A$ and $\Sigma_T$ are the signatures of EXT-ARRAY and TABLE are exactly the ones associated by free algebra semantics to the keywords fmod ... endfm enclosing the EXT-ARRAY and TABLE declarations.

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

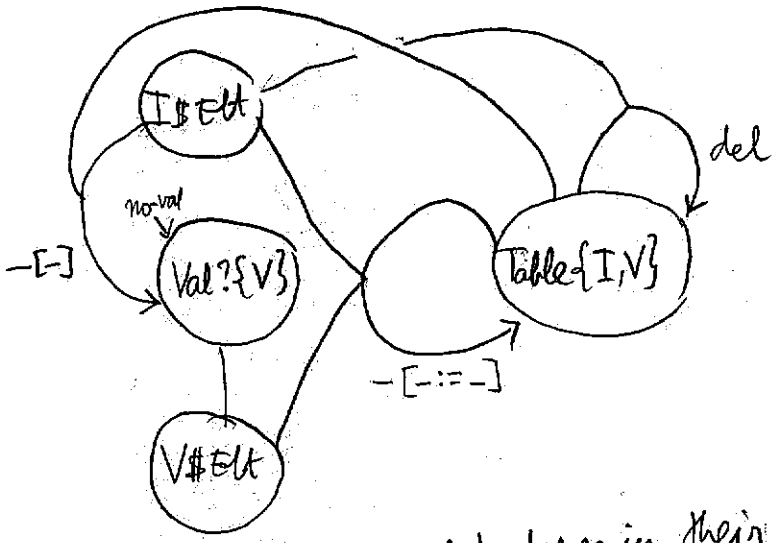
$$\text{dom} : \text{Rel}\{I, V\} \to \text{Set}\{I\}$$

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

$\Sigma_{A_0}$:



$\Sigma_{T_0}$ :




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

$$A = (\Sigma_{A_0}, \mathbb{A}|_{\Sigma_{A_0}}) \quad \text{and} \quad (\Sigma_{T_0}, \mathbb{T}|_{\Sigma_{T_0}}) : \mathsf{T}$$

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

$$\text{def} : \text{I\#Elt}\ \text{Array}\{I,V\}$$

$$\text{resp. def} : \text{I\#Elt}\ \text{Table}\{I,V\}$$

but such extensions are <u>purely definitional extensions</u> 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 $x : I\#Elt$, $A : \text{Array}\{I,V\}$, and $T : \text{Table}\{I,V\}$

therefore, there is <u>no need to extend</u> $\Sigma_{A_0}$ or $\Sigma_{T_0}$, since 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:

$$\left(T, QF(\Sigma_{T_0})\right) \underset{1_{\Sigma_{T_0}}}{\overset{dnf;\,cnjs;\,vals}{\longleftrightarrow}} \left(T, SALiA(\Sigma_{T_0})\right) \underset{Table\{I,V\}}{\overset{a2l}{\longleftrightarrow}} \left(LOOKUP, QF(\Sigma_{LOOKUP})\right)$$

where LOOKUP is the theory $(\Sigma_{LOOKUP}, \emptyset)$ defined in pg 9, and $Table\{I,V\}$ is the theory interpretation (view) also defined in pg. 9, which obviously factors as:

$$LOOKUP \xrightarrow{Table\{I,V\}} T \subseteq TABLE\{I,V\}$$

Note that this descent map is not simple, since the reduct functor $-|_{Table\{I,V\}} : Mod(T) \longrightarrow Mod(LOOKUP)$ is such that some models of $Mod(LOOKUP)$ cannot be expanded to models of $Mod(T)$. For example the model $M = (M, -_M) \in mod(LOOKUP)$ with $M_{Index} = \{a\}$, $M_{Value?} = \{b, no\text{-}val\}$, , $M_{Readable} = \{r, r'\}$

and $\text{no-val}_{\mathcal{M}} = \text{no-val}$, $\mathcal{M}[a]_{\mathcal{M}} = b$ cannot be expanded to a model $\mathcal{M}' \in \text{mod}(T)$ such that $\mathcal{M}'|_{\text{Table}\{I,V\}} = \mathcal{M}$.

Note that, since $\text{LOOKUP} = (\Sigma_{\text{LOOKUP}}, \emptyset)$, LOOKUP-satisfiability of any $\psi \in QF(\Sigma_{\text{LOOKUP}})$ is decidable by many-sorted congruence closure. Therefore, all we need to do is to define the function $a2\ell : S\wedge\text{Lit}(\Sigma_{T_0}) \longrightarrow QF(\Sigma_{\text{LOOKUP}})$ and to prove that $(\text{Table}\{I,V\}, a2\ell)$ is indeed a decent map. In this way we obtain a decision procedure for $(T, QF(\Sigma_{T_0}))$.

What does a formula $\varphi \in S\wedge\text{Lit}(\Sigma_{T_0})$ look like? It looks exactly as follows, where I list negated atoms below for readability:

$$\varphi = \bigwedge_i x_i = x'_i \wedge \bigwedge_j y_j = y'_j \wedge \bigwedge_\ell z_\ell = z'_\ell \wedge \bigwedge_p \text{no-val} = y_p \wedge \bigwedge_q y_q = z_q[x_q] \wedge \bigwedge_r z_r = z'_r[x_r := y_r]$$
$$\wedge \bigwedge_{i'} x_{i'} \neq x'_{i'} \wedge \bigwedge_{j'} y_{j'} \neq y'_{j'} \wedge \bigwedge_{\ell'} z_{\ell'} \neq z'_{\ell'} \wedge \bigwedge_{p'} \text{no-val} \neq y_{p'} \wedge \bigwedge_{q'} y_{q'} \neq z_{q'}[x_{q'}] \wedge \bigwedge_{r'} z_{r'} \neq z'_{r'}[x_{r'} := y_{r'}]$$
$$\wedge \bigwedge_s z_s = \text{del}(x_s, z'_s) \wedge \bigwedge_{s'} z_{s'} \neq \text{del}(x_{s'}, z'_{s'})$$

where the $x$'s belong to a (finite) set $X$ of variables of sort I#Elt, the $y$'s belong to $Y = Y_{V\#\text{Elt}} \dot\cup Y_{\text{val?}\{V\}}$, and the $z$'s to a set $Z$ of variables of sort $\text{Table}\{I,V\}$, and $X$, $Y$ and $Z$ only contain those variables appearing in the above formula. The $a2\ell$ function is defined as $a2\ell(\varphi) = a2\ell_0(\varphi) \wedge a2\ell_1(\varphi)$, with $a2\ell_1$ defined by:

$$a2\ell_1(\text{lit} \wedge C) = a2\ell_1(\text{lit}) \wedge a2\ell_1(C)$$
$$a2\ell_1(\top) = \top$$

where we assume $\wedge$ associative and commutative with identity $\top$

and where $lit$ and $C$ represent, respectively, a literal and a conjunction in $SAtLit(\Sigma_{T_0})$, and $a2l_1(lit)$ is defined as follows

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

except that the sorts of the variables are renamed (*) as follows: $x : I\#Elt \mapsto x : Index$, $y : V\#Elt \mapsto y : Value?$, $y' : Val?\{V\} \mapsto y' : Value?$, $z : Table\{I,V\} \mapsto z : Readable$

Let $\tilde{X}$, $\tilde{Y}_{V\#Elt}$, $\tilde{Y}_{Val?\{V\}}$, and $\tilde{Z}$ denote the renamed versions of $X$, $Y_{V\#Elt}$, $Y_{Val?\{V\}}$, and $Z$ according to the above renamings of variables.

(2) We introduce a new set $X_Z$ of variables of sort Index such that $\tilde{X} \cap X_Z = \emptyset$, defined as follows:

$$X_Z = \{ x_{z,z'} \mid (z,z') \in Z^2 \wedge z < z' \}$$

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

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

---

(*) To afford a simpler notation we assume without loss of generality that the names of variables in $Y_{V\#Elt}$ and $Y_{Val?\{V\}}$ are disjoint, so for any $y : V\#Elt$ and $y' : Val?\{V\}$ we must have $y : Value? \neq y' : Value?$.

$$(3)\quad a2l_1\big(z = z'[x:=y]\big) = \Big(z[x]=y \wedge \bigwedge_{x'\in(\widetilde{X}-\{x\})\dot{\cup}X_z} x\neq x' \Rightarrow z[x']=z'[x']\Big)$$

$$(3')\quad a2l_1\big(z \neq z'[x:=y]\big) = \Big(z[x]\neq y \vee \bigvee_{x'\in(\widetilde{X}-\{x\})\dot{\cup}X_z} z[x']\neq z'[x']\Big)$$

$$(4)\quad a2l_1\big(z = del(x,z')\big) = \Big(z[x]=\text{no-val} \wedge \bigwedge_{x'\in(\widetilde{X}-\{x\})\dot{\cup}X_z} x\neq x' \Rightarrow z[x']=z'[x']\Big)$$

$$(4')\quad a2l_1\big(z \neq del(x,z')\big) = \Big(z[x]\neq \text{no-val} \vee \bigvee_{x'\in(\widetilde{X}-\{x\})\dot{\cup}X_z} z[x']\neq z'[x']\Big)$$

Finally, $a2l_0(\varphi)$ is defined as follows:

$$a2l_0(\varphi) = \Big(\bigwedge_{z,z'\in\widetilde{Z}\,\wedge\, z<z'} z\neq z' \Rightarrow z[x_{z,z'}]\neq z'[x_{z,z'}]\Big) \wedge \Big(\bigwedge_{y\in\widetilde{Y}_v} y\neq \text{no-val}\Big)$$

That is, $a2l_0(\varphi)$ intuitively can be understood as saying:

(a) if readable $z$ is different from readable $z'$, they can be distinguished by ready $x_{z,z'}$

(b) the interpretation of a variable $y$ of sort $\widetilde{Y}_v$ can never be no-val.

**Theorem** $(T, SALit(\Sigma_{T_0})) \underset{Table[I,V]}{\overset{a2l}{\rightleftarrows}} (LOOKUP, QF(\Sigma_{LOOKUP}))$

is a descent map.

**Proof.** For each $\varphi \in SALit(\Sigma_{T_0})$ we have to prove:

$\varphi$ $T$-satisfiable $\iff$ $a2l(\varphi)$ LOOKUP-satisfiable.

($\Rightarrow$). It is enough to show that for any non-empty sets $I$ and $V$ and any $a \in [W \to T[I,V]]$ such that $\mathbb{T}[I,V], a \models \varphi$ (where $W = vars(\varphi)$, and $\mathbb{T}[I,V] = (T[I,V], -_{\mathbb{T}[I,V]})$) we have an $\tilde{a} \in [\tilde{W} \to T[I,V]|_{Table[I,V]}]$ such that $\mathbb{T}[I,V]|_{Table[I,V]}, \tilde{a} \models a2l(\varphi)$, where $\tilde{W} = vars(a2l(\varphi))$.

We choose $\tilde{a}$ as follows. Note that $\tilde{W} = \tilde{X} \dot{\cup} X_Z \dot{\cup} \tilde{Y}_V \dot{\cup} \tilde{Y}_{V?} \dot{\cup} \tilde{Z}$, where the $\tilde{X}$, $\tilde{Y}_V$, $\tilde{Y}_{V?}$ and $\tilde{Z}$ are in bijective correspondence with $X$, $Y_{V\#Elt}$, $Y_{Val?[V]}$, 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

$$\tilde{a}(X_{z,z'}) = \text{if } a(z) = a(z') \text{ then } i_0 \text{ else } i_{z,z'} \text{ fi}$$

where $i_{z,z} \in I$ is some chosen index such that $a(z)[i_{z,z}]_{\mathbb{T}[I,V]} \neq a(z')[i_{z,z'}]_{\mathbb{T}[I,V]}$, which always exists

since $a(z)$ and $a(z')$ are finite partial functions from $\underline{I}$ 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 $a2l(\varphi)$ is a conjunction of formulas, to show that

$$\Pi[\underline{I},V]|_{\text{Table}\{\underline{I},V\}}, \tilde{a} \models a2l(\varphi)$$

we just need to show that each conjunct in $a2l(\varphi)$ is satisfied. We reason by cases:

(1) for literals in $\varphi$ that are mapped identically except for variable renaming this is trivially the case

(2) $a2l_0(\varphi)$ is also trivially satisfied by the choice of the $i_{z,z'} = \tilde{a}(x_{z,z'})$ and the fact that $a(y) \neq$ no-val for any $y \in Y_{V\$Elt}$.

(3) (we prove the case (3') in the def. of $a2l_1$, the case (3) is similar) we reason by cases to see that $a2l_1(z \neq z'[x:=y])$ holds (for $\tilde{a}$) if

$z \neq z'[x:=y]$ does for $a$ — 3.1 If $a(z) = a(z')$ and ($a(z) \neq z'[x:=y]\,a$) we must have $a(z)[a(x)]_{\Pi[\underline{I},V]} \neq a(y)$

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

(4) (we prove case (4') and leave case (4) to the reader). We again reason by cases to see that $a2l_1(z \neq del(x,z'))$ holds for $\tilde{a}$ if $z \neq del(x,z')$ holds for $a$:

4.1 If $a(z) = a(z')$ and $a(z) \neq (del(x,z'))\,a$,
this means that $(z[x])\,a \in V$, and therefore
$(z[x])\,a \neq$ no-val.

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

($\Leftarrow$) Suppose that there is a $\Sigma_{LOOKUP}$-algebra $\mathcal{B} = (B, \tau_B)$
and $\tilde{a} \in [\tilde{W} \to B]$ such that $\mathcal{B}, \tilde{a} \models a2l(\varphi)$.

Define the following sets:

$$I = \{b_0\} \cup \{\tilde{a}(x) \in B_{Index} \mid x \in \tilde{X} \cup X_Z\}, \text{ where}$$

$b_0 \in B_{Index}$ is some chosen element

$$V = \{*\} \dot{\cup} \{\tilde{a}(y) \mid y \in \tilde{Y} \wedge \tilde{a}(y) \neq no\text{-}val_B\} \cup \{(z[x])\tilde{a} \mid {}^{z \in Z}_{x \in \tilde{X} \cup X_Z} \wedge (z[x])\tilde{a} \neq no\text{-}val_B\}$$

where $* \notin B_{Value}$ is a fresh constant

Then for each $z \in Z$ define the following table

$T_z$ in $\Pi[I,V]$:

$$T_z = \{[\tilde{a}(x), v] \mid x \in \tilde{X} \cup X_Z \wedge v = (z[x])\tilde{a} \wedge v \neq no\text{-}val_B\}$$

Then define $a \in [W \to T[I,V]]$ as the function such that
$a(x) = \tilde{a}(x)$ if $x \in X$, $a(y) = \tilde{a}(y)$ if $y \in Y$, and
$a(z) = T_z$ if $z \in Z$. This is well-defined, since
if $y \in Y_{V\$Elt}$ $\tilde{a}(y) \neq no\text{-}val_B$, by $a2l_0(\varphi)$, so $\tilde{a}(y) \in V$.

(*) Note that $a(y) = \tilde{a}(y)$ means $a(y) =$ no-val in case $\tilde{a}(y) = no\text{-}val_B$.

We will be done if we prove that $\mathbb{T}[I,V], a \models \varphi$.

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

2. For conjuncts $z_\ell = z'_\ell$ this follows from the definition of $T_z$ and $T_{z'}$

2'. For conjuncts $z_\ell \neq z'_\ell$ we must have $\tilde{a}(z_\ell) \neq \tilde{a}(z'_\ell)$ and therefore by $\mathcal{B}, a \models a2l_0(\varphi)$ we also must have $(z[x_{z,z'}])\tilde{a} \neq (z'[x_{z,z'}])\tilde{a}$, which forces

$$T_z[\tilde{a}(x_{z,z'})]_{\mathbb{T}[I,V]} \neq T_{z'}[\tilde{a}(x_{z,z'})]_{\mathbb{T}[I,V]}$$

2''. For conjuncts $\text{no-val} = y_p$, by $\mathcal{B}, a \models a2l_0(\varphi)$ we must have $y_p \in Y_{Val?\{V\}}$ and $\tilde{a}(y_p) = \text{no-val}_{\mathcal{B}}$, so that $a(y_p) = \text{no-val}$. Likewise, for $\text{no-val} \neq y_p$ we must have $\tilde{a}(y_p) \in V$, so that $a(y_p) \neq \text{no-val}$.

2'''. For conjuncts $y_q = z_q[x_q]$, if $\tilde{a}(y_q) = \text{no-val}_{\mathcal{B}}$, 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 $\tilde{a}(y_q) \neq \text{no-val}_{\mathcal{B}}$, then $[a(x_q), a(y_q)] \in T_{z_q}$ and $(z_q[x_q])a = a(y_q)$.

$2^{nr}$. For conjuncts $y_q \neq z_q[x_q]$ if $\tilde{a}(y_q) = \text{no-val}_{\mathcal{B}}$, then $a(y_q) = \text{no-val}$, and $(z_q[x_q])\tilde{a} \in V$. Therefore, $[a(x_q), \tilde{a}(z_q)[\tilde{a}(x_q)]_{\mathcal{B}}] \in T_{z_q}$, so that $a(y_q) \neq (z_q[x_q])a$ holds. If $\tilde{a}(y_q) \neq \text{no-val}_{\mathcal{B}}$ 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 $(z_q[x_q])\tilde{a} = \text{no-val}_{\mathcal{B}}$ or $\neq \text{no-val}_{\mathcal{B}}$ 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 subsets of $\tilde{a}(\tilde{X} \cup X_z)$, this follows if we show that for each $x \in \tilde{X} \cup X_z$ we have $T_{z_r}[\tilde{a}(x)]_{\Pi[I,V]} = T_{z'_r}[\tilde{a}(x_r) := a(y_r)]_{\Pi[I,V]}[\tilde{a}(x)]_{\Pi[I,V]}$. We reason by cases.

If $\tilde{a}(x) = \tilde{a}(x_r)$, then by (3) in the definition of a2l1 we must have $(z_r[x_r])\tilde{a} = \tilde{a}(y_r)$, which forces $[a(x_r), a(y_r)] \in T_{z,r}$ and therefore

$$a(y_r) = T_{z,r}[\tilde{a}(x_r)]_{\Pi[I,V]} = T_{z'_r}[\tilde{a}(x_r) := a(y_r)]_{\Pi[I,V]}[\tilde{a}(x_r)]_{\Pi[I,V]},$$

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

$$T_{z_r}[\tilde{a}(x)]_{\Pi[I,V]} = T_{z'_r}[\tilde{a}(x)]_{\Pi[I,V]} = T_{z'_r}[\tilde{a}(x_r) := a(y_r)]_{\Pi[I,V]}[\tilde{a}(x)]_{\Pi[I,V]}$$

as desired.

(3′) For conjuncts $z \neq z'[x := y]$ we must show that there is an index $\tilde{a}(x') \in I$ such that $T_z[\tilde{a}(x')]_{\Pi[I,V]} \neq$

$$T_{z'}[i(x) := a(y)]_{\Pi[I,V]}[\tilde{a}(x')]_{\Pi[I,V]}.$$

But this follows from case (3′) of the definition of a2l1, since either $(z[x])\tilde{a} \neq \tilde{a}(y)$, forcing $T_z[\tilde{a}(x)] \neq T_{z'}[\tilde{a}(x) := a(y)]_{\Pi[I,V]}[i(x)]_{\Pi[I,V]} = \tilde{a}(y)$, otherwise there is an $x' \in (\tilde{X} - \{x\}) \cup X_z$ such that $\hat{a}(x) \neq \tilde{a}(x')$ and $T_z[\tilde{a}(x')]_{\Pi[I,V]} = T_{z'}[\hat{a}(x')]_{\Pi[I,V]}$, forcing again the above disequality.

(4) For conjuncts $z = del(x, z')$ we must show that for each $x' \in X \cup X_z$ either $\tilde{a}(x') = a(x)$, and then $a(x) \notin dom(T_z)$, or $\tilde{a}(x') \neq a(x)$, and then $T_z[\tilde{a}(x')]_{\Pi[I,V]} = T_{z'}[\tilde{a}(x')]_{\Pi[I,V]}$.

But this follows from the definition of $T_z$ and $T_{z'}$, and clause (4) in the definition of a2l1.

(4') For conjuncts $z \neq del(x,z')$ we must show that there is an $x' \in X \cup X_z$ such that $T_z[\tilde{a}(x')]_{\mathbb{T}[I,V]} \neq (del(x,z')a)[\tilde{a}(x')]_{\mathbb{T}[I,V]}$. But this follows from clause (4) in the definition of a2l1, since either $T_z[\tilde{a}(x)]_{\mathbb{T}[I,V]} \neq$ no-val and the result trivially follows, or otherwise, by clause (4) in the definition of a2l1 we must have an $x' \in X \cup X_z$ such that $\tilde{a}(x') \neq \tilde{a}(x)$ and $\tilde{a}(z)[\tilde{a}(x')]_B \neq \tilde{a}(z)[a(x')]_B$, which forces $T_z[\tilde{a}(x')]_{\mathbb{T}[I,V]} \neq T_{z'}[\tilde{a}(x')]_{\mathbb{T}[I,V]}$ and therefore the inequality $T_z[\tilde{a}(x')]_{\mathbb{T}[I,V]} \neq (del(x,z')a)[\tilde{a}(x')]_{\mathbb{T}[I,V]}$ as desired.

This completes the proof of the Theorem. □

Note, finally, that since $I$ and $V$ are finite sets, the construction mapping $\mathfrak{M}, \tilde{a} \models a2l(\varphi)$ to $\mathbb{T}[I,V], a \models \varphi$ is indeed effective, so $(Table_{\{I,V\}}, a2l)$ is an effective descent map. In practice, it will be achieved with $\mathfrak{M}$ 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:

D. Kapur and C. Zarba, "A Reduction Approach to Decision Procedures", Technical Report, C.S. Dept., University of New Mexico, 2005, http://www.cs.unm.edu/~kapur/mypapers/reduction.pdf

However, the formula translation in the above-mentioned paper is used there to obtain a decision procedures just for arrays as total functions $A \in [I \to V]$, so the semantics given is completely different, suffers in their ease from the anomalies pointed out of understanding 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.