Delta Logics: Logics for Change

Department of Computer Science, University of Illinois, Urbana-Champaign

Abstract.

1 Delta Logics

In this section, we define a general delta-logic extending many-sorted first-order logic with least fixpoints with background axiomatizations of some of the sorts.

Let us fix a many-sorted first-order signature $\Sigma = (S, \mathcal{F}, \mathcal{P}, \mathcal{C}, \mathcal{G}, \mathcal{R})$ where $S = \{\sigma_0, \ldots, \sigma_n\}$ is a nonempty finite set of sorts, $\mathcal{F}$, $\mathcal{P}$, and $\mathcal{C}$ are sets of function symbols, relation symbols, and constant symbols, respectively, and $\mathcal{G}$ and $\mathcal{R}$ are function and relation symbols that will be recursively defined. These symbols have implicity defined an appropriate arity and a type signature.

Let σ_0 be a special sort that we refer to as the *location* sort, which will model locations of the heap. The other sorts, which we refer to as background sorts, can be arbitrary and constrained to conform to some theory (such as a theory of arithmetic or a theory of sets).

We assume the following restrictions:

- We assume all functions in $\mathcal{F}$ map either from tuples of one sort to itself or from the foreground sort σ_0 to a background sort σ_i. Relations in $\mathcal{P}$ are over tuples of one sort only.
- The functions in $\mathcal{F}$ whose domain is over the foreground sort σ_0 are *unary.* Also, relations over the foreground sort σ_0 are unary relations.
- Recursively defined functions (in $\mathcal{G}$) are all unary functions from the foreground sort σ_0 to the foreground sort or a background sort. Recursively defined relations (in $\mathcal{R}$) are all unary relations on the foreground sort σ_0.

The restriction to have unary functions from the foreground sort (which models locations) is sufficient to model pointers on the heap (unary functions from σ_0 to σ_0) and to model data stored in the heap (like the key stored at locations modeled as a function from σ_0 to a background sort of integers). This restriction will greatly simplify the presentation of delta-logics below. The restriction of having unary recursively defined functions and relations will also simplify the notation. Note that recursive definitions such as $lseg(x, y)$ that are binary can be written recursively as unary relations such as $lseg_y(x)$ (i.e., parameterized over the variable y) with recursion on the variable x.

Define FO+LFP: syntax and semantics.

1.1 Delta Logics

We parameterize delta logics by a finite set of first-order variables $\Delta = \{v_1, \dots v_n\}$.

Intuitively, a delta logic formula $\varphi(\boldsymbol{x})$ is a formula that evaluates on a model M while *ignoring* the functions/relations on the locations interpreted for the variables in Δ.

More precisely, a semantic definition of the delta logic over Σ with respect to Δ, is defined as below. First, let us define when a pair of models over the same universe and an interpretations differ only on Δ.

Definition 1 (Models differing only on Δ). *Let M and M' be two Σ-models with universe U that interpret constants the same way, and let I be an interpretation of variables over U. Then we say (M, I) and (M', I) differ only on Δ if:*

- *for every function symbol f and for every $l \in U$, $[\![f]\!]_M(l) \neq [\![f]\!]_{M'}(l)$ only if there exists some $v \in \Delta$ such that $I(v) = l$.*
- *for every relation symbol S and for every $l \in U$, $[\![S]\!]_M(l) \not\equiv [\![S]\!]_{M'}(l)$ only if there exists some $v \in \Delta$ such that $I(v) = l$.* □

Intuitively, the above says that the interpretation of the (unary) functions and relations of the two models are precisely the same for all elements in the universe that are not interpretations of the variables in Δ.

An FOL+lfp formula over Σ, $\varphi(\boldsymbol{x})$, is a delta-logic formula if the formula does not distinguish between models that differ only on Δ:

Definition 2 (Delta-logic formulas). *An FOL+lfp formula over Σ, $\varphi(\boldsymbol{x})$, is a delta-logic formula if for every two Σ-models M and M' with the same universe and every interpretation I such that (M, I) and (M', I) differ only on Δ, $M, I \models \varphi$ iff $M', I \models \varphi'$.* □

Delta-logic formulae can be easily written using syntactic restrictions where the formula (and the recursive definitions) are written so that every occurrence of $f(t)$ (where f is a function symbol) or $P(t)$ (where P is a relation symbol), where t is a term of type σ_0, is guarded by the clause "$t \notin \Delta$", which is short for $\bigwedge_{v \in \Delta} t \neq v$).
Let us now give some examples of delta-logic formulae.

Example: Let us fix a finite set Δ of first-order variables.
The recursive definition

$$ls(x) :=_{lfp} (x = nil \vee (x \neq nil \wedge x \notin \Delta \wedge ls(n(x))) \vee (x \neq nil \wedge \bigvee_{v \in \Delta} (x = v \wedge b_v)))$$

is a delta-logic definition with respect to Δ. The above defines lists where by "imbibing" facts about whether a location v in Δ is a list using the free Boolean variable b_v. The definition says that x points to a list if it either is equal to the constant nil, or is not equal to nil and either x is not in Δ and $n(x)$ is a list *or x is in Δ, and the corresponding Boolean variable holds. The least fixpoing semantics of the above definition gives a unique definition: $ls(u)$ is true iff there is a path using the pointer $n()$ that either ends in nil or ends in a node v in Δ where b_v is true. Note that the formula $n(x)$ is guarded by the check $x \in \Delta$, and hence this is a delta-logic formula. Changing the model to reinterpret $n()$ over Δ (but preserving the interpretation of the variables b_v, $v \in \Delta$) will not change the definition of ls in any way.*

The formula $u \notin \Delta \wedge u'{=}n(u) \wedge ls(u')$ is a delta-logic formula that uses the above definition (note that the formula $n(u)$ is again guarded by a check ensuring u is not in Δ). Again, changing the interpretation of $n()$ on Δ will not affect the truth of this formula (provided the interpretation of u and u' do not change).

2 Translating Verification Conditions to Delta Logics

3 A Decidable Delta Logic on Lists with List Measures

In this section, we will define a delta logic on linked lists equipped with *list measures*— measures of list segments that include length, its heaplet, the multiset of keys stored in the list (say, in a data-field `key`), and the minimum and maximum keys stored in it. We prove that the quantifier-free first-order logic fragment of this delta logic is *decidable*. More precisely, we show that the logic can be translated to an equisatisfiable quantifier-free first-order formula that is decidable by using a Nelson-Oppen combination of decidable theories of arithmetic, sets, and uninterpreted functions. The translation in fact will allow us to decide formulas of the form $\alpha \wedge \beta$ obtained as verification conditions in the section above, where α is a delta-logic formula on lists and list measures and β is over a decidable Nelson-Oppen combinable quantifier-free theories, and where α and β share Boolean and first-order variables. At the end of the section, we outline some generalizations of our result.

A delta logic over list measures

As usual, let us fix a set of first-order variables Δ. Let us also fix a single pointer field n.

Definition 3 (Recursive Definitions for the Logic of List Measures (LM)). *Let us fix a set of* parameter variables *P that consists of a tuple of sets of variables: a set of Boolean variables LS^v_z, a set of variables with type set of locations HLS^v_z, a set of variables of type multiset of keys $MSKEYS^v_z$, and a set of variables of type integer Max^v_z and Min^v_z, where z, v range over Δ.*

The delta-logic of list measures (LM) wrt Δ and parameter variables P is defined using the following recursive definitions, which depend crucially on the parameter variables P.

– *We have unary relations ls^P_z that capture linked list segments that end in z, where the relation for a location v in Δ is imbibed using the Boolean variables*

LS^v_z ($v \in \Delta$), and where z is any element of Δ or the constant location nil. (The relation $ls_{nil}()$ captures whether a location points to a list ending with nil.)
This is defined as follows:

$$ls^P_z(x) :=_{lfp} \Big(x{=}z \vee \big(x \neq z \wedge x \neq nil \wedge x \notin \Delta \wedge ls^P_z(n(x)) \big) \vee \Big(x \neq z \wedge x \in \Delta \wedge \bigwedge_{v \in \Delta} (x = v \Rightarrow LS^v_z) \Big) \Big)$$

– *We have recursive definitions that capture the heaplet of such list-segments, where the heaplet of list-segments from an element v in Δ to z (where $z \in \Delta \cup \{nil\}$) is imbibed from the set variable HLS^v_z:*

$$\begin{array}{lrl} hls^P_z(x) :=_{lfp} & \emptyset & \textit{if } [\![x]\!]{=}[\![z]\!] \\ & \{x\} \cup hls^P_z(n(x)) & \textit{if } [\![x]\!] \neq [\![z]\!] \wedge [\![x]\!] \neq [\![nil]\!] \wedge [\![x]\!] \notin [\![\Delta]\!] \\ & HLS^v_z & \textit{if } [\![x]\!] \neq [\![z]\!] \wedge [\![x]\!] = [\![v]\!] \wedge v \in \Delta \end{array}$$

– *We have recursive definitions that capture the multiset of data elements (through a data-field key) stored in list segments, where again the multiset of data of list-segments from an element v in Δ to z (where $z \in \Delta \cup \{nil\}$) is imbibed from the set variable $MSKeys^v_z$:*

$$\begin{array}{lrl} mskeys^P_z(x) :=_{lfp} & \emptyset & \textit{if } [\![x]\!]{=}[\![z]\!] \\ & \{key(x)\} \cup_m mskeys^P_z(n(x)) & \textit{if } [\![x]\!] \neq [\![z]\!] \wedge [\![x]\!] \neq [\![nil]\!] \wedge [\![x]\!] \notin [\![\Delta]\!] \\ & MSKeys^x_z & \textit{if } [\![x]\!] \neq [\![z]\!] \wedge [\![x]\!] = [\![v]\!] \wedge v \in \Delta \end{array}$$

– *We have recursive definitions that capture the maximum/minimum element of data elements stored in list segments, where again the maximum/minimum element of list-segments from an element v in Δ to z where $z \in \Delta \cup \{nil\}$) is imbibed from the data variable Max^v_z (or Min^v_z). We assume the data-domain has a linear-order $\leq$, and that there are special constants $-\infty$ and $+\infty$ that are the minimum and maximum elements of this order. Let $max(r_1, r_2) \equiv ite(r_1 \leq r_2, r_2, r_1)$.*

$$\begin{array}{lrl} Max^P_z(x) :=_{lfp} & -\infty & \textit{if } [\![x]\!]{=}[\![z]\!] \\ & max(key(x), Max^P_z(n(x))) & \textit{if } [\![x]\!] \neq [\![z]\!] \wedge [\![x]\!] \neq [\![nil]\!] \wedge [\![x]\!] \notin [\![\Delta]\!] \\ & MSKeys^v_z & \textit{if } [\![x]\!] \neq [\![z]\!] \wedge [\![x]\!] = [\![v]\!] \wedge v \in \Delta \end{array}$$

The function Min^P_z is similarly defined.

– *We have a recursive definition that captures sortedness, using the minimum measure.*

$$Sorted^P_z(x) :=_{lfp} \Big(\; x{=}z \; \vee \\ \big(x \neq z \wedge x \neq nil \wedge x \notin \Delta \wedge min^P_z(x) \neq \bot \wedge key(x) \leq min^P_z(x) \wedge Sorted^P_z(n(x)) \big) \vee \\ \Big(x \neq z \wedge x \in \Delta \wedge min^P_z(x) \neq \bot \wedge key(x) \leq min^P_z(x) \wedge \bigwedge_{v \in \Delta} (x = v \Rightarrow SORTED^v_z \Big) \Big)$$

The above definitions can be written in usual syntax using *ite* expressions; we omit this formulation.

We define the logic of list-measures (LM) to be quantifier-free formulas that use only the recursive definitions of LM mentioned above; the subformulae of the formula in LM are however allowed to refer to *different* sets of parameter variables.

3.1 Deciding the logic of list measures

We can now state the main result of this section:

Theorem 1. *Given a quantifier-free formula $\varphi(\boldsymbol{x})$ in LM with recursive definitions of ls and measures of length, heaplet, keys, max, and min, there is an effective procedure that constructs a quantifier-free FOL formula $\psi(\boldsymbol{x})$ over a decidable Nelson-Oppen combination of theories of quantifier-free Presburger arithmetic, sets with cardinality constraints, and uninterpreted functions such that for* any *interpretation of the variables $\boldsymbol{x}$, there is a model that satisfies φ iff there is a model that satisfies ψ.*

Corollary 1. *The satisfiability problem for quantifier-free LM formulas is decidable.*

Let us fix a set of sets of parameters $\mathcal{P} = \{P_1, \ldots P_k\}$ (we encourage the reader to fix $k = 2$ in their mind while reading the section, as it's the most common and the logic VCs translate to, as shown in Section **??**).

We will first describe the decision procedure and its proof of correctness for the fragment of LM that involves only the three recursive definitions ls_z^P, hls_z^P, and $rank_z^P$, where $P \in \{calP\}$, which we will refer to as $LM[ls, hls, rank]$. Then we will extend the procedure to handle the logic with all the other measures; this latter proof requires more expressive decision procedures and pseudo-measures that make its proof harder.

Let us assume a quantifier-free $LM[ls, hls, rank]$ formula φ which is a Δ-logic formula wrt a finite set of variables Δ. Assume the (free) location variables occurring in φ is $X = \{x_1, \ldots x_n\}$ with $\Delta \subseteq X$.

In order to determine whether there is a model satsifying φ, we need to construct a universe of locations, an interpretation of the variables in X, and the heap (with the single pointer field $n()$) on all locations *outside* Δ (the definition of $n()$ on Δ, by definition, does not matter).

Our decision procedure intuitively relies on the following observations. First, note that the locations reached by using the $n()$ pointer any number of times forms the relevant set of locations that φ's truth can depend on (as φ is quantifier-free and has recursive definitions that only use the n-pointer). When pursuing the paths using the n-pointer on a location x, there are three distinct cases that can happen: (a) the path may reach a node in Δ, (b) the path may reach a node that is reachable also from another location in X, or (c) the path may never reach a location in Δ nor a location that is reachable from another location in X.

The key idea is to *collapse* paths where the reachability of them from variables in X does not change. More precisely, let L be the set of all locations reachable from X such that l is in Δ or for every location l' reachable from X such that $n(l') = l$, the set of nodes in X that have a path to l' is different from the set of nodes in X that have a path to l.

It is easy to see that there are at most $|X|-1$ locations of the above kind that are distinct from Δ, since the paths can merge at most $|X| - 1$ times forming a tree-like structure. Our key idea is now to represent these list segments that connect these kinds of locations *symbolically*, summarizing the measures on these list segments. Since there are only a bounded number of such locations and hence list segments, we can compute recursive definitions of linear measures involving them using quantifier-free and recursive-definition-free formulae.

We construct a formula ψ that is satisfiable iff φ is satisfiable, as follows. First, we fix a new set (distinct from X) of location variables $V = v_1, \ldots v_{|X|-1}$, to stand for the merging locations described above. We introduce an uninterpreted function $T : V \cup (X \setminus \Delta) \longrightarrow V \cup X \cup \{\bot\}$. Let Z be the set of variables in Δ as well as the variables in $X \setminus \Delta$ such that the recursive definitions ls^P_z, hls^P_z, $rank^P_z$, for some $P \in \mathcal{P}$, occurs in φ.

ψ is the conjunct of the following formulas:

– The formula φ (but with recursive definitions treated as uninterpreted relations and functions).
– For every $z \in Z$, we introduce an uninterpreted function $Dist_z : V \cup (X \setminus \Delta) \longrightarrow \mathbb{N} \cup \{\bot\}$ that is meant to capture the distance from any location in $V \cup X$ to z, if z is reachable from that location without going through Δ, and is $\bot$ otherwise. We add the constraint:

$$\bigwedge_{v \in V \cup (X \setminus \Delta)} \big[(Dist_z(v) = 0 \Leftrightarrow v = z) \ \wedge$$

$$v \neq z \Rightarrow \big(\ ((T(v) = \bot \vee Dist_z(T(v)) = \bot) \Rightarrow Dist_z(v) = \bot)$$
$$\wedge\ ((T(v) \neq \bot \wedge Dist_z(T(v)) \neq \bot) \Rightarrow Dist_z(v) = Dist_z(T(v)) + 1\big) \ \big) \big]$$

– For every $x \in X$, and for every $P \in \mathcal{P}$, we have a conjunct:

$$ls^P_z(x) \Leftrightarrow (Dist_z(x) \neq \bot \vee \bigvee_{v \in \Delta} (Dist_v(x) \neq \bot \wedge LS^v_z))$$

– For every $x \in X$, $z \in Z$, and for every $P \in \mathcal{P}$, we have a conjunct:

$$\big(Dist_z(x) = \bot \Rightarrow rank^P_z(x) = \bot\big) \ \wedge \ \big(Dist_z(x) \neq \bot \Rightarrow rank^P_z(x) = RANK^P_z\big)$$

– We capture the heaplets of list-segments from $v \in V \cup (X \setminus \Delta)$ to $T(v)$ (excluding both end-points) using a multiset of locations $H(v)$ and constrain it so that it does not contain any elements in X:

$$\bigwedge_{x \in X, v \in V \cup (X \setminus \Delta)} x \notin H(v)$$

– We can then precisely capture the heaplet $hls_z^P(x)$ by taking the union of all heaplets of list segments lying on its path to z. We do this using the following constraint, for each $v \in X \cup V$:

$$(Dist_z(v) = \bot \Rightarrow hls_z^P(v) = \emptyset) \wedge (hls_z^P(z) = \emptyset) \wedge$$

$$(Dist_z(v) \neq \bot \wedge v \neq z) \Rightarrow hls_z(v) = H(v) \cup \{T(v)\} \cup hls_z(T(v))$$

Note that the formula ψ is quantifier-free and over the combined theory of arithmetic, uninterpreted functions, and sets, and hence is decidable.

We can show the correctness of the above translation:

Lemma 1. *For any quantifier-free formula $\varphi(\mathcal{P}, X)$ of $LM[ls, hls, rank]$, the quantifier-free and recursion-free formula $\psi(\mathcal{P}, X)$ obtained φ obtained from the translation above satisfies the following property: for any interpretation of the free variables in $\mathcal{P} \cup X$, there is a model for φ iff there is a model for ψ.*

We now turn to the more complex logic $LM[ls, hls, rank, len, MSKeys, Min, Max, Sorted]$, and show that any quantifier-free formula φ in the logic can be satisfied. First, we model the multi-set of keys, minimum and maximum values and sortedness of each list-segment from v to $T(v)$ (where $v \in (X \setminus \Delta) \cup V$), which is outside Δ, using multiset variables $M - MSKeys(v)$, integer variables $M - Min(v)$, $M - Max(v)$, and $M - Len(v)$ and Boolean variables $M - Sorted(v)$. We can also aggregate them, as above, to express the sets $MSKeys_z(x)$, $Min_z(x)$, $Max_z(x)$, $Len_z(x)$ and $Sorted_z(x)$, for each $z \in Z$ and each $x \in (X \setminus \Delta) \cup V$, similar to definitions of $hls_z(x)$ as defined above. One point to note is that the recursive definition of sortedness across segments will by expressed by using both $Min_z(x)$ and $Max_z(x)$ definitions, though the definition of sortedness is defined using only minimum— this is needed as expressing when concatenation of list segments is sorted requires the max value of the first segment. We skip these definitions as they are easy to derive.

The main problem that remains is in *constraining* these measures so that they can be the measures of the *same* list segment. The following constraints capture these constraints, for each $v \in (X \setminus \Delta) \cup V$:

– The cardinality of $M - MSKeys(v)$ must be $M - Len(v)$.
– $M - Min(v)$ and $M - Max(v)$ must be the minimum and maximum elements of $M - MSKeys(v)$.
– If $M - Min(v) = M - Max(v)$ (and they are not $\bot$), then $M - Sorted(v)$ cannot be false.

The intuition is that any measures meeting the above constraints can be realized using true list segments. As for the third clause above, notice that any list segment with minumum element different from maximum can be realized using either a sorted list or an unsorted list.

The above measures, though seemingly simple, are hard to shoehorn into existing decidable theories. The first constraint can be expressed using quantifier-free BAPA [] (Boolean Algebra with Presburger arithmetic) constraints, which is decidable. We can get around defining the minimum of list segments by having the set of keys store only offsets from the minimum (and including the key 0 always). However, capturing max and sortedness measures in addition while preserving decidability seems hard.

Consequently, we give a new decision procedure that exploits the setup we have here. First, note that we can restrict the formulas that use the keys stored in sets to involve only membership testing of free variables in them, combinations using union and intersection, and checking emptiness of derived sets. We can *disallow checking non-emptiness* as non-emptiness of a set S can always be captured by demanding $k \in S$, for a freshly introduced free variable k without affecting satisfiability.

Our primary observation is that we can then restrict the multiset of keys to be over a *bounded* universe of elements, involving at most one element for each free variable mentioned in the formula, plus one extra element ν.

4 Separability Theorem

In this section, we show a key result: that for any quantifier-free FO+*lfp* formula we can effectively find an equivalent delta-logic formula. We do this by reasoning separately with the elements of the formula that are specific to Δ, and those that are oblivious to Δ. We bring these separate analyses together with a set of parameters that we shall describe and justify below.

First, let us consider a recursively defined function R, with the restricted in form as above **??**. Let the set of functions $PF = \{p_i \mid 1 \leq i \leq k\}$ (for some k) model the pointer fields (we assume that have a clause $p_i(nil) = nil$ for every $1 \leq i \leq k$). We also assume that Δ is fixed for this discussion.
We define a set of variables $\{R^d \mid d \in \Delta\}$ of the type of the range of R. These variables are in the set of parameters P. Then if R is defined as $R(x) :=_{lfp} \varphi(x)$ we define a new function corresponding to R, namely R^P, that is recursively defined as follows:

$$R^P(x) :=_{lfp} \quad \begin{array}{rll} R^d & \mathit{if}\ [\![x]\!] = [\![d]\!] \text{ for some } d \in \Delta & \text{(delta case)} \\ \varphi[R^P/R] & \mathit{if}\ [\![x]\!] \notin [\![\Delta]\!] & \text{(recursive case)} \end{array}$$

It is easy to see that for a formula $R(x)$, writing it as $R^P(x)$ would be a formula in context-logic since any model of it would not depend on a valuation of PF over Δ.

To capture the semantics of the original *lfp* definition, we constrain these parameters. This will yield definitions that are equivalent in FO+*lfp* under such constraints.
We do this by writing constraints that, effectively, unfold the recursive definition over Δ. However in doing so we would run into a problem with cycles. Consider

the case of evaluating a list predicate at a node by simply imbibing the value from the node pointed to when we have a circular list.
We handle this by introducing the notion of the 'rank' of a location w.r.t R. In particular, for the example of a circular list, if we recursively defined rank as a natural number increasing on a list starting from 0 at the location *nil*, there is no way to provide a valuation of every element on the cycle as pointing to a list. However, since the rank will need to communicate through the elements outside Δ to maintain this order (pointer paths between elements interpreted in Δ need not lie within it), it will also be a similarly relativised *lfp* definition with a set of parameters $\{RANK_R^d \mid d \in \Delta\}$ which are also included in P. We choose to model the rank as a function to $\mathbb{N} \cup \{\, bot\}$ ($\perp$ signifies undefined rank) as follows given recursively defined function R:

$$\mathit{Rank}_R(x) :=_{\mathit{lfp}} \qquad RANK_R^d \ \mathit{if}\ [\![x]\!] = [\![d]\!] \text{ for some } d \in \Delta$$

$$\text{(delta case)}$$

$$0 \ \mathit{if}\ [\![x]\!] \notin [\![\Delta]\!] \wedge R^P(x) \neq \perp \wedge \bigvee_{1 \leq i \leq k} \big(R^P(p_i(x)) = \perp\big)$$

$$\text{(base case)}$$

$$\max_{1 \leq i \leq k} \{\mathit{Rank}_R(p_i(x))\} \ \mathit{if}\ [\![x]\!] \notin [\![\Delta]\!] \wedge R^P(x) \neq \perp$$

$$\text{(recursive case)}$$

$$\perp \ \mathit{if}\ [\![x]\!] \notin [\![\Delta]\!] \wedge R^P(x) = \perp \ \text{(undefined)}$$

Finally, we define some more parameters in P in similar vein as above to communicate between the constraints within Δ and the new *lfp* definitions that are in context-logic. This will be on variables in $p_i(\Delta) \cap \Delta^c$ for some i, and are therefore named thus:
Let $R^{p_i(\Delta)} = \{R^{p_i(d)} \mid d \in \Delta\}$ of the type of range of R
and $RANK_R^{p_i(\Delta)} = \{RANK_R^{p_i(d)} \mid d \in \Delta\} \subseteq \mathbb{N} \cup \{\perp\}$
for every $1 \leq i \leq k$.

We then denote the substitution $\varphi(x)[P_R/R]$ as replacing the term $R(p_i(x))$ with $R^{p_i(x)}$ for every $1 \leq i \leq k$, and $\varphi(x)[\perp/R]$ as replacing with $\perp$. With the above, we write the following delta-specific constraint β_R for a recursively defined function R:

$$\bigwedge_{d\in\Delta} \Big[\Big(\varphi(d)[\bot/R] \neq \bot \implies R^d = \varphi(d)[\bot/R] \wedge \Big(RANK_R^d = 0\Big)\Big) \quad \text{(base case)}$$

$$\wedge \ \big((\varphi(d)[\bot/R] = \bot \wedge \varphi(d)[P_R/R] \neq \bot) \implies \big(R^d = \varphi(d)[P_R/R]$$

$$\wedge \Big(RANK_R^d = \max_{1\leq i\leq k}(\{RANK_R^{p_i(d)}\}) + 1\Big)\Big)\Big) \qquad \text{(recursive case)}$$

$$\wedge \ \Big((\phi(d)[\bot/R] = \bot \wedge \phi(d)[P_R/R] = \bot) \implies \Big(R^d = \bot \wedge \Big(RANK_R^d = \bot\Big)\Big)\Big) \Big] \quad \text{(undefined)}$$

The above constraints capture the values of R accurately on Δ:

– the base case simply constrains the parameter at a node interpreting its corresponding variable to be the value provided by the function definition, and its rank to be 0 when the interpretation for that variables satisfies the base case of the recursive definition.
– the recursive case constrains the parameter (when it does not satisfy the base case) to be the value computed by one unfolding of the definition, where the values of the descendants are also denoted by their respective parameters (whether Δ or boundary) and its rank to be one more than the maximum rank among its descendants.
– the undefined case constrains the parameter to be undefined when it must be according to an unfolding of the definition, and its rank to be undefined as well.

Lastly, we must also have that the 'boundary' does in fact communicate the values of the context-logic recursive definition to Δ, i.e that the placeholder parameters for their values are indeed the values provided by the context-logic *lfp* definition:

$$\bigwedge_{1\leq i\leq k} \big[p_i(d) \notin \Delta \implies \Big(R^{p_i(d)} = R^P(p_i(d))\Big)$$

$$\wedge \Big(RANK_R^{p_i(d)} = Rank_R(p_i(d))\Big) \Big]$$

These constraints ensure that the values of the function are computed accurately on Δ.

We are now ready to state the main theorem, the proof of which will use the following lemma. Let P, which we shall write as P_R to indicate the function, include only the sets of parameters defined hitherto. Then:

Lemma 2. *For any recursively defined function R,* $(\exists P_R.\ \beta_R) \wedge \big(\forall P_R.\ \big(\beta_R \implies R^{P_R} = R\big)\big)$

If this is true then we have for any FO+*lfp* formula α an equivalent delta-logic formula. Let the set of recursive functions/predicates mentioned in α be $\mathcal{R}$, and Δ be fixed. Let $\mathcal{R}^P = \{R^{P_R} \mid R \in \mathcal{R}\}$, and $P_{\mathcal{R}} = \bigcup_{R \in \mathcal{R}} P_R$. Then:

Theorem 2 (Separability). $\alpha \equiv \exists P_{\mathcal{R}}.\, \alpha[\mathcal{R}^P/\mathcal{R}] \wedge \left(\bigwedge_{R\in\mathcal{R}} \beta_R\right)$.

Proof. Consider that α holds. From lemma 2, for every $R \in \mathcal{R}$, we can pick P_R such that β_R and, therefore, $R^{P_R} = R$. Thus, we have that $\alpha[\mathcal{R}^P/\mathcal{R}] \wedge \left(\bigwedge_{R\in\mathcal{R}} \beta_R\right)$ holds.

Conversely, let $\alpha[\mathcal{R}^P/\mathcal{R}] \wedge \left(\bigwedge_{R\in\mathcal{R}} \beta_R\right)$ hold. Again, from lemma 2 we have that for every $R \in \mathcal{R}$, the valuation given by the model for P_R satisfies β_R, and therefore $R^{P_R} = R$. Therefore, $\alpha[\mathcal{R}^P/\mathcal{R}][\mathcal{R}/\mathcal{R}^P] = \alpha$ holds.

Observe that the latter formula is a formula in delta-logic, a boolean combination of context-logic formulae and delta-specific formulae.

5 Translating VCs to Delta Logic

Let us consider the Hoare Triple: $(\alpha_{pre},\, T,\, \alpha_{post})$ such that α_{pre} and α_{post} are FO+*lfp* formulae. The program manipulates pointers and data fields, which we model using unary functions. We can then write α_{pre} over sets of variables, fields and recursively defined functions X, P and R respectively and similarly α_{post} over sets X', P' and R' such that for every $x \in X$, there is a corresponding $x' \in X'$ abd similarly for $f \in P$, there is a corresponding symbol $f' \in P'$. Intuitively, this is used to identify the values of program variables and distinguish the state of the pointer/data fields in the universe after the execution of the program. Consequently, for every recursively defined function $r \in R$, there is a corresponding $r' \in R'$ such that $r' = r[P'/P]$ is a substitution of f' for the corresponding f in the definition of r.

We can also write T, the formula describing the program transformation, over $X \cup X' \cup X_{tmp}$ and $P \cup P'$ such that $\Delta \subseteq X \cup X' \cup X_{tmp}$, where $T = T_1 \wedge T_2$ can be written as a conjuction of:

- A quantifier-free formula T_1 such that any subterm of the form $f(t)$ for some $f \in P \cup P'$ and some term t must have $t = v$ for some $v \in \Delta$ (this is to ensure that the pointer and data fields are only referenced at variables in Δ), and
- A formula T_2: $\bigwedge_{f\in P} (\forall z.\, z \notin \Delta \implies f'(z) = f(z))$ (this describes that the pointer and data fields are changed only on variables in Δ).

This is possible since the program changes the values of the data and pointer fields on elements only within Δ, and the values of the variables in the state resulting after the program execution can be written as expressions of the values in the state before, with only a finite number of temporary variables.

It is clear that the VC that captures the goven Hoare Triple will be $\alpha_{pre} \wedge T \wedge \neg\alpha_{post}$, which by the above is a formula in FO+*lfp* within our given signature. From Theorem 2, we have that an FO+*lfp* formula can be written equivalently as a delta-logic formula. Therefore, α_{pre} and α_{post} can be rewritten to equivalent delta-logic.
However, from the theorem we will also see in particular that the resursive definitions of functions and predicates in α_{post} use functions from P' only on arguments in Δ^c. Therefore, from T_2 we have that these can be replaced with corresponding functions in P since the pointer and data fields of (locations interpreted by) variables not in Δ are unaltered by the program. Then, we also remove T_2 since the symbols in P' are no longer referred to anywhere else on an argument not in Δ.

Therefore, the VC can be written equivalently as a delta-logic formula (since T does not contain any resursively defined functions as subterms). Since delta-logic formulae are boolean combinations of context-logic fomulae and delta-specific formulae, satisfiability of the VC then becomes the meaningful question of asking independently for a model of a context, a prior state and a resulting state (with a common valuation for finitely many shared first-order variables), such the context when applied over a model of the prior state satisfies the precondition, and applied over a model of the resulting state falsifies the postcondition.

References