CS 476 Lecture 28 4/20/17

J. Meseguer

1. Cleaning up the Mess of Recursive Data Structures

Typical "non-cyclic" recursive data structures include lists, trees, stacks, and so on. The seminal paper about them, both in terms of their formalization in an untyped setting and also in terms of decidable satisfiability is Derek Oppen's JACM, 27, 403-411, 1980 paper "Reasoning about Recursively Defined Data Structures".

The Mess. The whole area is conceptually a mess. The reason is the following. The function symbols of any such recursive data structure are split into:

1. Free constructors Σ, which build the data structures and obey no axioms, and
2. Selector functions Δ, which decompose a data structure into smaller components

Of course we want the selector functions to be defined functions that disappear after evaluating each term to its canonical form, so that the specification is sufficiently complete, so that the so-called constructors are really so.

The problem is that even for simple data structures such as lists this is <u>impossible</u> in a many-sorted setting and, a fortiori, even more impossible in an unsorted setting such as the one that Obj3 use, where he gives strange equations like

$$head(nil) = nil$$

that would not even type in a typed setting.

Let us see the impossibility for lists; but the same can be shown for trees or for stacks. The constructors are nil and "cons" $-\bullet-$: Elt List → List. And the selectors are head : List → Elt, and tail : List → List, called CAR and CDR for LISP lists.

The problem is that there is <u>no</u> satisfactory way of defining head(nil) as a constructor term of sort Elt.

The only possible way is to <u>postulate</u> the existence of a constant, say, <u>no-head</u> of sort Elt, belonging to S.

But no suppose we view lists as a parameterized data type List[X]. Then how is this constant going to be interpreted for each non-empty set A instantiating the parameter X? And in what sense will it be a constructor?

<u>The Unsatisfactory Way Out</u>. Current approaches to satisfiability of recursive data structures adopt the way out of declaring the value of selector expressions for problematic

cases such as head(nil) to be underspecified. Specifically, this means that the parameterized data type

$$(\Sigma_{LIST}, List[X])$$

does not have a single expansion of, say, a set of elements A to List[A], but as many as elements $no\text{-}head_A \in A$, each providing a different model (a different expansion).

This is clearly a mess. Suppose $A = \mathbb{N}$. Now consider the QF formulas: $1+1+1 = head(nil)$, $0 = head(nil)$,

$$x+1 = head(nil) \vee x+1+1+1 = head(nil)$$

is this formula satisfiable? How about their negations?

$1+1+1 \neq head(nil)$,
$0 \neq head(nil)$, $\quad x+1 \neq head(nil) \wedge x+1+1 \neq head(nil)$

It turns out all are satisfiable if just any choice of $no\text{-}head_{\mathbb{N}} \in \mathbb{N}$ is allowed. But of course if another tool has assumed that $no\text{-}head_{\mathbb{N}} = 0$, then some formulas (2nd, 4th, and 6th) are satisfiable and others not.

The upshot is that it is anybody's guess what an SMT solver will give as an answer to questions about the satisfiability of QF formulas such as the above, particularly if the have not adopted the convention of leaving the value of head(nil) underspecified but have instead adopted some other convention.

Furthermore, lists are supposed to be a data type, and in any data type worth its salt any expression should be either completely and uniquely defined, or be a type error, so there should be no ambiguity about validity of formulas. But validity of formulas becomes totally ambiguous in underspecified "data types".

The Ignored Solution. The entire problem was solved by Meseguer and Goguen a long time ago: J. Meseguer and J. A. Goguen, "Order-Sorted Algebra Solves the Constructor-Selector, Multiple Representation, and Coercion Problems," Information and Computation, 103, 114-158 (1993). The trivial observation is that head and tail are partial functions that become total in an order-sorted setup in which we add a Nelist subsort of non-empty lists. In general, if $c : A_1 \cdots A_n \longrightarrow B$ is a constructor, we can add a subsort $B_c < B$, and selector operators $\text{sel-}c_i : B_c \to A_i$, $1 \leq i \leq n$, defined by equations

$$\text{sel-}c_i\,(\,c\,(X_1 : A_1, \ldots, X_n : A_n)) = X_i : A_i$$

This is of course not the only option. For example, we could define an alternative solution for lists of the form:

```
fmod LIST{X :: TRIV} is
    sorts List{X}  Elt?{X} .
  subsort  X$Elt < Elt?{X} .
  op  no-head : ⟶ Elt?{X} [ctor].
  op  nil : ⟶ List{X} [ctor].
  op  _._ : X$Elt  List{X} ⟶ List{X} [ctor].
  op  head : List{X} ⟶ Elt?{X} .
  op  tail : List{X} ⟶ List{X}.
  var E : X#Elt .   var L : List{X}
  eq  head(E.L) = L .
  eq  head(nil) = no-head .
  eq  tail(E.L) = L .
  eq  tail(nil) = nil .
endfm
```

The key point however, is that in both cases subsorts allow a full definition of the desired data type, so that once we have fully specified such a data type there is nothing under-specified or ambiguous about the satisfaction of any QF formula. In this way no ambiguity whatsoever exists about validity of formulas in a theory of recursive data structures so defined. We will see that this also applies to parameterized ones because they have always a generic model, so any QF formula is either valid or not.

General

A Precise Notion of Recursive Data Structure Solving the Problem

specification

Definition. A recursive data structure is a possibly parameterized order-sorted equational theory of the form:

$$P \xrightarrow{J} B$$

where:

1. P consists of n disjoint copies of the TRIV theory (the case $n=0$ yields unparameterized recursive data structures like NAT in constructors 0 and s and selector p). Let $Elt_1, .., Elt_n$ be the sorts of P

2. Σ_B is a disjoint union $\Sigma_B = \Omega_B \dot{\cup} \Delta_B$ where each $c \in \Omega$ is called a constructor, and each $sel \in \Delta_B$ is called a selector.

3. The equations E_B of B are all of the form either:

   3.1 $sel(C(x_1, .., x_m)) = x_i$ for some $1 \leq i \leq n$, or

   3.2 $sel(a) = b$, for a, b constants in Ω.

4. The equations E_B are confluent, terminating and sufficiently complete (in the parameterized sense explained in J. Meseguer, "Order-Sorted Parameterization and Induction", Springer LNCS 5700, 43-80, 2009.

5. Each sort Elt_i in P is a minimal element in the sort poset $(S_B, \leq_B)$ of B, and for any $C : w \to s \in \Omega$, $s \neq Elt_i$, $1 \leq i \leq n$.

Here is a crucial observation:

Theorem. If $P \xrightarrow{J} B$ is a recursive data structure specification, then B has the finite variant property

Proof. All constructor operations $C(x_1 .. x_n)$ have themselves only variants. Given a selector sel its only possible variants are:

$$(sel(X), id), \quad (X_i, \{X \mapsto C(X_1 .., X_n)\}), \quad (b, \{X \mapsto a\}). \square$$

Definition. Given a recursive data structure specification or a (possibly parameterized) theory $P \xrightarrow{J} B$, say with $P = TRIV * \{Elt \to Elt_1\} \oplus \cdots \oplus TRIV * \{Elt \to Elt_n\}$ its semantics is the parameterized data type

$$(\Sigma_B, \mathbb{F}_J)$$

where $\mathbb{F}_J = \{ \mathbb{F}_J[A_1, .., A_n] \mid A_i \in Set_{1\phi}, 1 \le i \le n \}$.

Theorem. For $(\Sigma_B, \mathbb{F}_J)$ the semantics of a recursive data structure, $\mathbb{F}_J[A_1, .., A_n]$ is a generic model for QF satisfiability for A_i such that $|A_i| = \aleph_0$, $1 \le i \le n$.

Proof. Suppose $\varphi \in QF(\Sigma_B)$ is satisfiable in some $\mathbb{F}_J[B_1, .., B_n]$ with assignment $a \in [Y \to F_J[B_1, .., B_n]]$ where $Y = vars(\varphi)$. Define $[C_1, .., C_n]$ as follows

$$C_i = \{ b \in B_i \mid b \leq a(y)!_{E_B} \wedge y \in Y \} \cup \{ b_i^0 \}$$

where $b_i^0 \in B_i$ is arbitrarily chosen to make sure $C_i \neq \emptyset$. Note that then we have a retract (r) right inverse to the inclusion $j : [C_1, .., C_n] \hookrightarrow [B_1, .., B_n]$, so that $j ; r = 1_{[C_1, .., C_n]}$, with the $C_1, .., C_n$ finite non-empty sets. Furthermore, we have a factorization

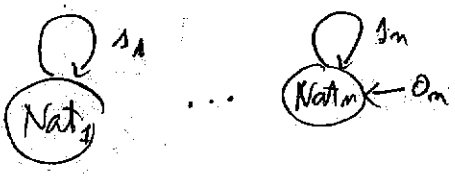
$$\begin{array}{ccc} T_{\Sigma_B}(Y) & \xrightarrow{-a} & \mathbb{F}_J[B_1, .., B_n] \\ & \searrow \quad \nearrow_{\mathbb{F}_J(j)} & \\ & \mathbb{F}_J[C_1, .., C_n] & \end{array}$$

so that $\mathbb{F}_J[C_1, .., C_n], a \models \varphi$. But then, we can always include each C_i into a set A_i with $|A_i| = \aleph_0$ to get an inclusion $[C_1, .., C_n] \xhookrightarrow{j'} [A_1, .., A_n]$ and a retract map r' with $j' ; r' = 1_{[C_1, .., C_n]}$, and since $\mathbb{F}_J$ is a functor this gives us a submodel inclusion $\mathbb{F}_J[C_1, .., C_n] \subseteq \mathbb{F}_J[A_1, .., A_n]$ so that by lemma 1 we have $\mathbb{F}_J[A_1, .., A_n], a \models \varphi$, as desired. □

Note that for any $[A'_1, .., A'_n]$ with $|A'_i| = \aleph_0$ we have a bijection $h : [A'_1, .., A'_n] \xrightarrow{\cong} [A_1, .., A_n]$ and therefore an isomorphism $\mathbb{F}_J[A'_1, .., A'_n] \overset{\mathbb{F}_J(h)}{\cong} \mathbb{F}_J[A_1, .., A_n]$.

Corollary. If $P \overset{J}{\hookrightarrow} B$ is a (possibly unparametrized) recursive data structure specification, then satisfiability of QF Σ_B-formulas in the theory $(\Sigma_B, \mathbb{F}_J)$ is decidable.

Proof. If $P = \emptyset$ the result follows trivially from B beg FVP and Ω beg free constructors so that T_J is OS-compact. If P has sorts $Elt_1 .., Elt_n$, instantiate then to n-rename copies of the theory NAT consisting of the constructor signature $0 : \to Nat$, $s : Nat \to Nat$, renamed as $0_i : \to Nat_i$, $s_i : Nat_i \to Nat_i$, $1 \le i \le n$. Recall from Var Sat that this just means replacing P by the theory:

$NAT_1 \oplus \cdots \oplus NAT_n =$ 

In this way we get the theory $B[X_1 \mapsto NAT_1, .., X_n \mapsto NAT_n]$, which is both FVP and OS-compact and has decidable satisfiability of QF formulas for its initial algebra. Now note that

$$\left(\Sigma_B, \; T_{B[X_1 \mapsto NAT_1, .. X_n \mapsto NAT_n]} \Big|_{\Sigma_B}\right)$$

is precisely of the form $\mathbb{F}_J[\mathbb{N}_1, .., \mathbb{N}_n]$. That is, the result follows by restricting the Var-Sat satisfiability of $T_{B[X_i \mapsto NAT_i]}$ to the sublanguage $QF(\Sigma_B)$. □

Exercise. Consider the parameterized data type $\mathcal{L}[X]$ of lists defined in VarSet, and the theory inclusion (view)

view Triv2Toset from TRIV to TOSET
  sort Elt to Elt.
endv

Then from LIST{X :: TRIV} we can obtain

fmod ORD-LIST{T :: TOSET} is
  protecting LIST{Triv2Toset}
endfm

where we have replaced the TRIV parameter theory by TOSET.

1. Prove that List[$\mathbb{N}_{\leq}$] the ordered lists with $\mathbb{N}_{\leq}$ the standard order on the naturals is a generic model of ORD-LIST{T}. How would use exploit this fact in proving correctness of a parameterized/polymorphic sorting module like INSET-SORT{T :: TOSET}?

2. Generalized this result to any recursive data structure

   a) with P a single copy of TRIV

   b) with P have $n \geq 1$ copies of TRIV and one or several of them replaced by TOSET as above.

3. A More General Notion of Optimally Intersectable Signatures

We can relax the conditions in the Addendum to Lecture 24 to arrive at the following more general notion:

Definition The order-sorted signatures Σ_1 and Σ_2 are optimally intersectable, denoting as $[s_i]_i$ the connected component in $(S_i, \leq_i)$ of a sort $s_i \in S_i$, $1 \leq i \leq 2$, either condition (1) or (2) holds, and condition (3) holds:

either 1. $[s_i]_i \cap [s_j]_j \neq \emptyset$

$\{i,j\} \subseteq \{1,2\}$, and for $k \in \{1,2\}$ we have:

1.1 $\exists k \in \{1,2\}$, $[s_k]_k$ has a top sort, $[s_k]_k \subseteq [s_\ell]_\ell$,

$\{k,\ell\} = \{1,2\}$

1.2 $\leq_k \cap\, [s_k]_2^2 = \leq_\ell \cap\, [s_k]_2^2$

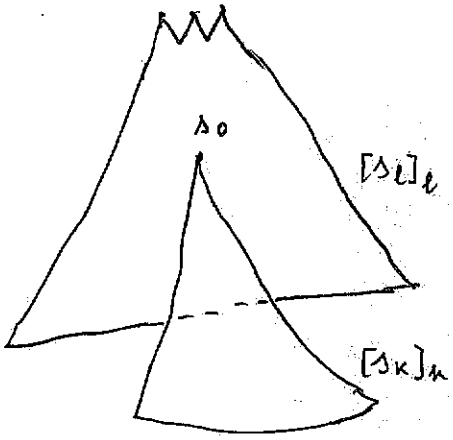
1.3. (downward closure). $\forall\, s \in [s_\ell]_\ell \;\forall\, s' \in [s_k]_k$,

$s \leq_\ell s' \Rightarrow s \in [s_k]_k$.

1.4 Uniqueness $[s_i]_i \cap S_j = [s_j]_j \cap S_i = [s_k]_k$

or 2. $[s_i]_i \cap [s_j]_j = \{s_0\}$, $\{i,j\} = \{1,2\}$, and

$\exists k \in \{1,2\}$ such that s_0 is top element of $[s_k]_k$



and furthermore:

(Uniqueness) $\quad [s_i]_i \cap S_j = [s_j]_j \cap S_i = \{s_0\}$

and 3. If $f \in fun(\Sigma_1) = fun(\Sigma_2)$ (resp. $p \in pred(\Sigma_1) \cap pred(\Sigma_2)$)

then $\exists\, i \in \{i,j\} = \{1,2\}$ such that:

3.1 If $(s_1, \dots, s_n, s) \in F_i(f)$ (resp. $(s_1, \dots, s_n) \in P_i(p)$)

then: • $F_i(f) = F_j(f) \cap \left([s_1]_i \times \dots \times [s_n]_i\right) \times [s_i]$

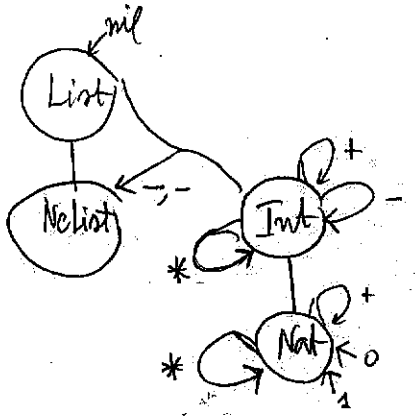
(resp. $P_i(p) = P_j(p) \cap [s_1]_i \times \dots \times [s_n]_i$)

• $[s_\ell]_i \subseteq [s_\ell]_j$ $\quad 1 \le \ell \le n$, and $[s]_i \subseteq [s]_j$

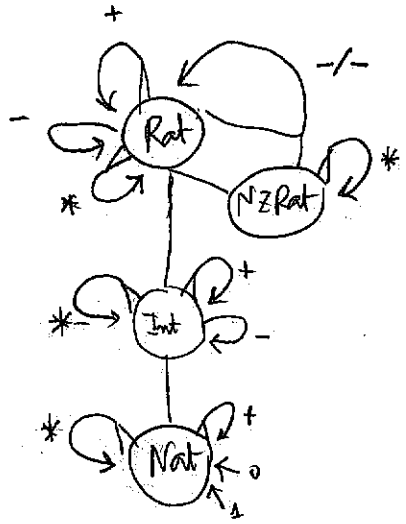
(resp. $[s_\ell]_i \subseteq [s_\ell]_j$ $\quad 1 \le \ell \le n$).

These conditions can be illustrated with the following example:

Σ_1:



Σ_2:



Exercise. Prove:

1. If Σ_1 and Σ_2 are optimally intersectable and $\Sigma_0 = \Sigma_1 \cap \Sigma_2$ then

$$\begin{array}{ccc} Mod(\Sigma_1) & \xleftarrow{-|_{\Sigma_1}} & Mod(\Sigma_1 \cup \Sigma_2) \\ {\scriptstyle -|_{\Sigma_0}}\downarrow & & \downarrow{\scriptstyle -|_{\Sigma_2}} \\ Mod(\Sigma_0) & \xleftarrow[-|_{\Sigma_0}]{} & Mod(\Sigma_2) \end{array}$$

is a pullback.

2. If Σ_0 consists only of sorts ($fun(\Sigma_1) \cap fun(\Sigma_2) = \emptyset$ and $pred(\Sigma_1) \cap pred(\Sigma_2) = \emptyset$), then $(1_{\Sigma_1 \cup \Sigma_2}, pair)$ is a correct ascent map.