

CS 476 Homework #7 Due 10:45am on 3/16

Note: Answers to the exercises listed below in *typewritten form* (latex formatting preferred) as well as code solutions should be emailed by the above deadline to `reedoei2@illinois.edu`.

1. Call a function $f : A^2 \rightarrow A$ *commutative* iff $(\forall (x, y) \in A^2) f(x, y) = f(y, x)$. Commutative binary functions on A define a subset $[A^2 \rightarrow A]^c \subseteq [A^2 \rightarrow A]$ of the function set $[A^2 \rightarrow A]$, namely,

$$[A^2 \rightarrow A]^c = \{f \in [A^2 \rightarrow A] \mid (\forall (x, y) \in A^2) f(x, y) = f(y, x)\}.$$

You are asked to do the following:

- Assume that A is finite with cardinality, say, $|A| = n$. Give a formula, depending on n , for the cardinality $|[A^2 \rightarrow A]^c|$ of the finite set $[A^2 \rightarrow A]^c$.

Hint. If $|A| = n$, then there is a bijective function $b : A \rightarrow [n]$, where $[n] = \{1, \dots, n\}$. Therefore, $f : A^2 \rightarrow A$ is commutative iff $(b^{-1} \times b^{-1}); f; b : [n]^2 \rightarrow [n]$ is commutative. Therefore, you can assume, without loss of generality, that $A = [n]$, and can just give a formula for $|[n]^2 \rightarrow [n]|^c$.

- Let Σ be a finite unsorted signature, i.e., for each arity k , the set Σ_k of function symbols of k arguments is finite, and $\Sigma_k = \emptyset$ for any $k > k_{max}$. That is, there are no function symbols with more than k_{max} arguments. Let $\Sigma_2^c \subseteq \Sigma_2$ be a subset of binary symbols. Assume $|A| = n$. Then, give a formula, depending on n , Σ and Σ_2^c , computing the total number of different Σ -algebras of the form $\mathbb{A} = (A, -_{\mathbb{A}})$ such that for each $f \in \Sigma_2^c$, the function $f_{\mathbb{A}}$ is *commutative*.

Remark on Terminology. In algebraic terminology, an algebra $\mathbb{A} = (A, -_{\mathbb{A}})$ such that for each $f \in \Sigma_2^c$, the function $f_{\mathbb{A}}$ is commutative is described as an algebras $\mathbb{A} = (A, -_{\mathbb{A}})$ that *satisfies the set of commutativity equations*: $\{f(x, y) = f(y, x) \mid f \in \Sigma_2^c\}$.

2. Consider the following module (available in the course web page) of lists with a list append functions that is associative and has an identity, but where associativity and identity are explicitly defined by equations:

```
fmod LIST-EXAMPLE is
  sorts Elt NeList List .
  subsorts Elt < NeList < List .
  op a : -> Elt [ctor] .
  op b : -> Elt [ctor] .
  op c : -> Elt [ctor] .
  op nil : -> List [ctor] .
  op _;_ : List List -> List .
  op _;- : Elt NeList -> NeList [ctor] .

  vars L P Q : List . vars R S T : NeList . vars X Y Z : Elt .
  vars L' P' Q' : List .
.

eq (L ; P) ; Q = L ; (P ; Q) .
eq L ; nil = L .
eq nil ; L = L .
endfm
```

The main goal of this exercise is to help you check your understanding of the algorithm that checks that a theory is confluent assuming it is terminating. The above equations are indeed terminating. You are not

required to prove this: as we shall see, there are methods and tool to do so. Instead, you are asked to prove, under the termination assumption, that the above equations E , when oriented as rules \vec{E} , are confluent. Recall that this requires checking two things:

- (a) The rules \vec{E} are sort-decreasing.
- (b) The rules \vec{E} are locally confluent, which by the Main Theorem in page 19 of Lecture 8 can be checked by checking that all the *critical pairs* associated to the rules \vec{E} are joinable.

For (a), since checking sort-decreasingness for the associativity equation involves a relatively long number of sort specializations, which can be tedious, you are asked to just check sort decreasingness for:

- The equation $L ; \text{nil} = L$ (for all its sort specializations).

Hint. You can use the variables of sorts `NeList` and `Elt` already declared in the module for this purpose. For example, the substitution $\{L \mapsto R\}$ specializes L of sort `List` to R of sort `NeList`.

- The equation $(L ; P) ; Q = L ; (P ; Q)$ only for the sort specialization $\{L \mapsto X, P \mapsto Y, Q \mapsto T\}$, where the sorts of the variables are as declared in the module.

Recall that to check sort decreasingness of an equation $u = v$ for a sort specialization ρ one has to check $ls(u\rho) \geq ls(v\rho)$. But this can be substantially automated by using the `parse` command in Maude. The `parse` command does not evaluate a term t : it just parses t with its least sort $ls(t)$. For example:

```
Maude> parse (a ; b) ; c .
List: (a ; b) ; c
Maude> parse a ; (b ; c) .
NeList: a ; (b ; c)
```

For (b), to make your life easier, note the $V = \text{vars}(E) = \{L, P, Q\}$, and that for $V' = \{L', P', Q'\}$ the renaming of variables $\gamma : V \rightarrow V'$ defined by: $\gamma = \{L \mapsto L', P \mapsto P', Q \mapsto Q'\}$, is both bijective and sort-preserving, and with $V \cap V' = \emptyset$. This allows you to easily produce renamed copies $E\gamma$ of the module's equations E .

Recall, also, that critical pairs are constructed by considering all pairs of equations $(u = v) \in E$ and $(u' = v') \in E\gamma$ (including the case $(u' = v') \equiv (u\gamma, v\gamma)$, where $u' = v'$ is just a variable renaming of $u = v$), all *non-variable positions* p in u , and all most general order-sorted unifiers θ of the equation $u_p = u'$.

In the case $B = \emptyset$ when, as in this example, there are no axioms, there are, however, some critical pairs that are *trivial*, that is, they are always joinable, namely, the case when $(u' = v') \equiv (u\gamma, v\gamma)$, and $p = \epsilon$ is the root position of u . You can omit those trivial cases of *self-overlap of a rule $u \rightarrow v$ with (a renamed version of) itself at the root position ϵ of u* . However, you should consider all other cases, including *all other cases of self-overlap of a rule with itself at non-root non-variable positions*.

A lot of your work can be automated, because of Maude's `unify` command computes the set of most general order-sorted unifiers of an equation. For example, we can compute the order-sorted unifiers of the equation: $(L ; P) ; Q = L' ; \text{nil}$ by typing in Maude:

```
Maude> unify (L ; P) ; Q =? L' ; nil .
```

```
Solution 1
L --> #1:List
P --> #2:List
Q --> nil
L' --> #1:List ; #2:List
```

which in this case has a single solution. Unifiers in Maude are computed mapping the variables of the given equation to fresh new variables introduced by Maude. Unifiers can always be expressed this way. Therefore, using the `unify` command you can automatically compute all the most general unifiers for equalities of the form: $u_p = u'$, and then use such unifiers to compute the corresponding critical pairs.

A further economy of effort, is that *you do not need to even compute* the unifiers of an equality $u_p = u'$ when the *top symbols* of u_p and u' are *different*. For example, the equation $L ; P = \text{nil}$ has no unifiers, so there is no need to even try to compute them.

Finally, you can also automate checking the joinability $t \downarrow_E t'$ of each critical pair thus computed, say $t = t'$, since you can just give in Maude the command:

```
red t == t' .
```

so that the critical pair will be joinable iff the result is **true** and not joinable iff the result is **false**.

A “hidden goal” of this exercise is to let you see how —thanks to subsorts and subsort overloading of operators— an operator like `_;` can be *both* a constructor with the typing `_;_ : Elt NeList -> NeList [ctor]` and a *defined symbol*, defined by the above three equations, with the typing `_;_ : List List -> List`. Of course, this is *impossible* to do in a many-sorted (also called “simply typed”) setting.

Note that this exactly means that the *only data* in this module are —as one would expect— the constructor terms, that is, *nil*, and the terms of the form $a_1; (a_2; \dots (a_{n-1}; a_n) \dots)$, with $a_i \in \{a, b, c\}$, $1 \leq i \leq n$, and $n \geq 1$. That is, (up to a slight change of representation) the data in this module is the set of strings $\{a, b, c\}^*$.

If you use Maude to help you in all these ways, you should include a screenshot of the answers you get from Maude for each problem you ask Maude to help you with.