AN ECONOMICAL METHOD FOR COMPARING DATA TYPE SPECIFICATIONS

J.L. Remy*
C.R.I.N., France

P.A.S. Veloso**
PUC-RJ, Brasil

Introduction

We present a simple method for checking the equivalence of two sets of rewriting rules alledgedly specifying the same abstract data type . The basic idea is comparison of normal formsin order to reduce the amount of work to be done.

Frequently when dealing with an abstract data type one has diverse specifications with distinct features. For instance, one may be easier to understand whereas another one may lead to more efficient implementations [Choppy, Lescanne, Remy'80]. This diversity of specifications often stems from different choices of constructors[Guttag, Horowitz, Musser'78]. In any case, one faces the problem of checking their equivalence, i.e. whether they specify the same abstract data type. In the terminology of [Goguen, Thatcher, Wagner'78], are the corresponding initial algebras isomorphic?

Example

In order to illustrate the method we shall employ the simple example of linear lists of atoms from D. We consider the following usual operations on lists: nil, cons, make (which makes the list <d> = cons(d,nil) out of atom d) and append (list concatenation).

It turns out that there are two distinct ways of constructing all lists by means of these operations. The first one uses only nil, make and append. Corresponding to this idea we have the specification S of fig. 1, with normal form F consisting of nil, make(d) for d in D, and append(make(d),f), for d in D and f in F, f ≠ nil.

The other specification, in fig. 2, uses only nil and cons to construct all lists, its normal form F' consisting of nil and cons(d,f) for d in D and f in F'.

Both specifications have the Church-Rosser property and that of finite termination [Huet, Oppen'80], these properties being easily checked for S' and with little work for S.

---

* C.R.I.N.; C.O. 140; 54037 Nancy Cedex; France
** PUC-RJ/Informática; R.M.S. Vicente, 225;
22453 - Rio de Janeiro - RJ; Brazil

```
Type  List(Atom)
  Sorts : L,D {L for lists, D for atoms }
  Operations: with  d:D; x,y:L
      {constructors }
           nil  : L
           make(d):L
           append(x,y):L
      {internal }
           cons(d,x):L
  Rules : for each  d:D; x,y:L
      {between constructors }
  (C1)       append(nil,y) → y
  (C2)       append(make(d),nil) → make(d)
  (C3)       append(append(make(d),x),y) →
                   → append(make(d),append(x,y))
      {defining internal operation }
  (I)        cons(d,y) → append(make(d),y)
endoftype
```

Fig.1: Specification S

```
Type  List(Atom)
  Sorts: L,D {L for lists, D for atoms }
  Operations: with d:D; x,y:L
      {constructors }
           nil  : L
           cons(d,x):L
      {internal }
           make(d):L
           append(x,y):L
  Rules: for each  d:D; x,y:L
      {defining internal operations }
  (I1)      make(d) → cons(d,nil)
  (I2)      append(nil,y) → y
  (I3)      append(cons(d,x),y) → cons(d,append(x,y))
endoftype
```

Fig.2: Specification S'

Method

The natural question now is : "Are S and S' equivalent?" For instance, if one is known to be correct, their equivalence will yield the correctness of the other.

The straightforward method for checking equivalence consists of verifying that each rule of S is a theorem of S', and vice-versa. We propose replacing one of these by verifying a correspondence between normal forms.

1. We first check that for each rule $u \to v$ of S' we can derive $u \equiv v$ in S.

(I1) In S we have
$\underline{\text{cons}}(d,\underline{\text{nil}}) \xrightarrow{(I)} \underline{\text{append}}(\underline{\text{make}}(d),\underline{\text{nil}}) \xrightarrow{(C2)} \underline{\text{make}}\ (d)$.

(I2) is rule (C1) of S .

(I3) In S we obtain

$$\underline{\text{append}}(\underline{\text{cons}}(d,x),y) \xrightarrow{(I)} \underline{\text{append}}(\underline{\text{append}}(\underline{\text{make}}(d),x),y) \xrightarrow{(C3)} \underline{\text{append}}(\underline{\text{make}}(d),\underline{\text{append}}(x,y))$$

and $\underline{\text{cons}}(d,\underline{\text{append}}(x,y)) \xrightarrow{(I)} \underline{\text{append}}(\underline{\text{make}}(d),\underline{\text{append}}(x,y))$

(Recall that $u \equiv v$ is a theorem in a rewriting system iff for some w $u \overset{*}{\to} w$ and $v \overset{*}{\to} w$)

2. Now, we check that for each $g \in F'$ there exists $f \in F$ such that, in S' we can derive $g \equiv f$.

   Here we proceed as follows.

(a) A $g \in F'$ is either $\underline{\text{nil}}$ or $\underline{\text{cons}}(d,g')$ with $g' \in F'$

(b) The following derived rules of S

(D1) $\underline{\text{cons}}(d,\underline{\text{nil}}) \xrightarrow{(I)} \underline{\text{append}}(\underline{\text{make}}(d), \underline{\text{nil}}) \xrightarrow{(C2)} \underline{\text{make}}(d)$

(D2) $\underline{\text{cons}}(d,y) \xrightarrow{(I)} \underline{\text{append}}(\underline{\text{make}}(d), y)$

describe how $\underline{\text{cons}}$ transforms $\underline{\text{nil}}$ and $y \neq \underline{\text{nil}}$ into F.

(c) Now we check (D1) and (D2) to be theorems of S'

For (D1), $\underline{\text{make}}(d) \to \underline{\text{cons}}(d,\underline{\text{nil}})$ is rule (I1) of S'
For (D2), we have in S':

$$\underline{\text{append}}(\underline{\text{make}}(d) , y) \xrightarrow{(I)} \underline{\text{append}}(\text{cons}(d,\underline{\text{nil}}),y) \xrightarrow{(I3)} \underline{\text{cons}}(d, \underline{\text{append}}(\underline{\text{nil}},y)) \xrightarrow{(I2)} \underline{\text{cons}}(d,y)$$

Thus, we have checked

- each one of the 3 rules of S' is a theorem of S
- each one of the 2 derived rules of S is a theorem of S'.

Having checked these 5 theorems we can conclude the equivalence of S and S'. (Notice that the straightforward method would involve checking 3+4 theorems).

Conclusion

Imagine that our data type List(Atom) were enriched with a Boolean sort (with the usual operations) and with operations to test equality of atoms and of lists. In order to define the external operation equality of lists, we would have to add some new rules: 11 rules to S and 4 rules to S' [Remy, Veloso '80]. Now the straightforward method for verifying equivalence would have to check 22 theorems, whereas the method proposed here would need only 9.

This approach also seems to be useful for dealing with related problems, such as specification improvement.

References

C. Choppy, P. Lescanne, J.L. Remy. Improving abstract data type specifications by appropriate choice of constructors Proc. Intern. Workshop on Program Construction; Bonas, France, 1980.

J.A. Goguen, J.W. Thatcher, E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. R.T. Yeh (ed.) Current trends in programming methodology IV, Prentice-Hall, 1978.

J.V. Guttag, E. Horowitz, D. Musser. The design of data type specifications. R.T. Yeh(ed.) Current trends in programming methodology IV, Prentice-Hall, 1980.

G. Huet, D. Oppen - Equations and rewrite rules: a survey. SRI International Rept. CSL-111, 1980.

J.L. Remy, P.A.S. Veloso. Comparing data type specifications via their normal forms.Forthcoming, 1980.