TRAVERSABLE STACK WITH FEWER ERRORS

Paulo A. S. Veloso

Department of Informatics
Catholic University
22453 Rio de Janeiro, RJ; Brazil

Abstract

A complete finite "algebraic" specification for a version of traversable stack with fewer error conditions is presented. This is used to illustrate some issues concerning formal specification of data types, namely naturalness and clarity. Uncontrolled errors are considered harmful and canonical-term specifications are considered helpful.

Introduction

The traversable stack has been posed by Mila E. Majster as an abstract data type that cannot be specified by the algebraic method [Goguen et al. 76, Guttag 77, Zilles 75]. [Majster 77] contends that there is no finite presentation for this data type. This seems to be a controversial issue, for since then several people [Martin 77, Hilfinger 78] have unsuccessfully tried to provide the required specification (cf. [Majster 78]).

Here it is shown that a natural extension of Majster's data type can be given a finite axiomatization. The points stressed are: (1) too many errors can be harmful (why demand an error if there is a "natural" value to be assigned?); (2) a shorthand convention for error propagation; (3) the usefulness of canonical terms as a precise yet fairly intuitive specification to aid an "operational" understanding of the data type (cf. [Levy 77]), and also to systematize the search for an axiomatic specification and its validation.

Informal description

A traversable stack of D (where D is some given data type, say, integers) is just like an ordinary pushdown stack with the extra feature that readout is not restricted to the top. An inner position pointer i indicates a position from which information can be read.

The operations createT, downT, returnT and readT have the same informal explanation as in [Majster 77, p.39]. The difference in this version resides in considering pushT and popT defined whenever possible, as follows.

pushT inserts an element d of D on top of t, incrementing the value of i by one;

popT removes the top element of t, decreasing the value of i by one, if possible; otherwise it gives errorT.

Notice that this is fairly natural. In Majster's explanation this incrementing/decrementing of i occurs only at the top; now it is allowed to happen anywhere it makes sense.

Operations and error propagation

One generally assumes that errors propagate without bothering to say it explicity. The following usual notation is employed.

createT : $\rightarrow$ T

pushT : T×D $\rightarrow$ T $\cup$ {errorT}

popT : T $\rightarrow$ T $\cup$ {errorT}

downT : T $\rightarrow$ T $\cup$ {errorT}

returnT : T $\rightarrow$ T

readT : T $\rightarrow$ D $\cup$ {errorD}

Here it should be understood that the domains of these operations include also errorT and errorD, as the case may be. But it would be foolish to apply them to these error-elements, for the result is again an error.

This notation is suggested as a shorthand to avoid having to write down error-propagation equations, such as pushT(errorT, d) = errorT, pushT(t, errorD) = errorT, and so forth, which would needlessly clutter the specification.

Canonical-term specification

An arbitrary object of the data type T can be represented as errorT, createT or as $\text{downT}^n\ \text{pushT}^m\ (d_1,\ldots,d_m)$ for some $0 \leq n < m$ with all the d_i's distinct and different from errorD, in a unique way. (Here the last expression abbreviates downT(...downT(pushT(...pushT(createT, d_1),...,d_m))...), cf. [Majster 77, p.40]).

If one specifies the effect of the operations on these canonical representatives one obtains a convenient and precise specification of the data type, a canonical term algebra ([Thatcher et al. 76, Goguen et al. 76]).

The effect on errorT is already covered by the above convention. For createT, the informal description suggests

C0 : $\underline{pushT}(\underline{createT}, d) = \underline{downT}^0\ \underline{pushT}^1(d)$

C1 : $\underline{popT}(\underline{createT}) = \underline{errorT}$

C2 : $\underline{downT}(\underline{createT}) = \underline{errorT}$

C3 : $\underline{returnT}(\underline{createT}) = \underline{createT}$

C4 : $\underline{readT}(\underline{createT}) = \underline{errorD}$

More interesting is the effect on a nontrivial canonical term $\underline{downT}^n\ \underline{pushT}^m(d_1,\ldots, d_m)$ with $0 \leq n < m$ (the third case above), given by

(ct1) $\underline{pushT}[\underline{downT}^n\ \underline{pushT}^m(d_1,\ldots, d_m), d] =$
$= \underline{downT}^n\ \underline{pushT}^{(m+1)}(d_1,\ldots, d_m, d)$

(ct2) $\underline{popT}[\underline{downT}^n\ \underline{pushT}^m(d_1,\ldots, d_m)] = \underline{if}\ n = m-1\ \underline{then}\ \underline{errorT}$
$\underline{else}\ \underline{downT}^n\ \underline{pushT}^{(m-1)}\ (d_1,\ldots, d_{m-1})$

(ct3) $\underline{downT}[\underline{downT}^n\ \underline{pushT}^m(d_1,\ldots, d_m)] = \underline{if}\ n+1 = m\ \underline{then}\ \underline{errorT}$
$\underline{else}\ \underline{downT}^{(n+1)}\ \underline{pushT}^m(d_1,\ldots, d_m)$

(ct4) $\underline{returnT}[\underline{downT}^n\ \underline{pushT}^m(d_1,\ldots,d_m)] = \underline{downT}^0\ \underline{pushT}^m(d_1,\ldots,d_m)$

(ct5) $\underline{readT}[\underline{downT}^n\ \underline{pushT}^m(d_1,\ldots,d_m)] = d_{(m-n)}$

Axioms

The specifications (ct1 - ct5) require the lefthand sides to be transformed into the canonical terms given in the righthand sides. This can be accomplished by means of the following axioms.

T1: $\underline{pushT}[\underline{downT}(t),d] = \underline{if}\ \underline{downT}(t) = \underline{errorT}\ \underline{then}\ \underline{errorT}$
$\underline{else}\ \underline{downT}[\underline{pushT}(t,d)]$

T2: $\underline{popT}[\underline{downT}(t)] = \underline{downT}[\underline{popT}(t)]$

T3: $\underline{popT}[\underline{pushT}(t,d)] = \underline{if}\ d = \underline{errorD}\ \underline{then}\ \underline{errorT}$
$\underline{else}\ t$

T4: $\underline{downT}[\underline{pushT}(\underline{createT}, d)] = \underline{errorT}$

T5: $\underline{returnT}[\underline{downT}(t)] = \underline{if}\ \underline{downT}(t) = \underline{errorT}\ \underline{then}\ \underline{errorT}$
$\underline{else}\ \underline{returnT}(t)$

T6: $\underline{returnT}[\underline{pushT}(t,d)] = \underline{pushT}(t,d)$

T7: $\underline{readT}[\underline{downT}(t)] = \underline{readT}[\underline{popT}(t)]$

T8: readT (returnT[pushT(t,d)]) = if t = errorT then errorD
else d

T1 through T8 together with C1 through C4 give a complete specification for the data type traversable stack of D, the error-propagation equations (five of them, as returnT(errorT) = errorT can be obtained from the other axioms) being implicit in the notational convention.

Notice that axioms T3, T5, T6 and T8 are similar to axioms L6, L13, L12 and L7 of [Majster 77, 78 and Hilfinger 78] (T6 is axiom 8 of [Martin 77]). Axiom T4 is pretty reasonable (cf. [Martin 78, p.10]). Axiom T7 says that readT does not distinguish between popT and downT. This points to the key axioms T1 and T2, stating that downT commutes with popT and conditionally with pushT. They are correct because of the wider domains of popT and pushT; for Majster's original specification they would be incorrect (cf. L5 and L4 of [Majster 77, p.39]).

It should be pointed out that these axioms were fairly systematically obtained from the canonical-term specification by means of a apparently widely applicable methodology ([Pequeno-Veloso 78]), which was found useful in specifying the following data types: queues, strings, binary trees, finite sets of naturals, SNOBOL patterns, among others.

Conclusion

The traversable stack of [Majster 77] was used as an example to illustrate the following points.

(a) Errors are a serious business (cf. [Goguen et al. 76]); the abundance of errors in Majster's version is here considered harmful, a natural version with fewer errors having a finite presentation.

(b) The error-propagation part of the specification is suggested to be separated from the nontrivial part in order to enhance clarity.

(c) A canonical-term specification can be a very helpful complement to a specification (formal or informal) [Levy 77, Hoare-Lauer 74], provided that a good canonical form can be found.

(d) A good canonical-term specification is also a guide in the selection and validation of the axioms for a formal specification [Pequeno-Veloso 78]

Acknowledgements

Many enlightening discussions with Tarcisio H. C. Pequeno on the subject of abstract data types are gratefully acknowledged.

References

J. A. Goguen, J. W. Thatcher and E. G. Wagner - An initial algebra approach to the specification, correctness and implementation of abstract data types; IBM Res. Rept. RC 6487, Yorktown Heights, NY, Oct. 1976.

J. V. Guttag - Abstract data types and the development of data structures; CACM 20(6), June 1977, pp. 396-404.

P. N. Hilfinger - Letter to the editor; SIGPLAN Notices 13(1), Jan. 1978, pp. 11-12.

C. A. R. Hoare and P. E. Lauer - Consistent and complementary formal theories of the semantics of programming languages; Acta Informatica 3(2), 1974, pp. 135-153.

M. R. Levy - Some remarks on abstract data types; SIGPLAN Notices 12(7), July 1977, pp. 8-10.

M. E. Majster - Limits of the "algebraic specification of abstract data types; SIGPLAN Notices 12(10), Oct. 1977, pp. 37-42.

M. E. Majster - Letter to the editor; SIGPLAN Notices 13(1), Jan. 1978, pp. 8-10.

J. J. Martin - Critique of Mila E. Majster's paper "Limits of the 'algebraic' specification of abstract data types"; SIGPLAN Notices 12(12); Dec. 1977, pp. 28-29.

T. H. C. Pequeno and P. A. S. Veloso - Do not write more axioms than you have to: a methodology for the specification of abstract data types; forthcoming, 1978.

J. W. Thatcher, E. G. Wagner and J. B. Wright - Specification of abstract data types using conditional axioms (Extended abstract); IBM Res. Rept. RC 6214, Yorktown Heights, NY, Sept. 1976.

S. N. Zilles - Algebraic specification of data types; in Computation Structures Group Memo 119, MIT, Mar. 1975, pp. 1-12.