An ASM Macro Language for Sets
TIK-Report Nr. 34, January 1998

Philipp W. Kutter
Federal Institute of Technology (ETH)
CH–8092 Zürich (Switzerland)
kutter@tik.ee.ethz.ch

Abstract

In the paper I introduce a macro language which allows to use in Gurevich's Abstract State Machines (ASMs) directly the set notation. I define families of sets, a language of *set terms* (union, intersection, instances of families, Cartesian products), their semantics if they appear in transition rules (extension of family-instances, vary over set terms, assignments of set terms to family-instances), and their semantics in boolean terms like set-inclusion and element-of relation. The semantics is given in terms of ASM-rules.

The idea of this macro language is to allow to manipulate sets directly without changing the semantics of ASMs [Gur95]. An integration of sets in the semantics of ASMs has been formalized in [BGS97]. The presented macros have shown to be very useful in the specification of SQL [DiF97].

In ASMs the state is an algebra which has one carrier set, the so called *SuperUniverse*. Subsets of the super universe, so called *universes*, are represented by their characteristic function, i.e. a function from the SuperUniverse to {true, false}.

$$SomeSet : SuperUniverse \rightarrow Bool$$

ASM do not allow the usual set operations on universes, and they do not allow to treat n-ary functions as families of universes. The reason is that both cases can be easily simulated using the existing constructs. In the following we introduce a macro language which allows to use directly the set notation.

We are especially interested in *families of sets*. A family of sets is a function

$$U_0 : U_1 \times \ldots \times U_n \times SuperUniverse \rightarrow Bool$$

where the universes $U_1 \ldots U_n$ are universes for the n indices of the family. Thus if we fix the first n arguments with terms $t_1, \ldots, t_n$, the function could be used like a universe. We introduce thus the notation

$$U_0 : U_1 \times \ldots \times U_n \rightarrow Set$$

and allow to use a term $U_0(t_1, \ldots, t_n)$ wherever a universe can be used. Going a step further, we define a language of *set terms*. A set term is constructed over the following four productions:

- $U_0(t_1, \ldots, t_n)$
- $\overline{S}$
- $S \cup S'$
- $S \cap S'$

where U_0 is a family of universes, $t_1, \ldots, t_n$ are terms fixing the indices of the family, and S, S' are set terms.

The following table gives on the left side the constructs of the set macro language and on the right side the ASM definitions which replace them. The right side may contain constructs of the set macro languages which have to be replaced as well. The replace process is guaranteed to terminate, since the right-hand-side contains always shorter constructs than the left-hand-side. Let S, S' be arbitrary set-terms, $t_0, \ldots, t_n$ ASM-terms, $RULE$ an ASM-transition rule (possibly containing set terms) and U_0 the name of a family of sets:

include t_0 **in** $U_0(t_1, \ldots, t_n)$
exclude t_0 **from** $U_0(t_1, \ldots, t_n)$
extend $U_0(t_1, \ldots, t_n)$ **with** o
  $RULE$
endextend
	$U_0(t_1, \ldots, t_n, t_0) := true$
	$U_0(t_1, \ldots, t_n, t_0) := false$
	import o
	  $U_0(t_1, \ldots, t_n, o) := true$
	  $RULE$
	endextend

$t_0 \in U_0(t_1, \ldots, t_n)$ $t_0 \in \overline{S}$ $t_0 \in S \cup S'$ $t_0 \in S \cap S'$	$U_0(t_1, \ldots, t_n, t_0)$ **not** $t_0 \in S$ $t_0 \in S$ **or** $t_0 \in S'$ $t_0 \in S$ **and** $t_0 \in S'$

vary o **over** S
  $RULE$
endvary
	vary o **over** $SuperUniverse$
	  satisfying $o \in S$
	  $RULE$
	endvary

$U_0(t_1, \ldots, t_n) := S$	**vary** o **over** $SuperUniverse$
	  if $o \in S$ **then**
	  include t_0 **in** $U_0(t_1, \ldots, t_n)$
	  else
	  exclude t_0 **in** $U_0(t_1, \ldots, t_n)$
	  endif
	endvary

$S \subseteq S'$	**for all** $s : SuperUniverse$ **holds** $x \in S$ **implies** $x \in S'$
$S \subset S'$	**exists** $s : SuperUniverse$ **such that** $x \in S'$ **and not** $x \in S$ **and** $S \subseteq S'$

The definitions for $=$ and $\neq$ are straightforward.

As next step we would like to use Cartesian products.The elements of Cartesian products are tuples of elements. We define two families of static functions, corresponding to the usual product and projections.

The n-ary tuple function is

$$(\_, \_, \ldots, \_) : S_1 \times S_2 \times \ldots \times S_n \rightarrow \mathit{Tuple(n)}$$

where Tuple(n) is a parameterized universe as introduced above. The projections are written as

$$\pi\_(\_) : \mathit{Nat} \times \mathit{Tuple(n)} \rightarrow S_n$$

Remark: these functions must be defined for the elements of the reserve as well. Another possibility is to define them for each newly created element on the fly.

We extend the productions of the set-terms with productions for binary and n-ary Cartesian products:

- $S \times S'$
- $\bigtimes_{i=t_0}^{t_1} S_i$

where S, S', are set-terms, t_0, t_1 are integer terms with $t_0 <= t_1$, and S_i is a set term whose parameter may depend from i. In the following table we give the ASM definitions for the new set-term constructors:

vary o **over** $S \times S'$ $RULE$ **endvary**	**vary** new_1 **over** S **vary** new_2 **over** S' let $o = (new_1, new_2)$ $RULE$ **endvary** **endvary**
vary o **over** $\bigtimes_{i=t_0}^{t_1} S_i$ $RULE$ **endvary**	**vary** o **over** $\mathit{Tuple}(t_1 - t_0)$ **satisfying** **for all** $i \in \{t_0, \ldots, t_1\}$ **holds** $\pi_i(o) \in S_i$ $RULE$ **endvary**
...	...

where new_1 andnew_2 are names which are used now-where else in the algebra. We used as well the notation

$$\{t_0, \ldots, t_1\}$$

which denotes the set of all integers from t_0 to t_1. An open question is how to calculate the number of elements in a set term.

Further Work The macro-language can as well be used for a refined version of type annotation, as they are traditionally used in ASMs. Instead of typing functions with universes, the new set terms (union, intersection, family instances e.t.c) can be used. Adding variables to such typing-rules allows to express dependent types, e.t.c.

A formal definition of the macro-language using Montages [AKP97] , and a by Gem-Mex [Anl97] generated stand-alone-translator (lex/yacc, c-code) from

$$\text{(ASM-rules with set-macros)} \rightarrow \text{(pure ASM-rules)}$$

will follow. The Montages of the macro-language formalize as well the static semantics and typing rules, and the generated translator will check these conditions.

In a comparison of ASM with Z, the proposed macro language could be used to show that the convenient set operations of Z can be added to ASMs as syntactic sugar.

References

[AKP97] M. Anlauff, P. Kutter, and A. Pierantonio. Formal aspects of and development environments for montages. In *ASF+SDF'97*, Workshops in Computing. Springer, Berling, 1997.

[Anl97] M. Anlauff. GemMex-Homepage, 1997. http://www.first.gmd.de/~ma/gemmex/.

[BGS97] A. Blass, Y. Gurevich, and S. Shelah. Choiceless polynomial time. Technical Report CSE-TR-338-97, University of Michigan, EECS Department, 1997.

[DiF97] B. DiFranco. Specification of ISO 9075 SQL using Montages. Master's thesis, Universitá di L'Aquila, 1997. (in italian).

[Gur95] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*. Oxford University Press, 1995.