Appendix 3

User documentation

In this appendix we give the user documentation for `am`, the simulator for the abstract definitive machine.

NAME

am - Abstract Machine

SYNOPSIS

am [-n] [-a] [-i*number*] [-s]

DESCRIPTION

am is an simulator for definitive programs. The program is read from the standard input. Output from the simulation caused by procedural actions is sent to the standard output. The simulator has two modes: command and execution. In command mode explicit definitions of variables and instantiations of entities are supplied. In execution mode the definitions and dynamic actions which are executed come from the action store A. In command mode the various stores can be examined:

`l en`	list the entities in the program store
`l ds`	list the contents of the definition store
`l as`	list the contents of the action store

`am` uses four control variables to affect the output produced during simulation. These are:

`nflag`	if true then print notifiable errors
`aflag`	if true then print avoidance errors
`silent`	if false then print information during simulation
`iterations`	the number of execution cycles which are to be performed in each simulation.

The first three variables are boolean, and `iterations` is an integer variable. If `silent` is false then a status message is printed when simulation starts or continues after stopping, and a "#" symbol is printed at the end of each execution cycle. A prompt is also printed when in command mode if `silent` is false - the prompt is of the form "*name*>", where *name* is the name with which `am` was invoked. The status of the control variables can be examined using the control mode command "`status`", which prints the current status of the control variables. If `iterations` has the value zero then the simulation will continue for as long as possible, i.e. it will not stop after a prespecified number of execution cycles. Their default values are:

```
nflag = true
aflag = true
silent = false
iterations = 0
```

The value of these control variables can be changed by using command line options when invoking `am` or by command mode commands. These are:

`set nflag =` *value*
: set the value of `nflag` to be *value*, where *value* is either `true` or `false`.

`set aflag =` *value*
: set the value of `aflag` to be *value*, where *value* is either `true` or `false`.

`set silent =` *value*
: set the value of `silent` to be *value*, where *value* is either `true` or `false`.

`set iterations =` *number*
: set the value of `iterations` to be the integer *number*, *number* ≥ 0

In command mode the run set can be loaded and examined:

`load runset`	load the contents of the run set
`l runset`	list the contents of the run set

Entities can be instantiated:

`name (` *parameter_list* `)`

where `name` is the name of the entity as stored in the program store P, and `parameter_list` is a comma-separated list of parameters, where each parameter is either a number or an arithmetic expression involving only numbers: no variables are allowed in the parameter list of a command mode instantiation. Two entities with the same name can be stored in P if they have different length parameter lists. The appropriate entity to instantiate is selected by matching both the name and the parameter list length before instantiation.

The currently instantiated enties can be listed:

`l in`	list the currently instantiated entities

The current definition and value of a variable can be requested:

`?(`*`variable`*`)`	print the definition and value of *`variable`*

where *`variable`* is a possibly parameterised variable which is in D. A variable can be redefined:

```
define variable = definition ;
```

The simulation can be started by the command "`start`". Once it has stopped, it can be continued by the command "`cont`". Both these commands change the simulator into execution mode, in which it remains until the simulation stops. The simulator is exited by the command mode instruction "`quit`".

The last action that can be performed in command mode is the entering of an entity description for storage in P. An entity description consists of a header and a body. A header is of the form

```
entity cheque (_bank,_number)
```

All parameters are prefixed with an underscore. The formal parameter list contains the parameters which can be used in the body of the entity description. A body is of the form:

```
{
definition
     signed[_bank,_number] = false,
     dated[_bank,_number] = false
action
     has_signed
print ("Cheque number ",_number," has been signed")
          -> signed[_bank,_number] = true ; valid = true,
     has_dated
print ("Cheque number ",_number," has been dated")
          -> dated[_bank,_number] = true
}
```

The definition list contains the variables which are to be added to the definition store D when the entity is instantiated. Initialisation of variables in the definition list is optional. If a variable is not parameterised then no square brackets are used, e.g. `valid`. The type of variables can be either integer or boolean, and is inferred from the values with which they are defined. If the variable is defined as true or false, then it is of type boolean. If it is defined by some integer expression then it is of type integer.

The action list contains the actions which are to be added to the action store A when the entity is instantiated. Each action is a guard, an optional procedural action, and an optional command list. The guard is an expression which can be evaluated. A guard is defined to be an *`expression`*, where *`expression`* is defined to be one of

```
(expression)
rand(n)                          /* random number between 1 and n */
if expression then expression else expression
variable
parameter
expression < expression
expression <= expression
expression == expression
expression > expression
expression >= expression
expression != expression     /* or <> */
```

```
expression && expression      /* or and or AND */
expression || expression      /* or or or OR */
expression + expression
expression - expression
expression * expression
expression / expression       /* 10 / 3 is 3 */
- expression                    /* unary minus */
true
false
number
```

where *variable* is an optionally parameterised variable, *parameter* is a parameter appearing in the formal parameter list in the entity header, and *number* is an integer. In am false is represented by zero and true is represented by any nonzero value. An integer variable can therefore be used as a boolean variable in a guard, and will evaluate to false if it has the value zero and true otherwise.

The optional procedural action has the form

```
print (print_list)
```

where each element of *print_list* is either a string, a variable, or a parameter. The elements of the *print_list* are separated by commas. A string is a portion of text delimited by double quotes, e.g. "The time is now: ".

The optional command list contains a list of commands, where each command is one of a redefinition, an entity deletion, an entity instantiation, and a stop command. A redefinition has the form

```
variable = expression1
```

where *variable* is a possibly parameterised variable, and *expression1* is the same as *expression* with two additions: a variable can be explicitly undefined using the symbol "@", e.g. "top[_stack] = @", and the evaluation operator can be used, e.g. "end_time = |time|+20".

An entity instantiation must supply the name and actual parameter list for the entity being instantiated, e.g. "car(type,year,0)". The parameter list can consist of constants, parameters, or variables which will be evaluated in the state in which the command is selected. An entity deletion must supply the name and actual parameter list for the entity being deleted and is indicated by the use of the keyword delete, e.g. "delete car(type,year,0)".

The "stop" command is used to halt simulation. All the other commands on the run set are executed and then execution ceases.

Any occurrence of "//" in an entity description is the start of a comment, which is terminated by a newline. Comments are ignored. If a mistake is made in typing in the command mode then typing "}<*return*>" will return to the command mode awaiting input.

Simulation terminates for one of four reasons: there are no true guards, there have been iterations number of execution cycles, a "stop" command is executed, or a fatal error has occurred. The reason for stopping will be given on termination.

OPTIONS

am can be invoked with four options:

-n	set the control variable nflag to false
-a	set the control variable aflag to false
-s	set the control variable silent to true
-i*n*	set the control variable iterations to the integer *n*.

DIAGNOSTICS

The errors which can be generated by am fall into five categories: unrecoverable, static, notifiable, avoidance and fatal. All error messages should be self-descriptive, and include appropriate details of the variable or entity which caused the error. We here just describe the different classes of errors which can occur.

Unrecoverable errors occur when some error has been detected in the internal data representation. The only way I know of to cause this error is outlined in the **BUGS** description.

A static error occurs during the execution of a command mode instruction. When entering an entity description a static error is caused if duplicate parameters appear in the formal parameter list or if a parameter is used in the body which is not in the formal parameter list. When instantiating an entity in command mode a static error is caused if an instance already exists with the same parameters, the entity is not in P, the entity is in P but with a different number of parameters, or a duplicate variable is created in D by the instantiation. When defining a variable in command mode a static error is caused if the variable being defined is not in D.

Notifiable errors occur during simulation when a variable is used but not in D, defined in a circular manner, not defined , or defined in terms of undefined variables. Dynamic actions in A which act on entities which either do not exist in P or only exist with a different number of parameters also cause notifiable errors.

Avoidance errors occur during simulation when guards are evaluated, because a variable in a guard is either undefined or not in the definition store D.

A fatal error occurs during simulation when the error is too severe to allow simulation to continue. A fatal error is caused when the run set contains a redefinition of a nonexistent variable, a redefinition involving evaluation of a non-evaluable expression, redefinitions of the same variable more than once, two dynamic actions on the same entity, a redefinition and deletion of a variable, an inappropriate dynamic action, instantiation of an entity not in P, an instantiation which leads to duplication of variables in D, or an instantiation with non-evaluable parameters.

BUGS

When an entity is instantiated the associated initialisations should be performed at that time, and use of the evaluation operator should entail evaluation in the state in which the entity is instantiated. This is not implemented, and so an initialisation of the form "`x=|time|`" will result in the definition "`|time|`" being stored in D, instead of the evaluation of the `time` variable in that state. This means that whenever the variable `x` is subsequently evaluated, it will have the current value of `time`, instead of the value when the entity was instantiated. If the evaluation operator is used in a redefinition in a command it is performed correctly. The problem can be partly circumvented by performing the initialisation in the first execution cycle after the entity has been instantiated by using the `init_flag` construct described in §5.4.1.

The rand() function can only use explicit values as its argument, i.e. not a variable.

When several variables are used in an `if...then...else` construct the deletion of the entity owning the variable defined in this way can cause a core dump.

When there are two dynamic actions on the same entity in the run set a fatal error is always generated. There is no testing for whether they are in a valid order in the same command list.