This section describes the principal features of am, the implemented abstract definitive machine. Familiarity with Chapters 2 and 3 of "Definitive Parallel Programming" is assumed.

PROGRAM SPECIFICATION

An adm program is specified by a set of entity descriptions, each of which consists of a header and a body. The entity descriptions describe the sets of variables and actions which can be instantiated.

· entity description

An entity description is of the form

The name of the entity is an alphanumeric identifier, and must start with a letter. The parameter list is a possibly empty list of comma-separated parameter names, where each parameter name starts with an underscore ("_"), e.g. (_bank,_number). A set of entity descriptions gives a program specification. Any occurrence of "//" is the start of a comment, which is terminated by a newline.

· entity body

An entity body is of the form

```
definition variable_list
```

action action list

where $variable_list$ is a list of variables, each of which can be optionally initialised with a definition (e.g. a = b * c * d). The $action_list$ is a comma-separated list of actions.

· action

An action is of the form

guard procedural_action -> command

where a guard is an arithmetic or boolean expression involving constants and other variables. It is

false if it evaluates to false or zero, and true otherwise. A procedural_action is of the form

print (message)

where message is a comma-separated list of quoted strings (e.g. "Number: "), parameters and parametric variables. The value of supplied parameters or variables is printed. A command is a semicolon-separated list of dynamic actions and definitions, or the keyword stop, which halts execution.

· dynamic action

A dynamic action involves the instantiation or deletion of an entity. An entity instantiation is of the form

name (parameter_list)

where name is the name of the entity to be instantiated, and parameter_list is the commaseparated list of parameters. Each parameter can be an arithmetic or boolean expression involving either parameters of the entity or constants, but not variables. An entity deletion is of the same form, but prefixed with the keyword delete. Parameters are used in a call-by-value manner, so a parameter cannot be redefined.

· definition

A definition is of the form

variable = expression

where expression is a boolean or integer arithmetic expression. The boolean operators available are <, <=, ==, >=, >, !=, && and ||, and the arithmetic operators available are the four standard operators (+, -, *, /), unary minus (-), the rand () function where rand (n) returns a value between 1 and n, and the arithmetic if...then...else... construct. Expressions can also contain constants (true, false, @ and integers), the evaluation operator (e.g. |date|), and other variables.

· variable

A variable is an identifier and possibly an associated parameter list. An identifier is an alphanumeric string which starts with a letter. The associated parameter list may be empty, or may consist of a list

of parameters enclosed in square brackets. Parameters are identifiers prefixed by an underscore(_). Examples of variables include valid, book [_number] and cheque [_date,_signed].

PROGRAM EXECUTION TO A TO THE PROGRAM DESIGNATED AND THE PROGRAM EXECUTION TO THE PROGRAM EXECUTION TO THE PROGRAM DESIGNATED AND THE PROGRAM DESIGNATION OF THE PROGRAM DESIGNATION OF

Once the entity descriptions are entered, the desired instantiations are made by commands of the form

name (parameter_list)

More than one instantiation of the same entity can occur, but each must be disambiguated by the use of distinct parameters. The program can be executed by typing start. Execution consists of repeated execution cycles. Each execution cycle involves evaluating all guards, performing the procedural actions associated with true guards and putting the associated command lists into the run set, and then executing the lists of commands on the run set. Evaluation is optimised, so "true | | x" always evaluates to true. Evaluation of guards and printing of messages is performed prior to any redefinitions in an execution cycle.

The program executes until a **stop** command is executed, no guards are true, the specified number of iterations have been performed, or an error occurs. Errors are classified into notifiable, avoidance and fatal, with explanatory diagnostics. A notifiable error occurs when a variable which is referred to is not in the definition store D, or a dynamic action refers to an entity which is not in the program store P. An avoidance error occurs when a guard cannot be evaluated. A fatal error occurs when there is interference between the commands in the run set.

· control variables in am

There are four control variables used by am (default values in brackets):

nflag (true) if true then print notifiable errors

aflag (true) if true then print avoidance errors

silent (false) if false then print information during simulation

iterations (true) the number of execution cycles which are to be performed in each simulation

Their values can be examined by the keyword status, and can be changed either by using the command line flags -n, -a, -s and -inumber respectively, or as a command from within am using the keyword set, e.g. "set iterations = 12" or "set aflag = false".

· commands to am

Generally a program will be executed for a specified number of execution cycles (according to the value of iterations), and then the user will be allowed to intervene. The contents of the various stores can be examined:

- 1 en list the contents of the program store P
- 1 ds list the contents of the definition store D
- 1 as list the contents of the action store A
- 1 in list the currently instantiated entities

The run set can be loaded and examined:

load runset load the run set

1 runset list the run set

Entities can be instantiated, in the same way as the initial instantiations were made. Variables can be redefined directly:

```
define variable = expression ;
```

The definition and value of a variable can be printed:

```
?(variable)
```

Notice that these facilities are intrinsic to the execution of an adm program, and are not merely debugging aids. In particular, it is intended that the user will intervene to effect changes of state by means of redefinitions and instantiations. Such changes will be guided by the current state, which is discerned by examination of the various stores. Execution can then be continued by using the keyword cont.

The implementation described here uses boolean and arithmetic data types and operators. Other underlying algebras could have been used, which would be more suitable for specialised applications. More sophisticated variants of the adm can be simulated by using the print command to generate input to an interpreter for a special purpose definitive notation, such as DoNaLD or SCOUT.

This section describes the principal features of any the implemented abstract definitive machine

Addendum

PROGRAM SPECIFICATION

An adm program is specified by a set of entity descriptions, each of which consists of a header and a body. The entity descriptions describe the sets of variables and actions which can be instantiated.

entity description

An entity description is of the form

entity name (parameter list)

(wood)

The name of the entity is an alphanomene identifier, and must start with a lotter. The parameter list is a possibly empty list of comma-separated parameter names, where each parameter name starts with an underscore ("_"), e.g. (_bank,_number). A set of entity descriptions gives a program specification.

Any occurrence of "//" is the start of a comment, which is tenninated by a newline.

· entity body

An entity body is of the form

definition variable list

action list

where vertable_itst is a list of variables, each of which can be optionally initialised with a definition (e.g. a = b * c * d). The action itst is a comma-separated list of actions.

noites:

Better is the end of a thing than the beginning thereof.

Ecclesiastes vii 8.

Action of Committee Action

Profilet .

read as real municipal

This section describes the principal features of am, the implemented abstract definitive machine. Familiarity with Chapters 2 and 3 of "Definitive Parallel Programming" is assumed.

in the least of the control of the c

oligin krana ninggar kalagar sa krajan di inggaran kranang i Pika ninggaran di Pika naingka kan di Pikangan k

PROGRAM SPECIFICATION TO A STATE OF THE STAT

An adm program is specified by a set of entity descriptions, each of which consists of a header and a body. The entity descriptions describe the sets of variables and actions which can be instantiated.

u de la ciglia despensa e comunidad e um em eficaçõe de la considera e exemplos a comunidades de municipal de

· entity description

An entity description is of the form

ं राज्य तेषका **१९ ४० तम्**कार करणे वर्षा वर्षा करणा करण करणे । यह है । । व्यवस्था करणा वर्षा के सम्बद्ध हुए

The name of the entity is an alphanumeric identifier, and must start with a letter. The parameter list is a possibly empty list of comma-separated parameter names, where each parameter name starts with an underscore ("_"), e.g. (_bank,_number). A set of entity descriptions gives a program specification. Any occurrence of "//" is the start of a comment, which is terminated by a newline.

· entity body

An entity body is of the form

definition variable list and a second of the second of the

lieb ar action action list signification and the second second

where variable_list is a list of variables, each of which can be optionally initialised with a definition (e.g. a = b * c * d). The action_list is a comma-separated list of actions.

tre i je i ko uktibukturo skupilika iš demovratika Poselst mekki kustuo iki i kil

· action

An action is of the form

guard procedural action -> command

where a guard is an arithmetic or boolean expression involving constants and other variables. It is

รายเกรียดรัฐที่ พระบางเดียกกระการการการการครัฐสรรมระบาก ผู้รัฐสมโดยการครัฐ และเรียงนักโรย และเหมียวการ

false if it evaluates to false or zero, and true otherwise. A procedural_action is of the form

print (message)

where message is a comma-separated list of quoted strings (e.g. "Number: "), parameters and parametric variables. The value of supplied parameters or variables is printed. A command is a semicolon-separated list of dynamic actions and definitions, or the keyword stop, which halts execution.

• dynamic action

A dynamic action involves the instantiation or deletion of an entity. An entity instantiation is of the form

ming anisate bite melate of Sham server and period anishments and became a state of a server and a server

name (parameter_list)

where name is the name of the entity to be instantiated, and parameter_list is the commaseparated list of parameters. Each parameter can be an arithmetic or boolean expression involving either parameters of the entity or constants, but not variables. An entity deletion is of the same form, but prefixed with the keyword delete. Parameters are used in a call-by-value manner, so a parameter cannot be redefined.

radinum ballipara del jodij jam semproti se peldelje, odo sukse koje a koje ješije a stelotrog so š

definition

A definition is of the form

margong variable = expression golds oments and control of control

where expression is a boolean or integer arithmetic expression. The boolean operators available are <, <=, ==, >=, >, !=, && and ||, and the arithmetic operators available are the four standard operators (+, -, *, \(\infty \), unary minus (-), the rand () function where rand (n) returns a value between 1 and n, and the arithmetic if, ..then ... else... (construct. Expressions can also contain constants (true, false, @ and integers), the evaluation operator (e.g. |date|), and other variables.

· variable

A variable is an identifier and possibly an associated parameter list. An identifier is an alphanumeric string which starts with a letter. The associated parameter list may be empty, or may consist of a list

if one then print avoidance errors

modsturme the property of the think of the first of the f

of parameters enclosed in square brackets. Parameters are identifiers prefixed by an underscore(_). Examples of variables include valid, book [_number] and cheque[_date,_signed].

commit-sengrated list of quoted surnes (e.g. "Mumber: "), parameters and

PROGRAM EXECUTION

acaro Rejerina de disc

Once the entity descriptions are entered, the desired instantiations are made by commands of the form

name (parameter list)

on-separated list of dynamic actions and definitions, or the

1 12 4

More than one instantiation of the same entity can occur, but each must be disambiguated by the use of distinct parameters. The program can be executed by typing start. Execution consists of repeated execution cycles. Each execution cycle involves evaluating all guards, performing the procedural actions associated with true guards and putting the associated command lists into the run set, and then executing the lists of commands on the run set. Evaluation is optimised, so "true | | x" always evaluates to true. Evaluation of guards and printing of messages is performed prior to any redefinitions in an execution cycle.

The program executes until a stop command is executed, no guards are true, the specified number of iterations have been performed, or an error occurs. Errors are classified into notifiable, avoidance and fatal, with explanatory diagnostics. A notifiable error occurs when a variable which is referred to is not in the definition store D, or a dynamic action refers to an entity which is not in the program store P. An avoidance error occurs when a guard cannot be evaluated. A fatal error occurs when there is interference between the commands in the run set.

· control variables in am

There are four control variables used by am (default values in brackets):

with the first that the first the first of the first state of the first state of the

nflag (true) if true then print notifiable errors

aflag (true) if true then print avoidance errors

silent (false) if false then print information during simulation

iterations (true) the number of execution cycles which are to be performed in each simulation

SOMETHING IN MARKET MARKET

and the second of the second and the

ylan, reduced as the finish of the rest

FOR THE OF BOTH FOR ALL THE TONE OF THE

(State of the state)

Line Communication in soil includes at the

Their values can be examined by the keyword status, and can be changed either by using the command line flags -n, -a, -s and -inumber respectively, or as a command from within am using the keyword set, e.g. "set iterations = 12" or "set aflag = false".

· commands to am

54. 50 H 30513

to authorize the sail

of implication of

Generally a program will be executed for a specified number of execution cycles (according to the value of iterations), and then the user will be allowed to intervene. The contents of the various stores can be examined:

- 1 en list the contents of the program store P
- 1 ds list the contents of the definition store D
- 1 as list the contents of the action store A
- 1 in list the currently instantiated entities

The run set can be loaded and examined:

load runset load the run set burnels of the most season with the season with t

1 runset list the run set

Entities can be instantiated, in the same way as the initial instantiations were made. Variables can be redefined directly:

define variable = expression ;

The definition and value of a variable can be printed:

?(variable)
msm(A) of the control of

3: 557T

Notice that these facilities are intrinsic to the execution of an adm program, and are not merely debugging aids. In particular, it is intended that the user will intervene to effect changes of state by means of redefinitions and instantiations. Such changes will be guided by the current state, which is discerned by examination of the various stores. Execution can then be continued by using the keyword cont.

The implementation described here uses boolean and arithmetic data types and operators. Other underlying algebras could have been used, which would be more suitable for specialised applications.

More sophisticated variants of the adm can be simulated by using the print command to generate input to an interpreter for a special purpose definitive notation, such as DoNaLD or SCOUT.

intermital upon the

Their values can be examined by the keyword status, and can be changed either by using the command line flags $-n_s - a_s - a_s$ and -i number respectively, or as a command from within amusing the keyword set, e.g. "set iterations = 12" or "set aflag = false".

· commands to a m

Generally a program will be executed for a specified number of execution cycles (according to the value of iterations), and then the user will be allowed to inservene. The contents of the various stores can be examined:

list the contents of the program store P	
list the contents of the definition store D	

1 in list the currently instantiated entitles

The run set can be loaded and examined:

load nunset load the nun set

l runset list the run se

Entities can be instantiated, in the same way as the initial instantiations were made. Variables can be redefined directly:

define variable = expression

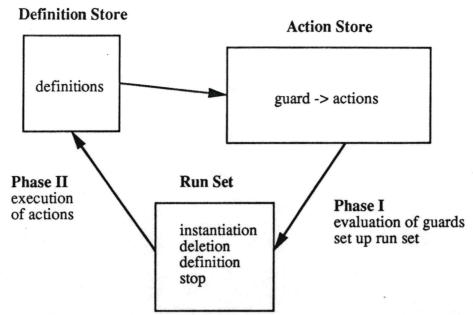
The definition and value of a variable can be printed:

? (variable)

Notice that these facilities are intrinsic to the execution of an adm program, and are not merely debugging aids. In particular, it is intended that the user will intervene to effect changes of state by means of redefinitions and instantiations. Such changes will be guided by the current vare, which is discerned by examination of the various stores. Execution can then be continued by using the keyword cont.

The implementation described here uses boolean and arithmetic data types and operators. Other underlying algebras could have been used, which would be more suitable for specialises applications. More sophisticated variants of the adm can be simulated by using the parameter command to generate input to an interpreter for a special purpose definitive notation, such as DoNaLD or SCOUT.

Execution of ADM



```
begin
                                                                                test for notifiable errors
    while there is another variable to be examined in D
        if the variable is defined in a non-evaluable manner
        then issue a notifiable error
                                                                                       evaluate guards,
    while there is another guard to be examined in A
                                                                            testing for avoidance errors
        begin
            evaluate the guard
            if the guard cannot be evaluated
            then issue an avoidance error
            else
                if the guard is true
                    then
                        begin
                                    (the procedural action involves a non-evaluable variable)
                               or (a non-evaluable variable is evaluated in a command)
                                or (a variable used as a parameter cannot be evaluated)
                            then halt execution due to a fatal error
                            add the associated command list to the run set
        end
                                                                stop execution when no guards are true
    if the run set is empty
        then halt
                                                                                    test whether run set
            (the run set contains a redefinition of a non-existent variable)
                                                                              contains invalid transition
        or (the run set contains an instantiation of an entity not in P)
        or (the run set contains two redefinition of the same variable which are not
                    in the same command list)
        or (the run set contains a redefinition of a variable and a dynamic action on the entity owning
                that variable, other than when the redefinition and dynamic action are in the same
                command list and either the redefinition comes before an entity deletion or after an
                entity instantiation)
        or (the run set contains an invalid dynamic action)
        or (the run set contains two dynamic actions on the same entity, other than when they are in
                the same command list and valid when performed sequentially)
    then halt execution due to a fatal error
    simulate the parallel execution of all the command lists in in the run set
                                                                                     execute commands
end
```