Lecture M3: Introduction to eden programming

M3.1. Background

The eden interpreter due to Y W (Edward) Yung in 1987.

Designed for UNIX/C environment

Cooperates with EX, the interface used to link eden to X-Windows.

The eden interpreter = evaluator for definitive notations

"hybrid" programming tool = definitive + procedural paradigms
essential link to drive UNIX utilities and hardware devices

M3.2. Overview of basic eden characteristics

C-like

- syntactic conventions and data types
- basic programming constructs: the for, while, if and switch

Main types for variables: float, integer, string and list.
Lists can be recursive and need not be homogeneous in type.
Comments are enclosed in /* .... */ parentheses.

Two sorts of variables in eden: formula and value variables.

Formula variables are definitive variables.

Value variables are traditional procedural variables.

The type of an eden variable is determined dynamically:

can be changed by re-assignment or re-definition.

How to program in eden?

Many eden programs can be built from three abstract programming features: definitions, functions and actions:

- definitions

  A formula variable v can be given a definition via

  ```
  v is f(a,b,c);
  ```

  The eden interpreter

  maintains the values of definitive variables automatically, records all the dependency information in a definitive script.

- functions

  The user can define special-purpose functions via

  ```
  func fn       /* function to compute result = fn (p1,p2,...,pn) */
  {
      para p1, p2, ...., pn              /* pars for the function */
      auto result, a1, a2, ...., am          /* local variables */
      <assignments and definitions>
      return result
  }
  ```

  User-defined functions can appear on the RHS of formulae.

Procedural aspect of eden .... complements definitive

- actions

An action is a "triggered procedure", specified via

```
proc pti : t1, t2, ... , tn    /* proc triggered by t1, t2,..., tn */
{
        auto result, a1, a2, ...., am              /* local variables */
        <assignments and definitions>
}
```

An action is a generalised procedure (for a procedure, n=0).

An action is triggered

whenever one of its triggering variables ti is updated

whether or not the value of ti is changed in the update.

M3.3. Illustrative examples of eden use

1. How to use eden to implement a definitive notation (see later)

A definitive script in eden has no direct visualisation as for DoNaLD

To interpret a donald script in eden:

translate donald definitions into eden definitions

associate display actions with variables.

Interpretation of a donald script made easy

as maintenance of definitive variables is done automatically.

Interrogating values and current definitions of variables in eden:

- use the procedure call

  writeln(v)

  to display the current value of an eden variable v
- use the query

  ?v;

  for the defining formulae and dependency status of v.

Values of formula variables in definitive scripts can be undefined.
The symbol @ is used to denote undefined, as in a=@
For triggering, the values of trigger variables must be defined.
Boolean expression v==@ tests for undefinedness of variable v.

2. The timetable.e

3. The tank.e program

M3.4. Using eden

Common mode of eden program development on workstation:

edit a program in one window

execute eden in another

Cut-and-paste from the editor window into the interpreter window.

In development process, useful to be able to undo design actions.

Restore scripts of definitions by restoring the original definitions

Hence comment out old fragments of scripts from the edited file.

In eden

```
include("filename.e");
```

causes the eden file filename.e to be read.

Can be used to restore a definitive script to its original form.

Using eden scout and/or donald translators

Typical command line:

```
cat filename.s - | scout  | donald | eden -n
```

"-" is a cat option: input from standard input after filename.s read

"-n" option for eden suppresses the eden prompt.

M3.5. Miscellaneous additional issues for eden

eden best suited for scripts of fixed length, persistent variables

- best in design: incrementally constructing a large model by refining and extending a set of definitions.
- not so good in concurrent systems modelling: set of observations changes dynamically as transient processes / agents are created and destroyed

Can make scripts dynamic to some extent using eden features:

- execute() cf specification of the integrators in the VCCS
- forget()
- `<string>` (turn string into variable name) cf digital watch

Common points of concern

- action is first defined, is a one-off action call
- can't define components of the list associated with a formula variable lv by defining lv[i]
- order of actions in an eden file can be significant
  [common problem of a rule-based programming paradigm]

Fundamental issues for eden

Experience shows

- eden is an exceptionally powerful programming tool
- adding definitive scripts in a procedural environment simplifies many programming tasks
- eden can be easily abused: programming principles that might underlie eden are hard to identify

Motivates need

- to relate programming in eden to other paradigms
- for a satisfactory abstract account of eden programming
- to identify what distinguishes good and bad eden programs.

Definitions vs Actions

Might suppose that .....

definition "v is f(a,b,c)"

= an assignment "v=f(a,b,c)" that is triggered by a, b and c

This can be true in operational terms in a sequential paradigm

Does this make eden a rule-based paradigm?

NO! major virtue of eden programming

use of definitions not actions disciplines execution of actions.

For instance:

- maintaining definitions takes priority over actions

  => consistent relationships between variables
- definitions cannot be cyclic,

  actions can trigger indefinitely through indirect self-reference.

Good programming philosophy for eden:

aim to express state-transitions in a computation

in terms of redefinition of definitive scripts

not as the cumulative side-effects of loosely synchronised actions.

References

Y W Yung The EDEN Handbook

Forward reference to the ADM

Fix(ed) donald to operate in the donald filename.d - | eden -n mode?