Topical concerns …

Coursework for CS405 module: WEB-EM-5

…/dcs/research/em/teaching/cs405/assignment

Practical basis for working with EDEN has been laid in Lab 2: adequate for coursework … but more advanced aspects to be covered

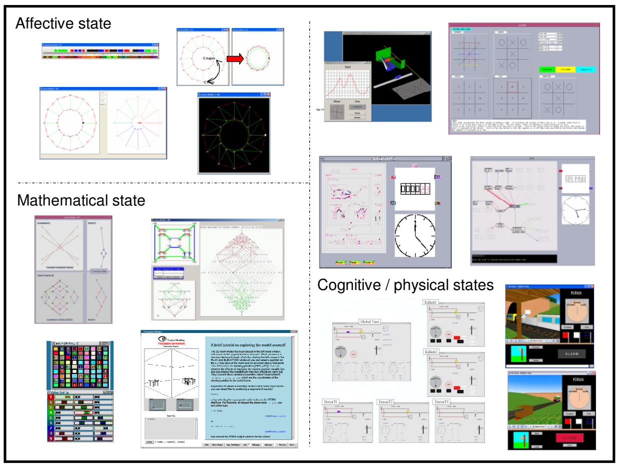
Examination paper to be based on questions posed in conjunction with the labs

Getting from …

modelling with definitive scripts

to

a conceptual framework
for very general computing applications



Single-agent modelling

Archetypal use of MWDS
human-computer-environment interaction

Variables in a definitive script represent
- the values that the ‘user’ can observe
- the parameters that the ‘user’ can manipulate
- the way that these are linked indivisibly in change

a definitive script can imitate physical experiments
[cf. the role of spreadsheets in describing and predicting]

Environment not document

A script = an **environment** rather than a document.

In a document:

- meaning of a symbol has to be represented in a stateless fashion
- the **reader** animates it by studying the contexts in which it occurs

In a definitive script:

can explore the significance of ‘symbols’ via experiment and observation (hence “observables”)

Variable values, observations and state

Definitive variables

- correspond to observables of phenomena external to the computer system
- have an identity and a value that can change according to the circumstances of observation.

The term *state* refers to what we understand by
a set of observations made 'simultaneously'

Interpreting the current state

The current state =
what I deem to be simultaneous observations

The concept of state is
- relative to the observer ("observing agent")
- relative to focus of attention and mode of observation

A definitive script can represent several states at once …

Several states at once …

Who is the agent?

What is the focus of their attention? their intention?

What mode of observation is being adopted?

What metaphors available within the modelling medium?

Different modes of observation …

roomYung1989
… *primitive DIY support for the designer and user*
roomviewerYung1991
… *enhancing the interface for the designer and user*
room3dMacDonald1998
… *introducing physical and mechanical concerns*
graphicspresHarfield2007
… *studying the projection from 3D to 2D*
room3dsasamiCarter1999
… *experimenting with lighting in rendering the room*

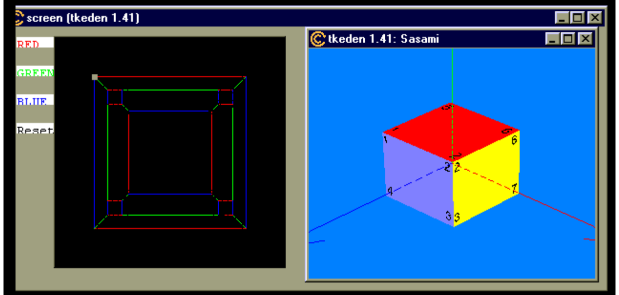
symcubeWong2001

Model illustrates combined use of ARCA and SASAMI

Note the many possible interpretations of nodes in S4:
points in 2-space and 3-space
permutations of {1,2,3,4}
transformations of cube
matrices (see the text output window)

Matrix algebra manipulation implicit in model: re-use

symcubeWong2001



Recall …

Single agent modelling

Archetypal use of MWDS
human-computer-environment interaction

Variables in a definitive script represent
- the values that the ‘user’ can observe ORACLE
- the parameters that the ‘user’ can manipulate HANDLE
- the way that these are linked indivisibly in change DERIVATE

definitive script can model physical experiments
[cf. the role of spreadsheets in describing and predicting]

LSD

“Language for Specification and Description”

Appropriate way to distinguish the perspectives of different agents on a single state …

… classify observables so as to discriminate between ORACLES that can be observed by the agent (if only conditionally and imperfectly), HANDLES that can be changed (perhaps conditionally), and what dependencies pertain in the view of the agent DERIVATES

… possibly also identify routine patterns of response that an agent might make (e.g. how a button works, user guides)

Beyond single agent modelling …

Motivate multi-agent modelling …

- dependency serves to identify state-change for which the observer/modeller can presume responsibility
- in most environments, there are (apparently!) other sources of state-change: attribute to other agents

state-changes can occur concurrently through agent action

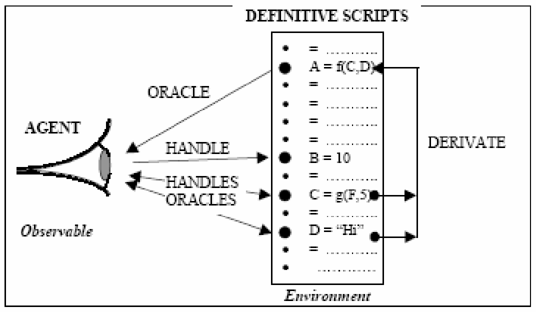


Figure 2-18: Definitive script as observer’s model of state (‘one-agent’ modelling)

Multi-agent Modelling

In EM for multi-agent modelling, model concurrency

- generalise MWDS for the user-computer interface to model the relationship between all interacting agents
- each agent-environment interface is treated as a domain for experiment

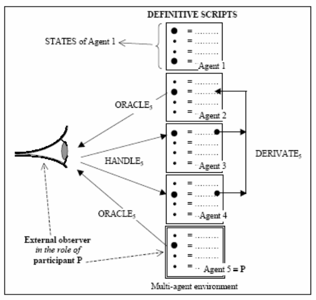


Figure 2-19: Definitive script as observer’s model of state (‘multi-agent’ modelling)

Broad objective in MWDS

Use as a basis for

universal agent-oriented modelling …

capture human-like and machine-like agency
not typically closed-world and system-like

Very general applications of computing can be treated as specialisations e.g. programming as a specialisation of concurrent systems modelling

The experimental paradox

A key element in specialising from multi-agent Empirical Modelling to traditional programming is concerned with a reinterpretation of action associated with experiment:

… an action is at first carried out with uncertain expectations about the outcome …

… the very same action is subsequently carried out with absolute conviction that the outcome will be as expected

Some modelling challenges …

- perception of state in everyday observation
- mode-switches in perceptions of state
  cf. developing functional program / evaluating function
- interpreting the constructs in program design and development with reference to MWDS
- the uses of reference that arise in mathematics:
  exposition of proof that can never be formalised

Programming from an Empirical Modelling perspective

From modelling with definitive scripts to programming …
- how to use EDEN
- comparative studies
  - Logo
  - object-orientation
  - functional programming

How to use procedural EDEN …

Hybrid notation
procedural and definitive

Procedural aspects
- procedures and procedurally expressed functions: use traditional C-like code
- parameters [para] and local variables [auto] in the bodies of procedures and functions

How to use definitive EDEN …

Hybrid notation
  procedural and definitive

Definitive aspects
- observables that refer directly to the external situation - have direct external counterparts
- relationships between them reflect observed dependencies between their counterparts
- cf. global observables (deprecated in programming)

Aspiration for definitive programming

The only state changes that can be observed externally are associated with meaningful interpretable changes of state to external observables associated with the artefact or the situation in which it is being used …

… this was also the original motivation for object-oriented programming (in the days of Simula - 1967)

… hence significance of using `auto` to hide local vars

How to use definitive EDEN …

Definitions of observables express dependencies to create the state of the environment mediating interaction: note that these can relate to the artefact or the context

Functions serve to enrich the range of operators available in the formulae that express dependencies

Actions ("triggered procedures") reflect actions of agents that are automatically invoked in appropriate states

Procedural aspects of JUGS

Parameters and local variables, procedural code

```
func repeatChar {
  auto s, i;      ## local variables
  s = substr("", 1, $2);
  ## note use of $2 - second parameter
  for (i = 1; i <= $2; i++) s[i] = $1;
  return s;
}
```

… using user-defined function to enrich formulae

```
cA is repeatChar('~', widthA*contentA);
```

Definitive aspects of JUGS

Observables and dependencies in the JUGS model

Observables:

```
capB, contentA, width, status, target
Afull, updating, valid1, …
```

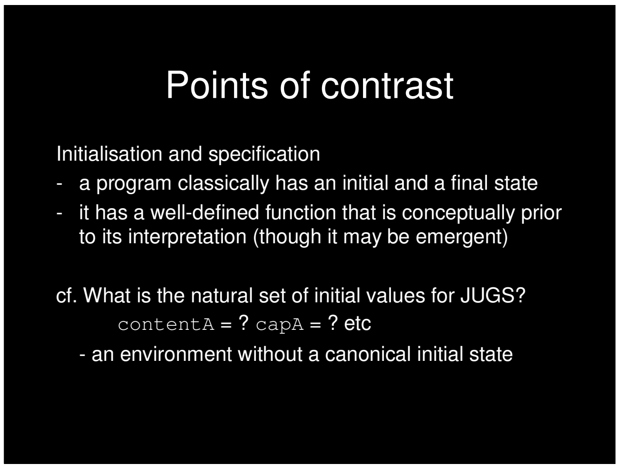
Dependencies

```
valid1 is !Afull;
Afull is capA==contentA;
```

Definitive aspects of JUGS

Agents and actions in the jugsBeynon2008 model:

```
proc fillingB: tick, option {
      if ((option==2) && avail(2)) {
             contentB = contentB + 1;
             jugBfilling = 1;
      }
      else if (jugBfilling==1)
                 jugBfilling = 0;
}
```



Points of contrast

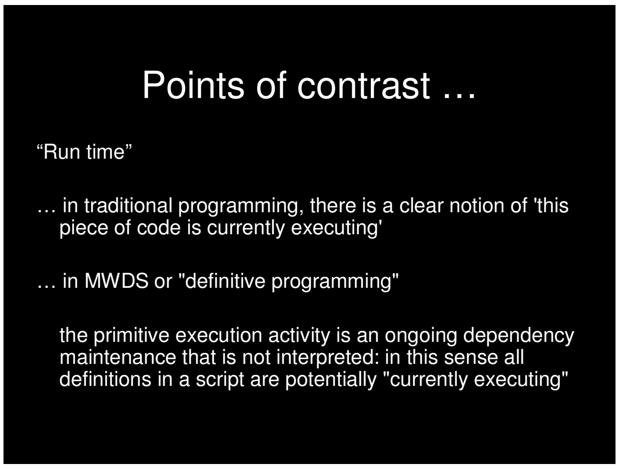
Initialisation and specification

- a program classically has an initial and a final state
- it has a well-defined function that is conceptually prior to its interpretation (though it may be emergent)

cf. What is the natural set of initial values for JUGS?

`contentA` = ? `capA` = ? etc

- an environment without a canonical initial state



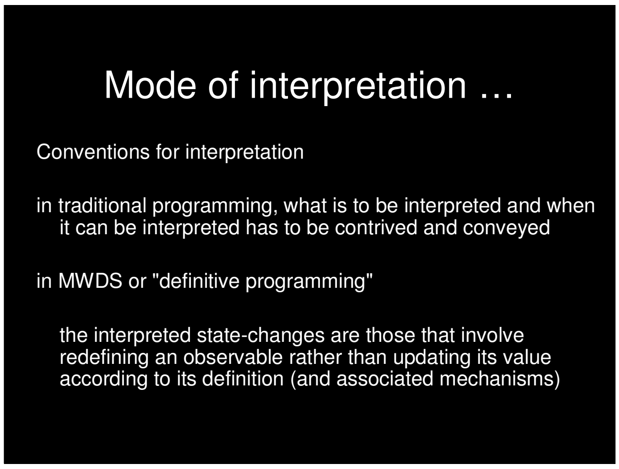
Points of contrast …

"Run time"

… in traditional programming, there is a clear notion of 'this piece of code is currently executing'

… in MWDS or "definitive programming"

the primitive execution activity is an ongoing dependency maintenance that is not interpreted: in this sense all definitions in a script are potentially "currently executing"



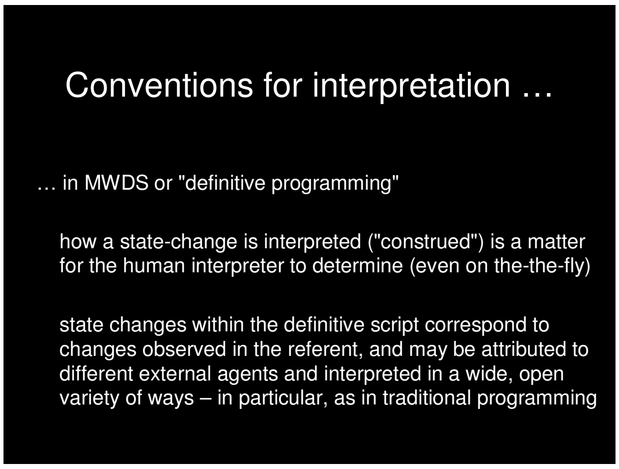
Mode of interpretation …

Conventions for interpretation

in traditional programming, what is to be interpreted and when it can be interpreted has to be contrived and conveyed

in MWDS or "definitive programming"

the interpreted state-changes are those that involve redefining an observable rather than updating its value according to its definition (and associated mechanisms)

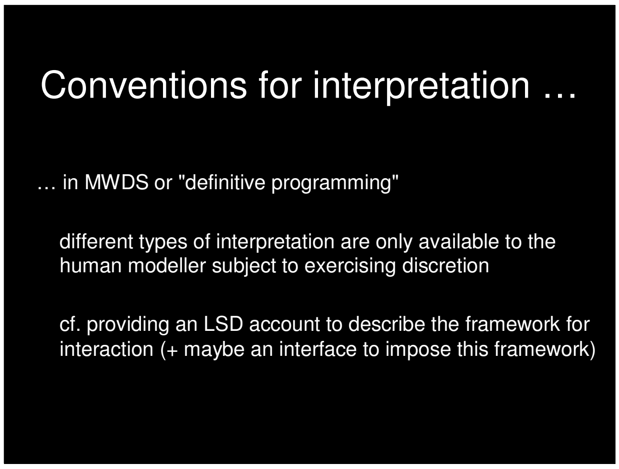


Conventions for interpretation …

… in MWDS or "definitive programming"

how a state-change is interpreted ("construed") is a matter for the human interpreter to determine (even on the-the-fly)

state changes within the definitive script correspond to changes observed in the referent, and may be attributed to different external agents and interpreted in a wide, open variety of ways – in particular, as in traditional programming

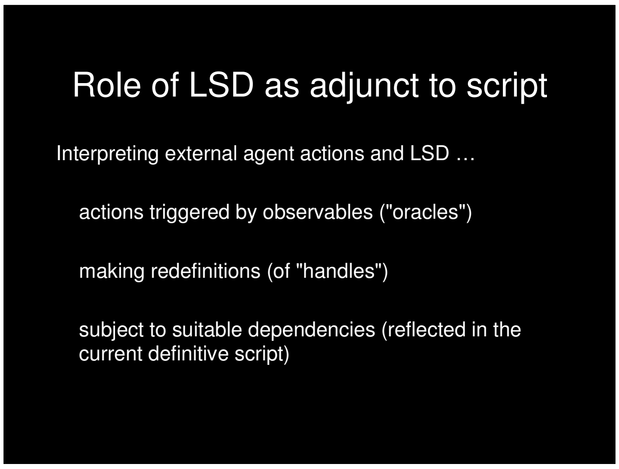


Conventions for interpretation …

… in MWDS or "definitive programming"

different types of interpretation are only available to the human modeller subject to exercising discretion

cf. providing an LSD account to describe the framework for interaction (+ maybe an interface to impose this framework)



Role of LSD as adjunct to script

Interpreting external agent actions and LSD …

actions triggered by observables ("oracles")

making redefinitions (of "handles")

subject to suitable dependencies (reflected in the current definitive script)

Illustrations from JUGS

- pouring actions are automated in jugsBeynon1988 so as to effect state changes identified as "program-like"
- pupil can fill jug A, pour from jug A to jug B etc, only teacher can change jug capacities, only modeller can set `contentA` to exceed `capA`
- buttons limit what the "pupil user" of the JUGS model can do

Illustrations from JUGS

In "making redefinitions subject to suitable dependencies": may need to set up the dependencies to suit a particular action

See the observables mediating the pouring activities in JUGS: to pour from one jug to another rather than fill or empty a jug …

```
if (int(input) == 5)    {     ## pouring option selected
          content5 = contentA + contentB;
          contentB is content5 - contentA;
          option = valid6 ? 6 : 7;
        } else …
```

Linking definitive and procedural

When bridging internal to external state changes …

internal variables in a procedure attaining values that are not be viewed externally (e.g. local variables)

… should be treated differently from

external observables whose values are manipulated in a procedure

Linking definitive and procedural

A traditional procedure will typically not disclose the intermediate values of the variables it manipulates …

… cf. good programming practice – "information hiding"

BUT if a procedure affects the values of external observables these are then by default not computed …

… use `eager()` to expose intermediate state changes to values of external observables

Illustrating `eager()` in JUGS

```
if (avail(option)) {
  switch (option) {
     case 1:
          contentA = contentA + 1; break;
          ….
     }
     eager();
     ## updates contentA externally
     step++;
  }
```

Discretion over agent action

With effective use of definitive programming principles can exploit the flexibility of agent interaction in EDEN …

modeller intervention on-the-fly / "at run-time"

free interleaving of agent interactions

potential for concurrency

…. contrast jugsBeynon1988 and jugsBeynon2008

as a case study in use and development of EDEN

Contrasting characteristics

procedural perspective	definitive perspective
function	artefact
abstraction	instrument
optimisation	evolving interpretation
algorithm	experiment
efficiency	skilful interaction
goal-driven	flexibility

Programming as specialisation

Modelling with definitive scripts allows the modeller to explore many possible scenarios in an open-ended fashion …

... this may lead to the identification of particular patterns of interaction and interpretation that can be imposed upon a potential ‘user’ of the model

… this is what is meant by a definitive *program*

Illustration from JUGS

Following the conventions of the JUGS program, the user never encounters a situation in which the state is stable (‘awaiting input’) and it is possible *both* to pour from jug A to jug B *and* vice versa.

This means that we only need a `Pour` button, though in fact the underlying mechanism is derived from a previous version of the model that admitted both kinds of pouring. These can still be accessed by entering:

`input = 6;` or `input=7;`