Programming from an Empirical Modelling perspective 2

From modelling with definitive scripts to programming:

- representing state in programming
- behaviour of programs
- the semantics of programs

State

States relevant to programming ...

- state within the executing program
- external state: what is visible?
- state in respect of interaction
- state in program development
- state significant in the external world

- Diverse representations are required:
- *state within the executing program*
  - Program variables, machine locations
- - *external state: what is visible?*
  - Graphical techniques
- - *state in respect of interaction*
  - Statechart, message sequence diagram

- Diverse representations required …
- *state in program development*

  UML diagrams, prototypes
- *state significant in the external world*

  apprehended by the human interpreter

cf. Brian Cantwell-Smith on semantics …

States within oxoGardner1999

Definitive scripts express …
- internal state – contents of squares
- visible state – appearance of the board
- interaction state: whose turn is it?
- state of development
- state of mind of the player: which square?

(demo)

Modelling with definitive scripts:
… a holistic view of state that integrates and conflates all the different perspectives

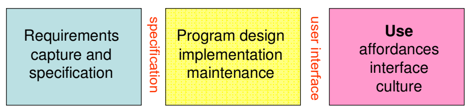
in contrast to

Programming-in-the-wild:
… an eclectic model of state in which many different strategies for representation and interpretation are jumbled up together

Two emphases

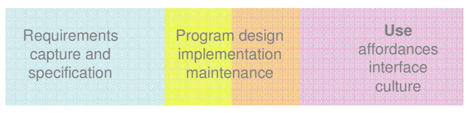
- Empirical Modelling encourages us to consider programming in a holistic way, using similar principles to deal with the entire process of development from conception to customisation and use
- It can also has a means to represent the specific activity that is captured by a traditional program

Traditional programming


Requirements capture and specification
specification
Program design implementation maintenance
user interface
Use affordances interface culture

Identifying agency in the machine-like components and in the human context for use	*constructing and programming the machine-like components*	*human factors study*
Framing goals for the design protocols for interaction and interpretation	*designing program by identfying objects and functions*	*interface design*
	technical interface development	*empirical studies of use*
		prototyping
e.g. devise UML	*e.g. writing Java code*	*e.g. goals, operators, methods (GOMS) evaluation*

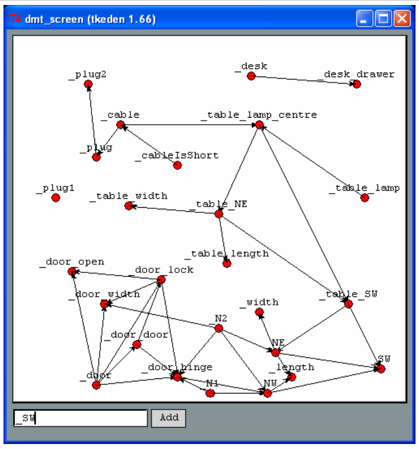
Empirical Modelling


Requirements capture and specification
Program design implementation maintenance
Use affordances interface culture

develop scripts in isolation as "furry blobs" that represent the observables and dependencies associated with putative machine-like components and human interactions and interpretations	*identify and document reliably reproducible sequences of redefinition / chains of "furry blobs" that correspond to programmable automatable machine behaviours and ritualisable human behaviours and interfaces*	*exercise, explore, customise, revise and adapt sequences of redefinition and interpretation to reflect emerging and evolving patterns of interaction and interpretation; extend and augment observables to support additional functionalities combining scripts*

Objects and dependencies

- An **object** corresponds to a particular way of associating observables: grouping together observables according to whether they exist concurrently
- A **dependency** links observables according to how they are linked in change: whether making a change to the value of one observable necessarily entails changing others



Object model vs. account of observation

An account of observation is a more primitive concept than an object model: it entails fewer preconceptions about what might be observed …

"**Definitive scripts are neutral wrt agent's views & privileges**"

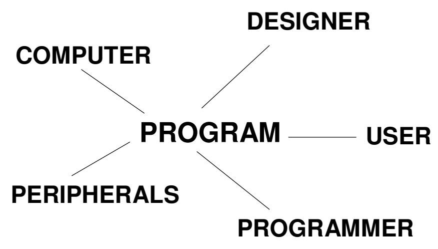
Object model vs. account of observation 2

Definitive script expresses different agent views and privileges to transform

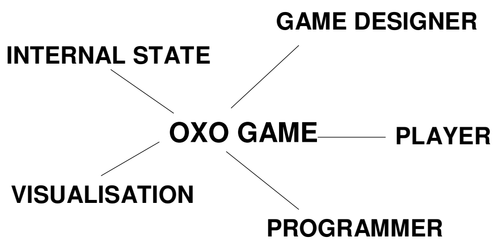
(cf. subject-oriented programming)

"What architect can do vs what user can do"

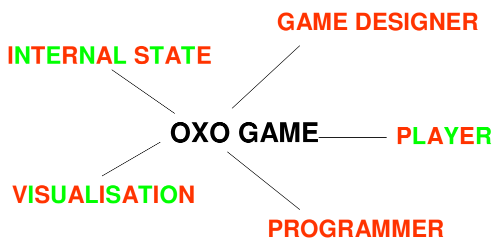
… highlights how the script affords *views of* and *access to* possible transformations



… compare this with the OXO laboratory

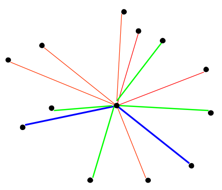


… Behaviour as programmed state change



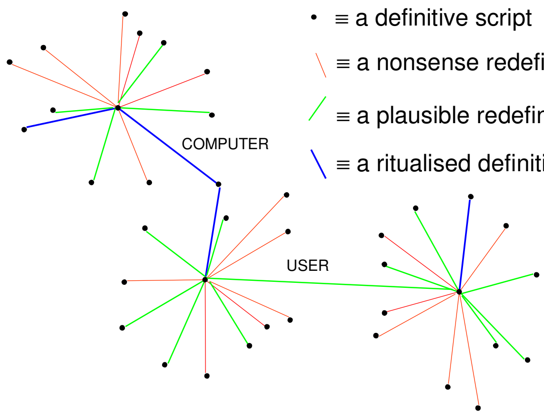
Static and dynamic elements of state

Definitive scripts as “furry blobs”



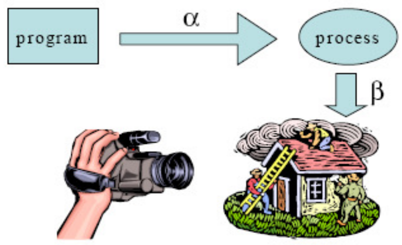
- • ≡ a definitive script
- \ ≡ a nonsense redefinition
- / ≡ a plausible redefinition
- \ ≡ a ritualised definition

Plausible : *could* open the desk drawer
 – note continuous spectrum of redefinitions
Ritualised : door *automatically* closes after being opened
Nonsense : opening the drawer makes the room smaller



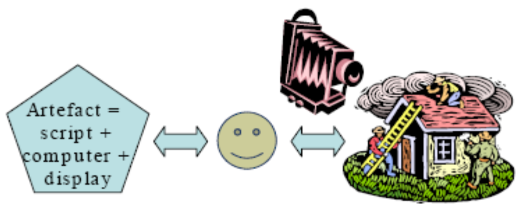
- • ≡ a definitive script
- \ ≡ a nonsense redefinition
- / ≡ a plausible redefinition
- \ ≡ a ritualised definition

Semantic Relations (I)



The semantics of a traditional program

Semantic Relations (II)



The semantics of a definitive program

Classical programming …1

Behaviour is derived from a pre-specified conception of function and purpose …

… based on interactions whose outcomes are reliable and for which the mode of interpretation is determined in advance

…motivates declarative approaches

Classical programming …2

… motivates declarative approaches:

```
output=F(input)
```

… problematic to deal with a dynamic input, as in playing a game

… hence add "lazy evaluation" to model as

```
stream_of_output=F(stream_of_input)
```

Significance of interpretation …

Miranda *can* be viewed as a definitive notation over an underlying algebra of functions and constructors

BUT this interpretation emphasises

program design as a state-based activity

NOT

declarative techniques for *program specification*

Illustrative example

… a version of 3D OXO written in the functional programming language Miranda

… to be compared with oxoJoy1994 which was in some respects 'derived' from it

Two experimental systems!

A definitive Miranda ("admira"): definitive notation with general functional programs and types as operators & data structures

The Kent Recursive Calculator (KRC): developing functional programs by framing definitive scripts

Objects vs observations 1

A definitive script

represents the atomic transformations of a geometric symbol

DoNaLD room can be transformed through redefinition in ways that correspond ‘exactly’ to the observed patterns of change associated with opening a door, or moving a table

Objects vs observations 2

Thesis:

- set of atomic transformations of a symbol captures its semantics [cf. Klein's view of a geometry as “the study of properties invariant under a family of transformations”]
- Illustration via a geometric pun (demo)

Is the DoNaLD room an object in the class-based OOP sense? 1

Can view each room transformation as a method for the object

BUT

definitive script is an object specification

only if

set_of_transformations_performed_on_room is **circumscribed**

Is the DoNaLD room an object in the class-based OOP sense? 2

Circumscription creates objects

BUT

a definitive script merely reflects observed latent transformations

Comprehending / designing an object = knowing / determining everything we can do with it

BUT

definitive script doesn't circumscribe the family of transformations that we can apply

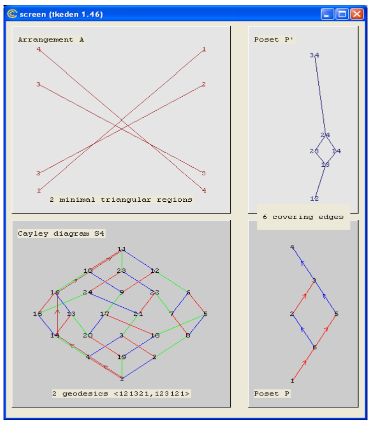
From logic to experience

- the computer enables us to use logical constructs to specify relationships that admit reliable interpretations and support robust physical realisations
- human skill and discretion plays a crucial role in crafting ritualisable experiences
- NB classical computer science doesn’t take explicit account of robust physical realisations or ritualisable experience

From experience to logic?

- open-ended interaction with what is experienced is a means to representing with a high degree of realism and subtlety (cf. the strained representation of observables in the Miranda 3D OXO)
- mathematical concepts such as abstract lines as “realised” in this fashion

linesBeynon1991



The linesBeynon1991 script …

```
real sc,size
point O,Oa,Ob
O,Oa,Ob={500,500},O-{size,size},O+{size,size}
size,sc = 380.0,2*size
real a12,a23,a34,b12,b23,b34
point A1,A2,A3,A4,B1,B2,B3,B4
line l1,l2,l3,l4
# a12 and b12 determine the distances between the L and R ends of lines 1 and 2
a12=1.0
a23=5.0
a34=2.0
b12=2.0
b23=5.0
b34=1.0
# A1 and B1 are the left and right endpoints of line 1
A1=Oa
A2=Oa+{0, a12 div (a12+a23+a34)}*sc
A3=Oa+{0,(a12+a23) div (a12+a23+a34)}*sc
A4=Oa+{0,(a12+a23+a34) div (a12+a23+a34)}*sc
B1=Ob
B2=Ob-{0,(b12) div (b12+b23+b34)}*sc
B3=Ob-{0,(b12+b23) div (b12+b23+b34)}*sc
B4=Ob-{0,(b12+b23+b34) div (b12+b23+b34)}*sc
l1=[A1,B1]
l2=[A2,B2]
l3=[A3,B3]
l4=[A4,B4]

label j1,j2,j3,j4,k1,k2,k3,k4
j1 = label("1",A1)
j2 = label("2",A2)
j3 = label("3",A3)
j4 = label("4",A4)
k1 = label("1",B1)
k2 = label("2",B2)
k3 = label("3",B3)
k4 = label("4",B4)
```

```
# this is the DoNaLD script that defines Figure 1(b) in the viewport "POSET"
viewport POSET

# r12 determines the LR position of the line 1,2 intersection
# x12 is to be the crossing index of the line 1,2 intersection
#     eg this is 1 if lines 1 and 2 are the top pair at their pt of Xn
real r12,r23,r34,r13,r24,r14
int x12,x23,x34,x13,x24,x14
r12,r23,r34 = a12 div b12 , a23 div b23, a34 div b34
r13 = (a12+a23) div (b12+b23)
r24 = (a23+a34) div (b23+b34)
r14 = (a12+a23+a34) div (b12+b23+b34)

# z123 = 1 if line 1 crosses line 2 before line 3 crosses line 2 in LR order
int z123,z124,z134,z234
int Z123,Z124,Z134,Z234
z123 = if r12<r23 then 1 else 0
z124 = if r12<r24 then 1 else 0
z134 = if r13<r34 then 1 else 0
z234 = if r23<r34 then 1 else 0
Z123 = if r13<r12 then 1 else 0
Z124 = if r14<r12 then 1 else 0
Z134 = if r14<r13 then 1 else 0
Z234 = if r24<r23 then 1 else 0

# x12 calcs the crossing index for lines 1 and 2
# "by default" this is 1 but is incremented if line 3 or 4 crosses line 1
# before line 2 crosses line 1 in LR order
x12 = 1+Z123+Z124
x13 = 1+(1-Z123)+Z134
x14 = 1+(1-Z124)+(1-Z134)
x23 = 2-z123+Z234
x24 = 2-z124+(1-Z234)
x34 = 3-z134-z234
```

```
# these are the points of the poset of intersections
# v is a vertical, m a global magnification factor
int v,m
v,m=8,50
point orig,p12,p23,p34,p13,p24,p14
p12 = orig+{x12,r12*v}*m
p23 = orig+{x23,r23*v}*m
p24 = orig+{x24,r24*v}*m
p34 = orig+{x34,r34*v}*m
p14 = orig+{x14,r14*v}*m
p13 = orig+{x13,r13*v}*m
orig = {400,400}

# Line l1213 occurs in the poset if the intersection of lines 1 and 2
# and the intersection of lines 1 and 3 corresponds to a covering edge
# Line l1213 is present if the parameter d1213 evaluates to 1
# it otherwise contracts to the origin
line l1213,l1214,l1314,l2324,l1223,l1224,l1334,l2334,l1323,l1424,l2434,l1434
int d1213,d1214,d1314,d2324,d1223,d1224,d1334,d2334,d1323,d1424,d2434,d1434
l1214 = [p12*d1214,p14*d1214]
l1323 = [p13*d1323,p23*d1323]
l1224 = [p12*d1224,p24*d1224]
l1424 = [p14*d1424,p24*d1424]
l2324 = [p23*d2324,p24*d2324]
l1334 = [p13*d1334,p34*d1334]
l1213 = [p12*d1213,p13*d1213]
l1434 = [p14*d1434,p34*d1434]
l1314 = [p13*d1314,p14*d1314]
l1223 = [p12*d1223,p23*d1223]
l2434 = [p24*d2434,p34*d2434]
l2334 = [p23*d2334,p34*d2334]

# d1213 is 1 if the crossing index of lines 1 and 2 differs from that of
# lines 1 and 3  and line 4 doesn't cross line 1 between its points
# of intersection with lines 2 and 3
d1334 = if !(x13==x34) && ((r23-r13)*(r23-r34)>0) then 1 else 0
d2334 = if !(x23==x34) && ((r13-r23)*(r13-r34)>0) then 1 else 0
d1224 = if !(x12==x24) && ((r23-r12)*(r23-r24)>0) then 1 else 0
d1223 = if !(x12==x23) && ((r24-r12)*(r24-r23)>0) then 1 else 0
d1323 = if !(x13==x23) && ((r34-r13)*(r34-r23)>0) then 1 else 0
d1424 = if !(x14==x24) && ((r34-r14)*(r34-r24)>0) then 1 else 0
d2434 = if !(x24==x34) && ((r14-r24)*(r14-r34)>0) then 1 else 0
d1434 = if !(x14==x34) && ((r24-r14)*(r24-r34)>0) then 1 else 0
d1213 = if !(x12==x13) && ((r14-r12)*(r14-r13)>0) then 1 else 0
d1214 = if !(x12==x14) && ((r13-r12)*(r13-r14)>0) then 1 else 0
d1314 = if !(x13==x14) && ((r12-r13)*(r12-r14)>0) then 1 else 0
d2324 = if !(x23==x24) && ((r12-r23)*(r12-r24)>0) then 1 else 0
```

```
int numcovedge
numcovedge = d1213+d1223+d1224+d1214+d1314+d1334+d1323+d2324+d2334+d2434+d1434+d1424

int g1213,g1214,g1314,g2324,g1223,g1224,g1334,g2334,g1323,g1424,g2434,g1434
g1213 = if (((p12.1-p13.1)*(p12.2-p13.2)) < 0) then 0 else 1
g1214 = if (((p12.1-p14.1)*(p12.2-p14.2)) < 0) then 0 else 1
g1314 = if (((p13.1-p14.1)*(p13.2-p14.2)) < 0) then 0 else 1
g2324 = if (((p23.1-p24.1)*(p23.2-p24.2)) < 0) then 0 else 1
g1223 = if (((p12.1-p23.1)*(p12.2-p23.2)) < 0) then 0 else 1
g1224 = if (((p12.1-p24.1)*(p12.2-p24.2)) < 0) then 0 else 1
g1334 = if (((p13.1-p34.1)*(p13.2-p34.2)) < 0) then 0 else 1
g2334 = if (((p23.1-p34.1)*(p23.2-p34.2)) < 0) then 0 else 1
g1323 = if (((p13.1-p23.1)*(p13.2-p23.2)) < 0) then 0 else 1
g1424 = if (((p14.1-p24.1)*(p14.2-p24.2)) < 0) then 0 else 1
g2434 = if (((p24.1-p34.1)*(p24.2-p34.2)) < 0) then 0 else 1
g1434 = if (((p14.1-p34.1)*(p14.2-p34.2)) < 0) then 0 else 1


int r1213,r1214,r1314,r2324,r1223,r1224,r1334,r2334,r1323,r1424,r2434,r1434
r1213 = if ((r12-r13) < 0) then 0 else 1
r1214 = if ((r12-r14) < 0) then 0 else 1
r1314 = if ((r13-r14) < 0) then 0 else 1
r2324 = if ((r23-r24) < 0) then 0 else 1
r1223 = if ((r12-r23) < 0) then 0 else 1
r1224 = if ((r12-r24) < 0) then 0 else 1
r1334 = if ((r13-r34) < 0) then 0 else 1
r2334 = if ((r23-r34) < 0) then 0 else 1
r1323 = if ((r13-r23) < 0) then 0 else 1
r1424 = if ((r14-r24) < 0) then 0 else 1
r2434 = if ((r24-r34) < 0) then 0 else 1
r1434 = if ((r14-r34) < 0) then 0 else 1
```

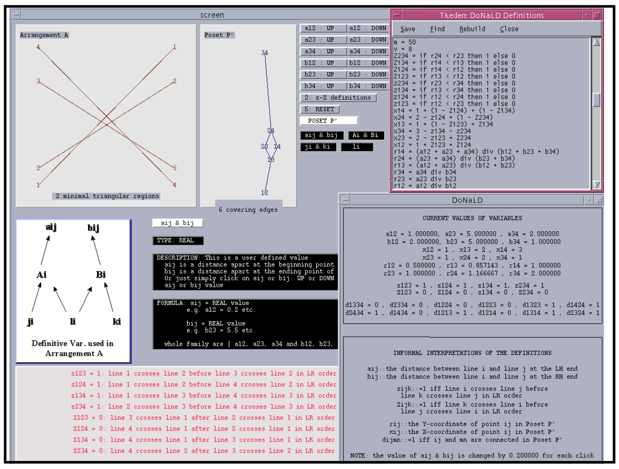
```
int U1213,U1214,U1314,U2324,U1223,U1224,U1334,U2334,U1323,U1424,U2434,U1434
U1213 = g1213 * r1213 * d1213
U1214 = g1214 * r1214 * d1214
U1314 = g1314 * r1314 * d1314
U2324 = g2324 * r2324 * d2324
U1223 = g1223 * r1223 * d1223
U1224 = g1224 * r1224 * d1224
U1334 = g1334 * r1334 * d1334
U2334 = g2334 * r2334 * d2334
U1323 = g1323 * r1323 * d1323
U1424 = g1424 * r1424 * d1424
U1434 = g1434 * r1434 * d1434
U2434 = g2434 * r2434 * d2434

int u1213,u1214,u1314,u2324,u1223,u1224,u1334,u2334,u1323,u1424,u2434,u1434
u1213 = g1213 * (1-r1213) * d1213
u1214 = g1214 * (1-r1214) * d1214
u1314 = g1314 * (1-r1314) * d1314
u2324 = g2324 * (1-r2324) * d2324
u1223 = g1223 * (1-r1223) * d1223
u1224 = g1224 * (1-r1224) * d1224
u1334 = g1334 * (1-r1334) * d1334
u2334 = g2334 * (1-r2334) * d2334
u1323 = g1323 * (1-r1323) * d1323
u1424 = g1424 * (1-r1424) * d1424
u1434 = g1434 * (1-r1434) * d1434
u2434 = g2434 * (1-r2434) * d2434

int v1213,v1214,v1314,v2324,v1223,v1224,v1334,v2334,v1323,v1424,v2434,v1434
v1213 = (1-g1213) * (1-r1213) * d1213
v1214 = (1-g1214) * (1-r1214) * d1214
v1314 = (1-g1314) * (1-r1314) * d1314
v2324 = (1-g2324) * (1-r2324) * d2324
v1223 = (1-g1223) * (1-r1223) * d1223
v1224 = (1-g1224) * (1-r1224) * d1224
v1334 = (1-g1334) * (1-r1334) * d1334
v2334 = (1-g2334) * (1-r2334) * d2334
v1323 = (1-g1323) * (1-r1323) * d1323
v1424 = (1-g1424) * (1-r1424) * d1424
v1434 = (1-g1434) * (1-r1434) * d1434
v2434 = (1-g2434) * (1-r2434) * d2434
```

```
int V1213,V1214,V1314,V2324,V1223,V1224,V1334,V2334,V1323,V1424,V2434,V1434
V1213 = (1-g1213) * r1213 * d1213
V1214 = (1-g1214) * r1214 * d1214
V1314 = (1-g1314) * r1314 * d1314
V2324 = (1-g2324) * r2324 * d2324
V1223 = (1-g1223) * r1223 * d1223
V1224 = (1-g1224) * r1224 * d1224
V1334 = (1-g1334) * r1334 * d1334
V2334 = (1-g2334) * r2334 * d2334
V1323 = (1-g1323) * r1323 * d1323
V1424 = (1-g1424) * r1424 * d1424
V1434 = (1-g1434) * r1434 * d1434
V2434 = (1-g2434) * r2434 * d2434

%donald
int in12, in23, in24, in34, in14, in13
in13 = U1314+U1334+U1323+u1213+v1213+V1334+V1323+V1314
in23 = U2324+U2334+u1223+u1323+v1323+v1223+V2324+V2334
in24 = U2434+u2324+u1224+u1424+V2434+v2324+v1224+v1424
in34 = u1334+u2334+u1434+u2434+v1334+v2334+v1434+v2434
in14 = U1424+U1434+u1214+u1314+V1424+V1434+v1214+v1314
in12 = U1213+U1214+U1223+U1224+V1224+V1223+V1213+V1214
```

```
int V1213,V1214,V1314,V2324,V1223,V1224,V1334,V2334,V1323,V1424,V2434,V1434
V1213 = (1-g1213) * r1213 * d1213
V1214 = (1-g1214) * r1214 * d1214
V1314 = (1-g1314) * r1314 * d1314
V2324 = (1-g2324) * r2324 * d2324
V1223 = (1-g1223) * r1223 * d1223
V1224 = (1-g1224) * r1224 * d1224
V1334 = (1-g1334) * r1334 * d1334
V2334 = (1-g2334) * r2334 * d2334
V1323 = (1-g1323) * r1323 * d1323
V1424 = (1-g1424) * r1424 * d1424
V1434 = (1-g1434) * r1434 * d1434
V2434 = (1-g2434) * r2434 * d2434


%donald
int in12, in23, in24, in34, in14, in13
in13 = U1314+U1334+U1323+u1213+v1213+V1334+V1323+V1314
in23 = U2324+U2334+u1223+u1323+v1323+v1223+V2324+V2334
in24 = U2434+u2324+u1224+u1424+V2434+v2324+v1224+v1424
in34 = u1334+u2334+u1434+u2434+v1334+v2334+v1434+v2434
in14 = U1424+U1434+u1214+u1314+V1424+V1434+v1214+v1314
in12 = U1213+U1214+U1223+U1224+V1224+V1223+V1213+V1214
```

Interesting comparisons …

- the lines script as not object-oriented – most of its core observables are associated with relationships that cannot be identified with any single object
- the lines script as resembling a functional programming script in its homogeneity (“all definitions”), but associated with directly accessible external observables …



```
aij & bij

TYPE: REAL

DESCRIPTION: This is a user defined value
  aij is a distance apart at the beginning point
  bij is a distance apart at the ending point of
  Or just simply click on aij or bij: UP or DOWN
  aij or bij value

FORMULA: aij = REAL value
        e.g. a12 = 0.2 etc.

        bij = REAL value
        e.g. b23 = 5.5 etc.

  whole family are [ a12, a23, a34 and b12, b23,
```

```
z123 = 1: line 1 crosses line 2 before line 3 crosses line 2 in LR order
z124 = 1: line 1 crosses line 2 before line 4 crosses line 2 in LR order
z134 = 1: line 1 crosses line 3 before line 4 crosses line 3 in LR order
z234 = 1: line 2 crosses line 3 before line 4 crosses line 3 in LR order
Z123 = 0: line 3 crosses line 1 after line 2 crosses line 1 in LR order
Z124 = 0: line 4 crosses line 1 after line 2 crosses line 1 in LR order
Z134 = 0: line 4 crosses line 1 after line 3 crosses line 1 in LR order
Z234 = 0: line 4 crosses line 2 after line 3 crosses line 2 in LR order
```

```
m = 50
v = 8
Z234 = if r24 < r23 then 1 else 0
Z134 = if r14 < r13 then 1 else 0
Z124 = if r14 < r12 then 1 else 0
Z123 = if r13 < r12 then 1 else 0
z234 = if r23 < r34 then 1 else 0
z134 = if r13 < r34 then 1 else 0
z124 = if r12 < r24 then 1 else 0
z123 = if r12 < r23 then 1 else 0
x14 = 1 + (1 - Z124) + (1 - Z134)
x24 = 2 - z124 + (1 - Z234)
x13 = 1 + (1 - Z123) + Z134
x34 = 3 - z134 - z234
x23 = 2 - z123 + Z234
x12 = 1 + Z123 + Z124
r14 = (a12 + a23 + a34) div (b12 + b23 + b34)
r24 = (a23 + a34) div (b23 + b34)
r13 = (a12 + a23) div (b12 + b23)
r34 = a34 div b34
r23 = a23 div b23
r12 = a12 div b12
```

```
CURRENT VALUES OF VARIABLES

a12 = 1.000000, a23 = 5.000000 , a34 = 2.000000
b12 = 2.000000, b23 = 5.000000 , b34 = 1.000000
x12 = 1 , x13 = 2 , x14 = 3
x23 = 1 , x24 = 2 , x34 = 1
r12 = 0.500000 , r13 = 0.857143 , r14 = 1.000000
r23 = 1.000000 , r24 = 1.166667 , r34 = 2.000000

z123 = 1 , z124 = 1 , z134 = 1, z234 = 1
Z123 = 0 , Z124 = 0 , z134 = 0 , Z234 = 0

d1334 = 0 , d2334 = 0 , d1224 = 0 , d1223 = 0 , d1323 = 1 , d1424 = 1
d2434 = 1 , d1434 = 0 , d1213 = 1 , d1214 = 0 , d1314 = 1 , d2324 = 1
```

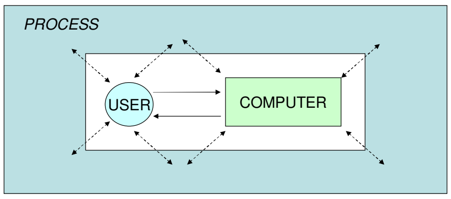
```
INFORMAL INTERPRETATIONS OF THE DEFINITIONS

aij::the distance between line i and line j at the LH end
bij::the distance between line i and line j at the RH end

zijk::=1 iff line i crosses line j before
 line k crosses line j in LR order
Zijk::=1 iff line k crosses line i before
 line j crosses line i in LR order

rij::the Y-coordinate of point ij in Poset P'
xij::the X-coordinate of point ij in Poset P'
dijmn::=1 iff ij and mn are connected in Poset P'

NOTE: the value of aij & bij is changed by 0.200000 for each click
```

Features of the lines model …

- directly accessible external observables: z123 = 1 means that line 1 crosses line 2 before line 3 crosses line 2 in L-to-R order
- the ideal geometry as associated with a mode of interaction with the model (subject to being able to enhance the accuracy of arithmetic indefinitely on-the-fly)

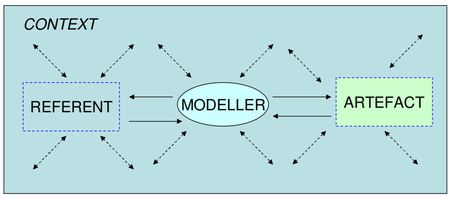
Programming from two perspectives

- a program is conceived with reference to how its behaviour participates in a wider process with functional objectives: states emerge as the side-effects of behaviours
- a computer artefact is developed so as to reflect the agency within an environment: the artefact and environment evolve until (possibly) program-like processes emerge



Conventional programs as embedded in *processes* of interaction with the world

Programs are understood in relation to processes in their surrounding environment



Artefacts and their referents as sculpted out of open interaction with the world

States of the referent and the artefact are connected through experience of interacting with the referent and the artefact

… but this presents some philosophical challenges …

An EM perspective on programming …
… some problematic issues

In focusing on current state-as-experienced, we have some problems to resolve:

- Behaviour raises questions about agency: what is the status of a “computer” action?
- How do we deal with state-as-experienced in semantic terms?
- How do we make science of activities in which human interpretation is so critical?