1992

3 November, Tuesday 4pm in P523

Organisation of the talk. (1)

Software Development using Definitive Scripts:
Experiments and Observations

Meurig Beynon and Simon Yung

****

A definitive script is a set of definitions that describes the dependencies between the values of procedural variables. In typical use, a script represents relationships between observations of a physical object whose state can be transformed by experiment.

This talk will describe how the study of definitive scripts has led us to look at software development in a new light. The main themes of the talk, to be illustrated by examples of software developed using our approach, are:

* foundations for programming in observation and experiment
* programming as modelling
* what is a program?
* design vs simulation
* from agents and privileges to protocols
* synchronous propagation of state-change
* new abstractions for state.

The software experiments on which our observations are based include:

Expts in design and modelling of objects
Expts in concurrent systems simulation
Expts in reactive systems specification
Expts in translating definitive models into procedural programs
Expts in abstract development of functional programs

The aim of the talk is to assess the prospects for future development of our concepts and techniques as a new basis for software construction.

SLIDES

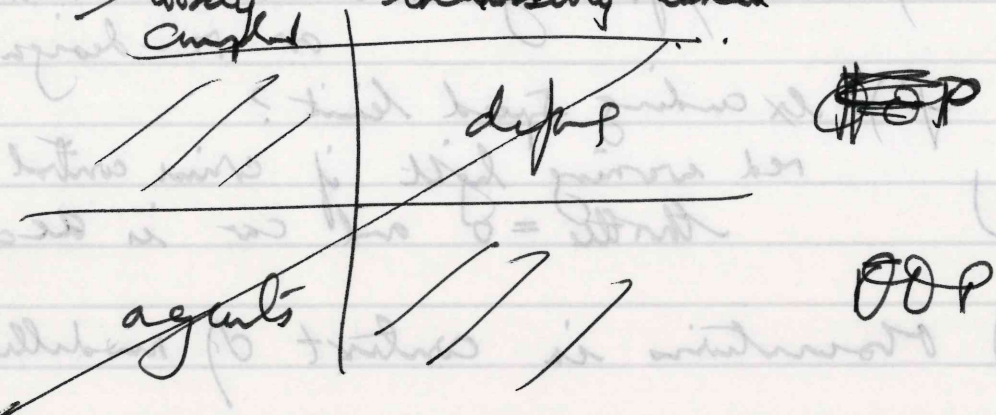
1) VCCS.

2) Abstract description of VCCS script
LSD description.

3) Concept: programming as modelling | DESCRIPTION → PRESCRIPTION

4) How observations are linked in change
loosely coupled / indivisibly linked

5)



"Definitive programming" Agent-oriented modelling over definitive types of state
e.g. button: colour speed loosely coupled

6) Relate to VCCS: examples of observations and of their interrelationships (p6')
Ells in description: various sorts of indivisibility

7) Observation and experiment p.6
Issues 1) 2) 3)
Time, expectation, conventions for 'what if?' observation.

8) Mathematics: foundations for functions variables

9) Quotation: Russian historian of math.

Observations & Expts in relation to Modelling

10) How represent observation / expt : definitive script.

Speedometer example.

a) "semantics of semantics" transformations like OOP
b) designer — what on display, how, take account of negative speed.
c) modes of propagation : mechanical link vs sampling
d) idealisation : speedo shows actual speed?
e) views / privileges : driver can't move speedo or re-design etc.
f) exceeding speed limit?
red warning light if cruise control has throttle = 0 and car is decelerating downhill

individually linked observations
maintaining an expt'l context

11) Observation in context of modelling (p7)

a) Engineering justification : small expt & complex system
b) Observation of system → { individually linked / independent state-change
B.R. quote : extraction of agents of CSP.
c) 1-agent model over definitive type of state

PROGRAMMING vs MODELLING

12) TRAIN : LSD spec. animation (p8)
Who can / might do what? Eng. significance.
13) FUGS FUGS → EDEN — dyns.

form / content
B.C. Smith.

~~ABSTRACT~~ OPER. MODEL | INTERPRETED MODEL

cf. 10 a) - f)

The VCCS.

What creates it?

"state of running simulation"

DEFNS

[dynamic model] — defns of analogue variables
def's fn's of time: position, vel, accel.

"state of (a frozen) screen"

[geometric defns] — defn of graphical line drawings
speeds, car display, position, accel.
force animation

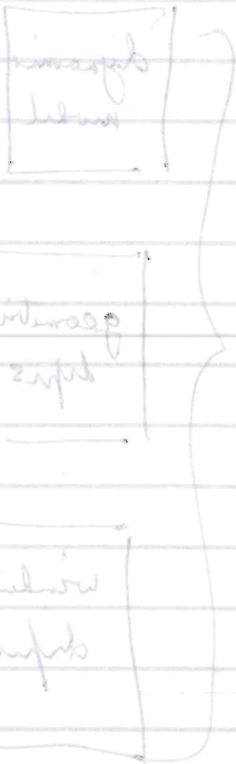
[window defns] — disposition of screen elts
status of buttons — PRIVILEGES.
textual display — VIEWS

AGENTS

[interface spec] — LSD spec for driver + interface agents

The VCCS

What even is it?



TRIDENT
EDEN } significant points.

Cruise Control — reactive systems spec. design.

LSD + ADM — train

jugs

abstract data. programming

Programming as Modelling.

Not new. **SIMULA**

How it works?

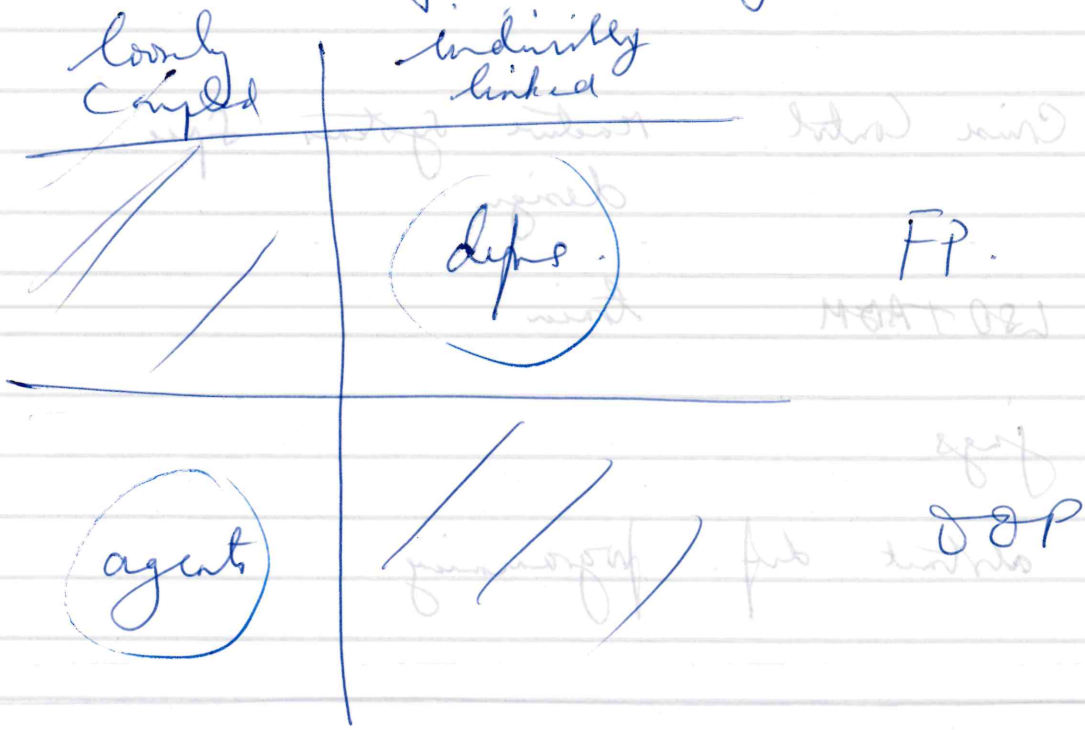
Underlying Presumptions about rhalle change of expt.
Explicit modelling of the artefacts between agents.

Requirements specification as "deciding what observations must be made"

Color of a car doesn't affect speed
no point in Modeling whether there is ash in the cigarette tray.
or whether its Monday or Tuesday.

How abstraction are linked in change

How abstractions are grouped into objects X



Batsmen cross — { "in cricket" moves from L to R
game is won

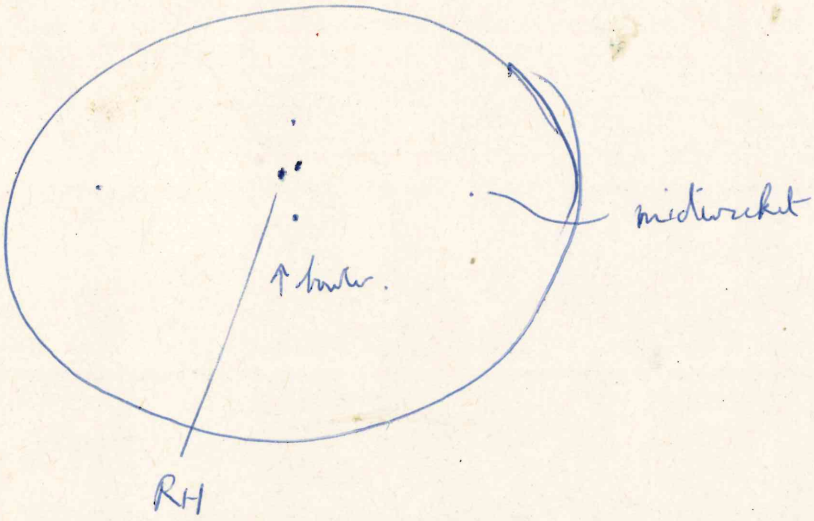
visible actions affect "indirectly" context for reference & interpretation.

Controversial catch — is the game won?
did he hit the ball?

"non-computation" involved in interpretation & context changes.

c.f. p.6 what a definitive script conveys

Cricket pitch



Change ends not midwicket

Batsman cross, may not be midwicket. (*)

Dependency "how output" ⟺ "what happens"

⟵ (*)

⟶ if 6 needed last ball, batsman ventures shot.

Overall idea: not v. good at representing v simple forms of state change

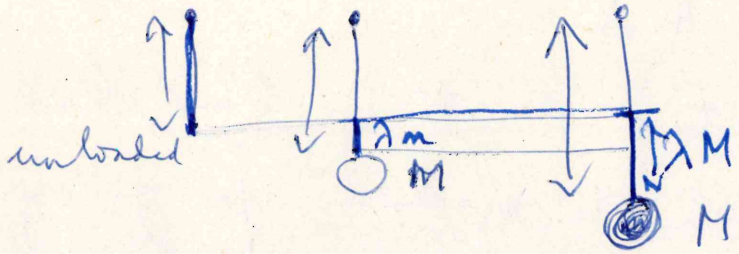
⟶ objects, algorithms, procedures not satisfactory primitives

time and view are involved in perception of state.

How do we distinguish diff. kinds of state change?

Observation / experiment

Hookes Law.



"Extension proportional to mass subject to some physical assumptions." (Young's Modulus)

"String doesn't weigh anything"

Basic framework

Experimentator changes the mass — waits for equilibrium

then measures the extension

correlates (mass, extension) pairs.

Time don't measure mass & extension at the same time.

(mass, extension) pairs ∈ context

Assumptions not heating, gravity doesn't fluctuate

No change apart from the experimentator's action

Conventions for observation that need to be rigorously applied to get useful results

When we say

$$e = m\lambda$$

we are asserting something like there is an independent "reliable method" for getting e from m (to within expt error). Computing e from m is an independent activity that is also fundamentally related to state-based expt of some kind.

<u>Falsifiability</u>

Each time is another time — perhaps expt won't work next time. No guarantee, unlike a math. computation or piece of logic.

Consider if we had no means to detect ~~heat~~ temperature — in this case, how would ~~measure~~ meas come out?

e and m would seem to be independent measurements

This would affect ~~how~~ what we perceive the effect of an action to be. "m doesn't affect e in a way that is easy to predict"

Refined model — pass current via wire (meas it) vs consume expt on a wire
correlate current vs ext vs mass — temperature situation

Summary of experiment / observation

1) "Time" repts a context for observation

What we deem to be synchronous observation ⟺ constant

2) Pre-conceived idea about what will happen ⟹ makes it "an expt"
Expectation ideally fulfilled. Start of
Paradoxically, also poss. that it might not.
"What if" study-based nature of the expt.

3) V. significant role for conventions of observation

"Other things being equal"
"To within expt'l error"
"If nothing else intervenes"

Expectation expressed in terms of functional relationship

correlation between observations

Verification of correlation via appeal to an independent "expt". That is: don't need to make the ~~expt~~ observation; can model compute the expected result by some other physical process.

Mathematical perspective

* functional abstraction ⟺ rel between observations made "in same context"
* to formulate need "genuine variable", not status. Can be measured in diff contexts, has identity

Historically primary notions of { functions / variable

< computation >

Talk format initially:

VECS display

preliminary discussion (cf. research proposal)
central ideas: agents, agents, views, privileges

Programming as modelling

"Reactive system" illustrations | LSD spec. (IB)

Examples of loosely-coupled intrinsically linked change:
- speed transducer
- drive
- cruise controller
- cut-out

Definition scripts and recording expts for "Virtues of definitive script" "Testable, objective"

different from defn's serve in script < semantics of semantics
serve designer
model modes of propagati
model views/privileges

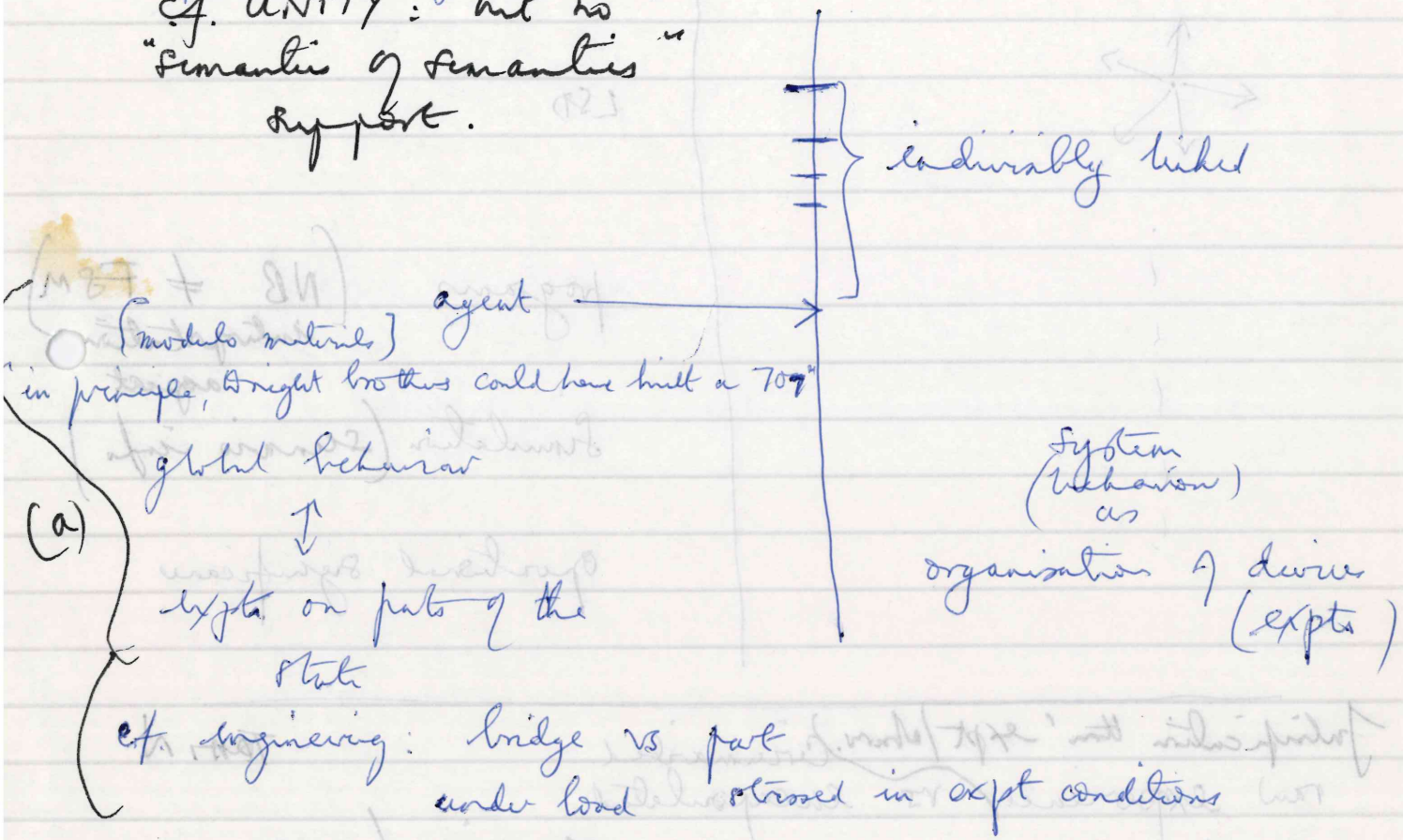
non-computable effects.

actual/idealised

freedom from idealisation "more alive"

Any pattern of state change in complex system
(known initiated)
can be modelled by recording observations and synchronised updates
("parallel assignment")

cf. UNITY: but no "semantics of semantics" support.



(c)
- 1-agent model over definitive reps of state.
- action ⟷ linked family of rdrfns
- designer plays the roles of agents

[dfns for the designer
INTELLIGIBILITY.
CONVENIENCE IN RETHINKING]

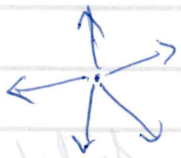
(b)
- Observation of system → { indivisibly linked state change / notion of independent state change
- [Bertrand Russell quote]

cf. CSP Tony Hoare

Programming as Modelling ←

Programming vs Modelling ↓

Train simulation not a program.

State-transition models	Interpretation
	LSD
	programs (NB ≠ FSM)
	interpretation aspect
	Simulation (scenario info.)
	operational significance

Justification thru' expt/observ. Encapsulated TRAIN

raw experience vs encapsulated

reasoning / proof

"I don't know what will happen next" vs "I believe my expectations about what can happen next are reliable"

Modelling Programming

Jacobs

EDEN —TRIDENT→ ~~EDEN~~ (⟷ C)

choose to restrict

am restricted

privilege can do

protocol must do

Two ways to construct { circumscribed / encapsulated } behaviours

1) chose the/a restricted set of privileges

[cf experimenter, could heat the wire / cut
do expt on a train / rocket etc]

2) provide an interface that restricts / possible actions
≡ construct an object

---

cf. batsman can do anything with the bat
e.g. lie on back with handle of bat in air
hold up etc.
in practice repertoire of more or less conventional shots.

design don't know what privileges most / appropriate advantageous
until we expt.
e.g. in mechanical device don't add safety stops
till after designed

---

de facto

Create objects when restrict usage e.g. as when demonstrate simulation in particular modes of change

But haven't really fixed objects
Can undo descriptions calmly, tolerant to unforeseen ways.

Key distinction: preconceived interaction vs genuine experiment | "defining the machine" atomic ops.

⑧ Relation to programming design "as we know it"

Initial focus: "exploratory programming paradigm"

≡ state-based modelling for requirement.

~~State~~ State-based modelling applies also to

- design status representation: version control
  (*) [alternative scripts ⟷ different designs
  generic part of script — template for design
  diff specialisations (as eg product range)
  [choice of parameters]
  "lexi" comprehensions of specialisations
  selection

[parallel to use of variables in mathematical exposition
FBR's parabola illustration —]

Programming centrally involves state

class — instance — statements about ~~set of states of~~ possible experiments

NOT formalised in OOP. class.h] create state class.c calling prog's

states co-exist: function of focus / nature of attention
state hierarchies of STATECHARTS ⟷ LSD agent

Dfn of a functional program can be viewed as built up via contemplation of a family of states.

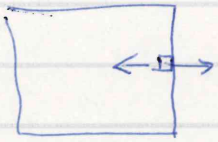
To remind programmer of the interpretation of a functional script create a family of states (simple definitive scripts) suitably related and composed! Devices at (*) apply.

Abstract programming environment based on scripts
Family of environments ⟶ functional program.

The future

Limitations of present model.

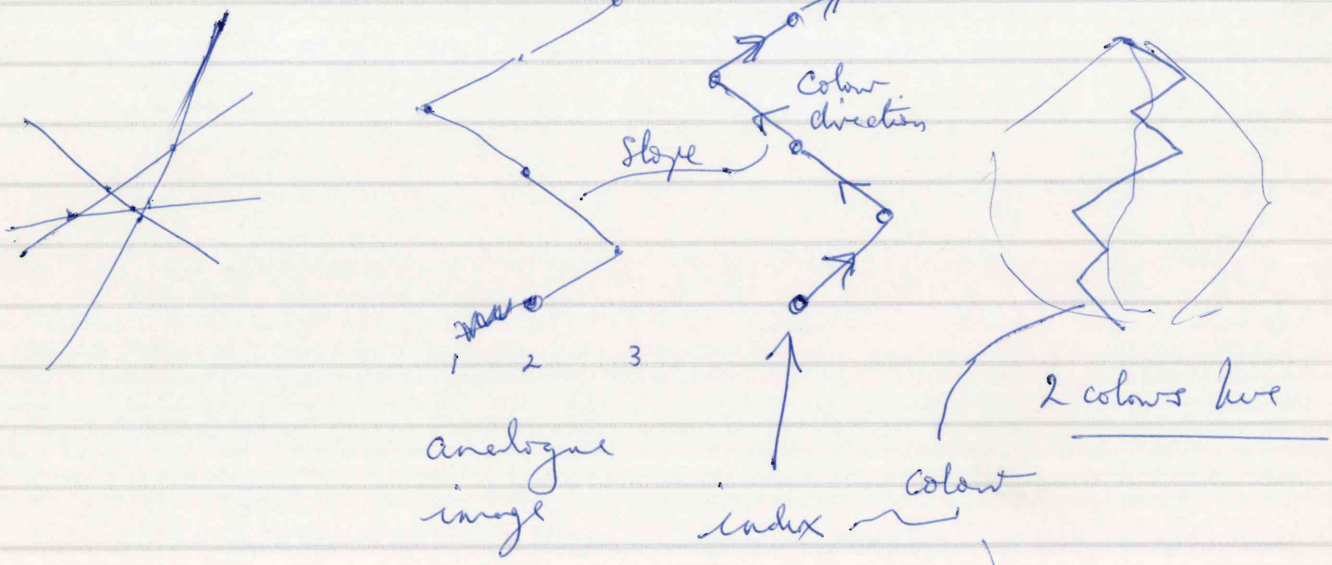
1) ~~form~~ of dependency indivisibly linked to state.



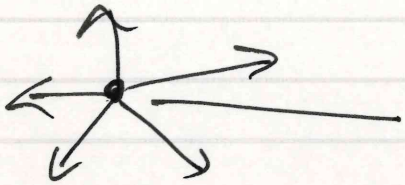
text editor

2) forms of dependency difficult to express by dyns.

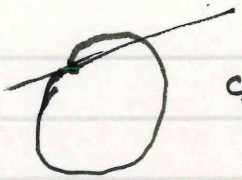
Lines demo.



The future



many sheets



of Riemannian surface

(same values: diff dep's)

bifurcation wrt action.
Singular pt in space.

Operation on DST model similar to graph contraction.

Identification of two nodes or more