

Whither with 'with'? – new prospects for programming

Nicolas Pope
Computer Science
University of Warwick
nwpopo@gmail.com

Elizabeth Hudnott
Computer Science
University of Warwick
lizzy.hudnott@gmail.com

Jonathan Foss
Computer Science
University of Warwick
jonny@dcs.warwick.ac.uk

Meurig Beynon
Computer Science
University of Warwick
wmb@dcs.warwick.ac.uk

Abstract

The role that dependency can play in developing programs is a longstanding topic of interest. The idea that spreadsheet principles are the key to new more accessible ways of programming continues to be a focus for current research. This paper further elaborates an idea whose development has been documented in several previous PPIG meetings: that programming can be best understood by construing human and automated agency in observational terms. Its main focus is on illustrating and discussing a new concept: a **with**-construct that has been introduced in a recently developed environment for making such construals (the 'MCE'). Preliminary work indicates that introducing this concept has a transformative impact both on the practice of making construals and its relationship to conventional programming.

1. Introduction

This paper could be regarded as a sequel to a paper presented at the 6th PPIG meeting in 1994 [6]. The title of that paper, by Beynon and Joy, was "Computer Programming for Noughts-and-Crosses: New Frontiers". Its main purpose was to introduce an approach to agent-oriented modelling in which the interfaces that mediate the interaction of each agent are expressed as networks of dependencies between observables and to show how this might be applied to developing programs. By way of illustration, that paper explored how programs for playing noughts-and-crosses – and variants of noughts-and-crosses that were characterised as 'OXO-like games' – could be flexibly developed in this way. In keeping with the spirit of PPIG, the emphasis in the paper was on the way in which the networks of dependencies, as represented by sets of definitions (aka "definitive scripts"), could be interpreted in cognitive terms. A basic cognitive function in playing noughts-and-crosses would be 'recognising the grid geometry' for instance; a more sophisticated one would be evaluating the merits of playing in a specific grid square. The development of scripts to express the agency in a program to play an OXO-like game reflected the way in which such cognitive capabilities could be construed as layered: one capability being prerequisite for another (cf. the script extracts in Figure 3 below).

Appending the words 'New Frontiers' to the core title of the paper was in part intended to be facetious – surely the idea that programming a simple game could be so challenging as to trespass on the boundaries of programming practice was absurd, and, more pretentiously, prescient – reflecting the idea that programming was not delivering to its full potential and in great need of an alternative vision. By comparison with conventional programming, agent-oriented modelling with definitive scripts clearly offered prospects for development more closely matched to human cognitive functions and with far greater openness and flexibility to change (cf. [6,5]). It was also an approach that had quite the opposite effect from conventional practices: rather than modelling the application domain by abstracting and simplifying, it engaged in the first instance and in its essence with the concrete experience of the modeller, and encouraged reflection that exposed the cognitive complexity that underlies the simplest tasks. The practical implications for giving digital support were intimidating: though each individual observable in a script might have an intuitive counterpart and hence be simple to interpret, specifying and managing the dependencies between observables and taking account of the open-ended range of perspectives on agency generated unprecedented problems. For instance, it was hard to imagine how to express the observation involved in surveying the grid to identify a winning

move, or developing a program that exploited a standard minimax strategy, on account of the number of observables involved and the subtlety of their interrelationship, conceptual, spatial and temporal.

This paper discusses how the principles proposed in [6] are currently being developed and supported in such a way that the prospects for more serious application to programming are much enhanced. The four main sections which follow relate to: the conceptual framework that has now been adopted – that of 'making construals' [3,2]; the **with**-construct that has recently been introduced into the environment for making construals (the 'MCE'); the significance of the **with**-construct from a programming perspective; and the prospects and challenges for future development and application.

2. Making construals

The emphasis in thinking about computing, and the central place of *programming* within it, owes much to Turing's seminal account of 'a mind following rules' [13,16,4]. In placing the emphasis on 'computational thinking', there is a danger of neglecting an equally significant preoccupation for the human mind: that of 'making sense of a situation'. The premise here is that an excellent working understanding of a situation is a prerequisite for being able to act by following rules. Informally, it is this sense-making activity to which the term 'making construals' used throughout this paper refers.

In this paper, the term "construal" is used in a distinctive way to refer to an interactive physical artefact that embodies our 'working understanding' of situation in terms of three primary interrelated notions: agency, observation and dependency: a concept that is the central theme of many previous publications from the *Empirical Modelling* project [21]. In any situation, *agency* relates to what we believe to be responsible for changes we observe, *observation* to the specific entities to which we believe agents respond and through which their state-changing action is mediated, and *dependency* to the way in which changes to observables are perceived by agents to be concomitant. There is a distinguished role in this sense-making activity for the personal agency, observation and dependency that is perceived by the maker of the construal, but the understanding of the situation encompasses putative agency, observation and dependency that is projected onto other agents of which the maker can have no direct experience. Understanding the way in which agency, observation and dependency interrelate establishes a template that is characteristic of a situation. In making a construal of a situation using contemporary technologies in which computing is an essential ingredient, the aspiration is to develop an interactive artefact that embodies counterparts of the agency, observation and dependency that interrelate according to this same template. In a computational environment, the situations that arise must be crafted so that agency, observation and dependency exhibit highly stable patterns of interrelationship that can be reliably reproduced.

In [9], the psychologist and philosopher of science David Gooding introduced the notion of a construal to account for the way in which Michael Faraday achieved the 'working understanding' of electromagnetism that led him to prototype the first electric motor. This is a precedent for applying the term 'construal' to hybrid constructions that have both a physical aspect and an associated cloud of interactions and interpretations. The spreadsheet is another example of such a blend of physical and mental ingredients that serves a sense-making purpose in which the concepts of agency, observation and dependency are more explicitly represented. Previous work shows broad potential application for making construals with this distinctive emphasis on agency, observation and dependency [21], but this has been constrained by the technical limitations of the approach described in [6].

The most familiar context in which the term 'construal' arises is in natural language understanding. An expression such as 'the polar bear is chilling' can *be construed* in several ways. We may imagine a polar bear lying in the shade of an iceberg on a sunny day, or a cartoon polar bear with sun glasses and a pint of beer lolling on a beach, or a hostile bear standing on its haunches towering above us ready to pounce with 'arms' and claws outstretched. Observables, dependencies and agents are clearly relevant to understanding these scenarios, though these are of a more complex and subtle nature than are illustrated in Figure 3. Whether or not the polar bear is seen as 'chilling' is an *observable*. Its status is determined by *dependency* on more prosaic observables: Is the sun shining? Is the bear in the shade? Are the bear's limbs configured in a benign way? Different construals reflect different kinds of *agency*: Is it the bear or the observer who experiences the chilling? At a meta-level, other varieties of observation, dependency and agency may be significant. A teacher might use this expression as a way

of assessing natural language understanding: the capacity of the student to observe different meanings being dependent on their level of familiarity with English.

Construal is an important ingredient in problem-solving. Creating artificial contexts for solving puzzles is a good way of exposing characteristic features of construal. In a cryptic crossword puzzle, each clue is a phrase that at face value has a natural language meaning, but – with some ingenuity – can also be construed as specifying a word to be entered into the grid. There are then standard conventions to guide the solver. The solution is always given in a non-cryptic form in the initial or final segment of the clue. Words such as ‘leading’, ‘embrace’ or ‘scrambled’ that have literal meanings may in fact refer to the disposition of letters in the solution. By way of illustration, consider the phrase ‘Force a writer to absorb Republican material’, which is the clue to a 6 letter word in a Guardian newspaper crossword [17]. A plausible solution is ‘police’: **a writer** (construed as Edgar Allan ‘Poe’) **absorbs** (construed as ‘contains’) **Republican material** (construed as the letters ‘lic’ extracted from the word Republican), the first word in the clue – **force** – being construed as a non-cryptic reference to the police force.

The nature of the activity involved in arriving at this solution illustrates several distinctive qualities of making construals. At its core, making construals is a subjective activity that relies upon associations that can be experienced by the maker on account of their general knowledge and cultural background (the fact that Poe is a writer, and that the police can be described as a force are well-known in the UK and USA). These associations can be loose and informal (how many different combinations of letters might qualify as ‘Republican material’?) and cut across semantic categories (Poe is perceived both as an author and as the string of letters ‘poe’). The plausibility of a construal depends in general on conventions for interpretation (how well has the crossword solver – or indeed the crossword setter – observed the accepted code of practice). The justification and evaluation of a construal is not amenable to abstract logic – it relies upon a process of exposition in which proposed connections have to be presented in conjunction with informal explanations and critically examined (consider what steps lead us to conclude that the answer to the above clue is ‘a force’ rather than ‘a material’). To accept the validity of a construal is to make a pragmatic judgement that takes account of all manner of observations about connections being proposed (for instance, Poe is a writer [12] – though not quite as celebrated as Poe, ‘ubl’ is a substring of “Republican”, and there is an online reference under a Guardian headline from December 2014 citing “a rise in the rouble ... as *forcing* people to live within their means” [19] – but *rouble* would hardly be deemed to be a convincing solution). In general, the status of construal as right or wrong, good or bad, appropriate or inappropriate, is likewise to be construed as context-dependent (for example, though ‘police’ is in some sense a good and appropriate proposed solution, it is actually *incorrect*, as reference to [18] (‘spoiler alert’) will show. That is to say, it is not consistent with the answers to other crossword clues). In practice, the way in which the maker is guided towards a construal can be very personal matter that may depend on the process of construction itself (‘police’ is an alternative solution that would not have been devised had other clues been solved first). This interplay between the construction process and the personal perspective of the maker of a construal is even more vividly highlighted by the famous NYT crossword published prior to the 1996 US election [20], where the answer to one clue could be construed to be either of the presidential candidates “Clinton” and “Bob Dole”. This evoked amazement or indignation from solvers that reflected their temperament, political affiliation and the solution they found.

In building on [6], the aspiration is develop principles and instruments for making construals that can do justice to construing in its broadest sense. Though the central focus of the paper is on a technical construct that can create a more expressive instrument, the concern is much broader than issues of programming language design.

3. Introducing the with-construct

Making construals is the central theme of the EU Erasmus+ CONSTRUIT! project [22]. A core aim of CONSTRUIT! is to create an online environment for making construals (the ‘MCE’). The latest version of the MCE [23], whose principal architect is Nicolas Pope (cf. Figure 1), incorporates several new features that promise to transform the practices that were first prototyped in [6]. One feature in particular has most significant implications for specifying families of definitions that correspond

closely to the more complex modes of observation encountered in making non-trivial construals. This is a "**with-construct**" that makes it possible to replicate patterns of dependency without needing to explicitly clone observables and create bloated scripts with perhaps many thousands of definitions.

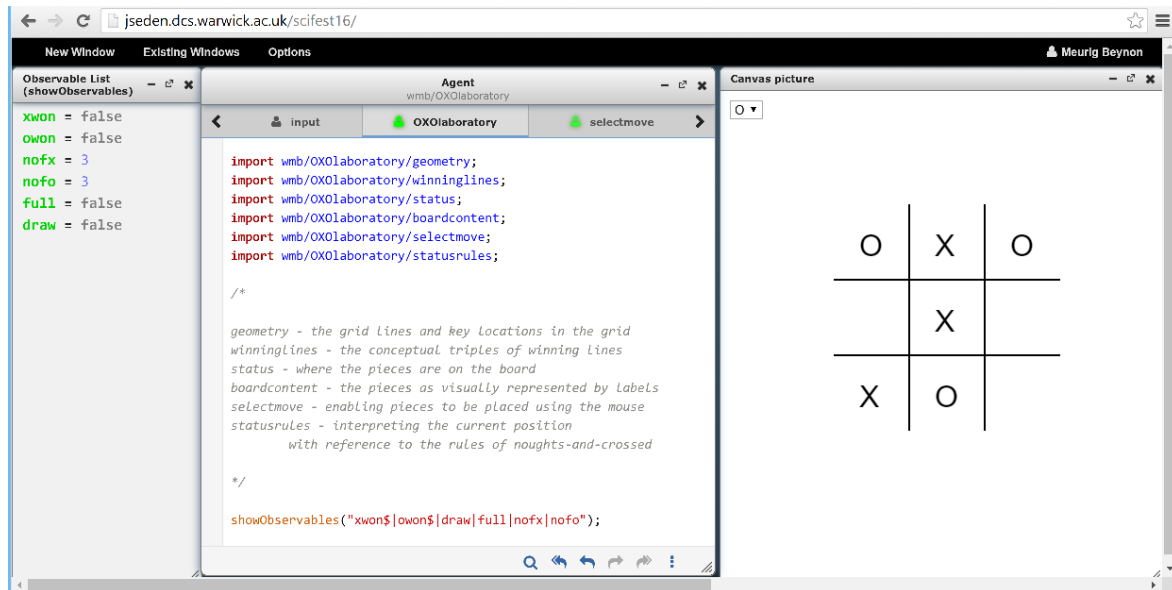


Figure 1: Reconstructing the OXO laboratory in the environment for making construals (the MCE)

The basic principle behind the with-construct is that where an observable A is defined by an (acyclic) network of dependencies in which other observables B and C occur, it is possible to introduce a variant of the observable A that makes use of the same dependency network that specifies A but incorporates new definitions for the observables B and C. Syntactically, this is expressed:

newA is A **with** B is newB, C is newC

The **with-construct** has a wide range of potential applications that both have the effect of making scripts much more concise and making it possible to express more complex forms of observation. A key element in this is being able to generate a set of instances in the form of a list defined via

list_of_newA is A **with** B ranging over list_of_newB, C ranging over list_of_newC

The nature of the with-construct and its full implications for making construals are yet to be explored. The illustrative examples represent some of the possibilities to be further discussed and developed.

Where applications are concerned, several broad categories of use have been identified to date. Each of these modes of use of '**with**' is illustrated in scripts that can be found in the project repository within the MCE [23,24]. A recent version of the MCE can be invoked at the url cited below [23]. A basic introductory tutorial for the MCE together with instructions for loading example construals can be found via a webpage prepared for use by teachers [24]. Example uses of '**with**' include:

- **prototyping in what resembles an object-oriented style:** This is the most orthodox application of the **with-construct**, to derive variants from a prototype: "X is a Y with a Z like this ...". To explore this, invoke the MCE and load the script at tutorials/scoping/arrow as described above.

- **generation of loci that reflect sophisticated forms of observation:** For instance (see Figure 2a), if the observable 'mercury' is a circle that represents Mercury in a simple planetary model, and the position of 'mercury' depends on the time-based observable 'tick' then

smallMercury is mercury **with** radius is smallRadius
 traceMercury is smallMercury **with** tick ranging over (tick - memory) up to tick

Parametrically defined geometric figures can also be defined in this way. For instance (cf. Figure 2b):

the ellipse {(a*cos(theta), b*sin(theta)) | 0<=theta<=360} and
 the sine curve {(theta, sin(theta)) | 0 <= theta <= n*360}

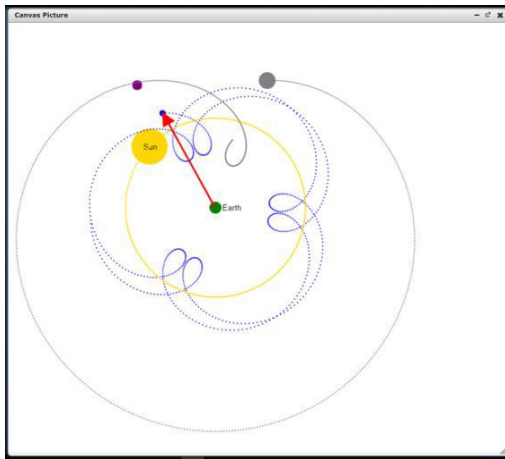


Figure 2a: A planetary orbit

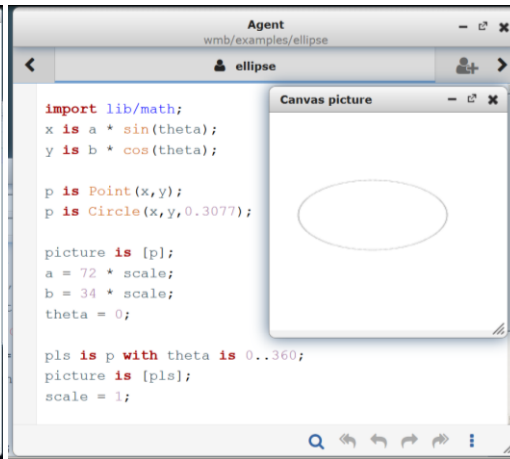


Figure 2b: A simple geometric locus

- **creating generic configurations of mathematical interest:** A good application for making construals has been creating artefacts that can help to expose abstract mathematical concepts and results. For instance, the Marriage Theorem is a basic theorem in matching theory which gives a simple criterion for a bipartite graph to have a perfect matching. In the past, it has been relatively straightforward to make a construal that captures the relevant observables for a specific small bipartite graph (say m by n , where $m=n=4$), but exceedingly tiresome to handle larger examples, and impossible to generalise for larger m and n without introducing complex macros. The **with**-construct makes it possible to construct generic observables to represent bipartite graphs and matchings in ways that are cognitively more congenial and better capture the spirit of mathematical abstraction.

- **expressing modes of observation that involve surveying many states:** as when examining several game states (cf. the listing below), or generalising geometric configurations featuring in games, such as the OXO-like games discussed in [6], which include 'cubic', a 3D version of noughts-and-crosses.

```

end_of_game is owon || xwon || draw; ## the criterion for the game to be over
lin_w is lin[1]+lin[2]+lin[3] == -2; ## whether the line lin has two Os on it
winlineix is lin_w with lin is alllines[ix];
winlines is winlineix with ix is 1..8; ## which of the winning lines has two Os, if any
gapinlin is 1 if lin[1]==0
      else (2 if lin[2]==0
      else (3 if lin[3]==0 else 0));      ## where there is no entry on line lin
playonlinix is gapinlin with lin is alllines[ix];
playonlines is playonlinix with ix is 1..8; ## where you can play on a winning line
alllinesindices is alllines with
      s1 is 1, s2 is 2, s3 is 3,
      s4 is 4, s5 is 5, s6 is 6,
      s7 is 7, s8 is 8, s9 is 9;
winindex is _index if winlines[_index] else 0;
iswinindex is winindex with _index is 1..8; ## indices of lines with two Os, if any
wline is max(iswinindex);
when ((player==o) && wline>0 && !end_of_game) {
      boardstate[alllinesindices[wline][playonlines[wline]]] = o;
}
## if player O is to move and there is a line with two Os and the game is not over
## make a winning move by placing an O in a gap on such a line

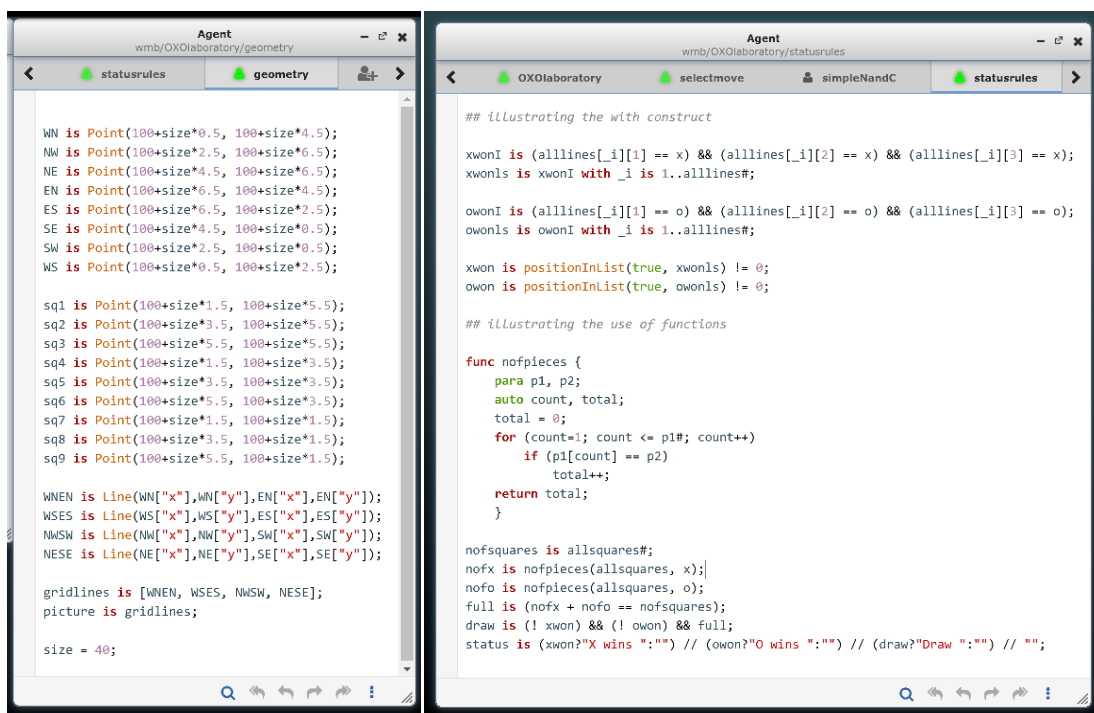
```

Listing 1: An experimental script to reflect human observation when seeking a winning move

4. The with-construct from a programming perspective

What is the significance of **with** from a programming perspective? In [8], where the relationship between ‘making construals’ and proposals for new approaches to programming by Bret Victor [14] and Chris Granger [10] is discussed, much emphasis is placed on the distinction making construals and programming. This seems to be appropriate, not least because the program-like behaviours that are realised through what was characterised in [6] as ‘agent-oriented modelling with definitive representations of state’ have clear limitations. Framing behaviours in terms of interactions with definitive scripts is ill-suited to programming dynamical systems, does not address the challenges of achieving computational efficiency, and (the **with**-construct notwithstanding) does not suit large-scale computational applications where anonymous data variables cannot be avoided. But where definitive scripts were recognised in [6] as a powerful way of making the interface that mediates an agent’s *action* intelligible, the **with**-construct may be seen as serving a complementary function: enabling the context for an agent’s *observation* to be conveniently transformed. In this way, it becomes possible to make more direct links between programming and observation than before.

Listing 1 above is an experimental script that illustrates some of the significant features of introducing the **with**-construct. It is drawn from [2], where it can be studied in context. This script exploits many instances of **with** to describe the observation required by player O when making a winning move in noughts-and-crosses. The use of **with**’s may be compared and contrasted with the directly translated extracts from the original script for the OXO grid (cf.the ‘outline of the OXO-model’ presented in ‘Listing 1’ as an Appendix to [6]) that are shown in the left hand panel and in the bottom half of the right hand panel of Figure 3 below. Though these extracts illustrate a mode of observation much simpler than that expressed in the above listing, it requires definitions that exploit maker-defined functions (such as nofpieces()) formulated in a conventional procedural programming style. This tends to subvert the notion that making construals through modelling with definitive scripts is conceptually simpler than traditional programming. As the comments in the Listing 1 show, each of the **with**-definitions admits a conceptually simple interpretation in observational terms. In a similar manner, the definition of the observable xwon in the top half of the right hand panel of Figure 3 expresses the notion of surveying each of the possible winning lines (‘alllines’) in a fashion that is faithful to human observation and is in sharp contrast to the clumsy procedurally expressed function that is used to check the status of each of the potential winning lines in the original OXO-model.



```
WN is Point(100+size*0.5, 100+size*4.5);
NW is Point(100+size*2.5, 100+size*6.5);
NE is Point(100+size*4.5, 100+size*6.5);
EN is Point(100+size*6.5, 100+size*4.5);
ES is Point(100+size*6.5, 100+size*2.5);
SE is Point(100+size*4.5, 100+size*0.5);
SW is Point(100+size*2.5, 100+size*0.5);
WS is Point(100+size*0.5, 100+size*2.5);

sq1 is Point(100+size*1.5, 100+size*5.5);
sq2 is Point(100+size*3.5, 100+size*5.5);
sq3 is Point(100+size*5.5, 100+size*5.5);
sq4 is Point(100+size*1.5, 100+size*3.5);
sq5 is Point(100+size*3.5, 100+size*3.5);
sq6 is Point(100+size*5.5, 100+size*3.5);
sq7 is Point(100+size*1.5, 100+size*1.5);
sq8 is Point(100+size*3.5, 100+size*1.5);
sq9 is Point(100+size*5.5, 100+size*1.5);

WNEN is Line(WN["x"],WN["y"],EN["x"],EN["y"]);
WSES is Line(WS["x"],WS["y"],ES["x"],ES["y"]);
NWSW is Line(NW["x"],NW["y"],SW["x"],SW["y"]);
NESE is Line(NE["x"],NE["y"],SE["x"],SE["y"]);

gridlines is [WNEN, WSES, NWSW, NESE];
picture is gridlines;

size = 40;
```

```
## illustrating the with construct
xwonI is (alllines[_i][1] == x) && (alllines[_i][2] == x) && (alllines[_i][3] == x);
xwonIs is xwonI with _i is 1..alllines#;

owonI is (alllines[_i][1] == o) && (alllines[_i][2] == o) && (alllines[_i][3] == o);
owonIs is owonI with _i is 1..alllines#;

xwon is positionInList(true, xwonIs) != 0;
owon is positionInList(true, owonIs) != 0;

## illustrating the use of functions

func nofpieces {
  para p1, p2;
  auto count, total;
  total = 0;
  for (count=1; count <= p1#; count++)
    if (p1[count] == p2)
      total++;
  return total;
}

nofsquares is allsquares#;
nofx is nofpieces(allsquares, x);
nofo is nofpieces(allsquares, o);
full is (nofx + nofo == nofsquares);
draw is (! xwon) && (! owon) && full;
status is (xwon?"X wins ":"") // (owon?"O wins ":"") // (draw?"Draw ":"") // "";
```

Figure 3: Representing the OXO grid without and game status with and without **with**’s

The value of linking programming with observation is being recognised in teaching programming. In his account of work with kindergarten children, Ivan Kalas [11] alludes to Blackwell's characterisation of programming (cf. [7]), whereby "the user is not directly manipulating observable things, but specifying behaviour to occur at some future time". He then highlights the importance of scaffolding the transition from specifying action through direct manipulation of observables to full automation.

Similar principles are illustrated in the work of Dave White [15] on teaching programming in an 'unplugged' fashion by encouraging learners to conceive systematic ways of carrying out tasks such as 'drawing a grid'. White has developed some ingeniously crafted environments that support the transition from walk-through to automatic execution of such systematic procedures. His work highlights the critical role played by the mode of observation of a situation in specifying programmed activity within it. Figures 4a and 4b below show how the different modes of observation of a grid of this nature can be expressed in the MCE using the **with**-construct.

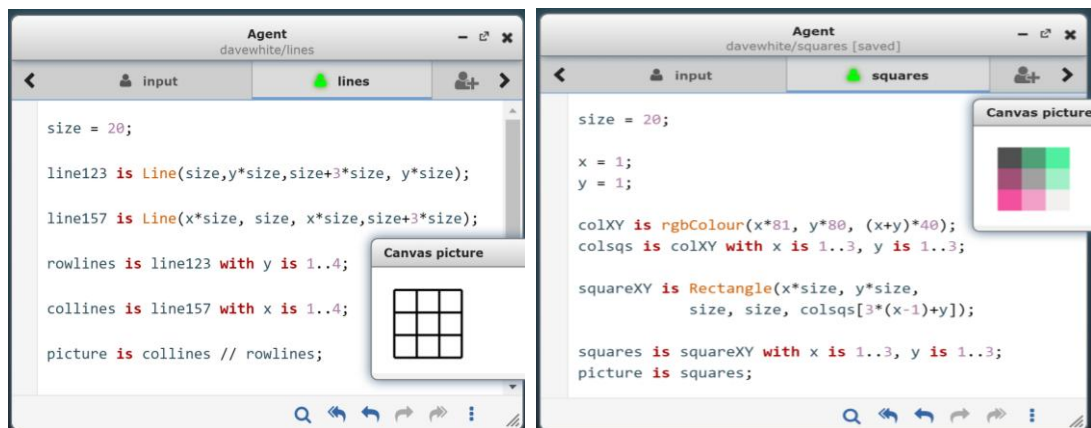


Figure 4: Conceiving a 3 by 3 grid (a) in terms of lines and (b) in terms of squares

A construal of 'giving change' inspired by Phil Bagge's Coins example in [1] shows how this notion can be generalised to transform a strategy for teaching a learner to observe coins in such a way that they are eventually able to give change into a 'program' for giving change that is flexible enough e.g. to cope with missing denominations and can deal with different currencies. This example also exhibits the use of recursively scoped **with**-definitions. A presentation, built into the MCE [23], that introduces this construal can be accessed online by loading the script givingchange/commentary2 from the project manager of the MCE.

5. More about the with-construct

The fundamental respect in which realising program-like behaviours through making construals differs from traditional programming is that it makes a clear separation between two aspects of the computer's role:

- the computer performs *dependency maintenance*, ensuring that the relationship between observables is maintained so that it is closely matched to the observations of significant entities in the application domain. This dependency maintenance is a background process that provides the stage on which the state-changing actions of the agents within the application are played out.
- the computer automates the state changing actions that are characteristic of the application by redefining observables explicitly following a prescribed algorithmic pattern. The algorithmic specification is much simpler than that in a conventional program, since a single redefinition of an observable affects the values of all dependent observables without the need for explicit updates.

These two aspects correspond to what was characterised in [6] as 'definitive representations of state' and 'agent-oriented modelling' respectively. The powerful advantage of the above separation of concerns is that it makes it possible to integrate modes of agency that in traditional program are conceptually quite distinct. As explained in detail in [3], after making an appropriate construal of the application domain, implicit dependency maintenance sets the stage for redefinition of observables to

represent state-changing actions that might be interpreted, according to the situation, at the discretion of the maker, as ‘program design’, ‘implementation’ or ‘use’.

Prior to the introduction of the **with**-construct, the benefits of being able to blend design, implementation and use in this fashion were compromised by the fact that scripts of definitions were statically defined and could not vary in size dynamically as observables in the application domain varied. As a simple example, in making a construal of a simple game such as Conway's Life [25], where the growth of the population of cells cannot be predicted, there was no easy way to meet the need for an unbounded number of observables. This meant that, whilst the model of cell population was in some respects far richer than that supported by (say) a functional programming implementation of Life (for instance, modelling each game state explicitly and enabling the observer to intervene in the simulation to add or remove cells or to adapt the rules on-the-fly), it could not emulate the scope for unbounded extension afforded by lazy evaluation.

As far as the nature of the **with**-construct is concerned, there are clear parallels with prototype-based object-oriented development and with the replication of patterns of dependency that can be achieved (e.g.) through cutting-and-pasting rows in a spreadsheet. The **with**-construct can also be regarded as performing a *scoping* function somewhat resembling that available in JavaScript, in that it changes the context in which observables in a script are interpreted. Long experience of trying to integrate standard object-oriented notions into modelling with dependency networks suggests that the role that dependency plays in this context is semantically and logistically crucial. On this basis, the parallel with spreadsheets is probably the most significant. The extent to which this is corroborated by our practical examples will be an interesting topic of discussion. The PPIG audience may be familiar with other potential parallels to be considered.

A further aspect of the **with**-construct that may be of particular interest to a PPIG audience concerns its most effective representation. At present, whilst it is not too painful to trace the stream-of-thought that leads to the construction of a script that includes several **with**'s, it is not so easy to reconstruct this stream-of-thought, which is closely linked to a sequence of specific kinds of observation, by casual inspection of the script (cf. Listing 1 above!). This is in contrast to the directness of the connection between an observable in a construal and its counterpart in the referent.

A further consideration is that, in order to make most effective use of the potential for replication that **with** affords, it seems to be necessary to allow the component observables introduced in a **with**-clause to have independent definitions. This is in danger of undermining a highly significant feature of definitive scripts as so far conceived: the fact that the value associated with an observable or component of an observable can always be inferred from a *single* definition. This principle, which is in some sense a counterpart – in a semantic frame that has inherent a notion of ‘the present moment’ – of referential transparency in a declarative programming language, makes a major contribution to the intelligibility of scripts. If overriding of component definitions is permitted, it becomes essential to structure scripts so as to make it possible to retrieve the authoritative as-of-now definition of all observable components. For instance, when the script fragment below is interpreted sequentially:

```
a is 5;  
alist is a with ix is 1..10;  
alist[2] is 3*a;
```

the current value of the second component of the list `alist` is 15, whilst all other components are 5. Some syntactic convention is then needed to ensure that the overriding that occurs in this context can be easily detected when reviewing the script.

The need for independent definitions of the observables generated by a range scope can also be seen when observables such as buttons are generated in this way. It is essential to be able to distinguish between clicking on different buttons, and to be able to associate different actions with clicks accordingly. For instance, it is evident that in Figure 1 it is necessary to generate different actions according to which of the squares in the grid is selected on a mouse click. Figure 6 below (in which the syntax ‘::’ has been introduced as a synonym for **with**) illustrates work-in-progress on refinements of the MCE which help to address the concerns raised above.

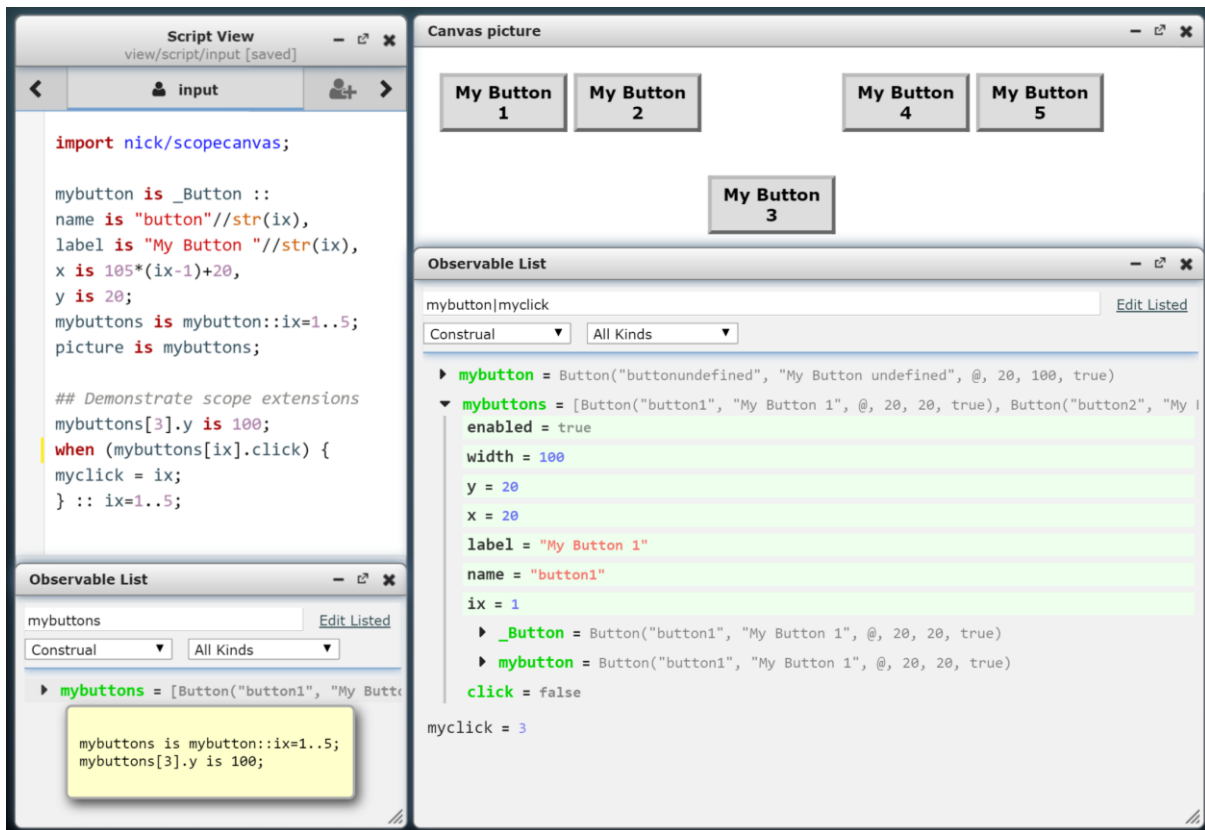


Figure 5: Extending the MCE to support overriding of range observables and scope browsing

6. Acknowledgements

Thanks are due to Dave White for introducing us to ISPY, to Jane Waite for helpful advice on the 'giving change' construal and to Mike Joy for acting as the Coordinator of the CONSTRUIT! project. We are also grateful to three anonymous reviewers for their helpful feedback. This project has been funded with support from the European Commission under the Erasmus+ programme (2014-1-UK01-KA200-001818). This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

7. References

1. Bagge, P. How to teach primary programming using Scratch. University of Buckingham Press, 2015, ISBN 981908684578
2. Beynon, M. et al. Games with observables, dependency and agency in a new environment of making construals. Proc. of the 9th International Conference on Interactive Technologies and Games (ITAG) 2016, 26-27 October 2016, Nottingham, UK (to appear).
3. Beynon, M. et al. Making construals as a new digital skill: dissolving the program - and the programmer - interface, Proceedings of the 8th International Conference on Interactive Technologies and Games (ITAG), 22-23 October 2015, Nottingham, UK, 9-16. See go.warwick.ac.uk/em/publications/papers/128
4. Beynon, M. Turing's approach to modelling states of mind. In (eds. S Barry Cooper and Jan van Leeuwen) Alan Turing - His Work and Impact, Elsevier, May 2013, 85-91. See go.warwick.ac.uk/em/publications/papers/115
5. Beynon, M. and Russ, S.B. Experimenting with Computing. Journal of Applied Logic 6 (2008), pp. 476-489. See go.warwick.ac.uk/em/publications/papers/098

6. Beynon, W.M. and Joy, M.S. Computer Programming for Noughts-and-Crosses: New Frontiers. Proc.of 6th Annual Psychology of Programming Interest Group Conference PPIG'94, Open University, 27-37, January 1994. *See* go.warwick.ac.uk/em/publications/papers/033
7. Blackwell, A. What is Programming?. In J. Kuljis, L. Baldwin & R. Scoble (Eds). Proc. PPIG 14, Brunel University, 2002, 204-218.
8. Boyatt, R., Beynon. W.M and Beynon. M. Ghosts of Programming Past, Present and Yet to Come. Proceedings of 25th Annual Psychology of Programming Interest Group Annual Conference 2014 (ed. Benedict du Boulay & Judith Good), Brighton, June 25-27th, 2014, 171-182. *See* go.warwick.ac.uk/em/publications/papers/125
9. Gooding, D. Experiment and the Making of Meaning: Human Agency in Scientific Observation and Experiment. Dordrecht: Kluwer. 1990.
10. Granger, C. Towards A Better Programming, March 27th 2014. Available from <http://www.chris-granger.com/2014/03/27/toward-a-better-programming/>
11. Kalas, I. On the Road to Sustainable Primary Programming. Constructionism 2016, Feb. 1-5, Bangkok, Thailand. <https://www.youtube.com/watch?v=55hlqh7g-xk> [Accessed: 22/5/2016]
12. Roe, C. Computers for Learning: An Empirical Modelling Perspective. PhD thesis, Department of Computer Science, University of Warwick, UK (November 2003).
13. Turing, A.M. On Computable Numbers, with an Application to the Entscheidungsproblem, Proc Lond Math Soc (2, 42), 1936, 230-265
14. Victor, B. Inventing on principle. Invited Talk at Canadian University Software Engineering Conference (CUSEC). 2012
15. White, D. ISPY and PUSH-PYTHON. <http://ispython.com/>
16. Stanford Encyclopedia of Philosophy <http://plato.stanford.edu/entries/turing/>
17. <https://www.theguardian.com/crosswords/prize/26741>
18. <https://www.theguardian.com/crosswords/2015/dec/04/annotated-solutions-for-prize-26741>
19. <https://www.theguardian.com/business/live/2014/dec/17/rouble-russia-crisis-greek-elections-uk-unemployment-fed-live>
20. <http://twentytwowords.com/the-crossword-puzzle-that-predicted-the-future-with-complete-confidence/>
21. The Empirical Modelling website: go.warwick.ac.uk/em
22. The CONSTRUIT! project website: www.construit.org
23. The environment for making construals. <http://jseden.dcs.warwick.ac.uk/scifest16/>
24. Getting Started with JS-Eden
<https://www2.warwick.ac.uk/fac/sci/dcs/research/em/construit/year2/c15/forteachers/>
25. Conway's Game of Life: https://en.wikipedia.org/wiki/Conway's_Game_of_Life