Program Synthesis

MPhil ACS module P230 - Alan Blackwell

You do the rest!

Principles of program synthesis, from HCI perspective

- The user experience of ML-based synthesis:
  - The user says: "Here is an example of what I want to do"
  - Followed by: "You do the rest"
- System response: "OK, I'll do others the same way"
  - How does it know what "others" are?
  - How does it know what "the same way" is?
- Usability issues:
  - How to specify applicability?
  - How to control generalisation?
  - How to understand what was inferred?
  - How to modify the synthesised program?

Classic programming by example

- Keyboard macros – demo in Emacs

- Get a plain text file containing semi-structured text
- `<Ctrl+x> (` starts macro recording
- Perhaps search for context, cut and paste, add text …
- Remember to go to known location (e.g. start of next line)
- `<Ctrl+x> )` ends recording
- `<Ctrl+x> e` plays back once
- `<ESC> 1 0 0 <Ctrl+x> e` repeats 100 time

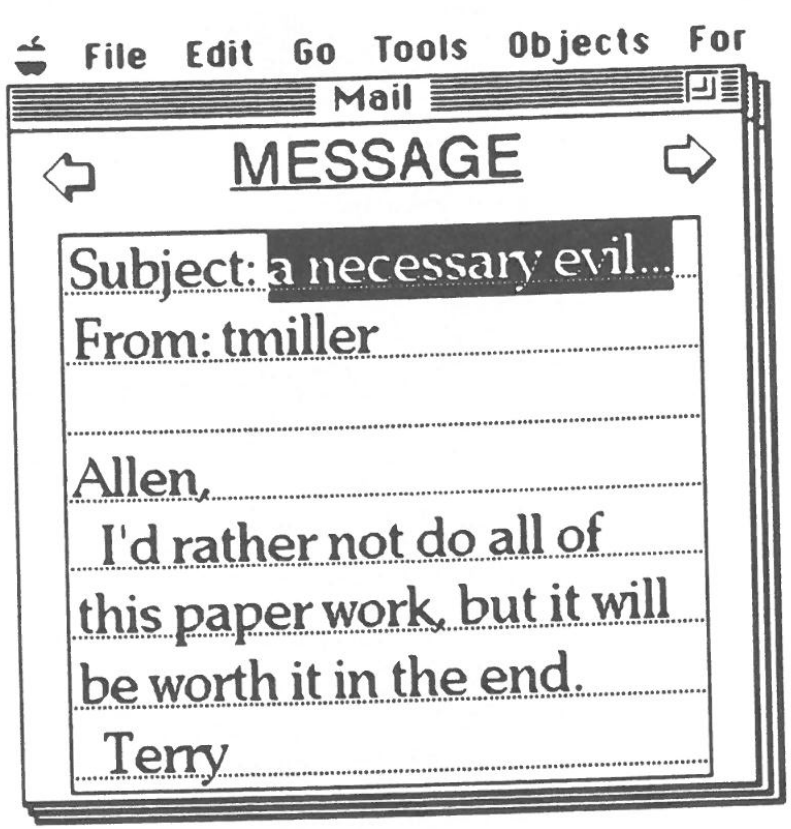
Value proposition

- The next generation of AI: "Intelligent tools"
- If a user knows how to perform a task on a computer, that should be sufficient to create a program to perform the task.
  - Early research aimed to achieve "programming in the user interface"
- Macro recorders are one model, but they are "too literal"
  - Do only what they are shown (no generalisation)
  - Unable to adjust for different cases (no inference)
- Other models:
  - Automation of repetitive activities
  - Creation of custom applications
- Machine learning problem is to create a model of user *intent*
  - Ideally informed by prior likelihood – from this user, and other users

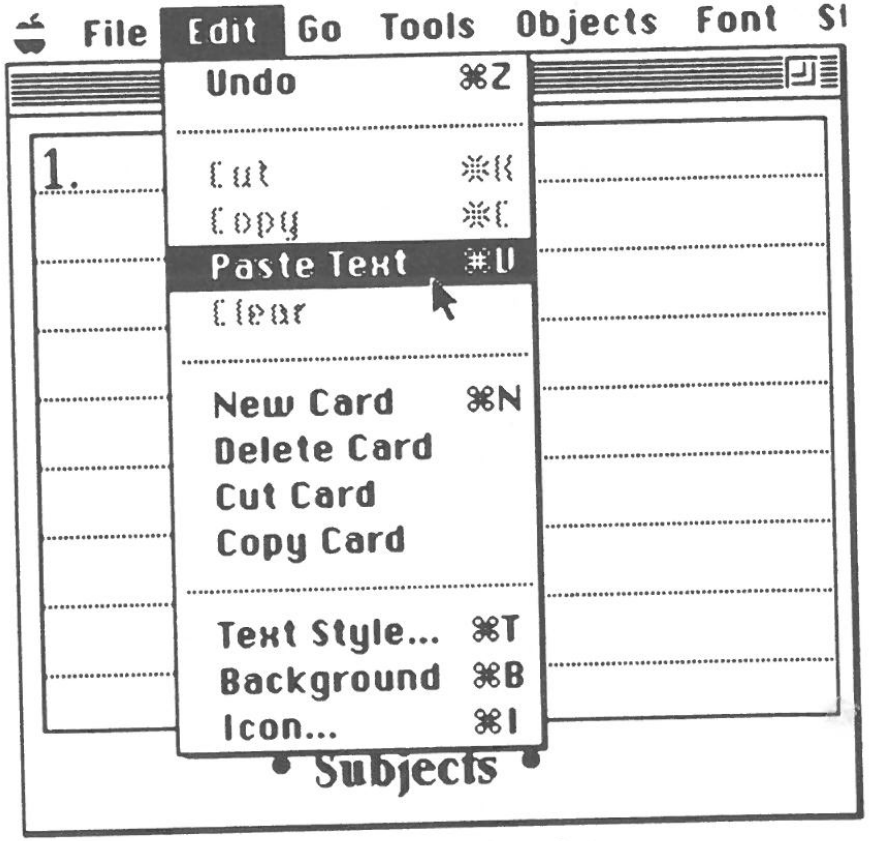
Eager

Classic mixed-initiative programming by example

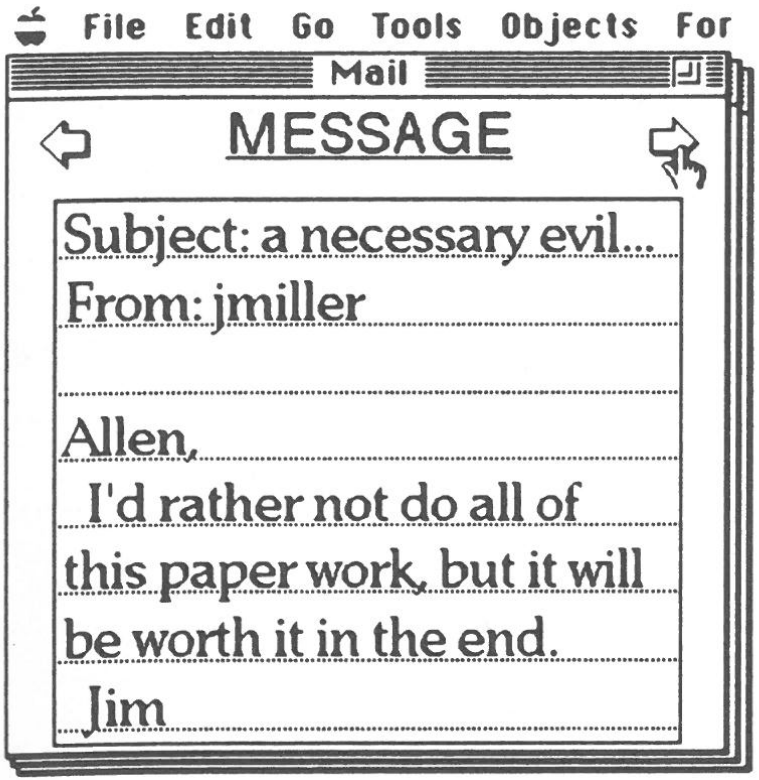
- Allen Cypher's "Eager" created at Apple research in 1990
  - Implemented as extension to Hypercard (event capture + injection)
  - Machine learning implemented in LISP
- Scenario – create a script to produce a list of subject lines from messages



a) copy first subject

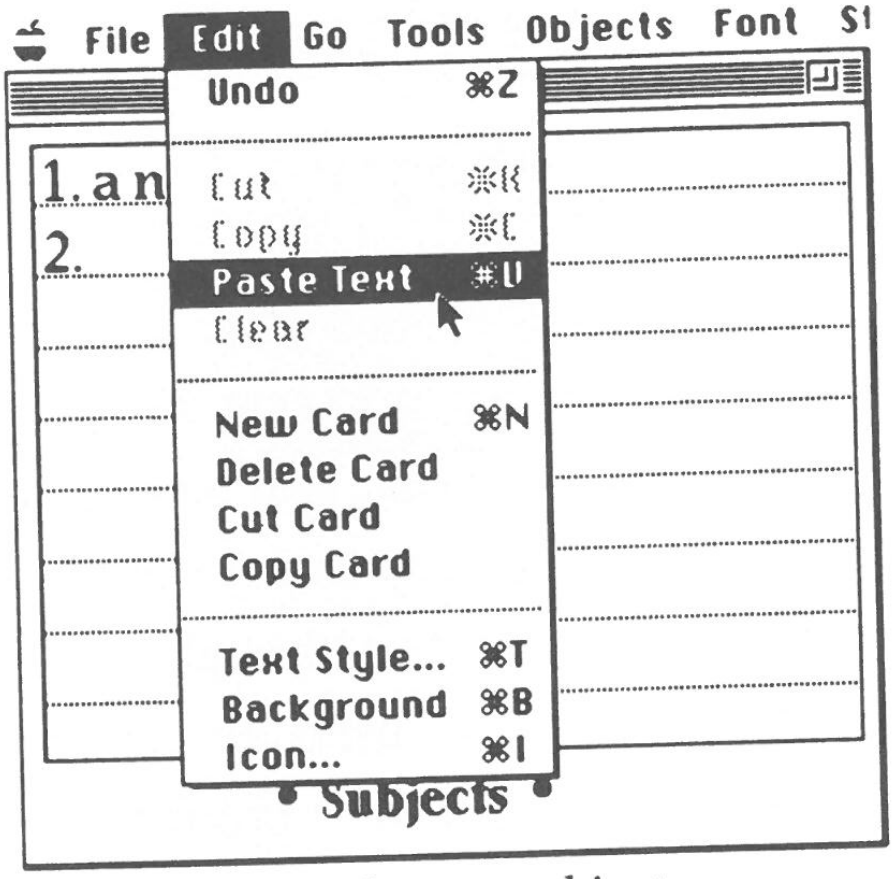


b) type "1. " and paste subject



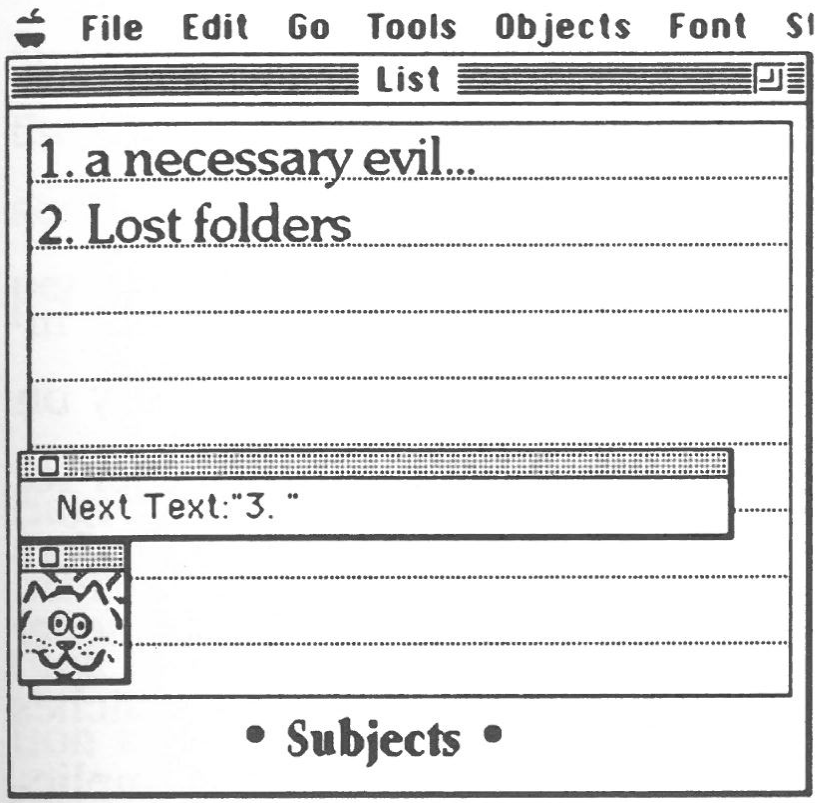
c) go to next message

d) copy second subject

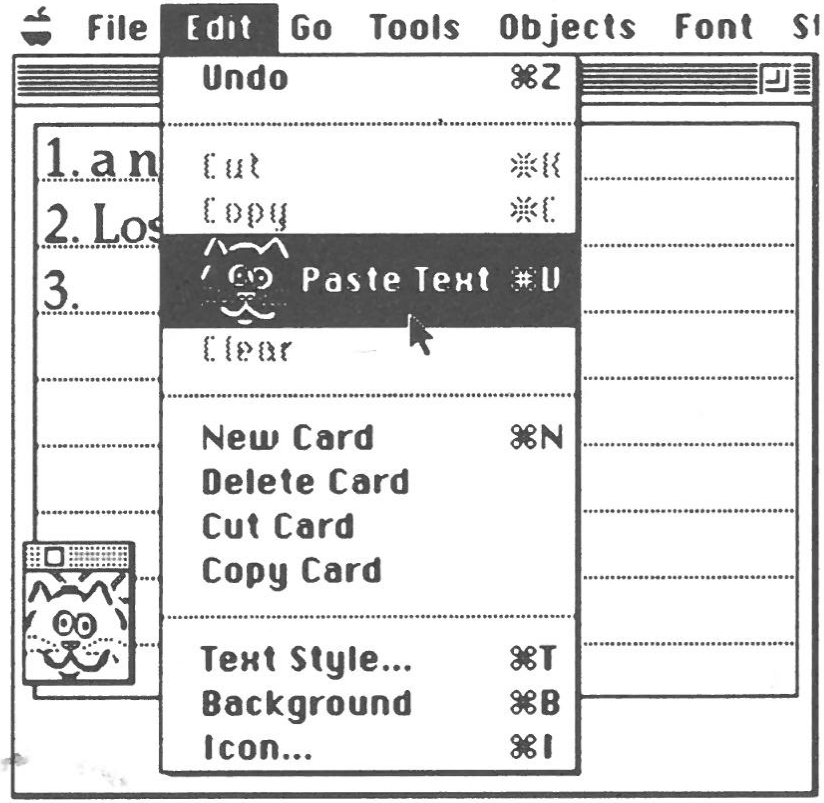


e) type "2. " and paste subject

f) Eager appears

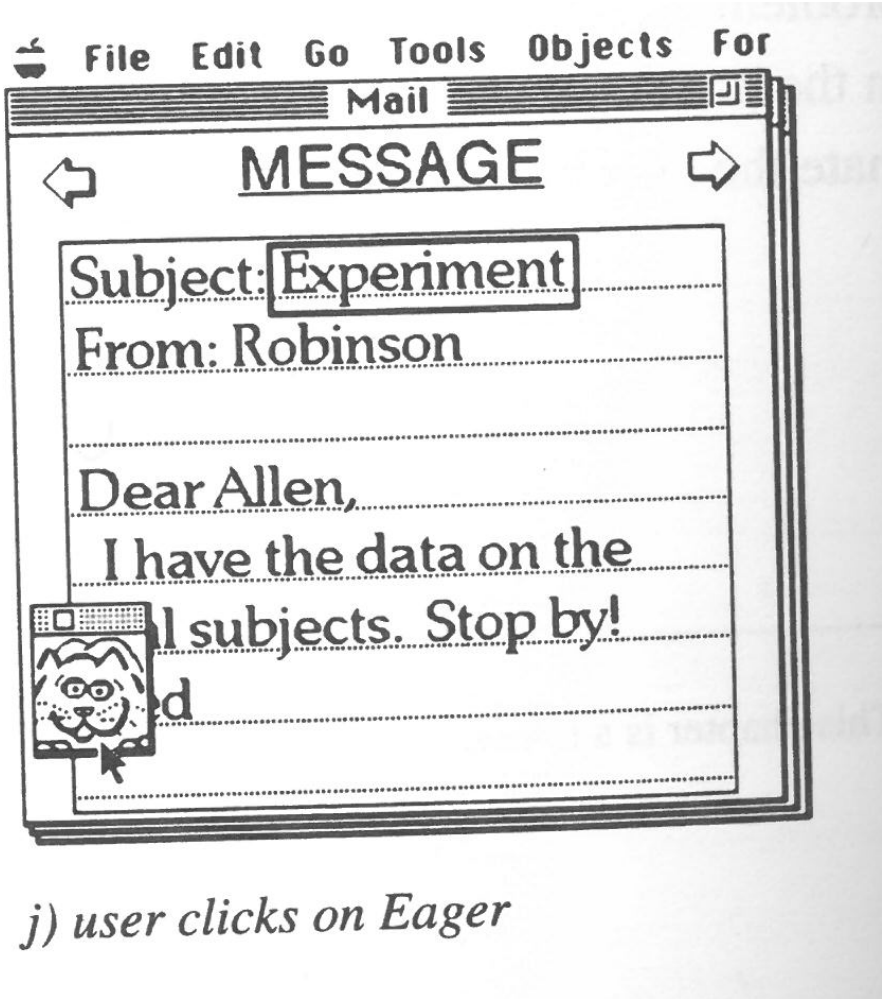


g) anticipate typing "3. "

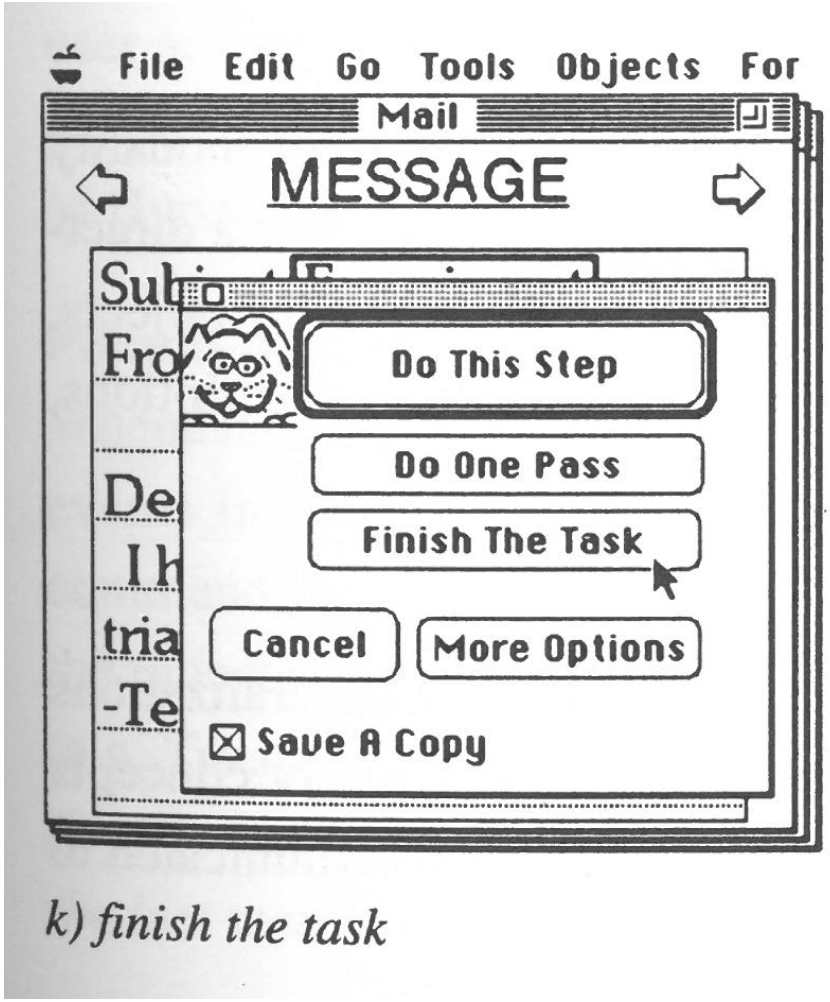


h) anticipate paste

i) anticipate going to next message



j) user clicks on Eager



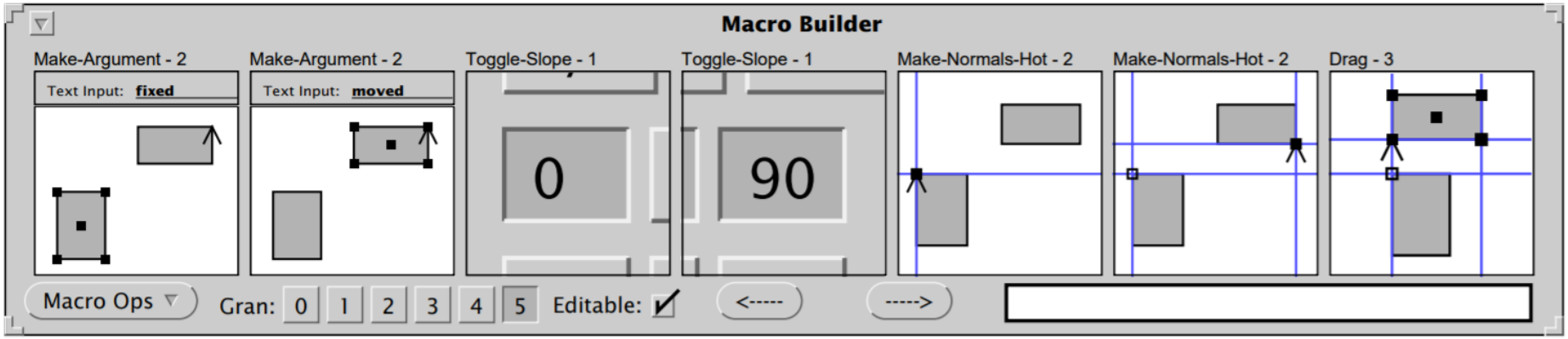
k) finish the task

l) Eager finishes

Chimera

Programming by demonstration in the graphics domain

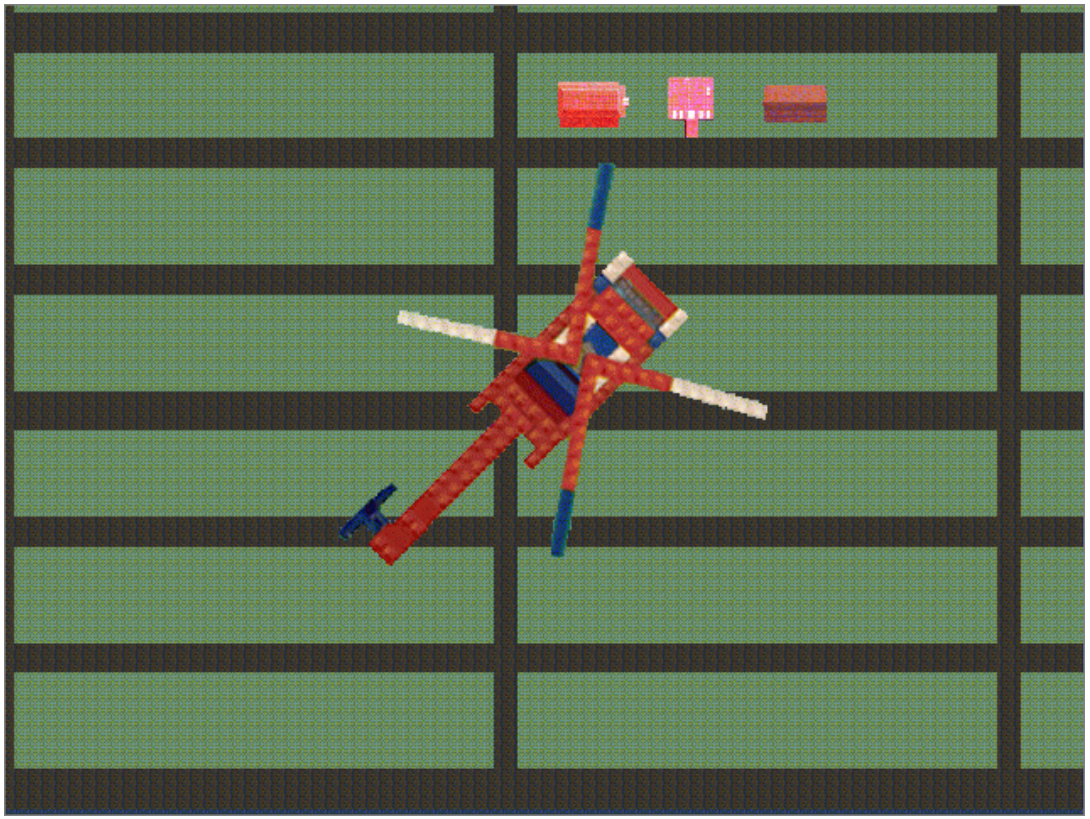
- Classic example: David Kurlander's Chimera
  - Infers constraints via heuristics, from snapshots of drawing editor state
  - Users can generalise a "graphical macro" in editable history of operations
  - https://youtu.be/JbrJQW25ekI?t=7m7s



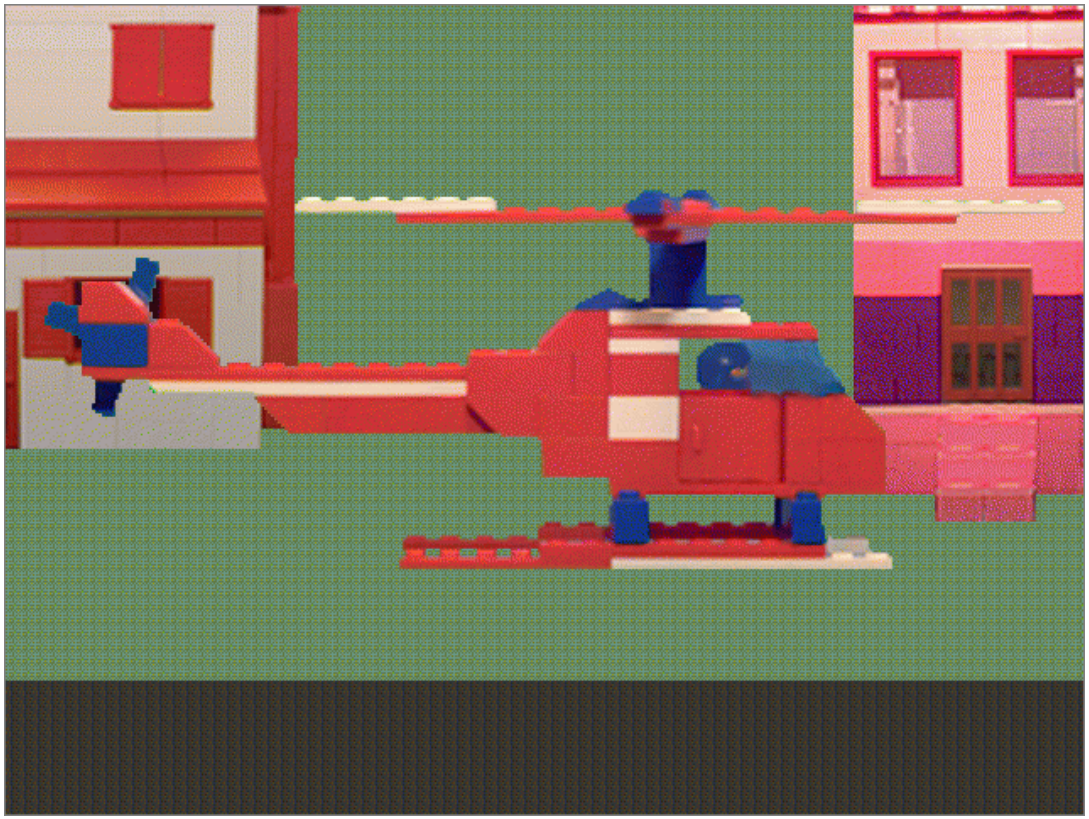
- D. Kurlander *Graphical Editing by Example* (1993)
  - PhD thesis, Columbia University. CS Tech/ Report CUCS-023-93

ToonTalk

Ken Kahn's ToonTalk – user control of generalisation

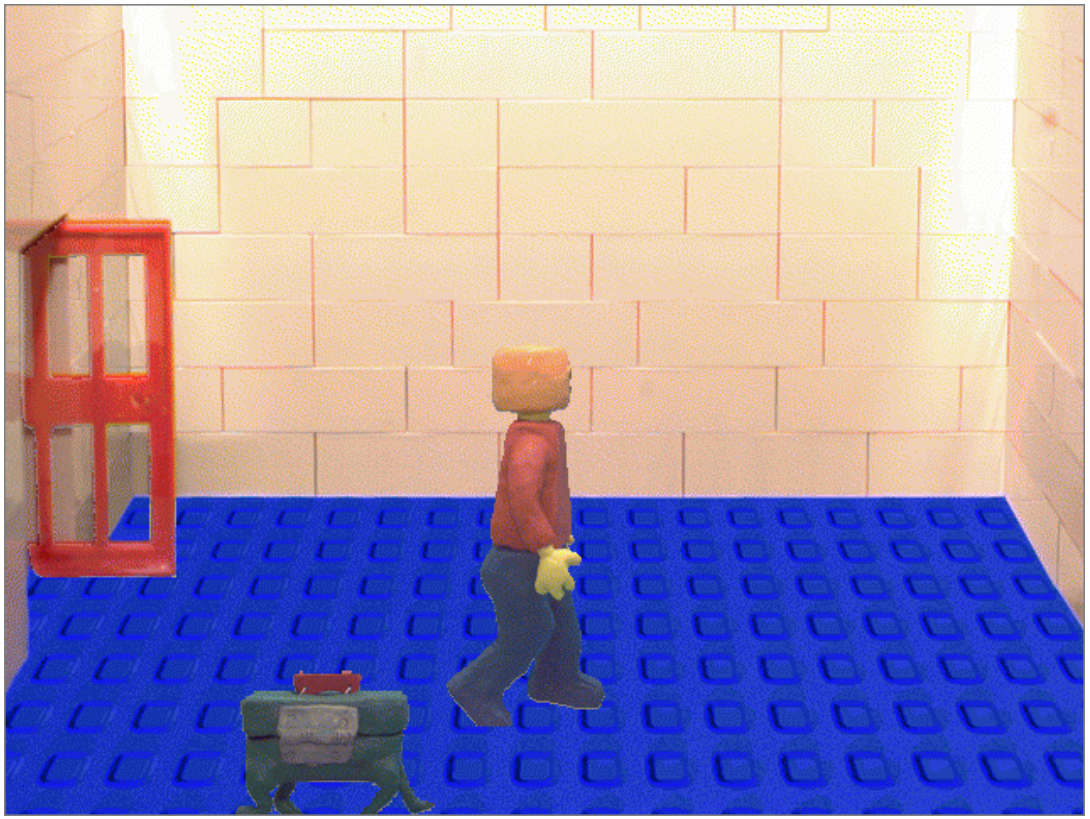


Ken Kahn's ToonTalk – user control of generalisation

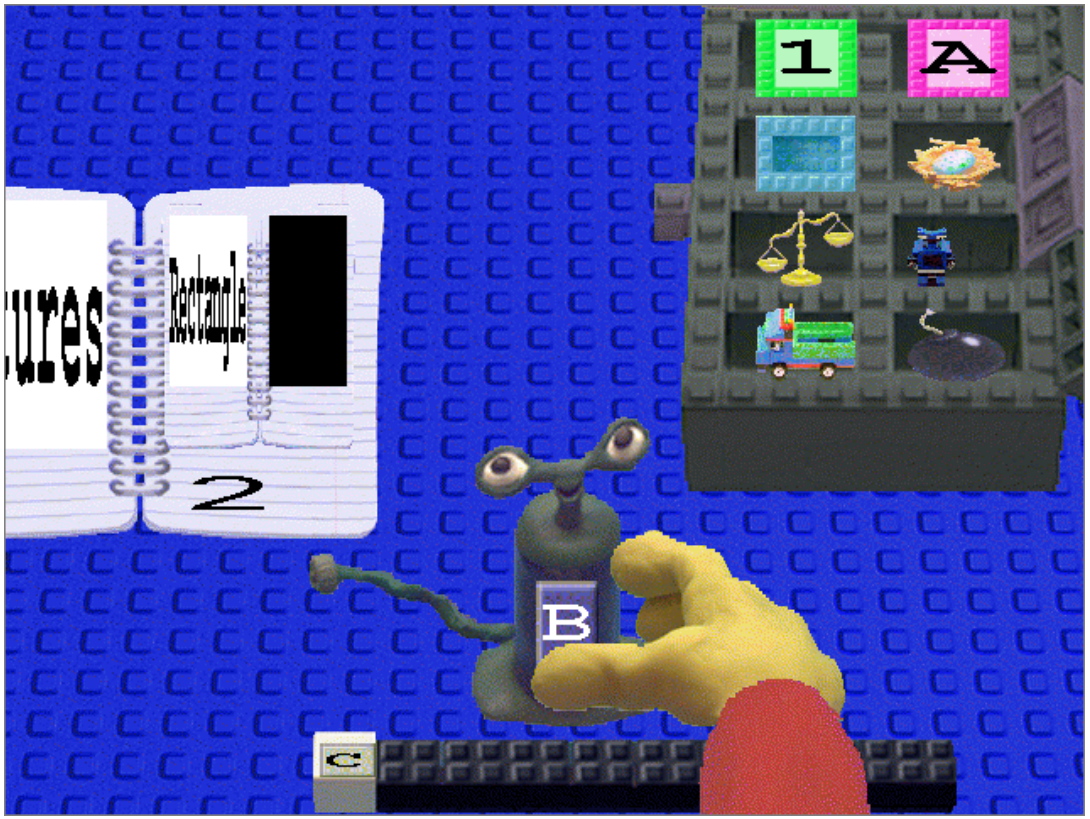


Ken Kahn's ToonTalk – user control of generalisation

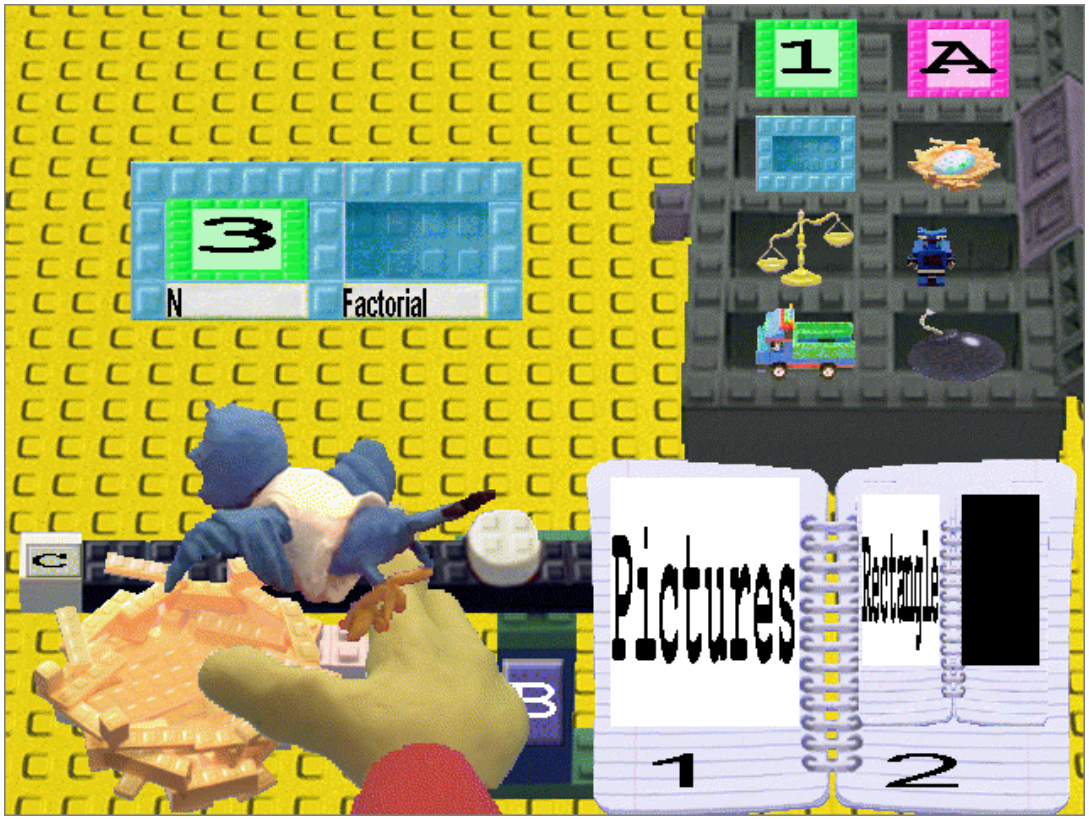
Ken Kahn's ToonTalk – user control of generalisation



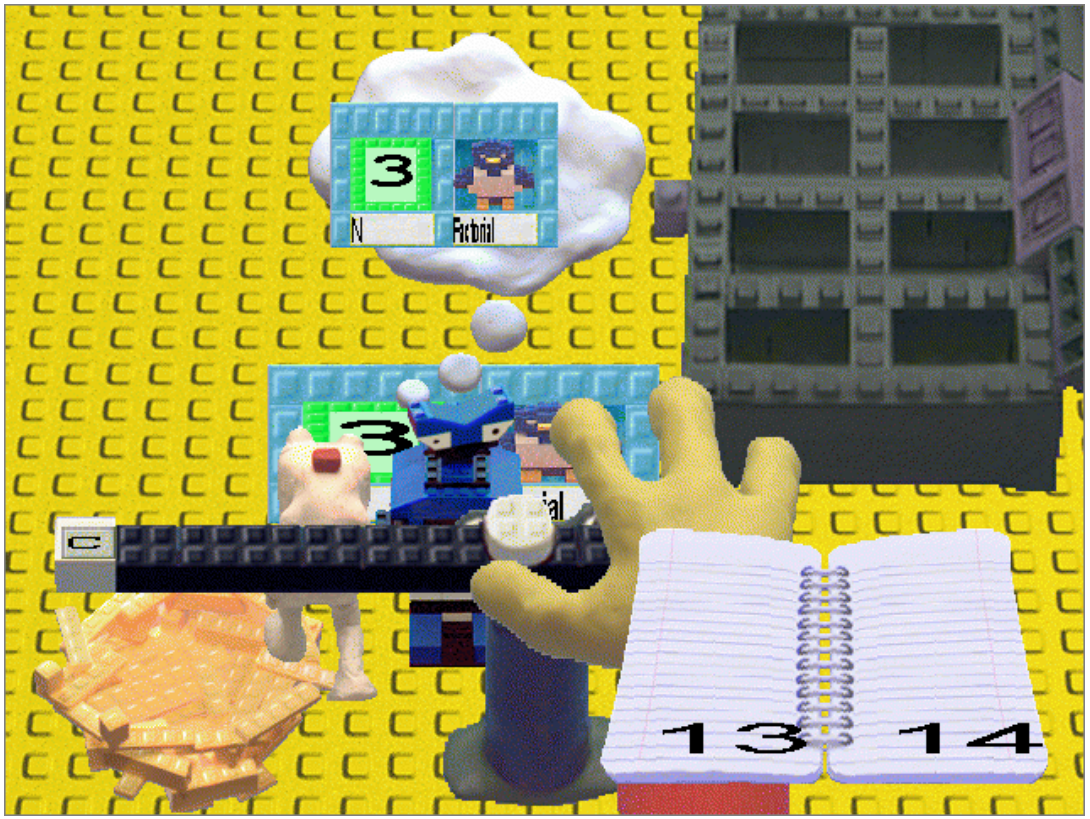
Ken Kahn's ToonTalk – user control of generalisation



Ken Kahn's ToonTalk – user control of generalisation

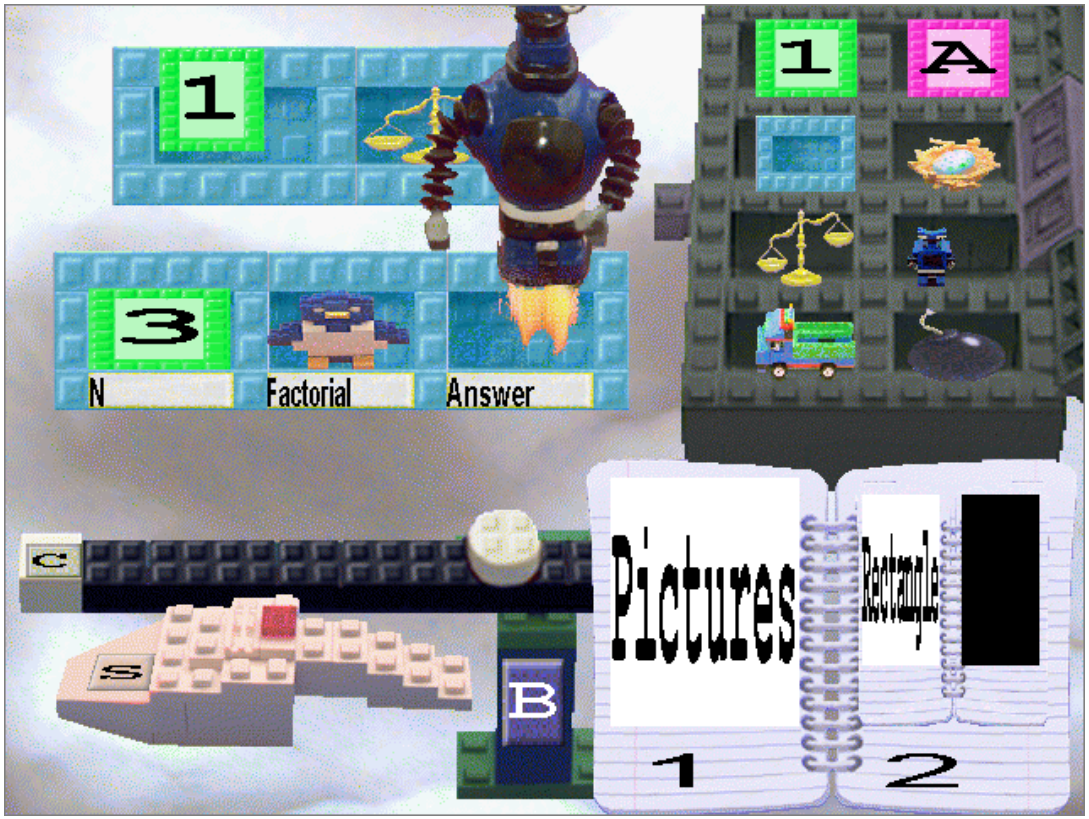


Ken Kahn's ToonTalk – user control of generalisation

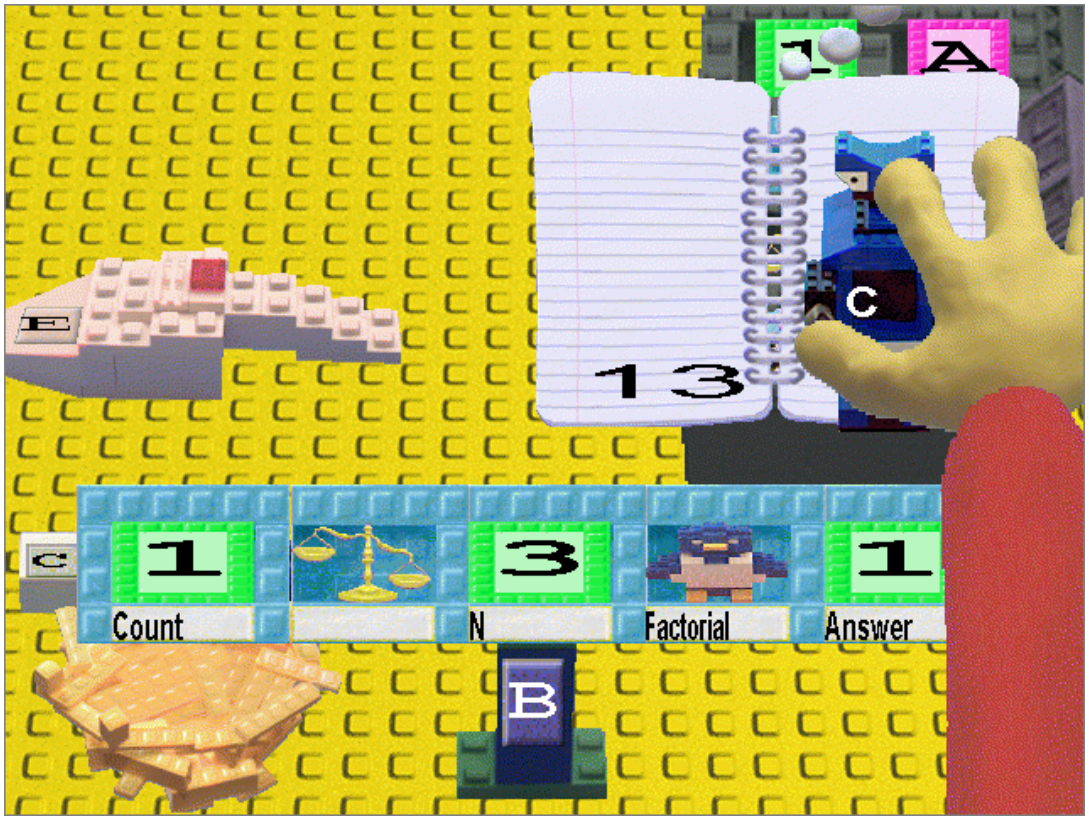


Ken Kahn's ToonTalk – user control of generalisation

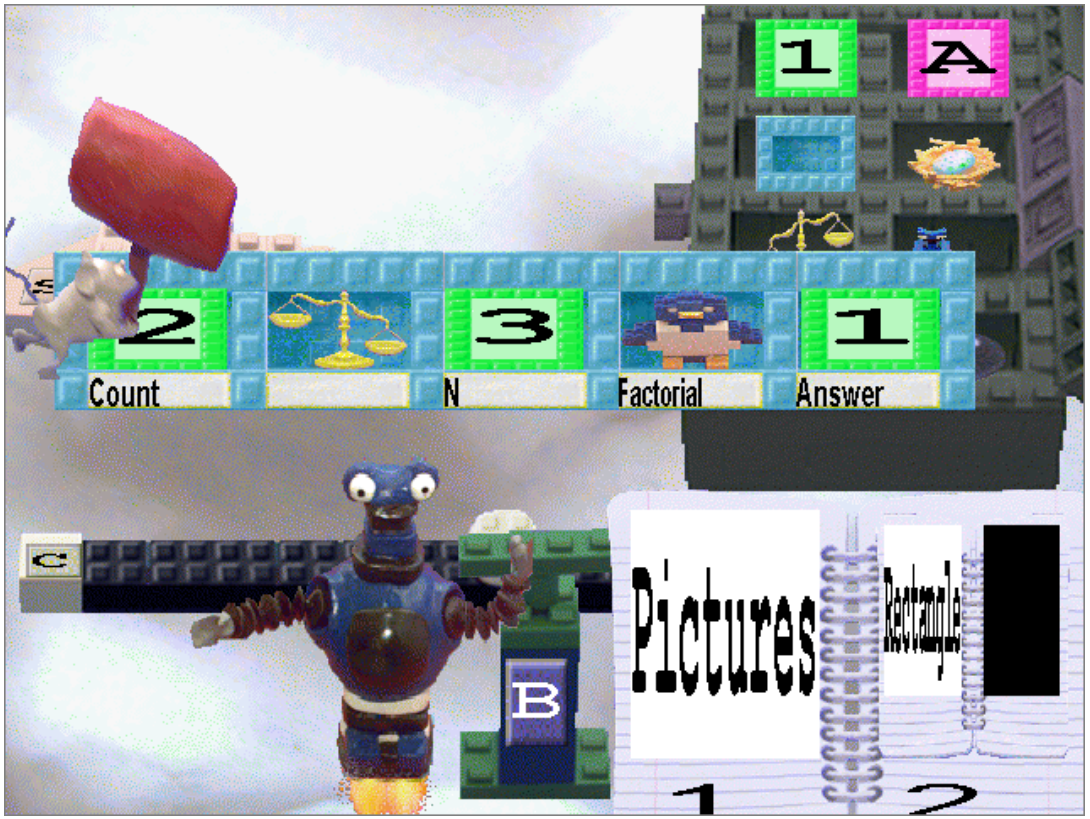
Ken Kahn's ToonTalk – user control of generalisation



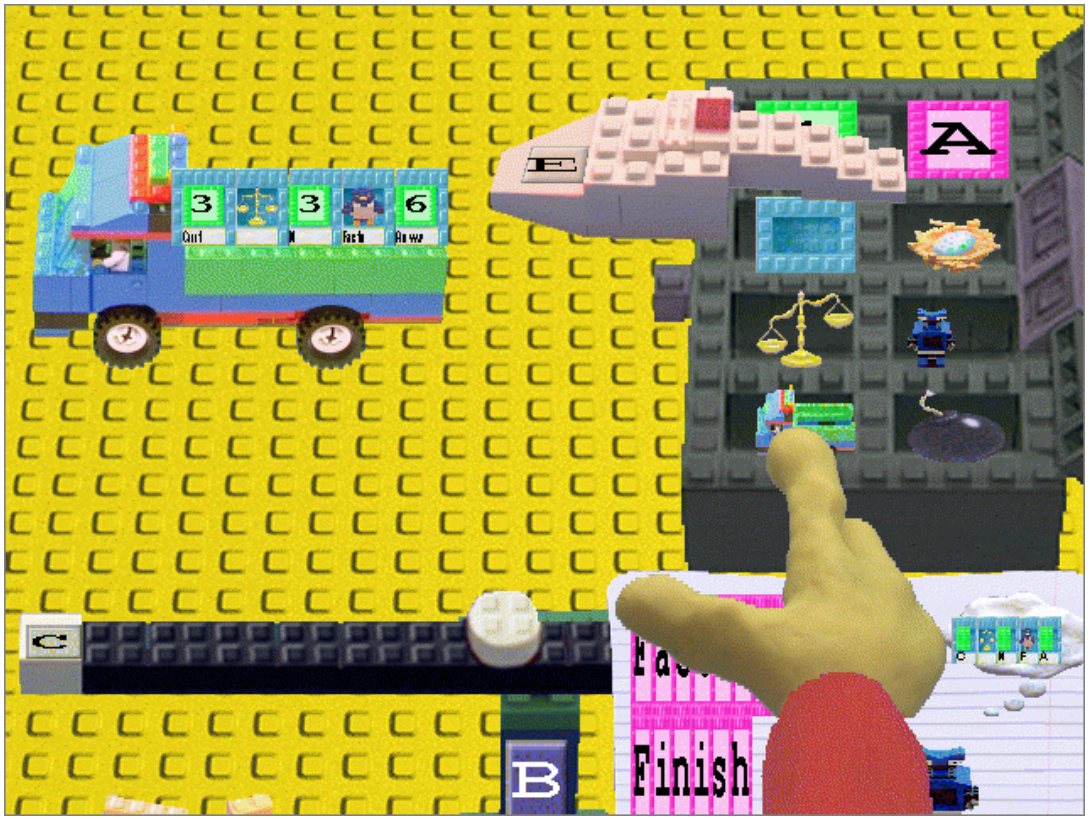
Ken Kahn's ToonTalk – user control of generalisation



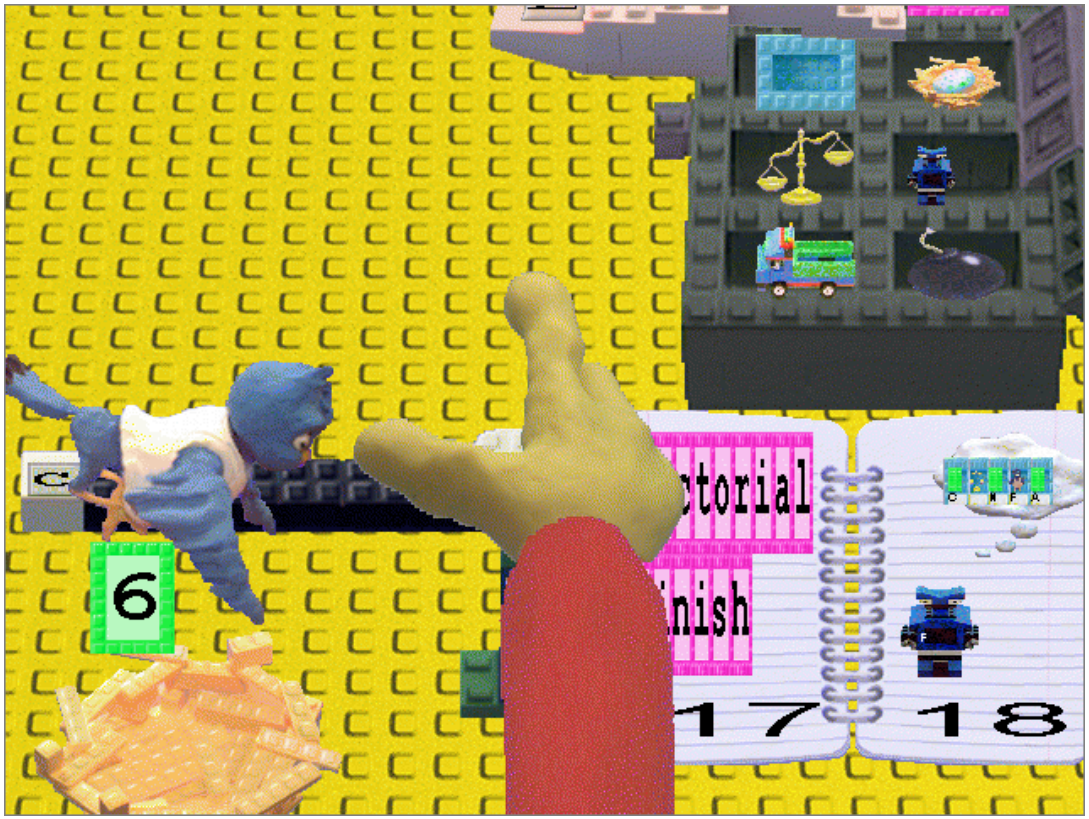
Ken Kahn's ToonTalk – user control of generalisation



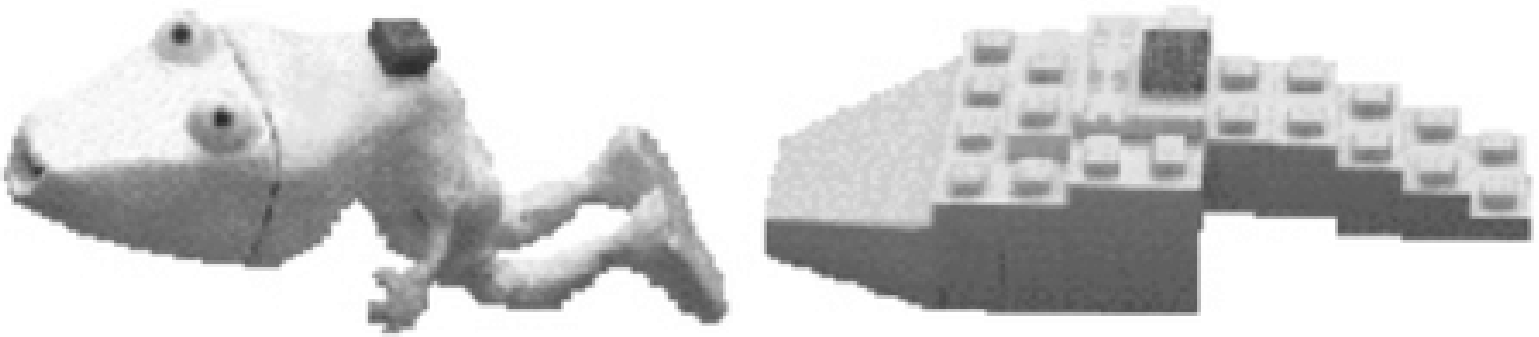
Ken Kahn's ToonTalk – user control of generalisation



Ken Kahn's ToonTalk – user control of generalisation



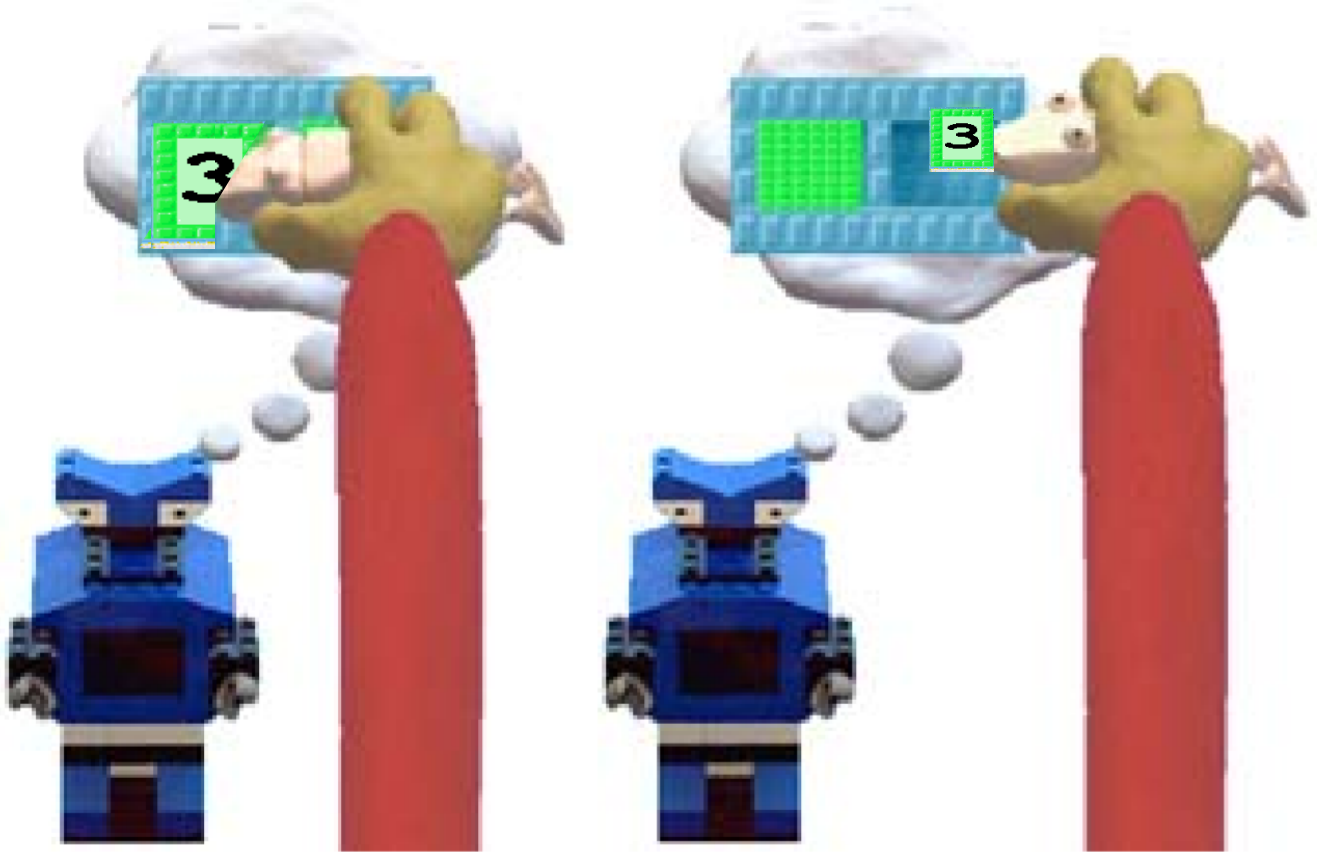
Generalising a constraint with Dusty



Dusty (a)

Dusty (b)

Generalising a constraint with Dusty



Generalisation

Why is the generalisation step so significant?

- Generalisation from examples is fundamental to mental abstraction
  - Repetition of concrete instances (i.e. direct manipulation) does not require abstraction
  - Any automated action (i.e. programming) does require abstraction
- So program synthesis requires the user to conceptualise their problem in an abstract way
  - Programming by example is a strategy for achieving this …
  - … the user can become comfortable with individual cases, while
  - … the system formulates abstractions at the same time the user does.
- Essential that user & system can "discuss" what they are concluding:
  - So is this what you want me to do?
  - No, here is a case where you should do something else.
  - Oh, I see, so like this?

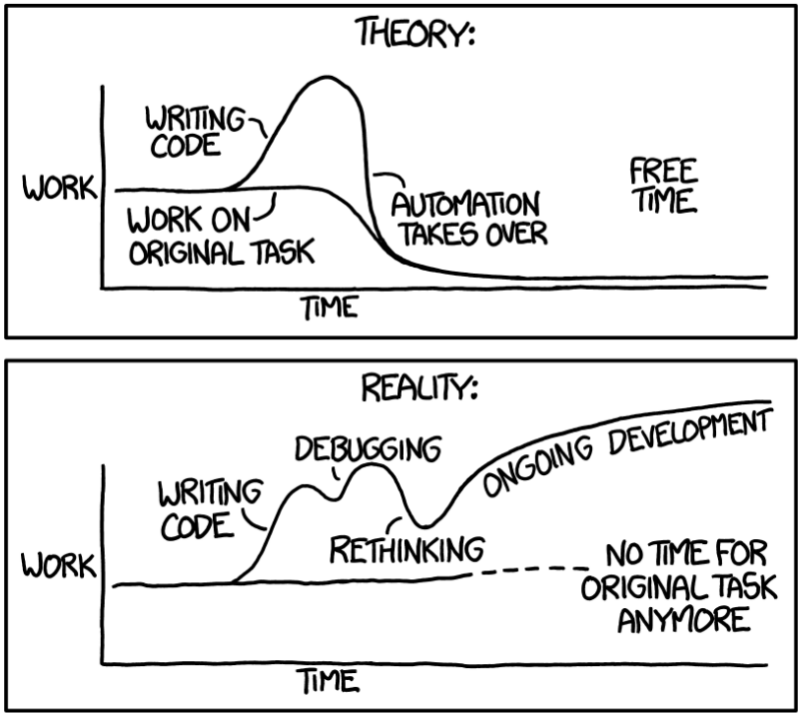
The Attention Investment model of abstraction use

- Programming is not like direct manipulation, so the standard rules of usability (Shneiderman's direct manipulation principles) do not apply:
  - Incremental action
  - Fully visible state
  - Immediate feedback
  - Easily reversible actions
- Making abstractions is cognitively hard, because *actions take place in the future*, and they apply to *multiple potential contexts*.
  - Automating repetitive actions does save time and (mental) effort
  - But formulating and refining abstractions *costs* time and mental effort!
  - What leads a user to approach their tasks in this way?
    - Richard Potter's "Just In Time Programming"
    - Rosson and Carroll's "Paradox of the Active User"
    - Bainbridge's "Ironies of Automation"
    - Burnett's "Surprise, Explain, Reward" (cf mixed-initiative design strategies, including Clippy)

Automation

|< < Prev Random Next > >|

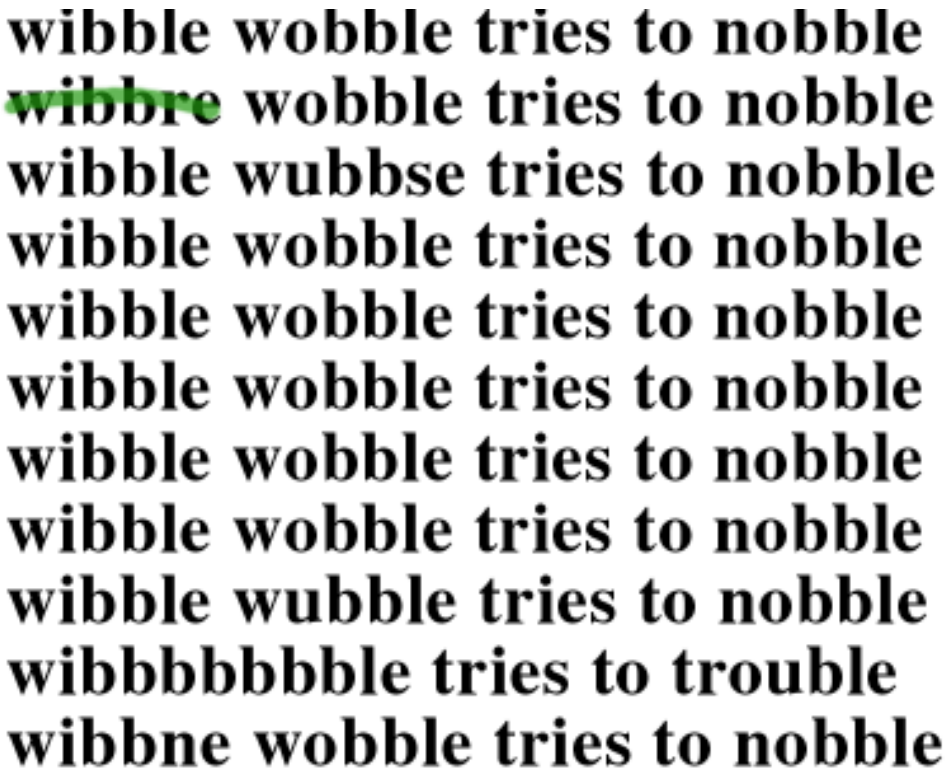
"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"



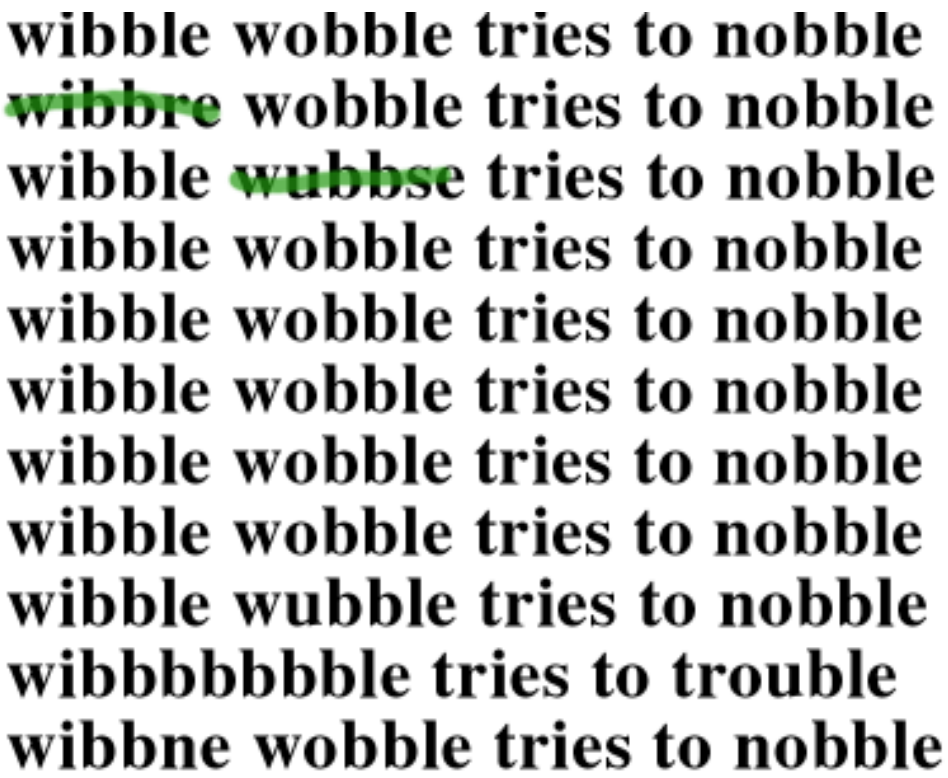
|< < Prev Random Next > >|

SWYN: See What You Need

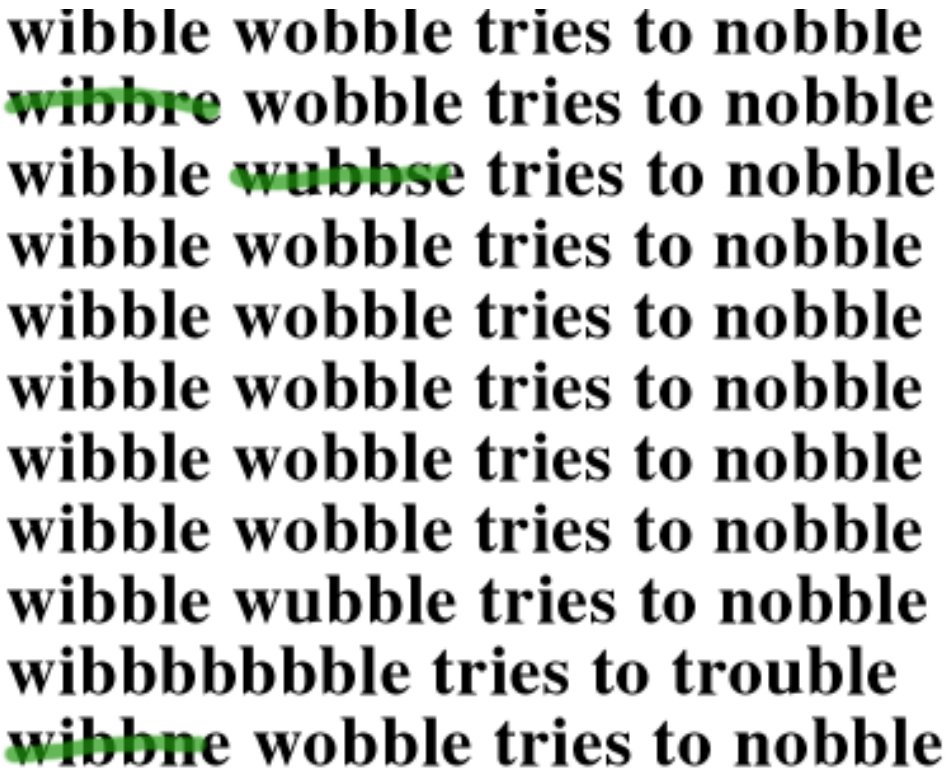
Swyn: inferring regexps to generalise text macros



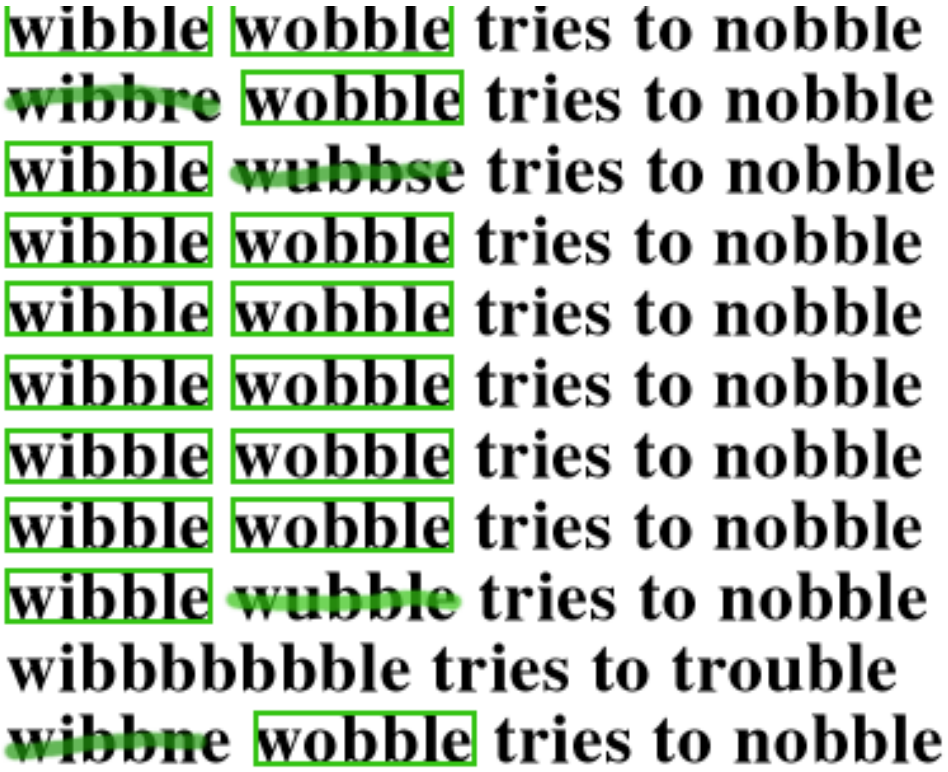
Swyn: inferring regexps to generalise text macros



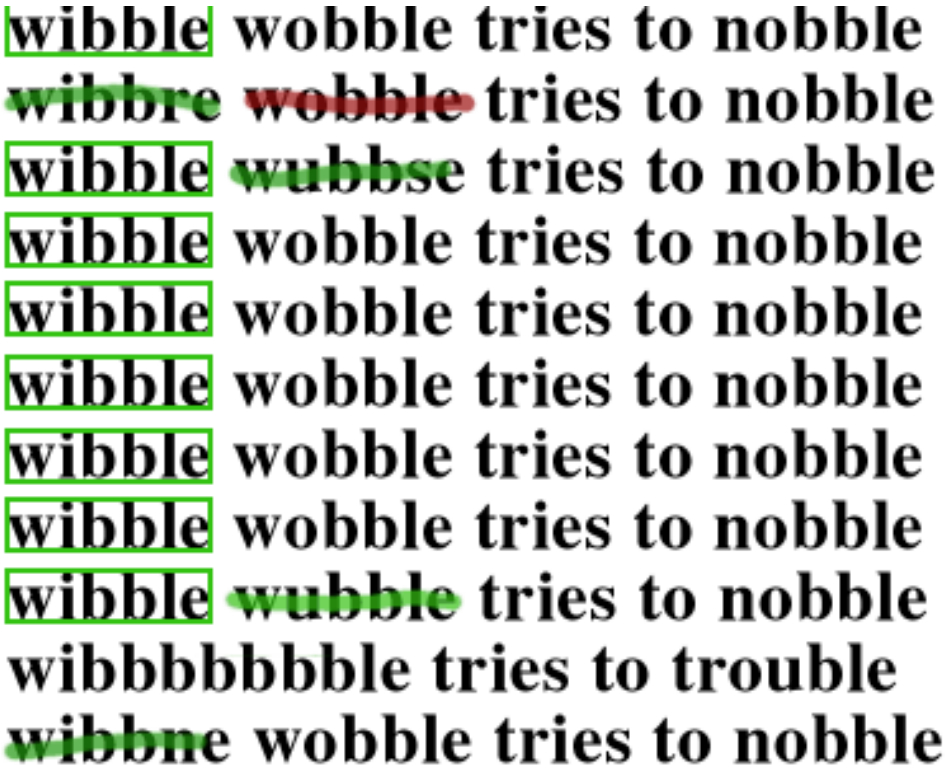
Swyn: inferring regexps to generalise text macros



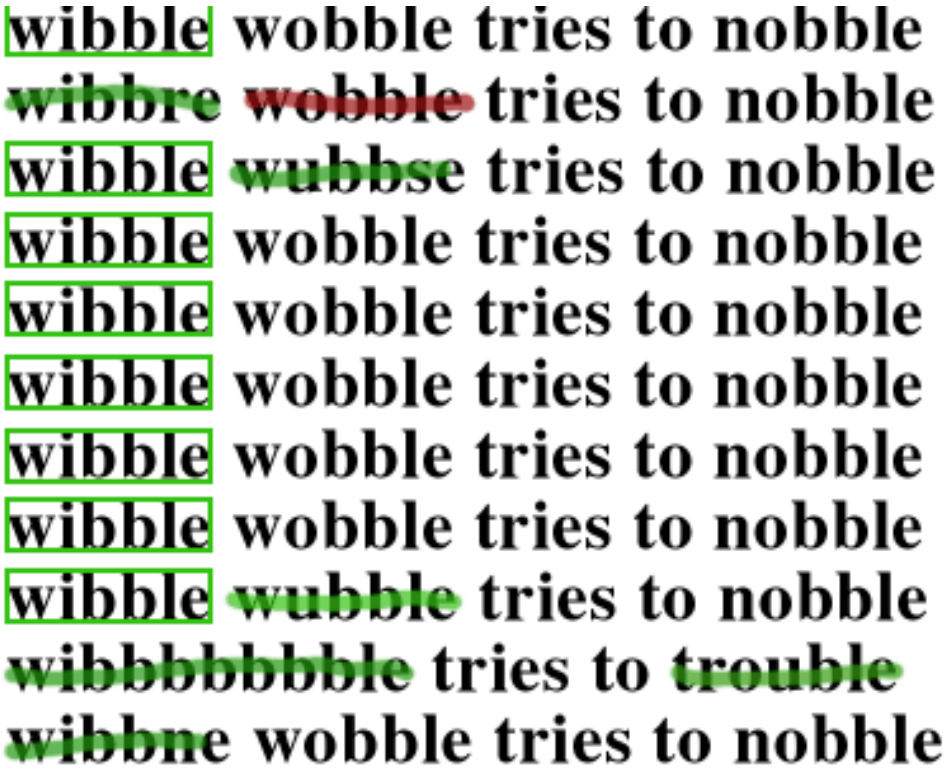
Swyn: inferring regexps to generalise text macros



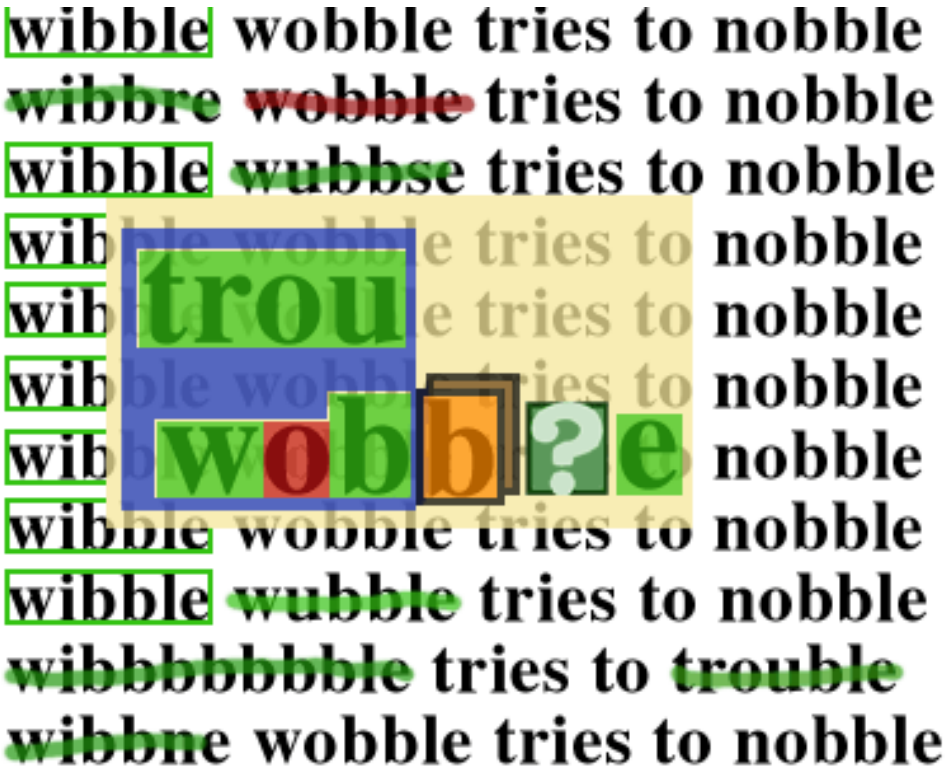
Swyn: inferring regexps to generalise text macros



Swyn: inferring regexps to generalise text macros



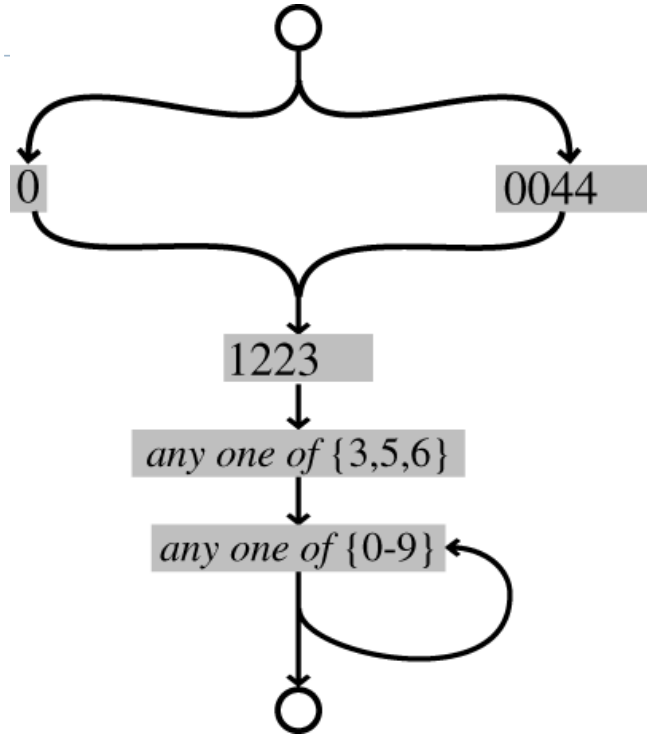
Swyn: inferring regexps to generalise text macros



Communicating inference to the user

- (0|0044)1223[356][0–9]+

- Find one of the following:
  - a) either the sequence "0"
  - b) the sequence "0044"
- followed by the sequence "1223"
- followed by
  - any one of these characters: "3" or "5" or "6"
- followed by at least one, possibly more, of the following:
  - any one of these characters: any one from "0" to "9"



Structured text editing as an ML application

- Aimed at the kind of things people did with sed/awk/perl
  - Many automated text operations involved regexps
  - But users found these the hardest thing to understand …
  - … research agenda for machine learning: sed/awk/perl/swyn
- Similar goals to Witten and Mo's TELS (1989)
  - Learning Text Editing Tasks from Examples
  - See Cypher book chapter 8
- Luke Church demonstrated working solution (2007)
  - Recursive language model "Structured Prediction by Partial Match"
  - Prior expectation based on harvested corpus of regular expressions

Example applications

The *Programmer's Assistant* project from 1978 onwards

- Implemented as Knowledge-Based Emacs (KB-Emacs)
  - PhD project of Charles Rich at MIT
  - Aimed to recognise cognitive plan elements within source code
- In practice, programmer-assist features in modern IDEs are implemented using heuristics rather than AI models
  - Syntax-directed editing
  - Auto-complete of standard constructs
  - Refactoring
  - Inference from identifier names (e.g. follow x=x+1; with y=y+1;)
  - Navigate-by-completion for library APIs
- There is significant research inferring more such patterns from code bases, and a few products coming onstream with more generative predictions
  - https://www.tabnine.com

Working in a data-centric paradigm: FlashFill for Excel

- Building on this paper by Sumit Gulwani (MSR Redmond)
  - *Automating String Processing in Spreadsheets using Input-Output Examples*, Proc. POPL 2011
  - https://www.microsoft.com/en-us/research/publication/automating-string-processing-spreadsheets-using-input-output-examples/
- Live Demo
  - Paste a list of semi-structured text data into the left column
  - Type an example transform result in top cell to the right, then <Enter>
  - Press <Ctrl+E>
- "Synthesises a program from input-output examples"
  - How do you choose the examples?
  - How do you know what will happen?
  - Using this 'program' as a component of a larger system is still a research topic

Visualising abstract structure: Data Noodles

- https://www.youtube.com/watch?v=hyCVBxfx7VE
- Applies a transformation paradigm
  - Directed search for fold/unfold transforms that will achieve the demonstrated result
- Search procedure uses off-the-shelf program synthesis toolkit
  - PROSE SDK from Gulwani team at MSR Redmond
- Custom-built front-end
  - The "spreadsheet" is purely for familiarity of presentation
    - No actual spreadsheet calculation is performed
  - Drag-and-drop target previews allow user to anticipate inference
  - Noodles preserve and visualise the demonstrated actions
    - Allow reasoning about causality from example to synthesised program
    - Potentially support modification/correction of examples