Concrete Abstractions for End-User Computing in Creative Applications

Eli Barzilay  Mira Balaban  Michael Elhadad

eli@cs.cornell.edu  mira@cs.bgu.ac.il  elhadad@cs.bgu.ac.il

November 2, 2001
Motivation — Computer Music

Goal: Provide a better Computer-Music Environment

Capture intentions
Disclaimers

- We did not invent lambda calculus.
- We did not invent editing.
- Being a Programmer is a Good Thing.
- eli@cs.bgu.ac.il
- I just like music.
A fundamental concept of CS:
Express intentions using sharing and abstractions

- Reference objects by identities.
- Generalize common patterns.

Not only in CS…
Examples

- Natural language.
- Learning, remembering.
- Music is a good example.
If programming is so great, just use it to make music:

- **Common Music** — extending Lisp with music.
  
  [http://sourceforge.net/projects/commonmusic](http://sourceforge.net/projects/commonmusic)

- **Haskore** — a younger example in Haskell
  

A major problem with this approach...
Programming is Hard

• Takes years to be a good programmer.
  (still, even expert programmers have problems)

• Much worse for non-programmers.
  (many attempts, no substantial success)

• The challenge: programming power to the people.
  (composing is a good candidate)
Concrete Abstractions

- Suggested solution from Grame.
  
  http://www.grame.fr/Research/

- Basic idea: don’t leave the concrete world.
Problems So Far

- Technicality: names.
- Domain extensions are problematic.
- Still complicated.
- Non-trivial examples are a mess.
A real system that implements these principles.

http://www.grame.fr/Elody/
Same Problems

- Still very complex.
  (requires almost the same effort as programming)

- Domain extension: can’t unify the treatment of abstraction.
  (what is the meaning of “λC.CDC”?)
  (similar problems in GCalc)
The basic idea is still attractive:

- It *does* allow people to create functions without programming.
- All you need to know is music.
  (well, almost...)
Generalized Abstractions

What all this is trying to get at is a general way of discovering a function, given a concrete input/output pair.

- Works for what they have.
- Shows how it would behave in an ideal world.
  (applying “λ123 . 123213” on “5678” yields “5678765678”)
- Impossible to get:
  ambiguous because there is no way to know the intention.
  (“(λ2 . 4)3 =?”)
Ideal World

Assume that there was some black magic that made it possible.

- Still a problem: when we have some transformation in mind, we have to specify it indirectly through two concrete examples.
- Useful only in cases you *already have* two values.
- We use the computer to transform an object, then expect it to guess what we just did...

But this is a good hint for what we *can* do!
Creative activity is usually spontaneous.
Still, the result is structured.
We want both.
Editor Classification

- Roughly place editors on an extension—intension range.
  (WYSIWYG—hierarchy)

- WYSIWYG is usually preferred,
  (less “unrelated junk”)

- but the general direction is to add information on top.
Editor Classification

- Structure is unrelated to abstractions.
- Example: Elody vs. XFig.
Editor Classification

- Further distinguish between abstractions with and without structure.
- Value abstractions (done on a simple flat object).
- Structure abstractions (generated by references).
Question — can we have both.

- BOOMS was started as an attempt to have both.
- And other goodies to.
- Side goal: clean system.
Identities: an important feature.

- Being the same vs. observably equal.
- Trivial with a structure editor.
- Impossible with extensional editors.

Abstractions can play an additional role to achieve this.
Demo.
Identity with Structures

Eli Barzilay  Concrete Abstractions for Creative Applications  21/33
Structure Abstractions

Structure abstraction
Abstraction body
Abstraction node
Bound nodes
Structure Abstractions

\[ \text{ap} \]

\[ \lambda \]

\[ x \]

\[ y \]

\[ z \]

\[ x \]

\[ z \]
Restrictions on Structure Abstractions
Double View

- Obviously, structure editing is more expressive.
- But there is still an important advantage for extensional editing.
- Structure is on still distracting from the real object.
- Lose the immediate connection with the object you manage.
- Elody stuck to real world object and nothing more.
- So, we want to get both options.
BOOMS and Elody

- Constructor nodes of BOOMS correspond to extensional editing operations.
- So why not let users do exactly that to create the hierarchy?
- While they work in an WYSIWYG environment, keep the generated editing structure.
- Example: copy-paste create a link to the same object.
Double View
Double View
Double View

- Claim: useful structures will be created.
- The hierarchy can be used to express sharing as well as abstractions.
Getting Value Abstractions Back

- There is no problems in having this together with Elody’s value abstractions.
- BOOMS has ‘normal’ abstraction and application nodes, corresponding to these Elody editing operations.
- It is also useful to solve the problem of making music from single notes in BOOMS.
Interaction Modes

Users will move:

- from a WYSIWYG-oriented mode,
- to understanding and using the hierarchy on occasions,
- to finally using it in its full power.
  (still using the WYSIWYG view)
Such a hybrid system has all the benefits of both editing modes, while keeping away from irrelevant language formalism.