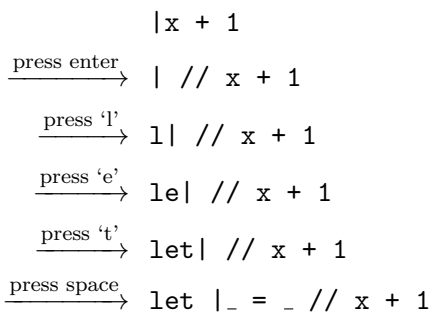
David Moon
CSCI 5849
02/27/2019

For Project 1, I improved the usability of Hazel, a structure editor in which every edit state has both a well-defined (possibly incomplete) type and a well-defined (possibly incomplete) result. Naively, a structure editor that maintains a well-defined type would require constructing expressions in a rigid "outside-in" manner (e.g., I could not first enter `incr` then `1` in a position that expects a number type, since `incr` is a function, so I'd have to construct the function application form first). Hazel circumvents this issue by placing inconsistently typed expression inside holes, which defers the type consistency check until the hole is finished. In addition to maintaining a well-defined (possibly incomplete) type, Hazel is a *live* programming environment, where a result can be computed from any edit state. This means Hazel can provide always-on dynamic feedback as you program. The well-typed invariance and liveness of Hazel add significant additional motivation to solve the structure editing problem, which has been noted in the literature for being hard to use.[1]

The two features I implemented for this project are steps toward that vision. Specifically, I implemented line items and keywords. In the previous iteration of Hazel, let expressions (e.g., `let x = 1 in x + 1`) had to be constructed in an outside manner. You would enter `=` to get `let _ = _ in _`, then you would fill in the holes appropriately. Line items give the programmer flexibility in adding let lines prior to their current expressions, thereby enabling a test-driven development workflow. With line items, you can type `x + 1`, then go to the beginning of the line and hit enter, then construct `let x = 1` in the previous line. You could imagine being to make other declarations, e.g., define a new type. Meanwhile, keywords ease the burden of having to know keyboard shortcuts for every expression construct—to construct a let line, I can simply type out `let`, hit space, and the let construct is generated.

A representative workflow is sketched below. The vertical bar `|` denotes the cursor, underscores `_` denote holes, double-slashes `//` denote newlines.



Links:

- video demo

[1] A. Ko, H. Aung, B. Myers. Design Requirements for More Flexible Structured Editors from a Study of Programmers Text Editing. CHI 2005, Late Breaking Results: Posters. link

- source code
- working demo (without new features)
- working demo (with new features)