

Research Statement

Khoo Yit Phang
<khooy@cs.umd.edu>

Programming is becoming a task that requires significant expertise from programmers as software complexity has grown with users' demands and expectations. Recent advances in programming language and software engineering research can potentially reduce the burden on programmers to manage this complexity. For example, static analysis-based software defect detection tools can be used to quickly identify potential errors in source code. However, many of these techniques remain inaccessible to programmers because they are difficult to use and understand.

In my research, I seek to apply a *user-centered* approach to the design of programming tools. I believe that a good tool should consider the user interface to be just as important as the underlying analysis algorithm, because ultimately, a tool is only successful if a programmer can use it and understand its results. Therefore, I am interested in applying techniques from human-computer interaction and information visualization to research in programming languages and software engineering, to design tools that are easy to use and understand, for expert or novice programmers.

Path Projection

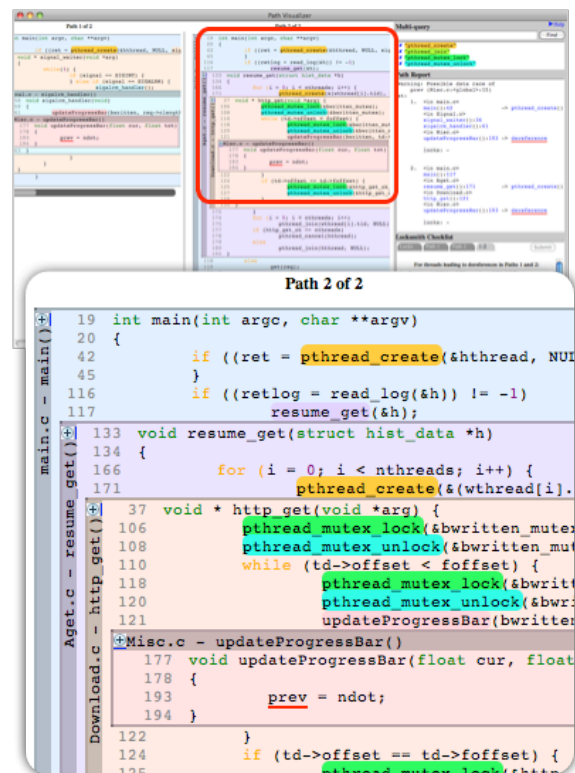
My main research work is *Path Projection*, a new user interface that applies techniques from information visualization to the output of static analysis tools [1]. Path Projection is designed to help users visualize, navigate, and understand program paths such as call stacks or execution traces, which are common in the output of static analysis tools.

Manually inspecting the code in a path is often difficult and time-consuming, since paths may be long and the code spread across many files and functions. Path Projection combines two features—*function call inlining* and *path-derived code folding*—to “project” the source code onto the error path, as shown on the right. This ensures that lines of code relevant to a path are placed close together and in control-flow order, making it possible for programmers to follow a path by reading from top-to-bottom. Programmers may also use a multi-query feature to unfold and highlight code of interest.

In an initial experiment, we asked users to triage errors reported by Locksmith, a data race detection tool for C, using either Path Projection or a conventional source code viewer as the user interface. This experiment showed that users completed the task 18% faster when using Path Projection, and users preferred it to the conventional viewer.

Arrowlets

My interest in usability extends beyond programming tools and includes programming library design. Another current work of mine is a JavaScript library called *Arrowlets* [2], based on Haskell's arrows [3], that is designed to simplify the task of writing asynchronous event-driven programs such as Path Projection. Typically, event-driven programs in JavaScript (as well as GUI libraries in other languages



such as Java/Swing) are written using callbacks; however, using callbacks to chain sequences of events is tedious and makes the control-flow difficult to understand. Worse, this style tends to result in non-modular code because callbacks are usually hard-coded into event handlers.

Arrowlets unifies synchronous function composition and asynchronous callback composition under the arrow interface, eliminating the messiness of manual callback plumbing. Event listeners and event handlers can then be written as modular units of code and flexibly composed using arrow combinators. For example, the snippet below expresses the complete control-flow of drag-and-drop in three parts, and demonstrates how Arrowlets makes the control-flow of event-driven programs clear and concise:

```
var dragOrDropA = /* keep dragging until button is released */
  (
    (EventA("mousemove").next(dragA).next(Repeat))
    .or(EventA("mouseup").next(dropA).next(Done))
  ).repeat();

var dragDropOrCancelA = /* cancel if button is released before dragging */
  (EventA("mousemove").next(dragA).next(dragOrDropA))
  .or(EventA("mouseup").next(cancelA));

var dragAndDropA = /* press to begin drag-and-drop */
  EventA("mousedown").next(setupA).next(dragDropOrCancelA);

ElementA("dragtarget").next(dragAndDropA).run(); /* use drag-and-drop */
```

Future Work

I believe Path Projection has great potential as a user interface for many programming tasks. In particular, Path Projection would be an excellent interface for interactive debugging. My idea is to allow programmers to incrementally “build” a path in Path Projection while stepping through a program, and automatically inline functions that are stepped into. The resulting visualization becomes in effect a record of the debugging session, potentially reducing the mental burden on programmers to keep track of the program execution. This visual record could also be used as a bug reporting format, as a concrete way of explaining bugs to other programmers.

Additionally, I plan to study how program analysis tools can be tuned for use with Path Projection or other user interfaces. During our experiment, we discovered that, commonly, errors can be easily triaged if the appropriate information can be quickly found in the code. I am currently investigating two approaches: first, a *triaging checklist* designed with knowledge of the underlying algorithm can be used to systematically guide programmers towards potential sources of imprecision in the error path. Second, the underlying analysis can be augmented with a suite of lightweight *analysis assistants*, such as the multi-query feature in Path Projection, to enable programmers to perform supplementary queries on the result.

As for Arrowlets, I am quite interested in exploring other possible applications of this library, for example, in AJAX client-server applications. I also believe that library-level techniques from functional programming, such as monads and applicative functors, can make JavaScript, the *lingua franca* of the Internet, a more pleasant language to program in.

References

1. [Khoo](#) Yit Phang, Jeffrey S. Foster, Michael Hicks, and Vibha Sazawal. Path Projection for User-Centered Static Analysis Tools. In *Proceedings of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE 2008)*. November 2008.
2. [Khoo](#) Yit Phang, Michael Hicks, Jeffrey S. Foster, and Vibha Sazawal. Directing Javascript with Arrows (Poster Summary). In *Poster Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming (ICFP 2008)*. September 2008.
3. John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37:67–111, May 2000.