Andrew Begel

CS294-1

Chapter 11: Intentional Programming Review

March 11, 2002

Whomever wrote this article is in the pocket of Charles Simonyi. The entire chapter reads like an Intentional Programming (IP) lovefest. There's very little hard science in here, just a lot of hand-waving and vague references to how IP will save the world (pg 508: "One way of thinking about IP is to view it as the perfect environment for metaprogramming").

Let's discuss a few of the bad points.

1. A point was made that domain-specific library vendors have to keep rewriting their libraries as new languages come out. a. Has the author heard of foreign function interfaces? They exist in almost all languages to enable cross-library compatibility. b. .NET? This is technology from Microsoft that allows one to write a library in one language and have it accessible/callable from any other.
2. Is R-code a universal intermediate language? The article realizes that its impossible to reduce any R-code to all target languages, but is R-code semantics universal across source languages or are they just talking about R-code syntax? If semantics, that'd be impressive, since the programming language community hasn't figured out how to do this yet.
3. IP's editor is a syntax-directed editor. No matter how many times they try to deny it, it's a tree-based editor. Jeez.
4. Mike Toomim's idea for correspondences between chunks of code is here. I guess they've already decided that the way to represent this is structural with parameters. Funny, Mike hasn't settled on that yet. Perhaps he thinks he should do a user study first. What a novel idea.
5. How do interactions between intentions work during code reductions. It's fine that I can recurse on my node's children, but what about reductions or optimizations that require full tree knowledge? Global common subexpression elimination? Receiver-class analysis? How do they fit in here?
6. It's not acceptable to the programming community to store source code in a database. No matter how good idea it seems, people just will not accept it.
7. If you can select structural items, what do you do about selecting non-terminals in the tree that are part of chain reductions? The visible selection won't change as you go up and down in the tree in the chain. Harmonia restricts the selection to particular language-module designer chosen non-terminals for this reason (and because many nonterminals aren't relevant to the end-user programmer).
8. If you have so many different ways to enter code, programmers will have to master many different ways to get the code in there. The "show me your type-in" methods on intentions can easily lead to a proliferation of non-consistent code entry mechanisms which could confuse the programmer, and make the system difficult to learn.
9. That picture of a C function rendered in English? Um, no. Removing the punctuation from a C program does not make it English.
10. The requirement that semantic information only pass to a direct ancestor or child of a node causes the far-off information flow problem common in attribute grammars. You have to insert many many extraneous copy rules into the grammar to pass information from one location to another, when an external data structure could have optimized this.
11. Marat, please don't get your ideas about multi-file dependencies in Harmonia from the IP chapter (ref: pg 548 2nd paragraph).
12. Reductions of intentions are not guaranteed to be confluent. That's why optimizations have to be repeated over and over again. Order almost always matters in compilation and semantics. Conflicting intentions written by many non-communicating vendors becomes quite likely.
13. While revolutionary ideas are good to hear about every so often, it's the evolutionary ones that make it into practice. Tease apart the fifty or so interesting ideas in IP, and try to put them into practice individually, rather than as a gigantic monolithic system.
14. The claim that programming with containers in Java is tedious due to the lack of genericity is made without support. In fact, subtype polymorphism suffices quite well in the absence of parametric polymorphism.
15. Programmer extensible languages are icky. Ever hear of the John Doe abstraction barrier? It's when one programmer uses too many macros in their code, and makes it impossible for anyone else to understand it without their help.
16. I want to see examples of how IP can apply domain-specific transformations to eliminate the overhead of using encapsulated language extensions.

Good things in the work:

1. It's good to recognize that an editor that supports embedded languages (SQL in C) can help identify errors in the sublanguage before runtime.
2. IP makes pretty editors.