From Interpreters and Abstract Machines to the λ Insight

1 The Mechanical Evaluation of Expressions

```
@article{ pl:secd,
 author={Landin, P. J.},
 title={The Mechanical Evaluation of Expressions},
 journal={Computer Journal},
 volume=6,
 issue=4,
 pages={308--320},
 year=1964
}
```

Summary: Landin introduces the language of applicative structures as a small but abstract language that can naturally express various forms of calculations such as calculations with numbers, booleans and lists. Landin gives meaning to programs in the language through their evaluation by an abstract and formally defined machine.

Evaluation: This paper introduces numerous novel and important ideas: 1) it makes an explicit syntactic connection between the λ-calculus and programming languages; 2) it splits the essence of the notation of a language a set of features that address the core computational requirements of any language and a set of features that are specific to the domain of the problem that the language is intended to help with; 3) it suggests the use of abstract applicative structures as a uniform syntactic description of the core computational requirements of different languages; 4) it coins the term “syntactic sugar” to explain how we can grow a core language with more programmer-friendly forms that are in reality compositions of applicative structures; 5) introduces (and coins the name for) closures as we know them today; 6) it pioneers the formal definition of programming languages with the innovation of abstract machines that manipulate the abstract syntax of programs. Overall, this paper has shaped research on formal models and design of programming languages.

2 The Next 700 Programming Languages

```
@article{ pl:iswim,
 author={Landin, P. J.}
 title={The Next 700 Programming Languages},
 journal={Communications of the ACM},
 volume=9,
 issue=3,
 pages={157--166},
 year=1966
}
```

Summary: Landin builds on his previous work on the language of applicative structures to suggest a systematic way to design new languages. In particular, the paper proposes ISWIM, a framework that splits the design of a programming language into syntactic and semantic concerns. In particular, it suggests that the syntax of seemingly different languages can map, in a structure preserving way, to an abstract syntax that consists of an applicative core, mechanisms that manage the scope of user-defined names (names) and domain-specific non-algorithmically specified operations. The abstract syntax can then be translated to applicative structures that can be evaluated on an abstract machine. Furthermore, the abstract syntax allows the comparison between different languages whose syntax seems different via the application simple (local and semantic preserving) transformations. One of those is described with the β axiom of the lambda calculus adapted to the non-imperative subset of the syntax of abstract ISWIM.

Evaluation: This paper together with "The Mechanical Evaluation of Expressions" has had significant impact on the way programming languages researchers analyze and design languages. First off, it reinforces the message of "The Mechanical Evaluation of Expressions" and makes it clear that from a semantics perspective many syntactic details do not matter. As a result it frees the technical aspects of the design of programming languages (in contrast to the human ones) from syntactic details that prevented the comparison of different languages. Moreover, the axiomatization of the syntactic transformations of abstract ISWIM (and especially the β axiom) foreshadows Plotkin's work. Overall, the analysis and design of programming languages using a formally specified core model whose "meaning" is defined via syntactic rewritings or abstract machines is nowadays the norm.