Abstraction

1 Protection in Programming Languages

```
@article{m:protection
  author = {Morris, J. H.},
  title = {Protection in Programming Languages},
  journal = {Communications of the ACM},
  volume = {16},
  number = {1},
  pages = {15--21},
  year = {1973}
}
```

Summary: Morris examines how lignuistic support for user-defined abstractions enables modular programming. Lexical scoping and first-class functions help programmers create private "channels" of communication between different uses of the same or different parts of their code. Beyond that, in a language with a static type system, the ability to define new types allows programmers to hide the implementation of data types and enforce interraction with their instances only through a restricted abstract interface phrased in terms of the new types. Morris shows that in languages without static type systems, the same is possible if the language provides operations for creating new tags at run time, using the tags to mark values and inspecting the tags of a value. Finally Morris discusses how such tag-related features are sufficient to construct programmatically authorization and authentication mechanisms.

Evaluation: This is a very important paper. It introduces linguistic ideas about data abstraction, information hiding and encapsulation that have influenced language design ever since. For example dynamic sealing has been the cornerstone for enforcing parametric polymorphism in languages without static type systems.

2 Programming with Abstract Data Types

```
@inproceedings{lz:adts
  author = {Liskov, B. and Zilles, S.},
  title = {Programming with Abstract Data Types},
  booktitle = {Symposium on Very High Level Languages},
  pages = {50--59},
```

```
  year = {1974}
}
```

Summary: This paper introduces CLU, one of the first languages where features for defining abstract data types are the centerpiece of the language design. In CLU a program consists of clusters, i.e.. data type definitions, and functions that operate on instances of clusters. Clusters declare a new abstract type, specify its translation, dubbed representation, to other (primitive) data types, and the implementation and interface of the operations that can inspect and construct instances of the new data type. All other code should be oblivious to the implementation details of the data type. The paper envisions that a type system can enforce this invariant and thus bring to CLU programmers the benefits of encapsulation, information hiding and modular program reasoning.

Evaluation: Traces of the design of CLU can be found in the design of pretty much every modern typed language.

3 Towards a Theory of Type Structure

```
@inproceedings{t:type-struct
  author = {Reynolds, J. C.},
  title = {Towards a Theory of Type Structure},
  booktitle = {Programming Symposium, Proceedings Colloque sur la Pro-
grammation},
  pages = {408--423},
  year = {1974}
}
```

Summary: Reynolds sets to develop a formal framework for investigating abstract data types and parametric polymorphism, ideas were at the center of a fierce debate between language designers at the time. To do so, he invents an extension of the λ-calculus with universal quantification over types. He states and proves a first version of the representation independence property as a key semantic property of terms in the polymorphic λ-calculus. Representation independence captures that the meaning of a term does not depend on the implementation details of any instances of abstract data types it uses, i.e., it is independent of the representation of such data types.

Evaluation: Today, Reynold's calculus, also known as System F, is the default model basis for typed languages with polymorphic features (System F was discovered independently in the field of mathematical logic by Girard a couple of years earlier). Besides that, the paper introduces the main property, representation independence, and proof technique, logical relations, for studying polymorphic types and their semantics and initiated a whole new line of research for getting all their technical details right.